BOXER - A Design-to-Build System

June 12th, 1987

Timothy Patrick Daly

IBM Thomas Watson Research Center
Yorktown Heights, New York

June 12th, 1987 3:54 p.m.
# CONTENTS

Abstract ................................................................................................................. 1

Background ............................................................................................................. 2
Introduction .............................................................................................................. 2
Prior Work ................................................................................................................ 2

Toplevel - Design-to-Build ..................................................................................... 4
Overview .................................................................................................................. 4
Architecture ............................................................................................................. 4
  Design ..................................................................................................................... 4
  Planning ................................................................................................................. 4
  Build ....................................................................................................................... 4
  Learning ............................................................................................................... 5
Issues ......................................................................................................................... 5
  Objects ............................................................................................................... 5
  Rules .................................................................................................................... 6
  Error Model ........................................................................................................ 6
    Errors and Planning ........................................................................................... 6
    Error classes .................................................................................................... 6

The Design System .................................................................................................. 9
Overview ................................................................................................................ 9
Architecture ........................................................................................................... 9
  Design system interfaces ..................................................................................... 9
  Definitions .......................................................................................................... 9
  Constructing a design ......................................................................................... 9
  Editing a design ................................................................................................. 10
Issues ....................................................................................................................... 10
  User interface .................................................................................................. 10
  Analogy ............................................................................................................ 10
  Error handler specification .............................................................................. 10
  Knowledge base transparency and superobjects ............................................ 10

The Planner ............................................................................................................ 13
Overview .............................................................................................................. 13
Architecture ......................................................................................................... 13
  Definitions ....................................................................................................... 13
  Expanding a design into a plan ....................................................................... 13
    The ideal robot world .................................................................................. 13
    Macro expansion ....................................................................................... 13
    Pose expansion ......................................................................................... 13
    Pose precomputation ............................................................................. 13
    Technology clauses ................................................................................. 14
    User clauses ............................................................................................. 14
  Verification of the plan ................................................................................... 14
    Affixment .................................................................................................. 14
    Stability ..................................................................................................... 14
    Path Planning ............................................................................................. 14
    Collision .................................................................................................... 14
    State Transition .......................................................................................... 14
    Technology Rules ...................................................................................... 16
Generating the task-specific program ........................................ 16
  Hand assignment .................................................................. 16
  Robot motion calls ............................................................ 16
  Error Objects, Sensors, and Transducers .............................. 16
  The virtual to real robot correspondence .............................. 16

Issues ................................................................................. 16

Expand .................................................................................. 16
  Macro expansion .................................................................. 16
  Non-picture object representation ........................................ 17

Verify ................................................................................... 17
  Non-Commutative operations .............................................. 17
  Concurrent Events .............................................................. 17

Generate ................................................................................ 17
  Ambiguity .......................................................................... 17

The Build System .................................................................... 19

Overview .............................................................................. 19

Architecture ......................................................................... 19
  Definitions ......................................................................... 19
  Starting a task specific program .......................................... 19
  Executing a task specific program ....................................... 19
  Error handling in a task specific program ........................... 19

Issues .................................................................................... 19
  State maintenance .............................................................. 19
  Tolerance (how do you know that you got there?) ............... 20

Knowledge base ..................................................................... 22

Overview .............................................................................. 22

Architecture ......................................................................... 22
  Definitions ......................................................................... 22
  Knowledge Base Structure .................................................. 22
  Knowledge Base Access ...................................................... 22

Issues .................................................................................... 22
  The object library and the knowledge base ......................... 22

Learning component ............................................................. 24

Overview .............................................................................. 24

Architecture ......................................................................... 24
  Definitions ......................................................................... 24
  Learning Component Structure .......................................... 24
  Knowledge Base Access ...................................................... 24

Issues .................................................................................... 24
  The object library and the knowledge base ......................... 24

Conclusions .......................................................................... 25

Conclusions .......................................................................... 25

Acknowledgements ............................................................... 25

References ........................................................................... 26
Abstract

The BOXER system will allow a designer using a solid modelling system to define an assembly from predefined parts and have the assembly built automatically. One key concept is to capture sequence information at design time and use this sequence as a basis for building the object. A second key concept is learning constraints from failed plans and feeding these constraints back to the designer. This document is a discussion of the BOXER system.
Background

Introduction
Since the early days of robotics there has been the dream of creating a system that will allow a user to specify what he wants a robot to accomplish but not how to accomplish it. Specifying only WHAT to accomplish is called task-level programming. The problem of building such a system is very complex and provides a fertile ground for research.

There are several approaches to task-level programming. Some systems approach the problem from a textual view. The task is specified in some new language and converted into commands for the robot system. This approach suffers from the problem of inadequate descriptions of the objects to be manipulated.

Another approach is from a solid modelling view. The task is specified as a set of static relations between parts and the system plans how to achieve this static relation. While the objects to be manipulated are described well, the system must figure out sequence information.

In this paper a different approach is proposed. The idea is to combine solid modelling with textual programming. In particular BOXER will capture sequence information during the design session in textual form. This sequence information is then used as a skeleton for the planner.

Section 1 discusses an overview of the BOXER system architecture. Section 2 discusses the design architecture. Section 3 discusses the planning architecture. Section 4 discusses the build architecture. Section 5 discusses the knowledge base. Section 6 discusses the learning component. Section 7 is the conclusion.

Each of the sections is broken into two parts. The first part is an architecture and the second is a discussion of some relevant issues.

Prior Work
In 1976, in work done at the MIT AI labs, Tomas Lozano-Perez’s thesis describes "a mechanical assembly system called LAMA (Language for Automatic Mechanical Assembly). The goal of the work was to create a mechanical assembly system that transforms a high-level description of an automatic assembly operation into a program for execution by a computer controlled manipulator. This system allows the initial description of the assembly to be in terms of the desired effects on the parts being assembled."

"This research concentrates on the spatial complexity of mechanical assembly operations. The assembly problem is seen as the problem of achieving a certain set of geometrical constraints between basic objects while avoiding unwanted collisions. The thesis explores how these two facets, desired constraints and unwanted collisions, affect the primitive operations of the domain."

In 1978 Lieberman and Wesley wrote a paper which defines the AUTOPASS language. While AUTOPASS has been widely quoted as the highest level programming language for robots, there are no implementations. The paper is an excellent introduction to this area and is strongly recommended as background reading.

AUTOPASS was designed to "demonstrate that a language close to the level of the assembly language sheet ... provides a natural interface for assembly or design engineers and that this language can be implemented."

In 1978 Grossman and Taylor, working at the Stanford AI labs, built POINTY. This work dealt with interactive generation of object models with a manipulator. Here are a few relevant quotes from this paper.

"The ultimate goal of research on high-level manipulator languages is a language in which very few statements are needed to describe a highly complex assembly. For an object with N parts, perhaps a realistic goal would be to have a language in which the assembly can be described in about N statements."

"In the" ultimate language, a program for assembling 1000 water pumps might have this form:
DECLARE WATER_PUMP etc;
MAKE PLAN
  (WATER_PUMP, ASSEMBLY_PLAN);
EXECUTE_PLAN(ASSEMBLY_PLAN, 1000);

"... all the effort of writing the program would be in expanding the "etc." in the first line.

The expansion of the "etc" constitutes the world model used by MAKE_PLAN. This world model is a complex data base, including such information as the specification of mechanical parts, component hierarchies, affixment relationships, geometric shapes, Cartesian transformations between features, material properties, and so forth. It is clear that the declaration of such a detailed world model would be a lengthy process, possibly requiring hundreds of lines of text. The development of high level manipulation languages, therefore, will shift the problem from writing procedures to writing declaration."

In 1983 Tomas Lozano-Perez "filOZ2" proposed a task-level programming system called ATLAS (Automatic Task Level Assembly Synthesizer). The user of the system is required to specify only goals for the physical relationships among objects instead of the robot motions for achieving those goals. This system was never implemented.

That same year, at Cambridge University, Alan Weatherall "filWEAl" connected the technology of CADAM and robots. It took the output from a CADAM system and created input to a Series/1 running AML. The robot is modelled in CADAM as a machine tool. It uses CADAM's ability to model the path of a cutting tool end effector to generate path information.

Also in 1983 Anthony Levas "filLEV1" dealt with teaching robots plans by example. The robot is led through a series of motions using a teach pendant. Each significant point, called a "watch point", is recorded. Watch points correspond to terminal symbols in a finite grammar. Sequences of these watch points are recognized by the program and the recognized sequences are replaced by their non-terminal forms. At a later time these non-terminals can be expanded to generate the original sequence.

This work was done at the University of Connecticut. It is of interest as an alternate method of generating task-level assembly sequence information by abstraction from examples.

In 1984 Dan Russell "filRUS1" built a Schema-based planning system. This was an effort to build a system that, given a statement of the goal object, builds that object. The method is to build AND-OR goal trees of processes. Each process consists of a set of subprocesses that solve smaller subproblems necessary to achieve the goal. This work was done on a LOOPS system at Xerox PARC. It is written in SMALLTALK. The SMALLTALK objects are called schemas.

An interesting feature of this work is that it allows multiple plans to exist. Since each schema broadcasts its message to all other schemas, multiple plans can execute in parallel. At any given time only one plan is actually doing the work. All other plans are tracking the progress of the working plan. If the working plan fails a backup plan is selected and the construction continues from the point of failure.

Most of the mechanism for this parallel tracking is inherent in the SMALLTALK system. Messages for executing each step of the plan are broadcast to all objects. Objects not actually performing the operation can "listen" and track the world state.

In 1987 Alberto Segre "filSEG1" built a system (ARMS) to do explanation-based learning of generalized robot assembly plans. This was an effort to build a system that acquires generalized problem-solving knowledge on the basis of a single program-solving example. It is applied to the 'robot retraining problem' which is defined to be the 'difficulties encountered when preparing a robot to perform some novel task'.

The goal of ARMS is to capture the plan taught by an expert and convert the plan to a more efficient and more general plan.

One curious aspect of this system is that the goal state is described by specifying the desired function:eph/. , not the desired description. It is not clear to this author how describing the function of a telephone affects its assembly.
Toplevel - Design-to-Build

Overview
A Design-to-build system is a system that allows a user to define an assembly from a predefined set of parts and automates the steps necessary to have the assembly built. The BOXER system breaks into 5 parts (see figure 1a), the DESIGN :eph1 system which is the primary user interface, the PLANNER :eph1, the BUILD system which is the runtime environment, the KNOWLEDGE BASE :eph1 and the LEARNING COMPONENT :eph1. The technology to build each part exists in some form today. BOXER is a synthesis of these subsystems into a larger task-specific system.

Architecture
The architecture of the system is best characterized by the flow of information through the system. The primary steps are:

Design

Constructing a design: The design consists of using parts that are predefined to construct a goal object. The construction process takes place at a solid modelling system (SMS) terminal. During construction the system monitors sequence information and uses this information to construct a skeleton plan for assembling the goal object.

Editing a design: The user may edit the design information at any time. The sequence information can be edited to re-arrange, add or delete steps that may have been defined graphically. Other information, such as default pallet positions may also be edited.

Once the user is satisfied with his goal object, the design session is ended and the planner is invoked.

Planning

Expanding a design into a plan: Once the goal object has been defined and can be correctly redrawn within the SMS the user has the option of creating a real object. The sequence information, the goal object, the part definitions, etc. are combined to define an internal form of a plan to construct the goal object. The internal form of the plan is a set of states, relationships, and actions, similar to RAPT ff1POP1, POP2". The planner is implemented in a language similar to KROS ff1DALS".

Verifying a plan: Once a plan has been expanded it is "executed" to ensure that the plan does not contain any detectable errors. The verification phase assumes a world model and a virtual robot and steps through the plan checking pre- and post-conditions at each step. Affixment relations are built, stability is checked, collisions are examined, etc.

Error checks and handlers are added to verify that the assumptions made by the planner are valid at runtime.

Generating the task-specific program: When the planner has checked all that it can then the task specific program for a given target system is generated. This program will contain target system dependent calls to library routines. Code that can be directly executed by the target system is generated.

Once the planner completes the task-specific program is passed to the build system for execution.

Build

Starting a task specific program: When the target system starts it must make sure that the real world state satisfies all of the assumptions of the planner. In particular, fiducial points such as the pallet position must be calibrated.

Executing a task specific program: Once the startup assumptions are valid the system executes the task as requested. The program should build the goal object from the available set of parts.

Error Handling in a task specific program: Before, during, and after each motion the pre- and post-conditions and assumptions made by the planner are checked. If it is found that one of these conditions or assumptions is not valid then an error handler is invoked. The handler examines the current
state and attempts to recover the desired world state.

Learning

Failure detection: The error handler for recoverable failures records the failure information and continues with the program. If the failure re-occurs then a more general error handler is invoked. If there are no more general error handlers available then the operator is asked to correct the problem. The failure information is recorded including information about the failure from the operator. This is the source of negative feedback training examples.

Success detection: Plan steps may be marked as 'expected failure' steps if the user has decided to construct an assembly which the system feels it cannot construct. Success of these steps is recorded. This is the source of positive feedback training examples.

Feedback learning: The failure or success information is used by the learning component to modify the knowledge base. The rules used by the planner are specialized or generalized. New rules may be added. New concepts may be added. The failure or success and a record of its effects are added to the knowledge base for future reference since it provided the assumptions on which the learning depends. A plan may provide several successes or failures.

Learning effects: The new knowledge has two primary effects. First, the planner will construct different plans based on the new knowledge. Second, the designer is provided with feedback about assembly constructs that the system decides it cannot manufacture. The user has the ability to override the system's decision. This is the source of positive feedback training examples.

Issues

Three of the key global issues in BOXER are the uniformity of representation, consistency and the modelling of errors.

The diverse representations currently used in existing subsystems to do solid modelling, planning, and building makes using them in an integrated manner difficult. The BOXER system, by creating a uniformity of representation has 2 advantages. First, it limits the complexity by reducing the number of design decisions necessary. Second, it increases the flexibility by increasing the number of possible connections between system components.

Consistency has the overriding advantage of making the system understandable. The complexity of the problem and of an integrated solution demands it.

In order to be successful a design-to-build system must deal with errors. Errors cannot be avoided during the actual assembly process. A model that enables the classification and analysis of errors is sorely needed. A solution to this problem is a key contribution of the BOXER system.

Objects

Object oriented programming combined with procedural programming is used throughout the BOXER system. This style of programming gives considerable flexibility while maintaining strong consistency of expression and a uniform representation. The object oriented programming style within BOXER is closely related to the Flavors system [f1fSYM1, SYM2, CAN1, MOO1].

Parts, tools, sensors, rules, etc. are objects. Each object is related to the others by its place in a hierarchy. For example, assume we have 3 parts; PART1 is a cube, PART2 and PART3 are cuboids with 1 long side. So the hierarchy looks like:

```
OBJECT
SOLIDOBJECT isa OBJECT
CUBE isa SOLIDOBJECT
PART1 isa CUBE
CUBOID isa SOLIDOBJECT
PART2 isa CUBOID
PART3 isa CUBOID
```

Functions that objects can perform, called methods, are inherited from the hierarchy. Hierarchical inheritance is used to limit the complexity of programming.

The structure of the hierarchy of parts requested by the user is ported from the Design stage, through Planning and into the Build stage. The methods attached to objects in the hierarchy changes. Thus, a request to an object to move itself during the design phase will
cause it to be redisplayed in the new position. The same request during planning will cause the world state to change to reflect postconditions of the move. During build the same request will cause the arm to move the part.

Since parts are predefined the hierarchy of part instances and classes are predefined. The subpart-to-part relationship is not predetermined, however. This relation is determined at design time by the user using the TDS CALL statement. (For details on TDS see JIDAL1, DAL2, DAL3). All operations in the system are done using message passing. This gives a simplified and uniform interface to all functions.

**Rules**

Rules are used in several ways in the system. They are used to state technology constraints on the design (e.g., if a measurement crosses a weld then the tolerance on the measurement cannot be less than +.010). Rules are used in the planning stage to prune the tree of possible next states.

At any given time there is a current rule environment which consists of a set of contexts. Each context object is composed of a set of rules. When a new object is added the rules associated with the object are added to the current context and affect the current rule environment.

**Error Model**

**Errors and Planning**

A novel feature of BOXER is that planning is actually split into two parts. One part of the planning is done using a world model and a virtual robot. This planning has the responsibility to verify that the sequence of actions that the user requested will in fact have all of its pre- and post-conditions satisfied. This type of planning is distinguished by its ability to insert states into the user plan to avoid known collisions or incorrect states.

A second form of planning occurs during the build cycle. This is the precomputed sequence of actions, known as schemas JIRUS1, necessary to recover from errors that occur during the assembly. For example, a tolerance error will raise a preplanned sequence of actions to recover from the error and return the system to a known state.

**Error classes**

BOXER contains an error model (see figure 1b). The error model is a merging of four areas: errors, sensors, program control flow and runtime planning.

BOXER introduces the concept of a SENSE-OBJECT. Observe that sensors are used because errors can occur. If there were no errors then there would be no need for sensors. Also observe that error recovery represents a departure from the main task that is triggered by some sensor. Thus there are 4 parts to the SENSE-OBJECT: There is the cause (the error), the trigger (the sensor), the effect (the error handler call), and the goal (the handler must return to the main task). These can be nested to any arbitrary depth.

Errors are checked for, and corrected, at all stages of the processing. At the design step each TDS PLACE is checked to ensure that objects do not intersect, for example. If a plan has been designed then the action object is recreated by "replaying" the steps of the design to rebuild the object. If the edited version contains an error the user is informed and the change is not made. All of the errors that occur during design are considered recoverable since the user is able to correct them.

At plan time errors take the form of states that have no transition rules between them. There are known techniques for handling some of these problems [NILL], most notably the DCOMP and ABSTRIPS work.

Certain plan time errors cause situations to arise which have no known solutions. This represents an area of research.

At runtime, errors are due to the difference between the model world and the real world. They are also caused by statistical variations, operator mistakes, etc. Certain of these errors are recoverable and BOXER has a control flow architecture that deals with the recovery. Others cannot be recovered and operator intervention is required.
BOXER MODEL

FIGURE 1a
<table>
<thead>
<tr>
<th>ERROR</th>
<th>SENSOR</th>
<th>STRATEGY</th>
<th>HANDLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect World</td>
<td>-------</td>
<td>None</td>
<td>User</td>
</tr>
<tr>
<td>Tolerance</td>
<td>Force</td>
<td>Retry</td>
<td>User</td>
</tr>
<tr>
<td>Misalignment</td>
<td>Touch</td>
<td>Search</td>
<td>User</td>
</tr>
<tr>
<td>Missing Feature</td>
<td>Vision</td>
<td>New Part</td>
<td>System</td>
</tr>
<tr>
<td>Missing Part</td>
<td>Vision</td>
<td>New Part</td>
<td>System</td>
</tr>
<tr>
<td>Dropped Part</td>
<td>Vision</td>
<td>Clear and New Part</td>
<td>Operator</td>
</tr>
<tr>
<td>Other Error</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ERROR MODEL**

*FIGURE 1b*
The Design System

Overview

The primary purpose of the design system is to capture the information necessary to correctly assemble the desired object from the set of predefined parts.

BOXER's primary user interface is through a solid modelling system (SMS).

A novel feature of BOXER is that the design system's intent is to capture assembly information, not to design parts. BOXER's SMS differs considerably in intent from other solid modelling systems although it is similar in appearance.

There are 3 categories where BOXER's SMS differs in concept from other systems (see figure 2A). First, BOXER does not include a group of operators for defining new objects because objects are assumed to be defined before the system is used. Thus, there is no "merge" operator, for example.

Second, BOXER redefines the meaning of certain SMS operators. A notable example is "join". Joining two objects will cause them to appear connected within the system, as is usual for an SMS, but joints of intersecting parts are not legal.

Third, BOXER adds functions, including the ability to handle text forms of objects. In fact, one of the primary goals of the BOXER SMS is the capture and manipulation of these text forms.

Architecture

Design System Interfaces

There are 3 interfaces to the design system. (See Figure 2B). The first is the user interface which consists of various graphic devices, pick devices and windows. This interface is used to extract goal object information from the user.

The second is the program interface which consists of a set of calls that allow a program to use the facilities (eg swept volume) of the SMS system. This interface is used by the planner's verify stage for things like collision avoidance and path planning.

The third is the knowledge base interface which consists of a set of facilities and knowledge base access procedures. This interface is used by the knowledge base to provide information to the SMS and to provide storage of the goal object information.

Definitions

Design sequence information is captured in text form called TDS (a homonym for tedious).

A design is complete when the goal object (that is, the object currently built from the set of selected parts) can be rebuilt from the TDS sequence specification. Note that a complete design does not imply that the designer has finished building the object. The DESIGN system attempts to achieve the complete state at the end of every TDS operation. Generally, the PLANNER may be invoked at any time on a complete design.

A design is finished when the designer has a complete design of his desired goal object.

A design is correct when the desired goal object can be built by the BUILD system.

Constructing a Design

The user interacts with the design system by manipulating part images on a screen. He is expected to combine these parts into the final assembly using a valid assembly sequence.

As the user requests parts the system remembers the part type (eg PART1) and the user's name for the part (eg LEG1) in a TDS GET statement. (eg GET PART1 AS LEG1). As the part definition is retrieved from the knowledge base active elements in the part definition are invoked. This will cause side effects to occur such as computing a default pallet position for the part.

Another important side effect is that all of the superclasses of the requested part are also retrieved (eg PART will bring in the objects RECTANGLEOBJECT, PLANEROBJECT, and OBJECT if they are not already defined).
**Editing a design**

As mentioned above design sequence information is captured in text form in TDS. The user may edit the text object directly rather than using the SMS front end. Direct editing of the text object will cause the system to "re-execute" the design sequence. In general this will ensure that the modified sequence is a valid assembly sequence.

In addition to editing sequence information it is possible to edit other objects such as the pallet object. This object has the information about where the parts will be in the workspace at build time. Normally, when a part is selected, a default pallet position is computed. The user can change this default position by editing the pallet object.

**Issues**

**User interface**

The user interface is crucial to the success of a design-to-build system. The system must appear fast. Experience has shown that users will reject a system that operates slowly even if it provides significant new functionality. Thus, there needs to be a partitioning between functions that appear fast to the user and those that do not. Functions that take a larger amount of time are either executed in parallel with user interactions or are deferred until plan time.

**Analogy**

There must be a clear analogy drawn between screen manipulations and real world manipulations. This gives a sense of meaning to the user’s actions. For example, selecting a part to add to the assembly during the DESIGN phase is equivalent to retrieving the real part from the pallet location during the BUILD phase.

**Error handler specification**

One long-term problem is to state how the user can specify an error handler. This will allow the user to control the recovery from expected errors that he can foresee. Schemas appear to be a reasonable solution to this problem. The user is given a set of default schemas that can be customized to fit the user’s need.

So, for example, in figure 1b the SENSE-OBJECT for a MISSING FEATURE error has an error handler named NEW PART. The general form of NEW PART is a sequence of states and actions that mean:

1. remove the part
2. move part to junk
3. request new part
4. get new part
5. return to initial pose
6. re-enter program

where removing the part requires reversing the steps to return the system to a major state. Major states occur at strategic points with one of the requirements being that all steps between major states are reversible. The initial pose position at the start of a GET or PLACE command is always a major state.

If the user has a requirement, for example, not to junk the part he can modify this schema by creating a new SENSE-OBJECT. The new SENSE-OBJECT might be called NOJUNK. The error handler within NOJUNK might have the modified sequence of:

1. remove the part
2. reorient the part
3. return to initial pose
4. re-enter program

with a similar trigger of MISSING FEATURE. The user can then specify that this SENSE-OBJECT should be used in lieu of the default (or prior to the default). The planner will insert the necessary runtime checks to perform the error recovery.

**Knowledge base transparency and superobjects**

The knowledge base is not reflected to the user. He requests parts from a part list. Parts are represented by objects in an object-oriented programming system. When an object is requested then that object, its methods and demons, and its superobjects are brought into memory.
Overlap of Solid Modelling System functions

FIGURE 2a
FIGURE 2b
The Planner

Overview

The primary purpose of the planner is to convert user task level descriptions into a task specific program. The planner can best be characterized as a hierarchical planner since there are 2 levels of plans developed. The first is the plan the user defines and the second is the refinement done during verification.

Architecture

The Planner operates in 3 phases, expand, verify, and generate.

The expand phase takes the design system output and converts the sequence information to an internal form. The design system output consists of several objects including the sequence information (TDS code), the pose information, the user classes, the goal object and the parts list. The knowledge base contributes the technology clauses and rules, the sensor models, the tool libraries, the macro code, and the part definitions to the expansion.

Verify examines this internal form of the user's plan (the macro plan) to detect and correct inconsistencies in the plan. The corrections to the plan (micro-planning) takes the form of rearrangements or insertion of states into the user's plan.

Checks are made to ensure that errors such as moving an object that has been affixed to another object or collisions between two objects do not occur. (Collision detection is done by using the SMS to generate swept volumes and doing intersection checking).

The generate phase takes the verified internal form plan and generates executable code for the target build system. The target build system is assumed to be a high function robot with a general purpose programming environment capable of error detection.

Definitions

A plan is complete when the task-specific code has been generated from the TDS code and has successfully passed all of the plan time checks.

Expanding a design into a plan

The ideal robot world

In order to make this part of BOXER tractable an ideal robot in an ideal world is used as the target of the expansion phase. Error modelling is added during verification. The concept of a “virtual robot” is defined. Such a robot has useful properties:

- Rectangular workspace coordinates
- unlimited joint travel
- sensors (not transducers)
- unlimited joint velocity and acceleration
- perfect accuracy
- volume-less joint elements
- unlimited payload

Macro expansion

A useful analogy can be made at this stage with the act of compiling a High Level Language. During compilation the external source code is converted into an internal form of tables containing state and type information. In BOXER, this phase changes TDS (source code) into an internal form suitable for representing actions and states in an ideal robot world.

Pose expansion

One of the key pieces of information available from the SMS is the position and orientation (pose) of the parts relative to key points (fiducial points) in the workspace. Within the SMS these poses are referred to symbolically. At this time this information is expanded into numeric form as DH matrices.

Pose precomputation

Many times a specification of a location may be relative to a point which is relative to another point (which may be relative to still another point). Ultimately the whole chain must be relative to a fiducial point. In some cases it is possible to precompute the final pose and eliminate the intermediate transforms.
Technology clauses

In TDS parts are connected to each other using an ATTACH statement. The attach statement does not specify what technology is being used to perform the attachment. Since this decision is not made at design time (although it can be if the user needs to) the code to perform intermediate steps (eg pick up the tool) is not part of the user plan.

User clauses

User clauses fall into two categories. First, the user may have chosen to add a new statement to the language. This is possible because the underlying development environment is available. A user statement (eg. RE-ORIENT) will have several components that the user will have to define. The one of interest here is the expansion of the user statement into the internal form used by the planner.

The second, and probably more common, user clause is the addition of a new SENSE-OBJECT to the system. The user will have specified a (possibly) new cause, trigger, effect and goal. These are expanded and added to the planning.

Verification of the plan

In general the planning that is required during verification is of the means-end analysis class. One is usually required to generate a micro-plan that will match up two closely related states. This occurs because the user has already done the macro-planning for us.

Another form of planning is checking for conditions that can be found through analysis of the object positions and motions. For example, one can check the stability of the object after final placement.

Affixment

The affixment tree can be built to show how things are attached to each other. Attachments are of four flavors: not-attached, jointed attachment, non-rigid attachment, and rigid attachment.

Not-attached parts are obvious. Jointed attachment are connections between parts that may have some degrees of freedom. Non-rigid attachment is where one part depends on the position of another but not vice-versa. For example, a coffee cup on a saucer de-
then the planner will back track to the previous step, or the following step if the previous step has been applied.

In general, the naive expansion of the TDS sequence into plan states and actions will leave gaps between two adjacent plan steps. For example, an ATTACH operation may leave the tool in the hand. The subsequent step may be to pick up another part. The verify stage will insert the states and actions necessary to put the tool down and guarantee that the pre-conditions of the subsequent step are satisfied.

If an operation has both normal and exception outcomes there may be a different postcondition for each possible outcome. BOXER generates these multiple postcondition states for each checked condition on each motion. Then it verifies that the error handler will successfully recover the desired state from the error state.

Previous planners used states as the primary planning element. The states were modelled as predicates. For example, STRIPS modelled the blocks world using ON(A,B) to represent the state of block A being on block B. Strips rules are not appropriate for actions requiring naive physics (e.g. dominos) "NINIL"

BOXER models states but uses action predicates as the focus of its attention. Thus MOVE(B) would be a primary working predicate. States and actions need to be clearly distinguished.

During the verify step the planner has a set of states that represent the current world state and a set of actions that represent the possible actions that can be performed on the world state. The world state is represented as a list of states of objects in the world. The plan is represented as a sequence of these lists.

\[
\text{step1 = (handempty initialpose} \\
\text{(force\text{-}transducer .7)} \\
\text{(...))}
\]

\[
\text{step2 = (handempty} \\
\text{(over part1)} \\
\text{(force\text{-}transducer .7)} \\
\text{(...))}
\]

\[
\text{stepN = (handempty initialpose} \\
\text{(force\text{-}transducer .7)}
\]

\[
\text{action1 = ((preconditions handempty} \\
\text{initialpose} \\
\text{(force < 1.) ...}) \\
\text{(add (over x)))} \\
\text{(delete initialpose))}
\]

\[
\text{actionN = ((preconditions handempty} \\
\text{initialpose} \\
\text{(force < 1.) ...}) \\
\text{(add initialpose))} \\
\text{(delete (over x)))}
\]

BOXER differs from other planning efforts in that it does not try to plan the assembly from scratch but uses planning only to ensure that the steps that occur from one world state to another are achieved. Since the user have done the gross level planning then the system need only deal with problems like making sure that the hand is empty when the next action requires picking up the next part. BOXER has a novel interaction between actions and contexts that is illustrated in this example.

Let's assume that the next step is to pick up a part and that a precondition of picking up a part is to have the hand be empty:

\[
\text{actionx = ((precondition} \\
\text{(handempty initialpose ...}) \\
\text{(add (over x)))} \\
\text{(delete initialpose))}
\]

Also assume that the hand is not empty but is holding a tool:

\[
\text{stax = ((handfull toolx) initialpose} \\
\text{(force\text{-}transducer .7)...})
\]

Now we have reached the traditional planning problem of how to achieve the HANDEMTPTY state given that we are holding TOOLX. There are 2 things to note. First, the planning is being applied in a very restricted problem so that the complexity is reduced. Second, within BOXER there is normally an action that will achieve the HANDEMTPTY state called OPENHAND. However, this is not an appropriate operator in this context because that will cause the tool
to fall. When TOOLX is initially picked up the planner hides the rule OPENHAND beneath a rule called PUTTOOL. This rule will cause the system to put the tool away rather than drop the tool. The PUTTOOL action is added to the set of actions when TOOLX was picked up. This set of actions is called the "context set". Every tool has a PUTTOOL action associated with it.

**Technology Rules**

Each fastening technology has a different set of rules for generating correct tasks. For example, welded frame fastening requires a certain sequence of welds to ensure that the frame does not warp. Each weld heats the frame which causes it to distort. Making a weld close to a recently completed weld will cause the warped member to be welded in the wrong position.

The rule set associated with a technology checks the generated code for correctness according to the rules.

**Generating the task-specific program**

**Hand assignment**

For multi-armed robots, or eventually for multiple discrete robots, there is a question of hand assignment. Generally, if the hands interact one hand will be used for transport and holding and the other hand will be the tool using hand. When the hands do not interact then the task can be partitioned along the midpoint of the goal object.

There is the possibility of an optimal assignment of sub-tasks to hands but it will not be addressed here.

**Robot motion calls**

The plans required to map planner internal actions to robot motion call sequences are created during this phase. In general, one has a plan represented as a set of internal states and actions and one needs to find a sequence of robot actions that will properly execute the plan. What that sequence of actions is depends strongly on the target BUILD system. Fortunately, because a virtual robot is used during planning, one can work out in advance the mapping from a virtual robot motion to the corresponding real robot motion. This ignores problems like point correspondence and range of travel which are dealt with in the BUILD system.

**Error Objects, Sensors, and Transducers**

During the verify phase certain virtual robot actions generate error checks (e.g. closing the hand will generate a presence check). During generate these error checks are mapped to error objects that contain methods capable of detecting the error. The particular sensor and transducer combination is sensitive to the final BUILD system configuration. For example, presence checking could be sensed by force transducers in the finger or by a camera system mounted above the robot workspace. The error objects contain error handlers that get added to the final code stream.

**The virtual to real robot correspondence**

The planner has assumed a virtual robot that has certain useful properties. In general, these properties will not be fully available in the real robot (e.g. joints do not have unlimited travel). Part of the correspondence problem is solved in the generate phase of the planner. For example, the plans required to map planner internal actions to robot motion call sequences. Part of the correspondence problem is solved during the startup phase of the BUILD system. For example, there is a mapping of virtual points to points in the robot workspace that gets fixed during calibration. Part of the correspondence problem is solved by explicit error checks made during pre- and post-motion error checking. For example, a range check is enabled for motion that will exceed the robot's range of travel.

**Issues**

Discussion of problems within the planner naturally break into the 3 categories of expand, verify and generate.

**Expand**

**Macro expansion**

There are several issues in the expand phase that require detailed examination. First, there is the mapping from the TDS code to the internal form used for planning. The TDS code represents a desired sequence of operations. However, this sequence is really a list
of actions to be performed on the world state to achieve the final assembly.

BOXER assumes an initial world state (which the user can change by selecting the world state window in the SMS). Each TDS statement translates into a set of preconditions, actions, and states that can be applied to the world state.

In general, the generated sequence will not be complete. Preconditions in the generated code will not match the present world state. These problems are resolved by the verify phase.

Non-picture object representation
One of the difficulties of using solid models is that, to use them for planning purposes, one has to reason with them. The science in this area is not well developed. BOXER deals with solid models by making calls to the SMS system.

Verify
Non-Commutative operations
Non-commutative (eg WELD) operations cause permanent state changes and may inhibit regression of goals. All non-commutative operations generate a major state and represent boundaries for the error recovery routines.

Concurrent Events
Concurrent events are not handled by the planner except to generate constraints for the hand assignment algorithm. Certain operations will require holding the part after it has been placed because the part is not stable. The hand assignment algorithm must assign the other arm to a micro-plan that will make the part stable and free the first arm.

Generate

Ambiguity
In general, the conversion of task space, non-ambiguous pose information into robot space poses will generate ambiguous cases (eg a point is reachable in LEFT and RIGHT hand mode). There must be a rule set associated with each BUILD system robot that removes the ambiguity. Conversion of task space, non-ambiguous pose into robot space ambiguous pose.
FIGURE 3
The Build System

Overview
The primary purpose of the build system is to provide a reliable, fault tolerant environment for the task specific program.

The build system provides an execution environment for the task specific program. This execution environment is composed of 3 parts. The startup phase initializes the world and finds the fiducial points. The execution phase provides the services necessary for the running program and raises the level of abstraction to make the runtime environment convenient. The error handling system creates a fault tolerant environment to deal with real world errors.

Architecture

Definitions
A task specific program is correct when the goal object specified in the Design stage has been constructed.

A task specific program is finished when it exits.

Starting a task specific program
Starting a task specific program requires actions that bring the world to a known initial state and verifying that all assumptions made by the planner are true. Calibrating a fiducial point is one of the tasks that bring the world to a known state. Verifying that the initial silhouette of the pallet matches the stored silhouette is an example of a task that verifies an assumption made by the planner.

Executing a task specific program
Every actual robot operation (eg MOVE) is surrounded by three primitives:

checkbefore(collision, bounds, state)
checkduring(collision, tolerance)
move()
checkafter(position, state)

where each of the parameters (eg tolerance) is a SENSE-OBJECT. That is, it enables a trigger (some set of transducers that make

up a sensor) which, if tripped, will call an error handler with a specific goal.

Error handling in a task specific program
One of the key problems of the build system is designing a control flow so that the system can detect and recover from errors. BOXER follows several rules that make this easier to accomplish.

There are several key elements within the BOXER control flow. First, there is the idea of an initial pose. This is a state that is achieved a critical points in the program. It causes the trajectory from picking up a part to placing a part to be broken into two steps. This idea gives the planner a method of independently planning the pickup and the place motions.

Second, there is the idea of major and minor states within the program. Every major state represents a place where an error handler can branch to rejoin a program that was in progress. Major states always begin and end in an initial pose state. Minor states are actions that occur between major states. All minor states are reversible.

If an error does occur then the error is signalled and an error handler is dispatched. The error handler reverses the sequence of minor moves until the initial pose at the last major state is achieved. It then executes the recovery sequence (eg trashes the offending part, requests a new one, places the new part at the correct position, and returns to the initial pose). The recovery sequence ends in the initial pose state. The error handler will then rejoin the task in progress.

Issues

State maintenance
The use of error handlers and runtime planning requires a systematic method of state maintenance. Every operation (eg move) has to be enhanced to update the state of the world. Certain operations (eg GETTOOL) will also cause new context sets to be established. These are necessary so that the error
recovery routine will correctly return to the current state using the proper operators (eg PUTTOOL rather than OPENHAND).

Tolerance (how do you know that you got there?)

Tolerance has two meanings within the BOXER context. The first meaning is the question of how to specify tolerance within the part definitions. This is a current research issue and is not addressed.

The second meaning is stated best as the question: How do you know you got there? This is addressed using the force sensors to sense final forces in placing an object. An out of range force (too low or too high) is generally taken to mean that an error has occurred. This may not necessarily be true. This will be one source of spurious error events.
FIGURE 4
Knowledge base

Overview
The primary purpose of the knowledge base is to be an environment that links the various stages of BOXER into a consistent whole.
The careful design of the knowledge base is important. The knowledge base elements embody key representational decisions.

Architecture

Definitions
Objects are the usual object oriented programming constructs. Concepts are objects that have roles and exist in the knowledge base. Parts are both objects and concepts and they correspond to things in the real world. Transducers correspond to physical objects in the real world capable of detecting and reporting changes. Sensors correspond to a logical function performed by one or more transducers. Usually one sensor is mapped onto one transducer but this is not always the case. One counter-example is a vision system. The camera (transducer) can be used as a presence sensor (sensor) and as an edge detector (sensor).

Knowledge Base Structure
Currently, the knowledge base is physically organized as a linear sequence of object definitions. The logical organization is a set of overlapping trees. Objects within the database are linked to other objects by the inheritance mechanism. Objects can inherit the methods of multiple ancestors (ala the Flavors system—SYM1, SYM2, CAN1, MOO1).

Knowledge Base Access
Accessing an object will cause "active elements" within the object to be triggered. These active elements have side effects such as computing a default.
Accessing an object will also cause the ancestors of an object to be defined if they do not already exist. At the time an object is defined it is sent an initialization message.

Issues

The object library and the knowledge base
It is assumed that the BUILD system is sufficiently rich to enable the object oriented programming style to be used. The hierarchy of object relations and their associated methods are common throughout the system.
The BUILD system will have objects that are sensors, transducers, tools, parts, etc. These objects are all representative of corresponding objects in the real world. There is a piece of research required to examine how best to represent the real world objects.
KNOWLEDGE BASE

Tools
- Screwdriver
- Tack Welder
- Glue Gun

Feeders
- Vibrating Bowl
- Conveyor
- Slide Tracks

Fixtures
- Part Set
- Rule Set
- Methods Set

Parts
- Part Set
- Rule Set
- Methods Set

Sensors
- Strain Guage
- Leds
- Vision

User Objects

Technology
- Screwing
- Welding
- Glueing

Robots
- 7565
- ASEA
- PUMA

Process
- GET
- PLACE
- ATTACH

FIGURE 5
Learning component

Overview
The primary purpose of the learning component is to analyze a plan failure and update what the BOXER system knows.
This learning has two effects: plans will be modified by the new knowledge and the user will be warned that constructs might not be manufacturable. These form the criteria for measuring performance in the learning task.

Architecture

Definitions
A failure represents an error state of a plan that has been detected by a sensor.

Learning Component Structure
Currently, the learning component is organized

Knowledge Base Access

Issues

The object library and the knowledge base
Conclusions

Conclusions

BOXER is a system that merges the research in solid modelling, planning systems, knowledge bases, and high function robots. Its major contribution is combining these in a coherent framework that can be applied to assembly automation.

An error model has been constructed that will deal with runtime errors normally encountered during the assembly process. This error model represents a key step forward in automating assembly.

It has been shown how sequence information derived from the user can be used to automatically construct a task-specific program for an assembly.

One of the key benefits of a design-to-build system will be an improvement in the turn-around time for new products. Instead of taking several months to get a prototype assembly, a BOXER system will generate the prototype immediately. Ultimately, this translates to lower cost for products and shorter product cycles, faster response to marketing demands and higher profits.

Work is being done at this time to implement the planner subsystem.

Acknowledgements

The author would like to thank the following people for their efforts, observations and contributions: Fred Blair, Ed Collins, Joanne Daly, Franklin Graeber, Dave Grossman, Ralph Hollis, Ranga Jayaraman, Jehuda Ish-Shalom, Robert Kay, Mike Kutzer, Mark Lavin, Tony Levas, Larry Lieberman, Bela Musits, Elaine Palmer, William Platt, Rick Sheftic, Phil Summers, Russ Taylor, Don Tolchin, and Mike Wesley.

These people have contributed generously of their time and talents to make this project what it is. To those people I have forgotten to mention, my sincere apologies and thanks.
References

AMB1
Ambler, A., Barrow, H., Brown, C., Burstell, R., Popplestone, R.
"A Versatile System for Computer-Controlled Assembly"
Artificial Intelligence Vol 6, pp 129-156 1975

AMB2
Ambler, A. and Popplestone, R.
"Infererring the Positions of Bodies from Specified Spatial Relationships"
Artificial Intelligence Vol 6, pp 157-174 1975

ARA1
Arai, Tamio
"A robot language system with a colour graphic simulator"

ASA1
Asada, Haruhiko and Bernard, Andre
"Implementing Automatic Setup Change Via Robots To Achieve Adaptable Assembly"
MIT 1984

BAR1
Barlow, R. and Proschan, F.
"Statistical Theory of Reliability and Life Testing - Probability Models"
Holt, Rinehart and Winston 1975 page 192

BEZ1
Bezier, Pierre
"UNISURF, from Styling to Toolshop"
North-Holland Computers in Industry 4 pp 115-126 1983

BIN1
Binford, T., Grossman, D., Miyamoto, E. Finkel, R., Shimano, B., Taylor, R., Bolles, R., Roderick, M., Muijaba, M., Gafford, T.
"Exploratory Study of Computer Integrated Assembly Systems - Progress Report"
Stanford Artificial Intelligence Laboratory 9/15/74 to 11/30/75

BLU1
Blume, Christian and Jakob, Wilfried
"Design of the structured robot language (SRL)"

BOB1
Bobrow, Daniel and Collins, Allan
"Representation and Understanding"
Academic Press, Inc. 1975

BUN1
Bundy, Alan
"Will It Reach The Top? Prediction in the Mechanics World"
Artificial Intelligence Vol 10 pp 129-146 1978

CAN1
Cannon, Howard
"Flavors - A non-hierarchical approach to object-oriented programming"
Draft Paper 1982

CLO1
Clocksin, W. and Mellish, C.
"Programming in Prolog"
Springer-Verlag 1981

COH1
Cohen, Paul and Feigenbaum, Edward
"The Handbook of Artificial Intelligence - Volume 3"
William Kaufmann, Inc. Los Alitos, Calif. pp 515-562 1982

DAL1
Daly, T.
"BOXER A Design-to-Build System"
Automation Research Working Report No. 4 IBM T. J. Watson Research Center Sept 19, 1984

DAL2
Daly, T.
"BOXER A Design-to-Build System - User Manual"
In preparation IBM T. J. Watson Research Center 1985

DAL3
Daly, T.
"BOXER A Design-to-Build System - Internals"
In preparation IBM T. J. Watson Research Center 1985

DAL4
Daly, T.
"BOXER A Design-to-Build System"
Proceedings of the ASME 1986 Winter Annual Meeting Anaheim, Ca. 12/7/86

DAL5
Daly, T.
"KROPS - Integrating Knowledge Representation and Rules"
In preparation IBM T. J. Watson Research Center 1985

DAR1
Darringer, J. and Blasgen, M.
"MAPLE: A High Level Language for Research in Mechanical Assembly"
IBM research report RC5606 9/3/75

END1
Platt, William
Meeting notes 11/28/83

FAH1
Fahlman, Scott
"A Planning System for Robot Construction Tasks"
Artificial Intelligence Vol 5 pp 1-49 1974

FIK1
Fikes, R., Hart, P. and Nilsson, N.
"Some New Directions in Robot Problem Solving"
Edinburgh University Press 1972

FIK2
Fikes, R., Hart, P. and Nilsson, N.
"Learning and Executing Generalized Robot Plans"
Artificial Intelligence Vol 3 pp 251-288 1972

FIK3
Fikes, R., and Nilsson, N.
"STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving"
Artificial Intelligence Vol 2 pp 189-208 1971

FIT1
Fitzgerald, W. Gracer, F., Wesley, M., Wolfe, R.
"The IBM SOLID modeling system user guide"
IBM research report raf 42 6/1/82

FOL1
Foley, James and Van Dam, Andries
"Fundamentals of Interactive Computer Graphics"
Addison-Wesley Publishing Company, Inc. 1982

FOX1
Fox, Barry R. and Ho, C.Y.
"A relational control mechanism for flexible assembly"

GOL1
Goldberg, Adele and Robson, David
"SMALLTALK-80 The language and its implementation"
Addison-Wesley Publishing Company 5/83

GOS1
Gossard, David
"CAD research activities at MIT"
Lecture at T. J. Watson Research Lab 9/83

GRE1
Green, Cordell

GRO1
Grossman, David
"A Critical Evaluation of MAPLE as the Second System User Language"
IBM internal memo. unpublished. 3/1/76

GRO2
Grossman, David and Taylor, Russell
"Interactive Generation of Object Models with a Manipulator"
IEEE transactions on systems, man. and cybernetics vol smc-8, no. 9 9/78

HAR1
Harrington, Joseph Jr.
"Understanding the Manufacturing Process"
Marcel Dekker, Inc. N.Y. 1984

HAY1
Hayes-Roth, Frederick, Waterman, Donald, and Lenat, Douglas (ed.)

HIN1
Hinton, E. and Owen, D.
"Finite Element Programming"
HOLI
Ralph Hollis
Academic Press 1977 3/1/76
Meeting Notes
11/84

IBM1
IBM
"A Manufacturing Language Reference"
AMS Boca Raton, Fl Pub #8509015
11/82

IBM2
IBM
"A Manufacturing Language Concepts
and Users Guide"
AMS Boca Raton, Fl Pub #8509012
12/82

IBM3
IBM
"Graphical Data Display Manager
Base Programming Reference"
White Plains, N.Y. Pub #SC33-0101-3
June 1983

IBM4
IBM
"LISP/VM Reference Manual"
T. J. Watson Research Center
Computer Science Department May 31,
1984

JAP1
Jappinen, Harri
"A model for representing and invoking
robot bodily skills"
Danthine, A. and Geradin, M. (eds)
Advanced Software in Robotics pp
333-342 Elsevier Science Publishers
B.V. (North-Holland) 1984

KAY1
Kay, Alan
"Computer Software"
Scientific American Volume 251 Number
3 pp 53-59 9/84

KOM1
Komorowski, H. Jan
"An Abstract Prolog Machine"
Integrated Interactive Computing Systems
P. Degano and E. Sandewall (ed)
North Holland Publishing Company pp
183-191 1983

KOR1
Koren, James
"Solid Modelling Requirements for
Robotic Applications"
IBM Thomas Watson Research Center
11/12/84

KRA1
Krackinowski, Morris
Private Communications
unpublished undated

KUT1
Kutcher, M.
Private Communications
unpublished undated

LAV1
Lavin, C. and Pertin, J.
"Automatic Grasping: A case study in
accessibility analysis"
Danthine, A. and Geradin, M. (eds)
Advanced Software in Robotics pp
201-214 Elsevier Science Publishers
B.V. (North-Holland) 1984

LAV2
Lavin, M., Taylor, R., Daly, T.,
Tolchin, D. and Nackman, L.
"Gold Filling Enhancementis Release
2.0"
IBM Research Division. Automation
Research Dept. 7/7/83

LAV3
Lavin, M., Tsai, Roger, Korein, Jim,
and Srinivasan, Vijay
"GEMS: A Data-Driven Vision/Robot
System for Measuring 3D Mechanical
Parts"
Data Driven Automation Symposium
Nashville, Tennessee September 1984

LEV1
Levas, Anthony
"Teaching Robots Assembly Plans by
Example"
University of Connecticut Technical
Report CS-83-10 1983

LEE1
Liebmann, L. and Wesley, M.
"AUTOPASS: An Automatic
Programming System for Computer
Controlled Mechanical Assembly"
LOZ1
Lozano-Perez, Tomas
"The Design of a Mechanical Assembly System"
MIT AI Lab report AI-TR-397 December 1976

LOZ2
Lozano-Perez, Tomas and Brooks, Rodney
"An Approach to Automatic Robot Programming"
MIT AI Lab report August 1983

LOZ3
Lozano-Perez, Tomas
"Task Planning"
Robot Motion pp 473-498
Brady, Hollerback, Johnson, Lozano-Perez, Mason (ed) MIT Press 1982

LOZ4
Lozano-Perez, Tomas
"Automatic Planning of Manipulator Transfer Movements"
Robot Motion pp 499-535
Brady, Hollerback, Johnson, Lozano-Perez, Mason (ed) MIT Press 1982

MAZ1
Mazer, E.
"LM-GEO Geometric Programming of Assembly Robots"

MCC1
McCarty, J. and Hayes, P.
"Some Philosophical Problems from the Standpoint of Artificial Intelligence"

MCL1
McLellan, Elizabeth
"Robot Languages - the current position"

MEA1
Meagher, Donald
"The Octree Encoding Method for Efficient Solid Modeling"

References


MEY1
Meyer, J.
Private Communications unpublished undated

MEY2
Meyer, J.
"EMULA" published dated

MOO1
Moon, David and Wierreb, J.

MUJ1
Muftiha, S. and Goldman, R.
"AL Users Manual" Stanford University report no STAN-CS-81-889 12/181

MUL1
Mullish, Henry
"A Short Course in SNOBOL 4"
New York University October 1971

NBS1
Leonard Haynes
NBS Talk: RCS: The NBS Real-Time Control System 1/11/85

NIL1
Nilsson, Nils

NOV1
Novak, Gordon Jr.

OZA1
Ozaki, Hiroaki, Mohri, Akira and Takata Masaru
"On the collision free movement of a manipulator"

PAU1
Paul, Richard P.
Robot Manipulators 1982

POK1
Ralston, J., Vinciguerra, F., and Lifshay, I.
Meeting Notes  
unpublished 11/83

**POP1**  
Poplestone, R.J., Ambler, A.P., and Bellos, J.  
"RAPI: A Language for Describing Assemblies"  
The Industrial Robot pp 131-137 9/78

**POP2**  
Poplestone, R.J. and Ambler, A.P.  
"A language for specifying robot manipulations"  

**POP3**  
Poplestone, R., Ambler, A. P., and Bellos, J.  
"An interpreter for a Language for Describing Assemblies"  
Robot Motion pp 537-566  
Brady, Holehback, Johnson, Lozano-Perez, Mason (ed) MIT Press 1982

**REM1**  
Rembold, U and Blume, C.  
"Programming Languages and Systems for Assembly Robots"  
CIME 1/84 pp 61-68

**RIC1**  
Rich, Elaine  
"Artificial Intelligence"  
McGraw-Hill Book Co. 1983

**ROA1**  
Roach, J.  
"Robot Problem Solving Using An Incomplete Model"  

**ROA2**  
Roach, J.  
"A Planning-Executing Robot Simulation"  
Term Project Report CS381 unpublished date unknown

**RUO1**  
Ruoff, Carl F.  
"Comments on Robot Control Systems"  
Robotics Research pp 941-948 Brady and Paul (eds) 1984

**RUS1**  
Russell, Dan  
"SHEM: Schema based problem solving"

University of Rochester, PH.D. Thesis 1984

**SAC1**  
Sacerdoti, Earl  
"Planning in a Hierarchy of Abstraction Spaces"  
Artificial Intelligence Vol 5 pp 115-135 1974

**SEG1**  
Segre, Alberto Maria  
"Explanation-Based Learning of Generalized Robot Assembly Plans"  
University of Illinois Ph.D. ThesisUILU-ENG-87-2208 1987

**SHA1**  
Sharman, Graham  
"The rediscovery of logistics"  
Harvard Business Review September-October 1984 pp 71-79

**SIK1**  
Siklossy, L. and Roach, J.  
"Model Verification and Improvement Using DISPROVER"  
Artificial Intelligence Vol 6 pp 41-52 1975

**STE1**  
Steels, Luc  
"ORBIS: An Applicative View of Object-Oriented Programming"  

**STE2**  
Stešik, Mark  
"Planning with Constraints (MOLGEN: Part 1)"  
Artificial Intelligence Vol 16 pp 111-140 1981

**STE3**  
Stešik, Mark  
"Planning and Meta-Planning (MOLGEN: Part 2)"  
Artificial Intelligence Vol 16 pp 141-170 1981

**STR1**  
Strom, Rob and Yemini, Shaula  
"The NII Distributed Systems Programming Language: A status report"  
IBM T.J. Watson Research Center unpublished paper 1/83

**SYM1**  
"Objects, Message Passing, and Flavors"
Symbolics, Inc. February 1984

SYM2
"Program Development Tools and Techniques"
Chapter 5 1984

TAY1
Taylor, R.
"EMILY Programming Manual"
IBM Research Report RC6437 3/30/77

TAY2
Taylor, Russell
"A Synthesis of Manipulator Control Programs from Task-Level Specifications"
Stanford University report no STAN-CS-75-560 7/76

TEO1
Teorey, Toby and Fry, James
"Design of Database Structures"
Prentice Hall, Inc. 1982

TSA1
Tsai, Roger and Lavin, Mark
"Three-Dimensional Mechanical Part Measurement using a Vision/Robot System"
IBM Thomas Watson Research Center RC10506 5/14/84

UNK1
IBM
"Architecture Design Notes"
IBM Corp. Project 4610 notebook. unpublished. 9/76

UNK2
IBM

"IBM 168 frame assembly sheets and blueprints"
unknown author. unpublished. date unknown

WAL1
Waldinger, Richard
"Achieving Several Goals Simultaneously"

WEA1
IBM
Videotape of CADAM-RS/1 link
Weatherall, Alan 1983

WEL1
Wellman, B. Leighton
"Technical Descriptive Geometry"
1957

WIL1
Wilde, Douglass
"Optimum Seeking Methods"
Prentice-Hall, Inc. 1964

WIN1
Winston, Patrick H. and Horn, B.
"Lisp"
Addison-Wesley Publishing Company 1984

WIN2
Winston, Patrick H.
"Artificial Intelligence"
Addison-Wesley Publishing Company 1984

WIN3
Winston, Patrick H.
"The MIT Robot"