A FRAMEWORK FOR HNADLINC VISION DATA IN AN OBJECT LEVEL ROBOT LANCUACE——RAPT

YIN Baolin

Department of Artificial Intelligence
University of Edinburgh Scotland

Abstract

This paper describes the work on using vision verification within an object level language for robot assenbly (RAPT). The framework which handles vision data is discussed in detail. The framework enables us to combine a verification vision facility with an object level language in an intelligent way. It can also handle other kinds of sensory data.

1. Introduction

There have been up to now two major approaches available for programming robots [7]. One is teach-mode programming: this is well known as programming by showing. The other is off-line programming and is referred to as programming by using some formal language. Since the off-line programming method has many advantages over the teach-mode one, there is a trend for robot users to program industrial robots by using robot languages.

It is also believed that if sensory information is used in conjunction with the robot, then the robot can do a better job. In this paper we will discuss the work of combining a special kind of vision, verification vision, with a particular robot language - RAPT [1,2,3,4]. However, the author believes that the principle of the work can also be used to deal with some other kinds of sensory data.

2. The Current RAPT Language and Its New Vision Commands

2.1. The Current RAPT Language

RAPT is a model-based object level robot command language [6,7]. Object level languages allow the human user to describe the task that he wants the robot to perform by describing the objects that are to be handled and the things he wants done with them in terms that are natural to him rather than to the robot. This information has to be converted by some computational system into run time commands that the robot can obey. In RAPT, the environment of the robot is modelled by an incomplete geometrical modelling system. Each object in the environment is represented by those of Its features which are to be used in the associated RAPT program. These features, such as planes, edges, points, etc., may be finite or infinite. In RAPT programs, the objects which are to be manipulated by the robot are explicitly represented, and the programmer specifies Intermediate states and actions on objects. These are transformed by the RAPT system into movements of the robot which will bring about the desired goal state.

RAPT is designed mainly for doing automatic assembly and allows the user to specify a set of bodies, the spatial relationships that are to hold between their features in each goal state, and their movements between one situation and the next. Spatial relationships constrain the relative positions of bodies. If multiple spatial relationships hold between two bodies then new, more restrictive relationships may be deduced to hold between the bodies. This ability is a result of spatial relationship reasoning.

The output of normal RAPT is a series *of* positions of objects in each situation. These are only planned positions in as much as they have been determined taking no account of the inaccuracies Inherent in the real world. The purpose of the current work is to allow statements to be added to the RAPT system which will allow use to be made of vision data to modify the planned positions .

The work of the RAPT system is divided into two parts: compile time reasoning and run time execution. At compile time, the system performs off-line reasoning about the positions of the objects and the actions of the robot. The results of the compile time reasoning will guide the actions of the robot at run time.

Readers who are not familiar with RAPT are recommended to refer to [1,2,3,4] in order to understand the paper better.

2.2. Verification Vision and Vision Task Specification in RAPT

A wide assortment of devices and systems exist for robot sensing of the environment. They fall Into two generic classes: contact and non-contact [8]. Vision is one of the most important non-contact sensing methods. Verification vision, as a type of visual information processing, is quite

* This research i supported by a Chinese government studentship It is also supported by the University of Edinburgh.

different from other types of vision system, and is very useful in conjunction with robots. The main characteristics that distinguish it from others are [5,10] :

(1) the system has a great deal of prior knowledge about the type, placement and appearance of the objects that form the scene,
(2) the goal is to verify and refine the location of one or more objects in the scene rather than to recognize them.

Before the verification system can be used, it must be told exactly what to look for and approximately where to look. The error between the expected ("nominal") positions and the actual ones should not be too big.

Since RAPT reasons about spatial relationships between body features, we may ask the vision system to verify the positions of some special features, e.g. edges, and then send the vision data to the RAPT reasoning system so that the actual position of the body to be verified can be deduced. Since images of edges are easy to detect we prefer to use edges in the system.

To introduce verification vision facilities into the RAPT system, a number of vision commands must be added to the RAPT language. They are the LOOK statement, the INVIOLATE statement, the TOLERANCE statement and the COMBINE statement. Some auxiliary commands are also needed in order to specify cameras and some low level vision details.

The LOOK statement. is used to specify the features to be verified by the vision system. Its main effect is to form symbolic features and relationships in the RAPT reasoning network. It also sends the necessary information to the low level vision system to enable it to find the expected edge image at run time.

when a camera has been used to find the image of an object feature in the scene, the RAPT system creates a new virtual feature for the camera and establishes a spatial relationship between this new feature and the observed feature of the object. For example, it may establish an "against" between a newly created plane feature of the camera and an edge of the object. Since the relevant vision data is not available at compile time, the new feature of the camera, and therefore the new spatial relationship, will have symbolic forms during the compilation phase.

The INVIOLATE statement specifies a constraint on the position of the object to be verified in terms of a relationship that must hold between the object and the world. For example, INVIOLATE can be used to Indicate that the bottom of the object must be against the top of the table no matter what the inaccuracies of the placing of the object are. From the view point of geometric reasoning, it provides a reliable relationship in the relationship network. This will enable the geometric reasoning system to explain vision data in a correct way.

The TOLERANCE statement specifies the maximum translational error along all the three axes of the body coordinate system. The rotation error tolerance is not discussed in this paper. The restriction of the translation error should indicate the range in which the feature is likely to be found.

The COMBINE statement provides a package for the vision task. It invokes the symbolic reasoning facility to deduce the symbolic position of the object by use of all the information included in the package. It checks whether the statements in the package are compatible or not. It also combines the information given by the INVIOLATE and TOLERANCE statements in order to deduce real restrictions on the position of the object to be verified.

The format of the package of the vision commands is like this:

```
COMBINE;
 VIOLATE/ against, bottom of bodyl, top of table;
 LOOK/ edgel of bodyl , cameral;
 LOOK/ edge2 of bodyl, camera2;
TERCOM;
```

where TERCOM terminates the COMBINE package. Detailed discussion of these vision statements can be found in f24].

2.3. The Symbolic Reasoning Facility

The symbolic reasoning facility is required in the RAPT system for dealing with vision data at compile time. During compile time, the positions of features which are created by LOOK statements have only a symbolic form and cannot be evaluated until run time, when the vision data is acquired. In this case, the RAPT inference system must deal with the symbolic form of the feature positions rather than their actual values. The result of the process will be evaluated during run time when the real feature positions have taken the places of the symbolic ones. Therefore, a symbolic reasoning facility is essential for combining sensory data with the RAPT system.

The symbolic reasoning system works in a similar way to the current RAPT cycle finder [14]. Being given two relationship chains between two objects, it will produce a constrained new relationship between the objects by means of a set of rewrite rules. The difference between the symbolic reasoning system and the cycle finder is that the features in both the input and output of the symbolic reasoning system may be symbolic and their parameters may be symbolic expressions. Detailed discussion of the symbolic reasoning facility can be found in [24].

3. The Framework for Handling Vision Information

So far we have discussed how to specify a vision task and how to reason about vision information symbolically. Now we will discuss how to provide a framework to handle the symbolic position expressions caused by introducing verification vision.

3.1. Outline of the Framework

It is connonly the case that vision data will verify not only the position of the specified body at the current situation (body instance), but also some other body instances whose positions are relevant to or deduced from this body instance. The verification vision indicates discrepancies between an expected position and an actual position in a particular situation and it will be necessary to nodifv subsequent positions in the light of this information. For example, suppose a robot moved a block to a specified position and then moved away a fixed amount waiting for the vision system to operate. 11 the verified position shows that the block is not exactly at the specified position, then we are sure that the robot hand is also not at the position where we supposed it to have been.

Wo want to avoid, as much as possible, reasoning which involves the symbolic expressions representing verified positions. Otherwise these expressions, or parts of then, would appear in a large number of places in the run time code, and the evaluation of each of them would take too much time. On the other hand, if we can find a method of using the verified position only at run time, then the run time system will work faster. Here we outline a method of using verified positions at run tine only. We will enploy two reasonable assumptions in the following discussion. They are:

1. The nominal position of a body is assumed to be accurate unless there is some evidence (e.g. vision data) to the contrary.
?.. The movement of a robot arm is assumed to be accurate over small distances.

3.1.1. Analysis of Expressions of the Body Instance Position

We first need to establish how the actual position of a body instance is related to its nominal position and the vision verification data, even after some movements have been made. A movement of a body (b) in one situation (i) to the next (i-l) can be represented by a matrix Tb(i+1) such that

PNb(i+l) = PNbi * Tb(i+1) (1)

where PNbi and PNb(i+1) are matrices representing the -.nominal positions of the body in situations i and i+1. If we consider that the body b makes a virtual movement from its nominal to its virtual position then the movement can be represented by a matrix FMbi:

FMbi * PNbi = PVbi (2)

and we call FMbi the <u>modifying factor</u> of the body instance PNbi.

We know from (2) that:

F?!bi = PVbi * PNbi"' (3)

How suppose that the nominal position of the body b in situation j, PNbj, is produced by a sequence of specified movements from its nominal position in situation i, then it can be seen that

PHbj = PNbi * Tb(i4l ,j) (4)

where
Tb(i+l,j) = Tb(i+1) * ... * Tbj (S)
and j > i .

The actual position of the body in situation j (the actual position is not a "verified position" since we have not verified it by vision commands, but we consider that it is equivalent to a verified position in our discussion) will be related to the actual position of the body in situation i bv exactly the same sequence of movements, and therefore

PVbj = PVbi * Tb(i+1 , j)
- PVbi * PNbi"^{1}* PNbi * Tb(i+l,j)
- PVbi * PNbi-' * PNbj
= FMhj * PNbj (f>)

where FIlbj - PVbi * PNbi⁻' (/)

FMbj is referred! to as the <u>modifying factor</u> of the body instance PNbj.

We can see from the discussion above that the actual position of a body instance is determined by two parts: the nominal position and a modifying factor, and it is onlv the modifying factor that is affected by the vision data. When a body instance is verified, its modifying factor is defined by the vision data and by the nominal position of the body instance. Furthermore, if we know the modifying factor of a body instance and the subsequent nominal movements of that bodv, then we can determine the modifying factors for the instances of that body in the subsequent situations. As we can obtain the nominal position for each body instance by using the current RAPT cycle finder system, we can determine the nominal movement of a body instance between any situations. Therefore, we can obtain the modifying factor for every body instance by working forward from the modifying factor of the verified body instance. We can see from equations (6) and (7) that we can also assume that once the actual position *of* a body has been found to be different from its nominal one, accurate positions of the body in subsequent situations will bear the same relationship to their nominal positions until a new modifying factor is found either by new vision data or by a specified action (see Sec. 3.2.1). Therefore, in our vision verification system we may deduce the nominal position of each body instance at compile time in the usual way, then evaluate the modifying factors at run time, and get the actual positions by matrix multiplication.

The introduction of modifying factors will affect the actions of bodies. We will refer to the action between two nominal positions as nominal action and that between two actual positions as actual action. Suppose body b is moved from nominal position PNbi to PNb(i+1) by a nominal action TNbi. We have

$$\begin{aligned} &PNbi * TNbi = PNb(i+1) \\ &TNbi = PNbi^{-1} * PNb(i+1) \qquad (8) \end{aligned}$$

On introducing the modifying factor, the positions of both the starting point and the destination of an action may be changed. Suppose body b is moved from actual position PVbi to PVb(i+1) by an actual action TAbi, then we have

$$\begin{aligned} &PVbi * TAbi = PVb(i+1) \\ &FMbi * PNbi * TAbi = FMb(i+1) * PNb(i+1) \\ &TAbi = PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNb(i+1) \qquad (9) \end{aligned}$$

3.1.2 The Run Time Data Structure

The modifying factors are stored at run time in an array indexed by body instances. The elements of the array may have three forms. The modifying factor will be an identity matrix symbol ("1") if the corresponding body instance can be assumed to be in its nominal position. The modifying factor will be a position matrix if the corresponding body instance is verified by vision commands. The modifying factor will be a pointer if the position of the corresponding body instance is dependent on another body instance which has been verified in the current situation or one of the preceding situations. The pointers and identity matrix symbols are assigned at compile time, while the position matrices are assigned at run time after the vision data have been obtained and the corresponding symbolic positions have been evaluated.

There are two possible ways to use the modifying factor array in controlling the actual action of a robot. In the first method we use equation (9) to calculate the actual action directly and then order the robot to move accordingly in one step. In the second method we divide the actual action into two parts and control the robot to move accordingly in two steps. The first part indicates the action which is caused by errors in the end points of the motion while the second part indicates the nominal movement specified by the RAPT program. From equation (9) we have

$$\begin{aligned} TAbi &= PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNb(i+1) \\ &= PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNbi * PNbi^{-1} * PNb(i+1) \\ &= TCbi * TNbi \qquad (10) \end{aligned}$$

where

$$TCbi = PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNbi \qquad (11)$$

The robot moves by an amount of TCbi in the first step and then moves exactly the amount indicated by the RAPT program to achieve its actual destination. The second method is important in some circumstances and is discussed in detail in [24].

3.1.3. Compile Time Work

The work to create the run time modifying factor array can he divided into two stapes, the reasoning stage and the simplifying stage. In the reasoning stage, all the modifying factors for the body instances in the initial "situation will he assigned an identity natrix symbol ("1"). In each of the following situations, if a bodv is verified in that situation the corresponding modifying factor will be assigned the symbolic position expression PVbi ("P"), otherwise a pointer or a set of pointers which points to another modifying factor will be assigned to the modifying factor according to rules which will be discussed later.

In the simplifying stage, the modifying factor array is simplified according to a set of rules. This process makes the array more compact. *and* speeds up the run time evaluation of the array.

.'1.1.4. Modifying Factors in Symbolic Reasoning

As well as influencing the symbolic reasoning,, the introduction of modifying iactors will also influeuce the way vision commands are used. The modifying factors from previous vision commands may enable us to make better predictions for positions of bodies in subsequent vision commands.

In a vision command package, some INVIOLATE statements may be used to indicate the most reliable relationships holding between the object to be verified and some other objects. The modifying factors for the body instances of these reference objects may not be identity matrices. This means that they may not be at their nominal positions, though the relationships mentioned in the INVIOLATE statements still hold. Discussion of how to deal with these two points can be found in [24].

3.1.2. Pules for Setting; the Pointers

The rules discussed below are used for setting the pointers in the modifying, factor array for bodies according to their status in the RAPT program such as being MOVEd , TIEd and so on. For convenience, we will refer to the rules for making the pointers as linking rules.

3.2.1. Actions

There are two kinds of action statements in RAPT: MOVE statements and TURN statements. For making a modifying factor array, however, we are more concerned with spatial relationship specifications about bodies and the association between the relationship specifications and action statements. If the position of a body instance is restricted by some spatial relationship specifications, then we refer to the position of the body instance as a <u>specified position</u>. On the other hand, if there are no explicit spatial relationship specifications constraining the position of the body Instance, then we refer to the position as an <u>unspecified position</u>.

The following are the linking rules for the body instances which are moved (i.e. MOVEd or TURNd) directly by action statements rather than by the effects of TIEs and SUBASSEMBLIES. The linking rules for TIEs and SUBASSEMBLIES will be discussed in Sec. 3.2.2 and Sec. 3.2.3.

Al If a body Is noved to an unspecified position, then the pointer of the body instance points to its modifying factor in the previous situation.

A2 If a body is moved to a specified position, then the pointer of the body instance points to the modifying factor for the body instance of the body between which the relationships hold (reference body) in the current situation unless the condition mentioned in rule (A3) is net.

A3. If the reference body in a specified position is TIEd to the bodv to be moved or belongs to the same subassembly as the body to be moved then the pointer of the body instance points to its modifying factor in the previous situation.

3.2.2. Ties

In the RAPT language bodies can be tied together during an action, and this means that they maintain the same relative position before and after the action. Therefore, any descriptions of the motion of one body which is tied to another must apply to the motions of that other. TIEs are made and revoked by TIED and UNTIED statements.

We can see, from the definition of TIE, that the modifying factors for bodies which have been tied together must keep the same relations to each other throughout the existence of the tic, except when local vision commands (see [24] for details) are used. This means that a change in the modifying factor of one member of a tie must be applied to that of the other member. Moreover, this effect will continue after the two bodies are untied until one of them Is moved by specified actions or both of them are verified by global vision commands individually (see [24] for details).

Let us consider an example in order to understand how to keep the same relationship between two modifying factors when one of them is changing. Suppose bodies A and B are tied together with different modifying factors Mal and Mbl respectively. If Mal is changed to Ma2 then the relative change for modifying factor Mal is Tr. We have

$$\begin{aligned} Mal * Tr &= Ma2 \\ Tr &= Mal^{-'} * Ma2 \end{aligned} \qquad (12)$$

and the new modifying factor Mb2 for B is

$$\begin{aligned} Mb2 &= Mbl * Tr \\ &= Mbl * Mal^{-1} * Ma2 \end{aligned} \qquad (13)$$

Therefore the modifying factor for body A in the new situation is Ma2 while Mb2 can be represented by a pointer triple [pl,p2,p3] in which pl points to Mbl, p2 points to Mal and p3 points to Ma2. At run time the triple will be evaluated by the equation

$$(pl) * (p2)^{-'} * (p3) \qquad (14)$$

The discussion above can be expanded to cover a tie linking more than two bodies and the circumstances when this large tie has been broken. Suppose n bodies bl, ..., bn are tied together by n-1 TIE statements. For convenience we will call the result a "super" tie in the following discussion.

The linking rules for TIE statements can be summarized as follows:

Tl. In a super tie, if body bj is moved in situation i by a specified action which brings about some relationships between body bj and body C, then the pointer of body bj will point to the modifying factor for body C in situation i while the modifying factors of other bodies in the super tie are pointer triples. The triples have the form [pl,p2,p3] where pi points to the modifying factor for the same body in situation (i-1), p2 to the modifying factor for bj in situation (i-1) and p3 points to the modifying factor for bj in situation i.

T2. In a super tie, if body bj is verified by a set of global vision commands in situation i, then its modifying factor will be assigned a symbolic position expression "P" while that of other bodies in the super tie are pointer triples. The triples have the same form and contents as in rule (Tl).

T3. In a super tie, if body bj is verified by a set of local vision commands in situation i, then its modifying factor will be assigned a symbolic position expression "P" while that of other bodies in the super tie will refer to their modifying factor in situation (i-1).

Five similar rules are applied after a super tie has been broken. Details can be found in [24].

3.2.3. Subassembly

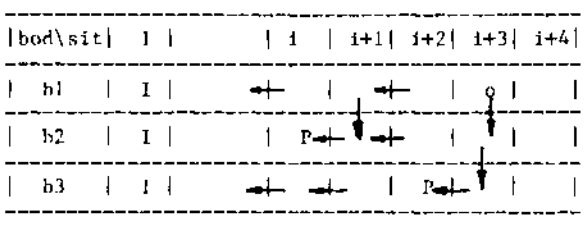
The subassembly is a set of bodies between whose features certain specified relationships hold for the duration of the existence of the subassembly. Subassemblies differ from ties in that there may be more than two bodies within a subassembly and the components of a subassembly can move with respect to each other during the existence of the subassembly, provided that the relations remain valid. Treatment of subassemblies is similar to that for HEs. Detailed discussion can be found In [24].

The following simple example shows how the linking rules are used.

```
remark bodies bl b2 b3 have been defined;

verify/b2;    remark abbreviation for vision
          command package, now in situation i;
move/bl;
   fixed/bl , b2; remark abbreviation for a set
          of relations which completely defines
          the position of bl with b2, sit i+1 ;
```

```
tied/bl,b2;
verify/b3;    remark sit i+2;
nove/b2;
  fixed/b2,b3;    remark sit i+3;
```



** o⟶ represents the pointer triple.

4. The Position of the Camera

The camera is defined in RAPT in the usual way as an ordinary body with a .specified focal length. This means that as with any other body it can be operated on by the system, being noved , tied and so on. It can also be mounted on a robot hand.]n most cases the position of a camera body can be modified by the modifying factor array according to the linking rules. For example, the position of one camera can be verified by another camera. However, if the result of a global vision command package will affect the position of the camera to be used (according to the linking rules listed in See. 3.2) then the vision command package will he dealt with as a local one. In this case, only the position of the body to be verified is subject to modification while the position of the camera is unchanged.

5. The Generalily of the Framework

So far we have discussed how to use the framework to handle verification vision information. In fact, the framework also has the capability to handle other kinds of sensor information for the purposes of verification. Generally speaking, in RAPT, when we use a sensor to detect a feature of an object, we create a new relationship between that object and the world, as the absolute position of the sensor in the world is usually known when the sensor is used. The nature of the relationship depends entirely on the type of sensor. Since sensor data is available at run time, at compile time we can only obtain symbolic relationships. when the symbolic relations are sufficient to "fix" the object, we can use a suitable symbolic reasoning system to deduce the symbolic position of the object. We can then evaluate the symbolic position expression at run time when the corresponding sensor data is available. In order to introduce a new kind oi sensor, we need only to provide a set of commands to specify how to use the new sensor, and a new symbolic reasoning system which is capable of dealing with the new relationships created by this sensor. The framework handles new kinds of sensor information in the same way as it does vision data.

Now let. us consider an example. Suppose we use a touch sensor to detect a plane feature pi of a body B. when we detect a point on pl, we create a relationship between the v/orld and the body, the "AGAINST" relationship between a sphere feature (a vertex is considered as a sphere with radius 0 in RAPT) of the world and the plane pl. If we detect some carefully chosen planes of the body B with the touch sensor and so obtain enough AGAINST relationships, then we will be able to deduce the position of the body B. The symbolic position expression of B can be stored in the modifying factor array and can be used in exactly the same way as verification vision data.

Some authors (e.g. [15]) have suggested that vision sensors have poor precision and therefore it is better to use both vision and contact sensors. Vision sensors can be used for coarse sensing while contact sensors can be used for fine resolution. In this case, the generality of the framework to handle both kinds of sensor information is very important.

6. Discussion

There are two main problems in using sensory information in programming and controlling robots. The first is how to get accurate sensory Information and the second is how to process and use the information in the system. The latter problem, as Harmon points out [8], seems the harder of the two and has more implications for the theoretical use of sensors.

Some work on using vision data in robot programming, languages has been reported, such as VAL [19, 231 and AL [20]. In the VAL system, vision is used to recognize objects in a picture taken by a TV camera, and to locate them. The recognition is done by comparing the blobs in the picture with trained characteristics of given prototypes in an effort to find a match. The vision technique used in the VAL system is of the first generation [12] , i.e. it uses a binary picture and produces a 2-9 explanation. In the AL system, vision is used to fulfill the verification task [20]. Both the languages mentioned above are classified as end-effector level languages by Koutsou [7], i.e. only the end-effector of the robot appears explicitly in the program.

Some object level languages have the ability to use some touch sensor information. For example, in AUTOPASS [21] force or torque sensor information is used to provide special threshold values in order to constrain or terminate some actions. In the LAMA system [221, the force sensor is used to detect whether a specified action terminates at a correct position or not. In both the above systems, sensory information is used as a condition in a decision tree and no further explanation of the information is made.

In contrast to these systems, the RAPT language with the verification vision facility provides a method of specifying vision tasks and reasoning about the vision data symbolically off-line in an object level language. It makes no assumptions about the relative positions of objects and sensors as a teach-method does. It allows partial information about positions to be combined with sensor information in a general way. At run time, when the vision data is available, the results of symbolic reasoning are evaluated and are used to improve the system's knowledge about its environment. This method uses grey level pictures and produces a 3-D interpretation rather than a 2-D one. The framework described in this paper enables us to combine the verification vision facility with an object level language in an intelligent way. The framework which is used by the system for handling vision data has some generality. It is not only able to handle the vision data, but also able to deal with some other kinds of sensory information, such as touch sensor information. Using the framework to implement the vision facility within RAPT requires few changes in the current RAPT system.

In this paper we have only discussed the theoretical framework. The system has been implemented within RAPT and used with simulated vision data, but as yet no link has been made with a real vision system. This will, of course, provide a real test, and we anticipate some problems in ensuring the correctness of match between the model features and their images.

Acknowledgement

I am grateful to A. P. Ambler and R. J. Popplestone for their supervision. I would also like to thank P. Brna, D. Corner, S. Cameron, R. Featherstone, R. Fisher and J. Hallam, for their helpful criticisms of my work.

References

[1] R. J. Popplestone, A. P. Ambler, I- Bellos RAPT: A Language for Describing Assemblies DAI. Research Paper. No.79. 1978

[2] R. J. Popplestone Specifying Manipulation in Terms of Spatial Relationships DAI. Research Paper. No.117. 1979

[3] R. J. Popplestone, A. P. Ambler, I. II. Bellos An Interpreter for a Language for Describing Assemblies; DAI. Research Paper. No.125. 1980

[4] R. J. Popplestone, A.P. Ambler A Language for Specifying Robot Manipulations DAI. Research Paper. No.161. 1981

[5] R. C. Bolles Verification Vision for Programmable Assembly 5th IJCAI. 1977

[6] J. C. Latombe Une Analyse Structuree d'Outils de Programmation pour la Robotique Industrielle (in French) Proc. of the International Seminar on Programming Methods and Languages for Industrial Robots. IRIA, France, June, 1979

(7] A. Koutsou A Survey of Model-Based Robot Programming Languages DAI, Working Paper No. 108. Dec. 1981, University of Edinburgh

[8] L. D. Harmon Touch-Sensing Technology Dept. of Biomedical Engineering, Case Western Reserve University Cleveland, Ohio 44106

[91 A. K. Bejczy Manipulator Control Automation Using Smart Sensors Electro/79 Convention, New York NY. April 24-26, 1979. pl6.

[10] R. C. Bolles Verification Vision Within a Programmable- Assembly System Stanford AI. Lab Memo AIM-275 1975.

[11] D. McChie, J. W. Hill Vision-Controlled Subassembly Station AI. Center, SRI International, Menlo Park, California 94025. Oct. 1978.

[12] C. Loughlin Robot Vision Survey Robotics Research Unit, Dept. of Electronic Eng., University of Hull, 1981.

[13] W. A. Perkins A Model-Based Vision System for Industrial Parts IEEE Trans, on Computers"; Vol C-27, No. ?., Feb. 1978.

[14] A. P. Ambler, D. F. Corner & R. J. Popplestone Reasoning about the spatial relationships derived from a RAPT program for describing assembly by robot. Proc. of IJCAI-83. (This Conference)

[15] C. A. Rosen, D. Nitzan Use of Sensors in Programmable Automation Computer. Dec. 1977, pl3.

[16] J. Y. S. Luh et al. 3-D Vision for Robotic Systems. Proc. of the 1st International Conference on Robot Vision and Sensory Controls, April 1-3, 1981, Stratford upon Avon, UK.

[17] J. F. Harris An Overview of Robot Vision R and D in UK and Abroad Image Analysis Croup University of Oxford, 1981.

[18] P. Villers Present Industrial Use of Vision Sensor of Robot Guidance Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France.

[19] Anonymous Unimation Vision Manual Unimation, Inc.

[20] R. Goldman Recent Work with the AL System Proc. of the 5th IJCAI, MIT, USA. 1977.

[21] P. M. Will Very High Level Languages for Robots Proc. of the International Seminar on Programming Methods & Languages for Industrial Robots, IRIA, Rocquencourt, France. June, 1979.

[22] T. Lozano-Perez The Design of a Mechnical Assembly System MIT AI. Lab. Technical Report 397. Dec. 1976.

[23] Anonymous User's Guide to VAL: A Robot Programming and Control System Unimation, Inc., Danbury, Connecticut, Feb. 1979.

[24] B. Yin Towards Combining Vision with a High Level Robot Language DAI Working Paper No.133, University of Edinburgh.