A Hierarchically-Organised Robot Vision System.

David Vernon,
Department of Computer Science,
Trinity College,
Dublin,
Ireland.

ABSTRACT

This paper is concerned with grey-scale robot vision and the interaction and interface between robot control languages and robot vision facilities. A grey-level vision system is described which provides general-purpose two-dimensional robot vision. A significant feature this system is that its organisation should facilitate the integration of several visual cues in a coherent manner allowing investigation of robot vision in less constrained environments.

The complete system is organised in a three-level analysis hierarchy corresponding to peripheral, attentive, and cognitive processing. The peripheral level is the lowest in the hierarchy and corresponds to conventional low-level visual processing such as edge-detection and the generation of edge and grey-scale information at several resolutions. The attentive level is concerned with guiding the peripheral process on a local level, specifically to build the object boundaries. The top-level, corresponding to the cognitive processing phase, is concerned with the overall scheduling of activity within the vision system and with analysis of the boundaries passed to it by the attentive level. The use of grey-scale images facilitates segmentation by boundary detection using a simple edge-detector and a dynamic boundary following algorithm. This allows the system to restrict its processing exclusively to the section of the boundary currently being constructed; a local processing technique which affords significant computational savings.

A brief review of robot programming techniques and languages is given and a simple new robot programming language, RCL (Robot Control Language), is described. This is an explicit-level or robot-level language but provides facilities for robot control in a Cartesian reference frame and for object description using frames (homogeneous transformations). The integration of this language with the vision system results in a general-purpose robot programming vision facility incorporating high-level user-trainability and user-definable language/vision interfaces. The interface enables a robot programmer to identify an arbitrary frame which he wishes to associate with, and embed in, the object in an interactive manner. A second frame which defines the way in which the object will be grasped by the robot is associated in a like manner. These frames will normally be used in the task specification to allow appropriate object manipulation by the robot.

A REVIEW

The use of grey-scale vision systems in industrial applications is very desirable [Wallace 1983, p.178]; grey-scale techniques facilitate more robust image segmentation and the ability to incorporate more sophisticated visual cues, for example, edges, texture, shading, and reflectance. Grey-scale techniques are normally more computationally expensive than those based on binary images since binary images are inherently segmented, exhibiting a simple and explicit object representation amenable to subsequent processing and analysis. This is not the case with grey-scale images: the process of segmentation is non-trivial and the two approaches to segmentation, boundary detection and region growing, are both computationally expensive. Region-growing techniques are less suited to industrial vision systems as they are more complex, more time consuming, and more difficult to realise in hardware than boundary detection techniques [Gonzalez and Safabakhsh 1982, p.21; Wallace 1983, p.180]. Consequently, segmentation by boundary detection is used almost exclusively by industrial grey-scale vision systems. For example, see [Ayache et al. 1984; deCoulon et al. 1983; Berman et al. 1982; Sze 1982; Kelley et al. 1981].

Process times quoted for grey-scale vision systems vary considerably: the system described in [Berman et al. 1982] requires 15 to 20 seconds for object identification using a VAX-11/780 minicomputer. An industrial robot vision system called Ibot, manufactured by Object Recognition Systems Inc., is based on a 16-bit processor (type upspecified) and takes in the order of 5 seconds for sensing [Iversen 1982]. The system described by Kelley et al. takes approximately 8 seconds to identify the position and orientation of a suitable object for grasping [Kelley et al. 1981, p.169]. The host computer used by Kelley et al. was not specified. A low-cost vision sytem, based on a Motorola 6800 microprocessor using a 1 MHz clock, requires 4.6 seconds to effect edge-detection in a 128x128 image [Li et al. 1983]. These systems all illustrate the inherent computational complexity of grey-scale processing and analysis. Since one wishes to reduce the sensing cycle time as much as possible, one must find some method of increasing the efficiency of the techniques used. Kelly has designed a program for extracting an accurate outline of a man's head from digital images using edge detection and demonstrates the feasibility of significantly reducing the overall computation time by the use of planning. A reduced resolution image is generated and edges detected in the reduced resolution image are used to plan edge detection in the higher resolution image [Kelly 1971]. Kelly indicates a reduction in processing time from 234 seconds without planning to 6 seconds with planning; all algorithms were implemented on a PDP-10 microcomputer. This use of multiple resolutions, i.e. image pyramids [Tanimoto 1980] or edge pyramids [Levine 1980], is recognised as being helpful for increasing efficiency or, indeed, for more sophisticated image analysis (for example, a scene analysis system VISIONS described by Hanson and Riseman uses a processing-cone architecture, i.e, a simulation of hierarchically organised parallel arrays of microcomputers using image information at decreasing spatial resolution [Hanson and Riseman 1978]). Berthod proposes the use of successive resolutions to implement an efficient search for templates of icons in an iconic matching system [Berthod 1982].

An alternative approach to the use of reduced resolution images to effect efficient image analysis, is the possibility of extracting object boundaries on a purely local basis using dynamic edge tracking [Shirai 1978, p.355-356]. A robot vision system, ANIMA-1 [Juvin and Dupreyrat 1981] and its successor ANIMA-2 [Juvin and deCosnac 1984], effects object segmentation and subsequent shape description by dynamic edge following, extracting local gradient information and using this to guide the boundary building process. Thus only those parts of the image on or close to the boundary are used, resulting is a significant computational saving. The ANIMA-2 system, which is based on the Intel 8086 microprocessor, can segment and identify an object in between 0.15 seconds and 0.5 seconds [Juvin and deCosnac 1984, p.165].

The importance of attentive processes to segmentation and analysis problems [Lee and Fu 1981, p.256; Hanson and Riseman 1978, p.130] makes it desirable that these techniques be incorporated in a processing and analysis organisation or architecture which recognises explicitly their relevance and relationship to other visual processes. Martin and Aggarwal in a review article on dynamic scene analysis indicate that much research in computer vision has followed the human perceptual divisions comprising cognitive, attentive, and peripheral processes [Martin and Aggarwal 1978]. Attentive processes operate on stimuli which impinge on the fovea, peripheral processes operate on the periphery of the visual field, "watching" for interesting features such as motion, texture, and colour, and direct the attentive processes to it: the attentive processes must be able to track the movement and attend to the detail of the objects in motion. Cognitive processes decide which area of interest should next be attended to and relates peripheral and attentive processes to the person, his knowlege, and expectations [Martin and Aggarwal 1978, pp.356-357). A more recent dynamic scene analysis system VILI [Jain and Haynes 1982] is also organised according to this three-level hierarchy and incorporates knowledge sources at each of these levels [Jain and Haynes 1982, p.43]. The knowledge sources at each level contain information pertinent to analysis at that level alone, e.g. the knowledge source at the cognitive level would contain image independent general knowledge whereas the peripheral level would be able to access image-specific information but not general information. Pau identifies a similar hierarchical architecture comprising cognitive, attentive, and feature extraction processes [Pau 1984, p.138].

A GREY-SCALE VISION SYSTEM

This paper describes the use of the three key ideas outlined above, i.e. dynamic boundary following, planning based on reduced resolution images, and an elegant organisation based on a peripheral, attentive, and cognitive hierarchical architecture. The last of these three is intended to facilitate a general-purpose and flexible environment suitable, not only for the approach being described here, but also for future research using more sophisticated visual cues. The first two ideas facilitate efficient analysis and compensate for the additional computational complexity of grey-scale techniques. The system is based on 256x256 pixel resolution images; the reduced resolution image is generated by local averaging in every 2x2 non-overlapping region in the 512x512 image captured by the framestore.

The peripheral level is the lowest in the processing hierarchy and corresponds to conventional low-level visual processing, specifically edge detection and the generation of edge and grey-scale information at several resolutions. Edge detection is accomplished using the Prewitt gradient-approximation operator [Prewitt 1970].

The attentive level is concerned with guiding the peripheral process on a local level, specifically to build the object boundaries. There are several approaches which may be taken to boundary building; this system uses a dynamic contour following algorithm which follows the local maximum gradient (derived using the edge-detector, i.e. at the peripheral level) and is capable of bridging gaps and linking short edge-segments.

The top level, corresponding to the cognitive phase, is concerned with the overall scheduling of activity within the vision system and with the transformation and analysis of the boundaries passed to it by the attentive level. The boundaries passed to the cognitive level by the attentive level are represented by a Boundary Chain Code (BCC), as described in [Freeman 1974]. Since the image is based on a square grid this chain code is a non-uniformly sampled representation of the boundary and is dependent on the orientation of the object. A process at the cognitive level resamples this BCC to generate a uniformly sampled BCC.

A shape descriptor, called the Foveal Chain Code (FCC), is used for object recognition. This descriptor is based on local edge activity around the object boundary, utilising image gradient information derived at two resolutions, both lower than the original image resolution. The components of the FCC, the foveal icons, are generated at the peripheral level; a foveal icon generation utility has been implemented which will build the icon representation at any point in the image. The cognitive level merely passes the boundary position, based on the resampled BCC, to the utility and builds a chain of these icons.

This shape descriptor is used to implement a general-purpose vision facility which allows an object, or more specifically the shape describing the object, to be automatically learned by the system using an off-line interactive training facility. The shape is taught by allowing the user to guide a cursor to the approximate location of the object, the boundary is then built automatically using the segmentation process of the attentive and peripheral levels, and the resulting resampled BCC is used to generate the FCC. This is repeated for a number of samples (the number is specified by the user) and the FCC giving the greatest similarity to all others is chosen as the template FCC. The FCC icon values are then written to a user-defined file for later use by on-line shape indentification routines. A facility has been implemented which effects a simple image search for an object. It checks the segmented shape against a library of trained shapes, all represented by FCCs. The facility returns an integer value denoting the template file number (in the directory of defined shape files) that best matches the extracted shape.

In addition to its shape, it is necessary to describe the object in some useful manner to facilitate manipulation using the robot control language. Homogeneous transformations can be used to describe the position and orientation of objects in a manner which is particularly useful for computer vision and robot manipulation [Paul 1981, p.9]. The homogeneous transformation was first introduced as a data structure for this type of description by Roberts [Roberts 1963]. A significant advantage is that if the relative position and orientation between two objects is represented by homogeneous transformations, the operation of matrix multiplication of homogeneous transformations can establish the overall relationship between any two objects. The robot programming language RCL, described below, uses frames (homogeneous transformations) to describe object position and orientation. The cognitive level interfaces to RCL, using the RCL image analysis functional primitive VISION, by returning two transformations. The first transformation describes the position and orientation of a frame defined by the user to be associated with, and embedded in, the object in an interactive manner. The second frame which defines the way in which the object will be grasped by the robot is associated in a like manner. Both these frames are defined during the off-line training phase and stored as part of the shape description template. Using the VISION primitive, the RCL programmer may also stipulate a local area-of-interest within the image to restrict image analysis based on available a priori knowledge.

A REVIEW OF ROBOT PROGRAMMING METHODOLOGIES

Modern commercially-available robot manipulators make use of many programming methodologies which exhibit a wide spectrum of sophistication; this reflects the evolutionary nature of industrial robots and their associated programming systems. Lozano-Perez identifies three main categories of robot programming system: guiding systems, robot-level or explicit-level systems, and task level systems [Lozano-Perez 1982, p.1-2]. These first two of these systems, which are listed in order of increasing sophistication, are typified by the manual lead-through approach and the language-based position specification approach. The task-level systems are concerned less with the movement of the manipulator and more with the movement of the objects comprising the task; tasks are specified typically in a goal-oriented fashion. Bonner and Shin further categorise the robot-level systems into a primitive motion level and a structured robot programming level [Bonner and Shin 1982]. Each of these four approaches will be discussed briefly in turn.

The guiding, or point-to-point, level represents the most commonly-available type of robot programming system [Bonner and Shin 1982, p.86; Latombe n.d., p.1]. These systems provide programmed robot control by allowing the user to save a series of joint coordinates (specifying the manipulator position and orientation) by guiding the robot through the required motions. T3, the programming system for the Cincinnati Milacron T3 manipulator [Tarvin 1980], is an example of such a system. T3 facilites programming in Cartesian, cylindrical, and joint coordinates systems and program editing and revision are accomplished by stepping through the motions using a Teach Pendant (a special-purpose keypad). A more sophisticated guiding system, Funky [Grossman 1977], developed by IBM, exceeds the capabilities of T3 in that it provides a mechanism for centering the gripper about an object using tactile sensors in the end-effector. The principle advantages of these point-to-point systems include the fact that they are commercially available and have a proven track record, the task can be repeated without operator intervention, and the resultant programs are easy to debug since testing is inherent in the teaching process. However, this programming method has several severe disadvantages. There exist minimal facilities for branching and for generating subroutines and, typically, there is no software to handle emergencies, for example, manipulator/object collisions. The systems can not be expanded to allow off-line programming and consequently the robot is out of service during the entire programming phase. The emphasis of the programming is on the motion of the robot rather than on the task and its component objects. Since the system is trained to operate in a fixed manner, interaction with the robot work-cell using sophisticated sensors is difficult and very limited [Lozano-Perez 1982, p.1]. Additionally, recorded programs may be difficult to edit and coordination of several robots is impossible [Latombe n.d., p.1].

Robot programming systems which correspond to the primitive motion level effectively implement the point-to-point motion specification (described above) in language form. This language medium affords the possibility of including simple branching and subroutining facilities (generally with parameter passing) and the use of rudimentary parallel task execution. These types of system typically exhibit more powerful sensing mechanisms and capabilities: for example, the RPL robot language [Rosen et al. 1978], developed at the Stanford Research Institute, incorporates a vision facility which can determine object features and perform object recognition. Motion can usually be specified using Cartesian coordinates or using joint angles and the use of frames may be incorporated, though in a very limited sense; VAL [Unimation 1979], the programming language developed by Unimation, has limited coordinate transformation capabilities. Though this approach is superior to the guiding systems the emphasis is still on the robot motion, rather than the actual task, and one has the additional problem that motion specification using such languages may not be appropriate for inexperienced factory-floor workers [Lozano-Perez 1982, p.1].

Structured robot programming languages represent a major improvement of primitive motion level languages because they incorporate structured control constructs and they make extensive use of coordinate transformations and frames [Bonner and Shin 1982, p.87]. Robot motion, at the structured level, is defined in terms of transformations on a frame which is associated with the robot hand. Off-line programming is more feasible as long as the transformations representing the relationship between the frames describing objects in the robot environment are accurate. The ability to use coordinate transformations and frames is a major asset of these robot languages. One of the main problems of robot programming, collision avoidance, is not, unfortunately, addressed by this type of programming language and remains the responsibility of the user. The feasibility of using sophisticated sensing techniques, for example: visual and tactile sensing, to supply information about the changing and possibly undetermined robot environment is much greater among structured robot programming languages. Suprisingly, however, Bonner and Shin indicate that of the five structured-level languages that they review, only one language offers robot vision capabilities. This language is MCL [Baumann 1981], a language developed by McDonnell Douglas Corp. specifically for off-line robot programming. A structured programming robot language, AML [Taylor et al. 1982], which was not available to Bonner and Shin for review, also offers vision capabilites [Lavin and Lieberman 1982]. AML (A Manufacturing Language) was developed at IBM and was designed to be a structured, semantically powerful, interactive language for robot programming; one of the main design objectives was that AML should exhibit functional transparency so that AML subroutines can be written and then used exactly like built-in system commands. This approach is well exemplified by the AML/V sub-system: an extension of AML which provides visual sensing capabilities and is actually implemented using AML itself [Lavin and Lieberman 1982]. Another language worthy of mention is the AL programming system developed at the Stanford Artificial Intelligence Laboratory [Mujtaba and Goldman 1981]. AL offers Algol-like control structures, local coordinate systems which can be affixed to one another, and, significantly, the ability to specify motion in terms of objects grasped in the robot hand. Several extensions of AL have been developed; for example, Mujtaba describes an adaptation of MTM (Motion Time Measurement) techniques to manipulator programming with the intention of providing more succint, simpler, and more efficient programming facilities [Mujtaba 1982]. Goldman describes a system which includes knowledge about the environment and the manner in which it changes due to manipulation to provide a highly interactive robot programming system [Goldman 1982]. The major features of the structured programming robot languages are their use of frames for object description and motion specification, the ability to interact with sophisticated sensing devices, and the availability of structured programming constructs. RCL, the language described briefly in the next section exhibits all of these properties, though the structured programming constructs have not yet been fully developed.

Task-level robot programming languages attempt to describe assembly tasks as sequences of goal spatial relationships between objects [Latombe n.d.]. Thus, they differ from all the other categories in that they are object-oriented, rather than manipulator oriented. There are very few existing languages that might qualify as a task-level language. Bonner and Shin identify just one: AUTOPASS, a language being developed by IBM [Lieberman and Wesley 1977]. Unfortunately, AUTOPASS is not fully implemented. Indeed, such a language which completely conceals aids like sensors and frames from the user has not yet been realised [Bonner and Shin 1982, p.87] and is the subject of active research. Task-level languages, in general, necessitate the use of a world-modelling system, incorporating task planning, path planning, and collision avoidance; Latombe indicates that a current trend in the development of task level languages is to first solve these more delimited sub-problems [Latombe n.d., p.2].

RCL: ROBOT MANIPULATOR TASK SPECIFICATION USING FRAMES

The most common representation for the description of an objects position and orientation in robotics and graphics is the homogeneous transformation [Lozano-Perez 1982, p.12]. The use of homogeneous transformations, i.e. coordinate frames, has two drawbacks however. Firstly, the frame does not, in general, specify a robot configuration uniquely. For example, with a six degree of freedom robot, there are usually on the order of eight robot configurations which can place the gripper at a specified frame [Lozano-Perez 1982, p.12]. Secondly, coordinate frames may overspecify a configuration. Despite these drawbacks, Lozano-Perez states that frames are likely to continue to be the primary representation of positions in robot programs and, hence, a robot programming system should support the representation of coordinate frames and computations on frames using transforms. Further, transforms should be broken into translations and rotations to make them as easy to use as possible [Bonner and Shin 1982, p.95]. Since robot manipulation is concerned with the relationship between objects and manipulators, and since coordinate frames can conveniently represent such relationships, the homogeneous transformation can be used not only for the description of an isolated object but also for the description of the manipulation task itself. Paul describes an elegant approach to structural task description in terms of homogeneous coordinate transforms [Paul 1981, p.119-130]. This approach forms the basis for RCL.

The actual structure of the task is described by considering the structure of the task's component objects and, in particular, the explicit positional relationships between these objects. Since coordinate frames are to be used to describe object position and orientation, and since it may be required to describe a coordinate frame in two or more ways, a mechanism for representing and manipulating these descriptions is required. Paul suggests the use of transform equations and transform graphs for this purpose [Paul 1981, p. 37-39]. A simple example will serve to illustrate these techniques. Consider the situation, depicted in diagram 1, of a manipulator grasping a key. The coordinate frames which describe this situation are as follows.

Z is the transform which describes the position of the manipulator with respect to the base coordinate reference frame.

${}^{Z}T6$ describes the end of the manipulator with respect to the base of the manipulator

${}^{T6}E$ describes the end-effector with respect to the end of the manipulator, i.e., with respect to T6.

K describes the key's position with respect to the base coordinate reference frame.

${}^{K}G$ describes the manipulator end-effector with respect to the key, i.e., with respect to K.

The leading superscript on a frame refers to the coordinate system that the frame is defined with respect to. A major advantage of frame representations is that they can be combined, by homogeneous transformation matrix multiplication, so that one can generate a frame describing the relationship of one arbitrary (but defined) object to another arbitrary (but defined) object. Observing that in the above example the end-effector is described in two ways, one may generate two equivalent descriptions of the end-effector position by combining these frames.
Thus the the end-effector is described by both

$$Z\,{}^{Z}T6\,{}^{T6}E \qquad \text{and by} \qquad K\,{}^{K}G.$$

Equating these descriptions, one obtains the following transform equation:

$$Z\,{}^{Z}T6\,{}^{T6}E = K\,{}^{K}G$$

Solving for T6:

$${}^{Z}T6 = Z^{-1}\,K\,{}^{K}G\,{}^{T6}E^{-1}$$

T6 is a function of the joint variables of the manipulator and if it is known, then the appropriate joint variables may be computed (using the inverse kinematic solution of the manipulator).

In general, the task movements Mn, say, will be represented in terms of $Z\,{}^{Z}T6\,{}^{T6}E$ and this transform definition can then be equated to the other transforms (representing the tasks component objects) which describe the task structure. Ultimately, each transform equation may then be solved in terms of T6, which is a computable function of the joint variables, and thus T6 is used to determine the effective manipulator action. RCL is, in essence, a robot programming language to facilitate the direct interpretation of these transform equations using normal structured programming constructs. Thus, a move to the key grasp position defined above would be written in RCL as follows:

```
^T6 := INV(^Z) * ^K * ^G * INV(^E)
 MOVE(^T6)
```

The ^ suffix on the variable name is a convention intended to explicitly distinguish frame variables from other variables. If the position and orientation of the block were to be ascertained by visual means then the two frames ^K and ^G would be returned by the VISION primitive. Diagram 2 shows an image of this key; diagram 3 illustrates the frame manipulation menu used when embedding these frames in the template shape; diagram 4 and 5 illustrate the frames ^K and ^G superimposed on a boundary representation of the key.

SUMMARY

The grey-level vision system described here is organised as a three-level hierarchy, comprising a peripheral level, an attentive level, and a cognitive level. All shape identification and analysis is based on boundary descriptors built dynamically by the attentive level using edge information generated at the peripheral level. The cognitive level is responsible for overall scheduling of activity, shape description, and shape matching. The use of an area-of-interest operator facilitates efficient image analysis by restricting the contour following to specific high-interest sub-areas in the image. A 2-D shape descriptor, the FCC, is used to implement a user-friendly object training and identification facility. The general-purpose shape identification routines interface to the robot programming language RCL in a standard and coherent manner using the languages own object description data-structures: this results in a general-purpose robot programming vision facility incorporating high-level user-trainability and user-definable language/vision interfaces. The system is restricted in that it is confined to two-dimensional shape identification and non-overlapping objects. It remains to be seen whether the shape descriptors discussed here can be adapted through the use of more flexible matching strategies to allow for partially visible objects. More sophisticated low-level image primitives will be required if 2 1/2-D or 3-D information is required but it is likely that the software architecture described here will facilitate the integration of these capabilities into a coherent and efficient system.

REFERENCES

Ayache, N., Faverjon, B., Boissonnat, J.D., and Bollack, B. 1984. "Manipulation Automatique de Pieces Industrielles en Vrac Planaire", Proceedings of the First Image Symposium, Biarritz, pp.869-875.

Berman, S., Parikh, P., and Lee, C.S.G. 1981, "Computer Recognition of Overlapping Parts using a Single Camera", Proc. of IEEE Computer Society Conference on Pattern Recognition and Image Processing, Dallas, pp.650-655.

Berthod, M. 1982. "Iconic Matching", INRIA-CREST Course on Computer Vision, Rocquencourt, France.

Bonner, S. and Shin, K.G. 1982. "A Comparative Study of Robot Languages", Computer, Vol. 15, No. 12, pp.82-96.

Engleberger, J.F. 1980. "Robotics in Practice: Management and Applications of Industrial Robots".

Freeman, H. 1974. "Computer Processing of Line-Drawing Images", ACM Computing Surveys, Vol. 6, No. 1, pp.57-97.

Goldman, R. 1982. "Design of an Interactive Manipulator Programming Environment", Department of Computer Science, Stanford University, Stanford, STAN-CS-82-955.

Gonzalez, R.C. and Safabakhsh, R. 1982. "Computer Vision Techniques for Industrial Applications and Robot Control", Computer, Vol. 15, No. 12, pp.17-32.

Grossman, D.D. 1977. "Programming of a Computer Controlled Industrial Manipulator by Guiding through the Motions", IBM Research Report RC6393, IBM T.J. Watson Research Centre, Yorktown Heights, N.Y.

Hanson, A.R. and Riseman, E.M. 1978. "Segmentation of Natural Scenes", in "Computer Vision Systems", Hanson, A.R. and Riseman, E.M. (Eds.).

Iversen, W. 1982. "Robots Pick Parts out of a Bin", Electronics, November 30, p.50.

Jain, R. and Haynes, S. 1982. "Imprecision in Computer Vision", Computer, pp.39-48.

Juvin, D. and Dupreyrat, B. 1981. "ANIMA (Analysis of Images): A Quasi Real-Time System", IEEE Computer Socity Conference on Pattern Recognition and Image Processing, pp.358-361.

Juvin, D. and de Cosnac, B. 1984. "ANIMA 2: Un Systeme Generale de Vision Pour la Robotique", Proc. Premier Colloque Image, Biarritz, pp.165-169.

Kelley, R.B., Birk,J.R., Martins,H.A.S., and Tella,R. 1982. "A Robot System which Acquires Cylindrical Workpieces from Bins", IEEE Transactions on Sytems, Man, and Cybernetics, Vol. SMC-12, No. 2, pp.204-213.

Kelly, M.D. 1971. "Edge Detection in Pictures by Computer using Planning", Machine Intelligence, B. Meltzer and D. Michie (Eds.), Vol. 6, pp.397-409.

Latombe, J.C. n.d. "Automatic Synthesis of Robot Programs from CAD Specifications", IMAG, BP 68, 38041 Saint-Martin-d'Heres, CEDEX, France.

Lavin, M.A. and Lieberman, L.I. 1982. "AML/V: An Industrial Machine Vision Programming System", The International Journal of Robotics Research", Vol. 1, No. 3, pp.42-56.

Lee, H.C. and Fu, K.S. 1981. "The GLGS Image Representation and its Application to Preliminary Segmentation and Pre-attentive Visual Search", IEEE Computer Society Conference on Pattern Recognition and Image Processing, pp.256-261.

Levine, M.D. 1980. "Region Analysis using Pyramid Data Structures", in "Structured Computer Vision", Tanimoto, S. and Klinger, A. (Eds.), Academic Press, Inc., New York, pp.57-100.

Li, H.F., Tsang, C.M., and Cheung, Y.S. 1983. "A low-Cost Real-Time Imaging and Processing System", Software and Micorsystems, Vol. 2, No. 5, pp.121-129.

Lieberman, L.I. and Wesely, M.A., "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," IBM Journal of Research and Development, Vol. 21, No. 4, pp.321-333.

Lozano-Perez, T. 1982. "Robot Programming", MIT AI Lab, AI Memo 698.

Martin, W.N. and Aggarwal, J.K. 1978. "Survey - Dynamic Scene Analysis", Computer Graphics and Image Processing, Vol. 7, No. 3, pp. 356-374.

Mujtaba, M. 1982. "Motion Sequencing of Manipulators", Ph.D. Thesis, Stanford University, Report No. STAN-CS-82-917.

Mujtaba, M. and Goldman, R. 1979. "The AL User's Manual", STAN-CS-79-718, Stanford University.

Pau, L.F. 1984. "Approaches to Industrial Image Processing and their Limitations", Electronics and Power, February, pp.135-140.

Paul, R. 1981. "Robot Manipulators: Mathematics, Programming, and Control", MIT Press, Cambridge, Massachusetts, 1981.

Prewitt, J.M.S. 1970. "Object Enhancement and Extraction" in "Picture Processing and Psychopictorics", B. Lipkin and A. rosenfeld (Eds.), Academic Press, pp.75-149.

Roberts, L.G. 1965. "Machine Perception of Three-Dimensional Solids" in "Optical and Electro-Optical Information Processing", J.T. Tippett et al. (Eds.), MIT Press, Cambridge, Massachusetts, pp.159-197.

Rosen, C.A. 1978. "Machine Vision and Robotics: Industrial Requirements", Technical Note No. 174, SRI International.

Shirai, Y. 1978. "Recognition of Real-World Objects using Edge Cue" in "Computer Vision Systems", Hanson, A. and Riseman, F. (Eds.), Academic Press, pp.353-362.

Sze, T-W. and Yang, Y-H. 1982. "Goal Directed Segmentation", IEEE Computer Society Conference on Pattern Recognition and Image Processing, pp.504-510.

Tanimoto, S.L. 1980. "Image Data Structures" in "Structured Computer Vision", Tanimoto, S. and Klinger, A. (Eds.), Academic Press, Inc., New York, pp.31-55.

Tarvin, R.L. 1980. "Considerations for Off-Line Programming of a Heavy Duty Industrial Robot", Proc. 10th. International Symposium on Industrial Robots, Milan, pp.109-116.

Taylor, R.H., Summers, P.D., and Meyer, J.M. 1982. "AML: A Manufacturing Language", The International Journal of Robotics Research, Vol. 1, No. 3, pp.19-41.

Unimation 1979. "User's Guide to VAL", Version 11, 2nd. edition, Unimation Inc.

Wallace, A.M. 1983. "Grey-Scale Image Processing For Industrial Applications", Image and Vision Computing, Vol. 1, No. 4, pp.178-188.

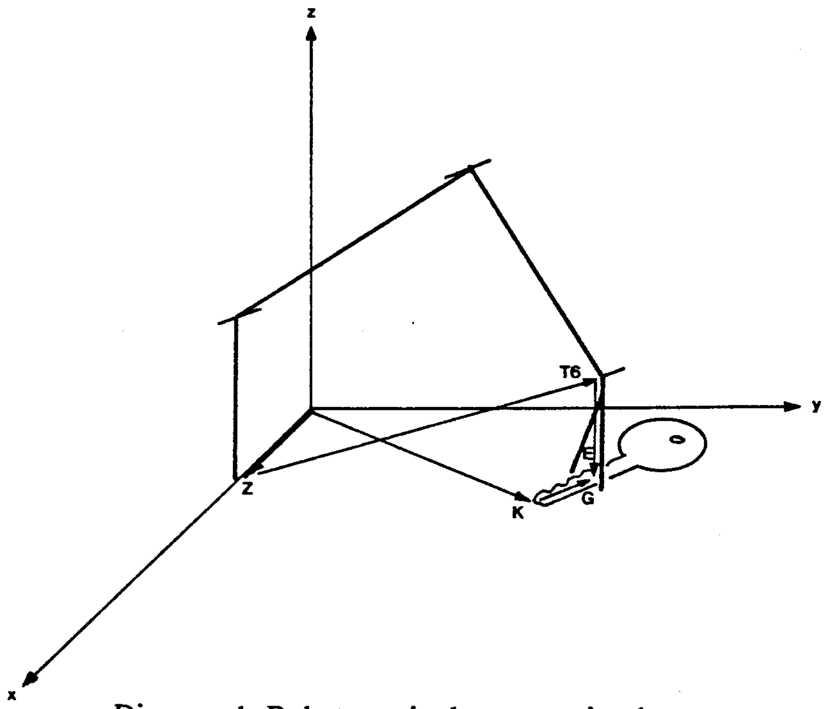
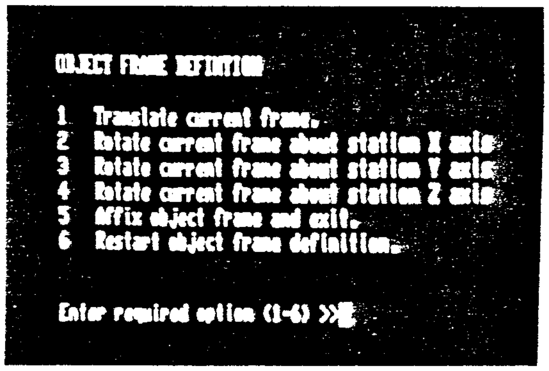


Diagram 1: Robot manipulator grasping key.

Diagram 2: Image of key.



```
OBJECT FRAME DEFINITION

1  Translate current frame.
2  Rotate current frame about station X axis.
3  Rotate current frame about station Y axis.
4  Rotate current frame about station Z axis.
5  Affix object frame and exit.
6  Restart object frame definition.

Enter required option (1-6) >>
```

Diagram 3: Menu for interactive frame definition.

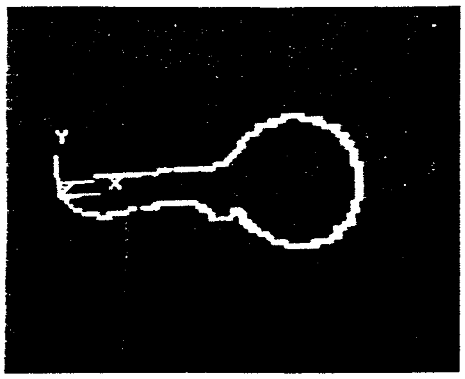


Diagram 4: Segmented key with object frame definition ^K.

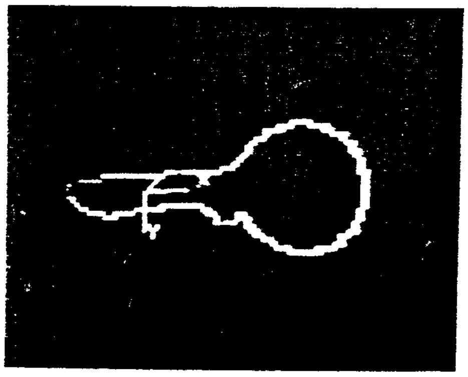


Diagram 5: Segmented key with gripper frame definition ^G.