PROGRAMMING VISION SYSTEMS AND INTEGRATING THEM WITH ROBOTS

G. Gini* and M. Gini**

*Dipartimento di Elettronica Politecnico, Milano, Italy
**Department of Computer Science, University of Minnesota, MI, USA

ABSTRACT

We present LIVIA, a language for writing application programs for industrial vision systems. LIVIA is a Pascal-like language with specific instructions for vision. The objects are identified by the values of their invariant features. Operations on the objects can be done using the elementary set theory. The recognition of objects in a data base and the selection of any of their features is easily programmed.

INTRODUCTION

The importance of providing robots with more sophisticated vision sensors is widely recognized. Visual feedback for manipulation and assembly operations is highly desirable to increase reliability and to reduce the constraints on the positioning of parts. Visual inspection of manufactured products to determine if they meet their standards plays an important role in industrial automation [1], [5], [12].

The availability of industrial vision systems and their integration into the manufacturing process indicate a growing interest by industry in automatic vision.

Flexibility, programmability, and reliability are the key issues for vision systems, although cost and processing time are still important. Portability will become more and more important as soon as software packages become available for various applications.

In our laboratory we have developed a vision machine for industrial applications [9], and we have provided it with a language, LIVIA [4]. In LIVIA the user can write programs to recognize objects or to locate parts.

The main assumptions in LIVIA are that the user in an industrial setting is interested in systems that can be easily programmed for specific applications. On the other hand a single vision system will never be applicable to all kinds of industrial tasks and also be cost effective. We propose a philosophy of design of vision systems that allows programmability at different levels. We provide the user with a simple language, and at the same time we allow the reconfiguration of the whole software for different applications and exigencies.

In the following we discuss the role of vision and of vision programming in industrial settings. Then we present the language LIVIA.

We support our presentation with practical examples of integrated systems. In particular we will refer to the system of the Politecnico of Milan [3]; we consider also the system of the Stanford Artificial Intelligence Laboratory [8] and some commercially available systems [2], [7].

VISION PROGRAMMING FOR INDUSTRIAL APPLICATIONS

The complexity of the vision tasks requires a decomposition of the problem into manageable subunits [15]. The identification of the function of each subunit and of its relationships with the other subunit is not a trivial task. Most of the systems apply a fixed sequence of functions to the images. Each change in the sequence requires a rewriting of a large part of the system.

What is needed is a language in which one can specify the control structures that relate the various functions to be performed on the image as well as a language in which one can specify models of objects. A language and a programming environment specifically designed for industrial vision systems will dramatically reduce the software costs, opening new application areas.

For many aspects we can foresee the same development for vision software we have seen in languages for controlling robots. From simple systems based on "teaching by training", sophisticated programming languages have been developed that make it easier to program robots for complex tasks. Unfortunately not too many languages are available for vision [6].

If we examine the solutions implemented by producers of vision systems for industrial applications we can identify two different approaches:

- The vision system is programmed through menus or special functional keys, allowing the user to select among a predefined set of basic functions. The communication with an external computer is provided to allow integration with additional hardware and software.
- A programming language is provided to let the user write his own application programs. Different solutions have been

selected by different producers ranging from the definition of highly specialized languages for vision to the definition of general purpose languages with few instructions for vision.

We examine in more detail some of those solutions.

The first example of industrial vision systems, the VS-100 of the Machine Intelligence Corporation, is programmable through menus and a light pen. It derives, as most of the systems in industrial use, from the SRI vision machine [1]. Object images are analyzed during the teaching phase to acquire their features and to build a model. The model is a set of features precisely describing the object. In the operation phase the images are analyzed to compare their features with the models.

Unimation [2] has decided to extend its language for robot programming, VAL, by adding a few instructions to communicate with the VS-100. In this case the robot programming language takes advantage of the communication protocol available in the vision system. Similar solutions have been adopted by Brown Boveri for its OMS system [11] and by Control Automation for its V-1000, just to mention a few examples.

The vision system of the Machine Intelligence Corporation has been used in the Stanford Hand-Eye project with the language AL [8], [10]. The system is composed by two Stanford Scheinman arms, two PUMA 500 arms, a Grinnel Graphic system, and a VS-100 vision machine, all interconnected. The software for controlling and programming the whole system is developed and compiled on a PDP 10, and runs on a devoted PDP 11/45 plus some LSI 11's. The availability of the language AL and of the POINTY programming environment [10] allowed the creation of a unified programming system for manipulation and vision. The possibility of defining procedures and macros in AL has been used to define operations with the vision system. Only one general procedure for communication through the parallel interface with the VS-100 is needed to handle the dialogue between the manipulation and the vision activities. In a similar way VAL instructions can be sent from AL to the PUMA's.

We want to point out how in those cases the vision system is essentially considered as an intelligent peripheral of the robot. The vision system executes always the same recognition algorithm.

Moving to the second approach we consider RAIL and AML/V.

RAIL [7], [16] has been designed by Automatix as the programming language for its vision and robot systems. It has a Pascal like form and it is very easy to use.

AML/V [13] is an extension of the AML language developed at the IBM to control robots. AML/V is a set of subroutines, devoted to camera control and image acquisition, binary region analysis, and image arithmetic. AML/V is a highly programmable system which provides direct access to image data. It is an "integrated extension of a good general programming language" and, as such, it is considered by its designers as a very powerful language. In fact in AML/V it is possible to program new operations on images.

In our opinion a compound approach should be used. We do agree with Automatix and other companies on the need of a simple programming language. We do agree also with IBM when they point out the need for a complete programming language. In fact using the present industrial vision systems the user can only write application programs in which the vision processing is based on the same fixed algorithm. Using AML/V other algorithms can be programmed (see for instance the thinning algorithm presented in [13]), even though this process requires a certain expertise.

We see two contraddictory requirements, the first aiming at the ease in writing applications programs, the second aiming at the completeness of the programming system. In the first case the final user can easily interact with a system with fixed capabilities; in the second case the system is much more flexible at the cost of being difficult to use. Both those points are important and deserve the greatest attention.

Our idea is to have different programming levels for different users. We let the system programmer free to modify any of the programs in the vision machine and to connect them together specifying various sequences of operations. The final user writes application programs in a specialized programming language.

We have implemented this methodology in our vision system. We provide programmability at three different levels:

- user level. The final user of the system is not interested in the internal aspects; he wants a simple way to communicate what he wants to do. A programming language, LIVIA, is available for writing application programs. Even in the case in which grey level processing is used, as in [14], the user is concerned only with recognition of objects, without having to know how the processing is done;
- application level. Application dependent packages can be written for specific fields of applications as arc welding, or manipulation of particular types of objects;
- system level. Any of the basic components of the system can be modified either by changing some code or by adding new functions. Since the system is programmed in high level language using well-defined interfaces it is easy to put modules together to make a system.

ARCHITECTURE OF THE VISION SYSTEM

The main task we had in mind in developing our vision system was to develop a flexible system that can be ready for effective factory use in a short time, but that allows the easy incorporation of

extensions.

We have set up a general framework and some tools, that allow us to configure the system, to verify the module interfaces, and to write application programs. We have first developed a collection of modules to form a skeleton system. We have incorporated modules to handle several applications and we are in the process of adding more.

The main use is currently for identification of objects to be assembled by a robot. A large set of characteristics is computed, scenes with many objects are analyzed, objects inside objects are detected. Even upside down parts can be recognized.

From a logical point of view the system is divided into two levels, a low level process and a high level process. The low level process, connected to the camera, performs the basic computations on the image. The high level process executes application programs and communicates with higher level processes. This organization has been designed to take advantage of the multiprocessing capabilities of MODIAC, a multi-microprocessor system specifically designed in Italy for industrial automation.

Each process is composed of modules that the user can assemble. Each module has a well defined interface with a specification of the input to the module, the output from the module and the global variables used. Modules with the same interface can be freely interchanged.

The lower level process can be considered as a vision machine. It includes threshold selection, binarization, run-length coding, sequential extraction of features, recognition, teaching by showing, management of a library of models, and other modules for image preprocessing. Other modules for screen display which are dependent on the specific hardware have to be provided. Modules that depend on specific applications can be inserted here.

The user can interact through a terminal on which a menu of commands is available to start various programs. An operating system allows him to reconfigure the system by adding or changing any of the modules. System level programming takes place mainly at this level.

The communication with the high level process is through a communication handler. In the present implementation both the processes run on the same computer. The high level process has to wait for the answer from the low level process before proceeding. Eventually the two processes will run asynchronously.

The higher level process is mainly used for programming the vision machine (low level process). It includes the compiler for the LIVIA language, the interpreter of the intermediate code, editor and general purpose software.

Application dependent programs can be added. If they require specific modules on the low level system the user is responsible for adding the appropriate routines. A second terminal is available for entering LIVIA programs.

Other processes can communicate with the vision machine both directly by feeding commands through the communication handler or through the high level process by sending commands in LIVIA to be compiled and interpreted or by sending commands already compiled to be interpreted.

The communication with the robot mainly takes place by sending directly commands to the communication handler or, for more complex operations, through LIVIA programs. In many cases the robot asks the vision system directly to take a picture and to return the identity, position, and orientation of the object in view. The responsibility of transforming the coordinates of the vision system into the coordinates of the robot is given to the robot program.

Our vision system is fully portable. This aim has been achieved by using a high level language for the implementation and by using simple, clear interfaces between modules. We have decided not to use any special hardware. The final customer will be able to decide whether to develop specialized hardware to speed up some operations. In this way we can keep our development system independent of any particular application.

We can easily integrate our vision system into an industrial automation system. The final implementation will be on the multi-microprocessor system MODIAC. It will be very important to be able to operate a vision system that interfaces naturally with the system that controls the manufacturing process.

The present implementation is based on the use of a GE TN-2200 solid state TV camera with 128 x 128 pixels and 256 grey levels. The camera is connected through a DMA interface to a DEC MINC-11 computer with 28K words of memory and a LSI 11/23 CPU. The same computer is used to supervise the various robots available at the Robotics Laboratory of the Politecnico of Milan. The robots, the vision system, and a conveyor belt can be controlled to cooperate for assembly ad manipulation tasks. A video terminal DEC VT 105 is used for interaction with the user, and a color monitor ISC 8001 provides a graphic display of the image and the results. The programs are written in OMSI Pascal, except the camera acquisition routine which is in MACRO-11. RT-11 is the standard operating system.

LIVIA: THE USER PROGRAMMING LANGUAGE

We have designed a language, LIVIA, for programming vision tasks. LIVIA is a Pascal-like language with special instructions for vision [4]. We have decided to base most of the operations on the elementary set theory.

LIVIA has two data types, numbers and sets. Numbers can be integer or real. Sets represent groups of objects. Each object is described by a set of feature values. Some features are position dependent, some are position independent. The last ones characterize the objects and constitute

what we call an object model.

The model is usually constructed with a teaching-by-showing process by showing various times the same object in different positions and orientations. Data from CAD data bases could be used to obtain the model. Various libraries of models can be stored on disks. Each library of models can contain any number of elements.

The internal model of the sets is composed by three component parts. The memory of blobs contains all the objects identified in the image with their feature values. The memory of models contains all the models that should be used. The correspondence table is constructed connecting the blobs with the models. The correspondence is generated by a classification algorithm. Each blob is connected to the nearest model. The value of the distance considered acceptable for recognition can be set by the user.

Each time a picture is taken the system constructs a set of the blobs seen by the camera. We call this set TEL. We may construct a set of all the elements that can be found in any of the available libraries of models. We call this set MEM. All the other sets are created from those two sets.

Using the set operations we can intersect the sets TEL and MEM, or find any subset with specific features, as a certain number of holes or a certain range of perimeter.

Operations available on sets include elementary set operations (union, intersection, and set difference), selection of elements with specific features, and extraction of single elements from any set. Those single elements are considered as sets with a single element, so that the same operations can be applied to them.

The operation INTER computes the intersection of two sets, i.e. the set of elements that are members of both the sets. The correspondence table is used to figure out what objects are in both the sets. The operation UNION computes the union of two sets, i.e. the set of elements that are members of either one of the two sets. DIFF computes the set difference, i.e. the set of elements that are members of the first set and not members of the second. Those operations can be freely intermixed in the same instruction.

Operators are available to select objects with specific features. Selections can be applied to any set to construct subsets of elements satisfying the desired properties.

The control structures available in LIVIA are the same as in Pascal (do..while, repeat..until, if..then, if..then..else, go to). I/O is performed exactly as in Pascal. Procedures are not yet available.

There are a few specific instructions for vision operations.

TAKEAPICTURE takes a picture, computes all the features of the objects and construct the set TEL. The previous content of TEL and of any set derived from it are cleared.

TAKEMEM (name) constructs the set MEM inserting into it all the models present in the named library. The previous content of MEM is cleared.

SETAREA (val) sets the minimum area (in pixels) that will be considered by the system. In this way the user can decide to do not consider objects that are too small or that can be little spots.

SETTHRESHOLD (val) sets the threshold used for binarization.

SETDISTANCE (val) sets the distance that is considered acceptable by the classification algorithm.

The function DISTANCE returns the distance of the indicated blob from the corresponding model. NEAREST returns a set constituted by the model nearest to the indicated blob. Using them the user can write different classification algorithms.

The various elements of the set can be selected by using INIT to start at the beginning of the set, and NEXT to go to the next element in the set. The predicate ENDSET tells whether the set has been completely examined. ELEM returns the current element of the set.

Once a particular element of the set has been identified it is possible to extract each one of its feature values by means of extraction operators. The extraction operators are the same as the selectors. When they are used as extractors they return a numeric value.

Some examples of short programs are illustrated.

The first program takes a picture, selects objects with specific features, and recognizes them using a library. The intersection between TEL and MEM contains the recognized objects. We use then two different extractors for the same feature, since we want to compare the value of the blob with the value of the model. AREA returns the area of the object and AREAMD is the area from the corresponding model. NAMEINT indicates the internal identifier of an object and NAMEEXT is the corresponding external name.

```
PROGRAM SELECT;
VAR NUM: NAME, AR, ARMD;
    SET: BB;
BEGIN
    TAKEAPICTURE;
    TAKEMEM ("GROUP#1");
    BB:= TEL INTER MEM;
    INIT (BB);
    NEXT (BB);
    PRINTLN ("RECOGNIZED OBJECTS");
    PRINTLN ("NAMES   AREA OBJECTS",
             "   AREA MODELS");
    WHILE NOT ENDSET (BB) DO BEGIN
        NAME:= NAMEINT (ELEM (BB));
        PRINT (NAMEEXT (NAME));
        AR:= AREA (ELEM (BB));
        ARMD:= AREAMD (ELEM (BB));
        PRINTLN ("       ", AR,
                 "       ", ARMD);
        NEXT (BB);
        END;
END.
```

The next program is used to inspect parts.

Objects not recognized and without holes are to be discarded. Their position is printed so that a robot could be used to remove them. The set UNKNOWN is constituted by the objects that are not recognized and that have no holes. In fact the intersection between TEL and MEM is the set of recognized objects. The difference between TEL and the set of recognized objects constitutes the set of non recognized objects. Among this set the objects that do not have holes are then selected using the selector NHOLES. The coordinates of their centers of gravity are then printed.

```
PROGRAM INSPECTION;
VAR NUM: COORDX, COORDY;
    SET: UNKNOWN;
BEGIN
    TAKEMEM ("KNOWN");
    SETAREA (200);
    TAKEAPICTURE;
    UNKNOWN:= (TEL DIFF (TEL INTER MEM))
              * NHOLES=0;
    INIT (UNKNOWN);
    NEXT (UNKNOWN);
    WHILE NOT ENDSET (UNKNOWN) DO BEGIN
        COORDX:= XBAR (ELEM (UNKNOWN));
        COORDY:= YBAR (ELEM (UNKNOWN));
        PRINTLN ("DEFECTIVE PART IN ",
                  COORDX, " ",COORDY);
        NEXT (UNKNOWN);
        END;
END.
```

The last program shows the use of the functions DISTANCE and NEAREST. The set OBJECTS is constituted by the objects seen by the camera, that have at least one hole, ad an area greater than 200 pixels. The function NEAREST returns the model closest to the indicated element and DISTANCE returns the value of the distance. The program print for each element seen by the camera the name of the closest model and the value of the distance. In this way it is possible to program different recognition strategies.

```
PROGRAM RECOGNIZE;
VAR NUM: NAME, DIST;
    SET: OBJ;
BEGIN
    TAKEMEM("PARTS");
    TAKEAPICTURE;
    OBJ:= (TEL * NHOLES>=1) * AREA>200;
    INIT(OBJ);
    NEXT(OBJ);
    PRINTLN("NEAREST OBJECT     DISTANCE");
    WHILE NOT ENDSET(OBJ) DO BEGIN
        NAME:= NAMEINT(NEAREST(ELEM(OBJ)));
        DIST:= DISTANCE(ELEM(OBJ));
        PRINTLN(NAMEEXT(NAME),"   ", DIST);
        NEXT(OBJ);
        END;
END.
```

CONCLUSIONS

We have presented LIVIA, a high level language for programming industrial vision systems. We have given examples of use.

This work was partially supported by the Progetto Finalizzato per l' Informatica, Sottoprogetto P3, Obiettivo MODIAC of the National Council of Research of Italy.

REFERENCES

1. Agin, G. J., Computer vision systems for industrial inspection and assembly, Computer Magaz., pp 21-31, May 1980.
2. Carlisle, B., et al, PUMA/VS-100 robot vision system, in A. Pugh (ed), Robot Vision, IFS (Publications) Ltd. and Springer-Verlag, 1983.
3. Cassinis, R., Mezzalira, L., A multimicroprocessor system for the control of an industrial robot, Proc. 7th International Symposium on Industrial Robots, Tokyo, Japan, 1977.
4. Cividini, G., Gini, M., Villa, G., Programming a vision system, Proc. 13th International Symposium on Industrial Robots, Chicago, Ill, 1983.
5. Dodd, G. G., Rossol, L., Computer vision and sensor based robots, Plenum Press, New York, 1979.
6. Duff, M.J.B., Levialdi, S. (Eds), Languages and architectures for Image Processing, Academic Press, 1981.
7. Franklin, J. W., VanderBrug, G. J., Programming vision and robotics systems with RAIL, in K. Rathmill (ed), Robotic Assembly, IFS (Publications) Ltd. and Springer-Verlag, 1985.
8. Gini, G., Gini, M., The integration of manipulation and vision for assembly and quality control, International Journal of Production Research, Vol 21, N 2, 1983, pp 279-292.
9. Gini, G., Gini, M., A Software Laboratory for Visual Inspection and Recognition, Pattern Recognition, Vol 18, N 1, pp 43-51, 1985.
10. Gini, G., Gini, M., Dealing with world model based programs, ACM TOPLAS, Vol 7, N 2, 1985, pp 334-347.
11. Hewkin, P. F., Fuchs, H. J., OMS-vision system, in A. Pugh (ed), Robot Vision, IFS (Publications) Ltd. and Springer-Verlag, 1983.
12. Kruger, R. P., Thompson, W. B., A technical and economic assessment of computer vision for industrial inspection and robotic assembly, Proc. IEEE, Vol. 69, pp 1524-1538, December 1981.
13. Lavin, M. A., Lieberman L. I., AML/V: An industrial machine vision programming system, The International Journal of Robotics Research, Vol 1, N. 3, Fall 1982.
14. Makhlin, A. G., Tinsdale, G. E., Westinghouse grey scale vision system for real time control and inspection, in A. Pugh (ed), Robot Vision, IFS (Publications) Ltd. and Springer-Verlag, 1983.
15. Rosenfeld, A., Kak, A., Digital Picture Processing, Academic Press, New York, 1976.
16. Villers, P., Present industrial use of vision sensors for robot guidance, in A. Pugh (ed), Robot Vision, IFS (Publications) Ltd. and Springer-Verlag, 1983.