'87 KACC 1987.10.16~17

A VISION-BASED ROBOTIC ASSEMBLY SYSTEM

*Sang-Rok Oh**, *Joonhong Lim***, *Il Hong Suh*+,
*You-Shik Shin**, *Byoung Il Rhee**, *and Zeungnam Bien**

* Dept. of EE, KAIST, Seoul, Korea
** Dept. of Avionics, Hankuk Aviation College
+ Dept. of EE, Hanyang University

Abstract

In this paper, design and development experiences of a vision based robotic assembly system for electronic components are described.

Specifically, the overall system consists of the following three subsystems each of which employs a 16 bit microprocessor MC 68000 : supervisory controller, real-time vision system, and servo system. The three microprocessors are interconnected using the time shared common memory bus structure with hardwired bus arbitration scheme and operated as a master-slave type in which each slave is functionally fixed in view of software. With this system architecture, the followings are developed and implemented in this research;

(i) the system programming language, called 'CLRC', for man-machine interface including the robot motion and vision primitives,

(ii) real-time vision system using hardwired chain coder,

(iii) the high-precision servo techniques for high speed dc motors and high speed stepping motors.

The proposed control system were implemented and tested in real-time successfully.

I. INTRODUCTION

Automations of various assembly works of electronic components have been attempted to reduce manufacturing cost as well as to increase productivity. One of the important design goals of the automation is to integrate cost-effectively the major system functions such as visual inspection and guidance of components, control of robot manipulators, sequential control of mechanical fixturing devices, and material transport machines. To be cost-effective, the system should be reliable, capable of fast information-processing, and flexible enough to easily adapt to various complex machanical fixturing devices for the component types to be assembled.

In this paper, design experiences of a project on building a vision-based robotic assembly system, which consists of three 16 bit microprocessor-based subsystems, are reported. A special feature is that the system is designed to be fully automatic in the sense that the manual operation is minimized via automatic measurements and automatic corrections for the orientation and size of the components. For this, the followings are performed in this project;

(1) The system programming language, which we call 'CLRC' (C Library for Robot Control), was designed and implemented for man-machine interface of the system in such a way that the user can easily operate the robot manipulator and the visual pattern recognition system with the high level language "C". In particular, the robot motion primitives and the vision primitives are defined and realized as a "C" library.

(2) To improve the visual sensing capability, the chain coding algorithm was modified and implemented as hardware. The processing time was greatly reduced as compared to the software algorithm so that the hardware chain coder may be adequate for realtime applications. Furthermore, using this chain coder, some algorithms were developed in which the vision system can learn the features of objects and recognize objects from the learned features.

(3) The high-precision servo techniques of the actuators were developed. A time optimal controller for DC motor, which is robust with respect to disturbances, was designed. A chopper type controller for stepping motor was designed to enhance the power efficiency.

To verify the proposed design techniques, an experimental control system is developed and applied for the assembly process of small-size electronic components.

II. OVERALL SYSTEM CONFIGURATION

The assembly control system for electronic components is essentially a concurrently multi-functional system. Thus, the system should contain multiprocessors, where one central processor plays the role of the master and the other processors are the slaves. Our overall control system configuration, a master-slave type, is shown in Fig.2.1. Each slave is a functionally dedicated system in view of software and the time shared/common bus structure with hardwired bus arbitration scheme is utilized[1]. In the system, three 16 bit microprocessors (MC 68000) interfaced to global bus

are used for the system master, called a supervisory controller, the real time vision system, and the servo controller.

Since the design of the servo controller was presented at another session, we are going to put emphasis on describing the system programming language and the real time vision system.

III. ROBOT PROGRAMMING LANGUAGES

The robot programming langauge, called "CLRC", is designed for man-machine interface of the system so that the user can easily operate the manipulator and the visual pattern recognition system. It is a compiler type robot language using the high-level language "C" as a base language. In CLRC, the data type 'frame' is defined and the coordinate transformation is provided so that the robot motion is specified and controlled in Cartesian coordinate system. The PTP (point-to-point) motions and CP (continuous path) motions such as straight line, circle, and parabola, are realized and implemented for a SCARA-type robot. Some primitives are also provided to control the operation of the visual pattern recognition system which will be explained in the next section. The robot motion primitives and the vision primitives are realized as a "C" library. The CLRC language may be classified as a structured-programming level language in that some task-oriented programmings are possible.

3.1. CLRC Operation

Fig.3.1. shows the structure and use the CLRC. The user writes an application program in C-language and compile it with C-complier. The object code is linked with the standard C library and CLRC to generate the execution code. The execution code is down-loaded to the supervisory computer in which the application program is executed.

The CLRC operation is based on the following five operation modes:

1) Home Mode

In this mode, the robot system is initialzed. When the power is applied to the robot system, each joint of the robot moves slowly to its mechanical home switch. Then the robot moves to its logical home, which serves as the reference point.

2) Teach Mode

In this mode, the user can teach some interesting points in Cartesian coordinate system by moving the robot from the terminal and stores them. The type of the data to be taught is 'frame', which is essentially a 4 by 4 homogeneous transformation matrix. The stored data are used to plan and control the robot motions.

3) Execution Mode

In this mode, the robot is controlled to execute the given task. The necessary data are provided from either the Teach Mode or the external sensors such as the vision system. The PTP and CP motions are available. To realize CP motions, the Taylor's algorithm for straight line is used [2].

4) Hand Mode

In this mode, the robot hand is controlled. The robot hand can be moved up and down. It can be opened and closed to grasp or release object.

5) Vision System Control Mode

In this mode, the user can operate the visual pattern recognition system. The user can teach the objects to be manipulated and the vision system learn features of the object. The learned features are used to recognize and identify the objects. The vision system transfer the positions and orientations of the recognized objects to the robot control system.

3.2. The Language Primitives

The CLRC laguage primitives are consisted of base language C and the additional robot primirtives. The robot primitives concerns the robot motion, the robot direct/inverse kinematics, the vision system, and the softwares for overall system operations. A new data type 'frame' is defined by a 4 by 4 homogeneous trasformation matrix. The language primitives for each operation mode are as follows;

1) Home Mode

home()

:The robot locaton(position and orientation) is initialized.

2) Teach Mode

teach(loc)

:The robot moves by the manual operation. Then, the value of 'loc', which is a 'frame' to represent the location of the robot, is stored in the memory.

3) Execution Mode

move(loc1,loc2,speed)

:The variable 'speed' is declared as an integer between 1 and 100. The robot moves from 'loc1' to 'loc2' by PTP motion. The joint variables corresponding to 'loc1' and 'loc2' are obtained by solving the inverse kinematics. The given value of 'speed' determines the robot speed in Cartesian space.

straight(loc1,loc2,speed)

:The robot moves from 'loc1' to 'loc2' along the straight line with the given speed.

circle(loc1,loc2,loc3,speed)

:The robot moves along the circle determined by 'loc1', 'loc2', and 'loc3'. The direction is from 'loc1' via 'loc2' and loc3' to 'loc1'.

arc(loc1,loc2,loc3,speed)

:The robot moves along the arc of the circle deter-

mined by 'loc1', 'loc2', and 'loc3'. The direction is from 'loc1' via 'loc2' to 'loc3'.

parabola(loc1,loc2,loc3,speed)

:The robot moves along the parabola determined by 'loc1', 'loc2', and 'loc3'. The direction is from 'loc1' via 'loc2' to 'loc3'.

rectangle(loc1,loc2,loc3,loc4,speed)

:The robot moves along the rectangle determined by 'loc1', 'loc2', 'loc3', and 'loc4'. The direction is from 'loc1' via 'loc2', 'loc3,' and 'loc4' to 'loc1'. The motion of each side of the rectangle is actually the straight line motion.

4) Hand Mode

up()

:The robot hand moves up.

down()

:The robot hand moves down.

open()

:The robot hand opens.

close()

:The robot hand closes.

rotate(degree,dir)

:The robot hand rotates by the angle given in 'degree'. The direction is counterclockwise if 'dir' is 1 and clockwise if -1.

absrotate(angle)

:The robot hand rotates to the orientation given in 'angle', which is the angle from the base coordinate. The value of the variable 'angle' is between 0 and 360 degrees.

5) Vision System Control Mode

V_set(mode,level)

:The thresholding level is determined to convert the gray image to the binary image. If 'mode' is 1, the thresholding process is automatic. If it is 0, the theresholing is manual. The value of 'level' is an integer between 0 and 255.

V_cal(rx,ry)

:The camera is calibrated using the least square method. The parameters 'rx' and 'ry' are the known positions in terms of the robot reference coordinate frame.

V_add(name)

:When the user teach a new objet to the vision system, the features of the object are added in the memory. The name of the object is assigned as 'name'.

V_del(name)

:The features of the object, whose name is 'name', is deleted from the memory.

V_rec(name,rx,ry,rth)

:The vision system recognize the object given in 'name' by the use of the learned features. Then, it gives the position and orientation of the object to 'rx', 'ry', and 'rth'.

V_col(p)

:The vision system determines the corner points of the objects. These points are later used to plan a collision-free path.

6) Others

make_coord(obj)

:'obj' is declared as a 'frame'. The object coordinate frame is assigned to the variable 'obj' in terms of the base coordinate frame.

coord(obj)

:'obj' is declared as a 'frame'. The reference coordinate frame of the data after this primitive becomes the value given in 'obj'.

tool(tool1)

:"tool1' is declared as a 'frame'. The tool of the statements followed by this primitive becomes 'tool1'. The default tool is the robot hand.

IV. REAL-TIME VISION SYSTEM

The visual image processing system for the automatic assembly work should have the characteristics of fast processing capability together with high reliability. This implies that the vision system should be insensitive to various noises due to the quantization and ill-defined light conditions, and must be designed by using cost-effective hardwares. To meet such requirements, it is reported that a new type of frame grabber is designed by employing the TMS4161 dual ported dynamic RAM [3] together with on-line processor including LUT(look-up table) and ALU(arithmatic logic unit). In particular, a hardwired chain coder is also designed to extract the objects from background in real time. Furthermore, image processing softwares are designed by utilizing the obtained chain code to extract some features such as area, perimeter, orientation, and size of the object.

The overall structure and the basic operation sequence of the designed real-time vision system are shown in Fig.4.1 and Fig.4.2, respectively.

4.1. A/D, D/A Module

In this module, the composite video signal from camera is converted to digital image signal with 256 gray level, the internal sync signal is generated by separating a sync

signal from composite video signal. Also the image informations in memory, A/D converted image, and some graphic data are overlaid and converted to analog signal to display those at monitor.

Up to 8 cameras can be attached in this module and 3 kinds of syncronous modes can be selected to make composite video signal, to control graphic display controller, and to make system internal sync signal.

4.2. Video-rate Frame Grabber

To discretize a video frame by the 256 x 256 resolution within a 1/60 second frame-time, we must convert the horizontal analog video signal to the 8 bit digital signal with the sampling time of 156 *nsec*. In this case, the sampling time is shorter than the response and/or access-time of a typical dynamic RAM and static RAM. This implies that conventional memories may not be suitable for the video-rate frame grabber. To resolve the problem, a dual ported dynamic RAM, called as TMS4161, is employed, in which 256 shift registers with fast access-time are converted to the conventional 64K bit dynamic RAM in a parallel way. Thus, in TMS4161, the digitized visual image data are firstly stored in the serially-connected 256 shift registers, and then 256 bit image data stored in the 256 shift registers are transferred to the dynamic RAM within 500 *nsec*. It is remarked that only 500 *nsec* is required to use the dynamic memory to store a line data per every 63.5 μsec. This enables the CPU to use the video memory for other purposes.

To resolve some design difficulties such as refreshment, addressing requirement of CPU and line generator, 8203 DRAM controller and specially designed multiplexer and arbiter are utilized as shown in Fig. 4.3.

4.3. LUT & ALU Module

This module has two LUT(Look-Up Table) and one ALU (Arithmatic Logic Unit). Gray level image can be modified by selecting a gray level function of LUT such as binarization, inversion, and etc. The digital image signals output from two LUTs are logically operated in the ALU of which operations includes ADD, SUBTRACT, AND, OR, and NOT. With this module, high level image processing techniques can be implemented in a real-time. The structure of this module is shown in Fig.4.4.

4.4. Chain Coder Module

One of the major issues of the industrial vision system is lengthy processing time due to software implementation and bulk data for image coding. To reduce the processing time of compression of the image data, software implementation should be replaced by hardware implementation. For this, hardwired chain coder is designed and implemented in this module. Specifically, chain coding algorithm[4] is modified to be suitable for hardware implementation based on the state of neighbor-value of boundary pixel and tracking mode. In Fig.4.5, 3×3 window and defined neighbor values are shown. With this window, we show a procedure to obtain neighbor-value in Fig.4.6. And also in Fig.4.7, we show a table generation procedure to generate chain code of next step using neighbor-value and previous step chain code. The hardwired chain coder is easily implemented by utilizing Look-Up Table which contains above two procedure tables. This concept is shown in Fig.4.8 and the structure of the proposed chain coder is also shown in Fig.4.9.

4.5 Software Algorithms

Software algorithms are consisted as follows :

(i) Communication routine takes charge of communication between the robot controller and the vision system via common memory.

(ii) System initialization routine initializes the vision bus, chain coder, and LUT & ALU module.

(iii) Camera calibration routine transforms the coordinate value in image plane into the coordinate value in real world, and vice versa.

(iv) Major vision algorithms are designed for chain coding, feature extraction, and learning and recognition. The flow charts are shown in Figs.4.10, 4.11, and 4.12.

V. CONCLUDING REMARKS

In this paper, a design method of an automatic assembly control system for electronic components is described. The system consists of the supervisory controller, real time vision system, display and servo system. It is noted that the proposed system can be called automatic in the sense that the system can perform the functions of automatic measurements and corrections of object orientation and component size with the aid of the novel type of the vision system. These techniques may be applicable to the assembly process such as die-bonding process for semiconductor, die-attachment devices, and assembly process of SMD(Surface Mounting Devices).

ACKNOWLEDGEMENTS

This work was carried out for two years as a project sponsored by the Ministry of Science and Technology of R.O.K. Much of the work was done as a group effort by the members of Control System Laboratory, Department of Electrical Engineering, KAIST.

REFERENCES

[1] P. H. Enslow Jr., "Multiprocessor Organization - A Survey", *Computing Surveys*, vol.9, No.1, March, 1977.

[2] R. H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories", *IBM Journal of Research and Design*, vol.23, No.4, July, 1979.

[3] R. Pinkam, M. Novak, and K. Guttag, "Video RAM excels at fast graphics", *Electronic Design*, August, 1983.

[4] H. Freeman, "Computer Processing of Line Drawing Images", *Computing Surveys*, vol.6, No.1, March, 1974.

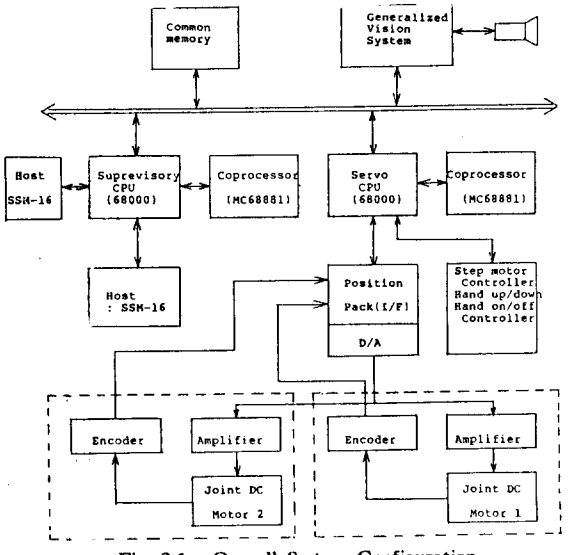

Fig. 2.1. Overall System Configuration

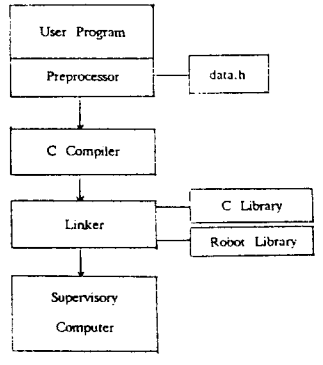

Fig. 3.1. CLRC Structure

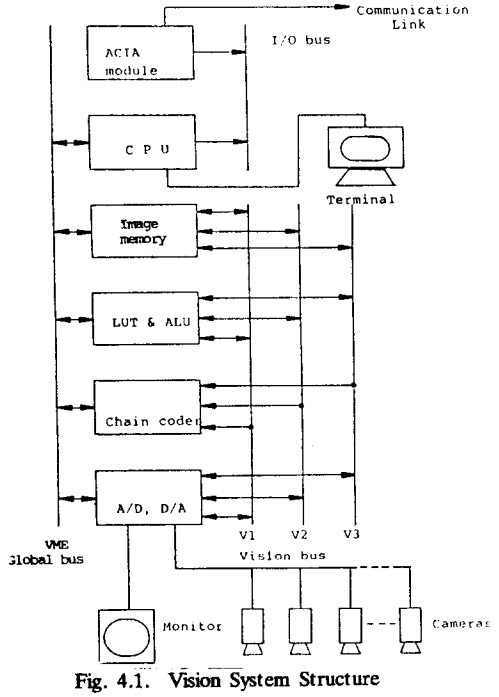

Fig. 4.1. Vision System Structure

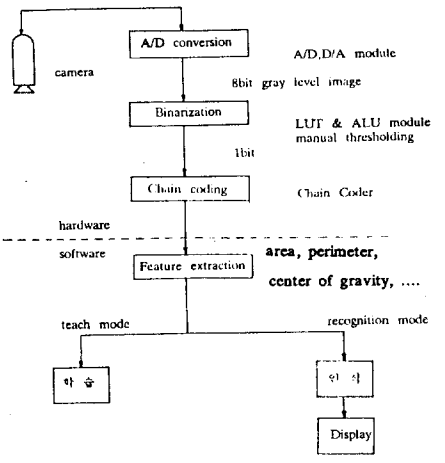

Fig. 4.2. Basic Operation of Vision System

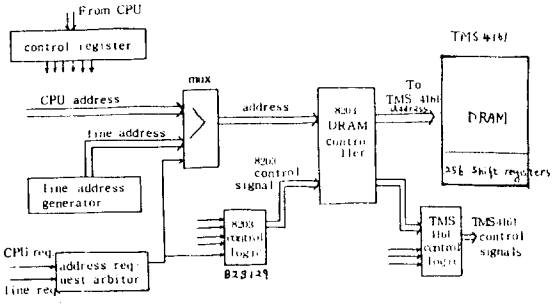

Fig. 4.3. Video-RAM Access Arbiter

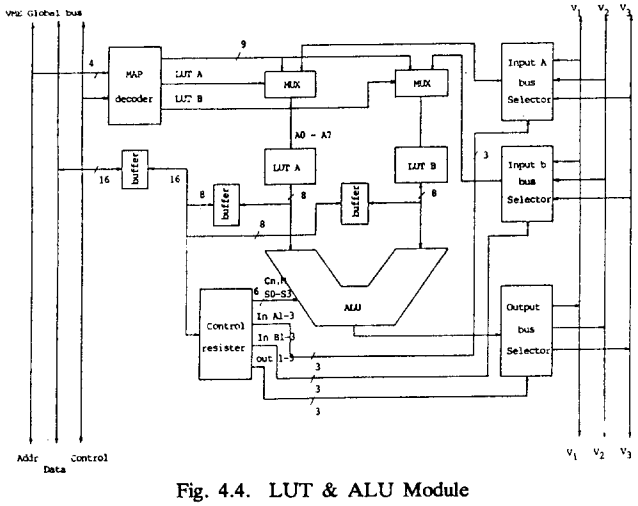

Fig. 4.4. LUT & ALU Module

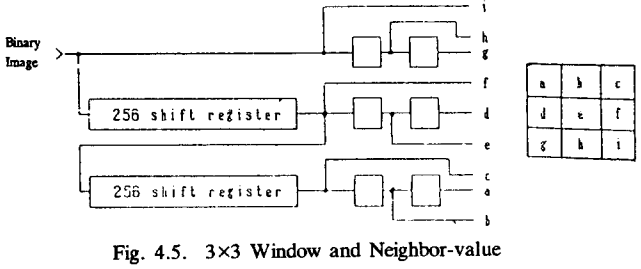

Fig. 4.5. 3×3 Window and Neighbor-value

```
procedure  neighbor−value  (a − i)

{ a − i : pixels in the 3 X 3 window
  if (e == 0) { neighbor−value = 0; }

else{ neighbor−value = i * 2^7 + h * s^6 + g * 2^5
                     + d * s^4 + a * 2^3
                     + b * 2^2 + c * 2^1 + f ;
      if (neighbor−value == FFH ) neighbor−value = 0 ; } }
```

Fig. 4.6. Procedure of Obtaining Neighbor-value

```
procedure chain table ( previous chain code , neighbor−value , mode)

{  if ( ( mode  ==  CCW tracking and RR search ) and
        ( start point ) ) { next chain = RR search
neighbor-value, previous chain code); }

else { next chain = RL search
       (neighbor-value, previous chain code) } }
```

Fig. 4.7. Procedure of Table Generation

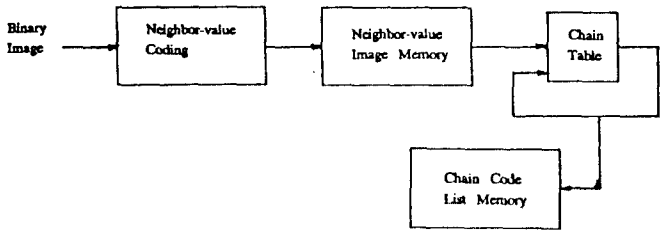

Fig. 4.8. Conceptual Block Diagram of Chain Coder

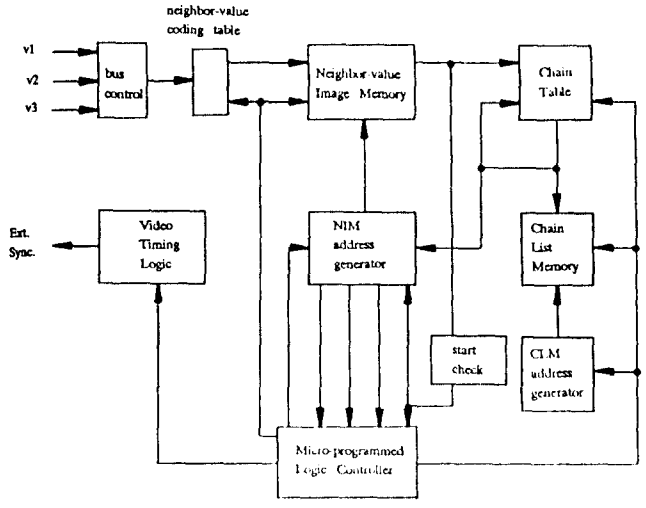

Fig. 4.9. Structure of Chain Coder

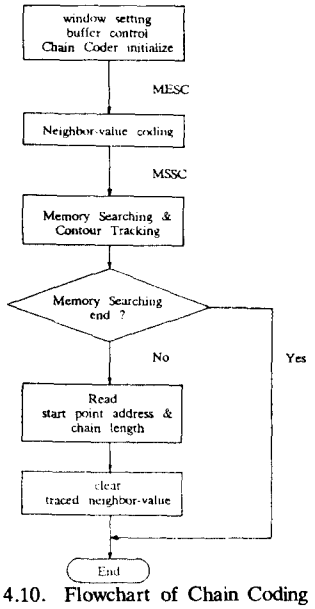

Fig. 4.10. Flowchart of Chain Coding

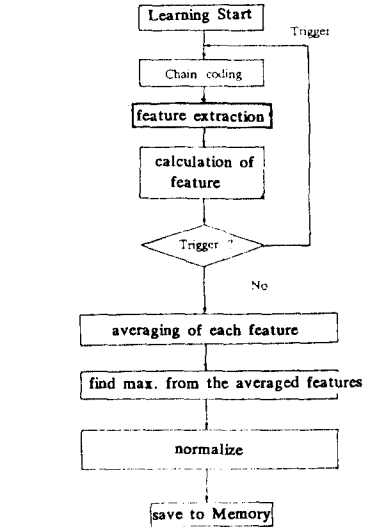

Fig. 4.11. Flowchart of Learning Process

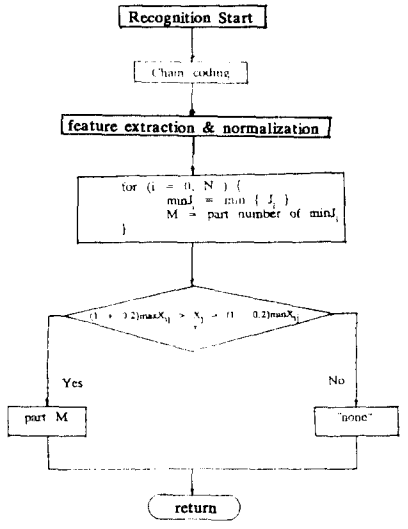

Fig. 4.12. Flowchart of Recognition Process