'89 KACC 1989. 10. 27～28

A WORLD MODEL BASED OFF-LINE ROBOT PROGRAMMING SYSTEM

J.H. Ko, J.H. Park, M.J. Chung

Dept. of Electrical Engineering, KAIST
P.O. Box 150, Cheongryang
Seoul, Korea

In this paper, a programming system for robot-based manufacturing cell which can control and simulate manufacturing devices as well as robots in workcell is proposed and developed. The system is based on world model, and modern textual and object-level robot programming language and interactive graphic world modeler are used to construct and exploit world model. Graphic simulation is used as an efficient and easy to use debugging or verifying tool for user written robot programs. Machine dependency is minimized by adopting the hierarchical control structure and by assuming all the workcell components as virtual ones.

1. Introduction

Industrial robots generally work with other pieces of equipments such as part feeders, sensors, and conveyors ,and they together constitute manufacturing workcells. In these robot-based workcells, the main role of robots are serving tools to production machines or doing the production itself.

The robot-based workcell has some characteristics: 1) Most of tasks in the cell are performed in cooperation of various kinds of cell activities[1]. 2) Relatively easy to adapt to the new manufacturing process. This versatility of robots is well suited for small production quantity and short product life cycle[2]. 3) The programming can be done in two distinct phases: planning phase and run time phase. Workcell layout, programming, and debugging are accomplished in the off-line planning phase, but actual motion control is done in the run time on-line phase[3]. 4) The classes of users are diverse from the sophisticated to shop floor level operators. The sophisticated users want a complex and expressive programming system, but the shop floor operators want a simple and easy-to-use system[2].

From these characteristics of the robot-based workcell, a robot programming system is required to have the following features: 1) Supervise all activities in the cell as a whole; 2) Provide off-line programming facilities such as high level programming languages and simulation system to cut down cost and to reduce production downtime during reprogramming; 3) Be flexible to adapt to the dynamically varying environments and applicable to diverse tasks without serious change of equipment; 4) Have abundant expressions and the ability to abstract the workcell for different classes of users and to provide increased user-friendliness; and 5) Control the cell equipments in real time.

Before we describe a new programming system, we will briefly review the existing programming systems. 1) In robot teaching, most of programming systems use textual languages complemented with leadthrough. Leadthrough has the advantage in teaching specific desired positions in the workspace. On the other hand, textual language can provide complex calculations and utilize sensory informations, and communicate with other computer-based systems[4]. 2) For off-line programming, the information about the cell must be introduced into the programming system through a modeling system and planned tasks must be verified through an appropriate simulation system. However, few of the systems use these facilities in programming[5]. 3) Almost all systems are designed for the specific robot types and cell configuration. In these cases, it is almost impossible to change the constituents or configuration of the cell[6]. 4) Most of the programming systems are suitable for a single robot system. The cooperation with other robots and computer-controlled devices is performed by serializing the actions of each piece of equipment along a time axis. With this scheme, only loosely coupled tasks can be expressed. For closely coupled tasks, more powerful communication means are required[7].

As shown in the above descriptions, the previously develped programming systems have some drawbacks especially in their programmability and flexibility. With these limitations in mind, we propose a new programming system which has the following properties:
1) The programming system uses a *world model* both for increasing programmability and for providing communication channels between programming modules. The world model includes not only geometric information but also various properties of every cell component in structured way. Since the world model is a central information database, the world model is shared by each programming module, such as a geometric modeling system (GMS), a language interpreter, a graphic simulator, and a source-level debugger.
2) To provide increased adaptability, the programming system adopts a *hierarchical control structure*[8][9] in which machine dependent parts of the system are localized and tasks are carried out by multiple dedicated processors in parallel.
3) For increased programmability, it allows not only manipulator-level programming but also *object-level programming*. Programmers can easily express various classes of task from simple to complex one by mixing the commands of these two different levels.
4) Since the workcell components and their properties are descibed symbolically, more *portable, machine independent*[10], and self-documented programs can be written.

This paper consists of six chapters. In Section 2, we describe the proposed system architecture, the world model. Section 3 describes the GMS in detail, and Sections 4 and 5 present language descriptions and a simple example using the language with the simulation result.

2. System Architecture

(1) Hierarchical Control Structure

The control structure of the programming system is divided into cell control system, workstation control system, and device control system according to the level of abstraction as shown in Fig. 1.

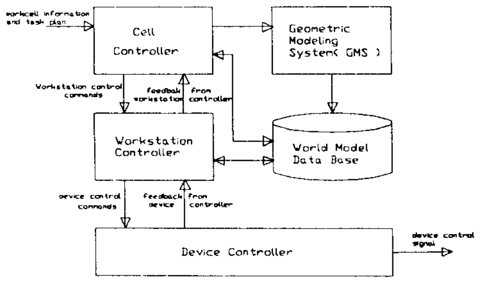
workcell information and task plan
Cell Controller
Geometric Modeling System(GMS)
Workstation control commands
feedback from workstation controller
Workstation Controller
World Model Data Base
device control commands
feedback from device controller
Device Controller
device control signal

Fig. 1. Control system structure

Cell control system: User interfaces such as layout, programming, and debugging are accomplished in this level. As a user lays out the workcell using GMS, the geometric information of the cell is stored in the world model for later use in programming and simulation. Once the layout is done, the user can program using a textual language based on the geometric information. The program may be verified or debugged on the graphic display using a simulator and a source code level debugger in off-line. If the verification is satisfactory, an interpreter translates object-oriented commands (e.g. "MOVE object TO goal WITH robot;") into device-oriented commands ("MOVE robot TO object.grasp_point; GRASP; MOVE robot TO goal; RELEASE;") and then, the device-oriented commands are issued to the lower level controllers of workstation. The feedback information from the workstation control systems is gathered also in this level to update the world model or to change the flow of operations according to feedback information.

Workstation control system: This level of the control system is responsible for supervising various devices and reporting feedback information from the device controllers to the cell control system. During the workstation control, some calculations(e.g. path interpolation or kinematics solution) or conversion of command codes into device code is done.

Device control system: This level of the control system is responsible for controlling physical devices according to the commands from a workstation controller and sending back the status of physical devices to the workstation controller. If a device has its own processor, it is controlled by high level commands from a workstation controller, and if a device has no processor, it is directly controlled by low level control signals from the workstation controller.

In the cell control level, the access and control of the workcell equipment is accomplished via virtual equipment which is described in the world model. And a task in this level is executed with other tasks in time sharing basis. But tasks in the workstation control level are executed in real time. Therefore, there is inevitable timing discrepancy between these two levels. To overcome the timing discrepancy we adopt a concept of *periodic interrupt* and *command buffers* as in RCCL[11]. That is, the workstation controllers respond to the buffered requests from the virtual equipment in every T seconds. If task commands do not require any timing synchronization with other activities, the commands are issued through the command queues, and the commands which require synchronization are directly issued command channels. For virtual robots and machines which have their own processors, commands may be preplanned and issued in advance through command queues, provided the commands do not require any interaction with other equipment. On the other hand, the virtual devices are linked via the command channels, since they require the commands to be processed in the specified time slot.

(2) World Model

A world model is defined as the internal computer representation of a manufacturing workcell. Therefore, the world model is responsible for representing current states of the real manufacturing workcell as exactly as possible and having all the information about the workcell components to assist the modules which require the current state of the workcell. However, a world model has been used only for structured position description. In the proposed system, a world model is extensively used not only for structured geometric descriptions of a workcell but also for tracing assembly status when the interpreter decodes object-level commands and also for integrating the function modules of the system. Since every function module is integrated through the world model, the world model is designed to guarantee mutual exclusion between competing modules and to reflect the modification of the workcell during task execution dynamically.

To model a robot-based workcell, we categorize the cell elements into five groups : robot, machine, device, object, and communication port. The world model consists of linked list structures of these elements. The attributes of each element are shown in Fig. 2.

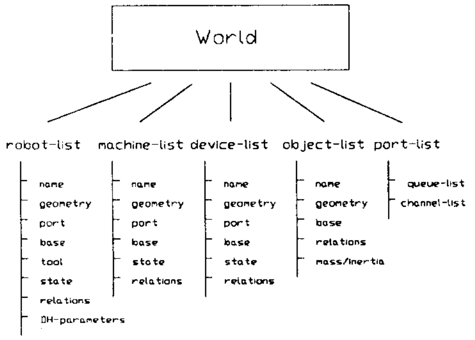

Fig. 2. Structure of world model

Geometric description is the representation of the shape of an object, which is represented as a set of vertex point list, edge list, and surface list.

Base is a homogeneous transformation matrix, which represents relative transformation with respect to its parent object.

Status is current state of a component. It is updated automatically to represent correct values.

Port is the communication buffer which links a virtual equipment with its actual equipment. For example, as a communication port, a robot and a device has a queue and a channel, respectively.

3. Geometric Modeling System and Simulator

(1) Geometric Modeling System

The GMS models real world objects geometrically and connects them to form a world model. This world modeling process can be divided into two phases: object modeling phase and workcell layout phase.

In object modeling phase, objects are constructed geometrically out of primitive objects through Boolean operations. The resultant object contains its sub-objects and their relationships. Geometric information of primitive objects are stored as boundary components such as surrounding surfaces, edges and vertices. These three groups of the geometric information are stored separately but they are inter-related through pointers, so there exists minimal redundancy while the completeness of geometric information is maintained. The information of boundary surfaces of an object is stored in the surface group and each boundary surface has pointers to boundary edges. Similarly, boundary edge information is stored in the edge group and each boundary edge has pointers to two end vertices. The vertex group has the coordinate values of each vertex point in real world, which are represented in Cartesian coordinate system. Thus, a graphic image can be reconstructed without ambiguities in either wire-framed mode or surface-shaded mode.

The integration of primitives or simple objects for complex ones is done by determining the mutual relationship of objects and connecting them. The relationship between primitive objects is expressed by a homogeneous transformation matrix which defines the relative positional and rotational relationship between local coordinate frames of these objects. This relationship may be varied or fixed, depending on whether the object is a moving object such as a robot or stationary object such as a table. In this way, robots or other cell components are treated in the same manner. A robot link is constructed by integrating primitive objects until a realistic model is acquired, and a robot itself is constructed by integrating such links. This construction procedure of objects is done interactively on the graphic terminal, so it provides substantially fast and errorless geometric modeling of robots and other workcell equipments.

In workcell layout phase, previously modeled and stored objects are retrieved from the database and placed at the desired position with proper orientation and initial posture in workcell. Since this phase is done also graphically, errorless and efficient layout can be accomplished in relatively short time.

(2) Graphic Simulator

The graphic simulator shows the motion of the cell components graphically in user-defined viewing environment. This viewing environment includes variable view direction, zooming ratio and multi-view mode. In multi-view mode, the same object is displayed in four different view directions, so its 3-dimensional shape can be easily understood.

The motion of robots is controlled in either passive or active mode. In passive mode, the graphic simulator displays the current state of a world model faithfully as it is changed by other modules which access and update the world model. In active mode, the graphic simulator changes the geometric information contained in the world model through a teach pendant emulator or a pre-planned robot trajectory. Using the teach pendant emulator, user can generate legitimate paths for any moving workcell constituents on the graphic monitor. The pre-planned trajectory, which is generated by an external trajectory planning system, can be graphically simulated and checked for collision or joint limit excess and to improve the trajectory by eliminating or reducing redundant motions of robot.

4. Language Description

There are two basic approaches to the design of textual programming language: One is the extension of an existing language by a library of functions (e.g. RCCL); The other is the development of a new language. In the proposed system, we have taken the second approach. One merit of the second approach is that it makes programs easily understandable owing to an appropriate syntax. Another merit is that it makes possible to detect the programming errors during syntactical analysis. Robot programming languages are often classified into several categories according to the level of the abstraction of a workcell. Three frequently classified levels are: task level, object level, and manipulator level, although there is no operational task level programming language yet.[12]

In the new language, we provide both object level commands and manipulator level commands, for a simple description and a detailed description respectively. The language also allows users to describe the workcell symbolically in the program so as to increase the programmability and readability. The language descriptions are divided into five groups : statements for the symbolic description of a workcell, statements for coordinating cell activities, statements for structured programming, statements for managing a world model, and data types and expressions

(1) Workcell Description

To describe a workcell, its components should be suitably abstracted for the readability and portability of a program. Considering it, in the proposed language a user can specify cell components with their attributes in the beginning of the program to prevent excessive dependency on the environment.

The description of a workcell is accomplished by simply describing attributes of cell components with their names. Some important attributes are as follows : MODEL for referencing geometric information which is already defined in the GMS; AFFIX for indicating geometric relationships with other components; BASE for expressing the location of the base frame of a component; PORT for denoting the communication port of equipment.

```
<workcell description> ::= WORKCELL { <descriptions> }
<descriptions> ::= <description>
                 | <descriptions> <description>
<description> ::= <component type> <name> <attribute list>;
<component type> ::= ROBOT | MACHINE | DEVICE
                 | OBJECT | QUEUE | CHANNEL
<attribute list> ::= MODEL = <model name>
          | AFFIX <component> [ BY <relation> ]
          | STATE = <state name> | PORT = <port name>
          | TOOL = <tool name> | S_RATE = <expression>
          | SIZE = <expression> | KEY = <expression>
          | TYPE = <type declaration> | BASE = <location>
```

When the interpreter decodes the description statements, a world model is constructed. For example, let a robot named 'puma' be connected to a vision system, 'vision'. Then, the declaration of the robot and the vision sensor is written as follows :

```
ROBOT puma MODEL="puma560" PORT=Puma; DEVICE vision
MODEL="ccd" PORT=Vision;
```

(2) Workcell control

Tasks are accomplished by controlling the tools and other manufacturing devices along the desired locations and paths in the workspace. Since tools are moved by robots and NC-machines in most cases, appropriate means of controlling and coordinating these equipments are required.

Programs for workcell control differ from other programs because the sequence of controlling the workcell depends upon unpredicted external factors which cannot be controlled. To overcome this difficulty, the proposed language provides some means for adapting to uncertain working environment such as status feedback, active motion correction, abrupt stop of motion, and monitoring the guarded conditions.

motion control statement: In the proposed system, transfer of an object or a tool in three dimensional space is accomplished by a simple MOVE command. This command directs the robot(s) to move an object or a tool from one location to another while satisfying some constraints. On the other hand, operation control of a device or a machine is executed through an OPERATE command.

Here, *prefix* defines a motion path of an object. Available paths are straight line (SMOVE), circular line (CMOVE), spline (PMOVE), and continuous path by leadthrough (LMOVE).

Object represents the subject in motion. Robot and tool as well as real object may be substituted for "object" to

```
<motion control>
      ::= <prefix> MOVE <object> TO <goal location>
                [ WITH <specifications> ] [ THEN_HOLD ];
<operation control>
      ::= OPERATE <device> [ WITH ( <expression_list> ) ]
                [ THEN_HOLD ];
<abrupt stop> ::= STOP <robot or machine>;
<specifications> ::=  <specification>
                | <specifications>  <specification>
<specification> ::= DURATION = <expression>
                | SPEED = <expression>
                | FORCE = <expression>
                | APPROACH = <vector>
                | DEPARTURE = <vector>
                | VIA <location list>
                | AFFIX ( <component list> )
                | PARAMETER = ( <expression list> )
                | GRASP ( <grasp robot>, <grasp point> )
                | SYNC /*  ctive motion control flag */
```

represent the subject of motion.
Specification is for annexing restriction or providing more detailed information to the motion planner. The information that can be designated through the specification are motion parameter, motion path, and motion control flag. The motion parameters which can be specified are velocity, duration of motion, and magnitude of applying force, etc. The motion path is described by specifying via points, approach vector, and departure vector. The motion control flag SYNC is specified with motion control statement for the tracking and the active motion correction. If the flag is set, then motion control is performed immediately without delay.
Then_hold is for the synchronization between motion and command. When this word is used at the end of motion control command, the rest of statements are performed after the motion control statement is finished. Otherwise, there may be a difference between the time of issuing a motion command and that of executing the motion.
Interlock & communication: Important requisites for controlling workcell are message communication and synchronization between tasks or activities. In our language, the communication and synchronization between tasks, which is called communication, or activities, which are called interlock, are accomplished via communication ports and channels. For communication, three kinds of primitives are provided: one for simple synchronization between tasks (SIGNAL/WAIT), another for synchronous message communication (SEND/RECEIVE), and the other is asynchronous message communication(*channel expression*). But, for interlock, only the last two primitives are provided.

In synchronous communication, a source is blocked until a destination receives the messages, and the destination is also blocked until the source sends the message. Hence, data transmission is always successful though there is a chance of indefinite time delay. However, in asynchronous communication, data transmission is not always successful, because a source or a destination does not wait for actual message transfers and there is a chance to overwrite on the previous data. *Channel expression* is an expression which uses channels as if they are variables.

```
<statements for synchroni ion between tasks>
      ::= SIGNAL  <event>;   |  WAIT  <event>;
<statements for synchronous communication>
      ::= SEND  <data>  TO  <channel>;
        | RECEIVE  <data>  FROM  <channel>;
<statements for asynchronous communication>
      ::= <channel> = <expression>;
        | <variable> = function(<channels>, <variables>);
<statements for unconditional delay>
      ::= DELAY  <expression>;
```

For guarded motion, UNTIL clause can be used with cell control statements.   e.g.:

```
MOVE  puma  TO  goal  UNTIL  pressure < 1.4 ;
```

(3) Basic Primitives for Structured Programming

The proposed system has an alternative construct (IF ˜ THEN ˜ ELSE), a repetitive construct (WHILE ˜, DO ˜ WHILE, FOR ˜), a parallel construct (COBEGIN ˜ COEND) as well as normal sequential program composition as a fundamental program structuring primitives. In addition, the language has the formatted sequential read/write primitives with file open/close primitives for input and output with the external environment.

A parallel command specifies concurrent execution of its constituent tasks. The constituent tasks are started simultaneously and the execution of parallel command is finished when the constituent tasks have all terminated. Every task started by the parallel command has no common variables except for the world model. The communication between the tasks is, therefore, possible only through communication primitives and the world model.

The proposed language has a general-purpose text macro system as in C language. It reduces the amount of repetitive typing, and allows the symbolic definition of constants and variables. Likewise, parametered functions are provided to reduce the amount of codes when similar computations or operations are required at several points in the program.

(4) World Model Control

In the integrated system consisting of various programming modules, the operation results of one module should be propagated to other modules. The propagation is accomplished through world model control primitives.

To represent the geometric relationship between each member of an assembly, the proposed system provides special pseudo commands (AFFIX/UNFIX). These pseudo commands are not for actual motion control, but for declaration only. They merely declare that one object is affixed to others. When an object is affixed to others, a user can specify the relative transformation between the objects. On the other hand, access to the data about the world model is accomplished by specifying the attribute with the name of the component. These facilities are indispensable for utilizing a world model in the cell programming as well as in the data transmission between programming modules.

AFFIX obj_1 TO obj_2 [BY <transformation>];
UNFIX obj_1 FROM obj_2;
<world access> ::= <component name> . <attribute name>

(5) Data Types and Expressions

Since geometric expressions for positions, orientation, and paths of an object in Cartesian space are frequently used in cell programming, a cell programming language should provide the representation and computation of geometric informations as well as the basic data types and expressions.

Table. 1. Geometric functions

geometric functions	vector(), rot(), frame(), joint()

The basic data types in our language are CHAR, INT, REAL, VECTOR, ROT, and FRAME. Vector consists of three real numbers specifying (x, y, z) values, which represent quantities like translation, velocity. ROT(rotation) is a 3×3 matrix representing orientation about an axis. Frame is a 4×4 matrix that is used to represent a local coordinate system or a transformation from one coordinate system to another. Operators and data types are summarized in Table. 2.

Table. 2. Data types and available operators

data types	operators
CHAR, INT, REAL	+ − * / % = && || ! < <= > >= == !=
VECTOR, ROT, FRAME	+ − * = W.R.T

5. Example

As a simple example of the programming system, we construct a robot-based workcell as shown in Fig. 3. The cell is composed of two robots(puma and tiger) to assembly workparts cooperatively, three conveyor belts, and one pallet. As parts (prisms and cubes) are transfered on the conveyor belts. The position and orientation of each part is determined using vision system (left_eye and right_eye). The function of each robot is to pick up a part from the conveyor belt and insert it into a fitted hole of a pallet in parallel, and then lift up the pallet together and transfer it to the third conveyor belt.

(1) Workcell Program

```
#define ROBOT     "ttya"
#define VISION    "ttyb"
#define LEFT_WS   "tty1"
#define RIGHT_WS  "tty2"

define workcell {
  robot  puma  model="puma"  port=Puma,
         tiger model="tiger"  port=Tiger;
  machine left_eye  model="left_eye" port=Left_eye,
          right_eye model="right_eye" port=Right_eye;
  object  prism  model="prism", cube model="cube",
          pallet model="pallet";
  queue  Puma  size=50  port=ROBOT of LEFT_WS,
         Tiger size=50  port=ROBOT of RIGHT_WS,
         Left_eye  size=2  port=VISION of LEFT_WS,
         Right_eye size=2  port=VISION of RIGHT_WS;
  channel left_camera  type=struct
                { int signal; frame loc; },
          right_camera  type=struct
                { int signal; frame loc; };
}

main()
{ rot    gp_rot={0,1,0, 1,0,0, 0,0,-1};
  frame  gp=frame(gp_rot, vector(0,0,50));
  frame  puma_grasp=frame(rot(yaxis,-90),vector(120,0,10)),
         tiger_grasp=frame(rot(yaxis,90),vector(-120,0,10));

  cobegin
    {   /* task of the left workstation*/
      frame hole = frame(rot(xaxis,0), vector(60,0,10));

        operate left_eye with parameter=( left_camera );
        while( ! left_camera.signal );
        prism.base = left_camera.loc;
        move prism to hole w.r.t pallet
                with grasp(puma, gp)
                approach=vector(0,0,200) then_hold;
      affix prism to pallet;
    }
    {   /* task of the right workstation*/
      frame hole = frame(rot(xaxis,0), vector(-35,0,10));

        operate right_eye with parameter=( right_camera );
        while( ! right_camera.signal );
        cube.base = right_camera.loc;
        move cube to hole w.r.t pallet
                with grasp(tiger, gp)
                approach=vector(0,0,100) then_hold;
      affix cube to pallet;
        move tiger to frame(gp_rot,vector(0,-300,200) w.r.t cube;
    }
  coend
    move pallet to frame(rot(xaxis,0),vector(0,600,0)) w.r.t pallet
        with grasp(tiger,tiger_grasp) grasp(puma, puma_grasp)
            departure=vector(0,0,200) approach(0,0,200);
}
```

(2) Simulation Results

Fig. 3 and Fig. 4 show the intermediate results of example program.

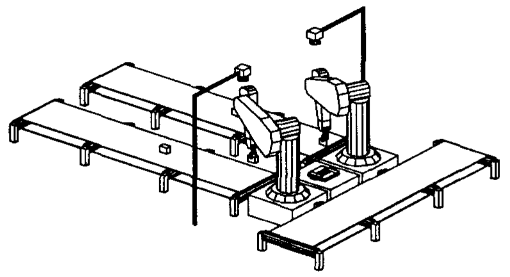
Fig. 3. A robot-based workcell

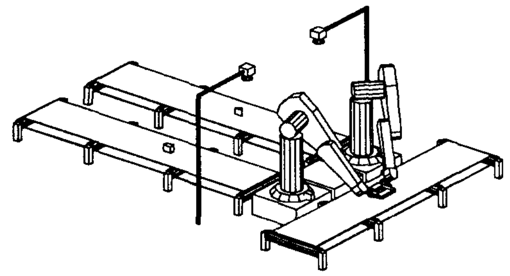
Fig. 4. Final result

6. Conclusion

Robot-based manufacturing workcell consists of various pieces of equipment such as robots, manufacturing devices, and special tools and sensors. The importance of the cooperation of these pieces of equipment is increasingly required as tasks are more sophisticated. For robot-based workcell programming, we have proposed and implemented a new programming system. The proposed system is based on world model and has classical hierarchical structure to reduce the machine dependency and to exploit the advantages of parallel processing by multi-processor.

The system is divided into four programming modules : One is the GMS which is used to design and layout workcell, another is a high level programming language, the third is the graphic simulator for conforming tasks before real execution, and the fourth is the world model to combine these modules into one integrated system. The system offers users the convenience of programming by allowing to access cell equipment and information symbolically through the world model.

The system has been implemented on a Sun-3 workstation under the UNIX operating system as a cell controller, MVME-133 32bit monoboard microcomputers as workstation controllers, and two Rhino robots as robot manipulators as in Fig. 5. Communication between these layers are accomplished through RS-232C serial communication lines. The language interpreter, the GMS, and the graphic simulator have been implemented using UNIX utilities (Lex and Yacc), Sun graphic package (SunView), and C language.

Some experiments with this system have shown its validity in off-line programming and the advantage of using the world model. But there were some bottlenecks between the cell controller and the workstation controllers in physical runs because of communication speed. Improvement in communication using a high speed network remains for further study.

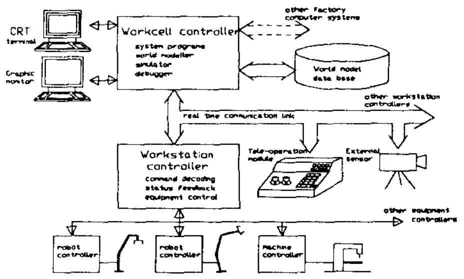

Fig. 5. Workcell control architecture

References

1. B. E. Shimano et al, "VAL-II : A New Robot Control System for Automatic Manufacturing", *Int. conf. on Robotics*, 1984
2. T. Lozano-Perez, "Robot Programming", *Proc. IEEE*, vol.71, no.7, 1983
3. S. Mujtaba and R. Goldman, *AL User's Manual*, Stanford Univ., Dec. 1981
4. S. Y. Nof, *Handbook of Industrial Robotics*, John Wiley & Sons, 1985
5. M. P. Groover et al, *Industrial Robotics Technology, Programming, and Applications*, McGraw-Hill, 1986
6. U. Rembold and K. Horman, *Languages for Sensor-Based Control in Robotics*, Spinger-Verlag, Germany, 1987

7 K. G. Shin and M. E. Epstein, "Intertask Communications in an Integrated Multirobot System", *IEEE J. of Robotics and Automation*, vol.4, no.1, 1988

8. A. Jones et al, "A Proposed Hierarchical Control Model for Automated Manufacturing systems", *J. of Manufacturing Systems*, vol.5, no.1, 1986
9. B. O. Wood et al, "MCL, The Manufacturing Control Language", *Proc. of the 13th ISIR*, 1983
10. A. Danthine and M. Geradin, *Advanced Software in Robotics*, North-Holend, 1983
11. V. Hayward and R. P. Paul, "Robot manipulator Control under Unix RCCL: A Robot Control "C" Library", *The Int. J. of Robotics Research*, vol.5, no.4, 1986
12. R. A. Volz, "Report of the Robot Programming Language Working Group : NATO Workshop on Robot Programming Languages", *IEEE J. of Robotics and Automation*, vol.4, no.1, 1988