An Example of Open Robot Controller Architecture - For Power Distribution Line Maintenance Robot System -

Yingxin He, and Kyouichi Tatsuno

Abstract—In this paper, we propose an architecture for easily constructing a robot controller. The architecture is a multi-agent system which has eight agents: the Man-machine interface, Task planner, Task teaching editor, Motion planner, Arm controller, Vehicle controller, Vision system and CG display. The controller has three databases: the Task knowledge database, the Robot database and the Environment database. Based on this controller architecture, we are constructing an experimental power distribution line maintenance robot system and are doing the experiment for the maintenance tasks, for example, "Bolt insertion task".

Keywords—Robot controller, Software library, Maintenance robot, Robot language, Agent system.

I. INTRODUCTION

WE propose a robot controller architecture for easily constructing robot systems in this paper. The architecture is composed of modularized functional agents and databases. We will show some samples of computer program functions for the agents and databases.

The features of the controller architecture are as follows:

1) It is easy to build a robot controller which is composed of the modularized components (software agents and databases).

2) Robot motions are written in a robot language. Task instruction (for example "Insert bolt") is written in the robot language.

3) System designers can define any robot language by written the corresponding program functions in C language. For example, "Find/Bolt" is defined and the instruction is executed by the function ExecuteFind() that is written in C language.

The architecture has the following two advantages compared with the other robot system architectures, for example, the RT middleware [1]. One is that the soft ware is easy to read, because all the source programs are written in C language and the databases are the text files. It is easy to understand and modify. The other is that we will open the source program library after we completed it. Using the library, the robot system designer can construct the system in short time and can add new functions.
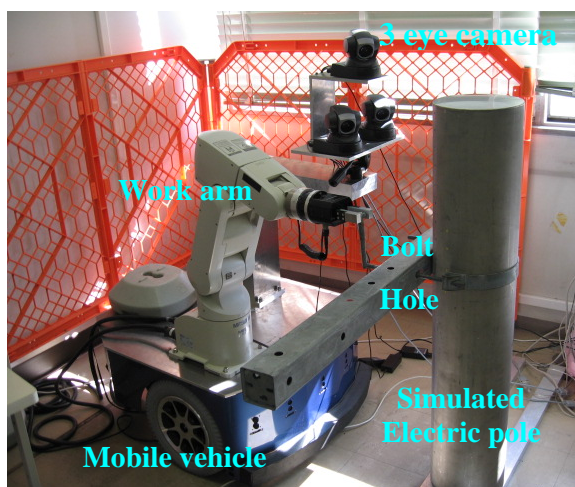

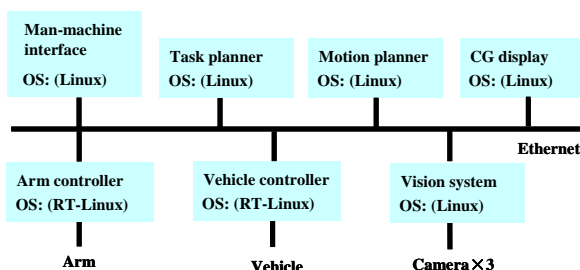Fig. 1 Experimental power distribution line maintenance robot system

Fig. 2 Hard ware architecture of an experimental power distribution line maintenance robot system

We are writing computer program functions for an experimental power distribution line maintenance robot system (Fig. 1). In this paper, we will introduce the proposed architecture which is computer software architecture. We will introduce the computer programs in the case of "Insert bolt".

Yingxin He is with Meijo University, Nagoya, Japan (e-mail: m0641504@ccmailg.meijo-u.ac.jp).

Kyouichi Tatsuno is with Meijo University, Nagoya, Japan (e-mail: tatsuno@ccmfs.meijo-u.ac.jp).
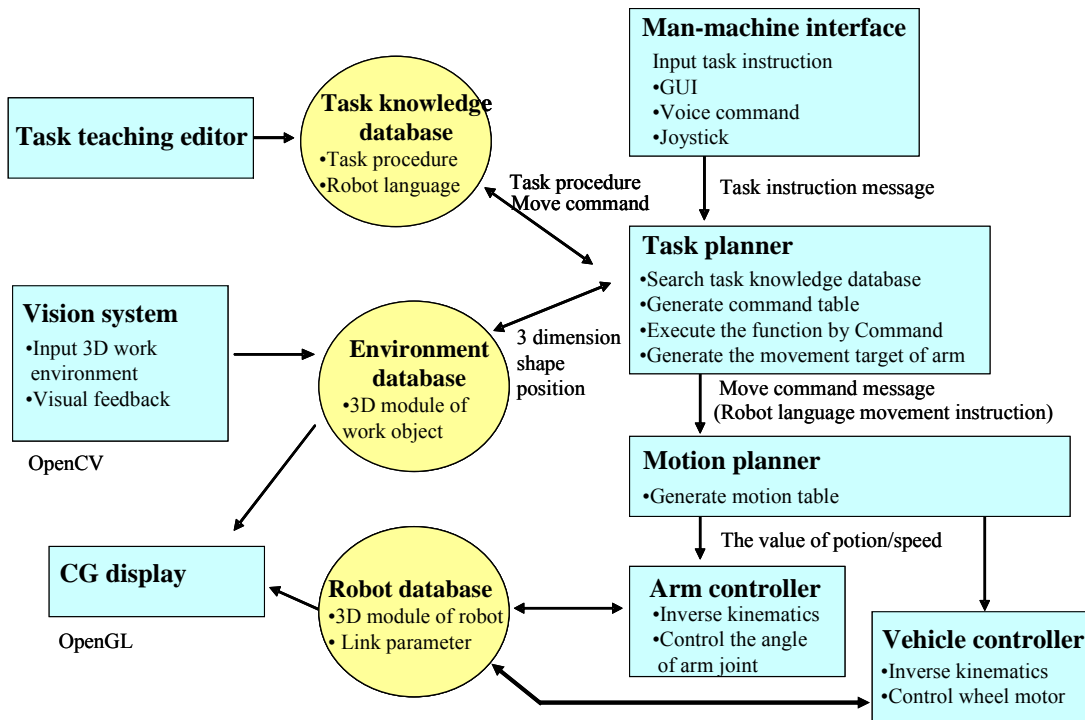
Fig. 3 Software architecture of the experimental Power distribution line maintenance robot system

II. OUTLINE OF THE PROPOSED ROBOT SYSTEM ARCHITECTURE

A. System Architecture

1) Hardware Architecture

Fig. 2 shows the hardware architecture of the experimental power distribution line maintenance robot system. In the figure, each block represents a PC (Personal Computer) and all the PC are connected by Ethernet. The software agents such as Man-machine interface, Task planner, Motion planner, and so on, are installed on the PCs.

2) Software Architecture

Fig. 3 shows the software architecture. This system is a multi-agent system composed of eight agents and three databases. The eight agents are Man-machine interface, Task planner, Motion planner, Arm controller, Vehicle controller, CG display, Vision system and Task teaching editor. The three databases are Task knowledge database, Robot database and Environment database. In this system, the instructions and requests among agents are sent as messages in the character code. By exchanging the messages among agents, the robot system executes their tasks.

B. System Operations

1. Preparation for Task Execution

The robot system has the Task knowledge database, Environment database and robot database. Before executing the robot work, we need to make these databases.

Task knowledge data bases is a set of instruction sheets of the task procedure (text file) written in the robot language. The robot language is to direct motion sequences of the robot arms and vehicle for executing the task. Task teaching editor edits this database. Task planner reads the motion sequences and executes the robot languages.

Environment database is a set of objects that describe the shape and three-dimensional position of work objects which are composed of work environments. Robot database is a database that represents the shape, link parameter and three dimensional position of the robot. Before we direct a task, we prepare 3D models of the work environments and the work objects, then, the vision system constructs the 3D models of the work environment by superimposing the 3D frame model of the work objects onto photographs that is taken by the cameras from two different positions. Fig. 4 shows the superimposing of the 3D frame model on the photograph. The Environment database and Robot database are referred when visual feedback, obstacle avoidance and CG display.
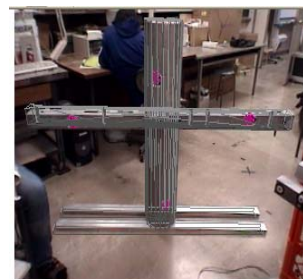
Fig. 4 The 3D frame is overlap with the mock electrical pole of the photograph

2. Task Execution

The procedure of execution is as follows:

The robot operator directs a task through a GUI (Graphical User interface) or voice input in the Man-machine interface. The GUI lists the executable task commands. For example, the operator selects the command "Insert bolt", by a mouse click or voice. The Man-machine interface recognizes the request, and sends the message of the corresponding task command to the Task planner. The message is written in character code.

Task planner receives the task command from Man-machine interface. It reads the task instruction sheet from the Task knowledge database. Then, the Task planner generates the command tables of motion instructions that are written in the robot language. Finally, it sends the each command to Monition planner, Vision and so on.

Monition planner generates an S shape trajectory of position/velocity from the current position to the target position of the arm tip. The generated trajectory (position table) is sent to the Arm controller.

Arm controller transforms the positions and attitudes of the arm tip into joint angles by the inverse kinematics. The arm controller controls the joint angles of the robot arm. It also updates the present position and attitude to the Robot database.

CG display reads Robot database and Environment database and draws the shapes of the arm links, the work objects in the environments. The OpenGL (Open graphics library) [2] is used to draw the 3D models of the robot and the objects in the environments.

From the next chapter, we will introduce the computer programs for the databases and agents.

III. DATABASE STRUCTURE

A. Structure of Task Knowledge Database

Task knowledge data base is a set of text files which describe the task procedures in robot language such as a human language, for examples, "Find" (Find the object using vision system), "MoveP" (Move under position control), "MoveVF" (Move under visual feedback), "MoveFC" (Move under force control).

Fig. 5 shows the content of the task instruction in the task knowledge database (text file). In this text file, a statement of one command is written within a line, and one line is composed of a command name and some parameter (The parameter is different by each task). Moreover, command name and parameter data are divided by the character /.

B. Structure of Environment Database and Robot Database

Environment database is a set of objects that shows the shape and three-dimensional position of work objects which are composed of work environments.

The robot database is a database that represents the shape, link parameter and three-dimensional position of the robot.

These databases are the text files that transferred from 3ds Studio max file or other CAD file. The contents of these databases are sets of structures that describe the shape, position coordinates, and the color of the object. Fig. 6 shows the structure format of the objects.
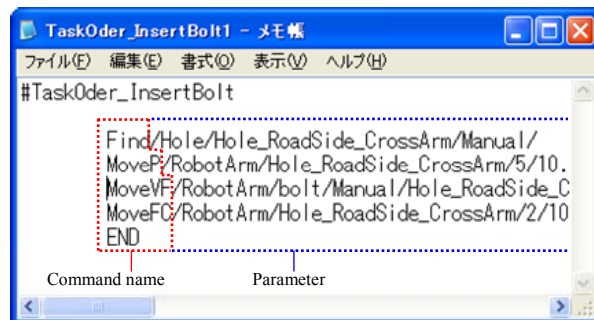

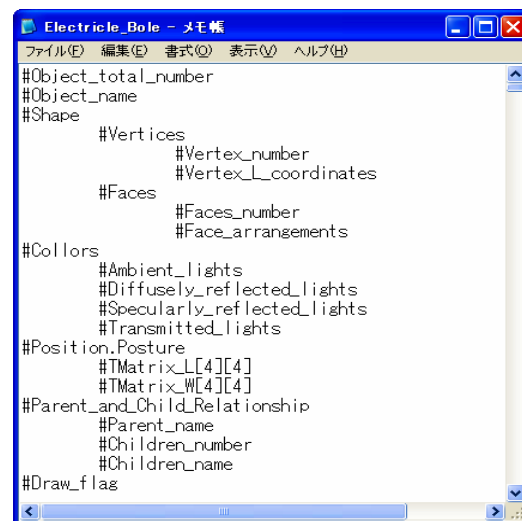Fig. 5 The task knowledge database of bolt insertion task

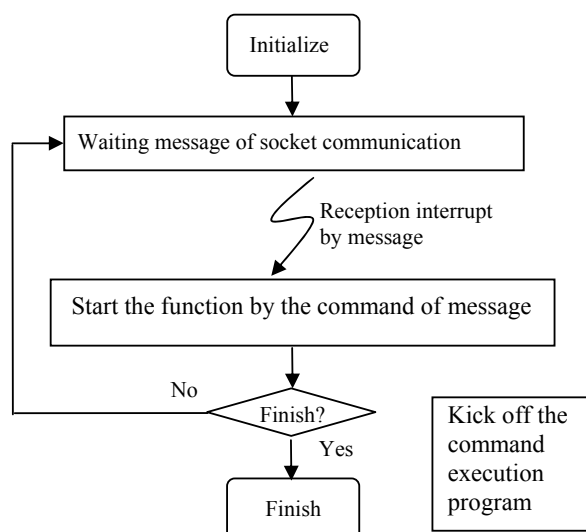Fig. 6 The structure format of the object in the Environment database

Fig. 7 Main processing flow of agent program

IV. AGENT PROGRAMS

A. The Features of Agent Programs

Each agent program starts by receiving the message and executes the corresponding programs to the commands. Fig. 7 shows the flow of a common main process.

B. Processing of Agent Program

1. Man-Machine Interface

Operating input the task instruction by Man-machine interface. For example, the user using mouse select the task command "InsertBolt" in the GUI. Then, the Man-machine interface recognizes the command and generate a message that is "Man-machineInterface/TaskInstruction/InsertBolt". It make and send this massage using function "SocketSendMessage ()".

Function: int SocketSendMessage(int SessionID, char
            *MyAgentName, char * CommandName, char
            *Contents)

Argument: SessionID: a number of sessions. The session
            has the information of destination agent,
            MyAgentName: Man-machineInterface ,
            CommandName: TaskInstruction,
            Contents: InsertBolt.

Processing:
1. Generate a message data. It is to put the character / into
    the middle of MyAgentName, CommandName and
    Contents.
2. Send message data to Task planner.
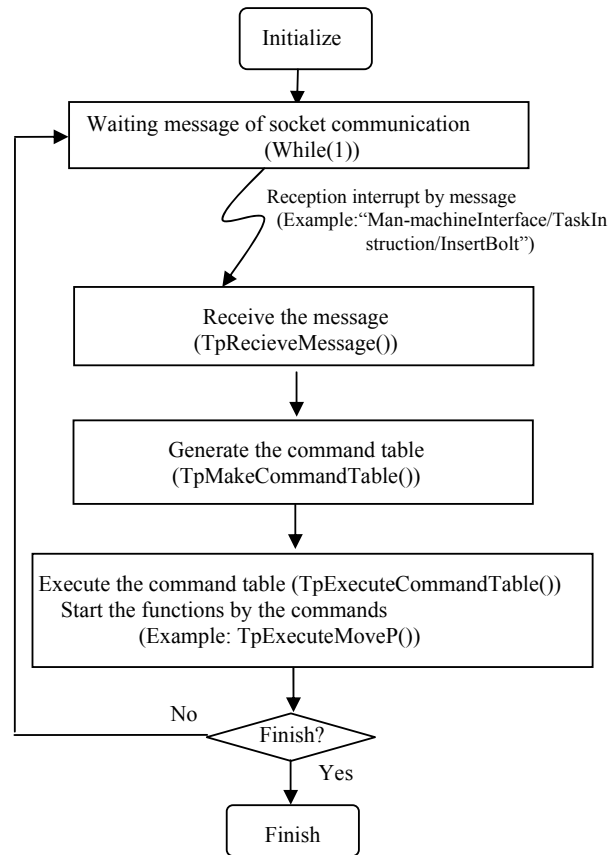   SocketSend(Socket, MessageData)

---

2. Task Planner

The main program of Task planner is a little different from the other agents. Fig. 8 shows the main flow of Task planner. After Task planner receive the message, it generates a command table first, and then it read these command in order, and start the corresponding function program to the command.

1) Receive the Message: TpReceiveMessage()

Function: void* TpReceiveMessage(char
            *MessageData, LPREQUEST lpReq)

Argument: MessageData: message data,
            lpReq: The information of socket communication.

Processing:
    1. Get the message from the reception cue
        Get_Que(&(lpReq->RecieveQue))
    If have a message
    2. The received data is saved to the "MassageData".
    If do not have message
    3. Clear the reception cue:
        Clear_Que(&(lpReq->RecieveQue)).

Fig. 8 Main processing flow of Task planner

2) Generate the command table: TpMakeCommandTable()

Function: TpMakeCommandTable(char *MessageData,
            int MaxCommandNo, char *CommandTable
            [100][300])

Argument: MessageData: Received message data,
            MaxCommandNo: Amount of generated
            command,
            CommandTable: Data of generated command
            table.

Processing:
1. IF the command is a task instruction (For example:
    "TaskInstructuion/InsetBolt"):
    • Searching the task knowledge database.
       TpSearchTaskKnowledgeDataBase()
           - Read the text file "TaskOlder_InsertBolt.txt".
    • Save the command sequences in the file to the
       Command table (Refer to Table I).
2. IF the command is not a task instruction (For example:
    "MoveP/Parameter").
    • Save the command to Command table.

Table I shows the generated command table, the commands and the parameters are the character strings (With in 300 characters) in the array. Moreover, each data of the parameter is divided by the character /. Since the concrete parameter in MoveP has too much data, it is omitted. The concrete parameter will show in Table II.

3) Execute the command table: TpExecuteCommandTable()

Function: TpExecuteCommandTable(int
      MaxCommandNo, char
      *CommandTable[100][300])
Argument:
    MaxCommandNo: Amount of generated command,
    CommandTable: Data of generated command table.
Processing:
1. Take the command name and the parameter from the first line of "CommandTable".
2. Judged the command name, and start the function that is corresponding to the command. The function corresponding to the command is as follows.
    Find/Hole → TpExecuteFind(char *parameter)
    MoveP → TpExecuteMoveP(char *parameter)
    MoveCF → TpExecuteMoveVF(char *parameter)
    MoveFC → TpExecuteMoveFC(char *parameter)
3. Execute the command by turn until the number same as the "MaxCommandNo".

4) The sample function: TpExecuteMoveP()

Here, we explain the processing of function "TpExecuteMoveP". First, Let us see the details of the MoveP command and the parameter. It is as follows:

MoveP/RobotArm/Hole_RoadSide_CrossArm/5/10/100/Position/0.000000/-0.000001/0.999999/41.198987/0.000003/-1.000000/-0.000001/-0.001019/1.000000/0.000003/0.000000/63.992144/0/0/0/1/FALSH

The explanation of the parameter is showing in Table II. And the explanation of Table II is as follows:

ⅰ）Command name
ⅱ）Arm name to be moved
ⅲ）A coordinate system for expressing an object position
ⅳ）Average velocity of moving arm
ⅴ）Average angular velocity of moving arm joint
ⅵ）Sampling time for expressing an arm trajectory
ⅶ）Control method (Position control, Force control and ets.)
ⅷ）Target position/attitude of an arm (that is a 4*4 matrix)
ⅸ）It is a selection which obstacle avoidance path is generated or not.

TABLE I
COMMAND TABLE

Leading address of table	Table contents
CommandTable[0][300]	Find/Hole/Hole_RoadSide_CrossArm/Manual
CommandTable[1][300]	MoveP/Parameter
CommandTable[2][300]	MoveVF/Parameter
CommandTable[3][300]	MoveFC/Parameter

Function: TpExecuteMoveP(char *Parameter [300])
Argument: Parameter: the parameter of MoveP.
Processing:
1. Data taransformation from character to number such as "int", "double" and so on.
2. Get the position and attitude of an arm and object from Robot database and Environment database.
3. Coordination transformation of an arm position and attitude from the object coordinate system (Refer in Parameter data ⅲ) to the arm base coordinate system.
4. Make the message "TrajectoryMoveP/Parameter".
5. Send the message to Motion planner.

3. Motion Planner

When the Motion planner receive the message "TrajectoryMoveP/Parameter", the function "MpTrajectoryMoveP()" will be started.

Function: MpExecuteTrajectoryMoveP(char* Parameter)
Argument: Parameter: the parameter of TrajectoryMoveP
Processing:
1. Data transform from character to number.
2. Trajectory of position is generated.
3. Trajectory of attitude is generated.
4. Send the message "ControlMoveP/Parameter" to Arm controller.

4. Arm Controller

When the Arm controller receive the message "ControlMoveP/Parameter", the function "AcControlMoveP()" is started.

Function: AcExecuteControlMoveP(char* Parameter)
Argument: Parameter: the parameter of ControlMoveP
Processing:
1. Data transform from character to number.
2. Invert kinematics (Transform from the target position of arm tip to the arm joint angles).
3. Servo controls the arm joint angles.
4. Send the message "UpdateRobotDB/Parameter" to the Task planner.

TABLE II
PARAMETER OF MOVEP COMMAND

Command name	Arm name	Instruction coordinate	Velocity [cm/s]	Angle velocity [deg/s]	Delta time [msec]	Control type	Target value	Obstacle avoidance
MoveP	Robot Arm	Hole_RoadSide_CrossArm	5	10	100	Position	4*4Matrix	FALSH
ⅰ	ⅱ	ⅲ	ⅳ	ⅴ	ⅵ	ⅶ	ⅷ	ⅸ

5. Vision System

The program of Vision system is made using OpenCV(Open source computer vision library) [3].

The Task knowledge database has a command "Find/Hole". We will explain the execution of this command in the Vision system. First, Task planner read this command and sent the message "Find/Hole" to the Vision system. The Vision system execute the function "VsExecuteFind()".

Function: void VsExecuteFind(char *Parameter)
Argument: Parameter: the parameter of Find.
Processing:
1. Data transform from character to number.
2. Input the target position: Click the hole position on the picture by mouse (Manual mode).
3. Measure the hole position by the triangulation method.
4. Send the message "UpdateEnvironmentDB/Parameter" to the Task planner.

V. BOLT INSERTION EXPERIMENT

Please review the task instruction for "Bolt insert" in Fig. 5. The explanation of the task procedure is as follow:

Motion1: Find/Hole, The three dimensional position of hole is measurement by Vision system. (Fig. 9 (a))

Motion2: MoveP, Move the bolt to a place which is 2cm above the hole. (Fig. 9 (b))

Motion3: MoveVF, Align the bolt tip to the hole under visual feedback. (Fig. 9 (c))

Motion4: MoveFC, Insert the bolt into the hole using force control (Fig. 9 (d))
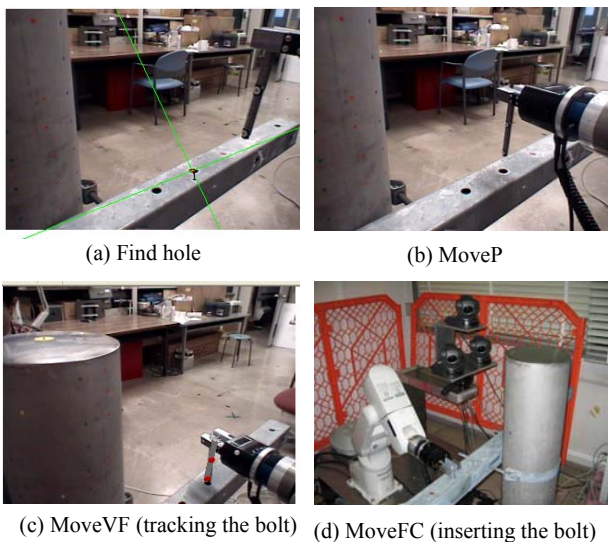
(a) Find hole   (b) MoveP

(c) MoveVF (tracking the bolt)   (d) MoveFC (inserting the bolt)

Fig. 9 The result of bolt insertion experiment

VI. CONCLUSION

We proposed a robot controller architecture to construct the robot system easily. We are building up the software library and database. We introduced the part of the software library in this paper.

The features of the controller are as the follows:

1) It is easy to build a robot controller which is composed of the modularized components (software agents and databases).

2) Robot motions are written in a robot language. Task instruction for example "Insert bolt", is written in the robot language.

3) System designers can define any robot language by written the corresponding program functions in C language. For example, "Find/Bolt" is defined and the instruction is executed by the function ExecuteFind() that is written in C language.

REFERENCES

[1] http://www.is.aist.go.jp/rt/
[2] OpenGL architecture review board, Mason Woo, Jackie Neider, TomDavis, "OpenGL Program Guide", vol.2, 2003, pp91~128, pp535~542.
[3] http://opencv.jp/