A Distributed Control and Simulation System for Dual Arm Mobile Robot

Cao Qixin, Member, IEEE, Zhang Zhen, Student, IEEE, Gu Jiajun

Abstract—In order to realize the dual arm mobile robot control and simulation, a distributed robotic system is presented. The system provides the robot’s model building, path planning, robot task planning, simulation and actual robot control functions in an indoor environment. This paper is focused on two major issues: the distributed system’s modularity and the key technologies based on the robot’s key-values in robot construction and multi-robot simulation. Experiments were conducted in order to compare the performances of the proposed system with the performances of a former centralized system. The results show that the distributed system uses less system resources, has better real time performances and satisfies the requests of dual arm mobile robot.

I. INTRODUCTION

In recent years there is an increasing use of robot into fields of replacing human to fulfill complex and dangerous tasks. The robotic system must be reliable and coordinated while finishing different subtasks such as perception, planning and navigation. Thus some robotic platforms have been developed to test the control or coordination algorithms, and evaluate the robot performances. The Player/Stage/Gazebo project is an open architecture to provide robot control and simulation in 2D and 3D environment [1], [2]. Breve is a simulation environment meant for the development of artificial life in a physically simulated world. It uses a scripting language that allows control strategies and event-based reactions to the environment for large numbers of agents [3]. CARMEN [4] use the middleware framework MARIE (Mobile and Autonomous Robot Integrated Environment) [5] to build the mobile robot control and simulation system. However the problem that all of these systems have is that their robots are simple and the task planning and simulation for the movement of complex mobile robots with redundant dual arm mobile robots can not be achieved. Our previous work [6] realizes a centralized simulation and control system. The user can easily program a dual arm mobile robot through graphical programming by editing a set of icons, preview and check the robot motion in a 3D simulation environment. However, due to precise calculation, the system costs a considerable amount of computing resources, which makes it difficult to be used on a normal PC for multiple mobile robots simulations.

Considering the above inconvenience, the robot producer YASKAWA Electric Corporation (Japan) and the Research Institute of Robotics in Shanghai Jiao Tong University (P.R.China) have initiated a project called “Simulation of Mobile Robots Navigation (SMRN)” based on their interest in distributed precise simulation and control for dual arm mobile robots. Thus a distributed navigation simulation system based on CORBA [7]-[9] was developed. Some key technologies, such as robot construction and multi-robot simulation mechanism are proposed to improve the system’s real time performance.

The remainder of the paper is organized as follows: Section II introduces the dual arm robot SmartPal, section III presents our distributed control and simulation system architecture. The proposed key technologies are proposed in section IV. Experiments and Results are shown in section V.

II. DUAL-ARM MOBILE ROBOT

The system which is presented in this paper has been realized on the dual-arm mobile robot SmartPal. By far it is mainly used as service robot in indoor environments. Fig. 1 shows the robot SmartPal used for experiments in an indoor environment. For manipulation, it is equipped with 7 DoF (degrees of freedom) arm and a finger hand. An omni-directional wheel platform serves for motion. The sensors equipped in the robot are a laser range sensor in the waist and 8 proximity sensors mounted on the

Fig. 1 SmartPal robot
omni-directional wheel platform. The Liquid Crystal Display and touch panel are attached to the front of the robot’s upper body. They are used for displaying the robotic current internal state (e.g. working, waiting, exceptional) or for user interaction (e.g. choosing the work model).

III. SYSTEM ARCHITECTURE

The architecture of the distributed system is presented in fig. 2. It includes three functional parts: the MapEditor server, the simulation server and some robot clients. The MapEditor server is responsible for indoor simulation environment model building and path planning services. The simulation server provides the virtual sensor and 2D simulation services. Its omni database keeps records of all the information of the simulation. The robot client is the client which invokes the methods from the CORBA server and presents the 3D simulation result to the users.

Due to CORBA, the service implement object in these servers can be remotely but transparently invoked from the clients regardless of their hardware, operating systems, and programming language[7]-[9]. All of the servers and clients can be distributed on different computers using CORBA middleware the Java™ IDL developed by Sun Microsystems[10]. Java™ IDL is freely available and fully compliant implementation of the CORBA standard. It provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. Each module of system is presented as follows:

A. MapEditor Server

In order to realize a multiple robot simulation, a unique virtual environment should be built using the MapEditor server. The server contains two agents: a map building agent and a path planning agent. The user can build the simulation environment by a map building agent, save the objects’ shapes, positions and other geometrical data in a geometrical map, and save the path nodes in a topological map (fig. 3);

The path planning agent is used to determine a feasible path between the start and goal points specified by the user. Fig. 4 shows a communication example between the robot client and the Path Planning agent server. In fig. 4(a), the client specifies the start point and the end point, calls the “getPath” method using the CORBA interface, and receives a set of path points from path planning agent server. Fig. 4(b) shows the map topological information. Fig. 4(c) shows the format of path points returned by the server.

B. Simulation Server

In the Simulation Server, the communication manage agent is responsible to record all the registered information from the clients and to realize a simple load balance. All robot clients wanting to join in the simulation must first register with this agent. Only the manage agent accepts the client’s request and puts the client’s name in the register list, the robot client can call the methods using the CORBA interface. If the
All the simulation information is stored in the Omni Database. It is structured in an elevators data list and a floor data list. This data structure makes it easier to exchange the single floor’s data between the Omni database (simulator server) and the Data pool of a robot client.

The 2D simulator server loads the data from the Omni database, draws the simulation scene in 2D image. This tool is capable of displaying video of the simulation scene, and allowing the simulation administrator to monitor the simulation process in the whole building. In the monitor panel, users can observe different floor scenes, see the robots’ positions, velocities and receive the data from the virtual sensor.

The virtual sensor server provides virtual laser range finder and proximity sensor agent’s services. It receives the robot client’s request with parameters such as the position of the robot and the position and direction of the virtual sensor, and returns laser scan angle, distance information. The information is similar with the information returned by the real sensor, and used by the virtual robot to detect the positions of the obstacles.

C. Robot Client

The robot agent is a development module for users to program, control and observe the virtual robot in simulation environment or actual robot in the real world. It is structured in the CORBA client that calls methods from different servers. One robot agent stands for one virtual or actual robot. The components in robot agent include:

- Graphic programming environment (GPE): it is used to specify the robot’s task by using a list of icons. Fig. 5 shows the GPE interface, here one icon describes one movement of the robot and using the icons the user can teach the robot to go to another floor, go to another room and pick a piece of paper, etc.

- Control Agent: There are two modes: virtual robot mode and real robot mode. In the virtual mode, the agent calls SmartPal robot’ virtual function library [11] to calculate the each part’s position of the robot, and return results to the GPE. Then the GPE sends the robot’s data to data pool for 3D simulation in robot client. In the real robot mode, the agent can load the codes to a real robot to execute the related task.

- Data Pool: It is used to save simulation data related with the current robot. For example, if a virtual robot is in floor 2 in simulation environment, its data pool only keeps the objects’ information on floor 2.

- 3D Simulator & Monitor Agent: It loads the simulation data from the Data pool, and constructs the robot in the 3D virtual environment.
B. Robot’s Construction and Motion in 3D Environment

In order to simplify transformation calculation, the origin of each coordinate frame is set on rod joint. The axis $Z_i$ always overlaps the rotation axis of rod joint $i$, and axis $X_i$ always on the common normal that $Z_{i-1}$ makes with $Z_i$. Fig. 6 displays the coordinate frame chain, rotation directions and angles of robot rotation.

Using D-H parameters, the relation between $Rod_{i-1}$ and $Rod_i$ can be described with four homogeneous transformations. Knowing these homogeneous transformations, the transformation matrix can be calculated as

$$A_i = Rot(z, \theta_i)Trans(0,0,d_i)Trans(a_i,0,0)Rot(x, \omega_i)$$

$$= \begin{bmatrix}
C\theta_i & -S\theta_iC\omega_i & S\theta_iS\omega_i & a_iC\theta_i \\
S\theta_i & C\theta_iC\omega_i & -C\theta_iS\omega_i & a_iS\theta_i \\
0 & S\omega_i & C\omega_i & d_i \\
0 & 0 & 0 & 1
\end{bmatrix}$$

(1)

where $A_i$ is the transformation matrix representing transformation from coordinate $i-1$ to coordinate $i$.

$Rot(z, \theta_i)$, $Trans(0,0,d_i)$, $Trans(a_i,0,0)$, $Rot(x, \omega_i)$ are the four homogeneous transformations by which we transform coordinate $i-1$ to coordinate $i$. $Rot(z, \theta_i)$ means rotation around Z axis by $\theta_i$ angle. $Trans(a_i,0,0)$ means translation from original point to a new point with increment $a_i$ in X direction, 0 in Y and Z directions. $a_i$ is the length of the common perpendicular and $d_i$ is the algebraic distance along axis $Z_{i-1}$ to the point where the common perpendicular to axis $Z_i$ is located. $\omega_i$ is the torsion angle, about axis $Z_{i-1}$, that the common perpendicular $X_i$ makes with $X_{i-1}$. $\omega_i$ is the projected angle, about $X_i$, that axis $Z_i$ makes with axis $Z_{i-1}$.

Knowing the transformation from coordinate $i-1$ to coordinate $i$, the transformation matrix from coordinate $i$ to coordinate $n$ is derived through continued product from $A_i$ to $A_n$:

$$i^{-1}T_n = A_iA_{i+1} \cdots A_n$$

(2)

Where $i^{-1}T_n$ is the transformation from coordinate $n$ to coordinate $i-1$.

In this way, the position of each joint in robot structure is described. The location of mobile robot is determined by transformation from the coordinates of the robot base frame and the Cartesian coordinates of the environment. Thus the system constructed the robot body and realized the robot moving and motion simulation. A viewing class of objects was defined to provide functions for matrix transformations, in order to support changes in the viewing angle and shape transformation.

C. Multi-Robot Simulation Mechanism

Java multi-threads are used to realize the multi-robot simulation. Two threads are set up as a manipulating thread and a detecting thread. In order to improve the real time performance of system, a data pool is created in client to store the current local environment and robots’ information. So the robot client need not always invoke the whole Omni database information in the server side, but just call the local environment data from his own data pool. The two above threads access the data pool alternately. There’s a producer and consumer relationship between them.

As presented in fig. 7, the manipulate thread calls the control agent to get the current robot information, updates
data in data pool. The data pool also exchange data with the omni database. For more efficiency, if a robot is positioned at a certain floor, in the client’s data pool. Only the data of robots and objects on that specific floor is loaded. The detect thread checks the data pool constantly and, as soon as there is a change in the data pool, the robot is rebuilt in 3D simulator and monitor agent. This mechanism is implemented for all the robots in a multi-robot system.

V. EXPERIMENTS AND RESULTS

There are performed experiments in order to compare the system load between the centralized system [6] and the distributed system which is used for controlling real robot or simulation one.

First, the centralized system is used to perform multiple robot simulation. The computer that is used has a Celeron (R) CPU 2.40GHz and 1230M usable memory (512M physical memory and 718M virtual memory). The operating system is Microsoft Windows XP Professional with Service Pack 2. Fig. 8 presents utilization rate of the CPU (fig. 8(a)) and the memory (fig. 8(b)) in the centralized system. The utilization rate of both resources increases linearly with the number of robots in the simulation increasing. The experiment in which four or more robots were considered failed for lacking memory.

![Fig. 8 Centralized system load: (a) CPU utilization; (b) Memory utilization](image1)

![Fig. 9 Distributed system load: (a) Server computer CPU utilization; (b) Server computer memory utilization](image2)

![Fig. 10 Response time in distributed system communication: (a) Instantaneous response time; (b) Average response time](image3)
In the distributed system, the MapEditor server and the simulation server are installed on a server computer that provides the services for multiple clients. Five robot clients are installed on other five separate computers. All the computers have the same hardware and software structure as the computer used for the centralized system tests. The computers were interconnected via a fast Ethernet (10 Base T) for control-communication. Fig. 9 presents the rate of the CPU and memory utilization with the increase of the number of connected clients. Both the CPU usage and memory usage of the distributed systems are increasing slower than in the case of the centralized system. In the case that three clients connect to the server computer, for a three robot simulation, fig. 9(b) shows that the server computer only spends 63 MB of memory, and the CPU average utilization is 26.57% (fig. 9(a)). The maximum CPU usage is 43.9% which is smaller than in the case of the centralized system.

The advantage brought by the distributed system consists in less CPU and memory usage. However, its shortcoming is that the clients need longer response time between the start of a method invocation and the arrival of the returned data [7]. The fig. 10(a) presents the instantaneous response time recorded over 20 seconds of simulation. The average response time value increases with the number of robots in the simulation increasing, as shown in fig. 10(b). In the case that five clients call the service from the server, the maximum of instantaneous response time is less than 100ms value that meets the requirements of a mobile robot simulation.

VI. CONCLUSIONS

In this paper, a CORBA-based distributed programming system is proposed to realize multiple dual arm mobile robots' control and simulation. Using the CORBA architecture, it’s easy to install the servers and the multiple robot clients on different computers. Each client processes computationally expensive tasks such as calculating the related robot’s key-value, rebuilding of the robot and its local environment scene. Compared to our former centralized dual arm robot system, the proposed distributed system results in less usage of the system’s resources and improved real-time performances satisfying the requirements of a complex multi mobile robots simulation.

REFERENCES


