SOFTWARE ARCHITECTURE AND TASK DEFINITION OF A MULTIPLE HUMANOID COOPERATIVE CONTROL SYSTEM

HEONYOUNG LIM*,†,‡, YEONSIK KANG†,‡, JOONGJAE LEE†,∥
JONGWON KIM†,¶ and BUM-JAE YOU†,**

*School of Mechanical and Aerospace Engineering,
Seoul National University, 599 Gwanak-ro, Gwanak-gu, Seoul, Korea
†Center for Cognitive Robotics Research,
Korea Institute of Science and Technology, Hawolgok-dong,
Wolsong-ri 5, Seongbuk-gu, Seoul, Korea
‡hylim@rodel.snu.ac.kr
§yeonsikkang@gmail.com
¶arbitlee@kist.re.kr
∥jongkim@snu.ac.kr
**ybj@kist.re.kr

Received 24 July 2008
Revised 9 March 2009
Accepted 31 March 2009

This paper presents a cooperative control software architecture that coordinates a team of multiple humanoid to complete a mission by collaborating with each other. The mission of the humanoid team is decomposed into tasks and distributed to each humanoid to be executed. Each task is described by the proposed humanoid action primitives, which are designed to abstract broad classes of humanoid tasks appropriately. In particular, missions and tasks for the humanoid team are designed by using a finite state machine with a developed user interface. The multiple humanoid cooperative control software consists of 3 layers: the mission layer, task layer, and action layer. The software architecture has scalability to the number of humanoids and the number of assigned missions with its framework based on the CORBA middleware, which integrates many different functionalities of the humanoid. The feasibility and robustness of the implemented software architecture are verified through successful completion of the mission assigned to the humanoid team while each humanoid performs its given task sequentially.

Keywords: Humanoid team; multiple robot; cooperative control; robot control software.

1. Introduction

Recently, control software architecture and cooperative control strategy for multiple robots have been drawing a lot of attention from many robotics researchers. Their applications are focused on robot soccer, exploration, rescue, surveillance and etc. In particular, ALLIANCE was developed as a control software architecture dealing with the fault tolerant cooperative control for a heterogeneous mobile
robot team performing a mission that is composed of several subtasks.\textsuperscript{5} CAM-POUT (Control Architecture for Multi robot Planetary OUTposts) was developed by the Jet Propulsion Lab as a behavior based three layered architecture in which state transition is defined by using a finite state machine at the planning level.\textsuperscript{6} BERRA (BEhavior-based Robot Research Architecture) was designed for a mobile service robot to make its functionality easy to integrate using a hybrid deliberative and reactive concept.\textsuperscript{13} The multiple robot control system handling dynamic role assignment problem for distributed multiple robot coordination should be robust against failures for application in a dynamic environment.\textsuperscript{7,14}

With these meaningful researches, the multiple robot team is currently performing well in specific mission such as an exploration, robot soccer and etc.\textsuperscript{12,15} However, more advanced research will be necessary to develop a multiple robot control system that can support different types of tasks using a heterogeneous robot team. Recently, there has been increased focus on robot assistance in activities of daily life, such as cleaning a room and washing a dish, and the humanoid is considered to be promising candidate as a home service robots.\textsuperscript{16,17} However, few attempts are being performed to develop a multiple humanoid team that can assist in our daily life. One ongoing effort at the Korea Institute of Science and Technology (KIST) has been developing network based humanoids, called “MAHRU”, “AHRA”, and “Uria” focused on assisting in human daily life based on the previous researches achievement of stable walking pattern generation and natural arm motion generation.\textsuperscript{18–20} Thus, many related works such as task handling algorithms, recognition and manipulation are being performed.

In this paper, a multiple humanoid cooperative control system (MHCCS) is explained, which is our initial effort for the development of a control system that handles multiple humanoid executing tasks by cooperating with each other. When multiple robot are operated sharing workspace, collision avoidance, and multiple task allocations are essential problems to be solved. Especially, the proposed multiple humanoid cooperative control system has differences as follows. First, in order to coordinate a humanoid team, task definition for the humanoid team is proposed, and the user can design tasks easily using a finite state diagram and XML based interface. Second, software architecture can handle multiple tasks without conflict for the heterogeneous humanoid team using a task execution algorithm. With a CORBA based framework distributed over humanoid software components for humanoid sensing, communication, and command, the MHCCS can share the information database with other components such as each humanoid control software, multiple humanoid cooperative software, and sensing software. The messages and the sensor measurements are passed between components based on events or at a pre-determined update rate. The multiple humanoid cooperative control software architecture consists of three layers: the mission layer, task layer and action layer. Once a humanoid mission is assigned by the user, it is decomposed into tasks and allocated to each humanoid in the mission layer. In the task layer, allocated tasks are again decomposed into actions and consist of a finite state diagram. In
the action layer, humanoid control commands are generated corresponding to the action. Most of our previous efforts in humanoid research, such as humanoid walking, visual servoing for grasping, dancing, recognition, and human gesture imitation could be abstracted as an unit action and integrated into an MHCCS. Furthermore, an intelligent task scheduling and event handling scheme will allow humanoid to do sophisticated and meaningful work, such as “table serving”, “cleaning” and etc.

The rest of this paper is organized as follows. Section 2 introduces the hardware and software configuration based on a CORBA based control framework that integrates various systems. Section 3 describes the definition of a multiple humanoid cooperative task that controls humanoids through high-level commands based on the finite state diagram defined by a user. In Sec. 4, a multiple humanoid cooperative control software architecture is introduced that can plan and allocate tasks. Section 5 describes the humanoid localization and path planning algorithm that get the position of a humanoid and move the humanoid to the target position. In Sec. 6, simulation and experimental results are presented to verify feasibility and performance. Finally, conclusions and future work are presented in Sec. 7.

2. Hardware and Software Configuration

2.1. Multiple humanoid team MAHRU, AHRA, Uria

The multiple humanoid team is composed of two different types of humanoid platforms; the MAHRU and Uria type platforms. MAHRU and Uria have different

Fig. 1. Researches related to the network based humanoid platform.
sensing units and also different performance levels in walking and manipulation which is because of the difference in their kinematic parameters. Therefore, the multiple humanoid team is heterogeneous with a different step size, workspace and etc.

- MAHRU was developed as a platform for network-based humanoid research. MAHRU has 6 degrees of freedom (DOF) for each arm and leg, 2 DOF for a neck, 1 DOF for a waist and 4 DOF for each hand. For real-time control, real-time linux Xenomai was used, which is a real-time development framework that cooperating with the Linux kernel. MAHRU is 150 cm in height and weight 67 kg and can be categorized as a human-size humanoid.  

- Uria was developed as a platform for research into activities of daily life under a PC based development environment. Uria has 24 DOF and is 57 cm in height and weight 5.5 kg. An embedded PC and LCD monitor are embedded in its chest and can develop even more intelligent and interactive robots using various I/O interfaces including USB camera, microphone, Gyro sensor, PIR sensor, FSR, stereo speaker and WiFi communication system. Windows XP Embedded edition is selected as the operating system.

### 2.2. StarGazer sensor system

The StarGazer sensor system was developed as an indoor localization instrument for the mobile robot and is composed of a passive land mark and StarGazer. A landmark is printed with a pattern reflecting an infra-red light attached on the ceiling. The landmark roles as an independent coordinate system with an identity number. The identity number of a landmark is computed by using a combination of circles at the intersection of the second row and column. StarGazer consists of an infra-red beam emitter and an infra-red image camera. Figure 3 shows the components of StarGazer and indicates the location, direction and ID circles of
a landmark. As shown in Fig. 3, the IR image of the pattern on the ceiling is analyzed, and the StarGazer sensor system outputs the position and heading angle of the robot. Generally, the StarGazer is fixed to the head of the robot, and the landmark is attached on the ceiling.

2.3. CORBA based framework

To efficiently realize the client and server architecture in the distributed environment, robot software components such as those in charge of vision, path planning, and control should be developed independently and integrated without difficulty. CORBA is distributed object middleware introduced by OMG.\textsuperscript{21} CORBA was most widely used as middleware framework, and various CORBA based service frameworks for a distributed robotic control system have been proposed.\textsuperscript{8,9,22} For example, Open Control Platform (OCP) was developed for an uninhabited aerial vehicle control using RT-CORBA.\textsuperscript{10} CORBA enables us to integrate several applications in a distributed environment and provides mutual operation ability such as location transparency, program transparency, and hardware and software as well as network transparency.

Without considering where the necessary server object is located or where the platform for the object is, CORBA just calls to the method of necessary object as if it calls to the object located in its machine. Thus, CORBA provides not only the network transparency, but also a function to integrate various objects in different platforms. Figure 4 shows the overall architecture of the CORBA based service framework for MHCCS.

It is composed of six main components:

- Naming server
  All CORBA objects are identified by their unique names. The naming server searches for all the objects and services by the unique name. When the naming server is being operates, the name of each service object should be registered in advance.
- Visual servoing server
  The visual servoing server calculates the 3D locations of visual objects to grasp. It supplies obtained 3D information to the motion control PC and performs the servoing procedure. Here, the motion control PC is regarded as an end point.
- Global localization server
  The global localization server offers the position of visual objects (Pack, Can) in world coordinates using two global cameras attached on the ceiling. In the case of small-sized humanoids, the global localization server also provides the MHCCS server with their position as well as their orientation information in real-time.
- Multiple Humanoid Cooperative Control System server
  The MHCCS consists of a mission, task and action manager. If a mission is loaded and the user requests execution of the mission, then humanoid control command is generated. This server sends the humanoid control command to the motion control PC and receives the position of the humanoid and object from the Global localization server.

Fig. 4. Configuration of MHCCS using a CORBA based service framework.
• Vision control PC
The vision control PC uses Windows XP as the operating system. It transfers image data captured from the Bumblebee stereo camera to the visual servoing server.

• Motion control PC
It controls various robot motions. For hard real-time control, Linux based Xenomai is used. Xenomai is a real-time development framework that cooperate with the Linux kernel to provide pervasive, interface-agnostic, hard-real-time support to user space applications, seamlessly integrated into the GNU and Linux environment.

When a given mission is performed by a robot, many interactions among main components are needed to successfully perform the mission. To demonstrate those interactions, we use the following UML sequence diagram, which shows the ordering of messages for executing a given mission in a graphical manner.

Specifically, the boxes across the top of the diagram represent classes or their instances in Fig. 5.

The vertical dotted lines indicate the timeline; time flows from top to bottom. The horizontal arrows represent messages from an object to the other objects. Solid arrows with full heads are synchronous calls, solid arrows with stick heads are asynchronous calls, and dashed arrows are return messages.

Figure 5 shows the process of performing a mission, i.e., “Clean the Table” as an example. Roughly speaking, the mission is completed in the following way. First, the mission manager generates and allocates a task for cleaning the table. Next, the task manager divides the task into three actions: walking, picking up, and dropping. Finally, these actions are passed to each corresponding component and executed sequentially.

3. Multiple Humanoid Cooperative Task Definition
3.1. Humanoid task definition
With bipedal locomotion and manipulation using two arms, the humanoid can provide services in most residential and office spaces without constraints on mobility. Therefore, the hierarchical humanoid control structure is proposed with: mission, task, action, and command levels.

Figure 6 shows a decomposition of a mission and the hierarchical structure of the humanoid control language from the mission to the command, which has a hierarchical relationship. The mission is a unit of service defined for a user that
Fig. 5. Sequence diagram representing dynamic interaction.

Fig. 6. Structure of humanoid control structure.
is assigned to multiple humanoid and decomposed into several tasks. The task is a unit of work defined for each humanoid to perform the mission. The action is a basic function specifying sensing or motion of humanoid such as voice recognition, vision sensing, walking, grasping and etc. The command is a low level humanoid control command that is used in humanoid motion planning software. Each task consists of actions, and each action consists of commands that are high-level control commands representing a unit motion such as “Walk Forward for One Meter”. The task is characterized by the combination of actions that are triggered by an event defined by a user in terms of environmental data or robot status. An action primitive describes a basic function of the humanoid type platform regardless of the type of humanoid. These basic functions should abstract a common element of humanoid action necessary to describe diverse humanoid tasks.

3.2. Actions list for Humanoid task definition

Humanoid can imitate human behavior better than mobile robot because humanoid has a greater degree of freedom similar to the human body. In particular, humanoid’s manipulation ability is higher than other robots because a humanoid has two arms with hands and fingers. Table 1 shows an example of the action units necessary for the humanoid to abstract its mission and replicate basic tasks such as “Clean the Table” or “Fetch an Object”. In Table 1, it is shown that the actions are classified into atomic actions and composite actions. The atomic action refers to a unit action, and the composite action is the combination of multiple actions. From the viewpoint of the type of humanoid motion, the action list can be divided into mobility action, arm manipulation action and mobile manipulation action. The mobile manipulation action means two actions that are performed concurrently such as Walk with Object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Action</td>
<td>Walk_Normal</td>
<td>move to the target position with normal speed</td>
</tr>
<tr>
<td></td>
<td>Walk_Slow</td>
<td>move to the target position with low speed</td>
</tr>
<tr>
<td></td>
<td>Grab_Object</td>
<td>grasp an object using hand</td>
</tr>
<tr>
<td></td>
<td>Release_Object</td>
<td>release an object using hand</td>
</tr>
<tr>
<td></td>
<td>Place_Object</td>
<td>release an object while considering object pose</td>
</tr>
<tr>
<td></td>
<td>Pull_Object</td>
<td>pull an object such as a door</td>
</tr>
<tr>
<td></td>
<td>Push_Object</td>
<td>push an object such as a door</td>
</tr>
<tr>
<td></td>
<td>Search_Object</td>
<td>search an object using the stereo vision camera</td>
</tr>
<tr>
<td></td>
<td>Approach_Object</td>
<td>move the hand to an object</td>
</tr>
<tr>
<td>Composite</td>
<td>Pick_Object</td>
<td>Search_Object→Approach_Object→Grab_Object</td>
</tr>
<tr>
<td>Action</td>
<td>Drop_Object</td>
<td>Search_Object→Approach_Object→Drop_Object</td>
</tr>
<tr>
<td></td>
<td>Open_Door</td>
<td>Search_Object→Approach_Object→Pull_Object</td>
</tr>
<tr>
<td></td>
<td>Close_Door</td>
<td>Search_Object→Approach_Object→Close_Object</td>
</tr>
<tr>
<td></td>
<td>Walk_with_Object</td>
<td>walk while holding an object in hand</td>
</tr>
</tbody>
</table>
3.3. Finite state diagram based humanoid task

By using a human robot interface based on a finite state diagram, a user can describe a mission for multiple humanoids. In addition, a user can use a predefined mission or modify the existing mission easily by using a state diagram editor. Figure 7 shows the state diagram for the task of “Fetch an Object” mission. The double lined circle is a composite action, while the single lined circle is an atomic action. State transition occurs, when the exit condition is satisfied. For example, the humanoid position is used to trigger the exit condition of a “Walk” action. And the status of the humanoid arm is used to estimate the exit condition in the pick or drop action. If the final action is completed, then the task is completed accordingly.

Figure 7 shows the state diagram of a task in “Fetch an Object” mission. There are four composite actions: “Open Door”, “Close Door”, “Pick Object”, and “Drop Object”. The user can design an action list more easily for various missions such as “Bring a Water Bottle in the Refrigerator”, and the action status is monitored through the state diagram.

As shown in Fig. 8, the composite action “Pick Object” is combined with the “Search Object”, “Approach Object”, and “Grab Object” actions. “Search Object” is a humanoid action initiating a vision algorithm looking for the object specified by the user. “Approach Object” is a unit action that commands the humanoid to move its arm to reach an object. “Grab Object” is a humanoid hand motion to grasp an object. In order to execute the composite action “Pick Object”, a humanoid should perform these atomic actions in a sequence defined by the “Pick Object” action. The predefined action list is completed sequentially through event based state transition, which is triggered by checking the exit condition for each action.

After the user finishes editing the state diagram, an XML file equivalent to the state diagram is generated from the user interface software. Table 2 shows an example of the XML file format for the “Fetch an Object” mission.
This XML file defines the mission, task, action, and event and minimizes the influence of change for the number of tasks and the environment. After the XML file is loaded, it is saved in the knowledge based database for reuse in a similar mission.

3.4. Task design using graphical user interface

The MHCCS graphical user interface is designed for the user to define the task and monitor the mission, task, action status and event. Therefore, the user can design a task for humanoids by drawing a state diagram using a developed graphical user interface (GUI), or edit an XML-based command script directly. Figure 9 shows the user interface of the MHCCS, displaying the information about the mission and the humanoid’s walking trajectory.

The graphical user interface consists of two parts. The left hand part of the GUI in Fig. 9 consists of a tap menu that displays the mission status and the task generation process, the right hand part indicates the humanoid pose and displays the experimental environment including the table, cans, trash box, and humanoids.

The left hand part of Fig. 10 indicates the task parameters of the humanoid, such as a ID, property, type, status as well as action list. Through the GUI, trajectory, task, action, command status and etc, for each humanoid can be saved as a txt format file, so that this log file can be used to analyze performance of the MHCCS.

4. Multiple Humanoid Cooperative Control Software Architecture

4.1. Software architecture

There are three types of control software architecture for a mobile robot such as deliberative, reactive, hybrid control architecture. Deliberative control architecture determines the control action through complicated sequential reasoning and planning based on the environment model, which has an advantage of reasoning and planning but has a lower response time. Reactive control architecture has tightly coupled behavior set between a sensor and an actuator; the control action...
Table 2. XML Format based on finite state diagram for “Fetch an Object”.

```xml
<MissionSet Date="2009/01/25" Description="Multiple-Humanoid MissionSet">
  <Mission id="BringObject" Property="HumanoidMission" Date="2009/01/25">
    <Task No="1" Property="OneHumanoidTask" Priority="Medium" Date="2009/01/25">
      <ActionList>
        <Action No="1" Property="Base" Type="Walk Normal">
          <WalkPar from="CurrentPosition" to="(100.00, 150.00)">
            <Transit Condition="CloseToRefrigerator" Destination="3">
              <Action>
                <Action No="2" Property="Conditional" Type="Walk Slow">
                  <WalkPar from="CurrentPosition" to="(1200.00, 1200.00)">
                    <Transit Condition="CloseToTable" Destination="7">
                      <Action>
                        <Action No="3" Property="Conditional" Type="Open Door">
                          <OpenDoorPar Size="0.5" Where="(100.00, 150.00, 100.00)">
                            <Transit Condition="Opened" Destination="6">
                              <Action>
                                <Action No="5" Property="Conditional" Type="Close Door">
                                  <Transit Condition="Closed" Destination="2">
                                    <Action>
                                      <Action No="6" Property="Conditional" Type="Pick Object">
                                        <PickPar Size="0.5" Where="(100.00, 150.00, 100.00)">
                                          <Transit Condition="Picked" Destination="5">
                                            <Action>
                                              <Action No="7" Property="Conditional" Type="Place Object">
                                                <PlacePar Size="0.5" Where="(100.00, 150.00, 100.00)"/>
                                              </Action>
                                              </Action>
                                            </Action>
                                          </Transit Condition="Picked" Destination="5">
                                        </PickPar Size="0.5" Where="(100.00, 150.00, 100.00)">
                                      </Action>
                                    </Transit Condition="Closed" Destination="2">
                                  </Action>
                                </Transit Condition="Opened" Destination="6">
                              </Action>
                            </Transit Condition="CloseToTable" Destination="7">
                          </OpenDoorPar Size="0.5" Where="(100.00, 150.00, 100.00)">
                        </Action>
                      </Transit Condition="CloseToTable" Destination="7">
                    </WalkPar from="CurrentPosition" to="(1200.00, 1200.00)">
                  </Transit Condition="CloseToTable" Destination="7">
                </Action>
              </Transit Condition="CloseToRefrigerator" Destination="3">
            </WalkPar from="CurrentPosition" to="(100.00, 150.00)">
          </Transit Condition="CloseToRefrigerator" Destination="3">
        </Action>
      </ActionList>
    </Task>
  </Mission>
</MissionSet>
```

is determined by selecting the proper behavior corresponding to the sensor value in the behavior set. Reactive control architecture responds directly but cannot support global planning and prediction. Thus, many control architectures take advantage of both deliberative and reactive control architecture, called hybrid control architecture. Most hybrid control architecture has three layers, a top layer, a middle layer, and a lower layer. To take advantage of deliberative and reactive control architecture, multiple humanoid cooperative control software is designed with three layers like most other systems with hybrid control architecture. Global planning and reasoning for a specified mission are performed in the mission layer such as task selecting and task planning and allocation. Whenever an event is generated by a humanoid and a sensor system, the reactive response is performed by detecting an exit condition in the action layer. Figure 11 shows a diagram of the mission, task and action layers and the essential components in the software architecture.
4.1.1. Mission layer

The mission layer is at the top of the multiple humanoid cooperative control software and responsible for the task handling as well as providing an interface for the user. The mission layer generates tasks and allocates the tasks to the humanoid. The Human Robot Interface Module processes the user’s request, which is given
as a state diagram or XML file. The Task Selection Module selects an appropriate task in order to carry out the mission. The Task Planning Module generates tasks according to the type of mission and the number of jobs to be performed by the humanoid. The tasks that have been planned are allocated to each humanoid by the Task Allocation Module. The task allocation needs to solve an optimization problem considering the type of the tasks, task completion time, available humanoids and etc. In particular, for the “Clean the Table” mission, each task is associated with the objects to be cleaned and the task is allocated by considering the number, workspace, and position of the humanoids.
4.1.2. Task layer

The task layer is placed in the middle layer of the software architecture where the allocated task is checked and monitored during the execution. The main function of the task layer is to generate an action list to carry out the task allocated to the humanoid. Because the task layer interacts with multiple humanoid, the task layer takes into account the collision between the humanoids. The task layer consists of six independent modules. The Workspace and Collision Avoidance Analysis Module checks the possibility of conflicts between the workspace and collision with other humanoid for performing assigned task. The workspace is predetermined considering the humanoid type. The Action Generation Module includes suitable action information to the predefined humanoid action list.

4.1.3. Action layer

The action layer is placed at the bottom layer of the multiple humanoid cooperative control software and is responsible for connecting the software architecture with external hardware such as other humanoids, sensor system and etc. The Sensing
Module requests to the sensor system for sensing of humanoids and objects and receives the position and heading of the humanoid and object. The Command Coordination Module generates a humanoid control command list corresponding to the action. The Motion Planning Module plans the motion and manipulation trajectory of the humanoid. The Exit Condition Detection Module checks the status of the humanoid and detects whether or not the current command should be terminated.

4.2. Execution of a mission for multiple humanoid team

When a user requests the execution of a mission, the XML based mission is loaded from the knowledge based database. Then, the task manager generates tasks and allocates tasks to each humanoid. The command coordinator sends humanoid commands that correspond to units of action. The humanoid status and sensor values in each sampling period are used for checking the exit condition for each humanoid action.

Figure 12 shows the processes used for generating and sending humanoid control commands for a given mission.

Once the task selection is performed, task planning and task allocation are executed in each sampling period until all the decomposed tasks are completed. The process of the task planning and task allocation involves the generation of a task execution list on the basis of the execution rule for all decomposed tasks considering the execution time and the number of humanoid. The task execution list is defined as a sequence of decomposed tasks and is updated and resorted in each sampling period using the status of the humanoid and sensing data. Thus, the activities of each humanoid are coordinated by the task execution process which consists of task planning and task allocation.

Generally, a multiple robot team is regarded as being coupled when the task is shared with other robots or allocated according to the type of robot and other tasks. In the “Clean the Table” mission, each humanoid has only one task. Hence, the task is allocated according to the type and workspace of the humanoid as well as execution time. As an example, the large sized humanoid at the front of the table cannot pick up a can placed the center point of the table while the small sized humanoid on the table has a shorter step size than the large sized humanoid. Thus, the task in the center point of the table is allocated to the small sized humanoid. If the status of large and small sized humanoids is idle in the same position, the task that requires a long moving distance is allocated to the large sized humanoid. Therefore, the task is not allocated sequentially, and the humanoid team is coupled heterogeneously. Figure 13 shows the process of the task execution. The task is characterized by priority, type and etc. The task execution list is updated and the ordering changed based on the status of the humanoids and tasks.

In Fig. 13, humanoid #1 completed the task earlier than humanoid #2, but the new task is allocated to humanoid #2, because humanoid #1 does not satisfy the workspace for that task. Although the first task of humanoid #3 and #4 is
Fig. 13. Process of the task execution.

completed concurrently, the other task in the common workspace is allocated to humanoid #3, because the predicted execution time of that task for humanoid #3 is shorter than that for humanoid #4.

5. Humanoid Localization and Path Planning

This section describes the localization algorithm for the humanoid. The position of the humanoid is measured by using the sensor system, which outputs the position of the humanoid’s head according to the global coordinate system.

5.1. StarGazer coordinate matching algorithm

Figure 14 illustrates a landmark array used in the experiment, which is composed of 30 landmarks and covers an approximate area of 500 cm $\times$ 600 cm. Because each landmark roles as a local coordination system, a matching algorithm is required to transform each local coordinate system into the reference coordinate system.

Equation (1) shows the transformation matrix used to match a local coordinate frame to the reference coordinate frame of the StarGazer sensor system. This equation is used to calibrate rotational deviation, $\theta_e$, and translational distance, $x_e$, $y_e$, $z_e$ between the local coordinate frame and the reference frame to achieve a 1–2 cm tolerance.

$$
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} =
\begin{bmatrix}
\cos \theta_e & -\sin \theta_e & 0 & x_e \\
-\sin \theta_e & \cos \theta_e & 0 & y_e \\
0 & 0 & 1 & z_e
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}.
$$

As shown in Fig. 15, the StarGazer system sends position measurements to the MHCCS software via wireless Bluetooth communication.

To achieve better accuracy, the StarGazer sensor platform outputs position estimates based on the local coordinate of the multiple markers in its camera field of view.
view. By using the nearest position estimate from the predicted humanoid position out of multiple position estimates, the probability of getting a faulty measurement is greatly reduced.

5.2. Path planning for humanoid

Humanoid motion is fundamentally different from mobile robot motion due to the humanoid’s bipedal locomotion. From the navigation point of view, there are also many differences which must be considered in order to let the humanoid safely reach the target desired position. As illustrated in Fig. 16, a humanoid requires a certain turning radius in order to change its heading to avoid a collision between two feet even though the humanoid is in a stationary state. Since humanoid’s motion is composed of a finite number of steps, the optimal path is not necessarily the shortest path, but the path that minimizes the number of steps required to reach the target position. In addition, humanoid can walk side by side. Therefore, the path planning algorithm for the humanoid must take into account these differences to allow the robot to follow the designed path and achieve natural behavior. In the MHCCS, the path is planned globally in the action layer of the software architecture.
while avoiding overlapping with obstacles and paths of the other humanoids. If an unpredicted situation is encountered, the path is replanned. The humanoid walking pattern is classified as “walk forward”, “walk backward”, “turn left”, “turn right”, “walk left”, “walk right”, and etc. To better reproduce natural human walking, the commands that induce walk forward and turning have more priority than other commands such as walk backward, walk left. A path planning algorithm based on the A star algorithm is developed to design the optimal path considering different weights on forward walking and turning based on given obstacle maps. The details of the path planning algorithm are not discussed here, because it is beyond the scope of this paper. Figure 16 shows an example of humanoid path planning in which the humanoid can avoid obstacles and arrives at the optimal target position to grab the object. The humanoid command list is generated as walk forward, turn right, turn right, walk forward, and turn right while representing the turn radius and less turning. If the path does not consider these kinematic constraints, a path may be generated that is shorter than a humanoid path such as the striped line in Fig. 16.

Since humanoid move with two legs, we need a path planning strategy unique to the humanoid types. In order for a humanoid to move toward the target point, the MHCCS sends out commands, including the number of steps and the length of each step to the humanoid. The number of steps, which is based on the length of each step, is calculated using the distance from the start point to the target point. Using Eqs. (2) and (3), the number of steps and the length of each step are computed.

\[
\text{Number of Steps for forward walking} = \frac{\text{Distance to move}}{\text{Length of Step}} + 1, \quad (2)
\]

\[
\text{Number of Steps for forward walking} = \frac{\text{Distance to move}}{2 \times \text{Length of Step}} + 1. \quad (3)
\]
6. Simulation and Experimental Results

In this section, simulation and experimental results are presented. The simulation is focused on evaluating the operation of the task execution and state transition. The experiments are performed to confirm the performance of hardware such as sensing and grasping.

6.1. Mission and task definition

The mission was “Clean the Table”, which represents organizing cans randomly placed on the table by putting them into the trash box located at the corner of the table. Thus, the number of tasks is computed by the number of cans to be cleaned up. When the user starts the mission, each humanoid walks to a can and picks up the can.

Then humanoid walk to the trash box and drop the can into the trash box. Figure 17 shows the mission environment, which consists of a table, cans and trash boxes. MAHRU and AHRA work at the front while Uria_1 and Uria_2 work on the table.

Although this is a very simple task for humans, there are many complicated processes for a humanoid to go through such as task planning, task assignment, path-planning, humanoid manipulation, motion control modules, and many essential sensing devices, in coordination under the developed MHCCS management.

Figure 18 shows the state diagram for tasks in the “Clean the Table” mission, which consist of walk, pick, walk and drop actions.

6.2. Simulation result

In the simulation, the multiple humanoid team is composed of two large sized humanoids and two small sized humanoids. The simulation aims to evaluate the

![Fig. 17. Experimental environment, humanoid, table, can, trash box.](image-url)
algorithms such as task planning, path planning, and state transition using a dynamic based simulation program called SimStudio.\textsuperscript{26}

In Fig. 19, MAHRU and AHRA performed two tasks in which each humanoid moved a can to the trash box twice. But Uria_1 and Uira_2 completed only one task. In all, six cans on the table were moved into the trash box. (a)\textendash(h) shows Uria’s sequential walk action, pick action, walk action, and drop action. Uria turns and walks to approach the trash box and then drops a can into the trash box. (i) shows the result of a completed mission. All the cans are located in the trash box, and the humanoid team waits for a request from the user. This simulation result showed two important features: task allocation and re-planning algorithm. MAHRU, AHRA, Uria_1, and Uria_2 execute concurrently the assigned first task. And then, a new task is assigned automatically by the task execution algorithm when MAHRU or AHRA has completed the assigned task. In (f)\textendash(g), the lower side Uria turns left to move to the trash box, and then the walk action is executed repeatedly. Because the lower side Uria could not reach the target position, the state transition was not triggered, and the walk action is performed repeatedly by re-planning. After the Uria arrived at the target position, the state transition is triggered, and the current walk action is changed to the drop action.

6.3. \textit{Experimental results}

6.3.1. \textit{Experimental environment}

The Multiple Humanoid Cooperative Control System is implemented with a naming server, global vision server, multi humanoid cooperative control server, motion control server and etc.
Figure 20 shows the experimental environment of the MHCCS, aluminum cans, and vision sensor. A StarGazer sensor is equipped on MAHRU’s the head and the landmark is attached to the ceiling.

6.3.2. MAHRU trajectory

In order to apply the thus developed MHCCS to a real environment, it is important to know the accurate position of a humanoid. The position of the humanoid is used for triggering the state transition as well as the task planning algorithm. This experiment was conducted in order to verify whether or not the humanoid can arrive at the target pose without deviation error using the localization system we employed. Figure 21 shows examples of the experimental results; the gray line denotes the StarGazer sensor data installed on the head of the humanoid. The dark gray line denotes the COM (Center Of Mass) point of the humanoid estimated
using the inertial measurements combined with the walking model. Since the COM estimation is based on inertial measurements, it is vulnerable to the accumulation of error originating from the sensor noise and foot slippage. Figure 21 shows the trajectory of MAHRU for a “walk right” command, MAHRU walks 120 cm from 22 cm to -98 cm along the x-axis and 1 cm to 2 cm along the y-axis. From these results, it was verified that the humanoid can move toward the target point and StarGazer can measure the humanoid position with reasonable accuracy.

6.3.3. Vision sensor system

The vision sensor system is used to get the position of humanoids and objects placed on the table. The vision sensor system consists of two web cameras installed on the ceiling and a vision server PC that discerns between humanoids and cans. More accurate position data can be obtained by calibrating the two cameras. Figure 22
presents vision sensor software that calculates the position of an object and displays the images of the cameras. In order to obtain the position and orientation of the small sized humanoid called Uria, Red and Orange marks are attached on Uria’s head. The vision sensor system tracks the pose of each Uria for each sampling time, while it obtains the position of an object following a request from the MHCCS software.

6.3.4. Two small sized humanoids experiment

Figure 23 shows the experimental results for the vision sensing and task allocation algorithm. The vision sensor can get the number and position of the can as well as Uria. Therefore, seven tasks are generated according to the seven cans on the table. The task is allocated to the Uria that is the closest. In order to get the minimum execution time, the task allocation algorithm assigns the task to the humanoid in terms of the moving distance.

6.3.5. One small sized and one large sized humanoid experiment

Figure 24 shows snapshots of the experiment. In (a)∼(e) MAHRU performed actions sequentially that are composed of walk, search, left arm grab, walk, search, right arm grab, walk, right arm drop, and left arm drop. The pose of MAHRU’s head
is changed according to position of an object to get position and perform visual servoing. (h) shows the completion of MAHRU’s assigned task. (i) shows the new can on the table that is put down after the assigned task is completed. (j) and (k) shows the task execution restarted. After another can is added, the vision sensor gets the position of the new can. Then, the task execution algorithm generates the new task according to the position of the new can. Through the experimental results, the task execution and visual servoing algorithm showed robustness.

6.3.6. Multiple humanoid team experiment

This experiment is performed to verify the proposed software architecture and task execution algorithm. Figure 25 shows the top view of the experiment where the heterogeneous robot team perform the “Clean the Table” mission which is captured by vision sensing cameras and focuses on the movement of Uria. In (a)∼(h), Uria completed the walk action, pick action, and walk action sequentially. (i) and (j) show the completion of Uria’s drop action. In (k)∼(l), Upper Uria walks backward to avoid MAHRU’s arm and completes the assigned task.

Figure 26 shows the side view of the experiment that is focused on task execution by MAHRU. In (a)∼(b) show the pictures of the walk action when MAHRU walks forward to the target position to pick up the can. (c)∼(k) show the sequential execution of the search and approach an object action, and the grab action with the can picked by the humanoid’ hand. (f) shows the results of the walk action. (g)∼(i) show the results of the search and approach an object action, and the grab action using right arm. (l)∼(o) show the results of the walk action and the right arm drop action. (n) shows a walk action and left arm drop action. (p) shows the result of the completion of the mission.
7. Conclusion and Future Work

This paper presents a multiple humanoid cooperative control system (MHCCS) in which a heterogeneous humanoid team collaborates to complete a mission assigned by a user. For efficient communication between a human and a humanoid, a human robot interface was developed for the MHCCS using a finite state diagram and XML editor. A CORBA based framework was developed for integration and communication among various components including humanoid control software, MHCCS.
software, and sensing software. To verify the performance of the MHCCS and confirm the operation of the developed software components, an experiment was performed. Experimental results showed the effectiveness of the developed MHCCS and the potential that the humanoid can be employed to assist humans in daily life.

In the future, we will extend the current action list to handle diverse tasks occurring in daily life at home and office. More complicated layers of the state diagram will be required to handle more complicated tasks. By modularizing the atomic actions into composite actions and selecting only the least number of essential actions, a vast number of tasks will be covered using a minimum number of action unit types.
Fig. 26. Experiment of the multiple humanoid team focused on MAHRU’s movement.

Acknowledgments
This work was supported by the IT R&D program of Ministry of Information and Communication/Institute for Information Technology Advancement of Republic of Korea. [2006-5028-01, Development of Cooperative Network-based Humanoids Technology]

References


Heonyoung Lim is a doctoral student at the Seoul National University’s School of Mechanical and Aerospace Engineering and received his BS and MS degrees from the Korea University of Department of Mechanical Engineering, in 1998 and 2003, respectively. From 2005 to 2007, he was Researcher at the Research Institute of Maritime and Shipbuilding Engineering (RIMSE) in Seoul National University and worked on the project of development for autonomous mobile welding robot. From 2007 to 2008, he joined at the Center for Cognitive Robotics Research in Korea Institute of Science and Technology and worked on the project of Multiple Humanoid Cooperative Control. His research interests include Multiple Humanoid Cooperative Control, Robot Control Software and Industrial Welding Robot.

Yeonsik Kang received his PhD degrees from the University of California at Berkeley in 2006. From 2006 to 2007, he was a postdoctoral researcher at the Center for Collaborative Control of Unmanned Vehicles (C3UV) of University of California at Berkeley. From 2007, he is working as a senior researcher scientist for the Center for Cognitive Robotics Research in the Korea Institute of Science and Technology. His research interests include mobile robot & humanoid navigation, nonlinear control and sensor data association for the obstacle detection.

Joongjae Lee received his BS, MS, PhD degrees in computer science from Soongsil University, Korea, in 1999, 2001, and 2005, respectively. He is presently a postdoctoral fellow for the Center for Cognitive Robotics Research in the Korea Institute of Science and Technology. His research interests include robot vision, visual serving, multiple-view geometry and augmented reality.
Jongwon Kim received his MS degree in Mechanical Engineering from Seoul National University, Seoul, Korea, in 1978, his MS degree from the Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 1980, and his PhD degree from the University of Wisconsin, Madison, in 1987. He is currently a Full Professor in the School of Mechanical and Aerospace Engineering, Seoul National University and Director with the center for intelligent robotics in Advanced Institute of Convergence Technology at Seoul National University, where he is responsible for research projects in the area of robotics. He has also been a Senior Manager at Daewoo Heavy Industries, Ltd. His research interests include intelligent manufacturing systems, parallel kinematic machines and autonomous mobile welding robot. Dr. Kim received the Best Paper Award from the ASME Manufacturing Engineering Division in 1997 and the Society of Manufacturing Engineers University LEAD Award in 1996.

Bum-Jae You received his BS degree in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1985, and his MS and PhD degrees in electrical and electronic engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1987 and 1991, respectively. He was the Head of the Robotics Division at TurboTek Co., Ltd., Korea, from 1991 to 1994, conducting the development of a high-speed image processor and motion controller for robots. He joined the Korea Institute of Science and Technology (KIST), Seoul, Korea, in 1994, where he is the Center Director at the Center for Cognitive Robotics Research. He was a Visiting Fellow in the Department of Computer Science at Yale University, New Heaven, CT, for three months in 1997, conducting three-dimensional visual tracking of polyhedral objects. His research interests include humanoids, vision-based robotics, vision-based control, real-time computer vision, intelligent service robots, and digital signal processor applications. Dr. You is a member of the Korea Robotics Society, the Korea Institute of Electrical Engineering, and the Institute of Control, Automation, and System Engineering, Korea.