REAL-TIME SPOKEN-LANGUAGE PROGRAMMING
FOR COOPERATIVE INTERACTION WITH
A HUMANOID APPRENTICE

PETER FORD DOMINEY
INSERM U846 & CNRS, Lyon, France 69675
Peter.Dominey@inserm.fr

ANTHONY MALLET
LAAS-CNRS, Toulouse, France
anthony.mallet@laas.fr

EIICHI YOSHIDA
CNRS-AIST JRL (Joint Robotics Laboratory),
UMI 3218/CRT, National Institute of Advanced Industrial Science and Technology (AIST),
Umezono 1-1-1, Tsukuba, Ibaraki, 305-8568, Japan
Eiichi.yoshida@laas.fr

Received 7 September 2007
Accepted 27 February 2009

An apprentice is an able-bodied individual that will interactively assist an expert,
and through this interaction, acquire knowledge and skill in the given task domain.
A humanoid apprentice should have a useful repertoire of sensory-motor acts that the
human can command with spoken language, along with a real-time behavioral sequence
acquisition ability. The learned sequences should function as executable procedures that
can operate in a flexible manner that are not rigidly sensitive to initial conditions. Our
study integrates these capabilities in a real-time system on the HRP-2 humanoid, for
learning a cooperative assembly task. We previously defined a system for Spoken Lan-
guage Programming (SLP) that allowed the user to guide the robot through an arbi-
trary, task relevant, motor sequence via spoken commands, and to store this sequence as
re-usable macro. Here, we significantly extend the SPL system: It integrates vision and
motion planning into the SLP framework, providing a new level of flexibility in the
actions that can be created, and it allows the user to create “generic” functions with
arguments (e.g. Give me X), and it allows multiple functions to be created.

Keywords: Real-time spoken language; cooperative interaction; humanoid apprentice.

1. Introduction
In order for humanoid robots to become useful to human users, it must be pos-
sible for the user to communicate action commands to the robot, and to transfer
knowledge of how complex tasks can be accomplished in terms of more basic capabilities within the robot’s repertoire. In this context, consider an apprentice as an able-bodied individual that will interactively assist an expert, and through this interaction, acquire knowledge and skill in the given task domain. Because of their human-like form, and sensory-motor capabilities, humanoid robots are potentially well adapted to interact with humans in this apprentice role. In this “apprentice” paradigm, one would expect that the behavioral repertoire of the robot would be expanded as a function of the robot’s experience with the human user, i.e. new behaviors are created, through SLP interaction with the human user. Our objective is to integrate robotic technology including vision and motion planning together with aspects of cooperative behavior and language-based communication, in order to provide a coherent system for adaptive human-humanoid interaction.

This research takes place in the context of our long term objective of linking language to perception and action in the context of human-robot cooperation. In this context, we demonstrated a learning mechanism that could discover the underlying grammatical constructions that related meaning representations extracted from on-line video of actions, and sentences narrating those actions generated by naive human subjects. Using this knowledge, the system could then describe new actions that it had not previously been exposed to. Nicolescu and Mataric employed spoken language to allow the user to clarify what the robot learned by demonstration. In order to explore how language can be used more directly, Lauria et al. asked naive subjects to provide verbal instructions to a robot in a visual navigation task. Their analysis of the resulting speech corpora, yielded a set of verbal action chunks that could map onto robot control primitives. They demonstrated the effectiveness of such instructions translated into these primitive procedures for actual robot navigation. This indicates the importance of implementing the mapping between language and behavioral primitives for natural language instruction or programming. Learning by imitation and/or demonstration likewise provide methods for humans to transmit desired behavior to robots. Thus, different methods have been used to allow users to transfer task knowledge to the robot, including traditional keyboard programming and the use of motion tracking for learning by demonstration. Our goal — and the novelty of this work — is to allow a hands-free condition in which the user can actively perform one role in a cooperative task, while instructing the robot at the same time, such that the robot acquired new behaviors via its interaction with the human. The difficulty is to provide a mechanism for the human to instruct the robot, while at the same time allowing him/her to continue to actively participate in the cooperative activity. The solution that we have chosen is to use spoken language in order to allow the human to communicate two distinct types of information: commands for specific actions, and “meta” commands that structure what the robot is to learn.

In this context, we have recently used spoken language to command and to “program” the HRP-2 humanoid to learn behavioral sequences in real-time. The learned programs were fixed sequences of actions, and thus had rather strict requirements
on the fixed conditions of execution — i.e. any kind of object localization was purely
open loop or based on predefined and fixed values, so objects to be manipulated
had to be in specific locations. The current study extends such methods in a com-
plimentary way.

Here, for the first time, spoken language is used for interaction with a bimanual
humanoid robot in real-time to command and create complex motor procedures
or behaviors that are robust to changes in the environment. Real-time, in this
context, refers to the fact that the robot learns as it is executing a sequence of
commands, and can immediately apply what it has learned in the next instant.
Robustness to environmental changes is provided through the learning of procedures
(i.e. actions) that take arguments (i.e. objects that can be manipulated in the
action). The learned behaviors correspond to procedures that take arguments, e.g.
“Give me X”, where the robot uses vision and motion planning to localize X which
can be arbitrarily located within the robots perceivable works space. As in Ref. 7, to
do this, we must first determine the set of action/command primitives that satisfy
two requirements: (1) They should allow a logical decomposition of the task into
units that are neither too small (i.e. move a single joint) nor too large (perform
the whole task). (2) They should be of general utility so that other tasks can be
performed with the same set of primitives.

We thus implement and investigate a real-time capability for on-line program-
ing of a bimanual humanoid, featuring “procedure learning”. We believe that this
is important in the domain of human-humanoid interaction, because in certain con-
ditions, it is more natural for the human user to teach by explaining rather than by
demonstration, though clearly both methods are of great utility. In the context of
interaction with a humanoid that has similar affordances to those of the human, this
“programming” method is also more natural because the basis of the commands is
the same.

1.1. The scenario

Figure 1 illustrates our general HRI scenario which involves humans and the HRP-2
cooperating in the construction of a small table. This construction task7 involves
attaching the legs to the surface of the table with wood screws. One user interacts
with the robot and the second user via spoken language. The user will command
the robot to give him one of the table legs from the second user. As illustrated in
Fig. 1(A), the second user holds a leg for the robot, and the robot grasps the leg
and then passes it to User 1. The user takes the leg (Fig. 1(B)) and then asks the
robot to hold the table top steady (Fig. 1(C)), while he attaches the leg to the table
(Fig. 1(D)), and finally, tells the robot he is finished.

1.2. On-line commanding and programming

On-line commanding allows the user to be responsive to new situations, and to
learn him/herself by taking the robot through a given task or tasks. On the other
Fig. 1. Human-robot interaction in a table-construction scenario. (A) Robot extends left arm with hand open, and User2 hands it the leg. (B) Robot now turns right and passes leg to User1. (C) User1 places table in robot's hand so robot can hold the table. (D) Robot holds table steady so User1 can attach the leg.

hand, for tasks that are well-defined, the user should be able to program the robot by saying the sequence of commands and storing it before the actual execution. In between these two conditions, there may arise situations in which, during the course of solving a cooperative problem with the robot, the user comes to see that despite the “open-ended” nature of a given problem set, there may be repetitive subtasks that occur in a larger context in which some uncertainty can exist. In this type of situation, the human user may want to teach the robot about the repetitive part so this can be executed as an autonomous “macro” while the user still remains in the execution loop for the components that require his/her decision.

The table assembly task corresponds to this situation. For each of the four legs, the robot should receive the leg from User2, pass it to User1 and then, hold the table surface in place while User1 fixes the leg to the table, before repeating the same procedure for the next leg. After 1 or two repetitions of this exercise, for the first leg or two, User1 should have a good idea of how this repeating subsequence goes, and can thus teach it to the robot so that the entire behavior can be accessed by a single command.
In the remainder of the paper, we present results from three distinct experiments. Section 2 introduces the hardware and software systems, and then presents the first experiment in which the humanoid apprentice learns the table assembly task from its interaction with the human user. While the learning yields significant performance improvements, the system revealed a serious limitation, based on the use of fixed postures, which made the system vulnerable to changes in the environment. This limitation is addressed in the experiment described in Sec. 3, through the introduction of a vision-based grasping capability, which allows the system to cope with variability in the locations of objects in the environment. These results are then discussed in Sec. 4.

2. Hardware and Software for Spoken Language Programming of Motor Sequences

The current studies are performed with the Kawada Industries HRP-2 humanoid robot\textsuperscript{14} under the control of the OpenHRP controller.\textsuperscript{15} The HRP-2 has 30 controlled degrees of freedom, 13 of which are used in this study. The spoken language interface technology is provided by the CSLU RAD system. This runs on a PC Pentium III Windows machine, which communicates with the OpenHRP controller via wireless internet with an ssh connection. The system is quite modular however, and the robot controller for the OpenHRP can be replaced by the controller for other robots. We have used the AIBO ERS7 with a WIFI interface, the Lynxmotion 6 DOF robot arm, and Khepera mobile robots with a serial port controller.\textsuperscript{2} Part of the novelty here is the use of the HRP-2 with many more effective degrees of freedom and possibilities for rich cooperative interaction.

2.1. Dialog management

Dialog management and spoken language processing (voice recognition, and synthesis) is provided by the CSLU Rapid Application Development (RAD) Toolkit (http://cslu.cse.ogi.edu/toolkit/). RAD provides a state-based dialog system capability, in which the passage from one state to another occurs as a function of recognition of spoken words or phrases; or evaluation of Boolean expressions. Based on the preliminary analysis of the scenario described above, we identified a set of behavioral primitives (see Sec. 2.2), and a corresponding vocabulary of words mapping onto these primitives. This vocabulary was thus provided to the system as part of its basic knowledge of language. In addition, the system was provided with some very basic knowledge of grammatical constructions to allow it to accommodate sentences like “Give me X” and extract X as the argument. This is based in part on the neural basis of how grammatical constructions achieve the mapping between sentences and meaning.\textsuperscript{16}

In the user-initiative dialog system we developed, the system prompts the user with “I am ready” and waits for the user to respond with one of the commands (Table 1) and these are immediately executed. The user can also issue commands...
for programming the robot (Table 2). These commands include “learn” and “ok” which indicate the beginning and end of a macro sequence to be stored. Thus, in a single session, a user might first operate in direct mode to become familiar with how to solve a given problem, then pass into macro learning mode, generate a new program and run it in order to simplify subsequent task execution.

2.2. Motor command primitives

We developed such a system,\textsuperscript{7} based around the table construction interaction scenario. Based on the preliminary analysis of the table-building scenario, a set of primitive actions was identified for the HRP-2. Each of these functions corresponds to a particular posture that is specified as the angles for a subset of the 30 DOFs. These actions were implemented by hand-written python scripts that specify final, hard-coded, joint angles and motion durations for the given postures. The python script execution is triggered remotely by the dialog management system, described below, and communicates directly with the low-level OpenHRP framework (Fig. 2). The motion is achieved by linearly interpolating joint angles between the starting and final configurations, for each specific action.

2.3. Learning and control primitives

In addition to the HRP-2 specific motion commands, the system requires a set of commands that allow the user to control the actual programming and program execution. These commands and their consequences are presented in Table 2. When the user invokes the “Learn” command, the dialog system begins to encode the
Table 1. Action commands.

<table>
<thead>
<tr>
<th>Motor Command</th>
<th>Resulting Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare</td>
<td>Prepare to grasp — Move both arms to neutral position, rotate chest to center, elevate left arm, avoiding contact with the work surface (6 DOF)</td>
</tr>
<tr>
<td>Left open</td>
<td>Open left hand (1 DOF)</td>
</tr>
<tr>
<td>Left close</td>
<td>Close left hand (1 DOF)</td>
</tr>
<tr>
<td>Give it to me</td>
<td>Rotate hip to pass the object in left hand to user on the right (1 DOF)</td>
</tr>
<tr>
<td>Hold</td>
<td>Center hip, raise right arm preparing to hold table top (6 DOF)</td>
</tr>
<tr>
<td>Right open</td>
<td>Open right hand (1 DOF)</td>
</tr>
<tr>
<td>Right close</td>
<td>Close right hand (1 DOF)</td>
</tr>
</tbody>
</table>

Table 2. Learning and control commands.

<table>
<thead>
<tr>
<th>Commands</th>
<th>Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn</td>
<td>Begin encoding subsequent commands</td>
</tr>
<tr>
<td>OK</td>
<td>Store encoded command sequence in macro</td>
</tr>
<tr>
<td>Macro</td>
<td>Execute the stored macro</td>
</tr>
<tr>
<td>Wait</td>
<td>Interrupt command execution until a spoken “continue” command is issued</td>
</tr>
<tr>
<td>Continue</td>
<td>Terminate the “wait” pause and resume execution</td>
</tr>
</tbody>
</table>

sequence of the subsequent commands that are issued. The user proceeds to issue action commands to achieve the desired task that will make up this sequence. When the user has finished the part of the task he wants to program, he issues the “OK” command. This results in the action sequence being written to a file. Now, when the “Macro” command is issued, this file is read into an array, and the commands are sequentially executed. Analysis of the table construction scenario above also identified the need for a “wait” capability, such that before the execution of a particular command in the stored sequence, the system should wait for the user to finish what he is doing, which the user signifies with the “continue” command. Thus, when the “wait” condition is issued, the system pauses until the “continue” command is issued.

2.4. Experiments

Using this capability, we performed two experiments in which the users and robot cooperated to assemble the table, and then to disassemble it, respectively. In each of the two experiments, there was a highly repetitive structure to the task, due to the fact that the same operations were applied to each of the four legs.

2.4.1. Experiment 1 — Assembling the table

Here is an example dialog for installing the first leg. In the dialog, the robot always indicates that it is ready to receive a command by saying “I am ready”, and it verifies what it hears by repeating the command issued by the user. These elements are removed for readability. Human speech is marked in bold, and robot speech in italics.
Prepare.  <Robot raises left hand>
Left open.  <Robot opens left hand, and User2 places the table leg in the robot’s
left hand as in Fig. 1(A)>
Left close.  <Robot closes left hand to grasp the table leg>
Give it to me.  <Robot turns to the right, holding the table leg out to User1>
Left open.  <As the Robot opens the hand, User1 reaches for and grasps the table
leg (as in Fig. 1(B))>
Hold.  <The robot orients to the work surface and raises its right hand:>
Right open.  <With the robot’s right hand open, User1 places the table top within
the robot’s grasp (Fig. 1(C))>
Right close.  <Robot closes hand to grasp and hold the table so User1 can work
on it:>
Wait.  Waiting for your signal  <During this wait period, User1 attaches the leg to
the table, and then gives the “continue” command when ready. When the robot
prompts the user for the next command, it expects a response within 30sec. If
the user anticipates that he or she will need more time, then the “wait” command
is appropriate.>
Continue.  <Robot ends the wait period and indicates it is ready to continue>
I am ready.
Right open.  <User1 takes the table from the robot and prepares for the next leg:>

At this point, the first leg has been attached, and the users have come to be
familiar with how the task goes. This allows User1 to now take the initiative of
“programming” a “macro” that will allow the whole sequence described above to be
invoked with a single command. By issuing the “Learn” command, User1 instructs
the robot to begin to make a record of the subsequent commands. He then repeats
the interaction described above. After the last “Right open” command, User1 then
indicates that the program definition is finished, by issuing the command “OK”,
and the robot stores the sequence, and associates it with the command “Macro”. At
this point, the first two legs have been installed. To initiate work with the third leg,
User1 now says “Macro” and the robot subsequently begins to execute the stored
sequence, opening its left hand while User2 places the third leg, then closing its
left hand, turning to User1 and giving him the leg, preparing to hold the table and
then holding it. In true “cooperation”, the robot waits, holding the table, while
User1 attaches the third leg, and then, gives the table back to User1 when he says
“continue”. The robot is then ready to proceed with the fourth leg.

2.4.2. Experiment 2 — Generalization to a new task: Dis-assembling
the table

Given the success with this first experiment, we then decided to test the ability
of the system to be used in a related but different task — the taking apart of the
table. This is an important test, as it allows us to determine to what extent the
set of postures we defined can generalize to a related but new task. We proceeded
in the same manner, with User1 first issuing command in “on-line” mode, to have the robot hold the table while he detached the leg, then commanding the robot to pass the leg to User2 who put it away. Given the acquired knowledge, User1 then programmed the robot during the processing of the second leg, and used the resulting macro for legs 3 and 4. It is important to note that the action primitive and learning commands in Tables 1 and 2 were developed based on an analysis of the table construction scenario. These same command components were clearly demonstrated to be of general purpose — not strictly related to one specific task — as they were perfectly adequate for allowing the robot to learn the separate and distinct behavior required for taking the table apart.

With respect to performance improvements related to the SLP, during the assembly of the first two legs, User1 issued 10 distinct commands for each of the two legs. Installing legs 1 and 2 took 3 min 25 sec and 3 min 24 sec, respectively. Once the program was learned, for legs 3 and 4, a single command initiated the program, and the user was only required to issue 3 “continue” commands in order to indicate to the robot that he was ready to proceed. Execution time was reduced to 2 min 11 sec and 2 min 33 sec, respectively. The ability to execute a suite of primitive behaviors with a single command had a significant impact on the execution time for completion of the task. Figure 3 indicates the average execution times for the individual commands under the explicit command conditions without the programmed macro (CMD) at 25.14 sec/operation, and under macro program execution conditions (PRG) at 15.00 sec/operation. Completion times were measured as the interval between the initiation of the command (spoken or derived from the learned program) and the termination of execution of the robot action. We performed a statistical analysis of the variance (ANOVA) in these completion times examining the effects of Repetition (i.e. first and second trials in either the CMD or PRG mode), and Programming condition (i.e. CMD versus PRG). Only the Programming condition had a significant effect on the completion times (ANOVA, Programming Effect: F(1, 6) = 109, p < 0.0001).

In the disassembly task of Experiment 2, as in Experiment 1, the use of the programming capability for the third and fourth leg (executed in 2:51 and 2:51, respectively) yielded significant reductions in execution time as compared with the first two legs (executed in 3:57 and 4:11, respectively). To compare performances in the two experiments, we performed a 3 way ANOVA with the factors Experiment (Exp1 versus Exp2), Programming (with versus without, i.e. PRG versus CMD), and Repetition (First versus second repetition in each condition). Figure 3 indicates that both for Exp1 and Exp2, the completion times were elevated for the CMD versus PRG conditions, i.e. action execution was slower when programming was not used. The ANOVA revealed that only the Programming effect was significant (F(1, 6) = 277, p < 0.0001).

It should be noted that the actual execution times of the robot actions do not change in the command and programmed conditions. What changes is the length of the interval between the onset of the imperative to move (from the spoken
Fig. 3. Average command execution times for the Building (Exp1) and Undoing or taking apart (Exp2) task using spoken language for on-line commanding (CMD) and for macro programming (PRG).

command, or from the learned program) and the completion of the movement. Thus, the essential part of the speedup is due to the elimination of the spoken language processing, and the command verification. The programming mode also significantly reduces the number of spoken interventions that the user must make in order to execute the same sequence of robot commands. Thus, the principal concrete performance impact is that the sequence is executed with a single command (and several “continue” commands) thus allowing the composition of a new higher level command that allows a simplified and more rapid execution of the desired behavior.

3. Programming Generalized Procedural Behavior

Two principal lessons were learned in the spoken language programming experiments. First, we learned that it is possible to define a set of behavioral primitives that can be composed into novel programs. We demonstrated this with two distinct programs — one that was used to cooperatively assemble the table, and the other to disassemble the table. We also learned that there are certain limitations to the sequencing of fixed postures. In particular, this solution requires that there is no variability in the environment for the robot. Ideally, the robot should be robust to certain levels of variability in the environment, thus providing the basis for more generalized behavior. In particular, in the table assembly, the second user was required place the legs into the gripper of the robot.

In order to address these limitations, the system must have some perceptual capabilities, and must also be able to generate motor behavior based on this
perception. In this context, we focused on the perceptual ability to visually identify and localize the four legs, and the perceptual motor ability to transport the hand to the vicinity of a localized object, and then to grasp that object.

3.1. Vision processing

The detection and localization of the table legs is performed with the HRP-2 stereo-vision system in 4 steps: image acquisition, 3D image computing, colour object detection, and 3D object localization. The off-board image acquisition module reads images from the HRP-2 stereo cameras via a firewire bus and performs distortion correction, de-bayerization and 3 channel colour gain averaging (Fig. 4(A)) in real-time. The 3D image is then computed by using a standard dense stereo-correlation algorithm, based on the Zero mean Normalized Cross-Correlation function (ZNCC) and a $9 \times 9$ pixel correlation window. The disparity image (Fig. 4(B)) is of sufficient quality (3D precision $\sim 1$ cm) that the legs position allows accurate grasping. The colour video images are then processed in order to detect the chosen leg. For each leg of a different colour, the matching module has a pre-calibrated image template which is used as an a priori model. A state-of-the-art Continuously Adaptive Mean Shift (CamShift) algorithm found in the OpenCV library is used to match those templates in the images. Figure 4(C) shows the result of the matching for the rose leg.

![Fig. 4. Vision processing. (A) Video image with distortion corrected and automatic white-balance correction. (B) Corresponding depth image — of poor quality, but precise enough to perform the grasping. (C) Video image in HSV colour space. (D) Camshift algorithm detection of rose leg.](image-url)
3.2. Inverse kinematics for reach and grasp

The 3D coordinates provided by the vision system correspond to the center of gravity of the detected object in task space, and are used to direct the hand to that location for grasping by an analytic inverse kinematics of the arm of six degrees of freedom. To generate the arm motion, first, a trajectory is computed between the current and target positions, and orientations of the hand through minimum-jerk interpolation.\textsuperscript{17} To interpolate orientations smoothly, we utilize quaternion and its spherical linear interpolation (SLERP)\textsuperscript{18} whose interpolating parameter $s \in [0, 1]$ is also calculated using the minimum-jerk trajectory. Then the calculated trajectory is sampled by the control cycle time of 5 [ms] and joint angles are obtained from the inverse kinematics. Those joint angles are provided as input to the position-based controller implemented on OpenHRP\textsuperscript{15} that generates the motions for HRP-2 humanoid robot.\textsuperscript{14} Combining object vision and inverse kinematics with the common coordinate frame thus provides the robot with a behavioral capability to localize a specified object and grasp it.

3.3. HRP-2 specific commands

These visual-motor processing capabilities are then integrated into the spoken language programming system. The behavioral result of a spoken action command that is issued either directly, or as part of a learned plan, is the execution of the corresponding action on the robot. Again, based on the preliminary analysis of the table-building scenario described above, and the experience gained from the experiments in Sec. 2, a set of primitive actions was identified for the HRP-2. With the exception of the “Take” command, each of these actions, specified in Table 3, corresponds to a particular posture or posture sequence that is specified as the angles for a subset of the 30 DOFs. These actions have been implemented in python scripts that specify final joint angles and motion durations for the given postures. In this context, the only existing HRP-2 capability we use is that of commanding joint angles and movement time in python scripts. Each of these scripts is associated

<table>
<thead>
<tr>
<th>Motor Command</th>
<th>Resulting Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready position</td>
<td>Move both arms to a ready position above the workspace (6 DOF)</td>
</tr>
<tr>
<td>Take the $\text{VAR1}$ leg.</td>
<td>Visually localize the $\text{VAR1}$ colored object, and grasp it. $\text{VAR1} = \left{\text{green, yellow, rose, orange}\right}$. (6 DOF) See Secs. 3.1 and 3.2</td>
</tr>
<tr>
<td>Open (right, left)</td>
<td>Open right/left hand (1 DOF)</td>
</tr>
<tr>
<td>Close (right, left)</td>
<td>Close right/left hand (1 DOF)</td>
</tr>
<tr>
<td>Turn (left, right, center)</td>
<td>Rotate chest left, right or center (1 DOF)</td>
</tr>
<tr>
<td>Reach (left, right)</td>
<td>Extend left/right arm (6 DOF)</td>
</tr>
</tbody>
</table>
with a spoken language command as identified in Table 3. Script execution is triggered remotely by the CSLU toolkit, and communicates directly with OpenHRP in order to achieve the required joint configurations (Fig. 1). The “Take” command is exceptional in that it is not directly related to a specific posture. Instead, it relies on the vision and inverse kinematics processing described above, in order to generate reach and grasp trajectories appropriate for grasping the indicated object.

We implemented a visual-based grasping capability as described above. The motion is achieved by interpolating between starting and final configurations in workspace and then computing joint angles using inverse kinematics presented in Sec. 3.2, for each specific action. In order to exploit this visually based grasping capability, we also introduced a command that has the form of a simple grammatical construction. Interestingly, the fact that the visually guided grasping function takes an argument (the object to be grasped) necessarily implies that this function cannot be evoked with a simple one word command, but rather that it requires a “construction” that identifies the command itself and the corresponding argument. This corresponds to the “Take the $VAR1 leg” command in Table 3.

### 3.4. Learning and interaction commands

In addition to the HRP-2 specific motion commands, the system requires a set of commands that allow the user to control the actual programming and program execution. These commands and their consequences are presented in Table 4. In Sec. 2, only a single procedure sequence could be learned, and the procedure could take no arguments. Clearly, this imposes serious limitations on generalization. The current system allows multiple procedures to be created in the spoken language programming context, and allows these procedures to take arguments. While the general solution is to allow an arbitrary number of procedures to be created and named, the current study focuses on two procedures, “Give me the $X” and “Hold this”.

When the user issues a command such as “Give me the green leg”, the system determines if it has a corresponding procedure (i.e. if this procedure has been programmed), by looking up that command in the command database. If there is no procedure defined, the robot asks the user to “show me how to give you the green leg”. The user proceeds to issue action commands to achieve the desired task that will make up this sequence for the unknown command “Give me”. Of particular interest in this case is the presence of an argument in the command. During the

<table>
<thead>
<tr>
<th>Commands</th>
<th>Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Give me the $VAR1 leg</td>
<td>Learned procedure</td>
</tr>
<tr>
<td>Hold this</td>
<td>Learned procedure</td>
</tr>
<tr>
<td>OK</td>
<td>Store encoded command sequence</td>
</tr>
<tr>
<td>Wait</td>
<td>Interrupt command execution until a spoken “continue” command is issued</td>
</tr>
<tr>
<td>Continue</td>
<td>Terminate the “wait” pause and resume execution</td>
</tr>
</tbody>
</table>
elaboration of the sequence of primitives that will make up the definition of this new command, if the user issues an embedded command with an argument, such as “Take the green leg” then the argument to the take command becomes bound to the argument for the give-me command. The assumption is that if the bounding procedure takes an argument, and an internal procedure takes an argument, then these will in fact refer to the same item. This is indeed the case in the current example. We thus make this assumption but acknowledge that more complex cases could arise where this assumption would fail. When the user has finished the part of the task he wants to program, he issues the “OK” command. This results in the action sequence being written to a file. Now, when “give me the rose leg” is issued, this file is read into an array, and the commands are sequentially executed. Importantly, the value “rose” is bound to variable $VAR1 which is passed to any commands that take arguments in the procedure, notably the “take the X leg” command. This will be more clearly illustrated in an example in Sec. 3.5.1.

As before, work with this behavioral scenario also identified the requirement for a conditional “wait”, in which the execution of a stored sequence waits for the user to finish what he is doing, which the user signifies with the “continue” command. Thus, when the “wait” condition is issued, the system pauses until the “continue” command is issued.

3.5. Experiment 3

In order to evaluate the implemented system, we performed an experiment that involved human-robot cooperation via SLP. In this experiment, the human user and the robot again cooperate to construct a small table. Here, we provide the literal transcription of the human-robot dialog, annotated with explanation about the corresponding human and robot behavior and key system functions in < > brackets. Robot language will be indicated in italics, and human language in plain text, with main commands in bold. Robot confirmation of understood commands is removed for better readability.

3.5.1. Spoken language programming for the “Give me the X” procedure

Dialog:

Describe an action. <Robot initiates the interaction>

Give me the green leg.

Please show me how to give you the green leg.

Take the green leg.

Taking the green leg. <System executes the Vision and Motion Planning to place the hand above the leg (see Fig. 5(A)).>

Is that good? <Robot asks user if the hand is correctly placed>.

Yes.

Attempting to grasp. <Robot descends hand, grasps the object and raises the hand — executing the Take command>
Describe an action.

Turn right. <Robot turns right.>
Open right hand. <Robot opens right hand, giving object to user (Fig. 5(B)).>
Ready position. <Robot pulls right hand back to ready position>
OK — Finish learning. You said OK, so I will store the learned program.

At this state in the dialog, the system writes the list of commands to a file:

```
Give me the $VAR1 leg {
  Take the $VAR1 leg
  Turn right
  Open right hand
  Ready position
}
```

It is important to note that when this procedure is subsequently invoked, the word that is recognized in the $VAR1 position for the “give” command will be passed as an argument to the take command.

3.5.2. Spoken language programming of the “hold this” procedure

At this point, the user needs the robot to hold the table while he attaches the leg.

Dialog:

Hold this. <Robot does not have a stored procedure for “hold this”>
Can you show me how to hold?
Reach left. <Robot extends left arm, and user places table in the hand (Fig. 5(C)).>
Close left hand. <Robot closes left hand to hold the table (Fig. 5(C))>
Wait. <User requests robot to wait while he attaches the leg to the table (Fig. 5(D)).>
Continue. <User indicates he is finished>
Open left hand. <Robot opens left hand to release the table>
OK — Finish learning.
You said OK, so I will store the learned program.

At this point the sequence of commands including the conditional wait are now stored as an SLP procedure for immediate re-use.

3.5.3. Execution of learned procedures

Now the user can apply these learned procedures for the 2–4th legs.

Dialog:

Give me the orange leg. <System executes the Vision and Motion Planning to place the hand above the leg.>
Fig. 5. Cooperative task execution. (A) Robot grasping rose leg. (B) Robot hands leg to User. (C) User gives table to Robot to hold. (D) Robot holds table and waits while User attaches leg.

Is that good? <Robot asks user if the hand is correctly placed>.
Yes.

Attempting to grasp. <Robot descends hand, grasps the object and raises the hand — executing the Take command>

Turning right. Opening right hand. <to give the leg to the user> Moving to ready position. <“Give” procedure finished>

Describe an action.

Hold this.

Reaching left. <Robot begins to execute the stored procedure and reaches with left hand>

Closing left hand. <User places the table in the robot’s hand, and robot closes hand to hold the table.>

Waiting for your signal. <User attaches the leg and then indicates to the robot to go on.>

Continue. Opening left hand. <Robot releases the table, assumes the ready position and turns to the center>. Moving to ready position. Turning to center <Leg is now attached, using the “give” and “hold” procedures>.
3.5.4. Performance analysis of SLP effects

In order to quantify the benefits of the learning capability, we can analyze the number of commands and the time required to perform the “give” and “hold” procedures when each command was individually enumerated during learning, versus when the learned procedures were employed. During the attachment of the first leg, the user issued 13 distinct commands, and getting the leg and holding the table took approximately 2:00 and 2:16, respectively. Once the procedures were learned, each of the remaining 3 legs was attached using only 4 commands, and the total execution time for the three legs was 5:24, reducing the assembly time for a given leg by more than half. In other words, this means that in a total task completion time of less than 10 min, the human was able to program two different behavior procedures and then immediately use them in order to assemble the table, yielding a significant reduction in the number of commands and execution time required.

Figure 6 displays the average execution times for the individual commands in the table assembly Experiment 1 and the current Experiment 3. In Experiment 3, under the explicit command conditions (CMD), execution took place at 11.9 sec/operation, and under learned program execution conditions (PRG), at 4.4 sec/operation. Completion times were measured as the interval between the initiation of the command (spoken or derived from the learned program) and the termination of execution of the robot action. We performed a statistical analysis of the variance (ANOVA) in these completion times examining the effects of Programming condition, i.e. explicitly commanding during learning, versus the program execution mode (CMD versus PRG). This Programming condition had a significant effect on the completion times (ANOVA, Programming Effect: F(1, 9) = 67, p < 0.0001).

We also compared the results from Experiment 1 and Experiment 3, with two factors — Experiment and learning (Cmd/Prg). The effect of learning was
highly significant \((F(1, 6) = 111, p > 0.0001)\). While Experiment 1 was slower than Experiment 3, the effect was at the limit of significance (due to the sample size) \((F(1, 6) = 5.6, p = 0.055)\). Most importantly, the Experiment \(\times\) Learning interaction was not reliable \((F(1, 6) = 0.27, p = 0.62)\) confirming that the significant Learning effect was not dependant on the experiment.

4. Discussion

Ideally, the robot apprentice will be able to acquire task knowledge and become more of an expert through its interaction with the human expert. We attempt to make progress in this endeavor by developing a spoken language capability for programming behavioral procedures. In the future, this should be combined with existing and developing methods for learning by demonstration and by imitation to yield a multimodal interaction capability based on spoken language programming, imitation, and demonstration.

4.1. Related work

Progress in human-robot cooperation is being made in part via well-documented methods for action learning that include demonstration and imitation. Language has been used in this context for correcting and clarifying what is being learned by demonstration. One of the fundamental requirements is to establish the grounded meaning at the base of the communication, that is, the link between human language, and robot action, and perception. This has recently been explored and developed in the domain of language based navigation. Roy and colleagues further establish these links via an amodal Grounded Situation Model that integrates perception, action, and language in a common framework for language based human-robot cooperation. Roy et al. have also demonstrated how visual perception can be used to generate an internal model of the physical scene that can then be used to allow the robot to act in the scene and describe the scene based on this internal representation. In this context, verbs correspond to sensory-motor networks or perceptually guided actions that the robot carries out (lift, pickup, touch). The internal model allows the robot to appropriately understand and use spatial relation words (e.g. left of, above) based on the human’s perspective or its own. In that implementation, the system had a fixed repertoire of actions, and it did not allow any on-line modification of the behavioral repertoire of the robot, which is part of the goal of the work we describe here.

We have made progress with a system that can learn grammatical constructions which make the mapping between predicate-argument representation of action as perceived by a robot vision system, and natural language sentences that describe that action, generalizing to new action scenes. Iwahashi and colleagues have similarly demonstrated a robust capability to use \(<\text{sentence, meaning}>\) pair corpora in order to extract the grammatical structure of sentences that describe visual scenes. Once such mappings between sentences and meaning have been established, they can be used in dialog contexts for human-robot interaction. Asoh et al.
have implemented Jijo-2, a mobile robot that performs office tasks (notification of presence or absence of individuals, guiding within the office, updating schedules etc.) based on spoken language interaction. In related work on the robot assistant BIRON, particular attention has been paid to robustness in the speech understanding component by exploiting the appropriate situated semantic contexts.23

In the context of the relation between meaning and language, we can consider that single events can be described at the level of single sentences. In contrast, an interlaced sequence of human and robot actions in a cooperation context is more appropriately described at the level of a dialog. Depending on the nature of the task, the structure of the dialog will vary. Lemon and colleagues21 have developed a development environment that allows non-expert developers to produce complete spoken dialogue systems based only on a standardized description of the process flow of control of their activity, such as verify customer identity, or pay a bill. It will be interesting to apply this technique in the development of more extended tasks for human-robot interaction.

4.2. Noise and scalability

Two principal sources of noise and error have been in spoken language understanding on one hand, and vision for inverse kinematics on the other. In both cases, we introduced failsafe devices into the system. After each spoken language command (e.g. “Give me the green leg”), the robot verified that it had understood the command correctly. Likewise, in cases of visually guided grasping, once the robot had oriented its hand to the object to be grasped, it asks the user “Is that good?”. If the inverse kinematics have correctly placed the hand above the object to be grasped, then the user replies yes, otherwise, no. In the negative case, the system retries the operation. Thus, by involving the user in the verification loop, the potential impact of speech and vision errors is minimized.

A third potential source of noise is related to the complexity of the “programs” that are created through the spoken language interface. How will the current “programming” method scale with the complexity of the task? Imagine a task requiring nested macros, or conditional behaviors, or loops. For example, how might one program the robot to (conditionally) warn the user if a part is missing, or (iteratively) pound in a peg with a mallet. Can users keep track of where they are in these more complex programs, or does some kind of GUI become necessary? We investigated one of these issues in Ref. 2 in one of the tasks required a navigating robot (Khepera) to choose one of two directions to proceed, conditional upon which of the two was blocked, as indicated by a proximity sensor. We developed an “if <condition> then <sequence1> otherwise <sequence2>" construction which allowed the user to specify arbitrary if-then-else conditions, conditional upon the set of available sensor states (e.g. left clear, right clear). We found that this construction was useful, but that the resulting increase in the complexity of the program led to the inevitable — need for debugging! That is, during the subsequent execution of a program, the
user could discover that there was an error in the program. We thus introduced a simple GUI that displayed the program, and then allowed the user to perform simple editing of the program using spoken language. The lesson learned was the following: Rather than specify a complete behavior in one complex “program”, it is better to divide the behavior into more pertinent pieces (corresponding to “give me the green leg” or “hold this”), and to maintain on-line flexibility via the interaction of the human user.

A related issue of scalability has to do with task specificity. In the current experiments, a set of postures was created, based on an analysis of the task. Thus, the necessary building blocks were in place for the user to compose into useful sequences. How will this scale to new tasks? We can propose three responses. First, in Sec. 2, we saw that within a well defined context, a set of useful postures can be recombined to perform different tasks. In this case, the postures that were developed for the assembly task were perfectly adequate for the disassembly task. As the repertoire of available postures increases, it will likely reach a plateau. Still, there is a need for on-line flexibility. Thus, second, generalization can be vastly increased by the introduction of sensory-guided behavior. As we saw in Sec. 3, the introduction of visually based grasping provided a significant capability to generalize over the position of objects to be grasped. This also introduced the notion of behavioral commands or predicates which can take arguments. Finally, the use of on-line segmentation and naming of new action primitives based on interaction with the human will allow the primitive vocabulary to grow as needed.

The use of multiple arguments also introduces some scalability issues in terms of argument reference. In the case of multiple arguments, it is necessary for the robot to understand the difference between “Give X Y” and “Give Y to X”. Indeed, we have clearly demonstrated that our grammatical construction learning capability can distinguish between these two cases, based on the presence of the preposition “to”. Indeed, we have demonstrated for English, French and Japanese, that the model is capable of using cues including word order and grammatical markers (e.g. to, by in English, go, wa in Japanese, à, par in French) to learn the correct mapping between the order of arguments in the sentence and their thematic role in the meaning representation.

We are still at the stage where the “apprentice” is fairly limited and is essentially being used by the human in an instrumental way. The “apprentice” currently has no real understanding of what is happening, and it is best for the human to retain overall control of the situation. This corresponds to the early career of the human apprentice who does little more than follow orders at the beginning without really understanding what he is doing. However, from the perspective of scalability of the current approach, this is not as bad as it seems. The naive apprentice can still do anything it is told. Thus, as long as the set of sensory motor primitives is rich enough, the appropriate composition of these primitives can allow the system an open-ended behavioral repertoire. This was most clearly demonstrated in our experiment with the HRP-2 in which, with absolutely no modification of the
control software, we could use the SLP capability to instruct the robot on how to cooperate in taking the table apart after it had been built. In the future, it will be of interest to enable the robot to extract regularities automatically from the interaction history with the user, in order to begin to construct the behavioral hierarchy more autonomously.\textsuperscript{25}

4.3. Language and meaning for robot control

Part of the richness of grammatical constructions in language is that they encode predicate — argument relations in a natural and understandable manner. These predicates can be complex relations between (or operations on) the set of accompanying arguments. This provides a vastly richer communication ability than does the use of single word commands. In the current study, we extended our previous research in spoken language programming of the HRP-2\textsuperscript{7} in this context. Here, we demonstrate, for the first time, the ability to adaptively modify the behavior of a bimanual humanoid in real time, by specifying a procedural behavior that takes arguments. This allows the procedure to operate on an entire class of objects (i.e. those that can be “taken” by vision-based grasping capability), and makes the procedure robust to variability in the position of the argument objects.

This represents a significant technical advance over the simpler ability to string together a sequence of “atomic” action or posture primitives. Learning generalized procedures that can take an arbitrary number of arguments requires specific methods for handling the functional and computational distinction between predicates and arguments. In the current study, we “prewired” the system to learn two procedures. One procedure was “Give me the X leg” where X was defined over the set \{green, yellow, rose, orange\}. Thus, the system was “prewired” to recognize commands of the form “Give me the X leg”. More generally, the system should be ready to recognize and learn the meaning of commands made up of a predicate word and an arbitrary number of arguments. This may seem quite complex, but it is not so difficult as it may seem, and part of the answer comes directly from the structure of language. Lexical categories are groups of words that are functionally related, like transitive verbs, concrete nouns, adjectives, etc. Transitive verbs, like give, take, put, throw, etc. correspond to predicates or actions with some aspect of object displacement. Concrete nouns, like block, leg, table, book, plate, etc. correspond to physical objects that can be the arguments of these predicates. Prepositions like on, by, behind, above, with, etc. correspond to configuration information concerning the spatial and temporal interaction of predicates and arguments. Exploiting this knowledge of lexical categories, the system can be prewired to recognize an open set of sentences with different predicates and multiple arguments, and to learn the sequence of predicate-argument operators that are to make up the corresponding new behavior. Thus, what we have demonstrated here with the case of the “Give me the X leg” predicate can extend to an open set of different predicates, each taking an arbitrary number of arguments. This ability to
learn grammars with predicate-argument structures based on learning over corpora of <utterance, meaning> pairs has been demonstrated in human robot interaction contexts.\textsuperscript{3,20} The distinction in the current case is that learning should take place in real time during a single interaction.

A crucial aspect of spoken language programming is the proper definition of the functional vocabulary, i.e. the behavioral primitives, and how they are mapped onto the vocabulary and the grammar. Recent studies have demonstrated how, within a given task domain such as that of providing navigation instructions, human users tend to converge on a finite set of operations and grammatical constructions that assemble these operations. Likewise, in our table construction scenario, we were able to identify a small set of behavioral primitives related to object manipulation and cooperation, and to map these behaviors into the linguistic domain of a small vocabulary and set of grammatical constructions. One method of measuring the success of such a mapping of behavioral primitive onto language is to determine if the same vocabulary can be used to allow the user to solve a related problem in SLP. Thus, in Ref. 7, we determined that the same SLP set up that we used to program the table construction behavior could be used, with no changes (other than the different program created by the user) to specify to the robot how to help in taking the table apart.

The link between grammatical argument structure and the predicate-argument structure of robot sensory-motor command structures is a natural result of the inherent link between language and meaning. This has recently been exploited in the work of Roy \textit{et al.}\textsuperscript{11,19} in which verbs correspond to sensory-motor actions (e.g. take, pick up, touch), and nouns correspond to objects which can be the arguments of these predicates. Likewise, Dominey and Boucher\textsuperscript{2–6} applied this concept to much more complex grammatical structures. Indeed, they demonstrated that a generic structure-mapping system could actually discover a variety of different grammatical constructions from a corpus of sentence-meaning pairs. These pairs were generated as naive subjects performed sensory-motor actions and simultaneously described their actions. The learned constructions then allowed the system to accurately generate the new meanings corresponding to a test corpus that had not been used in training.

**Acknowledgments**

Supported by FP7 IST Grant 215805 CHRIS. This work was carried out in the context of the AIST-CNRS JRL — the French/Japanese Joint Robotics Laboratory. We, the authors, thank Jean-Paul Laumond, Co-Director of the French-Japanese Joint Robotics Laboratory of the CNRS, for support and comments on the ms. We also thank Hajime Saito of General Robotix for invaluable assistance in technical aspects of the interface to OpenHRP. Supported by EU IST Project 215805 CHRIS, and ANR Projects Comprendre and Amorces.
References


Peter Ford Dominey is a CNRS Research Director at the Stem Cell and Brain Research Institute INSERM U846 in Lyon, France. He completed his BA in Cognitive Psychology and Artificial Intelligence, at Cornell University in 1984. In 1989 and 1993, respectively, he obtained his M.Sc. and Ph.D. in Computer Science from the University of Southern California, developing neural network models of sensorimotor sequence learning, including the first simulations of the role of dopamine in sensorimotor associative learning. From 1984 to 1986, he was a Software Engineer at Data General, and from 1986 to 1993, he was a Systems Engineer at NASA/JPL/CalTech. In 1997, he became a tenured researcher, and in 2005, a Research Director with the CNRS in Lyon, France. His research interests include understanding and simulating the neurophysiology of cognitive sequence processing, action and language, and their application to robot cognition and language processing, and human-robot cooperation. He is currently participating in several French and European projects in this context.

Anthony Mallet received his M.S. and Ph.D. degrees from the École Centrale de Lyon in 1996 and Institut National Polytechnique de Toulouse in 2001, respectively. From 2001 to 2003, he was at the LAAS-CNRS as a postdoctoral researcher and in 2004, he was at the EPFL in the ASL1 laboratory, also as a postdoctoral researcher, working on software architecture and software reusability for robotics. From 2005 to 2006, he worked as a research and development Engineer in a startup company,
working on a distributed, virtual storage system for clusters and computing grids. Since 2006, he occupies a research engineer position at LAAS-CNRS, working on the robotics platform entity of the laboratory.

Eiichi Yoshida received his M.E and Ph.D. degrees on Precision Machinery Engineering from Graduate School of Engineering, the University of Tokyo in 1993 and 1996, respectively. In 1996 he joined the former Mechanical Engineering Laboratory, Tsukuba, Japan. He is currently a senior research scientist in Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan. From 1990 to 1991, he was a visiting research associate at Swiss Federal Institute of Technology at Lausanne (EPFL). From 2004, he has been at LAAS-CNRS, Toulouse, France, as Co-Director of former CNRS-AIST Joint French-Japanese Robotics Laboratory (JRL). He currently serves as Co-Director of CNRS-AIST JRL (Joint Robotics Laboratory), UMI 3218/CRT, AIST, Japan. His research interests include humanoid robotics, task and motion planning, and modular robotic systems.