5

Handling Manually Programmed Task Procedures in Human–Service Robot Interactions

Yo Chan Kim and Wan Chul Yoon
Korea Advanced Institute of Science and Technology
Republic of Korea

1. Introduction

Although a few robots such as vacuum cleaning robots (Jones, 2006; Zhang et al., 2006), lawn mowing robots (Husqvarna; Friendlyrobotics), and some toy robots (Takara; Hasbro) have single functions or perform simple tasks, almost all other service robots perform diverse and complex tasks. Such robots share their work domains with humans, with whom they must constantly interact. In fact, the complexity of the tasks performed by such robots is a result of their interactions with humans. For example, consider a scenario wherein a robot is required to fetch and carry beverages: the methods of delivery are numerous and vary depending on user requirements such as type of beverage, the needs for a container, etc. For a robot designed to control various household devices such as illuminators, windows, television, and other appliances, several services must be provided in various situations; hence, a question-and-answer interaction or some method to infer the necessity of the services is required.

For service robots to perform these complex behaviors and collaborate with humans, the programming of robot behavior has been proposed as a natural solution (Knoop et al., 2008). Robot behavior can be programmed manually using text-based and graphical systems, or automatically by demonstration or instructive systems (Biggs & MacDonald, 2003). Recently, many researchers have proposed methods for a service robot to learn high-level tasks. The two main methods are (1) learning by observing human behaviors (Argall et al., 2009) and (2) learning by using procedures defined by humans (a support system can be used to define these procedures) (Lego, 2003; Ekvall et al., 2006)..

Manual programming systems are more efficient in creating procedures necessary to cope with various interactive situations than automatic programming systems since the latter require demonstrations and advice for every situation. However, in the process of programming behavior, there exist sub-optimalities (Chen & Zelinsky, 2003), and manual programming systems are more brittle than automatic programming systems.

The sub-optimalities of manual programming systems are as follows: (a) in the writing process, humans can make syntactic errors when describing task procedures. For example, writers often misspell the names of actions or important annotations. However, if the errors do not alter the semantic meaning, the problem can be prevented by writing support

Source: Human-Robot Interaction, Book edited by: Daisuke Chugo,
ISBN 978-953-307-051-3, pp. 288, February 2010, INTECH, Croatia, downloaded from SCIYO.COM

systems such as in the case of Lego Mindstorms. (b) Another sub-optimality can occur if humans fail to devise all possible behaviors for situations that a robot will confront. In the example of beverage-delivery errands, a writer may describe a sequence in a scene wherein a robot picks up a cup. However, the writer might possibly omit a sequence in a scene wherein a robot lifts a cup *after* picking up the beverage. It is not easy for humans to infer all possible situations and consequent branching out of behavior procedures; hence, an automated system should be able to support such inference and manage robots by inferring new situations based on the given information. (c) The sequence written by a human may be wrong semantically. Humans can insert wrong actions, omit important actions, and reverse action orders by making mistakes or slips. For example, a procedure for a robot for setting a dinner table might not contain actions for placing a fork; this is an example of omission. Another procedure might consist of actions for placing a saucer after placing a teacup. Some researches have attempted to resolve this problem by synthesizing or evaluating a set of procedures based on pro-conditions and the effects of knowledge of each unit action, for example, such as in the case of conventional planning approaches in artificial intelligence field (Ekvall et al., 2006; Ekvall & Kragic, 2008). Moreover, it is possible to search for wrong sequences in procedures by using rules that deal with sequential relations between actions; such rules can be extracted using a statistical data mining method (Kwon et al., 2008). Despite these efforts, the problem of identifying whether a procedure is natural and acceptable to humans continues to be a difficult problem.

In this chapter, we propose methodologies to mitigate the last two sub-optimalities (b and c) using a programming language that can be used to describe the various task procedures that exist in human–service robot interactions.

2. Scripts, abstract task procedures for service robots

In this section, we explain task procedures that are programmed by humans. These task procedures refer to abstract robot behaviors occurring in service domains (Kim et al., 2007). The script is expressed by using a generic procedural language and can be written by humans, especially non-experts, via graphic-based or text-based interface systems. Each script contains several actions and branch-able states (explained in 2.2).

2.1 Action

Action primitives in scripts are the basic units of scripts. These are black boxes from a user's viewpoint, because the user does not have to possess detailed knowledge of their functioning, even when the units are applied to specific hardware platforms via many different modules. There are two types of action primitives: physical actions such as "move to location of object A" or "open the facing door" and cognitive actions such as "find location of object A" or "decide which beverage to take." Physical actions are performed by physical action executors, and the actions play roles as the goals of the executors (Fig. 1.). When cognitive actions are performed, knowledge inference engines explore or reason the related information. Based on the reasoned information, the Decision Manager asks questions to users. The process of asking question has been explained in our previous report (Kim et al., 2007). Some rewriteable sets of action primitives can be defined as abstract actions and used in the script database.

2.2 Branch-able state

A branch-able state refers to an interaction-related state that determines the characteristic of the script in which it is included. "Does the user want to turn on a television? Yes" or "Is it necessary to use a cup to fill the beverage? No" are examples of the branch-able state. These states are the principal evidences for checking whether a script coincides with the current situations or user's demands when a Script-based Task Planner handles scripts.

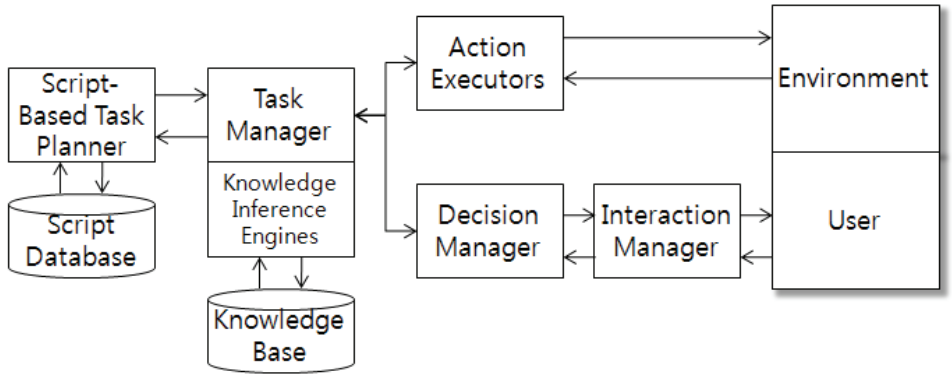

Fig. 1. Configuration diagram of the developed system

2.3 Script

A script is a sequential set of actions and branch-able states. A script database contains the scripts, and it is described in XML. As an example, Fig. 2 shows a script for the delivery of beverages.

```
<script goal="FetchAndCarryBeverage" scriptID="FCB002">
        <action decomposetype="concrete" acttype="cognitive">DecideTargetBeverage</action>
        <action decomposetype="concrete" acttype="cognitive">IdentifyLocationOfBeverage</action>
        <action decomposetype="concrete" acttype="physical">MoveToLocationOfBeverage</action>
        <BranchableProperty>InvisibilityOfBeverage:yes</BranchableProperty>
        <action decomposetype="concrete" acttype="physical">UncoverTargetLoc</action>
        <action decomposetype="concrete" acttype="physical">PickUpTargetBeverage</action>
        <action decomposetype="concrete" acttype="physical">CoverTargetLoc</action>
        <action decomposetype="concrete" acttype="cognitive">DecideNecOfContainer</action>
        <BranchableProperty>NecessityOfContainer:no</BranchableProperty>
        <action decomposetype="concrete" acttype="physical">MoveToDrink</action>
        <action decomposetype="concrete" acttype="physical">DeliverBeverage</action>
</script>
```

Fig. 2. An example of script describing delivery of beverage

3. Related work

To solve the problem of the two sub-optimalities mentioned in the introductory section, it is useful to identify the relationships between several scripts. Some researchers in the field of programming by demonstration analyzed various programmed procedures and derived information that is referable for enhancing the performance of a procedure, for example, the relationships between actions or more abstract procedures (Breazeal et al., 2004; Nicolescu & Matarić, 2003; Ekvall & Kragic, 2008; Pardowitz et al., 2005). Nicolescu and Matarić (2003)

represented each demonstration as a directed acyclic graph (DAG) and computed their longest common subsequence in order to generalize over multiple given demonstrations. Ekvall and Kragic (2008) converted sequential relationships between all two states as temporal constraints. Whenever a sequence was added, the constraints that contain contradictions with the constraints of the new sequence were eliminated in order to extract general state constraints. Pardowitz et al. (2005) formed task precedence graphs by computing the similarity of accumulating demonstrations. Each task precedence graph is a DAG that explains the necessity of specific actions or sequential relationships between the actions.
These researches are appropriate for obtaining task knowledge from a small number of demonstrations. However, when a large number of procedures are demonstrated or programmed, these approaches continue to generate one or two constraints. These strict constraints are not sufficient to generate variations in the given demonstrations or to evaluate them.

4. Handling scripts

We propose two algorithms for reducing the sub-optimalities from a large number of scripts. One is an algorithm that generates script variations based on the written scripts. Since the written scripts are composed from a human's imagination, they cannot be systematic or complete. The set of scripts takes either a total-ordered form or a mixture of total-ordered and partial-ordered forms. Our algorithm generates a DAG of all scripts, and hence, it permits the revealing of branches and joints buried among the scripts. The other algorithm is for evaluating the representativeness of a specific script by comparing the given script set. We can generate a sequence that is able to represent entire given scripts. If almost all scripts are semantically correct and natural, the naturalness of a specific script can be estimated by evaluating its similarity with the representative script. Therefore, this algorithm involves an algorithm that generates a representative script and an algorithm that measures similarities with it.
These two algorithms are based on an algorithm for partial order alignment (POA, Lee et al., 2002). Hence, we first explain POA before describing the two algorithms.

4.1 POA algorithm

We utilized an algorithm from multiple sequence alignment (MSA), which is an important subject in the field of Bioinformatics, to identify the relationships among scripts. In MSA, several sequences are arranged to identify regions of similarity. The arrangement can be depicted by placing sequences in a rectangle and inserting some blanks at each column appropriately (Fig. 3.). When we attempt to obtain an optimal solution by dynamic programming, this process becomes an NP-complete problem. Therefore, several heuristics are presented (POA (Lee et al., 2002), ClustalW (Thompson et al., 1994), T-Coffee (Notredame et al., 2000), DIALIGN (Brudno et al., 1998), MUSCLE (Edgar, 2004), and SAGA (Notredame & Higgins, 1996)).
POA is an algorithm that represents multiple sequences as multiple DAGs and arranges them. POA runs in polynomial time and is considered a generally efficient method that produces good results for complex sequence families. Figure 4 shows the strategy of POA. POA redraws each sequence as a linear series of nodes connected by a single incoming edge

and a single outgoing edge (Fig. 4b.). By using a score matrix that contains similarity values between letters, POA aligns two sequences by dynamic programming that finds maximum similarity (Fig. 4c.). The aligned and identical letters are then fused as a single node, while the others are represented as separate nodes.

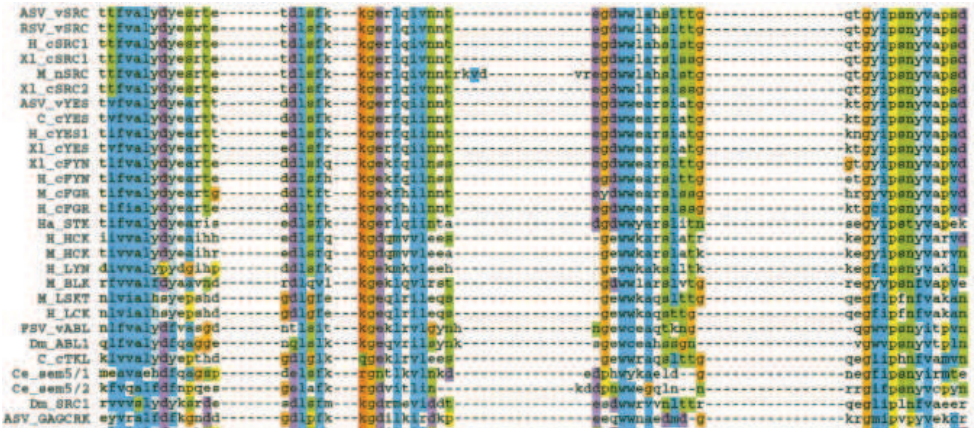

Fig. 3. Example of multiple sequence alignment (MSA) by CLUSTALW (Thompson et al., 1994)

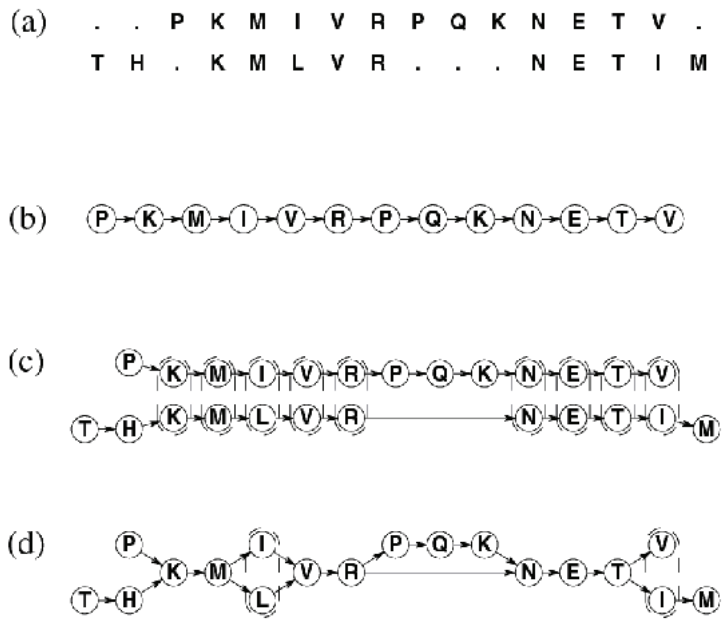

Fig. 4. MSA representation by partial order alignment (POA) algorithm. (a) General representation of MSA, (b) Single representation of POA, (c) Two sequences aligned by POA algorithm, and (d) Aligned result of POA

4.2 Generating script variations

If we regard all scripts as sequences of POA, and all actions and states as nodes or letters of POA, the scripts can be aligned by POA. For example, the parts in two scripts will be aligned as shown in Fig. 5.

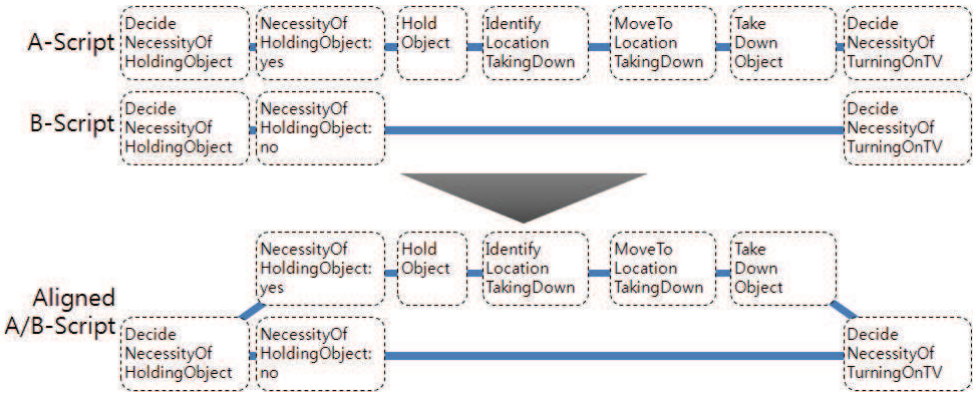

Fig. 5. Two scripts, A and B, are aligned as a form of directed acyclic graph (DAG)

The arranged DAG (ADAG) produced by the POA algorithm allows the generation of script variations and evaluation of script representativeness. Hence, we established a model to generate script variations by using ADAG. This approach attempts to maintain the semantic naturalness of scripts by employing sequential patterns in the given scripts.

The mechanism is as follows: the ADAG of the given scripts is produced by POA. All the paths on the ADAG are searched by employing the "breadth-first" method. New paths that do not have the given sequences still remain, while some deficient scripts are eliminated.

Although script variations are produced by ADAG, there can be deficiencies in some new script variations. Deficiencies in scripts are inspected by using two types of information. One is the basic state relationship of availability. We can predefine each action's preconditions and effects; they are generally domain-independent. For example, a precondition of an action "shifting something down" may be a state wherein the robot is holding the object. By using the information on these states, the model checks whether there is any action that is not satisfied under its preconditions.

The other is user-defined action relationship rules. Any user can pre-describe sequential or associational rules between several actions. Kwon et al. (2008) developed a support system that automatically finds some frequently associative or sequencing actions to aid users to find the action relationship rules. An example of action relationship rules is that a TV channel should not be changed before turning on the TV.

4.3 Evaluating representativeness of scripts

There are paths on which many scripts are overlapped as well as paths on which only one or two scripts are related to the paths on the ADAG. It is possible to link the paths on which many scripts are overlapped; Lee (2003) called the linked paths the consensus sequences of POA (Fig. 6.).

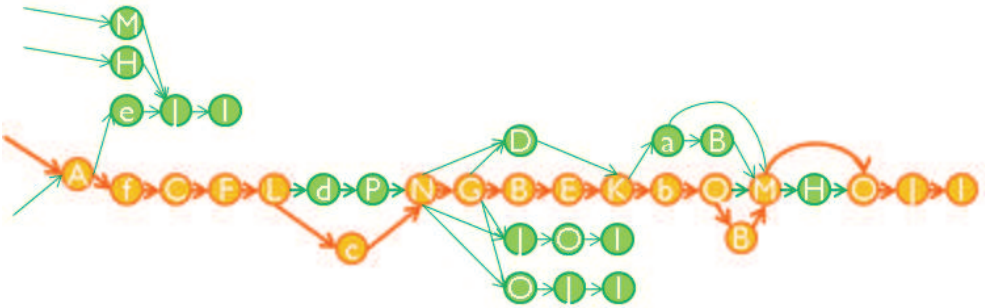

Fig. 6. An example of ADAG and consensus sequence generated from scripts for "greeting user" scenario

The heaviest bundling strategy for discovering consensus sequences is as follows: There are edges between all the actions or nodes on ADAG. The heaviest bundle model attaches 1 *edge_weight* of sequences to every edge on the DAG, and adds each number of aligned edges where two or more sequences are aligned. In such a case, every edge on the DAG has one or more *edge_weight*. While traversing from start nodes to end nodes, the heaviest bundle algorithm finds a path that has the largest sum of *edge_weight* among all paths. The algorithm uses a dynamic traversal algorithm, and an example is shown in Fig. 7. After excluding sequences that contribute to prior consensus generation, the algorithm iterates the consensus generation. Further, the algorithm calculates how many actions are identical to the actions of consensus sequence. We set the exclusion threshold such that scripts coincide over the threshold percentage and the consensus sequences are excluded from each iteration. Iteration continues until no contributed sequence is found.

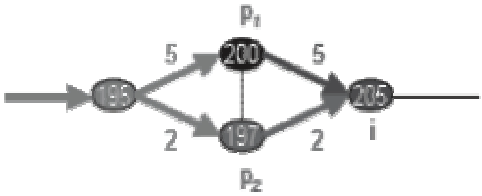

Fig. 7. Dynamic programming for construction of consensus sequences (Lee, 2003)

The representativeness of a script is calculated by computing how the script coincides with the consensus sequences. The equation of representativeness is given as follows:

$$\text{Representativeness} = Coin^{i}\left(Thres\right)^{iteration}, \tag{1}$$

Where i is the index of script; Coin, coincidence variable; and Thres, threshold variable. Coincidence is the number of actions that are identical to those of consensus sequence divided by the total number of actions. Threshold and iteration imply the threshold percentage and the number of iterations in the heaviest bundle algorithm. For example, when the threshold is 80% and a script has ten actions, the script whose nine actions are identical to those of the generated consensus sequence at first iteration has a

representativeness value of 0.72(0.9*0.8). If eight actions are the same with the second consensus sequence, the script has a representativeness value of 0.512(0.8*0.8^2).

5. Implementation

To examine the effectiveness of the proposed methodologies, we implemented the algorithms on a set of scripts. We wrote 160 scripts for a "greeting user" task. In the script database, a robot greets a user and suggests many services such as turning on the TV, delivering something, or reporting house status. There are not only very busy scripts but also simple ones. We established a score matrix in which POA scores only identical actions. The system produced 400 new scripts. Two hundred and fifty of them were meaningfully acceptable, for example, the ones human wrote, and the others were eliminated by deficiency inspectors.
We also re-evaluated the representativeness of approximately 160 scripts. Every script was given a value ranging from zero to a positive one. We then added a wrong script in which two actions were inverted from a script having the positive value. The wrong script's representativeness was 0.7, which is lower than that of the original one.

6. Conclusion

The demand for programming systems that do not require complex programming skills to perform tasks is increasing, especially in the case of programming by demonstration. Further, in the manual programming environment, which is more efficient than programming by demonstration, two critical sub-optimalities are present. We applied POA and heaviest bundling to solve the two problems and implemented the applied algorithms. To prevent the problem of writers omitting combinational procedures, an algorithm for script variation generation was proposed. Further, to evaluate how a specific script is semantically acceptable, an automatic evaluation process of representativeness was established. The evaluation of representativeness is a good attempt to estimate the script's naturalness. However, this evaluation only demonstrates that a good script has a high representativeness value; it does not show that a script having a low representativeness value is unnatural. It is still not easy to automatically maintain the semantic naturalness of task plans or evaluate them. We expect that interactive systems that are not only intelligent but also convenient to users will be continuously developed in the future; this is a promising future research direction.

7. Acknowledgement

This work was supported by the Industrial Foundation Technology Development Program of MKE/KEIT. [2008-S-030-02, Development of OPRoS(Open Platform for Robotic Services) Technology].

8. References

Argall, B.D.; Chernova, S.; Veloso, M. & Browning B. (2009). A survey of robot learning from demonstration, *Robotics and Autonomous Systems*, Vol. 57, No. 5, 469-483, 0921-8890

Biggs, G. & MacDonald, B. (2003). A survey of robot programming systems, *Australasian Conference on Robotics and Automation*, Australia, 2003, Brisbane

Breazeal, C.; Brooks, A.; Gray, J.; Hoffman, G.; Kidd, C.; Lieberman, J.; Lockerd, A. & Mulanda, D. (2004). Humanoid robots as cooperative partners for people, *International Journal of Humanoid Robotics*, Vol. 1, No. 2, 1-34, 0219-8436

Brudno, M.; Chapman, M.; Gottgens, B.; Batzoglou, S. & Morgenstern, B. (2003). Fast and sensitive multiple alignment of large genomic sequences, *BMC. Bioinformatics*,Vol. 4, No. 66, 1471-2105, 1-11

Chen, J. & Zelinsky, A. (2003). Programing by demonstration: Coping with suboptimal teaching actions, *The International Journal of Robotics Research*, Vol. 22, No. 5, 299-319, 0278-3649

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput, *Nucleic Acids Research*, Vol. 32, No. 5, 1792-1797, 0305-1048

Ekvall, S.; Aarno D. & Kragic D. (2006). Task learning using graphical programming and human demonstrations, *Robot and Human Interactive Communication*, UK, Sept., 2006, Hatfield

Ekvall, S. & Kragic, D. (2008). Robot Learning from Demonstration: A Task-level Planning Approach, *International Journal of Advanced Robotic Systems*, Vol. 5, No. 3, 1729-8806

Friendlyrobotics, Robomow, http://www.friendlyrobotics.com/robomow/

Hasbro, i-dog, http://www.hasbro.com/idog/

Husqvarna, Automower, http://www.automower.com

Jones, J. L. (2006). Robots at the tipping point: the road to iRobot Roomba, *IEEE Robotics & Automation Magazine*, Vol. 13, No. 1, 76-78, 1070-9932

Kim, Y.C.; Yoon, W.C.; Kwon, H.T. & Kwon, G.Y. (2007). Multiple Script-based Task Model and Decision/Interaction Model for Fetch-and-carry Robot, *The 16th IEEE International Symposium on Robot and Human interactive Communication*, Korea, August, 2008, Jeju

Knoop, S.; Pardowitz, M & Dillmann, R. (2008). From Abstract Task Knowledge to Executable Robot Programs, *Journal of Intelligent and Robotic Systems*, Vol. 52, No. 3-4, 343-362, 0921-0296

Kwon, G. Y.; Yoon, W. C., Kim, Y. C. & Kwon, H. T. (2008). Designing a Support System for Action Rule Extraction in Script-Based Robot Action Planning, *Proceedings of the 39nd ISR(International Symposium on Robotics)*, Korea, October, 2008, Seoul

Lee, C. (2003). Generating consensus sequences from partial order multiple sequence alignment graphs, *Bioinformatics*, Vol. 19, No. 8, 999-1008, 1367-4803

Lee, C.; Grasso, C. & Sharlow, M. F. (2002). Multiple sequence alignment using partial order graphs, *Bioinformatics*, Vol. 18, No. 3, 452-464, 1367-4803

Lego (2003). Lego Mindstorms, http://mindstorms.lego.com/Products/default.aspx

Nicolescu, M. N. & Matari´c, M. J. (2003). Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice, *In Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Australia, July, 2003, Melbourne

Notredame C.; Higgins D.G. & Heringa J. (2000). T-Coffee: A novel method for fast and accurate multiple sequence alignment, *Journal of Molecular Biology*, Vol. 302, No. 1, 205-217, 0022-2836

Notredame, C. & Higgins, D. G. (1996). SAGA: sequence alignment by genetic algorithm, *Nucleic Acids Research*, Vol. 24, No. 8, 1515-1524, 0305-1048

Pardowitz, M.; Zollner, R. & Dillmann, R. (2005). Learning sequential constraints of tasks from user demonstrations, *IEEE-RAS International Conference on Humanoid Robots*, Japan, December, 2005, Tsukuba

Takara, Tera robot, http://plusd.itmedia.co.jp/lifestyle/articles/0501/20/news030.html

Thompson, J. D.; Higgins, D. G. & Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties, *Nucleic Acids Research*, Vol. 22, No. 22, 4673-80, 0305-1048

Zhang, H.; Zhang, J.; Zong, G.; Wang, W. & Liu R. (2006). SkyCleaner3: a real pneumatic climbing robot for glass-wall cleaning, *IEEE Robotics & Automation Magazine*, Vol. 13, No. 1, 32-41, 1070-9932

Human-Robot Interaction

Edited by Daisuke Chugo

ISBN 978-953-307-051-3
Hard cover, 288 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Human-robot interaction (HRI) is the study of interactions between people (users) and robots. HRI is multidisciplinary with contributions from the fields of human-computer interaction, artificial intelligence, robotics, speech recognition, and social sciences (psychology, cognitive science, anthropology, and human factors). There has been a great deal of work done in the area of human-robot interaction to understand how a human interacts with a computer. However, there has been very little work done in understanding how people interact with robots. For robots becoming our friends, these studies will be required more and more.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yo Chan Kim and Wan Chul Yoon (2010). Handling Manually Programmed Task Procedures in Human–Service Robot Interactions, Human-Robot Interaction, Daisuke Chugo (Ed.), ISBN: 978-953-307-051-3, InTech, Available from: http://www.intechopen.com/books/human-robot-interaction/handling-manually-programmed-task-procedures-in-human-service-robot-interactions

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821