2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

# Automatic Extraction of Command Hierarchies for Adaptive Brain-Robot Interfacing

Matthew Bryan*,‡, Griffin Nicoll†, Vibinash Thomas†, Mike Chung*, Joshua R. Smith†,‡, Rajesh P. N. Rao*,‡
*University of Washington Neural Systems Lab, Seattle, USA
†University of Washington Sensor Systems Lab, Seattle, USA
‡Member, IEEE
Email: mmattb@cs.washington.edu, vibintho@uw.edu, rgnicoll@uw.edu,
mjyc@cs.washington.edu, jrs@cs.washington.edu, rao@cs.washington.edu

***Abstract*— Recent advances in neuroscience and robotics have allowed initial demonstrations of brain-computer interfaces (BCIs) for controlling wheeled and humanoid robots. However, further advances have proved challenging due to the low throughput of the interfaces and the high degrees-of-freedom (DOF) of the robots. In this paper, we build on our previous work on Hierarchical BCIs (HBCIs) which seek to mitigate this problem. We extend HBCIs to allow training of arbitrarily complex tasks, with training no longer restricted to a particular robot state space (such as Cartesian space for a navigation task). We present two algorithms for learning command hierarchies by automatically extracting patterns from a user's command history. The first algorithm builds an arbitrary-level hierarchical structure (a "control grammar") whose elements can represent skills, whole tasks, collections of tasks, etc. The user "executes" single symbols from this grammar, which produce sequences of lower-level commands. The second algorithm, which is probabilistic, also learns sequences which can be executed as high-level commands, but does not build an explicit hierarchical structure. Both algorithms provide a *de facto* form of dictionary compression, which enhances the effective throughput of the BCI. We present results from two human subjects who successfully used the hierarchical BCI to control a simulated PR2 robot using brain signals recorded non-invasively through electroencephalography (EEG).***

## I. INTRODUCTION

Using humanoid robots to perform remote tasks with human supervision has been a subject of considerable interest in the robotics community[1], [2], [3]. Humanoid robots are often considered proxies or assistants to humans, performing tasks in both real-world environments designed for humans and in environments considered too dangerous for humans.

One method of controlling an assistant robot is the brain computer interface. This method of control has particular appeal since it could allow a severely paralyzed or "locked in" patient to manipulate their external environment via a robotic avatar, allowing them to gain some amount of independence [4], [5], [6].

However, this endeavor has proven challenging due to the low throughput of traditional non-invasive BCIs and the high DOF of the robot. The BCI's low signal-to-noise ratio means that reliable information can be extracted from the brain only over long time intervals. At the same time, the high DOF of the robot means a large amount of information needs to be extracted over short intervals of time. We therefore propose that a BCI/robot system should provide a high-level interface which summarizes the information necessary for full control of the robot (e.g. presenting an interface for a grasping routine where the BCI user only selects an object to be grasped). At the same time, it must also balance this high-level summary with the flexibility to achieve the tasks desired by the BCI user [7].

Previous BCIs for robot control have relied on a static set of pre-programmed behaviors. This allows the user to have some basic control over the robot, but such an approach is inherently inflexible. An alternate approach is to control the robot using low-level moment-by-moment commands but the low throughput of the interface means accomplishing even basic tasks is tedious and time consuming. The cognitive load imposed by this tedium makes such BCI/robot systems impractical for real-world use in accomplishing complex tasks.

We previously proposed *hierarchical BCIs (HBCIs)* [8], [9], [10] which address this problem by allowing the user to teach the robot new skills using individual, relatively low-level commands. Since we allow the user to execute these low-level commands, the system can provide enough flexibility for the user to perform her/his desired tasks. At the same time, a newly-taught skill can later be executed with a single command, effectively increasing the throughput of the interface and relieving the user of some amount of lower-level control.

This previous work on HBCIs did not allow arbitrary-levelled skill hierarchies. In [8], [9], and [10] we explored the benefits of a two-level hierarchy: low-level navigational commands and trained high-level trajectory commands. In [11] we trained gripper trajectories and allowed the user to add a third level to this hierarchy by explicitly specifying a sequence of commands that represent a larger task. With restrictions on the height of the command hierarchy the user was able to specify individual skills but not larger, more complex tasks or sets of tasks. Also, these previous systems restricted learning to particular state spaces and thus did not allow for training of more general tasks. As a result, the throughput gains were limited.

In this paper, we compare two algorithms which automat-

This research was supported by ARO Award no. W911NF-11-1-0307, NSF award no. 0930908, the Office of Naval Research (ONR), and a Mary Gates Research Scholarship awarded to MB.

ically learn command hierarchies from the user's command history: recurring patterns are recognized and made available as new high-level abstract commands. This approach provides a method for training new robot skills without tying a learning algorithm to a specific state space, thus allowing the user to specify skills involving any number of spaces. Moreover, the fact that learning is based on recurring patterns means that the noise in low-level control is likely to be ignored.

The first algorithm extracts an arbitrary-levelled hierarchical skill structure, which we call a "control grammar." This structure contains both lower-level commands and higher-level commands such as whole skills, tasks, sets of tasks, etc. It is extracted from the user's command history using the Sequitur (Nevill-Manning) algorithm. This algorithm infers a context-free grammar from a sequence of discrete symbols [12]. The symbols in this grammar ultimately produce a set of lower-level commands. Non-terminal symbols in this case represent commonly occurring patterns of terminal symbols or other non-terminal symbols. These non-terminal symbols represent higher-level skills such as a whole task. By identifying recurring sequences of commands/symbols, we take advantage of patterns in the user's command history.

For illustration, suppose the HBCI observed the user's command history to be "a,1,b,c,a,2,b,c,b,3,a,c,b,4,a,c,a,2,b,c,b,3,a,c" where each given letter is an arbitrary lower-level command (this is adapted from one of our subject's results). Sequitur in this case would return the following output:

- S0: a,1,S1,S2,b,4,S3,S2
- S1: b,c
- S2: a,2,S1,b,3,S3
- S3: a,c

In this case, all S# symbols are non-terminal symbols - sequences of other symbols. We present the non-terminal symbols in this vocabulary, which represent higher-level commands, to the user for execution, with some exceptions (see System Architecture). If the user were to "execute" S2, this symbol would be parsed into lower-level commands and the robot would receive the command string "a,2,b,c,b,3,a,c."

The second algorithm extracts maximum-length sequences which appear multiple times. It begins with all observed sequences and recursively grows the sequences longer if, given the observed data and the sequence built thus far, the next symbol to appear can be inferred with relatively high probability. Once this process is complete, the remaining sequences are eliminated using a set of filters (see System Architecture). Unlike Algorithm 1, Algorithm 2 does not build sequences into an explicit hierarchy. However, it imposes no restriction on whether sequences can overlap in any way. This means Algorithm 2 can represent sub-tasks, tasks, etc., just as Algorithm 1, but will not organize them in the same way. Using the same example as with Algorithm 1, Algorithm 2 would return only one result after filtering, roughly equivalent to S2 above: "c,a,2,b,c,b,3,a,c."

As with our previous HBCI proposals, the use of these algorithms increases the effective throughput of the HBCI.

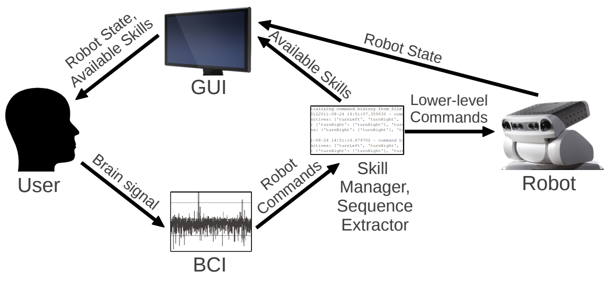

Fig. 1. Overview of system architecture

Other forms of communication use similar methods to increase effective throughput (e.g., compression in modern networks). Compression increases effective throughput by encoding messages into forms that occupy a smaller data space; these encoded messages are transmitted through the communications medium and can be decoded by the receiver into a form which is the same or similar to the original, larger message. As a result, the effective throughput of the communications device is increased. Similarly, the HBCI encodes whole sequences of commands into single commands which can be decoded and sent to the robot for execution, increasing the effective throughput of the system.

Our command sequence extraction algorithms share similarities with auto-completion algorithms used in text interfaces such as search engines and "texting" apps for mobile platforms (e.g. [13], [14], [15]). These auto-completion algorithms observe partially entered data and attempt to match them to patterns observed in an existing corpus such as a dictionary or a set of usage data. Once such matches have been made, they are suggested to the user as a way of automatically completing the data entry task. This raises the effective throughput of the text interface by reducing the number of keystrokes necessary. Likewise, our HBCI design tracks user data and makes observed patterns available as single commands, allowing the user to abbreviate their input.

The framework we propose can be applied to any complex robotic task where there are repeated (and possibly nested) sub-tasks which can occur in different orders. For the purposes of illustration, we use a simple task that involves mixing a set of ingredients to obtain a final product (e.g., a drink). We give the user recipes for making drinks and the user must use a simulated PR2 robot to make them. Low-level commands include turning the robot, grasping an ingredient, and pouring it into a mixture. Since recipes will share ingredients, patterns can be expected to emerge as soon as multiple recipes have been mixed. From that point forward, the user will be able to complete the task again using fewer commands. This process repeats four times. We show that as more data is collected (the size of the command history increases), the corresponding increase in recognized patterns (sequences) leads to a drastic increase in the effective throughput of the HBCI.

## II. SYSTEM ARCHITECTURE

As in our previous work, this HBCI's architecture consists of three modular elements:

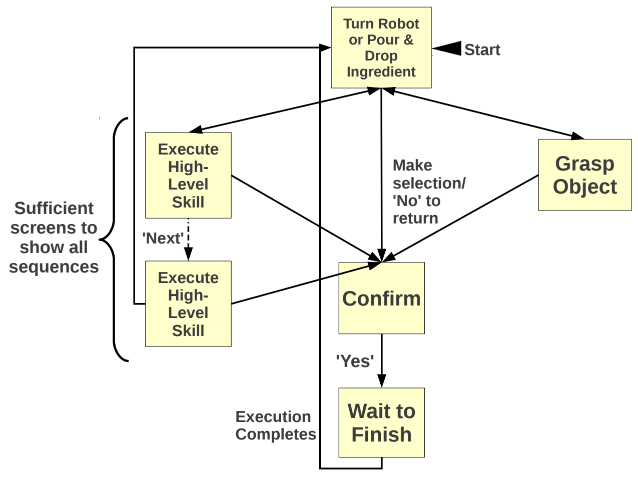

Fig. 2. Overview of menu system flow. Each node represents a screen presented to the user. Black triangle indicates starting screen.

1) A low-throughput control interface which receives noisy input from the brain, in this case a BCI based on Steady State Visually-Evoked Potentials (SSVEP) [16]. Other BCI paradigms with discrete classification could be used as well;
2) A hierarchical and adaptive menu which displays the robot's current state and the list of available skills;
3) A high DOF robot.

### A. SSVEP-based BCI

We use the same EEG-setup and classification process as in [11]. Please refer to it for details.

### B. Robot Simulation

We use the Gazebo robot simulator with a model of the PR2 semi-humanoid robot. Other grasping robots could also be used. To simplify, we use hard-coded poses to turn the robot in place, grasp, and pour rather than using more dynamic action mechanisms (e.g. collision avoidance). Grasping in this case is based on assumed positions of the ingredients. While this approach is limiting, we use it only for simplification. The HBCI does not preclude the use of true object detection and recognition with dynamic grasping or other, more robust, forms of control.

### C. Hierarchical Adaptive Menu

We use a menu system with five command options per screen. The initial screen displays lower-level commands such as turning the robot, etc. It also presents two additional commands: one to transition to the "Grasping" screen, and one to the "Sequences" screen (Fig. 2).

The Grasping screen allows the user to select an object currently in view and to grasp that object. This menu labels the graspable objects visually and allows the user to select them by their labels. An additional command allows return to the initial screen.

The Sequences screen displays all higher-level skills currently recorded in the Skill Manager. Sufficient screens to display all of them can be accessed by selecting 'Next.' The user selects a higher-level command for execution using these screens. Upon doing so, the command is parsed into a string of lower-level commands to be sent to the robot. The screen displays the sequences by listing the lower-level commands into which they will be parsed. While this may impose some burden on the user who must read the command list and understand it, it nevertheless appears the users in our experiment found it easy to use for our simple test. Moreover, more sophisticated alternatives likely exist. For instance, a visual simulation of a sequence's effect may allow the user to have a better understanding of what a sequence would do.

The menu system displays the Confirmation screen after the selection of a command for execution. The user can select 'No,' which will return the user to their previous position in the menu. Selecting 'Yes' sends the command to the robot. During execution, the user sees a Wait screen where they can observe the execution of the command, but cannot make any commands until it completes. For the sake of system safety, one could allow for a "Cancel" command on this screen. However, for the sake of simplicity we did not implement that for these experiments. Once the robot completes the task, control returns to the user.

### D. Skill Management, Sequence Extraction

The Skill Manager tracks all currently known robot skills, command history, and robot state. It also filters the results of the command sequence extraction process. Filtering is used here to increase the efficiency of the system.

With the first filter for the Sequitur algorithm, we exclude symbols which appear only once in the command history, which is always the entire command history itself (S0 in the example above). By filtering to include only symbols which appear multiple times, we make it more likely the user will want to use the symbols which are displayed. This prevents the user from having to manually filter through many superfluous choices. Second, we exclude any symbols which parse into fewer than three terminal symbols. Given that the user must look through a list of available commands to execute, and that the user must confirm each selection they make, the presentation of symbols of length two does not net any throughput gains to the HBCI system.

With the Max Chaining algorithm, we filter short and infrequent sequences in the same way. Additionally, remaining sequences which appear as the right-hand elements in a longer sequence are removed (expanded below).

### E. Algorithm 1: Sequitur algorithm

The Sequitur algorithm works by generating non-terminal symbols over patterns which it observes in a sequence of discrete terminal symbols [12]. It scans the sequence, and as it observes patterns, it replaces them with single non-terminal symbols. It inserts the non-terminal symbols in a rule table. This table is similar to the dictionary in a dictionary compression algorithm. Each non-terminal symbol definition may contain other non-terminal symbols, thus creating a hierarchical structure. The encoded original sequence is also added to the table as a top-level rule (labelled 'S0' above). The final structure constitutes a context-free "control grammar."

Using this algorithm over the user's command history can be likened to automatic task identification and decomposition. If a task appears multiple times in the user's command history, Sequitur may identify it and label it with a non-terminal symbol. Likewise, if a sub-task appears across multiple tasks, it may be labelled and the tasks of which it is a part will be partly decomposed to include it. In this sense, the use of the Sequitur algorithm allows for automatic learning of an arbitrary-levelled control hierarchy.

### *F. Algorithm 2: Maximum-length chaining*

This algorithm begins with all observed sequences and sub-sequences. If a given symbol always or nearly always follows the sequence in the observed history, it is appended on the end, the shorter sequence is deleted, and the process is repeated again. In this case "nearly always" is defined by a probability threshold, which we set to 0.75. This can be likened to our previous work in [10] where the robot would execute a policy until it no longer had sufficient confidence to proceed.

To illustrate, consider the input sequence "a,b,c,1,a,b,c,2." Max Chaining begins with all observed length-2 sequences and determines the most likely postfix to them. For instance, "a,b" is always followed by "c," which is the most likely postfix. If the most likely postfix appears with at least 0.75 probability, we discard the shorter sequence, "a,b" in this case, and form a length-3 sequence, "a,b,c." Next we observe that "a,b,c" is not followed by any single postfix with probability at least 0.75, so we cannot extend it further. We move on to "a,b,c,1" and "a,b,c,2" and attempt to extend them further. This process continues until we consume all observed sequences, Once this completes we have this remaining list:

- a,b,c
- b,c,2
- a,b,c,2
- 1,a,b,c,2
- c,1,a,b,c,2
- b,c,1,a,b,c,2
- a,b,c,1,a,b,c,2

Finally, we apply our filters, which eliminate all but "a,b,c," which becomes our only output.

Unlike Sequitur, this algorithm does not create a hierarchical structure. However, it has no restrictions on the sequences which it can identify, other than the filtering rules, and it therefore could include the same list of sequences as the Sequitur algorithm. Also, though it does not generate a hierarchical structure, the user experiences this algorithm in the same way as the other since in both cases the user sees only the sequences generated and not the underlying structure itself.

Since this algorithm incorporates some probabilistic elements, it may be more robust than Sequitur with respect to noisy control. If a given control sequence includes some amount of noise, this algorithm will identify that the noisy form is an approximation of another form, provided that sufficient clean examples appear elsewhere in the user's command history.

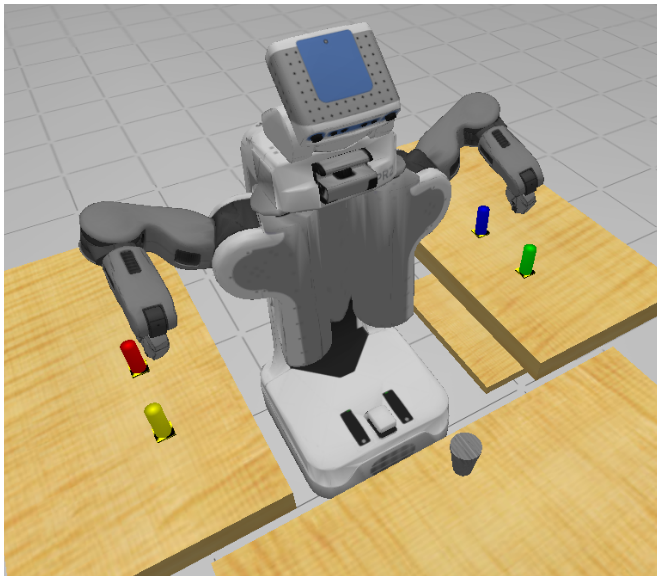
Fig. 3. Overview of the experimental environment

This algorithm makes a trade-off between over-fitting to past experience and quickly chaining whole tasks together. By pruning extend-able sequences, Max Chaining may lose some sub-sequences which the user could find useful in the future. However, this is balanced by tending towards longer sequences, which may make the algorithm tend towards larger throughput gains if the user does not need the discarded sub-sequences. For BCI-in-a-home applications, it may make sense to compress longer sequences since the user will most likely be following lengthy daily routines which the system can learn quickly.

## III. EXPERIMENT

### *A. Human Subjects*

Two able-bodied male subjects (ages: 20, 27) participated in the study, which was approved by the University of Washington Institutional Review Board. All subjects gave written informed consent.

We gave each subject two drink recipes, which they were to mix using the PR2 robot simulation controlled using the HBCI. We placed four ingredients for these recipes around the robot in predefined locations (Fig. 3). We labelled the ingredients 'Blue', 'Green', 'Yellow', and 'Red.' Each recipe consisted of three ingredients. The two recipes contained at least one different ingredient than the other.

In front of PR2 sat a table with a container in which the recipe was to be mixed. Ingredients sat on tables to the left and right of PR2. We programmed the PR2 to have the following lower-level commands:

- Rotate PR2 left or right 90 degrees
- Pour the ingredient currently in the right hand and discard the empty container
- Grasp left or right object on the table

Each subject performed two experiments: one for each algorithm. Each experiment took place over four phases.

During each phase the user was to mix both recipes once. The order in which the recipes were mixed varied from one phase to the next in order to induce more variation in the user's command history. Phase 1 was performed only once and was shared across experiments since it did not include the execution of any extracted sequences. It involved only lower-level commands. We gave the user the option to perform these phases across multiple days rather than in one sitting.

In the second phase, the user mixed the same two recipes again, but in reverse order. This reverse order ensured that the user had to select each recipe separately in phase three, allowing us to demonstrate the full use of the control hierarchy. The HBCI carried the command history over from the user's previous session. The sharing of ingredients between the two recipes made it likely that some repetition occurred in the command history coming into phase 2. As a result, the control grammars contained some higher-level commands that allowed the user to make the same recipes with fewer commands. The new commands made during this session were appended to the end of the command history as they were made.

In the third and fourth phases, the user once again mixed the same two recipes in the original order. Since each recipe appeared multiple times in the user's total command history, longer patterns tended to emerge in these phases, often representing an entire recipe or both recipes. Once again, the HBCI appended the new commands to the end of the user's command history.

### B. Monte Carlo Simulation

We use human subject testing to demonstrate that this HBCI improves effective throughput in a real-world situation. However, due to resource constraints human subject testing permits only a limited number of tests. We therefore performed more extensive testing of our HBCI framework using a simulated user. This user simulates the general behavior of a human user including some amount of noisy control and occasionally using other sub-optimal sequences of commands. We use the simulated user to repeat the same experiments given to human users for a total of 500 iterations. We repeat this set of experiments twice - once with noise levels equal to the average noise level of the two human subjects, and once with noise level 2.5 time higher than the average.

## IV. RESULTS

In order to measure improvements in effective throughput, we recorded the number of commands issued by the user to accomplish each task rather than directly measuring the average number of bits transmitted per second. While the actual time to complete a task may decrease somewhat as the HBCI identifies more useful sequences, experiment time was largely dominated by the time the robot took to execute commands. As a result, we evaluate throughput increase in terms of the decrease in the number of commands transmitted by the interface to complete the task.

We measure three types of commands made. The first is "action commands" i.e., the number of lower-level commands or sequences chosen for execution. We also measure every command issued by the user, including pure menu-related commands. This method is a less direct way of measuring the effect of sequence extraction and command hierarchy learning since it includes noise in user control. Finally, we measure the effective number of commands, including menu commands, but excluding noise that did not result in an action being made. An advantage of this method is that it measures menu design in addition to the number of actual commands made to the robot system. We present all three measurements below, but analyze and simulate only the action command series (See Table I).

### A. Human Subject

Both subjects successfully completed the four phases of both experiments. Subject 1 completed the experiments across three sessions, while Subject 2 completed them across two. In all cases the number of action commands necessary to accomplish the task decreased in later phases. This is a direct result of the increasingly useful high-level commands learned by the HBCI.

While having results from only two subjects prevents us from presenting a robust statistical analysis, Max Chaining appears, at least anecdotally, to have out-performed Sequitur in dealing with Subject 1's noisy input. Subject 1 made two control errors in phase 1, making the initial training data less useful. The commands made by Subject 1 in phase 1 were as follows (bold text represents superfluous commands, the vertical separator divides recipe A from recipe B):

> *TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspYellow, TurnLeft, PourAndDrop, TurnRight, GraspRed, TurnLeft, PourAndDrop,* || ***TurnRight, TurnLeft,*** *TurnLeft, GraspGreen, TurnRight, PourAndDrop, TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspRed, TurnLeft,* ***TurnLeft, TurnRight,*** *PourAndDrop*

While Sequitur nevertheless extracted useful sequences from this data, its results were "fragmented" relative to Max Chaining, which was specifically designed to create long sequences. This fragmentation meant some sequences appeared in more parts and that some sequences had an omitted beginning or ending portion relative to Max Chaining. This directly contributed to a longer phase 3 for Subject 1 while using Sequitur (see Table. I). To compare, observe Subject 1's extracted sequences for the two algorithms going into phase 3; these are the sequences presented to the user after filtering and parsing to lower-level commands. The first and second sequences returned by Max Chaining represent all or nearly all of the commands necessary to mix whole recipes:

Algorithm 1 - Sequitur:

1) *TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspYellow*
2) *TurnLeft, PourAndDrop, TurnRight*

TABLE I

NUMBER OF COMMANDS AS A FUNCTION OF PHASE AND EXPERIMENT

| | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| **Action Commands** | | | | |
| Subject 1 Sequitur | 28 | 15 | 9 | 2 |
| Subject 1 Max Chaining | 28 | 13 | 3 | 2 |
| Subject 2 Sequitur | 24 | 10 | 2 | 1 |
| Subject 2 Max Chaining | 24 | 8 | 2 | 1 |
| **Action + Menu Commands** | | | | |
| Subject 1 Sequitur | 62 | 37 | 22 | 6 |
| Subject 1 Max Chaining | 62 | 34 | 8 | 6 |
| Subject 2 Sequitur | 56 | 24 | 6 | 3 |
| Subject 2 Max Chaining | 56 | 20 | 6 | 3 |
| **All Commands** | | | | |
| Subject 1 Sequitur | 72 | 57 | 34 | 6 |
| Subject 1 Max Chaining | 72 | 80 | 8 | 11 |
| Subject 2 Sequitur | 80 | 32 | 10 | 3 |
| Subject 2 Max Chaining | 80 | 28 | 10 | 3 |

3) *GraspGreen, TurnRight, PourAndDrop, TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspRed*

Algorithm 2 - Max Chaining:

1) *TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspYellow, TurnLeft, PourAndDrop, TurnRight, GraspRed, TurnLeft, PourAndDrop*
2) *TurnLeft, GraspGreen, TurnRight, PourAndDrop, TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight, GraspRed, TurnLeft*
3) *PourAndDrop, TurnLeft, GraspBlue, TurnRight, PourAndDrop, TurnRight*
4) *TurnLeft, PourAndDrop, TurnRight*

In this particular test, it appears that Max Chaining's tradeoff between closely fitting the training data and discarding potentially useful sub-sequences was not problematic. However, without greater statistical evidence, we cannot conclude with confidence that Max Chaining out-performs Sequitur in this context.

Subject 2 had near-optimal control during all phases. This meant the user did not deviate from the optimal control sequence. In this case, Algorithms 1 and 2 returned nearly identical results in all phases, generating similar command counts. In phase 2, the user encountered sequences representing significant portions of each recipe. In phase 3, the user encountered two sequences, one representing the command for each recipe. In phase 4, a single command could be used to complete the entire task.

### *B. Monte Carlo Simulation*

The data from conducting the same experiments with a simulated user generally harmonize with the results of human subject testing. Both algorithms significantly lowered the number of commands necessary to accomplish the assigned task.

These simulation results support the conclusion that Max Chaining out-performs Sequitur in the presence of noisy control (See Table II). We applied a one-sided two-sample t-test to the experimental results in order to test the probability that the Max Chaining and Sequitur results are drawn from the same distribution. Once again, phase 1 did not involve the use of the algorithms, but noisy control was still present. In the average noise case, the Max Chaining experiment happened to experience more noise than Sequitur. Despite this, the simulated user was able to use fewer commands by phase 4 than Sequitur ($P<1\%$). With higher noise applied, the Max Chaining experiment happened to experience less noise than the Sequitur experiment in phase 1. Once again, Max Chaining out-performed Sequitur ($P<1\%$).

TABLE II

AVERAGE NUMBER OF ACTION COMMANDS BY SIMULATED USER AS A FUNCTION OF PHASE AND EXPERIMENT (STD)

| | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| Sequitur - Avg Noise | 26.39 (1.60) | 13.25 (4.46) | 5.00 (3.00) | 3.24 (2.77) |
| Max Chaining - Avg Noise | 26.48 (1.64) | 13.66 (3.36) | 4.67 (3.19) | 2.42 (2.30) |
| | | | | |
| Sequitur - High Noise | 29.42 (2.92) | 17.01 (5.26) | 8.95 (4.39) | 5.68 (3.61) |
| Max Chaining - High Noise | 29.20 (2.66) | 15.40 (3.71) | 7.83 (3.88) | 4.44 (3.30) |
| | | | | |
| Low Noise - 2 sample t: P | $8.20\times10^{-2}$ | $9.50\times10^{-2}$ | $4.00\times10^{-4}$ | $2.02\times10^{-7}$ |
| High Noise - 2 sample t: P | $1.11\times10^{-1}$ | $1.41\times10^{-8}$ | $1.03\times10^{-5}$ | $1.17\times10^{-8}$ |

## V. CONCLUSIONS

Our results suggest that learning command hierarchies via sequence extraction offers a potentially powerful way of enhancing the effective throughput of a BCI for robot control. In our approach, command sequences are automatically extracted by observing patterns in the user's command history. The result is a form of dictionary compression over lower-level robot commands, increasing the effective throughput of the BCI system. As this system accumulates more usage data, it makes a richer set of identified patterns available to the user, leading to consistently increasing throughput.

We recommend further research in the human-computer interaction aspects of HBCIs. One potential question is determining the best way to present available higher-level skills to a user. How can we help the user understand semantically what a higher-level skill does without simply presenting lists of lower-level commands? Also, how can commands be connected to the state space so that only immediately relevant commands are presented?

Our ongoing work focuses on additional ways of circumventing the low throughput issue for robust and efficient brain-robot interfacing:

1) Utilizing more sophisticated machine learning and probabilistic reasoning algorithms to handle the uncertainty and noise inherent in brain-based robotic control;
2) Augmenting brain signals with other signals such as eye movement, voice, and muscle-based commands to explore the full-range of human biological control of robots.

## REFERENCES

[1] T. Nishiyama, H. Hoshino, K. Sawada, Y. Tokunaga, H. Shinomiya, M. Yoneda, I. Takeuchi, Y. Ichige, S. Hattori, and A. Takanishi, "Development of user interface for humanoid service robot system,"

in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 3. IEEE, 2003, pp. 2979–2984.

[2] R. Ambrose, H. Aldridge, R. Askew, R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark, "Robonaut: Nasa's space humanoid," *Intelligent Systems and their Applications, IEEE*, vol. 15, no. 4, pp. 57–63, 2000.

[3] A. Meltzoff, R. Brooks, A. Shon, and R. Rao, "Social robots are psychological agents for infants: A test of gaze following," *Neural Networks*, vol. 23, no. 8-9, pp. 966–972, 2010.

[4] C. Bell, P. Shenoy, R. Chalodhorn, and R. Rao, "Control of a humanoid robot by a noninvasive brain–computer interface in humans," *Journal of Neural Engineering*, vol. 5, p. 214, 2008.

[5] Y. Chae, S. Jo, and J. Jeong, "Brain-actuated humanoid robot navigation control using asynchronous brain-computer interface," in *Neural Engineering, 2011. NER'11. 5th International IEEE/EMBS Conference on*. IEEE, 2011.

[6] J. Millan, F. Renkens, J. Mouriño, and W. Gerstner, "Noninvasive brain-actuated control of a mobile robot by human EEG," *Biomedical Engineering, IEEE Transactions on*, vol. 51, no. 6, pp. 1026–1033, 2004.

[7] O. C. Jenkins, "Sparse control for high-DOF assistive robots," *Intelligent Service Robotics*, vol. 1, no. 2, pp. 123–134, Apr 2008. [Online]. Available: http://www.cs.brown.edu/~cjenkins/papers/cjenkins_isr2007.pdf

[8] M. Chung, W. Cheung, R. Scherer, and R. Rao, "Towards hierarchical bcis for robotic control," in *Neural Engineering, 2011. NER'11. 5th International IEEE/EMBS Conference on*. IEEE, 2011.

[9] ——, "A hierarchical architecture for adaptive brain-computer interfacing," in *Twenty-Second International Joint Conference on Artificial intelligence (IJCAI11)*, Barcelona, Spain, July 2011.

[10] M. Chung, M. Bryan, W. Cheung, R. Scherer, and R. Rao, "Interactive hierarchical brain-computer interfacing: Uncertainty-based interaction between humans and robots," in *Fifth International Brain-Computer Interface Conference 2011 (BCI2011) (to appear)*, Graz, Austria, September 2011.

[11] M. Bryan, J. Green, M. Chung, L. Chang, R. Scherer, J. Smith, and R. Rao, "An adaptive brain-computer interface for humanoid robot control," in *Humanoids, 2011 (Humanoids2011). 11th IEEE-RAS International Conference on (to appear)*. IEEE, 2011.

[12] C. Nevill-Manning and I. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, vol. 7, pp. 67–82, 1997.

[13] D. J. Hachamovitch, R. A. Fein, and E. J. Fries, "Automatic word completion system for partially entered data," Patent US 6 377 965, 04, 2002.

[14] K.-W. Lee and K. E. Wales, "Methods and systems for implementing auto-complete in a web page," Patent US 7 185 271, 02, 2007.

[15] D. J. Ward and D. J. C. MacKay, "Fast hands-free writing by gaze direction," *Nature*, vol. 418, no. 6900, p. 838, 2002. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/abstracts/eyeshortpaper.html

[16] G. R. Müller-Putz and G. Pfurtscheller, "Control of an electrical prosthesis with an SSVEP-based BCI," *Biomedical Engineering, IEEE Transactions on*, vol. 55, no. 1, pp. 361–364, 2007.