A Method to Extend Functionality of Pointer Input Devices

Oleg Gusyatin[1,2], Mikhail Urinson[3], and Margrit Betke[1]

[1] Department of Computer Science, Boston University, Boston, MA 02215
{gusyatin,betke}@bu.edu
http://www.cs.bu.edu/~betke

[2] Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215
gusyatin@cns.bu.edu

[3] Department of Computer Science, Tufts University, Medford, MA 02155
Mikhail.Urinson@tufts.edu

Abstract. We describe a general method for extending any pointer input device with an arbitrary set of commands. The proposed interface can be trained by the user to recognize certain cursor movement patterns and interpret them as special input events. Methods for extraction and recognition of such patterns are general enough to work with low-precision pointing devices, and they can be adjusted to provide computer access for people with disabilities. The core of the system is a trainable classifier, in the current implementation an artificial neural network. The architecture of the neural network automatically adjusts according to complexity of the classification task. The system demonstrated good accuracy and responsiveness during extensive experiments. Some tests included a severely motion-impaired individual.

1 Introduction

As miniature computers, such as personal digital assistants (PDAs), tablet and wearable computers gain popularity, more people face the problem of finding a mobile alternative for the traditional keyboard. While most portable devices offer a comfortable way to perform pointer input, their keyboard substitutes often have significant drawbacks. On-screen keyboard and graffiti symbol recognition software are among the most common keyboard alternatives for portable computers with pointer input. The disadvantage of on-screen keyboard is that it occupies valuable screen space. The graffiti symbol recognition method (sometimes it also employs a special area, but this can be avoided) is usually very inflexible: the user has to learn predefined symbols that might only slightly resemble real letters and digits. Accuracy of recognition and speed of input depends on the user's experience with this symbol language, which will take time to learn. Furthermore, users of traditional desktop and notebook computers, who employ wireless pointing input devices, such as camera-based [1], eye-gaze [9] and gyroscopic mice, often find keyboards, even wireless keyboards, inconvenient, for example, during a presentation. For many tasks, being able to access quickly and accurately even a small number of keys and key combinations could be very helpful. PDAs and advanced mice address this by having a few customizable buttons. Similarly, a small number of commands can be recognized by camera-based interfaces, for example, using eye blinks or eyebrow raises [6].

Some alternative pointer input techniques can be employed by people with disabilities to communicate with the computer. Vision-based human-computer interfaces allow users with limited range of voluntary motions to control the cursor using head, eye, or tongue movements [1]. From a more general point of view, the problem of adding an accessible keyboard to the computer with pointer input is similar to the problem of developing a convenient keyboard under mobility constraints.

To summarize all the above: several groups of people can benefit from an interface that allows to quickly access an arbitrary set of commands (keyboard keys and key combinations, internet browser commands, customizable "shortcut"-like functions, macros) on a computing device designed for pointer input. This interface should require minimum resources (no screen space, no additional hardware, small amount of processor time) and should not demand high precision of cursor control. Finally, rather than asking the user to memorize or practice predefined symbols, the interface should "learn" and adjust to "understand" the user's own commands. This is particularly important to people with severe motion impairments because with the proposed system they can design symbols according to their physical abilities.

In this paper we describe a general mechanism that extends pointer input functionality with an arbitrary set of commands. The core of the system is a trainable classifier, in the current implementation, an Artificial Neural Network (ANN), that is used to recognize user-defined Spatio-Temporal Patterns (STPs) produced by the cursor. The classifier has an adaptive architecture that enables it to automatically and efficiently adjust to accommodate the necessary number of classes. We attempted to make as few assumptions about potential STPs and user abilities and preferences as possible. As a result, the system can be used with different pointer input devices, and it can be modified to provide computer access for people with disabilities.

2 System Overview

The system consists of three major components: a preprocessor, a classifier, and an interpreter as shown in Figure 1. It takes an STP produced by the pointer input device and produces an operating system event.

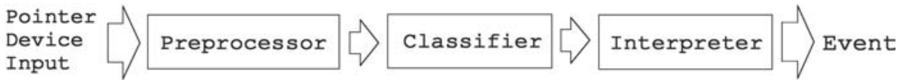

Fig. 1. General System Overview.

In our approach, the collection of STPs used by the system, the alphabet, is completely determined by the user. As a result, the interface is general and can be conveniently used for many different tasks. However, the fact that no a priori information is available about the alphabet and its elements makes the classification problem difficult. In particular, the classifier has to be able to efficiently distinguish between an unknown number of classes of unknown appearance and structure. In addition, we want to allow the user to change the alphabet, i.e. add and remove elements at any moment. Hence, the choice of classifier is limited to models that can dynamically adjust their architectures without loosing the data that was already computed. A static classifier is not an option for three reasons. First, its potential, the number of classes it can distinguish, has an upper bound that may limit the size of alphabet. Second, among classifiers with enough potential to distinguish between members of a user-defined alphabet, we want to choose the smallest one for performance reasons. Third, the alphabet size may change during run time if the user decides to add or delete symbols. Instead of a static classifier, we use a dynamic one, namely, a neural network with an adaptive architecture as described in Section 4.

As mentioned above, we employ spatio-temporal patterns as alphabet elements. Simply speaking, an STP is a path produced by the cursor. The temporal component, mouse events interarrival times, plays secondary role in recognition and its analysis is described in the next section. The spatial component constituted by path points is what the classifier actually learns to recognize. In our system, we attempted to minimize the restrictions on path configuration and to let the user decide which patterns are to be remembered and reproduced easily. However, it was necessary to limit the path complexity from both below and above. The former was needed because primitive patterns (lines, arcs etc) often emerge during cursor movements and would result in frequent false recognitions. The upper bound for the path complexity is set indirectly by two parameters, input buffer length and maximum position age, which can be adjusted for optimal performance (as described in the next section).

STPs undergo only minor preprocessing before being fed to the classifier. The system does not employ any sophisticated feature extraction algorithms, for example, path direction is omitted. However, if poor input quality due to noise, low precision of pointing device and/or human factors is expected, it might be necessary to employ a more elaborate preprocessing scheme. In particular, we conducted a number of experiments to record and analyze input patterns produced by severely disabled people using the Camera Mouse [1] as the pointer input device. To solve the problem of poor input quality, we attempted to extract regions to which the cursor seemed to converge. We will return to this issue in the discussion section. Let us now describe the primary components of the system: the high-level recognition procedure and the low-level classifier.

3 Recognition System

This section describes how input data is preprocessed before being supplied to the classifier and how the classifier's output is analyzed and interpreted. These procedures require both intensive processing and close interaction with the input hardware. A simple but effective scheme is employed to ensure efficient use of computational resources as well as responsiveness of the system. A recognition pipeline overview is given in Figure 2.

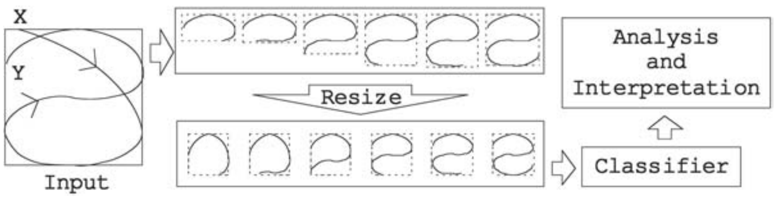

Fig. 2. Recognition System: The input pattern, drawn from X to Y, is shown processed in segments (top row). Each segment includes the previous segment plus additional cursor positions on the input path. The bounding boxes of the segments are resized (bottom row, magnified for visualization purposes) to fit a fixed size rectangle. The pixels of this rectangle constitute the classifier input.

3.1 Input Acquisition and Buffering

The system employs two buffers. The main buffer stores the current input segment processed by the system. The preliminary buffer is necessary to store cursor positions that do not introduce significant changes to the pointer path in the main buffer. When the preliminary buffer accumulates enough data, its contents are appended to the main buffer and recognition is restarted. The purpose of having this temporary buffer is to prevent interruption of the recognition process due to minor (perhaps, involuntary) cursor motion. After the main buffer is updated with the new path segment, some of the old pointer positions are removed from it to ensure it does not exceed its length limit. Moreover, all positions whose age is older than a certain maximum (typically 1 sec) are also removed. The main buffer length limit and maximum cursor position age are determined based on the trained STP's complexity and cursor movement speed. Their main purpose is to aid the classifier's operation under the real-time constraints.

3.2 Cursor Path Segmentation and Processing

Because recognition is done continuously and there are no special rules for entering STPs (unlike graffiti interfaces), the system expects that a pattern may start with any cursor position (stored in the buffer) and must end with the most recent position in the buffer. If exhaustive processing of the accumulated input were performed after each buffer update (which includes path segmentation, multiple scaling operations and multiple classifier evaluations), a heavy load of computational resources would result, which would be unacceptable for an interface. To reduce computation time, the recognition process is suspended until cursor motion reduces, which results in a smaller rate of buffer reads. Consequently, the time-consuming recognition process usually runs after all input, which may contain a valid STP, has been acquired, while the buffer is kept up to date at all times. Furthermore, this allows controlling an acceptable processor load by adjusting the preliminary buffer length and the main buffer update-rate threshold.

When the system decides that the current buffer update rate gives it enough time to process the accumulated input, recognition is initiated. First, the system finds all cursor path segments that can potentially be recognized as valid STPs. Each segment starts with the most recent position in the buffer. The bounding box of the minimal segment has to have an area that is at least as large as the classifier input size and it must be more complex than a line segment. Each following segment contains the previous one plus enough cursor positions to allow resizing of the bounding box (see Figure 2). Resizing is necessary to equate the area of the segment bounding box to the classifier's input size. The fact that the original segments are usually much larger than their resized versions allows us to grow segments several pointer positions at a time.

Each resized segment is evaluated by the classifier whose output is stored for analysis. This operation ends a single processing cycle (segment extraction, resizing, and evaluation). At this point, the recognition process might be forced to halt until the interpretation of a command is completed. In this case, the system either interprets information collected so far, or discards it. Otherwise, a new segment is extracted and processed. The next section describes how the processing results are analyzed and interpreted in each case.

3.3 Classifier Output Analysis and Interpretation

As described in the previous part, input path segments are extracted and processed in order of their length starting from the shortest. As a result, simple STPs, which can represent frequently used commands (mouse clicks, browse back/forward etc.) are recognized almost instantly. Very often, however, simple patterns emerge within more complex ones. A circle, for example, is a very convenient pattern to use, but it is a part of many digits and letters. For this reason, it is necessary to complete all processing cycles even if one of the first segments was positively recognized by the classifier.

In case the system succeeds in recognizing one of the early segments, but then is interrupted, it has to quickly decide what to do with the recognized STP and start processing more up-to-date input. It turns out that the best way to make this decision depends on the user's proficiency with the system and the currently performed task. Experienced users who trained the classifier to accurately recognize their input usually do not need feedback on one command before they can start entering the next one. During text processing, for example, one can enter letters with only small delays between each two. On the other hand, an inexperienced user makes more pronounced delays before starting an STP and after ending it. Moreover, certain tasks, like web browsing, usually require feedback on one action before the next one can be performed. Hence, in all cases, the time stamps that are stored with all cursor positions provide the final piece of information necessary to positively identify STPs. If the endpoints of some input path segment correspond to peaks in the interarrival time of input events (see Figure 3, bottom), the system can be confident that it did, in fact, recognize a valid STP. This method significantly increases the system responsiveness. However, this method is inappropriate during early stages of training and in cases of bad control and/or poor precision of the input device. If no good segment candidate was found by the time recognition is interrupted, the collected data is discarded. Otherwise, the best candidate has to be interpreted, as discussed in the last paragraph of this section.

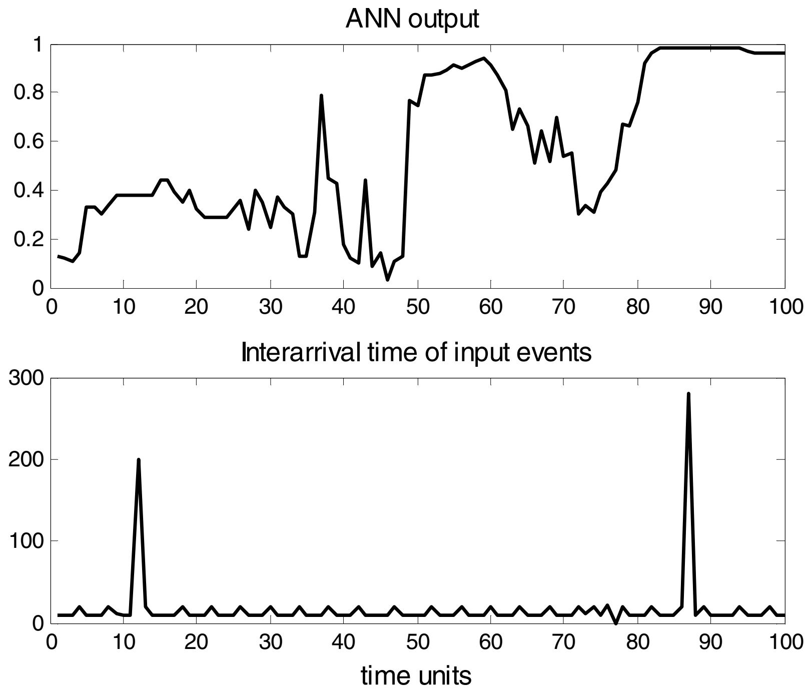

Fig. 3. Using the rate of input events to choose an STP. Top: ANN output over a period of 100 time units. Bottom: Interarrival time of input events over a period of 100 time units. At time t=12, a significant pause in cursor movement was registered, but no candidate STP was recognized by the ANN at that time. At time t=60, the ANN recognized a candidate STP, but no cursor pause was registered. At time t=88, the ANN recognized a candidate STP and a cursor pause was registered. Only in the last case does the system output that this candidate STP is a valid STP.

If two or more overlapping input path segments are recognized by the classifier (Figure 3, top), the one that correlates in time with the peak in input event interarrival time is the true STP. The reason why this works is that the speed of the cursor movement is usually rather stable when a valid pattern is being drawn. Tests showed that this assertion is true in the great majority of cases. STP candidates are compared in this fashion after all segments have been processed. The best candidate (if there is one) is then interpreted as described in the next paragraph.

The user can associate each STP with an input event. Any event that an operating system can process can be chosen, for example, keyboard keys, combination of keys pressed, mouse clicks, special buttons like back, forward, refresh etc. When the system positively recognizes an STP, it generates the corresponding event. In case of mouse clicks, the exact position of each click event occurrence is determined using the position of a predefined "hot-spot" relative to the pattern's bounding box. The default location of the hot-spot is the center of the bounding box, but any point within the box can be chosen. Note that it is possible to associate series of events or macros with STPs, which may be convenient in some applications.

4 Artificial Neural Network

The current implementation of the system employs an ANN as an STP classifier. As described in the overview section, we had two primary requirements for the choice of the classifier: real-time performance and an adaptive architecture. Although training an ANN can take a significant amount of time, the evaluation complexity of even large networks is rather low. In fact, some real-time applications utilize ensembles of neural networks and still meet all the deadlines. On the other hand, fulfilling the requirement of an adaptive architecture is somewhat problematic because most neural networks are used with static architectures. In this section we describe a simple neural network that dynamically changes its architecture to distinguish between an arbitrary number of classes efficiently. For this problem, we turn to supervised classification methods using a multi-layer perceptron (MLP) [4] as a base classifier. It is possible to use supervised classifiers because the alphabet is predefined by the user and constitutes the set of training as well as teaching inputs. We now proceed to discuss the network architecture, training set acquisition and training methods that we used with the ANN.

4.1 Adaptive Architecture

The neural network receives the scaled pointer path image pixels as input. Thus, the size of the input layer equals the resolution of the scaled image, which is a parameter to the system (16x16 was chosen experimentally). Higher resolution can represent more complex patters, but takes longer to evaluate. The resolution should be chosen according to alphabet size and input device precision.

The number of nodes in the output layer equals the number of classes defined by the user. It changes when the user adds or deletes commands to the alphabet. Note that adding or deleting a node does not result in the loss of data (e.g. weights) that was already computed for other nodes.

The size of the hidden layer (only one layer is used), on the other hand, is a parameter that can be adjusted, and is critical for the network performance and potential. From the performance point of view, the number of hidden nodes has to be minimized. Unfortunately, there is no simple relation between the network potential and the number of nodes and/or weights. As a result, neural network parameters are usually selected through experimentation rather than computed from strict rules and formulas. To optimize the architecture and the training of our neural network, we employ a combination of a method to automatically choose the size of the hidden layer and a probabilistic technique to assess the likelihood of training convergence.

Given the dimensionality of a classification task, one can estimate the sufficient number of hidden nodes in a two-layer (one hidden layer) fully connected feed-forward neural network by bounding the number of weights N_w [7,11]:

$$\frac{N_y N_p}{1+\log_2(N_p)} \le N_w \le N_y\left(\frac{N_p}{N_x}+1\right)\left(N_x+N_y+1\right)+N_y \tag{1}$$

where N_x denotes the number of input nodes, N_y denotes the number of output nodes, and N_p denote the size of the training set. Number of hidden nodes N_z is then given by:

$$N_z = \frac{N_w}{N_x + N_y} \tag{2}$$

Figure 4 shows minimum, maximum and average N_z values as given in Eq. (2). The best and worst cases correspond to "easy" classification tasks (close to linear) and "hard" ones (highly non-linear) respectively. Because no a priori information is available about pattern configurations, it is reasonable to guess that the actual value lies somewhere in the middle. We initialized the number of hidden nodes to the average of maximum and minimum N_z values.

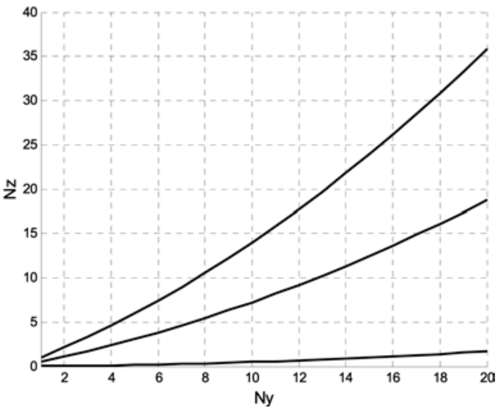

Fig. 4. Minimum, maximum and average number of hidden nodes N_z for 1–20 classes N_y, 10 examples per class and 256 input nodes N_x.

Slightly overestimating the network potential proved to be not harmful, whereas a far more complex problem arises if the potential associated with the size of the hidden layer is not sufficient to accommodate all classes. This can happen if either the initial guess is too small or if the user adds more classes (commands) to the existing network. In such case, it will be impossible to train the network to produce the desired classification. The main difficulty comes from the fact that one cannot tell what compromises the training procedure: it might just get stuck in a local minimum, or there might not be a suitable minimum at all. In the first case, training should just be restarted with random weights, while in the second case, the number of hidden nodes should be increased – otherwise training may not converge. Taking a probabilistic approach to this problem, one can estimate the number of training attempts that should be made before one establishes the inability of the training procedure to converge for a given network state by computing [8]:

$$N = \frac{\ln(1 - F_W(a))}{\ln(1 - F_X(a))} \tag{3}$$

Here N is the number of training attempts taken before changing the architecture, X is the sum of squared errors on any individual attempt, W is the best (lowest) value for X, $F_X(a)$ is the fraction of attempts which would result in a value of X less than or equal to *a*, a confidence threshold, and $F_W(a)$ is the fraction of X values that result in a value of W less than or equal to *a*. Once N attempts to converge have been made, the number of hidden nodes should be increased.

We must note the principal difference between our approach and the approach of a modified Cascade Correlation network (CasCor) [5,10]. Although a CasCor network implements a dynamical architecture by sequentially increasing (cascading) the number of hidden layers, it was shown that the potential of such network can be utilized only with a large (unfeasible in this case) number of training examples [10]. Expanding on this idea, architecture with one hidden layer to which nodes are sequentially added was later introduced [10]. The resulting network performed at least as well as the original CasCor on a set of benchmark problems [12]. Although seemingly efficient, the modified CasCor architecture does not take into account a priori information about the problem at hand, for example, input dimensionality. In our approach, the size of the hidden layer is selected according to the complexity of the classification task without resorting to traverse architectures which are unlikely to deliver the potential required for the given task.

4.2 Training

Neural network training proceeds in two stages (see Figure 5). During the first stage, which we call basic training, the network is presented with only few examples of each pattern (usually five). In rare cases, these examples are sufficient to train the network to stably recognize a pattern. However, most STPs require larger training sets to ensure correct classification. Basic training provides the network with a rough estimate of the classification task, so the error rate might be high due to partially learned decision boundaries of certain classes. In other words, the neural network was not presented with enough examples of one or more classes to be able to distinguish between all accurately. The system identifies problematic classes by assigning a confidence value to each member of the alphabet. Initially, all classes have the same value.

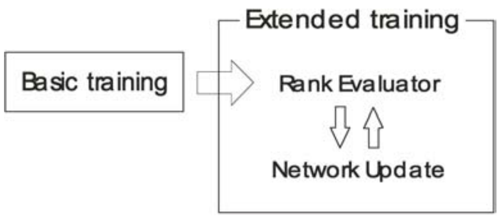

Fig. 5. Training overview.

In the second stage, which we call extended training, the user can start experimenting with the system. During this experimentation, the recognition error rate is usually still high. The user is asked to notify the system about its mistakes in order to update the confidence values of the alphabet elements to reflect their recognition error rates. The confidence values are increased for every correct classification and decreased for every incorrect one. To speed up the training, the user is asked to provide additional examples for alphabet elements. The chance of an element being selected is proportional to its confidence value. As a result, STPs that were recognized poorly after basic training are emphasized during extended training. Each new sample STP is added to the training set if the current network does not recognize it correctly and is discarded otherwise. In addition, for each new sample STP, the system continues to adjust confidence values. The network is updated after each time an additional STP is entered by the user. This procedure, if continued long enough, typically results in a sufficient number of examples for each class. In practice, it takes about 5 minutes to decrease the classification error rate to a point where the system can be used reliably. The purpose of dividing training into two stages is to ensure that correct classification will be produced by the network trained on as few examples as possible, hence minimizing training time.

We employ a standard back-propagation training algorithm with gradient descent and conjugate gradient descent methods [3]. To increase the speed of training and reduce the likelihood of converging into local minima, we employed different training modes (stochastic, batch), a momentum term and a variable learning rate [3]. The first few examples of an alphabet member are trained sequentially in batch mode, which is quite robust for small training sets. Then training switches to a stochastic mode, which is more robust for large training sets. Such a combination of training modes and the techniques mentioned above resulted in rapid convergence.

5 Testing

In this section we describe the methodology and results of the quantitative experiments. In the next section we describe our experiences working with a subject with severe disabilities and discuss possible extensions of our system.

5.1 Participants

Formal testing was conducted with participation of twenty subjects. The first ten subjects (Group A) were sophomore year college students with above average computer literacy. The next ten subjects (Group B) were individuals with average computer skills. None of the subjects was previously exposed to the system.

5.2 Methods and Apparatus

The testing procedure consisted of two tasks designed to test both the usability and the accuracy of the system. In the first task, which targeted primarily usability, subjects were asked to use the system to create an alphabet of five commands by mapping four arbitrary symbols to the following standard internet browser functions: "back", "forward", "stop", and "favorites", respectively. Subjects were asked to map the last symbol in the alphabet to the mouse single-click event. Upon completion of this configuration step, subjects were asked to perform basic training and proceed to interact with the system in its recognition mode to acquire alphabet rankings. After a period of interaction, subjects were asked to conduct an extended training step for 3 minutes. In this step, the subjects were asked to open an internet browser and, using only the five newly created symbols, browse the internet in a natural way with the mouse as the input device.

The second task required subjects to expand an existing alphabet to include ten symbols to be mapped to keyboard events that correspond to digits 0 through 9 and the last symbol to the mouse single-click event. The training procedure was the same as in the first task. Subjects were then asked to launch a word processor and produce all ten digits in a consecutive order using the mouse as a pointing device.

Note that access to the keyboard and full mouse functionality was allowed only during the configuration and training stages in both tasks. Results reflecting accuracy of the system (Recognition Accuracy) were recorded during the actual recognition stage and consisted of percent of correctly performed actions out of all attempted. The numbers of attempts were 50 for Task 1 and 10 for Task 2. As a measure of usability, the time it took each subject to configure the system (Average Adjustment Time) for each task was recorded. Another parameter that was recorded across all subjects was the average processing time of one symbol (Response Time) during the session.

All tests were conducted on Pentium IV 1.4 GHz machines running the Windows XP operating system.

Table 1. Average recognition accuracy of the system and the average adjustment time as measure of usability for two groups of 10 subjects each performing Task 1 50 times and Task 2 10 times.

Subject Groups	Average Recognition Accuracy		Average Adjustment Time	
	Task 1	Task 2	Task 1	Task 2
A (subjects with above average computer skills)	95 %	87 %	3 min	12 min
B (subjects with average computer skills)	85 %	75 %	6 min	18 min

5.3 Results

The results indicate that both groups of users were able to successfully complete the tasks assigned during testing (see Table 1). Although it is evident that individuals with less computer experience (Group B) took longer to adjust to the interface, high average recognition accuracy was achieved for both groups. Recognition accuracy depended directly on the duration of the extended training procedure, which was here limited to 3 minutes (importance of prolonged extended training is discussed in Section 4.2). Samples of symbols created by users are shown in Figures 6 and 7.

Results show that an average user as well as a more advanced user can configure this interface in a relatively short time (see Table 1). At the end of the second testing task, subjects were capable of using such tools as a calculator user interface without resorting to a keyboard and using a mouse merely as a pointer input device. The system proved to be robust with the average response time under 600 milliseconds.

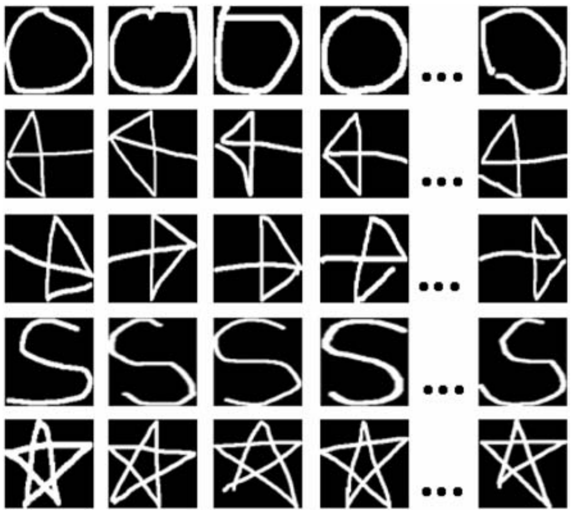

Fig. 6. Instances of five symbols as produced by a user in Group B in Task 1.

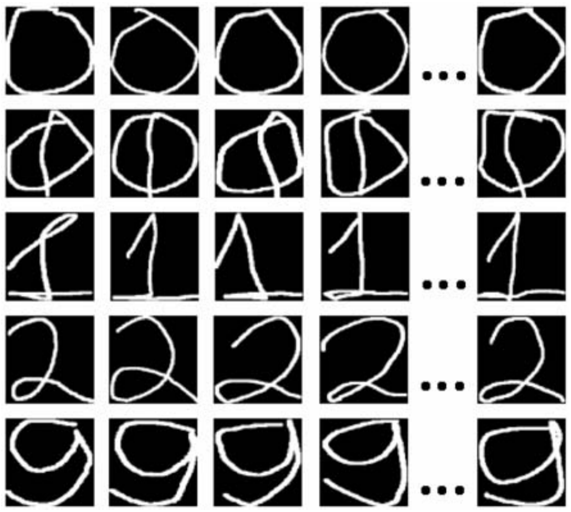

Fig. 7. Instances of five out of eleven symbols as produced by a user in Group B in Task 2.

6 Discussion

The system described in this paper is a general interface for communication with the computer. Since it does not need information about the type of pointer device used to control the cursor, it provides a general extension to the device's capabilities. The system was designed to be modular: each of the main components – input processor, classifier, and interpreter – can be changed or completely altered to fit particular tasks and input devices.

The system is intended to replace the functionality of the mouse input device and some functionality of the keyboard input device. For this purpose, the number of commands (order of 10-20) and the number of instances of training patterns for each command (20-30) used in the testing phase were appropriate. The system can potentially be expanded to handle a larger number of commands, for example, fully replacing the functionality of the keyboard. Such expansion, however, will raise the computational demands of the system requiring a redesign of its major components.

The two major approaches to improve input processing are feature extraction (reduces input dimensionality) and to use additional input characteristics (cursor movement direction, speed etc). Feature extraction is beneficial if input characteristics that emphasize differences between classes can be identified and no significant computational costs are associated with their extraction. As for using additional input characteristics, this can only be justified if the readily available spatio-temporal information is not a sufficient representation of the user's input. We did not choose to extract additional information because the STPs used carried sufficient information for symbol classification.

The fact that the system performed well during testing, and that users quickly became comfortable with it, encouraged us to tackle a more difficult problem. Taking into account the constraints placed on the input devices that can be used by disabled individuals we examined how to modify it to work with camera-based human-computer interfaces (in particular, the Camera Mouse [1]) to provide an accessibility solution for these individuals. The major difficulty about recognizing input produced by motion-impaired users is that, in most cases, the cursor path is significantly distorted by involuntary movements [2]. In other words, a subject can move the cursor sooner or later to a desired screen position, but the cursor's trajectory on the way to the target point is difficult for the subject to reproduce. To address this problem we explored the idea to discretize the input signal further in order to extract "pivot points," i.e., points or regions that the cursor's path must travel through. The neural network would then be utilized to classify patterns constituted by pivot points. We conducted preliminary experiments with a motion-impaired user due to severe cerebral palsy, showing that pivot points can indeed be detected and analyzed for classification. This approach is a subject of a future investigation.

Acknowledgements. We wish to thank John J. Magee, Rick Hoydt, and the students of the Boston University Video Game Creators Consortium for assistance. The work was supported by the National Science Foundation, grants IIS-0093367, IIS-0308213, IIS-0329009, and EIA-0202067.

References

1. Betke, M., Gips J., Fleming P.: The Camera Mouse: Visual Tracking of Body Features to Provide Computer Access for People With Severe Disabilities. IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 10, no. 1 (2002)
2. Cloud, R.L., Betke, M., Gips, J.: "Experiments with a Camera-Based Human-Computer Interface System." Proceedings of the 7th ERCIM Workshop "User Interfaces for All," UI4ALL 2002, pp. 103-110, Paris, France, October 2002.
3. Cun, Y. L., Bottou, L., Orr, G., Muller, K.: Efficient BackProp, Neural Networks: Tricks of the trade. Springer Lecture Notes in Computer Sciences 1524, pp.5-50 (1998)
4. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. New York: Wiley Interscience (2001)
5. Fahlman, S.E., Lebiere, C.: The Cascade-Correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburg PA (1991)
6. Grauman, K., Betke, J., Lombardi, J., Gips, J., Bradski, G.: Communication via Eye Blinks and Eyebrow Raises: Video-Based Human-Computer Interfaces. Universal Access in the Information Society, 2(4), pp. 359-373 (2003)
7. Hecht-Nielsen, R.: Kolmogorov's Mapping Neural Network Existence Theorem. Proceedings of IEEE First Annual Int. Conf. on Neural Networks, San Diego, Vol. 3, pp. 11-13 (1987)
8. Iyer, M.S., Rhinehart, R.R.: A Method to Determine the Required Number of Neural Network Training Repetitions. Proceedings of IEEE Transactions on Neural Networks, vol. 10, no. 2, pp 427-432 (1999)
9. Kim, K.-N., Ramakrishna, R.S.: Vision-Based Eye-Gaze Tracking for Human Interface. IEEE International Conf. on Systems, Man, and Cybernetics, Tokyo, Japan (1999)
10. Sjogaard, S.: A Conceptual Approach to Generalization in Dynamic Neural Networks. Ph.D. Thesis, Aarhus University, Aarhus, Denmark (1991)
11. Widrow, B., Lehr, M.A.: 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. Proceedings of the IEEE, vol. 78, No. 9, pp. 1415-1442 (1990)
12. Yeung, D.-Y.: Node Splitting: A Constructive Algorithm for Feed-Forward Neural Networks. In Moody, J.E., Hanson, S.J., Lippman, R.P. (eds.) Advances in Neural Information Processing Systems 4, San Mateo, CA. Morgan Kaufman Publishers, pp. 1072-1079 (1991)