International Journal of Knowledge-based and Intelligent Engineering Systems 12 (2008) 187–199
IOS Press

Incremental evolution of a signal classification hardware architecture for prosthetic hand control

Jim Torresen
Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: jimtoer@ifi.uio.no

Abstract. Evolvable Hardware (EHW) is a new method for designing electronic circuits. However, there are several problems to solve for making high performance systems. One is the limited scalability of the ordinary approach. To reduce this problem, a novel digital signal classification architecture has been developed that allows for incremental evolution. This is based on initially evolving subcircuits for each category to be detected.
The architecture is applied for classifying sensor data in a prosthetic hand controller. By applying the proposed method, the best performance achieved is substantially better than that obtained by more ordinary approaches. Analysis of the best circuit shows the importance of having an architecture containing some gates with random connections. Results of the analysis can also be used to collect better training data from users in the future.

1. Introduction

Recently, several automated approaches (including artificial neural networks) suited for design of pattern recognition and control problems have been developed. One of these is called evolvable hardware (EHW). It has been applied to a large range of real-world applications. The application applied in this paper is prosthetic hand control.

Instead of manually designing a circuit, only input/output-relations are specified in evolvable hardware. The circuit is automatically designed using an adaptive algorithm inspired from natural evolution. The algorithm is illustrated in Fig. 1. In this algorithm, a set (population) of circuits – i.e. circuit representations, are first randomly generated. The behaviour of each circuit is evaluated and the best circuits are combined – using crossover and mutation operators, to generate new and hopefully better circuits. Thus, the design is based on incremental improvement of a population of initially randomly generated circuits. Circuits among the best ones have the highest probability of being combined to generate new and possibly better circuits. The evaluation is according to the behaviour initially specified by the user. After a number of iterations, the fittest circuit is to behave according to the initial specification. The most commonly used evolutionary algorithm is genetic algorithm (GA) [3]. The algorithm – which follows the steps described above, contains important operators like crossover and mutation of the circuit representations for making new circuits.

To enhance the lives of people who have lost a hand, prosthetic hands have existed for a long time. These are operated by the signals generated by contracting muscles – named electromyography (EMG) signals, in the remaining part of the arm [13]. Presently available systems normally provide only two motions: Open and close hand grip. The systems are based on the user adapting *himself* to a fixed controller. That is, he must train himself to issue muscular motions trigging the wanted action in the prosthetic hand. A long time is often required for rehabilitation.

By using EHW it is possible to make the *controller* itself adapt to each disabled person. The controller is constructed as a pattern classification hardware which maps input patterns to desired actions of the prosthetic

ISSN 1327-2314/08/$17.00

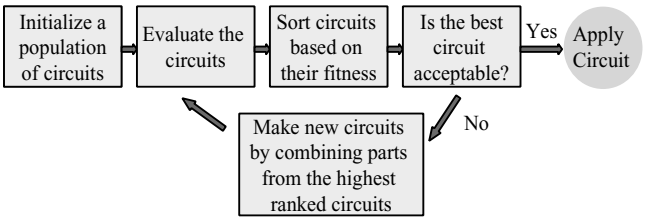

Fig. 1. The algorithm for evolving circuits.

hand. Adaptable controllers have been proposed based on neural networks [2]. These require a floating point CPU or a neural network chip. EHW based controllers, on the other hand, use a few layers of digital logic gates for the processing. Thus, a more compact implementation with *faster processing* can be provided making it more feasible to be applied in a prosthetic hand. However, the signal classification performance is most important, and our goal in this paper is to evolve a controller with better performance than a neural network.

Experiments based on the EHW approach have already been undertaken by Kajitani et al. [7–9]. The research on adaptable controllers is based on designing a controller providing six different motions in three different degrees of freedom. Such a complex controller could probably only be designed by *adapting* the controller to each dedicated user. It consists of AND gates succeeded by OR gates (Programmable Logic Array). The latter gates are the outputs of the controller, and the controller is evolved as one complete circuit. The simulation indicates a similar performance as artificial neural network but since the EHW controller requires a much smaller hardware and provide faster processing speed, it is to be preferred. The approach proposed in this paper is distinguished both with a more advanced architecture as well as applying incremental evolution.

One of the main problems in evolving hardware systems seems to be the limitation in the chromosome string length [11,19]. A long string is normally required for solving a complex problem as seen in Fig. 2. However, a larger number of generations is normally required by the evolutionary algorithm as the string increases. This often makes the search space becoming too large. Thus, work has been undertaken to try to diminish this limitation. Various experiments on speeding up the GA computation have been undertaken [1]. The schemes involve fitness computation in parallel or a partitioned population evolved in parallel – by parallel computation. Other approaches to the problem have been by using variable length chromosome [6] and reduced chromosome representation [4]. Another option, called function level evolution, is to evolve at a higher

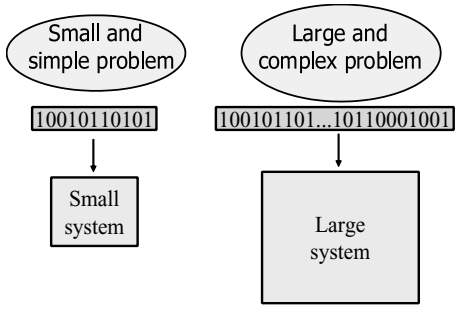

Fig. 2. The chromosome string length and representation ability.

level than gate level [12]. That is, instead of applying gates as building blocks, higher level functions are applied. Most work is based on fixed functions. However, there has been work in Genetic Programming for *evolving* the functions – called Automatically Defined Functions (ADF) [10].

Another improvement to artificial evolution – called co-evolution, has been proposed [5]. In co-evolution, a part of the data, which defines the problem, co-evolves simultaneously with a population of individuals solving the problem. This could lead to a solution with a better generalization than a solution evolved based on the initial data. Further overview of related works can be found in [17].

Incremental evolution for EHW was first introduced in [14] for a character recognition system. The approach is a divide-and-conquer on the evolution of the EHW system, and thus, named *increased complexity evolution*. It consists of a division of the *problem* domain together with incremental evolution of the hardware system. This can be seen as dividing the problem on the right-hand side in Fig. 2 into a set of simpler and smaller problems – as given by the one on the lefthand side of the figure, which can be more easily evolved. Thus, evolution is first undertaken individually on a set of sub-systems, based on the assumption that the total evolution effort is less than for evolving a single system. The evolved sub-systems are the building blocks used in further evolution of a larger and more complex system. The benefits of applying this scheme is

both a *simpler* and *smaller* search space compared to conducting evolution in one single run [17]. The goal is to develop a scheme that could evolve systems for complex real-world applications.

In this paper, the approach is applied to evolve a prosthetic hand controller circuit. A new EHW architecture as well as how incremental evolution is applied are presented. The goal is to provide a signal classification architecture – based on digital gates, providing a high classification accuracy to make it a strong alternative to artificial neural networks. Others benefits of the approach would be compactness and speed of classification. The architecture for prosthetic hand control was first proposed in [16]. However, this paper contains a more thorough comparison with neural network performance as well as analysis of the evolved circuit.

The next two sections introduce the concepts of the evolvable hardware based prosthetic hand controller. Then results are given in Section 4 with conclusions in Section 5.

2. Prosthetic hand control

The research on adaptable controllers presented in this paper provides control of six different motions in three different degrees of freedom: Open and Close hand, Extension and Flection of wrist, Pronation and Supination of wrist. The data set consists of the same motions as used in earlier work [8], and it has been collected by Dr. Kajitani at National Institute of Advanced Industrial Science and Technology (AIST) in Japan. The classification of the different motions could be undertaken by:

- **Frequency domain:** The EMG input is converted by Fast Fourier Transform (FFT) into a frequency spectrum.
- **Time domain:** The absolute value of the EMG signal is integrated for a certain time.

The latter scheme is used since the amount of computation and information is less than in the former. With EMG signals being a very coarse or imprecise measurement of nerve signals, it would be limited benefit of frequency analysis compared to time domain analysis.

The published results on adaptive controllers are usually based on data for non-disabled persons. Since you may observe the hand motions, a good training set can be generated. For the disabled person this is not possible since there is no hand to observe. The person would have to by himself distinguish the different motions.

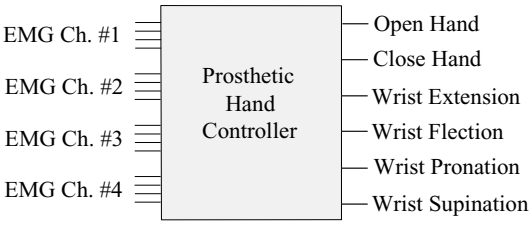

Fig. 3. Illustration of the controller interfaces.

Thus, it would be a harder task to get a high performance for such a training set but it will indicate the expected response to be obtainable by the prosthesis user. This kind of training set is applied in this paper. Some of the initial results using this data set can be found in [16,17].

2.1. Data set

Four EMG channels are applied in total where each channel records signals from one muscular area on the remaining part of the arm. Several channels are used to improve the performance of the controller. The collection of the EMG signals are undertaken using three sensors for each channel. The difference in signal amplitude between the two of them, together with using the third as a reference, gave the resulting EMG signal for each channel. The absolute value of the EMG signal is integrated for 1 s and the resulting value is coded by *four* bits. With *four* channels used in total, this results in an input vector of $4 \times 4 = 16$ bits. The controller interfaces are illustrated in Fig. 3.

A subset of the training set inputs – consisting of pre-processed EMG signals, is given in Fig. 4. For each motion, 10 samples are included. The *output* vector consists of one binary output for each hand motion, and therefore, the output vector is coded by *six* bits. For each vector only *one* bit is "1". Thus, the data set is collected from a disabled person by considering *one* motion at a time in the following way:

1. The person contracts muscles corresponding to one of the six motions. A personal computer (PC) samples the EMG signals.
2. The key corresponding to the motion is entered on the keyboard.

For each of the six possible motions, a total of 50 data vectors are collected, resulting in a total of: $6 \times 50 = 300$ vectors. Further, *two* such sets were made, one to be used for evolution (training) and the other to be used as a separate test set for validating the best circuit *after* evolution is finished.

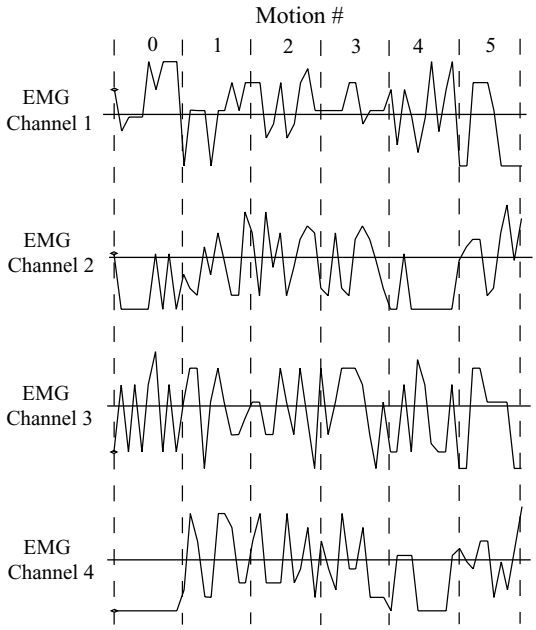

Fig. 4. EMG signals from the training set. 10 samples of data for each motion.

There have been undertaken experiments on using a *look-up table* for solving this problem [18]. That is, each output vector in the data set is stored in a table addressed by the corresponding input vector. The data set is also used in various ways to program the rest of the empty locations in the table. In the best case, the test set performance was 58% (72.3% training set performance). The best training set performance obtained was 90%, but such a table gave a low test set performance. In this experiment – using a data set from a disabled person distinguishing six different motions, we could not expect a high performance as also discussed in the beginning of Section 2. Thus, the goal of the following work is to demonstrate how high performance is obtainable with the challenging data set.

3. An architecture for incremental evolution

In this section, the architecture for the controller is described [16]. This includes the algorithm for undertaking the incremental evolution. This is all based on the principle of *increased complexity evolution* introduced in Section 1.

The architecture is illustrated in Fig. 5. It consists of one subsystem for *each* of the six prosthetic motions. In each subsystem, the binary inputs $x_0 \ldots x_{15}$ are processed by a number of different units, starting by the AND-OR unit. This is a layer of AND gates followed by a layer of OR gates. Each gate has the same number of inputs, and the number can be selected to be two, three or four. The outputs of the OR gates are routed through a kind of filter called Selector. This unit selects *which* of these outputs that are to be counted by the succeeding counter. That is, for each new input, the Counter is counting the number of *selected* outputs being "1" from the corresponding AND-OR unit. The main motivation for introducing the selectors is to be able to select a set of outputs from each AND-OR unit in a flexible way to possibly improve the performance. Finally, the Max Detector outputs which counter – corresponding to *one* specific motion, is having the largest value.[1] Each output from the Max Detector is connected to the corresponding motor in the prosthesis. If the Counter having the *largest* value corresponds to the correct hand motion, the input has been correctly classified. The output results are used in the following experiments for computing the fitness function – see Section 3.1.1.

A scheme, based on using multi-input AND gates together with counters, has been proposed earlier [20]. However, the architecture used in this paper is distinguished by including OR-gates, together with the selector units involving incremental evolution.

The incremental evolution of this system can be described by the following steps:

1. **Step 1 evolution.** Evolve the AND-OR unit for each subsystem *separately* one at a time. Apply *all* vectors in the training set for the evolution of each subsystem. There are no interaction among the subsystems at this step, and the fitness is measured on the output of the AND-OR units. A largest possible number of OR gates should output "1" for the 50 patterns the subsystem is set to respond to. For all other patterns, the number of gates outputting "1" should be as small as possible. That is, each subsystem should ideally respond only to the patterns for one specific prosthesis motion. Fitness is measured in a special way during evolution of each subsystem as outlined in Section 3.1.
2. **Step 2 evolution.** Assemble the six AND-OR units into one system as the system seen in Fig. 5. The AND-OR units are now fixed and the *Selectors* are to be evolved in the assembled system – in

[1] If two counters have the same maximum value, the system with highest id is selected.

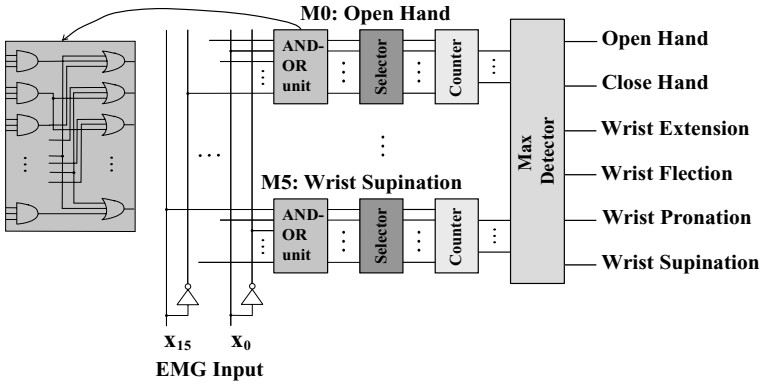

Fig. 5. The digital gate based architecture of the prosthetic hand controller.

one common run. The fitness is measured using the same training set as in step 1 but the evaluation is now on the output of the Max Detector.

3. The system is now ready to be applied in the prosthesis.

In the first step, subsystems are evolved separately, while in the second step these are assembled and evolved together by using the selectors. The motivation for evolving separate subsystems – instead of a single system in one operation, is that earlier work has shown that the evolution time can be substantially reduced by this approach [14,15].

The layers of AND and OR gates in one AND-OR unit consist of 32 gates each. This number has been selected to give a chromosome string of about 1000 bits which has been shown earlier to be appropriate. A larger number would have been beneficial for expressing more complex Boolean functions. However, the search space for evolution could easily become too large. For the step 1 evolution, each gate's *inputs* are determined by evolution. The encoding of each gate in the binary chromosome string is as follows:

Inp.1 (5 bit)	Inp.2 (5 bit)	(Inp.3 (5 bit))	(Inp.4 (5 bit))

As described in the previous section, the EMG signal input consists of 16 bits. Inverted versions of these are made available on the inputs as well, making up a total of 32 input lines to each AND-OR unit. The evolution is based on gate level building blocks. However, since several output bits are used to represent one motion, the signal resolution becomes increased from the two binary levels.

For the step 2 evolution, each line in each selector is represented by *one* bit in the chromosome. This makes a chromosome of 32×6 bits $= 192$ bits. If a bit is "0", the corresponding line should *not* be input to the counter, whereas if the bit is "1", the line *should* be input.

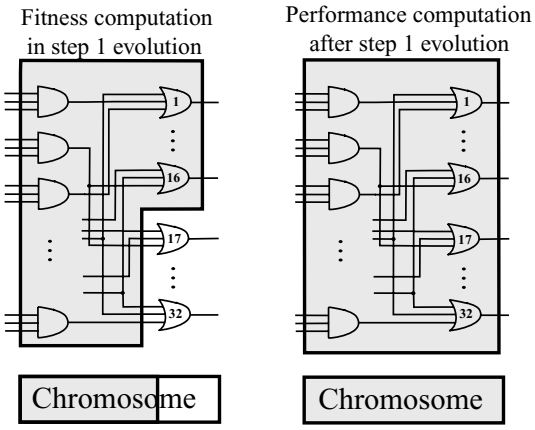

Fig. 6. A "fitness measure" equal to 16.

3.1. Fitness measure

In step 1 evolution, the fitness is measured on all the 32 outputs of each AND-OR unit. In an alternative experiment, it was found that the performance could be substantially improved if the fitness is measured on a *limited* number (16 is here used as an example) of the outputs [16]. That is, each AND-OR unit still has 32 outputs but – as seen in Fig. 6, only 16 are included in the computation of the fitness function:

$$\text{Fitness} = \sum_{i=1}^{16} \text{Output OR gate } i \text{ match target } i \quad (1)$$

The input connections to the 16 OR gates not used by the fitness function are included in the chromosome and these have *random* values. Thus, half of the OR gates have *random connections* to the AND gates. After

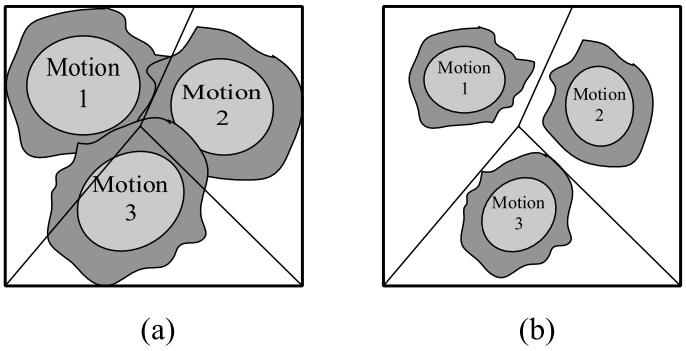

Fig. 7. Illustration of noise added to (a) a plain signal and (b) a pre-processed signal.

evolution, all 32 outputs are applied for computing the performance:

$$\text{Performance} = \sum_{i=1}^{32} \text{Output OR gate } i \text{ match target } i \quad (2)$$

Since 16 OR gates are used for fitness computation, the "fitness measure" equals to 16. In the figure, gate 1 to 16 are used for the fitness function. However, in principle any 16 gates out of the 32 can be used. Other numbers than 16 were tested in experiments but 16 showed to give the best performance results and was used in the following reported experiments.

There are several possible benefits of this approach. Firstly, it could make the evolution easier find good circuits since there is less number of outputs to measure fitness on. Thus, it could give a higher training set performance. Secondly, this could be an interesting approach to improve the generalization of the circuit – i.e. the test set performance. Only the OR gates in the AND-OR unit are "floating" during the evolution since all AND gates may be inputs to the 16 OR gates used by the fitness function. The 16 "floating" OR-gates then provide additional combination of these *trained* AND gates.

Another way to look at this is that the "floating" gates provide "noise" since the inputs to the "floating gates" are *randomly* connected to the AND gates. However, the noise is not added to the input signal but to a pre-processed and *improved* signal (output from the AND gates) as illustrated in Fig. 7. The inner circle for each motion indicates the *training set* domain, with the outer circle indicating the added generalization obtained by adding "noise". In (a) the input signal is not pre-processed and adding noise makes the interference among classes worse while in (b) it improves the generalization rather than introducing interference. The step 2 evolution will be evolving the ratio of noise in the final system by adjusting the number of selector bits set for gates 1 to 16 (evolved OR gates) compared to for gates 17 to 32 (OR gates with random connections).

3.1.1. Fitness function

The fitness function is important for the performance when evolving circuits. For the step 1 evolution, the fitness function – applied for each AND-OR unit separately, is as follows for the motion m ($m \in [0,5]$) unit:

$$F_1(m) = \frac{1}{s}\sum_{j=0}^{50m-1}\sum_{i=1}^{O} x + \sum_{j=50m}^{50m+49}\sum_{i=1}^{O} x + \frac{1}{s}\sum_{j=50m+50}^{P-1}\sum_{i=1}^{O} x$$

$$\text{where } x = \begin{cases} 0 \text{ if } y_{i,j} \neq d_{m,j} \\ 1 \text{ if } y_{i,j} = d_{m,j} \end{cases} \quad (3)$$

where $y_{i,j}$ is the computed output of OR gate i, and $d_{m,j}$ is the corresponding target value of the training vector j. P is the total number of vectors in the training set ($P = 300$). As mentioned in earlier, each subsystem is trained for one motion (the middle expression of F_1). This includes outputting "0" for input vectors for other motions (the first and last expressions of F_1).

The s is a scaling factor to implicitly emphasize on the vectors for the motion the given subsystem is assigned to detect. An appropriate value ($s = 4$) was found after some initial experiments. The O is the number of outputs included in the fitness function and is either 16 or 32 in the following experiments (referred to as "fitness measure" in the previous section).

The fitness function for the step 2 evolution is applied on the complete system and is given as follows:

$$F_2 = \sum_{j=0}^{P-1} x \text{ where}$$

$$x = \begin{cases} 1 \text{ if } d_{m,j} = 1 \text{ and } m = i \\ \quad \text{for which } \max_{i=0}^{5}(\textit{Counter}_i) \\ 0 \text{ else} \end{cases} \quad (4)$$

This fitness function counts the number of training vectors for which the target *output* being "1" *equals* the *id* of the counter having the maximum output (as mentioned in Section 2.1, only *one* output bit is "1" for each training vector).

3.2. The Evolutionary Algorithm

The simple Genetic Algorithm (GA) – given by Goldberg [3], was applied for the evolution with a population size of 50.

Elitism is used, thus, the best individual from each generation is carried over to the next generation. The (single point) crossover rate is 0.8, thus the cloning rate is 0.2. Roulette wheel selection scheme is applied. The mutation rate – the probability of bit inversion for each bit in the binary chromosome string, is 0.01.

A number of experiments were undertaken to find appropriate GA parameters. The ones that seemed to give the best results were selected and fixed for all the experiments. This was necessary due to the large number of experiments that would have been required if GA parameters should be able to vary through all the experiments. The preliminary experiments indicated that the parameter setting was not a major critical issue.

The experiments reported in this paper are undertaken off-line using software simulation. However, the proposed architecture fits into most FPGAs. Since no feed-back connections are used and the number of gates between the input and output is limited, the classification performance of implementation in hardware should equal the simulation. Any spikes could be removed using registers in the circuit. Further, there are many benefits of being able to implement a system in hardware like cost, power consumption and speed of operation.

For each experiment presented, four different runs of GA were performed. Thus, *each* of the four resulting circuits from step 1 evolution is taken to step 2 evolution and evolved for four runs.

4. Results

This section reports the experiments undertaken to search for an optimal configuration of the prosthetic hand controller. They will be targeted at obtaining the best possible performance for the *test* set.

Table 1 shows the main results – in percentage of correct classification. Several different ways of evolving the controller are included. The training set and test set performances are listed on separate lines in the table. The "# inp/gate" column includes the number of inputs for each gate in the AND-OR units. The columns beneath "Step 1 evolution" report the performance after only the *first* step of evolution. That is, each subsystem is evolved separately, and afterwards they become assembled to compute their total performance. The "Step 1+2 evolution" columns show the performance when the *selector units* have been evolved as well (step 2 of evolution). In average, there is an improvement in the performance for the latter. Thus, the proposed *increased complexity evolution* give rise to improved performances.

In total, the best way of evolving the controller is the one listed *first* in the table. The circuit evolved with the best *test set* performance obtained 67% correct classification. Figure 8 shows the step 2 evolution of this circuit. The training set performance is monotone increasing which is demanded by the fitness function. The test set performance is increasing with a couple of valleys. The circuit had a 60.7% test set performance after step 1 evolution.[2] Thus, the step 2 evolution provides a substantial increase up to 67%. Other circuits did not perform that well but the important issue is that it has been shown that the proposed architecture provides the *potential* for achieving a high degree of generalization.

A feed-forward neural network was trained and tested with the same data sets. The network consisted of (two weight layers with) 16 inputs, a variable number of hidden units and 6 outputs and it was trained with the backpropagation algorithm. Table 2 shows the main results – in percentage of correct classification. Training was run for 1000 iterations which were enough to find the maximum obtainable training set performance. The corresponding test set performance measured after training is given in the column "Final". However, to have a measure of the maximum possible test set performance, the test set performance was *monitored*

[2] Evaluated with all 32 outputs of the subsystems.

Table 1
The results of evolving the prosthetic hand controller in several different ways

Type of system	# inp/gate	Step 1 evolution			Step 1+2 evolution		
		Min	Max	Avr	Min	Max	Avr
A: Fitness measure 16 (train)	3	63.7	69.7	65.5	71.33	76.33	73.1
A: Fitness measure 16 (test)	3	50.3	60.7	55.7	44	67	55.1
B: Fitness measure 32 (train)	3	51	57.7	53.4	70	76	72.9
B: Fitness measure 32 (test)	3	40	46.7	44.4	45	54.3	50.1
C: Fitness measure 16 (train)	2	51.3	60.7	54.8	64.3	71.3	67.5
C: Fitness measure 16 (test)	2	46	51.7	49	44.3	54.7	50
D: Fitness measure 16 (train)	4	59.3	71.3	65.5	70	76	73.4
D: Fitness measure 16 (test)	4	52.7	59.7	55.3	48.3	56.3	52.7
E: Direct evolution (train)	4	56.7	63.3	59.3	–	–	–
E: Direct evolution (test)	4	32.7	43.7	36.6	–	–	–

Table 2
The results of neural network training of the prosthetic hand controller

Number of hidden units	Training performance	Test set performance	
		Final	Maximum
20	88	54	60
30	86.7	54.3	59
40	88	58.7	60
50	86.7	57	60.3

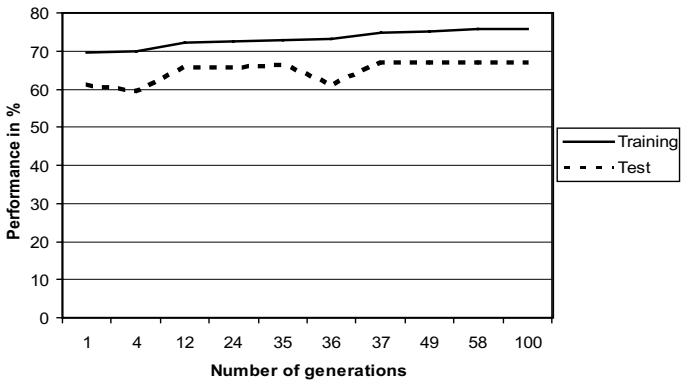

Fig. 8. Plot of the step 2 evolution of the best performing circuit.

throughout training and the maximum value is given in the column called "Maximum". There is some difference between these two columns but it is not very large – especially for the networks with a large number of hidden units. Thus, we see that it is beneficial to use a network with a large number of hidden units to obtain the highest training set performance.

In the best case, a test set performance of 60.3% correct classification was obtained. The best training set performance was 88%. Thus, a higher training set performance but a lower test set performance than for the best EHW circuit. The performance of the look-up table approach of 58% – reported in Section 2.1, is again less than that obtained by neural networks. This shows that the EHW architecture holds good generalization properties. However, future work should include comparison with other classification methods like Support Vector Machine (SVM).

The experiment B is the same as A except that in B all 32 outputs of each AND-OR unit are used to compute the fitness function in the step 1 evolution. In A, each AND-OR unit also has 32 outputs but only 16 are included in the computation of the fitness function as described in Section 3.1. The performance of A in the table for the step 1 evolution is computed by using *all* the 32 outputs. Thus, over 10% better training set as well as the test set performance (in average) are obtained by having 16 outputs "floating" rather than measuring their fitness during the evolution!

What is also interesting is that if the performance of

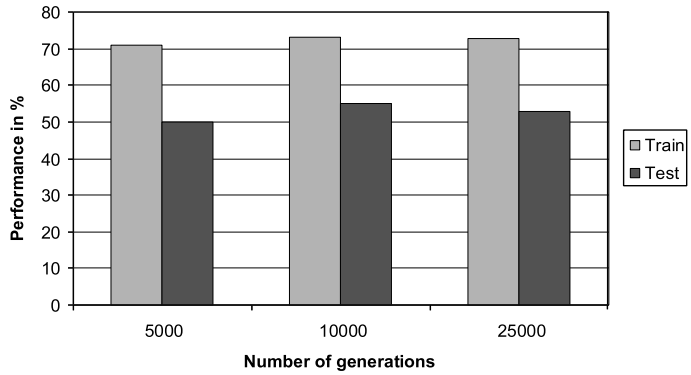

Fig. 9. Performance of three different numbers of generations in step 1 evolution (average of four runs).

the circuits in A is measured (after step 1 evolution) with only 16 outputs, the performance is not very impressive. Thus, "floating" outputs in the evolution substantially improve the performance – including the *test* set performance. This may seem strange but has a reasonable explanation – given in Section 3.1. The reason for B having lower performance could be explained by a more complex as well as larger search space for the evolution. In this experiment, a larger number of OR gates should give a correct output and the chromosome is longer since in A the bits assigned to the 16 "floating" OR-gates are not used. Other numbers of "floating" OR gates (8 and 24) were tested but the results were best for 16.

The C and D rows in the table contain the results when the gates in the AND-OR units each consists of two and four inputs, respectively. The lowest figures are for two input gates indicating that the architecture is too small to represent the given problem. Four inputs, on the other hand, could be too complex since having *three* input gates give a slightly better results.

Each subsystem is evolved for 10,000 generations each, whereas the step 2 evolution was applied for 100 generation. These numbers were selected after a number of experiments. One comparison of the step 1 evolution (each gate having three inputs) is included in Fig. 9 and shows that the best average performance is achieved when evolving for 10,000 generations.

The circuits evolved with direct evolution (E) were undertaken for 100,000 generations.[3] The training set performance is impressive when thinking of the simple circuit used. Each motion is controlled by a *single* four input OR gate. However, the test set performance is very much lower than what is achieved by the other approaches. This is explained by having an architecture that is too small to provide good generalization. A larger one, on the other hand, would make the chromosome string longer with the problems introduced earlier (larger and more complex search space). This once again emphasizes the importance of applying the *increased complexity evolution* scheme.

[3]This is more than six times 10,000 which were used in the other experiments.

Table 3
The number of vectors for which a subsystem wrongly outputs a larger number of "1"s than the correct subsystem for the best circuit of A after step 1 evolution. One row in the table represents the wrong responses of one subsystem (motion)

5	0	7	0	17	0	–
4	25	0	1	0	–	0
3	0	15	10	–	0	2
2	0	1	–	7	0	0
1	0	–	8	22	0	0
0	–	0	1	2	0	0
	0	1	2	3	4	5

In the sections below, the *best* evolved circuit in A will be analysed in more detail. The motivation for this is to get more information about how the architecture is configured. This could be helpful to further improve the architecture.

4.1. Signal interference

In this section, it will be analysed how vectors from different motions interfere with each other. Table 3 shows this for the *best* circuit evolved in step 1 evolution of A in Table 1 when the test set is applied.

One row in the table represents the wrong responses of one subsystem (motion). E.g. the output for motion 1 is mostly "disturbed" by vectors for motion 3 (22 out of 50 vectors for motion 1 are miss-classified). Similarly, the other way, vectors from motion 1 interferes the

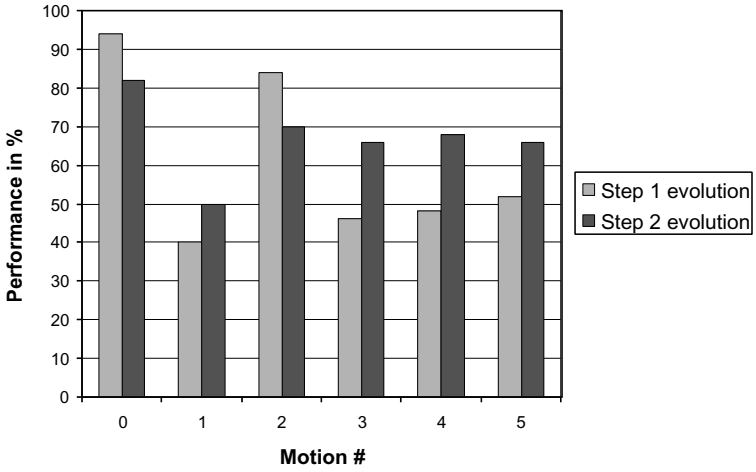

Fig. 10. The test set performance for each motion for the best circuit evolved.

Table 4
The number of vectors for which a subsystem wrongly outputs a larger number of "1"s than the correct subsystem for the best circuit of A after step 2 evolution. One row in the table represents the wrong responses of one subsystem (motion)

5	0	5	0	12	0	–
4	15	0	1	0	–	0
3	5	8	2	–	0	2
2	0	1	–	12	0	2
1	0	–	8	16	0	1
0	–	0	1	2	1	5
	0	1	2	3	4	5

Table 5
The number of vectors for which a subsystem wrongly outputs a larger number of "1"s than the correct subsystem for the worst performing circuit of A after step 1 evolution. One row in the table represents the wrong responses of one subsystem (motion)

5	0	24	1	7	0	–
4	7	0	1	1	–	0
3	0	31	11	–	0	2
2	0	0	–	8	0	0
1	0	–	10	4	0	0
0	–	0	1	17	24	0
	0	1	2	3	4	5

motion 3 classifier (15 vectors). This indicate vectors from motion 1 and 3 are not sufficiently different in the given data set. Similarly after step 2 evolution, the numbers are as given in Table 4. We see the same disturbance but the numbers are lower. Thus, it would be very important to be aware of this when data is collected from a user. It could be beneficial to collect some data where the user is switching between motions with much overlap. This could make it easier to get *different* signal response for the two motions.

Table 5 represents the same as given in Table 3, except that it is measured on the *worst* performing circuit of A. There is some tendency of problem vectors being in the same position as in the previous table. However, it is not all consistent.

4.2. Comparing step 1 and step 2 evolution

In Fig. 10, the performance of each motion of the best circuit is compared for step 1 and step 2 evolution, respectively. For some motions the performance after step 1 evolution is higher than that achieved after step 2 evolution. However, after step 2 evolution and as seen in the figure, the performance of the *worst* performing motions is improved. The latter would be most important to provide the best usability of the prosthesis. These results can – like those in Section 4.1, be used to collect better data from the user. It could be beneficial to collect additional data for the motions with low performance and have the user switch between motions with much overlap during data collection. This is based on the conclusions from the previous section.

4.3. Selector settings of the best performing circuit

In this section we would like to study the use of evolved OR gates compared to floating OR gates – introduced in Section 3.1. Table 6 shows the selector settings of the best performing circuit. It is interesting to observe that in average about half of the selector lines are connected (1 in the table indicates a connection) while the other half is not – see the total number in the rightmost column. For each of the motions, the number of selectors connected varies from 14 to 20. An

Table 6
The selector settings of the best performing circuit. Selector 1–16 is connected to evolved OR gates while selector 17–32 is connected to "floating" OR gates

Motion	Selector 1–16																Selector 17–32																Total # 1's
0	0	0	0	1	0	0	1	0	1	1	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	1	1	1	0	1	8 + 10=18
1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	0	0	0	11 + 8=19
2	1	0	0	1	1	0	1	0	0	0	1	1	0	0	0	0	1	1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	6 + 8=14
3	0	0	1	1	0	1	1	1	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	8 + 10=18
4	0	1	0	1	1	0	1	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0	1	1	0	1	0	1	0	0	0	1	11 + 8=19
5	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	1	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0	0	12 + 8=20

interesting observation is that there are almost as many lines connected to "floating OR gates" (52 in total) as non-floating gates (56 in total). This shows that there is no indication that the floating OR gates are eliminated through step 2 evolution, and they are as important as the evolved OR gates. There would be a potential for post evolution circuit optimisation since close to half of the OR gates are not used in the evolved circuit. This would lead to a circuit having smaller size.

4.4. Analysing the AND-OR unit organization

In this section, the AND-OR unit for motion 0 of the best system in A after step 1 evolution is analysed. There is room for much adaptivity in the AND-OR unit:

- An AND or OR gate may reduce its number of inputs by inputting the *same* source (or physical line) several times. The source is an EMG input for an AND gate and output from an AND gate for an OR gate, respectively. In the circuit, four AND gates and one OR gate applied this reduction.
- The output of an AND gate may be fixed to "0" by inputting both the inverted as well as the non-inverted signal from the same input. The OR gate connecting to such a kind of gate implicitly reduces its number of inputs. Two AND gates had this kind of input in the circuit.
- The number of times a line is connected to different gates in the next layer, the more weight is put on this line.

The weight on each of the inputs to the AND layer is illustrated in Fig. 11. There is a dominant number of non-inverted inputs used compared to inverted ones. However, *all* inputs are necessary in the system.

Similarly, Fig. 12 shows the number of times each AND gate is connected to OR gates. OR gates 1–16 are the "non-floating" gates while OR gates 17-32 are the "floating" gates during evolution. There is a larger variance among the first class of gates than for the latter ones. This is reasonable since the floating gates are *randomly* connected to AND gates. The architecture seems to be fully utilized since mainly all AND gates are connected to one or more OR gate as seen in Fig. 12. Thus, shrinking the size (to get faster evolution) may therefore lead to less performance.

4.5. Discussion

The experiments show that it is possible to evolve a classifier circuit that performs better than an artificial neural network. In operation it can also provide fast processing with its limited number of gate layers (which can be computed in parallel) compared to a neural network based on floating point computation. However, the challenge of evolvable hardware is the evolution time. This is long compared to the training time for a neural network (some reduction of evolution time could be achieved by optimising the program code). However, the evolution can easily be parallelized in hardware. This could be necessary if online evolvable systems should be made.

Future work would contain experiments with other data sets and test of architectures that allow for online adaptation. The latter would be important for users where the performance drops due to e.g. sweating. This would require an architecture where evolution can be run in the background while the prosthesis is still operating. Experiments should include real prosthesis users to judge what level of misclassification is acceptable. Further, the user would probably adapt to the behaviour of the controller that may lead to an increase in the degree of correct performance. Another way of improving the performance in the future could be by undertaking analogue pre-processing of the input signals.

The architecture should be beneficial for other applications as well. Thus, we would like to study applications that could be beneficial for the novel architecture. It would not be appropriate for all kinds of applications but applications with a small number of data variables with limited resolution should be appropriate. Examples could be various signal processing tasks for e.g. medical applications or communication systems.

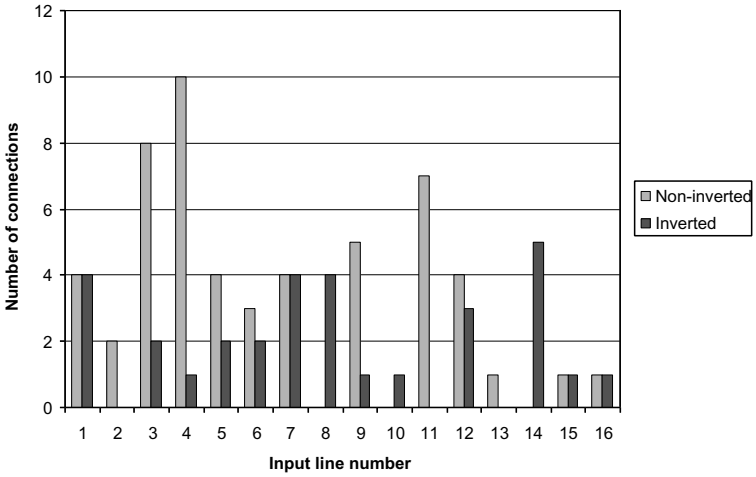

Fig. 11. The number of times an input is applied (connected) to the AND gates for the best performing circuit.

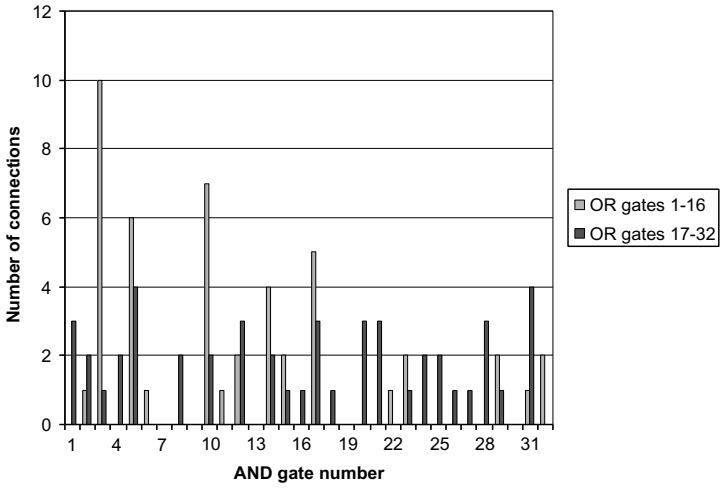

Fig. 12. The number of times each AND gates output is connected to the OR gates for the best performing circuit.

5. Conclusions

In this paper, an EHW architecture for signal classification including incremental evolution has been introduced. It is based on evolving one subsystem for each category at a time followed by a step where the complete system is further optimised with evolution.

Experiments have been undertaken on signal classification for prosthetic hand control. The best circuit evolved shows a much better performance than what was obtained by artificial neural networks. Thus, the results illustrate that this is a promising approach for evolving systems for complex real-world applications. The results also show the importance of partitioning the problem into sub-problems evolved independently. However, more experiments are necessary in the future to judge if the performance is acceptable for a prosthetic hand user.

Analysis of the best circuit shows the importance of having an architecture containing OR gates with random connections as well as allowing for incremental evolution to get the highest possible performance. However, data from a user should be collected with an emphasis on distinguishing motions that have signal responses which easily overlap.

Acknowledgments

The research is funded by the Research Council of Norway through the project *Biological-Inspired De-*

sign of Systems for Complex Real-World Applications (project no 160308/V30). The author would like to thank Dr. Kajitani at National Institute of Advanced Industrial Science and Technology (AIST) in Japan for providing the data set used in the experiments in this paper.

References

[1] E. Cantu-Paz, A survey of parallel genetic algorithms, *Calculateurs Paralleles, Reseaux et Systems Repartis* **10**(2) (1998), 141–171.

[2] S. Fuji, Development of prosthetic hand using adaptable control method for human characteristics, In *Proc. of Fifth International Conference on Intelligent Autonomous Systems*, 1998, 360–367.

[3] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[4] P. Haddow and G. Tufte, An evolvable hardware FPGA for adaptive hardware. In *Proc. of Congress on Evolutionary Computation*, 2000.

[5] W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, *Physica D* **42**(1–3) (1990), 228–234.

[6] M. Iwata, I. Kajitani, H. Yamada, H. Iba and T. Higuchi, A pattern recognition system using evolvable hardware. In *Proc. of Parallel Problem Solving from Nature IV* (*PPSN IV*), volume 1141 of *Lecture Notes in Computer Science*, Springer-Verlag, September 1996, 761–770.

[7] I. Kajitani and T. Higuchi, Developments of myoelectric controllers for hand prostheses. In *Proc. of Myoelectric Controls Symposium*, 2005, 107–111.

[8] I. Kajitani, T. Hoshino, N. Kajihara, M. Iwata and T. Higuchi, An evolvable hardware chip and its application as a multifunction prosthetic hand controller. In *Proc. of 16th National Conference on Artificial Intelligence* (*AAAI-99*), 1999, 182–187.

[9] I. Kajitani, M. Iwata, M. Harada and T. Higuchi, A myoelectric controlled prosthetic hand with an evolvable hardware lsi chip, *Technology and Disability* **15**(2) (2003), 129–143.

[10] J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, The MIT Press, 1994.

[11] W.-P. Lee, J. Hallam and H.H. Lund, Learning complex robot behaviours by evolutionary computing with task decomposition, in: *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, volume 1545 of *Lecture Notes in Artificial Intelligence*, A. Birk and J. Demiris, eds, Springer-Verlag, 1997, pp. 155–172.

[12] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata and T. Higuchi, Hardware evolution at function level. In *Proc. of Parallel Problem Solving from Nature IV* (*PPSN IV*), volume 1141 of *Lecture Notes in Computer Science*, Springer-Verlag, September 1996, 62–71.

[13] R.N. Scott and P.A. Parker, Myoelectric prostheses: State of the art, *Journal of Medical Engineering and Technology* **12**(4) (July–August 1988), 143–151.

[14] J. Torresen, A divide-and-conquer approach to evolvable hardware, in: *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, volume 1478 of *Lecture Notes in Computer Science*, M. Sipper et al., eds, Springer-Verlag, 1998, pp. 57–65.

[15] J. Torresen, Scalable evolvable hardware applied to road image recognition, in: *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn et al., eds, IEEE Computer Society, Silicon Valley, USA, July 2000, pp. 245–252.

[16] J. Torresen, Two-step incremental evolution of a digital logic gate based prosthetic hand controller, in *Evolvable Systems: From Biology to Hardware. Fourth International Conference,* (*ICES'01*), volume 2210 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001, 1–13.

[17] J. Torresen, A scalable approach to evolvable hardware, *Journal of Genetic Programming and Evolvable Machines* **3**(3) (2002), 259–282.

[18] J. Torresen and V.E. Skaugen, A signal processing architecture based on RAM technology, in *Proc. of the 16th European Simulation Multiconference* (*ESM2002*), SCS Europe, June 2002, 317–319.

[19] X. Yao and T. Higuchi, Promises and challenges of evolvable hardware, in: *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, T. Higuchi et al., eds, Springer-Verlag, 1997, pp. 55–78.

[20] M. Yasunaga, T. Nakamura, I. Yoshihara and J.H. Kim, Genetic algorithm-based design methodology for pattern recognition hardware, in: *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, J. Miller et al., eds, Springer-Verlag, 2000, pp. 264–273.