Evolution of a Brain-Computer Interface Mouse via Genetic Programming

Riccardo Poli, Mathew Salvaris, and Caterina Cinel

School of Computer Science and Electronic Engineering,
University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, UK
{rpoli,mssalv,ccinel}@essex.ac.uk

Abstract. We propose the use of genetic programming as a means to evolve brain-computer interfaces for mouse control. Our objective is to synthesise complete systems, which analyse electroencephalographic signals and directly transform them into pointer movements, almost from scratch, the only input provided by us in the process being the set of visual stimuli to be used to generate recognisable brain activity. Experimental results with our GP approach are very promising and compare favourably with those produced by support vector machines.

Keywords: Genetic Programming, Brain-Computer Interfaces, Mouse, Support-vector Machines.

1 Introduction

Over the past few years an increasing number of studies (e.g., [5,12,7,2,11]) have shown that Brain-Computer Interfaces (BCIs) are possible. These convert signals generated from the brain into control commands for devices such as computers, wheel chairs or prostheses. Such systems are often based on the analysis of brain electrical activity obtained via electroencephalography (EEG).

Within EEG-based BCIs, the analysis of P300 waves and other event related potentials (ERPs) has been particularly effective [5, 9, 1, 3]. ERPs are electrical components of brain activity produced in response to external stimuli. The P300 is a positive ERP which, in experimental conditions, can be generated by the brain when an observer attends to rare and/or significant stimuli, around 300 ms after the presentation of such stimuli. P300 waves can be used to determine user intentions in BCIs by relating their presence (or an increase in their amplitude) to specific external stimuli.

Over the years, there have been some attempts to develop BCI systems aimed at controlling 2–D pointer movements. The most successful of these, to date, are those based on the detection of μ or β rhythms in EEG [11], which require subjects to spend weeks or months training their brain to produce such rhythms, and those using invasive cortical interfaces (e.g., [4]), which require surgery.

Much less troublesome are systems based on P300 waves since they require neither user training nor invasive procedures. Some initial success in developing a P300-based mouse has been reported in [1], where rather long intervals between the end of a stimulus and the beginning of the following one were used. This resulted in the pointer moving at very slow rate (one movement every 10 seconds). Slightly better performance

(one cursor movement every 4 seconds) was achieved in [9], but accuracy in detecting P300s was only about 50% leading to many incorrect movements.

A more responsive P300-based system for the 2–D control of a cursor on a computer screen was presented in [3]. In this system four randomly-flashing squares are displayed on the screen to represent four directions of movement. Users focus their attention on the flashes of the square representing the desired direction of motion for the cursor. Flashing of the attended square produces ERPs — including P300 components — which are larger in amplitude compared to ERPs produced by the flashing of non-attended squares. Through the analyses of such brain potentials, the BCI system can then infer the user's intentions and move the cursor. The system presents three important features: it completely dispenses with the problem of detecting P300s (a notoriously difficult task) by logically behaving as an *analogue* device (i.e., a device where the output is a continuous function of the input, as opposed to a binary classifier), it uses a single trial approach where the mouse performs an action after every trial (once per second), and it *heavily relies on a genetic algorithm* for the selection of the EEG best channels, time steps and wavelet scales to use as features in the control of the mouse. The use of an analogues approach provides the system with more information about the brain state, which, in turn, makes it a more accurate, gradual and controllable mouse. We will use similar ingredients in the work presented here.

A variety of alternatives to this scheme were explored in [10] where 8 different stimuli (4 for up, down, left and right, and 4 for the 45 degree diagonal directions) were used. A linear support-vector machine (SVM) was trained to classify the ERPs produced in response to stimuli into two classes: target and non-target. After training, the raw (continuous) output produced by the SVM was used to score ERPs (the higher the score, the higher the "targetness" of an ERP). The SVM's score for each flash was then turned into a vector pointing in the direction of the stimulus and with an length proportional to the score. This was then used together with the vectors associated with other directions to determined the pointer's motion.

Despite these successes, the trajectories of the BCI-mouse pointer tend to be very convoluted and indirect. This is mainly because of the noise present in EEG signals (which is often bigger than the signal itself), the presence of eye-blinks and other muscle contractions (which produce artifacts up to two orders of magnitude bigger than the signal) and the difficulty of maintaining a user's mind focused on the target stimulus. However, the success of an analogue approach to BCI mouse design and the benefits accrued in it through the use a machine learning techniques suggest that there is more mileage in this approach. In particular, we should note that, effectively, although both in [3] and [10] feature selection was performed using powerful algorithms, only semi-linear transformations were used to transform EEG signals into mouse movements. Linear systems have obvious limitations, particularly in relation to noise and artefact rejection. So, we wondered if genetic programming (GP) [8], with its ability to explore the huge space of computer programs, could produce even more powerful transformations while also performing feature selection and artefact handling at the same time.

In this paper we report the results of our efforts to use GP as a means to evolve complete and effective BCI mouse controllers. More specifically, in Section 2 we describe the stimuli, procedure, participants and analysis performed in our BCI mouse. Section 3

describes the GP system used, its primitives, parameter settings and fitness function. In Section 4 we report our experimental results, while we provide some conclusions in Section 5.

2 BCI Mouse

Our BCI mouse uses a flashing-stimuli protocol with some similarities to the P300-based BCI mice described in the previous section. More specifically, we used visual displays showing 8 circles (with a diameter of 1.5 cm) arranged around a circle at the centre of the display as in Figure 1(far left). Each circle represents a direction of movement for the mouse cursor. Circles temporarily changed colour – from grey to either red or green – for a fraction of a second. We will call this a *flash*. The aim was to obtain mouse control by mentally focusing on the flashes of the stimulus representing the desired direction and mentally naming the colour of the flash. Flashes lasted for 100 ms and the inter-stimulus interval was 0 ms. Stimuli flashed in clockwise order. This meant that the interval between two target events for the protocol was 800 ms. We used a black background, grey neutral stimuli and either red or green flashing stimuli.

Data from 2 healthy participants aged 29 (male, normal eyesight) and 40 (female, corrected to normal eyesight) were used for GP training. The ERPs of these subjects present significant morphological differences and are affected by different types of artifacts. Each session was divided into runs, which we will call *direction epochs*. Each participant carried out 16 direction epochs, this resulted in the 8 possible directions being carried out twice.

Each direction epoch started with a blank screen and after 2 seconds the eight circles appeared near the centre of the screen. A red arrow then appeared for 1 second pointing to the target (representing the direction for that epoch). Subjects were instructed to mentally name the colour of the flashes for that target. After 2 seconds the flashing of the stimuli started. This stopped after 20 to 24 trials, with a trial consisting of the sequential activation of each of the 8 circles. In other words each direction epoch involves between $20 \times 8 = 160$ and $24 \times 8 = 192$ flashes. After the direction epoch had been completed, subjects were requested to verbally communicate the colour of the last target flash.

Participants were seated comfortably at approximately 80 cm from an LCD screen. Data were collected from 64 electrode sites using a BioSemi ActiveTwo EEG system. The EEG channels were referenced to the mean of the electrodes placed on either earlobe. The data were initially sampled at 2048 Hz, the filtered between 0.15 and 30 Hz and finally sub-sampled to 128 samples per second. Then, from each channel an 800 ms
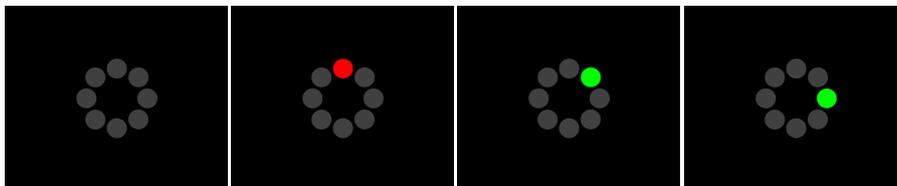

Fig. 1. Stimuli used in our BCI experiments: initial display and three sequentially flashed stimuli

epoch following each flash was extracted and further sub-sampled to 32 samples per second. This resulted in 26 data points per channel. Thus each epoch is described by a feature vector of $26 \times 64 = 1,664$ elements.

Our training set for GP included approximately 2,800 such feature vectors per subject (16 direction epochs per subject, 160–192 trials per direction epoch). Note that we didn't have a target output for each feature vector: we have only have a target direction for each of the 16 direction epochs.

3   GP System and Parameter Settings

We used a strongly-typed GP system implemented in Python with all numerical calculations done using the Numpy library (which is implemented in C). Since fitness evaluation in our domain of application is extremely computationally intensive, we created a parallel implementation which performs fitness evaluations across multiple CPU cores (via farming).

The system uses a steady-state update policy. It evolves a population of 10,000 individuals with tournament selection with a tournament size of 5, a strongly-typed version of the grow method with a maximum initial depth of 4, and strongly-typed versions of sub-tree crossover and sub-tree mutation. Both are applied with a 50% rate and use a uniform selection of crossover/mutation points. The system uses the primitive set shown in Table 1. Program trees were required to have a `Float` return type.

Table 1. Primitive set used in our application

| Primitive	Output Type	Input Type(s)	Functionality
`0.5, -0.5, 0.1, -0.1, 0, 1, ⋯, 25`	Float	None	Floating point constants used for numeric calculations and as array indexes (see below)
`Fp1, AF7, AF3, F1, ...` (60 more channel names)	Array	None	Returns an array of 26 samples following a flash from one of the channels. The samples are of type Float.
`+, -, *, min, max`	Float	(Float, Float)	Standard arithmetic operations plus maximum and minimum on floats.
`>, <`	Bool	(Float, Float)	Standard relational operations on floats
`if`	Float	(Bool, Float, Float)	If-then-else function. If the first argument evaluates to True, then the result of evaluating its second argument is returned. Otherwise the result of evaluating the third argument is returned.
`abs`	Float	Float	Returns the absolute value of a Float.
`mean, median, std, Amin, Amax`	Float	(Float, Float, Array)	Given an 26-sample Array and two floats, treat the floats as indices for the array by casting them to integer via truncation and then applying a modulus 26 operation (if the indices are identical, one is increment by 1). Then compute the mean, median, standard deviation, minimum or maximum, respectively, of the samples in the Array falling between such indices (inclusive).
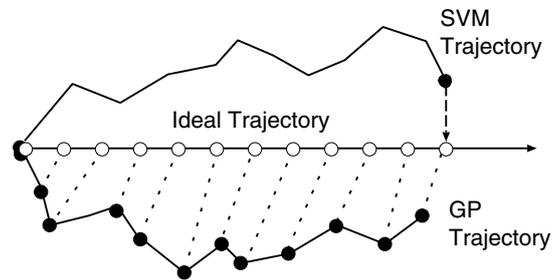
Fig. 2. Ideal and actual trajectories used in the fitness calculation. Dashed lines indicate pairs of matching points. Fitness is the average distance between such points across 16 trajectories. The end point of the ideal trajectory is computed by projecting the end point of the trajectory produced by an SVM-based flash scorer.

With this setup we performed runs of up to 50 generations, manually stopping them whenever we felt they were unlikely to make further significant progress. Because of the extreme computational load required by our fitness evaluation and the complexity of the problem (which forced us to use a relatively large population), here we only report the results of one run per subject.[1] We feel this is reasonable since we are really interested in the output produced by GP — as is the case in many practical applications of GP — rather than in optimising the process leading to such output. Each run took approximately 40 *CPU days* to complete.

Let us now turn to the fitness function we used to guide evolution. Fitness is the dissimilarity between the ideal trajectory and the actual trajectory produced by a program averaged over the 16 direction epochs. Measuring this requires executing each program for over 2,800 times. Being an error measure, fitness is, naturally, minimised in our system. We describe its elements below.

The actual trajectory produced by a program on a training epoch is obtained by iteratively evaluating the program, each time feeding the samples relating to a new flash into the Fp1, AF7, etc. terminals (which effectively act as a sliding window on the EEG). The output of the program, which, as noted above, is of type Float, is multiplied by a unit vector representing the direction corresponding to the stimulus that flashed on the screen. This produces a result of the form $(\Delta x, \Delta y)$ which is used as a displacement to be applied to the current mouse position.

As illustrated in Figure 2, the ideal trajectory for each direction epoch is obtained by sampling at regular intervals the line segment connecting the origin to a point along the desired direction. The point is chosen by projecting the end-point of the trajectory produced by a linear SVM trained on the same data as GP in the same way as in [10]. In this way, when comparing the results obtained by GP to those produced by an SVM, we ensure both had the same ideal trajectories. The ideal trajectory is sampled in such a way to have the same number of samples as the actual trajectory. The comparison between actual and ideal trajectory is then a matter of measuring the Euclidean distance between pairs of corresponding points in the two trajectories and taking an average. Notice that any detours from the ideal line and any slow-downs in the march along it in the actual trajectory are strongly penalised with our fitness measure.

---

[1] Naturally, we performed a number of smaller runs while developing the fitness function.

4   Experimental Results

Figure 3 shows the dynamics of the median and best program's fitness in our runs. The best evolved programs are presented in simplified tree form in Figure 6.[2] To evaluate their performance we will compare their output to the output produced by the SVM on the same ERP data.
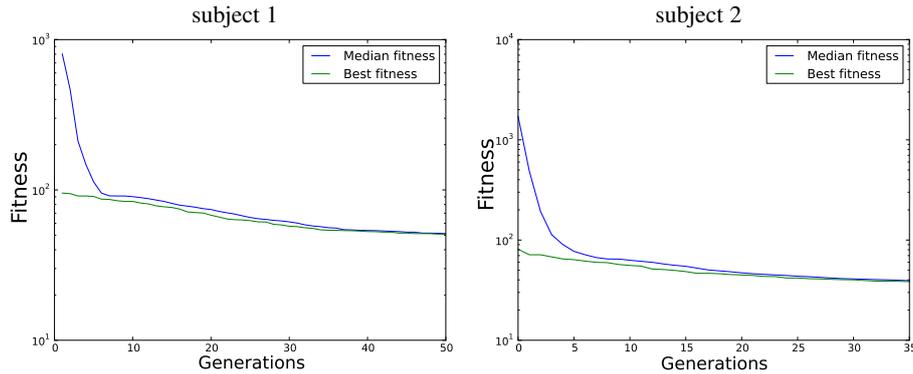

Fig. 3. Plots of the median and best fitness vs generation number in our runs

Let us start from a qualitative analysis. Figure 4(top) shows the output produced by SVM for each of the direction epochs in the training set of each subject (after the transformation of the SVM scores into $(\Delta x, \Delta y)$ displacements). Ignoring the small scales wiggles in the trajectories (which are due to the periodicity of the flashing stimuli), we see that the SVMs do a reasonably good job at producing straight trajectories. This feature is the result of the SVM being trained to respond only when the target stimulus is flashed, and to produce much smaller outputs for all other (seven) stimuli. A close inspection of the trajectories, however, reveals that while they are often straight, often they do not point exactly in the desired direction. For example, one of the two trajectories labelled as $(-0.7, 0.7)$ in subject 1 points almost directly upwards instead of pointing towards the top-left corner of the plot, one of the two trajectories labelled as $(0.7, -0.7)$ in subject 2 points almost directly downwards instead of pointing towards the bottom-right corner of the plot, and one of the two trajectories labelled as $(1.0, 0.0)$ in subject 2 points towards the top-right corner of the plot instead of pointing to the right. These biases are the result of the SVM training algorithm not being able to consider an important fact: in order for overall trajectories to point in the correct direction, the reduced (but often non-zero) output produced in the presence of flashes of stimuli that are one (two) before and one (two) after a target stimulus must be symmetric (at least on average). Overall, because of this phenomenon, SVM produces trajectories which show unsatisfactory clustering towards the 8 prescribed directions of motion.

---

[2] The programs in Figure 6 were first syntactically simplified replacing expressions involving only constants with their value and expressions of the form `(if True ExpA ExpB)` or `(if False ExpA ExpB)` with `ExpA` and `ExpB`, respectively. Then, starting from the leaves, we replaced sub-trees with their median output if the replacement influenced fitness by less than 2%.

Fig. 4. Graphical representation of the 16 sequences of SVM scores (top) and the evolved-program scores (bottom) for each of our two subjects. The tuples of numbers labelling trajectory endpoints are unit vectors indicating the desired direction of motion.

Figure 4(bottom) shows the corresponding trajectories produced by our best evolved programs. Qualitatively it is clear that these trajectories are more convoluted. This has to be expected since, unlike SVM, GP does not try to classify flashes as targets or non-targets. So, there is no explicit bias towards suppressing the output in the presence of non-target flashes. There is, however, a strong pressure in the fitness measure towards ensuring that there is overall motion in the target direction. Also, there is pressure to ensure that the outputs produced for non-target flashes are such that they either cancel out or add up in such a way to contribute to the motion in the target direction, with minimum deviations from the ideal trajectory. As a result, the trajectories produced by the best programs evolved by GP are quite close to the ideal line in each of the prescribed directions of motion. Note, for example, how close the labels at the end of each trajectory in Figure 4(bottom) are to the corresponding target directions.

To quantitatively verify these observations, Tables 2(a)–(c) show a statistical comparison between the trajectories produced by GP and those produced by SVM. More specifically: Tables 2(a) shows the mean, median, standard deviation and standard

Table 2. Statistical comparison between SVM and evolved solutions: (a) basic statistics of the distribution of distances between ideal and actual mouse trajectories, (b) *p*-values for the Kolmogorov-Smirnov one-sided two-sample test for pairwise comparison of distributions, and (c) *p*-values for the one-sided Wilcoxon signed rank test for paired data

Subject	Program	Mean	Median	Standard Deviation	Standard Error
1	GP	50.813	44.367	17.440	4.503
	SVM	52.638	43.709	30.550	7.888
2	GP	38.424	37.797	10.030	2.590
	SVM	69.766	59.997	34.518	8.913

(a)

Subject		GP	SVM
1	GP	—	0.210
	SVM	0.779	—
2	GP	—	1.000
	SVM	0.018	—

(b)

Subject		GP	SVM
1	GP	—	0.590
	SVM	0.430	—
2	GP	—	1.000
	SVM	0.000	—

(c)

error of the mean of the distances between the ideal trajectory and the actual trajectory recorded in each of the 16 direction trials of each subject; Table 2(b) reports the *p*-values for the Kolmogorov-Smirnov one-sided two-sample test for pairwise comparison of distributions; and, Table 2(c) shows the one-sided Wilcoxon signed rank test for paired data. For both subjects the evolved programs produce trajectories which are on average closer to the ideal line than the corresponding trajectories produced by SVM. In addition, GP programs shows a much more consistent behaviour having much smaller standard deviations from the ideal line than SVM. Statistically, the evolved programs produces trajectories that are significantly better than those produced by SVM for subject 2 and are on par with SVM in the case of subject 1.

Naturally, the more pronounced swirls present in the trajectories produced by the evolved programs may be a distraction for a user. It is, however, easy to remove them by post-processing the mouse movements $(\Delta x_i, \Delta y_i)$ produced by the system via a smoothing function. To illustrate the benefits of this, we have used an exponential IIR filter of the form: $(\Delta x_i^s, \Delta y_i^s) = \alpha \cdot (\Delta x_i, \Delta y_i) + (1 - \alpha) \cdot (\Delta x_{i-1}^s, \Delta y_{i-1}^s)$, with $\alpha = 1/30$ and initialisation $(\Delta x_0^s, \Delta y_0^s) = (\Delta x_0, \Delta y_0)$, which is effectively equivalent to associating a momentum to the mouse cursor. Figure 5 shows the resulting trajectories.

Clearly smoothing improves significantly the straightness of the trajectories produced by evolved programs, thereby removing the potential negative effects of the swirls in the raw trajectories. Smoothing, of course, does not improve the directional biases in the SVM-produced trajectories, although it makes them more rectilinear.

While all this is very encouraging, it is also important to see what lessons can be learnt from the evolved programs themselves. Looking at the program evolved for subject 1, we see that its output is simply the mean of a consecutive block of samples taken from channel P8. The choice of P8 is good, since this and other parietal channels are often the locations where the strongest P300 are recorded. The choice of sample 15 (which corresponds to approximately 470 ms after the presentation of the stimulus) as one of the two extremes of the averaging block is also very appropriate, since this
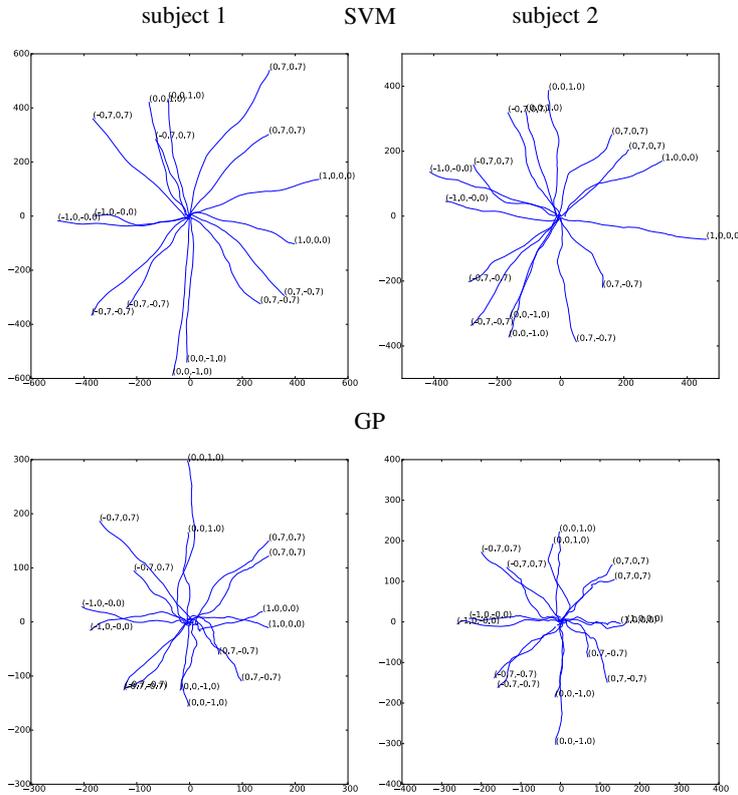
Fig. 5. Trajectories produced by SVM and GP after the application of a low-pass filter

sample falls right in the middle of typical P300 waves. The sample marking the other end of the averaging block is the result of a more elaborate choice determined by three cascaded `if` statements. Effectively this sample can take four different values: 0 (0.5 is truncated to 0), 4 (125 ms), 10 (310 ms) or 20 (625 ms). Sample 0 is chosen when the leftmost shaded sub-tree in Figure 6(left) returns `True`. Careful analysis has clarified that this tree is specialised in detecting near targets, i.e., flashes that immediately precede or follow the flashing of a target. When not in the presence of a near target, sample 4 is chosen if the second shaded sub-tree returns `True`. Analysis suggests that this sub-tree is a negated detector of eye-blinks in the early part of the data. If an eye-blink is detected, then control passes to the last `if` in the chain which moves away the averaging window from the early samples, forcing the average to cover either the range 10–15 or the range 15–20 depending on the likelihood of finding a target related P300 in the corresponding time windows (310–470 ms and 470–625 ms, respectively). This decision is taken by the right-most shaded sub-tree.

Turning now to the program evolved for subject 2, we see that it too uses the strategy of returning the average of a particular channel, FC2 in this case, where the average is between a fixed sample (13) and a second sample which is computed based on a number
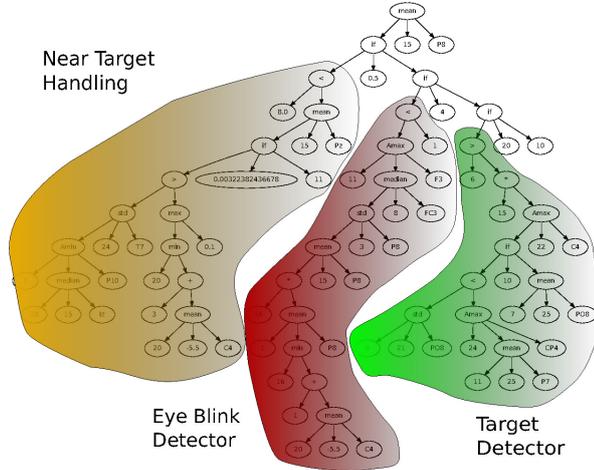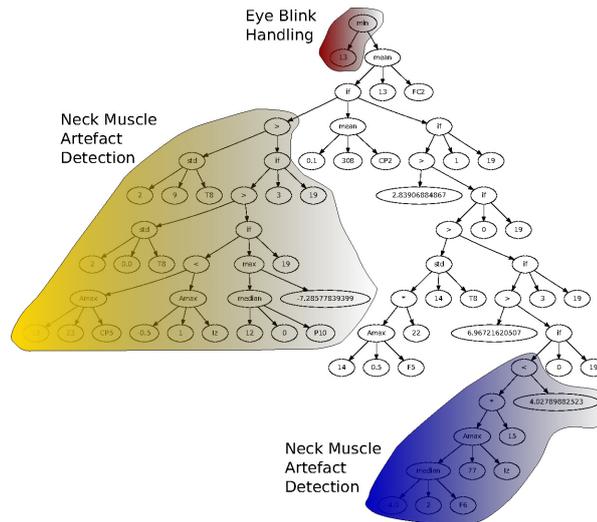
Fig. 6. Best programs evolved in our runs after syntactic and semantic simplification (see text) along with the approximate interpretation of the functionality of their sub-trees

of conditions. In this case, the output is clipped to 13. We think this clipping is an eye-blink handling strategy. Since channel FC2 is relatively frontal, the system needed to prevent eye-blinks from producing disastrous off course mouse movements. By clipping the output to 13, eye-blinks (which typically produce huge voltages for many hundreds of milliseconds, i.e., many stimulus flashes) will cause the pointer trajectory to loop, thereby never going too far off course. Note that the choice of sample 13 (406 ms) as one extreme for the windows where the mean is computed is quite appropriate since this sample, too, falls in the middle of typical P300s.

The values used as a second extreme of the averaging window depends on the result of the comparison carried out in the left-most shaded sub-tree in Figure 6(right). In this subtree electrode Iz (which is at the very base of the scalp near the neck) has a prominent role. The subtree is effectively a detector for neck movements and other muscular artifacts in the early samples of an epoch. If one such artefact is detected the second extreme for the mean is based on the calculation (mean 0.1 308 CP2) which is internally transformed into (mean 0 22 CP2). The stronger the effect of the artefact on centro-parietal areas, the more the resulting sample moves away from the beginning of the epoch, thereby avoiding the influence of spurious data in the determination of the program output. If no early muscle artefact is detected then the second extreme of the averaging block is either 1 (32 ms) or 19 (590 ms).

The decision about which sample to use is essentially made by a second artefact detection subtree (right-most shaded tree in Figure 6(right)). When activated this checks for muscle artifacts over a wider range of samples (including the end of the epoch). If none is detected, this causes a domino effect involving the five `if` statements connecting the subtree to the `mean` over FC2 instruction near the top of the program, with all such `if` statements returning 19. Sample 19 is then used as the second extreme of the averaging window for FC2. In the absence of eye-blinks, the output of the program is thus the average of FC2 over the range 406–590 ms. This makes complete sense since the range effectively covers most of the energy in the P300 wave.

5   Conclusions

Brain-computer interfaces are an exciting research area which one day will hopefully turn into reality the dream of controlling computers hands-free through intelligent interfaces capable of interpreting users' commands directly from electrical brain signals. Progress is constantly made in BCI but it is slowed down by many factors including the noise present in brain signals, muscular artefacts and the inconsistency and variability of user attention and intentions.

Recent research has shown that genetic algorithms and support vector machines can be of great help in the selection of the best channels, time steps and/or filtering kernels to use for the control of an ERP-based BCI mouse. In this paper we propose the use of genetic programming as a means to evolve BCI mouse controllers — a very difficult task that has never been attempted before.

Our objective was to synthesise complete systems, which analyse electroencephalographic signals and directly transform them into pointer movements. The only input we provided in this process was the set of visual stimuli to be used to generate recognisable brain activity. Note that there is no high-performance human-designed system for ERP-based mouse pointer control in BCI. There simply aren't any design guidelines for a domain such as this. This is why the very few systems of this kind reported in the literature all use some form of machine learning (either a GA or an SVM).

Experimental results with our approach show that GP can produce very effective BCI mouse controllers which include clever mechanisms for artefact reduction. The picture emerging from our experiments is that not only GP has been successful in the automated design of a control system for a BCI mouse, but it has also been able to

perform better than SVM — which until now has been considered perhaps the best machine-learning technology available for BCI. Additionally, GP produces controllers that can be understood and from which we can learn new effective strategies for BCI.

All this suggests that our evolved systems are at or above state-of-the-art, indicating that, perhaps, they qualify for the attribute of human-competitive, in the sense indicated by Koza (e.g., see [6]).

Acknowledgements

References

 1. Beverina, F., Palmas, G., Silvoni, S., Piccione, F., Giove, S.: User adaptive BCIs: SSVEP and P300 based interfaces. PsychNology Journal 1(4), 331–354 (2003)
 2. Birbaumer, N., Ghanayim, N., Hinterberger, T., Iversen, I., Kotchoubey, B., Kübler, A., Perelmouter, J., Taub, E., Flor, H.: A spelling device for the paralysed. Nature 398(6725), 297–298 (1999)
 3. Citi, L., Poli, R., Cinel, C., Sepulveda, F.: P300-based BCI mouse with genetically-optimized analogue control. IEEE Transactions on Neural Systems and Rehabilitation Egineering 16(1), 51–61 (2008)
 4. Donoghue, J.: Connecting cortex to machines: recent advances in brain interfaces. Nature Neuroscience 5, 1085–1088 (2002)
 5. Farwell, L.A., Donchin, E.: Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. Electroencephalography and Clinical Neurophysiology 70(6), 510–523 (1988)
 6. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3/4), 251–284 (2010)
 7. Pfurtscheller, G., Flotzinger, D., Kalcher, J.: Brain-computer interface: a new communication device for handicapped persons. Journal of Microcomputer Applications 16(3), 293–299 (1993)
 8. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (2008), Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (With contributions by J. R. Koza)
 9. Polikoff, J.B., Bunnell, H.T., Borkowski Jr., W.J.: Toward a P300-based computer interface. In: Proc. Rehab. Eng. and Assistive Technology Society of North America (RESNA 1995), pp. 178–180. Resna Press, Arlington (1995)
10. Salvaris, M., Cinel, C., Poli, R., Citi, L., Sepulveda, F.: Exploring multiple protocols for a brain-computer interface mouse. In: Proceedings of the 32nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS), Buenos Aires, September 2010, pp. 4189–4192 (2010)
11. Wolpaw, J.R., McFarland, D.J.: Control of a two-dimensional movement signal by a non-invasive brain-computer interface in humans. Proceedings of the National Academy of Sciences 101(51), 17849–17854 (2004)
12. Wolpaw, J.R., McFarland, D.J., Neat, G.W., Forneris, C.A.: An EEG-based brain-computer interface for cursor control. Electroencephalography and Clinical Neurophysiology 78(3), 252–259 (1991)