Evolutionary Synthesis of a Trajectory Integrator for an Analogue Brain-Computer Interface Mouse

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. Recently significant steps have been made towards effective EEG-based brain-computer interfaces for mouse control. A major obstacle in this line of research, however, is the integration of the noisy and contradictory information provided at each time step by the signal processing systems into a coherent and precise trajectory for the mouse pointer. In this paper we attack this difficult problem using genetic programming, obtaining extremely promising results.

Keywords: Genetic Programming, Brain-Computer Interfaces, Mouse.

1 Introduction

Over the past few years an increasing number of studies (e.g., [2,6,8,12,13,16,17]) have evaluated the possibility of converting signals generated from the brain into commands for the control of computers, wheel chairs, etc. The resulting systems go under the name of Brain-Computer Interfaces (BCIs).

BCIs are often based on the analysis of brain electrical activity recorded via electroencephalography (EEG). The EEG components most often used in BCI are the P300 wave [6] and other event related potentials (ERPs), μ or β rhythms [17], evoked potentials (EPs) [7,14,15], and others. ERPs are relatively well defined shape-wise variations to the ongoing EEG elicited by a stimulus and temporally linked to it. They include an early exogenous response, due to the sensory processing of the stimulus, as well as an endogenous response, which is a reflection of higher order cognitive processing induced by the stimulus [4]. The P300 is a positive ERP with a latency of around 300 ms which can be elicited in experimental conditions where an observer attends to a rare and/or significant stimuli (e.g., the recognition of a specific target stimulus embedded in a sequence of other non-target stimuli). This makes it possible to use P300s in BCI systems to determine user intentions.

Given the point-and-click nature of most modern user interfaces, an important application of BCI is controlling 2–D pointer movements. Over the years, there have been some attempts to develop BCI systems for this purpose, the most successful of which, to date, being those based on the detection of μ or β rhythms [16], and those using invasive cortical interfaces (e.g., [5]). The former, however, require lengthy training periods before users can control them, while the latter are not very practical, being very invasive.

C. Di Chio et al. (Eds.): EvoApplications 2011, Part I, LNCS 6624, pp. 214–223, 2011.

These problems can be overcome by systems based on the use of P300s. Some success with this approach has been reported in [1] where rather long inter-stimulus intervals led to the pointer moving at the rate of one movement every 10 seconds, and [10] where a speed of one cursor movement every 4 seconds was achieved but accuracy in detecting P300s was only about 50%.

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 devote their attention to the flashes of the square towards which the cursor should move. This produces endogenous EEG components following each stimulus, which the system analyses to infer the user's intentions and move the cursor. The system presents two unique features: it completely dispenses with the problem of detecting P300s (a notoriously difficult task) by logically behaving as an *analogue* device (as opposed to a binary classifier), and it uses a single trial approach where the mouse performs an action after every trial (once per second). The use of an analogue approach provides the system with more information about the brain state, which, in turn, makes it a more accurate, gradual and controllable mouse. However, it also opens up the problem of how to use and integrate the analogue information obtained from the brain at different time steps. In [3] the integration was simply performed by subtracting the output produced by the ERP analyser when the "up" and "down" stimuli flashed to determine the vertical displacement to be applied to the mouse cursor. The horizontal displacement was similarly obtained by subtraction of the outputs associated with the "left" and "right" stimuli.

A variety of alternatives to this approach were explored in [11] where 8 different stimuli (4 for "up", "down", "left" and "right", and 4 for the 45 degree diagonal directions) were used. Integration was based on the idea of: (a) turning each direction's flash into a vector originating from the centre of the screen, pointing in the direction of the stimulus and having an amplitude proportional to the ERP analyser's output, and then (b) performing a vector sum of the vectors associated with all 8 directions. This is effectively a generalisation of the system used in [3].

In experiments with these systems, we found that this, hand-designed, integration strategy based on vector sums does not perform optimally. In particular, because of the enormous noise present in EEG signals, muscular artifacts and the objective difficulty of maintaining a user's mind focused on the flashing stimuli, trajectories can be very convoluted and indirect. During on-line use, this causes a negative feedback loop, with subjects constantly gazing at the imperfect trajectory while attempting to improve it, thereby adding even more noise to the signals. In turn, this makes trajectories even more undirected, resulting in a confusing and, ultimately, discouraging interaction for the user. Our attempts to come up with better manually-designed integration strategies have been only partially satisfactory. So, we decided to explore the possibility of using genetic programming (GP) [9] to discover better integration strategies. In this paper we report the results of this effort.

The paper has the following structure. 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.

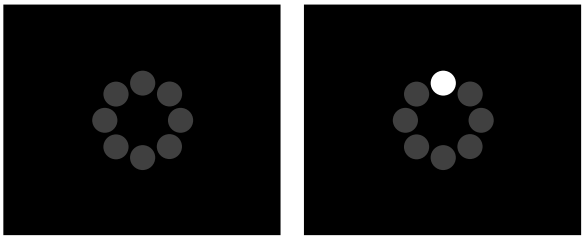

Fig. 1. Stimuli used in our BCI experiments: initial display (left) and a flashing stimulus (right)

2 BCI Mouse

Our system uses the same flashing-stimuli protocol used in the P300-based BCI mice described in the previous section. EEG signals are preprocessed and presented to a Support Vector Machine (SVM) which, for each flashed stimulus on the screen, provides an indication (a score) of how likely that stimulus was attended by the user (i.e., the intended direction of motion). Naturally, before the SVM can be used it needs to be trained. Below we describe the protocol used to gather training data for the SVM, the stimuli used, our equipment details, and the subject group used.

We adopted the SlowFlashColour protocol described in [11]. More specifically, we used visual displays showing 8 circles (with a diameter of 1.5 cm) arranged around an imaginary circle at the centre of the display as in Figure 1. Each circle represents a direction of movement for the mouse cursor. Circles flashed in random order (i.e., they temporarily changed colour – from grey to white – for a fraction of a second), similarly to other P300 based BCI protocols derived from the oddball paradigm. Mouse control was obtained by mentally focusing on the flashes of the stimulus representing the desired direction (e.g., by counting them mentally). The delay between flashes was 200 ms ($\simeq 12 \times 60^{-1}$, as permitted by the 60 Hz refresh rate of the LCD monitor used). We used black for the background, grey for the neutral stimuli and white for the highlighted stimuli. The protocol used an inter-stimulus interval of 0 ms. So, in order to avoid undesirable visual effects, stimuli adjacent to a previously flashed stimulus were prevented from flashing immediately after it. Furthermore, stimuli were not allowed to be active twice in succession. This meant that the minimum interval between two target events for the protocol was 400 ms.

Data were collected from 11 participants with an average age of 26.5. 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. Within an experiment the direction epochs were randomised.

Each direction epoch started with a blank screen and after a short period 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 count the number of flashes for that target. After 2 seconds the random flashing of the stimuli started. This stopped after between 20 and 24 trials, with a trial consisting of the activation of each of the 8 stimuli (randomised without replacement). 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, the subject was requested to verbally communicate the number of times the target stimulus flashed.

Participants were seated comfortably at approximately 80 cm from an LCD screen, their neck supported by a C-shaped inflatable travel pillow to reduce muscular artifacts. 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.

Classification was carried out using a linear SVM, trained with data collected across all the channels. The data were filtered between 0.15 and 30 Hz and initially downsampled to 128 Hz. Then, from each channel an 800 ms epoch was extracted and further decimated to 32 Hz.

3 GP System and Parameter Settings

We used a strongly-typed GP system implemented in Python. 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.

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 `Sample` return type.

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 load required by our fitness evaluation and the complexity of the problem (which forced us to use a relatively large population), in this paper we only report the results of one run. The run took approximately 30 *CPU days* to complete. 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.

Let us now turn to the fitness function we used to guide evolution. From each of the 11 subjects tested in this study, we selected one direction epoch (see previous section) for each of the 8 possible directions of motion, with the exception of one subject where both direction epochs for one direction contained huge artifacts and had to be discarded. This gave us 87 sets, each containing between 160 and 192 scores. Since, for each set, we knew the target direction, scores were converted into $(\Delta x, \Delta y)$ pairs via simple trigonometry. These data, which we will call *training arrays* hereafter, were used for fitness evaluation.

Fitness is the dissimilarity between the ideal trajectory and the actual trajectory produced by a program averaged over the 87 training arrays. Measuring this requires executing each program nearly 13,000 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 array is obtained by iteratively evaluating the program, each time feeding 32 samples of the training array

Table 1. Primitive set used in our application. The arity of the primitives can be inferred from their input type signature.

Primitive	Output Type	Input Type(s)	Functionality
`0.5`, `-0.5`, `0`, `1`, $\cdots$, `31`	Float	None	Floating point constants used for numeric calculations and as array indexes (see below)
`MouseData`	Array	None	Returns a 32-sample long window of Samples from the BCI mouse scorer block (stored in a training array). The Samples are $(\Delta x, \Delta y)$ pairs of mouse pointer displacements.
`+`, `-`, `*`	Float	(Float, Float)	Standard arithmetic operations of 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.
`normS`	Float	Sample	Given a Sample, i.e., a $(\Delta x, \Delta y)$ pair, treat it as a vector and return its norm, $\sqrt{\Delta x^2 + \Delta y^2}$.
`meanSample`, `medianSample`	Sample	(Float, Float, Array)	Given a 32-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 32 operation (if the indices are identical, one is increment by 1). Then compute the mean (median) of the samples in the Array falling between such indices (inclusive).

into the `MouseData` terminal (which effectively acts as a sliding window on the training array). The output of the program, which, as noted above, is of type `Sample`, is taken as a $(\Delta x, \Delta y)$ to be applied to the current mouse position. Integration of this time sequence produces the actual trajectory.

As illustrated in Figure 2, the ideal trajectory for each array 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, $\sum(\Delta x, \Delta y)$, of the trajectory obtained by directly executing the pointer moves in a training array onto the desired direction line. This ensures that the ideal trajectory has not only the correct direction but also a length similar to the length of the trajectory produced by the raw score data. 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 in this fitness measure.

4 Experimental Results

Figure 3 shows the dynamics of the median and best program's fitness in our run. The best evolved program is presented in tree form in Figure 4. To evaluate its performance

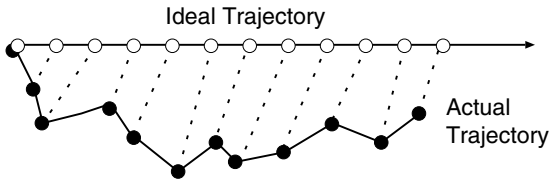

Fig. 2. Ideal and actual trajectories used in the fitness calculation. Dashed lines indicate pairs of matching points. Fitness is the average distance between such points across 87 trajectories.

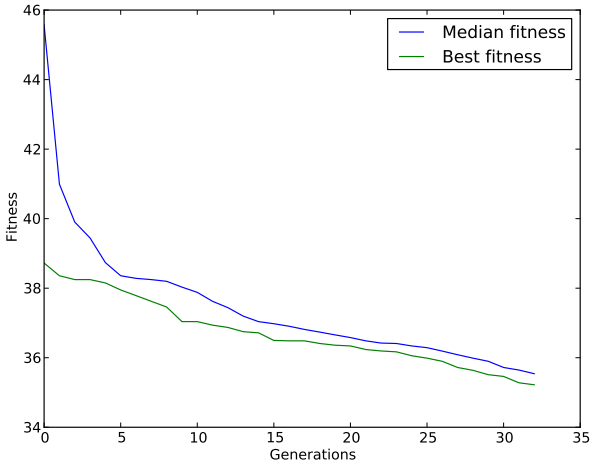

Fig. 3. Dynamics of the median and best fitness in our run

we need to compare its behaviour to both the raw input data and the output produced by the standard integrator used in previous work.

Let us start from a qualitative analysis. Figure 5(left) shows our set of 87 training arrays (sequences raw flash scores after their transformation into $(\Delta x, \Delta y)$ displacements). Note how convoluted the trajectories are and how little clustering towards the 8 prescribed directions of motion there is. Figure 5(right) shows how these data are transformed by the standard integration algorithm adopted in [3, 11]. Clearly, this technique has a positive effect on the smoothness of trajectories, but these remain very contorted and still not very close to the 8 required directions. Figure 6 shows the corresponding trajectories produced by our best evolved program. Qualitatively it is clear that these trajectories are much smoother than those in Figure 5. They also appear to cluster more towards the prescribed directions of motion.

To quantitatively verify these observations, Tables 2(a)–(b) show a statistical comparison between the raw trajectories, those produced by the standard method and those produced by the best evolved program. More specifically, Table 2(a) shows the mean, median, standard deviation and standard error of the mean of the distances between the ideal trajectory and the actual trajectory recorded in each of our 87 direction

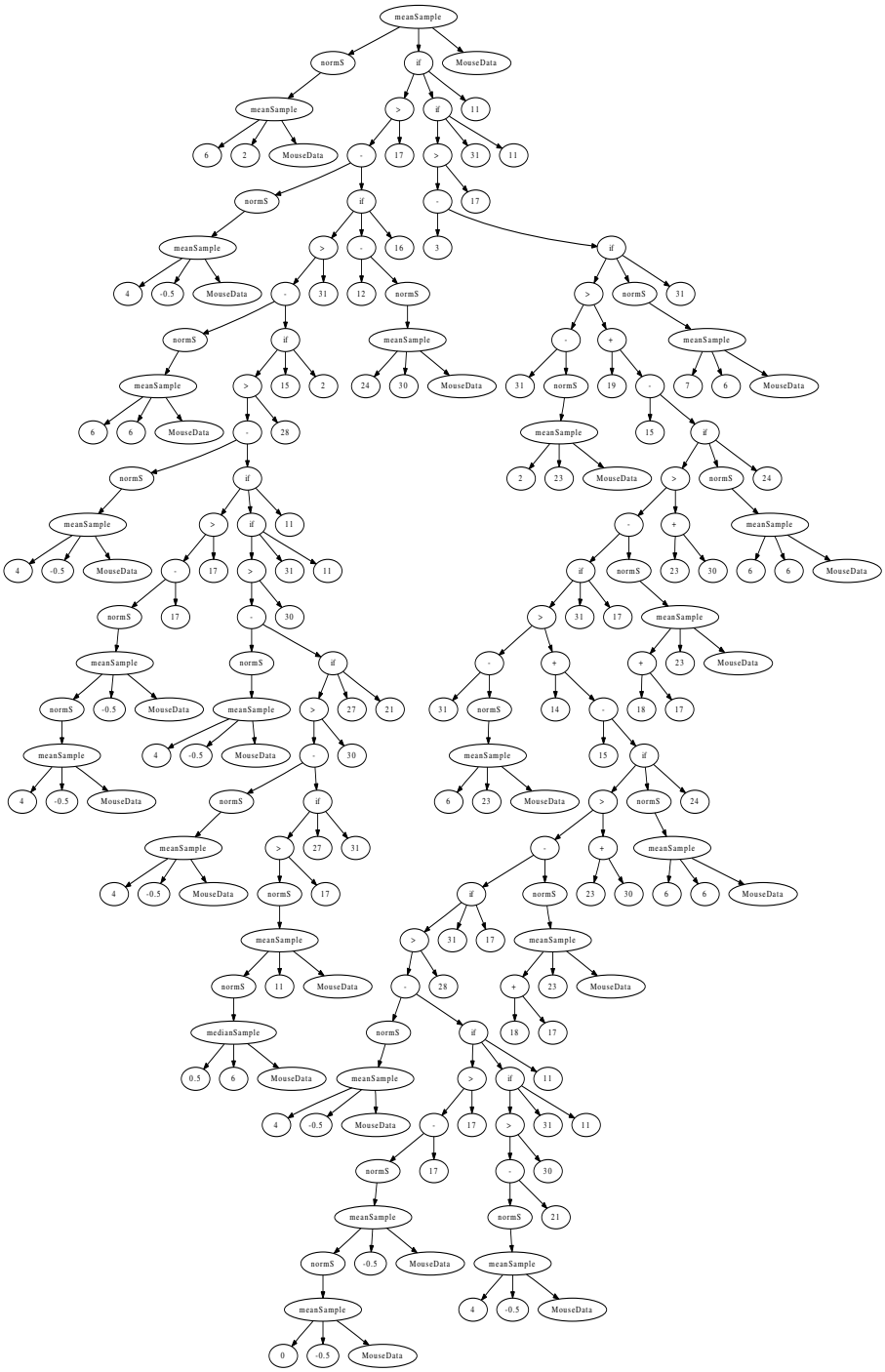

Fig. 4. Best program evolved in our runs (after minor manual simplifications)

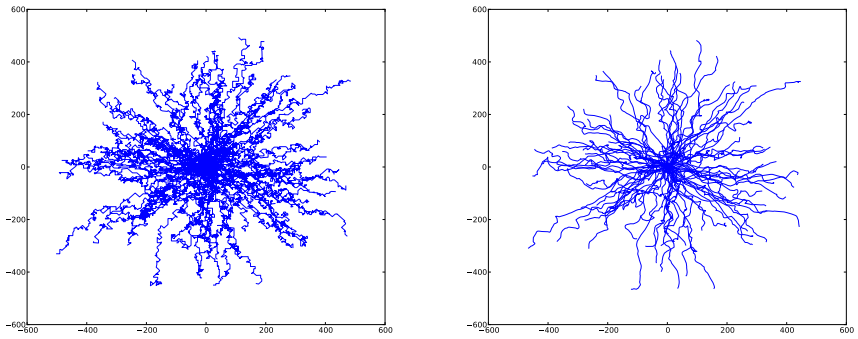

Fig. 5. Graphical representation of the 87 sequences of raw SVM scores produced by our BCI mouse (left) and the output produced by the classical technique used in previous work (right)

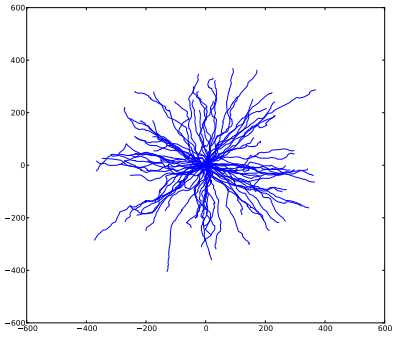

Fig. 6. Behaviour of our best evolved trajectory information integrator

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

Program	Mean	Median	Standard Deviation	Standard Error
Evolved Solution	35.2218	31.9809	17.0202	1.8353
Raw Scores	60.9022	56.5324	22.6211	2.4393
Standard Control	56.5609	51.8479	22.0939	2.3824

(a)

	Evolved Solution	Raw Scores	Standard Control
Evolved Solution	—	1.0000	1.0000
Raw Scores	0.0000	—	0.1911
Standard Control	0.0000	0.9886	—

(b)

trials. Table 2(b), instead, reports the p-values for the Kolmogorov-Smirnov one-sided two-sample test for the pairwise comparison of distributions. The evolved program produces trajectories that are better than both the raw data and the standard trajectories by a considerable margin. The difference is highly statistically significant. Surprisingly, the difference between the standard method and the raw data does not reach the significance level (although it is possible that with a larger dataset it would).

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 and the inconsistency and variability of user attention and intentions.

Recent research has made significant steps towards the achievement of an effective form of analogue BCI mouse control, but an important problem has presented itself: the integration of the noisy and contradictory information provided at each time step by the signal processing and scoring systems into a coherent and precise trajectory for the mouse pointer. In this paper we have attacked this problem using genetic programming, obtaining results that are significantly better than those obtained with the integration method described in previous work.

Acknowledgements

The authors thank EPSRC (grant EP/F033818/1) for financial support.

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., Kbler, 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 Engineering 16(1), 51–61 (2008)
4. Donchin, E., Coles, M.G.H.: Is the P300 a manifestation of context updating? Behavioral and Brain Sciences 11, 355–372 (1988)
5. Donoghue, J.: Connecting cortex to machines: recent advances in brain interfaces. Nature Neuroscience 5, 1085–1088 (2002)
6. 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)

7. Middendorf, M., McMillan, G., Calhoun, G., Jones, K.S.: Brain-computer interfaces based on the steady-state visual-evoked response. IEEE Transactions on Rehabilitation Engineering 8(2), 211–214 (2000)
8. 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)
9. 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)
10. 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), Arlington, Va, pp. 178–180. RESNA PRESS (1995)
11. 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, pp. 4189–4192 (September 2010)
12. Schwartz, A.B.: Cortical neural prosthetics. Annual Review of Neuroscience 27, 487–507 (2004)
13. Sellers, E.W., Donchin, E.: A P300-based brain-computer interface: Initial tests by ALS patients. Clinical Neurophysiology 117(3), 538–548 (2006)
14. Sutter, E.E.: The brain response interface: communication through visually-induced electrical brain responses. Journal of Microcomputer Applications 15(1), 31–45 (1992)
15. Wang, Y., Wang, R., Gao, X., Hong, B., Gai, S.: A practical VEP-based brain-computer interface. IEEE Transactions on Neural Systems and Rehabilitation Engineering 14(2), 234–239 (2006)
16. Wolpaw, J.R., McFarland, D.J.: Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans. Proceedings of the National Academy of Sciences 101(51), 17849–17854 (2004)
17. 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)