The Braincon Platform Software – A Closed-Loop Brain-Computer Interface Software for Research and Medical Applications

Dissertation zur Erlangung des Doktorgrades der technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

vorgelegt von
Jörg Daniel Fischer

aus Oberndorf am Neckar

Freiburg 08/2014

Dekan	Prof. Dr. Georg Lausen
Referenten	Prof. Dr. Gerhard Schneider (Erstgutachter) Prof. Dr. Thomas Stieglitz (Zweitgutachter)
Datum der Promotion	11. Mai 2015

Comment on References

This work contains text and figures from

> Fischer, J., Milekovic, T., Schneider, G. and Mehring, C. (2014), "*Low-latency multi-threaded pro-cessing of neuronal signals for brain-computer interfaces*", Frontiers in Neuroengineering, Vol. 7, p. 1, doi 10.3389/fneng.2014.00001

that were slightly adapted to fit in smoothly with the whole text. Such text passages are marked in *cursive font in the text body* and in *cursive font in figure captions*.

This work also contains text and figures from

> Kohler, F., Fischer, J., Gierthmuehlen, M., Gkogkidis, A., Henle, C., Ball, T., Wang, X., Rickert, J., Stieglitz, T. and Schuettler, M., *Long-term in vivo validation of a fully-implantable, wireless brain-computer interface for cortical recording and stimulation*, in preparation

that were slightly adapted to fit in smoothly with the whole text. Such text passages are marked in *cursive font in the text body* and in cursive serif font in figure captions.

Zusammenfassung

Gehirn-Computer Schnittstellen (brain-computer interfaces, BCIs) bieten eine vielversprechende Möglichkeit zur Wiederherstellung der Bewegungsfähigkeit von schwerstgelähmten Menschen, zur Kommunikation mit Patienten, die in ihrem eigenen Körper gefangen sind oder zur Verbesserung der Wirkung von Maßnahmen zur Schlaganfallsrehabilitation. Deshalb hat dieser Forschungsbereich in den letzten Jahren großes Interesse erhalten und viele Forschungsstudien erzielten gute Ergebnisse. Jedoch haben BCIs nicht den Übergang aus der Forschung zum klinischen und täglichen Heimgebrauch vollzogen. Ein medizinisches BCI muss Gehirnaktivität mindestens über mehrere Jahre stabil und sicher aufzeichnen können und dabei im Alltagsgebrauch minimale Vorbereitungszeit benötigen. Außerdem muss ein medizinisches BCI sicher für Patienten und Benutzer sein und muss als Medizinprodukt für die Verwendung am Menschen zugelassen sein. Die beiden zuletzt genannten zwei regulatorischen und sicherheitsrelevanten Anforderungen bedingen einen enormen Anstieg des Entwicklungs- und Testaufwands.

Das Ziel von Braincon ist es, ein universelles, medizinisches BCI zu sein, d.h. anwendbar für eine Vielzahl von Forschungs- und medizinischen Anwendungen, bei gleichzeitiger Reduzierung des Aufwands für Testung und für die Einhaltung der regulatorischen Vorgaben. Braincon besteht aus einem Implantat zum Aufzeichnen von Gehirnsignalen und zur elektrischen Stimulation von Gehirnarealen sowie einer Software zur Verarbeitung von Gehirnsignalen zur Laufzeit, um die Stimulationsfunktion des Implantats zu steuern.

Diese Dissertation konzentriert sich auf die Software von Braincon, d.h. die Braincon Plattform Software, und stellt eine allgemeine, flexible und verifizierbare BCI-Software-Architektur mit einer „multi-threaded“ Filter-Pipeline für die Verarbeitung von Gehirnsignalen mit niedriger Latenz. Zudem wird ein Leitfaden zu den gesetzlichen und regulatorischen Rahmenbedingungen bei der Entwicklung von medizinischer Software gegeben, zusammen mit einer Beschreibung der Teststrategie und den Testwerkzeugen, die für die regulatorische Verifizierung der Braincon Plattform Software verwendet wurden. *Die Rechenlast und die Latenz, d.h. die Zeit die ein BCI-System für die Reaktion auf eine Benutzereingabe benötigt, von verschiedenen Implementierungen der Filter-Pipeline werden für eine unterschiedliche Anzahl von Threads sowie für typische Merkmalsextraktions- und Dekodierungsalgorithmen gemessen. Die Ergebnisse zeigen, dass BCIs im Allgemeinen von der hier vorgestellten Parallelisierung profitieren können: zum einen durch die Reduzierung von Latenz, zum anderen durch Erhöhung der Aufzeichnungskanäle und Merkmale, die zur Dekodierung verwendet werden können jenseits der Menge, die durch einen einzelnen „thread“ verarbeitet werden kann.*

Die hier vorgestellte Software-Architektur wurde erfolgreich in einer BCI Studie am Menschen eingesetzt, welche deren Fähigkeit zur Online-Dekodierung von Gehirnsignalen zur Laufzeit belegt. Des Weiteren wurde Braincon in einer *in vivo* Studie im Schafsmodell erprobt. Die Ergebnisse zeigen, dass die von Braincon aufgezeichneten Gehirnsignale mit den von einem kommerziell erhältlichen, nicht implantierbaren Verstärker gemessenen Gehirnsignalen vergleichbar sind. Ein Schaf wurde chronisch implantiert und belegte die erfolgreiche Verifizierung von Braincon‘s Mess- und Stimulationsfunktionen *in vivo*. Zusammenfassend ist die Braincon Plattform Software ein flexibles und leistungsstarkes Werkzeug für die BCI Forschung und hat das Potenzial, bei minimalem regulatorischen Aufwand, die Entwicklung von BCI-basierten Behandlungsmethoden für den Menschen zur fördern.

Abstract

Brain-Computer Interfaces (BCIs) provide an auspicious opportunity for restoring movement to severely paralyzed persons, enabling communication with locked-in patients or improving efficacy in stroke rehabilitation. Therefore, this area of research has received great in interest in recent years and many research studies yielded excellent results. However, BCIs did not make the transition from research to clinical application and everyday home use. A medical BCI needs to provide the means for recording neural activity in a stable and safe manner over several years while requiring minimal preparation time for everyday use. In addition, a medical BCI has to be safe for patients and users and needs to be certified as a medical device for use with human patients. These latter two regulatory and safety requirements cause an enormous increase of effort during development and testing.

The goal of Braincon is to be a general-purpose medical BCI (i.e., applicable to a wide range of research and medical indications) while reducing the effort for testing and regulatory compliance. Braincon consists of an implant for recording of neuronal signals and for electrical stimulation of brain areas and a software that processes neural data at run-time to control the implant's electrical stimulation functionality.

This thesis focuses on the software component of Braincon (i.e., the Braincon Platform Software) and proposes a general, flexible and verifiable BCI software architecture with a filter pipeline for low-latency multi-threaded processing of neuronal signals. In addition, a guide on the juristic and regulatory environment for the development of medical software is given together with a description of the test strategy and test tools employed for a regulatorily compliant verification of the Braincon Platform Software. *The computational load and latency (i.e., the time that a BCI system needs to react to user input) are measured for different filter pipeline implementations, for different numbers of threads and for typical feature extraction and decoding algorithms from the BCI domain. Results show that BCIs in general can benefit from the proposed parallelization: firstly, by reducing the latency and secondly, by increasing the amount of recording channels and signal features that can be used for decoding beyond the amount which can be handled by a single thread.* The proposed software architecture was successfully employed in a human BCI study to show its capability for online decoding of neuronal signals. Furthermore, Braincon was put to the test in an *in vivo* sheep study. Results show that the neuronal signals recorded by Braincon are comparable to the signals recorded by a commercially available, non-implantable recording device. One sheep was chronically implanted, yielding successful verification of Braincon's *in vivo* measurement and stimulation capabilities. In conclusion, the Braincon Platform Software is a flexible and powerful tool for BCI research and has the potential to promote the development of BCI-based treatments for human patients with minimal regulatory effort.

To my family.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my supervisor, Professor Dr. Gerhard Schneider for the helpful discussions and his excellent scientific guidance. Without his encouragement, patience and invaluable experience, this thesis would not have been completed. I will miss our regular meetings on Wednesday, 11am.

I would also like to thank Professor Dr. Gerhard Schneider, Professor Dr. Thomas Stieglitz, Professor Dr. Ulrich Egert and Professor Dr. Bernd Becker for serving as committee members and for turning my defense into a positive experience which I will always remember.

I am very grateful to my fellow researchers in the research groups of Professor Dr. Carsten Mehring, Professor Dr. Thomas Stieglitz and Dr. Tonio Ball. All of them have been a great source of inspiration during the long and sometimes gloomy road to the completion of this thesis, but also provided alternative perspectives on the problems at hand and valuable scientific advice.

During my research I had the privilege to dive into fields of knowledge not typically encountered in computer science. Therefore, I would like to thank Professor Dr. Mehring and Dr. Tomislav Milekovic who introduced me into the exciting domain of brain-computer interfaces and the analysis of neural data. I am also very grateful to Fabian Kohler and Dr. Martin Schüttler for giving me insight into the art of designing brain implant hardware. My gratitude also goes to Professor Dr. Jörg Haberstroh and his team. Doing research during a neurosurgery was a rare and valuable experience for me.

I would also like to thank all patients, nurses and staff of the university clinics of Freiburg who participated or supported me in conducting the brain-computer interface study.

I am also grateful to my colleagues from CorTec GmbH, especially to its CEO, Dr. Jörn Rickert, for his continuing support of my research, to Christian Stolle for our invaluable discussions about software architecture, to Markus Raab for sharing his knowledge of regulatory affairs and to Lukas Feinweber for helping me with the intricacies of the English language.

I would also like to thank the German Federal Ministry of Education and Research for funding my research.

I thank my parents for raising me and giving me the possibility to go to university. I treasure the faith you put in me.

Finally, I would like to gratefully and sincerely thank my wife Martina and our daughters Miriam and Anna. Their support, love, care and motivation were essential and invaluable for me. Without my family, I would not have succeeded.

Table of contents

1 Introduction

Brain-Computer Interfaces (BCIs; Nicolas-Alonso and Jaime, 2012; Shih *et al.*, 2012; Wolpaw *et al.*, 2002; Vidal, 1973) promise to help patients with a wide array of diseases, disabilities and disorders.

Amyotrophic lateral sclerosis (ALS) is a neurodegenerative disease (Kiernan *et al.*, 2011) where patients become increasingly paralyzed until they lose all voluntary control over musculature (i.e., they are completely locked inside their own body while retaining full consciousness (Bauer *et al.*, 1979)). Birbaumer *et al.* trained ALS patients to voluntarily modulate certain brain rhythms, so-called slow cortical potentials. These were recorded and decoded with a BCI to operate a spelling program so that the patients could write short messages (Birbaumer *et al.*, 2000; Birbaumer *et al.*, 1999). Today, several research studies show that BCIs can be employed to re-enable locked-in patients to communicate (Oken *et al.*, 2014; Kim *et al.*, 2011).

There are 15 million patients worldwide suffering a stroke each year, of which 5 million patients remain disabled permanently (Gonzalez Andino *et al.*, 2011). Stroke patient rehabilitation can be improved by combining conventional physiotherapy with BCI training (Broetz *et al.*, 2010). Depending on the magnitude of the μ-rhythms, a brain signal related to motor activity, a BCI controls an orthosis that assists the patient's limb movements. A clinical study shows that this approach improves rehabilitation results compared to the conventional physiotherapy control group (Ramos-Murguialday *et al.*, 2013).

Hochberg *et al.* (2006) shows that a tetraplegic patient with spinal cord injury is able to use a BCI to open and close a prosthetic hand, control a computer cursor in a simple email program and operate a TV. Several research studies show that monkeys can control robotic arms with multiple degrees of freedom (Shpigelman *et al.*, 2009; Velliste *et al.*, 2008; Lebedev *et al.*, 2005). Similar results were achieved for human patients (Collinger *et al.*, 2013; Hochberg *et al.*, 2012). BCIs can also help severely paralyzed patients to regain some degree of mobility by controlling their wheelchair (Galán *et al.*, 2008; Leeb *et al.*, 2007).

BCIs typically perform four steps (Figure 1): First, they record the patient's neurophysiologic signals, for example the electroencephalogram (EEG) or neuronal spike activity and then extract features from the recorded signals. Typical features can be power in a certain frequency band or the number of detected neuronal spikes in a certain time period. An algorithm then decodes the desired information from the features (e.g., intended movement or the probability of an imminent epileptic seizure). Depending on the outcome of the decoding step, feedback is provided to the patient which can be visually (e.g., by moving a cursor on a screen), auditory or in the form of electrical stimulation applied to the brain. These steps are continuously repeated, creating a continuous closed loop of interactions between BCI and patient.

1.1 Motivation: Towards a Medical BCI

Although many research studies show good results, BCIs did not make the transition from research to clinical or every-day home use (Rouse *et al.*, 2011; Ryu and Shenoy, 2009). So far, only the Neuropace RNS® System (Morrell, 2011) has received FDA approval as a supplement for epilepsy therapy in 2013. Many promising BCI approaches stay on the level of a proof-of-concept (Silvoni *et al.*, 2013; Brunner *et al.*, 2011; Broetz *et al.*, 2010; Rebsamen *et al.*, 2010; Blakely *et al.*, 2009; Cincotti *et al.*,

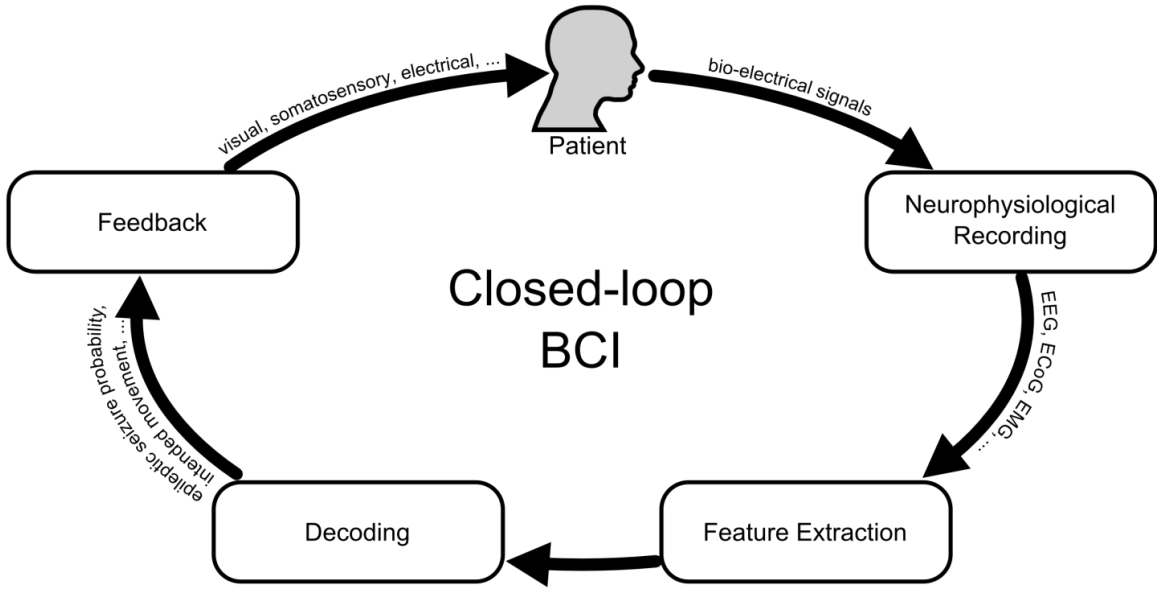


Figure 1: Components of a closed-loop BCI

Figure 1 shows typical components of a BCI depicted as rectangles and their interactions with a patient. Information flow between components is depicted as arrows annotated with examples for the information type.

2008; Galán *et al.*, 2008; Schalk *et al.*, 2008; Krepki *et al.*, 2007; Mellinger *et al.*, 2007; Wolpaw and McFarland, 2004), on the level of an animal study (Chao *et al.*, 2010; Velliste *et al.*, 2008; Lebedev *et al.*, 2005; Carmena *et al.*, 2003; Taylor *et al.*, 2002) or, as a patient-specific study, report on only one or few human patients (Collinger *et al.*, 2013; Wang *et al.*, 2013; Hochberg *et al.*, 2012; Walter *et al.*, 2012a; Hochberg *et al.*, 2006). According to Ryu and Shenoy *et al.*, a clinical BCI needs to have several properties in order to be feasible:

1. the bio-electrical signals need to be recorded stably and informatively for the time the system is in use,
2. preparation time for each session has to be as short as possible,
3. the system has to be safe for the patient and
4. the system has to be certified as a medical device for use in human patients.

1.2 Properties of a Clinical BCI

There are multiple techniques for recording brain signals, each influencing the properties of the resulting BCI regarding signal quality, stability, usability and safety. A clinical BCI is a medical device and therefore the applicable regulatory framework indirectly influences the development and design of a clinical BCI.

1.2.1 Recording Techniques

Action potentials can be recorded invasively with needle arrays from neurons, either single unit activity (SUA) or multi-unit activity (MUA) or local field potentials (LFPs, Waldert *et al.*, 2009). Electrocorticography (ECoG) records electrical activity from the surface of the brain with an epidurally or subdurally implanted foil electrode grid (Schwartz *et al.*, 2006). Electroencephalography measures the electrical field of the brain non-invasively with electrodes placed on the head (Schwartz

et al., 2006). Similar to EEG, magnetoencephalography (MEG) measures the magnetic field of the brain (Knowlton and Shih, 2004). Functional magnetic resonance imaging (fMRI) and near-infrared spectroscopy (NIRS) indirectly measure neuronal activity by measuring the oxygen saturation of the blood (Huettel *et al.*, 2008; Jöbsis, 1977). As MEG, fMRI and NIRS systems are expensive and such devices are too big for home use, these recording techniques will not be considered further.

1.2.2 Signal Quality

The recording technique has a significant influence on the spatiotemporal resolution and the information contained in the recorded signal. As a rule of thumb it can be said that the more invasive a recording technique is, the better its spatiotemporal resolution and information content (Slutzky *et al.*, 2010; Stieglitz *et al.*, 2009; Schwartz *et al.*, 2006). SUA/MUA signals are typically sampled with high frequencies up to 10 000Hz in order to capture the wave form of neuronal action potential and provide the highest fidelity of all signal types considered here. Spatial resolution of SUA/MUA activity is also very high, ranging from 0.2mm to 1mm (Schwartz *et al.*, 2006). As activity is sampled from the atomic building block of the brain (i.e., neurons) it seems plausible that SUA/MUA activity has the highest information content. Waldert *et al.* (2009) corroborates this by providing evidence that SUA/MUA activity contains more directional information than less invasive methods.

EEG signals are recorded from the scalp and due to their non-invasiveness are often the preferred choice for BCI studies. The skin, fat tissue and skull between the neurons and the electrodes attenuate and low-pass filter the signal, resulting in a lower frequency range for EEG and a typical spatial resolution around 3-5cm (Slutzky *et al.*, 2010; Stieglitz *et al.*, 2009; Schwartz *et al.*, 2006). In contrast to SUA/MUA activity, EEG provides only ambiguous localization of the neuronal source of a signal (Sitaram *et al.*, 2007) and is prone to ambient noise and muscle artifacts, especially eye movements, blinking and heartbeat (Kern *et al.*, 2013; Millan and Carmena, 2010; Ball *et al.*, 2009; Freeman *et al.*, 2000).

Electrocorticography is a kind of compromise between the high invasiveness and information content of SUA/MUA activity and the non-invasiveness and lower information content of EEG. Without the low-pass filter properties of skin, bone and fat tissue, ECoG signals have a higher amplitude and fidelity (up to 500Hz) compared to EEG (Millan and Carmena, 2010). Spatial resolution of ECoG is with 0.5cm typically between SUA/MUA and EEG (Stieglitz *et al.*, 2009; Schwartz *et al.*, 2006). Although there are artifacts (e.g., due to blood vessels, heart beat or blinking), they are weaker than in EEG (Kern *et al.*, 2013; Millan and Carmena, 2010; Ball *et al.*, 2009; Freeman *et al.*, 2000). ECoG signals contain auditory-, language- and motor-related information (Flinker *et al.*, 2011) and are therefore well suited for BCI applications.

1.2.3 Stability and Usability

The recording technique also influences the stability of the recorded signal and the effort required to perform a BCI session. SUA/MUA recordings are typically performed by inserting needle arrays into the brain. The signal quality therefore tends to degrade over time due to the formation of scar tissue and the death of neurons adjacent to the recording sites (Williams *et al.*, 2007; Otto *et al.*, 2006). In addition, recording single neurons requires rather complex spike sorting algorithms. Although there are partial advances in automation of the spike sorting, this still has to be done manually by experts before each recording session (Franke *et al.*, 2012). This makes SUA/MUA recordings rather unsuitable for long-term everyday use. EEG signal quality does not degrade over time, but preparing conven-

tional wet EEG electrodes can be very time-consuming and has to be done for every recording session (Stieglitz *et al.*, 2009). EEG electrode position changes slightly with each preparation which could require recalibration of decoding parameters. There is evidence that ECoG recordings provide stable recordings over months (Chao *et al.*, 2010; Schalk, 2010; Blakely *et al.*, 2009). Recording location for ECoG recording is selected once during the initial surgery and then the electrode grid stays in place (Harvey and Nudo, 2007) without need of daily preparation, which further supports ECoG as feasible choice for long-term everyday clinical and home use.

1.2.4 Patient Safety

Invasive recordings clearly involve a risk for the patient from the initial surgery. For implantation of needle arrays there is no broad experience so far regarding the risks for implantations in humans. However, the risk of an ECoG electrode implantation can be estimated by the risk of the established and well-known surgical intervention for treatment of medical refractory epilepsy (Ryu and Shenoy, 2009; Nair *et al.*, 2008). Electrode grids for epilepsy surgery are relatively big compared to grids intended for BCI use. Since the implantation risk also correlates with the electrode size (Wong *et al.*, 2009; Hamer *et al.*, 2002), the risk for BCI grid implantations might actually be lower compared to epilepsy surgery. There are significant risks of complication for epilepsy surgery, but they are mostly transient (Wong *et al.*, 2009). According to Bilir *et al.* (1996), there is no increased morbidity for patients who had surgical epilepsy treatment.

Conventional invasive recording techniques use cables connected to a body-external amplifier for recording. Due to these cables, there is a permanent open wound with the risk of infection increasing over time (Nair *et al.*, 2008; Hamer *et al.*, 2002). In contrast, an implant using a wireless communication and energy supply would not cause such a permanent wound and the risk of infection would be reduced. Such a solution would be a necessary prerequisite for a medical BCI for chronic use over several years.

1.2.5 Certification as a Medical Device

A medical BCI, even if it consists exclusively of software, is a medical device and therefore its development, manufacturing and certification is governed by the directive of the European Community for active implantable medical devices (AIMD, Council of the European Community, 1993a, in the latest revision from 21.09.2007). The higher the risk caused by a medical device, the higher the regulatory requirements. For implants, a quality management system compliant with EN ISO 13485 (European Committee for Electrotechnical Standardization, 2012b) has to be applied for development and manufacturing in order to guarantee that development and manufacturing processes are traceable, documented and developed using state-of-the-art processes. It has to be verified through extensive testing that the medical BCI meets the desired performance specifications. Manufacturing processes need to be verified to yield devices that meet their design specifications. Risk management according to EN ISO 14971 (European Committee for Electrotechnical Standardization, 2012a) has to be employed during development to verify that a device is designed to be safe for patient and user. Development processes, manufacturing processes and the medical device itself have to be certified to be compliant with law and applicable standards. For new medical devices, where no sufficiently comparable other certified medical devices exist, clinical studies in human patients have to be conducted to verify the device's efficacy and safety (Council of the European Community, 1990, 1993a).

1.3 Why Cortical Stimulation?

Currently much research activity is dedicated to improving existing BCI approaches as well as therapies for stroke rehabilitation, medical refractory pain and motor disorders like Parkinson's disease by employing closed-loop cortical stimulation. In all of these research areas, there is the assumption or evidence that cortical electrical stimulation should be employed, as outlined in the following text. Therefore, it would be desirable to include the ability to perform cortical stimulation into a medical BCI.

1.3.1 Parkinson's Disease

Deep brain stimulation (DBS) is an established and safe treatment for hyperkinetic movement disorders like the Parkinson's disease (Rosin *et al.*, 2011; Kenney *et al.*, 2007). Stimulation is applied continuously to the patient's brain. Rosin *et al.* report on closed-loop stimulation having greater efficacy on akinesia than open-loop stimulation. As many researchers propose the evaluation of closed-loop stimulation for Parkinson's disease treatment (Modolo *et al.*, 2012; Rouse *et al.*, 2011; Marceglia *et al.*, 2007), one can expect a demand for closed-loop implants capable of stimulation.

1.3.2 Somatosensory Feedback

The type of feedback provided by most closed-loop BCIs is often visual (LaFleur *et al.*, 2013; Milekovic *et al.*, 2012), auditory (McCreadie *et al.*, 2013; Nijboer *et al.*, 2008) or somatosensory (Chatterjee *et al.*, 2007; Cincotti *et al.*, 2007). With visual feedback, the patients are required to focus their visual attention to a screen. For example, when using a P300-based spelling device, letters are displayed that flash in turn. In order to select a letter, the patient has to concentrate on the letter, which can be quite tedious. Similarly, auditory feedback can be experienced by the patient as annoying when interfering while simultaneously talking to people. An alternative feedback method for BCIs without the aforementioned disadvantages is to use cortical stimulation (i.e., to use electrical stimulation of brain areas to elicit somatosensory sensations). This is an ongoing area of BCI research (Millan and Carmena, 2010; Wang *et al.*, 2010). In animal studies it was shown that cortical stimulation of somatosensory brain areas can be used to cue the animal's actions (Venkatraman and Carmena, 2011; Fitzsimmons *et al.*, 2007) and that different stimulation patterns can be distinguished by the animal (O'Doherty *et al.*, 2009; Schwartz *et al.*, 2006; Romo *et al.*, 2000). Tabot *et al.* (2013) report on mimicking the touching sense with electrical stimulation in an animal study. Researchers aim to use cortical stimulation as a more fine-grained feedback channel and to integrate the ability to feel into prosthesis (Millan and Carmena, 2010; Wang *et al.*, 2010).

1.3.3 Pain Therapy

Worldwide, up to 1.2 million stroke patients per year suffer from intermittent or permanent pain within the first year after stroke. Conventional treatments with analgesics rarely shows significant effects (Gonzalez Andino *et al.*, 2011; Andersen *et al.*, 1995; Bowsher, 1995). Tsubokawa *et al.* report first on achieving alleviation of neurogenic pain by stimulating the patient's motor cortex for 5-10 minutes (Tsubokawa *et al.*, 1993, 1991; Tsubokawa *et al.*, 1985). Today, cortical stimulation of the motor cortex has become a viable option for medical treatment (Brown, 2001). However, stimulation is applied constantly in an open-loop manner (i.e., largely independently from the patient's actual state of pain). The physicians select stimulation parameters by trial-and-error, which could reduce the effect of pain relief (Zuo *et al.*, 2012; Aydin, 2011). Closed-loop approaches can result in a lower daily stimulation dosage and fewer dosage-dependent adverse effects so that the resulting treatment is better tolerable by the patient (Stanslaski *et al.*, 2012; Halpern *et al.*, 2008; Osorio *et al.*,

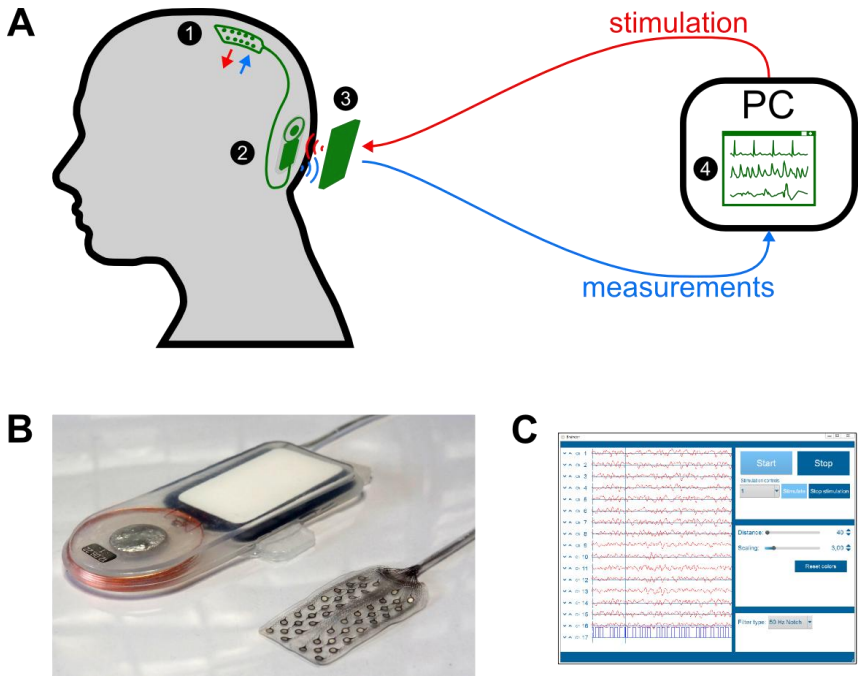


Figure 2: Braincon overview

A: Schematic overview of Braincon. It consists of a foil electrode (1), an implant (2) and a body-external unit (3) which is connected to a PC running the Braincon Platform Software (4). B: Image of Braincon implant with foil electrode. C: Screenshot of the Braincon Platform Software.

2001). Therefore, cortical closed-loop stimulation is considered auspicious for improving existing pain therapies (Edwardson *et al.*, 2013; Walter *et al.*, 2012b; Zuo *et al.*, 2012; Gonzalez Andino *et al.*, 2011; Rouse *et al.*, 2011; Plow *et al.*, 2009).

1.3.4 Stroke Motor Rehabilitation

Every year, there are 5 million people worldwide who suffering from a stroke and remaining permanently disabled. Researchers aim to improve stroke rehabilitation by using a closed-loop BCI approach (Gharabaghi *et al.*, 2014; Walter *et al.*, 2012a; Gonzalez Andino *et al.*, 2011). The basic idea is to induce neuroplasticity based on Hebbian learning by using cortical stimulation to artificially associate activities of two separate sites within the motor cortex (Wang *et al.*, 2010; Jackson *et al.*, 2006; Bütefisch *et al.*, 2004). It has been shown that rehabilitation effects of physiotherapy can be improved by applying concurrent non-invasive transcranial magnetic stimulation (TMS) to the motor cortex in an open-loop manner (Bolognini *et al.*, 2009; Plow *et al.*, 2009; Bütefisch *et al.*, 2004). However, the non-invasive TMS stimulation device limits the patient's movements during physiotherapy. The 'hot spot' on the patient's motor cortex has to be re-located for each physiotherapy session and kept fixed for the duration of each session (Harvey and Nudo, 2007). In addition, stimulation parameters might be adapted in a closed-loop approach to further improve rehabilitation effects (Gonzalez Andino *et al.*, 2011; Plow *et al.*, 2009). As intracortical stimulation mimics the effects of TMS (Bolognini *et al.*, 2009), researchers consider intracortical stimulation an attractive approach to improve stroke motor rehabilitation (Levy *et al.*, 2008; Bütefisch *et al.*, 2004).

1.4 Braincon in a Nutshell

This thesis describes Braincon with focus on its software component, the Braincon Platform Software. It consists of a hermetically encapsulated implant which could record ECoG and simultaneously perform electrical stimulation of brain areas through a foil electrode. The implant communicates wirelessly through the skin with a body-external unit. Power supply is also provided wirelessly by the body-external unit. Processing of the recorded signals and generation of stimulation commands is done on a standard personal computer (PC) running the Braincon Platform Software. For a more detailed overview of Braincon see Section 2.

1.5 Braincon Software Requirements

Braincon is intended as a tool for neuroscientists and physicians to further their understanding of the underlying neural mechanisms of movement disorders like Parkinson's disease, stroke motor rehabilitation and treatment of medical refractory pain as well as to foster research of new treatments based on a closed-loop approach for these medical indications. Promising approaches will have to go through initial preclinical tests, probably several patient-specific studies on few selected patients until the efficacy and safety of such new treatments finally have to be proven in a clinical study with human patients. The Braincon Platform Software is an integral part of Braincon, because it controls the implant by continuously reading out measurements, processing them and issuing stimulation commands. Therefore, the Braincon Platform Software has to be considered as a medical device software and needs to be developed according to the regulations and standards for active implantable medical devices (Council of the European Community, 1993b in the latest revision from 21.09.2007). For a better readability, the term medical software will be used as abbreviation for medical device software in the following. Similarly, software that does not satisfy the criteria for medical software (e.g., research software) will be termed non-medical software. In addition to regulatory requirements it was also mandatory from an ethical perspective to apply state-of-the-art development and risk management processes for developing and testing this kind of software. The intended use of the Braincon system imposed several requirements on the Braincon Platform Software, as described in the following.

1.5.1 Modular Software Architecture

Firstly, the Braincon Platform Software should be used in different clinical studies with different software components for signal processing algorithms, stimulation paradigms, user interfaces and maybe even other neurophysiologic recording devices in addition to the Braincon implant. It might even be necessary to frequently modify or exchange any of these components for each individual patient. Therefore the software architecture should support the exchange of these components as well as easy modification of existing components and the addition of new ones.

1.5.2 Medical Platform Software

Development of implants as medical devices, including software which is an integral components of the device, has high regulatory hurdles and requires large amounts of effort and money (Rouse *et al.*, 2011; Ryu and Shenoy, 2009). In the European Union, software development for medical software is regulated mainly by the standards EN ISO 13485 (quality management), EN ISO 14971 (risk management) and EN IEC 62304 (software development for medical software) These standards require extensive verification of the software to meet the design specifications and extensive validation of the software to operate with the specified efficacy (European Committee for Electrotechnical Standardization, 2006, 2012b, 2012a). Writing medical software from scratch for one specific clinical study

might therefore be close to impossible within given research budgets and might be the reason why many successful BCI research studies did not yet make the transition to clinical use (Rouse *et al.*, 2011; Ryu and Shenoy, 2009). One way to alleviate this problem could be to provide component-oriented platform software for the scientific community which was developed in compliance with regulatory requirements but can be used for a wide range of applications, as the term 'platform' implies. Then scientists could re-use existing and already extensively tested components like algorithms or user interfaces to create medical software for their needs. Development and testing effort under the regime of the regulatory requirements would be reduced to new or modified components plus the integration test and certification of the final medical device software. Using medical platform software might even be an attractive choice for research studies because due to the extensive testing the results of the research studies would be more reliable. Bearing in mind that clinical studies can follow from successful research studies, the same medical platform software can be used for these follow-up clinical studies, therefore no effort would be necessary for switching from another non-medical software to a medical software. Although non-medical software can be used in patient-specific studies on selected human patients if justifiable, the attending physician is responsible for the selection of the software tools used in such a study (Council of the European Community, 1990, 1993a). In the risk- and benefit analysis of the attending physician, a software based on a tested and safe medical platform software is likely to be preferred to pure non-medical software.

1.5.3 Low-Latency Processing

Every BCI system is limited in its applications by the time it needs to react to neurophysiologic measurements (i.e., the latency to react on user input). For motor BCIs with motor prostheses, low latency is especially important (Ryu and Shenoy, 2009), because a fast and responsive prosthesis is clearly preferable to a slow one. There were two main sources of latency for the Braincon system: Firstly, the transmission of data to and from the implant and secondly, the time needed for signal processing by the Braincon software. The latency for signal processing was composed of time needed by the signal processing algorithms, time needed for distribution of the measurement data stream to different CPU cores for processing and the time needed for thread synchronization. Therefore the goal was to reduce the latter overhead to free up computational resources for additional signal processing and to reduce latency.

1.5.4 Computational Power

Decoding neuronal activity typically involves multivariate signal processing (multiple channels and multiple signal features) and linear or non-linear methods for classification and regression. Therefore, the overall decoding process can be computationally demanding. On standard desktop computers or laptops, the amount of computations a processor core can execute per time is limited and may be insufficient for BCI signal processing. One can alleviate such limitations by distributing calculations among multiple cores. A higher amount of computational power available for decoding can be beneficial for several reasons: (i) Processing of neural activity measurements at a higher spatiotemporal resolution (i.e., more recording channels and higher sampling rates) can potentially yield more accurate decoding of the subject's intentions (Gunduz et al., 2009; Carmena et al., 2003; Nicolelis et al., 2003). Moreover, the decoding accuracy can be improved by using multiple signal features from a single neuronal signal simultaneously (e.g., by decoding from multiple frequency bands of the same local field potential or EEG channel (Kai Keng Ang et al., 2008; Woon and Cichocki, 2007; Rickert et al., 2005)). In both cases, the computational demand increases with the number of channels or the number of features per channel. (ii) More complex classification or regression algorithms can improve

decoding accuracy (e.g., by incorporating nonlinearities (Shpigelman et al., 2009; Gao et al., 2003)) or by adapting to non-stationary neuronal signals (Shpigelman et al., 2009; Wu and Hatsopoulos, 2008; Blumberg et al., 2007; Rotermund et al., 2006). Such algorithms typically require a higher amount of computational power. (iii) The computational load increases with the number of degrees of freedom (DOF) of the external actuator. For example, in the often used linear filter (Collinger et al., 2013; Schalk et al., 2008; Hochberg et al., 2006; Carmena et al., 2003), computational demand of the filter grows linearly with the number of DOFs. (iv) The Braincon Platform Software has to support future, yet unknown algorithms for signal processing which can be computationally demanding. (v) Higher computational power can increase the number of decoding steps per second. Hence, BCI users can experience a smoother control and might, therefore, feel more comfortable with the BCI. Furthermore, by reducing the time between two decoding steps, the BCI user will receive faster feedback on the movements of the actuator, which can improve the performance (Cunningham et al., 2011). In contrast, long delays and low update rates can substantially decrease the user's comfort and performance while using the interface.

1.6 Scientific Contribution

Braincon was intended to meet the criteria for a medical BCI system and to meet the demand of scientists as a research tool for stable long-term recording and cortical stimulation in human patients (Edwardson *et al.*, 2013; Zuo *et al.*, 2012; Rouse *et al.*, 2011; Marceglia *et al.*, 2007). From this data scientists expect new insights into the underlying mechanics of neural motor diseases (which are often still unknown) or psychic disorders (Maling *et al.*, 2012; Modolo *et al.*, 2012; Rosin *et al.*, 2011) or into system neuroscience in general (Bergmann *et al.*, 2012; Jensen *et al.*, 2011). The author hopes to contribute to the provision of a tool to obtain these insights which will promote development of improved treatments for patients by leveraging existing open-loop to closed-loop approaches.

However, there is still a long way to go. The development, verification and certification of Braincon are a huge endeavor and therefore still on-going. However, it is reasonable that Braincon will meet the criteria for a medical BCI: It uses ECoG recordings because they have a good signal-to-noise ratio, a high spatial resolution and promise good long-term stability (cf. Section 1.2). Implantation should be safe for the patient, based on the experience in epilepsy surgery (Wong *et al.*, 2009; Ryu and Shenoy, 2009; Bilir *et al.*, 1996). The wireless data and energy transmission reduces the risk of infection (Ryu and Shenoy, 2009) and the implant's life time is not limited by a battery. Development is conducted in compliance with regulatory requirements and standards as necessary prerequisite for Braincon's certification as a medical device.

This thesis reports on the development of the Braincon Platform Software as a medical software for BCI research. Due to the enormous monetary effort required for its development, it was necessary to do a balancing act between contributing the obtained results to the scientific community and financial requirements. Therefore Braincon Platform Software is under a closed-source license, despite the fact that the scientific community would benefit most from an open-source license. Instead, the key choices and decisions that led to its software design will be provided so that the approach is traceable for other scientists and can be used for similar undertakings. These choices and decisions had to cover regulatory requirements for medical software development. Therefore, a guide for scientists is provided describing the surrounding juristic conditions, including the major applicable standards for quality management, risk management and software life cycle and the resulting consequences for medical implant software development.

The Braincon Platform Software was developed according to the standards and regulations applicable for medical software and therefore has to be tested rigorously prior to certification. Here an account is given on the design decisions, test tools and programming paradigms used to make the software verifiable, a necessary requirement for certification.

A BCI software architecture is presented where new functionality (e.g., signal processing algorithms) can be added and tested in a fast and easy manner. It is also an efficient software architecture designed for multi-core processing of neuronal signals. Using a technique termed 'independent substream parallelization', the processing of neuronal signals can be divided into independent processing steps. Each of these steps can then be run as an individual thread, thereby making adequate use of multi-threading, prevailing protocol for parallel computations supported on desktop and laptop computers. Furthermore, this demonstrates that the performance of multi-threaded processing of neuronal signals is highly dependent on the waiting strategy (i.e., the algorithm used to exchange data between threads). Results show a trade-off between the speed at which the data is exchanged (latency) and the processor (CPU) usage: low latency comes at the cost of high CPU usage and vice versa.

The proposed BCI software architecture was employed in two research studies. Here, the software components are presented that were used to realize these studies, further corroborating the claim that the software architecture is suitable for a wide range of applications. In a closed-loop BCI study with human patients, wrist movement direction was successfully decoded. In a chronic *in vivo* animal study, the implant's measurement and stimulation capabilities were evaluated. Results show that Braincon is comparable to an established non-implantable amplifier in respect to signal quality. Stimulation from the technical perspective also works *in vivo*.

In the following text, first an overview over the whole Braincon system will be given, followed by a review of the current state of the art regarding similar software and implants. Then a summary of the juristic and regulatory environment and its consequences for the development of medical implants with focus on software development is provided as guidance for scientists. In the Methods Section, the Braincon Platform Software architecture with focus on the filter pipeline will be presented first. Then the setup for the evaluation the filter pipeline's performance is given and the software components for the closed-loop BCI study and the *in vivo* animal study are provided. The Results Section reports on the two research studies and also contains an assessment of the filter pipeline performance on typical feature extraction and decoding algorithms from the BCI domain with respect to latency and CPU load for different number of threads. Finally, implementation alternatives of the Braincon Platform Software are discussed and future work is addressed.

Figure 3: Braincon foil electrode

Electrode design implementing meander-shaped tracks and an intermediate parylene C layer. **1**: Electrode contact sites **2**: Meander-shaped tracks **3**: Silicone rubber with embedded parylene C foil **4**: Transition electrode to cable (welded). (Caption and figure from Kohler *et al.* (2012); Copyright © 2012 IEEE).

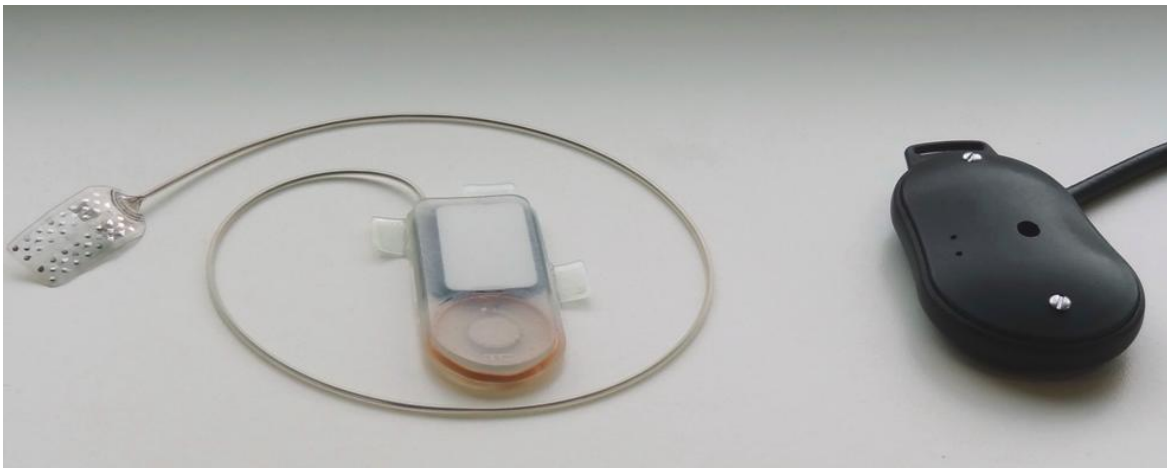


Figure 4: Braincon implant with body-external unit

Image of implant (left) and body-external unit (right).

2 Braincon Overview

The Braincon system consists of three parts: an implant, a body-external unit and the Braincon Platform Software running on a PC. The implant and body-external unit are currently subject to on-going development and optimization. Here, an overview of the implant prototype and the body-external unit used in the *in vivo* animal study is provided. The Braincon Platform Software will be described in more detail in Sections 6.1 and 6.2.

Figure 4 shows an image of the implant consisting of an ECoG electrode grid, a cable connecting the grid to a hermetic electronic package with a magnet and an inductive coil, all cast in medical grade silicone rubber.

The electrode was manufactured with a computer aided manufacturing technique for small, high density microelectrode arrays as described in Schuettler *et al.*, 2005. First, the grid layout was de-

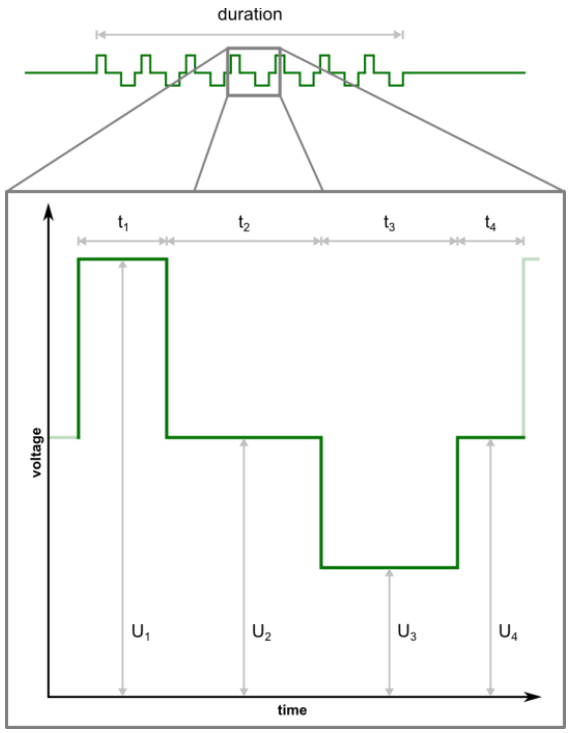


Figure 5: Scheme of stimulation parameters

Green enlarged curve shows one repetition of a stimulation wave form specified by voltages U_1, U_2, U_3, U_4 and step widths t_1, t_2, t_3 and t_4.

signed with a CAD application. Then a laser cut the grid layout from a single platinum/iridium foil which was laminated onto a layer of silicone rubber. Excess platinum/iridium was removed and a second silicone rubber layer was added. An embedded layer of parylene C provided additional mechanical stabilization (Henle *et al.*, 2011). For Braincon, a foil electrode with 32 platinum/iridium contacts (Ø = 1.12mm) suitable for either measurement or stimulation was designed (Figure 3). Two interconnected rows of seven contacts were provided as ground or reference. Overall dimensions of the electrode grid were 0.4mm x 20mm x 36mm. Due to restrictions of the implant's electronics, 16 contacts were addressable for measurement and eight contacts for electrical stimulation. These contacts were connected to the electronics inside the implant housing through a cable with a length of 50cm. The electrode and its manufacturing process are described in more detail in (Kohler *et al.*, 2012; Henle *et al.*, 2011; Schuettler *et al.*, 2005)

A microcontroller on the electronics in the implant housing was employed to manage measurement data, stimulation and communication with the body-external unit. Electronics and firmware were implemented by Multi Channel Systems GmbH (Reutlingen, Germany) and used off-the-shelf components. The electronics provided functionality for ECoG measurement on 16 channels with 1000Hz sampling frequency. Over eight additional channels, stimulation pulses could be issued for voltage stimulation of brain areas, however only one channel could stimulate at a time. A schematic of a stimulation waveform together with its constituent parameters is shown in Figure 5. Stimulation consisted of four steps with varying amplitude (0-17V) and step widths (1-10ms) which could be repeated for a specified duration. Blocking capacitors were switched between the waveform generator and the electrodes, removing the direct current (DC) component from the waveform. DC-free stimulation is required for minimizing the risk of electrode corrosion and potential biological tissue irritation. For diagnostic and safety purposes (i.e., failure prediction of the device) temperature and humidity sensors were incorporated in the implant housing.

As Braincon was intended for chronic implantation over several decades, the implant housing had to be hermetic against vapor in order to protect the electronics from body moisture, but also had to be permeable for infrared light to allow communication to the outside of the body. An exploded schematic view of the implant’s main components is shown in figure 6. The electronic components were bonded to a base substrate with screen-printed feedthroughs. These feedthroughs provided electrical connections to the inside of the hermetic housing where the electronics were located. To achieve hermeticity, a metal frame was placed between the base substrate around the electronics and a ceramic lid on top (figure 6, parts 1-3). The seams between lid, frame and base substrate were sealed

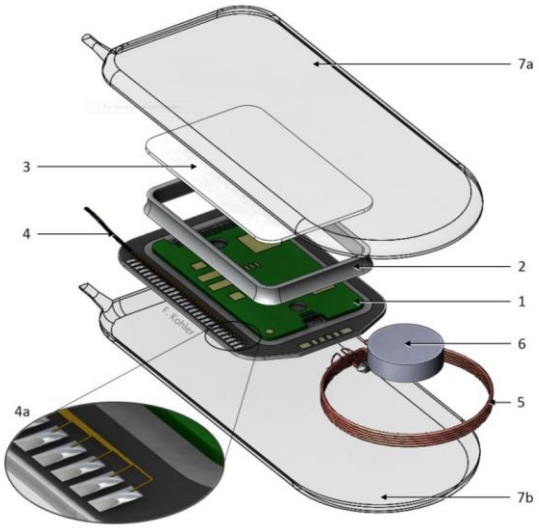


Figure 6: Exploded CAD view of the Braincon implant without electrode array

Numbers indicate the different parts of the system. 1: the base incl. screen printed feedthrough-substrate and electronics, 2: metal frame, 3: lid, 4: cable to electrode incl. substrate interconnection, 5: transmission coil, 6: alignment magnet, 7a,b: rubber molding (Caption and figure from Kohler *et al.* (2013); Copyright © 2013 IEEE).

by soldering under helium atmosphere. After the soldering process, a leak test was performed with a helium leak detector to verify hermeticity. Data transmission was realized with a half-duplex infrared optical link (1Mbit/s), allowing transmission through the ceramic lid. Energy was harvested with a transmission coil outside the implant housing (Figure 6, part 5). A body-external counterpart to this coil inside the body-external unit was placed above the implant's coil to enable an inductive energy transfer. One magnet (Figure 6, part 6) was placed in the center of each coil for easy alignment of the coils and to provide an adhesive force to fixate the body-external unit to the implant. The whole implant housing, transmission coil and magnet were cast in medical grade silicone (Figure 6, parts 7a+b) for electrical insulation and protection from a harsh body environment. The implant housing and its manufacturing process are described in more detail in Kohler *et al.* (2013) and Schuettler *et al.* (2012).

The body-external unit (Figure 3), placed above the implantation site, was also equipped with an optical infrared link for data communication with the implant, a transmission coil for energy supply and an additional trigger input was provided to mark the time of external trigger events within the measured data. Consider as an example an experiment where an optical stimulus is presented repeatedly while the implant measures the subject's brain activity. Through the trigger input, the stimulus onset could be marked within the measured data so that the brain activity at the time of stimulus onset can be reconstructed later. Firmware for the microcontroller of the body-external unit was implemented by Multi Channel Systems. The body-external unit was connected via USB to a PC or laptop where the Braincon Platform Software was executed for processing of the measurement data and issuing of stimulation commands (Figure 2 C). Depending on the application, the Braincon Platform Software can be configured for a wide range of open- and closed-loop applications.

3 State of the Art

3.1 Implants

There are several key aspects for comparing neural implants related to BCI applications. One aspect is the capability to measure and stimulate (e.g., type of measured neurophysiologic signals, number of measurement and stimulation channels, type of stimulation and sampling frequency). Another important aspect is whether the implant is fully implantable or if there are percutaneous leads. This has influence on implant size, energy supply (e.g., battery or wireless energy transmission through the skin), communication to and from the implant and implant life time. Implant size and the power supply in turn influence an implant's signal processing capability. In general, there are tradeoffs between implant size and number of channels, signal processing capabilities and energy consumption respectively battery longevity. Finally, implants differ by their intended use: there are implants for studies on cell cultures, implants for animal studies and implants which were or are being developed for use in human patients.

Many systems for *in vitro* studies on neuronal cell cultures have facilities for closed-loop measurement of neuronal activity, signal processing and electrical stimulation and can be used to explore and influence neuroplasticity. Because of this similarity to closed-loop implants, they have to be considered in this state of the art section. The **NeuroRighter Platform** is an open-source multichannel neural interfacing platform for bi-directional, real-time communication with neuronal networks (Newman *et al.*, 2013). It uses commercially available recording boards, but also provides open-source printed circuit board (PCB) layouts for voltage- and current-controlled stimulation of up to 64 channels from a microelectrode array (MEA). These layouts are modular in a sense that boards can be stacked and combined. NeuroRighter also contains a platform software running on a standard PC which is capable of closed-loop and will be discussed in more details below. The system from **Müller *et al.*** is also designed for closed-loop studies and supports stimulation while measuring. It uses a MEA with 126 measurement channels and 42 stimulation channels. The feedback loop is optimized for low latency. Configurable spike sorting and signal processing is realized using field-programmable gate arrays (FPGA) which gives the whole system hard real-time response times below 1ms (Müller *et al.*, 2013).

For animal studies in small rodents the size of the implant is important. Wires are prone to be chewed on by the animals; therefore wireless techniques for data transmission are very useful in these scenarios. **Ativanichayaphong *et al.*** (2008) report on a wireless system for recording and electrical stimulation in free-moving rodents. It has one microelectrode for measuring SUA activity with 10kHz and two microelectrodes for stimulation. Up to four different stimulation voltages with a maximum of 18V can be configured by choosing appropriate resistors on the electronics circuit. The electrodes are connected via percutaneous leads to the body-external electronics for measurement and stimulation and for wireless transmission to an up to 300m distant PC. Recording sessions are limited to six hours due to battery capacity. So far, this system was successfully employed to study closed-loop approaches for pain relief in rats (Zuo *et al.*, 2012; Aydin, 2011; Ativanichayaphong *et al.*, 2008). Another wireless system similar to the system by Ativanichayaphong *et al.* with more recording channels but no stimulation capability was developed by **Aydin** (2011). It has seven ECoG channels at 1000Hz and a battery with capacity for an operation time of up to 80 hours.

For animal studies in large animals (e.g., non-human primates) **Borton *et al.*** (2013) report on an implant with a microelectrode array that can record 100 channels of SUA/LFP activity with 7.8kHz. It has no stimulation capabilities. The hermetic titanium housing is fully implantable for more than one year. Energy supply is provided by a rechargeable battery which allows for up to seven hours of operating time. Recharging and wireless data transmission are conducted through an electromagnetic transparent single-crystal sapphire window. The whole system has been evaluated in swine and non-human primates. The implant from **Rizk *et al.*** (2009) uses a hand-assembled tungsten wire electrode array for recording SUA activity from 96 channels with 31.25kHz. It has a wireless percutaneous energy supply and uses wireless data transmission to an external PC. Due to the relatively high sampling frequency only the measurements from one channel can be transmitted. To reduce bandwidth usage, spike sorting for all channels is done inside the implant. This allows to either send the spike counts in bins of 1ms or to send all detected spike waveforms, together with a timestamp, every 50ms. So far, this implant has been evaluated in sheep in an acute setting.

The **WIMAGINE implant** is intended for human motor BCI applications according was developed compliant with the AIMD. Therefore the standards EN 45502-2-1 for cardiac pacemakers and EN IEC 62304 for firmware development were applied (Charvet *et al.*, 2013; European Committee for Electrotechnical Standardization, 2006, 2003). It is fully implantable and has a hermetic titanium housing with wireless percutaneous data transmission and energy supply. The electrode for epidural ECoG recordings is integrated at the bottom of the housing. In order to reduce energy consumption, signal acquisition is performed by two application-specific integrated circuits (ASICs), each capable of recording 32 channels at 1kHz. From the 64 channels available up to 32 can be selected for transmission to an external PC for signal processing. This implant is currently in the preclinical phase and has been verified in a rodent study (Charvet *et al.*, 2011) and a monkey BCI study (Charvet *et al.*, 2012). A long-term biocompatibility study in non-human primates showed no negative effects (Charvet *et al.*, 2013).

The **NeuroVista Seizure Advisory System** (NeuroVista Corporation, Seattle, USA) developed for prediction of epilepsy seizures in human patients. It consists of a rechargeable, long-term implantable device for measuring of 16 ECoG channels at 400Hz. Signal processing was conducted on a body-external pager-like device. The NeuroVista system's efficacy was used in preclinical research on canines and later evaluated in a clinical study with 15 human patients (Davis *et al.*, 2011; Cook *et al.*, 2013). This study was stopped in 2012 due to 'sponsor restructuring' (clinicaltrials.gov, U.S. Government).

The bi-directional **neural interface (NI) system** is designed as general purpose platform for closed-loop approaches in human patients (Rouse *et al.*, 2011; Afshar *et al.*, 2013). Its hardware is based on a certified medical device, the Activa® PC Neurostimulator (Medtronic, Minneapolis, USA), and adds ECoG measurement, acceleration measurement and signal processing capabilities. With up to two epidural or subdural electrode strips up to four channels can be used for simultaneous stimulation and measurement (Rouse *et al.*, 2011). Signal processing is performed by the implant. Each channel can either be pre-processed in the time domain with 200Hz or, for at most 2 channels, spectral band power for up to two frequency bands (DC to 500Hz) can be calculated for each channel. In addition to the fixed hardware signal processing, custom pre-processing and decoding algorithms can be applied by firmware (e.g., median filtering or support vector regression). The firmware can be flashed wireless after implantation (Stanslaski *et al.*, 2009; Rouse *et al.*, 2011). The NI system uses a non-

rechargeable battery as energy supply, therefore the amount and frequency of computations influence battery longevity. In order to reduce energy consumption, an ASIC was employed for measurement and calculation of spectral features. Battery life-time ranges from approximately two years when doing time-domain signal processing to several decades for spectral signal processing (Avestruz *et al.*, 2008; Rouse *et al.*, 2011). The NI system has additional temperature and humidity sensors for technical diagnosis and an accelerometer for detection of the patient's posture, activity or tremor state. The implant hardware was developed and verified according to IEC 60601 for safety and effectiveness of medical electrical equipment. Pre-clincial tests were done in a ovine animal model for epilepsy and in a monkey BCI study to verify closed-loop capability (Shafquat, 2011; Rouse *et al.*, 2011; Stanslaski *et al.*, 2012; Stypulkowski *et al.*, 2013). According to a press release from Medtronic, the NI system obtained certification as medical device in the EU for simultaneous measurement and stimulation with epilepsy treatment as medical indication (Medtronic, Inc., 2013). However, there was no comment on the NI system's closed-loop capabilities. This makes it unlikely that its medical indication includes closed-loop stimulation.

There are numerous implants for open-loop neuromodulation certified for medical usage in human patients, for some examples see Arle and Shils, 2011. However, to the best of the author's knowledge, only the **NeuroPace® RNS® System** (Sun *et al.*, 2008), an implantable closed-loop neurostimulator for treatment of medically refractory epilepsy, has received FDA approval in 2013. This chronic implant is powered by a battery with an expected lifetime of approximately four years and supports up to eight stimulation channels. Of these channels, up to four channels can be used alternatively for measuring ECoG. Signal processing for triggering stimulation is facilitated by the implant based on configurable parameters. Up to 30 minutes of ECoG activity can be recorded on the implant for later upload and analysis on an external PC (NeuroPace, Inc., 2013).

3.2 BCI Software Platforms

BCI software platforms can be characterized by several aspects. Their software architecture and domain model (Fowler, 2003) define the data structures for data acquisition, signal processing, patient feedback and the interactions between these data structures. As a rule of thumb, it can be said that the more generic a BCI domain model, the wider the range of supported applications is. The aspect of modularity encompasses how fine-grained functionalities of a software platform can be added, modified and exchanged. Modularity is often related to testability. Closed-loop software can either be hard or soft real-time software. Hard real-time software guarantees that the software can respond to an event within a fixed time (Kopetz, 2011). In the BCI context, this hard real-time requirement applies when studying neuroplasticity on the level of neurons. For example, Jackson *et al.*, 2006 applied electrical stimulation to one neuron after another neuron had fired with delays in the range of few milliseconds. In contrast, soft real-time software is only required to respond in the mean within a fixed time, occasional violations of this time limit are tolerable (Kopetz, 2011). Another aspect is whether the BCI software platform was developed as medical device software under the regime of medical software development.

The **NI System** provides facilities to wirelessly flash the implant's firmware (e.g., after implantation). This enables the use of different signal processing algorithms for different use-cases which is a necessary requirement for the NI system's platform claim. The firmware algorithms used are constrained by power consumption and computational power of the implant's microcontroller. The algorithms for processing of acceleration measurements were taken from a CE marked medical device, but for

certification, the whole firmware, including signal processing algorithms, has to be certified, as well as all subsequent updates (Rouse *et al.*, 2011).

The system for closed-loop MEA studies from **Müller *et al.*** (2013) uses a hybrid approach between hardware and software by using FPGAs for signal processing. Stimulation responses can be detected by an 'event engine' which can be composed of several interconnected 'modules'. Modules can be combined to elicit stimulation when certain events occur, resulting in a set of rules. A colloquially example for such a rule could be: '*IF spike detected on channel 1 AND within 1ms a spike is detected on channel 2 THEN stimulate on channel 3*'. A set of such rules is programmed on the FPGAs for subsequent signal processing. Latency is of importance when doing closed-loop studies on the level of neurons and therefore the system is optimized for low latency. Due to the usage of FPGAs signal processing can be done in hard real-time and reaches latencies of 1.25ms between the triggering of neuron activity and detection of the elicited spike (Müller *et al.*, 2013).

The **NeuroRighter platform** for MEA studies performs signal processing on a standard PC and is implemented in C# (ISO/IEC 23270:2006, International Organization for Standards, 2006). It provides a graphical user interface (GUI) for visualization of measurement data, manual control of stimulation and modular signal processing for closed-loop studies. Modularity is achieved by organizing signal processing functions in a set of tasks. Each task has one method for processing which is called either in regular time intervals or after occurrence of an event (e.g., when new measurements are available). There is a globally accessible 'DataSrv' object, from which tasks can obtain measurement data. Similarly, there is a 'StimSrv' for controlling stimulation. Both global objects provide an abstraction layer to facilitate transparent exchange of the underlying hardware. NeuroRighter is operating under soft real-time conditions, therefore response time ranges, depending on configuration, between 7.1ms and 46.9ms (Newman *et al.*, 2013).

BioSig is an open-source software library for BCI research. It is based on Matlab/Simulink (Mathworks, Natick, USA) and consists mainly of a large library of algorithms for artifact processing, adaptive signal processing, feature extraction and classification. For hard real-time BCI applications the rtsBCI package can be used (Vidaurre *et al.*, 2011; Schlögl *et al.*, 2007).

BCI++ is a framework for fast prototyping of BCI applications written in C++ and Matlab. It consists of two modules, both communicating with each other over TCP/IP. The 'Hardware Interface Module' (HIM) is used for measurement acquisition, visualization, data storage and signal processing. Algorithms for signal processing can be added to through plug-ins loadable at runtime. HIM supports signal acquisition from different devices, new devices can also be added. Graphical user feedback is provided by the second module called 'AEnima' using a 3D graphics engine which allows fast and easy creation of 3D environments. BCI++ is free of charge for researchers (Perego *et al.*, 2009).

The **Berlin BCI** (BBCI) separates signal acquisition, signal processing and user feedback into separate processes which communicate over network. Signal processing is parallelized in a fixed manner: two threads are used for feature extraction and two threads for classification. It is used for research on neuromuscular disorders, multimedia-based bio-feedback and brain-games (Krepki *et al.*, 2003, 2007).

BCI2000 is an open-source general purpose BCI research and development platform written in C++. Its main features are a common model based on interchangeable modules to represent most BCI

systems, scalability (i.e., no constraint on number of channels, sampling rate, number of signal processing steps...) and soft real-time capability. The common model defines three sequential processing steps: signal acquisition, signal processing and user feedback. Each processing step is implemented by a separate process which communicates over TCP/IP protocol with the next processing step's process. Another operator process manages configuration and starts/stops the other processes. Modularity is achieved by defining a sequence of 'operators' for each processing step at compile-time. An operator is an arbitrary function that modifies a stream of data (e.g., a band-pass filter or support vector classification algorithm). BCI2000 natively supports parallel processing of data in a sense that two different operators are applied to the same input. There are also extensions for BCI implementing multi-threaded and graphics processing unit-based (GPU-based) accelerations. This sequence of operators does not support backward connections (i.e., the output of an operator cannot be connected as input to a previous operator). However, this architectural constraint can be partially circumvented by using global state variables for inter-process backward connections. BCI2000 has been used successfully in multiple research studies (Schalk *et al.*, 2004; Schalk, 2009; Wilson and Williams, 2009).

Similarly to BCI2000, **OpenVibe** is an open-source BCI platform software for research written in C++. It has its focus on modularity and reusability and 3D virtual environments for immersive user feedback. Data acquisition is conducted by an acquisition server that supports acquisition from different, exchangeable hardware devices. Signal processing is divided into five steps: preprocessing (e.g., for artifact reduction), feature extraction, classification, translation to a command (e.g., for an external effector) and finally user feedback. Each of these steps is done by one or more ‘boxes’. A box has facilities for data input and for providing the processed output. Boxes can process data either in regular time intervals or when new data is available for processing. OpenVibe allows configuration of signal processing with visual programming techniques. Boxes and the connections between outputs and inputs are represented as rectangles and arrows respectively in a GUI called ‘scenario editor’. Connections between boxes and box properties can be set via scenario editor so that no programming skills are required to configure OpenVibe (Renard *et al.*, 2010).

4 Regulatory Requirements

This Section describes the requirements imposed on the developers of medical software (i.e., the directives, laws and standards a scientist has to comply with when creating software for a clinical study). It focuses on how these requirements influence the development of software systems for closed-loop implants. These requirements apply for all member states of the European Community. Although it can be expected that the requirements imposed by legislation of other countries are similar, it is beyond the scope of this thesis to cover and compare these. Please note that this Section is a guidance for scientists, not a legally complete description.

4.1 Juristic Environment for Medical Software Development

The European Council releases directives that regulate development and distribution of medical devices in the European Community. Johner *et al.* (2011) identify three basic directives for medical devices:

- the **M**edical **D**evice **D**irective (MDD) with amendments (Council of the European Community, 1993a; European Parliament and Council, 1998c, 2000, 2001, 2003b, 2007);
- the ***in vitro*** diagnostic medical device directive (INVITRO) with amendments and corrigenda (European Parliament and Council, 1998c, 1999, 2002, 2003b, 2009, 2011)
- and the directive for **A**ctive **I**mplantable **M**edical **D**evices (AIMD) with amendments and corrigendum (Council of the European Community, 1990, 1993a, 1993b, 1994; European Parliament and Council, 2003b, 2007)

For the sake of readability these directives will be referenced in the following text with their abbreviations (MDD, INVITRO, AIMD). Unless stated otherwise, a directive will be referenced in its latest revision (i.e., directives with all amendments and corrigenda applied).

Each member state of the European Community has to adopt and publish national laws that adopt these European directives, e.g. in Germany the 'Medizinproduktegesetz' implements all three directives (MDD article 16(1), INVITRO article 22(1), AIMD article 22(1), Bundesregierung der Bundesrepublik Deutschland, 2012).

The contents of all three directives are quite similar (Johner *et al.*, 2011). Because the AIMD applies to Braincon, this Section focuses on the AIMD as representative for the other directives. The AIMD incrementally defines the term 'active implantable medical device' in order to define its scope. It starts with the definition of the term 'medical device'. From article 1(2) a:

'*medical device*' *means any instrument, apparatus, appliance, software, material or other article, whether used alone or in combination, together with any accessories, including the software intended by its manufacturer to be used specifically for diagnostic and/or therapeutic purposes and necessary for its proper application, intended by the manufacturer to be used for human beings for the purpose of:*

- *diagnosis, prevention, monitoring, treatment or alleviation of disease,*
- *diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap,*
- *investigation, replacement or modification of the anatomy or of a physiological process,*
- *control of conception,*

and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means;'

In this definition, the classification of a device as medical device depends on the medical indication as designated by the manufacturer (e.g., a temperature sensor is not a medical device, but it would be one if the manufacturer claimed that this sensor can be used for diagnosis of fever with human patients). As laid out later in this section, the medical indication of a medical device is relevant for usability analysis, requirements analysis and risk assessment. This definition also explicitly includes software[1] as part of a medical device, used in combination with a medical device or used alone. The AIMD further defines the term 'active medical device' as medical device with '*… a source of electrical energy or any source of power …' (*AIMD article 1(2) b). It also introduces the term 'active implantable medical device' as '*... any active medical device which is intended to be totally or partially introduced, surgically or medically, into the human body or by medical intervention into a natural orifice, and which is intended to remain after the procedure;*' (AIMD Article 1(2) c). For the sake of readability, the term 'active implantable medical device' will be abbreviated with 'medical device' or simply 'device' in the following text.

The contents of the AIMD can be divided roughly in three topics.

(1) Market regulations define how medical devices can be placed on the market. In summary, all member states of the European Union allow manufacturers access to their national market if and only if their device complies with the AIMD (i.e., has the CE marking of conformity (AIMD article 1(2), article 12(1))). Please note that market access here is not restricted to selling a device, it already includes making a device '*available to medical profession for implantation*' (AIMD article 1(2) g, article 2) (i.e., for a clinical study).

(2) Conformity assessment procedures defines the procedures and the involved organizations for assessment of a medical device's conformity to the AIMD (AIMD article 4 (1), article 9). A procedure is defined for obtaining the CE-marking. Besides the manufacturer, a so-called notified body is involved in this process. A notified body is an organization appointed by any European member state to carry out conformity assessment procedures. All member states have to notify the European Commission of the bodies they appoint, hence the term notified body (AIMD article 11 (1)). During this procedure, the notified body conducts one or more audits to check if the requirements as specified in the AIMD annexes 1-4 are fulfilled. In effect, the notified body checks

- if the device satisfies the so-called 'essential requirements';
- if the manufacturer has established a quality management system suitable for design and production of active implantable medical devices;
- if the produced devices match their technical design specification;
- and verifies that the manufacturer's production processes can continuously yield devices that match their technical design specifications.

There are two procedures that allow the use of medical devices without obtaining the CE marking.

[1] The original AIMD directive AIMD did not include software. The definition of 'medical device' was amended to include software in 2007, see Council of the European Community (1990); European Parliament and Council (2007).

The first procedure allows the use of 'custom-made devices' designed specifically for one particular patient (AIMD article 1(2) d) and is used by scientists to conduct patient-specific studies. Such devices also have to fulfill the essential requirements, unless sufficient reason for not fulfilling a particular requirement is provided (AIMD annex 6 2.1). The design and manufacturing of custom made devices also has to be documented. Please note that this procedure is not suitable for systematically assessing the efficacy of a device by testing this device with multiple patients.

The second procedure allows the use of devices intended for clinical investigations (i.e., clinical studies). Since amendment 2007/47/EC in 2007 the procedure for assessment of the essential requirements demands to demonstrate a medical device's conformity with a clinical evaluation (Council of the European Community, 1990; European Parliament and Council, 2007 annex 1-I article 5a). The purpose of a clinical evaluation is to verify that under normal conditions of use the device works as expected and to determine any undesirable side effects caused by the device (AIMD annex 7 article 2.1). Clinical evaluations can be conducted legally using medical devices without CE markings, as otherwise one could never completely determine a device's conformity with the AIMD. However, a device intended for clinical investigation is required to fulfill all essential requirements except the '*… aspects constituting the object of the investigations*' (AIMD annex 6 article 2.2).

Clinical evaluation can be omitted if it can be reasoned with the notified body, based on existing clinical data, that no clinical evaluation is necessary. This can be the case if the device is similar to existing devices.

(3) Conformity requirements: The AIMD states in article 3 that the '*… active implantable medical devices referred to in Article 1(2)(c), (d) and (e), hereinafter referred to as 'devices', shall satisfy the essential requirements set out in Annex 1…*'. Please note that this demands fulfillment of the essential requirements (AIMD annex 1) from devices intended for market, custom-made devices and devices intended for clinical studies. In summary, these requirements demand that a device is safe, that the device does what it is designed to do and that during design and manufacturing of the device state of the art safety principles must be applied. Especially software '*… must be validated according to the state of the art taking into account the principles of development lifecycle, risk management, validation and verification*' (AIMD annex 1 article 9).

The formulation of the essential requirements is kept relatively general in a sense that no test procedures or critical values are provided. Instead, AIMD article 5 binds the European member states to presume a device to be compliant with the essential requirements if so-called harmonized standards were used for its design and production. Harmonized standards are standards that can be used to show compliance with European laws, regulations or administrative provisions. Similar to European directives, national standardization bodies are required to adopt harmonized standards adopted by European standardization bodies (European Parliament and Council, 1998a amended by European Parliament and Council, 1998b, 2003a; Council of the European Community, 2006). In the Official Journal of the European Union, a list of harmonized standards for the AIMD is published and maintained (European Commission, 2013).

By applying the appropriate harmonized standards, sufficient reason can be given for the presumption of a device's compliance with the AIMD's essential requirements. Please note that AIMD article 5 only suggests one way to show compliance with the essential requirements and does not prohibit other ways. Therefore, one could also try to show compliance in a different way (e.g., by using cus-

tom standards). This way, however, during the conformity assessment procedure, one would have to argue why one did not apply established state of the art principles as defined in harmonized standards. This would probably turn out to be difficult and laborious to argue with a notified body. Therefore, in the following subsection, the main applicable harmonized standards will be presented as well as the resulting consequences of applying them to the development of medical software systems for closed-loop implants.

4.2 Consequences for Medical Software Development

In the list of harmonized standards for active implantable medical devices (European Commission, 2013) three important standards were identified that apply to software development for closed-loop implants.

4.2.1 EN ISO 13485 – Quality Management System

The EN ISO 13485 (European Committee for Electrotechnical Standardization, 2012b) defines the requirements for a quality management system for an organization (e.g., a company or university department that develops and manufactures medical devices). Although it is a standalone standard, it is derived from the ISO 9001 standard for quality management systems (International Organization for Standards, 2008) and adapted for medical devices (clause 0.3.1). The three key concepts of the EN ISO 13485 relevant medical software development are its process approach, traceability of these processes and validation of requirements.

Process approach: The EN ISO 13485 adopts the definition of 'process' from ISO 9000 (International Organization for Standards, 2005) as a transformation of inputs into outputs under correlated activities. In the context of the EN ISO 13485 all activities related to medical devices (e.g., technical design of an implant or manufacturing of an implant) are seen as a set of processes which must be identified, defined, implemented and controlled (clause 4.1). When a quality management system is newly created for an organization, it is important to decide for each activity performed in that organization if this activity influences the quality, the users or the patients of the medical device. For example, the activity of buying office materials like paper and ball pens is probability highly irrelevant for a medical quality management system, but buying silicone for an implant housing intended for use in human patients is very likely an activity that needs to be executed according to a process. Regarding medical software development, all activities related to it need to be identified and written in a process description (= identified and defined). During medical software development, these processes have to be applied (= implementation). Evidence has to be provided that these processes have been implemented by documenting their progress and results (clause 4.2.4) in so-called records. The implementation of each process can be controlled (e.g., in internal audits or external audits from notified bodies) by reviewing the process' description and the records created during its execution (= control). The minimal set of activities for medical software development is described in more detail later in the subsection 4.2.3 for EN ISO 62304.

Control of documents and records: The EN ISO 13485 requires that documents and records related to the quality management system are controlled whereas in this context control means that a process for updating, reviewing, approval, distribution of documents is implemented (clauses 4.2.3 and 4.2.4). As people develop and manufacture medical devices according to the processes described in documents, any changes in these documents need to be carefully reviewed. In addition, one has to make sure that after a change to a document has been approved, all persons affected by this docu-

ment change are informed of the change. Finally, at least one copy of the old, now obsolete document has to be archived.

To illustrate this process, consider a design document that defines the software environment (i.e., operating system and patches) for a medical software system. This document shall be changed to include a new patch. The process for document control could proceed like this: first the software developers analyze the effects of adding this patch with respect to safety. Could the addition create new safety or stability hazards? If the answer is yes appropriate measures are applied. Then the document is reviewed and approved. After approval, the persons performing software tests are informed that the test environment has changed, the updated document is handed out and all obsolete copies of the old document are collected.

Please note the central role of the documentation during a quality management system activity: if some activity of a process is not recorded (i.e., there is a record from this activity) then the auditor has no evidence that this activity has been executed. In this perspective, it would be difficult to provide evidence for the safety of a medical software system during an audit if the risk analysis sessions and the resulting risk control measures were not recorded completely. In the worst case, this could mean that compliance with the AIMD cannot be shown.

Requirements validation: The requirements together with the medical indication are central inputs for risk and usability analysis (see Section 4.2.2). Therefore the EN ISO 13485 demands the requirements for a medical device to be defined (and documented) in advance for they are to be used as inputs for these design and development activities (clauses 7.3.1 and 7.3.2). The outputs of the design and development shall be verified and validated (clauses 7.3.5 and 7.3.6). In contrast to the generally applicable standard ISO 9000, the EN ISO 13485 additionally demands that the outputs of device design and development shall be validated and '*completed prior to the delivery*'. This validation includes clinical evaluations or evaluation of performance of the device if required (clause 7.3.6).

The definitions of the terms 'verification' and 'validation' vary across standards and journals, sometimes these terms are even used interchangeably (Maropoulos and Ceglarek, 2010). In the following text, we will use the definitions from EN ISO 13485[2]:

Verification: '*Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.*' (International Organization for Standards, 2005)

Validation: '*Confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled.*' (International Organization for Standards, 2005)

In the context of medical device design and development, one could say that verification tests if a device has been built according to its specifications (Was the device built as specified?) and that validation tests if a device has the performance according to its requirements (Does the device do what it should do?).

In practice this means for medical software development that requirements have to be documented (e.g., in a requirements specification document) that the requirements have to be validated after

[2] The EN ISO 13485 in turn uses the definitions of ISO 9000 by reference.

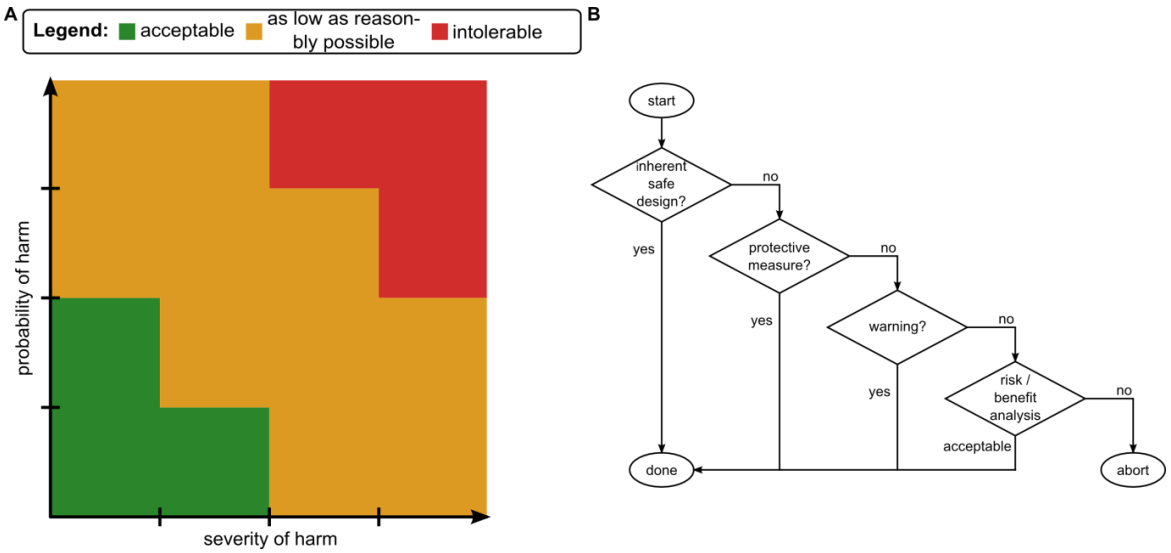


Figure 7: Risk chart and option analysis flow chart

Example of a risk chart. The axis for severity of harm and probability of harm are discretized, each into four regions. Combinations with high probability and high severity are defined as intolerable (red), combinations of low probability and low severity are defined as acceptable (green). Risks in between are defined to be kept as low as reasonably possible (orange). (B) Flowchart of option analysis for deriving risk control measures according to EN ISO 14971 clause 6.2.

completion of design and development and that the outcome of this validation is recorded. Design specifications also have to be documented and verified. For each requirement and each design specification, a test has to be defined with an objective acceptance criterion. Please note the implicit dependency between design and development requirements and their acceptance criterion: all requirements have to be formulated in a way that an objective criterion exists.

4.2.2 EN ISO 14971 – Risk Management

The EN ISO 14971 (European Committee for Electrotechnical Standardization, 2012a) addresses risk management for medical devices by defining a risk management process that should be applied throughout the whole life-cycle of a medical device (i.e., it starts with the design of a device and ends after the last device has been put out of service). All activities of the risk management process have to be documented, similarly to EN ISO 13485. The activities of the risk management process can be grouped into four parts (clause 3.2)

Risk analysis: During risk analysis, potential hazards that can result from using the device are identified. The risk of each hazard is estimated (i.e., the probability of its occurrence and the severity of the harm in case of occurrence). The standard explicitly requires using the device's intended use and medical indication as input to risk analysis (clause 4.2).

Risk evaluation: An organization developing an implantable medical device has to decide[3] which risks it defines acceptable and which are intolerable. As the EN ISO 14971 defines risk as combination of probability of occurrence of harm and severity of harm, it would be a straight forward approach to define the acceptance for each combination, yielding a so-called 'risk chart', cf. Figure 7 A for an example of a risk chart. Please note that the EN ISO 14971 neither requires the grouping of probability

[3] The decision is usually approved by the management.

and severity into discrete classes, nor does it define what acceptable risk levels are (EN ISO 14971 clause 1). It is in the responsibility of the organization applying the EN ISO 14971 to define an appropriate risk chart.

Risk control: For the identified risks defined as acceptable during risk evaluation no control measures have to be implemented. Intolerable risks have to be controlled to reduce the remaining risk (i.e., risk control measures have to be defined that reduce a hazard's severity or probability of occurrence so that it becomes acceptable). In addition, '*risk control measures shall be reviewed to identify if other hazards are introduced*' (EN ISO 14971 clause 6.6). The EN ISO 14971 defines a priority order of application on the type of risk control measures (clause 6.2), cf. Figure 7 B. If possible, the risk shall be removed completely by making the design inherently safe. Otherwise, protective measures should be applied in the medical device itself, otherwise information for safety can be provided (e.g., a warning in the manual). If the residual risk is still intolerable after applying all risk control measures, the EN ISO 14971 allows a risk/benefit analysis to determine if the medical benefits outweigh the residual risk (clause 6.4). This decision has to be recorded. In the worst case, the benefits do not outweigh the risk and the whole device development has to be aborted. As an example, consider a heart defibrillator. When used, there is always a chance that the patient dies from defibrillation. But as defibrillation is only applied to get the patient out of an already life-threatening state, the medical benefit outweighs the risk. All risk control measures shall be implemented and their effectiveness has to be verified (clause 6.3). After implementation and verification of a risk control measure, the residual risk shall be evaluated (clause 6.4). If necessary, further risk control measures have to be applied.

Post-production information: After development and production of a medical device, it is put into service. The EN ISO14971 defines that in this life cycle phase information gained from the usage of the medical device and similar devices shall be reviewed continuously, especially for previously unrecognized hazards, if the risk evaluations during design and development are still valid (EN ISO 14971 clause 9).

Although the risk management process described by this standard is generally applicable, some modifications apply to this general process for medical software development as can be seen in the following section.

4.2.3 EN 62304 – Software Life-Cycle Processes for Medical Device Software

The essential requirements of the AIMD demand that '*the software must be validated according to the state of the art taking into account the principles of development lifecycle, risk management, validation and verification*' (Council of the European Community, 1990; European Parliament and Council, 2007 Annex I Article 9). This requirement can be fulfilled by applying the EN 62304 (Annex ZZ) which defines a set of requirements and activities for software life cycle processes for medical software. Please note that the term 'software life cycle processes' includes the time after design and development (i.e., the time when the software has been released). The EN 62304 does not '*prescribe a specific life cycle model*' (EN 62304 page 7), it only poses requirements a life cycle model has to fulfill. As a consequence, one can freely choose the software development model (e.g., waterfall model (Royce, 1970; Boehm, 1981)) V-model XT (Bundesministerium des Inneren, 2014), SCRUM (Schwaber, 1995) or even extreme programming (Beck and Andres, 2005), if and only if these models incorporate the required activities and processes.

The EN 62304 also imposes some general demands on the environment in which medical software is developed and maintained. Firstly, one shall demonstrate that the medical software meets user requirements and applicable regulatory requirements. Although the standard leaves the actual fulfillment of this requirement open, it recommends the application of ISO 13485 to establish a quality management system (clause 4.1). Secondly, the EN 62304 explicitly demands to '*apply a risk management process complying with ISO 14971*' (clause 4.2). Similarly to the EN ISO 13485, the EN 62304 demands traceability from user requirements to the resulting source code and the respective verification tests and also demands traceability from identified risks over risk control measures to the verification tests to the released software.

In the context of the EN 62304, a medical software system is divided into software items and software units. A software item is defined as '*any identifiable part of a computer program*' (clause 3.25). A software unit is a '*software item that is not subdivided into other items*' (clause 3.28). It is up to the developer to define what comprises a software unit and a software item for the actual software system and the used programming language(s). According to these definitions, a medical device can include several software systems (e.g., one software system used for therapeutic treatment by nurses or patients and one for diagnosis by physicians). Each software system, software item and software unit has an associated software safety class, either 'A', 'B' or 'C'. For the sake of brevity and readability, the term 'software safety class' will be shortened to 'safety class' in the following text. The safety classes are defined as follows (clause 4.3a):

- Class A: No injury or damage to health is possible
- Class B: Non-serious injury[4] is possible
- Class C: Death or serious injury is possible

The bigger the potential harm that can be caused by a software system, item or unit, the higher is its safety class. Required activities demanded by the EN 62304 vary depending on the software safety class of a software system, item or unit. The EN 62304 requires more activities for software systems, items and units with safety class C than for class B and more for class B than for class A, cf. EN 62304 annex A for an overview. Software items (and systems) inherit the maximum safety class from the software items and units they are composed of, cf. Figure 8 for an example.

4.2.3.1 Software Development Process

Prior to the actual development of a software system, development has to be planned in a document called software development plan. This plan will be updated and maintained throughout the whole development process. It contains a description of the software development life cycle model and the outputs (e.g., documents) of the activities resulting from software development. The software development plan also addresses software configuration management, change management, software integration and how software problems are resolved (clause 5). If there are already existing process definitions, (e.g., provided by a quality management system according to EN ISO 13485), the software development plan can refer to these process descriptions instead of providing complete process descriptions.

[4] In informal language, serious injury is an injury that permanently damages or impairs a person. It also includes injuries that need medical or surgical treatment to prevent such damage (Article 3.23).

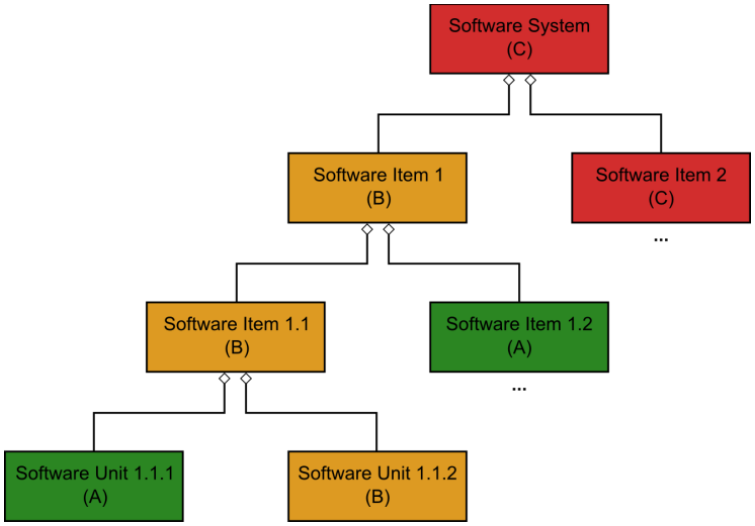


Figure 8: Division of a software system into safety classes

Example of the division of a software system into software items and units according to EN 62304. Rectangles represent parts of the software system together with the associated safety class (A, B or C) in brackets. Arrows with diamond tip indicate a composition relation, e.g. software unit 1.1.1 is part of software item 1.1.

After planning, requirements are identified and risk management is started. The identified requirements and risk control measures are transformed into an architectural design which contains a description of the structure of the software by defining the software items and describing the interfaces these software items expose to other software items (clause 5.3). With a risk analysis, a safety class is derived for each software item. Based on the safety class, a so-called detailed design is created, where software items with safety class B or C are recursively refined to the level of software units. Please note that the granularity of the detailed design strongly depends on the definition of 'software unit' for the software system.

After top-down decomposition of the software system into software units and planning the software architecture and detailed design, the implementation and integration is done. First, all software units are implemented. For each software unit with safety class B or C, acceptance criteria have to be defined and verified (clauses 5.5.2 and 5.5.3). Integration of software units into software items and finally the software system is done according to procedures defined in the software development plan (clauses 5.1.5 and 5.6). After integration, one has to test if the software system meets the defined requirements *a priori* (clause 5.7). The software system can be released if it passes all tests. If it does not, some anomalies remain and have to be evaluated regarding their risk. If this risk evaluation shows that these anomalies do not contribute to an unacceptable risk, the software can be released despite the anomalies (clause 5.8). After release the software maintenance process starts.

4.2.3.2 Software Problem Resolution and Change Control Processes

Problems can be detected by users and developers (e.g., when the software fails to meet its requirements or new risks are discovered). There might also be new features that need to be incorporated in the existing software system. This can occur after a software system has been released or during development and may require a change of the software system. The two processes described in this subsection aim to avoid the introduction of new risks through these changes and that the changed software system still meets its requirements.

For each problem, a problem report has to be created in which the scope of the problem is defined (i.e., affected parts of source code and hardware / software configurations, clause 9.1). Each problem report is evaluated with respect to the problem's relevance to safety. Based on this evaluation, a change request is created defining what actions have to be taken to solve the problem (clause 9.2). A change might introduce new risks, therefore it might be necessary to perform a risk analysis as part of the change request creation. If and only if the change request has been approved for implementation, then the changes are applied according to the change control process (clause 9.4).

Changing a software system might invalidate some previously obtained results of verifications or validations or change the safety class of a software item. Therefore implementing a change always includes three steps: the actual implementation of the changes, then the identification of activities that have to be repeated due to the change and finally the repetition of change dependent activities including the verification of the change itself (clauses 8.2.2 and 8.2.3).

4.2.3.3 Software Maintenance Process

The EN 62304 explicitly requires software to be maintained after release by monitoring the feedback reports from its users (clause 6.1). Prior to release, it has to be defines how feedback is received, evaluated and resolved. Every feedback that reports on adverse events or deviations from specification is considered as a problem and a problem report is created. Risk analysis has to be applied in order to determine whether the releases software system needs to be changed (clause 6.2.1). If yes, the software problem resolution process is started to handle the problem report (clause 6.2.2).

4.2.3.4 Software Risk management Process

The EN ISO 14971 defines the risk of a hazard as the combination of the probability of its occurrence and the severity of the harm in case of occurrence. This definition is simplified by EN 62304 by defining the probability of occurrence for software hazards as 100%. As a consequence, the only remaining degree of freedom for software-related risks is the severity of harm, resulting in the grouping into the safety classes A, B and C.

During risk analysis, the risks for every software item have to be identified, explicitly including third-party libraries (clauses 7.1.1 and 7.1.2). Similar to EN ISO 14971, risk control measures can be implemented and have to be verified for efficacy. A software item that is used to control a risk always inherits the safety class of the risk it is controlling (clause 7.2.2). Software items of a hierarchically composed software system can have different safety classes if and only if it can be shown that the items with different safety classes are segregated (clause 4.3 d). Two software items can be seen as segregated if they do not share resources like memory and processor units (clause 5.3.5). Although segregation might be expensive to achieve, it can be worth the effort: a single software item with safety class C makes all other software items of the whole software system also class C. The EN 62304 requires more activities to be executed for class C than for B and A, resulting in much higher development effort. If segregation of this software item can be achieved, then the other software items can have a lower safety class and can thus be developed with less effort.

4.2.3.5 Software Configuration Management Process

Medical software interacts with other software (e.g., the operating system or third party libraries) that were not developed according to EN 62304 and thus do not necessarily comply with all process and documentation requirements demanded by this standard. This kind of software is called 'software of unknown provenance' (clause 3.23), abbreviated as SOUP . Software tools such as an inte-

grated development environment or a compiler used for the development and testing of the medical software itself are also considered SOUPs. All SOUPs and the hardware platform(s) on which the medical software is intended to run are considered configuration items. The management of all configuration items is addressed by the configuration management process. Its goal is to ensure that any software item can be recreated and its constituent parts identified, and to provide a history of the changes applied to each software item (annex B.8). As a consequence, each configuration item needs to be identified uniquely (clause 8.1.1). All valid software system configurations (i.e., sets of configuration items) have to be documented (clause 8.1.3). Changes to configuration items (e.g., upgrade of a third party library) are subject to change control: prior to the implementation of a change, an approval of the intended change is needed and afterwards the change has to be verified and any verification tests have to be repeated that might be invalidated by the change (clauses 8.2.2 and 8.2.3).

5 Scientific Objectives

5.1 Own Approach

Given the current state of the art for chronic, bi-directional neural implants for BCIs applications, it can be said that hardware technologies are still evolving and it can currently not be predicted which will prevail. Similarly, the implementation details of medical indications for such implants (i.e., the neurophysiologic signals used, signal processing algorithms and patient feedback) are not yet fully defined. The software used in this phase of research and development has to cope with these unknowns and the constant change regarding the experimental paradigm. In addition, software used in clinical studies on human patients has to be developed under the regime of medical software development. To the best of the author's knowledge, there is no BCI platform that offers a flexible software architecture as well as low-latency multi-threaded signal processing required for computationally demanding research studies *and* is developed as a medical software compliant with the regulatory requirements of the AIMD.

5.2 Roadmap

The goal of a medical BCI platform, including hardware and software, is to provide scientists with the means to evaluate the safety and efficacy of their BCI approach in clinical studies on human patients, hopefully yielding new certified medical indications. Developing such a platform is a substantial effort consisting of many steps over a long period of time. Therefore it is beyond the scope of this thesis to cover all steps towards certification. Here, an outline of the roadmap towards this goal shall be given including the steps addressed in this thesis (Figure 9). These steps can roughly be divided into three phases: research phase, preclinical phase and clinical phase.

In the research phase, the Freiburg BCI Software was developed for evaluation in research BCI studies. Based on these first experiences in closed-loop software, a refined software architecture was derived, suitable for a BCI platform that is modular, flexible and supports multi-threaded processing of neuronal data. A test strategy was developed by selecting a set of software tools and making the software architecture suitable for future verification necessary for medical software development. Meanwhile the development of the implant which would later be termed 'Braincon implant' started and an interface to the Freiburg BCI Software was implemented. The Freiburg BCI Software's signal processing capabilities were optimized in a series of benchmark tests for low-latency multi-threaded processing. Research BCI studies were conducted to validate its closed-loop capability.

In the preclinical phase, the optimized software architecture of the Freiburg BCI Software was taken to 're-developed' a new medical software termed 'Braincon Platform Software' according to AIMD-compliant software development and risk management processes. As a consequence, the two software systems are very similar regarding software architecture and performance. The following text has to be read with this evolution from Freiburg BCI Software to Braincon Platform Software in mind: some results will refer to the Freiburg BCI Software, but are also applicable to the Braincon Platform Software as well due to the almost identical software architecture of the software systems. Extensive testing was conducted to verify that the Braincon Platform Software meets its specifications. Meanwhile, the implant's measurement and stimulation capabilities were evaluated in a bovine animal study. These processes are not yet completed. Although the core functions of the Braincon Platform software are implemented and tested, application-specific algorithms still need to be added for each medical indication. More animal studies will be necessary to evaluate the safety and efficacy of the

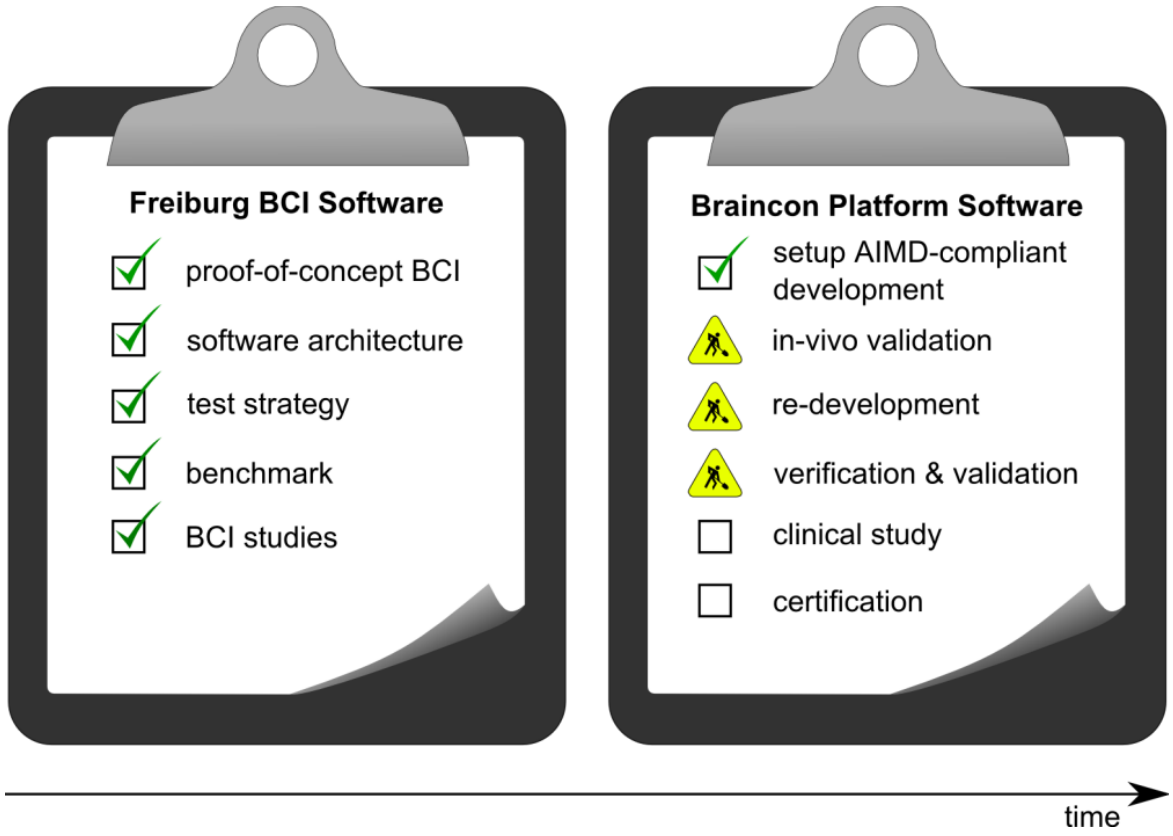


Figure 9: Roadmap

List of tasks with their current state of completion towards certification of Braincon as a medical device. A task can be either done (checked checkbox), in progress (work-in-progress symbol) or not yet done (unchecked checkbox).

whole system (i.e., implant and Braincon Platform Software). The final clinical phase will evaluate the system's safety and efficacy for one specific medical indication in human patients. If successful, Braincon will be certified as a medical device. Thereafter there is the possibility that Braincon will, due to its platform character, foster further closed-loop BCI studies and certifications for other medical indications.

5.3 Software Objectives

Programmers write software systems to help users to solve a problem. Depending on the user and problem, different requirements are made to a software system. The better these requirements are understood and incorporated in design and implementation of a software system, the better the software system helps to solve a problem and the more the user is satisfied. The main requirements that influenced design decisions as outlined in the following section are described here. A tabular overview of these requirements is given in Table 1.

Requirement	Freiburg BCI Software	Braincon Platform Software	Implementation of Requirement
Platform			
Support wide range of medical & research applications	++	++	– Modularity (page 35) – Domain model (page 43)
Conformity to essential requirements of AIMD	+	++	– Software development according to standards – Unit tests& regression tests (page 35) – 'All-or-nothing' rule (page 39) – Mock objects (page 36)
Fast development for new clinical studies	+	++	– Modularity (page 35) – Domain model (page 43)
Performance			
High computational power for yet unknown demanding algorithms	++	++	– Multi-threaded filter-pipeline (pages 43ff, 51ff and 64ff)
Low latency	++	++	– Optimized waiting strategies (pages 51ff and 64ff) – Small block sizes from implant (page 59)
Flexibility			
Fast addition of new functionality	++	+	– Modularity (page 35) – Domain model (page 43)
Fast modification of existing functionality	++	+	– Modularity (page 35) – Domain model (page 43)
Stability, correctness & verifiability			
Verification of classes	++	++	– Unit tests & regression tests (page 35)
Verification during integration	+	++	– Unit tests & regression tests (page 35)
Input parameter checks	+	++	– 'All-or-nothing' rule (page 39)
Isolated tests	+	++	– Mock objects (page 36)
Understandability			
User: Researchers	++	+	– 'No data sharing' rule (page 40) – Simple coding convention (page 42)
User: Software developers	+	++	– Advanced coding convention (page 42)

Table 1: Requirements for the Freiburg BCI Software and the Braincon Platform Software

In the first column, requirements are grouped into requirements for platform, performance, stability, and usability aspects. Second column quantifies the importance of each requirement for the Freiburg BCI Software, third column for the Braincon Platform Software respectively. A '+' stands for nice-to-have features, '++' for features that are mandatory. Fourth column provides references to what was done to address each requirement.

Platform: For a BCI platform, software is needed that can handle a wide range of research and clinical studies. This generality has to be reflected by the software's domain model, *which describes a domain of interest in terms of function and data (Fowler, 2003). Leuthardt et al. (2009)* define four essential components necessary for BCI platforms: a component for signal acquisition, one for signal processing, one for providing feedback for the BCI user and a component that controls the operations of all other components. This domain model is used by BCI2000 (Schalk *et al.*, 2004), a general-purpose BCI software which was used in a wide range of research studies (Schalk, 2009). OpenVibe

(Renard *et al.*, 2010), also a widely used research BCI system, distinguishes six processing steps (acquisition, preprocessing, feature extraction, decoding, translation into commands and finally user feedback, see Renard *et al.* (2010) for details). The Berlin BCI Software only distinguishes 3 steps (Krepki *et al.*, 2003). One goal for the Braincon Platform Software is to provide a domain model that is at least as general as the domain models of established BCI software, because then it can be assumed that such a domain model will be applicable to a wide range of medical and research BCI applications. Existing general-purpose BCI software systems like BCI2000, OpenVibe or BBCI were not developed according to these standards and therefore cannot be used in a medical scenario. In order to comply with the essential requirements of the AIMD, the Braincon Platform Software needs to be developed according to the applicable medical standards (see Section 4.2). For a clinical study with human patients, development according to the applicable medical standards can be very time-consuming, mainly due to the mandatory testing and documentation activities. This can make a clinical study unfeasible. Therefore the Braincon Platform Software should provide commonly used and tested functionality, including utility functions such as file or thread handling, signal visualization, common filter algorithms, ECoG recording and electrical stimulation with the Braincon implant. For a new clinical study, one only needs to implement, document and test additional signal processing algorithms and graphical user interface(s). Existing development documentation of software components can be reused: risk and usability analysis already cover the general use cases of implantation, explantation, ECoG measurement and electrical stimulation. Then it only has to be considered if the documentation of these general use cases have to be extended for each clinical study.

Performance: The Freiburg BCI Software and Braincon Platform Software both should be able to accommodate future, yet unknown and possibly computationally demanding algorithms. In this domain, latency determines how fast a system can react to user (brain signal) input. The lower the latency, the more responsive the reaction of the system is, which potentially increases user convenience and is important for motor BCIs (Ryu and Shenoy, 2009). The more computation time a system can provide, the more decoding steps can be executed per second. More decoding steps mean that the BCI user will receive faster feedback, which can improve the performance (Cunningham *et al.*, 2011). If a BCI software uses less CPU load, one can employ computationally more demanding decoding algorithms.

Flexibility: The number of patients available for an invasive study is relatively low compared to non-invasive studies; it is often not clear when a suitable patient is available, and experiments have to be tailored to the peculiarities of a single patient, sometimes literally over night. These adaptations could incorporate implementation of new or modification of existing algorithms and graphical paradigms. Therefore the ability to modify and extend the functionality is crucial for invasive BCI research. For the Braincon Platform Software as medical software, functional requirements do not change so fast, but as a platform it still benefits from fast addition and modification of functionality.

Stability, correctness and verifiability: BCI software often does complex calculations to a continuous stream of data which can make the detection and fixing of programming errors time-consuming and often difficult. Obviously BCI software that crashes due to a programming error during an experiment or, in the worst case, harms a patient, is not desirable. Therefore a crucial requirement for BCI software is stability. In this context, a stable software system is as a software system running without programming errors and that does terminate in a controlled manner if a runtime error beyond the control of the software occurs. It is also crucial that BCI software is correct software (i.e., working

according to its specification). The standard EN 62304 states that the correctness of a software system can be ascertained through verification (EN 62304 clause 3.33). The verification of the correctness of a medical software system is explicitly required by the EN ISO 62304 (Sections 5.6 and 5.7). In addition, all software units[5] with safety class B or C have to be verified (EN 62304, Section 5.5). As a consequence, all parts and the software in its entirety should be verifiable. For the Braincon Platform Software written in C++, this requires verifiability on the level of single classes as well as on the level of multiple classes together in integration tests (Royer, 1993). Besides the regulatory and normative obligation, thorough testing of a medical BCI software is also mandatory from the ethical perspective: scientists owe it to the users and patients to test their software properly. The properties of stability, correctness and verifiability are closely related. The correctness of a software system can also influence its stability: if any part of a software does not work correctly, this might induce a crash. Verifiable software is a necessary prerequisite for confirming the properties of stability and correctness of the software with a sufficiently high probability. The effectiveness of verification can be increased by verifying classes in isolation. If class A is being verified and uses another class B, it is possible that a programming error in B can make the verification to be shown as correct erroneously. Therefore the requirement was added that classes should be verifiable in isolation. Other sources of errors are parameters provided by user input (e.g., a user tries to open a non-existing file) or from parameter files (e.g., filter coefficients for an unstable infinite impulse response filter, Oppenheim *et al.* (1999)). Therefore facilities for checking input parameters are required.

Understandability: Here, understandability encompasses the ease of use and the comprehensibility of the source code. Obviously, any proposed BCI software platform is useless if the researchers or software developers cannot implement their own experiment or medical indication with it. Therefore the programming skills of the intended users had to be taken into account. The intended users for the Freiburg BCI Software are researchers, mainly *Ph. D.* students, with varying degrees of programming skills. In contrast, the Braincon Platform Software will also be used by professional software developers. For both software systems there have to be means to enable its users to understand and work with the software's source code.

In the following section, it will be described how the aforementioned requirements were addressed.

[5] For the Braincon Platform Software, a single C++ class was defined as software unit.

6 Methods

6.1 Design Principles of the Software Architecture

6.1.1 Modularity

The Freiburg BCI Software and the Braincon Platform Software were written in C++ (Stroustrup, 2000) using the object-oriented programming paradigm. The additional overhead that comes with the use of objects, polymorphism and interfaces was accepted because the benefits of object-oriented programming outweighed the computational costs. One benefit was that now a domain model could be formulated, enabling the exchange of different implementations of the same interface and thereby contributing to flexibility: then functionality could be exchanged or added by changing the implementations of interfaces or by creating a new implementation. For a detailed description of the domain model see Section 6.2.1. Another benefit of the object-oriented programming paradigm is that objects can be reused, in the BCI domain this was used mainly to create multiple instances of the same class (e.g., to create several infinite impulse response filters with different parameters). The division of the whole systems functionality into parts (e.g., a class or multiple related classes) simplifies verification because each part can be verified separately.

6.1.2 Verifiability

Here, the test tools used for the Braincon Platform Software are described. Based on these tools, a test strategy for verification of specifications and detection of programming errors is derived.

6.1.2.1 Test Tools

6.1.2.1.1 Unit Tests

Unit tests are usually written with the help of a unit test framework, a library that provides the means to implementing and execute tests (Hamill, 2004). For the Braincon Platform Software, the Google C++ testing framework (googletest community, 2014) was used to define a test class consisting of a set of test cases for each class. Each test case executed method calls to the tested class and checked the returned result(s). Unit tests require no modification of the tested class and allow splitting test code and productive code in two separate sets. This avoids any influence of test code on the productive code. If testing is done before integration for each class, errors contained within each class can be detected earlier. Test cases can be seen as examples for the usage of the tested class, therefore test cases extend the documentation of classes by providing examples for their usage. For medical software, the EN 62304 demands verification of single software units (in this case classes) for safety class B and C to fulfill the requirements defined by software architecture and detailed design (EN 62304 clauses 5.5-5.7). Many such requirements can be verified by unit tests that therefore provide a method for verification in the sense of EN 62304.

6.1.2.1.2 Automatic Regression Tests

Regression tests are tests that check whether a change of a software system affected functionality, reliability, performance or cause additional defects (EN 62304 clause 3.15). While unit tests are created during initial implementation of a software, they can also be used as regression tests when the software is modified. Regression tests are required by EN 62304 for all parts of a software affected by a change. It can be difficult and time-consuming to determine the affected parts. However, one can simply execute all unit tests after a change and thus the determination of the affected parts can be omitted. The proposed test strategy endorses automated regression test execution with a contin-

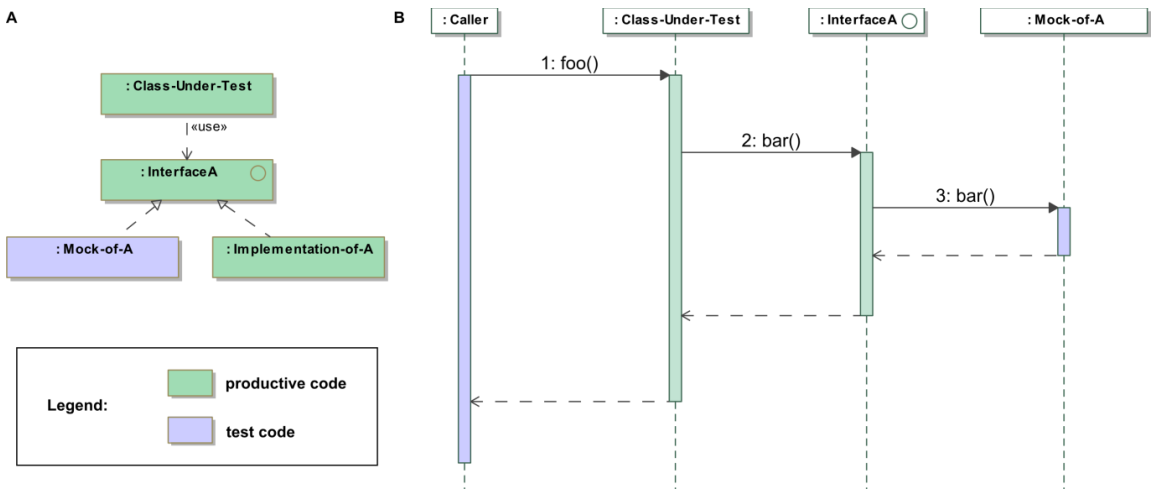


Figure 10: Test pattern for isolation of tested class

Classes belonging to productive code are shown in green, classes used solely for testing in lilac. A: UML class diagram shows the relationship between the tested class and the classes it interacts with. B: UML sequence diagram shows calling sequence during a test-case. Actual call to method bar() is re-directed to a mock implementation of InterfaceA.

uous integration (CI) server (Duvall *et al.*, 2007; Zaytsev and Morrison, 2012). Such a CI server automatically compiles, links and then runs regression tests every time the source code changes. If there are any failed tests, developers are informed by email. Although the usage of a CI server is not mandatory in this test strategy, it reduces development effort by automating the recurrent task of regression test execution.

6.1.2.1.3 Mock Objects

During program execution classes use other classes (i.e., methods call other classes' methods). An error in the called method could influence the result of the calling class. This effect also influences all unit tests where the tested class calls methods from other classes. If a test case fails, it is not *a priori* clear if the error is located in the tested class or in the methods this class calls during testing. In the worst case, a test case that should fail actually succeeds due to an error in a called method. Testing is also more complicated if the class interacts with external resources (e.g., a database accessible via network connection or a robotic arm). In a straight-forward test case implementation, one could provide actual instances of all necessary external resources (e.g., setup a database and a robotic arm). In order to test the behavior of the tested class in the presence of runtime errors, one would have to generate runtime errors during testing. This can be very time-consuming and complex (e.g., generate a network disconnection or a defect in a joint actuator of a robotic arm). In addition, for each test case execution one would have to reset these external resources into a well-known state.

The proposed test strategy requires that each class should be tested in isolation which in turn has consequences for the design of classes and the interfaces they use. In order to isolate one class, in each test case one needs to replace all classes used by the class under test with so-called mock classes that behave according to the test case. For example when a test case tests a query to a database, the mock class simulating the database will return a set of data. In another test case, the query to the database could throw an exception to simulate a runtime error. In the C++ language, one can facilitate this replacement with templates or with polymorphism (Stroustrup, 2000). When using templates, a template class would maintain all classes that need to be replaced by mock objects as tem-

```
// Create an instance of the mock class
CMockDatabase* db = new CMockDatabase();

// Set the expectations of the mock class
{
      InSequence s;
      EXPECT_CALL(db, connect()).Times(1).willOnce(Return(true));
      EXPECT_CALL(db, query("SELECT * FROM COUNTRY")).Times(1)
      .willOnce(Throw(runtime_error("Network error"));
}

// run the test
CDatabaseFacade facade(db);
EXPECT_THROW({facade.getCountries();}, connection_error);

// expectations are checked when db is being deleted.
```

Code example 1: Mock object usage example

Example of a test case using a mock object from Google mocking framework (googlemock team, 2014).

plate parameter. With polymorphism, all classes interact only through virtual methods to facilitate the replacement. Here, replacement would be implemented by overriding these virtual methods. The test-pattern used to test classes in isolation is shown in Figure 10: The tested class uses a pointer to an instance of type 'InterfaceA'. During testing, this instance is a mock object, while in productive usage a separately tested implementation of 'InterfaceA' is used.

One could achieve isolation during testing by implementing mock classes for each test case, which can lead to large test code. In addition, such mock classes would have to be verified too. There are libraries called mocking frameworks (Freeman and Pryce, 2011) for reducing the programming effort when implementing mock classes. In order to perform a test case with the help of a mocking framework, the following steps are executed:

1. **Derive a mock class from the class that should be mocked.** This mock class can be reused when mocking the same class in a different test case, so this step is necessary only once for each interface.
2. **Create an object of the mock class.**
3. **Set the expectations of the mock object.** Expectations are declarative statements that define the behavior and expected usage of the mock object. This could be for example how often each method of the mock object should be called, what arguments the methods should expect at each call and what to return.
4. **Run the test.** At the end of the test the mock object verifies if the expectations have been fulfilled. If not, the test case fails.

In contrast to implementing the behavior of a mock object, declaring the behavior has two main advantages: repetitive checks like ‘method is called exactly two times’ need to be implemented only once and thus need to be verified only once, and declarative statements are easier to comprehend, because the expectations roughly resemble English sentences. Code example 1 gives a concrete example of mock object usage in a test. In this test, the method getCountries() of class

CDatabaseFacade should throw an connection_error exception when the database object signals a network error. After mock object ‘db’ is created, two expectations are set: firstly, ‘db’ expects its connect() method to be called. The call to connect() should succeed, therefore ‘db’ will return true. Secondly, a call to query method expects as argument the string “SELECT *FROM COUNTRY” and will throw an exception to indicate that during this query a network error has occurred. Please note that with the statement “InSequence s” the call to connect is required to be executed before the call to query. After setting up the expectations, the test is executed by calling façade.getCountries() and will succeed if the expectations of db are fulfilled and getCountries() throws a connection_error exception.

For the Braincon Platform Software, the Google mocking framework (googlemock community, 2014) is used, because it is actively maintained, is compatible to the Google testing framework (googletest community, 2014) and provides a large library of functions that can be used for setting expectations.

6.1.2.2 Test Strategy

In the object-oriented programming paradigm, software is composed of one or more objects that interact with each other through method[6] calls (Freeman and Pryce, 2011), based on the parameters provided to software at start. Each method call is composed of the following steps: First, the method call is initiated by the calling object. If the method requires arguments, they are provided by the calling object. Then the method is executed and, if the method returns a result, the result is returned to the calling object. If an error occurs during execution, the error is signaled to the calling object. Hence, four types of errors could be identified during a method call:

1. **Calling object violates the pre-condition of the called method.** Each method might require constraints on the input arguments (e.g., only non-empty strings or only numbers > 0) but also on the state of the object whose methods is being called (e.g., stop()-method may only be called once after a call to start() method).
2. **Called method violates its invariant.** During each method execution, the method’s invariant has to be true. For example in a method that counts objects within a loop, an invariant might be that the current sum of objects must be greater or equal to 0. Such a violation could be caused by a programming error within a method.
3. **Runtime error is not correctly handled during method execution.** If a runtime error occurs, a method has to handle this error appropriately. Consider as an example a method that calls another method for inversion of a matrix. If this matrix inversion fails, the calling method has to detect this failure and act accordingly (e.g., by stopping the software system). If this failure could not be detected, the method would continue execution with an invalid matrix and would most likely return incorrect results.
4. **Invalid input at application start.** Similar to runtime errors, users may provide inputs that the starting application cannot handle or can lead to an error during method execution if they are not detected. Here, inputs encompass command line parameters as well as parameters from a configuration file or database. Examples would be unstable filter parameters or a path to a file that does not exist.

In the following, for each error type a programming strategy is provided to detect it.

[6] The same statement holds for constructors, operators and global static functions because they can also be called like methods and are therefore as special cases of methods seen in the context of testing.

6.1.2.2.1 Violation of Pre-Conditions

Pre-conditions impose restrictions on a method's arguments. If the pre-condition of a method was required to be checked by the calling code every time, then every time there would be a chance that the programmer forgets to add the check. In addition, if the pre-condition changed, then these checks would have to be modified everywhere the method is called. Again, such a modification can be forgotten. The chance to forget would increase linearly with the number of method calls. Therefore this approach would lead to a potentially error-prone source code which is more laborious to maintain, contradicting the requirements of stability and flexibility. Instead, the test strategy proposed here requires each method to check its own pre-condition. This way, the check of pre-condition is implemented once for each method and the programmer is less likely to forget to add a check. Likewise, modifications to pre-conditions can be done in only one place (i.e., the method). Unit tests can be used to verify that the method works for arguments that satisfy the pre-condition as specified and that for arguments that violate the pre-condition an error is signaled by the method in the specified manner (e.g., error code or exception).

6.1.2.2.2 Violation of Invariants

If a method's pre-condition is satisfied, a method's invariant can only be violated by a programming error. The proposed test strategy provides two methods for detection of invariant violations. (1) Unit tests: A programming error is likely to make a method return the wrong results. Therefore unit tests can be used to check a method's results. (2) Assertions: When using assertion, one formulates Boolean conditions in the method's body and evaluates these conditions with 'assert' statements from the standard C library (ISO IEC 9899:1999, International Organization for Standards). Every time an invariant is violated (i.e., the assertion fails) program execution is stopped. One property of assert statements in C++ is that they are evaluated only if the code is compiled in 'debug'. This makes the execution of code compiled in 'release' mode faster than in 'debug' mode and the programmer does not need to worry that these checks degrade the application's performance. The use of assertions promises to yield more robust applications as assertions alert the programmer immediately at runtime when and where within the method body something is awry.

For verification, it is actually sufficient to use only unit tests to check a method's results. However, assertions can give more detailed information to the programmer about where the invariant violation occurs, as they can be employed within the method body. This might accelerate fixing of programming errors, especially if the method body contains complex code. Therefore, the proposed test strategy always requires unit tests to check method results (i.e., to verify each method's conformity with its specification) and endorses the additional use of assertion in complex methods.

6.1.2.2.3 Unhandled Runtime Errors

Each method needs to detect and handle runtime errors according to its specification which might involve the returning of an error code or to throwing of an exception. Consider the general case where a method A is called by method B. In a unit test, the proposed test strategy employs mock objects for letting A deliberately fail (e.g., throw an exception) and then verifying if the runtime error is handled according to the specification of B.

6.1.2.2.4 Invalid Input at Application Start

The more complex a software system, the more likely an error caused by wrong configuration or user-provided input parameters can occur (e.g., a parameter is in the wrong range). When such an

error is detected by the application, there are several options how to continue: ignore this error, try to fix it and continue or abort program execution. In the context of medical software, ignoring is not an option. Although in some cases it might be possible to fix the error (e.g., by setting this parameter to a default value) this might deteriorate the reliability of the whole application. Consider a simple recording of neuronal signals done with some parameters (e.g., a set of channels to record and a band pass filter for each channel). The user cannot know if the signals on screen are actually recorded with the chosen parameters or some fixed parameters. Therefore, the proposed test strategy requires to use the last option and stipulates the rule that all input parameters have to be checked programmatically by the reading object at application startup. If an error is detected, application execution has to be aborted. This rule was termed 'all-or-nothing rule', because it guarantees the user that with a successful application start *all* parameter checks succeeded, otherwise *nothing* starts and the user is forced to correct the parameters.

6.1.3 No Data-Sharing

The Freiburg BCI Software and Braincon Platform Software were intended to be modular BCIs. Therefore their software architecture consists of multiple modules[7] that exchange objects (e.g., measurements) from an amplifier module are sent to other modules like processing filters. Each module might process its objects in a separate thread. For any object that is used by more than one module the question arises whether these modules should work on the same object or use an own copy of this object. Sharing objects requires objects to be thread-safe, which is often difficult to implement. Copying objects increases memory consumption but obviously needs no synchronization, as each module maintains its own copy. In the BCI domain, two typical cases were identified. In both cases it was decided to not share data between threads. The reasons behind these decisions are provided in the following.

6.1.3.1 Big Permanent Objects

A BCI system can contain buffers (e.g., buffers to accumulate measurement data from multiple time points for a processing with a sliding window like short-time Fourier transform (Allen, 1977)). Besides buffers, the following arguments apply to all big permanent data structures to which multiple modules might want to have shared access.

As buffers can be memory consuming, one might apply the singleton pattern (Gamma *et al.*, 1994; Alexandrescu, 2001) to reduce memory consumption by sharing one data structure between all modules that need access to it. Although this minimizes memory consumption, the implementation yields are two drawbacks.

Object lifecycle operations like initialization, reset and cleanup of a singleton object have to be managed. These life cycle operations either have to be provided by the modules that use a singleton or by the global application that manages the modules. In the first case it is not clear which of the modules is responsible for its initialization and cleanup, because no module is guaranteed to be used in all configurations, therefore each module has to provide these methods. This leads to redundant code which is error-prone and difficult to maintain (e.g., if a newly written module initializes differently or not at all). In the second case, while a global life-cycle method has no redundant code, the global application needs to know if there are any modules that use a singleton object at all. So either a sin-

[7] In Section 6.2 Software Architecture the general term 'module' will be divided more precisely into 'processing object', 'active filter' and 'passive filter'.

gleton object is always created, even if it is not used at all, or there has to be a query mechanism between global application and modules so that the need for a singleton object can be queried. This leads to a cyclic dependency between application and module, which contradicts to the modularity principle (see Section 6.1.1).

Access to singleton objects has to be synchronized between threads. This synchronization costs additional computation time (e.g., when one module has to wait until another object has finished its access to a singleton object). In the worst case this creates performance bottlenecks. Additionally, creation of synchronized data structures can be difficult and is error-prone. Programming errors related to synchronization, especially livelocks and deadlocks (Roscoe, 1998; Schneider, 1998; Unger, 1995), are often non-deterministic and difficult to fix. In the worst case, such bugs do not occur on the development system, but only non-deterministically on the productive system.

If big permanent objects are not shared, the responsibility for their life cycles clearly lies with the using module and synchronization is not required. From the experience with the Freiburg BCI Software and given the current state of memory technology, it is feasible to accept the additional memory overhead through redundant objects, as the following calculation shows: It is assumed that 8 bytes are used to store each value because this is sufficient for high precision data types (e.g., double precision floating point numbers (IEEE 754:2008, IEEE Standards Association) or 64-bit integer numbers). The following values were chosen from Fischer *et al.* (2014) as they represent typical parameters in the BCI domain and were then doubled to get an upper bound estimate. It was further assumed that 2s of data is buffered from a data stream with 2000 channels and a sampling frequency of 2000Hz. This yields a buffer size of 61MB. In a computer system with 4GB of memory, excluding 2GB for the operating system and other data, there would still be enough memory left for 33 buffers. However, the previous calculation only shows that it is very likely that all big and permanent objects needed for BCI operations fit in memory without sharing. There can always be exceptions that have to be dealt with separately.

Given these arguments, the disadvantage of additional memory overhead is outweighed by the benefits of more simple non-synchronized objects and without need for global life-cycle methods and therefore, each module should maintain its own big permanent objects.

6.1.3.2 Small Short-Lived Objects

In a closed-loop BCI system, a stream of measurement data is continuously processed. This means that data has to be passed among modules (e.g., from amplifier through feature extraction to decoding modules). Consider the case where an amplifier distributes its measurement data to multiple other modules. The data can either be copied for each module or be shared amongst them.

The copy strategy is simple to implement, as it requires from each object only to be able to make a complete copy of itself. No synchronization between threads is necessary, as each thread has its own copy of the data. But it would always require a call to a memory allocation mechanism at the data objects creation, in C++ usually the ‘new’ operator, and a call to a memory deallocation mechanism, in C++ usually the ‘delete’ operator, when the object is not needed anymore. In addition, copying the content from the original data object to its copy also costs time and CPU load. As the object can pass thread boundaries when it is moved between modules, no thread-local storage (ISO IEC 14882:2011, International Organization for Standards; Microsoft Developer Network Library, 2014) can be used. Therefore the global heap has to be used and locked for allocation and deallocation, creating a po-

tential bottleneck, as there can be experimental settings with many allocations and deallocations per time unit (e.g., consider an experiment with an amplifier that has a sampling rate of 10 kHz). In addition, continuous allocation and deallocation of object with different sizes can lead to heap fragmentation (Wilson *et al.*, 1995) which degrades system performance or, in the worst case, the application runs out of memory. One could use preallocated pools of data objects (e.g., segregated free lists (Wilson *et al.*, 1995)), which makes repeated (de)allocations from global heap unnecessary, thus avoids memory fragmentation and promises faster object creation and deletion.

Sharing small, short-lived objects could be achieved by using smart pointers (Alexandrescu, 2001) with reference counting (e.g., as provided by the boost library (Karlsson, 2006; boost community)). From the perspective of a module usage of such a pointer would be identical to regular pointers: the pointer would be deleted when the object is not needed anymore, thus shared pointers would not complicate pointer management. But internally, data objects and shared pointers need to be thread-safe, as in the BCI scenario, multiple modules could access the same data object simultaneously, adding more complexity to data objects. Now consider the case where two modules want to modify the same data object in different ways (e.g., a module that applies common average referencing (Ludwig *et al.*, 2009) to measurement data and a module that normalizes measurement data into the interval [-1, 1]). Even with proper synchronization, one of these modules will not produce the correct output as the data object can only hold the result from one module. This example can be generalized to the requirement that shared data objects have to be copied prior to modification. This forces the programmer of a module to create a copy of a shared object each time before a modification to its data is applied. If a module erroneously forgets to create a copy, these modifications can induce errors in other modules using the same shared data object. This kind of errors is difficult to reproduce, as they are time dependent, and their cause is difficult to locate, as it is inside another module.

At the time when the software architecture for the Freiburg BCI Software was developed, it was not clear how big the performance loss due continuous allocations/deallocations from the copy strategy would be. As the software architecture of the Freiburg BCI Software was also a test environment for the Braincon Platform Software, it was decided to use the copy strategy and evaluate its performance. It turned out that the costs are sufficiently small, see results in Section 7.1. In addition, the simplicity of the copy strategy promised a more robust system in contrast to shared pointers. For all these reasons, the copy strategy was also used for small short-lived objects in the Braincon Platform Software.

6.1.4 Coding Convention

A coding convention imposes syntactic and semantic rules on source code. Typically, it regulates the naming of files, classes, methods and variables, code structure and documentation, thus making it easier to understand source code written by other programmers. If programmers understand what a method does, then then they are less likely to use this method wrong, yielding a potentially more reliable application. The initial coding convention for the Freiburg BCI software was written with Ph.D. students as programmers in mind. Their programming skills can vary from novice to expert. Therefore it contained mainly rules for better readability of code and rules that helped to avoid typical novice programming errors. Additionally, programming techniques deemed insecure or error prone were forbidden. The coding convention used for the Braincon Platform Software was written for expert C++ programmers with the aim to increase safety and reliability of the source code. Many rules were adopted from the coding standard used in the ‘Joint Strike Fighter’ program, which was

intended to enable programmers 'to employ good programming style and proven programming practices leading to safe, reliable, testable and maintainable code' (Lockheed Martin Corporation, 2005).

6.2 Software Architecture

6.2.1 Domain Model

The domain model of a software describes a domain of interest from the existing world in a set of data structures and function definitions (Fowler, 2003). The BCI domain of interest considered here encompasses a patient interacting with a computer in a closed-loop manner. Therefore there has to be a software representation of the data source (e.g., an implant) that provides a continuous stream of neurophysiologic measurements. If the BCI supports electrical stimulation of brain areas, there have to be facilities for executing stimulation commands in its software representation of the data source. Depending on the experimental setup, there might also be additional neurophysiologic recording systems (e.g., for measuring SUA/MUA activity, ECoG, MEG or muscle activity (Merletti and Parker, 2004)). There can also be recording systems for capturing the patient's movement (e.g., a computer mouse or a joystick). Besides passive movement recording, some movement recording systems also actively execute movements (e.g., an active orthosis (Kawamoto *et al.*, 2010; Banala *et al.*, 2010)) that moves the patient's limb, or provide force feedback (e.g., a SensAble Phantom Omni haptic device (SenseGraphics AB, Kista, Sweden) or a robotic arm). A BCI extracts information from neurophysiologic measurements (e.g., intended movement (LaFleur *et al.*, 2013; Milekovic *et al.*, 2012) or epileptic seizure probability (Morrell, 2011)). First, features are extracted from these measurements by using one or more feature extraction algorithms (e.g., low-pass filtered by a linear filter (Oppenheim *et al.*, 1999) or short time Fourier transform (Allen, 1977)). Then the desired information is decoded from the previously generated features. Decoding can be done by decoding algorithms (e.g., linear discriminant analysis (Hastie *et al.*, 2009), Kalman Filter (Haykin, 2001; Kalman, 1960) or support vector regression (Cristianini and Shawe-Taylor, 2000; Vapnik and Chervonenkis, 1974)) or might involve multiple decoding algorithms working together in a boosting approach (Hastie *et al.*, 2009). Depending on the experimental setup, feedback is provided to the patient (e.g., decoded movement intention can be provided as visual feedback or used to control an external effector or applied directly to the brain as electrical stimulation feedback). There can also be cases where patient and implant are connected in an open-loop configuration (e.g., for recording calibration data prior to training of decoding algorithms or for functional brain mapping).

The goal of the domain model proposed here was to be general enough to handle all these configurations as well as many yet unknown future BCI applications.

6.2.2 Implementation

From a more general perspective, a BCI application can be seen as a stream of data which is being processed. Therefore for the proposed domain model three main types of objects were identified: processing objects (PO), data objects and connection objects. Figure 11 shows the interfaces related to these objects.

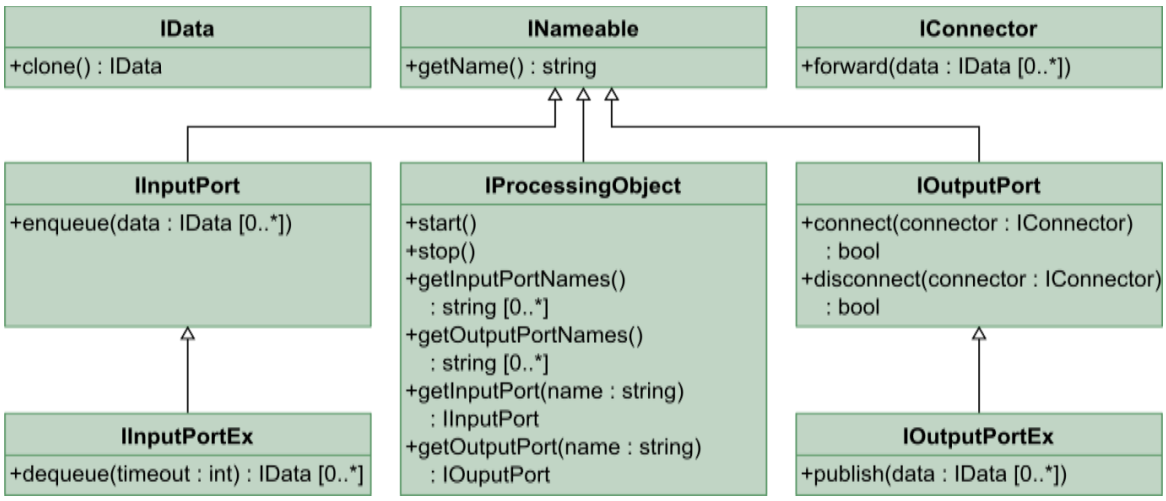


Figure 11: Interfaces related to processing objects

Class diagram of the interfaces related to processing objects in unified markup language notation (UML, Booch et al., 2005). Every IProcessingObject implementation can have zero or more instances of IInputPort or IOutputPort. IConnector is the interface from which connection classes inherit in order to receive data from an output port (cf. observer pattern in Gamma et al., 1994). C++-specific data type modifiers like references, pointers or 'const' omitted for the sake of readability.

Data objects represent the data to be processed by processing objects (e.g., neuronal activity measurements or measurements of limb movements). They are used to move information between other classes. All data classes are derived from the IData[8] interface.

A processing object takes input streams consisting of data objects, processes them and provides output streams. There can be processing objects that do not need input streams (e.g., a processing object that provides a data stream of limb movement measurements from a joystick). There can also be processing objects that do not provide any output streams (e.g., processing objects that convert input streams into visual feedback for the patient). Processing objects can be run in different threads to make efficient use of multi-core processor computers and thus meet the requirement for computational performance as defined in Section 5.3.

A connection class connects two processing objects by forwarding the output data objects from one processing object to the input data stream of another processing object. By different configurations of connection objects, one can realize open-loop or close-loop BCI applications.

According to the design principles, no object instances are shared between threads (Section 6.1.3). This was implemented by requiring all data classes to provide a 'clone' method for creating a deep copy of itself. When passing a data object instance between two threads, actually a pointer to this data object is passed. Either the sending processing object relinquishes ownership of the data object and passes the data object to the receiving processing object or it creates a clone of the data object and actually passes the clone to the receiving object. This pattern can also be extended to the case

[8] In the used class naming convention, interfaces start with an 'I', all other classes start with a 'C'. For both class types we use so-called camel hump notation, where each word of a compound name starts with a capital letter (e.g., CNameWithMultipleWords).

when a data object should be passed to multiple processing objects: one simply creates multiple clones.

Input ports are classes that receive streams of data objects for processing by a processing object. They are derived from the IInputPort interface (Figure 11). One can insert multiple IData instances into an input port by calling its 'enqueue' method. The 'enqueue' method has to be thread-safe so that data objects can be inserted into the input port from arbitrary threads. An IInputPort object guarantees that the data objects are passed on to the processing object in the temporal order in which they were enqueued. The current input port implementation is essentially a thread-safe queue, but the IInputPort interface allows for different implementations like an input port that serializes queued data and sends it to another application running on a different computer system via network socket (e.g., a cluster computer system for complex computations which cannot be handled in time by one computer system).

Processing objects are derived from the IProcessingObject interface (Figure 11). They provide methods for starting and stopping processing. Each processing object can own input and output ports by owning instances of the IInputPortEx interface. IInputPortEx extends the IInputPort interface by a method for dequeueing data objects previously inserted via the 'enqueue' method. By separating input port functionality into two interfaces, a processing object can selectively expose input port functionality to other objects: other objects can enqueue data objects to exposed IInputPort instances, but only the processing object owning an input port can dequeue data objects from it by using the 'dequeue' method from IInputPortEx. The result of a processing object is a set of data objects which is distributed through its output ports. The process of dequeueing, processing and distribution of results is repeated. There are different implementations of this cycle which will be described in detail in Section 6.2.4.

Output ports use the observer pattern (Gamma *et al.*, 1994) to distribute the results of a processing object to other process object input ports. This pattern works similarly to a mailing list: all persons registered to a mailing list receive emails addressed to this mailing list. One does not need to know which persons are registered for a mailing list in order to send a mail to all registered persons. For example, consider a processing object A that wants to receive the data objects from the output port of another processing object B. A connects a connector (i.e., an object derived from the IConnector interface) to B's output port by calling its connect method (Figure 11). In the email list analogy, this would correspond to registering an email address for a mailing list. Each output port maintains a set of connectors which are connected to it. When the processing object has new results, it calls the output port's 'publish' method which notifies all connected listeners by calling their 'forward' method. This step would correspond to writing an email to a mailing list in the email list analogy. The connector forwards the data to another input port where the whole process is repeated.

In Figure 12 a UML sequence diagram shows the method invocation sequence of a processing object with one input port and one output port. Initially, a connector object enqueues data objects. Then the processing object, running in another thread, iterates over three steps: First, it dequeues the data object from the input ports, secondly it processes the data objects and finally it distributes the resulting data objects by invoking the output port's 'publish' method, which in turn notifies all connectors attached to it by invoking their 'forward' method. In the case of processing objects with n

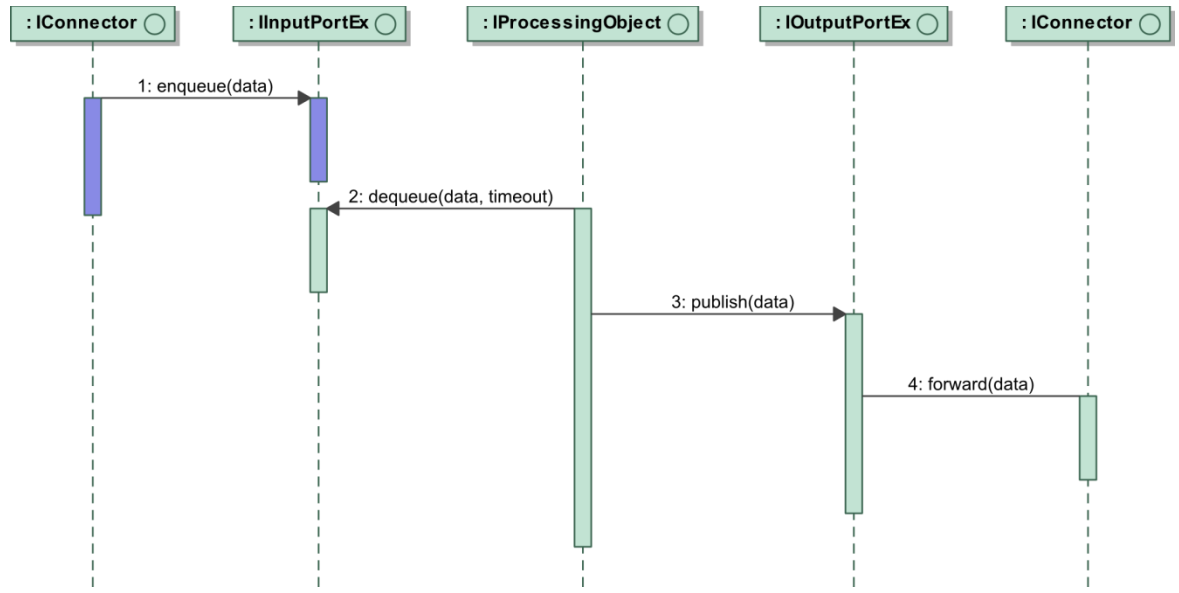


Figure 12: Sequence diagram for passing of data objects

UML sequence diagram that shows passing of data objects to a processing object with one input port and one output port. Processing object runs in its own thread (green life lines) as well as the connector (blue life lines) which enqueues data objects.

input ports and m output ports, method call sequence is similar as calls to 'dequeue' and 'publish' are repeated n respectively m times.

The division into processing objects and connector objects separates sender and receiver in a sense that the sender does not need to know its receiver, similar to the mailing list analogy: the sender does not need to know the subscribers of a mailing list. The sending source code only interacts with instances of the IConnector interface and therefore makes the source code of the sending processing object independent from the source code of the receiving processing object. These IConnector instances can be exchanged. As a consequence, any processing object class can be instantiated (multiple times) independently from other processing objects and can be arbitrarily connected to other processing objects, even at runtime. These properties of the proposed architecture achieve the required flexibility (see Section 5.3).

6.2.3 Startup and Shutdown Order

In the proposed software architecture, a set of processing objects is being maintained for providing the BCI functionality (e.g., reading of neurophysiologic data, feature extraction and decoding and visual or electrical feedback) where each processing object can operate in its own thread. When the user starts the software these threads have to be started. The order in which these processing objects are started or stopped is of importance, as outlined in the following. Consider the following use-case: an amplifier processing object of a neurophysiologic recording system is connected to a utility window view which displays the measured data at runtime. Both processing objects cannot be started simultaneously. If the amplifier is started first, measurements can be queued into the view's input port while the view is not yet started. This creates a transient stall of measurements in the view's input port. In such a case, the view has now two options: ignore the stalled measurements or display all stalled measurements in an initial burst at the same time. Both options are not desirable because in the first option data is ignored and in the second option an unsmooth burst of information is dis-

played at startup. This transient stall can be avoided if the amplifier is started last. In general, processing objects that create data objects like amplifiers or trackers have to be started after the other processing objects.

One could assign a numerical startup priority to each processing object and start the processing objects according to their priority. These priorities have to be assigned manually for each configuration, as each processing object does not know the other processing objects it is interacting with.

Another approach would be to identify processing objects that produce data by deriving them from a special interface termed IDataSource. This interface derives from IProcessingObject and adds no methods; its sole purpose is to flag a processing object as data source. At startup, each processing object of a configuration is tested via runtime type information if it is of type IDataSource. If yes, it is started after all the other processing objects. The advantage of this approach is that startup order is derived automatically for all combinations of processing objects, therefore this approach was chosen for the proposed software architecture.

6.2.4 Filter Pipeline

The process of neural decoding used in BCIs can be seen as a sequence of signal processing steps applied to a stream of neural data. The processing steps vary depending on the signal type (EEG, electrocorticogram, spiking activity of individual neurons, MEG, etc.) and the BCI application. Here, a single processing step is called a 'filter' and the whole sequence of processing steps is called a 'filter pipeline'. A filter is therefore a transformation (linear or non-linear) of an n-dimensional input signal to an m-dimensional output signal. With this terminology, low-pass, band-pass or high-pass filters are included in the definition of 'filters', as well as the short-time Fourier transform (Allen, 1977) or classification and regression methods, such as linear discriminant analysis (Hastie et al., 2009), support vector regression (Cristianini and Shawe-Taylor, 2000; Vapnik and Chervonenkis, 1974) and Kalman filter (Haykin, 2001; Kalman, 1960). It also includes state-machine-like functionality used for modeling the state of a visual feedback paradigm (e.g., cursor and target positions in a center-out task (Waldert *et al.*, 2009)). From the software architecture perspective, each filter is implemented as a processing object and the whole filter pipeline is implemented as a set of connected processing objects.

A filter's worker thread iterates over three steps, as shown in Figure 13*. First, it fetches a new set of input data objects by dequeueing its input port. Then the input data objects are processed according to the signal processing algorithm implemented by the filter, resulting in a set of output data objects. Finally, these data objects are distributed to the connection objects connected to the output ports.* Two types of filters were identified as necessary for implementing BCI filter pipelines: active and passive filters.

6.2.4.1 Active Filters

An active filter's worker thread checks each input port for newly arrived input data objects at regular time intervals. This filter type was termed 'active' because it actively checks for input, in contrast to the 'passive filter' described later which passively waits for input to arrive. If there are any inputs, it dequeues them all from the input port and passes them to the signal processing algorithm. If there are no inputs in a time interval, the processing algorithm is still called. This property also justifies the name 'active filter' because such a filter can autonomously generate output data objects (i.e., without input data objects).

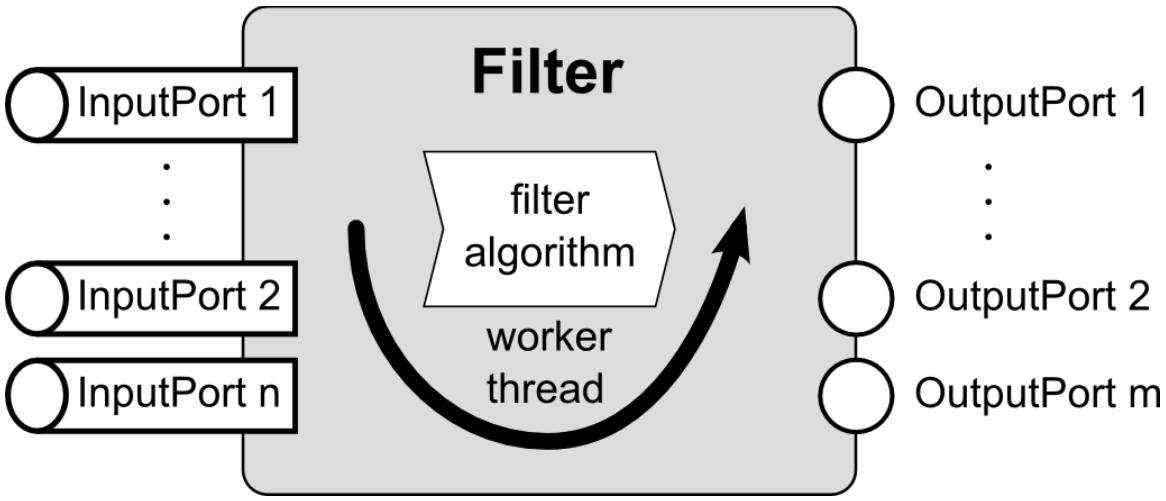


Figure 13: Schematic representation of the filter implementation

Figure depicts the main working steps of a filter. After a filter is started, its worker thread collects the input data objects from all input ports. Then an exchangeable algorithm processes the data objects and its results are distributed to the output ports. This is repeated until the filter is stopped. Figure from (Fischer et al., 2014).

This type of filter can be used to implement state machines that depend on time. As an example consider a filter that implements a visual feedback paradigm for a 'center-out' task where a cursor should be moved for five seconds after a go-cue from the center of the screen to a target at the rim of the screen. Every five seconds it creates a new output data object that signals alternating go or pause cues. This filter also receives a data object containing decoded movement every second. In order to make cursor movement appear smoother, it interpolates the decoded movement trajectory. For every decoded movement data object, the active filter creates ten interpolated movement data objects. Every 0.1 seconds a data object with interpolated movement is sent to an output port of the active filter object.

6.2.4.2 Passive Filters

Passive filters can be used to implement mathematical functions like short-time Fourier transform, linear filters or support vector regression. They need exactly one data object from each input port in order to create a set of output data objects. If there is no data object available in one input port, the passive filter's worker thread has to wait until at least one data object has arrived. The way a worker thread waits is defined by its waiting strategy. This waiting strategy influences the performance of a passive filter with regard to latency and CPU load as shown in Section 7.1.

An operating system usually provides facilities for threads to wait until certain events happen (Hart, 2010; Corbet and Rubini, 2005; Halvorsen and Clarke, 2011), therefore a worker thread could wait for the event 'new data has arrived'. This waiting strategy was termed $\text{Wait}_{\text{event}}$. $\text{Wait}_{\text{event}}$ has the advantage that the worker thread only waits as long as necessary, but it could require potentially expensive kernel calls for synchronization.

Another option would be to wait for a fixed amount of time and then check again whether new input has arrived in the meantime. Then one has to choose the amount of waiting time, which is generally not known. Here, three choices of waiting time were considered, resulting in the waiting strategies 'Polling', 'Wait_0' and 'Wait_1'.They were defined in the following.

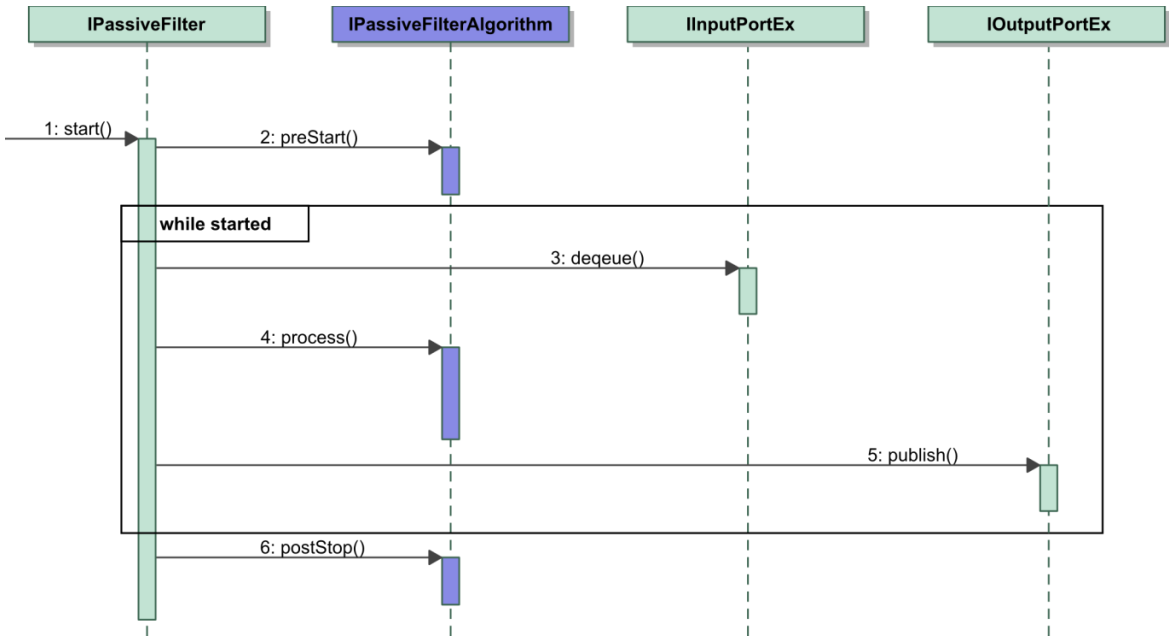


Figure 14: Sequence diagram for passive filter

Figure shows UML sequence diagram for passive filter with one input and one output port. Method preStart() is called once after filter has been started. Likewise, postStop() is called after processing was stopped. Signal processing is performed by the process() method.

The waiting strategy Polling does not wait at all, it continuously polls (i.e., queries) the input port for new data.

In a multi-threading environment governed by a scheduler (Stallings, 2012), a thread gets so-called time slices allocated by the scheduler in which it is executed on a CPU core. If the operating system's scheduler provides facilities for a thread to yield the rest of a time slice (Li *et al.*, 2009) a waiting strategy could stop execution during a time slice and make the remaining time available for execution of other threads. This waiting strategy was termed '$Wait_0$'.

Waiting strategy '$Wait_1$' waits for one tick of the operating system's event timer, which is approximately 16 milliseconds (ms) in Windows.

6.2.4.3 Implementation

The signal processing algorithms which should be performed by filters for BCIs vary. Therefore the strategy pattern (Gamma *et al.*, 1994) was employed to make the algorithm part exchangeable, see Figure 11. Each filter implementation (CActiveFilter and CPassiveFilter) has an algorithm (i.e., an instance of IActiveFilterAlgorithm or IActiveFilterAlgorithm). With this separation, the signal processing algorithm is only responsible for the processing of input data. The responsibility of the filter encompasses management of the data flow (i.e., create a worker thread which repeatedly fetches inputs, passes them on to the algorithm and distributes the results to the output ports until it is stopped (Figure 14)). This separation provides two benefits:

Firstly, one can reuse the two filter implementations because their task of worker thread management and data flow management is independent from the signal processing algorithm. New func-

tionality can be added faster because only a filter algorithm class needs to be implemented. This contributes to satisfying the requirements of fast extension of functionality (see Section 5.3).

Secondly, it simplifies testing because filters and filter algorithms can be tested separately from their filter classes: the reusable classes CActiveFilter and CPassiveFilter with their thread and data flow management are tested once. Filter algorithms need to be tested only for data processing, no threading is involved, thus improving the verifiability of these classes.

6.2.5 Parallelization

The filter pipeline continuously applies a sequence of signal processing steps (i.e., filters) to a stream of neural data. In a typical BCI application, such signal processing steps encompass the acquisition of signals from a neurophysiologic recording system, feature extraction and movement decoding by the filter pipeline and generation of feedback. Here, the case is considered that these computations are executed on a computer system with multiple CPU cores: on each core only one thread of execution can be executed at any point in time. These computations have to process this stream of data at least as fast as new neural data is acquired.

If all computations are fast enough to be carried out by one thread before a new neural data is acquired there is no need for parallelization and one can do the computations with one thread, using at most one CPU core, see Figure 15 A. In the proposed software architecture, one can implement such a configuration by concatenating filter algorithms to a compound filter algorithm, here termed 'filter algorithm chain'. The output of each filter is forwarded as input to the next filter in the chain. The filter algorithm chain is then given to a filter for execution by its worker thread.

If the computation time required by the signal processing steps is too long to be handled by one thread, one can use a first level of parallelization by dividing the main processing steps (i.e., signal acquisition, feature extraction, movement decoding, and the generation of feedback) into separate threads (Figure 15 B). This degree of parallelization can use as many cores as there are processing steps since every thread can occupy only one separate core at any time. Existing facilities of the operating system, in this case the Windows scheduler, are used to distribute the workload of the threads among CPU cores. However, since each step depends on the result of the previous step most computations are not carried out in parallel.

In BCI software applications, many processing steps transform the signal on one channel independently from the signal on other channels (e.g., Fourier transformation or band-pass filters). This can be exploited for a higher degree of parallelization, where a processing step can be divided into functionally independent substreams, each processing a subset of channels. Each data substream can then be processed by a separate thread (Figure 15 C*, for an example see Wilson and Williams (2009)). Here, this level of parallelization was termed as 'independent substream parallelization'. The same principle can be extended to the processing of the decoding step. For example, if the BCI application is decoding the intended movement and each DOF of the movement can be computed independently from the other movement DOF (e.g., in a linear filter) then the decoding algorithms for each DOF can be run on a separate thread.*

There are ways to achieve even higher degrees of parallelization. In principle, all functionally independent parts of a processing step can be run on separate threads. However, such 'fine grain' parallelization requires exploitation of the specific processing algorithm used. Therefore, there are no gen-

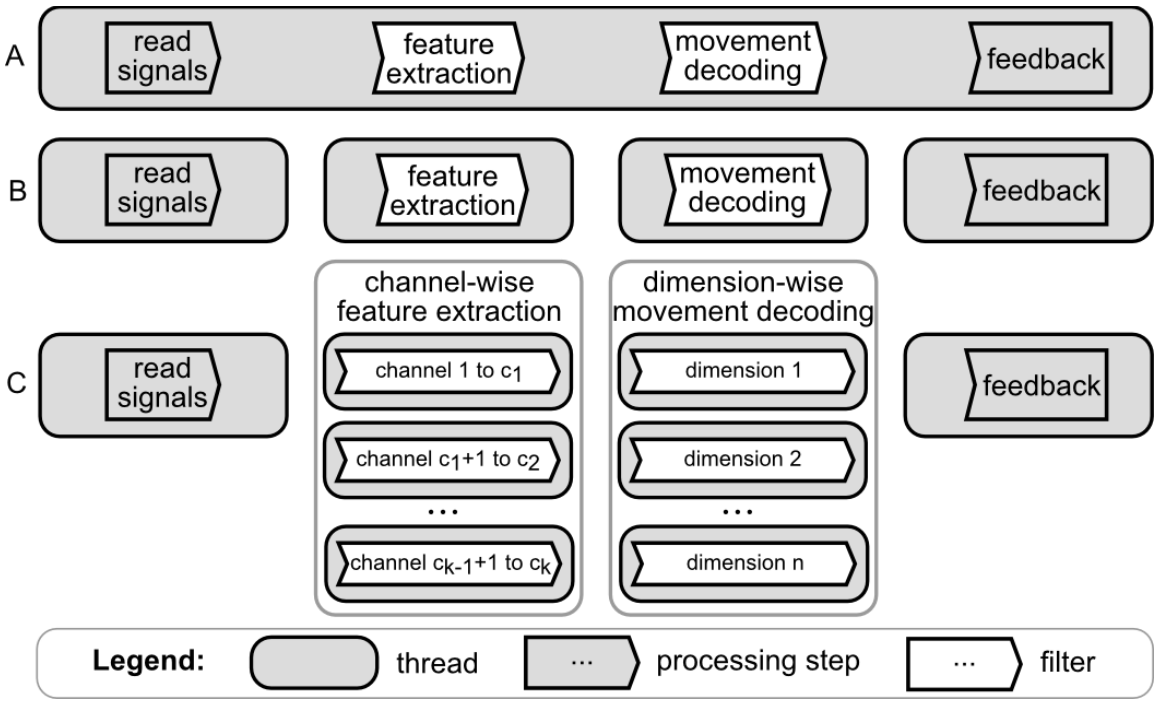


Figure 15: Degrees of parallelization for a typical BCI software

Figure shows processing steps applied to a stream of neural data by a BCI application. A: No parallelization (i.e., all processing steps are executed sequentially in one thread on a single CPU core). B: each processing step is executed in a separate thread. The computations of each thread can be handled by a different core. C: independent substream parallelization. Feature extraction and movement decoding steps are split into functionally independent computations (e.g., by computing the features for each channel in a different thread). Figure from Fischer et al. (2014).

eral ways to explore such ways of parallelization. Furthermore, current BCI systems process recordings from a large number of channels. Therefore, even by using 'independent substream parallelization' one can divide the processing into a number of substreams that exceed the number of available CPUs used to run BCI software applications by far. Later in Section 7.1 it is shown that, when the number of threads is much larger than the number of available CPUs, one cannot expect additional gains in performance. For all these reasons, parallelization on a degree finer than the 'independent substream parallelization' was not considered.

6.3 Performance Analysis

6.3.1 Performance Definition

In Section 5.3 performance of a BCI was defined to encompass computational power, latency and CPU load. The methods for the assessment of the proposed software architecture's performance are described in the following.

The overall latency L of a BCI (i.e., the time the system needs to react to the user's (neurophysiologic) input) was defined as follows:

$$(1)\ L \coloneqq L_{acq} + L_{comp} + L_{par}$$

The term L_{acq} encompassed the time span needed by the data acquisition hardware system to digitize the analog neurophysiologic signal and make it available to the BCI software for processing. The digit-

ized recording values are usually sent in blocks comprising of recording values from multiple points in time. If the block size is greater than 1 (i.e., comprises recording values from more than one point in time) it inherently adds additional latency to L_{acq}. As the worst case, consider an algorithm that wants to process the recorded values from the least recent point in time of a block. It has to wait until all values of a block have been recorded (and made available to the BCI software system) before it can start processing. In the following analysis, block sizes larger than 1 were not considered for two reasons: Firstly, because the block size is determined by the recording system and cannot be influenced by the proposed software architecture. Secondly, the final Braincon implant will use a block size of 1, so the results of this analysis will be valid for this version of Braincon.

The latency resulting from the time needed to process the recorded data (i.e., execute the filter algorithms) is denoted as L_{comp}. Therefore L_{comp} depends on the implementation of the used filter algorithm(s). It was beyond the scope of this performance analysis to evaluate the effects of different implementations of the same filter algorithm because this performance analysis focusses on the proposed software architecture.

On standard desktop computers or laptops the amount of computations a processor core can execute per time is limited and may be insufficient for BCI signal processing. One can alleviate such limitations by distributing calculations among multiple cores. According to Amdahl's law (Amdahl, 1967) computation time L_{comp} of an algorithm decreases linearly with the number of processor cores used. However this is the optimal case. There are additional latencies caused by parallelization which are denoted as L_{par}. It takes additional time to split the recorded data into multiple parts for independent processing and time for merging the processing results. Synchronization between threads and processes also takes time (e.g., when one thread waits for another thread to yield some input or to release a shared resource). L_{par} was influenced by the way the parallelization is implemented (i.e., by the software architecture) as shown in Section 7.1.

Another aspect of a BCI system's performance is the CPU load it consumes while processing neurophysiologic data. Low CPU usage is desirable for several reasons. Firstly, low CPU usage leaves more CPU time as a reserve for other processes or the operating system. Given a sufficiently large reserve, a BCI system still has enough CPU time and is not influenced by sudden increases in CPU load (e.g., if a virus scan starts or the operating system performs a CPU-consuming background operation). Secondly, if the BCI system is running on a mobile battery-powered computer system, a lower CPU usage can increase the run-time of such mobile BCI systems by consuming less energy. Thirdly, by using less CPU time, one can use the remaining CPU time as additional computational power for additional or more demanding feature extraction or decoding algorithms to improve decoding performance.

Active filters were excluded from the following analysis for two reasons: Firstly, the latency caused by active filters is dominated by the time interval t in which the filters check for new input. In the worst case, input data arrives immediately after the last check, so in the worst case latency is <t. This time interval can be chosen and adapted by the operator: if t is increased CPU load decreases while latency increases because data processing is initiated less frequently. Secondly, for computationally expensive feature extraction and decoding algorithms it usually does not make sense to use an active filter that starts processing at fixed time intervals. Instead one would prefer to use a passive filter that processes data as soon as it can.

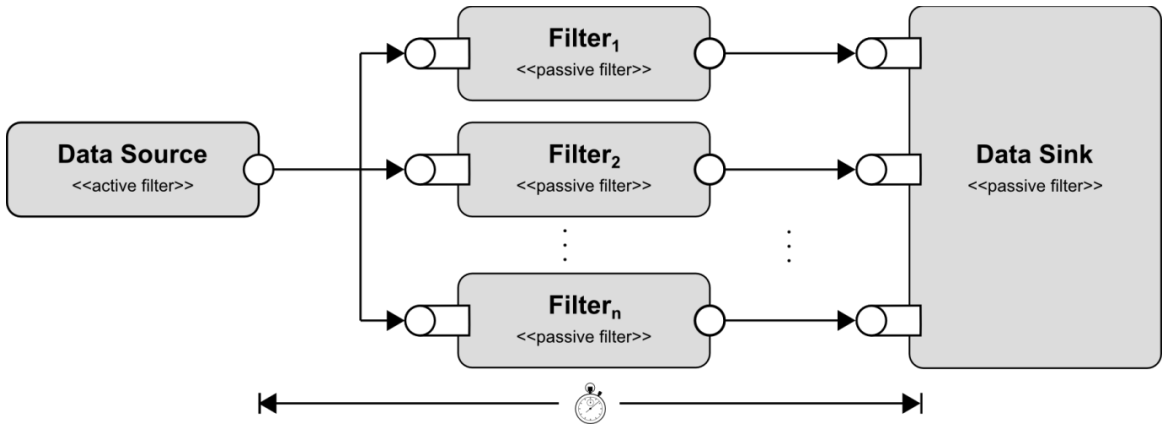


Figure 16: Filter pipeline setup for simulation

Data is generated from a data source and distributed to n filters for processing. Results are sent to a data sink. Processing time (i.e., the time from data source to data sink) was measured. Double brackets indicate the used filter type. As the data source generates data with a fixed frequency, it has to be realized as an active filter.

6.3.2 Simulation Setup

Performance of the proposed software architecture was evaluated on a standard desktop PC with an Intel Core i7 970 computer with 3.2GHz and 24GB memory running 64-bit Microsoft Windows 7 Enterprise operating system. The performance values will vary on different hardware and software configurations. However, the author is convinced that the conclusions drawn from the quantitative results will be valid for a wide range of commonly used desktop and laptop computer systems.

To measure the performance, simulations of a filter pipeline were run as shown in Figure 16. From an artificial data source data objects were passed to the filter pipeline at a fixed frequency. This artificial data source was a software simulation of a neurophysiologic recording system. In each simulation session, the data source provided 120 seconds (s) of recordings from 256, 512, 768 or 1024 channels, sampled at a frequency of 256Hz, 512Hz, 768Hz or 1024Hz. The filter pipeline consisted of one or more filters that processed the data (described in more detail in Sections 6.2.4 and 6.3.4). This setup simulated a parallelized processing step like feature extraction or decoding as outlined in Figure 15 C. The degree of parallelization (i.e., the number of filters and therefore the number threads) could be varied.

For each waiting strategy $w \in \{Polling, Wait_0, Wait_1, Wait_{event}\}$ *and number of threads the time required for the processing of a data object was measured as the latency (i.e., the time between the injection of the data object into the filter pipeline and the reception of the resulting all data objects of one processing step at the data sink). Since some of the algorithms considered here used a sliding window with a step size of n sample points, only every n-th input packet yields an output data packet and thus a latency value. One latency measurement is denoted as* $L_{w,i,j}$*, where w is the waiting strategy used, i is the number of threads and* $j \in \{1,2,\cdots,k\}$ *is the j-th measurement of the latency. The performance of the filter pipeline was quantified by three measures:*

Median latency: *Due to computational processes initiated by the operating system, the processors used for simulations would occasionally be used for other computations. This would cause latencies much higher than expected for one or more consecutive measurement points in time. To reduce the*

effect of such outliers, the median latency for a waiting strategy w and i threads was defined as follows:

$$(2)\ M_{w,i} \coloneqq median_{\ j\in\{1,2,\cdots,k\}}\left(L_{w,i,j}\right)$$

In addition, in each simulation, the first 16 latency values (corresponding to 0.5s of data) were removed from each simulation in order exclude transient latency values at the start of the simulation.

Median of relative latency reduction (MRLR): *The normalization factor $N_1 \coloneqq min_{w\in W}\left(M_{w,1}\right)$ was defined as the median latency of the waiting strategy with the minimum median latency when only one thread was used. All other latencies were compared to such a normalization factor. Furthermore, for each latency $L_{w,i,j}$, the relative latency reduction (RLR) was define defined as*

$$(3) RLR_{w,i,j} \coloneqq \frac{N_1}{L_{w,i,j}}$$

By construction, $RLR_{w,i,j}$ equals 1 if the latency measurement equals median latency of the fastest waiting strategy with one thread, is above 1 if $L_{w,i,j}$ is smaller than N_1 indicating an increase of performance relative to the fastest waiting strategy with one thread and is below 1 if $L_{w,i,j}$ is larger than N_1. To capture the general dependence of the latencies as a function of the number of threads, median of relative latency reduction was defined as

$$(4)\ MRLR_{w,i} \coloneqq median_{j\in\{1,2,\cdots,k\}}\left(RLR_{w,i,j}\right)$$

Thus, $MRLR_{w,i}$ is larger than 1 if the computations are (in the median) performed faster than with the fastest waiting strategy using one thread.

CPU load: *Once per second during each simulation, CPU load was measured. Similar to latency measurement, computational processes can be initiated by the operating system and thus the processors used for simulations would occasionally be used for other computations. This would cause a CPU load which is temporarily much higher than expected for one or more consecutive CPU load measurements. To reduce the effect of such outliers, if was reported on the median percentage of the total possible load of one core (100%). CPU load measurement was started some time before the simulation started and thus initially CPU load was measured from a period of inactivity. To exclude this period, the first two CPU load measurements (corresponding to 2s of data) were removed. The test system contained 6 cores with Hyper Threading which can allow two threads being processed in parallel on one core (Marr et al., 2002), so the maximum number of parallel threads, that can run at 100% CPU load, is 12. However, during simulations, one thread simulated the data source and one thread was used to receive the results of the filter pipeline (Figure 16). In addition, some processing was required by the operating system. Therefore the real number of threads that could be used exclusively by the simulated filter pipeline in parallel was lower than 12. It was assumed that at least 9 threads were available for full 100% CPU load assuming that the operating system, the data source simulation and the acquisition of results from the simulation each occupied one thread at most.*

6.3.3 Stall Definition

A minimum requirement for a BCI application is that the filter can process the input data at least at the rate it is coming in. If processing is not fast enough the data objects will accumulate at the input ports of a filter. Consequently, the latencies will add up, making the application unable to react to

inputs. Additionally, the memory usage will grow until it reaches the limit of available memory which will cause the application to terminate. Here this scenario was referred to as stall. During simulations a test run was considered to be stalled if one of the following conditions was true:

- *Memory consumption exceeded a threshold of 750 megabytes (MB).*
- *The overall calculation time exceeded the expected time by more than 1.2s.*

The memory limit of 750MB was exceeded if more than 5s of data accumulated at the input port of one of the filters. The memory limit was chosen big enough to reduce the number of falsely detected stalls as much as possible while giving the application sufficient remaining memory to shut down normally. If the system took more than 1.2s (1%) beyond the expected execution time of 120s, it was fairly safe to conclude that the latency of the processing was too high. If the application was allowed to continue to run the data would accumulate and the program would eventually stall. While there is some necessary arbitrariness in the time limit of 1.2s, it was chosen to reduce falsely detected stalls due to temporal jitter caused by other processes.

6.3.4 Algorithms

Algorithms used in the BCI applications mostly fall into two groups: (i) feature extraction algorithms and (ii) decoding (classification and regression) algorithms. In most BCI applications these algorithms will take most of the computational time. Therefore, to evaluate the performance of the proposed software architecture several feature extraction and decoding algorithms frequently used in BCI applications were tested.

6.3.4.1 Feature Extraction Algorithms

In the context of BCIs, feature extraction algorithms are algorithms used to calculate features from neuronal signals. Two frequently used feature extraction algorithms were considered: short-time Fourier transform (Allen, 1977) and Savitzky-Golay filter (Milekovic et al., 2012; Savitzky and Golay, 1964) as an example of a linear filter for smoothing signals. The implementation of the Savitzky-Golay filter used here was more demanding than the implementation of the short-time Fourier transform, since it used one forward Fourier transform and one inverse Fourier transform in each calculation step.

6.3.4.2 Decoding Algorithms

BCI related decoding algorithms can be grouped into algorithms for inference of continuous variables (regression) and algorithms for inference of discrete variables (classification). For classification, filter pipelines implementing linear discriminant analysis (LDA) (Hastie et al., 2009) and support vector machines (SVM) (Vapnik and Chervonenkis, 1974) were tested. For regression the linear filter (LF), support vector regression (SVR) and the Kalman filter (KF) (Kalman, 1960; Haykin, 2001) were tested. The libsvm library (Chang and Lin, 2011) was used to implement SVM and SVR, and the GNU Scientific Library (Galassi, 2009) was used for linear algebra operations. For all tests of classification/regression algorithms, the sampling frequency was kept fixed at 1024Hz and decoded one DOF (i.e., a one-dimensional movement). In the context of classification or regression algorithms, the computational complexity regarding the inputs is best measured by the number of features that have to be processed in each decoding step, rather than by the number of channels, as each channel might provide multiple features (e.g., data from multiple delays in respect to the current time or from multiple frequency bands). In addition to the computational complexity arising from the number of features, the computational complexity of the SVR and SVM algorithms depends on the number of support vectors.

Therefore the number of support vectors for all tests involving the SVR and SVM algorithms was varied. All feature extraction and decoding algorithms were executed 32 times per second (in steps of 31.25ms).

6.3.4.3 Decoding Algorithm Complexity

For each algorithm, parametric models of the latency L as a function of feature dimension were derived. Since only the feature dimension varied, the terms L_{acq} and L_{par} (cf. Equation 1) are constant and can thus be expressed as constant terms in each model. For SVR and SVM latency was also a function of the number of support vectors. These models were fitted to the measured latency values (cf. Section 7.1) to estimate the parameter values. In each model n_f denotes the number of features and n_s the number of support vectors.

Linear filter and linear discriminant analysis: *One DOF decoding using LF models or binary classification using LDA models is equivalent to a scalar product between a feature vector and a vector of fixed coefficients (obtained by calibrating the model on the training data). Therefore the computation time is linear in the number of features (Hastie et al., 2009). For LF and LDA decoders the latencies were therefore modeled as $L = b \cdot n_f + a$.*

Kalman filter: *In the limit of high number of features the computation time of the Kalman filter algorithm is dominated by the inversion of a symmetric matrix. The Cholesky factorization algorithm was used for matrix inversion (as the matrix to be inverted is symmetric) and the computational complexity of Cholesky factorization is a cubic function of a number of features (Santos and Chu, 2003). All other operations have a linear or quadratic dependence on the number of features. Profiling the implementation of the Kalman filter used here revealed that with $n_f = 128$ and decoding one DOF around 94% of the computation time in each iteration was used for the matrix inversion. Thus all simulations were made in the regime of a high number of features. Therefore the contributions of all other Kalman filter calculations were neglected in the model of the Kalman filter latencies: $L = c \cdot n_f^3 + a$.*

SVR/SVM: *Computational costs of SVM/SVR scale as $O(n_f \cdot n_s)$ (Burges, 1998). Latencies were therefore modeled as $L = d \cdot n_f \cdot n_s + a$.*

6.3.5 Statistical Analysis

Four different waiting strategies for the filter pipeline were considered (cf. Section 6.2.4.2) and their performance was measured using three different measures: median latency, median of relative latency reduction and CPU load. Performance was examined as a function of the number of threads and across a range of sampling frequencies and numbers of channels (Supplementary Figures i-iii, Figure 19). For each combination of waiting strategy, number of threads, sampling frequency and number of channels ten simulations were run yielding 38 240 latency measurement and 1180 CPU load measurements in total. To assess the significance of the performance increase with an increasing number of threads the performance measures were compared between 1 and 9 threads by a Wilcoxon rank sum test (Gibbons and Chakraborti, 2011). The Wilcoxon rank sum test of MATLAB R2010a (The Mathworks Inc., Nattick Massachusetts, United States) uses an approximation to compute p-values in the case of large samples. Given potential inaccuracies of this approximation, here was simply reported $p < 0.0001$ even if actual p-values were much smaller. The null hypothesis was considered as rejected if $p < 0.05$. This test was carried out for different sampling frequencies and number of channels (cf. Sections 7.1 and 10.1).

6.4 Closed-Loop BCI Study

In order to evaluate whether the domain model presented in this thesis was sufficient to map a closed-loop BCI application and to show its closed-loop capability, the Freiburg BCI Software was employed in a closed-loop BCI study for decoding of kinematic movement parameters from ECoG.

Five human patients suffering from intractable pharmacological resistant epilepsy participated in this study after having given their informed consent. The study was approved by the Freiburg University Ethics Committee (Milekovic *et al.*, 2012).

There were two types of experimental sessions: training sessions to collect ECoG data while the patient performed wrist movements, and brain-control sessions where movements were decoded online from ECoG data. Previously collected data (e.g., from training sessions) was analyzed offline with MATLAB to derive the parameters used for decoding in brain-control sessions.

A session consisted of multiple trials and each trial consisted of several phases. At the beginning of each trial, a visual cue displayed on a screen instructed the subject whether to move left or right (cue phase). After a pause (pause phase), a go cue was displayed (movement phase). In training sessions subjects then had to move a joystick, either to the left or to the right, and then hold the joystick position for two seconds. Then a visual feedback was presented by a ball moving from the center of the screen to a target in the cued direction (feedback phase). If the movement was not executed correctly (e.g., into the wrong direction or movement was started too early or too late) an error message was displayed and the trial was discarded (error phase). Brain-control sessions were similar to training session, except that movement direction was controlled by the posterior probability of the cued direction. Posterior probability was decoded online from the ECoG data. The ball in the visual feedback moved proportional to the posterior probability towards the target (e.g., if the decoded posterior probability for direction 'left' was 0.75, then the ball would move 75% of the distance towards the left target). A trial was considered successful, if the posterior probability was greater than 0.5. There were two variants of brain-control sessions: in one variant subjects performed actual wrist movements. In the other variant subjects imagined wrist movements. However, in both variants ball movement was controlled by the decoded posterior probability. For more details on the task, cf. Milekovic *et al.* (2012).

Several processing objects were developed for the Freiburg BCI Software to realize this experimental setup in terms of software.

Joystick reader: This processing object continuously polled the position of a joystick. To support different joysticks with different resolutions along x- and y-axis the position was normalized between [-1.0, +1.0] for each axis. An output port provided the normalized position for further processing.

Movement discretizer: As input, this processing object required a two-dimensional movement position, ranging from -1.0 to +1.0 for both x- and y-axis. The position along each axis was divided into five regions, depending on the distance to the center (0, 0): full negative reflection, negative deflection, centered, positive deflection and full positive deflection. These regions were defined for each axis by thresholds. Every time a threshold was crossed (i.e., movement position reached a new region) an event was generated containing the current x- and y-regions and thus, the continuous two-dimensional space was transformed into a discrete space of 25 distinct positions. Consider as example that the joystick would be deflected horizontally to the very left (i.e., normalized position would

be (-1, 0)). Then the corresponding two-dimensional discretized position would be (full negative deflection, centered).

Paradigm model: In the context of this BCI study a 'paradigm' was the virtual setting with which the patient interacted. This encompassed the position of the ball, the target position, the current trial number and the phase of the current trial. The paradigm model was a processing object which used a finite state machine. Each phase was represented as a state. Either actual patient movement, decoded movement or time (e.g., by a timeout) triggered a state transition. Therefore it had two input ports, one for actual movement (i.e., discretized position from movement discretizer (see above)) and one for decoded movement (i.e., decoded posteriori probability from RLDA decoder (see below)). An output port published state transition events which contained all relevant information to display the current state of the paradigm model.

Paradigm view: The paradigm view displayed the state of a paradigm model. Its graphical user interface used the user interface development framework Qt (Digia Qt, Oslo, Norway). A paradigm view could operate in two modes: experimenter mode and subject mode. In subject mode the paradigm was rendered normally as it should be seen by the subject. In experimenter mode, additional information relevant for the experimenter was displayed like current posterior probability, decoding accuracy and trial number. With this information the experimenter could give detailed feedback to the subject (e.g., by saying 'You're doing well' when decoding accuracy is good or 'Almost done.' to motivate the subject for the remainder of the session).

Common average referencing filter: This processing object applied the common average referencing algorithm (Ludwig *et al.*, 2009) to a stream of measurements. A measurement consisted of a set of n voltage values $(u_1, u_2, \cdots, u_n)$ measured at one point in time where n was the number of channels. Let $S \subseteq \{1,2,\cdots,n\}$ be a non-empty subset of the available channels. For each channel $c \in \{1,2,\cdots,n\}$ the common average referenced voltage value u'_c of u_c was defined as followed:

$$u'_c := u_c - \frac{1}{|S|}\sum_{s \in S} u_s$$

Running average filter: The running average processing object removed temporary offsets from measurement values similarly to a high-pass filter. Let u_t be a measurement value from one channel at time t, where $t \in \mathbb{N}$. Let further $n \in \mathbb{N}^+$ be the number of points in time to be included into the calculation of the running average. For each u_t, the running average filtered result was defined as followed:

$$u'_t := \begin{cases} u_t - \frac{1}{t+1}\sum_{i=0}^{t} u_i & if\ t < n \\ u_t - \frac{1}{n}\sum_{i=t-n+1}^{t} u_i & if\ t \geq n \end{cases}$$

The number of points in time could be adjusted for each subject.

Decoding algorithm: A processing object implementing a decoding algorithm continuously buffered the measurement values from the last n seconds, whereas n was adjustable for each patient. This buffer computed features, as determined by the previous feature selection process, from the past

measurement points using a Savitzky-Golay low-pass filter (Savitzky and Golay, 1964). For details on the feature selection process cf. Milekovic *et al.* (2012). A regularized linear discriminant analysis (RLDA) classifier (Friedman, 1989) calculated the posterior probabilities for the movement classes 'left' and 'right' from the features. This processing object did not yield a continuous stream of decoded probabilities. Instead, decoding could be triggered through an input port. This way the time of the decoding was more closely coupled to the time of external events like 'joystick position reached maximum deflection'.

Brainbox amplifier: The processing object read out the measurement values of a Brainbox EEG-1164 amplifier device (Braintronics B. V., Almere, Netherlands). This EEG-1164 amplifier had a sampling frequency of 1024Hz. It provided one output port for publishing of measurements for further processing. In addition, the processing object wrote the measurements onto a hard disk.

Neuvo amplifier: Similar to the Brainbox amplifier, this processing object read measurements from a Neuvo amplifier device (Compumedics Limited, Abotsford, Australia). Here the sampling frequency was 2500Hz.

6.5 Preclinical *in vivo* Animal Study

The objective of the following preclinical animal study was to validate the measurement and stimulation functionality of Braincon *in vivo* and to assess long-term stability timed until the implant eventually fails. This study was approved by the Animal Committee of the University of Freiburg and the Regierungspräsidium Freiburg, Baden-Württemberg. It was conducted in compliance with Directive 2010/63/EU (European Parliament and Council, 2010).

6.5.1 Software for *in vivo* Study

For this study, the Freiburg BCI Software was used together with five processing objects which are presented in the following paragraphs.

Braincon implant PO: The Braincon implant processing object (BPO) implemented access to Braincon implants. With this processing object, ECoG measurements in the form of a continuous data stream were read out from an implant. Every time electrical stimulation was started or stopped by the implant, these events were marked in the data stream. Other modules used such events to determine the sections of the data stream where stimulation was active for further analysis or signal processing, see the EPA module as an example below. Due to the wireless data transmission from and to the implant the BPO had to cope with connection losses. In the case of a connection loss the stream of measurements was marked invalid with special values. This way the data stream continued to flow and all other processing objects that process this stream could decide individually how to handle gaps in the measurement stream (e.g., they could choose to ignore the data or could try to interpolate the missing measurements). The implant sent data in blocks of 10ms.

The BPO could issue electrical stimulation. Since Windows 7 is no real-time operating system, it is not guaranteed that the Freiburg BCI Software always has the computational resources to control the electrical stimulation of the implant (i.e., to turn stimulation and off). Therefore a dead man switch in the implant's firmware protected against unwanted endless stimulation. The maximum duration of a stimulation command accepted by the firmware was internally restricted, currently to 1s. The firmware could guarantee this temporal constraint because the firmware operated under hard real-time conditions. The BPO internally decomposed stimulations with durations greater than 1s into sequences of short stimulation commands with a duration of <1s.

The measurement stream had an additional 32-bit counter for the measurement stream to check its completeness. To assess the implant's power supply its supply voltage was measured with 1kHz and sent along the measurement stream. In addition, the implant's temperature and internal humidity could be read out by the BPO.

gUSBamp amplifier PO: The gUSBamp processing object (GPO) recorded measurements from the gUSBamp amplifier (gtec, Schiedlberg, Austria). This amplifier served as a reference device for the Braincon implant.

Evoked Potential Analyzer: Evoked Potential Analyzer (EPA) was a processing object that served to detect evoked potentials at runtime. Whenever a stimulation pulse was emitted an event marker was inserted into the stream of ECoG measurements. When the EPA detected such an event marker, the EPA took a window (with a configurable width) of the neural signal data around the event for each channel, then processed it according to an exchangeable algorithm (e.g., incremental mean and standard deviation for each point in time) and yielded a set of representable traces for on-screen display.

Stimulation script editor: The stimulation script editor (SSE) was a processing object for automating the stimulation procedure. Such scripts could be used in experiments that involved long and complex sequences of stimulation commands or experiments where stimulation parameters had to be changed faster than a human operator could enter them through the GUI. The two basic commands of a script were stimulation and pause commands, each with different parameters. The parameters of a stimulation command were amplitude, pulse width and duration. A pause command had its duration as a parameter. The basic commands could be executed in sequences. These sequences in turn could be iterated over any parameter (e.g., one could iterate a sequence by increasing the amplitude). The SSE allowed creation and execution of scripts through a GUI as well as loading and saving scripts from and to a file.

Stimulation script controller: The stimulation script controller (SSC) executed the scripts created by the SSE by translating stimulation sequences and iterations into actual stimulation commands for a Braincon implant. The separation of script execution from stimulation control allowed the usage of different script sources: a script could either originate from the user through the script editor or programmatically from another processing object. This allowed open-loop application where scripts were issued from the SSE, (e.g., for brain mapping) but also supported future closed-loop applications (e.g., a decoder issuing stimulation sequences based on current neural activity).

6.5.2 Acute Animal Study

In the following the methods for validation of the Braincon implant's *in vivo* measurement functionality will be presented. The ECoG signals recorded by the Braincon implant are compared with ECoG signals recorded by an established and commercially available ECoG amplifier. This study used sheep as animal model because these animals were big enough for subsequent chronic implantation studies (Gierthmuehlen *et al.*, 2014). The main idea was to electrically stimulate the sheep's nose and compare the resulting evoked potentials on the sheep's somatosensory cortex when measured by a Braincon implant and a gUSBamp device. Two sheep, A1 and A2, were implanted in this acute setting.

A schematic overview of the experimental setup is provided in Figure 17 A. Surgeons implanted a 32 channel micro ECoG electrode grid (cf. Figure 17 A-2, Figure 18 A and B, Kohler *et al.*, 2012) above the sheep's somatosensory cortex because a sheep's facial part is represented with a fairly large area in the sheep's somatosensory cortex (Johnson *et al.*, 1974). Electrical stimulation was applied to different parts of the sheep's mouth and nose (lower lip, upper lip, chin and nose, each laterally and ipsilaterally) with an isolated ML180 stimulator controlled by a PowerLab 8/30 (all AD-Instruments,

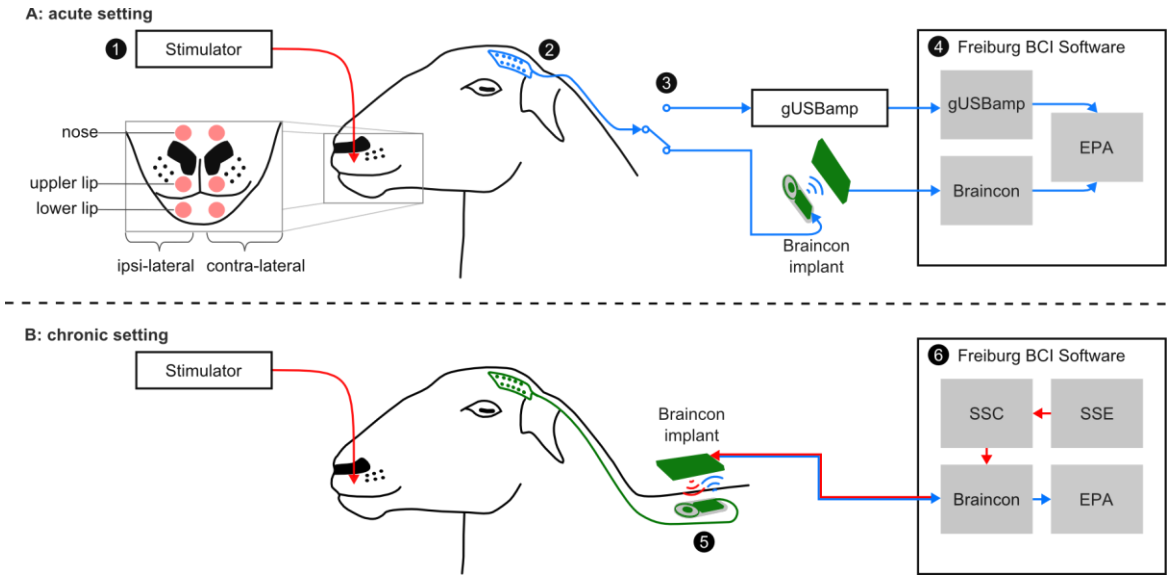


Figure 17: Scheme experimental setups

Color red indicates stimulation, blue indicates measurement. Gray boxes depict processing objects. A: Scheme of acute setup. 1: Body-external electrical stimulator connectable to various facial locations. 2: ECoG electrode grid over sheep's somatosensory cortex with percutaneous leads. 3: Connector solution which allows switching of recording between a Braincon implant and a gUSBamp device. 4: Freiburg BCI Software configured for online evoked potential detection. B: Scheme of chronic setup. 5: Braincon implant housing between sheep's shoulder blades. 6: Freiburg BCI Software configured for online evoked potential detection and automatic stimulation. Figure from Kohler and Fischer.

Spechbach, Germany), see Figure 17 A-1. Although the electrode grid had 32 electrodes the Braincon implant supported only 16 recording channels. In order to achieve a good coverage of the eloquent areas with 16 channels, ECoG was first recorded from all 32 contacts with the gUSBamp. Based on this information, an area comprised of 16 electrode contacts with good coverage of the eloquent areas was selected. The selected channels for sheep A1 and A2 are shown in Figure 18 A and B.

Each stimulation episode lasted 300s, stimulation pulses varied in intensity (1mA, 2mA and 4mA) while pulse width and frequency were constant at 100µs and 2.0202Hz respectively. The interval between two stimulation pulses was chosen to be long enough so that one can safely assume that the effects of the previous pulse would not overlap with the effects of the current impulse. In addition, the chosen frequency was not a divisor of the sampling frequencies of the Braincon implant, gUSBamp amplifier or power supply (f=50Hz), so that one could exclude recording of periodic technical artifacts from sampling or synchronization with 50Hz interference. Over a percutaneous cable the electrode grid was connected to a connector solution which permitted the manual switching between Braincon implant and gUSBamp amplifier (Figure 17 A-3). This way, different devices could record ECoG signals without relocation of the electrode grid. During all experiments, gUSBamp amplifier recorded with a sampling frequency of 4800Hz and Braincon implant recorded with a sampling frequency of 1000Hz.

Both devices were connected to a PC running the Freiburg BCI Software (Figure 17 A-4) for data recording and online signal processing by the EPA. For each channel the EPA applied a three-step online processing algorithm: First, for each trigger event and each channel a window of measurement data ranging from 100ms before until 200ms after the trigger event was extracted for further processing. Then each window was normalized by subtracting the mean of the window from 100ms to

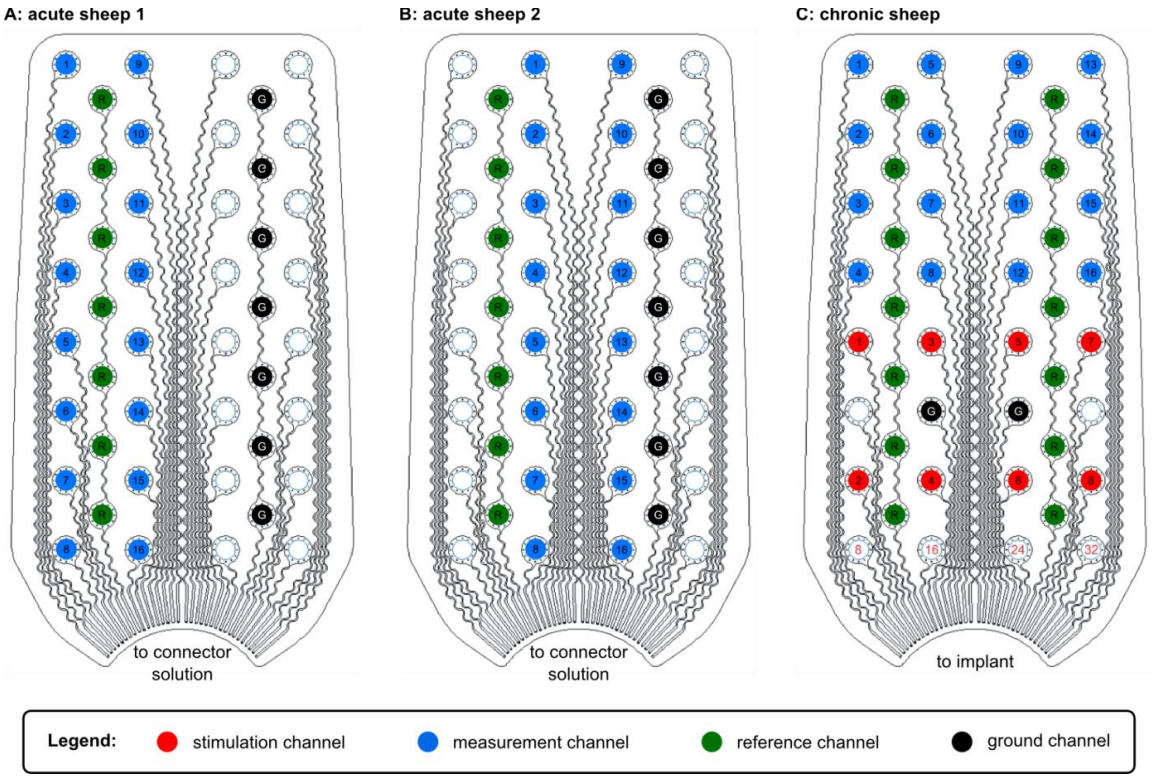


Figure 18: Clinician's view of the electrodes used for the acute and chronic setting

Numbers indicate assigned channel numbers used for analysis. Colors blue, red, green and black indicate the type of channel: measurement, stimulation, reference or ground channel, respectively. A: Electrode used for first acute sheep. B: Electrode used for second acute sheep. C: Electrode used for chronically implanted sheep. Parts of this figure (i.e., the electrode layout scheme in the background) were designed and manufactured by Christian Henle. Figure from Kohler and Fischer.

50ms before the trigger event. Finally the EPA calculated and displayed the median and the interquartile range over all windows available so far on screen.

6.5.3 Chronic Animal study

For chronic evaluation of the Braincon implant's measurement and stimulation functionality, a sheep C1 was chronically implanted with a Braincon implant. Surgeons implanted the electrode grid over somatosensory cortex and the implant housing between the shoulder blades (Figure 17 B-5). Implantation procedure is described in detail by Gierthmuehlen *et al.* (2014). In contrast to the acute setting, the connector solution was omitted in the chronic setting and the electrode grid connected directly to the implant, cf. Figure 17 B. Prior to implantation, 16 of the 32 available contacts of the electrode were chosen for recording and 8 contacts were chosen for stimulation (Figure 18 B). The first intended use of this configuration was to record somatosensory evoked potentials elicited by the body-external ML180 stimulator as in the acute setting. The second intended use was to analyze ECoG measurements with the EPA for any delayed responses from electrical stimulation by the Braincon implant itself. To facilitate this, the Freiburg BCI Software additionally employed the SSE and SSC processing objects for automatic execution of stimulation scripts (Figure 17 B-6). Using the SSE, experimenters applied stimulation pulses (1ms pulse width, 5Hz) to each channel with increasing amplitude (1V, 4V, 7V, 10V and 13V).

In order to predict failure of the Braincon system humidity and temperature values from the built-in sensor on the implant's electronics were collected during every recording event.

After the Braincon implant was explanted, the integrity of the capsule's solder-seal was analyzed by means of computer tomography (CT). This method allowed for the identification of any voids or non-adherent parts which could have caused an increase of humidity within the package and hence posed a hazard for the electronics. CT measurements were conducted at the Technisches Pruefzentrum in Neuss, Germany utilizing a nanoCT® system (phoenix nanotom m, General Electric Company, Fairfield, CT, USA). The 3D data was analyzed with the software tool myVGL 2.2 (Volume Graphics GmbH, Heidelberg, Germany).

7 Results

7.1 Performance Analysis

Figure 19 shows latencies, MRLR and CPU load at the extreme points of investigated parameter range (256Hz / 256 channels and 1024Hz / 1024 channels) for different numbers of threads. For all waiting strategies except for $Wait_1$, the latency generally decreased and the MRLR generally increased with more threads. Data was processed significantly faster when nine threads were used instead of one thread ($p < 0.0001$, Figure 19 and Supplementary Figures I and ii). Among all four waiting strategies, $Wait_1$ yielded the poorest overall performance gain when more than 1 thread was used. This was particularly pronounced for low sampling frequencies and low channel numbers (Supplementary Figures i and ii). The inferior performance of $Wait_1$ was due to querying for data at fixed intervals of 16ms which can be highly suboptimal if the data arrived at the input port just after the last query, thereby causing a higher increase in latency compared to the other waiting strategies. $Wait_{event}$ performs better for higher sampling frequencies and numbers of channels (Figure 19 B, D and Supplementary Figures i and ii). Compared to $Wait_0$ and Polling, $Wait_{event}$ performed worse for 256Hz and 256 channels ($p < 0.0001$), whereas for 1024Hz and 1024 channels $Wait_{event}$ performed similarly for up to four threads (median latency of $Wait_{event}$ differs at most 0.15ms, $p < 0.0001$), better for five threads ($p < 0.0001$) and slightly worse for up to nine threads ($p < 0.0001$, Figure 19 A-D).

Latency and MRLR measurements showed a drop in performance for five threads for Polling and $Wait_0$ and for six threads for $Wait_{event}$ and $Wait_1$ waiting strategies. Fischer et al. suspected that the combination of the system hardware and the scheduler of Windows 7 caused the system to perform slower for particular numbers of threads (e.g., five threads for Polling and $Wait_0$ waiting strategies and six threads for $Wait_1$ and $Wait_{event}$ strategies), depending on the number of cores available. To test this hypothesis simulations were rerun on a four-core machine and indeed showed the same drop in performance for three and four threads, respectively. A further test using the same hardware running Windows XP instead of Windows 7 revealed that the performance drop disappeared under Windows XP. Taken together, these additional tests support the initial presumption that the performance drop of Polling and $Wait_{event}$ was caused by Windows 7 when the number of used threads is equal to or one less than the number of available cores.

Further possible improvements in performance for the number of filter threads of up to 25 were explored, well above the total number of threads that could be used in parallel exclusively for filter pipeline simulation on the used six-core Hyper Threading system. In this regime, several threads would compete for the use of different cores. These simulations were run for a sampling frequency of 1024Hz and 1024 channels to get a lower bound estimate of MRLR and an upper bound estimate of latency and CPU load (Figure 19 B, D, F). While the performance of $Wait_{event}$ and $Wait_1$ increased further, MRLR of $Wait_0$ decreased when the number of threads was greater or equal to 15 and simulations stalled when 25 threads were used. Polling performance decreased for more than 10 threads and simulations stalled for 15, 20 and 25 threads (Figure 19 B, D, F). CPU load of Polling and $Wait_0$ strategies increased linearly with the number of threads until for 10 and more threads where the maximum CPU load was observed. For the $Wait_{event}$ and $Wait_1$ strategies CPU load remained at its low level of below 200%, even for up to 25 threads (Figure 19 E, F). Therefore, for 10 and more threads performance of the system may be influenced by the insufficient processing resources.

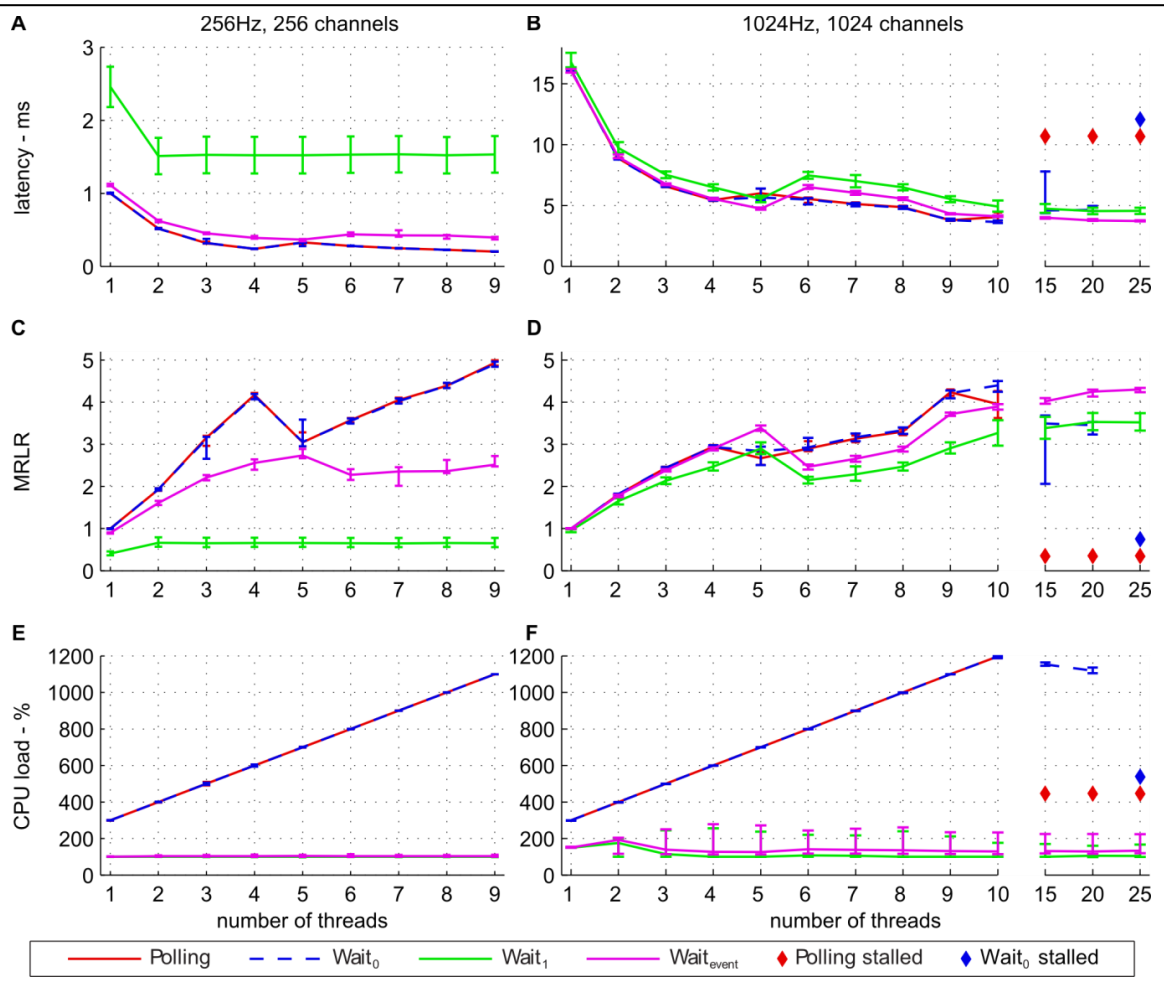


Figure 19: Performance of the filter pipeline

Performance of the filter pipeline implementing the short-time Fourier transform algorithm for different waiting strategies and different numbers of threads. Lines show median of latencies (A, B), median of relative latency reduction (C, D) and median CPU load (E, F). The latencies, MRLR and CPU load of the waiting strategies Polling and $Wait_0$ are very similar so that their graphs mostly coincide. Error bars show the 25% and 75% percentiles. Panels on the left (A, C, E) show results for simulations with 256Hz and 256 channels while panels on the right (B, D, F) show results for simulations with 1024Hz and 1024 channels. In addition, panels B, D and F (simulations with 1024Hz and 1024 channels) show median latencies, MRLR and CPU load for 10, 15, 20 and 25 threads. For high numbers of threads a diamond symbol indicates that at least one out of ten simulations stalled when the corresponding waiting strategy was used. Figure from Fischer et al. (2014).

7.1.1 Stalls of the Filter Pipeline

If the processing of one decoding step exceeds the amount of time available during two consecutive decoding steps, data will accumulate at the input ports of the filter pipeline and therefore the system will eventually be delayed increasingly and run out of memory. Here this incident was called a stall (cf. Section 6.3.3). It was investigated under which circumstances a stall could occur when the application required the complete translation of the neuronal signals to control signals (i.e., feature extraction and decoding) to be performed every 32ms. Such a decoding rate was typical for continuous control BCI applications and let the BCI user experience a smooth and virtually instantaneous control. For each number of channels and sampling frequency ten simulations were run. It was assumed that a certain combination of number of channels and sampling frequency can be handled if no stalls were detected in all ten repetitions. For all tested sampling frequencies and for all tested numbers of chan-

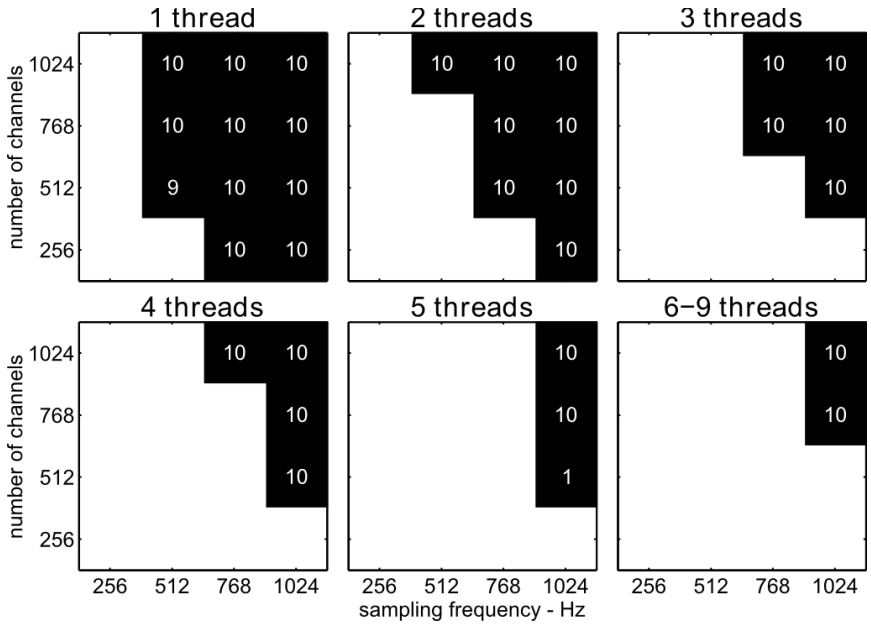


Figure 20: Stalls for a filter pipeline

Stalls for a filter pipeline implementing the Savitzky-Golay algorithm and using the Waitevent waiting strategy, shown for different channel number/sampling frequency combinations and different numbers of threads (1 to 9). A black square indicates combinations for which at least one out of ten simulations stalled, while a white square stands for no stalls. Numbers in black squares indicate how many stalls out of ten simulations occurred. Figure from (Fischer et al., 2014).

nels the Fourier transform algorithm could be handled by only one thread. Due to the computationally more demanding Savitzky-Golay filter algorithm, implemented as a forward and inverse FFT, the Savitzky-Golay filter resulted in stalls for various parameter settings (Figure 20). Stalls already occurred for moderate sampling frequencies and for realistic numbers of channels if only one thread was used. However, by increasing the number of threads, independent substream parallelization allowed to handle an increasing number of channels and higher sampling rates without stalls.

7.1.2 Performance of Classification/Regression Algorithms

Latencies were measured for filter pipelines that implemented either one DOF regression (LF, KF and SVR) or binary classification (LDA and SVM) algorithms (cf. Section 6.3.4.2 for details). Measurements were made for different numbers of features while the sampling frequency was fixed at 1024Hz. The $Wait_{event}$ filter implementation was used for all of these simulations since it provided a good compromise between performance (latency reduction) and efficient CPU usage (cf. Section 7.1).

The latencies of the LF and the LDA algorithms stayed below 0.1ms for up to 1024 features, increasing slowly with the number of features (cf. Figure 21 A, B) with a linear model fitting the latencies well. The computation time of LF and LDA decoders was negligible compared to the short-time Fourier transform feature extraction (which required a computation time of more than 15ms with 1 thread, 1024Hz sampling frequency and 1024 channels) with the used test system. The measured latencies of SVR increased linearly with the number of features and with the number of support vectors (Figure 21D, E). The measured latencies of SVM were almost identical to the latencies of SVR (Supplementary Figure iv).The latency values for SVM and SVR (Figure 21 D, E, F) were below 4ms for all tested values of the number of features (up to 1024) and support vectors (up to 1000), far below the time needed

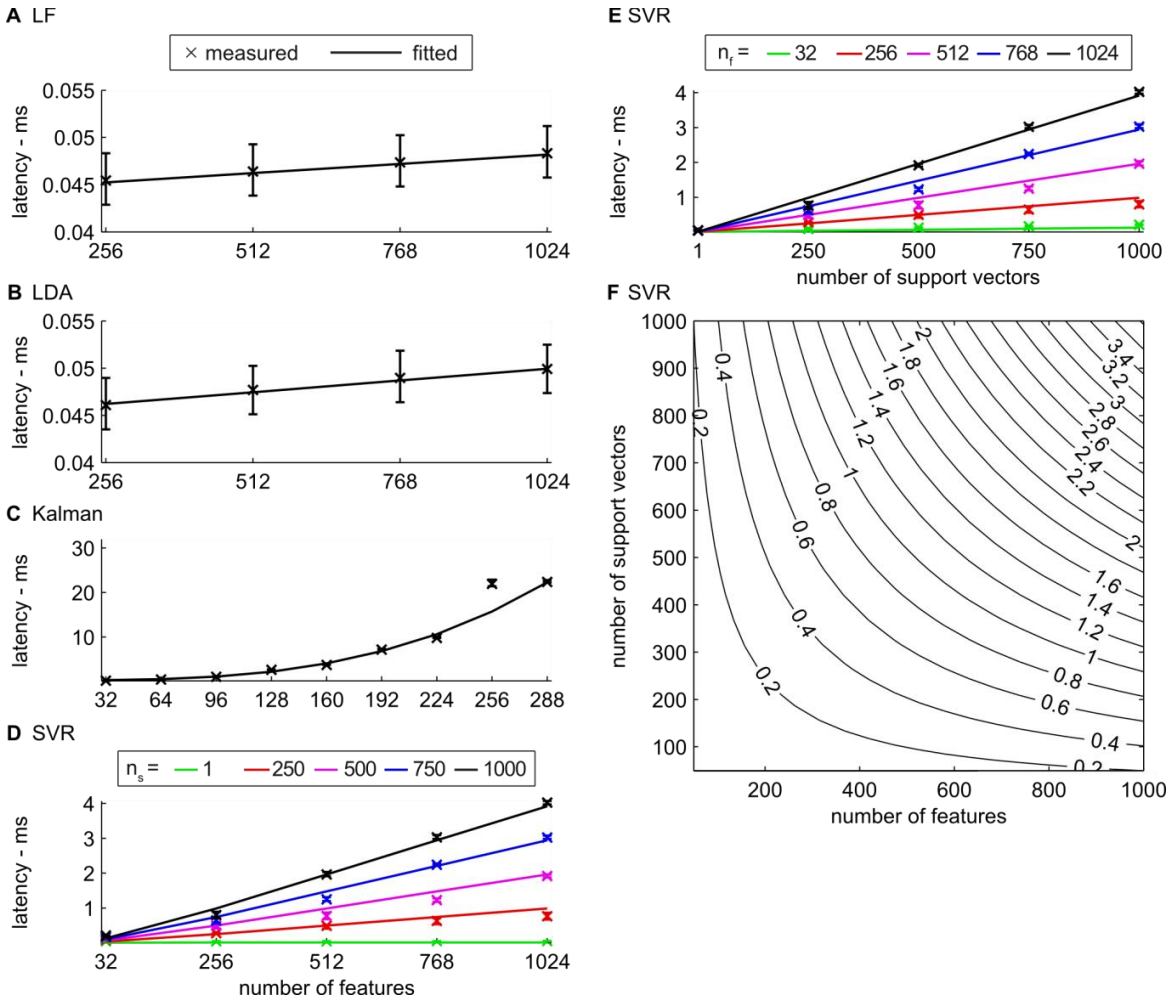


Figure 21: Latencies for classification/regression algorithms

(A) LF, (B) LDA, (C) KF and (D, E and F) SVR. Crosses depict median latencies and error bars show 25% and 75% percentiles. Solid lines show fits of algorithm specific models to the measured values. (D) Latencies for the SVR algorithm as a function of the number of features for n_s = 1, 250, 500, 750 and 1000. (E) Latencies for the SVR algorithm as a function of the number of support vectors for n_f = 32, 256, 512, 768 and 1024. (F) Latencies given by the model of the SVR algorithm as a function of n_f and n_s. Figure from Fischer et al. (2014).

for feature extraction. In summary, the parallelization of LF, LDA, SVM and SVR was not required, at least with the parameters and hardware considered here. In contrast, the computation times of the KF algorithm reached values similar to the feature extraction for about 200 features (Figure 21 C). For about 200 features and more the latency of the KF algorithm was within the range of the total time that was available for the decoding in continuous BCI applications. In such cases, parallelization of the KF might thus be desirable. Due to the nature of the algorithm, KF could not be parallelized using the independent substream parallelization suggested here. However, more complex parallelization schemes of the underlying computations could be employed (e.g., see Santos and Chu, 2003).

To estimate the latencies of the classification/regression algorithms for parameter values (e.g., number of features or support vectors) beyond the investigated values, the computations that underlie each algorithm were analyzed and for each algorithm a model was derived that relates the latencies to the number of features and support vectors (cf. Section 6.3.4.3 for details).

decoder	coefficient	Lower CI	Value	Upper CI
LF	*a (µs)*	*44.22*	*44.26*	*44.30*
	b (µs/feature)	*0.00376*	*0.00382*	*0.00388*
LDA	*a (µs)*	*44.92*	*44.96*	*45.01*
	b (µs/feature)	*0.00478*	*0.00484*	*0.00490*
KF	*a (µs)*	*236.82*	*238.74*	*240.66*
	b (µs/feature)	*0.0009230*	*0.009231*	*0.0009233*
SVR	*a (µs)*	*12.03*	*12.36*	*12.69*
	b (µs/feature)	*0.0038148*	*0.0038156*	*0.0038165*
SVM	*a (µs)*	*12.66*	*12.98*	*13.31*
	b (µs/feature)	*0.0037928*	*0.0037936*	*0.0037945*

Table 2: Parameters of latency models for decoding algorithms

Fitted parameter values of the models describing the latencies of different classification/regression algorithms together with their lower and upper 5% confidence interval (CI) bounds. Table from Fischer *et al.* (2014).

The free model parameters were fitted to the measured latencies (Table 2, Figure 19, Supplementary Figure iv). These models predicted the maximum number of features that could be handled within a given maximally allowed time by a tested decoding algorithm running in a single thread. Assuming that the maximal latency of the overall filter pipeline (feature extraction and decoding) should not exceed 32ms for a smooth control, 10ms (about one third) were allowed for the decoding algorithm and the remaining 22ms were allotted for the feature extraction, other computations and as a safe-guard against stalls. Under these conditions, one can use up to $5.7 \cdot 10^6$, $4.5 \cdot 10^6$ and 286 features for LF, LDA and KF respectively, and 57625 and 57958 features for SVR and SVM, if 100 support vectors are used for decoding. This further corroborated the finding that, in most currently realistic scenarios all decoding algorithms except the Kalman filter did not require parallelization. If the Kalman filter was used with hundreds of features, parallelization of the KF algorithm would increasingly become necessary to ensure smooth and instantaneous BCI control.

Summary

Latency and CPU load can be reduced significantly when using an appropriate waiting strategy. The waiting strategy $Wait_{event}$ might be the preferred choice for many applications as it combines low latencies with low CPU usage and robustness against stalls. For realistic parameters the considered feature extraction algorithm can already be too demanding for an application without using independent substream parallelization. All considered decoding algorithms except the Kalman filter did not require parallelization.

7.2 Closed-Loop BCI Study

The Freiburg BCI Software was deployed in a closed-loop BCI study. Two different configurations were used for training and brain-control sessions respectively. A configuration consisted of a set of processing objects and the connections between processing objects. Brain-control configuration based on the training configuration but was extended by additional filter algorithms for feature extraction and decoding as well as the necessary additional connections. For three subjects a Brainbox amplifier was used, for the remaining two subjects a Neuvo amplifier was used. Experimenters exchanged the Brainbox and Neuvo amplifier processing object according to the amplifier hardware

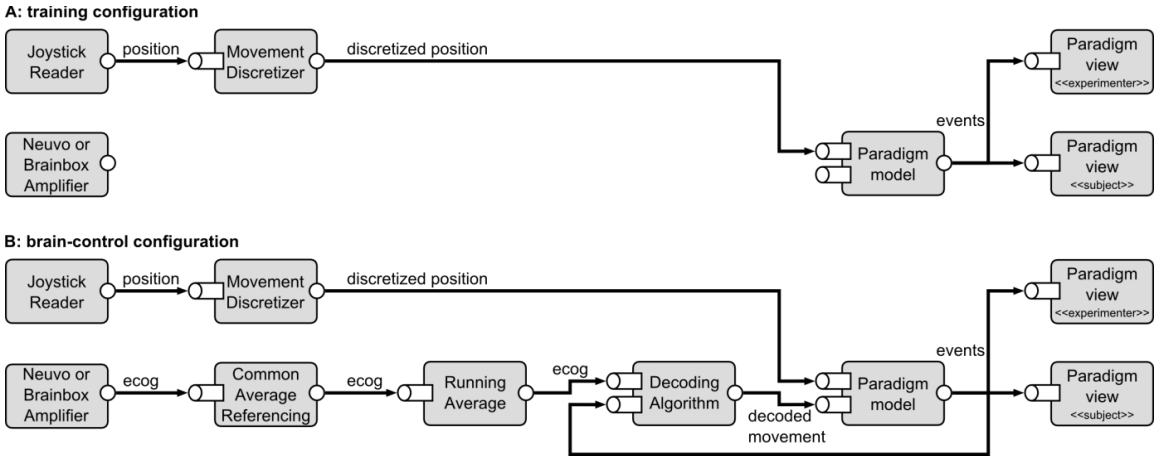


Figure 22: Configurations for closed-loop BCI study

Black arrows denote connectors between output ports (circles) and input ports (circles with rectangular ending). Type of data which is being transferred by a connector is written besides it. A: configuration for training sessions. B: configuration for brain-control sessions.

that was actually used. For both configurations ECoG measurements were recorded to hard disk for later offline analysis.

Training session configuration is shown in Figure 22 A. A joystick reader read subject wrist movements which were then discretized and sent to the paradigm model. There were two screens, one for the subject and one for the experimenter. Two paradigm views, one for the subject and one with additional information for the experimenter (i.e., in experimenter mode) displayed the paradigm model on screen.

Feedback session configuration (Figure 22 B) was based on the training session configuration but added a filter pipeline for closed-loop decoding. Measurements were preprocessed by a common average algorithm. As the Neuvo amplifier did not have a high pass filter, in contrast to Brainbox amplifier, an additional running average algorithm removed offsets when measuring with a Neuvo amplifier. The wrist position was decoded from preprocessed measurements and forwarded to the paradigm model. The paradigm model also used discretized joystick positions, but only to detect erroneous trials (i.e., if actual movement deviated from instructed movement). Over the paradigm model's output port, a backward connection to the decoding algorithm triggered decoding after movement phase.

Training and feedback configurations included connections where two data streams converged into one processing object, one data stream was distributed to two processing objects and one backward connection (Figure 22 B; from paradigm model to decoder). This confirmed in practice that the domain model can map different experimental settings by exchanging, adding and removing processing objects and the connections between them. It furthermore corroborated the assumption that the domain model was general enough to map a wide range of experimental settings.

Summary

Significant online decoding of movement direction was achieved in four out of five subjects with an 75% average of correct trials (Milekovic *et al.*, 2012). Seen as an acceptance test, this successful de-

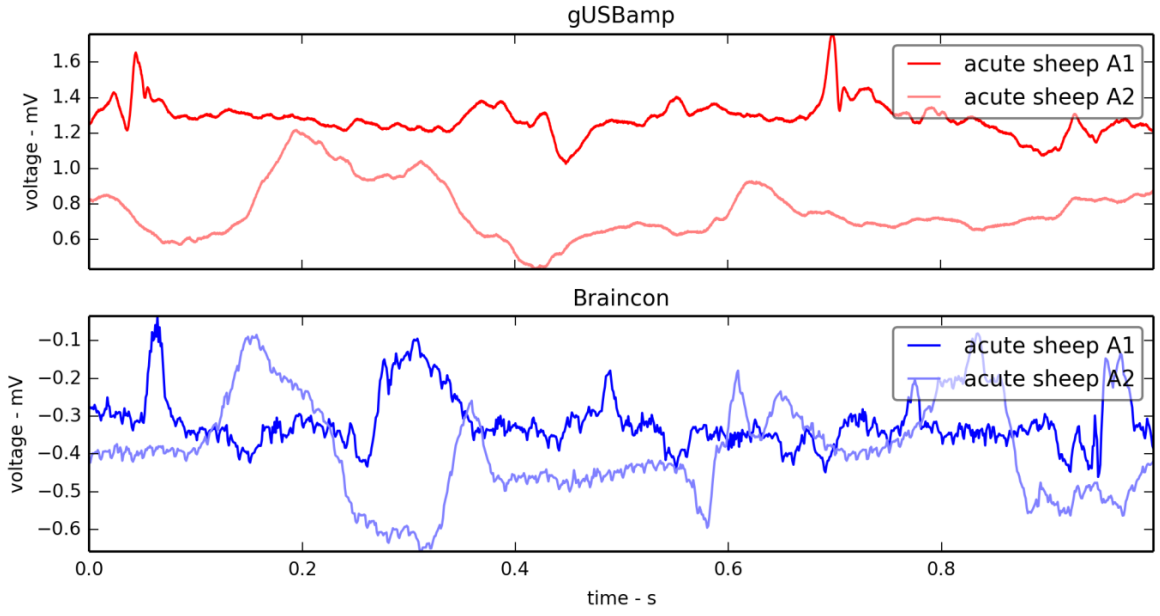


Figure 23: Exemplary ECoG measurements

Measurements recorded by gUSBamp and Braincon implant in acute sheep A1 and A2. Figure from Kohler and Fischer.

ployment of the software in a closed-loop BCI study demonstrated the flexibility of the proposed software architecture as well as the closed-loop capability of the Freiburg BCI Software in its entirety.

7.3 Preclinical *in vivo* Animal Study

7.3.1 Acute Setting

The goal of the acute setting was to compare the measurement functionality of the Braincon implant to an established ECoG amplifier. Therefore two sheep (A1 and A2) were implanted with an experimental setup that allowed ECoG recording with different amplifiers but from the same brain area (cf. Section 6.5.2).

Both amplifiers (i.e., Braincon implant and gUSBamp) recorded measurements from A1 and A2 at rest condition. During rest condition no electrical stimulation was applied. Figure 23 shows one second of exemplary measurements from these recordings in the time domain. All measurements were from the same electrode grid using the same physical electrode contact (channel 9 in Figure 18 A and channel 1 in Figure 18 B, respectively). Regarding amplitude fidelity all recordings were similar. The signals recorded with Braincon looked rougher, because an artifact overlaid the recorded signal. The artifact was caused by the transmission of data from the implant to the external unit: a block of data was sent every ten milliseconds which consumes additional energy. As the energy was transmitted in a constant manner, supply voltage of the implant dropped and caused a bias with a periodicity of 100Hz in the measured values. Therefore this effect was termed '100Hz artifact'. In addition, a lead fracture occurred in one measurement channel of the implant (channel 11 in A1, channel 3 in A2), this channel was therefore excluded from this analysis.

For the Braincon implant, the mean power spectral densities (PSD) over 100s of recordings at rest condition were calculated for each channel. For each acute implantation, the PSDs of all single channels recorded by Braincon were similar and the same was true for the gUSBamp amplifier, see Sup-

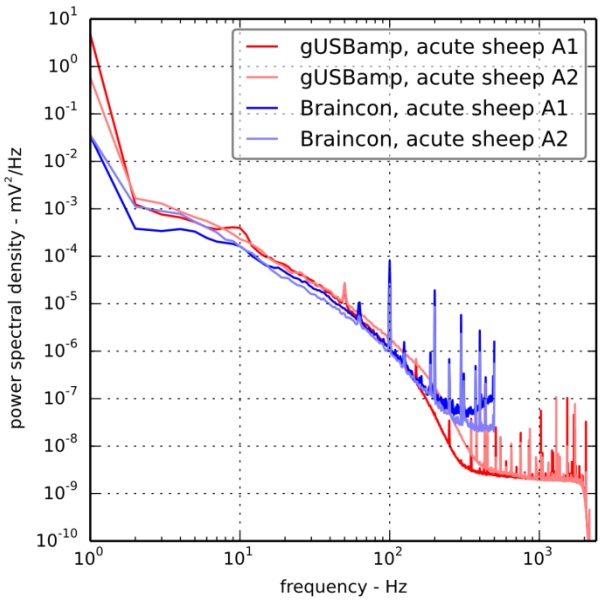


Figure 24: Mean power spectral density over all channels

Mean power spectral density of gUSBamp and Braincon for acute sheep A1 and A2. Power spectral densities were calculated from 100 seconds of recordings while sheep were anesthetized. For the Braincon implant, one channel was excluded due to lead fracture, therefore mean was calculated over 15 channels. For gUSBamp, mean was calculated over 16 channels. Figure from Kohler and Fischer.

plementary Figure v. It is therefore sufficient to report only the mean PSDs over all channels. For both sheep A1 and A2, Braincon's mean PSDs were similar with a decrease of power for increasing frequency until ~300Hz when the noise floor was reached (Figure 24). Above 300Hz, Braincon's mean power stayed roughly constant for A2, but increased for A1. As this increase was only visible in A2 but never in A1 and C1 (cf. Figure 27), this was likely to be a temporary effect (e.g., bad contacts between electrode and brain). The 100Hz artifact and its harmonics clearly dominated frequency bands around 100Hz, 200Hz, 300Hz and 400Hz. Braincon's PSDs were comparable to gUSBamp spectra in the lower frequencies up to ~200Hz. The gUSBamp reached its noise floor around 400Hz.

The body-external ML180 stimulator applied electrical stimulation to the sheep's nose while either gUSBamp amplifier or Braincon implant recorded sheep's ECoG. The EPA processing object analyzed these measurements for somatosensory evoked potentials (SEPs), cf. Section 7.3.1.

Figure 25 shows exemplary median of evoked potentials from A1 when the lower left lip was stimulated with 4mA. The 100Hz artifact and its harmonics were removed with a third order Butterworth band-stop filter (cut-off frequencies at [f-10Hz, f+10Hz] for f=100Hz, 200Hz, 300Hz and 400Hz). The amplitudes of the evoked potential varied depending on the channel, whereas channel 16 showed the most pronounced evoked potential. For each channel the evoked potentials recorded by Braincon and gUSBamp amplifier were similar in respect to form, amplitude and onset. This similarity was found for different stimulation sites and also with sheep A2. It was beyond the scope of this work to provide a neurobiological interpretation for the evoked potentials (e.g., the relation between stimulation parameters, evoked potentials and brain topology). This aspect is provided by Gierthmuehlen *et al.* (2014).

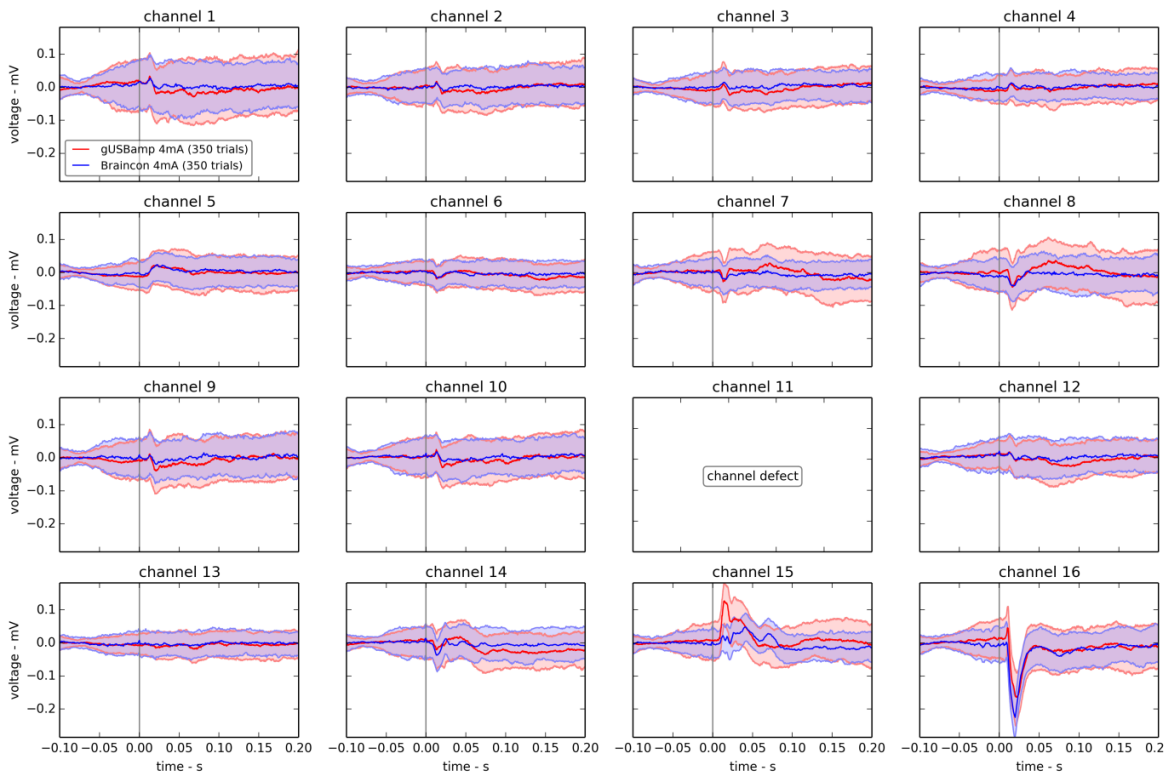


Figure 25: Evoked potentials from acute sheep A1

Stimulation site was at the lower lip with 4mA, stimulation pulse occurred at t = 0. Red line shows median over 350 stimulation events for gUSBamp, blue line shows median over 350 stimulation events for Braincon implant. Light red and light blue areas show interquartile range for gUSBamp and Braincon, respectively. Subplot position corresponds to the electrode contact position in the electrode grid, cf. Figure 18 A. Figure from Kohler and Fischer.

To sum up the results, the recorded signals from Braincon and the gUSBamp were similar in time- and frequency-domain for both acute sheep for up to 200Hz. However, the 100Hz artifact dominated the frequency bands around 100Hz and its harmonics. The recorded SEPs were similar regarding form and amplitude for different stimulation sites and for both sheep. This provided strong evidence for the assumption that the measurements yielded by the Braincon implant were technically comparable to a commercially available, non-implantable amplifier. This was a necessary pre-requisite for the interpretation of the signals recorded from Braincon in the chronic setting.

7.3.2 Chronic Setting

One sheep (C1) was chronically implanted to technically assess Braincon's measurement and stimulation functionality in the chronic setting.

C1 was anesthetized for multiple measurement sessions at days 29, 114, 281 and 308 after implantation. The positions of implant and electrode grid were verified by x-ray at days 0, 114 and 308 after implantation. At day 308, x-ray showed a dislocation and folding of the electrode grid. Temperature within implant housing was monitored by the built-in temperature sensor and was mostly within the typical range for sheep body temperature (Lorenz, 2006). Only during measurements under anesthesia, temperature was higher due to the permanent operation of the implant during this session. However, temperature was always below the limit of 2K above average body temperature as required by EN 45502-1 (European Committee for Electrotechnical Standardization, 1997). Humidity within the implant housing was very low (i.e., below detection threshold) until day 281. Negative humidity values

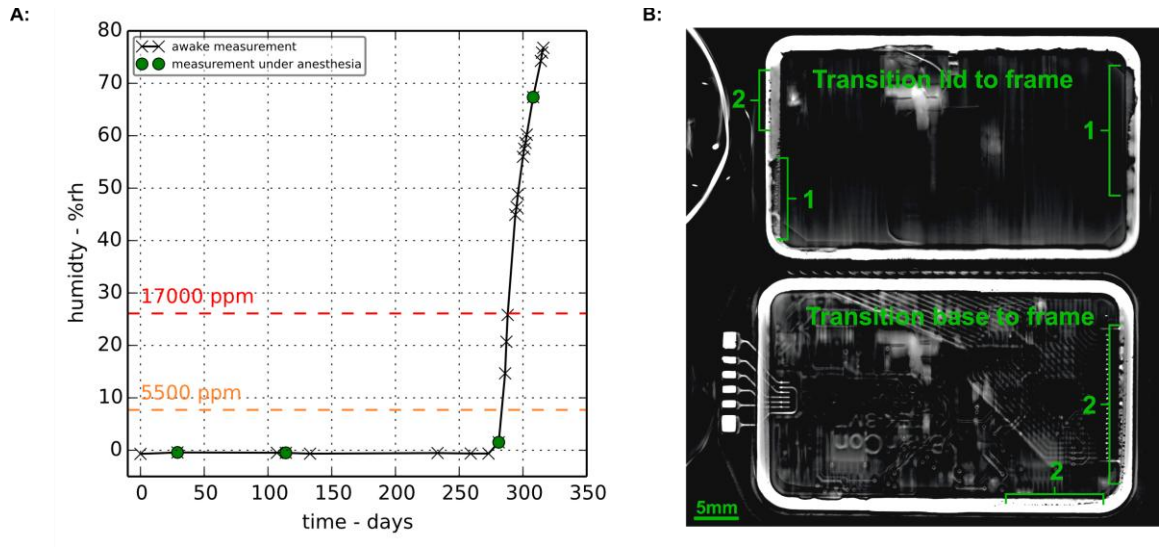


Figure 26: Humidity over time and µCT of Braincon implant housing

A: Humidity values measured within C1 implant housing over time after implantation. Green dots mark measurements under anesthesia, black crosses indicate measurements while C1 was awake. Black line is linear interpolation between measurements. Orange line at 7.7%rh, corresponding to 5500ppm water, marks critical threshold where water could condensate within the housing (Jiang and Zhou, 2010). Red line at 26.1%rh, corresponding to 17000ppm, marks critical threshold for ongoing corrosion mechanisms between metal tracks (Thomas, 1976). For calculation of both thresholds, average sheep body temperature of 39.5°C was assumed (Lorenz, 2006). B: The top image shows the transition between the soldered ribbon and the screen-printed metal frame on the lid. The bottom image illustrates the transition between base and ribbon. Defective sites are distinguished between layer delamination (1) and voids (2) within the material. Figure from Kohler and Fischer.

were attributable to the sensor's measurement tolerance in this range (i.e., ±2 %). Then humidity increased rapidly. The implant stopped transmission after 316 days, probably due to humidity (Figure 26 A). Explantation revealed that a cyst has grown between electrode grid and brain surface. Computer tomography analysis of implant housing after explantation revealed two drawbacks of the sealing and assembly process (Figure 26 B). During this process, first a rectangular metal frame is soldered onto the implant's base and then a ceramic lid is soldered onto the top of the metal frame, cf. Figure 6. Please see Kohler *et al.* for details about this process. Reduced surface areas of the screen-printed frames on top of the implant's base (Figure 26 B (1)) as well as voids within the solder bulk (Figure 26 B (2)) were identified. Especially the first defect easily leads to possible moisture pathways which could cause eventual implant failure.

Figure 27 shows the mean PSDs over all channels for C1 from the measurements in rest condition at 29, 114, 281 and 308 days after implantation. Mean PSDs from acute setting are shown in gray for comparison. Five channels yielded no signal, probably due to lead fracture, and were thus excluded from further analysis. Compared to acute PSDs from A1 and A2 (gray lines), the power in the chronic spectra was lower but constant up to 114 days for all frequencies (blue lines). Power for frequencies below 100Hz decreased to a minimum at 281 days and increased again at 308 days (green lines). Noise floor was reached at ~300Hz on days 29, 114, 308. However, on day 281 the noise floor was already reached at ~100Hz. As humidity was at day 281 still below any critical threshold with 1.52%rh, an electronics malfunction could be excluded. Therefore possible causes for these differences between the first and the last two measurements could be the growing of the cyst between electrode grid and brain surface or the folding of the electrode grid.

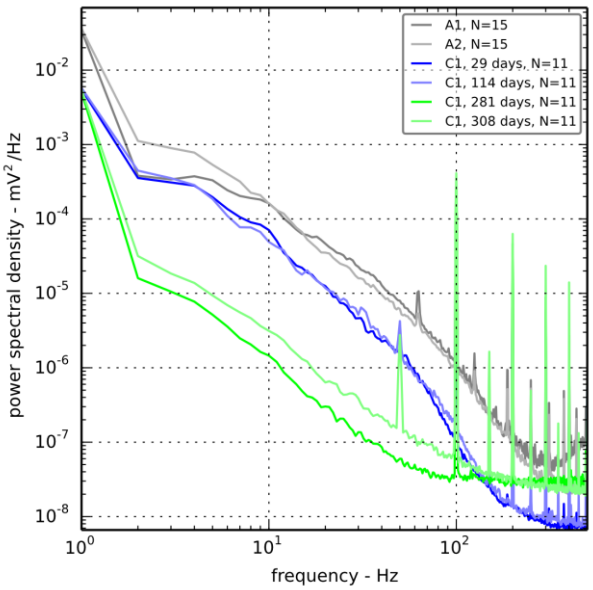


Figure 27: Mean power spectral densities for C1 at different times after implantation

Blue and green lines show mean PSDs from C1 measured at different times after implantation (29days, 114 days, 281 days and 308 day). Gray lines depict mean PSDs of Braincon from A1 and A2. For each PSD, N denotes the number of channels used for calculation of mean. Figure from Kohler and Fischer.

As in the acute setting, different facial areas of C1 were stimulated with the body-external ML180 stimulator. Stimulation current varied between 2mA, 4mA and 8mA. The Braincon implant together with the Freiburg BCI Software measured and analyzed the resulting SEPs. As expected, and consistent with the experiences from the acute setting, not all channels responded equally well to stimulation (i.e., not all channels exhibited SEPs). Figure 28 exemplarily shows median SEPs recorded on day 29 at channels 7 and 11. These channels exhibited SEPs when the sheep's lower or upper lips were stimulated. As in the previous analysis (cf. Section 7.3.1), the 100Hz artifact and its harmonics were removed. For both channels and both stimulation sites, amplitudes of SEPs increased with increasing stimulation currents. However, channel 7 showed higher SEP amplitudes for upper lip stimulation than for lower lip stimulation. Channel 11 exhibited an opposite behavior. The two channels considered here were located adjacent on the foil electrode. This demonstrated that Braincon could record with a spatial resolution of 4mm, the center-to-center distance of the electrode contacts used in this study.

To assess the stimulation functionality of Braincon, the implant applied voltage stimulation to each channel. Stimulation waveform was a rectangular pulse with a pulse width of 1ms with 5Hz repetition frequency. Amplitude was either 1V, 4V, 7V, 10V and 13V. All measurement channels showed a response to stimulation. This observation was expected because the measurement range of the implant was in the range of millivolts and the stimulation amplitudes were three orders of magnitude higher. Figure 29 shows three exemplary responses from a measurement under anesthesia at day 281. There were always sharp peaks at the time t = 0, most likely caused by the 1ms stimulation pulse, but after this peak, the form of the response varied for each measurement channel (Figure 29 A-C). A removal of the 100Hz artifact was not done for this figure because this would also remove necessary frequency components from these sharp peaks and thus falsify their form too much. For some channels the response amplitude scaled with the stimulation response (e.g., Figure 29 A). For other channels the response amplitudes first increased, then decreased (e.g., Figure 29 C). Taking into account that signal quality at day 281 was the worst compared to other recording sessions (cf. Figure 27) and that the foil

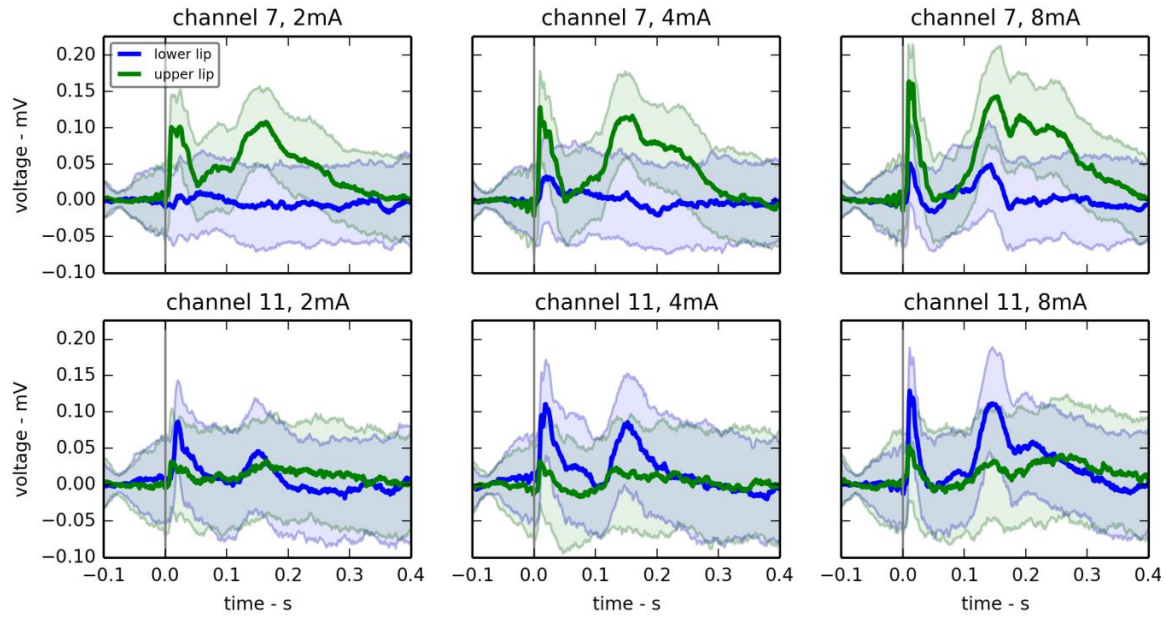


Figure 28: SEPs from C1 for different stimulation sites and intensities

Examples of SEPs from two adjacent measurement channels (channels 7 and 11, cf. Figure 18 C). Each subplot shows the stimulation response measured on one channel for one stimulation intensity. Stimulation intensity varied between 2mA, 4mA and 8mA. Blue line is median SEP response over 360 trials when stimulation is applied to the sheep's lower lip. Light blue area mark the 25 and 75 percentile. Similary, green lines shows median SEP and percentiles for upper lip stimulation. Figure from Kohler and Fischer.

electrode was dislocated, it is likely that the measured responses were volume conduction effects. Still, these results demonstrated that Braincon could, from technical perspective, perform *in vivo* stimulation.

Summary

This study confirmed the Braincon implant's in vivo technical measurement capability for up to 316 days and provided an upper bound estimation on the minimal spatial resolution achievable with Braincon. First evidence was provided that Braincon can perform in vivo stimulation. In addition, the proposed software architecture was successfully applied to a study involving brain mapping, providing more evidence in favor of the claim that the proposed architecture is applicable to a wide range of BCI applications.

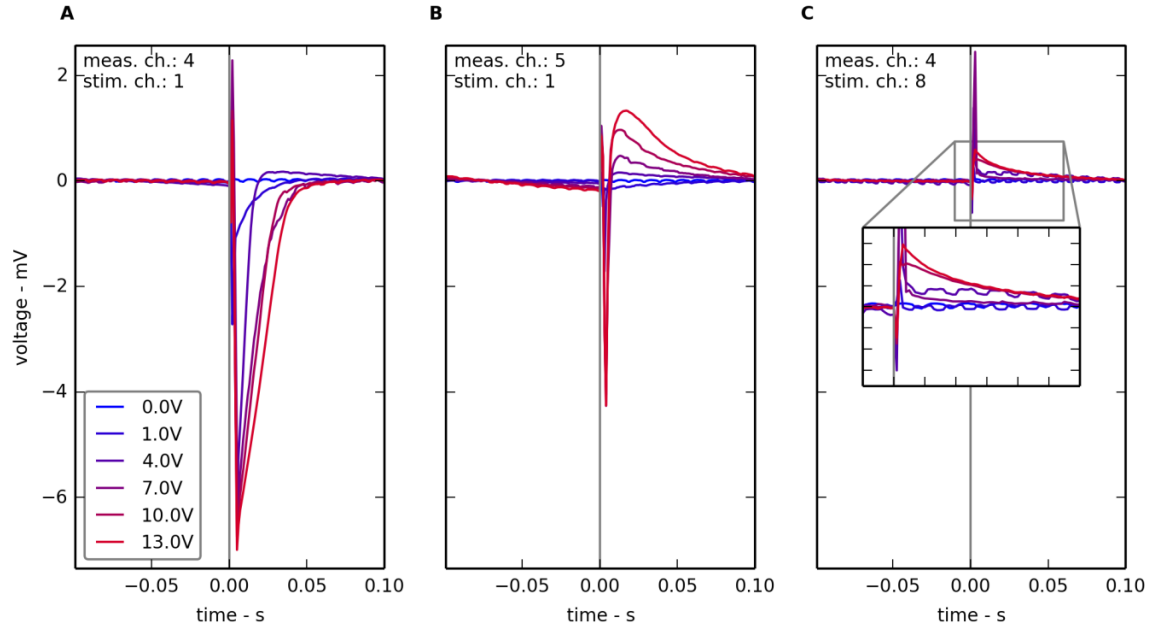


Figure 29 Exemplary responses to stimulation with Braincon implant

Each subplot shows responses to stimulations with varying amplitudes (0V, 1V, 4V, 7V, 10V and 13V). Stimulation occurred at time t = 0. A: Responses from measurement channel 4 while stimulating on stimulation channel 8. B: Responses from measurement channel 5 while stimulating on stimulation channel 1. C: Responses from measurement channel 1 while stimulating on stimulation channel 8. Figure from Kohler and Fischer.

8 Discussion

8.1 Platform Software from the Regulatory Perspective

Braincon is a platform system for closed-loop measurement and electrical stimulation of brain areas intended for chronic implantation in human patients. The AIMD defines a medical device, in essence, as a device for the purpose of diagnosis, treatment or alleviation of disease, injury or handicap (Council of the European Community, 1990; Article 1(2) a in the latest version from 2007). According to this definition, the Braincon Platform is not a medical device, because there is no specific medical indication to treat, diagnose or alleviate a disease, injury or handicap defined for it. In addition, the risk management according to EN ISO 14971 requires the medical indication and intended use as input for risk analysis (European Committee for Electrotechnical Standardization, 2012a, clause 4.2). The development of the platform in compliance with the regulatory requirements of medical device development can therefore not be done completely. However, risk management and usability studies can be done for use cases which all potential medical indications of Braincon have in common: implantation, measurement, stimulation, impedance measurement and explantation. Consider now the case that the Braincon Platform should be used for a new specific medical indication. From the regulatory perspective, one has to start a completely new development of a medical device. But in practice, most of the existing development documentation, especially risk management and usability studies, can probably be re-used. However, a gap analysis has to be conducted to check if identified risks, risk control measures and use cases still adequately cover this specific medical indication as well as to identify not yet considered risks and use cases. This should accelerate the development process and reduce development time and effort to commence a clinical study, but the acceleration depends on how much can be re-used from the Braincon Platform. As the functionality of the implant hardware is rather generic (i.e., measure ECoG signals and perform cortical stimulation) the adaptation of the Braincon Platform to different medical indications will most likely involve mainly changes to the Braincon Platform Software. This further corroborates the necessity of the Braincon Platform Software for a modular software architecture with a general domain model.

The way for a BCI system towards certification of as a medical device is usually divided in three phases: In the research phase a proof-of-concept is developed (e.g., in an animal model or non-invasively with healthy subjects in a setting comparable to the aspired clinical setting). In the second phase the AIMD allows patient-specific studies with a ‘custom-made device’ designed specifically for one particular patient. The third phase consists of one or more clinical studies with multiple patients to assess the BCI's efficacy and safety in a systematic manner. Using the Braincon Platform Software already in the research phase offers several advantages: one can re-use existing and already extensively tested components, yielding reliable results. Results from early preclinical experiments are also better comparable to later results, as same the software was used. Using the same software reduces the programmer's effort and learning curve for switching from a non-medical software to a medical software. When progressing from research study over patient-specific study to clinical study, the experimental paradigm and thus the finally needed software components become clearer so that the used non-medical components can be subsequently replaced with medical components. Waiting times due to development of medical software are less likely, as the transition from non-medical to full medical software can be performed smoothly over time.

However, there is one caveat when mixing non-medical and medical components. As the whole software development has to be conducted according to the AIMD's requirements (Council of the

European Community, 1990 Annex 1 Article 9; in the latest revision from 2007), a BCI platform configured to use medical and non-medical components cannot be certified. A similar situation occurs on the level of source code: if non-medical source code is incorporated into a medical source code base in an untraceable manner, one cannot verify that the whole software was developed according to the AIMD's requirements. Therefore, when using the Braincon Platform Software for a 'soft transition' towards medical software, it is crucial to establish an effective segregation between medical and non-medical source. In addition, the effectiveness of this segregation has to be argued to the certifying notified body. One possible solution could be to encapsulate all medical and non-medical components in plug-ins which can be loaded at runtime dynamically. On the Windows operating system, each plug-in would reside in a separate dynamic loadable library (DLL). A plug-in within a DLL does not require linkage to the main application at compile time and therefore the plug-in's source code can be kept completely separated from the medical source code of the main application (e.g., in different source code repositories) and thus provides a plausible segregation between non-medical and medical source code. The Braincon Platform Software uses such a plug-in loading mechanism. One further improvement could be to implement facilities to reliably distinguish medical and non-medical plug-ins, to protect against accidental loading of non-medical plug-ins (e.g., by using cryptographic signatures). In summary, the Braincon Software Platform with its modular plug-in loading facility supports a smooth transition from the research phase to the clinical phase.

8.2 Filter Pipeline

In this thesis, a software architecture for BCI applications was presented and its performance evaluated. The modular design of the proposed architecture makes it easy to parallelize typical processing steps of BCI applications by making use of multi-core computer processors and multi-processor configurations, which are nowadays available inside standard desktop and laptop computers. The proposed domain model makes the integration of any type of processing algorithm simple. Algorithms are housed inside a filter implementation, general and reusable elements of the architecture that enable immediate use of the parallelized data processing. It was shown that, by using the proposed software architecture, it is simple to parallelize many time-consuming parts of the neuronal data processing in typical BCI applications. Results demonstrate that using multiple threads in BCI signal processing leads to a substantial reduction of computing time required for one decoding step, further corroborating the findings of Wilson and Williams, 2009. Wilson et al. evaluated the effects of multi-threading on computation time for two feature extraction algorithms, in one case *using independent substream parallelization. Here, different implementations of multi-threading for BCI systems were examined (i.e., different waiting strategies) and it was shown that the choice of the waiting strategy is crucial for CPU load and latency reduction. This reduction in latency determines how fast a BCI reacts to the user's input which can directly affect the BCI performance: lower latencies can lead to improved performance while higher latencies can lead to deterioration of performance (Cunningham et al., 2011). In addition, BCI users may experience a control latency themselves which might reduce their motivation and convenience. In contrast, the experience of a quickly responsive, smooth control may motivate users. Moreover, by reducing the latency, the remaining time allotted for one processing step can be used to process recordings from additional channels, more signal features, signals recorded at higher sampling frequencies and to decode using computationally more complex algorithms. All of these possibilities can lead to an improvement in performance (Bansal et al., 2012; Milekovic et al., 2013) and might therefore substantially increase the user's convenience with the BCI system. The findings presented here also show that for a realistic number of channels and a realistic*

sampling frequency the BCI signal processing task can already be too demanding for a standard desktop computer if no parallelization is used. Hence, without parallelization the delay of the BCI feedback would need to be increased, resulting in a less smooth and delayed control.

By using the proposed software architecture, one can easily split the time consuming parts of the processing into separate threads: This is particularly straightforward for feature extraction algorithms, which can often be applied to each channel independently (e.g., low-/band-/high-pass filter, Fourier transform). In this case, each channel or different groups of channels could be contained in separate filters and run in separate threads. The results presented here show that among all waiting strategies Polling and $Wait_0$ provide the best overall scaling of latency reduction with increasing number of threads. However, these two waiting strategies result in the highest CPU load, leaving fewer computational resources for additional applications to run in parallel. Moreover, the number of threads has to be chosen carefully for these strategies as the BCI application can stall if the number of threads exceeds the number of available cores. In contrast, the waiting strategy $Wait_{event}$ does not stall and performance still increases even if the number of threads is higher than the number of cores. Therefore, fine tuning of the number of threads to the hardware specifications of the computing system is not necessary for $Wait_{event}$ (at least for up to twice the number of threads in relation to the number of cores). In addition, $Wait_{event}$ achieves the highest latency reduction among all waiting strategies. While the scaling of the latency with numbers of threads is slightly worse compared to Polling and $Wait_0$ this maximal latency reduction can be obtained by using more threads than cores. A further advantage of $Wait_{event}$ is that the CPU load remains low even if high numbers of threads are used. The waiting strategy $Wait_{event}$ might therefore be the preferred choice for many applications as it combines low latencies with low CPU usage and robustness against stalls.

In this thesis, BCI performance was improved by applying different waiting strategies to multi-threaded neuronal signal processing and decoding. The proposed software architecture is also useful for improving BCI performance as follows. (1) For multi-DOF regression multiple LFs and SVRs can be used. This increases the computation time approximately by a factor of the number of DOFs. However, as the computations of these separate LFs and SVRs are independent for each dimension, they can be parallelized in a straightforward way by using the proposed independent substream parallelization with each thread computing the LF/SVR for one dimension. Similarly, multi-class classification with C classes using SVM can be handled by C independent one-vs.-rest SVMs or by $C \cdot (C - 1) / 2$ pairwise SVMs which are independent as well. Multi-class LDA requires the computation of $C - 1$ decision functions, all of which can be calculated independently. Therefore, LF and SVM algorithms can also be efficiently parallelized using the proposed software architecture. (2) A boosting approach (Hastie et al., 2009) where many eventually weak decoders are combined into one stronger decoder promises to be suitable for BCI scenarios with noisy signals. As the individual decoders process the signals independently, the boosting approach can also be parallelized directly with the software architecture proposed here. (3) Another scenario might be an adaptive decoder (e.g., Shpigelman et al., 2009) where time-consuming adaptation operations are run in one or more background threads in addition to the thread running the decoder. (4) Neuronal signal decoding can also be temporally independent (i.e., separate and independent computations have to applied to the signals at different points in time) and these computations could be carried out by separate threads, see Wilson and Williams (2009).

It is possible to further parallelize the BCI application beyond the parallelization approaches presented here. For example, the processing on the level of the algorithm itself could be parallelized (e.g., by

using OpenMP (2014)). Such approaches are more complex than the solutions proposed here and require custom made solutions for each filter. Further optimization could also be achieved by integrating computations on graphics processing units into the proposed software architecture which provide fast and optimized algorithms for matrix and vector operations (Wilson and Williams, 2009). Some algorithms, for example the Kalman filter, cannot benefit directly from independent substream parallelization as their most time-consuming computations are not independent from the substream. For these algorithms, custom solutions have to be implemented, such as using parallelized code for the required matrix/vector operations (e.g., Cholesky factorization as described by Santos and Chu (2003)).

8.3 Distributed System

The aforementioned approaches for parallelization all involved parallelization within one process. One can also use multiple processes for parallelization, as for example BCI2000, the Berlin BCI and BCI++ do (Schalk *et al.*, 2004; Krepki *et al.*, 2003; Perego *et al.*, 2009). Transferring data between two processes or two computers requires additional complexity and development effort. Prior to sending data to the other process, all transferred data has to be serialized and after transfer, received data has to be deserialized. It might even be necessary to account for different binary data formats between two computers (e.g., little and big endian data format). Also, the transfer could introduce a latency jitter, especially when using transfer over network, which might be difficult to control. But splitting a BCI among multiple computers also has its benefits. One can increase the available computational power with additional computers[9]. Threads within one process share the same memory range, making it possible that one thread accidently modifies memory used by another thread (e.g., due to a programming error). In contrast to threads, different processes use different memory ranges, so that accidental modification of memory is prevented by the operating system. Consider a medical software system consisting of software items from all safety classes (A, B and C; see EN 62304, European Committee for Electrotechnical Standardization, 2006). This software system would have the maximum safety class of all its constituent software items (i.e., safety class C). One would have to apply more of the laborious development and testing processes required for safety class C during implementation of all software items. Alternatively, one could split this software system into two systems, one system with safety class C items and one for the remaining class A and B items, each running on different computers. This would reduce the required development effort because from the regulatory and risk management perspective, this division is an effective segregation (EN 62304 Clause 5.3.5, European Committee for Electrotechnical Standardization, 2006) of software systems: there is no sharing of resources (CPU, memory) between them. The Braincon Platform Software could be leveraged to a multi-process application, if a processing object was added that serialized outgoing IData objects and transmitted them to another computer running another instance of the Braincon Platform software or even another software system (e.g., over network). Similarly, incoming data would be deserialized into IData objects for further processing.

8.4 Soft Real-Time versus Hard Real-Time

A software with hard real-time capability guarantees to respond within a certain time limit. In contrast to this, soft real-time software guarantees only to respond in the mean within a certain time limit (Kopetz, 2011). Closed-loop BCI software is clearly subject to real-time requirements. However,

[9] However, the communication overhead will increase with each additional computer. This makes adding more computers unfeasible at some point.

whether soft or hard real-time is required and the time limit depends on the concrete use case. There was the question whether the Braincon Platform Software should provide hard real-time capabilities. As it was intended as a platform software, no specific use case could be used for deriving real-time requirements. Instead, comparable BCI platform software systems were examined with respect to their real-time capabilities as guidance for this decision. BCI2000, the Berlin BCI and OpenVibe are capable of soft real-time, whereas BioSig provides hard real-time capabilities (Schalk *et al.*, 2004; Krepki *et al.*, 2003; Renard *et al.*, 2010; Schlögl *et al.*, 2007). The NeuroRighter platform for closed-loop experiments on the level of single neurons provides only soft real-time, which according to Newman *et al.* (2013), is sufficient for most of the authors' closed-loop experiment. In contrast, the system by Müller *et al.* (2013) is also intended for closed-loop experiments with single neurons but is designed for hard real-time capabilities in the sub-millisecond range. To summarize, from the application perspective, three out of four of the BCI platforms (BCI2000, OpenVibe, Berlin BCI) which are most similar to the Braincon Platform Software are designed for soft real-time. If recordings on the level of neurons were involved, hard real-time requirements were encountered more often, but were still not always required.

The BCI platforms considered previously were not intended as medical devices (i.e., for human patients). But for the Braincon Platform Software, safety considerations had to be taken into account when deciding on its real-time capability. It would clearly be beneficial for safety if one could implement safety-critical functionality with a hard real-time guarantee. Two safety-related features, F1 and F2, were identified as possible sources of hard real-time requirements: (F1) to guarantee that stimulation was started or stopped within a certain time limit and (F2) to guarantee that the nursing staff would be informed within a certain time limit that the patient needs assistance. It would be technically feasible to add a third-party library for enhancing the Braincon Platform Software with hard real-time capabilities (e.g., Kithara RealTime SuiteTM (Kithara Software GmbH, Berlin, Germany)). But the regulatory consequences of adding safety-critical features like the two outlined above would be the following: the software safety class of the software item for F1 is determined by the harm that could be caused if the software item malfunctioned. As one can never completely exclude a programming error, one cannot completely exclude that the software item fails to stop stimulation which results, in the worst case, in unwanted stimulation causing the patient's death. Therefore, this software item would have software safety class C. Similarly, the software item for F2 would have safety class C because one cannot exclude the case that the call for assistance fails. As a consequence, the whole software system and all its constituent software items, would have to be developed according to the laborious development requirements for safety class C. Therefore the decision was made to exclude such safety-critical features from the Braincon Platform Software. As electrical stimulation was an essential part of the Braincon system, at least F1 had to be addressed. Therefore F1 was realized as part of the firmware with a dead man switch that turns off stimulation automatically when no more stimulation commands are sent by the Braincon Platform Software, see Section 6.5.1 for details. The implant’s firmware has safety class C and is considerably smaller than the Braincon Platform Software. Therefore less validation effort was needed and the firmware can give real-time guarantees (e.g., by using interrupts). This design decision has one important consequence: in the case of a malfunction (e.g., connection between PC and implant is interrupted) the implant stops stimulation. This in turn restricts the medical indications for Braincon to indications where stimulation is not used for life-sustaining treatments.

Real-time capabilities also played a decisive role when choosing the programming language of the Braincon Platform Software. Managed languages (e.g., Java or C#) have excellent tool support for development and testing and come with an extensive standard library. They are therefore very appealing for the development of medical software. But these managed languages also have a garbage collector (Domani *et al.*, 2000). Garbage collection could hamper the soft real-time capabilities by infrequent freezes of the whole application, namely every time a garbage collection is executed. This resulted in additional temporal jitter (Azatchi *et al.*, 2003), even in the case where most of the garbage collection is done concurrently in the background. Therefore the decision was made to use C++ because there memory management could be fully controlled by the programmer.

To summarize, hard real-time capabilities for Braincon Platform Software were technically realizable. However, hard real-time was not implemented because from the application perspective, soft real-time capabilities were sufficient so far and from safety perspective it is more feasible to implement hard real-time safety features in a separate software system.

8.5 On-Implant- versus Body-External Signal Processing

The key element that makes a chronic implantable closed-loop BCI system a platform is its signal processing capability. The more different algorithms are provided, the more applications can be handled by the platform. Therefore, it is important where signal processing is actually implemented: either on the implant or not. If signal processing is located on the implant, there is clearly the advantage that processing is faster because the latencies involved with sending the data to an external processing device do not occur. One can probably give hard real-time guarantees, because it is likely that processing will be done on a microcontroller without operating system. However, all functionality implemented inside the implant is subject to size and energy constraints. The algorithms implemented on a microcontroller are limited by its small computational performance, compared to conventional PCs. Adding signal processing to the implant increases its energy consumption which in turn either reduces battery longevity or requires a more powerful percutaneous energy supply. Additionally, the heat generated by the implant increases. The issue of size and energy consumption can be alleviated by using ASICs for the most demanding algorithms regarding computational complexity and/or energy consumption, see Avestruz *et al.* (2008) for an example. However, the algorithms implemented in ASICs are less flexible than their software equivalents. In addition, the more algorithms are 'rendered into hardware', the more testing effort has to be invested for verification of the ASICs. If signal processing is located outside the implant (e.g., on a body-external device like a PC, laptop or embedded system), size and energy constraints are more relaxed for both the body-external device and the implant. The implant's functionality would be reduced to only measurement and stimulation. Removing signal processing capabilities from an implant could free up space, therefore the implant could either be smaller or use the free space for additional measurement and/or stimulation channels. On body-external devices, especially on PCs, computationally more demanding algorithms can be used. One can use classic BCI platform software with a large built-in library of algorithms (e.g., BCI2000 or OpenVibe). To the best knowledge of the author, there is currently no BCI platform known which provides such a library of algorithms for firmware. The system could profit from advances in computer technology (e.g., faster CPUs or more memory) by upgrading the body-external device without the need to exchange or modify the implant. Finally, the learning effort when programming and testing algorithms for classic BCI platforms is likely to be smaller, compared to pro-

gramming firmware, because the algorithms are often written in C++, Matlab or Python, where ample documentation and testing tools are available.

The NI system from Rouse *et al.* (2011) is an example for a BCI platform with on-implant signal processing. They based their implant on a cardiac pacemaker and added measurement and signal processing. The firmware of this implant can be flashed, even when already implanted and thus new algorithms for signal processing can be set, making the NI system suitable for multiple BCI applications and has thus to be considered a BCI platform. Deriving the system from a certified medical device is also an auspicious approach because much of the development effort (e.g., for hermetic housing, manufacturing, bio-compatibility validation etc.) has already been done and much of the required regulatory documentation has already been created. However, the trade-off for these features, compared the Braincon Platform, is a four times smaller number of measurement channels, fewer stimulation channels and limitations to the signal processing algorithms (Rouse *et al.*, 2011).

It is currently not clear whether to prefer on-implant signal processing or the body-external signal processing approach. In an early stage of a BCI study, the signal processing algorithms are less clear and thus a body-external signal processing with its higher flexibility regarding algorithms might be preferable to on-implant signal processing. In later stages, one can expect that the signal processing algorithms are better known and therefore an on-implant signal processing approach might be better suitable. Therefore it is likely that this question has to be decided separately for each BCI study.

8.6 Implant Housing

The built-in humidity sensor reliably delivered implant vitals for every measurement session. As soon as a humidity increase within the package was observable, the time until failure for the implant could be predicted. Hence, the humidity sensor concept works reliably as a diagnostic safety measure. Three reasons for the solder-seal being prone to defects such as illustrated in µCT scans (cf. Figure 26) are important to mention. Firstly, as the implant housing was assembled from three parts, it possessed two interfaces which had to be solder-sealed, providing an increased overall volume and more critical spots for defects. Secondly, the hand-crafted metal ribbon, bent to form the intermediate frame between screen-printed lid and base, was inherently not perfectly even. It hence left small gaps and crevices within the stack assembly which needed to be solder-filled. Again, this meant an increased likelihood for voids. Finally, the utilization of more solder material to form a regular shaped seal demanded increased soldering intervals. As the solder liquidus time was increased, side effects such as leaching occurred which in turn affected thickfilm adhesion and integrity.

8.7 Future Work

One topic that has to be addressed in future hardware-generations of the Braincon Platform is miniaturization. Clearly, the current size of the used PC or laptop for running the Braincon Platform Software is not optimal for everyday clinics or at home use. This can be alleviated, probably at the cost of computational power, by using micro-PCs or by porting the Braincon Platform Software to an embedded hardware. The implant size should also be reduced, as the implant size is likely to correlate with the risk for the patient. Therefore, there will be a transition from the currently used off-the-shelf electronic component towards ASICs. Such ASICs might also be used on the body-external device for signal processing (e.g., to provide complex algorithms in a fast and energy-efficient manner) in the future. However, then additional methods for verification of these ASICs have to be applied (e.g., by model checking (Miller *et al.*, 2011) or fault injection (Sauer *et al.*, 2011b)). For risk manage-

ment there are methods for identifying the critical parts of an ASIC (e.g., Sauer *et al.* (2011a)) that might be used to derive risk control measures.

One goal of the Braincon Platform is to provide an implant that can be implanted and used, in the optimal case, for the entire life time of a patient. The EN 62304 requires software developers to take into account maintenance over the software's whole life cycle, which can mean up to several decades for the Braincon Platform Software. During such a time period, one can expect that the manufacturers of computer hardware components for the Braincon system will sooner or later cease to produce these components. Likewise, at some point in time the operating system's vendor will stop to maintain it. In the worst case, this would mean a laborious and continuous maintenance effort to migrate the Braincon Platform Software to future operating systems and hardware platforms. The issue regarding the computer hardware components could be addressed by building up sufficiently large stocks of hardware components and operating system licenses. But then one could not profit from better future hardware. One solution could come from the domain of digital libraries: many libraries have an increasingly large amount of digital artifacts. Such artifacts encompass plain data, but also interactive data (e.g., digital art) and executable works (i.e., software) and all have to be kept accessible virtually forever. Like in the domain of medical software development, migration to no data formats and hardware platforms is laborious. Suchodoletz *et al.* (2013, 2012) propose to use emulation instead of migration: a big syndicate[10] with sufficient development resources should provide and maintain an emulator for future hardware platforms. Then there would be no need for migration of the digital artifact. The Braincon Platform Software, but also medical software with comparable maintenance requirements, could also use such an emulator to reduce migration effort.

8.8 Summary and Outlook

In this thesis, the Braincon Platform for chronic closed-loop measurement and electrical stimulation of brain areas in human patients is presented with focus on the Braincon Platform Software. The software's requirements were derived under the constraint of the juristic and normative regulations that apply for medical software. A BCI software architecture was defined to meet these requirements. The principles used for designing such a BCI software architecture, which satisfies the regulatory requirement of testability, but also has the desired performance properties necessary for BCI research, were given. As part of the architecture, a domain model was implemented which is flexible enough for a wide range of BCI applications. *Its filter pipeline substantially increases the computational power available for BCI signal processing while reducing latency. The architecture runs on standard desktop PCs and laptops and makes use of their multi-core/multiprocessor hardware. The proposed software architecture's modular design enables BCI researchers to quickly modify, extend and reuse existing algorithms, as well as to implement new algorithms for neuronal signal processing. The algorithms immediately benefit from parallelization without requiring the programmer to possess any knowledge about multi-threaded programming.* For the filter pipeline, the effects of waiting strategies on latency and CPU load were evaluated with respect to latency, CPU usage and different number of threads on typical feature extraction and decoding algorithms from the BCI domain. Results show that latency and CPU load can be reduced significantly when using an appropriate waiting strategy. They also show that for realistic parameters (i.e., number of channels and sampling fre-

[10] To the best of the author's knowledge, vendors of established virtualization software like VirtualBox (www.virtualbox.org) or VMWare (www.vmware.com) do not give the necessary guarantees to support old hardware platforms over decades.

quency) BCI signal processing can already be too demanding for an application without using multiple threads. The software architecture was employed in two research studies to show that its domain model is flexible enough to map different BCI settings but also as acceptance tests for the software architecture in its entirety. In the first study, wrist movement direction in five human subjects was decoded in a closed-loop manner. In a second animal study, the Braincon implant's measurement and stimulation capabilities were evaluated. Results show that measurements are comparable to a non-implantable, commercially available ECoG amplifier and verified its stimulation functionality *in vivo*.

The goal of the Braincon Platform Software is to provide researchers with the software means to evaluate the safety and efficacy of their BCI approach in research studies and clinical studies with human patients. This undertaking involves substantial development and testing effort under the regime of medical software development. Future works will have to include finishing the preclinical phase for one specific medical indication (e.g., stroke rehabilitation or pain therapy) and then conducting a successful clinical study. The author hopes that the Braincon Platform Software will support researchers in BCI research and thus foster the development of new closed-loop medical treatments.

9 References

Afshar, P., Khambhati, A., Stanslaski, S.R., Carlson, D., Jensen, R., Linde, D., Dani, S., Lazarewicz, M., Cong, P., Giftakis, J., Stypulkowski, P.H. and Denison, T.J. (2013), "A translational platform for prototyping closed-loop neuromodulation systems", *Frontiers in Neural Circuits*, Vol. 6, doi 10.3389/fncir.2012.00117.

Alexandrescu, A. (2001), *Modern C++ design: Generic programming and design patterns applied, C++ In-Depth Series*, Addison-Wesley, Boston, MA.

Allen, J.B. (1977), "Short term spectral analysis, synthesis, and modification by discrete Fourier transform", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 25 No. 3, pp. 235–238, doi 10.1109/TASSP.1977.1162950.

Amdahl, G.M. (1967), "Validity of the single processor approach to achieving large scale computing capabilities", in *Proceedings of the April 18-20, 1967, spring joint computer conference, Atlantic City, New Jersey, USA, April 18-20, 1967*, ACM, New York, NY, USA, pp. 483–485, doi 10.1145/1465482.1465560.

Andersen, G., Vestergaard, K., Ingeman-Nielsen, M. and Jensen, T.S. (1995), "Incidence of central post-stroke pain", *Pain*, Vol. 61 No. 2, pp. 187–193, doi 10.1016/0304-3959(94)00144-4.

Arle, J.E. and Shils, J.L. (Eds.) (2011), *Essential neuromodulation*, Academic Press, London, Burlington, MA.

Ativanichayaphong, T., He, J.W., Hagains, C.E., Peng, Y.B. and Chiao, J.-C. (2008), "A combined wireless neural stimulating and recording system for study of pain processing", *Journal of Neuroscience Methods*, Vol. 170 No. 1, pp. 25–34, doi 10.1016/j.jneumeth.2007.12.014.

Avestruz, A.-T., Santa, W., Carlson, D., Jensen, R., Stanslaski, S.R., Helfenstine, A. and Denison, T.J. (2008), "A 5 μW/Channel Spectral Analysis IC for Chronic Bidirectional Brain-Machine Interfaces", *IEEE Journal of Solid-State Circuits*, Vol. 43 No. 12, pp. 3006–3024, doi 10.1109/JSSC.2008.2006460.

Aydin, F. (2011), "Wireless closed-loop feedback systems for automatic detection and suppression of nociceptive signals", University of Texas, Arlington, 2011.

Azatchi, H., Levanoni, Y., Paz, H. and Petrank, E. (2003), "An on-the-fly mark and sweep garbage collector based on sliding views", *ACM SIGPLAN Notices*, Vol. 38 No. 11, pp. 269–281, doi 10.1145/949343.949329.

Ball, T., Kern, M., Mutschler, I., Aertsen, A. and Schulze-Bonhage, A. (2009), "Signal quality of simultaneously recorded invasive and non-invasive EEG", *NeuroImage*, Vol. 46 No. 3, pp. 708–716, doi 10.1016/j.neuroimage.2009.02.028.

Banala, S.K., Agrawal, S.K., Kim, S.H. and Scholz, J.P. (2010), "Novel Gait Adaptation and Neuromotor Training Results Using an Active Leg Exoskeleton", *IEEE/ASME Transactions on Mechatronics*, Vol. 15 No. 2, pp. 216–225, doi 10.1109/TMECH.2010.2041245.

Bansal, A.K., Truccolo, W., Vargas-Irwin, C.E. and Donoghue, J.P. (2012), "Decoding 3D reach and grasp from hybrid signals in motor and premotor cortices: spikes, multiunit activity, and local field potentials", *Journal of Neurophysiology*, Vol. 107 No. 5, pp. 1337–1355, doi 10.1152/jn.00781.2011.

Bauer, G., Gerstenbrand, F. and Rumpl, E. (1979), "Varieties of the locked-in syndrome", *Journal of Neurology*, Vol. 221 No. 2, pp. 77–91.

Beck, K. and Andres, C. (2005), *Extreme programming explained: Embrace change,* 2nd ed., Addison-Wesley, Boston, MA.

Bergmann, T.O., Mölle, M., Schmidt, M.A., Lindner, C., Marshall, L., Born, J. and Siebner, H.R. (2012), "EEG-guided transcranial magnetic stimulation reveals rapid shifts in motor cortical excitability during the human sleep slow oscillation", *Journal of Neuroscience*, Vol. 32 No. 1, pp. 243–253, doi 10.1523/JNEUROSCI.4792-11.2012.

Bilir, E., Faught, E., Kundu, S., Kuzniecky, R., Zeiger, E. and Morawetz, R. (1996), "Morbidity of epidural strip electrode implantation for epilepsy presurgical evaluation", *Journal of Epilepsy*, Vol. 9 No. 1, pp. 52–55, doi 10.1016/0896-6974(95)00057-7.

Birbaumer, N., Ghanayim, N., Hinterberger, T., Iversen, I., Kotchoubey, B., Kübler, A., Perelmouter, J., Taub, E. and Flor, H. (1999), "A spelling device for the paralysed", *Nature*, Vol. 398 No. 6725, pp. 297–298, doi 10.1038/18581.

Birbaumer, N., Kübler, A., Ghanayim, N., Hinterberger, T., Perelmouter, J., Kaiser, J., Iversen, I., Kotchoubey, B., Neumann, N. and Flor, H. (2000), "The thought translation device (TTD) for completely paralyzed patients", *IEEE Transactions on Rehabilitation Engineering*, Vol. 8 No. 2, pp. 190–193, doi 10.1109/86.847812.

Blakely, T., Miller, K.J., Zanos, S.P., Rao, R.P.N. and Ojemann, J.G. (2009), "Robust, long-term control of an electrocorticographic brain-computer interface with fixed parameters", *Neurosurgical Focus*, Vol. 27 No. 1, pp. E13, doi 10.3171/2009.4.FOCUS0977.

Blumberg, J., Rickert, J., Waldert, S., Schulze-Bonhage, A., Aertsen, A. and Mehring, C. (2007), "Adaptive classification for brain computer interfaces", in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2007), Lyon, France, August 22-26, 2007*, IEEE press, Piscataway, NJ, pp. 2536–2539, doi 10.1109/IEMBS.2007.4352845.

Boehm, B.W. (1981), *Software engineering economics, Prentice-Hall Advances in Computing Science and Technology Series*, Prentice-Hall, Inc, Englewood Cliffs, N.J.

Bolognini, N., Pascual-Leone, A. and Fregni, F. (2009), "Using non-invasive brain stimulation to augment motor training-induced plasticity", *Journal of Neuroengineering and Rehabilitation*, Vol. 6, p. 8, doi 10.1186/1743-0003-6-8.

boost community, *Boost C++ Libraries*.

Borton, D.A., Yin, M., Aceros, J. and Nurmikko, A. (2013), "An implantable wireless neural interface for recording cortical circuit dynamics in moving primates", *Journal of Neural Engineering*, Vol. 10 No. 2, p. 26010, doi 10.1088/1741-2560/10/2/026010.

Bowsher, D. (1995), "The management of central post-stroke pain", *Postgraduate Medical Journal*, Vol. 71 No. 840, pp. 598–604, doi 10.1136/pgmj.71.840.598.

Broetz, D., Braun, C., Weber, C., Soekadar, S.R., Caria, A. and Birbaumer, N. (2010), "Combination of brain-computer interface training and goal-directed physical therapy in chronic stroke: a case report", *Neurorehabilitation and Neural Repair*, Vol. 24 No. 7, pp. 674–679, doi 10.1177/1545968310368683.

Brown, J.A. (2001), "Motor cortex stimulation", *Neurosurgical Focus*, Vol. 11 No. 3, pp. E5.

Brunner, P., Ritaccio, A.L., Emrich, J.F., Bischof, H. and Schalk, G. (2011), "Rapid Communication with a "P300" Matrix Speller Using Electrocorticographic Signals (ECoG)", *Frontiers in Neuroscience*, Vol. 5, doi 10.3389/fnins.2011.00005.

Bundesministerium des Inneren (2014), "Das V-Modell XT", available at: http://www.v-modell-xt.de/ (accessed 1 August 2014).

Bundesregierung der Bundesrepublik Deutschland (2012), *Gesetz über Medizinprodukte (Medizinproduktegesetz - MPG): MPG*.

Burges, C.J. (1998), “A tutorial on support vector machines for pattern recognition”, *Data Mining and Knowledge Discovery*, Vol. 2 No. 2, pp. 121–167, doi 10.1023/A:1009715923555.

Bütefisch, C.M., Khurana, V., Kopylev, L. and Cohen, L.G. (2004), “Enhancing encoding of a motor memory in the primary motor cortex by cortical stimulation”, *Journal of Neurophysiology*, Vol. 91 No. 5, pp. 2110–2116, doi 10.1152/jn.01038.2003.

Carmena, J.M., Lebedev, M.A., Crist, R.E., O'Doherty, J.E., Santucci, D.M., Dimitrov, D.F., Patil, P.G., Henriquez, C.S. and Nicolelis, M.A.L. (2003), “Learning to Control a Brain–Machine Interface for Reaching and Grasping by Primates”, *PLoS Biology*, Vol. 1 No. 2, pp. 192–208, doi 10.1371/journal.pbio.0000042.

Chang, C.-C. and Lin, C.-J. (2011), “LIBSVM: A library for support vector machines”, *ACM Transactions on Intelligent Systems and Technology*, Vol. 2 No. 3, pp. 1–27, doi 10.1145/1961189.1961199.

Chao, Z., Nagasaka, Y. and Fuji, N. (2010), “Long-term asynchronous decoding of arm motion using electrocorticographic signals in monkey”, *Frontiers in Neuroengineering*, Vol. 3, doi 10.3389/fneng.2010.00003.

Charvet, G., Foerster, M., Chatalic, G., Michea, A., Porcherot, J., Bonnet, S., Filipe, S., Audebert, P., Robinet, S., Josselin, V., Reverdy, J., D'Errico, R., Sauter, F., Mestais, C. and Benabid, A.L. (2012), “A wireless 64-channel ECoG recording electronic for implantable monitoring and BCI applications: WIMAGINE”, in *Proceedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2012), San Diego, CA, USA, August 28 - September 1, 2012*, IEEE press, pp. 783–786, doi 10.1109/EMBC.2012.6346048.

Charvet, G., Foerster, M., Filipe, S., Porcherot, J., Beche, J.F., Guillemaud, R., Audebert, P., Regis, G., Zongo, B., Robinet, S., Condemine, C., Tetu, Y., Sauter, F., Mestais, C. and Benabid, A.L. (2011), “WIMAGINE: A wireless, low power, 64-channel ECoG recording platform for implantable BCI applications”, in *Proceedings of the 5th International IEEE EMBS Conference on Neural Engineering, Cancun, Mexico, April 27 - May 1, 2011*, IEEE press, pp. 356–359, doi 10.1109/NER.2011.5910560.

Charvet, G., Sauter-Starace, F., Foerster, M., Ratel, D., Chabrol, C., Porcherot, J., Robinet, S., Reverdy, J., D'Errico, R., Mestais, C. and Benabid, A.L. (2013), “WIMAGINE(®): 64-channel ECoG recording implant for human applications”, in *Proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2013), Osaka, Japan, July 3-7, 2013*, IEEE press, pp. 2756–2759, doi 10.1109/EMBC.2013.6610111.

Chatterjee, A., Aggarwal, V., Ramos, A., Acharya, S. and Thakor, N.V. (2007), “A brain-computer interface with vibrotactile biofeedback for haptic information”, *Journal of NeuroEngineering and Rehabilitation*, Vol. 4, p. 40, doi 10.1186/1743-0003-4-40.

Cincotti, F., Kauhanen, L., Aloise, F., Palomaki, T., Caporusso, N., Jylanki, P., Mattia, D., Babiloni, F., Vanacker, G., Nuttin, M., Marciani, M.G. and Millan, J.d.R. (2007), “Preliminary Experimentation on Vibrotactile Feedback in the context of Mu-rhythm Based BCI”, in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2007), Lyon, France, August 22-26, 2007*, IEEE press, Piscataway, NJ, pp. 4739–4742, doi 10.1109/IEMBS.2007.4353398.

Cincotti, F., Mattia, D., Aloise, F., Bufalari, S., Schalk, G., Oriolo, G., Cherubini, A., Marciani, M.G. and Babiloni, F. (2008), “Non-invasive brain-computer interface system: Towards its application as assistive technology”, *Brain Research Bulletin*, Vol. 75 No. 6, pp. 796–803, doi 10.1016/j.brainresbull.2008.01.007.

Collinger, J.L., Wodlinger, B., Downey, J.E., Wang, W., Tyler-Kabara, E.C., Weber, D.J., McMorland, A.J.C., Velliste, M., Boninger, M.L. and Schwartz, A.B. (2013), “High-performance neuroprosthetic

control by an individual with tetraplegia", *The Lancet*, Vol. 381 No. 9866, pp. 557–564, doi 10.1016/S0140-6736(12)61816-9.

Cook, M.J., O'Brien, T.J., Berkovic, S.F., Murphy, M., Morokoff, A., Fabinyi, G., D'Souza, W., Yerra, R., Archer, J., Litewka, L., Hosking, S., Lightfoot, P., Ruedebusch, V., Sheffield, W.D., Snyder, D., Leyde, K. and Himes, D. (2013), "Prediction of seizure likelihood with a long-term, implanted seizure advisory system in patients with drug-resistant epilepsy: a first-in-man study", *The Lancet Neurology*, Vol. 12 No. 6, pp. 563–571, doi 10.1016/S1474-4422(13)70075-9.

Corbet, J. and Rubini, A. (2005), *Linux Device Drivers,* 3rd ed., O'Reilly Media, Inc.

Council of the European Community (1990), *Council Directive of 20 June 1990 on the approximation of the laws of the Member States relating to active implantable medical devices: AIMD*.

Council of the European Community (1993a), *Council Directive 93/42/EEC of 14 June concerning medical devices: MDD*.

Council of the European Community (1993b), *Council Directive 93/68/EEC of 22 July 1993 amending 90/385/EEC (active implantable medical devices)*.

Council of the European Community (1994), *Corrigendum to the Council Directive 90/385/EEC of 20 june 1990 on the approximation of the laws of the Member States relating to active implantable medical devices*.

Council of the European Community (2006), *Council Directive 2006/96/EC of 20 November 2006 adapting certain Directives in the field of free movement of goods, by reason of the accession of Bulgaria and Romania*.

Cristianini, N. and Shawe-Taylor, J. (2000), *An introduction to support vector machines: And other kernel-based learning methods,* 1st ed., Cambridge University Press, Cambridge, New York.

Cunningham, J.P., Nuyujukian, P., Gilja, V., Chestek, C.A., Ryu, S.I. and Shenoy, K.V. (2011), "A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces", *Journal of Neurophysiology*, Vol. 105 No. 4, pp. 1932–1949, doi 10.1152/jn.00503.2010.

Davis, K.A., Sturges, B.K., Vite, C.H., Ruedebusch, V., Worrell, G., Gardner, A.B., Leyde, K., Sheffield, W.D. and Litt, B. (2011), "A novel implanted device to wirelessly record and analyze continuous intracranial canine EEG", *Epilepsy Research*, Vol. 96 1-2, pp. 116–122, doi 10.1016/j.eplepsyres.2011.05.011.

Domani, T., Kolodner, E.K. and Petrank, E. (2000), "A generational on-the-fly garbage collector for Java", *ACM SIGPLAN Notices*, Vol. 35 No. 5, pp. 274–284, doi 10.1145/358438.349336.

Duvall, P.M., Matyas, S. and Glover, A. (2007), *Continuous integration: Improving software quality and reducing risk, Addison-Wesley signature series*, Addison-Wesley, Upper Saddle River, NJ.

Edwardson, M.A., Lucas, T.H., Carey, J.R. and Fetz, E.E. (2013), "New modalities of brain stimulation for stroke rehabilitation", *Experimental Brain Research*, Vol. 224 No. 3, pp. 335–358, doi 10.1007/s00221-012-3315-1.

European Commission (2013), *Commission communication in the framework of the implementation of the Council Directive 90/385/EEC of 20 June 1990 on the approximation of the laws of the Member States relating to active implantable medical devices*.

European Committee for Electrotechnical Standardization (1997), *Active implantable medical devices - Part 1: General requirements for safety, marking and information to be provided by the manufacturer* EN 45502-1.

European Committee for Electrotechnical Standardization (2003), *Active implantable medical devices - Part 2-1: Particular requirements for active implantable medical devices intended to treat bradyarrhythmia (cardiac pacemakers)* EN 45502-2-1.

European Committee for Electrotechnical Standardization (2006), *Medical device software - Software life-cycle processes* EN IEC 62304.

European Committee for Electrotechnical Standardization (2012a), *Medical devices - Application of risk management to medical devices* EN ISO 14971.

European Committee for Electrotechnical Standardization (2012b), *Medical devices - Quality management systems- Requirements for regulatory purposes* EN ISO 13485.

European Parliament and Council (1998a), *Directive 98/34/EC of the European Parliament and of the Council of 22 June 1998 laying down a procedure for the provision of information in the field of technical standards and regulations*.

European Parliament and Council (1998b), *Directive 98/48/EC of the European Parliament and of the Council of 20 July 1998 amending Directive 98/34/EC laying down a procedure for the provision of information in the field of technical standards and regulations*.

European Parliament and Council (1998c), *Directive 98/79/EC of the European Parliament and of the Council of 27 October 1998 on in vitro diagnostic medical devices*.

European Parliament and Council (1999), *Corrigendum to Directive 98/79/EC of the European Parliament and of the Council of 27 October 1998 on in vitro diagnostic medical devices*.

European Parliament and Council (2000), *Directive 2000/70/EC of the European Parliament and of the Council of 16 November 2000 amending Council Directive 93/42/EEC as regards medical devices incorporating stable derivates of human blood or human plasma*.

European Parliament and Council (2001), *Directive 2001/104/EC of the European Parliament and of the Council of 7 December 2001 amending Council Directive 93/42/EEC concerning medical devices*.

European Parliament and Council (2002), *Corrigendum to Directive 98/79/EC of the European Parliament and of the Council of 27 October 1998 on in vitro diagnostic medical devices*.

European Parliament and Council (2003a), *Act concerning the conditions of accession of the Czech Republic, the Republic of Estonia, the Republic of Cyprus, the Republic of Latvia, the Republic of Lithuania, the Republic of Hungary, the Republic of Malta, the Republic of Poland, the Republic of Slovenia and the Slovak Republic and the adjustments to the Treaties on which the European Union is founded*.

European Parliament and Council (2003b), *Regulation (EC) No 1882/2003 of the European Parliament and of the Council of 29 September 2003 adapting to Council Decision 1999/468/EC the provisions relating to committees which assist the Commission in the exercise of its implementing powers laid down in instruments subject to the procedure referred to in Article 251 of the EC Treaty*.

European Parliament and Council (2007), *Directive 2007/47/EC of the European Parliament and of the Council of 5 September 2007 amending Council Directive 90/385/EEC on the approximation of the laws of the Member States relating to active implantable medical devices, Council Directive 93/42/EEC concerning medical devices and Directive 98/8/EC concerning the placing of biocidal products on the market*.

European Parliament and Council (2009), *Regulation (EC) No 596/2009 of the European Parliament and of the Council of 18 June 2009 adapting a number of instruments subject to the procedure referred to in Article 251 of the Treaty to Council Decision 1999/468/EC with regard to the regulatory procedure with scrutiny — Adaptation to the regulatory procedure with scrutiny — Part Four*.

European Parliament and Council (2010), *Directive 2010/63/EU of the European Parliament and of the Council of 22 September 2010 on the protection of animals used for scientific purposes*.

European Parliament and Council (2011), *Commission Directive 2011/100/EU of 20 December 2011 amending Directive 98/79/EC of the European Parliament and of the Council on in-vitro diagnostic medical devices*.

Fischer, J., Milekovic, T., Schneider, G. and Mehring, C. (2014), "Low-latency multi-threaded processing of neuronal signals for brain-computer interfaces", *Frontiers in Neuroengineering*, Vol. 7, p. 1, doi 10.3389/fneng.2014.00001.

Fitzsimmons, N., Drake, W., Hanson, T.L., Lebedev, M.A. and Nicolelis, M.A.L. (2007), "Primate reaching cued by multichannel spatiotemporal cortical microstimulation", *Journal of Neuroscience*, Vol. 27 No. 21, pp. 5593–5602.

Flinker, A., Chang, E.F., Barbaro, N.M., Berger, M.S. and Knight, R.T. (2011), "Sub-centimeter language organization in the human temporal lobe", *Brain and Language*, Vol. 117 No. 3, pp. 103–109, doi 10.1016/j.bandl.2010.09.009.

Fowler, M. (2003), *Patterns of enterprise application architecture, The Addison-Wesley Signature Series*, Addison-Wesley, Boston.

Franke, F., Jäckel, D., Dragas, J., Müller, J., Radivojevic, M., Bakkum, D.J. and Hierlemann, A. (2012), "High-density microelectrode array recordings and real-time spike sorting for closed-loop experiments: an emerging technology to study neural plasticity", *Frontiers in Neural Circuits*, Vol. 6, doi 10.3389/fncir.2012.00105.

Freeman, S. and Pryce, N. (2011), *Growing object-oriented software, guided by tests, The Addison-Wesley Signature Series,* 5th ed., Addison-Wesley, Boston, MA.

Freeman, W.J., Rogers, L.J., Holmes, M.D. and Silbergeld, D.L. (2000), "Spatial spectral analysis of human electrocorticograms including the alpha and gamma bands", *Journal of Neuroscience Methods*, Vol. 95 No. 2, pp. 111–121, doi 10.1016/S0165-0270(99)00160-0.

Friedman, J.H. (1989), "Regularized Discriminant Analysis", *Journal of the American Statistical Association*, Vol. 84 No. 405, pp. 165–175, doi 10.2307/2289860.

Galán, F., Nuttin, M., Lew, E., Ferrez, P.W., Vanacker, G., Philips, J. and Millan, J.d.R. (2008), "A brain-actuated wheelchair: Asynchronous and non-invasive Brain–computer interfaces for continuous control of robots", *Clinical Neurophysiology*, Vol. 119 No. 9, pp. 2159–2169, doi 10.1016/j.clinph.2008.06.001.

Galassi, M. (2009), *GNU scientific library: Reference manual,* 3rd ed., Network Theory Ltd, Bristol.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994), *Design patterns: Elements of reusable object-oriented software, Addison Wesley Professional Computing Series*, Addison-Wesley, Reading, MA.

Gao, Y., Black, M.J., Bienenstock, E., Wu, W. and Donoghue, J.P. (2003), "A quantitative comparison of linear and non-linear models of motor cortical activity for the encoding and decoding of arm motions", in *Proceedings of the 1st International IEEE EMBS Conference on Neural Engineering, Capri Island, Italy, March 20-22, 2003*, IEEE press, pp. 189–192, doi 10.1109/CNE.2003.1196789.

Gharabaghi, A., Kraus, D., Leão, M.T., Spüler, M., Walter, A., Bogdan, M., Rosenstiel, W., Naros, G. and Ziemann, U. (2014), "Coupling brain-machine interfaces with cortical stimulation for brain-state dependent stimulation: enhancing motor cortex excitability for neurorehabilitation", *Frontiers in Human Neuroscience*, Vol. 8, doi 10.3389/fnhum.2014.00122.

Gibbons, J.D. and Chakraborti, S. (2011), *Nonparametric statistical inference, Statistics Textbooks and Monographs,* 5th ed., Chapman & Hall/Taylor & Francis, Boca Raton.

Gierthmuehlen, M., Wang, X., Gkogkidis, A., Henle, C., Fischer, J., Fehrenbacher, T., Kohler, F., Raab, M., Mader, I., Kuehn, C., Foerster, K., Haberstroh, J., Freiman, T.M., Stieglitz, T., Rickert, J.,

Schuettler, M. and Ball, T. (2014), “Mapping of sheep sensory cortex with a novel micro-electrocorticography grid [epub ahead of print]”, *Journal of Comparative Neurology*, doi 10.1002/cne.23631.

Gonzalez Andino, S.L., Herrera-Rincon, C., Panetsos, F. and Grave de Peralta, R. (2011), “Combining BMI Stimulation and Mathematical Modeling for Acute Stroke Recovery and Neural Repair”, *Frontiers in Neuroscience*, Vol. 5, doi 10.3389/fnins.2011.00087.

googlemock community (2014), “googlemock. google C++ Mocking Framework”, available at: http://code.google.com/p/googlemock/ (accessed 1 August 2014).

googletest community (2014), “googletest. google C++ Testing Framework”, available at: http://code.google.com/p/googletest/ (accessed 1 August 2014).

Gunduz, A., Sanchez, J.C., Carney, P.R. and Principe, J.C. (2009), “Mapping broadband electrocorticographic recordings to two-dimensional hand trajectories in humans”, *Neural Networks*, Vol. 22 No. 9, pp. 1257–1270, doi 10.1016/j.neunet.2009.06.036.

Halpern, C.H., Samadani, U., Litt, B., Jaggi, J.L. and Baltuch, G.H. (2008), “Deep brain stimulation for epilepsy”, *Neurotherapeutics*, Vol. 5 No. 1, pp. 59–67, doi 10.1016/j.nurt.2007.10.065.

Halvorsen, O.H. and Clarke, D. (2011), *OS X and iOS kernel programming,* 1st ed., Apress; Distributed to the book trade by Springer, Berkeley, CA, New York.

Hamer, H.M., Morris, H.H., Mascha, E.J., Karafa, M.T., Bingaman, W.E., Bej, M.D., Burgess, R., Dinner, D.S., Foldvary, N.R., Hahn, J.F., Kotagal, P., Najm, I., Wyllie, E. and Lüders, H. (2002), “Complications of invasive video-EEG monitoring with subdural grid electrodes”, *Neurology*, Vol. 58 No. 1, pp. 97–103, doi 10.1212/WNL.58.1.97.

Hamill, P. (2004), *Unit test frameworks,* 1st ed., O’Reilly Media, Inc, Sebastopol, CA.

Hart, J.M. (2010), *Windows system programming: Description based on print version record, Addison-Wesley Microsoft Technology Series,* 4th ed., Addison-Wesley, Upper Saddle River, N.J.

Harvey, R.L. and Nudo, R.J. (2007), “Cortical Brain Stimulation: A Potential Therapeutic Agent for Upper Limb Motor Recovery Following Stroke”, *Topics in Stroke Rehabilitation*, Vol. 14 No. 6, pp. 54–67, doi 10.1310/tsr1406-54.

Hastie, T., Tibshirani, R. and Friedman, J.H. (2009), *The elements of statistical learning: Data mining, inference, and prediction, Springer Series in Statistics,* 2nd ed., Springer-Verlag, New York.

Haykin, S.S. (Ed.) (2001), *Kalman filtering and neural networks, Adaptive and Learning Systems for Signal Processing, Communications, and Control*, John Wiley & Sons, Inc, New York.

Henle, C., Hassler, C., Kohler, F., Schuettler, M. and Stieglitz, T. (2011), “Mechanical characterization of neural electrodes based on PDMS-parylene C-PDMS sandwiched system”, in *Proceedings of the 33th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2011), Boston, Massachusetts, USA, August 30 - September 3, 2011*, IEEE press, pp. 640–643, doi 10.1109/IEMBS.2011.6090142.

Hochberg, L.R., Bacher, D., Jarosiewicz, B., Masse, N.Y., Simeral, J.D., Vogel, J., Haddadin, S., Liu, J., Cash, S.S., van der Smagt, Patrick and Donoghue, J.P. (2012), “Reach and grasp by people with tetraplegia using a neurally controlled robotic arm”, *Nature*, Vol. 485 No. 7398, pp. 372–375, doi 10.1038/nature11076.

Hochberg, L.R., Serruya, M.D., Friehs, G.M., Mukand, J.A., Saleh, M., Caplan, A.H., Branner, A., Chen, D., Penn, R.D. and Donoghue, J.P. (2006), “Neuronal ensemble control of prosthetic devices by a human with tetraplegia”, *Nature*, Vol. 442 No. 7099, pp. 164–171, doi 10.1038/nature04970.

Huettel, S.A., Song, A.W. and McCarthy, G. (2008), *Functional magnetic resonance imaging,* 2nd ed., Sinauer Associates, Sunderland, Mass.

IEEE Standards Association, *IEEE Standard for Floating-Point Arithmetic* IEEE Std 754:2008.

International Organization for Standards, *Programming Language C++* ISO/IEC 14882:2011.

International Organization for Standards, *Programming Languages - C* ISO/IEC 9899:1999.

International Organization for Standards (2005), *Quality management systems - Fundamentals and vocabulary* ISO 9000.

International Organization for Standards (2006), *Information technology - Programming languages - C#* ISO/IEC 23270:2006.

International Organization for Standards (2008), *Quality management systems - Requirements* ISO 9001.

Jackson, A., Mavoori, J. and Fetz, E.E. (2006), "Long-term motor cortex plasticity induced by an electronic neural implant", *Nature*, Vol. 444 No. 7115, pp. 56–60, doi 10.1038/nature05226.

Jensen, O., Bahramisharif, A., Oostenveld, R., Klanke, S., Hadjipapas, A., Okazaki, Y.O. and van Gerven, M.A.J. (2011), "Using brain-computer interfaces and brain-state dependent stimulation as tools in cognitive neuroscience", *Frontiers in Psychology*, Vol. 2, doi 10.3389/fpsyg.2011.00100.

Jöbsis, F.F. (1977), "Noninvasive, infrared monitoring of cerebral and myocardial oxygen sufficiency and circulatory parameters", *Science*, Vol. 198 No. 4323, pp. 1264–1267, doi 10.1126/science.929199.

Johner, C., Hölzer-Klüpfel, M. and Wittorf, S. (2011), *Basiswissen medizinische Software: Aus- und Weiterbildung zum Certified Professional for Medical Software,* 1st ed., dpunkt-Verlag, Heidelberg.

Johnson, J.I., Rubel, E.W. and Hatton, G.I. (1974), "Mechanosensory projections to cerebral cortex of sheep", *Journal of Comparative Neurology*, Vol. 158 No. 1, pp. 81–108, doi 10.1002/cne.901580106.

Kai Keng Ang, Zhang Yang Chin, Haihong Zhang and Cuntai Guan (2008), "Filter Bank Common Spatial Pattern (FBCSP) in Brain-Computer Interface", in *International Joint Conference on Neural Networks (IJCNN 2008), Hong Kong, China, June 1-8, 2008*, IEEE press, pp. 2390–2397, doi 10.1109/IJCNN.2008.4634130.

Kalman, R.E. (1960), "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering*, Vol. 82 No. 1, pp. 35–45, doi 10.1115/1.3662552.

Karlsson, B. (2006), *Beyond the C++ standard library: An introduction to Boost,* 6th ed., Addison-Wesley, Upper Saddle River, NJ.

Kawamoto, H., Taal, S., Niniss, H., Hayashi, T., Kamibayashi, K., Eguchi, K. and Sankai, Y. (2010), "Voluntary motion support control of Robot Suit HAL triggered by bioelectrical signal for hemiplegia", in *Proceedings of the 32nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2010), Buenos Aires, Argentina, August 31 - September 4, 2010*, IEEE press, pp. 462–466, doi 10.1109/IEMBS.2010.5626191.

Kenney, C., Simpson, R., Hunter, C., Ondo, W., Almaguer, M., Davidson, A. and Jankovic, J. (2007), "Short-term and long-term safety of deep brain stimulation in the treatment of movement disorders", *Journal of Neurosurgery*, Vol. 106 No. 4, pp. 621–625, doi 10.3171/jns.2007.106.4.621.

Kern, M., Aertsen, A., Schulze-Bonhage, A. and Ball, T. (2013), "Heart cycle-related effects on event-related potentials, spectral power changes, and connectivity patterns in the human ECoG", *NeuroImage*, Vol. 81, pp. 178–190, doi 10.1016/j.neuroimage.2013.05.042.

Kiernan, M.C., Vucic, S., Cheah, B.C., Turner, M.R., Eisen, A., Hardiman, O., Burrell, J.R. and Zoing, M.C. (2011), “Amyotrophic lateral sclerosis”, *The Lancet*, Vol. 377 No. 9769, pp. 942–955, doi 10.1016/S0140-6736(10)61156-7.

Kim, S.-P., Simeral, J.D., Hochberg, L.R., Donoghue, J.P., Friehs, G.M. and Black, M.J. (2011), “Point-and-click cursor control with an intracortical neural interface system by humans with tetraplegia”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 19 No. 2, pp. 193–203, doi 10.1109/TNSRE.2011.2107750.

Knowlton, R.C. and Shih, J. (2004), “Magnetoencephalography in Epilepsy”, *Epilepsia*, Vol. 45 Suppl. 4, pp. 61–71, doi 10.1111/j.0013-9580.2004.04012.x.

Kohler, F., Fischer, J., Gierthmuehlen, M., Gkogkidis, A., Henle, C., Ball, T., Wang, X., Rickert, J., Stieglitz, T. and Schuettler, M., *Long-term in vivo validation of a fully-implantable, wireless brain-computer interface for cortical recording and stimulation*, in preparation.

Kohler, F., Schuettler, M. and Stieglitz, T. (2012), “Parylene-coated metal tracks for neural electrode arrays - fabrication approaches and improvements utilizing different laser systems”, in *Proceedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2012), San Diego, CA, USA, August 28 - September 1, 2012*, IEEE press, pp. 5130–5133, doi 10.1109/EMBC.2012.6347148.

Kohler, F., Ulloa, M.A., Ordonez, J.S., Stieglitz, T. and Schuettler, M. (2013), “Reliability investigations and improvements of interconnection technologies for the wireless brain-machine interface — ‘BrainCon’”, in *6th Annual International IEEE EMBS Conference on Neural Engineering (NER 2013), San Diego, California, USA, November 6-8, 2013*, IEEE press, pp. 1013–1016, doi 10.1109/NER.2013.6696108.

Kopetz, H. (2011), *Real-Time Systems: Design Principles for Distributed Embedded Applications, Real-Time Systems Series,* 2nd ed., Springer-Verlag, Boston, MA, UAS.

Krepki, R., Blankertz, B., Curio, G. and Müller, K.-R. (2003), “The Berlin Brain-Computer Interface (BBCI): towards a new communication channel for online control of multimedia applications and computer games”, in *9th International Conference on Distributed Multimedia Systems (DMS-2003), Miami, Florida, USA, September 24-26, 2003*, Knowledge Systems Institute, pp. 237–244.

Krepki, R., Blankertz, B., Curio, G. and Müller, K.-R. (2007), “The Berlin Brain-Computer Interface (BBCI) – towards a new communication channel for online control in gaming applications”, *Multimedia Tools and Applications*, Vol. 33 No. 1, pp. 73–90, doi 10.1007/s11042-006-0094-3.

LaFleur, K., Cassady, K., Doud, A., Shades, K., Rogin, E. and He, B. (2013), “Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain–computer interface”, *Journal of Neural Engineering*, Vol. 10 No. 4, p. 46003, doi 10.1088/1741-2560/10/4/046003.

Lebedev, M.A., Carmena, J.M., O'Doherty, J.E., Zacksenhouse, M., Herniquez, C.S., Principe, J.C. and Nicolelis, M.A.L. (2005), “Cortical Ensemble Adaptation to Represent Velocity of an Artificial Actuator Controlled by a Brain-Machine Interface”, *Journal of Neuroscience*, Vol. 25 No. 19, pp. 4681–4693, doi 10.1523/JNEUROSCI.4088-04.2005.

Leeb, R., Friedman, D., Müller-Putz, G.R., Scherer, R., Slater, M. and Pfurtscheller, G. (2007), “Self-paced (asynchronous) BCI control of a wheelchair in virtual environments: a case study with a tetraplegic”, *Computational Intelligence and Neuroscience*, Vol. 2007, p. 79642, doi 10.1155/2007/79642.

Leuthardt, E.C., Schalk, G., Roland, J., Rouse, A. and Moran, D.W. (2009), “Evolution of brain-computer interfaces: going beyond classic motor physiology”, *Neurosurgical Focus*, Vol. 27 No. 1, pp. E4, doi 10.3171/2009.4.FOCUS0979.

Levy, R., Ruland, S., Weinand, M., Lowry, D., Dafer, R. and Bakay, R. (2008), "Cortical stimulation for the rehabilitation of patients with hemiparetic stroke: a multicenter feasibility study of safety and efficacy", *Journal of Neurosurgery*, Vol. 108 No. 4, pp. 707–714, doi 10.3171/JNS/2008/108/4/0707.

Li, T., Baumberger, D. and Hahn, S. (2009), "Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin", in Reed, D.A. and Sarkar, V. (Eds.), *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPOPP-09), Raleigh, North Carolina, USA, February 14-18, 2009*, ACM, pp. 65–74, doi 10.1145/1594835.1504188.

Lockheed Martin Corporation (2005), *Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program*.

Lorenz, S. (2006), *Lehrbuch der anthroposophischen Tiermedizin: 39 Tabellen*, Sonntag, Stuttgart.

Ludwig, K.A., Miriani, R.M., Langhals, N.B., Joseph, M.D., Anderson, D.J. and Kipke, D.R. (2009), "Using a common average reference to improve cortical neuron recordings from microelectrode arrays", *Journal of Neurophysiology*, Vol. 101 No. 3, pp. 1679–1689, doi 10.1152/jn.90989.2008.

Maling, N., Hashemiyoon, R., Foote, K.D., Okun, M.S., Sanchez, J.C. and Baumert, M. (2012), "Increased Thalamic Gamma Band Activity Correlates with Symptom Relief following Deep Brain Stimulation in Humans with Tourette's Syndrome", *PLoS ONE*, Vol. 7 No. 9, pp. E44215, doi 10.1371/journal.pone.0044215.

Marceglia, S., Rossi, L., Foffani, G., Bianchi, A., Cerutti, S. and Priori, A. (2007), "Basal ganglia local field potentials: applications in the development of new deep brain stimulation devices for movement disorders", *Expert Review of Medical Devices*, Vol. 4 No. 5, pp. 605–614, doi 10.1586/17434440.4.5.605.

Maropoulos, P.G. and Ceglarek, D. (2010), "Design verification and validation in product lifecycle", *CIRP Annals - Manufacturing Technology*, Vol. 59 No. 2, pp. 740–759, doi 10.1016/j.cirp.2010.05.005.

Marr, D.T., Binns, F., Hill, D.L., Hinton, G., Koufaty, D.A., Miller, J.A. and Upton, M. (2002), "Hyper-Threading Technology Architecture and Microarchitecture", *Intel Technology Journal*, Vol. 6 No. 1, pp. 1–12.

McCreadie, K.A., Coyle, D.H. and Prasad, G. (2013), "Sensorimotor learning with stereo auditory feedback for a brain-computer interface", *Medical & Biological Engineering & Computing*, Vol. 51 No. 3, pp. 285–293, doi 10.1007/s11517-012-0992-7.

Medtronic, Inc. (2013), *New Medtronic Deep Brain Stimulation System the First to Sense and Record Brain Activity While Delivering Therapy*.

Mellinger, J., Schalk, G., Braun, C., Preissl, H., Rosenstiel, W., Birbaumer, N. and Kübler, A. (2007), "An MEG-based brain–computer interface (BCI)", *NeuroImage*, Vol. 36 No. 3, pp. 581–593, doi 10.1016/j.neuroimage.2007.03.019.

Merletti, R. and Parker, P.M. (Eds.) (2004), *Electromyography: Physiology, engineering and non-invasive applications, IEEE Press Series in Biomedical Engineering*, John Wiley & Sons, Inc; IEEE press, Hoboken, N.J.

Microsoft Developer Network Library (2014), "Microsoft-Specific Modifiers _declspec(thread)", available at: http://msdn.microsoft.com/en-us/library/9w1sdazb(v=vs.110).aspx (accessed 1 August 2014).

Milekovic, T., Ball, T., Schulze-Bonhage, A., Aertsen, A. and Mehring, C. (2013), "Detection of Error Related Neuronal Responses Recorded by Electrocorticography in Humans during Continuous Movements", *PLoS ONE*, Vol. 8 No. 2, pp. E55235, doi 10.1371/journal.pone.0055235.

Milekovic, T., Fischer, J., Pistohl, T., Ruescher, J., Schulze-Bonhage, A., Aertsen, A., Rickert, J., Ball, T. and Mehring, C. (2012), "An online brain-machine interface using decoding of movement direction from the human electrocorticogram", *Journal of Neural Engineering*, Vol. 9 No. 4, p. 46003, doi 10.1088/1741-2560/9/4/046003.

Millan, J.d.R. and Carmena, J.M. (2010), "Invasive or Noninvasive: Understanding Brain-Machine Interface Technology. Conversations in BME", *IEEE Engineering in Medicine and Biology Magazine*, Vol. 29 No. 1, pp. 16–22, doi 10.1109/MEMB.2009.935475.

Miller, C., Gitina, K. and Becker, B. (2011), "Bounded Model Checking of Incomplete Real-time Systems Using Quantified SMT Formulas", in Abadir, M.S., Bhadra, J. and Wang, L.-C. (Eds.), *2011 12th International Workshop on Microprocessor Test and Verification (MTV-2011), Austin, TX, USA, December 5-7, 2011*, IEEE press, Los Alamitos, Calif, pp. 22–27, doi 10.1109/MTV.2011.13.

Modolo, J., Beuter, A., Thomas, A.W. and Legros, A. (2012), "Using "Smart Stimulators" to Treat Parkinson's Disease: Re-Engineering Neurostimulation Devices", *Frontiers in Computational Neuroscience*, Vol. 6, doi 10.3389/fncom.2012.00069.

Morrell, M.J. (2011), "Responsive cortical stimulation for the treatment of medically intractable partial epilepsy", *Neurology*, Vol. 77 No. 13, pp. 1295–1304, doi 10.1212/wnl.0b013e3182302056.

Müller, J., Bakkum, D.J. and Hierlemann, A. (2013), "Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons", *Frontiers in Neural Circuits*, Vol. 6, doi 10.3389/fncir.2012.00121.

Nair, D.R., Burgess, R., McIntyre, C.C. and Lüders, H. (2008), "Chronic subdural electrodes in the management of epilepsy", *Clinical Neurophysiology*, Vol. 119 No. 1, pp. 11–28, doi 10.1016/j.clinph.2007.09.117.

NeuroPace, Inc. (2013), *RNS® System User Manual*.

Newman, J.P., Zeller-Townson, R., Fong, M.-F., Arcot Desai, S., Gross, R.E. and Potter, S.M. (2013), "Closed-Loop, Multichannel Experimentation Using the Open-Source NeuroRighter Electrophysiology Platform", *Frontiers in Neural Circuits*, Vol. 6, doi 10.3389/fncir.2012.00098.

Nicolas-Alonso, L.F. and Jaime, G.-G. (2012), "Brain Computer Interfaces, a Review", *Sensors*, Vol. 12 No. 2, pp. 1211–1279, doi 10.3390/s120201211.

Nicolelis, M.A.L., Dimitrov, D.F., Carmena, J.M., Crist, R.E., Lehew, G., Kralik, J.D. and Wise, S.P. (2003), "Chronic, multisite, multielectrode recordings in macaque monkeys", *Proceedings of the National Academy of Sciences*, Vol. 100 No. 19, pp. 11041–11046, doi 10.1073/pnas.1934665100.

Nijboer, F., Furdea, A., Gunst, I., Mellinger, J., McFarland, D.J., Birbaumer, N. and Kübler, A. (2008), "An auditory brain-computer interface (BCI)", *Journal of Neuroscience Methods*, Vol. 167 No. 1, pp. 43–50, doi 10.1016/j.jneumeth.2007.02.009.

O'Doherty, J.E., Lebedev, M.A., Hanson, T.L., Fitzsimmons, N. and Nicolelis, M.A.L. (2009), "A brain-machine interface instructed by direct intracortical microstimulation", *Frontiers in Integrative Neuroscience*, Vol. 3, doi 10.3389/neuro.07.020.2009.

Oken, B.S., Orhan, U., Roark, B., Erdogmus, D., Fowler, A., Mooney, A., Peters, B., Miller, M. and Fried-Oken, M.B. (2014), "Brain-Computer Interface With Language Model-Electroencephalography Fusion for Locked-In Syndrome", *Neurorehabilitation and Neural Repair*, Vol. 28 No. 4, pp. 387–394, doi 10.1177/1545968313516867.

OpenMP (2014), “OpenMP. The OpenMP API specification for parallel programming”, available at: http://openmp.org (accessed 1 August 2014).

Oppenheim, A.V., Schafer, R.W. and Buck, J.R. (1999), *Discrete time signal processing, Prentice-Hall Signal Processing Series,* 2nd ed., Prentice-Hall, Inc, Upper Saddler River, NJ.

Osorio, I., Frei, M.G., Manly, B.F.J., Sunderam, S., Bhavaraju, N.C. and Wilkinson, S.B. (2001), “An Introduction to Contingent (Closed-Loop) Brain Electrical Stimulation for Seizure Blockage, to Ultra-short-term Clinical Trials, and to Multidimensional Statistical Analysis of Therapeutic Efficacy”, *Journal of Clinical Neurophysiology*, Vol. 18 No. 6, pp. 533–544, doi 10.1097/00004691-200111000-00003.

Otto, K.J., Johnson, M.D. and Kipke, D.R. (2006), “Voltage pulses change neural interface properties and improve unit recordings with chronically implanted microelectrodes”, *IEEE Transactions on Biomedical Engineering*, Vol. 53 No. 2, pp. 333–340, doi 10.1109/TBME.2005.862530.

Perego, P., Maggi, L., Parini, S. and Andreoni, G. (2009), “BCI++. A New Framework for Brain Computer Interface Application”, in Al-Mubaid, H. and Dascalu, A. (Eds.), *18th International Conference on Software Engineering and Data Engineering (SEDE-2009), Las Vegas, Nevada, USA, June 22-24, 2009*, ISCA, pp. 37–41.

Plow, E.B., Carey, J.R., Nudo, R.J. and Pascual-Leone, A. (2009), “Invasive cortical stimulation to promote recovery of function after stroke: a critical appraisal”, *Stroke*, Vol. 40 No. 5, pp. 1926–1931, doi 10.1161/STROKEAHA.108.540823.

Ramos-Murguialday, A., Broetz, D., Rea, M., Läer, L., Yilmaz, O., Brasil, F.L., Liberati, G., Curado, M.R., Garcia-Cossio, E., Vyziotis, A., Cho, W., Agostini, M., Soares, E., Soekadar, S., Caria, A., Cohen, L.G. and Birbaumer, N. (2013), “Brain-machine interface in chronic stroke rehabilitation: a controlled study”, *Annals of Neurology*, Vol. 74 No. 1, pp. 100–108, doi 10.1002/ana.23879.

Rebsamen, B., Guan, C., Zhang, H., Wang, C., Teo, C., Ang, M.H. and Burdet, E. (2010), “A brain controlled wheelchair to navigate in familiar environments”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 18 No. 6, pp. 590–598, doi 10.1109/TNSRE.2010.2049862.

Renard, Y., Lotte, F., Gibert, G., Congedo, M., Maby, E., Delannoy, V., Bertrand, O. and Lécuyer, A. (2010), “OpenViBE: An Open-source Software Platform to Design, Test, and Use Brain-computer Interfaces in Real and Virtual Environments”, *Presence Teleoperators and Virtual Environments*, Vol. 19 No. 1, pp. 35–53, doi 10.1162/pres.19.1.35.

Rickert, J., Cardoso de Oliveira, S., Vaadia, E., Aertsen, A., Rotter, S. and Mehring, C. (2005), “Encoding of movement direction in different frequency ranges of motor cortical local field potentials”, *Journal of Neuroscience*, Vol. 25 No. 39, pp. 8815–8824, doi 10.1523/JNEUROSCI.0816-05.2005.

Rizk, M., Bossetti, C.A., Jochum, T.A., Callender, S.H., Nicolelis, M.A.L., Turner, D.A. and Wolf, P.D. (2009), “A fully implantable 96-channel neural data acquisition system”, *Journal of Neural Engineering*, Vol. 6 No. 2, p. 26002, doi 10.1088/1741-2560/6/2/026002.

Romo, R., Hernández, A., Zainos, A., Brody, C.D. and Lemus, L. (2000), “Sensing without Touching”, *Neuron*, Vol. 26 No. 1, pp. 273–278, doi 10.1016/S0896-6273(00)81156-3.

Roscoe, A.W. (1998), *The theory and practice of concurrency, Prentice Hall Series in Computer Science*, Prentice-Hall, Inc, London, New York.

Rosin, B., Slovik, M., Mitelman, R., Rivlin-Etzion, M., Haber, S.N., Israel, Z., Vaadia, E. and Bergman, H. (2011), “Closed-loop deep brain stimulation is superior in ameliorating parkinsonism”, *Neuron*, Vol. 72 No. 2, pp. 370–384, doi 10.1016/j.neuron.2011.08.023.

Rotermund, D., Ernst, U.A. and Pawelzik, K.R. (2006), "Towards On-line Adaptation of Neuroprostheses with Neuronal Evaluation Signals", *Biological Cybernetics*, Vol. 95 No. 3, pp. 243–257, doi 10.1007/s00422-006-0083-7.

Rouse, A.G., Stanslaski, S.R., Cong, P., Jensen, R.M., Afshar, P., Ullestad, D., Gupta, R., Molnar, G.F., Moran, D.W. and Denison, T.J. (2011), "A chronic generalized bi-directional brain–machine interface", *Journal of Neural Engineering*, Vol. 8 No. 3, p. 36018, doi 10.1088/1741-2560/8/3/036018.

Royce, W.W. (1970), "Managing the Development of Large Software Systems", in *Proceeding of the IEEE Western Electronic Show and Convention (Wescon), Los Angeles, California, USA, August 25-28, 1970*, IEEE press, pp. 328–338.

Royer, T.C. (1993), *Software testing management: Life on the critical path*, Prentice-Hall, Inc, Englewood Cliffs, N.J.

Ryu, S.I. and Shenoy, K.V. (2009), "Human cortical prostheses: lost in translation?", *Neurosurgical Focus*, Vol. 27 No. 1, E5, doi 10.3171/2009.4.FOCUS0987.

Santos, E.E. and Chu, P.-Y. (2003), "Efficient and Optimal Parallel Algorithms for Cholesky Decomposition", *Journal of Mathematical Modelling and Algorithms*, Vol. 2 No. 3, pp. 217–234, doi 10.1023/B:JMMA.0000015832.41014.ed.

Sauer, M., Czutro, A., Polian, I. and Becker, B. (2011a), "Estimation of component criticality in early design steps", in *2011 IEEE 17th International On-Line Testing Symposium (IOLTS 2011), Athens, Greece, July 13-15, 2011*, IEEE press, pp. 104–110, doi 10.1109/IOLTS.2011.5993819.

Sauer, M., Tomashevich, V., Muller, J., Lewis, M., Spilla, A., Polian, I., Becker, B. and Burgard, W. (2011b), "An FPGA-based framework for run-time injection and analysis of soft errors in microprocessors", in *2011 IEEE 17th International On-Line Testing Symposium (IOLTS 2011), Athens, Greece, July 13-15, 2011*, IEEE press, pp. 182–185, doi 10.1109/IOLTS.2011.5993836.

Savitzky, A. and Golay, M.J.E. (1964), "Smoothing and Differentiation of Data by Simplified Least Squares Procedures", *Analytical Chemistry*, Vol. 36 No. 8, pp. 1627–1639, doi 10.1021/ac60214a047.

Schalk, G. (2009), "Effective brain-computer interfacing using BCI2000", in *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2009), Minneapolis, Minnesota, USA, September 2-6. 2009*, IEEE press, pp. 5498–5501, doi 10.1109/IEMBS.2009.5334558.

Schalk, G. (2010), "Can Electrocorticography (ECoG) Support Robust and Powerful Brain-Computer Interfaces?", *Frontiers in Neuroengineering*, Vol. 3, doi 10.3389/fneng.2010.00009.

Schalk, G., McFarland, D.J., Hinterberger, T., Birbaumer, N. and Wolpaw, J.R. (2004), "BCI2000: a general-purpose brain-computer interface (BCI) system", *IEEE Trans. on Biomed. Eng.*, Vol. 51 No. 6, pp. 1034–1043, doi 10.1109/TBME.2004.827072.

Schalk, G., Miller, K.J., Anderson, N.R., Wilson, J.A., Smyth, M.D., Ojemann, J.G., Moran, D.W., Wolpaw, J.R. and Leuthardt, E.C. (2008), "Two-dimensional movement control using electrocorticographic signals in humans", *Journal of Neural Engineering*, Vol. 5 No. 1, pp. 75–84, doi 10.1088/1741-2560/5/1/008.

Schlögl, A., Brunner, C., Scherer, R. and Glatz, A. (2007), "BioSig: An Open-Source Software Library for BCI Research", in Dornhege, G., Millan, J.d.R., Hinterberger, T., McFarland, D.J. and Müller, K.-R. (Eds.), *Toward brain-computer interfacing*, *Neural Information Processing Series*, MIT Press, Cambridge, Mass, pp. 347–358.

Schneider, F.B. (1998), "On concurrent programming", *Communications of the ACM*, Vol. 41 No. 4, p. 128, doi 10.1145/273035.273072.

Schuettler, M., Kohler, F., Ordonez, J.S. and Stieglitz, T. (2012), "Hermetic electronic packaging of an implantable brain-machine-interface with transcutaneous optical data communication", in *Proceedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS-2012), San Diego, CA, USA, August 28 - September 1, 2012*, IEEE press, pp. 3886–3889, doi 10.1109/EMBC.2012.6346816.

Schuettler, M., Stiess, S., King, B.V. and Suaning, G.J. (2005), "Fabrication of implantable microelectrode arrays by laser cutting of silicone rubber and platinum foil", *Journal of Neural Engineering*, Vol. 2 No. 1, pp. S121-S128, doi 10.1088/1741-2560/2/1/013.

Schwaber, K. (1995), "SCRUM Development Process", in Wirfs-Brock, R. (Ed.), *Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA-95, Austin, Texas, USA, October 15-19, 1995*, ACM, New York, NY, pp. 117–134.

Schwartz, A.B., Cui, X.T., Weber, D.J. and Moran, D.W. (2006), "Brain-controlled interfaces: movement restoration with neural prosthetics", *Neuron*, Vol. 52 No. 1, pp. 205–220, doi 10.1016/j.neuron.2006.09.019.

Shafquat, A. (2011), "Design and Validation of Chronic Research Tools for an Implantable Closed-Loop Neurostimulator", Master, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2011.

Shih, J.J., Krusienski, D.J. and Wolpaw, J.R. (2012), "Brain-Computer Interfaces in Medicine", *Mayo Clinic Proceedings*, Vol. 87 No. 3, pp. 268–279, doi 10.1016/j.mayocp.2011.12.008.

Shpigelman, L., Lalazar, H. and Vaadia, E. (2009), "Kernel-ARMA for Hand Tracking and Brain-Machine interfacing During 3D Motor Control", in Koller, D. (Ed.), *Advances in Neural Information Processing Systems 21: 22nd Annual Conference on Neural Information Processing Systems 2008, Vancouver, British Columbia, Canada, December 8-10, 2008*, Curran Associates Inc, pp. 1489–1496.

Silvoni, S., Cavinato, M., Volpato, C., Cisotto, G., Clara, G., Agostini, M., Turolla, A., Ramos-Murguialday, A. and Piccione, F. (2013), "Kinematic and neurophysiological consequences of an assisted-force-feedback brain-machine interface training: a case study", *Frontiers in Neurology*, Vol. 4, doi 10.3389/fneur.2013.00173.

Sitaram, R., Caria, A., Veit, R., Gaber, T., Rota, G., Kübler, A. and Birbaumer, N. (2007), "fMRI Brain-Computer Interface: A Tool for Neuroscientific Research and Treatment", *Computational Intelligence and Neuroscience*, Vol. 2007 No. 5, p. 25487, doi 10.1155/2007/25487.

Slutzky, M.W., Jordan, L.R., Krieg, T., Chen, M., Mogul, D.J. and Miller, L.E. (2010), "Optimal spacing of surface electrode arrays for brain-machine interface applications", *Journal of Neural Engineering*, Vol. 7 No. 2, p. 26004, doi 10.1088/1741-2560/7/2/026004.

Stallings, W. (2012), *Operating systems: Internals and design principles,* 7th ed., Prentice-Hall, Inc, Boston.

Stanslaski, S.R., Afshar, P., Cong, P., Giftakis, J., Stypulkowski, P.H., Carlson, D., Linde, D., Ullestad, D., Avestruz, A.-T. and Denison, T.J. (2012), "Design and validation of a fully implantable, chronic, closed-loop neuromodulation device with concurrent sensing and stimulation", *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 20 No. 4, pp. 410–421, doi 10.1109/TNSRE.2012.2183617.

Stanslaski, S.R., Cong, P., Carlson, D., Santa, W., Jensen, R., Molnar, G., Marks, W.J., Shafquat, A. and Denison, T.J. (2009), "An implantable bi-directional brain-machine interface system for chronic neuroprosthesis research", in *Proceedings of the 31st Annual International Conference of the IEEE*

Engineering in Medicine and Biology Society (EMBS-2009), Minneapolis, Minnesota, USA, September 2-6. 2009, IEEE press, pp. 5494–5497, doi 10.1109/IEMBS.2009.5334562.

Stieglitz, T., Rubehn, B., Henle, C., Kisban, S., Herwik, S., Ruther, P. and Schuettler, M. (2009), "Brain–computer interfaces: an overview of the hardware to record neural signals from the cortex", in Verhaagen, J., Hol, E.M., Huitenga, I., Wijnholds, J., Bergen, A.B., Boer, G.J. and Swaab, D.F. (Eds.), *Neurotherapy: Progress in Restorative Neuroscience and Neurology*, *Progress in Brain Research*, Vol. 175, Elsevier Science, pp. 297–315, doi 10.1016/S0079-6123(09)17521-0.

Stroustrup, B. (2000), *The C++ programming language: Special Edition,* 3rd ed., Addison-Wesley, Boston.

Stypulkowski, P.H., Stanslaski, S.R., Denison, T.J. and Giftakis, J.E. (2013), "Chronic Evaluation of a Clinical System for Deep Brain Stimulation and Recording of Neural Network Activity", *Stereotactic and Functional Neurosurgery*, Vol. 91 No. 4, pp. 220–232, doi 10.1159/000345493.

Suchodoletz, D. von, Rechert, K., Valizada, I. and Strauch, A. (2013), "Emulation as an Alternative Preservation Strategy - Use-Cases, Tools and Lessons Learned", in Horbach, M. (Ed.), *Informatik 2013: Informatik angepasst an Mensch, Organisation und Umwelt, Koblenz, Germany, September 16-20, 2013*, Köllen, Bonn, pp. 592–606.

Suchodoletz, D. von, Rechert, K. and van der Werf, Bram (2012), "Long-term Preservation in the Digital Age – Emulation as a Generic Preservation Strategy", *PIK - Praxis der Informationsverarbeitung und Kommunikation*, Vol. 35 No. 4, pp. 225–226, doi 10.1515/pik-2012-0051.

Sun, F.T., Morrell, M.J. and Wharen, R.E., Jr. (2008), "Responsive cortical stimulation for the treatment of epilepsy", *Neurotherapeutics*, Vol. 5 No. 1, pp. 68–74, doi 10.1016/j.nurt.2007.10.069.

Tabot, G.A., Dammann, J.F., Berg, J.A., Tenore, F.V., Boback, J.L., Vogelstein, R.J. and Bensmaia, S.J. (2013), "Restoring the sense of touch with a prosthetic hand through a brain interface", *Proceedings of the National Academy of Sciences*, Vol. 110 No. 45, pp. 18279–18284, doi 10.1073/pnas.1221113110.

Taylor, D.M., Tillery, Stephen I Helms and Schwartz, A.B. (2002), "Direct cortical control of 3D neuroprosthetic devices", *Science*, Vol. 296 No. 5574, pp. 1829–1832, doi 10.1126/science.1070291.

Tsubokawa, T., Katayama, Y. and Yamamoto, T. (1985), "Deafferentiation pain and stimulation of thalamic sensory relay nucleus: clinical and experimental study", *Applied Neurophysiology*, Vol. 48 1-6, pp. 166–171.

Tsubokawa, T., Katayama, Y., Yamamoto, T., Hirayama, T. and Koyama, S. (1991), "Chronic Motor Cortex Stimulation for the Treatment of Central Pain", in Hitchcock, E.R., Broggi, G., Burzaco, J., Martin-Rodriguez, J., Meyerson, B.A. and Toth, S. (Eds.), *Advances in Stereotactic and Functional Neurosurgery 9*, *Acta Neurochirurgica Supplementum*, Vol. 52, Springer-Verlag, pp. 137–139, doi 10.1007/978-3-7091-9160-6_37.

Tsubokawa, T., Katayama, Y., Yamamoto, T., Hirayama, T. and Koyama, S. (1993), "Chronic motor cortex stimulation in patients with thalamic pain", *Journal of Neurosurgery*, Vol. 78 No. 3, pp. 393–401, doi 10.3171/jns.1993.78.3.0393.

U.S. Government, "ClinicalTrials.gov. A service of the U.S. National Institutes of Health", available at: http://clinicaltrials.gov (accessed 1 August 2014).

Unger, S.H. (1995), "Hazards, critical races, and metastability", *IEEE Transactions on Computers*, Vol. 44 No. 6, pp. 754–768, doi 10.1109/12.391185.

Vapnik, V.N. and Chervonenkis, A.Y. (1974), *Theory of Pattern Recognition [in Russian]*, Nauka.

Velliste, M., Perel, S., Spalding, M.C., Whitford, A.S. and Schwartz, A.B. (2008), “Cortical control of a prosthetic arm for self-feeding”, *Nature*, Vol. 453 No. 7198, pp. 1098–1101, doi 10.1038/nature06996.

Venkatraman, S. and Carmena, J.M. (2011), “Active Sensing of Target Location Encoded by Cortical Microstimulation”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 19 No. 3, pp. 317–324, doi 10.1109/TNSRE.2011.2117441.

Vidal, J.J. (1973), “Toward direct brain-computer communication”, *Annual Review of Biophysics and Bioengineering*, Vol. 2, pp. 157–180, doi 10.1146/annurev.bb.02.060173.001105.

Vidaurre, C., Sander, T.H. and Schlögl, A. (2011), “BioSig: the free and open source software library for biomedical signal processing”, *Computational Intelligence and Neuroscience*, Vol. 2011, p. 935364, doi 10.1155/2011/935364.

Waldert, S., Pistohl, T., Braun, C., Ball, T., Aertsen, A. and Mehring, C. (2009), “A review on directional information in neural signals for brain-machine interfaces”, *Journal of Physiology - Paris*, Vol. 103 3-5, pp. 244–254, doi 10.1016/j.jphysparis.2009.08.007.

Walter, A., Murguialday, A.R., Spüler, M., Naros, G., Leão, M.T., Gharabaghi, A., Rosenstiel, W., Birbaumer, N. and Bogdan, M. (2012a), “Coupling BCI and cortical stimulation for brain-state-dependent stimulation: methods for spectral estimation in the presence of stimulation after-effects”, *Frontiers in Neural Circuits*, Vol. 6, doi 10.3389/fncir.2012.00087.

Walter, A., Naros, G., Roth, A., Rosenstiel, W., Gharabaghi, A. and Bogdan, M. (2012b), “A brain-computer interface for chronic pain patients using epidural ECoG and visual feedback”, in *2012 IEEE 12th International Conference on Bioinformatics & Bioengineering (BIBE), Larnaca, Cyprus, November 11-13, 2012*, IEEE press, pp. 380–385, doi 10.1109/BIBE.2012.6399654.

Wang, W., Collinger, J.L., Degenhart, A.D., Tyler-Kabara, E.C., Schwartz, A.B., Moran, D.W., Weber, D.J., Wodlinger, B., Vinjamuri, R.K., Ashmore, R.C., Kelly, J.W. and Boninger, M.L. (2013), “An Electrocorticographic Brain Interface in an Individual with Tetraplegia”, *PLoS ONE*, Vol. 8 No. 2, pp. E55344, doi 10.1371/journal.pone.0055344.

Wang, W., Collinger, J.L., Perez, M.A., Tyler-Kabara, E.C., Cohen, L.G., Birbaumer, N., Brose, S.W., Schwartz, A.B., Boninger, M.L. and Weber, D.J. (2010), “Neural Interface Technology for Rehabilitation: Exploiting and Promoting Neuroplasticity”, *Physical Medicine and Rehabilitation Clinics of North America*, Vol. 21 No. 1, pp. 157–178, doi 10.1016/j.pmr.2009.07.003.

Williams, J.C., Hippensteel, J.A., Dilgen, J., Shain, W. and Kipke, D.R. (2007), “Complex impedance spectroscopy for monitoring tissue responses to inserted neural implants”, *Journal of Neural Engineering*, Vol. 4 No. 4, pp. 410–423, doi 10.1088/1741-2560/4/4/007.

Wilson, J.A. and Williams, J.C. (2009), “Massively Parallel Signal Processing using the Graphics Processing Unit for Real-Time Brain-Computer Interface Feature Extraction”, *Frontiers in Neuroengineering*, Vol. 2, doi 10.3389/neuro.16.011.2009.

Wilson, P.R., Johnstone, M.S., Neely, M. and Boles, D. (1995), “Dynamic storage allocation: A survey and critical review”, in Baker, H.G. (Ed.), *Proceedings of the International Workshop on Memory Management (IWMM ’95), Kinross, UK, September 27-29, 1995*, Springer-Verlag, London, UK, pp. 1–116, doi 10.1007/3-540-60368-9_19.

Wolpaw, J.R., Birbaumer, N., McFarland, D.J., Pfurtscheller, G. and Vaughan, T.M. (2002), “Brain-computer interfaces for communication and control”, *Clinical Neurophysiology*, Vol. 113 No. 6, pp. 767–791, doi 10.1016/S1388-2457(02)00057-3.

Wolpaw, J.R. and McFarland, D.J. (2004), “Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans”, *Proceedings of the National Academy of Sciences*, Vol. 101 No. 51, pp. 17849–17854, doi 10.1073/pnas.0403504101.

Wong, C.H., Birkett, J., Byth, K., Dexter, M., Somerville, E., Gill, D., Chaseling, R., Fearnside, M. and Bleasel, A. (2009), “Risk factors for complications during intracranial electrode recording in presurgical evaluation of drug resistant partial epilepsy”, *Acta Neurochirurgica*, Vol. 151 No. 1, pp. 37–50, doi 10.1007/s00701-008-0171-7.

Woon, W.L. and Cichocki, A. (2007), “Novel features for brain-computer interfaces”, *Computational Intelligence and Neuroscience*, Vol. 2007, p. 82827, doi 10.1155/2007/82827.

Wu, W. and Hatsopoulos, N.G. (2008), “Real-Time Decoding of Nonstationary Neural Activity in Motor Cortex”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 16 No. 3, pp. 213–222, doi 10.1109/TNSRE.2008.922679.

Zaytsev, Y.V. and Morrison, A. (2012), “Increasing quality and managing complexity in neuroinformatics software development with continuous integration”, *Frontiers in Neuroinformatics*, Vol. 6, doi 10.3389/fninf.2012.00031.

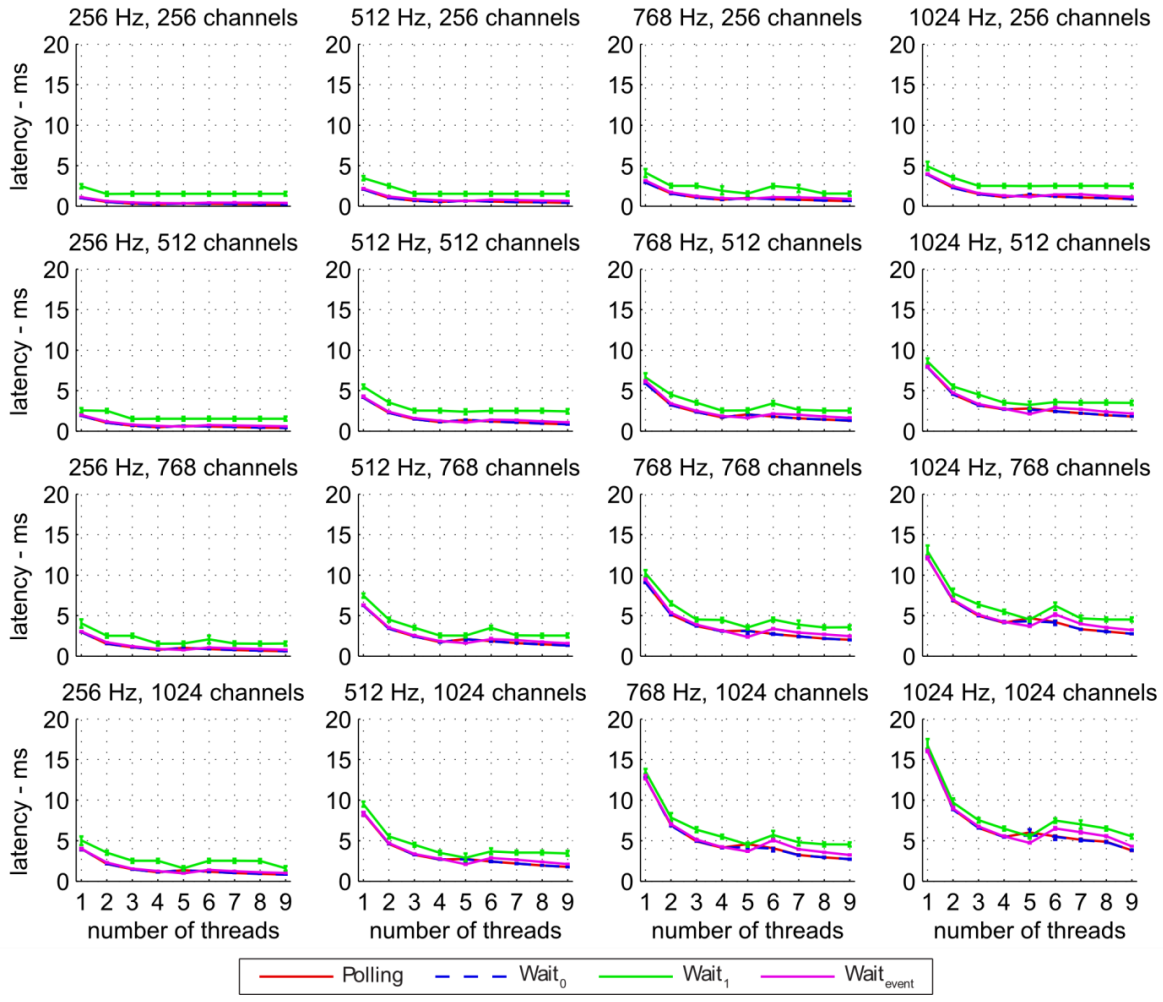
Zuo, C., Yang, X., Wang, Y., Hagains, C.E., Li, A.-L., Peng, Y.B. and Chiao, J.-C. (2012), “A digital wireless system for closed-loop inhibition of nociceptive signals”, *Journal of Neural Engineering*, Vol. 9 No. 5, p. 56010, doi 10.1088/1741-2560/9/5/056010.

10 Appendix

10.1 Supplementary Materials

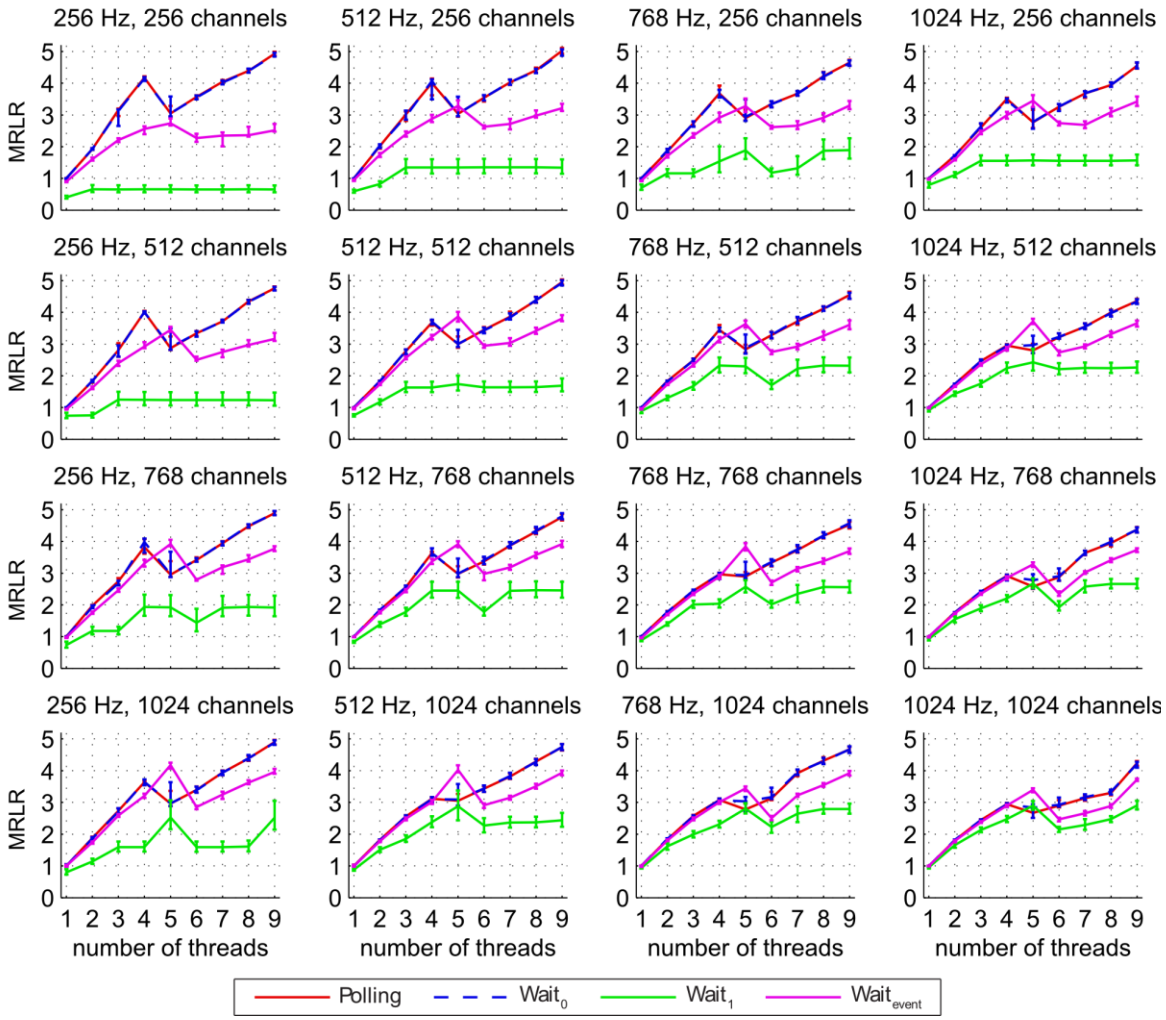
The following Supplementary Figures i-iii depict median latencies, MRLR and CPU load as a function of the number of threads for different sampling frequencies (256Hz-1024Hz) and different number of channels (256-1024). All graphs show the results for the short-time Fourier transform algorithm. Independent of sampling rate and channel count, the performance increased with increasing number of threads. Additionally, for all investigated sampling frequencies and number of channels the waiting strategy $Wait_{event}$ yielded the best compromise between low-latency and low CPU usage. Hence, these figures demonstrate that the results presented in section 7.1 are valid for a wide range of different sampling frequencies and number of channels. Supplementary Figure iv demonstrates that the computing times of support vector machines (SVM; for classification) are very similar to the computing times of support vector regression (SVR; for regression) shown in Figure 21 D–F.

Supplementary Figure v shows the mean power spectral densities of ECoG recordings from sheep A1 and A2 for individual channels using either the Braincon implant or the gUSBamp device. For each sheep and each amplifier, the PSDs of the individual channels were similar and there was no outlier. Therefore it is sufficient for both recording devices to compare the PSDs over all channels for signal quality assessment (cf. Figure 24 and Figure 27).



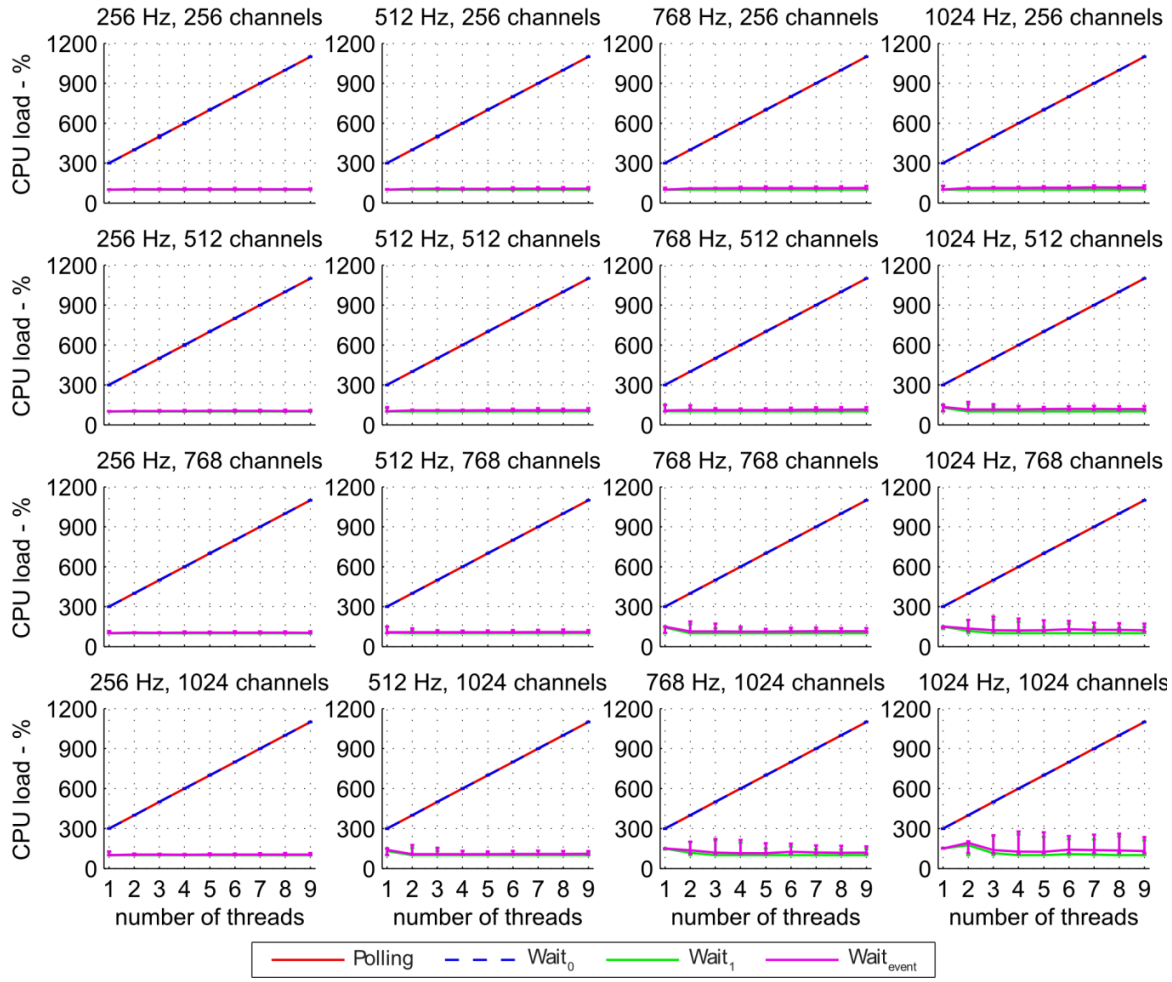
Supplementary figure i: Latency of the filter pipeline for short-time Fourier transform

Latency of the filter pipeline implementing short-time Fourier transform algorithm for different waiting strategies and numbers of threads. Each subplot shows the median of latencies (lines) with 25% and 75% percentiles (error bars) for one combination of sampling frequency and number of channels. Figure from Fischer et al. (2014).



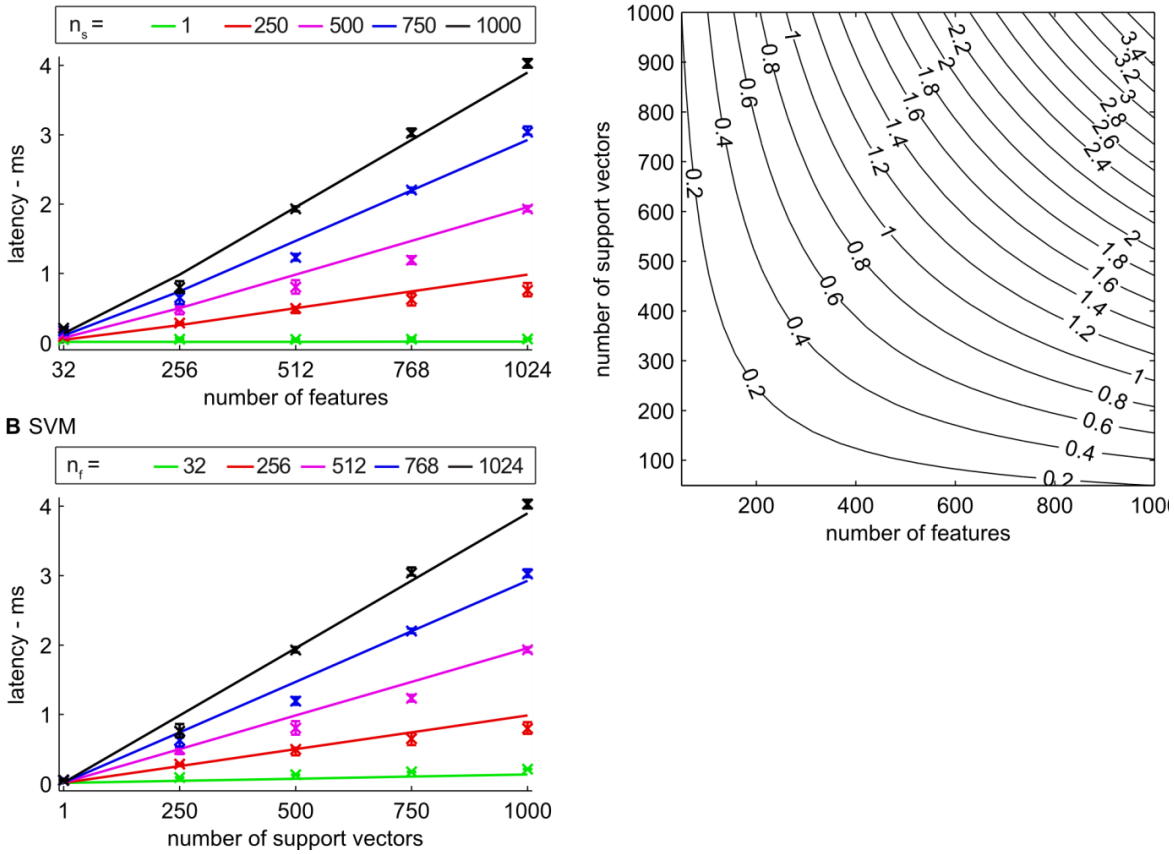
Supplementary figure ii: MRLR of the filter pipeline for short-time Fourier transform

MRLR for the filter pipeline implementing the short-time Fourier transform algorithm for different waiting strategies and numbers of threads. Each subplot shows MRLR (lines) with 25% and 75% percentiles (error bars) for one combination of sampling frequency and number of channels. Figure from Fischer et al. (2014).



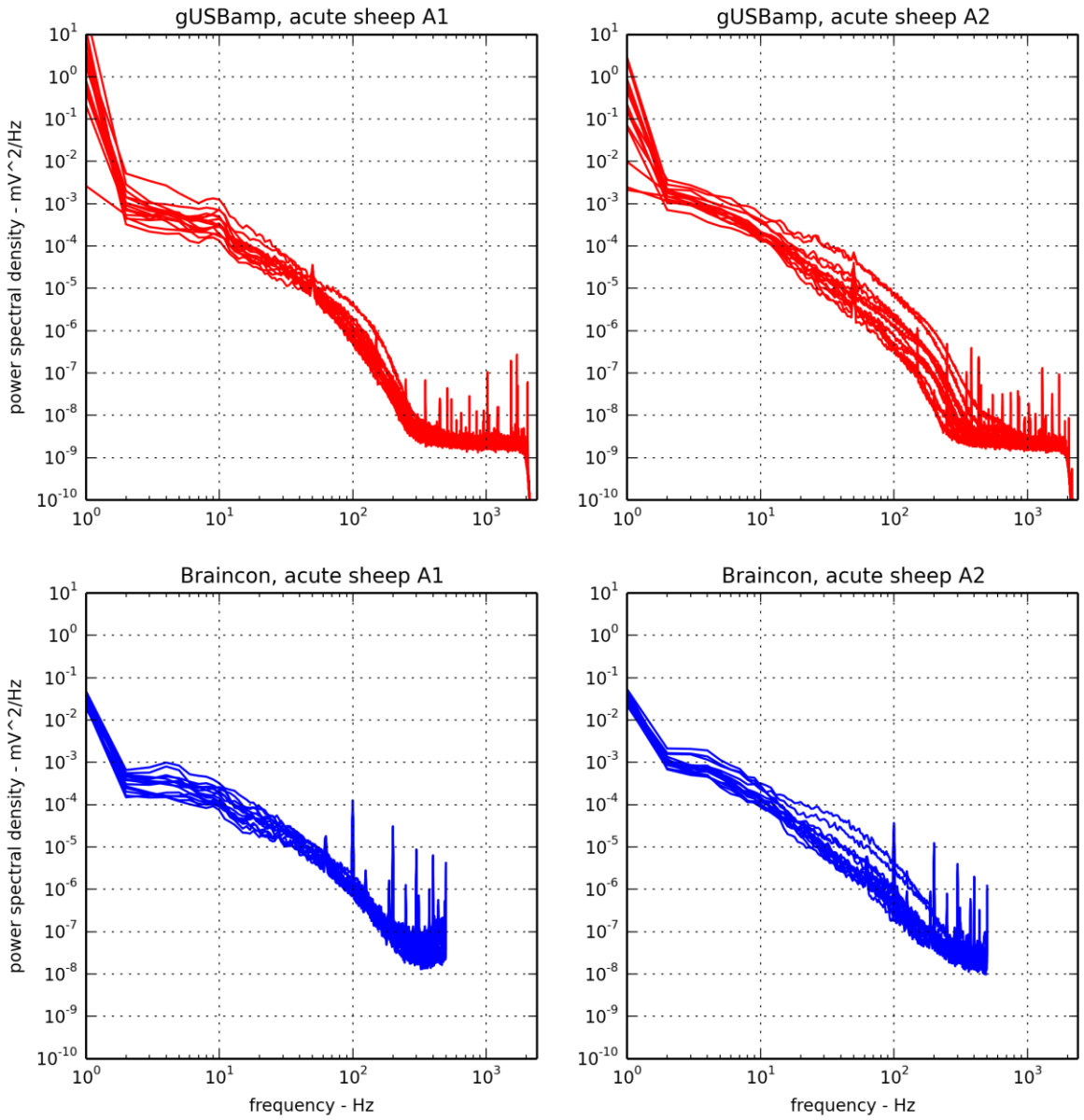
Supplementary figure iii: CPU load of test computer running data source simulator

CPU load of the test computer running data source simulator module that receives the processed data and the filter pipeline implementing the short-time Fourier transform algorithm for different waiting strategies and numbers of threads. Each subplot shows the median of CPU load (lines) with 25% and 75% percentiles (error bars) for one combination of sampling frequency and number of channels. Figure from Fischer et al. (2014).



Supplementary figure iv: Latencies for the filter pipeline implementing the SVM algorithm

Markers show the median of measured latencies with error bars showing the 25% and 75% percentiles. Solid lines show latencies predicted by algorithm specific models. Models in (A) used the SVM algorithm with 1, 250, 500, 750, and 1000 support vectors. Models in (B) used the SVM algorithm with 32, 256, 512, 768, and 1024 features as input. (C) Model prediction of SVM latencies in milliseconds (numbers on the solid lines) as a function of number of features and number of support vectors. Model assumes a linear increase of latency with the product of n_s and n_f. Figure from Fischer et al. (2014).



Supplementary figure v: Mean power spectral density for individual channels

Mean power spectral densities of individual recording channels from A1 and A2, measured either with the Braincon implant or the gUSBamp device. PSDs were calculated from 100 seconds of recordings while sheep were anesthetized. For the Braincon implant, one channel was excluded due to lead fracture.

10.2 List of Figures

10.3 List of Tables

10.4 List of Abbreviations

AIMD *active implantable medical device directive*
ALS *amyotrophic lateral sclerosis*
ASIC *application-specific integrated circuit*
BBCI *Berlin BCI*
BCI *brain-computer interface*
BPO *Braincon implant processing object*
CI *confidence interval, continuous integration*
CT *computer tomography*
DBS *deep brain stimulation*
DC *direct current*
DOF *degree of freedom*
ECoG *electrocorticography*
EEG *electroencephalogram*
EPA *evoked potential analyzer*
FDA *food and drug administration of the united states*
fMRI *functional magnetic resonance imaging*
FPGA *field-programmable gate arrays*
GB *gigabyte*
GHz *gigahertz*
GPO *gUSBamp amplifier processing object*
GPU *graphics processing unit*
GUI graphical user interface
HIM *hardware interface module*
INVITRO *in vitro diagnostic medical device directive*
KF *Kalman filter*
LDA linear discriminant analysis
LFP *local field potential*
MB *megabyte*
MDD *medical device directive*
MEA *microelectrode array*
MEG *magnetoencephalography*
MRLR *median of relative latency reduction*
MUA *multi unit activity*
NIRS *near-infrared spectroscopy*
PC *personal computer*
PCB *printed circuit board*
PO *processing object*
PSD *power spectral density*
RLDA *regularized linear discriminant analysis*
RLR *relative latency reduction*
SEP *somatosensory evoked potential*
SOUP *software of unknown provenance*
SSC *stimulation script controller*
SSE *stimulation script editor*
SUA *single unit activity*
SVM *support vector machine*
SVR *support vector regression*
TCP/IP *transmission control protocol and internet protocol*
TMS *transcranial magnetic stimulation*

10.5 Figure Copyright

Figure 3: Caption and figure taken from Kohler et al., 2012; Copyright © 2012 IEEE. Permission to reuse obtained from IEEE via Copyright Clearance Center's RightsLink service.

Figure 6: Caption and figure taken from Kohler et al., 2013; Copyright © 2013 IEEE. Permission to reuse obtained from IEEE via Copyright Clearance Center's RightsLink service.

Parts of Figure 19 (i.e., the electrode layout scheme in the background) was designed and manufactured by Christian Henle.

Figures 13, 15, 20, 21, 22 and Supplementary Figures i, ii, iii, iv taken from Fischer *et al.* (2014) under the Creative Commons Public License (http://creativecommons.org/licenses/by/3.0/).

Figures 17-18 and 23-29 taken from Kohler *et al.*. Copyright remains with the authors of this manuscript.

10.6 Funding

This work was funded by the German Federal Ministry of Education and Research (BMBF) grant BMBF GoBio grant 0313891 and KMU-Innovativ.

10.7 List of Publications

Journal Publications

Fischer, J., Milekovic, T., Schneider, G. and Mehring, C. (2014), "Low-latency multi-threaded processing of neuronal signals for brain-computer interfaces", *Frontiers in Neuroengineering*, Vol. 7, p. 1, doi 10.3389/fneng.2014.00001.

Kohler, F., Fischer, J., Gierthmuehlen, M., Gkogkidis, A., Henle, C., Ball, T., Wang, X., Rickert, J., Stieglitz, T. and Schuettler, M., *Long-term in vivo validation of a fully-implantable, wireless brain-computer interface for cortical recording and stimulation*, in preparation.

Milekovic, T., Fischer, J., Pistohl, T., Ruescher, J., Schulze-Bonhage, A., Aertsen, A., Rickert, J., Ball, T. and Mehring, C. (2012), "An online brain-machine interface using decoding of movement direction from the human electrocorticogram", *Journal of Neural Engineering*, Vol. 9 No. 4, p. 46003, doi 10.1088/1741-2560/9/4/046003.

Hammer, J., Fischer, J., Ruescher, J., Schulze-Bonhage, A., Aertsen, A. and Ball, T. (2013), "The role of ECoG magnitude and phase in decoding position, velocity, and acceleration during continuous motor behavior", *Frontiers in Neuroscience*, Vol. 7, doi 10.3389/fnins.2013.00200.

Gierthmuehlen, M., Wang, X., Gkogkidis, A., Henle, C., Fischer, J., Fehrenbacher, T., Kohler, F., Raab, M., Mader, I., Kuehn, C., Foerster, K., Haberstroh, J., Freiman, T.M., Stieglitz, T., Rickert, J., Schuettler, M. and Ball, T. (2014), "Mapping of sheep sensory cortex with a novel micro-electrocorticography grid [epub ahead of print]", *Journal of Comparative Neurology*, doi 10.1002/cne.23631.

Conference Contributions

Fischer, J., Henle, C., Paetzold, J., Mohrlok, R., Raab, M., Moeller, A., Rickert, J. and Schuettler, M. (2011), "BRAINCON – A wireless implantable system for long-term recording of electrocorticogram signals and electrical stimulation", poster presented at 45. DGBMT Jahrestagung (BMT 2011), September 27-30, 2011, Freiburg, Germany, doi 10.1515/BMT.2011.859.

Fischer, J., Milekovic, T., Mehring, C., Hammer, J., Rickert, J., Aertsen, A. and Ball, T. (2013), "The Braincon platform software architecture for invasive closed-loop Brain-Machine Interfaces and electrical stimulation", poster presented at SfN 2013, November 9-13, 2013, San Diego, California, USA.