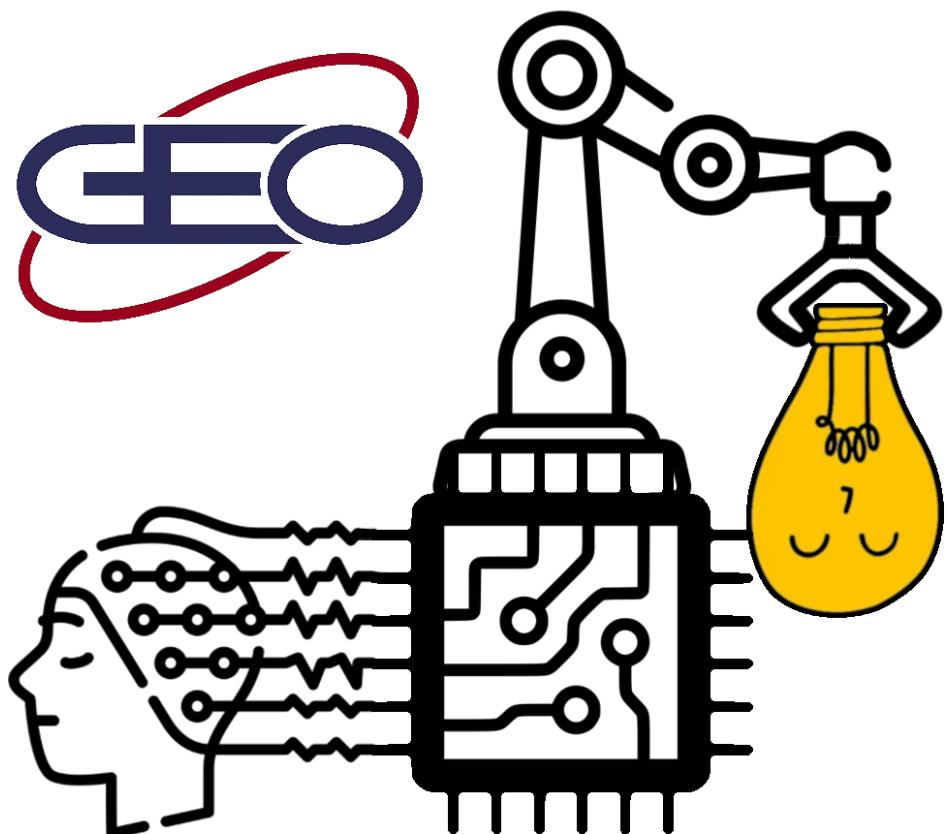


Steuerung per Lidschluss

Erkennung ereigniskorrelierter Potenziale eines
Elektroenzephalogramms durch eine KI



Alexander Reimer und Matteo Friedrich
Gymnasium Eversten Oldenburg

Inhaltsverzeichnis

1 Kurzfassung	1
2 Einleitung	1
3 Methode und Vorgehensweise	1
3.1 Materialien	1
3.2 Vorgehensweise	2
3.2.1 Elektroenzephalographie	3
3.2.2 Fourier-Analyse	5
3.2.3 Neuronales Netz	6
3.2.4 Roboter	9
3.2.5 Übersicht des Ablaufs unseres Programmes in einem Diagramm	11
4 Ergebnisse	11
5 Diskussion	14
6 Danksagung	16
6.1 Finanzierung	16
6.2 Unterstützung	16
7 Quellen & Referenzen	16
7.1 Abbildungen	16
7.2 Literatur	16
7.3 Videos	17
7.4 Programme	17

1 Kurzfassung

Brain-Computer-Interfaces (BCIs) werden in den Medien immer präsenter, meist in Form von Durchbrüchen durch Universitäten, Institute oder große Unternehmen. Wir haben uns vorgenommen, mit günstiger Hardware und eigener Software selbst ein BCI zu entwickeln. Unser BCI soll anderen Leuten die Entwicklung von BCIs leichter machen, und dafür nicht an einen bestimmten Zweck gebunden sein.

2 Einleitung

Ziel des Projektes ist es, zuerst ein BCI – Brain-Computer-Interface – zu entwickeln, welches verschiedene Ereignisse erkennen kann – momentan, ob man gerade blinzelt. Diese Erkennung wollen wir dann zur Steuerung eines Roboters nutzen.

BCIs sind Schnittstellen zwischen dem Gehirn und einem Computer, die eine Kommunikation vom Gehirn zum Computer ermöglichen. Man kann BCIs für die Steuerung von Prothesen, Drohnen, Robotern, und vielem mehr nutzen. [1] Zum Beispiel haben Forscher aus Stanford ein BCI entwickelt, welches in der Lage ist, behinderten Personen die Möglichkeit zu geben, über ihre Gedanken Nachrichten zu schreiben und so kommunizieren zu können. [2] Andere Forscher haben es geschafft, Menschen mit vollständigem Locked-in-Syndrom (CLIS) mithilfe eines BCIs wieder die Kommunikation zu ermöglichen, trotz Fehlen von jeglicher willentlicher Muskelbewegungen. [3]

Wir hoffen, bei der Erarbeitung unseres Projektes neben dem Erlangen von Erfahrung in diesem interessanten Bereich auch selbst dazu beizutragen. BCIs, die mit sehr teurer Hardware entwickelt werden, sind bereits gut erforscht. Jedoch sind diese BCIs aufgrund ihres hohen Preises für viele Konsumenten nicht interessant. Genau bei diesem Problem wollen wir mit unserem Projekt ansetzen: Wir wollen ein BCI entwickeln, welches auch mit günstiger Hardware funktioniert.

Dabei ist es uns wichtig, dass unser BCI ohne spezielles Training auf eine bestimmte Person für jeden beliebigen Menschen funktioniert, und dass unser BCI auch leicht für die eigenen Zwecke anpassbar ist, z.B. die Umstellung von der Erkennung von Augenschließen zur Erkennung von Armbewegungen. Deswegen benutzen wir Open-Source-Software und haben unseren kompletten Code ebenfalls auf GitHub veröffentlicht [19].

3 Methode und Vorgehensweise

3.1 Materialien

Alle mit ★ gekennzeichneten Materialien wurden von der Gesellschaft der Freunde des Gymnasium Eversten e.V. gesponsort (s. Danksagung).

- EEG
 - 4 Channel Ganglion Board von OpenBCI ★
 - 4x Spike-Elektroden und 2x flache Elektroden ★
 - Klettband für die Elektroden ★
 - 2x Ohrclips ★
 - Lithium-Polymer-Akku und Ladegerät ★
 - Plastik-Hülle für das Ganglion Board
- Roboter
 - Lego Mindstorms EV3 Brick
 - Raspberry Pi 3B+ 1 GB

- 2x EV3 großer Motor
- SD-Karte (8 GB)
- diverse LEGO Teile
- Computer: Aorus 15P
 - CPU: i7-11800H
 - GPU: RTX 3060
 - RAM: 16 GB
- Software
 - Julia als Programmiersprache für unser ganzes Projekt [20]
 - Flux.jl für das neuronale Netz [21] [22]
 - BrainFlow.jl als Schnittstelle zum EEG [23]
 - FFTW.jl für die Fast Fourier Transformation [24]
 - CUDA.jl zum effektiven Nutzen einer NVIDIA GPU [25]
 - PyPlot.jl zum Plotten [26]
 - BSON.jl zum Speichern und Laden von Netzwerken
 - ev3dev als Betriebssystem auf dem EV3-Brick [4]
 - ev3dev.jl zum Steuern eines EV3 Roboters durch einen Raspberry Pi in Julia [27]

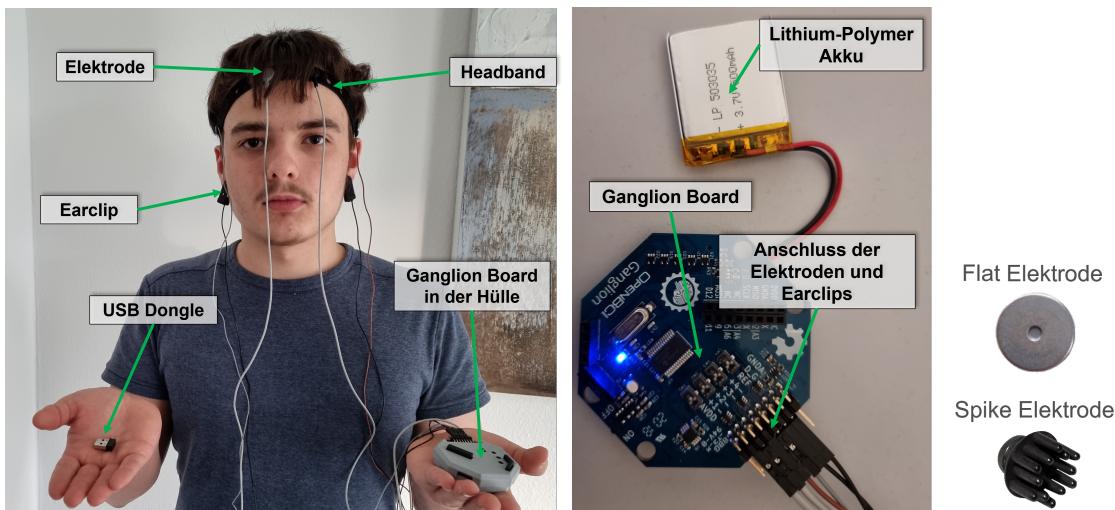


Abbildung 1: Das EEG und Zubehör. Links sieht man die typische Anwendung des EEG im Versuchsaufbau (Konfiguration 1, siehe Abbildung 3). In der Mitte ist das Ganglion Board genauer dargestellt. Rechts sind eine flache Elektrode und eine Spike-Elektrode zu sehen.

Unser EEG-Gerät besteht aus einem Klettband mit Löchern für die Elektroden, einer Platine (Ganglion Board), in welche die Elektroden eingesteckt werden, einer Kunststoff-Hülle zum Schutz des Ganglion Boards, und einem USB-Dongle, mit welchem die Signale der Platine kabellos empfangen werden können (siehe Abbildung 1).

3.2 Vorgehensweise

Unser Projekt lässt sich grob in drei Bereiche unterteilen. Zum einen gibt es den neurobiologischen Teil. Dieser besteht aus der Messung von Gehirnaktivität und der Umwandlung dieser Aktivität in für uns nutzbare Daten. Der zweite Teil besteht aus der Verarbeitung dieser Signale. Hierfür

nutzen wir ein neuronales Netz, welches Muster in den Gehirnaktivitäten erkennen kann. Der letzte Teil von unserem Projekt beinhaltet die Steuerung eines EV3 Roboters. Je nachdem, was das neuronale Netz ausgibt, soll sich dieser Roboter anders verhalten und so über Gedanken steuerbar sein.

3.2.1 Elektroenzephalographie

Bei der Elektroenzephalographie (EEG) werden Elektroden an der Kopfoberfläche platziert. Diese können sehr kleine Spannungen messen, die durch Potentialänderungen in Neuronengruppen im Gehirn entstehen und durch den Schädel dringen. Eine Elektrode kann meist nur die Summe aller lokalen Potentialänderungen messen. Man kann also auch nur ungefähr sagen, wo genau im Gehirn eine Potentialänderung stattgefunden hat. Mit mehr Elektroden kann die Genauigkeit erhöht werden. [5] [6]

Um eine Differenz bilden zu können, wird eine Referenzelektrode benötigt. Wir befestigen diese am Ohrläppchen, wo sie ein von Gehirnströmen weitgehend unbeeinflusstes, konstantes Potential ableiten kann. Alle vier am Kopf platzierten Elektroden nutzen diese Referenzelektrode. Somit handelt es sich bei allen vier um eine unipolare Ableitung. [7]

Wir untersuchen ereigniskorrelierte Potentiale (EKPs). Dies sind bestimmte Spannungsschwankungen („Potentiale“), welche in Zusammenhang mit einem internen oder externen Ereignis stehen, wie z.B. einem lauten Ton, einer Körperbewegung oder hoher Konzentration. [6] [7] Dabei haben wir uns zuerst für das Blinzeln entschieden, da wir es im Graphen zumindest mit bloßen Augen sehr gut erkennen konnten (siehe Ausschlag nach 600 ms in Abbildung 2).

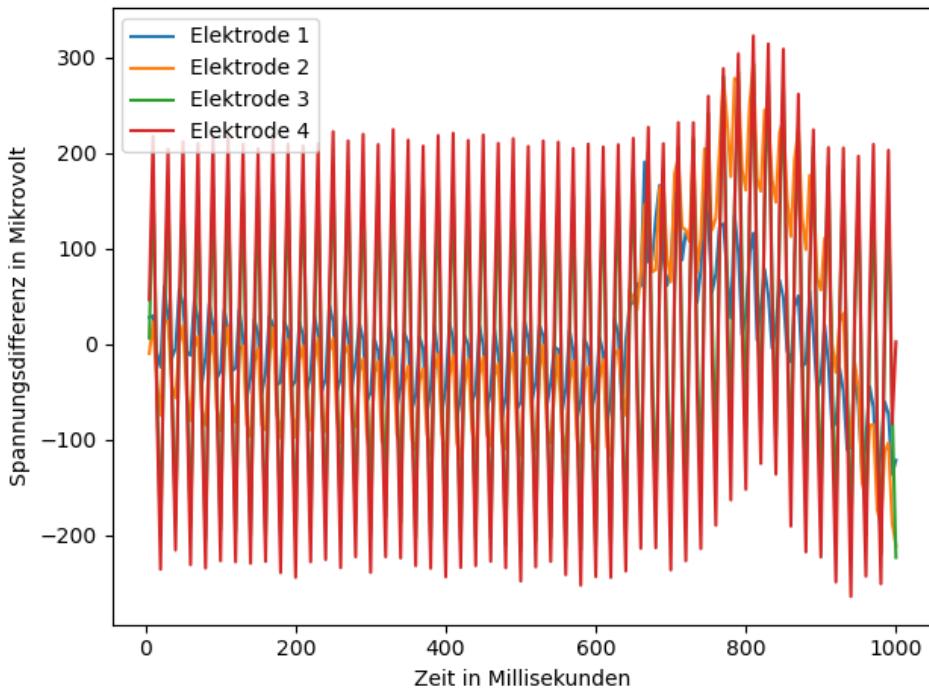


Abbildung 2: Ausschnitt eines EMG beim Blinzeln (Konfiguration 1).

Weiter ist es möglich, nur durch gezielte Gehirnaktivität eine Steuerung auszuführen. Dies funktioniert jedoch meist durch instrumentelle oder klassische Konditionierung, also durch das Bestrafen und Belohnen auf Basis der Messungen des EEG. So kann das Gehirn darauf trainiert werden, auf Verlangen eine bestimmte, vorher festgelegte Aktivität auszulösen, die dann gemessen und ausgewertet werden kann. [3] Darauf wollen wir verzichten, da die Konditionierung Zeit benötigt und nicht unser Ziel einer allgemeinen Anwendbarkeit erfüllen würde.

Zur Messung der EKPs haben wir zu Beginn zwei flache Elektroden (Elektroden mit glatter Kontaktobерfläche) über den Augen und zwei Spike-Elektroden (Elektroden mit langen „Spitzen“)

an den Schläfen platziert (siehe Abbildung 3). Flache Elektroden haben zwar aufgrund ihrer größeren Oberfläche auf freier Haut eine etwas bessere Signalqualität, aber die von uns gewählten Stellen an den Schläfen waren leicht behaart. Die Spike-Elektroden können durch die Haare leichter Kontakt zur Kopfoberfläche herstellen.

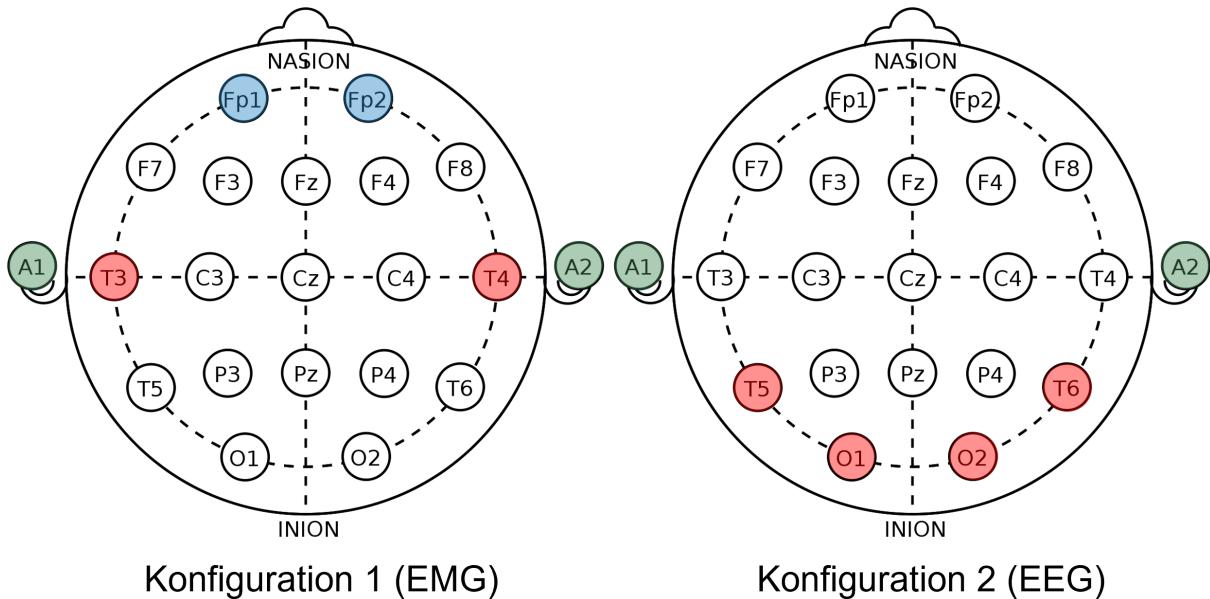


Abbildung 3: 10-20 System mit den von uns genutzten flachen Elektroden blau, Spike-Elektroden rot, und Erdung- & Referenzohrclip grün gefärbt. Links für EMG („Konfiguration 1“), rechts für EEG („Konfiguration 2“).

Jedoch handelt es sich hier eigentlich nicht um ein EEG. Der Grund für den so starken Ausschlag ist, dass wir Elektromyographie (EMG) nutzten. Dabei werden die Spannungsdifferenzen von Muskeln gemessen, in unserem Fall die Muskeln, welche das Augenlid bewegen. [8] Aber wir haben EMG trotzdem zu Beginn genutzt, da wir erstmal eine funktionierende Basis schaffen wollten.

Nachdem die Analyse der EMG funktioniert hat, haben wir dann tatsächliche Elektroenzephalographie genutzt. Dabei haben wir weiterhin Blinzeln untersucht. Die Elektroden haben wir dafür alle nebeneinander am Okzipitalen Kortex (Visueller Kortex) platziert (siehe Abbildung 3 rechts), welcher sich am Hinterkopf befindet und sehr stark mit den Augen und dem Sehvermögen verknüpft ist. [6] Wir entschieden uns, trotz des Umstiegs auf EEG bei Blinzeln als EKP zu bleiben, da wir bereits wussten, dass geschlossene Augen klar erkennbar sein sollten: Bei offenen Augen findet eine sogenannte Alpha-Blockade statt, bei der – vor allem im Okzipitalem Kortex – die Alphawellen (ca. 7-13 Hertz) stark sinken. Entsprechend steigen sie stark an, wenn die Augen geschlossen sind. Dieser Effekt wird auch Berger-Effekt genannt. [9] [7] [10]

Uns ist nach der Aufnahme der Trainingsdaten in der ersten Konfiguration (EMG) aufgefallen, dass bei den Elektroden 3 und 4 das Blinzeln deutlich schlechter erkennbar war als bei den Elektroden 1 und 2 (siehe Abbildung 2), vermutlich weil die Elektroden 3 und 4 Spike-Elektroden und weiter von den Augen entfernt waren. Deswegen haben wir uns für die weitere Analyse entschieden, nur die Elektroden 1 und 2 zu verwenden. Wir haben immer eine Sekunde an EMG oder EEG-Daten am Stück aufgenommen, sowohl für die Trainingsdaten als auch die Live-Tests. Der Grund für diese Entscheidung ist, dass die Auswirkung von Blinzeln immer in diesem Rahmen erkennbar ist. Ein kürzeres Zeitintervall hätte im Prinzip auch funktioniert, jedoch ist die Fourier-Analyse (siehe unten) effektiver, je länger das Zeitintervall ist. Da jede Elektrode eine zeitliche Auflösung von 200 Hertz (200 Messungen pro Sekunde) hat, messen wir also in der ersten Elektrodenkonfiguration (EMG) $2 * 200 = 400$ und in der zweiten Elektrodenkonfiguration (EEG) $4 * 200 = 800$ Signale pro Sekunde.

3.2.2 Fourier-Analyse

Uns wurde von Professor Everling (s. Danksagung – Unterstützung) empfohlen, die Anwendung der Fourier-Analyse zur Vorbereitung der Daten für das neuronale Netz zu bedenken. Die Fourier-Analyse kann die verschiedenen zugrundeliegenden Frequenzen von Datenfolgen, Funktionen, und mehr bestimmen, indem diese in Sinus-Kurven zerlegt werden, sie dient also zur Spektralanalyse. [13] Wir nutzen dafür die Fast Fourier Transformation (FFT), welche lediglich eine komplexere aber effizientere Form der Diskreten Fourier Transformation (DFT) ist. [11]

Aus der FFT folgt ein Array (eine Liste) an komplexen Zahlen. Der Index eines Wertes in der Liste bestimmt, für welche Frequenz der Wert gilt (erster Wert: 1 Hertz, zweiter Wert: 2 Hertz, etc.). Nun muss für jede komplexe Zahl der Abstand zum Ursprung bestimmt werden, also der absolute Betrag. Dieser entspricht dann der Amplitude der Frequenz. So lässt sich bestimmen, welche Frequenzen am stärksten vorkommen. Außerdem können dann Frequenzen herausgefiltert werden, indem die Werte bei den entsprechenden Indices auf 0 gesetzt werden.

Eine FFT kann Elektroenzephalogramme fast verlustfrei repräsentieren. Dies lässt sich erkennen, wenn man mithilfe der durch die FFT entstandenen Spektralanalyse die Daten rekonstruiert (Inverse Fast Fourier Transformation, IFFT) (siehe Abbildung 4). Es werden dazu die Sinus-Kurven der Frequenzen mit den entsprechenden Amplituden multipliziert und dann addiert.

Eine große Amplitude bei 50 Hertz, wie z.B. in Abbildung 4, kann aufgrund des Wechselstroms in Deutschland entstehen, welcher eine Frequenz von 50 Hertz hat. [7] Um eine Verwirrung der KI zu verhindern, können wir bei der Verwendung von FFT die Amplitude für 50 Hertz auf 0 setzen.

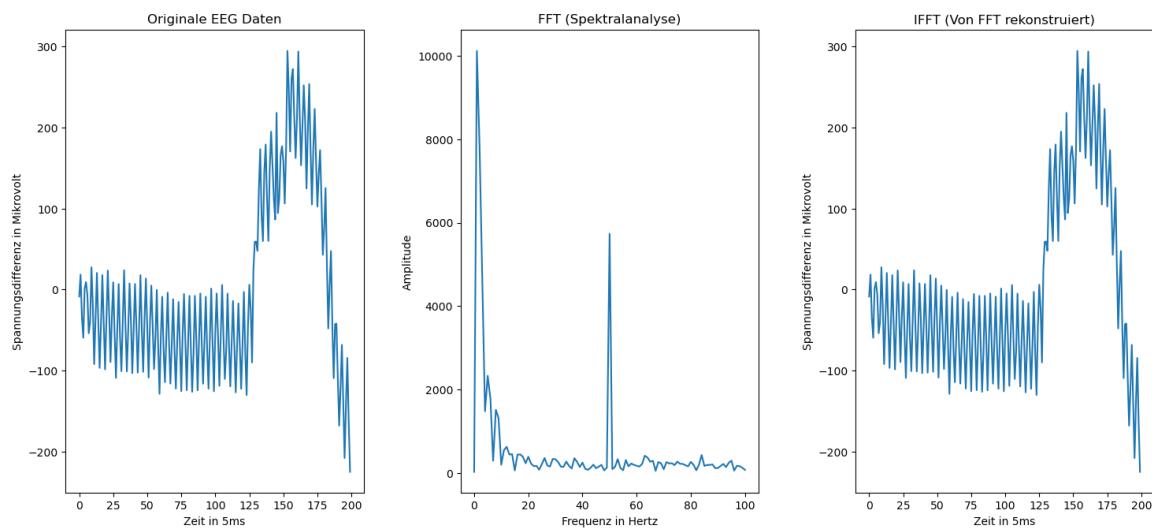


Abbildung 4: Das EMG-Signal eines Blinzelns, die Amplituden des FFT dieses Signales, und eine Rekonstruktion des Signals durch IFFT

Vor der Implementierung von FFT haben wir zuerst getestet, ob eine Spektralanalyse für unseren Zweck überhaupt sinnvoll wäre. Dazu haben wir die Ergebnisse der FFT von Elektroenzephalogrammen mit Blinzeln und ohne Blinzeln verglichen. Wie man in Abbildung 5 sehen kann, gibt es aufgrund des Berger-Effekts vor allem im niedrigeren Frequenzbereich einen klaren Unterschied zwischen geöffneten und geschlossenen Augen. Die KI sollte in der Lage sein, diesen Unterschied zu erkennen. Eine FFT wäre also zur Vorverarbeitung der Daten geeignet.

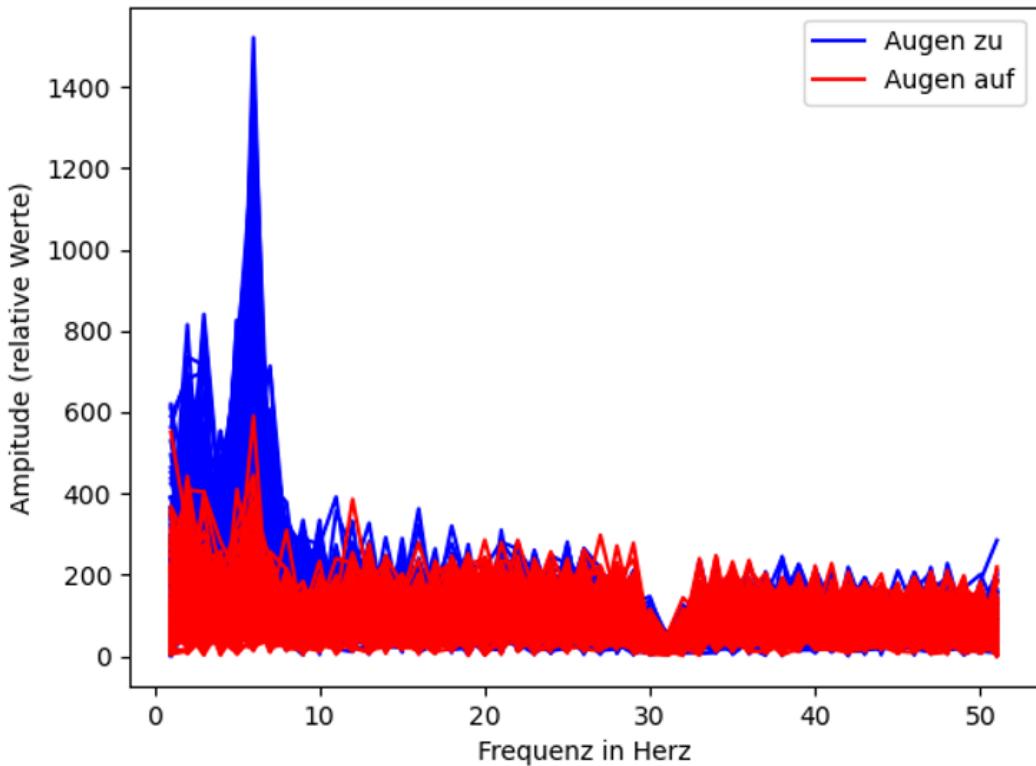


Abbildung 5: Die Amplituden für die Frequenzen 1-50 Hertz von unseren Trainings- und Testdaten (EEG, Konfiguration 2), berechnet durch die FFT. Jede Linie entspricht einem Datensatz. Blaue Linien: durchgehend geschlossene Augen, rote Linien: geöffnete Augen.

Ob die FFT Analyse einen großen Vorteil zu den Rohdaten darstellen wird, werden wir in den Ergebnissen sehen müssen, jedoch liegt es nahe, da die KI sich auf einige wenige, wichtige Eingaben konzentrieren kann. Denn wie man sehen kann, ist der Unterschied in den höheren Frequenzbereichen nicht so groß und vermutlich für eine sichere Erkennung nicht unbedingt notwendig.

Die FFT ist außerdem sehr schnell, auf unserem Test-System braucht sie für die Verarbeitung von einer Sekunde Messdaten von zwei Elektroden im Schnitt 22 µs. Ein weiterer Vorteil der FFT ist, dass wir weniger Eingaben benötigen. Wir brauchen allerhöchstens die Amplituden für 1-100 Hertz, alles darüber hinaus kann unser EEG sowieso nicht akkurat wahrnehmen. Mit 100 Eingaben (die Frequenzen) statt 200 Eingaben (jede einzelne Spannungsdifferenz) pro Elektrode kann unsere KI schneller lernen und schneller eine Erkennung durchführen.

Dennoch wollen wir das Programm auch ohne FFT ausprobieren.

3.2.3 Neuronales Netz

Ein neuronales Netzwerk besteht aus drei Teilen: der Eingabeschicht, den verdeckten Schichten und der Ausgabeschicht. Die Eingabeschicht ist eine Liste aus Zahlen. Sie gibt an, welche Eingaben das Netzwerk bekommen soll, z. B. die Grauwerte der Pixel eines Bildes. Die verdeckten Schichten sind eine Ansammlung von in mehrere Schichten unterteilten Neuronen. Jedes Neuron besitzt eine Aktivierung, die als Zahl zwischen 0 und 1 angegeben werden kann, und einen Bias (Verzerrung), der eine beliebige Zahl ist. Die Neuronen verschiedener Schichten sind alle durch sogenannte Gewichte (*weights*) verbunden, die ebenfalls einen beliebigen Wert haben. Die Ausgaben / Vorhersagen des neuronalen Netzwerkes sind dann lediglich die Aktivierungen der Neuronen in der Ausgabeschicht.

Forward pass

Zur Berechnung der Aktivierung der Neuronen gibt es den sogenannten *forward pass*. Dabei beginnt man in der ersten verdeckten Schicht damit, für alle Neuronen die sogenannte Netzeingabe zu berechnen. Um die Netzeingabe eines Neurons zu berechnen, werden alle Aktivierungen der vorherigen Schicht mit den von dem Neuron dorthin führenden Gewichten multipliziert und aufsummiert. Der Bias ist eigentlich auch ein Gewicht, jedoch ist er mit einem Neuron verbunden, das immer die Aktivierung 1 hat.

Um aus dieser Netzeingabe nun die Aktivierung zu berechnen, benötigt man eine Aktivierungsfunktion, die dafür sorgt, dass die Aktivierung immer zwischen 0 und 1 liegt. Wir haben dafür eine Sigmoidfunktion benutzt, die eine Zahl nimmt und einen Wert zwischen 0 und 1 ausgibt (siehe Abbildung 6). Dies wird dann für jedes Neuron in jeder Schicht wiederholt. Da man aber immer die Aktivierungen der vorherigen Schicht benötigt, muss man das Ganze von der Eingabeschicht zur Ausgabeschicht durchführen. Daher auch der Name *forward pass*. Die Interpretation der Ausgaben hängt von den Trainingsdaten ab.

Die allgemeine Formel für die Aktivierung eines Neurons lautet also:

$$\text{sig}(x) = \frac{1}{1 + e^{-x}}$$

$$a_j = \text{sig} \left(\sum_L (a_L * W_{Lj}) + b_j \right)$$

wobei a_j die Aktivierung des Neurons j , L die vorherige Schicht, a_L der Vektor aller Aktivierungen der Schicht L , W_{Lj} der Vektor aller Gewichte zwischen dem Neuron j und den Neuronen der Schicht L , und b_j der Bias des Neurons j ist. [14]

Loss

Um zu bestimmen, wie gut ein neuronales Netz ist, gibt es die sogenannte Verlustfunktion (engl. *loss function*), die die Datensätze verwendet, um die Abweichung des Netzwerks vom Ideal zu bestimmen. Diese Abweichung wird auch *loss* genannt.

Trainingsdaten bestehen aus einer Liste aus Trainingsdatensätzen. Jeder dieser Datensätze beinhaltet Eingaben für das Netzwerk und die richtigen Ausgaben dafür. Machine Learning, das mit solchen Trainingsdaten arbeitet, wird Supervised Learning genannt.

Die Funktion für den *loss* der Ausgabeschicht und somit des gesamten Netzwerkes (für einen Datensatz) lautet wie folgt:

$$C_0 = (a_L - y)^2$$

wobei C_0 der *loss* der Ausgabeschicht, L die letzte Schicht (Ausgabeschicht), a_L der Vektor aller Aktivierungen der Schicht L , und y ein Vektor der richtigen Ausgaben für die Eingaben, mit denen die Aktivierungen berechnet wurden, ist.

Um den *loss* zu berechnen, muss man also für alle Neuronen der Ausgabeschicht die Differenz der durchs Netzwerk gegebenen und der in den Datensätzen vorgegebenen Aktivierungen bilden. Danach muss man diese Differenzen quadrieren und am Ende alle Ergebnisse aufsummieren. Dies kann man für alle Testdatensätze wiederholen und von allen *losses* den Durchschnitt nehmen, um die allgemeine Performance eines Netzwerkes zu überprüfen. Diese Art der Verlustfunktion wird mittlere quadratische Abweichung (engl. *mean squared error*, kurz MSE) genannt.

Zusammenfassend kann man also sagen, dass der *loss* die Abweichung von den aktuellen Ausgaben des Netzwerkes zu den idealen Ausgaben des Netzwerkes darstellt – ein perfektes

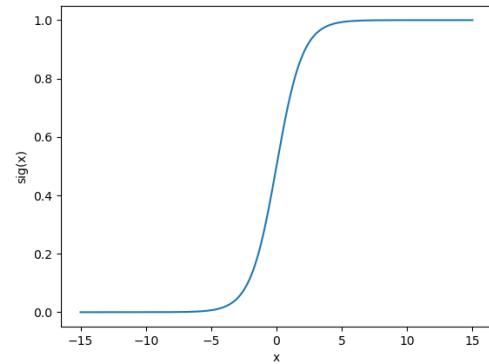


Abbildung 6: Graph der Sigmoidfunktion. Wir nutzen die Sigmoidfunktion um die Aktivierung, $\text{sig}(x)$, aus der Netzeingabe, x , zu berechnen.

neuronales Netz hätte einen *loss* von 0. Aufgrund der Trainingsdatensätze und der Verlustfunktion weiß man nun, wie stark die Ausgabeschicht abweicht und entsprechend verändert werden muss.

Backpropagation

Mithilfe der sogenannten Backpropagation werden die Gewichte des Netzwerkes mithilfe der Datensätze angepasst, um den *loss* zu senken.

Die von uns verwendete Form der Backpropagation ist das simple sogenannte Gradientenverfahren.

Um die Aktivierungen der Ausgabeschicht anzupassen, müssen alle Gewichte und Biases davor angepasst werden. Um nun zu wissen, wie ein Gewicht verändert werden muss, gibt es beim Gradientenverfahren folgende Funktion:

$$\Delta W_{ij} = \epsilon * \delta_i * a_j$$

Hier ist ΔW_{ij} der Wert, um den das Gewicht W zwischen den Neuronen j und i verändert werden muss, ϵ die Lernrate (meist ein kleiner Wert wie 0.001), δ_i eine Annäherung der Ableitung des *loss* des Neurons i im Verhältnis zum Gewicht W_{ij} , und a_j die Aktivierung des Neurons j . Dabei ist oft verwirrend, dass die Schicht des Neurons i aus der Sicht des *forward pass* weiter hinten liegt als die Schicht des Neurons j .

Für den Bias wird die gleiche Formel benutzt, mit der Ausnahme, dass a_j immer 1 ist und so wegfällt. Der Grund dafür liegt darin, dass der Bias, wie im Abschnitt *forward pass* beschrieben, eigentlich nur ein Gewicht ist, das mit einem Neuron verbunden ist, welches immer eine Aktivierung von eins hat.

Um nun δ_i für die Ausgabeschicht zu berechnen, gibt es folgende Gleichung:

$$\delta_i = \text{sig}'(x_i) * (a_i(\text{soll}) - a_i(\text{ist}))$$

wobei $\text{sig}'(x)$ die Ableitung von $\text{sig}(x)$ ist, also $\text{sig}'(x) = \text{sig}(x) * (1 - \text{sig}(x))$, x_i der Netzeingabe des Neurons i , $a_i(\text{soll})$ die Aktivierung, die das Neuron haben sollte (also das gleiche wie y), und $a_i(\text{ist})$ die Aktivierung, die das Neuron hat. Mit dieser Formel wird berechnet, welche Aktivierung das Neuron haben sollte, was an $a_i(\text{soll}) - a_i(\text{ist})$ erkennbar ist. Die Aktivierungsfunktion mit der Netzeingabe wird als Faktor mit einberechnet, da möglichst nur die Gewichte stark verändert werden sollen, die bei dem Trainingsdatensatz eine hohe Aktivierung haben, also durch diese Eingaben besonders angesprochen werden. So werden zum Beispiel beim Sortieren nur die Neuronen stark miteinander verknüpft, die für ein bestimmtes Muster verantwortlich sind.

Für die Neuronen der verdeckten Schichten muss man alle δ 's der nächsten Schicht mit den von dem Neuron dorthin führenden Gewichten multiplizieren und dann summieren. Dadurch werden die Änderungen, die die Aktivierungen dieser Neuronen brauchen (δ), zusammengerechnet, da natürlich die Aktivierungen in der nächsten Schicht unterschiedliche Änderungen in dem gleichen Neuron benötigen.

Durch die Multiplikation mit den dahin führenden Gewichten werden diese Änderungen gewichtet, da sie auf einige Neuronen größere Auswirkungen haben als auf andere. Wie bei der Ausgabeschicht auch wird diese Summe noch mit $\text{sig}'(x)$ multipliziert, um die Aktivierung durch bestimmte Muster angesprochener Neuronen noch weiter zu erhöhen und weniger/kaum angesprochener Neuronen zu senken, sodass die Ergebnisse besser und eindeutiger werden. Die Formel hierfür lautet:

$$\text{sig}'(x_i) * \sum_L (\delta_L * W_{Li})$$

wobei L die nächste Schicht, δ_L der Vektor aller δ 's der Schicht L , und W_{Li} der Vektor aller Gewichte, die ein Neuron der nächsten Schicht und Neuron i verbinden, ist. Da immer die (in Reihenfolge des *forward pass* gesehene) nächste Schicht benötigt wird, ergibt es Sinn, diese Optimierung bei der Ausgabeschicht zu starten und dann rückwärts die δ -Werte für jede Schicht zu berechnen

und für die nächsten Berechnungen zu speichern – daher auch der Name Backpropagation. [12] [15] [16]

Flux

Wir haben in unserem letzten Projekt bereits ein neuronales Netzwerk mit dieser Funktionsweise selbst programmiert, um es besser verstehen zu können. [28] Für dieses Projekt haben wir allerdings das Package Flux benutzt, welches die gleichen oder besseren Ergebnisse liefern sollte, mit deutlich besserer Performance, da es stark optimiert und sehr viel weiter entwickelt wurde. [21]

3.2.4 Roboter

Für unseren Roboter haben wir uns entschieden, Lego Mindstorms EV3 Motoren zu benutzen, die von einem EV3-Brick gesteuert werden, der wiederum von einem Raspberry Pi 3B+ gesteuert wird (siehe Abbildungen 7 und 8). Der Grund dafür, dass wir keine Motoren direkt mit dem Raspberry Pi steuern, ist, dass wir schon alle Teile für einen EV3 Roboter haben. Somit müssten wir entweder neue Motoren kaufen oder passende Adapter finden, welche meist nur mit der Sprache C und Python funktionieren und teuer sind.

Der Roboter, zu sehen in Abbildung 8, wurde von Alexander Reimer, Ino Saathoff, und Oliver Samkovskij im Rahmen der Robotik-AG unserer Schule für den RoboCup entwickelt. Die Steuerung in Julia wurde jedoch von uns selbst programmiert.

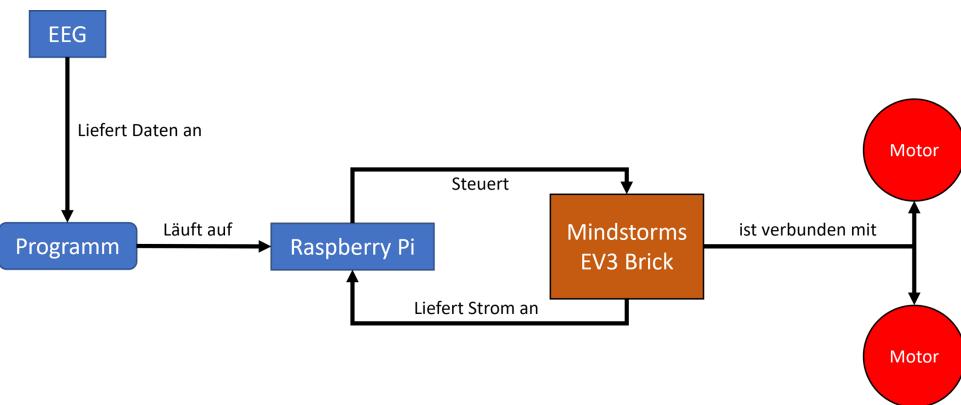


Abbildung 7: Darstellung des Informationsflusses in unserem Experiment. Die über das EEG aufgenommen Gehirnaktivitäten werden durch das neuronale Netz analysiert. Das Ergebnis wird an den Raspberry Pi weitergeleitet, welcher mittels des EV3-Bricks die Motoren ansteuert.

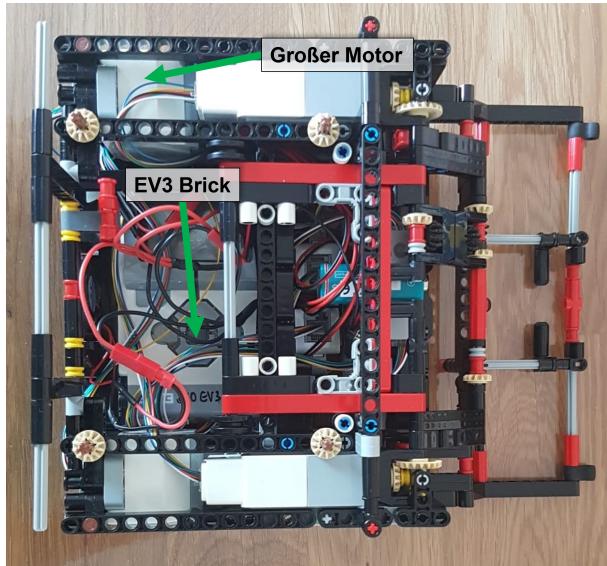


Abbildung 8: Unser Roboter (ohne Raspberry Pi, der über dem EV3-Brick angebracht wird). Die extra mittleren Motoren, die Konstruktion vorne, sowie die Sensoren vorne und unten sind für unser Projekt nicht relevant.

Die Steuerung besteht aus zwei Komponenten: dem Brick und dem Computer. Auf dem EV3-Brick wurde mit einer SD-Karte das Betriebssystem ev3dev installiert. Dieses gibt neben komplettem Zugriff auf das Debian-Betriebssystem des Bricks auch Zugriff auf die Motoren. Ev3dev übernimmt die tiefgreifende, direkte Kontrolle der Motoren, und ermöglicht eine Steuerung / einen Informationsabruft über verschiedene Dateien im `/sys/class` Verzeichnis (s. Beispiele in Tabelle 1)

Dateipfad	Funktion
<code>/sys/class/tacho-motor/motor<x>/address</code>	Der Port, an dem der Motor angeschlossen ist (outA, outB, etc.)
<code>/sys/class/tacho-motor/motor<x>/command</code>	Einen „Befehl“ an den Motor senden, wie z.B. <code>stop</code> , <code>run-timed</code> oder <code>run-forever</code>
<code>/sys/class/tacho-motor/motor<x>/commands</code>	Alle verfügbaren Befehle.
<code>/sys/class/tacho-motor/motor<x>/speed_sp</code>	Die aktuelle Geschwindigkeit des Motors auslesen oder setzen.
<code>/sys/class/tacho-motor/motor<x>/duty_cycle_sp</code>	
...	...

Tabelle 1: Einige Beispieldateien für die Steuerung eines Motors auf ev3dev. <x> ist die Nummer des Motors. Mehr Informationen dazu gibt es in der Dokumentation von ev3dev. [4]

Auf dem Computer (ein Raspberry Pi 3B+) verwenden wir SSHFS, um einen Mount zu erstellen, mithilfe dessen wir über USB vom Pi aus auf diese Dateien zugreifen können. In einer von uns eigens entwickelten Julia-API haben wir dann Funktionen implementiert, welche dem Nutzer ein unkompliziertes Steuern des Roboters erlaubt. Dabei machen die low-level Funktionen nichts anderes, als z.B. die vom Nutzer gewünschte Geschwindigkeit in die entsprechende Datei zu schreiben. Den Code dazu haben wir auf GitHub veröffentlicht, unter dem Namen ev3dev.jl. [27]

Das Programm sollte ebenfalls mit dem Raspberry Pi 4B funktionieren, jedoch haben wir momentan nur einen Raspberry Pi 3B+. Dieser ist vergleichsweise sehr schwach, mit nur 1 GB Arbeitsspeicher. Vor allem die Menge an Arbeitsspeicher stellt ein Problem dar, da sie nicht groß genug ist, um überhaupt alle benötigten Packages zu laden. Deshalb müssen wir jetzt erstmal einen Umweg gehen: Anstatt das ganze Programm direkt auf dem Raspberry Pi am

Roboter auszuführen, erstellen wir mit SSHFS auf unserem Laptop einen Mountpoint, welcher über WLAN auf den Mountpoint auf dem Raspberry zugreift, und so über diesen Zugriff auf das `/sys/class`-Verzeichnis des Bricks hat. Die Auslesung der EEG-Daten, das Auswerten durch das neuronale Netzwerk, und die Steuerung der Motoren können so alle auf dem Laptop stattfinden.

3.2.5 Übersicht des Ablaufs unseres Programmes in einem Diagramm

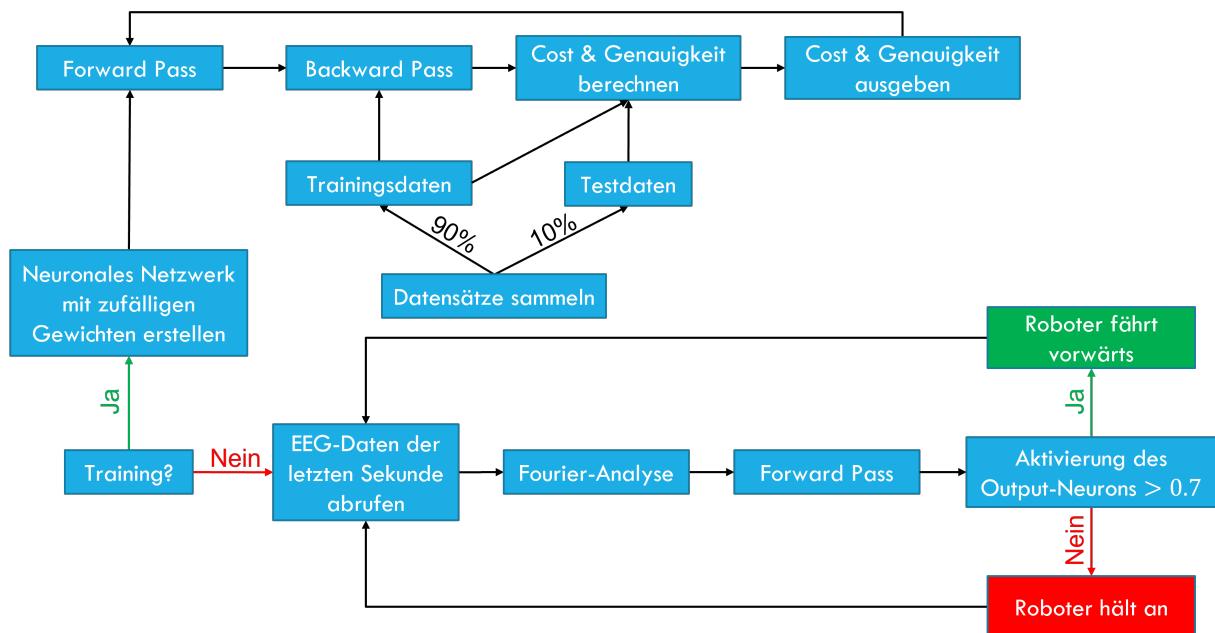


Abbildung 9: Eine Übersicht des Ablaufs unseres Programmes in einem Diagramm.

Wenn wir trainieren, dann erstellen wir zuerst ein neuronales Netzwerk mit zufälligen Gewichten. Danach trainieren wir es mithilfe der Trainingsdaten, die 90% unserer vorher gesammelten Datensätze sind. Mit den restlichen 10%, die die Testdaten formen, sowie den Trainingsdaten, berechnen wir die aktuelle Genauigkeit und loss. Dies wird wiederholt, bis eine zufriedenstellende Leistung erreicht wurde. Wenn wir das Programm nutzen, also nicht trainieren wollen, dann rufen wir durchgehend die EEG-Daten der letzten Sekunde ab, berechnen die Amplituden der Frequenzen mit FFT, und geben diese an das neuronale Netz als Eingaben. Falls die Aktivierung des Neurons der Ausgabeschicht über dem Schwellenwert liegt, dann fährt der Roboter vorwärts, und falls sie darunter liegt, hält der Roboter an. Den Schwellenwert für die Aktivierung des Neurons der Ausgabeschicht haben wir durch Beobachtung des Wertes in Tests sowie Ausprobieren bestimmt. In Abbildung 9 ist der Ablauf nochmal als Diagramm dargestellt.

4 Ergebnisse

Beim Erkennen von Blinzeln mit EMG (Konfiguration 1) erhielten wir, sowohl mit als auch ohne FFT, eine Testdaten-Genauigkeit von 95% (siehe Abbildung 10), in einigen undokumentierten Durchläufen auch 100%. Mit der Genauigkeit ist gemeint, welchen Anteil der gegebenen Datensätze das neuronale Netzwerk korrekt klassifizieren konnte. Dabei ist vor allem die Testdaten-Genauigkeit relevant, da diese die Genauigkeit mit den Testdaten angibt. Die Testdaten sind nicht in den Trainingsdaten enthalten, sodass das neuronale Netzwerk nicht mit ihnen trainiert wurde und so komplett neue, unbekannte Fälle simuliert werden können.

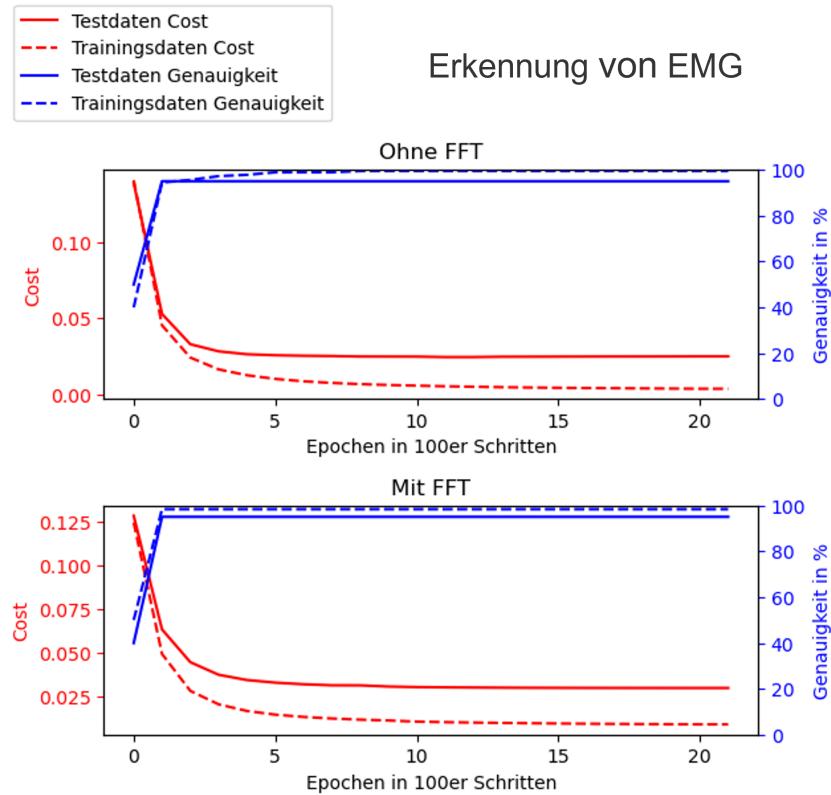


Abbildung 10: Es wurde EMG benutzt. Zu sehen ist der *loss*- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netzwerkes, oben ohne Vorverarbeitung durch FFT und unten mit. Eine Epoche entspricht jeweils einer Backpropagation mit allen Trainingsdatensätzen.

Wir hatten jeweils 100 Trainingsdatensätze für die Fälle, dass geblinzelt und nicht geblinzelt wurde. Für die Struktur der verdeckten Schichten des Netzwerkes haben wir uns entschieden, da wir einige ausprobiert haben und diese konsistent gute Ergebnisse (90-100% Testdaten-Genauigkeit) mit kurzen Trainingszeiten (ca. 3 Minuten für 1000 Epochen, mit FFT) geliefert haben. Sie ist zu sehen in Abbildung 11. Auch ein realer Test mit einer Testperson lief gut, wie man in der Aufnahme davon sehen kann. [17]

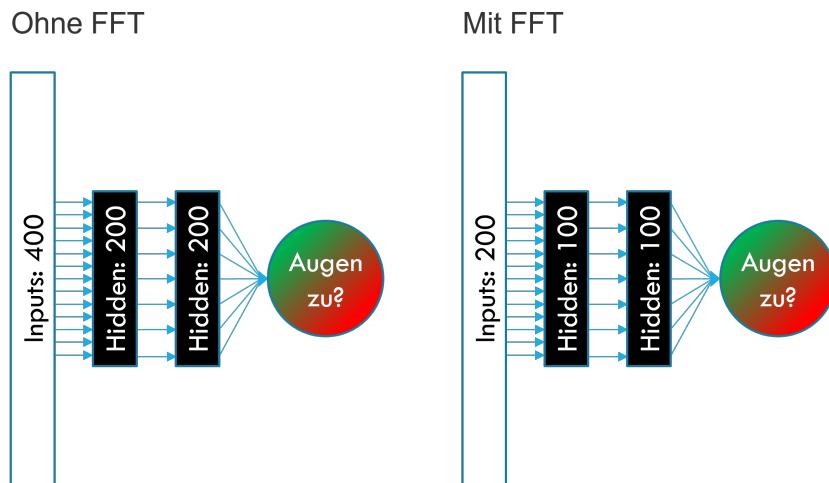


Abbildung 11: Struktur unseres neuronalen Netzwerks bei Verwendung von EMG (Konfiguration 1), links bei Vorverarbeitung mit FFT, rechts ohne. Mit FFT haben wir pro Elektrode nur 100 Eingaben, da wir die Amplituden für Frequenzen 1-100 verwenden.

Nach diesem Erfolg gingen wir nun weiter zur Verwendung von EEG (Konfiguration 2). Dabei

konnte das neuronale Netzwerk ebenfalls eine Testdaten-Genauigkeit von 95% erreichen, jedoch nur mit FFT. Ohne FFT sah es deutlich schlechter aus (siehe Abbildung 12): Die Genauigkeit mit den Trainingsdaten konnte zwar 100% erreichen, jedoch blieb die Testdaten-Genauigkeit, welche viel repräsentativer für die Leistung bei der realistischen Verwendung ist, bei 47,5% stehen.

Der Grund dafür ist vermutlich ein Phänomen namens Überanpassung. Dabei passt sich das neuronale Netzwerk zu stark an die Trainingsdatensätze an, wodurch das Netzwerk die Fähigkeit, auch neue, leicht unterschiedliche Daten korrekt zu erkennen, verliert / nicht erlernt. Dafür verantwortlich könnte sein, dass das Netzwerk zu lange trainiert wurde, oder was in unserem Fall wahrscheinlicher ist, ohne FFT keine tieferen Zusammenhänge finden konnte.

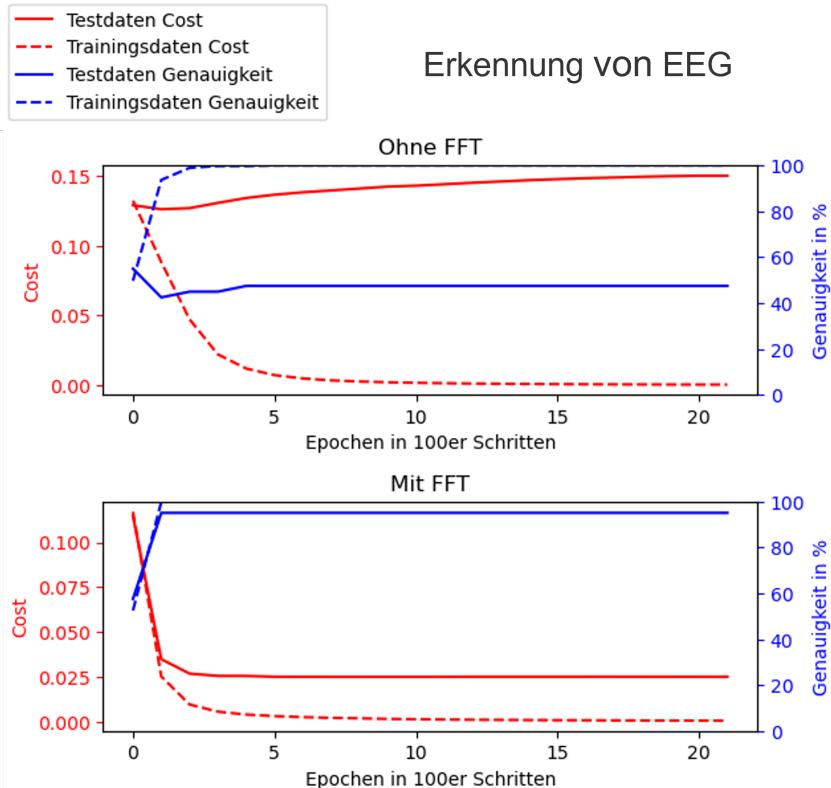


Abbildung 12: Es wurde EEG benutzt. Zu sehen ist der *loss-* und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netwerkes, oben ohne Vorverarbeitung durch FFT und unten mit. Eine Epoche entspricht jeweils einer Backpropagation mit allen Trainingsdatensätzen.

Ein Live-Test mit FFT verlief gut, wie man in der Aufnahme dessen sehen kann. [18] Die Netzwerkstruktur für EEG ist in Abbildung 13 zu sehen.

Der gesamte Quellcode unseres Projektes ist auf unserem GitHub Repository „Interpreting-EEG-with-AI“ veröffentlicht. [19]

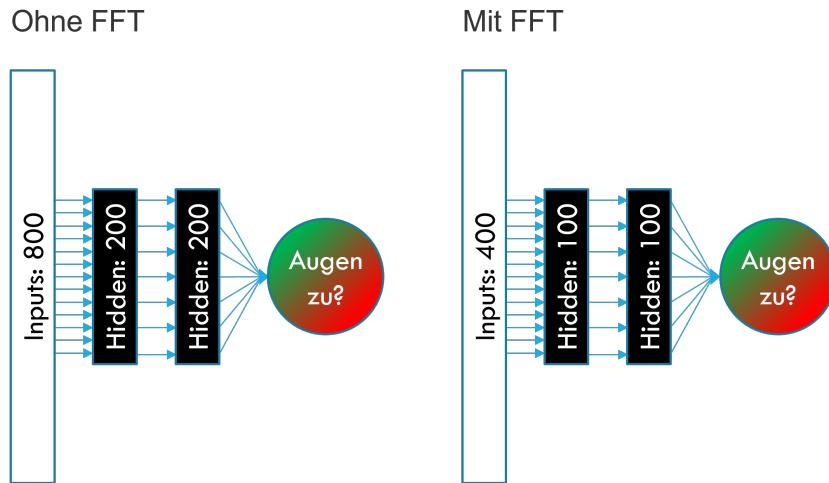


Abbildung 13: Struktur unseres neuronalen Netzwerks bei Verwendung von EEG (Konfiguration 2), links bei Vorverarbeitung mit FFT, rechts ohne. Mit FFT haben wir pro Elektrode nur 100 Eingaben, da wir die Amplituden für Frequenzen 1-100 verwenden.

5 Diskussion

Wir sind zufrieden damit, dass wir unser erstes Teilziel erreicht haben: Der Roboter fährt sicher, und es gibt kaum eine Verzögerung. Die Kosten der benutzten Hardware halten sich noch in Grenzen: Beim Kauf hat die gesamte EEG-Ausstattung ca. 550€ gekostet, was zwar eigentlich eine hohe Summe ist, aber professionelle Ausrüstung kostet deutlich mehr. Die Performance des Programms kann außerdem zwar noch verbessert werden, jedoch dauern beim Trainieren 1000 Epochen (genug für ca. 100% Genauigkeit) auf dem Test-System bereits lediglich 5 Minuten.

Die Anpassbarkeit ist begrenzt gegeben, da es zwar theoretisch funktioniert, wie unser Wechsel von EMG zu EEG zeigt, aber noch nicht sehr leicht machbar ist. Zuerst fokussieren wir uns darauf, das BCI an sich möglichst gut zu machen und bis an seine Grenzen zu bringen, da wir durch die dafür benötigten Verbesserungen auch die Limits für zukünftige Nutzer erhöhen können. Trotzdem hoffen wir, möglichst bald das Programm nutzerfreundlich zu machen, womöglich sogar mit einem GUI.

Um diese Limits zu erreichen, wollen wir als Nächstes unser Programm auch mit Personen testen, von denen wir vorher keine Trainingsdaten gesammelt haben. Danach wird das Ziel sein, auf andere EKPs wie Konzentration umzusteigen, und vielleicht sogar mehrere gleichzeitig zu erkennen, damit wir für den Roboter mehr „Kontrolldimensionen“ haben können, z.B. für Geschwindigkeit, rechts / links, etc. Wenn wir genügend EKPs sicher erkennen können, dann könnten wir sogar die Steuerung einer Drohne versuchen. Man könnte dafür auch für mehrere Befehle ein EKP nutzen. Z.B. könnte zweimal schnell hintereinander Blinzeln für rechts, dreimal für links, viermal für oben, etc. stehen. Man könnte für eine Texteingabe auch versuchen, „Protokolle“ wie Morsecode zu erkennen. Für eine korrekte Klassifizierung eines einzelnen Ereignisses (z.B. Augenschließen) braucht unser System eine Sekunde. Das heißt, wir könnten Signale mit einer Datenübertragungsrate von 1 Bit pro Sekunde verarbeiten.

Ein Problem, welches wir hatten, war unsere EEG-Messausstattung. Sie war sehr anfällig für Störungen: Die Kabel durften nicht berührt werden, nicht zu viele technische Geräte durften in der Nähe sein, und in einigen Räumen war die Signalqualität manchmal sehr stark gestört, ohne dass wir wussten, warum. Also wollen wir mögliche Störfaktoren finden und eliminieren, damit wir konsistente Ergebnisse haben können.

Außerdem wollen wir uns einen Raspberry Pi 4B mit 8 GB Arbeitsspeicher besorgen. Dieser sollte leistungsfähig genug sein, um unser Programm auch direkt auf dem Roboter ausführen zu können. So wären das EEG-Gerät und der Roboter auch ohne einen extra Laptop nutzbar.

Eine weitere Verbesserungsmöglichkeit sehen wir in der Fourier-Analyse: Bei unserem aktuellen

Aufbau sind vor allem die niedrigen Frequenzen wichtig, da wir die Alpha-Blockade zur Erkennung nutzen und am Okzipitalen Kortex die Alpha-Wellen dominant sind. Die Bedeutung des niedrigen Frequenzbereichs kann man auch in Abbildung 5 sehen. Es könnte sich also lohnen, z.B. nur die Amplituden der Frequenzen 1-20 Hertz an das neuronale Netz als Eingaben zu geben, um die Leistung und Performance des neuronalen Netzes weiter zu steigern.

Außerdem ist der Aufbau des Netzwerks sehr einfach, da er nur aus den beschriebenen, einfachen *fully connected* Schichten besteht. Es gibt inzwischen schon fortgeschrittenere Arten von Schichten in künstlichen neuronalen Netzwerken, wie z.B. *convolutional layer*, und Bauweisen, wie z.B. *recurrent neural networks*. Außerdem ist das Gradientenverfahren ein sehr grundlegender Optimierungs-Algorithmus. Inzwischen sind viel fortgeschrittenere entwickelt worden, wie z.B. ADAM, was inzwischen praktische der Standard in Machine Learning ist. Wir erhoffen uns durch eine Verbesserung des neuronalen Netzwerkes die Anwendungsmöglichkeiten des BCIs stark zu erweitern.

6 Danksagung

6.1 Finanzierung

Danke an den Förderverein „Gesellschaft der Freunde des Gymnasium Eversten e.V.“ für die Finanzierung des EEG-Geräts. Ohne diese Hilfe wäre dieses Projekt nie zustande gekommen. Alle finanzierten Teile sind in der Materialliste mit ★ gekennzeichnet.

6.2 Unterstützung

Vielen Dank an Professor Everling vom „The Brain and Mind Institute“ der Western University in Kanada (<https://www.uwo.ca/bmi/investigators/stefan-everling.html>), der uns in einem Gespräch mit ihm geholfen hat, einen Ansatz in diesem komplizierten Thema zu finden, und praktische Tipps zum Umgang mit dem EEG gegeben hat.

Dank geht auch an Ino Saathoff, der für uns die Hülle der EEG-Platine mit seinem 3D-Drucker erstellt hat.

Wir danken ebenfalls Oliver Samkovskij, der die Lego-Struktur des Roboters gebaut hat, und Ino, der für die Verkabelung und Stromversorgung verantwortlich war. Der Roboter ist zwar mit viel mehr ausgestattet als für unser Projekt notwendig, aber dank euch mussten wir nicht unnötig einen neuen bauen.

Größter Dank geht an unsere Projektbetreuer Herr Dr. Glade und Herr Husemeyer, die unser Projekt begleitet haben. Sie haben uns bei der wissenschaftlichen Methodik geholfen und uns wichtige Ressourcen gegeben sowie bei Fragen weitergeholfen.

7 Quellen & Referenzen

7.1 Abbildungen

Die Vorlage des 10-20-Systems, welche wir in Abbildung 3 verwendet haben, stammt von https://en.wikipedia.org/wiki/File:21_electrodes_of_International_10-20_system_for_EEG.svg und liegt in der Public Domain.

Alle anderen Abbildungen und Fotos wurden von uns selbst erstellt.

7.2 Literatur

- [1] Andrea Bonci u. a. „An Introductory Tutorial on Brain–Computer Interfaces and Their Applications“. In: *Electronics* 10.5 (Feb. 2021), S. 560. DOI: 10.3390/electronics10050560. URL: <https://doi.org/10.3390/electronics10050560>.
- [2] Francis R. Willett u. a. „High-performance brain-to-text communication via imagined handwriting“. In: *bioRxiv* (2020). DOI: 10.1101/2020.07.01.183384. eprint: [https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384.pdf](https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384.full.pdf). URL: <https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384>.
- [3] Ujwal Chaudhary, Niels Birbaumer und Ander Ramos-Murguialday. „Brain-computer interfaces for communication and rehabilitation“. In: *Macmillan Publishers Limited* 12 (Sep. 2016), S. 513–525.
- [4] *Linux Kernel Drivers for ev3dev-stretch*. Dokumentation. URL: <https://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-stretch/>.
- [5] Wikipedia. *Elektroenzephalografie – Wikipedia, The Free Encyclopedia*. <http://de.wikipedia.org/w/index.php?title=Elektroenzephalografie&oldid=216289194>. 2022. (Besucht am 13.01.2022).
- [6] N. Birbaumer und R. F. Schmidt. „Biologische Psychologie“. In: Springer Verlag, Berlin Heidelberg, 2010. Kap. 20.5, S. 468–493.

- [7] Universität Lübeck. *Elektroenzephalographie (EEG) und Ereigniskorrelierte Potentiale (EKP)*. Praktikumsskript.
- [8] Wikipedia. *Electromyography — Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Electromyography&oldid=1070043736>. 2022. (Besucht am 16.02.2022).
- [9] Helga Peter und Thomas Penzel. „Berger-Effekt“. In: *Springer Reference Medizin*. Springer Berlin Heidelberg, 2020, S. 1–1. DOI: 10.1007/978-3-642-54672-3_350-1. URL: https://doi.org/10.1007/978-3-642-54672-3_350-1 (besucht am 30.03.2022).
- [10] Wikipedia. *Berger-Effekt — Wikipedia, The Free Encyclopedia*. <http://de.wikipedia.org/w/index.php?title=Berger-Effekt&oldid=208486396>. 2022. (Besucht am 29.03.2022).
- [11] Sagar Khillar. „Difference Between FFT and DFT“. In: *Difference Between Similar Terms and Objects* (Sep. 2021). URL: <http://www.differencebetween.net/technology/difference-between-fft-and-dft/> (besucht am 18.02.2022).
- [12] Larry Hardesty. „Explained: Neural networks“. In: *MIT News* (2017). URL: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.

7.3 Videos

- [13] Grant Sanderson 3blue1brown. *But what is the Fourier Transform? A visual introduction*. YouTube Video. 2018. URL: <https://www.youtube.com/watch?v=spUNpyF58BY> (besucht am 10.01.2022).
- [14] Brotcruncher. *Neuronale Netze - Backpropagation - Forwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=YIqYBxpv53A> (besucht am 15.01.2022).
- [15] Grant Sanderson 3blue1brown. *But what is a Neural Network?* YouTube Playlist. 2017. URL: https://www.youtube.com/watch?v=aircArUvnKk&list=PLZHQB0b0WTQDNU6R1_67000Dx_ZCJB-3pi (besucht am 10.01.2022).
- [16] Brotcruncher. *Neuronale Netze - Backpropagation - Backwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=EAtnQCut6Qno> (besucht am 15.01.2022).
- [17] Alexander Reimer und Matteo Friedrich. *Live-Test des Programms für Jugend Forscht 2022*. YouTube Video. 2022. URL: <https://www.youtube.com/watch?v=I2osIVPmt20> (besucht am 18.02.2022).
- [18] Alexander Reimer und Matteo Friedrich. *Live-Test des Projekts für Landeswettbewerb Jugend Forscht 2022*. YouTube Video. 2022. URL: <https://www.youtube.com/watch?v=k7P1B90SHLw> (besucht am 31.03.2022).

7.4 Programme

- [19] Alexander Reimer und Matteo Friedrich. *Interpreting EEG with AI*. GitHub Repository. 2022. URL: <https://github.com/AR102/Interpreting-EEG-with-AI>.
- [20] Jeff Bezanson u. a. „Julia: A Fresh Approach to Numerical Computing“. In: *SIAM Review* 59.1 (Jan. 2017), S. 65–98. DOI: 10.1137/141000671. URL: <https://doi.org/10.1137/141000671>.
- [21] Michael Innes u. a. „Fashionable Modelling with Flux“. In: *CoRR* abs/1811.01457 (2018). arXiv: 1811.01457. URL: <https://arxiv.org/abs/1811.01457>.
- [22] Mike Innes. „Flux: Elegant Machine Learning with Julia“. In: *Journal of Open Source Software* (2018). DOI: 10.21105/joss.00602.
- [23] Andrey Parfenov et al. *Brainflow*. GitHub Repository. 2018. URL: <https://github.com/braintflow-dev/brainflow>.

- [24] Matteo Frigo und Steven G. Johnson. „The Design and Implementation of FFTW3“. In: *Proceedings of the IEEE* 93.2 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”, S. 216–231. DOI: 10.1109/JPROC.2004.840301.
- [25] Tim Besard, Christophe Foket und Bjorn De Sutter. „Effective Extensible Programming: Unleashing Julia on GPUs“. In: *IEEE Transactions on Parallel and Distributed Systems* (2018). ISSN: 1045-9219. DOI: 10.1109/TPDS.2018.2872064. arXiv: 1712.03112 [cs.PL].
- [26] Steven G. Johnson. *PyPlot.jl*. GitHub Repository. 2012. URL: <https://github.com/JuliaPy/PyPlot.jl>.
- [27] Alexander Reimer. *ev3dev.jl*. GitHub Repository. 2022. URL: <https://github.com/AR102/ev3dev.jl>.
- [28] Alexander Reimer und Matteo Friedrich. *AI-Composer*. GitHub Repository. 2021. URL: <https://github.com/AR102/AI-Composer.jl>.