

Studienarbeit

T3100

**Demonstrator für die diskrete Fourier-Transformation
in JavaScript**

Studiengang Elektrotechnik

Dualen Hochschule Baden-Württemberg Mannheim

von

Niklas Herhoffer

Bearbeitungszeit:	30.09.24 - 02.01.25
Matrikelnummer:	9325144
Kurs:	TEL22AT1
Fachlicher Betreuung:	Prof. Dr. Rüdiger Heintz

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema:

Demonstrator für die diskrete Fourier-Transformation in JavaScript

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Niklas Herhoffer

Kurzfassung

Die Studienarbeit befasst sich mit der Entwicklung eines Demonstrators für die diskrete Fouriertransformation (DFT) in der Programmiersprache JavaScript. Ziel ist es, die Funktionsweise der DFT interaktiv und anschaulich darzustellen. Der Demonstrator verarbeitet Audiosignale, die entweder über ein Mikrofon oder einen Sinus-Generator mit variabler Frequenz bereitgestellt werden. Anschließend wird die Fourier-Transforamtion des Signals als 3D-Plot visualisiert.

Um die Anforderungen zu erfüllen, werden diverse JavaScript-Bibliotheken wie die Web Audio API zur Signalanalyse und Plotly.js zur Visualisierung verwendet. Die Benutzerinteraktion erfolgt über eine grafische Oberfläche (GUI), die mithilfe von lili-gui.js realisiert wird. Das Programm ist in die Module main.js, microphone.js und sineGenerator.js aufgeteilt.

Der Demonstrator zeigt in einer browserbasierten Umgebung die Funktionsweise der DFT und bietet eine intuitive Möglichkeit, deren Prinzipien zu erkunden. Künftige Verbesserungen umfassen die Optimierung der Performance und die Erweiterung der Benutzeranpassungen, wie z. B. die dynamische Anpassung des Abtastzeitraums.

Abstract

This study presents the development of an interactive web-based demonstrator for the discrete Fourier transformation (DFT) using JavaScript. The goal is to visualize the DFT's functionality through real-time signal analysis and frequency spectrum representation. The application supports two input modes: microphone input and a sine wave generator with adjustable frequency. The resulting frequency analysis is displayed as a 3D plot.

Key technologies include the Web Audio API for signal processing and Plotly.js for 3D visualization. User interaction is facilitated by a graphical user interface (GUI) built with lil-gui.js. The implementation is modular, with core components such as main.js, microphone.js, and sineGenerator.js.

The demonstrator effectively illustrates the DFT's principles in a browser environment, offering an accessible tool for learning and experimentation. Future enhancements aim to improve performance and provide more customization options, such as adjustable sampling intervals.

Inhaltsverzeichnis

Erklärung	i
Kurzfassung	ii
Abstract	iii
Abbildungsverzeichnis	vi
Tabellenverzeichnis	vii
Programmlistings	viii
Abkürzungsverzeichnis	ix
1 Einleitung	1
2 Aufgabenstellung und Ziele	2
3 Stand der Technik	4
3.1 Bibliothek für den Audio Input	4
3.2 Bibliothek für 3D-Visualisierung	5
3.2.1 Three.js	5
3.2.2 d3.js	6
3.2.3 Plotly.js	7
3.3 Bibliothek für Fast Fouriertransformation (FFT)	7
3.3.1 fft.js	7
3.3.2 KissFFT	8
3.3.3 Web Audio API	9
4 Technische Grundlagen	11
4.1 JavaScript	11
4.2 Diskrete Fouriertransformation	12

5 Programmentwicklung des Demonstrators	14
5.1 Entwicklungsumgebung	14
5.2 Programmierung	14
5.2.1 index.html	15
5.2.2 microphone.js	15
5.2.3 sineGenerator.js	17
5.2.4 main.js	19
5.3 Ergebnis	25
6 Zusammenfassung	28
Literatur	30
A Anhang	32
A.1 Statistik Programmiersprachen	33

Abbildungsverzeichnis

Abbildung 2.1 Schematischer Aufbau des Demonstrators. (eigene Darstellung)	2
Abbildung 2.2 3D-Spektrogramm eines Batterieladegeräts (horizontal: Zeit, vertikal: Frequenz) [19]	3
Abbildung 3.1 Einfachster Audio-Kontext, welcher mit der Web Audio API erstellt werden kann. ([16] 1. Fundamentals)	5
Abbildung 3.2 Grafik einer Three.js Applikation. [17]	6
Abbildung 4.1 Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. (Ausschnitt aus A.1) [12]	11
Abbildung 5.1 Anfangszustand des Demonstrators nach Erlauben des Mikrofonzugriffs.	25
Abbildung 5.2 Ausgabe des Plots mit aktivem Mikrofon.	26
Abbildung 5.3 Ausgabe des Plots mit aktiviertem Sinus-Generator.	26
Abbildung 5.4 Graphical User Interface (GUI) der Ausgabe	27
Abbildung A.1 Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. [12]	33

Tabellenverzeichnis

Tabelle 3.1 Kompatibilitätsliste der benötigten Funktionen der Web Audio API. (vgl. [7, 8, 10, 11])	4
Tabelle 3.2 Vergleich verschiedener FFT Algorithmen für JavaScript. [3]	9

Programmlistings

5.1	HTML-Datei des Demontsrators	15
5.2	Objekt-Datei zur Nutzung des Mikrofons über den Browser	16
5.3	Objekt-Datei zur Nutzung des Sinus-Generators	18
5.4	Programmteil zur Initzialisierung und Konfigurierung der GUI mit <code>lil-gui.js</code>	19
5.5	Programmteil zum Erstellen des 2-dimensionalen Daten Arrays, sowie Auswahldes DIV-Elements.	20
5.6	Programmteil zu Konfigurierung der Plotumgebung.	20
5.7	Programmteil zur Konfigurierung des Plot-Layouts und erstes Anzeigen des Plots in der HTML-DIV	21
5.8	Programmteil zur Erstellung von Objektinstanzen und Aufruf der <code>redraw()</code> -Funtion mit entsprechender Instanz.	22
5.9	Programmteil mit abruf der Frequenzdaten der entsprechenden Objektinstanz, sowie einem Rekursiven Aufruf des eigenen Funtion.	23
5.10	Programmteil mit Funktion zum Leeren des Plots.	24

Abkürzungsverzeichnis

VSCODE	Visual Studio Code
DFT	Diskrete Fouriertransformation
FFT	Fast Fouriertransformation
GPU	Graphics Processing Unit
GUI	Graphical User Interface

1 Einleitung

In der Elektrotechnik gewinnt das Thema Signalverarbeitung in Zeiten der Digitalisierung an immer größerer Bedeutung. Besonders in den Bereichen Bildverarbeitung und Akustik kommt die Fourier-Transformation zur Anwendung. Dort wird diese für Kompressionsverfahren oder Filterung verwendet. Außerdem vereinfacht die Fourier-Transformation die Berechnung einer Signalfaltung. Aus diesem Grund ist die Fourier-Transformation bereits früh im Studium der Elektrotechnik verankert. Für die Fourier-Transformation wird das sogenannte Spektrum (auch Bildbereich genannt) eingeführt, in die die Signale transformiert werden, dort verrechnet und anschließend rücktransformiert werden können. Aufgrund der Abstraktion, die durch das Erschaffen dieser neuen Dimension geschaffen wird, ergibt sich die Hürde, das Prinzip verständlich erklären zu können. Der in diesem Projekt zu entwickelnde Demonstrator dient hier zur weiteren Veranschaulichung. Im Verlauf der Bearbeitung wird im ersten Teil die Aufgaben- und Zielstellung ausformuliert. Über eine Betrachtung des Stands der Technik werden die unterschiedlichen Möglichkeiten der Herangehensweise aufgezeigt. Anschließend werden die technischen Grundlagen geschaffen, die sich zum Verständnis und zur Ausarbeitung als notwendig herausstellen. Der nächste Abschnitt der Arbeit stellt eine Dokumentation der Realisierung des Projekts dar, in der die einzelnen Komponenten des Projekts anhand ihrer Funktion und ihres Nutzens erklärt werden. Abschließend wird im Ergebnis der finale Stand präsentiert.

2 Aufgabenstellung und Ziele

Der Demonstrator ist in der Programmiersprache JavaScript zu entwickeln. Dies hat den Zweck, dass es sich um eine Webanwendung handelt, welche leicht mit jedem Browser verwendet werden kann. Zum Demonstrieren der Diskrete Fouriertransformation (DFT) ist die Möglichkeit eines Mikrofon-Input, als auch die Möglichkeit eines Sinussignals mit veränderlicher Frequenz zu implementieren. Mit dem Sinussignal wird die Funktionalität des Demonstrators überprüft und vorgeführt. Der Schematische Aufbau des Programms wird in der folgenden Abbildung 2.1 aufgeführt.

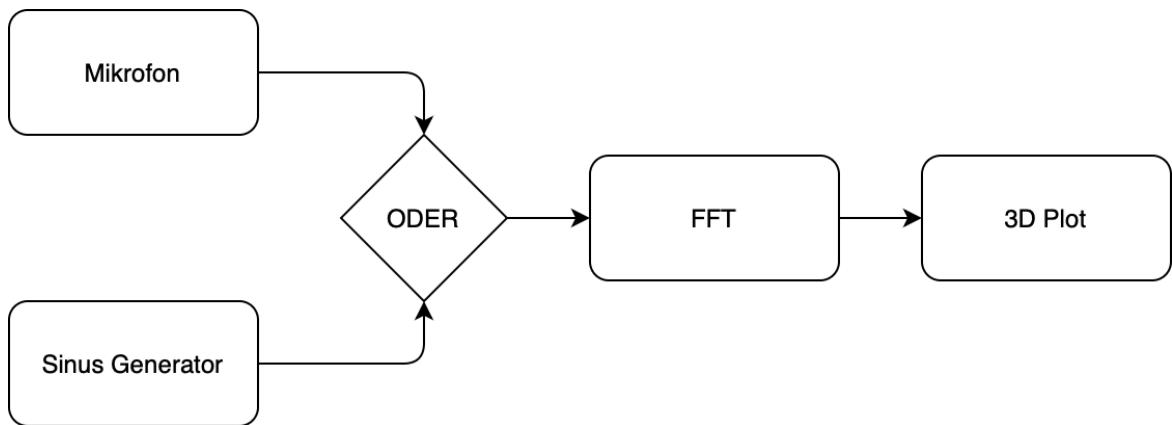


Abbildung 2.1: Schematischer Aufbau des Demonstrators. (eigene Darstellung)

Hier ist zu sehen, dass zwischen den Signaleingängen "Mikrofon" und "Sinus" gewählt werden kann. Das resultierende Signal wird daraufhin mit Hilfe einer Bibliothek für FFT in Zeit und Wert diskretisiert und daraufhin transformiert. Daraus resultiert dann das zeitliche Signal in spektraler Darstellung. Dieses wird dann grafisch in einem 3-dimensionalen Plot dargestellt mit einer Achse jeweils für Amplitude, Zeit und Frequenz. Als Vorbild für die grafische Darstellung dient die folgende Abbildung 2.2. Diese zeigt den grafischen Output eines 3D-Spekrogramms.

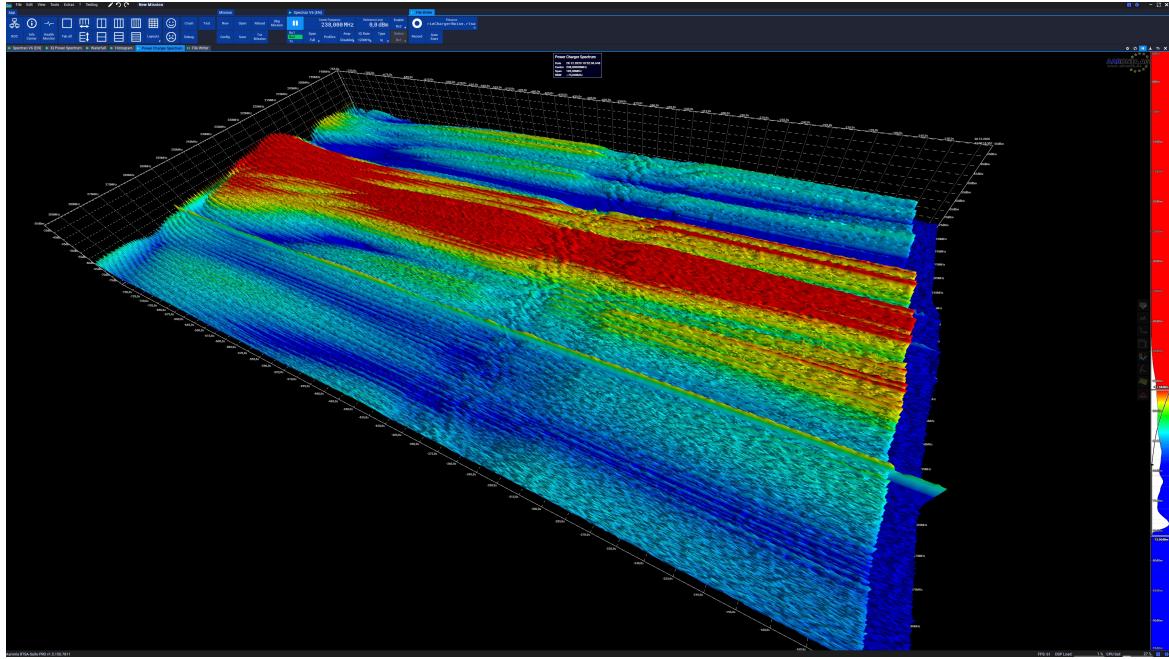


Abbildung 2.2: 3D-Spekrogramm eines Batterieladegeräts (horizontal: Zeit, vertikal: Frequenz) [19]

Solch eine Analyse ist besonders für Audio-Signale sinnvoll. Ziel dieser Arbeit ist die Entwicklung einer 3-dimensionalen Frequenzanalyse für Audio-Signale, sowie eines Frequenzgenerators mit variabler Frequenz. Dieser Demonstrator ist in mindestens zwei verschiedenen Browsern auf Funktion zu testen.

3 Stand der Technik

In diesem Unterkapitel wird der Stand der Technik aufbereitet. Relevant für dieses Projekt sind hier vorrangig die Auswahl der 3D-Visualisierungsbibliothek und die Auswahl der Bibliothek für FFT und Audio-Input.

3.1 Bibliothek für den Audio Input

Die beste Möglichkeit der Nutzung des Mikrofons in JavaScript bietet die Web Audio API. Diese ist ein Standardbibliothek, welche in vielen Browsern vorhanden ist. Die Kompatibilität zeigt folgende Tabelle.

Tabelle 3.1: Kompatibilitätsliste der benötigten Funktionen der Web Audio API. (vgl. [7, 8, 10, 11])

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS
AudioContext	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AnalyserNode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OscillatorNode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MediaDevices	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Die Web Audio API nutzt das Konzept eines Audio-Kontexts. Dieser ist ein gerichteter Graph, welcher zeigt, wie der Audio-Stream von dessen Quelle zu dessen Ziel fließt. Der einfachste Audio-Kontext wird in Abbildung 3.1 dargestellt. Hier ist zu sehen, dass der Audio-Stream von der Quelle direkt zum Ziel fließt. Dies ist auch der Graph, welcher in dem Demonstrator Anwendung findet.

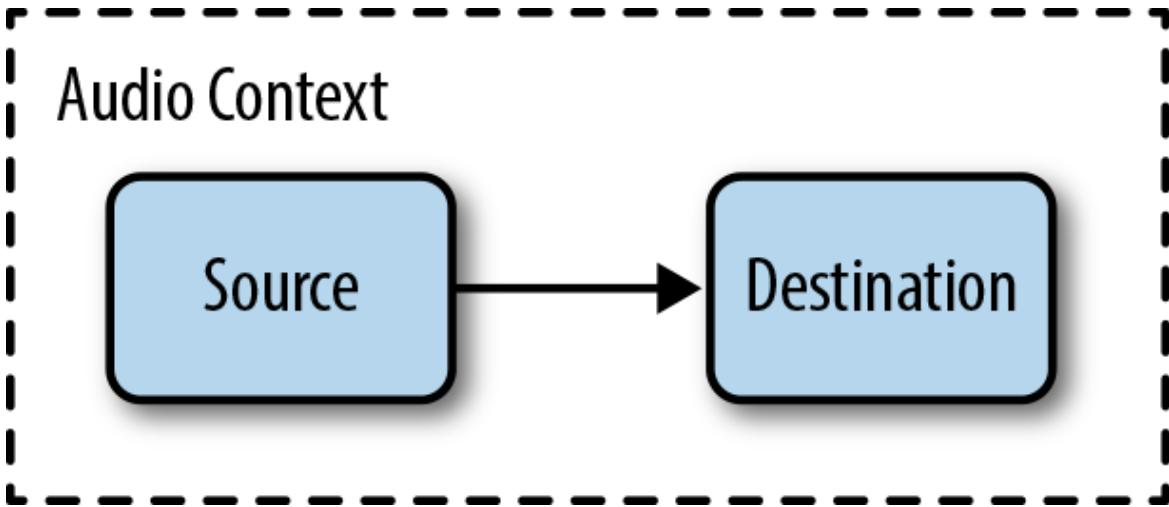


Abbildung 3.1: Einfachster Audio-Kontext, welcher mit der Web Audio API erstellt werden kann. ([16] 1. Fundamentals)

Eine Alternative zur Nutzung der Web Audio API bietet die Bibliothek „microphonejs“. Diese Bibliothek nutzt jedoch auch die Web Audio API und ist seit 2018 nicht mehr aktualisiert worden. Deshalb stellt die Nutzung der Web Audio API ohne zusätzliche Bibliothek für dieses Projekt eine höhere Nutzen dar. (vgl. [18])

3.2 Bibliothek für 3D-Visualisierung

Für die Visualisierung der Daten in einem 3-dimensionalen Plot stehen einige verschiedene Bibliotheken für JavaScript zur Verfügung. Eine Auswahl dieser werden hier behandelt.

3.2.1 Three.js

Three.js ist eine Bibliothek, mit der 3D Animationen und Modelle in JavaScript erstellt und angezeigt werden können. Die Bibliothek nutzt als Renderer WebGL. Mit WebGL werden jedoch nur Punkte, Linien und Dreiecke gezeichnet. Three.js stellt hier also eine Bibliothek für die einfache Verwendung von WebGL als Renderer dar, um mit weniger Aufwand 3D Objekte darstellen zu können. Die in Abbildung 3.2 dargestellte Grafik veranschaulicht den Zusammenhang verschiedener Three.js Komponenten, die

für eine Applikation benötigt werden. Wichtige Bestandteile sind hier der Renderer, die Kamera und die Szene.

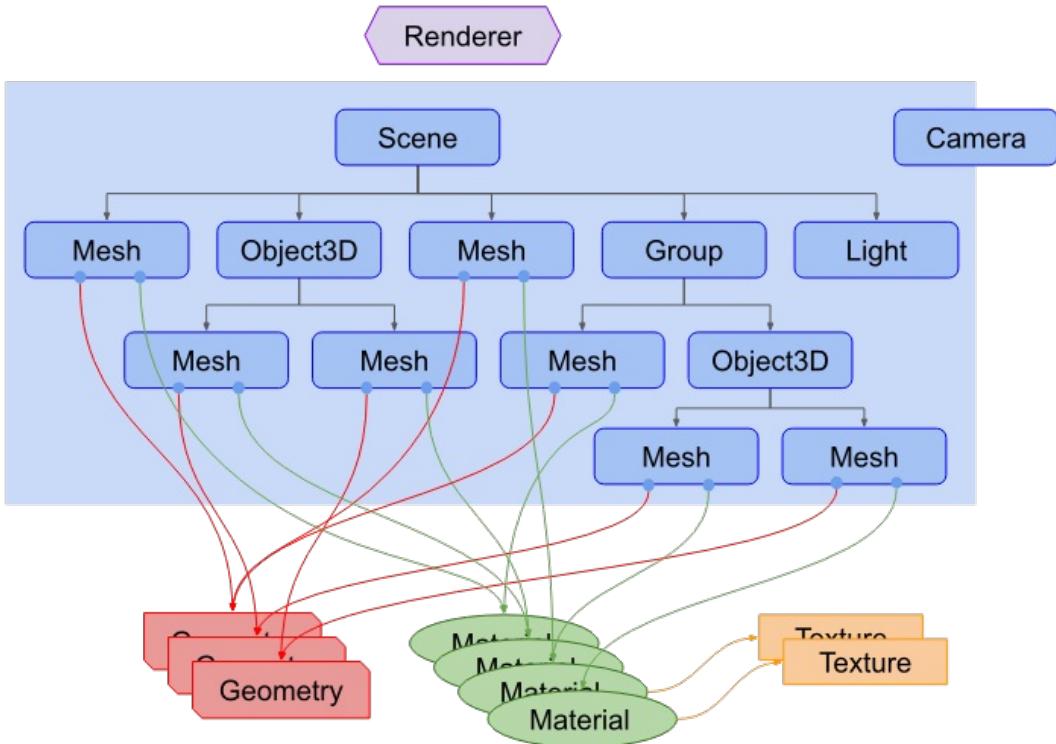


Abbildung 3.2: Grafik einer Three.js Applikation. [17]

Mit Three.js kann jegliche 3D Applikation bewerkstelligt werden, jedoch ist für die Darstellung eines Plots einiges an Arbeit notwendig, da Three.js hierfür keine vorgefertigte Funktion bietet.

3.2.2 d3.js

Diese Bibliothek ist eine mächtige „low-level“ Bibliothek für die Visualisierung von Daten und wurde 2011 von Mike Bostock entwickelt. „d3“ steht hier für "Data Driven Documents". d3.js bietet eine besonders hohe Flexibilität, mit der die Daten in 2-dimensionalen Plots visualisiert werden können. Jedoch bietet die Bibliothek von Haus aus keine Möglichkeit der Visualisierung in drei Dimensionen. (vgl. [2])

3.2.3 Plotly.js

Plotly.js ist eine alleinstehende Bibliothek zur Visualisierung von Daten in JavaScript. Diese Bibliothek bietet viele verschiedene Plots zur Auswahl, unter anderem 3D-Plots. Plotly.js nutzt, wie Three.js, WebGL als Renderer, um die Graphics Processing Unit (GPU) für das Rendern der Plots zu verwenden. Dadurch werden die Webanwendungen beschleunigt. (vgl. [14])

Außerdem bietet Plotly.js mehrere Funktion zum aktualisieren des Plots an. Diese sind bei Änderungen der angezeigten Werte zu verwenden und laut [13] viel effizienter als ein neu zeichnen des Plots.

Für dieses Projekt wird Plotly.js für die Visualisierung der Daten verwendet. Diese Entscheidung ist auf den einfachen Aufbau und die Einfache Bedienung zurückzuführen.

3.3 Bibliothek für FFT

Um das Audiosignal im Spektrum betrachten zu können wird eine FFT durchgeführt. Diese Arbeiten nach dem „divide and conquer“ Prinzip. Dadurch wird eine große DFT in viele kleine DFTs geteilt und somit wird der Rechenaufwand verringert. (vgl. [4] Seite 2)

Weiter Inhalte zu DFT und FFT werden in Kapitel 4.2 genauer betrachtet.

3.3.1 fft.js

Diese Bibliothek ist eine reine Bibliothek zur FFT. Diese nutzt den Radix-4 FFT Algorithmus. Dieser teilt die N-Punkt DFT in mehrere 4-Punkt DFTs (vgl. [4] Seite 5).

Vorteil dieser Bibliothek ist die schnelle Berechnung der FFT, besonders für rein reelle Werte, wodurch die FFT weitere 25% schneller berechnet werden kann. (vgl. [5] README.md)

Nachteil dieser Bibliothek ist, dass die Einbindung dieser via „npm“, ein Paketmanager für „node.js“, erforderlich ist. „node.js“ ist ein JavaScript-Interpreter, der es

ermöglicht Desktop-Anwendungen zu implementieren. Dies ist in diesem Projekt nicht erwünscht.

3.3.2 KissFFT

„KissFFT“ ist eine Bibliothek, welche für die Programmiersprache „C“ entwickelt wurde. Diese wurde von den Autoren der JavaScript-Version für JavaScript übersetzt. Der Algorithmus, welcher von dieser Bibliothek verwendet wird ist der so genannte „Split-Radix“ Algorithmus. Diese vereint die Funktionalität von Radix-4 und Radix-2. Vorteil ist die besonders schnelle Berechnung der FFT. Dies zeigt Tabelle 3.2, welche verschiedene FFT Algorithmen für JavaScript vergleicht. Der KissFFT Algorithmus steht hier mit $80144 \frac{itr}{s}$ auf dem dritten Platz der schnellsten Algorithmen aus dieser Liste. Nachteil dieser Bibliothek ist hier die Einbindung der Bibliothek mit „npm“, wie es auch bei fft.js der Fall ist.

Tabelle 3.2: Vergleich verschiedener FFT Algorithmen für JavaScript. [3]

Implementation	Result	Time (first half)	Time (second half)	Rate (second half)
Nayuki	512000	92 ms	73 ms	27255 itr/sec
	2048000	299 ms	303 ms	6602 itr/sec
Nayuki (obj)	512000	74 ms	58 ms	34707 itr/sec
	2048000	250 ms	232 ms	8614 itr/sec
Nayuki (C)	512000	68 ms	63 ms	31749 itr/sec
	2048000	271 ms	244 ms	8186 itr/sec
Nayuki (C-float)	512000	66 ms	61 ms	32835 itr/sec
	2048000	228 ms	226 ms	8866 itr/sec
KissFFT	512000	33 ms	25 ms	80144 itr/sec
	2048000	88 ms	73 ms	27399 itr/sec
KissFFT (c2c)	512000	40 ms	36 ms	55394 itr/sec
	2048000	132 ms	127 ms	15757 itr/sec
DSP.js	512000	41 ms	29 ms	68977 itr/sec
	2048000	111 ms	95 ms	21055 itr/sec
Cross	512000	53 ms	43 ms	46115 itr/sec
	2048000.0000000002	212 ms	196 ms	10180 itr/sec
FFTW	512000	27 ms	19 ms	102696 itr/sec
	2048000	79 ms	62 ms	32172 itr/sec
RFFTW	512000	26 ms	18 ms	113058 itr/sec
	2048000	70 ms	58 ms	34545 itr/sec
Nockert	512000	117 ms	103 ms	19439 itr/sec
	2048000	411 ms	393 ms	5091 itr/sec
Dntj	511999.9912375561	107 ms	82 ms	24340 itr/sec
	2047999.9649502244	337 ms	324 ms	6175 itr/sec

3.3.3 Web Audio API

Diese Bibliothek beinhaltet neben einer Audio Eingabe, welche in Kapitel 3.1 beschrieben ist, zusätzlich auch die Möglichkeit einen Analyser zu verwenden, mit dem eine FFT der Audio Eingabe durchgeführt werden kann. Die Audio Eingabe ist die Quelle,

die auf den Stream geladen wird und der Analyser ist das Ziel des Streams (wie in Abbildung 3.1 gezeigt). Die Kompatibilität des Analysers der Web Audio API ist in der Tabelle 3.1 dargestellt. Die Berechnung der FFT wird folgendermaßen durchgeführt:

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} \hat{x}[n] \cdot W_N^{-kn} \quad (3.1)$$

mit

$$k = 0, \dots, \frac{N}{2} - 1$$

$$W_N = e^{\frac{j2\pi}{N}}$$

und $\hat{x}[n]$ als mit dem Blackman Fenster gefenstertes Signal. (vgl. [9])

4 Technische Grundlagen

In diesem Kapitel werden die benötigten Grundlagen für das Projekt beschrieben und erklärt. Hierzu zählen Grundlagen zu JavaScript und zur diskreten Fouriertransformation.

4.1 JavaScript

JavaScript ist eine interpretierte Programmiersprache, welche 1995 von Brendan Eich entwickelt wurde. Sie wird vor allem dazu genutzt, um Webseiten dynamischer und interaktiver zu gestalten, bietet jedoch auch die Möglichkeit Desktop-Apps und Backend-APIs zu erstellen. Zudem zählt sie zu den beliebtesten Programmiersprachen weltweit . (vgl. [15] Seite 1)

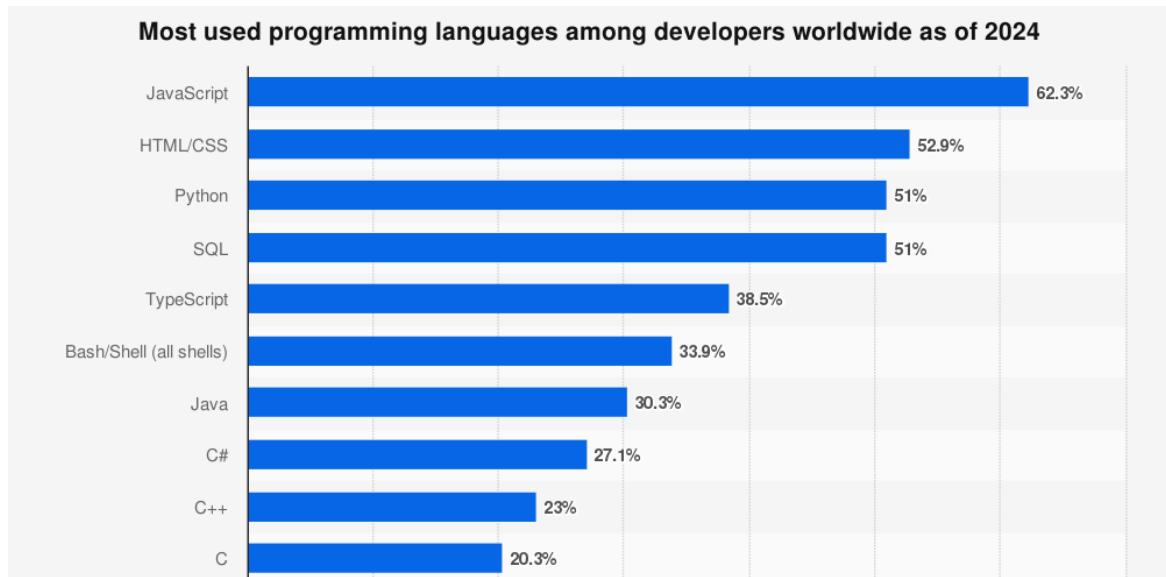


Abbildung 4.1: Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. (Ausschnitt aus A.1) [12]

Die Beliebtheit von JavaScript wird durch die Statistik in Abbildung 4.1 bestätigt. Diese zeigt das Ergebnis einer Umfrage von Stack Overflow im Juli 2024. 62,3% der

Teilnehmer wählten hier JavaScript als Programmiersprache, welche sie verwenden und gerne weiterhin benutzen möchten.

Die Hauptmerkmale von JavaScript sind:

- Dynamische Typisierung von Variablen
- Event-Basierte Funktionalität
- Plattformunabhängig
- Objektorientiert

Die dynamische Typisierung bedeutet, dass die Variablen vor der Nutzung nicht deklariert werden müssen. Dies wird vom Interpreter während der Laufzeit übernommen. Durch die Event-Basierte Funktionalität kann ein JavaScript Programm auf Nutzereingaben über beispielsweise eine GUI reagieren. Die Plattformunabhängigkeit dieser Programmiersprache hat zu Folge, dass die in JavaScript verfassten Programme sich in jedem Browser ausführen lassen.

4.2 Diskrete Fouriertransformation

Die Fouriertransformation ist ein nützliches Werkzeug zur Zerlegung von kontinuierlichen Signalen in dessen Frequenzkomponenten. Dies geschieht über folgenden Zusammenhang von Signal oder Funktion $f(t)$ mit der Transformierten $F(\omega)$:

$$f(t) \rightarrow F(\omega) := \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} dt \quad (4.1)$$

(vgl. [6] Seite 854)

Für die digitale Signalverarbeitung ist es wichtig, dass die Fouriertransformation von Rechnern berechnet werden kann. Da digitalisierte Signale jedoch immer abgetastete und nicht kontinuierlich sind, wird hier eine Abwandlung der Fouriertransformation benötigt. Die DFT bietet hier Abhilfe. Im Gegensatz zur kontinuierlichen Fouriertransformation wird die DFT auf eine endliche Anzahl an Werten angewendet und kann deshalb numerisch bestimmt werden.

Betrachtet man also ein abgetastetes Signal $x[n]$ mit $n = 0 \dots N - 1$, können die abgetasteten Werte folgendermaßen bezeichnet werden:

$$(x_n)_{n=0}^{N-1} \quad (4.2)$$

mit N als Anzahl der abgetasteten Werte. Außerhalb dieses Intervalls sind die Werte für $x_n = 0$.

Für die DFT ergibt sich, analog zu Formel 4.1, folgender Zusammenhang zwischen den abgetasteten Werten aus 4.2 und dessen DFT:

$$(x_n)_{n=0}^{N-1} \rightarrow X_{DFT}(m) := \sum_{n=0}^{N-1} x_n \cdot e^{-jn\Omega_m} \quad (4.3)$$

mit

$$\Omega_m := \frac{2\pi m}{N} \quad (4.4)$$

$$m = 0, \dots, N - 1 \quad (4.5)$$

Die Anzahl der Ergebnisse einer DFT ist immer genau so groß wie die Anzahl der abgetasteten Werte N , die der DFT übergeben werden.

(vgl. [1] Seite 318f.)

Die FFT nutzt das Prinzip des „divide and conquer“, und funktioniert deshalb nur für abgetastete Werte nach Formel 4.2 mit $N = 2^k$ mit $k \in \mathbb{N}$. Die FFT teilt die DFT in viele kleine DFTs und kann somit die Laufzeitkomplexität der Transformation von $\mathcal{O}(N^2)$ zu $\mathcal{O}(N \cdot \log_2(N))$.

(vgl. [4] Seite 1)

5 Programmierung des Demonstrators

In diesem Kapitel wird die Durchführung des Projekts mit den einzelnen Meilensteinen und Sackgassen beschrieben. Außerdem wird auch das Endprodukt genau beschrieben.

5.1 Entwicklungsumgebung

Als Entwicklungsumgebung wird Visual Studio Code (VSCode) verwendet. Dieser Editor bietet die Möglichkeit die Funktionalität mittels Erweiterungen zu vergrößern. In diesem Projekt wird die Erweiterung „Live Server“ verwendet, welche das Hosten eines lokalen Servers für HTML-Dateien ermöglicht. JavaScript Dateien, welche in die HTML-Datei eingebettet werden auf der Client-Seite mit dessen Hardware ausgeführt. Dies ermöglicht eine geringe Auslastung auf der Server-Seite. Weiter hat die Erweiterung „Live Server“ die Funktion „live reload“ die eine dynamische Entwicklung von Webseiten und Webanwendungen ermöglicht. Mit „live reload“ wird der Server automatisch neu geladen, sobald eine Änderung in den Quelldateien vorgenommen wird.

5.2 Programmierung

Der Demonstrator ist wie schon erwähnt in JavaScript programmiert. Um diesen auszuführen benötigt es einer Einbindung in eine HTML-Datei. Dies geschieht in dem Projekt in der so genannten index.html.

Genauere Erklärungen zu den einzelnen Dateien erfolgen im Folgenden zu den selbst programmierten Dateien. Zu Funktionen der Bibliotheken wird auf die jeweilige Dokumentation der Bibliothek verwiesen.

```

1  <!DOCTYPE html>
2
3  <head></head>
4  <body>
5      <div id = "demonstrator">
6          <p style="height:100vh; width:100vw;"></p>
7      </div>
8      <script src="microphone.js"></script>
9      <script src="sineGenerator.js"></script>
10     <script type="module" src="lil-gui.js"></script>
11     <script type="module" src="plotly-2.35.2.min.js"></script>
12     <script type="module" src="main.js"></script>
13
14 </body>
```

Listing 5.1: HTML-Datei des Demontsrators

5.2.1 index.html

Listing 5.1 ist die HTML Datei des Demonstrators. Hier wird in Zeile 5-7 das DIV-Element festgelegt und konfiguriert. Das DIV-Element wird benötigt, um als Referenz für die JavaScript-Datei zu dienen. Direkt unter dem DIV-Element in Zeile 8-12 werden die einzelnen JavaScript-Dateien und Bibliotheken, die für den Demonstrator benötigt werden, aufgerufen. Die zwei Dateien „microphone.js“ und „sineGenerator.js“ beinhalten die Objekte für den Mikrofon-Eingang und des Sinus-Generator, sowie dazugehörige Methoden zur Frequenzanalyse. Die Datei „lil-gui.js“ ist eine Bibliothek für Nutzereingaben in JavaScript. Diese erstellt mit gegebenen Parametern eine Nutzeroberfläche, mit der Änderungen an Parametern durchgeführt werden können. Die Datei „plotly-2.35.2.min.js“ ist die hier genutzte Bibliothek zur grafischen Ausgabe von 3-dimensionalen Plots in das oben genannte DIV-Element. Die Datei „main.js“ vereint bereits genannten Dateien und Bibliotheken. Diese nutzt die aus diesen gegebenen Funktionen und Methoden, um den Demonstrator zu realisieren.

5.2.2 microphone.js

„microphone.js“ ist in Listing 5.2 abgebildet. Diese ist die JavaScript-Datei, welche die Klasse „Microphone“, sowie die dazugehörigen Methoden enthält. Hier wird die

Standardbibliothek „Web Audio API“ verwendet. Der Konstruktor der Klasse initialisiert zuerst die Verbindung zum Mikrofon. Hierfür wird die Funktion „navigator.mediaDevices.getUserMedia()“ mit Parameter „audio: true“ aufgerufen. Dies führt dazu, dass der Browser eine die Mikrofonfreigabe bei dem Nutzer erfragt. Wird die Freigabe erteilt, so kann die Verbindung im Konstruktor weiter initialisiert werden und das Signal des Mikrofons wird auf den „stream“ gelegt. Im Konstruktor wird dann eine AudioContext-Instanz erstellt, welche genutzt wird, um das Mikrofonsignal aus „stream“ mit der Variablen „this.microphone“ verknüpft. Ab Zeile 8 wird für die AudioContext-Instanz eine Frequenzanalyse initialisiert. In den Zeilen 9 und 10 werden zwei Parameter festgelegt. „fftSize“ sagt an, mit wie vielen Punkten die Fouriertransformation aufgelöst ist. Hier ist der Wert 512 festgelegt, da bei der Ausgabe der Frequenzen die Anzahl der Datenpunkte der Hälfte der „fftSize“ entspricht. Die „smoothingTimeConstant“ bestimmt eine Konstante, mit welcher die Frequenzanalyse über die Zeit geglättet wird. Dies ist hier unerwünscht und deshalb ist der Wert auf „0“ gesetzt, da der Default-Wert bei „0,8“ liegt. In Zeile 12 wird ein „dataArray“ erstellt mit der Länge „frequencyBinCount“. Dies hat den Wert 256, da die Länge der Ausgabe der FFT die Hälfte der Anzahl der Datenpunkte entspricht. „dataArray“ wird in der Methode „getFrequencyData()“ zur Ausgabe der Frequenzanalyse verwendet. In Zeile 13 wird dann der Mikrofonstream aus der Variablen „microphone“ mit dem „analyser“ verbunden. Somit bekommt die Frequenzanalyse die Daten direkt vom Mikrofonstream. Falls die Initialisierung des Mikrofonobjekts (durch ablehnen der Mikrofonberechtigung) nicht durchgeführt werden kann, so wird eine Errormeldung ausgeworfen.

Die Methode „getFrequencyData()“ wird ab Zeile 19 initialisiert. Hier wird mit der Funktion „getByteFrequencyData()“ des Analysers die Werte der FFT in das Array „dataArray“ geladen. Die Ausgabe des der Frequenzen erfolgt dann über das Array „frequencyData“.

```

1 class Microphone {
2     constructor(){
3         this.initialized = false;
4         navigator.mediaDevices.getUserMedia({audio: true})
5         .then(function(stream){
6             this.audioContext = new AudioContext();
7             this.microphone = this.audioContext.

```

```

8      this.analyser = this.audioContext.createAnalyser();
9      this.analyser.fftSize = 512;
10     this.analyser.smoothingTimeConstant = 0;
11     const bufferLength = this.analyser.
12         frequencyBinCount;
13     thisdataArray = new Uint8Array(bufferLength);
14     this.microphone.connect(this.analyser);
15     this.initialized = true;
16   }.bind(this)).catch(function(err){
17     alert(err);
18   })
19   getFrequencys(){
20     this.analyser.getByteFrequencyData(thisdataArray);
21     let frequencys = [...thisdataArray];
22
23     return frequencys;
24   }
25 }
```

Listing 5.2: Objekt-Datei zur Nutzung des Mikrofons über den Browser

5.2.3 sineGenerator.js

„sineGenerator.js“ wird in Listing 5.3 abgebildet. Diese Datei enthält die Klasse des „SineGenerators“ und ist ähnlich aufgebaut wie die Klasse des Mikrofons. Unterschiede finden sich hier in der Zeile 7-9, 15 und 16 im Konstruktor, sowie in der Methode „setFrequency“ ab Zeile 28. Im Konstruktor in Zeile 7 wird als Funktion der AudioContext-Instanz ein Oszillatior generiert und als „sineGenerator“ gespeichert. In der Zeile 8 wird die Wellenform als „sine“ (Sinus) festgelegt. Mit „sineGenerator.frequency.setValueAtTime()“ wird die Frequenz der Sinus-Generators festgelegt. Der Parameter „f“ ist hierbei die Frequenz und der Parameter „this.audioContext.currentTime“ beschreibt die aktuelle Zeit, da die Funktion „setValueAtTime()“ eine Zeitangabe benötigt. In Zeile 15 wird der Sinus-Generator mit dem Frequenzanalyser verknüpft und in der folgenden Zeile gestartet. Die Methode

„setFrequency(f)“ erhält bei Aufruf den Parameter „f“ und führt die Funktion „frequency.setValueAtTime()“ mit dem Parameter „f“ und der aktuellen Zeit aus, um die Frequenz des Generators zu ändern.

```
1 class SineGenerator {
2     constructor(f){
3         this.initialized = false;
4         navigator.mediaDevices.getUserMedia({audio: true})
5         .then(function(stream){
6             this.audioContext = new AudioContext();
7             this.sineGenerator = this.audioContext.
8                 createOscillator();
9             this.sineGenerator.type = "sine";
10            this.sineGenerator.frequency.setValueAtTime(f, this
11                .audioContext.currentTime);
12            this.analyser = this.audioContext.createAnalyser();
13            this.analyser.fftSize = 512;
14            this.analyser.smoothingTimeConstant = 0;
15            const bufferLength = this.analyser.
16                frequencyBinCount;
17            thisdataArray = new Uint8Array(bufferLength);
18            this.sineGenerator.connect(this.analyser);
19            this.sineGenerator.start();
20            this.initialized = true;
21        }).bind(this)).catch(function(err){
22            alert(err);
23        })
24    }
25    getFrequencys(){
26        this.analyser.getByteFrequencyData(thisdataArray);
27        let frequencys = [...thisdataArray];
28
29        return frequencys;
30    }
31    setFrequency(f){
32        this.sineGenerator.frequency.value = f;
33    }
34}
```

```

29         this.sineGenerator.frequency.setValueAtTime(f, this.
30             audioContext.currentTime);
31     }

```

Listing 5.3: Objekt-Datei zur Nutzung des Sinus-Generators

5.2.4 main.js

„main.js“ ist das Hauptprogramm des Demonstrators und vereint alle Klassen und Bibliotheken, die hierfür benötigt werden. Am Anfang der „main.js“ befindet sich der Programmabschnitt, der in Listing 5.4 zu sehen ist. Hier wird eine Instanz der Klasse „GUI“ aus der Bibliothek „lil-gui.js“ erstellt und konfiguriert. Eine GUI bietet die Möglichkeit einer Nutzerinteraktion mit der Webseite. Die hier erstellte Instanz „gui“ benötigt ein Objekt mit Attributen, die von der GUI verändert oder als Funktion ausgeführt werden. Dieses Objekt wird als „controls“ initiiert und enthält die in Zeile 4-8 dargestellten Attribute. Diese Attribute werden ab Zeile 10 der „gui“-Instanz hinzugefügt. Die Bibliothek „lil-gui.js“ ändert die Art des Nutzereingabe anhand des Typs des Attributs. Ist das Attribut eine Funktion, so wird in der GUI ein Taster angezeigt. Ist das Attribut eine Zahl und es wird ein Intervall angegeben, so ist in der GUI ein Schieberegler eingebunden.

```

1 let gui = new lil.GUI();
2
3 var stop = false;
4 var controls = {
5     mikrofon: function(){startMikrofon()},
6     sinus: function(){startSinus()},
7     frequenz: 1,
8     stop: function(){stop = true;},
9     clear: function(){clearPlot()}
10 }
11 gui.add(controls, 'mikrofon');
12 gui.add(controls, 'sinus');
13 gui.add(controls, 'frequenz', 0, 20000).onChange( value => {
14     sinus.setFrequency(value);

```

```

15 });
16 gui.add(controls, 'stop');
17 gui.add(controls, 'clear');
```

Listing 5.4: Programmteil zur Initialisierung und Konfigurierung der GUI mit `gui.js`

Der nächste Programmabschnitt aus Listing 5.5 beginnt mit dem Verknüpfen der Variablen „DEMONSTRATOR“ mit dem DIV-Element der HTML-Datei. in Zeile 3 Wird ein 2-dimensionales Array als Platzhalter für die Werte der DFT erstellt. Dieses Array wird dann in das Array aus Objekten „data“ eingebunden. Zudem wird der Typ des Plots in Zeile 7 als „surface“ Festgelegt, was in ein Surfaceplot resultiert. Die beiden weiteren Attribute „showscale“ und „displayModeBar“ haben nur kosmetische Effekte.

```

1 var DEMONSTRATOR = document.getElementById('demonstrator');

2

3 let z_data = new Array(200).fill(0).map(()=>new Array(256).fill
    (0));

4

5 var data = [
6     z: z_data,
7     type: 'surface',
8     showscale: false,
9     displayModeBar: false
10];
```

Listing 5.5: Programmteil zum Erstellen des 2-dimensionalen Daten Arrays, sowie Auswahl des DIV-Elements.

Der Programmausschnitt aus Listing 5.6 enthält die Konfiguration des Plots. Hier werden einige Elemente, die von „Plotly.js“ standartmäßig angezeigt werden, ausgeblendet.

```

1 var defaultPlotlyConfiguration = {
2     modeBarButtonsToRemove: [
3         'pan',
4         'orbitRotation',
```

```

5      'tableRotation',
6      'resetCameraDefault3d',
7      'resetCameraLastSave3d',
8      'zoom',
9      'toImage',
10     'sendDataToCloud',
11     'autoScale2d',
12     'hoverClosestCartesian',
13     'hoverCompareCartesian',
14     'lasso2d',
15     'select2d'
16   ],
17   displaylogo: false,
18   showTips: false
19 };

```

Listing 5.6: Programmteil zu Konfigurierung der Plotumgebung.

Der folgende Programmausschnitt Listing 5.7 enthält das Layout des Plots , sowie die initiale Ausgabe des Plots in der letzten Zeile. Das Layout umfasst den Titel des Plots „Demonstrator für die diskrete Fouriertransformation“, sowie Einstellungen zur Szene. In der Szene wird definiert von welchem Punkt aus initial auf das Plot geschaut wird. Außerdem wird hier auch die Skalierung der Achsen in Zeile 8 festgelegt, sodass sich die Größe des Plots nicht dynamisch mit den Werten verändert. In Zeile 9-15 wird die X-Achse konfiguriert. Auf dieser werden die Frequenzen abgetragen. Deshalb erhält die X-Achse den Titel „Frequenz“ und die Achsenbeschriftung wird in Zeile 13 und 14 angepasst. Für die Y-Achse wird der Titel zu „Zeit“ festgelegt. In Zeile 22-24 wird der gesamte Plot auf die Größe des DIV-Elements skaliert.

```

1 var layout = {
2   title: {
3     text: 'Demonstrator für die diskrete Fouriertransformation'
4   },
5   scene: {
6     camera: {eye: {x: 1.87, y: 0.88, z: 0.64}},
7     aspectmode: "manual",
8     aspectratio: {x: 1, y: 2, z: 0.2},

```

```

9      xaxis: {
10        title: {
11          text: 'Frequenz',
12        },
13        tickvals: [0, 53, 106, 159, 212],
14        ticktext: [0, 5000, 10000, 15000, 20000]
15      },
16      yaxis: {
17        title: {
18          text: 'Zeit',
19        }
20      },
21    },
22    autosize: true,
23    width: DEMONSTRATOR.clientWidth,
24    height: DEMONSTRATOR.clientHeight,
25  };
26
27 Plotly.newPlot(DEMONSTRATOR, data, layout,
  defaultPlotlyConfiguration);

```

Listing 5.7: Programmteil zur Konfigurierung des Plot-Layouts und erstes Anzeigen des Plots in der HTML-DIV

Der Programmabschnitt aus Listing 5.8 enthält die Initialisierungen der Instanzen der Klassen „Microphone“ und „SineGenerator“ in Zeile 1 und 8. Zeile 2-6 und Zeile 9-13 enthalten die Funktionen, die durch eine entsprechende GUI-Interaktion ausgeführt werden. Im Funktionskörper wird die Funktion „redraw()“ mit der entsprechende Instanz der Klasse ausgeführt.

```

1 const microphone = new Microphone();
2 function startMikrofon(){
3
4   redraw(microphone);
5
6 }
7

```

```

8 const sinus = new SineGenerator(controls.frequenz);
9 function startSinus(){
10
11     redraw(sinus);
12
13 }

```

Listing 5.8: Programmteil zur Erstellung von Objektinstanzen und Aufruf der `redraw()`-Funtion mit entsprechender Instanz.

Der folgende Programmabschnitt Listing 5.9 enthält die Funtion „`redraw()`“, sowie die Initialisierung der Variablen „`frequenzen`“ in Zeile 1. Die Funktion „`redraw()`“ ruft zu beginn die Methode „`getFrequency()`“ des entsprechenden Objekts auf mit dem es als Parameter Aufgerufen wurde und speichert die Frequenzwerte in dem Array „`frequenzen`“ zwischen. In Zeile 8 und 9 wird das 2-dimensionale Array „`z_data`“ verändert. In Zeile 8 wird der „`head`“ entfernt und in Zeile 9 wird das Array „`frequenzen`“ an das Ende angehängt. Dem Objekt „`update`“ wird das Array „`z_data`“ übergeben und wird benötigt, um den Plot mit der Plotly-Funktion „`update()`“ zu aktualisieren. Die Funtion „`setTimeout()`“ ist eine Standardfunktion von JavaScript, welche einen Funktionskörper, sowie einen Parameter hinter dem Funktionskörper enthält. Dieser Parameter bestimmt, nach welcher Zeit (in Millisekunden) der Funktionskörper ausgeführt werden soll. Hier ist der Parameter auf 10ms festgelegt. Da der Funktionskörper die Funktion „`redraw()`“ aufruft und die Funktion „`setTimeout()`“ in von der Funktion „`redraw()`“ ausgeführt wird, so ruft die Funktion „`redraw()`“ rekursiv sich selbst auf, bis die Variable „`stop`“ den boolschen Wert „`false`“ enthält. Die Variable „`stop`“ wird durch die GUI-Interaktion „`stop`“ verändert.

```

1 var frequenzen;
2
3 function redraw(srcObject){
4
5
6     frequenzen = srcObject.getFrequency();
7
8     z_data = z_data.slice(1);
9     z_data.push(...[frequenzen]);
10

```

```

11     var update = {
12         z: [z_data]
13     }
14
15     Plotly.update(DEMONSTRATOR, update, 0);
16
17     setTimeout(function() {
18         if (stop === true) {
19             stop = false;
20             return;
21         }
22         redraw(srcObject);
23     }, 10);
24 }
```

Listing 5.9: Programmteil mit abruf der Frequenzdaten der entsprechenden Objektinstanz, sowie einem Rekursiven Aufruf des eigenen Funtion.

Der letzte Programmabschnitt Listing 5.10 enthält die Funktion „clearPlot()“. Diese wird durch die GUI-Interaktion „clear“ ausgeführt und leert das Array „z_data“, sowie zeichnet das geleerte Plot neu.

```

1 function clearPlot(){
2     z_data = new Array(200).fill(0).map(()=>new Array(256).fill
3         (0));
4     var update = {
5         z: [z_data]
6     }
7
8     Plotly.update(DEMONSTRATOR, update, 0);
9 }
```

Listing 5.10: Programmteil mit Funktion zum Leeren des Plots.

5.3 Ergebnis

Im Folgenden ist das grafische Ergebnis der programmierten Elemente aus dem vorangehenden Kapitel 5.2 zu sehen. In Abbildung 5.1 ist der Ausgangszustand bei starten des Programms zu sehen. Hier ist das Zugriffsrecht auf das Mikrofon erteilt. Zudem bietet die Abbildung 5.1 eine Übersicht über das gesamte Fenster des Browsers und somit ist die Anordnung der einzelnen Elemente (Plot, GUI) zu sehen. In diesem Anfangszustand lässt sich mit der Maus der Plot noch frei drehen.

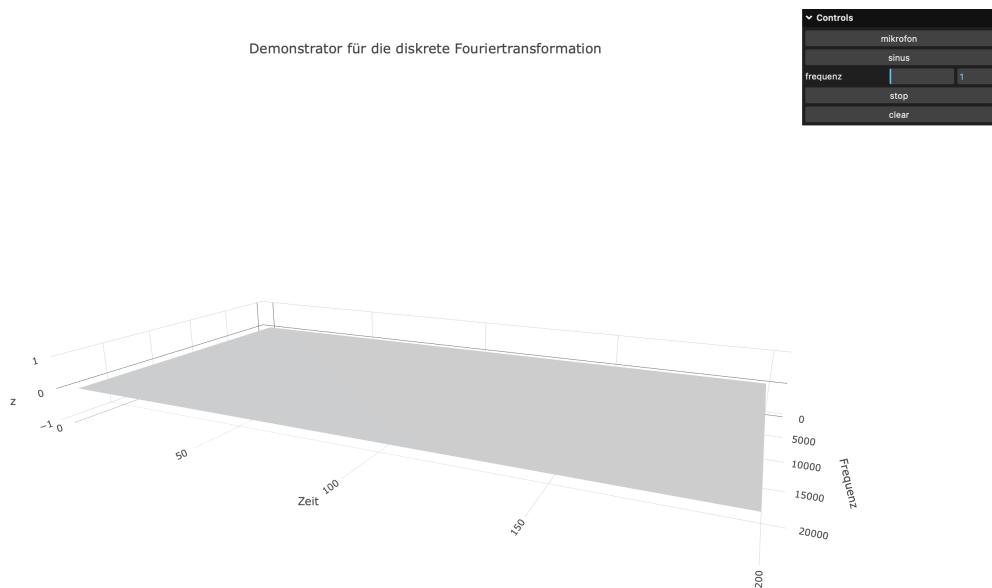


Abbildung 5.1: Anfangszustand des Demonstrators nach Erlauben des Mikrofonzugriffs.

Die Abbildung 5.2 zeigt die Ausgabe des Plots nach Betätigung des „mikrofon“-Tasters in der GUI nach wenigen Sekunden. Hier sind die einzelnen Frequenzanalysen ca. alle 10ms abgebildet. Die in Kapitel 5.2.2 erwähnte Auflösung der FFT mit 256 Datenpunkten ist aus Performancegründen gewählt.

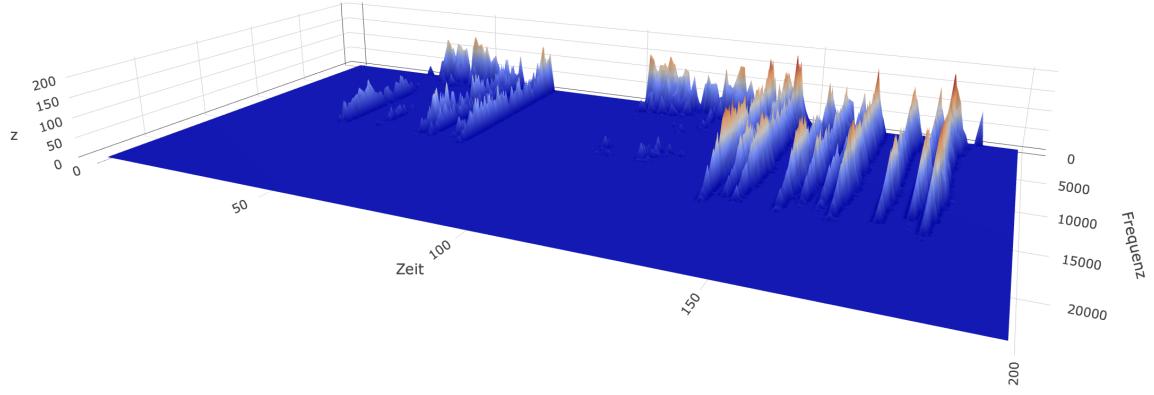


Abbildung 5.2: Ausgabe des Plots mit aktivem Mikrofon.

Die folgender Abbildung 5.3 stellt den Plot dar, welcher nach betätigen des „sinus“-Tasters zu sehen ist. Hier ist die Frequenz während der Aufzeichnung variiert um zu zeigen, dass die Frequenzanalyse korrekt funktioniert.

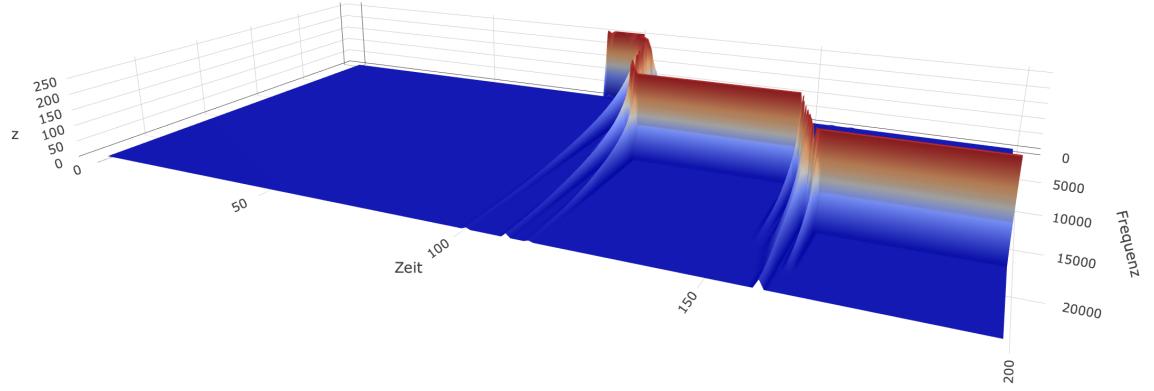


Abbildung 5.3: Ausgabe des Plots mit aktiverem Sinus-Generator.

Die Abbildung 5.4 zeigt die GUI als Nahaufnahme. Hier sind die Taster „mikrofon“, „sinus“, „stop“, „clear“, sowie den Schieberegler für die Frequenz „frequenz“ zu sehen. Durch betätigen des Tasters „mikrofon“ oder „sinus“ wird die Frequenzanalyse der jeweiligen Klassen-Instanz gestartet. Wichtig ist hier zu beachten, dass vor dem Wechsel zwischen Mikrofoneingang und Sinus-Generator die „stop“-Taste gedrückt werden muss, da es sonst zu ungewollten Fehlern kommen kann. Während der Frequenzanalyse des Sinus-Generators kann im Betrieb die Frequenz mit dem Schieberegler „frequenz“

verändert werden. Hier bedarf es keiner Pausierung der Analyse. Die Taste „clear“ löst, wie schon in Kapitel 5.2.4 zu Listing 5.10 beschrieben, den Reset des Plots aus.

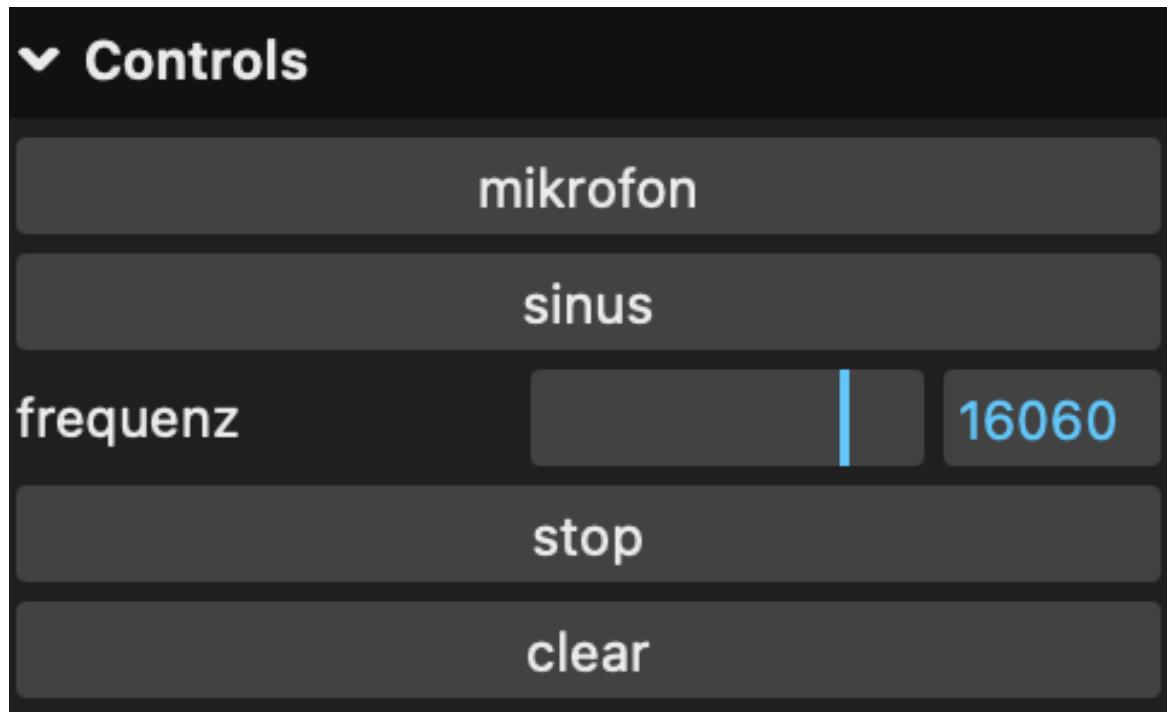


Abbildung 5.4: GUI der Ausgabe

6 Zusammenfassung

Dieses Projekt hat als Ziel die interaktive Darstellung der DFT in einer Webanwendung und wird im Rahmen des Studiengangs Elektrotechnik an der Dualen Hochschule Baden-Württemberg Mannheim durchgeführt. Die Zielsetzung der Arbeit konzentriert sich auf folgende Punkte:

- Verarbeiten von Audiosignalen eines Mikrofons.
- Simulieren eines Sinus-Generators mit variabler Frequenz.
- Visualisierung der FFT des Audiosignals und des Sinus-Generators in einem 3-dimensionalen Plot.

Umgesetzt werden diese Ziele folgendermaßen:

- Nutzung der Web Audio API für die Mikrofoneingabe, den Sinus-Generator und die Umsetzung der FFT mit 256 Datenpunkten.
- 3D-Visualisierung der FFT mittels der JavaScript Bibliothek „Plotly.js“.
- Interaktion mittels GUI mithilfe der Bibliothek „lil-gui.js“

Die Programmstruktur unterteilt sich in mehrere Dateien.

- index.html
- main.js
- microphone.js
- sineGenerator.js

Die HTML-Datei wird benötigt um die JavaScript-Dateien im Browser ausführen zu können. Die Datei main.js vereint die verwendeten Bibliotheken und die zwei weiteren JavaScript-Dateien um den Demonstrator zu realisieren.

Ergebnis dieser Arbeit ist ein intuitive Visualisierung der FFT mittels JavaScript im Browser mit Wählbarer Signalquelle, sowie der Möglichkeit einer Löschung des Plots und beenden der Analyse.

Verbessert werden kann das Ergebnis mit einer Performancesteigerung um eine höhere Auflösung der FFT zu erlauben, sowie eines dynamisch einstellbaren Abtastzeitraums, um die Eigenheiten der DFT besser darstellen zu können.

Literatur

- [1] Ottmar Beucher. *Signale und Systeme: Theorie, Simulation, Anwendung: Eine beispielorientierte Einführung mit MATLAB*. de. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019. ISBN: 978-3-662-58043-1 978-3-662-58044-8. DOI: 10.1007/978-3-662-58044-8. URL: <http://link.springer.com/10.1007/978-3-662-58044-8> (besucht am 29.12.2024).
- [2] d3js. *What is D3*. URL: <https://d3js.org/what-is-d3> (besucht am 01.01.2025).
- [3] Julien Funk. *Quick evaluation of some FFT jobs in Javascript*. Tabelle. Dez. 2017. URL: <https://github.com/j-funk/js-dsp-test/blob/master/screenshot.png> (besucht am 31.12.2024).
- [4] Mahsa Shirzadian Gilan und Behrouz Maham. “Optimized power and speed of Split-Radix, Radix-4 and Radix-2 FFT structures”. en. In: *EURASIP Journal on Advances in Signal Processing* 2024.1 (Aug. 2024), S. 81. ISSN: 1687-6180. DOI: 10.1186/s13634-024-01178-4. URL: <https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-024-01178-4> (besucht am 31.12.2024).
- [5] Fedor Indutny u. a. *The fastest JS Radix-4/Radix-2 FFT implementation*. Jan. 2021. URL: <https://github.com/indutny/fft.js/> (besucht am 31.12.2024).
- [6] Christian Karpfinger. *Hilfe Mathematik in Rezepten: Begriffe, Sätze und zahlreiche Beispiele in kurzen Lerneinheiten*. de. Berlin, Heidelberg: Springer Berlin Heidelberg, 2022. ISBN: 978-3-662-63304-5 978-3-662-63305-2. DOI: 10.1007/978-3-662-63305-2. URL: <https://link.springer.com/10.1007/978-3-662-63305-2> (besucht am 01.01.2025).
- [7] Mozilla. *AnalyserNode*. Okt. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode> (besucht am 01.01.2025).
- [8] Mozilla. *AudioContext*. Juli 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/API/AudioContext> (besucht am 01.01.2025).

- [9] Mozilla. *Fourier Transform*. URL: <https://webaudio.github.io/web-audio-api/#fourier-transform> (besucht am 01.01.2025).
- [10] Mozilla. *MediaDevices*. Feb. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices> (besucht am 01.01.2025).
- [11] Mozilla. *OscillatorNode*. Juli 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/API/OscillatorNode> (besucht am 01.01.2025).
- [12] Stack Overflow. *Most used programming languages among developers worldwide as of 2024*. Juli 2024. URL: <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/> (besucht am 29.12.2024).
- [13] Plotly. *Function Reference in JavaScript*. URL: <https://plotly.com/javascript/plotlyjs-function-reference/> (besucht am 01.01.2025).
- [14] Plotly. *Plotly JavaScript Open Source Graphing Library*. 2024. URL: <https://plotly.com/javascript/> (besucht am 01.01.2025).
- [15] Jonathon Simpson. *How JavaScript Works: Master the Basics of JavaScript and Modern Web App Development*. en. Berkeley, CA: Apress, 2023. ISBN: 978-1-4842-9737-7 978-1-4842-9738-4. DOI: 10.1007/978-1-4842-9738-4. URL: <https://link.springer.com/10.1007/978-1-4842-9738-4> (besucht am 29.12.2024).
- [16] Boris Smus. *Web Audio API: advanced sound for games and interactive apps*. eng. 1. ed. Beijing: O'Reilly, 2013. ISBN: 978-1-4493-3268-6.
- [17] Threejs. *Fundamentals*. URL: <https://threejs.org/manual/#en/fundamentals> (besucht am 01.01.2025).
- [18] Erik Werner und Ahmed Hassan. *microphonejs*. März 2018. URL: <https://github.com/flowkey/microphonejs> (besucht am 01.01.2025).
- [19] Wikibrief. *Spektrogramm*. URL: https://upload.wikimedia.org/wikipedia/commons/0/08/3D_battery_charger_RF_spectrum_over_time.jpg (besucht am 01.01.2025).

A Anhang

A.1 Statistik Programmiersprachen

Abbildung A.1: Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. [12]

