DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Studienarbeit

T3100

Demonstrator für die diskrete Fourier-Transformation in JavaScript

Studiengang Elektrotechnik

Dualen Hochschule Baden-Württemberg Mannheim

von

Niklas Herhoffer

Bearbeitungszeit:	12 Wochen
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 (Abstract)

Hier können Sie die Kurzfassung der Arbeit schreiben. Beispielsweise behandelt diese Arbeit eine Darstellung, wie die Bearbeitung von wissenschaftlichen Texten auf Basis des LaTeX-Textsatzsystems mit Nutzung des LyX-Editors umgesetzt werden kann. Durch diesen exemplarischen Text werden sowohl einige grundlegende Möglichkeiten dargestellt, die sich durch Nutzung der Software ergeben, als auch einige Hilfen gegeben, wie eine Umsetzung erfolgen kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmlistings

Abkürzungsverzeichnis

VSCode	Visual Studio Code
DFT	Diskrete Fouriertransformation
FFT	Fast Fouriertransformation
GUI	Graphical User Interface

1 Einleitung

Dieses Kapitel unterteilt sich in 5 Unterkapitel, in welchen die Wichtigkeit des Projektes anhand der Aufgaben- und Zielstellung, sowie der Einführung in das Projekt beschrieben wird. Zudem wird hier der Stand der Technik und der Aufbau der Arbeit im Gesamten behandelt.

1.1 Einführung in das Projekt

Die Fouriertransformation ist besonders für Studierende des Das Projekt befasst sich mit der Endwicklung eines Demonstrators für die diskrete Fourier-Transformation in der Programmiersprache JavaScript.

1.2 Aufgabenstellung

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 diskreten Fourier-Transformation ist die Möglichkeit eines Mikrofon-Input, als die Möglichkeit eines Sinussignaks mit veränderlicher Frequenz zu implementieren. Mit dem Sinussignal wird die Funktionabilität des Demonstrators überprüft und vorgeführt. Der Mikrofoninput wird verwendet um die genauigkeit des Demonstrators zu aufzuzeigen. Der Schematische Aufbau des Programms wird in der folgenden Abbildung aufgeführt. Hier ist zu sehen, dass zwischen den Signaleingängen "Mikrofon" und "Sinus" gewählt erden kann. Das resultierende Signal wird daraufhin mit Hilfe einer Bibliothek für Disktreten Forier-Transformation in Zeit und Wert dikretisiert und daraufhin transformiert. Daraus resultiert dann das Zeitliche Signal in spektraler auflösung. 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. Diese zeigt den grafischen Output eines 3-dimensionalen Frequenzanalyse. Solch eine Analyse ist besonders für Audio-Signale sinnvoll.

1.3 Ziele der Arbeit

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 verscheidenen Browsern auf Funktion zu testen.

1.4 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.

1.4.1 Bibliothek für 3D-Visualisierungsbibliothek

Für die Visualisierung der Daten in einem 3-dimensionalen Plot stehen einige verschiedene Bibliotheken für JavaScript zur verfügung. Eine Auswahl dieser sind

Three.JS

d3.JS

Plotly.js

Three.js:

Three.js ist eine Bibliothek, mit welcher einfach 3D Animationen in JavaScript erstellt und angezeigt werden können. Hier kann mit dem nötigen Aufwand alles genau so dargestellt werden, wie man es möchte.

d3.js:

Diese Bibliothek ist eine mächtige low level Bibliothek für die Visualisierung von Daten. d3 steht hier für "Data Driven Documents". Mit dieser Bibliothek lassen sich 2D, sowie 3D plots von Daten erstellen. Heir jedoch mit, im Vergleich zu Three.js, hohen Aufwand.

Plotly.js:

Diese Bibliothek steht für verschiedene Programmiersprachen zur verfügung und konzentriert sich nur auf das Anzeigen und Animieren von Plots im speziellen. Der Fokus bei dieser Bibliothek liegt also nicht auf das Abdecken von möglichst vielen Anwendungsfällen, sondern darauf, Plots mit möglichst wenig Aufwand und in kurzer Zeit darstellen zu können.

Für dieses Projekt ist letztendlich Plotly.js für die Visulaisierung der Daten gewählt worden. Diese Entscheidung ist auf den einfachen Aufbau und die Einfache Bedienung zurückzuführen.

1.4.2 FFT Bibliotheken

Um das Audiosignal im Spektrum betrachten zu können wird eine Fast Fourier Transformation (FFT) durchgeführt. Dies kann nach verschiedenen Algorithmen geschehen. Solche Algorithmen sind häufig in verschiedenen Bibliotheken eingebunden und können somit leicht verwendet werden.

FFt.js

KissFFT

Web Audio APIs MediaDevices

Diese Bibliothek beinhaltet neben einer Audio Eingabe zusätzlich auch die Möglichkeit einer Frequenzanalyse des Audio Inputs. Hier ist jedoch darauf zu achten, dass diese Web API nur funktioniert, wenn auf den Web Server, auf dem das JavaScript-Programm läuft, mit "https" zugegriffen wird. Diese Bibliothek nutzt hier den Algorithmus

1.4.3 Bibliothek für den Audio Input

Um die Eingabe des Mikrofons zu ermöglichen muss diese implementiert werden. Hierfür kann mit einer Bibliothek gearbeitet werden oder die oben genannte Web Audio API verwendet werden. In dieser Arbeit wird auf die Verwendung einer zusätzlichen Bibliothek verzichtet, da eine solche Bibliothek wie z.B. Microphone.js nur eine leicht einfachere Verwendung der Web Audio API darstellt. Da jedoch die Signalverarbeitung mittels FFT über die Web Audio API leichter zu implementieren, wenn auch

dessen eigener AudioInput genutzt wird, wird deshalb darauf zurückgegriffen. Zudem ist mit der Web Audio API Browserübergreifend die Funktionalität sichergestellt.

2 Technische Grundlagen

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

2.1 JavaScript

JavaScript ist eine Programmiersprache, welche 1995 erschienen ist. Sie wird vor allem dazu genutzt, um Webseiten dynamischer und interaktiver zu gestallten, bietet jedoch auch die Möglichkeit Desktop-Apps und Backend-APIs zu erstellen.Zudem zählt sie zu den beliebtesten Programmiersprachen weltweit . (vgl. [3] Seite 1)

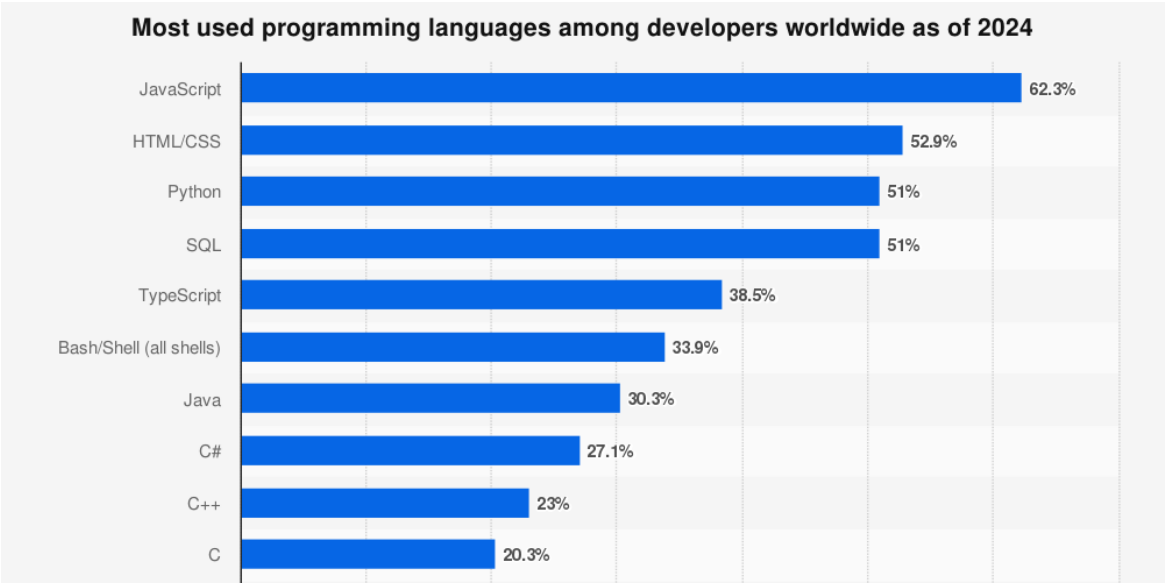

Abbildung 2.1: Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. (Ausschnitt aus A.1) [2]

Die Beliebtheit von JavaScript wird durch die Statistik in Abbildung 2.1 bestätigt. Diese zeigt das Ergebnis einer Umfrage von Stack Overflow im Juli 2024. 62,3% der

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

JavaScript bietet einige Vorteile, welche das Programmieren vereinfachen. Diese sind zum Beispiel die dynamische Typisierung von Variablen. Denn hier ist ähnlich wie bei Python der Typ einer Variable nicht weiter wichtig. Dieser wird vom Interpreter automatisch nach Inhalt der Variable festgelegt.

2.2 Diskrete Fouriertransformation

Die Diskrete Fouriertransformation (DFT) ermöglicht es Rechnern eine Fouriertransformation durchzuführen. Die herkömmliche Fouriertransformation benötigt immer alle Abtastwerte $(x_n)_{n\epsilon\mathbb{Z}}$.Die DFT kann jedoch nur bei zeit- und wertdiskreten Signalen verwendet werden. Hier wird also eine endliche Anzahl von Werten betrachtet:

$$(x_n)_{n=0}^{N-1} \tag{2.1}$$

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

Für die DFTergibt sich die Formel:

$$(x_n)_{n=0}^{N-1} \to X_{DFT} := \sum_{n=0}^{N-1} x_n \cdot e^{-jn\Omega_m} \tag{2.2}$$

mit

$$\Omega_m := \frac{2\pi m}{N} \tag{2.3}$$

$$m = 0, \ldots N-1 \tag{2.4}$$

Die Anzahl der Ergebnisse einer DFTist immer genau so groß wie die Anzahl der Abtastwerte, die der DFTübergeben werden.

(vgl.[1]Seite 318f.)

3 Demonstrator für die diskrete Fouriertransformation

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

3.1 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, wie schon in Kapitel ... erwähnt, 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.

3.2 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="sinusGenerator.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 3.1: HTML-Datei des Demontsrators

3.2.1 index.html

Listing 3.1 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 „sinusGenerator.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.

3.2.2 microphone.js

„microphone.js“ ist in Listing 3.2 abgebildet. Diese ist die JavaScript-Datei, welche das Objekt für den Mikrofoneingang, sowie die dazugehörigen Methoden enthält. Hier

wird die Standardbibliothek „Web Audio API“ verwendet. Der Konstruktor des Objekts 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 Fast Fouriertransformation (FFT) die hälfte der Anzahl der Datenpunkte entspricht. „dataArray“ wird in der Methode „getFrequencys“ 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 „getFrequencys()“ 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 „frequencys“.

```
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.
```

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

Listing 3.2: Objekt-Datei zur Nutzung des Mikrofons über den Browser

3.2.3 sinusGenerator.js

„sinusGenerator.js“ wird in Listing 3.3 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 Oszillator 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 Funtion „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.

```
class SineGenerator {
    constructor(f){
        this.initialized = false;
        navigator.mediaDevices.getUserMedia({audio: true})
        .then(function(stream){
            this.audioContext = new AudioContext();
            this.sineGenerator = this.audioContext.
               createOscillator();
            this.sineGenerator.type = "sine";
            this.sineGenerator.frequency.setValueAtTime(f, this
               .audioContext.currentTime);
            this.analyser = this.audioContext.createAnalyser();
            this.analyser.fftSize = 512;
            this.analyser.smoothingTimeConstant = 0;
            const bufferLength = this.analyser.
               frequencyBinCount;
            this.dataArray = new Uint8Array(bufferLength);
            this.sineGenerator.connect(this.analyser);
            this.sineGenerator.start();
            this.initialized = true;
        }.bind(this)).catch(function(err){
            alert(err);
        })
    }
    getFrequencys(){
        this.analyser.getByteFrequencyData(this.dataArray);
        let frequencys = [...this.dataArray];

        return frequencys;
    }
    setFrequency(f){
```

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

Listing 3.3: Objekt-Datei zur Nutzung des Sinus-Generators

3.2.4 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 3.4 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 3.4: Programmteil zur Initzialisierung und Konfigurierung der GUI mit lil-gui.js

Der nächste Programmabschnitt aus Listing 3.5 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 3.5: Programmteil zum Erstellen des 2-dimensionalen Daten Arrays, sowie Auswahl des DIV-Elements.

Der Programmausschnitt aus Listing 3.6 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 3.6: Programmteil zu Konfigurierung der Plotumgebung.

Der folgende Programmausschnitt Listing 3.7 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 3.7: Programmteil zur Konfigurierung des Plot-Layouts und erstes Anzeigen des Plots in der HTML-DIV

Der Programmabschnitt aus Listing 3.8 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 3.8: Programmteil zur Erstellung von Objektinstanzen und Aufruf der redraw()-Funtion mit entsprechender Instanz.

Der folgende Programmabschnitt Listing 3.9 enthält die Funtion „redraw()“, sowie die Initialisierung der Variablen „frequenzen“ in Zeile 1. Die Funktion „redraw()“ ruft zu beginn die Methode „getFrequencys()“ 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 duch die GUI-Interaktion „stop“ verändert.

```
1  var frequenzen;
2
3  function redraw(srcObject){
4
5
6      frequenzen = srcObject.getFrequencys();
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 3.9: Programmteil mit abruf der Frequenzdaten der entsprechenden Objektinstanz, sowie einem Rekursiven Aufruf des eigenen Funtion.

Der letzte Programmabschnitt Listing 3.10 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
            (0));
3       var update = {
4           z: [z_data]
5       }
6
7       Plotly.update(DEMONSTRATOR, update, 0);
8   }
```

Listing 3.10: Programmteil mit Funktion zum Leeren des Plots.

3.3 Ergebnis

Im Folgenden ist das grafische Ergebnis der Programmierten Elemente aus dem vorangehenden Kapitel zu sehen. In Abbildung 3.1 ist der Ausgangszustand bei starten des Programms zu sehen. Hier ist das Zugriffsrecht auf das Mikrofon erteilt. Zudem bietet die Abbildung 3.1 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.

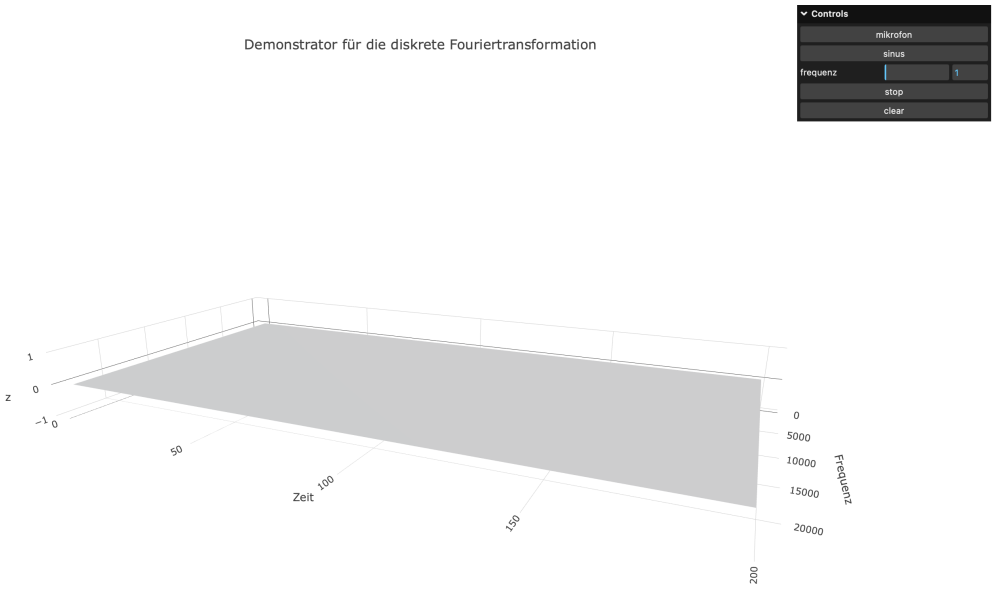

Abbildung 3.1: Anfangszustand des Demonstrators nach Erlauben des Mikrofonzugriffs.

Die Abbildung 3.2 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. Wie schon in Kapitel ... erwähnt ist hier die Auflösung der Frequenzanalyse aus Performance-Gründen auf 256 Punkte beschränkt.

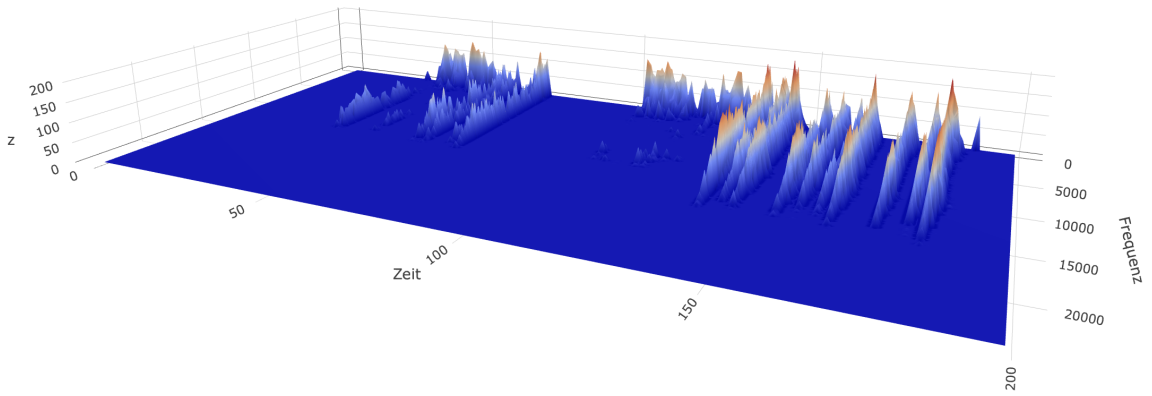

Abbildung 3.2: Ausgabe des Plots mit aktivem Mikrofon.

Die folgender Abbildung 3.3 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.

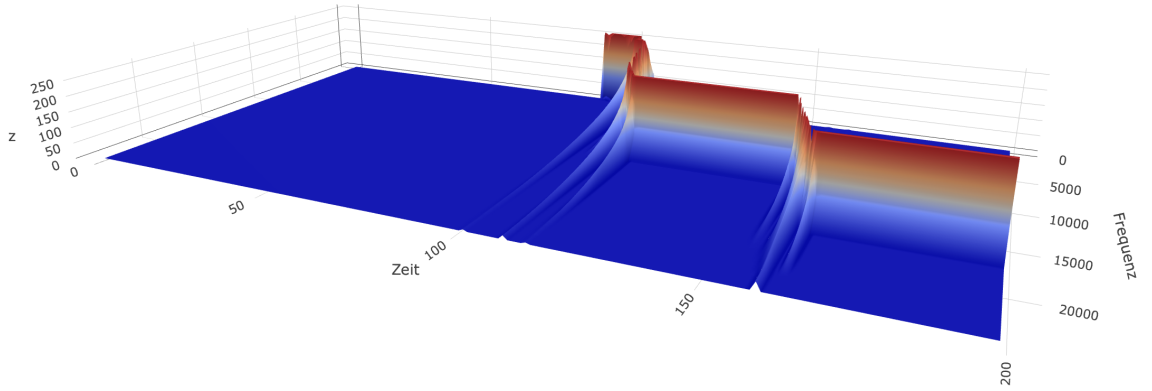

Abbildung 3.3: Ausgabe des Plots mit aktiviertem Sinus-Generator.

Die Abbildung 3.4 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 ... beschrieben, den Reset des Plots aus.

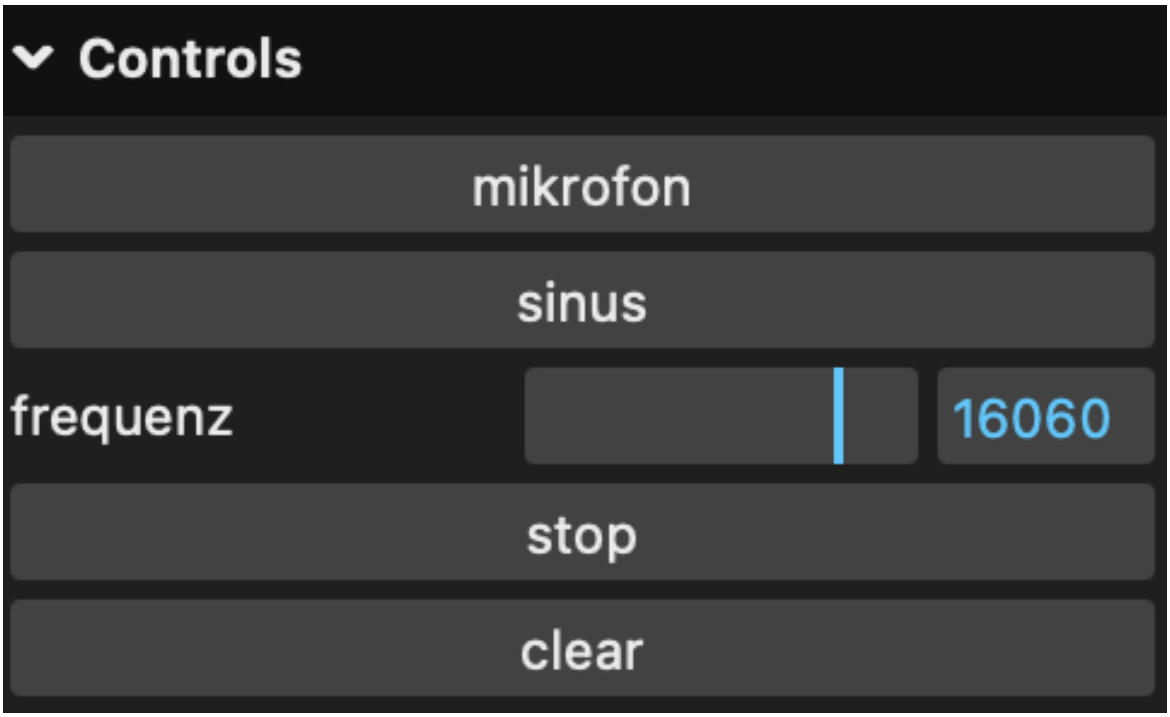

Abbildung 3.4: GUI der Ausgabe

4 Ausblick

In diesem Kapitel wird ein Ausblich darauf gegeben, wie das Endresultat noch verbessert werden könnte.

Literatur

[1] Ottmar Beucher. *Signale und Systeme: Theorie, Simulation, Anwendung: Eine beispielorientierte EinfÃŒhrung 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] 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).

[3] 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).

A Anhang

A.1 Statistik Programmiersprachen

Abbildung A.1: Statistik zu den meistgenutzten Programmiersprachen bei Entwicklern im Jahr 2024. [2]