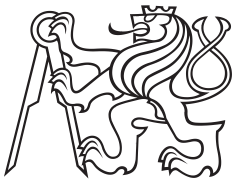
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Detekce dominantní frekvence

Lukáš Kotrbatý

Školitel: RNDr. Ondřej Žára
Obor: Softwarové inženýrství a technologie
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kotrbatý** Jméno: **Lukáš** Osobní číslo: **457184**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Detekce dominantní frekvence

Název bakalářské práce anglicky:

Pitch Detection

Pokyny pro vypracování:

Seznamte se se základy hudební teorie a terminologie: diatonické a chromatické stupnice, frekvence tónů a půltónů, centy. Dále nastudujte ta klientská JavaScriptová rozhraní, která dovolují pracovat se zvkovým signálem (getUserMedia, Web Audio a další).
Pomocí technologie Web Audio API naimplementujte nástroj, který bude v reálném čase provádět analýzu zvukového signálu z několika zdrojů (zvukový soubor, mikrofon). Pro úlohu nalezení dominantní frekvence použijte frekvenční analýzu (Fourierova transformace) a porovnejte kvalitu výsledku se statistickými metodami (autokorelace).
Navrhněte vhodný způsob vizualizace dominantní frekvence s ohledem na použití coby ladičky hudebního nástroje.
Vyzkoušejte výsledný nástroj na mobilních zařízeních, otestujte jejich výkonová omezení a zjistěte, jaká přesnost detekce je k dispozici při analýze v reálném čase.

Seznam doporučené literatury:

Grigsby, J.: Progressive Web Apps, ISBN: 978-1-937557-72-0, https://abookapart.com/products/progressive-web-apps
https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
https://aerotwist.com/blog/guitar-tuner/
Žára, O.: JavaScript, ISBN 8025145735,
https://www.knihydobrovsky.cz/javascript-programatorske-techniky-a-webove-technologie-659033

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ondřej Žára, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **05.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

RNDr. Ondřej Žára	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat RNDr. Onřeji Žárovi, zadavateli a vedoucímu bakalářské práce, za odborné vedení a rady při vypracování této práce. Dále bych chtěl poděkovat rodině za podporu během studia a svým bratrům za pocit, že stíhám a mám spoustu času.

Prohlášení

Čestně prohlašuji, že jsem bakalářskou práci na téma „Detekce dominantní frekvence“ vypracoval samostatně a s použitím uvedené literatury a pramenů.

V Praze, 24. května 2019

Abstrakt

Hlavním cílem této bakalářské práce je teoreticky navrhnout a naprogramovat mobilní webovou aplikaci, která bude v reálném čase provádět analýzu zvukového signálu. Tato analýza je následně využita jako ladička na kytaru. Aplikace je založena na řešení problému detekce dominantní frekvence pomocí metod autokorelace a rychlé Fourierovy transformace. Výsledný produkt využívá obou těchto metod k rozpoznání základní frekvence.

Klíčová slova: detekce dominantní frekvence, JavaScript, rychlá Fourierova transformace, autokorelace

Školitel: RNDr. Ondřej Žára
Ondrej.Zara@firma.seznam.cz

Abstract

The main goal of this bachelor thesis is to design and implement a mobile web application that will analyze the audio signal in real time. This analysis is then used as a guitar tuner. The application is based on solving the problem of pitch detection by two methods – autocorrelation and fast Fourier transform. The final version of this application uses both of these methods to recognize the fundamental frequency.

Keywords: pitch detection, JavaScript, fast Fourier transform, autocorrelation

Title translation: Pitch detection

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Pro naši dobu je typické naprosto každodenní používání internetu většinou populace. Není se tedy čemu divit, že vznikají další a další aplikace dostupné online. Prostředí webu se stalo velmi oblíbeným i mezi vývojáři nejen kvůli lepší optimalizaci, a tedy i lepšímu výkonu, ale také kvůli své přenositelnosti mezi jednotlivými platformami. Web nám nyní nabízí mnoho možností, které lze využít k vylepšení starých či k tvorbě úplně nových technologií.

Houfně se přesouváme z počítačů k mobilům. V roce 2019 přistupuje 63% uživatelů k internetu pomocí mobilních zařízení, v roce 2014 to bylo 48%. [1]

Se stoupajícím zájmem o mobilní internet stoupá poptávka po mobilních aplikacích ke každodennímu použití. Svět nativních aplikací je velmi rozmanitý a nesjednocený, oproti tomu webové aplikace jsou stabilní jistotou. Mají své výhody (platformová nezávislost, jednodušší tvorba), ale také nevýhody (malá rychlost, nízký výkon).

JavaScript je technologie sloužící k oživení webových stránek a právě díky této technologii vzniká nejvíce mobilních webových aplikací.

Kapitola 2

Cíl práce

Hlavním cílem zadání této bakalářské práce je teoreticky navrhnout a naprogramovat aplikaci, která bude v reálném čase provádět analýzu zvukového signálu z několika zdrojů (zvukový soubor, mikrofon). K tomuto úkolu mám vyzkoušet použitelné metody a zároveň otestovat jejich rychlost a přesnost, především na chytrých telefonech.

Test rychlosti potřebujeme kvůli druhému úkolu, kdy budu pomocí jedné z nastudovaných a otestovaných metod vytvářet ladičku pro kytaru v JavaScriptu. První doporučenou metodou je použití Fourierovy transformace a druhou použití autokorelace.

Během zkoumání jednotlivých metod jsem se měl také teoreticky seznámit se základní hudební teorií, základy JavaScriptu a jeho knihovnami. Nejpodrobněji pak s knihovnou Web Audio API a UserMedia.

Kapitola 3

Metodika a struktura

Ke splnění cílů uvedených v předchozí kapitole je nutné se nejprve seznámit se základy hudební teorie, která se nám bude hodit k pochopení metod využitelných k detekci dominantní frekvence. Zavedeme nutné pojmy, jako jsou například: stupnice, tóny, půltóny a centy. Tím se budeme zabývat v kapitole 4.

Poté stručně v kapitole 5 prozkoumáme použitou technologii, tedy JavaScript. Podíváme se na jeho historii a vývoj, popíšeme si jeho dnešní podobu a možnosti, které nám může poskytnout díky svým knihovnám (Web Audio API, UserMedia).

V kapitole 6 zanalyzujeme metody, které můžeme použít k detekci dominantní frekvence. Mezi ty základní patří zero-crossing, rychlá Fourierova transformace a autokorelace. Porovnáme jejich rychlost a přesnost. V závěru kapitoly vybereme jednu, kterou použijeme ke splnění druhého úkolu, tedy k vytvoření ladičky pro kytaru.

V sedmé kapitole projdeme analýzu a návrh řešení. Podíváme se na požadavky na aplikaci, možnou architekturu a stavový diagram.

Následující osmou kapitolu stručně věnujeme designu finální aplikace.

V předposlední kapitole se podíváme na konkrétní implementaci, ukážeme si slabiny a omezení aplikace. Stručně okomentujeme zdrojový kód a postupně se probereme vývojem aplikace.

V textu budou několikrát zmíněny výsledky měření, všechna měření proběhla na těchto zařízeních:

- desktop - Lenovo E550 s procesorem Intel(R) Core(TM) i5-5200U CPU @ 2,20 GHz 2,19 Ghz
- chytrý telefon - Huawei P20 lite s processorem Hisilicon Kirin 659 (8x Cortex-A53, 2,4 GHz, 16nm)

Kapitola 4
Hudební teorie

V této práci občas využívám některé teoretické hudební koncepty a termíny. Na tomto místě bych je chtěl stručně objasnit.

4.1 Hudba

Neexistuje žádná všeobecně přijímaná definice hudby, ale většina lidí se shodne na tom, že hudba je výstupem lidské činnosti. Jde o řazení zvuků za sebe uměleckým způsobem.

4.2 Zvuk

Zvuk chápeme jako mechanické vlnění v látkovém prostředí, které je schopno vyvolat sluchový vjem. Frekvence tohoto vlnění, které je člověk schopen vnímat, jsou značně individuální a leží v intervalu přibližně 16 Hz až 20000 Hz. Mechanické vlnění mimo tento frekvenční rozsah sluchový vjem nevyvolává, přesto se někdy také označuje jako zvuk. [2]

4.3 Tón

Tón v hudbě nebo akustice chápeme jako zvuk, který má periodický nebo alespoň přibližně periodický průběh kmitů. Může vznikat různými způsoby, ale vždy jde o určitý druh chvění (struny, hlasivek, vzduchového sloupce, membrány, ozvučné desky apod.). Jako grafické značky pro tóny používáme noty, které zapisujeme do notové osnovy. Hlavními atributy tónů v hudbě jsou výška, síla, délka a barva. Jako opak tónu můžeme chápat šum, který nemá periodický průběh kmitů a vzniká většinou mechanickými rozruchy. [3] [4]

4.3.1 Výška tónu

Výška tónu je určená frekvencí kmitů. Čím vyšší je frekvence, tím je vyšší počet kmitů, a tedy i vyšší tón. Nicméně výška tónu je pouze náš sluchový vjem

a záleží na posluchači, jakou pozici na hudební stupnici přiřadí jednotlivým tónům podle svého vnímání frekvence vibrací. Ačkoliv má výška tónu velmi těsnou souvislost s frekvencí tónu, nelze tyto dva pojmy libovolně zaměňovat. Frekvence je objektivní, vědecká vlastnost tónu, kterou lze změřit. Výška je pouze subjektivní vnímání zvukového vlnění konkrétní osobou, které nijak změřit nemůžeme. Toto tvrzení však neznamená, že se většina lidí neshodne, jestli je určitý tón vyšší nebo nižší. [3] [4]

4.3.2 Síla tónu

Síla tónu, v akustice také nazývaná hlasitostí tónu, je vlastnost daná amplitudou kmitu neboli velikostí rozkmitu při chvění. Názorně si to lze představit na chvějící se struně: čím více se střed struny vychyluje od osy, tím zní tón silněji. Síla tónu udává, jak hlasitě ho vnímáme. Sílu udáváme v různých jednotkách, nejčastěji v decibelech (zkratka dB). Nejsilnější lidským uchem vnímatelné zvuky mají kolem 130 decibelů. Při zpěvu nebo hře na hudební nástroje používáme italské hudební názvosloví k popisu hlasitosti tónů. Pianissimo odpovídá 40-50 decibelům a fortissimo 80-100 decibelům. [3] [4]

4.3.3 Délka tónu

Délkou tónu je myšlena časová hodnota jeho trvání, která je fyzikálně měřitelná pomocí běžných jednotek času (ms, s, min apod.). Mezi hudebníky se používá spíše notový zápis. V něm je délka tónu spojena s určitou hodnotou noty, která není vyjádřitelná absolutně, ale pouze relativně k dalším hodnotám. Ke kontrole dodržování délky tónů se používají metronomy, na kterých lze vidět, že notový zápis je relativní, co se týče doby trvání. Můžeme ho zahrát rychle při metronomu nastaveném např.: na 200 BPM (Beats Per Minute) nebo pomalu při 40 BPM. [3] [4]

4.3.4 Barva tónu

Barvou tónů myslíme vlastnost, která je charakterizuje především podle původu jejich vzniku. Umožňuje nám od sebe odlišit dva stejné tóny (shodují se ve výšce a hlasitosti), které vydávají různé zdroje zvuku a popisuje nám, jak tón vnímáme. Když slyšíme rozdíl mezi dvěma stejnými tóny, říkáme, že mají různou barvu. Jinak reagujeme na tóny mužských, ženských nebo dětských hlasů nebo tóny klaksonů automobilů, sirén či píšťal. Vlastní sluchový vjem tónu můžeme také subjektivně popsat. Tóny mohou znít jasně, temně, ostře, drsně, zastřeně, dutě, plně, sametově apod. [3] [5] [6]

4.4 Půltón

Půltón je nejmenším možným intervalem, velmi často používaným v současné evropské hudbě. Je definován jako interval mezi dvěma sousedícími

tóny na dvanáctitónové stupnici. V této stupnici lze taky pomocí něj definovat libovolný jiný interval. Například ze 4 půltónů dostaneme velkou tercii, ze sedmi čistou kvintu (viz tabulku 4.1). Považuje se za nejvíce disonantní, tedy nejméně příjemně znělý. [3]

4.5 Cent

Cent je bezrozměrná logaritmická jednotka používaná pro měření vzdáleností hudebních intervalů. Definice centu vychází z rovnoměrně temperovaného ladění, které dělí oktávu na dvanáct stejně velkých půltónů, mezi kterými je vzdálenost vždy 100 centů. Jeden cent je tedy 1/100 půltónu nebo 1/1200 oktávy.

Nejčastěji se centy používají k vyjádření malých intervalů nebo ke srovnání velikostí intervalů v různých ladicích systémech. Při použití čistých či přirozených ladění můžeme frekvence tónů v určitém intervalu vyjádřit poměrem celých čísel. Toto číslo dostaneme zkrácením poměru jejich frekvencí. Kupříkladu oktávu lze vyjádřit poměrem 2:1, čistá kvinta je 3:2 atd. Pokud poměr frekvencí nelze vyjádřit jako jednoduchý zlomek, tak je vhodné použít desetinné číslo nebo právě vyjádření v centech.

V Evropě se od 19. století používá převážně rovnoměrně temperované ladění, ve kterém jde jako jednoduchý zlomek vyjádřit pouze interval čisté oktávy. Pro ostatní intervaly se běžně používají právě centy. Protože jde o rovnoměrné ladění, není žádným překvapením, že hodnoty vzdálenosti v centech jsou násobky 100. Např. velká tercie má velikost 1,259921, což je 400 centů. Pro lidské ucho je jeden cent naprosto zanedbatelný rozdíl mezi dvěma tóny, který nemá šanci postřehnout. [7]

4.6 Stupnice

Stupnice, v kontextu hudby, je řada po sobě jdoucích tónů, upravená podle určitých pravidel. Může mít různou délku. I soubor tónů jdoucí přes 3 oktávy je stupnice. Nicméně zde se omezíme na chápání stupnice jako sled tónů v rámci jedné oktávy.

Pravidla určují vzdálenosti mezi jednotlivými stupni (intervaly). Stupnice rozlišujeme především podle počtu tónů a vzdáleností mezi nimi (např. umístěním půltónů a celých tónů). Záleží také na použitém ladění.

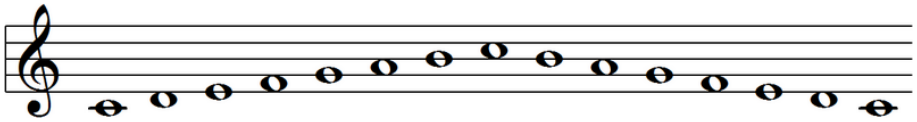
Každá stupnice má jeden základní tón, který nazýváme tónika. Tímto tónem stupnice začíná a jak se občas uvádí, tak i končí. Toto tvrzení není úplně správné. Přestože se lidé občas mylně domnívají, že stupnice má osm nebo třináct stupňů, má jich jen sedm nebo dvanáct. Poslední tón je již znovu opakovaný první tón o oktávu výše. Od tóniky se pak odvozuje název dané stupnice. Ke správnému označení používáme písmena. Stupnici začínající tónem C označíme za C-dur/C-moll apod. Dříve používané označování pomocí solmizačních slabik (do-re-mi-fa-sol-la-si) se dnes již moc nepoužívá.

Stupnice můžeme rozdělit podle počtu tónů na pětitónové, tedy pentatonické (ty jsou typické pro hudbu z oblasti Číny a Japonska), šestitónové (celotónové), sedmitónové (diatonické, církevní, cikánské) a dvanáctitónové (chromatická, alterovaná). V průběhu časů většina z nich vymizela a dnes se využívají už především jen sedmitónové a dvanáctitónové. Nás budou zajímat stupnice diatonické a chromatické. [3] [8] [9]

4.6.1 Diatonické stupnice

Diatonická stupnice je sedmitónová stupnice, která obsahuje pět celých tónů a dva půltóny v každé oktávě, ve které jsou oba půltóny od sebe odděleny buď dvěma nebo třemi celými tóny, v závislosti na jejich poloze. Tyto stupnice můžeme vytvořit od libovolného tónu. Konkrétní stupnice je určena předznamenáním (křížky, béčka). Může se stát, že dvě stupnice budou znít stejně, přestože budou mít jiné označení. Například Fis-dur a Ges-dur.

V dnešní době se význam diatonické stupnice zúžil pouze na dva konkrétní typy stupnic: běžně používané durové stupnice a mollové stupnice. První z nich je také někdy označována jako diatonická dur (durová diatonika), většinou ji vnímáme jako tu veselejší nebo neutrální a značíme ji velkým písmenem a příponou („-dur“). Tu druhou označujeme jako diatonickou moll (mollová diatonika), tu vnímáme spíše jako smutnou a značíme ji malým písmenem a příponou („-moll“). [8]

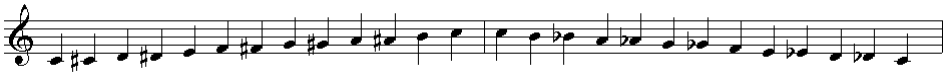


[10]

Obrázek 4.1: Ukázka diatonické stupnice C-dur

4.6.2 Chromatické stupnice

Chromatická stupnice je dvanáctitónová a skládá se pouze z půltónů. Protože pokrývá všechny možné půltóny, existuje jen jedna. Při použití dvanáctistupňového rovnoměrného temperovaného ladění nás zaujmou přesné rozestupy mezi jednotlivými stupni. Každý půltón má stejnou velikost, a to přesně 100 centů. Toto ladění je v současné době v Evropě nejčastěji používané. Ostatní stupnice, v dnešní době běžně používané v evropské hudbě (diatonické – dur a moll), lze vytvořit jako podmnožinu této stupnice. [8]



[11]

Obrázek 4.2: Ukázka chromatické stupnice C-dur

4.7 Hudební intervaly

Interval v hudebním kontextu je vzdálenost mezi dvěma tóny. Nejmenším intervalem je půltón (např. C – C♯). Pro dva půltóny za sebou se používá označení celý tón, aby se nepletlo s tónem jako zvukem, který má periodický průběh. Další intervaly již nemají české názvy, a proto pro jejich označení používáme pro češtinu upravené latinské řadové číslovky v ženském rodě (prima = první, secunda = druhá, tertia = třetí atd.).

Intervaly dělíme na melodické a harmonické. Melodické se skládají ze dvou tónů, které zazní po sobě, zatímco harmonické zazní ve stejnou dobu.

V tabulce 4.1 jsou uvedeny všechny intervaly v rámci jedné oktávy. Existují i intervaly větší než oktáva, ale protože jde vlastně jen o oktávu posunuté jiné intervaly a hodnoty jsou tedy také jen posunuté, nebudeme se jimi dále zabývat. Poslední sloupec nám ukazuje vzdálenost mezi jednotlivými intervaly v bezrozměrné jednotce cent.

Stupeň	Půltóny	Název intervalu	Základní tón: C	Centy
1	0	čistá prima	C – C	0
2	1	malá sekunda	C – C♯	100
2	2	velká sekunda	C – D	200
3	3	malá tercie	C – D♯	300
3	4	velká tercie	C – E	400
4	5	čistá kvarta	C – F	500
4	6	velká kvarta	C – F♯	600
5	7	čistá kvinta	C – G	700
6	8	malá sexta	C – G♯	800
6	9	velká sexta	C – A	900
7	10	malá septima	C – A♯	1000
7	11	velká septima	C – H	1100
8	12	čistá oktáva	C – C	1200

Tabulka 4.1: Přehled hudebních intervalů

Akustika také definuje pojem konsonance ve významu souznění nebo souzvuku. Nejsilnější konsonanci mají čistá prima a čistá oktáva. Nejpříjemněji lidskému uchu znějí dva stejné tóny, pak následuje čistá kvinta, čistá kvarta, velká sexta a velká tercie. Opak konsonance je disonance, ta tedy znamená souzvuk dvou nebo několika tónů, který působí drsně až nepříjemně. [3] [9]

4.8 Frekvence ve spojitosti s tóny

Za každým tónem se skrývá nějaká frekvence. Je zajímavé pozorovat, jak spolu souvisí tóny, mezi kterými je určitý interval. Pro hudebníky je použití konkrétních frekvencí zbytečně složité, a proto si většina pamatuje jen komorní A jako 440 Hz. Ostatním tónům přiřazujeme jejich písmenné pojmenování.

Vynásobení frekvence tónu číslem $\sqrt[12]{2}$ získáme frekvenci následujícího tónu,

$$f_i = f_j \cdot \sqrt[i]{2}$$

kde f_i je frekvence tónu právě i půltónů nad referenční frekvencí f_j. Sled tónů vytvořených tímto vzorcem nám dá to, co nazýváme chromatickou stupnicí. Přístroje, které jsou naladěné tímto způsobem jsou nejčastěji používané v západní hudbě. Výše uvedené ladění nazýváme dvanáctistupňové rovnoměrné temperované ladění.

Konkrétní frekvence tónů úzce souvisí s použitým laděním. Zde se budeme zabývat jen již zmíněným dvanáctistupňovým rovnoměrným temperovaným laděním, protože je nejpoužívanější. Toto ladění se vždy odvíjí od komorního A, tedy od tónu A4.

Jednotlivé posuny tónů označujeme čísly, které zapisujeme jako dolní indexy. Nejhlubší tóny, tedy ty, které mají nejnižší frekvenci, mají index nula, ty nejvyšší mají až index osm. Vyšší označení se nepoužívá, protože vyšších tónů většina hudebních nástrojů nedosáhne.

Ve dvanáctistupňovém rovnoměrném temperovaném ladění si můžeme všimnout, že poměr tónu a jeho o čistou oktávu zvýšeného posunu je přibližně 1:2. Například poměr A4 a A5 je 440:880, tedy 1:2. Další poměry můžete vyčíst z tabulky 4.2. Frekvence je uvedena v jednotce Hertz. „Klavír-1“ označuje nejhlubší tón klasického klavíru (například Yamaha GB1 K PE) a „Klavír-2“ pak označuje nejvyšší tón. Stejně také pro westernovou kytaru s 20 pražci (například LAG Tramontane T70DC) [12] [13]

4.9 Krátká historie sjednocování ladění ve světě

V hudební historii se používalo mnoho ladění. Hlavním důvodem bylo především to, že hudba se vyvíjela odděleně v různých částech světa. Většinou se ladění v dané oblasti odvíjelo od prvního nástroje, který byl používán. Valná většina Evropy používala ještě v 19. století různá ladění. Pro Francii byla typická frekvence 432 Hz pro A4. V Itálii používali různé druhy ladění od 420 Hz po 460 Hz, nejčastěji však 435 Hz pro A4.

Období nejednotného ladění trvalo poměrně dlouho, změna přišla až v roce 1885, kdy hudební komise italské vlády rozhodla o sjednocení ladění všech nástrojů na území Itálie. Od té doby se všechny nástroje ladily na 440 Hz pro komorní A. V roce 1917 se ke změně rozhodla i americká federace hudebníků a nejdůležitější zlom přišel v roce 1953, kdy byla podepsána celosvětová dohoda. Signatáři prohlásili, že střední (komorní) A na klavíru bude navždy laděno přesně na 440 Hz. Tato frekvence se stala standardem ISO-16 pro ladění všech hudebních nástrojů založených na chromatické stupnici, která se nejčastěji používá pro hudbu v dnešní Evropě. Všechny ostatní tóny jsou naladěny podle standardních matematických poměrů k a od tohoto komorního A. Tato standardizace nám zajistila, že například klavír v Dublinu bude znít stejně jako v Pekingu.

Nabízí se otázky: „Je to tak správně?“, „Mělo by se umění standardizovat?“. Ani na jednu z těchto otázek neexistuje správná odpověď. Sjednocení

ladění po celém světě nám zaručilo, že si hudebníci mohou jednoduše zahrát spolu, bez nutnosti přeladění nástrojů. Na druhou stranu hudba ztratila rozmanitost. Při používání různých ladění bychom měli možnost slyšet více melodií.

Používání pouze jednoho ladění způsobilo, že si na něj lidské ucho zvyklo a přijde nám nejpřirozenější, slyšíme ho přece všude. Všechny poslední hity byly nahrány v tomto ladění. Takže při použití jiného ladění většina lidí zbystří a řekne si, že takhle ta píseň přece nemá znít.

Lze předpokládat, že postupem času dojde k návratu rozmanitostí v ladění. Důvodem může být skutečnost, že se tak zvýší množství možných nových melodií, které mohou vzniknout. V dnešní době již začínají posluchači upozorňovat na podobnost nových skladeb s dříve vzniklými. [13]

Nástroj	Tón	Frekvence	Nástroj	Tón	Frekvence
Klavír-1	C_1	32.7		$F^\sharp_4/G^\flat_4$	369.99
	$C^\sharp_1/D^\flat_1$	34.65		G_4	392
	D_1	36.71		$G^\sharp_4/A^\flat_4$	415.3
	$D^\sharp_1/E^\flat_1$	38.89		A_4	440
	E_1	41.2		$A^\sharp_4/H^\flat_4$	466.16
	F_1	43.65		H_4	493.88
	$F^\sharp_1/G^\flat_1$	46.25		C_5	523.25
	G_1	49		$C^\sharp_5/D^\flat_5$	554.37
	$G^\sharp_1/A^\flat_1$	51.91		D_5	587.33
	A_1	55		$D^\sharp_5/E^\flat_5$	622.25
	$A^\sharp_1/H^\flat_1$	58.27		E_5	659.25
	H_1	61.74		F_5	698.46
	C_2	65.41		$F^\sharp_5/G^\flat_5$	739.99
	$C^\sharp_2/D^\flat_2$	69.3		G_5	783.99
	D_2	73.42		$G^\sharp_5/A^\flat_5$	830.61
	$D^\sharp_2/E^\flat_2$	77.78		A_5	880
Kytara-1	E_2	82.41		$A^\sharp_5/H^\flat_5$	932.33
	F_2	87.31		H_5	987.77
	$F^\sharp_2/G^\flat_2$	92.5	Kytara-2	C_6	1046.5
	G_2	98		$C^\sharp_6/D^\flat_6$	1108.73
	$G^\sharp_2/A^\flat_2$	103.83		D_6	1174.66
	A_2	110		$D^\sharp_6/E^\flat_6$	1244.51
	$A^\sharp_2/H^\flat_2$	116.54		E_6	1318.51
	H_2	123.47		F_6	1396.91
	C_3	130.81		$F^\sharp_6/G^\flat_6$	1479.98
	$C^\sharp_3/D^\flat_3$	138.59		G_6	1567.98
	D_3	146.83		$G^\sharp_6/A^\flat_6$	1661.22
	$D^\sharp_3/E^\flat_3$	155.56		A_6	1760
	E_3	164.81		$A^\sharp_6/H^\flat_6$	1864.66
	F_3	174.61		H_6	1975.53
	$F^\sharp_3/G^\flat_3$	185		C_7	2093
	G_3	196		$C^\sharp_7/D^\flat_7$	2217.46
	$G^\sharp_3/A^\flat_3$	207.65		D_7	2349.32
	A_3	220		$D^\sharp_7/E^\flat_7$	2489.02
	$A^\sharp_3/H^\flat_3$	233.08		E_7	2637.02
	H_3	246.94		F_7	2793.83
	C_4	261.63		$F^\sharp_7/G^\flat_7$	2959.96
	$C^\sharp_4/D^\flat_4$	277.18		G_7	3135.96
	D_4	293.66		$G^\sharp_7/A^\flat_7$	3322.44
	$D^\sharp_4/E^\flat_4$	311.13		A_7	3520
	E_4	329.63		$A^\sharp_7/H^\flat_7$	3729.31
	F_4	349.23	Klavír-2	H_7	3951.07

Tabulka 4.2: Přehled tónů a jejich frekvencí

Kapitola 5
JavaScript

JavaScript (JS) je multiplatformní, objektově orientovaný skriptovací jazyk, který se většinou používá k dynamickému oživení webových stránek na straně klienta. Také je možné ho využít na straně serveru pomocí speciálních prostředí, jako je například Node.js. Někteří lidé ho chybně zaměňují s programovacím jazykem Java. Oba jazyky „Java“ i „JavaScript“ jsou registrované ochranné známky společnosti Oracle v USA i dalších zemích. Tyto programovací jazyky se od sebe výrazně liší jak ve své syntaxi a sémantice, tak ve svém přístupu k základním problémům programovacích jazyků. [14]

5.1 Historie

Tvůrce Brendan Eich původně zamýšlel vytvořit z jazyku Mocha, což je původní název pro JavaScript, programovací jazyk na straně serveru, avšak brzy po změně zaměstnavatele tuto myšlenku přehodnotil. Jeho nový zaměstnavatel Netscape Corporation navázal na jeho nápad a JavaScript se již v září 1995 objevuje v tehdy nejznámějším prohlížeči Netscape Navigator 2.0. Tento prohlížeč v té době používalo 80% všech uživatelů internetu.

Okamžitě zaznamenal úspěch, především proto, že do té doby se webové stránky skládaly pouze z HTML a CSS. JavaScript je oživil a rozhýbal, na tento úspěch musela nutně zareagovat konkurence. Microsoft, který se svým prohlížečem Internet Explorer 3.0 chtěl převzít vládu nad online světem, zavedl JavaScriptovou podporu pod jménem JScript již v srpnu 1996.

Původní jméno Mocha bylo brzy změněno na LiveScript a poté na JavaScript. Šlo o licenční dohodu mezi společností Sun (nyní Oracle) a Netscape. Sun vytvořil první podobu Javy také v roce 1995 a ta se rychle stala velmi oblíbenou. Podobnost v názvu JavaScriptu s programovacím jazykem Java tedy není náhodná, šlo o marketingový tah, který vyšel. Jak řekl Brendan Eich v rozhovoru pro InfoWorld: „Ke všem změnám došlo během šesti měsíců od května do prosince 1995, nejprve to byla Mocha, pak LiveScript a na začátku prosince Netscape a Sun uzavřeli licenční smlouvu a vznikl JavaScript. Hlavním smyslem vzniku bylo vytvořit doplňkový skriptovací jazyk k Javě.“ [15]

V listopadu 1996 společnost Netscape začala spolupracovat s ECMA International, aby se stal JavaScript průmyslovým standardem. Od té doby je

standardizovaný JavaScript nazýván ECMAScript a jeho specifikaci lze nalézt pod ECMA-262.

Tento jazyk si prošel několikrát změnou názvu, pro lepší orientaci ho ve všech stádiích jeho historie budu nazývat jeho běžně používaným jménem – JavaScript. [16] [14] [17]

5.2 Dnešní verze

V dnešní době je JavaScript již běžnou součástí webových stránek. V červnu 2017 byla zveřejněna jeho osmá verze. Skript se vkládá přímo do HTML dokumentu pomocí tagu <script>. Samotný kód pak může být vepsaný mezi tento párový tag, popřípadě do samotného souboru, na který tag odkazuje. Skript slouží hlavně k ovládání prvků grafického rozhraní (GUI), jako jsou například tlačítka, formulářová pole apod.

Kromě oživení stránek je dnes často používán při tvorbě Single Page Aplikací (SPA). To jsou webové aplikace nebo webové stránky zobrazující pouze jednu webovou stránku, která dynamicky mění svůj obsah pomocí JavaScriptu, s cílem poskytnout podobný uživatelský prožitek jako desktopová aplikace. K tvorbě Single Page Aplikací se hojně využívají frameworky a knihovny, mezi ty nejoblíbenější patří React, Vue nebo Angular. [18] [19]

Ačkoliv jde o běžně využívaný programovací jazyk, stále má nálepku hloupého jazyka mezi ostatními. Vychází to pravděpodobně z původní verze. Šlo o dynamicky typovaný, událostmi řízený jazyk, který neznal třídy, ale používal prototypy k zachování přístupů objektově orientovaného programování (OOP).

Svou oblíbenost JavaScript nabyl také díky snadnému použití webových aplikačních programovacích rozhraní (API). Jsou to konstrukce dostupné ve většině programovacích jazyků, které umožňují vývojářům použit další, často složitější struktury a funkce. Rozhraní od vývojáře odstiňuje složitější kód a dává mu možnost přistoupit k tomuto kódu za pomocí lehce pochopitelné syntaxe.

Pro lepší názornost bych uvedl příklad - elektřina v domě, kde rozhraním jsou zásuvky, do kterých jednoduše připojíme konkrétní zařízení. Pokud bychom toto rozhraní neměli, museli bychom se pokusit zapojit zařízení přímo do sítě, což může být pro neznalého člověka nebezpečné. Analogicky je to s API v programovacím světě. Pro vykreslení 3D grafiky v C/C++ je jednodušší použít API nějakého z vyšších programovacích jazyků, jako je například JavaScript nebo Python, než se trápit s nižším jazykem, který byl určený na úplně jiné úlohy. [18] [20]

Webová API dělíme na dvě skupiny, a to prohlížečové API a API třetích stran.

5.2.1 Prohlížečové API

Prohlížečové API jsou vestavěné funkce dostupné přímo z prohlížeče. Dávají nám přístup k datům, která souvisejí s daným prostředím, na němž prohlížeč

běží. Například Geolocation API nám dává přístup k datům o umístění zařízení, Web Video API nám dává přístup k úpravě videa, které můžeme přehrát přímo v prohlížeči atd. [20] [21]

5.2.2 API třetích stran

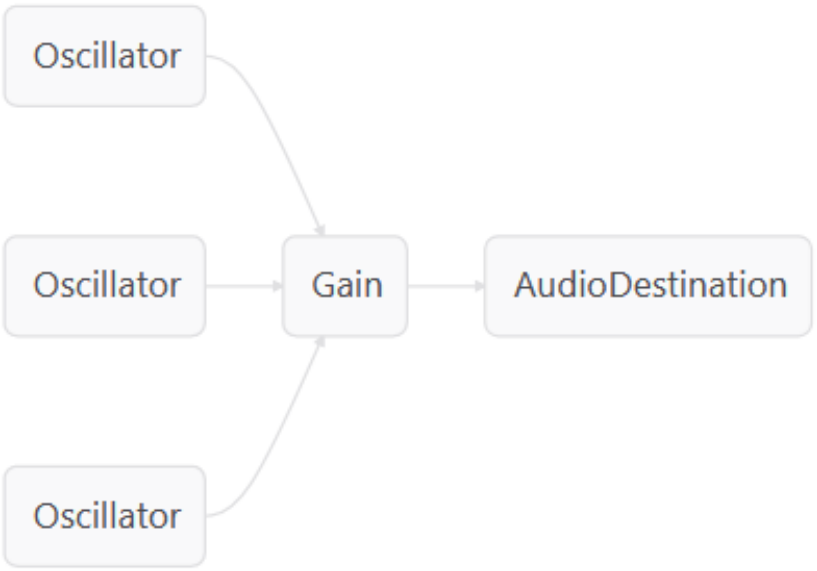
API třetích stran nejsou přímo vestavěná v prohlížeči, ale je nutné zkopírovat jejich zdrojový kód do webové stránky nebo na něj aspoň odkázat, pokud je dostupný online. Například Mapy.cz API od Seznamu nám dává možnost vložit si část mapy do své vlastní stránky. [20] [21]

Pro svou práci potřebuji využít Web Audio API.

5.3 Web Audio API

Web Audio API je poměrně výkonný a všestranný systém, který slouží k ovládání zvuku na webu. Umožňuje vývojářům využívat zvukové zdroje, přidávat efekty do zvuku, vytvářet zvukové vizualizace, aplikovat prostorové efekty (angl. panning) a mnoho dalšího.

Funguje na základě propojování výstupů a vstupů jednotlivých uzlů Web Audia. Tyto uzly nám poskytuje `audioContext`. Na začátku většinou máme nějaký zdroj. To může být buď oscilátor (`OscillatorNode`), který nám vytvoří zvuk o určité frekvenci, nebo stream (`AudioBufferSourceNode` nebo `MediaElementAudioSourceNode`), který načte data z nahraného zvukového souboru. Tento zdroj pak můžeme napojit například na zesilovač (`GainNode`) a ten následně do výstupu (`destinationNode`). Výsledné zapojení si můžeme prohlédnout například ve Firefoxu.



Obrázek 5.1: Ukázka vizualizace struktury Web Audia API

Každý z uzlů nám dává přístup k mnoha atributům a funkcím, kterými můžeme výsledné audio měnit. Oscilátoru můžeme nastavit frekvenci, rozladění a typ vlny, zesilovači hodnotu zesílení a analyzéru (`AnalyserNode`) například počet frekvenčních pásem. [20] [21]

5.3.1 AnalyserNode

Rozhraní `AnalyserNodu` nám poskytuje jak frekvenční, tak časová data o zvuku v reálném čase. Tento uzel zvuk nijak nezmění, co do něj přijde na vstupu, to z něj zase vyjde. Umožňuje nám vzít zvuková data, zpracovat je a pomocí nich vygenerovat například zvukovou vizualizaci.

`AnalyserNode` má právě jeden vstup a jeden výstup, ale na rozdíl třeba od zesilovače bude fungovat i bez připojeného výstupu. Mezi jeho atributy patří:

- `fftSize` – velikost pole s frekvenčními daty
- `minDecibels` - minimální hodnota pro výstup při generování frekvenčních dat
- `maxDecibels` – maximální hodnota pro výstup při generování frekvenčních dat
- `smoothingTimeConstant` – konstanta na zprůměrování předchozího vzorku dat s novým
- `frequencyBinCount` – počet frekvenčních pásů, polovina `fftSize`

[20]

Kapitola 6
Metody

Určení dominantní frekvence je poměrně složitý úkol, kterým se zabývalo hodně subjektů. Byly nalezeny metody a postupy, pomocí kterých lze tento úkol splnit. Metody jsou různě složité a různě vhodné pro určité platformy. V této kapitole se budu věnovat spíš těm jednodušším metodám, které pravděpodobně budou použitelné pro můj výsledný produkt - ladičku.

Kritéria hodnocení jednotlivých metod jsou především rychlost a přesnost. Mým cílem je tedy analyzovat zvuk a najít dominantní frekvenci v reálném čase. K měření jsem používal vlastní skript, který měřil počet milisekund mezi dvěma událostmi. První událostí byla změna frekvence, tedy vygenerování nového vzorku dat a druhou určení dominantní frekvence.

V této kapitole používám slovní spojení „dominantní frekvence“ a „základní frekvence“. Tato slovní spojení mají stejný význam a v angličtině se běžně zaměňují (dominant frequency a fundamental frequency).

Metody, které jsem chtěl otestovat jsou:

- Zero-crossing
- Rychlá Fourierova transformace
- Autokorelace

6.1 Zero-crossing

Zero-crossing, jak už název napovídá, je metoda, která je založena na křížení signálů s referenční nulou (osou x). Jde zřejmě o nejjednodušší metodu, kterou k tomuto úkolu můžeme použít. Jak uvádí Miguel Amado a Juarez Hoppe Filho [22], tato metoda je velmi jednoduchá a nenákladná, ale není příliš přesná. Pokud není signál čistý a obsahuje šumy či mnoho harmonií, získáme touto metodou velmi špatné výsledky. Důvodem je vysoce pravděpodobný výskyt jiných částí signálů, které jsou silnější než základní frekvence. Problém s detekcí dominantní frekvence pomocí této metody také může nastat, pokud signál hodně osciluje kolem referenční nuly.

Po přečtení hodnocení této metody a základním otestování jsem tuto metodu vyloučil z dalšího zkoumání a dále ji netestoval. K vytvoření ladičky potřebuji dost přesné určení frekvence, což tato metoda neumožňuje.

6.2 Rychlá Fourierova transformace

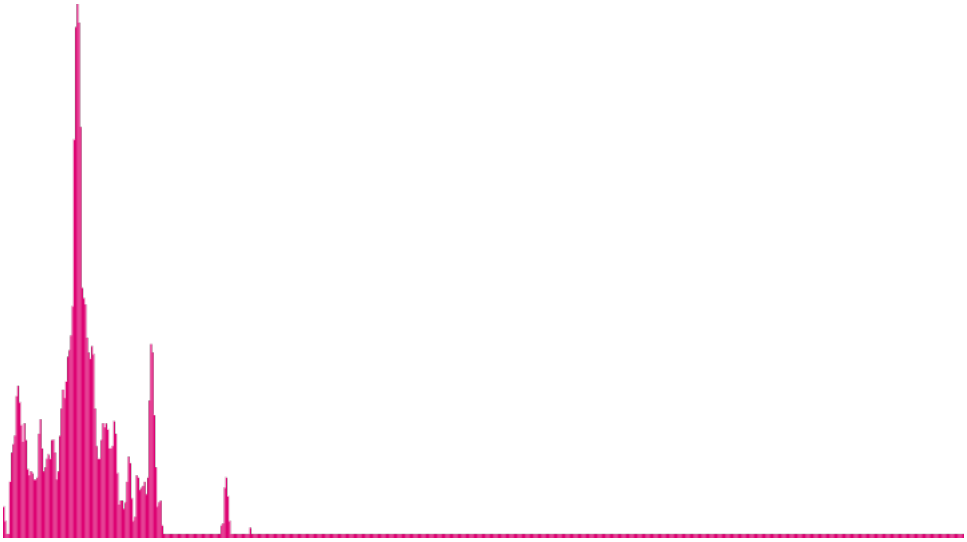
Rychlá Fourierova transformace (FFT) je algoritmus, který počítá diskrétní Fourierovu transformaci (DFT) nebo její inverzi (IDFT). Fourierova analýza převádí signály z původního časového spektra na frekvenční spektrum a naopak.

Podrobnější informace ohledně fungování tohoto algoritmu jsou uvedeny v [23].

V této práci budu používat termín vzorkovací frekvence (`SR`), který udává, kolik vzorků vznikne za jednu vteřinu. Používá se několik hodnot. Pro webové prohlížeče jsou typické hodnoty dvě, a to 48000 Hz a 44100 Hz. Můj prohlížeč používá defaultně první hodnotu, tedy 48000 Hz. V případě, že budu odkazovat na vzorkovací frekvenci, je tím myšlena právě zmíněná hodnota.

6.2.1 Použití rychlé Fourierovy transformace

Aplikace rychlé Fourierovy transformace na zvukový signál nám tento signál zkonvertuje do frekvenčního spektra. Jak již bylo uvedeno, jakýkoliv zvuk, který není šumem je jen složenina několika sinusoid. FFT nám říká, jak moc je která frekvence v daném zvuku zastoupena. Výstup rychlé Fourierovy transformace je znázorněn na obrázku 6.1.



[24]

Obrázek 6.1: Ukázka vizualizace zvuku pomocí FFT

Frekvenční spektrum si lze představit jako pole všech možných frekvencí, kde každý index pole odpovídá určitému frekvenčnímu pásu. Frekvenční rozpětí odpovídá intervalu 0 až $SR/2$. Pokud má tedy náš AudioContext vzorkovací frekvenci předdefinovanou na 48000 Hz, pole bude začínat na 0 Hz a končit na 24000 Hz.

Tento frekvenční pás má určitý rozsah, který závisí na zvolené hodnotě

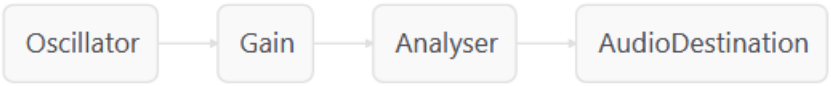
`fftSize` z `AnalyserNodu`. Např.: pro `fftSize` nastavenou na 512 každý pás odpovídá hodnotě přibližně 93 Hz, což je skutečně nepoužitelná hodnota. Pro své další zkoumání použiji `fftSize` nastavenou na její nejvyšší hodnotu, kterou je 32768.

Toto nastavení vytvoří pásy o velikosti přibližně $24000/16384 = 1,5$ Hz, to už je použitelnější hodnota.

Jednotlivé pásy obsahují informaci o tom, jak moc je daná frekvence v analyzovaném zvuku zastoupena. V poli těchto pásů hledám ten, který bude obsahovat dominantní frekvenci, tedy ten s nejvyšší hodnotou.

6.2.2 Postup

Napsal jsem si jednoduchou webovou stránku, na které jsem chtěl vyzkoušet, co mi dokáže analyzér říct o frekvenčním spektru zvuku. Po nastavení všech uzlů jsem dostal tuto síť.



Obrázek 6.2: Setup Web Audia API pro testovací prostředí

Na stránce se daly nastavit všechny atributy `AnalyserNodu`, kromě `minDecibels` a `maxDecibels`, protože pro tento typ úlohy nedává smysl data nikterak omezovat.

V `AnalyserNodu` existují dvě metody, které dokáží generovat data z frekvenčního spektra. Jsou to `getByteFrequencyData` a `getFloatFrequencyData`. Obě aplikují na zvuk rychlou Fourierovu transformaci, která nám poskytne rozklad vstupního zvuku na frekvenční pásy. Metody se od sebe liší jen v podobě výstupu. První do pole vloží bytové hodnoty a druhá čísla s plovoucí desetinnou čárkou.

V těchto datech jsem chtěl hledat nejvyšší hodnotu, ze které bych mohl určit dominantní frekvenci. Po nalezení indexu nejvyšší hodnoty v poli mohu získat frekvenci. Předpis vypadá následovně:

$$f = \frac{i \cdot SR}{2 \cdot FBC} = \frac{i \cdot 48000}{fftSize}$$

kde f je hledaná frekvence, SR je vzorkovací frekvence a FBC je `frequencyBinCount`, vlastnost `AnalyserNodu`. FBC je počet hodnot v poli vygenerovaném FFT. Jeho dvojnásobek nám dá `fftSize`.

Použitá metoda k určení frekvence vypadá následovně. Hodnota 365 Hz vznikla z frekvence struny E_4 - 331 Hz. K té přidávám pár hertzů navíc, kdyby byla kytara příliš rozladěná.

```
/*
class method for finding frequency
argument: data - new data from analyserNode (getFloatFrequencyData)
*/
findFrequency(data) {
    /* Max value */
    let max = data[0];
    /* Size of one bin depends on browser sample rate and fftSize */
    let binSize = this.context.sampleRate / (2 *
        this.analyserNode.frequencyBinCount);
    /* 365 Hz = 330 Hz + space for mistakes. 331 Hz is frequency of
        E4 */
    let maxIndex = Math.floor(365/binSize);
    /* Index of max value */
    let indexOfMax = 0;
    /* There is no reason to go through the whole array, bin number
        maxIndex corresponds to frequency 365 Hz, enough space for
        mistakes */
    for(let i=1;i<maxIndex;i++){
        if(data[i] > max){
            max = data[i];
            indexOfMax = i;
        }
    }

    /* finding frequency by multiplying index by the bin size*/
    let frequency = indexOfMax * binSize;
    return frequency;
}
```

Listing 6.1: Metoda k nalezení dominantní frekvence pomocí FFT

Jako první jsem si vybral metodu `getByteFrequencyData`. Vygenerované data jsem vykreslil do kreslícího plátna JavaScriptu (canvas) a získal vizualizovaný výstup Fourierovy transformace. Použitím metody `requestAnimationFrame` jsem dostal animaci této vizualizace. [20]

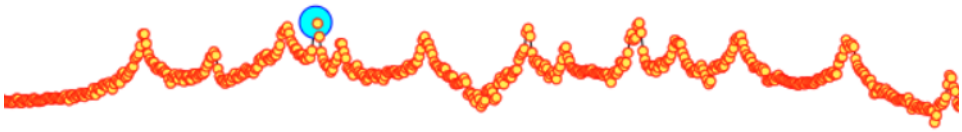
Vizualizovaný výstup této metody je znázorněn na obrázku 6.3. Žluté body značí hodnoty jednotlivých frekvenčních pásů, modrý pak nejvyšší nalezenou hodnotu. Data zobrazená na obrázku pocházejí z oscilátoru. Na obrázku jsou patrny harmonie.

Harmonie (oficiální český ekvivalent neexistuje, vyházím tedy z anglického pojmu harmonics) jsou členy harmonické řady. Harmonie zvukové vlny jsou vlny o frekvenci, která je kladným celočíselným násobkem originální frekvence. Původní zvukovou vlnu nazýváme první harmonií, dalším násobkům pak říkáme vyšší harmonie.

Všechny vyšší harmonie mají periodu v původní frekvenci. Toto platí i pro součet těchto harmonií. Např.: pokud je základní frekvence 50 Hz, první 3 vyšší harmonie budou 100 Hz (druhá harmonie), 150 Hz (třetí harmonie) a 200 Hz (čtvrtá harmonie). Jakýkoliv součet těchto zvukových vln má periodu v původní frekvenci 50 Hz. [25]

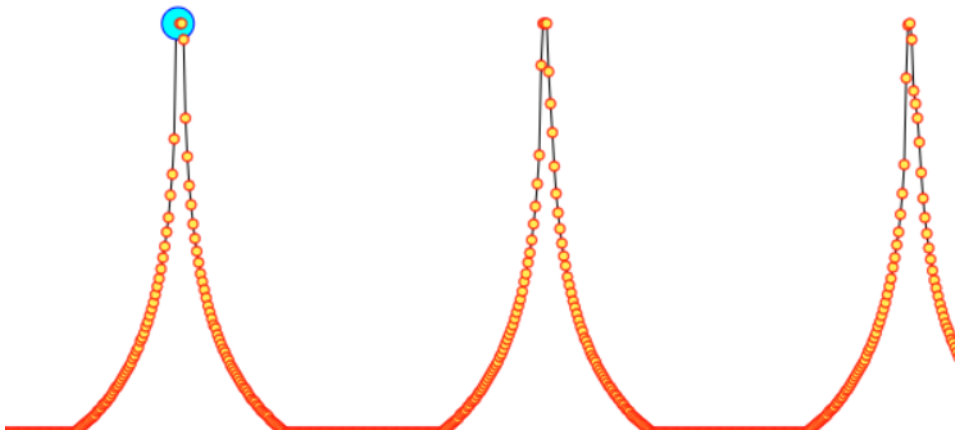
6.2.3 Problémy

První problém byl již nastíněn, jsou jimi zmíněné harmonie. Ty se hodnotou blíží té nejvyšší a mohlo by tak docházet ke špatnému určení nejvyšší hodnoty. Doposud byl řešen pouze výstup z oscilátoru, data z kytary budou logicky obsahovat harmonií ještě mnohem více (viz obrázek 6.3). Proto s daty z kytary nebude prakticky možné pracovat.



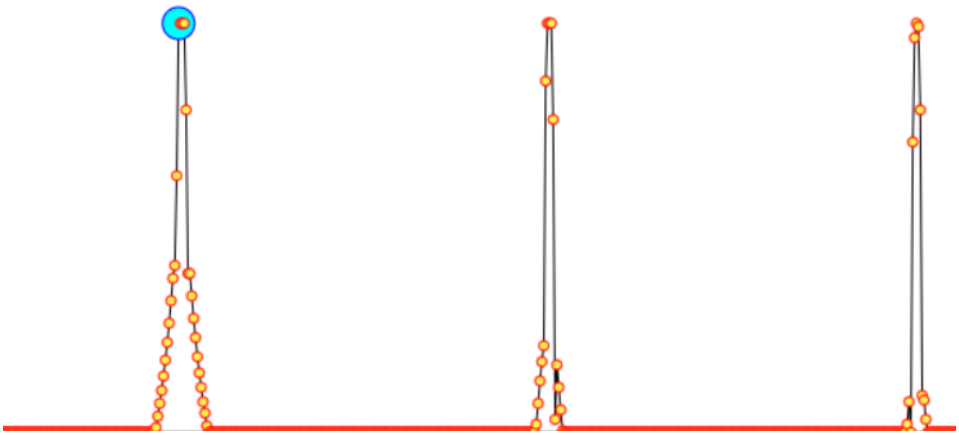
Obrázek 6.3: Vizualizace FFT pro vstup z kytary

Při testování jsem si také všiml, že hodnoty, které by byly dobře použitelné pro nalezení maxima, dostaneme až po ustálení (viz obrázky 6.4 a 6.5). Za tento jev může především zahlazovací konstanta, další atribut `AnalyserNodu`, který nám říká, jak moc se zkombinuje starší vzorek dat s tím novým. Může nabývat hodnot od nuly po jedna včetně. Nula značí, že se starší vzorek neprolne vůbec a v tu chvíli nám metoda getByteFrequencyData vrátí čistá, nová data z `AnalyserNodu`. Hodnota jedna naopak značí, že vezme kompletně celý starý vzorek a zobrazí se místo toho nového, což u nás způsobí zastavení animace. Defaultní hodnota je 0,8, která animaci dost zpomalí. Příčinou tohoto zpomalení je skutečnost, že `AnalyserNode` vezme starý vzorek a vynásobí ho konstantou. Poté přidá nový, vynásobený doplňkem konstanty do 1. Takže hodnoty sice stoupají, ale potřebujeme na to pár cyklů.



Obrázek 6.4: Vizualizace FFT před ustálením

Protože nám jde o co nejrychlejší vyhledání vrcholu, měřil jsem hodnoty se zahlazovací konstantou rovnou nule.



Obrázek 6.5: Vizualizace FFT po ustálení

Dalším problémem je rychlost metody. Naplnění pole o 16384 položkách je zdlouhavé, zvlášť pokud se cyklicky opakuje v animační smyčce. Při dobrém větru zvládá tuto činnost desktop v průměru za 4 ms a chytrý telefon za 14 ms. Pokud bychom snížili počet generovaných hodnot, snížila by se přesnost, která ani při nejvyšší možné `fftSize` není dostačující.

6.2.4 Konkluze

Největší přesnosti dosáhneme nastavením atributu `fftSize` na nejvyšší možnou hodnotu, tedy 32768. Počet frekvenčních pásem je pak 16384 a jedno má rozsah přibližně 1,3458 Hz. Lepší přesnosti také dosáhneme použitím metody getFloatFrequencyData. Ve výstupním poli najdeme jen jeden nejvyšší prvek, při použití druhé metody jsme mohli najít více prvků s hodnotou 255.

Rychlost můžeme ovlivnit nastavením zahlazovací konstanty na nulu.

Pokud bychom chtěli používat tuto metodu k identifikaci frekvence zvuku, byla by nejen pomalá, ale také celkem nepřesná.

6.3 Autokorelace

Slovo autokorelace je složeno ze dvou slov. Auto znamená se sebou samým a korelace znamená porovnání. Autokorelace, jak název napovídá, tedy posuzuje shodu signálu se stejným signálem, který je posunutý v čase.

6.3.1 Použití autokorelace

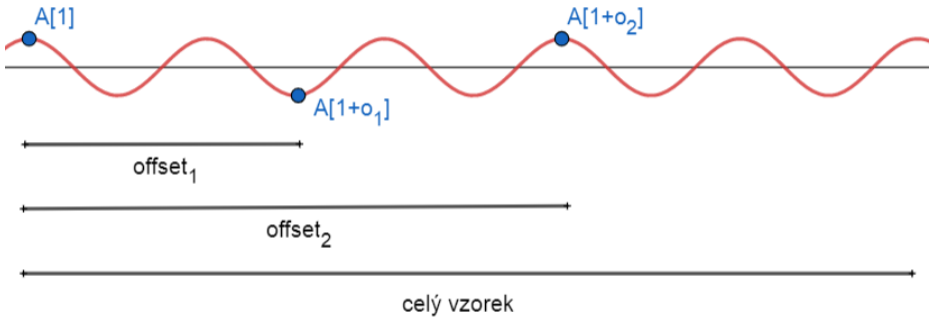
Autokorelace nám porovná podobnost signálu se svou jinou částí, typicky posunutou o několik kroků. Tyto kroky budu také nazývat offsety. Pokud je signál periodický, pak bude posunutá část signálu dokonale korelovat s originální částí za předpokladu, že časové zpoždění, tedy počet offsetů, je celé číslo.

Autokorelace vychází z předpokladu, že když vzorky nějaké známe periodické funkce posuneme o periodu, neměly by se vůbec změnit.

Pomocí offsetu jsme schopni dopočítat frekvenci. Vzorkovací frekvence nám udává, kolik vzorků vznikne za jednu vteřinu. Frekvence je počet opakování nějakého děje za jednotku času (pro nás 1 sekunda), lze tedy tvrdit, že vydělením vzorkovací frekvence tím správným offsetem získáme frekvenci daného zvuku. Frekvenci značíme f, vzrokovací frekvenci neboli sample rate pak SR a offset o.

$$f = \frac{SR}{o}$$

Vhodný offset zvolíme dle následujícího postupu. Vybereme si část signálu, tu označíme A. Poté vezmeme první hodnotu A, označme ji $A[1]$, a tu porovnáme s hodnotou $A[1 + o]$, kde o je offset neboli o kolik kroků je druhá část signálu posunutá. Stejný postup aplikujeme pro další hodnoty signálu A. Tento proces následně opakujeme pro všechny použitelné offsety. Na obrázku 6.6 můžeme vidět, že offset o_1 rozhodně nebude mít tak dobrou shodu jako offset o_2.



Obrázek 6.6: Ukázka porovnávání hodnot signálu

Samotné porovnání můžeme dělat dvěma způsoby, prvním je prostý rozdíl hodnot. Hodnotu $A[i]$ odečteme od $A[i + o]$ a zjistíme absolutní hodnotu tohoto rozdílu. Následně sčítáme rozdíly všech porovnání. Čím nižší je tento součet, tím podobnější si jsou tyto části signálu. Zjednodušený postup výpočtu autokorelace touto metodou pro konkrétní offset o by vypadal takto:

- Vybrat část signálu
- Najít hodnotu signálu v čase i - najít $A[i]$
- Najít hodnotu signálu v čase $i + o$ - najít $A[i + o]$
- Zjistit absolutní hodnotu rozdílu těchto dvou hodnot
- Opakovat pro celou délku vzorku

Vzorec na výpočet autokorelace tímto způsobem pro konkrétní offset o by pak vypadal takto:

$$R(o) = \int_0^i |A[i] - A[i+o]|\, di$$

Druhou možností je násobení hodnot. Hodnotu $A[i]$ vynásobíme hodnotou $A[i+o]$ a všechny násobky sečteme. Čím vyšší je součet všech násobků, tím podobnější jsou si části signálu. Zjednodušený postup výpočtu autokorelace touto metodou pro konkrétní offset o by vypadal takto:

- Vybrat část signálu
- Najít hodnotu signálu v čase i - najít $A[i]$
- Najít hodnotu signálu v čase $i + o$ - najít $A[i+o]$
- Vynásobit tyto dvě hodnoty spolu
- Opakovat pro celou délku vzorku

Vzorec na výpočet autokorelace tímto způsobem pro konkrétní offset o by pak vypadal takto:

$$R(o) = \int_0^i A[i] \cdot A[i+o] di$$

Testování zmíněných metod mi ukázalo, že násobení dává lepší výsledky autokorelace než prostý rozdíl. Dále budu tedy autokorelaci zkoumat prostřednictvím použití druhé metody - tedy pomocí násobení.

6.3.2 Postup

Použil jsem stejnou webovou stránku jako pro testování rychlé Fourierovy transformace, na které jsem chtěl vyzkoušet, co mi může `AnalyserNode` říci o časovém spektru. Pro generování dat z časového spektra můžeme použít metody `getByteTimeDomainData` a `getFloatTimeDomainData`. Obě se opět liší jen v podobě výstupu. První nám do pole vloží bytové hodnoty a druhá desetinná čísla v intervalu -1 až 1.

Pro lepší pochopení si představme pole naplněné metodou `getFloatTimeDomainData` jako řadu po sobě jdoucích hodnot, které společně definují tvar zvukové vlny. Každý člen této řady obsahuje určitou malou část časového spektra zvuku. Velikost této části závisí na hodnotě `fftSize` z `AnalyserNodu`. Tu jsem nastavil na 4096, což mi poskytlo přibližně $4096/48000 = 85\ ms$ vzorku v každé iteraci. Každý člen pole má v sobě tedy hodnotu, která popisuje přibližně $85/2048 = 0,04\ ms$.

Pokusil jsem se vizualizovat získané hodnoty pomocí canvasu. Jednotlivé hodnoty jsem zobrazil na ose y (amplituda) a souřadnice x mi reprezentovala čas. Výsledkem byl statický graf zvukové vlny. Po přidání metody `requestAnimationFrame` jsem dostal pohybující se animaci osciloskopu. [20]

Původně jsem používal metodu rozdílu, ale ta občas vracela nekonzistentní výsledky, zaměřil jsem se proto na druhou metodu, tedy porovnání za pomocí násobení hodnot.

Testovací skript, především metodu sloužící k nalezení frekvence, bylo potřeba optimalizovat. Zaměřil jsem se jen na offsety, které byly použitelné pro rozsah kytary. Nejnižší struna kytary má tón E_2 a její frekvence je přibližně 82,41 Hz. Této frekvenci nejlépe odpovídá offset 582. Nejvyšší struna kytary má tón o 2 oktávy vyšší, tedy E_4. Její frekvence je přibližně 329,63 Hz a nejlépe ji odpovídá offset 146.

Zároveň je potřeba předpokládat, že kytara bude rozladěná. Je tedy nutno přidat pár hodnot navíc. Vytvořil jsem si tabulku všech použitelných offsetů, ty byly v rozsahu od 142-619, což odpovídá zaokrouhleným frekvencím 338 Hz a 78 Hz.

Použitá metoda vypadala následovně. [18] [19] [21]

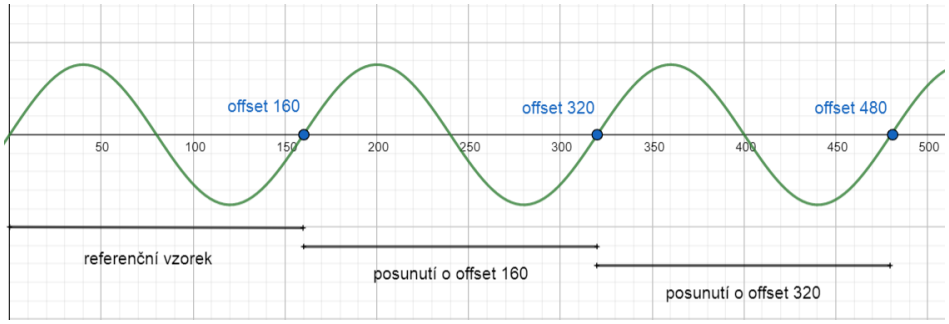
```
/*
class method for finding frequency
argument: data - new data from analyserNode (getFloatTimeDomainData)
*/
findFrequency(data) {

    function compare(a, b) {
        return b.sum - a.sum;
    }

    /* We cannot use the whole length of data */
    let halfDataLength = Math.floor(data.length * 0.5);
    /* Sum of multiplications*/
    let sum = 0;
    /* Array of objects, OFFSETS is array of all usable offsets*/
    let allSums = OFFSETS.map(o => {
        return {'offset': o, 'sum': 0};
    });
    let sampleRate = 48000;

    /* Cycle for all usable offsets */
    for (let o = 0; o < allSums.length; o++) {
        sum = 0;
        /* Cycle for the sample length */
        for (let i = 0; i < halfDataLength; i++) {
            /* sum of multiplication */
            sum += data[i] * data [i + allSums[o].offset];
        }
        allSums[o].sum = sum;
    }
    /* Sort, greater sum is better */
    this.allSums.sort(compare);

    /* Returning frequency */
    return sampleRate / allSums[0].offset;
}
```

Listing 6.2: Metoda k nalezení dominantní frekvence pomocí autokorelace

6.3.3 Problémy

Prvním problémem, kterého jsem si všiml, bylo nesprávné určení frekvence. Zde jsem vypozoroval jisté souvislosti. Při nastavení oscilátoru na 300 Hz mi skript běžně vrátil výsledek 150 Hz nebo 100 Hz. Nebylo tomu tak vždy, protože i data z oscilátoru se pokaždé trochu lišila.

Pro uvedený jev mám následující vysvětlení. Předpokládejme sinusoidu, která formuje zvukouvou vlnu o frekvenci 300 Hz. Tato sinusoida má periodu, která odpovídá nějakému počtu offsetů, velmi pravděpodobně nebude odpovídat úplně přesně. Tento počet offsetů je konkrétně 160 při vzorkovací frekvenci 48000 Hz. Další perioda se pak nachází u offsetu 320, 480 atd. Tyto offsety odpovídají frekvencím polovičním, třetinovým apod. Důvodem, proč je někdy násobný offset označen jako ten správný je to, že offsety přesně nekorelují s periodou a může se tak stát, že násobný offset bude bližší periodě než ten originální. Vznik polovičních a třetinových hodnot výsledku autokorelace signálu je uveden na obrázku 6.7.



Obrázek 6.7: Ukázka vzniku polovičních a třetinových hodnot výsledku autokorelace signálu

Dalším problémem byla přesnost metody. Počet offsetů je celkem omezený, takže můžeme dostat jen omezený počet frekvencí. Se stoupající frekvencí klesá přesnost. Např.: pro strunu E_2 autokorelace najde největší shodu signálu s offsetem 146, což odpovídá frekvenci $48000/146 = 328,77\ Hz$. Správná frekvence struny E_2 je 329,63 Hz. Není to velký rozdíl, přesto by se struny dávající vyšší tóny ladily hůře.

Pokud bychom chtěli více offsetů a tím pádem více získatelných frekvencí, potřebovali bychom používat offsety, které nejsou celá čísla. Tuto úvahu se mi nepodařilo uvést do praxe v rámci skriptu, protože vlnu mi reprezentuje pole, které má omezenou délku. Pokud bychom chtěli získat více dat a tedy delší pole, museli bychom zvýšit `fftSize`. To bychom mohli udělat pouze na úkor rychlosti detekce.

6.3.4 Konkluze

Autokorelace se ukázala být poměrně spolehlivá i rychlá. Samotnou metodu lze implementovat dvěma způsoby, a to rozdílem hodnot nebo součinem hodnot. Druhý způsob se jeví jako lepší z nich. Problémy zmíněné v předešlé

kapitole jsou řešitelné a neomezují funkčnost ladičky.

6.4 Porovnání metod

Otestoval jsem metody na sto pseudonáhodně vybraných frekvencích z oscilátoru a zajímala mě rychlost a správnost detekce dominantní frekvence. Oscilátor jsem napojil na dva analyzéry, na kterých běžely testované metody. Musel jsem použít dva, protože `fftSize` se pro obě metody liší. Po vypočítání průměru jsem dostal následující výsledky. Odchylka rychlé Fourierovy transformace byla 23,13 Hz, pro autokorelaci 16,57 Hz. Časově na tom byly obě metody dost podobně. Nalezení jedné frekvence pomocí FFT trvalo v průměru 11 ms na desktopu a 30 ms na chytrém telefonu. Pro autokorelaci to bylo 10 ms na desktopu a 26 ms na chytrém telefonu.

Pro návrh ladičky jsem tedy vybral metodu autokorelace, a to z důvodu možnosti velmi dobré optimalizace. Jak v rychlosti, např.: pomocí omezení offsetů, tak v přesnosti, pomocí filtrování nekvalitního zvuku, výběru možných kandidátu apod. Oproti tomu rychlou Fourierovu transformaci už moc optimalizovat nelze.

Kapitola 7
Analýza a návrh

Aplikaci jsem nazval GT-une, v textu k ní budu odkazovat jako k GT.

7.1 Požadavky

Mezi nejdůležitější požadavky patří:

- Instrukce k ladění – sloužící k informování uživatele o snížení či zvýšení frekvence struny
- Informace o laděné struně – zobrazení tónu struny, která je pravděpodobně laděná
- Rychlost – aby byla ladička použitelná, je nutno dát uživateli včasnou odezvu
- Použitelnost – aplikace fungující ve webových prohlížečích chytrých telefonů
- Praktický design – jednoduše prezentující výše zmíněné požadavky

Protože jde o poměrně omezenou aplikaci, co se týče využití, rozdělování požadavků na funkční a nefunkční moc nedává smysl.

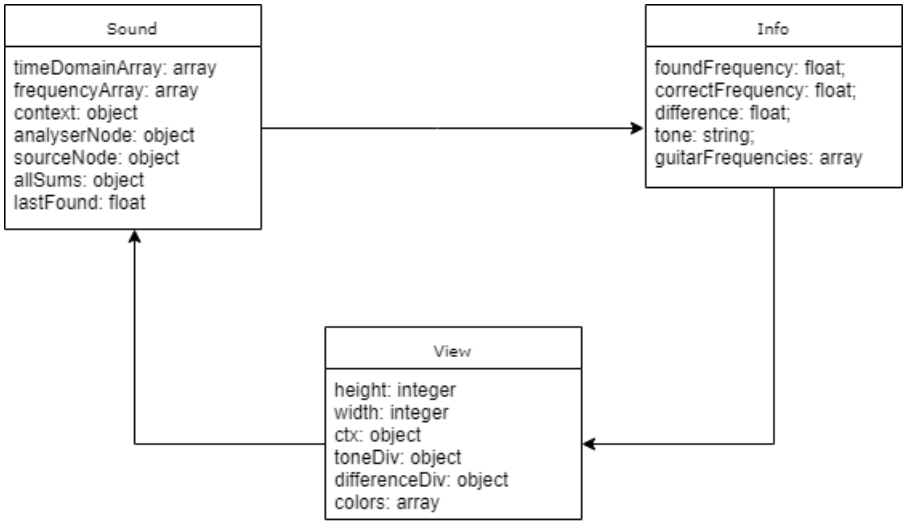
7.2 Architektura

Architektura aplikace GT je poměrně jednoduchá. K sestrojení funkční použitelné ladičky potřebujeme tyto komponenty – načítání zvuku a jeho analýzu (třída Sound), informace o ladění (třída Info), grafiku (třída View).

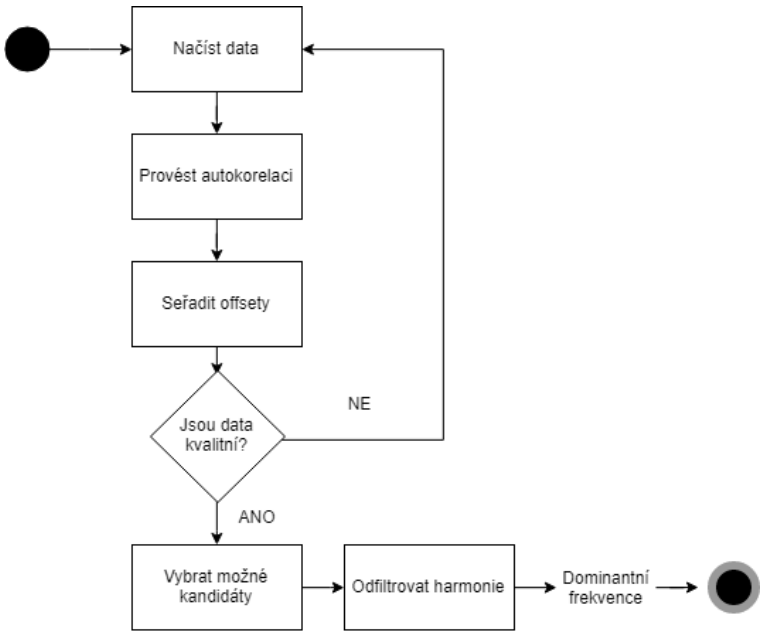
Protože komponent není mnoho, UML class model má kruhovou strukturu a jeho podobu znázorňuje obrázek 7.1.

Dokumentaci jednotlivých tříd uvádím v kapitole 9 Implementace.

Stavový diagram na obrázku 7.2 popisuje činnost aplikace GT. Je zde zřejmý postup detekce dominantní frekvence. Rozhodovacím krokem o kvalitě dat chceme ušetřit procesorový čas.



Obrázek 7.1: UML class model aplikace GT



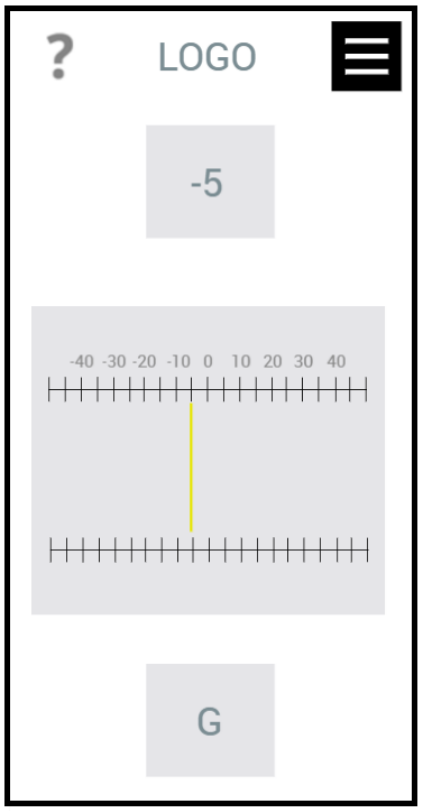
Obrázek 7.2: Stavový diagram aplikace GT

Kapitola 8

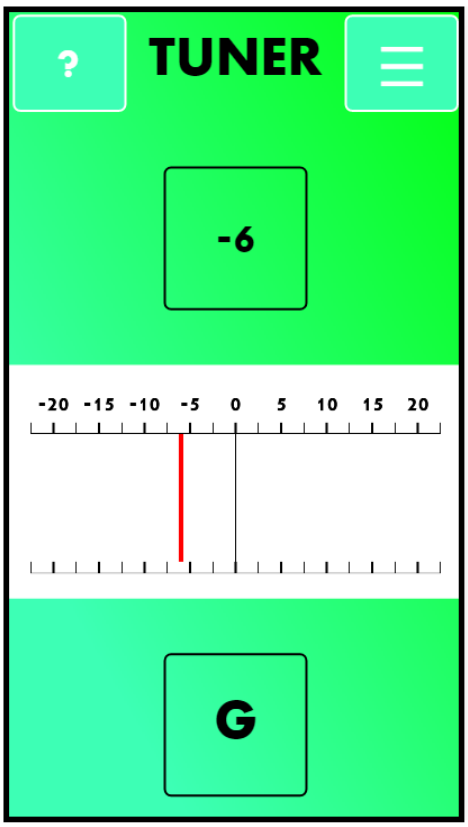
Design

Výsledný vzhled mobilní aplikace je důležitou součástí celého vývoje. Design určí, zda bude aplikace mezi uživateli oblíbená či ne. U mobilních aplikací, jak nativních, tak webových, jsou lidé zvyklí na poměrně jasně stanovená pravidla užívání, které jsou dána architekturou operačních systémů telefonů. Ve svém návrhu jsem se snažil respektovat tyto pravidla.

Jednoduchost je silnou stránkou GT. Aplikace musí uživateli předat informace o laděné struně a odchylce frekvence v prosté formě. Navrhnul jsem design, který splňuje funkční požadavek zmíněný výše. Jeho zjednodušenou podobu představuje obrázek 8.1., finální podobu pak obrázek 8.2.



Obrázek 8.1: Prvotní návrh aplikace GT



Obrázek 8.2: Finální podoba aplikace GT

Kapitola 9
Implementace

V této kapitole se zabývám konkrétní implementací navrhnutého řešení. Rozebírám zde jednotlivé třídy aplikace GT, jejich vznik, funkce a omezení.

9.1 Použité technologie

Protože jde o webovou aplikaci, použití JavaScriptu bylo nevyhnutelné. Aplikace je tedy složena pouze z HTML a JavaScriptu. Je psána jako SPA, ke stylování byl použit jazyk CSS.

Pro spolehlivou funkčnost aplikace jsou nezbytná tyto JavaScriptová API - Web Audio API a Stream API.

9.2 Omezení

Výhodou webových aplikací je přenosnost mezi jednotlivými platformami. Na druhou stranu zjevnou nevýhodou jsou nesjednocené prohlížeče. Vývojáři běžně tvoří více verzí pro jednotlivé z nich, případně jsou nuceni stylovat aplikaci pomocí prefixů.

Aplikace byla vytvářena čistě pro mobilní zařízení. Na desktopových zařízeních se nezobrazí správně, ale lze ji také používat po přepnutí do režimu chytrých telefonů ve vývojářské konzoli.

Aplikace je optimalizována pouze pro prohlížeč Google Chrome. Při testování jsem opět měřil dobu nalezení dominantní frekvence a přesnost. Testoval jsem pouze nejpoužívanější prohlížeče pro platformu Android, tedy Google Chrome a Firefox.

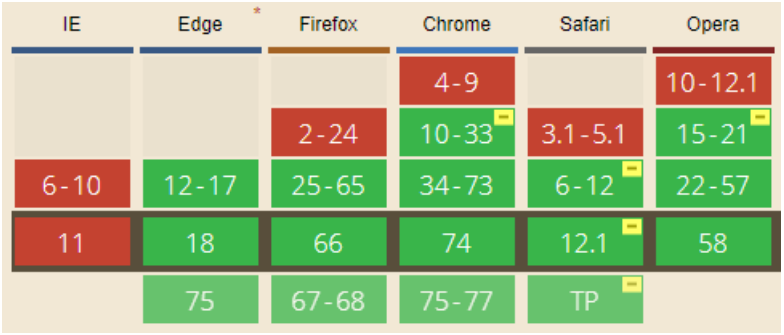
V rychlosti byly oba prohlížeče přibližně vyrovnané. Nalezení frekvence trvalo přibližně 25 ms. Větší rozdíly se projevily v přesnosti. Google Chrome detekoval frekvence s přesností 4,49 Hz, Firefox s přesností 19,59 Hz. Naladit kytaru s využitím Firefoxu nebylo prakticky možné.

Určitého zlepšení se mi podařilo dosáhnout změnou konstant (viz podkapitola vývoj), ale v tu chvíli se zase zhoršil výkon aplikace v Google Chrome.

Pro případ dalšího zkoumání této oblasti uvádím omezení použitých API v dalších prohlížečích.

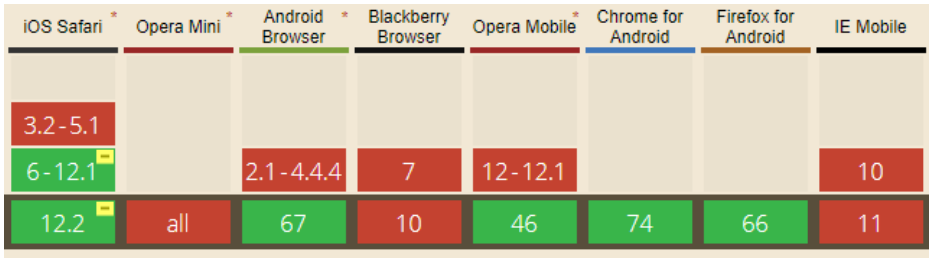
9.2.1 Web Audio API

Web Audio API je podporováno většinou moderních prohlížečů, jak na desktopech, tak chytrých telefonech. Jediným zarytým odpůrcem této technologie je Internet Explorer, a to pro obě platformy. Obrázky 9.1 a 9.2 znázorňují přehled nejčastěji používaných prohlížečů a verzí, které podporují Web Audio API.



[26]

Obrázek 9.1: Přehled nejčastěji používaných prohlížečů a verzí, které podporují Web Audio API pro desktopová zařízení



[26]

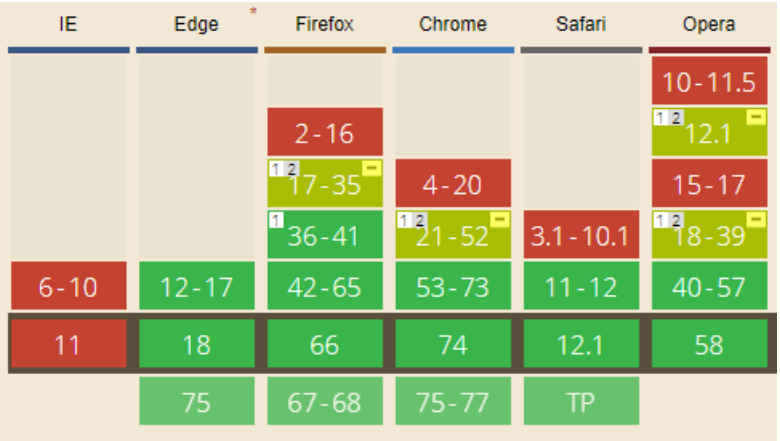
Obrázek 9.2: Přehled nejčastěji používaných prohlížečů a verzí, které podporují Web Audio API pro chytré telefony

9.2.2 Stream API

Situace Stream API je trochu složitější. Přístup ke zvuku a videu z prohlížeče byl dříve poskytován přímo z rozhraní Navigator - `Navigator.getUserMedia()`. Navigator je objekt, který reprezentuje prohlížeč uživatele. S přibývajícími funkcemi prohlížeče rostl i počet proměnných a metod Navigatoru. Proto byl celý Navigator restrukturalizován.

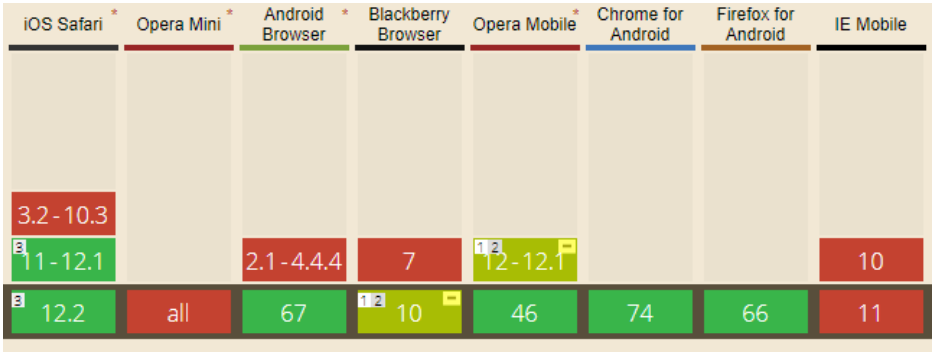
V dnešní verzi JavaScriptu najdeme přístup ke zvuku a videu ve Stream API - `navigator.mediaDevices.getUserMedia()`.

Na obrázcích 9.3 a 9.4 je uveden přehled nejčastěji používaných prohlížečů a verzí, které podporují Stream API.



[27]

Obrázek 9.3: Přehled nejčastěji používaných prohlížečů a verzí, které podporují Stream API pro desktopová zařízení



[27]

Obrázek 9.4: Přehled nejčastěji používaných prohlížečů a verzí, které podporují Stream API pro chytré telefony

9.3 Struktura projektu

V této podkapitole vysvětluji finální podobu ladičky. Zabývám se také proměnnými a metodami jednotlivých komponent.

9.3.1 Sound - Načítání a analýza zvuku

Třída Sound reprezentuje vstup do aplikace. Data nutná k detekci dominantní frekvence jsou získávána prostřednictvím mikrofonu zařízení. Použití mikrofonu nikdy nebude tak přesné jako by bylo použití chromatické ladičky, která využívá vibrace krku kytary, ale pro splnění úkolu bude stačit.

Analýza zvuku je pak prováděna pomocí `AnalyserNodu`, který nám poskytne data jak o časové, tak o frekvenční části spektra.

Mezi třídní proměnné patří:

- `timeDomainArray` - předdefinované pole, do kterého se ukládají data z metody getFloatTimeDomainData `AnalyserNodu`
- `frequencyArray` - předdefinované pole, do kterého se ukládají data z metody getFloatTimeFrequencyData `AnalyserNodu`
- `context` - objekt zodpovídající za celé Web Audio API
- `SR` - vzorkovací frekvence daného prohlížeče
- `binSize` - rozsah jednoho frekvenčního pásu vystupujícího z FFT
- `analyserNode` - `AnalyserNode` Web Audia API
- `sourceNode` - node zodpovědný za stream zvuku z mikrofonu
- `info` - komponenta Info
- `allSums` - předdefinovaný objekt offsetů, ke kterým se přiřazují výsledky autokorelace, součty násobků
- `lastFound` - v případě chybně nalezené hodnoty se zobrazí poslední

Mezi metody třídy patří:

- `setUp` - nastavuje konstanty a další proměnné
- `connectToMic` - zkontroluje přístup k mikrofonu a připojí stream z něj
- `hasGetUserMedia` - kontroluje, zda má prohlížeč přístup k Stream API
- `updateData` - vygeneruje novou sadu dat z frekvenčního i časového spektra
- `findFrequency` - autokorelace samotná (viz stavový diagram na obrázku 7.2)
- `isSoundGood` - filtr zvuku (když je vzorek nekvalitní a součet násobků není nad určitou hodnotou, vygeneruje se nový)
- `chooseCandidate` - vybere prvních x kandidátů (tato metoda není použita ve finálním produktu)
- `filter` - odfiltruje harmonie, výstupem by měla být nalezená dominantní frekvence (tato metoda není použita ve finálním produktu)
- `getIndexOfMaximum` - hledá pás, ve kterém je dominantní frekvence, pomocí FFT (tato metoda není použita ve finálním produktu)
- `notifyInfo` - předá informace třídě Info

9.3.2 Info - Informace o ladění

Třída Info slouží k identifikaci pravděpodobně laděné struny a k výpočtu odchylky od správné frekvence této struny. Zformuje data získaná třídou Sound do čitelnější podoby a předá je třídě View.

Mezi třídní proměnné patří:

- `view` - komponenta View
- `foundFrequency` - nalezená frekvence předaná třídou Sound
- `correctFrequency` - frekvence odhadnuté struny, o kterou se pravděpodobně jedná
- `difference` - rozdíl foundFrequency a correctFrequency
- `tone` - tón odpovídající odhadnuté struně
- `guitarFrequencies` - objekt obsahující předdefinované frekvence kytarových strun a odpovídající tónové pojmenování

Mezi metody třídy patří:

- `whatString` - identifikuje strunu podle frekvence
- `count` - podle metody `whatString` doplní třídní proměnné
- `notifyView` - předá dopočítané informace třídě View

9.3.3 View - Grafika

Třída View má přístup k vykreslovací ploše. Podle dat dopočítaných ve třídě Info vypíše tón laděné struny, vykreslí odchylku na stupnici a zobrazí ji v číselné podobě.

Mezi třídní proměnné patří:

- `height` - výška kreslícího plátna
- `width` - šířka kreslícího plátna
- `ctx` - objekt zodpovídající za celý canvas
- `sound` - třída Sound
- `toneDiv` - HTML tag, do kterého se vypisuje tón laděné struny
- `differenceDiv` - HTML tag, do kterého se vypisuje odchylka identifikované a správné frekvence
- `colors` - předdefinované barvy. Odchylka se vykreslí v barvě podle vzdálenosti od nuly.

Mezi metody třídy patří:

- `init` - nastavuje parametry kreslícího plátna
- `drawSkeleton` - vykreslí do plátna kostru, která se nemaže (stupnice pro odchylku)
- `clear` - vymaže měnící se část plátna
- `draw` - vykreslí nová data
- `drawDownScale` - vykreslí spodní část stupnice
- `drawUpScale` - vykreslí horní část stupnice
- `drawScale` - vykreslí celou stupnici
- `writeInfo` - zapíše získaná data do předdefinovaných HTML tagů
- `drawTuning` - vykreslí odchylku do plátna
- `drawMiddleLine` - obnoví středovou linii stupnice
- `drawNumbers` - vykreslí čísla na stupnici
- `drawLine` - metoda na zjednodušení kreslení čar v canvasu

9.4 Vývoj

Pro vývoj finálního produktu jsem vybral metodu autokorelace. Podle navrhnuté architektury jsem vytvořil třídy a použil testovaný algoritmus autokorelace.

První testování odhalilo několik nedostatků. Prvním z nich byla vysoká spotřeba paměti. Bez ohledu na to, že JavaScript obsahuje účinný garbage collector, spotřeba paměti byla opravdu vysoká. Hlavním důvodem bylo realokování několika proměnných. Především pak pole objektů `allSums`, které obsahuje offsety a k nim odpovídající součty násobků. Toto pole bylo opět vytvořeno v každém cyklu. Řešením tohoto nedostatku bylo definování těchto proměnných jako třídní. Snažil jsem se najít ideální hranici mezi ušetřenou pamětí a dobrou čitelností kódu.

Dalším problémem bylo použití stream API, konkrétně `getUserMedia`. Při testování aplikace na localhostu vše fungovalo v pořádku, problém nastal při nasazení na webu. Postoj firem k bezpečnosti aplikací se různí. Google Chrome zablokuje uživateli možnost použít kameru nebo mikrofon, pokud web neběží přes šifrovaný protokol https. Firefox povolí streamování i na nešifrovaném protokolu http. Řešením bylo přidání https na hosting.

Po prvním testování přišla na řadu optimalizace. Zlepšení rychlosti a přesnosti jsem dosáhl pomocí následujících změn: omezení offsetů, přidání prahování, lepší animační smyčky, lepšího výběru kandidátů a filtrování falešných frekvencí.

9.4.1 Omezení offsetů

Je zbytečné iterovat přes offsety odpovídající tónům, které kytara nemůže nikdy dosáhnout. Nejnižší tón kytary E_2 má frekvenci 82,41 Hz, po přidání možné odchylky jsem určil hodnotu 70 Hz jako nejnižší zkoumanou frekvenci. Jako nejvyšší pak frekvenci 365 Hz.

9.4.2 Přidání prahování

V případě získání nekvalitního vzorku dat je zbytečné hledat dominantní frekvenci, protože by pravděpodobně byla určena špatně. Kvalitu dat určujeme podle velikosti součtu násobků vycházejících v autokorelaci (proměnná `allSums`).

Testoval jsem několik hodnot této konstanty. Při použití oscilátoru se můžeme dostat na součet o hodnotě několika set, avšak s daty z kytary jsou součty většinou nižší, nejlepší kolem 150, průměrně však kolem 2.

Samozřejmě záleží na mnoha dalších faktorech, jako například druh mikrofonu, kytary, vzdálenost kytary od mikrofonu a ruch prostředí. Těmto faktorům by bylo dobré přizpůsobit hodnotu konstanty.

Použití vyšší konstanty (např.: 30 nebo 20) způsobí nedostatek dat, se kterými můžeme pracovat. Většinou jsem musel několikrát rozeznít strunu, aby ladička získala dostatečně kvalitní data.

Použití nižší konstanty (např.: 0,005) způsobí provedení autokorelace nad horšími daty.

Zkoušel jsem i další hodnoty (např.: 2, 5, 0,5), ale nakonec jsem vybral hodnotu 10, se kterou jsem dostával nejlepší výsledky. Hodnot bylo dost a detekce většinou určila frekvenci správně.

9.4.3 Animační smyčka

Funkce `requestAnimationFrame` provede kód každých 16 ms, což odpovídá přibližně 60 FPS. Do této hodnoty se nevejde provedení celého jednoho cyklu detekce dominantní frekvence. Použití této hodnoty by mohlo vyústit v neurčené chování JavaScriptu. Lepších výsledků jsem dosahoval použitím metody `setTimeout` s opakováním každých 100 ms.

9.4.4 Lepší výběr kandidátů

V ideálním světě by nám pouhé seřazení `allSums` mělo na první místo dát offset dominantní frekvence (favorit), nicméně realita je trochu jiná. Často se na prvních několika místech nacházejí násobky offsetů. Otázkou je, kde se favorit opravdu nachází.

Vybral jsem možnost vzít pouze prvních 10 hodnot pole `allSums`. V těch jsem většinou našel hodnotu dominantní frekvence. Dalším možným přístupem bylo vybrání hodnot, které se lišily maximálně o n% od součtu násobku hodnoty první. Tato možnost je časově náročnější, protože je nutné projít celé pole `allSums`.

9.4.5 Filtrování falešných frekvencí

Z vybraných kandidátů nakonec musíme odfiltrovat ty špatné tak, aby nám zůstaly nejlépe jen offsety odpovídající přibližně stejné frekvenci. Z těch pak vybereme tu první, neboli tu, která má nejvyšší součet násobků.

Jak už bylo zmíněno v části 6.3.1, autokorelace může vrátit dobré výsledky i pro vyšší offsety (tedy nižší frekvence). Tyto nižší frekvence můžeme odfiltrovat buď manuálně nebo pomocí FFT.

První možností je projít v prvním cyklu (A) všechny kandidáty a v druhém (B) porovnat jejich offsety s ostatními. Když je offset z A přibližně poloviční než ten z B, pak smažeme kandidáta z B.

Druhou možností je použití rychlé Fourierovy transformace. Můžeme použít `fftSize` nastavenou na hodnotu pro autokorelaci (4096). Tím dostaneme pásy o šířce kolem 11 Hz. V cyklu projdeme všechny kandidáty, když narazíme na kandidáta, který spadá do pásu nalezeného pomocí FFT, tak jsme našli dominantní frekvenci.

Výše uvedená první možnost má několik nevýhod. Jednou z nich je, že slovo „přibližně“ nelze v programování úplně dobře definovat. Poloviční hodnoty se mohou lišit o jednotky, ale také o desítky. Další nevýhodou je nízká rychlost. Použití FFT je rozhodně rychlejší.

Poslední testování nakonec ukázalo, že použití filtrování ani není potřeba v případě, že zvýšíme `fftSize`. Při `fftSize` nastavené na 8192 mi autokorelace vracela dostatečně kvalitní výsledky. K určení dominantní frekvence jsem použil offset s nejvyšší hodnotou součtu násobků (`allSums[0]`). Autokorelace s ještě vyšší `fftSize` (16384 nebo 32768) dosahovala stejných výsledků, ale byla pomalejší.

Kapitola 10

Závěr

V této práci jsem se zabýval detekcí dominantní frekvence díky níž jsem vytvořil ladičku pro kytaru pouze pomocí HTML, CSS a JavaScriptu.

V angličtině název práce zní Pitch Detection. Zaujala mě skutečnost, že čeština je v tomto ohledu mírně omezená. Pitch se často překládá jako dominantní/základní frekvence, přestože korektnější překlad by byl čistý tón. V angličtině je mezi slovy „pitch“ a „fundamental frequency“ poměrně důležitý rozdíl. Jak uvádí Ron Nicholson [28] – jedním z důvodů, proč čistý tón není vždy dominantní frekvence, je to, že spousta příjemných zvuků obsahuje mnoho harmonií a podtónů (to jsou tóny patřící do vyšších harmonií), které mohou vynikat více než tón s dominantní frekvencí. Právě harmonie a podtóny jsou to, co dělá tyto zvuky zajímavé. Pokud bychom tvořili zvuky bez nich, náš svět by vypadal jako oscilátor generující pouze jednotlivé sinusoidy.

Ve chvíli, kdy nástroj vydává jeden čistý tón, jeho tělo dovytváří díky rezonanci další tóny (vyšší harmonie). Tyto vyšší harmonie posluchač může vnímat dokonce jako výraznější. To samé platí i pro zvuky generované hlasivkami, zesilovačem je pak lebka.

Uvedený jev lze snadno ověřit. Pomocí zvukového editoru a jeho vestavěného oscilátoru vytvoříme vlnu o vysoké frekvenci, např.: 1568 Hz (ta odpovídá tónu G_6). Přehrajeme ji a následně zkrátíme trvání tohoto tónu tak, aby byl kratší než 10 ms. Poté tento segment necháme opakovat stokrát za vteřinu. Po spuštění uslyšíme mnohem nižší tón než na začátku. Posluchač nyní neslyší tón G_6 o frekvenci 1568 Hz, ale jiný tón přibližně G_2 o frekvenci asi 100 Hz. Ačkoliv FFT určí jako dominantní frekvenci 1568 Hz, protože tato frekvence tvoří převážnou většinu křivky, čistý tón má frekvenci přibližně 100 Hz.

Čistý tón se tedy liší od dominantní frekvence a také detekce a odhad hudebního tónu se liší od odhadu frekvence. Odhady čistého tónu hledají periodicitu, ne frekvenci.

Během testování jednotlivých metod jsem se seznámil s hudební teorií. V dnešní době celý svět používá rovnoměrně temperované ladění. Po sobě jdoucí tóny stupnice C-dur pro naši kulturu jsou C-D-E-F-G-A-H. Pro zbytek světa by posloupnost tónů ve stupnici C-dur vypadala jako C-D-E-F-G-A-B. Na úkor českých hudebníků, kteří občas hrají s hudebníky z jiných zemí je na území České Republiky běžné označení tónu B jako H. Používání jiného označení pro tón B má čistě praktický důvod, jinak by docházelo ke zbytečným

zmatkům z překrývání slov béčko (posuvka) a béčko (tón B).

I přesto, že velká část programátorů nehodnotí dobře kvalitu jazyku JavaScript, zjistil jsem, že nám může nabídnout mnoho aplikačních rozhraní, se kterými je radost pracovat. Je pravda, že v některých ohledech je složitější ho pochopit, ale kniha Javascript od RNDr. Ondřeje Žáry [29] mi pomohla překonat většinu problémů.

Otestoval jsem obě metody. Metoda autokorelace se ukázala být rychlejší i přesnější, ale především jsem ji mohl optimalizovat. Po nalezení lepší z metod jsem se pustil do vývoje finálního produktu – ladičky na kytaru.

Úkol se mi podařilo splnit. Použití autokorelace na `analyserNodu` s `fftSize` nastavenou na 8192 najde deset nejlepších kandidátů na dominantní frekvenci a následně stačí vybrat prvního z nich.

Přesnost ladičky se nemůže měřit s těmi chromatickými, ale kytara se s ní naladit dá.

Okomentovaný zdrojový kód a spustitelná ukázka se nachází na přiloženém CD, popřípadě na http://www.mojebakalarka.8u.cz/.

Literatura

[1] Statista. Mobile phone internet user penetration worldwide from 2014 to 2019. 2019 [cit. 02.05.2019].
Dostupné z: https://www.statista.com/statistics/284202/mobile-phone-internet-user-penetration-worldwide/.

[2] Aron Kornhall. Pitch detection for mobile devices. 2006 [cit. 06.01.2019].
Dostupné z: http://fileadmin.cs.lth.se/serg/old-serg-dok/docs-masterthesis/109_15-2006.pdf.

[3] Luděk Zenkl. Abc hudební nauky. *Editio Bärenreiter Praha*, 2003, ISBN 80-86385-21-3.

[4] AGADIR. Hudební teorie [online]. 2004 [cit. 07.01.2019].
Dostupné z: https://www.agadir.cz/teorie.php?vyber=1.

[5] Neznámý autor. Hudební teorie [online]. 2004 [cit. 08.01.2019].
Dostupné z: http://webnet.wz.cz/ukazky/hudebni_akustika/Applet/index.htm.

[6] Martin Všetička Jaroslav Reichl. Hudební teorie [online]. 2006 [cit. 10.01.2019].
Dostupné z: http://fyzika.jreichl.com/main.article/view/195-barva-tonu.

[7] B. H. Suits Michigan Technological University. Making sense of cents [online]. 1998 [cit. 09.01.2019].
Dostupné z: https://pages.mtu.edu/~suits/cents.html.

[8] All About Music. Scales and key signatures - the method behind the music [online]. 1999 [cit. 07.01.2019].
Dostupné z:
https://method-behind-the-music.com/theory/scalesandkeys.

[9] Dušan Janovský. Stupnice [online]. 2001 [cit. 07.01.2019].
Dostupné z: https://dusan.pc-slany.cz/hudba/stupnice.htm.

[10] WikiWand. Ukázka diatonické stupnice c-dur. 2012 [cit. 08.01.2019].
Dostupné z: http://www.wikiwand.com/cs/Durov%C3%A1_stupnice.

[11] Hyacinth Wikipedia. Ukázka chromatické stupnice. 2004 [cit. 08.01.2019]. Dostupné z: https://commons.wikimedia.org/w/index.php?curid=1530043.

[12] Interactive Mathematics. What are the frequencies of music notes? [online]. 2017 [cit. 08.01.2019]. Dostupné z: https://www.intmath.com/trigonometric-graphs/music.php.

[13] Rudolf Klusal. Hudba ve 432 vs 440 hz [online]. 2015 [cit. 08.01.2019]. Dostupné z: http://fyzika.klusik.cz/2015/03/09/hudba-ve-432-vs-440-hz/.

[14] MDN. Oficiální dokumentace javascriptu [online]. 2005 [cit. 09.01.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/JavaScript.

[15] Paul Krill. Javascript creator ponders past, future [online]. 2008 [cit. 09.01.2019]. Dostupné z: https://www.infoworld.com/article/2653798/application-development/javascript-creator-ponders-past–future.html.

[16] Ben Aston. A brief history of javascript [online]. 2015 [cit. 09.01.2019]. Dostupné z: https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17.

[17] MDN. Oficiální dokumentace javascriptu [online]. 2005 [cit. 09.01.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[18] MDN. Introduction to web apis [online]. 2005 [cit. 10.01.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction.

[19] MDN. Co je javascript [online]. 2005 [cit. 10.01.2019]. Dostupné z: https://developer.mozilla.org/cs/docs/Learn/JavaScript/First_steps.

[20] MDN. Web audio api [online]. 2005 [cit. 10.01.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API.

[21] MDN. Audio and video delivery [online]. 2005 [cit. 10.01.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Audio_and_video_delivery.

[22] Miguel Amado and Juarez Hoppe Filho. Pitch detection algorithms based on zero-cross rate and autocorrelation function for musical notes. *2008 International Conference on Audio, Language and Image Processing*, pages 449–454, 2008.

[23] Ph.D. Steven W. Smith. The scientist and engineer's guide todigital signal processing - how the fft works. 1997 [cit. 30.03.2019]. Dostupné z: https://www.dspguide.com/ch12/2.htm.

[24] Paul Lewis. Aerotwist - guitar tuner [online]. 2015 [cit. 11.01.2019]. Dostupné z: https://aerotwist.com/blog/guitar-tuner/.

[25] Mike Betthauser. Harmonics - definition (artopium's music dictionary). 2018 [cit. 30.03.2019]. Dostupné z: https://musicterms.artopium.com/h/Harmonics.htm.

[26] Can I Use. Web audio api. 2019 [cit. 30.04.2019]. Dostupné z: https://caniuse.com/search=web%20audio%20api.

[27] Can I Use. Web audio api. 2019 [cit. 30.04.2019]. Dostupné z: https://caniuse.com/search=getusermedia.

[28] Ron Nicholson. Musical pitch is not just fft frequency. 2012 [cit. 27.04.2019]. Dostupné z: http://www.musingpaw.com/2012/04/musical-pitch-is-not-just-fft-frequency.html.

[29] Ondřej Žára. Javascript. *Computer Press*, 2015, ISBN 8025145735.

Příloha A

Seznam použitých zkratek

dB decibel – jednotka hlasitosti zvuku

BPM Beats Per Minute – počet úderů za minutu, jednotka používaná na metronomech

HTML Hyper Text Markup Language – značkovací jazyk používaný k tvorbě webových stránek

CSS Cascading Style Sheets – jazyk pro popis způsobu zobrazení elementů HTML

SPA Single Page Aplication – webová aplikace zobrazující pouze jednu webovou stránku, která dynamicky mění svůj obsah pomocí JavaScriptu

API Aplication Interface – rozhraní aplikace poskytující složitější struktury za pomocí jednodušší syntaxe

FFT Fast Fourier Transform – rychlá Fourierova transformace

DFT Discrete Fourier Transform - diskrétní Fourierova transformace

IDFT Inverse Discrete Fourier Transform - inverzní diskrétní Fourierova transformace

SR Sample Rate - vzorkovací frekvence

FBC Frequency Bin Count - vlastnost analyserNodu

GT-une Guitar Tuner – 1 z francouzského un

FBC Frequency Bin Count – vlastnost analyserNodu

GC Garbage Collector – objekt JavaScriptu starající se o uvolňování paměti mazáním již nepotřebných proměnných

https Hypertext Transfer Protocol Secure – šifrovaná varianta internetového protokolu HTTP pro přenos webových stránek

FPS Frames Per Second – počet snímků za vteřinu

CD Compact Disc – kompaktní disk

Příloha B

Obsah přiloženého CD

GT-une obsahuje zdrojový kód aplikace

pdf obsahuje tuto práci ve formátu PDF

source obsahuje zdrojový kód psané části bakalářské práce