TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Inteligentní ladička v chytrém mobilním telefonu

An intelligent tuner for smartphones

Bakalářská práce

Autor: **Michael Müller**

Vedoucí práce: Doc. Ing. Zbyněk Koldovský, Ph.D.

Konzultant: Ing. Jiří Málek, Ph.D.

V Liberci 17. 5. 2013

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Liberci dne: Podpis:

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Doc. Ing. Zbyňku Koldovskému, Ph.D. za poskytnutí odborných rad, věcné připomínky, ochotu a vstřícný přístup během zpracování této práce.

Velké poděkování náleží celé mé rodině za podporu, trpělivost a povzbuzování po dobu mého studia.

Obsah

Abstrakt

Android je velmi rozšířený operační systém pro chytré mobilní telefony. Nabízí mnoho funkcí, které je možné využít. Aplikace je nativně možné programovat v jazyce Java, který je poměrně jednoduchý, proto je vývoj menších projektů snadný. V této bakalářské práci je popsáno, jak aplikace pro systém Android fungují, jak si nakonfigurovat vývojové prostředí a jakým způsobem byla programována ladička hudebních nástrojů s využitím metody FFT (Fast Fourier Transformation). Tato metoda slouží pro výpočet spektra, ze kterého lze vyčíst frekvenci signálu. Pro zlepšení výpočtu je použita metoda MUSIC (Multiple Signal Classification).

Abstract

Android is a very popular operating system for smartphones. Android has many functions that can be used. Applications can be programmed in Java. This programming language is pretty simple, so the development of small projects is simple as well. In this thesis, the approach to configure the development kit is described. Next, how applications work in Android is described. Then how the tuner for musical instruments was programmed. The FFT (Fast Fourier Transformation) is used in this project. The FFT is used for calculate spectrum. Then, the maximal frequency is found. Method MUSIC (Multiple Signal Classification) was chosen for better calculate spectrum

Klíčová slova

Android, Java, fundamentální frekvence, programování, zvukový signál, Rychlá Fourierova transformace, Multiple Signal Classification

Keywords

Android, Java, fundamental frequency, programming, sound signal, Fast Fourier Transformation, Multiple Signal Classification

Úvod

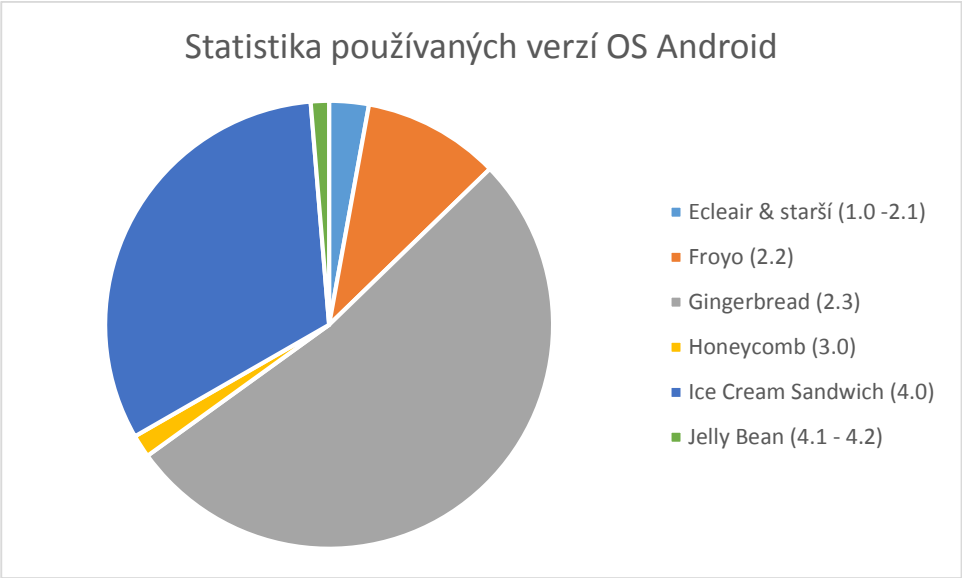
Chytrý mobilní telefon se dnes stává standardem, který už si může dovolit každý. Nabídka chytrých mobilních telefonů je velice široká. Telefony jsou k dispozici s různými specifikacemi a funkcemi, ale také s různými operačními systémy. V této práci je popsán vývoj aplikace pro Android, který se stává nejrozšířenějším operačním systémem. Pro Android jsou již statisíce aplikací, které využívají veškeré funkce, které telefon nabízí. V této aplikaci je využit pouze mikrofon pro sledování zvukového signálu.

V první kapitole je pojednáno o tom, co je to Android, jak se chová a funguje. Dále v kapitole *Vývoj aplikací obecně* je popsáno, bez čeho všeho se neobejdou aplikace pro OS Android. Ve třetí kapitole je popsán způsob zaznamenávání zvukového signálu a použité metody pro měření fundamentální frekvence. V kapitole *Grafické uživatelské prostředí* je vysvětleno navržení grafického rozhraní a způsob jakým je realizováno zobrazování nalezeného tónu a zda je tón přeladěn nebo podladěn. Nakonec jsou uvedeny dvě ukázky funkce této aplikace.

1 Android

Android je již několik let známý operační systém od stejnojmenné společnosti, kterou v roce 2005 koupila společnost Google. Největší boom Androidu ale nastal až v roce 2010, kdy Google uvedl na trh své první Android zařízení s názvem Nexus One.

Android je určený především pro chytré telefony a dnes už i tablety. Bylo vydáno již několik verzí a s každou novou verzí je tento systém dokonalejší a přináší mnoho nových funkcí. Poslední verzí je verze s označením Jelly Bean (Android 4.2.2). Ovšem, nejrozšířenější je stále verze Gingerbread (Android 2.3). Více je možné nalézt v [1].



Obr. 1 Graf instalovaných verzí v zařízeních. Převzato z webu developers.android.com [2].

Android se tak postupně stává nejrozšířenější mobilní platformou pro chytré telefony na světě. Google uvádí více než 1 milion aktivovaných zařízení s OS Android za den. Více zde [2].

1.1 Architektura

Architektura OS Android je rozdělena do 5 vrstev. Každá z nich plní svůj účel, ale může zasahovat i do vrstev ostatních. Nejnižší vrstvou je jádro OS, které je postaveno na jádře Linuxu verze 2.6. Jádro tvoří vrstvu mezi hardwarem a zbytkem software ve vyšších vrstvách.

Další vrstvou jsou knihovny, které jsou napsané v jazyce C/C++, využívají je různé komponenty systému. Vývojářům jsou funkce knihoven poskytovány prostřednictvím Android Application Frameworku. Příklady knihoven:

- OpenGL – knihovna pro vykreslování 3D grafiky
- SQLite – knihovna odlehčené relační databáze
- Media Libraries – knihovna podporující přehrávání audio a video formátů

Android Runtime je třetí vrstva, obsahující virtuální stroj Dalvik (Dalvik Virtual Machine). Dalvik slouží pro spouštění aplikací v systému Android, využívá vlastností Linuxového jádra, jako je správa paměti nebo práce s vlákny. Dalvik byl vyvíjen od roku 2005 společností Google. Vyvíjen byl kvůli licenčním právům, jelikož Java je volně šiřitelná, ale JVM (Java Virtual Machine) ne. A také kvůli optimalizaci virtuálního stroje pro mobilní zařízení a poměru jejich výkonu a úspory energie.

Nejdůležitější vrstvou pro vývojáře je již zmíněný Application Framework, který umožňuje využívat všech služeb, hardwaru a funkcí telefonu v aplikacích.

Poslední vrstvu tvoří základní aplikace, které mohou být v zařízení již obsaženy, nebo se mohou doinstalovat z obchodu Google Play. Více informací o architektuře je možné dohledat v [5].

1.2 Vzhled a funkce

Jak již bylo zmíněno Android je postaven na jádře Linuxu, pod licencí open-source, díky které může být tento systém modifikován podle uživatele.

Zejména výrobci telefonů upravují vzhled a funkčnost systému tak, aby perfektně fungoval na jejich zařízeních a reprezentoval jejich společnost. Mezi tyto společnosti patří například HTC, Samsung, Sony Ericsson (dnes už jen Sony) na Obr. 2. Tyto úpravy nemají na běh systému žádný vliv, jedná se spíše o úpravy vzhledu nativních aplikací, případně různých ikonek v notifikační liště.

Obr. 2 Nadstavby Androidu (Samsung, HTC, Sony Ericsson)

Mnoho uživatelů, proto vyhledává alternativní verze Androidu, které si po provedení odemčení práv administrátora (roota) a následného přehrání bootloaderu mohou nahrát do svého zařízení. Hlavním důvodem je buď použití novější, neoficiální verze Androidu, protože výrobce už neposkytuje podporu pro dané zařízení. Případně uživatel chce urychlit zařízení tím, že je nahrán systém bez zmiňovaných nadstaveb od výrobců.

Hlavní funkce všech verzí systému Android jsou stejné a většinou typické pro každé mobilní zařízení. Jsou zde funkce pro volání a posílání SMS/MMS zpráv. Toto však od verze 3.0 (Honeycomb) není podmínkou, jelikož od této verze se systém začal používat i na tabletech, z nichž mnoho nemá 3G modul pro připojení do mobilní sítě. Protože Android je systém určený pro chytré mobilní telefony, tak může využít vše, co telefon nabízí, typicky je to fotoaparát, senzor světla, senzory pohybu, reproduktor, mikrofon, WIFI, GPS, 3G. Další funkce se mohou lišit jako například hardwarová tlačítka, čelní kamera.

Pro systém Android existuje mnoho aplikací, které tyto a další funkce využívají. Vzhledem k výkonu a funkcím, které dnešní zařízení poskytují, bylo překonvertováno pro Android několik her, které byly původně určeny pro platformu Windows.

2 Vývoj aplikací obecně

Počet aplikací pro systém Android, které jsou v oficiálním online obchodě, se pohybuje v řádech sta tisíců a to nejen proto, že je tato mobilní platforma velice oblíbená, ale také proto, že aplikace se vyvíjí v jazyce JAVA, který je jeden ze známých a jednodušších programovacích jazyků. Android aplikace dále využívají jazyk XML.

2.1 Android SDK

Android SDK je balík knihoven pro práci se všemi I/O periferiemi vyskytujícími se na Android zařízeních. Také jsou zde knihovny pro návrh grafického prostředí. Jde o knihovny pro programovací jazyk JAVA. SDK obsahuje i nástroje pro testování a ladění aplikací.

S Android SDK souvisí také API, které je neustále ve vývoji spolu se samotným systémem Android, v současné době je k dispozici API 17 (Android 4.2.2). Pokud se objeví nová verze API, programátor se nemusí obávat nefunkčnosti na novější verzi systému. Jen to, co v době vývoje aplikace bylo považováno za nejlepší metodu pro vyřešení určitého algoritmu, může být s novým API považováno za zastaralé. V tomto projektu byly postupně použity verze od API 15 po současné API 17.

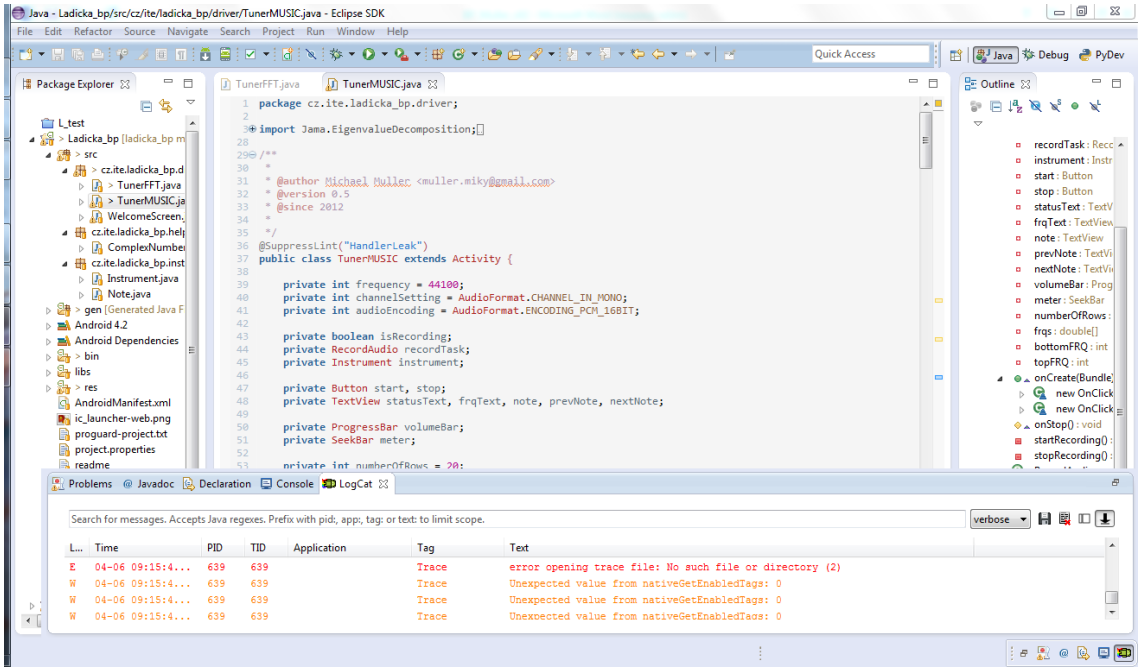
2.2 V čem vyvíjet?

Vyvíjet je možné v několika vývojových prostředích. Mezi ně patří Eclipse na Obr. 3, pro který je primárně vyvíjen Android SDK a všechny jeho nástroje. Dále lze vyvíjet v prostředí Aptana Studio. Aptana vychází z prostředí Eclipse, s tím rozdílem, že je primárně určena pro vývoj aplikací v Pythonu, ale po nainstalování všech potřebných pluginů a knihoven ji lze využívat i pro vývoj aplikací pro Android. Poslední z možných prostředí je IntelliJ IDEA.

V této práci bude zmiňováno prostředí Eclipse, ale to samé bude platit i pro Aptana Studio, jelikož prostředí jsou totožná až na pár výjimek.

Vyzkoušena byla obě prostředí, zpočátku Eclipse, který se zdál nevyhovující, díky svému pomalému běhu a neustálému neodpovídání programu. Proto bylo vyzkoušeno i prostředí Aptana Studio, jež se zdálo jako nejlepší volba díky své rychlosti. Ovšem nyní je na tom i prostředí Eclipse velmi dobře. Po posledních

aktualizacích je na tom v rychlosti komunikace s uživatelem a stabilnímu běhu srovnatelně s prostředím Aptana Studio.



Obr. 3 Prostředí Eclipse

2.3 ADT Plugin

Pro správnou funkci Android SDK ve vývojovém prostředí Eclipse nebo Aptana je potřeba nainstalovat ADT plugin, který obsahuje Android SDK Manager, pomocí kterého je možné aktualizovat SDK na nejnovější API Androidu. Plugin dále obsahuje Android Virtual Device Manager. Tento nástroj slouží pro správu všech virtuálních Android zařízení. Zařízení je možné přidávat, odebírat a upravovat. Při přidání či úpravě je možné zvolit z několika voleb, podle kterých se zařízení může chovat jako fyzické zařízení.

Pro vývoj je možné použít i fyzické Android zařízení. Předtím musí být splněno několik bodů:

- Musí být povoleno ladění v zařízení.
- Musí být nainstalovány ovladače zařízení.
- Musí být zařízení připojeno k počítači.

Plugin obsahuje i nástroje pro ladění a různé kontrolní zaznamenávání při ladění aplikace. Kontroluje správnou syntaxi a sémantiku kódu JAVA a XML souborů,

které Android využívá. Umožňuje uživateli i návrh GUI, který je doprovázen grafickým náhledem.

2.4 Základní komponenty aplikace

2.4.1 Aktivity

Jedná se o hlavní část programu, bez které by nemohla být aplikace spuštěna. Aktivita je to co uživatel po spuštění aplikace uvidí jako první. Slouží tedy hlavně pro komunikaci s uživatelem. Zpracovává to, co uživatel zadá, na co klikne. Nebo zobrazuje uživateli výsledek nějaké operace. Aktivity obsluhují i zbytek aplikace, mohou spouštět jiné aktivity nebo aplikace. Proto má každá z aktivit svůj životní cyklus, který vypadá následovně:

```
public class Activity extends ApplicationContext {
      protected void onCreate(Bundle savedInstanceState);
      protected void onStart();
      protected void onRestart();
      protected void onResume();
      protected void onPause();
      protected void onStop();
      protected void onDestroy();
  }
```

Aktivita je z pohledu programátora obyčejná třída v programovacím jazyce Java. Jsou zde již zmíněné metody, které se starají o životní cyklus aplikace. Mohou v ní být i metody vytvořené programátorem. V aktivitách mohou být umístěny i vnořené třídy, které nemohou být umístěny do samostatného souboru, jelikož spolupracující s proměnnými dané aktivity. Více je možné dohledat v [4].

2.4.2 View

View je návrh toho jak budou rozmístěny prvky, které aktivity poté zobrazí. View do sebe mohou být vkládány. Takže může být předem připraven layout pro pozadí s přechodem, které poté můžeme použít v jiném view.

V aplikacích pro Android jsou view zapisovány v jazyce XML, ten je svým zápisem tagů do špičatých závorek velice podobný HTML.

```
<View
    android:id="@+id/back"
    android:background="@drawable/background" />
<Button
    android:id="@+id/start"
    android:text="@string/start_button"
    android:textColor="@color/light_red"/>
```

Každá aktivita si vytvoří vlastní view (může i několik), jejichž data poté využívá. Když je použito implicitní nastavení při vytváření aplikace v Aptana studiu či Eclipse je view vytvořeno ve složce layout, což je jakýsi návrh.

V již zmiňovaných vývojových prostředích je k dispozici WYSIWYG editor XML souborů obsahující view. Nedoporučuje se ovšem vytvářet view přímo v tomto editoru, přestože vytvořené view může vypadat skvěle, ale pokud například bude telefon překlopen na bok, aby se aplikace zobrazila v režimu landscape, tak se celý návrh rozbije. Z tohoto důvodu je lepší, aby bylo view napsáno manuálně v XML, který není v tomto případě nijak složitý.

2.4.3 Identifikátory

Identifikátory jsou v Android aplikacích statickými vlastnostmi třídy , která je nazvána *R*. To je automaticky generovaná třída na základě obsahu složky `res`. Jako příklad si můžeme uvést identifikátor `R.layout.activity_ladicka`, který nás odkazuje na soubor `activity_ladicka.xml` ve složce `/res/layout`. Takto mohou být použity identifikátory i pro konkrétní prvky v layoutu, pokud tedy bude potřeba změnit text v TextView tak bude použito `R.id.myTextView`.

2.4.4 Intenty

Intenty jsou základním komunikačním prvkem mezi aplikacemi v Androidu. Intent obsahuje popis nějakého záměru, případně nějaké argumenty. Starají se tedy o předávání dat mezi aktivitami jedné aplikace, nebo mohou předávat data i dalším aplikacím. Pokud je v telefonu více aplikací, které mohou daný požadavek obsloužit, je uživateli nabídnuto vybrání jedné z nich. Vybraná aplikace pak požadavek zpracuje. Příklad vytvoření intentu:

```
Intent intent = new Intent(this, TunerFFT.class);
intent.putExtra("id", volba);
startActivity(intent);
```

První řádek je vytvoření instance třídy Intent, kde první parametr je upřesnění, které aplikace má případně systém nabídnout (v tomto případě `this`, jelikož spouštíme konkrétní aktivitu), druhý parametr je konkrétní aktivita, která se má spustit. Na druhém řádku, předáváme extra data, která chceme předat navíc. A třetí řádek spustí danou Aktivitu. V následujícím kódu je možné vidět rozdíly od spouštění aktivit a jiných aplikací.

```
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

Zde se vyhledají všechny aplikace, které jsou schopny vytočit telefonní číslo.

2.4.5 Resources

Resources jsou jakési zdroje nebo suroviny, které jsou využity v kódu aplikace. Za suroviny jsou považovány například řetězce, barvy nebo styly. Mohou být namíchány všechny v jednom souboru, ale všeobecně se tato data separují podle druhu, každá do jednoho souboru, tzn. `color.xml, strings.xml.`

Každý soubor obsahuje element `<resources>`, který následně obsahuje potomky jako `<string>`, `<color>` apod. Tyto tagy jsou většinou párové a hodnotu mají mezi sebou. Podle těchto potomků se vytvoří identifikátor např.: `R.string.` Každý z těchto potomků má atribut `name`, který určuje poslední část identifikátoru př.: `R.string.nadpis`.

```
<resources>
    <color name="light_red">#FF4444</color>
    <color name="dark_red">#CC0000</color>
    <color name="light_green">#99CC00</color>
    <color name="dark_green">#669900</color>
    <color name="light_yellow">#FFBB33</color>
    <color name="dark_yellow">#FF8800</color>
    <color name="light_blue">#33B5E5</color>
    <color name="dark_blue">#0099CC</color>
    <color name="light_purple">#AA66CC</color>
    <color name="dark_purple">#9933CC</color>
</resources>
```

2.5 Možnosti sledování zvukového signálu v OS Android

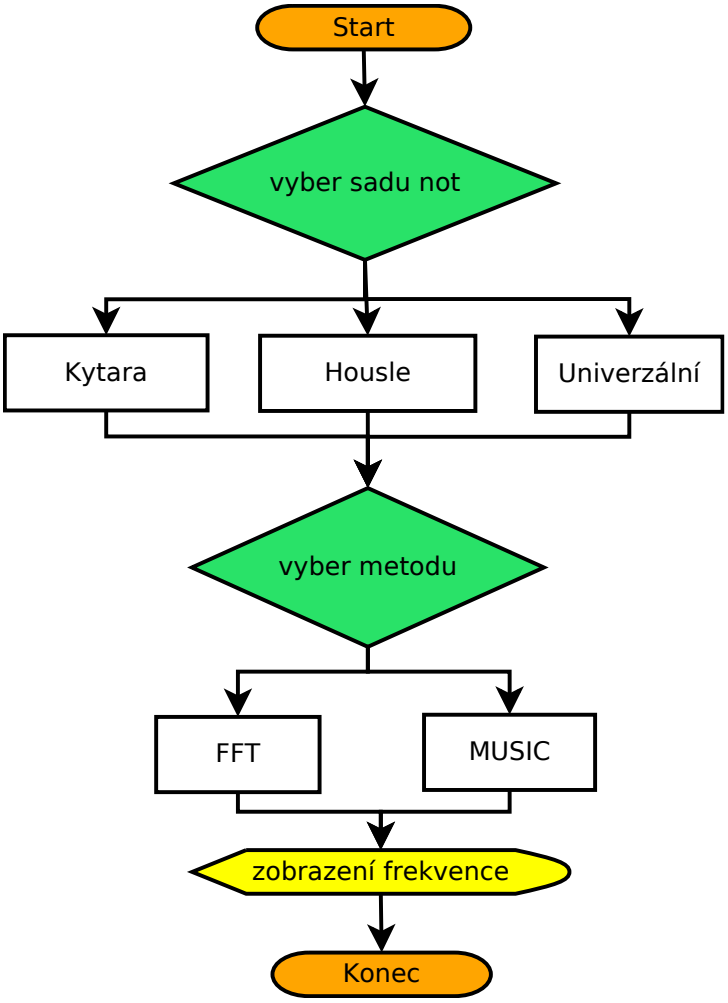
Zaznamenávání zvukového signálu je možné v OS Android několika způsoby. První způsob je zavolání již předinstalované aplikace pro nahrávání zvuku pomocí intentu. Ten spustí aplikaci pro nahrávání zvuku, pomocí aplikace je zaznamenán signál, který je poté uložen do úložiště zařízení. Tento způsob není vhodný pro sledování zvukového signálu v reálném čase, jelikož signál musí být nejprve zaznamenán a uložen, teprve poté je možné se souborem pracovat. S tím souvisí další problém, a to ten, že by bylo nutné soubor zpracovat tak, abychom dostali přesná data signálu. A jelikož formáty pro ukládání, které výchozí aplikace pro zaznamenávání zvuku využívá, obsahují jistou kompresi, je tento způsob sledování zvukového signálu značně neefektivní.

Další možností je vlastní naprogramování části zaznamenávání zvuku s využitím třídy MediaRecorder, a to tak, že je vytvořena instance třídy MediaRecorder, které jsou nastaveny hlavní parametry jako zdroj signálu, formát pro ukládání, dekodér a soubor do kterého se výsledný zaznamenaný zvuk zapíše. Problém této možnosti je stejný jako v předchozím případě. A to že nelze zvuk zpracovávat ihned při zaznamenávání, ale až po tom co je uložen do souboru v nějakém formátu s kompresí.

Poslední možností je využití třídy AudioRecord. Třída AudioRecord umožňuje zaznamenávat zvuk jako „čistá data“, ta je potom možné dále v reálném čase zpracovávat. Tato třída byla použita i v této aplikaci a její bližší popis bude uveden v následující kapitole.

3 Aplikace Inteligentní ladička

Úkolem aplikace je pomoci uživateli naladit téměř jakýkoliv hudební nástroj. Aplikace sleduje zvukový signál, který je zaznamenáván pomocí mikrofonu zařízení s OS Android. Následně použije jednu z metod pro detekování fundamentální frekvence, v aplikaci jsou použity dvě. První z nich je metoda Rychlé Fourierovy transformace (FFT – Fast Fourier Transform) a druhá je metoda MUSIC (Multiple Signal Classification). Po vypočítání frekvence se pokusí nalézt odpovídající tón a jeho příslušnou notu. Tu zobrazí uživateli na display. Pro zobrazení bylo vytvořeno jednoduché grafické rozhraní. Detailněji budou metody, jejich implementace a grafické rozhraní popsány níže. Obecný vývojový diagram aplikace je na Obr. 4.



Obr. 4 Obecný vývojový diagram funkce aplikace

3.1 Struktura aplikace

Hlavní části aplikace, tedy programový kód, byl rozdělen do tří balíčků. V prvním balíčku s názvem driver jsou umístěny hlavní aktivity, které řídí celou aplikaci. Je zde úvodní aktivita a aktivity, které zpracovávají zaznamenaný zvukový signál. V balíčku nazvaném helpers, jsou umístěny pomocné třídy pro vyhodnocení noty, a počítání s komplexními čísly. Posledním balíčkem je balíček dialogs. Zde jsou umístěny dialogy, které se zobrazují při různých akcích v aplikaci. Struktura aplikace je tedy následující:

- Driver – hlavní aktivity aplikace
  - TunerFFT
  - TunerMUSIC
  - WelcomeScreen
- Dialogs – dialogy aplikace
  - AboutDialog
  - ChangeNotesDialog
- Helpers – pomocné třídy
  - Note
  - Instrument
  - ComplexNumber

3.2 Použití AudioRecord

Tato třída je už součástí programovacího Jazyka JAVA, není to tedy něco, co nám přináší nově Android i když se v něm dá tato třída využít. AudioRecord slouží v podstatě jako správce zvukových zdrojů. Objekt této třídy lze použít několika způsoby v závislosti na tom, v jakém datovém typu se mají data zaznamenat. Podrobnosti k třídě AudioRecord naleznete zde [6].

Možnosti použití objektu třídy AudioRecord:

- `read(byte[], int, int);`
- `read(short[], int, int);`
- `read(ByteBuffer, int);`

Po vytvoření objektu se inicializuje vyrovnávací paměť, do které se potom průběžně zaznamenávaná data ukládají. Velikost této paměti určuje, jakou délku dat může AudioRecord zaznamenat předtím, než budou data dále zpracována. Proto jsou data z hardware zaznamenávána v blocích.

Při využití třídy AudioRecord v aplikaci, byla zvolena možnost zpracovávat data asynchronně na pozadí. Proto musí být vytvořena vlastní vnořená třída v každé aktivitě, která bude dědit od třídy AsyncTask. Pak je v metodě `doInBackground` vytvořena instance třídy AudioRecord, té nastavíme zdroj signálu, vzorkovací frekvenci, počet kanálů, kódování signálu a nakonec velikost zásobníku. Pokud je toto nastaveno, tak stačí spustit čtení dat z mikrofonu a následně je možné vždy načtený zásobník dat zpracovat.

3.3 Měření frekvence pomocí FFT

První částí aplikace je měření fundamentální frekvence pomocí metody Rychlé Fourierovy transformace.

3.3.1 Rychlá Fourierova transformace (FFT)

FFT vychází z Diskrétní Fourierovy transformace, jen je výpočet optimalizován pro rychlost. Verzí tohoto algoritmu existuje více, v aplikaci je však použit algoritmus Radix-2. Algoritmus funguje na principu postupného dělení signálu. Tím se zrychlí výpočet a složitost původního algoritmu DFT klesne. Tento algoritmus funguje jen tehdy, je-li velikost zpracovávaného signálu mocninou dvojky.

V aplikaci je počítáno s N hodnotami. Těchto hodnot je v tomto případě 4096 a jsou zaznamenávány mikrofonem mobilního zařízení každých 92ms. Hodnoty jsou v rozmezí od - 32768 do +32767. Počet těchto hodnot nám určí frekvenční rozlišení. To je určeno vzorcem $k \cdot Fs/N$, kde k je index vzorku, Fs je vzorkovací frekvence a N je velikost bloku dat. Frekvenční rozlišení je tedy závislé na N, které určuje jeho velikost. Na základě vzorkovacího teorému pro $k > N/2$ už nezískáme žádnou novou informaci. Pro příklad je dána vzorkovací frekvence 16kHz a signál o frekvenci 440Hz. Tuto frekvenci je poté možné určit jen jako 441,4Hz.

Definiční vztah Fourierovy transformace:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-jft}dt \tag{1}$$

Definiční vztah pro výpočet Diskrétní Fourierovy transformace:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi nk}{N}} \qquad k = 0 \dots N-1 \tag{2}$$

3.3.2 Implementace algoritmu v Aplikaci

V aplikaci byla pro výpočet rychlé Fourierovy transformace použita knihovna libGDX, která je pod licencí Apache Licence 2.0. Pro použití funkcí knihovny musela být nejprve vytvořena komplexní čísla ze vstupního signálu, ve kterém jsou čísla od -32768 do +32767. Tato komplexní čísla byla vytvořena jako dvě pole typu float.

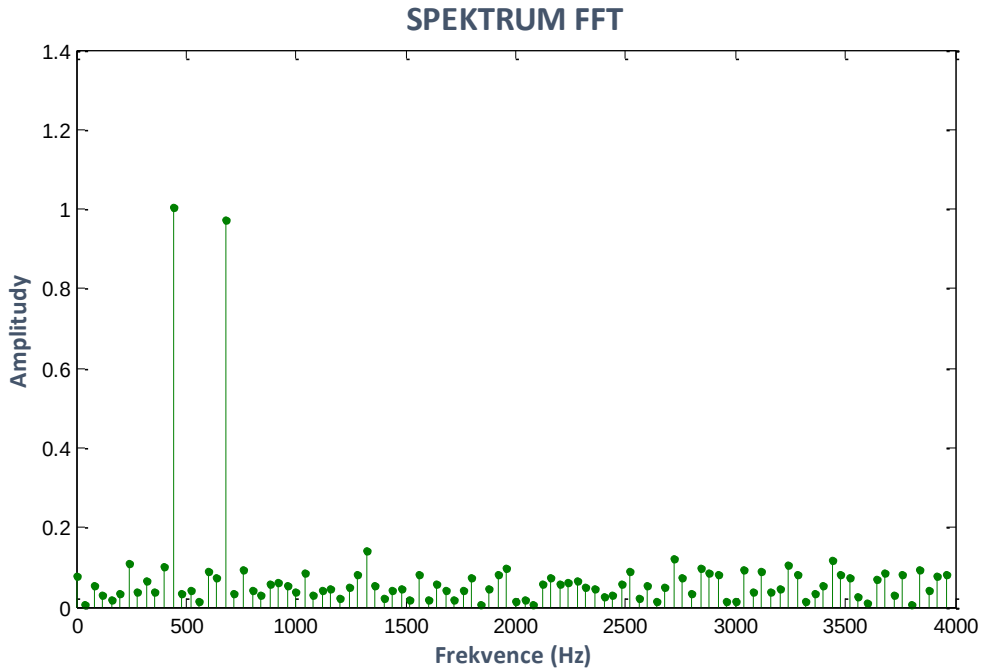
```
final float[] realBuf = new float[bufferSize];
final float[] imagBuf = new float[bufferSize];
```

Tato pole mají velikost stejnou jako zásobník třídy AudioRecord. Pole reálných částí je naplněno přímo změřenými daty, které obsahují jen reálnou část a pole imaginárních částí se naplňovat nemusí, protože ve výchozím stavu jsou zde nuly. Následně je zavolána metoda pro výpočet frekvence. Zde je vytvořena instance třídy FFT, které se předá parametr o velikosti signálu a vzorkovací frekvenci.

```
FFT f = new FFT(bufferSize, 44100);
f.forward(re,im);
float[] real = f.getRealPart();
float[] imag = f.getImaginaryPart();
```

Kód ukazuje, jak již bylo zmíněno vytvoření instance, dále je použití metody forward, která provede klasickou FFT, jejíž parametry jsou výše zmíněná pole. Nakonec musíme získat komplexní čísla, opět ve dvou polích a to pomocí metody `getRealPart` a `getImaginaryPart`. Nyní je možné spočítat amplitudy a vytvořit tak spektrum. V tomto spektru je nakonec nalezena maximální hodnota, jejíž index je hledaná frekvence. Nalezení a zobrazení noty odpovídající frekvenci je popsáno v kapitole 3.4.6.

Výsledné spektrum vytvořené pomocí Rychlé Fourierovy transformace pro vygenerovaný signál s hlavními frekvencemi 400Hz a 640Hz může vypadat jako na Obr. 5.



Obr. 5 Graf spektra FFT

3.4 Měření frekvence pomocí MUSIC

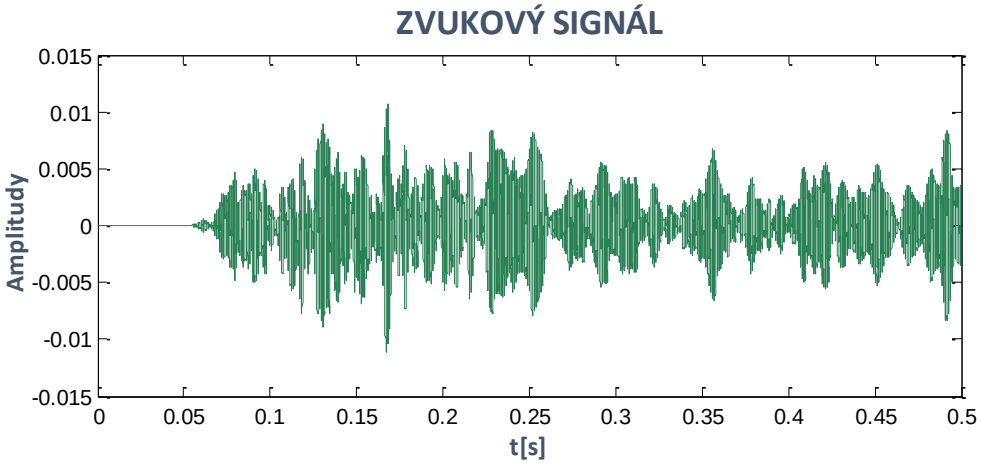
Druhou částí bylo vyhodnocování fundamentální frekvence pomocí metody zvané MUSIC (Multiple Signal Classification).

3.4.1 Multiple Signal Classification (MUSIC)

Metoda MUSIC je odvozena na základě předpokladu, že signál je součtem komplexních exponenciel. Předpoklad toho jak signál vypadá, je tedy následující:

$$x[t] = \sum_{i=1}^{p} a_i e^{j(2\pi f_i t + \phi_i)}, \tag{3}$$

kde f_i je frekvence, a_i je amplituda a ϕ_i je fáze i-té části signálu. Pro příklad zaznamenaný signál může vypadat jako na Obr. 6.



Obr. 6 Graf zvukového signálu

Následně se musí vytvořit ze zaznamenaného signálu matice X, která bude v následujícím tvaru:

$$X = \begin{bmatrix} x[t] \\ x[t+1] \\ \vdots \\ x[t+M-1] \end{bmatrix}. \tag{4}$$

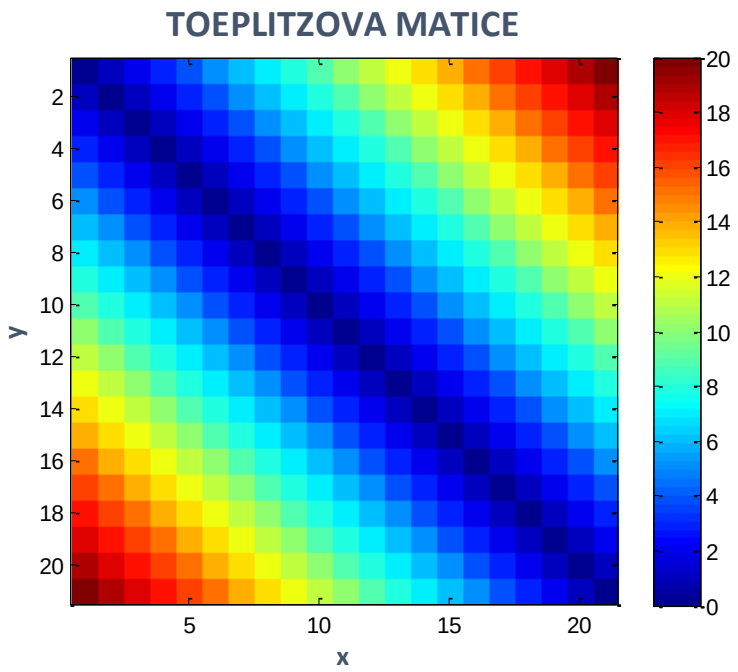
Z této matice je nutné vytvořit autokorelační matici R_{xx} o velikosti $M \times M$ (M je velikost signálu), bude tedy vypadat následovně:

$$R_{xx} = E[XX^T], \tag{5}$$

kde X je matice vytvořená ze zaznamenaného signálu, X^T je matice X transponovaná, E je operátor střední hodnoty, který je v praxi nahrazen průměrováním. Ve výsledné matici, se musí pro každý prvek matice vypočítat střední hodnota. Ta je v tomto případě počítána vydělením každého prvku matice XX^T velikostí signálu. Tato matice má následující tvar:

$$R_{xx} = \begin{bmatrix} r_{xx}[0] & r_{xx}[1] & \dots & r_{xx}[M-1] \\ r_{xx}[1] & r_{xx}[0] & \dots & r_{xx}[M-2] \\ \vdots & & & \\ r_{xx}[M-1] & r_{xx}[M-2] & \cdots & r_{xx}[0] \end{bmatrix}, \tag{6}$$

takovou matici lze nazvat maticí Toeplizovou. Toeplitzova matice má na každé diagonále konstantní hodnoty. Grafický náhled takové matice je znázorněn na Obr. 7. Jednotlivé odstíny barev znázorňují vždy jednu diagonálu matice.



Obr. 7 Grafické znázornění Toeplizovy matice

Nyní je možné z autokorelační matice vypočítat vlastní vektory a vlastní čísla. Vlastní čísla jsou seřazena od největšího po nejmenší. Každému vlastnímu číslu pak odpovídá jeden vlastní vektor.

Z vlastních čísel musí být zjištěno, kolik sinusoid signál obsahuje. Tyto sinusoidy nám určí, jaká část zaznamenaného signálu nás zajímá. Počet sinusoid bude určovat hodnota nazvaná p, tuto hodnotu není možné určit jako hodnost matice R_{xx}, protože tato matice má plnou hodnost. Hodnost matice by odpovídala hodnotě p jen v případě, že by signál obsahoval jen hledané exponenciely. Předpoklad je ale takový, že signál obsahuje šum, proto musí být nalezeny největší vlastní vektory. Hodnost p není možné nastavit na pevno, protože vlastní čísla jsou závislá na hlasitosti signálu – čím vyšší hlasitost, tím vyšší hodnoty vlastních čísel. Hodnota p je tedy zjištěna spočítáním střední hodnoty vlastních čísel. Počet vlastních čísel větších než střední hodnota je hledané p.

Hodnota p rozdělí zaznamenaný signál na podprostor šumu a podprostor signálu. Podprostor signálu může být dále zpracován metodou MUSIC. Ta se opírá o fakt, že podprostor signálu je kolmý na podprostor šumu. Mějme vektor:

$$s^T(f) = [1 \quad e^{j2\pi f} \quad e^{j2\pi 2f} \quad \dots \quad e^{j2\pi(M-1)f}], \tag{7}$$

když $f = f_i$ (jedna z frekvencí, které jsou obsaženy v signálu), potom pro jakýkoliv vektor $\boldsymbol{x}$ v podprostoru šumu je součin vektoru s a vektoru $\boldsymbol{x}$ roven nule, protože jsou na sebe kolmé.

$$s(f)\boldsymbol{x} = 0 \tag{8}$$

Pokud tedy máme,

$$M(f) = \sum_{k=p+1}^{M} |s(f)\boldsymbol{u}_k|^2 \tag{9}$$

kde s je výše zmíněný vektor (7) a $\boldsymbol{u}_k$ jsou vlastní vektory matice R_{xx}. Když se tedy $f = f_i$ potom se $M(f)$ musí rovnat nule a $1/M(f)$ musí být nekonečno. Potom graf funkce $1/M(f)$ by měl mít nejvyšší vrchol v $f = f_i$ pro každou vstupní frekvenci.

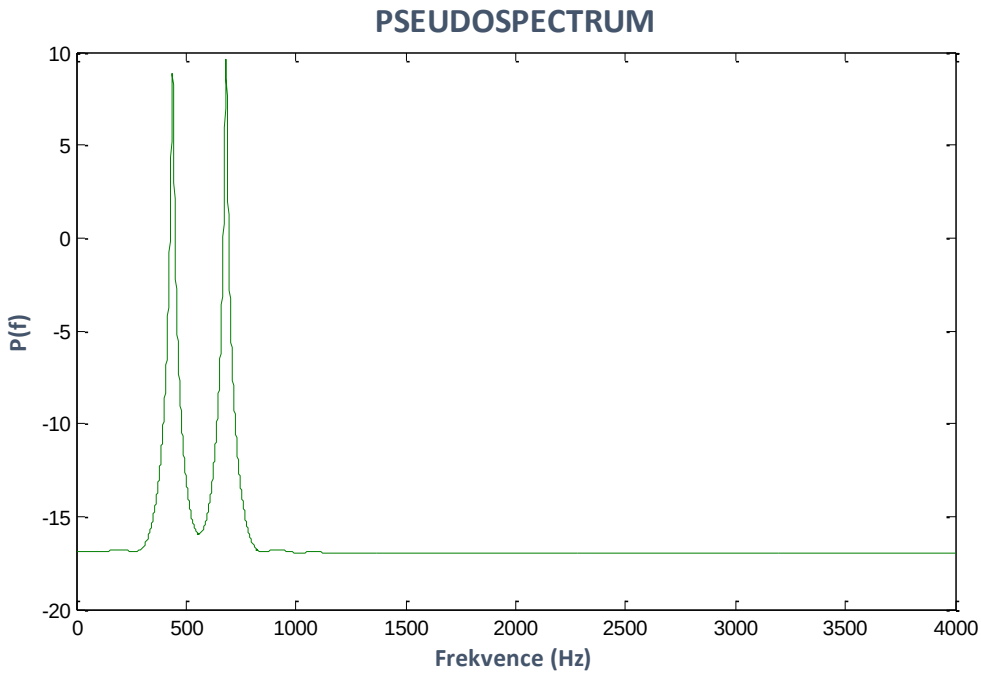
Funkce

$$P(f) = \frac{1}{\sum_{k=p+1}^{M} |s(f)\boldsymbol{u}_k|^2} \tag{10}$$

je někdy označována jako MUSIC spektrum nebo také pseudospektrum. Více je možné dozvědět se v [7].

Metoda MUSIC může tedy být přesnější než FFT, jelikož FFT má rozlišení omezené délkou bloku dat. Metoda MUSIC může být přesnější i díky tomu, že identifikuje podprostor šumu a signálu. Kdežto FFT vyhodnocuje frekvenci i ze zašuměného signálu. Použití metody MUSIC je založeno na předpokladu, že v signálu je méně sinusovek *p.* Takový předpoklad, ale není obecně platný u reálných zvuků. Aplikace je ale určena především pro ladění nástrojů, proto je možné předpokládat, že nástroj hraje jen jeden tón a ten obsahuje jen hlavní frekvenci a několik harmonických.

Výsledné spektrum vytvořené pomocí metody MUSIC pro vygenerovaný signál s hlavními frekvencemi 400Hz a 640Hz může vypadat tak, jak je znázorněno na Obr. 8.



Obr. 8 Graf pseudospektra

3.4.2

3.4.3 Vyhledání maxima – Newtonova optimalizační metoda

Pro lepší výpočet minima v pseudospektru metody MUSIC, byla zvolena Newtonova optimalizační metoda. Tato metoda spočívá v sestrojení tečny v bodě $[x_k, f'(x_k)]^T$ funkce $f'(x)$. Potom bod x_{k+1}je průsečík s osou x a je to tedy $k+1$-ní iterace optimálního bodu x^*. Funkce je znázorněna prvními třemi členy Taylorova rozvoje

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 \tag{11}$$

Derivováním tohoto rozvoje a vyřešením $f(x_{i+1}) = 0$ vznikne:

$$f'(x_i) + f''(x_i)(x_{i+1} - x_i) = \ 0, \tag{12}$$

z tohoto potom plyne iterační vztah pro x_{i+1}, který má následující tvar:

$$x_{i+1} = \ x_i - \frac{f'(x_i)}{f''(x_i)}. \tag{13}$$

Newtonova metoda vyžaduje existenci druhé derivace. Více o této metodě je možné dohledat v Diplomové práci s názvem Numerické metody optimalizace od Miloše Jurka [8].

Tato metoda byla použita jako součást metody MUSIC, musely proto být vytvořeny derivace její funkce. Původní funkce metody MUSIC vypadá takto,

$$P(f) = \frac{1}{\sum_{k=p+1}^{M} |s(f)\boldsymbol{u}_k|^2} \tag{14}$$

pro derivování je použit pouze jmenovatel, nejprve je nutné, aby byl upraven následovně,

$$\sum_{k=p+1}^{M} \left(\sum_{l=0}^{M-1} e^{-j2\pi Fl}\, u_{kl} \right) \left(\sum_{m=0}^{M-1} e^{j2\pi Fm} \bar{u}_{km} \right) \tag{15}$$

toto může být upraveno následovně,

$$\sum_{k=p+1}^{M} \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} e^{j2\pi F(m-l)}\, u_{kl} \bar{u}_{km} = f(F) \tag{16}$$

Část $e^{j2\pi F(m-l)}$ může být zapsána goniometricky takto:

$$\cos\big(j2\pi F(m-l)\big) + j sin\big(j2\pi F(m-l)\big), \tag{17}$$

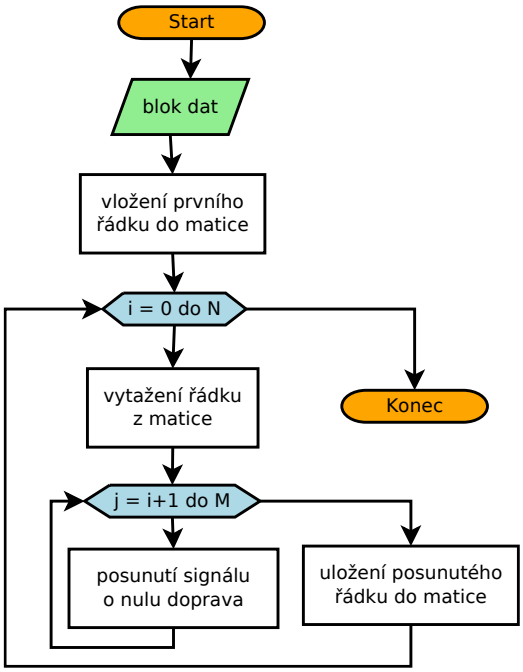
imaginární část může být vynechána, jelikož výsledná hodnota musí být reálná. Tím je funkce zbavena komplexní jednotky a výpočet se zjednodušuje. Nyní musí být vypočítána první a druhá derivace. Jejich tvar je na rovnicích (18) a (19).

$$\sum_{k=p+1}^{M} \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} -2\pi(\mathrm{m}-\mathrm{l})\sin(2\pi F(m-l))\, u_{kl} \bar{u}_{km} = f'(F) \tag{18}$$

$$\sum_{k=p+1}^{M} \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} -(2\pi)^2 (m-l)^2 \cos(2\pi F(m-l))\, u_{kl} \bar{u}_{km} = f''(F) \tag{19}$$

3.4.4 Implementace algoritmu v aplikaci

Jelikož metoda MUSIC počítá s maticemi, byla pro tyto operace vybrána knihovna Jama. Jedná se o knihovnu lineární algebry pro jazyk Java. Využita je hlavně pro výpočet vlastních vektorů a vlastních čísel.



Obr. 9 Vývojový diagram vytvoření matice ze signálu

Diagram na Obr. 9 popisuje vytvoření matice, která nese název *X,* ze zaznamenaného signálu. Před vstoupením do cyklu se vytvoří první řádek matice, který má stejnou podobu jako samotný signál. První cyklus se provádí N-krát, N je počet řádků, který může být menší než velikost signálu. Druhý cyklus je prováděn M-krát, kdy M je délka řádku matice. V cyklu je vyjmut řádek matice s indexem *i*. Tento řádek je posunut vždy o jednu nulu doprava a uložen jako další řádek do matice.

Když je vytvořena matice *X*, vytvoří se transponovaná matice *XT.* Pro vytvoření této matice je třeba vytvořit novou instanci třídy `Matrix` z knihovny Jama a zavolat metodu `transpose()` na již vytvořenou matici *X*.

```
Matrix XT = X.transpose();
```

Z těchto dvou matic musí být vytvořena autokorelační matice R_{xx}. To je provedeno součinem matic *X* a *XT*, tento součin je potom vydělen velikostí zaznamenaného signálu. Na toto je použita metoda `times()` a je použita následovně:

```
Matrix Rxx = (X.times(XT)).times(1.0/signal.length);
```

Následně je potřeba vytvořit instanci třídy EigenvalueDecomposition, která obsahuje metody pro výpočet vlastních vektorů a vlastních čísel dané matice. Konstruktor této třídy přebírá právě matici, u které se toto spočítá. Pro získání

vlastních vektorů a čísel je nutné zavolat metody `getV()` a `getD()`. Které vracejí matice.

```
Matrix D = ed.getD();
Matrix V = ed.getV();
```

Vlastní čísla jsou nyní uložena v matici, jsou ovšem jen na diagonále, zbytek matice je naplněn nulami. Proto stačí, aby byla diagonála uložena do nového jednorozměrného pole. Pokud jsou čísla uložena ve vlastním poli, je možné spočítat hodnotu *p,* podle které bude vypočítána, která část signálu je šum a která je signál.

Tabulka 1 Rozdělení signálu podle hodnoty "p" vypočítané z vlastních čísel

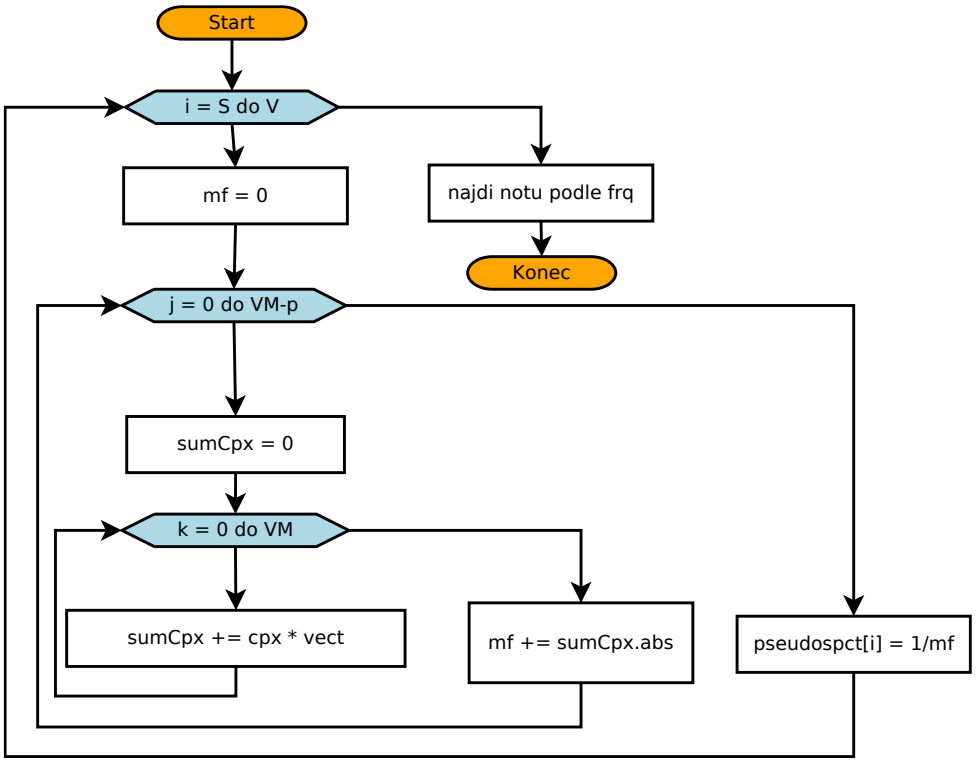
0,09232	0,09592	0,09635	0.09891	...	8.0690	8.2465	11.4031	11.9095
← šum →				*p* = 4	← signál →			

Nyní je signál zpracován a připraven tak, aby mohl být použit algoritmus pro výpočet pseudospektra.

Do algoritmu je zároveň zakomponována Newtonova optimalizační metoda, která je popsána v kapitole 3.4.2. Algoritmus musí být proveden třemi cykly, jelikož programovací jazyk Java neumí násobit vektory tak snadno, jako například Matlab.

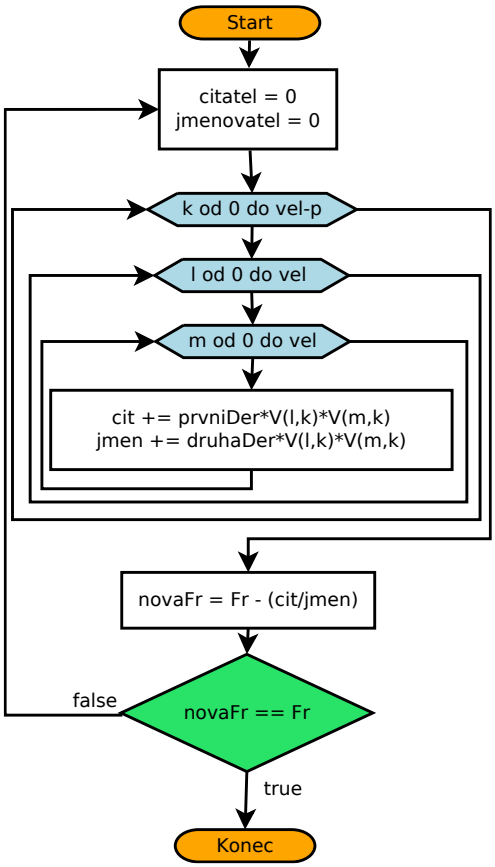
Původní algoritmus byl navržen tak, že počítal metodu MUSIC s komplexními čísly. Vývojový diagram takového algoritmu je na Obr. 10. V tomto algoritmu se nejprve vytvoří suma komplexních čísel, která jsou násobena vlastními vektory. Tato suma se provede několikrát, jelikož je umístěna v další sumě, kde se pouze sčítají výsledky první vnitřní sumy. A v sumě, která je vnější se vždy provede převrácená hodnota výsledku, předchozích sum a uloží se do pole (pseudospektra).

Hodnoty v poli jsou zde jen převrácené hodnoty součtů sum, proto zde stačí najít minimum těchto hodnot. Minimum je hledáno pomocí jednoduchého porovnání hodnot s ostatními. Index nejmenší hodnoty je hledaná frekvence.



Obr. 10 Vývojový diagram algoritmu výpočtu pseudospektra

Protože byla zvolena varianta využití Newtonovy metody, byl tento algoritmus upraven. Algoritmus znázorňuje vývojový diagram na Obr. 11. Protože s Newtonovou metodou, bylo možné zbavit se komplexních čísel. Tato třída není v aplikaci dále používána. Algoritmus obsahuje tři cykly, v každém kroku se pouze sčítají součiny prvních a druhých derivací s vlastními vektory. Po výpočtu těchto sum se provede výpočet další iterace pomocí Newtonovy metody. Pokud se nerovná frekvenci, která byla zadána jako nejbližší nalezená, je tato metoda spuštěna rekurzivně znovu. V opačném případě je nalezená frekvence vrácena z algoritmu.



Obr. 11 Vývojový diagram metody MUSIC s Newtonovou metodou

3.4.5 Urychlení výpočtu

Metoda MUSIC má poměrně vysokou složitost a tím pádem i výpočetní náročnost. Jelikož aplikace Inteligentní ladička je určena pro chytré mobilní telefony, měla by být proto co nejméně náročná na výpočet, i když už dnes mají mobilní zařízení vysoký výpočetní výkon. Proto byla zvolena úprava na zúžení intervalu prohledávaných frekvencí za pomoci FFT.

První úprava zmenší interval prohledávaných frekvencí tak, že se nejprve v daném bloku dat nalezne frekvence pomocí rychlé Fourierovy transformace, aby mohla být určena počáteční frekvence pro využití Newtonovy metody, která je aplikována na metodu MUSIC.

3.4.6 Nalezení tónu

Pro vyhledání odpovídajícího tónu dané frekvenci, byly vytvořeny dvě třídy jedna s názvem Notes a druhá s názvem Instrument. Ve třídě Notes je pouze konstruktor a odpovídající metody `get()` a `set()`. Slouží pouze k identifikaci jednotlivých tónů. Tomu odpovídají i parametry konstruktoru a těmi jsou: název,

frekvence, předchozí tón a následující tón. Druhá třída Instrument, reprezentuje sadu not, pro daný hudební nástroj, kromě konstruktoru, je zde metoda pro inicializaci hudebního nástroje a metoda pro vyhledání odpovídající noty.

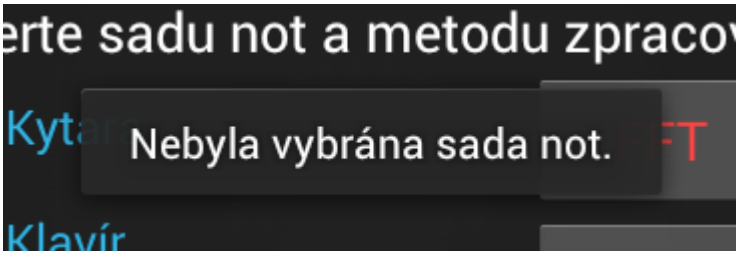
Při vytvoření instance třídy Instrument konstruktoru stačí pouze parametr `id`, kterým se specifikuje, jaký nástroj se má inicializovat. O inicializaci se už postará metoda `initialize()`. V aplikaci jsou nyní pouze tři sady not (kytara, housle, univerzální). Metoda pro vyhledání noty postupně vytváří intervaly mezi jednotlivými tóny. Pokud se hledaná frekvence vyskytuje v některém z intervalu, metoda vrátí název daného tónu. Pokud by byla mikrofonem zaznamenána frekvence vyšší než je rozsah vybrané sady not, metoda vrátí notu nazvanou nulová nota.

3.5 Grafické uživatelské prostředí

Grafické uživatelské prostředí bylo navrženo v jednoduchém stylu s využitím několika speciálních funkcí, které nabízí nejnovější verze systému Android.

3.5.1 Úvodní obrazovka

Na úvodní obrazovce je uživateli zobrazen jen uvítací text a možné volby ze sad tónů, které může použít v některé z metod. Tyto sady tónů je možné měnit i v průběhu ladění výběrem položky pro změnu sady v menu. Dále zde jsou umístěna dvě tlačítka, která po vybrání ze sady tónů, přepnou na danou metodu. Pokud se uživatel pokusí vybrat metodu bez vybrání sady tónů, je zobrazeno upozornění na tuto skutečnost jako na Obr. 12.



Obr. 12 Upozornění

Pro zobrazování takovýchto jednoduchých upozornění byla použita třída Toast. Po vytvoření její instance s daným textem stačí zavolat metodu `show()`.

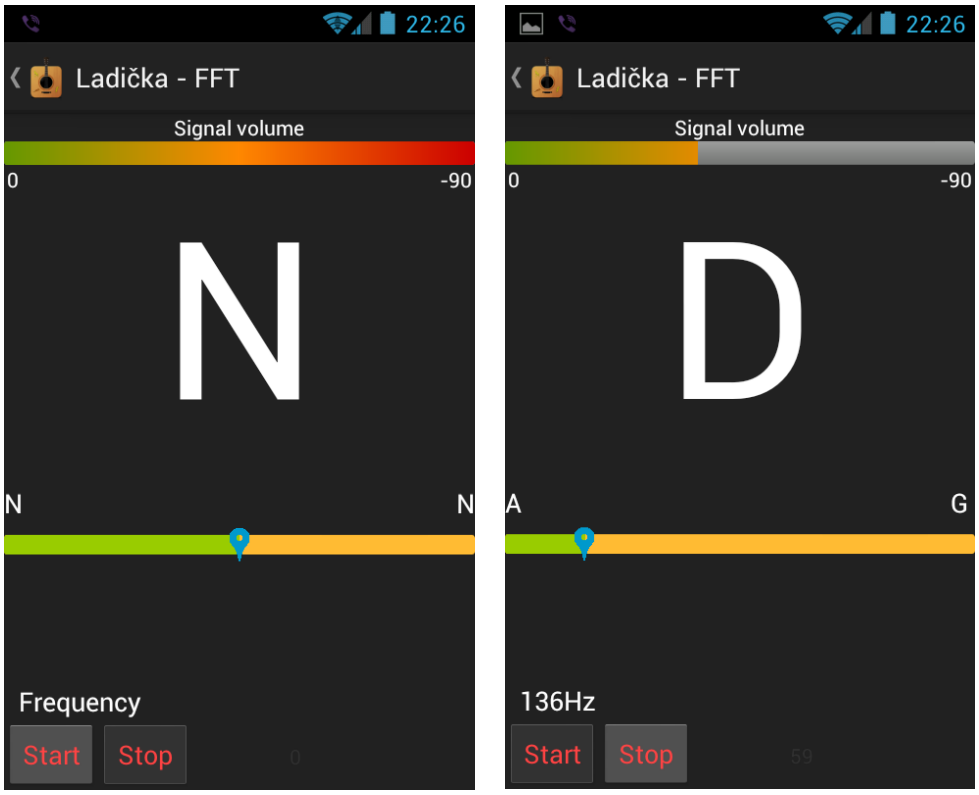
```
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Pro přenesení id vybrané sady not do aktivity vybrané metody počítání fundamentální frekvence, musel být vytvořen jednoduchý intent pro přenos celého čísla. Pokud uživatel na úvodní obrazovce, stiskne tlačítko menu na svém zařízení, je mu zobrazena pouze možnost zobrazení informací o aplikaci.

Obr. 13 Úvodní obrazovka

3.5.2 Zobrazení tónu

Zobrazování frekvence je řešeno pro obě metody výpočtu fundamentální frekvence stejný. Proto každá z těchto aktivit obsahuje v horní části progressbar, který slouží jako ukazatel úrovně hlasitosti snímaného zvukového signálu. Ve střední části se nachází ukazatel zjištěného tónu. Pod tímto ukazatelem se nachází seekbar, který je zde realizován jako ukazatel přeladění nebo podladění daného tónu. Po stranách jsou ukazatele pro předcházející a následující tón. V levé spodní části nalezneme tlačítka na spuštění a zastavení snímání zvukového signálu s ukazatelem naměřené frekvence viz Obr. 14.



Obr. 14 Vlevo výchozí stav, vpravo stav při měření

Jelikož měření probíhá na pozadí aplikace, musel být pro zobrazení nalezeného, předchozího a následujícího tónu vytvořen handler, který předá data z vlákna na pozadí, do hlavního vlákna kde se nachází i prvky grafického rozhraní. Pro použití handler, je nutné vytvořit instanci třídy `Handler` v níž je vytvořena metoda `handleMessage` následovně:

```
final Handler handler = new Handler(){
        @Override
          public void handleMessage(Message msg) {
            if(msg.what==1){
               Note n = (Note)msg.obj;
               note.setText(n.toString());
               //frqText.setText("" + valueFRQ + "Hz");
               nextNote.setText(n.getNext().getName()+" ");
               prevNote.setText(n.getPrev().getName()+" ");
               frqText.setText(msg.arg1+"Hz");
               ZobrazNaMeteru(n, msg.arg1);
            }
            super.handleMessage(msg);
          }
 };
```

Tato metoda je pak zavolána v metodě `spocitejFrekvenci()`, která počítá frekvenci a vloží do ní potřebná data, která jsou poté zobrazena.

3.5.3 Ukazatel vzdálenosti od nejbližšího tónu

Ukazatel slouží k indikaci podladění či přeladění tónu. Pro tento účel byla vybrána komponenta seekbar a vytvořena jediná metoda, která se stará o správné přepočítání posunu frekvence na procenta. Nejprve musí být vytvořeny intervaly mezi předchozím a následujícím tónem. Střed tohoto intervalu je nalezený tón. Vytvoření intervalu znázorňuje tento kód:

```
dolni = (nota.getFrq() + nota.getPrev().getFrq())/2;
horni = (nota.getFrq() + nota.getNext().getFrq())/2;
```

jedná se vždy o polovinu součtu nalezeného tónu s předchozím nebo následujícím tónem a vydělením dvěma pro nalezení středu.

Když je vytvořen interval je nutné spočítat procentuální vyjádření posunutí od středu takto:

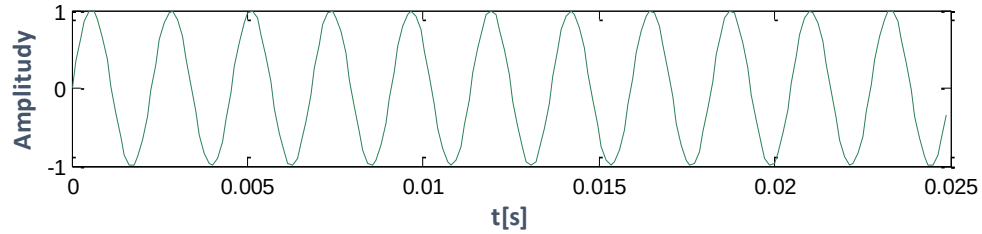
```
int procento = (int)((100/(horni-dolni))*(frq-dolni));
```

4 Ukázky funkce

Pro názornou ukázku funkce aplikace, byli vytvořeny dva příklady. První z nich je spíše simulovaný. Je vygenerován signál s jednou hlavní frekvencí pomocí programu Matlab a přehráván z reproduktoru. Druhý příklad ukazuje měření frekvence reálného zvuku. Zde je jen zahrán tón na kytaru.

4.1 Vygenerovaný zvukový signál

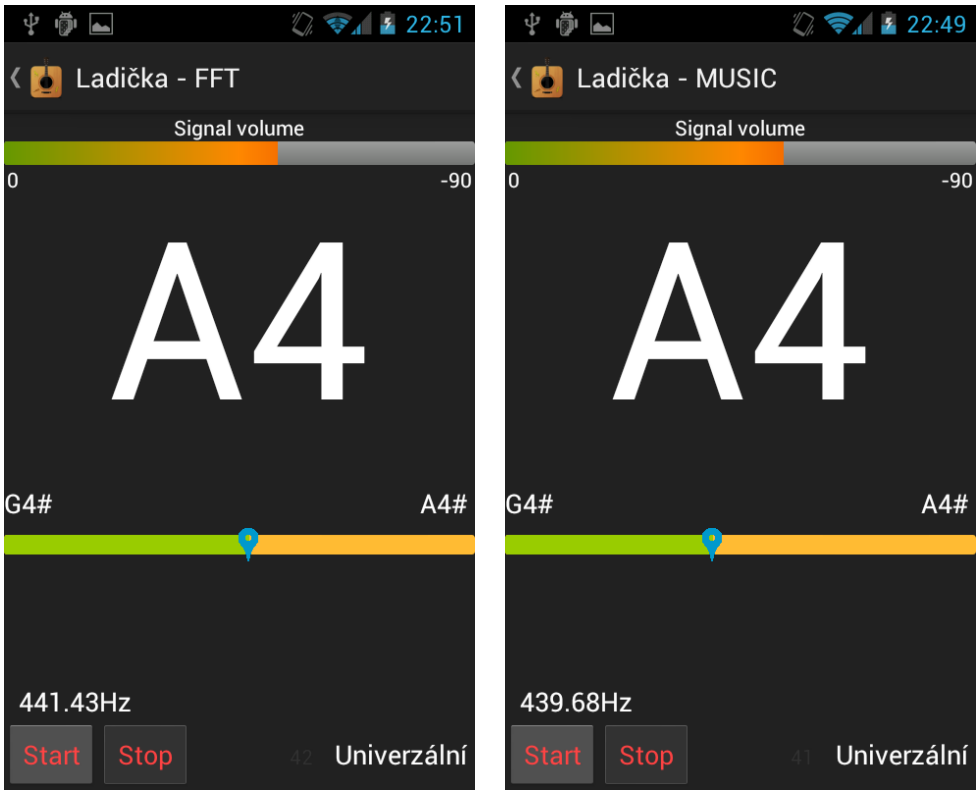
Pro ukázkový příklad byl vygenerován harmonický signál o frekvenci 440Hz. Signál je na Obr. 15.



Obr. 15 Vygenerovaný signál

Pokud je použita metoda FFT, je frekvence zobrazena téměř okamžitě, ale vlivem frekvenčního rozlišení není výpočet úplně přesný. U metody MUSIC, je 1 – 2 sekundové zpoždění zobrazení výsledku a to díky výpočetní náročnosti této metody. Výsledek výpočtu pomocí FFT je tedy 441.43Hz. Když je to samé změřeno pomocí

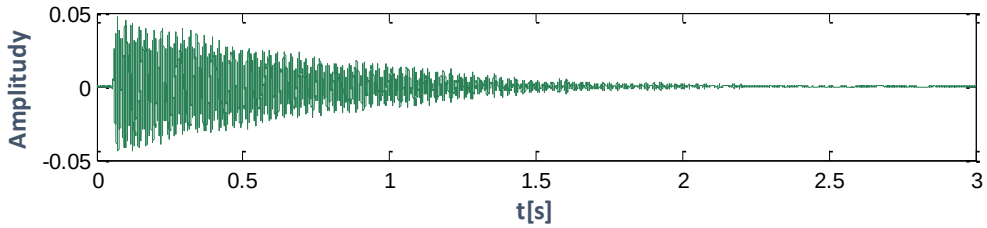
metody MUSIC vyjde nám hodnota bližší 439.68Hz. Tyto výsledky je vidět na Obr. 16.



Obr. 16 Vlevo naměřeno pomocí FFT, vpravo naměřeno pomocí MUSIC

4.2 Zvukový signál z hudebního nástroje

Pro záznam reálného zvukového signálu z hudebního nástroje byla vybrána kytara, protože se jedná o jeden z nejvíc rozšířených hudebních nástrojů, pro který se využívá ladička jako tato. Jako snímaný tón byl vybrán tón G, jehož frekvence byla nejprve změřena pomocí programu Matlab a je tedy 193Hz. Graf takového signálu je vyobrazen na Obr. 17.



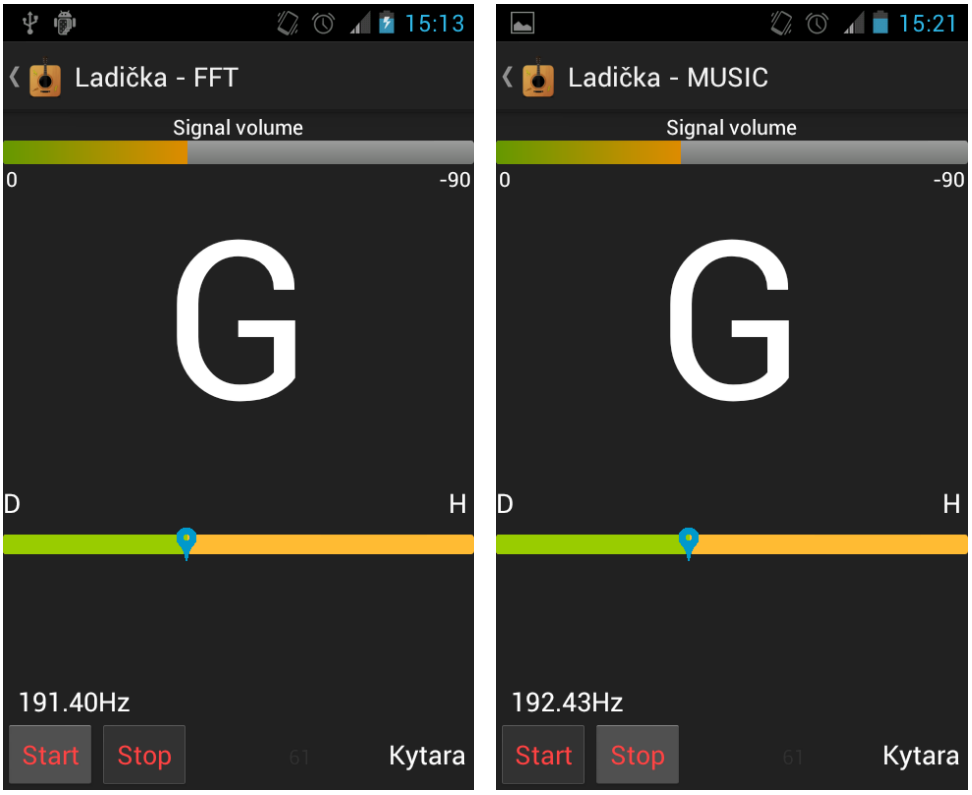
Obr. 17 Graf zahraného tónu G

Výsledek měření pomocí FFT byl při všech pokusech 191,4Hz. Zatímco u měření pomocí metody MUSIC se frekvence přibližovala ke 193Hz. Naměřené hodnoty je vidět v tabulce Měření frekvencí.

Tabulka 2 Měření frekvencí

Měření	Matlab	FFT	MUSIC
1.	193Hz	191,4Hz	192,28Hz
2.	193Hz	191,4Hz	192,96Hz
3.	193Hz	191,4Hz	191,55Hz
4.	193Hz	191,4Hz	192,43Hz
5.	193Hz	191,4Hz	192,22Hz

Ale i zde byla metoda MUSIC mnohem blíže správnému výsledku. Výsledek měření je na Obr. 18.



Obr. 18 Vlevo naměřeno pomocí FFT, vpravo naměřeno pomocí MUSIC

Lze tedy říci, že metoda FFT naměří vždy alespoň přibližné hodnoty. Kdežto metoda MUSIC vždy najde nejvyšší frekvenci v zaznamenaném signálu.

Závěr

Mým úkolem bylo vytvořit Aplikaci Inteligentní ladičky pro operační systém Android. Vybral jsem tedy nejvhodnější způsob pro snímání zvukového signálu. A následně jsem realizoval dva algoritmy pro měření fundamentální frekvence a to pro FFT a metodu MUSIC. Aplikace je určená pro mobilní zařízení, proto musela být nakonec optimalizována pro rychlost a menší výpočetní náročnost. V metodě MUSIC byla nakonec změněna její realizace ještě pomocí Newtonovy optimalizační metody. Pro co nejpřesnější nalezení maximální frekvence v pseudospektru.

Pro Aplikaci jsem vytvořil velmi jednoduché uživatelské rozhraní pro snadné ovládání. S ukazatelem naměřené frekvence, a jí odpovídající tón, jeho předchozí a následující tón. Vytvořen byl i ukazatel úrovně signálu, který spustí samotné měření frekvence a také ukazatel vzdálenosti od nejbližšího tónu. Vyhledání tónu, který odpovídá nalezené frekvenci je možné ve třech sadách tónů pro různé hudební nástroje. Pomocí aplikace je tedy možné naladit jakýkoliv hudební nástroj.

Seznam použité literatury

[1] Dashboards. GOOGLE. Android Developers [online]. 2013 [cit. 2013-02-04]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[2] Welcome: Android, the world's most popular mobile platform. Android Developers [online]. 2013 [cit. 2013-02-04]. Dostupné z: http://developer.android.com/about/index.html

[3] KONEČNÝ, Matěj. Vyvíjíme pro Android: První krůčky. Vyvíjíme pro Android [online]. 2012 [cit. 2013-02-14]. Dostupné z: http://www.zdrojak.cz/clanky/vyvijime-pro-android-prvni-krucky/

[4] Android Developers: Activity. Android Developers [online]. 2013, 13. 2. 2013 [cit. 2013-02-14]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html

[5] Android (operační systém). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-03-12]. Dostupné z: http://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m)

[6] Android Developers: AudioRecord. GOOGLE. Android Developers [online]. 2013, 2. 4. 2013 [cit. 2013-04-08]. Dostupné z: http://developer.android.com/reference/android/media/AudioRecord.html

[7] MOON, Todd K a Wynn C STIRLING. *Mathematical methods and algorithms for signal processing*. Upper Saddle River: Prentice Hall, c2000, 937 s. ISBN 02-013-6186-8.

[8] JUREK, Miloš. *Numerické metody optimalizace*. Zlín, 2007. Dostupné z: http://dspace.k.utb.cz/bitstream/handle/10563/3990/jurek_2007_dp.pdf?sequence=1. Diplomová práce. Univerzita Tomáše Bati ve Zlíně. Vedoucí práce Ing. Radek Matušů.

Seznam obrázků