TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Multiplatformní aplikace pro hlasové ovládání

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Václav Langr**
Vedoucí práce: Ing. Petr Červa, Ph.D.
Konzultant: Ing. Ondřej Smola

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Multi-platform application for voice control

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Václav Langr**
Supervisor: Ing. Petr Červa, Ph.D.
Consultant: Ing. Ondřej Smola

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Václav Langr**
Osobní číslo:	**M16000176**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Multiplatformní aplikace pro hlasové ovládání**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Implementuje v jazyce Javascript knihovnu, která umožní v reálném čase snímat signál z mikrofonu a provádět jeho parametrizaci.
2. Rozšiřte funkcionalitu knihovny o možnost provádět na základě parametrizace detekci řečové aktivity. Vytvořený detektor bude využívat model založený na hlubokých neuronových sítích a dekodér ve formě stavového automatu.
3. Experimentálně ověřte a vyhodnoťte funkčnost knihovny na vhodně zvolených testovacích datech.
4. Ve zvoleném Javascriptovém frameworku vytvořte aplikaci s jednoduchým GUI, která bude využívat vytvořený detektor, zasílat detekované řečové segmenty na rozpoznávací server a zobrazovat výsledky rozpoznávání.
5. Rozšiřte vytvořenou aplikaci o modul, který umožní provádět na základě výsledků rozpoznávání povelové hlasové ovládání zvolené platformy (např. systému Windows).

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40-50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Řeč a počítač: principy hlasové komunikace, úlohy, metody a aplikace, sborník článků. Technická univerzita v Liberci, 2009.**

[2] **Eric Elliott: Programming Javascript Applications: Robust Web Architecture with Node, Html5, and Modern JS Libraries, O'Reilly Media, Inc.., 2014.**

Vedoucí diplomové práce:	**Ing. Petr Červa, Ph.D.** Ústav informačních technologií a elektroniky
Konzultant diplomové práce:	**Ing. Ondřej Smola** Ústav informačních technologií a elektroniky
Datum zadání diplomové práce:	**19. října 2017**
Termín odevzdání diplomové práce:	**14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou 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 diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou 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.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2018

Podpis:

Poděkování

Chtěl bych poděkovat vedoucímu této práce Ing. Petru Červovi, Ph.D. a také Ing. Ondřeji Smolovi za velice přínosné konzultace, rady a trpělivost, bez kterých by tato práce nevznikla.

Abstrakt

Tato diplomová práce se zaměřuje na vývoj nové generace již existujícího řešení pro ovládání osobního počítače pouze pomocí rozpoznaných hlasových povelů. Diplomová práce provází použitým typem parametrizace, samotným detekováním řečových a neřečových segmentů, rozpoznáním hlasového povelu a následným vykonáním rozpoznaného povelu. Zároveň ukazuje strukturu možných povelů a její rozšiřitelnost.

Vypracování probíhalo v několika navazujících krocích. Nejdůležitější částí bylo vytvoření knihovny pro samotnou parametrizaci vstupního audio signálu. Po této fázi následovalo vytvoření dalšího modulu pro detekci řečových úseků a jejich odeslání rozpoznávacímu serveru. Po úspěšně zvládnutém odesílání řečových segmentů následuje konstrukce struktury pro samotné ovládání počítače.

Po konečném vytvoření potřebných modulů autor přistoupil k tvorbě samotné multiplatformní aplikace. Vytváření probíhalo pomocí knihovny Electron, která umožňuje tvorbu aplikací s využitím běžně dostupných webových technologií, např. HTML, CSS nebo JavaScript.

Klíčová slova: Electron, JavaScript, hloubkové neuronové sítě, hlasové ovládání, detektor řečové aktivity

Abstract

The focus of this master thesis is on the development of a new generation based on an existing solution for controlling a personal computer only by using recognized spoken commands. The master thesis describes the type of parameterization used, detection of speech and non-speech segments, recognition of spoken commands and execution of a recognized command. At the same time, the master thesis describes the structure of possible commands and its extensibility.

The elaboration has been divided into separate steps. The most important part was the creation of a library for parameterization of an audio signal input. After this step a new module for detection of speech segments and its dispatch to a recognition server was created.

After the successful creation of a speech segment sender the construction of a computer controller structure followed. After the successful creation of all necessary modules, the author created the final cross-platform application. The application was created by using an Electron framework, which allows the creation of applications by using common web technologies, for example HTML, CSS or JavaScript.

Keywords: Electron, JavaScript, deep neural networks, voice control, speech activity detector

Obsah

Seznam obrázků

Seznam vzorců

1 Úvod

Hlavním záměrem vzniku této diplomové práce bylo vytvoření nové generace již existujícího řešení a to v podobě programu MyVoice. Ten se zaměřuje obzvláště na lidi s tělesnými postiženími a jejich potřebu ovládat počítač výlučně pomocí hlasových povelů, tj. bez použití rukou. Obecně lze popsat dvě existující varianty přístupu. První možností je izolované zpracování a v tomto případě bude tedy program očekávat vždy pouze jediný povel v promluvě. To je přístup využitý již ve zmíněném programu MyVoice. Druhou alternativou je pak spojité zpracování, které umožňuje uživateli vyslovit několik povelů za sebou bez toho, aby musel čekat po vyřčení prvního povelu na jeho provedení.

Na trhu existuje několik možných řešení. Jednou z možností je využít funkce operačního systému Windows a to funkce Rozpoznávání řeči. Funguje velice obdobně jako program MyVoice, bohužel není připraven pro český jazyk. V operačním systému Windows ale existuje další funkce to a Cortana. Ta umožňuje, podobně jako konkurenční Siri od firmy Apple Inc., vytvářet události v kalendáři, diktovat email, ovládat nastavení počítače, apod. Nicméně neumožňuje ovládání pohybu a simulování stisku tlačítek myši.

Další z možností, která se nabízí pro český trh, je program JetVoice. Ten je velice podobný programu MyVoice, umožňuje také ovládání počítače pouze pomocí hlasových povelů, přidávání nových povelů, problémem však je, že vznikl již v roce 2002 a byl udržován pouze do roku 2016. A navíc je tento program dostupný, jako program MyVoice, pouze pro uživatele operačního systému Windows.

Aktuální generace využívá standardní algoritmy pro rozpoznání hlasového povelu izolovaného diktování. Problémem této verze jsou ale obtíže spojené s distribucí

a udržovatelností. Program MyVoice je kompletně instalován na straně uživatele a tedy při aktualizaci je nutný jeho aktivní zásah. Tato nevýhoda by měla být odstraněna v nové verzi tak, že se bude jednat o aplikaci typu klient-server a tedy pro aktualizaci nebude muset uživatel podnikat žádné další kroky.

Dalším přínosem nové generace by měla být možnost ovládat i jiné platformy, než současné systémy podporují. Ve stávající verzi je program MyVoice dostupný pouze pro operační systém Windows. S novou generací dojde k rozšíření i na další operační systémy, např. macOS nebo Linux.

Neposledním důvodem vzniku nové generace je i to, že v aktuální generaci je nutná synchronizace uživatele s programem a to tak, že uživatel vyřkne vybraný povel, následně čeká na rozpoznání povelu a poté ještě na jeho vykonání. Zároveň je uživatel omezen tím, že nemůže říct více příkazů za sebou. Tato práce se snaží popsané problémy odstranit a aplikaci tak zefektivnit.

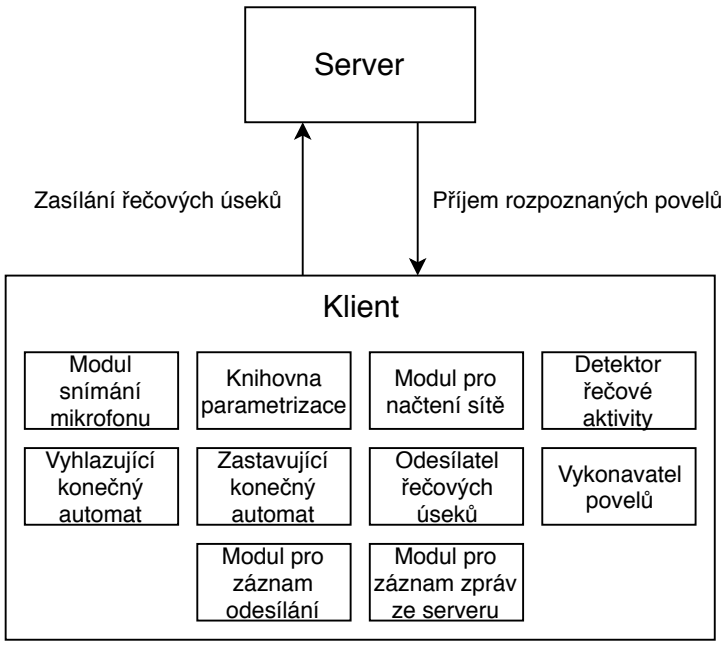
1.1 Cíle práce

Během vypracovávání diplomové práce tak vznikne nový systém zaměřený primárně na handicapované, kteří nemají jinou možnost, než využít hlasového ovládání pro kontrolu osobního počítače. Tento systém pak bude fungovat jako multiplatformní, zároveň bude také lépe udržovatelný. Nová aplikace by pak měla využívat spojitého zpracování a docílit tak komfortnějšího a intuitivnějšího používání. Zároveň tak bude eliminován problém synchronizace uživatele s aplikací.

Výsledkem této diplomové práce by měl být prototyp nové generace. Ten bude obsahovat pouze vybranou skupinu hlasových povelů pro ovládání myši. Po dokončení prototypu se předpokládá další rozšiřování tak, aby byla umožněna plná konfigurovatelnost aplikace, tak jak je to možné v současné verzi programu MyVoice.

2 Struktura navržené multiplatformní aplikace

Jak již bylo naznačeno v předchozí kapitole, struktura aplikace byla navržena typu klient-server. Server rozpoznávající hlasové povely je již připravený Ústavem Informačních technologií a elektroniky a je tedy nutné vytvořit klienta. Výsledná struktura má pak následující podobu:



Obrázek 2.1: Struktura multiplatformní aplikace

S navrženou strukturou souvisí i volba jazyka. V dnešní době existuje několik přístupů k vývoji aplikací s různými jazyky. Podle zvoleného jazyka je následné

přenesení aplikace pro další platformu různě obtížné. Je možné zahájit vývoj aplikace ze zadaného tématu s některým z tradičnějších jazyků, jako je např. C++ nebo Java, ale v dnešní době, kdy je zaměřena pozornost nejvíce na webové aplikace, je vhodné myslet i na tuto alternativu. Proto byl vybrán jazyk JavaScript, který je nejvíce používaný právě v této oblasti.

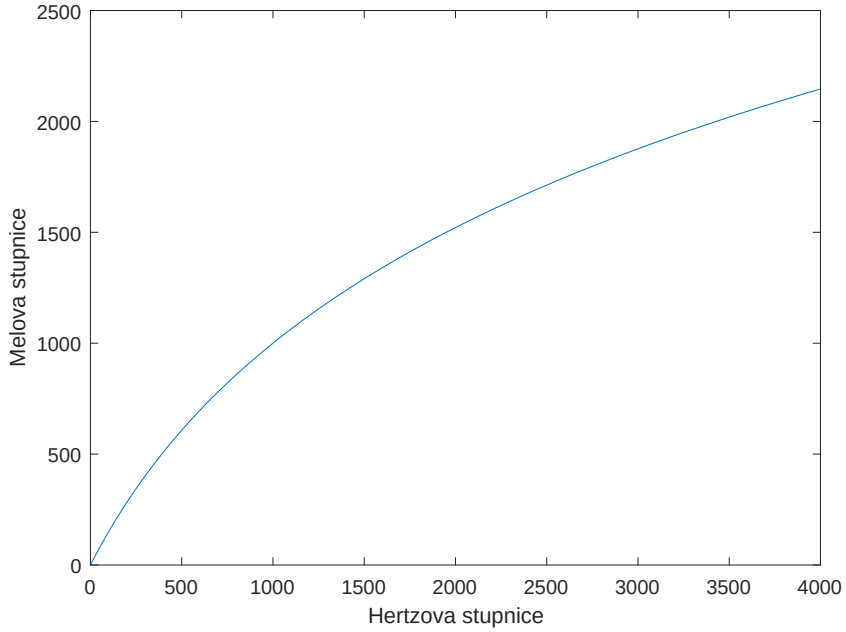
Jedná se o interpretovaný multiplatformní jazyk s podporou objektově orientovaného programování. Ve většině případů se používá jako doplňující element webových stránek umožňující pokročilou funkcionalitu. Avšak existují i knihovny umožňující vytvoření klasické aplikace pro stolní počítače.

Vývoj klienta byl, dle zvolené struktury, rozdělen do nezávislých modulů. Ty provádějí jednotlivé činnosti, popsané v kapitole 3 a komunikují spolu přes využití zpětného volání nebo asynchronních volání. Právě toto synchronizování je velice obtížné, protože jazyk JavaScript je pouze jednovlákenný a neumožňuje vytváření pomocných vláken jako např. jazyk Java nebo C++. Veškerá činnost pak probíhá tak, že snímaný signál z mikrofonu je vždy předán kaskádě funkcí pro parametrizaci z vytvořené knihovny. Vektor parametrů je následně předán modulu ShiftBuffer, který uchovává potřebný počet těchto vektorů pro dopředný průchod neuronovou sítí a vyhlazovacím automatem. Pouze když je zjištěna řeč, dojde k nastavení příznaků v modulu snímání mikrofonu a modulu odesilatel. Uvedený příznak pak indikuje řečový stav. Odesilatel následně zahájí komunikaci se serverem a začne odesílat na vyžádání serveru řečové úseky. Ze serveru přichází postupně rozpoznané hluky a povely, které se předají modulu vykonávajícímu povely, kde je využita knihovna RobotJS pro vykonávání jednotlivých povelů, viz kapitola 3.4. Použitá knihovna je multiplatformní a dokáže tudíž ovládat různé operační systémy. Takto vznikne provazující modul, který využívá grafické uživatelské rozhraní vytvořené pomocí knihovny Electron. To je popsáno v kapitole 4.

3 Vyvinuté moduly

3.1 Parametrizace

V této kapitole bude podrobně popsán postup získání vstupního signálu ve zvoleném jazyce a následná parametrizace. Lze pracovat v klasické Hertzové stupnici, avšak pro reálnou řeč toto není výhodné. Klasická Hertzova stupnice dává důraz na všechny frekvence ve stejné míře, ačkoliv pro zpracování řeči je důležitá oblast nižších frekvencí, ve kterých probíhá řeč a naopak je méně důležitá oblast vyšších frekvencí, kde je již velmi málo řečové aktivity. Z popsaného je výhodnější využít Melovu stupnici, která zohledňuje důležité frekvenční oblasti. Ukázku Melovy stupnice lze vidět na následujícím obrázku:

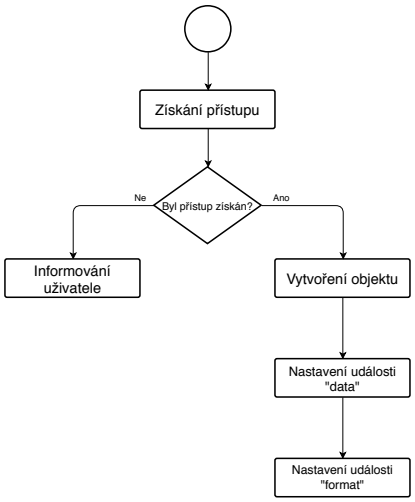


Obrázek 3.1: Ukázka Melovy stupnice

3.1.1 Příprava signálu

Prvním, a také nejdůležitějším, krokem je získání vstupního signálu z analogově digitálního převodníku. Toho lze ve zvoleném jazyce docílit několika způsoby. Jednou z možností je využití některé z běžně dostupných knihoven ve správci balíčků Node.JS[15]. Takových knihoven existuje nepřeberné množství, např. RecordRTC, React Native audio recording a byly pro účely této práce postupně vyzkoušeny. Alternativou je velmi jednoduchá knihovna microphone-stream, která ale neposkytuje žádnou další funkcionalitu. Postupným testováním vybraných knihoven se ovšem narazilo na problém. Ne všechny knihovny spolupracují se zvolenou knihovnou pro vytvoření výsledné aplikace, o které bude řeč v kapitole 4. Postupným zkoušením, proč k tomuto problému dochází, bylo zjištěno, že knihovny nesprávně žádají o potřebné oprávnění a tudíž se žádost o oprávnění k využití uživatelova mikrofonu nezobrazí. Takovéto chování není žádoucí a aplikace by se měla chovat tak, že obdrží přístup k mikrofonu, aniž by uživatel musel žádost potvrzovat. Proto byla zvolena knihovna microphone-stream, která toto splňuje.

Knihovna poskytuje velmi jednoduché aplikační rozhraní a postupuje se dle následujícího diagramu:

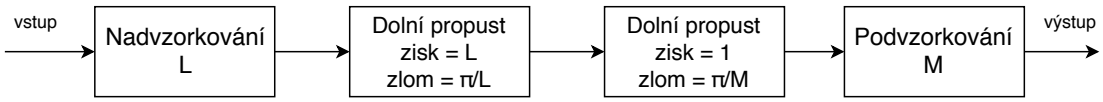


Obrázek 3.2: Postup užití vybrané knihovny

Jak je patrné z diagramu, nejdříve se získá potřebné oprávnění pro přístup k mikrofonu. Následuje vytvoření a nastavení instance třídy MicrophoneStream. Po úspěšném vytvoření instance je dále důležité nastavit potřebné události a to konkrétně událost „data“ a „format“. První z těchto událostí slouží k určení, jak se bude zpracovávat získaná část dat. Naopak událost „format“ slouží k získání hodnoty vzorkovací frekvence, se kterou jsou data snímána z mikrofonu. To je velmi podstatný údaj. V dalších krocích je pak využívána neuronová síť, která byla trénována na datech s konkrétní vzorkovací frekvencí.

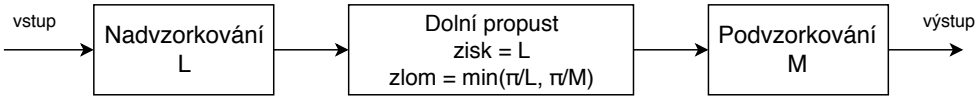
Po úspěšném vytvoření modulu, který se stará o snímání mikrofonu, se mohlo přistoupit k další části přípravy signálu, a to převzorkování. Převzorkování vstupního signálu je zásadní, jelikož zvolený jazyk neumožňuje nastavení konkrétních hodnot pro snímání a používá základní hodnoty zjištěné operačním systémem. Pro tento modul jsou v zásadě možné dvě varianty řešení. První z nich, je vytvoření kaskády zpracovávajících bloků a to například z:

1. Nadvzorkování signálu s vložením $L - 1$ nul mezi jednotlivé vzorky
2. Dolnopropustní filtr s hraniční frekvencí $\frac{\pi}{L}$
3. Dolnopropustní filtr s hraniční frekvencí $\frac{\pi}{M}$
4. Podvzorkování signálu s uložením každého Mtého vzorku



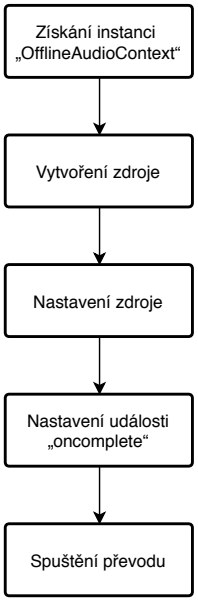
Obrázek 3.3: Základní kaskáda převzorkování

Vznikne tak kaskáda jako na obrázku 3.3. Dále je možné uvedenou strukturu upravit. Jelikož jsou v kaskádě dva dolnopropustní filtry, lze je nahradit pouze jedním dolnopropustním filtrem s hraniční frekvencí $min\left(\frac{\pi}{L}, \frac{\pi}{M}\right)$.



Obrázek 3.4: Zjednodušená kaskáda převzorkování

Alternativou ke kaskádovému zpracování je využití možnosti zvoleného jazyka, jež v sobě obsahuje již implementovanou možnost převzorkování signálu. Z dokumentace [10] bylo zjištěno, že k tomu lze využít dostupný objekt OfflineAudioContext s postupem dle následujícího diagramu:



Obrázek 3.5: Převzorkování pomocí objektu OfflineAudioContext

Při vytváření nové instance třídy se zadá počet kanálů vstupního signálu, počet vzorků a cílová vzorkovací frekvence. Dále je důležité nastavit proměnnou „oncomplete“. V tomto případě se bude jednat o funkci udávající, co se stane po převzorkování signálu. V následujícím kroku se pomocí tohoto objektu vytvoří nový

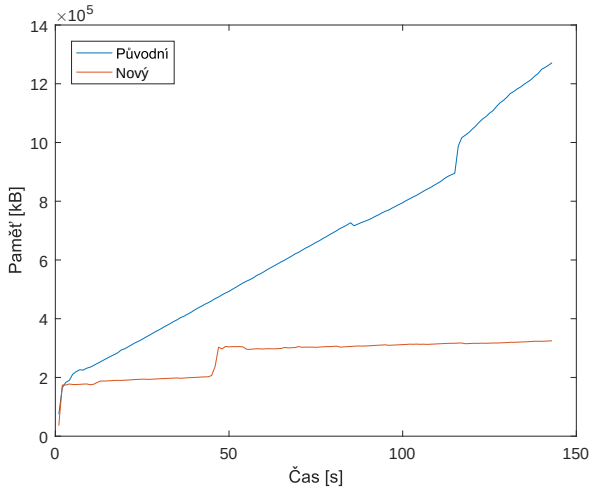
objekt typu AudioBuffer s obdobnými parametry, jen vzorkovací frekvence je tentokrát vstupní. Právě do tohoto objektu se zkopírují vstupní data. Následuje vytvoření objektu typu AudioBufferSourceNode pomocí funkce „createBufferSource“ objektu AudioBuffer. Po potřebném propojení objektů již dochází pouze ke spuštění převodu.

Při prvním testování tohoto modulu se ukázalo, že modul není dostatečně rychlý pro reálné použití. Postupným zjišťováním příčiny bylo odhaleno, že je tento problém způsoben modulem pro převzorkování. Při pozorování chování tohoto modulu zároveň byl odhalen nedostatek vybraného přístupu k řešení daného problému a tím jsou vysoké paměťové nároky. Jelikož se jedná primárně o prostředek pro převzorkování audio nahrávky a nikoliv potenciálně nekonečného proudu dat, výsledná data zůstávají v dočasné paměti jako aktivní, tj. s referencí na ně, takže je nelze odstranit pomocí algoritmu Garbage Collector.

Jak již bylo zmíněno, jazyk JavaScript neumožňuje zvolení konkrétních hodnot pro snímání mikrofonu. Proto byl přepracován i modul pro snímání mikrofonu a nahrazen pokročilejší knihovnou, která toto umožňuje. Z uvedeného důvodu byla vybrána knihovna RecordRTC jako vhodná náhrada. Ta, dle dokumentace [7], funguje velice obdobně jako dosud navržený přístup, avšak jsou zde drobné odlišnosti. Jak je již zřejmé, základní odlišnost je možnost volby vzorkovací frekvence a bitové hloubky. Ačkoliv se může jevit, že tato knihovna upravuje snímání mikrofonu, probíhá zde zároveň se snímáním také převzorkování, čímž je dosaženo možnosti volby parametrů snímání mikrofonu. Zároveň umožňuje definovat v jakém datovém formátu chceme získat snímaná data.

Poslední odlišností je formát, v jakém jsou snímaná data získána. V předchozím případě se jednalo o vzorky signálu, nyní jsou získány objekty typu Blob. Tyto objekty jsou ve skutečnosti soubory zadaného datového typu, které jsou uchovávány pouze v paměti. Proto je nutné je před dalším zpracováním načíst. K tomu lze využít standardní třídy FileReader skriptovacího jazyka JavaScript, a následně je dekódovat podle typu souboru. Další vlastností tohoto objektu je to, že ho lze označit jako neaktivní a tak umožnit automatické spravování paměti pomocí algoritmu Garbage

Collector. Touto změnou došlo nejen ke zrychlení celého vytvářeného modulu, ale také paměťové nároky se velice snížily, viz 3.6.



Obrázek 3.6: Porovnání paměťových nároků

3.1.2 MelFBank příznaky

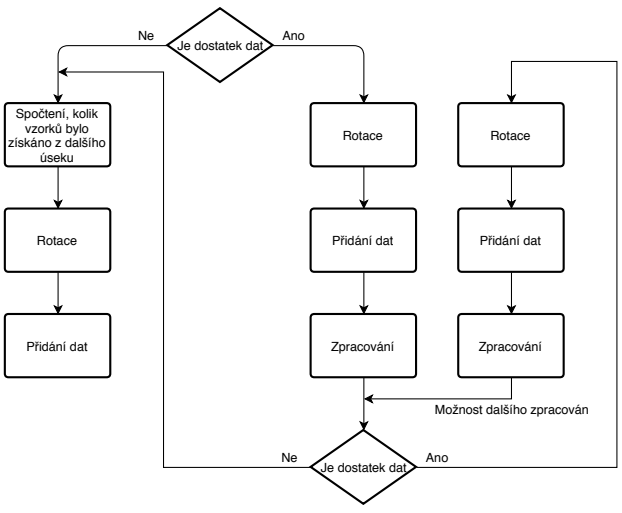
Po přípravě signálu je možné přistoupit k samotné parametrizaci. Ta je rozdělena do jednotlivých kroků. Prvním z nich je přičtení náhodného šumu o velmi malé hodnotě ke každému vzorku signálu, typicky v rozsahu $\langle -1, 1 \rangle$. To se provádí z důvodu typových rozdílů převodníků používaných u zvukových karet. Na nich lze pozorovat přidání několika nulových hodnot na začátku a na konci vzorkovacího procesu. To by značně komplikovalo výpočetně výhodnější zpracování pomocí funkce logaritmu, který není definován pro hodnotu v bodě 0.

Po přidání náhodného šumu následuje odstranění stejnosměrné složky zvukové karty. Stejnosměrnou složku je možné odstranit aplikováním hornopropustního filtru dle následujícího předpisu:

$$y(n) = x(n) - \alpha \cdot x(n-1) \qquad \text{(Vzorec 1)}$$

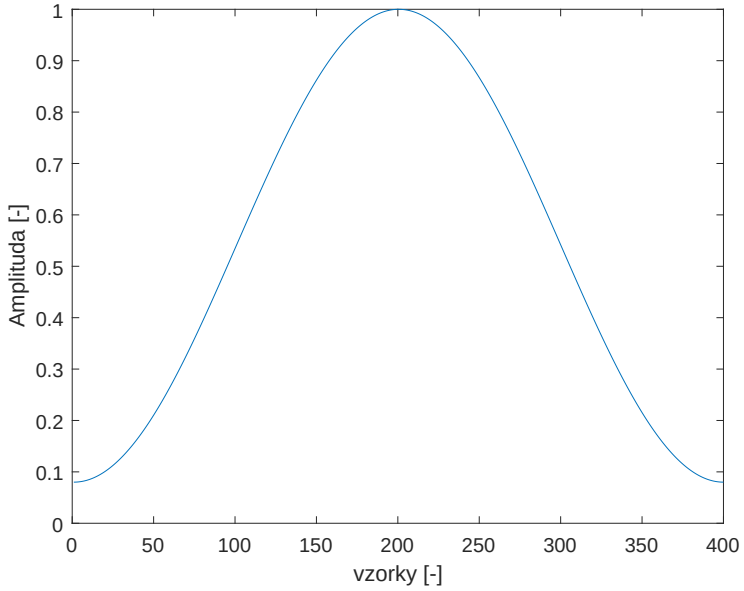
Jak je patrné, pro prvek vektoru y s indexem n platí, že je rozdílem prvku vstupního signálu x s indexem n a upravené hodnoty vstupního signálu s indexem $n-1$ proměnnou α. Pro α proměnnou platí $\alpha \doteq 1$, obvykle je pro tuto proměnnou udávána hodnota $\alpha = 0,97$, což vychází z praktických zkušeností, kdy se tato hodnota jeví jako nejideálnější.

Dalším krokem je rozdělení vstupního signálu do krátkých úseků s požadovanou délkou a překryvem. K tomu byla vytvořeno třída ShiftBuffer, která bude toto realizovat. Při vytváření nové instance je nutné specifikovat parametry jako velikost sledovaného okna, velikost překryvu jednotlivých oken a funkce, které se předá sledovaný úsek dat. Třída samotná pak již obsahuje jen dvě podstatné funkce a to „addData“ a „rotateBuffer“. Funkce „rotateBuffer“ je velice jednoduchá, stará se pouze o posun prvků v interní proměnné pomocí cyklu. Naopak funkce „addData“ je složitější a obsahuje tedy prakticky veškerou funkcionalitu. Objekt v sobě uchovává, kolik prvků již obsahuje z následujícího úseku. Pokud je zbytek menší nebo roven délce nových dat, dojde k překopírování části dat a následnému zpracování úseku. Následuje kontrola, zda jsou k dispozici další prvky. Pokud jsou, tak dokud je to možné, dojde k jejich překopírování a zpracování. Zbytek dat se také zkopíruje do vnitřní proměnné a počet prvků z dalšího úseku se změní dle zbylých dat. Celý algoritmus je popsán na následujícím diagramu:



Obrázek 3.7: Algoritmus funkce „addData“

Pro takto rozdělené úseky již lze spočítat MelFBank příznaky. Nejprve se daný úsek vynásobí Hammingovým okénkem po prvcích. Průběh tohoto okénka je znázorněn na obrázku 3.8.



Obrázek 3.8: Ukázka Hammingova okénka

Rozhodující je vytvoření trojúhelníkových filtrů k určení váhy magnitudového spektra, což bude popsáno dále. K vytvoření těchto filtrů v Melově stupnici je nutné specifikovat nejmenší povolenou frekvenci, největší povolenou frekvenci a také počet filtrů N. Pro převod z klasické Hertzové stupnice do Melovy stupnice je použit následující vzorec:

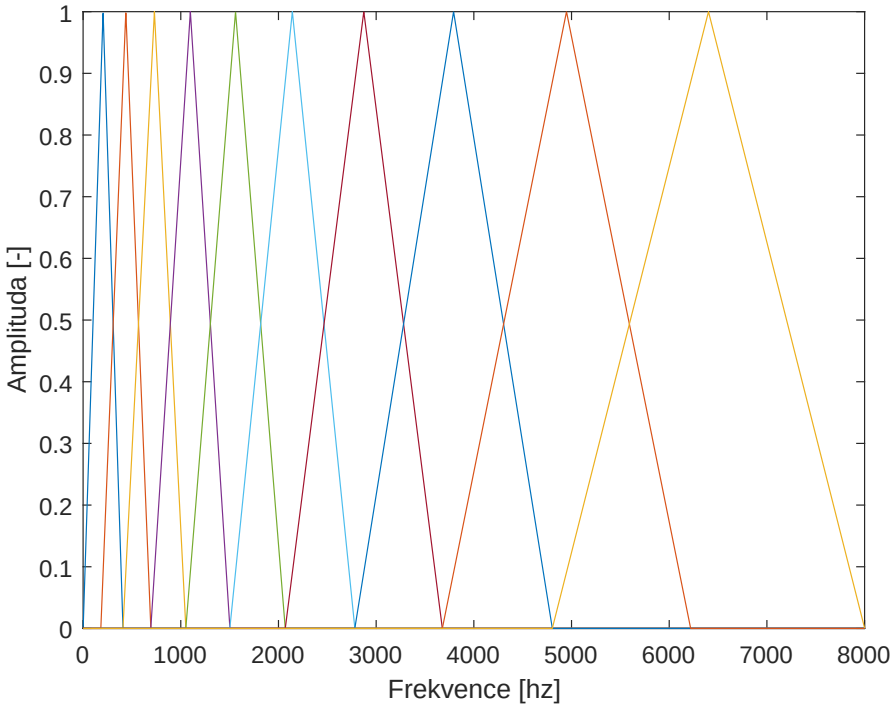
$$mel = 1127 \cdot \ln\left(1 + \frac{hz}{700}\right) \qquad \text{(Vzorec 2)}$$

Pomocí tohoto vztahu jsou zjištěny hodnoty pro nejmenší, resp. největší, povolenou frekvenci v Melově stupnici. Následně lze vytvořit Nprvkový vektor aritmetické posloupnosti s krajními body v nejnižší a nejvyšší povolené frekvenci v Melově

stupnici. Takto získaný vektor je následně zpětně převeden do klasické Hertzovy stupnice a to pomocí inverzní funkce k Vzorec 2. Celý tento proces byl proveden z důvodu získání reprezentativnějších příznaků, než by bylo možné získat v klasické Hertzově stupnici.

$$hz = 700 \cdot \left(e^{\frac{mel}{1127}} - 1\right) \qquad \text{(Vzorec 3)}$$

S takto připraveným vektorem již lze sestavit trojúhelníkové filtry, viz obrázek 3.9. Jak je z něho patrno, vytvořené průběhy jednotlivých filtrů mají tvar rovnoramenných trojúhelníků, kde každý další trojúhelník daného filtru začíná přesně pod vrcholem trojúhelníku předchozího filtru.



Obrázek 3.9: Ukázka 10 trojúhelníkových filtrů pro vzorkovací frekvenci o hodnotě 16 kHz

Jelikož se bude dále využívat Fourierova transformace, resp. její rychlá alternativa, je důležité zkontrolovat délku dat a případně ji doplnit nulovými prvky. To umožní použití výpočetně méně náročných algoritmů. S takto upravenými daty již lze provést rychlou Fourierovu transformaci. Pro další kroky je možné zpracovávat pouze polovinu výsledného spektra, využívá se pro tyto účely výhodná vlastnost zrcadlové symetrie spektra. Ze spektra, které je v oblasti imaginárních čísel, je následně zjištěno magnitudové spektrum v oblasti reálných čísel a to vypočtením absolutní hodnoty frekvenční oblasti. Pro získání příznaků již stačí vynásobit matici trojúhelníkových filtrů spolu se zjištěným magnitudovým spektrem. Vznikne tak Nprvkový vektor příznaků. Ovšem i ten je nutné z parametrických důvodů prahovat a to tak, že při hodnotě menší než je daný limit, bude nastavena nová hodnota. Pokud je však hodnota větší než zvolený práh, je nová hodnota zjištěna její logaritmickou hodnotou.

Dalším krokem k vypočtení MelFBank příznaků jsou normalizace. První z nich využívá průměrů zjištěných pomocí několika příznakových vektorů. Normalizace pak probíhá odečtením zjištěného lokálního průměru od jednotlivých hodnot. Druhá z normalizací probíhá již pomocí globálně zjištěných hodnot. Ty byly zjištěny a poskytnuty Ústavem informačních technologií a elektroniky. Zde již normalizace neprobíhá pouze odečtením průměrné hodnoty, ale také jejím vydělením zjištěnou směrodatnou odchylkou.

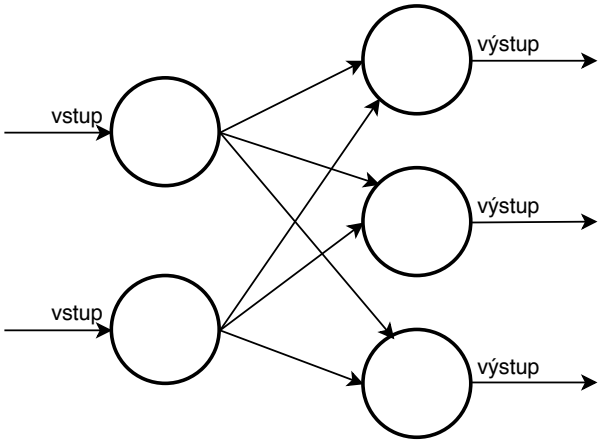
V následující fázi se bude využívat neuronová síť pracující s kontextem. To bude zpracován aktuální příznakový vektor a jeho okolí, které lze nastavit před samotným trénováním sítě. V tomto případě se zpracovává aktuální příznakový vektor, 25 minulých a 25 budoucích příznakových vektorů. Jak je tedy patrné, je zde nutné zavést zpoždění. Musí totiž dojít k načtení prvních 25 hodnot, než bude možné získat relevantní hodnoty. K těmto účelům lze využít již vytvořenou třídu ShiftBuffer, která přesně tuto funkcionalitu obsahuje, stačí jen nastavit příslušné parametry, tj. velikost okna, překryvu okna a zpracovávající funkce.

3.2 Detekce řečových segmentů

Pro optimalizaci velikosti rozpoznávaných dat je důležité detekovat pouze řečové segmenty a teprve ty rozpoznávat. K tomu lze využít princip hlubokých neuronových sítí.

3.2.1 Použitá neuronová síť

Jelikož navržení a natrénování neuronové sítě není hlavním cílem této diplomové práce, byla již natrénovaná síť poskytnuta Ústavem informačních technologií a elektroniky. Jelikož již byly získány reprezentativní příznaky, není nutné využít zde některé ze složitějších typů vrstev neuronových sítí, ale pouze plně propojené vrstvy a ušetřit tak výpočetní výkon a případně také paměť.



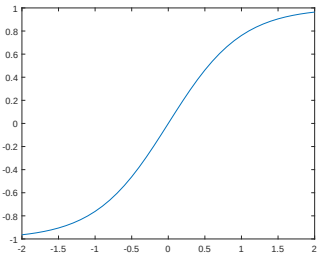
Obrázek 3.10: Ukázka plně propojené vrstvy

Využitá neuronová síť se skládá z plně propojených vrstev a má tak podobu jako na obrázku 3.10. Je složena ze vstupní vrstvy, šesti skrytých vrstev a výstupní vrstvy. Aby bylo možné vůbec neuronovou síť použít, musely být vytvořeny další moduly. První z nich slouží k načtení neuronové sítě z binárního souboru. To se skládá z načtení hlavičky souboru a pomocí zjištěných hodnot, tj. počtu vrstev

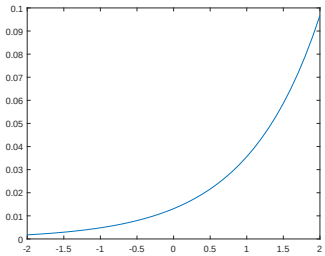
a jejich počtu neuronů. Další součástí uvedeného modulu je pak načtení zbytku souboru.

Jako druhý byl vytvořen modul realizující dopředný průchod neuronovou sítí. Ten probíhá v cyklu tak, že pro každou vrstvu se spočítá výstup z konkrétní vrstvy, pomocí konfigurace je zjištěna aktivační funkce a v dalším kroku je použit výstup z vybrané aktivační funkce, které byl předán výstup z dané vrstvy neuronové sítě. Získání konkrétní funkce pak probíhá pomocí návrhového vzoru Factory, kdy za pomoci názvu funkce je vrácena reference na příslušnou realizující funkci.

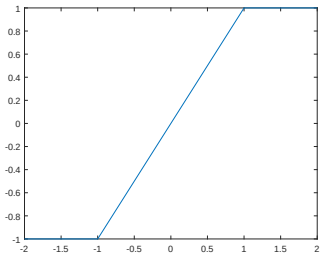
Jako aktivační funkce bylo implementováno hned několik funkcí, aktuálně používaná funkce *tanh* a pro budoucí potřeby funkce $SoftMax$ a *hard tanh*. Průběhy těchto funkcí lze sledovat na připojených zobrazeních:



Obrázek 3.11: tanh



Obrázek 3.12: SoftMax



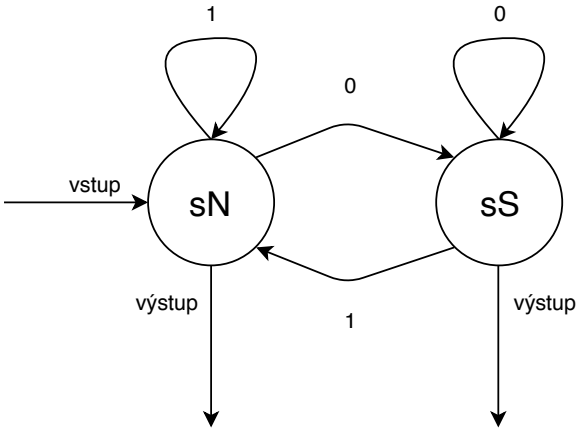
Obrázek 3.13: hard tanh

3.2.2 Vyhlazení výstupu neuronové sítě

Výstup neuronové sítě však může kolísat, je proto nutné výstup z neuronové sítě vyhladit tak, aby se toto co nejvíce eliminovalo. Z tohoto důvodu byl využit a naimplementován konečný automat, který umožňuje velice efektivně řešit popsaný problém.

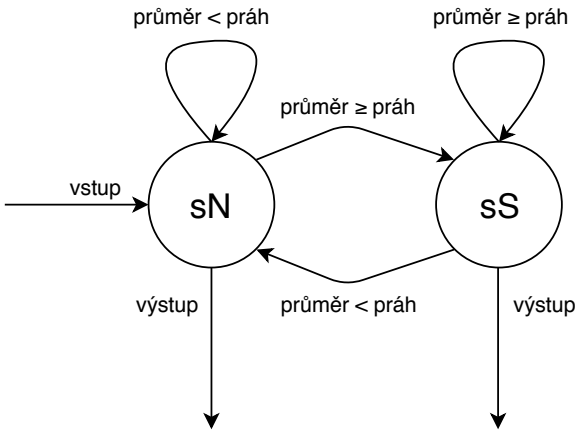
Jako první byl vytvořen samotný konečný automat. Ten se skládal pouze ze dvou stavů „sS“ reprezentující řečový stav a „sN“ reprezentující klidový stav. Pro přechod z jednoho stavu do druhého stačí, aby na vstupu byla hodnota 0,

reprezentující řečový úsek, nebo 1 reprezenzující klidový úsek. Automat tak získal následující podobu:



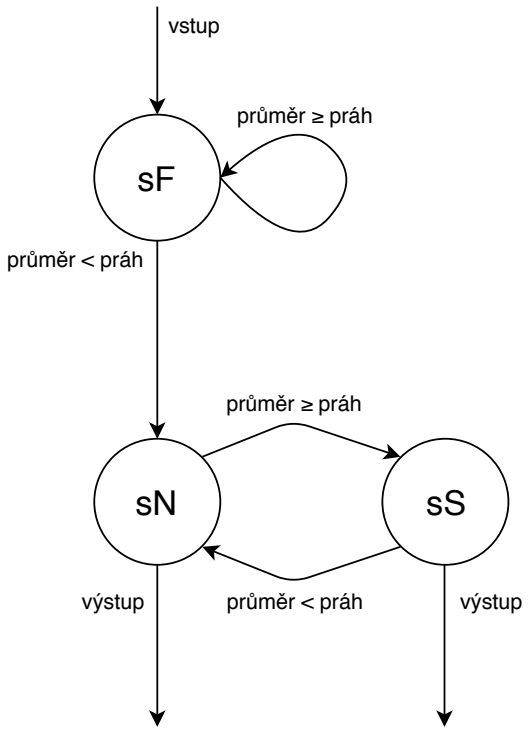
Obrázek 3.14: Základní model automatu

Nicméně tato struktura neřeší daný problém, jelikož dochází k vracení stále stejné hodnoty, jako udává neuronová síť, dochází prakticky pouze ke kopírování hodnot. Proto musela být přidána logika, která bude ovlivňovat výstup konečného automatu, např. klouzavý průměr. Ten funguje na principu ukládání n posledních hodnot, ze kterých se zjišťuje průměr a podle zadaného prahu se rozhoduje, zda jde o řečový nebo klidový úsek. Automat má po této úpravě podobu jako na obrázku 3.15.



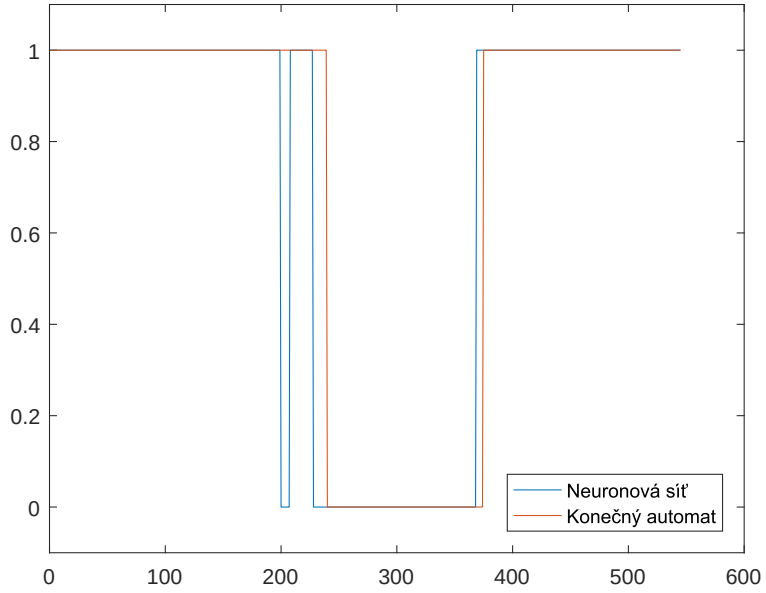
Obrázek 3.15: Model automatu s klouzavým průměrem

Během testování však bylo zjištěno, že ani tento model není dostačující. Při začátku snímání dat z mikrofonu je chybně rozpoznáno několik úseků jako řečových a dojde tak k pokusu o rozpoznání. To je dáno primární vlastností zvukových karet a jejich dynamickým nástupem, který je pak chybně detekován jako řečové úseky. Jelikož je součástí vytvářené aplikace i vizualizace detektoru řeči, mohl by se koncový uživatel chybně domnívat, že aplikace nefunguje správně. Proto byl přidán další stav a to filtrační. Základním principem jeho fungování je, že při začátku detekce kontroluje, zda je klouzavý průměr větší než zvolený práh. Pokud je průměr větší, pak zůstává ve stejném stavu a pouze pokud je průměr menší, tak se stav změní na „sN“. Konečná podoba automatu je tedy následující:



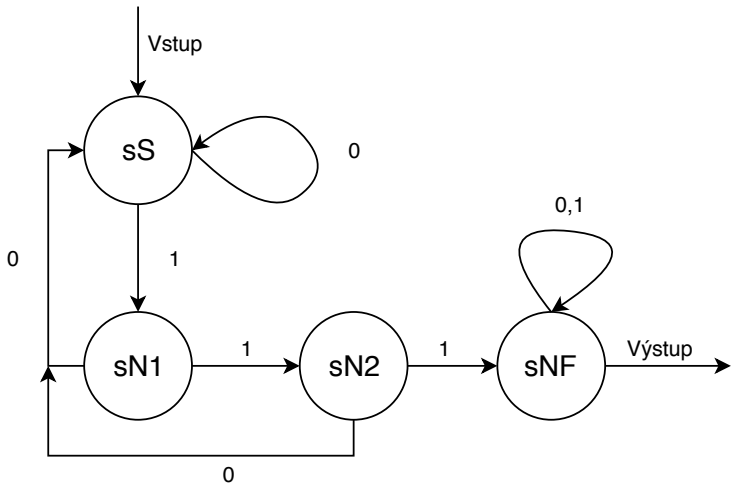
Obrázek 3.16: Model automatu s klouzavým průměrem a filtrací

Takto připravený konečný automat již úspěšně vyhlazuje výstup neuronové sítě. Ukázku vyhlazení výstupu lze pak vidět na následujícím obrázku:



Obrázek 3.17: Ukázka vyhlazení výstupu neuronové sítě konečným automatem

I s takto vyhlazeným výstupem neuronové sítě však může dojít k tomu, že daný řečový úsek bude chybně rozpoznán jako klidový a tedy nedojde k jeho odeslání na rozpoznávající server. Proto je tento automat použit pouze pro detekci začátku řeči a tedy musel být vytvořen další modul, který detekuje její konec. K tomu lze opět využít přístup konečných automatů. Byl tedy zkonstruován modul, který sestaví automat dle zadaných parametrů. Zjednodušeně lze říci, že navrhovaný automat funguje tak, že sleduje, kolik po sobě jdoucích klidových úseků bylo zjištěno z vyhlazeného výstupu neuronové sítě. Pouze pokud byl detekován zadaný počet klidových úseků, změní se stav automatu a zastaví se odesílání na rozpoznávající server. Automat se resetuje pouze tehdy, pokud se nedošlo do koncového stavu a byl rozpoznán řečový úsek a nebo v případě, že byl detekován začátek nového řečového úseku. Automat má pak následující podobu:



Obrázek 3.18: Model automatu detekující konec řeči pomocí tří neřečových úseků

3.3 Odesílatel a rozpoznání hlasového povelu

Jak již bylo předestřeno v kapitole 2, je nutné detekované hlasové úseky odeslat na server rozpoznávající, jaký hlasový povel byl řečen. Je možné vybrat z několika typů úloh, např. hlasové ovládání, diktování a další. Přístup k tomuto serveru spolu s dokumentací [4] byl poskytnut Ústavem informačních technologií a elektroniky.

Veškerá komunikace začíná přihlášením uživatele a získáním příslušných autorizačních a autentizačních tokenů. Následně je využita knihovna umožňující komunikaci se serverem a získávání odpovědí z něj. Ta umožňuje vytvoření klienta, který se nastaví pomocí následujících parametrů:

1. Doména serveru
2. Autorizační token vybrané úlohy
3. Formát snímaného zvuku, tj. datový typ vzorků, vzorkovací frekvence a počet kanálů

4. Slovník uživatele

Následně se při příchodu řečových úseků naváže spojení a určí se, jak mají být získané výsledky zpracovány. Existuje několik způsobů, jak výsledky zpracovávat a to podle typu úlohy. Jelikož se získávají i nepotvrzené hypotézy, je možné jich využít k vykonání operace a při příchodu potvrzené hypotézy buď vykonané operace opravit, při rozpoznání jiného povelu, nebo nevykonat nic při potvrzení správnosti původní hypotézy. Jelikož tyto nepotvrzené hypotézy lze získat výrazně rychleji než potvrzené, má to tu výhodu, že uživatel prakticky okamžitě vidí odezvu aplikace. To lze využít například u hlasového diktování, při kterém tak může dojít k opravě chybně rozpoznaných slov. Druhý přístup je filtrovat veškeré získané výsledky a zpracovat pouze potvrzené hypotézy. Zde je rychlost odezvy o něco nižší než v předchozím případě, ale nevyskytuje se zde komplikace s uchováním informace o předchozích stavech. Právě tento přístup byl zvolen pro další řešení.

3.3.1 Nekorektní rozpoznání

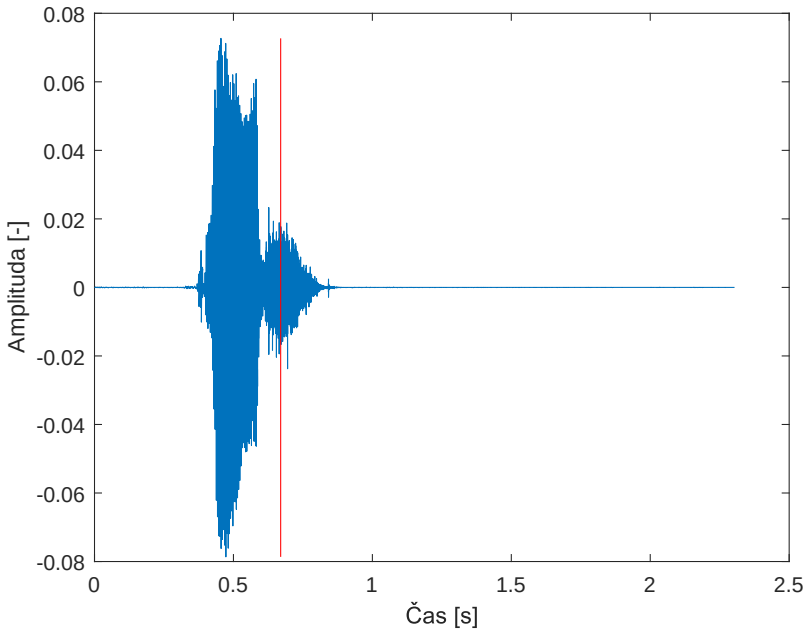
Během testování tohoto modulu však bylo zjištěno nekorektní rozpoznávání povelů. Pro účely testování byly vytvořeny další dva moduly, které se starají o zaznamenávání činnosti.

První z nich se stará o záznam odeslaných úseků. Zaznamenávání probíhá tak, že na začátku inicializace je modulu AudioLogger předána vzorkovací frekvence spolu s požadovaným názvem. Následně při odesílání úseků jsou tyto úseky předány modulu pro uchování v paměti a teprve při příjmu ukončovací zprávy, získané od serveru, jsou úseky uloženy do formátu WAV pomocí knihovny wav-encoder[9].

Druhý modul pro zaznamenávání, CommandLogger, se stará o zachování zpráv přijatých od serveru, což jsou informace o rozpoznaných ruších a povelech. Na začátku inicializace je modulu předán název tak, aby byl jednoznačně spárován audio záznam a k němu náležející informace. Modul následně při přijmu zprávy uloží získané obsahy zpráv do textové souboru TXT.

Po zakomponování do odesilatele AudioSender, z kapitoly 3.3, však bylo

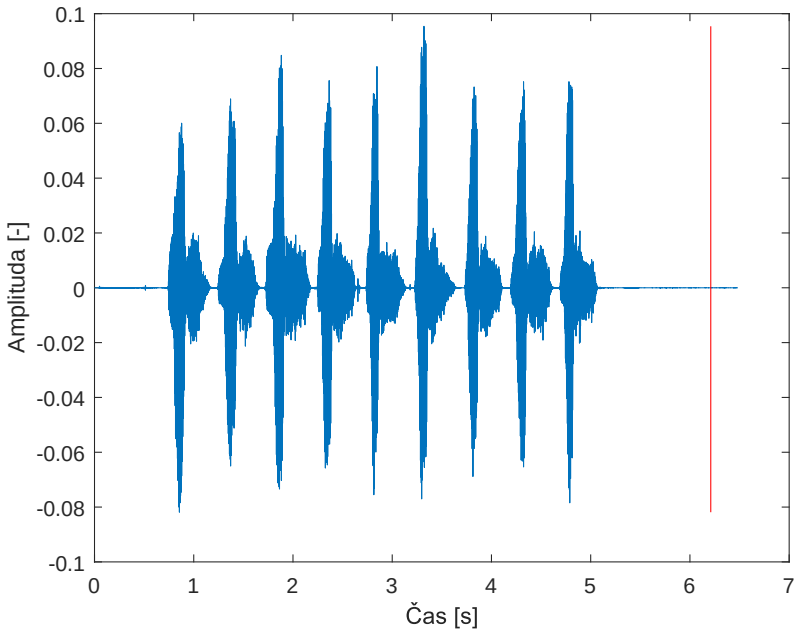
zjištěno, že získané nahrávky jako takové jsou odesílány korektně bez přerušení mezi jednotlivými úseky nebo předčasného ukončení. Při testování na získané nahrávce však byl povel úspěšně rozpoznán. Díky této informaci mohla být modulu CommandLogger následně předávána navíc informace o tom, jak dlouhou část nahrávky server rozpoznával. S touto informací a porovnáním délky audio nahrávky bylo pozorováno, že modul AudioSender neodesílá veškerá data, viz 3.19, zejména při delších nahrávkách.



Obrázek 3.19: Audio nahrávka s ukončením přenosu

Po tomto zjištění vznikla nová verze modulu odesilatele, která neposílá pouze detekované řečové úseky, ale veškeré získané úseky. Nicméně výsledek byl velice podobný předchozímu případu. Proto byl dále sledován modul AudioSender. Při pozorování chování modulu bylo zjištěno, že po různě dlouhé době byla komunikace s rozpoznávacím serverem ukončena. Uvedená pozorování vedla k domněnce, že modul nezískává úseky s dostatečnou rychlostí. Při jednoduchém testu, kdy bylo přidáno čekání po vybranou dobu, byl tento předpoklad potvrzen. Při přidání čekání s dobou

100 ms již k tomuto jevu nedocházelo, ale nastal nový problém. Modul se výrazně zpomaloval a to v závislosti na délce odesílaných úseků. Proto byly přepracovány moduly pro snímání signálu z mikrofonu a odesílání. Do modulu AudioSender byl přidán nový příznak určující, zda detektor stále rozpoznává řečové segmenty. Tento příznak je předán modulem pro snímání. Následně, pokud server požaduje další úseky, ale zároveň je stále detekována řeč, je odeslána odpověď s prázdným vektorem dat.



Obrázek 3.20: Audio nahrávka s ukončením přenosu po úpravě modulu

Pokud by nebyl odeslán prázdný vektor dat, server by poslal zprávu o ukončení rozpoznávání a komunikace by byla přerušena během odesílání dalších dat. Díky této modifikaci je již odesílána kompletní nahrávka, kromě několika posledních klidových úseků, viz obrázek 3.20.

3.4 Vykonavatel hlasového povelu

Po získání rozpoznaných hlasových povelů je nutné naimplementovat modul, který bude dané povely následně vykonávat. Zde je možné opět volit ze dvou způsobů řešení. První z nich je pro dané platformy, Windows, macOS, atd., vytvořit jednoduchý program, který by se spouštěl pomocí funkce v JavaScriptu s patřičným parametrem. Druhý přístup, který byl vybrán jako vhodnější, je využití již existující knihovny a pouze její rozšíření o další funkcionalitu pro potřeby aplikace.

3.4.1 Knihovna RobotJS

Jednou z dostupných knihoven je knihovna RobotJS. Ta umožňuje ovládat kurzor myši, klávesnici a dokonce i snímat obrazovku monitoru. Z dokumentace [8] bylo zjištěno, že pro samotné ovládání kurzoru myši jsou pak důležité funkce „getMousePos“, „moveMouseSmooth“, „mouseClick“ a „getScreenSize“. Jak již názvy funkcí napovídají, jedná se o funkce pro zjištění pozice, vykonání pohybu a vytvoření události tlačítka myši. Pro samotný pohyb kurzoru myši pak existuje ještě alternativní funkce „moveMouse“, která vykonává stejnou činnost, ale kurzor se posune do daného bodu okamžitě a ne, jako v případě funkce „moveMouseSmooth“, postupně.

Jelikož jsou tyto operace blokující, tedy po zavolání dané funkce se musí čekat na její dokončení, vyvstává zde otázka, kterou z nich je vhodnější použít. Při využití okamžitého přesunu, v případě „moveMouse“, nedochází k zablokování aplikace na tak dlouhou dobu, jako v případě „moveMouseSmooth“, ale uživatel nebude moci sledovat plynulý pohyb kurzoru do příslušného místa. To může vést ke zmatení uživatele a snížení uživatelského komfortu.

Co je však nutné podotknout, je fakt, že tato knihovna podporuje pouze ovládání pro hlavní monitor počítače. V případě, že tedy bude kurzor na jiném monitoru než hlavním, není možné jej takto ovládat.

3.4.2 Struktura povelů

Po seznámení se s knihovnou RobotJS bylo možno přistoupit k návrhu struktury povelů. Jelikož původní verze programu MyVoice měla rozděleny příkazy do skupin podle toho, co chtěl uživatel dělat, bylo i zde vytvořeno obdobné uspořádání. Proto byla navržena stromová struktura, která tyto požadavky nejlépe vystihuje.

Základem je abstraktní třída AbstractController, kterou budou následně další třídy reprezentující skupinu povelů implementovat. Abstraktní třída obsahuje dvě důležité metody, „doOperation“ a „getPossibilities“. Funkce „doOperation“ prochází strukturu do hloubky a teprve v poslední skupině testuje veškeré dostupné povely. Při průchodu strukturou zpět se testují již jen návratové povely pro zpětný pohyb mezi skupinami. Funkce „getPossibilities“ funguje velice obdobně, ale nevykonává žádný povel. Tato funkce pouze vrací veškeré možnosti aktivní skupiny a návratové povely.

Byl vytvořen slovník povelů dle pravidel pro přepis z grafémové podoby do fonetické dle [1]. Byly tak vytvořeny první dvě skupiny povelů, základní skupina a skupina myš. Základní skupina je pouze kořenem celé struktury a obsahuje tedy skupinu myš jako možný povel. Pro skupinu myš bylo vytvořeno prvních 17 povelů a to pro pohyb myši o různě velkou vzdálenost, kliknutí myší, pravé tlačítko myši a dvojitý klik.

4 Popis vytvořeného uživatelského rozhraní

Pro vytvoření grafického zobrazení aplikace je možné využít některou z veřejně dostupných knihoven. Takovýchto knihoven je dnes větší množství a je tedy jen na programátorovi, jakou z nich si vybere. V této kapitole budou popsány nejznámější z veřejně dostupných knihoven.

4.1 Angular

Jako první možnost se nabízí knihovna Angular. Ta je vyvíjena společností Google. Jedná se o knihovnu pro vytváření jednostránkových aplikací implementující návrhový vzor MVC. Lze tedy vytvořit šablonu pomocí značkovacího jazyka HTML a pomocí speciálního zápisu ji dynamicky doplňovat.

Veškerá tvorba začíná vytvořením standardní HTML struktury. Co je však rozdílné, je to, že již v elementu „html“ je použit atribut „ng-app“ identifikující aplikaci. Obdobně je tomu s elementem „body“, který představuje kontrolér a má tedy atribut „ng-controller“. S takto připraveným dokumentem již lze pracovat pomocí knihovny Angular.

Co bylo však zjištěno při testování, je to, že při použití této knihovny nemusí dojít ke správnému importu knihoven. To bylo pozorováno například s knihovnou keras-js sloužící k použití natrénované sítě v knihovně jazyka Python Keras.

Z popsaného důvodu byla vybrána jiná knihovna, a sice Electron, pro vytvoření grafické uživatelské aplikace pro zajištění bezproblémového importu knihoven a rychlejší vývoj.

4.2 Electron

Vhodnou alternativou je knihovna Electron. Ta byla původně určena pro editor Atom, avšak s přibývající popularitou se v této knihovně začaly vytvářet i další aplikace. Pomocí ní tak vznikly např. aplikace jako Skype nebo Visual Studio Code.

Prvotní struktura aplikace je vždy rozdělena do tří souborů:

1. package.json
2. main.js
3. index.html

Základní informace o názvu, verzi, apod. obsahuje soubor „package.json“, ve kterém jsou zároveň uchovány potřebné závislosti na další knihovny. O vytvoření okna aplikace a jejího spuštění se pak stará soubor „main.js“. V tomto souboru jsou definována jednotlivá okna aplikace, jejich základní vzhled, události, ale také zachycování asynchronních zpráv z jednotlivých oken aplikace. Naproti tomu soubor „index.html“ již obsahuje informaci o prvcích okna, jejich umístění, stylizaci pomocí CSS apod.

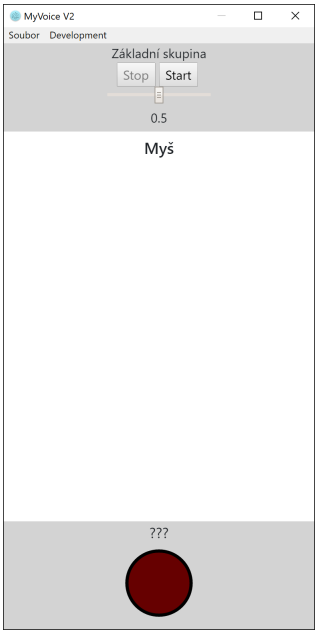
4.3 Vytvoření aplikace pomocí knihovny Electron

Nejprve bylo nutno definovat vytvoření požadovaného uživatelského okna aplikace. Dle dokumentace [3], k tomu slouží objekt BrowserWindow. Jelikož není žádoucí, aby bylo možno skrýt okno nebo ho překrýt jinou aplikací, je nutné nastavit potřebné proměnné, tj. AlwaysOnTop a Minimizable. Protože je nutné zadat uživatelské jméno a heslo pro přístup k rozpoznávajícímu serveru, byla přidána další definice pro okno obsahující nastavení.

Podle toho, zda se jedná o vývojářskou nebo zákaznickou verzi se zároveň přidá možnost zobrazit další dvě okna. Okno s nastavením parametrizace, segmentace, apod. a rozhraní pro odesílání souborů.

Pro zachování stejného uživatelského komfortu, byla vizuální podoba navržena co nejpodobněji původní verzi programu MyVoice, tedy přehled aktuálně dostupných povelů, aktuální skupina a tlačítka pro spuštění, resp. zastavení, snímání mikrofonu. Dále byl přidán vizuální prvek indikující aktivitu detektoru a nastavení prahu pro konečný automat s klouzavým průměrem z kapitoly 3.2.

Pro prvek reprezentující nastavení prahu byl použit posuvník spolu s událostí „change“, pomocí které se zobrazí aktuální hodnota posuvníku, resp. prahu konečného automatu. Pro tlačítka byla využita událost „click“, ve které se již využije funkce z nového modulu „renderer“. Výsledná podoba aplikace je tedy následující:



Obrázek 4.1: Základní vzhled aplikace

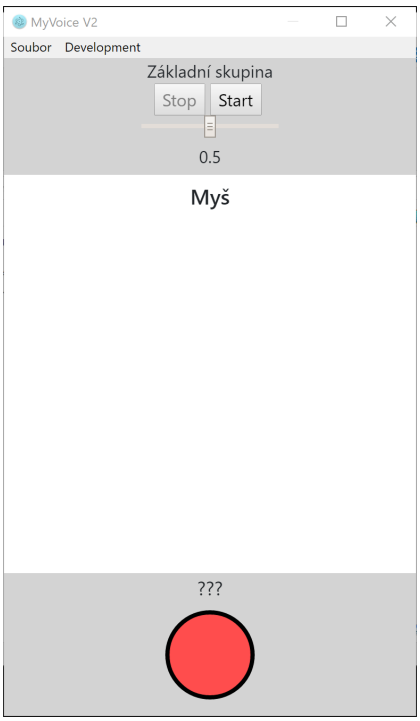
Jak již bylo předestřeno, vznikl nový modul, který vytváří provázání mezi dosud vytvořenými moduly. V událostech tlačítek se následně použijí funkce „startRecording“, resp. „stopRecording“. Aby bylo snímání co možná nejobecnější, bylo nutno předat funkci modulu snímajícím mikrofon. Tato funkce budu zpracovávat získané časové úseky. Popsaná funkce vznikla využitím již existujících funkcí z knihovny pro přípravu signálu a pouze se aplikují jednotlivé funkce na získaná data.

Takto zpracovaná data jsou vstupem, jak již bylo řečeno, objektu ShiftBuffer pro získání úseku o konkrétní délce.

Protože je třída ShiftBuffer implementována obecně, očekává také jako svůj parametr funkci, kterou se bude zpracovávat úsek signálu. Bylo tedy nutné vytvořit další funkci, ve které probíhá aplikování okénka, vypočtení MelFBank příznaků a transformace.

Jelikož je následujícím krokem získání určitého počtu těchto příznakových vektorů, byla zde opět využita třída ShiftBuffer, tentokrát s jinou zpracovávající funkcí. Tato funkce již používá získaná data jako vstup neuronové sítě. Popsaný výstup je následně vyhlazen konečným automatem s klouzavým průměrem a dochází k určení, zda se jedná o řečový či klidový úsek.

Při detekci řečového úseku dojde k vytvoření nového objektu typu AudioSender, kterému se předá část uchovaných dat pro korektní rozpoznání na straně serveru a kompenzace časové prodlevy. Zároveň, jelikož došlo k detekci řečových úseků, dojde k vizuální signalizaci uživateli. Tuto signalizaci lze pozorovat na následujícím obrázku:



Obrázek 4.2: Signalizace detektoru řeči

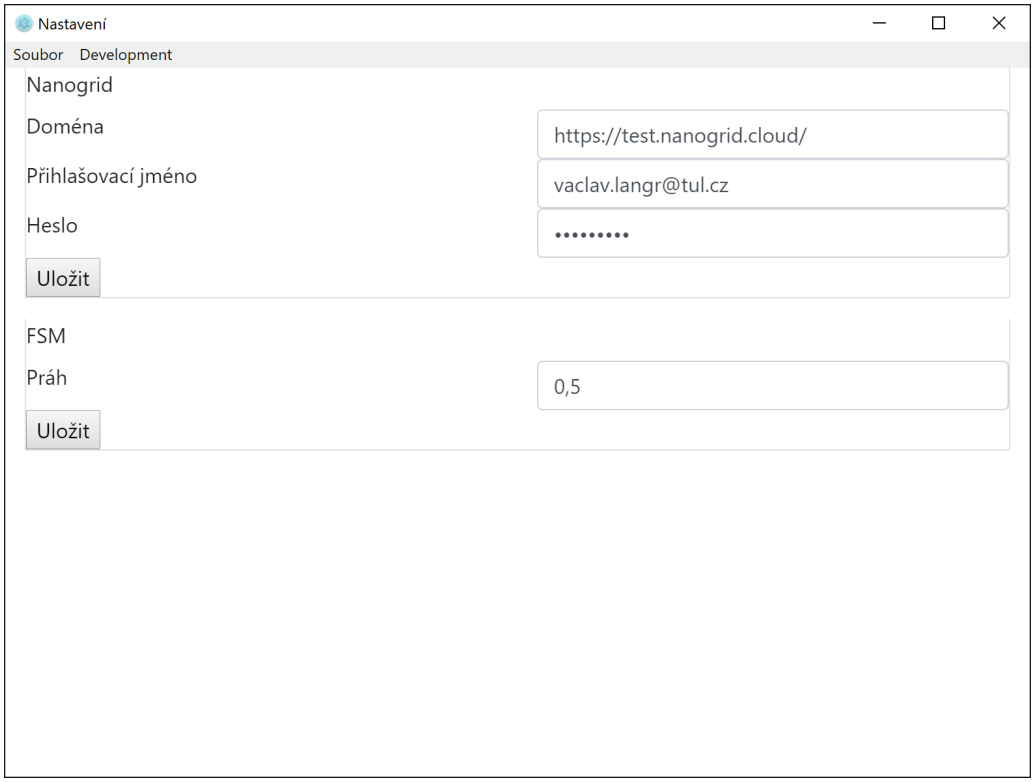
Od tohoto momentu již nedochází k uchovávání několika málo posledních úseků a jsou předávány přímo odesilateli a také parametrizaci pro detekci ukončení hlasového povelu. Následuje odeslání již získaných úseků na server a zachycení příchozích zpráv.

Pro aktualizaci možných povelů byla vytvořena třída, MainController implementující AbstractController. Připravené funkce fungují obdobně, ale je zde zpracováván výstup funkce „doOperation“ určující, zda byl povel vykonán. To bylo provedeno z toho důvodu, aby ovládání uživatelského rozhraní nebylo součástí abstraktní třídy a bylo tak možné ji použít i v jiném projektu než tomto. Aktualizace uživatelského rozhraní probíhá pak tak, že je změněn text elementu zobrazující poslední příkaz, byl-li vykonán a také elementu zobrazujícího aktuální skupinu. Ovšem s takto vytvořenou funkcí pro aktualizaci uživatelského rozhraní by koncový uživatel neviděl, jaké povely může dále vykonávat. Proto funkce navíc při změně skupiny maže seznam povelů a pro každý možný povel přidá nový element, např. typu „h5“ s dostatečným zvýrazněním, s textem povelu. Třída MainController je pak použita jako kořen celé struktury. Ukázku aktualizace grafického uživatelského rozhraní lze pak vidět na následujících obrázcích:

Obrázek 4.3: Základní okno aplikace

Obrázek 4.4: Aktualizované okno aplikace

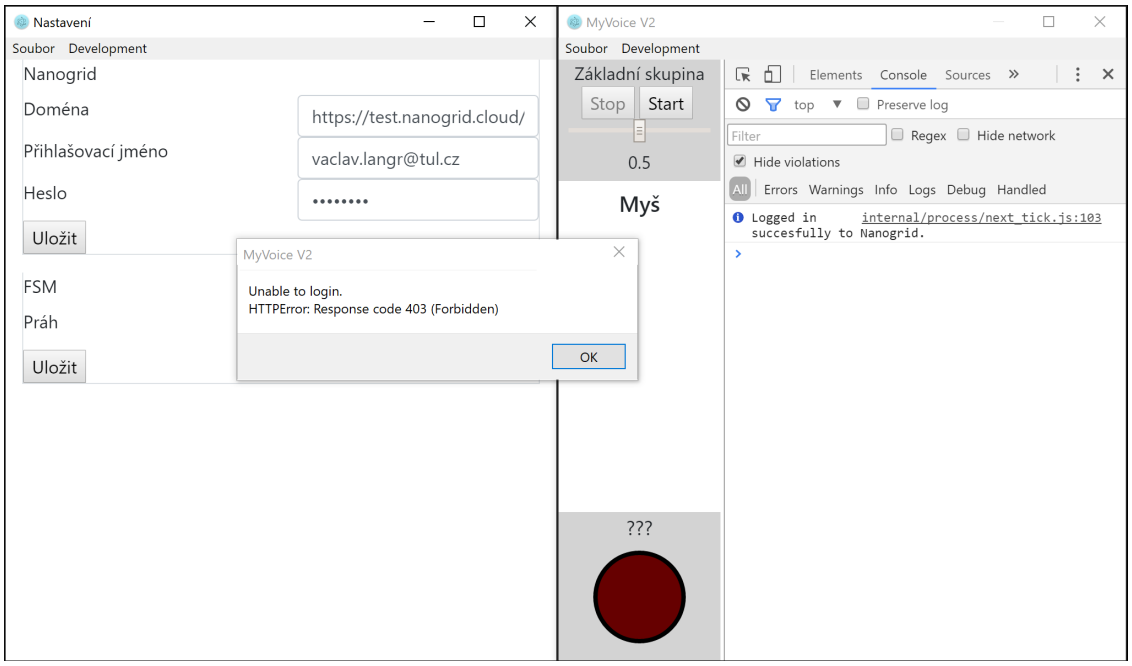
Dalším oknem je uživatelské nastavení. To obsahuje informace o doméně, přihlašovacím jméně a hesle, ale také prahu, který si může každý uživatel nastavit podle svých potřeb. Nicméně zde vznikla nutnost perzistentního ukládání nastavení. Právě proto, že toto knihovna Electron neumožňuje, byla použita další knihovna a to electron-store. Ta umožňuje na jakémkoliv místě vytvořit soubor typu JSON a do něj ukládat údaje. Zároveň obsahuje možnost šifrování tohoto souboru a zamezení tak čtení a úpravy uživatelem. S touto knihovnou tedy vznikl další modul a to konfigurační vytvořený podle [5]. Díky tomu, bylo možné nahradit pevně nastavené hodnoty jednoduchým voláním příslušné funkce v tomto modulu.



Obrázek 4.5: Okno uživatelského nastavení

S takto připraveným modulem již lze vytvořit okno uživatelského nastavení. Při prvotním vytvoření okna se koncovému uživateli zobrazí tak, jako je na obrázku 4.5. Zbývá jen provázání událostí tlačítek s potřebnými funkcemi pro uložení dat.

V případě změny hodnoty prahu pro konečný automat s klouzavým průměrem je toto velmi jednoduché. Uložení se provede pouze zavoláním příslušné funkce v konfiguračním modulu. Ovšem v případě nastavení domény, přihlašovacího jména nebo hesla nelze takto postupovat. Při zadání nesprávné informace, např. nesprávného uživatelského jména nebo hesla, by uživatel nebyl nijak upozorněn a při příštím spuštění aplikace by to vedlo k nefunkčnosti aplikace. Proto se při události tlačítka zároveň odesílá požadavek o autentizační a autorizační tokeny. Podle odpovědi je pak buď uživateli zobrazeno upozornění na nesprávnost jména či hesla a nebo se při úspěšném přihlášení vypíše jen informační zpráva do vývojářské konzole, což se využívá např. při dalším vývoji.



Obrázek 4.6: Ukázka přihlášení uživatele

Zde se ale můžeme setkat s další komplikací. Při žádosti, která je asynchronní, a následném zavření okna nastavení, dojde k přerušení komunikace se serverem. Proto je nutné předat zprávu o události jiné komponentě. K tomuto předání zprávy existuje v knihovně Electron třída ipcRenderer, která asynchronně předá zprávu

hlavní části aplikace s identifikátorem a případně dalšími parametry. V hlavní části je pak použita komponenta ipcMain z knihovny pro příjem zprávy. Nicméně při žádosti z hlavní části aplikace dojde k zablokování celé aplikace po dobu vykonávání žádosti. Tedy ani toto není dostačující a vyhovující řešení. Nicméně existuje ještě další komponenta, jíž lze využít a to hlavní okno aplikace. Zde je postup velice obdobný, z hlavní části aplikace je odeslána další zpráva, tentokrát ale s jiným identifikátorem, která se přijme v hlavním okně aplikace opět pomocí ipcRenderer. Tím je zajištěno vykonání přihlášení po delší dobu a je tedy méně pravděpodobné přerušení přihlašování. Po úspěšném přihlášení jsou zároveň uloženy autentizační a autorizační tokeny pro další využití.

4.4 Distribuce aplikace

Jak bylo řečeno v kapitole 1, aplikace by měla být funkční pod různými operačními systémy. Tak jak je vytvořená aplikace popsána do kapitoly 4.3 by nebyla vhodná ke spouštění na počítači koncového uživatele tak, jak je zvyklý v klasických aplikací. Bylo by nutno instalovat další podpůrné programy. Toto bude vyeliminováno zabalením aplikace a vytvořením instalátoru.

4.4.1 Zabalení aplikace

Prvním krokem k distribuci aplikace je zabalení do spustitelné podoby. To probíhá pomocí knihovny Electron packager. Proto byla přidána reference na tuto knihovnu do souboru „package.json“, jež byl zmíněn v kapitole 4.

Jako první byl zvolen operační systém Windows. Pro ten byl nejprve připraven základní příkaz definující platformu, architekturu, název aplikace spolu s přepínačem signalizujícím zabalení kompletního projektu do souboru „app.asar“. Avšak takto připravený příkaz by výsledek uložil do stejného místa, jako je celý projekt a bylo by tedy obtížné oddělit, co je součástí projektu a co součástí zabalené aplikace. Proto byla definována také cesta, kam bude výsledek uložen spolu s přepínačem signalizujícím přepis staré verze. Ačkoliv je toto dle dokumentace [11] správný po-

stup, při spuštění aplikace byla objevena řada chyb týkajících se špatného importu podpůrných knihoven. Při bližším zkoumání souboru „app.asar“ bylo zjištěno, že i přes kompletní zabalení projektu do tohoto souboru došlo k odstranění veškerých knihoven potřebných pro korektní fungování aplikace. Při procházení dokumentace bylo následně zjištěno, že s takto připraveným příkazem dochází k vynechání nepotřebných knihoven z projektu pro optimalizaci velikosti aplikace. Byl tedy přidán parametr „--no-prune“ zamezující této optimalizaci. Po použití uvedeného příkazu je již aplikace korektně zabalena pro spuštění bez dalších podpůrných prostředků.

Pro zabalení aplikace pro další platformy již je možné použít stejný příkaz, avšak s patřičnou úpravou architektury a platformy.

4.4.2 Vytvoření instalátoru

Pro instalaci multiplatformní aplikace u koncového uživatele je vhodné vytvořit instalátor. Proto byly využity další knihovny, které vytvoří požadované instalátory ze zabalených verzí aplikace.

Jako první operační systém, pro který byl vytvořen instalátor, je systém Windows. Pro účely vytvoření instalátoru byla použita knihovna electron-winstaller. Ta vyžaduje vytvoření skriptu, ve kterém je funkce obsahující parametry, jako jméno autora, výstupní složka nebo název instalátoru. Poté již lze pouze spustit tento skript pomocí jednoduchého příkazu. Nicméně s takto připraveným instalátorem nedojde např. k vytvoření zástupce na ploše, naopak aplikace se spustí při samotné instalaci, ale i odinstalaci, apod. Pomocí dokumentace [12] bylo zjištěno, že s touto knihovnou je nutné zachytit v hlavní části aplikace i nové události. Při jednotlivých fázích dochází k předání některého z parametrů aplikaci a právě pomocí těchto parametrů lze ovlivnit chování aplikace. Tedy např. vytvořit zástupce na ploše uživatele během instalace nebo aktualizace aplikace.

Dalším vybraným operačním systémem byl systém macOS. Pro vytvoření instalátoru pro tento systém byla využita knihovna electron-installer-dmg. Zde je postup velice jednoduchý, pouze byl dle dokumentace [14] sestaven příkaz, který vytvoří instalátor. Příkaz se skládá z:

1. Umístění zabalené aplikace
2. Názvu instalátoru
3. Výstupní složky
4. Příznaku pro přepis již existujícího instalátoru

Jako poslední byl vybrán operační systém Debian a další, které z uvedeného vycházejí. Zde byla použita knihovna electron-installer-debian[13]. U tohoto operačního systému je postup obdobný jako v případě vytvoření instalátoru pro systém Windows. Nejprve byl vytvořen soubor typu JSON, definující vlastnosti jako je název, výstupní složka, apod. S takto připraveným skriptem již bylo možné použít příkaz s parametry jako je: umístění aplikace, konfigurace.

5 Experimentální vyhodnocení

5.1 Vyhodnocení detektoru

Experimentálního vyhodnocení detektoru řeči z kapitoly 3.2 bylo provedeno ve spolupráci s Ústavem Informačních technologií a elektroniky. K dispozici byla data tvořená televizními a rádiovými zpravodajstvími v různých jazycích o celkové délce 7 hodin. Uvedená data byla tvořena cca ze 70 % řečí, zbytek nahrávek byl tvořen hudbou, znělkami nebo reklamami.

Pro vyhodnocení slouží několik metrik. První z nich je procentuální neúspěšnost označená zkratkou „FER“ s předpisem:

$$FER = \frac{počet\ nekorektně\ detekovaných}{celkový\ počet} \cdot 100 \qquad \text{(Vzorec 4)}$$

Další z metrik slouží pro vyjádření chybovosti detekce řečových segmentů. Ta má velice podobný předpis jako první metrika, nicméně je zde zásadní odlišnost, používá se jen počet nekorektně detekovaných řečových úseků ku celkovému počtu řečových úseků. Pro odlišení byla této metrice přiřazena zkratka „MR“.

$$MR = \frac{počet\ nekorektně\ detekovaných_{\text{řeč}}}{celkový\ počet_{\text{řeč}}} \cdot 100 \qquad \text{(Vzorec 5)}$$

Poslední z použitých metrik, „FAR“,je procentuální vyjádření chybovosti detekování klidových úseků jako řečových:

$$FAR = \frac{počet\ nekorektně\ detekovaných_{\text{klid}}}{celkový\ počet_{\text{klid}}} \cdot 100 \qquad \text{(Vzorec 6)}$$

Autorovi byly poskytnuty celkové výsledky úspěšnosti, zároveň byly poskytnuty i výsledky ukazující, v jaké míře došlo k chybné klasifikaci řečových úseků jako klidových, resp. klidových úseků jako řečových.

Při vynechání vyhlazovacího konečného automatu z kapitoly 3.2.2 bylo experimentálně zjištěno, že je detekující neuronová síť neúspěšná v $4,7$ %. V $3,7$ % došlo k chybné klasifikaci řečových úseků jako klidových, v $7,1$ % byly mylně klasifikovány klidové úseky jako řečové.

Dále probíhalo experimentální ověření s použitím vyhlazujícího automatu. Zde došlo ke zlepšení detektoru s celkovou neúspěšností $2,9$ %. V případě chybné klasifikace řečových úseků došlo ke zlepšení na $2,2$ %. V chybné klasifikace klidových úseků jako řečových došlo ke zlepšení na $4,7$ %.

5.2 Vyhodnocení rozpoznávače

Současně došlo k experimentálnímu ověření rozpoznávače. Celkem bylo nashromážděno 10 nahrávek vybraných povelů od několika uživatelů. Bylo tak nasbíráno přes 500 krátkých nahrávek, obsahujících vždy pouze jediný povel. Mezi vybrané povely patří:

1. Myš
2. Základní skupina
3. Klikni
4. Pravý klik
5. Dvojklik
6. Doleva 500
7. Doprava 100
8. Dolů 50

9. Nahoru 50

Výsledky jsou pak znázorněny následující maticí záměn:

Skutečný \ Rozpoznaný	1	2	3	4	5	6	7	8	9	šum
1	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	10%	80%	0%	0%	0%	0%	0%	10%
5	0%	0%	20%	0%	80%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	90%	0%	0%	0%	10%
7	10%	0%	0%	0%	0%	0%	80%	0%	0%	10%
8	0%	0%	0%	10%	0%	0%	0%	90%	0%	0%
9	0%	0%	0%	0%	0%	0%	10%	0%	90%	0%

Při detailnějším zkoumání získaných nahrávek bylo zjištěno, že u některých z nich došlo např. k přeřeknutí nebo odkašlání uživatele. Celková úspěšnost rozpoznávání je tedy 90 %.

6 Závěr

Jak je patrné z této dokumentace, diplomová práce byla rozdělena do několika na sebe navazujících kroků. Postupně tak byly vytvořeny jednotlivé moduly pro:

1. Snímání mikrofonu: Zásadní změnou bylo nahrazení knihovny pokročilejší verzí tak, aby umožňovala nastavení konkrétních parametrů pro snímání signálu z mikrofonu.

2. Parametrizaci: Byl vytvořen modul, který aplikuje Hammingovo okénko, sestavuje Melovy trojúhelníky a získává vektor příznaků z úseku signálu.

3. Načtení neuronové sítě ze souboru: Jedná se o konstrukci podpůrného modulu, který je využíván pro načtení neuronové sítě k dalšímu využití.

4. Dopředný průchod neuronové sítě, detekující řečovou aktivitu: Nejdůležitější bylo implementování dopředného průchodu tak, aby bylo možno volit aktivační funkci libovolně dle typu sítě.

5. Vyhlazení výstupu neuronové sítě pomocí konečného automatu: Daný modul vyřešil problémy související s kolísáním výstupu z neuronové sítě a detekováním začátku promluvy.

6. Zastavující konečný automat detekující konec řeči: Zde byl vyřešen požadavek na správné detekování ukončení řeči. Sestavený modul funguje na dynamické bázi tak, že sestavuje automat podle zadaného parametru, tím je počet klidových úseků pro ukončení detekování.

7. Odesílání řečových úseků na server: Konstrukce tohoto modulu vyžadovala složité testování, které ukázalo na problém se synchronizací se serverem a

nutnost ošetření chování v jednovlákenném zpracování. Synchronizace byla vyřešena tak, že v případě potřeby byla data doplněna o prázdné úseky tak, aby bylo vyhověno žádosti serveru o další data.

8. Vykonání rozpoznaného povelu: Zde byl nejdůležitější správný návrh struktury dosud vytvořených povelů tak, aby bylo umožněno zobrazení pomocí grafického uživatelského rozhraní a zároveň vykonávání povelů. Nejvýhodnější se ukázala stromová struktura povelů, která splňuje veškeré požadavky.

9. Zaznamenávání přijatých zpráv: Modul, jenž uloží přijaté zprávy ze serveru do textové podoby. Ten funguje na principu procházení přijatých zpráv a pouze při detekování zvoleného typu zprávy ji uloží.

10. Zaznamenávání odeslaných řečových úseků: Poslední vytvořený modul slouží k zaznamenávání odeslaných řečových úseků do podoby souboru typu WAV. Byl využíván při testování chování aplikace a odstraňování nedostatků.

Při testování byly jednotlivé moduly upraveny tak, aby aplikaci bylo možné používat v reálných podmínkách a uživatel nemusel čekat po dlouhou dobu, než je povel rozpoznaný.

6.1 Přínosy práce

Během vypracovávání této diplomové práce vznikly celkem tři verze prototypu. Kromě finální verze, která komplexně zpracovává veškeré dané požadavky, vznikala současně i verze, která fungovala bez použití detektoru řeči a tedy docházelo k neustálému odesílání snímaných signálů z mikrofonu. Tato verze se ukázala jako neefektivní vzhledem k velikosti odesílaných dat. Jak je patrno z kapitoly 3.1.1, další verze pracovala s využitím prostředků JavaScriptu pro snímání mikrofonu. Zároveň všechny tři popsaná vypracování jsou rozdělena podle toho, zda se jedná o vývojářskou nebo zákaznickou verzi. Vývojářská verze navíc umožňuje zaznamenávání snímaného signálu do souboru a příchozích zpráv ze serveru. Tento prototyp je navíc multiplatformní a umožňuje jednoduchou údržbu. Využitím výhody

spojitého zpracování je umožněno zadávání více povelů než v aktuální generaci programu MyVoice.

6.2 Další rozvoj aplikace

Pro získání distribuovatelné podoby vhodné pro plnohodnotné použití je nutnost přidání modulu umožňujícího plnou konfigurovatelnost, tak jak to obsahuje program MyVoice. Nutností je tak vznik více skupin, než aktuální prototyp obsahuje. Ty budou umožňovat například ovládání klávesnice, spouštění dalších programů apod. Zároveň bude žádoucí vytvořit nápovědu programu, i s demonstračními videi, pro vysvětlení odlišností aktuálním uživatelům programu MyVoice.

Jako první rozšíření této diplomové práce se nabízí přidání možnosti pro skrytí okna aplikace do systémové lišty a při rozpoznání povelu uživateli pouze zobrazit upozornění o provedení povelu. Vhodné by to bylo zejména pro pokročilé uživatele, kteří již mají zkušenost se stávající generací aplikace MyVoice jako jediným prostředkem pro ovládání osobního počítače a tedy vědí, jaké povely jsou v dané skupině dostupné.

Další možnou modifikací se jeví vytvoření rozšiřujícího modulu, který by se staral o vytváření struktury skupin a povelů uživatelem, viz kapitola 3.4. Zároveň by bylo vhodné daný modul upravit o načtení této struktury pro docílení výrazně větších možností použití, než je v aktuální verzi zohledněno.

Logickým pokračováním se pak jeví zacílení na mobilní zařízení, kde by uživatel neviděl přímo seznam povelů, ale např. na vybraný povel by se zobrazilo upozornění, jež by obsahovalo povely dané skupiny. Jistě se najde ještě řada dalších možností, jak aplikaci vylepšit, upravit apod.

Literatura

[1] EDITOŘI JAN NOUZA, Zbyněk KOLDOVSKÝ a Robert VÍCH. Řeč a počítač: principy hlasové komunikace, úlohy, metody a aplikace : sborník článků. Liberec: Technická univerzita v Liberci, 2009. ISBN 8073725487.

[2] ELLIOTT, Eric. Programming JavaScript applications. Sebastopol: O'Reilly, 2014. ISBN 1491950293.

[3] Documentation — Electron. Electron — Build cross platform desktop apps with JavaScript, HTML, and CSS. [online]. Dostupné z: https://electronjs.org/docs

[4] API Docs - NanoGrid API. [online]. Dostupné z: https://nanotrix.gitlab.io/nanogrid-docs/

[5] GitHub - sindresorhus/electron-store: Simple data persistence for your Electron app or module - Save and load user preferences, app state, cache, etc. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/sindresorhus/electron-store

[6] GitHub - sindresorhus/got: Simplified HTTP requests. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/sindresorhus/got

[7] RecordRTC: Class: RecordRTC. RecordRTC: Index [online]. Copyright © [cit. 03.05.2018]. Dostupné z: http://recordrtc.org/RecordRTC.html

[8] Documentation - RobotJS. RobotJS - Node.js Desktop Automation [online]. Dostupné z: http://robotjs.io/docs/

[9] GitHub - mohayonao/wav-encoder: promise-based wav encoder. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/mohayonao/wav-encoder

[10] JavaScript — MDN. [online]. Copyright © 2005 [cit. 03.05.2018]. Dostupné z: https://developer.mozilla.org/bm/docs/Web/JavaScript

[11] GitHub - electron-userland/electron-packager: Customize and package your Electron app with OS-specific bundles (.app, .exe, etc.) via JS or CLI. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/electron-userland/electron-packager

[12] GitHub - electron/windows-installer: Build Windows Installers for Electron apps. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/electron/windows-installer

[13] GitHub - unindented/electron-installer-debian: Create a Debian package for your Electron app.. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/unindented/electron-installer-debian

[14] GitHub - mongodb-js/electron-installer-dmg: Create DMG installers for your electron apps using appdmg.. The world's leading software development platform · GitHub [online]. Copyright © 2018 [cit. 03.05.2018]. Dostupné z: https://github.com/mongodb-js/electron-installer-dmg

[15] Docs — Node.js. [online]. Copyright © Node.js Foundation. All Rights Reserved. Portions of this site originally [cit. 03.05.2018]. Dostupné z: https://nodejs.org/en/docs/

[16] SCHMIDHUBER, Jürgen. Deep learning in neural networks: An overview. Neural Networks. 2015, 61, 85-117. DOI:

10.1016/j.neunet.2014.09.003. ISSN 08936080. Dostupné také z: http://linkinghub.elsevier.com/retrieve/pii/S0893608014002135

[17] MyVoice. [online]. Copyright © Speechlab. All Rights Reserved. [cit. 05.05.2018]. Dostupné z: https://www.ite.tul.cz/speechlab/index.php/myvoice.html

Přílohy

Příloha A: Obsah přiloženého CD

Součástí této diplomové práce je CD obsahující kompletní technickou dokumentaci a zdrojové kódy, které vznikly během zpracování práce.

- **Dokumentace.pdf** - Technická dokumentace ve formátu PDF
- **src** - Složka obsahující zdrojové kódy
- **install** - Složka obsahující instalátory
  - TUL_DP.dmg - Instalátor pro operační systém macOS
  - TUL_DP_Installer.exe - Instalátor pro operační systém Windows