České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství

Hlasem ovládaný robot

Voice controlled robot

BAKALÁŘSKÁ PRÁCE

Vypracoval: Michal Kusý
Vedoucí práce: Ing. Josef Nový, Ph.D.
Rok: 2023

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství Akademický rok 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Michal Kusý**
Studijní program:	**Aplikace přírodních věd**
Obor:	**Aplikace softwarového inženýrství**
Název práce česky:	**Hlasem ovládaný robot**
Název práce anglicky:	**Voice controlled robot**

Pokyny pro vypracování:

1. Nastudujte problematiku rozponávání hlasu.
2. Seznamte se s platformou NVIDIA Jetson a Jetbot.
3. Navrhněte ovládací software robota využívající řízení hlasem.
4. Implementujte navržený software.
5. Implementovaný systém otestujte.

Doporučená literatura:

[1] L. Joseph, *Learning Robotics Using Python.* Packt Publishing, 2015. ISBN 1783287543

[2] A. Kurniawan, *Getting Started with NVIDIA Jetson Nano.* PE Press, 2019

[3] F. Chollet, *Deep learning v jazyku Python: knihovny Keras, Tensorflow.* Grada Publishing, a.s., 2019. ISBN 8024731002

Jméno a pracoviště vedoucího práce:

Ing. Josef Nový, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

Jméno a pracoviště konzultanta:

–

.....
vedoucí práce

Datum zadání bakalářské práce: 5. 10. 2022

Termín odevzdání bakalářské práce: 5. 1. 2023

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

.....
děkan

V Praze dne 5.10.2022

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Děčíně dne ....

Michal Kusý

Poděkování

Děkuji Ing. Josefu Novému za zapůjčení robota a zaopatření mikrofonu, aby se tato práce dala uskutečnit.

Michal Kusý

Název práce:
Hlasem ovládaný robot

Autor: Michal Kusý

Studijní program: Aplikace přírodních věd
Obor: Aplikace softwarového inženýrství
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Josef Nový, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze
Konzultant:

Abstrakt: Tato práce se zabývá hlasovým ovládáním pojízdného robota. Cílem práce bylo navrhnout ovládací software robota, tak aby poslouchal hlasové příkazy a pohyboval se řečeným směrem. Ovládací software robota byl vytvořen v Jupyter notebooku dvěma způsoby. Jednou pomocí neuronové sítě od Google API, zatímco podruhé pomocí lokální neuronové sítě. Nahrávání zvuku z mikrofonu zajistil bash skript volaný jako samostané vlákno. V teoretické části práce je popsáno fungování pripcipu neuronové sítě, o který se obě implementace opírají. Ovládací software podporuje základní směrové příkazy, které jsou zmíněny v závěru. V práci je také rozebrán větší vliv okolního šumu a problémy spojené s tímto šumem.
Klíčová slova: Robot ovládaní hlas neuronová síť

Title:
Voice controlled robot

Author: Michal Kusý

Abstract: This thesis is focused on voice control of a mobile robot. The objective was to design a controller software, which is able to identify voice commands and begin moving in said direction. The controller software has been coded in Jupyter based platform using a neural network from Google API and a second implementation using a local neural network. Audio is recorded from bash console called by a separate thread. The theoretical part describes the functioning of the neural network, used in both implementations. The program supports basic czech movement directions as commands, some of which are listed in the conclusion. The thesis is also talking about the effect of background noise on the neural network and its problems.
Key words: Robot control voice neural network

Obsah

Úvod

Tématem této práce je robot ovládaný hlasovými příkazy. Téma jsem si vybral, jelikož mě zajímá robotika a neuronové sítě. V této práci jsem se nejdříve zaměřil na vysvětlení na jakých frekvencích operuje lidský hlas a jeho snímání. Zahrnul jsem i úvod do neuronových sítí a popis sítě RNN a RNN-T. Součástí výběru a návrhu neurové sítě bylo prozkoumání dostupných API a jejich funkcí. Ve druhé kapitole je popsán robot Jetbot využitý v této práci a také počítač od NVIDIA, který robota řídí. V té samé kapitole se také zmíním o knihovně Jetbot pro Python. Tato knihovna poskytuje příkazy, kterými je možné ovládat robota jednoduchými funkcemi. V další kapitole bude probrán návrh ovládacího software ve formách lokální sítě a použití Google API. Implementace nahrávání pomocí 2 alternujících vláken, aby se zabránilo ztrátě slov. Také je zahrnuto řešení spojování nahrávek z obou vláken. Využití Google API a jaký výstup jeho neuronová síť poskytuje. V práci je také zmíněno, jak jsem zpracoval výstup neuronové sítě, který poskytlo Google API, i vyhodnocení řetězce slov do příkazu. V kapitole s popisem implementace je také popsáno nasazení neuronové sítě, její předpokládáné fungování a její trénink pomocí předem nahraných nahrávek.

Kapitola 1

Problematika rozpoznávání hlasu

Ve světě domácí i industriální IT techniky se rozvíjí produkty postavené na hlasovém ovládání, například u domácích produktů najdeme domácí chytré asistenty, na které promluvíte a oni vykonají vaši řečenou žádost. V dnešní době tímto způsobem v chytrých domácnostech lze ovládat většina elektrických spotřebičů se zabudovanou podporou internetu, lze tímto způsobem i ovládat osvětlení a i topení. V industriální sféře se může jednat o integrace systémů zahrnující ovládání robotů nebo i hledání dokumentů. Toto všechno je možné ovládat hlasem člověka, je-li tato možnost integrována.

Lidský hlas se skládá ze zvukových vln vytvořených v hlasivkách. Tyto zvuky potřebujeme strojově zpracovat. Není potřeba však zpracovávat celé zvukové spektrum ale zvuky zhruba mezi 125Hz a 8kHz[16], v tomto rozmezí se totiž pohybuje lidský hlas.

Mikrofon je jedno ze zařízení, které dokážou hlas nahrávat a převádět do elektronické formy. V elektronické podobě se pak nahrávka pošle k dalšímu zpracování, nebo se uloží. Tyto elektronické nahrávky se pak dají použít např v hudbě, nebo u chytrých asistentů s hlasovým ovládáním. Pro záměry této práce nahrávky využijeme jako trénovací materiál pro neuronovou síť, aby ji bylo možné ovládat hlasem. Druhým způsobem rozpoznání řečených slov jsou Markovovi modely. Tato práce je zaměřená na první způsob.

1.1 Mikrofon

Existuje několik druhů mikrofonu s rozdílnou konstrukcí i různým využitím. Rozenáváme například kondenzátorový, elektretový, dynamický či membránový mikrofon. Každý mikrofon zároveň má rozdílné charakteristiky v závislosti na konstrukci mikrofonu, mezi ně například patří charakteristiky všesměrové, kardioidní, hyperkadioidní a osmičková.

Kondenzátorový mikrofon[17] snímá kmitání membrány způsobené zvukovými vlnami. Membrána je jednou z elektrod kondenzátoru elektrického obvodu mikrofonu. Kapacita kondenzátoru se mění dle pohybu membrány. Tato změna se převádí na

elektrický signál. Kondenzátorové mikrofony obecně vyžadují elektrické napájení, aby mohl obvod fungovat. Při vhodné konstrukci mikrofonní vložky je také možné měnit směrové charakteristiky mikrofonu.

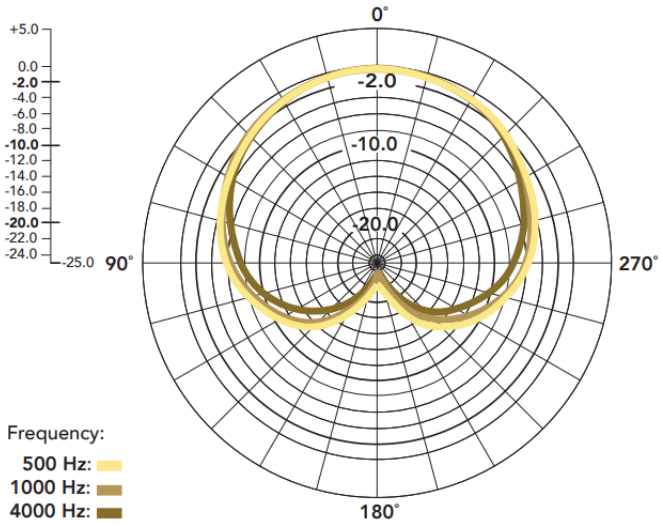
Kondenzátorové mikrofony jsou pokládány za nejkvalitnější a používají se často pro profesionální záznam. Také se vyrábějí pro měřicí účely.

Elektretový mikrofon[17] je postaven na podobné bázi jako kondenzátorový mikrofon, akorát elektrické nápájení je nahrazeno elektretem. Elektret je feroelektrický materiál, který byl permanentně elektricky nabit nebo polarizován. Výsledkem tohoto nahrazení je jednodušší výroba a tudíž i nižší cena.

1.1.1 Směrové charakteristiky mikrofonů

V závislosti na konstrukci pouzdra mikrofonu může tento přijímat zvuk z různých směrů v různé intenzitě. Směrová charakteristika je frekvenčně závislá – projevuje se zpravidla u vysokých tónů, zatímco hluboké zůstávají nepoznamenány.

Všesměrová charakteristika přijímá zvuk stejně kvalitně ze všech stran a je nejjednodušší na dosažení při konstrukci. Kardioidní neboli ledvinová charakteristika potlačuje příjem zvuku „zezadu“ mikrofonu, jak lze vidět na obr. 1.1. Jde o typickou charakteristiku dynamických mikrofonů pro zpěváky, neboť potlačuje zpětnou vazbu. Superkardioidní přijímá zvuk částečně i zezadu. Hyperkardioidní příjímá ještě více zvuku zezadu mikrofonu. Osmičková neboli bidirekcionální charakteristika je taková, při které mikrofon přijímá zvuk pouze zepředu a zezadu, nikoliv však ze stran.



Obrázek 1.1: Znázornění kardioidní charakteristiky

1.2 Formáty audio souborů

Existuje mnoho druhů formátů audio souborů, převážně se dělí na dvě skupiny: ztrátové a bezztrátové. Distribuované audio je převážné ve ztrátovém formátu, který má nižší velikost, ale zároveň nižší kvalitu kvůli ztrátové kompresi. Mezi ztrátové formáty patří: MP3, AAC, MULAW, AMR, AMR_WB, OGG, SPEEX a WebM.

Bezztrátové formáty si naopak snaží zachovat svoji kvalitu za cenu větší velikosti. Některé formáty jsou i nekompresované. Mezi bezztrátové formáty patří: FLAC, WAV, Linear16.

V rámci této práce je potřeba zachovat kvalitu nahrávky na co nejvyšší úrovni, aby neuronová síť mohla co nejjednodušeji rozpoznat řečená slova v nahrávce. Tudíž byl zvolem formát Wav, především kvůli bezztrátovosti.

1.2.1 MP3

MP3, formálně znám jako MPEG Audio Layer III, je kódovací formát pro digitální audio vyvinut převážně Společností Fraunhofer pod vedením Karlheinz Brandenburg[34]. Formát souboru MP3 označuje soubory obsahující datový tok dat zákodovánými formátem MPEG1, nebo MPEG2. Tento formát je znám použitím ztrátové komprese pro kódováním dat pomocí nepřesním aproximací a zahozením některých dat. Tento proces umožňuje velkou redukci velikosti souboru oproti nekompresovaným formátům. [35]

1.2.2 OGG

OGG je formát kontejneru pro multimédia, nativní formát souborů a streamovací formát nadace Xiph.org[32]. Podobně jako ostatní formáty kompresovaná data jsou zapouzdřena a toto umožňuje prolínání audio a video dat v jednom formátu. Formát také poskytuje rámcování paketů, detekci chyb, periodické časová značení.

1.2.3 WebM

WebM je otevřený media formát navržen pro použití na webu. Je bez licenčních poplatků.[33] Soubory tohoto formátu se skládají z kompresovaných video streamů spolu s VP8 nebo VP9 video kodekem a audio streamu kompresovaným pomocí kodeku Vorbis nebo Opus.

1.2.4 FLAC

FLAC[22] neboli Free Lossless Audio Codec, je bezztrátový audio formát podobný MP3. Toto znamená, že audio v tomto formátu je kompresováno beze ztráty kvality. Funkcionálně je to podobné formátu ZIP, akorát FLAC je specializovaný pro

audiozáznamy. FLAC je jedním z nejrychlejších a je jeden z nevíce podporovaných bezztrátových zvukových formátů. Navíc je open-source a není omezený žádnými patenty. Podporuje také značky neboli tagy a například také přebaly u hudebních skladeb.

Některé z význačných vlastností zahrnují bezztrátovost, rychlost dekódování, širokou hardware podporu, flexibilní metadata, streamovatelnost, archivovatelnost a odolnost vůči chybám. FLAC nehodlá přidat žádnou DRM ochranu proti kopírování.

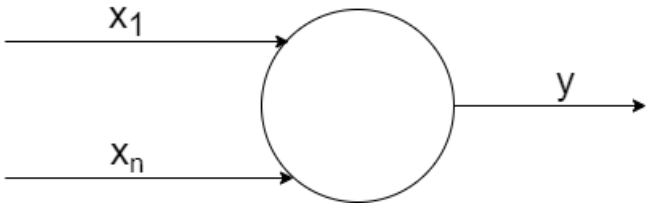
1.2.5 Waveform - WAV\WAVE

Waveform Audio File Format[23] neboli WAV\WAVE je audio formát vytvořený firmou IBM a Microsoftem, pro ukládaní audio bitstreamů. V systému Windows je hlavním formátem pro nekompresované audio, ale WAV může obsahovat i kompresované audio. Nekompresovaná audiostopa se ukládá pod LPCM (Linear Pulse-Code Modulation) formátem, který obsahuje 2 kanály snímané na 44 100Hz s 16 bitovými snímky. Tento formát je standardní pro Audio CD.

S Windows 2000 formát WAV začal obsahovat i hlavičku. V této části jsou definovány audio data pro jednotlivé kanály, pozice reproduktorů a sjednocuje snímkovámí v souboru. Avšak nacházejí se nedostatky například v duplicitě informací v jednotlivých blocích, což naznačuje že soubory mají velkou velikost, a také absence záporných hodnot a 8-bit kódování oproti 16-bit. WAV soubory jsou také limitovány do velikosti 4GiB kvůli limitaci hlavičky velikosti souboru na 32-bit číslo bez znaménka.[23]

1.3 Neuronové sítě

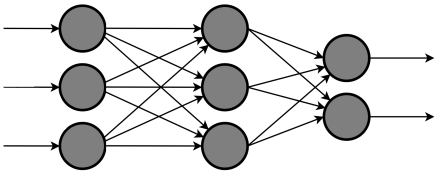
Pro zpracování zvuku vytvořeným lidským hlasem jsem použil neuronová síť. Počítačová neuronová síť funguje na bázi umělého neuronu. Inspirace přisla z lidského mozku, který je složen z mnoha neuronů. Tyto neurony pomáhají mozku se rozhodovat a toto chování počítačové neuronové sítě napodobují. Základní jednotkou je perceptron, který má n vstupů a obvykle jeden výstup.[18]



Obrázek 1.2: Jednoduchý perceptron

Neuronová síť je vlasně složení spojených perceptronů, které se kolektivně rozhodují na celkovém výsledku. Každý perceptron provede rozhodnutí a to se pak odešle do další jednotky v síti, dokud nedorazí na konec, finální rozhodnutí sítě. Rozhodnutí perceptronu je vážená suma všech vstupů. Každý perceptron má také svůj

bias, neboli hodnotu přičtenou k vážené sumě. Toto číslo se pak předá většinou[20] nelineární aktivační funci. Výsledek aktivační funkce je poté výstup perceptronu. Matematicky zapsáno: $y = f\left(\sum_{i=1}^{N} x_i * w_i\right)$, kde f je aktivační funkce a w je váha vstupu ve vážené sumě. Perceptron je znázorněn na obr. 1.2



Obrázek 1.3: Znázornění sítě

U neuronové sítě počet těchto jednotek může dosahovat vysokého čísla, a také mohou být rozděleny do více vrstev než jen vstupní a výstupní. Tyto vrstvy navíc se nazývají skryté vrstvy a slouží ke zpracovávání výstupů předchozích vrstev, skryté vrsty mohou vést k lepší přesnosti výsledku. V některých případech jsou dokonce nutné, aby síť byla schopna vyřešit daný problém. Příklad sítě se skrytou vrstvou je znázorněn na obr. 1.3

Většinou je také nutné data předzpracovat do určitého intervalu. Tento proces se nazývá preprocessing. U většiny sítí volíme interval $\langle -1;1 \rangle$. Díky tomuto procesu budeme mít sjednocená data, což síti pomůže s rozhodováním a počítáním.

1.3.1 Učení neuronové sítě

Neuronová síť se učí vyhodnocováním a analýzou vstupních dat, pomocí kterých s využitím interních proměnných poskytnou výstupní hodnoty. Po těchto hodnotách požadujeme, aby byli co nejblíže správnému, námi očekávanému, výsledku. S rostoucím počtem dat poskytnutých neuronové síťi, se síť optimalizuje danému problému, díky čemu je přesnější a rychlejší v jeho řešení.

Učení se rozděluje do čtyřech hlavních tříd. [18] Učení pod dohledem, učení s částečným dohledem, učení bez dohledu a zpětnovazební učení.

Učení pod dohledem

Síť se cvičí na datovém souboru se známými vstupy i výstupy a v průběhu cvičení se síť snaží najít vzor chování dat. Hledá také spojitosti a stanovuje odhady, ty jsou pak opravovány do dosáhnutí vysoké přesnosti a výkonu sítě.

Učení pod dohledem se používá především pro tyto typy úloh: Klasifikace, Regrese a Předopověď.

1. Klasifikace: V klasifikačních úlohách program kategorizuje objekty na základě předchozích pozorování.
2. Regrese: V regresních úlohách program potřebuje odhadovat vztahy mezi proměnnými.

3. Předpověď: Program stanovuje odhady o budoucnu na základě dat z minulosti, tento proces se také často používá na analýzu trendů.

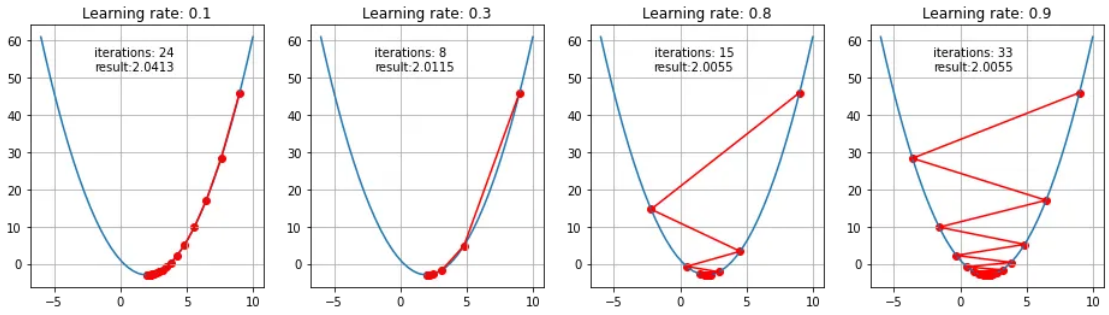
Gradientní sestup

Mezi učení pod dohledem patří metoda Gradientního sestupu[36](anglicky Gradient descent), jedná se o optimalizační algoritmus prvního řádu s cílem najít lokální minimum optimalizované funkce. Tento algoritmus není exkluzivní pouze pro strojové, či hluboké učení, je využíván i v mechanickém a řídícím inženýrství, nebo i počítačových hrách. Gradientní sestup má 2 spefické požadavky na optimalizavanou funkci. Funkce musí být diferencovatelná a konvexní. Pokud optimalizovaná funkce splňuje tyto požadavky, tak lze najít lokální, nebo i globální minimum funkce. Gradientem označujeme hodnotu první derivace funkce v určitém bodě. Gradientnímu sestupu je dána sada dat pro minimalizovanou funkci, algoritmus poté iteruje napříč celou sadou dat. V případě náhodného vybrání podmnožiny dat z poskytnuté sady, jednalo by se o stochastický gradientní sestup (SGD).

Algoritmus iterativně počítá další bod do kterého vkročí použitím gradientu na aktuální pozici. Aktuální gradient je škálován mírou učení a odečten od aktuální pozice. Matematicky zapsáno rovnicí: $p_{n+1} = p_n - Lr * f'(p_n)$. Když shrneme kroky algoritmu tak provádí následující:

1. Inicializace počátečního bodu
2. Výpočet gradientu v aktuálním bodě
3. Provede se krok v opačném směru proti gradientu
4. Opakujeme body 2 a 3, dokud algoritmus nedosáhne maximálního počtu iterací, nebo krok je menší než tolerance

Proces učení je znázorněn na obrázku 1.4, kde lze vidět průběh učení a dopad míry učení na počet iterací.



Obrázek 1.4: Průběh učení metodou gradientního sestupu

Učení s částečným dohledem

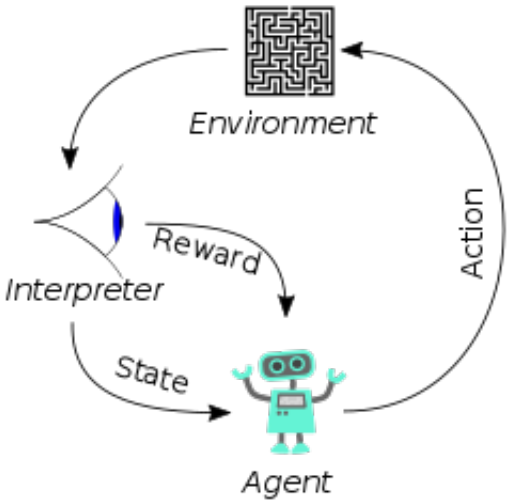
Je podobné učení s dohledem, ale používá označená i neoznačená data. Označená data nesou užitečnou informaci, zatímco u neoznačených taková informace chybí. Při použití tohoto postupu se program může naučit označovat data relevantní informací.

Učení bez dohledu

Studuje vzor chování dat, není zde klíč nebo operátor, který by napravil špatnou odpověď. Místo toho jsou dostupná data analyzována a pokouší se zjistit vztahy a korelace mezi jednotlivými daty. V procesu učení se programu poskytne velký datový soubor, který si pak program musí zorganizovat. Toto může znamenat, že se data budou shlukovat a nebo jinak seskupovat.

Zpětnovazební učení

Je metoda učení založená na odměňování požadovaného a trestání nevyhovujícího chování. Ve zpětnovazebném učení bývá agent, který je schopen vnímat a interpretovat své okolí a konat rozhodnutí a učit se metodou *pokus a omyl*. Programátor určuje odměny pro správná optimální rozhodnutí ve formě pozitivních hodnot a nulových nebo negativních hodnot pro nežádoucí rozhodnutí. Rozhodnutí agenta vyhodnocuje interpret, který zná hodnoty jednotlivých rozhodnutí a agentovi uděluje danou odměnu či trest spojený s učiněným rozhodnutím, znázorněno na obrázku 1.5.



Obrázek 1.5: Znázornění procesu odměny na základě rozhodnutí agenta

1.3.2 Sítě typu Recurrent neural network

RNN (Recurrent neural network)[19] - Třída neuronových sítí, která používá sekvenční data nebo časové řady. Tento algoritmus se používá hlavně u ordinálních

nebo časově závislých problémů, mezi které patří například: překlad vět a zpracování mluveného slova. Tuto technologii používají populární aplikace jako například Siri, hlasové hledání a nebo i Google překladač.

Stejně jako běžné neuronové sítě[19] tento typ sítě využívá trénovací sadu dat. Hlavní distinktivní vlastnost tohoto typu sítě je paměť, kde si uchovává informace z předchozích vstupů a tyto informace ovlivňují další rozhodování sítě. Běžné neuronové sítě neuvažují o existence spojitosti v pořadí dat, zatímco síť RNN tuto spojitost vidí a využívají ji v plném potencionálu při překladu cizích jazyků a zpracovávání řeci.

Další charakteristikou této sítě je sdílení parametrů perceptrony napříč jednotlivými vrstvami sítě, oproti tradičním sítím, kde se parametry liší u každého perceptronu. I přesto je potřeba tyto parametry přizpůsobovat procesem zpětnovazebního učení a gradientního sestupu.

Sítě RNN pro získání gradientu využívají zpětnovazební učení v čase, které se odlišuje od tradičního zpětnovazebného učení jeho přizpůsobení sekvenčním datům. Tato metoda avšak trpí dvěma problémy. Těmito problémy jsou tzv. explodující a mizející váhy. Síť se neustále učí a zpětnou propagací si upravuje váhu, pomocí které se rozhoduje. Vzhledem k tomu že váha je sdílená napříč modelem, tak se může stát že u explodující váhy bude váha nekonečně růst dokud nebudou výsledky NaN. U mizející váhy se budou váhy naopak zmenšovat.

Variace sítě RNN

- Obousměrná RNN - Variace RNN, kde síť analyzuje vstupní data spolu s kopií vstupních dat v invertovaném pořadí a tímto zvyšuje přesnost odhadu.

- Long-short term memory (LSTM) - Populární architektura sítě RNN. Adresuje problém mizejícího gradientu, také umí řešit dlouhodobé úlohy, které dříve nebyli řešitelné.[25].

  Později implementovaná verze LSTM je definována následující kompozitní funkcí:

$$
\begin{aligned}
\alpha_n &= \sigma(W_{i\alpha} i_n + W_{h\alpha} h_{n-1} + W_{s\alpha} s_{n-1}) \\
\beta_n &= \sigma(W_{i\beta} i_n + W_{h\beta} h_{n-1} + W_{s\beta} s_{n-1}) \\
s_n &= \beta_n s_{n-1} + \alpha_n \tanh\left(W_{is} i_n + W_{hs}\right) \\
\gamma_n &= \sigma(W_{i\gamma} i_n + W_{h\gamma} h_{n-1} + W_{s\gamma} s_n) \\
h_n &= \gamma_n \tanh\left(sn\right)
\end{aligned}
$$

  kde α, β, γ a s jsou respektivě vstup, zapomínací brána, výstup a stavový vektor.

- Gated recurrent units (GRUs) - Podobné architektuře LSTM, soustředí se na krátkodobé vazby.

1.3.3 Sítě typu RNN-T

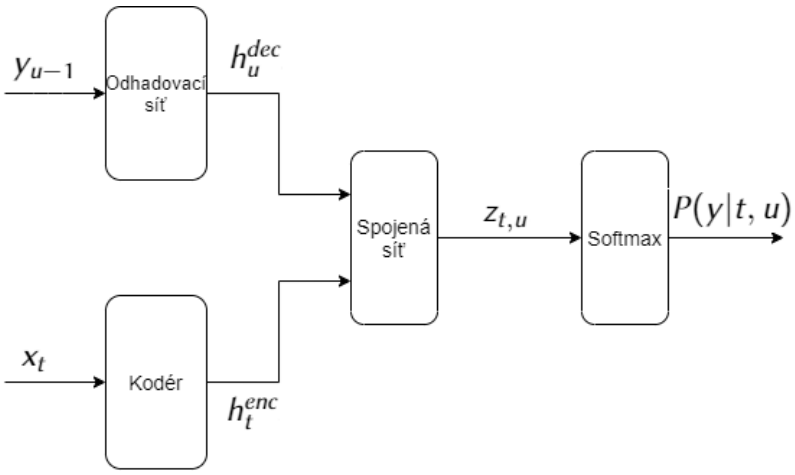
Architektura sítě RNN-T navrhnutá Gravesem[1] se skládá z kodéru, odhadovací sítě a typicky jedné společné sítě. Tato síť přímo odhaduje sekvence slov bez použití externího modelu.

Kodérem je zde zamýšlen akustický model, přijímá zvukové nahrávky a zpracuje je do vektoru o velikosti $K + 1$, kde K je počet různých výstupů. Tento model v Gravesově architektuře je obousměrná RNN síť.[1] Síť analyzuje vstup v obou směrech pomocí dvou skrytých vrstev. Tento typ sítě je preferován, protože každý vektor ve výstupu závisí na celém vstupu.

Odhadovací síť, narozdíl od kodéru, obsahuje pouze jednu skrytou vrstvu. Na základě vstupních data síť předpovídá znaky a výstupní vektor obsahuje i prázdný znak. Pro K znaků, délku výstupu U a $y_u = k(k \in K)$, poté $\hat{y_u}$ je vektor délky K, kde všechny členy vektoru jsou nula mimo k-tý člen. Prázdný znak je nulový vektor. Máme-li $\hat{y}$ síť počítá sekvenci skrytého vektoru $(h_0, .., h_U)$ iterováním následujících vzorců:

$$h_u = H(W_{ih}\hat{y}_u + W_{hh}h_{u-1} + b_h)$$
$$g_u = W_{ho}h_u + b_o$$

Kde W je váhová matice, W_{ih} označuje vstupní váhy, W_{hh} váhy skrytého vektoru h, W_{ho} váhy výstupu a b označuje bias. H označuje aktivační funkci, ve tradičních aplikacích RNN to bývá funkce $tanh$, a nebo logistický sigmoid.



Obrázek 1.6: Znázornění sítě typu RNN-T

Vstup spojené sítě jsou výstupní vektory odhadovací sítě a kodéru. Spojená síť vrací vektor pravděpodobností, který pak putuje do vrstvy s aktivační funkcí. Tato poslední vstva odhadne podslovo, z výsledných podslov pak síť sestaví slova, dokonce i řečenou sekvenci slov.

Kapitola 2

Návrh robota

V rámci návrhu robota je třeba vzít v úvahu složitost programu, aby robot byl schopen zpracovávat instrukce relativně rychle.

V rámci průzkumu různých mikropočítačů, které robota mohou řídit, jsou blíže specifikovány Rapsberry Pi 4 a NVIDIA Jetson.

2.1 Rapsberry Pi 4

Rapsberry Pi čtvrté generace[30] je malý počítač, vyzobrazen na obr.2.1, který je možné použít jako řídící jednotku pro chytrou domácnost, server pro média a jiné projekty, zahrnující strojové učení.

Aktuálně dostupný model B disponuje ARM procesorem Cortex-A72 se čtyřmi jádry na 1.8GHz. Paměť RAM je k dispozici v provedení 1GB,2GB,4GB a 8GB LPDDR4-3200. Síťové připojení je možné pomocí Wifi nebo Gigabit Ethernet portu. Na desce se nachází také standardní Rapsberry Pi 40pin GPIO konektor, 2 micro-HDMI konektory s podporou až dvou monitorů s rozlišením až 4k při 60Hz, port MIPI pro display a kameru.

Obrázek 2.1: Rapsberry Pi 4 8GB[29]

2.2 NVIDIA Jetson

NVIDIA Jetson[10] je vestavěný počítačový systém, který je optimalizován pro strojové učení a neuronové sítě. Je možné provozovat několik neuronových sítí. Aplikace využívající tyto sítě můžou klasifikovat obrázky, detekovat objekty a zpracovávat řeč. Jetson je vyzobrazen na obrázku 2.2

Tento počítač obsahuje čtyřjádrový ARM procesor běžící na 1.43Ghz. 128 jádrové Maxwell GPU. 4GB DDR4 paměti, sériový MIPI CSI-2 port pro videokameru a PCIe gen2 slot. Jetson Nano disponuje 472 GFLOPS výpočetního výkonu na 16bitových číslech při spotřebě 5-10W.

Obrázek 2.2: NVIDIA Jetson

K dispozici je také levnější model Jetsonu Nano, tento model má pouze 2GB RAM, méně USB 3.0 portů a jeden MIPI port na kameru.

U obou modelů se jako paměťové úložište využívá SD karta, na tuto SD kartu se instaluje image linuxového systému, který byl předpřipraven od společnosti NVIDIA.

2.3 Porovnání systémů

V tabulce 2.1, je vypsáno porovnání základních vlastností obou zmíněných počítačů, z tabulky jednoznačně vyplývá, že si Jetson povede lépe při složitějších úlohách strojového učení kvůli přítomnosti grafické karty. Navíc narozdíl od Rapsberry má RAM modul typu DDR4 který lze vyměnit. U Rapsberry je RAM modul přivařený na desku a tudíž prakticky nevyměnitelný. Výkon procesoru je témeř ekvivalentní a USB-C port u Rapsberry by byl využitelný pro nové periferie na bázi USB-C. Některé revize Jetsona jsou bohužel bez USB-C portu. Vzhledem ale k absenci grafické karty u Rapsberry je lepší využít Jetson pro využití v této práci, kvůli složitosti neuronové sítě a velikosti vstupních dat.

	Nvidia Jetson	Rapsberry Pi 4
Počet jader procesoru	4	4
Takt jádra	1,43GHz	1,8GHz
RAM	4GB DDR4	1-8GB LPDDR4
Grafická karta	128 jádrový Maxwell	bez grafické karty
Grafický výkon	472GFLOPS	/
Počet USB portů	3x USB 2.0 1x microUSB-B	2x USB 2.0 2x USB 3.0, 1x USB-C
Wifi	Ano	Ano

Tabulka 2.1: Porovnání systémů

2.4 Seznámení s NVIDIA Jetbot

NVIDIA Jetbot se skládá z NVIDIA Jetson, šasi a kol. Zakoupit se dá jako kompletní předvyrobený balíček a nebo jako DIY sada, ve které je NVIDIA Jetson, seznam materiálů, návod popisující tisk šasi a kol. Příklad sestaveného Jetbota je ukázán na obrázku 2.3

Obrázek 2.3: NVIDIA Jetbot

Jetbot je v základu vybaven programy napsanými v Jupyter, které využijí jeho základní výbavu. Příkladem je třeba program, který se naučí rozpoznat cestu podle obrazu na kameře. Jeden z dalších programů dokáže rozpoznat pohyb člověka a rozpoložení těla.

2.4.1 Nastavení sítě u robota

Jetbot se může programovat přímo na stroji nebo přes internet v prohlížeči. Pro programování v prostředí Jupyter je třeba robota pouze připojit k internetu přes kabel nebo Wi-fi. Robot pomocí tohoto síťového připojení získá adresu IP, pomocí které je možné se připojit na server s programovacím prostředím. Wi-fi lze na robotovi nastavit přes běžné uživatelské rozhraní pomocí myši, klávesnice a monitoru. Je také možné se připojit přes SSH relaci a nastavit připojení Wi-fi pomocí této vzdálené relace. U tohoto typu nastavování se nejdříve zapojí internetový kabel do robota a počká se než se na displeji robota objeví IP adresa. Se získanou IP adresou je pak možné připojení na SSH server přes vhodný software, například Putty. Po připojení a přihlášení se v konzoli zadá příkaz *nmcli device* a bude se hledat řádek *wlan0*. Přítomnost řádku s *wlan0* znamená, že robot podporuje připojení Wi-fi, a bude možné nastavit připojení. Přes příkaz *nmcli device wifi list* lze pak vypsat veškeré viditelné Wi-fi sítě v dosahu robota. Samotné připojení k Wi-fi síti pak zajistí příkaz sudo *nmcli device wifi connect <MY_WIFI_AP> password <MY_WIFI_PASSWORD>*, kde *MY_WIFI_AP* označuje SSID, neboli název Wi-fi a *MY_WIFI_PASSWORD* poté označuje heslo k této síti.

2.5 NVIDIA Jetbot použitý k realizaci

Použitý Jetbot byl zapůjčen na fakultě od vedoucího této bakalářské práce. Jedná se o model Waveshare, který má kovový podvozek se čtyřmi koly, Wi-fi antény, kameru a baterii. K robotu se dokoupil kondenzátorový USB mikrofon s kardinoidní charakteristikou pro možnost nahrání hlasových příkazů.

Operační systém robota je v tomto případě Ubuntu s přídavnými systémy od společnosti NVIDIA, jako například zabudovaný Jupyter server pro sepsání a vykonání kódu, pomocí kterého lze robota ovládat.

2.6 Knihovna Jetbot v Pythonu

Jedná se o open-source knihovnu od společnosti NVIDIA pro Jetbota. Obsahuje příkazy pro zjednodušení kódu ovládání robota, jeho kol a kamery. Knihovna dostává časté aktualizace od komunity uživatelů a také od společnosti samotné. Součástí knihovny jsou příkladové programy, ukazující základní pohyb, rozpoznání objektů a využití kamery.

Mezi příklady je například program na ovládání robota pomocí ovladače[37], ukázka 2.6

```
from jetbot import Robot
import traitlets
robot = Robot()
```
Zde se nastavuje otáčení kol robota pomocí ovladače
```
left_link = traitlets.dlink((controller.axes[1], 'value'), (
    robot.left_motor, 'value'), transform=lambda x: -x)
right_link = traitlets.dlink((controller.axes[3], 'value'),
    (robot.right_motor, 'value'), transform=lambda x: -x)
```

V ukázce níže je zahrnuto, jak sledovat obraz kamery přímo z prostředí Jupyter notebook
```
image = widgets.Image(format='jpeg', width=300, height=300)
display(image)
from jetbot import Camera
camera = Camera.instance()
from jetbot import bgr8_to_jpeg
camera_link = traitlets.dlink((camera, 'value'), (image, '
    value'), transform=bgr8_to_jpeg)
```

Kapitola 3

Návrh ovládacího softwaru

3.1 Použitý jazyk - Python

Python je interpretovaný, objektově orientovaný, vysokoúrovňový jazyk s dynamickou sémantikou. Python má vysokoúrovňové datové struktury s kombinací dynamiky Pythonu. Je velmi lákavý pro rychlý vývoj a jednoduchý debug. Je jednoduchý na naučení, díky jeho syntaxi a čitelnosti. Součástí Pythonu je také balíček Pip, pomocí kterého se dají jednoduše stahovat knihovny. Python také obsahuje Garbage collector pro správu využité paměti.[11]

Filozofie Pythonu je sumarizována dle dokumentu pod názvem The Zen of Python pomocí krátkých vět jako např. [26]:

- Krásný je lepší než ošklivý
- Explicitní je lepší než implicitní
- Jednoduché je lepší než složité
- Čitelnost se počítá
- Chyby by nikdy neměli projít potichu

Python používá odsazení pomocí mezer namísto závorek nebo klíčových slov. Po určitých slovech se odsazení zvětší a na konci bloku se opět sníží. Díky tomuto způsobu sémantiky se dosáhlo toho, že visuální struktura reprezentuje strukturu sémantickou.[27]

3.2 Prostředí Jupyter notebook

Jupyter je neziskový open-source projekt, který vznikl z IPythonu. Jupyter se snaží podporovat interaktivní data a vědecké výpočty napříč všemi programovacími jazyky. [24]

Jupyter notebook je webová aplikace, která umožnuje vytvářet a sdílet dokumenty se spustitelným kódem přímo z této aplikace, umožňuje zobrazit obrázky, grafy, výpočty a doprovodný text, který může vysvětlovat postup, nebo funkci daného kódu.

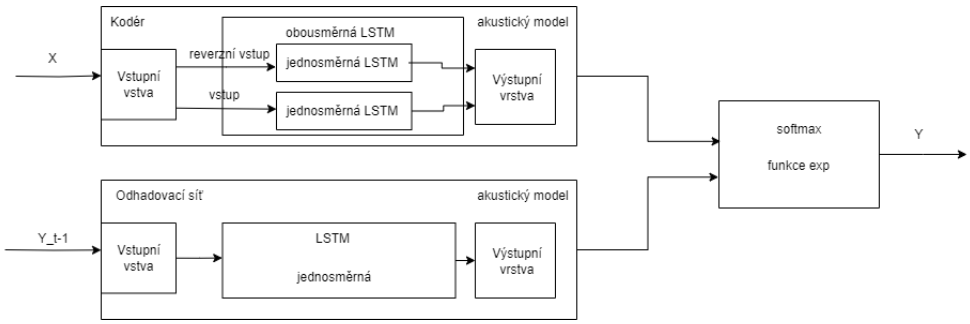
3.3 Možnosti implementace převodu řeci na text

Ovládací software lze implementovat s lokální neuronovou sítí, která by zajistila i fungování bez nutnosti internetu. V tomto případě by bylo nutno síť natrénovat, aby reagovala na určitá slova, ideálně od většího počtu různých mluvčích, a tím se zajistí rozpoznání klíčových slov. U většího počtu slov se může jednat klidně o stovky nahrávek slov a tím pádem i delší dobu trénování sítě a více zabraného místa na disku.

3.3.1 Návrh lokální sítě

Návrh implementované sítě se nebude moc lišit od architektury Gravese[1]. Znázorněný návrh 3.1 je doplněn o zpracování zvukové stopy na zpracovatelná data pro samotnou síť. Tyto zvuková data jsou odeslány do kodéru, který je převede na binární kód typu 1 z N, tento odhad se poté ve společné síti spojí s výstupem odhadovací sítě ve slovo dokud se neprojde celý zvukový soubor.

Pro příkazy bude vhodné zvolit jednoduchá slova, která se výrazně akusticky nepřekrývají. Síť poté bude možné jednoduše naučit a příkazy si nebude plést kvůli podobnému znění slov.



Obrázek 3.1: Návrh struktury neuronové sítě

Při návrhu lokální sítě je třeba brát v úvahu i jak se síť bude učit. Model Gravese využívá metodu sestupu gradientu[1], tuto metodu bych v implementaci využil také. Pro učení vnořené LSTM buňky bych využil vzorce gradientů, které jsou popsány v práci *LSTM: A Search Space Odyssey*[31]. Relevantní vzorce k učení jsou vyzobrazení na obr. 3.2.

Druhá možnost by bylo využít přednatrénované API s velmi vysokým počtem trénovacích nahrávek, tudíž i vysokou spolehlivostí, pokud lze zajistit nahrávku dostatečné kvality. Pokud nahrávka bude obsahovat okolní šum, tak je možné že síť

$$
\begin{aligned}
\delta \mathbf{y}^t &= \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1} \\
\delta \bar{\mathbf{o}}^t &= \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t) \\
\delta \mathbf{c}^t &= \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \bar{\mathbf{o}}^t + \mathbf{p}_i \odot \delta \bar{\mathbf{i}}^{t+1} \\
&\quad + \mathbf{p}_f \odot \delta \bar{\mathbf{f}}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1} \\
\delta \bar{\mathbf{f}}^t &= \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t) \\
\delta \bar{\mathbf{i}}^t &= \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t) \\
\delta \bar{\mathbf{z}}^t &= \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)
\end{aligned}
$$

Obrázek 3.2: Vzorce zpětného průchodu LSTM

tomuto jevu nebude přizpůsobená oproti lokální variantě a budou vznikat chyby. Výhodou je ovšem velmi rychlé vyhodnocení v datacentrech externích API, které jsou optimalizovány na tyto rozpoznávací operace.

3.3.2 Výběr vhodné API pro ovládací software

Pro návrh programu využívající API bylo třeba provést průzkum a vybrat vhodnou API. Níže jsou uvedeny jedny ze známnějších rozhraní API.

CMU Sphinx

Open source API pro rozpoznání mluveného slova od Carnegie Mellon University[7]

Má několik různých verzí pro různé aplikace a programovací jazyky např. PocketSphinx napsaný v C, SphinxTrain obsahující nástroje pro trénování akustických modelů a základní knihovnu SphinxBase. CMU Sphinx nasbírala přes 20 let výzkumu a dat. Podporuje několik jazyků, je jednoduše modifikovatelný, lze přidat i vlastní model - český jazyk bohužel není podporován.

Wit.ai

Wit.ai je jazykové rozhraní pro převod vět na strukturovaná data.[15] Nabízí jednoduché vytvoření chatovacích botů a jednoduché vytvoření multiplatformních aplikací, a dokonce podporuje technologii Smart Home a chytré příslušenství. Toto rozhraní poskytuje Facebook, také podporuje mnoho jazyků pro převod psaného textu do dat. Pro převod řeči na text je podpora jazyků skromnější. Český jazyk není podporován, tato služba je zdarma.

Microsoft Azure

Microsoft nabízí vlastní API k převodu řeči na text. Nabízí tvorbu vlastních modelů spolu s jejich přednaučenými modely, flexibilní nasazení a rozpoznání mluvčího.

Dále zaručuje bezpečnost dat na jejich cloudu.[12] Na stránkách Microsoftu jsou také ukázkové aplikace k této API. Podpora několika jazyků, český jazyk není podporován. 5 hodin zvuku na měsíc zdarma.

IBM Watson Speech to Text

IBM nabízí převod řeči na text přes jejich platformu Watson. Nabízí převod řeči na text v reálném čase pro 7 řečí, vysokou přesnost, upravitelnost modelu. Podporuje nahrávky až do délky 1000s. IBM zaručuje rychlý a přesný převod řeči na text i z nahrávek nižší kvality. [14] Podporují několik jazyků, český jazyk se mezi nimi bohužel nenachází. IBM nabízí plán na 500minut měsíčně zdarma.

Google API

Google nabízí svoji API pro převod řeči na text. Momentálně je podporováno 125 jazyků. Google nabízí velmi přesné rozpoznání, díky neuronovým sítím s hlubokým učením. Jednoduchou úpravu modelu pomocí uživatelského rozhraní. Google také nabízí službu se kterou se model uloží na lokální server a data se nemusí posílat na servery Google. [13] Nabízí možnost převodu na text v reálném čase, nebo přes soubor. Cenový plán Googlu je 60minut měsíčne zdarma, a poté se přejde na placený režim s cenou zhruba 0,5Kč/min viz. Obrázek 3.3

Feature	Standard models (all models except enhanced video and phone call)		Enhanced models (video, phone call)	
	0-60 Minutes	Over 60 Mins up to 1 Million Mins	0-60 Minutes	Over 60 Mins up to 1 Million Mins
Speech Recognition (without Data Logging - default)	Free	$0.006 / 15 seconds **	Free	$0.009 / 15 seconds **
Speech Recognition (with Data Logging opt-in)	Free	$0.004 / 15 seconds **	Free	$0.006 / 15 seconds **

Obrázek 3.3: Ceník Google API

Google API má také omezení ve formátu, velikosti a délky nahrávek. Podporovány jsou nahrávky v bezztrátovém formátu *FLAC* a v *LINEAR16*. Platforma ale také podporuje ztrátová kódovaní, mezi která patří *MULAW*, *AMR*, *AMR_WB*, *OGG_OPUS*, *SPEEX_WITH_HEADER_BYTE*, *MP3* a *WEBM_OPUS*.

Formáty *WAV* a *FLAC* obsahují záhlaví popisující vlastnosti audionahrávky. Díky tomuto záhlaví není potřeba ve volání API konfigurovat kódování, bitovou šířku nahrávky a jiná potřebná metadata týkající se samotného souboru. Google obecně doporučuje používat kódování *FLAC*.

Samotná nahrávka však nesmí překročit velikost 10MB a nesmí být delší jak 480 minut v případě asynchronního volání funkce *LongRunningRecognize*, nebo být delší jak minuta v případě synchronního volání funkce *Recognize*. Existuje také možnost nahrávku streamovat pomocí funkce *StreamingRecognize* a tento způsob má limit

délky streamování 5 minut. Pokud by bylo třeba streamovat déle jak 5 minut, tak Google připravil návod s kódem pro nekonečný stream.

K volání rozponávacích funkcí je také možné přidat vlastní sadu frází, celkem až 5000 frází do 100 000 znaků s maximem 100 znaků pro jedno slovo. Aby Google zabráníl přetížení svých serverů, tak limitoval počet požadavků na 900 denně s maximálním zpracováním 480 hodin zvukové stopy.

Porovnání API

Při zvolení API pro vývoj softwaru jednoznačně vyhrálo Google API kvůli podpoře českého jazyka. Cenově je však nejméně výhodný, jak lze vidět v tabulce 3.1.

Pokud by se zvolila varianta ovládání robota pomocí anglických příkazů, byla by zvolena jedna z open source variant. Tato volba by se projevila i jako levnější alternativa za cenu možné nižší kvality datové sady a rozpoznávání oproti placeným API, které jsou poskytovány od velkých korporací.

Název API	Podpora CZ	Cena
CMU Sphinx	Ne	Zdarma
Wit.ai	Ne	Zdarma
Microsoft Azure	Ne	300minut měsíčně zdarma
IBM Watson Speech to Text	Ne	500minut měsíčně zdarma
Google API	Ano	60minut měsíčně zdarma

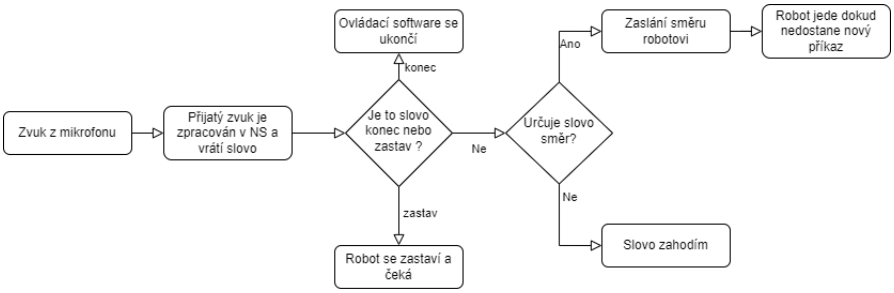
Tabulka 3.1: Porovnání API

3.4 Popis ovládacího software

Program je rozdělen na několik částí. Nahrávací část, kde se ve 2 proudech nahrává zvuk. Nejprve nahrává první proud po dobu několika vteřin, poté začne nahrávat proud druhý, a následně se vrací k prvnímu proudu. Tento postup zajištuje, že robot vždy poslouchá. Jakmile jakýkoli proud nahraje svoji stopu, tak se poslední dvě stopy spojí a odešlou se do další části programu.

V druhé části programu se přijatý zvukový soubor odesílá ke zpracování do rozpoznávacího software, který vrací textovou formou slov, která byla v nahrávce řečena. Přijatá slova se následně odesílají k dalšímu zpracování uvnitř robota. Software filtruje slova na příkazy, které program zná a je schopen je zpracovat. Zpracované příkazy odesílá pohybové funkci. Pohybová funkce tento vstup zpracuje a sdělí robotovi kam se pohybovat, či jestli se má zastavit.

Běh programu je znázorněn na diagramu 3.4.



Obrázek 3.4: Diagram programu

3.4.1 Diagram ovládacího software

Na diagramech 3.5 a 3.6 lze vidět jak program komunikuje s bash konzolí a rozpoznávacím modulem programu. Program začíná v hlavní smyčce, kterou hostuje Jupyter notebook. Tato smyčka sekvenčně volá funkce *nahraj*, *rozpoznej*, *urci smer* a *pohni*, tyto jednotlivé funkce zajištují funkcionality po kterých jsou pojmenovány.

Funkce *nahraj* inicializuje nahrávací vlákna a poté zavolá funkci *zapniskript*, aby rozlišila jaké vlákno momentálně nahrává, poté spojí poslední nahrávku z obou vláken a pošle výsledek zpět.

Tato nahrávka se pošle do funkce *rozpoznej*, zde se převede nahrávka do textové podoby. Textová forma nahrávky je poté odeslána do funkce *urci smer* tato funkce z pole vybere příkaz a převede na číselnou hodnotu určující směr ve funkci *pohni*.

Diagram lokální sítě

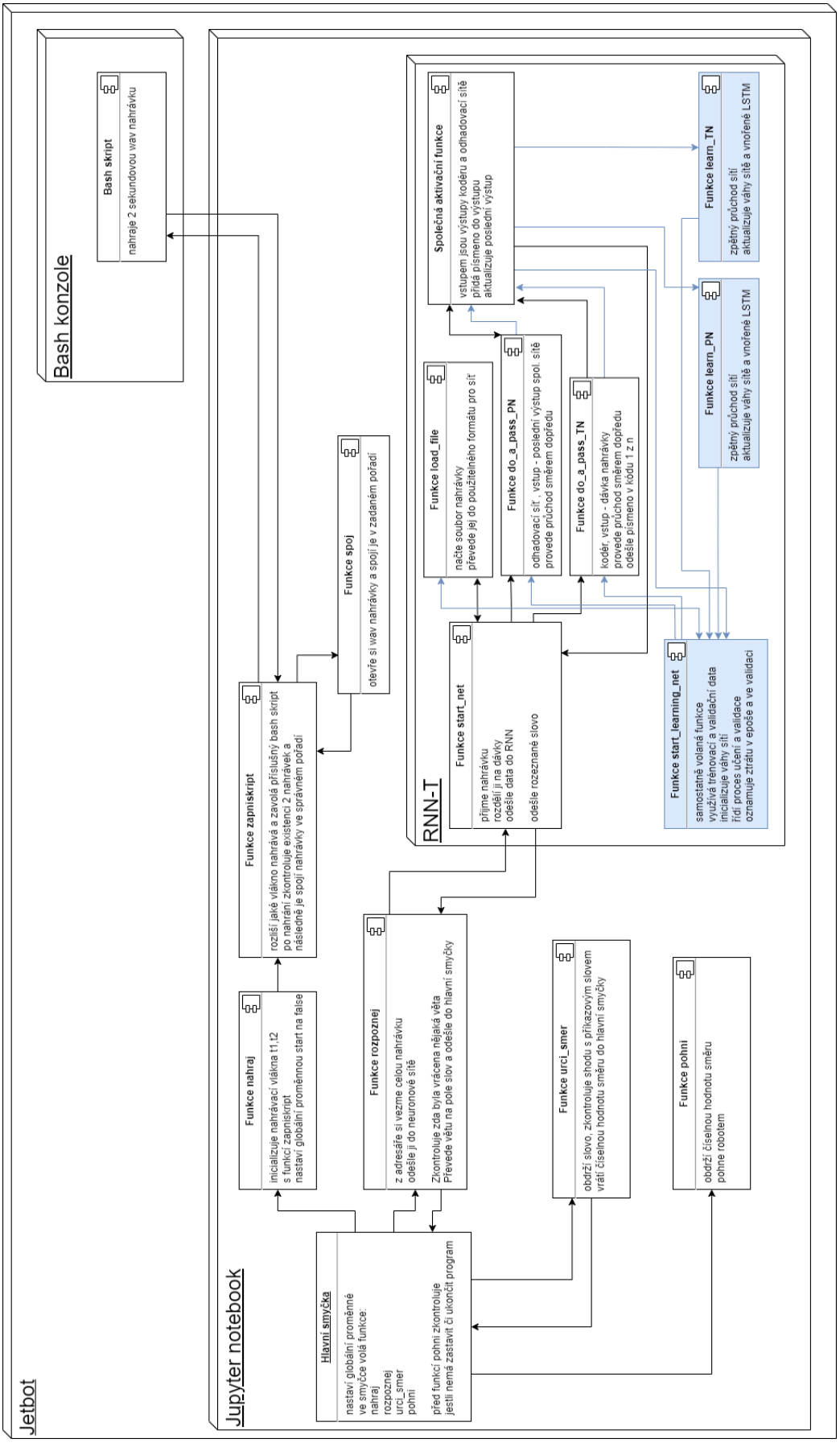
Rozpoznávací modul je zde implementován pomocí sítě typu RNN-T, která má funkci *start_net*. Tato funkce započne průchod sítí dopředu, přijme cestu nahraného souboru a zavolá funkci *load_file*. Tato funkce zajistí načtení souboru do paměti a převede ji do formátu, který neuronová síť umí zpracovat. Zpracovaná nahrávka se odešle do sítě kodéru pomoci funkce *do_a_pass_TN*. Síť kodéru jsem popisoval již v kapitole návrhu lokální sítě a obsahuje 2 jednosměrné LSTM buňky, kde jedna obdrží invertovaný vstup. Výstup z této sítě je poté předán do společné sítě. Síť RNN-T si také pamatuje poslední výstup ze společné sítě, který se posílá do odhadovací sítě, implementovanou funkcí *do_a_pass_PN*. Tato síť zpracovává poslední výstup společné sítě a na základě toho odhaduje další znak v sekvenci (neboli slovu). Výstup odhadovací sítě a kodéru je poté vyhodnocem společnou sítí, sloužící jako softmax vrstva sítě. V diagramu 3.5 je znázorněna komponentou nazvanou *Společná aktivační funkce*. Výstup z této společné sítě je poté odeslán zpět na začátek pro další iteraci a také aktualizuje co si síť myslí, že bylo řečeno v nahrávce.

V diagramu 3.5 si lze všimnout také modrých kompoment, které jsou odpojeny od hlavní smyčky. Zavoláním funkce *start_learning_net* se spustí učení neuronové sítě, tato funkce má variabilní počet epoch a míru učení. Ve funkci se načtou cesty k trénovací a validační sadě nahrávek obsahující jednotlivé příkazy, které jsou od robota požadovány. Tyto cesty se pak načítají pomocí funkce *load_file* v náhodném

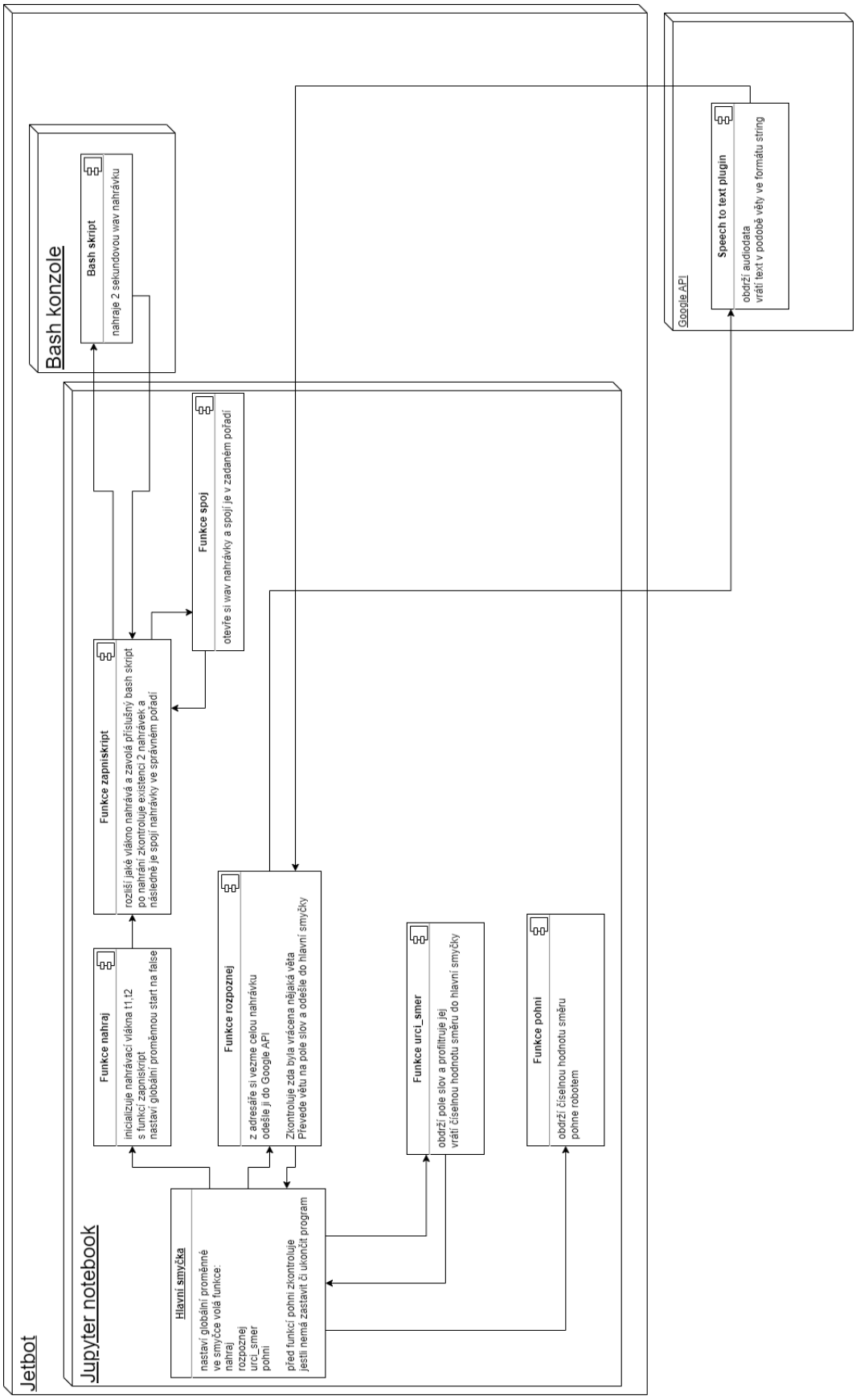
pořadí, tento proces zajistí více efektivní učení, oproti učení jednoho slova několikrát za sebou. Nahrávka projde stejným procesem jako ve výše popsaném předním průchodu. Na konci průchodu dopředu je poté výstup ze společné aktivační funkce, tento výstup je odeslán do funkcí *learn_ PN* a *learn_ TN*. Tyto funkce zajištují trénovaní jednotlivých sítí metodou gradientního sestupu popsaném v teoretické části práce. Tento proces se opakuje v závislosti na počtu epoch a po určitém počtu epoch se průchodem dopředu vyhodnotí validační nahrávka, která se liší od trénovacích. Tento validační proces poté oznamuje jakou přesnost síť má.

Diagram programu využívající Google API

V diagramu 3.6 je rozopoznávání zajištěno pomocí Google API. Nahrávka je odeslána do Google API a zpracována neuronovou sítí, kterou tato API poskytuje. Po dokončení rozpoznání API vrátí pole slov, které se odešle do funkce *urci smer* tato funkce z pole vybere příkaz a převede na číselnou hodnotu určující směr ve funkci *pohni*.



Obrázek 3.5: Deployment diagram pro ovládací software s lokální sítí



Obrázek 3.6: Deployment diagram pro ovládací software používající Google API

Kapitola 4

Popis implementace

Program byl vyhotoven ve dvou verzích. Jedna využívá neuronovou síť typu RNN-T inspirovaná návrhem Gravese[1], kterou popisuji jako první. Poté popisuji verzi s využitím API od Google.

4.1 Varianta s lokální sítí

Síť je sestavená podle návrhu popsaném v podkapitole 3.3.1. Základními prvky sítě jsou kodér, odhadovací síť a společná síť. V ukázce 4.1 je kódová struktura kodéru a odhadovací sítě.

V kodéru je implementován výpočet matice vystupující ze vstupní vrstvy pomocí vstupního vektoru x a vetktoru váhy W_ih, Tato hodnota je poté předána do skryté vrstvy kde se nejdříve spočíta hodnota odeslaná do LSTM buňky pomocí výstupu vstupní vrstvy a předchozího výstupu sítě vynásobeného váhou W_hh, k výsledku se přidá bias bh. Tato matice se odešle do LSTM buňky (ukázka 4.1), kde tato matice skryté vrstvy projde všema bránama LSTM a vrátí výstup z výstupní brány. LSTM si uloží jednotlivé vypočtené hodnoty bran, aby se nemuseli poté znova počítat.

```
def do_a_pass_PN(x, Weights, prev_h, prediction_net):
    #Prediction network - unidirectional LSTM
    W_ih, W_hh, W_hy, bh, bo = Weights

    net_input = (x@W_ih.T).T

    hidden = net_input+(W_hh.T*prev_h)+bh
    activation_hidden = prediction_net.feedfoward(hidden)

    output = (W_hy.T@activation_hidden).T
    output += bo

    prev_h = activation_hidden

    return hidden, prev_h, output

def do_a_pass_TN(x, x_rev, Weights, prev_h_f, prev_h_b,
    prediction_nets):
    #Transcription network - bidirectional LSTM
    prediction_net_f, prediction_net_b = prediction_nets
    W_ihf, W_ihb, W_hhf, W_hhb, W_hfy, W_hby, bhf, bhb, bo =
        Weights

    hidden_f = (x @ W_ihf.T) + (W_hhf.T*prev_h_f)+bhf
    hidden_b = (x_rev @ W_ihb.T) + (W_hhb.T*prev_h_b)+bhb

    activation_hidden_f = prediction_net_f.feedfoward(
        hidden_f)
    activation_hidden_b = prediction_net_b.feedfoward(
        hidden_b)

    output = (W_hfy.T@activation_hidden_f).T
    output += (W_hby.T@activation_hidden_b).T
    output += bo
    output = output.reshape(np.shape(bo))

    prev_h_f = activation_hidden_f
    prev_h_b = activation_hidden_b

    return prev_h_f, prev_h_b, hidden_f, hidden_b, output
```

Listing 4.1: Ukázka sítě kodéru a odhadovací sítě

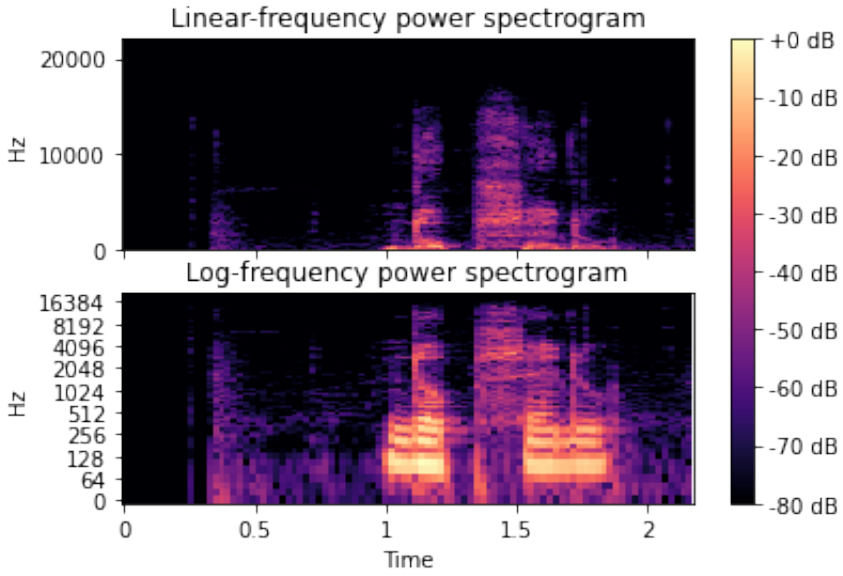
```
def feedfoward(self,x):
    #for bidirectional feed reversed x to a second lstm
    self.z = self.block_input(x)
    self.a = self.input_gate(x)#alfa_n
    self.b = self.forget_gate(x)#beta_n
    self.s = self.state_vector(self.a,self.b,self.z,
        self.prev_state)#s_n - cell state(also known as c
        )
    self.y = self.output_gate(x,self.s)#gamma_n
    self.h = self.hidden_vector(self.y,self.s)#h_n -
        block output (yt)

    self.prev_hidden = self.h
    self.prev_state = self.s

    assert np.shape(self.z) == np.shape(self.Wsy), f"{np
        .shape(self.z)} /= {np.shape(self.Wsy)}"
    assert np.shape(self.a) == np.shape(self.Wsy), f"{np
        .shape(self.a)} /= {np.shape(self.Wsy)}"
    assert np.shape(self.b) == np.shape(self.Wsb), f"{np
        .shape(self.b)} /= {np.shape(self.Wsb)}"
    assert np.shape(self.y) == np.shape(self.Wsa), f"{np
        .shape(self.y)} /= {np.shape(self.Wsa)}"
    assert np.shape(self.h) == np.shape(self.Wsa), f"{np
        .shape(self.h)} /= {np.shape(self.Wsa)}"

    self.states = np.concatenate((self.states,self.s),
        axis=1)
    self.hidds = np.concatenate((self.hidds,self.h),axis
        =1)
    self.output_gs = np.concatenate((self.output_gs,self
        .y),axis=1)
    self.forgets = np.concatenate((self.forgets,self.b),
        axis=1)
    self.inputs = np.concatenate((self.inputs,self.a),
        axis=1)
    self.blocks = np.concatenate((self.blocks,self.z),
        axis=1)
    return self.h
```

Listing 4.2: Ukázka předního průchodu LSTM buňkou

Vstupem sítě je audiostopa, kterou poskytne nahrávací smyčka. Aby síť s ní mohla pracovat, je třeba ji převést na data. Konverzi zajišťuje Python knihovna Librosa.[28]

Pomocí knihovny Librosa lze audionahrávky graficky zobrazit pomocí spektrografu. Viz obrázek 4.1

Tato knihovna určerná pro analýzu audia a muziky nahrávku načte a převede ji



Obrázek 4.1: Znázornění nahrávky ve spektrogramu

na data, která síť obrží. Tyto data dorazí do kodéru, který je zpracuje a pošle do společné sítě.

Aby síť byla schopná provozu je jí potřeba natrénovat pomocí sady nahrávek a výsledných slov. Toto proběhne procesem zpětnovazebního učení s využitím algoritmu klesajícího gradientu pro každou nahrávku. Po dokončení trénování síť bude schopna rozlišit naučená slova a tím ovládat robota. V rámci zjednodušení trénování sítě byla zvolena výrazná slova jako: Dopředu, Dozadu, Vlevo, Vpravo, Stop, Konec. Z těcho slov jsem si vybral unikátní znaky a ty zákodoval do kódu 1 z N, kódem v ukázce 4.1. S tímto kódem pak síť umí pracovat. Trénovací sadu dat jsem vytvořil tím že jsem se nahrál vyslovení daného příkazu a vytvořil jsem 2 různé nahrávky pro každé slovo.

```
straight = ["dopredu"]
left = ["vlevo"]
right = ["vpravo"]
back = ["dozadu"]
ukoncit = ["konec","stop"]

all_chars = ""
for word in (straight+left+right+back+ukoncit):
    all_chars += word

unique_chars = set(all_chars)
char_nr =len(set(all_chars))
enc_table = np.eye(char_nr)

paired = {}
for pair in zip(unique_chars,enc_table):
```

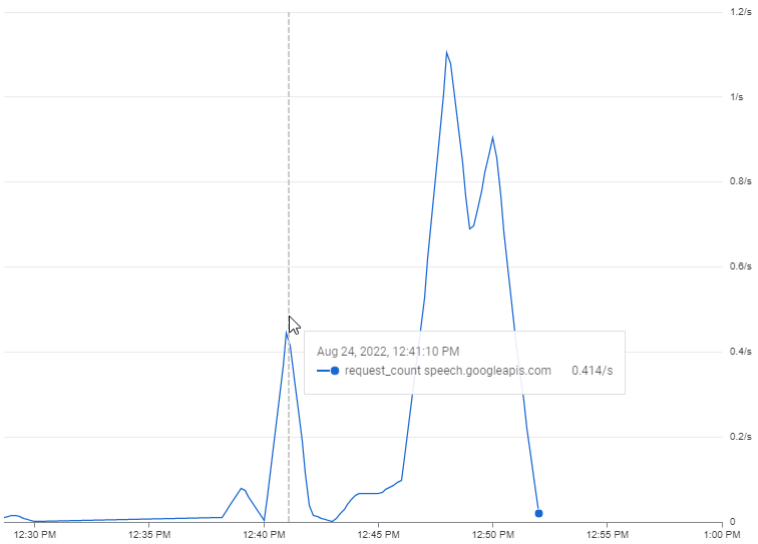
```
        paired.update({pair[0]: pair[1]})

paired.update({'∅': paired['d'].copy()*0})
```

Listing 4.3: Kódování slov do 1 z N

4.2 Varianta využívající Google API

Program jsem začal vyvíjet v Google Colab, kde byl sestaven prototyp využívající Google API. Pro jeho použití třeba založit projekt v Google Cloud, abych mohl vytvořit autorizační klíč, který poté program využíval. Tento program se skládal ze skriptu na nahrávání zvuku, uložení nahrávky na disk Google a odeslání do Google API. V Google Colab bylo nutné povolit přístup k prostředkům prohlížeče a povolit nahrávání mikrofonem pomocí javascriptu.

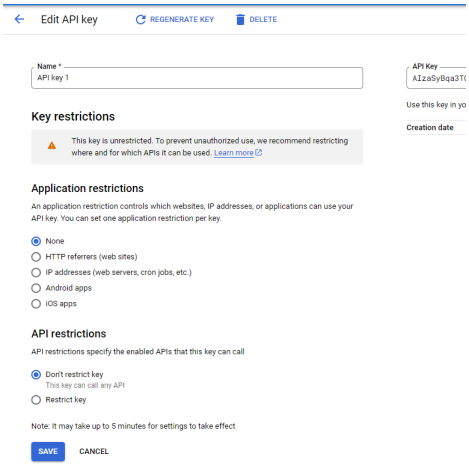
V založeném projektu lze sledovat používání služeb Google sledovat, pomocí vestavěného reporting systému. Na obrázku 4.2 máme znázorněn graf využití volání Google API. Dodatečná informace u myši zobrazuje, kolikrát za vteřinu se volala API.



Obrázek 4.2: Tabulka volání API

Pomocí této cloudové konzole lze nastavit funkci OAuth, a nebo také různé služby API nebo i SDK, jako například: SDK pro mapy Google a pro Android/iOS, Text-to-Speech, Speech-to-Text, různé služby do Youtube, strojový překlad a mnoho jiných. Po povolení API potřebné pro tuto práci bylo nutné vygenerovat autorizační klíč v nastavení projektu. U tohoto klíče je zapotřebí nastavit práva přístupu (viz obrázek 4.3), aby nedocházelo k neoprávněnému použití klíče. Tento klíč se pak použije v příkazu uvnitř kódu pro autorizaci.

Po testování různých frází jsem vývoj přesunul spolu s testováním na Jetbota s vývojovým prostředím Jupyter notebook. Při prvním testu jsem zjistil, že nefungoval

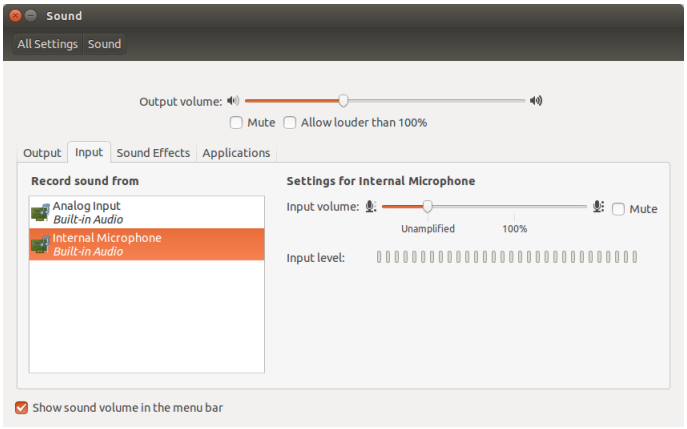


Obrázek 4.3: Nastavení přístupového klíče do Google API

dříve napsaný nahrávací skript. Při hledání řešení tohoto problému jsem testoval ovládání robota s pevně stanovenými frázemi. U nastavení směru pohybu robota bylo třeba stanovit sílu motoru, nebo zda-li robot správně reaguje na stanovené fráze. Po dokončení testování jsem se vrátil k implementaci nahrávání, kde jsem zvolil nahrávání přes bash soubor spuštěný z vývojového prostředí. Na další testování jsem si připravil čistou instalaci Ubuntu na notebooku pro možnost testování mikrofonu před nasazením a testem na robotovi. Na notebook bylo třeba nainstalovat Jupyter a knihovny potřebné pro běh programu. Server Jupyteru jsem posléze spustil z příkazového řádku ve složce s programem pomocí příkazu *jupyter notebook*.

4.3 Konfigurace mikrofonu

Mikrofon bylo třeba nastavit na správnou úroveň citlivosti snímání, aby negeneroval šum (Obrázek 4.4). V případě vysokého šumu by pak neuronová síť v další části programu nemohla rozponat řečená slova.



Obrázek 4.4: Příklad nastavení mikrofonu při testování

4.4 Příprava bash souboru

Bash obsahuje příkaz *arecord*. Tento příkaz spustí nahrávání výchozím systémovým mikrofonem a pomocí parametrů je možné upravit jeho chování. V základní podobě tento příkaz nahrává v 8-bit formátu a se vzorkovací frekvencí 8000Hz. Při testování příkazu byl formát změněn na S16_LE a frekvenci 44 100 Hz. Tyto parametry se osvědčili vyšší kvalitou nahrávky a lepším rozpoznáním v Google API. Pomocí parametrů jsem stanovil formát nahrávky na příponu *wav* formátu Waveform. Parametr *-c 2* nastavuje nahrávaní na Stereo místo základního jednokanálového Mono a při testování zvýšil kvalitu nahrávky. Délka nahrávky byla stanovena na 2 sekundy a ukládala se pod názvem *nahravka.wav*.

Při testování této metody se projevil problém usekávání slov a zpožděného nového spuštění nahrávání kvůli sekvenčnímu zpracování programu. Řešením tohoto problému byla paralelizace dvou nahrávacích proudů a jejich následné spojení do jedné nahrávky. V příkazu bash skriptu se změnil pouze název souboru, aby bylo poznat, jestli nahrál první či druhý proud. Do skriptu bylo třeba přidat záhlaví pro systém, aby poznal, že se jedná o shell skript pomocí *#!/bin/sh*. Za zahlávím pokračuje kód skriptu: *arecord -t wav -d 2 -f S16_LE -c2 -r44100 "nahravka1"*. Aby skript bylo možné programově spustit, bylo třeba nastavit práva pomocí příkazu *chmod +x "název skriptu"*.

Po otestování funkčnosti skriptu jsem sepsal funkci, která bude ve vlastním vlákně volat nahrávání každé 2 sekundy, a po dokončení nahrávání spojí nahrávky z obou vláken ve správném pořadí. Také bylo třeba zahrnout kontrolu, jestli program neukončil běh. Spojená nahrávka se pak odeslala do Google API.

4.5 Hlavní smyčka

Hlavní smyčka se skládá ze základních volání funkcí, které řeší jednotlivé segmenty programu. V následujícím textu budu popisovat ukázku 4.5. Funkce *nahraj* nahraje nahrávku, ta se pošle do funkce *start_net*, která rozpozná řečené slovo a to se poté odešle do funkce *urci_smer*, která zvolí směr pohybu robota odeslaný do funkce pohni.

```
global t1,t2,start,konec

i = 0
start = True
konec = False

try:
    while True:
        i+=1
        if i%6 == 0:
            clear_output()
            vyčistí výstup notebooku
```

```
            i-=6
            po každých 6 cyklech

        nahravka = nahraj()
        nahraje audiozáznam

        slovo = start_net(nahravka)
        záznam se odešle do RNN-T

        sm = urci_smer(slovo)
        z neuronové sítě se vrátí slovo, které se nyní zpracuje

        if sm<-10:
        kontrola ukončení smyčky

            break
        pohni(sm)
        změna směru pohybu robota

except Exception as e:
    print("\n\n\033[91m\033[1m"+repr(e)+"\033[0m\033[0m")

ukončení vláken
konec = True

zastavení robota
robot.stop()

program počká než vlákno dokončí poslední cyklus
t1.join()
t2.join()
```

Listing 4.4: Hlavní smyčka

Program začíná vytvořením počítadla, které počítá počet oznámení programu do prostředí Jupyter a pokračuje nekonečnou smyčkou, ve které začíná běžet hlavní část programu. Funkce *try* kontroluje, zda-li se neobjevila v programu neočekávaná chyba. Pokud rozpoznávací funkce nepoznala řečená slova, nebo žádná neslyšela, program nahlásí, že nic nezaregistroval.

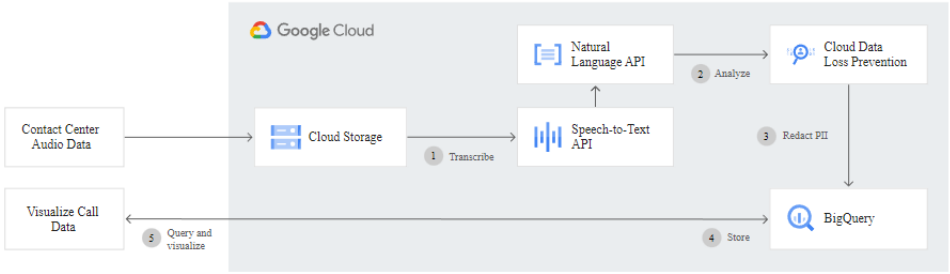
V první sekci programu se počítá počet výstupů do prostředí Jupyter, a pokud počítadlo dosáhne stanovené hodnoty (momentálně 6), tak program vstoupí do sekce funkce, která vyčistí výstup vypsaný robotem. Také jsou stanoveny globální proměnné určující, zda-li se program nyní zapnul, nebo jestli je program ve stavu ukončení.

V další sekci je nahrávací funkce realizovaná pomocí multiprocessingu. Multiprocessing umožní programu pracovat ve více vláknech, která mohou běžet současně

nezávisle na stádiu programu. Pomocí této technologie mohu spouštět externí soubor zajišťující nahrávání přes Linuxové bash příkazy. Toto nahrávání je postaveno na příkazu *arecord*, pomocí kterého volám nativně mikrofon bez dodatečných knihoven, které by mohli ovlivnit kvalitu a rychlost zpracování nahrávání. Sestavený příkaz nahrává po dobu 2 sekund z připojeného mikrofonu, a po této době se nahrávka uloží. Po 2 sekundách se zapne druhý proud nahrávání, který vytváří nahrávku v době kdy se zpracovává první nahrávka. Program dále kontroluje, jestli existuje nahrávka z obou těchto proudů, aby mohli být spojeny do jedné nahrávky ve správném pořadí. Spojování jednotlivých nahrávek zajisťuje Python knihovna wave, která umí zpracovávat soubory typu Waveform.Tato sloučená nahrávka se pak odešle do rozpoznávací funkce.

Rozpoznávací funkce přijme nahrávku a nahraje ji do paměti. Nahrávku si funkce poté rozdělí na dávky, které půjdou dále do sítě. Pokud se jedná o první běh rozpoznávací funkce, funkce si také inicializuje váhy neuronových sítí. Dávka dat z nahrávky se poté odešle do kodérů, který data zpracuje a pošle napříč neuronovou sítí a vnořenou LSTM buňkou, implementace ukázána na 4.1. Výstup kodéru jde poté do společné sítě, kde se spojí výstup z kodéru a odhadovací sítě a vyhodnotí se jejich součet pomocí vzorce: $Y_t = softmax(koder + odhad)$. Odhadovací síť jako vstup přijímá výstup společné sítě v čase $t - 1$ a pomocí tohoto vstupu se pokusí odhadnout další znak v sekvenci. Tento odhad poté odešle do společné sítě.

Variace Google API



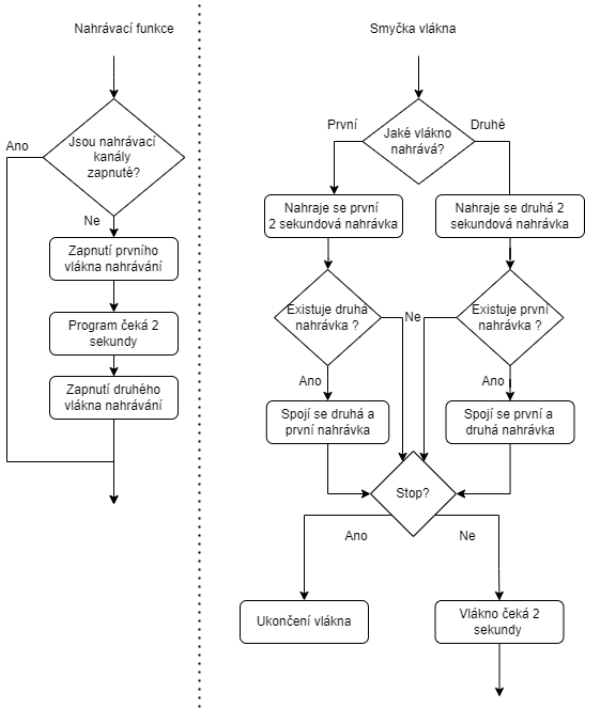
Obrázek 4.5: proces zpracování požadavku v Google API[13]

Rozpoznávací funkce zašle požadavek na Google API s autorizačním klíčem, který se vygeneruje na Google stránkách. Tento klíč se musí nastavit, kvůli autorizaci požadavku. Případně i fakturaci pokud by počet požadavků překročil rámec bezplatné verze. Po přijetí požadavku se v Google API zpracuje odeslaná nahrávka neuronovou sítí RNN-T, proces je vyzobrazen na obrázku 4.5. Po zpracování nahrávky neuronová síť vrátí větu, která byla v nahrávce řečena a tuto větu obdrží program zpět od Google API. Tato věta je odeslána do funkce, která větu rozdělí na pole slov. Toto pole je pak dále zpracováno filtrující funkcí, která vybere předdefinovaná směrová slova, převede je na číslo určující směr, kterým robot pojede, nebo jestli se zastaví.

Kapitola 5

Testování a ladění

Během vývoje a testování programu byli objeveny různé nedostatky prvotních návrhů. Jedním z problémů bylo usekávání slov v nahrávce způsobené koncem nahrávání, aby se vytvořil soubor nahrávky. Tento styl nahrávání slov byl v jednom proudu problémový, protože běžel pár vteřin, a poté byl program na nějakou chvíli hluchý také se stávalo, že bylo useknuto příkazové slovo. Tento problém se opravil implementací dvouproudového nahrávání, které nahrává v jednom proudu a mezitím druhý proud ukládá nahrávku. Tento přístup zajistí, že robot nebude hluchý v jakémkoli stavu programu. Toto řešení požadovalo implementaci multiprocessingu a spouštění proudů ve dvou vláknech.



Obrázek 5.1: Proces nahrávání pomocí vláken

Spolu s implementací vláken do programu se objevil problém s jejich chováním po ukončení hlavní smyčky. V testovacích verzích se vlákna spustili, ale nevěděli,

jestli je program ukončen. Toto způsobilo, že i po ukončení programu se generovali nové nahrávky. Při opětovném zapnutí programu se pak vytvořila další nová vlákna, která také vytvářela nahrávky a způsobovalo to problémy s nahráváním, a také toto neustálé přepisování rovněž není vhodné pro životnost disku. Tento problém byl původně řešen vypnutím celého procesu Pythonu. Toto řešení nebylo ideální, jelikož bylo třeba definovat všechny funkce a proměnné v programu znovu. Jelikož vlákna sdílí všechny globální proměnné, tak jsem zvolil možnost smyčky řízené globální proměnnou *konec* typu Boolean, která zastaví smyčku vlákna po ukončení hlavní smyčky. Nakonec si nechám programem potvrdit zastavení smyčky vlákna, pomocí zavolání funkce *join*, která je definována v objektu vlákna.

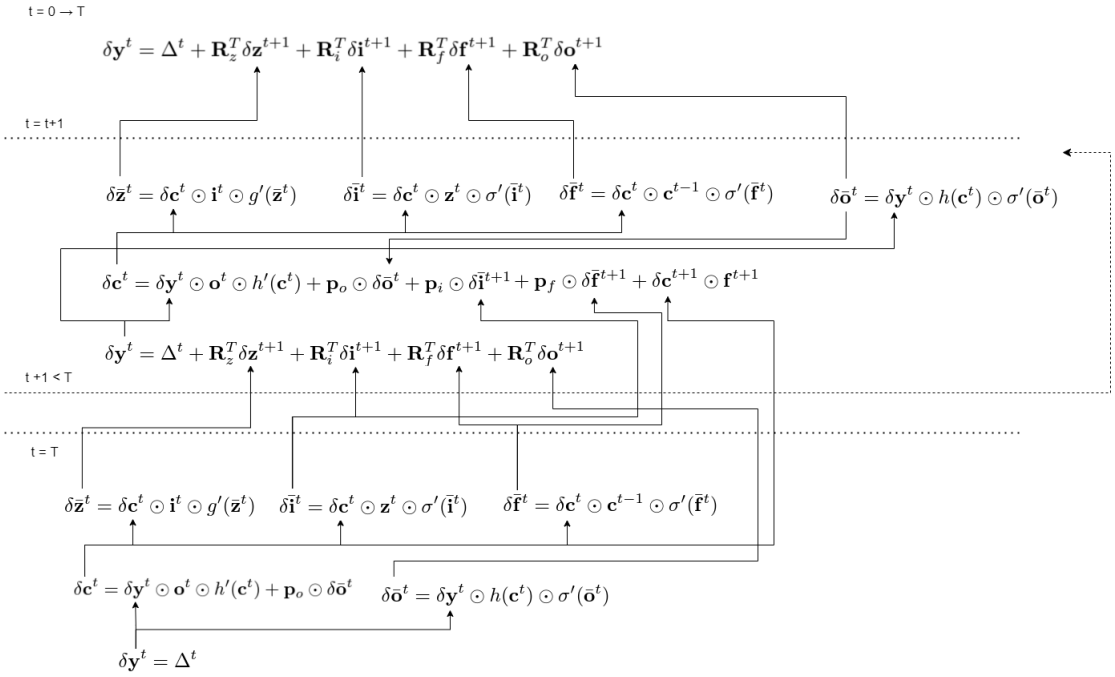
5.1 Učení a validace neuronové sítě

Implementace sítě začala vytvořením jednoduchého RNN modelu s jednou LSTM buňkou ve skryté vrstvě. Stanovil jsem si vstupní množinu o 50, 100 a 500 jednotkách, kde jsem vypozoroval dramatický nárust doby nutné ke zpracování jedné iterace trénovacích dat. Což vede k velmi dlouhým intervalům učení. Navíc při zkoušení sítě jsem narazil na zmíněné problémy mizejích a explodujích vah, kde jsem tento problém manuálně ošetřil nahrazením záporného, kladného nekonečna a hodnoty *NaN* čísly *-1*, *1* a *0*.

Učení sítí bylo implementováno rekurzivně s použitím vzorců, které použil Graves[1] ve své práci. K inspiraci způsobu implementace jsem využil sadu návodu zvanou *Zero to GPT*[38]. V této sadě návodu je do detailu popsána implementace zpětného průchodu RNN sítí. Učení LSTM bylo implementováno pomocí vzorců uvedených v práci *LSTM: A Search Space Odyssey*[31]. Všiml jsem si dlouhé doby zpracování kvůli hloubce rekurze.

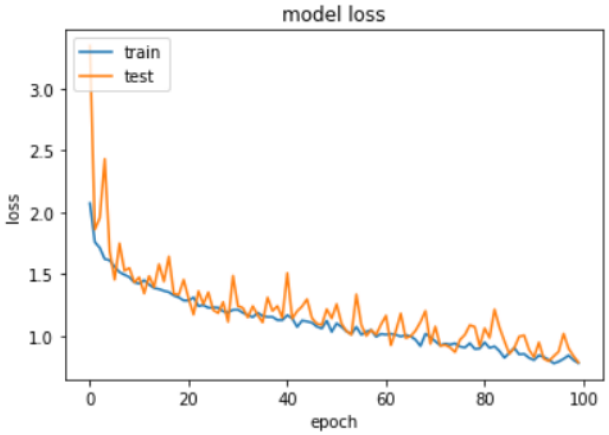
Při implementaci LSTM buňky rekurzivní metodou, jsem narazil na chybu s maximální hloubkou rekurze a bylo třeba převést hluboce rekurzivní funkce na iterativní. Tuto konverzi jsem provedl způsobem rozložení funkce do několika časových okamžiků $t \in T$, pomocí čehož jsem dorazil ke vzorcům, které lze vykonávat v jednom časovém okamkžiku a výsledky odeslat do další iterace výpočtu. Tento proces je znázorněn na diagramu 5.2. Začíná se vzorcem navrchu diagramu v čase $t = 0$ a rekurzivně se propadá dolů jednotlivými vzorci až do času $t = T$. Šipkama je znázorněn průběh iterace, kde se začíná vespod diagramu v času $t = T$ a iterativně se program protočítává až do času $t = 0$.

Vyzkoušel jsem implementovat grafickou akceleraci výpočtů pro zrychlení, u malé datové sady učení trvalo zhruba o 50% déle než bez grafické akcelerace. Implementace grafické akcelerace pro malou datovou sadu přidala akorát režijní náklady, které nepřinášely dostatečný benefit. Naopak u velké datové sady, v mém případě nahrávky, spadla doba učení sítě z 3 hodin na 5 minut. Po implementaci variabilního dávkového zpracování se rychlost opět zvýšila díky omezení délky vstupní množiny. Bylo však složité dle velikosti dávky správně nastavit váhy, aby se program neočekávaně neukončil.



Obrázek 5.2: Převod rekurze na iteraci

U učení sítě jsem si nechal hlásit hodnotu průběhu minimalizace funkce loss po určitém počtu epoch a stejnou hodnotu jsem si nechal hlásit u vyhodnocení validační ztráty. Tato funkce je implementována funkcí střední kvadratické chyby. Její cílem minimalizace této je funkce je obdržet hodnotu 0.0. Volnost interpretace matic knihony *NumPy*, dospěla k chybám, že chyba vycházela v hodnotách *NaN*, kladné nekonečno a záponé nekonečno. Tyto hodnoty bylo pak nutné ošetřit, jejich nahrazením. Na obrázku 5.3 je vyzobrazen průběh minimalizace ztrátové funkce loss.



Obrázek 5.3: Průběh minimalizace

Dalším problémem byla kvalita a hlasitost nahrávané řeči, program jsem testoval na dvou různých mikrofonech, bohužel jeden z nich nebyl vhodný k využití vzhledem k mobilní povaze robota. Hlavním problémem byl šum. Při kalibraci citlivosti mikrofonu bylo třeba dbát na sílu šumu na jednotlivých úrovních nastavení, také bylo

třeba hlídat hlasitost nahrávky, aby nebyla příliš tichá. Pokud by hlas v nahrávce byl příliš tichý, síť by měla problém rozpoznat, jaká slova byla řečena. Při zvyšování výstupní hlastosti se také objevoval šum. U kvalitnějších mikrofonů nebylo potřeba výrazné nastavování, zatímco u levnějšího mikrofonu byl přítomen šum způsobený elektrickým polem usb portu robota, nebo nedostačným odstíněním samotného mikrofonu.

Google API

U verze využívající Google API jsem síť testoval nejdříve odesláním nahrávek jednotlivých slov, pro kontrolu jestli API komunikuje se servery Google Cloud. Toto bylo vidět na grafu 4.2, kde se vytvořila špička v době volání a počtu volání API konektoru. Při zasílání výrazů do API jsem testoval nastavení mikrofonu, jestli nahrává dostatečně nahlas a zda-li moc nešumí. Po ověření nastavení jsem se přesunul k zasílání nahrávek vět, abych věděl jak bude vypadat výstup API v živém nasazení a mohl tento výstup poté dále zpravcovat. API byla daleko citlivější na kvalitu nahrávky než lokální síť, jelikož Google API trénuje na nahrávkách, které šum obsahují v minimální formě, nebo jej neobsahují.

Při dlouhodobém testování robota, se mi podařilo vyčerpat rámec výpočetní doby, která mi byla poskytnuta zadarmo. API plynule přesla do placeného režimu, aniž bych si toho všiml, nebo byl upozorněn. V Google Cloud konzoli bylo poté možné vyčíst vyučtování od Google vycházející na 0,35 Kč za překročenou výpočetní dobu.

5.2 Mikrofony použité k testování

MC-1UN0 mini5.4

MC-1UN0[22] mini je malý mikrofon, který se dá zapojit pomocí USB konektoru do počítače. Tento mikrofon je kondenzátorový s kardinoidní charakteristikou což znamená, že mikrofon sníma nejsilněji z místa, kde jsou otvory k okruhu mikrofonu. Jeho frekvenční rozsah zahrnuje frekvence mezi 100Hz a 16KHz. Má impedanci 2.2KOhm s citlivostí -47dB (± 4dB).

U tohoto mikrofonu se projevoval šum způsobený elektrickým polem robota. Podobný šum se projevoval i při testování na stolním počítači. U mikrofonu bylo potřeba zmenšit citlivost tak, aby šum nebyl výrazný a nedělal problémy s identifikací slov. I přes tyto změny šum stále ovlivňoval výsledky sítě. Při pokusu o odstranění šumu v profesionálním nahrávacím programu, zůstala zvuková stopa velmi tichá.

Při podařené identifikaci začal mikrofon snímat zvuk motoru a kol robota, takže bylo nutné mluvit více nahlas, jinak by byl zvuk jízdy robota hlasitější než hlasové příkazy.

Obrázek 5.4: MC-1UN0 mini

Audio-Technica ATGM25.5

Audio-Technica ATGM2[23] je nasazovací mikrofon s lepivým úchytkem na sluchátko. Jeho hlavní využitím je volání a streamování. Mikrofon je flexibilní a dá se různě ohýbat, umožnuje dosáhnutí optimální pozice. S hyperkardiodní charakteristikou je zajištěna minimalizace okolního hluku, a také je přibalená ochranná vrstva pro omezení zvuků dýchání. Kabel mikrofonu je 3m dlouhý a obsahuje tlačítko na vypnutí mikrofonu. K počítači se připojuje 3.5mm jack konektorem, k připojení do robota byla využita externí ADC zvuková karta připojená přes USB. Citlivost tohoto mikrofonu je -37dB s impedancí 2,1KOhm.

Používání tohoto mikrofonu je určeno na malou vzdálenost díky jeho konstrukci. Proto není velmi vhodný pro toto využití jako snímač hlasu na robotovi, uživatel by musel chodit s mikrofonem v ruce za robotem pro zaručení ovladatelnosti a rozpoznání řečených slov.

Obrázek 5.5: Audio-Technica ATGM2

Závěr

Implementace programu umožnila realizovat veškeré požadované funkce, které může uživatel od daného robota požadovat. Při testování se objevili problémy hlavně u vývoje a ladění lokální sítě a zvukových částí programu.

U lokální sítě jsem se setkal s problémy mizejících a explodujících vah popsánými v teoretické části, což síť činilo nestabilní a konvergující v nekonečnu resp. v záporném nekonečnu. Samotné učení sítě je delší než při využití API, kde je tento proces optimalizovaný pro rychlost. Kvůli popsaným problémům s váhami, jsou výstupy ze sítě nepřesné.

Verze aplikace využívající API, poskytovala přesné výsledky, ale bylo potřeba nepřerušené online připojení. Tato verze byla úspěsně otestována na robotovi.

Pro ideální implementaci z hlediska šumu je vhodné použít mikrofon s konfigurovatelnou funkcí gain a rozsahem snímaní 360°. Ve zvukové sekci byly problémy především s nahráváním, kde bylo potřeba správně specifikovat a nastavit nahrávací zařízení. Dále byla zapotřebí i kontrola šumu, jelikož lokální síť a i API je velmi citlivé na šum a při použití šumového filtry bude mít dopad na přesnost detekce mluvených slov.

Předpokládám však, že robot bude moci operovat i v prostředí se zvýšeným šumem a jiným okolním hlukem. Robot reaguje na základní směrové příkazy jako *dopředu*, *dozadu*, *doleva*, *doprava* a také na příkazy *zastavit* a *stát* při kterých se přestane pohybovat. Rovněž umí rozpoznat příkaz *konec*, kde se zastaví smyčka progamu a program se vypne.

Literatura

[1] GRAVES, Alex. *Sequence Transduction with Recurrent Neural Networks* [online]. Department of Computer Science, University of Toronto, Canada: University of Toronto, Canada, 2012 [cit. 2021-9-6]. Dostupné z: https://arxiv.org/pdf/1211.3711.pdf

[2] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 8024731002.

[3] JOSEPH, Lentin. *Learning Robotics Using Python*. Birmingham: Packt Publishing, 2015. ISBN 978-1783287536.

[4] KURNIAWAN, Agus. *Getting Started with NVIDIA Jetson Nano*. PE Press, 2019. ISBN 9780359978212.

[5] *JetBot* [online]. Santa Clara, California, United States: NVIDIA, 2021 [cit. 2021-9-6]. Dostupné z: jetbot.org

[6] *Jetson FAQ* [online]. Santa Clara, California, United States: NVIDIA, 2021 [cit. 2021-9-6]. Dostupné z: https://developer.nvidia.com/embedded/faq

[7] CMU Sphinx. *CMU Sphinx* [online]. Carnegie Mellon University: -, 2021 [cit. 2021-9-6]. Dostupné z: https://cmusphinx.github.io/

[8] Convolutional Neural Networks (CNNs / ConvNets). *CS231n Convolutional Neural Networks for Visual Recognition* [online]. [cit. 2021-3-11]. Dostupné z: https://cs231n.github.io/convolutional-networks/

[9] *Language model fusion for streaming end to end speech recognition* [online]. Mountain View, California, United States: Google, 2021 [cit. 2021-9-6]. Dostupné z: https://arxiv.org/pdf/2104.04487.pdf

[10] *NVIDIA Jetson* [online]. Santa Clara, California, United States: -, 2021 [cit. 2021-9-6]. Dostupné z: https://www.nvidia.com/cs-cz/autonomous-machines/jetson-store/

[11] *Python* [online]. Python, 2021 [cit. 2021-9-6]. Dostupné z: python.org

[12] *Speech to Text: A Speech service feature that accurately transcribes spoken audio to text* [online]. Redmond, Washington, United States: Microsoft, 2021 [cit.

2021-9-6]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/#documentation

[13] *Speech-to-Text* [online]. Mountain View, California, United States: Google, 2021 [cit. 2021-9-6]. Dostupné z: https://cloud.google.com/speech-to-text

[14] *Watson Speech to Text* [online]. Armonk, New York, United States: IBM, 2021 [cit. 2021-9-6]. Dostupné z: https://www.ibm.com/cz-en/cloud/watson-speech-to-text

[15] *WIT.AI* [online]. Wit.ai, 2020 [cit. 2021-9-6]. Dostupné z: https://wit.ai/

[16] Lidský hlas. *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2022, 13. 4. 2022 [cit. 2022-06-18]. Dostupné z: https://cs.wikipedia.org/wiki/Hlas

[17] Mikrofon. *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2022, 13. 4. 2022 [cit. 2022-06-18]. Dostupné z: https://en.wikipedia.org/wiki/Microphone

[18] Neuronové sítě. *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2022, 13. 4. 2022 [cit. 2022-06-18]. Dostupné z: https://en.wikipedia.org/wiki/Artificial_neural_network

[19] Recurrent neural networks. *IBM* [online]. Armonk, New York, United States: IBM Cloud Education, 2020, 14.9.2020 [cit. 2022-07-02]. Dostupné z: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[20] Mc-1un0-mini. *NEVEN.CZ* [online]. BRNO: NEVEN.CZ, 2022, 2.7.2022 [cit. 2022-07-02]. Dostupné z: https://www.neven.cz/kategorie/pocitace-a-kancelar/mikrofony/mc-1un0-mini-usb-pc-mikrofon/#cerna/

[21] ATGM2. *Audio-Technica* [online]. Machida, Tokyo, Japan: Audio-Technica, 2022, 2.7.2022 [cit. 2022-07-02]. Dostupné z: https://www.audio-technica.com/en-us/atgm2

[22] FLAC - introduction. *FLAC* [online]. Xiph.Org: Xiph.Org Foundation, 2022 [cit. 2022-07-28]. Dostupné z: https://xiph.org/flac/features.html

[23] Wav. *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2022, 28. 7. 2022 [cit. 2022-07-28]. Dostupné z: https://en.wikipedia.org/wiki/WAV

[24] Jupyter. *Jupyter* [online]. USA, 2022, 13. 9. 2022 [cit. 2022-09-13]. Dostupné z: https://jupyter.org/

[25] HOCHREITER, Sepp a Jurgen SCHMIDHUBER. *Long short term memory* [online]. Neural Computation, vol. 9, nr. 8, pp. 1735-1780, 1997 [cit. 2023-05-27]. Dostupné z: https://www.bioinf.jku.at/publications/older/2604.pdf. Publikace. Institute of Bioinformatics, Johannes Kepler University Linz.

[26] Zen of Python. *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2023 [cit. 2023-05-27]. Dostupné z: https://en.wikipedia.org/wiki/Zen_of_Python

[27] Python (programming language). *Wikipedia* [online]. San Francisco, USA: nadace Wikimedia, 2023 [cit. 2023-05-27]. Dostupné z: https://en.wikipedia.org/wiki/Python_(programming_language)

[28] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[29] Raspberry Pi 4 Model B: 8GB RAM. *RPishop.cz* [online]. Roudné: RPishop.cz, 2023, 27.06.2023 [cit. 2023-06-27]. Dostupné z: https://rpishop.cz/raspberry-pi-4/2611-raspberry-pi-4-model-b-8gb-ram-0765756931199.html

[30] Raspberry Pi 4 Model B. *Rapsberry Pi* [online]. United Kingdom: Raspberry Pi Foundation, 2023, 27.06.2023 [cit. 2023-06-27]. Dostupné z: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[31] SCHMIDHUBER, Jurgen, Klaus GREFF, Rupesh K. SRIVASTAVA, Jan KOUTNÍK a Bas R. STEUNEBRINK. LSTM: A Search Space Odyssey: Transactions on neural networks and learning systems [online]. Cornell University, 2017 [cit. 2023-07-07]. Dostupné z: https://arxiv.org/pdf/1503.04069.pdf. Publikace. Cornell University.

[32] Ogg [online]. USA: Xiph.Org Foundation, 2016 [cit. 2023-07-19]. Dostupné z: https://www.xiph.org/ogg/

[33] WebM [online]. USA: WebM Project, 2023 [cit. 2023-07-19]. Dostupné z: https://www.webmproject.org/about/

[34] KARLHEINZ BRANDENBURG - MP3 [online]. Německo: WebM Project, 2015 [cit. 2023-07-19]. Dostupné z: https://www.internethistorypodcast.com/2015/07/on-the-20th-birthday-of-the-mp3-an-interview-with-the-father-of-the-mp3-karlheinz-brandenburg/

[35] MP3 [online]. San Francisco: nadace Wikimedia, 2023, 08.07.2023 [cit. 2023-07-19]. Dostupné z: https://en.wikipedia.org/wiki/MP3

[36] KWIATKOWSKI, Robert. Gradient Descent Algorithm — a deep dive. Towards Data Science [online]. Towards Data Science: Medium.com, 2021 [cit. 2023-07-20]. Dostupné z: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

[37] Teleoperation. NVIDIA-AI-IOT: Jetbot [online]. GitHub: NVIDIA, 2021 [cit. 2023-07-20]. Dostupné z: https://github.com/NVIDIA-AI-IOT/jetbot/blob/master/notebooks/teleoperation/teleoperation.ipynb

[38] PARUCHURI, Vik. Zero to GPT: Explanations. Github [online]. Github: Github, 2023, 12.07.2023 [cit. 2023-07-25]. Dostupné z: https://github.com/VikParuchuri/zero_to_gpt/tree/master/explanations

Obsah CD

Datová struktura na CD obsahuje tuto práci s názvem *Kusy_ BP.pdf* a složky pro jednotlivá řešení ovládacího software - *Google API* a *Lokalni sit*. Ve složce *Google API* naleznete nahrávací skripty *record1.sh* a *record2.sh* a Jupyter notebook s názvem *Voice_ recog Google API.ipynb*. Ve složce *Lokalni sit* naleznete rozmanitější datovou strukturu, kde se nachází složka *training* s nahrávkama příkazů, Jupyter notebooky *LSTM object.ipynb*, *pomocne funkce.ipynb* a *Voice_ recog.ipynb*, kde poslední zmíněný je hlavní skriptem pro program. A také stejné nahrávací skripty *record1.sh* a *record2.sh* jako u Google API verze.