Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01685/2010
Datum: 01.09.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **KATJA CETINSKI**

Naslov: **GLASOVNO UPRAVLJANJE TELEVIZORJA Z UPORABO SISTEMA SPHINX-4**

VOICE CONTROLLED TELEVISION SET USING SYSTEM SPHINX-4

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Govor je za človeka najbolj naraven način komunikacije. Zato se že dolgo razmišlja o uporabi avtomatskega razpoznavanja govora za upravljanje in krmiljenje naprav raznih vrst. Ena od posebno primernih naprav za glasovno upravljanje je televizijski sprejemnik, ki se običajno upravlja preko tipk na daljinskem upravljalniku. S pomočjo programskega orodja Sphinx-4 razvijte in izdelajte razpoznavalnik govora, ki razpoznava množico slovenskih besed primernih za upravljanje televizorja. Za to množico izdelajte akustični model in podatkovno bazo govornih posnetkov, na kateri izvedite učenje razpoznavalnika. Razvijte in izdelajte tudi elektronski vmesnik, ki razpoznane besede pretvori v ustrezni infrardeči signal, s katerim se daljinsko krmili televizor. Glasovno upravljanje televizorja preizkusite in izmerite uspešnost njegovega delovanja.

Mentor:
prof. dr. Dušan Kodek

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:
prof. dr. Nikolaj Zimic

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Katja Cetinski

GLASOVNO UPRAVLJANJE TELEVIZIJE Z UPORABO SISTEMA SPHINX-4

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01685/2010
Datum: 01.09.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **KATJA CETINSKI**

Naslov: **GLASOVNO UPRAVLJANJE TELEVIZORJA Z UPORABO SISTEMA SPHINX-4**

VOICE CONTROLLED TELEVISION SET USING SYSTEM SPHINX-4

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Govor je za človeka najbolj naraven način komunikacije. Zato se že dolgo razmišlja o uporabi avtomatskega razpoznavanja govora za upravljanje in krmiljenje naprav raznih vrst. Ena od posebno primernih naprav za glasovno upravljanje je televizijski sprejemnik, ki se običajno upravlja preko tipk na daljinskem upravljalniku. S pomočjo programskega orodja Sphinx-4 razvijte in izdelajte razpoznavalnik govora, ki razpoznava množico slovenskih besed primernih za upravljanje televizorja. Za to množico izdelajte akustični model in podatkovno bazo govornih posnetkov, na kateri izvedite učenje razpoznavalnika. Razvijte in izdelajte tudi elektronski vmesnik, ki razpoznane besede pretvori v ustrezni infrardeči signal, s katerim se daljinsko krmili televizor. Glasovno upravljanje televizorja preizkusite in izmerite uspešnost njegovega delovanja.

Mentor:
prof. dr. Dušan Kodek

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:
prof. dr. Nikolaj Zimic

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Katja Cetinski

GLASOVNO UPRAVLJANJE TELEVIZIJE Z UPORABO SISTEMA SPHINX-4

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor:
prof. dr. Dušan Kodek, univ. dipl. ing.

Ljubljana, 2010

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisana **Katja Cetinski**,

z vpisno številko **63050437**,

sem avtorica diplomskega dela z naslovom:

Glasovno upravljanje televizije z uporabo sistema Sphinx-4

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek) **prof. dr. Dušana Kodeka**
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 03.11.2010 Podpis avtorice:

ZAHVALA

Rada bi se zahvalila svojemu mentorju prof. dr. Dušanu Kodeku, za strokovno pomoč in vodenje pri izdelavi diplomske naloge. Zahvaljujem se tudi asistentu dr. Robertu Rozmanu za spodbudo in koristne nasvete.

Največja zahvala gre mojim staršem za vso podporo in finančno pomoč v času študija, ter mojemu Mateju za moralno podporo in ideje.

Hvala tudi vsem ostalim, ki ste na kakršen koli način prispevali k nastanku diplomske naloge, oziroma ste mi stali ob strani.

KAZALO

SEZNAM UPORABLJENIH KRATIC

HMM	prikriti modeli Markova (angl. Hidden Markov Model)
CI	kontekstno neodvisni modeli (angl. Context Independent Models)
CD	kontekstno odvisni modeli (angl. Context Dependent Models)
PC	osebni računalnik (angl. Personal Computer)
USB	univerzalno serijsko vodilo (angl. Universal Serial Bus)
IDE	integrirano okolje za razvoj (angl. Integrated Development Environment)
XML	razširljiv označevalni jezik (angl. Extensible Markup Language)
CMU	Univerza Carnegie Mellon (angl. Carnegie Mellon University)

POVZETEK

Namen diplomske naloge je bila izdelava sistema za govorno upravljanje televizije.

Za razpoznavo ukazov sem uporabila sistem Sphinx-4. Posebej za ta namen sem izdelala lastno govorno bazo, saj prosto dostopnih posnetkov slovenskega govora močno primanjkuje. S pomočjo orodja NetBeans sem nato v programskem jeziku Java razvila aplikacijo, ki razpozna določen ukaz in ga preko serijskih vrat pošlje na razvojno ploščico Arduino. Glede na prejet podatek pa Arduino nato pošlje primeren IR signal televiziji.

Če povzamem vsebino diplomske naloge se na začetku posvetimo predvsem teoretičnemu ozadju. V uvodu se seznanimo s pojmom razpoznava govora ter s splošnim sistemov za razpoznavo govora. Sledijo teoretične osnove o prikritih modelih Markova, kateri se uporabljajo kot eden izmed pristopov pri razpoznavi govora.

V nadaljevanju se seznanimo s sistemom Sphinx-4, sistemom za učenje SphinxTrain in na koncu še z razvojno ploščico Arduino. S tem izvemo ključne stvari, ki so potrebne za razumevanje delovanja takšnega sistema. V drugi polovici si lahko ogledamo podrobno shemo sistema in se obenem seznanimo s praktično izdelavo sestavnih delov. Na koncu si v razpredelnici ogledamo rezultate testiranj pri dejanski uporabi sistema. Sledijo priloge, ki vsebujejo programsko kodo celotnega sistema. Celotna programska koda se nahaja na zgoščenki na zadnji strani diplome, nekaj pomembnejših delov pa tudi v poglavju priloge v obliki teksta.

Ključne besede:

razpoznava govora, Sphinx-4, televizija, Arduino

ABSTRACT

The main purpose of this diploma thesis was development of a system for voice controlled TV set.

I used Sphinx-4 system for recognition of television commands. Because there is a big problem of getting free Slovenian speech recordings, I created my own speech database. I used NetBeans IDE for developing my application, which is written in Java programming language. Application recognizes a certain voice command and then sends corresponding data to Arduino development board. According to received data, Arduino sends appropriate IR signal to television.

The content of diploma thesis is divided into two parts. First part is mainly theoretical. We learn the basis of speech recognition and general recognition system. This is followed by theoretical basis of Hidden Markov Models which are one of the principles used in speech recognition systems. Next we learn about Sphinx-4 system and SphinxTrain. The last one is used for acoustic model learning. The first part of diploma thesis ends with a description of Arduino development board. That way we learn the basic things that helps us understand the whole system.

The second part describes the development of a speech controlled television system. It includes testing and conclusion. Some smaller parts of source code are written in contents chapter, others can be found on a CD that is part of the thesis.

Key words:

speech recognition, Sphinx-4, television, Arduino

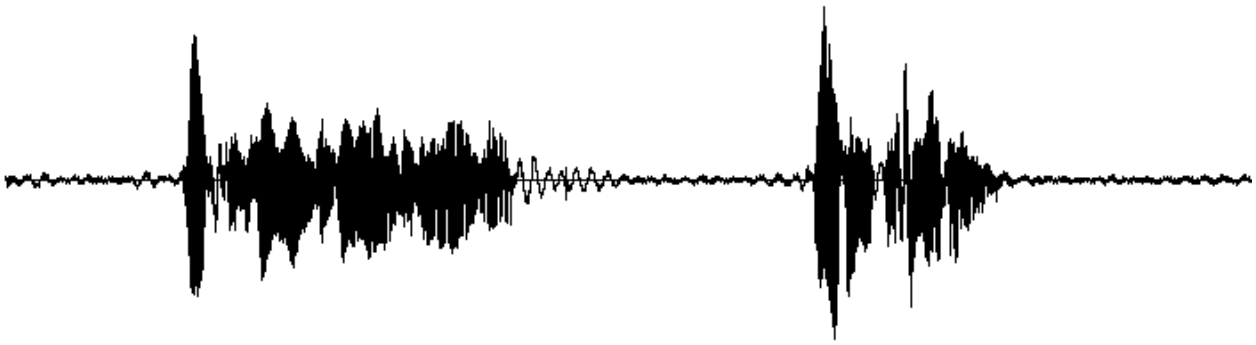
1. UVOD

Običajno televizijo upravljamo z daljinskim upravljalcem s pomočjo tipk. Namen te diplomske naloge pa je izdelati sistem, s pomočjo katerega lahko televizijo upravljamo z govorom – najbolj naravnim načinom komunikacije za človeka. To pomeni menjavo programov, spreminjanje glasnosti in podobne funkcije brez nepotrebnega vstajanja oziroma nasploh brez premikanja.

Diplomska naloga se glede na vsebino uvršča na področje avtomatske razpoznave govora in s tem tudi na področje digitalnega procesiranja signalov. Razpoznava govora v grobem pomeni pretvorbo človeškega govora (stavkov, ki jih posnamemo preko mikrofona) v tekstovni zapis, na podlagi katerega lahko izvedemo kakšno funkcijo ali pa samo shranimo govor v tekstovno obliko, kar je alternativa uporabi tipkovnice in miške.

Sistem je v grobem razdeljen na dva dela – prvi del se ukvarja s pošiljanjem ustreznih IR signalov na sprejemnik televizije (to je realizirano s pomočjo razvojne ploščice Arduino), drugi del pa se ukvarja z razpoznavo uporabnikovih ukazov (npr. tišje, ugasni,…), kar se izvaja na računalniku. Najprej se bom posvetila razpoznavi govora.

Govor je akustični signal, ki ga ljudje uporabljamo za sporazumevanje in velja za zelo kompleksen pojav. Ponavadi si predstavljamo, da je sestavljen iz besed, vsaka beseda pa iz fonemov, vendar to ni popolnoma res. Govor je namreč dinamični proces, kjer posamezni deli niso popolnoma ločeni med sabo. Kot primer poglejmo sliko govornega posnetka.



Slika 1. Primer govornega posnetka.

Govor je podvržen številnim dejavnikom in je zato močno variabilen. Nekateri izmed dejavnikov so na primer fiziološke karakteristike govorca, čustveno razpoloženje govorca, okolje v katerem poteka govor, neenakomerno spreminjanje hitrosti izgovorjave glasov in podobno.

Vse današnje definicije govora temeljijo na neki verjetnosti – to pomeni da ne obstajajo začrtane meje med posameznimi deli govora. Proces pretvorbe govora v besedilo zato nikoli ni 100%

pravilen. To prinaša precej zmede pri razvijalcih programske opreme, saj so vajeni dela z deterministčnimi sistemi, kar pa razpoznava govora vsekakor ni.

Razpoznava govora je torej v grobem proces preslikave akustičnega signala v tekstovni zapis.

Zadnji štirje odstavki so povzeti po [1]. Preostali del uvoda je večinoma povzet po [2].

1.1 Uporaba razpoznave govora

- narekovanje besedila (diktacija);
- sistemi za samodejno poizvedovanje preko telefona (telefonske centrale);
- upravljanje s stroji;
- komunikacija z računalnikom in drugimi napravami za invalidne, poškodovane ali bolne osebe.

1.2 Prednosti razpoznave govora

- omogoča mobilnost uporabnikov;
- omogoča komunikacijo s strojem za večji krog uporabnikov;
- omogoča večjo storilnost in človeku prijaznejše delovno okolje;
- za človeka je to najnaravnejši način komunikacije.

1.3 Slabosti razpoznave govora

- možnost nezanesljivega delovanja (hitrost govorjenja, šum okolice, narečje govorca);
- potrebno je uvajanje uporabnikov na nov način komunikacije;
- sisteme je potrebno vsaj v določeni meri razviti za vsak jezik posebej.

1.4 Razvrstitev sistemov za razpoznavo govora

- glede na način razpoznavanja govora (tekoči govor ali izolirane besede);
- glede na število govorcev (en govorec ali več govorcev);
- glede na število besed v slovarju (majhen, srednje velik, velik slovar).

Pri razpoznavanju izoliranih besed morajo biti besede izgovorjene tako, da lahko določimo začetek in konec besede. To pomeni, da morajo biti premori med besedami, izgovorjenimi v stavku, dolgi vsaj 200 ms. Takšni sistemi so preprosti, vendar je njihova uporaba omejena na aplikacije, kjer posamezna beseda že predstavlja ukaz in kjer odzivni čas ni tako pomemben.

Razpoznavanje tekočega govora je najbolj zapleten problem. Sistem mora namreč znati procesirati neznane časovne meje v govornem signalu (ločevanje besed) in pravilno obvladovati vse koartikulacijske pojave ter slabo izgovorjavo, ki je prisotna v tekočem govoru.

Pri sistemu avtomatskega razpoznavanja govora enega govorca mora sistem naučiti določen govorec – tisti, ki sistem pozneje tudi uporablja. Na tak način lahko dosežemo visoko stopnjo zanesljivosti razpoznavanja.

Pri sistemu razpoznavanja govora z več govorci, moramo imeti bazo izgovorjav neke množice govorcev. Sistem lahko uporabljajo tudi govorci, ki niso bili v učni množici – namen je namreč izdelati čimbolj neodvisen sistem.

Sistemi z majhnim slovarjem so tisti, kjer slovar ne preseže 99 besed. Uporabimo jih lahko npr. v aplikacijah preverjanja številk kreditnih kartic, razpoznavanju telefonskih številk in podobno. Slovarji sistemov s srednje velikim slovarjem vsebujejo med 100 in 999 besed. Največkrat se uporabljajo kot eksperimentalni laboratorijski sistemi za razvoj sistemov razpoznavanja tekočega govora. Sistemi z velikim slovarjem imajo slovarje velikosti 1000 in več besed. Takšni slovarji so običajno dosegljivi na tržišču za uporabo na osebnih računalnikih in gre za sisteme razpoznavanja tekočega govora odvisnega govorca ali razpoznavanje izoliranih besed neodvisnega govorca.

1.5 Orodja

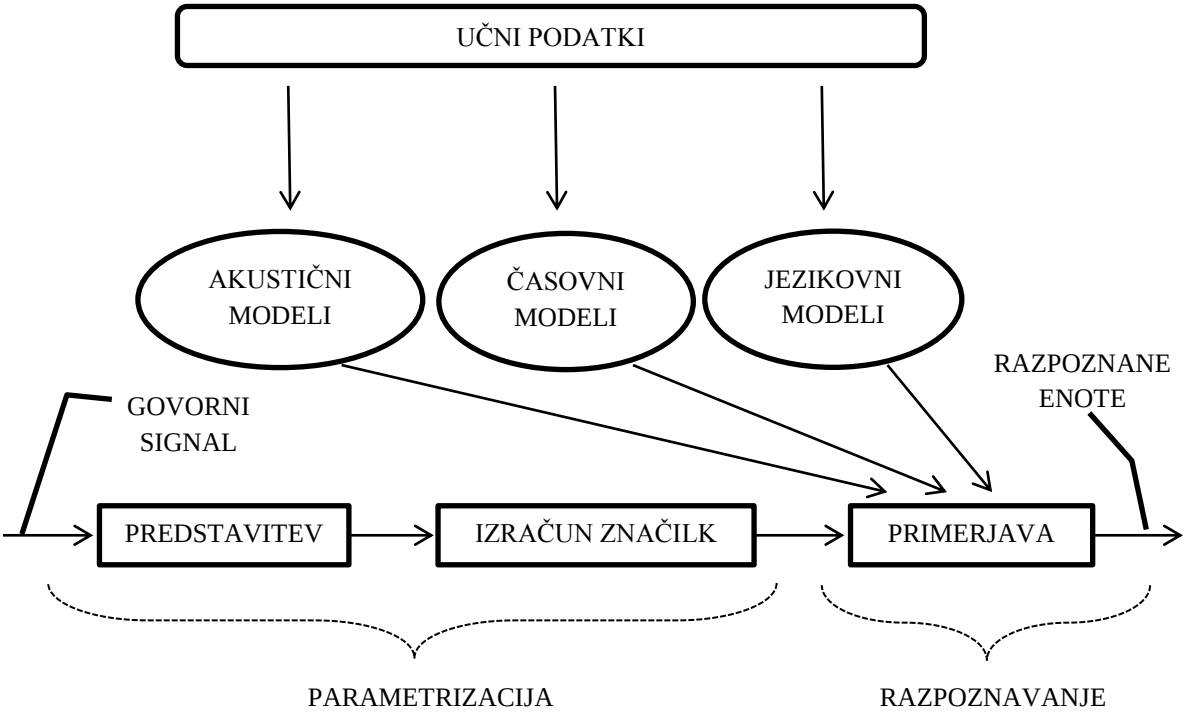
- odprtokodna orodja
    - CMU Sphinx (SphinxI, SphinxII, SphinxIII, SphinxIV, SphinxTrain);
    - HTK (Hidden Markov Model Toolkit);
    - ISIP Toolkit;
    - Julius;
    - Xvoice (IBM);
    - Open Mind Speech.

- komercialna orodja
    - Dragon NaturallySpeaking;
    - IBM Via Voice;
    - Vocalis Speechware;
    - iListen;
    - Entropic.

1.6 Splošni sistem za razpoznavo govora

Opis splošnega sistema za razpoznavo govora sem povzela po [3].

Zgradbo splošnega sistema za razpoznavanje govora prikazuje slika 2. Razpoznavanje govora v ožjem smislu lahko razdelimo na dva glavna koraka. V prvem se pripravi kompakten, reprezentativen opis vhodnega govornega signala – t.j. proces parametrizacije. V njem se najpogosteje uporablja kratkočasovna Fourierova transformacija skupaj z ostalimi tehnikami s področja digitalnega procesiranja signalov. Iz vzorcev vhodnega govornega signala se v postopku frekvenčne ali (in) časovne analize najprej oblikuje začetna predstavitev signala. Ta predstavlja osnovo za nadaljnje postopke računanja končnega zaporedja vektorjev značilk[1], ki vstopa v proces razpoznavanja oziroma primerjave.



Slika 2. Zgradba splošnega sistema za razpoznavo govora.

[1] Končno množico parametrov, s katerimi opišemo končni izsek nekega signala imenujemo tudi vektor značilk, posamezne komponente pa značilke.

Dobljen opis se v drugem koraku primerja z že prej pripravljenimi referenčnimi opisi[1] oziroma modeli – t.j. proces razpoznavanja. Najprej se vhodni opis na osnovi različnih kriterijev (statistična podobnost, spektralna razdalja, …) primerja z akustičnimi referenčnimi modeli, ki opisujejo tovrstne značilnosti posameznih govornih enot v slovarju sistema. Zaporedje posameznih klasifikacij se nato primerja s časovnimi modeli, ki opisujejo dinamiko spreminjanja akustičnih značilnosti govora. Dobljena ocena časovnega ujemanja se na koncu ovrednoti še v jezikovnih modelih, ki opisujejo relacije elementarnih govornih enot v daljših zaporedjih (npr. besede, fraze, stavki, povedi). Za končni rezultat razpoznavanja se izbere najboljša (najverjetnejša) hipoteza in njej ustrezno zaporedje govornih enot.

V procesu primerjave se uporabljajo tehnike in metode, ki temeljijo na različnih pristopih.

Trenutno sta daleč najpogostejša le dva formalna koncepta in sicer:

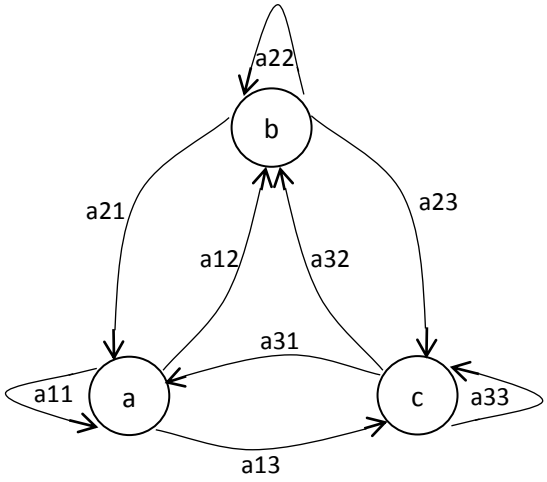
- statistični modeli, ki opisujejo globalne lastnosti signalov in podobnosti med trenutnimi in referenčnimi opisi signalov;
- mrežni modeli, ki so sestavljeni iz večjega števila enostavnih, med seboj povezanih vozlišč in razvrščajo (klasificirajo) vhodne vzorce v predhodno naučene razrede.

Statistični pristop je, predvsem po zaslugi prikritih modelov Markova[2], na tem področju prisoten že zelo dolgo. Kljub nekaterim pomanjkljivostim je še vedno najbolj razširjen. Ker ta pristop uporablja tudi sistem Sphinx4, katerega bom uporabila za razpoznavanje govora, se bom v nadaljevanju posvetila le tovrstnim sistemom.

[1] Opisi, ki se pripravijo v postopku učenja.
[2] V nadaljevanju HMM ali angl. Hidden Markov Model.

2. PRIKRITI MODELI MARKOVA

Definicija prikritega modela Markova je povezana z Markovim procesom. Vsak naključni časovno diskretni proces, kjer je verjetnost vsakega dogodka odvisna od j prejšnjih dogodkov, se imenuje Markov proces j-tega reda.



Slika 3. Model Markova za Markov proces prvega reda.

Markove procese prvega reda predstavljamo z diagramom prehajanja stanj – takšnim diagramom pravimo modeli Markova. Na sliki 3 si lahko ogledamo model Markova za Markov proces prvega reda. Model predstavlja naključni proces, kjer se v določenih časovnih presledkih pojavljajo trije znaki a, b in c. Pri tem velja, da je verjetnost pojavljanja določenega znaka odvisna samo od prejšnjega znaka.

Proces opišemo s pogojnimi verjetnostimi P(x|y), pri čemer P(x|y) označuje verjetnost, da se je po znaku y pojavil x. V modelu so prehodi označeni s puščicami, številke ob posameznih puščicah pa predstavljajo verjetnost posameznega prehoda.

Model Markova zapišemo z matriko prehodnih verjetnosti

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \qquad (1)$$

pri čemer element a_{ij} podaja verjetnost prehoda iz stanja i v stanje j. Vsota vseh verjetnosti v posamezni vrstici je enaka 1, to pomeni, da kateremukoli simbolu zagotovo sledi eden izmed treh simbolov.

Tako definiran model Markova predstavlja matematični model naključnega časovno diskretnega procesa pojavljanja znakov a, b in c. Za modeliranje kompleksnejših procesov pa tak model ni uporaben. Zmožnost modeliranja naključnih procesov zelo povečamo, če ob verjetnostih prehodov med stanji uvedemo še emisijske verjetnosti. Število stanj sedaj ni več povezano s številom simbolov. Sedaj lahko vsako stanje odda katerikoli znak in zaporedja stanj ne moremo več neposredno povezati z zaporedjem znakov. Zaporedje stanj je prikrito in zato takšen tip modela imenujemo prikriti model Markova (HMM).

Prikriti model Markova opišemo z matriko prehajanja stanj A in z matriko emisijskih verjetnosti B. Ob omenjenih matrikah, pa je pri prikritem modelu Markova zelo pomembna tudi začetna porazdelitev stanj π. Ta določa verjetnost, da se bo modeliranje začelo v določenem začetnem stanju. Nabor parametrov, s katerim opišemo prikriti model Markova, označimo z λ, parametre pa združimo v izrazu:

$$\lambda = (A, B, \pi). \tag{2}$$

2.1 Osnovni postopki

Pri modeliranju stohastičnega procesa, ki generira zaporedje X

$$X = \{x_1, x_2, \dots, x_t, \dots, x_T\}, \tag{3}$$

pri čemer x_t označuje vrednost naključne spremenljivke x v trenutku t, se srečamo z naslednjimi tremi problemi:

1. Problem ocenjevanja – kako oceniti verjetnost P(X|λ) dogodka, da je neki HMM, opisan z naborom parametrov $\lambda = (A, B, \pi)$, generiral opazovano zaporedje X.
2. Problem učenja – kako za dano zaporedje X določiti parametre HMM $\lambda = (A, B, \pi)$, ki bi zagotavljali največjo verjetnost P(X|λ).
3. Problem dekodiranja – kako pri danem zaporedju X določiti najverjetnejše zaporedje stanj
$$S = S_1, S_2, \dots, S_t, \dots, S_T, \tag{4}$$
pri čemer S_t označuje stanje HMM, v katerem se nahajamo v trenutku t.

S prvim problemom se srečamo pri razpoznavanju zaporedij, ko želimo ugotoviti, kateri izmed znanih modelov je najverjetneje generiral opazovano zaporedje, oziroma, kako ustreza opazovano zaporedje posameznim modelom.

Drugi problem zadeva postopek učenja HMM, ko želimo z danim zaporedjem, ki ga imenujemo učno zaporedje, ustrezno določiti parametre modela. Problem učenja ima ključen pomen pri veliki večini uporab HMM. Problem dekodiranja pa predstavlja razkrivanje zaporedja stanj v modelu. Pri HMM ne moremo govoriti o pravilnem zaporedju stanj, kot to velja za osnovni model Markova. Govorimo lahko le o najbolj verjetnem zaporedju, saj lahko različna zaporedja stanj generirajo opazovano zaporedje, vendar so posamezna zaporedja stanj različno verjetna.

2.1.1 Problem ocenjevanja

Imamo zaporedje $X = \{x_1, x_2, \dots, x_t, \dots, x_T\}$ in želimo ugotoviti, kako verjetno je, da je model z naborom parametrov $\lambda = (A, B, \pi)$ generiral opazovano zaporedje X, oziroma želimo določiti verjetnost P(X|λ).

Najbolj neposreden način je, da upoštevamo vsa možna zaporedja stanj dolžine T, ki določajo poti skozi model λ. Ena izmed možnih poti je

$$\Theta = \theta_1, \theta_2, \dots, \theta_t, \dots, \theta_T, \tag{5}$$

pri tem θ_t označuje stanje modela v trenutku t, θ_1 pa začetno stanje.

Iskano verjetnost P(X|λ), da je model λ generiral opazovano zaporedje X, dobimo na naslednji način

$$P(X|\lambda) = \sum_{vse\ \Theta} P(X|\Theta, \lambda) P(\Theta|\lambda) = \sum_{vse\ \Theta} \pi_{\theta_1} b_{\theta_1}(x_1) \prod_{t=2}^{T} a_{\theta_{t-1}\theta_t} b_{\theta_t}(x_t). \tag{6}$$

Iz zgornje enačbe lahko razberemo, da problem ocenjevanja rešimo tako, da za izbrano pot Θ po modelu λ in za vsak t zmnožimo emisijsko verjetnost $b_{\theta_t}(x_t)$ z verjetnostjo prehoda $a_{\theta_{t-1}\theta_t}$, ter seštejemo prispevke z vseh možnih poti. Takšen postopek je računsko zelo zahteven, saj je vseh možnih poti skozi model izredno veliko tudi pri majhnem številu stanj. Ravno zaradi tega razloga se veliko uporabljata postopka “naprej-nazaj” in “Viterbijev postopek”.

2.1.2 Problem učenja

Pri danem zaporedju $X = x_1, x_2, \dots, x_t, \dots, x_T$ želimo določiti parametre $\lambda = (A, B, \pi)$ tako, da dosežemo največjo verjetnost P(X|λ) – verjetnost, da je model λ generiral opazovano zaporedje X. Problem učenja je najtežji izmed vseh treh omenjenih problemov. Ne poznamo namreč nobene analitične metode določanja optimalnega nabora parametrov modela, obstaja pa nekaj iterativnih metod, ki zagotavljajo lokalno maksimiziranje verjetnosti P(X|λ). Za učenje se najpogosteje uporabljata postopka:

- Baum-Welchov postopek učenja in
- Viterbijev postopek učenja.

2.1.3 Problem dekodiranja

Pri modeliranju naključnega procesa s HMM ostane pot skozi model prikrita. Pri osnovnem modelu Markova je zaporedje stanj enoumno določljivo iz opazovanega zaporedja X. Pri HMM pa lahko vsako zaporedje stanj generira opazovano zaporedje, zato lahko govorimo le o zaporedju stanj, ki je najverjetneje generiralo opazovano zaporedje X, oziroma o optimalnem zaporedju stanj. Pri tem pa se srečamo s problemom, kako definirati optimalno zaporedje stanj.

Najpogosteje je optimalno zaporedje definirano kot tisto zaporedje stanj $\Theta^* = \Theta_1^*, \Theta_2^*, \dots, \Theta_T^*$ modela λ, ki da največjo verjetnost modela $P(X, \Theta^* | \lambda)$. Za iskanje takšnega zaporedja je najbolj primeren Viterbijev postopek. Problem iskanja zaporedja stanj lahko rešujemo vzporedno z reševanjem problema ocenjevanja.

2.2 Tipi HMM

2.2.1 Diskretni HMM

Naj bo X zaporedje vektorjev značilk, ki smo ga z vektorsko kvantizacijo pretvorili v zaporedje simbolov. Naj x_t predstavlja indeks particije, v katero smo z vektorsko kvantizacijo razvrstili vhodni vektor značilk v času t. Govorni signal smo tako predstavili kot zaporedje indeksov particij. S HMM želimo modelirati dobljeno zaporedje simbolov. Ker smo s postopkom kvantizacije diskretizirali zvezni prostor vektorjev značilk, pravimo modelom, s katerimi modeliramo tako dobljena zaporedja, diskretni HMM. V vsakem stanju modela definiramo za vsakega izmed L simbolov emisijsko verjetnost $b_i(b)$, ki pove, kolikšna je verjetnost, da je model v stanju S_j oddal simbol b.

Diskretne HMM opišemo z naborom parametrov $\lambda = (A, B, \pi)$. Za njih veljajo vsi omenjeni postopki za ocenjevanje, učenje in dekodiranje.

2.2.2 Zvezni HMM

Pri diskretnih HMM vnesemo z vektorsko kvantizacijo v postopek modeliranja naključnega procesa kvantizacijsko napako, ki slabša njihovo uspešnost.

Druga možnost uporabe HMM je, da z njimi neposredno modeliramo zaporedje vektorjev značilk, brez uporabe vektorske kvantizacije – pri tem želimo čim bolj zvesto modelirati razpršenost oziroma porazdelitev elementov vektorjev značilk. Razpršenost elementov je posledica že omenjenih dejavnikov, kot so: govor neodvisnega govorca, psihofizično stanje govorca, vpliv prenosnega kanala itd. Elementi vektorja podajajo normirane maksimalne amplitude frekvenčnega spektra v posameznih frekvenčnih pasovih.

V vsakem stanju je za vsak element x_{t_n}, vektorja značilk $\overrightarrow{x_t}$

$$X = (\overrightarrow{x_1}, \overrightarrow{x_2}, \dots, \overrightarrow{x_T}), \tag{6}$$

$$\overrightarrow{x_1} = (x_{t_1}, x_{t_2}, \dots, x_{t_N}), \tag{7}$$

definirana funkcija porazdelitve verjetnosti $\chi_n(x_n)$, ki opisuje, kolikšna je verjetnost, da posamezni element vektorja značilk zavzame določene vrednosti. Emisijska verjetnost vektorja značilk $\overrightarrow{x_t}$ v stanju s modela i je za posamezne elemente vektorja značilk izražena s funkcijami porazdelitve verjetnosti $\chi^i_{s_n}(x_n)$

$$b^i_s(\overrightarrow{x_t}) = p(\overrightarrow{x_t}|s, \lambda_i); \tag{8}$$

$$\begin{aligned} p(x_1) &= \chi^i_{s_1}(x_{t_1}), \\ p(x_2) &= \chi^i_{s_2}(x_{t_2}), \\ &\vdots \\ p(x_n) &= \chi^i_{s_n}(x_{t_n}), \\ &\vdots \\ p(x_N) &= \chi^i_{s_N}(x_{t_N}). \end{aligned} \tag{9}$$

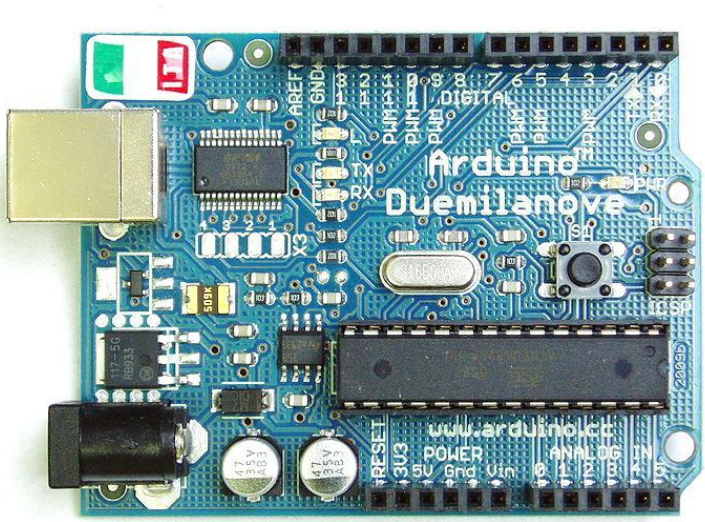
Tako definiran model imenujemo zvezni HMM. Funkcija porazdelitve verjetnosti $\chi_{s_n}^{i}(x_n)$ predstavlja model dejanske porazdelitve vrednosti n-tega elementa vektorjev značilk. Izbrati želimo takšen model, ki bo čim bolj zvesto predstavil dejansko porazdelitev elementov vektorjev značilk, hkrati pa ne bo preveč računsko zapleten, saj zapletenost modela porazdelitve močno vpliva na računsko zahtevnost modeliranja z zveznimi HMM.

3. RAZVOJNA PLOŠČICA ARDUINO

Pri realizaciji upravljanja z različnimi funkcijami televizije sem uporabila razvojno ploščico Arduino. Opis slednje sem povzela po [4].

Arduino je enostavno razvojno okolje. Vhodno-izhodno vezje z mikrokrmilnikom programiramo s programskim jezikom, ki se zgleduje po jeziku Wiring ter v razvojnem okolju, ki je začetnikom prijazno in izhaja iz projekta Processing.

S pomočjo Arduina lahko razvijamo interaktivne predmete, ki na podlagi branja senzorjev in stikal kontrolirajo lučke, motorje in podobne izhode. Vezje deluje samostojno, lahko pa ga povežemo s programi, ki tečejo na PC-jih. Sestavimo ga lahko tudi sami ali pa kupimo že narejenega. Razvojno okolje je odprtokodno in ga lahko brezplačno prenesemo iz spleta.



Slika 4. Arduino Duemilanove.

Ploščica, katere srce je že rahlo zastarel mikrokontroler ATmega8, ima 13 digitalnih nožic[1], od katerih sta dva za RS232 komunikacijo, 6 od njih pa podpira tudi PWM. Poleg 10 digitalnih imamo na razpolago še 6 analognih 10-bitnih vhodov. Programiranje in napajanje je uporabniku zelo prijazno saj se ploščico priključi kar na USB[2] (na PC-ju se predstavlja prek FTDI[3] čipa in

[1] angl. pin
[2] univerzalno serijsko vodilo, angl. Universal Serial Bus
[3] Future Technology Devices International - podjetje, ki se ukvarja s polprevodniškimi napravami

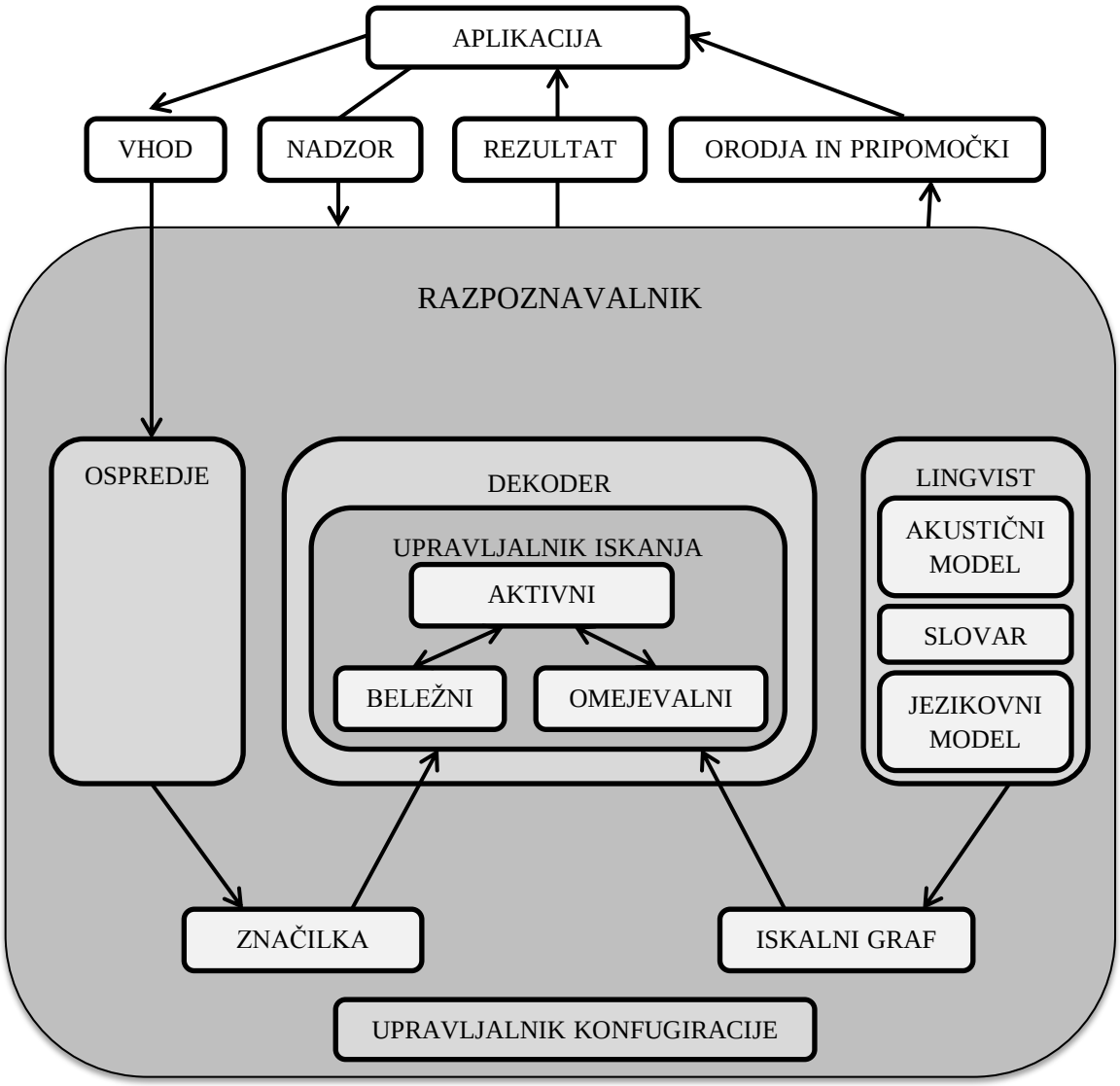
usb->serial driverja[1], kot serijska naprava). Tako se napajanje in programiranje izvaja kar preko USB-ja.

Pri skupnosti, ki razvija Arduino, so razvili tudi posebej prirejen odprtokodni IDE za vse 3 večje operacijske sisteme Windows, Linux in Mac OS X. Program ima trenutno verzijo 0.18, kar uporabnika sicer ne navdaja z zaupanjem v program, vendar se v praksi odlično obnese. Preko tega programiramo v nekoliko prirejenem jeziku C++ in programsko kodo enostavno nalagamo na mikroprocesor. Program ima še eno zelo uporabno funkcijo, saj vsebuje monitor serijskih vhodov, tako da lahko nadziramo kaj mikroprocesor prejema oziroma pošilja preko RS232 komunikacije.

[1] gonilnik, angl. driver

4. OPIS SISTEMA SPHINX-4

CMU Sphinx ali krajše Sphinx je skupno ime za skupino sistemov za razpoznavanje govora, ki so bili razviti na univerzi CMU[1], ki se nahaja v ameriški zvezni državi Pennsylvania. V to skupino med drugim spada tudi sistem Sphinx-4 ter sistem za učenje akustičnih modelov SphinxTrain, ki je opisan v naslednjem poglavju.



Slika 5. Zgradba sistema Sphinx-4.

[1] Carnegie Mellon University

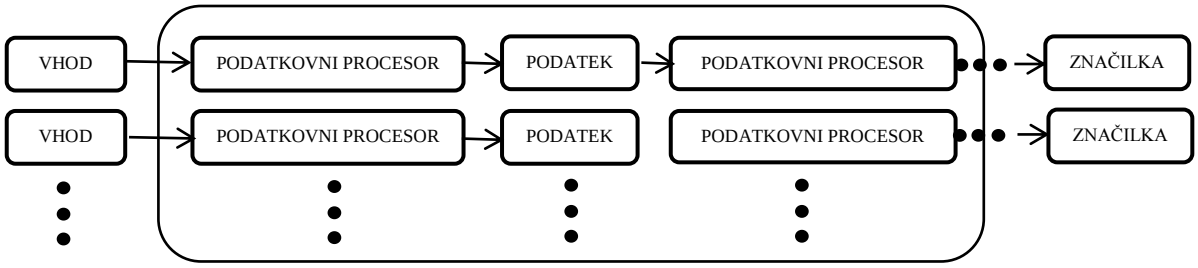
Sphinx-4 je prilagodljiv odprtokodni sistem za razpoznavo govora, ki uporablja HMM pristop in je v celoti napisan v programskem jeziku Java. [6]

Kot je razvidno iz slike 5, sistem Sphinx-4 sestavljajo trije moduli:

- ospredje (javanski razred FrontEnd);
- lingvist (javanski razred Linguist);
- dekoder (javanski razred Decoder).

3.1 Ospredje

Naloga ospredja je parametrizacija enega ali več vhodnih signalov v zaporedje izhodnih značilk[1]. Kot je razvidno iz Slike 6, je ospredje sestavljeno iz ene ali več vzporednih verig objektov DataProcessor[2], pri čemer vsak od njih obdeluje vhodne signale. To omogoča sočasni izračun različnih parametrov istih ali različnih vhodnih signalov ter odpira vrata sistemom, ki lahko istočasno izvajajo funkcijo dekodiranja pri čemer uporabljajo različne tipe parametrov[3].



Slika 6. Zgradba ospredja.

Vsak objekt DataProcessor vsebuje vhod in izhod, ki lahko vodi v drug objekt DataProcessor, kar omogoča poljubno dolgo zaporedje posamezne verige. Vhod in izhod posameznega objekta DataProcessor je generični podatkovni objekt, ki enkapsulira tako obdelane vhodne podatke, kot tudi oznako za detekcijo končne točke. Komunikacija med objekti DataProcessor temelji na principu "povleci"[4]. Objekt DataProcessor zahteva izhod prejšnjega module le, ko potrebuje podatke. Model "povleci" omogoča medpomnenje in iskanje naprej in nazaj v času, zaradi česar lahko izvedeno okensko sinhronizirano Viterbijevo iskanje, iskanje v globino in še nekatere druge oblike iskanj.

[1] angl. features
[2] Javanski razred DataProcessor
[3] MFCC in PLP
[4] angl. pull design

Objekte DataProcessor med seboj povezujemo z upravljalnikom konfiguracije[1].

3.2 Lingvist

Naloga modula lingvist je izdelava iskalnega grafa[2].

Lingvist prevede katerikoli standardni tip jezikovnega modela[3] skupaj z informacijo o izgovorjavi, ki jo dobi iz slovarja[4] in informacijo o strukturi enega ali več akustičnih modelov, v iskalni graf.

Linvgist je sestavljen iz treh komponent:

- jezikovni model (javanski razred LanguageModel);
- slovar (javanski razred Dictionary);
- akustični model (javanski razred AcousticModel).

Jezikovni model predstavlja strukturo jezika na nivoju besede. To lahko predstavimo z dvema kategorijama slovnic – stohastičnim N-gram modelom ali slovarjem, predstavljenim z grafom. Prva določa verjetnosti pojavitve besed na podlagi opazovanja preteklih n-1 besed, pri drugi pa vsaka beseda tvori vozlišče, povezava med vozlišči pa predstavlja verjetnost prehoda. Uporabimo lahko več formatov jezikovnih modelov:

- razred SimpleWordListGrammar;
- razred JSGFGrammar;
- razred LMGrammar;
- razred FSTGrammar;
- razred SimpleNGramModel;
- razred LargeTrigramModel.

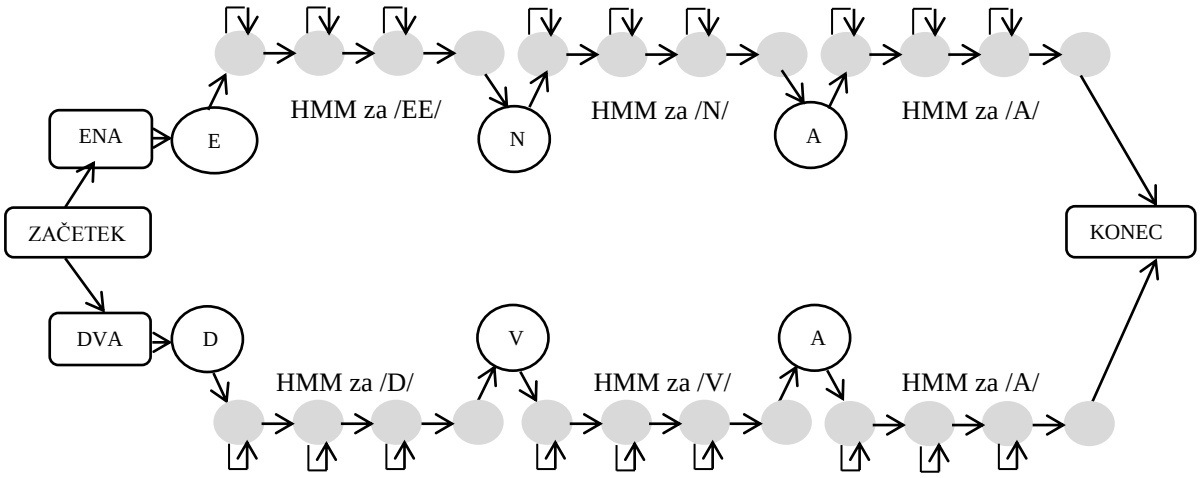
Slovar predstavlja izgovorjave posameznih besed v jezikovnem modelu. Izgovorjave razdelijo besede v zaporedja podbesednih enot, ki jih najdemo v akustičnem modelu.

Akustični model vsebuje preslikave med enotami govora in HMM, ki je dosežen glede na značilke. Preslikava lahko uporablja tudi podatke o kontekstnem in gramatičnem položaju. Kot primer lahko omenimo trifone, kjer predstavljata kontekst fonema, ki sta levo in desno od opazovanega fonema. Položaj besede določimo tako, da ugotovimo kje se nahaja trifon – na začetku, na sredini ali na koncu besede.

---

[1] Javanski razred ConfigurationManager
[2] Javanski razred SearchGraph
[3] angl. language model
[4] angl. dictionary

Linvgist kot že omenjeno vsako besedo razčleni na zaporedje kontekstno neodvisnih podbesednih enot. Podbesedne enote in njihov kontekst nato poda akustičnemu modelu ter nazaj dobi grafe HMM, ki so povezani s temi enotami. Lingvist nato uporabi dobljene grafe in zgradi iskalni graf.



Slika 7. Primer iskalnega grafa za besedi ena in dva.

Primer iskalnega grafa si lahko ogledamo na sliki 7. Graf je usmerjen in posamezno vozlišče, ki ga imenujemo tudi iskalno stanje[1], predstavlja emisijsko ali neemisijsko stanje. V emisijsko stanje lahko pridemo glede na akustične značilke, neemisijska stanja pa uporabljamo za predstavitev višjih lingvističnih konstruktov, kot so besede ali fonemi in do njih ne pridemo na podlagi akustičnih značilk. Povezave med stanji predstavljajo verjetnost prehoda iz enega stanja v drugo.

Sphinx-4 omogoča, da za različne naloge uporabimo različne implementacije lingvista:

- razred FlatLinguist;
- razred DynamicLinguist;
- razred LexTreeLinguist.

[1] Javanski razred SearchState

3.3 Dekoder

V splošnem je naloga dekoderja tvorba rezultatov. Pri tem mu je v pomoč najpomembnejša komponenta, ki se imenuje upravljalnik iskanja[1]. Slednji uporabi značilke, ki jih tvori modul ospredje in iskalni graf, ki ga tvori modul lingvist ter izvede dekodiranje, pri čemer se generirajo rezultati.

Dekoder naroči upravljalniku iskanja, naj razpozna niz okvirjev značilk. Na vsakem koraku tega procesa upravljalnik iskanja ustvari rezultat. Objekt rezultat pa vsebuje vse poti, ki so dosegle končno neemisijsko stanje.

Sphinx-4 omogoča štiri različne implementacije upravljalnika iskanja:

- razred SimpleBreadthFirstSearchManager;
- razred WordPruningBreadthSearchManager;
- razred BushderbySearchManager;
- razred ParallelSearchManager.

[1] Javanski razred SearchManager

5. OPIS ORODJA SPHINXTRAIN

SphinxTrain je odprtokodno orodje, ki se uporablja za učenje akustičnih modelov. Opis orodja in njegove uporabe sem povzela po [5].

Postopek učenja je pomnilniško zelo potraten, vendar moramo vedeti, da na obremenitev sistemskih virov vpliva več stvari:

- tip uporabljenega modela (zvezni model[1] ali polzvezni model[2]);
- velikost in konfiguracija vektorjev značilk;
- število HMM stanj.

Kot primer lahko navedemo učenje 5-stanjskih modelov za 10.000 trifonov. V tem primeru imamo torej 50.000 stanj. Če uporabljamo polzvezne HMM modele, potem vsakemu stanju pripada 1024 realnih števil. V primeru da so realna števila predstavljena s 4 bajti, potem to pomeni, da podatki zasedejo približno 205 MB medpomnilnika. Pri uporabi zveznih HMM modelov je vektor značilk običajno predstavljen z 39 komponentami. Vsakemu stanju pripada 39 povprečij in 39 standardnih odklonov Gaussovih funkcij. To pomeni, da je vsako stanje predstavljeno z 79 realnimi števili in podatki v primeru učenja zveznih HMM modelov za 10.000 trifonov zasedejo le 15,8 MB medpomnilnika. Torej lahko z enako velikostjo pomnilnika z zveznimi HMM modeli naučimo 12-krat več trifonov kot z uporabo polzveznih modelov.

4.1 Priprava podatkov

Preden začnemo z učenjem, moramo pripraviti naslednje datoteke:

1. Zvočne posnetke v .wav formatu.

2. Datoteke z značilkami, ki smo jih s pomočjo programa wave2feat (program je del SphinxTrain paketa) izračunali iz zvočnih posnetkov.

3. Kontrolno datoteko, ki vsebuje imena datotek z značilkami vključno s celotno potjo. Končnice datotek izpustimo (.mfc).
   mapa/podmapa1/stavek1;
   mapa/podmapa2/stavek2;
   mapa/podmapa2/stavek1.
4. Transkripcijsko datoteko, s katero opišemo vsebino datoteke z značilkami v enakem vrstnem redu.

[1] angl. Continuous Model
[2] angl. Semi-Continuous Model

<s> NAPREJ </s> (mapa/podmapa1/stavek1),
<s> UGASNI </s> (mapa/podmapa2/stavek2).

5. Glavni slovar, ki vsebuje vse besede in njihovo fonemsko predstavitev. Beseda ima lahko več možnih izgovorjav.
   SEDEM S E D EE M,
   SEDEM(2) S E D @ M.

6. Slovar mašil, ki vsebuje zvoke, ki niso besede (vzdih, pok, šum, tišina).
   <s> SIL,
   <sil> SIL,
   </s> SIL.
   Pri tem <s> pomeni tišino na začetku stavka, <sil> tišino vmes in </s> tišino na koncu stavka.

7. Datoteko s seznamom vseh fonemov, ki smo jih uporabili v glavnem slovarju.
   EE,
   A,
   CH.

4.2 Postopek učenja za zvezne modele

Učenje lahko izvedemo s pomočjo skripte RunAll.pl ali pa po korakih zaganjamo posamezne programe. Preden se lotimo učenja, moramo imeti pripravljene vse potrebne datoteke – to so datoteke, ki smo jih omenili zgoraj.

Učenje poteka po naslednjih korakih:

- **izdelava definicijske datoteke za kontekstno neodvisni model;**
  Prvi korak je izdelava datoteke, ki definira kontekstno neodvisni model. Namen tega koraka je enolični opis vsakega stanja HMM, ki ga bomo učili. Vsakemu stanju tako določimo številko in se kasneje nanj sklicujemo le s to številko. Korak se izvede s pomočjo programa *mk_model_def*.

- **izdelava datoteke s topologijo HMM;**
  V tem koraku kreiramo datoteko, ki vsebujo podatke o HMM topologiji. Korak izvedemo s pomočjo Perl skripte *make_topology.pl*. Slednja generira matriko, ki vsebuje elemente z vrednostmi 1.0 ali 0.0 – vsak element matrike torej pove, ali je določen prehod v HMM dovoljen ali ne.

Kot primer lahko navedemo 3-stanjski HMM, ki ne vsebuje stanj, ki bi jih lahko preskočili.

$$\begin{array}{cccc} & & 4 & \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \end{array} \qquad (10)$$

Številka 4 predstavlja število stanj HMM. Čeprav imamo 3-stanjski model, skripta avtomatsko zapiše še četrto stanje. To stanje ni emisijsko in iz njega ni prehodov v druga stanja.

Vrednost 1.0 torej pomeni, da je prehod iz stanja 1 nazaj v isto stanje dovoljen. Medtem ko vrednost 0.0 pomeni, da prehod ni dovoljen.

- **inicializacija parametrov HMM modela**

  Kontekstno neodvisne modele določajo naslednje datoteke:
  - ✓ *mixture_weights*: uteži, ki so podane za vsako Gaussovo porazdelitveno funkcijo skladno s stanjem;
  - ✓ *transition_matrices*: matrika z verjetnostmi prehodov med posameznimi stanji;
  - ✓ *means*: povprečja Gaussovih porazdelitvenih funkcij;
  - ✓ *variances*: standardni odkloni Gaussovih porazdelitvenih funkcij.

  Da lahko začnemo z učenjem kontekstno neodvisnih modelov, morajo biti zgoraj omenjene datoteke inicializirane, torej jim moramo določiti začetne vrednosti.

  Datoteki *mixture_weights* in *transition_matrices* inicializiramo s pomočjo programa *mk_flat*. Za inicializacijo datotek *means* in *variances*, se najprej ocenijo globalne vrednosti in se nato prekopirajo na ustrezna mesta v obe datoteki. Globalno povprečje se izračuna z uporabo vseh vektorjev, ki so vsebovani v datotekah z značilkami. Število vektorjev je ponavadi precej velika številka, zato je ta operacija razdeljena na več delov (na koliko, se odločimo sami).

  Sphinx nato zbere vektorje za vsak posamezni del operacije in jih zapiše v medpomnilnik. To izvedemo s pomočjo programa *init_gau*. Ko je zapis v medpomnilnik končan, se njegova vsebina uporabi za izračun globalnega povprečja, kar izvedemo s programom *norm*.

V naslednjem koraku je treba zbrati vektorje in vrednosti globalne variance, zato se ponovno zažene program *init_gau*. Ko so podatki zapisani v medpomnilnik, se s pomočjo programa *norm* normalizira vrednost globalne variance za vektorje značilk.

Ko imamo izračunano globalno povprečje in globalno varianco, se omenjeni vrednosti s pomočjo programa *cp_parm* preneseta v povprečje in varianco vsakega stanja v HMM. Program *cp_parm* mora biti zagnan dvakrat – prvič za kopiranje povprečij in drugič za kopiranje varianc.

- **učenje kontekstno neodvisnih modelov**
  Ko zaključimo z inicializacijo, smo pripravljeni na učenje akustičnih modelov. S pomočjo Baum-Welchovega algoritma[1] se v prejšnjem koraku inicializiran model ponovno oceni in spremeni. Gre za iterativen proces, zato moramo Baum-Welchov algoritem pognati večkrat (ponavadi je 5-8 krat dovolj).

  Nastaviti je potrebno še stopnjo konvergence, kar storimo v konfiguracijski datoteki za izdelavo akustičnih modelov. Vrednosti za stopnjo konvergence se gibajo med 0,1 in 0,001. Program, s katerim izvedemo pravkar opisan postopek, se imenuje *bw*. Na koncu je potrebno s pomočjo programa *norm* izvesti še normalizacijo.

- **izdelava definicijske datoteke za kontekstno odvisni ločen model**
  Za učenje kontekstno odvisnega ločenega modela, moramo najprej generirati definicijsko datoteko, ki vsebuje vse trifone, ki se pojavljajo v učni množici. To naredimo v naslednjih korakih:
  - ✓ najprej se s pomočjo slovarja generira seznam vseh trifonov;
  - ✓ v naslednjem koraku se generira začasni slovar, ki vsebuje vse besede z izjemo mašil z dodanim naslednjim zapisom:
    SIL SIL;
  - ✓ sedaj se izvede program *quick_count*, ki generira seznam vseh trifonov iz začasnega slovarja;
  - ✓ seznam se nato pretvori v definicijsko datoteko modela s programom *mk_model_def*;
  - ✓ v naslednjem koraku se s pomočjo programa *param_cnt* poišče vrednost, ki predstavlja število pojavitev posameznega trifona v seznamu, ki smo ga generirali v prejšnjem koraku.

[1] algoritem za iskanje neznanih parametrov HMM

- **inicializacija parametrov kontekstno odvisnega ločenega modela**
  Ko imamo definicijsko datoteko za kontekstno odvisni ločeni model, se lahko osredotočimo na inicializacijo parametrov modela. V tem koraku se generirajo štiri datoteke: *means, variances, transition_matrices* in *mixture_weights*. Za vsako od teh datotek se vrednosti najprej prekopirajo iz ustreznih datotek kontekstno neodvisnega modela. Korak izvedemo s programom *init_mixw*.

- **učenje kontekstno odvisnih ločenih modelov**
  Tako kot pri učenju kontekstno neodvisnih modelov, se tudi tukaj uporablja Baum-Welchov algoritem. Pri vsaki iteraciji se izvede program *bw*, kar se zaključi z normalizacijo (program *norm*).

- **gradnja odločitvenih dreves za skupno rabo parametrov**
  Odločitvena drevesa se uporabljajo za odločanje o tem, katera HMM stanja so si med sabo podobna, zato da se podatki iz vseh podobnih stanj združijo in definirajo posamezna globalna stanja, imenovana senoni. To so fonemski razredi, ki služijo za generiranje lingvističnih vprašanj.

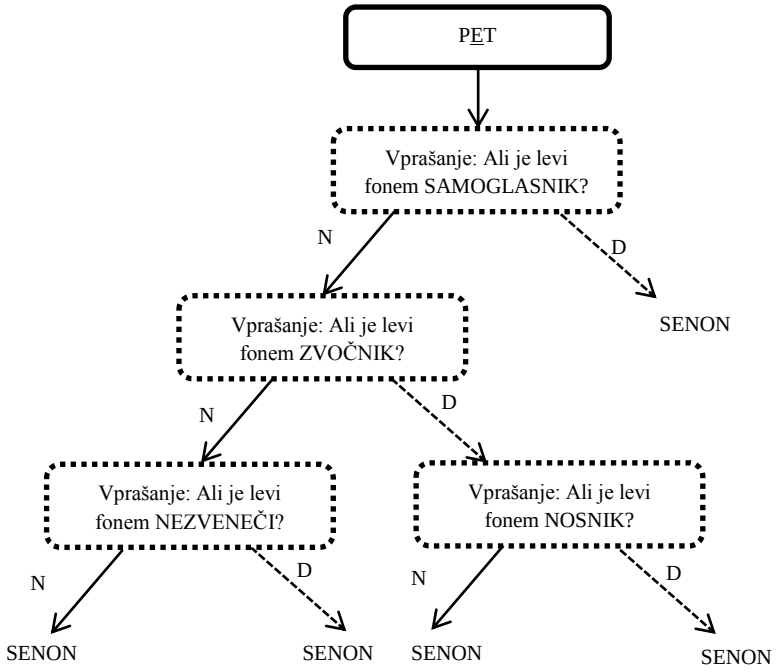
  Odločitveno drevo se začne s korenskim vozliščem, ki predstavlja osrednji fonem v opazovanem trifonu. Z lingvističnimi vprašanji glede fonemov, ki so levo in desno os srednjega fonema, se drevo razdeli v dve vozlišči. Listom pravimo tudi senoni[1] ali združena stanja[2] in njihovo število določi uporabnik. Vsa lingvistična vprašanja so zapisana v eni datoteki.

  Primer datoteke z lingvističnimi vprašanji:

SAMOGLASNIKI	A E EE I O U
ZVOCNIKI	M N R L V J
ZVENECI	B D G Z ZH
NEZVENECI	P T K C CH F S SH H
NOSNIKI	M N
USTNIKI	V L R J
SILENCE	SIL

  Levi stolpec vsebuje ime razreda, desni pa foneme, ki posameznemu razredu pripadajo.

[1] angl. senons
[2] angl. tied-state



Slika 8. Primer odločitvenega drevesa za besedo pet.

Če v našem modelu akustičnih enot ne predstavljajo fonemi, lahko lingvistična vprašanja generiramo avtomatično s programom *make_quests*. Ko so vprašanja generirana, se s programom *bldtree* zgradijo drevesa za posamezni fonem. Za mašila se odločitvena drevesa ne generirajo.

- **rezanje odločitvenih dreves**
  Ko enkrat imamo odločitvena drevesa, morajo biti še porezana, saj mora število listov ustrezati želenemu številu senonov, ki jih bomo učili. Za rezanje se uporabi program *prunetree*.

- **izdelava definicijske datoteke za kontekstno odvisni združeni model**
  Ko porežemo drevesa, moramo izdelati nove definicijske datoteke. Slednje vsebujejo vse trifone, ki se pojavijo med učenjem in so identificirani z ustreznimi senoni. Ta korak uporablja program *tiestate*.

- **inicializacija in učenje kontekstno odvisnih združenih modelov z mešanico Gaussov**
  Naslednji korak je učenje kontekstno odvisnih združenih modelov. V primeru zveznih modelov so lahko stanja HMM modelirana z eno Gaussovo porazdelitvijo ali mešanico Gaussovih porazdelitev. Če želimo modelirati z 8 Gaussovimi porazdelitvami, moramo najprej naučiti modele z 1 Gaussovo porazdelitvijo. Vsaka Gaussova porazdelitev je nato

razdeljena na dva dela, rezultat pa se uporabi za nadaljnje učenje modelov z 2 Gaussovima porazdelitvama. Postopek se ponavlja, dokler ne pridemo do želenega števila porazdelitev. Torej če želimo naučiti modele z 2^N Gaussovimi porazdelitvami, to naredimo v N+1 korakih. Vsak od teh korakov vsebuje:

- ✓ inicializacijo (program *init_mixw*);
- ✓ več iteracij Baum-Welchovega algoritma (program *bw*), ki mu sledi normalizacija (program *norm*);
- ✓ razcep Gaussovih porazdelitev (program *inc_comp* – samo do predzadnjega koraka).

6. IZDELAVA AKUSTIČNEGA MODELA

Pri izdelavi akustičnega modela sem sledila navodilom za uporabo orodja SphinxTrain, ki sem jih opisala v prejšnjem poglavju.

Najprej je bila na vrsti pridobitev zvočnih posnetkov za posamezne ukaze. Kot nabor ukazov za aktivacijo posameznih televizijskih funkcij sem izbrala naslednje besede:

- številke od 0 do 9 (predstavljajo posamezni televizijski program);
- naprej in nazaj (menjava programov);
- glasneje, tišje in tiho (spreminjanje glasnosti);
- prižgi in ugasni (vklop in izklop televizije).

Skupno torej akustični model zajema 17 besed. Sprva sem hotela pridobiti čimveč posnetkov od sorodnikov, prijateljev in sodelavcev, vendar se je izkazalo, da bi potrebovala veliko več časa, kot sem ga imela na razpolago. Za dobro razpoznavo večih govorcev[1] bi namreč potrebovala vsaj 5 ur zvočnih posnetkov, za razliko od razpoznave za enega samega govorca[2], kjer bi potrebovala samo 1 uro. Zaradi tega sem se odločila za naslednjo rešitev.

Podatkovno bazo sem napolnila z govornimi posnetki za enega samega govorca in sicer zase. Tako sem najhitreje prišla do kvalitetnih posnetkov v mirnem okolju. Sklepam tudi, da je v tem primeru razpoznavalnik bolj točen, kot bi bil, če bi baza vsebovala posnetke različnih govorcev.

Besede oziroma zaporedja besed sem posnela s prosto dostopnim programom za snemanje[3] v formatu WAV[4], s frekvenco vzorčenja 8000 Hz, s 16-bitno kvantizacijo in v enokanalnem načinu.

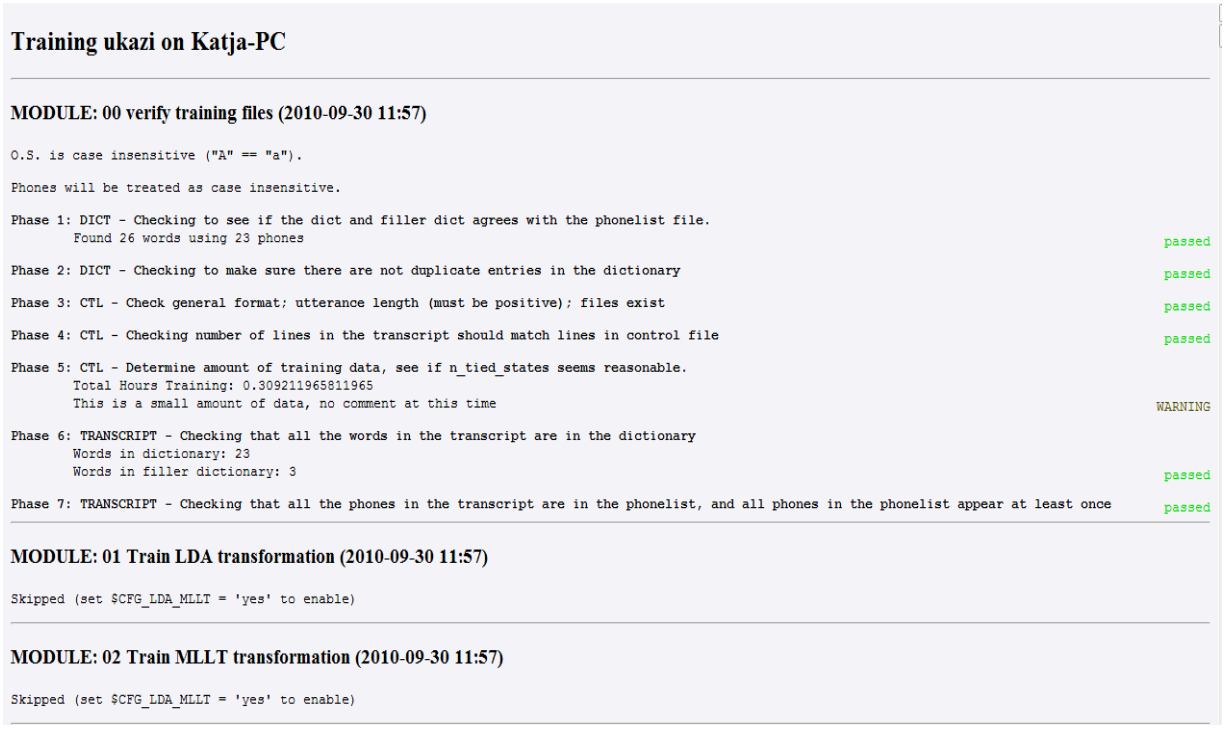
Kot sem že omenila je učenje časovno zelo zahteven postopek, zato sem preizkušanje parametrov učenja in njihov vpliv na uspešnost razpoznave preizkušala samo na delu učnih podatkov. Kar se tiče parametrov sem uporabila HMM s petimi stanji, število senonov pa sem zaradi majhne učne množice postavila na 500. Stopnja konvergence za Baum-Welchov algoritem znaša 0,004.

Preden sem pognala postopek učenja sem pripravila naslednje datoteke:

- glavni slovar in slovar mašil (prilogi 11.1 in 11.2);
- seznam vseh uporabljenih fonemov skupaj z mašili (priloga 11.3);

[1] angl. many speakers dictation
[2] angl. single speaker dictation
[3] Audio Recorder for Free 2010 12.8.1
[4] Waveform audio format

- transkripcije (opis vsebine datotek) in seznam vseh datotek (del datotek se nahaja v prilogah 11.4 in 11.5);
- datoteko z lingvističnimi vprašanji za gradnjo odločitvenih dreves (priloga 11.6);
- popravki v konfiguracijski datoteki za učenje, kot so na primer določitev števila HMM stanj in senonov.



Training ukazi on Katja-PC

MODULE: 00 verify training files (2010-09-30 11:57)

```
O.S. is case insensitive ("A" == "a").

Phones will be treated as case insensitive.

Phase 1: DICT - Checking to see if the dict and filler dict agrees with the phonelist file.
        Found 26 words using 23 phones                                                                  passed

Phase 2: DICT - Checking to make sure there are not duplicate entries in the dictionary                 passed

Phase 3: CTL - Check general format; utterance length (must be positive); files exist                   passed

Phase 4: CTL - Checking number of lines in the transcript should match lines in control file            passed

Phase 5: CTL - Determine amount of training data, see if n_tied_states seems reasonable.
        Total Hours Training: 0.309211965811965
        This is a small amount of data, no comment at this time                                         WARNING

Phase 6: TRANSCRIPT - Checking that all the words in the transcript are in the dictionary
        Words in dictionary: 23
        Words in filler dictionary: 3                                                                   passed

Phase 7: TRANSCRIPT - Checking that all the phones in the transcript are in the phonelist, and all phones in the phonelist appear at least once    passed
```

MODULE: 01 Train LDA transformation (2010-09-30 11:57)

```
Skipped (set $CFG_LDA_MLLT = 'yes' to enable)
```

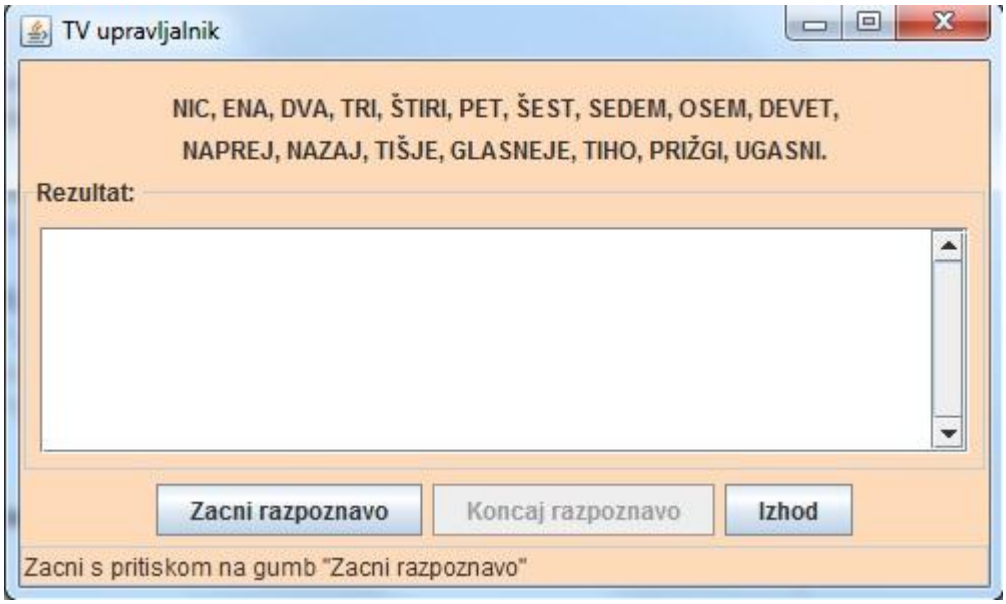
MODULE: 02 Train MLLT transformation (2010-09-30 11:57)

```
Skipped (set $CFG_LDA_MLLT = 'yes' to enable)
```

Slika 9. Primer datoteke, ki se generira med procesom učenja.

7. APLIKACIJA ZA RAZPOZNAVO TELEVIZIJSKIH UKAZOV

Za namen uporabe razpoznavalnika sem s pomočjo testnih primerov, ki so priloženi izvorni kodi sistema Sphinx-4 izdelala aplikacijo, ki zazna izgovorjen televizijski ukaz. Aplikacija nato informacijo o tem, kateri ukaz je bil zaznan pošlje na razvojno ploščico Arduino preko serijski vrat. Del aplikacije, ki poskrbi za pošiljanje podatkov na Arduino, je v priložen na CD-ju. Na Arduinu se nato izvede program (izvorna koda je priložena na CD-ju), ki glede na prejet podatek pošlje IR signal na televizijo.



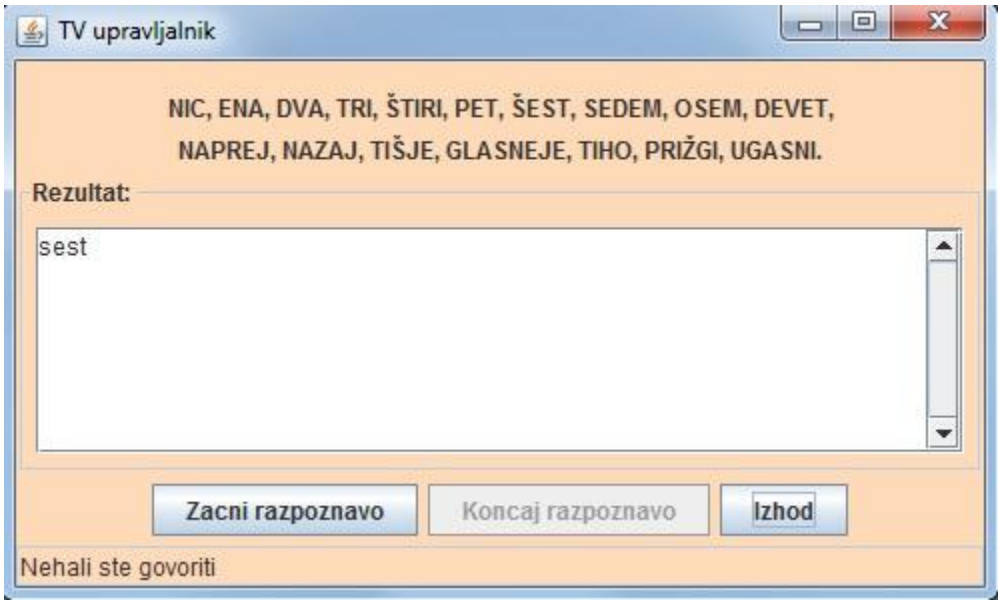
Slika 10. Uporabniški vmesnik aplikacije za razpoznavo ukazov.

Aplikacijo sem razvila s pomočjo prosto dostopnega orodja NetBeans[1]. Najpomembnejši del programa je konfiguracijska datoteka, s katero lahko upravljamo posamezne komponente sistema Sphinx-4. V tej datoteki sem morala določiti poti do akustičnega modela, jezikovnega modela (JSGF[2] format) in še nekaterih drugih datotek, ki so nastale med učenjem. Datoteka je v formatu XML[3] in je v celoti priložena na CD-ju.

[1] NetBeans IDE verzija 6.9.1, dostopno na http://netbeans.org/
[2] Java Speech Grammar Format
[3] angl. Extensible Markup Language

Nekaj besed moram nameniti tudi formatu zgoraj omenjenega jezikovnega modela, to je JSGF format. Slednji se uporablja za predstavitev gramatik[1] pri razpoznavi govora. Gramatike vsebujejo opise izrazov, ki jih uporabnik razpoznavalnika lahko uporabi in razpoznavalnemu sistemu torej služijo kot pokazatelj tega, kaj lahko sploh sliši. Datoteka z jezikovnim modelom je predstavljena v prilogi 11.7.

Aplikacija je zelo enostavna za uporabo, kar je razvidno tudi iz slike 11. V zgornjem delu se zaradi pomoči nahaja seznam vseh možnih ukazov, pod njim je okno, v katero razpoznavalnik vpiše razpoznane ukaze, spodaj pa trije gumbi – "Začni razpoznavo", "Končaj razpoznavo" in "Izhod". S pritiskom na gumb "Začni razpoznavo" dejansko poženemo razpoznavalnik, ki nato razpoznava dokler ne pritisnemo gumba "Končaj razpoznavo" ali zapremo aplikacijo. V spodnjem levem kotu se tekom izvajanja aplikacije pojavljajo sporočila, opozorila in morebitne napake.

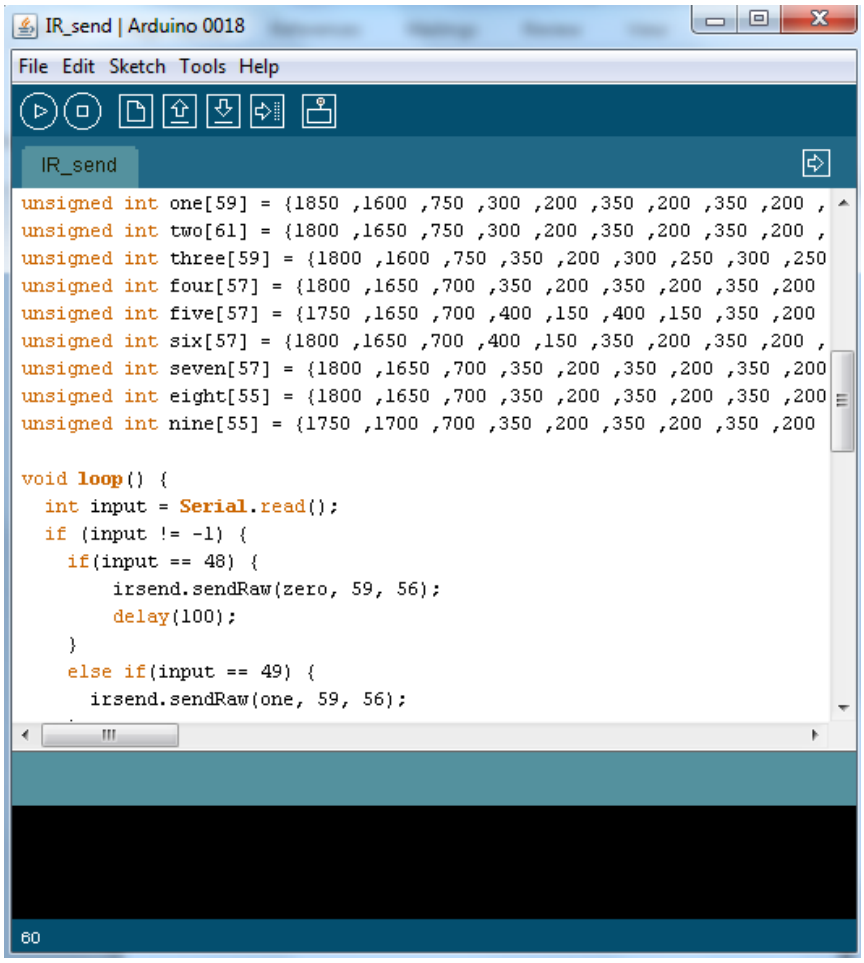


Slika 11. Primer uporabe aplikacije pri upravljanju televizije.

[1] slovnice, angl. grammars

Lastnosti programa:

- razpoznava določenih besed (številke 0-9, naprej, nazaj, tišje, glasneje, tiho, prižgi, ugasni) izgovorjenih preko mikrofona;
- konstantno razpoznavanje (ni potrebno omejiti čas razpoznave);
- možnost razpoznave ene besede ali zaporedja besed (npr. 1 1 – televizija preklopi na program 11).

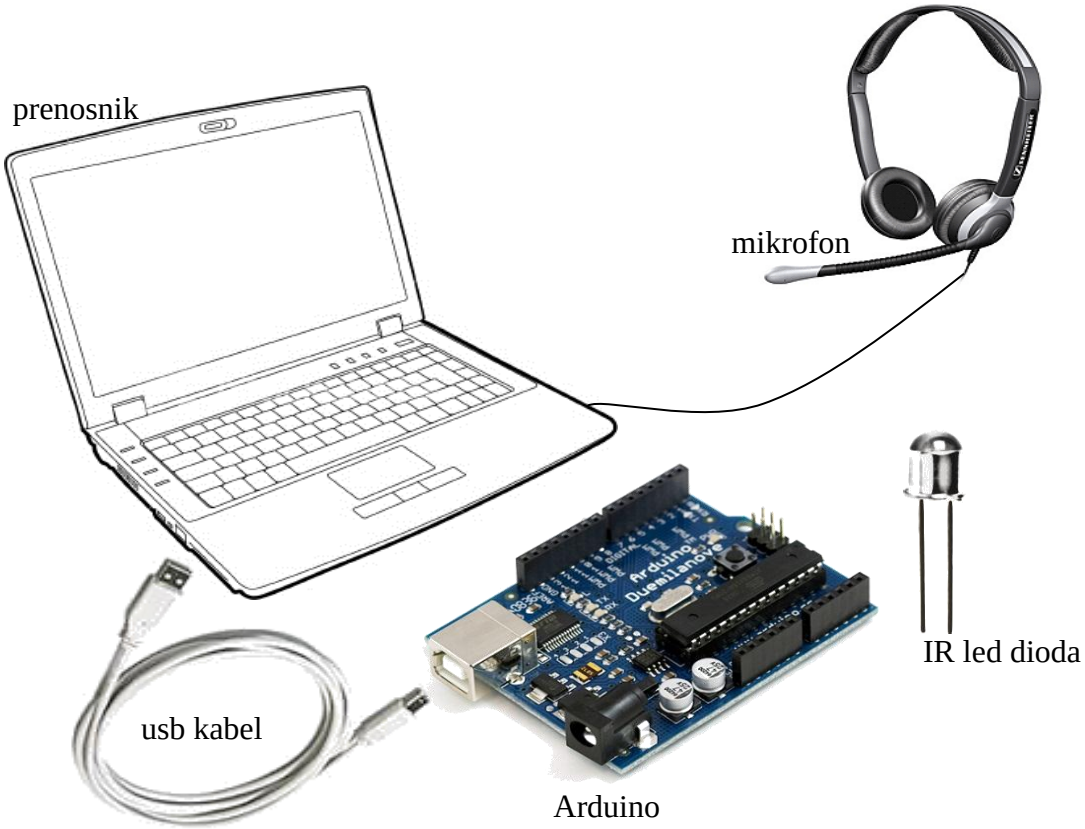


Slika 12. Razvojno okolje Arduino.

8. SHEMA CELOTNEGA SISTEMA

Na sliki 13 si lahko ogledamo shemo sistema za govorno upravljanje televizije.

Mikrofon je priključen na računalnik, kjer teče program[1], ki razpoznava določene besede. Ko določeno besedo razpozna, ta podatek preko serijskih vrat pošlje na Arduino, ki je z računalnikom povezan preko usb kabla. Na Arduinu prav tako teče program, ki glede na prejet podatek pošlje IR signal na televizijo. S tem namenom, je na Arduino priključena IR led dioda.



Slika 13. Shema celotnega sistema.

[1] aplikacija je opisana v prejšnjem poglavju

9. TESTIRANJE IN REZULTATI

Uspešnost akustičnega modela sem preverila z regresijskimi testi[1], ki so v sistem Sphinx-4 že implementirani. V splošnem je regresijsko testiranje testiranje programske opreme, ko po vsakem popravku poženemo že obstoječe teste z namenom odkritja napak, ki smo jih s popravki vnesli v aplikacijo. V tem primeru se regresijski testi uporabljajo za iskanje napak pri razpoznavi in merjenje uspešnosti razpoznave.

Za izvedbo testa potrebujemo naslednje datoteke:

- testne podatke;
- batch datoteko[2];
- akustični model;
- slovar;
- jezikovni model;
- konfiguracijsko datoteko.

Zgoraj omenjena batch datoteka je v bistvu tekstovna datoteka, ki vsebuje seznam datotek, ki morajo biti procesirane skupaj z opisom vsebine vsake izmed datotek. Sphinx-4 primerja razpoznano vsebino datotek s pravilno in tako izračuna točnost oziroma napako. S konfiguracijsko datoteko določimo kateri akustični in jezikovni model bomo uporabili, nastavimo parametre razpoznavalnika in po želji v proces razpoznave vključimo monitorje. Monitorjev je več vrst. Nekateri nam lahko pokažejo rezultate uspešnosti razpoznavanja, drugi pa na primer porabo pomnilnika.

Pri razpoznavanju tekočega govora poznamo tri tipe napak [2]:

- zamenjave, kjer sistem enoto v govornem signalu pravilno razmeji, vendar jo napačno razpozna;
- brisanja, kjer sistem enote, ki je prisotna v govornem signalu ne razmeji in je tako tudi ne razpozna;
- vrinjanja, kjer sistem enoto v govornem signalu napačno razmeji in jo zato tudi napačno razpozna.

Napaka razpoznave se izračuna po naslednji formuli

[1] angl. regression tests
[2] angl. batch file

$$napaka = 100\% * \frac{S+D+I}{N}, \quad (11)$$

pri čemer je S število zamenjav, D število brisanj, I število vrinjanj in N število besed v stavku.

Testiranje sem razdelila na dva dela in ju med sabo primerjala. Najprej sem testirala uspešnost razpoznave, če imamo mikrofon s slušalkami na glavi, nato pa še uspešnost razpoznave, če je mikrofon postavljen na mizi (glasovno upravljanje televizije na daljavo).

	Direktna razpoznava[1]	Razpoznava na daljavo[2]
Natančnost	96,863 %	34,902 %
Število napak	8	166
Zamenjave	8	152
Vrinjanja	0	0
Brisanja	0	14
Število besed	255	255
Pravilno razpoznane	247	89
WER	3,137 %	65,098 %

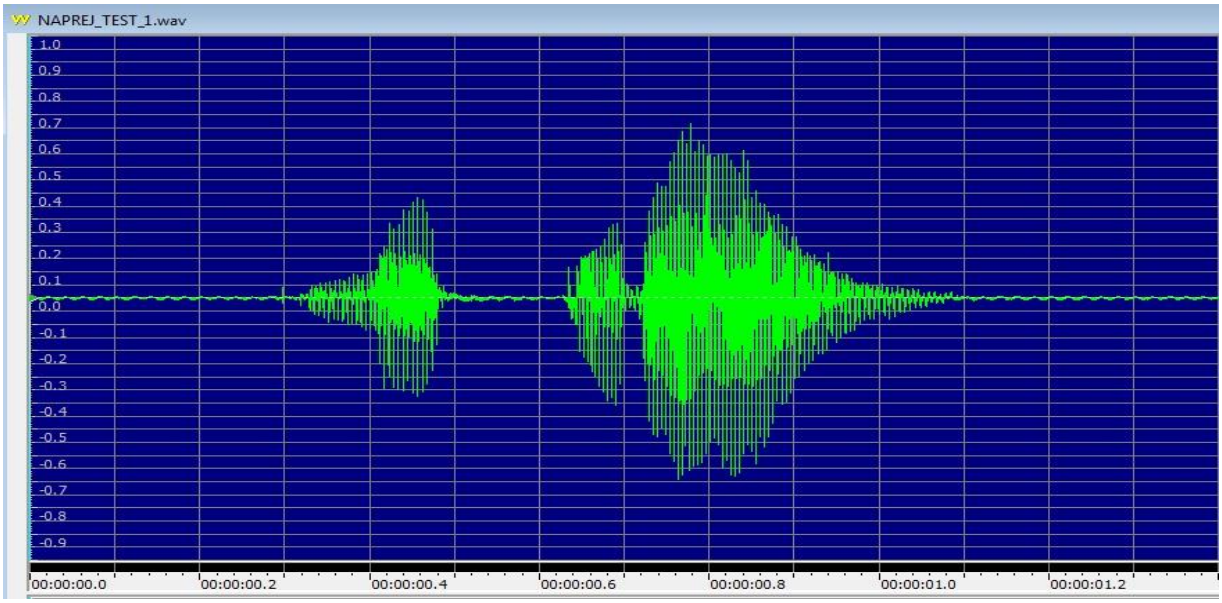
Tabela 1. Rezultati testiranja.

V prvem primeru je bila uspešnost razpoznavalnika 96,863 %, kar je precej dobro. Če imamo mikrofon v neposredni bližini ust lahko njegovo občutljivost[3] zelo zmanjšamo in na tak način zajamemo manj šuma. Občutljivosti pa seveda ne smemo preveč zmanjšati, saj bi na tak način zmanjšali prepoznavnost posamezne besede (manj razločen signal).

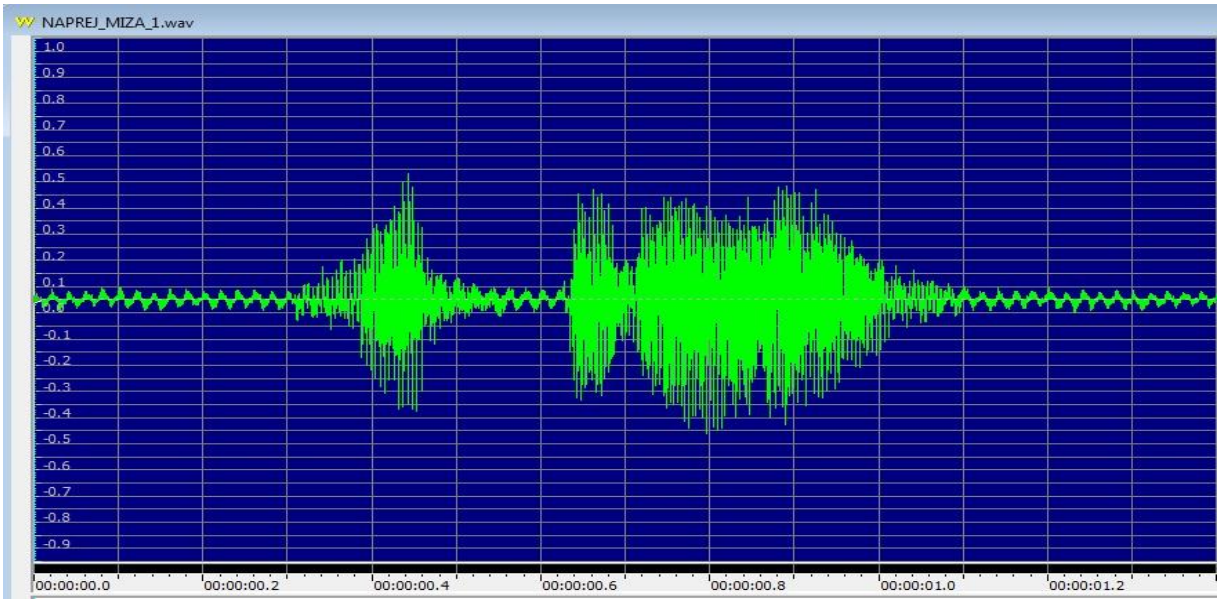
V primeru, ko razpoznavamo na daljavo, je potrebno občutljivost mikrofona povečati, kar pa prinese več šuma. Razpoznava je prav zaradi šuma dosti slabša. Pri testiranju enako velike testne množice je bila uspešnost razpoznave samo 34,902 %.

Kot primer si lahko ogledamo sliko govornega signala za besedo “naprej” v prvem in drugem primeru. Na sliki 15 opazimo precej več šuma, pa tudi značilnosti celotnega signala so bolj zabrisane.

[1] Mikrofon s slušalkami imamo na glavi
[2] Mikrofon je postavljen na mizi
[3] angl. sensitivity



Slika 14. Primer govornega posnetka za besedo “naprej” pri direktni razpoznavi.



Slika 15. Primer govornega posnetka za besedo "naprej" pri razpoznavi na daljavo.

10. SKLEP

Med izdelavo diplomske naloge sem ugotovila, da sta sistem Sphinx-4 prav tako kot orodje SphinxTrain dokaj enostavna za uporabo. Na voljo je dovolj dokumentacije in primerov uporabe, da lahko brez večjih težav izdelamo akustični model, ter ga skupaj z jezikovnim modelom uporabimo pri izdelavi aplikacije za razpoznavo govora.

Do manjših problemov pa vseeno vedno pride. Kar nekaj sem jih namreč imela z določanjem parametrov učenja, katerim je v dokumentaciji posvečeno premalo razlage, čeprav zelo vplivajo na kvaliteto naučenega modela. Dobre akustične modele sem tako dosegla s preizkušanjem različnih parametrov in so se na testih zadovoljivo izkazali.

Sistem za govorno upravljanje televizije, ki sem ga tekom izdelave diplomske naloge zgradila, bi zaradi fleksibilnosti vsekakor nadgradila še z razpoznavo za več govorcev. To je ključna stvar za vsak tak sistem, saj elektronske naprave nikoli ne uporablja samo ena oseba.

Poleg zgoraj omenjenega, bi bil sistem za govorno upravljanje televizije daleč bolj uporaben, če bi mikrofon postavili recimo na mizo v dnevni sobi, vendar bi na tak način pridobili veliko več šuma. To je za proces razpoznave vsekakor zelo slabo, kar sem ugotovila tudi v prejšnjem poglavju pri testiranju.

Slika 16. Primer priročnejšega mikrofona.

Mislim, da so zgornje ugotovitve eden izmed glavnih razlogov, zakaj se govorno upravljanje televizije ne uporablja pogosteje v praksi, saj bi potrebovali priročnejše mikrofone, ki pa so za takšen namen predragi. Primer enega izmed takšnih si lahko ogledamo na sliki 16. Zelo verjetno je, da se bo v prihodnosti, ko bo postopek izdelave takšnih mikrofonov postal cenejši, raba govornega upravljanja elektronike v naših domovih precej bolj razširila.

11. PRILOGE

11.1 Glavni slovar: ukazi.dic

DEVET	D EE V E T
DEVET(2)	D @ V E T
DVA	D V A
ENA	EE N A
GLASNEJE	G L A S N E J EE
GLASNEJE(2)	G L A S N E J @
NAPREJ	N A P R E J
NAZAJ	N A Z A J
NIC	N I CH
OSEM	O S EE M
OSEM(2)	O S @ M
PET	P E T
PRIZGI	P R I ZH G I
PRIZGI(2)	P R @ ZH G I
SEDEM	S E D EE M
SEDEM(2)	S E D @ M
SEST	SH E S T
STIRI	SH T I R I
STIRI(2)	SH T I R @
TIHO	T I H O
TIHO(2)	T I H @
TISJE	T I SH J EE
TRI	T R I
UGASNI	U G A S N I
UGASNI(2)	U G A S N @

11.2 Seznam mašil: ukazi.filler

<s>	SIL
</s>	SIL
<sil>	SIL

11.3 Seznam vseh uporabljenih fonemov: ukazi.phone

@
A
CH
D
E
EE
G
H
I
J
L
M
N
O
P
R
S
SH
T
U
V
Z
ZH
SIL

11.4 Transkripcije: ukazi_train.transcription

<s> NIC </s> (NIC_1)
<s> NIC </s> (NIC_2)
<s> ENA </s> (ENA_1)
<s> DVA </s> (DVA_1)
<s> DVA </s> (DVA_2)
<s> TRI </s> (TRI_1)
<s> STIRI </s> (STIRI_1)
<s> STIRI(2) </s> (STIRI2_1)
…

11.5 Seznam vseh datotek: ukazi_train.fileids

katja/NIC_1
katja/NIC_2
katja/ENA_1
katja/DVA_1
katja/DVA_2
katja/TRI_1
katja/STIRI_1
katja/STIRI2_1
…

11.6 Datoteka z lingvističnimi vprašanji: ukazi.tree_questions

SAMOGLASNIKI	A E EE I O U
ZVOCNIKI	M N R L V J
ZVENECI	D G Z ZH
NEZVENECI	P T CH S SH
NOSNIKI	M N
USTNIKI	V L R J

11.7 Jezikovni model: ukazi.gram

#JSGF V1.0;

grammar ukazi;

public <ukazi> = <programi> | <commands>;
public <programi> = (nic | ena | dva | tri | stiri | pet | sest | sedem | osem | devet)*;
public <commands> = [naprej | nazaj | tisje | glasneje | tiho | prizgi | ugasni];

11.8 Seznam datotek na CD-ju

- izvorna koda razpoznavalnika
- konfiguracijska datoteka razpoznavalnika
- akustični model tv ukazov
- jezikovni model tv ukazov
- datoteke, ki se uporabljajo pri učenju akustičnega modela tv ukazov
- izvorna koda programa, ki se izvaja na Arduinu

VIRI

[1] Basic concepts of Speech, dostopno na:
http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

[2] Z. Kačič, "Komunikacija človek-stroj", Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 1995, pogl. 5

[3] R. Rozman, "Nesimetrične okenske funkcije v sistemih za razpoznavanje govora", doktorska disertacija, Fakulteta za računalništvo in informatiko, Ljubljana, 2005, pogl. 2

[4] Opis razvojnega sistema Arduino, dostopno na:
http://en.wikipedia.org/wiki/Arduino

[5] SphinxTrain documentation, dostopno na:
http://www.speech.cs.cmu.edu/sphinxman/scriptman1.html

[6] Sphinx-4 Whitepaper: A Flexible Opensource Framework For Speech Recognition
http://cmusphinx.sourceforge.net/sphinx4/doc/Sphinx4Whitepaper.pdf