Glasovno upravljanje robotom

Klisović, Nikola

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Nikola Klisović

Zagreb, 2011.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Voditelj rada:

Prof.dr.sc. Bojan Jerbić

Nikola Klisović

Zagreb, 2011.

ZAHVALA

Zahvaljujem se mentoru prof.dr.sc Bojanu Jerbiću na stručnom vodstvu, savjetima i pruženoj pomoći tijekom izrade ovoga rada. Zahvaljujem i asistentima na tehničkoj podršci i korisnim savjetima.

Od srca zahvaljujem roditeljima Diani i Zlatku na neizmjernoj podršci u svim oblicima i vjeri u mene tijekom studija.

Također zahvaljujem ostatku obitelji, djevojci i prijateljima na motivaciji, strpljenju i razumijevanju tijekom studija i izrade ovoga rada.

SAŽETAK

U ovom diplomskom radu obrađuje se tematika prepoznavanja i sinteze govora od strane računala, te sva problematika vezana za tu temu. Izloženi su i objašnjeni matematički modeli i algoritmi koji se trenutačno koriste u tu svrhu, te njihova implementacija u postojećim računalnim sustavima. Također, na konkretnom primjeru upravljanja robotskom rukom, demonstrirana je primjena sinteze i prepoznavanja govora. Radi se o izradi računalnog programa koji koristeći eternet mrežu kao sredstvo komunikacije s robotom, upravlja robotom prethodno obrađenim glasovnim naredbama.

SADRŽAJ

POPIS SLIKA

IZJAVA

Izjavljujem da sam ovaj rad radio samostalno, uz korištenje navedene literature i konzultacija.

Nikola Klisović _____

1 Uvod

Jezik je čovjekovo najvažnije sredstvo komunikacije, a govor primarni medij. Govorna interakcija imanentna je ljudskoj vrsti te se po tome razlikuje od svih ostalih vrsta na zemlji. Svaka interakcija koju čovjek ostvaruje posredno, putem tipkovnice, miša, upravljačkih konzola i drugih sučelja povezana s poteškoćama: gubitkom informacija, greškama i vremenom koje se troši na prijenos i interpretaciju. Stoga se neizbježno nameće potreba implementacije govorne interakcije i u slučaju interakcije čovjeka i stroja. U ovom radu govori se o interakciji između čovjeka i računala kroz sintezu i prepoznavanje aplikacija. Također obuhvaća govorne tehnologije i pretvaranje govora u analogni i digitalni valni oblik koji je razumljiv računalima. Prepoznavanje govora ili koverzija govora u tekst uključuje hvatanje i digitalizaciju zvučnih valova, pretvaranje u osnovne jezične jedinice ili foneme, izgradnju riječi od fonema, i kontekstualne analize riječi kako bi se osigurao ispravan pravopis za riječi koje su zvučno slične. Prepoznavanje govora je sposobnost računala da prepozna opće, prirodne izraze od strane različitih korisnika. Naglasak je na modeliranju govornih jedinica i gramatike na osnovi skrivenog Markovljevog modela. Prepoznavanja govora omogućava unos ulaznih podataka koristeći se glasom.Primjene su nebrojene, od medicine, avio industrije do robotike itd.Iako postoji još mnogo prostora za unapređenja, današnji sustavi daju zadovoljavajuće rezultate, te daju potvrdu da se ova tehnologija razvija u pravom smjeru.

Napredak računalne tehnologije uzrokuje eksplozivni rast u korištenju računala u obradi podataka. U većini slučajeva podaci potječu od čovjeka, te se u konačnici i koriste od strane čovjeka. Tu su stoga potrebni učinkoviti načini prijenosa informacija između ljudi i računala, u oba smjera. Jedan vrlo prikladan način, dolazi u obliku govora, jer govor je komunikacijska metoda koja najčešće koristi između ljudi, te je vrlo prirodna i ne zahtijeva posebnu obuku.Tu su, naravno, mnoge okolnosti u kojima govor nije najbolja metoda za komuniciranje sa računalima. Na primjer, velike količine teksta se mnogo lakše čitaju sa ekrana, a rukovanje pojedinim značajkama unutar samog operativnog sustava je lakše izravnom ručnom manipulacijom. Međutim, za interaktivni dijalog i za unos velike količine teksta ili brojčanih podataka, govor nudi velike prednosti.

2. Komunikacija uporabom govora [1]

Učinkovito komuniciranje s računalom putem govora zahtjeva razumjevanje osnovne činjenice o tome kako ljudi koriste govor za komunikaciju jedni s drugima. Cilj ljudskog govora jest razmjena misli, a riječi i rečenice kojima koristimo obično nisu važne kao takve. Međutim, razvoj intelektualnih aktivnosti i učenje jezika kod čovjeka odvijaju se paralelno tijekom ranog djetinjstva, također i sposobnost da jezik kodira misli u prikladan obrazac za mentalnu preradu dovodi do zaključka da u velikoj mjeri ljudi zapravo formuliraju misli u riječi i rečenice. Uporaba jezika na taj način je samo zgodan način kodiranja misli. Očito govornik stranog jezika će kodirati iste koncepte različitim riječima, također pojedinci unutar jedne jezične skupine mogu imati različite nijanse značenja za iste riječi .

2.1 Odnos jezika u pisanom i govornom obliku

Izum pisanog oblika jezika došao je dugo nakon što su ljudi uspostavili sustave govorne komunikacije, ljudi obično nauče govoriti puno prije nego što nauče čitati i pisati. Međutim, velika ovisnost o pisanom jeziku današnje moderne civilizacije je proizvela sklonost ljudi da razmotre jezik prije svega u pisanom obliku, te da uzimaju govor samo kao izgovorenu formu napisanog teksta, smatrajući je vjerovatno inferiornom zato jer je neprecizna i često puna pogrešaka. U stvari, izgovoreni i pisani oblik jezika su različiti na mnogo načina, a govor ima sposobnost prikazivanja suptilne nijanse značenja koja se teško mogu izraziti u tekstu, gdje je jedina opcija izbor riječi i interpunkcijskih znakova. Govor i tekst imaju svoje vlastite karakteristike kao načini za prenošenja misli i bilo bi pogrešno promatrati jednog kao inferiornu zamjenu za drugog.

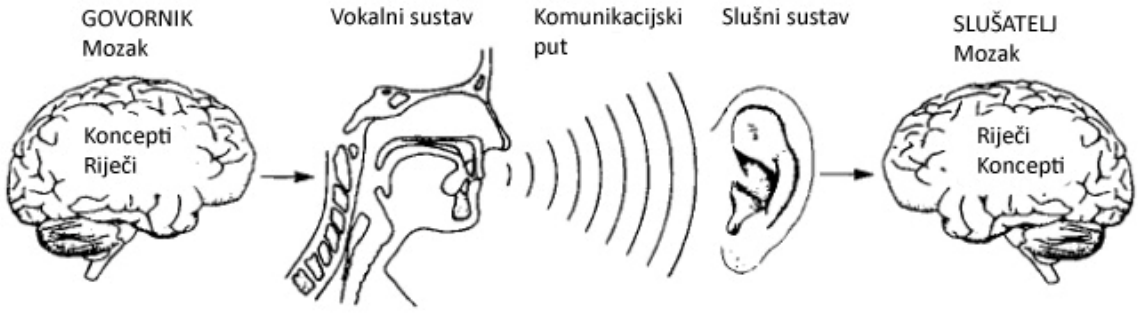
2.2 Fonetika i fonologija

Studija o tome kako se zvučno producira ljudski govor i kako se koristi u jeziku jest utemeljena znanstvena disciplina s dobro razvijenom teoretskom pozadinom. Dijeli se na dvije grane: stvarna generacija i klasifikacija govornih zvukova spada u područje koje se naziva fonetika, a njihove funkcije u jeziku spadaju u područje fonologije. Neke fonetske i fonološke aspekte produciranja i uporabe govora moraju se uzeti u obzir u tehnologiji prepoznavanja govora.

2.3 Zvučni signal

Cilj govornika je prenijeti misli, stavljajući jezik u oblik govora koji uključuje dodatno izuzetno kompliciran proces kodiranja (Slika 1.). Stvarni signal koji se prenosi je zvuk, odnosno oscilacije zvučnog tlaka u jedinici vremena. Iako pojedini govorni zvukovi imaju poprilično karakteristična svojstva, postoji velika varijabilnost u odnosu između zvučnog signala i jezičnih jedinica koje predstavlja. U analizi izražavanja s lingvističke strane jedinice su općenito diskretne tj. riječi, fraze, rečenice. U govoru zvučni signal je kontinuiran, te nije moguće precizno utvrditi mapiranje između vremenskih intervala u govornom signalu i riječi koje predstavljaju. Riječi se normalno udružuju, te u mnogim slučajevima ne postoji jasna zvučna naznaka gdje jedna riječ završava, a sljedeća počinje.

Iako pojedine zvučne komponente u govoru nisu jednoznačno povezane sa značenjem riječi, postoji visok stupanj sustavnog odnosa koji vrijedi većinu vremena.Jer govor je generiran od strane ljudskih vokalnih organa, zvučna svojstva mogu biti povezana s pozicijom govornika. Uz dovoljno treninga, fonetičar može na temelju slušanja govor opisati u smislu sekvenci događaja vezanih uz artikulacijske geste. Analiza slušaoca je uglavnom neovisna o dobi ili spolu govornika. Međunarodna Fonetska Abeceda (IPA) je sustav zapisa kojima fonetičar može opisati vlastitu analizu kao niz diskretnih jedinica. Iako će biti popriličan stupanj jednoglasnosti između fonetičara o transkripciji pojedinih izraza, treba uzeti u obzir da su parametri govorne artikulacije kontinuirano promjenjivi.



Slika 1. Ilustracija procesa razmjena ideja govorom[1]

2.4 Fonemi, foni i alofoni

Mnoge različitosti koje se mogu naći u fonetskoj transkripciji, naprimjer izgovor istih riječi dvoje ljudi na nešto drugačiji način, neće imati utjecaja na značenje. Da bi se napravila razlika u značenju u fonologiji korisno je definirati fonem. Najmanja jedinica u govoru gdje se zamjenom jedne jedinice za drugu može napraviti razlika u značenju. Na primjer, u hrvatskom jeziku se riječi "oni" i "ona“ razlikuju u krajnjim fonemima. Može postojati puno različitih značajki zvučnih uzoraka koji doprinose fonemskoj razlici. Sličan popis simbola koristi se za fonemski zapis kao i za detaljniju fonetsku transkripciju, iako skup glasova je specifičan za jezik koji opisuje. Za svaki jezik samo mali podskup IPA simbola se koristi za predstavljanje fonema, a svaki simbol obično obuhvaća čitav niz fonetskih varijacija. To znači da postoji puno neznatno različitih zvukova koji predstavljaju sve manifestacije istog fonema, a nazivaju se alofoni.

Fonolozi se razlikuju prema načinu na koji analiziraju govor rastavljajući ga u sekvence fonema, posebno za zvukove samoglasnika. Neki predstavljaju duge samoglasnike u engleskom jeziku kao par fonema, dok kratke samoglasnike uzimaju u obzir kao jedan fonem. Drugi¸ uzimaju u obzir duge i kratke samoglasnike kao različite pojedinačne foneme. Takva analiza je korisna za uviđanje razlika u fonetskoj kvaliteti između dugih samoglasnika i najbližih kratkih samoglasnika. Upotrebom ove analize nalaze se oko 44 fonema u engleskom jeziku, a u hrvatskom jeziku 30 fonema. Ponekad se riječ fon(glas) koristi kao opći termin za opisivanje zvučnih ostvarenja fonema kada se varijacije između različitih alofona ne uzimaju u obzir.

Postoji širok spektar zvučnih svojstava alofona koji predstavljaju pojedine foneme. U nekim slučajevima spomenute razlike su posljedica utjecaja susjednih zvukova vezanih za poziciju organa jezika. Ovaj efekt je poznat i kao koartikulacija. U drugim slučajevima razlika može biti značajka jezika razvijena tijekom nekog vremenskog razdoblja, koju novi korisnici prisvajaju učeći jezik u dječjoj dobi. Zamjenjujući jedan samoglasnik za drugi u bilo kojoj riječi nekog jezika ne uzrokuje promjenu identiteta riječi, iako bi rezultat sigurno bio drugačiji izgovor.

2.5 Samoglasnici, suglasnici i slogovi

Samoglasnici i suglasnici se odnose na slova koja čine abecedu. Tijekom izgovora samoglasnika prolaz za protok zraka kroz usta i grlo relativno je sužen i originalni izvor zvuka nalazi se u larinksu, dok kod većine suglasnika postoji znatno suženje prolaza za protok zraka određenu količinu vremena. Kod nekih suglasnika koji su poznati kao „stop“ suglasnici ili plozivi, protok zraka se u potpunosti blokira na nekoliko desetaka milisekundi.Iako govorni zvukovi koji su klasificirani kao samoglasnici obično se razlikuju od suglasnika po ovom kriteriju, postoje neki slučajevi u kojima razlika i nije tako jasna. Vjerojatno je korisnije razlikovati samoglasnike od suglasnika fonološki, na temelju toga kako se oni koriste u stvaranju riječi jezika. S obzirom na njihove funkcije i raspodjele u strukturi jezika, obično je prilično lako odlučiti, za svaki fonem, da li treba biti klasificiran kao samoglasnik ili suglasnik.

U engleskom jeziku postoji mnogo fonema samoglasnika koje nastaju tranzicijom s jedne na drugu razinu kvalitete samoglasnika, iako ih se smatra samostalnim fonemima prema fonološkom sustavu. Takvi vokali su poznati kao diftonzi ili dvoglasi.Zvukovi samoglasnika u riječima "by", "boy" i "bough" su tipični primjeri i ne bi se trebala pridodavati važnost činjenici da je prva riječ određena samo jedni slovom, a druge dvije sa "oy" i "ough".Samoglasnici koji ne uključuju takve kvalitativne tranzicije poznati su kao monotonzi ili jednoglasi.

Postoje neki slučajevi gdje će različite fonološke strukture uzrokovati fonetski slične zvukove koji će biti klasificirani kao samoglasnici u jednom jeziku, a suglasnici u drugom. Na primjer, engleska riječ "pie" i švedska riječ "paj", obje imaju isto značenje, te također zvuče prilično slično. Glavna fonetska razlika je u tome što će na kraju riječi jezik biti bliže nepcu u švedskoj verziji. Međutim, engleska riječ ima dva fonema: početni suglasnik, nakon kojeg slijedi dvoglas za samoglasnik za razliku od švedske riječi koja ima tri fonema: početni suglasnik, nakon čega slijedi jednoglas samoglasnika i konačni suglasnik. Završni suglasnik je fonetski vrlo sličan početnom suglasniku engleske riječi "yet".

Svi govorni jezici imaju slogovnu strukturu, te svi jezici dozvoljavaju da se slogovi sastoje od suglasnika koje slijedi samoglasnik. Ova univerzalna činjenica vjerojatno potječe iz ranih dana razvoja jezika prije mnogo tisuća godina. Prirodna gesta otvaranja usta i generiranje zvuka u larinksu uvijek će proizvesti zvuk sličan samoglasniku. Neki jezici (kao što je japanski) još uvijek imaju vrlo jednostavnu slogovnu strukturu, gdje se većina slogova sastoje od jednog suglasnika iza kojeg slijedi samoglasnik. U jezicima ovog tipa slogovne sekvence vezane su uz izmjenjivanje povećanja i smanjenja glasnoće kako se samoglasnici i suglasnici izmjenjuju. U mnogim drugim jezicima razvio se puno se veći raspon tipova slogova, gdje se slogovi mogu sastojati od samo jednog samoglasnika, ili mogu sadržavati jedan ili više suglasnika na početku i kraju. Slog nikada ne može sadržavati više od jednog samoglasničkog fonema (iako to može biti dvoglas), ali ponekad ne smije sadržavati niti jedan.U drugom slogu izgovora engleskih riječi poput "button", "prism" i "little", kod puno

ljudi konačni suglasnik zvuči nešto produljeno, a da mu ne prethodi samoglasnik. Potrebno artikulacijsko ograničenje za suglasnik na kraju prvog sloga vrijedi i za konačni suglasnik. Drugi engleski govornici mogu proizvesti kratki neutralni samoglasnik između dvaju suglasnika, krajnji rezultat će zvučati prilično slično jer sadrži dva sloga u oba slučaja.Kad je samoglasnik izostavljen završni suglasnik je određen kao slog.

Percepcija slogova u jeziku kao što je engleski ovisi o mnogo različitih faktora. Smanjenje razine signala između dva područja općenito podrazumijeva granicu sloga, ali promjena tona često se koristi za odvajanje slogova. Na primjer, u slučaju kratica " i.e" i "I.E.E" nema promjene kvalitete samoglasnika u drugoj kratica, što znaci da se "EE" sastoji od dva sloga, a ne postoji očigledna promjena razine signala. Međutim, tu je obično uočljiv pad tonaliteta koji će označiti granice između dva slova. Za svaku riječ, odluka o tome da li postoji jedan E ili dva će se osloniti na kombinaciju trajanja, promjene razine glasnosti i tonaliteta, a promjena tonaliteta tendira imati najveći utjecaj. Problemi s određivanjem koliko slogova postoje u riječi nastaju uglavnom u slučaju riječi koje nominalno sadrže sekvence samoglasničkih fonema. U svakodnevnom govoru ponekad se mogu spojiti sekvence samoglasnika, tako da ne postoji prirodna zvučna granica između njih, a ponekad nema očitih promjena fonetske kvalitete tijekom samoglasničkog dijela riječi. Primjer je riječ "tower", u većini izgovora ova riječ ima dva sloga, gdje je samoglasnik u prvom slogu dvoglas, a drugi slog je kratak neutralni samoglasnik; obično nema suglasničkog fonema između njih. Neki ljudi, međutim, oblikuju u slovo o svoje usne tako snažno između dva samoglasnika da /w/ suglasnik obitava u sredini riječi. Drugi ljudi idu u drugu krajnost, te spajaju samoglasnike dvaju slogova u jedan dugi jednoglas, koji se ne razlikuje mnogo od produljene verzije kvalitete samoglasnika na početku uobičajenog dvoglasa. U slučaju kao što je ovaj riječ se može sastojati od samo jednog sloga. Ali postoje neki izgovori koje su u graničnom području gdje je nemoguće odrediti da li postoji jedan slog ili dva.

2.6 Fonemi i pravopis

Vrlo je važno u proučavanju govora ne ostati zbunjen konvencionalnim pravopisom, osobito što se tiče engleskog jezika, gdje je odnos između pravopisa i izgovora jako nepredvidljiv. Iako razlika samoglasnik/suglasnik u engleskom pravopisu nije jako različita od onoga u fonetici i fonologiji, postoje očite anomalije.U riječi "gypsy", naprimjer, obje pojave slova y odražavaju se kao da se radi o samoglasniku, dok je u "yet" y očito suglasnik. Slovo x u "vex" predstavlja slijed od dva suglasnika (fonemski kao/ veks /), ali „gh“ u "cough" predstavlja jedan fonem, / f /.Postoji mnogo slučajeva u engleskom jeziku gdje se slovo e nakon suglasnika ne izgovara, ali njegova prisutnost mijenja fonemski identitet samoglasnika prije suglasnika (npr. "dote" u suprotnosti s "dot").Kombinacije slova samoglasnika često se koriste za predstavljanje jednog samoglasničkog fonema (kao što je "bean") i u nekoliko primjera englesko slovo r nakon samoglasnika a se ne izgovara kao suglasnik, ali uzrokuje da

samoglasnici predstavljaju različite foneme (primjerice, promjena "had" u "hard" i "cod" u "cord" uključuje samo promjenu kvalitete samoglasnika).

2.7 Prozodijske značajke

Nisu samo identiteti fonema nositelji jezičnih informacija u govoru. Tonalitet, intenzitet i vrijeme su također važni u ljudskoj govornoj komunikaciji. U nekim jezicima, od kojih kineski je najočitiji primjer, uzorak tonaliteta unutar riječi je potreban da bi se upotpunilo znanje o fonemima, te utvrdio indentitet riječi.U mandarinskom kineskom postoje četiri različita tona koji se mogu koristiti u svakom slogu, koji predstavljaju četiri različita uzorka tonaliteta. U većini europskih jezika tonalitet,intenzitet i vrijeme (poznati kao prozodijska obilježja govora) ne utječu na identitet riječi, ali daju korisne dodatne informacije o tome što je rečeno.

Prozodijske značajke mogu se koristiti u svrhu određivanja raspoloženja govornika, te da se istaknu određene riječi. Prozodija je glavni čimbenik odgovoran za utvrđivanje koji su slogovi naglašeniji u višesložnim riječima. Najistaknutija prozodijska osobina za isticanje slogova nije, kao što se moglo očekivati, intenzitet, već tonalitet; posebno promjena tonaliteta naglašenih slogova. Dužina trajanja zvuka također povećava naglasak na pojedine slogove, ali postoje mnogi drugi faktori koji utječu na trajanje zvuka, kao što su njihove pozicije u rečenici i identitet susjednih zvukova. Iako naglašeni slogovi imaju tendenciju da su intenzivniji, a zvukovi nižeg tonaliteta na krajevima fraza često su nekoliko decibela tiši, intenzitet je manje značajan u tumačenju govora nego tonalitet i trajanje.

Usredotočujući pozornost na najvažnije riječi, ispravna prozodija je od velike pomoći u tumačenju govornog engleskog jezika. Govor u kojem je prozodija znatno različita od normalne koju koristi izvorni govornik može biti vrlo teško razumljiv.Iako detaljna struktura uzorka tonaliteta može varirati između različitih lokalnih engleskih naglasaka, općenito način na koji se prozodija koristi je sličan. Ritmička struktura je međutim, potpuno drugačija u nekim drugim jezicima, kao što su francuski jezik. U tim slogovno-tempirano određenim jezicima, slogovi se čini dolaze u mnogo ravnomjernijim intervalima u odnosu na naglasak-tempirano određenim jezicima kao što je engleski jezik, gdje postoji tendencija stavljanja dobe kao ritmičke mjerne jedinice na glavni naglašeni slog. Implikacija je da u engleskom jeziku nenaglašeni slogovi koji se nalaze između slogova koje se najviše ističu jesu kraći ako postoje više njih. Iako je ova razlika u vrsti ritma između engleskog i francuskog je jasno među slušateljima, bilo je mnogo kontroverzi oko njihove fizičke korelacije. Pokušaji da se nađe sustavna razlika u izmjerenim uzorcima dužine trajanja sloga između engleskog i francuskog jezika u spontanom razgovoru nisu bila jako uspješna.

2.8 Jezik, naglasak i dijalekt

Različiti jezici se često koriste sasvim različitim fonetskim kontrastima da bi naglasili fonemske razlike. Ta činjenica uzrokuje velike poteškoće učenicima stranog jezika, osobito ako su njihove govorne navike već čvrsto uspostavljene na njihovom materinjem jeziku.

Različiti naglasci na istom jeziku, iako mogu imati dovoljno veliku zvučnu razliku kao što je razlika između prikaza ekvivalentnog fonema kod različitih jezika, obično ne uzrokuje mnogo poteškoća za izvorne govornike. Budući da je temeljna lingvistička struktura gotovo identična, ne postoji mnogo slučajeva u kojima razlike u fonetskoj kvaliteti između naglasaka zapravo uzrokuju konfuziju.

Termin dijalekt često se odnosi na različite tipove izgovora u osnovi istog jezika. Osim što postoje značajne varijacije izgovora, dijalekti su često vezani uz upotrebu alternativnih riječi, a ponekad i sa gramatičkim promjenama, koje se ne susreću izvan područja gdje se dijalekt upotrebljava.

2.9 Nadopunjavanje zvučnog signala

Kada ljudi slušaju govor oni ne čuju nedvosmislen niz zvukova, koji se može dešifrirati jedan po jedan u foneme i grupirati u riječi. U mnogim slučajevima, čak i za jednoznačan niz fonema, postoji dvosmislenost u samom slijedu riječi. Razmatrajući rečenice: " It was a grey day. " i " It was a grade A. ", u tečnom govoru često će biti slučaj da je zvučni obrazac povezan s fonemima, osobito u nenaglašenim pozicijama, u dovoljnoj mjeri se ne razlikuje od zvuka alternativnih fonema. U normalnom razgovoru krivo započete riječi, oklijevanje i blago mucanje posve je uobičajeno. U prisutnosti pozadinske buke govorni signal može biti dodatno iskrivljen. Ipak, ljudi komuniciraju pomoću govora lako i jednostavno.

U normalnom jeziku postoji toliko obilja u jezičnoj poruci tako da je samo mali dio informacija potreban za slušatelja da shvati značenje. Relevantne informacije uključuju ono što slušatelj zna o govorniku, i o čemu će on / ona vjerojatno govoriti. Nakon što se slušatelj priviknuo na glas govornika, prilagoditi će se i naglasku za slučaj rješavanja nekih fonemskih nejasnoća. Ali najviše od svega, za svaku rečenicu ili izraz, slušatelj će izabrati jedno tumačenje koje ima najviše smisla uzimajući u obzir sve raspoložive informacije, zvučne i kontekstualane. U nekim slučajevima konačna odluka će zapravo uključivati odbijanje nekih fonema koji su u skladu s zvučnim signalom u korist drugih koji će činiti manje vjerojatnima na temelju samih zvučnih dokaza. Osim kada je zvučni dokaz u velikoj suprotnosti s normom za izabrane foneme, slušatelj obično neće biti svjestan da zvučni uzorak nije u potpunosti ispravan.

Većina je upoznata s činjenicom da se u prostoriji punoj ljudi može razgovarati sa grupom ljudi u njihovoj neposrednoj blizini, čak iako postoji mnogo ometajućeg govora u okolini.Razlog leži u dodatnim informacijama koje čovjek dobiva. Radi se o mogućnosti čovjeka da sa svojim slušnim sustavom eliminira ostale zvukove tako da mozak odijeli pravce od kuda zvuk dolazi uz pomoć podataka kao što je jačina zvuka i vrijeme dolaska zvuka. Drugi važan čimbenik u je komunikacija licem u lice gdje je govornik u vizualnom polju slušatelja, te slušatelj povezuje zvučni signal s pokretima usana, te s drugim gestama koje se mogu koristiti kao dopuna govoru.

2.10 Kompleksnost procesiranja govora

Jasno je da ljudski perceptivni i kognitivni sustavi moraju biti neizmjerno složeni da bi bili u mogućnosti obaviti zadatak jezične obrade. Vrlo velik broj neurona rade u paraleli. Tako, iako je stvarna brzina procesiranja u bilo kojem dijelu središnjeg živčanog sustava vrlo spora u usporedbi s brzinom modernih elektroničkih sklopova, ukupne perceptivne odluke mogu biti donesene u roku od nekoliko stotina milisekundi. Kako računala moraju prepoznati i interpretirati govor, očito je da oponašati ljudske performanse u obradi normalnog opuštenog razgovora neće biti moguće bez da računalo ima veliko znanje o jeziku i vrlo visoku sposobnost simuliranja ljudske inteligencije. Međutim, ako je zadatak računala pojednostavljen postavljanjem ograničenja na ono što se može reći, tada je već moguće koristiti govor za mnoge vrste interakcije čovjeka i računala. Zbog napretka u tehnologiji uvelike se povećao opseg i složenost poslova za koje se govor može primijeniti, a sposobnosti govorne tehnologije unapređuje se iz dana u dan.

3. Sinteza govora iz tekstualnog zapisa[1]

3.1 Emulacija ljudskih govornih procesa

Kad ljudi govore, mnogi čimbenici koji kontroliraju kako zvuk koji proizlazi iz njih je u vezi s jezičnim sadržajem njihovih iskaza. S jedne strane, postoje određena fiziološka ograničenja njihovog vokalnog aparata. Iako je ljudska fiziologija općenito slična, ona je uzrokovana genetskim razlikama između pojedinaca. Za dani vokalni sustav, govor ovisi o nizu mišićnih aktivnosti koji kontroliraju artikulacijske geste.Te geste se uče od ranog djetinjstva i njihovi detalji su određeni dijelom svojstvima naslijeđenim od središnjeg živčanog sustava, ali i od govorne sredine u kojoj se odrasta. Upravo je govorna sredina u potpunosti odgovorna za utvrđivanje fonetske „riznice“ svakog pojedinca, koja je usko vezana za materinji jezik. Na višoj razini, odnos između ideja koje treba izraziti izborom riječi određene tonalitetom,intezitetom i vremenom, u potpunosti je određen jezikom.

U stjecanju govornih sposobnosti, čovjek ima dva oblika povratne informacije. Prva se odnosi na uspoređivanje proizvedenog zvučnog uzorka s onim koji predstavlja referetni model određenog iskaza. Drugi glavni oblik povratne informacije je odgovor od strane drugih ljudi u svrhu nadodavanja nesavršenosti iskazima nastalim tijekom stjecanja jezika. Jednom kada se može postići proizvesti pravi tip iskaza i nauče potrebne geste, kinestetičke povratne informacije mogu se koristiti za detaljnu kontrolu artikulacijskih pozicija, te se može osigurati nastavak govora čak i ako povratna informacija nije dostupna iz bilo kojeg razloga.

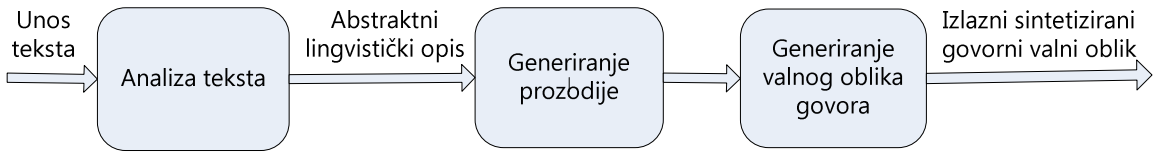
Svi navedeni aspekti stjecanja govora impliciraju da ljudi razvijaju niz pravila na različitim razinama kako bi pretvorili koncepte u govor. Iako se neki dijelovi tih pravila određuju naslijeđenom fiziologijom, a neki učeći iz okoliša, nije lako odvojiti ta dva aspekta. Međutim, jasno je da mora postojati skup pravila za generiranje govora, iako u mnogim slučajevima iskazi će se mijenjati slučajno ili nekim oblikom kreativne varijacije, u granicama onoga štoje prihvatljivo, da se zadrži željeni učinak na slušatelja. Za utjelovljenje kompletnog procesa ljudskog govora ova pravila postaju prekomplicirana posebno u procesu iskazivanja suptilnih crta značenja odabirom riječi i prozodije.

Cilj računalne sinteza govora, bilo iz tekstualnih ili konceptualnih ulaza, jest oponašati osobine tipičnog ljudskog govornog procesa dovoljno dobro da može proizvesti sintetički govor koji je prihvatljiv za ljudskog slušatelja. Sinteza iz teksta trebala bi moći primijeniti pravila koja koristi dobar čitač tijekom interpretacije napisanog teksta i generiranja govora. U svojem najnaprednijem obliku takav sustav bi trebao biti u mogućnosti primijeniti semantičko tumačenje, tako da se način govora prikladan za tekst može prenijeti tamo gdje to nije odmah vidljivo iz kratkog pregleda sekvenci riječi. Sinteza iz koncepta predstavlja poprilično različite izazove, kako računalo već ima neka reprezentativna značenja za prijenos, ali se odgovarajući niz riječi mora generirati za tražene pojmove prije nego što riječi mogu

biti dodatno pretvorene u svoju zvučnu inačicu. Većina radova na sintezi govora je koncentrirana na pretvorbu „teksta u govor (TTS)“.

3.2 Pretvorba teksta u govor

Generiranje sintetičkog govora iz teksta često se karakterizira kao proces koji se sastoji od dva stupnja: analize i sinteze (Slika 2.). Prvi dio ovog procesa uključuje analizu teksta za utvrđivanje temeljne jezične strukture. Ovaj sažeti lingvistički opis će sadržavati nizove fonema i sve ostale informacije, kao što je uzorak naglaska i sintaktičku strukturu, što može utjecati na način izgovaranja teksta. U drugom dijelu procesa TTS pretvorbe stvara se sintetski govor iz jezičnog opisa. Ova faza sinteze može se dalje podijeliti u generiranje prozodije nakon koje slijedi generiranje sintetičkih govornih valnih oblika iz fonemskih i prozodijskih specifikacija.

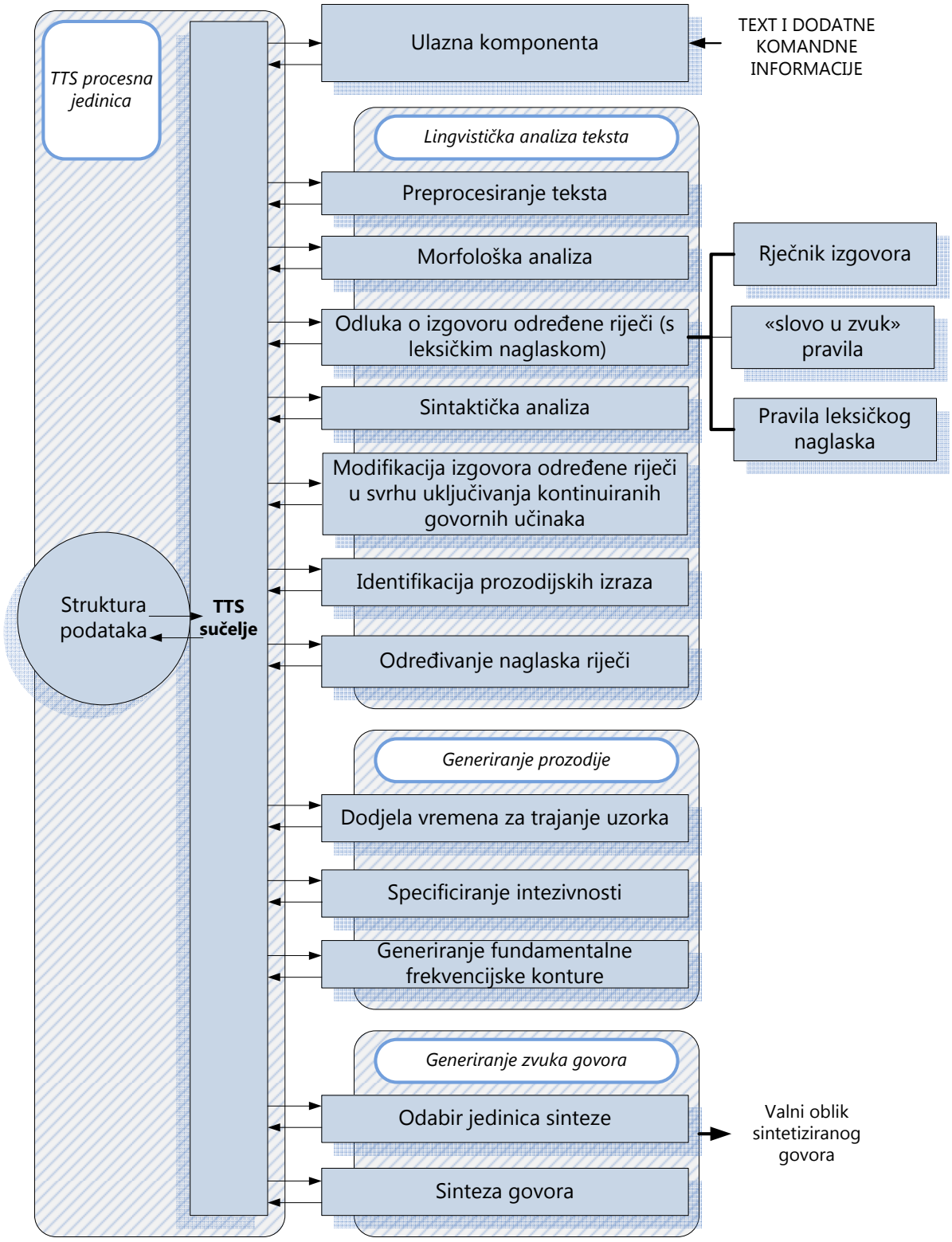


Slika 2. Konverzija teksta u govor prikazan kao analiza-sinteza proces[1]

3.2 TTS sistemska arhitektura

Analiza i sinteza procesa TTS pretvorbe uključuju brojne procesne operacije, a većina modernih TTS sustava ugrađuju te različite operacije unutar modularne arhitekture (Slika 3.). Kada se tekst unosi u sustav, svaki od modula ima neki od ulaz u vezi s tekstom, koji će možda biti generirani od strane drugih modula u sustavu, te generira neki izlaz kojim se onda mogu koristiti daljni moduli, sve dok se ne generira konačni sintetički govorni valni oblik. Međutim, sve informacije unutar sustava prelaze iz jednog modula na drugi putem odvojene obradne jedinice, a moduli ne komuniciraju izravno jedni s drugima. Obradna jedinica kontrolira niz operacija koje se izvode, pohranjuje sve podatke u odgovarajuće podatkovne strukture i bavi se sučeljima potrebnima za pojedine module. Glavna prednost ove vrste arhitekture je lakoća s kojom se pojedini moduli mogu zamjeniti ili novi moduli nadodati.

Jedine promjene koje su nužne vezane su za pristupanje modulima u TTS obradnoj jedinici, a da se na rad pojedinih modula ne utječe. Osim toga, podaci potrebni za sustav (kao što je rječnik izgovora) se odvajaju od obrade nad podacima. Ova struktura ima prednost da je relativno jednostavna za prilagodbu općeg TTS sustava na specifičnu aplikaciju ili poseban naglasak ili čak novi jezik. Postoji sve veći interes za višejezičnom TTS sintezom, pri čemu je cilj da se koriste isti TTS sustavi za sintezu, samo mijenjajući specifične jezične podatke. Opis ovdje će se koncentrirati uglavnom na engleski jezik, ali cjelokupni pristup i mnoge tehnike za pojedine module se također odnose i na druge jezike.



Slika 3. Blok dijagram modularne arhitekture TTS sustava[1]

3.3 Pregled koraka potrebnih za ostvarivanje TTS konverzije

Lingvistička analiza teksta

Tekst se sastoji od alfanumeričkih znakova, praznina i eventualno raznih posebnih znakova. Prvi korak u analizi teksta obično uključuje preobradu unesenog teksta kako bi se konvertirao u niz riječi. Faza prije obrade obično će također otkriti i zabilježiti primjere interpunkcije i druge relevantne informacije za oblikovanje teksta, kao što su prekidi između paragrafa. Sljedeći moduli za analizu teksta zatim rade pretvorbu niza riječi u jezični opis. Glavna funkcija ovih modula je utvrđivanje izgovora pojedinih riječi. U jeziku kao što je engleski odnos između pravopisa riječi i fonemskih prijepisa izuzetno je kompliciran. Nadalje, taj odnos može biti različit za različite riječi sa istom strukturom, kao što je prikazano u izgovoru slova "ough" u riječi "through", "though", "bough","rough" i "cough".

Izgovor riječi normalno je dobiven s pomoću kombinacije rječnika izgovora i pravila kako pojedino slovo treba zvučati. U ranijim TTS sustavima naglasak je bio na izvođenju izgovora po pravilu i koristeći male rječnike iznimaka za uobičajene riječi sa nepravilnim izgovorom (kao što je "one", "two", "said", itd.). Međutim, sada kada su velike računalne memorije dostupne po niskoj cijeni, više je uobičajeno koristiti vrlo velike rječnike (koji mogu sadržati nekoliko desetaka tisuća riječi) kako bi se osiguralo da se poznate riječi izgovaraju pravilno. Pravila se ipak još uvijek moraju baviti s nepoznatim riječima, kao što su naprimjer nove riječi koje se stalno nadodaju jezicima. Zadatak određivanja izgovora riječi je jednostavniji ako je struktura, odnosno morfologija riječi poznata, a većina TTS sustava uključuju neke morfološke analize. Takvom analizom utvrđuje se korijen svake riječi (na primjer, korijen za "gives" je "give"), te se izbjegava potreba uključivanja svih izvedenica u rječniku.Sintaktična analiza teksta je također potrebna kako bi utvrdili izgovor određene riječi. Na primjer, riječ "live" je drugačije izražena ovisno o tome dali se radi o glagolu ("they live here") ili pridjevu ("live wire"). Nakon što dobijemo izgovor za pojedine riječi kao da su izgovorene u izolaciji, potrebno je uključiti neke prilagodbe poput uključivanja fonetskih učinaka, kako bi poboljšalii prirodnost sintetičkog govora.

Osim određivanja izgovora niza riječi, moduli za analizu teksta moraju odrediti i druge relevantne podatke o tome kako tekst treba biti izrečen. Ova informacija, koja uključuje i formuliranje, leksički naglasak (na razini riječi) i obrazac naglašavanja različitih riječi (naglasak na razini rečenice), koristiti će se pri generiranju prozodije za sintetizirani govor. Markeri za leksički naglasak mogu se uključiti za svaku riječ u rječniku, ali pravila će također biti potrebna za dodijeljivanje leksičkog naglaska bilo kojim riječima koje se ne nalaze u rječniku. Rezultat sintaktičke analize također se može koristiti za grupiranje riječi u prozodijske fraze, te da se utvrdi koje bi riječi trebale biti naglašene tako da im se može nadodati odgovarajući obrazac naglašavanja. Dok sintaktička struktura pruža korisne podatke za naglašavanje i formulaciju (a time i prozodiju), u mnogim slučajevima je nemoguće postići doista izražajnu prozodiju bez stvarnog razumijevanja značenja teksta.

Sinteza govora

Informacije dobivene pri analizi teksta se mogu koristiti za generiranje prozodije, uključujući vremenske uzorke, ukupnu razinu intenziteta i temeljnu frekvencijsku (pitch) konturu. Konačni moduli u TTS sustavu obavljaju funkciju generiranje govora odabirom odgovarajućih jedinica za sintezu, a onda slijedi sintetiziranje iz tih jedinica zajedno s prozodijskim informacijama.

3.4 Analiza tekstualnog zapisa

3.4.1 Preobrada tekstualnog zapisa

Tekst ulazi u TTS sustav kao niz znakova u nekom kodiranom formatu, što je u slučaju engleskog jezika ASCII kod. Prva faza u analizi teksta je segmentacija teksta, pri čemu je niz znakova podijeljen na manje dijelove, obično rečenice, te svaka rečenice podijeljena na pojedinačne riječi.Za jezik kao što je engleski odvajanje u riječi je prilično jednostavno pošto se riječi obično odvajaju razmakom. Otkrivanje granica rečenica je manje jasno.Na primjer, točka se obično može se protumačiti kao označavanje kraja rečenice, ali se također koristi za druge funkcije, kao što je označavanje kratica ili kao decimalna točka u slučaju brojeva. Bilo koji neograničen unos teksta će vjerojatno uključivati brojke, kratice, posebne simbole, kao što su %, *, itd., velika slova i razne interpunkcijske i oblikovne informacije (bijeli prostor, znakove tabulatora, itd.). Stoga je uobičajeno da se za preobradu teksta također uključuje proces normalizacije teksta, u kojem je ulazni tekst pretvara u niz izgovorljivih riječi. Normalizirani tekst obično se sastoji se od niza izričito odvojenih riječi, koji se sastoje samo od malih slova i interpunkcije povezane s nekom od riječi. Svaka riječ može biti označena oznakama koje označavaju proširenu kraticu, proširene brojke, velika slova itd. Na taj način, sve informacije mogu se prenijeti, ali u isto vrijeme staviti tekst u format koji je pogodan za daljnju obradu. Većina TTS sustava uključuje veliki broj pravila koji se bave različitim formatima teksta koji mogu nastati u procesu.

Općenito, kratice su prilično jednostavne tako dugo dok su one unaprijed poznate i uključene u tablice za pretvorbu. Međutim, biti će nemoguće predvidjeti sve kratice koje se mogu pojaviti u bilo kojem proizvoljnom ulaznom tekstu, pa je uobičajeno uključiti pravila za otkrivanje kratica. Prisutnost točaka između slova može se uzeti kao dobar pokazatelj da slova trebaju biti izražena odvojeno. Za riječ koja se sastoji od velikih slova je također vrlo vjerojatno da je kratica, barem ako se okolne riječi sastoje od malih slova. Ako je niz slova izgovorljiva riječ, radi se vjerojatno o akronimu (npr. "NATO"), te se stoga treba tretirati kao riječ, ali inače kratica se može izreći kao niz slova.Međutim, neke izgovorljive sekvence isto tako bi trebali biti izgovorene kao pojedinačna slova (npr. "MIT"). Najbolja strategija je

tretirati kratice od četiri ili više slova kao riječi ako su izgovorljive. Za kraće kratice ili ako je kombinacija slova teško izgovorljiva, prikladanije je sricati pojedinačna slova.

Pravila preobrade teksta mogu se nositi adekvatno sa raznim pojavama unutar teksta, ali neograničen tekst će vjerovatno uvijek sadržavati neke značajke oblikovanja teksta koje će biti teško dešifrirati bez sofisticirane analize sintakse, pa čak i značenja. Trenutno, najbolje rješenje je priprema TTS sustava za poznati ograničeni raspon primjene, tako da se preobrada teksta može prilagoditi.

3.4.2 Morfološka analiza

Morfemi su najmanje smislene jedinice jezika. Na primjer, riječ "played" sadrži dva morfema: "play" i morfem vezan za upotrebu prošlog vremena. Morfemi su apstraktne jedinice koje se mogu pojaviti u nekoliko oblika u riječima na koje utječu, tako da na primjer riječ "thought" obuhvaća morfem "think" zajedno s morfemom prošlog vremena. Kada postoji izravna poveznica između apstraktnog morfema i segmenta u tekstualnom obliku, ti segmenti teksta nazivaju se morfi. U mnogim riječima, kao "carrot", cijela riječ se sastoji od jednog morfa. Druga, kao "lighthouse", imaju dva ili više. Morfem se može podijeliti na korijenske morfove i dodatke. Na primjer, "antidisestablishmentarianism" sadrži šest morfema ako se "establish" smatra korijenskim morfom. Velika većina riječi u engleskom jeziku može se kombinirati s prefiksima i / ili sufiksima tako tvoreći nove riječi, ali izgovor izvedenih oblika usko je povezan s izgovorom korijena riječi.

Mogu se osmisliti pravila za pravilnu razgradnju većine riječi (općenito, barem 95% riječi) u svoje konstitutivne morfeme. Morfološka analiza je koristan rani korak u TTS pretvorbi iz nekoliko razloga:

- nije potrebno sve izvedene oblike riječi držati pohranjene u riječniku izgovora

- ako je izgovor individualnih morfema poznat, moguće je pokriti velik broj složenica i visok postotak ukupnog vokabulara, ne mijenjajući veličinu riječnika. Leksikon od N morfema može generirati između 5N i 10N riječi. Cijele riječi trebaju biti samo uključene u riječnik ako ne slijede zakone regularne morfemske kompozicije. Također je koristan u prdviđanju izgovora novih riječi u riječniku

- morfološka analiza daje podatke o atributima kao što je sintaktička kategorija, broj, rod (u slučaju nekih jezika) itd. Ova informacija je korisna za kasniju sintaktičku analizu

3.4.3 Fonetska transkripcija

Uobičajeno je pri utvrđivanju izgovora teksta započeti dodjeljivanjem idealizirane fonemske transkripcije svakoj riječi pojedinačno. Danas, većina TTS sustava koriste velike rječnike.Rječnik uglavnom sadrži samo korijene riječi, te ne i njihove morfološke derivate, osim onih derivata koji se ne mogu ispravno predvidjeti u odnosu na pravila. U slučaju riječi s alternativnim izgovorom, obje mogućnosti mogu biti ponuđene od strane rječnika, a sintaktička analiza koristiti se za odabir. Tipična strategija pri određivanju izgovora riječi je za početak pretražiti rječnik radi provjere dali je u njega uključena cijela riječ. Ako nije, traže se komponente morfema od kojih se riječ sastoji. Pod uvjetom da se individualni morfemi nalaze u rječniku, izgovor izvedene riječi može se odrediti po pravilima za izgovor sastavnih morfema. Kod izgovora izvedenih riječi iz korijenskog oblika, potrebno je uzeti u obzir bilo koja izgovorna modifikacijska pravila povezana s nastavcima. Na primjer, nastavak "ion" mijenja fonemske tumačenje završnog / t / zvuka u riječi kao što je "create". Za bilo koje riječi (ili komponentu morfema) čiji se izgovor ne može utvrditi pomoću rječnika, potrebna su pravila za pretvorbu slova u zvuk. Složenost odnosa između pravopisa riječi i njihove fonemske transkripcije je različita za različite jezike. Međutim, čak i u jeziku, kao što je engleski jezik u kojem je posebno kompliciranо mapiranje između slova i fonema, očito je da moraju postojati neka pravila za odnose pravopisa i sekvenci fonema, jer čovjek može obično razumno nagađati izgovor nepoznatih riječi. Dok pravila za pretvorbu slova u zvuk ne mogu jamčiti da će uvijek dati izgovor koji bi većina ljudi uzelo u obzir kao ispravan, ljudski čitatelj će također često činiti pogreške s nepoznatim riječima.

Predviđanje izgovora vlastitih imena je posebno izazovan zadatak, zato što za imena često vrijede potpuno druga pravila i potječu iz puno različitih jezika. Mnogi TTS sustavi sastoje se od specijalnih pravila za imena, ponekad koristeći shemu temeljenu na analogiji s poznatim imenima.U prirodno izgovorenom kontinuiranom govoru, izgovorena riječ je pod utjecajem identiteta okolnih riječi. TTS sustavi inkorporiraju ove učinke primjenom post-leksičkih pravila za fonetske prilagodbe fonemskih transkripcija pojedinih riječi. Drugi učinci pri izgovoru se odnose na posljedice koartikulacije i željeni rezultat može ovisiti o govornom stilu.

3.4.4 Sintaktička analiza i prozodijski izrazi

Pojedine sintaktičke analize su potrebne kako bi razriješile nejasnoće u izgovoru i utvrdile kako bi iskazi trebali biti strukturirani u izraze. Mogući sintaktički razredi mogu biti uključeni sa svakim unosom u rječnik, a morfološke analize će također pružiti korisne informacije. Međutim, velik broj engleskih riječi može se koristiti i kao imenica i kao glagol, a nekoliko također mogu biti i pridjevi, pa vrlo malo sigurnih informacija o sintaksi može biti dobiveno bez uzimanja u obzir odnos između riječi. Dodjela sintaktičkih razreda, ili oznaka

dijela govora, često se postiže korištenjem statističkog modela jezika, temelji se na vjerojatnostima za pojedine oznake koje se pojavljuju u određenom kontekstu i na vjerojatnostima za oznake koje su povezane s riječima. Modeli vjerojatnosti mogu biti izvedeni iz velike količine ispravno označenog teksta, a tehnika modeliranja se naširoko koristi za modeliranje jezika u automatskom prepoznavanju govora. Nakon što se dio govora odredi za svaku riječ u rečenici, može se utvrditi struktura izraza rečenice. Kako bi se pogodna prozodija mogla generirati, potrebno je odlučiti o tipu rečenice (deklarativno, imperativ ili uptna rečenica), i identificirati fraze i članove. Neki sustavi posjeduju cijelo sintaktičko parsiranje, drugi obavljaju više površne sintaktičke analize, na primjer lociranje imeničnih i glagolskih izraza, te eventualno njihovo grupiranje u rečenice. Također postoje metode koje koriste statističke modele za predviđanje prozodijskih izraza izravno iz podataka o dijelovima govora, naglasku, položaju u rečenici i drugim relevantnim čimbenicima. Opći cilj je proizvesti razumnu analizu za bilo koji tekst, čak i ako tekst sadrži sintaktičke pogreške.

3.4.5 Dodjela leksičkog naglaska i uzorak naglaska riječi

U slučaju višesložnih riječi, obično jedan slog daje osnovni naglasak, a drugi su nenaglašeni, ili nose manje izražena sekundarni naglasak. Ove leksičke oznake naglaska mogu biti definirane za svaki unos u rječnik. Kada se izgovor riječi dobiva kombiniranjem morfa, uzorak naglaska za pojedine morfove može biti promijenjen, pa je neophodno da se primjenjuju pravila za određivanje uzorka naglaska na cijelu riječ. Na primjer, dodavanje sufiksa "ity" u "electric" pomiče primarni naglasak s drugog na treći slog. Za neke riječi, kao što je "permit", naglasak ovisi o sintaktičkoj kategoriji, pa izbor između alternativnih uzoraka naglaska mora biti izveden nakon sintaktičke analize.

Za bilo koju riječ čiji se izgovor definira po „slovo-u-zvuk“ pravilima, dodatna pravila su također potrebna za dodijeljivanje leksičkog naglaska. Za mnoge višesložne riječi u engleskom jeziku položaj primarnih i sekundarnih naglasaka na slogovima može biti određen točno koristeći vrlo komplicirana pravila koja ovise o tome koliko samoglasnika ima u riječi, koliko suglasnika slijedi svaki samoglasnik, duljina samoglasnika, itd. Postoje, međutim, mnoge riječi za koje normalna pravila ne vrijede, kao i činjenica da su neki parovi riječi slične strukture, ali drugačije naglašeni. Primjeri su "Kanada" i "camera", u kontrastu s "Granada" i "banana". Riječi poput ovih će morati biti uključeni u rječniku kako bi se osiguralo ispravno dodjeljivanje leksičkog naglaska.

Jedan od zadnjih zadataka u analizi teksta je dodijeliti naglasak na razini rečenice, pri čemu su različite riječi u rečenici naglašene različitim intezitetom. Dodjela naglasaka ovisi o nizu čimbenika. Funkcijske riječi (kao što su članovi, veznici, prijedlozi i pomoćni glagoli) služe za označavanje odnosa između riječi sadržaja koje nose glavnu informaciju iskaza. Funkcijske riječi nisu obično naglašene, dok riječi sadržaja se naglašavaju različitim intezitetima u ovisnosti o čimbenicima kao što su dijelovi govora i strukture izraza. Kao dodatak sintaksnom

potaknutom pozicioniranju naglaska, naglasak može biti stavljen na važne riječi u rečenici. Na primjer, kada govornik želi naglasiti svoj stav prema nečemu, riječi kao što su "surely", "might" i "not" mogu se koristiti s naglaskom. Naglasak se također može koristiti za definiranje razlika između novih i starih informacija ili skretanja pažnje na kontrast. Uzorak naglaska na različite riječi će obično biti definiran kao promjena u fundamentalnoj frekvenciji, te se često nazivaju “ pitch(tonalitet)“ naglascima.

3.5 Generiranje prozodije

Zvučne poveznice prozodije jesu intenzitet, vrijeme trajanja uzorka i frekvencija. Intenzitet je uglavnom određen glasovnim identitetom, iako također varira u odnosu na naglasak. Iz perspektive prozodije, varijacije u intezitetu općenito su manje utjecajne od varijacija u vremenu trajanja uzorka i frekvencijama.

3.5.1 Vrijeme trajanja uzorka

U veznoj sintezi i u većini sinteza koje funkcioniraju prema skupu pravila, iskazi se generiraju kao sekvence govornih segmenata. Za svaki iskaz, treba biti određeno trajanje za svaki segment, tako da sintetizirani govor oponaša vremensku strukturu tipičnog ljudskog iskaza. Na vremensku strukturu ljudskog govora utječe niz čimbenika koji uzrokuju varijacije trajanja govornih segmenata.Zapažanja o toj varijabilnosti uključuju sljedeće:

- Trajanje različitih govornih zvukova znatno se razlikuju. Neki samoglasnici su istinski kratki, a drugi dugi. Dvoglasi su obično duži nego jednoglasi, a među suglasnicima također postoje sustavne razlike.

- Trajanje se razlikuju ovisno o brzini govora, ali zvukovi pojedinačnih slova koji su uglavnom stabilani, kao što su frikativi i samoglasnici, imaju tendenciju razlikovanja u trajanju više od inherentno prolaznih zvukova, kao što su stop suglasnici.
- Ako je pojedina riječi u rečenici naglašena, njezin najistaknutiji slog je obično produljen.

- Trajanje glasa razlikuje prema položaju u riječi, osobito ako postoji nekoliko slogova.

- Kada se nalazi na kraju fraze, postoji tendencija da slog bude duži nego kad se javlja na drugim mjestima u frazi.

- Samoglasnici prije suglasnika koji se izgovaraju obično su duži nego kad se nalaze ispred onih koji se ne izgovaraju. Na primjer, u engleskom jeziku kod riječi "feed" i "feet", samoglasnik je znatno duži u slučaju riječi "feed". Tu su i druge modifikacije sustavnog trajanja koje ovise o identitetima susjednih glasova.

- Neki dokazi upućuju na to da u jeziku gdje naglasak ovisi o duljini trajanja, nenaglašeni slogovi imaju tendenciju da su kraći, ako postoji nekoliko njih između dva naglašena sloga.

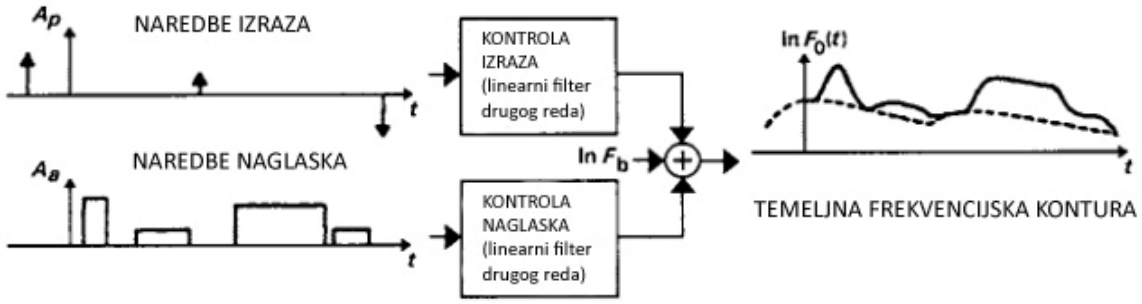
Velik broj sustava je razvijen u svrhu izvođenja trajanja segmenata primjenom niza pravila. Ta pravila su bazirana na fonetskoj transkripciji sa označenim naglašenim slogovima i definiranoj brzini govora. Tada je moguće procijeniti vrijeme za svaki glas imajući definirano „unutarnje“ trajanje pojedinog glasa, te biti ga u stanju modificirati. U kojoj mjeri ga modificirati u cjelini ovisi o uzročnim okolnostima i o identitetu glasa. Skup pravila je razrađen i rafiniran na temelju fonetskog znanja u kombinaciji s statistikom trajanja segmenta govora i rezultatima eksperimenata u kojima se istražuje utjecaj različitih faktora na kvalitetu sinteze. Dok je postignut relativan uspjeh u dobivanju prihvatljivih vremenskih uzoraka, ovaj pristup ne može garantirati optimizaciju pravila za istodobno vrlo širok spektar izraza kojima se opći TTS sustavi moraju baviti.

Automatske metode su postigle određeni napredak u odnosu na starije sustave bazirane na pravilima. Međutim, sadašnji TTS sustavi još uvijek nisu u mogućnosti generirati ritam koji ljudi mogu prirodno integrirati u rečenice koje sadrže rimu ili ostvariti druge sustavne varijacije koje se odnose na značenje. Govoru sintetiziranom iz teksta, također nedostaje uzorak stanki i usporavanja koji se nalaze u govoru dobrog čitača, koje služe kako bi se poboljšalo razumijevanje slušatelja. Više razrađena jezična analiza biti će potrebna u svrhu dobivanja željenog učinka.

3.5.2 Temeljna frekvencijska kontura

Osnovne frekvencije govora, koje određuje tonalitet(pitch) koriste se u svim jezicima u svrhu prijenosa informacije koja se nadovezuje na slijed fonema. U nekim jezicima, kao što je kineski, promjena tonalteta(pitcha) koristi se za razlikovanje različitih značenja slogova koji su fonetski slični. U većini zapadnih jezika promjene tonaliteta(pitch) ne pomažu direktno u identificiranju riječi, ali pružaju dodatne informacije, kao što je definiranje najistaknutije riječi u rečenici, dali se radi o upitnoj rečenici, izjavnoj ili naredbi, raspoloženju govornika itd. Čak i za zapadne jezike, vrsta intonacijskog uzorka koji se koristi za postizanje određenih učinaka znatno varira od jednog jezika do drugog, pa čak i kod različitih naglasaka istog jezika. Očito je da model za generiranje odgovarajućeg intonacijskog uzorka mora biti razvijen tako da odgovara specifičnom jeziku.

Većina rečenica na engleskom pokazuju opću tendenciju k padu tonaliteta postupno prema kraju svake rečenice, ali ta činjenica varira. Dva glavna čimbenika koja određuju varijacije su način na koji je moguće rečenicu podijeliti u izraze i uzorak naglaska rečenice. Najznačajnije varijacije u tonalitetu se javljaju na granicama između izraza i na riječima koje korisnik želi da više istaknuti.U slučaju višesložanih riječi, slog sa primarnim naglaskom je nositelj glavnog tonaliteta.



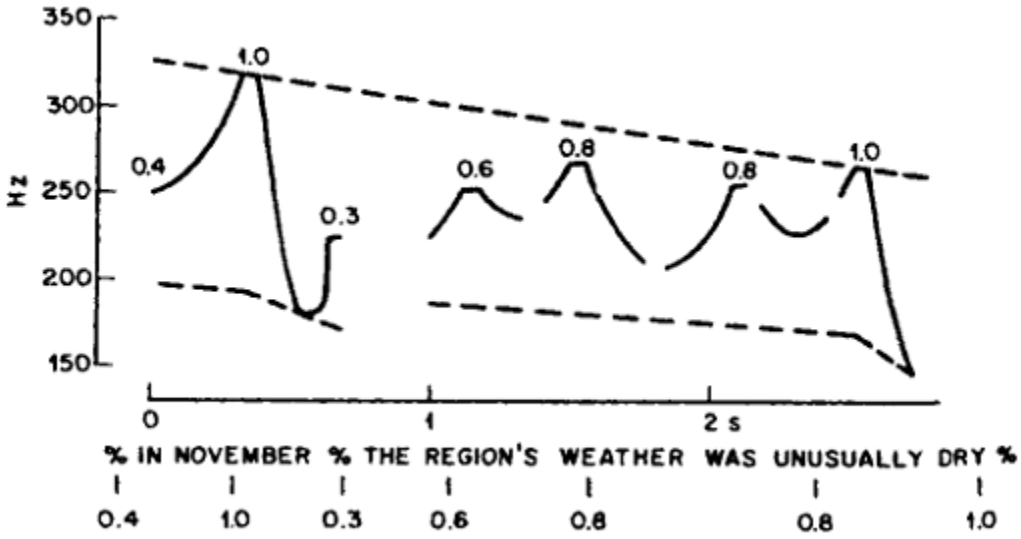
Slika 4. Generiranje temeljne frekvencijske (F_0) konture kao filtrirani niz naredbi izraza i naredbi naglaska, u kombinaciji s osnovnom F_0 vrijednosti (F_b). Generirana F_0 kontura je označena punom linijom u desnom grafu. Točkasto označena linija prikazuje samo konturu naredbi izraza [1]

Različiti modeli su predloženi za generiranje karakteristike osnovne frekvencije (F_0). Postoje razlike između nekih modela koji se odnose na razlike između teorija intonacije na kojima se temelje. Međutim, opća karakteristika svih modela jest da oni djeluju u dva faze. U prvoj fazi se stvara apstraktni opis intonacijske konture (koja će uključivati i neke izraze za tonalitete naglaska), a druga faza služi za pretvorbu abstraktnih opisa u niz F_0 vrijednosti.

Modeli superpozicija hijerarhijski su organizirani i generiraju F_0 konture preklapanjem više različitih tipova komponenata. Primjer ovakvog pristupa generiranja intonacije prikazan je na slici 4. Ovaj model razlikuje naredbe vezane za izraz i naredbe vezane za naglasak. Naredbe su diskretni događaji, prezentirani kao impulsi za naredbu vezanu za izraz, te kao „step“ funkcija za naredbe vezane za naglasak. Kontura F_0 se dobiva filtriranjem svakog slijeda naredbi i kombiniranjem izlaza dvaju filtera, superponiranih na početnoj F_0 vrijednosti.

Nasuprot tome, modeli slijeda tonova generiraju F_0 konturu iz niza diskretnih tonova koji se određuju na lokalnoj razini bez međusobnog utjecaja. Jedan od osobito značajan model je razvio Pierrehumbert (1980). Ton je definiran kao visok ili nizak, te drugačijeg tipa, ovisno o tome da li je povezan s tonalitetom naglaska, granicom izraza ili srednjim položajem između tonaliteta naglaska i granice tona. Za korištenje tog modela za sintezu, Pierrehumbert (1981) definira F_0 raspon koji varira u odnosu na vrijeme i koristi se pravilima za dodjelu visokog ili niskog tona unutar tog raspona za svaki naglašeni slog. Kontura F_0 je generirana primjenom kvadratne funkcije za interpolaciju, kao što je prikazano na slici 5. Pierrehumbertov pristup označavanja intonacije je formirao osnovu za sustav intonacijske transkripcije koji se naziva Tobi (Tones and Break Indices), koji je predložio Silverman. Postoji govorna baza podataka u skladu s Tobi sustavom, koja olakšava razvoj različitih automatskih metoda za obuku modela u svrhu predviđanja abstraktnih Tobi oznaka dobivenih od uzorka naglaska i strukture izraza, te za stvaranje F_0 kontura jednom kada su Tobi simboli dostupni.

Iako su neka istraživanja demonstrirala dobivanje jako prirodnih iskaza uporabom intonacija generiranih od strane modela, za postizanje tih rezultata potrebna je detaljna specifikacija strukture iskaza. TTS sustavi općenito ne potižu takve prirodan govor,a intonacija zvuči manje „zanimljiva“ nego što bi se moglo očekivati od ljudskog govornika. Vjerovatno glavno ograničenje je težina stvaranja dovoljno točne lingvističke analize teksta da bi mogla pružiti odgovarajuće informacije kao ulaz u model za predviđanje intonacije.



Slika 5. Generiranje fundamentalne frekvencijske (F_0) konture interpolacijom između točaka.Isprekidanom linijom naznačen je F_0 raspon, a brojevi predstavljaju lokalne ciljane F_0 vrijednosti, izražene kao udio trenutnog raspona. Točke ciljeva se nalaze na naglašenim slogovima riječi sadržaja i na granicama izraza (označeno s % u tekstu).

4. Uvod u automatsko prepoznavanje govora(ASR) [1]

Velik dio ranih radova vezanih automatsko prepoznavanje govora (ASR) početkom 1950., a pokušava primijeniti pravila bazirana na temelju znanja o zvuku i fonetici ili u mnogim slučajevima na jednostavnim ad hoc mjerenjima svojstava govornog signala za različite vrste zvuka govora. Namjera je bila dekodiranje signala izravno u niz fonemskih jedinica. Ove rane metode postizale su vrlo malo uspjeha. Loši rezultati su bili uglavnom zbog koartikulacije koja uzrokuje da zvučna svojstva pojedinih glasova variraju u širokom pojasu, te bilo koje pravilo temeljeno na identitetu glasa će često biti u krivu, ako se koristi samo lokalna informacija. Nakon pogrešno donesih odluka u ranoj fazi, vrlo teško je ispraviti pogreške kasnije.

Alternativa metodi zasnovanoj na skupu pravila je metoda uspoređivanja uzoraka. Rani pristupi metodi istraživani su u otprilike isto vrijeme kad i rane metode zasnovane na skupu pravila, ali do značajnijih poboljšanja u prepoznavanju govora nije došlo dok nisu razvijene općenitije metode.Iako su se ove metode su naširoko koristile u komercijalnim prepoznavateljima govara tijekom 1970-ih i 1980-ih, oni su sada uglavnom zamijenjene boljim metodama, koja se mogu definirati kao poopćenja ovih jednostavnih metoda uspoređivanje uzoraka. Temeljno razumijevanje principa prvih uspješnih metoda uspoređivanja uzoraka je važan uvod u kasnije metode.

4.1 Opća načela metode uspoređivanja uzoraka

Kada netko izgovori riječ, riječ se može smatrati slijedom fonema (jezične jedinice), a fonemi će biti realizirani u glasove. Zbog neminovne koartikulacije, zvučni uzorci povezani s individualnim glasovima preklapaju se u ovisnosti o vremenu, te ovise o identitetima svojih susjeda. Čak i za riječi izgovorene u izolaciji, zvučni uzorak je na kompliciran način povezan s jezičnim strukturom riječi.

Ipak, ako ista osoba ponavlja istu izoliranu riječ u niz navrata, uzorak će vjerojatno biti uglavnom sličan, jer će vrijediti isti fonetski odnosi. Naravno, vjerojatno će također biti razlika koje proizlaze iz mnogih uzroka. Na primjer, drugi put izgovorena riječ može biti izgovorena brže ili sporije, tonalitet i njegove varijacije mogu biti različiti, precizniji izgovor itd. Očito je da valni oblik odvojenih iskaza iste riječi može biti vrlo različit. Vjerojatno će biti više sličnosti između spektrograma jer oni bolje ilustriraju rezonancije govornog trakta, koje su usko povezane s pozicijom artikulatora. Ali čak i spektrogrami će se detaljno razlikovati, zbog navedenih tipova razlika, a razlike u vremenskim pomacima će biti osobito vidljive.

Dobar pristup ASR-u je pohranjivanje primjera zvučnih uzoraka (tzv. predložaka) tako da sve riječi budu prepoznate od osobe koja koristiti računalo. Bilo koja dolazna riječ tada se može usporediti sa svim pohranjenim riječima, a za onu koja je najsličnija pretpostavlja se da je ispravna. U principu niti jedan od predložaka neće savršeno odgovarati, tako da uspjeh ove metode se mora osloniti na činjenicu da ispravna riječ mora biti što sličnija vlastitom predložku nego bilo kojoj alternativnoj riječi.

Očito je da će u određenom smislu zvučni predložak ispravne riječi bolje odgovarati nego predložak krive riječi, jer je napravljeno više sličnih artikulacijskih pokreta. Iskorištavanje ove sličnosti kritički ovisi o tome kako se predlošci riječi uspoređuju, odnosno o tome kako se različitost između dvije riječi izračunava. Na primjer, bilo bi beskorisno uspoređivati valne oblike jer čak i vrlo slična ponavljanja riječi značajno će se razlikovati u detaljima iz trenutka u trenutak, uglavnom zbog poteškoća ponavljanja intonacije i vremenskog trajanja izgovora riječi. Implicitno mora biti moguće utvrditi početne i krajnje točke riječi koje se uspoređuju.

4.2 Uvod u stohastičko modeliranje

4.2.1 Varijabilnost značajki u metodi uspoređivanja uzoraka

Metode prepoznavanja koriste činjenicu da ponavljani iskaz iste riječi obično ima više sličnih zvučnih uzoraka od iskaza različitih riječi. Međutim, za očekivati je da neki dijelovi uzoraka mogu varirati više ili manje od slučaja do slučaja. U slučaju povezanih riječi, završeci predloška koji predstavljaju svaku riječ vjerojatno će imati vrlo promjenjivu razinu preklapanja, ovisno u kojoj mjeri je ulazni uzorak promjenjen koartikulacijom sa susjednim riječima. Također nema razloga za pretpostaviti da individualne značajke vektora značajki koji predstavlja određeni fonetski događaj su jednake konzistencije. U stvari, moguće je da se dogodi da vrijednost značajke može biti vrlo kritičke naravi na određenoj poziciji u riječi, dok u drugoj riječi pokazati se kao vrlo varijabilna i stoga neznačajna.

Uvijek je poželjna mogućnost ispravka distorzija na vremenskoj skali, jer trajanje govornih zvukova nije obično jako različito između više pojava iste riječi. Međutim, nema razloga pretpostaviti da ispravak distorzija na vremenskoj skali treba biti konstantan za svaki dio svih riječi. Na primjer, poznato je da dugi samoglasnici mogu jako varirati u duljini, dok je većina spektralnih prijelaza povezanih s promjenama u trajanju suglasnika relativno malo.

Može se vidjeti da će se sposobnost prepoznavanja razlike između riječi vjerojatno poboljšati ako se varijabilnosti uzoraka može uzeti u obzir. Ne treba pokušati ispravljati usklađivanje pojedinih riječi ako dijelovi koji se slabo preklapaju su dijelovi za koje se zna da se razlikuju u velikoj mjeri od iskaza do iskaza. Za ispravno korištenje informacija o varijabilnosti treba postojati način za prikupljanje statističkih podataka koji predstavljaju varijabilnost uzoraka riječi, te način korištenja varijabilnosti u procesu uspoređivanja uzoraka.

Primjena statističkih tehnika za rješavanje ovog problema počinju se istraživati tijekom 1970. s prvim objavljenim publikacijama napravljenim od strane Baker-a (1975.) sa Carnegie-Mellon University (CMU) i Jelinek-a (1976.) iz IBM-a. Te moćne tehnike koje uzimaju u obzir varijabilnost postupno su preuzete od jednostavnog uspoređivanja uzoraka. Od tih početaka postoji velik broj istraživanja u svrhu usavršavanja statističkih metoda za prepoznavanje govora, a neke varijante ove metode gotovo su univerzalno usvojene u postojećim sustavima. Ove metode koriste prilično drugačiji način definiranja stupnja poklapanja između riječi i glasovnog podatka. Ova mjera stupnja preklapanja temelji se na pojmu vjerojatnosti. U praksi, većina sadašnjih sustava za raspoznavanje definiraju riječi kao niz jedinica od kojih se riječi sastoje.

4.2.2 Uvod u skriveni Markovljev model

Razmatranje odabira najboljeg izbora riječi tražeći predložak koji daje minimalnu različitost uz optimalno podudaranje je jedan od pristupa. Alternativni pristup bi bio da se za svaku moguću riječ predvidi model, koji može generirati značajke u svrhu predstavljanja riječi. Svaki put kada je model za određenu riječ aktiviran, on će proizvesti niz vektora značajki koji predstavljaju primjer riječi, i ako je model dobar, statistika velikog broja takvih kompleta vektora značajki biti će slična statistici ljudskih iskaza te iste riječi. Najbolja odgovarajuća riječ pri procesu prepoznavanja može se definirati kao ona čiji će model vrlo vjerojatno generirati promatrani niz vektora značajki. Ono što treba izračunati za svaku riječ nije različitost od predloška, nego vjerojatnost da model te riječi može proizvesti promatrani skup vektora značajki. Zapravo se i ne mora napraviti model koji generira vektore značajki, nego koristiti poznata svojstva svakog modela za izračun vjerojatnosti. Treba pretpostaviti da su riječi „izolirane“, tako da se zna početak i kraj svake riječi, a zadatak je identificirati riječi.

Cilj je izračunavanje *posteriorne* vjerojatnosti, *P(w|Y),* za određenu riječ w, nakon što je izgovorena tijekom generiranja skupa rezultata promatranja značajki *Y.* Koristi se model riječi *w* za izračunavanje *P(Y|w)*, te predstavlja vjerojatnost *Y* uvjetovanu riječi w. Za dobivanje *P(w|Y)* mora se također uključiti *a priori* vjerojatnost riječi *w*. Odnos između tih vjerojatnosti je definirana Bayes-ovim pravilom:

$$P(w|Y) = \frac{P(Y|w)P(w)}{P(Y)} \qquad (1)$$

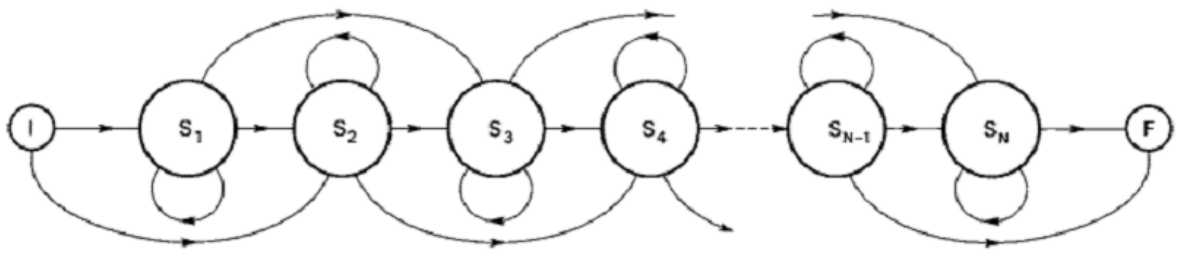
Ova jednadžba govori da je vjerojatnost riječi s obzirom na promatranja jednaka vjerojatnošću promatranja s obzirom na riječ, pomnoženoj s vjerojatnošću riječi (bez obzira na promatranja), te podijeljena s vjerojatnošću promatranja. Vjerojatnost *P(Y)*, za određeni skup promatranih značajki *Y*, ne ovisi o riječi koja se smatra mogućim izborom, te stoga djeluje samo kao faktor skaliranja vjerojatnosti. Dakle, ako je cilj je pronaći riječi w koja maksimizira *P(w/Y)*, a *P(Y)* izraz se može ignorirati, jer ne utječe na izbor riječi. Ako su za određenu primjenu sve dopuštene riječi jednako vjerojatne, onda se *P(w)* može također zanemariti, tako da se mora samo odabrati model riječi koja maksimizira vjerojatnost *P(Y|w)*, generiranjem skupa promatranih značajki *Y*. U praksi za sve, osim za najjednostavnije prepoznavatelje govora, vjerojatnost svake pojedine riječi koja se pojavljuje ovisi o mnogim čimbenicima, a za velike rječnike ovisi o statistici pojavljivanja riječi u jeziku.

Način na koji su riječi definirane kao niz okvirnih predložaka daje polazište za oblik mogućeg modela. Neka model za bilo koju riječ bude sposoban biti u jednom od niza stanja, od kojih svaki može biti povezan s jednim ili više okvira za unos. Općenito model seli iz

jednog stanja u drugo u pravilnim vremenskim intervalima jednakima okvirnim intervalima zvučne analize. Međutim, zna se da riječi mogu varirati na vremenskoj skali. U ovom modelu ova mogućnost može biti predstavljena u nizu stanja, omogućujući modelu da ostane u istom stanju uzastopan niz okvira vremena, ili da zaobiđe sljedeće stanje u nizu. U stvari, ako predložak riječi ima niz vrlo sličnih okvira, kao u slučaju dugih samoglasnika, dopušteno je smanjiti broj stanja u modelu dopuštajući da ostanu u istom stanju u nekoliko uzastopnih okvira.

Matematika povezana s modelom može se učiniti prilogodljivijom donošenjem određenih pretpostavki. Konkretno, pretpostavlja se da je izlaz iz modela stohastički proces (tj. njegovo djelovanje je vođeno potpuno skupom vjerojatnosti), te da vjerojatnosti svih alternativnih akcija u bilo kojem trenutku *t* ovisi samo o stanju koje su poprimile u tom trenutku, a ne o cjelokupnoj vrijednosti *t*. Trenutni izlaz modela ovisi o identitetu sadašnjeg stanja, ali je inače neovisan od niza prethodnih stanja koja je prošao. Tako da je model operacija Markovljev proces prvog reda, a niz stanja Markovljev lanac prvog reda. Iako je struktura modela prikazanog na slici 6. sasvim prikladna za opisivanje riječi koje variraju unutar vremenskog okvira, jednadžbe koje predstavljaju ponašanje modela imaju točno isti oblik u općem slučaju, gdje su dozvoljeni prijelazi između svih mogućih parova stanja.

U svakom vremenskom okviru model je u mogućnosti promijeniti stanje, te će učiniti to nasumično na način određen skupom prijelaznih vjerojatnosti povezanih s stanjem u kojem se trenutačno nalazi. Po definiciji, vrijednost sume vjerojatnosti svih prijelaza stanja u bilo kojem vremenskom okviru mora iznositi 1, ali suma uključuje i vjerojatnost prijelaza ponovnog ulaska u isto stanje. Kada je model aktiviran, niz vektora značajki se odašilje u istom obliku u kojem se nalazi pri promatranju tijekom izgovora riječi. Međutim, u ovoj vrsti modela promatrajući vektore značajki ne može se u potpunosti odrediti kakav je niz stanja. Osim vjerojatnosti prijelaza, svako stanje također je povezano s njim funkcijom gustoće vjerojatnosti (p.d.f) za vektore značajki. Svaki p.d.f. može se koristiti za izračun vjerojatnosti koju bilo koji određeni skup vrijednosti značajki može odašiljati kada je model u pridruženom stanju. Ova vjerojatnost je obično poznata kao emisijska vjerojatnost. Stvarne vrijednosti promatranih značajki su probabilističke funkcije stanja, a sama stanja su skrivena od promatrača. Iz tog razloga ovaj tip modela se zove skriveni Markovljev model (HMM).



Slika 6. Tranzicije stanja za jednostavne modele riječi, od početnog stanja I do konačnog stanja F.[1]

Emisija p.d.f.-a za stanja može biti predstavljena kao diskretna razdioba, s vjerojatnošću navedenom zasebno za svaki mogući vektor značajki. Alternativno, moguće je koristiti parametriziranu kontinuiranu razdiobu, u kojoj su vjerojatnosti vektora značajki definirane parametrima razdiobe. Iako postoje značajne prednosti u modeliranju vjerojatnosti značajki kao kontinuiranih funkcija, radi pojednostavljenja za početak treba uzeti u obzir samo diskretne razdiobe vjerojatnosti.

4.2.3 Izračuni vjerojatnosti u skrivenom Markovljevom modelu

U svrhu objašnjenja HMM izračuna vjerojatnosti, mora se uvesti simbolički zapis za opisivanje različitih vrijednosti koje se izračunavaju. Određeni simboli su konvencionalno povezani s određenim vrijednostima, iako još postoje varijacije u detaljima opisa koji se koristi. Opis u ovom diplomskom radu je u skladu s onim što se čini da se najčešće koristi u objavljenoj literaturi, a također je i konceptualno najjednostavniji.

Uz pretpostavku da su izvedene dobre procjene parametara svih riječi modela. Zadatak prepoznavanja je utvrđivanje najvjerojatnije riječi, s obzirom na promatranja (tj. riječ *w* za koje je *P(w|Y)* maksimalna). Stoga je potrebno izračunati vjerojatnost svakog modela odašiljući promatrani niz značajki (tj. vrijednost *P(Y|w)* za svaku riječ *w*).

Promatrajući jedan model, izlaz predstavlja cijelu riječ koja proizlazi iz modela prolazeći kroz niz stanja, jednakih po dužini broju promatranih vektora značajki *T*, koji predstavlja riječ. Neka je ukupan broj stanja u modelu *N*, te neka *s* označava stanja koja su zauzeta tijekom vremenskog okvira *t* izlaza modela. Također treba definirati početno stanje *I* i konačno stanje *F*, koja nisu povezana niti s jednom odašiljanom vektorskom značajkom i imaju samo ograničen skup mogućih prijelaza. Početno stanje se koristi za određivanje vjerojatnosti prijelaza od početka do svih dopuštenih prvih stanja modela, dok konačno stanje daje vjerojatnosti prijelaza iz svih mogućih zadnjih odašiljačkih stanja sve do kraja riječi. Model mora početi u stanju *I* i završiti u stanju *F*, tako da će ukupno model proći kroz niz *T* +2 stanja za generiranje *T* promatranja.

Korištenje neodašiljčkog inicijalnog i finalnog stanja pruža praktičan način za modeliranje činjenice nekih stanja da imaju veću vjerojatnost od drugih da budu povezana s prvim i zadnjim okvirom riječi.

Najčešće korištena oznaka za vjerojatnost prijelaza iz stanja *i* u stanje *j* je a. Emisijska vjerojatnost stanja *j* koju generira promatrani vektor značajki *y*, obično se označava s *b(y)*.

Potrebno je izračunati vjerojatnost danog modela generirajući promatrani niz vektora značajki, od y_1 do y_T. Poznato je da ovaj niz promatranja treba biti generiran za niz stanja duljine *T* (plus posebna inicijalna i finalna stanja), ali zato što je model skriven, ne znamo identitet stanja. Stoga moramo uzeti u obzir sve moguće nizove stanja duljine *T*. Vjerojatnost

modela koji generira promatranja može se ostvariti pronalaženjem zajedničkih vjerojatnosti promatranja i bilo kojeg niza stanja, te sumiranjem kroz sve moguće nizove stanja ispravne duljine:

$$P(y_1, y_2, \ldots, y_T) = \sum P(y_1, y_2, \ldots, y_T, s_1, s_2, \ldots, s_T)$$

duž svih mogućih sekvenci stanja dužine T

$$= \sum P(y_1, y_2, \ldots, y_T | s_1, s_2, , \ldots, s_T) P(s_1, s_2, \ldots, s_T) \quad (2)$$

duž svih mogućih sekvenci stanja dužine T

gdje se radi jednostavnosti zapisa, u jednadžbi izostavlja ovisnost svih vjerojatnosti o identitetu modela.

Sada je vjerojatnost svakog pojedinog niza stanja dana od strane produkta vjerojatnosti prijelaza:

$$P(s_1, s_2, \ldots, s_T) = a_{ls_1} \left(\prod_{t=1}^{T-1} a_{s_t} a_{s_{t+1}}\right) a_{s_{TF}} \quad (3)$$

gdje je $a_{s_t} a_{s_{t+1}}$ vjerojatnost prijelaza iz stanja zauzetog u vremenskom okviru *t* u stanje u vremenskom okviru *t*+1, a a_{ls_1} i $a_{s_{TF}}$ na sličan način definiraju vjerojatnosti prijelaza iz početnog stanje *I* u konačno stanje *F*. Ako se pretpostavi da se vektori značajki generiraju neovisno za svako stanje, vjerojatnost promatranja s obzirom na određeni niz stanja trajanja *T* je produkt individualne emisijske vjerojatnosti za navedena stanja:

$$P(y_1, y_2, \ldots, y_T | s_1, s_2, , \ldots, s_T) = \prod_{t=1}^{T} b_{s_t}(y_t) \quad (4)$$

Tako da je vjerojatnost modela koji odašilje kompletan niz opažanja :

$$P(y_1, y_2, \ldots, y_T) = \sum a_{ls_1} \left(\prod_{t=1}^{T-1} b_{s_t}(y_t) a_{s_t s_{t+1}}\right) b_{s_t}(y_t) a_{s_t F} \quad (5)$$

duž svih mogućih sekvenci stanja dužine T

Osim ako model ima mali broj stanja, a *T* je također mali, biti će astronomsko veliki broj mogućih nizova stanja, te je potpuno nepraktično napraviti proračune jednadžbe (5) direktno za sve nizove. Međutim, vjerojatnost se može izračunati neizravno pomoću ponavljajućeg odnosa. Koristi se simbol $\alpha_j(t)$ za vjerojatnost modela koji generira prvih *t* promatranih vektora značajki i u stanju je *j* za vremenski okvir *t*. Ponavljanja možemo izračunati u ovisnosti o vrijednosti $\alpha_i(t-1)$ za sva moguća prethodna stanja, *i*.

$$a_j(t) = P(y_1, y_2, \dots., y_1, s_1 = j) \tag{6}$$

$$(\sum_{i=1}^{N} a_j\,(t-1)a_{ij})b_j(y_T)\ \ za\ \ i < t \leq T \tag{7}$$

Vrijednost *a(1)*, za prvi okvir je produkt vjerojatnosti prijelaza a_{Ij} od početnog stanje *I* i emisijske vjerojatnosti $b_J(y_1)$.

$$a_j(1) = a_{Ij}b_j(y_1) \tag{8}$$

Vrijednost $a_J(T)$ za posljednji okvir u promatranom nizu, može se izračunati za bilo koje odašiljačko stanje ponavljanom primjenom jednadžbe (7), počevši od rezultata jednadžbe (8). Ukupna vjerojatnost potpunog skupa promatranja generirana od strane modela mora također uključivati vjerojatnosti prijelaza u konačno stanje *F*. Ta veličina biti će definirana kao $a_F(T)$, odnosno:

$$P(y_1, y_2, \dots., y_T) = a_F(T) = \sum_{i=1}^{N} a_i\,(T)a_{iF} \tag{9}$$

Jednadžba (9) daje vjerojatnost da model generira promatrane podatke, uzimajući u obzir sve moguće nizove stanja. Ova velična predstavlja vjerojatnost promatranja s obzirom na model riječi ($P(Y|w)$ izraz u jednadžbi (1)). Sjedinjavanje vjerojatnosti riječi *P(w)*, daje vjerojatnost koja je umanjena verzija $P(w|Y)$, tj. vjerojatnost izgovorene riječi. Pod uvjetom da je model dobar prikaz željene riječi, ta vjerojatnost pruža koristan podatak koji se može usporediti s vjerojatnostima alternativnih modela riječi kako bi se identificirao najvjerojatniji izbor riječi.

4.2.4 Viterbi algoritam

Vjerojatnost opažanja s obzirom na model, sastoji se od doprinosa iz vrlo velikog broja alternativnih slijedova stanja. Međutim, razdioba vjerojatnosti povezana sa stanjima će biti takva da vjerojatnost promatranih vektora značajki koji su generirani od strane mnogih slijedova stanja biti će mikroskopski mala u usporedbi s vjerojatnošću povezanom s drugim slijedovima stanja. Jedna mogućnost je da se ignorira sve osim jednog najvjerojatnijeg slijeda stanja. Jednadžba (2) može se mijenjati u skladu s tim da daje vjerojatnost, $\hat{P}$ od promatranja za najvjerojatniji slijed stanja:

$$\hat{P}\left(y_{1,}y_{2,}\ldots\ldots,y_T\right) = max(P(y_1,y_2,\ldots.,y_T,s_{1,}s_2,\ldots.,s_T)) \quad (10)$$

duž svih mogućih sekvenci stanja dužine T

Vjerojatnosti povezane s najvjerojatnijim slijedom stanja mogu se izračunati koristeći se Viterbi algoritmom (Viterbi, 1967), koji predstavlja dinamički programabilni algoritam primijenjen na vjerojatnosti. Neka se definira nova vjerojatnost, $\widehat{a_j}(t)$ koja je vjerojatnost postojanja u j-tom stanju, nakon što je emitirano prvih *t* vektora značajki i nakon što se prošlo kroz najvjerojatnije slijedove *t*-1 prethodnih stanja u procesu. Ponovno se pojavljuje povratni odnos, ekvivalent onom prikazanom u jednadžbi (7):

$$\widehat{a_j}(t) = \max_{\text{iznad } i}\left(\hat{a}_i(t-1)a_{ij}\right) b_j(y_T) \quad za \;\; 1 < t \leq T \quad (11)$$

Uvjeti za prvo stanje su jednaki kao i za konačnu vjerojatnost, kao što je bilo prikazano u jednadžbi (8):

$$\widehat{a_j}(1) = a_j(1) = a_{Ij}b_j(y_1) \quad (12)$$

Uzastopne primjene jednadžbe (11), s vremenom će doprinjeti vrijednostima za $\widehat{a_j}(T)$. Definiranje $\widehat{a_F}(T)$ kao vjerojatnost za cijeli skup opažanja definiran od najviše vjerojatnog slijeda stanja, definirana je izrazom:

$$\hat{P}(y_1, y_1, \ldots, y_1) = \hat{a}_F(T) = \max_{\text{iznad } i} (\hat{a}_i(T)\hat{a}_{iF}) \quad (13)$$

Razlika između ukupne vjerojatnosti i vjerojatnosti dane od Viterbi algoritma ovisi o veličini doprinosa najboljeg slijeda ukupnoj vjerojatnosti sumiranoj iznad svih mogućih slijedova stanja. Ako su vektori značajki p.d.f.-a svih stanja bitno različiti jedni od drugih, vjerojatnosti opažanja koje se generiraju od strane najboljeg slijeda ne mogu biti znatno manji od ukupne vjerojatnosti koja uključuje sve moguće slijedove stanja. Razlika između ukupne vjerojatnosti i vjerojatnost za najbolji slijed će biti i veća ako najbolji put uključuje nekoliko uzastopnih okvira djeljenih između skupina od dva ili više stanja koje imaju vrlo slične p.d.f.s za vektore značajki. Tada bi vjerojatnost generiranja promatranih vektora značajki bila gotovo neovisna o tome kako model distribuira svoje vrijeme između stanja unutar te grupe. Ukupna vjerojatnost, što je zbroj svih mogućih raspodjela okvira stanjima, mogla bi biti nekoliko puta vjerojatnost za najbolji slijed. Međutim, dizajn modela koji se koriste u današnjim prepoznavateljima je takav da se slijedovi stanja sa sličnim emisijama p.d.f.s općenito ne pojavljuju. Kao posljedica, bez obzira na teorijske nedostatke ignoriranja svih osim najboljeg puta, u praksi razlike u performansama između dvije metode su obično male. Neke varijante Viterbi algoritma su usvojene za dekodiranje u praktičnim govornim prepoznavateljima, tako što koristeći samo najbolji put zahtijeva manje računanja.

5. Sučelje za programiranje ASR-a i TTS-a(Speech API)

5.1 Uvod

Govorno sučelje za programiranje (SAPI) je API razvijen od tvrtke Microsoft kako bi se omogućilo korištenje prepoznavanja govora i sinteze govora unutar Windows aplikacija.Do danas, nekoliko verzija API-a je izadano, koji su isporučeni ili kao dio govornog SDK, ili kao dio Windows operativnog sustava. Aplikacijama koje koriste SAPI pripadaju Microsoft Office, Microsoft Agent i Microsoft Speech Server. Općenito sve verzije API-a su dizajnirane tako da programer može razviti program za izvođenje prepoznavanja govora i sinteze koristeći standardni set sučelja, dostupan iz različitih programskih jezika. Općenito SAPI je slobodno distribuirajuća komponenta koja može biti isporučena sa bilo kojom Windows aplikacijom u kojoj je potrebno koristiti govorne tehnologije. Mnoge verzije (iako ne sve) od prepoznavanja govora i sinteza motori također su slobodne distribucije.

Verzije SAPI-a od 1 do 4 su slične osim dodatnih mogućnosti koje su bile nadodane sa svakom novijom verzijom.SAPI 5, međutim bio je potpuno novo sučelje, objavljeno 2000 god. Od tada nekoliko verzija ovog API-a je izdano. Zadnja verzija je SAPI 5.4 i sastavni je dio Windows 7 operativnog sustava.[2]

5.2 Općeniti opis rada SAPI-a [2]

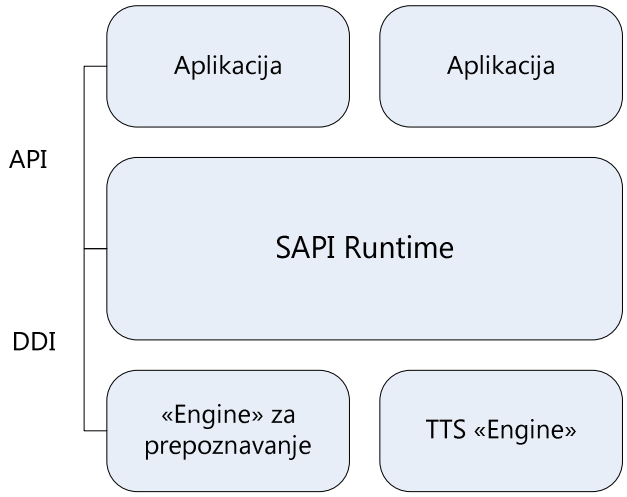
Općenito SAPI se može promatrati kao sučelje ili dio „software-a“ koji je smješten između aplikacije i govornog „engine-a“ (prepoznavanje i sinteza). U SAPI verzijama 1 do 4, aplikacije su mogle izravno mogli komunicirati s „engine-om“. API se sastojao od apstraktnih definicija sučelja koja su bila usklađena s aplikacijom i „engine-om“. Aplikacije su također mogle koristiti pojednostavljene objekte na višoj razini nego direktne pozivne metode „engine-a“.

U SAPI 5, međutim, aplikacije i „engine“ ne komuniciraju izravno. Umjesto toga komuniciraju s „runtime“ komponentom (sapi.dll). Postoji API implementiran od ove komponente kojima se koriste aplikacije, te set sučelja za „engine“.

Obično u SAPI 5 aplikacija upućuje pozive putem API-a (na primjer za učitavanje gramatike, započeti prepoznavanje, ili osigurati sintetiziranje teksta). Sapi.dll „runtime“ komponenta tumači te naredbe i procesuira, te kada je to potrebno poziva „engine“ preko sučelja (na primjer, učitavanje gramatike iz datoteka, ali se tada gramatika proslijeđuje „engine-u“ za prepoznavanje). Engine-i za prepoznavanje i sintezu također stvaraju događaje tijekom obrade (na primjer, da bi indicirali da je iskaz prepoznat ili za označavanje granice riječi u sintetiziranom govoru). Ove radnje idu u obrnutom smjeru, od „engine-a“, preko „runtime“ DLL-a, pa sve do aplikacije.

Osim API definicije i „runtime“ DLL-a i druge komponente se integrirane u sve verzijama SAPI-a, te zajedno čine da kompletan Govorni(Speech) SDK(Software Development Kit). Slijedeći dijelovi se nalaze u većini verzija Govornog(Speech) SDK:

- API definicijske datoteke
- „runtime“ komponente
- Control Panel sučelje – za konfuguriranje sintetizatora i prepoznavanja govora
- TTS „engine“ podrška za više jezika
- ASR „engine“ podrška za više jezika
- Redistribucijske komponente koje omogućuju programerima povezivanje „engine-a“ i „runtime-a“ sa vlastitim aplikacijama
- Dokumentacija



Slika 7. Prikaz načina rada SAPI-a [3]

5.3 SAPI 5 [2]

Govorni(Speech) SDK 5.0, koji uključuje SAPI 5 „runtime“ izdan 2000. To je bio potpuni redizajn od prethodnih verzija, a ni „engine-i“, ni aplikacije koje koriste starije verzije SAPI-a nisu mogli koristiti novu verziju bez značajnih izmjena.

Dizajn novog API uključuje koncept strogo razdvajanje aplikacije i „engine-a“ tako da su svi pozivi preusmjereni kroz „runtime“ sapi.dll. Ova promjena je namijenjena da bi API bio više neovisniji od „engine-a“, sprečavajući ovisnost aplikacija o značajkama određenog “engine-a“. Ove promjene su usmjerene prema mnogo lakšem uključivanju govorne tehnologije u aplikacije pomicanjem upravljačkog i inicijalizacijskog koda u „runtime“.

Novi API je čisti COM API i može se koristiti samo od strane programskih jezika C/C++. Podrška za C# , VB i skriptne jezike nadodana je kasnije.

Glavne značajke API-a:

- **Shared Recognizer**. U desktop aplikacijama za prepoznavanje govora, objekt prepoznavatelja se može koristiti izvođenjem u odvojenom procesu (sapisvr.exe). Sve aplikacije koristeći zajedničko prepoznavanje komuniciraju s ovom jednom instancom. To omogućava dijeljenje resursa, otklanja sukobe za mikrofon i omogućuje globalno sučelje za kontrolu svih govornih aplikacija.
- **In-proc Recognizer**. Za aplikacije koje zahtijevaju eksplicitnu kontrolu procesa prepoznavanja .
- **Grammar objects**. Govorne gramatike koriste se za određivanje riječi koje prepoznavatelj osluškuje.SAPI 5 definira XML oznake za određivanje gramatike, kao i mehanizme za njihovo dinamično stvaranje unutar koda.

- **Voice objects.** Obavlja sinteza govora, za generiranje zvuka iz teksta. Markup Language (slično XML-u) može se koristiti za regulaciju procesa sinteze.
- **Audio interfaces.** „runtime“ obuhvaća objekte za bavljenje ulaznim govorom u mikrofon ili izlaznim govorom na zvučnicima. Također je moguće napisati vlastite audio objekte za generiranje zvuka s nestandardnih lokacija.
- **User lexicon object.** Omogućava dodavanje vlastitih riječi i izgovora od strane korisnika ili aplikacije.
- **Object tokens.** Koncept koji omogućuje registraciju i instaciranje ASR i TTS „engine-a“, audio objekata, leksikona itd.

5.3.1 API za sintetiziranje govora iz teksta [3]

Aplikacije mogu kontrolirati TTS koristeći **ISpVoice** Component Object Model (COM) sučelje. Nakon što je aplikacija stvorila ISpVoice objekt, aplikacija treba samo pozvati **ISpVoice::Speak** za generiranje govora iz tekstualnog podatka. Osim toga, IspVoice sučelje također nudi nekoliko načina za promjenu svojstava glasa i sinteze, kao što je govorna stopa **ISpVoice:: SetRate**, izlazna jačina **ISpVoice:: SetVolume** i promjena sadašnjeg glasa govora **ISpVoice:: SetVoice.**

Također posebna SAPI kontrola također može biti umetnuta uz unos u svrhu promjene svojstava sinteze u realnom vremenu, radi se o svojstvima kao što su glas, tonalitet, naglasak riječi, govorna stopa i jačina. To označavanje sinteze sapi.xsd, koristeći standardni XML format je jednostavan, ali snažan način prilagođavanja TTS govora, neovisno o konkretnom „engine-u“ ili glasu trenutačno u uporabi.

IspVoice:: Speak metoda može raditi sinkronizirano (vratiti tek kada je u potpunosti završila s govorom) ili asinkrono (odmah vratiti i govoriti kao pozadinski proces). Kada se govori asinkrono (SPF_ASYNC), real-time informacije o statusu, kao što je govorno stanje i trenutna lokacija teksta mogu se ispitati koristeći **ISpVoice::GetStatus**. Također, govoreći asinkrono, novi tekst može se izgovorati odmah ometajući trenutačni izlaz (SPF_PURGEBEFORESPEAK), ili automatski dodavanjem novog teksta trenutačnom izlazu.

Osim ISpVoice sučelje, SAPI također pruža mnoga COM sučelja za izradu naprednih TTS aplikacija.

Događaji

SAPI komunicira s aplikacijama slanjem događaja koristeći standardne povratne mehanizme (Window Message, callback proc ili Win32 događaj). Za TTS, događaji se uglavnom koriste za sinkronizaciju s izlaznim govorom.Aplikacije se mogu sinkronizirati s radnjama u realnom vremenu kako se javljaju, kao što su granice riječi, fonem ili visem (animacija usta) granice ili

primjena vlastitih oznaka.Aplikacije mogu inicijalizirati i obraditi ta događanja u realnom vremenu koristeći **ISpNotifySource**, **ISpNotifySink**, **ISpNotifyTranslator**, **ISpEventSink**, **ISpEventSource**, i **ISpNotifyCallback**.

Leksikoni

Aplikacije mogu pružiti prilagođeni izgovor riječi za sintezu govora koristeći metode koje nude **ISpContainerLexicon**, **ISpLexicon** i **ISpPhoneConverter**.

Resursi

Pronalaženje i odabir SAPI govornih podataka kao što su glasovne datoteke i leksikoni izgovora biti će odrađeno uporabom slijedećih COM sučelja: **ISpDataKey, ISpRegDataKey, ISpObjectTokenInit, ISpObjectTokenCategory, ISpObjectToken, IEnumSpObjectTokens, ISpObjectWithToken, ISpResourceManager i ISpTask.**

Audio

Konačno, tu je sučelje za prilagođavanje audio izlaza nekim posebnim odredištima kao što su telefonija i vlastiti hardware (**ISpAudio, ISpMMSysAudio, ISpStream, ISpStreamFormat, ISpStreamFormatConverter**).

5.3.1 API za prepoznavanje govora [3]

Baš kao što je ISpVoice glavnom sučelju za sintezu govora, **ISpRecoContext** je glavno sučelje za prepoznavanje govora. Kao ISpVoice, to je ISpEventSource, što znači da govornoj aplikaciji služi za primanje obavijesti za tražene događaje prepoznavanja govora.

Aplikacija ima izbor odabira dva različita tipa „engine-a“ za prepoznavanja govora (**ISpRecognizer**). Zajedničko prepoznavanje koje se dijeli s drugim aplikacijama za prepoznavanje govora preporuča se za većinu govornih aplikacija. U svrhu kreiranja ISpRecoContext za zajednički ISpRecognizer, aplikacija samo treba pozvati COM CoCreateInstance na komponentu CLSID_SpSharedRecoContext. U tom slučaju, SAPI će postaviti audio tok na ulaz, postavljajući ga na SAPI standardni audio ulazni tok. Za velike poslužiteljske aplikacije koje će izvoditi samo na sustavu, a za koje su ključne performanse, InProc govorni „engine“ je prikladniji. U cilju stvaranja ISpRecoContext za InProc ISpRecognizer, zahtjev mora prvo pozvati CoCreateInstance na komponentu CLSID_SpInprocRecoInstance u svrhu kreiranja vlastitog InProc ISpRecognizer. Tada aplikacija mora uputiti poziv **ISpRecognizer::SetInput** u svrhu postavljanja audio ulaza. Konačno, program može pozvati **ISpRecognizer::CreateRecoContext** u svrhu dobivanja ISpRecoContext.

Sljedeći korak je postavljanje obavijesti za događaje za koje se aplikacija interesira. Kao što je ISpRecognizer također **ISpEventSource**, koji je **ISpNotifySource**, program može pozvati jednu od ISpNotifySource metoda iz ISpRecoContext da naznači gdje da događaji za taj ISpRecoContext budu prijavljeni. Onda to treba pozvati **ISpEventSource:: SetInterest** koji označavaju događaje o kojima treba biti obaviješten. Najvažniji događaj je SPEI_RECOGNITION, koji ukazuje da je ISpRecognizer prepoznao neki govor za ovaj ISpRecoContext.

Na kraju, govorna aplikacija mora stvoriti, pokrenuti i aktivirati ISpRecoGrammar, što ukazuje na koji tip iskaza treba prepoznati, tj. diktat ili zapovijedna gramatika. Prvo, program stvara ISpRecoGrammar koristeći **ISpRecoContext:: CreateGrammar**. Zatim, aplikacija učitava odgovarajuću gramatiku, bilo pozivom **ISpRecoGrammar:: LoadDictation** za diktat ili jedan od **ISpRecoGrammar:: LoadCmdxxx** metode za naredbe. Konačno, kako bi se aktivirala gramatika, te prepoznavanje moglo započeti, program poziva **ISpRecoGrammar:: SetDictationState** za diktat ili **ISpRecoGrammar:: SetRuleState** ili **ISpRecoGrammar:: SetRuleIdState** za naredbe.

6.0 Glasovno upravljanje robotom

U ovom poglavlju je opisana primjena glasovnog prepoznavanja i sinteze na konkretnom primjeru. Radi se o glasovnom upravljanju robotskom rukom Fanuc 200iC.

Izrada cijelog projekta može se podijeliti u nekoliko faza:

- Izrada aplikacije u C# programskom jeziku sa svrhom prepoznavanja glasovnih naredbi, dodjeljivanja oznaka naredbama, te slanja istih oznaka na mrežu koristeći TCP/IP protokol
- Postavljanje i konfiguracija mrežnih postavki i servera u sustavu robotske ruke preko upravljačke jedinice Teach Pendant
- Izrada aplikacije u Karel programskom jeziku sa svrhom pokretanja Teach Pendant programa u ovisnosti od oznaka dobivenih putem mreže

Princip rada sastoji se od poketanja aplikacije RoboVox na udaljenom računalu spojenom na mrežu. Aplikacija RoboVox koristeći Microsoftov SpeechAPI opisan u prethodnom poglavlju prepoznaje unaprijed definirane glasovne naredbe, te konvertira svaku pojedinu prepoznatu naredbu u *integer* oznaku. Nadalje, pošto se aplikacija, a samim tim i računalo, ponaša kao klijent na mreži, šalje *integer* oznake(naredbe) putem mreže na server konfiguriran na sustavu robotske ruke.

Sustav robotske ruke sastoji se od od robotske ruke, kontrolera, Teach Pendant upravljačke jedinice, te se može dopuniti Roboguide programskim paketom čiji je sastavni dio Karel programski jezik. Upravo u Karelu potrebno je napisati program čiji je zadatak ostvarivanje komunikacije između klijenta i servera, te osigurati da dobiveni podaci(naredbe) od strane klijenta trigeriraju željene programe pohranjene u sustavu robotske ruke.

Programski jezik C# [4]

C# je multi-paradigmatski programski jezik koji obuhvaća imperativno, deklarativno, funkcionalno, generičko, objektno-orijentirano(bazirano na klasama) programiranje. Razvijen od strane Microsofta u .NET okružju.

Programska paradigma predstavlja temeljni stil računalnog programiranja. Paradigme razlikuju pojmove i apstrakcije koje se koriste za predstavljanje elemenata programa (kao što su objekti, funkcije, varijable, ograničenja, itd.) i korake sastavljanja proračuna (zadatak, evaluaciju, kontinuitet, protok podataka, itd.).

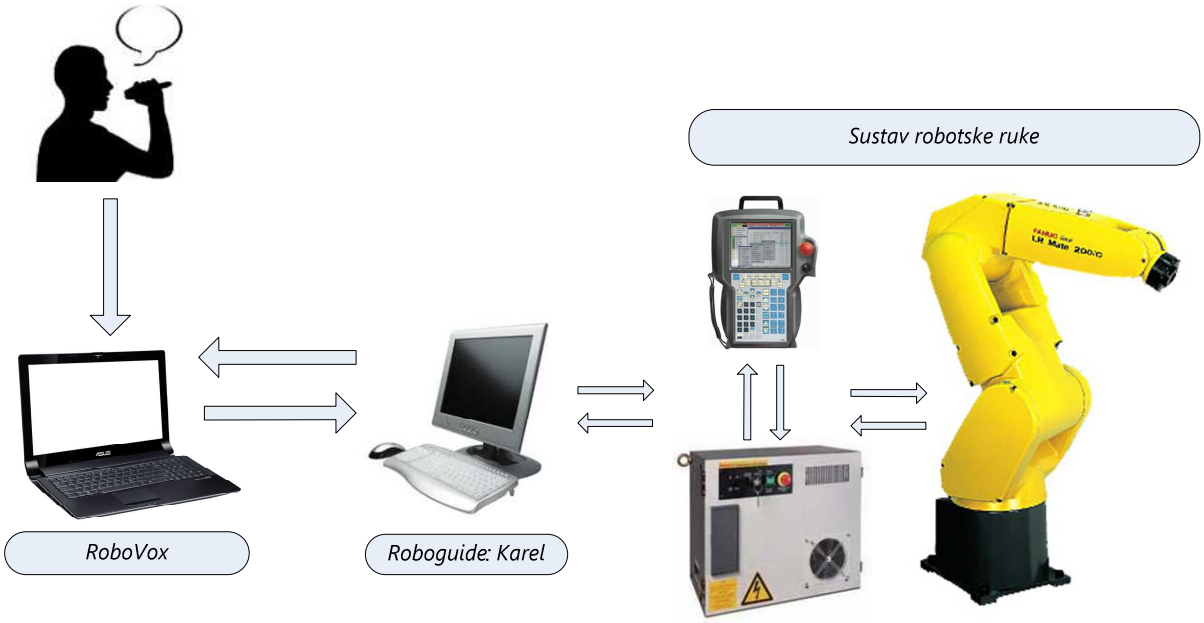
Multi-paradigmatski programski jezik je programski jezik koji podržava više od jedne programske paradigme. Cilj takvog jezika je omogućiti programerima da koriste najbolji alat za obavljanje posla, potvrđujući činjenicu da jedna paradigma ne rješava sve probleme na najlakši i najučinkovitiji način.

Objektno-orijentirano programiranje (OOP) je paradigma programiranja pomoću "objekata" - strukture podataka koja se sastoji od polja podataka i metoda zajedno s njihovim interakcijama - za razvijanje aplikacija i računalnih programa. Objektno-orijentirani pristup potiče programera da stavi podatke na mjesto gdje nisu izravno dostupni ostatku programa. Umjesto toga, podacima se može pristupiti pozivom posebno pisanih 'funkcija', obično nazivanih metodama, koje su ili u paketu s podacima ili naslijeđene iz "klasa objekata" i djeluju kao posrednici za dohvat ili izmjenu podataka. Konstrukcija programiranja koja kombinira podatke sa skupom metoda za pristup i upravljanje podacima naziva se objekt.

Roboguide: Karel programski jezik

Roboguide je programski paket koji omogućuje projektiranje, programiranje sustava robotske ruke u realnom vremenu. Roboguide pruža potrebne alate za razvoj i testiranje potpune robotske aplikacije u 3D okruženju bez troškova povezanih s razvojem prototipnih radnih stanica. Roboguide možete brzo i jednostavno pružiti pouzdane i točne informacije za određenu primjenu. To pomaže u smanjenju količine vremena potrebnog za projektiranje i testiranje rješenja, te ubrzava procjenu održivosti projekta. Roboguide omogućuje unos jedinstvenih CAD modela i stvara virtualne radne stanice koje obuhvaćaju strojeve, uređaja za transport dijelova i prepreke unutar tvornice. Roboguide onda uči putanje robota u svrhu simulacije operacija i performansi robotske ruke uzimajući u obzir fizičke prepreke. Također simulirano u Roboguide 3D okruženju je verifikacija dosega robotske ruke, detekcija sudara, procjene točnih vremen ciklusa i druge operacije vizualnog sustava. Također ima integriran virtualni Teach Pendant koji se ponaša i izgleda kao pravi.

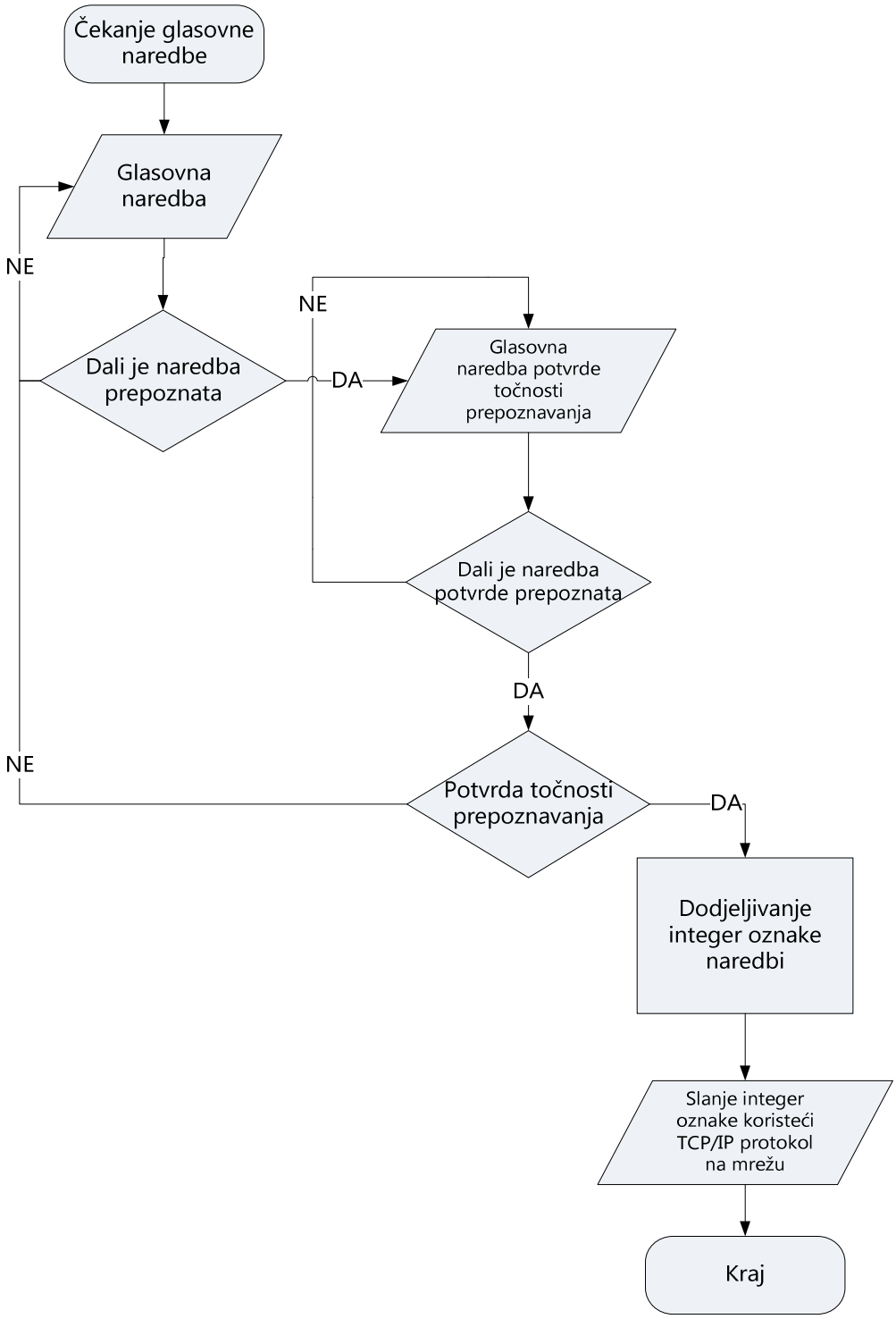
Karel je programski paket za razvoj aplikacija integriran u Fanuc robotski sustav. Po sintaksi najsličniji je Pascal programskom jeziku, ali je specijaliziran za Fanuc robotske sustave.



Slika 8. Princip načina rada glasovnog upravljanja Fanuc 200iC robotskom rukom

6.1 Aplikacija za prepoznavanje glasa(RoboVox)

U ovom poglavlju opisana je izrada i rad aplikacije RoboVox unutar C# programskog jezika.



Slika 9. Dijagram toka aplikacije RoboVox

Programski jezik C# sastavni je dio Microsoft Visual Studio programskog paketa koji služi za razvoj Windows programa i Web aplikacija. Važno je napomenuti da Visual Studio funkcionira u .NET okružju(framework-u) koji sadrži velike knjižnice(library-e) resursa potrebnih za razvijanje aplikacija(subrutine, klase). Podržava nekoliko programskih jezika i programsku interoperabilnost, što znači da svaki jezik može upotrijebiti kod napisan u nekom drugom jeziku.

RoboVox aplikacija isključivo radi u .NET 4.0 okružju na operativnom sustavu Windows 7.

Izrada aplikacije je konceptualno zamišljena tako da se svaka klasa nalazi unutar vlastite datoteke .cs koje međusobno komuniciraju, te čine aplikaciju jednom cjelinom.

Izrada aplikacije započinje izradom klase za prepoznavanje glasovnih naredbi.

Klasa SpeechRecognitionModule

```
using SpeechLib;

namespace RoboSpeechWin
{
        public class SpeechRecognitionModule
        {

                public enum SpeechRecognitionState
                {
                        WaitingForCommand,
                        WaitingForConfirm
                }

                private const string RULE_NAME = "RoboHand Commands";

                private SpeechRecognitionState currentState;

                private SpeechLib.SpSharedRecoContext objRecoContext = null;
                private SpeechLib.ISpeechRecoGrammar grammar = null;
                private SpeechLib.ISpeechGrammarRule roboHandRules = null;


                public event CommandRecognizedEventHandler CommandRecognizedEvent;


                public SpeechRecognitionModule()
                {

// Inicijalizacijsko stanje – čekanje na naredbu predstavlja inicijalizacijsko stanje.
                        currentState = SpeechRecognitionState.WaitingForCommand;

// Dobavljanje instance RecoContext-a. Upotrebljen je djeljeni RecoContext.
                        objRecoContext = new SpeechLib.SpSharedRecoContext();

// Dodjeljivanje eventhandler-a za događaj prepoznavanja naredbe.
                        objRecoContext.Recognition += new
_ISpeechRecoContextEvents_RecognitionEventHandler(RecognizedCommandEventHandler);

// Dodjeljivanje „eventhandler-a“ za događaj neprepoznavanja naredbe.
                objRecoContext.FalseRecognition += new
_ISpeechRecoContextEvents_FalseRecognitionEventHandler(FalseRecognitionEventHandler);

// Stvaranje instance objekta rječnika koji se koristi.
                        grammar = objRecoContext.CreateGrammar();

// Definiranje stanja rječnika.Osiguranje za korištenje rječnika dodjeljenog samo od korisnika.
                        grammar.State = SpeechGrammarState.SGSExclusive;
```

```
//Aktivacija RoboVox naredbi.
                        roboHandRules = grammar.Rules.Add(RULE_NAME, SpeechRuleAttributes.SRATopLevel
| SpeechRuleAttributes.SRADynamic, 1);

                        object PropValue = "";
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Start", " ",
SpeechGrammarWordType.SGLexical, "Robovox Start", 1, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Stop", " ",
SpeechGrammarWordType.SGLexical, "Robovox Stop", 2, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Go Left", " ",
SpeechGrammarWordType.SGLexical, "Robovox Go Left", 3, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Go Right", " ",
SpeechGrammarWordType.SGLexical, "Robovox Go Right", 4, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Go Up", " ",
SpeechGrammarWordType.SGLexical, "Robovox Go Up", 5, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Go Down", " ",
SpeechGrammarWordType.SGLexical, "Robovox Go Down", 6, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Grab", " ",
SpeechGrammarWordType.SGLexical, "Robovox Grab", 7, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Release", " ",
SpeechGrammarWordType.SGLexical, "Robovox Release", 8, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox Yes", " ",
SpeechGrammarWordType.SGLexical, "Robovox Yes", 9, ref PropValue, 1.0F);
                        roboHandRules.InitialState.AddWordTransition(null, "Robovox No", " ",
SpeechGrammarWordType.SGLexical, "Robovox No", 10, ref PropValue, 1.0F);
                        grammar.Rules.Commit();
                        objRecoContext.State = SpeechRecoContextState.SRCS_Enabled;
                        grammar.CmdSetRuleState(RULE_NAME, SpeechRuleState.SGDSActive);

                }

// Definiranje „eventhandler-a“ za slučaj prepoznavanja naredbe. Također definiranje dva različita
stanja unutar tog eventhandler-a: stanja kada je naredba prepoznata i stanja kada je potvrda naredbe
prepoznata.
                private void RecognizedCommandEventHandler(int StreamNumber, object StreamPosition,
SpeechRecognitionType RecognitionType, ISpeechRecoResult Result)
                {
                        var text = Result.PhraseInfo.GetText(0, -1, true);

                        switch (currentState)
                        {
                                case SpeechRecognitionState.WaitingForCommand:
                                        CommandCode.Code cmdCode = CommandCode.CodeFromText(text);
                                        RoboCommand rc = new RoboCommand(cmdCode);
                                        CommandRecognizedEventArgs commandEventArgs = new
CommandRecognizedEventArgs(rc);
                                        OnCommandRecognizedEvent(commandEventArgs);
                                        break;

                                case SpeechRecognitionState.WaitingForConfirm:
                                        CommandConfirm.YesNo cmdYesNo =
CommandConfirm.CodeFromText(text);
                                        RoboConfirm rcc = new RoboConfirm(cmdYesNo);
                                        CommandRecognizedEventArgs confirmEventArgs = new
CommandRecognizedEventArgs(rcc);
                                        OnCommandRecognizedEvent(confirmEventArgs);
                                        break;
                        }
                }
// Automat kojim se prebacuje iz trenutnog stanja u slijedeće u ovisnosti radi li se o naredbi ili
potvrdi naredbe.
                public void ToggleState()
                {
                        switch (currentState)
                        {
                                case SpeechRecognitionState.WaitingForCommand:
                                        currentState = SpeechRecognitionState.WaitingForConfirm;
                                        break;

                                case SpeechRecognitionState.WaitingForConfirm:
                                        currentState = SpeechRecognitionState.WaitingForCommand;
                                        break;
                        }
                }
```

```
// Definiranje „eventhandler-a“ za slučaj neprepoznavanja naredbe. Također definiranje dva različita stanja unutar tog eventhandler-a: stanja kada je naredba neprepoznata i stanja kada je potvrda naredbe neprepoznata.


                private void FalseRecognitionEventHandler(int StreamNumber, object StreamPosition,
ISpeechRecoResult Result)
                {
                        // program (forma) zatrazi od korisnika da ponovi naredbu ako nije prepoznata
                        {
                                RoboCommand rc = new RoboCommand(CommandCode.Code.UNDETERMINED);
                                CommandRecognizedEventArgs eventArgs = new
CommandRecognizedEventArgs(rc);
                                OnCommandRecognizedEvent(eventArgs);

                        }
                        {

                                RoboConfirm rcc = new RoboConfirm(CommandConfirm.YesNo.UNDETERMINED);
                                CommandRecognizedEventArgs eventArgs = new
CommandRecognizedEventArgs(rcc);
                                OnCommandRecognizedEvent(eventArgs);
                        }

                }


                protected void OnCommandRecognizedEvent(CommandRecognizedEventArgs e)
                {
                        CommandRecognizedEvent(this, e);
                }

        }
}
```

Klasa CommandCode

U ovoj klasi uvodi se enumerator koji u ovisnosti o prepoznatoj naredbi dodjeljuje svakoj naredbi neku proizvoljnu integer vrijednost sa svrhom slanja iste na mrežu.

```
namespace RoboSpeechWin
{
        public class CommandCode
        {
                public enum Code
                {
                        UNDETERMINED = -1,

                        START = 10,
                        STOP = 20,
                        LEFT = 100,
                        RIGHT = 110,
                        UP = 120,
                        DOWN = 130,
                        GRAB = 200,
                        RELEASE = 210
                }

                public static Code CodeFromText(string text)
                {
                        Code determinedCode = Code.UNDETERMINED;
                        switch (text.ToLower())
                        {
                                case "robovox start":
                                        determinedCode = Code.START;
                                        break;

                                case "robovox stop":
                                        determinedCode = Code.STOP;
                                        break;
```

```
                        case "robovox go left":
                                determinedCode = Code.LEFT;
                                break;

                        case "robovox go right":
                                determinedCode = Code.RIGHT;
                                break;

                        case "robovox go up":
                                determinedCode = Code.UP;
                                break;

                        case "robovox go down":
                                determinedCode = Code.DOWN;
                                break;

                        case "robovox grab":
                                determinedCode = Code.GRAB;
                                break;

                        case "robovox release":
                                determinedCode = Code.RELEASE;
                                break;
                }

                return determinedCode;
        }
    }
}
```

Klasa CommandConfirm

Klasa indentična CommandCode klasi samo se ne radi o prepoznatim naredbama, nego o prepoznatim potvrdama naredbi, te se dodjeljuje integer oznaka.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RoboSpeechWin
{
    public class CommandConfirm
    {
        public enum YesNo
        {
            UNDETERMINED = -1,
            YES,
            NO,

        }
        public static YesNo CodeFromText(string text)
        {
            YesNo determinedCode = YesNo.UNDETERMINED;
            switch (text.ToLower())
            {
                case "robovox yes":
                    determinedCode = YesNo.YES;
                    break;
                case "robovox no":
                    determinedCode = YesNo.NO;
                    break;

            }

            return determinedCode;
        }

    }
}
```

Klasa TcpIpClient

Zbog toga što komunikacija između udaljenog računala preko kojeg se aplikacijom RoboVox glasovno upravlja robotskom rukom funkcionira po načelu klijent->server, u ovoj klasi nalazi se kod koji aplikaciji RoboVox dodjeljuje ulogu klijenta i šalje podatke na server robotskog sustava koristeći TCP/IP protokol.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace RoboSpeechWin
{
        public class RoboTcpIpClient
        {
                private IPEndPoint ipep;
                private TcpClient tcpClient;

// Definiranje instance objekta TCP/IP klijenta.
                public RoboTcpIpClient(IPAddress serverIpAddress, int port)
                {
                        ipep = new IPEndPoint(serverIpAddress, port);
                        tcpClient = new TcpClient();
                        tcpClient.Connect(ipep);
                }

// Definiranje slanja i primanja paketa podataka.
                public string SendCommand(CommandCode.Code commandCode)
                {
                        var netStream = tcpClient.GetStream();
                        netStream.ReadTimeout = 200;

                        StreamWriter sw = new StreamWriter(netStream);
                        sw.WriteLine((int)commandCode);
                        sw.Flush();

                        //StreamReader sr = new StreamReader(netStream);
                        //string response = sr.ReadLine();
                        return null;// response;
                }


                public void CloseTcpConnection()
                {
                        tcpClient.Close();
                }
        }
}
```

Klasa RoboCommand

```
namespace RoboSpeechWin
{
        public class RoboCommand
        {
                private CommandCode.Code selectedCommand;

                public RoboCommand(CommandCode.Code cmdCode)
                {
                        selectedCommand = cmdCode;
                }

                public CommandCode.Code CommandCode { get { return selectedCommand; } }

                public int CommandCodeValue { get { return (int)selectedCommand; } }

        }
}
```

Klasa CommandRecognizedEventArgs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RoboSpeechWin
{

public delegate void CommandRecognizedEventHandler(object sender, CommandRecognizedEventArgs e);

        public class CommandRecognizedEventArgs : EventArgs
        {
                public RoboCommand RoboCommand { get; protected set; }

                public CommandRecognizedEventArgs(RoboCommand command)
                {
                        RoboCommand = command;

                }


                public RoboConfirm RoboConfirm { get; protected set; }

                public CommandRecognizedEventArgs(RoboConfirm command)
                {
                        RoboConfirm = command;
                }


        }
}
```

Klasa Program

Klasa Program predstavlja glavnu ulaznu točku za aplikaciju, te aktivira GUI(Graphic User Interface) za grafički prikaz forme aplikacije, u što je također uključen i prikaz teksta.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace RoboSpeechWin
{
        static class Program
        {
                /// <summary>
                /// </summary>
                [STAThread]
                static void Main()
                {
                        Application.EnableVisualStyles();
                        Application.SetCompatibleTextRenderingDefault(false);
                        Application.Run(new Form1());
                }
        }
}
```

Klasa Form

Klasa Form se sastoji od grafičkog prikaza aplikacije i koda koji objedinjuje sve ostale klase i generalno definira način rada same aplikacije(Slika 9.).

Grafički prikaz aplikacije djelomično se sastoji od automatski generiranog koda, djelomično od objekata koji se moraju nadodati od strane programera, a definiraju i stavljaju u funkciju neke značajke u samom prozoru aplikacije(Textbox, Label, Button).

```
namespace RoboSpeechWin
{
        partial class Form1
        {
                /// <summary>
                /// Required designer variable.
                /// </summary>
                private System.ComponentModel.IContainer components = null;

                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
                /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
                protected override void Dispose(bool disposing)
                {
                        if (disposing && (components != null))
                        {
                                components.Dispose();
                        }
                        base.Dispose(disposing);
                }

                #region Windows Form Designer generated code
```

```
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        this.lblServerEcho = new System.Windows.Forms.Label();
        this.txtIpAddr = new System.Windows.Forms.TextBox();
        this.txtPort = new System.Windows.Forms.TextBox();
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.btnInitialize = new System.Windows.Forms.Button();
        this.SuspendLayout();
        //
        // Label koji prikazuje povratnu informaciju od servera.
        //
        this.lblServerEcho.Location = new System.Drawing.Point(15, 119);
        this.lblServerEcho.Name = "lblServerEcho";
        this.lblServerEcho.Size = new System.Drawing.Size(257, 134);
        this.lblServerEcho.TabIndex = 1;
        //
        // Textbox za upis IP adrese
        //
        this.txtIpAddr.Location = new System.Drawing.Point(70, 14);
        this.txtIpAddr.MaxLength = 15;
        this.txtIpAddr.Name = "txtIpAddr";
        this.txtIpAddr.Size = new System.Drawing.Size(104, 20);
        this.txtIpAddr.TabIndex = 2;
        this.txtIpAddr.Text = "127.0.0.1";
        //
        // Textbox za upis porta
        //
        this.txtPort.Location = new System.Drawing.Point(70, 40);
        this.txtPort.MaxLength = 5;
        this.txtPort.Name = "txtPort";
        this.txtPort.Size = new System.Drawing.Size(55, 20);
        this.txtPort.TabIndex = 3;
        this.txtPort.Text = "4120";
        //
        // Label koji označava textbox za upis IP adrese
        //
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(12, 21);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(52, 13);
        this.label1.TabIndex = 4;
        this.label1.Text = "IP adresa";
        //
        // Label koji označava textbox za upis porta
        //
        this.label2.AutoSize = true;
        this.label2.Location = new System.Drawing.Point(12, 47);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(26, 13);
        this.label2.TabIndex = 5;
        this.label2.Text = "Port";
        //
        // Button za incijalizaciju aplikacije
        //
        this.btnInitialize.Location = new System.Drawing.Point(99, 83);
        this.btnInitialize.Name = "btnInitialize";
        this.btnInitialize.Size = new System.Drawing.Size(75, 23);
        this.btnInitialize.TabIndex = 6;
        this.btnInitialize.Text = "Inicijalizacija";
        this.btnInitialize.UseVisualStyleBackColor = true;
        this.btnInitialize.Click += new System.EventHandler(this.btnInitialize_Click);
        //
        // Definiranje i povezivanje značajki prozora s određenim eventhandlerima
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(284, 262);
        this.Controls.Add(this.btnInitialize);
        this.Controls.Add(this.label2);
```

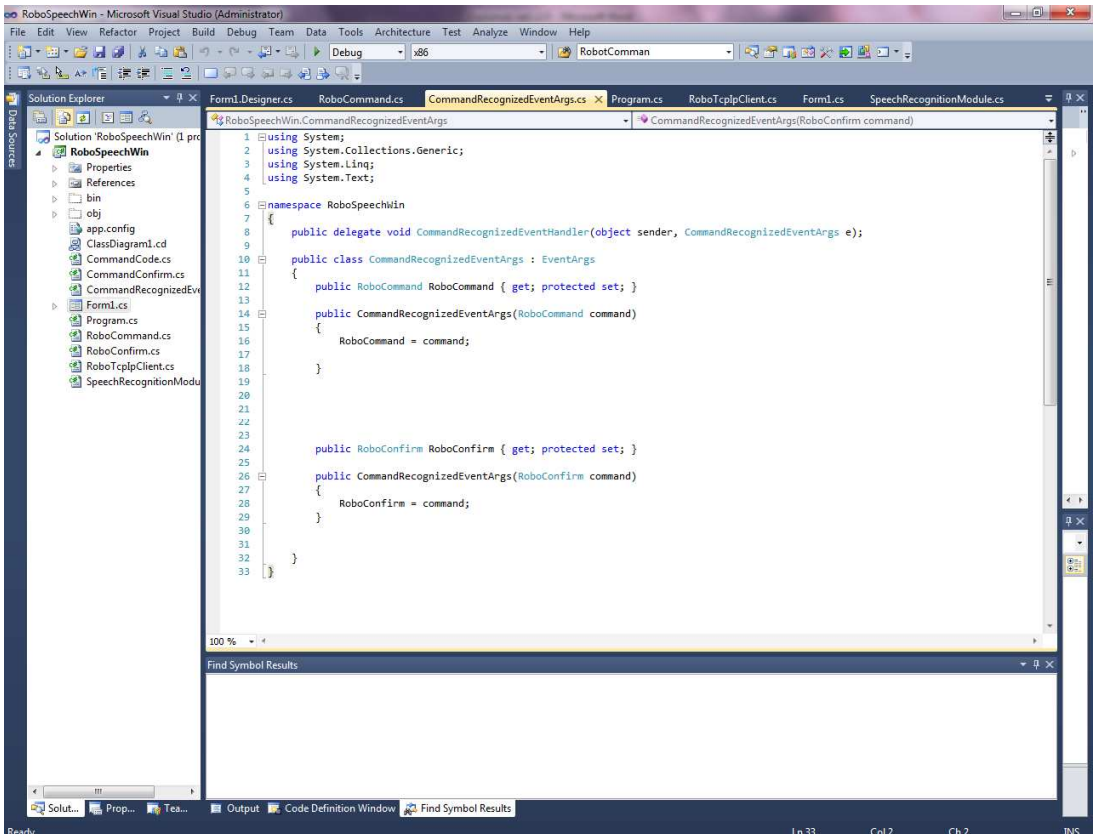
```
                        this.Controls.Add(this.label1);
                        this.Controls.Add(this.txtPort);
                        this.Controls.Add(this.txtIpAddr);
                        this.Controls.Add(this.lblServerEcho);
                        this.Name = "Form1";
                        this.Text = "RoboSpeech";
                        this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.Form1_FormClosing);
                        this.Shown += new System.EventHandler(this.Form1_Shown);
                        this.ResumeLayout(false);
                        this.PerformLayout();

                }

                #endregion

                private System.Windows.Forms.Label lblServerEcho;
                private System.Windows.Forms.TextBox txtIpAddr;
                private System.Windows.Forms.TextBox txtPort;
                private System.Windows.Forms.Label label1;
                private System.Windows.Forms.Label label2;
                private System.Windows.Forms.Button btnInitialize;
        }
}
```

Nadalje slijedi kod koji komunicira sa svim klasama i definira rad aplikacije RoboVox. Ovaj kod nalazi se u zasebnoj .cs datoteci.

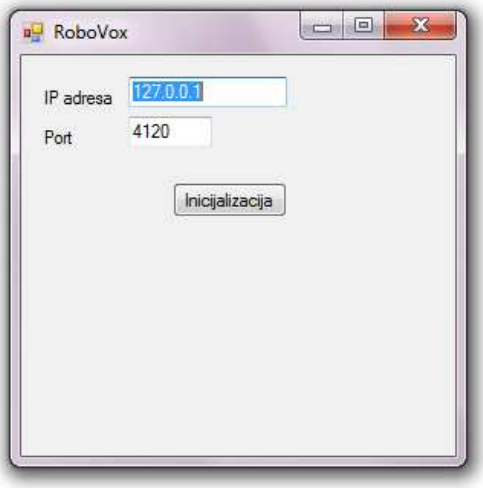
```
using System.Windows.Forms;
using SpeechLib;
using System;

namespace RoboSpeechWin
{
        public partial class Form1 : Form
        {

                private const int NEW_CMD_SLEEP_MILLIS = 200;

// Definiranje stanja aplikacije enumeratorom
                public enum State
                {
                        WaitingForCommand,
                        CommandRecognizedWaitingForConfirmation,
                        CommandConfirmed,
                        CommandRejected
                }

                private State currentState;

                RoboCommand lastReceivedCommand;

// Aktivacija klasa za sintezu i prepoznavanje glasa, te klase TCP/IP klijenta
                private SpVoice spVoice;
                private SpeechRecognitionModule srm;
                private RoboTcpIpClient tcpClient;
```

```
                public Form1()
                {
                        InitializeComponent();
                }


                private void Form1_Shown(object sender, System.EventArgs e)
                {
                }

// Inicijalizacija „button-a“ i „textbox-a“.
                private void btnInitialize_Click(object sender, EventArgs e)
                {
                        string inputIpAddr = txtIpAddr.Text.Trim();
                        string inputPort = txtPort.Text.Trim();
                        try
                        {
        tcpClient = new RoboTcpIpClient(System.Net.IPAddress.Parse(inputIpAddr), int.Parse(inputPort));
                        }
                        catch (Exception exc)
                        {
        MessageBox.Show(string.Format("Error: {0}", exc.Message), "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
                                return;
                        }


                        spVoice = new SpVoice();

                        srm = new SpeechRecognitionModule();
                srm.CommandRecognizedEvent += new CommandRecognizedEventHandler(CommandRecognized);

// U trenutku pokretanja programa i čekanja naredbe
                        currentState = State.WaitingForCommand;
                        spVoice.Speak("Speech engine ready");
                        btnInitialize.Enabled = false;
                }

// U slučaju zatvaranja programa , gasi se i TCP/IP veza.
                private void Form1_FormClosing(object sender, FormClosingEventArgs e)
                {
                        tcpClient.CloseTcpConnection();
                }


// Izmjenjivanje stanja za slučaje čekanja naredbe i čekanja potvrde naredbe.
                protected void CommandRecognized(object sender, CommandRecognizedEventArgs args)
                {
                        switch (currentState)
                        {
                                case State.WaitingForCommand:
                                        ProcessCommandReceived(args.RoboCommand);
                                        break;

                                case State.CommandRecognizedWaitingForConfirmation:
                                        ProcessConfirmationReceived(args.RoboConfirm);
                                        break;
                        }

                }


//Kada je naredba upućena definiranje postupaka za neprepoznata ili prepoznatu naredbu.
                private void ProcessCommandReceived(RoboCommand roboCommand)
                {
                        if (roboCommand.CommandCode == CommandCode.Code.UNDETERMINED)
                        {
                                lastReceivedCommand = null;
                                spVoice.Speak("Command not recognized. Please repeat...");
                }
                        else
                        {
                                lastReceivedCommand = roboCommand;
                                spVoice.Speak(roboCommand.CommandCode.ToString());
```

```
                                spVoice.Speak("Please confirm your command...");
                                srm.ToggleState();
                                currentState = State.CommandRecognizedWaitingForConfirmation;
                        }
                }

//Kada je potvrda naredbe upućena definiranje postupaka za neprepoznatu ili prepoznatu potvrdu naredbe.

                private void ProcessConfirmationReceived(RoboConfirm roboConfirm)
                {
                        switch (roboConfirm.CommandConfirm)
                        {
                                case CommandConfirm.YesNo.UNDETERMINED:
                                spVoice.Speak("Confirmation not recognized. Please repeat...");
                                        break;

                                case CommandConfirm.YesNo.NO:
                                        spVoice.Speak("Command cancelled...");
                                        srm.ToggleState();
                                        currentState = State.WaitingForCommand;

                                        System.Threading.Thread.Sleep(NEW_CMD_SLEEP_MILLIS);
                                        spVoice.Speak("Please enter new command.");
                                        break;

                                case CommandConfirm.YesNo.YES:
                                        try
                                        {
// Delegirati komandu TCP/IP modulu koji ce poslati naredbu na server.
                                var response = tcpClient.SendCommand(lastReceivedCommand.CommandCode);

        //lblServerEcho.Text = string.Format("{0} [{1}]: {2}",
        lastReceivedCommand.CommandCode.ToString(), lastReceivedCommand.CommandCodeValue, response);
                                        }
                                        //catch (Exception e)
                                        //{
                                //lblServerEcho.Text = string.Format("Error occured: {0}", e.Message);
                                        //}

                                        finally
                                        {
// Vratiti se u pocetno stanje cekanja naredbe.
                                                srm.ToggleState();
                                                currentState = State.WaitingForCommand;
                                                spVoice.Speak("Command executed.");

                                                System.Threading.Thread.Sleep(NEW_CMD_SLEEP_MILLIS);
                                                spVoice.Speak("Please enter new command.");
                                        }
                                        break;
                        }
                }


        }

}
```



Slika 10. Prikaz programskog koda unutar Visual Studio paketa



Slika 11. Prozor aplikacije RoboVox

6.2 Komunikacija klijent-server

Nakon dovršetka aplikacije RoboVox koja ima za glavni cilj prepoznavanje glasovnih naredbi i slanje istih na mrežu koristeći TCP/IP protokol preuzimajući ulogu klijenta, slijedeći korak je ostvarivanje komunikacije između klijenta i servera.

Komunikaciju se ostvaruje korištenjem opcije *Socket Messaging* koja omogućuje programeru razvijanje Karel aplikacije na robotu za komunikaciju s jedinstvenim protokolima koji su temeljeni na TCP/IP-u i „socket“ sučelju. Također, prvo se trebaju definirati TCP/IP parametri prije konfiguracije i uporabe ove opcije.[5]

Transmission Control Protocol (TCP) je namijenjen za korištenje kao pouzdan host-to host protokol između „host-ova“ kod razmjene pakata unutar računalnih komunikacijskih mreža. Uklapa se u slojevitu protokolarnu arhitekturu kao nadogradnja osnovnom *Internet Protocol-u* (IP). Internet Protocol omogućuje TCP-u slanje i primanje segmenata podataka promjenljive duljine priloženih u Internet datagram paketima.[5]

6.2.1 Komunikacija između „host-ova“ [5]

File Transfer Protocol(FTP) sučelje omugućava kontroleru u robotskom sustavu komunikaciju sa vanjskim ili „host“ uređajima preko eternet mreže. FTP koristi komunikaciju između „host-ova“ za obavljanje određenih operacija.

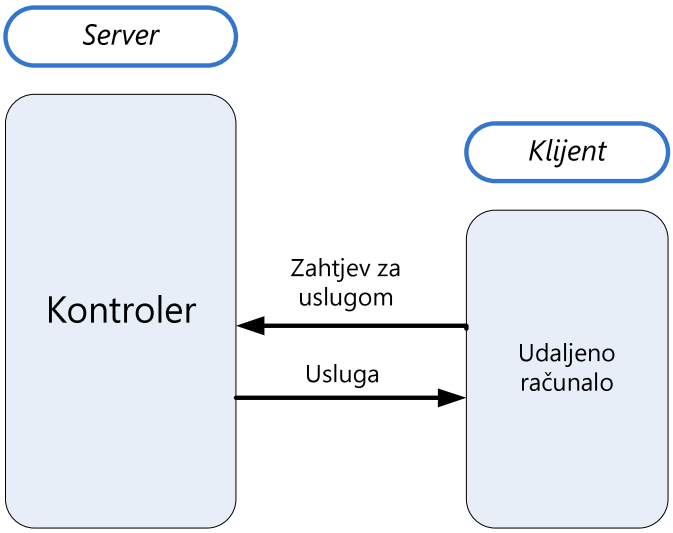
Arhitektura

Arhitektura za komunikaciju između „host-ova“ temelji se na klijent-server modelu. U tom modelu klijent predstavlja uređaj koji treba uslugu, a server uređaj koji pruža uslugu.

Kada se govori konkretno o tematici ovog rada, glasovnom upravljanju robotom, kontroler robotskog sustava preuzet će ulogu servera, dok udaljeno računalo na mreži kojim se glasovno upravlja robotskom rukom ulogu klijenta.

Server(poslužitelj)

Serveru robota može se pristupiti koristeći naziv uređaja servera, koji se naziva *tag*(oznaka, etiketa). Serveri za „host“ komunikaciju pokrenuti su na uređajima sa serverskim oznakama od S1: do S8:. Ovim uređajima ne može se pristupiti izravno. Server se uobičajeno pokreće na jednoj od ponuđenih oznaka i radi transparentno za kontroler. Uređaj koji ima ulogu klijenta imati će zahtjeve usluga prema serveru tj. kontroleru u robotskom sustavu.



Slika 12. Prikaz rada kontrolera u ulozi servera [5]

Uređaji

Uređaji za „host“ komunikaciju sastoje se od:

- Komunikacijske oznake(*tag*)
- Komunikacijskog protokola
- Neobaveznog imena serijskog port-a

Komunikacijski uređaji poznati sustavu moraju se prvo definirati. Definiranje komunikacijskog uređaja uključuje specificiranje komunikacijske oznake i protokola. Definiranje uređaja čini uređaj poznat sustavu, ali ne dodjeljuje sredstva potrebna uređaju. Za uklanjanje komunikacijskog uređaja iz sustava, mora se „oddefinirati“. To oslobađa oznaku tako da se može dodjeliti nekom drugom uređaju uređaju.

Serveri od S1: do S8: moraju se pokrenuti prije nego bilo koji zahtjevi za uslugom mogu biti upućeni od strane klijenta. Serveri započinju s radom automatski stavljanjem kontrolera u pogon.

6.2.2 Konfiguracija TCP/IP protokola [5]

Nekoliko parametara se koristi za konfiguriranje i postavljanje funkcija TCP / IP veze.

Robot name

Ova stavka određuje ime kontrolera robota. Unaprijed zadano ime robota je ROBOT. Ime ovog polja je zajedničko između eternet „port-ova“. U slučaju robota upotrebljenog pri izradi ovog diplomskog rada koristi se imr ROBOT2.

Port # IP Address

Ova stavka određuje jedinstvenu internet (IP) adresu za eternet sučelje robota. U slučaju robota upotrebljenog pri izradi ovog diplomskog rada koristi se IP adresa 192.168.123.26.

Port # označava da li se radi o port-u #1 (gornji RJ45 priključak označen kao CD38A) ili portu #2 (donji RJ45 priključak označen kao CD38B). Koristiti(F3) port FUNCTION tipku za promjenu portova za konfiguriranje.

Router IP Address

Ova stavka određuje Internet (IP) adresu „routera“. Ova postavka je zajednička između eternet „port-ova“. IP adresa „routera“ mora biti na istoj podmreži(subnet) kao i eternet „port-ovi“.

Subnet Mask

Ova stavka se koristi za razlikovanje lokalnih „host-ova“ od „host-ova“ s kojima mora uspostaviti komunikacija preko „router-a“. Unaprijed zadana vrijednost je 255.255.255.0.

Board Address

Ova stavka pokazuje eternet hardversku (MAC) adresu za eternet sučelje. Ovo polje je samo za čitanje.

Host Name

Ova stavka određuje internet ime „host-a“ s kojim će kontroler komunicirati kao s klijentom.

Internet Address

Ova stavka određuje odgovarajuću internet adresu svakog „host-a“ s kojim će kontroler komunicirati kao s klijentom.

Konfiguracija na Teach Pendantu

Slijedi konfiguracija prethodno navedenih stavki na samomTeac Pendantu:

1) Pritisnuti MENU
2) Odabrati SETUP
3) Pritisnuti F1, [TYPE]
4) Odabrati iz izbornika Host Comm. Pojaviti će se slijedeći ekran.

```
SETUP Protocols
    Protocol          Description
 1  TCP/IP            TCP/IP Detailed Setup
 2  FTP               File Transfer Protocol
 3  NONE              Connects tag to port
```

5) Odabrati TCP/IP
6) Pritisnuti F3, DETAIL. Pojaviti će se slijedeći ekran gdje definiramo prethodno navedene stavke

```
SETUP HOST COMM
TCP/IP
   Robot name:                  PDEROB024
   Port # IP addr:          172.22.194.24
   Subnet mask:             255.255.240.0
   Board address:       08:00:19:02:F2:22
   Router IP addr:           172.22.192.1

Host Name (LOCAL)    Internet Address
     1  *********    ******************
     2  *********    ******************
     3  **********   ******************
     4  **********   ******************
     5  **********   ******************
     6  **********   ******************
     7  **********   ******************
     8  **********   ******************
     9  **********   ******************
     10  **********   ******************
     11  **********   ******************
     12  **********   ******************
     13  **********   ******************
     14  **********   ******************
     15  **********   ******************
     16  **********   ******************
  Host Name (SHARED) Internet Address
     1  **********   ******************
     2  **********   ******************
     3  **********   ******************
     4  **********   ******************
     5  **********   ******************
     6  **********   ******************
     7  **********   ******************
     8  **********   ******************
     9  **********   ******************
     10  **********   ******************
     11  **********   ******************
     12  **********   ******************
     13  **********   ******************
     14  **********   ******************
     15  **********   ******************
     16  **********   ******************
     17  **********   ******************
     18  **********   ******************
     19  **********   ******************
     20  **********   ******************
```

6.2.3 Socket Messaging [5]

Socket Messaging opcija daje prednost korištenja TCP/IP „socket“ poruka iz KAREL-a. *Socket Messaging* omogućuje razmjenu podataka između mreže 200iC robota i udaljenog računala s Windows, Linux ili Unix radnim stanicama. Za tematiku ovo diplomskog rada primjena *Socket Messaging-a* je da 200iC robot izvodi Karel program koji dobiva ulazne podatke sa udaljenog računala.

Socket Messaging koristi TCP/IP protokol za prijenos neformtiranih podataka ili podataka u svom izvornom obliku, preko mreže. Naredbe i metode koje *Socket Messaging* koristi za prijenos podataka su dio TCP/IP protokola. Budući da *Socket Messaging* podržava oznake klijente i servera, aplikacije koje zahtijevaju vremensku pauzu ili naredbe formatiranja podataka mogu pružiti ovu dodatnu semantiku s obje strane *Socket Messaging* veze tj. klijenta i servera.

Konfiguracija Socket Messaging-a

U svrhu korištenja *Socket Messaging-a*, moraju se podesiti sljedeći mrežni hardverski i softver parametri:

- Na serveru:

  -port koji se želi upotrebljvati za *Socket Messaging* (u slučaju ovog diplomskog rada to je port 12350)

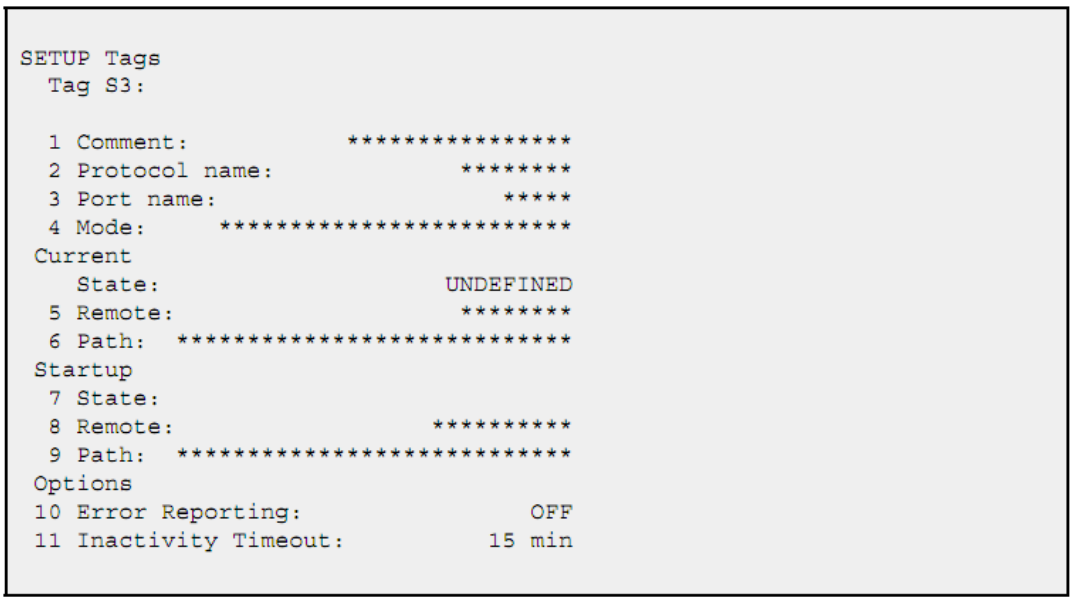
- Na klijentu:

  -IP adresa ili ime servera (u slučaju ovog diplomskog rada to je IP adresa 192.168.123.26)
  -port na serveru koji se upotrebljava za *Socket Messaging*(port 12350)

Zbog toga što se konkretno u slučaju ovog diplomskog rada server nalazi na robotskom sustavu, a udaljeno računalo s kojeg se šalju naredbe ima ulogu klijenta moraju se konfigurirati serverske oznake. Treba paziti da serverske oznake koje će se koristiti ne oristi niti jedan drugi mrežni protokol, osim TCP/IP-a, a ako je to slučaj treba oddefinirati oznake prije upotrebe za *Socket Messaging.*

Za postavljanje serverskih oznaka potrebno je učiniti slijedeće korake:

1) Na Teach Pendantu pritisnuti tipku MENU.
2) Odabrati SETUP.
3) Pritisnuti F1, [TYPE].
4) Odabrati iz izbornika Host Comm.
5) Pritisnuti F4, [SHOW].
6) Odabrati Servers .
7) Pomaknuti kursor do oznake severa koja se želi osposobiti za *Socket Messaging* i pritisnuti F3,DETAIL. U okviru ovog diplomskog rada koristi se oznaka S5: .



8) Pomaknuti kursor do Protocol name i pritisnuti F4, [CHOICE].
9) Odabrati SM.
10) Pomaknuti kursor do Startup State i pritisnuti F4, [CHOICE].
11) Odabrati START.
12) Pritisnuti F2, [ACTION].
13) Odabrati Define.
14) Pritisnuti F2, [ACTION].
15) Postavljanje sistemskih varijabli:

    a) Pritisnuti MENU.
    b) Odabrati NEXT.
    c) Odabrati SYSTEM i pritisnuti F1, [TYPE].
    d) Odabrati Variables.
    e) Pomaknuti kursor do $HOSTS_CFG i pritisnuti ENTER.

f) Pomaknuti kursor do reda s brojem koji odgovara oznaci servera(u ovom slucaju S5:).

```
SYSTEM Variables
$HOSTS_CFG
   1    [1]           HOST_CFG_T
   2    [2]           HOST_CFG_T
   3    [3]           HOST_CFG_T
  4    [4]           HOST_CFG_T
   5    [5]           HOST_CFG_T
   6    [6]           HOST_CFG_T
   7    [7]           HOST_CFG_T
   8    [8]           HOST_CFG_T
```

g) Pritisnuti ENTER. Slijedi ekran sličan slijedećem.

```
SYSTEM Variables
$HOSTS_CFG[3]
  1 $COMMENT         *uninit*
  2 $PROTOCOL        'SM'
  3 $PORT            *uninit*
  4 $OPER            3
  5 $STATE           3
  6 $MODE            *uninit*
  7 $REMOTE          *uninit*
  8 $REPERRS         FALSE
  9 $TIMEOUT         15
 10 $PATH            *uninit*
 11 $STRT_PATH       *uninit*
 12 $STRT_REMOTE     *uninit*
 13 $USERNAME        *uninit*
 14 $PWRD_TIMOUT     0
 15 $SERVER_PORT     0
```

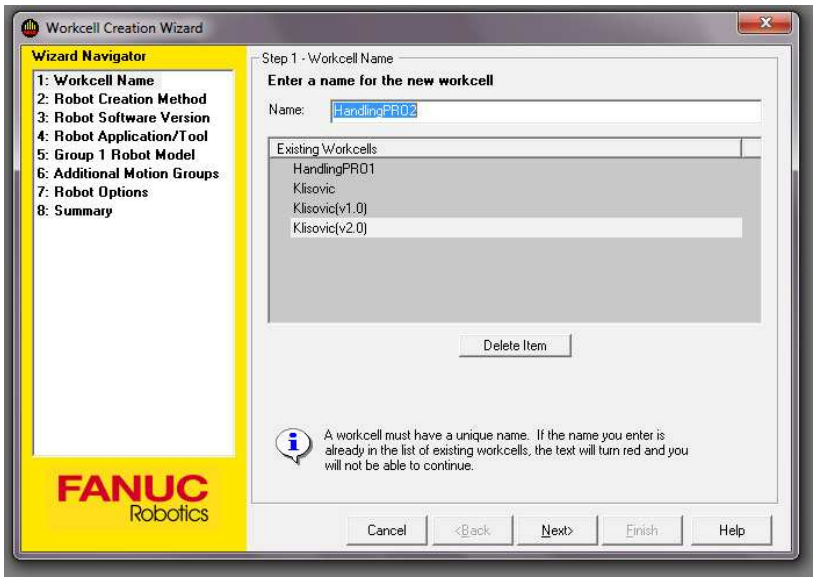
h) Pomaknuti kursor na $SERVER_PORT. Utipkati ime TCP/IP porta koji se upotrebljava za *Socket Messaging*. Sada se može upotrebljavati serverska oznaka iz Karel programa.

6.3 Izrada programa u Karelu

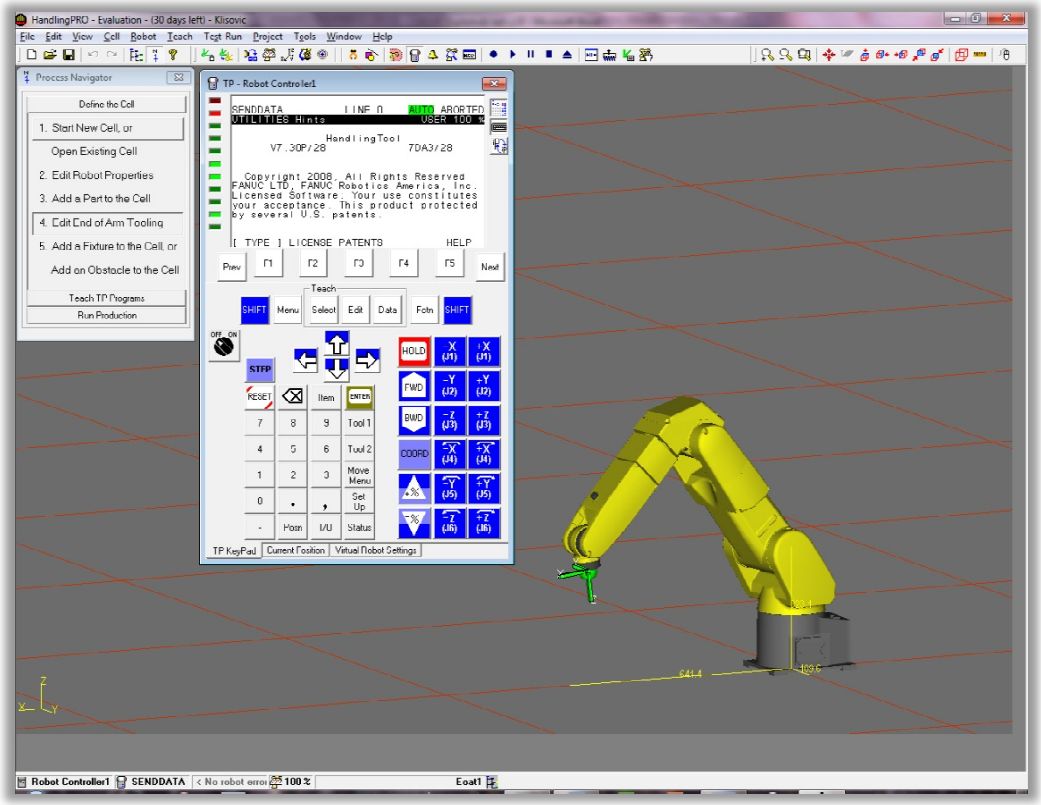
Kao što je već napomenuto Karel je integralni dio programskog paketa RoboGuide. Tako da je jedan od prvih koraka pokrenuti Roboguide programski paket koji se nalazi na računalu koje je u sklopu Fanuc robotskog sustava, te unutar njega napraviti novu radnu ćeliju. Unutar ovog diplomskog rada neće se opisivati cijeli Roboguide programski paket i sve njegove mogućnosti s obzirom da je ta tematika obrađena u nekim prijašnjim diplomskim radovima.

Pri izradi nove ćelije podešavamo niz postavki koji uključuju odabir modela robotske ruke, alata koji se nalazi na kraju robotske ruke, te okruženja u kojem se robotska ruka nalazi. Pri završetku Roboguide generira CAD model robotske ruke i alata, izrada ćelije je završena, te je sve spremno za virtualno manipuliranje CAD modelom robotske ruke, alatom, te za pisanje Karel ili Teach Pendant upravljačkih programa. Upravljanje CAD modelom robotske ruke vrši se preko virtualnog Teach Pendant-a. Važno je napomenuti da sve postavke koje smo podestili pri izradi nove ćelije daju se naknadno prepraviti preko *Process Navigator* prozora unutar Roboguid-a.

Za ovaj diplomski rad odabran je model robota Fanuc LR Mate 200iC/5L.



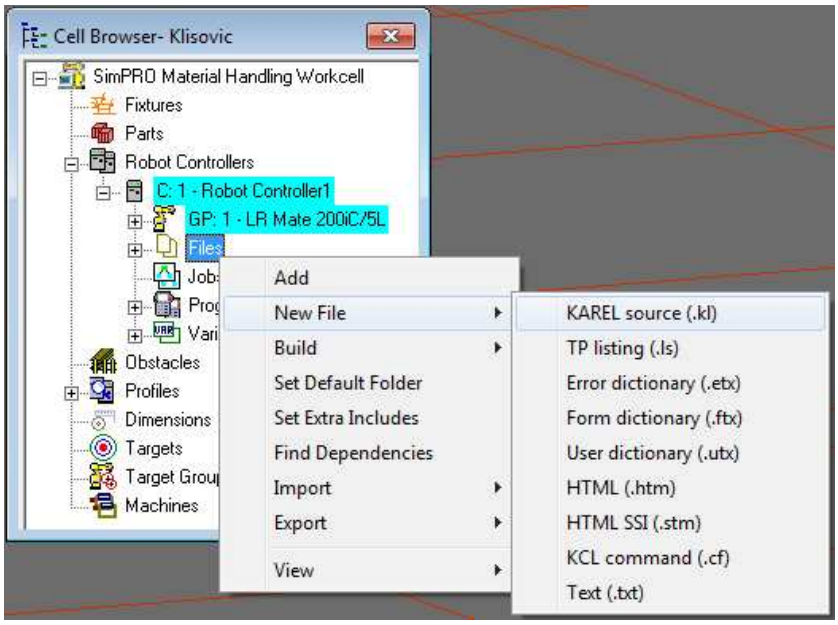
Slika 13. Prikaz stvaranja nove radne ćelije unutar Roboguide-a



Slika 14. Prikaz CAD modela robotske ruke LR Mate 200iC s virtualnim Teach Pendantom i otvorenim *Process Navigator* prozorom

Nadalje, unutar *Cell Browser* prozora nalazi se drvo svih procesa vezanih za trenutnu radnu ćeliju. Pri tome se misli na sve djelove, kontrolere, robotske ruke s pridajućim Karel i Teach Pendant upravljačkim programima, kompletno okružje u kojem se robotska ruka nalazi npr. transportne linije, te razne prepreke koje se nalaze u stvarnim okružjima gdje će se robotska ruka primjenjivati.

Unutar drveta radne ćelije kursorom se odabire opcija Robot Controllers/ Robot Controller1 /Files, gdje se klikom desne tipke miša odabire opcija New File /Karel Source. Ovim postupkom otvara se prozor za pisanje koda u Karelu.



Slika 15. Stvaranje nove Karel datoteke za pisanje upravljačkog koda

6.3.1 Karel programski kod

Socket Messaging je komponenta integrirana u Karel. Kada se upotrebljavaju *Socket Messaging* funkcije i usluge iz Karel programa, sintaksa je slična standardnim „read“ i „write“ operacijama, osim što je potrebno uspostaviti vezu sa mrežom.

Konkretno u slučaju ovog diplomskog rada postoje dvije Karel datoteke tj. dva Karel programska koda. Jedan služi za uspostavu veze servera s mrežom, te sadrži takozvane rutine na koje se drugi Karel kod(datoteka) poziva. U drugom kodu se obrađuju podaci koji su stigli preko mreže tj. integer oznake koje trigeriraju pojedine Teach Pendant programe.

Slijedeće Karel *Socket Messaging* funkcije omogućavaju serveru uspostavu veze sa udaljenim „host-om“ na mreži.

MSG_CONN(string,integer)

Treba se pozvati MSG_CONN funkcija prije nego što se i jedna oznaka servera može upotrijebiti za *Socket Messaging*. Prvi parametar ove naredbe sadrži naziv oznake (npr. "S5:"), a drugi parametar je cijeli broj koji sadrži status operacije. Ako se ova naredba koristi za spajanje na server tag, ova naredba će vratiti status vrijednost tek nakon što udaljeni klijent uređaj uspostavi vezu s ovom serverskom oznakom. [5]

Tijekom *Socket Messaging-a*, mora se koristiti MSG_DISCO za zatvaranje „socket“ veze s oznakama klijenta ili servera prije bilo kakvog naknadnog pokušaja spajanja na klijent ili server oznaku upotrebom MSG_CONN.

MSG_DISCO(string,integer)

MSG_DISCO se koristi za zatvaranje *Socket Messaging* veze. Ako je veza je prekinuta, možda zbog READ ili WRITE pogreške, mora se koristiti MSG_DISCO funkcija za zatvaranje veze s udaljenim klijentom ili serverom, prije nego što se MSG_CONN može se koristiti za uspostavu druge veze s udaljenim klijentom ili serverom.[5]

LIB_FILE.KL Karel programski kod

Kao što je već spomenuto ovaj kod u Karelu predstavlja skupine rutina za uspostavu veze sa određenom serverskom oznakom. Slijedeći Karel kod opisan u ovom diplomskom radu pozivat će se na rutine iz ovog koda za uspostavu te veze.

Rutine, slične u strukturi programu, pružaju način modulariziranja KAREL programa. Rutine mogu uključivati VAR i / ili CONST deklaracije i izvršna izjave. Za razliku od programa, rutina mora biti deklarirana velikim slovima u programu, te ne može obuhvatiti druge deklaracije rutina.[6]

Karel podržava dva tipa rutina:

- Proceduralne rutine - ne vraćaju vrijednost
- Funkcijske rutine - vraćaju vrijednost

Karel rutine mogu biti predefinirane rutine koje se nazivaju ugrađenim rutinama ili mogu biti definirane od korisnika.

Slijedeća pravila vrijede za sve Karel rutine [6]:

- Mogu se uključiti parametri u deklaraciju rutine. To omogućava proslijeđivanje podataka rutini u trenutku kada je pozvana i vraća rezultate programu od kojeg je pozvana.
- Rutine se mogu pozivati:
  - Od programa u kojem su deklarirane
  - Od bilo koje rutine sadržane u tom programu
  - S pomoću deklaracija nekog drugog programa

```
PROGRAM LIB_FILE
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE

--***************************VARIJABLE*************************************************
VAR
  STATUS,entry:INTEGER
--*************************************************************************************

  --*************************************************************************************
  --*****************************   RUTINE   ********************************************
  --*************************************************************************************

  ROUTINE ROBOT_(ID_:STRING; HOSTNAME_:STRING): ARRAY OF INTEGER FROM LIB_BASE
  ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING)
    VAR
        CLIENT:BOOLEAN; SC:STRING[1]; i:INTEGER
    BEGIN
      CLIENT=FALSE; SC = SUB_STR(TAG_, 1, 1);IF SC='C' THEN; CLIENT=TRUE; ENDIF
      CONNECT_::;
      CLR_IO_STAT(FILE_)
      MSG_DISCO(TAG_,STATUS); WRITE('CONNECTING.... ',TAG_,CR);
      MSG_CONNECT(TAG_,STATUS)
      OPEN FILE FILE_('rw',TAG_); STATUS = IO_STATUS(FILE_)

      IF STATUS<>0 THEN
        FOR i=1 TO 3 DO
          WRITE(CHR(128),CR); WRITE('RECONNECTING AT...',3-i,CR); DELAY 10
        ENDFOR
        GOTO CONNECT_
      ENDIF

      WRITE('Status open file:',STATUS,' TAG:',TAG_,CR)
    END OPEN_FILE_
```

```
--*****************************************************************************
ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING)
  BEGIN
    CLOSE FILE FILE_ ;  WRITE('Disconnecting..',CR)
    MSG_DISCO(TAG_,STATUS); WRITE(TAG_,'status disco:',STATUS,CR)
  END CLOSE_FILE_
--*****************************************************************************
-- ID:  OZNAKA ROBOTA SA KOJIM SE ZELI KOMUNICIRATI: >R1< . >R2< . >R3<
-- TIP: oznaka koja oznacava da li se podaci salju ili primaju: >S< . >C<
-- SERVER je onaj koji prvi otvarat vezu tj. daje DO i ceka na DI
-- CLIENT je onaj koji prvo ceka na servera sa wait DI, i onda daje izlaz DO da je primio
-- konekciju od SERVERA
ROUTINE HANDSHAKING_ (ID_ : STRING; TIP_ : STRING)
  VAR
    NIZ : ARRAY[10] OF INTEGER

  BEGIN
    --DOHVAT PODATAKA O ID-U TJ. ROBOTU SA KOJIM SE ZELI NAPRAVITI HANDSHAKING
    NIZ = ROBOT_(ID_,'')
   --SERVER:
    IF TIP_ = 'S'  THEN
      WRITE('DOUT',NIZ[2],'=ON',CR)
      DOUT[ NIZ[2] ] = ON
      WRITE('SERVER WAIT FOR DIN:',NIZ[2],CR)

      WHILE DIN[ NIZ[1] ] = OFF DO
         DOUT[ NIZ[2] ] = ON
         DELAY 100
      ENDWHILE

      DELAY 150
      DOUT[ NIZ[2] ] = OFF
      ENDIF
```

```
--CLIENT:
      IF TIP_ = 'C' THEN
        WRITE('CLIENT WAIT FOR DIN:',NIZ[1],CR)
        WAIT FOR DIN[ NIZ[1] ] = ON
        DOUT[ NIZ[2] ] = ON
        DELAY 150
        DOUT[ NIZ[2] ] = OFF
      ENDIF
    END HANDSHAKING_
  --*************************************************************************
  ROUTINE WRITE_(STRING_ : STRING)
    BEGIN
      WRITE(STRING_,CR);
    END WRITE_
BEGIN
END LIB_FILE
```

ROBOVOX_COMM Karel programski kod

Ovaj kod(program) poziva rutine definirane u LIB_FILE kodu(programu) za uspostavu veze sa određenim serverskim oznakama. Točnije koristi se rutinama OPEN_FILE_ (FILE_ : FILE; TAG_ : STRING) i CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING) koji sadrže prethodno opisane MSG_CONN(string,integer) i MSG_DISCO(string,integer) funkcije.

Nadalje, Teach Pendant programi koji se pozivaju iz ROBOVOX_COMM programa trigerirani integer oznakama dobivenim od udaljenog klijenta prethodno su ručno definirani na Teach Pendantu standardnom procedurom za to namjenjenom. U slučaju ovog diplomskog rada oni su samo simbolični, predstavljaju neke osnovne radnje robotske ruke služe samo za demonstraciju glasovnog upravljanja robotom. Cijela je poanta da se bilo koji Teach Pendant programi mogu pozivati ovim Karel programom ovisno o željama korisnika.

U ROBOVOX_COMM Karel programu upotrebljavaju se slijedeće Karel funkcije [6]:

- READ funkcija se koristi za čitanje jedne i više specificiranih stavki podatka sa naznačenog uređaja. Neka od navedenih pravila vrijede za READ funkciju:
    - Kada se izvršava READ funkcija (za ASCII datoteke), podaci se čitaju s početkom na prvom ulaznom znaku koji ne prestavlja prazno mjesto i završetkom na zadnjem znaku prije praznog mjesta, kraja linije, kraja datoteke za sve tipove ulaza osim za tip ulaza STRING.
    - Kod STRING vrijednosti, ulazno polje počinje sa slijedećim znakom i nastavlja se sve do kraja linije ili kraja datoteke. Ako nakon STRING znaka slijedi polje koje nije STRING, sva prazna mjesta koja služe za odvajanje uključena su u STRING.
    - ARRAY varijable moraju se čitati element po element.
- WRITE funkcija se koristi za zapisivanje jedne i više specificiranih stavki podatka na naznačeni uređaj.
    - ARRAY varijable moraju se zapisivati element po element.
- IO_STATUS funkcija vraća integer oznaku koja ukazuje na uspjeh ili tip pogreške tijekom zadnje operacije. Može s upotrebljavati nakon OPEN FILE, READ, WRITE, CANCEL, CLOSE FILE funkcija. U ovisnosti o rezultatu operacije vraća 0 ako je bila uspješna ili jedan od tipova pogreške u slučaju neuspjeha.
- BYTES_AHEAD(file_id, n_bytes, status) funkcija vraća broj „byte-ova“ ulaznih podataka koji se nalaze trenutačno u spremniku(buffer-u) koji se čita unaprijed. Dopušta Karel programu da trenutačno provjeri dali su podaci primljeni sa serijskog porta i dali su dostupni za čitanje od strane programa.
    - *file_id* određuje datoteku koja je otvorena
    - *file_id* mora biti sa ATR_READAHD atributom postavljenim na vrijednost veću od nule
    - *n_bytes* predstavlja broj byte-ova u spremniku
    - *status* predstavlja status operacije. Ako nije jednak 0, dogodila se pogreška

U ovom diplomskom radu BYTES_AHEAD se koristi za pročišćavanje porta od bilo kakvih byte-ova preostalih za čitanje.

- CALL_PROG(prog_name, prog_index) funkcija dopušta KAREL programu da pozove vanjski Karel ili Teach Pendant program.
  - prog_name prestavlja ime programa koji se treba izvršiti
  - prog_index vraća broj reda programa iz tablice programa u kojoj se program nalazi. Ovu varijablu nije potrebno inicijalizirati, te niju je preporučljivo mijenjati

```
PROGRAM robovox_comm
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
VAR

  i,n,tmp_int,STATUS:INTEGER
  file_var:FILE
  vox_str:STRING[128]
  n_bytes,STAT,prog_index:INTEGER
  FINISHED:BOOLEAN

  ----------------------VANJSKE RUTINE-------------------------
  ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
  ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
  ROUTINE WRITE_(STRING_ : STRING) FROM LIB_FILE
  ROUTINE HANDSHAKING_(ID_ : STRING; TIP_: STRING) FROM LIB_FILE
  -------------------------------------------------------------
 79
BEGIN
  SET_FILE_ATR(file_var,ATR_IA)
  stat=SET_PORT_ATR (PORT_1, ATR_READAHD,1) -- čitati unaprijed buffer od 128 byta79
  --postavljanje port-a na serveru prije uspostave konekcije
  SET_VAR(entry, '*SYSTEM*','$HOSTS_CFG[5].$SERVER_PORT',12350,STATUS)
79
  --Spajanje tag-a
  WRITE TPDISPLAY('Uspostava veze sa R2...',CR)
  CLOSE_FILE_(file_var,'S5:')
  OPEN_FILE_(file_var,'S5:')

  REPEAT
  BYTES_AHEAD (file_var, n_bytes, STAT)--dohvati broj byteova spremnih za pročitati
  IF (n_bytes >= 1) THEN --postoje bytovi za pročitati
  READ file_var(vox_str::1) --pročitaj byte po byte
  stat=IO_STATUS (file_var) --status operacije read
  ENDIF
  UNTIL stat <> 0 --nastavi dok ne preostane niti jedan byte

  IF IO_STATUS <>0 THEN
    FINISHED=TRUE
  ENDIF

  REPEAT
  FINISHED=FALSE
  --Čitanje dolazne naredbe "Robovox go up"
  READ file_var(vox_str::3)
  IF UNINIT(vox_str) THEN
   vox_str=''
   ENDIF
```

```
  IF (vox_str='120') THEN
  CALL_PROG('NIK_UP',prog_index)
  ENDIF

   --Čitanje dolazne naredbe "Robovox go down"

  READ file_var(vox_str::3)
  IF UNINIT(vox_str) THEN
   vox_str=''
   ENDIF
  IF (vox_str='130') THEN
  CALL_PROG('NIK_DOWN',prog_index)
  ENDIF

   --Čitanje dolazne naredbe "Robovox go left"
   READ file_var(vox_str::3)
   IF UNINIT(vox_str) THEN
    vox_str=''
   ENDIF
   IF (vox_str='100') THEN
   CALL_PROG('NIK_LEFT',prog_index)
   ENDIF

   --Čitanje dolazne naredbe "Robovox go right"
   READ file_var(vox_str::3)
   IF UNINIT(vox_str) THEN
    vox_str=''
   ENDIF
   IF (vox_str='110') THEN
   CALL_PROG('NIK_RIGHT.tp',prog_index)
   ENDIF

   --Čitanje dolazne naredbe "Robovox grab"
   READ file_var(vox_str::3)
   IF UNINIT(vox_str) THEN
    vox_str=''
   ENDIF
   IF (vox_str='200') THEN
   CALL_PROG('NIK_GRAB.tp',prog_index)
   ENDIF

   --Čitanje dolazne naredbe "Robovox release"
   READ file_var(vox_str::3)
   IF UNINIT(vox_str) THEN
    vox_str=''
   ENDIF
   IF (vox_str='210') THEN
   CALL_PROG('NIK_RELEASE.tp',prog_index)
   ENDIF

   --Čitanje dolazne naredbe "Robovox stop"
   READ file_var(vox_str::3)
   IF UNINIT(vox_str) THEN
    vox_str=''
   ENDIF
   IF (vox_str='20') THEN
   CALL_PROG('NIK_STOP.tp',prog_index)
   ENDIF


   UNTIL (FINISHED=TRUE)

   CLOSE_FILE_(file_var,'S5:')

END robovox_comm
```

7. Zaključak

Najbolji TTS sustavi sada su u mogućnosti proizvesti sintetički govor, u neutralnom govornom stilu čitanja, te rezultat zvuči razumljivo i prirodno za mnoge kratke odlomke teksta. Međutim, sustavi koji daju najbolju kvalitetu govora posjeduju vrlo mali izbor različitih glasova, te obično imaju vrlo malo fleksibilnosti u mijenjanju karakteristika glasa ili govornog stila. Nadalje, u slučaju obrade odlomaka od samo nekoliko rečenica je čak i najbolje sustave dosadno slušati. Ako je potreban izražajan govorni stil, kvaliteta govora generirana od strane TTS sustava još uvijek nije stvarno dovoljno dobra da bude korisna, osim korisnicima koji nemaju drugog izbora. Postizanje raznolikosti, fleksibilnost i izražajnosti u sintezi govora zahtijeva istraživanje za poboljšanje na svim razinama procesa generiranja automatskog govora.

Trenutačno ASR performanse mogu biti vrlo impresivne, čak i za zadatke koji uključuju vrlo velike rječnike. Međutim, postoje nedostaci. Konkretno, ASR sustavi imaju tendenciju velike osjetljivosti na varijacije: promjene u zvučnoj kulisi okoliša, prijenosnom kanalu, identitetu govornika, govornom stilu itd. To sve uzrokuje mnogo više problema za prepoznavanje govora računalima nego ljudima. Najuspješniji ASR sustavi do sada se gotovo univerzalno temelje na HMM, kao što je opisano u poglavlju 4. Tijekom godina bilo je mnogo poboljšanja načina na koji se HMM koristi, a čini se vjerojatnim da će se ta poboljšanja nastaviti i u bliskoj budućnosti. Međutim, postoje interesi za istraživanjima u smjeru više značajnijih promjena i napretka izvan trenutne HMM metode. Nova istraživanja bi mogla dovesti do poboljšanja u performansama prepoznavanja govora.

Postoji rastuća potražnja za interaktivnim govornim sustavom koji uključuju dijalog između osoba i računala. U budućnosti biti će potrebna veća prirodnost kako u jeziku koji čovjek koristi, tako i u odgovorima koje generira sustav. Takva prirodnost je vjerojatno samo ostvariva ako računalo ima dobar model interakcije i neko „razumijevanje“ informacija koje se razmjenjuju u govoru. Poteškoće se obično smatraju općenitim problemima umjetne inteligencije. Iako dostignuća na ovom području su impresivna, govorni jezik postavlja dodatne izazove. Konkretno, spontani govor može biti znatno drugačiji od pročitanog govora: ne samo što govor ima tendenciju da je više opušten i uključuje kolebanja, ispravke itd., nego je korištenje jezika vrlo drugačije kada je dio interaktivne komunikacije. Budućnost govornih sustava trebati će imati uključen model tih efekata, kako za potrebe visokih performansi u prepoznavanju govora, tako i za prirodno generiranje govora. Drugi aspekt rastuće važnosti je potražnja za višejezičnim govornim sustavima. Višejezični sustav treba obuhvaćati temeljni prikaz pojmova, zajedno s metodama za povezivanje tih pojmova s iskazima na različitim jezicima na način da iskorištavaju istovjetnost između različitih jezika, dok se također oblikuju i razlike između njih. Takva sposobnost je vjerojatno potrebna ako se želi postići veliki napredak u najviše izazovnom problemu prijevoda govornog jezika (prepoznavanje u jednom jeziku i sinteza u drugom).

Metode umjetne inteligencije morati će koristiti informacije o trenutnom govornom komunikacijskom zadatku bez obzira odvija li se sinteza ili prepoznavanje. Konkretno, znanje o tematici razgovora je vrlo važno za generiranje i tumačenje iskaza u dijalogu između čovjeka i računala, a mora uključivati i efekt prethodnih iskaza u svrhu očekivanja onoga što će uslijediti. Do sada, najbolji govorni sustavi rezultat su mnogo ručnog podešavanja za specifičnu domenu primjene, pa postavljanje sustava za novu domenu je dugotrajan i naporan rad. Potrebne su kvalitetne automatske metode za treniranje modela za semantiku domene iz odgovarajućih materijala.

Poboljšanja u tehnologiji sinteze i prepoznavanja govora trebala bi doći s razvojem boljih modela govornog jezika, dostizanjem svih razina odnosa između apstraktnih jezičnih koncepata i generiranog zvučnog signala.

8. Literatura

[1] Holmes.J, Holmes.W, Speech Synthesis and Recognition, Taylor&Francis, New York, 2003.

[2] Wikipedia, http://en.wikipedia.org/wiki/Microsoft_Speech_API

[3] MSDN, http://msdn.microsoft.com/en-us/library/ee125077%28v=VS.85%29.aspx

[4] Wikipedia, http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29

[5] FANUC Robotics SYSTEM R-J3iC Controller Internet Options Setup and Operations Manual, 2006.

[6] FANUC Robotics SYSTEM R-30iA Controller KAREL Reference Manual, 2007.