NTNU
Det skapende universitet

Høykvalitets norsk talesyntese

Rasmus Rane Bauck

Master i elektronikk
Oppgaven levert: Juli 2009
Hovedveileder: Torbjørn Svendsen, IET

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

I prosjektet FONEMA samarbeidet NTNU og Telenor R&I om å utvikle verktøy og metoder for høykvalitets norsk talesyntese basert på prinsippet med enhetsutvelgelse. I prosjektet er det etablert databaser for norsk talesyntese, og vi har utviklet verktøy som automatisk kan generere syntetiske stemmer fra slike databaser. Noen av de gjenstående utfordringene angår kvalitet og effektivitet i syntesen. Denne oppgaven vil gå ut på å utforske og videreutvikle de metodene som benyttes for automatisk stemmegenerering for å forbedre kvaliteten på den syntetiske stemmen og å vurdere hvordan den genererte syntesedatabasen kan effektiviseres for å redusere kravene til minne og beregningskapasitet.

Oppgaven gitt: 16. januar 2009
Hovedveileder: Torbjørn Svendsen, IET

Sammendrag

Talesyntese er en teknologi som søker å omdanne skreven tekst til naturlig tale. Arbeidet i denne masteroppgaven har tatt utgangspunkt i resultatene fra FONEMA-prosjektet, et samarbeid mellom NTNU og Telenor. Dette prosjektet har produsert verktøy og metoder for å kunne utføre høykvalitets norsk talesyntese, men det er ønske om å forbedre ytelsen til disse systemene. Denne masteroppgaven har fokusert på to punkter i den forbindelse: effektivisering av syntesedatabase og forbedring av kvalitet for syntetisert stemme.

TTS-systemet som har blitt brukt i denne masteroppgaven utfører talesyntese ved hjelp av teknikken skjøtesyntese. Dette krever en stor syntesedatabase bestående av en stor mengde lydklipp med innlest tale. Systemet henter så ut biter av disse lydklippene og setter de sammen for å syntetisere en vilkårlig setning. Syntesedatabasen som har blitt brukt inneholder opprinnelig 5.363 lydklipp, og i en omfattende analyse av ble det avdekket en klar skjevhet i hvor ofte de forskjellige lydklippene faktisk ble benyttet til syntetisering. På bakgrunn av analyseresultatene har det blitt utviklet to nye syntesedatabaser som kun inneholder 90 % av lydklippene fra den opprinnelige syntesedatabasen, mens de 10 % minst brukte ytringene er tatt vekk. Til tross for reduksjonen, viser objektive og subjektive tester at ytelsen er like god for alle de tre syntesedatabasene. Det er ingen markant forskjell på databasenes difondekning eller hvilke setninger som kan syntetiseres, og den produserte talen er i stor grad av samme kvalitet.

Det opprinnelige TTS-systemet produserer i stor grad tale av god kvalitet, men håndterer ikke situasjoner hvor en ytring med difoner som ikke eksisterer i syntesedatabasen forsøkes syntetisert. Resultatet er at ingen deler av en slik ytring lar seg syntetisere. Det har derfor blitt utviklet et sett med substitusjons- og omskrivningsregler for å endre om på den fonemiske transkripsjonen til ytringene dette gjelder. Tanken er å bytte ut et difon med et annet som høres tilnærmet likt ut, men som også er representert i syntesedatabasen. Arbeidet har resultert i at alle setninger lot seg syntetisere under testing av systemet. I tillegg er den produserte talen i stor grad av god eller akseptabel kvalitet.

Forord

Denne masteroppgaven er avslutningen på min tid som student ved sivilingeniørstudiet i elektronikk, med studieretning multimedia signalbehandling, ved Institutt for Elektronikk og Telekommunikasjon på Norges Teknisk-Naturvitenskapelige Universitet.

Oppgaven tar utgangspunktet i et samarbeidsprosjekt mellom NTNU og Telenor, FONEMA, og arbeidet har blitt utført ved NTNUs campus på Gløshaugen i Trondheim, samt avslutnigsvis i Oslo. Oppgavebeskrivelsen er utarbeidet av professor Torbjørn Svendsen ved Institutt fore Elektronikk og Telekommunikasjon, og han har også vært veileder for arbeidet.

Arbeidet med oppgaven har vært utfordrende, og samtidig lærerikt. Jeg har først og fremst fått økt innsikt i hvilke utfordringer man står ovenfor ved utvikling av systemer fo talesyntese, men også fått oppfrisket og utvidet programmeringsferdighetene mine. Det har vært inspirerende med en oppgave som også har en praktisk side, bevis på de gradvise fremskrittene har vært tilgjenglig i form av stadig nye lydklipp.

En stor takk rettes til professor Torbjørn Svendsen for gode samtaler, nyttige tips underveis og en imøtekommende innstilling til mine utfordringer. En takk må også rettes til stipendiat Dyre Meen for hans hjelp til bruk av Festival med tilleggsfunksjoner.

Til slutt ønsker jeg også å takke min kjæreste, Silje, for all støtte underveis gjennom semesteret. På grunn av sykdom så jeg ikke alltid like lyst på avslutningen av masteroppgaven, men hennes oppmuntrende ord har vært til stor hjelp.

Oslo, 21. juli 2009

Rasmus Rane Bauck

Innhold

Figurer

Tabeller

Forkortelser

LPC	Lineær-prediktiv koding
NTNU	Norges teknisk-naturvitenskapelige universitet
PSOLA	Pitch Synchronous Overlap and Add
TTS	Tekst-til-talesyntese

Kapittel 1

Introduksjon

Taleteknologi er en del av fagområdet språkteknologi, og kan deles inn i to hoveddeler: talegjenkjenning og talesyntese. De to delene kan sees på som inverse operasjoner, en talegjenkjenner søker å ekstrahere informasjon fra menneskelig tale og omdanne dette til skreven tekst eller lignende, mens talesyntese søker å omdanne skreven tekst til naturlig tale. Utvikling av gode systemer for utførelse av disse oppgavene avhenger av kunnskap fra en rekke fagområder, blant annet statistikk, lingvistikk og signalbehandling.

Kommunikasjon mellom mennesker og maskiner ved hjelp av tale har mange fordeler. Tale er en naturlig måte å kommunisere på for mennesker, det stiller ikke krav til opplæring eller spesielle ferdigheter. I tillegg frigjør dette hender og øyne til å utføre andre oppgaver, slik at oppgaver kan løses raskere, tryggere og lettere. For blinde og funksjonshemmede kan bruk av stemmen åpne nye muligheter i hverdagen og øke livskvaliteten. Bruk av taleteknologi kombinert med mobiltelefon gir også mulighet for å finne informasjon uansett hvor og når man måtte ønske denne. Kort sagt er det et stort potensial for applikasjoner som benyttes seg av taleteknologi, men suksessen avhenger av kvaliteten på gjenkjennelsen og syntesen.

Denne masteroppgaven tar utgangspunkt i arbeidet gjort i prosjektet FONEMA [2], et samarbeid mellom NTNU og Telenor om utvikling av verktøy og metoder for høykvalitets norsk talesyntese. FONEMA ble startet opp i 2003 på bakgrunn av erkjennelsen at daværende norske systemer for tekst-til-talesyntese var av for dårlig kvalitet.

Inntil årtusenskiftet var de beste tekst-til-talesyntese basert på ferdig innleste, komplette setninger. Dette er en løsning som gir naturtro tale, men også systemer som er lite fleksible i sitt bruk, da hver eneste ønskede setning må leses inn individuelt. Et stort gjennombrudd kom i 2000/2001 ved lansering av applikasjoner som benytter en ny synteseteknikk, såkalt datadrevet skjøtesyntese. Denne teknikken skjøter sammen deler av de innleste setningene i taledatabasen for å lage de ønskede setningene. Så lenge alle mulige lyder i et språk er representert i taledatabasen, skal det da være mulig å

utføre tekst-til-talesyntese for alle tenkelige setninger med en relativt liten taledatabase.

Hovedmålet for FONEMA var å bidra med forskningsresultater og kunnskap som grunnlag for utvikling av norsk tekst-til-talesyntese basert på datadrevet skjøtesyntese. Arbeidet har resultert i databaser og verktøy som er nødvendige for norsk talesyntese og generering av syntetiske stemmer, men det er fortsatt en vei å gå når det gjelder kvalitet og effektivitet i talesyntesen. Det er videre arbeid på disse to punktene som ønskes utført i denne masteroppgaven.

1.1 Oppgavens oppbygning

I den resternde delen av denne oppgaven vil arbeidet som er gjort denne våren beskrives nærmere. Kapittel 2 gir et innblikk i den teorien som er nødvendig for å forstå problemstillingene og løsningene i denne oppgaven. Det vil også gis en beskrivelse av benyttet programvare. Kapittel 3 omhandler den første hoveddelen av oppgaven, nemlig effektivisering av syntesedatabasen. Arbeidet her omfatter analyse av den eksisterende databasen, samt utvikling av en ny, redusert syntesedatabase med tilsvarende kvalitet. Kapittel 4 omhandler kvalitetsforbedring av den syntetiske stemmen, oppgavens andre hoveddel. Arbeidet her har fokusert på å utvikle metoder for å håndtere syntese av difoner som ikke er representert i den gjeldende syntesedatabasen. Kapittel 5 presenterer resultatene som er oppnådd, samt en diskusjon av disse, mens oppgavens konklusjon kan finnes i kapittel 6.

Kapittel 2

Bakgrunnsteori

Dette kapittelet søker å gi leseren en innføring i den teorien som trengs for å forstå problemstillingene og utfordringene i forbindelse med denne masteroppgaven. Først vil det gis en kort introduksjon til fagfeltet taleteknologi, før talesyntese vil omtales mer detaljert og til slutt introduseres benyttet programvare.

2.1 Taleteknologi

Som nevnt i kapittel 1, deles taleteknologien gjerne inn i to hovedområder, talegjenkjenning og talesyntese, som litt forenklet kan sees som inverse operasjoner. Dette gjør også at de to områdene har mye til felles med tanke på teknologien som ligger i bunn. På lik linje med andre teknologier som er basert på datamaskiner, har taleteknologi opplevd en rivende utvikling i takt med økt prosesseringskraft og større lagringsplass. I tillegg bidrar nye teknikker til at taleteknologien går en spennende fremtid i møte. Bruksmulighetene for teknologien er mange, fra automatiske opplysningstjenester på telefon, via kommunikasjon mellom sjåfør og bil, til ID-verifisering ved inngangsdører.

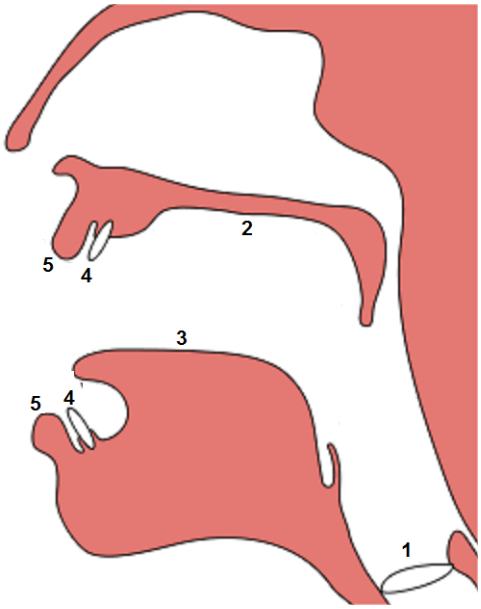
2.1.1 Menneskets taleproduksjon

Mennesker produserer tale ved å presse luft ut gjennom munn og nese. Talen består av en mengde ulike lydenheter, såkalte fonemer[1], satt sammen på forskjellige måter. De forskjellige fonemene skapes ved å endre på formen eller plasseringene av såkalte artikulatorer ved å stramme eller slappe av i ulike muskler i munnen og halsen. Menneskets taleproduksjonsapparat består av følgende artikulatorer:

- Lungene: *Luftkilde.*

[1] Definisjon: minste meningsbærende enhet i tale.

- Stemmebåndene: *Avgjør om lyden blir stemt eller ustemt.*
- Ganen: *Deles inn i hard og myk gane. Den myke ganen fungerer som en ventil, og slipper luft opp til nesegangene og videre ut gjennom nesen. Den harde ganen er en stor, hard flate øverst inne i munnen. Ved å plassere tungen mot den, er det mulig å artikulere konsonanter.*
- Tungen: *En fleksibel artikulator som holdes vekk fra ganen for å artikulere vokallyder eller mot ganen og andre harde flater for å artikulere konsonanter.*
- Tennene: *Tungen spennes mot tennene for å artikulere enkelte konsonanter.*
- Leppene: *Formes runde eller spredet for å artikulere vokaler, mens de lukkes helt for å stoppe luftstrømmen i enkelte konsonanter (p, b, m).*



Figur 2.1: Artikulatorer hos mennesker: stemmebåndene (1), ganen (2), tungen (3), tennene (4) og leppene (5).

Det mest fundamentale skillet mellom lyder i tale går mellom stemte og ustemte lyder. Stemte lyder har et repeterende mønster, både i tids- og frekvensdomenet, og inneholder som regel mer energi enn ustemte lyder. Stemte lyder skapes ved at stemmebåndene vibrerer under fonemartikulasjonen, og det er frekvensen til denne vibrasjonen som også kalles lydens grunnfrekvens, f_0. Vibrasjonene kan variere i frekvens fra 60 Hz for en storvokst mann og opp til 300 Hz for små damer eller barn. Det er også f_0 som

er grunnlaget for de høyere ordens harmoniske svingningene som kan oppstå i hulromene i svelget og munnen.

2.1.2 Fonetikk

Fonetikk defineres som studien av språklyder, samt deres generering og klassifisering. Dette avsnittet gir ingen komplett introduksjon til dette fagfeltet, mens søker å gi en innføring i de ulike lydene i det norske språk og deres klassifisering. Språklyder klassifiseres ofte i de to hovedgruppene vokaler og konsonanter. I tillegg har det norske språk en rekke allofoner med bestemt fonemisk transkripsjon.

Vokaler produseres ved å la luftstrømmen passere tilnærmet fritt gjennom ansatsrøret, men med en rytmisk vibrasjon i stemmebåndet. Frekvensinnholdet til de ulike vokalene avgjøres av de små endringene i utformingen av ansatsrøret. I norsk tale har vi ni forskjellige vokaler, som finnes både i korte og lange varianter, samt syv diftonger. En diftong er en sammensetning av to etterfølgende vokaler, hvor de to vokalene påvirker hverandre. Man klarer da typisk ikke å produsere samme frekvensinnhold for de to delene av diftongen som hvis hver vokallyd uttales for seg selv. Se tabell 2.1 for en oversikt over de ulike vokalene og diftongene.

Konsonanter produseres ved å gjøre betydelige innsnevringer i ansatsrøret, noe som hindrer luften i å flyte fritt. Konsonanter deles gjerne inn i flere undergrupper, basert på hvilke artikulatorer som benyttes for å lage lyden. Plosiver er konsonanter som produseres ved hjelp av blokkering og etterfølgende åpning av ansatsrøret. I blokkeringsfasen bygges trykket opp, før trykket slippes ut når ansatsrøret åpnes igjen. Frikativer er stasjonære lyder, som dannes ved sterk innsnevring, men ikke komplett blokkering, av ansatsrøret. Sonorante konsonanter er en gruppe konsonanter som produseres uten særlige hindring av artikulatorene, noe som gir et vokal-lignende preg over lyden. Se tabell 2.2 for en oversikt over de ulike konsonantene.

Et allofon er en måte å uttale et fonem på. For de vanligste allofonene i det norske språk er det fastlagte fonemiske transkripsjoner. Se tabell 2.3 for en oversikt over de ulike allofonene.

Oversikten over fonemene i det norske språk er hentet fra [8].

Difoner

Et fonem er altså definert som den minste meningsbærende enhet i et språk, det vil si at ved å endre et fonem kan man endre et ords mening. En avart av et fonem er et difon. Et difon er en kombinasjon av to fonemer, nærmere bestemt slutten av et fonem og begynnelsen av et annet fonem. Forskjellen mellom difonsekvens og fonemsekvens for ordet *rapport* er illustrert i figur 2.2.

Det gjøres ellers oppmerksom på at i syntesedatabasen benyttet i denne

Tabell 2.1: Vokaler i norsk tale.

Gruppe	Symbol	Ord	Transkripsjon
Lange vokaler	A:	bak	bA:k
	e:	sen	se:n
	i:	vin	vi:n
	u:	bok	bu:k
	}:	tun	t}:n
	y:	syn	sy:n
	{:	vær	v{:r
	2:	snø	sn2:
	O:	båt	bO:t
Korte vokaler	A	hatt	hAt
	e	bekk	bek
	i	vind	vin
	u	bokk	buk
	}:	hund	h}n
	y:	synd	syn
	{:	vært	v{rt
	2	søtt	s2tt
	O	vått	bOt
Diftonger	{i	vei	v{i
	2y	høy	h2y
	A}	sau	sA}
	Ai	kai	kAi
	Oy	konvoi	kunvOy
	}il	hui	h}i
	ui	hoi	hui

Rapport.
Fonemer: / sil r A p O rt sil /
Difoner: / sil_r r_A A_p p_O O_rt rt_sil /

Figur 2.2: Illustrasjon av forskjellen på fonem og difon.

Tabell 2.2: Konsonanter i norsk tale.

Gruppe	Symbol	Ord	Transkripsjon
Plosiver	p	hopp	hOp
	b	labb	lAb
	t	lat	lA:t
	d	ladd	lAd
	k	takk	tAk
	g	tagg	tAg
Frikativer	f	fin	fi:n
	v	vin	vi:n
	s	lass	lAs
	S	skyt	Sy:t
	C	kino	Ci:nu
	j	gi	ji:
	h	ha	hA:
Sonorante konsonanter	m	lam	lAm
	n	vann	vAn
	N	sang	sAN
	l	fall	fAl
	r	prøv	pr2:v

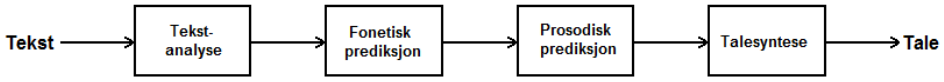
Tabell 2.3: Allofoner i norsk tale.

Symbol	Ord	Transkripsjon
rt	hardt	hArt (retrofleks t)
rd	verdi	v{rdi: (retrofleks d)
rl	ærlig	{:rli (retrofleks l)
rn	garn	gA:rn (retrofleks n)
rL	blå	brLO: (retrofleks "flap")

oppgaven, er difonene merket på en noe annen måte enn forfatteren er vant til fra andre sammenhenger. Dersom et difon er satt sammen av slutten av et */a/*-fonem og starten på et */b/*-fonem, er det normalt å merke difonet som et *a_b*-difon. I den praktiske delen av denne oppgaven er det imidlertid motsatt, dette difonet vil merkes med *b_a*.

2.2 Tekst-til-talesyntese

Prosessen med å omgjøre skreven tekst til uttalte ytringer deles ofte opp i ulike moduler, som vist i figur 2.3. I de etterfølgende delkapitlene vil de ulike stegene omtales nærmere.



Figur 2.3: Modulene i et tekst-til-tale-system.

2.2.1 Tekstanalyse

I denne modulen blir den innkommende teksten analysert, noe som innebærer tekstnormalisering og lingvistisk analyse. Normaliseringen går ut på å omforme blant annet tall og forkortelser til tekst, og utføres ved hjelp av oppslagsverk eller regler for hvordan omformingen gjøres. Ett eksempel på tekstnormalisering er å omforme "kl. 8" til "klokken åtte."

I den lingvistiske analysen (også kalt semantisk[2] og syntaktisk[3] parsing) forsøker systemet å finne informasjon om for eksempel ordklasser, idiomer, plassering av trykk, setningstype og sjanger/stil.

2.2.2 Fonetisk prediskjon

I den fonetiske prediksjonen omdannes den normaliserte teksten til en streng av fonemer. Den kvalitetsmessig beste måten å gjennomføre denne omdannelsen på er ved å benytte en omfattende ordliste, hvor hver enkelt ord kan slås opp. En slik ordliste trenger ikke kreve spesielt stor lagringsplass, men dens omfang begrenses av den tid og innsats man ønsker å legge ned i å utvikle den.

En annen tilnærming til problemstillingen er å utforme et sett med uttaleregler for et språk, noe som ble sett på som hovedkilden til kunnskap om tekst-til-fonem-konvertering i dens tidlige stadier. Dette bygde på erkjennelsen om at ingen ordliste vil kunne inneholde alle mulige ord, all den tid nye

[2]Semantikk: studie av ordenes betydning, både enkeltvis og i sammenstilte strukturer (setninger).

[3]Syntaks: læren om hvordan ord settes sammen til større enheter (setninger og fraser).

ord dukker opp hele tiden. Regler for generell bokstav-til-lyd-konvertering er derfor nødvendig. For å dekke et tilstrekkelig antall ord i et språk, kan det være nødvendig med hundrevis eller tusenvis av slike regler, og det kan ofte være tidkrevende å utlede disse.

I dagens systemer benyttes som regel en kombinasjon av de to tilnærmingene, noe som også er tilfellet for programvaren benyttet i denne oppgaven. Det letes først etter en oppføring i leksikon, og dersom dette ikke finnes benyttes generelle uttaleregler.

2.2.3 Prosodisk prediksjon

Gjennom prosodisk prediksjon gjøres det et forsøk på å videreformidle talerens intensjoner til lytteren. Dette gjøres ved å regulere fonemenes varighet, grunntone og energi, samt ved å legge inn pauser på passende steder. Dette kan være med på å endre trykklegging (aksent) og setningsmelodi (intonasjon), som igjen påvirker budskapet lytteren mottar. Det er altså ikke hva man sier, men hvordan man sier det som behandles i prosodien.

I et eksempel hentet fra [5] ser man tydelig hvordan prosodi kan endre betydningen til en setning. Eksemplet tar utgangspunkt i følgende setning: *"Da hun så at han dro til henne, begynte hun å gråte."* Skriftlig er denne setningen tvetydig, men intonasjonen kan avklare hva taleren ønsker å formidle. Det er ordene *dro til* som er den avgjørende delen av setningen, og forskjellig tonelag her avgjør om lytteren oppfatter at en mann forlater en kvinne til fordel for en annen eller utøver vold mot henne. På samme måte kan forskjellig trykklegging i en setning fremheve ulike deler av setningen og dermed understreke talerens poeng.

2.2.4 Talesyntese

Etter at all preprossesering er gjennomført, gjenstår oppgaven med å omdanne en symbolstreng med fonemisk transkripsjon og tilhørende prosodisk annotering til lydbølger. Det er forskjellige metoder for å gjøre dette, som gjerne deles opp i tre hovedgrupper:

- Aritkulatorsyntese: *benytter en fysisk modell for taleproduksjon, inkluderer artikulatorene omtalt i avsnitt 2.1.1.*
- Formantsyntese: *benytter en kilde-filter-modell, hvor filteret karakteriseres av sakte varierende formantfrekvenser.*
- Skjøtesyntese: *Tale genereres ved å skjøte sammen talesegmenter fra en syntesedatabase.*

I dagens talesyntese er det skjøtesyntese som er den klart dominerende grenen, og det er også denne teknikken som benyttes i arbeidet i denne

masteroppgaven. Aritkulator- og formantsyntese vil derfor ikke beskrives nærmere her, leseren henvises til [9] for mer informasjon om disse teknikkene.

Ved overgangen fra formantsyntese til skjøtesyntese, ble kvaliteten på den syntetiserte talen også markant forbedret. De første forsøkene innenfor skjøtesyntesen fokuserte på teknikken bølegformsyntese, hvor syntesen henter korte lydklipp fra databasen og skjøter disse sammen. Mens formantsyntesen baserte seg på å generere lyder ut fra et sett regler (såkalt "synthesis-by-rule"), benytter altså bølgeformsyntesen original tale og får dermed et langt mer naturlig preg. I databasen med lydklipp finnes det normalt ett eksemplar av hver lydenhet (typisk difoner), og så utfører systemet den nødvendige signalbehandlingen på dette eksemplaret for å få ønsket prosodi i setningen. Teknikker som benyttes i forbindelse med denne modifiseringen er for eksempel PSOLA og lineær prediktiv koding.

Den forbedrete kvaliteten som ble opplevd ved introduksjon av bølgeformsyntesen inspirerte til videre utvikling av syntseteknikken. Den prosodiske modifiseringen som blir utført på hvert difon fører fremdeles til en unaturlig klang i stemmen, så den naturlige utviklingen er da en metode som i enda større grad bruker original tale. Det neste steget er derfor en teknikk kalt datadreven skjøtesyntese, hvor taledatabasen går fra å inneholder ett eksemplar til flere eksemplarer av hver enhet. Ved å ha eksemplarer av en enhet i ulike kontekster er tanken at prosodisk modifikasjon ikke lenger skal være nødvendig. Den resulterende lyden er da sammensatt av uendrete, originale taleklipp, en teknikk som gir mulighet for enda bedre kvalitet på sluttproduktet.

Prisen å betale for den forbedrete kvaliteten i datadreven skjøtesyntese er kravet om en langt større taledatabase. I teorien bør databasen inneholde eksemplarer av alle enheter i alle mulige kontekster. Databasen som er benyttet i denne masteroppgaven inneholder opprinnelig 5.363 setninger, bestående av 230.099 eksemplarer av 1.657 ulike difoner. Hvordan utformingen av en slik database gjøres, er nærmere beskrevet i avsnitt 2.3.

Under selve syntesen, mottar syntesemodulen altås en fonemisk transkripsjon med tilhørende prosodisk annotasjon og skal så plukke ut de enhetene i databasen som til sammen gir best resultat. For å kunne utføre denne oppgaven er man avhengig av å definere en kostnadsfunksjon som både angir hvor godt en enhet i databasen passer i forhold til den ønskede enheten (målkostnad, $d_u(\Theta_j, T)$ i ligning 2.1), samt hvor godt de utvalget enhetene passer sammen (skjøtekostnad, $d_t(\Theta_j, \Theta_{j+1})$ i ligning 2.1).

$$d(\Theta, T) = \sum_{j=1}^{N} d_u(\Theta_j, T) + \sum_{j=1}^{N-1} d_t(\Theta_j, \Theta_{j+1}) \tag{2.1}$$

Verdisettingen av de ulike mål- og skjøtekostnadene kan enten gjøres empirisk eller datadrevet. Empirisk verdisetting gir skjøtekostnad lik 0 for enheter som etterfølger hverandre i taledatabasen, mens øvrige skjøtekostnader

kan defineres ut fra forskjell i f_0 (prosodi) og kunnskap om oppfattet distanse mellom enheter (koartikulasjon). Empirisk målkostnad avhenger av prosodi og kontekst, og også her er det vanlig å velge en prosodisk kostnad proporsjonalt med forskjellen i f_0. En kontekstuell kostnad må baseres på empirisk data om hvor mye kvaliteten forringes ved å benytte enheter fra en annen kontekst.

I en datadrevet kostnadsfunksjon er skjøtekostnaden basert på diskontinuitet i frekvensspekteret i overgangsområdet. Målkostnaden baseres på kontekst, for eksempel kan enheter fra samme kontekst gis målkostnad lik 0, mens enheter fra ulike kontekster gis uendelig målkostnad.

Når kostnadsfunksjonen er definert, finnes det optimale valget av enheter ved å minimere denne. Dette kan gjøres ved hjelp av Viterbi-lignende algoritmer [1]. Effektivisering av søket i databasen kan gjøres ved hjelp av clustering, noe som omtales nærmere i avsnitt 2.3.

2.3 Utvikling av ny syntesedatabase

Arbeidet med å designe en ny database for datadreven skjøtesyntese er en tid- og ressurskrevende prosess, hvor hvert steg er avgjørende for den endelige kvaliteten til den syntetiserte stemmen. I dette avsnittet vil de forskjellige stegene i prosessen gjennomgås.

2.3.1 Valg av enhet

En av de aller første avgjørelsene som må tas i forbindelse med design av en ny syntesedatabase, er hvilken taleenhet som skal benyttes i skjøtesyntesen. Som vist i tabell 2.4, hentet fra [9, s. 805], ser vi at dette valget sterkt påvirker kompleksiteten i databasen. Tabellen inneholder tall for det engelsk tale, men verdier i samme størrelsesorden er gjeldende i norsk tale. Enhetene i tabellen er rangert etter varighet, med den korteste enheten (fonem) øverst.

Tabell 2.4: Oversikt over enheter i det engelske språk.

Enhetstype	Antall enheter
Fonem	42
Difon	1500
Trifon	30.000
Halvstavelse	2000
Stavelse	15.000
Ord	100.000-1.500.000
Frase	∞
Setning	∞

Det er fire hovedønsker som kan etterspøres hos en enhet. Det første er et ønske om lav distorsjon i skjøtene mellom enhetene. Dette kan oppnås ved å velge lange enheter, og dermed få skjøtepunkter, og ved å skjøte enheter på gunstige steder. I tillegg ønskes det lav prosodisk distorsjon, noe som krever enten et større antall eksemplarer i databasen eller muligheter for prosodisk modifikasjon. For stor prosodisk modifikasjon er dog ikke ønskelig, som nevnt i avsnitt 2.2.4.

Det tredje punktet omhandler enhetens generaliserbarhet. I TTS-systemer som tar sikte på å syntetisere all mulig tekst, noe som er tilfelle for FONEMA-prosjektet, er det nødvendig å kunne sette sammen enhetene representert i databasen til ethvert ord, inkludert ukjente ord, som måtte dukke opp under bruk. Dersom ord, fraser eller setninger velges som enhet, er ikke denne generaliserbarheten oppnådd. Generelt er det slik at jo lenger enhet man velger, jo større syntesedatabase er nødvendig for å dekke alle mulige scenarioer som kan dukke opp.

Til slutt er det viktig at enheten er trenbar, det vil si at kun en begrenset mengde treningsdata er nødvendig for å kunne estimere alle enhetene. Som tabell 2.4 viser, vokser antallet av en enhet med lengden på enheten, og krever dermed enorme treningsdatabaser.

På grunn av ønsket om generaliserbarhet og trenbarhet, begrenser valgmulighetene for enhet i FONEMA-prosjektet seg til fonemer og difoner. Valget har falt på difoner fordi man da er sikret naturlige overganger mellom fonemer [4]. Valg av lengre enheter er først og fremst relevant for systemer som har begrenset operasjonsområde, for eksempel opplysningstjenester for bussruter. Man vil da kun ha behov for et forholdsvis lite antall setninger i syntesedatabasen.

2.3.2 Manuskript

Utformingen av et manuskript for taledatabasen er kanskje det viktigste steget i utviklingen av et TTS-system, da databasens fonetiske og prosodiske innhold sterkt påvirker den endelige kvaliteten på den syntetiserte stemmen.

Ideelt sett kunne en slik taledatabase inneholde flere varianter av alle fonetiske og prosodiske varianter som ønskes tilgjengelig i syntesen, slik at en det alltid finnes en perfekt enhet i databasen. Dette er imidlertid lite gunstig med tanke på systemets ressursbruk. En slik database vil kreve mye minne og mange beregninger underveis for å finne de rette enhetene. Løsningen blir å utforme et manuskript som gir best mulig fonetisk og prosodisk dekning på en så effektiv måte som mulig.

Utviklingen av et slikt kompakt manuskript kan gjøres på flere måter, men en ofte brukt metode er som følger. Først samles det sammen et tekstskorpus av langt større størrelse enn ønsket for det endelige manuskriptet. Setningene i dette korpuset bearbeides så ved hjelp av TTS-systemets front-

end"[4]. Deretter benyttes en grådighetsalgoritme som iterativt velger ut setninger fra korpuset til manuskriptet ved å alltid inkludere den av de ikke allerede valgte setningene som tilfører manuskriptet mest ny informasjon. Den første setningen som velges vil da typisk være en lang setning bestående av et stort antall ulike difoner, mens den neste setningen som velges vil ha et svært annerledes innhold. Denne grådighetsalgoritmen kjøres inntil en forhåndsbestemt antall setninger er valgt ut eller ønsket antall difoner er representert i databasen. I tillegg til å kun se på difonene i setningene som velges ut, finnes det andre varianter av denne algoritmen som også tar hensyn til akustiske likheter og ulikheter mellom enheter [3].

2.3.3 Prosess

Selv om valg av enhet og utforming av manuskript er viktige punkter, gjenstår det fortsatt en del arbeid før syntesedatabasen er klar til bruk. Det neste steget er å lese inn manuskriptet og lagre dette i digitale lydfiler. Personen som leser inn manuskriptet bør naturlig nok ha en klar og tydelig stemme, slik at den syntetiserte talen har mulighet for å fremstå best mulig.

De gjenstående stegene er her beskrevet slik de utføres for syntesedatabasen som er benyttet i denne oppgaven. Dette kan avvike noe for andre databaser, men i bunn og grunn er det den samme jobbem som gjøres. Skriptene som kjøres er vist i listen under:

- Pmark.pl: *Prosessering av taleklipp, inkluderer blant annet 18. ordens LPC-analyse, høyoppløselig f_0-estimering og beregning av LPC-rest.*
- AlignNSTVoice.py: *Laster inn ytringer (manuskript og lydfiler), bygger ytringer og foretar autosegmentering*[5] *av lydfiler.*
- CreateNST.pl: *Benytter .utt-, .pm- og .f0-filer som er laget av AlignNSTVoice.py, samt de innleste .wav-filene, konstruerer .mcep-filer (inneholder kepstralegenskaper*[6]*). Bygger så stemmen ved hjelp av Festival og parametere definert i build_clunits.scm.*

I tillegg til det som er nevnt i punktene over, utføres det også en klynging av difonene i databasen. Klynging innebærer å dele opp databasen i en rekker undergrupper (klynger), hvor hver klynge inneholder difoner som ligner på hverandre. Slik oppsettet har vært i arbeidet med denne masteroppgaven, er lydklippene først blitt delt opp i en sekvens av difoner. Ut fra denne oppdelingen er det utarbeidet en "fasit" for hvert difon, det vil si hvilke egenskaper et difon av denne typen bør ha. Ut fra denne fasiten, samt den valgte målkostnadsfunksjonen, klynges de (opptil) 40 beste utgavene av

[4] Inkluderer de tre første modulene vist i figur 2.3

[5] Fonemisk transkripsjon av lydfil, med bestemmelse av fonem- og difongrenser

[6] Et kepstrum er Fourier-transformen til logaritmen til et signals spektrum.

hvert difon sammen. Under talesyntesen slipper så systemet å søke gjennom hele syntesedatabasen, men trenger kun å beregne skjøte- og målkostnader for 40 valgmuligheter.

2.4 Programvare

I denne masteroppgaven er det i hovedsak Festival Speech Synthesis System som har blitt benyttet som programvare. Denne presenteres i lenger ned i teksten. I tillegg har arbeidet blitt utført ved hjelp av en lang rekke skript. Disse skriptene har vært skrevet i programmeringsspråkene Python[7], Matlab[8] og UNIX[9].

2.4.1 Festival Speech Synthesis System

Festival Speech Synthesis System [7] er et rammeverk for systemutvikling i forbindelse med talesyntese. Rammeverket er tilgjengelig både via kommandolinje, et Scheme-grensesnitt, C++- og Java-bibliotek, samt et Emacs-grensesnitt. Fra programvarens nettsider kan programmet lastes ned med syntesestemmer for engelsk, walisisk og spansk, mens FestVox[10] gir deg de nødvendige hjelpemidler for å utvikle nye stemmer.

Grunnlaget for Festival er det såkalte Edinburgh Speech Tools Library [6]. Dette er et bibliotek bestående av ulike C++-klasser, -funksjoner og tilhørende programmer som trengs for å utføre taleteknologisk signalbehandling[11].

I denne masteroppgaven er det Festival-programvaren som sørger for tekst-til-talesyntesen. Etter at syntesedatbasen er generert er det Festival som tar inn tekststreng, søker igjennom databasene og gjør oppslag i de nødvendige egenskapsfilene. Ved hjelp av en tilleggsfunksjon, beskrevet nærmere i tillegg A, er storskala analyse av syntesedatabasen med hensyn på å effektivisere denne. Dette arbeidet er nærmere beskrevet i kapittel 3. I arbeidet med å forbedre kvaliteten til den syntetiske stemmen, har resultatene fra analysen av syntesedatabasen blitt brukt videre. Festival brukes her igjen til å utføre talesyntese, etter at et egenutviklet Python-skript har kontrollert ytringen.

2.4.2 Festivals ytringsstruktur

Sentralt i all dataprosesseringen som utføres i forbindelse med talesyntesen i Festival, er en ytringsstruktur. Det er i denne strukturen at all den

[7] http://www.python.org/
[8] http://www.mathworks.com/
[9] http://www.unix.org/
[10] http://festvox.org/
[11] For eksempel merking av databaser eller egenskapsuttrekning fra lydfiler.

nødvendige informasjonen ligger lagret. Hvilke kategorier informasjon som ligger lagret i en slik ytringsstruktur kan endres av brukerne for å tilpasses ønsket bruk. Beskrivelsen her er basert på den typen ytringsstruktur som er benyttet i forbindelse med denne masteroppgaven.

For å lage en ny ytring via Festivals kommandolinje, blir følgende kommando benyttet:

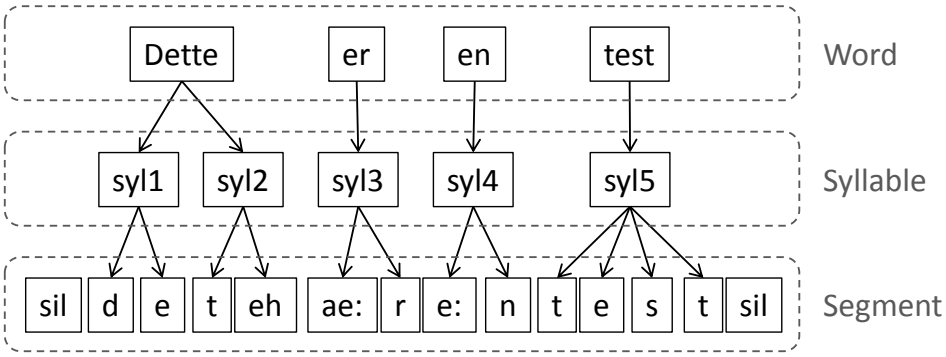
```
festival> (set! utt (fonema_makeutt "Dette er en test."))
```

Gjennom funksjonen *fonema_makeutt*, fylles også resten av ytringsstrukturen ut umiddelbart. Informasjonen i ytringsstrukturen er organisert i forskjellige relasjoner. Hver relasjon inneholder data om ytringen på forskjellige nivåer. I tillegg er det koblinger mellom de ulike relasjonsnivåene, slik at det er mulig å se hvilke data på hvert nivå som hører sammen. Relasjonene som er tilgjengelige i en ytring ved hjelp av *fonema_makeutt* er vist i tabell 2.5.

Tabell 2.5: Relasjoner i en Festival-ytring.

Relasjon	**Beskrivelse**
Word	Liste av ord, et element for hvert ord i ytringen.
Syllable	Liste av stavelser, et element for hver stavelse i ytringen.
Segment	Liste av fonemer (setningens fonemiske transkripsjon).
Target	Relaterer Segment-elementene til ønskede f_0-verdier.
Foot	Angir hvilke stavelser som skal trykklegges.
IntEvent	Liste over intonasjonshendelser.
Intonation	Liste med trær som knytter elementene i IntEvent- (løvnoder) og Syllable-relasjonene sammen (røtter).
SylStructure	Liste med trær som knytter sammen elementene i Word- (røtter), Syllable- og Segment-relasjonene (løvnoder). Se figur 2.4.
Phrase	Liste med trær, hvor ord (løvnoder) kobles til fraser (røtter)

Av særlig interesse for arbeidet utført i kapittel 4 er SylStructure-relasjonen. Ved hjelp av informasjonen som er funnet her, er det eksempelvis mulig å finne ut om to etterfølgende fonemer tilhører det samme ordet. Innholdet i SylStructure-relasjonen er illustrert i figur 2.4.



Figur 2.4: Illustrasjon av SylStrucutre-relasjonen i en ytringsstruktur.

Kapittel 3

Effektivisering av syntesedatabase

Dette kapittel omhandler arbeidet som ble gjort i forbindelse med den ene av oppgavens to hoveddeler, nemlig effektivisering av den genererte syntesedatabasen. FONEMA-prosjektet baserer sin talesyntese på teknikken datadrevet skjøtesyntese, en teknikk som krever en stor database med innlest tale. En stor andel av det nødvendige minnet som programvaren trenger er knyttet til lydfilene som inneholer denne talen, men det antas at mange av disse filene er overflødige i forhold til kvaliteten på den resulterende talesyntesen.

Avsnitt 3.1 omhandler analysen av den allerede eksisterende syntesedatabasen, mens avsnitt 3.2 presenterer de nye syntesedatabasene som ble designet. Testingen av de ulike syntesedatabasene og deres tilhørende stemmer beskrives i avsnitt 3.3.

3.1 Analyse av eksisterende syntesedatabase

En effektivisering av den genererte syntesedatabasen vil i klarttekst gå ut på å redusere kravene til lagrings- og beregningskapasitet for applikasjonen. Det er derfor nødvendig å kartlegge hvordan den eksisterende syntesedatabasen benyttes.

Den opprinnelige syntetiske stemmen benytter seg av en syntesedatabase bestående av 5.363 setninger hentet fra ulike kilder, blant annet nyhetssendinger. Setningene er tilgjengelige både som tekststrenger og innlest tale, og under bygging av stemmen genereres en rekke filer som inneholder informasjon om deres fonemiske transkripsjon og andre egenskaper. For en mer detaljert beskrivelse av hvordan en stemme bygges, se avsnitt 2.3.

3.1.1 Testprosedyre

Det ble besluttet å studere bruken av de 5.363 setningene i syntesedatabasen under talesyntese i storskala. Formålet er å kartlegge hvilke ytringer som brukes sjelden eller aldri, og dermed finne kandidater som kan fjernes fra databasen.

Det arbeides dermed på ytringsnivå, det vil si hele setninger, men en annen mulighet kunne vært å se på hvilke difoner fra de ulike setningene som brukes sjelden. Dette ble ikke gjort, da det ville gitt et langt mer komplisert arbeid med å luke ut de delene av syntesedatabasen man ønsket å fjerne. I stedet for å fjerne hele lydfiler og setninger, ville man da vært nødt til å fjerne deler av lydfiler og setninger og deretter "lime sammen" de gjenværende delene.

For å få en tilstrekkelig test av den eksisterende syntesedatabasen, vil det naturlig nok være nødvendig å syntetisere tale fra et stort antall setninger. Disse setningene ble hentet fra manuskriptet "fondat3_pruned_cands". bestående av setninger som ble "til overs" da manuskriptet for den eksisterende syntesedatabasen ble valgt ut. Det er til sammen 165.000 setninger i denne filen, men testingen ble besluttet utført ved hjelp av tilfeldige subsett av denne databasen. Til de 15 gjennomkjøringene av testprosedyren ble følgende tekstmanus generert:

- Fem manus à 10.000 setninger.
- 10 manus à 1.000 setninger.

Den automatiske kjøringen av testen, samt prosessering av testresultatene, gjennomføres ved hjelp av ulike Python- og Unix-script, samt tilleggsfunksjoner for Festival-programmet:

- manus_generator.py: *Lager manus av ønsket størrelse til talesyntese. Plukker ut tilfeldige setninger fra "fondat3 pruned cands".*
- fonema_manuscript.scm: *Definerer tilleggsfunksjonen fonema_say_manuscript for Festival. Henter setning for setning fra det angitte manuset og lagrer resultatene fra talesyntesen i ønsket målmappe. Resultatene lagres i en egen fil for hver setning som syntetiseres. Resultatene består av informasjon om hvilke difoner som benyttes og hvilke lydfiler i syntesedatabasen disse difonene hentes fra.*
- summarizer: *Et Unix-skript som går gjennom resultatfilene fra talesyntesen og teller hvor mange ganger de ulike filene benyttes.*
- find_unused_utt.py: *Lager en oversikt over ubrukte ytringer, basert på resultatfilene produsert av summarizer.*

- find_utt_used_count.py: *Formaterer resultatfilene produsert av summarizer slik at resultatene kan importeres til Microsoft Excel og fremstilles grafisk.*

Alle disse filene ble innlevert som digitale vedlegg sammen med denne masteroppgaven.

3.1.2 Analyseresultater

Resultatene fra analysen kan sees i figurer 3.1, 3.2 og 3.3, samt i tabeller 3.1 og 3.2.

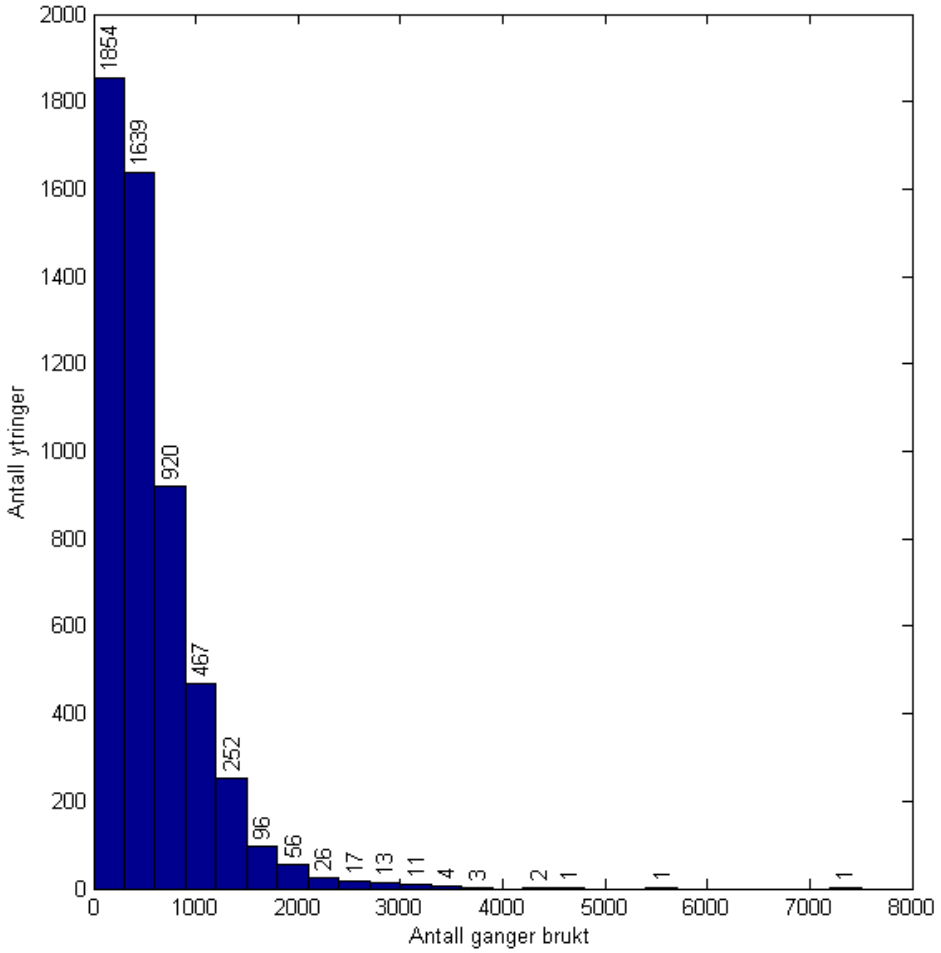
Tabell 3.1 gir en oversikt over talesyntesens kvalitet vurdert ut fra hvor stor andel av setningene som faktisk blir syntetisert. Tallene varierer litt for de forskjellige testkjøringene med ulike manus, men i snitt syntetiseres 97.9% av setningene. Problemer oppstår når programmet ønsker å syntetisere difoner som ikke er representert i syntesedatabasen. Dette skjer hovedsaklig i forbindelse med utenlandske ord som dukker opp i manus.

Tabell 3.1: Fordeling av syntetiserte og ikke syntetiserte setninger i analysemanus.

Manus	**Syntetisert**	**Ikke syntetisert**	**Andel syntetisert**
manus_1000_0	972	28	97.2%
manus_1000_1	979	21	97.9%
manus_1000_2	984	16	98.4%
manus_1000_3	985	15	98.5%
manus_1000_4	981	19	98.1%
manus_1000_5	983	17	98.3%
manus_1000_6	983	17	98.3%
manus_1000_7	981	19	98.1%
manus_1000_8	976	24	97.6%
manus_1000_9	981	19	98.1%
manus_10000_1	9.787	213	97.9%
manus_10000_2	9.780	220	97.8%
manus_10000_3	9.793	207	97.9%
manus_10000_4	9.763	237	97.6%
manus_10000_5	9.812	188	98.1%
Totalt	58.740	1.260	97.9%

Som figur 3.1, 3.2 og 3.3, samt tabell 3.2, tydelig viser, er det en klar skjevhet i bruken av de forskjellige ytringene. Nesten 2/3 av de 5.363 ytringsfilene benyttes mindre enn 600 ganger i forbindelse med syntetiseringen av

totalt 58.740 setninger. Den meste brukte ytringen benyttes 7.364 ganger, men kun 23 ytringer benyttes mer enn 3.000 ganger.

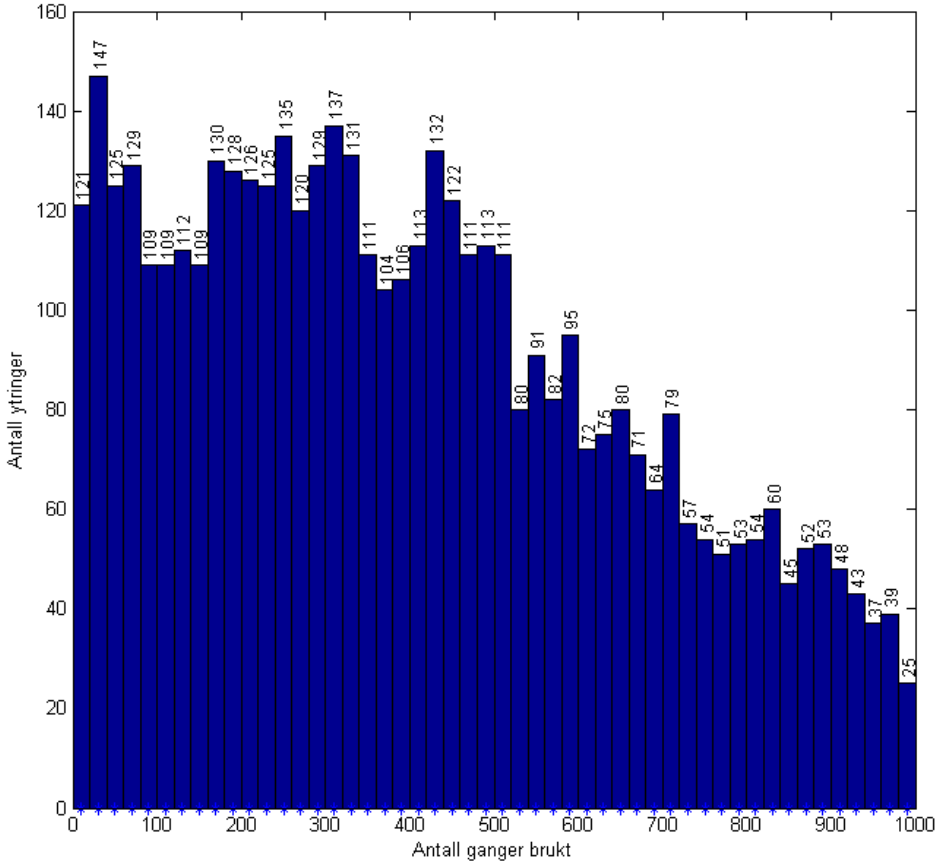


Figur 3.1: Histogram over bruk av alle ytringsfiler i taledatabasen (intervaller med bredde 300).

Detaljert oversikt over testresultatene er å finne blant rapportens digitale vedlegg.

3.2 Nye syntesedatabaser

Med utgangspunkt i analyseresultatene presentert i avsnitt 3.1.2, er to nye syntesedatabaser blitt utformet. De er begge forminskede varianter av den opprinnelige syntesedatabasen, heretter omtalt som voice_1, og vil presenteres nærmere her.



Figur 3.2: Detaljert histogram over bruk for de minst brukte ytringsfilene i taledatabasen (intervaller med bredde 20).

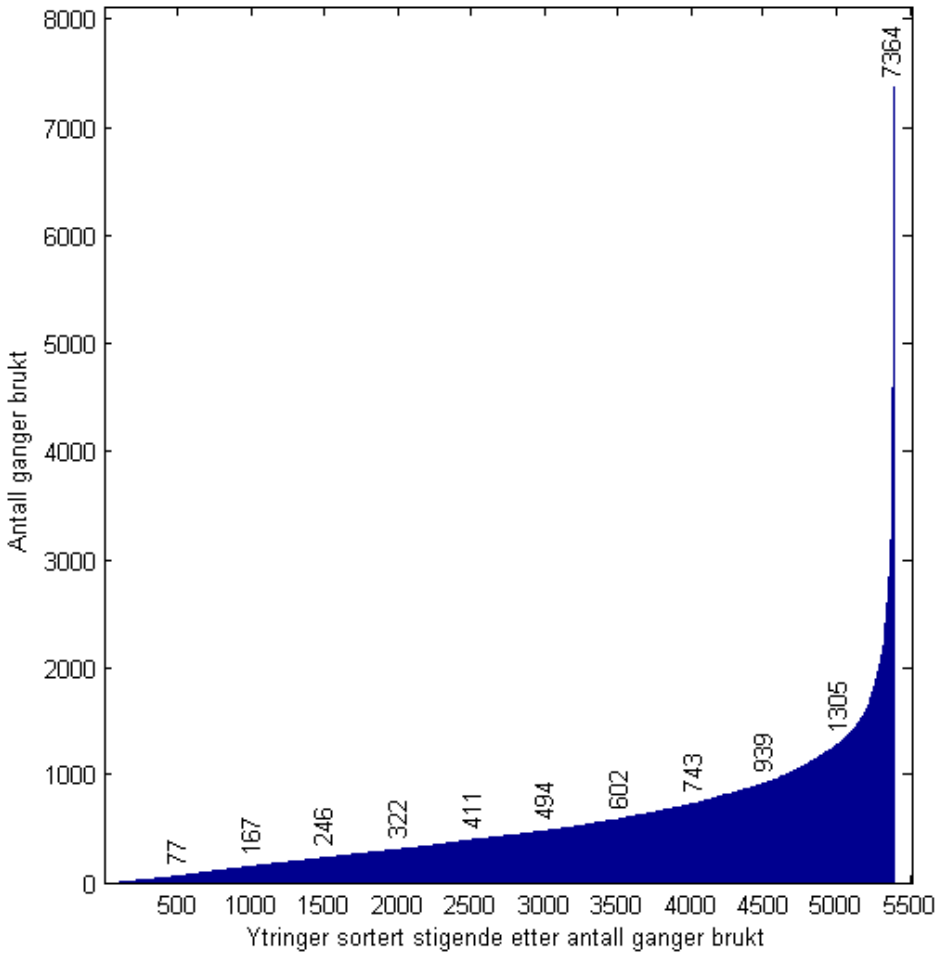
3.2.1 voice_2

På bakgrunn av resultatene presentert i kapittel 3.1.2, ble de 10% minst brukte setningene i manuskriptet fjernet når denne syntesedatabasen, heretter referert til som voice_2, skulle lages. Setningene ble fjernet uten å se på deres innhold, det ble for eksempel ikke tatt hensyn til om en setning inneholdt sjeldne difoner. Det gjenværende manuskriptet består dermed av 4.827 setninger, altså er 536 setninger klippet bort.

En komplett liste over hvilke ytringer som er fjernet i syntesedatabase voice_2 i forhold til voice_1 er inkludert som digitalt vedlegg ved innlevering av masteroppgaven.

3.2.2 voice_3

Den tredje stemmen, heretter referert til som voice_3, er også basert på 90% av syntesedatabasen til den opprinnelige stemmen, voice_1. Endringene i



Figur 3.3: Ytringsfiler sortert etter antall ganger brukt.

forhold til voice_2 er gjort med tanke på å dekke flest mulig difoner i den gjenværende databasen.

For å oppnå dette, er ni setninger i voice_2s database byttet ut med ni setninger fra den komplette databasen som voice_1 benytter. Disse ni setningene som har kommet inn inneholder samtlige tilfeller av de fire difonene som forsvant i voice_2 sammenlignet med voice_1 (*sil_ae:*, *d_Z_*, *e:_y* og *i:_Ai*), i tillegg til to tilfeller av difonet *n_aeu* (kun ett tilfelle av dette difonet i voice_2s database). Setningene som måtte vike plass er de ni minst brukte setningene fra analysen presentert i avsnitt 3.1 som ble beholdt for voice_2, men som kun inneholder vanlige difoner (difoner representert ved minst 20 tilfeller i databasen). Det er da rimelig å anta at kvaliteten på talesyntesen ikke bør forrringes nevneverdig ved å fjerne disse setningene.

En komplett liste over hvilke ytringer som er fjernet og tatt med i syn-

Tabell 3.2: Ytringer gruppert med hensyn på brukshyppighet

Intervall	**Antall ytringer**	**Intervall**	**Antall ytringer**
0-300	1854	3901-4200	0
301-600	1639	4201-4500	2
601-900	920	4501-4800	1
901-1200	467	4801-5100	0
1201-1500	252	5101-5400	0
1501-1800	96	5401-5700	1
1801-2100	56	5701-6000	0
2101-2400	26	6001-6300	0
2401-2700	17	6301-6600	0
2701-3000	13	6601-6900	0
3001-3300	11	6901-7200	0
3301-3600	4	7201-7500	1
3601-3900	3		

tesedatabase voice_3 i forhold til voice_2 er inkludert som digitalt vedlegg ved innlevering av masteroppgaven.

3.3 Testprosedyre

For å teste de ulike syntesedatabasene, vil det utføres både objektive og subjektive tester, som beskrevet i de etterfølgende avsnittene.

3.3.1 Objektiv test

For å teste de ulike syntesedatabasene objektivt, vil det samme manuskriptet med 10.000 setninger bli forsøkt syntetisert ved hjelp av hver database. I tillegg vil databasenes difondekning analyseres, det vil si å finne ut hvor mange ulike difoner som er representert og hvor mange difoneksemplarer som finnes totalt i databasen.

3.3.2 Subjektiv test

Et manuskript på 100 setninger syntetiseres av alle tre databasene, og lydklippene lagres. Disse vil så lyttes på med hensikt å detektere eventuelt hørbare forskjeller i ytelse.

3.3.3 Kommentarer til testprosedyrene

Disse testprosedyrene er forholdsvis simple, i og med at de ulike databasene vil vurderes kun ut fra tre tallverdier og en lytters synspunkter. For å få et

mer vitenskaplig fundament for resultatene, bør den subjektive lyttetesten utføres av flere personer og den objektive testen bør være mer omfattende enn den nåværende prosedyren. Likevel vil disse testene gi en pekepinn på hvor godt de forskjellige syntesedatabasene fungerer.

Kapittel 4

Forbedring av kvalitet for syntetisk stemme

I tillegg til arbeidet med å effektivisere ressursbruken i syntesedatabasen, som beskrevet i kapittel 3, er det ønskelig å forbedre kvaliteten på talen som produseres. Dette ønskes gjennomført uten å øke den opprinnelige ressursbruken nevneverdig.

Den allerede eksisterende stemmen gir stort sett akseptabel eller bedre talekvalitet, i det minste når setningene som skal syntetiseres består av velkjente ord. Ved sammensatte og ukjente ord kan talen være vanskelig å forstå på grunn av feil trykklegging, mens nye fremmedord kan gi feil fonemisk transkripsjon.

Det er flere måter å forsøke å forbedre talesyntesens kvalitet på. Av mulige tilnærminger kan nevnes justering av kostnadsfunksjon i enhetsutvelgelsen og forbedring av algoritmer for automatisk enhetssegmentering av de ulike lydklippene. I denne masteroppgaven har imidlertid fokus vært rettet mot håndtering av tilfeller hvor den fonemiske transkripsjonen inneholder difoner som ikke er representert i syntesedatabasen.

4.1 Utgangspunkt

Under arbeidet med effektiviseringen av syntesedatabasen, som beskrevet i kapittel 3, ble det avdekket at talesyntesen gjentatte ganger ble avbrutt grunnet difoner som ikke er representert i syntesedatabasen. Som tabell 5.1 viser, skjer dette kun i 2 % av setningene som forsøkes syntetisert, men det har allikevel mye å si for hvilken kvalitet brukeren oppfatter at TTS-systemet har.

Ideelt sett skal et TTS-system kunne syntetisere en hvilken som helst setning brukeren angir. Den eksisterende løsningen med å ikke syntetisere noe av setningen fordi et difon mangler, er i så måte en veldig dårlig håndtering av situasjonen. Brukeren vil da ikke motta noe av informasjonen i setningen.

Analysen av den eksisterende stemmen avdekker at det i all hovedsak er fremmedord med lyder som sjelden forekommer i det norske språk som er opphavet til difonene som ikke finnes i syntesedatabasen. Tabell 4.1 gir en oversikt over de 13 difonene som oftest er årsaken til at syntesen avbrytes. Dataene er basert på loggføring av 576 setninger som ikke lot seg syntetitsere.

Tabell 4.1: Ofte forekomne difoner som ikke kan syntetiseres.

Difon	**Antall**	**Andel**	**Eksempel**
m_ae	36	6.25%	stedsnavnet Birmingh***am***.
Ai_n	28	4.86%	det engelske ordet "u***ni***ted".
l_ae	27	4.69%	det engelske ordet r***al***ly".
eh_Ch	24	4.17%	egennavnet ***Che***ney.
r__O:	22	3.82%	stedsnavnet New Y***or***k.
t_ae	20	3.47%	det engelske ordet "m***at***ch".
ou_e	16	2.78%	det norske ordet petrol***eu***m".
j_rl	15	2.60%	adressen Ka***rl*** ***J***ohans gate.
A:_Oy	14	2.43%	det engelske ordet r***oya***l".
r__f	14	2.43%	sykkelrittet Tour de ***Fr***ance.
eI_l	12	2.08%	det britiske politiske partiet ***La***bour.
r__k	10	1.74%	egennavnet ***Cr***aig.
u_u	10	1.74%	egennavnet Eastw***oo***d.

4.2 Løsning

Under arbeidet med masteroppgaven er det blitt utviklet et Python-skript, med navn tts_bauck.py, som tar sikte på å håndtere situasjoner hvor den fonemiske transkripsjonen inneholder difoner som ikke er tilgjengelig i syntesedatabasen. Kildekoden til skriptet er å finne i tillegg B.

Skriptet søker å endre den fonemiske transkripsjonen gjennom metoder beskrevet i avsnitt 4.2.1. Avsnitt 4.2.2 gir en gjennomgang av hvordan skriptet går frem for å endre den fonemiske transkripsjonen til en setning.

4.2.1 Metoder

Som tabell 4.1 viser, er det i overgangen mellom et vanlig[1] fonem og et uvanlig fonem at Festival har problemer med å finne et passende difon i sin syntesedatabase. Det er i utgangspunktet tre forskjellige måter å håndtere et slikt problem på:

[1] Ofte forekommende i norsk tale.

- Hoppe over: *Ekskludere det uvanlige fonemet fra setningens fonemiske transkripsjon.*
- Innsetting: *Sette inn et annet fonem før eller etter det uvanlige fonemet, slik at de resulterende difonene finnes i syntesedatabasen.*
- Substitusjon: *Bytte ut det uvanlige difonet med et lignende, vanlig difon, slik at de resulterende difonene finnes i syntesedatabasen.*

De tre forskjellige metodene har alle sine fordeler og ulemper. Å hoppe over det uvanlige difonet er naturligvis den enkleste løsningen, men dette har sjelden et heldig resultat. Et fonem fjernes fra setningen, noe som fører til at ord endres og hele setningen kan få ny mening. Dette er derfor en metode som bør brukes som en siste utvei, de andre metodene må forsøkes først.

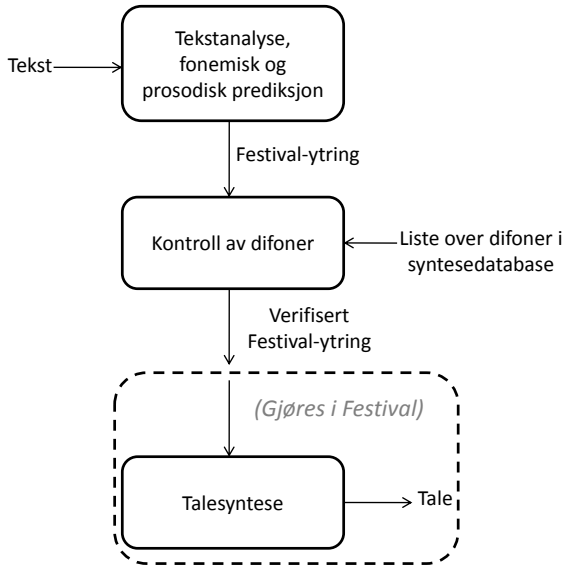
Innsetting av et ekstra fonem i den fonemiske transkripsjonen er en løsning som kan gi gode resultater. Slik metoden er implementert her, forsøkes det å sette inn stillhet[2] mellom de to fonemene som til sammen utgjør difonet som ikke finnes i syntesedatabasen. Ved å sette inn stillhet, skapes det naturlig nok en pause i uttalen av setningen. Av denne grunn fungerer metoden best dersom difonet som ikke finnes i syntesedatabasen er i en ordovergang, det vil si at difonets to fonemer stammer fra hvert sitt ord. Metoden har den fordelen at den originale fonemiske transkripsjonen beholdes, med unntak av det innsatte /sil/-fonemet, men andre metoder bør benyttes når difonet ikke opptrer i en ordovergang.

Substitusjon av fonem er den metoden som er mest komplisert, men kan samtidig gi den beste ytelsen dersom det gjøres på en god måte. Dersom man tar utgangspunkt i at den originale fonemiske transkripsjonen er korrekt og at enhetene i den benyttede syntesedatabasen er merket riktig, vil en substitusjon av et eller flere fonem nødvendigvis føre til at man ikke oppnår optimal kvalitet i den syntetiserte setningene. Den endrete fonemiske transkripsjonen vil nødvendigvis ha et annet innhold, men med gode substitusjoner kan endringene ha liten eller ingen merkbar påvirkning på den syntetiserte setningen. Utfordringen for en slik metode ligger altså i å lage gode substitusjonsregler som ikke endrer den fonemiske transkripsjonen unødvendig mye.

4.2.2 Virkemåte

I avsnitt 4.2.1 ble de tre grunnleggende metodene som skriptet benytter beskrevet. I dette avsnittet vil skriptets fremgangsmåte beskrives nærmere, det vil si hvordan skriptet går frem for å konvertere en tekststreng til en komplett Festival-ytring med gyldig fonemisk transkripsjon.

[2] fonemet /sil/



Figur 4.1: Blokkdiagram for tts_bauck.py

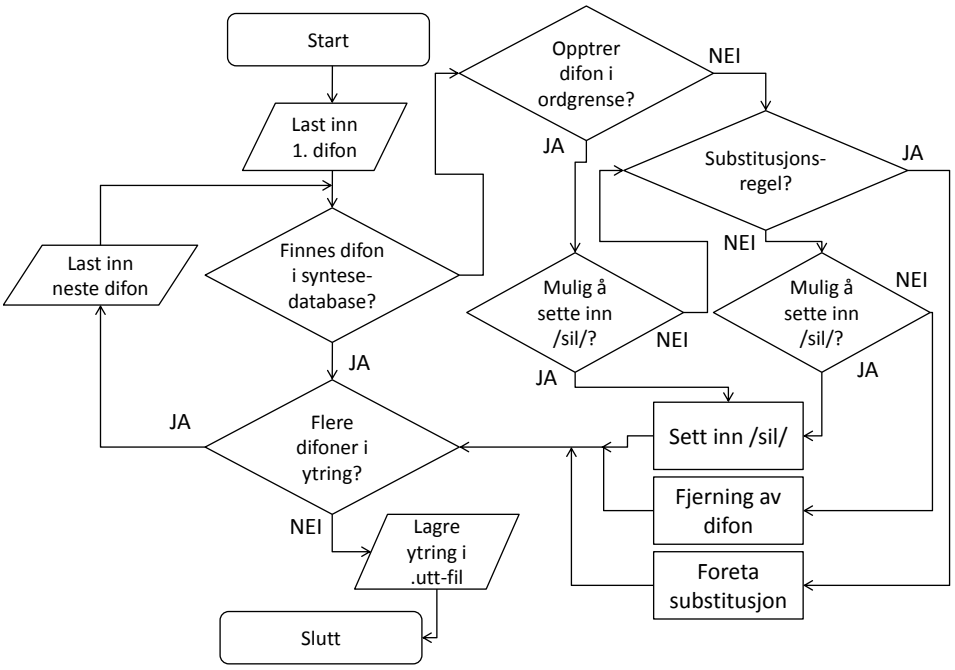
Den overordnede fremgangsmåten til skriptet tts_bauck.py er vist i figur 4.1. Tekstanalyse, fonemisk og prosodisk prediksjon utføres ved hjelp av PyEST, en Python-modul for Festival utviklet av Dyre Meen, stipendiat ved Institutt for eletronikk og telekommunikasjon på NTNU. Dataene er nå lagret i en Festival-ytringsstruktur, som beskrevet i avsnitt 2.4.2. Denne forprosesseringen er identisk med arbeidet som opprinnelig gjøres i Festival når man lager en ny ytring. Det er i den påfølgende delen av blokkskjemaet, "Kontroll av difoner", at endringen i forhold til det opprinnelige TTS-systemet i Festival gjøres. Dette omtales mer detaljert under. Ut fra denne modulen kommer det en .utt-fil som må lastes inn i Festival manuelt[3] og selve talesyntesen gjøres med Festivals Wave_Synth-funksjon.

Kontrollen av difonene i den fonemiske transkripsjonen til ytringen utføres med virkemåte som vist i flytdiagrammet i figur 4.2. Python-skriptet starter med det første difonet i ytringen, og itererer seg steg for steg gjennom samtlige difoner.

En liste over syntesedatabasens difoner lastes inn fra en tekstfil, og for hvert difon i ytringen gjøres det først en sjekk om det aktuelle difonet er representert i databasens liste. Dersom difonet er i databasen, er det ikke nødvendig å gjøre noe spesielt, og man kan gå videre til neste difon eller avslutte kontrollen av difoner.

Hvis difonet ikke finnes i syntesedatabasen, er man derimot nødt til å gjøre noen justeringer. Første kontrollspørsmål går på om difonet finnes i en ordgrense. Dersom dette er tilfellet legges det inn et /sil/-fonem, gitt at

[3]Ved hjelp av kommandolinje-grensesnitt eller lignende.



Figur 4.2: Flytskjema for kontroll av difoner i tts_bauck.py

de nye difonene som oppstår er representert i syntesedatabasen[4], og skriptet går videre til neste fonem (eller avslutter hvis ingen flere fonemer gjenstår).

Dersom difonet ikke opptrer i en ordgrense, eller det ikke var mulig å sette inn en et /sil/-fonem i ordgrensen, forsøker skriptet så å gjøre en substitusjon. Skriptet ser først etter om det finnes en spesiell substitusjons- eller omskrivningsregel (se henholdsvis tabell 4.2 og tabell 4.3) for akkurat dette difonet, før det eventuelt ser etter en generell substitusjonsregel (se tabell4.4). Dersom det ikke finnes noen substitusjonsregel, gjøres det et forsøk på å sette inn et /sil/-fonem. Hvis dette heller ikke lar seg gjøre, forsøker skriptet å fjerne det fonemet som skaper problemet.

Begrunnelsen for å benytte de ulike metodene i denne rekkefølgen er basert på et ønske om å endre minst mulig på de opprinnelige ordlydene. Det er bedre å la lytteren høre de ønskede fonemene, med noen innlagte pauser mellom ordene, enn å endre unødigvendig mye. Det er allikevel mindre sjenerende å endre på et fonem i et ord enn å sette inn pauser eller fjerne fonemer midt i ordet.

[4]Eksempel: /sil/ settes inn etter /b/ og før /m/ kun hvis difonene /b sil/ og /sil m/ er i syntesedatabasen.

Tabell 4.2: Spesielle substitusjonsregler i tts_bauck.py.

Opprinnelig difon	**Nytt difon**
rL__ae	rL__ae:
ae_Z_	e_Z_
Ai_n	aei_n
eh_Ch	e_Ch
ou_e	ou_eh
j_rl	j_rL_
b_rl	b_rL_
h_rL_	h_l
Sh_rl	s_rl
A:_Oy	A:_ou
Ng_oe	n_oe
e:_e	e_e
eh_eh	e_eh
O_oey	O:_oey
ou_Ch	ou:_Ch
u:_y	O:_y
m_h (hvis i ordet "Ahmed")	m_k
oe_ou (hvis i ordet "forfulgt")[5]	ou_f
Oy_l	oey_l
Oy_k	oey_k
Oy_m	oey_m
Oy_n	oey_n
Oy_b	oey_b
u_Oy	O:_oey
t_Oy	t_oey
b_Oy	b_oey
h_Oy	h_oey
n_Oy	n_oey
i_y	i_y:
rL__ou:	rl_ou:
t_j	t_i
Ch_Ai	Sh_aei
eI_Z_	eh_Z_
eh_h	e_h
r_Ai (hvis i ordet "Air*")	r_e
u:_u	u:_u:
u_rd	u:_rd
r__ae	r_ae
eh_O	e_O

Tabell 4.3: Omskrivningsregler i tts_bauck.py.

Opprinnelig fonem	**Nytt fonem**
/Z_ i/	/Z_ i: i/
/@U/	/O: u/
/Ai/	/A i/
/rt/	/r t/
/rd/	/r d/
/u u/	/u/
/Ai ae:/	/A j aei/
/Z sil/	/Z eh sil/

Tabell 4.4: Generelle substitusjonsregler i tts_bauck.py.

Opprinnelig difon	**Nytt difon**
th_* (engelsk th-lyd)	t_*
*_ae	*_aei
ae_*	aei_*
eI_*	aei_*
w_*	v_*
r__* (engelsk r-lyd)	r_* (norsk r-lyd)
*_r_ (engelsk r-lyd)	*_r (norsk r-lyd)
aU_*	aeu_*
D__*	d_*
rl_*	l_*
*_rL_ (hvis ikke i ordgrense)	*_l

Kapittel 5

Resultater

Dette kapittelet presenterer resultatene av arbeidet som ble utført i masteroppgaven. Avsnitt 5.1 omhandler effektivisering av syntesedatabasen, mens avsnitt 5.2 omhandler arbeidet med å forbedre kvaliteten på den syntetiserte stemmen. Til slutt vil de oppnådde resultatene diskuteres i avsnitt 5.3

5.1 Effektivisering av syntesedatabasen

I denne delen presenteres resultatene som ble oppnådd under testingen av den opprinnelige og de to nyutviklede syntesedatabasene. Først presenteres de objektive resultatene, det vil si resultater som kan tallfestes nøyaktig. Til slutt presenteres de subjektive resultatene, som er basert på lyttetester. Lyttetestene er kun utført av én person og i et forholdsvis lite omfang, og må mer anses som stikkprøver enn endelige konklusjoner.

5.1.1 Objektive resultater

Resultatene fra den objektive testingen av de tre syntesedatabasene er vist i tabell 5.1.

Tabell 5.1: Testresultater, objektiv test.

Kategori	**voice_1**	**voice_2**	**voice_3**
Setninger i syntesedatabase	5.363	4.827	4.827
Antall difontyper	1.657	1.656	1.658
Totalt antall difoner	230.099	220.349	220.258
Syntetiserte setninger (av 10.000)	9.804	9.803	9.801

Fire difoner fra voice_1 er ikke representert i voice_2, nemlig d_Z_, e:_y, i:_Ai og sil_ae:. Tre difoner fra voice_2 er ikke representert i voice_1, nemlig ae:_aeu, rt_sil og u:_y:.

Ett difon fra voice_1 er ikke representert i voice_3, nemlig m_b. To difoner fra voice_3 er ikke representert i voice_1, nemlig ae:_aeu og u:_y:.

To difoner fra voice_2 er ikke representert i voice_3, nemlig m_b og rt_sil. Fire difoner fra voice_3 er ikke representert i voice_2, nemlig nemlig d_Z_, e:_y, i:_Ai og sil_ae:.

Av de setningene som ikke kan syntetiseres, er det 195 setninger som ingen av de tre syntesedatabasene klarer å syntetisere. Disse setningene inneholder hovedsaklig fremmedord eller uvanlige egennavn, som krever sjeldne difoner som ikke er representert i databasene. Dette er samme fenomen som funnet i forhåndsanalysen av den originale syntesedatabasen og nærmere beskrevet i avsnitt 3.1.2.

I tillegg til dette er det totalt syv setninger som ikke kan syntetiseres av én av de tre syntesedatabasene. Dette er setninger som inneholder et difon som en av de tre databasene mangler sammenlignet med de to andre. En oversikt over disse syv setningene vises i tabell 5.2.

Tabell 5.2: Ikke syntetiserbare setninger i kun én av tre databaser.

Database	**Setning**	**Difon**
voice_1	"Alternativet er å følge Kyoto-avtalen og kvotesystemet her."	u:_y:
voice_2	"Siden 2000 er ni mennesker drept av hai i australske farvann." "Han har forbud mot å oppsøke advokat Elden fram til mai i år."	i:_Ai
voice_3	"Mange irakere misliker også at utlendinger får jobb mens mange irakere går uten arbeid." "Men folk flest syns ikke det er greit å gå på jobb med en øl og tre akevitt innabords." "Jeg bor på Vestlandet, og jobbmulighetene er ikke like gode som på Østlandet." "Eller er det de sosialhjelpsmottakerne som er i jobb men har for liten lønn til å klare seg?"	m_b

5.1.2 Subjektive resultater

Resultatet fra den subjektive testingen er 97 lydklipp for hver av de tre syntesedatabase. Tre setninger lot seg ikke syntetisere av noen av de tre databasene. Alle lydklippene er innlevert som digitale vedlegg sammen med denne masteroppgaven.

Lytting på disse lydklippene viser at det generelt sett ikke er noen hørbar forskjell i kvalitet mellom de ulike stemmene. voice_2 og voice_3 velger i stor

grad nøyaktig de samme enhetene fra sine respektive databaser, og deres lydklipp blir naturlig nok veldig like. voice_1 har en større syntesedatabase å plukke fra og velger også andre enheter av og til, men kvaliteten er altså den samme som oppnådd med voice_2 og voice_3.

Blant de 97 lydklippene er det to tilfeller hvor det er markant forskjell i kvalitet, og begge gangene går dette i disfavør av voice_1. I det ene tilfellet slukes e-lyden i slutten av ordet "kvinne' av voice_1, mens voice_2 og voice_3 har en langt mer markant e-lyd. Alle tre syntesedatabaser gjør identisk fonemisk transkripsjon av den gjeldende setningen under preprosessering av tekststrengen. Det er altså et "uheldig"valg av enhet fra syntesedatabasen som utgjør forskjellen. Det aktuelle lydklippet er vedlagt med filnavn *s3140167.wav*.

Det andre tilfellet gjelder uttalen av egennavnet Bush, som gjøres korrekt av voice_2 og voice_3, mens voice_1 er uheldig i sitt valg av enhet for å representere difonet Sh_ou. I stedet for en *sh*-lyd, høres det ut mer ut som en overgang fra *s* til *e*. Som for det forrige tilfellet har alle tre syntesedatabasene den samme fonemiske transkripsjonen som grunnlag for sin skjøtesyntese. Det aktuelle lydklippet er vedlagt med filnavn *s3174873.wav*.

5.2 Forbedring av kvalitet for syntetisk stemme

I denne delen presenteres resultatene som ble oppnådd under arbeidet med å forbedre kvaliteten til den syntetiske stemmen.

5.2.1 Objektive resultater

I tabell 5.3 presenteres resultatene fra forsøket på å syntetisere et manuskript bestående av 10.000 setninger. Dette manuskriptet er det samme som ble benyttet for objektiv testing i forbindelse med effektivisering av syntesedatabasene, se avsnitt 5.1.1.

Tabell 5.3: Testresultater, objektiv test.

Endring	**Antall**
Uendrete setninger	9.803
Innsatt /sil/ mellom ord	12
Substitusjoner/omskrivninger	211
Fjernet fonem	6
Setninger som ikke kan syntetiseres	0

9.803 setninger ble syntetisert uten at det var nødvendig å gjøre noen endringer i den fonemiske transkripsjonen. 12 ganger var det nødvendig å

sette inn et /sil/-fonem i en ordgrense, mens 211 omskrivninger eller substitusjoner ble utført. 6 ganger var det nødvendig å fjerne et fonem. Totalt 229 endringer ble utført på 197 setninger. Samtlige 10.000 setninger er mulige å syntetisere etter at difonene ble kontrollert.

5.2.2 Subjektive resultater

De ulike substitusjons- og omskrivningsreglene har blitt forsøkt kvalitetsmessig vurdert ut fra stikkprøver underveis i arbeidet med å utvikle reglene. Reglene har blitt klassifisert i tre grupper etter hvor bra de fungerer:

- God: Ingen eller lite merkbar forskjell fra sånn som setningen bør høres ut.
- OK: Noe forskjell fra sånn som setningen bør høres ut, men lett å forstå hva som sies.
- Dårlig: Klar forskjell fra sånn som setningen bør høres ut, vanskelig å forstå hva som sies.

Dette er selvfølgelig ingen inndeling med vitenskapelig nøyaktighet, men det gir en pekepinn på hvor godt reglene fungerer.

Som vi ser av tabellene 5.4, 5.5 og 5.6, er 24 substitusjonsregler vurdert som gode, 20 substitusjonsregler er vurdert som OK, mens 14 substitusjonsregler er klassifisert som dårlige. Det kan også tillegges at metoden med å sette inn /sil/-fonem i ordgrenser fungerer godt, mens den siste utveien med å fjerne fonemer fungerer dårlig.

En gjennomgående trend blant substitusjonsreglene som er vurdert som gode, er at det fonemet som skaper problemer i forhold til syntesedatabasen, har et lignende, vanlig fonem som man kan endre til. Et eksempel er å endre et fonem fra /eh/ til /e/ eller /e:/. Satt inn mellom andre fonemer er det ikke mulig å høre at det har blitt gjort noen endringer på den fonemiske transkripsjonen.

Blant substitusjonsreglene som er vurdert som dårlige, er trenden at problemet stammer fra fonemer som er markant forskjellige fra de øvrige fonemene. Det er derfor vanskelig å finne gode substitutter i syntesedatabasen.

Eksempler på sluttresultat ved bruk av disse substitusjons- og omskrivningsreglene er innlevert som digitale vedlegg sammen med denne masteroppgaven.

5.3 Diskusjon

Dette avsnittet tar sikte på å diskutere de resultatene som er oppnådd gjennom arbeidet utført i denne masteroppgaven.

Tabell 5.4: Gode substitusjons- og omskrivningsregler i tts_bauck.py.

Vurdering	Opprinnelig difon	Nytt difon
God	eh_Ch	e_Ch
	ou_e	ou_eh
	Sh_rl	s_rl
	e:_e	e_e
	eh_eh	e_eh
	O_oey	O:_oey
	u:_y	O:_y
	m_h (hvis i ordet "Ahmed")	m_k
	oe_ou (hvis i ordet "forfulgt")[1]	ou_f
	i_y	i_y:
	rL__ou:	rl_ou:
	eh_h	e_h
	r_Ai (hvis i ordet "Air*")	r_e
	/Z_ i/	/Z_ i: i/
	/@U/	/O: u/
	/Ai/	/A i/
	/rd/	/r d/
	/u u/	/u/
	u:_u	u:_u:
	*_ae	*_aei
	ae_*	aei_*
	eI_*	aei_*
	rl_*	l_*
	*_rL_ (hvis ikke i ordgrense)	*_l

5.3.1 Effektivisering av syntesedatabase

Arbeidet med å effektivisere syntesedatabasen har resultert i to nye syntesedatabaser, begge med en reduksjon av antall lydklipp på 10 % i forhold til den originale syntesedatabasen. Under den subjektive testingen av den syntetiserte talen fra disse databasene har det ikke blitt avdekket noen merkbar endring av kvaliteten. Den opprinnelige mistanken om at syntesedatabasen inneholdt mange lydklipp som hadde lite å si for selve talesyntesen må derfor sies å ha blitt bekreftet.

De to nye syntesedatabasene, voice_2 og voice_3, er designet på noe ulike måter. Mens voice_2 kun inneholder de 90 % mest brukte lydklippene fra storskala talesyntese, er det gjort enkelte endringer i voice_3 i et forsøk på å sikre optimal difondekning i syntesedatabasen. Dette er imidlertid ingen eksakt vitenskap, da identifiseringen av difoner i lydklipp under trening av en syntesedatabase avhenger av de difoner som allerede er identifisert. Lydklipp

Tabell 5.5: OK substitusjons- og omskrivningsregler i tts_bauck.py.

Vurdering	**Opprinnelig difon**	**Nytt difon**
OK	rL__ae	rL__ae:
	ae_Z_	e_Z_
	j_rl	j_rL_
	b_rl	b_rL_
	h_rL_	h_l
	A:_Oy	A:_ou
	Ng_oe	n_oe
	ou_Ch	ou:_Ch
	t_j	t_i
	Ch_Ai	Sh_aei
	u_rd	u:_rd
	r__ae	r_ae
	eh_O	e_O
	/rt/	/r t/
	th_* (engelsk th-lyd)	t_*
	w_*	v_*
	r__* (engelsk r-lyd)	r_*
	*_r_ (engelsk r-lyd)	*_r
	aU_*	aeu_*
	D__*	d_*

y kan få en annen fonemisk transkripsjon dersom ikke lydklipp x lenger har blitt transkribert på forhånd. Dette eksemplifiseres ved det faktum at nye difoner dukker opp i syntesedatabasene voice_2 og voice_3, men ikke i voice_1. For å sikre en mer deterministisk difondekning i databasen, kan det derfor være en idé å være mer bevisst på å fjerne lydklipp som kun inneholder ofte forekomne difoner[2]. Det er rimelig å anta at dette vil ha mindre påvirkning på difonidentifisering enn å fjerne lydklipp med sjeldne forekomne difoner.

Selv om 10 % reduksjon i antall lydklipp i syntesedatabasen er en markant endring, er det fristende å tenke at det kan være mulig å redusere antall lydklipp ytterligere. Det er 42 difoner som er representert mer enn 1.000 ganger i den opprinnelige syntesedatabasen, og ytterligere 71 difoner er representert over 500 ganger. Ved å gå databasen litt nærmere i sømmene er det trolig mulig å finne setninger som ikke tilfører databasen noe særlig informasjon, og dermed kunne fjerne også disse. Resultatene oppnådd i denne masteroppgaven bør i alle fall kunne motivere forsøk på ytterligere kutt i databasen.

[2]For eksempel n_eh og r_eh.

Tabell 5.6: Dårlige substitusjons- og omskrivningsregler i tts_bauck.py.

Vurdering	Opprinnelig difon	Nytt difon
Dårlig	Ai_n	aei_n
	Oy_l	oey_l
	Oy_k	oey_k
	Oy_m	oey_m
	Oy_n	oey_n
	Oy_b	oey_b
	u_Oy	O:_oey
	t_Oy	t_oey
	b_Oy	b_oey
	h_Oy	h_oey
	n_Oy	n_oey
	eI_Z_	eh_Z_
	/Ai ae:/	/A j aei/
	/Z sil/	/Z eh sil/

5.3.2 Forbedring av kvalitet for syntetisert stemme

Under arbeidet med å forbedre kvaliteten for den syntetiserte stemmen har fokuset vært på å håndtere situasjoner hvor syntesedatabasen mangler difonene som den fonemiske transkripsjonen ønsker. Som vi ser av resultatene presentert i avsnitt 5.2, tar det utviklede Python-skriptet seg av alle de utfordringene de ble stilt overfor i løpet av testen.

44 av de 58 substitusjons- og omskrivningsreglene fungerer godt eller OK. Selv om 14 av de 58 substitusjons- og omskrivningsreglene, samt nødløsningen med å fjerne et fonem hvis ikke noe annet er mulig, ikke gir optimal kvalitet, gir også disse bidrag til å øke kvaliteten på talesyntesen. Det er en langt bedre løsning at resten av setningen syntetiseres godt, mens det manglende difonet høres merkelig ut, enn at ingen deler av setningen lar seg syntetisere.

Utviklingen av substitusjons- og omskrivningsregler er gjort ved hjelp av mer eller mindre kvalifisert prøving og feiling. De gode resultatene som er oppnådd med forholdsvis enkle grep kan virke som motivasjon til å forsøke å utvikle disse reglene ytterligere. Det siktes da først og fremst til de reglene som ble vurdert å fungere dårlige, se tabell 5.6.

Kapittel 6

Konklusjon

Denne masteroppgaven har tatt utgangspunkt i arbeidet som ble utført i FONEMA-prosjektet med å etablere verktøy og metoder for høykvalitets norsk talesyntese. Målsetningen med vårens arbeid har vært å effektivisere den eksisterende syntesedatabasen og å forbedre kvaliteten på den syntetiserte stemmen.

Ubegrenset talesyntese utført ved hjelp av skjøtesyntese, med difoner som grunnenhet, krever en database med lydklipp som kan settes sammen for å lage syntetisert tale. En slik database må nødvendigvis være omfattende, på den måten at den inneholder alle difoner som er nødvendig for å kunne uttale ethvert ord. Begrensninger med tanke på lagringsplass og søkekostnader gjør imidlertid at design av en syntesedatabase ikke er en triviell sak. En syntesedatabase bør derfor være så kompakt som mulig: flest mulig difoner ønskes så godt representert som mulig i en database av en gitt størrelse.

Arbeidet med å effektivisere databasen har tatt utgangspunkt i en antakelse om at den eksisterende databasen inneholder en rekke lydklipp som ikke er nødvendige. I de 5.363 lydklippene som benyttes er en rekke difoner unødvendig ofte representert, talesyntesen vil kunne høres like god ut selv med en mindre syntesedatabase. Ved hjelp av storskala talesyntese, ble det kartlagt i hvor stor grad de ulike lydklippene ble benyttet og ut fra disse resultatene ble nye syntesedatabaser designet. Effektivisering har resultert i to nye syntesedatabaser (voice_2 og voice_3) som hver inneholder 4.827 lydklipp, en størrelsesreduksjon på 10 % i forhold til den opprinnelige syntesedatabasen (voice_1).

Gjennom objektiv testing har det blitt klarlagt at voice_2 og voice_3 har beholdt ytelsen til voice_1. Et manuskript bestående av 10.000 setninger har blitt forsøkt syntetisert av alle tre syntesedatabaser. Mens voice_1 klarte å syntetisere 9.804 av setningene, klarte voice_2 og voice_3 å syntetisere henholdsvis 9.803 og 9.801 setninger. Når det kommer til antall unike difoner i de ulike representert, er resultatene igjen like. 1.657 unike difoner er rep-

resentert i voice_1, 1.656 i voice_2 og 1.658 i voice_3. Subjektive lyttetester er med på å underbygge de objektive resultatene, det er sjelden mulig å høre forskjell på talen produsert av de ulike syntesedatabasene. De to eneste gangene hvor det er markant forskjell, er det faktisk voice_2 og voice_3 som gir bedre kvalitet enn voice_1.

Arbeidet med å forbedre kvaliteten på den syntetisete stemmen har konsentrert seg om å håndtere situasjoner hvor den fonemiske transkripsjonen inneholder difoner som ikke er representert i den benyttede syntesedatabasen. Under storskala talesyntese utført i forbindelse med effektivisering av syntesedatabasen kom det frem at 2 % av setningene som ble forsøkt syntetisert krevde difoner som ikke var tilgjengelige for TTS-systemet. Systemets måte å takle dette på var å nekte syntetisering av noe som helst av setningen. Ønsket var å utvikle verktøy for å håndtere dette problemet.

Gjennom nevnte storskala talesyntese ble det samlet inn informasjon om hvilke difoner som skapte disse problemene, og ut fra dette utviklet et Python-skript med et sett med substitusjons- og omskrivningsregler. Reglene forsøker å bytte ut hele eller deler av et difon som ikke eksisterer i syntesedatabasen slik at det resulterende difonet både eksisterer i syntesedatabasen og høres tilnærmet likt ut som det opprinnelige difonet.

Ytelsen til substitusjons- og omskrivningsreglene har blitt forsøkt målt ved hjelp av både objektive og subjektive tester. Det er nå mulig for TTS-systemet å syntetisere samtlige 10.000 setninger i manuskriptet som også ble brukt for å teste de effektiviserte syntesedatabasene. 9.803 setninger lar seg syntetisere uten endringer, mens det er nødvendig å sette inn et /sil/-fonem ved 12 anledninger. 211 substitusjoner- og omskrivninger utføres, mens seks fonemer må fjernes for å kunne syntetisere setningene. Lyttetester er brukt for å vurdere substitusjons- og omskrivingsreglene. 24 av reglene klassifiseres som gode, 20 regler klassifiseres som OK, mens 14 regler klassifiseres som dårlige. Det fungerer i tilegg godt å sette inn /sil/-fonem i ordgrenser, mens praksisen med å fjerne fonemer gir dårlig kvalitet.

På bakgrunn av arbeidet som er beskrevet over, kan det konkluderes med at målene som var satt for oppgaven har blitt nådd. Syntesedatabasens størrelse har blitt redusert med 10 % uten merkbar endring i kvalitet på den syntetiserte talen. Kvaliteten på den syntetiserte talen har blitt bedret gjennom det faktum at flere setninger nå lar seg syntetisere ved å gjøre små endringer i den fonemiske transkripsjonen.

6.1 Videre arbeid

- Et intuitivt videre steg i arbeidet med å effektivisere syntesedatabasen er å forsøke mer drastiske kutt. Motivasjonen for dette er en kombinasjon av resultatene oppnådd i denne masteroppgavens arbeid, samt det faktum at mange av difonene er overrepresentert i syntesedatabasen.

- En annen mulighet er å øke difondekningen i syntesedatabasen. Dette gjelder først og fremst difoner fra fremmedord, spesielt engelsk. Eksempler på slike difoner er *t_ae* (som i m*at*ch), samt engelske *th*-, *r*- og *w*-lyder. Dette er imidlertid en tid- og ressurskrevende jobb, da dette krever manuskript og innlesing.

- Mange av substitusjons- og omskrivningsreglene fungerer godt, men ikke alle er perfekte. Det kan være av interesse å gjøre forsøk på utbedre disse reglene ytterligere.

Bibliografi

[1] G.D. Forney Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[2] NTNU, Institutt for elektronikk og telekommunikasjon. FONEMA. http://www.iet.ntnu.no/projects/fonema/.

[3] Alan W Black og Kevin A. Lenzo. Optimal data selection for unit selection synthesis. In *4rd ESCA Workshop on Speech Synthesis*, pages 63–67, 2001.

[4] Ingunn Amdal og Torbjørn Svendsen. State-of-the-art for datadrevet skjøtesyntese, 2004.

[5] Lena Kildal Skaalbones. Samspillet mellom intonasjon og syntaks. http://www.ntnu.no/eksternweb/multimedia/archive/00030/Skaalbones07_30145a.pdf.

[6] The University of Edinburgh, Center for Speech Technology Research. Edinburgh speech tools library. http://www.cstr.ed.ac.uk/projects/speech_tools/.

[7] The University of Edinburgh, Center for Speech Technology Research. Festival speech synthesis system. http://www.cstr.ed.ac.uk/projects/festival/.

[8] University College London, Division of Psychology & Language Sciences. SAMPA for Norwegian. http://www.phon.ucl.ac.uk/home/sampa/norweg.htm.

[9] Hsiao-Wuen Hon Xuedong Huang, Alex Acero. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development.* Prentice Hall, 2001.

Tillegg A

Tilleggsfunksjon til Festival Speech Synthesis System

For å forenkle storskala testing av syntesedatabaser, ble en tilleggsfunksjon til Festival Speech Synthesis System. Tilleggsfunksjonen er i hovedsak av utviklet av Dyre Meen, stipendiat ved Institutt for eletronikk og telekommunikasjon på NTNU, men er modifisert noe for å tilfredsstille behovene til denne masteroppgaven.

Tilleggsfunksjonen er skrevet i programmeringsspråket Scheme[1], en dialekt av Lisp[2]. Kildekoden lagres i en fil med navn fonema_manuscript.scm og ligger vedlagt under.

Listing A.1: fonema_manuscript.scm

```
1  (define (fonema_speak_manuscript tgt_dir src_fn)
2  "(fonema_speak_manuscript tgt_dir src_fn)
3  Synthesize and store multiple sentences from a given manuscript
4  "
5  (let ((p (load src_fn t)))
6    (mapcar
7     (lambda (l)
8       (let ((u (fonema_makeutt (cadr l))))
9       ; Funksjonene under opererer på en ytring, u.
10      (print (cadr l))
11      (utt.relation.print u "Word")
12      ; Går fra fonemisk transkripsjon til bølgeform
13      (unwind-protect
14        (Wave_Synth u)
15        (begin
16          (format t (string-append "Error in " (car l) "\n"))))
17      ; Lagrer informasjon om benyttede enheter i tekstfil
18      (unwind-protect
19        (utt.save.relation u "Unit" (string-append tgt_dir "/
              unit_" (car l) ".txt"))
```

[1] http://www.schemers.org/

[2] http://lisp.org/alu/home

```
20          (begin
21            (format t (string-append "Error in " (car l) "\n"))
22            (utt.save.words u (string-append tgt_dir "/error_unit_"
                  (car l) ".txt"))))
23        ; Lagrer lyd som "<tgt_dir>/fnm_<ID>.wav".
24        ; Kommentert vekk når endelig lydfil ikke var nødvendig.
25        (unwind-protect
26          (utt.save.wave u (string-append tgt_dir "/" (car l) ".wav
                "))
27          (begin
28            (format t (string-append "Error in " (car l) "\n"))))
29        )
30       t)
31     p)
32   t)
33 )
34
35 ; Gjør pakken fonema_manuscript tilgjengelig for Festival Speech
       Synthesis System
36 (provide 'fonema_manuscript)
```

For å få tilgang til funksjonene i Scheme-filen må følgende linje legges til i filen .festivalrc på brukerens hjemmeområde:

Listing A.2: Utdrag fra .festivalrc

```
1 (require "<sti_til_rett_mappe>/fonema_manuscript")
```

Tillegg B

Kildekode for tts_bauck.py

Under vises kildekoden for Python-skriptet tts_bauck.py. For å kjøre skriptet kreves Python 2.4 (eller nyere), med tilleggspakken fonema[1], samt PyEST[2] installert.

Skriptet kan kjøres på følgende måter:

Listing B.1: Eksempel på bruk av tts_bauck.py

```
$ python tts_bauck.py '<Setning som skal syntetiseres>'
$ python tts_bauck.py '<Navn på manuskriptfil med setninger som
    skal syntetiseres>'
```

Den første varianten syntetiserer en enkelt setning, mens variant nummer 2 syntetiseres et helt manuskript med setninger. Manuskriptet må være formatert på samme måte som manuskripter som skal leses inn i Festival.

Listing B.2: tts_bauck.py

```
# -*- coding: cp1252 -*-
import string, sys
from os import path
from os.path import dirname, join
from est import *
from est.item import *
from fonema.tts.make_utts import *
from fonema.teksfon import Teksfon, sent2utt
from fonema import teksfon

# Define which text-to-diphone rules should be used
lexdir = "/usr/local/src/FonemaCVS/Lexica/teksfon/data"
teksfon.user_lex = path.join(lexdir,"lu.crp")
teksfon.main_lex = path.join(lexdir,"lm.crp")
teksfon.name_lex = path.join(lexdir,"ln.crp")
teksfon.pron_rls = path.join(lexdir,"pr.crp")

```

[1] Tilleggspakke med TTS-funksjoner for Python.

[2] En Python-wrapper for Edinburgh Speech Tools.

```
18 utt_score = [0, 0, 0, 0, 0] # [uendret, /sil/, substitusjon,
       fjerning, ingen triks]
19
20 def synthezise(sentence, utt_id):
21     t = Teksfon()
22     s = t.proc_string(sentence)
23     u = sent2utt(s)
24     u_orig = sent2utt(s)
25
26     u2 = EST_Utterance()
27
28     # Loading file with all the voice's diphones and put into a
           list
29     diphone_file = open('diphones_voice_1.txt', 'r')
30     diphones_list = []
31     for line in diphone_file.readlines():
32         diphones_list.append(str(line)[:-2])
33
34     relations = ["Word", "Syllable", "Target", "Foot", "
           Intonation", "IntEvent", "SylStructure", "Phrase"]
35
36     # Moving all other relations than Segment from u to u2
37     for rel in relations:
38         u2_rel = u2.create_relation(rel)
39         for item in u[rel]:
40             u2_rel.append(item)
41
42     # Creating some objects to use under utt manipulation
43     previous = None
44     item_orig = None
45     word_orig = None
46     previous_orig = None
47     append_next = True
48
49     silence = False
50     remove = False
51     substitution = False
52     unable = False
53
54     sil_score = 0
55     sub_score = 0
56     remove_score = 0
57     unable_score = 0
58
59     # Time to look at the diphones...
60     u2_seg = u2.create_relation("Segment")
61     for item in u["Segment"]:
62
63         if item_orig != None:
64             item_orig = next(item_orig)
65             word_orig = parent(parent(item_orig, "SylStructure")
                   , "SylStructure")
66             print get_name(item_orig), get_feature(word_orig, "
                   name"),
```

```
67          else:
68              item_orig = u_orig["Segment"].head()
69              word_orig = parent(parent(item_orig, "SylStructure")
                    , "SylStructure")
70              print get_name(item_orig), get_feature(word_orig, "
                    name"),
71
72          diph = str(get_feature(item, "name")) + '_' + str(
                get_feature(previous, "name"))
73          print diph,
74
75          if diph not in diphones_list and diph != 'sil_0':
76              # sil_0 is a "dummy" phonem at the start of all
                    sentences
77              # sil_0 is not included in the speech synthesis
78
79              if diph == "sil_sil": append_next = False
80
81              if str(get_feature(word_orig, "name")) != str(
                    get_feature(parent(parent(previous_orig, "
                    SylStructure"), "SylStructure"), "name")) and'
                    sil_'+str(get_feature(previous, "name")) in
                    diphones_list and str(get_feature(item, "name"))+
                    '_sil' in diphones_list:
82                  sil = EST_Item()
83                  sil["name"] = 'sil'
84                  sil_end = item["end"]
85                  sil_parent = parent(item, "SylStructure")
86                  item["end"] = float(str(float(sil_end) - 0.01)
                        [:5])
87                  sil["end"] = sil_end
88                  previous = u2_seg.append(sil)
89                  append_daughter(sil_parent, "SylStructure", sil)
90                  silence = True
91              elif diph == 'rL__ae': previous["name"] = 'ae:'
92
93              # As in Jackson. Sounds a bit weird
94              elif diph == 'ae_Z_': item["name"] = 'e'
95
96              elif diph == 'i_Z_': # As in Beijing. MAJOR HACK !!
97                  i_diph = EST_Item()
98                  i_diph["name"] = 'i:'
99                  i_diph_end = item["end"]
100                 i_diph_parent = parent(item, "SylStructure")
101                 previous["end"] = float(str(float(i_diph_end) -
                        0.01)[:5])
102                 i_diph["end"] = i_diph_end
103                 previous = u2_seg.append(i_diph)
104                 append_daughter(i_diph_parent, "SylStructure",
                        i_diph)
105
106                 if word_orig["name"] == 'Beijing':
107                     # Har igjen /i Ng/, ønsker /i Ng g/
108                     # Legger til nytt fonem /i/
```

```
109                     print 'Trouble',
110                     i__diph = EST_Item()
111                     i__diph["name"] = 'i'
112                     i__diph_end = item["end"]
113                     i__diph_parent = parent(item, "SylStructure"
                            )
114                     previous["end"] = float(str(float(
                            i__diph_end) - 0.01)[:5])
115                     i__diph["end"] = i__diph_end
116                     previous = u2_seg.append(i__diph)
117                     append_daughter(i_diph_parent, "SylStructure
                            ", i_diph)
118
119                     # Endrer navn i resterende difoner, fra /i
                            Ng/ til /Ng g/
120                     item["name"] = 'Ng'
121                     item_next = next(item)
122                     item_next["name"] = 'g'
123
124                 elif diph == 'Ai_n': item["name"] = 'aei'
125                 elif diph == 'eh_Ch': item["name"] = 'e'
126                 elif diph == 'ou_e': previous["name"] = 'eh'
127                 elif diph == 'j_rl': previous["name"] = 'rL_'
128                 elif diph == 'b_rl': previous["name"] = 'rL_'
129                 elif diph == 'h_rL_': previous["name"] = 'l'
130                 elif diph == 'Sh_rl': item["name"] = 's'
131                 elif diph == 'A:_Oy': previous["name"] = 'ou'
132                 elif diph == 'Ng_oe': item["name"] = 'n'
133                 elif diph == 'e:_e' or diph == 'eh_eh': item["name"]
                        = 'e'
134                 elif diph == 'O_oey': item["name"] = 'O:'
135                 elif diph == 'ou_Ch': item["name"] = 'ou:'
136                 elif diph == 'u:_y': item["name"] = 'O:'
137                 elif diph == 'm_h' and str(get_feature(word_orig, "
                        name")) == 'Ahmed': previous["name"] = 'k'
138                 elif diph == 'oe_ou' and str(get_feature(word_orig,
                        "name")) == 'forfulgt':
139                     previous["name"] = 'f'
140                     item["name"] = 'ou'
141
142                 # Sounds a bit weird, but best found...
143                 elif diph == 'Oy_l': item["name"] = 'oey'
144                 elif diph == 'Oy_k': item["name"] = 'oey'
145                 elif diph == 'Oy_m': item["name"] = 'oey'
146                 elif diph == 'Oy_n': item["name"] = 'oey'
147                 elif diph == 'Oy_b': item["name"] = 'oey'
148                 elif diph == 'u_Oy':
149                     previous["name"] = 'oey'
150                     item["name"] = 'O:'
151                 elif diph == 't_Oy': previous["name"] = 'oey'
152                 elif diph == 'b_Oy': previous["name"] = 'oey'
153                 elif diph == 'h_Oy': previous["name"] = 'oey'
154                 elif diph == 'n_Oy': previous["name"] = 'oey'
155
```

```
156             elif diph == 'i_y': previous["name"] = 'y:'
157
158             elif diph == 'rL__ou:': item["name"] = 'rl'
159
160             elif diph == 't_j': previous["name"] = 'i'
161
162             elif diph == 'Ch_Ai':
163                 previous["name"] = 'aei'
164                 item["name"] = 'Sh'
165
166             elif diph == 'eI_Z_': item["name"] = 'eh'
167             elif diph == 'eh_h': item["name"] = 'e'
168             elif diph == 'r_Ai' and 'air' == string.lower(str(
                    word_orig["name"])[:3]): previous["name"] = 'e:'
169             elif diph == 'u:_u': previous["name"] = 'u:'
170             elif diph == 'u_rd': item["name"] = 'u:'
171             elif diph == 'r__ae': item["name"] = 'r'
172             elif diph == 'eh_O': item["name"] = 'e'
173
174             elif diph[:3] == 'th_': item["name"] = 't'
175
176             # Could be better... from /k @U/ to /k O: u/
177             elif diph[:3] == '@U_':
178                 item["name"] = 'u'
179                 O_diph = EST_Item()
180                 O_diph["name"] = 'O:'
181                 O_diph_end = item["end"]
182                 O_diph_parent = parent(item, "SylStructure")
183                 previous["end"] = float(str(float(O_diph_end) -
                        0.01)[:5])
184                 O_diph["end"] = O_diph_end
185                 previous = u2_seg.append(O_diph)
186                 append_daughter(O_diph_parent, "SylStructure",
                        O_diph)
187
188             elif diph[:3] == 'Ai_': # Could be better...
189                 item["name"] = 'i'
190                 A_diph = EST_Item()
191                 A_diph["name"] = 'A'
192                 A_diph_end = item["end"]
193                 A_diph_parent = parent(item, "SylStructure")
194                 previous["end"] = float(str(float(A_diph_end) -
                        0.01)[:5])
195                 A_diph["end"] = A_diph_end
196                 previous = u2_seg.append(A_diph)
197                 append_daughter(A_diph_parent, "SylStructure",
                        A_diph)
198
199             elif diph[:3] == 'rt_': # Could be better...
200                 item["name"] = 't'
201                 r_diph = EST_Item()
202                 r_diph["name"] = 'r'
203                 r_diph_end = item["end"]
204                 r_diph_parent = parent(item, "SylStructure")
```

```
205                 previous["end"] = float(str(float(r_diph_end) -
                        0.01)[:5])
206                 r_diph["end"] = r_diph_end
207                 previous = u2_seg.append(r_diph)
208                 append_daughter(r_diph_parent, "SylStructure",
                        r_diph)
209
210             elif diph[:3] == 'rd_': # Could be better...
211                 item["name"] = 'd'
212                 r_diph = EST_Item()
213                 r_diph["name"] = 'r'
214                 r_diph_end = item["end"]
215                 r_diph_parent = parent(item, "SylStructure")
216                 previous["end"] = float(str(float(r_diph_end) -
                        0.01)[:5])
217                 r_diph["end"] = r_diph_end
218                 previous = u2_seg.append(r_diph)
219                 append_daughter(r_diph_parent, "SylStructure",
                        r_diph)
220
221             elif diph == 'u_u' and str(get_feature(word_orig, "
                    name")) == str(get_feature(parent(parent(
                    previous_orig, "SylStructure"), "SylStructure"),
                    "name")):
222                 # Should do something here if the 'u_u' diphone
                        is in the same word, e.g the word "wood"
223                 # Sets append_next to false, to avoid /u/ twice
                        in a row.
224                 append_next = False
225
226             elif diph == 'ae:_Ai': # Could be better...
227                 previous["name"] = 'A'
228                 item["name"] = 'aei'
229                 j_diph = EST_Item()
230                 j_diph["name"] = 'j'
231                 j_diph_end = item["end"]
232                 j_diph_parent = parent(item, "SylStructure")
233                 previous["end"] = float(str(float(j_diph_end) -
                        0.01)[:5])
234                 j_diph["end"] = j_diph_end
235                 previous = u2_seg.append(j_diph)
236                 append_daughter(j_diph_parent, "SylStructure",
                        j_diph)
237
238             elif diph == 'sil_Z_': # Could be better...
239                 eh_diph = EST_Item()
240                 eh_diph["name"] = 'eh'
241                 eh_diph_end = item["end"]
242                 eh_diph_parent = parent(item, "SylStructure")
243                 previous["end"] = float(str(float(eh_diph_end) -
                        0.01)[:5])
244                 eh_diph["end"] = eh_diph_end
245                 previous = u2_seg.append(eh_diph)
246                 append_daughter(eh_diph_parent, "SylStructure",
```

```
                eh_diph)
247
248             elif diph[-3:] == '_ae': previous["name"] = 'aei'
249             elif 'ae_' in diph: item["name"] = 'aei'
250             elif 'eI_' in diph: item["name"] = 'aei'
251
252             # --> Norsk-engelsk :-)
253             elif 'w_' in diph: item["name"] = 'v'
254             elif '_r_' in diph: previous["name"] = 'r'
255             elif 'r__' in diph: item["name"] = 'r'
256
257             # Ikke god...
258             elif 'aU_' in diph: item["name"] = 'aeu'
259             elif 'D__' in diph: item["name"] = 'd'
260             elif 'rl_' in diph: item["name"] = 'l'
261             elif '_rL_' in diph and str(get_feature(word_orig, "
                    name")) == str(get_feature(parent(parent(
                    previous_orig, "SylStructure"), "SylStructure"),
                    "name")): previous["name"] = 'l'
262
263             elif 'sil_'+str(get_feature(previous, "name")) in
                    diphones_list and str(get_feature(item, "name"))+
                    '_sil' in diphones_list:
264                 sil = EST_Item()
265                 sil["name"] = 'sil'
266                 sil_end = float(str(float(previous["end"]) +
                        0.01)[:5])
267                 sil_parent = parent(item, "SylStructure")
268                 item["end"] = float(str(float(sil_end) + 0.01)
                        [:5])
269                 sil["end"] = sil_end
270                 previous = u2_seg.append(sil)
271                 append_daughter(sil_parent, "SylStructure", sil)
272             elif str(get_feature(next(item), "name"))+'_'+str(
                    get_feature(previous, "name")) in diphones_list:
273                 #Legger ikke til item
274                 append_next = False
275                 remove = True
276             elif str(get_feature(item, "name"))+'_'+str(
                    get_feature(prev(previous), "name")) in
                    diphones_list:
277                 #Legger ikke til previous (dvs endrer navn på
                        previous til items navn, og legger ikke til
                        item)
278                 append_next = False
279                 previous["name"] = item["name"]
280                 remove = True
281             else:
282                 print ' Could not solve missing diphones by
                        inserting /sil/ phonem between /' + str(
                        get_feature(previous, "name")) + '/ and /' +
                        str(get_feature(item, "name")) + '/ or other
                        solutions.'
283                 unable = True
```

```
284             diph = str(get_feature(item, "name")) + '-' + str(
                    get_feature(previous, "name"))
285             print 'endret til', diph,
286
287             if silence:
288                 sil_score = sil_score + 1
289             elif remove:
290                 remove_score = remove_score + 1
291             elif unable:
292                 unable_score = unable_score + 1
293             else:
294                 sub_score = sub_score + 1
295         if append_next:
296             previous = u2_seg.append(item)
297         else:
298             append_next = True
299         previous_orig = item_orig
300         print ''
301
302     print ''
303
304     # Saving the new utterance in a .utt file that can be loaded
            into Festival
305     utt_id = utt_id + '.utt'
306     u2.save(utt_id)
307     if sil_score + sub_score + remove_score + unable_score == 0:
308         utt_score[0] = utt_score[0] + 1
309     else:
310         utt_score[1] = utt_score[1] + sil_score
311         utt_score[2] = utt_score[2] + sub_score
312         utt_score[3] = utt_score[3] + remove_score
313         utt_score[4] = utt_score[4] + unable_score
314
315     sil_score = 0
316     sub_score = 0
317     remove_score = 0
318     unable_score = 0
319
320 manuscript = None
321 utt_id = 'saved'
322 if sys.argv[1]:
323     src = str(sys.argv[1])
324     print src
325     if src[-4:] == '.txt':
326         manuscript = True
327     elif src[-1] not in ['.', '!', '?']:
328         src = src + '.'
329 else:
330     print 'Ingen setning å syntetisere'
331     src = '.'
332
333 if manuscript:
334     src_file = open(src, 'r')
335     for line in src_file.readlines():
```

```
336             utt_id = line[1:9]
337             src = str(line[11:-3])
338             print utt_id, src
339             synthezise(src, utt_id)
340     elif src != '.':
341         synthezise(src, utt_id)
342
343     print utt_score
```