Taligenkänning i en flygledningssimulator-miljö

Niclas Reisnert

Examensarbete för 15 hp
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds Universitet

Thesis for a diploma in computer science, 15 ECTS credits
Department of computer science, Faculty of science, Lund University

Taligenkänning i en flygledningssimulator-miljö

Att vara en del av en flyglednings-simulation är ansträngande, särskilt för piloterna i simulationen, som emellanåt kan behöva kontrollera tiotalet flygplan samtidigt. En stor mängd information måste tas emot och skickas vidare, ofta igenom ett aningen klumpigt mus-och-tangentbords-interface.

Målet för detta examensarbete var i första hand att utveckla en applikation som kunde utöka, eller möjligtvis även ersätta, den vanliga sättet att ge indata till simulatorn, med ett taligenkännings-system. Målapplikationen skulle vara flyglednings-simulatorn NARSIM. Resonemanget bakom idén var att ett passade taligenkännings-interface skulle kunna göra det lättare att ge kommandon till simulatorn. En demoversion implementerades, tillsammans med en grammatik innehållande en del av de kommandon tillgängliga i simulatorn. En Telnet-session användes för att skicka användarens kommandon till NARSIM, detta gjorde applikationen fullt fungerade, om än med ett reducerat antal implementerade kommandon tillgängliga.

Testprocessen fokuserade på att mäta applikationens träffsäkerhet, och i förlängning även träffsäkerheten hos den underliggande Microsoft Speech API. Fokus fanns också på hur resultaten från en tränad och en otränad användarprofil skilde sig.

Slutligen tar denna rapport upp några av de potentiella förbättringarna som bör göras på applikationen, en bedömning av projektets potential att lämna utvecklingsstadiet, och saker som gick bra och mindre bra under utvecklingsprocessen.

Speech recognition in an Air Traffic Control simulator-environment

Being a part of an Air Traffic Control simulation is a exerting business, especially for the pilots, who sometimes are in control of as many as ten different aircraft at the same time. A staggering amount of information has to be received and relayed, often through a somewhat cumbersome mouse-and-keyboard-controlled interface.

The primary goal this thesis was to develop a application that could augment, or possibly even replace, the ordinary input configuration with a system that uses speech recognition, with the ATC simulator NARSIM as the target application. The reasoning behind this was that a suitable speech-controlled interface would make it easier to issue commands so the simulator. A demo was implemented, along with a grammar that covered a subset of the commands available. A Telnet session was used to pass on the users commands to NARSIM, making it a fully functional application, albeit with a reduced number of available commands.

The testing process was centered around measuring the applications accuracy, and by doing that, the accuracy of the underlying Microsoft Speech API. Focus was also on the differences in results between using a trained speech profile when giving commands, versus a untrained one.

Lastly, this report covers some of the potential improvements that should be made to the application, an assessment of the projects potential of ever leaving the development stage, and things that went well or less well during the development process.

Innehållsförteckning

Förord

Examensarbetet genomfördes på Luftfartsverkets ASD/DEV-avdelning på Sturups flygplats under tiden 17/11 2008 till 30/3 2009.

Jag vill tacka:

Min handledare Pierre Nugues, vars kurs i språkhantering fick mig intresserad av området taligenkänning, och även för all hjälp med rapporten.

Mina kolleger på LFV ASD/DEV för assistans, både under utvecklings- och testarbetet, och det välkomnande jag känt under den tid jag spenderat med dem.

Alla andra som lånat sina röster till testarbetet, vissa upprepade gånger under tiden som testparametrarna förändrades

Min far Christian, för den ursprungliga idén, resesällskapet till och från Sturup, och allt däremellan.

Och resten av er, som hjälp till med input och idéer. För många för att nämna här, men icke desto mindre signifikanta.

1. Inledning

Taligenkänning definieras som processen att konvertera mänskligt tal till någon form av data som en maskin, dator eller annan, kan hantera. Skall ej blandas ihop med det närliggande området *röstigenkänning*, vars fokus ligger på att identifiera rösterna som talar, och inte själva talet. Att kunna tala naturligt med sin dator och få den att svara på samma sätt ligger fortfarande bortom våra möjligheter, men med en passande grammatik och god artikulation kan man fortfarande uppnå tillfredsställande resultat [6].

1.1 Problemformulering

Målen jag hade när jag började arbeta var att ta fram en taligenkännings-applikation som skulle kunna underlätta och effektivisera interaktionen mellan användare och flygledningssimulatorn NARSIM, ett program som används för att simulera flygledning i utbildningssyfte. De faktorer som jag såg vara av särskild vikt inför utvecklingsarbetet var följande:

- Grammatikens skalbarhet – applikationens grammatik skulle gå att expandera tämligen smärtfritt
- Att programmet i princip skulle kunna förbi orört medan grammatiken expanderades
- Att programmet använde mjukvara som var hyfsat ny, det vill säga representativ för hur effektiv taligenkänning är idag.

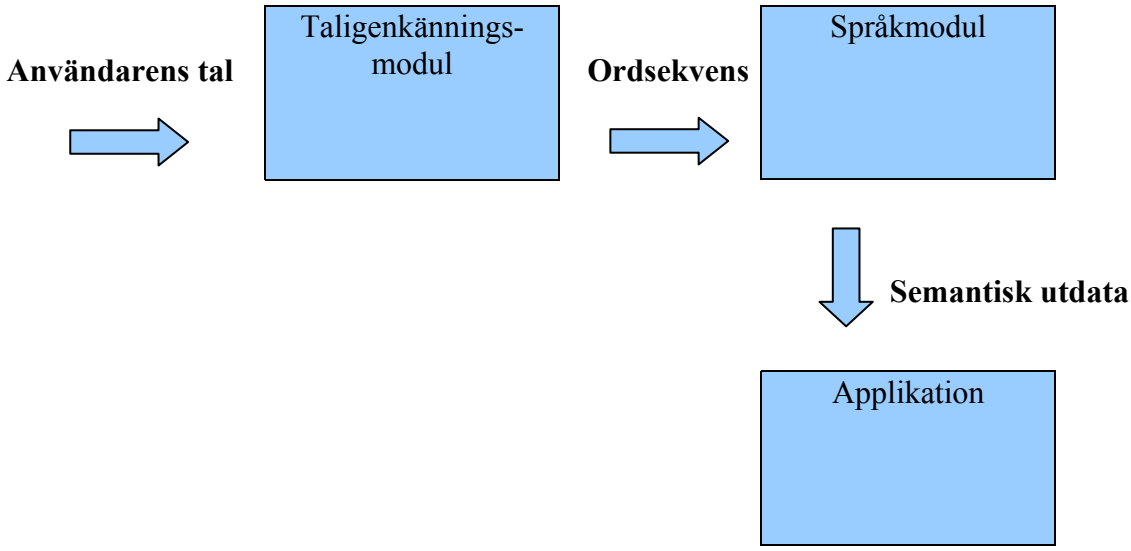
Efter att jag producerat en demoversion, som kunde förstå ett begränsat antal kommandon, konvertera dem till ett format som NARSIM förstår, och sedan skicka dem vidare till simulatorn för exekvering, planerade jag att göra en del tester på applikationens effektivitet i olika miljöer. Exempel på tester jag fann intressanta inkluderar:

- Dess träffsäkerhet, procentuellt.
- Hur bra den hanterade användare med dialekter som skiljer sig från min egen.
- Hur pass stor skillnad det var om programmet kördes på en dator som genomgått träning i att känna igen talkommandon, jämfört med om det kördes på en dator som inte hade genomgått samma träning.

Detta ämne har tidigare behandlats av mina arbetsgivare Luftfartsverket i ett projekt för cirka 10 år sedan i samarbete med KTH. Detaljer om detta projekt visade sig vara svåra att komma över, men det intryck jag fick av samtal med mina kolleger så verkade det inte ha varit särskilt lyckat, systemet ska inte ha varit träffsäkert nog. Det finns det ett antal liknande applikationer ute på marknaden, men dessa är integrerade i andra simulatorer, och är alltså inte passande att använda i konjunktion med NARSIM [12].

2. Bakgrund

2.1 Hur fungerar taligenkänning?



Stegvis, lite förenklat, fungerar taligenkänning så här[1]:

- Användaren talar i en mikrofon, och ger på så sätt indata till programmet. En menings slut identifieras antingen av ett förutbestämt ord, en förutbestämd fras, eller en viss tid av tystnad.
- Taligenkännings-modulen utför en akustisk analys av det som sades, och extraherar en serie akustiska symboler därifrån. Med hjälp av statistiska språkmodeller baserade på *n-gram*, alternativt en grammatik, bestäms vilken sekvens av ord det är störst sannolikhet att yttrandet innehöll.
- Denna sekvens av ord skickas vidare till en språkmodul som utifrån den genererar kommandon eller förfrågningar som målapplikationen kan förstå.

Det finns generellt två olika sätt att hantera uppbyggandet utav en taligenkännings-applikation, där det första är *command and control (c&c)*, och det andra är *dictation*. Det som skiljer dessa åt, är att dictation enbart använder sig av en språkspecifik språkmodell, medan c&c-varianten också använder en skräddarsydd grammatik som är specifik för just den applikationen, en *CFG (context-free grammar.)*

2.2 Hur effektivt är taligenkänning?

En taligenkännares effektivitet mäts oftast i termerna *träffsäkerhet* och *hastighet*. Träffsäkerhet handlar i stort sett om hur stor andel av det som sades, som uppfattades korrekt av applikationen, medan hastighet mäts i en realtidsfaktor RTF = P/I, där P är tiden det tar att behandla ett yttrande utav tiden I. En realtidsfaktor 1 betyder sålunda att taligenkänningen sker i realtid, och en faktor 4 betyder att det tar fyra gånger så lång tid att behandla ett yttrande, som att uttala den.

Inget taligenkännings-system är dock hundraprocentigt perfekt. Några av de vanligaste källorna till att ett yttrande misslyckas att kännas igen är:

- Systemet ”hör” dåligt. För mycket bakgrundsljud eller en dålig mikrofon påverkar kvalitén på den digitala representationen av yttrandet.
- Användaren talar otydligt eller för fort.
- Yttrandet innehöll en eller flera *homonymer* som misstolkades.

Och den självklara: Om användaren inte följer de regler som satts upp av en eventuell grammatik så kommer sällan yttringarna att tolkas, och än mer sällan kommer de tolkas på rätt sätt. Just homonymer, ord som låter likadant men har olika betydelse, är något som dagens program i princip inte kan särskilja enbart på en rent fonetisk basis. Detta problem kan ofta kringgås igenom att man tar hänsyn till ordens kontext, igenom extensiv träning och/eller statistiska modeller.

Det är värt att notera att de flesta taligenkännings-system är tämligen entusiastiska när det kommer till igenkänning, och försöker verkligen tolka det man just sade som *något* som *möjligtvis* passar det som grammatiken föreskriver. Denna tendens att ofta returnera den ”bästa gissningen” kan vara hjälpsam när man stöter på de ovan nämnda felen, men kan också resultera i förvirrande och emellanåt humoristiska misstolkningar.

2.3 Användningsområden för taligenkänning

Taligenkänning används i första hand som ett substitut eller som ett komplement till andra, mer traditionella sätt att ge kommandon till en exempelvis en dator, som exempelvis knapptryckningar, eller pekdon i stil med en datormus. Exempel på områden där taligenkänning används framgångsrikt inkluderar:

- Olika former av telefonbaserade dialogsystem, till exempelvis biljettbokning, supportverksamhet eller enkla former av bankärenden.
- Handikappanpassning av redan existerande applikationer.
- Mobilapplikationer, till exempel navigeringssystem, enkla spel och diktafon-funktioner.
- Andra områden som på olika sätt gör tangentbord + mus-konfigurationen opassande, till exempel vid arbeten som kräver skyddskläder av olika typ.

Ett område där taligenkänning något överraskande har genomgått extensiv testning är i styrning utav icke-kritiska processer i stridsflygplan. En begränsad grammatik, kombinerat med ett behov av att utföra en mängd olika uppgifter simultant, gör detta till en ideal kandidat för denna typ av utveckling. Även i en miljö med tämligen hög volym i form av till exempel motorljud, och andra mer oförutsedda negativa faktorer som G-kraft, så är det möjligt att nå den procentuella träffsäkerhet som krävs för att en pilot ska bli mer effektiv [6][13].

Ett annat ämne, som till sin struktur lämpar sig väldigt väl för implementation av stöd för taligenkänning är träning utav flygledare. Eftersom träningen redan sker i en datormiljö, och språket är designat för att vara strukturerat och icke-ambiguöst, underlättas skapandet av en grammatik för detta ämne. Det var detta jag hade i åtanke när jag definierade det som skulle bli grunden för det arbete den här rapporten ska summera.

3. Verktyg och plattform

3.1 NARSIM

NARSIM är en simulationsplattform för både flygledning- och tornsimulation framtagen av nederländska NLR - *Nationaal Lucht- en Ruimtevaartlaboratorium*. Detta program kan både användas i de inledande stadierna av en flygledares utbildning, och också för att simulera ovanliga, kritiska situationer i träningssyfte[14].

Simulatorn är uppbyggd med en central enhet, servern dit ett antal datorer, klienter är uppkopplade. Vid dessa datorer sitter användarna, antingen flygledare eller så kallade pseudopiloter. Flygledarna sköter sina vanliga roller, det vill säga vägleder de flygplan som finns i den sektor han eller hon har ansvar för. Varje pseudopilot har ett antal flygplan som han eller hon är ansvarig för att styra. Detta förhållande tjänar att uppnå ungefär samma grad utav mänsklig interaktion som det är i verkligheten inom detta område.

Det är på pseudopilotens sida jag såg möjligheten att implementera taligenkänning som ett sätt att effektivisera kommunikationen mellan piloten och de flygplan han eller hon ansvarar för. Vanligtvis sköts detta igenom standard mus/tangentbords-konfiguration. De anledningar jag såg att försöka ersätta detta med taligenkänning, eller åtminstone låta taligenkänning tjäna som ett komplement, är:

- En potentiellt högre frekvens av givna kommandon, när behovet av att navigera i och mellan olika fönster minskar.
- Piloten upprepar alltid det kommando han fått av flygledaren, som en bekräftelse. Om taligenkännings-applikationen använder samma grammatiska struktur så skulle övergången till ett röststyrt program vara naturlig.
- Den begränsade grammatiken och strikta frasstrukturen gör att en applikation tämligen enkelt kan nå goda resultat inom träffsäkerhet.

Det finns en mängd olika system för röstigenkänning tillgängliga på marknaden, spridda över ett antal plattformar. Vilket som är bäst beror i allmänhet vem man frågar, de har alla sina styrkor eller svagheter i form av portabilitet, integrering med plattformen, support, användarvänlighet, och även pris. Det system jag valde att använda, efter råd ifrån min handledare, är det Speech API 5.3 som finns integrerat i Windows Vista, och som även följer med i Microsofts SDK:er. De fördelar jag såg med detta val är[15][16]:

- Användarvänligt, eftersom SDK:n är integrerad i programmeringsverktyg som Microsoft Visual Studio.
- Hyfsat nytt och genomarbetat. Windows Vista släpptes 2006, och innehåller version 5.3 av Windows SAPI. Den första versionen, släpptes 1995 och ingick i Windows 95, och utvecklingen har pågått sedan dess. Vista är det första Windows- operativsystem som har fullt integrerad taligenkänning.
- Stödjer flera olika användarprofiler på samma dator, vilket betydde att jag skulle kunna testa vikten av träning utav taligenkännings-systemet på en enda dator

Jag valde i ett ganska tidigt stadium att förlägga min utveckling till en Windows-plattform, då det är det operativsystem jag har mest erfarenhet att arbeta i, och då taligenkänning var så pass integrerat i Vista var det ett naturligt val. SDK 5.3 är också svårt, om inte omöjligt, att få att fungera tillfredsställande på tidigare versioner av Windows som till exempel Windows XP, utan då hade jag fått nöjd mig med det tidigare 5.1 som saknade 5.3s stöd för semantisk datahantering. De alternativa system jag undersökte visade sig i allmänhet ha en lägre effektivitet, eller sämre eller inga möjligheter att använda dem i en tredjepartsapplikation[6][17][18]. De taligenkänningsprojekt som fanns tillgängliga på Linux-plattformar var i allmänhet på en lägre nivå utvecklingsmässigt, och saknade stöd för antingen ett programspråk jag var någorlunda bekväm med, eller helt enkelt stöd för engelska, det språk jag skulle basera min applikation runt [9][10].

3.2 SAPI 5.3 – Speech application programming interface

SAPI 5.3 tillhör SAPI 5 API-familjen, ursprungligen släppt år 2000. Traditionellt har alla Microsofts SAPI-er levererats direkt med operativsystemet eller funnits i det aktuella SDK:et. De är designade dels för att vara väl integrerade med operativsystemet i sig, andra Windows-applikationer, som Microsoft Office, men också för att tillhandahålla gränsytor som programmerare kan använda för att skriva sina egna röstapplikationer. Programmerare som använder denna API har möjligheten att antingen använda en delad röstigenkännare som kör i en separat process, och därför är åtkomlig för alla processer, eller att skapa sin egen *InProc* instans av röstigenkännar-objektet för att på så sätt ha full kontroll över vad som det skall lyssna efter. Fördelen med att använda den delade igenkännaren är att man får tillgång till alla de kontroll-kommandon som det integrerade taligenkännings-programmet i Vista tillhandahåller. Om man å andra sidan anser att den typen av kommandon inte tillför något till den applikation man vill skriva, används med fördel ett inproc-objekt.

SAPI 5.3 stöder WC3s standarder för taligenkänning, SRGS (Speech Recognition Grammar Specification), och för talsyntes, SSML (Speech Synthesis Markup Language), med den begränsningen att den inte stöder grammatiker med Augmented BNF-syntax[19]. Det är enbart SRGS-standarden som är intressant i det här fallet, då tal bara ska kännas igen, inte genereras.

3.3 SRGS

SRGS är en standard för hur taligenkännings-grammatiker specificeras[4]. En taligenkännings-grammatik är ett set av ordmönster, som berättar för taligenkännaren vad den kan förvänta sig att användaren ska säga. SRGS stöder två olika syntaxer, en baserad på XML och en baserad på ABNF-syntax. XML-syntaxen är den vanligare och också den enda av de två som stöds utav Microsofts SAPI. En grammatik består av en eller flera regler, varav en är rotregeln, eller utgångspositionen för grammatiken. Dessa regler specificerar vad som kan sägas i varje given situation, och kan också användas för att definiera valmöjligheter, upprepningar och ljud som ska ignoreras.

En liten exempelgrammatik följer, den känner igen ett antal variationer på jakande eller nekande uttryck. Taligenkännaren laddar in den här grammatiken och väntar sedan på att användaren skall säga något som matchar rotregeln.

```
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
      xml:lang="en-US" tag-format="semantics/1.0" root="root">

      <rule id = ”root” scope = ”public”>
            <one-of>
                  <item>
                        <ruleref uri=”#yes_rule”/>
                  </item>
                  <item>
                        <ruleref uri=”#no_rule”/>
                  </item>
            </one-of>
      </rule>
      <rule id = ”yes_rule” scope = ”private”>
            <one-of>
                  <item>
                        yes
                  </item>
                  <item>
                        yeah
                  </item>
                  <item>
                        <item repeat= “0-1”>
                              it is
                        </item>
                        correct
                  </item>
            </one-of>
      </rule>
      <rule id = “no_rule” scope = “private”>
            <one-of>
                  <item>
                        no
                  </item>
                  <item>
                        nope
                  </item>
                  <item>
                        <item repeat= “0-1”>
                              it is
                        </item>
                        incorrect
                  </item>
            </one-of>
      </rule>
</grammar>
```

Grammatiken ovan kan även representeras på följande lite mer överskådliga sätt:

root → yes_rule | no_rule

yes_rule → yes | yeah | (it is)* correct

no_rule → no | nope | (it is)* incorrect

Denna grammatik kommer att känna igen något av de sex yttranden som finns beskrivna. One-of-taggen betyder att endast en av de efterföljande artiklarna kommer kännas igen, exakt en gång. Repeat-taggen under ”correct”- respektive ”incorrect”-artiklarna betyder å andra sidan att ”it is” kan förekomma före de här två artiklarna noll eller en gång, vilket gör skapar ett visst mått av valfrihet i vad man säger, så länge man håller sig innanför grammatikens ramar. Vanligtvis när ett igenkännande inträffar så returneras den igenkända strängen, men detta är ofta inte önskvärt i applikationer där man inte är intresserad av hela yttrandet, utan där man bara vill extrahera specifika data. Det är här som *SISR*, Semantic Interpretation for Speech Recognition kommer in.

3.4 SISR

SISR beskriver en syntax för tag-element i SRGS-grammatiker, som används för att modifiera den utdata som fås ifrån taligenkännaren vid ett lyckat yttrande[5]. Grammatiken på föregående sida kan utökas enligt följande för att uppnå ett mer konsekvent resultat:

```
<grammar>
…
    <rule id = ”yes_rule” scope = ”private”>
        <one-of>
            <item>
                yes<tag>out = “yes”;</tag>
            </item>
            <item>
                yeah<tag>out = “yes”;</tag>
            </item>
            <item>
                <item repeat= “0-1”>
                    it is
                </item>
                correct
                <tag>out = “yes”;</tag>
            </item>
        </one-of>
    </rule>
…
</grammar>
```

Vilket som helst av de jakande svarsalternativen skulle nu producera utdatan ”yes”, vilket gör det väldigt mycket lättare att programmera applikationer som drar nytta av den. Det är även möjligt att returnera ett ospecificerat antal separata utdata igenom att initiera ”out” som en lista. Exemplet nedan är hämtat från den grammatik jag skapat.

```
<rule id ="callsign_numbers" scope="private">
    <tag>out = new Array;</tag>
    <item repeat="1-4">
        <ruleref uri="#integers_digits"/>
        <tag>out.push(rules.integers_digits);</tag>
    </item>
</rule>
<rule id="integers_digits" scope="private">
    <one-of>
        <item>
            one
            <tag>out =1</tag>
        </item>
        <item>
            two
            <tag>out =2</tag>
        </item>
        <item>
            <one-of>
                <item>three</item>
                <item>tree</item>
            </one-of>
            <tag>out =3</tag>
        </item>
        <item>
            four
            <tag>out =4</tag>
        </item>
        <item>
            five
            <tag>out =5</tag>
        </item>
        <item>
            six
            <tag>out =6</tag>
        </item>
        …
    </one-of>
    …
</rule> (siffror 7-0 utelämnade)
```

Regeln “callsign_numbers” tillåter ett tal mellan 0 och 9, upprepat 1-4 gånger (Det behöver inte vara samma tal alla 1-4 gångerna). Utdatan ifrån taligenkännaren kommer att innehålla alla de tal som kändes igen med hjälp av den här regeln. De två olika sätten “3” kan kännas igen på är ett resultat av det lite speciella uttal som används inom flygledning.

4. Design

4.1 Programmet

Det program jag producerat, som är en del av mitt slutresultat, är skrivet i C#, och är inte mer än ett par hundra rader långt. Jag använder mig av biblioteket System.Speech som gör det mesta av grovjobbet.

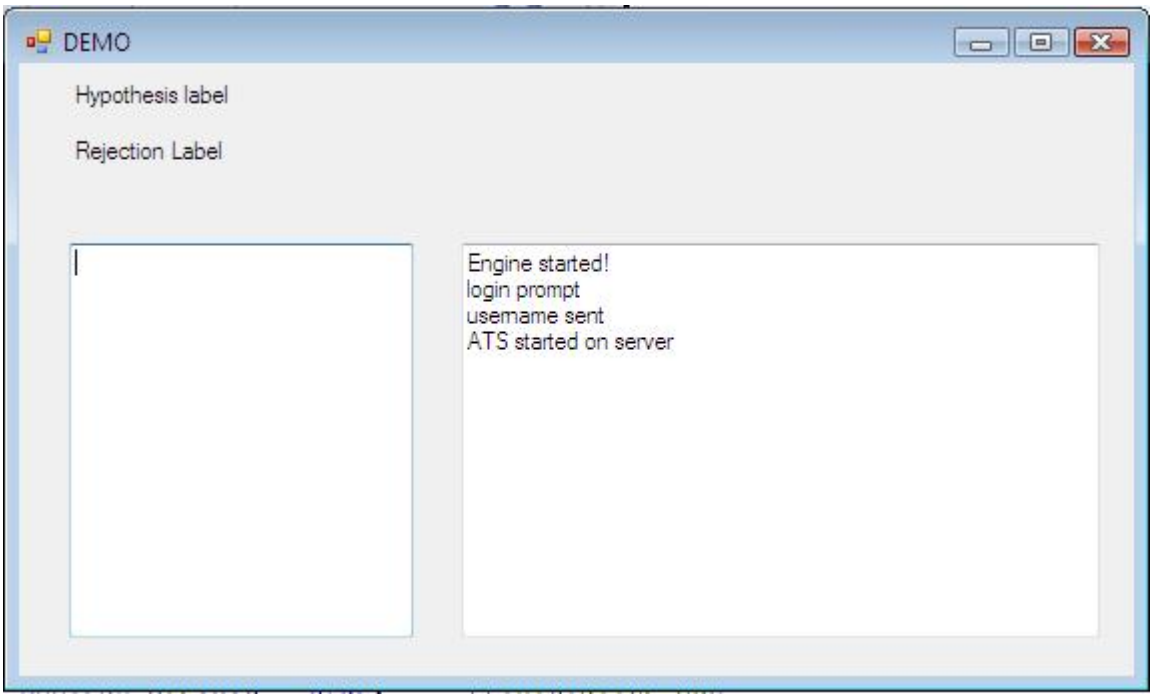
Programmet består av användarens val av antingen ett SpeechRecognizer-objekt eller ett SpeechRecognitionEngine-objekt. Det som skiljer de åt är att SpeechRecognizer-objektet delas mellan alla taligenkännings-processer, inklusive Vistas egna. Detta innebär att kommandon som ”open”, ”file”, ”start” och så vidare kommer att tolkas av Vistas egna taligenkännings-program. Detta kan producera underliga resultat, så mina rekommendationer är att använda ett SpeechRecognitionEngine-objekt istället. Detta är ett så kallat InProc-objekt som är unikt för just denna process. Därför så är det endast den specificerade grammatiken som kan generera ett igenkännande.

Detta objekt initieras och laddar sedan in en specifik grammatik, som i det här fallet innehåller det språk som används för att styra flygplan i NARSIM-miljö. Det är fullt möjligt att ha flera olika grammatiker inladdade och/eller aktiva samtidigt, samt att ge de olika prioritet.

Definierade i programmet är också ett antal *EventHandlers*, som beskriver vilka instruktioner som skall utföras när taligenkännaren har en hypotes av vad som sägs, när den förkastar något som sagts på grund av den inte är tillräckligt säker, eller när ett igenkännande gjorts.

Själva programmet är egentligen inte mycket mer än ett mellanled mellan användaren och NARSIM, med vilken den kommunicerar via en Telnet-session som startas samtidigt som resten av programmet initieras[3].

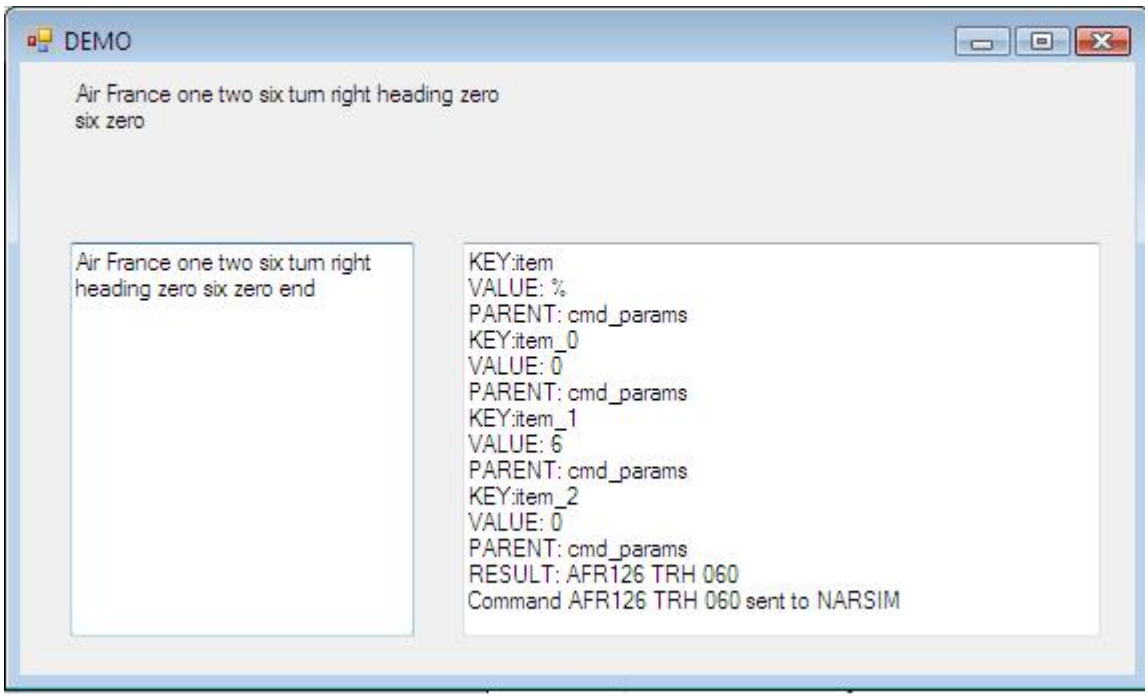
Programmets GUI är tämligen simpel, och består endast av ett fönster. I detta fönster finns två textrutor, varav en, den vänstra, är avsedd att skriva ut lyckade igenkännanden, och den högra används för att presentera debug-meddelanden. Etiketten benämnd ”Hypothesis label” i exemplet på nästa sida låter användaren veta taligenkännarens hypotes, som uppdateras kontinuerligt under ett yttrande. Etiketten ”Rejection label” uppdateras när taligenkännaren misslyckats med att producera en fullgod tolkning av det som sades.



Det initiala läget

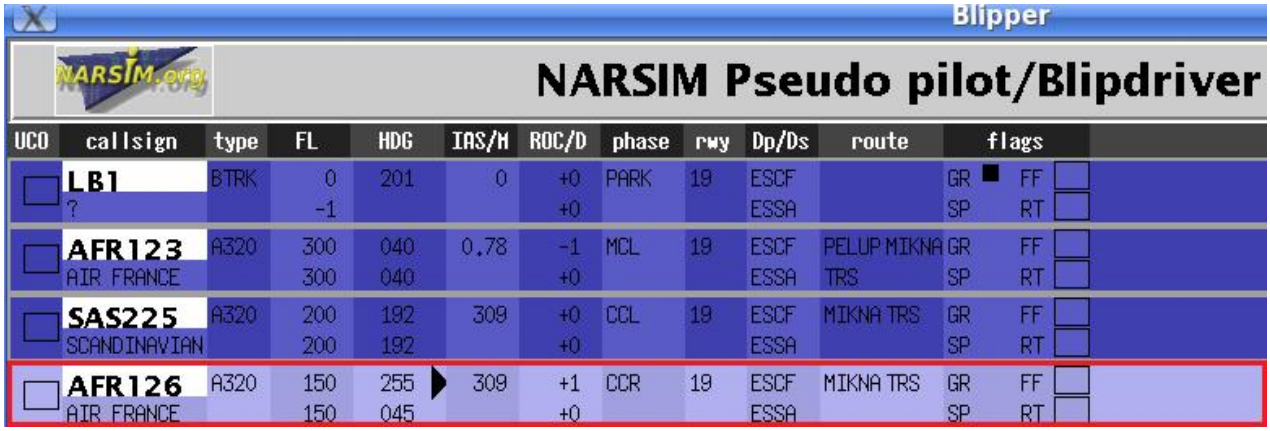
Ett exempel på hur programmet fungerar kan se ut så här:

- Användaren yttrar frasen ”Air France one two six turn right heading zero six zero end” Det här är en fras som kommer att accepteras utav taligenkännaren, då den ingår i den specificerade grammatiken.
- Ett SpeechRecognizedEventArgs-objekt sänds till metoden för hantering av utdata. Detta objekt innehåller all den semantiska information som grammatiken utökats med. Denna information är i det här fallet kommando- och parametersträngar som NARSIM förstår.
- Den semantiska informationen extraheras ifrån objektet, och en sträng skapas, som sedan ska skickas vidare till NARSIM. Den aktuella strängen skulle vid korrekt igenkänning se ut så här: AFR126 TRH 060.
- Strängen sänds till NARSIM via den sedan innan öppnade Telnetsessionen. NARSIM tolkar kommandot, och utför det om det är kontextuellt korrekt. Det är t.ex. upp till användaren att avgöra om det flygplan han eller hon vill ge order finns i simulationen.



Programmet har här känt igen det som sades, och skickat vidare kommandot till NARSIM

- Resultatet syns nu i NARSIM i pseudopilotens fönster, eller "Blipdriver" Det flygplan som är markerat är det AFR126 som precis gavs kommandot att svänga åt höger till kurs 060. Den svarta pilen i HDG-kolumnen betyder att flygplanet är i färd att svänga åt höger.



AFR126 svänger åt höger från sin tidigare kurs 255

- Kursändringarna kan också spåras visuellt via ett annat av NARSIMs fönster: wARP. Detta fönster ger en bild av var flygplanen i simulationen befinner sig i relation till varandra, och man kan även följa dem medan de utför sina instruktioner.

wARP med AFR126 markerad. 150 är flygplanets höjd i tusentals fot.

- Efter att kommandot skickas till NARSIM går programmet tillbaka till utgångsläget och väntar på nästa yttring från användaren.

4.2 Grammatiken

Min applikation är av typen command-and-control, och använder sig därför av en grammatik som är specialskriven för just den applikationen. Den grammatik jag tagit fram definierar kommandosträngar på följande form (en mer uttömmande beskrivning finns i Bilaga B):

<CALLSIGN> <COMMAND> end

Där <CALLSIGN> består av ett flygbolags specifika, inte alltid intuitiva, callsign (Scandinavian för Scandinavian Airlines System, Speedbird för British Airways, och så vidare), samt ett id-nummer på 1-4 siffror. <COMMAND> är i sin tur uppdelad i tre underklasser, där jag sorterat kommandona efter hur svåra de är att implementera i grammatiken. Dessa klasser är:

- Simple commands, som bara består av ett eller två ord, till exempel *startup* eller *take off*.
- Semi-complex commands, som består av ett kommando och maximalt en parameter, som till exempel *set landing gear*- eller *set flaps*-kommandona, som tar parametrarna *up*|*down* eller *retracted*|*extended*, respektive.
- Complex commands, som består av ett kommando och en serie parametrar, ofta någon kombination av bokstäver och/eller siffror. Då det inte på förhand går att förutse det antal tecken parametrarna kommer att bestå av, så kräver det lite mer av grammatiken. Ett exempel är *taxto*-kommandot, som tar n bokstäver och/eller siffror som parametrar. Dessa ska tillsammans bilda ett ID för den punkt som flygplanet skall taxa till.

Slutet av en kommandosträng markeras med ett ”end”. Detta är i syfte att göra det enklare för taligenkännaren att veta när man är färdig med sin mening och reducera antalet förhastade tolkningar.

Förutom dessa regler så innehåller grammatiken ett antal hjälpregler, till exempel en som känner igen NATOs fonetiska alfabet (alfa, bravo, charlie, delta,..,zulu) en som känner igen hastigheter i mach-form, och en serie regler som tillsammans gör det möjligt att känna igen alla heltal < 100000. Dessa regler är fullt återanvändningsbara, vilket tillför en viss skalbarhet till grammatiken.

Grammatiken är utökad med SISR-notation, vilket låter mig kontrollera i högre grad vad som sänds tillbaka till programmet vid en lyckad igenkänning. Detta låter mig översätta användarens tal till kommandon som NARSIM förstår direkt i taligenkännaren, istället för att använda stora, klumpiga switch-satser i själva programmet, som dessutom måste skrivas om varje gång som grammatiken skall expanderas med nya kommandon.

4.3 Testresultat

De tester jag utförde efter att utvecklingsarbetet var avslutat koncentrerade sig i huvudsak på att mäta programmets, och i förlängningen även Microsofts APIs, effektivitet. Utförandet jag valde var att jag valde ut 10 meningar som jag uppfattade täckte in de mest relevanta delarna av den grammatik jag konstruerat. Dessa meningar är följande:

Scandinavian two two five take off end
Scandinavian one three four resume own speed end
Air France one eight set landing gear down end
Air France three six three set flaps retracted end
Scandinavian nine nine five speed three five zero end
Scandinavian two two five speed point eight zero end
Scandinavian one three four set speed limit three hundred and fifty end
Air France one eight turn left heading one four zero end
Air France three six three altitude three thousand end
Scandinavian nine nine five turn right heading zero five zero end

20 personer, 13 män och 7 kvinnor fick läsa upp de här meningarna, användandes samma headset som jag använt under utvecklingsfasen. Jag gav dem ingen annan hjälp på förhand förutom det tips att en taligenkännings-applikation uppfattar vad man säger bättre om man artikulerar ordentligt, till exempel som en radio- eller TV-programledare. Kön, eventuella intressanta dialekter och andra relevanta saker noterades för varje person (Se bilaga A).

Stödet för att ha multipla användarprofiler i Windows Vista gjorde att jag lätt kunde göra två separata tester med meningarna ovan, ett test där min egen talprofil som jag tränat extensivt, var aktiv, samt ett test med en helt ny och otränad profil. Alla testpersoner fick läsa upp meningarna en gång med var profil aktiv, i ett försök att utröna om taligenkännarens träning var en viktig faktor i detta fall.

Snitt-träffsäkerheten med den sedan innan tränade användarprofilen var 85,5%, medan snittet för den icke tränade för samma meningar var 81,0%. 8 personer fick sämre resultat med den otränade profilen, för 6 var det oförändrat, medan 6 fick bättre resultat.

Snitt-träffsäkerheten för de tio meningarna befann sig inom intervallet 92,5% och 70%, med meningar nr. 4 och nr. 8 med bäst resultat, och mening nr. 5 med sämst.

Det som jag noterade var gemensamt för de två personer med sämst testresultat var att de båda hade flygledarutbildning. Inga andra speciella företeelser, av de jag lade märke till, så som dialekt eller kön, verkade påverka resultaten.

Tester gjordes också på hur pass mycket resultaten påverkades om man fann sig i en omgivning med mycket störande ljud runt omkring. Jag satt på flertalet ställen med mycket mänsklig aktivitet, exempelvis samtal, runt omkring mig, och läste upp fraser. Jag fann att det gick att upprätthålla en tillfredsställande hög proportion framgångsrika igenkännanden, så länge man tog sig tid att artikulera ordentligt. Ett byte utav headset, till ett Logitech USB 350, som skedde ungefär halvvägs genom testningen gjorde att bullerproblemet reducerades till den grad att det inte var någon större skillnad mellan att befinna sig i en tyst miljö, eller en lite mer livlig.

5. Diskussion

5.1 Angående testresultaten

Träffsäkerheten var i genomgående aningen högre än vad jag förväntat mig, både för den profil jag tränat sedan tidigare, och för den helt otränade profilen. Resultaten beror med stor sannolikhet på det faktum att grammatiken är så pass liten och snäv att det inte finns mycket rum för taligenkännaren att göra feltolkningar. Träning kan komma att bli viktigare i takt med att grammatiken expanderas. I allmänhet iakttogs dessa svagheter hos applikationen:

- Ord som låter ungefär likadant, till exempel thirteen och thirty, sixteen och sixty, kräver att man talar tydligt och aningen långsamt för goda resultat.
- En allmän svaghet när användaren talar fort eller otydligt, det vill säga, när användaren inte är informerad om vilka krav applikationen ställer på honom eller henne.
- En grammatik som är lite annorlunda jämfört med den standard som används, både när flygledare ger instruktioner till piloter, och när piloter upprepar dessa order.
- Applikationen uppvisade viss känslighet för buller. Detta kunde effektivt motverkas när mikrofonen byttes ut mot en bättre modell som var mer anpassad för taligenkänning.

Man gör iakttagandet att systemet uppvisar, i viss mån, alla de problem som listades i bakgrundsdelen. Dessa problem dras alla talsystem med i någon mån, och det är utvecklarens uppgift att minimera dem.

Det är intressant att notera att den användargrupp som är potentiella framtida användare är de som fick något av de sämsta resultaten, nämligen de som har tidigare erfarenhet av flygledning och/ eller rollen som pseudopilot. Detta ser jag som en kombination av flera faktorer, först och främst det faktum att jag lade märke till att de talade rejält mycket snabbare än de övriga testpersonerna. Också det faktum att den grammatik mitt program använder skiljer sig från det som är vedertaget, om än på vissa ställen så lite, var ett problem. Till exempel så uttalas vissa ord annorlunda inom flygledning, som three (uttalas *tree*) och nine (uttalas *niner.*)

Dessa problem är klart lösbara, grammatiken kan modifieras för att bättre efterlikna det språk som flygledare och piloter använder, så länge man är uppmärksam på att en mer extensiv grammatik förlorar en del av den naturliga resistans till små felsägningar som en mindre har. Eftersom grammatiken ändå måste expanderas om systemet ska ha någon som helst möjlighet att tas

i drift, så är detta ett av de naturliga stegen man kan ta i en vidare-utveckling av grammatiken.

Att systemet är känsligt för användare som talar fort är lite svårare att lösa programmeringsmässigt, utan det är något som eventuella användare själv får upptäcka och bedöma. Förhoppningsvis får man fortfarande en ökning i effektivitet jämfört med användande utav mus och tangentbord.

Buller-känsligheten var, med rätt utrustning, i princip minimal. Detta är högst relevant eftersom en simulation involverar att ett antal människor befinner sig i varandras omedelbara närhet, och då är det av vikt att man inte stör varandra.

5.2 Uppfyllda mål

I allmänhet anser jag mig ha uppnått de mål jag satte upp för mig själv i arbetes början. Även om den applikation jag producerat är långt ifrån kapabel att ersätta tangentbordet och musen för pseudopiloterna, så finns det möjlighet för att i alla fall vara ett komplement till dem. Grammatiken är tämligen skalbar, även om tiden får utvisa om jag lyckats nog inom det området. Det skulle inte förvåna mig om jag är tvungen att skriva om det åtminstone en gång till i takt med att den expanderar. Det är dock bara grammatiken som behöver expanderas vid detta tillfälle, tack vare SISR kan jag lämna programmet orört.

5.3 Problem under utvecklingens gång

Det som har tagit mest tid under utvecklingsfasens gång har avgjort varit skrivandet av grammatiken. Själva programmet blev funktionellt ganska tidigt, i och med att det mest rörde sig om att instantiera rätt klasser och hantera indatan på ett korrekt sätt. Grammatiken har skrivits om både en och tre gånger när jag upptäckt svagheter eller felaktigheter i den.

Innan jag blev medveten om SISR-standarden var mitt absolut största problem hur jag skulle kunna uppnå ett skalbart program om jag skulle vara tvungen modifiera det på flera olika ställen varje gång jag ville lägga till ett kommando.

Kommunikationen var ett problem som jag ägnade ganska mycket tid till. Först och främst skulle det hittas ett sätt på vilket NARSIM kunde ta emot de strängar jag genererade, och sedan skulle denna kommunikation implementeras. Efter att ha hittat en konsol som kunde hantera en Telnet-session blev Telnet det självklara valet. Att implementera en sådan klient var dock svårare än vad jag trodde, så till sist föll jag till föga och lånade ett bibliotek jag fann på internet[3].

5.4 Framtida arbete

En möjlig vidareutveckling är att låta applikationen ersätta pseudopiloten istället för att assistera honom/henne. I detta fall skulle flygledaren tala direkt till sina flygplanet istället för att gå via piloten. Detta skulle vara intressant i ett tidigt skede av en flygledares träning, då scenarierna är enklare och man inte förlorar så mycket realism om man tar bort den mänskliga piloten.

Något annat som jag själv gärna skulle förändra är det sätt som applikationen kommunicerar med NARSIM. Den Telnet-lösning som jag har just nu är tämligen *ad hoc*, om ändå en väl fungerade sådan.

I nuläget läggs den mesta tiden på att expandera grammatiken med ytterligare kommandon, till exempel kommandon för rörelse på marken, men även på att kontinuerligt uppdatera de redan färdiga delarna utav grammatiken.

5.5 Slutsats

Som nämndes under ”**5.2 Uppfyllda mål**” så anser jag mig nöjd med det som min applikation presterat i de tester jag kört. Även om den är långt ifrån att kunna tävla med simulator-modulerna som redan finns på marknaden, så har den potential att kunna göra det, särskilt eftersom den kan skräddarsys till NARSIM-miljön.

Med de lättillgängliga verktyg som Microsoft tillhandahåller är det tämligen lätt att producera en applikation som fungerar väl, och som kan fungera som bas för en fortsatt utveckling. Den relativa effektivitet som mitt program uppvisat med tämligen enkla medel får mig att tro att taligenkänning med stor sannolikhet kan ersätta eller tjäna som komplement till de aningen ålderstigna tangentborden och datormössen.

Litteraturförteckning

[1] Nugues P.M. (Springer, 2006) An Introduction to Language Processing with Perl and Prolog.

[2] Air Traffic Server Documentation. Tillhandahållen av LFV, framtagen av NLR

[3] Klaus W. Basan (2009-01-12) C# Telnet Client.
http://www.klausbasan.de/misc/telnet/index.html

[4] Speech Recognition Grammar Specification Version 1.0 (2008-12-02)
http://www.w3.org/TR/speech-grammar/#ref-sem

[5] Semantic Interpretation for Speech Recognition (SISR) Version 1.0. (2008-12-02).
http://www.w3.org/TR/2007/REC-semantic-interpretation-20070405/#SI3.2.2

[6] En.wikipedia.org. (2008-02-23). Speech Recognition
http://en.wikipedia.org/wiki/Speech_recognition.

[7] XML Speech Grammar Format. (2009-01-12).
http://cafe.bevocal.com/docs/grammar/xml.html

[8] Microsoft Developer Network.
http://msdn.microsoft.com/en-us/library
Grammar XML
http://msdn.microsoft.com/en-us/library/ms870140.aspx
Using the <tag> Element.
http://msdn.microsoft.com/en-us/library/ms870077.aspx
Serialization of Developer-defined Rule Variable Properties
http://msdn.microsoft.com/en-us/library/ms870096.aspx
Grammar Example: Solitaire
http://msdn.microsoft.com/en-us/library/ms870147.aspx
Serialization of Multiple Rule Grammars
http://msdn.microsoft.com/en-us/library/ms870098.aspx

[9] En.wikipedia.org (2008-11-04) Speech Recognition in Linux
http://en.wikipedia.org/wiki/Speech_recognition_in_Linux

[10] http://tldp.org. (2008-11-04). Speech Recognition Software
http://tldp.org/HOWTO/Speech-Recognition-HOWTO/software.html

[11] En.wikipedia.org (2008-12-01). Speech Recognition Grammar Specification
http://en.wikipedia.org/wiki/SRGS

[12] Supremis ATCC LTD (2009-04-06)
http://79.170.45.25/supremis/

[13] Englund C. Speech recognition in the JAS 39 Gripen aircraft - adaptation to speech at different G-loads (2004)
http://www.speech.kth.se/prod/publications/files/1664.pdf

[14] NARSIM home page (2008-11-04)
www.narsim.org

[15] En.wikipedia.org (2009-02-23). Windows Speech Recognition
http://en.wikipedia.org/wiki/Windows_Speech_Recognition

[16] microsoft.com (2009-02-23), Speech Recognition in Windows Vista
http://www.microsoft.com/enable/products/windowsvista/speech.aspx

[17] Sourceforge.net (2008-10-01). jvoicexml – Open Source VoiceXML Interpreter
http://jvoicexml.sourceforge.net/

[18] Carnegie Mellon University (2008-10-01). Vocalocity's OpenVXI 3.0
http://www.speech.cs.cmu.edu/openvxi/index.html

[19] microsoft.com (2008-10-01) - Microsoft Speech API (SAPI) 5.3
http://msdn.microsoft.com/en-us/library/ms723627(VS.85).aspx

[20] Readings in Speech Recognition (Morgan Kaufmann; 1 edition 1990)

Bilaga A – testresultat

Test med tränad profil (x indikerar lyckat igenkännande, - misslyckat)

	Fras 1	Fras 2	Fras 3	Fras 4	Fras 5	Fras 6	Fras 7	Fras 8	Fras 9	Fras 10
Person 1	x	x	x	x	x	x	x	x	x	x
Person 2	x	x	x	x	-	-	-	x	x	-
Person 3	x	-	x	x	x	x	-	x	-	-
Person 4	x	x	x	x	x	x	x	x	x	x
Person 5	x	x	-	x	x	x	x	x	x	x
Person 6	x	x	x	x	x	-	x	x	x	x
Person 7	x	x	x	x	-	-	x	x	-	x
Person 8	x	x	x	x	x	x	x	x	x	x
Person 9	x	x	x	x	-	x	x	x	x	x
Person 10	x	x	x	x	x	x	x	-	-	x

	Fras 1	Fras 2	Fras 3	Fras 4	Fras 5	Fras 6	Fras 7	Fras 8	Fras 9	Fras 10
Person 11	x	x	-	x	-	x	-	x	x	x
Person 12	x	x	x	x	x	x	x	x	x	x
Person 13	x	x	x	x	x	-	x	-	x	x
Person 14	x	x	-	x	x	x	x	x	x	x
Person 15	x	x	x	x	-	x	x	x	x	x
Person 16	x	x	x	x	x	-	x	x	x	x
Person 17	x	-	x	x	x	x	x	x	-	x
Person 18	x	x	x	x	x	x	x	x	x	x
Person 19	x	x	x	x	-	x	x	x	-	x
Person 20	x	x	x	x	x	x	-	x	x	x

Test med otränad profil (x indikerar lyckat igenkännande, - misslyckat)

Pe/Fr	Fras 1	Fras 2	Fras 3	Fras 4	Fras 5	Fras 6	Fras 7	Fras 8	Fras 9	Fras 10
Person 1	x	x	x	x	x	x	x	x	x	x
Person 2	-	-	x	x	x	x	x	x	x	-
Person 3	x	-	-	x	x	x	x	x	x	-
Person 4	x	x	x	x	x	-	-	x	x	x
Person 5	-	-	x	-	-	x	x	x	x	x
Person 6	x	x	x	x	x	x	x	x	-	x
Person 7	x	x	x	x	x	x	x	-	x	x
Person 8	x	x	x	-	x	x	-	x	-	x
Person 9	x	x	x	x	x	x	-	x	-	x
Person 10	x	x	x	-	-	x	x	x	x	x

	Fras 1	Fras 2	Fras 3	Fras 4	Fras 5	Fras 6	Fras 7	Fras 8	Fras 9	Fras 10
Person 11	x	x	x	x	x	-	-	x	x	-
Person 12	x	x	x	x	-	-	x	x	-	x
Person 13	x	x	x	x	x	x	x	x	x	x
Person 14	x	x	x	x	x	x	x	x	x	x
Person 15	x	x	x	x	-	x	x	x	x	x
Person 16	-	x	x	x	-	x	x	x	x	x
Person 17	x	x	x	x	x	x	x	x	-	-
Person 18	x	-	x	x	x	x	x	x	x	x
Person 19	x	-	x	x	x	x	x	x	x	x
Person 20	-	x	x	x	-	-	x	x	x	-

testfraser:

1. Scandinavian two two five take off end
2. Scandinavian one three four resume own speed end
3. Air France one eight set landing gear down end
4. Air France three six three set flaps retracted end
5. Scandinavian nine nine five speed three five zero end
6. Scandinavian two two five speed point eight zero end
7. Scandinavian one three four set speed limit three hundred and fifty end
8. Air France one eight turn left heading one four zero end
9. Air France three six three altitude three thousand end
10. Scandinavian nine nine five turn right heading zero five zero end

Anmärkningar:

1: Man
2: Man – har flygledarutbildning, talar snabbt
3: Man – har flygledarutbildning, talar snabbt, Stockholmsdialekt
4: Man – talar dalmålsdialekt
5: Man – talar bred skånska
6: Kvinna
7: Kvinna
8: Man – talar småländska
9: Kvinna – talar engelska med polsk accent
10: Kvinna – talar brittisk engelska
11: Man
12: Man – kraftigt förkyld, rösten mörk
13: Man
14: Man - talar bred skånska
15: Man
16: Kvinna – talar göteborgska
17: Man – talar engelska med fransk accent
18: Kvinna - talar engelska med fransk accent
19: Man – talar engelska med rysk accent
20: Kvinna

Bilaga B – Grammatik

root → (callsign | command) “end”

callsign → callsign_letters callsign_numbers

command → simple_cmd | semi _complex_cmd | complex_cmd

simple_cmd → ”expedite climb” |
”expedite descent” |
”link” |
”resume own speed” |
”startup” |
”take off” |
”phase”

semi_complex_cmd → on_off_command | ret_ext_command | up_down_command

complex_cmd → int_param_command | deg_param_command | taxto_param_command

on_off_command → on_off_cmd_id on_off_param

ret_ext_command → ret_ext_cmd_id ret_ext_param

up_down_command → up_down_cmd_id up_down_param

int_param_command → int_param_cmd_id numbers_main

deg_param_command → deg_param_cmd_id degrees

taxto_param_command → taxto_param_cmd_id taxto_params

callsign_letters → ”Air France” |
”Scandinavian” |
”Speedbird”

callsign_numbers → integers_digits(1-4)

on_off_cmd_id → ”expedite climb” |
”expedite turn” |
”set half bank”

ret_ext_cmd_id → ”set spoilers” |
”set flaps”

up_down_cmd_id → ”set landing gear”

int_param_cmd_id →”speed” |
”set speed limit” |
”altitude” |
”flight level”

deg_param_cmd_id → **”turn left” |**
”turn right” |
”turn left heading” |
”turn right heading”

taxto_param_cmd_id → **”tax to” |**
”tax to add”

on_off_param → **”on” |**
”off” |
”free”

ret_ext_param → **”retracted” |**
”extended” |
”free”

up_down_param → **”up” |**
”down”|
”free”

taxto_params → ”holding” | (integers_digits | letters)+

phase_cmd → **”gate” |**
”startup” |
”pushback” |
”taxi” |
”lineup” |
”take off run”

integers_digits → **”one” |**
”two” |
(”three” | ”tree”) |
”four” |
”five” |
”six” |
”seven” |
”eight” |
(”nine” | ”niner”)

integers_teens → **”eleven” |**
”twelve” |
”thirteen” |
”fourteen” |

”fifteen” |

”sixteen” |

”seventeen” |

”eighteen” |

”nineteen” |

integers_tens → **”ten” |**

”twenty” |

”thirty” |

”forty” |

”fifty” |

”sixty” |

”seventy” |

”eighty” |

”ninety”

degrees → integers_digits(1-3)

dummy → NULL

numbers_main → **numbers_regular | numbers_n_digits | numbers_digit_ten_teen | speed_mach**

numbers_regular → sub_hundred_thousand | sub_thousand | sub_hundred

sub_hundred_thousand → sub_hundred ”thousand” (”and”* sub_thousand)*

sub_thousand → sub_hundred ”hundred” (”and”* sub_hundred)*

sub_hundred → ”zero” | integers_teens | integers_tens integers_digits* | integers_digits

numbers_n_digits → integers_digits+

numbers_digit_ten_teen → integers_digits (integers_tens | integers_teens)

speech_mach → ”point” integers_digits+

letters → **”alfa” |**

”bravo” |

”charlie” |

”delta” |

”echo” |

”foxtrot” |

”golf” |

”hotel” |

”india” |

”juliet” |

”kilo” |

”lima” |

”mike” |

”november” |

”oscar” |

”papa” |

”quebec” |

”romeo” |

”sierra” |

”tango” |

”uniform” |

”victor” |

”whiskey” |

”x-ray” |

”yankee” |

”zulu”