NTNU
Det skapende universitet

Turbokoding basert på parallell konkatenering av konvolusjonskoder

Christian Nordahl

Master i kommunikasjonsteknologi
Oppgaven levert: Juli 2008
Hovedveileder: Geir Egil Øien, IET
Biveileder(e): Per Arne Grotthing, STM Norway

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

I de senere år har Turbokoder blitt et begrep innenfor feilrettende koding. Turbokoding er en strukturert metode for å lage koder med lange kodeord, som likevel kan dekodes effektivt med akseptabel kompleksitet. Kjernen i turbokodingsprinsippet er en enkoder bestående av konkatenerte komponentkoder, konvolusjonskoder eller blokk-koder, adskilt av en interleaver. Dekodingen er iterativ med utveksling av soft-informasjon. Ytelsen ligger tett opp til grensene foreskrevet av Shannons informasjonsteori.

Oppgaven omhandler studier og simulering av parallelt konkatenerte konvolusjonskoder. Utgangspunktet skal være gjeldene kanalkode for DVB-RCS (ETSI). Det skal utvikles effektive simuleringsprogrammer som muliggjør enkoding og dekoding av denne koden. Ytelsen i form av pakkefeilrate over en kanal med additiv hvit støy skal simuleres for forskjellige koderater og verifiseres mot publiserte resultater. Koden skal videre utvides i kompleksitet i henhold til prinsippet for TurboPhi, som er en aktuell kodekandidat for neste generasjon DVB-RCS. Pakkefeilrate for den utvidete koden skal simuleres for forskjellige blokklengder og koderater.

Oppgaven gitt: 27. januar 2008
Hovedveileder: Geir Egil Øien, IET

Sammendrag

Turbokoder er en familie av feilkorrigerende koder som først og fremst er kjent for å kunne opprettholde god ytelse ved svært lave signal-til-støy forhold. Denne masteroppgaven tar utgangspunkt i prosjektoppgaven [1], og har som mål å gjennomføre simuleringer av turbokoden i DVB-RCS og forslaget til utvidelse av denne, TurboΦ. Resultatene presenteres som plott av bit- og pakkefeilrater. Ettersom denne oppgaven er en ren videreføring av litteraturstudiet i prosjektoppgaven, og for å kunne leses som en uavhengig rapport, henter den stoff fra prosjektet der dette er nødvendig.

Rapporten presenterer først relevant bakgrunnsteori, før strukturen i den opprinnelige turbokoden gjennomgås. Den består av to parallellkonkatenerte RSC-enkodere, og dekodes iterativt av to parallelle SISO-moduler. Parallellene er sammenkoblet via en interleaver. Videre forklares turbokodene i DVB-RCS og TurboΦ, med vekt på forskjellene fra den originale koden. Disse kodene er blant annet tilpasset ikke-binære inngangssymboler, og endringene i komponentenkoderene, interleaveren og SISO-modulene forklares. Til slutt utledes optimale og sub-optimale dekodingsalgoritmer, med ekstra fokus på algoritmen som skal implementeres, Max-Log-MAP. Ytelsesforskjellen mellom denne og den optimale algoritmen utgjør kun 0,5 dB i signal-til-støy forhold.

Med dette som bakteppe presenteres modellen for implementasjonen, som skal programmeres i en kombinasjon av Matlab og C. Den generelle signalgangen gjennomgås, test-parametre forklares og forventede resultater drøftes. Simulering innebærer prinsipielt at det gjøres en viss grad av forenkling, så det gjøres rede for hvilke begrensninger og forenklinger som gjelder for denne modellen. Modellen blir også verifisert.

Avslutningsvis drøftes resultater fra simuleringene. Implementasjonen har i prinsippet vært vellykket: Turbokoden i DVB-RCS har gitt gode resultater, og TurboΦ oppnår mindre utflating av ytelseskurvene samt lavere pakkefeilrater for en del blokklengder og rater. Allikevel er resultatene noe varierende for TurboΦ, og innfrir ikke helt sammenlignet med tidligere publiserte resultater. Dette skyldes til dels lav testpopulasjon, men det er ikke den viktigste årsaken; det konkluderes nemlig med at mangel på gode parametre for interleaveren i TurboΦ er hovedårsaken til at den gode ytelsen uteblir. Masteroppgaven avsluttes derfor med forslag til videre raffinering av implementasjonen, spesielt med tanke på å skaffe til veie gode interleaver-parametre.

Forord

Denne masteroppgaven avslutter mine fem år som student på sivilingeniørstudiet Kommunikasjonsteknologi, på studieretningen Digital Kommunikasjon. Studiet er underlagt Institutt for Elektronikk og Telekommunikasjon ved Norges Teknisk-Naturvitenskapelige Universitet.

Oppgaven er en videreføring av prosjektoppgaven fra i høst, som var et litteraturstudium av turbokoden benyttet i DVB-RCS, samt forslaget til utvidelse av denne, TurboΦ. I denne oppgaven vil jeg i hovedsak gjennomføre implementasjon og simulering av turbokodene fra prosjektet.

Det har vært en utfordrende og lærerik prosess, og dette er garantert den mest omfattende oppgaven jeg har gitt meg i kast med hittil. Det praktiske aspektet ved oppgaven har vært inspirerende, selv om det naturlig nok også har vært kilde til en del hodebry.

Per Arne Grotthing fra STM Norway har veiledet oppgaven, og faglærer har vært Prof. Geir E. Øien, slik de også var for prosjektet. Jeg vil nok en gang takke begge for god dialog, nyttige innspill og tips underveis. Jeg vil også takke Øyvind Lensjø for hjelp til korrekturlesing og konstruktiv tilbakemelding.

Mest av alt vil jeg takke min kjæreste, Ingvild T. Voldhaug, for solid, vedvarende og kjærkommen støtte, og for aldri å ha mistet troen på meg selv når det har stått på som verst.

Drammen, 28/06-2008

Christian Nordahl

Innhold

Figurer

Kapittel 1

Introduksjon

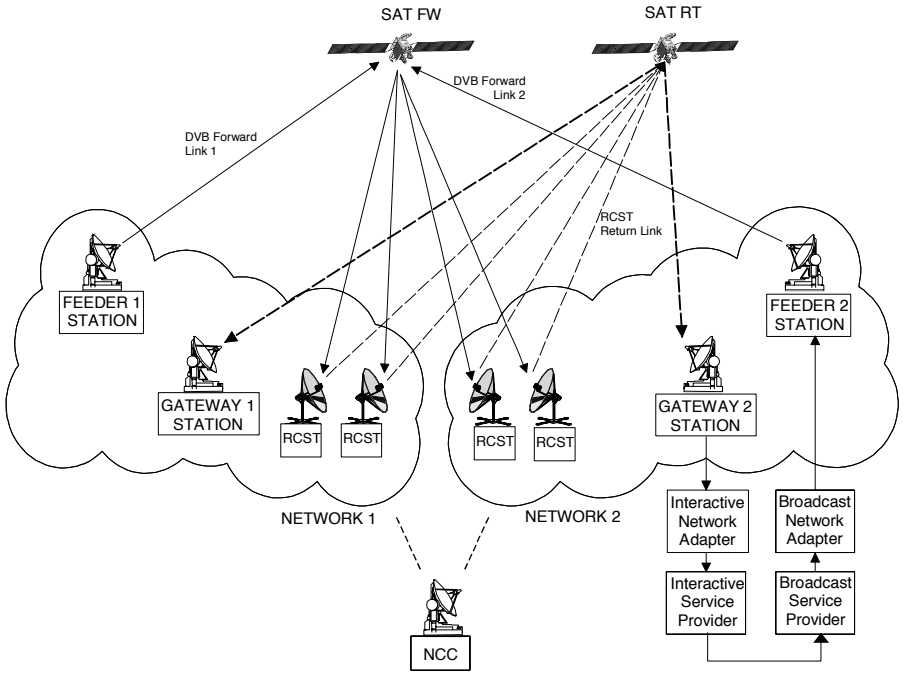
1.1 Bakgrunn

Satellittbaserte kommunikasjonssystemer og tjenester er idag i utstrakt bruk, både i områder der det ikke er mulig å sette opp fastlinjer, og som en konkurrent til tjenester levert over fastlinje. En vesentlig detalj ved et satellittbasert kommunikasjonssystem er kanalkoden. Det er først og fremst denne som bestemmer hvordan ytelsen til systemet påvirkes av transmisjonsstøy, og det er naturligvis ønskelig med høy gevinst i forhold til ukodet modulasjon.

Den totale ytelsen til et kommunikasjonssystem er underlagt mange, og til dels motstridende, begrensinger og krav. Noen av de viktigste er effektbegrensninger, tilgjengelig båndbredde og kompleksitetsbegrensninger, både som følge av sanntidsbetraktninger og at systemet må være fysisk realiserbart. Økonomi og lønnsomhet vil alltid være en viktig faktor, som i tillegg er nært knyttet til de fleste andre parametrene. Det endelige systemet fremkommer som et kompromiss mellom alle disse, og det er derfor fordelaktig med en kanalkode som gir best mulig ytelse, på minst mulig bekostning av de andre faktorene.

Kanalkodens ytelse er øvre begrenset av kanalkapasiteten, definert av Shannon i hans historiske artikkel fra 1948 [6]. Koder som *oppnådde* denne ytelsen var dog lenge ansett som en utopi. Derfor var det intet mindre enn en revolusjon da Berrou et al. i 1993 presenterte en kanalkode kalt *turbokode* [7], som bare var en brøkdel av en desibel unna den teoretiske grensen definert av Shannon 45 år tidligere. Denne fantastiske ytelse gjorde umiddelbart turbokode til en attraktiv kanalkode, og idag brukes forskjellige turbokoder blant annet i WiMax [8], 3G [9], satellittbaserte systemer som SMART-1 [3] og ikke minst i Digital Video Broadcasting - Return Channel Satellite (DVB-RCS) [2].

DVB-RCS er standardisert i ETSI EN 301 790 [2] og spesifiserer returkanalen i et nettverk basert på GEO-satellitter. Sluttbrukers terminal mottar IP-trafikk på foroverlinken i henhold til DVB-S [10] eller DVB-S2 [11], og sender feedback til gatewayen/huben via en satellittlink. Systemet vises i grove trekk i figur 1.1. Satellittlinkene kan utvise svært lav *signal to noise ratio* (SNR), blant annet fordi den lange transmisjonsveien medfører sterk nedgang i mottatt signaleffekt, og fordi linkbudsjettet som regel har mindre marginer å gå på. Slike potensielt dårlige SNR-forhold nødvendiggjør at de feilkorrigerende egenskapene til kanalkoden opprettholdes også ved lav SNR, og det er derfor forståelig at turbokode er et av alternativene til kanalkoden på returlinken.



Figur 1.1: Referansemodell for Satellite Interactive Network. DVB-RCS spesifiserer kommunikasjonen over linkene markert med "RCST Return Link". [2]

1.2 Målsetning og motivasjon

Oppgaven bygger videre på litteraturstudiet fra prosjektoppgaven [1], og omfatter implementasjon og ytelsessammenligninger av parallelt konkatenerte konvolusjonskoder. Utgangspunktet er gjeldene kanalkode for DVB-RCS [12]. Det skal utvikles effektive simuleringsprogrammer som muliggjør enkoding og dekoding av denne koden. Ytelsen i form av *frame error rate* (FER) over en kanal med *additive white gaussian noise* (AWGN) skal simuleres for forskjellige koderater og verifiseres mot publiserte resultater. Koden skal videre utvides i kompleksitet i henhold til prinsippet for TurboΦ, som er en aktuell kodekandidat for neste generasjon DVB-RCS. Pakkefeilrater for den utvidede koden skal simuleres for forskjellige blokklengder og koderater.

Oppgaven er foreslått av STM Norway (tidligere Nera Broadband Satellite), som er en betydelig aktør innen bredbåndskommunikasjon over satelitt. R&D-avdelingen til STM Norway jobber med videreutvikling av Satlink-produktet, som DVB-RCS og turbokoder er en sentral del av. Tidligere har de brukt komponenter med ferdigimplementert turbokode, men de ønsker nå å se på mulighetene for å implementere noe av dette selv. I tillegg utviser den nåværende turbokoden i DVB-RCS et feilgulv ved lave (FER)-verdier ($10^{-7} < FER < 10^{-4}$), i form av at ytelseskurven flater ut. Det samme gjelder for lave *bit error rate* (BER)-verdier. Ettersom TurboΦ viser mindre tegn til slike feilgulv, er den en spesielt interessant kodekandidat.

1.3 Omfang og forenklinger

Formålet med denne oppgaven er å sammenligne TurboΦ med den eksisterende turbokoden i DVB-RCS, primært ved implementasjon og simulering av de to turbokodene. DVB-RCS har en

rekke parametre og krav i tillegg til kanalkoden, som for eksempel forskjellige pakketyper for kontroll- og nyttetrafikk, mulighet for feildetekterende *cyclic redundancy check* (CRC) koding, krav om randomisering av input ved hjelp av linjekoder, muligheter for å justere modulasjonen, krav om omrokkering internt i blokkene før de sendes på kanalen, osv. Siden en sammenligning av de to kodene er hovedfokus for oppgaven, faller det meste av dette utenfor dens omfang. De fleste publiserte resultatene antar kun en minneløs kanal med additiv hvit støy, og det er naturlig å følge deres eksempel. System-modellen holdes derfor så enkel som mulig, og det fokuseres heller på effektive implementasjoner av turbokoden i DVB-RCS og TurboΦ. Dette drøftes videre i kapittel 4.

1.4 Fremgangsmåte og implementasjon

Det vil bli laget et rammeverk i Matlab, og simulering av både enkoder- og dekoderfunksjonene til den eksisterende turbokoden vil først bli implementert i dette språket. På grunn av dekoderfunksjonens kompleksitet og kjøretid er det ønskelig med varianter kodet i C, siden dette ligger nærmere maskinkode og stort sett gir bedre hastigheter. Etter en verifikasjon av Matlab-modellen vil denne overføres til C og tilføres visse utvidelser. Om denne versjonen viser seg å yte opp mot tidligere publiserte resultater, vil den være utgangspunktet for utvidelse til TurboΦ. I forhold til plotting av resultatene virker det fornuftig å behandle resultatene i Matlab, derfor vil det også implementeres funksjoner for å fasilitere overgangen mellom de to språkene. Det eksisterer både rammeverk og god dokumentasjon for dette formålet, og det forventes ikke at dette skal by på problemer. Dermed kan funksjonskall fra Matlab videreføres til de mer effektive funksjonene i C, som etter endt kjøring returnerer resultatene til Matlab.

Sammenligninger vil hovedsaklig presenteres i form av FER- og BER-plott som en funksjon av SNR, og vil vise simuleringer for forskjellige blokklengder og rater. Resultatene vil bli diskutert og satt i sammenheng med forskjellene mellom de to kodene.

1.5 Rapportens struktur

Kapittel 2 presenterer den grunnleggende teorien for oppgaven. Konvolusjonskoder blir forklart, i tillegg til detaljer som MHD, soft desisjon og Viterbi-algoritmen.
Kapittel 3 tar for seg turbokodene som er relevant for denne masteroppgaven. Først gjennomgås den originale turbokoden [7, 13] og dens forskjellige komponenter. Deretter gis det en kort introduksjon av DVB-RCS [2], før turbokoden i standarden presenteres. TurboΦ er et forslag til en utvidelse av turbokoden i DVB-RCS, som også legges frem. Til slutt i kapitlet vil de aktuelle dekodingsalgoritmene bli nøye gjennomgått.
Kapittel 4 presenterer de implementerte system-modellene. Disse verifiseres før deres overordnede signalgang gjennomgås. I tillegg vil kodens oppbygning og moduler bli forklart.
Kapittel 5 presenterer relevante resultater fra simuleringene og drøfter deres gyldighet. Det diskuteres også hvorvidt implementasjonen innfrir forventningene i form av ytelse.
Kapittel 6 gir en oppsummering av viktige resultater og erfaringer. Helt til slutt gis forslag til raffinering av modellen og videre arbeid.

Kapittel 2

Teori

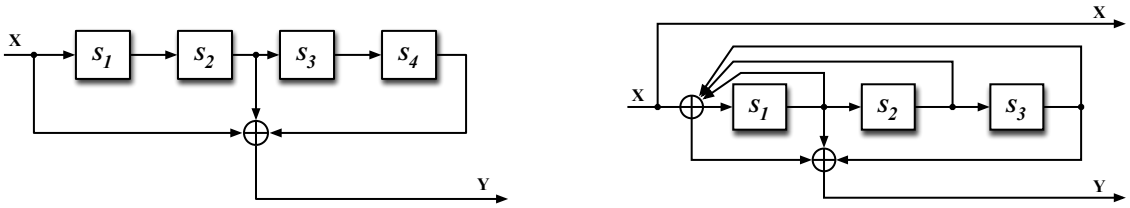
Formålet med denne oppgaven er implementasjon og sammenligning av to forskjellige turbokoder, men for å forstå gjennomgangen av turbokodene er det nødvendig å etablere noen grunnleggende begreper. Dette kapittelet presenterer derfor relevant bakgrunnsteori, og gir en kort gjennomgang av konvolusjonskoder og andre sentrale begrep basert på [14]. Det antas at leseren har forståelse for generell kodeteori.

2.1 Minimum Hamming Distance

Minimum Hamming distance (MHD) er et grunnleggende begrep for mye av det som gjennomgås i denne oppgaven, og bør derfor klargjøres med en gang. Kort forklart er MHD en verdi som representerer minimumsavstanden mellom kodeordene til en digital kode, i form av antall bit de avviker fra hverandre med. Desto større denne avstanden er, jo flere feil vil koden ha potensiale til å rette. Om det oppstår bitfeil og et kodeord som ikke finnes i koden blir mottatt, dekodes dette nemlig som det lovlige symbolet det har minst avstand til. Om færre bitfeil enn MHD/2 oppstår, vil det riktige symbolet fortsatt være det nærmeste, og symbolet dekodes korrekt. Økning av MHD gir derfor økning av feiltoleransen til koden.

2.2 Konvolusjonskoder

Man genererer en binær konvolusjonskode ved å sende k inngangsbit gjennom et lineært skiftregister med et endelig antall tilstander. Utvalgte posisjoner i registeret summeres modulo 2 i ett eller flere resultatbit, som representerer et symbol på totalt n utgangsbit. Lengden på registeret kalles inntrengningsdybden til koden, og betegner antall tidssteg ett enkelt inngangsbit kan påvirke utgangen. Inntrengingsdybden bestemmer også antall mulige tilstander. For et binært register med lengde K, vil koden ha 2^{K-1} forskjellige tilstander, og to mulige overganger fra hver tilstand. Derfor sier man også at kodens minne er gitt ved $\upsilon = K - 1$. Koderaten er lik k/n, og dersom ett av de n utgangsbitene er lik inngangsbitet, er koden *systematisk*. To vanlige typer konvolusjonskoder er *non-systematic convolutional* (NSC) og *recursive systematic convolutional* (RSC). Figur 2.1 viser eksempler på begge typer enkodere. Uttrykkene som definerer kodens rekursive og redundante summer kalles generatorer, og uttrykkes gjerne oktalt. De bestemmer altså



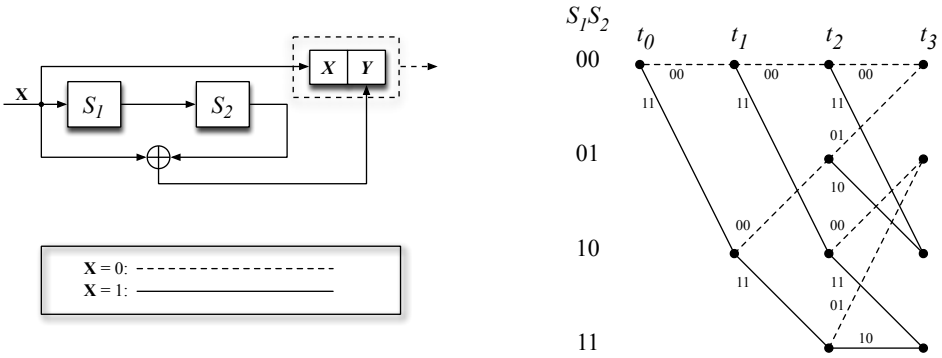
(a) NCS enkoder, K=5, k=1 og n=1

(b) RSC enkoder, K=4, k=1 og n=2

Figur 2.1: NSC- og RSC-enkoder

hvilke bit som skal brukes fra skiftregisteret i hvert tidssteg. Enkoderen i 2.1(a) har $G_Y = \{25\}_8$, mens 2.1(b) har $G_Y = \{15\}_8$ og $G_r = \{17\}_8$. Dette er lettere å se med de binære uttrykkene: Enkoderen i 2.1(b) har altså $G_Y = \{15\}_8$, som binært blir $G_Y = \{1\ 101\}_2$. Dette angir at redundansen skal summeres over inngangsbitet, det første og det siste bitet i skiftregisteret.

Konvolusjonskoden kan representeres ved et tilstandsdiagram der tilstanden tilsvarer de $K-1$ bitene i minnet, og overgangen til neste tilstand markeres med inngangssymbolet, samt utgangssymbolet det genererer. Det er allikevel mer vanlig å karakterisere tilstanden til koden og de forskjellige overgangene ved hjelp av et trellisdiagram som i figur 2.2. Registeret initialiseres med



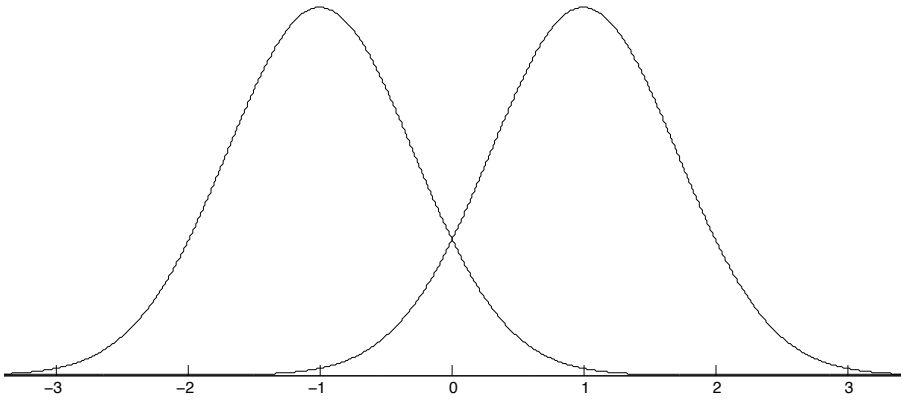
Figur 2.2: NCS-enkoder (K=3, k=1 og n=2) med tilhørende trellisdiagram

null-tilstanden i tidspunkt t_0. Tidspunkt t_n viser da alle tilstander det er mulig å nå i løpet av n tidssteg, med utgangspunkt i null-tilstanden. Overgangene representeres som grener mellom tilstandene, og merkes med utgangen. Når alle tilstander kan nås fra to av de foregående tilstandene, er trellisen for en binær konvolusjonskode i såkalt stasjonær tilstand. Dette vil generelt inntreffe i tid t_K for en trellis som starter i null-tilstanden i t_0. En mottatt symbolsekvens kan sees på som en sti av overganger gjennom trellisen.

2.3 Soft desisjon

Man kan bruke enten "harde" eller "softe" verdier i dekodingen. Ved *hard decision decoding* (HDD) demoduleres hvert mottatte bit enten som 0 eller 1, som deretter brukes i dekodingen. Med en *binary phase-shift keying* (BPSK)-modulasjon gitt av (1-2X_k) blir X_k=0/1 modulert som +1/-1, dekodes altså mottatte negative verdier som 1 mens positive verdier dekodes som 0. Problemet

med HDD er at informasjon som burde vært utnyttet går til spille, derfor er det som regel ønskelig å benytte *soft decision decoding* (SDD) istedet. SDD bruker kanalverdiene direkte i dekodingen, og siden amplituden til symbolet sier noe om hvor sikker verdien er, muliggjør dette utnyttelse av all tilgjengelig informasjon fra alle mottatte symboler. SDD er altså en kombinasjon av demodulasjon og dekoding. Fra figur 2.3 er det klart at et mottak av "3" er et langt sikrere tegn



Figur 2.3: BPSK-modulerte kanalsymboler (-1/+1) med normalfordelt støy

på at symbolet "1" ble sendt enn for eksempel mottak av "0.05" er, men med harde desisjoner ville de altså blitt tolket som like sannsynlige. Bruk av soft dekoding fører til store ytelsesforbedringer, og det er også det som blir brukt av dekoderene i turbokoden.

2.4 Dekoding av konvolusjonskoder

Dekoding av den binære konvolusjonskoden kan gjøres med *maximum likelihood* (ML) dekoding, som beregner den mest sannsynlige symbolsekvensen $\mathbf{C} = (C_0, ..., C_{L-1})$, gitt en mottatt symbolsekvens $\mathbf{R} = (R_0, ..., R_{L-1})$ med lengde L. Dersom hvert symbol består av n bit, kan sannsynligheten for én sti gjennom trellisen uttrykkes som

$$p(\mathbf{R}|\mathbf{C}) = \prod_{i=0}^{L-1} p(R_i|C_i) = \prod_{i=0}^{L-1} \prod_{j=1}^{n} p(R_{ij}|C_{ij}), \tag{2.1}$$

der C_i representerer den i'te grenen av $\mathbf{C}$, og C_{ij} representerer det j'te bitet av C_i. Det tilsvarende gjelder for R_i og R_{ij}. Ved å maksimalisere dette uttrykket over alle mulige symbolsekvenser $\mathbf{C}$, finner man den mest sannsynlige sekvensen. Maksimaliseringen gjøres gjerne på logaritmen av sannsynlighetene, da dette gir samme resultat, og letter utregningene. Dermed får man uttrykket

$$\log p(\mathbf{R}|\mathbf{C}) = \sum_{i=0}^{L-1} \log p(R_i|C_i) = \sum_{i=0}^{L-1} \sum_{j=1}^{n} \log p(R_{ij}|C_{ij}), \tag{2.2}$$

hvor $B_i = \sum_{j=1}^{n} \log p(R_{ij}|C_{ij})$ kalles *grenmetrikken*. Denne avhenger både av egenskapene til kanalen og modulasjonen man benytter. Ved å utnytte dette, samt at konstanter[1] og like skaleringsfaktorer ikke påvirker maksimeringen, kan man forenkle uttrykket. Dersom man for eksempel

[1] i forhold til variabelen det maksimeres over

antar en minneløs kanal med AWGN og BPSK-modulasjon, kan maksimalisering av (2.2) gjøres ved maksimalisering av det ekvivalente uttrykket

$$\sum_{i=0}^{L-1} \sum_{j=1}^{n} R_{ij}(1 - 2C_{ij})). \tag{2.3}$$

Vanskeligheten med ML-dekoding er at kompleksiteten av å regne ut (2.2) øker eksponensielt med minnet v, samtidig som utregningen må gjøres for hver eneste sti i trellisen. Neste avsnitt ser på en algoritme som reduserer denne kompleksiteten.

2.5 Viterbi-algoritmen

Viterbi-algoritmen [15] utnytter strukturen i trellisen til å senke både antallet metrikker som må regnes ut, og antall stier som må lagres underveis. Et viktig poeng den benytter seg av, er at dersom en av grenene *ut* av tilstand t_n er en del av den optimale stien, må nødvendigvis den beste grenen *inn* i tilstanden også være det. Derfor trenger man ikke ta vare på de andre, sub-optimale inngangssekvensene til tilstanden, og man har kun like mange overlevende, akkumulerte grenmetrikker som det er antall tilstander til enhver tid. Man må altså ta vare på 2^{K-1} forskjellige stier og deres akkumulerte metrikker, og i tillegg regne ut 2^K grenmetrikker for hvert tidspunkt. Kompleksiteten til Viterbi-algoritmen øker dermed eksponensielt med K, men den beregner kun de stiene den må. Viterbi kan benytte både hard eller soft input, og returnerer den mest sannsynlige symbolsekvensen. Den kan også modifiseres til å returnere soft output, i form av sannsynlighetsverdier for bitene.

Kapittel 3

Turbokode

Et generelt prinsipp for digitale koder er at økt kompleksitet en nødvendighet for bedre ytelse. Kompleksiteten legger dermed en begrensning på mulige ytelsesforbedringer, ettersom koden til slutt ikke vil være realiserbar. Dette gjelder også for turbokoder, men her er det først og fremst *designet* og *sammensetningen* av mer eller mindre komplekse komponenter som gir den gode ytelsen. Ved å finjustere de forskjellige komponentene i en turbokode kan man oppnå vesentlige ytelsesforbedringer, uten økt kompleksitet. Dette kapitlet har derfor et overordnet fokus på turbokodenes sammensetning, og hvilke faktorer som i så måte er kritiske for dens gode ytelse.

Først gjennomgås den opprinnelige turbokoden til Berrou et al. [7, 13][1]. Denne danner grunnlaget for de fleste turbokoder, så også for de aktuelle turbokodene i denne masteroppgaven. Gjennomgangen vil etablere de sentrale prinsippene i turbokoden, og vise hvordan disse gir den revolusjonerende ytelsen. Deretter presenteres turbokoden spesifisert i DVB-RCS, samt forslaget til en utvidelse av denne, TurboΦ. Dekodingen er den mest intrikate prosessen i en turbokode, derfor vies de aktuelle dekodingsalgoritmene ekstra oppmerksomhet i seksjon 3.4.

3.1 Den originale turbokoden

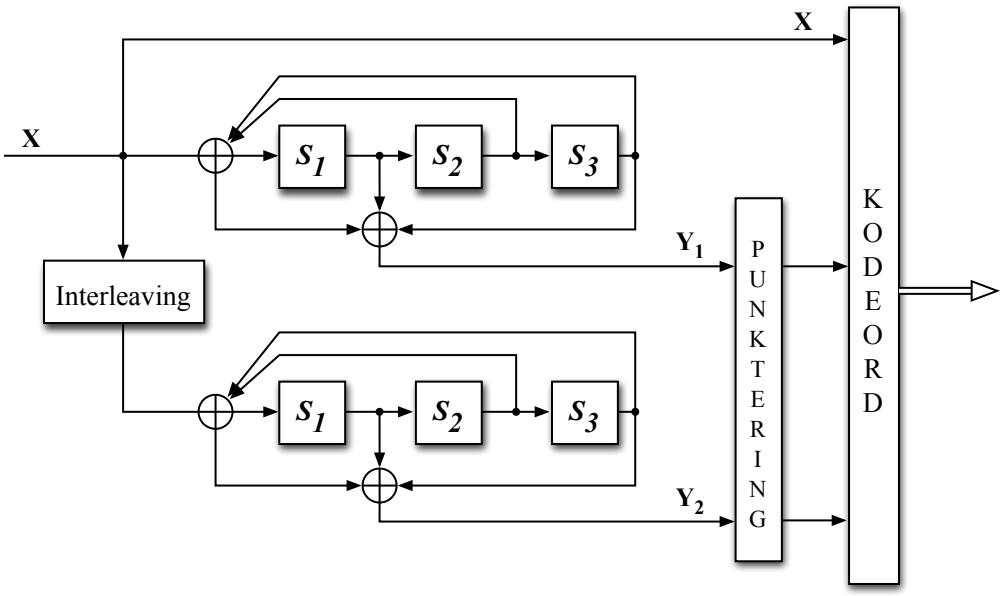
Den binære turbokoden som Berrou et al. presenterte i [7, 13] bygger på to grunnleggende prinsipper: parallellkonkatenert enkoding med to binære RCS komponentenkodere, og iterativ, "turbo" dekoding ved hjelp av to parallelle *soft-in/soft-out* (SISO) moduler. Parallellene sammenkobles ved hjelp av en interleaver, som innfører en pseudotilfeldig permutasjon.

3.1.1 Enkoding

Enkoderen er forholdsvis enkel, og består av to konkatenerte RSC-enkodere, C_1 og C_2, adskilt av en interleaver. Generatorene G_{ir}, G_{iY} og G_{iW} til de to komponentenkoderene trenger ikke å være parvis like, men det kan være en fordel om de velges slik, fordi enkoderen da kan klare seg med én RCS-enkoder implementert i hardware. Selv om denne rent fysisk opererer serielt på dataene, utgjør det fortsatt en parallellkonkatenert turbokode, så lenge systemklokka er høy nok til at

[1] Denne utledningen ble gjort i prosjektet [1] og hentes derfra, dog med noen tillegg og modifikasjoner. Utledningen av dekodingsalgoritmen gjøres nå i delkapitlet 3.4.1

komponentenkoderen rekker gjennom inngangssekvensen både i regulær og interleavet rekkefølge. En annen løsning vises i figur 3.1, der begge RCS-enkoderene implementeres i hardware. Denne har lavere krav til frekvensen på systemklokka, men krever til gjengjeld mer hardware. Hva man bør velge påvirkes av de andre kravene til systemet, men uansett produserer det altså identisk turbokode. Systemklokka til enkoderen har generelt en høyere frekvens enn dataraten inn i enkoderen [12], men må tilpasses raten slik at man er sikker på å unngå en flaskehals i systemet.



Figur 3.1: Turboenkoder med like komponentenkodere

Grunnraten for enkoderen i figur 3.1 er 1/3, men ved å utelate deler av redundansen, såkalt punktering, kan man oppnå høyere rater. Punkteringen sørger for at turbokoden kan operere med adaptive rater, og defineres i et sett av punkteringsmønstre tilsvarende de forskjellige ratene. Disse er definert på forhånd og ligger lagret i minnemoduler.

3.1.2 Interleaving

Interleaveren er selve nøkkelen til en god turbokode. Det er nemlig ikke MHD til kodeordene fra komponentenkoderene som først og fremst avgjør total MHD (og dermed også feilrettingspotensialet) til en inngangssekvens; dersom interleaveren konstrueres riktig er nemlig dens bidrag til MHD sterkt dominerende. Interleaveren består av en $M \cdot M$ matrise, der M bør være en potens av 2 og oppfylle $M \geq 2^K$. Regulær interleaving gjøres ved å skrive dataene inn i matrisen radvis, og lese de ut kolonnevis. Dette gir maksimal spredning av nærliggende bit, men er ikke spesielt godt egnet til turbokoder, som avsnittene under vil vise.

Et elementært *finite codeword* (FC) til en RSC-enkoder defineres som en utgangssekvens med en endelig avstand fra null-sekvensen (det vil si et endelig antall 1'ere i utgangssekvenser av X_k og Y_k). Et *globalt* FC er da en utgangssekvens av hele turbokoderen med en endelig avstand fra null-sekvensen (det vil si, et endelig antall 1'ere i utgangssekvenser av X_k, Y_{1k} og Y_{2k}). På grunn av rekursiviteten, er det ikke alle inngangssekvenser som vil være i stand til å generere disse

FC'ene. Det viser seg at FC-generende inngangssekvenser (FC-mønstre, også kalt *return to zero* (RTZ)-sekvenser [16]) må tilpasses G_Y og dens periode $p = 2^K - 1$.

Regulære interleavere har vist seg å være svært sårbare for sammensatte inngangssekvenser som er elementære FC-mønstre for både C_1 og C_2 [13]. For å unngå dette er man nødt til å innføre ikke-regulære operasjoner i interleaveren. Men, ved lange RTZ-sekvenser er det ønskelig med så regulære permutasjoner som mulig, fordi dette gir den største økningen i MHD [16]. En eksplisitt sammenheng mellom M og noen av de kortere RTZ-sekvensene vises i [13]. For å løse dette kom man frem til følgende to krav: For det første bør interleaveren spre inngangsdataene mest mulig. For det andre bør den gjøre generasjonen av redundansene så frakoblede som mulig, slik at om en avgjørelse fra C_1 avhenger av få bit i Y_1, vil den korresponderende avgjørelsen i C_2 avhenge av mange bit i Y_2. Det siste kravet kan være motstridende med det første, men vil muliggjøre en langt høyere minimumsavstand i turbokoden enn en regulær interleaver, fordi den er motstandsdyktig mot de nevnte feilmønstrene.

Permutasjonsregelen for den ikke-regulære interleaveren defineres som følger: Dersom (i,j) er (*rad,kolonne*)-adressen ved skriving, er (i_r, j_r) de tilsvarende adressene ved lesing. For en $M \cdot M$ interleaver der M oppfyller retningslinjene gitt over, vil i, j, i_r og j_r ha verdier mellom 0 og $M - 1$, og være gitt som

$$\begin{cases} i_r &= (M/2+1)(i+j) \quad mod(M) \\ \xi &= (i+j) \quad mod(8) \\ j_r &= [P(\xi)(j+1)] - 1 \quad mod(M) \end{cases} \tag{3.1}$$

Der P$(\cdot)$ blir relativt primsk med M. Siden den originale turbokoden fungerer som en konvolusjonskode og M således ikke må samsvare med bestemte blokklengder, kan den velges nokså fritt. I den originale artikkelen ble det for øvrig simulert med $M = 256$ [7].

3.1.3 Dekoding

Man antar en minneløs kanal med AWGN, der bitene moduleres med BPSK. Dekoding av turbokoder foregår iterativt, ved at to SISO-moduler beregner pålitelighetsverdier for inngangsdatene, som de utvekslerer deler av mellom hver iterasjon. Dersom permutasjonen i ligning (3.1) benyttes, vil det generelt være lav korrelasjon mellom inngangsdataene til de to modulene, og dette gir SISO'ene større potensiale til å utnytte informasjonen fra hverandre. Man har følgende signaler inn i selve dekoderen:

$$\begin{aligned} x_k &= (1 - 2X_k) + n_k \\ y_{1k} &= (1 - 2Y_{1k}) + q_{1k} \\ y_{2k} &= (1 - 2Y_{2k}) + q_{2k}, \end{aligned} \tag{3.2}$$

der Y_{1k} og Y_{2k} er redundansene generert henholdsvis fra den ikke-permuterte og den permuterte sekvensen. n_k, q_{1k} og q_{2k} er uavhengige, normalfordelte støykomponenter, alle med varians σ^2. SISO$_1$-modulen får $R_{1k} = (x_k, y_{1k})$ på inngangen, mens SISO$_2$ får den permuterte sekvensen og dens tilhørende redundans, $R_{2k} = (x_{2k}, y_{2k})$, der $x_{2k} = \Pi(x_k)$.

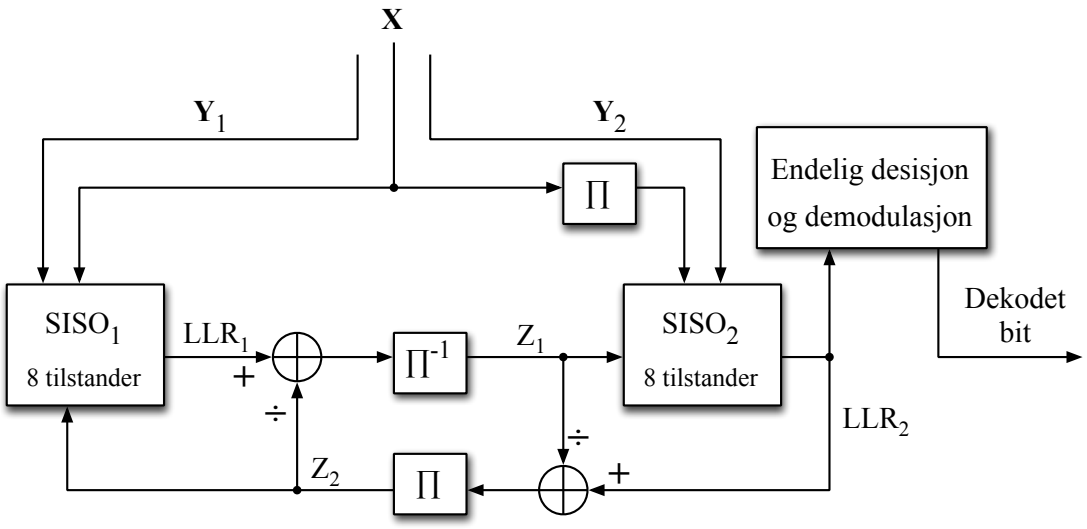
RCS-enkoderene i turboenkoderen vil oppføre seg som Markovkilder med 2^{K-1} tilstander, som ved mottak er påvirket av støy. Pålitelighetsverdien som SISO$_i$ regner ut kalles *log likelihood ratio* (LLR) og uttrykkes

$$L_i(d_k) = \log \frac{Pr(d_k = 1|observasjon)}{Pr(d_k = 0|observasjon)} \tag{3.3}$$

der $Pr(d_k = i|observasjon),\ i \in 0, 1$, er *a posteriori probability* (APP) til bitet d_k, altså sannsynligheten for at bit d_k er lik i, gitt all annen informasjon vi har tilgjengelig[2]. LLR'en består av to deler: En del avhenger av inngangssymbolet i tidspunkt k, men den andre fremkommer uavhengig av dette inngangssymbolet, og kalles *ekstrinsikk* informasjon. Den ekstrinsikke informasjonen er en funksjon av redundansen innført av enkoderen, som fremkommer ved å kombinere redundansen y_k med kjennskapen til RCS-enkoderens egenskaper, i form av kodetrellisen og overgangene den representerer.

Man har flere valgmuligheter når det gjelder dekodingsalgoritmen i SISO-modulen, men ikke alle er optimale i APP-forstand. Viterbi-algoritmen [15] gir for eksempel optimal dekoding av en *sekvens* fra en Markovkilde, men selv om den bruker soft-desisjoner underveis, er den ikke i stand til å gi oss APP-estimater for hvert enkelt bit, som er nødvendig for å gi optimal dekoding av *bitene*. For å oppnå optimal dekoding med hensyn på APP må SISO'ene bruke *maximum a posteriori* (MAP)-algoritmen, som beskrives i 3.4.1.

Iterativ dekoding



Figur 3.2: Iterativ turbodekoder

Optimal dekoding oppnås dersom MAP-algoritmen benyttes i begge SISO-modulene i turbodekoderen, samtidig som den ekstrinsikke informasjonen tilbakekobles. Dersom inngangene på modulen er uavhengige, kan LLR på utgangen generelt uttrykkes som en sum av uttrykk basert på hver av inngangene. Selv om det eksisterer sammenheng mellom x_k og den permuterte sekvensen $\Pi(x_k)$, er denne svak nok til at vi kan anse inngangen R_{ik} og den ekstrinsikke informasjonen fra den andre SISO-modulen som tilnærmet uavhengige[3]. I tillegg tilnærmes $z_{ik} = L_i^e(d_k)$ som en

[2]Først og fremst kjennskap til enkoderne og trellisen i kombinasjon med informasjonssekvensen og redundansen

[3]Etter hver iterasjon vil imidlertid korrelasjonen mellom hver av SISO-modulene øke, siden de "bruker opp" mer og mer av den redundante informasjonen. Antagelsen om uavhengighet blir dermed svakere, som medfører at ytelsesforbedringen mellom hver iterasjon avtar

Gaussisk variabel, og vi får denne formelen for LLR på utgangen av SISO-modulene:

$$L_1(d_k) = \frac{2}{\sigma^2}(x_k) + \frac{2}{\sigma_{z2}^2} z_{2k} + L_1^e(d_k)$$
$$L_2(d_k) = \frac{2}{\sigma^2}\Pi(x_k) + \frac{2}{\sigma_{z1}^2} z_{1k} + L_2^e(d_k). \tag{3.4}$$

Ved bitfeil har ofte $L^e(d_k)$ motsatt fortegn av $L(d_k)$, og kan derfor bidra til at sannsynlighetsverdien blir iterativt forbedret. Det samme gjelder ved korrekt mottatt bit, fordi $L^e(d_k)$ som regel da har samme fortegn som $L(d_k)$, og dermed forsterker sannsynlighetsverdien. Siden det er høy korrelasjon mellom inngang og utgang av SISO'ene, er det viktig at den ekstrinsikke informasjonen fra en modul ikke kommer inn på inngangen til den samme modulen, fordi dette vil skape en selvforsterkende feil. Derfor trekkes z_{1k} og z_{2k} fra inngangen på sine tilhørende SISO'er, som vist i figur 3.2.

3.1.4 Ytelse

Et viktig prinsipp i turbokoder er konkatenering av komponentenkodere med liten constraintlengde (og dermed også liten MHD). Dette høres kanskje merkelig ut, men poenget er at siden interleaveren står for nesten all økningen i MHD, vil høye MDH'er i komponentenkoderene bare medføre at det lettere oppstår fastlåste feilmønstre når det opereres på svært lav SNR. Dette er fordi en høyere MDH betyr at feilsekvensen også blir lengre, dersom det først gjøres en feil. Ved små MDH'er blir feilsekvensene kortere, og det er dermed lettere for den iterative dekoderen å oppdage feilen. Økningen i total MDH ved høyere inntrengningsdybder er heller ikke stor nok til å kompensere for denne ulempen, siden interleaveren allikevel vil stå for en betydelig større del av feilkorrigeringspotensialet. Som forfatterne i [13] nevner, er det langt ifra trivielt å finne gode, ikke-regulære permutasjoner som oppfyller begge kriteriene oppgitt i 3.1.2. De fant frem til permutasjonen i ligning (3.1) gjennom grundige empiriske forsøk.

Da denne turbokoden ble presentert i 1993, var ytelsen intet mindre enn enestående sammenlignet med datidens feilkorrigerende koder. Ved den laveste raten får de iterative dekoderne fullt utbytte av den redundante informasjonen, og man ender da opp kun 0,7 dB unna Shannons grenser (med en referanseverdi for $P_e = 0$ gitt av $P_e = 10^{-5}$). Riktignok var dette etter 18 iterasjoner med den optimale, mest komplekse dekodingsalgoritmen, i tillegg til at akkurat denne innstillingen var blitt ekstra finjustert med skalerings- og korreksjonsledd [7, 13]. Tidskritiske systemer legger som regel en øvre begrensning på maksimalt antall iterasjoner, og må kanskje begrense seg til sub-optimale dekodingsalgoritmer, som for eksempel Max-Log-MAP.

I etterkant var fagfeltet noe overrasket over at denne typen kode ikke hadde blitt oppdaget før, ettersom den var langt mindre kompleks enn man tidligere hadde antatt at en kapasitetsoppfyllende kode måtte være. En mulig forklaring er at mens de fleste andre søkte optimale koder, og dermed møtte tilsynelatende uoverkommelige kompleksitetshindre, forsøkte isteden Berrou, Glavieux og Thitimajshima å finne en kode som presterte *godt nok*. Én felles dekoder ville for eksempel kunne gi bedre ytelse enn de to iterative SISO'ene, men kompleksiteten forbundet med optimaliseringen av denne ville gjort den urealiserbar.

3.2 Turbokoden i DVB-RCS

I DVB-RCS har man to valgmuligheter for kanalkodingen. Man kan enten benytte konkatenering av Reed-Solomon og en indre konvolusjonskode, eller velge å kun bruke turbokode. I tillegg er det mulig å bruke en *cyclic redundancy check* (CRC)-kode før all annen kanalkoding, for å øke mulighetene for feildeteksjon. For noen av kombinasjonene av kanalkoding og bursttype er CRC-koden også påbudt. Modulasjonen over kanalen er *quadrature phase-shift keying* (QPSK).

Turbokoden i DVB-RCS er basert på koden definert av Berrou og Glavieux [13], og spesifiseres nærmere i [2, 12]. Koden skal kunne takle sju forskjellige rater: $R = 1/3$, $2/5$, $1/2$, $2/3$, $3/4$, $4/5$ og $6/7$. Disse realiseres ved hjelp av predefinerte punkteringsmønstre. To viktige forskjeller fra de tidligere turbokodene [7, 13] er at komponentenkoderne er *duo-binære* og *circularly recursive systematic convolutional* (CRSC). Sirkulære komponentenkodere medfører blant annet at turbokoden kan benyttes som en vanlig blokk-kode, i tillegg til at det gir bedre ytelse.

3.2.1 CRSC enkoding

Turbokoden i DVB-RCS benytter komponentenkodere med liten constraint-lengde, og er designet etter samme mal som i seksjon 3.1.1. Enkoderen består av to parallellkonkatenerte CRSC-komponentenkodere, C_1 og C_2, adskilt av en modifisert interleaver som bygger opp under den nye kodens egenskaper. Generatorene til de to enkoderene er like, og gis av $G_r = [15]_8$, $G_Y = [13]_8$ og $G_W = [11]_8$. Grunnraten er med andre ord $1/3$, mens punkteringsmønstrene for de 6 andre ratene bestemmes på forhånd og lagres i minnemoduler. Enkoderen vises i figur 3.3.

Turbokoden får inn blokker med k bit organisert i N symboler/par, der $k = 2N$ og hver blokk kodes selvstendig. Komponentenkoderene opererer på to og to bit, og kalles derfor duo-binære. Fordelene med slike komponentenkoderene diskuteres i seksjon 3.2.4.

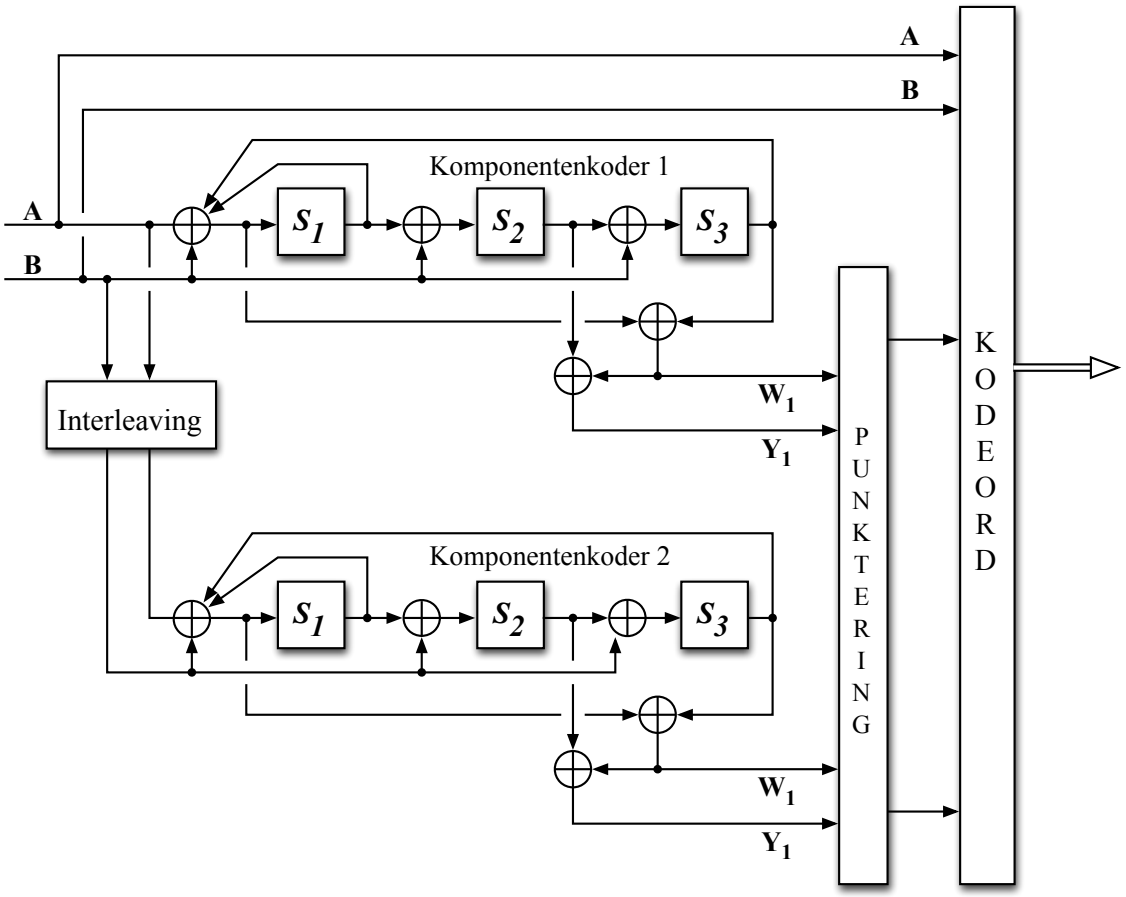
Mens turbokodene i [7, 13] fungerer som vanlige konvolusjonskoder, er turbokoden i DVB-RCS som nevnt sirkulær. Det innebærer at hver komponentenkoder først må gjøre en *prekoding*. Da initialiseres den med nulltilstanden $\mathbf{S}_0$, før sekvensen på inngangen kodes som vanlig. I prekodingen behøver man imidlertid bare den avsluttende tilstanden $\mathbf{S}_N^0$, så redundansen forkastes. $\mathbf{S}_N^0$ brukes deretter til å finne den såkalte *sirkulærtilstanden* $\mathbf{S}_C$. Den beregnes ved hjelp av

$$\mathbf{S}_C = \left[\mathbf{I} + \mathbf{G}^N\right]^{-1} \mathbf{S}_N^0, \tag{3.5}$$

der $\mathbf{G}$ er generatormatrisen for skiftregisteret. I praksis brukes et tabelloppslag basert på $\mathbf{S}_N^0$ og $N \ mod \ (2^\upsilon - 1)$ [2]. Ved enkodingen av sekvensen initialiseres hver komponentenkoder med sin tilhørende sirkulærtilstand, som de også ender opp i etter enkodingen. Trellisen kan dermed sees på som en roterende sylinder.

3.2.2 Interleaving

Som nevnt i seksjon 3.1.2 har interleaveren mye å si for MHD'en til turbokoden, og interleaveren i DVB-RCS representerer en klar forbedring i forhold til interleaveren fra seksjon 3.1.2. Ved å gjøre en permutasjon både innad i parene og parene seg imellom, oppnår man en bedre sikring mot sammensatte FC'er. Man kan altså utføre tilnærmet regulær interleaving på symbol-basis, som er best for lengre RTZ-sekvenser, og så interleave parene internt, for å øke motstandsdyktigheten



Figur 3.3: Enkoderen i DVB-RCS, med identiske komponentenkodere.

mot sammensatte FC'er. Har blokker med N symboler, permutasjonen $j = \Pi(i)$ for $i = 0, ..., N-1$ er da som følger:

Nivå 1:

$$(A_i, B_i) = (B_i, A_i) \quad \text{for} \quad i \bmod 2 = 0$$

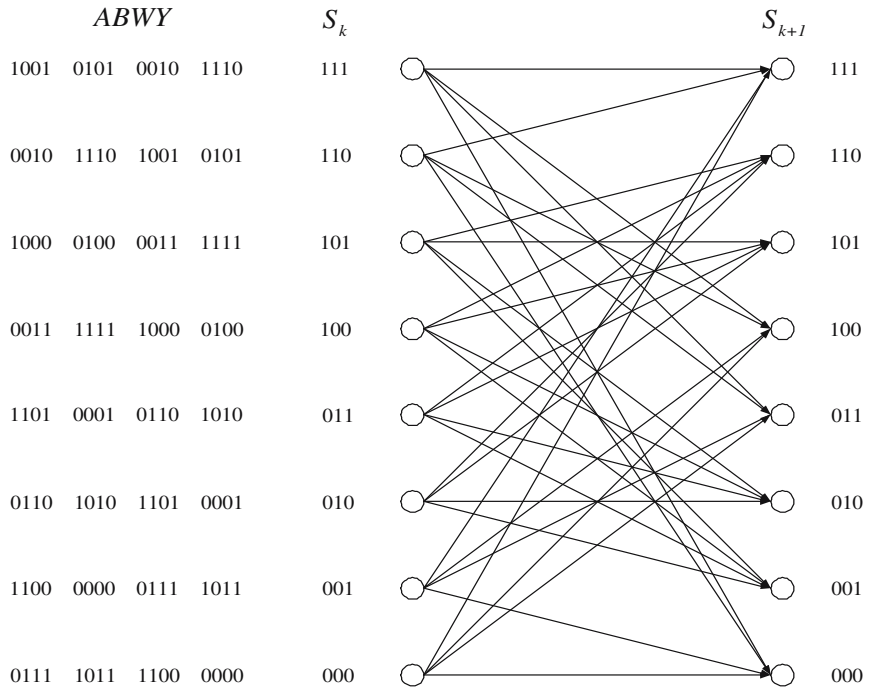
Nivå 2:

$$j = (i \cdot P_0 + P + 1) \bmod N, \quad \text{der} \quad \begin{cases} P = 0 & \text{for} \quad i \bmod N = 0 \\ P = N/2 + P_1 & \text{for} \quad i \bmod N = 1 \\ P = P_2 & \text{for} \quad i \bmod N = 2 \\ P = N/2 + P_3 & \text{for} \quad i \bmod N = 3, \end{cases}$$

der P_0, P_1, P_2 og P_3 er tilpasset forskjellige blokklengder, og blant annet kan finnes i [2].

3.2.3 Dekoding

Det er tre viktige endringer i forutsetningene for dekoderen i DVB-RCS, i forhold til turbokoden i [13]: Hvert av de mottatte symbolene består av to bit modulert over en QPSK-kanal, kompo-



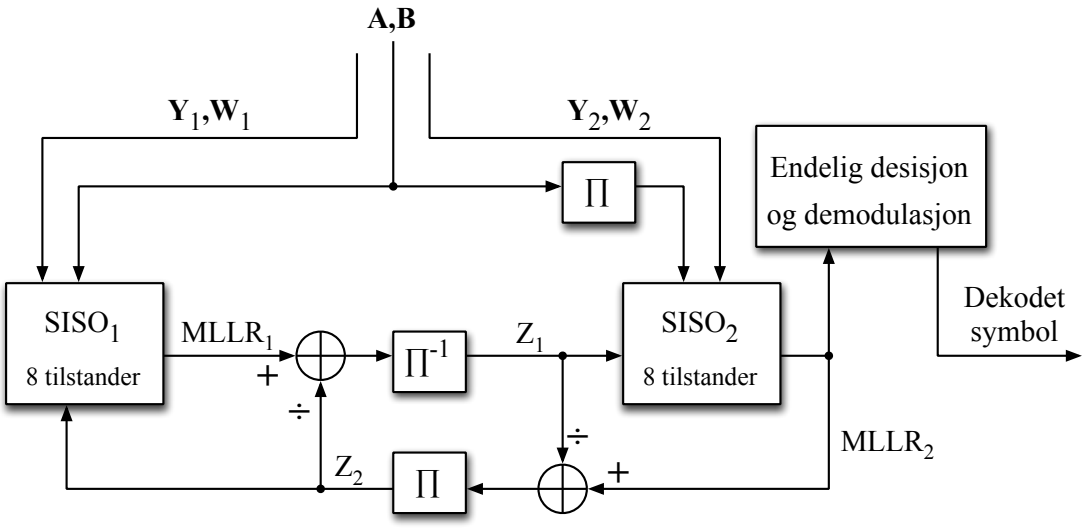
Figur 3.4: Trellis for én av komponentenkoderene i DVB-RCS. AB viser de 4 mulige inngangssymbolene, mens WY er de tilhørende redundanssymbolene på utgangen. ABWY betegner dermed overgangsgrenene fra hver tilstand i rekkefølgen øverst → nederst. [3]

nentenkoderene er duo-binære, og det brukes i tillegg prekoding for å gjøre komponentenkoderen sirkulær. En optimal algoritme tilpasset disse forutsetningene kan utledes etter samme prinsipp som i [13]. Siden koden er duo-binær har man nå fire mulige inngangssymboler istedenfor to, og man må regne ut en multidimensional LLR (MLLR). For å få med all relevant informasjon må denne inneholde tre distinkte LLR'er [17]. Den modifiserte dekoderen ser ut som i figur 3.5, der SISO'ene nå beregner MLLR.

Betegner de to systematiske bitene som $\boldsymbol{d}_k = (A_k, B_k)$. Symbolene ut fra enkoderen er gitt av $\mathbf{C}_1^N = (C_1, ..., C_k, ..., C_N)$, der $C_k = (A_k, B_k, Y_{1k}, Y_{2k}, W_{1k}, W_{2k})$. Hvert av disse bitene kan moduleres med QPSK etter regelen $d_{qpsk} = 1 - 2d_i;\ d_i = 1, 0$. De tilsvarende bitene mottatt over AWGN-kanalen blir da

$$\begin{aligned} a_k &= (1 - 2A_k) + n_{ak} \\ b_k &= (1 - 2B_k) + n_{bk} \\ y_{1k} &= (1 - 2Y_{1k}) + n_{y1k} \\ y_{2k} &= (1 - 2Y_{2k}) + n_{y2k} \\ w_{1k} &= (1 - 2W_{1k}) + n_{w1k} \\ w_{2k} &= (1 - 2W_{2k}) + n_{w2k}, \end{aligned} \tag{3.6}$$

og den mottatte sekvensen er nå $\mathbf{R}_1^N = (R_1, ..., R_k, ..., R_N)$, der $R_k = (a_k, b_k, y_{1k}, y_{2k}, w_{1k}, w_{2k})$. Dekoderen opererer også med bit-par istedenfor enkeltbit, men ellers er selve signalgangen i figur 3.5 lik som i figur 3.2. Som nevnt over må hver SISO nå regne ut en MLLR, $\mathbf{L}(\boldsymbol{d}_k) =$



Figur 3.5: Duo-binær dekoder med 8 tilstander

$[L_1(\boldsymbol{d}_k), L_2(\boldsymbol{d}_k), L_3(\boldsymbol{d}_k)]$. Et LLR-utvalg som dekker all informasjon er gitt ved

$$L_1(\boldsymbol{d}_k) = \log \frac{Pr(\boldsymbol{d}_k = (0,1)|\boldsymbol{R}_1^N)}{Pr(\boldsymbol{d}_k = (0,0)|\boldsymbol{R}_1^N)} \tag{3.7}$$

$$L_2(\boldsymbol{d}_k) = \log \frac{Pr(\boldsymbol{d}_k = (1,0)|\boldsymbol{R}_1^N)}{Pr(\boldsymbol{d}_k = (0,0)|\boldsymbol{R}_1^N)} \tag{3.8}$$

$$L_3(\boldsymbol{d}_k) = \log \frac{Pr(\boldsymbol{d}_k = (1,1)|\boldsymbol{R}_1^N)}{Pr(\boldsymbol{d}_k = (0,0)|\boldsymbol{R}_1^N)}. \tag{3.9}$$

Ved å sammenligne disse uttrykkene med (3.3), ser man at de er svært like, og faktisk kan hver av de kan regnes ut med for eksempel MAP-algoritmen. Den gir som nevnt optimal dekoding av konvolusjonskoder, men kompleksiteten kan være noe hindrende ved høye krav til sanntidsberegninger. Dersom dette er tilfellet, kan man for eksempel benytte forenklingen Max-Log-Map. Begge disse algoritmene er aktuelle for masteroppgaven, og utledes for duo-binære koder i 3.4. Utledningene tar utgangspunkt i resultatene fra seksjon 3.4.1.

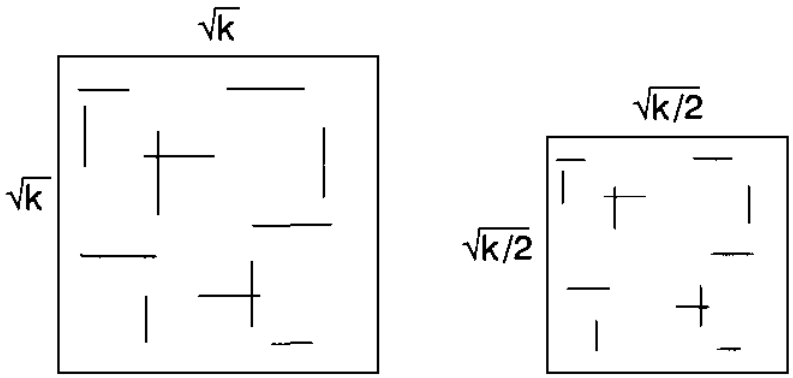
Iterativ dekoding

Følger samme prinsipp som i subseksjon 3.1.3: Optimal dekoding oppnås dersom MAP-algoritmen benyttes i begge SISO-modulene i turbodekoderen, samtidig som den ekstrinsikke informasjonen, som nå er flerdimensjonal, tilbakekobles. Dersom man bruker en sub-optimal dekodingsalgoritme, vil duo-binære koder degraderes i mindre grad enn binære, som er nok en grunn til å bruke ikke-binære koder.

3.2.4 Ytelse

Turbokoden i DVB-RCS er duo-binær, dette innebærer ytelsesforbedringer i forhold til binære koder: Gitt en binær og en duo-binær enkoder som begge skal kode totalt k bit, vil den duo-binære

oppleve kortere og færre fastlåste feilsekvenser. Dette fordi de tilhørende dekoderene bruker den samme mengden informasjon i dekodingen, tilsvarer det halvparten så mange transisjoner for den duo-binære koden, som igjen tilsvarer en halvering av lengden på feilstiene i matrisen [16]. Videre har interleaveren til den binære koden dimensjon $\sqrt{k} \times \sqrt{k}$, mens det bare er $N = k/2$ symboler som skal permuteres i det duo-binære tilfellet, slik at denne interleaveren får dimensjonen $\sqrt{k/2} \times \sqrt{k/2}$. Siden feilstiene halveres, mens matrisedimensjonen bare deles med $\sqrt{2}$, får vi en netto gevinst ved å benytte duo-binære koder. Figur 3.6 demonstrerer prinsippet for noen tilfeldige feilmønstre.



Figur 3.6: Feilstier i en binær kode med 2N tilstander, kontra en duo-binær kode med N tilstander. [4]

Gevinsten kommer i form av bedre konvergens i den iterative dekodingen, lavere feilgulv, større MHD'er, mindre sensitivitet for punktering, mindre latens og mindre degradering ved bruk av en sub-optimal dekodingsalgoritme, for å nevne det viktigste [4]. Duo-binære koder blir også mindre påvirket av usikkerhet rundt sirkulærtilstanden. Et aber er dog at en duo-binær kode har dobbelt så mange overganger mellom tilstandene, og derfor krever økt kompleksitet i dekoderen.

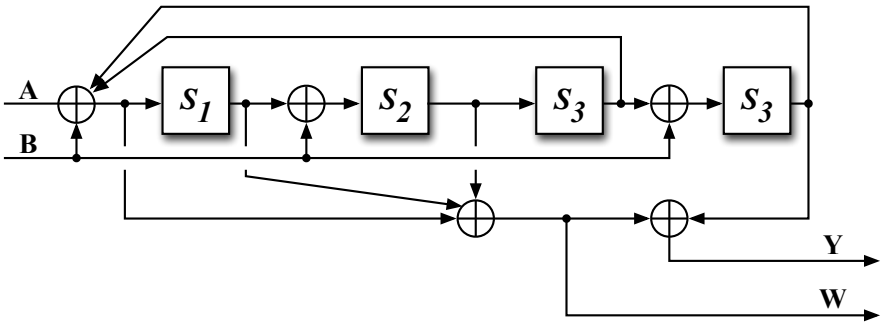
3.3 TurboΦ

TurboΦ er en kodekandidat for neste generasjon DVB-RCS. Hovedsakelig går den ut på en tilstandsutvidelse og modifikasjon av den eksisterende turbokoden [12]. Utvidelsen innebærer at inntrengningsdybden til enkoderne økes fra $K = 4$ til $K = 5$, som også betyr at antallet tilstander dobles fra 8 til 16.

Benedetto et al. mener at TurboΦ kan gi en ytelsesforbedring i SNR på 1,5 dB ved FER på 10^{-6} [5], sammenlignet med den gamle turbokoden. Den kraftige forbedringen skyldes først og fremst at TurboΦ i liten grad opplever det samme feilgulvet som den gamle koden, og da er det naturligvis mye å hente ved FER=10^{-6}. Den nye CRSC-komponentenkoderen vises i figur 3.3.

3.3.1 Permutasjonen i TurboΦ

Baseres på *almost regular permutation* (ARP) [16], og jobber i likhet med interleaveren i DVB-RCS på to nivåer. Har blokker med N symboler, permutasjonen $j = \Pi(i)$ er da som følger for $i = 0, ..., N - 1$:



Figur 3.7: En av CRSC komponentenkoderene i TurboΦ, med to redundansbit.

Nivå 1:

$$(A_i, B_i) = (B_i, A_i) \quad \text{for} \quad i \bmod 2 = 0$$

Nivå 2:

$$j = \Pi(i) = (i \cdot P + Q + 3) \bmod N, \text{ der } \begin{cases} Q = 0 & \text{for} \quad i \bmod 4 = 0 \\ Q = 4Q_1 & \text{for} \quad i \bmod 4 = 1 \\ Q = 4P + 4(Q_2 + 1) & \text{for} \quad i \bmod 4 = 2 \\ Q = 4P + 4(Q_3 + 1) & \text{for} \quad i \bmod 4 = 3, \end{cases}$$

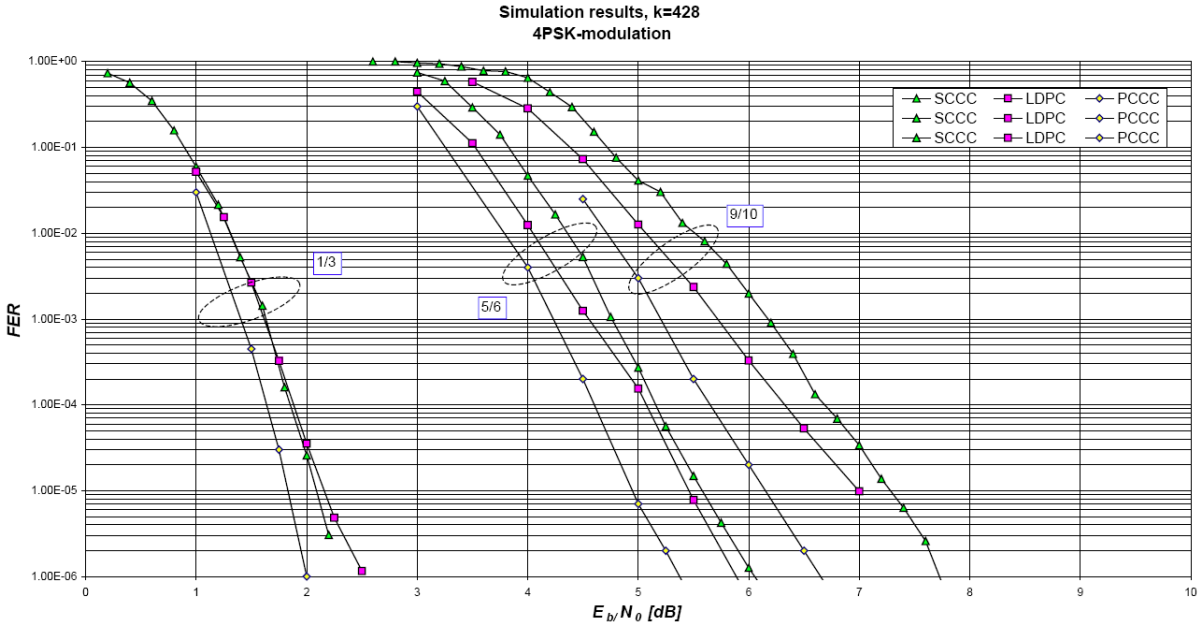
der P er et heltall relativt primsk med N, og Q_1, Q_2 og Q_3 er små heltall fra 0 til 8. Disse parametrene må bestemmes for hver blokklengde, og gode verdier må finnes gjennom empiriske forsøk.

Siden TurboΦ kun er en utvidelse av en velkjent og godt dokumentert standard, representerer den som sådan ikke noe revolusjonerende nytt, selv om ytelsesforbedringene virker lovende. I tillegg er den kun ett av forslagene til en ny kanalkode for DVB-RCS, og ettersom den ikke er trukket ut i en egen standard, eksisterer det heller ikke så veldig mye dokumentasjon på den. Det ble funnet noen simuleringer med en kode som har tilsvarende spesifikasjoner som TurboΦ, altså duo-binær inngang, 16 tilstander og med CRSC komponentenkodere i [4].

TurboΦ skal kunne håndtere rater fra 1/4 opp til 8/9, samt tilhørende modulasjon. De adaptive ratene burde ikke være noe problem, for i følge [5] kan TurboΦ håndtere en nesten kontinuerlig overgang fra 1/4 til $k/(k+1)$.

3.3.2 Ytelse

I [18] påpekes det at turbokoder med 16 tilstander er det mest hensiktsmessige valget for $10^{-9} < FER < 10^{-4}$, og at valget mellom binær kontra duo-binær kode avhenger av ratene man ønsker. Dersom man er avhengig av $R > 1/2$, bør man velge duo-binære koder, fordi sammenlignet med binære koder opplever de mindre ytelsestap som følge av punktering. Med økt ytelse kommer dessverre også økt kompleksitet, som forklart i seksjon 3.2.3 krever nemlig duo-binære turbokoder at man regner ut en 3-dimensjonal LLR. Hver av de tre LLR'ene i denne er mer komplekse enn for binære koder, siden man har dobbelt så mange mulige overganger mellom hver tilstand. Figuren 3.8 viser simuleringsresultater fra [5] for en blokklengde av samme størrelsesorden som de som er aktuelle i denne oppgaven.



Figur 3.8: Sammenligning av TurboΦ, SCCC og LDPC ved ratene 1/3, 5/6 og 9/10 for små blokklengder. [5]

3.4 Dekodingsalgoritmer

Turbokoder har etter introduksjonen i 1993 vært et populært forskningsfelt, hvor spesielt dekodingsalgoritmen har fått mye fokus. I og med at MAP-algoritmen er optimal med hensyn på BER og FER og i tillegg er forholdsvis kompleks, går mye av forskningen på modifikasjon eller forenkling av denne. Om turbokoden skal brukes i et sanntidssystem med stramt ressursbudsjett, bør man for eksempel vurdere forenklede utgaver. Max-Log-MAP er én slik variant, mens Log-MAP gjør den samme forenklingen som Max-Log-MAP, for deretter å korrigere unøyaktigheten ved hjelp av et tabelloppslag. Sistnevnte oppnår de samme ytelsene som MAP, men med mindre kompleksitet [19]. Den følgende seksjonen tar for seg den grunnleggende algoritmen MAP, før den deretter utleder Log-MAP og dekodingsalgoritmen som er implementert i denne masteroppgaven, Max-Log-MAP[4].

3.4.1 Modifisert Bahl et al.: MAP dekoding

Bahl et al. presenterte i [21] en algoritme for optimal dekoding av lineære koder, som i tillegg gir APP for hvert enkelt bit. Berrou et al. benyttet en modifisert utgave av denne algoritmen i [7, 13], kalt MAP-algoritmen. I artiklene deres ble den brukt på turbokoder med binær modulasjon, men prinsippet er også gyldig for modulasjoner med et høyere antall symboler, som for eksempel QPSK. Ettersom utledningen for binær modulasjon er den mest intuitive, og det også var slik den ble presentert i [7, 13], gjøres gjennomgangen for dette tilfellet. Seksjon 3.4.2viser deretter endringene som kreves for å bruke MAP på QPSK-modulerte symboler, som blant annet benyttes i DVB-RCS.

Gitt en RCS-enkoder med inntrengningsdybde K, kan dens tilstand ved tid k representeres som

[4]Utledningen av disse to baseres på [20]

$\mathbf{S}_k = (a_k, a_{k-1}, ..., a_{k-K+2})$. Det antas at informasjonssekvensen $\{d_k\}$ består av N uavhengige bit d_k, som alle tar verdien 1 eller 0 med lik sannsynlighet, og i tillegg at enkoderen starter og stopper i null-tilstanden, det vil si $S_0 = S_N = (0, 0, ..., 0)$. Utgangssekvensen fra enkoderen er $\mathbf{C}_1^N = (C_1, ..., C_k, ..., C_N)$, der $C_k = (X_k, Y_k)$, sendes gjennom den minneløse AWGN-kanalen med BPSK-modulasjon. Den mottatte sekvensen blir da $\mathbf{R}_1^N = (R_1, ..., R_k, ..., R_N)$, der $R_k = (x_k, y_k)$ gis av ligning (3.2)[5].

APP til hvert dekodede bit d_k kan regnes ut fra simultanfordelingen

$$\lambda_k^i(\mathbf{m}) = \Pr(d_k = i, \mathbf{S}_k = \mathbf{m}|\mathbf{R}_1^N), \tag{3.10}$$

siden dette er sannsynligheten for at bit d_k er lik i samtidig som enkodertilstanden $\mathbf{S}_k$ er lik m, gitt den mottatte sekvensen $\mathbf{R}_1^N$. Ved å summere over alle mulige enkodertilstander får vi da

$$Pr(d_k = i|\mathbf{R}_1^N) = \sum_{\mathbf{m}} \lambda_k^i(\mathbf{m}), \quad i = 1, 0. \tag{3.11}$$

Fra (3.3) og (3.11) kan vi dermed skrive LLR til det dekodede bitet d_k som

$$L(d_k) = \log \frac{\sum_{\mathbf{m}} \lambda_k^1(\mathbf{m})}{\sum_{\mathbf{m}} \lambda_k^0(\mathbf{m})}, \tag{3.12}$$

og dekoderen kan gjøre desisjonen basert på

$$\begin{array}{lll} \hat{d}_k = 1 & \text{for} & L(d_k) \geq 0 \\ \hat{d}_k = 0 & \text{for} & L(d_k) < 0. \end{array} \tag{3.13}$$

Ved å utvide (3.10) kan vi skrive $L(d_k)$ som

$$L(d_k) = \log \frac{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} Pr(d_k = 1, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}, \mathbf{R}_1^{k-1}, R_k, \mathbf{R}_{k+1}^N)}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} Pr(d_k = 0, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}, \mathbf{R}_1^{k-1}, R_k, \mathbf{R}_{k+1}^N)}, \tag{3.14}$$

og ved å bruke Bayes' regel og Markovegenskapene til kilden, kan dette igjen skrives som

$$\begin{aligned} L(d_k) = \log & \frac{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} Pr(\mathbf{R}_{k+1}^N|\mathbf{S}_k = \mathbf{m}) Pr(\mathbf{S}_{k-1} = \mathbf{m'}|\mathbf{R}_1^{k-1})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} Pr(\mathbf{R}_{k+1}^N|\mathbf{S}_k = \mathbf{m}) Pr(\mathbf{S}_{k-1} = \mathbf{m'}|\mathbf{R}_1^{k-1})} \\ & \cdot \frac{Pr(d_k = 1, \mathbf{S}_k = \mathbf{m}, R_k|\mathbf{S}_{k-1} = \mathbf{m'})}{Pr(d_k = 0, \mathbf{S}_k = \mathbf{m}, R_k|\mathbf{S}_{k-1} = \mathbf{m'})}. \end{aligned} \tag{3.15}$$

For å regne ut dette, innføres i [22] sannsynlighetsfunksjonene $\alpha_k(\mathbf{m})$, $\beta_k(\mathbf{m})$ og $\gamma_i(R_k, \mathbf{m}, \mathbf{m'})$, definert som

$$\alpha_k(\mathbf{m}) = Pr(\mathbf{S}_k = \mathbf{m}|\mathbf{R}_1^k) \tag{3.16}$$

$$\beta_k(\mathbf{m}) = \frac{Pr(\mathbf{R}_{k+1}^N|\mathbf{S}_k = \mathbf{m})}{Pr(\mathbf{R}_{k+1}^N|\mathbf{R}_1^k)} \tag{3.17}$$

$$\gamma_i(R_k, \mathbf{m'}, \mathbf{m}) = Pr(d_k = i, \mathbf{S}_k = \mathbf{m}, R_k|\mathbf{S}_{k-1} = \mathbf{m'}). \tag{3.18}$$

[5]Siden dette er utledningen for algoritmen i én dekoder-modul, brukes for enklere notasjons skyld $R_k = R_{ik}$, $Y_{ik} = Y_k$ og $y_{ik} = y_k$ i resten av seksjonen. For den permuterte dekoderen gjelder også $R_k = (\Pi(x_k), y_k)$

Ved hjelp av disse kan man skrive om (3.15) til

$$L(d_k) = \log \frac{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_1(R_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_0(R_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}. \tag{3.19}$$

$\gamma_i(R_k, \mathbf{m'}, \mathbf{m})$ kan beregnes ved hjelp av overgangssannsynlighetene til kanalen og enkodertrellisen. Ved å skrive om uttrykket fra (3.16) får vi

$$\begin{aligned} \gamma_i(R_k, \mathbf{m'}, \mathbf{m}) &= p(R_k | d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}) \\ &\cdot\ q(d_k = i | \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}) \\ &\cdot\ \pi(\mathbf{S}_k = \mathbf{m} | \mathbf{S}_{k-1} = \mathbf{m'}). \end{aligned} \tag{3.20}$$

$\pi(\mathbf{S}_k = \mathbf{m} | \mathbf{S}_{k-1} = \mathbf{m'})$ er overgangssannsynlighetene til enkodertrellisen, og representerer a priori informasjon. Siden det er antatt at $Pr(d_k = 1) = Pr(d_k = 0)$, vil $\pi(\cdot) = 1/2$ for de to aktuelle transisjonene. $q(d_k = i | \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'})$ er lik 1 dersom det eksisterer en overgang fra $\mathbf{S}_{k-1} = \mathbf{m}$ til $\mathbf{S}_k = \mathbf{m'}$ for $d_k = i$, og null ellers. $p(R_k | d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'})$ representerer overgangssannsynlighetene til den diskrete, minneløse, Gaussiske kanalen. På grunn av den betingede informasjonen, er denne uavhengig for x_k og y_k ($R_k = (x_k, y_k)$), og vi får[6]

$$\begin{aligned} p(R_k | d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}) &= p(x_k | d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}) \\ &\cdot\ p(y_k | d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'}). \end{aligned} \tag{3.21}$$

I [7] vises det at $\alpha_k(\mathbf{m})$ og $\beta_k(\mathbf{m})$ kan beregnes rekursivt som

$$\alpha_k(\mathbf{m}) = \frac{\sum_{\mathbf{m'}} \sum_{i=0}^{1} \gamma_i(R_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \sum_{i=0}^{1} \gamma_i(R_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'})} \tag{3.22}$$

$$\beta_k(\mathbf{m}) = \frac{\sum_{\mathbf{m'}} \sum_{i=0}^{1} \gamma_i(R_{k+1}, \mathbf{m}, \mathbf{m'}) \beta_{k+1}(\mathbf{m'})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \sum_{i=0}^{1} \gamma_i(R_{k+1}, \mathbf{m'}, \mathbf{m}) \alpha_k(\mathbf{m'})}. \tag{3.23}$$

For å regne ut $L(d_k)$ med denne algoritmen går man frem som følger:

Initialisering: $S_0 = S_N = (0, 0, ..., 0)$ gir

$$\alpha_0(\mathbf{0}) = 1; \quad \alpha_0(\mathbf{m}) = 0 \quad \forall \mathbf{m} \neq \mathbf{0} \tag{3.24}$$

$$\beta_N(\mathbf{0}) = 1; \quad \beta_N(\mathbf{m}) = 0 \quad \forall \mathbf{m} \neq \mathbf{0}, \tag{3.25}$$

Steg 1: For hver observasjon R_k regner man ut $\gamma_i(R_k, \mathbf{m'}, \mathbf{m})$ og $\alpha_k(\mathbf{m})$ fra henholdsvis (3.20) og (3.22), og lagrer verdiene.

Steg 2: Når dette er gjort for hele $\mathbf{R}_1^N$, regner man ut $\beta_k(\mathbf{m})$ fra (3.23), og til slutt kan man da beregne LLR for d_k med (3.19).

[6] Disse uttrykkene kan presiseres videre, basert på antagelser om kanalen og feilfordeling. Dette vil først bli gjort i avsnittene om duo-binære koder, ettersom denne masteroppgaven ikke skal gjennomføre simulering av den opprinnelige turbokoden

Ekstrinsikk informasjon

For å bruke resultatene til iterativ dekoding må man også beregne den ekstrinsikke informasjonen, slik at denne kan tilbakekobles til den andre dekodermodulen. Det følgende gjelder for det binære tilfellet: Alle deler som er avhengige av x_k skilles ut av uttrykket for $L(d_k)$. Siden enkoderen er systematisk ($X_k = d_k$), er $p(x_k|d_k = i, \mathbf{S}_k = \mathbf{m}, \mathbf{S}_{k-1} = \mathbf{m'})$ i uttrykket for $\gamma_i(R_k, \mathbf{m'}, \mathbf{m})$ uavhengig av tilstandene, og kan derfor faktoriseres ut av ligning (3.19)

$$L(d_k) = \log \frac{p(x_k|d_k = 1)}{p(x_k|d_k = 0)} + \log \frac{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_1(y_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_0(y_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}. \tag{3.26}$$

Betinget på $d_k = i$ er x_k gaussisk med forventningsverdi $x_k = (1 - 2i)$ og varians σ^2, slik at

$$L(d_k) = \frac{2}{\sigma^2} x_k + L_k^e(d_k), \tag{3.27}$$

med

$$L_k^e(d_k) \;=\; log \frac{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_1(y_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}{\sum_{\mathbf{m}} \sum_{\mathbf{m'}} \gamma_0(y_k, \mathbf{m'}, \mathbf{m}) \alpha_{k-1}(\mathbf{m'}) \beta_k(\mathbf{m})}. \tag{3.28}$$

$L_k^e(d_k)$ representerer den ekstrinsikke informasjonen beskrevet i seksjon 3.1.3, og som man ser er den uavhengig av x_k. Den kan regnes ut ved hjelp av ligning (3.28), men i praksis kalkulerer man isteden $L(d_k)$ og bruker ligning 3.27 til å finne den indirekte.

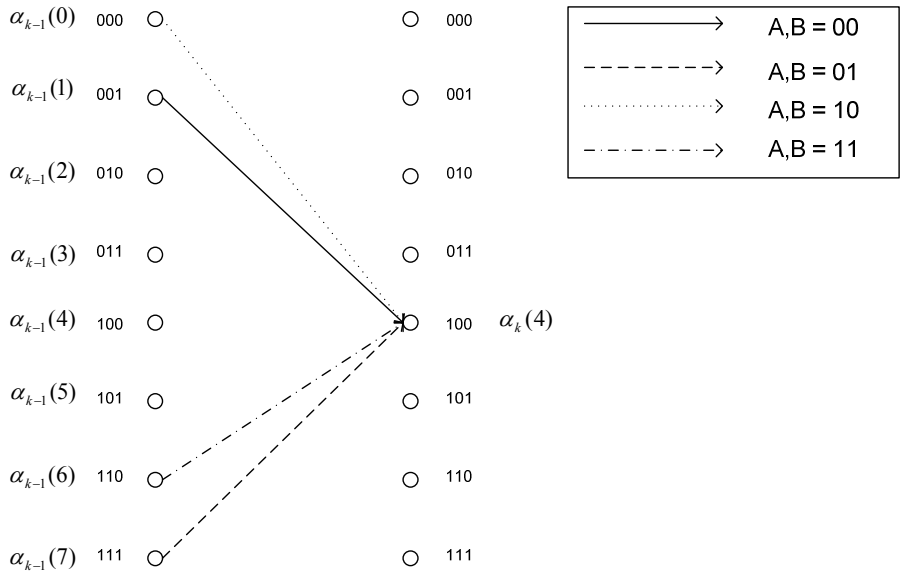
3.4.2 Duo-binær MAP og Log-MAP

Det tas utgangspunkt i utledningen fra seksjon 3.4.1, ettersom MAP også fungerer for ikke-binære symboler. Utledningen gjelder også her for én av dekoder-modulene. For enklere notasjon brukes desimaltall for å representere $d_k = 00, 01, 10, 11$, av samme grunn brukes $\gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)$ istedenfor $\gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1} = \mathbf{m'}, \mathbf{S}_k = \mathbf{m})$, og $\alpha_k(\mathbf{S}_k)$ istedenfor $\alpha_k(\mathbf{S}_k = \mathbf{m'})$ (tilsvarende for β). Med denne notasjonen får vi fra ligning (3.22) og (3.23)

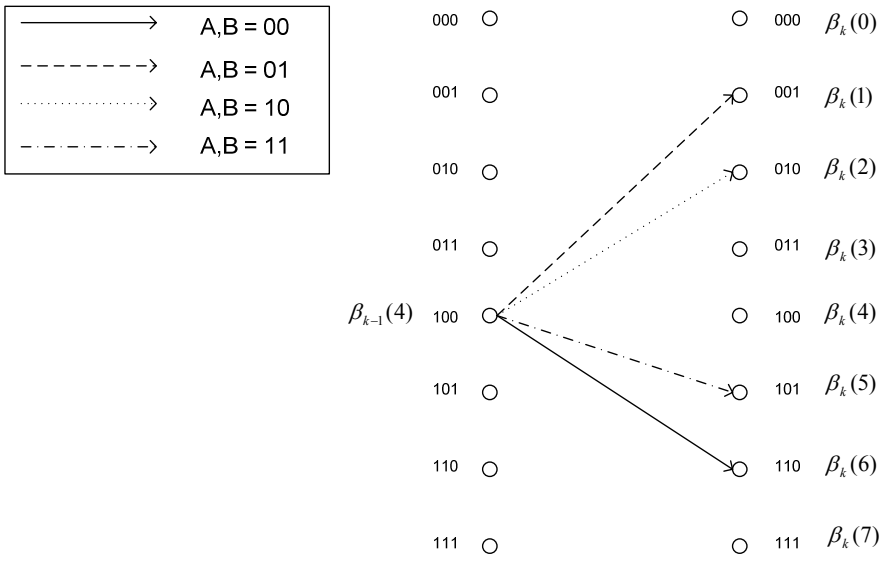
$$\alpha_k(\mathbf{S}_k) = \frac{\sum_{\mathbf{S}_{k-1}} \gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \alpha_{k-1}(\mathbf{S}_{k-1})}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} \gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \alpha_{k-1}(\mathbf{S}_{k-1})} \tag{3.29}$$

$$\beta_k(\mathbf{S}_k) = \frac{\sum_{\mathbf{S}_{k+1}} \gamma_i(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1}) \beta_{k+1}(\mathbf{S}_{k+1})}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k+1}} \gamma_i(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1}) \alpha_k(\mathbf{S}_k)}. \tag{3.30}$$

Beregningen av disse vises i figur 3.9 og 3.10. Tilsvarende får vi fra (3.19)



Figur 3.9: Kalkulering av $\alpha_k(4)$

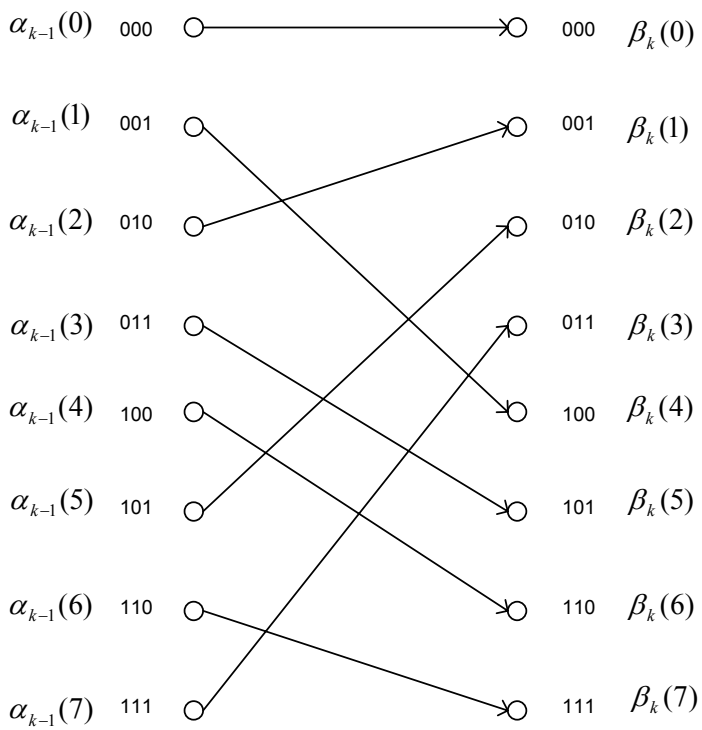


Figur 3.10: Kalkulering av $\beta_{k-1}(4)$

$$L_i(d_k) = \ln \frac{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} \gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \alpha_{k-1}(\mathbf{S}_{k-1}) \beta_k(\mathbf{S}_k)}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} \gamma_0(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \alpha_{k-1}(\mathbf{S}_{k-1}) \beta_k(\mathbf{S}_k)} \quad \text{for} \quad i = 1, 2, 3, \tag{3.31}$$

Her er summeringen av $\gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)$ over i utelatt, den er isteden innbakt i summen over de mulige tilstandene. I ligning (3.29) og (3.30) er det er kun fire overganger inn og ut av tilstandene (siden koden er duo-binær), og summen over tilstandene tar kun med disse gyldige overgangene.

For ligning (3.31) vil det være 8 gyldige overganger som må inkluderes. Beregningen av nevneren i $L_i(d_k)$ vises figur 3.11.



Figur 3.11: Kalkulering av nevneren i $L_i(d_k)$, det vil si grenmetrikker for i=0 (AB = 00)

Uttrykket for γ fra ligning (3.20) kan nå skrives

$$\gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) = p(\mathbf{R}_k | d_k = i, \mathbf{S}_k, \mathbf{S}_{k-1}) \cdot P(d_k = i) \quad \text{for} \quad i = 0, 1, 2, 3, \tag{3.32}$$

hvor $P(d_k = i)$ representerer a priori informasjon om symbolet d_k. I det følgende utledes et helt presist uttrykk for ligning (3.32) for duo-binære, QPSK-modulerte signaler, under antagelsen om en minneløs kanal med AWGN. Angående notasjonen: $\mathbf{R}_k$ brukt i dette avsnittet er ikke den samme $\mathbf{R}_k$ som i seksjon 3.2.3. Der omfatter $\mathbf{R}_k$ systembit og redundansene fra både den regulære og den permuterte sekvensen, mens det her er snakk om inngangen på én av modulene, altså én av de to bitsekvensene og dennes tilhørende redundans. Derfor brukes $\mathbf{R}_k = (\mathbf{r}_k^s, \mathbf{r}_k^p) = (r_k^{s,I}, r_k^{s,Q}, r_k^{p,I}, r_k^{p,Q})$ i det videre, der s står for systembitene, p står for de tilhørende redundansbitene, og I og Q refererer til I- og Q-komponenten av QPSK-symbolene $\mathbf{r}_k^s$ og $\mathbf{r}_k^p$. På samme måte er $x_k^{s,I}(i)$, $x_k^{s,Q}(i)$, $x_k^{p,I}(i, \mathbf{S}_k, \mathbf{S}_{k-1})$ og $x_k^{p,Q}(i, \mathbf{S}_k, \mathbf{S}_{k-1})$ dekoderens system- og paritetsbit for $d_k = i$, lest ut av trellisen og modulert med QPSK. Med utgangspunkt i (3.21) har man

$$p(\mathbf{R}_k | d_k = i, \mathbf{S}_k, \mathbf{S}_{k-1}) = p(\mathbf{r}_k^s | i, \mathbf{S}_k, \mathbf{S}_{k-1}) \cdot p(\mathbf{r}_k^p | i, \mathbf{S}_k, \mathbf{S}_{k-1}) \tag{3.33}$$

$$= p(\mathbf{r}_k^s | \mathbf{x}_k^s(i)) \cdot p(\mathbf{r}_k^p | \mathbf{x}_k^p(i, \mathbf{S}_k, \mathbf{S}_{k-1})) \quad \text{for} \quad i = 0, 1, 2, 3, \tag{3.34}$$

der uttrykket med systemsymbolet $\mathbf{r}_k^s$ er uavhengig av tilstandene $\mathbf{S}_{k-1}$ og $\mathbf{S}_k$, gitt $d_k = i$. Under

antagelsen om en minneløs AWGN-kanal påvirkes symbolene av normalfordelt støy, som gir

$$\begin{aligned}
& p(\mathbf{R}_k|d_k=i,\mathbf{S}_k,\mathbf{S}_{k-1}) \qquad \text{for } \; i=0,1,2,3 \\
= \;& \frac{1}{\pi\cdot N_0} exp\left\{-\frac{E_s}{N_0}\left[(r_k^{s,I}-x_k^{s,I}(i))^2+(r_k^{s,Q}-x_k^{s,Q}(i))^2\right]\right\} \\
& \cdot\frac{1}{\pi\cdot N_0} exp\left\{-\frac{E_s}{N_0}\left[(r_k^{p,I}-x_k^{p,I}(i,\mathbf{S}_k,\mathbf{S}_{k-1}))^2+(r_k^{p,Q}-x_k^{p,Q}(i,\mathbf{S}_k,\mathbf{S}_{k-1}))^2\right]\right\} \\
= \;& B_k\cdot exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{s,I}\cdot x_k^{s,I}(i)+r_k^{s,Q}\cdot x_k^{s,Q}(i)\right]\right\} \\
& \cdot exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{p,I}\cdot x_k^{p,I}(i,\mathbf{S}_k,\mathbf{S}_{k-1})+r_k^{p,Q}\cdot x_k^{p,Q}(i,\mathbf{S}_k,\mathbf{S}_{k-1})\right]\right\}, \qquad (3.35)
\end{aligned}$$

der $L_c{=}4\frac{E_s}{N_0}{=}4R_c\frac{E_b}{N_0}$ er kanalens pålitelighetsverdi, og B_k inneholder konstanter og ledd som vil falle bort når det settes inn i ligning (3.31). Videre får man

$$\begin{aligned}
& \gamma_i(\mathbf{R}_k,\mathbf{S}_{k-1},\mathbf{S}_k) \qquad \text{for } \; i=0,1,2,3 \\
= \;& p(\mathbf{R}_k|d_k=i,\mathbf{S}_{k-1},\mathbf{S}_k)\cdot P(d_k=i) \\
= \;& B_k\cdot exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{s,I}\cdot x_k^{s,I}(i)+r_k^{s,Q}\cdot x_k^{s,Q}(i)\right]\right\} \\
& \cdot exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{p,I}\cdot x_k^{p,I}(i,\mathbf{S}_k,\mathbf{S}_{k-1})+r_k^{p,Q}\cdot x_k^{p,Q}(i,\mathbf{S}_k,\mathbf{S}_{k-1})\right]\right\}\cdot P(d_k=i). \qquad (3.36)
\end{aligned}$$

Legg merke til at informasjonen knyttet til redundansen holdes adskilt, siden denne vil inngå i den ekstrinsikke informasjonen. I den sammenheng er det nyttig med følgende hjelpevariabel

$$\gamma_i^e(\mathbf{r}_k^p,\mathbf{S}_{k-1},\mathbf{S}_k)=exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{p,I}\cdot x_k^{p,I}(i,\mathbf{S}_k,\mathbf{S}_{k-1})+r_k^{p,Q}\cdot x_k^{p,Q}(i,\mathbf{S}_k,\mathbf{S}_{k-1})\right]\right\}. \qquad (3.37)$$

Ved å bruke de forskjellige uttrykkene ovenfor i ligning (3.31) og forkorte, får man

$$\begin{aligned}
L_i(d_k) \;=\; & \ln\frac{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_i(\mathbf{R}_k,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)}{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_0(\mathbf{R}_k,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)} \\
= \;& \ln\frac{exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{s,I}\cdot x_k^{s,I}(i)+r_k^{s,Q}\cdot x_k^{s,Q}(i)\right]\right\}\cdot P(d_k=i)}{exp\left\{\frac{1}{2}L_c\cdot\left[r_k^{s,I}\cdot x_k^{s,I}(0)+r_k^{s,Q}\cdot x_k^{s,Q}(0)\right]\right\}\cdot P(d_k=0)} \\
& +\ln\frac{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_i^e(\mathbf{r}_k^p,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)}{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_0^e(\mathbf{r}_k^p,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)} \\
= \;& \frac{1}{2}L_c\cdot\left[(r_k^{s,I}\cdot x_k^{s,I}(i)+r_k^{s,Q}\cdot x_k^{s,Q}(i))-(r_k^{s,I}\cdot x_k^{s,I}(0)+r_k^{s,Q}\cdot x_k^{s,Q}(0))\right] \\
& +\underbrace{\ln\frac{P(d_k=i)}{P(d_k=0)}}_{L_i^a(d_k)}+\underbrace{\ln\frac{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_i^e(\mathbf{r}_k^p,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)}{\sum_{\mathbf{S}_k}\sum_{\mathbf{S}_{k-1}}\gamma_0^e(\mathbf{r}_k^p,\mathbf{S}_{k-1},\mathbf{S}_k)\alpha_{k-1}(\mathbf{S}_{k-1})\beta_k(\mathbf{S}_k)}}_{L_i^e(d_k)}. \qquad (3.38)
\end{aligned}$$

Under dekoding beregnes først $L_i(d_k)$ i ligning (3.36) ved hjelp av uttrykkene fra ligning (3.29), (3.30) og (3.36), for deretter å beregne den ekstrinsikke informasjonen $L_i^e(d_k)$ slik

$$\begin{aligned} L_i^e(d_k) &= L_i(d_k) - L_i^a(d_k) \\ &\quad -\frac{1}{2}L_c \cdot \left[(r_k^{s,I} \cdot x_k^{s,I}(i) + r_k^{s,Q} \cdot x_k^{s,Q}(i)) - (r_k^{s,I} \cdot x_k^{s,I}(0) + r_k^{s,Q} \cdot x_k^{s,Q}(0)) \right]. \end{aligned} \tag{3.39}$$

$L_i^a(d_k)$ representerer a priori informasjon, og under den iterative dekodingen mates den ekstrinsikke informasjonen fra den andre dekoderen inn via denne variabelen. Det vil si, SISO_1 lager $L_i^1(d_k)$, bruker ligning (3.39) til å beregne $L_i^{e1}(d_k)$, som deretter interleaves og lagres i $L_i^{a2}(d_k)$. SISO_2 får denne på inngangen i tillegg til de interleavede systembitene.

På samme måte beregner SISO_2 først $L_i^2(d_k)$ og deretter $L_i^{e2}(d_k)$. Denne blir de-interleavet og lagret i $L_i^{a1}(d_k)$, som SISO_1 får inn i tillegg til den regulære bitsekvensen ved neste iterasjon. Dermed får SISO'ene hele tiden oppdaterte sannsynlighetsverdier for bitsekvensene sine. Dette er selve essensen i den iterative dekodingen, og etter et gitt antall iterasjoner brukes til slutt $L_i^2(d_k)$ for å dekode og demodulere symbolene.

Log MAP

Ligningene i 3.4.2 innebærer mye multiplikasjon, og er lettere å håndtere i logaritmiske versjoner. Definerer derfor

$$\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) = \ln \gamma_i(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \qquad \text{for} \quad i = 0, 1, 2, 3. \tag{3.40}$$

Brukt i ligning (3.29) og (3.30) og med litt omskriving gir dette

$$\overline{\alpha_k}(\mathbf{S}_k) = \ln \frac{\sum_{\mathbf{S}_{k-1}} e^{\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)} e^{\overline{\alpha_{k-1}}(\mathbf{S}_{k-1})}}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} e^{\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)} e^{\overline{\alpha_{k-1}}(\mathbf{S}_{k-1})}} \tag{3.41}$$

$$\overline{\beta_k}(\mathbf{S}_k) = \ln \frac{\sum_{\mathbf{S}_{k+1}} e^{\overline{\gamma_i}(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1})} e^{\overline{\beta_{k+1}}(\mathbf{S}_{k+1})}}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k+1}} e^{\overline{\gamma_i}(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1})} e^{\overline{\alpha_k}(\mathbf{S}_k)}}. \tag{3.42}$$

Dersom dette nå brukes i ligning (3.31), får man Log-MAP:

$$L_i(d_k) = \ln \frac{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} e^{\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)} e^{\overline{\alpha_{k-1}}(\mathbf{S}_{k-1})} e^{\overline{\beta_k}(\mathbf{S}_k)}}{\sum_{\mathbf{S}_k} \sum_{\mathbf{S}_{k-1}} e^{\overline{\gamma_0}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k)} e^{\overline{\alpha_{k-1}}(\mathbf{S}_{k-1})} e^{\overline{\beta_k}(\mathbf{S}_k)}} \quad \text{for} \quad i = 1, 2, 3. \tag{3.43}$$

I utgangspunktet kan dette se minst like komplekst ut, men summene over eksponentialuttrykkene kan omformes. Til dette brukes sammenhengen

$$\ln \sum_{i=1}^{k} e^{x_i} = \max_i(x_i) + \underbrace{\ln \sum_{i=1}^{k} e^{x_i - \max_i(x_i)}}_{f_c} \approx \max_i(x_i), \tag{3.44}$$

der f_c er et korreksjonsledd som kan hentes ut av en liten oppslagstabell [19]. Ved hjelp av maksimalisering og oppslag i korreksjonstabellen kan dermed ligningene (3.41), (3.42) og (3.43) beregnes svært effektivt. Log-MAP oppnår samme nøyaktighet som vanlig MAP, men med mindre komplekse operasjoner.

Ved å gjøre forenklingen i siste ledd av ligning (3.44) får man Max-Log-MAP, som også er algoritmen som er blitt implementert i denne oppgaven. De modifiserte formlene vises i neste seksjon, før detaljene rundt den iterative dekodingen gjennomgås.

3.4.3 Duo-binær Max-Log-MAP

Max-Log-MAP reduserer beregningen av ligningene (3.41), (3.42) og (3.43) til maksimaliseringer over tilstandene istedenfor nøyaktig beregning av uttrykkene. Dette gir omtrent en halvering i antall operasjoner i forhold til vanlig MAP, utføres med enklere operasjoner, og resulterer i en ytelsesnedgang i SNR på kun 0,5 dB i forhold til MAP [19]. Fra ligning (3.36) får man

$$\begin{aligned}
&\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) \qquad \text{for } \; i = 0,1,2,3 \\
&= B'_k + \frac{1}{2} L_c \cdot \left[r_k^{s,I} \cdot x_k^{s,I}(i) + r_k^{s,Q} \cdot x_k^{s,Q}(i) \right] \\
&+ \frac{1}{2} L_c \cdot \left[r_k^{p,I} \cdot x_k^{p,I}(i, \mathbf{S}_k, \mathbf{S}_{k-1}) + r_k^{p,Q} \cdot x_k^{p,Q}(i, \mathbf{S}_k, \mathbf{S}_{k-1}) \right] + \ln P(d_k = i),
\end{aligned} \tag{3.45}$$

mens bruk av forenklingen fra (3.44) i ligningene (3.41) og (3.42) gir

$$\begin{aligned}
\overline{\alpha_k}(\mathbf{S}_k) \approx & \max_{(\mathbf{S}_{k-1})} [\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) + \overline{\alpha_{k-1}}(\mathbf{S}_{k-1})] \\
& - \max_{(\mathbf{S}_k, \mathbf{S}_{k-1})} [\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) + \overline{\alpha_{k-1}}(\mathbf{S}_{k-1})]
\end{aligned} \tag{3.46}$$

$$\begin{aligned}
\overline{\beta_k}(\mathbf{S}_k) \approx & \max_{(\mathbf{S}_{k+1})} [\overline{\gamma_i}(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1}) + \overline{\beta_{k+1}}(\mathbf{S}_{k+1})] \\
& - \max_{(\mathbf{S}_{k+1}, \mathbf{S}_k)} [\overline{\gamma_i}(\mathbf{R}_{k+1}, \mathbf{S}_k, \mathbf{S}_{k+1}) + \overline{\alpha_k}(\mathbf{S}_k)],
\end{aligned} \tag{3.47}$$

hvor det andre leddet i hvert av uttrykkene representerer et normaliseringsledd. Innsatt i (3.43) får man til slutt

$$\begin{aligned}
L_i(\hat{d}_k) \approx & \max_{(\mathbf{S}_k, \mathbf{S}_{k-1})} [\overline{\gamma_i}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) + \overline{\alpha_{k-1}}(\mathbf{S}_{k-1}) + \overline{\beta_k}(\mathbf{S}_k)] \\
& - \max_{(\mathbf{S}_k, \mathbf{S}_{k-1})} [\overline{\gamma_0}(\mathbf{R}_k, \mathbf{S}_{k-1}, \mathbf{S}_k) + \overline{\alpha_{k-1}}(\mathbf{S}_{k-1}) + \overline{\beta_k}(\mathbf{S}_k)] \qquad \text{for } \; i = 1,2,3,
\end{aligned} \tag{3.48}$$

som er formelen for LLR'en ved bruk av Max-Log-MAP. På samme måte som i (3.39) beregnes ekstrinsikk informasjon i Max-Log-MAP ut som følger

$$\begin{aligned}
L_i^e(\hat{d}_k) \;=\; & L_i(\hat{d}_k) - L_i^a(d_k) \\
& - \frac{1}{2} L_c \cdot \left[(r_k^{s,I} \cdot x_k^{s,I}(i) + r_k^{s,Q} \cdot x_k^{s,Q}(i)) - (r_k^{s,I} \cdot x_k^{s,I}(0) + r_k^{s,Q} \cdot x_k^{s,Q}(0)) \right].
\end{aligned} \tag{3.49}$$

Ettersom kanalens pålitelighetsverdi L_c benyttes til å vekte kanalverdiene i beregningene over, er det essensielt at denne kan estimeres korrekt.

Iterativ dekoding med Max-Log-MAP

Denne seksjonen tar for seg hvordan parametrene fra seksjonene over brukes i den iterative dekodingen, og beskriver også initialisering og terminering. Siden koden er sirkulær gjelder

$$\overline{\alpha_0}(\mathbf{S}_i) = \overline{\alpha_N}(\mathbf{S}_i) \qquad \text{for } \forall\ \mathbf{S}_i \tag{3.50}$$

$$\overline{\beta_N}(\mathbf{S}_i) = \overline{\beta_0}(\mathbf{S}_i) \qquad \text{for } \forall\ \mathbf{S}_i. \tag{3.51}$$

I den iterative dekodingen brukes ekstrinsikk informasjon fra forrige dekodermodul som a priori informasjon i den neste. For $L_i^a(d_k)$ i neste dekodermodul gjelder altså følgende, dersom $L_i^e(\hat{d}_k)$ er ekstrinsikk informasjon fra forrige dekodermodul:

$$L_i^a(d_k) = \ln \frac{P(d_k = i)}{P(d_k = 0)} = L_i^e(d_k). \tag{3.52}$$

Fra dette kan man sette opp

$$\begin{cases} L_0^e(\hat{d}_k) = \ln \dfrac{P(d_k = 00)}{P(d_k = 00)} = 0 \\ L_1^e(\hat{d}_k) = \ln \dfrac{P(d_k = 01)}{P(d_k = 00)} \\ L_2^e(\hat{d}_k) = \ln \dfrac{P(d_k = 10)}{P(d_k = 00)} \\ L_3^e(\hat{d}_k) = \ln \dfrac{P(d_k = 11)}{P(d_k = 00)}, \end{cases} \tag{3.53}$$

som deretter kan omformes til

$$\begin{cases} P(d_k = 00) = \dfrac{1}{1 + e^{L_1^e(\hat{d}_k)} + e^{L_2^e(\hat{d}_k)} + e^{L_3^e(\hat{d}_k)}} \\ P(d_k = 01) = \dfrac{e^{L_1^e(\hat{d}_k)}}{1 + e^{L_1^e(\hat{d}_k)} + e^{L_2^e(\hat{d}_k)} + e^{L_3^e(\hat{d}_k)}} \\ P(d_k = 10) = \dfrac{e^{L_2^e(\hat{d}_k)}}{1 + e^{L_1^e(\hat{d}_k)} + e^{L_2^e(\hat{d}_k)} + e^{L_3^e(\hat{d}_k)}} \\ P(d_k = 11) = \dfrac{e^{L_3^e(\hat{d}_k)}}{1 + e^{L_1^e(\hat{d}_k)} + e^{L_2^e(\hat{d}_k)} + e^{L_3^e(\hat{d}_k)}}. \end{cases} \tag{3.54}$$

Ved å ta logaritmen av dette uttrykket samt bruke forenklingen fra ligning (3.44), får man til slutt

$$\begin{cases} \ln P(d_k = 00) = -\max[0, L_1^e(\hat{d}_k), L_2^e(\hat{d}_k), L_3^e(\hat{d}_k)] \\ \ln P(d_k = 01) = L_1^e - \max[0, L_1^e(\hat{d}_k), L_2^e(\hat{d}_k), L_3^e(\hat{d}_k)] \\ \ln P(d_k = 10) = L_2^e - \max[0, L_1^e(\hat{d}_k), L_2^e(\hat{d}_k), L_3^e(\hat{d}_k)] \\ \ln P(d_k = 11) = L_3^e - \max[0, L_1^e(\hat{d}_k), L_2^e(\hat{d}_k), L_3^e(\hat{d}_k)]. \end{cases} \tag{3.55}$$

Disse formlene sørger for at SISO-modulene stadig får oppdaterte sannsynlighetsverdier på inngangen, og dermed kan forbedre estimatene sine.

Initialisering og terminering

Ved initialisering har vi ingen ekstrinsikk informasjon å basere a priori-verdiene på, og under antagelsen om like sannsynlige symboler får man via (3.53)

$$\begin{cases} \ln P(d_k = 00) = 0 \\ \ln P(d_k = 01) = 0 \\ \ln P(d_k = 10) = 0 \\ \ln P(d_k = 11) = 0. \end{cases} \tag{3.56}$$

Denne antagelsen gir oss også

$$\overline{\alpha_0}(\mathbf{S}_i) = \ln(\alpha_0(\mathbf{S}_i) = 1) = 0 \qquad \text{for } \forall\ \mathbf{S}_i \tag{3.57}$$

$$\overline{\beta_N}(\mathbf{S}_i) = \ln(\beta_N(\mathbf{S}_i) = 1) = 0 \qquad \text{for } \forall\ \mathbf{S}_i. \tag{3.58}$$

Etter å ha initialisert SISO_1 med disse verdiene, kjøres et visst antall dekodingsiterasjoner. Etter at disse er fullført de-interleaves den siste LLR'en fra SISO_2, og brukes i den endelige dekodingen som vist under

$$\hat{d}_k = \begin{cases} 01 & \text{for } L(\hat{d}_k) = L_1(\hat{d}_k);\ L_1(\hat{d}_k) > 0 \\ 10 & \text{for } L(\hat{d}_k) = L_2(\hat{d}_k);\ L_2(\hat{d}_k) > 0 \\ 11 & \text{for } L(\hat{d}_k) = L_3(\hat{d}_k);\ L_3(\hat{d}_k) > 0 \\ 00 & \text{ellers.} \end{cases} \tag{3.59}$$

der

$$L(\hat{d}_k) = \max[L_1(\hat{d}_k), L_2(\hat{d}_k), L_3(\hat{d}_k)]. \tag{3.60}$$

Kapittel 4

Metode og implementasjon

For å undersøke forskjellene mellom turbokoden i DVB-RCS og TurboΦ nøyere har denne masteroppgaven som mål å gjennomføre simuleringer av de to turbokodene. TurboΦ er som tidligere nevnt en av kandidatene til en oppgradering av den eksisterende DVB-RCS-koden, og begge turbokodene testes i den samme modellen av DVB-RCS. Dette kapitlet vil ta for seg systemmodellen og implementasjonen av de to kodene, samt gjennomgå og begrunne de designvalg som er gjort med tanke på arkitektur, omfang og eventuelle forenklinger. Verifikasjon er et viktige aspekt ved simulering, og gjøres for alle deler av implementasjonen. Mulige kilder til usikkerhet og/eller feil diskuteres også, samt hvilke parametre og resultater det er hensiktsmessig å hente ut.

Implementasjonen av turbokoden i DVB-RCS og TurboΦ, samt støttefunksjoner, hjelpefiler og overgangsfiler for Matlab/C leveres sammen med denne rapporten på NTNUs nettbaserte system DAIM (Digital Arkivering og Innlevering av Masteroppgaver), og er derfor tilgjengelige der.

4.1 DVB-RCS og TurboΦ

Disse turbokodene er på mange måter like, og det er også helt naturlig ettersom TurboΦ er foreslått som en oppgradering av turbokoden spesifisert i DVB-RCS [12]. Forskjellen ligger først og fremst i at TurboΦ skal kunne operere med et større utvalg av rater og blokklengder, samtidig som den reduserer feilgulvet man opplever med den eksisterende koden. I og med at den tilbyr en ytelsesforbedring, kan det tenkes at den også vil muliggjøre adaptiv modulasjon for turbokoden i DVB-RCS, som per nå kun benytter QPSK.

I [5] beskrives kun generatoren for én redundans, $G_Y = \{35\}_8$. Hvorvidt man velger å implementere rater$<1/2$ ved å gjøre koden binær eller ved hjelp av et ekstra redundansbit, overlates til hver enkelt. Simuleringene i den samme artikkelen bruker en binær utgave av TurboΦ, på tross av fordelene ved duo-binære koder som nevnes i seksjon 3.2.4. Dette kan skyldes at de ser for seg de fleste operasjonene på rater$<1/2$, der binære koder faktisk kan være bedre. På bakgrunn av de nevnte fordeler, og for bedre å kunne sammenligne med den duo-binære koden i DVB-RCS, ble det allikevel besluttet å generere et ekstra redundansbit med $G_W = \{35\}_8$.

Forskjellene mellom kodene skal være mest synlig ved lave feilrater, noe som kan være litt problematisk for simuleringene. Dette fordi simuleringer for å finne en gitt FER, som et absolutt

minimum, krever simulering av et inverst antall pakker. Det betyr at om man ønsker resultater ved $FER = 10^{-7}$, så må man gjennomføre en simulering med *minst* ti millioner pakker. For å oppnå bedre sikkerhet rundt resultatet bør man også simulere inntil et visst antall feil er oppnådd, som medfører enda høyere krav til antallet. Av samme grunn bør man også ha en minimumsgrense på antall simulerte pakker, slik at simuleringer der mye feil genereres også oppnår en viss sikkerhet rundt sine resultater.

Utgangspunktet for denne oppgaven var simuleringer ned mot $10^{-6} < FER < 10^{-5}$, som med en sikkerhetsmargin på 50 feil krever simulering av opptil $5 \cdot 10^7$ pakker. Simuleringene gjennomføres ved å iterere over et utvalg av SNR-verdier, og man vet ikke på forhånd helt hvilken FER en gitt SNR vil gi. Om man er uheldig vil kanskje de høyeste SNR-verdiene i en simulering tilsvare en FER på 10^{-9}, som ville gitt en "evigvarende" simulering dersom kun sikkerhetsmarginen på feil brukes som stoppkriterium. Av tidshensyn settes derfor et tak på antall simulerte pakker, slik at simuleringen stoppes dersom man når denne øvre grensen.

4.2 Implementasjon

All kildekode er skrevet i Matlab og C, ved hjelp av Matlab r2007b og XCode. I første omgang skulle turbokoden fra DVB-RCS implementeres og testes, ettersom dette er en veletablert turbokode som det finnes mye tidligere resultater for. Disse ble brukt som en indikator på hvorvidt implementasjonen med rimelig sikkerhet kunne antas å være korrekt, og i så fall hvor godt den presterte. Da den begynte å produsere tilfredsstillende resultater, ble det tatt utgangspunkt i denne kildekoden for å utforme en versjon av TurboΦ. DVB-RCS-resultatene ble blant annet sammenlignet med resultater fra [12, 20].

Kildekoden for DVB-RCS ble skrevet så modulær som mulig, slik at overgangen til TurboΦ kunne utføres uten å påvirke noen av de andre komponentene i systemet. Dermed kunne man ved å simulere og verifisere resultatene fra DVB-RCS forsikre seg om at resten av modellen var korrekt, slik at kun feil eller dårlig design av komponentene i TurboΦ ville påvirke dennes simuleringsresultater. Siden de aktuelle turbokodene fungerer som blokkoder har dataene i en gitt pakke ingen innvirkning på andre pakker, og de kan derfor simuleres uavhengig av hverandre. Implementasjonen genererer altså én og én pakke, inntil en av grensene fra seksjon 4.1 nås. Denne uavhengigheten betyr også at simuleringer ved de laveste feilratene kan splittes opp og kjøres hver for seg, for deretter å kombinere resultatet etter endt simulering.

Implementasjonsvalg

Som forklart i 4.1 vil det kreves et særdeles høyt antall pakker for noen av simuleringene. Matlab er i utgangspunktet godt egnet til simuleringer av digitale systemer, og har blant annet en egen modul med funksjoner spesialdesignet for slike formål kalt "Digital Signaling Toolbox". Men på grunn av den enorme mengder pakker ble det i samråd med veileder besluttet å implementere en løsning der deler av simuleringene gjennomføres i C.

C ligger nærmere ren maskinkode, og har ikke like mange av de innebygde fasilitetene som et høynivåspråk som Matlab eller Java har. Dette innebærer at man må være mer påpasselig, og koden blir også noe mer tungvint å skrive. Fordelen ved C er nettopp fraværet av høynivåspråkenes forsinkende funksjoner og feilsjekker. I de fleste tilfeller vil nemlig en implementasjon i C

ha langt raskere kjøretider enn tilsvarende kode i et høynivåspråk.

Noen av funksjonene ble implementert i Matlab først, for lettere å kunne gjøre feilsøk og verifikasjon underveis. I [23] finnes datasett man kan benytte til å verifisere enkoderen i en DVB-RCS turbokode, dette ble gjort for Matlab-versjonen av denne enkoderen. Det førte også med seg en uventet oppdagelse, som diskuteres videre i 4.4.

Siden alle deler av turbokoden i DVB-RCS er fullstendig spesifisert, hadde man faste holdepunkter å basere hele implementasjonen på. Med TurboΦ var det noe verre. Det viste seg vanskelig å lokalisere gode verdier for interleaveren, og lenge hadde man bare tilgang på verdier for et par blokker. Nokså sent ble det funnet et sett med verdier for de aller fleste blokklengdene i [24], men som det viste seg under simuleringen var det varierende kvalitet på disse parametrene.

4.3 System-modell

Implementasjonen i C har en egen header-fil, der man kan stille inn følgende: blokklengde, rate, feilmargin, maksimalt og minimalt antall pakker, antall dekodingsiterasjoner og spennvidden man ønsker å simulere i SNR (spesifisert i E_b/N_0). Blant de publiserte resultatene ser dog 8 dekodingsiterasjoner ut til å være en de facto standard, så det har av den grunn ikke blitt simulert med annet enn dette.

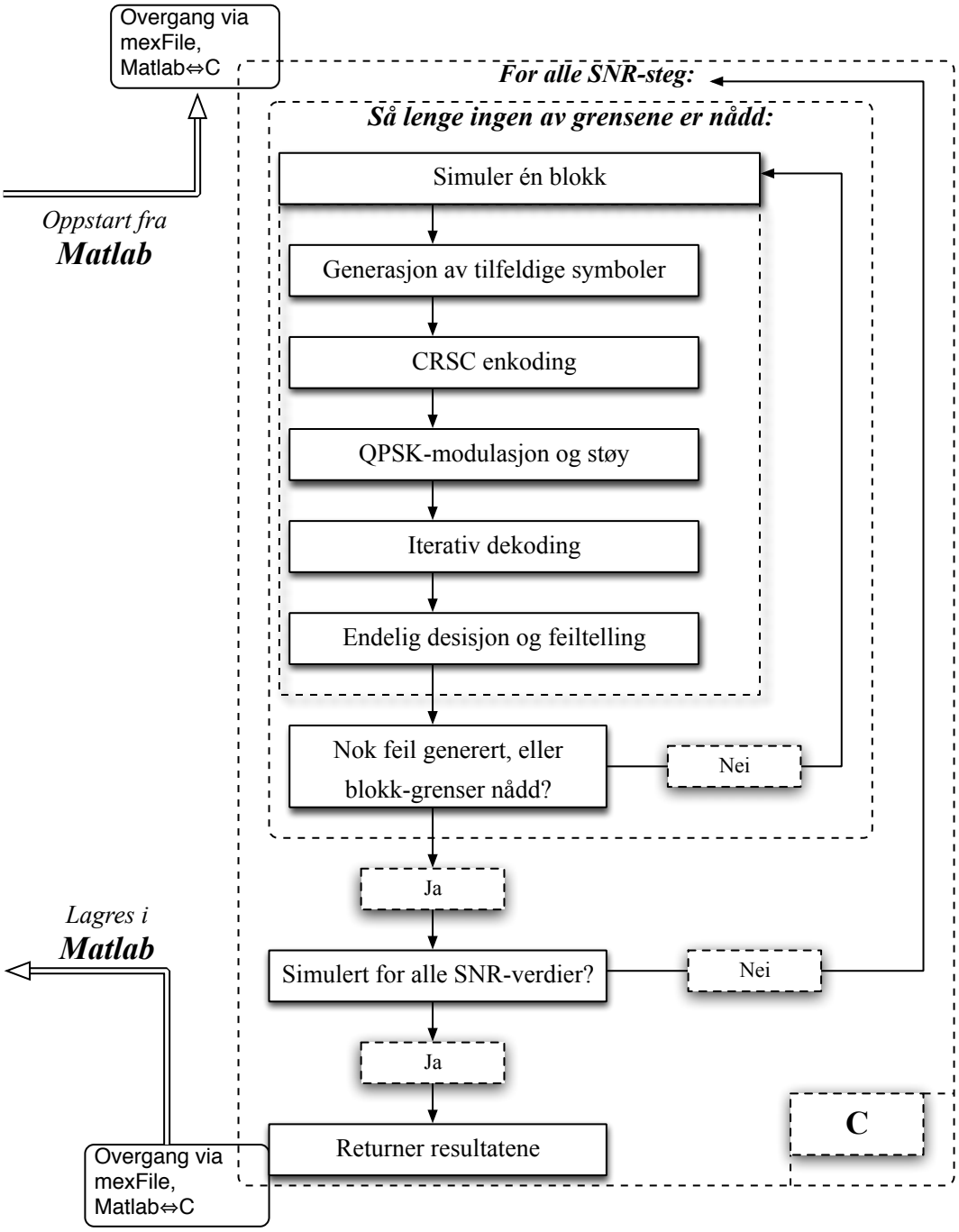
Etter innstilling av ønskede parametre i header-filen og kompilering, settes simuleringen igang via et enkelt Matlab-script. Koden itererer gjennom de ønskede SNR-verdiene og tar vare på FER, BER og enkelte verifikasjonsdata for hver iterasjon. Hver SNR-iterasjon holder frem til enten feilmarginen eller maksgrensen nås, og så lenge minimumsantallet frames ikke er simulert. Etter endt simulering returneres resultatene til Matlab hvor de lagres i en .mat-fil. Den overordnede system-modellen vises i figur 4.1.

Den overordnede fremgangsmåten i simuleringene er identisk for turbokoden i DVB-RCS og TurboΦ. Det er kun innmaten i CRCS-enkoderen og interleaveren, samt de endringer dette medfører i dekoderen, som er forskjellig. Enkoderoperasjonen demonstreres i figur 4.2, mens den iterative dekodingen vises i figur 4.3

4.4 Verifikasjon

Under testing av den tidlige Matlab-versjonen av enkoderen, dukket det opp et problem som lenge var forvirrende: Uansett hvor nøye man gikk implementasjonen av enkoderen etter i sømmene, og samme hvor korrekt den så ut til å være kodet (som den senere viste seg å ha vært hele tiden), gav den ikke de samme resultatene som verifikasjonsdataene fra [23]. Etter en en god porsjon med feilsøk og forargelse, viste det seg at permutasjonen som oppgis i [23], *ikke* er den samme som verifiseres i det samme dokumentet.

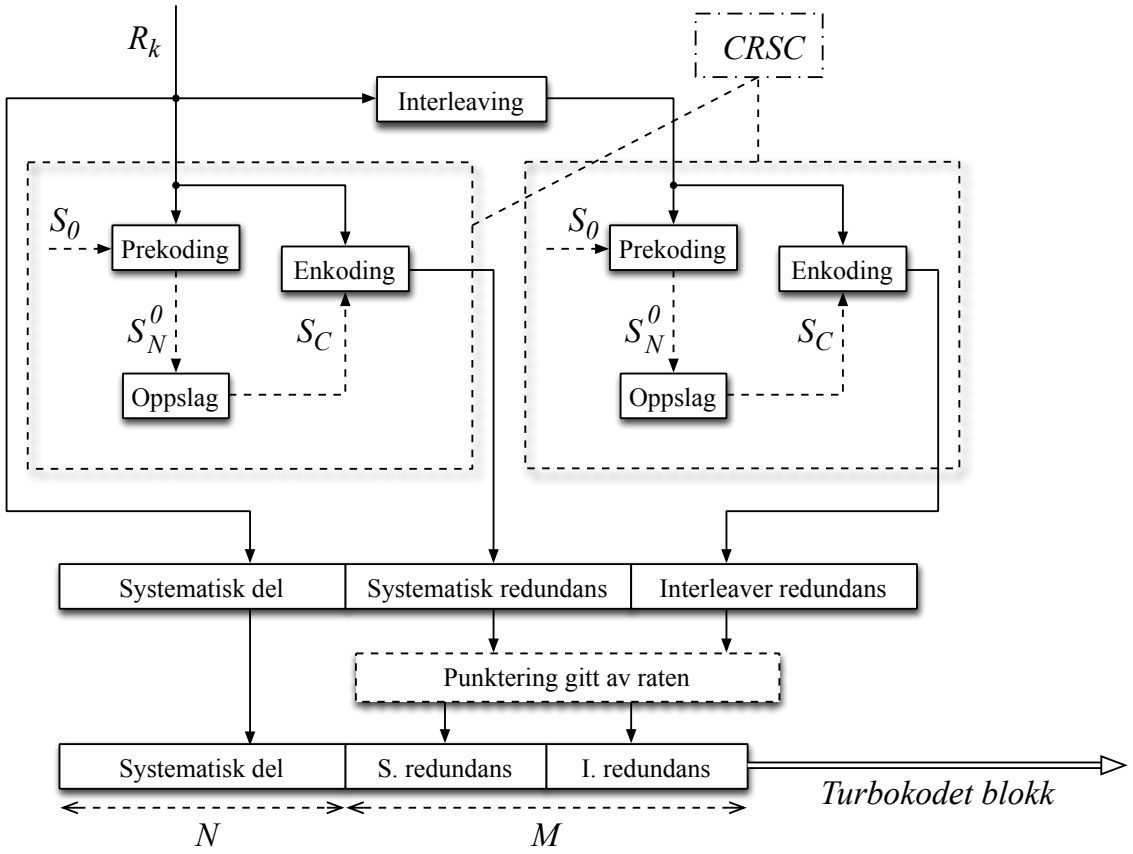
Etter hvert ble det nemlig oppdaget at mens permutasjonen i [23, 2] har $j = (i \cdot P_0 + P + 1) \bmod N$ i formelen for interleaveren, stemmer verifikasjonsdataene i [23] isteden overens med den følgende formelen: $j = (i \cdot P_0 + P) \bmod N$. Det viste seg at dette finnes i artikkelen til Berrou et al., som definerer turbokoden for DVB-RCS [12]. ETSI EN 301 790, som er standardiseringen av DVB-RCS, påstår at den følger denne permutasjonen [2], men det gjør den altså ikke. Hvorvidt



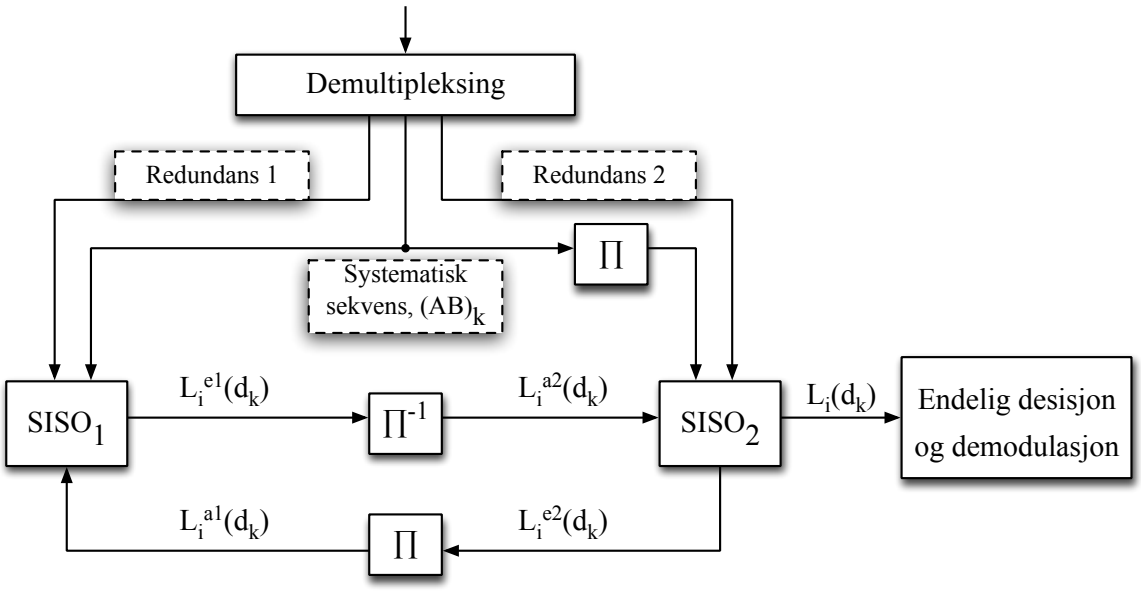
Figur 4.1: System-modellen for simuleringene

det er hos Berrou eller ETSI det har sneket seg inn en trykkfeil vites ikke, men detaljen er nok ikke så viktig for selve ytelsen. Den gjorde det imidlertid umulig å verifisere implementasjonen ved hjelp av [23].

Implementasjonen av DVB-RCS-turbokoden bruker derfor interleaverfunksjonen fra Berrous artikkel, og stemte med verifikasjonsdataene straks dette ble rettet opp. Dette gjør det rimelig sikkert at enkoder-funksjonen er programmert korrekt, mens dekoderen først og fremst vurderes



Figur 4.2: Generering av en turbokodet blokk



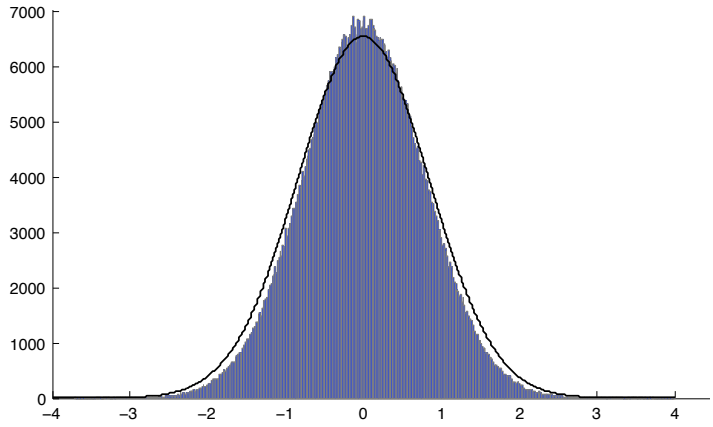
Figur 4.3: Iterativ dekoding av en turbokodet blokk

ut i fra ytelsen den leverer. DVB-RCS er som nevnt meget godt spesifisert, i [20] gis for eksempel en inngående gjennomgang av både DVB-RCS, Log-MAP, Max-Log-MAP og fremgangsmåten ved iterativ dekoding.

4.5 Vurdering av modellen

Modellen gjør en forenkling av de fleste detaljer i DVB-RCS som ikke har direkte med turbokoden å gjøre, og representerer nok ikke en fullgod test av hvordan koden vil virke i et fysisk system. På den annen side har alle de publiserte resultatene man har kommet over i løpet av oppgaven kun simulert turbokoden og ikke delene av systemet den skulle brukes i, så derfor ble dette gjort også i denne oppgaven.

Bruk av C kan medføre problemer, ettersom det har mindre innebygde funksjoner og feilsjekker. Man må derfor utvise påpasselighet ved minneallokering og ta seg av flere feilsjekker på egen hånd. C har heller ingen ferdig implementert generator for normalfordelte tall, slik som i Matlab. Av tidshensyn er det uansett bedre å generere verdiene i C, ettersom det ville tatt mye lenger tid å skifte mellom C og Matlab for hver eneste pakke. Sammenhengen mellom en Rayleigh-fordelt variabel og to uavhengige, normalfordelte variable utnyttes til å skape den normalfordelte støyen på kanalen. Koden ble verifisert underveis, og selv om støyen ikke blir en perfekt normalfordeling viser histogrammet i figur 4.4 at den ser ut til å være nærme nok.



Figur 4.4: Histogram over normalfordelte variable generert av implementsjonen, plottet mot en korrekt normalfordeling med samme varians

I C ble `srand(time(NULL))` brukt for å starte pseudorandom-generatoren på et tilfeldig sted for hver simulering. Om man ikke gjør det, starter nemlig `rand()` på den samme, forhåndsbestemte plassen hver gang, og siden randomgeneratoren i C er realisert som et langt skiftregister (og dermed er både deterministisk og periodisk), vil alle simuleringer bli like. En litt merkelig sak ble oppdaget i forbindelse med bruk av `srand(time(NULL))`. Til å begynne med var funksjonskallet plassert i en løkke helt ned på pakkenivå, som gjorde at det ble kjørt svært ofte. Dette gav langt bedre ytelse enn det som skulle være mulig, med N=212, R=1/2 og SNR på 1.5 dB fikk man for eksempel FER=10^{-5}, som i virkeligheten burde være nærmere FER=10^{-2}. En kjapp kontroll av støyen viste at den ikke på langt nær var normalfordelt lenger, og den så heller ikke ut til å ha særlig høye verdier. Dette forklarte saken, og ved å flytte kallet ut i den ytterste

SNR-løkka unngikk man problemet. Det er viktig å være klar over at nettopp slike ting kan skje med implementasjoner i et lavnivå-språk.

Ut i fra testing av de enkelte komponentene underveis og etterhvert også større deler av systemet, ser koden ut til å implementere både turbokoden i DVB-RCS og TurboΦ korrekt. Man kan naturligvis ikke garantere 100% for at en gitt kode er feilfri, men ut i fra de testene som er gjort underveis ser koden ut til å fungere etter intensjonen. Hvorvidt den er istand til å oppnå de virkelig gode ytelsene er en litt annen sak, men grunnstrukturen er etter alt å dømme korrekt.

Til slutt en kommentar om tidsbruk, i forhold til effektiv kode: Etter å ha kjørt en del lengre simuleringer, så det ut til at tidsbruken for en gitt blokklengde er en bortimot lineær funksjon av antall pakker. Ved hjelp av et par referansepunkter er det trivielt å regne ut dette forholdet, og for den ene serveren som ble brukt under simuleringen, en AMD64 dual core @ 2.0 GHz, kom det frem at den fikk unna ca 5 millioner pakker om dagen for N=752 og R=1/3. Det betyr at om man skal simulere FER=10^{-6} (som innebærer minst 10^6 genererte pakker) og i tillegg ønsker en sikkerhetsmargin på 50 feil, så må man regne med at simuleringen kan bli stående i opptil 10 dager, bare med det siste SNR-steget(!).

Denne tidshorisonten kombinert med at interleaver-parametrene for TurboΦ ble ikke funnet før mot slutten av oppgaveperioden, er hovedårsaken til at det ble valgt å sette en maksimal grense på antall simulerte pakker. Slik rekker man gjennom flere simuleringer, men konsekvensen er at de laveste feilratene ikke simuleres like nøyaktig, ettersom man vil oppleve at færre feil blir generert før simuleringen avsluttes. Avslutningsvis kan det nevnes at serveren hadde fire kjerner, slik at det kunne kjøres fire simuleringer om gangen.

Kapittel 5

Resultater og diskusjon

All simulering innebærer en viss grad av forenkling, så når man vurderer resultatenes gyldighet er det viktig å ta dette med i betraktning. For eksempel kan man ha oversett utslagsgivende detaljer under implementasjonen, som resulterer i tilsynelatende oppløftende resultater; dette er i realiteten ugyldige data. Om man får overraskende resultater er det derfor essensielt å overveie om alle viktige detaljer ved systemet er med, og om sammensetningen kan kunne skape uforutsette effekter. Med dette i bakhodet vil dette kapitlet presentere og drøfte resultatene fra simulering av DVB-RCS og TurboΦ. Først vil verifikasjonen av turbokoden i DVB-RCS gjennomgås, deretter sammenlignes dennes ytelse med simuleringene av TurboΦ. Diskusjonen setter resultatenes gyldighet, og setter de i sammenheng med egenskapene presentert i kapittel 3.

5.1 Simuleringsresultater

Simuleringene ble kjørt på servere disponert av NTNU, og resultatene ble lagret i .mat filer. Fra disse er det så hentet ut BER- og FER-verdier, plottet som en funksjon av SNR. Simuleringene har generelt blitt kjørt til et helt antall SNR i dB, men i noen av figurene er det allikevel kurver som stopper før dette. I disse tilfellene ble det maksimale antall pakker simulert uten at feil inntraff.

Verifikasjon av turbokoden i DVB-RCS

Rate	**Fra [12]**	**Sim. DVB-RCS**
1/2	2,3	2,4
3/4	3,9	3,95
6/7	5,2	5,25

Tabell 5.1: Verifikasjon av DVB-RCS mot tidligere publiserte resultater. Viser E_b/N_0(dB) @ FER=10^{-4} og N=212 (53 byte). Max-Log-MAP @ 8 iterasjoner.

Turbokoden i DVB-RCS er veletablert, og det finnes mye tidligere publiserte resultater. For å vise noe av verifikasjonen av simuleringene er det blitt valgt tre rater for blokklengden N=212,

sammenlignet over hvilket SNR-nivå som kreves for å oppnå FER=10^{-4}. Resultatene må kunne sies å være lovende, og vises i tabell 5.1. Ytelsen for de andre blokklengdene var av tilsvarende kvalitet, og koden ble med rimelig sikkerhet antatt å være en korrekt implementasjon av turbokoden i DVB-RCS. Den ble derfor utvidet til TurboΦ.

Verifikasjon av TurboΦ

Rate	**DVB-RCS**		**TurboΦ**	
	Fra [5]	Sim	Fra [5]	Sim
1/2	1,8	1,95	1,5	1,85
3/4	3,2	3,4	3,0	3,25

Tabell 5.2: Sammenligning mot tidligere publiserte resultater. Viser E_b/N_0(dB) @ FER=10^{-4} og N=752 (188 byte). Max-Log-MAP @ 8 iterasjoner.

Det eksisterer ikke det samme utvalget av simuleringsresultater for TurboΦ som for turbokoden i DVB-RCS, men det finnes allikevel en del i artiklene som introduserte TurboΦ [5, 25]. For å verifisere resultatene av simuleringene er det blitt valgt to rater for blokklengden N=752, sammenlignet over hvilket SNR-nivå som kreves for å oppnå FER=10^{-4}. Resultatene er noe varierende, og det kan se ut som implementasjonen må utbedres noe før man har en fullgod versjon av TurboΦ. Resultatene vises i tabell 5.2, og inneholder også resultater for turbokoden i DVB-RCS for de samme simuleringsparametrene.

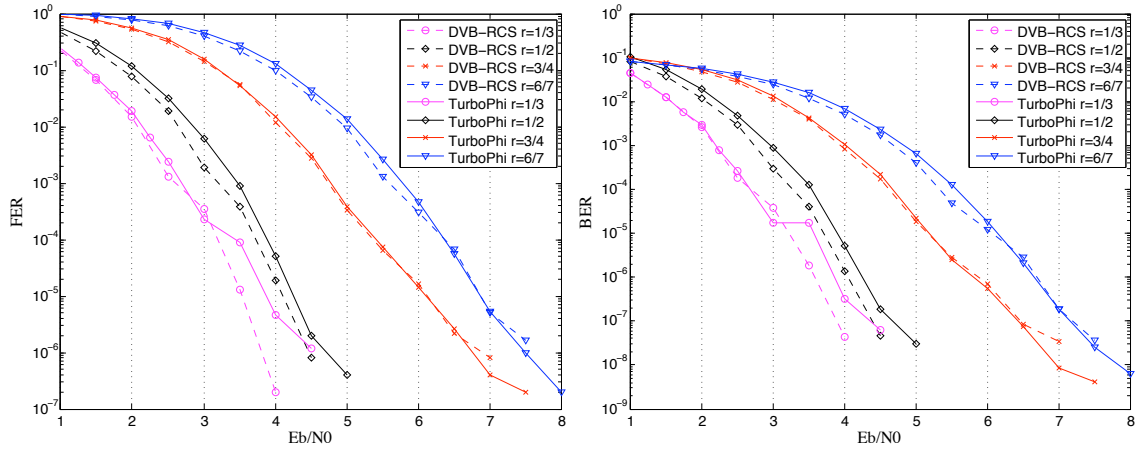
Sammenligning av DVB-RCS og TurboΦ

Rate	**N=48**		**N=212**		**N=752**	
	DVB-RCS	TurboΦ	DVB-RCS	TurboΦ	DVB-RCS	TurboΦ
1/3	3,2	3,5	1,8	1,7	1,6	1,2
1/2	3,7	3,85	2,4	2,45	1,95	1,85
3/4	5,4	5,4	3,95	4,15	3,4	3,25
6/7	6,4	6,4	5,25	5,15	4,8	4,3

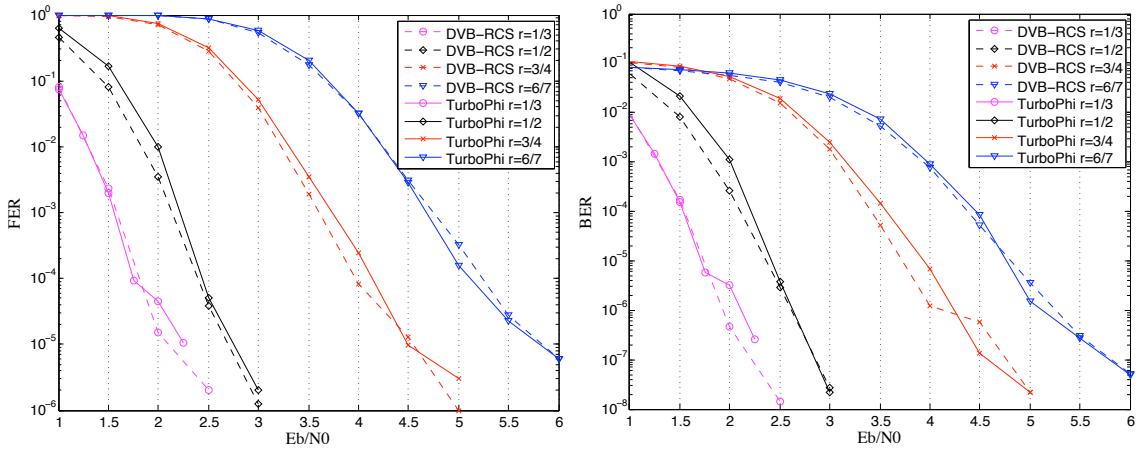
Tabell 5.3: Sammenligning av de to implementasjonene, med E_b/N_0(dB) @ FER=10^{-4}. Max-Log-MAP @ 8 iterasjoner.

For bedre å kunne bedømme implementasjonen av TurboΦ ble det gjennomført simuleringer av de to turbokodene for N=48, N=212 og N=752 (tilsvarende 12, 53 og 188 byte) for ratene R=1/3, 1/2, 3/4 og 6/7. Tabell 5.3 viser ytelsesforskjellen ved FER=10^{-4}, mens figurene 5.1, 5.2 og 5.3 viser hele plottet.

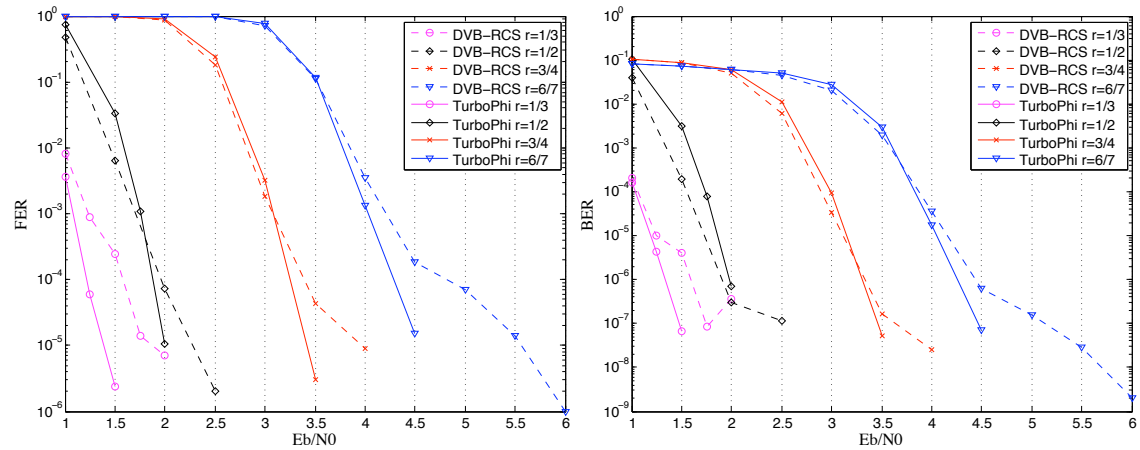
Det er rimelig klart at denne implementasjonen av TurboΦ ikke innfrir helt som forventet, og i flere av tilfellene ligger faktisk DVB-RCS-ytelsen foran eller veldig likt helt ned til FER=10^{-6}. Man ser dog at TurboΦ yter bedre enn turbokoden i DVB-RCS for enkelte av de høye ratene også her. For N=752 yter TurboΦ bedre enn turbokoden i DVB-RCS for alle rater, men sett over ett er ikke implementasjonen av TurboΦ tilstrekkelig nære tidligere publiserte resultater.



Figur 5.1: TurboΦ og DVB-RCS-turbo for N=48. Max-Log-MAP @ 8 iterasjoner.



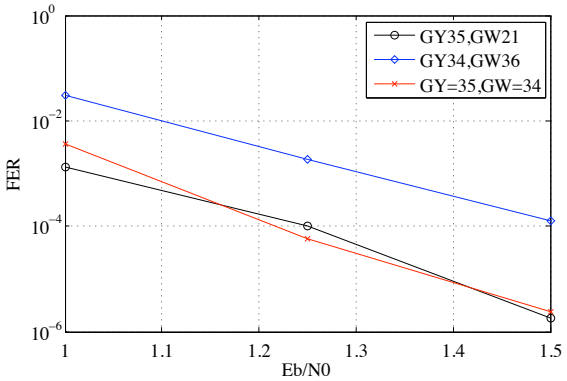
Figur 5.2: TurboΦ og DVB-RCS-turbo for N=212. Max-Log-MAP @ 8 iterasjoner.



Figur 5.3: TurboΦ og DVB-RCS-turbo for N=752. Max-Log-MAP @ 8 iterasjoner.

Andre simuleringer

Redundans-generatorene har mye å si for ytelsen, så det ble gjennomført simuleringer med forskjellige kombinasjoner av disse. Man kan også variere den rekursive generatoren, men om alle parametrene varieres blir det vanskeligere å slå fast både hvilke parametre som påvirker resultatene, og i hvilken grad de gjør det. G_R holdes derfor fast mens G_Y og G_W varieres. Fra figur 5.4 ser vi at generatoren som brukes i standarden presterer rimelig godt, den blir kun slått hårfint av settet $G_Y = \{35\}_8, G_W = \{21\}_8$ ved de laveste FER-verdiene.



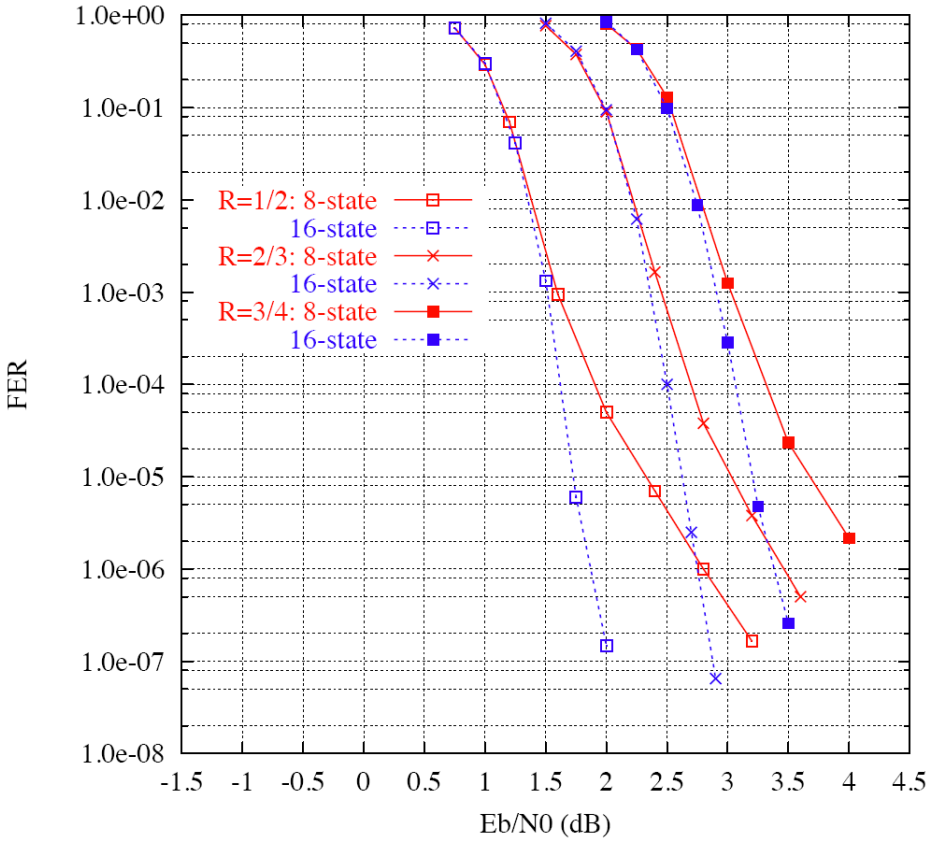
Figur 5.4: Sammenligning av generatorer for TurboΦ for $N = 752$ og $R = 1/3$. $G_Y = \{35\}_8$ og $G_W = \{34\}_8$ er brukt i de øvrige simuleringene. Max-Log-MAP @ 8 iterasjoner.

I denne sammenheng bør det nevnes at Berrou et al. gjorde simuleringer av forskjellige generatorer i [7], selv om dette da var med binære, ikke-sirkulære RSC-enkodere. For turbokoder med samme antall tilstander som TurboΦ (2^{K-1}=16) kom de frem til at $G_r = \{37\}_8$ og $G_Y = \{21\}_8$ gav de beste resultatene i forhold til FER. $G_r = \{23\}_8$ og $G_Y = \{35\}_8$ ble også utprøvd, men hadde dårligere asymptotisk ytelse. Dette andre settet hadde dog bedre konvergens-egenskaper, og er faktisk det samme som brukes i TurboΦ. Det kan derfor tenkes at konvergens-egenskapene gjør at dette settet har en bedre ytelse etter 8 iterasjoner, som er antallet de fleste publiserte resultater er gjort for. Det er for så vidt mulig at sammenhengen er tilfeldig, ettersom det er snakk om to ganske forskjellige turbokoder.

5.2 Diskusjon

Blokklengden på 188 byte (N=752) er den eneste hvor TurboΦ har bedre ytelse enn turbokoden i DVB-RCS for alle ratene, og her ser man også mer til den karakteristiske utflatingen av feilkurven for DVB-RCS-versjonen. Figur 5.5 gir en idé av hva man optimalt kunne tenkt seg av resultater for N=752, og det er rimelig klart at implementasjonen har mer å gå på. Simuleringene av TurboΦ i forhold til DVB-RCS varierer nokså voldsomt over de forskjellige blokklengdene, og selv om resultatene rundt FER=10^{-6} er mindre pålitelige enn resten av verdiene på grunn av valgene gjort i 4.2 er tendensen også tilstede for høyere (og mer pålitelige) FER-verdier. Det er derfor hevet over enhver tvil at blokklengden har en sterk påvirkning på resultatene, og siden systemmodellen er den samme for begge implementasjonene, må dette skyldes den eneste komponenten i turbokoden som varierer med blokklengden. Det er altså noe galt med *interleaveren* i TurboΦ.

ARP danner grunnlaget for permutasjonen i TurboΦ, og er en permutasjon som i utgangspunktet



Figur 5.5: Sammenligning av turbokoden i DVB-RCS og TurboΦ for N=752. Max-Log-MAP @ 8 iterasjoner. [5]

skal gi god ytelse. Dessverre oppgir [26] kun en fremgangsmåte for å kunne simulere seg frem til gode parametre, noe som var såpass omfattende og tidkrevende at det falt utenfor omfanget for denne oppgaven. En kort gjennomgang av fremgangsmåten gis dog her: Først settes noen grunnleggende krav til interleaveren, og ved hjelp av disse finner man frem til flere sett av mulige interleaver-parametre. Disse må deretter testes utførlig mot mulige feilmønstre. Et uttømmende søk over alle disse er naturlig nok ikke praktisk gjennomførbart, som regel tester man derfor over et sett av de mest utslagsgivende av mønstrene. Dette innebærer som regel mange av de korteste variantene, i tillegg til feilmønstre sammensatt av noen av disse kortere feilmønstrene, ettersom slike sammensatte mønstre kan medføre en kraftig reduksjon av ytelsen. ARP danner også grunnlaget for interleaveren i DVB-RCS, men her er parametrene oppgitt, og har også vært gjennom grundige tester og søk. De har derfor en verifisert kvalitet.

Man fant i utgangspunktet svært få verdier for parametrene for ARP-permutasjonen i TurboΦ, men etterhvert ble det i [24] oppdaget et sett for de aller fleste blokklengdene. Det ble dog ikke funnet noen resultater man kunne verifisere disse med, og simuleringene tyder også på at disse parametrene er av varierende kvalitet. Som nevnt i kapittel 3 er det interleaveren som i det store og hele bestemmer ytelsen turbokoden kan oppnå, som betyr at varierende kvalitet på interleaver-parametrene vil gi direkte utslag i form av varierende ytelse. På tross av dette var ikke resultatene for TurboΦ så altfor ille, selv om de burde prestert enda bedre i forhold til DVB-RCS for flere av blokklengdene. Resultatene for N=752 er for eksempel gode i forhold til turbokoden i DVB-RCS, og viser heller ikke antydning til feilgulvet man ser i resultatene for DVB-RCS.

Totalt sett har implementasjonene vært vellykket, i særdeleshet kommer turbokoden i DVB-RCS svært nærme resultatene fra tidligere publikasjoner. Implementasjonen av TurboΦ er ikke like god, og sliter som nevnt med varierende kvalitet på interleaver-parametrene. Resultatene kan derfor ikke sies å demonstrere denne kodens fulle potensiale.

Kapittel 6

Konklusjon

Turbokoder er et veletablert alternativ til feilkorrigerende kanalkoder, og brukes idag i en rekke applikasjoner. De revolusjonerende resultatene presentert i 1993 fanget umiddelbart fagfeltets interesse, og fortsatt virker det å være et svært aktivt forskningsområde. Denne oppgaven var en videreføring av litteraturstudiet fra prosjektet [1], og hadde som mål å implementere og gjennomføre simuleringer av TurboΦ og turbokoden i DVB-RCS. Ytelsen i form av pakkefeilraten over en kanal med AWGN skulle simuleres for forskjellige blokklengder og koderater, og verifiseres mot publiserte resultater.

For å kunne gi en grundig gjennomgang av de aktuelle turbokodene ble først nødvendige begreper etablert. Deretter ble den originale turbokoden analysert som en introduksjon til turbo-konseptet. Den består av to parallellkonkatenerte RSC-enkodere, skilt av en interleaver. Dekodingen er iterativ med utveksling av soft-informasjon mellom to SISO-moduler. De duo-binære turbokodene i DVB-RCS og TurboΦ ble deretter gjennomgått, med vekt på fordelene ved bruk av ikke-binære turbokoder. De viktigste i så måte er mindre følsomhet for både sub-optimal dekoding og punktering, større total MHD samt mindre antydning til feilgulv.

Mye av forskningen på turbokoder går på forbedring av interleaverpermutasjonen, som er langt ifra trivielt. Interleaveren er svært sentral i forhold til hvilke ytelser turbokoder kan oppnå, fordi den innfører den klart største økningen i MHD i forhold til resten av komponentene i koden. MHD påvirker feilrettingspotensialet til koden, og dermed også feilgulvet som inntreffer ved svært lave feilsannsynligheter og lav SNR. Ikke-regulær interleaving er helt nødvendig for en god turbokode, fordi dette bryter opp sammensatte feilsekvenser som ellers ville gitt en drastisk senkning av MHD. Begge de implementerte turbokodene benytter ikke-regulære interleavere, men for TurboΦ viste det seg imidlertid vanskelig å finne parametre til interleaver-funksjonen. Noen ble omsider funnet i [24], men som simuleringene viser er disse av varierende kvalitet.

Det ble gitt en gjennomgang av den optimale dekodingsalgoritmen MAP og hvordan denne brukes for å generere ekstrinsikk informasjon i den iterative dekodingen. Denne ble utvidet og nøye gjennomgått for duo-binære koder, og også overført til det logaritmiske plan i form av Log-MAP. Forenklingen av denne versjonen, Max-Log-MAP, ble benyttet i simuleringene. Utledningen av denne ble derfor viet ekstra oppmerksomhet, blant annet i form av detaljer rundt den iterative dekodingen, initialisering og endelig desisjon. Forskjellen i SNR mellom optimal dekoding og Max-Log-MAP er kun 0,5 dB.

Implementasjonen av de to aktuelle turbokodene ble deretter gjennomgått. Det ble programmert

i en kombinasjon av Matlab og C, med simuleringen i C for å oppnå raskest mulig kjøretider. Matlab på sin side ble brukt til behandling og presentasjon av resultatene. For å rekke gjennom flere kombinasjoner av rate og blokklengder ble det besluttet å legge et tak på antall genererte pakker i hver simulering, men dette medførte også at resultatene på de laveste FER-nivåene ble mindre nøyaktige.

Verifikasjon av enkoderen i DVB-RCS ble vanskeliggjort av en uoverensstemmelse mellom standardiseringsdokumentene ETSI EN 301 790 og ETSI TR 301 790, men da dette ble oppdaget og håndtert gikk verifikasjonen greit. Gyldigheten av hele modellen ble også vurdert, og et viktig poeng her er at programmering i lavnivåspråk kan medføre uforutsette bivirkninger. På et tidspunkt ga for eksempel turbokoden en umulig god ytelse, som viste seg å skyldes at implementasjonen av normalfordelingen i C ble påvirket av en tilsynelatende uavhengig detalj. I C må man ofte skrive slike funksjoner selv, og det er derfor ekstra viktig å verifisere at alle delene av koden, inkludert støttefunksjonene, fungerer som de skal etter at det er gjort endringer. Slik ble også denne feilen oppdaget og utbedret. Etter å ha verifisert system-modellen ble simuleringene av turbokoden i DVB-RCS utført.

Siste del av verifikasjonen av turbokoden i DVB-RCS var sammenligning mot tidligere publiserte resultater. I så måte var simuleringene for DVB-RCS meget bra, og på FER=10^{-4} var de på det verste bare 0,1 dB dårligere for N=212 og 0,2 dB dårligere for N=752. Man anså derfor denne koden som en verifisert implementasjon av DVB-RCS, og på bakgrunn av dette ble implementasjonen utvidet til TurboΦ. Dette innebar kun utskiftning av selve turbokoden, slik at begge turbokodene benyttet den nøyaktig samme system-modellen. Dette betød at sammenligningen av de to implementasjonene kun ville påvirkes av selve turbokodene, og ikke andre eventuelle unøyaktigheter i system-modellen.

For blokklengder på 188 byte var ytelsen til TurboΦ på sitt beste, her var implementasjonen bedre enn DVB-RCS for alle ratene. Det burde den vært for alle blokklengdene, men for blokker på 12 og 53 byte presterte den faktisk noe dårligere enn turbokoden i DVB-RCS ved enkelte av ratene. Høyst sannsynlig skyldes dette interleaveren, siden parametrene i den tilpasses til hver enkelt blokklengde, og det er nettopp over blokklengdene ytelsen varierer mest. Interleaveren i DVB-RCS er fullstendig spesifisert, mens TurboΦ kun oppgir formelen for parametrene [5], ikke parametrene i seg selv. Ettersom det ikke ble funnet noen resultater for parametrene i [24] har man ingenting å sammenligne mot, allikevel er det rimelig sikkert at varierende kvalitet på disse parametrene medførte varierende ytelse i implementasjonen av TurboΦ.

Alt i alt må oppgaven kunne sies å ha nådd sine målsetninger: Det ble utviklet effektive simuleringsprogrammer for både turbokoden i DVB-RCS og TurboΦ, som også ble verifisert mot publiserte resultater. Deres ytelse i form av FER og BER ble simulert og sammenlignet for forskjellige koderater og blokklengder, og det ble også eksperimentert med andre parametre. Simuleringene legger seg tett opptil tidligere resultater for turbokoden i DVB-RCS, men bare til dels for TurboΦ. Sistnevnte har altså rom for forbedringer, spesielt med tanke på å bestemme gode verdier for parametrene i interleaveren.

God ytelse for $10^{-6} < FER < 10^{-3}$ var viktig for STM Norge, og i seksjon 3.3.2 drøftes duobinære turbokoder med 16 tilstander som det potensielt beste valget for $10^{-9} < FER < 10^{-4}$. Ettersom TurboΦ er en slik kode, er denne oppgaven et godt utgangspunkt for videre evaluering av kodekandidaten for neste generasjons DVB-RCS.

6.1 Videre arbeid

- Det desidert viktigste for videre evaluering av TurboΦ er søk etter og verifisering av flere og bedre parametre for interleaveren. Den baserer seg som nevnt på ARP, og i [16, 27] presenteres fremgangsmåter for å bestemme og måle parametrene i forhold til økningen de potensielt kan gi i MHD. Endelig verifikasjon av parametrene utføres ved simulering med forskjellige parametersett, og sammenligning mot publiserte resultater for TurboΦ. Søk etter og verifikasjon av nye interleaver-parametre ble for omfangsrikt for denne oppgaven, men er det første som burde gjøres av videre arbeid. Parametrene må som nevnt tilpasses individuelt for hver blokklengde.
- Etter å har funnet bedre parametre for interleaveren, kan man gjennomføre simuleringer av rater og blokklengder som ikke finnes i DVB-RCS. Det bør også simuleres med flere pakker for de laveste FER-verdiene, ettersom de kan være noe unøyaktige i den nåværende implementasjonen. Dekodingen kan utvides slik at man også kan velge Log-MAP som dekodingsalgoritme, som i så fall burde gi en ytelsesforbedring i SNR på ca 0,5 dB.

Forkortelser

APP	A Posteriori Probability
ARP	Almost Regular Permutation
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
CRC	Cyclic Redundancy Check
CRSC	Circularly Recursive Systematic Convolutional
DVB	Digital Video Broadcasting
DVB-RCS	Digital Video Broadcasting - Return Channel Satellite
FC	Finite Codeword
FER	Frame Error Rate
GEO	Geostationary Earth Orbit
GPS	Global Positioning System
HDD	Hard Decision Decoding
IP	Internet Protocol
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
MAP	Maximum A Posteriori
MHD	Minimum Hamming Distance
ML	Maximum Likelihood
MLLR	Multidimensional Log Likelihood Ratio
MPEG	Moving Picture Experts Group
NCC	Network Control Centre

NSC	Non-Systematic Convolutional
QPSK	Quadrature Phase-Shift Keying
RCST	Return Channel Satellite Terminal
RSC	Recursive Systematic Convolutional
RTZ	Return To Zero
SCCC	Serial Concatenated Convolutional Code
SDD	Soft Decision Decoding
SIN	Satellite Interactive Network
SISO	Soft In/Soft Out
SNR	Signal to Noise Ratio

Bibliografi

[1] Christian Nordahl. Turbokoder for adaptivt kodet modulasjon i DVB-RCS. 2007.

[2] DVB. *Interaction Channel for Satellite Distribution Systems.* ETSI EN 301 790, des 2000.

[3] Keattisak Sripimanwat (Editor). *Turbo Code Applications - A Journey From a Paper to Realization.* Springer, 2005.

[4] C. Berrou, M. Jézéquel, C. Douillard, and S. Kerouédan. The Advantages of Non-Binary Turbo Codes. In *IEEE Information Theory Workshop*, sept, 2001.

[5] C. Berrou, R. De Gaudenzi, C. Douillard, G. Gallinaro, R. Garello, D. Giancristofaro, A. Ginesi, M. Luise, G. Montorsi, R. Novello, and A. Vernucci. High Speed Modem Concepts and Demonstrator for Adaptive Coding and Modulations with High Order in Satellite Applications. In *ESA conference on Signal Processing for Space Communiactions (SPSC)*, Catania, Italia, 2003.

[6] C. E. Shannon. A Mathematical Theory of Communication. *Bell Sys. Tech. J.*, 1948.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. In *IEEE Int. Conf. Communications (ICC)*, Geneve, Sveits, mai, 1993.

[8] Jeffrey G. Andrews, Arunabha Ghosh, and Rias Muhamed. *Fundamentals of WiMAX.* Prentice Hall, 2007.

[9] 3GPP2. *Physical Layer Standard For CDMA2000 Spread Spectrum Systems.* rev. D, 3GPP2 C.S0002-D, 2004.

[10] DVB. *Framing Structure, Channel Coding and Modulation For 11/12 GHz Satellite Services.* ETSI EN 300 421.

[11] DVB. *Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications.* ETSI EN 302 307.

[12] C. Douillard, M. Jézéquel, C. Berrou, N. Brengarth, J. Tousch, and N. Pham. The Turbo Code Standard for DVB-RCS. In *2nd International Symposium on Turbo Codes & Related Topics*, Brest, Frankrike, 2000.

[13] C. Berrou and A. Glavieux. Near Optimum Error Correcting Coding And Decoding: Turbo-Codes. *IEEE Transactions On Communications*, okt, 1996.

[14] Andrea Goldsmith. *Wireless Communications.* Cambridge University Press, 2005.

[15] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, vol. IT-13, 1967.

[16] C. Berrou, Y. Saouter, C. Douillard, S. Kerouédan, and M. Jézéquel. Designing Good Permutations for Turbo Codes: Towards a Single Model. In *IEE Colloquium: Turbocodes in Digital Broadcasting - Could it Double Capacity?*, 1999.

[17] Giancristofaro D. and Bartolazzi A. Novel DVB-RCS Standard Turbo Code: Details and Performances of a Decoding Algorithm. In *ESA & IT: Seventh International Workshop on Digital Signal Processing Techniques for Space Communications (DSP 2001)*, 2001.

[18] C. Berrou, R. Pyndiah, P. Adde, C. Douillard, and R. Le Bidan. An Overview of Turbo Codes and Their Applications. In *The European Conference on Wireless Technology, 2005*, 2005.

[19] P. Robertson, E. Villebrun, and P. Hoeher. A Comparison og Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. In *Proc. IEEE Int. Conf. Commun*, 1995.

[20] M. Reza Soleymani, Y. Gao, and U. Vilaipornsawai. *Turbo Coding for Satellite and Wireless Communication*. Kluwer Academic Publishers, 2002.

[21] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Transactions on Information Theory*, vol. IT-20, 1974.

[22] J. Hagenhauer, P. Robertson, and L. Papke. Iterative ("Turbo") Decoding of Systematic Convolutional Codes with the MAP and SOVA algorithms. In *Proc. ITG'94*, 1994.

[23] DVB. *Interaction Channel for Satellite Distribution Systems; Guidelines for the use of EN 301 790*. ETSI TR 301 790, sept 2006.

[24] Motorola. *ARP Interleaver Design for LTE*. 3GPP TSG RAN WG1, R1-070061.zip.

[25] S. Benedetto, C. Berrou, C. Douillard, R. Garello, D. Giancristofaro, A. Ginesi, L. Giugno, M. Luise, and G. Montorsi. MHOMS: High Speed ACM Modem for Satellite Applications. In *ESA conference on Signal Processing for Space Communiactions (SPSC)*, Catania, Italia, 2003.

[26] C. Berrou, C. Douillard, and M. Jézéquel. Designing Turbo Codes for Low Error Rates. In *IEEE Int. Conf. Communications (ICC)*, 2004.

[27] A. Nimbalker, Y. Blankenship, B. Classon, and T.K. Blankenship. PHY 40-1 - ARP and QPP Interleavers for LTE Turbo Coding. *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, 2008.