Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Kybernetikk	Vår semesteret, 2010 Åpen / ~~Konfidensiell~~
Forfatter: Pål Henning Olsen	…… (signatur forfatter)
Fagansvarlig: John Håkon Husøy Veileder(e): Svein Fagerlund	
Tittel på masteroppgaven: Multisensor Målfølging Av To Flygende Mål I Formasjon Engelsk tittel: Multisensor Tracking Of Two Flying Targets In Formation	
Studiepoeng: 30	
Emneord: Målfølging, dataassosiasjon, Kalmanfilter, GNN, MHT, Radar, Multisensor	Sidetall: …… + vedlegg/annet: …… Stavanger, …….. dato/år

Universitetet i Stavanger

Det Teknisk-Naturvitenskapelige Fakultet

Multisensor Målfølging Av To Flygende Mål I Formasjon

Skrevet av:
Pål Henning Olsen

Veileder:
Svein Fagerlund

Fagansvarlig:
John Håkon Husøy

15. juni 2011

Sammendrag

Denne oppgaven omhandler målfølging i multisensorsystemer. De ulike delene til målfølgingssystem blir presentert samt en gjennomgang av ulike algoritmer benyttet innenfor målfølging. I første del av oppgaven ser man nærmere på målfiltreringsalgoritmer. Det blir her gjennomgått standard Kalmanfilter, Utvidet Kalmanfilter, "Unscented Kalmanfilter", samt informasjonsvarianten av disse tre. Disse algoritmene har deretter blitt implementert i MatLab, og simulert mot hverandre. Simuleringen viser at de forskjellige filterene gir et veldig likt resultat når det kommer til målfiltrering.

I del to presenteres man for ulike former for assosiasjonsalgoritmer og teorien rundt disse. GNN og MHT algoritmen er deretter blitt implementert i MatLab og simulert under varierende forhold.

Simulatoren benyttet for assosiasjonstestene har også blitt utvidet til å kunne benytte fire sensorer, som returnerer observasjoner fra det samme målet, men fra forskjellige vinkler. Man gjennomgår typisk arkitektur for et multisensor system, samt noen av problemene knyttet til å benytte flere sensorer.

Forord

Dette er en oppgave skrevet i tilknytning til fullføring av en mastergrad i kybernetikk ved Universitet i Stavanger. Arbeidet har blitt fullført våren 2011. Oppgaven er gitt av Kongsberg Defence & Aerospace (KDA), divisjon Integrated Defence Systems (IDS) og tar i hovedsak for seg målfølging i multisensorsystemer.

Denne rapporten gir et innblikk i ulike algoritmer som er aktuelle for et målfølgingssystem. Den presenterer flere ulike problemstillinger som oppstår ved bruk av flere radarsystemer i et nettverk.

Jeg vil med dette rette en takk til min fagligansvarlig ved universitet John Håkon Husøy. En stor takk også til Svein Fagerlund ved KDA for en utfordrende oppgave og all hjelpen han har bidratt med underveis i oppgaven.

Pål Henning Olsen
Stavanger 15. juni 2011

Innhold

Kapittel 1

Innledning

1.1 Bakgrunn for oppgaven

Denne oppgaven har blitt gitt av Kongsberg Defence & Aerospace. Selskapet er verdensledende innen utviklingen av luftvernsystemer. I etthvert luftvern system har man en eller flere sensorer som benyttes til å detektere mål innenfor et geografisk område. For at man til enhver tid skal ha best mulig sjanse til å reagere på unormal aktivitet innefor det beskyttede luftrommet er det viktig at man klarer å utnytte den informasjonen man har tilgjengelig best mulig. Det er her ønskelig å undersøke situasjoner der man har mål som flyr nær hverandre i kortere eller lengere tid. Spesielt gjelder dette dersom målene befinner seg i ytterkant av sensorenes rekkevidde, hvor man ikke alltid får deteksjoner av målene til enhver tid.

1.2 Oppgavebeskrivelse

En litteraturstudie har blitt utført der man har sett nærmere på ulike algoritmer for dataassosiasjon. To av algoritmene har deretter blitt valgt ut til å bli implentert i en simulator der man tester egenskapene og ser på problemområder knyttet til algoritmen. Spesielt har man ønsket å undersøke dersom man har to mål i tett formasjon i ytterkant av sensorens rekkevidde. Man får her en lav sannsynelighet for deteksjon, noe som gjør at målfølgingen ofte feiler. De to assosiasjonsalgoritmene har også blitt simulert i et miljø der man benytter 4 sensorer. Selv om man umidelbart skulle tro at flere sensorer gir bedre resultat, så fører dette også til en del problemer som blir nærmere undersøkt, og en løsning på noen av de presentert.

1.3 Rapportstruktur

Det blir her gitt en kort oversikt over de ulike delene av rapporten.

1. **Innledning.** En kort innføring i oppbygning av et målfølgingssystem og dets ulike komponenter.
2. **Målfiltrering.** Teori knyttet til målfiltrering. Det blir her gitt en gjen-

nomgang av ulike algoritmer benyttet til målfiltrering samt noen problemer og løsninger på disse.

3. **Dataassosiasjon.** Fire forskjellige algoritmer som kan benyttes til assosiasjon blir presentert, sammen med fordelene og ulempene med disse.

4. **Multisensor Målfølging.** Systemet blir nå utvidet til å inkludere flere sensorer. Dette medfører en del ekstra problemer man må ta hensyn til.

5. **Resultater og diskusjon.** Det har blitt implementert flere simulatorer i MatLab der man har testet de forskjellige algoritmene presentert tidligere. Resultatene av disse blir her diskutert nærmere.

1.4 Historie

Radarens opprinnelse kan ikke spores tilbake til en bestemt person. De første forsøkene man kjenner til er fra 1887. Den tyske fysikeren Heinrich Hertz eksperimenterte med elektromagnetisk bølger. Han fant ut at bølgene bevegde seg gjennom noen typer materialer, mens de ble reflektert fra andre [39]. Dette ble i 1904 utnyttet av ingeniøren Christian Hülsmeyer. Han tok da patent på sitt "Telemobiloskop". Apparatet var ment brukt ombord på skip for å detektere andre skip i området når sikten var dårlig. Han kunne detektere skip i en avstand på opptil 3 km. De første modellene ga ingen informasjon om avstanden til andre skip, kun at et annet objekt var detektert i området. Hülsmeyer tok senere et patent på et trianguleringssystem som gjorde at man kunne beregne avstanden til det detekterte skipet [39]. Dessverre var det liten interesse for apparatet, og det ble snart glemt. I tiden frem mot andre verdenskrig ble det utviklet flere ulike radarsystemer. Innen 1939 hadde flere land utviklet sine egne radarsystemer, det meste kjente av disse var det britiske "Chain Home" systemet.

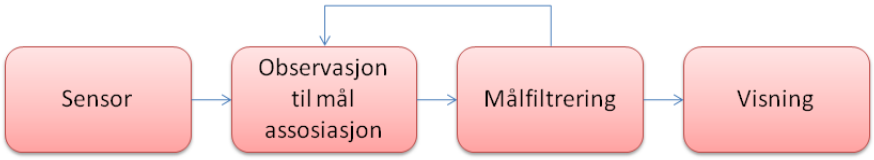
"Chain Home" var en stor suksess for britene. Det muligjorde at de britiske styrkene hadde kjennskap til de fleste tysk angrep, og hadde tid til å sende opp sine egne luftstyrker for å møte de tyske bombeflyene. Da "Chain Home" ble utviklet hadde man dårlig tid, og systemet hadde ofte feilrapporteringer. Radartårnene i "Chain Home" lignet lite på dagens radarer. De benyttet en fast antenne montert på et 110 meter høyt tårn til å sende ut en puls med ca 100° bredde. Retursignaler ble deretter plukket opp av to antenner som stod vinkelrett på hverandre. Dette gjorde at hver mottakerantenne mottok signalet med forskjellig signalstyrke. En operatør måte deretter manuelt beregne vinkelen til målet ved å sammenligne de to signalene. Tilsvarende kunne man også finne elevasjonen til målet vha et annet sett med antenner. Man kunne også finne avstanden ved å se på tiden signalet brukte på å reise mellom radaren og målet [38].

Det første målfølgingssystemet ble påtenkt i 1955 av Nelson Wax. Wax observerte radaroperatører som tegnet linjer mellom punkter på radarskjermen. Slik kunne de forutsi den fremtidige posisjonen til flyene. Han utviklet deretter flere matematiske modeller for å automatisere denne prosessen. Men selv om man nå hadde systemer som automatisk kunne beregne banene til fly manglet man systemer som knyttet sammen radarobservasjoner og eksisterende mål. Arbeidet med dette startet i 1964 da Robert Sittler benyttet bayesiske formuleringer for å løse problemet. Kombinert med introduksjo-

nen av kalmanfilteret ble det i begynnelsen av 1970 utviklet flere teorier av Yaakov Bar-Shalom og Robert Singer for å løse problemet med å følge flere fly på en gang [6]. Disse har etterhvert lagt mye av grunnlaget for moderne målfølging.

1.5 Oppbyggning

Det blir her presentert en forenklet oversikt over hva et målfølgingssystem er og hvilke komponenter det består av. En oversikt over et enkelt målfølgingssystem kan ses i figur 1.1. En sensor benyttes til å detektere posisjonen til eventuelle fly, som befinner seg innenfor området som sensoren overvåker. Sensoren vil gi ut observasjonsdata til målfølgingssystemet. En observasjon-til-mål assosiasjon utføres deretter for å koble sammen observasjonene med allerede bekreftede mål. De assosierte observasjonene sendes deretter gjennom et målfølgingsfilter, før det endelige resultatet presenteres til brukeren.



Figur 1.1: Forenklet oppbygning av et målfølgingssytem

1.5.1 Sensor

Den første delen i systemet er sensoren. I et multisensor system trenger man ikke ha bare sensorer av samme type. Man kan kombinere ulike typer sensorer. Fordelen med dette er at ulike sensorer ofte benytter ulike metoder for å detektere et objekt. Det er derfor mulig at en sensortype klarer å detektere objektet, mens en annen sensortype ikke detekterer objektet. Et eksempel på dette kan være kombinasjonen av en optisk sensor (kamera), med en radar. Dersom man har en dag der man har et tykt skydekke, og et fly som flyr over skydekket vil ikke den optiske sensoren detektere flyet, radaren har derimot ikke noe problem med dette. En annen fordel med kombinasjon av ulike sensorer er knyttet til nøyaktigheten til sensoren. En radar vil typisk gi en god nøyaktighet på avstanden til et objekt, mens en optisk sensor vil være lite egnet til å angi avstand.

I denne oppgaven benyttes kun radar som sensor. En typisk radar består av en roterende antenne. Mens antennen roterer rundt sender den ut elektromagnetiske pulser. Disse pulsene vil reflektere fra eventuelle objekter de

treffer, men også fra omgivelsene. En rotasjon av radarantennen kalles ofte for et scan. Mens radaren utfører et scan benyttes signalprosessering til å skille ut mulige fly fra bakgrunnsstøyen. Moderne radar benytter en teknikk kalt dopplerfiltrering. Dopplerfiltrering benyttes for å avgjøre om avstanden til en deteksjon endrer seg. Dette gjør at moderne systemer er veldig effektive på å filterer vekk feildeteksjoner. For hvert scan som radaren utfører generers informasjon om de forskjellige deteksjonene. Hver deteksjon blir i denne oppgaven kalt for en observasjon. En observasjon inneholder informasjon om tidspunktet for deteksjon, og posisjonsinformasjon om målet relativt til radaren. En del radarer kan også gi informasjon om dopplerrate og signal-til-støy forhold. Dette er ikke tatt hensyn til i denne oppgaven. Posisjonsinformasjonen til en observasjon inneholder avstand, asimut og elevasjon. Disse blir nærmere forklart i seksjon 2.1.

1.5.2 Assosiasjon

For hvert scan av radaren, vil målfølgingssystemet motta et sett med observasjoner. Hver observasjon representerer det radaren tror kan være et mulig mål. Målfølgingssystemet vet ikke noe om opprinnelsen til disse observasjonene. Man må derfor først knytte sammen observasjoner med eksisterende informasjon i området man overvåker. Det er her dataassosiasjonsalgoritmen benyttes. I dataassosiasjon benytter man forskjellige metoder til å knytte sammen observasjoner med eksisterende mål. Denne informasjonen blir deretter videresendt til målfølgingsfilteret.

1.5.3 Målfølgingsfilter

Alle observasjonene som radaren generer inneholder en del usikkerhet i posisjonen man måler. Målfølgingsfilteret har som oppgave å minimere effekten av denne usikkerheten så mye som mulig. Målfølgingsfilteret kan også forutse fremtidig posisjon til målet. Dette gjør at man får en mye bedre representasjon av faktisk posisjon for detekterte mål. Det er også målfølgingsfilteret sin oppgave å kombinere observasjoner fra flere sensorer til et felles estimat. For at målfølgingsfilteret skal gjøre en god jobb er det avhengig av at assosiasjonsprosessen har klart å koble riktig observasjon til riktig mål. Til gjengjeld er assosiasjonsprosessen avhengig av at målfølgingsfilteret gir en god prediksjon av fremtidig posisjon til et mål for å kunne utføre en god kobling av

observasjon og mål! Denne koblingen gjør at assosiasjon og målfølgingsfilter er nært knyttet sammen. En forbedring i den ene kan derfor ofte føre til at den andre yter bedre.

Kapittel 2

Målfølgingsfilter

2.1 Koordinatsystemer

I denne oppgaven benyttes to forskjellige koordinatsystemer til å representere posisjon til et mål, henholdsvis kartesiske koordinater og sfæriske koordinater. Det blir her gitt en kort introduksjon til de to koordinatsystemene og hvordan man kan konvertere koordinater mellom kartesiske- og sfæriske koordinater.

Til å representere den fysiske plasseringen til en radar og global posisjon til et mål benyttes kartesiske koordinater. Dette fordi det er enkelt å visualisere og kjent for de fleste personer. I denne oppgaven benyttes x-aksen positiv i nordlig retning, y-aksen positiv i østlig retning og z-aksen positiv nedover. Dette kalles ofte for et NED-system (North, East, Down) [1].

Alle radarene i oppgaven benytter sfæriske koordinater når de måler posisjonen til et mål. Hver eneste radar har altså sitt eget lokale koordinatsystem og alle observasjoner som radaren gjør er relativt til dette. Sfæriske koordinater måles i henholdsvis avstand, asimut og elevasjon. Avstand er som ordet tilsier avstanden fra koordinatsystemets senter til målet. I denne oppgaven er asimut definert som 0 radianer i nordlig retning. Den er deretter økende med klokka. Intervallet er fra 0 til 2π. Elevasjon er definert som vinkelen mellom xy-planet og avstanden til målet. Elevasjon har verdien 0 dersom målet befinner seg i xy-planet og $\pi/2$ dersom målet er rett over senter av koordinatsystemet.

Følgende ligninger benyttes for å konvertere fra det ene koordinatsystemet til det andre. Her representerer x_1, y_2 og z_3 de kartesiske koordinatene, R, Az og El er henholdsvis avstand, asimut og elevasjon i sfæriske koordinater. x_0, y_0 og z_0 er de globale kartesiske koordinatene for senteret til det lokale koordinatsystemet som den aktuelle radaren benytter.

$$x_1 = R\cos(El)\cos(Az) + x_0 \tag{2.1}$$

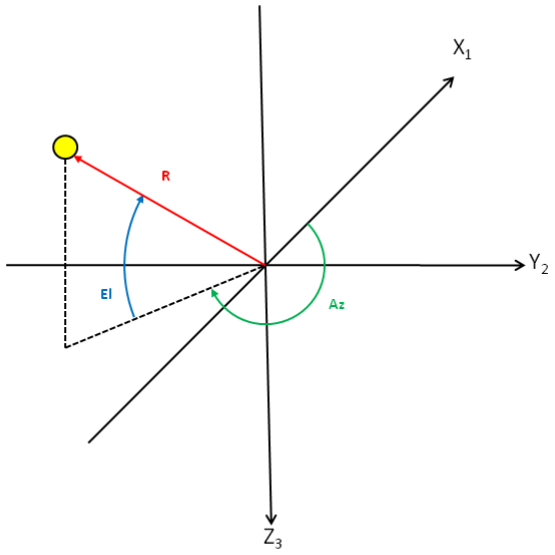
$$y_2 = R\cos(El)\sin(Az) + y_0 \tag{2.2}$$

$$z_3 = -R\sin(El) + z_0 \tag{2.3}$$

$$R = \sqrt{(x_1 - x_0)^2 + (y_2 - y_0)^2 + (z_3 - z_0)^2} \tag{2.4}$$

$$Az = atan(\frac{(y_2 - y_0)}{(x_1 - x_0)}) \tag{2.5}$$

$$El = atan(\frac{-(z_3 - z_0)}{\sqrt{(x_1 - x_0)^2 + (y_2 - y_0)^2}}) \tag{2.6}$$



Figur 2.1: Koordinatsystemene slik de er definert i oppgaven.

2.2 Modell

Jobben til et målfølgingsfilter er å forbedre vår oppfatning av virkelig tilstand til et mål. Tilstanden til et mål vil i et målfølgingssystem i hovedsak være posisjonen til målet, men man kan også ønske å finne hastighet og akselerasjon. Observasjonene vi gjør av målet vil inneholde en del støy og unøyaktigheter. Det er derfor blitt utviklet flere teknikker for å reduserer denne effekten. Felles for de fleste av disse er derimot at man trenger en modell som beskriver oppførselen til systemet man observerer. I målfølgingstilfellet må altså denne modellen beskrive bevegelsen til målet og hvordan sammenhengen mellom observasjon og virkelig tilstand er. En ulinær modell som beskriver denne bevegelsen har følgende form.

$$x(k) = f\left[x(k-1)\right] + w(k) \tag{2.7}$$
$$z(k) = h\left[x(k)\right] + v(k) \tag{2.8}$$

Her er $x(k)$ og $x(k-1)$ virkelig tilstand ved to forskjellige tidspunkter. I denne oppgaven er vi interresert i å kjenne posisjon og hastighet til målet. $x(k)$ vil derfor være en vektor bestående av følgende komponenter $x(k) = [x_1(k), y_2(k), z_3(k), v_{x_1}(k), v_{y_2}(k), v_{z_3}(k)]^T$. Videre har vi $z(k)$ som er observasjonen man gjør av målet ved tiden k. $h(\cdot)$ er en målefunksjon og beskriver koblingen mellom en observasjon og virkelig tilstand, mens $f(\cdot)$ er en transisjonsfunksjon og beskriver overgangen fra en tidsperiode til den neste. Både $h(\cdot)$ og $f(\cdot)$ er kjente funksjoner man enkelt kan finne. Hadde systemet vært fritt for støy hadde man derfor enkelt kunne finne virkelig tilstand basert på observasjon. Dessverre består systemt vårt også av støy. Disse representeres som målestøy $v(k)$ og prosesstøy $w(k)$. Målestøyen representerer usikkerheten i observasjonen man gjør og denne er ofte kjent. Dette fordi sensorene ofte har en karakteristikk som tilsier at man kjenner til hvordan målingene fordeler seg. Prosesstøyen er derimot vanskeligere. Denne representerer i hovedsak feil i modellen. Altså avvik mellom modellen og virkelig situasjon. Modellen baserer seg ofte på at man har mål som beveger seg i rette linjer. Dersom et mål avviker vesentlig fra denne bevegelsen, sier man at målet manøvrerer. Dette gir opphav til enda et problem kalt manøverdeteksjon, som vil bli gjennomgått i et senere kapittel.

For å kunne benytte modellen i et kalmanfilter er man nødt til å gjøre noen antagelser om støyen. Man antar at både målestøyen og prosesstøyen

er hvit støy. Dvs at støyen er normalfordelt med en (u)kjent varians, null i middelverdi, og ukorrelert med seg selv og hverandre. Dette utrykkes på følgende vis.

$$w(k) = N(0, Q(k)) \tag{2.9}$$
$$v(k) = N(0, R(k)) \tag{2.10}$$

$Q(k)$ og $R(k)$ er her to matriser hvor man da finner variansen til støyen på diagonalen.

$$R = \begin{bmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_{Az}^2 & 0 \\ 0 & 0 & \sigma_{El}^2 \end{bmatrix} \tag{2.11}$$

$$Q = \begin{bmatrix} \sigma_{x_1}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{y_2}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{z_3}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{v_{x_1}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{v_{y_2}}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{v_{z_3}}^2 \end{bmatrix} \tag{2.12}$$

2.3 Standard Kalmanfilter

Kalmanfilteret ble utviklet av Rudolf E. Kalman rundt 1960 [9]. Filteret er en tilstandsestimator som produserer et optimalt estimat i den forstand at gjennomsnittsverdien av summen av estimeringsavviket minimeres [18]. Det benytter en dynamisk modell av systemet kombinert med eventuelle observasjoner av systemet for å konstruere et estimat av systemets tilstand. Dette estimatet vil være mer nøyaktig enn om man kun hadde benyttet observasjoner alene [18]. For at Kalmanfilteret skal gi et optimalt estimat forutsetter det at modellen er lineær. Vår ulineære modell presentert tidligere kan derfor omskrives noe.

$$x(k) = \mathbf{F}(k)x(k-1) + w(k) \tag{2.13}$$
$$z(k) = \mathbf{G}(k)x(k) + v(k) \tag{2.14}$$

Forskjellen fra den ulineære modellen presentert tidligere, er at de to funksjonene $f(\cdot)$ og $h(\cdot)$ nå kan representeres som to matriser $F(k)$ og $G(k)$. I denne oppgaven kalles det lineære Kalmanfilteret for et standard Kalmanfilter (SKF). Dette for å lettere unngå forveksling med de andre typene Kalmanfilter som presenteres senere. Kalmanfilteret har også noen forventninger knyttet til støyen i systemet. Både prosesstøyen og observasjonsstøyen må være hvit. Vi har altså at $w(k)$ og $v(k)$ må være ukorrelert med hverandre og seg selv, og ha middelverdi lik null [18]. Dette er heldigvis en antagelse man kan gjøre i målfølgingssystem. Selve KF algoritmen blir ofte presentert som en del. Den kan imidlertid deles opp i to separate operasjoner som kan utføres hver for seg. Den første delen kalles prediksjon og benytter systemmodellen til å forutse fremtidig tilstand. Deretter følger en oppdatering hvor prediksjonen korrigeres ved hjelp av en observasjon. Ligningene for SKF blir her presentert hver for seg. Det er her viktig å merke seg at det er et diskret Kalmanfilter man benytter her.

$$\overline{x}(k) = \mathbf{F}(k)\hat{x}(k-1) \tag{2.15}$$

Første del av Kalmanfilteret forutser neste tilstand til estimatet. Dette gjøres ved hjelp av transisjonsmatrisa $\mathbf{F}(k)$. Transisjonsmatrisa beskriver overgangen fra et tidssteg til det neste, og er gitt av modellen.

$$\bar{P}(k) = \mathbf{F}(k)\hat{P}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k) \tag{2.16}$$

Tilsvarende som for estimatet må man også forutse kovariansmatrisa, $\mathbf{P}(k)$, til neste tidssteg. Kovariansmatrisa forteller oss noe om usikkerheten til estimatet vi har. Diagonalen i kovariansmatrisa vil inneholde variansen for hvert av estimatene i $\hat{x}(k)$. For eksempel vil det første elementet i kovariansmatrisa tilsvare variansen for posisjon i x-retning av estimatet. Som man kan se av ligningen, predikteres kovariansmatrisen fremover slik som for estimatet, men i tillegg legger man til prosesstøyen $\mathbf{Q}(k)$. Prosesstøyen tar høyde for eventuelle usikkerheter i modellen, og er en parameter som benyttes for å justere Kalmanfilteret.

$$\tilde{z}(k) = z(k) - \mathbf{G}(k)\overline{x}(k) \tag{2.17}$$

$\tilde{z}(k)$ representerer hvor mye feil det er mellom observasjon og estimat. Det er her viktig å merke seg at observasjonen $z(k)$ og estimatet $\overline{x}(k)$ må være på samme format. I vårt tilfelle vil det si at begge må være på kartesiske koordinater. Siden en radar returnerer observasjoner gitt i sfærsike koordinater, må altså disse konverteres til kartesiske koordinater før de kan benyttes i et SKF. Siden $z(k)$ kun vil inneholde posisjonskoordinater, benyttes målematrisa $\mathbf{G}(k)$ til å plukke ut kun posisjonskoordinater fra estimatet $\bar{x}(k)$.

$$\mathbf{S}(k) = \mathbf{G}(k)\overline{\mathbf{P}}(k)\mathbf{G}(k)^T + \mathbf{R}(k) \tag{2.18}$$

Tilsvarende som vi har at $\mathbf{P}(k)$ forteller oss noe om usikkerheten til estimatet, forteller $\mathbf{S}(k)$ oss noe om usikkerheten til observasjonene. $\mathbf{S}(k)$ forteller oss altså om vi skal stole mer på modellen eller mer på målingen. Som man kan se av ligningen, inkluderer dette også observasjonsstøyen $\mathbf{R}(k)$. $\mathbf{S}(k)$ kalles ofte for innovasjonskovariansen. Innovasjonskovariansen har en viktig rolle innenfor målfølging. Under dataassosiasjonen, der man finner koblinger mellom observasjoner og etablerte mål, benyttes innovasjonskovariansen til å avgjøre hvilke målinger som er aktuelle kandidater for assosiasjon.

$$\mathbf{K}(k) = \overline{\mathbf{P}}(k)\mathbf{G}(k)^T\mathbf{S}(k)^{-1} \tag{2.19}$$

Nå som man har verdier for både usikkerheten til estimatet og usikkerheten til observasjonene, kan man beregne kalmanforsterkningen $\mathbf{K}(k)$. Kalmanforsterkningen avgjør om man skal vektlegge estimatet eller målingen mer når man senere kombinerer disse to. En høyere kalmanforsterkning gjør at man inkluderer mer av målingen enn av estimatet.

$$\hat{x}(k) = \bar{x}(k) + \mathbf{K}(k)\tilde{z}(k) \tag{2.20}$$
$$\hat{\mathbf{P}}(k) = [\mathbf{I} - \mathbf{K}(k)\mathbf{G}(k)]\,\bar{\mathbf{P}}(k) \tag{2.21}$$

Siste trinn i Kalmanfilteret er å korrigere vårt tidligere estimat ved hjelp av målingen. Her benyttes Kalmanforsterkningen til å bestemme hvor mye man vektlegger målingen i forhold til estimatet. Resultatet er at man ender opp med et nytt estimat der man forhåpentligvis har redusert effekten av unøyaktige målinger og modellfeil.

Algoritme 1 Kalmanfilter

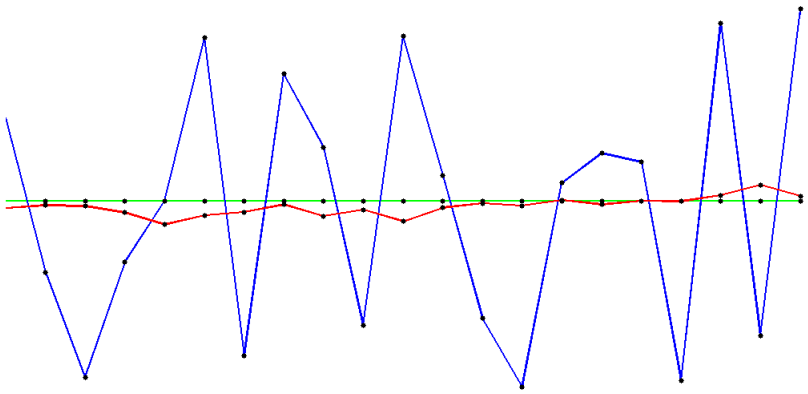
Prediksjon:

$$\bar{x}(k) = \mathbf{F}(k)\hat{x}(k-1)$$
$$\bar{\mathbf{P}}(k) = \mathbf{F}(k)\hat{\mathbf{P}}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)$$

Oppdatering:

$$\tilde{z}(k) = z(k) - \mathbf{G}(k)\bar{x}(k)$$
$$\mathbf{S}(k) = \mathbf{G}(k)\bar{\mathbf{P}}(k)\mathbf{G}(k)^T + \mathbf{R}(k)$$
$$\mathbf{K}(k) = \bar{\mathbf{P}}(k)\mathbf{G}(k)^T\mathbf{S}(k)^{-1}$$
$$\hat{x}(k) = \bar{x}(k) + \mathbf{K}(k)\tilde{z}(k)$$
$$\hat{\mathbf{P}}(k) = [\mathbf{I} - \mathbf{K}(k)\mathbf{G}(k)]\,\bar{\mathbf{P}}(k)$$

Eksempel 1: Kalmanfilter



Figur 2.2: Eksempel på filtrering ved hjelp av Kalmanfilter. Grønn er her virkelig posisjon til målet. Blå viser observasjonene fra radaren. Rød viser den estimerte banen gitt av kalmanfilteret.

I figur 2.2 har man en enkel illustrasjon på bruk av et Kalmanfilter. Den grønne kurven viser et mål som beveger seg med konstant hastighet. Modellen vi benytter i Kalmanfilteret beskriver denne bevegelsen så nøyaktig som mulig. Fra radaren får vi observasjonsdata (blå) om posisjonen til målet. Disse inneholder en del støy, og vi får dermed et dårlig estimat over posisjonen til målet. Ved å benytte et Kalmanfilter til å finne estimatet av den virkelige posisjonen til målet får vi den røde kurven. Dette viser også hvorfor man kaller det for et filter. Kalmanfilteret fungerer som et slags adaptivt lavpassfilter av observasjonene.

SKF kan i utgangspunktet kun benyttes på lineære modeller. Problemet med målfølging er at modellen man benytter ikke er linær. Dette skyldes av at observasjonene fra radaren er i sfæriske koordinater. Konverteringen fra sfæriske til kartesiske koordinater er ikke en lineær operasjon. Et alternativ er å konvertere observasjonene til kartesiske koordinater. Man kan da benytte et SKF. Dette medfører derimot en del problemer med korrelasjon mellom observasjonsparameterene. I kapittel 2.3.2 gjennomgås nærmere hvorfor, samt en løsning på problemet.

I kapittel 2.4.1 presenteres nærmere de forskjellige verdiene for de forskjellige matrisene benyttet i Kalmanfilteret. Den eneste matrisa som er forskjellig, er observasjonsmatrisa, som for et SKF vil se ut som følger.

$$\mathbf{G}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.22}$$

2.3.1 Kalmanfilter Triks

Ved implementasjon av Kalmanfilteret kan man ofte møte et problem med numerisk stabilitet. Dette kan skyldes ørsmå avrundingsfeil i datamaskinen. Disse kan fort summeres opp i Kalmanfiltert og føre til at kovariansmatrisene ikke blir symmetriske. Det har derfor blitt benyttet noen små triks for å redusere effekten av avrundingsfeil. Den første av disse er å benytte Josephs form for korreksjon av kovariansmatrisen. Denne formen er litt tyngere beregningsmesssig, men er mindre sensitiv for avrundingsfeil [2].

$$\hat{\mathbf{P}}(k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{G}(k))\overline{\mathbf{P}}(k)(\mathbf{I} - \mathbf{K}(k)\mathbf{G}(k)^T + \mathbf{K}(k)\mathbf{R}(k)\mathbf{K}(k)^T \tag{2.23}$$

Et annet triks som også ble benyttet er å tvinge kovariansmatrisene til alltid å være symmetriske. Dette gjøres kun til slutt, etter at Kalmanfilteret er ferdig med beregningene.

$$\hat{\mathbf{P}}(k) = 0.5(\hat{\mathbf{P}}(k) + \hat{\mathbf{P}}(k)^T) \tag{2.24}$$

Bruken av Josephs form skal derimot gjøre at denne beregningen ikke er nødvendig. Dersom man alikevel skulle oppleve ustabilitet i kovariansen, kan man prøve dette trikset før man undersøker andre forhold ved filterdesignet.

2.3.2 Forventningsrett Konvertering

Et av problemene med bruk av et SKF, er at radarobservasjonen er i sfæriske koordinater. Konverteringen fra sfæreiske til kartesiske koordinater er en ikke-lineær operasjon, og en konvertering må derfor gjøres før man benytter observasjonene i kalmanfilteret. Denne konverteringen er ikke helt problemfri da den også fører til små unøyaktigheter estimatet. Dette skyldes at parameterene i sfæriske koordinater ikke er uavhengige av hverandre. For eksempel vil en reduksjon i elevasjon også føre til en reduksjon i avstand. Når man konverterer til kartesiske koordinater, mister man denne koblingen. Dette medfører at SKF gir et dårligere estimat. Det har derfor blitt utviklet metoder som dekobler konverteringen, slik at de konverterte verdiene er forventningsrett. Man får da en markant forbedring av estimatet, enn om man bare hadde benyttet en ren konvertering [3]. Ligningene gjengitt her er hentet fra [35].

$$x_1 = R\cos(El)\cos(Az) - x_c \tag{2.25}$$
$$y_2 = R\cos(El)\sin(Az) - y_c \tag{2.26}$$
$$z_3 = -R\sin(El) - z_c \tag{2.27}$$
$$\tag{2.28}$$
$$x_c = R\cos(El)\cos(Az)[\exp(-\sigma_{Az}^2 - \sigma_{El}^2) - \exp(-\sigma_{Az}^2/2 - \sigma_{El}^2/2)] \tag{2.29}$$
$$y_c = R\cos(El)\sin(Az)[\exp(-\sigma_{Az}^2 - \sigma_{El}^2) - \exp(-\sigma_{Az}^2/2 - \sigma_{El}^2/2)] \tag{2.30}$$
$$z_c = -R\sin(El)[\exp(-\sigma_{El}^2) - \exp(-\sigma_{El}^2/2)] \tag{2.31}$$
$$\tag{2.32}$$
$$R = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{xy} & R_{yy} & R_{yz} \\ R_{xy} & R_{yz} & R_{zz} \end{bmatrix} \tag{2.33}$$
$$\tag{2.34}$$
$$R_{xx} = [R^2(\beta_x\beta_{xy} - \alpha_x\alpha_{xy}) + \sigma_R^2(2\beta_x\beta_{xy} - \alpha_x\alpha_{xy})]\exp(-2\sigma_{Az}^2 - 2\sigma_{El}^2) \tag{2.35}$$
$$R_{xy} = [R^2(\beta_{xy} - \alpha_{xy}\exp\sigma_{Az}^2) + \tag{2.36}$$
$$\sigma_R^2(2\beta_{xy} - \alpha_{xy}\exp\sigma_{Az}^2)]\sin(Az)\cos(Az)\exp(-4\sigma_{Az}^2 - 2\sigma_{El}^2) \tag{2.37}$$
$$R_{xz} = [R^2(1 - \exp\sigma_{El}^2) + \tag{2.38}$$
$$\sigma_R^2(2 - \exp\sigma_{El}^2)]\cos(Az)\sin(El)\cos(El)\exp(-\sigma_{Az}^2 - 4\sigma_{El}^2) \tag{2.39}$$
$$R_{yy} = [R^2(\beta_y\beta_{xy} - \alpha_y\alpha_{xy}) + \sigma_R^2(2\beta_y\beta_{xy} - \alpha_y\alpha_{xy})]\exp(-2\sigma_\theta^2 - 2\sigma_\varphi^2) \tag{2.40}$$
$$R_{yz} = [R^2(1 - \exp\sigma_{El}^2) + \tag{2.41}$$
$$\sigma_R^2(2 - \exp\sigma_{El}^2)]\sin(Az)\sin(El)\cos(El)\exp(-\sigma_{Az}^2 - 4\sigma_{El}^2) \tag{2.42}$$
$$R_{zz} = [R^2(\beta_z - \alpha_z) + \sigma_R^2(2\beta_z - \alpha_z)]\exp(-2\sigma_{El}^2) \tag{2.43}$$
$$\tag{2.44}$$
$$\alpha_x = \sin^2 Az\sinh\sigma_{Az}^2 + \cos^2 Az\cosh\sigma_{Az}^2 \tag{2.45}$$
$$\alpha_y = \sin^2 Az\cosh\sigma_{Az}^2 + \cos^2 Az\sinh\sigma_{Az}^2 \tag{2.46}$$
$$\alpha_z = \sin^2 El\cosh\sigma_{El}^2 + \cos^2 El\sinh\sigma_{El}^2 \tag{2.47}$$
$$\alpha_{xy} = \sin^2 El\sinh\sigma_{El}^2 + \cos^2 El\cosh\sigma_{El}^2 \tag{2.48}$$
$$\beta_x = \sin^2 Az\sinh(2\sigma_{Az}^2) + \cos^2 Az\cosh(2\sigma_{Az}^2) \tag{2.49}$$
$$\beta_y = \sin^2 Az\cosh(2\sigma_{Az}^2) + \cos^2 Az\sinh(2\sigma_{Az}^2) \tag{2.50}$$
$$\beta_z = \sin^2 El\cosh(2\sigma_{El}^2) + \cos^2 El\sinh(2\sigma_{El}^2) \tag{2.51}$$
$$\beta_{xy} = \sin^2 El\sinh(2\sigma_{El}^2) + \cos^2 El\cosh(2\sigma_{El}^2) \tag{2.52}$$

2.4 Utvidet Kalmanfilter

Noe av problemet med SKF, er at det forutsetter at man har en lineær modell. I målfølging har man ofte observasjonene i sfæriske koordinater, mens estimatet er i kartesiske koordinater. Konverteringen fra sfæriske til kartesiske koordinater er ikke lineær, og SKF kan derfor ikke benyttes direkte. En løsning ble presentert i kapittel 2.3.2. Denne metoden fungerer derimot kun for det spesifiserte tilfellet hvor man har observasjoner i sfæriske koordinater og estimat i kartesiske koordinater. Man ønsker isteden å ha et filter der man kan benytte observasjonene direkte, uten å måte foreta en forventningsrett konvertering. En av løsningene på dette problemet er det utvidede Kalmanfilteret (EKF).

I EKF utfører man en fortløpende linearisering [18]. Man kan ha både transisjon og observasjonsfunksjonen ulineær. For målfølging der estimatet er angitt i kartesiske koordinater og observasjonene i sfæriske koordinater er kun observasjonsfunksjonen ulineær. Transisjonsfunksjonen vil fortsatt være lineær og kan derfor representeres som en matrise slik som for SKF. Modellen for et EKF benyttet til målfølging blir da som følger.

$$x(k) = \mathbf{F}(k)x(k-1) + w(k) \tag{2.53}$$
$$z(k) = h(x(k)) + v(k) \tag{2.54}$$

Ligningenen for EKF er nesten like som for SKF. Utregningen av differansen mellom estimat og måling blir nå:

$$\tilde{z}(k) = z(k) - h\left[\bar{x}(k)\right] \tag{2.55}$$

Her konverterer man estimatet til sfæriske koordinater, slik at man kan finne differansen mellom observasjon og estimat i sfæriske koordinater. $h(\cdot)$ er her en funksjon som konverterer fra kartesiske koordinater til sfæriske koordinater. Man er også nødt til å finne en ny matrise $\mathbf{H}(k)$. $\mathbf{H}(k)$ representerer en linearisering av systemet rundt det gjeldende estimatet. Selve lineariseringen beregnes ved å finne Jacobimatrisa av det gjeldende estimatet.

$$\mathbf{H}(k) = \left.\frac{\partial h}{\partial x}\right|_{\bar{x}(k)} \tag{2.56}$$

Selve utregningen av $h(k)$ og $\mathbf{H}(k)$ blir presentert i kapittel 2.4.1.

Algoritme 2 Utvidet Kalmanfilter

Prediksjon:

$$\begin{aligned}\bar{x}(k) &= \mathbf{F}(k)\hat{x}(k-1)\\ \bar{P}(k) &= \mathbf{F}(k)\hat{P}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)\end{aligned}$$

Oppdatering:

$$\begin{aligned}\tilde{z}(k) &= z(k) - h(\bar{x}(k))\\ \mathbf{S}(k) &= \mathbf{H}(k)\bar{\mathbf{P}}(k)\mathbf{H}(k)^T + \mathbf{R}(k)\\ \mathbf{K}(k) &= \bar{\mathbf{P}}(k)\mathbf{H}(k)^T\mathbf{S}(k)^{-1}\\ \hat{x}(k) &= \bar{x}(k) + \mathbf{K}(k)\tilde{z}(k)\\ \hat{\mathbf{P}}(k) &= (\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))\bar{\mathbf{P}}(k)\end{aligned}$$

2.4.1 Noen Utvalgte Parametere

For å kunne benytte EKF, er man først nødt til å finne uttrykk et for de forskjellige matrisene. For denne oppgaven ønsker vi å finne et estimat av posisjon og hastighet for hvert mål i henholdsvis x,y og z akse. Man ønsker med andre ord å finne et estimat for 6 forskjellige tilstander. Estimatvektoren vil da ha følgende form.

$$\hat{x}(k) = \begin{bmatrix} x_1(k) & y_2(k) & z_3(k) & v_{x_1}(k) & v_{y_2}(k) & v_{z_3}(k)\end{bmatrix}^T \tag{2.57}$$

Når det kommer til bevegelsen til et mål, så antar man at dette beveger seg i en rettlinjet bevegelse, og med konstant hastighet.

$$x(t) = x(t-1) + v(t-1)\Delta t \tag{2.58}$$

Skrevet på matriseform, for hver av de tre tilstandene x_1, y_2 og z_3, får man følgende transisjonsmatrise.

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.59}$$

De to neste matrisene er for henholdsvis målestøy og prosesstøy. Målestøyen er som oftest gitt av sensoren. Denne er derfor lik for alle mål som kommer fra den samme sensoren. Noen sensorer vil inkludere informasjon om størrelsene på verdiene i målestøymatrisa som en del av målingene. R vil da variere med tiden og målingen, men det er antatt at R er konstant i denne oppgaven.

$$R = \begin{bmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_{Az}^2 & 0 \\ 0 & 0 & \sigma_{El}^2 \end{bmatrix} \tag{2.60}$$

Prosesstøymatrisa representerer alle andre feil i systemet som ikke dekkes av målestøymatrisa. Siden målestøymatrisa ofte er kjent grunnet valg av sensor, benyttes som oftest prosesstøymatrisa for innstilling av Kalmanfilteret. Verdiene i prosesstøymatrisa vil være konstante i perioder. Under kapittel 2.9.6, som omhandler manøverdeteksjon, vil det derimot utvikles metoder der man justerer prosesstøyen for å få et bedre estimat.

$$Q = \begin{bmatrix} \sigma_{x_1}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{y_2}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{z_3}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{v_{x_1}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{v_{y_2}}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{v_{z_3}}^2 \end{bmatrix} \tag{2.61}$$

EKF må kunne konvertere estimatet fra kartesiske koordinater til sfæriske koordinater slik at man kan finne differansen mellom observasjon og estimat. Dette gjøres ved å benytte ligningene for konvertering mellom kartesiske og sfæriske koordinater.

$$h\left[\bar{x}(k)\right] = \begin{bmatrix} h_r\left[\bar{x}(k)\right] \\ h_a\left[\bar{x}(k)\right] \\ h_e\left[\bar{x}(k)\right] \end{bmatrix} \tag{2.62}$$

$$h_r\left[\bar{x}(k)\right] = \sqrt{x_1(k)^2 + y_2(k)^2 + z_3(k)^2} \tag{2.63}$$

$$h_a\left[\bar{x}(k)\right] = atan\left(\frac{y_1(k)}{x_2(k)}\right) \tag{2.64}$$

$$h_e\left[\bar{x}(k)\right] = atan\left(\frac{-z_3(k)}{\sqrt{x_1(k)^2 + y_2(k)^2}}\right) \tag{2.65}$$

Den siste matrisa vi må finne er Jacobimatrisa $\mathbf{H}(k)$. Den beregnes ved å finne de partiellderiverte av $h(k)$ med hensyn på alle tilstandene i estimatet.

$$H(k) = \begin{bmatrix} \frac{\partial h_r(\bar{x}(k))}{\partial \bar{x}(k)} \\ \frac{\partial h_a(\bar{x}(k))}{\partial \bar{x}(k)} \\ \frac{\partial h_e(\bar{x}(k))}{\partial \bar{x}(k)} \end{bmatrix} = \begin{bmatrix} H_r(k) \\ H_a(k) \\ H_e(k) \end{bmatrix} \tag{2.66}$$

$$H_r(k) = \begin{bmatrix} \frac{x_1(k)}{r(k)} & \frac{y_2(k)}{r(k)} & \frac{z_3(k)}{r(k)} & 0 & 0 & 0 \end{bmatrix} \tag{2.67}$$

$$H_b(k) = \begin{bmatrix} -\frac{y_2(k)}{rh(k)^2} & \frac{x_1(k)}{rh(k)^2} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.68}$$

$$H_e(k) = \begin{bmatrix} \frac{x_1(k)z_3(k)}{r(k)^2 rh(k)} & \frac{y_2(k)z_3(k)}{r(k)^2 rh(k)} & -\frac{rh(k)}{r(k)^2} & 0 & 0 & 0 \end{bmatrix} \tag{2.69}$$

$$r(k) = \sqrt{x_1(k)^2 + y_2(k)^2 + z_3(k)^2} \tag{2.70}$$

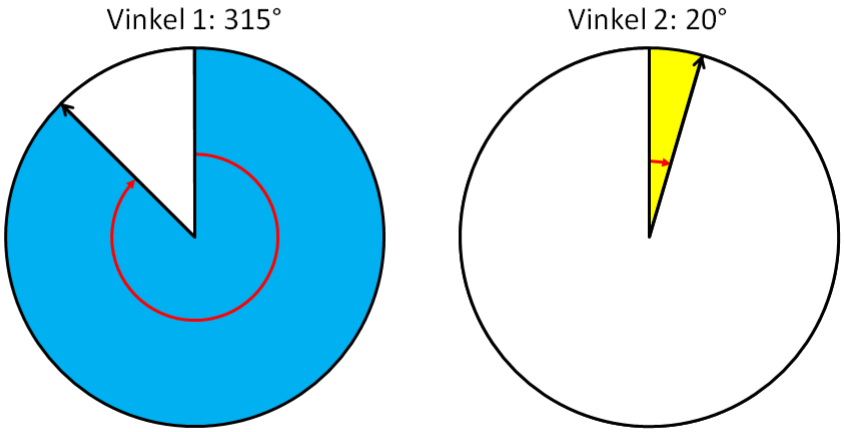
$$rh(k) = \sqrt{x_1(k)^2 + y_2(k)^2} \tag{2.71}$$

2.4.2 Vinkelberegning

Et problem som dukket opp underveis under testing av EKF, viste seg å være knyttet til beregning av vinkler. Dette er egentlig et trivielt problem, men nevnes her slik at man er klar over det. I EKF beregner man differansen mellom to sfæriske koordinater som følger.

$$\tilde{z}(k) = z(k) - h\left[\bar{x}(k)\right] \tag{2.72}$$

Det å finne differansen mellom avstand og elevasjon fungerte som forventet. Det viste seg derimot at man ikke kan finne differansen i asimut ved ren subtraksjon. Problemet illustreres best ved å finne differansen mellom to vinkler gitt i grader. Metoden kan derimot like enkelt benyttes på vinkler angitt i radianer.



Figur 2.3: Beregning av vinkeldifferanse

Anta at man ønsker å finne differansen mellom to vinkler. Den første vinkelen er på 315° og den andre er på 20°. Differansen kan nå enkelt finnes.

$$\begin{aligned} \texttt{Vinkeldifferanse} &= \texttt{Vinkel1} - \texttt{Vinkel2} && (2.73)\\ &= 315 - 20 && (2.74)\\ &= 295 && (2.75) \end{aligned}$$

295° var definitivt ikke det man forventet! Riktig svar her skulle vært −65°. Siden vi benytter negative verdier i differansen, ønsker vi at vinkelen skal befinne seg mellom −180° og 180°. Følgende pseudokode løser problemet med vinkeldifferanse ved å alltid sørge for at differansen er innenfor riktig intervall. Pseudokoden gitt her er beregnet brukt på radianer.

Algoritme 3 Vinkeldifferanse

$diff \Leftarrow angle1 - angle2$
while $diff < -pi$ **do**
  $diff \Leftarrow diff + 2 * pi$
end while
while $diff > pi$ **do**
  $diff \Leftarrow diff - 2 * pi$
end while

2.5 Unscented Kalmanfilter

I 1997 skrev Simon J. Julier og Jeffrey K. Uhlmann den første artikkelen om "Unscented Kalman Filter" (UKF) [20]. For ulinære systemer er det i dag vanlig å benytte EKF. Problemet med EKF er at det må utføres en linearisering av systemet. EKF antar deretter at systemet er lineært rundt arbeidspunktet man lineariserer rundt. Julier og Uhlman påpeker at dette gir to store problemer ved bruk av EKF:

1. Dersom systemet ikke er tilnærmet lineært rundt det lineariserte aribeidspunktet risikerer man at filteret blir ustabilt.
2. Beregningen av Jacobimatrisa kan være svært komplisert, både å finne matematisk og å implementere i praksis.

Disse to problemene er derfor utgangspunktet for utviklingen av UKF. UKF løser dette problemet ved å beregne sigmapunkter rundt estimatet. Disse punktene propageres deretter gjennom ulineæriteten, og vektes forskjellig før de benyttes til å rekonstruere estimatet. I ettertid har det blitt utviklet utallige varianter av UKF algoritmen. S. Kolas et al gir i [22] en oversikt over flere ulike implementasjoner. Forskjellen mellom de er hvordan man velger sigmapunkter, hvordan man vekter de forskjellige sigmapunktene og hvordan man behandler prosess- og observasjonsstøy.

Algoritmen som presenteres her er den samme som benyttes i [37], og beregner totalt $2n + 1$ sigmapunkter, der n er antall tilstandsparametere i estimatet. Dvs at man for denne oppgaven beregner totalt 13 sigmapunkter. Det antas også at man har additiv hvit støy. Ved beregning av vektene, benyttes α, β og κ til å bestemme spredningen av sigmapunktene. κ forteller hvor langt unna estimatet sigmapunktene skal velges. En høyere verdi flytter sigmapunktene lenger vekk fra estimatet. κ kan velges negativ, men det anbefales ikke, da man risikerer å ende opp med en kovariansmatrise hvor man ikke kan beregne kvadratroten. α indikerer hvor spredt sigmapunktene skal velges sett i forhold til kovariansen til estimatet. Tilslutt har vi β som representerer hvor gaussisk systemet er. Denne settes til 2 for et rent gaussisk system. Ifølge [37] er det noe usikkert hvordan man finner optimale parametere. I simuleringer ser det derimot ikke ut til at valg av parameter påvirker estimatet nevneverdig. Det blir derfor anbefalt følgende verdier $[\alpha, \beta, \kappa] = [1, 2, 0]$. Dersom filteret ikke produserer tilfredstillende resultater

kan man prøve å gjøre justeringer på parameterene. Som man kan se, så er vektene uavhengig av tiden, og kan derfor beregnes på forhånd.

$$\Gamma = \sqrt{n+\gamma} \tag{2.76}$$
$$\gamma = \alpha^2(n+\kappa) - n \tag{2.77}$$
$$W_0^x = \frac{\gamma}{n+\gamma} \tag{2.78}$$
$$W_0^c = \frac{\gamma}{n+\gamma} + (1-\alpha^2+\beta) \tag{2.79}$$
$$W_i^x = W_i^c = \frac{1}{2(n+\gamma)} \tag{2.80}$$

Nå som man har vektene ønsker, man å prediktere estimatet til gjeldende tid. Siden vi i målfølging har en lineær prediksjon, kan man benytte samme prediksjon som for SKF. Ligningene i UKF har derfor blitt modifisert for å benytte lineær prediksjon. Prediksjonen blir dermed.

$$\bar{x}(k) = \mathbf{F}(k)\hat{x}(k-1) \tag{2.81}$$
$$\bar{\mathbf{P}}(k) = \mathbf{F}(k)\hat{\mathbf{P}}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k) \tag{2.82}$$
$$\tag{2.83}$$

Først når man kommer til korreksjonsdelen behøver man å beregne sigmapunktene.

$$\chi_i(k) = \begin{bmatrix} \hat{x}(k) & \hat{x}(k) + \Gamma\sqrt{\bar{\mathbf{P}}(k)} & \hat{x}(k) - \Gamma\sqrt{\bar{\mathbf{P}}(k)} \end{bmatrix} \tag{2.84}$$

For å beregne $\sqrt{P}$ må man benytte en metode slik som Cholesky dekomposisjon. Svaret multipliseres deretter med skaleringsfaktoren Γ. En ting som er veldig viktig å merke seg er at, adderingen/subtraksjonene er for hver enkelt kolonne i kovariansmatrisa. Man skal altså addere eller subtraktere estimatet fra hver kolonne i kovariansmatrisa for seg selv. For denne oppgaven ender man altså opp med totalt 13 sigmapunkter. Vi må nå rekonstruere

kovariansmatrisa utifra sigmapunktene. Disse propageres deretter gjennom den ulineære målefunksjonen.

$$\bar{\mathbf{P}}(x) = \sum_{i=1}^{2n} W_i^c \left[\chi_i(k) - \bar{x}(k)\right] \left[\chi_i(k) - \bar{x}(k)\right]^T \tag{2.85}$$

$$\gamma_i(k) = h\left[\chi_i(k)\right] \tag{2.86}$$

$$\tag{2.87}$$

Man kan så beregne prediktert observasjon.

$$\hat{z}(k) = \sum_{i=1}^{2n} W_i^x \gamma_i(k) \tag{2.88}$$

Deretter beregnes kalmanforsterkningen. Dette gjøres ved å først beregne en observasjonskovariansmatrise, og en krysskovariansmatrise mellom estimatkovarians og observasjonskovarians.

$$\mathbf{P}_{zz}(k) = \sum_{i=1}^{2n} W_i^c (\gamma_i(k) - \hat{z}(k))(\gamma_i(k) - \hat{z}(k))^T + \mathbf{R}(k) \tag{2.89}$$

$$\mathbf{P}_{xz}(k) = \sum_{i=1}^{2n} W_i^c (\chi_i(k) - \overline{x}(k))(\gamma_i(k) - \hat{z}(k))^T \tag{2.90}$$

$$\mathbf{K}(k) = \mathbf{P}_{xz}(k)\mathbf{P}_{zz}^{-1}(k) \tag{2.91}$$

$$\tag{2.92}$$

Og til slutt oppdateres estimatet med den nye observasjonen.

$$\hat{x}(k) = \bar{x}(k) + \mathbf{K}(k)\left[z(k) - \hat{z}(k)\right] \tag{2.93}$$

$$\hat{\mathbf{P}}(k) = \bar{\mathbf{P}}(k) - \mathbf{K}(k)\mathbf{P}_{yy}(k)\mathbf{K}^T(k) \tag{2.94}$$

UKF er noe tyngere beregningsmessig enn EKF. Dette fordi man er nødt til å propagere 13 sigmapunkter gjennom ulineariteten istedenfor et enkelt estimat. Fordelene med bruk av UKF er derimot at man slipper å linearisere systemet. Man har ikke lenger behov for å finne Jacobimatrisa, slik som for EKF, og kan isteden benytte den ulineære transformen direkte.

Vekter:

$$\begin{aligned}
\Gamma &= \sqrt{n+\gamma} \\
\gamma &= \alpha^2(n+\kappa) - n \\
W_0^x &= \frac{\gamma}{n+\gamma} \\
W_0^c &= \frac{\gamma}{n+\gamma} + (1-\alpha^2+\beta) \\
W_i^x &= W_i^c = \frac{1}{2(n+\gamma)}
\end{aligned}$$

Prediksjon:

$$\begin{aligned}
\hat{x}(k) &= \mathbf{F}(k)\hat{x}(k-1) \\
\bar{\mathbf{P}}(k) &= \mathbf{F}(k)\hat{\mathbf{P}}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)
\end{aligned}$$

Oppdatering:

$$\begin{aligned}
\chi_i(k) &= \left[\hat{x}(k) \quad \hat{x}(k) + \Gamma\sqrt{\hat{\mathbf{P}}(k)} \quad \hat{x}(k) - \Gamma\sqrt{\hat{\mathbf{P}}(k)}\right] \\
\bar{\mathbf{P}}(x) &= \sum_{i=1}^{2n} W_i^c(\chi_i(k) - \bar{x}(k))(\chi_i(k) - \bar{x}(k))^T \\
\gamma_i(k) &= h(\chi_i(k)) \\
\hat{z}(k) &= \sum_{i=1}^{2n} W_i^x \gamma_i(k) \\
\mathbf{P}_{zz}(k) &= \sum_{i=1}^{2n} W_i^c(\gamma_i(k) - \hat{z}(k))(\gamma_i(k) - \hat{z}(k))^T + \mathbf{R}(k) \\
\mathbf{P}_{xz}(k) &= \sum_{i=1}^{2n} W_i^c(\chi_i(k) - \overline{x}(k))(\gamma_i(k) - \hat{z}(k))^T \\
\mathbf{K}(k) &= \mathbf{P}_{xz}(k)\mathbf{P}_{zz}(k)^{-1} \\
\hat{x}(k) &= \overline{x}(k) + \mathbf{K}(k)(z(k) - \hat{z}(k)) \\
\hat{\mathbf{P}}(k) &= \overline{\mathbf{P}}(k) - \mathbf{K}(k)\mathbf{P}_{zz}(k)\mathbf{K}^T(k)
\end{aligned}$$

2.6 Standard Informasjonsfilter

Dersom man ønsker å kombinere flere observasjoner til et felles estimat kan det være hensiktsmessig å benytte informasjonsformen til kalmanfilteret. I et informasjonsfilter (IF) summerer man sammen informasjon, noe som gjør at filteret beregningsmessig er enklere. For et informasjonsfilter benytter man en informasjonsvektor og en informasjonsmatrise istedenfor tilstandsvektor og kovariansmatrise.

$$y(k) = \mathbf{P}^{-1}(k)x(k) \tag{2.95}$$
$$\mathbf{Y}(k) = \mathbf{P}^{-1}(k) \tag{2.96}$$

Her er $y(k)$ informasjonsvektoren og $\mathbf{Y}(k)$ informasjonsmatrisa. I motsetning til Kalmanfilteret, har informasjonsfilteret en komplisert prediksjonsdel.

$$\mathbf{M}(k) = \mathbf{F}(k)^{-T}\mathbf{Y}(k-1)\mathbf{F}(k)^{-1} \tag{2.97}$$
$$\mathbf{C}(k) = \mathbf{M}(k)\left[\mathbf{M}(k) + \mathbf{Q}(k)^{-1}\right]^{-1} \tag{2.98}$$
$$\mathbf{L}(k) = \mathbf{I} - \mathbf{C}(k) \tag{2.99}$$
$$\mathbf{Y}(k) = \mathbf{L}(k)\mathbf{M}(k)\mathbf{L}(k)^T + \mathbf{C}(k)\mathbf{Q}(k)^{-1}\mathbf{C}(k)^T \tag{2.100}$$
$$y(k) = \mathbf{L}(k)\mathbf{F}(k)^{-T}y(k-1) \tag{2.101}$$

Siden prediksjondelen er såpass komplisert, er det ofte like enkelt å benytte Kalmanfilteret til prediksjon. Det er først i korreksjonsdelen at informasjonsfilteret er beregningsmessig enklere. For en enkelt observasjon er oppdateringen som følger.

$$\mathbf{Y}(k) = \mathbf{Y}(k) + \mathbf{G}^T(k)\mathbf{R}(k)^{-1}\mathbf{G}(k) \tag{2.102}$$
$$y(k) = y(k) + \mathbf{G}(k)^T\mathbf{R}(k)^{-1}z(k) \tag{2.103}$$

Som man kan se av ligningene, er selve oppdateringen i det siste leddet. Dersom man har flere observasjoner, kan disse enkelt kombineres ved å summere opp informasjonen som hver av observasjonene bidrar med.

$$\mathbf{I}_i(k) = \mathbf{G}_i(k)^T \mathbf{R}_i(k)^{-1} \mathbf{G}_i(k) \tag{2.104}$$

$$i_i(k) = \mathbf{G}_i(k)^T \mathbf{R}_i(k)^{-1} z_i(k) \tag{2.105}$$

$$\mathbf{Y}(k) = \mathbf{Y}(k) + \sum_{i=1}^{S} \mathbf{I}_i(k) \tag{2.106}$$

$$y(k) = y(k) + \sum_{i=1}^{S} i_i(k) \tag{2.107}$$

Selve informasjonsfilteret er algebraisk likt som Kalmanfilteret. Et Kalmanfilter og et informasjonsfilter som estimerer det samme systemet, skal derfor i teorien produsere eksakt like estimat. Fordelene ved bruk av informasjonsfilteret, er at korreksjonsdelen er mye enklere beregningsmessig. Matrisene man må invertere i et informasjonfilter har samme dimensjon som estimatet, men for et Kalmanfilter har de samme dimensjon som observasjonene. I praktisk implementasjon er også informasjonsfilteret oftere mer numerisk stabilt enn Kalmanfilteret [23]. Ulempen er at informasjonsvektoren og informasjonsmatrisa ikke kan benyttes direkte i andre algoritmer. De må først konverters tilbake til estimat og kovarians slik den er definert for Kalmanfilteret.

Siden prediksjonsdelen av informasjonsfilteret er ganske komplisert, benyttes i hovedsak kun korreksjonsdelen ved implementasjon i målfølgingssystemer. For en fullstendig estimator benyttes derfor kalmanfilteret til prediksjon, og informasjonsfilteret til kombinering av observasjonsdata.

Algoritme 4 Informasjonsfilter

Prediksjon:

$$\bar{x}(k) = \mathbf{F}(k)\hat{x}(k-1)$$
$$\bar{\mathbf{P}}(k) = \mathbf{F}(k)\hat{P}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)$$

Oppdatering:

$$\begin{aligned}
\mathbf{Y}(k) &= \mathbf{P}(k)^{-1} \\
\hat{y}(k) &= \mathbf{Y}(k)\hat{x}(k) \\
\mathbf{I}(k) &= \mathbf{G}(k)^T\mathbf{R}(k)^{-1}\mathbf{G}(k) \\
i &= \mathbf{G}(k)^T\mathbf{R}^{-1}(k)z(k) \\
\mathbf{Y}(k) &= \mathbf{Y}(k) + \sum \mathbf{I}(k) \\
\hat{y}(k) &= \hat{y}(k) + \sum i(k) \\
\mathbf{P}(k) &= \mathbf{Y}(k)^{-1} \\
\hat{x}(k) &= \mathbf{P}(k)\hat{y}(k)
\end{aligned}$$

2.7 Utvidet Informasjonsfilter

Tilsvarende som for SKF kan ikke SIF benyttes direkte med radarobservasjonene. Man kunne også her ha benyttet en forventningsrett konvertering av observasjonene og så benyttet SIF direkte. Isteden kan man benytte et utvidet informasjonsfilter (Extended Information Filter, EIF). Slik SIF er motparten til SKF, så er EIF motparten til EKF. Dessverre blir EIF noe mer komplisert når man tar hensyn til ulineariteten.

Siden vi har en lineær transisjonsmatrise, så kan vi benytte prediksjonsdelen fra SKF. Denne blir derfor ikke presentert her igjen. Når det kommer til korreksjonsdelen, er det kun en liten endring man må utføre. Man kan ikke lenger benytte observasjonene direkte under oppdateringen av estimatet. Isteden må følgende ligning benyttes.

$$\tilde{z}(k) = z(k) - h\left[\hat{x}(k)\right] + \mathbf{H}(k)\hat{x}(k) \tag{2.108}$$

Som man kan se, så er vi her avhengig av å ha tilstandsestimatet $\hat{x}(k)$ for å kunne benytte EIF. Funksjonen $h(\cdot)$ er her konvertering fra kartesiske til sfæriske koordinater, mens $\mathbf{H}(k)$ er den lineariserte observasjonsmatrisa. Begge disse er altså akkurat like som for EKF.

Algoritme 5 Utvidet Informasjonsfilter

Prediksjon:

$$\bar{x}(k) = \mathbf{F}(k)\hat{x}(k-1)$$
$$\bar{\mathbf{P}}(k) = \mathbf{F}(k)\hat{\mathbf{P}}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)$$

Oppdatering:

$$\mathbf{Y}(k) = \mathbf{P}(k)^{-1}$$
$$\hat{y}(k) = \mathbf{P}(k)^{-1}\hat{x}(k)$$
$$\mathbf{I}(k) = \mathbf{H}(k)^T\mathbf{R}(k)^{-1}\mathbf{H}(k)$$
$$i(k) = \mathbf{H}(k)^T\mathbf{R}(k)^{-1}(z(k) - h(x(k)) + \mathbf{H}(k)x(k))$$
$$\mathbf{Y}(k) = \mathbf{Y}(k) + \sum \mathbf{I}(k)$$
$$\hat{y}(k) = \hat{y}(k) + \sum i(k)$$

2.8 Unscented Informasjonsfilter

Det siste av informasjonsfilterene er "unscented" informasjonsfilter [24] (Unscented Information Filter, UIF). UIF er altså informasjonsformen av UKF.. Som tidligere benyttes lineær prediksjon også for UIF.

$$\bar{x}(k) = \mathbf{F}(k)\hat{x}(k-1) \tag{2.109}$$
$$\bar{\mathbf{P}}(k) = \mathbf{F}(k)\hat{P}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k) \tag{2.110}$$

Siden man nå har estimatet og kovariansen, må disse konverteres til henholdsvis informasjonsvektor og informasjonsmatrise.

$$\bar{\mathbf{Y}}(k) = \bar{\mathbf{P}}^{-1}(k) \tag{2.111}$$
$$\bar{y}(k) = \bar{\mathbf{Y}}(k)\bar{x}(k) \tag{2.112}$$
$$\tag{2.113}$$

De neste ligningene er dertter tilsvarende som for UKF.

$$\chi_i(k) = \begin{bmatrix} \hat{x}(k) & \hat{x}(k) \pm \Gamma\sqrt{\hat{\mathbf{P}}(k)} \end{bmatrix} \tag{2.114}$$
$$\bar{\mathbf{P}}(x) = \sum_{i=1}^{2n} W_i^c(\chi_i(k) - \bar{x}(k))(\chi_i(k) - \bar{x}(k))^T \tag{2.115}$$
$$\gamma_i(k) = h(\chi_i(k)) \tag{2.116}$$
$$\mathbf{P}_{xz}(k) = \sum_{i=1}^{2n} W_i^c(\chi_i(k) - x)(\mathcal{Y}_i(k) - z(k))^T \tag{2.117}$$
$$\tag{2.118}$$

Når man så har beregnet $\gamma_i(k)$ og $\mathbf{P}_{xz}(k)$ kan man finne informasjonsvektoren og informasjonsmatrisa. Ligningene som nå følger skiller seg noe fra de benyttet i UKF.

$$\mathbf{D}(k) = \bar{\mathbf{P}}(k)^{-1}\mathbf{P}_{xz}(k) \tag{2.119}$$

$$\hat{z}(k) = \sum_{i=1}^{2n} W_i^x \gamma_i(k) \tag{2.120}$$

$$i(k) = \mathbf{D}(k)\mathbf{R}(k)^{-1}(z(k) - \hat{z}(k) + \mathbf{D}^T(k)\bar{x}(k)) \tag{2.121}$$

$$\mathbf{I}(k) = \mathbf{D}(k)\mathbf{R}(k)^{-1}\mathbf{D}^T(k) \tag{2.122}$$

$$\tag{2.123}$$

Som for vanlig SIF beregnes $i(k)$ og $I(k)$ for alle observasjoner. Disse kan deretter enkelt summeres opp.

$$\hat{y}(k) = \bar{y}(k) + \sum i(k) \tag{2.124}$$

$$\hat{\mathbf{Y}}(k) = \bar{Y}(k) + \sum \mathbf{I}(k) \tag{2.125}$$

Og helt til slutt konverteres informasjonen tilbake til tilstandsestimat og kovarians.

$$\hat{\mathbf{P}}(k) = \hat{\mathbf{Y}}(k)^{-1} \tag{2.126}$$

$$\hat{x}(k) = \hat{\mathbf{P}}(k)\hat{y}(k) \tag{2.127}$$

Vekter:

$$\begin{aligned}
\Gamma &= \sqrt{n+\gamma} \\
\gamma &= \alpha^2(n+\kappa) - n \\
W_0^x &= \frac{\gamma}{n+\gamma} \\
W_0^c &= \frac{\gamma}{n+\gamma} + (1-\alpha^2+\beta) \\
W_i^x &= W_i^c = \frac{1}{2(n+\gamma)}
\end{aligned}$$

Prediksjon:

$$\begin{aligned}
\bar{x}(k) &= \mathbf{F}(k)\hat{x}(k-1) \\
\bar{\mathbf{P}}(k) &= \mathbf{F}(k)\hat{\mathbf{P}}(k-1)\mathbf{F}(k)^T + \mathbf{Q}(k)
\end{aligned}$$

Oppdatering:

$$\begin{aligned}
\bar{\mathbf{Y}}(k) &= \bar{\mathbf{P}}^{-1}(k) \\
\bar{y}(k) &= \bar{\mathbf{Y}}(k)\bar{x}(k) \\
\chi_i(k) &= \begin{bmatrix} \hat{x}(k) & \hat{x}(k) \pm \Gamma\sqrt{\hat{\mathbf{P}}(k)} \end{bmatrix} \\
\bar{\mathbf{P}}(x) &= \sum_{i=1}^{2n} W_i^c(\chi_i(k) - \bar{x}(k))(\chi_i(k) - \bar{x}(k))^T \\
\gamma_i(k) &= h(\chi_i(k)) \\
\mathbf{P}_{xz}(k) &= \sum_{i=1}^{2n} W_i^c(\chi_i(k) - x)(\mathcal{Y}_i(k) - z(k))^T \\
\mathbf{D}(k) &= \bar{\mathbf{P}}(k)^{-1}\mathbf{P}_{xz}(k) \\
\hat{z}(k) &= \sum_{i=1}^{2n} W_i^x\gamma_i(k) \\
i(k) &= \mathbf{D}(k)\mathbf{R}(k)^{-1}(z(k) - \hat{z}(k) + \mathbf{D}^T(k)\bar{x}(k)) \\
\mathbf{I}(k) &= \mathbf{D}(k)\mathbf{R}(k)^{-1}\mathbf{D}^T(k) \\
\hat{y}(k) &= \bar{y}(k) + \sum i(k) \\
\hat{\mathbf{Y}}(k) &= \bar{Y}(k) + \sum \mathbf{I}(k) \\
\hat{\mathbf{P}}(k) &= \hat{\mathbf{Y}}(k)^{-1} \\
\hat{x}(k) &= \hat{\mathbf{P}}(k)\hat{y}(k)
\end{aligned}$$

2.9 Filterjustering

For at Kalmanfilteret skal gi korrekte estimat, er det viktig å velge korrekte parametere. Observasjonsstøyen er typisk kjent i et målfølgingssystem, denne har man derfor gode verdier for allerede. Dette overlater selve justeringen til å skje vha prosesstøymatrisa $\mathbf{Q}(k)$. Men hva er egentlig gode parametere? Det blir her definert en del metoder for å kunne avgjøre om den aktuelle parameteren er "god nok".

2.9.1 Betingelser

I teorien skal estimatet til Kalmanfilteret konvergere til virkelig verdi. Dette er dessverre ikke alltid sant. Da Kalmanfilteret først ble introdusert i flyindustrien på 1960-tallet ble det kalt "Verste oppfinnelsen dette tiåret" [2]. Dette fordi Kalmanfilteret ofte divergerte eller produserte alt for store kovariansmatriser. Årsaken til disse problemene skyldes som oftest en eller flere av følgende feil:

- Modellfeil
- Numeriskefeil
- Programmeringsfeil

Fra definisjonen av Kalmanfilteret har man følgende.

$$\tilde{x}(k) = x(k) - \hat{x}(k) \tag{2.128}$$
$$E[\tilde{x}(k)] = 0 \tag{2.129}$$
$$E[\tilde{x}(k)\tilde{x}(k)^T] = \hat{\mathbf{P}}(k) \tag{2.130}$$

Kalmanfilteret skal altså gi oss et estimat som er lik virkelig verdi, og en kovariansmatrise som representerer usikkerheten til estimatet. For å sjekke om kalmanfilteret tilfredstiller disse kravene, er det tre forskjellige kriterier man må teste.

1. Tilstandsfeilen skal ha null i middelverdi og en størrelse som tilsvarer kovariansen som filteret har beregnet.
2. Innovasjonen må også tilfredstille forrige krav.
3. Innovasjonen må være hvit.

2.9.2 Normalisert Estimeringsfeil Kvadrert

Den første teststatistikken er normalisert estimeringsfeil kvadrert (Normalized Estimation Error Squared, NEES) [2]. NEES tester om vi har null middelverdi, og om kovariansmatrisa er korrekt.

$$\epsilon(k) = [x(k) - \hat{x}(k)]^T \hat{\mathbf{P}}(k)^{-1} [x(k) - \hat{x}(k)] \tag{2.131}$$
$$= \tilde{x}(k)^T \hat{\mathbf{P}}(k)^{-1} \tilde{x}(k) \tag{2.132}$$

Dersom filteret er konsistent vil $\epsilon(k)$ ha en kjikvadrat distribusjon med n_x frihetsgrader. n_x vil her ha størrelse lik dimensjonen til estimatet. For å kunne avgjøre om simuleringen faktisk stemmer er vi nødt til å utføre flere simuleringer. Anta at man utfører totalt N simuleringer, der man for hver simulering bergner $\epsilon_i(k), i = 1, \ldots, N$. Total teststatistikk vil da være.

$$\overline{\epsilon}(k) = \frac{1}{N} \sum_{i=1}^{N} \epsilon_i(k) \tag{2.133}$$

Man kan deretter konstruere en tosidig test, der $\overline{\epsilon}(k)$ er forventet å ha en kjikvadert fordeling med $N * n_x$ frihetsgrader.

2.9.3 Normalisert Innovasjon Kvadrert

Filterets innovasjon kan testes tilsvarende som presentert i forrige del. Vi definerer her normalisert innovasjon kvadrert (Normalized Innovation Squared, NIS) som følger [3].

$$\tilde{z}(k) = z(k) - \hat{z}(k) \tag{2.134}$$
$$\epsilon_\nu(k) = \tilde{z}(k)^T \mathbf{S}(k)^{-1} \tilde{z}(k) \tag{2.135}$$

Og tilsvarende har man for N simuleringer.

$$\overline{\epsilon_\nu}(k) = \frac{1}{N} \sum_{i=1}^{N} \epsilon_{i,\nu}(k) \tag{2.136}$$

S(k) er her beregnet av Kalmanfilteret. $\bar{\epsilon_\nu}(k)$ vil da ha en kjikvadrat fordeling med $N * n_z$ frihetsgrader der n_z er dimensjonen på observasjonene.

2.9.4 Absolutt Feil

For målfølgingssystemer er det også en teststatistisk til som er ganske viktig. Nemlig hvor feil estimatet er i forhold til sann verdi. Ofte kan denne verdien være viktigere enn om filteret faktisk er konsistent [2].

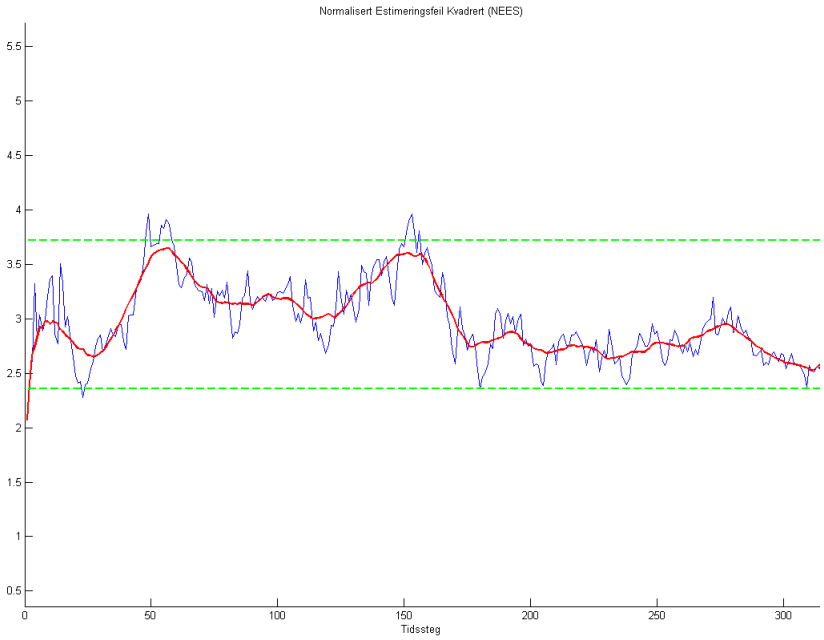
$$RMS(\tilde{x}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} ((\hat{x}_i - x)(\hat{x}_i - x)^T} \tag{2.137}$$

2.9.5 Eksempler

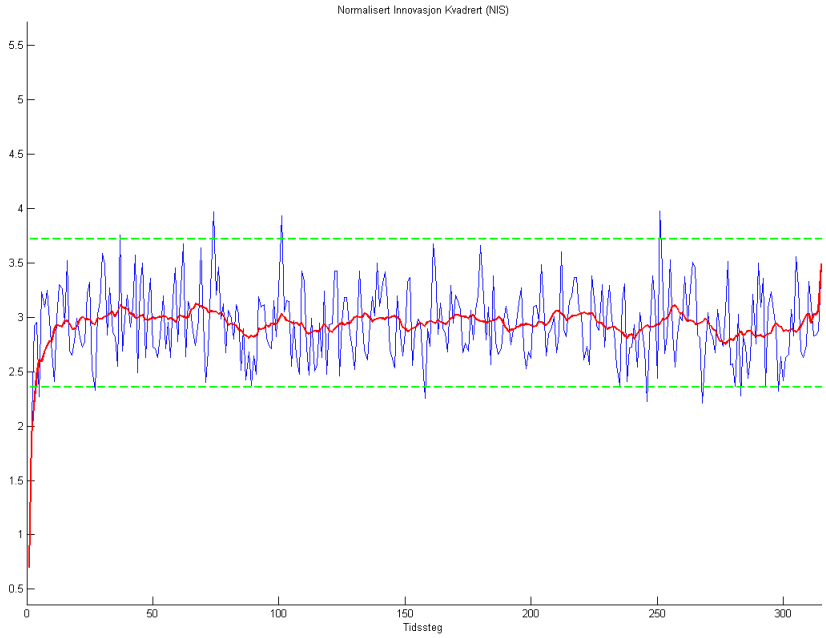
Under vises noen eksempler på kalmanfilter der man har både gode og dårlige verdier for prosesstøyen. Disse har blitt laget ved å benytte et EKF til å estimere banen til et mål som beveger seg i en rett linje med konstant hastighet [14]. Simuleringen ble kjørt totalt 100 ganger.

Korrekt Justert Kalmanfilter

I figure 2.4 og 2.5 ser man eksempler på NEES og NIS for et korrekt justert Kalmanfilter. Man kan tydelig se at hovedelen av verdiene ligger innefor det forventede 95% konfidensintervalet. Den røde kurven er kun en filtrert versjon av signalet.



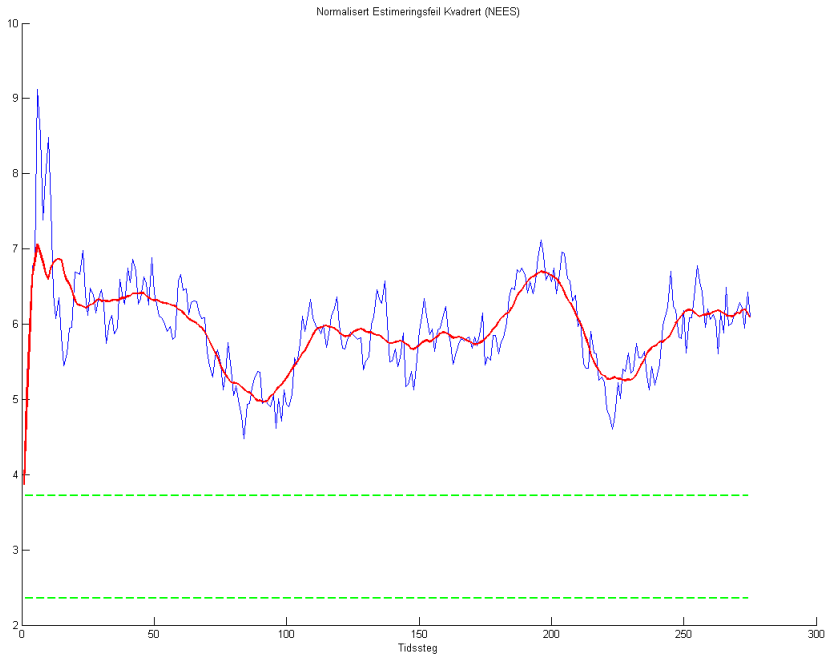
Figur 2.4: NEES for et korrekt justert Kalmanfilter



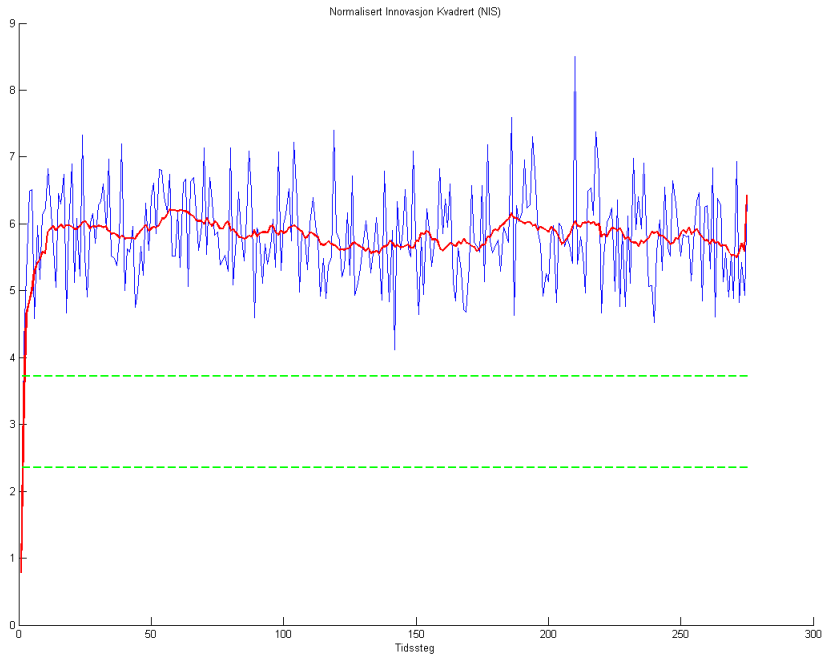
Figur 2.5: NIS for et korrekt justert Kalmanfilter

Feiljustert Kalmanfilter

I figure 2.6 og 2.7 vises et eksempel der man har for høy prosesstøy i forhold til observasjonsstøy. Man kan her tydelig se at hele kurven har flyttet seg utenfor det fastsatte konfidensintervalet.



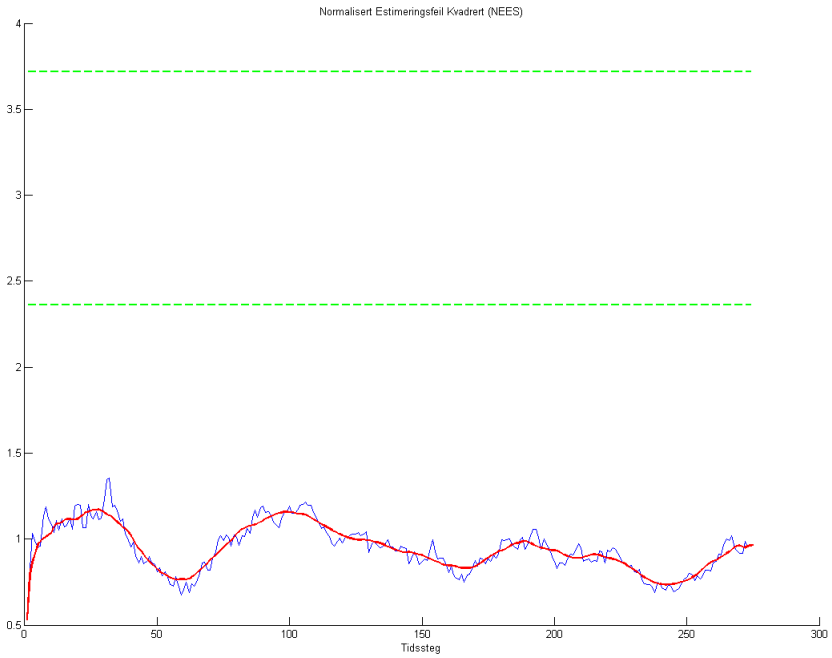
Figur 2.6: NEES for et Kalmanfilter der man har for høy prosesstøy.



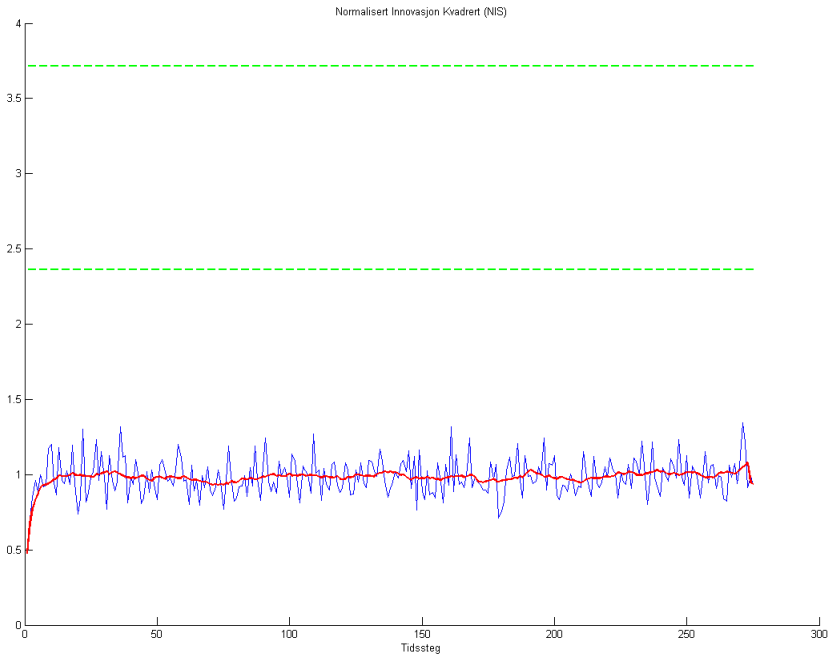
Figur 2.7: NIS for et Kalmanfilter der man har for høy prosesstøy.

Feiljustert Kalmanfilter

Tilsvarende har man i figure 2.8 og 2.9 eksempler på for lite prosesstøy. Kurven befinner seg nå under konfidensintervalet.



Figur 2.8: NEES for et Kalmanfilter der man har for lav prosesstøy.



Figur 2.9: NIS for et Kalmanfilter der man har for lav prosesstøy.

2.9.6 Manøverdeteksjon

Målfølgingsfilterene presentert tidligere fungerer utmerket dersom målet beveger seg ihenhold til den definerte modellen. Dessverre er dette som oftest ikke tilfellet i virkeligheten. Dersom et mål avviker nevneverdig fra modellen sier man at målet manøvrer. Det er utviklet mange metoder for å kunne detektere om et mål begynner å manøvrere og antagelig en tilsvarende mengde metoder for å korrigere filteret for en manøver.

En av disse metodene er å benytte en restmålingsbasert kjikvadrat detektor [30]. Når en manøver så detekteres, justerer man parameterene til Kalmanfilteret. Dette gjøres enkelt ved å øke verdiene i prosesstøymatrisa $\mathbf{Q}(k)$ og øke elementene på diagonalen i kovariansmatrisa $\hat{\mathbf{P}}(k)$. Dette har den effekten at man gir målet mer rom for manøvrering og Kalmanfilteret vil vektlegge observasjonene mer enn de estimerte verdiene. Deteksjon av en manøver gjøres som følger:

$$d^2(k) = [z(k) - \hat{z}(k)]^T \mathbf{S}(k) [z(k) - \hat{z}(k)] \tag{2.138}$$

$$= \tilde{z}(k)^T \mathbf{S}(k)^{-1} \tilde{z}(k) \tag{2.139}$$

Her er $z(k)$ faktisk en faktisk observasjon, mens $\hat{z}(k)$ er innovasjonsobservasjonen fra kalmanfilteret. I teorien holder det å beregne en enkelt verdi av $d^2(k)$. Dersom denne overstiger en gitt verdi så antar man at en manøver har inntruffet. Problemet er at algoritmen da blir fryktelig følsom for støy og feilassosiasjoner. Isteden benytter man de s siste $d^2(k)$ verdiene. Disse sammenlignes deretter mot en kjikvadrat distribusjon med $s \cdot n$ frihetsgrader. Dersom man har en høyere verdi enn kjikvadrat har man en manøver og de aktuelle verdiene for kalmanfilteret justeres.

Man har nå to muligheter for å detektere en manøver [30]. I den første metoden benytter man summen av de s siste $d^2(k)$. Jo høyere s er, desto mindre er sjansen for at man detekterer en falsk manøver. Problemet er at en for stor s, også gjør at man bruker lengre tid på å detektere en faktisk manøver.

$$d_s^2(k) = \sum_{j=k-s+1}^{k} d^2(j) \tag{2.140}$$

$$d_s^2(k) > \chi_{sn}^2(\alpha) \tag{2.141}$$

Alternativt kan man også benytte en rekursiv algoritme. α er her en parameter som reduserer effekten av tidligere $d^2(k)$.

$$d_r^2(k) = \alpha d_r^2(k-1) + d^2(k) \tag{2.142}$$

$$d_r^2(k) > \chi_{sn}^2 \tag{2.143}$$

$$\alpha \approx \frac{s-1}{s} \tag{2.144}$$

I en praktisk implementasjon risikerer man derimot at man kan få oscillasjon mellom manøver og ikke manøver. For å forhindre dette benyttes isteden et sett med grenseverdier for når en manøver oppstår og når man ikke har en manøver lenger [30]. For denne oppgaven ble det benyttet en manøvervindu med lengde 3. Dette reduserer de fleste feildeteksjonene samtidig som det ikke går for lang tid før man detekterer en faktisk manøver. Grenseverdien for deteksjon av en manøver blir da på 17. Nedre grenseverdi for avsluting av manøver ble deretter valgt 5 lavere, altså 12. Dette viste seg å gi tilfredstillende resultater under simulering.

Når man senere i oppgaven skal utvide simulatoren til å benytte 4 radarer til deteksjon av mål, så må også manøverdeteksjone modifiseres noe. Dette ble løst ved å utvide vindusstørrelsen med antall radarer. Ved 4 radar får man altså en total vindusstørrelse på $4s$. Denne ble så sammenlignet mot en kjikvadrat fordeling med $4 \cdot s \cdot n$ frihetsgrader. Med en vinduslengde på 3 gir dette øvre grenseverdi på 51 og nedre grenseverdi på 46.

	1 Radar	4 Radarer
Manøver	17	51
Ikke Manøver	12	46

Tabell 2.1: Grenseverdier for manøverdeteksjon med en vinduslengde på 3.

2.10 Multiple Samvirkende Filter

En helt annen metode for å forbedre målfølgingen ved manøvre er å benytte flere kalmanfilter som kjøres i parallell. Hvert Kalmanfilter har sin egen modell som beskriver forskjellige bevegelser til målet. Deretter benyttes det Kalmanfilteret som gir det "beste" resultatet. Ideen er at man kan aldri klare å beskrive en målbane med kun en modell. Bruken av flere forskjellige modeller er med på å sikre at man alltid har minst et filter med en modell som kan beskrive den aktuelle banen som målet følger [7].

Ideen med å benytte flere Kalmanfilter i parallell ble videreutviklet i en publikasjon av Blom i 1984 [8]. Han utviklet her en metode kalt multiple samvirkende filter (Interacting Multiple Model, IMM). Istedenfor å benytte kun resultatet fra det beste Kalmanfilteret så benytter IMM en kombinasjon av alle filtrene for å danne et felles estimat.

Utgangspunktet for IMM er å benytte en Markovkjede til å beskrive overgangen fra en modell, til en annen. Sannsynligheten for å gå fra en modell til en annen modell, er definert i en transisjonsmatrise. p_{ij} er altså sannsynligheten for å bevege seg fra tilstand i til tilstand j.

$$\mathbf{p}_{ij} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{22} & p_{33} \end{bmatrix} \tag{2.145}$$

Det er her viktig å merke seg at summen av en rad alltid er 1. Mye av utfordringen rundt IMM er hvor mange modeller man ønsker å benytte, samt sannsynligheten til transisjonsmatrisa. Følgende beskriver et steg av IMM algoritmen [7, 21]. Først må man finne kombinasjonssannsynlighetene:

$$\mu_{ij}(k-1) = \frac{\mathbf{p}_{ij}\mu_i(k-1))}{\sum \mathbf{p}_{ij}\mu_i(k-1)} \tag{2.146}$$

$$\tag{2.147}$$

Her er $\mathbf{p}_{ij}$ de forskjellige elementene i transisjonsmatrisa og μ_i er sannsyneligheten til modell i. IMM generer så nye sammensatte tilstandsestimat og kovarianser.

$$\hat{x}_j^0(k-1) = \sum \hat{x}_i(k-1)\mu_{ij}(k-1) \tag{2.148}$$

$$\hat{P}_j^0(k-1) = \sum \mu_{ij}(k-1)\left[\hat{P}_i(k-1) + [\hat{x}_i(k-1) - \hat{x}_j(k-1)]\,[\hat{x}_i(k-1) - \hat{x}_j(k-1)]^T\right] \tag{2.149}$$

De forskjellige $\hat{x}_j^0(k-1)$ og $\hat{P}_j^0(k-1)$ predikteres og oppdateres deretter av hvert sitt Kalmanfilter. Tilslutt beregnes nye kombinasjonssannsynligheter. Det antas her at man har dimensjon 3 på målingene.

$$\Lambda_j(k) = \frac{1}{(2\pi)^{3/2}\sqrt{|S_j(k)|}}\exp^{-d^2/2} \tag{2.150}$$

$$d^2 = (z(k) - \hat{z}(k))^T \mathbf{S}(k)^{-1}(z(k) - \hat{z}(k)) \tag{2.151}$$

$$\mu_j(k) = \frac{\Lambda_j(k)\sum \mathbf{p}_{ij}\mu_i(k-1)}{\sum \Lambda_j \sum \mathbf{p}_{ij}\mu_i(k-1)} \tag{2.152}$$

2.10.1 Kombinere Estimat

Ved visning av estimatet kan man benytte følgende formel for å beregne et felles estimat etter endt oppdatering.

$$\hat{x}(k) = \sum \mu_i \hat{x}_i \tag{2.153}$$

$$\hat{\mathbf{P}}(k) = \sum \mu_i \left[\hat{\mathbf{P}}_i(k) + (\hat{x}_i(k) - \hat{x}(k))(\hat{x}_i(k) - \hat{x}(k))^T\right] \tag{2.154}$$

Det gjøres oppmerksom på at dette er kun for å finne et felles estimat fra filtrene og vil på ingen måte benyttes i utregning av IMM algoritmen.

2.10.2 Validering og Assosiasjon

Validering og assosiasjon (Se kapittel 3) er ikke lenger like enkelt ved bruk av IMM algoritmen. En metode som kan benyttes er å finne felles prediktert mål

og innovasjonsmatrise for alle filterene. Disse kan så benyttes som normalt i både validering og assosiasjon.

$$\hat{z}(k) = \sum \mu_i(k)\hat{z}(k) \tag{2.155}$$

$$\mathbf{S}(k) = \sum \mu_i(k) \left[\mathbf{S}_i(k) + (\hat{z}_i(k) - \hat{z}(k))(\hat{z}_i(k) - \hat{z}(k))^T\right] \tag{2.156}$$

For de fleste tilfeller vil denne metoden fungere utmerket. En mer nøyaktig metode vil derimot være å utføre separat validering for hver enkelt modell. Deretter benyttes unionen av valideringene til å utføre en assosiasjon. En felles assosiasjonsmatrise kan nå konstrueres vha følgende [30].

$$l_{ij} = \sum \mu_i l_{ij} \tag{2.157}$$

Kapittel 3

Dataassosiasjon

I dataassosiasjon ønsker man å finne koblinger mellom observasjoner og mål. Som kjent fra tidligere benytter kalmanfilteret observasjoner til å oppdatere estimatet. Man vil få et dårligere estimat dersom man benytter en observasjon som ikke tilhører det aktuelle målet. Det finnes mange forskjellige algoritmer for dataassosiasjon. Man skiller i hovedsak mellom to hovedtyper, "hard avgjørelse" og "myk avgjørelse" [7]. I en "hard avgjørelse" assosieres hver enkelt observasjon med kun et mål, mens i en "myk avgjørelse" kan flere observasjoner assosieres med det samme målet.

3.1 Validering

Validering er en teknikk der man ønsker å eliminere usansynnelige observasjon-til-mål assosieringer. Dette gjøres ved at man definerer et område rundt hvert mål. Alle observasjoner innenfor dette området blir da vurdert som mulige kandidater for assosiasjon med det aktuelle målet. Selve assosiasjonsprossesen er i utgangspunktet ikke avhengig av at en validering blir utført først. Validering benyttes for å redusere bruken av regnekraft i selve assosiasjonsalgoritmen. Dette er en nødvendighet for at man skal klare å prosessere alle observasjonene i sanntid. Ofte benytter et målfølgingssystem seg av flere ulike typer validering. Hver validering reduserer det aktuelle området ytterligere, men er samtidig også oftere regnemessig tyngere å utføre.

3.1.1 Hastighetsvalidering

Ved hastighetsvalidering ser man på hastigheten som målet har dersom det assosieres med den gjeldende observasjonen. Hastigheten blir beregnet ved å finne avstanden mellom en observasjon og målets tidligere kjente posisjon. Deretter beregnes hastigheten mellom de to punktene ved å dele på tiden. Denne hastigheten må være mindre enn en definert maksimal hastighet. Dersom dette kravet er tildfredstilt er observasjonen en mulig kandidat for assosiasjon med målet. Maksimal hastighet er her en justeringsparameter. Det er her viktig å merke seg at observasjonen z(k) først må konverteres til kartesiskekoordinater.

$$\frac{||z(k) - \hat{x}(k-1)||}{\Delta t} < v_{maks} \tag{3.1}$$

3.1.2 Ellipsoidevalidering

Ved ellipsoidevalidering lager man en "boble" rundt estimatet. Eventuelle observasjoner som havner innefor denne "boblen" er mulige kandidater for assosiasjon med målet. For å beregne størrelsen på "boblen" benyttes innovasjonsmatrisen til Kalmanfiltert, samt prediktert observasjon.

$$d^2(k) = [z(k) - \hat{z}(k)]^T S(k)^{-1} [z(k) - \hat{z}(k)] \tag{3.2}$$
$$= \tilde{z}(k)^T S(k)^{-1} \tilde{z}(k) \tag{3.3}$$

Definerer deretter at målingen må tilfredstille følgende [6]:

$$d^2(k) \leq G_0 \tag{3.4}$$

G_0 defineres slik [6, 7]:

$$G_0 = 2 \ln \left[\frac{P_D}{(1 - P_D)(2\pi)^{3/2} \beta \sqrt{|S|}} \right] \tag{3.5}$$
$$\beta = \beta_{NT} + \beta_{FT} \tag{3.6}$$

Her er β_{NT} og β_{FT} sannsynligheten for at observasjonen stammer fra henholdsvis et nytt mål, eller er en falsk observasjon. P_D er sannsynligheten for deteksjon av målet. Dette gir da et ellipsoidevalideringsområde som vil tilpasse seg forholdene. Som nevnt i [6] vil d^2 ha en kjikvadratfordeling. Dette gjør at man også kan benytte en fast verdi for d^2 basert på en kjikvadratfordeling.

Det er også vert å nevne at det i noen tilfeller kan være lurt å velge et større valideringsområde med vilje. Dette for å ta hensyn til eventuelle

manøvere som målet kan tenke seg å utføre. Noe litteratur forslår å benytte to valideringsområder, et for manøver og et for ikke manøver. Dette for å sikre at man får en assosiasjon dersom målet utfører en manøver.

3.2 Målpoeng

Ønsker her å definere en verdi som kan fortelle oss noe om sannsynligheten for at målet vi har er eksisterer. Dette gjøres ved å først definere sannsynlighetsforholdet slik [7, 3].

$$LR = \frac{P_{NT}}{P_{FA}} \tag{3.7}$$

Her er P_{NT} sannsynligheten for at vi har et nytt mål, mens P_{FA} er sannsynligheten for at vi har falsk alarm. Videre antar man at usikkerheten til målingene er uavhengig av de faktiskene målene, og at alle målinger er uavhengige fra skan til skan. Kombinert med at man har K skan av data tilgjengelig kan følgende uttryk settes opp.

$$LR(K) = L_0 \prod_{k=1}^{K} LR_K(k) LR_S(k) \tag{3.8}$$

Av praktiske årsaker benyttes ofte logaritmen ved implementering. Dette gjør at man får en sum av uttrykk istendefor en multiplikasjon.

$$L(K) = ln[LR(K)] \tag{3.9}$$

$$= ln(L_0) + \sum_{k=1}^{K} [LLR_K(k) + LLR_S(k)] \tag{3.10}$$

Her har vi L_0 som initiell sannsynlighet. $LLR_K(k)$ representerer bidrag fra kinematiken til målet, mens $LLR_S(k)$ representerer bidrag fra signalrelaterte verdier. Signalrelaterte verdier vil typisk være informasjon om signal-til-støy forholdet for målingene. Det blir i denne oppgaven ikke benyttet informasjon om støy-til-signal forhold. LLR_S settes derfor lik 0. $L(K)$ vil inneholde informasjon om målet fra første deteksjon og frem til gjeldende tiddsteg k. Siden vi her summerer over alle tidssteg kan algoritmen enkelt implementeres rekursivt.

$$L(k) = L(k-1) + \Delta L(k) \tag{3.11}$$

$$\Delta L(k) = \begin{cases} ln(1 - P_D) & \text{ingen oppdatering} \\ \Delta L_U(k) & \text{mål oppdatert med observasjon} \end{cases} \tag{3.12}$$

$$\Delta L_u = ln \left[\frac{P_D}{(2\pi)^{3/2} \beta_{FT} \sqrt{|S|}} \right] - \frac{d^2}{2} \tag{3.13}$$

For hver tidsperiode oppdateres altså målpoengene basert på om man har en observasjon assosiert til målet eller ikke.

3.2.1 Målinitalisering

Dersom man får en observasjon som ikke kan assosieres med et eksisterende mål antar man at observasjonen stammer fra et nytt mål. Man må da opprette et nytt mål i målfølgingsystemet og en del parametere må initialiseres. Det finnes to ulike metoder for å initialisere et nytt mål, etpunktsmetoden og topunktsdifferansemetoden [27].

I etpunktsmetoden benytter man kun en enkelt observasjon til å initialisere det aktuelle målet. Observasjonen konverteres til kartesiske koordinater og disse benyttes som initielt posisjonsestimat for målet. Hastighetsestimatet setter vi til null. Kovariansmatrisa initialiseres deretter etter følgende formel.

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & 0 \\ 0 & d\mathbf{I}_3 \end{bmatrix} \tag{3.14}$$

$$d = \frac{v_{max}^2}{3} \tag{3.15}$$

Her er $\mathbf{R}$ kovariansmatrisa til observasjonsstøyen og v_{maks} er den største hastigheten man kan tenke seg at målet har. Ved neste tidssteg etter initialisering utføres deretter assosiasjon og målfiltrering som normalt.

Den andre metoden, topunktsdifferensiering, benytter seg isteden av to observasjoner for å initialisere målet. Ved første observasjon initialiseres posisjon og hastighetestimatet likt som for etpunktsinitialisering. Når man så har assosiert en andre observasjon til målet setter man posisjon, hastighet og kovarianse slik [27].

$$\hat{x}(2) = [x_1(2), y_2(2), z_3(2), \frac{v_{x_1}(2) - v_{x_1}(1)}{\Delta t}, \frac{v_{y_2}(2) - v_{y_2}(1)}{\Delta t}, \frac{v_{z_3}(2) - v_{z_3}(1)}{\Delta t}]^T \tag{3.16}$$

$$\mathbf{P}(2) = \begin{bmatrix} \mathbf{R}(2) & \frac{\mathbf{R}(2)}{\Delta t} \\ \frac{\mathbf{R}(2)}{\Delta t} & \frac{\mathbf{R}(1)+\mathbf{R}(2)}{\Delta t^2} \end{bmatrix} \tag{3.17}$$

Her er $R(1)$ og $R(2)$ kovariansmatrisa for målestøyen ved henholdsvis tid 1 og 2. Disse er i denne oppgaven like. Umidelbart skulle man tro at topunktdifferansemetoden gir det beste initiele estimatet. Det er da også topunktsdifferansemetoden som er benyttet i de fleste implementasjoner. Simuleringer utført av Mallick et al [27] viser derimot at etpunktsmetoden oftere gir et bedre initielt estimat av posisjon og hastighet enn topunktdifferansemetoden. Kombinert med at etpunktsmetoden er enklere å implementere er det derfor blitt besluttet å benytte etpunktsinitialisering i denne oppgaven.

3.2.2 Målbekreftelse

En av tingene man benytter målpoeng til er initialisering av potensielle mål. Alle mål har i utgangspunktet 4 forskjellige mulige tilstander: Potensiell, tentativ, bekreftet og slettet.

Potensiell	Man har kun en observasjon.
Tentativ	Man har to eller flere observasjoner, men man er fortsatt ikke sikker på at man har et faktisk mål.
Bekreftet	Man er nå sikre på at man har et faktisk mål.
Slettet	Et mål som ikke lenger er aktivt.

Tabell 3.1: De ulike tilstandene et mål kan ha.

Den første verdien for målpoeng settes til følgende:

$$L(1) = ln\left(\frac{P_D\beta_{NT}}{\beta_{FT}}\right) \tag{3.18}$$

Siden dette ofte er en liten verdi, og siden initiell målpoeng ikke påvirker ytelsen til systemet nevneverdig settes denne ofte isteden for null. Dersom man benytter en initiell verdi for målpoeng må man huske å også justere grensene for målbekrefting tilsvarende. Målbekreftelse tar for seg overgangen mellom tentativ til bekreftet. Ved å benytte målpoeng defineres følgende grenser.

$$\begin{aligned} L(k) &\geq T_2 \quad \text{Bekreft mål} \\ T_1 < L(k) &< T_2 \quad \text{Tentativt mål} \\ L(k) &\leq T_1 \quad \text{Slett mål} \end{aligned}$$

Grenseverdiene bestemmes videre som følger [7]:

$$T_1 = ln\left[\frac{\beta}{1-\alpha}\right], \qquad T_2 = ln\left[\frac{1-\beta}{\alpha}\right] \tag{3.19}$$

Her har man α = sannsynligheten for bekreftelse av falskt mål og β = sannsynligheten for sletting av virkelig mål.

3.2.3 Målsletting

Målpoeng kan også benyttes til å avgjøre når et mål ikke lenger er aktivt. Dette gjøres ved å sjekke om differansen mellom gjeldende poeng og maksimalt oppnådde poeng er over en viss grenseverdi.

$$\Delta L(k) = L(k) - L_{max}(k_m) \tag{3.20}$$

$$\Delta L(k) \leq THD \qquad \text{Slett mål} \tag{3.21}$$

THD er her grenseverdien man må under før målet slettes. Ofte innenfor målfølging ønsker man å fjerne et mål som ikke mottar observasjoner i løpet av N scan. For å få til dette med målpoeng kan man bruke følgende formel for å finne grenseverdien [7, 30].

$$THD = N \log(1 - P_D) \tag{3.22}$$

Her er N antall scan uten deteksjon og P_D sannsynligheten for deteksjon. Hvis man tar utgangspunkt i at man ønsker $N = 5$ får man følgende tabell for over ulike grenseverdier for forskjellie dektesjonssannsynligheter. For beregning av $P_D = 1$ har det blitt benyttet $P_D = 0.99$. Dette fordi $\log(0) = -\infty$.

P_D	THD
1,0	-23,03
0,9	-11,51
0,8	-8,05
0,7	-6,02
0,6	-4,58

Tabell 3.2: Grenseverdier for målsletting ved ulike deteksjonssannsyneligheter.

3.3 Nærmeste Nabo Assosiasjon

En av de første og enkleste metodene for assosiasjon er nærmeste nabo assosiasjon (NN). NN benytter seg av avstanden mellom observasjon og mål for avgjøre hvilke observasjoner og mål som skal assosieres med hverandre. Noe av problemet med NN er at man kun ser på gjeldende tidssteg når man vurderer mulige assosiasjoner. Dette fører til at algoritmen ofte gir feilassosiasjoner. Dette gjelder spesielt dersom man ikke har observasjoner fra alle mål for en tidsenhet.

Istedenfor å benytte faktisk avstand mellom mål og observasjon benyttes istedenfor statistisk avstand. Statistisk avstand defineres likt som når man regner ut ellipsoidevalideringsområde [6].

$$g(k) = \frac{1}{(2 * pi)^{3/2}\sqrt{|S(k)|}} \exp^{-0.5d^2(k)} \tag{3.23}$$

$|S|$ er determinanten av innovasjonen. Denne blir utregnet av kalmanfilteret. d^2 er normalisert statistisk avstand og beregnes som følger:

$$d^2(k) = [z(k) - \hat{z}(k)]^T \mathbf{S}(k)^{-1} [z(k) - \hat{z}(k)] \tag{3.24}$$
$$= \tilde{z}(k)^T \mathbf{S}(k)^{-1} \tilde{z}(k) \tag{3.25}$$
$$\tag{3.26}$$

Her er $z(k)$ observasjonen og $\hat{z}(k)$ er observasjonsestimatet fra Kalmanfilteret. Ligning 3.23 må beregnes for alle mulige kombinasjoner av mål og observasjoner. For å forenkle beregningene en del benyttes derfor logaritmen av uttrykket. Siden alle radarene returnerer observasjoner med samme dimensjon kan man fjerne de leddene som er konstante. Det nye uttrykket blir da som følger [6].

$$l = d^2 + \ln(|\mathbf{S}(k)|) \tag{3.27}$$

	T_1	T_2	NT_1	NT_2
z_1	l_{11}	l_{12}	$\ln(\beta_{NT})$	0
z_2	l_{21}	l_{22}	0	$\ln(\beta_{NT})$

Tabell 3.3: Assosiasjonsmatrise

I noen versjoner av algoritmen velger man å utelate $\ln(|\mathbf{S}|)$. Dette leddet kan derimot være lurt å ha med da man her tar hensyn til usikkerheten til målet. Ved å ha med dette leddet gjør man algoritmen mer robust mot at man "stjeler" observasjoner fra mål man der man er mer sikre på faktisk posisjon. Etter at man har beregnet l for alle mulige assosiasjoner benyttes verdiene til å lage en assosiasjonsmatrise.

I assosiasjonmatrisa inkluderer man også muligheten for at en observasjon kan stamme fra et nytt mål β_{NT}. Den ferdige assosiasjonsmatrisa benyttes så for å finne korrekt assosiasjon. Det finnes flere muligheter for hvordan man skal løse dette. Tre muligheter blir presentert her.

3.3.1 Nærmeste Nabo

I den enkleste formen for NN kobler man sammen den observasjonen og det målet som er nærmest hverandre. Problemet her er at man kan risikere å koble en observasjon til flere mål. Siden man vet at hver observasjon kun kan tilhøre et mål er ikke dette så veldig lurt. Fordelen er at algoritmen er enkel å implementere og kjapp i bruk.

3.3.2 Suboptimal Nærmeste Nabo

Suboptimal nærmeste nabo (SNN) er en utvidelse av NN der man tar hensyn til at hver observasjon kun tilhører et mål. Algoritmen begynner med et mål og assosierer dette med den nærmeste observasjonen. Det aktuelle målet og observasjonen fjernes deretter fra assosiasjonsmatrisa og man fortsetter med neste mål. Man har nå løst problemet med at en observasjon kan assosieres med flere mål. Det som er noe spesielt med algoritmen er at assosiasjonen avhenger av hvilket mål man begynner med og rekkefølgen man vurderer målene.

3.3.3 Global Nærmeste Nabo

Også i global nærmeste nabo (GNN) algoritmen tar man hensyn til at hver observasjon kun kan tilhøre et mål. Man benytter her isteden en spesiel type algoritme for å løse dette problemet ved hjelp av assosiasjonsmatrisa [7]. I matematikken kalles slike problemer for lineære tildelingsproblemer. Aktuelle algoritmer som kan løse dette optimaliseringsproblemet er Munkres, Auksjon eller JVC. Forskjellen på disse er hastigheten de bruker på å finne den optimale løsningen. JVC regnes som den kjappeste av disse algoritmene etterfulgt av hnholdsvis Auksjon og Munkres algoritmen. Alle tre finner altså samme løsning på problemet.

GNN algoritmen kalles også ofte for enkelhypotese assosiasjon. Dette fordi man kun finner den beste løsningen. Dette er også den store svakheten til algoritmen. Den tar kun hensyn til informasjonen den har for gjeldende tidssteg. Den benytter seg ikke av noe informasjon fra tidligere tidssteg. Det er også utviklet algoritmer som benytter seg av informasjon om tidligere assosiasjoner til å ta en korrekt avgjørelse. En av disse er MHT som blir nærmere presentert senere i dennne rapporten.

3.4 PDA

GNN metoden blir ofte kalt for en hard metode. Dvs at den benytter kun en observasjon per mål. Og hver observasjon kan kun assosieres med et mål. Bar-Shalom og Tse utivklet i 1975 [4] en ny teknikk kalt probabilistisk data-assosiasjon (Probabilistic data association, PDA) [3, 7]. PDA benytter en alle-naboer teknikk. Den vurderer alle mulige observasjoner innenfor et måls valideringsområde for assosiasjon. Den benytter deretter en kombinasjon av alle observasjonene for oppdatering av målet.

$$p_{ij} = \begin{cases} \frac{b}{b+\sum_{l=0}^{N} \alpha_{il}} & j = 0 \\ \frac{\alpha_{ij}}{b+\sum_{l=0}^{N} \alpha_{il}} & j \neq 0 \end{cases} \tag{3.28}$$

$$b = (1 - P_D)\beta(2\pi)^{3/2}\sqrt{|\mathbf{S}_i|} \tag{3.29}$$

$$\alpha_{ij} = P_D e^{-d_{ij}^2/2} \tag{3.30}$$

Her er p_{ij} sannsyneligheten for at observasjon j tilhører mål i. Legg merke til at det kun er observasjoner innenfor valideringsområdet som blir benyttet i beregningen av p_{ij}. I tilfellet der j = 0 er dette sannsyneligheten for at ingen av observasjonene tilhører det aktuelle målet. Når man så har funnet p_{ij} konstruerer man en felles observasjon bestående av alle de aktuelle observasjonene.

$$\tilde{z}_i = \sum_{j=1}^{N} p_{ij}\tilde{z}_{ij}(k) \tag{3.31}$$

$$\tilde{z}_{ij} = z_j(k) - \hat{z}_i(k) \tag{3.32}$$

Nå som man har en felles observasjon følger en modifisert form for kalmanfiltrering. Ved bruk av PDA har man altså ikke noe behov for en egen målfiltreringsalgoritme, som utføres etter at man har utført en assosiasjon.

$$\mathbf{K}(k) = \mathbf{P}(k)\mathbf{H}(k)[\mathbf{H}(k)\mathbf{P}(k)\mathbf{H}(k)' + \mathbf{R}(k)]^{-1} \tag{3.33}$$

$$\mathbf{P}^*(k) = \mathbf{P}(k) - \mathbf{K}(k)\mathbf{H}(k)\mathbf{P}(k) \tag{3.34}$$

$$\mathbf{P}^0(k) = p_{i0}\mathbf{P}(k) + (1 - p_{i0})\mathbf{P}^*(k) \tag{3.35}$$

$$\mathbf{dP}(k) = \mathbf{K}(k)\left[\sum_{j=1}^{N} p_{ij}\tilde{z}_{ij}\tilde{z}_{ij}^T - \tilde{z}_{ij}\tilde{z}_{ij}^T\right]\mathbf{K}(k)^T \tag{3.36}$$

$$\hat{x}(k) = \hat{x}(k) + \mathbf{K}(k)\tilde{z}(k) \tag{3.37}$$

$$\mathbf{P}(k) = \mathbf{P}^0(k) + \mathbf{dP}(k) \tag{3.38}$$

For å forenkle notasjonen har man valgt å droppe indeksen i der det ikke er noen tvil om hvilket mål det er snakk om. Sammenlignet med et vanlig Kalmanfilter kan man se at det spesielt for beregningen av kovariansen er en endring. $P^0(k)$ er kovariansen man hadde fått dersom det kun hadde vært en observasjon man kunne assosiert med det aktuelle målet. I tillegg legger man til $dP(k)$ for å representere usikkerheten rundt at man har valgt korrekt assosiasjon, dersom man har flere observasjoner som kan assosieres med målet.

I utregning av sannsyneligheten for mulige assosiasjoner ble det benyttet en verdi β. Denne representerer tettheten til observasjonene. En mulighet for definisjon av β er kalt for ikke-parametrisk PDA. β defineres da slik.

$$\beta = \frac{N}{V_G} \tag{3.39}$$

$$V_G = \frac{4\pi}{3}\sqrt{|\mathbf{S}|}G^{3/2} \tag{3.40}$$

Man har her antatt at observasjonene har tre dimensjoner. G er verdien som ble benyttet for grense for valideringsområdet. N er antall observasjoner innenfor valideringsområdet, og $\mathbf{S}$ er innovasjonen til det aktuelle målet.

PDA har et stort problem. Den antar at det kun finnes et mål som kjemper om observasjonene. I et multimålfølgingssystem er ikke det en antagelse man kan ta. En utvidelse av PDA algoritmen som tar hensyn til at flere mål kan kjempe om de samme observasjonene er JPDA (Joint probabilistic data asssociation, JPDA).

3.5 JPDA

I motsetning til PDA tar JPDA hensyn til at det kan være flere mål i det samme området. JPDA kan ses på som en redusert versjon av MHT (som blir presentert senere). I JPDA evaluerer man alle mulige assosiasjoner mellom observasjoner og mål for hvert tidssteg. Deretter kombineres disse ved hjelp av vekting til å danne et felles estimat. Dette er forskjellig fra MHT der man beholder alle mulige assosiasjoner for hvert tidssteg.

$$P\left\{\theta_l(k)|Z(k)\right\} = \frac{1}{c}(\beta_{FT})^{N_f}(\beta_{NT})^{N_n}(P_D)^{N_c}(1-P_D)^{N_t-N_c}\prod_{j=1}^{N_c}\frac{1}{2\pi^{3/2}\sqrt{|S_j|}}\exp^{-0.5d^2} \tag{3.41}$$

$$p_{jt} = \sum_{\theta:\theta_{jt}\in\theta} P\{\theta|Z(k)\} \tag{3.42}$$

$\theta(k)$ representerer en gitt hypotese, mens $Z(k)$ inneholder alle observasjoner for gjeldende tidssteg $Z(k) = \{z_1(k), z_2(k) \ldots z_m(k)\}$. N_c er antall mål med en gyldig assosiasjon, N_t er totalt antall mål, N_n antall nye mål og N_f antall falske observasjoner. β_{FT} og β_{NT} representerer sannsyneligheten for en falsk observasjon og et nytt mål. P_D er sannsyneligheten for deteksjon. Deretter utføres assosiasjon tilsvarende som for PDA [7].

Problemet med JPDA har vist seg at den ofte konvergerer dersom man har to mål som flyr i tett formasjon. Dette fører til at man mister det ene målet [16]. Denne ulempen har ført til at man har valgt å ikke undersøke PDA og JPDA i denne oppgaven. Disse er derfor presentert her kun for å poengtere at de eksisterer, og at det er gode metoder som er mye brukt.

3.6 Multipel Hypoteseassosiasjon

En av problemene med GNN algoritmen er at den kun benytter observasjoner for gjeldende tidssteg til å ta en avgjørelse. Ved neste tidssteg har den ingen hukommelse om tidligere assosiasjoner. En ny assosiasjon utføres derfor uten å ta hensyn til tidligere assosiasjoner. Men hva skjer hvis GNN utfører en feil assosiasjon? GNN har ingen mulighet til å rette opp i denne feilen. Det er her multipelhypotesealgoritmen kommer til unnsetning. Multipelhypotesemålfølging (Multipel Hypothesis Tracking, MHT) benytter samme utgangspunkt som GNN, men istedenfor å kun finne den beste assosiasjonen mellom observasjoner og mål så finner MHT alle mulige assosiasjoner. Dette inkluderer også at en observasjon kan gi opphav til et nytt mål, eller være en ugyldig observasjon. Ved neste tidssteg benytter MHT informasjonen om assosiasjoner fra forrige tidssteg til å generere nye hypoteser. Fordelen med MHT er at den ikke trenger å ta en avgjørelse om hva som er korrekt assosiasjon ved et gitt tidssteg, dersom det oppstår usikkerhet rundt en assosiasjon kan MHT vente til flere observasjoner er tilgjengelig med å ta en avgjørelse.

Et av problemene med MHT er antall hypoteser som blir generert. For hver eneste mulige kombinasjon av assosiasjoner genereres en hypotese. Disse blir ved neste tidssteg igjen splittet i nye mulige hypoteser osv. Dette gjør at man ganske raskt for et enormt antall hypoteser, noe som igjen gjør at en sanntidsimplementasjon fort blir urealistisk. Mye av fokuset på en MHT går derfor på hvordan man kan begrense antall hypoteser uten at man reduserer effektiviteten. Dette har lenge vært det store problemet med å benytte MHT i praktiske sammenhenger. Man har ikke hatt nok tilgjengelig regnerkraft til å kunne evaluere alle hypothesene. I nyere tid har det derimot dukket opp flere teknikker for reduksjon av antall hypotheser. Regnekraften i datamaskiner har også blitt vesentlig styrket, noe som gjør at man i dag anser MHT som en fullt brukbar assosiasjonsalgoritme [5].

3.6.1 MHT Implementasjon

De første implementasjonene av MHT algoritmen var hypotesebasert [32]. Dvs at man for hver observasjon evaluerte alle mulige assosiasjoner og genererte en ny hypothese for hvert alternativ. Donald Reid innså at en slik implementasjon var altfor vanskelig regnemessig [31]. Han benyttet derfor en målbasert implementasjon av algoritmen. Bakgrunnen for denne imple-

mentasjonen er at noen mål vil eksistere i flere forskjellige hypotheser. Dette fører igjen til at man som oftest har færre mål en man har hypoteser. Hver gang nye observasjoner mottas rekalkulerer man de forskjellige hypotesene, assosiasjon utføres og målene oppdateres. Hypoteser som man nå anser som usannsynlige blir fjernet, sammen med eventuelle mål som tilhører hypotesen. De resterende målene beholdes, men selve hypotesene "glemmer" man. Isteden konstruerer man en oversikt over hvilke mål som tilhører hvilke hypoteser. Ulempen med denne implementasjonen er at man for hvert tidssteg er nødt til å rekalkulere hver hypotese slik at man kan utføre assosiasjoner, beregne sannsynligheter og fjerne usannsynlige hypoteser. Det at en målbasert MHT er beregningsmessig mye enklere enn en hypotesebasert MHT har gjort at man kun har vurdert målbasert MHT som praktisk mulig fram til nylig.

I 1968 publiserte Murty [29] en algoritme der man ikke bare fant den beste løsningen på et tildelingsproblem. Murtys algoritme kunne finne de M-beste løsningene. Dette ble utnyttet av Ingemar Cox og Sunita Hingorani i 1996 [13] til å omformulere Reids originale definisjon av MHT algoritmen. Vha Murtys algoritme kunne de nå finne de M-beste hypothesene direkte. Dette gjorde at man ikke lenger trengte å finne alle mulige assosiasjoner for en hypotesebasert MHT. Selv om Murtys algoritme finner de M-beste assosiasjonene har man fortsatt behov for å utføre en form for hypotesefjerning. I denne oppgaven fokuseres det på en hypotesebasert implementasjon av MHT ved bruk av Murtys algoritme.

3.6.2 Sannsynlighetsberegning

Selve implementasjonen av en MHT algoritme er i seg selv ganske enkelt. Problemet oppstår i at man også trenger et mål på hvor sannsynlig en hypothese er.

Vi definerer oss først et hypotese sett bestående av tidligere hypoteser og gjeldende hypotese.

$$\Theta_l(k) = \{\Theta_l(k-1), \theta_l(k)\} \tag{3.43}$$

Her er Θ et sett med hypoteser, mens θ angir gjeldende hypotese som

altså angir et sett med mulige assosiasjoner. Man kan nå sette opp et uttryk som beskriver sannsynligheten til en hypotese [34].

$$P\left\{\Theta_l(k)|Z(k)\right\} = \frac{1}{c}p\left[Z(k)|\theta_l(k), \Theta_l(k-1), Z(k-1)\right] \tag{3.44}$$

$$= \frac{1}{c}(\beta_{FT})^{N_f}(\beta_{NT})^{N_n}(P_D)^{N_c}(1-P_D)^{N_t-N_c}\prod_{j=1}^{N_c}\frac{1}{2\pi^{3/2}\sqrt{|S_j|}}\exp^{-0.5d^2} \tag{3.45}$$

Her er N_c antall mål med assosiasjoner, N_t totalt antall mål, N_f antall falske observasjoner og N_n antall nye mål. Vi har også sannsynligheten for at en observasjon er falsk β_{FT} eller tilhører et nytt mål β_{NT} Det antas her at alle mål har samme P_D, hvis ikke må man også ta hensyn til dette i beregningene [7]. c er her en normaliseringsfaktor, som sikrer at at summen av sannsynligheten til alle hypotesene alltid er 1.

3.6.3 Hypothesereduksjon

Selv om bruken av Murtys algoritme i seg selv gjør en god jobb med å redusere antall hypoteser så anbefales det å benytte flere metoder for å redusere hypoteseantallet. Dette fordi Murtys algoritme vil finne de M-beste hypotesene for hver av de forrige hypotesene. Dersom man ved første tidssteg har M mulige hypoteser, så vil man altså ved neste tidssteg få M^2 mulige hypoteser osv. En metode for å redusere hypoteseantallet er å sortere hypotesene etter sannsynligheten. Deretter beholder man de M hypotesene med størst sannsynlighet. Et potensielt problem man kan oppleve med denne metoden er at man fjerner hypoteser med relativt høy sannsynlighet. Dette kan igjen føre til at MHT algoritmen ikke gjør en optimal jobb. En bedre metode er derfor å definere seg en terskelverdi. Alle hypoteser som har en sannsynlighet under denne terskelverdien vil bli fjernet [31, 13]. Både valg av M, for Murtys algortitme, og terskelverdi vil være justeringsparametere for algoritmen.

3.6.4 Hypothese Kombinering

Enda et problem i implementasjon av MHT er at man risikerer å få to like hypoteser. For å redusere antall hypoteser bør disse kombineres til en felles hypotese. En metode for å sjekke hvordan to hypoteser er like er presentert i [6]. Første kriterie er at begge hypotesene må inneholde samme antall mål. Dersom dette er tilfellet må man finne ut hvilke mål som kan høre sammen.

I [6] vises to metoder for å detektere om to mål er like. Den første er å se på assosiasjonshistorien N-tidssteg tilbake. Dersom de samme observasjonene har blitt assosiert med målene, antar man at det er snakk om samme mål. Alternativt kan man benytte kovariansen og estimatet på følgende vis.

$$|x_{Ai} - x_{Bi}| \leq \beta\sqrt{P_{Aii} + P_{Bii}} \tag{3.46}$$

$$P_{Aii} < \gamma P_{Bii} \tag{3.47}$$

$$P_{Bii} < \gamma P_{Aii} \tag{3.48}$$

x er tilstandsestimatet til de to målene A og B, mens P er kovariansen. Notasjonen x_{Ai} indikerer at man benytter element i fra estimatvektoren til mål A. Tilsvarende for P_{Aii}. Anbefalte verdier for β og γ er henholdsvis 0.1 og 2 [6].

Dersom man konstanterer at to hypoteser er like må disse kombineres. Den enkleste metoden å slette den med lavest sannsynlighet. Sannsynligheten til den gjenværende hypotesen vil da være summen av de to sannsynlighetene. Alternativt kan man kombinere de to hypotesene til en felles hypotese. Hvert mål må da kombineres med følgende formel [6].

$$x = \frac{p_1 x_A + p_2 x_B}{p_1 + p_2} \tag{3.49}$$

$$P = \frac{p_a P_A + p_2 P_B + \frac{p_1 p_2}{p_1 + p_2}(x_A - x_B)(x_A - x_B)^T}{p_1 + p_2} \tag{3.50}$$

$$p_c = p_1 + p_2 \tag{3.51}$$

Her er p_1 og p_2 sannsynligheten til henholdsvis hypotese 1 og 2, mens p_c er

sannsynligheten til den nye hypotesa. X_A, X_B, P_A og P_B er tilstandsestimat og kovarians for mål A og B.

Kapittel 4

Multisensor Målfølging

4.1 Målfølgingsarkitektur

Når man konstruerer et multisensor målfølgingssystem vil det være flere mulige måter å konstruere kommunikasjonsnettverket mellom sensorene på. Et stadig tilbakevendende problem innen multisensornettverk er kommunikasjonsbehovet mellom sensorplattformene. Ofte har man her en begrenset båndbredde tilgjengelig for kommunikasjon og man ønsker derfor å kunne utnytte denne kapasiteten på best mulig måte, samtidig som man får det beste totalte bildet av situasjonen. I de etterfølgende seksjonene presenteres det 5 ulike måter å bygge et nettverk på. Disse er igjen delt opp i sentralisert og desentralisert. I et sentralisert nettverk er det en lokal enhet som har ansvaret for å generere det totale situasjonsbildet. I et luftvernsystem har en sentralisert struktur en del ulemper som gjør at man ikke benytter denne i praktisk sammenheng. En mer realistisk implementasjon er derfor et desentralisert system, der hver enhet selv er ansvarlig for å danne det totale situasjonsbildet.

For å illustrere kommunikasjonsbehovet for de forskjellige arkitekturene har dataene i tabell 4.1 blitt benyttet. Eksemplene gitt i dette kapittelet gir ikke et reelt overblikk over kommunikasjonsbehovet, men kan benyttes til å sammenligne de ulike arkitekturene mot hverandre.

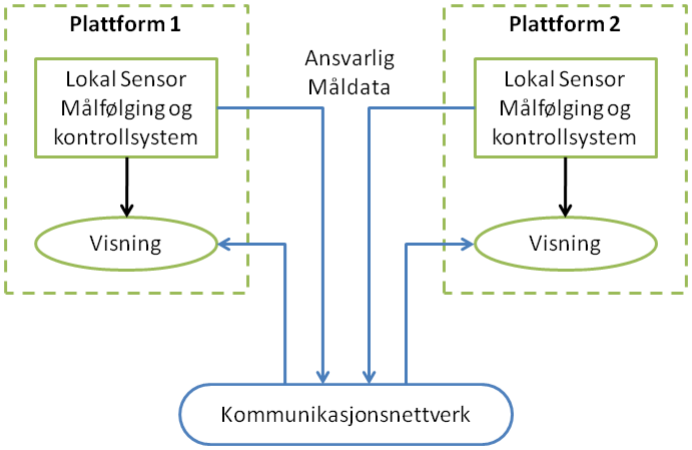
	Maks Verdi	Presisjon	Fortegnsbit	Antall Bit
Posisjon	500km	1m	✓	20bit
Hastighet	1000m/s	0,1m/s	✓	15bit
Avstand	75km	1m	✗	17bit
Asimut	360°	$0,05°$	✗	13bit
Elevasjon	90°	$0,05°$	✓	12bit
σ_R	300m	1m	✗	9bit
σ_{Az}	5°	$0,05°$	✗	7bit
σ_{El}	5°	$0,05°$	✗	7bit
Tid	86400s	0,1s	✗	20bit
Id	1000	1	✗	10bit

Tabell 4.1: Data benyttet for beregning av båndbredde.

Beregningen av antall bit som trengs gjøres som følger:

$$\text{Bit} \geq \log_2 \left(\frac{\text{Max Verdi}}{\text{Presisjon}} \right) + \text{Fortegnsbit} \tag{4.1}$$

4.1.1 Rapporteringsansvar



Figur 4.1: Rapporteringsansvar

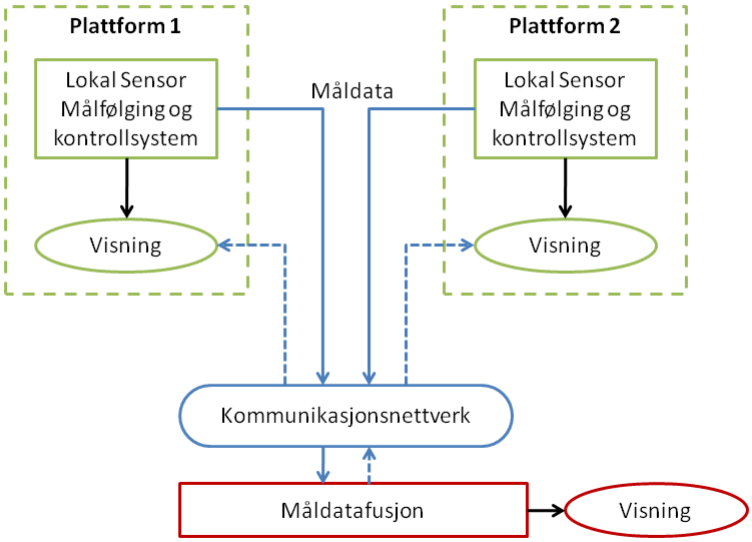
Rapporteringsansvar har den store fordelen at det er lite informasjon man trenger å kommunisere mellom hver platform. Alle platformene utfører lokal målfølging, men de har en enighet innbyrdes om hvilken platform som har best estimat av hvert enkelt mål. Den platformen som har det beste estimatet på et mål er ansvarlig for å distribuere informasjonen om dette målet til de andre sensorene. Det utføres med andre ord ingen datafusjon på platformen. Man benytter kun det estimatet man mener er best. Siden man ikke utfører noen fusjon er man heller ikke avhengig av å kommunisere annet enn helt

grunnleggende data om hvert mål. Dette sparer båndbredde på kommunikasjonsnettverket. Det at man ikke utfører noen fusjon på hver platform gjør også systemet veldig enkelt å implementere.

For hvert mål som må rapporteres antas det at man må oversende informasjon om posisjon, hastighet, tid, og et idnummer. Dette er minste antall bit man trenger for å oversende en enhet av hver type. For eksempel må man sende over tre enhter for posisjon, x,y og z posisjon. Dette fører til følgende teoretiske båndbredde for oversendelse av data for et mål.

$$R_{\text{ut}} = 3n_{\text{posisjon}} + 3n_{\text{hastighet}} + n_{\text{tid}} + n_{\text{id}} \tag{4.2}$$
$$= 135\text{bit/mål} \tag{4.3}$$

4.1.2 Sentralisert Målfølging Med Målfusjon

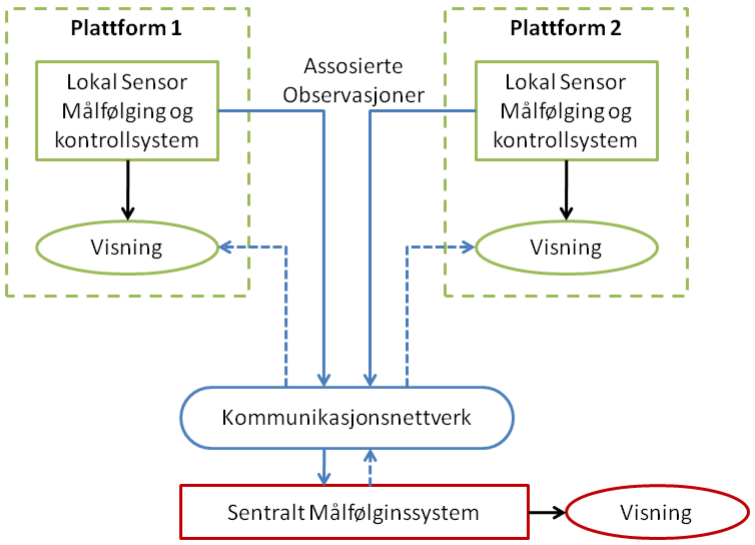


Figur 4.2: Sentralisert målfølging med målfusjon

Man har her samme oppsett som for forrige eksempel. Hver sensorplatform inneholder et lokalt målfølgingssystem. Men istedenfor å sende bekreftede observasjoner til det lokale målfølgingssystemet sender man isteden måldata. Det sentrale målfølgingssystemet mottar altså måldata fra hver enkelt sensor i nettverket og fusjonerer så dette til et felles estimat. Måldata inneholder informasjon om posisjon, hastighet, tid og idnummer. I tillegg må man også sende med kovariansmatrisen som forteller om usikkerheten til estimatet. Dersom man har 3 tilstander for posisjon og hastighet vil kovariansmatrisa være en 6x6 matrise. Siden den er symmetrisk betyr det at man må oversende totalt 21 elementer fra kovariansen. Hver av disse må representeres med 32bit for å sikre at man ikke mister for mye informasjon under overføringen. Etter at man har fusjonert informasjonen må så denne distribueres ut til de forskjellige sensorplatformene, noe som bidrar til et enda høyere kommunikasjonsbehov. Det antas her at man må sende tilbake tilstandsvektoren (som inneholder posisjon og hastighet), tid, samt en id.

$$R_{\text{inn}} = 3n_{posisjon} + 3n_{hastighet} + n_{tid} + n_{id} + 21n_{kovarians} \tag{4.4}$$
$$= 942\text{bit/mål} \tag{4.5}$$
$$R_{\text{ut}} = 3n_{posisjon} + 3n_{hastighet} + n_{tid} + n_{id} \tag{4.6}$$
$$= 135\text{bit/mål} \tag{4.7}$$

4.1.3 Sentralisert Observasjons Målfølging



Figur 4.3: Sentralisert Observasjons Målfølging

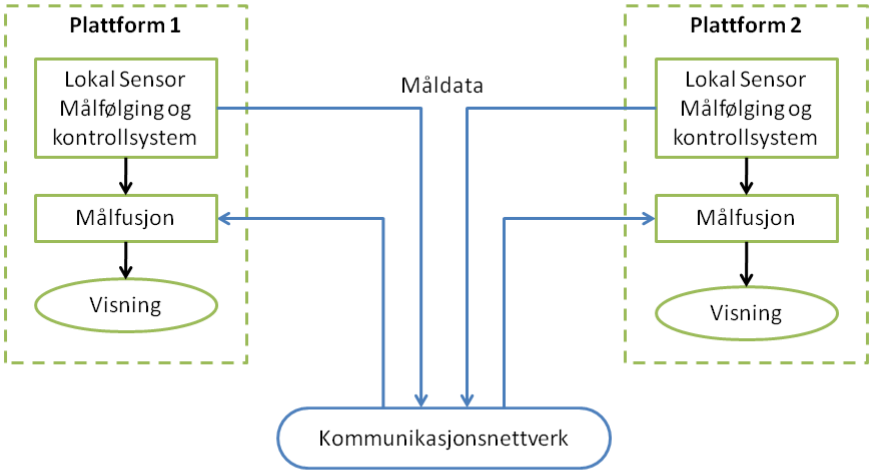
I et helt sentralisert system sendes alle observasjonsdata fra hver sensor til en lokal platform som utfører fusjon og målfølging. Informasjonen fra det sentraliserte målfølgingssystemet kan deretter sendes tilbake til hver sensor for visning, eller vises lokalt. Fordelen med denne metoden er at man har direkte tilgang til observasjonsdata fra alle sensorene i nettverket. Dette muligjør at man har en bedre sjanse til å detektere mål som kun et fåtall sensorer detekterer. Ulempen er kommunikasjonsbehovet. Dersom man har sensorer som generer mye støymålinger risikere man å benytte mye båndbredde til ingen nytte. Moderne radarer krever også styresystemer som bestemmer hvor radaren skal se etter mål. Dette krever informasjon om detekterte mål og denne informasjonen må derfor kommuniseres fra det sentrale målfølgingssystemet og ut til hver sensor. En mer praktisk implementasjon av et rent sentralisert system vil være å ha et lokalt målfølgingssystem for hver sensor. Man sender deretter kun observasjoner som man bekrefter tilhører mål til det sentrale målfølgingssystemet. Det sentrale målfølgingssystemet utfører deretter en lo-

kal fusjon av observasjonsdataene. Man vil fortsatt være avhengig av en gitt synkronisering mellom de ulike målfølginssystemen. De fusjonerte dataene må altså sendes tilbake til hver enkelt platform.

Man antar at hver sensor generer data for avstand, asimut og elevasjon, og at minimalt med støyobservasjoner blir oversendt. I tillegg må man ha informasjon om usikkerheten til hver observasjon samt tid. Den fusjonerte informasjonen må også sendes tilbake til hver platform.

$$
\begin{aligned}
n_{\text{observasjon}} &= n_{\text{avstand}} + n_{\text{asimut}} + n_{\text{elevasjon}} && (4.8)\\
&= 42\text{bit} && (4.9)\\
n_{\text{usikkerhet}} &= n_{\text{avstandsusikkerhet}} + n_{\text{asimutusikkerhet}} + n_{\text{elevasjonsusikkerhet}} && (4.10)\\
&= 23\text{bit} && (4.11)\\
R_{\text{inn}} &= n_{\text{observasjon}} + n_{\text{usikkerhet}} + n_{\text{tid}} && (4.12)\\
&= 85\text{bit/mål} && (4.13)\\
R_{\text{ut}} &= 3n_{\text{posisjon}} + 3n_{\text{hastighet}} + n_{\text{tid}} + n_{\text{id}} && (4.14)\\
&= 135\text{bit/mål} && (4.15)
\end{aligned}
$$

4.1.4 Distribuert Målfølgingsfusjon

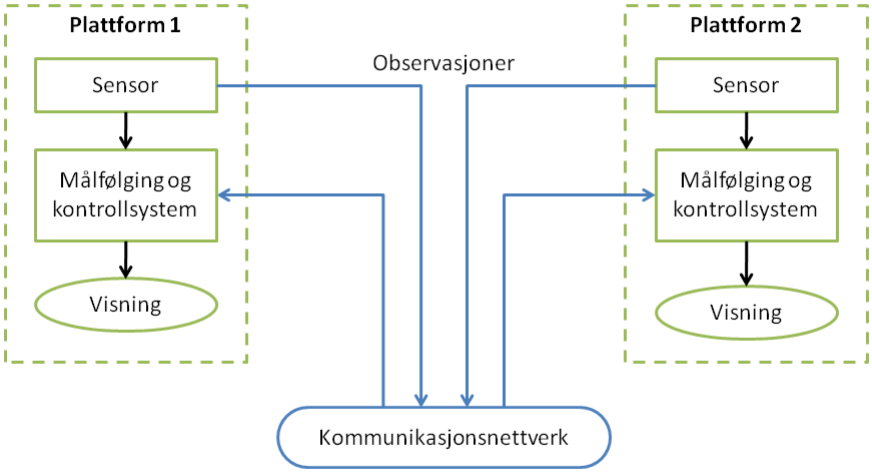


Figur 4.4: Distribuert Måldatafusjon

Her har man ikke lenger noe sentralt målfølgingssystem. Isteden inneholder hver plattform sitt eget målfølgingssystem og kommuniserer måldata til alle andre plattformer i nettverket. Fordelene er at siden hver platform til enhver tid mottar måldata fra alle andre plattformer er det ikke lenger noe behov for å kommunisere data tilbake ved hvert scan. Det kan derimot hende at man med jevne tidsrom må synkronisere de ulike plattformene. En ulempe er derimot at hver platform må være forsiktig så den ikke fusjonerer samme data flere ganger. Dette kan skje ved at feks sensor 1 sender data til sensor 2 og sensor 3. Sensor 2 sender deretter de fusjonerte dataene til sensor 3. Men sensor 3 har allerede fusjonert dataene fra sensor 1. Metoder for å fjerne denne redundante informasjonen må derfor benyttes.

$$R_{\text{inn}} = 3n_{posisjon} + 3n_{hastighet} + n_{tid} + n_{id}) + 21n_{kovarians} \quad (4.16)$$
$$= 807\text{bit/mål} \quad (4.17)$$
$$R_{ut} = 3n_{posisjon} + 3n_{hastighet} + n_{tid} + n_{id}) + 21n_{kovarians} \quad (4.18)$$
$$= 807\text{bit/mål} \quad (4.19)$$
$$(4.20)$$

4.1.5 Distribuert Observasjons Målfølging



Figur 4.5: Distribuert Observasjonsfusjon

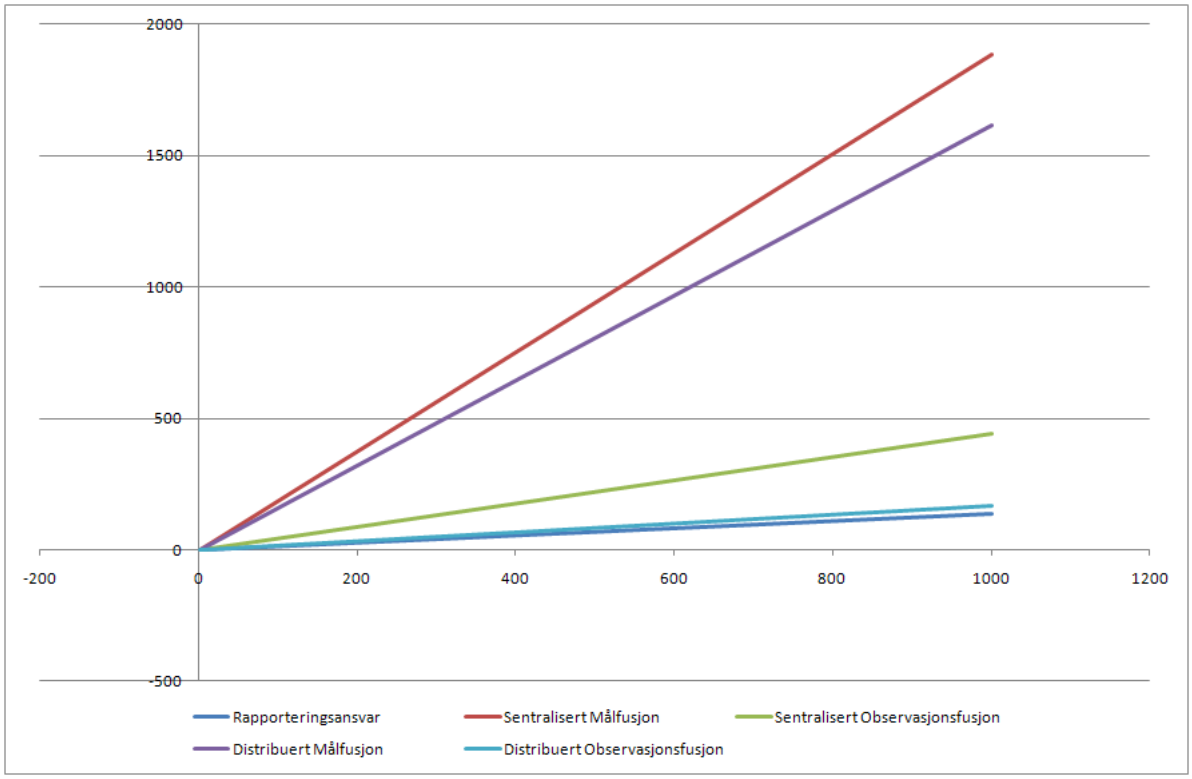
I det siste eksempelet har hver plattform sitt eget lokale målfølgingssystem. Hver platform kommuniserer bekreftede observasjoner til alle andre plattformer i nettverket. I teorien sikrer dette at man har det samme luftbildet på hver enkelt plattform. Behovet for synkronisering mellom plattformene skulle derfor være minimal. En stor fordel med denne metoden er at hver enkelt

plattform kan avgjøre selv om den mener at det er fornuftig å kommunisere en observasjon til de andre plattformene. For eksempel kan en observasjon inneholde så store unøyaktigheter at den vil forverre det totalte luftbildet for det aktuelle målet.

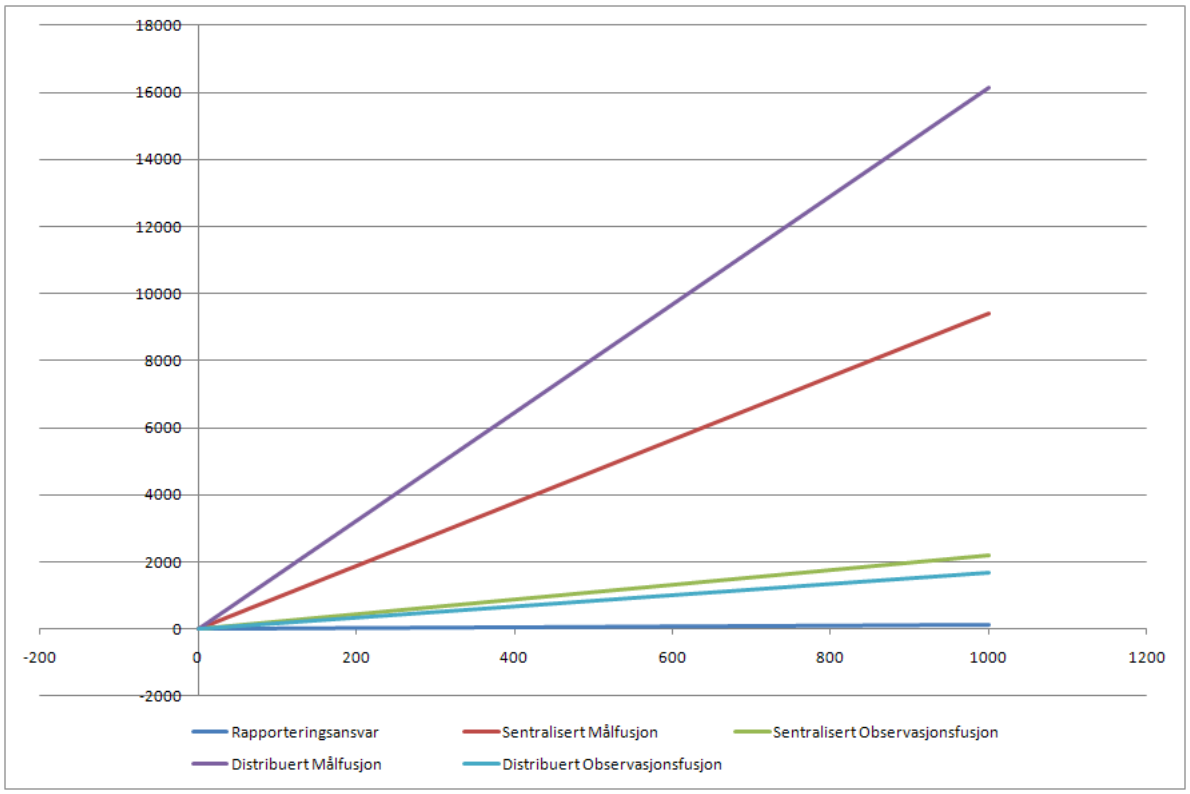
Hver observasjon som kommuniseres inneholder avstand, asimut og elevasjon, samt usikkerheten til disse. I tillegg er det en tidsmerking og et idnummer. Idnummeret sikrer at man har et felles luftbilde for alle plattformene.

$$
\begin{aligned}
n_{\text{observasjon}} &= n_{\text{avstand}} + n_{\text{asimut}} + n_{\text{elevasjon}} && (4.21)\\
&= 42\text{bit} && (4.22)\\
n_{\text{usikkerhet}} &= n_{\text{avstandsusikkerhet}} + n_{\text{asimutusikkerhet}} + n_{\text{elevasjonsusikkerhet}} && (4.23)\\
&= 23\text{bit} && (4.24)\\
R_{\text{inn}} &= n_{\text{observasjon}} + n_{\text{usikkerhet}} + n_{\text{tid}} && (4.25)\\
&= 85\text{bit}/\text{mål} && (4.26)\\
R_{\text{ut}} &= n_{\text{observasjon}} + n_{\text{usikkerhet}} + n_{\text{tid}} && (4.27)\\
&= 85\text{bit}/\text{mål} && (4.28)
\end{aligned}
$$

4.2 Oppsummering



Figur 4.6: Kommunikasjonsbehov for 2 radarer. Y-aksen er båndbredde i kbps, mens X-aksen er antall mål.

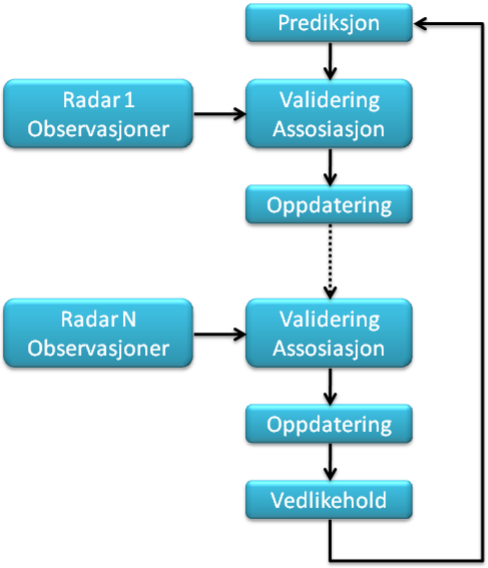


Figur 4.7: Kommunikasjonsbehov for 10 radarer. Y-aksen er båndbredde i kbps, mens X-aksen er antall mål.

I figur 4.6 og 4.7 kan man se en oversikt over minste nødvendige båndbredder for et nettverk bestående av henholdsvis 2 og 10 sensorer. Av dette ser man atrapporteringsansvar den minste kommunikasjonen i et sensornettverk. Kommunikasjonsbehovet vil heller ikke øke med antall sensorer i nettverket, noe som kan være en stor fordel. Den store ulempen er at man ikke utfører noen som helst form for datafusjon. Dette kan gi et dårligere totalt bilde av situasjonen. Spesielt gjelder dette dersom man benytter sensorer med forsjkellig usikkerhet knyttet til observasjonene. En ren sentralisert implementasjon vil ofte ikke være praktisk annen enn innefor små geografiske områder. En sentralisert implementasjon har også den ulempen at hele nettverket er avhengig av en sentral prosesseringsenhet. Dersom noe går feil i denne enheten risikerer man å miste det totale bildet av situasjonen. En mer realistisk løsning er derfor å benytte et desentralisert system. Her er hver enhet selv ansvarlig for å danne det totale luftbildet. Hvis man ser på

kommunikasjonsbehovet ser man tydelig at observasjonsfusjon gir det klart minste kommunikasjonsbehovet. Det er da viktig å merke seg at man kun har tatt hensyn til at assosierte observasjoner kommuniseres over nettverket.

4.3 Multisensor



Figur 4.8: Multisensor [6]

I denne oppgaven har man valgt å behandle dataene fra de ulike sensorene sekvensielt. Dette er presentert i figur 4.8. Ved hver tidsenhet blir eksisterende mål prediktert frem. Når man så mottar observasjoner fra en radar utføres validering og assosiasjon. Måldataene blir deretter oppdatert med de assosierte observasjonene fra radaren. For observasjonene fra den neste radaren utføres igjen validering og assosiasjon. Deretter oppdateres estimatet. Fordelen her er at man tar hensyn til informasjon som hver radar bidrar med. Typisk vil observasjoner som den første radaren bidrar med føre til at usikkerheten til estimatet blir redusert. Dette gjør at man får en bedre assosiasjon med observasjonene fra den neste radaren.

4.3.1 Multisensor Kalmanfilter

I et multisensorsystem må man ha en metode for å kunne kombinere observasjoner fra flere ulike sensorer til et felles estimat. Tidligere har Kalmanfilteret blitt benyttet til å beregne estimatet ved hjelp av kun en observasjon per tidssteg. Men kalmanfilteret kan også benyttes til å kombinere observasjoner og danne et felles estimat. Det er totalt tre forskjellige metoder man kan benytte [7].

1. **Sekvensiel oppdatering.** Hver observasjon blir behandlet som om den var den eneste. For hver enkelt observasjon utføres altså et komplett korreksjonssteg av kalmanfilteret.

2. **Kompositt Observasjoner.** Observasjonene, og kovariansmatrisene kombineres sammen til en felles kompositt observasjon. Denne ene kompositt observasjonen benyttes deretter til korreksjon vha av vanlig Kalmanfilter algoritme.

$$R_c^{-1} = \sum_{i=1}^{S} R_i^{-1} \tag{4.29}$$

$$z_c = R_c \sum_{i=1}^{S} R_i^{-1} z_i \tag{4.30}$$

3. **Informasjonsfiltrering.** Man benytter informasjonsformen til kalmanfilteret. Denne metoden har blitt presentert i seksjon 2.6

4.4 Sensorregistrering

Et av problemene som må løses i et multisensorsystem er relatert til posisjonering av radarene. Man kan ikke måle nøyaktig posisjon og tilt til radarene. Alle radarer vil ha en feil i rapportert posisjon og og dermed også i observasjonene. Denne feilen kan over tid utvikle seg til å bli ganske så stor og i verste tilfelle føre til "spøkelsesmål" [7]. For det samlede målfølgingssystemet vil det se ut som om man har flere mål, selv om det kun er snakk om et enkelt mål. Man må altså ha en metode for å kunne estimere og korrigere for dette problemet

Vi skiller mellom to mulige bias estimeringsmetoder. Absolut og relativ. Absolut biasestimering går utifra at alle sensorer har bias feil og estimerer bias for hver enkelt sensor. Absolut bias kan ofte være vanskelig å beregne, men er også den metoden som er best egnet for et distribuert system. Relativ biasestimering går utifra at man har en sensor (eller en annen referanse) uten bias feil. Alle bias estimat er relativt til denne [1]. Det er i hovedsak tre typer bias feil.

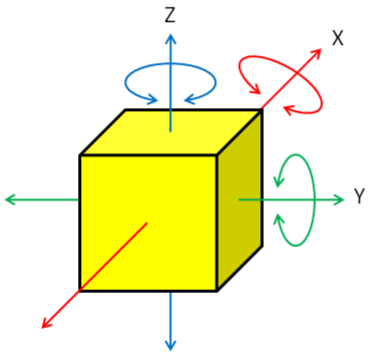
1. Posisjonsfeil.
2. Tiltfeilt.
3. Målefeil.

Posisjonsfeil skyldes at man ikke klarer å nøyaktig angi radarens koordinater. Man får altså en feil i x,y og z retning. Moderne radarer benytter som oftest en kombinasjon av treghetsmålere og GPS til å finne sin egen posisjon. Dette gjør at man klarer å finne en ganske så nøyaktig angivelse av posisjonen til radaren. Den feilen vil derfor påvirke observasjonsdatene minimalt.

Tiltfeil skyldes også plasseringen av radaren, men i motsetning til posisjonsfeil menes det her vinklene radaren er plassert i. Man vil ha et avvik i asimut grunnet feil innretning mot nord, og feil i elevasjon dersom radaren ikke står helt vannrett. Grunnet de lange avstandene som moderne radarsystemer måler over, så kan disse føre til ganske så store feil i ytterkant av radarens deteksjonsområde.

Den siste er målefeil og skyldes selve radaren. Radaren har en gitt nøyaktighet den klarer å rapportere fra hver observasjon. Dette skyldes flere for-

hold, både radarens oppbygning og forhold utenfor radarens kontroll. Dette er imidlertid feil som vises i asimut, avstand og elevasjon. I figur 4.9 kan man se hvor de forskjellige feilene oppstår.



Figur 4.9: Figuren viser hvordan man kan posisjonere og rotere et objekt i 3-dimensjoner. Det er også her man har opphavet til posisjonerings og tiltfeil i et radarsystem. I tillegg har man målefeil på selve observasjonen.

4.4.1 Kvasirekursiv Filtrering

Denne algoritmen har blitt beskrevet av Egils Sviestins i [36]. Den har siden blitt forbedret og analysert av Chittella et al [12]. Algoritmen beregner rekursivt et sett med vektorer på formen $(U + V)m = b$. Her er b en vektor inneholdende biasene man ønsker å finne og V er en statisk matrise. Noe av fordelene med algoritmen er at man beregner U og m hver gang man mottar et observasjonssett. Selve biasvektoren trenger man derimot ikke å finne før man har behov for den! Feks kan man motta observasjoner hvert 10sekund, men kun beregne selve biasvektoren hvert 60sekund. For å kunne beregne U

og m, stiller algoritmen noen krav til observasjonene. Flere observasjoner blir kombinert sammen til et observasjonssett. Hvert enkelt observasjonssett har følgende krav.

1. Alle observasjonene i et observasjonssett må komme fra samme virkelige mål.
2. Observasjonene må komme fra minst to ulike sensorer.
3. Observasjonssettet kan ikke inneholde to eller flere observasjoner fra den samme sensoren.
4. Alle observasjoner må være relativt nærme hverandre i tid. (Dette vil typisk si innenfor samme sensorscan).
5. Ingen observasjon kan eksistere i to ulike observasjonssett.

Algoritmen tar utgangspunktet i at man ønsker å minimere følgende funksjon.

$$W = \sum_{i,s} ||z_{is} - h_s(x_i, b)||^2_{Y_{is}} + ||b - \hat{b}||^2_V \tag{4.31}$$

Indeksen i og s representerer henholdsvis observasjonssett i og sensor s. En enkelt observasjon er z_{is} og posisjons estimatet for det gjeldende målet er x_i. b og $\hat{b}$ er biasvektorer inneholdende de biasverdiene man ønsker å beregne. $\hat{b}$ er apriori informasjonen om biasverdiene. Typisk vil denne vektoren være lik null dersom man ikke har none kjennskap til faktiske biasverdier. V er en apriori informasjonsmatrise. Denne representere den inverse kovariansmatrisa før man har mottatt noen observasjoner. V vil derfor ha følgende form.

$$\mathbf{V} = diag(\sigma_1^{-2}, \sigma_2^{-2}, \ldots, \sigma_N^{-2}) \tag{4.32}$$

Her er N antall biasparameterer man ønsker å finne, og σ er forventet standardavvik for hver biasparameter. Tilslutt har man $\mathbf{Y}_{is}$ som representerer

den inverse observasjons kovariansmatrisa. $\mathbf{R}_{is}$ er den samme som brukt i kalmanfilteret for observasjonsstøy.

$$\mathbf{Y}_{is} = \mathbf{R}_{is}^{-1} \tag{4.33}$$

Fullstendig utledning av hvordan man minimerer ligning 4.31 er presentert i [36]. Det blir her derfor kun presentert de utledede ligningene. Når man har et gyldig observasjonssett beregner algoritmen ligningene i den rekkefølgen som gjengitt under. Første steg er å finne differansen mellom faktisk observasjon, og forventet observasjon. Følgende blir derfor beregnet.

$$\hat{z}_{is} = h_s(\hat{x}_i + \hat{r}_i(t_{is} - \hat{t}_i), 0) \tag{4.34}$$
$$d_{is} = z_{is} - \hat{z}_{is} \tag{4.35}$$
$$\tag{4.36}$$

h_s er en konvertering fra globalt estimat til lokal observasjon. Funksjonen $\hat{r}$ tar hensyn til eventuelle tidsforskjeller mellom observasjonene. $t_{is} - \hat{t}_i$ representerer derfor tidsdifferansen mellom lokal sensortid og tiden man utførte observasjonen. $\hat{r}$ predikterer deretter estimatet til riktig posisjon. For denne oppgaven vil $\hat{r}$ være en en lineær funksjon som benytter hastigheten til estimatet, $v_{x_i}\Delta t$, for å korrigere for tidsforskjellen.

Deretter lineariseres systemet mtp henholdsvis estimatet og biasvektoren. Dersom man har et system bestående av to radarer og ønsker å estimere bias for avstand, asimut og elevasjon vil disse se slik ut.

$$r(k) = \sqrt{x(k)^2 + y(k)^2 + z(k)^2} \tag{4.37}$$

$$rh(k) = \sqrt{x(k)^2 + y(k)^2} \tag{4.38}$$

$$\mathbf{H}_{is} = \frac{\partial h_s(x_i, 0}{\partial x_i}|_{x_i=\hat{x_i}} \tag{4.39}$$

$$= \begin{bmatrix} \frac{x(k)}{r(k)} & \frac{y(k)}{r(k)} & \frac{z(k)}{r(k)} \\ \frac{y(k)}{rh(k)^2} & -\frac{x(k)}{rh(k)^2} & 0 \\ -\frac{x(k)z(k)}{r(k)^2 rh(k)} & -\frac{y(k)z(k)}{r(k)^2 rh(k)} & \frac{rh(k)}{r(k)^2} \end{bmatrix} \tag{4.40}$$

$$\mathbf{A}_{is} = \frac{\partial h_s(\hat{x_i}, 0}{\partial b}|_{b=0} \tag{4.41}$$

$$\mathbf{A}_{i1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.42}$$

$$\mathbf{A}_{i2} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.43}$$

$$\bar{d}_{is} = \mathbf{H}_{is} \left(\sum_{t=1}^{s} \mathbf{H}_{it}^T \mathbf{Y}_{it} \mathbf{H}_{it} \right) - 1 \sum_{t=1}^{s} \mathbf{H}_{it} \mathbf{Y}_{it} d_{it} \tag{4.44}$$

$$\bar{\mathbf{A}}_{is} = \mathbf{H}_{is} \left(\sum_{t=1}^{s} \mathbf{H}_{it}^T \mathbf{Y}_{it} \mathbf{H}_{it} \right) - 1 \sum_{t=1}^{s} \mathbf{H}_{it} \mathbf{Y}_{it} \mathbf{A}_{it} \tag{4.45}$$

I noen tilfeller risikerer man at matriseinverteringen i ligningene over ikke er mulig. Dersom denne situasjonen oppstår må man avbryte beregningen og unnlate å benytte resultatet i biasberegningene. Tilslutt beregner man U og m.

$$\mathbf{U} = \sum_{i,s} (\mathbf{A}_{is} - \bar{\mathbf{A}}_{is})^T \mathbf{Y}_{is} (\mathbf{A}_{is} - \bar{\mathbf{A}}_{is}) \tag{4.46}$$

$$m = \sum_{i,s} (\mathbf{A}_{is} - \bar{\mathbf{A}}_{is})^T \mathbf{Y}_{is} (d_{is} - \bar{d}_{is}) \tag{4.47}$$

$$\tag{4.48}$$

Hvert observasjonssett bidrar altså med mere informasjon til estimatoren. Når man har behov for å beregne selve biasvektoren løser man systemet

$(U + V)b = m.$

$$b = (\mathbf{U} + \mathbf{M})^{-1} m \tag{4.49}$$

Algoritme 6 Kvasibiasfilter

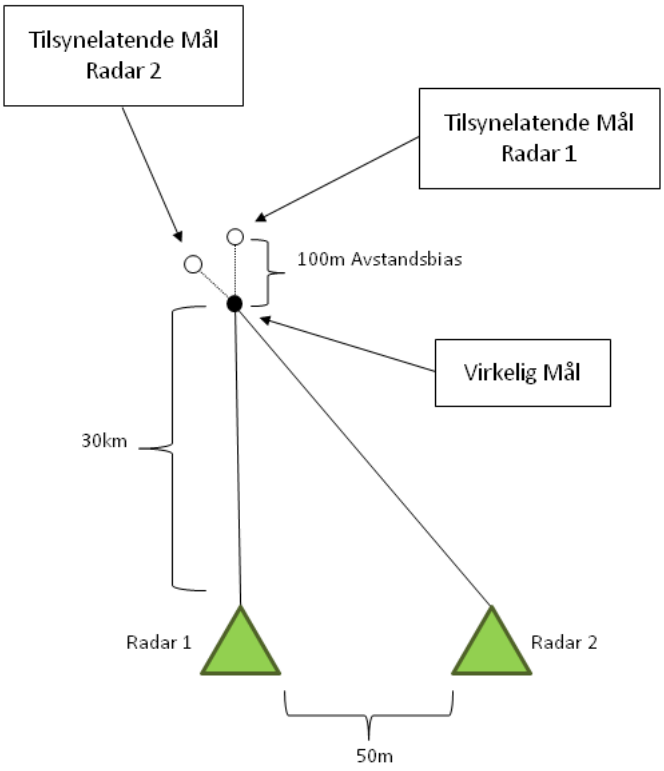
$$\hat{z}_{is} = h_s(\hat{x}_i, 0) \tag{4.50}$$
$$d_{is} = z_{is} - \hat{z}_{is} \tag{4.51}$$
$$H_{is} = \frac{\partial h_s(x_i, 0)}{\partial x_i}|_{x_i = \hat{x_i}} \tag{4.52}$$
$$A_{is} = \frac{\partial h_s(\hat{x}_i, 0)}{\partial b}|_{b=0} \tag{4.53}$$
$$\overline{d}_{is} = H_{is} \left(\sum_{t=1}^{s} H_{it}^T Y_{it} H_{it} \right)^{-1} \sum_{t=1}^{s} H_{it} Y_{it} d_{it} \tag{4.54}$$
$$\overline{A}_{is} = H_{is} \left(\sum_{t=1}^{s} H_{it}^T Y_{it} H_{it} \right)^{-1} \sum_{t=1}^{s} H_{it} Y_{it} A_{it} \tag{4.55}$$
$$U = \sum_{i,s} (A_{is} - \overline{A}_{is})^T Y_{is} (A_{is} - \overline{A}_{is}) \tag{4.56}$$
$$m = \sum_{i,s} (A_{is} - \overline{A}_{is})^T Y_{is} (d_{is} - \overline{d}_{is}) \tag{4.57}$$
$$b = (U + M)^{-1} m \tag{4.58}$$

4.5 Observerbarhet

Et problem med å finne biasparametere er å finne i observerbarheten til systemet. Ofte treffer man situasjoner der biasparameterene er slik at man ikke kan observere de direkte. Man har derfor heller ikke mulighet til å gjøre en nøyaktig estimering av biasparameterene.

Eksempel 2: Observerbarhet

I referanse [12] finner man en god forklaring på hvorfor man ofte ikke får korrekt estimering av biasparameterene.



Figur 4.10: Observerbarhet for radarbias

I figur 4.10 illustrueres dette i et forenklet eksempel. To radarer observerer et felles mål. Begge radarene har en avstandsbias på 100meter. Radar 1 er plassert i senter for koordinatsystemet, mens radar 2 er plassert 50 meter østenfor. For radar 1 måler man da en avstand til målet på 30100 meter nord, mens radar 2 har en avstand til målet på ca 30067 meter nord og 50.16 meter vest. Estimeringsalgoritmen klarer derimot ikke å tolke disse målingene korrekt. Istedenfor å estimere en avstandsbias på 100 meter vil den finne en asimutbias på 0.005mRad. Estimeringsalgoritmen har heller ikke mulighet til å avgjøre hvilken radar biasen tilhører, den estimerer derfor avstandsbiasen til 0 meter for begge radarene og gir hver radar en asimut bias på henholdsvis -0.0025mRad og +0.0025mRad.

4.6 Tidsforsinkede Observasjoner

I et multisensorsystem har man ingen garanti for at observasjonene kommer i korrekt rekkefølge. Dette kan for eksempel skyldes forsinkelser i kommunikasjonsnettverket. Ofte antar man i simuleringer at sensorene er synkronisert og at alle observasjoner blir gjort ved samme tid. I praksis er ikke dette tilfellet. Dersom en observasjon ankommer for sent i tid får man en negativ tidsoppdatering i målfølgingsfilteret. Problemet ligger da i at man må tilbakedatere målet. Det er i hovedsak to metoder man kan benytte. Den enkleste er å benytte en algoritme som tilbakedaterer målet, deretter integrerer observasjonen og så fremoverdaterer målet [7]. Følgende algoritme utfører dette og er basert på Kalmanfilter.

$$\hat{x}(k) = \mathbf{F}(t_k, t_{k-1})\hat{x}(k-1) \tag{4.59}$$
$$\mathbf{P}(k) = \mathbf{F}(t_k, t_{k-1})\mathbf{P}(k-1)\mathbf{F}(t_k, t_{k-1})^T \tag{4.60}$$
$$\nu(k) = z(k) - \mathbf{H}(k)\hat{x}(k) \tag{4.61}$$
$$\mathbf{S}(k) = \mathbf{H}(k)\mathbf{P}(k)\mathbf{H}(k)^T + \mathbf{R}(k) \tag{4.62}$$
$$\mathbf{K}(k) = \mathbf{P}(k)\mathbf{H}(k)\mathbf{S}(k)^{-1} \tag{4.63}$$
$$\hat{x}(k) = \hat{x}(k) + \mathbf{K}(k)\nu(k) \tag{4.64}$$
$$\mathbf{P}(k) = \mathbf{P}(k) - \mathbf{K}(k)\mathbf{S}(k)\mathbf{K}(k)^T \tag{4.65}$$

I denne algoritmen overser man prosesstøyen. Det er også utviklet metoder der man tar høyde for prosesstøyen [7].

Den andre metoden er å beholde en komplett oversikt over systemet en viss tid tilbake. Når man mottar tidsforskjøvede observasjoner benytter man derfor informasjon fra det aktuelle tidsrommet til å integrere målingen, deretter utføres en komplett assosiasjon/filterering fram til gjeldende tidssteg. Ulempen med denne metoden er at den krevere mye minne og prosesseringskraft. For større systemer kan dette være en tilnærmet umulig oppgave.

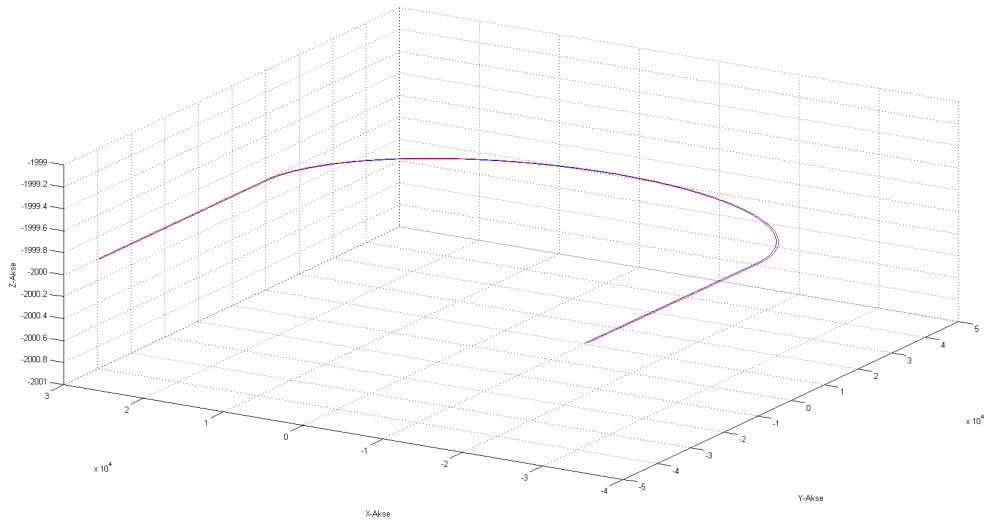
Kapittel 5

Simulator Og Resultater

5.1 Målbaner

For å teste de ulike algoritmene i et simulert miljø har det blitt generert totalt tre forskjellige målbaner. Målbanene har alle til hensikt å teste forskjellige egenskaper ved målfølgingssystemet. En kort presentasjon av hver av målbanene og hensikten med de blir derfor gitt her.

5.1.1 Målbane 1



Figur 5.1: Målbane 1

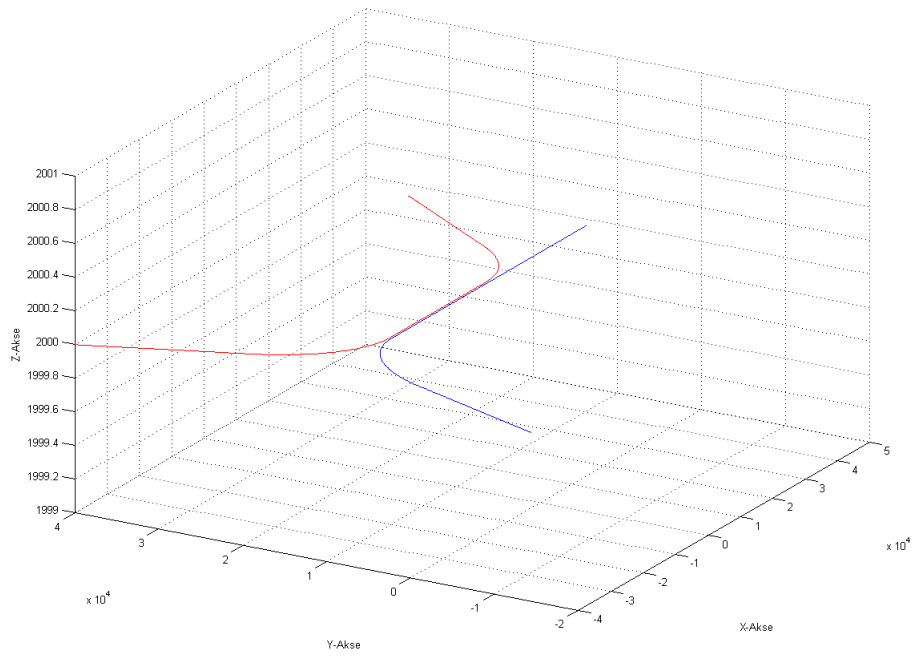
I den første målbanen har man to mål som beveger seg i en avstand av ca 200 meter mellom hverandre. De to målene starter med en rettlinjet bevegelse med en hastighet på 200 m/s. De gjennomfører deretter en manøver med radius 35 km. Denne manøveren varer til begge målene har gjennomført en 180 graders sving. Deretter fortsetter de i en rett linje.

Banen er laget slik at målene beveger seg helt i ytterkant av radar 1 sitt deteksjonsområde. Det er her radaren er mest unøyaktig og det er derfor interresant å teste effekten av algoritmene i dette området. Ved utvidelse til 4 radarer er disse plassert i et kvadrat i senter av målbanen.

	X	Y
Radar 1	0km	0km
Radar 2	10km	0km
Radar 3	0km	10km
Radar 4	10km	10km

Tabell 5.1: Radarposisjoner for målbane 1.

5.1.2 Målbane 2



Figur 5.2: Målbane 2

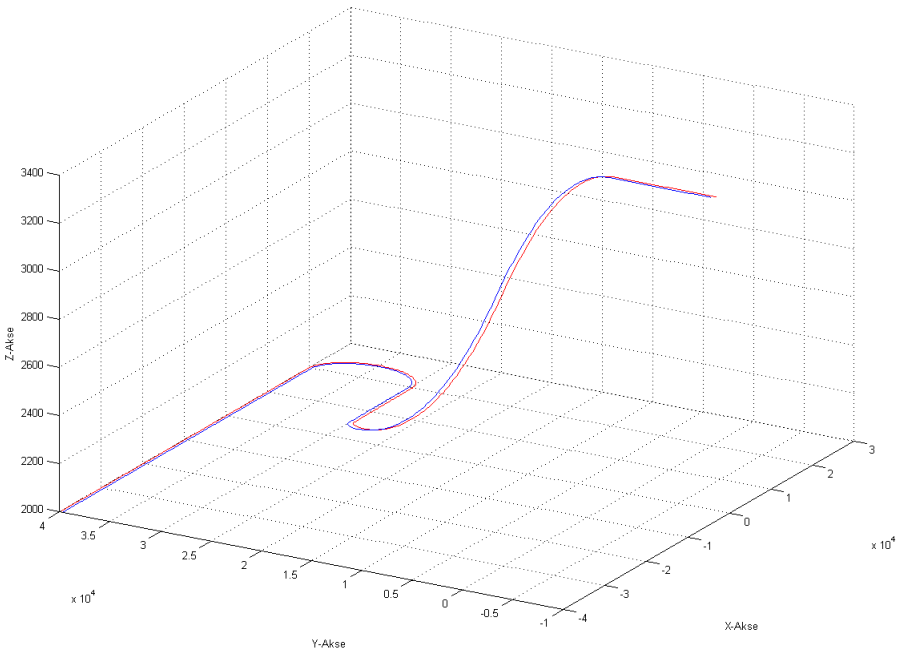
Her starter målene separert hver for seg. De flyr deretter mot hverandre i en hastighet på 200m/s. Rett før de møtes gjør begge mål en slakk manøver slik at de flyr ved siden av hverandre. Deretter gjør mål 1 en slakk manøver vekk fra mål 2. Mål 2 fortsetter rett frem. Simulatoren har her god tid til å

etablere separate mål der man har sikre data. Når de to målene nærmer seg må algoritmen detektere manøvren og passe på at den ikke forveksler de to målene. Det skal ideelt sett ikke forekomme at de to målene "bytter posisjon" med hverandre. Man får også undersøkt om algoritmen klarer å separere de to målene mot slutten av målbanen.

	X	Y
Radar 1	0km	0km
Radar 2	10km	0km
Radar 3	-10km	20km
Radar 4	10km	30km

Tabell 5.2: Radarposisjoner for målbane 2.

5.1.3 Målbane 3



Figur 5.3: Målbane 3

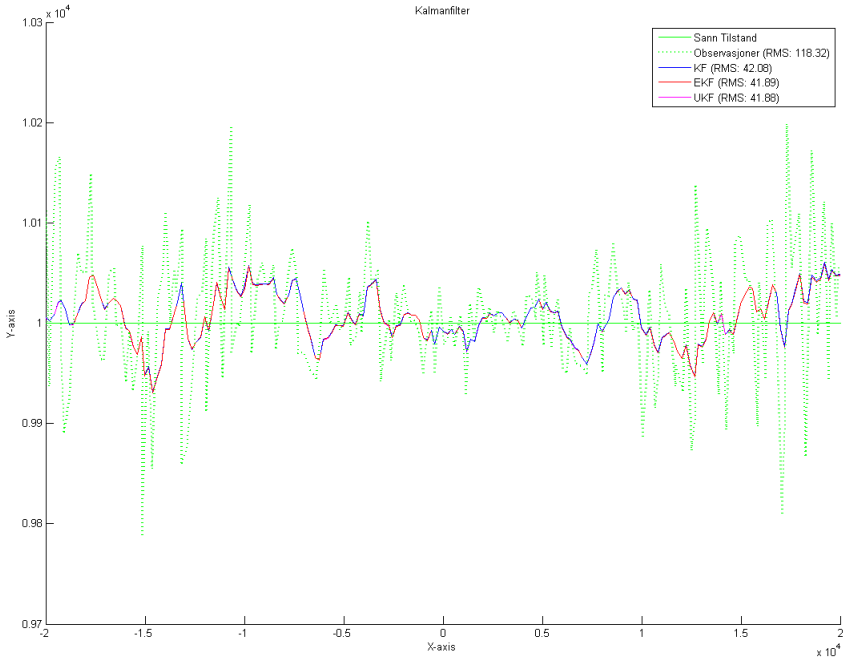
I den siste målbanen starter begge målene i tett formasjon med en hastigeht på 200m/s og en høyde på 2 km. De gjennomfører deretter en slakk usving med en radius på 5 km. Deretter akselerer de til en hastighet på 300m/s, foretar så en krap manøver med radius 1.5 km før de stiger til en ny høyde på 3.2 km samtidig som hastigheten reduseres til 100m/s. Denne målbanen sjekker om algoritmene klarer å følge og separere de to målene under varierende hastighet og krappe manøvere.

	X	Y
Radar 1	0km	0km
Radar 2	30km	0km
Radar 3	10km	30km
Radar 4	20km	15km

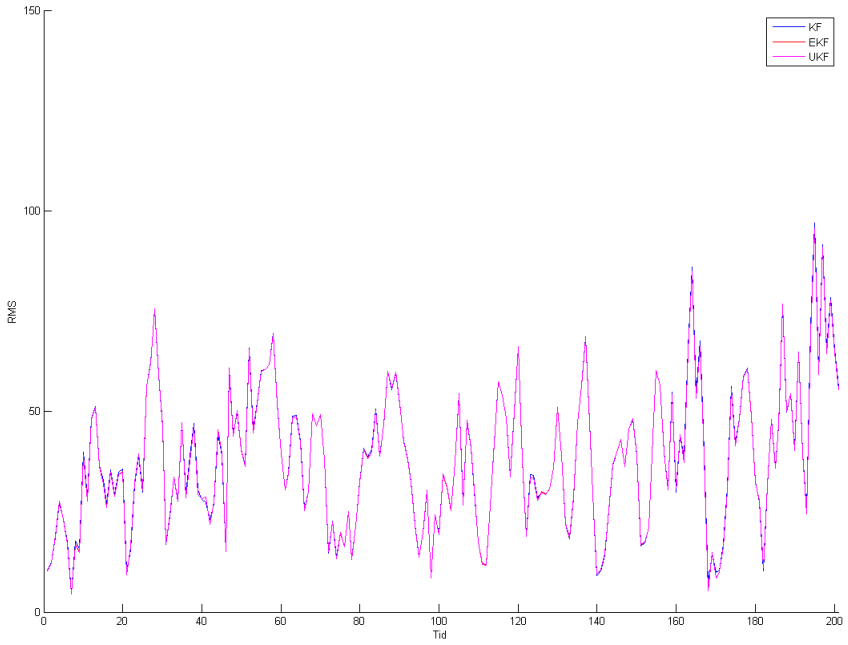
Tabell 5.3: Radarposisjoner for målbane 3.

5.2 Filtersammenligning

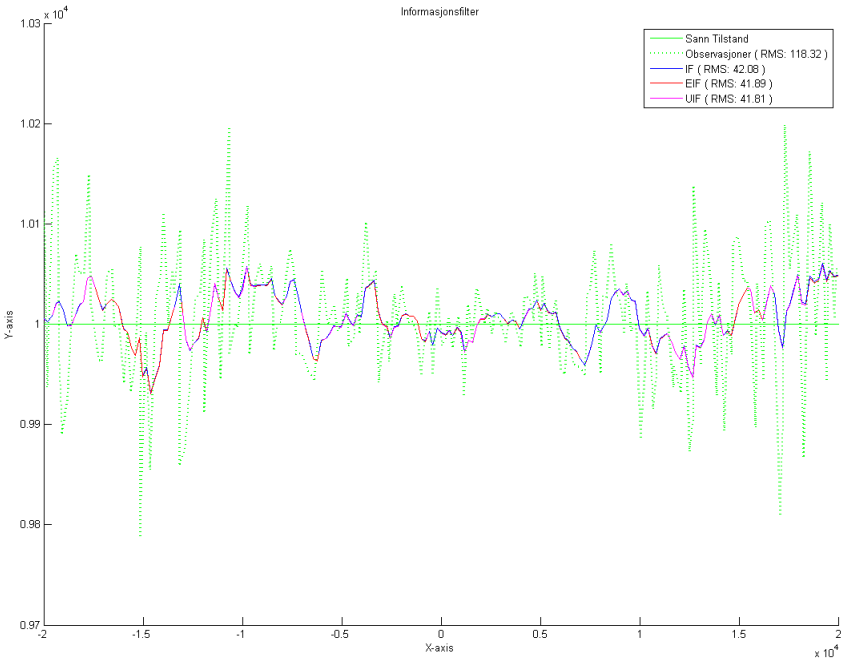
Det ble utviklet en forenklet simulator der de forskjellige målfiltreringsalgoritmene ble testet ut i samme miljø. Dette ble gjort både for å kunne finne gode verdier for prosesstøyen i kalmanfilteret og for å evaluere hvor gode de forskjellige filtrene var til å fjerne observasjonsstøyen. Det ble benyttet en rettlinjet målbane for alle de tre forskjellige filterene. RMS verdien som er gjengitt her er ved en prosesstøy $Q = diag(0, 0, 0, 1, 1, 1)$. Det implementerte KF og IF benytter forventningsrett konvertering av observasjonene slik som presentert i seksjon 2.3.2.



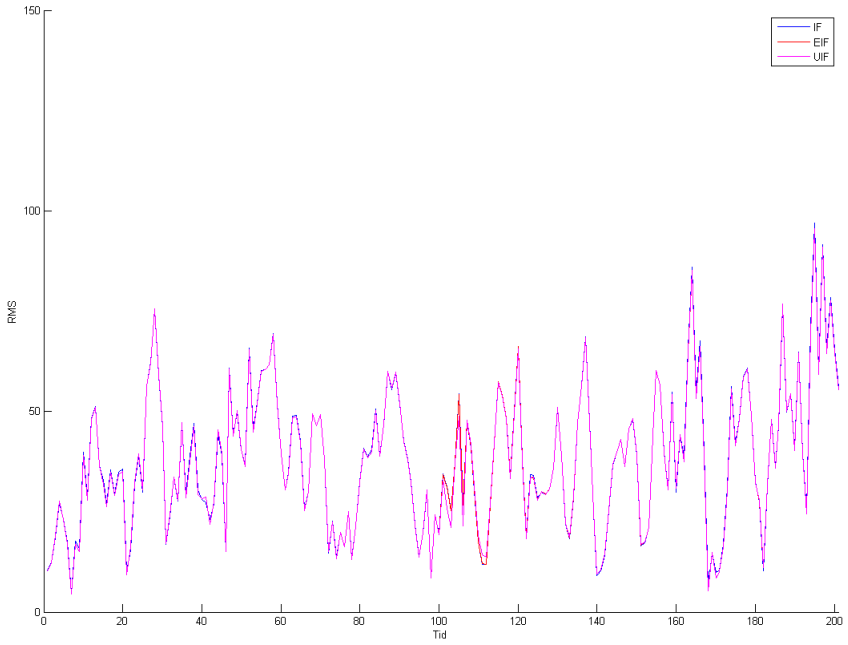
Figur 5.4: Målbane i XY-planet for KF, EKF og UKF



Figur 5.5: RMS over tid for KF, EKF, UKF



Figur 5.6: Målbane i XY-planet for IF, EIF og UIF



Figur 5.7: RMS over tid for IF, EIF og UIF

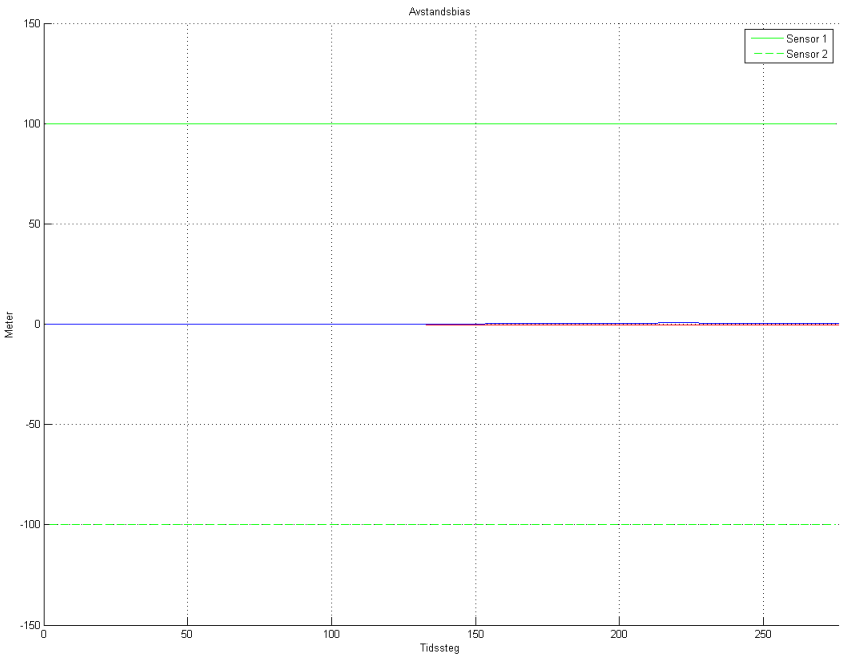
Som man kan se av de fire figurene over så gir Kalmanfilter og informasjonsfilter motsetningene akkuratt det samme resultatet. Dette var å forvente siden det teoretisk sett er snakk om samme filtertype, bare med en annen formulering. Den videre diskusjonen tar for seg Kalmanfilter, men er like gjeldende for informasjonsfilterdelen. Som man kan se gjør KF en like god jobb som EKF under estimering. Dette var vented da hele poenget med å benytte forventningsrett konvertering er å slippe å benytte EKF. KF burde derfor gi tilnærmet like godt resultat. Noe uventet var det derimot at UKF ikke hadde en mindre RMS enn EKF. Det var på forhånd forventet at UKF ville gi et bedre estimat, da hele poenget med UKF er å forbedre estimeringen til ulineære systemer. Det er ikke funnet noen god grunn til at UKF ikke levde opp til forventningene. Det er mulig at systemet ikke er ulineært nok til at UKF gir en vesentlig fordel. Dersom man ikke har noen problemer med tilgjengelig regnekraft anbefales likevel bruken av UKF. Dersom man senere skulle ha behov for implementering av andre ulineære systemer, så slipper man å måte beregne Jacobimatrisa til EKF. Ved bruk av UKF kan man bare benytte observasjonsfunksjonen direkte.

5.3 Biasestimering

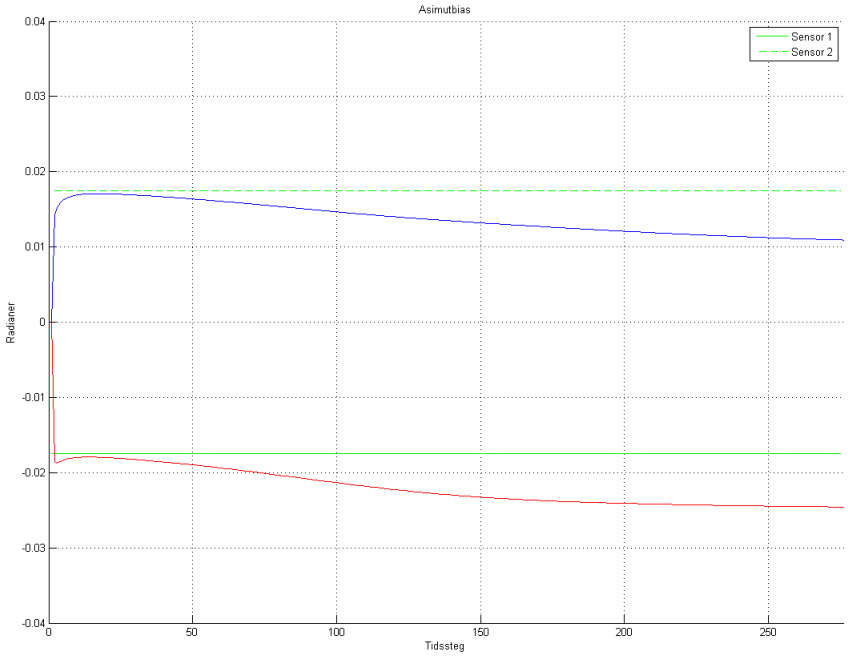
For å demonstrere biasestimeringalgoritmene ble følgende simulering konstruert. Et mål beveger seg i en rettlinjet bane med konstant hastighet. To radarer observerer målet fra hver sin posisjon. De to radarene har følgende faste biasfeil på observasjonene.

	Avstandsbias	Asimutbias	Elevasjonsbias
Radar 1	100 meter	-0,0175 radianer	0,0175 radianer
Radar 2	-100 meter	0,0175 radianer	-0,0175 radianer

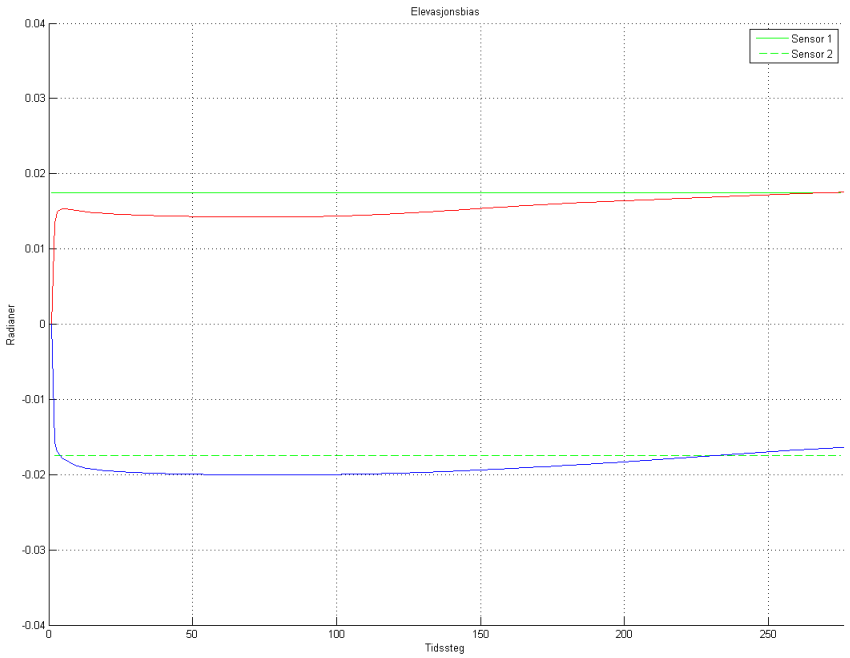
Det ble benyttet et EKF til å estimere posisjonen til målet.



Figur 5.8: Avstandsbias



Figur 5.9: Asimutbias

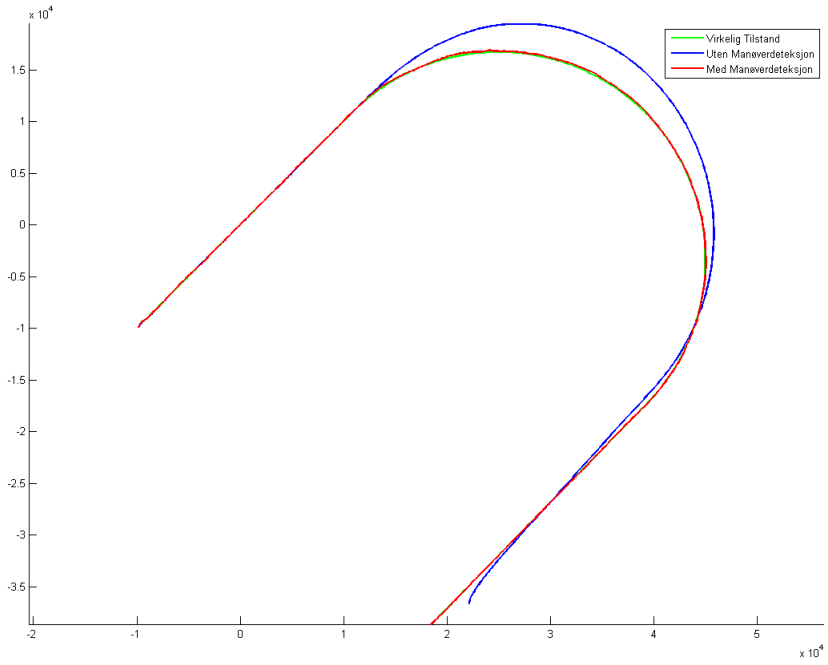


Figur 5.10: Elevasjonsbias

Som man kan se av figur 5.8, 5.9 og 5.10 så gjør det kvasirekurisve biasfilteret en god jobb med å estimere biasparameterene. Som forventet så estimerer filteret 0m i avstandsbias. Dette skyldes at man ikke direkte klarer å observere avstandsbiasen. Filteret gjør derimot en meget god jobb med å estimere asimut og elvasjonsbiasen. Estimatet for asimut har en tendens til å synke noe mot slutten av simuleringen, men absoluttverdien mellom de estimatet er den samme.

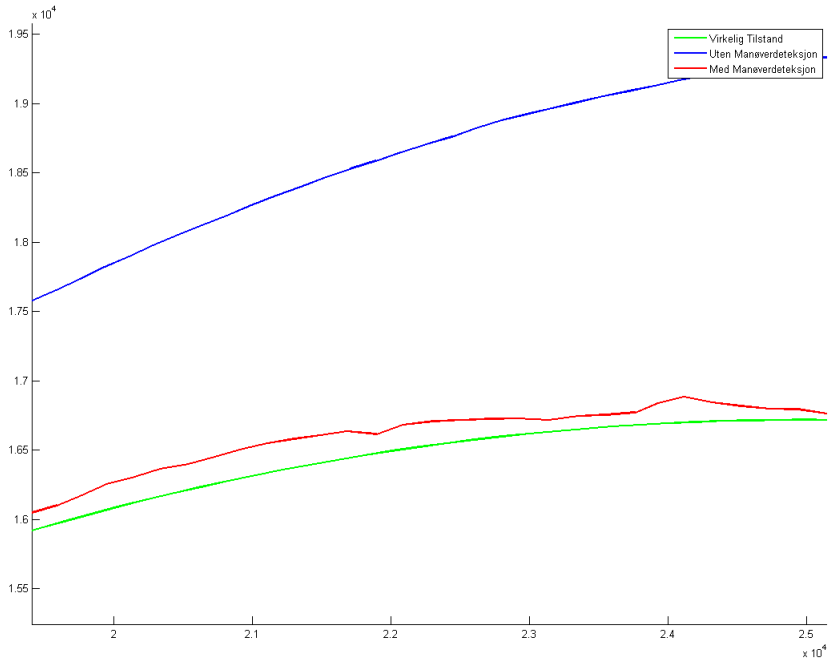
5.4 Manøverdeteksjon

Det demonstreres her hvorfor man er avhengig av å benytte en form for manøverdeteksjonm ved bruk av kalmanfilteret som målfølgingsalgoritme.



Figur 5.11: Manøverdeteksjon

I figur 5.11 vises en simulering av et mål som utfører en u-manøver. Grønn linje er her virkelig bane til målet. Den blå linja viser et EKF uten manøverdeteksjon. Man kan tydelig se at når manøveren begynner fortsetter estimatet rett frem. Etter en periode begynnner så filteret å justere for manøveren. Hadde det vært i et virkelig system der man hadde utført en form for dataassosiasjon hadde man nå mest sannsynelig mistet målet, og et nytt mål hadde måte blitt opprettet. Man kan også se at EKF aldri klarer å hente seg inn igjen etter manøveren. Den røde linjen viser derimot et EKF med manøverdeteksjon. Når en manøver detekteres øker man prosesstøymatrisa og filteret legger mer vekt på observasjonene enn modellen. Filteret klarer nå å følge målet gjennom hele manøveren.



Figur 5.12: Manøverdeteksjon Utsnitt

I figure 5.12 vises et utsnitt av målbanen ikke lenge etter at manøveren har begynt. Man kan se at det fortsatt er et avvik mellom den røde banen (som er med manøverdeteksjon) og den grønne banen (som er virkelig bane). Den økte prosesstøyen gjør derimot at filteret gjør en mye bedre jobb med å estimere korrekt bane. Forhåpentligvis er dette nok til at man ikke mister målet i en praktisk situasjon.

5.5 Dataassosiasjon

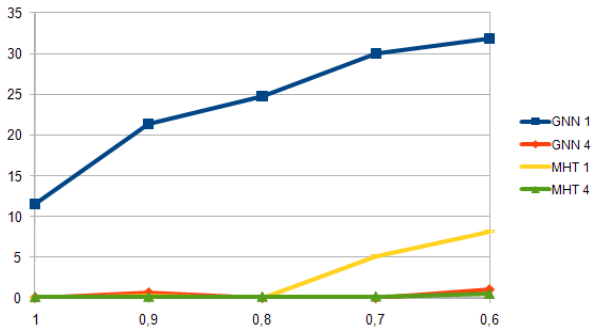
I den siste simuleringen ble GNN og MHT algoritmen testet ut på de tre målbanene presentert tidligere. For hver målbane ble det utført 10 simuleringer med forskjellige verdier for P_D og med henholdsvis 1 og 4 radarer. Det ble benyttet et EKF med manøverdeteksjon for alle simuleringene. For EKF ble det benyttet en prosesstøymatrise på $Q(k) = diag(4, 4, 4, 2, 2, 2)$, og for observasjonsstøyen ble følgende verdier benyttet:

σ_R	25m
σ_{Az}	$0,3°$
σ_{El}	$0,3°$

Tabell 5.4: Standardavvik for observasjonsstøy.

For assosiasjonsalgoritmene ble det benyttet verdier for β_{NT} og β_{FT} på henholdsvis 10^{-7} og 10^{-3}. I tillegg ble det for MHT algoritmen valgt å finne de 10 beste hypotesene for hvert scan, samt en nedre sannsynlighetsgrense på 0,05.

5.5.1 Målbane 1

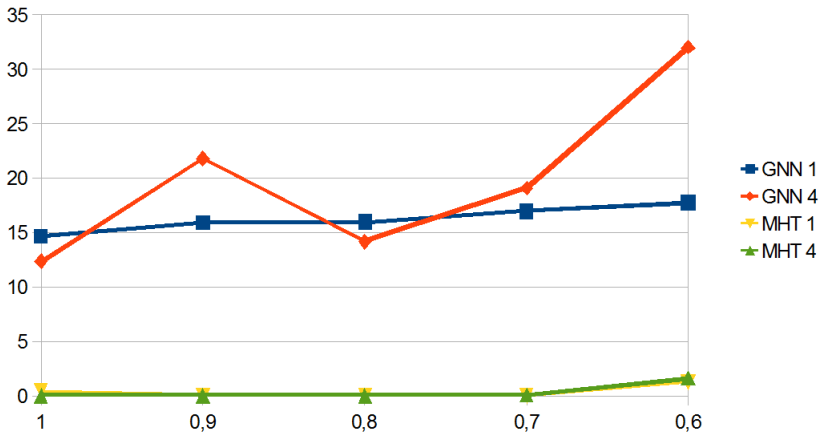


Figur 5.13: Resultater målbane 1. Y-aksen er prosent feilassosiasjoner, mens X-aksen er P_D.

Fra figure 5.13 ser man at GNN algoritmen med kun 1 radar gjør en fryktelig dårlig jobb. Algoritmen gir jevnt over ganske store feilassosiasjoner. Ved nærmere undersøkelse viste det seg at flertallet av disse feilassosisjonene befant seg i manøverpartiet av banen. For å gjøre forklaringen lettere kalles her det innerste målet for A, og det ytterste målet for B. Under manøveren vil estimatet til mål A bevege seg mot virkelig posisjon til mål B, samtidig som estimatet til B beveger seg vekk fra sin virkelige posisjon. Dette resulterer i at mål A stjeler observasjonene fra mål B. Samtidig blir det initialisert et nytt mål for den innerste banen, la oss kalle dette mål C. Mål B vil nå ikke lenger få assosiert noen observasjoner og blir slettet. Etter en liten stund får man igjen den samme situasjonen, men nå for mål B og C. Når man så benytter 4 radarer ser man en markant forbedring i feilassosiasjonene. Algoritmen klarer nå mye bedre å skille de to målene fra hverandre og man unngår at det ene målet "stjeler" observasjonene til det andre.

For MHT algoritmen ser man at denne klarer seg vesentlig bedre. Først ved en P_D på 0,7 og 0,6 får man noen feilassosiasjoner. Dette skyldes samme problematikk som for GNN algoritmen.

5.5.2 Målbane 2

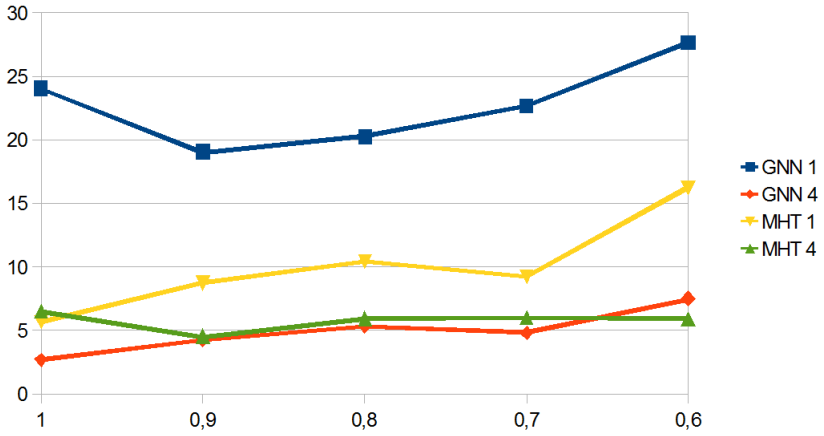


Figur 5.14: Resultater målbane 2. Y-aksen er prosent feilassosiasjoner, mens X-aksen er P_D.

Man kan her se fra figur 5.14 at GNN algoritmen er en god del dårligere enn MHT algoritmen for både 1 og 4 radarer. Den høye feilraten her skyldes derimot ikke at GNN algoritmen ikke klarer å skille de to målene fra hverandre. Problemet her oppstår i manøverpartiet rett før de to målene flyr sammen i formasjon. Modellen sier at målet skal bevege seg i en rett linje. I noen tilfeller ender man derfor opp med at GNN algoritmen tror de to målene krysser hverandre, for så å fly i formasjon et stykke.

MHT algoritmen klarer derimot utmerket å skille de to banene fra hverandre.

5.5.3 Målbane 3



Figur 5.15: Resultater målbane 3. Y-aksen er prosent feilassosiasjoner, mens X-aksen er P_D.

For den siste målbanen er resultatene noe mer utydelig. Dette er en bane som inneholder flere manøvere samt endring av hastigheten. Man kan her se paralleller til resultatet for målbane 1. GNN algoritmen yter også her dårlig, og da spesielt i manøverpartiene. Som for målbane 1 opplever man også her at det innerste målet assosieres med observasjonene til det ytterste målet. Ved bruk av 4 radarer gjør GNN algoritmen det vesentlig bedre.

Noe uventetet så gjør ikke MHT algoritmen det spesielt godt her. Verken med 1 radar eller med 4 radarer. Det er også ganske tydelig at dette er en målbane som begge algoritmene har problemer med.

5.6 Konklusjon

Det har i denne oppgaven blitt presentert flere forskjellige algoritmer benyttet innenfor målfølging. I seksjon 5.2 sammenlignet man tre ulike Kalmanfilter og informasjonsfilter. Det ble der konkludert med at alle filterene resulterte i omtrent samme RMS verdi. Dette var noe uventet da det på forhånd var forventet at UKF skulle gjøre det noe bedre. Dersom regnekraft ikke er noe problem kan derimot UKF trygt anbefales. UKF er enklere å implementere enn et EKF. Man slipper å beregne Jacobimatrisa, og kan isteden benytte målefunksjonen direkte.

I seksjon 5.3 ble det vist en enkel simulering, der man hadde to radarer med biasfeil. En Kvasirekursiv algoritme ble så benyttet for å estimere disse biasene. Det kvasirekursive filteret har den fordelen at det ikke setter noen krav til når man mottar observasjonene. Ved mottak av observasjoner summmeres disse enkelt opp i et rekursivt ligningssystem. Når man så har bruk for biasparameteren løses så dette ligningsystemet. Dette gjør at biasestimeringsalgoritmen enkelt lar seg tilpasse ulike systemer.

Tilslutt ble GNN og MHT algortimen simulert i tre forskjellige scenarier og med ulike parametere. Man kan fra dette konkludere med at MHT algoritmen er overlegen når det kommer til korrekt assosiering. Dette går derimot på bekostning av prosessorkraft. MHT algoritmen er langt mere krevende enn GNN algoritmen. Introduksjonen av flere sensorer førte også til at GNN algoritmen hadde langt færre feilobservasjoner. Dette kan antagelig knyttes til at man får et estimat som er mye nærmere virkelig tilstand til målet.

I et eventuelt videre arbeid med problemstillingene burde man benytte et IMM filter istedenfor EKF med manøverdetekesjon. IMM filteret er overlegent når det kommer til estimering der målet manøvrerer mye. Dette skulle være med på å ytterligere redusere antallet feilassosiasjoner, og da spesielt for GNN algoritmen.

Bibliografi

[1] Yaakov Bar-Shalom and William Dale Blair. *Multitarget-Multisensor Tracking: Applications and Advances*, volume Volume III. Artech House, editors edition, 2000.

[2] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation.* Wiley Interscience, 2001. Theory, Algorithms and Software.

[3] Yaakov Bar-Shalom and Xiao-Rong Li. *Multitarget-Multisensor Tracking: Principles and Techniques.* YBS, 1995.

[4] Yaakov Bar-Shalom and Edison Tse. Tracking in a cluttered environment with probabilistic data association. *Automatica 11*, pages 451 – 460, 1975.

[5] S. S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, 2004.

[6] Samuel S. Blackman. *Multiple-Target Tracking with Radar Application.* Artech House, 1986.

[7] Samuel S. Blackman and Robert Popoli. *Design and Analysis of Modern tracking Systems.* Artech House, 1999.

[8] H. A. P. Blom. An efficient filter for abruptly changing systems. In *Proc. 23rd IEEE Conf. Decision and Control*, volume 23, pages 656–658, 1984.

[9] Eli Brookner. *Tracking and Kalman Filtering Made Easy.* John Wiley and Sons, 1998.

[10] Richard R. Brooks and S.S. Iyengar. *Multi-Sensor Fusion.* Prentice Hall, 1998.

[11] David W. Casbeer. *Decentralized Estimation Using Information Consensus Filters With a Multi-Static UAV Radar Tracking System.* PhD thesis, Brigham Young University, April 2009.

[12] Kiran Chittella, Tanushree Garai, and Siddhartha Mukhopadhyay. Bias estimation of multiple radars by quasi-recursive filtering. In *Proc. Annual IEEE India Conf*, pages 1–6, 2006.

[13] Ingemar J. Cox and Sunita L Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, February 1996.

[14] Hugh Durrant-Whyte. Introduction to estimation and the kalman filter, Januar 2001. http://www.acfr.usyd.edu.au/pdfs/training/estKalFilter/EstimationNotesKC1.pdf.

[15] Atle Gjengedal. Sammenligning av ekf- og ukf- kalmanfilter anvendt på totankprosess. Master's thesis, Universitetet i Stavanger, 2009.

[16] Melita Hadzagic. Comparative analysis of the imm-jvc and the imm-jpda algorithms for multiple-target tracking. Master's thesis, McGill University, Montreal, Quebec, Canada, 2001.

[17] David L. Hall and James LLinas. *Handbook of Multisensor Data Fusion.* CRC Press, 2001.

[18] Finn Haugen. *Advanced Dynamics and Control.* TechTea, 2010.

[19] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 1987.

[20] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. pages 182–193, 1997.

[21] Jan Sigurd Karlsen. Radar målfølging. Master's thesis, NTNU, 2008.

[22] S. Kolås, B.A. Foss, and T.S. Schei. Constrained nonlinear state estimation based on the ukf approach. *Computers & Chemical Engineering*, 33(8):1386 – 1401, 2009.

[23] Tinne De Laet. Presentation on estimation, Oktober 2006. http://people.mech.kuleuven.be/ tdelaet/estimation/part3.pdf.

[24] Deok-Jin Lee. Nonlinear estimation and multiple sensor fusion using unscented information filtering. 15:861–864, 2008.

[25] X. Rong Li and Vesselin P. Jilkov. A survey of maneuvering target tracking. part iv: Decision-based methods.

[26] Martin E. Liggins, David L. Hall, and James LLinas. *Handbook of Multisensor Data Fusion*. CRC Press, 2009.

[27] Mahendra Mallick and Barbara La Scala. Comparison of single-point and two-point difference track initiation algorithms using position measurements. *Acta Automatica Sinica*, 34(3):258 – 265, March 2008.

[28] Harvey B. Mitchell. *Multi-Sensor Data Fusion*. Springer, 2007. An Introduction.

[29] Katta G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687, 1968.

[30] Umut Orguner. Target tracking, December 2010. http://www.control.isy.liu.se/student/graduate/TargetTracking/.

[31] D. Reid. An algorithm for tracking multiple targets. 24(6):843–854, 1979.

[32] Donald B. Reid. A multiple hypothesis filter for tracking multiple targets in a cluttered environment. Technical report, Lockheed Missiles & Space Company Inc., September 1977. LMSC-D560254.

[33] Jifeng Rua, Anwer Bashi, and X. Rong Li. Performance comparison of target maneuver onset detection algorithms.

[34] Elisa Shahbazian, Marc-Alain Simard, and Sylvain Bourassa. Implementation strategies for the central-level multihypothesis tracking fusion with multiple dissimilar sensors. *SPIE Signal Processing, Sensor Fusion, and Target Recognition II*, 1955:78 – 89, September 1993.

[35] P. Suchomski. Explicit expressions for debiased statistics of 3d converted measurements. 35(1):368–370, 1999.

[36] Egils Sviestins. On-line bias estimation for multisensor tracking. In Robin Evans, Lang White, Daniel McMichael, and Len Sciacca, editors, *Proceedings of Information Decision and Control 99*, pages 221–226, Adelaide, Australia, February 1999. Institute of Electrical and Electronic Engineers, Inc.

[37] Rudolph van der Merwe. *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, Oregon Health & Science University, April 2004.

[38] Wikipedia. Chain home. `http://en.wikipedia.org/wiki/Chain_Home`.

[39] Wikipedia. History of radar. `http://en.wikipedia.org/wiki/History_of_radar`.

[40] Ming Zhai and Shan Fu. Applying target maneuver onset detection algorithms to defects detection in aluminum foil.

[41] Yifeng Zhou, Henry Leung, and Keith C. C. Chang. A two-step extended kalman filter fusion approach for misaligned sensors. In *Fusion 98 International Conference*, 1998.

Tillegg A

Vedlegg

A.1 Matlabfiler

Følgende eksterne MatLab-filer er blitt benyttet i prosjektet. Det gis her en kort presentasjon av hva de gjør, og hvor de er funnet.

Murty's Algoritme
Dette er en implementasjon av Murty's algoritme for å finne de M-beste assosiasjonene. Denne er beregnet brukt sammen med auksjonsalgoritmen.
`http://www.control.isy.liu.se/student/graduate/TargetTracking/Murty.m`

Auksjonsalgoritmen
Dette er en implementasjon av auksjonsalgoritmen for å finne den beste assosiasjonen, gitt en assosiasjonsmatrise.
`http://www.control.isy.liu.se/student/graduate/TargetTracking/Auction.m`

Progressbar
Progressbar hjelper deg med å vise hvor langt et program har kommet under kjøring. Veldig nyttig dersom man har programmer som bruker lang tid på å kjøre, og man har lyst på en visuel indikator som indikerer hvor lenge programmet har igjen.
`http://www.mathworks.com/matlabcentral/fileexchange/6922-progressbar`

Circle
Circle er et skript til MatLab som forenkler tegningen av sirkler i grafer.
`http://www.mathworks.com/matlabcentral/fileexchange/2876`

A.2 Simulator

Av hensyn til miljøet ble det valgt å ikke legge med Matlab simulatoren som vedlegg. Filene til simulatoren kan isteden bli funnet i mappen *Simulator* på den vedlagte cd'en.