Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Automatisering og signalbehandling	Vårsemesteret, 2015 Åpen / ~~Konfidensiell~~
Forfatter: Geir Risvoll	..... (signatur forfatter)
Fagansvarlig: Karl Skretting Veileder(e): Håvard Gullbekk	
Tittel på masteroppgaven: Tilstandsovervåking av UAV (Unmanned Aerial Vehicle) Engelsk tittel:	
Studiepoeng: 30	
Emneord: UAV Tilstandsovervåking Klassifisering	Sidetall: 57 + vedlegg/annet: 12 sider vedlegg Stavanger, 15.06/2015 dato/år

Tilstandsovervåking av UAV (Unmanned Aerial Vehicle)

Masteroppgave innen
automatisering og signalbehandling

Våren 2015

Geir Risvoll

Universitetet i Stavanger
Institutt for data- og elektroteknikk
Det teknisk-naturvitenskapelige fakultet

Sammendrag

I denne oppgaven er muligheten for å automatisk detektere og identifisere feil som kan oppstå på en Unmanned Aerial Vehicle (UAV) undersøkt. Rapporten presenterer først en del teori om teknikker som blir brukt i rapporten før anvendelsene av disse teknikkene presenteres.

Det er blitt tatt opp to datasett fra UAV-en. Ett der det ikke var feil på UAV-en og ett der det var montert forhøyinger på propellene til UAV-en for å simulere is. Det er disse to datasettene som er brukt for å lage modeller av UAV-en.

Ettersom datasettet med normal drift ble tilgjengelig en stund før det andre, ble det først tatt utgangspunkt i datasettet med normal drift og noen antagelser med hensyn på hvordan is på propellene ville påvirke farkosten. Dette ble forsøkt simulert ved å modifisere de dataene som var blitt tatt opp ved normal drift. Det ble så utviklet en metode for å skille disse to datasettene fra hverandre.

Da datasettet med forhøyninger ble tilgjengelig, ble denne metoden forsøkt utprøvd på det nye datasettet, men det viste seg da å gi dårlige resultat. Det ble derfor konkludert med at antagelsene som ble tatt ikke var realistiske. Grunnen til dette kan være at UAV-en har en autopilot som regulerer systemet, og oppførselen til UAV-en med forhøyninger på propellene ble derfor tilnærmet lik som ved vanlig drift.

Ettersom den initielle tilnærmingen ikke ga gode nok resultater ble andre metoder for tilstandsovervåking forsøkt. Det ble utviklet flere metoder som alle ga brukbare resultater.

Metodene som baserer seg på strømforbruket til UAV-en fungerer noe tregere enn de som ikke gjør det. Dette er fordi loggefrekvensen til målt strømforbruk setter en begrensing da denne bare er 1 Hz.

Den metoden som har gitt best resultater i denne oppgaven baserer seg på å estimere akselerasjonen til UAV-en basert på motorpådraget til de fire motorene. Denne metoden klarer å identifisere feiltilstanden på UAV-en seks sekunder etter start. Metoden som baserte seg på å bare bruke strømforbruket til UAV-en brukte 21 sekunder.

Innhold

Sammendrag **I**

Innhold **II**

Figurer **IV**

Forkortelser **V**

1 Innledning **1**

1.1 Oppgaven 1
1.2 Hvordan arbeidet utføres 3
1.3 Tidligere arbeid 5
1.4 Rapportens innhold 6

2 Teori **7**

2.1 Hva er tilstandsovervåking? 7
2.2 Hvorfor er tilstandsovervåking ønskelig? 8
2.3 Hva blir tilstandsovervåking brukt til? 8
2.3.1 Diagnose 8
2.3.2 Prognose 9
2.4 Hvor blir tilstandsovervåking brukt? 9
2.5 Modellering 10
2.5.1 Fra fysisk system 12
2.5.2 Fra eksperiment 15
2.5.3 Eksempel på kurvetilpassing 16
2.5.4 Eksempel på ARMAX 18
2.5.5 Eksempel på ulineær ARX 22
2.6 Klassifisering 24
2.6.1 Terskelverdi 24
2.6.2 Statistikk 24
2.6.3 Maximum Likelihood 25
2.6.4 Support Vector Machines 26
2.7 Kalman 34

3 Eksperiment med ett datasett **36**

3.1 Matlab 36
3.2 Estimering av dynamikk 37

4 Eksperiment med to datasett **39**
4.1 Estimering av dynamikk 39
4.2 Strømforbruk 40
4.3 Strømforbruk fra motorpådrag 43
4.4 Strømforbruk vs. akselerasjon 46
4.5 Estimert akselerasjon 49

5 Videre arbeid **52**

6 Konklusjon **54**

Referanser **56**

A inspect.m **58**

B SVM.m **59**

C IDstep.m **60**

D Kalmanfilter.m **61**

E Velocity.m **62**

F Current.m **64**

G Motor.m **65**

H Acceleration.m **66**

I NLARX.m **68**

Figurer

1 UAV brukt i oppgaven. 3
2 Ardupilot brukt i denne oppgaven. [1] 4
3 Oversikt over tilgjengelige modeller som kan lastes inn på ArduPilot [1]. . 4
4 Generell systemidentifikasjon. 12
5 En UAV-modell brukt for utledning av formler [2]. 14
6 Kurvetilpassing fra data. 18
7 ARX estimering av step-respons. 22
8 Klassifisering ved bruk av maximum likelihood. 26
9 SVM klassifisering av lineært separabel data 27
10 Ulineært separabel data. 30
11 Ikke separabel data. 32
12 Målt og modifisert hastighet i X-retning. 37
13 Identifikasjon av step på målt data. 38
14 Klassifisering med forskjellige ARX modeller. 38
15 Identifikasjon av step på data med feil på UAV-en 39
16 Strømforbruk til UAV. 40
17 Strømforbruk til UAV kjørt gjennom to forskjellige lavpassfilter. 41
18 Strømforbruk til UAV kjørt gjennom et Kalman-filter. 42
19 Klassifisering basert på strømforbruk 43
20 Forhold mellom motorpådrag og strømforbruk 44
21 Et plott av strømforbruk gitt av motorpådrag med kurve. 45
22 Avvik i strømforbruk. 46
23 Strømforbruk gitt av akselerasjon i Z-retning. 47
24 Klassifisering på grunnlag av strømforbruk og akselerasjon i Z-retning. . . 48
25 Klassifiserte punkter fra strømforbruk og akselerasjon. 49
26 Målt og estimert akselerasjon. 50
27 Residual mellom målt og estimert akselerasjon. 50
28 Filtrert residual. 51

Forkortelser

APM	ArduPilot Mega
ARMAX	AutoRegressive Moving Average eXogenous
CBM	Condition-Based Maintenance
EKF	Extended Kalman Filter
FDI	Fault Detection and Isolation
FFI	Forsvarets ForskningsInstitutt
GPS	Global Positioning System
GUI	Graphical User Interface
MATLAB	Matrix Laboratory
ML	Maximum Likelihood
NARX	Nonlinear AutoRegressive eXogeneous
PWM	Pulse Width Modulation
RPM	Rounds Per Minute
SHM	System Health Management
SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle

1 Innledning

I denne oppgaven skal det utvikles metoder for tilstandsovervåking på en UAV (Unmanned Aerial Vehicle). Dette gjøres ved å lage forskjellige modeller som kan si noe om forventet oppførsel fra UAV-en. En kan så sammenligne den forventede oppførselen med den målte oppførselen og avgjøre om det er noe feil eller ikke. UAV-en er utstyrt med en autopilot, som betyr at den korrigerer for feil i oppførselen automatisk. Dette betyr igjen at det er krevende å finne modeller som kan brukes.

Dette er et arbeid som blir utført i samarbeid med Forsvarets ForskningsInstitutt (FFI) og Universitetet i Stavanger (UiS). Arbeidet er utført som en masteroppgave innen signalbehandling/kybernetikk ved UiS.

Motivasjonen for å utføre dette arbeidet er å gjøre det enklere for operatøren å styre UAV-en, uten å måtte analysere alle signalene som er tilgjengelig. Tanken er at operatøren får presentert enten et grønt lys dersom alt er normalt og et gult eller rødt lys dersom noe er galt. Bakgrunnen for oppgaven samt hvordan den tenkes utført skal bli kort forklart her.

1.1 Oppgaven

Oppgaveteksten som gitt av FFI er som følger:

> "Tilstandsovervåking for UAVer"
>
> En UAV er et system av systemer der mange typer feil kan oppstå. Noen feil fører umiddelbart til uhell, andre fører bare til redusert farkostytelse, mens andre feil lenge kan forbli "usynlige" før de etter noe tid fører til uhell. Enkelte feil er også ufarlige med mindre de oppstår samtidig med andre feil.
>
> Tradisjonelt blir en mengde målte sensordata sendt fra UAVen og ned til en bakkestasjon hvor en operatør følger med og sjekker at farkosten oppfører seg som forventet. Operatøren er også trent opp til å utføre bestemte prosedyrer når en gitt feil oppstår.
>
> Operatørens rolle i denne sammenhengen kan automatiseres. Dette vil bidra til redusert personellbehov og også sikrere operasjoner, spesielt i stressede situasjoner eller når oppdragene blir langvarige.
>
> Oppgaven går ut på å benytte modellbasert tilstandsestimering for å:
>
> - Overvåke om farkosten gjennomfører oppdraget slik det er planlagt. Det

vil i praksis si å sammenligne posisjon, hastighet/kurs med en missionplan under hensyntagen til vind, navigasjonsnøyaktighet og individuelle variasjoner i farkostytelse.

- Detektere og identifisere mer detaljerte feiltilstander i farkosten. Dette kan f.eks. være ising som påvirker aerodynamisk ytelse på skrog/vinger og propell, feil på aktuatorer/rorflater/propell, feil på motorer (økt strømforbruk), sensorfeil (f.eks. pitotrør) osv.

Oppgaven bør først ha en teoretisk del der en betrakter problemet generisk. Det kan så være aktuelt å modellere en bestemt type farkost og kjøre simuleringer på denne. Vi kan også generere reelle datasett (ved å fly en UAV) og introdusere "feil" som så kan forsøkes identifisert ved hjelp av tilstandsestimatoren.

I denne oppgaven fokuseres det på punktet om å detektere og identifisere feiltilstander i farkosten.

Formålet med oppgaven er å utvikle en tilstandsovervåking for bruk på en UAV. Dette blir gjort for å avlaste operatøren som styrer UAV-en. I dag er det slik at operatøren må følge med på alle tilstander til UAV-en og må så anslå om det er noe feil. Dette må utføres samtidig som et oppdrag skal bli gjennomført. For at operatøren skal kunne ha fullt fokus på oppdraget, vil det være nyttig dersom UAV-en selv sier ifra om det er noe feil.

Første oppgave er å kunne detektere om noe er feil, uten å si noe om hva som er feil. Senere vil det være ønskelig å kunne si noe om hvilke feil som kan identifiseres ut fra de måledata som er tilgjengelig og også implementere en feil-klassifiseringfunksjon.

I framtiden kan det være interessant å kunne la UAV-en ta avgjørelser basert på tilstandsovervåkingen, slik at en operatør har mulighet til å styre flere UAV-er samtidig. UAV-en vil da være nødt til å avgjøre om den er i stand til å utføre oppdraget, om den skal returnere hjem eller om den skal utføre oppdraget med redusert ytelse.

En av utfordringene ved denne oppgaven er at den skal basere seg kun på de signalene som allerede er tilgjengelige på UAV-en. Det skal med andre ord ikke monteres ekstra sensorer som bare skal brukes for tilstandsovervåking.

1.2 Hvordan arbeidet utføres

Arbeidet begynner med å utforske allerede eksisterende systemer, presentere disse og avgjøre om de er overførbare til denne oppgaven. Det skal prøves å skaffe en oversikt over hva som er gjort innen denne type oppgave og hvilke metoder som er blitt brukt i disse tilfellene.

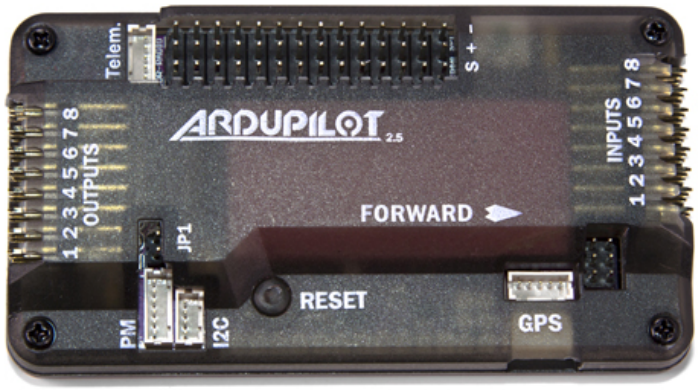
Det skal videre forsøkes å utvikles forskjellige modeller av UAV-en fra data som er tatt opp ved flyging. Disse modellene skal så brukes for å kunne si noe om hvilke verdier som er forventet fra UAV-en ved senere flyginger.

Ved flyging skal data fra modellene sammenlignes med faktisk måledata. For å avgjøre om det er feil skal forskjellen mellom estimert og målt signal bli undersøkt. Dersom forskjellen blir for stor tyder det på at UAV-en opererer utenfor det området som modellen er laget for og dette tyder på at noe er feil.

Flyging blir utført av FFI på en UAV som de har til disposisjon der. Denne er vist i fig. 1.

Figur 1: UAV brukt i denne oppgaven. Vekt er omtrent 2.5 kg og avstand fra motor til motor er omtrent 77 cm. Denne er utstyrt med en autopilot av typen ArduPilot. Flytid er estimert til 30 minutter.

UAV-en er utstyrt med en ArduPilot[1] av typen APM 2.6, som er en Arduino-basert[3] mikrokontroller som gjør quadcopteret i stand til å fly mer stabilt, eller til å fly seg selv basert på GPS-koordinater. Et bilde av en ArduPilot er vist i fig. 2.



Figur 2: Ardupilot brukt i denne oppgaven. [1]

ArduPilot er utstyrt med akselerometer, gyroskop og barometer. Det er i tillegg mulig å utstyre det med GPS og kompass for å tillate full automasjon av farkosten. En fordel med ArduPilot er at det er mulig å bruke den sammen med mange forskjellige farkoster. Den kan bli brukt til å styre alt fra et ubemannet kjørende fartøy (UGV) til et vanlig modellfly og forskjellige typer helikopter. De forskjellige mulighetene er vist i fig. 3. I denne oppgaven er det brukt ArduCopter V3.2 Quad.

Figur 3: Oversikt over tilgjengelige modeller som kan lastes inn på ArduPilot [1].

1.3 Tidligere arbeid

I mye av litteraturen baserer man seg på å bruke dedikerte sensorer slik som akselerometer og strekklapper for å overvåke spesielt utsatte komponenter. Dette er framgangsmåter som er godt dokumentert i blant annet [4] og [5]. Hovedforskjellen fra eksisterende litteratur er at det skal tas utgangspunkt i å bruke bare de tilgjengelige sensorene som eksisterer på UAV-en fra før for å drive feilsøking.

En artikkel som utforsker muligheten for å bruke eksisterende sensorer for feildeteksjon er [6]. Her forsøkes det å detektere akturator- og sensor-feil på et radiostyrt helikopter. Det blir her satt en del restriksjoner på bevegelsene til helikopteret og det blir utviklet en lineær modell som fungerer innenfor disse restriksjonene. Residualen mellom estimert og målt verdi blir så sammenlignet med en terskelverdi og dersom residualen kommer over terskelverdien tolkes det som en feil.

Lignende framgangsmåter er også dokumentert i [7] og [8]. Det blir her utviklet matematiske modeller for de ulike UAV-ene og residualer blir beregnet basert på estimert og målt data.

En av hovedforskjellene mellom disse og denne oppgaven er bruken av en autopilot som korrigerer for feil automatisk. Dette forverrer feildeteksjonsproblemet ved at det ikke vil være et statisk avvik selv om det blir en feil på UAV-en. Hvorvidt det vil være mulig å detektere dynamiske forskjeller vil være avhengig av autopiloten og samplefrekvensen til signalene.

1.4 Rapportens innhold

Oppgaven er delt opp i flere deler slik at en leser som er kjent med fagområdet kan hoppe over enkelte deler og allikevel forstå hva som er gjort og hvordan det er gjort.

Seksjon 2 gir en introduksjon i teorien som ligger til grunn for oppgaven. Det blir gått inn på generelt om tilstandsovervåking og hvor og hvorfor det er ønskelig å bruke tilstandsovervåking. Teorien bak metoder som er brukt i oppgaven presenteres også her og det gis enkle eksempel på implementasjon av disse.

Seksjon 3 presenterer modellering som ble gjort på et enkelt datasett. Det ble her tatt antagelser om hvordan feil ville påvirke UAV-en. Et andre datasett ble dermed laget basert på det første i kombinasjon med de antagelsene som ble tatt. En metode for å separere disse dataene fra hverandre blir så introdusert.

Seksjon 4 bruker i tillegg datasettet som ble tatt opp når det var montert en feil på UAV-en. Metoden som ble utviklet i seksjon 3 blir testet ut på det nye datasettet og nye metoder blir utviklet basert på de to datasettene.

Seksjon 5 tar for seg muligheter for å utvikle arbeidet i denne rapporten videre og hvordan det vil være naturlig å fortsette.

Seksjon 6 gir en oppsummering av resultatene og sammenligner disse med oppgavebeskrivelsen for å se hvor godt disse passer overens.

2 Teori

Tilstandsovervåking er på ingen måte et nytt fagområde. Et tidlig eksempel på dette er kanarifuglen som ble brukt i gruvedrift. Dersom fuglen døde, betydde det at luften i gruven var så dårlig at det var på tide å komme seg ut. På denne måten ble tilstanden til gruven overvåket. Det har selvsagt skjedd store forandringer fram til i dag, blant annet er bruken av kanarifugler kraftig redusert, og takket være datamaskiner er det i dag mulig å detektere feil mye tidligere og mer nøyaktig enn noen gang tidligere.

I dette kapitelet skal det introduseres litt teori rundt generell tilstandsovervåking og også noen viktige metoder som blir brukt innen tilstandsovervåking som vil bli brukt i denne oppgaven.

2.1 Hva er tilstandsovervåking?

I [5] brukes begrepet Condition Based Maintenance (CBM), eller tilstandsbasert vedlikehold. Definisjonen på dette blir beskrevet som bruken av sanntidsdata for å avgjøre tilstanden til maskinen, som igjen kan bli brukt til å planlegge reparasjon og vedlikehold. I denne oppgaven tas det utgangspunkt i dette og definerer tilstandsovervåking som bruken av sanntidsdata for å detektere feil på farkosten. Dette betyr at all relevant data som produseres av farkosten skal kunne analyseres umiddelbart og brukes til å avgjøre om farkosten flyr som normalt eller ikke.

En noe bredere definisjon brukes i [9], side tre, der det tas utgangspunkt i System Health Management(SHM). Dette begrepet defineres som

> ...the capabilities of a system that preserve the system's ability to function as intended.

Som denne definisjonen viser trenger ikke tilstandsovervåking bare å være en passiv prosess, men den kan også bli gjort om til en aktiv prosess. Et godt eksempel på en aktiv prosess er automatiske bremsesystem som er blitt implementert i mange moderne biler. Dette er et system som detekterer en uønsket tilstand, en kommende kollisjon, og som dermed påvirker systemet, bilen, til å unngå denne tilstanden. Dette er et system som kommer godt inn under SHM-definisjonen.

2.2 Hvorfor er tilstandsovervåking ønskelig?

Tilstandsovervåking kan være ønskelig for en rekke forskjellige grunner. En av de vanligste grunnene er for å kunne planlegge reparasjoner og vedlikehold av utstyr. Dette kan føre til betydelig besparelser ved å begrense uønskede reparasjoner og øke oppetiden til prosessen. Et eksempel på dette er gitt i [5], side 47-49, der man ser på en flåte av lastebiler. Ved å innføre tilstandsovervåking av dekkene er det mulig å minimere kostnaden ved dekkslitasje.

En annen grunn til å bruke tilstandsovervåking er at man slipper mye unødvendig vedlikehold. I motsetning til klassisk vedlikehold der man har X antall timer mellom hver gang vedlikehold blir utført, kan man ved tilstandsovervåking vente til det faktisk er behov for vedlikehold. Det er også tidsbesparende i forhold til reparatøren som slipper å gå over hele enheten for å lete etter en mulig feil. Tilstandsovervåking forteller om det er feil og i beste tilfelle forteller den også hvor feilen er.

Tilstandsovervåking kan også bli brukt for å detektere kommende feil og automatisk foreta handlinger for å unngå eller håndtere disse feilene. Et eksempel på slik bruk er gitt i [10]. Her utforskes muligheten for å håndtere en feil på en av de fire motorene som er montert på en quadrotor. Når en feil detekteres på en av motorene omdefineres quadrotoren til en trirotor og reguleres deretter. På denne måten er det mulig å fortsette å fly, selv med en skadet motor/propell.

2.3 Hva blir tilstandsovervåking brukt til?

Tilstandsovervåking kan bli brukt i mange ulike situasjoner og områder, med industriroboter, fly og romferjer som noen eksempler. I litteraturen finner man eksempel på at tilstandsovervåking er med på å øke lønnsomhet, sikkerhet og prestasjon i utallige applikasjoner [9][11].

Det er i hovedsak to områder innenfor tilstandsovervåking [5], dette er diagnose og prognose. Hovedideen bak disse blir presentert her.

2.3.1 Diagnose

Diagnose er den delen av tilstandsovervåking som detekterer og identifiserer en feil som er oppstått. Formålet her er å vite hva som er galt etter at noe er gått galt, eller i beste fall i det noe går galt. Denne delen av tilstandsovervåking gjør det mulig å si om et oppdrag er mulig å gjennomføre eller om tiltak er nødt til å bli utført for å ikke miste farkosten.

Dette kan blant annet innebære å gjennomføre resten av oppdraget med redusert ytelse eller å avbryte oppdraget totalt. I [4], side 31, illustreres diagnostisering. Det kommer fram til at dette består av noen konkrete steg. Dette er:

- Prosess-modell: En modell av prosessen som brukes til å estimere tilstander til farkosten, slik som fart, posisjon osv.
- Egenskapsgenerering: Et utdrag av tilstandene som kan brukes til diagnose.
- Endringsdeteksjon: Måling av forskjell mellom estimert og faktiske tilstander.
- Feil-diagnosering: Klassifisering av tilstandene.

Det er diagnosedelen av tilstandsovervåking som skal bli utforsket i denne oppgaven.

2.3.2 Prognose

Prognose tar diagnosedelen ett skritt videre og sier noe om hvordan tilstanden til farkosten vil være lengre frem i tid, og hvor lenge det er til vedlikehold er nødvendig. En åpenbar utfordring ved prognose er at det er umulig å se inn i fremtiden med 100 prosent sikkerhet. Denne delen av tilstandsovervåking er derfor avhengig av å ha gode matematiske modeller av de komponentene som overvåkes. Fordelen ved bruk av prognose er at behovet for vedlikehold vil være ulik for de forskjellige farkostene som er i bruk avhengig av hvilke påkjenninger de utsettes for. Dermed er det ikke nødvendig å ta alle farkostene inn til vedlikehold ved jevne mellomrom. Dette er noe som fører til mer oppetid for farkostene og er både tids- og kostnads-besparende. En enkel måte å predikere når en komponent vil feile er beskrevet i [11], side 7-8. Dette er ved å bruke simulerte kanarifugler. Disse fungerer på samme måte som i gruvene, men istedenfor virkelige fugler er det deler som er designet for å tåle mindre enn de delene som er nødvendig for å opprettholde funksjon. Disse 'kanarifuglene' kan bli kalibrert slik at når de blir ødelagt, så vet man omtrent hvor lenge det er til hovedkomponenten blir ødelagt.

2.4 Hvor blir tilstandsovervåking brukt?

Siden 1970-tallet har bruken av tilstandsovervåking økt drastisk [5]. Dette er på grunn av introduksjonen av datamaskiner for å utføre kalkulasjoner. Siden den gang har det blitt mer og mer automasjon og dermed også mer tilstandsovervåking av alt fra fly til industrianlegg. I litteraturen er det ofte uttrykket "Fault Detection and Isolation" (FDI) som blir brukt.

Det er mulig å finne mange forskjellige applikasjoner av tilstandsovervåking. I [9] nevnes det blant annet

- Overvåking av flymotorer
- Overvåking av elektriske system
- Overvåking av livsviktige systemer på romferjer.

I [4] vises det til applikasjoner som

- Feildeteksjon på industriroboter
- Feildeteksjon på pumper
- Feildeteksjon på varmeveklsere

Andre applikasjoner er blant annet

- Feildeteksjon på sensorer og akturatorer i et vannrenseri [12].
- Feildeteksjon på dampgenerator i et kjernekraftverk [13].
- Feildeteksjon på en fermenteringsprosess [14].

Det er tydelig at tilstandsovervåking er et svært utbredt område med mange applikasjonsmuligheter.

2.5 Modellering

Når man ønsker å bruke en eller annen form for feildeteksjon, er det ofte nyttig å ha en matematisk modell av den prosessen som skal overvåkes. På denne måten blir det mulig å gi en prediksjon på hvordan systemets tilstands burde være. Dette kombineres videre med målinger av hvordan tilstanden virkelig er. Forskjellen mellom estimert og målt tilstand blir brukt til å generere en residual, og ved å se på størrelsen til denne residualen er det mulig å avgjøre om tilstanden til systemet er normal eller ikke. Det er en slik framgangsmåte som er blitt benyttet i blant annet [6], [7] og [8].

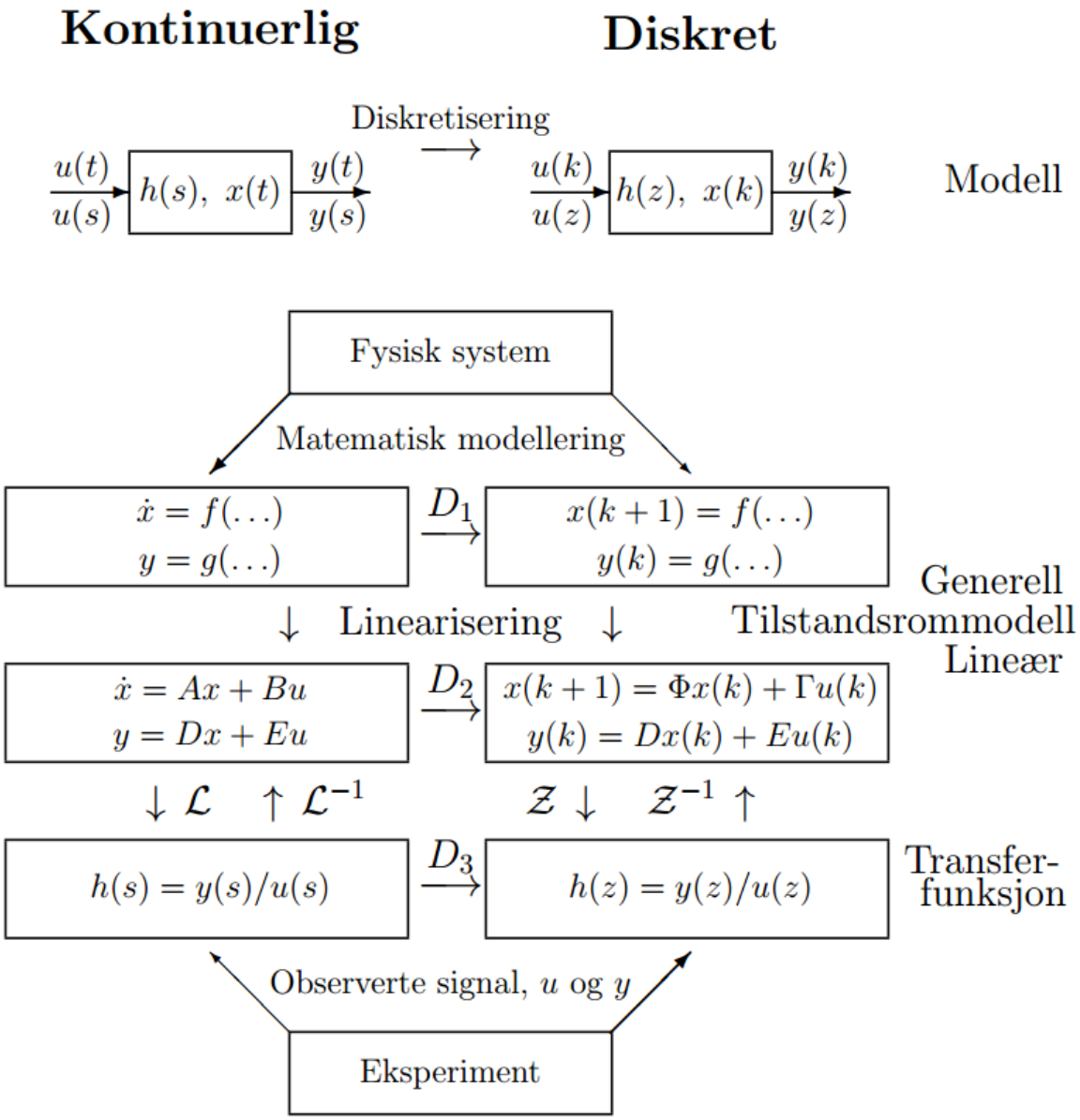
Det eksisterer flere metoder for å utvikle en matematisk modell av et system og en skisse er vist i fig. 4. Det er her vist at det er to hovedideer som eksisterer for å utvikle en matematisk modell, fra fysikk og fra eksperiment. Uavhengig av hvilke vei man velger å gå er det også mulig å komme til de andre formene ved hjelp av forskjellige transformasjoner.

Det er ikke mulig å si at en metode er bedre enn en annen, men de introduserer forskjellige unøyaktigheter ved bruk.

Det å modellere fra fysikk er ofte en del mer matematisk krevende ettersom man er nødt til å være kjent med en del fysiske begreper og formler. En modell utledet fra fysikk vil aldri være 100 prosent korrekt ettersom man alltid er nødt til å ta antagelser som gjør matematikken enklere, men dersom antagelsene man tar er akseptable innenfor det område modellen skal bli brukt, er det ofte godt nok. Fordelen med å bruke fysikk er at man vil kunne si noe om systemet uten å måtte gjøre eksperiment, noe som kan være dyrt og tidskrevende.

Dersom man velger å modellere basert på et eksperiment, er den første utfordringen å avgjøre hvordan modellen skal se ut. Ofte antas det at modellen er lineær, ihvertfall i det området der en skal se på systemet. Dersom man skal se på et stort område av systemet, må man kanskje velge å bruke en ulineær modell. Å velge feil modelltype kan gi store feil i de resulterende ligningene. Når modellen er valgt, er det ofte enklere å finne parametrene i denne enn å regne seg fram fra fysikk. Det eksisterer også mange verktøy for å estimere disse parametrene automatisk. Matlabs System Identification Toolbox er et eksempel på et slikt verktøy.

En god framgangsmåte kan være å finne formen på modellen fra fysikk for så å estimere parametrene i den modellen fra eksperiment.



Figur 4: Generell systemidentifikasjon [15]. Det vises her at modellering kan skje både fra det fysiske systemet ved bruk av matematisk modellering og via eksperiment utført på systemet.

2.5.1 Fra fysisk system

Generelt: Når man skal modellere et system ut fra fysikken til systemet, er det stort sett en balanseligning man tar utgangspunkt i. Dette kan blant annet være massebalanse, energibalanse eller impulsbalanse. Hovedideen bak disse er å sette opp endringen i systemet på en side og påvirkningene på den andre siden. Ved å utlede disse ender en så opp med en differensialligning eller en differanseligning avhengig av om man studerer

systemet i et kontinuerlig eller diskret tilfelle. Denne ligningen kan så brukes videre for å simulere hvordan systemet vil oppføre seg under forskjellige situasjoner.

Fordelen med å bruke matematisk modellering er at det er mulig å si noe om systemet uten å måtte gjøre eksperiment. Årsaken til å ønske å unngå eksperiment kan være at det er dyrt, tidskrevende eller at det i noen tilfeller er umulig å skape de situasjonene som man ønsker å lete etter. Det vil for eksempel være ugunstig å la et kjernekraftverk løpe løpsk fordi man ønsker å se hvilke målinger dette vil gi. Dersom man istedenfor lager en matematisk modell vil det være mulig å forutsi hva som kommer til å skje i de forskjellige tilfellene. Det er da også mulig å lage simuleringer som kan gjøres mye raskere og billigere enn ved å lage et eksperiment for hver enkel situasjon.

Eksempel: For en massebalanse blir det:

Endring i masse = massestrøm inn - massestrøm ut

$$\frac{dm}{dt} = \sum w_i - \sum w_u. \tag{2.5.1}$$

For en energibalanse blir oppsettet:

Endring i energi = energistrøm inn - energistrøm ut

$$\frac{dE}{dt} = \sum Q_i - \sum Q_u. \tag{2.5.2}$$

For å utvikle modellen må man så kunne si noe om de forskjellige størrelsene. Dersom man for eksempel ønsker å regulere høyden i en tank vil man kunne si noe om hvor mye masse det er i tanken basert på høyden og man vil typisk kunne si noe om volumflyten inn og ut.

$$m = hA\rho \tag{2.5.3}$$

$$\sum w_i = q_i\rho \tag{2.5.4}$$

$$\sum w_u = q_u\rho. \tag{2.5.5}$$

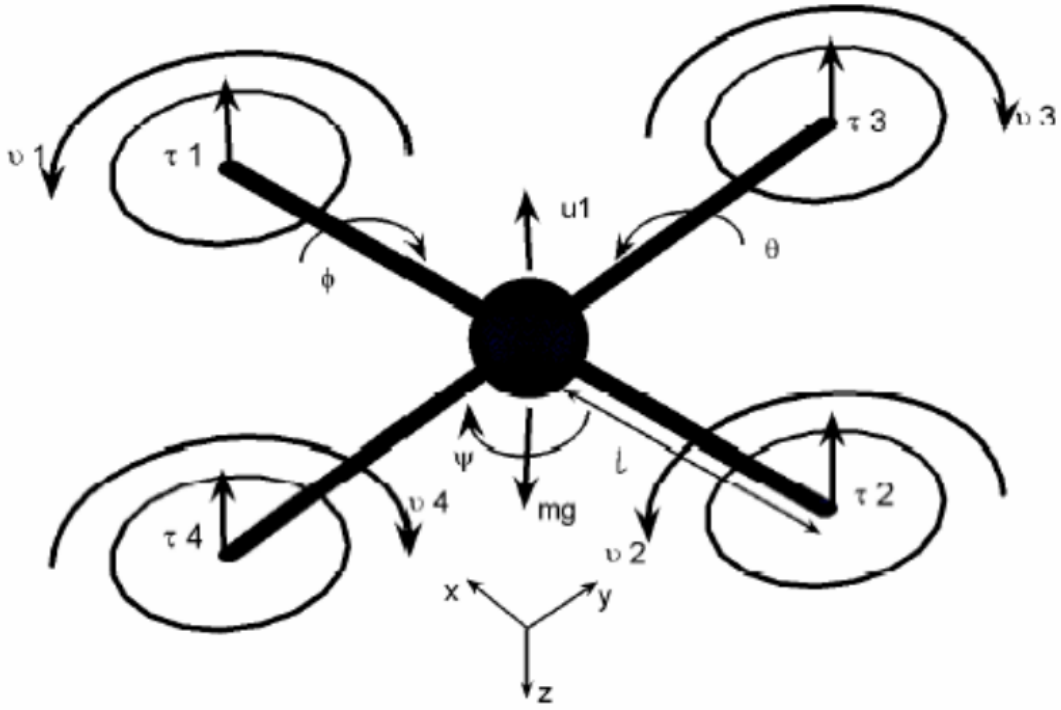
Den endelige formelen blir da:

$$\frac{dh}{dt} = \frac{q_i - q_u}{A}. \tag{2.5.6}$$

Denne kan så brukes for å simulere systemet ved forskjellige inn- og ut-strømmer.

UAV: I forbindelse med et quadcopter, som er blitt brukt i denne oppgaven, vil det være nødvendig med en del informasjon om farkosten for å kunne gjøre nytte av en matematisk modellering. Dette er fordi man i de matematiske formlene gjør bruk av parametrer som ikke er direkte tilgjengelige på UAV-en. I slike tilfeller er det nødvendig å utføre eksperiment for å komplementere den matematiske modelleringen. Det vil si at formlene som blir funnet inneholder parametrer som ikke kan bli funnet matematisk. Disse parametrene må da bli funnet eksperimentelt.

Ettersom det som blir logget på UAV-en er en Pulse Width Modulation (PWM) kommando til motorene vil det være nødvendig å utføre eksperiment som kartlegger hvordan dette gir utslag i Runder Per Minutt (RPM) og kraft. Det er ikke blitt gjort slike eksperiment i denne oppgaven og det er derfor heller ikke blitt fokusert på å lage en matematisk modell av UAV-en fra fysikk. Dette kan være aktuelt å gjøre ved videre arbeid og det kan da være av interesse å se på [16] og [2]. Begge disse utleder matematiske modeller for en UAV, men med litt forskjellig framgangsmåte.



Figur 5: En UAV-modell brukt for utledning av formler [2].

I fig. 5 er det vist utgangspunktet som er vanlig ved utledning av formler for en UAV.

Ligningene som settes opp for en UAV er i utgangspunktet:

$$u_1 = \tau_1 + \tau_2 + \tau_3 + \tau_4 \tag{2.5.7}$$

$$\theta = l(\tau_1 - \tau_2) \tag{2.5.8}$$

$$\phi = l(\tau_3 - \tau_4) \tag{2.5.9}$$

$$\psi = v_1 + v_2 - v_3 - v_4. \tag{2.5.10}$$

Der betegnelsene er som følger:

u_1: Vertikal kraft.
θ: Pitch moment.
ϕ: Roll moment.
ψ: Yaw moment.
τ_i: Kraft generert på motor i.
v_i: Dreiemoment generert på motor i.
l : Lengde fra senter til motor.

Ettersom man på denne UAV-en ikke måler kraft eller dreiemoment direkte er det nødvendig å utføre eksperiment for å finne disse forholdene.

2.5.2 Fra eksperiment

I denne oppgaven er Matlabs Curve Fitting Toolbox blitt brukt for å tilpasse data. Dette er blitt gjort i seksjon 4.3. Det er også gjort bruk av Matlabs System Identification Toolbox for å finne ARX-modeller. Dette er brukt i seksjon 3.2 og seksjon 4.1. I seksjon 4.5 er Matlab funksjonen *nlarx()* brukt for å estimere ulineære ARX-modeller.

Ved modellering fra data er det nødvendig med en annen angrepsvinkel enn ved fysikk. Man står her fritt til å velge hvordan modellen skal se ut. Dette høres i utgangspunktet bra ut, men det innfører utfordringen med at dersom modellen som velges ikke passer med dataen en ønsker å modellere vil man aldri kunne få gode resultater. Modellen man velger inneholder noen ukjente og det er disse ukjente som en ønsker å finne ved å bruke den dataen som er tilgjengelig. Det finnes mange måter å utføre modellering fra data på, og her presenteres de som er blitt brukt i denne oppgaven.

Felles for mange av metodene er at de ofte gjør bruk av en teknikk som kalles minste kvadraters metode. Dette er en metode som minimerer kvadratet av feil i estimatet. Den

tar utgangspunkt i at modellen kan skrives på formen:

$$Ax = b. \tag{2.5.11}$$

Metoden søker dermed å minimere uttrykket:

$$\|Ax - b\|^2. \tag{2.5.12}$$

Dette gjøres ved å derivere funksjonen og sette den lik null. Etter litt regning kommer man fram til at systemet som minimerer kvadratet av feilen da er gitt som:

$$Cx = f. \tag{2.5.13}$$

Der de forskjellige delene er gitt som:

$$C = A^T A \tag{2.5.14}$$

$$f = A^T b. \tag{2.5.15}$$

2.5.3 Eksempel på kurvetilpassing

Kurvetilpassing baserer seg på å se på en eller flere inngangsvariabler og ut fra de regne ut en tilhørende utgangsverdi. Metoden baserer seg på å ta utgangspunkt i en generell matematisk form for så å estimere parametrene i formen basert på tilgjengelig data. Valget av den matematiske formelen kan være grunnet i fysikken av problemet, studering av data som ønskes estimert eller prøving og feiling. Når valg av matematisk form er valgt er det ofte ganske rett fram å bruke minste kvadraters metode for å estimere parametrene, men iblant må man manipulere uttrykket litt for å få det på en form som kan brukes.

Et eksempel gis her der det er generert datapunkt i steg på 0.2 fra formelen:

$$y = 1.2e^{0.8x}, \qquad x \in [0, 4]. \tag{2.5.16}$$

Det skal forsøkes å tilnærme disse datapunktene ved å bruke en lineær funksjon og en eksponentiell funksjon. Den lineære funksjonen er på formen:

$$y = ax + b. \tag{2.5.17}$$

Bruker minste kvadraters metode på dette og får parametrene:

$$a = 6.2304 \tag{2.5.18}$$

$$b = -3.3089. \tag{2.5.19}$$

En eksponentiell funksjon har formen:

$$y = ae^{bx}. \tag{2.5.20}$$

For å finne parametrene i denne er det nødvendig å manipulere uttrykket litt, dette gjøres på følgende måte:

$$ln(y) = ln(ae^{bx}) \tag{2.5.21}$$

$$ln(y) = \underbrace{ln(a)}_{a_1} + ln(e^{bx}) \tag{2.5.22}$$

$$ln(y) = a_1 + bx. \tag{2.5.23}$$

Dette kan nå brukes med minste kvadraters metode og parametrene finnes til å være:

$$a_1 = 0.1823 \rightarrow a = e^{a_i} = 1.2 \tag{2.5.24}$$

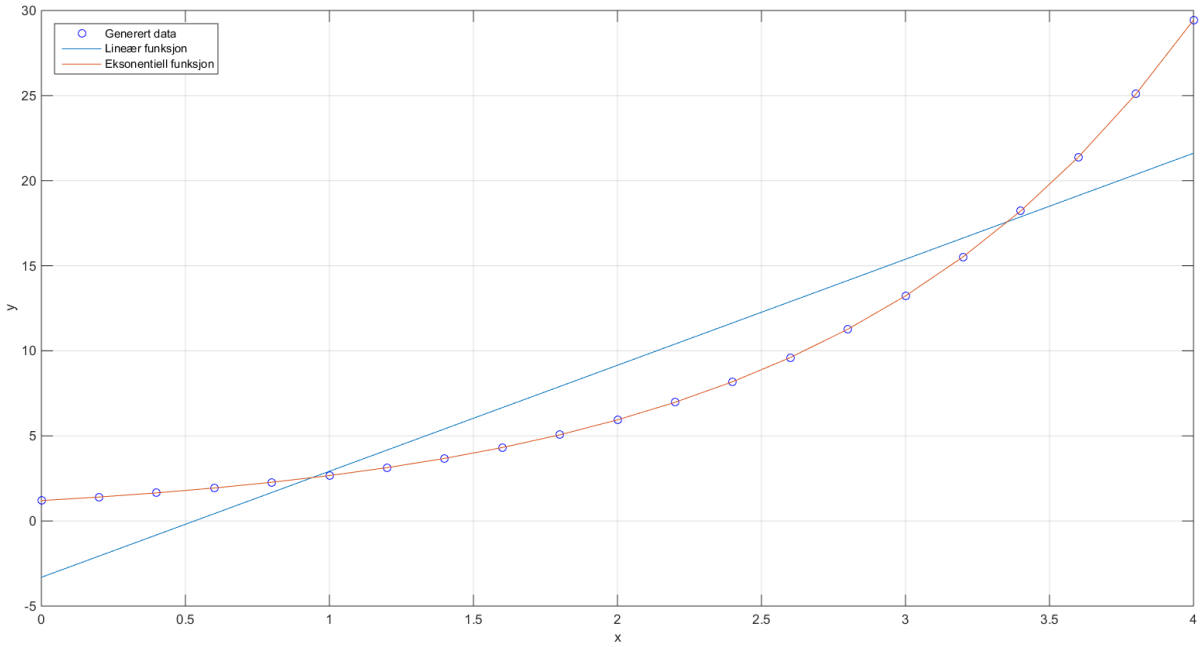
$$b = 0.8. \tag{2.5.25}$$

Dette medfører at den endelige formelen blir:

$$y = 1.2e^{0.8x}. \tag{2.5.26}$$

Som er den funksjonen dataen er generert fra.

Et plott av generert data samt de forskjellige estimerte kurvene er vist i fig. 6.



Figur 6: Kurvetilpassing fra data. Det vises her viktigheten av å bruke riktig modell ved estimering av parametrer.

Vi ser her faren ved å bruke modellering fra data. Dersom antagelsen om hvordan den underliggende funksjonene ser ut, vil resultatet bli dårlig. Det er tydelig at den lineære modellen gir et dårligere resultat når man forsøker å tilpasse den til eksponentiell data.

2.5.4 Eksempel på ARMAX

AutoRegressiv Mean Average eXogeneous (ARMAX) er en teknikk som brukes for å tilpasse data til formen:

$$A(q)\overline{y}(t) = B(q)\overline{u}(t) + C(q)\overline{e}(t), \tag{2.5.27}$$

$$A(q) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na} \tag{2.5.28}$$

$$B(q) = (b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{nb} z^{-nb+1}) z^{-nk} \tag{2.5.29}$$

$$C(q) = 1 + c_1 z^{-1} c_2 z^{-2} + \ldots + c_{nc} z^{-nc}. \tag{2.5.30}$$

Her er de forskjellige delene:

- A(q) er AR-delen, det vil si den autoregressive. Dette er den delen som husker gamle utgangsverdier.
- B(q) er X-delen, det vil si den delen som håndterer pådrag til prosessen.

- C(q) er MA-delen, det vil si den delen som håndterer støy.

For å estimere de forskjellige parametrene må en først bestemme størrelsen til de forskjellige polynomene. Dette kan i noen tilfeller være et resultat av prøving og feiling, mens noen programmer har innebygd en mulighet for å teste mange kombinasjoner og sammenligne disse, Matlab sin System Identification Toolbox er et eksempel på et slikt program.

Når størrelsen til de forskjellige polynomene er valgt, kan man bruke minste kvadraters metode for å estimere parametrene ved å minimere feilen i modellen:

$$y(k) = \phi^T(k)\Theta + e(k) \tag{2.5.31}$$

$$Y(k) = \Phi(k)\Theta + E(k) \tag{2.5.32}$$

$$\hat{\Theta}(k) = [\Phi^T(k)\Phi(k)]^{-1}\Phi^T(k)Y(k). \tag{2.5.33}$$

Et eksempel på hvordan det er mulig å estimere en ARX modell er gitt her:

Begynner med å generere data for en step-respons med transferfunksjon:

$$H(s) = \frac{0.85}{2s+1}. \tag{2.5.34}$$

Data fra denne modellen er generert med et tidssteg på 0.1 sekunder.

For å vise at ARX-modellen stemmer med virkeligheten, begynnes det her med å diskretisere modellen. Dette gjøres på følgende måte:

$$H(z) = (1 - z^{-1})\mathcal{Z}\left\{\mathcal{L}^{-1}\left[\frac{h(s)}{s}\right]|_{t=kT}\right\} \tag{2.5.35}$$

$$H(z) = (1 - z^{-1})\mathcal{Z}\left\{\mathcal{L}^{-1}\left[\frac{0.85)}{s(2s+1)}\right]|_{t=kT}\right\} \tag{2.5.36}$$

$$H(z) = (1 - z^{-1})\mathcal{Z}\left\{(0.85 - 0.85e^{-\frac{t}{2}})|_{t=kT}\right\} \tag{2.5.37}$$

$$H(z) = (1 - z^{-1})\mathcal{Z}\left\{0.85 - 0.85e^{\frac{kT}{2}}\right\} \tag{2.5.38}$$

$$H(z) = (1 - z^{-1})\left(0.85\frac{1}{1 - z^{-1}} - 0.85\frac{z}{z - e^{-\frac{T}{2}}}\right) \tag{2.5.39}$$

$$H(z) = 0.85 - 0.85\frac{z(1 - z^{-1})}{z - e^{-\frac{T}{2}}} \tag{2.5.40}$$

$$H(z) = 0.85 - 0.85\frac{z - 1}{z - e^{-\frac{T}{2}}} \tag{2.5.41}$$

$$H(z) = 0.85\frac{(z - e^{-\frac{T}{2}})}{z - e^{-\frac{T}{2}}} - 0.85\frac{z - 1}{z - e^{-\frac{T}{2}}} \tag{2.5.42}$$

$$H(z) = \frac{-0.85e^{z-\frac{T}{2}} + 0.85}{z - e^{-\frac{T}{2}}}. \tag{2.5.43}$$

Setter inn for T = 0.1 sekunder, som er sampletiden, og får:

$$H(z) = \frac{0.0415}{z - 0.9512}. \tag{2.5.44}$$

Noe som gir følgende differanseligning:

$$y(n + 1) = 0.9512y(n) + 0.0415x(n). \tag{2.5.45}$$

Nå skal modellen estimeres som en ARX-modell. Begynner da med å sette den opp på generell form:

$$y(n) = -ay(n - 1) + bu(n - 1) + e(n). \tag{2.5.46}$$

Her er $e(n)$ normalfordelt støy som er lagt til med *randn()* funksjonene i Matlab.

Alle målingene blir så slått sammen i en matrise og vi får dermed:

$$Y = \Phi\theta + E. \tag{2.5.47}$$

Der de forskjellige delene er:

$$Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{bmatrix}, \quad E = \begin{bmatrix} e(1) \\ e(2) \\ \vdots \\ e(n) \end{bmatrix}, \quad \Phi = \begin{bmatrix} \phi(1) \\ \phi(2) \\ \vdots \\ \phi(n) \end{bmatrix} \tag{2.5.48}$$

$$\phi(n) = [-y(n), u(n)] \tag{2.5.49}$$

$$\theta = \begin{bmatrix} a \\ b \end{bmatrix}. \tag{2.5.50}$$

Bruker minste kvadraters metode på dette og ender da opp med:

$$\hat{\theta} = [\Phi^T \Phi]^{-1} \Phi^T Y. \tag{2.5.51}$$

Når dette regnes ut på datasettet gir det:

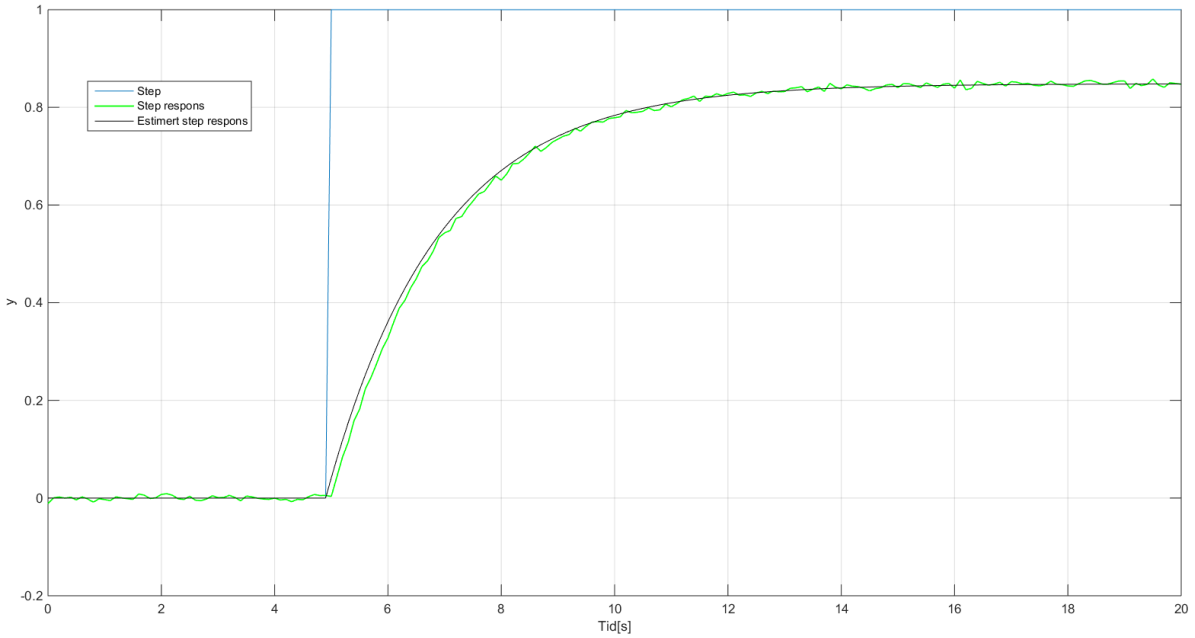
$$\hat{\theta} = \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -0.9508 \\ 0.0417 \end{bmatrix}. \tag{2.5.52}$$

Satt inn igjen i formelen gir dette:

$$y(n+1) = 0.9508y(n) + 0.0417x(n). \tag{2.5.53}$$

En ser her at parametrene ikke er helt like som de som ble funnet i ligning 2.5.45. Dette er fordi det ble lagt til litt støy på utgangen. Det viktige er at parametrene er ganske like

Et plot av simulert og estimert step-respons er vist i fig. 7. Step responsen er blitt pålagt litt normalfordelt støy for å simulere målestøy, men den estimerte step-responsen klarer likevel å finne parametrene rimelig bra.



Figur 7: ARX estimering av step-respons. Utgangen er pålagt litt støy for å simulere målestøy i systemet.

2.5.5 Eksempel på ulineær ARX

Når man har et system der forholdet ikke er lineært, vil det ikke gi gode resultater å bruke en ARMAX-modell. I slike tilfeller kan det allikevel være mulig å lage en modell ved å bruke en ulineær ARMAX-modell. Ettersom ulineariteten fører til mer kompliserte utregninger brukes ofte bare en ulineær ARX (NARX). Tanken bak NARX er å utvide ARX-modellen til å være ulineær. Det vil si at man får en modell på formen:

$$y(t) = A(q)f_2[y(t)] + B(q)f_1[u(t)] + e(t). \tag{2.5.54}$$

Der det er gitt at f_2 og f_1 er ulineære kontinuerlige funksjoner. Disse kan velges til hvilken som helst funksjon, men en populær versjon er å bruke 'cubic spline' for å tilnærme de forskjellige ulineære funksjonene. Dette er en funksjon som har formen:

$$f(x) = \sum_{k=2}^{m_1-1} \beta_k |x - x_k|^3 + \beta_{m_1} x + \beta_{m_1+1}. \tag{2.5.55}$$

Her er det β parametrene som må estimeres og u_k parametrene er knutepunkt. Det vil si

punkter på linjen som skal estimeres. Disse må følge formen:

$$u_1 = u_{min} < u_2 < \ldots < u_{m1} = u_{max}. \tag{2.5.56}$$

En metode for å finne parametrene i en slik modell er gitt i [17]. Algoritmen som brukes der er gitt som:

Initialisering Sett $f_1(u) = u$ og $f_2(y) = y$ og estimer $A(q)$ og $B(q)$ ved å bruke minste kvadraters metode.

Iterering Definer $\hat{A}_{(i)}(q)$, $\hat{B}_{(i)}(q)$, $\hat{f}_{1(i)}$ og $\hat{f}_{2(i)}$ som estimatene fra iterasjon i.

1. Kalkuler parametrene i $\hat{f}_{1(i+1)}[u(t)]$ ved å holde $\hat{f}_{2(i)}[y(t)]$, $\hat{A}_{(i)}(q)$ og $\hat{B}_{(i)}(q)$ konstante og minimere uttrykket:

$$\sum_{t=1}^{N} \{y(t) - \hat{A}_{(i)}(q)\hat{f}_{2(i)}[y(t)] - \hat{B}_{(i)}(q) f_{1(i+1)}[u(t)]\}^2. \tag{2.5.57}$$

2. Kalkuler parametrene i $\hat{f}_{2(i+1)}[u(t)]$ ved å holde $\hat{f}_{1(i+1)}[y(t)]$, $\hat{A}_{(i)}(q)$ og $\hat{B}_{(i)}(q)$ konstante og minimere uttrykket:

$$\sum_{t=1}^{N} \{y(t) - \hat{A}_{(i)}(q) f_{2(i+1)}[y(t)] - \hat{B}_{(i)}(q)\hat{f}_{1(i+1)}[u(t)]\}^2. \tag{2.5.58}$$

3. Kalkuler $\hat{A}_{(i+1)}(q)$ og $\hat{B}_{(i+1)}(q)$ ved å holde $\hat{f}_{1(i+1)}(u)$ og $\hat{f}_{2(i+1)}(y)$ konstante og minimere uttrykket:

$$\sum_{t=1}^{N} \{y(t) - A_{(i+1)}(q)\hat{f}_{2(i+1)}[y(t)] - B_{(i+1)}(q)\hat{f}_{1(i+1)}[u(t)]\}^2. \tag{2.5.59}$$

Iterer i, gå tilbake til steg 1. og gjenta til algoritmen konvergerer.

Alle stegene i algoritmen er det mulig å utføre ved bruk av minste kvadraters metode. Det er ikke blitt implementert en slik funksjon i denne oppgaven, det vises istedenfor til [17] for eksempel ved bruk av denne algoritmen.

2.6 Klassifisering

Når man har to eller flere typer data som man ønsker å skille fra hverandre, er det uttrykket klassifisering som oftest blir brukt. Det finnes mange måter å utføre klassifisering på, men felles er typisk at man tar utgangspunkt i en algoritme som må læres opp på et sett av de dataene som man ønsker å skille fra hverandre. Den opplærte algoritmen brukes så videre for å skille nye datapunkt inn i de forskjellige klassene som er definert.

2.6.1 Terskelverdi

Den enkleste metoden for klassifisering er å bruke en terskelverdi. Her velger man en verdi og sier at alle målinger under terskelen tilhører en klasse og alt over terskelen tilhører en annen klasse. Et enkelt og visuelt eksempel på mye brukt terskelverdi-klassifisering er svart/hvitt konvertering av bilder. Dersom man tar utgangspunkt i et gråtone-bilde sier man at dersom gråtonen er mørkere enn en angitt verdi skal pikselen tolkes som svart, ellers skal den tolkes som hvit. En åpenbar fordel med metoden er dens ekstremt lettfattelige implementering hvor den eneste utfordringen er å bestemme hva terskelverdien skal være. Ulemper ved metoden er at den er lite robust for endringer og det er vanskelig å gi en forklaring på hvorfor terskelverdien blir valgt til å være som den er. Valget bærer ofte preg av å være basert på empiriske forsøk der man prøver å få flest mulig av treningsdataene korrekt klassifisert. Et problem med denne tilnærmingen i denne situasjonen er at systemet ikke er statisk, men dynamisk. Det vil si at en terskelverdi ikke vil være gyldig for systemet i store deler av tiden. Det kan da være mulig å lage en adaptiv terskelverdi som endrer seg i takt med andre parametrer som blir målt.

2.6.2 Statistikk

Å bruke statistikk og statistiske metoder kan gi mer robuste og tilpasningsdyktige algoritmer. Dette er fordi de forsøker å minimere feilklassifisering basert på matematiske metoder. Det er forskjell på fremgangsmåtene, der noen tar antagelser om underliggende fordelinger, slik som Maximum likelihood (ML), mens andre, slik som k-means og parzen-window ikke gjør det. Felles for metodene er at de prøver å beskrive det underliggende systemet fra et statistisk perspektiv.

2.6.3 Maximum Likelihood

Maximum likelihood (ML) klassifisering tar utgangspunkt i at den underliggende statistiske modellen er på en kjent form, og ofte brukes en Gaussisk fordeling som utgangspunkt. Da har man allerede antatt at data er normalfordelt, noe som ofte er tilfelle, men ikke alltid. Det må derfor undersøkes om det er en rimelig antagelse eller ikke. Man trener så en klassifiserer på test-data for å bestemme middelverdi, μ og standardavvik, σ. Dette kan senere bli brukt på ny data for å si hvilken fordeling det er som best forklarer den nye dataen. En styrke ved denne fremgangsmåten er at den kan tilpasses data på en vitenskapelig måte og man kan si at det er en optimal klassifiserer i den forstand at ny data alltid blir klassifisert på en måte som minimerer feil. En ulempe er at man må anta noe om modellen, og dersom denne antagelsen ikke er rimelig vil klassifisereren fungere dårlig. Det er også mulig å legge inn flere kriterier i klassifiseringsmetoden slik som kost-funksjoner. Dette er typisk en konstant som sier noe om hvor dyrt det skal være å feilklassifisere de forskjellige klassene. I den enkleste formen der det er to klasser kan man si at det er mye farligere å feilklassifisere den ene klassen enn den andre. Dette vil da endre klassifiseringsgrensen bort fra den klassen som er dyrest. Det vil totalt sett gi flere feilklassifiseringer, men mindre total kost.

Dersom man antar en Gaussisk form på den underforliggende fordelingen vil man ha en matematisk formel på formen [18]:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}. \tag{2.6.1}$$

Her er det to ukjente parametre, μ og σ. Disse må estimeres basert på den dataen som er tilgjengelig.

$$\mu = \frac{1}{n}\sum x_k \tag{2.6.2}$$

$$\sigma = \frac{1}{n-1}\sum (x_i - \mu)^2. \tag{2.6.3}$$

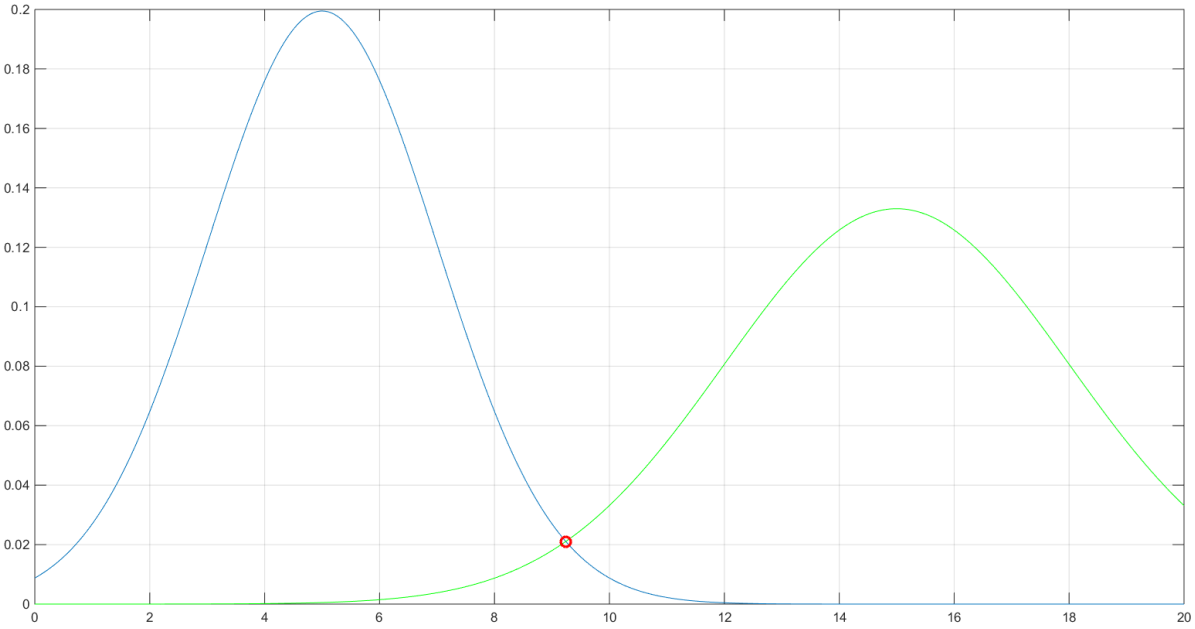
Dersom man har to klasser vil man finne to forskjellige parametre for de to klassene. For å minimere feilklassifisering, setter man så klassifiseringsgrensen til å være der sannsynligheten til de to klassene er lik. Dette er illustrert i fig. 8. For å finne denne grensen må

man løse ligningen:

$$p_1(x) = p_2(x) \tag{2.6.4}$$

$$\frac{1}{\sqrt{2\pi\sigma_1}} e^{-\frac{1}{2}\frac{(x-\mu_1)^2}{\sigma_1^2}} = \frac{1}{\sqrt{2\pi\sigma_2}} e^{-\frac{1}{2}\frac{(x-\mu_2)^2}{\sigma_2^2}}. \tag{2.6.5}$$

Dette er en metode som er blitt brukt i seksjon 4.2 for å klassifisere strømforbruket til UAV-en.



Figur 8: Klassifisering ved bruk av maximum likelihood. Det røde punktet viser klassifiseringsgrensen som gir minst feilklassifiseringer.

2.6.4 Support Vector Machines

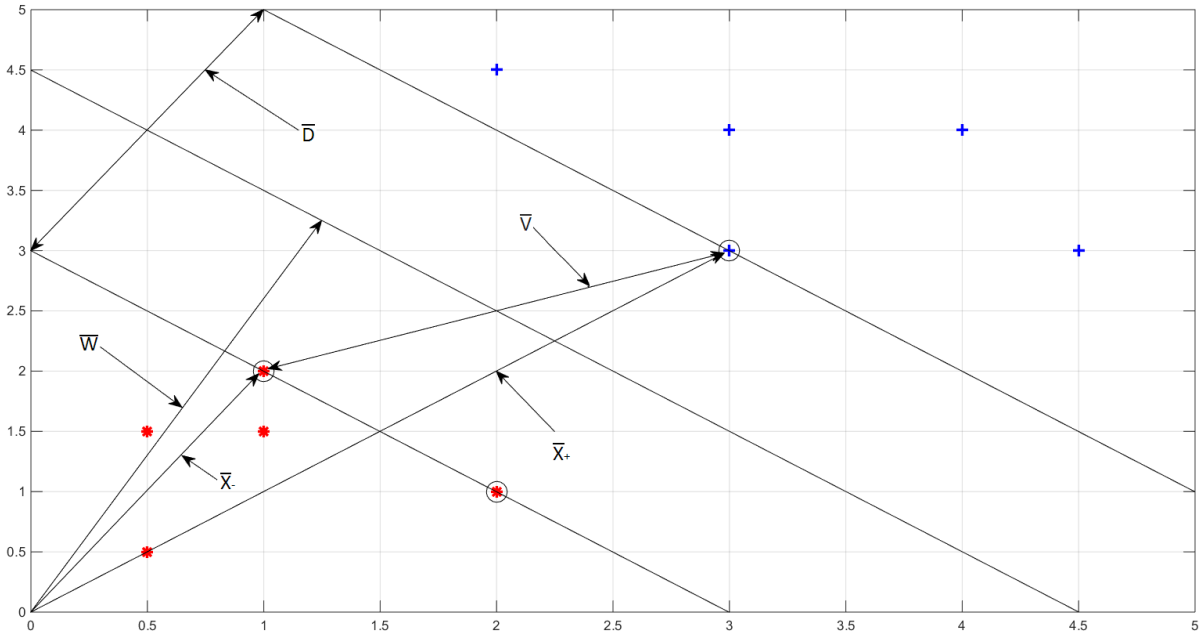
Dette er et kapittel som er basert på [19] og [20].

Support Vector machines (SVM) er en klassifiseringsalgoritme som plukker ut de viktigste punktene i et datasett og bruker disse for å lage en klassifiseringsgrense. Ved ulineære datasett brukes i tillegg en kernel-funksjon for å flytte klassifiseringsproblemet til en høyere dimensjon, noen ganger til og med til et uendelig dimensjoners rom, for så å klassifisere problemet der, før det flyttes tilbake til det opprinnelige rommet.

SVM ble først utviklet i Russland av Vladimir Vapnik allerede på 60-tallet, men på grunn av mangel på datamaskiner var det ikke mulig å utforske teorien videre. På 90-tallet flyttet Vapnik til USA og det var her, under utviklingen av automatisk gjenkjennelse av

håndskrift, at SVM viste seg å fungere.

For å forstå hvordan SVM fungerer skal vi her først se på en lineært separabel situasjon som er vist i fig. 9. Det er her to klasser, blå kryss og røde stjerner, med to datapunkter hver. Grunntanken bak SVM er å trekke en linje mellom datapunktene slik at bredden D maksimeres.



Figur 9: SVM klassifisering av lineært separabel data. SVM algoritmen forsøker å finne en klassifiseringsgrense som maksimerer $\overline{D}$. Datapunkter med svart ring rundt er de som blir brukt for å definere grensen.

For å utvikle denne ideen begynner vi med å definere en klassifiserings-regel. Dette gjør vi ved å innse at $\overline{w}$ står perpendikulært på linjen som trekkes. Prikkproduktet mellom denne vektoren og en datavektor, $\overline{u}$, blir da avstanden som det datapunktet har i forhold til en grense. Vi sier derfor at dersom:

$$\overline{w} \cdot \overline{u} + b \geq 0 \tag{2.6.6}$$

så klassifiseres punktet som et blått kryss.

Vi velger så å legge inn en begrensing på datapunktene ved å si at:

$$\overline{w} \cdot \overline{x}_+ + b \geq 1 \tag{2.6.7}$$

$$\overline{w} \cdot \overline{x}_- + b \leq 1. \tag{2.6.8}$$

Videre introduseres det en variabel, y_i som er +1 for positive datapunkt og -1 for negative datapunkt. Ved å gange dette inn i de to forrige ligningene får vi at begge blir til:

$$y_i(\overline{w} \cdot \overline{x}_i + b) \geq 1. \tag{2.6.9}$$

Vi har dermed fått redusert disse to ligningene til en enkel ligning. Og ved å manipulere denne litt, samt å si at vi bare er interessert i punkter som ligger akkurat på grensen, får vi følgende:

$$y_i(\overline{w} \cdot \overline{x}_i + b) - 1 = 0. \tag{2.6.10}$$

Ettersom vi ønsker å maksimere avstanden D, prøver vi nå å lage et uttrykk for den. Vi ser at vektoren $\overline{x}_+$ minus vektoren $\overline{x}_-$ i fig. 9 lager vektoren V. For å få bredden til D kan vi ta prikkproduktet mellom vektoren V og en enhetsvektor som peker i samme retning som $\overline{w}$. Denne vektoren kan vi finne ved å normalisere $\overline{w}$ på følgende måte:

$$\frac{\overline{w}}{\|\overline{w}\|}. \tag{2.6.11}$$

Vi får dermed at:

$$D = (\overline{x}_+ - \overline{x}_-) \cdot \frac{\overline{w}}{\|\overline{w}\|}. \tag{2.6.12}$$

Vi substituerer så inn ligning 2.6.10 for de to punktene, $\overline{x}_+$ og $\overline{x}_-$, og får dermed:

$$D = \frac{2}{\|\overline{w}\|}. \tag{2.6.13}$$

Det er denne vi ønsker å maksimere, og det blir det samme som å minimere$||\overline{w}||$. Ettersom dette skal deriveres senere, blir det matematisk lettere å minimere $\frac{1}{2}\|\overline{w}\|^2$.

Vi definerer nå en Lagrange-funksjon for å legge inn restriksjonen i ligning 2.6.10.

$$L = \frac{1}{2}\|\overline{w}\|^2 - \sum \alpha_i[y_i(\overline{w} \cdot \overline{x}_i + b) - 1]. \tag{2.6.14}$$

Vi tar så den deriverte av L.

$$\frac{\partial L}{\partial \overline{w}} = \overline{w} - \sum \alpha_i y_i \overline{x}_i = 0 \Rightarrow \overline{w} = \sum \alpha_i y_i \overline{x}_i \tag{2.6.15}$$

$$\frac{\partial L}{\partial b} = -\sum \alpha_i y_i = 0 \Rightarrow \sum \alpha_i y_i = 0. \tag{2.6.16}$$

Disse substitueres inn i funksjonen for L.

$$L = \frac{1}{2}(\sum(\alpha_i y_i \overline{x}_i)(\sum \alpha_j y_j \overline{x}_j) - (\sum \alpha_i y_i \overline{x}_i)(\sum \alpha_j y_j \overline{x}_j) - b\underbrace{\sum \alpha_i y_i}_{0} + \sum \alpha_i \quad (2.6.17)$$

$$L = \sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \overline{x}_i \cdot \overline{x}_j. \quad (2.6.18)$$

Ifølge teori om Lagrangian søker vi altså nå å maksimere L med betingelsene:

$$\sum_i y_i \alpha_i = 0 \quad (2.6.19)$$

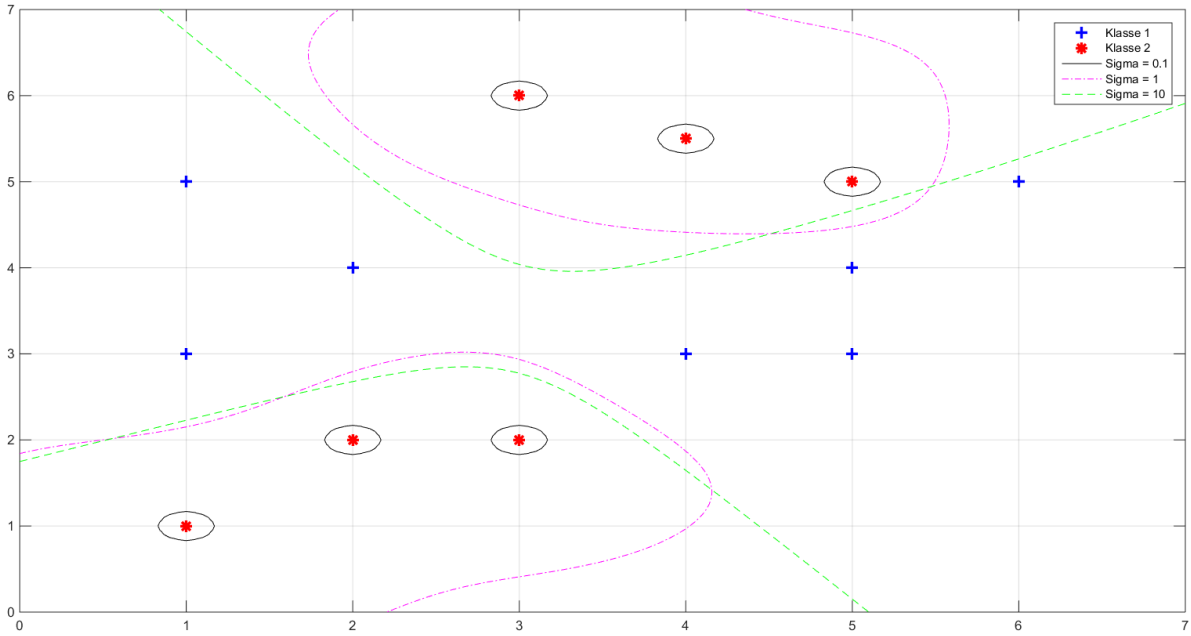
$$\alpha_i \geq 0. \quad (2.6.20)$$

Vi legger ligningene 2.6.15 og 2.6.16 inn igjen i klassifiseringsregelen vår og får:

$$\sum \alpha_i y_i \overline{x}_i \cdot \overline{u} + b \geq 0. \quad (2.6.21)$$

Teorien for en klassifiseringsalgoritme for lineært separabel data er nå ferdig og vi har en metode for å lære opp algoritmen og en metode for å klassifisere ny data basert på den opplærte algoritmen.

Dette er bra i seg selv, men tilfører ikke noe særlig i forhold til andre klassifiseringsalgoritmer. Styrken til SVM kommer tydeligere fram når man forsøker å klassifisere et datasett som ikke er lineært separabelt, som i fig. 10.



Figur 10: Ulineært separabel data. Her blir effekten av å variere 'kernel' størrelsen tydelig. Dersom denne er for liten, fører det til overtrening og grensene krymper inn rundt de forskjellige datapunktene.

For å få dette til å virke er det nødvendig å innføre det som er kjent som et 'kernel' triks. Dette er et triks som flytter den ikke lineært separable dataen til en høyere dimensjon, hvor den er lineært separabel. En viktig observasjon å gjøre før man ser på hvordan man kan bevege seg opp i dimensjoner er at klassifiseringsregelen kun er avhengig av prikkproduktet mellom $\overline{x}_i$ og $\overline{u}$

Vi begynner med å kallen transformasjonen til en høyere dimensjon for:

$$\Phi(\overline{x}). \tag{2.6.22}$$

Det vi trenger fra transformasjonen er som følger:

$$\Phi(\overline{x}_i) \cdot \Phi(\overline{x}_j) \tag{2.6.23}$$

$$\Phi(\overline{x}_i) \cdot \Phi(\overline{u}). \tag{2.6.24}$$

Vi ser derfor at dersom vi har en funksjon:

$$K(\overline{x}_i, \overline{x}_j) = \Phi(\overline{x}_i) \cdot \Phi(\overline{x}_j) \tag{2.6.25}$$

så er det ikke nødvendig å eksplisitt finne funksjonen:

$$\Phi(\overline{x}). \tag{2.6.26}$$

Det er denne funksjonen vi kaller en 'kernel' funksjon. Det eksisterer et par 'kernel' funksjoner som er normale å bruke, den polynomiske og den eksponentielle:

$$K = (\overline{u} \cdot \overline{v} + 1)^n \tag{2.6.27}$$

$$K = e^{-\frac{\|x_i - x_j\|}{\sigma}}. \tag{2.6.28}$$

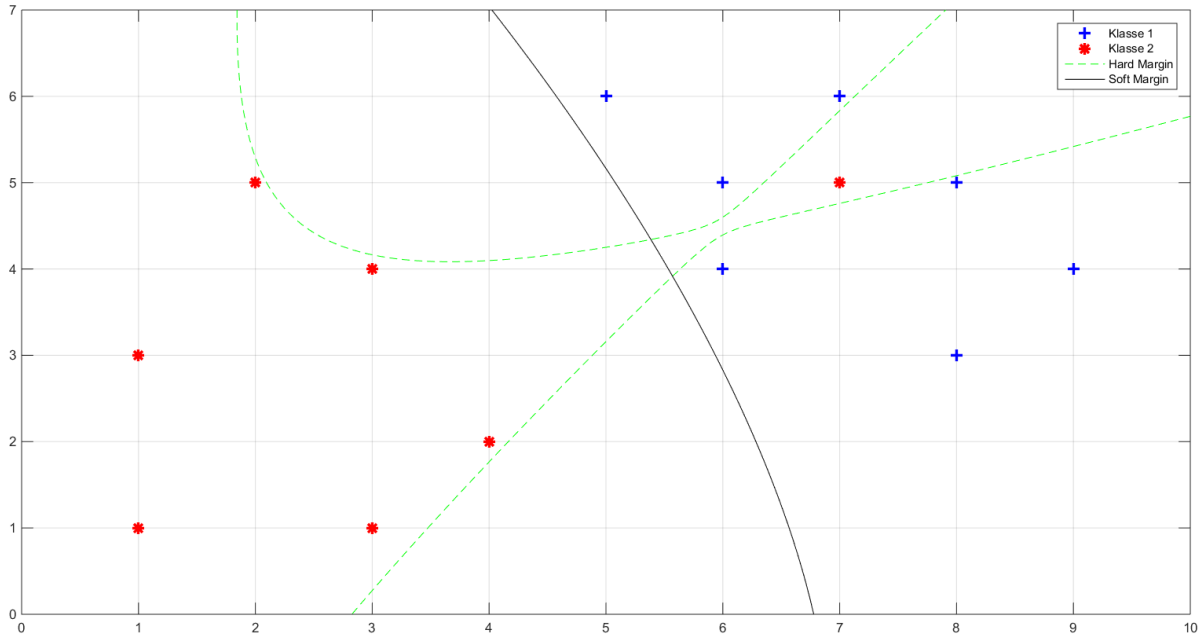
Ved å ta en Taylor-rekkeutvikling av den eksponentielle funksjonen, er det mulig å vise at dette faktisk er det samme som å bevege seg til et rom med uendelig mange dimensjoner.

Det er viktig å velge riktig funksjon til de dataene som skal trenes. Og det er viktig å velge riktig størrelse på funksjonen. Dersom man for eksempel velger for liten sigma i den eksponentielle funksjonen vil dette kunne føre til overtrening på dataen. Effekten av å variere sigma er vist i fig. 10, der sigma blir variert fra 0.1 til 10 med tilhørende klassifiseringsgrense vist. En ser her at når sigma blir for liten går bare grensen rundt datapunktene og gir dårlig generalisering.

Formlene for å trene og klassifisere er lik som de for lineært separabel data, men istedenfor å ta prikkproduktet mellom punktene direkte brukes en 'kernel'. Klassifiseringsregelen blir da:

$$\sum \alpha_i y_i K(\overline{x}_i, \overline{u}) + b \geq 0. \tag{2.6.29}$$

Det er nå mulig å klassifisere ulineært separabel data, men det er et annet problem som kan dukke opp når man prøver å klassifisere data, som er vanlig i praktiske applikasjoner. Dette er støy i målingene, noe som kan føre til at dataen ikke er 100 % separabel. Et eksempel er vist i fig. 11.



Figur 11: Ikke separabel data. En ser her at selv om det er mulig å klassifisere alle punktene rett vil det ikke nødvendigvis være det som gir best generalisering.

I disse tilfellene vil det være bedre å bruke det som kalles en 'Soft Margin', eller myk margin på godt norsk. Man tar utgangspunkt i å minimere $\|\overline{w}\|$ med betingelsen om at:

$$y_i(\overline{w} \cdot \overline{x}_i + b) - 1 = 0. \tag{2.6.30}$$

Det er nå nødvendig å introdusere en variabel som tillater et sampel å være på feil side av klassifiseringsgrensen, en slakk-variabel ζ. Vi får da følgende betingelse:

$$y_i(\overline{w} \cdot \overline{x}_i + b) \geq 1 - \zeta_i, \qquad \zeta_i \geq 0. \tag{2.6.31}$$

Det finnes flere måter å sette opp et uttrykk for å finne den optimale grensen, men de to vanligste er:

$$\|\overline{w}\| + C \sum \zeta_i \tag{2.6.32}$$

$$\|\overline{w}\| + C \sum \zeta_i^2. \tag{2.6.33}$$

Her blir den første varianten studert. Denne blir assosiert med to betingelser:

$$y_i(\overline{w} \cdot \overline{x}_i + b) \geq 1 - \zeta_i \tag{2.6.34}$$

$$\zeta_i \geq 0. \tag{2.6.35}$$

Det nye uttrykket blir nå satt opp som en Lagrangian sammen med de nye betingelsene og dette resulterer i uttrykket:

$$L = \frac{1}{2}\|\overline{w}\|^2 + C\sum_i \zeta_i - \sum_i \alpha_i[y_i(\overline{x}_i \cdot \overline{w} + b) - 1 + \zeta_i] - \sum_i r_i\zeta_i. \tag{2.6.36}$$

Vi fortsetter med å derivere denne på de forskjellige variablene og sette disse lik null. Man ender da opp med følgende ligninger:

$$\frac{\partial L}{\partial \overline{w}} = \overline{w} - \sum_i y_i\alpha_i\overline{x}_i = 0 \tag{2.6.37}$$

$$\frac{\partial L}{\partial \zeta_i} = C - \alpha_i - r_i = 0 \tag{2.6.38}$$

$$\frac{\partial L}{\partial b} = \sum_i y_i\alpha_i = 0. \tag{2.6.39}$$

Når dette substitueres tilbake i L får vi:

$$L = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j y_iy_j\alpha_i\alpha_j\overline{x}_i \cdot \overline{x}_j. \tag{2.6.40}$$

Vi ser at det er det samme som vi fikk før, men det er blitt innført en litt annen betingelse:

$$C \geq \alpha_i \geq 0. \tag{2.6.41}$$

Det er altså satt en begrensing på α_i. Betegnelsen boks-begrensing blir ofte brukt om denne formen, ettersom α_i blir begrenset til en boks med sidelengde C.

Med denne siste delen er utviklingen av SVM komplett og det er nå mulig å bruke denne algoritmen til å klassifisere et bredt spekter av parametrer, både lineært, ulineært og til og med data som ikke er 100 prosent separabel. Det er nå tydelig at SVM er en attraktiv klassifiseringsalgoritme.

I fig. 11 ser vi to forskjellige klassifiseringsgrenser. En der dataen er perfekt separert og en der et sampel er feilklassifisert. Hvilken av disse grensene som er best er det ikke mulig å gi et fullgodt svar på uten å prøve på ny data. Begge grensene er funnet i Matlab ved bruk av *fitcsvm()*, med en forskjell. I grensen som feilklassifiserer et punkt er det tatt høyde for at en forventer 10 prosent feilklassifisert data.

2.7 Kalman

En viktig redskap ved tilstandsestimering er Kalman-filteret. Kalman-filteret er en algoritme som kan implementere en tilstandsestimator, et adaptivt filter eller begge deler[15]. Kalman-filteret virker ved at det ser på de statistiske egenskapene til signalet det blir anvendt på og prøver så å minimere aposteriori-estimeringsavvikets varians. Det er dette som menes når man sier at Kalman-filteret er optimalt.

En forutsetning for å bruke Kalman-filteret er at systemet kan skrives på formen:

$$x(k+1) = \Phi x(k) + \Gamma u(k) + v(k) \tag{2.7.1}$$

$$y(k) = Dx(k) + w(k). \tag{2.7.2}$$

Dette betyr i at det bare er lineære system man kan anvende Kalman-filter på. Dersom man derimot har et ulinært system er det behov for å bruke et utvidet Kalman-filter (EKF). Hovedideen her er at man lineariserer systemet enten ved et arbeidspunkt, eller for hver måling. Den første metoden er minst prosesskrevende, men kan være unøyaktig dersom prosessen er langt unna arbeidspunktet. Ved å linearisere i hvert tidssteg vil man få et mer nøyaktig resultat, men det vil også være mer ressurskrevende.

Formlene for et vanlig Kalman-filter kan settes opp på følgende måte [15]:

$$\overline{x}(k) = \Phi\hat{x}(k-1) + \Gamma u(k-1) \tag{2.7.3}$$

$$\overline{P}(k) = \Phi\hat{P}(k-1)\Phi^T + Q \tag{2.7.4}$$

$$K(k) = \overline{P}(k)D^T(D\overline{P}(k)D^T + R)^{-1} \tag{2.7.5}$$

$$\hat{x}(k) = \overline{x}(k) + K(k)[y(k) - D\overline{x}(k)] \tag{2.7.6}$$

$$\hat{P}(k) = (I - K(k)D)\overline{P}(k). \tag{2.7.7}$$

Det er her gitt at:

Φ er transisjonsmatrisen ved tidssteg k.
Γ er pådragsmatrisen ved tidssteg k.
$u(k)$ er pådrag ved tidssteg k.
$y(k)$ er måling ved tidssteg k.
$v(k)$ er prosesstøy.
$w(k)$ er målestøy.
Q er prossesstøy.
R er målestøy.

D er målematrisen til ved tidssteg k.
$\overline{x}(k)$ er apriori tilstandsestimat ved tidssteg k.
$\hat{x}(k)$ er aposteriori tilstandsestimatet.
$\overline{P}(k)$ er kovariansmatrisen til apriori tilstandsestimatet.
$\hat{P}(k)$ er kovariansmatrisen til aposteriori tilstandsestimatet.
$K(k)$ er Kalman-forsterkning.

I denne oppgaven er det ikke behov for å bruke Kalman-filteret som en tilstandsestimator, ettersom autopiloten på UAV-en estimerer alle de tilstandene som er nødvendige. En del av målingene har derimot mye støy eller varierer veldig mye og det er derfor valgt å bruke et Kalman-filter som et adaptivt filter.

Systembeskrivelsen blir da som følger:

$$x(k+1) = x(k) + v(k) \tag{2.7.8}$$
$$y(k) = x(k) + w(k). \tag{2.7.9}$$

Vi ser her at Φ og D er blitt satt til 1 og Γ er blitt satt til 0. Dette er fordi vi bare ønsker å filtrere ett signal og ettersom det bare skal filtreres er det ingen pådrag. Prosesstøyen beholdes for å tillate systemet å følge endringer i signalet. Ved å justere Q kan man da velge hvor fort endringer i signalet skal kunne følges.

Det tilhørende Kalman-filteret blir da implementert som:

$$\overline{x}(k) = \hat{x}(k-1) \tag{2.7.10}$$
$$\overline{P}(k) = \hat{P}(k-1) + Q \tag{2.7.11}$$
$$K(k) = \overline{P}(k)(\overline{P}(k) + R)^{-1} \tag{2.7.12}$$
$$\hat{x}(k) = \overline{x}(k) + K(k)[y(k) - \overline{x}(k)] \tag{2.7.13}$$
$$\hat{P}(k) = (1 - K(k))\overline{P}(k). \tag{2.7.14}$$

Q og R må settes basert på hvilket signal det er som ønskes filtrert og hvor mye filtrering som ønskes. I denne rapporten er R blitt satt til variansen til signalet som ønskes filtrert mens Q er blitt variert til utgangssignalet ser bra ut. Et eksempel på bruk er vist i fig. 18, side 42.

3 Eksperiment med ett datasett

Det ble først foretatt en flyging av UAV-en under normale forhold, det vil si at det ikke var lagt på feil på UAV-en. Det tok litt tid for å finne en måte å generere en feil på UAV-en som etterlignet ising på propellene, og det ble derfor forsøkt å ta noen antagelser om hva som kom til å skje med farkosten ved ising. De eksisterende dataene ble så manipulert på en slik måte som antagelsene la til rette for.

Antagelsen som ble tatt var at hastigheten til farkosten ville bli noe tregere og ville få en noe tregere respons ved endringer. Dette ble implementert i Matlab ved å ta målingene av farten i x-retning og lavpassfiltrere denne, samt gange den med en faktor < 1.

Det vil si at det i denne delen er tatt utgangspunkt i ett datasett som ble logget på UAV-en ved flyging uten noen feil på farkosten. Et andre datasett ble så generert ved å gjøre noen endringer på det første på en slik måte som det ble antatt at ville være naturlig ved ising på propellene.

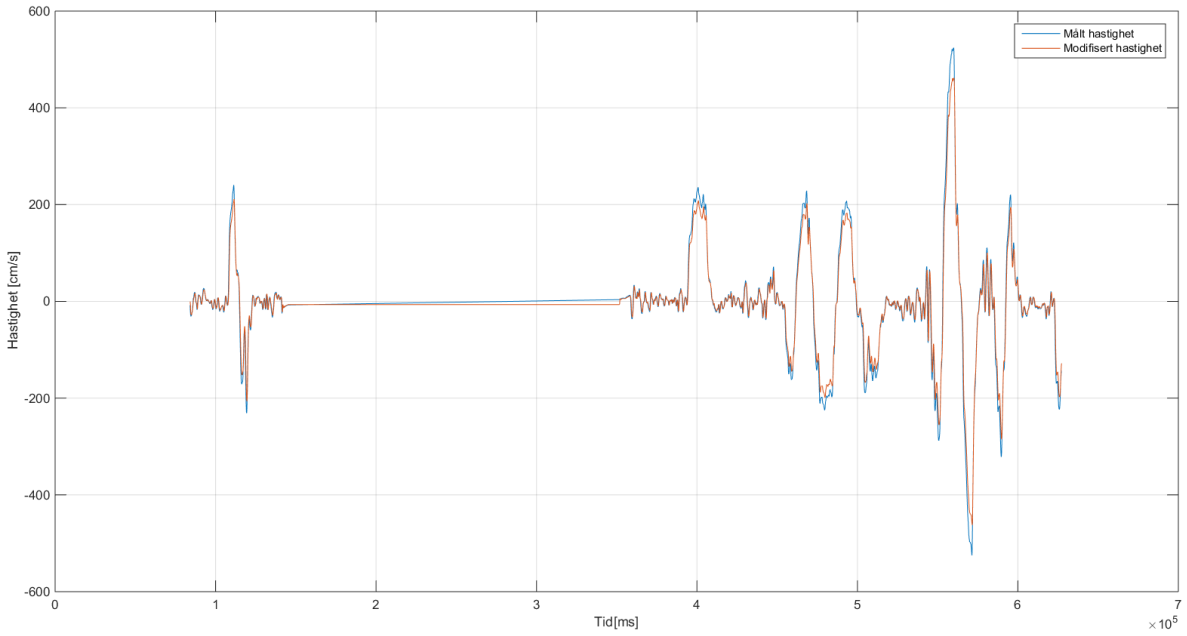
3.1 Matlab

Matlab-koden som ble brukt for å modifisere målingene er vist under.

```
%% Process VelX to create simulated data.

% Design lowpass filter
d = fdesign.lowpass('Fp,Fst,Ap,Ast',0.5,0.6,1,60);
D = design(d,'iir');
%Apply lowpass to velocity in X direction.
VelX_Sim = filter(D, VelX)*0.99;
```

En figur som viser den originale og simulerte hastigheten er vist i fig. 12. Vi ser her at det er en merkbar, men ikke overdreven forskjell mellom de to. Det ble derfor antatt at dette var innenfor det som kunne forventes ved ising på propellene.

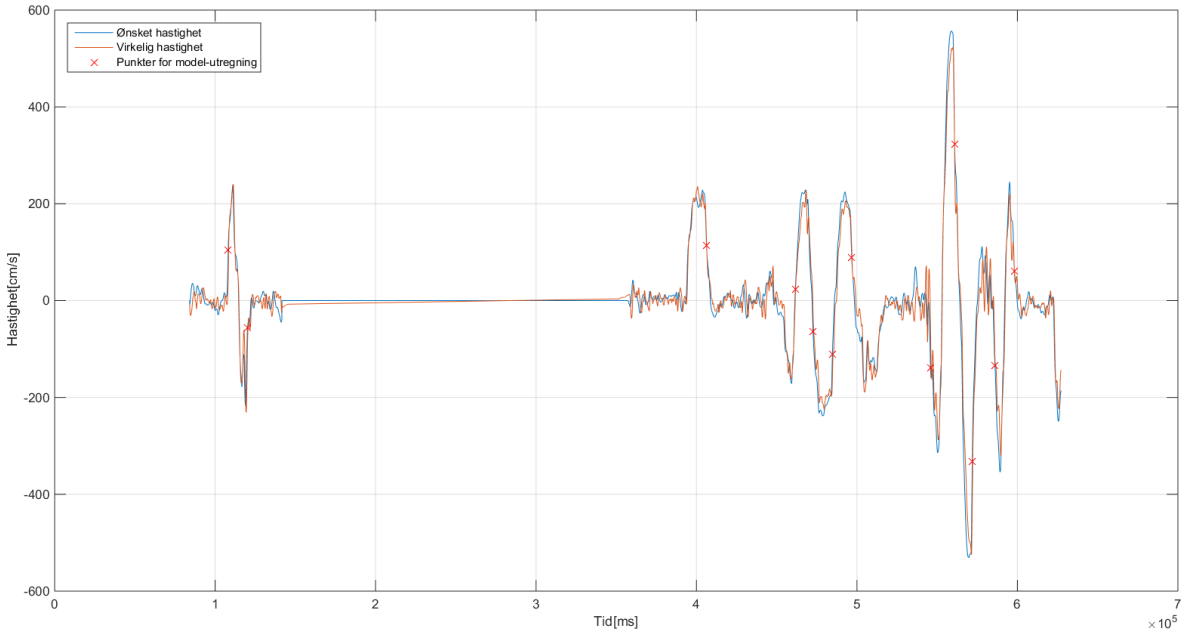


Figur 12: Målt og modifisert hastighet for UAV-en i X-retning.

3.2 Estimering av dynamikk

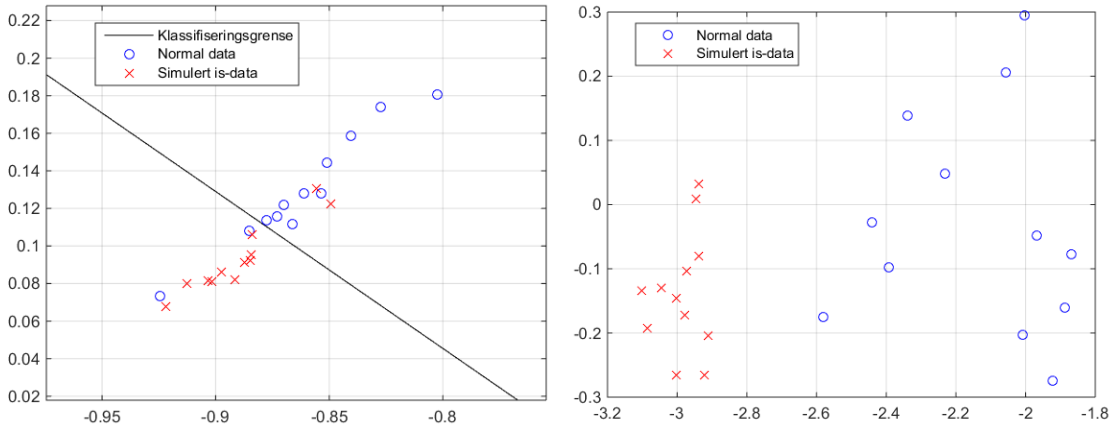
Ettersom det ble antatt at is på propellene ville medføre en endring i dynamikken til farkosten var det naturlig å estimere en ARMAX-modell ved forskjellige tider. Disse ble så sammenlignet for å se om det ble en stor nok forskjell ved den simulerte isingen til at det kunne bli brukt til klassifisering. For å kunne lage en god modell ble det implementert en kode for å identifisere områder der det var et step, slik at modelleringen ikke foregikk i områder der det ikke er store endringer. Dette er vist i fig. 13. Her kommer det fram at store endringer i hastigheten blir identifisert og rundt disse blir det estimert en ARMAX-modell. Det ble plukket ut 100 punkt både før og etter step-identifikasjonen, og ettersom denne målingen blir tatt med 10 Hz tilsvarer dette 20 sekund med data. Totalt ble det identifisert tolv områder der en ARMAX-modell ble estimert.

En tilsvarende prosedyre ble utført på de modifiserte dataene slik at det totalt ble estimert 24 ARMAX-modeller, tolv for den målte dataen og tolv for den modifiserte. Matlab-koden for dette er vist i vedlegg E.



Figur 13: Identifikasjon av step på målt data. Kryss markerer de områdene der en ARMAX-modell blir estimert for systemet.

Det ble forsøkt med litt forskjellige ARMAX-modeller og det ble funnet at ved ved å bruke en ARX-modell med fire poler og fire nullpunkt ble det mulig å separere modellene feilfritt ved å bruke SVM. Dette ga god grunn til å tro at det ville være mulig å få en god separasjon også ved virkelig data. Et plott av to av parametrene i ARX-modellene er vist i fig. 14. Vi ser her at det er mulig å få en lineær separasjon av dataene.



(a) Klassifisering med en pol og ett nullpunkt (b) Klassifisering med fire poler og fire nullpunkt

Figur 14: Klassifisering med forskjellige ARX modeller. I (a) er klassifiseringsgrensen for parametrene vist. I (b) er det vist to av åtte parametre, og det er de parametrene som ga best separasjon som er vist.

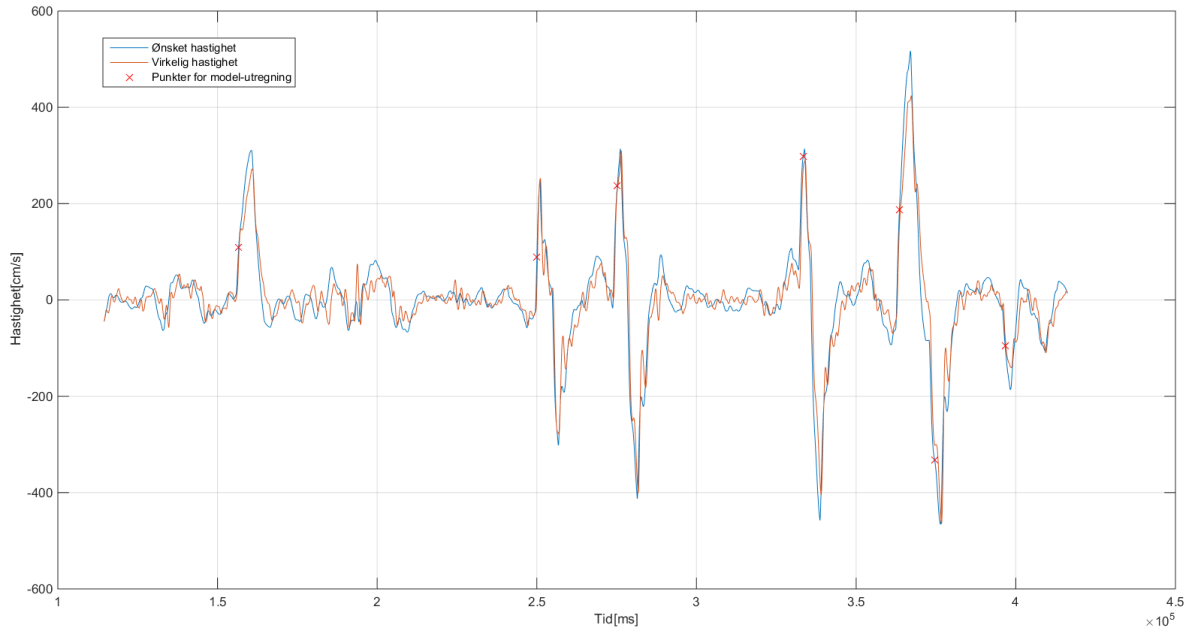
4 Eksperiment med to datasett

For å teste systemet på en mer realistisk måte enn ved å bare anta hvordan oppførselen til UAV-en ville bli ved en feil, ble det også utført en flyging av FFI der det var limt på en forhøyning på alle propellbladene. Dette gjorde at de aerodynamiske egenskapene til UAV-en ble kraftig redusert, noe som også vil være tilfellet ved is på propellene.

I denne delen er det de to datasettene som er blitt logget på UAV-en som blir sammenlignet. Det blir først undersøkt om metoden som ble utviklet i seksjon 3 fremdeles virker, og så blir flere av de loggede parametrene studert for å se om det finnes andre måter å klassifisere på.

4.1 Estimering av dynamikk

For å lage gode ARMAX-modeller av hastigheten til den virkelige dataen begynnes det med å identifisere step i hastigheten. Dette er vist i fig. 15.



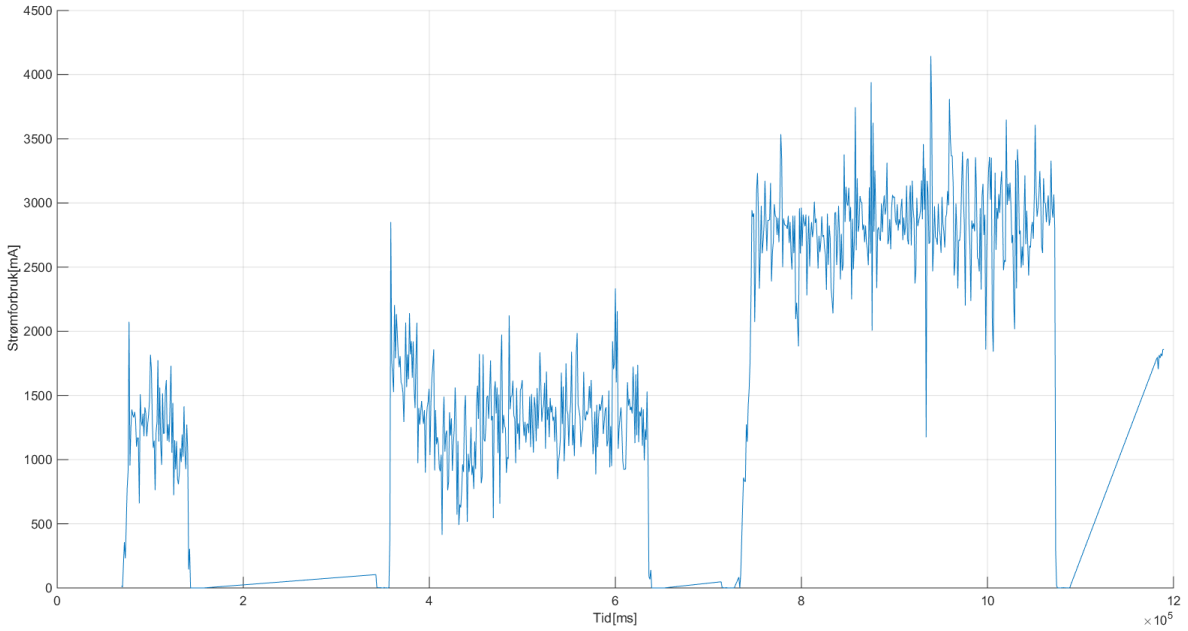
Figur 15: Identifikasjon av step på data med feil på UAV-en. Røde kryss viser områder der en ARMAX-modell ble estimert for systemet.

Ved bruk av ARX-modell med samme størrelsesorden som fungerte bra på simulerte data ble det dessverre ikke funnet noen god separasjon ved bruk av data som var tatt opp med påmontert feil på UAV-en. Det ble forsøkt med en del forskjellige ARMAX-modeller, men det ble ikke funnet en kombinasjon som ga god separasjon av normal-data og is-data.

Av denne grunn blir det antatt at fordi UAV-en er et regulert system, så klarer den å kompensere for den reduserte aerodynamiske effekten på en såpass god måte at det ikke er mulig å se en forskjell i disse dataene. På grunn av dette ble det nødvendig å se etter andre parametrer som endrer seg ved ising på propellene. En grunn til at denne metoden feilet kan også ha noe med den relativt lave loggefrekvensen å gjøre. Dersom disse parametrene ble logget fortere ville det kanskje vært mulig å se en endring i dynamikken.

4.2 Strømforbruk

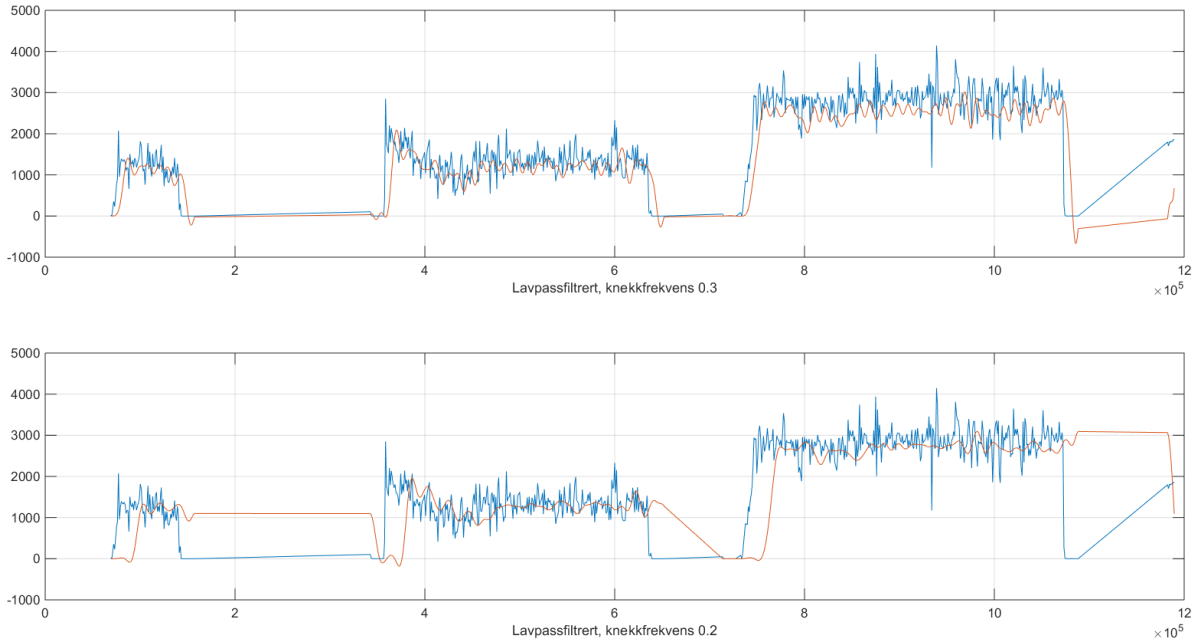
Etter å ha studert måledataene, samt kommunisert med FFI ble det funnet at strømforbruket var den parameteren som endret seg mest ved emulering av is på propellene. Dette er egentlig som forventet ettersom en redusert aerodynamisk ytelse vil føre til at motorene må levere mer kraft. I fig. 16 ser vi strømforbruket til UAV-en. De to flygingene er her blitt satt sammen i ett plott. Fram til omtrent 700 sekunder er det flyging uten feil. Fra 700 sekunder frem til slutt er det flyging med påmonterte forhøyninger på propellene.



Figur 16: Strømforbruk til UAV. Ved omtrent 700 sekund er det blitt påmontert forhøyninger på alle propellene til UAV-en for å etterligne ising.

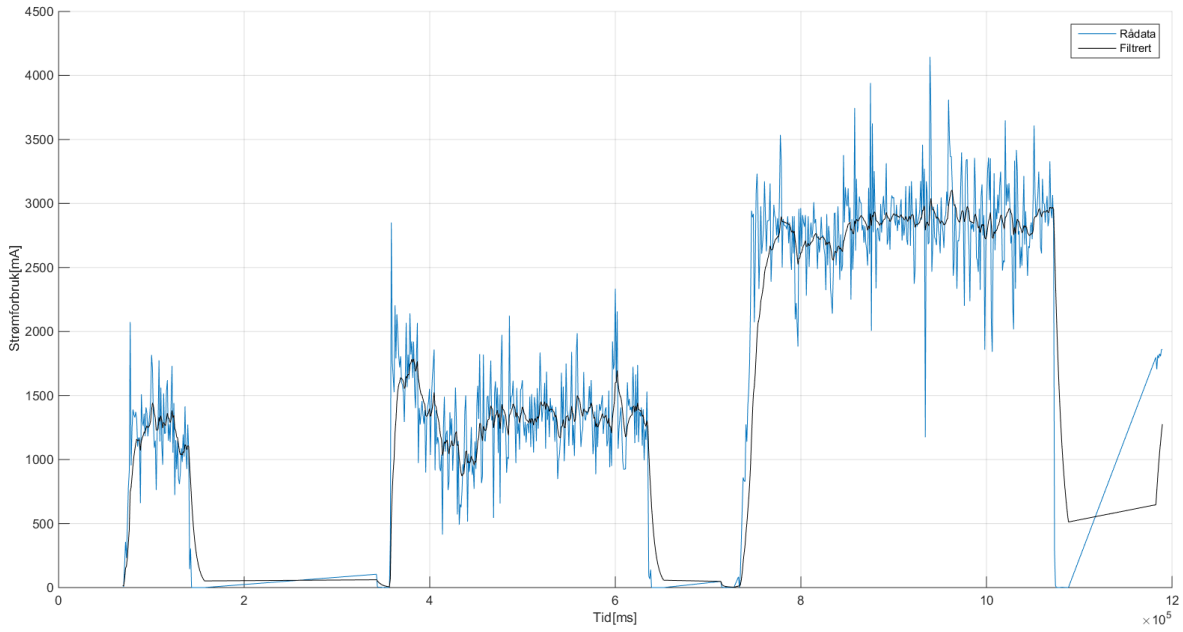
Det kommer tydelig fram av figuren at strømforbruket går vesentlig opp når det er montert forhøyninger på propellene. En enkel klassifiseringsalgoritme kan da være å legge inn en terskelverdi for strømforbruket og si at dersom strømforbruket går over denne så er det en feil på farkosten. Et problem her vil være at det er stor varians i strømforbruket og

det vil dermed bli en del feil-klassifiseringer på grunn av dette. Det er derfor ønskelig å fjerne en del av denne variansen. Det er først blitt forsøkt å bruke et lavpassfilter for å gjøre dette. Resultatet av to lavpassfilter er vist i fig. 17.



Figur 17: Strømforbruk til UAV kjørt gjennom to forskjellige lavpassfilter.

Det er tydelig at å lavpassfiltrere med knekkfrekvens 0.2 (normalisert frekvens) får en betydelig forsinkelse, mens ved bruk av knekkfrekvens 0.3 så oscillerer fremdeles signalet en del samtidig som det også viser et litt for lavt strømforbruk i andre halvdel. Det ble derfor implementert et enkelt Kalman-filter for å fjerne støy. Her ble målestøyen satt til å være variansen i strømforbruket og prosesstøyen ble justert med en faktor på ti fram til resultatet ble seende bra ut. Resultatet er vist i fig. 18.

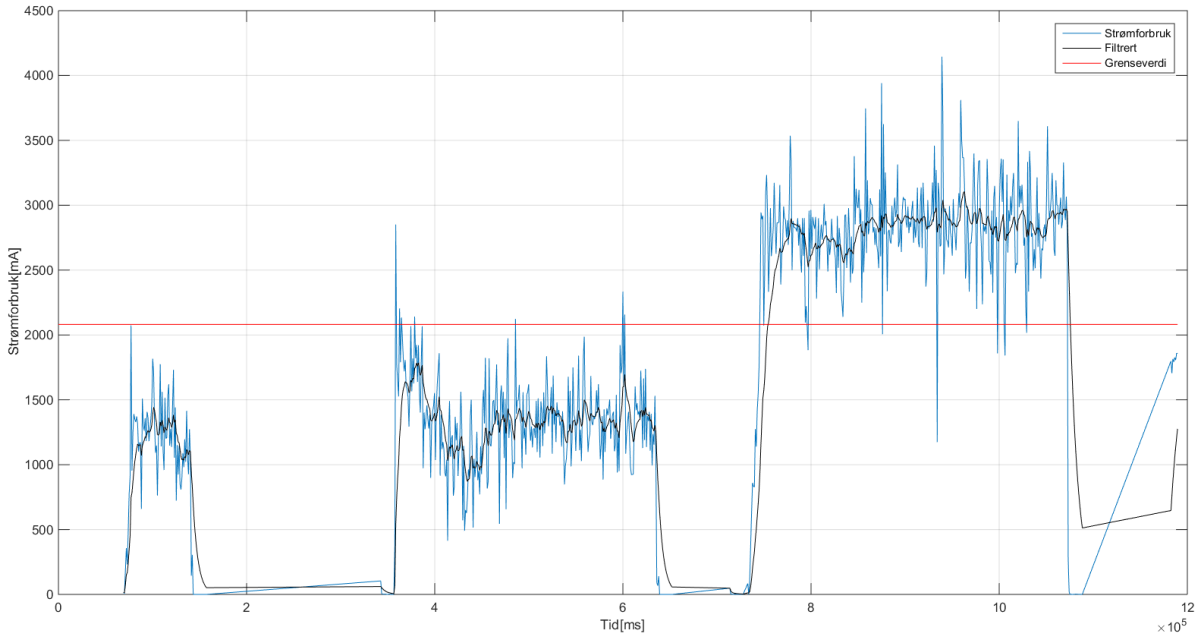


Figur 18: Strømforbruk til UAV kjørt gjennom et Kalman-filter.

Her ser vi at det filtrerte signalet følger hovedtrenden i strømforbruket, samtidig som det fjerner majoriteten av utliggerne. Det ble besluttet å prøve å definere en grenseverdi basert på det filtrerte strømforbruket. Det ble da naturlig å se på variansen og gjennomsnittsverdien til strømforbruket ved normal drift og ved is på propellene. Det ble tatt en antagelse om at denne dataen var normal-distribuert og det ble så regnet ut ved hvilket strømforbruk det var like sannsynlig at det tilhørte både normal drift og unormal drift. Det vil si ved å regne ut følgene ligning med hensyn på x:

$$\frac{1}{\sqrt{2\pi\sigma_1}} e^{-\frac{(x-\mu_1)^2}{2{\sigma_1}^2}} = \frac{1}{\sqrt{2\pi\sigma_2}} e^{-\frac{(x-\mu_2)^2}{2{\sigma_2}^2}}. \tag{4.2.1}$$

X ble funnet til å være omtrent 2080 mA og resulterende klassifisering er vist i fig. 19. Vi ser her at den ufiltrerte dataen klarer å detektere feilen litt tidligere, omtrent åtte sekund, men det er også en del feilklassifiseringer. Den filtrerte dataen har derimot ingen feilklassifiseringer. Totalt tar det omtrent 21 sekunder fra UAV-en begynner å trekke strøm fram til strømforbruket kommer over terskelverdien. Matlab-koden for denne delen er vist i vedlegg F



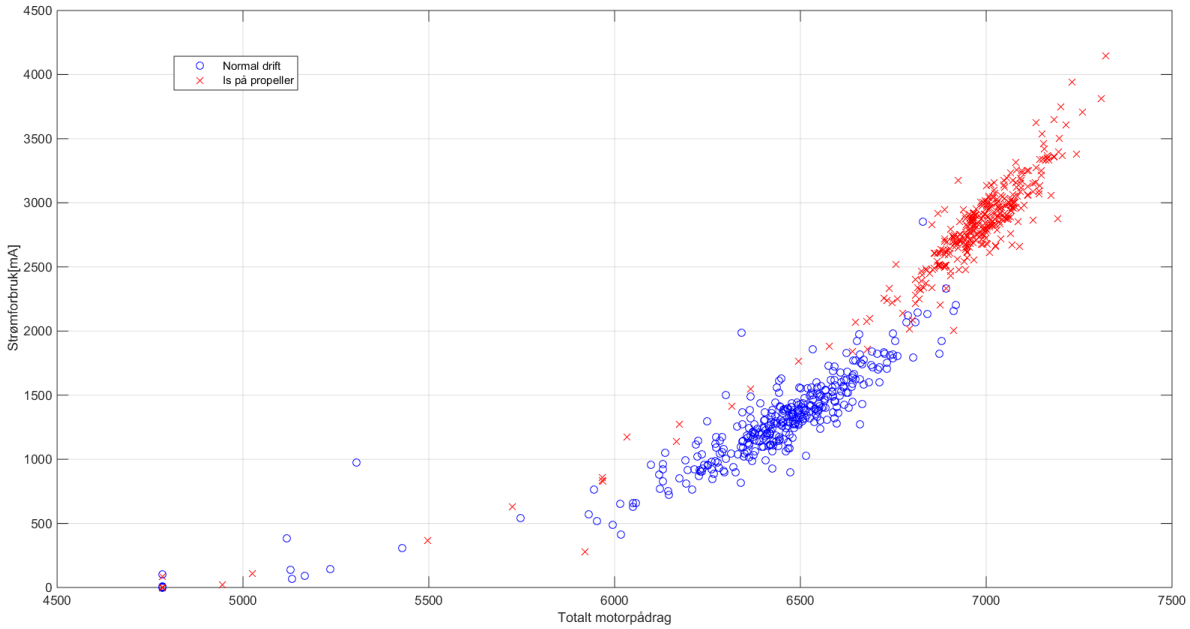
Figur 19: Klassifisering basert på strømforbruk. Figuren viser rådata og data filtrert ved hjelp av et Kalman-filter. Klassifisering basert på det filtrerte signalet gir litt tregere respons, men det fjerner alle feilklassifiseringene som følge av støy i signalet.

En forutsetning for at denne klassifiseringen skal kunne virke er at både farkosten og omstendighetene ved flyging er tilnærmet like. Dersom man for eksempel fester noe på UAV-en eller det er mye vind vil også strømforbruket til farkosten stige og en vil da kunne få feilklassifiseringer. Dette setter en del begrensinger på UAV-en, men det kan tenkes å implementere en funksjon som tillater operatøren å si hvor mye UAV-en er lastet med, og at modellen da tar høye for dette.

Et annet aspekt med denne klassifiseringen er at den er relativt treg, med nesten et halvt minutt før feilen blir klassifisert. Det er allikevel grunn til å tro at algoritmen vil fungere ettersom prosessen med å få is på propellene er en ganske langsom prosess. Det konkluderes derfor med at dette er en algoritme som sannsynligvis vil kunne fungere i praksis.

4.3 Strømforbruk fra motorpådrag

Ettersom hoveddelen av strømforbruket til en UAV er knyttet opp mot motorene, er det av interesse å kunne se på forholdet mellom disse. Et plott av dette er vist i fig. 20.

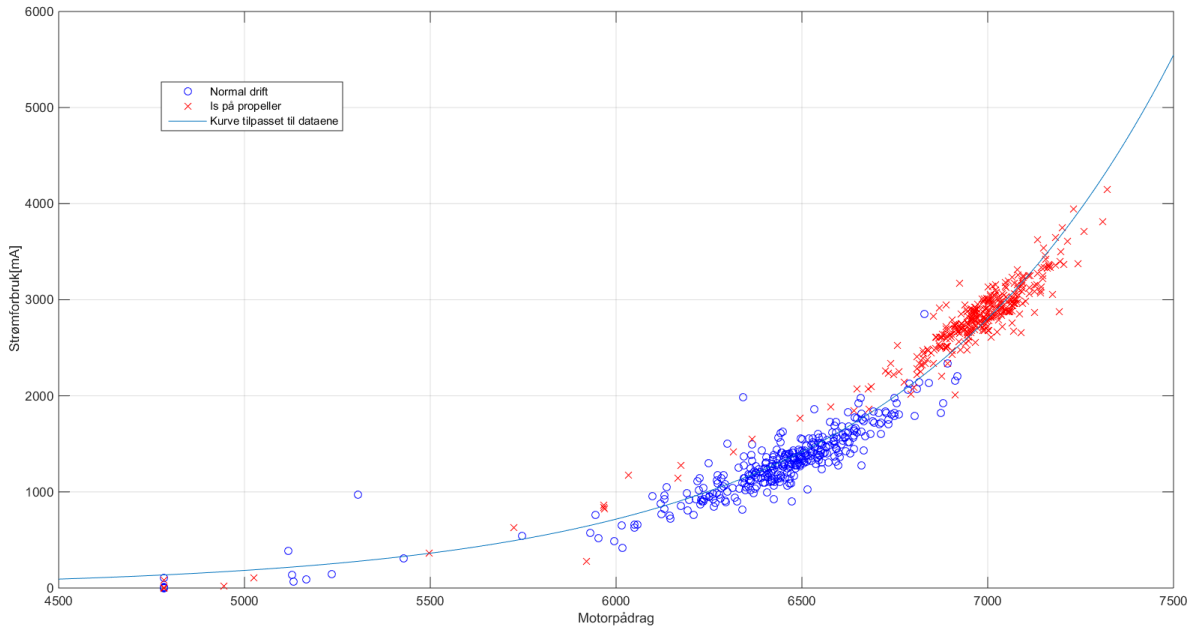


Figur 20: Forhold mellom motorpådrag og strømforbruk. Motorpådraget er summen av Pulse Width Modification(PWM)-en som blir sendt til de fire motorene.

Det kommer her tydelig fram at det ikke er et lineært forhold mellom disse. En annen ting som er viktig her er at strømforbruket bare blir logget med en rate på 1 Hz. Dette betyr at enhver transient som kan eksistere vil forsvinne. Det ble derfor benyttet kurvetilpasningsverktøy i Matlab for å finne et forhold mellom motorpådrag og strømforbruk. Etter å ha prøvd ut noen forskjellige modeller ble det funnet at en eksponentiell funksjon passet bra til dataen. Funksjonen ble funnet til å være:

$$y = 0.2001e^{0.001364x}. \tag{4.3.1}$$

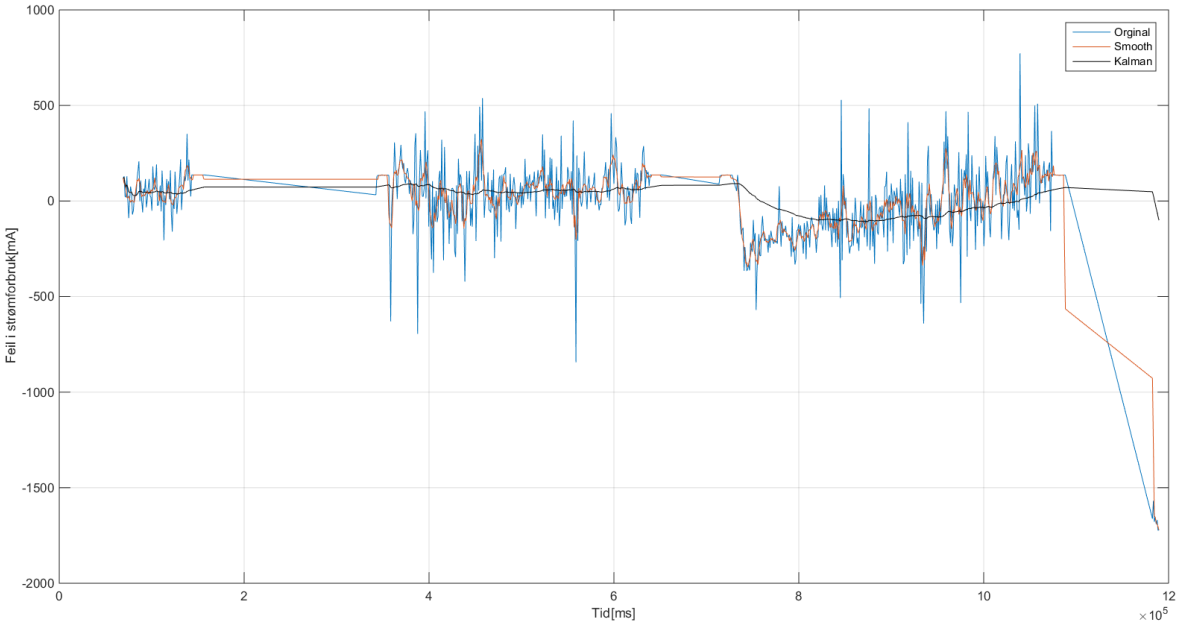
En figur med data og kurve er vist i fig. 21. Vi ser her at den eksponentielle kurven klarer å forklare dataen rimelig bra. Matlab-kode er vist i vedlegg G.



Figur 21: Et plott av strømforbruk gitt av motorpådrag med kurve.

Dette blir videre brukt til å estimere strømforbruk ut fra motorpådrag for så å trekke ifra den faktisk målte verdien. Resultatet av dette er vist i fig. 22. Det kommer fram at forskjellen mellom estimert og målt strømforbruk er omtrent ± 500 mA. Dette resultatet er blitt filtrert, en gang ved matlab-kommandoen *smooth()*, og ved bruk av et enkelt Kalman-filter. Resultatene av dette er også vist i fig. 22. Ved bruk av smooth kommandoen fjernes de verste utliggerne og feilen ligger i størrelsesordenen ± 300 mA. Ved bruk av Kalman-filteret, med ganske kraftig filtrering holder feilen seg stort sett innenfor ± 200 mA, og dersom man bare ser på området med normal drift holder feilen seg mellom 0 og 100 mA.

Dette viser at det bør være mulig å detektere lekkasjestrømmer på UAV-en ettersom man i slike tilfeller vil måle høyere strømtrekk enn det som estimeres. Men det er også tydelig at lekkasjestrømmen må være av en betydelig størrelse før den kan detekteres pålitelig. Men dersom man legger en grense på rundt 250 mA vil en lekkasjestrøm over denne størrelsen kunne bli detektert uten at en vil få noen feilklassifiseringer.



Figur 22: Avvik i strømforbruk. Original viser forskjellen mellom estimert og målt strømforbruk på UAV-en. Dette signalet er så blitt glattet ved å bruke Matlab-funksjonen *smooth()* og ved bruk av at Kalman-filter.

4.4 Strømforbruk vs. akselerasjon

En annen sammenheng som er av interesse er forholdet mellom akselerasjon og strømforbruk. En energibalanse settes derfor opp for UAV-en i Z-retning. Dette gjøres da det antas at hoveddelen av strømforbruket skyldes Z-retningen.

$$\frac{dE_z(t)}{dt} = \sum Q(t). \tag{4.4.1}$$

Energien som en UAV har er til enhver tid gitt ved:

$$E = \frac{1}{2}mv_z(t)^2 + mgh(t). \tag{4.4.2}$$

Når dette deriveres blir det:

$$\frac{dE_z(t)}{dt} = ma_z(t)v_z(t) + mgv_z(t). \tag{4.4.3}$$

Dette substitueres tilbake i energibalansen og vi får:

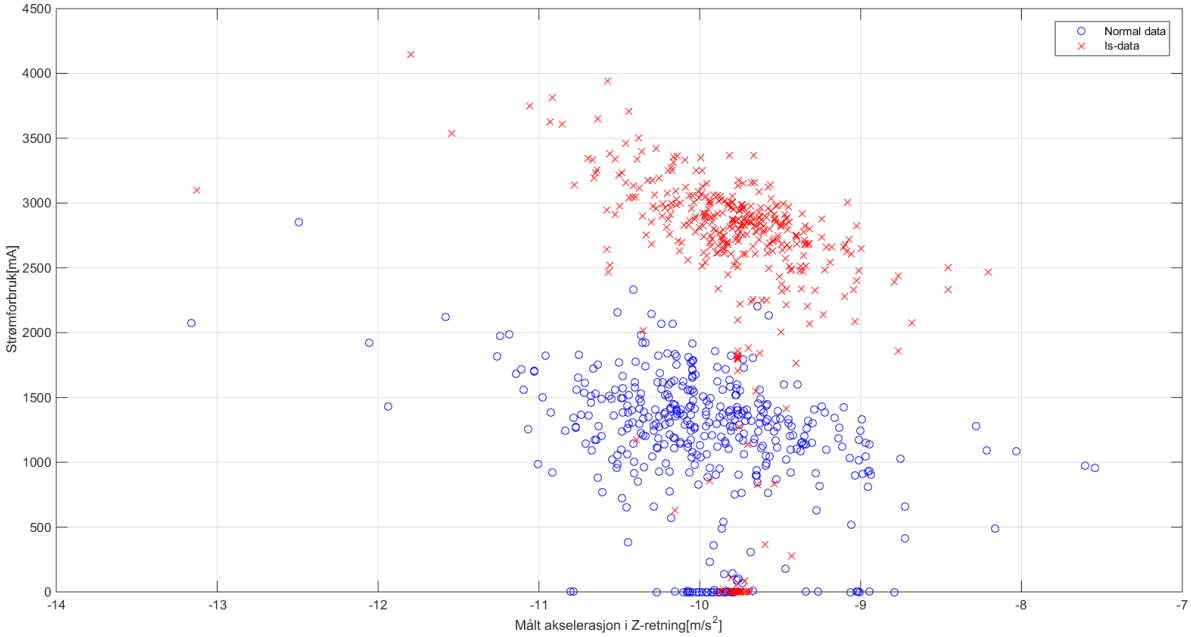
$$ma_z(t)v_z(t) + mgv_z(t) = Q(t). \tag{4.4.4}$$

$$(ma_z(t) + mg)v_z(t) = Q(t). \tag{4.4.5}$$

Der $Q(t)$ er effekt produsert fra de fire propellene. Effekten som blir generert antas å være proporsjonal med strømforbruket.

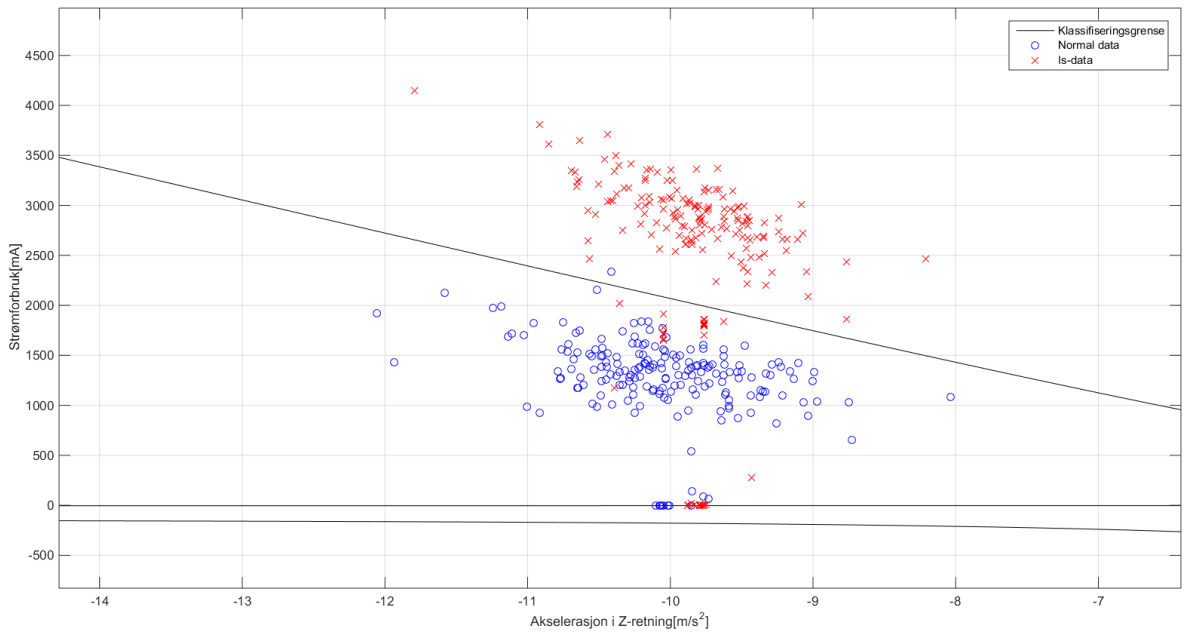
Ettersom mg er konstant antas det da at det bør være et forhold mellom strømforbruk, hastighet i Z-retningen og akselerasjon i Z-retningen.

Ved undersøkelse ble det funnet at det ga bedre separasjon å bare bruke akselerasjonen i Z-retning og forkaste hastigheten. Grunnen til dette kan være at hastigheten til UAV-en er så liten at den blir neglisjerbar i forhold til akselerasjonen. Et plot av akselerasjon i Z-retning mot strømforbruk er vist i fig. 23.



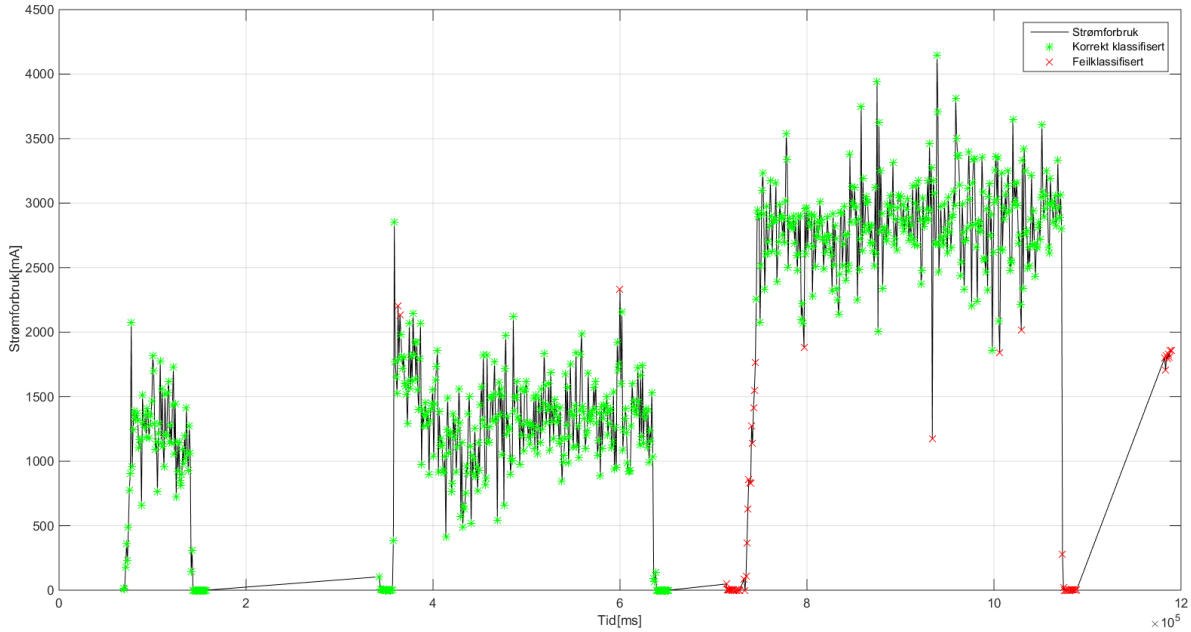
Figur 23: Strømforbruk gitt av akselerasjon i Z-retning.

Det er her mulig å se et forhold mellom akselerasjon og strømforbruk som kan bli brukt for å klassifisere tilstanden til UAV-en. Det velges derfor å bruke SVM og en kjerne som gir en god grense forsøkes å finnes. Etter litt prøving av forskjellige kjerner ble det funnet fram til en polynomisk kjerne som ga god separering. Algoritmen ble trent opp på halvparten av datasettet. Denne gav klassifiseringsgrensen som vist i fig. 24. Matlab-koden som ble brukt i denne delen er vist i vedlegg H.



Figur 24: Klassifisering på grunnlag av strømforbruk og akselerasjon i Z-retning.

Det kommer her fram at det er noen feilklassifiseringer, og feilprosenten er nesten åtte prosent. Ettersom feilprosenten er såpass høy er det interessant å se på hvilke punkter som blir feilklassifisert. Dette er vist i fig. 25. Her kommer det fram at hoveddelen av feilklassifiseringene er i områder som ikke er av stor interesse, som når UAV-en ikke er i drift. Det er allikevel fortsatt et par feilklassifiseringer, men det er ikke mange. En måte å kunne unngå disse på er å si at det må være to eller tre klassifiseringer på rad før klassifiseringen blir godkjent av algoritmen. På denne måten forsvinner alle feilklassifiseringen i de viktige områdene. Ulempen med denne framgangsmåten er selvsagt at det vil ta lengre tid før en reell feil oppdages. Men selv ved å kreve tre like klassifiseringer etter hverandre for å endre tilstand vil denne algoritmen detektere feiltilstanden i dette tilfellet etter omtrent 15 sekunder. Dette betyr at metoden er seks sekunder raskere enn ved bruk av bare strømmen som klassifisering som presentert i seksjon 4.2.

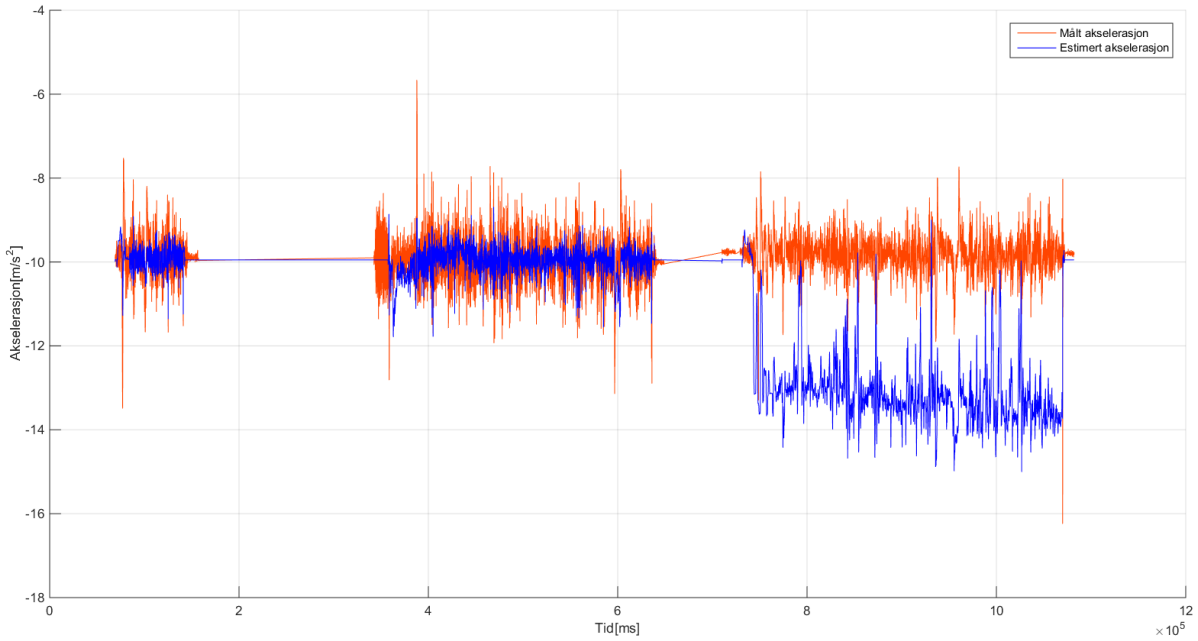


Figur 25: Klassifiserte punkter basert på akselerasjon og strømforbruk. Det er her vist hvilke punkter som ble feilklassifisert på en tidslinje. Det viser seg at hoveddelen av feilklassifiseringene skjer når UAV-en står stille.

4.5 Estimert akselerasjon

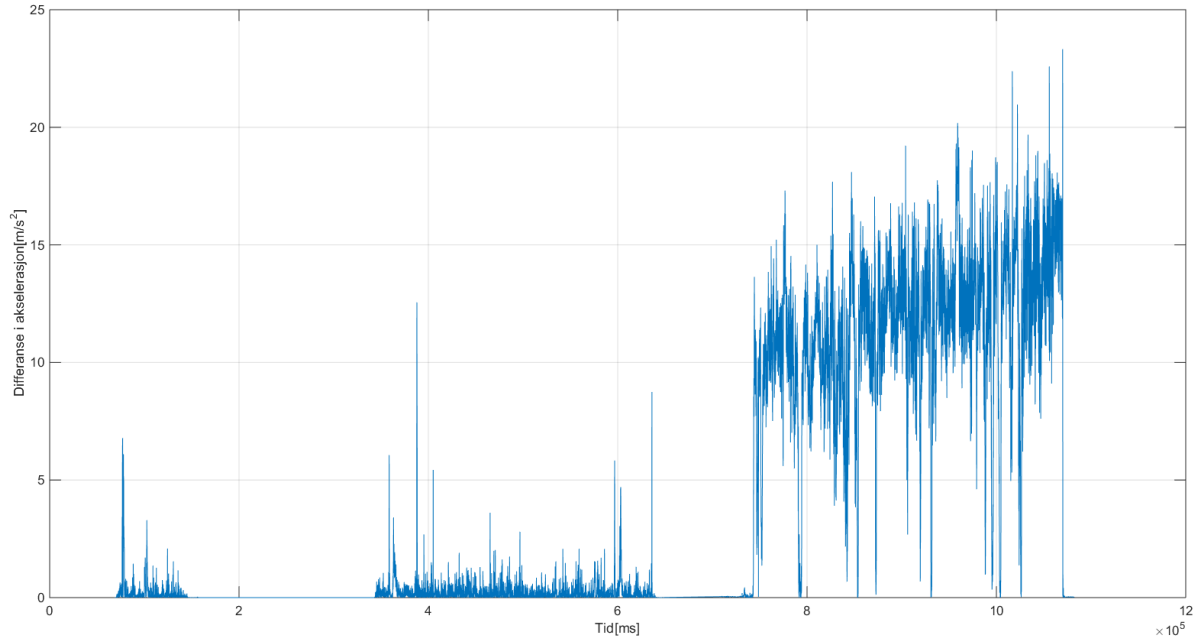
For å kompensere for at det er et ulineært forhold mellom motorpådrag og avgitt effekt fra motorene er det brukt en ulineær ARX modell (NARX) i oppgaven. Det er gjort bruk av Matlabs *nlarx()* kommando. Det er tatt utgangspunkt i resultatet funnet i seksjon 4.4 om at hastigheten i Z-retning blir neglisjerbar og det er derfor forsøkt å utvikle en modell som tar inn motorpådrag og gir ut forventet akselerasjon i Z-retning.

Flere forskjellige modell-størrelser og ulinearitetsfunksjoner ble prøvd ut og det ble til slutt funnet at en modell med to poler, to nullpunkt og en 'sigmoid' ulinearitetsfunksjon ga best resultat. Resulterende estimert og målt akselerasjon er vist i fig. 26. Selv om modellen ikke stemmer helt er det tydelig at det blir et mye større avvik når det blir montert en feil på UAV-en. Koden for dette er vist i vedlegg I.



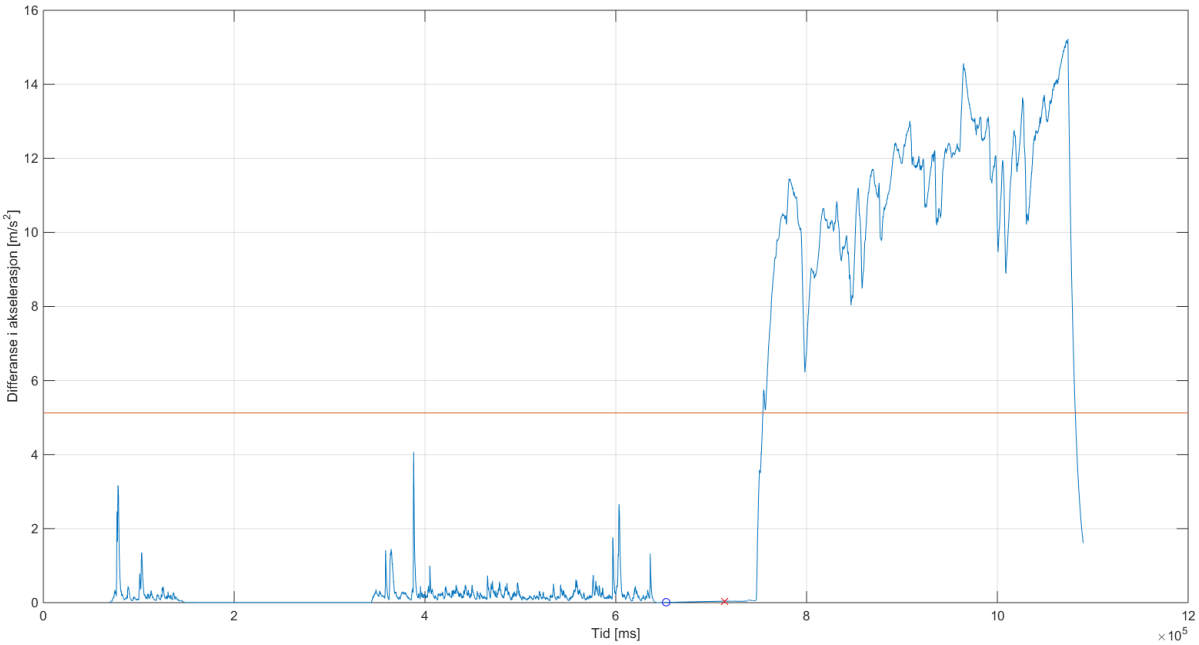
Figur 26: Målt og estimert akselerasjon. Estimeringen stemmer godt når det ikke er feil på UAV-en, men får et stort avvik når det blir påmontert forhøyninger på propellene.

I fig. 27 er det vist kvadratet av residualen mellom estimert og målt akselerasjon. Etter omtrent 700 sekunder med normal flyging kommer datasettet med forhøyninger på propellene. Det kommer da tydelig fram at residualen går vesentlig opp og gjør det mulig å klassifisere dette som unormal flyging.



Figur 27: Residual mellom målt og estimert akselerasjon. Det er tydelig at feilen i estimatet går opp når det er feil på UAV-en.

Det er derimot ikke mulig å gjøre en god klassifisering på disse dataene direkte da det er for mye støy. Signalet blir derfor filtrert med et Kalman-filter og det blir lagt inn en terskel som ligger midt mellom gjennomsnittsverdien til residualen ved normal flyging og ved flyging med feil. Resultatet av dette er vist i fig. 28.



Figur 28: Filtrert residual. Det vises her at det er mulig å klassifisere tilstanden til UAV-en baset på avviket mellom estimert og målt akselerasjon etter filtrering. Terskelverdien ligger midt mellom verdiene ved vanlig drift og drift ved feil.

Fra UAV-en begynner å fly med påmontert feil til den detekteres tar det omtrent seks sekunder. Dette er med andre ord den metoden som fungerer raskest av de som er prøvd ut i denne oppgaven. Den er 15 sekunder raskere enn ved å se på strømmen direkte og den er også ni sekunder raskere enn ved å prøve å klassifisere basert på strømforbruk og akselerasjon. En av grunnene til at denne fungerer bedre kan være at den ikke er begrenset av den trege loggehastigheten til strømmen som bare er 1 Hz. Den laveste loggehastigheten i denne metoden er knyttet til motorpådragene. Denne er på 10 Hz og er altså ti ganger raskere enn strømmen.

Denne metoden har omtrent de samme svakheten som å se på strømforbruket direkte. Dersom man fester noe til farkosten vil dette føre til at motorpådraget økes og man får da feilklassifiseringer. Styrken i forhold til å se på strømmen direkte er at det gir raskere klassifisering.

5 Videre arbeid

Et naturlig første steg i videre arbeid på denne oppgaven er å implementere de introduserte funksjonene på quadcopteret og gi tilbakemelding mens man flyr. En vil da kunne utføre mer omfattende testing av algoritmene for å se om de gir gode resultat ved forskjellige omstendigheter. Ettersom det her hovedsakelig er blitt sett på ising på propellene vil det være interessant å prøve dette i virkeligheten og se hvor lang tid og hvor mye is som må være på propellene før algoritmen klarer å detektere det. Ettersom poenget med algoritmen er å gi en tidlig advarsel er det nødvendig at det klarer å detektere isen før UAV-en bryter sammen.

Det er også av interesse å se hvilken av de introduserte algoritmene som virker best, enten den som ble introdusert i seksjon 4.2, 4.4 eller 4.5. Den sistnevnte gav best resultat på de dataene som var tilgjengelig i denne oppgaven, men hvilke av metodene som vil gi best generalisering og dermed best resultat på nye datasett må det gjøres nye forsøk for å avgjøre.

Andre muligheter for videre arbeid vil være å utvikle algoritmer for å detektere andre feilkilder. Dette kan for eksempel være å overvåke spenningsnivået eller det totale strømforbruket til UAVen og rapportere når batteriet begynner å bli tomt. Dette kan også knyttes opp mot hvor langt unna UAV-en er og dermed gi en form for 'point of no return' indikasjon. Dersom man for eksempel vet at UAV-en bruker 3000mA og batteriet inneholder 1500mAh vet man at det er mulig å fly 30 minutter. Dersom man da i tillegg vet hva den maksimale hastigheten til UAV-en er vil det være mulig å regne ut et punkt der det ikke lengre vil være mulig for UAV-en å returnere til start-punktet.

En annen feilkilde som kan være av interesse er en total stopp av en enkel motor/propell. En kontroller for da å styre quadcopteret som et tricopter er introdusert i [10]. Arbeidet vil da bestå av å klare å identifisere hvilken motor som er skadet for så å regulere UAV-en i forhold til det. En mulig plass å begynne med identifiseringen av dette er å se på motorpådraget direkte. Dersom en motor får fullt pådrag hele tiden, vil det indikere at noe er feil med denne motoren/propellen. Det vil da kanskje være aktuelt å ta den ut av drift på en kontrollert måte.

Ettersom basestasjonen til UAV-en typisk er en datamaskin, kan det også være av interesse å lage en Graphical User Interface (GUI) som forteller operatøren om det er feil på farkosten og hvilken type feil det er. Det kan da velges om rådata skal sendes til dataen for så å klassifisere den der, eller om det er bedre å klassifisere ombord på UAV-en og sende resultatene av dette til GUI-en, som da bare blir en måte å visualisere resultatene

på.

Et mer omfattende prosjekt kan være å gå mer i retningen av System Health Management (SHM) og faktisk gi UAV-en muligheten til selv å avgjøre om den skal fortsette med oppdraget, returnere til start eller til og med foreta en nødlanding.

Å se på problemstillingen ut fra matematisk modellering fra fysikken vil også kunne være en mulighet dersom det blir utført eksperiment for å kartlegge hvor mye kraft og moment hver motor genererer ved forskjellige pådrag. Man kan så ta utgangspunkt i formlene som ble presentert i seksjon 2.5.1. Ved å gjøre dette og dermed få et estimat for roll, pitch og yaw, som man kan sammenligne med de målte verdiene, vil det kanskje være mulig å gi mer detaljert informasjon om hvor en eventuell feil er.

En videreutvikling av den matematiske modelleringen vil være å lage et program som tar inn teknisk informasjon om UAV-en og dermed gjør automatisk endringer i ligningene slik at de estimerte verdiene passer til den valgte UAV-en. Dette vil gjøre det enklere å skifte UAV og dermed redusere fremtidig arbeid i forbindelse med tilstandsovervåking. Et av hovedmomentene her vil være å lage en oversikt over hvilken informasjon som er nødvendig for å få et slikt program til å virke.

6 Konklusjon

Hovedmålet med oppgaven har vært å utvikle en metode for å detektere og identifisere en feil på en autonom UAV med det formålet at operatøren av UAV-en skal slippe å bruke mye tid på å finne ut om det er feil på farkosten.

En av hovedutfordringene med oppgaven har vært det faktum at UAV-en er utstyrt med en autopilot som kompenserer for de fleste feil. På grunn av usikkerhet på hvordan denne er implementert i autopiloten har det ikke blitt laget en matematisk modell av UAV-en ut fra fysikk, noe som har gjort at oppgaven har vært basert på å lage modeller basert på data som har blitt logget.

En annen utfordring har vært loggefrekvensen som er blitt brukt på de forskjellige parametrene. Denne har variert en del på forskjellige parametre, der for eksempel strømforbruk logges med 1 Hz, mens hastighet logges med 10 Hz og akselerasjon logges med 50 Hz.

Det ble først brukt simulert data der det var tatt antagelser om hva som kom til å skje med farkosten ved is på propellene. Forsøk på å klassifisere disse dataene ga gode resultater, men da virkelige data ble gjort tilgjengelig viste det seg at regulatoren i autopiloten var mer effektiv en først antatt og de metodene som ble utviklet med simulert data måtte forkastes da de ikke ga noen brukbare resultater.

Det ble funnet at de eneste variablene som gav et pålitelig utslag var strømforbruket og motorpådraget som naturlig nok ble høyere når det var montert forhøyninger på propellene. Etter at disse variablene var identifisert ble det utviklet metoder for å skille det normale datasettet fra det med forhøyninger.

Den første metoden som ble utviklet var å bruke et Kalman-filter på strømforbruket for å bli kvitt støy. Det ble så brukt en Maximum Likelihood (ML) metode for å beregne en terskelverdi for hva som er normal drift og hva som er unormal. Dette ga god klassifisering mellom normal og unormal drift av UAV-en. Problemet her var at algoritmen var noe treg til å detektere feil på UAV-en.

Det ble så laget en modell som klarer å forutsi hvor mye strøm farkosten bør trekke basert på motorpådragene. Ved å se på differansen på dette estimatet og faktisk målt strømforbruk vil det være mulig å si om det er noe feil på motorer eller om det er lekkasjestrømmer andre plasser på farkosten. Disse lekkasjestrømmene må likevel være av en viss størrelse da differansen bør forventes å være minst 100mA. Men på en farkost som vanligvis trekker rundt 1500mA bør dette kunne gi en grei advarsel om unaturlige strømmer.

Videre ble det implementert en annen klassifiseringsalgoritme som ser på forholdet mellom strømforbruk og akselerasjon i Z-retning. Det ble vist en sammenheng mellom at dersom farkosten akselererer oppover så er også strømforbruket høyere og når farkosten akselererer nedover er strømforbruket tilsvarende lavere. Det ble her noen feilklassifiseringer, men det er en metode som kan gi bedre generelle resultater enn ved bruk av bare strømmen for å klassifisere. Denne metoden var i tillegg noe raskere enn ved å bruke bare strømmen for å klassifisere dataen. Det ble ikke funnet et forhold mellom strømforbruket og hastighet i Z-retning selv om dette var forventet. Ettersom det var en del forsinkelser i forbindelse med å få tak i data har det ikke vært tid til å undersøke dette innenfor rammene til denne oppgaven.

Til slutt ble det laget en ulineær modell for å modellere akselerasjon i Z-retning basert på motorpådrag. Modellen som ble funnet gir ikke identisk resultat som den målte akselerasjonen, men det viktige er at den gir et stort avvik når det var montert forhøyninger på propellene til UAV-en. Det ble dermed mulig å legge inn en terskelverdi for å klassifisere flygingen som normal eller unormal. Dette er den metoden som gir raskest utslag etter at det ble monter forhøyninger på propellene. Det antas at dette kan være på grunn av at det her ikke gjøres bruk av strømmålinger som i de andre metodene. Dette er fordi strømforbruket er den parameteren som logges treges, ved bare 1 Hz, og klassifiseringsalgoritmer som bruker denne parameteren blir derfor tregere.

Det er dermed vist at selv om UAV-en er et regulert system, så er det mulig å finne eksisterende parametrer som endrer seg i stor nok grad ved feil på farkosten slik at de kan bli brukt til å detektere disse feilene.

Referanser

[1] ArduPilot, "ArduPilot." [Online]. Available: http://ardupilot.com/

[2] C. Balas, "Modelling and Linear Control of a Quadrotor," Cranfield University, Tech. Rep., 2007.

[3] Arduino, "Arduino." [Online]. Available: http://www.arduino.cc/

[4] R. Isermann, *Fault-Diagnosis Applications*. New York: Springer, 2011.

[5] G. Vachtsevanos, F. L. Lewis, M. Roemer, A. Hess, and B. Wu, *Intelligent Fault Diagnosis and Prognosis for Engineering Systems*. New Jersey: Wiley, 2006.

[6] G. Heredia, A. Ollero, M. Bejar, and R. Mahtani, "Sensor and actuator fault detection in small autonomous helicopters," *Mechatronics*, vol. 18, no. 2, pp. 90–99, 2008.

[7] A. Ruangwiset and B. Suwantragul, "Wind Tunnel Test of UAV Fault Detection Using Principal Component Based Aerodynamic Model," in *IEEE International Conference on Mechatronics and Automation*, 2008, pp. 150–154.

[8] P. Freeman, R. Pandita, N. Srivastava, and G. J. Balas, "Model-Based and Data-Driven Fault Detection Performance for a Small UAV," *IEEE/ASME Transactions on mechatronics*, vol. 18, no. 4, pp. 1300–1309, 2013.

[9] S. B. Johnson, T. J. Gormley, S. S. Kessler, C. D. Mott, A. Patterson-Hine, K. M. Reichard, and P. A. Scandura, Eds., *System Health Management*. Wiley, 2011.

[10] A.-R. Merheb, H. Noura, and F. Bateman, "A Novel Emergency Controller for Quadrotor UAVs," in *IEEE Conference on Control Applications*, 2014, pp. 747–752.

[11] M. G. Pecht, *Prognostics and Health Management of Electronics*. New Jersey: Wiley, 2008.

[12] D. Fragkoulis, G. Roux, and B. Dahhou, "Detection, isolation and identification of multiple actuator and sensor faults in nonlinear dynamic systems: Application to a waste water treatment process," *Applied Mathematical Modelling*, vol. 35, no. 1, pp. 522–543, 2011.

[13] R. Razavi-Far, H. Davilu, V. Palade, and C. Lucas, "Model-based fault detection and isolation of a steam generator using neuro-fuzzy networks," *Neurocomputing*, vol. 72, no. 13-15, pp. 2939–2951, 2009.

[14] Z. Li and B. Dahhou, "A new fault isolation and identification method for nonlinear dynamic systems: Application to a fermentation process," *Applied Mathematical Modelling*, vol. 32, no. 12, pp. 2806–2830, 2008.

[15] K. Skretting, "ELE620 Systemidentifikasjon," Stavanger, 2014.

[16] O. y. Magnussen and K. E. Skjø nhaug, "Modeling, Design and Experimental Study for a Quadcopter System Construction," University of Agder, Master, 2011. [Online]. Available: http://brage.bibsys.no/xmlui/handle/11250/136686

[17] Y. Zhu, "Estimation of Nonlinear ARX Models," in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, 2002, pp. 2214–2219.

[18] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. New York: Wiley, 2001.

[19] N. Cristianini and J. Shawe-Taylor, *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge: Cambride University Press, 2003.

[20] MIT OpenCourseWare, "Lecture 16: Learning: Support Vector Machines," 2010. [Online]. Available: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-16-learning-support-vector-machines/

A inspect.m

```
% Start by clearing all
clear all; close all; clc;
% Load the log from mission planner.
% Some data at the end is removed as it is not part of the flight.
load('2015-03-25 13-08-11.log-49814.mat');

% Sample rate 10 Hz
NTUN = NTUN(1:end-200,:);
DVelX = NTUN(:,7);
VelX = NTUN(:,9);
TimeNTUN = NTUN(:,2);

% Sample rate 50 Hz.
IMU = IMU(1:end-200,:);
AccZ = IMU(:,8);
TimeIMU = IMU(:,2);

% Sample rate 10Hz
Mot1 = RCOU(:,3);
Mot2 = RCOU(:,4);
Mot3 = RCOU(:,5);
Mot4 = RCOU(:,6);
TimeRCOU = RCOU(:,2);

% Sample rate 1 Hz.
CURR = CURR(1:end-8,:);
Curr = CURR(:,6);
TimeCURR = CURR(:,2);

% Data with icing problem
load('2015-04-09 09-59-44.log-46132.mat');

% Sample rate 10Hz
NTUN = NTUN(1:end-200,:);
DVelX_Ice = NTUN(:,7);
VelX_Ice = NTUN(:,9);
TimeNTUN_Ice = NTUN(:,2);

% Sample rate 50 Hz.
IMU = IMU(1:end-200,:); % Remove bad data
AccZ_Ice = IMU(:,8);
TimeIMU_Ice = IMU(:,2);

% Sample rate 10Hz
Mot1_Ice = RCOU(:,3);
Mot2_Ice = RCOU(:,4);
Mot3_Ice = RCOU(:,5);
Mot4_Ice = RCOU(:,6);
TimeRCOU_Ice = RCOU(:,2);

% Sample rate 1 Hz.
Curr_Ice = CURR(:,6);
TimeCURR_Ice = CURR(:,2);
```

B SVM.m

```
function [classified] = SVM( TrainData, ClassifiedData, ClassifyData ,type,
    scale, fraction)
%A implementation of a SVM classifier using the built in matlab fitcsvm
%method
%   Input:
%   - TrainData: One matrix containing training data
%   - ClassifiedData: one vector which classifies the training data, 1 is
    good, -1 is bad
%   - ClassifyData: one matrix containing data to be classified
%   - type: What kernel to use
%   - fraction: Percentage of sample which is expected to be outliers.
%   - size: Scale of the kernel
%   Output:
%   - classified: A vector containing the classification of the input
%   - ~ : A plot containing the decision boundary

% Train the classifier.
cl = fitcsvm(TrainData',ClassifiedData,'KernelFunction',type,...
    'BoxConstraint',Inf,'ClassNames',[-1,1],'OutlierFraction', fraction,...
    'KernelScale', scale);

% Find dimensions of points to be classified.
[dim,numSamples] = size(TrainData);

% Classify the new points.
[classified,~] = predict(cl,ClassifyData');

if (dim == 2)

    Data = [TrainData, ClassifyData];

    % Decide on the size of the grid.
    l = max(Data(1,:)) - min(Data(1,:));
    b = max(Data(2,:)) - min(Data(2,:));
    d = min(l,b) / 5;

    [x1Grid,x2Grid] = meshgrid(min(Data(1,:))-l*0.2:d:max(Data(1,:))+l*0.2,
        ...
        min(Data(2,:))-0.2*b:d:max(Data(2,:))+0.2*b);

    xGrid = [x1Grid(:),x2Grid(:)];
    [~,scores] = predict(cl,xGrid);

    % plot the decision boundary.
    figure;
    contour(x1Grid,x2Grid,reshape(scores(:,2),size(x1Grid)),[0 0],'k');
    hold on;

end
end
```

C IDstep.m

```
function [ A, B, points ] = ID_step( Desired, Real, Time, na, nb )
%A function which identifes steps in a signal and then calculates a
%ARX-model at this point
%   Input:
%   - Desired: Input signal(typ. u)
%   - Real: Output signal(typ. y)
%   - Time: Timestamp for the signals
%   - na: Size of the a vector to be calculated
%   - nb: Size of the b vector to be calculated
%   Output:
%   - A: A matrix containing the a vectors for each step
%   - B: A matrix containing the b vectors for each step
%   - points: A matrix containing the points at which a step was found.

nc = 0;
nk = 0;
num = 1;
Ts = 0.1; % Sample time is 100 ms, 10 Hz,
count = 1;
thres = 75;
dist = 100; % Choose 200 data points to estimate model, that is 20 seconds.
step = 10;

while ( count < length(Time)-100)
    count = count + step;

    if(abs(Desired(count)-Desired(count-step)) > thres) % in order to be
        sure that a step is present.
        tic
        points(num,:) = [count,Desired(count)];
        lower = count-dist;
        upper = count + dist;
        data1 = iddata(Real(lower:upper),Desired(lower:upper),Ts);
        sys1 = armax(data1,[na,nb,nc,nk]); % Maybe just use arx
        A(:,num) = sys1.a;
        B(:,num) = sys1.b;
        num = num + 1;
        count = count + dist;
        toc;
    end
end
end
```

D Kalmanfilter.m

```
function [ x_hat ] = Kalman_filter( y, R, Q )
%A simple implementation of a Kalman filter
%   Input:
%   - y: Signal to be filtered
%   - R: Measurement noise
%   - Q: Process noise
%   Output:
%   - x_hat: The filtered signal

phi = 1;
D = 1;
x_hat(1) = y(1);
P_hat(1) = R;

for i=2:length(y)
    x_line(i) =  phi*x_hat(i-1);
    P_line(i) = phi*P_hat(i-1)*phi' + Q;
    K(i) = P_line(i)*D'/(D*P_line(i)*D' + R);
    x_hat(i) = x_line(i) + K(i)*(y(i)-D*x_line(i));
    P_hat(i) = (1-K(i)*D)*P_line(i);
end
end
```

E Velocity.m

```
%% Start by setting some constants
na = 1;
nb = 1;

%% Process VelX to create simulated data.
d = fdesign.lowpass('Fp,Fst,Ap,Ast',0.5,0.6,1,60); % Lowpass filter
D = design(d,'iir');
VelX_Sim = filter(D, VelX)*0.99; %Apply lowpass to velocity in X direction.

figure;
plot(TimeNTUN, VelX_Sim)
hold on
plot(TimeNTUN, VelX)
grid;
title('Simulert vs. normal hastighet i x-retning');
legend('Hastighet med simulert ising', 'Normal hastighet');

%% Identify step in command to compute armax model, normal data
[A1, B1, points1] = ID_step(DVelX, VelX, TimeNTUN, na, nb);
figure;
plot(TimeNTUN, DVelX);
hold on;
plot(TimeNTUN, VelX);
title('Hastighet i x-retning, normal flygning');
grid;
plot(TimeNTUN(points1(:,1)), points1(:,2), 'xr');
legend('Ønsket hastighet', 'Virkelig hastighet', 'Punkter for model-
    utregning', 'Location', 'northwest');

%% Identify model in step response, simulated data
[A2, B2, points2] = ID_step(DVelX, VelX_Sim, TimeNTUN, na, nb);
figure;
plot(TimeNTUN, DVelX);
hold on;
plot(TimeNTUN, VelX_Sim);
title('Hastighet i x retning, simulert ising på propeller');
grid;
plot(TimeNTUN(points2(:,1)), points2(:,2), 'xr');
legend('Ønsket hastighet', 'Simulert hastighet', 'Punkter for model-
    utregning', 'Location', 'northwest');

%% Identify model in step response, problem data
[A3, B3, points3] = ID_step(DVelX_Ice, VelX_Ice, TimeNTUN_Ice, na, nb);
figure;
plot(TimeNTUN_Ice, DVelX_Ice);
hold on;
plot(TimeNTUN_Ice, VelX_Ice);
title('Hastighet i x retning, med emulert ising på propeller');
grid;
plot(TimeNTUN_Ice(points3(:,1)), points3(:,2), 'xr');
legend('Ønsket hastighet', 'Virkelig hastighet', 'Punkter for model-
    utregning', 'Location', 'best');
```

```
%% Divide the data into training data and test data.
% Remove Ax(1,1:end) because this is alway 1 -> no information

TrainData_Sim = [A1(2:end,1:ceil(length(A1)/2)), A2(2:end,1:ceil(length(A2)
    /2));
    B1(1:end,1:ceil(length(B1)/2)), B2(1:end,1:ceil(length(B2)/2))];
VerData_Sim = [A1(2:end,1+ceil(length(A1)/2):end), A2(2:end,1+ceil(length(
    A2)/2):end);
    B1(1:end,1+ceil(length(B1)/2):end), B2(1:end,1+ceil(length(B2)/2):end)
        ];
TrainData_Real = [A1(2:end,1:ceil(length(A1)/2)), A3(2:end,1:ceil(length(A3
    )/2));
    B1(1:end,1:ceil(length(B1)/2)), B3(1:end,1:ceil(length(B3)/2))];
VerData_Real = [A1(2:end,1+ceil(length(A1)/2):end), A3(2:end,1+ceil(length(
    A3)/2):end);
    B1(1:end,1+ceil(length(B1)/2):end), B3(1:end,1+ceil(length(B3)/2):end)
        ];

ClassifiedData_Sim = [ones(1,ceil(length(A1)/2)),-ones(1,ceil(length(A2)/2)
    )];
ClassifiedData_Real = [ones(1,ceil(length(A1)/2)),-ones(1,ceil(length(A3)
    /2))];

%% Classify simulated data using SVM
class_Sim = SVM(TrainData_Sim,ClassifiedData_Sim,VerData_Sim,'linear','auto
    ', 0);

if(na+nb == 2)
    plot(A1(2,:),B1(1,:),'bo');
    hold on;
    plot(A2(2,:),B2(1,:),'xr');
    legend('Klassifiseringsgrense','Normal data', 'Simulert is-data', '
        Location', 'northwest');
    title('SVM klassifisering');
    grid;
end

NumMiss_Sim = sum(class_Sim(1:floor(length(A1)/2)) ~= 1) + sum(class_Sim(1+
    floor(length(A1)/2):end) ~= -1)

%% Classify real data using SVM
class_Real = SVM(TrainData_Real,ClassifiedData_Real,VerData_Real,'rbf','
    auto', 0);

if(na+nb == 2)
    plot(A1(2,:),B1(1,:),'bo');
    hold on;
    plot(A3(2,:),B3(1,:),'xr');
    legend('Klassifiseringsgrense','Normal data', 'Emulert is-data');
    title('SVM klassifisering');
    grid;
end

NumMiss_Real = sum(class_Real(1:floor(length(A1)/2)) ~= 1) + sum(class_Real
    (1+floor(length(A1)/2):end) ~= -1)
```

F Current.m

```
%% Use Kalman to filter current being drawn
Curr_Kalman = Kalman_filter(Curr, var(Curr),10000);
Curr_Ice_Kalman = Kalman_filter(Curr_Ice, var(Curr_Ice), 10000);
figure;
hold on;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr;Curr_Ice]);
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr_Kalman,Curr_Ice_Kalman],'k
    ');
legend('Rådata', 'Filtrert')
grid;

%% Test using low-pass filters.
d1 = fdesign.lowpass('Fp,Fst,Ap,Ast',0.2,0.3,1,60); % Lowpass filter
D1 = design(d1,'iir');
d2 = fdesign.lowpass('Fp,Fst,Ap,Ast',0.1,0.2,1,60); % Lowpass filter
D2 = design(d2,'iir');
Curr_LP1 = filter(D1, Curr); % Apply lowpass
Curr_Ice_LP1 = filter(D1,Curr_Ice);
Curr_LP2 = filter(D2, Curr); % Apply lowpass
Curr_Ice_LP2 = filter(D2,Curr_Ice);
figure;
subplot(2,1,1)
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr;Curr_Ice]);
hold on;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr_LP1;Curr_Ice_LP1]);
xlabel('Lavpassfiltrert, knekkfrekvens 0.3')
grid;
subplot(2,1,2)
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr;Curr_Ice]);
hold on;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr_LP2;Curr_Ice_LP2]);
xlabel('Lavpassfiltrert, knekkfrekvens 0.2')
grid;

%% Find threshold between normal and abnormal current.
figure;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr;Curr_Ice]);
hold on;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr_Kalman,Curr_Ice_Kalman],'k
    ');
m1 = mean(Curr_Kalman(110:375));
s1 = var(Curr_Kalman(110:375));
m2 = mean(Curr_Ice_Kalman(50:350));
s2 = var(Curr_Ice_Kalman(50:350));
%Assume normal distribution
syms x
fun = @(m,s) 1/(sqrt(2*pi*s*2)) * exp(-((x-m)^2)/(2*s^2));
line = eval(solve(fun(m1,s2) == fun(m2,s2), x));
tot = TimeCURR(end) + TimeCURR_Ice(end);
plot(1:tot, line*ones(1,tot), 'r');
legend('Strømforbruk', 'Filtrert', 'Grenseverdi')
ylabel('Strømforbruk[mA]')
xlabel('Tid[ms]')
```

G Motor.m

```
motor = Mot1+Mot2+Mot3+Mot4; % Sum of all motors
Motor_Ice = Mot1_Ice+Mot2_Ice+Mot3_Ice+Mot4_Ice;

% Find closest sample in motor
Idx = zeros(1,length(TimeCURR));
[~,Idx(1)] = min(abs(TimeCURR(1)-TimeRCOU(1:end)));
for i= 2:length(TimeCURR)
    [~,Idx(i)] = min(abs(TimeCURR(i)-TimeRCOU(1:end)));
end
Idx_Ice = zeros(1,length(TimeCURR_Ice));
[~,Idx_Ice(1)] = min(abs(TimeCURR_Ice(1)-TimeRCOU_Ice(1:end)));
for i= 2:length(TimeCURR_Ice)
    [~,Idx_Ice(i)] = min(abs(TimeCURR_Ice(i)-TimeRCOU_Ice(1:end)));
end

%% Plot mot vs curr
% Data where the UAV is stationary is removed.
figure;
plot(motor(Idx(10:385)), Curr(10:385), 'bo');
hold on;
plot(Motor_Ice(Idx_Ice(10:end-10)), Curr_Ice(10:end-10), 'xr');
legend('Normal drift', 'Is på propeller');
xlabel('Motorpådrag');
ylabel('Strømforbruk [mA]');
grid;

%% With curve fitting the following equation is found.
fun1 = @(x) 0.2001*exp(0.001364*x);
y1 = fun1(motor(Idx));
y2 = fun1(Motor_Ice(Idx_Ice));

%% Plot mot vs curr
figure;
plot(motor(Idx(10:385)), Curr(10:385), 'bo');
hold on;
plot(Motor_Ice(Idx_Ice(10:end-10)), Curr_Ice(10:end-10), 'xr');
plot(4500:7500,fun1(4500:7500));
legend('Normal drift', 'Is på propeller', 'Kurve tilpasset til dataene');
xlabel('Motorpådrag');
ylabel('Strømforbruk [mA]');
grid;

%% Plot error function
figure;
err = [y1-Curr; y2-Curr_Ice];
err2 = smooth(err);
err3 = Kalman_filter(err, var(err), 10);
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],err)
hold on;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],err2);
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],err3,'k');
grid;
legend('Orginal', 'Smooth', 'Kalman');
```

H Acceleration.m

```
% Find the acc sample which is closest to the current.
Idx = zeros(1,length(TimeCURR));
[~,Idx(1)] = min(abs(TimeCURR(1)-TimeIMU(1:end)));
for i= 2:length(TimeCURR)
    [~,Idx(i)] = min(abs(TimeCURR(i)-TimeIMU(1:end)));
end
Idx_Ice = zeros(1,length(TimeCURR_Ice));
[~,Idx_Ice(1)] = min(abs(TimeCURR_Ice(1)-TimeIMU_Ice(1:end)));
for i= 2:length(TimeCURR_Ice)
    [~,Idx_Ice(i)] = min(abs(TimeCURR_Ice(i)-TimeIMU_Ice(1:end)));
end

%%
% Here we see a dep. between acc. and curr.
figure;
plot((AccZ(Idx)), Curr, 'bo');
hold on;
plot((AccZ_Ice(Idx_Ice)), Curr_Ice, 'rx')
grid;
legend('Normal data', 'Is-data');
xlabel('Målt akselerasjon i Z-retning[m/s^2]');
ylabel('Strømforbruk[mA]');

%% Classify using svm
TrainData = [AccZ(Idx(1:200))', AccZ_Ice((Idx_Ice(1:200)))';
    Curr(1:200)', Curr_Ice(1:200)'];

classData = [-ones(1,200), ones(1,200)];

TestData = [AccZ(Idx)', AccZ_Ice((Idx_Ice))';
    Curr', Curr_Ice'];

%%
figure;
plot(TrainData(1,1:192), TrainData(2,1:192), 'bo');
hold on;
plot(TrainData(1,193:end), TrainData(2,193:end), 'rx');
legend('Normal data', 'Is-data');
grid;

%%
classes = SVM(TrainData, classData, TestData, 'polynomial', 10, 0.1);

title('Klassifisering av data');
plot(TestData(1,1:200), TestData(2,1:200), 'bo');
hold on;
plot(TestData(1,201:end), TestData(2,201:end), 'rx');
grid;
legend('Klassifiseringsgrense','Normal data', 'Is-data');
xlabel('Strømforbruk[mA]');
ylabel('Akselerasjon i Z-retning[m/s^2]');
```

```
%% Plot the current, blue for correct, red for wrong
figure;
plot([TimeCURR;TimeCURR(end)+TimeCURR_Ice],[Curr',Curr_Ice'],'k');
hold on;

[corr,corrIdx] = find(classes(1:length(AccZ(Idx)))==-1);
[wrong,wrongIdx] = find(classes(1:length(AccZ(Idx)))==1);

[corr_Ice,corrIdx_Ice] = find(classes(length(AccZ(Idx))+1:end)==1);
[wrong_Ice,wrongIdx_Ice] = find(classes(length(AccZ(Idx))+1:end)==-1);

plot(TimeCURR(corr),Curr(corr),'g*'); % Correct
plot(TimeCURR(wrong),Curr(wrong),'rx'); % Wrong

% Correctly classified points.
plot(TimeCURR_Ice(corr_Ice)+TimeCURR(end),Curr_Ice(corr_Ice),'g*');
% Misclassified points.
plot(TimeCURR_Ice(wrong_Ice)+TimeCURR(end),Curr_Ice(wrong_Ice),'rx');

legend('Strømforbruk', 'Korrekt klassifisert', 'Feilklassifisert');
grid;
xlabel('Tid[ms]');
ylabel('Strømforbruk[mA]');

%% Calculate some percentages.
numCorr = sum(corrIdx)+sum(corrIdx_Ice)
numWrong = sum(wrongIdx)+sum(wrongIdx_Ice)
percentCorr = numCorr/(numCorr+numWrong)*100
percentWrong = numWrong/(numCorr+numWrong)*100
```

I NLARX.m

```
%Filter the acceleration
AccZ_Kal = Kalman_filter(AccZ,var(AccZ),1)';
AccZ_Ice_Kal = Kalman_filter(AccZ_Ice,var(AccZ_Ice),1)';

% Pick out acceleration at the motor samples
Idx = zeros(1,length(TimeRCOU));
[~,Idx(1)] = min(abs(TimeRCOU(1)-TimeIMU(1:end)));
for i= 2:length(TimeRCOU)
    [~,Idx(i)] = min(abs(TimeRCOU(i)-TimeIMU(1:end)));
end
Idx_Ice = zeros(1,length(TimeRCOU_Ice));
[~,Idx_Ice(1)] = min(abs(TimeRCOU_Ice(1)-TimeIMU_Ice(1:end)));
for i= 2:length(TimeRCOU_Ice)
    [~,Idx_Ice(i)] = min(abs(TimeRCOU_Ice(i)-TimeIMU_Ice(1:end)));
end

% Create data structures
MotTot = Mot1+Mot2+Mot3+Mot4;
MotTot_Ice = Mot1_Ice+Mot2_Ice+Mot3_Ice+Mot4_Ice;
dataTot = iddata(AccZ_Kal(Idx),MotTot);

% Estimate non linear arx model
model = nlarx(dataTot,[2,2,0], 'sigmoidnet');

% Create simulated data based on the model.
x0 = [AccZ(1),AccZ(1),MotTot(1)]';
x0_Ice = [AccZ_Ice(1),AccZ_Ice(1),MotTot_Ice(1)]';
simOpt = simOptions('InitialCondition', x0);
simOpt_Ice = simOptions('InitialCondition',x0_Ice);
y1 = sim(model,MotTot,simOpt);
y2 = sim(model,MotTot_Ice,simOpt_Ice);

% Find residuals
r1 = (y1-AccZ_LP(Idx)).^2;
r2 = (y2-AccZ_Ice_LP(Idx_Ice)).^2;

%% Plot estimated and measured acceleration
figure;
hold on;
plot([TimeIMU(Idx);TimeIMU_Ice(Idx_Ice)+TimeIMU(Idx(end))],[AccZ(Idx);
    AccZ_Ice(Idx_Ice)], 'color', [1,69/255,0]);
hold on
plot([TimeIMU(Idx);TimeIMU_Ice(Idx_Ice)+TimeIMU(Idx(end))],[y1;y2],'b');
grid;
xlabel('Tid[ms]')
ylabel('Akselerasjon[m/s^2]')
legend('Målt akselerasjon', 'Estimert akselerasjon');
```

```
%% Plot the residual
figure;
plot([TimeIMU(Idx);TimeIMU_Ice(Idx_Ice)+TimeIMU(Idx(end))],[r1;r2]);
grid;
xlabel('Tid[ms]');
ylabel('Differanse i akselerasjon[m/s^2]');

%% Filter the residuals
r1_kal = Kalman_filter(r1,var(r1),0.005); % 0.005
r2_kal = Kalman_filter(r2,var(r2),0.005); % 0.005

% Plot the filtered residuals
figure;
plot([TimeRCOU;TimeRCOU(end)+TimeRCOU_Ice],[r1_kal,r2_kal]);
hold on;
plot(TimeRCOU(end),r1_kal(end),'bo');
plot(TimeRCOU_Ice(1)+TimeRCOU(end),r2_kal(1),'xr');
ylabel('Differanse i akselerasjon [m/s^2]')
xlabel('Tid [ms]');
grid;
thresh = (mean(r2_kal)+mean(r1_kal))/2;
plot(thresh*ones(1,12*10^5));
```