Analyse og Design av Kalmanfilter

Mohammad Hashmatullah Khan
Studieretning: Teknisk Kybernetikk
Universitetet i Oslo
mohamhas@ulrik.uio.no

17. desember 2007

Sammendrag

For å lette arbeidet med å designe og analysere optimale/suboptimale Kalmafiltere kreves det at en gjør en rekke analyser for å verifisere designet. For å gjøre dette ønsker vi å lage et Matlab-program med et grafisk brukergrensesnitt (GUI). Vi antar i første versjon av programmet at alle matriser er tidsinvariante.

I første versjon av programmet har vi implementert diskretisering, simulering og et optimalt Kalmanfilter. For å diskretisere og beregne systemmatrisene Φ, Λ og Γ bruker vi forskjellige numeriske metoder. Beregning av Φ og Λ har blitt gjort gjennom rekkeutvikling. For å forenkle denne utregningen har det blitt foreslått å bruke Matlab-kommandoen *expm*. I utregningen til Γ har det vært nødvendig å foreta Cholesky-faktorisering i tillegg til rekkeutvikling. Dette har gått bra og vi har oppnådd gode resultater. Algoritmen for simulering er konstruert slik at den også beregner apriori og aposteriori likningene i Kalmanfilteret i samme funksjon. Avhengig av hvilke initialbetingelser brukeren ønsker å starte med er simuleringsfunksjonen programmert til å avgjøre i hvilken rekkefølge Kalmanfilterlikningene skal eksekveres.
Det grafiske brukergrensesnittet (GB) er laget i GUIDE. GUIDE er et integrert designverktøy i Matlab for utvikling av GUI-grensesnitt. GUIDE autogenererer en M-fil som vi har brukt til å programmere alle de forskjellige grafiske objektene (knapper, paneler, akser, m.m.) til å utføre ønskede handlinger. Vår GB er designet rundt et menysystem hvor knapper synliggjør forskjellige grensesnitt avhenging av hvilken type oppgave som skal utføres, f. eks diskretisering. Denne måten å lage GB på muliggjør at programmet senere kan utvides til å ha mer funksjonalitet ved at flere knapper kan legges i menyen og utvides med tilhørende grensesnitt.

FORORD

Om rapporten

Denne rapporten er et resultat av et prosjekt som jeg har fullført som en del av min mastergrad - ved Universitet i Oslo, studieretning Tekninsk Kybernetikk.

I denne rapporten vil jeg se på to veldig omfattende temaer. Det første, som fortsatt 30 år etter sin oppdagelse, er et forskningsfelt for nye bruksområder, nemlig Kalmanfilteret. Det andre en applikasjon med et grafisk brukergrensesnitt skrevet i Matlab for å analysere og designe et Kalmanfilter.

Tradisjonelt har implementasjonen av et Kalmanfilter blitt utført ved å benytte seg av spesialdesignet programvare. Denne programvaren har ofte blitt designet, kodet og feiltestet fra grunnen av for hver Kalmanfilter applikasjon noe som har ført til lange utviklingstider. Jeg vil i denne rapporten gi grunnlaget for et testplattform slik at designprosessen for å analysere og verifisere et Kalmanfilter simplifiseres, og at utviklingstiden reduseres.

Målgruppe

Rapporten er rettet mot kybernetikk studenter på Cand.scient nivå med interesse for stokastiske systemer og Kalmanfiltere. Det forutsettes at leseren har grunnleggende forståelse for lineære dynamiske systemer under påvirkning av stokastiske forstyrrelser. Kjennskap til sannsynlighetsregning og statistisk modellering, samt diskret lineær algebra vil gjøre forsåelsen enklere. Mye av teksten omhandler programvareutvikling så noen elementære forkunnskaper i Matlab programmering herunder GUI-programmering i GUIDE anbefales å kunne.

Online

På min hjemmeområdet ved UIO finner dere en pdf versjon av denne rapporten. I tillegg ligger skildekoden der for det grafiske brukergrensesnittet skrevet i Matlab.

http://folk.uio.no/mohamhas/kalman/

Dersom dere ønsker å gi noen tilbakemeldinger kan jeg kontaktes på

hashmat@online.no

Takkskyldig

Jeg vil takke flere personer som har bidratt for å få denne rapporten fullført. Først og fremst vil jeg takke Prof. Dr. Oddvar Halligstad for hans veiledning og gjestfrihet. Hans kompetanse innen fagfeltet har vært av avgjørende betydning for meg.

Videre ønsker jeg å takke Zeghum Abbas for hans kontrolllesing og bevissjekking av materialet som omhandler Kalmanfilteret. Mange takk går også til Kandeeban Arumugam og Sumandeep Grewal Kaur, for deres kommentarer og tilbakemeldinger under arbeidet med programmeringen.

Mohammad H. Khan
Oslo, Desember 2007

Innhold

Kapittel 1

Introduksjon

Tema for oppgaven er å programmere et grafisk brukergrensesnitt i Matlab for å designe og analysere et Kalmanfilter. Oppgaven består av to deler. Det første, en oppsummering av de likninger som skal implementeres og de numeriske metoder som skal brukes. Den andre delen, å lage et Matlab-program med et grafisk brukergrensesnitt (GUI) og en kort beskrivelse for eventuelle brukere. Oppbygning av rapport følger denne strukturen:

- KAPITTEL 2: Vi oppsumerer viktig bakgrunnsstoff for leseren om stokastisk estimering og Kalmanfilter (KF). Vi beskriver den diskrete versjonen av KF. Til slutt oppsummeres likningene for diskretisering, simulering og optimal Kalmanfilter.
- KAPITTEL 3: De numeriske metoder som har blitt benyttet til å diskretisere systemet presenteres i dette kapittelet. Hvordan utregningen av initialtilstand beregnes gjennomgås også.
- KAPITTEL 4: Vi bruker Matlab til å implementere diksretisering, simulering og optimal Kalamanfilter beregning. Vi ser på programkoden for disse og gir en detaljert beskrivelse av virkemåten.
- KAPITTEL 5: Dette kapittelet er tredelt. Den første delen omhandler GUIDE. Det blir gitt en oversikt over hvilke komponenter som inngår i GUIDE og hvilke funksjoner som ligger i den autogenererte M-filen. Den andre delen beskriver hva grafiske objekter er, hvilken datastruktur som benyttes for å samle data på et sted, og om globale variable og bruken av disse.
- KAPITTEL 6: Vi ser på videre arbeid. Hvilken utvidelsesmulighet og forbedringer som vi ønsker skal gjøres for denne versjonen av Matlab-programmet uttrykkes.
- KAPITTEL 7: Dette kapittelet samler konklusjonene vi har kommet frem til.

Kapittel 2

Bakgrunn

2.1 Stokastisk estimering

Det finnes mange metoder for å estimere en ukjent prosesstilstand fra kjente målinger. Tilstandsestimatet er derfor ofte basert på elektriske målinger generert av mekaniske, magnetiske, optiske, treghets eller akustiske sensorer. Noe mange av disse metodene ikke tar hensyn til er at målingene nesten alltid er påvirket av *støy*, og at denne støyen er statistisk i sin natur. Et optimalt tilstandsestimat må derfor utledes fra *stokastiske* metoder.

Tilstandsromnotasjon

Anvendelsen av optimal estimering er basert på den matematiske beskrivelsen av et dynamisk system. Dynamiske prosesser (systemer[1]) og de tilhørende tilstandsvariabler beskrives ofte av n'te ordens differensiallikninger. En mer oversiktelig beskrivelse i form av tilstadsrommodeller benyttes i større grad. Denne formen er veldig nyttig når en ønsker å gi en statistisk beskrivelse av systemoppførsel. Betrakt et generell dynamisk prosess beskrevet i form av en differensiallikning på formen

$$\begin{aligned} \overset{(m)}{x}(t) + a_{m-1}x^{(m-1)}(t) + \ldots + a_0 x(t) &= u(t) \\ \overset{(m)}{x}(t) &= -a_{m-1}x^{(m-1)}(t) - \ldots - a_0 x(t) + u(t) \end{aligned}$$

hvor vi definerer et sett av tilstandsvariable $x_1(t), \ldots, x_m(t)$ slik

[1]Ordet prosess og system brukes ofte om hverandre.

$$\begin{aligned} x_1(t) &\triangleq x(t) \\ x_2(t) &\triangleq \dot{x}_1(t) \\ &\vdots \\ x_m(t) &\triangleq \dot{x}_m(t) \end{aligned}$$

Vi kan nå skrive den skalare differensiallikningen om til en vektor-matrise form

$$\underbrace{\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_{m-1}(t) \\ \dot{x}_m(t) \end{bmatrix}}_{\underline{\dot{x}}(t)} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{m-1} \end{bmatrix}}_{F} \underbrace{\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{m-1}(t) \\ x_m(t) \end{bmatrix}}_{\underline{x}(t)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}}_{L} u$$

som fører til tilstandsrommodellen

$$\underline{\dot{x}}(t) = F\underline{x}(t) + L\underline{u} \quad , \quad gitt\, \underline{x}_0 = \begin{bmatrix} x_1(t_0) \\ x_2(t_0) \\ \vdots \\ x_{m-1}(t_0) \\ x_m(t_0) \end{bmatrix} \tag{2.1}$$

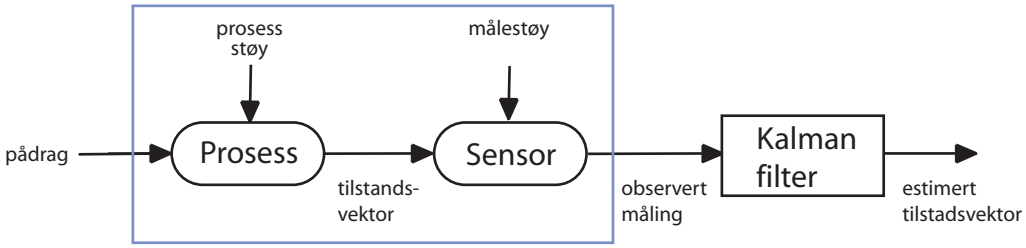
Med utgangspunkt i likning (2.1), kan en typisk systemmodell og dens dynamikk representeres ved en første ordens vektor-matrise differensiallikning

$$\underline{\dot{x}}(t) = F(t)\underline{x}(t) + L(t)\underline{u}(t) + G(t)\underline{v}(t) \tag{2.2}$$

hvor $\underline{x}(t)$ er tilstandsvektoren, $\underline{u}(t)$ er pådrag (deterministisk) og $\underline{v}(t)$ en stokastisk nullvarians hvit støysignal. Matrisene $F(t)$, $L(t)$ og $G(t)$ fremkommer fra formuleringen av systemmodellen. Dette er den kontinuerlige formen som ordinært benyttes i moderne estimeringsteorier. Tilstansvektoren for et dynamisk system består av de verdiene som fullstendig beskriver utviklingen av systemet, uten påvirkning av støy, $\underline{v}(t)$, og pådrag, $\underline{u}(t)$ [1]. Ved å anvende tilstandsvektoren i et gitt tidspunkt t_{start}, samt beskrivelsen av pådraget og støybidraget i tidsintervallet $[t_{start}, t_{final}]$, vil det være mulig å beregne enhver tilstand frem til t_{final}.

Observasjonsproblemet

Et relatert problem tilknyttet til lineær systemteori, hvor prosessen (systemet) er beskrevet som i (2.2), går ut på å estimere de interne tilstandene til systemet



Figur 2.1: Typisk oppsett av Kalman filter.

når kun utgangen og det kjente pådraget er tilgjengelig. Fremgangsmåten for å løse dette problemet er som sakt basert på (2.2). I tillegg brukes det en målelikning som beskriver relasjonen mellom tilstandsvariablene og målingene. En vanlig måte å representere denne sammenhengen på er:

$$\underline{z}(t) = H\underline{x}(t) + \underline{w}(t)$$

Der $\underline{w}(t)$ er en stokastisk variabel som representerer målestøyen. Ofte er det slik at noen tilstandsvariable ikke kan måles direkte, vi må da benytte all *tilgjengelig* data for å regne ut disse. Behovet for et "filter" som kombinere all tilgjengelig måling, sammen med kunnskapen om prosessen og måleenheten for å beregne (estimere) tilstandsverdiene er tydelig.

2.2 Kalman filter

En av de mest brukte og signifikante matematiske verktøyene som brukes til stokastisk estimering ut fra "støyende" sensor målinger er Kalmanfilteret [2], se figur 2.2. Kalmanfilteret er essensielt et sett av matematiske likninger som implementerer en prediktor-korrektor estimator, som både er optimal og rekursiv. Den er optimal fordi Kalmanfilteret behandler all informasjon som mates inn, den prosesserer alle tilgjengelige målinger uavhengig av presisjon for å estimere den ønskede tilstandsvariabelen. Dette gjør den ved å bruke kunnskapen om prosessen og måleenheten, de statistiske egenskapene til prosess- og målestøy, målefeil, pluss tilgjengelig informasjon om initialbetingelsene [3]. At Kalmanfilteret er rekursivt kommer av at foregående data ikke lagres og prosesseres ved hver måling.

Det finnes et kontinuerlig Kalmanfilter som beregner estimater for en kontinuerlig prosess, og et diskret filter for diskrete prosesser. Det diskrete Kalmanfilteret er det opprinnelige filteret som først ble oppfunnet av Rudolf E. Kalman. Det kontinuerlige Kalmanfilteret benyttes sjeldent, noe som henger sammen med at de prosesser som Kalmanfilteret i praksis brukes for, styres av diskrete signaler. Det er også slik at måleverdiene foreligger ved diskrete tidspunkter.

2.2.1 Diskret Kalman filter (DKF)

Det diskrete Kalmanfilteret har som formål å beregne et optimalt estimat av tilstanden $x \in \mathbb{R}^n$ for en tidsdiskret prosess, uttrykt ved en lineær stokastisk differenslikning på formen

$$\underline{x}_{k+1} = \Phi_k \underline{x}_k + \Lambda_k \underline{u}_k + \Gamma_k \underline{v}_k \tag{2.3}$$

med målinger $z \in \mathbb{R}^m$

$$\underline{z}_k = H\underline{x}_k + \underline{w}_k \tag{2.4}$$

De stokastiske variablene $\underline{v}_k$ og $\underline{w}_k$ representerer henholdsvis prosess- og målestøy. Vi antar at de er uavhengige av hverandre, er hvite og har normalfordelte verdier, slik at forventningsverdi av støybidragene blir

$$\begin{aligned} E\{\underline{v}_k\} &= \underline{0} \\ E\{\underline{w}_k\} &= \underline{0} \end{aligned}$$

og (auto)kovarianser

$$E\{\underline{v}_k \underline{v}_l^T\} = R_k \delta_{kl} \tag{2.5}$$
$$E\{\underline{w}_k \underline{w}_l^T\} = Q_k \delta_{kl} \tag{2.6}$$

hvor Q_k er prosesstøyens kovarians og R_k målestøyens kovarians. Videre har vi at $\underline{v}_h$ og $\underline{w}_k$ er ukorrelerte

$$E\{\underline{v}_k \underline{w}_k^T\} = \underline{0}$$

Initialbetingelsen $\underline{x}_0$ er en hvit stokastisk variabel med forventningsverdi $\overline{\underline{x}}_0$ og kovarians $\overline{P}_0$. Det er også ingen korrelasjon mellom initialbetingelsen og støybidragene $\underline{v}_k$ og $\underline{w}_k$

$$\begin{aligned} E\{\underline{x}_0\} &= \overline{\underline{x}}_0 \\ Kov\{\underline{x}_0\} &= \overline{P}_0 \\ E\{\underline{x}_0 \underline{v}_k^T\} &= \underline{0} \\ E\{\underline{x}_0 \underline{w}_k^T\} &= \underline{0} \end{aligned}$$

Basert på disse antagelsene og likningene (2.3) og (2.4) kan vi nå skrive opp Kalmanfilter likningene. Vi definerer $\overline{\underline{x}}_k \in \mathbb{R}^n$ som vår predikterte tilsand (apriori) og $\hat{\underline{x}}_k \in \mathbb{R}^n$ som vår estimerte tilstand (aposteriori). Vi kan nå definere kovariansmatrisene for apriori- og aposteriori estimeringsavviket henholdsvis slik

$$\overline{P}_{k+1} = E\{(\underline{x}_k - \overline{\underline{x}}_k)(\underline{x}_k - \overline{\underline{x}}_k)^T\} = \Phi_k \hat{P}_k \Phi_k^T + \Gamma_k Q_k \Gamma_k \quad (2.7)$$

$$\hat{P}_k = E\{(\underline{x}_k - \hat{\underline{x}}_k)(\underline{x}_k - \hat{\underline{x}}_k)^T\} = (I - K_k H)\overline{P}_k \quad (2.8)$$

Kalmanfilteret gir et optimalt tilstandsestimat i betydningen at aposteriori-avviket minimaliseres, det vil si slik at $\hat{P}_k$ får en minimal verdi. I utledningen til Kalmanfilteret ønsker vi å finne en likning som beregner et aposteriori-estimat (estimert) , $\hat{\underline{x}}_k$, som en lineær kombinasjon av en apriori-estimat (prediktert), $\overline{\underline{x}}_k$ og en vektet differanse mellom en virkelig måling $\underline{z}_k$ og en prediktert måleverdi $H\overline{\underline{x}}_k$, slik:

$$\hat{\underline{x}}_k = \overline{\underline{x}}_k - K_k(\underline{z}_k - H\overline{\underline{x}}_k) \quad (2.9)$$

Differansen $\underline{z}_k - H\overline{\underline{x}}$ er innovasjonen og forteller noe om avviket mellom prediktert måling $H\overline{\underline{x}}_k$ og virkelig måling $\underline{z}_k$. En innovasjonsverdi på null betyr at prediktert måling er helt lik den virkelige målte verdien. K_k matrisen er Kalmanfilter-forterkningen som skal minimalisere aposteriori-avviket $\hat{P}_k$, se likning (2.8), og er gitt ved

$$K_k = \overline{P}_k H^T (H\overline{P}_k H^T + R)^{-1} \quad (2.10)$$

Apriori-estimatet $\overline{\underline{x}}_k$ i likning (2.9) beregnes ved

$$\overline{\underline{x}}_{k+1} = \Phi \hat{\underline{x}}_k + \Lambda \underline{u}_k \quad (2.11)$$

Figur 2.2 viser blokkdiagram for prosess,sensor og Kalmanfilter. For mere detalier og utledniger se [1] [3]

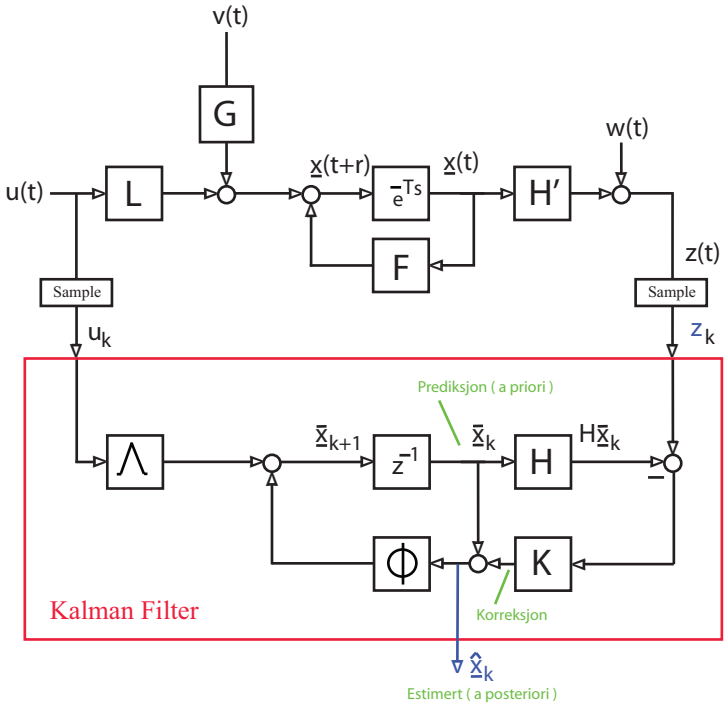
2.2.2 DKF algoritmen

Kalmanfilterlikningene består av to deler: *tidsoppdaterings*likningene (TO) og *måleoppdaterings*likningene (MO). TO har som formål å beregne apriori estimatet $\overline{\underline{x}}_{k+1}$, ved å projisere fremover i tid den aktuelle aposteriori estimat $\hat{\underline{x}}_k$. Kovariansmatrisen for apriori estimeringsavviket $\overline{P}_{k+1}$ blir også beregnet. MO vil legge inn en ny måling $\underline{z}_k$ sammen med apriori estimatet $\overline{\underline{x}}_k$ for å beregne en forbedret aposteriori estimat $\hat{\underline{x}}_k$. MO beregner også kovariansmatrisen for aposteriori estimeringsavviket $\hat{P}_k$. I tabell 2.1 og 2.2 er likningene for henholdsvis TO og MO presentert.

$$\begin{aligned} \overline{\underline{x}}_{k+1} &= \Phi_k \hat{\underline{x}}_k + \Lambda \underline{u}_k \\ \overline{P}_{k+1} &= \Phi_k \hat{P}_k \Phi_k^T + \Gamma_k Q_k \Gamma_k \end{aligned}$$

Tabell 2.1: Tidsoppdateringslikningene for det diskrete KF.

Systemmatrisen Φ_k, pådragsmatrisen Λ_k og prosesstøymatrisen Γ_k er fra likning (2.3), mens prosesstøyens kovariansmatrise Q_k er fra likning (2.6).



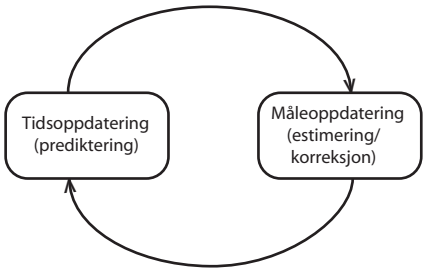
Figur 2.2: Prosess med Kalman filter.

$$
\begin{aligned}
K_k &= \overline{P}_k H^T (H \overline{P}_k H^T + R)^{-1} \\
\underline{\hat{x}}_k &= \underline{\overline{x}}_k - K_k(\underline{z}_k - H\underline{\overline{x}}_k) \\
\hat{P}_k &= (I - K_k H)\overline{P}_k
\end{aligned}
$$

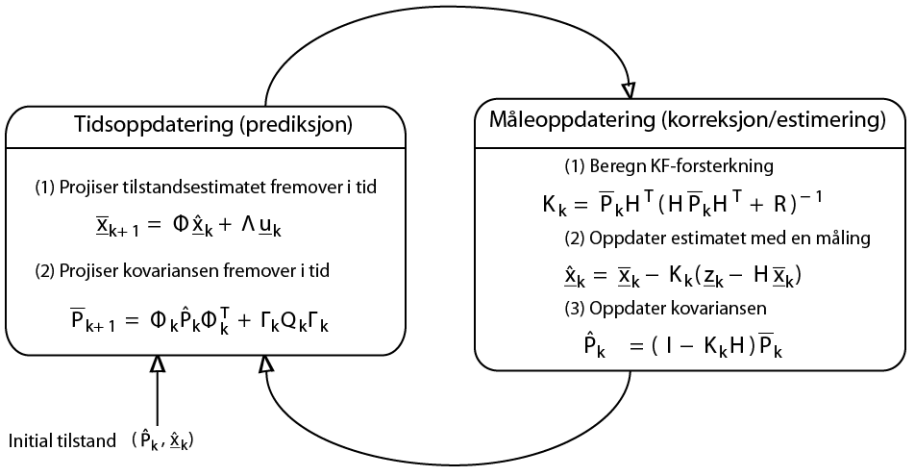
Tabell 2.2: Måleoppdateringslikningene for det diskrete KF.

Den diskrete Kalmanfilter (DKF) algoritmen har en løpende syklus, se figur 2.3. TO-likningen vil projisere fremover i tid tilstandsestimatet $\underline{\hat{x}}_k$, mens MO-likningen vil korrigere det projiserte estimatet med en virkelig måling.

Det første MO-likningene vil gjøre er å beregne Kalman-forsterkningen K_k. Det neste steget blir å utføre en måling på prosessen for å innhente $\underline{z}_k$. Denne blir brukt til å generere en aposteriori estimat $\underline{\hat{x}}_k$ slik som i tabell 2.2. Det siste steget, før TO-likningen starter sine beregninger, vil være å regne ut kovariansmatrisen for aposteriori estimeringsavviket, $\hat{P}_0$. Etter hver (TO,MO)-oppdateringspar, blir prosessen repetert slik at forrige aposteriori estimat blir brukt til å prediktere den nye apriori estimatet, $\underline{\overline{x}}_{k+1}$, figur 2.4. Denne rekursive strukturen som Kalmanfilter algoritmen følger er en attraktiv egenskap, det gjør selve implementasjonen av filteret mer praktisk siden tidligere verdier ikke nødvendigvis trenger å lagres. Hardwaren trenger da et mye mindre minneregister for å utføre beregningene.



Figur 2.3: Den løpende Kalman filter syklusen.



Figur 2.4: Komplett Kalman filter.

2.3 Oppsumering av likninger

Dette delkapittelet gir en oversikt over de matematiske likninger som har blitt implementert i MATLAB-programmet. Mens neste kapittel vil vise hvordan likningene er implementert og de numeriske metoder som har blitt brukt. Delkapittelet er delt opp i seksjoner som henholdsvis viser likningene for diskretisering, simulering og optimalt Kalmanfilter.

2.3.1 Diskretisering av kontinuerlige systemer

Gitt det lineære stokastisk kontinuerlige systemmodellen

$$\dot{\underline{x}}(t) = F(t)\underline{x}(t) + L(t)\underline{u}(t) + G(t)\underline{v}(t) \quad (2.12)$$
$$\underline{z}(t) = H\underline{x}(t) + \underline{w}(t) \quad (2.13)$$

med antagelsene

$$\begin{array}{rcll} E\{\underline{x}(t_0)\} & = & \overline{\underline{x}}_0 & E\{(\underline{x}(t_0)-\overline{\underline{x}}_0))(\underline{x}(t_0)-\overline{\underline{x}}_0)^T\} = \overline{P}_0 \quad E\{\underline{x}(t_0)\underline{v}^T(t)\} = \underline{0} \\ E\{\underline{v}(t)\} & = & \underline{0} & E\{\underline{v}(t)\underline{v}^T(\tau)\} = \delta(t-\tau)\tilde{Q}(t) \quad E\{\underline{x}(t_0)\underline{w}^T(t)\} = \underline{0} \\ E\{\underline{w}(t)\} & = & \underline{0} & E\{\underline{w}(t)\underline{w}^T(\tau)\} = \delta(t-\tau)\tilde{R}(t) \quad E\{\underline{v}(t)\underline{w}^T(t)\} = \underline{0} \end{array}$$

Ved å diskretisere det kontinuerlige systemmodellen (2.12) og (2.13) oppnår vi følgende lineære diskrete systemmodell [4]

$$\underline{x}_{k+1} = \Phi_k \underline{x}_k + \Lambda_k \underline{u}_k + \Gamma_k \underline{v}_k \tag{2.14}$$
$$\underline{z}_k = H\underline{x}_k + \underline{w}_k \tag{2.15}$$

Ifølge [5] er systemmatrisene gitt ved

$$\Phi = \Phi(t_{k+1}, t_k) \tag{2.16}$$
$$\Lambda \underline{u}_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) L\underline{u}(\tau) d\tau \tag{2.17}$$
$$\Gamma \underline{v}_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) G\underline{v}(\tau) d\tau \tag{2.18}$$
$$\Gamma Q \Gamma^T = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) G\tilde{Q}G^T \Phi(t_{k+1}, \tau)^T d\tau \tag{2.19}$$

Utfra disse kan vi beregne systemmatrisene, og siden vi antar tidsinvarians kan vi skrive $\Phi_k = \Phi$, $\Lambda_k = \lambda$ og $\Gamma_k = \Gamma$.

2.3.2 Simulering av stokastiske systemer

Etter å ha diskretisert det kontinuerlige systemet ved å finne systemmatrisene. Ønsker vi å simulere systemet ved å bruke likning (2.14), gjengitt her

$$\underline{x}_{k+1} = \Phi \underline{x}_k + \Lambda \underline{u}_k + \Gamma \underline{v}_k k$$

2.3.3 Optimalt Kalman filter

Kalmanfilter likningene for (2.14) og (2.15) er gitt i [1]. Disse er delt opp i tids-oppdatering (TO) og måleoppdatering (MO)

$$TO \begin{cases} \overline{\underline{x}}_{k+1} = \Phi \underline{\hat{x}}_k + \Lambda \underline{u}_k & \underline{\hat{x}}_0 ergitt \\ \overline{P}_{k+1} = \Phi \hat{P}_k \Phi^T + \Gamma Q \Gamma^T & \hat{P}_0 ergitt \end{cases} \tag{2.20}$$

$$MO \begin{cases} \underline{\hat{x}}_k = \overline{\underline{x}}_k + K_k(\underline{z}_k - H\overline{\underline{x}}_k) \\ K_k = \overline{P}_k H^T (H\overline{P}_k + R)^{-1} \\ \hat{P}_k = (I - K_k H)\overline{P}_k \end{cases} \tag{2.21}$$

her starter vi initielt med $\underline{\hat{x}}_0$ og $\hat{P}_0$ dermed vil TO-likningene utføres først. Første måling vil bli utført ved $k = 1$.

Kapittel 3

Numeriske metoder for diskretisering

De numeriske metoder som blir brukt for å løse likningene i delkapittel 2.3 (Oppsummering av likninger), vil bli presentert i dette kapittelet. Vi vil anta at likningene er tidsinvariante. Hvordan disse numeriske metodene blir implementert i Matlab vises i kapittel 4.

3.1 Systemmatriser: Φ, Λ og Γ

I [5] blir systemmatrisene Φ_k, Λ_k og Γ_k løst ved å anta tidsinvarians. Vi kan derfor skrive $\Phi_k = \Phi$, $\Lambda_k = \lambda$ og $\Gamma_k = \Gamma$. Matrisene ϕ og Λ finner vi ved å bruke *rekkeutvikling*. For å få en større nøyaktighet foretas rekkeutviklingen over et kortete tidsintervall. Denne nedskaleringen av tiden gjøres på følgende måte[HN5]

$$d = \frac{\Delta t}{2^m} \tag{3.1}$$

$$m = min_i\Big\{i\Big|\frac{\|F\|\Delta t}{2^i} < 1\Big\} \tag{3.2}$$

$$\|F\| = max_i \sum_{i=1}^{n} |f_{ij}| \tag{3.3}$$

Vi kan nå beregne $\Phi(d)$ og $\Lambda(d)$, se likning (2.16) og (2.17), med den nye tiden d. Det blir antatt at $\underline{u_k}$ er konstant slik at vi kan bestemme Λ entydig:

$$\Phi(d) \approx \Phi_d = \sum_{k=0}^{n} \frac{1}{k!}(Fd)^k \tag{3.4}$$

$$\Lambda(d) = \int_0^d \Phi(\tau) L d\tau \approx \Lambda_d = \sum_{k=0}^{n} \frac{1}{(k+1)!} F^k d^{k+1} L = \nabla L d \tag{3.5}$$

$$\nabla = \sum_{k=0}^{n} \frac{1}{(k+1)!}(Fd)^k \tag{3.6}$$

Antall ledd, n, vi skal ha med i rekkeutviklingen bestemmes utfra hvilken nøyaktighet man ønsker å oppnå [5]. Siden Matlab opererer med dobbelpresisjon kan man velge n = 18. Etter å beregnet summene er det nødvendig å tilbakeskalere tiden fra d til Δ t

$$\Phi(\Delta t) \approx \Phi_{\Delta t} = \Phi_d^{2^m} \tag{3.7}$$

$$\Lambda(\Delta t) \approx \Lambda_{\Delta t} = \sum_{k=0}^{2^m-1} \Phi_h^k \Lambda_h \tag{3.8}$$

Siden likning (2.18) inneholder produktet av $\Gamma \underline{v}_k$ og $\underline{v}_k$ er hvit støy, kan Γ ikke beregnes entydig fra denne. Vi må finne Γ på en annen måte. For å beregne Γ er vi nødt til å bruke likning (2.19). Systemmatrisen Γ kan nå beregnes ved å bruke Cholesky-faktorisering [8], man antar at $Q = I$. Vi definerer

$$S \triangleq \Gamma Q \Gamma^T \tag{3.9}$$

En *Cholesky-faktorisering* på S gir oss to matriser U og D. hvor U er en øvretriangular matrise og D en diagonalmatrise

$$U = \begin{bmatrix} 1 & & x \\ & 1 & \\ 0 & & 1 \end{bmatrix} \qquad D = \begin{bmatrix} d_{11} & & 0 \\ & d_{22} & \\ 0 & & d_{nn} \end{bmatrix}$$

diagonalelementer $d_{11}, \cdots, d_{nn} \geq 0$. X sier bare at det finnes andre elementer i matrisen også. Vi kan nå skrive sammenhengen mellom S og matrisene U og D slik

$$\underline{cholesky : S \mapsto \{U, D\}} \quad : \tag{3.10}$$

$$S = UDU^T = \underbrace{UD^{\frac{1}{2}}}_{\Gamma} \underbrace{I}_{Q} \underbrace{D^{\frac{1}{2}}U}_{\Gamma^T} \tag{3.11}$$

$$\Rightarrow \Gamma = UD^{\frac{1}{2}} \quad og \quad Q = I \tag{3.12}$$

Før vi kan foreta en Cholesky-faktorisering på S, må produktet $S = \Gamma Q \Gamma^T$ beregnes. I [6], kort oppsumert her, gjøres dette slik:

$$A = \begin{bmatrix} F & G\tilde{Q}G \\ \underline{0} & -F^T \end{bmatrix} \tag{3.13}$$

$$B = e^{A\Delta t} = \sum_{k=0}^{n} \frac{1}{(k)!}(A\Delta t)^k = \begin{bmatrix} B_{11} & B_{12} \\ \underline{0} & B_{22} \end{bmatrix} \tag{3.14}$$

$$S = \Gamma Q \Gamma^T = B_{12} B_{22}^{-1} \tag{3.15}$$

Elementene i A-matrisen hentes fra systemlikningen, se likning (2.12) og de gitte antagelser.

3.2 Målestøy: $\tilde{R}$

Vi går utfra at målelikningen er en differenslikning og antar derfor at bruker-en av programmet skal oppgi R, målestøyens kovarians. Men dersom vi hadde antatt en kontinuerlig målemodell ville det vært nødvendig å diskretisere slik vist under. For å diskretisere målestøyen bruker vi også her Cholesky-faktorisering.

$$\underline{cholesky : \tilde{R} \mapsto \{U, D\}} \tag{3.16}$$

$$UDU^T = \underbrace{UD^{\frac{1}{2}}}_{R} \underbrace{D^{\frac{1}{2}}U}_{R^T} \tag{3.17}$$

$$\Rightarrow R = UD^{\frac{1}{2}} \tag{3.18}$$

3.3 Initialbetingelser.

For å beregne en initial startverdi $\underline{x}_0$ så må man faktorisere initiell kovarians. Avhengig av hvilken type initiell kovarians man har (aprior eller aposteriori), vil $\underline{x}_0$ være normalfordelt på en av disse måtene:

$\underbrace{\underline{x}_0 \sim N(\underline{\hat{x}}_0, \hat{P}_0)}_{(1)}$ eller $\underbrace{\underline{x}_0 \sim N(\underline{\overline{x}}_0, \overline{P}_0)}_{(2)}$

Ved å trekke n-normalfordelte tall $\underline{\gamma}$, kan vi finne $\underline{x}_0$ på en av disse to måter:

$$\underline{cholesky : \hat{P}_0 \mapsto \{U, D\}} \tag{3.19}$$

$$(1) \quad \Rightarrow \quad \underline{x}_0 = \underline{\hat{x}}_0 + (UD^{\frac{1}{2}})\underline{\gamma} \tag{3.20}$$

$$\underline{cholesky : \overline{P}_0 \mapsto \{U, D\}} \tag{3.21}$$

$$(2) \quad \Rightarrow \quad \underline{x}_0 = \underline{\overline{x}}_0 + (UD^{\frac{1}{2}})\underline{\gamma} \tag{3.22}$$

Kapittel 4

Implementasjon i MATLAB

Dette kapittelet vil bl.a. beskrive i detalj hvordan de numeriske metodene i kapittel 3 er implementert i Matlab. Videre vil det bli gitt en grundig beskrivelse av hvordan diskretisering, simulering og optimalt Kalmanfilter har bitt kodet inn i applikasjonen.

4.1 Diskretisere kontinuerlige systemer

4.1.1 Beregning av Φ og Λ

Beregning av Φ og Λ er gjort i [5] og gjengit her.

```
function [La,Fi] = kp2dpLa(F, L, d)
nt = 18;
n = size(F,1);
%Nedskalering av tiden
m = ceil(log2(d*norm(F))); if (m<0) m=0; end; dd = d/2^m;

%Beregn Fi(dd) og La(dd) med den nye tiden dd.
W = diag(ones(n,1)); s=dd;
for k=2:nt
    i = nt + 2 - k;
    s = dd/i;
    Fi = F*W*s;
   W = Fi + diag(ones(n,1));
end
Fi = W * F * dd + diag(ones(n,1));
La = W*L*dd;

%Tilbakeskaler tiden fra dd -> d
%og beregn Fi(d) og La(d)
A = diag(ones(n,1));
if(m > 0)
    for i = 1:m
        W = Fi*A;
        A = W+A;
        W = Fi * Fi;
        Fi = W;
    end
end
La = A*La;
```

Kort forklart starter vi med å beregne det nedskalerte tidsintervallet *dd* på **linje 5**, se også likning 3.1 for sammenlikning. Fra **linje 8** til **14** beregnes det en hjelpestørrelse W slik at beregningen for Φ og Λ kan skje i samme *for*-løkke, dette forenkler utregningen betraktelig. Etter utregning av W brukes denne på **linje 15** og **16** sammen med diverse andre parametere for å finne Φ og Λ. Ettersom det ble gjort en nedskalering av tiden er det også nødvendig å tilbakeskalere tiden, se likning 3.7 og likning 3.8. Men dette gjøres kun hvis $m > 0$ på **linje 21**.

En alternativ og enklere måte for å regne ut Φ, er ved å bruke Matlab-kommandoen expm på denne måten:

$$\Phi(\Delta t) = expm(F * \Delta t) \tag{4.1}$$

4.1.2 Beregning av Γ

Beregningen av Γ foregår i to ledd. Det første er å beregne produktet $S = \Gamma Q \Gamma^T$. Deretter å utføre en Cholesky-faktorisering på S for å spalte ut Γ med den antagelsen at $Q = I$. Matlab koden under viser hvordan dette er implementert. Se likningene (3.10) til (3.15) for sammenlikning.

Listing 4.1: Funksjon for beregning av Γ

```
1 function Ga  = kp2dpGa(F, G, Q_tilde, d)
2     [m,n] = size(F);
3     A = [F G*Q_tilde*G'; zeros(m,n) -F'];
4     B = expm(A*d);
5     S = B(1:n,n+1:2*n) * inv(B(n+1:2*n,n+1:2*n));
6     Ga = chol(S,'lower');
```

Fra listing: 4.1 ser vi at på **linje 3** blir matrisen $A = \begin{bmatrix} F & G\tilde{Q}G \\ \underline{0} & -F^T \end{bmatrix}$ beregnet, og er en direkte gjengivelse av likning (3.13). På **linje 4** ønsker vi å beregne rekkeutviklingen for $B = \sum_{k=0}^{n} \frac{1}{(k)!}(A\Delta t)^k$. I Matlab er det enklest å bruke kommandoen $expm(Ad)$, ved å bruke denne trenger vi ikke å tenke på hvor mange ledd vi skal utvikle til. Vi oppnår dobbelpresisjon i Matlab. Vi trekker så ut submatrisene fra B i **linje 5** og setter dem til S. Første ledd er nå ferdig og produktet $S = \Gamma Q \Gamma^T$ er regnet ut. Det neste er å bruke Matlab kommandoen $chol(S,'lower')$ og trekke ut Γ. "lower" forteller Matlab at *chol* skal returnere en nedretriangulær matrise slik at S kan skrives på formen $S = RR^T(\Gamma\Gamma^T)$.

4.1.3 Simulering i Matlab

Etter å ha beregnet systemmatrisene Φ, Λ og Γ kan vi sette dem inn i differenslikning $\underline{x}_{k+1} = \Phi\underline{x}_k + \Lambda\underline{u}_k + \Gamma\underline{v}_k$ og simulere det stokastisk kontinuerlige systemet vist av likning (2.12). Vi antar i denne versjonen at $\underline{u}$ er en skalar.

Listing 4.2: Matlab-kode for simulering av system

```
1  F  = hfig_main.params.F;
2  Fi = hfig_main.params.Fi;
3  La = hfig_main.params.La;
4  Ga = hfig_main.params.Ga;
5  u  = hfig_main.params.u;
6
7  P0_hatt = hfig_main.params.P0_hatt;
8  x0_hatt = hfig_main.params.x0_hatt;
9
10 d  = hfig_main.params.d;  %tidsintervall
11 t0 = hfig_main.params.t0; %starttid
12 t_f = hfig_main.params.t_f; %slutttid
13
14 %antall diskrete steg + 0-steg lagt til
15 N = ((t_f - t0)/d) + 1;
16
17 %1. Beregn x0
18 x0 = x0_hatt + chol(P0_hatt)' * randn(n,1);
19
20 %2. Simuler
21 for k = 2:N
22     X(:,k) = Fi*X(:,k-1) + La*u + Ga*randn(n,1);
23     X_det(:,k) = Fi*X_det(:,k-1) + La*u;
24 end
```

Linje 1 til 12 leser inn de nødvendige globale[1] variabler og lagrer dem i lokale variabler med samme navn. Dette gjøes for å få en mer oversiktelig programkode, siden de globale variablene har et mye lengre navn. Etter utregningene er det derfor nødvendig å gi de globale variablene de nye verdiene dersom disse har blitt endret.*d* forteller oss hvilken tidsintervall vi skal diskretisere over. Mens *t0* og *t_f* er henholdsvis start- og sluttidspunkt for simulering. På **linje 15** ønsker vi å vite den totale antall tidsskritt, N, vi skal beregne tilstandene $\underline{x}$ for. Det finner man ved å beregne hvor *mange* tidsintervall d som inngår i simuleringstiden $(t\_f - t_0)$. Det lagges til ett ekstra tidsskritt $+1$ for å få med seg tidsskittet for t_0. Før simuleringen kan starte er det nødvendig å regne ut initialtilstanden $\underline{x}_0$. Siden vi på **linje 18** har antatt at vi har $\{\hat{P}_0, \underline{\hat{x}}_0\}$ tilgjengelig må vi bruke likning (3.20), gjengitt her for sammenlikning med **linje 18** i Matlab-koden:

$$\underline{x}_0 = \overline{\underline{x}}_0 + (UD^{\frac{1}{2}})\underline{\gamma}$$

Ved å sammenligne ser vi at $\underline{\gamma}$ regnes ut ved å bruke Matlab-funksjonen *randn*$(n_x, 1)$. Denne funksjonene gir oss tilfeldige standard normalfordelte tall. $UD^{\frac{1}{2}}$ beregnes ved å Cholesky-faktorisere den initielle kovrians $\hat{P}_0$, til dette benyttes *chol*$(\hat{P}_0)'$ [2]. Siden vi allerede har regnet ut første[3] tilstand $\underline{x}_0$, begynner vi beregningen av neste tilstand ved $k = 2$, **linje 21**. Vi setter inn systemmatrisene, pådraget u og den standard normalfordelte prosesstøyen inn i systemlikningen på **linje 22**. Prosesstøyen $\underline{v}_k$ er, slik nevnt tidligere i antagelsene, se delkapittel 2.3.1, normalfordelt med forventningsverdi lik $\underline{0}$

[1] Globale variable blir forklart i neste kapittel

[2] chol returnerer en nedretriangulær matrise R, slik at $R^T R = \hat{P}_0$, vi må transponere R slik at vi får den ønskede $RR^T = \hat{P}_0$

[3] Matlab starter indekseringer av vektorer på 1.

og varians $\delta_{kl}Q$. Siden vi har at $Q = 1$, og $\delta_{kl} = \begin{cases} 0 & \text{if } k \neq l \\ 1 & \text{if } k = l \end{cases}$ beregnes prosesstøyen av Matlab-funksjonen *randn*$(n, 1)$. På **linje 23** vil vi simulere samme systemmodell som på **linje 22**, men uten pådrag fra prosesstøy.

4.1.4 Kalman filter i Matlab

Vi lager det optimale Kalman filteret ved å simulere TO- og MO-likningene (2.20) og (2.21). Listing: 4.3 viser hvordan tids- og måleoppdateringslikningene er implementert i Matlab. Kalmanfilteret trenger gitte initialbetingelser for å starte. Det finnes to tilfeller. Dersom initialbetingelsene $\{\underline{\hat{x}}_0, \hat{P}_0\}$ er oppgitt vil Kalmanfilteret først beregne TO-likningene. Det motsatte vil være tilfellet om initialbetingelsene er $\{\underline{\overline{x}}_0, \overline{P}_0\}$, da vil Kalmanfilteret starte med MO-likningene. I listing: 4.3 beregner funksjonen *simulate()* det optimale Kalmanfilteret. Funksjonen viser her kun det ene tilfellet hvor vi starter med initialbetingelsene $\{\underline{\hat{x}}_0, \hat{P}_0\}$, og derfor beregnes TO før MO.

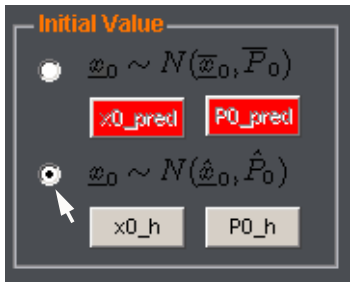
Listing 4.3: Matlab-kode for Kalman filter simulering

```
1  function simulate()
2      global hfig_main;
3
4  if ready2sim() %test om vi kan starte å simulere
5      Fi   = hfig_main.params.Fi;
6      % pluss diverse andre variable
7      % som trengs i denne funksjonen
8      ....
9
10     X      = zeros(n,N);   %Tilstandsvariable
11     X_det  = X;            %determenistiske tilstandsvariable
12     X_est  = zeros(n,N);   %a posteriori tilstandsvariable (estimert)
13     X_pred = zeros(n,N);   %a priori tilstandsvariable (prediktert)
14     P_pred = zeros(n,n,N); %a priori kovarians for prediksjonsavvik
15     P_est  = zeros(n,n,N); %a posteriori kovarians for estimerigsavvik
16     S_est  = zeros(n,N);   %standardavvik for P_est
17     S_pred = zeros(n,N);   %standardavvik for P_pred
18
19     switch hfig_main.system_status.startWith
20        case 'prediction' %start with timeupdate (TO)
21        if_measurement = 'true'; %because we start with x0_hat
22
23        %1. Beregn x0
24         ... %vist tidligere
25
26        %2. Simuler
27         ... %vist tidligere
28
29        P_est(:,:,1) = P0_hatt;
30        S_est(:,1) = sqrt(diag(P_est(:,:,1)));
31        for k = 2:N
32        %Feilanalyse
33        P_pred(:,:,k) = Fi*P_est(:,:,k-1)*Fi' + Ga*Q*Ga';
34        K(:,k)        = P_pred(:,:,k)*H'*inv((H*P_pred(:,:,k)*H' + R));
35        P_est(:,:,k)  = (eye(n) - K(:,k)*H)*P_pred(:,:,k);
36
37        %Standardavvikene
38        S_est(:,k) = sqrt(diag(P_est(:,:,k)));
39        S_pred(:,k) = sqrt(diag(P_est(:,:,k)));
40
```

```
41          %Tidsoppdatering
42          TOsteps = 1/(d*TO);   %TO-frequency koverted to TO_steps
43          if rem(k-1, TOsteps ) == 0
44              switch if_measurement
45                  case 'true'
46                      X_pred(:,k)    = Fi*X_est(:,k-1) + La*u;
47                  case 'false'
48                      X_pred(:,k)    = Fi*X_pred(:,k-1) + La*u;
49              end
50          else
51              if (k==2) % ved første TO, beregn uansett X_pred(:,2)
52                  X_pred(:,2) = Fi*X_est(:,1) + La*u;
53              else
54                  X_pred(:,k) = Fi*X_pred(:,k-1) + La*u;
55              end
56          end
57
58          %Måleoppdatering
59          MOsteps = 1/(d*MO);  %MO-frequency koverted to MO_steps
60          if rem(k-1, MOsteps ) == 0
61              if_measurement = 'true';
62              z              = H*X(:,k) + R*randn(1,1);
63              X_est(:,k)     = X_pred(:,k) + K(:,k)*(z - H*X_pred(:,k));
64          else
65              if_measurement = 'false';
66              X_est(:,k)     = X_pred(:,k); %ingen måling; K=0
67          end                                % => X_est(:,k) = X_pred(:,k)
68          end
69
70          case 'estimation'
71              %hvis Kalman filter syklusen skal
72              %starte med estimering først
73              ...
74      end
75  end
76
77      %Put parameters back to global variables
78      hfig_main.params.Fi = Fi;
79      %update system status
80      hfig_main.system_status.isSimulated = 'yes';
81  else
82      errordlg('Some system parameters are not initialized'...
83               'or the system is not in discrete form',
84               'Simulation Error');
85  end
```

I en funksjon, hvis vi ønsker å bruke globale variable som er tilgjengelige over flere funksjoner og grafiske vinduer, nødvendigvis ikke i samme fil, må disse deklareres med kommandoen *global* etterfulgt av en <handle>[4] som inneholder alle variablene. Dette gjøres på **linje 2**. Før vi kan starte simuleringen og beregningen av Kalmanfilteret trenger vi å kontrollere om all nødvendig data er tilgjengelig. Vi trenger å vite om all innputdata fra brukeren er komplett og tilstede i sine respektive globale variable. For å undersøke dette, og gi en feilmelding dersom noe mangler, kalles funksjonen *ready2sim()* på **linje 4**. Se Listing: 4.4. Funksjonen vil returnere en 1 om alle ønskede variabler har verdier og er klare til å brukes. **Linje 5** og frem til 8 henter inn alle de globale variablene (bl.a. Φ, Λ, Γ, Q, H. m.m.) og legger dem i lokale variable med samme navn, slik at koden skal bli mer oversiktelig. For

[4] bruken av <handle> og *global* forklares i kap. 5



Figur 4.1: GUI for valg av startbetingelse.

oversiktens skyld er ikke alle definisjonene vist i koden. Det er også viktig å legge tilbake de variabler som har endret sin verdi, slik at funksjoner som benytter de globale variablene andre steder for oppdaterte verdier av disse. **Linje 10** til **17** definerer tilstandsvariablene for stokastisk og deterministisk prosess, samt tilhørende matriser for kovarians og standardavvik. Det er lagt opp slik at brukeren skal, gjennom det grafiske brukergrensesnittet se figur 4.1, fortelle applikasjonen om det først skal estimeres eller predikteres, altså om hvilken av Kalmanfilter likning som skal eksekvere først. Dette er som nevnt avhengig av hvilken initialbetingelse man ønsker å starte med eller har tilgjengelig. Denne innputen fra brukeren vil sette feltet *startWith* i **struct** $hfig\_main.system\_status.startWith$ til enten strengen 'prediction' eller 'estimation'. Vi bruker på **Linje 19** en switch-konstruct på variabelen *startWith* (ofte kalt *field* når den er en del av en struct) for å avgjøre hvilke av de to strengene som er satt til *startWith*. Som nevnt har vi antatt initialbetingelsene til å være $\{\hat{\underline{x}}_0, \hat{P}_0\}$ og skal derfor starte med en TO (prediction). Dette vil føre til at *case* på **linje 20** vil reagere og koden som følger etter helt frem til neste *case* vil eksekvere, **linje 21** til **68**.

Variabelen *if_measurement* på **linje 21** blir brukt til å teste når vi har fått inn en måling. Den blir satt til *true* dersom vi har en måling, dette antar vi at vi har ved første tidsskritt, ellers settes den til *false*. Beregning av $\underline{x}_0$ og simulering er vist tidligere i listing: 4.2. En av fordelene med Kalmanfilteret er at vi kan utføre feilanalyse før målingene foreligger. Siden $\hat{P}_k$ og $\overline{P}_{k+1}$ ikke trenger noen målinger for å beregnes, se likning (2.20) og (2.21). Først legger vi inn den initiale kovariansmatrisen for estimerigsavviket, og dens standardavvik i deres respektive matriser $\hat{P}$ og $\hat{S}$ på **linje 29** og **30**. For å beregne Kalmanfilter verdiene for hvert tidsskritt benyttes det en *for*-løkke som itererer fra andre tidsskritt (første har vi initielle verdier for $\{\hat{\underline{x}}_0, \hat{P}_0\}$)[5] og opp til siste, dvs. N. Som nevnt tidligere kan feilanalyse utføres på forhånd, og dette gjøres i **linje 33, 34** og **35**. Matlab kommandoen *eye* er en identitetsmatrise med $dimensjon = n$. Standardavvikene for $\hat{P}$ og $\overline{P}$ på **linje 38** og **39**. Både TO og MO har frekvenser for når oppdateringene skal utføres. For å regne utfra oppdateringsfrekvensen hvilken tidsskritt oppdateringen skal forekomme på bruker vi denne fremgangsmåten:

[5]Merk: Matlab starter indeksering av vektorer/matriser fra 1, og ikke 0

$$\begin{aligned} \textit{Tidsoppdateringsfrekvens}: \qquad f_{TO} \Rightarrow T_{TO} &= \frac{1}{f_{TO}} \\ TO_{at\_step} &= \frac{T_{TO}}{\Delta t} = \frac{1}{\Delta t f_{TO}} \end{aligned}$$

dette er gjort på **Linje 42**. Siden både TO og MO vil forekomme ved bestemte tidsskritt, ønsker vi å skille mellom hendelsene for når en TO skal gjennomføres, og det tilfellet når det ikke skal. Til dette bruker vi en if-test sammen med Matlab-kommandoen rem[6] på **linje 43**. Når den iterative tidsskrittelleren (k-1) har nådd TO_{steps}, vil $rem(k-1, TO_{steps})$ bli lik 0. Vi er da klar for en TO. Da vil koden som følger if-testen, **linje 44** og **49**, eksekvere. Før vi kan beregne TO må vi undersøke om det har foreligget en måling ved forrige tidsskritt. Dersom dette er tilfellet vil $if\_measurement$ være *true*. For å teste dette bruker vi switch på $if\_measurement$, **linje 44**. Hvis utfallet av switch-testen blir **linje 45**; $case = 'true'$, skal vi beregne TO likningen for $\underline{\overline{x}}_k$ utfra den forrige estimerte (a posteriori) $\underline{\hat{x}}_{k-1}$:

$$\underline{\overline{x}}_k = \Phi \underline{\hat{x}}_{k-1} + \Lambda u$$

Dersom tifellet blir at $case = 'false'$, **linje 47**, har det ikke blitt gjort en måling ved forrige tidssktitt (k-1) og vi er nødt til å bruke den forrige predikterte (apriori) $\underline{\overline{x}}_{k-1}$ til å projisere frem i tid den nye predikterte $\underline{\overline{x}}_k$:

$$\underline{\overline{x}}_k = \Phi \underline{\overline{x}}_{k-1} + \Lambda u \tag{4.2}$$

Hvis *else* på **linje 50** er utfallet av if-testen, betyr det at vi ikke har en tidsoppdatering ennå. I dette tilfellet har vi samme situasjon som i 4.2. Vi er nødt til å bruke tidligere predikterte verdier **linje 54**. Vi ønsker at ved første tidsskritt (k=2), uavhengig om vi har TO eller ikke på det tidsskrittet, så skal initialbetingelsen $\underline{\hat{x}}_0$ projiseres frem i tid slik at vi har den første predikterte $\underline{\overline{x}}_1$ tilgjengelig. For dette bruker vi en if-test på **51**. MO følger samme mønster som TO. På **59** konverteres frekvensen for MO til den tilsvarende tidsskritt, MO_{steps}. Vi bruker rem i kombinasjon med if på **linje 60** slik at testen skal bli sann ved hver multiple tidsskritt av MO_{steps}. Når vi har dette tilfellet vil MO-likningene på **linje 62** og **63** beregnes. Vi stter også $if\_measurement = 'true'$ for å indikere at en måling har blitt gjort. Dersom tidsskrittet (k-1) i $rem(k-1, MO_{steps})$ er *ulik* 0 har vi ingen måling og eksekveringen vil hoppe til **linje 64**. Videre settes $if\_measurement = 'false'$. Merk at ingen måling betyr at målematrisen H i Kalmanfilter (KF) forterkningen $K_k = \overline{P}_k H^T (H\overline{P}_k + R)^{-1} \hat{P}_k$ settes til null. Dette fører til at $K_k = \underline{0}$. Setter vi dette inn i MO-likningen for aposteriori estimatet $\underline{\hat{x}}_k$ får vi:

$$\underline{\hat{x}}_k = \underline{\overline{x}}_k$$

dette er brukt på **linje 66**.

Listing 4.4: Matlab-kode for funksjonen ready2sim.m

[6] rem gir restleddet etter en divisjon.

```
1  function answer = ready2sim()
2      global hfig_main;
3
4      %check if all needed parmeters have values
5      if isequal(hfig_main.system_status.isDiscrete,'yes') ...
6         &&   ...
7         ~isempty(hfig_main.params.F)    ...
8         &&   ...
9         ~isempty(hfig_main.params.Fi)   ...
10        &&   ...
11        ~isempty(hfig_main.params.La)   ...
12        &&   ...
13        ~isempty(hfig_main.params.Ga)   ...
14        &&   ...
15        ~isempty(hfig_main.params.u)    ...
16        &&   ...
17        ~isempty(hfig_main.params.d)    ...
18        &&   ...
19        ~isempty(hfig_main.params.t0)   ...
20        &&   ...
21        ~isempty(hfig_main.params.t_f)  ...
22        &&   ...
23        ~isempty(hfig_main.params.MO)   ...
24        &&   ...
25        ~isempty(hfig_main.params.TO)
26
27        if isequal(hfig_main.system_status.startWith,'estimation') ...
28           && ...
29           ~isempty(hfig_main.params.P0_pred)   ...
30           && ...
31           ~isempty(hfig_main.params.x0_pred)
32                 answer = 1;
33        else if isequal(hfig_main.system_status.startWith,'prediction') ...
34           && ...
35           ~isempty(hfig_main.params.P0_hatt)   ...
36           && ...
37           ~isempty(hfig_main.params.x0_hatt)
38                answer = 1;
39            else
40                answer = 0;
41            end
42        end
43     else
44         answer = 0;
45     end
```

Som nevnt i beskrivelsen for listing: 4.3 vil funksjonen *ready2sim()* undersøke eksistensen av nødvendige parametere for simulering og KF beregning. Alle systemmatriser ligger i en <handle> $hfig\_main$. For å få tilgang til disse må vi på **linje 2** eksplisitt fortelle funksjonen at de ligger i $hfig\_main$. Dette gjøres med kommandoen global. Vi starter først med å undersøke om systemet er diskretisert. Dersom en diskretisering av systemet er utført på forhånd, vil variabelen *isDiscrete* være satt til $'yes'$. *isDiscrete* er plassert i en *struct*, *system_status*. For å kontrollere om *isDiscrete* har verdien $'yes'$ bruker vi Matlab-kommandoen isequal sammen med en if-test på **linje 5**. isequal returnerer en 1 dersom testen er vellykket og vi kan fortsette med if-testen. Det neste vi ønsker å undersøke er om alle nødvendige parametere (F, Φ, Λ, osv.) *ikke* er tomme. **Linje 7** til **25** gjør dette for oss. Her benyttes kommandoen $\sim$ isEmpty på de forskjellige parametrene. Vi har tilført en negasjons-operator, $\sim$, foran

isEmpty for å teste på det tilfellet at matrisene ***ikke*** er tomme. Det gjenstår nå å kontrollere initialbetingelsene for KF. Som nevnt tidligere vil variabelen *startWith* enten være satt til strengen $'estimation'$ eller $'prediction'$ avhengig av om vi henholdsvis starter med $\{\overline{x}_0, \overline{P}_0\}$ eller $\{\underline{\hat{x}}_0, \hat{P}_0\}$. **Linje 27** og **33** tester på dette. Hvis vi skal starte med MO-likningene (*startWith* er lik $'estimation'$) må $P0\_pred$ ($\overline{P}_0$) og $x0\_pred$ ($\overline{x}_0$) ikke være tomme. Vi undersøker dette på **linje 29** og **31**. Tilsvarende test utføres i det tilfellet hvor vi skal starte med TO-likningene (*startWith* er lik $'prediction'$). Funksjonen $ready2sim()$ har kun en utgangsvariabel, *answer*. Dersom systemet er klart til å simuleres og KF beregnes vil *answer* bli satt til 1 og returnert, dersom det er mangler vil *answer* settes til 0.

Kapittel 5

Oppbyggning av GUI program i MATLAB

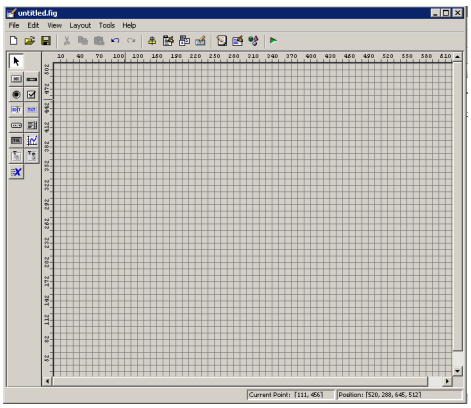
5.1 MATLAB GUIDE

Til å utforme det grafiske brukergrensesnittet for Kalmanfilter applikasjonen har GUIDE blitt benyttet. GUIDE - Graphical User Interface Development Environment [7], er et utviklingsverktøy bygd inn i Matlab for enkelt å utvikle GUI programmer. Utformingen av GUI programmer kan gjøre ved å bruke to separate metodologier. Enten ved lav-nivå M-fil koding eller ved å benytte et høgnivå grafisk innretning som GUIDE, se figur 5.1. I GUIDE får brukeren tilgang til alle GUI funksjoner som Matlab har tilgjengelig.

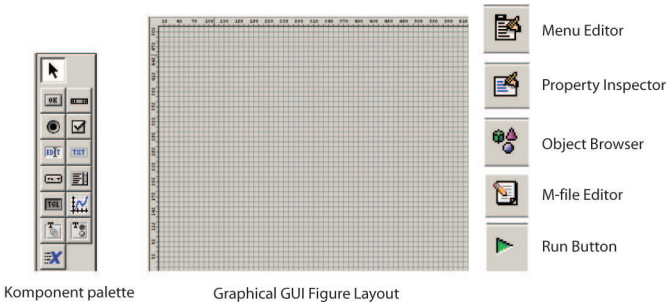
GUIDE består av flere viktige komponenter som brukes til å utforme det grafiske brukergrensesnittet. Figur 5.2 viser de forskjellige delene kort beskrevet her:

- **Component Palette:** Denne paletten gir tilgang til forskjellige grafiske objekter; knapper, akser, lister, m.m.).
- **Graphical GUI Figure Layout:** Er et grid-basert layout, som ved hjelp av dra-og-klikk brukes til å plassere ut forskjellige komponenter og dermed forme gui grensesnittet en ønsker.
- **Menu Editor:** Er et eget grensesnitt for å lage menyer til gui programmet.
- **Property Inspector:** En liste med opsjoner for det tilhørende grafiske objektet. Man kan endre oppførsel (position, aktivt/inaktivt, m.m.) og utseende (farger, fonter, m.m.) til de forskjellige grafisek objektene.
- **Object Browser:** Et hierarkisk overblikk over forskjellige grafiske objekter i programmet.
- **Run Button:** Brukes til å kjøre gui programmet for evaluering og testing.

Ettersom man plasserer og former det grafiske utseendet med GUIDE, genereres det automatisk en M-fil i bakgrunnen. Denne M-filen inneholder bl.a. alle funksjonene relatert til hvert enkelt grafisk objekt i programmet. Vi kaller



Figur 5.1: GUIDE med komponenter.



Figur 5.2: Diverse deler av GUIDE.

disse funksjonene **Callback**-funksjoner og de kalles når et objekt blir aktivert, f.eks ved at en knapp trykkes inn, eller at et element velges i en liste. Alle objekter har egne Callback-funksjoner. Det finnes forskjellige typer Callback-funksjoner avhengig av hvilken type grafisk objekt det er, og hvilken type handling som ble utført på/av de grafiske objektene. Listen under oppsumerer strukturen av en M-fil kode generert av GUIDE.

1. Main Function:
2. Opening Function:
3. Output Function:
4. Callback Functions:

Main Function

GUIDE lager en main-funksjon med samme navn som M-filen og fig[1]-filen. Funksjonen har standard inn- og utgangsvariable som henholdsvis heter *varargin* og *varargout*. Disse variablene brukes for å hente inn og sende ut data fra

[1] .fig er den filen som lagrer arbeidet gjort i GUIDE.

funksjonen på samme måte som alle andre funksjoner i Matlab. Formatet på denne funksjonen er slik:

$$function\ varargout = main\_gui(varargin)$$

$< main\_gui >$ er det navnet vi velger på filen laget i GUIDE.

Opening Function

Opening-funksjonen kalles rett før programvinduet blir synlig for brukeren. Vi kan i denne funksjonen plassere variabler (f.eks globale), og gjøre diverse endringer på gui programmet som vi ønsker skal utføres før brukeren får tilgang til programvinduet. Opening-funksjonen har denne formen:

$$function\ varargout = main\_gui\_OpeningFcn(hObject, eventdata, handles, varargin)$$

hObject (som også er en handle) kan ses på som en peker til figuren (programvinduet) denne funksjonen tilhører. Ved å bruke hObject sammen med prikk-operatoren (.), får en tilgang til alle dens grafiske opsjoner. **eventdata** er reservert og ment for å brukes i senere Matlab versjoner. **handles** er en spesiell datastruktur som inneholder handles, se delkapittel 5.2.2 og brukerdata. **varargin** er input-variabler fra main-funksjonen.

Output Function

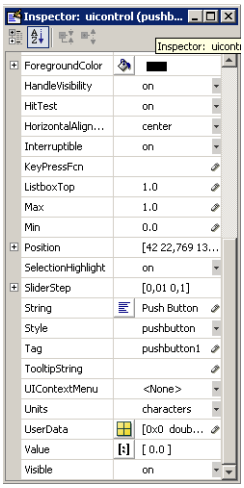
Output funksjonen har som oppgave å returnere ønskede variabler når main-funksjonen avsluttes. Formen er ganske lik den for "Opening Function", og med samme input parametere:

$$function\ varargout = main\_gui\_OutputFcn(hObject, eventdata, handles, varargin)$$

Callback Function

En Callback er en funksjon som kalles hver gang et event inntreffer på f. eks et grafisk objekt. Som sakt tidligere finnes det forskjellige type Callback-funksjoner, *Graphics Object Callbacks*, *User Interface Object Callbacks* og *Figure Callbacks*[Matlab Help]. Som et eksempel kan vi lage en Callback-funksjon for når en knapp (pushbutton) trykkes i vår *main_gui* på denne måte:

$$function\ pushbutton\_Callback(hObject, eventdata, handles)$$



Figur 5.3: Eksempel på egenskapene til en pushbutton-objekt

5.2 HG objekter og Handles

For å forstå oppbygningnen av GUI-programmet for Kalman filteret er det nødvendig å vite hva HG-objekter er og hva de inneholder. Det er også viktig å vite hvilken type datastruktur som benyttes for å organisere disse HG-objektene. Konseptene vil kun bli oppsumert her og leseren henvises til [7] for mere detaljer og eksempler.

5.2.1 Handle Graphics Objects: HG-Objekter

Som andre objektorienterte språk benytter også Matlab et hierarki av objekter som kalles *Handle Graphics Objects* (**HG objects**). Disse grafiske objektene, som vi også har nevnt tidligere, brukes til å bygge alle mulige grafiske brukergrensesnitt. Et enkelt eksempel kan være en figur generert av plot-kommandoen i Matlab. Denne figuren er da et HG-objekt. Andre eksempler er knapper, lister, editbox, akser, figurvinduer, m.m. . Hver HG-objekt inneholder egenskaper som definerer utseendet og oppførsel til det grafiske objektet. Figur 5.3 er hentet fra GUIDE grensesnittet og lister opp alle egenskapene til et pushbutton HG-objekt.

Disse egenskapene er innkapslet i HG-objektet og for å få tilgang til disse må vi gjennom en handle. Mer spesifikt må vi få tak i den unike identifikatoren kalt en ***Tag*** som hvert HG-objekt har, og som er en del av egenskapene til objektet. Tag-feltet er en streng som nås gjennom en handle. Dersom en pushbutton har en Tag: *pushbutton_Cancel*, kan denne fås tak i ved å først skrive navnet på handle, etterfulgt av prikk-operator og så navnet på Tag, altså *handle.pushbutton_Cancel*.

5.2.2 Handles

Handles er essensielt for gui-programmering i Matlab. Og er en måte å strukturere data på. Enhver figur med tilhørende HG-objekter vil lagre all informasjon om seg selv i en felles handle. På denne måten kan en handle ses på som en konteiner med all data samlet på et sted. Fordelen med dette er at vi kun trenger å passere et enkelt parameter (i dette tilfellet en handle) mellom funksjoner. Alternativet kunne vært å oppsummere alle de forskjellige egenskapene i funksjonens parameterliste. Dersom en ønsker (slik det har blitt gjort i Kalmanfilter applikasjonen) å ekspandere tilgjengeligheten til egendefinerte varabler kan disse legges inn i den samme handle på denne måten:

$F = [12; 34];$
handle.F; *F er lagt inn i handle*

5.2.3 Globale variable

Det mest vanlige ved utvikling av en gui-applikasjon i Matlab er å bruke en lokal handle datastruktur. Og på denne måten kommunisere forskjellige parameter mellom funksjoner i samme programvindu (figure). Dette er hva GUIDE bruker som standard ved automatisk kodegenerering av M-filen. Metoden er veldig effektiv for å lage applikasjoner basert på et enkelt figurvindu. Men dette begrenser mulighetene hvis en ønsker å utvide applikasjonen til å inneholde flere programvinduer (figurer). Måten denne begrensingen elimineres på er ved å bruke *globale* variable som strekker seg over alle funksjoner og figurer. For at dette skal være mulig må de globale variablene deklareres hver gang i den funksjonen der de trengs. Måten dette gjøre på er slik:

```
1  global my_handle;
2
3  function figur1_Callback(hObjekt, eventdata, handles)
4          global my_handle;
5
6          my_handle.F = ones(2);
7
8  function figur2_Callback(hObjekt, eventdata, handles)
9          global my_handle;
10
11         my_handle.G = zeros(2);
12         my_handle.F = zeros(2);
```

Linje 1 deklarerer en global handle-struktur (kan ses på som en variabel) slik at innholdet i denne kan gjøres tilgjengelig i andre funksjoner også. På **linje 3** opprettes det en funksjon som ønsker å benytte seg av den globale variabelen. Den må derfor eksplisitt deklarere den (selv om dette er gjort tidligere) på **linje 4**. Funksjonen ønsker å ekspandere innholdet i *my_handle*, og gjør dette på **linje 6** ved å tilføre variabelen *F*. Funksjon nummer to leser inn den globale handle-strukturen som nå også inneholder *F*. Den legger til variabelen *G* og

initialiserer den. Deretter endrer den F på **linje 13**, noe som virkelig viser at F er global siden denne opprinnelig ble deklarert utenfor funksjon 2, men ble endret et annet sted.

5.3 Kalmanfilter applikasjon

Dette kapittelet omhandler Kalmanfilter applikasjonen. En forklaring i form av et brukermanual for hvordan grensesnittet skal brukes vises. Den andre delen av kapittelet vil oppsummere alle funksjonene som Kalmanfilter applikasjonen er bygd opp av. En kort forklaring av hva funksjonene gjør og hvilke tjenester de tilbyr andre funksjoner vil bli presentert. For å få et mer oversiktelig bilde av Matlab-koden henvises det til vedlegg A der hele programkoden er lagt ved.

5.3.1 Manual

For å starte opp Kalmanfilter applikasjonen må Matlab kjøres i bakgrunnen. Gjør følgende for å starte applikasjonen:

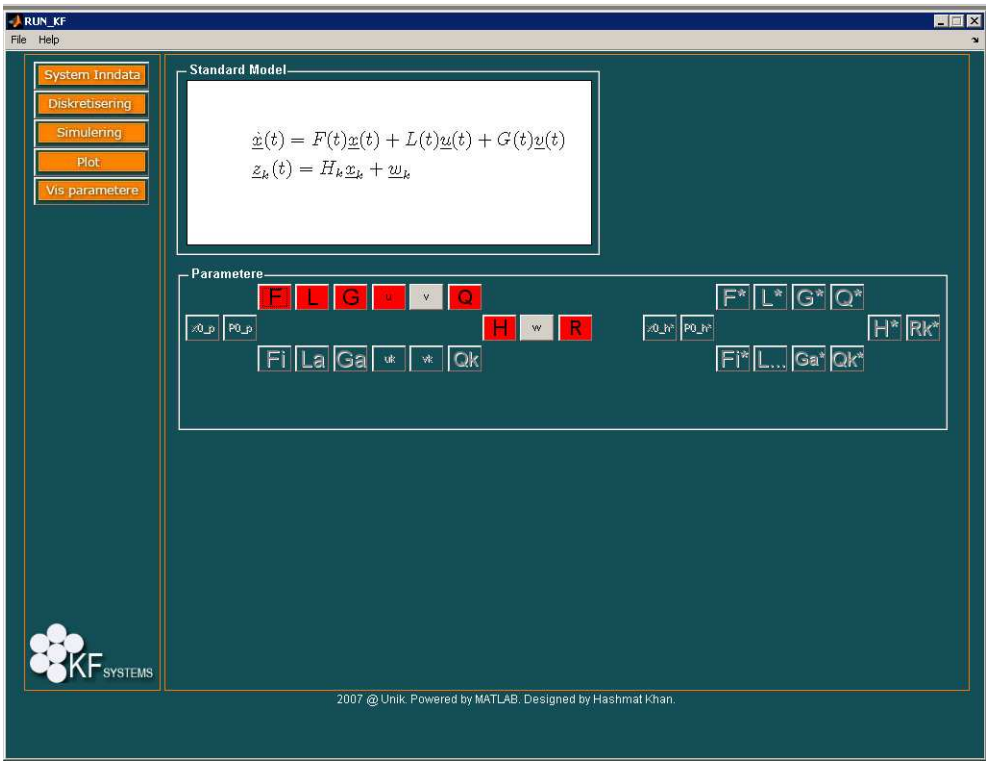
1. Start Matlab.
2. Velg *Current Directory* og sett den til mappen med alle Kalmanfilter filene.
3. I Matlab *Command Window* skriv **RUN_KF** og trykk Enter.

Det grafiske brukergrensesnittet er vist i figur 5.4. Når applikasjonen startes opp vil brukeren få dette vist frem. Til vestre i programvinduet har vi menyen hvor brukeren kan velge hvilken oppgaver som skal utføres. Ved å trykke på en av knappene i menyen vil høyre del av programvinduet endre seg. Hver knapp i menyen har sin egen grensesnitt mot brukeren, og dette grensesnittet avhenger av hva som skal utføres.

Meny: System Inndata

Denne meny-knappen har som oppgave å vise brukeren grensesnittet for innlesing av systemparametere, se figur 5.4. De forskjellige parametrene er laget som knapper man kan presse. I denne versjonen er kun de parametrene som brukes til diskretisering, simulering og Kalmanfilter beregn mulig å velge. Parametere som ennå ikke har noen verdi vil være markert med rød. Parametrene for prosesstøy $\underline{v}_k$ og målestøy $\underline{w}_k$ er farget grå og de er innaktive. Dette skyldes av vi antar på forhånd at støyskildene er hvite og de implementeres automatisk inn i simuleringen. For å lese inn en systemmatrise, som f. eks F gjør følgende:

1. Dersom du ikke har aktivert *System Inndata*-knappen så trykk først på denne.
2. For å lese inn F trykk på knappen merket F.
3. Et vindu kalt *Parameter Options* for innlesing av parametere vil åpne seg, se figur 5.5.



Figur 5.4: Hovedvindu for Kalmanfilter applikasjon

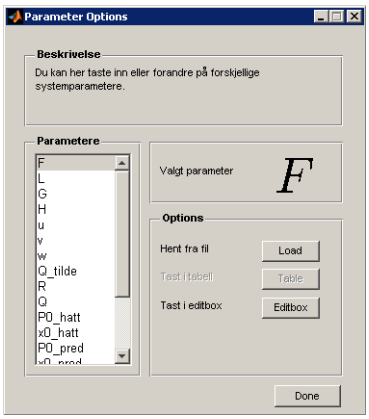
Parameter Options-vinduet er hvor vi kan lese inn forskjellige parametere. Som man ser fra figur 5.5, kan dette gjøres på to måter. Ved å trykke på knappen Load vil man få opp et standard *File Open*-vindu for å lese en fil. I vårt tilfelle er det kun mulig å lese M-filer med alle parametere skrevet inn. Det forutsettes at parametrene har samme navn som de i programkoden. Se Vedlegg B for hva som har blitt benyttet[2]. Den andre måten er ved å trykke på knappen Editbox. Vi vil da få opp en editbox hvor vi kan taste inn verdiene vi ønsker i vanlig Matlab-syntaks. I vinduet Parameter Option er det også en liste som indikerer hvilken parameter som er valgt. Uten å gå ut av dette vinduet er det mulig å velge en annen parameter ved å trykke i listen og legge inn verdi for denne også. Hvilken parameter en har markert i listen vises i Latex blokken ved siden av listen. Denne endrer også symbol avhenging av valgt parameter.

4. Bruk vinduet *Parameter Options* til å legge inn verdier for parametrene. Som et test eksempel last inn filen *system_init.m*[3].

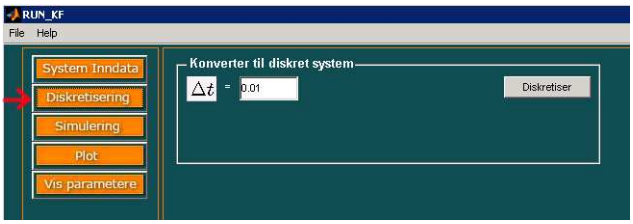
5. Trykk *Done* når du er ferdig.

[2]Dette gjelder kun hvis man skal lese parametrene fra fil.

[3]Parameterverdiene i denne filen er hentet fra prosjektoppgaven i Stokastiske systemer, UNIK4500. [9]



Figur 5.5: Grensesnitt for parameter innlesing fra Fil eller Textbox.



Figur 5.6: Grensesnitt for diskretisering.

Ettersom parametrene legges inn via grensesnittet for *System Inndata*-knappen, vil fargen på de parametrene som nå inneholder en verdi bli grå. Har du glemt å lese inn en parameter vil den fortsatt være markert med rød. Sørg for at alle systemmatriser er innlest før du går videre.

Meny: Diskretisering

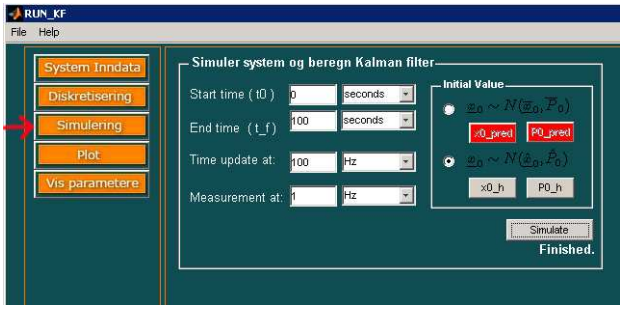
For å diskretisere systemet etter at du har lest inn alle nødvendige parametere gjør følgende:

1. Dersom du ikke har aktivert *Diskretisering*-knappen så trykk først på denne, se figur 5.6.

2. Bestem et tidsintervall Δt for diskretisering og tast denne inn i editboksen.

   Du vil merke at tekstfeltet for Δt vil være rød dersom ingen verdi er lagt inn. Har du i forrige steg gitt en verdi til Δt vil denne allerede være markert med hvit og ligge i editbox.

3. For å starte diskretiseringen trykk på knappen Diskretiser.



Figur 5.7: Grensesnitt for simulering og Kalman filter beregning.

Meny: Simulering (Kalmanfilter beregning)

Grensesnittet vist i figur 5.7 for simulering og Kalmanfilter (KF) beregning har flere opsjoner som en kan endre på. Før vi starter beregningene er det nødvendig å oppgi for hvilken tidsperiode vi ønsker å simulere og beregne KF verdier for. Editbox for starttidspunkt t_0 og sluttidspunkt t_f brukes til dette. Videre er det nødvendig at vi oppgir hvilken oppdateringsfrekvenser vi ønsker på tids- og måleoppdatering. To egne editbox er tilgjengelig for dette. Foreløpig er det begrenset til at tider oppgis i sekunder og frekvenser i Hz. For å legge inn initialbetingelsene finnes det et eget panel i samme grensesnitt til dette kalt *Initial Value*. Man velger først med et museklikk hvilken initialbetingelse man ønsker å starte med ved å klikke på en av to mulige radiobuttons. Avhengig av hvilken radiobutton man har valgt vil to tilhørende pushbuttons for henholdsvis $\underline{x}$ og P aktiveres. Vi kan nå klikke på disse for å legge inn verdier for initialbetingelsene. Det er mulig at dette er gjort i steg en når andre systemparametere ble lest inn fra fil, i så fall vil fargen på disse to knappene være grå. Vi kan nå starte simuleringen og KF beregningen. Gjør følgende:

1. Dersom du ikke har aktivert *Simulering*-knappen så trykk først på denne, se figur 5.7.
2. Legg inn ønskelige verdier for Starttid t_0 , Sluttid t_f, Tidsoppdateringsfrekvens TO og Måleoppdateringsfrekvens MO.
3. Velg ønskede initialbetingelser.
4. For å utføre beregningene trykk på knappen Simuler.

Meny: Plot

For å analysere verdiene beregnet under simulering og Kalmanfilter utregning kan Plot-grensesnittet brukes, vist i figure 5.8. Grensesnittet består av to deler. En editbox med mulighet til å skrive inn alle ønskelige plot i Matlab-syntax. Det er svært viktig at brukeren benytter de samme variabelnavnene som er brukt i applikasjonen. Disse er også oppgitt i tekstfeltet over slik at man har dem lett tilgjengelig. For en mer detaljert oversikt henvises det til Vedlegg B. Den andre delen er plot-knappen. Denne leser av innholdet i editbox og evaluerer innholdet. For å plotte gjør følgende:

Figur 5.8: Grensesnitt for plotting og analysering.

1. Dersom du ikke har aktivert *Plot*-knappen så trykk først på denne, se figur 5.8.
2. Skriv inn ønsket plot-kommando.
3. For å evaluere det som ble skrevet trykk på Plot.

Dersom syntaksen er riktig og Matlab aksepterer variabelnavnene vil det lages en figur med plot. For hver gang man plotter vil det lages en ny figur slik at man beholder den forrige. Disse kan lagres på standard Matalb måte og hentes frem senere for videre analyse.

Meny: Vis parametere

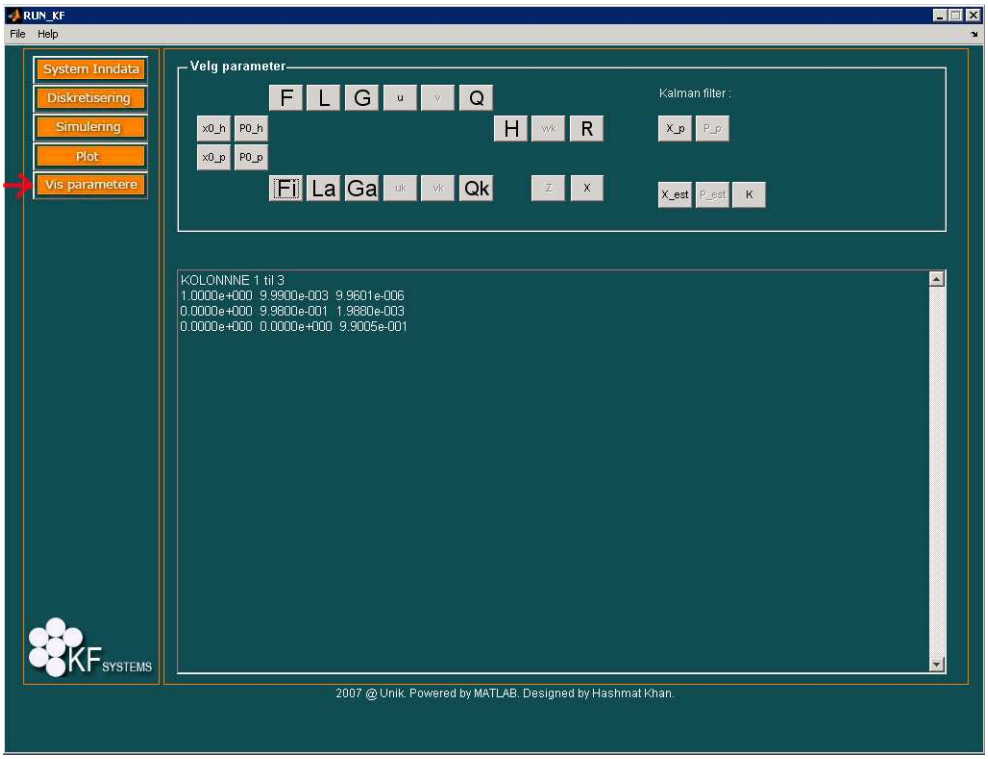
Vi ønsker å se hvilken verdier som ligger inne i viktige parametere. Disse verdiene kan komme fra innlesning av parametere fra fil, ved simulering eller at de stammer fra KF utregninger. Ved å trykke på meny-knappen *Vis parametere* vil grensesnittet som muliggjør visning av parameterverdier dukke opp, se figure 5.9. Knappene for å vise innholdet i kovariansmatrisene er deaktivert siden algoritmen for å skrive ut verdiene til en editbox ikke er utvidet til å gjelde for dimensjoner større en to. For å se verdiene for forskjellige parametere gjør følgende:

1. Dersom du ikke har aktivert *Vis parameter*-knappen så trykk først på denne, se figur 5.9.
2. Trykk på en parameter (pushbutton) du ønsker å se.

For store matriser med kolonner som strekker seg utover bredden til editbox vises disse lenger ned. Dette er den samme måten Matlab viser store matriser på. Det skrives også ut i tekst hvilken kolonnenummer som vises til enhver tid.

5.3.2 Funksjoner i programkoden

En oppsummering av de funksjonene som er konstruert for å bygge opp Kalmanfilter applikasjonene vil bli listet opp her. Det vil bli gitt forklaringer på hvilken tjenester disse funksjonene tilbyr og hva deres hovedoppgave er. I vedlegg A finner man programkoden med alle funksjonene og med detaljerte kommentarer underveis som en kan referere til. Funksjonen som inngår:



Figur 5.9: Grensesnitt for fremvisning av parameterinnhold.

1. RUN_KF: Dette er hovedfunksjonen som bygger opp selve programvinduet for Kalmanfilter applikasjonen, se figur 5.4. Den definerer alle systemparametrene, og samler de i en datastruktur som kan nås gjennom en global handle kalt hfig_main. Den har ansvaret for å hente inn jpeg-bilder og plasserer disse på knapper og andre diverse objekter for å oppnå ønsket utseende. Funksjonen *RUN_KF* har Callback-fuksjoner som venter på å bli kalt ved en event, som f. eks ved at brukeren trykker på en knapp. Det er fra denne funksjonen alle andre funksjoner blir kalt, enten direkte eller gjennom andre funksjoner.

2. place_colorANDText: Dette er en hjelpefunksjon til RUN_KF og har bl.a. som oppgave å gjøre synelig de delene av grensesnittet som vi ønsker å vise ved programstart. Den plasserer også text på en akse i grensesnittet for meny-knappen System Inndata, figur 5.4. Teksten den plasserer er likninger for prosessmodell og målemodell.

3. read_params: Når vi skal lese inn parametere må dette gjøres gjennom grensesnittet vist i figur 5.5. Denne funksjonen tar seg av dette. Verdier kan enten hentes inn fra fil, eller tastes in direkte i en editbox.

4. discrete: Brukes for å diskretisere de kontinuerlige systemparametere. Før den starter diskretiseringen undersøker den om alle de nødvendige parameter ikke er tomme. I så fall får bruker beskjed om dette via

en dialog-vindu for feilmeldinger. Funksjonene som blir brukt for å diskretiserer er $kp2dpGa$ og $kp2dpGa$.

5. simulate: Funksjonen skal simulere systemet og beregne Kalmanfilter verdiene. Se kap 4.1.4 for en beskrivelse.

6. update_parameter: Benyttes for å oppdatere diverse parametere som inngår i applikasjonen. Også tilhørende knapper blir oppdatert med riktig farge. Grå hvis en parameter har fått tildelt en verdi og rød dersom den er tom.

7. visParameter: Funksjonen tar inn en parameter. Avhenging av hvilken systemvariabel denne parameteren referer til, settes den til A, og skrives ut til en editboks på skjermen. Brukes i grensesnittet vist i figur 5.9.

8. fload_parameter: Leser inn fra fil alle parametere og legger dem i current workspace. Bruker kommandoen "who" til å finne frem navnene på alle parametere som ble lest inn i denne workspace. Dersom de har samme navn som de parametrene (variable) i vår egen applikasjon, brukes da de innleste verdiene til å initialisere diverse og tilhørende variable i vårt eget program.

Kapittel 6

Videre arbeid

- Kalmanfilter applikasjonen har stor potensial for videreutvikling. Det grafiske brukergrensesnittet er konstruert slik at menysystemet kan utvides med ekstra funksjonalitet ved å legge til flere meny-knapper. Hver knapp kan så styre sitt eget grensesnitt. Andre funksjoner som er nyttig å ha med og som ikke ble med i denne versjonen kan være Monte Carlo simulering, Kovariansanalyse og feilbudsjettering. Det er også ønskelig å utvide det som allerede har blitt implementert. Applikasjonen antar tidsinvarians slik at diskretisering, simulering og Kalmanfilter utregninger er basert på denne antagelsen. For å få et mer komplett design og analyse verktøy for Kalmanfiltere må funksjonaliteten utvides til å håndtere tidsvariante systemer.
  Kun et optimalt Kalmanfilter har blitt implementert, videre arbeid må gjøres slik at programmet også kan designe et suboptimalt Kalmanfilter. Til dette har det blitt plassert knapper for en filtermodell i grensesnittet for *System Inndata*-knappen, se figur 5.4. Her er det ønskelig at man skal kunne flytte parametrene for systemodel som ligger i venstre halvplan over til parametrene for filtermodell i venstre halvplan. En knapp med pil fra venstre til høyre kan f. eks lages for å utføre dette. Man skal også kunne gå inn og endre filtermodellen ved å klikke på tilhørende knapper og forandre likheten mellom systemmodell og filtermodell. Som en ser fra figuren er venstre del deaktivert siden denne ikke brukes ennå. Hvert grensesnitt med tilhørende meny-knapp har ekstra plass som en kan bruke til å utvide med andre funksjoner.

- Optimalisering av algoritmene bør gjøres slik at utregninger går fortere. For små matriser er dette ikke nødvendig men hvis matrisene er store kan det hende at utregningene går tregere.

- I grensesnittet for *Vis parametere*-knappen ligger det en algoritme bak som skriver ut innholdet til parametrene til en editbox. Denne må utvides til å skrive ut matriser med flere dimensjoner enn to slik at sekvenser med kovarianser kan vises. Siden muligheten for dette ikke er tilstede i denne versjonen har knappene for $\hat{P}_k$ $\overline{P}_k$ blitt deaktivert.

- Det er ønskelig at applikasjonen skal kunne kjøres uten å åpne/ha Matlab. En slik mulighet finnes ved at man kan kompilere programkoden

med Matlab sin egen *mcc*-kompilator. Ulempen er at ikke alle innebygde Matlab-funksjoner vil være tilgjengelig. Hvilken funksjoner dette gjelder og om det er mulig å bygge vårt program rundt denne begrensningen burde undersøkes.

Kapittel 7

Konklusjon

- Denne Kalmanfilter applikasjonen har vist seg å være et velegnet verktøy til å designe og analysere et Kalmanfilter. Siden prosessen med å designe et slik filter kan strekke seg fra fra noen dager til flere uker avhengig av hvilken applikasjon man lager, vil tilgjengeligheten av et slikt verktøy redusere kostnadene (tid, penger, arbeidskraft, osv.) relatert til slik utvikling. Som sakt må denne Kalmanfilter applikasjonen utvides med mer funksjonalitet slik at man kan designe flere typer Kalmanfiltere og gjøre grundigere analyser. De numeriske metodene som har blitt benyttet i denne rapporten er velprøvde metoder og egner selv veldig godt til dette. Siden vi har brukt Matlab som jobber med dobbelpresisjon oppnår vi også ganske god nøyaktighet.
- Det har vist seg at Matlab egner seg veldig godt til små grafiske applikasjoner, men hvis man ønsker å utvide programmet til å arbeide med faner, altså forskjellige grafiske grensesnitt i et og *samme* vindu, er dette en veldig uoversiktelig prosedyre i GUIDE. Ulempen har vært at man fort mister oversikt over alle paneler og andre grafisk objekter i designvinduet for GUIDE. Det hadde vært ønskelig med *Layers* hvor en kan skjule de paneler og grafiske objekter som en ikke trenger å se. Dette er mulig under kjøring slik det har blitt gjort i vår applikasjon ved å skru av synlighetsparameteren for hvert grafisk objekt. Dette lar seg ikke gjøre under selve designfasen i GUIDE. Alternativet kan være at man konstruerer det grafiske brukergrensesnitte i et annet språk, f. eks C++ og bruke MATLAB C/C++ Math Library til å utføre beregningene. Det finnes egne pakker og biblioteker for å integrere Matlab inn i andre språk (Java og Python).

Bibliografi

[1] Arthur Gelb. Applied Optimal Estimation, 1974. THE M.I.T. PRESS

[2] Greg Welch og Gary Bishop. An Introduction to the Kalman Filter, SIGGRAPH 2001 Course.

[3] Peter S. Maybeck. Stochastic Models, Estimation and Control. Vol 1, 1979. ACADEMIC PRESS, New York, San Fran, London.

[4] Oddvar Halligstad. "Standardmodeller og Kalmanfilterlikninger". Udervisningsnotat utgitt for UNIK4500 ved Unik, 2005.

[5] Oddvar Halligstad. "Diskretisering av Kontinuerlige Stokastiske System". Udervisningsnotat utgitt for UNIK4500 ved Unik, 2004.

[6] Oddvar Halligstad. "Stokastiske Systemer". Udervisningsnotat utgitt for UNIK4500 ved Unik, 2005.

[7] Scott T. Smith. MATLAB Advanced GUI Development, 2006. Dog Ear Publishing.

[8] Tom Lyche. Numeric Linear Algebra". Lecture Notes for Inf-Mat 3350/4350, 2006. WEB: http://heim.ifi.uio.no/ tom/book2006.pdf

[9] Oddvar Halligstad. "Prosjektoppgave; Stokastiske Systemer". Utgitt for UNIK4500 ved Unik, 2005.

Kapittel 8

Vedlegg A: Programkode

8.1 RUN_KF.m

```
function varargout = RUN_KF(varargin)
% RUN_KF M-file for RUN_KF.fig
%      RUN_KF, by itself, creates a new RUN_KF or raises the existing
%      singleton*.
%
%      H = RUN_KF returns the handle to a new RUN_KF or the handle to
%      the existing singleton*.
%
%      RUN_KF('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RUN_KF.M with the given input arguments.
%
%      RUN_KF('Property','Value',...) creates a new RUN_KF or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before RUN_KF_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to RUN_KF_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help RUN_KF

% Last Modified by GUIDE v2.5 12-Dec-2007 05:46:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 0;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RUN_KF_OpeningFcn, ...
                   'gui_OutputFcn',  @RUN_KF_OutputFcn, ...
```

```
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RUN_KF is made visible.
function RUN_KF_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RUN_KF (see VARARGIN)

    %Global Handle Sharing, GHS
    global hfig_main;          %definer global hadle (peker) sharing silk at vi
                               %kan lage globale variable.
    global hfig_read_params;
    hfig_main = handles; %legg all data for denne figuren i hfig_main

    format short e;

    %En struktur for å samle parametere relatert til
    %systemstatus.
    hfig_main.system_status = struct('isDiscrete'  , 'no', ...
                                     'isSimulated' , 'no', ...
                                     'startWith'   , 'prediction');

    %legger diverse systemparametere i en global handle(peker)
    %slik at de blir tilgjengelige i andre funksjoner og gui-vinduer.
    hfig_main.params.F = [];   %systemmatrise
    hfig_main.params.L = []';  %pådragsmatrise
    hfig_main.params.G = []';  %prosesstøymatrise
    hfig_main.params.H = [];   %målematrise

    hfig_main.params.u = [];   %pådrag
    hfig_main.params.v = [];   %prosesstøy
    hfig_main.params.w = [];   %målestøy

    hfig_main.params.Q_tilde = [];  %prosesstøyens spektraltetthet
    hfig_main.params.R       = [];  %målestøyens kovarians
    hfig_main.params.Q       = [];  %prosesstøyens kovarians
```

```
hfig_main.params.P0_hatt = [];  %initielt kovarians (a posteriori)
hfig_main.params.x0_hatt = [];  %initielt starttilstand (a posteriori)

hfig_main.params.P0_pred = [];  %initielt kovarians (a priori)
hfig_main.params.x0_pred = [];  %initielt starttilstand (a priori)

hfig_main.params.d   = [];  %tidsintervall
hfig_main.params.t0  = [];  %starttid
hfig_main.params.t_f = [];  %sluttid
hfig_main.params.T0 = [];   %tidsoppdaterings frekvens
hfig_main.params.M0 = [];   %måleoppdaterings frekvens

hfig_main.params.Fi = [];   %diskret systemmatrise
hfig_main.params.La = [];   %diskret pådragsmatrise
hfig_main.params.Ga = [];   %diskret prosesstøymatrise

hfig_main.params.N = [];    %antall diskrete steg
hfig_main.params.t = [];    % tk = d*k
                            %(sammenheng mellom kont. og disk. tid)

hfig_main.params.X      = []; %tilstandsvektor med hvit støy
hfig_main.params.X_det  = []; %determenistisk tilstandsvektor
hfig_main.params.X_pred = []; %a priori tilstandsvektor
hfig_main.params.X_est  = []; %a posteriori tilstandsvektor

hfig_main.params.P_pred  = []; %a priori kovarians
hfig_main.params.S_pred  = []; %a priori standardavvik


hfig_main.params.P_est  = [];  %a posteriori kovarians
hfig_main.params.S_est  = [];  %a posteriori standardavvik
hfig_main.params.K  = [];      %Kalman filter forsterkning

%velger en initiel startbetingelse ved KF-simulering
%kan endres bra brukeren i simuleringsvindu.
uipanel_x0_SelectionChangeFcn(hObject, [], hfig_main);

%plaserer systemlikningene som tekst skrevet i Latex
place_colorAndText();

%hent jgp-bilder til current-workspace
plotview   = imread('plotview.jpg', 'jpg');
paramsview = imread('paramsview.jpg', 'jpg');
system_input = imread('glossy_button.jpg', 'jpg');
menu_diskret = imread('menu_diskret.jpg', 'jpg');
menu_simulering = imread('menu_simulering.jpg', 'jpg');
menu_plot = imread('menu_plot.jpg', 'jpg');
menu_view_params = imread('menu_view_params.jpg', 'jpg');
```

```
    %plaser jgp-bilder på diverse knapper
    set(hfig_main.pushbutton_plotview, 'cdata', plotview);
    set(hfig_main.pushbutton_paramsview, 'cdata', paramsview);
    set(hfig_main.pushbutton_system_input, 'cdata', system_input);
    set(hfig_main.pushbutton_menu_diskret, 'cdata', menu_diskret);
    set(hfig_main.pushbutton_menu_simulering, 'cdata', menu_simulering);
    set(hfig_main.pushbutton_menu_plot, 'cdata', menu_plot);
    set(hfig_main.pushbutton_view_params, 'cdata', menu_view_params);

    %plaser bilder 'menu_axes.jpg' i aksen 'axes_logo'
    set(hfig_main.figure, 'currentaxes', hfig_main.axes_logo);
    imshow('menu_axes.jpg');

% Choose default command line output for RUN_KF
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = RUN_KF_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

%*****************************************************
%Bruk funksjonen read_params('X') for å åpne vinduet  *
%for innlesing av parameter                           *
%*****************************************************
% --- Executes on button press in pushbutton_F_Callback.
function pushbutton_F_Callback(hObject, eventdata, handles)
    read_params('F');

% --- Executes on button press in pushbutton_L_Callback.
function pushbutton_L_Callback(hObject, eventdata, handles)
    read_params('L');

% --- Executes on button press in pushbutton_G_Callback.
function pushbutton_G_Callback(hObject, eventdata, handles)
    read_params('G');

% --- Executes on button press in pushbutton_u_Callback.
function pushbutton_u_Callback(hObject, eventdata, handles)
    read_params('u');

% --- Executes on button press in pushbutton_v_Callback.
function pushbutton_v_Callback(hObject, eventdata, handles)
    read_params('v');

% --- Executes on button press in pushbutton_Q_tilde_Callback.
```

```
function pushbutton_Q_tilde_Callback(hObject, eventdata, handles)
    read_params('Q_tilde');

% --- Executes on button press in pushbutton_H_Callback.
function pushbutton_H_Callback(hObject, eventdata, handles)
    read_params('H');

% --- Executes on button press in pushbutton_w_Callback.
function pushbutton_w_Callback(hObject, eventdata, handles)
    read_params('w');

% --- Executes on button press in pushbutton_R_Callback.
function pushbutton_R_Callback(hObject, eventdata, handles)
    read_params('R');

% --- Executes on button press in pushbutton_P0_hatt_Callback.
function pushbutton_P0_hatt_Callback(hObject, eventdata, handles)
    read_params('P0_hatt');

% --- Executes on button press in pushbutton_x0_hatt_Callback.
function pushbutton_x0_hatt_Callback(hObject, eventdata, handles)
    read_params('x0_hatt');

% --- Executes on button press in pushbutton_x0_pred.
function pushbutton_x0_pred_Callback(hObject, eventdata, handles)
    read_params('x0_pred');

% --- Executes on button press in pushbutton_P0_pred.
function pushbutton_P0_pred_Callback(hObject, eventdata, handles)
    read_params('P0_pred');

% --- Executes on button press in pushbutton_discrete.
function pushbutton_discrete_Callback(hObject, eventdata, handles)
    discrete(); %diskretiser systemet dersom mulig
    % Update handles structure
    guidata(hObject, handles);

% --- Executes on button press in pushbutton_simulate.
function pushbutton_simulate_Callback(hObject, eventdata, handles)
    simulate();    %simuler hvis mulig
    % Update handles structure
    guidata(hObject, handles);


 %hvis vi ønsker å forandre på tidsintervall for diskretisering;
 function edit_d_Callback(hObject, eventdata, handles)
    global hfig_main;    %ønsker å bruke globale variable

    %les d (ny verdi) fra editbox (edit_d) og legg inn i value
    value = get(hfig_main.edit_d,'String');
```

```
    update_parameter('d',{value},'char');   %oppdater den tilhørende glo-
                                            %bale variabelen d med ny verdi
    % Update handles structure
    guidata(hObject, handles);

%ved endring av starttidspunkt t0
% + samme som forrige
function edit_t0_Callback(hObject, eventdata, handles)
    global hfig_main;

    value = get(hfig_main.edit_t0,'String');
    update_parameter('t0',{value},'char');

    % Update handles structure
    guidata(hObject, handles);

%ved endring av sluttidspunkt t_f
% + samme som forrige
function edit_tf_Callback(hObject, eventdata, handles)
    global hfig_main;

    value = get(hfig_main.edit_tf,'String');
    update_parameter('t_f',{value},'char');

    % Update handles structure
    guidata(hObject, handles);

%ved endring av tidsoppdateringsfrekvenss TO
% + samme som forrige
function edit_TO_Callback(hObject, eventdata, handles)
    global hfig_main;

    value = get(hfig_main.edit_TO,'String');
    update_parameter('TO',{value},'char');

    % Update handles structure
    guidata(hObject, handles);

%ved endring av måleoppdateringsfrekvenss TO
% + samme som forrige
function edit_MO_Callback(hObject, eventdata, handles)
    global hfig_main;

    value = get(hfig_main.edit_MO,'String');
    update_parameter('MO',{value},'char');

    % Update handles structure
    guidata(hObject, handles);

% --- Executes when selected object is changed in uipanel_x0.
```

```
%Funksjonen avgjør hvilken initialbetingelse vi ønsker å starte med.
function uipanel_x0_SelectionChangeFcn(hObject, eventdata, handles)
    global hfig_main;

    %test på tag for det grafiske objektet i uipanelet som ble aktivert ved
    %en muse klikk.
    switch get(get(hfig_main.uipanel_x0,'SelectedObject'),'Tag')
        %dersom radiobutton med tag lik radiobutton_pred ble presset, så
        %starter vi med estimering først. Oppdater status felt,og
        %tilhørende knapper med ønskelig handling
        case 'radiobutton_pred'
            hfig_main.system_status.startWith = 'estimation';
            set(hfig_main.pushbutton_x0_pred, 'Enable', 'on');
            set(hfig_main.pushbutton_P0_pred, 'Enable', 'on');
            set(hfig_main.pushbutton_x0_hatt , 'Enable', 'off');
            set(hfig_main.pushbutton_P0_hatt , 'Enable', 'off');
            set(hfig_main.pushbutton_init_x , 'String', 'x0_p');
            set(hfig_main.pushbutton_init_P , 'String', 'P0_p');

        case 'radiobutton_est'
            hfig_main.system_status.startWith = 'prediction';
            set(hfig_main.pushbutton_x0_pred, 'Enable', 'off');
            set(hfig_main.pushbutton_P0_pred, 'Enable', 'off');
            set(hfig_main.pushbutton_x0_hatt , 'Enable', 'on');
            set(hfig_main.pushbutton_P0_hatt , 'Enable', 'on');
            set(hfig_main.pushbutton_init_x , 'String', 'x0_h');
            set(hfig_main.pushbutton_init_P , 'String', 'P0_h');
    end

%Disse funksjonene vil kontrollere hvilken paneler som er synlige avhengig
%av hvilken meny-knapp som presses.
function pushbutton_system_input_Callback(hObject, eventdata, handles)
global hfig_main;
    set(hfig_main.uipanel_std_model, 'visible','on')
    set(hfig_main.uipanel_parameters, 'visible','on')
    set(hfig_main.uipanel_view_options, 'visible','off')
    set(hfig_main.uipanel_simulate, 'visible','off')
    set(hfig_main.uipanel_actions, 'visible','off')
    set(hfig_main.edit_parameter, 'visible','off')
    set(hfig_main.uipanel_view_param, 'visible','off')
    set(hfig_main.uipanel_plot, 'visible','off')

% --- Executes on button press in pushbutton_menu_simulering.
function pushbutton_menu_simulering_Callback(hObject, eventdata, handles)
global hfig_main;

    set(hfig_main.uipanel_simulate, 'visible','on');
    set(hfig_main.uipanel_actions, 'visible','off');
    set(hfig_main.uipanel_std_model, 'visible','off');
    set(hfig_main.uipanel_parameters, 'visible','off');
```

```
    set(hfig_main.uipanel_view_options, 'visible','off');
    set(hfig_main.edit_parameter, 'visible','off');
    set(hfig_main.uipanel_view_param, 'visible','off')
    set(hfig_main.uipanel_plot, 'visible','off')

% --- Executes on button press in pushbutton_menu_diskret.
function pushbutton_menu_diskret_Callback(hObject, eventdata, handles)
global hfig_main;

    set(hfig_main.uipanel_actions, 'visible','on');
    set(hfig_main.uipanel_simulate, 'visible','off');
    set(hfig_main.uipanel_std_model, 'visible','off');
    set(hfig_main.uipanel_parameters, 'visible','off');
    set(hfig_main.uipanel_view_options, 'visible','off');
    set(hfig_main.edit_parameter, 'visible','off');
    set(hfig_main.uipanel_view_param, 'visible','off')
    set(hfig_main.uipanel_plot, 'visible','off')

% --- Executes on button press in pushbutton_menu_plot.
function pushbutton_menu_plot_Callback(hObject, eventdata, handles)
global hfig_main;
    set(hfig_main.edit_parameter, 'visible','off');
    set(hfig_main.uipanel_actions, 'visible','off');
    set(hfig_main.uipanel_simulate, 'visible','off');
    set(hfig_main.uipanel_std_model, 'visible','off');
    set(hfig_main.uipanel_parameters, 'visible','off');
    set(hfig_main.uipanel_view_options, 'visible','off');
    set(hfig_main.uipanel_view_param, 'visible','off')
    set(hfig_main.uipanel_plot, 'visible','on')

% --- Executes on button press in pushbutton_view_params.
function pushbutton_view_params_Callback(hObject, eventdata, handles)
global hfig_main;
    set(hfig_main.uipanel_view_param, 'visible','on')
    set(hfig_main.edit_parameter, 'visible','on');
    set(hfig_main.uipanel_actions, 'visible','off');
    set(hfig_main.uipanel_simulate, 'visible','off');
    set(hfig_main.uipanel_std_model, 'visible','off');
    set(hfig_main.uipanel_parameters, 'visible','off');
    set(hfig_main.uipanel_view_options, 'visible','off');
    set(hfig_main.uipanel_plot, 'visible','off')



%Avhengig av hvilken knapp som presses vil funksjonen "visParameter('X')"
%kalles. Parameteren gis utfra hvilken knapp som kaller på funksjonen.
%Vi ønsker her å skrive ut forskjellige variabel verdier til en editbox, så
%brukeren kan se innholdet.
% --- Executes on button press in pushbutton_show_F.
function pushbutton_show_F_Callback(hObject, eventdata, handles)
```

```
    visParameter('F');
% --- Executes on button press in pushbutton_show_L.
function pushbutton_show_L_Callback(hObject, eventdata, handles)
visParameter('L');

% --- Executes on button press in pushbutton_show_u.
function pushbutton_show_u_Callback(hObject, eventdata, handles)
visParameter('u');

% --- Executes on button press in pushbutton_show_Q_tilde.
function pushbutton_show_Q_tilde_Callback(hObject, eventdata, handles)
visParameter('Q_tilde');

% --- Executes on button press in pushbutton_show_H.
function pushbutton_show_H_Callback(hObject, eventdata, handles)
visParameter('H');

% --- Executes on button press in pushbutton_show_wk.
function pushbutton_show_wk_Callback(hObject, eventdata, handles)
visParameter('w');

% --- Executes on button press in pushbutton_show_R.
function pushbutton_show_R_Callback(hObject, eventdata, handles)
visParameter('R');

% --- Executes on button press in pushbutton_show_Fi.
function pushbutton_show_Fi_Callback(hObject, eventdata, handles)
visParameter('Fi');

% --- Executes on button press in pushbutton_show_La.
function pushbutton_show_La_Callback(hObject, eventdata, handles)
visParameter('La');

% --- Executes on button press in pushbutton_show_Ga.
function pushbutton_show_Ga_Callback(hObject, eventdata, handles)
visParameter('Ga');

% --- Executes on button press in pushbutton_show_X.
function pushbutton_show_X_Callback(hObject, eventdata, handles)
visParameter('X');

% --- Executes on button press in pushbutton_show_x0_hatt.
function pushbutton_show_x0_hatt_Callback(hObject, eventdata, handles)
visParameter('x0_hatt');

% --- Executes on button press in pushbutton_show_P0_hatt.
function pushbutton_show_P0_hatt_Callback(hObject, eventdata, handles)
visParameter('P0_hatt');

% --- Executes on button press in pushbutton_show_Qk.
```

```
function pushbutton_show_Qk_Callback(hObject, eventdata, handles)
visParameter('Q');

% --- Executes on button press in pushbutton_show_X_est.
function pushbutton_show_X_est_Callback(hObject, eventdata, handles)
visParameter('X_est');

% --- Executes on button press in pushbutton_show_K.
function pushbutton_show_K_Callback(hObject, eventdata, handles)
visParameter('K');

% --- Executes on button press in pushbutton_show_G.
function pushbutton_show_G_Callback(hObject, eventdata, handles)
visParameter('G');

% --- Executes on button press in pushbutton_show_x0_pred.
function pushbutton_show_x0_pred_Callback(hObject, eventdata, handles)
visParameter('x0_pred');

% --- Executes on button press in pushbutton_show_X_pred.
function pushbutton_show_X_pred_Callback(hObject, eventdata, handles)
visParameter('X_pred');

% --- Executes on button press in pushbutton_show_P_pred.
function pushbutton_show_P_pred_Callback(hObject, eventdata, handles)
visParameter('P_pred');

% --- Executes on button press in pushbutton_show_P_est.
function pushbutton_show_P_est_Callback(hObject, eventdata, handles)
%visParameter('P_est');  %funksjonen må utvides til å gjelde for
                         %flere dimensjoner enn 2
% --- Executes on button press in pushbutton_show_P0_pred.
function pushbutton_show_P0_pred_Callback(hObject, eventdata, handles)
%visParameter('P_pred');  %funksjonen må utvides til å gjelde for
                         %flere dimensjoner enn 2


% --- Executes on button press in pushbutton_view_plot.
%Funksjonen leser inn alle plotte variabler så de blir tilgjengelige for
%funksjonen. Så henter den inn plott-kommandoen gitt av brukeren i
%editboksen (edit_plot) og evaluerer denne og plotter hvis syntaksen som
%ble gitt er riktig.
function pushbutton_view_plot_Callback(hObject, eventdata, handles)
global hfig_main;
X      = hfig_main.params.X;
X_est  = hfig_main.params.X_est;
X_pred = hfig_main.params.X_pred;
X_det  = hfig_main.params.X_det;

P_pred = hfig_main.params.P_pred;
```

```
P_est  = hfig_main.params.P_est;

S_pred = hfig_main.params.S_pred;
S_est  = hfig_main.params.S_est;
t  = hfig_main.params.t;
u  = hfig_main.params.u;

command = get(hfig_main.edit_plot, 'String'); %hent strengen fra editboks
figure %lag et et figurvindu for plotting
eval(command)   %evaluer plotte kommando gitt av bruker.

%Funksjonen lukker alle programvinduer når brukeren trykker menu -> exit
function menu_file_exit_Callback(hObject, eventdata, handles)
global hfig_main;
global hfig_read_params;
    %tester om programvindu har en aktiv handle, det vil den ha om vinduet
    %eksisterer, i så fall slett den med "delete()". Gjør dette både for
    %hovedvindu "hfig_main.figure" og parameter-innlesingsvindu
    %"hfig_read_params.figure".
    if ishandle(hfig_main.figure)
        if ishandle(hfig_read_params.figure)
            delete(hfig_read_params.figure);
        end
        delete(hfig_main.figure);
    end

%**********************************************************************
%Disse "pushbutton/editbox/menu" har ingen funksjon i denne versjonen*
%de er generert automatisk av GUIDE og er nødvendig å ha med         *
%**********************************************************************
% --- Executes on button press in pushbutton_uk.
function pushbutton_uk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_wk.
function pushbutton_wk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_Qk.
function pushbutton_Qk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_zk_Callback.
function pushbutton_zk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_Fi.
function pushbutton_Fi_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_La.
function pushbutton_La_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_Ga.
function pushbutton_Ga_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_Fi.
function pushbutton_filter_Fi_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_La.
function pushbutton_filter_La_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_Ga.
function pushbutton_filter_Ga_Callback(hObject, eventdata, handles)
```

```
% --- Executes on button press in pushbutton_filter_Qk.
function pushbutton_filter_Qk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_F.
function pushbutton_filter_F_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton65.
function pushbutton65_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_G.
function pushbutton_filter_G_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_Q.
function pushbutton_filter_Q_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_H.
function pushbutton_filter_H_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_Rk.
function pushbutton_filter_Rk_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_x0_hatt.
function pushbutton_filter_x0_hatt_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_P0_hatt.
function pushbutton_filter_P0_hatt_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_filter_L.
function pushbutton_filter_L_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_show_Z.
function pushbutton_show_Z_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_system_input.
function pushbutton_ekstra_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit_parameter_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton83.
function pushbutton83_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton103.
function pushbutton103_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit_parameter_CreateFcn(hObject, eventdata, handles)
% --- Executes on selection change in popupmenu_t0.
function popupmenu_t0_Callback(hObject, eventdata, handles)
% --- Executes on selection change in popupmenu_tf.
function popupmenu_tf_Callback(hObject, eventdata, handles)
%Ghør ingen ting i denne versjon
% --- Executes on selection change in popupmenu_T0.
function popupmenu_T0_Callback(hObject, eventdata, handles)
% --- Executes on selection change in popupmenu_M0.
function popupmenu_M0_Callback(hObject, eventdata, handles)
% --------------------------------------------------------------------
function menu_file_Callback(hObject, eventdata, handles)
function menu_help_Callback(hObject, eventdata, handles)
function menu_help_about_Callback(hObject, eventdata, handles)
function menu_file_savework_Callback(hObject, eventdata, handles)
function menu_file_loadwork_Callback(hObject, eventdata, handles)
% --------------------------------------------------------------------
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function edit_plot_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit_plot_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function edit_d_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function edit_tf_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function popupmenu_t0_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function popupmenu_tf_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function edit_T0_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function popupmenu_TO_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function edit_MO_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function popupmenu_MO_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in radiobutton_est.
function radiobutton_est_Callback(hObject, eventdata, handles)
%gjør ingen ting i denne versjon
% --- Executes during object creation, after setting all properties.
function edit_t0_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --------------------------------------------------------------------
```

8.2 place_colorANDText.m

```
%*************************************************************************
%Funksjonen har som oppgave å gjøre synelig de delene av grensesnittet som*
%vi ønsker å vise ved programstart.                                       *
%*************************************************************************
function place_colorAndText()
    global hfig_main;

    %gjør kun det grafiske objektet "uipanel_std_model" og
    %"uipanel_parameters" synlig. Disse tilhører meny-knappen "System
    %Inndata".
    set(hfig_main.uipanel_std_model, 'visible','on');
```

```
set(hfig_main.uipanel_parameters, 'visible','on');
set(hfig_main.uipanel_view_options, 'visible','off');
set(hfig_main.uipanel_simulate, 'visible','off');
set(hfig_main.uipanel_actions, 'visible','off');
set(hfig_main.edit_parameter, 'visible','off');
set(hfig_main.uipanel_view_param, 'visible','off');
set(hfig_main.uipanel_plot, 'visible','off')

%henter bakgrunnsfargen fra panelet "uipanel_initialVal" og bruker
%denen til å sette fargen på aksen "axes_x0_pred".
axes_color = get(hfig_main.uipanel_initialVal, 'backgroundcolor');
set(hfig_main.axes_x0_pred, 'color',  axes_color, ...
                            'xcolor', axes_color, ...
                            'ycolor', axes_color);

axes_color = get(hfig_main.uipanel_initialVal, 'backgroundcolor');
set(hfig_main.axes_x0_est, 'color',  axes_color, ...
                           'xcolor', axes_color, ...
                           'ycolor', axes_color);


%Lag text objekter for å vise systemmodell-
%og målelikning. Bruke Latex.
set(hfig_main.figure, 'currentaxes', hfig_main.axes_equations);
text( 'position', [70 110], ...
      'fontname', 'Arial', ...
      'fontsize', 11, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...  Latex som tekst kompilator.
      'string', '$${ \underline{\dot{x}}(t) = F(t)\underline{x}(t)
                + L(t)\underline{u}(t) + G(t)\underline{v}(t) }$$', ...
      'units', 'pixels');

 text( 'position', [70 80], ...
       'fontname', 'Arial', ...
       'fontsize', 11, ...
       'fontunits', 'pixels', ...
       'interpreter', 'latex', ...
       'string', '$${ \underline{{z}}_k(t) = H_k\underline{x}_k ...
                + \underline{w}_k }$$', ...
       'units', 'pixels');

 set(hfig_main.figure, 'currentaxes', hfig_main.axes_d);
 text( 'position', [5 15], ...
       'fontname', 'Arial', ...
       'fontsize', 12, ...
       'fontunits', 'pixels', ...
       'interpreter', 'latex', ...
       'string', '$${ \Delta{t} }$$', ...
       'units', 'pixels');
```

```
set(hfig_main.figure, 'currentaxes', hfig_main.axes_x0_pred);
text( 'position', [0 15], ...
      'fontname', 'Arial', ...
      'fontsize', 9, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...
      'string', '$${ \underline{x}_0 \sim ...
              N(\underline{\overline{x}}_0,\overline{P}_0) }$$', ...
      'units', 'pixels');

set(hfig_main.figure, 'currentaxes', hfig_main.axes_x0_est);
text( 'position', [0 15], ...
      'fontname', 'Arial', ...
      'fontsize', 9, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...
      'string', '$${ \underline{x}_0 \sim ...
              N(\underline{\hat{x}}_0,\hat{P}_0) }$$', ...
      'units', 'pixels');


set(hfig_main.figure, 'currentaxes', hfig_main.axes_plotview);
text( 'position', [5 15], ...
      'fontname', 'Arial', ...
      'fontsize', 7, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...
      'string', '{View plots}', ...
      'units', 'pixels');

set(hfig_main.figure, 'currentaxes', hfig_main.axes_paramsview);
text( 'position', [5 15], ...
      'fontname', 'Arial', ...
      'fontsize', 7, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...
      'string', '{View}', ...
      'units', 'pixels');

text( 'position', [5 4], ...
      'fontname', 'Arial', ...
      'fontsize', 7, ...
      'fontunits', 'pixels', ...
      'interpreter', 'latex', ...
      'string', '{Parameters}', ...
      'units', 'pixels');
```

8.3 read_params.m

```
%*************************************************************************
%Funksjonen lager et grafisk grensesnitt for innlesing av data for        *
%systemparametere. Verdier kan enten hentes inn fra fil, eller tastes     *
%direkte i en editbox.                                                    *
%*************************************************************************

function varargout = read_params(varargin)
% READ_PARAMS M-file for read_params.fig
%      READ_PARAMS, by itself, creates a new READ_PARAMS or raises the
%      existing singleton*.
%
%      H = READ_PARAMS returns the handle to a new READ_PARAMS or the
%      handle to the existing singleton*.
%
%      READ_PARAMS('CALLBACK',hObject,eventData,handles,...) calls the
%      local function named CALLBACK in READ_PARAMS.M with the given input
%      arguments.
%
%      READ_PARAMS('Property','Value',...) creates a new READ_PARAMS or
%      raises the existing singleton*.  Starting from the left, property
%      value pairs are applied to the GUI before
%      read_params_OpeningFunction gets called.  An unrecognized property
%      name or invalid value makes property application stop.  All inputs
%      are passed to read_params_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help read_params

% Last Modified by GUIDE v2.5 02-Nov-2007 20:52:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @read_params_OpeningFcn, ...
                   'gui_OutputFcn',  @read_params_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before read_params is made visible.
function read_params_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to read_params (see VARARGIN)
global hfig_main;
global hfig_read_params;
hfig_read_params = handles;

%sett input-variabelen "varargin" til selected_parameter.
hfig_read_params.selected_parameter = varargin;

%hent alle elementene i listen "listbox_parameters"
hfig_read_params.listbox_items = get(hfig_read_params.listbox_parameters,...
                                                              'string');

%bruke funksjonen "strmatch" for å finne på hvilken index i listen
%variabelen "selected_parameter" ligger på.Plasser den i "tmp".
tmp = strmatch(hfig_read_params.selected_parameter, ...
               hfig_read_params.listbox_items);

%set listboksen til å aktivere elementet på index/value "tmp".
set(hfig_read_params.listbox_parameters, 'value', tmp);

%Kall på Callback-funksjonen til listen for å oppdatere teksfeltet med den
%riktige symbolen (bokstaven som representerer f.eks systemmatrisen)
listbox_parameters_Callback(hObject, [], handles);

%set fargen til aksen - "axes_show_param" - til samme farge som bakgrunnen
%til uipanel - "uipanel_selected" - den ligger i
axes_color = get(hfig_read_params.uipanel_selected, 'backgroundcolor');
set(hfig_read_params.axes_show_param, 'color',  axes_color, ...
                                      'xcolor', axes_color, ...
                                      'ycolor', axes_color);


% Choose default command line output for read_params
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = read_params_OutputFcn(hObject, eventdata, handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in listbox_parameters.
%Callback-funksjonen til listen "listbox_parameters" vil kalles hver gang
%vi velger (trykker) et element i listen. Den skal lese av hvilken
%element vi valgte, og basert på dette valget oppdatere en akse med Latex
%tekst silk at vi får en forstørret representasjon av elementet.
function listbox_parameters_Callback(hObject, eventdata, handles)
    global hfig_main;
    global hfig_read_params;

    %hent plasseringen (indeksen) for det valgte elementet fra listen
    listbox_index = get(hfig_read_params.listbox_parameters, 'value');
    %hent ut selve strengen (F, G, L, u, osv.) som ligger på den
    %plasseringen fra listbox_items
    hfig_read_params.selected_parameter = ...
                            hfig_read_params.listbox_items{listbox_index};

    %set aksen "axes_show_param" til current for å vise valgt parameter i
    %Latex
    set(hfig_read_params.figure, 'CurrentAxes', ...
                                hfig_read_params.axes_show_param);

    %test hvilken streng som ligger i "selected_parameter" og basert på
    %dette oppdater tekst objektet i aksen med riktig tekst i Latex.
    switch hfig_read_params.selected_parameter
        case 'F'
            %fjern tidligere tekst fra aksen
            delete( get(hfig_read_params.axes_show_param, 'children') );
            text('Position', [0 15], ...
                 'FontName', 'Arial', ...
                 'FontSize', 30, ...
                 'FontUnits', 'pixels', ...
                 'Interpreter', 'Latex', ...
                 'String', '$${F}$$', ...
                 'Units', 'pixels');
        case 'L'
            delete( get(hfig_read_params.axes_show_param, 'children') );
            text('Position', [0 15], ...
                 'FontName', 'Arial', ...
                 'FontSize', 30, ...
                 'FontUnits', 'pixels', ...
                 'Interpreter', 'Latex', ...
```

```
            'String', '$${L}$$', ...
            'Units', 'pixels');
    case 'G'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${G}$$', ...
            'Units', 'pixels');
    case 'H'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${H}$$', ...
            'Units', 'pixels');
    case 'u'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\underline{u}}$$', ...
            'Units', 'pixels');
    case 'v'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\underline{v}}$$', ...
            'Units', 'pixels');
    case 'w'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\underline{w}}$$', ...
            'Units', 'pixels');
    case 'Q_tilde'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
```

```
            'FontName', 'Arial', ...
            'FontSize', 25, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\tilde{Q}}$$', ...
            'Units', 'pixels');
    case 'R'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${R}$$', ...
            'Units', 'pixels');
    case 'Q'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${Q}$$', ...
            'Units', 'pixels');
    case 'P0_hatt'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 25, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\hat{P}_0}$$', ...
            'Units', 'pixels');
    case 'x0_hatt'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\underline{\hat{x}}_0}$$', ...
            'Units', 'pixels');
    case 'P0_pred'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 25, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\overline{P}_0}$$', ...
```

```
            'Units', 'pixels');
    case 'x0_pred'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\underline{\overline{x}}_0}$$', ...
            'Units', 'pixels');
    case 'd'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 25, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${\Delta t}$$', ...
            'Units', 'pixels');
    case 't0'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${t_0}$$', ...
            'Units', 'pixels');
    case 't_f'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 30, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${t_f}$$', ...
            'Units', 'pixels');
     case 'T0'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
            'FontSize', 25, ...
            'FontUnits', 'pixels', ...
            'Interpreter', 'Latex', ...
            'String', '$${T0}$$', ...
            'Units', 'pixels');
    case 'M0'
        delete( get(hfig_read_params.axes_show_param, 'children') );
        text('Position', [0 15], ...
            'FontName', 'Arial', ...
```

```
                'FontSize', 20, ...
                'FontUnits', 'pixels', ...
                'Interpreter', 'Latex', ...
                'String', '$${MO}$$', ...
                'Units', 'pixels');
    end

    % Update handles structure
    guidata(hObject, handles);

% --- Executes on button press in pushbutton_load.
% Bruk funksjonen "uigetfile" for å lese inn fra fil. Funksjonene
% returnerer filnavnet og ålasseringen av den leste filen. Dette sendes som
% parameter videre til funksjonen "fload_parameter" som henter inn alle
% systemparameterene fra filen.
function pushbutton_load_Callback(hObject, eventdata, handles)
    [filename pathname] = uigetfile({'*.m'}, 'Open Parameter File *.m');
    file = [pathname filename];
    fload_parameter(file);

    % Update handles structure
    guidata(hObject, handles);

% --- Executes on button press in pushbutton_editbox.
%dersom vi ønsker å lese inn verdier fra editbox gjøres dette her.
function pushbutton_editbox_Callback(hObject, eventdata, handles)
    global hfig_main;
    global hfig_read_params;

    %les hvilken parameter som er valgt. Hent den gitte verdien fra
    %input-dialogboksen og legge i varaiabelen value. Send begge
    %"parameter" og "verdi" videre til "update_parameter" for å oppdatere
    %riktig systemparameter.
    parameter = hfig_read_params.selected_parameter;
    ex_string = 'Use Matlab notation. Ex: [1 2; 4 5]'; %info til bruker
    value     = inputdlg( {ex_string}, 'Matrix Input Box', [1 50]);
    update_parameter(parameter,value,'char');

    % Update handles structure
    guidata(hObject, handles);


% --- Executes on button press in pushbutton_done.
%Lukk dette vinduet for innlesing av parameterverdier når brukeren trykker
%på "Done"-knappejn.
function pushbutton_done_Callback(hObject, eventdata, handles)
    global hfig_read_params;

    %Test on vinduet er oppe, i så fall vil det eksistere en handle for
    %denne. Bruk funksjonen "ishandle()" forå gjøre dette. Hvis det
```

```
    %eksisterer så slett den, og vinduet vil lukke seg.
    if ishandle(hfig_read_params.figure)
        delete(hfig_read_params.figure);
    end


% --- Executes on button press in pushbutton_ok.
function pushbutton_ok_Callback(hObject, eventdata, handles)
% --- Executes on button press in pushbutton_table.
function pushbutton_table_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function listbox_parameters_CreateFcn(hObject, eventdata, handles)
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

8.4 discrete.m

```
%*****************************************************************************
%Funksjonen vil beregne systemmatrisene phi, lambda og gamma.            *
%*****************************************************************************

function descrete()
    global hfig_main;

    %test om alle parametere er lest inn
    %så vi kan diskretisere.
    if ready2discrete()
        %Skriv til skjerm at diskretiseringen pågår.
        set(hfig_main.text_discrete_status,'string','Working..')

        %Hent inn systemmatriser lokalt fra globale variable.
        %Vi kan da bruke kortere navn og få en mer oversiktelig kode
        F = hfig_main.params.F;
        L = hfig_main.params.L;
        G = hfig_main.params.G;
        d = hfig_main.params.d;
        Q_tilde = hfig_main.params.Q_tilde;

        %Beregn prosesstøymatrisen Gamma og systemmatrisen Phi
        [hfig_main.params.La, hfig_main.params.Fi] = kp2dpLa(F,L,d);
        hfig_main.params.Ga = kp2dpGa(F,G,Q_tilde,d);

       %oppdater systemstatus (struct) så vi vet at
```

```
       %systemet er diskretisert og kan teste på det senere
        hfig_main.system_status.isDiscrete = 'yes';

        %Aktiver knappen for simulering så dette kan utføres
        set(hfig_main.pushbutton_simulate,'Enable','on');

        %Skriv til skjerm at diskretiseringen er ferdig
        set(hfig_main.text_discrete_status,'string','Finished.')

    %dersom funksjonen "ready2discrete()" returnerer false (0)
    %send en dialogboks til skjerm med feilmelding.
    else
        errordlg('Some system parameters are not initialized', ...
                 'Discrete Error');
    end

    %funksjonen tester om parameterene som trengs
    %ikke er tomme. Returnerer 1 dersom alle har verdier, 0 ellers, i så
    %fall kan systemet ik. diskretiseres.
    function answer = ready2discrete()
    global hfig_main;

    %check if all needed parmeters have values
    if ~isempty(hfig_main.params.F)  ...
       &&   ...
       ~isempty(hfig_main.params.L) ...
       &&   ...
       ~isempty(hfig_main.params.G) ...
       &&   ...
       ~isempty(hfig_main.params.d) ...
       &&   ...
       ~isempty(hfig_main.params.Q_tilde)
       answer = 1;
    else
        answer = 0;
    end
```

8.5 simulate.m

```
%*************************************************************************
%Funksjonen skal simulere og beregne Kalman filter verdiene              *
%*************************************************************************

function simulate()
    global hfig_main;

if ready2sim() %test om vi kan starte å simulere

    %Skriv til skjerm (text object) at simulering pågår.
```

```
    set(hfig_main.text_status_simuler,'string','Calculating..');

    %Hent inn nødvendige systemmatriser lokalt fra globale variable.
    %Vi kan da bruke kortere navn og få en mer oversiktelig kode
    F  = hfig_main.params.F;
    Fi = hfig_main.params.Fi;
    La = hfig_main.params.La;
    Ga = hfig_main.params.Ga;
    Q = hfig_main.params.Q;
    H = hfig_main.params.H;
    R = hfig_main.params.R;
    u = hfig_main.params.u;
    %I denne versjon antas de å være vite og ik. nødvendig å lese inn.
%     v = hfig_main.params.v;
%     w = hfig_main.params.w;
    P0_hatt = hfig_main.params.P0_hatt;
    x0_hatt = hfig_main.params.x0_hatt;
    P0_pred = hfig_main.params.P0_pred;
    x0_pred = hfig_main.params.x0_pred;

    X      = hfig_main.params.X;
    X_det  = hfig_main.params.X_det;

    X_pred = hfig_main.params.X_pred;
    P_pred = hfig_main.params.P_pred;
    S_pred = hfig_main.params.S_pred;

    X_est  = hfig_main.params.X_est;
    P_est  = hfig_main.params.P_est;
    S_est  = hfig_main.params.S_est;
    K  = hfig_main.params.K;

    T0 = hfig_main.params.T0;
    M0 = hfig_main.params.M0;

    d   = hfig_main.params.d;
    t0  = hfig_main.params.t0;
    t_f = hfig_main.params.t_f;

    N = ((t_f - t0)/d) + 1;
    t = [0:(N-1)]*d;
    [m,n] = size(F);

    X      = zeros(n,N);   %Tilstandsvariable
    X_det  = X;            %determenistiske tilstandsvariable
    X_est  = zeros(n,N);   %a posteriori tilstandsvariable (estimert)
    X_pred = zeros(n,N);   %a priori tilstandsvariable (prediktert)
    P_pred = zeros(n,n,N); %a priori kovarians for prediksjonsavvik
    P_est  = zeros(n,n,N); %a posteriori kovarians for estimerigsavvik
    S_est  = zeros(n,N);   %standardavvik for P_est
```

```
S_pred = zeros(n,N);   %standardavvik for P_pred

switch hfig_main.system_status.startWith
    case 'prediction' %start with timeupdate (TO)
        if_measurement = 'true'; %because we start with x0_hat

        %1. Beregn x0
        x0 = x0_hatt + chol(P0_hatt)' * randn(n,1);

        %2. Simuler
        X(:,1) = x0;
        P_est(:,:,1) = P0_hatt;
        S_est(:,1) = sqrt(diag(P_est(:,:,1)));

        for k = 2:N
            X(:,k) = Fi*X(:,k-1) + La*u + Ga*randn(n,1);
            X_det(:,k) = Fi*X_det(:,k-1) + La*u;
        end

        for k = 2:N
        %Error analysis
        P_pred(:,:,k) = Fi*P_est(:,:,k-1)*Fi' + Ga*Q*Ga';
        K(:,k)        = P_pred(:,:,k)*H'*inv((H*P_pred(:,:,k)*H' + R));
        P_est(:,:,k)  = (eye(n) - K(:,k)*H)*P_pred(:,:,k);

        %Standard deviations
        S_est(:,k) = sqrt(diag(P_est(:,:,k)));
        S_pred(:,k) = sqrt(diag(P_est(:,:,k)));

        %Time update
        TOsteps = 1/(d*TO);         %TO-frequency koverted to TO_steps
        if rem(k-1, TOsteps ) == 0
            switch if_measurement
                case 'true'
                    X_pred(:,k)   = Fi*X_est(:,k-1) + La*u;
                case 'false'
                    X_pred(:,k)   = Fi*X_pred(:,k-1) + La*u;
            end
        else
            % dersom ingen TO, bruk uansett initiell x0_hatt til å
            % regne første prdikterte.
            if (k==2)
                X_pred(:,2) = Fi*X_est(:,1) + La*u;
            else
            X_pred(:,k) = Fi*X_pred(:,k-1) + La*u;
            end
        end

        %Measurement update
        MOsteps = 1/(d*MO);         %MO-frequency koverted to MO_steps
```

```
        if rem(k-1, MOsteps ) == 0
            if_measurement = 'true';
            z           = H*X(:,k) + R*randn(1,1); %R:målestøyens kov.
            X_est(:,k)  = X_pred(:,k) + K(:,k)*(z - H*X_pred(:,k));
        else
            %ingen måling; K=0 => X_est(:,k) = X_pred(:,k
            if_measurement = 'false';
            X_est(:,k)  = X_pred(:,k); )
        end

        end

    case 'estimation'
        if_measurement = 'true';

        %1. Calculate x0
        x0 = x0_pred + chol(P0_pred)' * randn(n,1);

        %2. Simulate
        X(:,1) = x0;
        for k = 2:N
            X(:,k) = Fi*X(:,k-1) + La*u + Ga*randn(n,1);
            X_det(:,k) = Fi*X_det(:,k-1) + La*u;
        end

        for k = 1:N
        %Error analysis
        K(:,k)       = P_pred(:,:,k)*H'*inv((H*P_pred(:,:,k)*H' + R));
        P_est(:,:,k) = (eye(n) - K(:,k)*H)*P_pred(:,:,k);
        if k < N
            P_pred(:,:,k+1) = Fi*P_est(:,:,k)*Fi' + Ga*Q*Ga';
        end

        %Standard deviations
        S_est(:,k) = sqrt(diag(P_est(:,:,k)));
        S_pred(:,k) = sqrt(diag(P_est(:,:,k)));

        %Measurement update
        MOsteps = 1/(d*MO);         %MO-frequency koverted to MO_steps
        if rem(k-1, MOsteps ) == 0
           if_measurement = 'true';
           z           = H*X(:,k) + chol(R)'*randn(1,1);
           X_est(:,k) = X_pred(:,k) + K(:,k)*(z - H*X_pred(:,k));
        else
           if_measurement = 'false';
           if k == 1 %vi antar alltid måling ved første steg
               if_measurement = 'true';
               z           = H*X(:,k) + R*randn(1,1);
               X_est(:,k) = X_pred(:,k) + K(:,k)*(z - H*X_pred(:,k));
           else
```

```
                X_est(:,k) = X_pred(:,k);
            end;
        end

        %Time update
        TOsteps = 1/(d*TO);         %TO-frequency koverted to TO_steps
        if rem(k-1, TOsteps ) == 0 && k < N
            switch if_measurement
                case 'true'
                    X_pred(:,k+1) = Fi*X_est(:,k) + La*u;
                case 'false'
                    X_pred(:,k+1) = Fi*X_pred(:,k) + La*u;
            end
        else if k < N %for å beregne kun opp til X_pred(:,N)
                X_pred(:,k+1) = Fi*X_pred(:,k) + La*u;
            end
        end
        end
end

%Put parameters back to global variables
hfig_main.params.F = F;

hfig_main.params.Fi = Fi;
hfig_main.params.La = La;
hfig_main.params.Ga = Ga;
hfig_main.params.Q = Q;
hfig_main.params.H = H;
hfig_main.params.R = R;

hfig_main.params.u = u;
hfig_main.params.X      = X;
hfig_main.params.X_det  = X_det;

hfig_main.params.X_pred = X_pred;
hfig_main.params.P_pred = P_pred;
hfig_main.params.S_pred = S_pred;

hfig_main.params.X_est  = X_est;
hfig_main.params.P_est  = P_est;
hfig_main.params.S_est  = S_est;
hfig_main.params.K  = K;

hfig_main.params.N = N;
hfig_main.params.t = t;

%update system status
hfig_main.system_status.isSimulated = 'yes';
set(hfig_main.text_status_simuler,'string','Finished.');
```

```
else
    set(hfig_main.text_status_simuler,'string','No result. Error');
    errordlg('Some system parameters are not initialized or the system ...
              is not in discrete form', ...
             'Simulation Error');
end


function answer = ready2sim()
    global hfig_main;

    set(hfig_main.text_status_simuler,'string','Checking..');

    %check if all needed parmeters have values
    if isequal(hfig_main.system_status.isDiscrete,'yes') ...
       &&   ...
       ~isempty(hfig_main.params.F)   ...
       &&   ...
       ~isempty(hfig_main.params.Fi)  ...
       &&   ...
       ~isempty(hfig_main.params.La)  ...
       &&   ...
       ~isempty(hfig_main.params.Ga)  ...
       &&   ...
       ~isempty(hfig_main.params.u)   ...
       &&   ...
       ~isempty(hfig_main.params.d)   ...
       &&   ...
       ~isempty(hfig_main.params.t0)  ...
       &&   ...
       ~isempty(hfig_main.params.t_f) ...
       &&   ...
       ~isempty(hfig_main.params.M0)  ...
       &&   ...
       ~isempty(hfig_main.params.T0)

       if isequal(hfig_main.system_status.startWith,'estimation') ...
          && ...
          ~isempty(hfig_main.params.P0_pred)  ...
          && ...
          ~isempty(hfig_main.params.x0_pred)
               answer = 1;
       else if isequal(hfig_main.system_status.startWith,'prediction') ...
          && ...
          ~isempty(hfig_main.params.P0_hatt)  ...
          && ...
          ~isempty(hfig_main.params.x0_hatt)
              answer = 1;
           else
              answer = 0;
```

```
        end
    end
else
    answer = 0;
end
```

8.6 update_parameter.m

```
%************************************************************************
%Funksjonen har som oppgave å oppdatere de globale variablene som bl.a.   *
%beskriver systemmatrisene (mm.).  Funksjonene tar inn tre parametere.    *
%************************************************************************

function update_parameter(parameter,value,value_type)
    global hfig_main;

    try %utfør testen i en try-catch blokk så feilmeldinger kan fanges.
        if isequal(value_type, 'char')   %hvis verdi komr. fra editbox
            if isnumeric(eval(value{:})) %godkjenn kun hvis numeric
                value = eval(value{:});  %evaluer og konverter til numerisk
            else
                %En feiloppdatering,gi beskjed til bruker
                %og avslutt her; returner fra funksjonen
                errordlg('No valid input was given. Try again', ...
                         'Input Value Error');
                return                   %ingen gyldig input,break function
            end

        else if isequal(value_type, 'numeric') %hvis verdi komr. fra fil
                if isnumeric(value)            %godkjenn kun hvis numeric
                    %bra, gjør ingen ting og behold verdien
                else
                    errordlg('No valid input was given. Try again', ...
                             'Input Value Error');
                    return
                end
            end
        end
    catch
        errordlg('No valid input was given. Try again', ...
                 'Input Value Error');
        return
    end

    %Dersom den andre parameteren (value) er gyldig, kan vi teste på
    %hvilken parameter som ble lest inn med en switch.
    switch parameter
        case 'F'
```

```
            %innlest verdi er for systemmatrisen F. Legg "value" inn i den
            %globale variabelen F.
            hfig_main.params.F = value;

            %Oppdater knapp som representerer F med riktig farge
            update_pushbuttonColor(hfig_main.params.F, ...
                                   hfig_main.pushbutton_F);
            set(hfig_main.pushbutton_F, 'TooltipString',mat2str(value));
        case 'L'
            hfig_main.params.L = value;
            update_pushbuttonColor(hfig_main.params.L, ...
                                   hfig_main.pushbutton_L);
        case 'G'
            hfig_main.params.G = value;
            update_pushbuttonColor(hfig_main.params.G, ...
                                   hfig_main.pushbutton_G);
        case 'H'
            hfig_main.params.H = value;
            update_pushbuttonColor(hfig_main.params.H, ...
                                   hfig_main.pushbutton_H);
        case 'u'
            hfig_main.params.u = value;
            update_pushbuttonColor(hfig_main.params.u, ...
                                   hfig_main.pushbutton_u);
        case 'v'
            hfig_main.params.v = value;
            update_pushbuttonColor(hfig_main.params.v, ...
                                   hfig_main.pushbutton_v);
        case 'w'
            hfig_main.params.w = value;
            update_pushbuttonColor(hfig_main.params.w, ...
                                   hfig_main.pushbutton_w);
        case 'Q_tilde'
            hfig_main.params.Q_tilde = value;
            update_pushbuttonColor(hfig_main.params.Q_tilde, ...
                                   hfig_main.pushbutton_Q_tilde);
        case 'R'
            hfig_main.params.R = value;
            update_pushbuttonColor(hfig_main.params.R, ...
                                   hfig_main.pushbutton_R);
        case 'Q'
            hfig_main.params.Q = value;
%             update_pushbuttonColor(hfig_main.params.Q, ...
%                                    hfig_main.pushbutton_Q);
        case 'P0_hatt'
            hfig_main.params.P0_hatt = value;
            update_pushbuttonColor(hfig_main.params.P0_hatt, ...
                                   hfig_main.pushbutton_P0_hatt);
        case 'x0_hatt'
            hfig_main.params.x0_hatt = value;
```

```
        update_pushbuttonColor(hfig_main.params.x0_hatt, ...
                               hfig_main.pushbutton_x0_hatt);
    case 'P0_pred'
        hfig_main.params.P0_pred = value;
        update_pushbuttonColor(hfig_main.params.P0_pred, ...
                               hfig_main.pushbutton_P0_pred);
    case 'x0_pred'
        hfig_main.params.x0_pred = value;
        update_pushbuttonColor(hfig_main.params.x0_pred, ...
                               hfig_main.pushbutton_x0_pred);
    case 'd'
        hfig_main.params.d = value;
        %oppdater to ting. Fargen på aksen hvor vi har skrevet d som en
        %tekst, og dens editbox hvor vi taster inn verdier med riktig
        %verdi.
        update_axesColor(value,hfig_main.axes_d);
        set(hfig_main.edit_d, 'String',num2str(value));

    case 't0'
        hfig_main.params.t0 = value;
        set(hfig_main.edit_t0,'string', num2str(value));
    case 't_f'
        hfig_main.params.t_f = value;
        set(hfig_main.edit_tf,'string', num2str(value));
    case 'T0'
        hfig_main.params.T0 = value;
        set(hfig_main.edit_T0,'string', num2str(value));
    case 'M0'
        hfig_main.params.M0 = value;
        set(hfig_main.edit_M0,'string', num2str(value));
end


%oppdater knapp "pushbutton" med riktig farge avhengig av om "value" er
%tom eller ikke.
function update_pushbuttonColor(value, pushbutton)
    if isempty(value)
        set(pushbutton, 'BackgroundColor', [1 0 0])
    else
        set(pushbutton, 'BackgroundColor', [0.83 0.81 0.78]);
    end

%oppdater aksen "this_axes" med riktig farge avhengig av om "value" er
%tom eller ikke.
function update_axesColor(value, this_axes)
    if isempty(value)
        set(this_axes, 'color', [1 0 0])
    else
        set(this_axes, 'color', [0.941 0.941 0.941])
    end
```

8.7 visParameter.m

```
%*************************************************************************
%Funksjonen tar inn en parameter. Avhenging av hvilken systemvariabel     *
%denne parameteren referer til, settes den til A, og skrives ut til en    *
%editboks på skjermen.                                                    *
%*************************************************************************

function visParameter(parameter)
global hfig_main;
switch parameter
    case 'F'
        A = hfig_main.params.F;
    case 'L'
        A = hfig_main.params.L;
    case 'G'
        A = hfig_main.params.G;
    case 'H'
        A = hfig_main.params.H;
    case 'u'
        A = hfig_main.params.u;
    case 'v'
        A = hfig_main.params.v;
    case 'w'
        A = hfig_main.params.w;
    case 'Q_tilde'
        A = hfig_main.params.Q_tilde;
    case 'R'
        A = hfig_main.params.R;
    case 'Q'
        A = hfig_main.params.Q;
    case 'P0_hatt'
        A = hfig_main.params.P0_hatt;
    case 'x0_hatt'
        A = hfig_main.params.x0_hatt;
    case 'P0_pred'
        A = hfig_main.params.P0_pred;
    case 'x0_pred'
        A = hfig_main.params.x0_pred;
    case 'd'
        A = hfig_main.params.d;
    case 't0'
        A = hfig_main.params.t0;
    case 't_f'
        A = hfig_main.params.t_f;
    case 'T0'
```

```
        A = hfig_main.params.TO;
    case 'MO'
        A = hfig_main.params.MO;
    case 'Fi'
        A = hfig_main.params.Fi;
    case 'La'
        A = hfig_main.params.La;
    case 'Ga'
        A = hfig_main.params.Ga;
    case 'X'
        A = hfig_main.params.X;
    case 'P_pred'
        A = hfig_main.params.P_pred;
    case 'X_pred'
        A = hfig_main.params.X_pred;
    case 'P_est'
        A = hfig_main.params.P_est;
    case 'X_est'
        A = hfig_main.params.X_est;
    case 'K'
        A = hfig_main.params.K;
end

[m n] = size(A);          %dimensjon på matrisen som skal skrives ut.
vindu     = 8;            %antall kolonner det er plass til i vår editbox.
restVindu = 0;            %antall kollener som blir igjen som ik fikk plass
                          %i de n antall multipler av editbox vindu.
row       = {};           %oppbevaring av rader med kolonnelengde lik vindu
nRounds = ceil(n/vindu); %antall set av kolonner i matrisen
siste = 0;                %med sammenlagt lengde lik vindu

if (n <= vindu)           %test om kolonnene som vi skal vise er midre en
    vindu = n;            %breden på på editbox. I så fall bruk n som vindu
else
     siste = rem(n,vindu);  %test hvor mange kolonner som havner utenfor
end                         %vindu

%iterer gjennom antall riktig {kolonner i vindu}-set og plasser dem riktig
%i editbox.
for k = 1:nRounds
    if(k > nRounds-1 && siste ~= 0)
        restVindu = vindu;
        vindu = siste;
    end
    for i = ( m*(k-1)+1 ) : m*k
        j = (k-1);
        row{i+k} = sprintf('%1.4e  ', A(i-j*m, j*vindu+1 : j*vindu+vindu));
        if(i == m*j+1 )
            if (k > nRounds-1 && siste ~= 0)
                kolFra = num2str(j*restVindu+1);
```

```
                kolTil = num2str(j*restVindu+vindu);
                row{ i+j } = char(['KOLONNE ' kolFra ' til ' kolTil ]);
            else
                row{ i+j } = char(['KOLONNNE ' num2str(j*vindu+1) ...
                                   ' til ' num2str(j*vindu+vindu)]);
            end
        end
    end
end

%legger alle radene med riktig kolonne lengde i editbox
set(hfig_main.edit_parameter, 'string',row);
```

8.8 fload_parameter.m

```
%*************************************************************************
%Les inn fra fil alle parametere og legg dem i current workspace. Bruk    *
%kommandoen "who" til å finne frem navnene på alle parametere som ble lest*
%inn i denne workspace. Dersom de har samme navn som de parameterene      *
%(variable) i vår egen applikasjon, bruk da de innleste verdiene til å    *
%initialisere diverse og tilhørende variable i vår program.               *
%*************************************************************************

function fload_parameter(file)
    global hfig_main;

    eval(['run ' file ';']);    %last inn parametere til lokal workspace.
    names = who;                %identidfiser de innleste parameternavn.

    for k = 1:size(names,1)     %gå gjennom alle navnene og se om de
        switch names{k}         %skal brukes til initialsering.
            case 'F'
                update_parameter('F',F,'numeric');
            case 'L'
                update_parameter('L',L,'numeric');
            case 'G'
                update_parameter('G',G,'numeric');
            case 'H'
                update_parameter('H',H,'numeric');
            case 'u'
                update_parameter('u',u,'numeric');
            case 'v'
                update_parameter('v',v,'numeric');
            case 'w'
                update_parameter('w',w,'numeric');
            case 'Q_tilde'
                update_parameter('Q_tilde',Q_tilde,'numeric');
            case 'R'
                update_parameter('R',R,'numeric');
```

```
            case 'Q'
                update_parameter('Q',Q,'numeric');
            case 'P0_hatt'
                update_parameter('P0_hatt',P0_hatt,'numeric');
            case 'x0_hatt'
                update_parameter('x0_hatt',x0_hatt,'numeric');
            case 'P0_pred'
                update_parameter('P0_pred',P0_pred,'numeric');
            case 'x0_pred'
                update_parameter('x0_pred',x0_pred,'numeric');
            case 'd'
                update_parameter('d',d,'numeric');
            case 't0'
                update_parameter('t0',t0,'numeric');
            case 't_f'
                update_parameter('t_f',t_f,'numeric');
            case 'T0'
                update_parameter('T0',T0,'numeric');
            case 'M0'
                update_parameter('M0',M0,'numeric');
        end
    end
```

8.9 kp2dpGa.m

```
%%*************************************************************************
%Beregner den diskrete Gamma numerisk                                     *
%*************************************************************************

function Ga  = kp2dpGa(F, G, Q_tilde, d)
    [m,n] = size(F);
    A = [F G*Q_tilde*G'; zeros(m,n) -F']*d;
    B = expm(A);
    S = B(1:n,n+1:2*n) * inv(B(n+1:2*n,n+1:2*n));
    Ga = chol(S,'lower');
```

8.10 kp2dpLa.m

```
%*************************************************************************
%Beregning av systemmatrisene Lambda og Phi                               *
%*************************************************************************

function [La,Fi] = kp2dpLa(F, L, d)
nt = 18;
n = size(F,1);
%Nedskalering av tiden
```

```
m = ceil(log2(d*norm(F))); if (m<0) m=0; end; dd = d/2^m;

%Beregn Fi(dd) og La(dd) med den nye tiden dd.
W = diag(ones(n,1)); s=dd;
for k=2:nt
    i = nt + 2 - k;
    s = dd/i;
    Fi = F*W*s;
    W = Fi + diag(ones(n,1));
end
Fi = W * F * dd + diag(ones(n,1));
La = W*L*dd;

%Tilbakeskaler tiden fra dd -> d
%og beregn Fi(d) og La(d)
A = diag(ones(n,1));
if(m > 0)
    for i = 1:m
        W = Fi*A;
        A = W+A;
        W = Fi * Fi;
        Fi = W;
    end
end
La = A*La;
```

8.11 system_init.m

```
%initial system values
T2 = 5;
T3 = 1;


P0_hatt = diag([1 0.1^2 0.1^2]);
x0_hatt = 0;

Q_tilde = 2*0.1^2;
R = 1;
t0 = 0;
d = 0.01;
u = 1;
t_f = 100;

%System
F = [0 1 0 ; 0 -1/T2 1/T2 ; 0 0 -1/T3];
L = [0 0 1/T3]';
G = [0 0 1]';
H = [1 0 0];
Q = eye(size(F,1));
```

```
TO = 100;
MO = 1;
```

Kapittel 9

Vedlegg B

9.1 Navnsetting av systemparametere

```
F :   %systemmatrise
L :   %pådragsmatrise
G :   %prosesstøymatrise
H :   %målematrise

u :   %pådrag
v :   %prosesstøy
w :   %målestøy

Q_tilde :  %prosesstøyens spektraltetthet
R       :  %målestøyens kovarians
Q       :  %prosesstøyens kovarians

P0_hatt :  %initielt kovarians (a posteriori)
x0_hatt :  %initielt starttilstand (a posteriori)

P0_pred :  %initielt kovarians (a priori)
x0_pred :  %initielt starttilstand (a priori)

d   :  %tidsintervall
t0  :  %starttid
t_f :  %sluttid
T0  :  %tidsoppdaterings frekvens
M0  :  %måleoppdaterings frekvens

Fi  :  %diskret systemmatrise
La  :  %diskret pådragsmatrise
Ga  :  %diskret prosesstøymatrise

N   :  %antall diskrete steg
t   :  % tk = d*k
       %(sammenheng mellom kont. og disk. tid)
```

```
X      : %tilstandsvektor med hvit støy
X_det  : %determenistisk tilstandsvektor
X_pred : %a priori tilstandsvektor
X_est  : %a posteriori tilstandsvektor

P_pred  : %a priori kovarians
S_pred  : %a priori standardavvik

P_est  :  %a posteriori kovarians
S_est  :  %a posteriori standardavvik
K      :  %Kalman filter forsterkning
```

Kapittel 10

Vedlegg C

10.1 CD

CD'en inneholder følgende:

1. Mappe kalt **Kalman_Application**, med alle filer som er nødvendig for å kjøre programmet.
2. Mappe kalt **Rapport**, som inneholder selve rapporten.
3. Mappe kalt **Prosjektbeskrivelse**, som inneholder prosjektbeskrivelsen.