

AARHUS UNIVERSITET

ADAPTIVE CONTROL AND AUTOMATION

7. SEMESTER

ACA projekt

Gruppemedlemmer:

Daniel Tøttrup

Stinus Lykke Skovgaard

AUID

au544366

au520659



14. december 2018

Indhold

1	Indledning	3
2	Identifikation af open-loop system	4
3	Pol-placering	7
3.1	Steady state error	8
4	Observer	10

Figurer

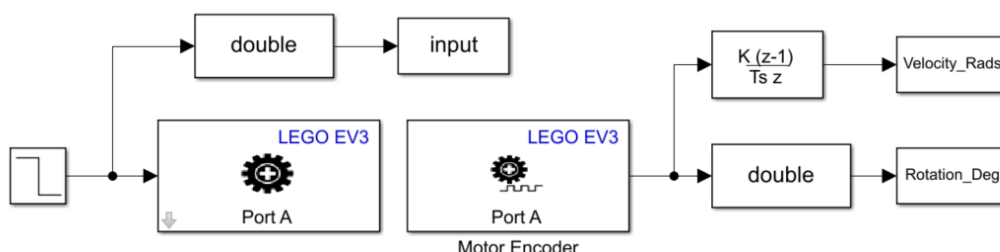
1	Simulink opstilling af måling	4
2	Måling af motor vinkel i grader	4
3	Måling af motorens hastighed i grader	5
4	State space block diagram uden D	6
5	Blok diagram over vores controller	7
6	Steprespons for closed loop system	8
7	Matrix når tredje pol indsættes	8
8	Matrix når tredje pol indsættes	9
9	State space block diagram uden D	10
10	Block diagram af systemet med observer og controller	11
11	Block diagram af systemet med observer og controller	12
12	Block diagram af systemet med observer og controller	12
13	Block diagram af systemet med observer og controller	13
14	Block diagram af systemet med observer og controller	14
15	Block diagram af systemet med observer og controller	14

1 Indledning

Vores første ide til projektet var at lave en ”adaptiv følgebil”. Tanken var at vores Lego bil kunne holde en bestemt afstand til et objekt i bevægelse, hvor svingende hastighed, skiftende hældning på underlaget, samt ændring af vægt monteret på vores Lego bil, ikke ville påvirke dens adfærd. Efter vi havde lavet test med udstyret til rådighed, besluttede vi os for at gå i en anden retning. Vi fik nemlig mere præcise målinger fra encoderen monteret inde i motoren, og ønskede derfor at gå videre den som vores sensor. Det betød dog også, at vores Lego bil ikke længere ville være i stand til at følge et objekt i bevægelse, da den ikke længere havde ”øjne” i form af en afstandssensor. Ideen vi gik videre med, blev derfor en Lego bil der vil være i stand til at køre en specifik afstand, upåvirket af f.eks. vægt monteret på Lego bilen eller underlagets hældning. Derfor endte vores system i sidste ende med at bestå af en motor, med en indlagt encoder som sensor.

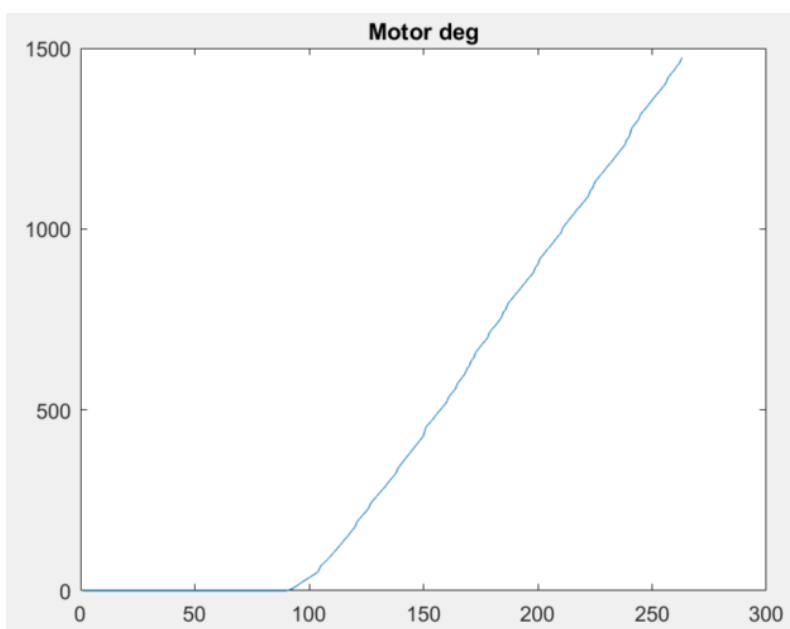
2 Identifikation af open-loop system

Efter vi har besluttet os for hvilket system vi ønskede at designe, og hvilke dele systemet skal bestå af, er det næste skridt at indsamle data fra vores motor via den indlagte encoder. Det gør vi ved at sende et step input ind på motoren. Vi aflæser to outputs, både det direkte output fra encodern, som er rotationen i grader, samt den afledte som er hastigheden i grader. Opstillingen af denne måling i Simulink kan ses på [Figure 1](#). Disse målinger gentog vi mange gange, for at få de bedst mulige målinger. Grunden til at vi ønsker de bedste mulige målinger er, så vi kan lave den bedst mulige repræsentation af vores system, så det system vi beregner på, er så tæt på det faktiske system som overhovedet muligt.

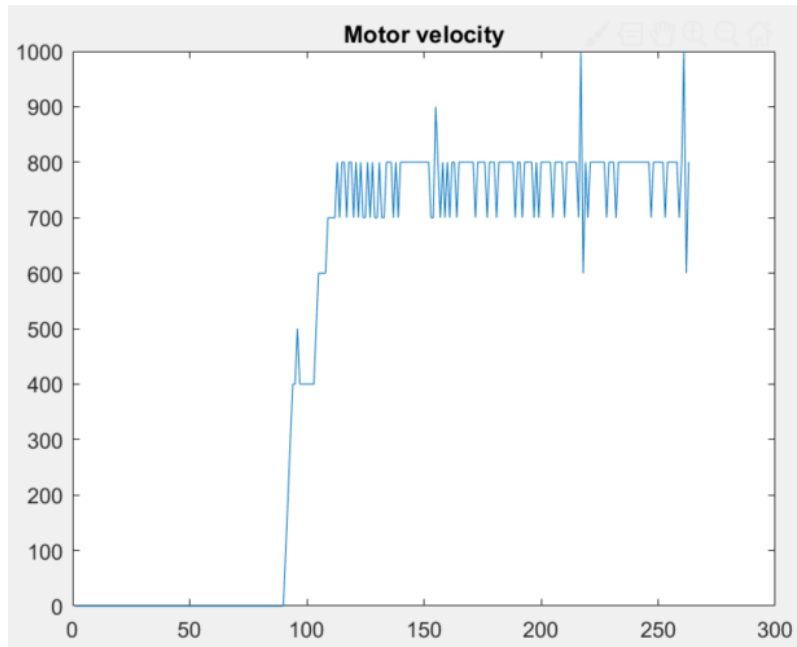


Figur 1: Simulink opstilling af måling

Grafen på [Figure 2](#) repræsenterer motorens rotation i grader. Grafen på [Figure 3](#) repræsenterer den afledte af positionen i grader, altså hastigheden i grader. Som man kan se på [Figure 3](#) opfører den afledte sig som et første ordens system, dog med et lille udfald i accelerationen, men da den er så lille vælger vi at se bort fra denne. Da vores afledte opfører sig som et første ordens system må det betyde at vores system er et 2. ordens system.



Figur 2: Måling af motor vinkel i grader



Figur 3: Måling af motorens hastighed i grader

Vi besluttede os for at gå videre med disse målinger, og benyttede os af "tfest" funktionen i matlab til, ud fra vores data, at estimere en 2. ordens transfer funktionen.

$$\frac{74.34}{s^2 + 8.028s + 0.1365} \quad (1)$$

Vi ønsker at opstille vores system på state space form, da det gør det mulig at evaluere systemet ved hjælp af matrix algebra. State space er specielt brugbart i systemer med flere input og flere outputs, i kontrast til klassisk kontrol metoder, så som PID-regulering, som bygger på komplekse lapace og fourier ligninger som skal transformeres frem og tilbage mellem tids- og frekvensdomænet. En af de helt store fordele ved state space kontrol er dens anvendelighed til et bredt spekter af systemer, det kan både anvendes på linær og ikke linære, tidsafhængelige og tids uafhængelige, single-input single-output og multiple-input og multiple-output systemer. State space bliver repræsenteret ved bare to ligninger. Første ligning, som ses på [Equation 2](#), giver forholdet mellem systemets nuværende state og inputtet til dens næste state. [Equation 3](#) viser at systemets output er afhængig af systemets nuværende state. På [Figure 4](#) ses et state space block diagram.

$$\dot{x} = A * x(t) + B * u(t) \quad (2)$$

$$y = C * x(t) + D * u(t) \quad (3)$$

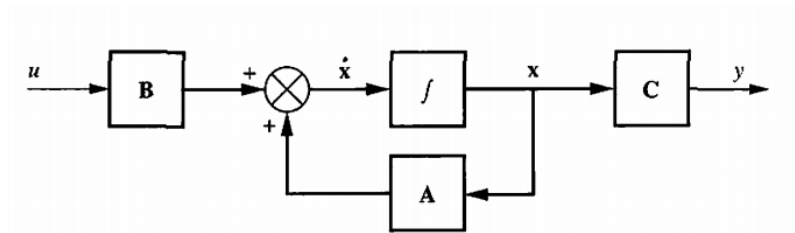
Hvor:

x er en vektor af alle state variabler

x' er en vektor af de afledte state variabler

A matrix er state matrixen

B matrix er input matrixen
 C matrix er output matrixen
 D matrix er feedforward matrixen



Figur 4: State space block diagram uden D

Derfor ønsker vi nu at få opstillet vores transfer funktion på state space form, det gør vi med matlab funktionen "tf2ss", som transformere vores transfer funktion til state space controllable canonical form. Denne state space repræsentation kan ses på [Equation 4](#) og [Equation 5](#). Vores D matrice er lige 0, og derfor ikke med i vores state space repræsentation.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.137 & -8.28 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} * u \quad (4)$$

$$y = \begin{bmatrix} 74.340 & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (5)$$

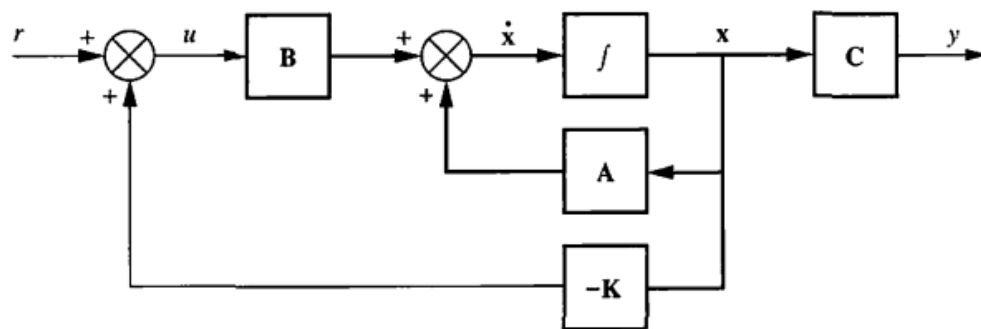
Vi har ud fra vores målinger estimeret en transfer funktion, og sat den op på state space form, og er nu klar til pol placering, som vil blive forklaret i næste afsnit.

3 Pol-placering

For at designe en lukket sløjfe controller for ens system, har vi gjort brug af pol-placerings princippet. Dette gøres ved at først at bestemme hvilken orden ens system har (første, anden osv.). Ud fra hvilken orden ens system har kan man lave en karakteristik ligning for denne. I vores tilfælde er det et anden-ordens system, hvilket for ligningen til at se således ud:

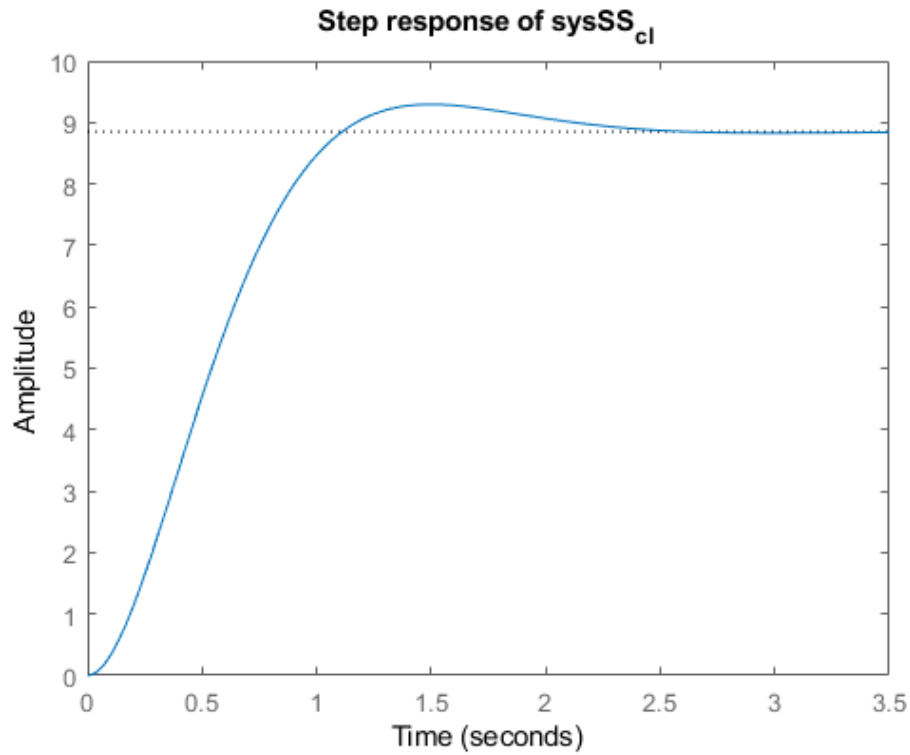
$$\frac{wn^2}{s^2 + 2 * \zeta * wn * s + wn^2} \quad (6)$$

Derefter skal vi bestemme nogle krav til systemet. Vi har valgt at vores overshoot skal ligge på 5% og vores settlingtime til 2 sekunder. Disse krav er forskellige fra system til system og bestemmes af hvad systemet skal bruges til. Når vi har valgt vores settlingtime og overshoot kan vi finde polerne for karakteristikligningen. Disse poler skal bruges til at designe vores controller, så den får den rette karakteristik, altså 5% overshoot og 2 sekunders settlingtime. Nedenunder kan man se blok opbygningen af et system med state-variable feedback.



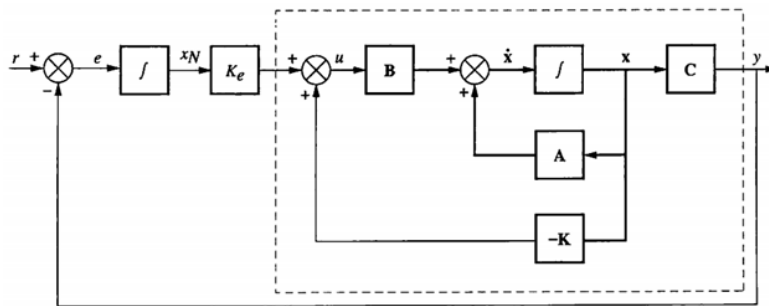
Figur 5: Blok diagram over vores controller

Vi gør brug af matlab funktionen place, som kan beregne vores state feedback matrix, K. Dette gør vi ved at indsætte state space variablerne A og B, som vi fandt frem til i starten og polerne fra vores karakteristik ligning. Dermed får vi beregnet vores feedback matrix så den passer med vores krav. Nu kan vi teste om vores controller fungerer efter hensigten ved at smide et stepinput ind i systemet og kigge på responsen. Dette kan ses nedenfor.



Figur 6: Steprespons for closed loop system

3.1 Steady state error



Figur 7: Matrix når tredje pol indsættes

For at fjerne steady state error skal en tredje pol indsættes

Man kan se på step responset at der er en stor steady state fejl. Denne fejl kan elimineres ved at indsætte endnu en pol i systemet. Denne pol skal gerne ligge så langt væk fra de eksisterende poler at det ikke har nogen indflydelse på systemets karakteristik. Til vores system vælger vi denne tredje pol til at ligge i -30.

Da vi tilføjer endnu en pol og dermed også gør det til et tredje ordens system, bliver vi nødt til at lave om på vores state space matricer A , B , C og D . De skal udvides så de kommer

til at se ud som på billedet nedenfor.

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{x}_N \end{bmatrix} = \begin{bmatrix} (\mathbf{A} - \mathbf{BK}) & \mathbf{BK}_e \\ -\mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ x_N \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} r$$

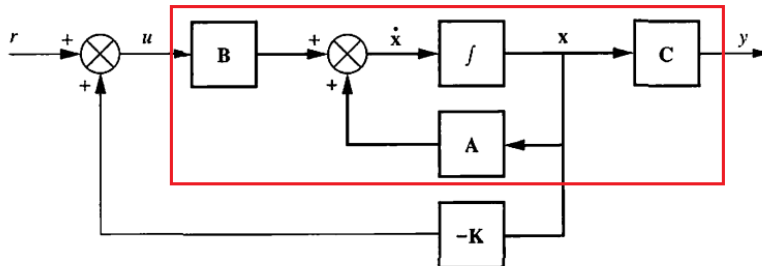
$$y = [\mathbf{C} \quad 0] \begin{bmatrix} \mathbf{x} \\ x_N \end{bmatrix}$$

Figur 8: Matrix når tredje pol indsættes

Denne matrix kan også laves på blokform.

4 Observer

I tidligere afsnit har vi beskrevet hvordan vi har designet vores controller. Men designet af en controller afhænger af at vi har adgang til state variablerne, så vi gennem feedback kan tillægge de ønskede forstærkninger. Måden hvorpå vi fik adgang til vores state variabler i sidste afsnit, vil give os et problem når vi ønsker at implementere vores controller på det virkelige system. På [Figure 9](#) kan man se hvordan vi hiver state variablerne ud direkte inde fra systemet, som er omridset med rød. Dette er ikke muligt på et virkeligt system.



Figur 9: State space block diagram uden D

Det er her vores observer design kommer ind i billedet. For hvis ikke det er mulig at tilgå de aktuelle states, grundet f.eks. omkostninger eller opbygningen af systemet. Er det i stedet mulig at estimere de aktuelle states via et observer design. Det er så disse estimerede states, frem for de aktuelle states, der bliver videregivet til controlleren. På [Equation 7](#) og [Equation 8](#) kan man se observereren som model af systemet. Hvis vi så sammenligner denne model med den faktiske model af systemet, ved at lave en subtraktion [Equation 9](#) og [Equation 10](#). Her bliver det tydeligt at forskellen mellem modellerne afhænger af forskellen på den faktiske state og den estimerede state.

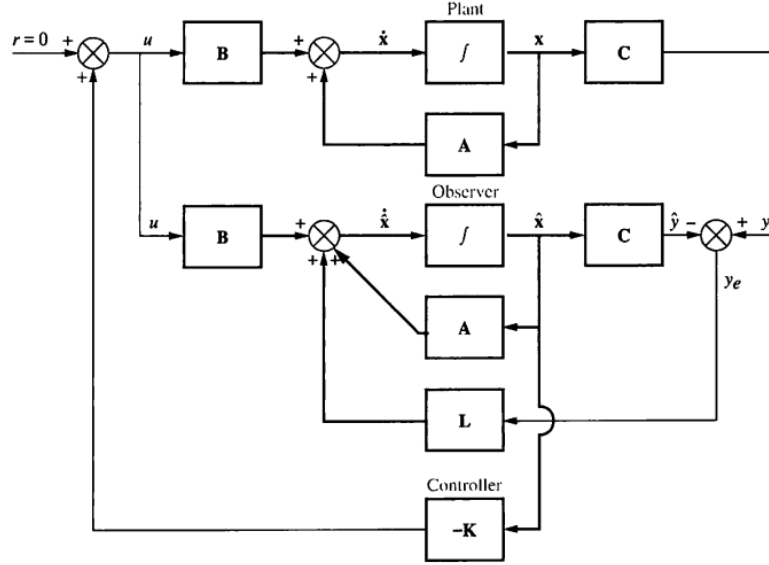
$$\dot{\hat{x}} = A * \hat{x} + B * u \quad (7)$$

$$\hat{y} = C * \hat{x} \quad (8)$$

$$\dot{x} - \dot{\hat{x}} = A * (x - \hat{x}) \quad (9)$$

$$y - \hat{y} = C * (x - \hat{x}) \quad (10)$$

Derfor er det vigtigt at vores estimerede state altid bliver beregnet meget hurtigere end den faktiske state. For at øge hastigheden for konvergens mellem den faktiske og estimerede state bruger vi et feedback og sammenligner de to outputs se [Figure 10](#). Fejlen mellem disse to outputs fører vi så tilbage til det afledte af observeres state. På den måde vil systemet forsøge at tvinge denne fejl til 0. Med dette feedback kan vi designe et transient respons som er meget hurtigere end fra det faktiske system.



Figur 10: Block diagram af systemet med observer og controller

Lige som da vi designede controller vektor, K , består vores observer design også af en konstant vektor, L . Denne vektor, L , skal designses så det transiente respons for observeren er hurtigere end responset fra controlleren. Derfor vælger vi at benytte de samme poler vi placerede for controlleren, vi flytter dem bare tilpas langt ud i det kontinuere domæne, så de bliver hurtige nok til ikke at have en betydelig indflydelse på dynamikken. I vores tilfælde virkede det at gange polerne med 30. State ligningen for observeren efter L er implementeret kan ses på [Equation 11](#) og [Equation 12](#).

$$\dot{\hat{x}} = A * \hat{x} + B * u + L(y - \hat{y}) \quad (11)$$

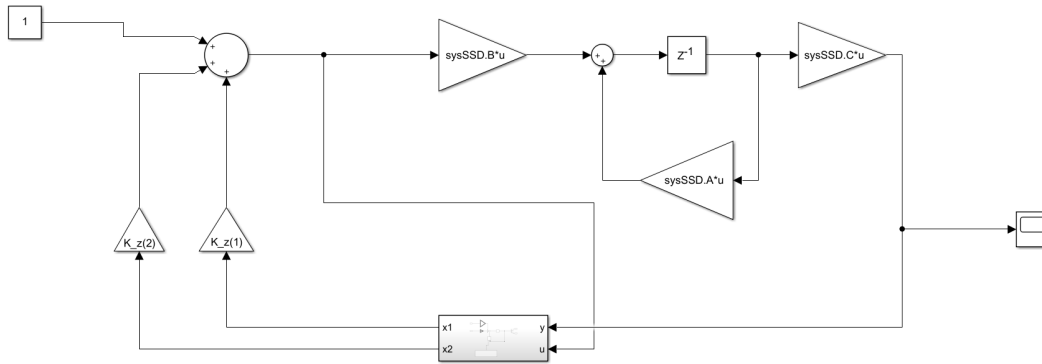
$$\hat{y} = C * \hat{x} \quad (12)$$

Det kan vi så omskrive til [Equation 13](#) og [Equation 14](#)

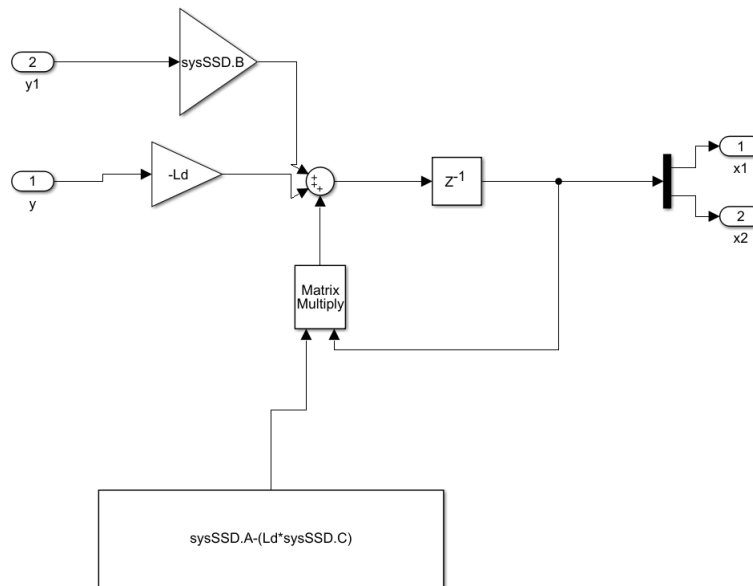
$$\dot{\hat{x}} = (A + B * K - L * C) * \hat{x} + L * y \quad (13)$$

$$u = K * \hat{x} \quad (14)$$

Det er [Equation 13](#) og [Equation 14](#) vi implementere i vores system. Denne implementering skete i Simulink og kan ses på [Figure 11](#) og [Figure 12](#). Hvor [Figure 11](#) er hele systemet med controller, og den lille kasse er lavet som et subsystem. [Figure 12](#) er så hvad der sker inde i maven på dette subsystem, som er vores observer.



Figur 11: Block diagram af systemet med observer og controller



Figur 12: Block diagram af systemet med observer og controller

Da vi ville implementere observer designet på vores system via Simulink, skulle vi først overføre vores system til det diskrete domæne, da vores system kører digitalt. Det gjorde vi ved "c2d" funktionen i matlab, som ses i den koden indsat under. Derudover har vi også benyttet os af "pole" funktionen som giver polerne ud, samt "place" funktionen som giver en vektor konstant.

```

1 %% Observer design in discrete form
2 sysSSD=c2d(sysSS,ts,'zoh'); %Discrete
3
4 ch_eqnD = wn^2/(s^2+2*zeta*wn*s+wn^2); %Characteristic equation
5 ch_eqnD = c2d(ch_eqnD,ts,'tustin');
6 polesD = pole(ch_eqnD);
7

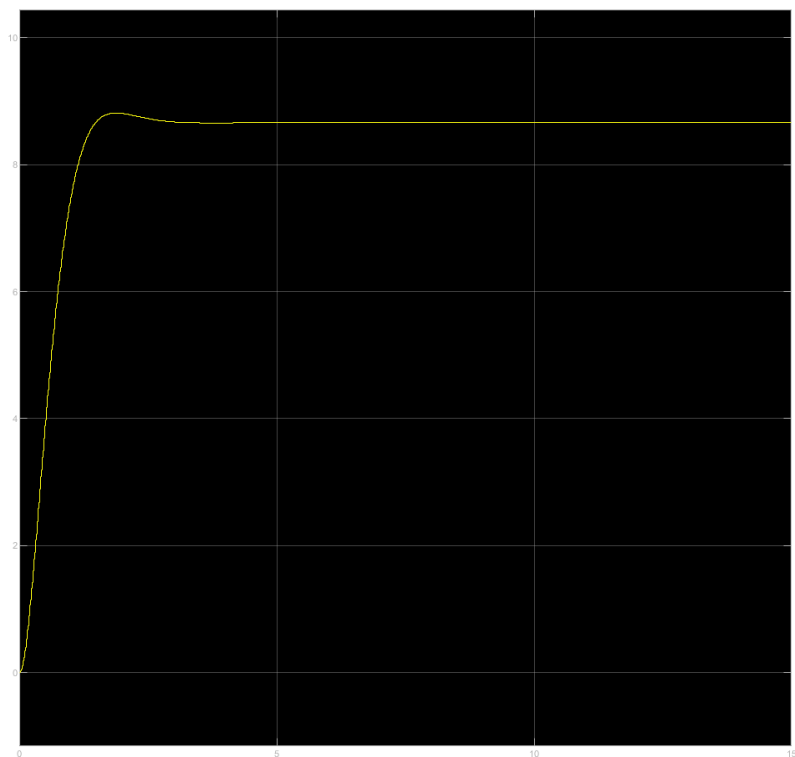
```

```

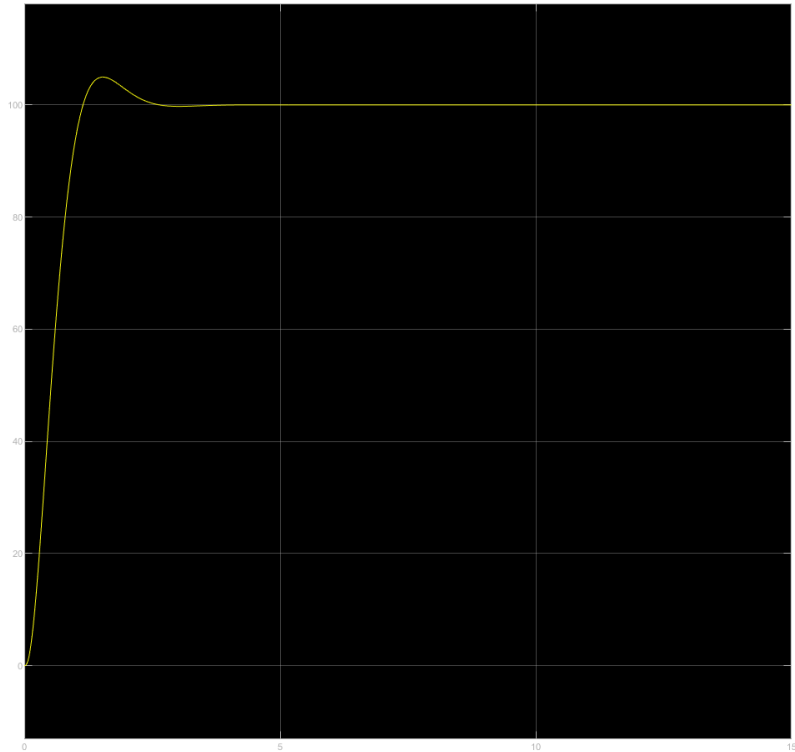
8 K_z = place(sysSSD.A,sysSSD.B,polesD);
9 A_cl = [sysSSD.A-sysSSD.B*K_z];
10
11 sysssD_cl = ss(A_cl,sysSSD.B,sysSSD.C,sysSSD.D,ts);
12
13 observer_reqZ = polesD/30;
14
15 Ld = (place(sysSSD.A',sysSSD.C',observer_reqZ))';

```

På [Figure 13](#) kan man se simulering af vores observer design i diskret domæne. Det er tydeligt at se at vores system er stabilt, desværre har vi en markant steady state fejl, som vi ikke har kunne fjerne. Hver gang vi har indsat en ekstra pol med forstrækning, blev vores system ustabil. Dog hvis vi sammenligner med vores closed loop opstilling uden korrigering for steady state fejl får vi meget tæt på det samme resultat. Se [Figure 14](#). Man kan se en lille ændre, hvilket er svært at undgå da, de ekstra poler der bliver tilføjet ved observeren påvirker dynamikken en smule.

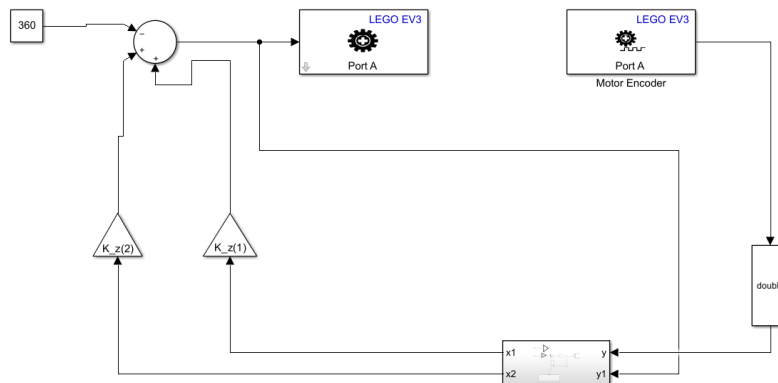


Figur 13: Block diagram af systemet med observer og controller



Figur 14: Block diagram af systemet med observer og controller

Til sidst har vi så implementeret dette observer design på vores bil. Simulink opstillingen af dette kan ses på figur [Figure 15](#). Bilen opførte sig helt efter forventningen, og kørte ca 8-9 gange længere end inputet, hvilket passer med steady state fejlen.



Figur 15: Block diagram af systemet med observer og controller

Litteratur