Aarhus universitet

Adaptive Control and Automation

7. Semester

ACA projekt

Gruppemedlemmer:	*AUID*
Daniel Tøttrup	au544366
Stinus Lykke Skovgaard	au520659

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

18. 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 **11**

5 Adaptiv controller **16**
5.1 MIAC 16
5.1.1 RLS online 16

Figurer

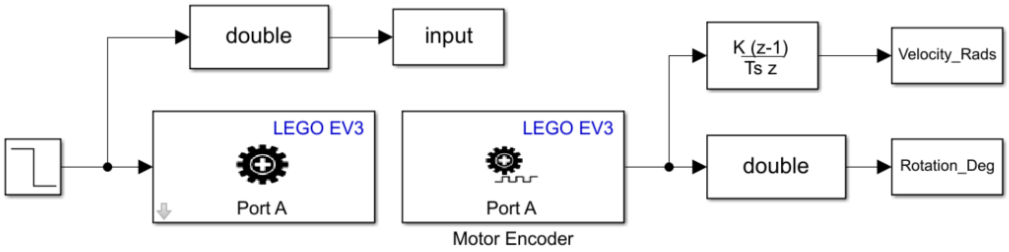
1 Simulink opstilling af måling 4
2 Måling af motor vinkel i grader 4
3 Måling af motorens hastighed i grader 5
4 State space block diagram uden D 6
5 Blok diagram over vores controller 7
6 Steprespons for closed loop system 8
7 Matrix når tredje pol indsættes 8
8 Matrix når tredje pol indsættes 9
9 Endelig matrix når steady state error skal fjernes 9
10 Step respons uden steady state error 10
11 State space block diagram uden D 11
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 13
15 Block diagram af systemet med observer og controller 14
16 Block diagram af systemet med observer og controller 15
17 Block diagram af systemet med observer og controller 15
18 MIAC blokdiagram 16
19 Parameter vektor 17
20 Målte output og input værdier 17
21 Formel for thetahat for RLS online 17

1 Indledning

Til dette eksamensprojekt var der opstillet nogle krav til selve systemet, som vi skulle arbejde på. Systemet skulle være et dynamisk system, med mindst én pol. Systemet skulle være stabilt, og tilgå mindst én aktuator og én sensor via EV3 hardware, hvor igennem regulatoren også skulle implementeres.
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.

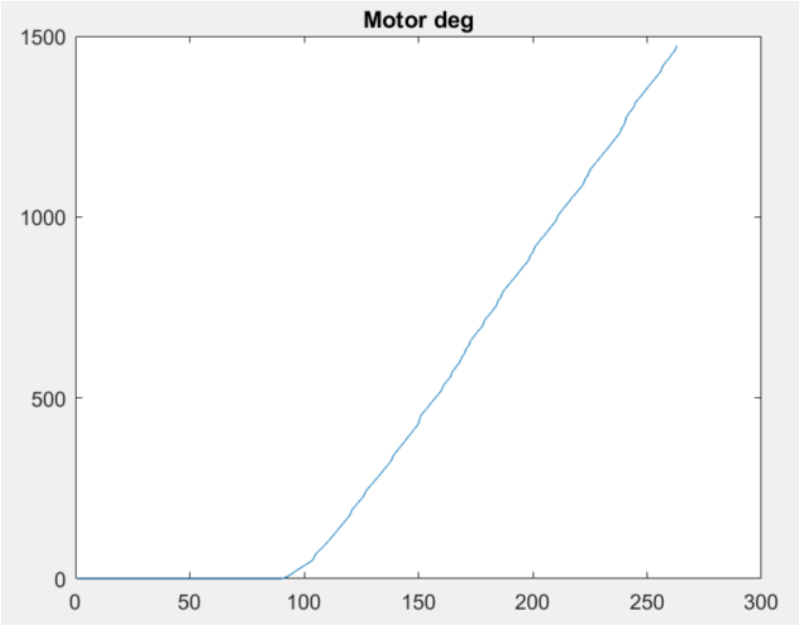
2 Identifikation af open-loop system

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 encoderen, 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.

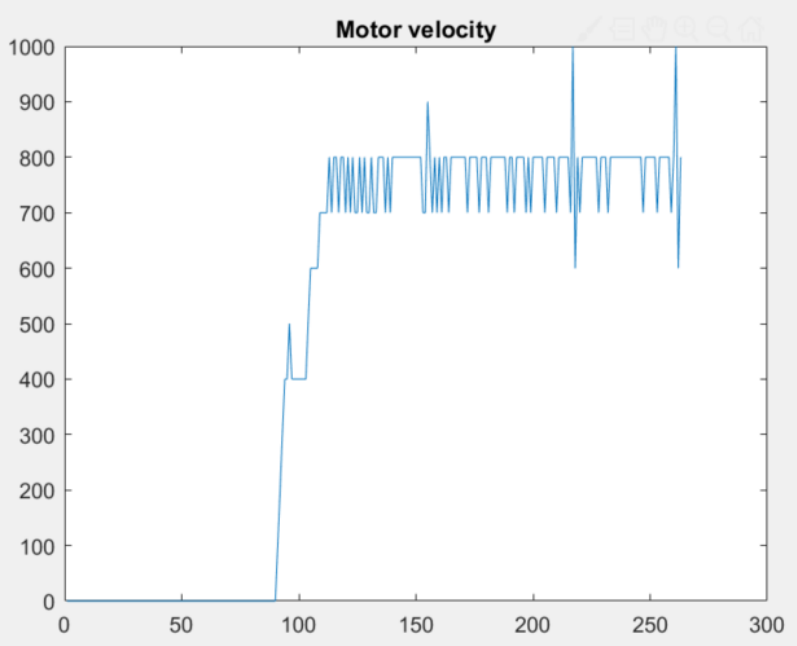


Figur 1: Simulink opstilling af måling

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.

$$\frac{74.34}{s^2 + 8.028s + 0.1365} \tag{1}$$

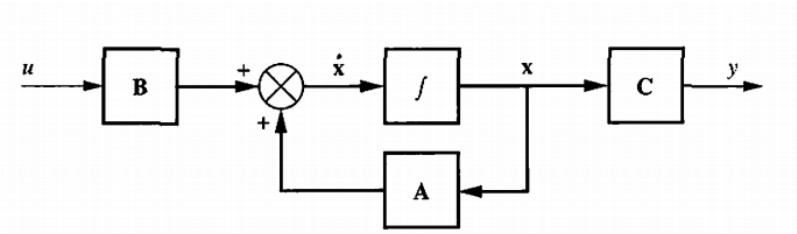
Vi ønsker at opstille vores system på state space form, da det gør det mulig at evaluere systemet ved hjælp af matrix algebra. State space er specielt brugbart i systemer med flere input og flere outputs, i kontrast til klassisk kontrol metoder, så som PID-regulering, som bygger på komplekse lapace og 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 uafhændelige, 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 den afledte state. Equation 3 viser at systemets output er afhængig af systemets nuværende state. På Figure 4 ses et state space block diagram.

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

$$y = C * x(t) + D * u(t) \tag{3}$$

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.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.137 & -8.28 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} * u \tag{4}$$

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

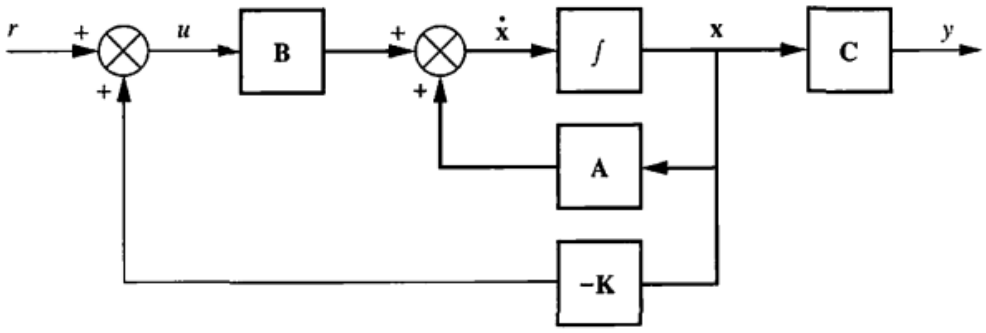
Vi har ud fra vores målinger estimeret en transfer funktion, og sat den op på state space form, og er nu klar til pol placering, som vil blive forklaret i næste afsnit.

3 Pol-placering

For at designe en lukket sløjfe controller for ens system, har vi gjort brug af pol-placerings princippet. Dette gøres ved 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:

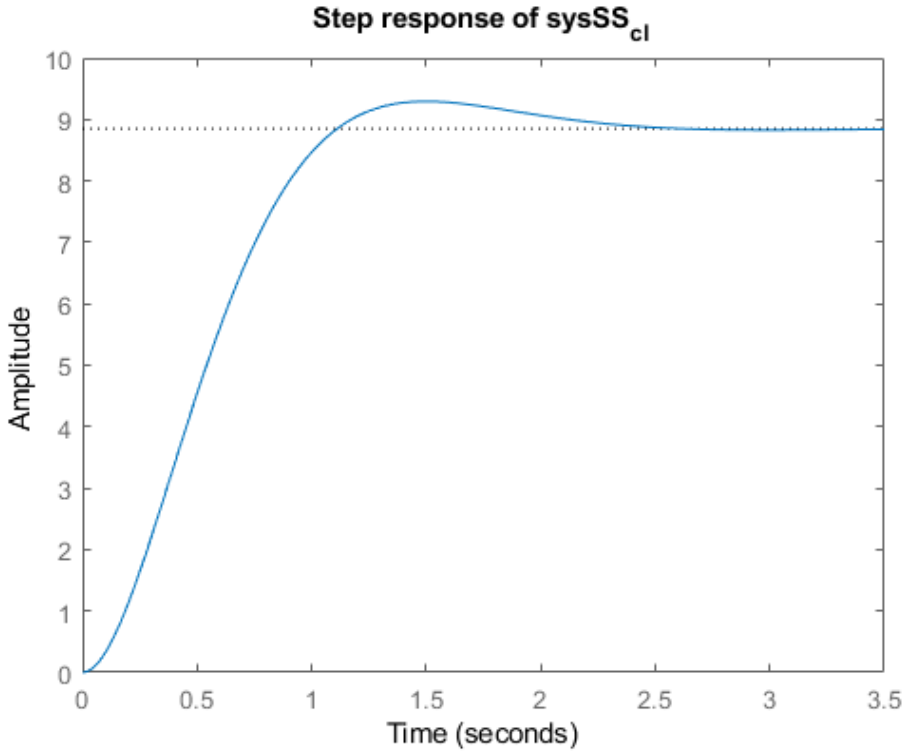
$$\frac{wn^2}{s^2 + 2 * \zeta * wn * s + wn^2} \tag{6}$$

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 på Figure 5



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 responset. Dette kan ses på Figure 6.

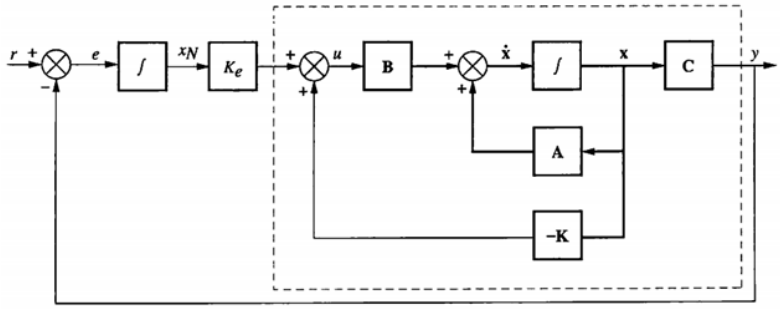


Figur 6: Steprespons for closed loop system

Man kan se systemet har det rette overshoot på 5% og en settling time på 2 sekunder, dog er der en betydelig steady state fejl da den ender helt oppe i ca 9. Dette betyder at bilen vil køre næsten 9 gange for langt. Dette er ikke optimalt og burde derfor rettes.

3.1 Steady state error

Man kan se på step responset på Figure 6, at der er en stor steady state fejl. Man kan se på Figure 5, at tilbagekoblingen tages fra punktet x, som ligger før C-blokken. Da tilbagekoblingen ikke trækkes fra undgangen y, kommer der højst sandsynligt en form for steady state error.



Figur 7: Matrix når tredje pol indsættes

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.

På Figure 7 kan man se hvordan en ny tilbagekobling introduceres fra punktet y, der derefter trækkes fra referencen r. Derudover er der også kommet en ekstra integrator og gain blok ind i systemet. Det betyder at steady state fejlen lige så stille bliver nul.

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{x}_N \end{bmatrix} = \begin{bmatrix} \mathbf{A} & 0 \\ -\mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ x_N \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} u + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} r$$

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

Figur 8: Matrix når tredje pol indsættes

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 være et tredje ordens system.

Men da inputtet u nu har ændret sig, skal matrixen laves om endnu engang. U er nu:

$$u = -Kx + K_e x_n \tag{7}$$

Til sidst indsætter man u ind i matrixen og får den endelige matrix på Figure 9

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

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

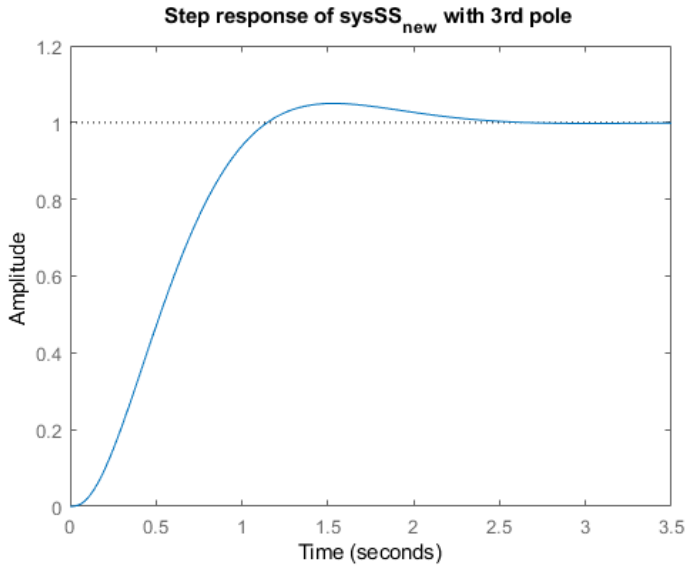
Figur 9: Endelig matrix når steady state error skal fjernes

Det har vi så gjort i matlab på følgende måde:

```
%% Insert 3rd pole to eliminate ss error
p3 = -30;
poles_new = [poles' p3];

A_new = [A [0;0];-C 0];
B_new = [B;0];
```

```
7
8  K_new = place(A_new,B_new,poles_new);
9  Ke = -K_new(3);
10 K_new = [K_new(1) K_new(2)];
11
12 A_cl_new = [A-B*K_new B*Ke;-C 0];
13 B_cl_new = [0;0;1];
14 C_cl_new = [C 0];
15
16 sysSS_new = ss(A_cl_new,B_cl_new,C_cl_new,D);
17
18 figure
19 step(sysSS_new);
20 title('Step response of sysSS_n_e_w with 3rd pole');
```

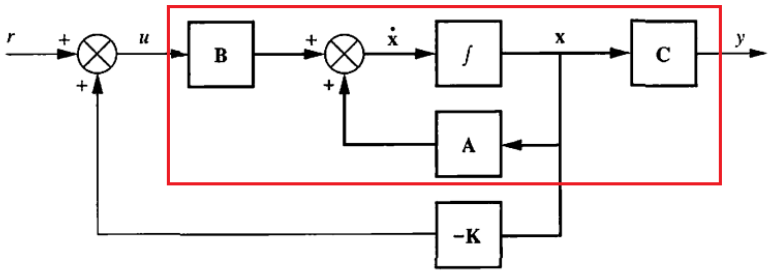
Hvis vi igen tester systemet med et step, kan man se at den stationære fejl nu er 0, og bilen vil altså nu kører præcis så langt, som vi siger den skal køre. Stepresponset kan ses på Figure 10



Figur 10: Step respons uden steady state error

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 11 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 11: 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 muligt 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 8 og Equation 9 kan man se observereren som model af systemet. Hvis vi sammenligner denne model med den faktiske model af systemet, ved at lave en subtraktion, som ses i Equation 10 og Equation 11. Her bliver det tydeligt at forskellen mellem modellerne afhænger af forskellen på den faktiske state og den estimerede state.

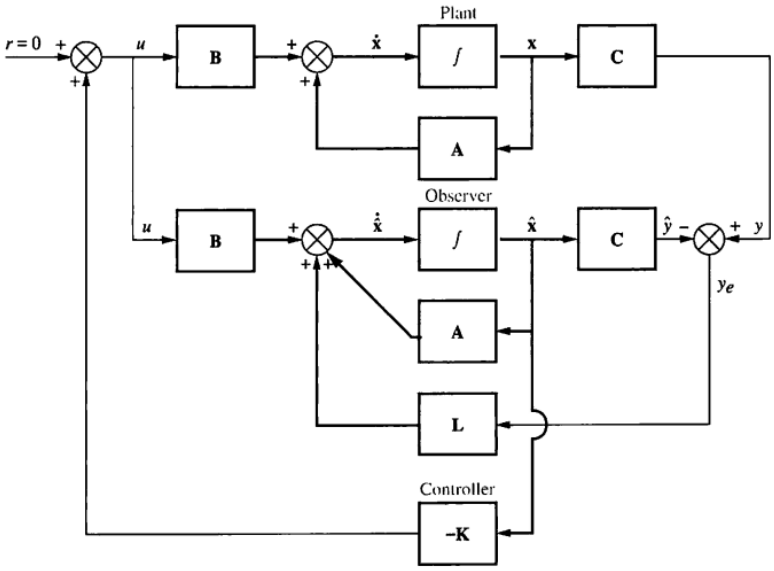
$$\dot{\hat{x}} = A * \hat{x} + B * u \tag{8}$$

$$\hat{y} = C * \hat{x} \tag{9}$$

$$\dot{x} - \dot{\hat{x}} = A * (x - \hat{x}) \tag{10}$$

$$y - \hat{y} = C * (x - \hat{x}) \tag{11}$$

Derfor er det vigtigt, at vores estimerede state altid bliver beregnet meget hurtigere end den faktiske state. For at øge hastigheden for konvergensen mellem den faktiske og estimerede state bruger vi et feedback og sammenligner de to outputs se Figure 12 . 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 det fra det faktiske system.



Figur 12: 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 designes, så det transiente respons for observereren 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 virkerede det at gange polerne med 30. State ligningen for observeren efter L er implementeret kan ses på Equation 12 og Equation 13 .

$$\dot{\hat{x}} = A * \hat{x} + B * u + L(y - \hat{y}) \tag{12}$$

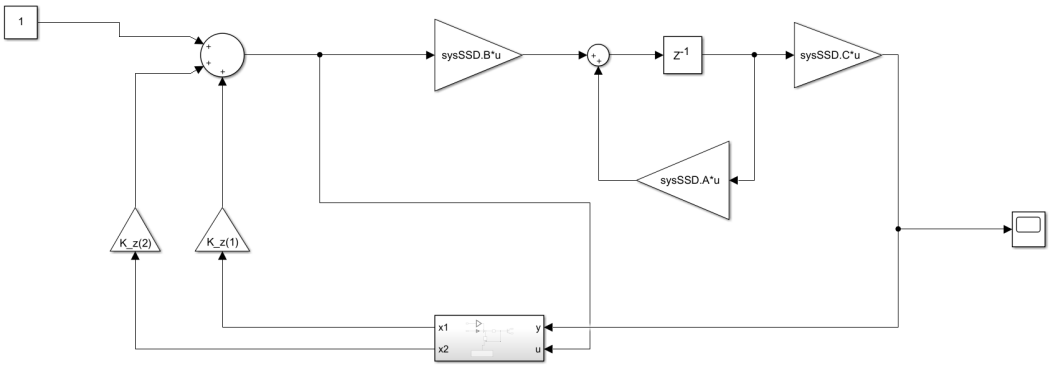
$$\hat{y} = C * \hat{x} \tag{13}$$

Det kan vi så omskrive til Equation 14 og Equation 15

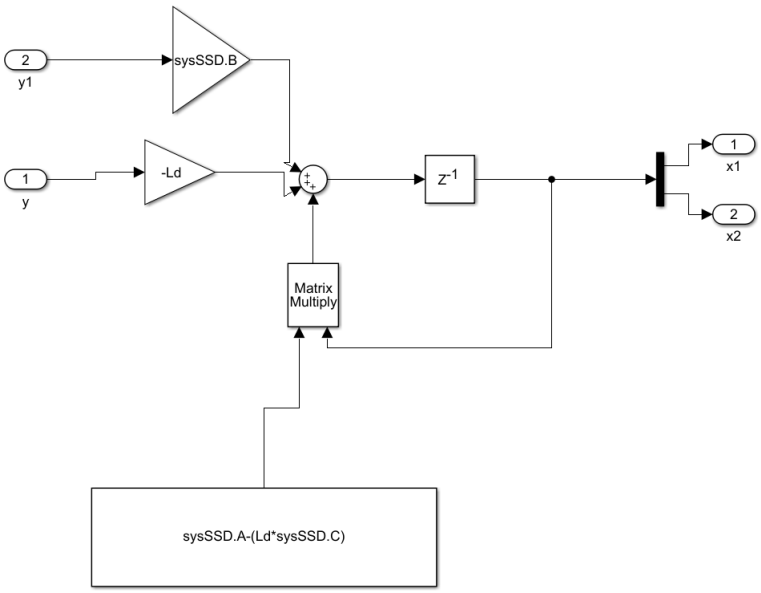
$$\dot{\hat{x}} = (A + B * K - L * C) * \hat{x} + L * y \tag{14}$$

$$u = K * \hat{x} \tag{15}$$

Det er Equation 14 og Equation 15 vi implementere i vores system. Denne implementering skete i Simulink og kan ses på Figure 13 og Figure 14. Hvor Figure 13 er hele systemet med controller, og den lille kasse er observeren lavet som et subsystem. Figure 14 er så hvad der sker inde i maven på dette subsystem, som er vores observer.



Figur 13: Block diagram af systemet med observer og controller



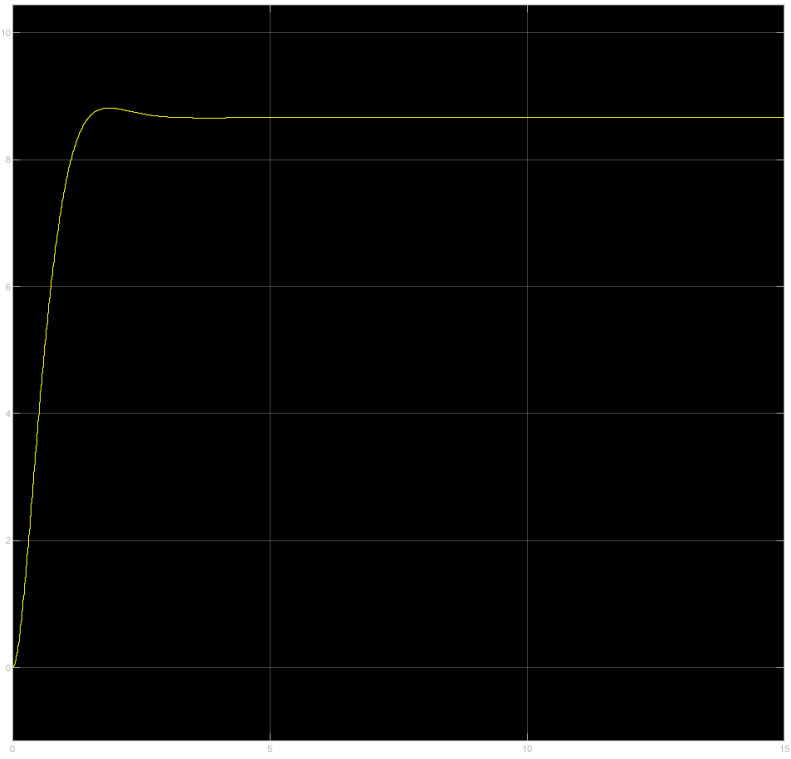
Figur 14: Block diagram af systemet med observer og controller

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"functionen i matlab, som ses i den koden indsat under. Derudover har vi også benyttet os af "pole"functionen som giver polerne ud, samt "place"funktionen som giver en vektor konstant.

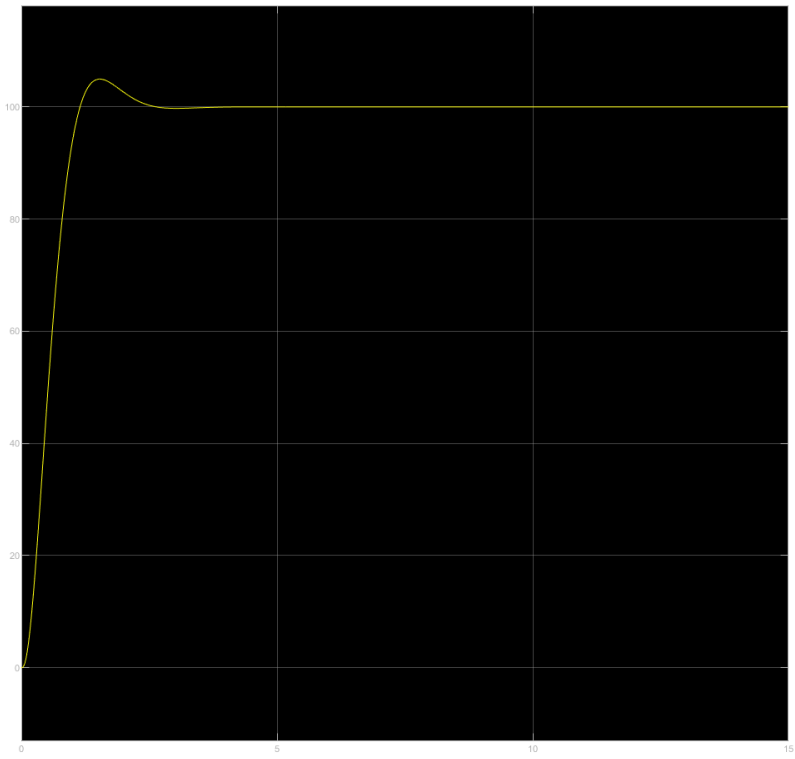
```
%% Oberserver design in discrete form
sysSSD=c2d(sysSS,ts,'zoh');%Discrete

ch_eqnD = wn^2/(s^2+2*zeta*wn*s+wn^2); %Charactaristic equation
ch_eqnD = c2d(ch_eqnD,ts,'tustin');
polesD = pole(ch_eqnD);

```

```
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 15 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 indsatte en ekstra pol med forstrækning, blev vores system ustabilt. 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 16. Man kan se en lille ændre, hvilket er svært at undgå da, de ektra poler der bliver tilføjet ved observeren påvirker dynamikken en smule.

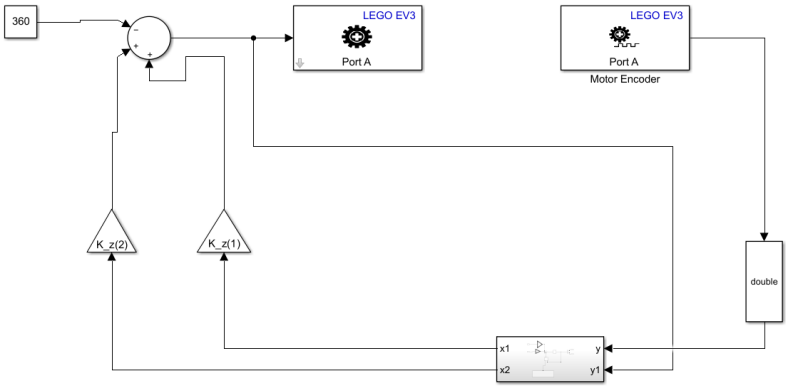


Figur 15: Block diagram af systemet med observer og controller



Figur 16: 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 17. Bilen opførte sig helt efter forventningnen, og kørte ca 8-9 gange længere end inputet, hvilket passer med steady state fejlen.



Figur 17: Block diagram af systemet med observer og controller

5 Adaptiv controller

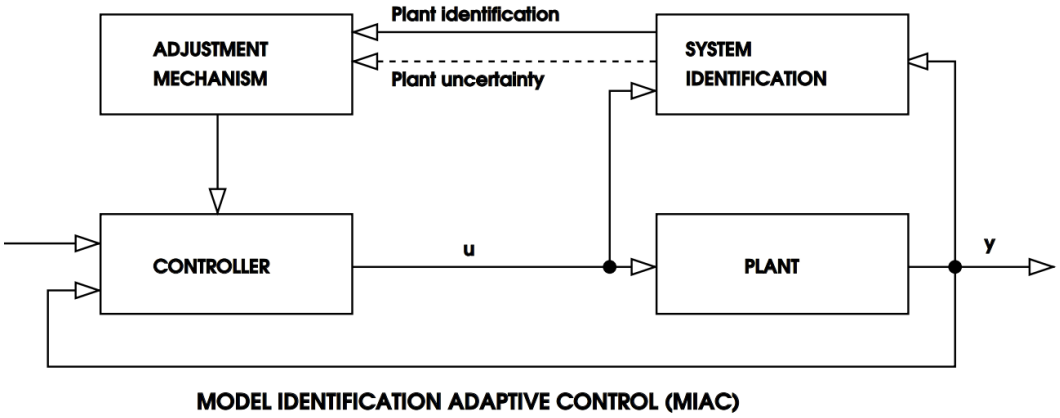
Fra start var det meningen at bilen skulle have en adaptiv controller, så selvom bilen bliver tungere, eller kører på et skævt underlag vil kunne fungerer lige så godt som før. Men på grund af tidsmangel er dette ikke implementeret i bilen, dog er der blevet gjort nogle overvejelser omkring hvordan sådan et system vil se ud.

5.1 MIAC

Vi har i gruppen valgt at bruge det adaptive system MIAC til dette projekt.

5.1.1 RLS online

For at bilen skal kunne klare ekstra vægt og et skævt underlag, kan man bruge to forskellige system estimerings algoritmer, RLS online eller RLS offline. For at bilen selv kan regulere for ekstra vægt eller en hældning, vil RLS online give mest mening. RLS online udregner løbende



Figur 18: MIAC blokdiagram

Til system identifikation kan RLS algoritmen bruges. Den fungerer ved løbende at udregne en parameter vektor, som bruges til at opdatere ens controller. Offline RLS vil ikke være det bedste valg til vores bil, da det kræver at man har hele outputtet til at lave estimation på. Det vil sige, at bilen først skal have kørt igennem den rute, man vil have den skal køre.

På Figure 18 vil RLS algoritmen implementeres i system identifikationsblokken.

$$\hat{\Theta} = \left(\sum_{n=1}^{N} \Phi(n)^T \Phi(n) \right)^{-1} \cdot \left(\sum_{n=1}^{N} \Phi(n)^T y(n) \right)$$

Figur 19: Parameter vektor

For at udregne parameter vektoren på Figure 19, skal man have sine målte output data og input data, kaldet phi her.

$$\Phi(n) = [-y(n-1), \cdots, -y(n-k), u(n-d), \cdots, u(n-d-m)]$$

Figur 20: Målte output og input værdier

Men da det kræver meget computerkræft at invetere matricer omskrives ligningen for thetahat.

$$R^{-1}(n) = \left(R(n-1)^{-1} + \Phi(n)^T \cdot 1 \cdot \Phi(n) \right)^{-1}$$

$$P(n) = P(n-1) - \frac{P(n-1) \cdot \Phi(n)^T \cdot \Phi(n) \cdot P(n-1)}{1 + \Phi(n) \cdot P(n-1) \cdot \Phi(n)^T}$$

Figur 21: Formel for thetahat for RLS online

I matlab kan det implementeres på denne måde:

```
%Initializering PsiPsi
PsiPsi = Psi'*Psi;
PsiY = Psi'.*output(1);
est_out(1)=0;
%Implementation af det digitale filter
for n = 2:length(Step)-1

Psi_y = [-output(n) Psi_y(1:end-1)];
Psi_u = [Step(n-1) Psi_u(1:end-1)];
Psi=[Psi_y Psi_u];

%Opdater paramter estimation
P=P-P*Psi'*Psi*P./(1+Psi*P*Psi');
K=P*Psi';
Thetahathat=Thetahathat+K*(output(n+1)-Psi*Thetahathat);

est_out(n)=Psi*Thetahathat; %Model output y_m(n)
end
```

Litteratur