Projekt i Prediktiv reglering

MPC-reglering av kula på bom

Anton Christensson, E09 anton.christensson@gmail.com
Erik Lingärde, E09 erik@lingarde.se

30 november 2012

Innehåll

1 Inledning

I detta projekt har målet varit att reglera labprocessen ”kula på bom” genom att använda *Model Predictive Control*. Detta implementerades med det Stanford-utvecklade verktyget cvxgen i kombination med Simulink.

2 Teori

2.1 Model Predictive Control

Model Predictive Control, eller *Receding Horizon Control*, är en regulatorstruktur som bygger på att regulatorn finner den serie styrutslag som minimerar kostnaden som till exempel reglerfel och styrutslag medför.

Principen bygger på att regulatorn har tillgång till en matematisk modell av processen samt vilka begränsningar som finns på till exempel styrsignal, förändringshastighet av styrsignal och processens utsignal. Dessutom specificeras en kostnadsfunktion, alltså den funktion som skall minimeras. Kostnadsfunktionen består främst av reglerfelet, men ofta ingår även andra komponenter som till exempel hur stora styrsignaler som krävs, vilket kan vara intressant för att minimera energiförbrukning eller ge en mjukare gång. De olika komponenternas viktning i kostnadsfunktionen kan ställas in för att ge dem olika prioritet, ofta är ett minimalt reglerfel viktigare än att använda försiktiga styrsignaler.

Vid varje sampel beräknar regulatorn den serie styrutslag som minimerar kostnadsfunktionen över en fördefinierad tid, *prediction horizon*, med hänsyn till de begränsningar som finns definierade. Normalt används endast den första styrsignalen i serien eftersom en ny optimering utförs nästa sampel. Hur lång *prediction horizon* som väljs beror på regulatorns prestanda. En längre tid ger (inom rimliga gränser) en bättre reglering, men å andra sidan blir optimeringen mer komplicerad och denna måste hinna utföras med den frekvens i vilken regleringen sker. Vill man ha längre *prediction horizon* än vad som kan beräknas varje sampel är ett alternativ att endast utföra en optimering med några sampels mellanrum och använda flera styrsignaler i serien.

2.2 Cvxgen

Cvxgen är ett verktyg som kan användas för att generera mycket snabb C-kod för lösning av konvexa optimeringsproblem. I praktiken definierar man sitt optimeringsproblem, dess bivillkor och begränsningar med kvadratisk programmering i verktygets gränssnitt, varefter en bedömning av problemets omfattning görs. Då verktyget fastställt att kod med rimlig prestanda kan genereras fås resultatet som C-kod. Verktyget innehåller också hjälpmedel för generering av mex-gränssnitt för användning av det resulterande programmet i matlab.

2.3 Process

Processen som skulle regleras, ”kula på bom”, kan lite förenklat sägas ha en dynamik från motorström till kulans position på bommen som beskrivs av en trippelintegrator. Det vill säga

$$G_{tot}(s) = G_{\phi}(s) \cdot G_x(s) = \frac{k_u}{s} \cdot \frac{-7}{s^2}$$

där det är experimentellt fastställt att $k_u \approx 4.4$ och där $G_x(s)$ gäller för små värden hos ϕ och $\dot{\phi}$.

referens

3 Genomförande

3.1 Process

Processens överföringsfunktion diskretiserades i matlab. Samplingshastigheten valdes till 20 Hz på inrådan från projekthandledaren, då detta borde vara en tillräcklig hastighet med hänsyn till processen dynamik. En alltför hög samplingsfrekvens hade inte gått att realisera eftersom optimeringsberäkningen är krävande.

Systemet skrevs om på tillståndsrepresentation för hantering i programvaran. I den tillståndsrepresentation som valdes var det dock inte möjligt att direkt relatera till några hos processen mätbara parametrar. På grund av detta behövde tillstånden skattas på annat vis, och därför designades ett Kalmanfilter.

3.2 Optimering

Optimeringen utfördes med en kostnadsfunktion som främst straffar reglerfelet. Även ett (mindre) straff på styrsignalen fanns dock med för att ge en mindre ryckig gång. Olika viktning av dessa termer testades för att hitta en, i någon mån, optimal kvot mellan dem. En lämplig konfiguration visade sig vara att straffa reglerfelet 20 gånger mer än styrsignalen.

För att hindra kulan från att ramla av bommen, till exempel vid överslängar i regleringen, sattes en positionsbegränsning vid de båda ändlägena. Även styrsignalen begränsades för att garanterat hålla sig under processens maxnivåer. För att undvika att kulan lyfter från bommen vid hastiga styrsignalförändringar sattes också en begränsing på styrsignalens förändringshastighet, en så kallad *slew rate limit*. Denna bestämdes experimentellt för att kunna hantera kulor med olika vikt. Ett problem med cvxgen som fanns vid implementeringen av denna begränsning var att om den lades under *subject to* i optimeringskoden tycktes det som att optimeringen inte tog hänsyn till begränsningen över huvud taget. När detta villkor istället lades under *constraints* fungerade begränsningen som avsett.

Eftersom kulans benägenhet att lyfta från bommen är större ute vid bommens ändar än i mitten skulle en idé vara att bruka en positionsberoende *slew rate limit*. Vi lyckades dock inte implementera detta framgångsrikt i optimeringsalgoritmen. Dessutom observerade vi att regulatorn ändå inte ger särskilt kraftiga styrutslagsförändringar när kulan befinner sig nära bommens mitt utan främst när den är ute i ändarna, varför denna idé slopades.

Med en *prediction horizon* på 40 samplingar var problemet i den största storleksordning som cvxgen kan generera effektiv kod för att lösa. Detta motsvarar en tid på cirka 2 sekunder och var den prediktionstid som i slutändan användes.

3.3 Simulink

För att i detta projekt slippa hantera de nödvändiga realtidsoperationerna som krävs för regleringen som möjliggör kontroll av processen används simulink för

att binda samman de olika delarna av vår regleralgoritm. Simulink möjliggör även simulering av systemet och kommunikation med den verkliga processen.

För att använda den C-kod som cvxgen skapar i matlab används ett mex-gränssnitt. Att anropa detta från simulink låter sig dock inte göras helt smärtfritt, då det vanliga matlabfunktionsblock som använder sig av kodgenerering inte har stöd för generering vid anrop av denna filtyp. För att komma runt detta användes istället ett block som kallas *Interpreted MATLAB Fcn*. Vid användning av denna typ av block görs ingen kompilering till C-kod för den anropade koden i simulink, vilket ger längre exekveringstider. I vårt fall är detta inte kritiskt för regleringen på grund av den relativt låga samplingsfrekvens som används, men det kan vara värt att notera med tanke på att programmet från cvxgen är speciellt optimerat för snabb exekvering. För system där högre prestanda eftersträvas kan detta således bli kontraproduktivt. Den typ av block som användes har bara stöd för en insignal, men både de skattade tillstånden och referenssignalen behövdes för att kunna utföra den optimering som görs för varje sampling. För att komma runt detta problem valde vi att konkatinera vektorn med de skattade tillstånden med referensvärdet, för att sedan ”packa upp” de olika värdena igen inuti blocket som anropar optimeringsprogrammet.

Kalmanfiltret som användes för att skatta processens tillstånd implementerades med ett matlabfunktionsblock med stöd för kodgenerering, och i en enkel simulering med pålagt brus visade sig filtret kunna skatta tillstånden med tillfredsställande prestanda.

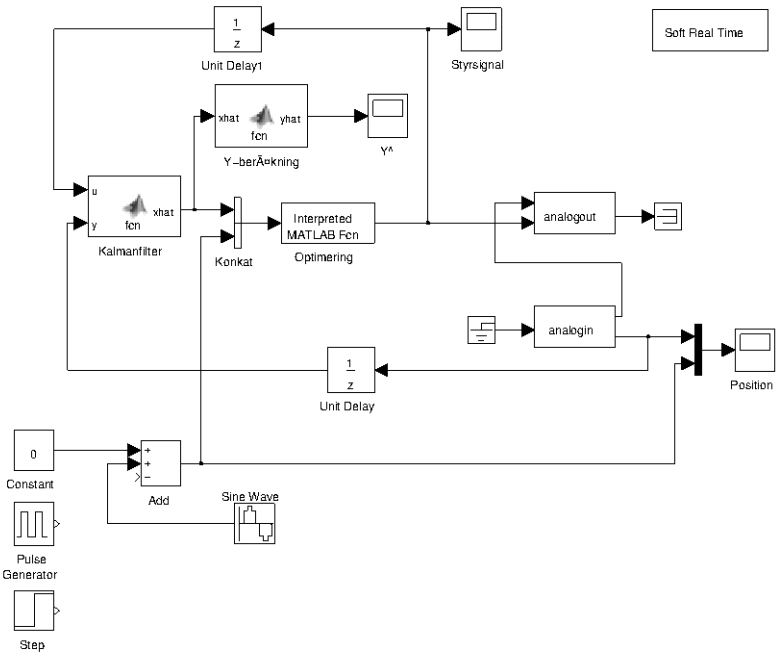
För att undvika problem med så kallade ”algebraiska loopar” i simulinkmodellen blev vi tvungna att införa fördröjningar i de återkopplingar som används. Eftersom dessa endast fördröjer signalen ett sample görs bedömningen att detta inte rimligtvis kan påverka regleringen i någon större omfattning med tanke på processens beskaffenhet i förhållande till samplingsfrekvensen.

Ett problem som upplevdes i samband med att simulinksystemet kördes för den verkliga processen var att exekveringen inte skedde i realtid, utan istället tycktes köra så snabbt datorn kunde – detta trots att alla kritiska block konfigurerats för avsedd samplingsfrekvens. Lösningen till detta problem blev att inkludera ett tredjepartsimplementerat block som tvingar simulink att exekvera i realtid.

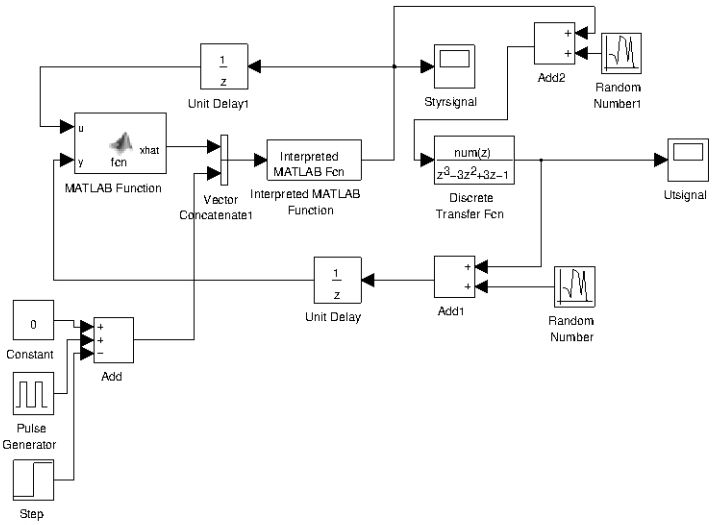
3.4 Verklig process

För att applicera regleringen på den verkliga processen byggdes en ny Simulinkmodell upp, i princip identisk med modellen för simulering, men där styrsignal och mätsignal skickas via labdatorns analoga in- och utgångar med hjälp av speciella simulinkblock som tillhandahölls av projekthandledaren. Olika referensblock användes för att styra kulan enligt till exempel en sinus- eller fyrkantsvåg. Se figur 1.

Det visade sig dock att den verkliga processen som väntat inte uppvisade precis samma dynamik som det simulerade systemet med den diskretiserade överföringsfunktionen. Detta kan till viss del bero på oprecis modellering av systemet, och även på för höga brusnivåer vid simulering av systemet. Därför krävdes det en del arbete med att justera de olika viktningarna och parametrarna för att komma från den reglering som gav bra resultat för den simulerade processen till att uppnå inställningar som fungerade i verkligheten. Till de ändringar som gjordes hör bland annat ökad prediktionshorisont, ökad kostnad för



Figur 1: Simulinkmodell för reglering av den verkliga processen.

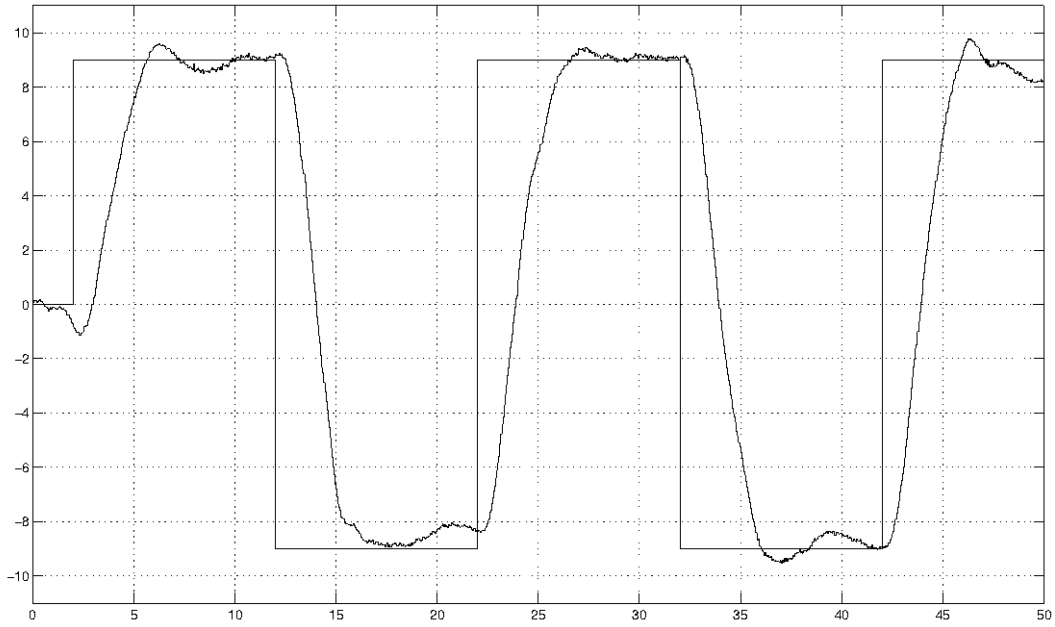


Figur 2: Simulinkmodell för simulering av reglersystemet.

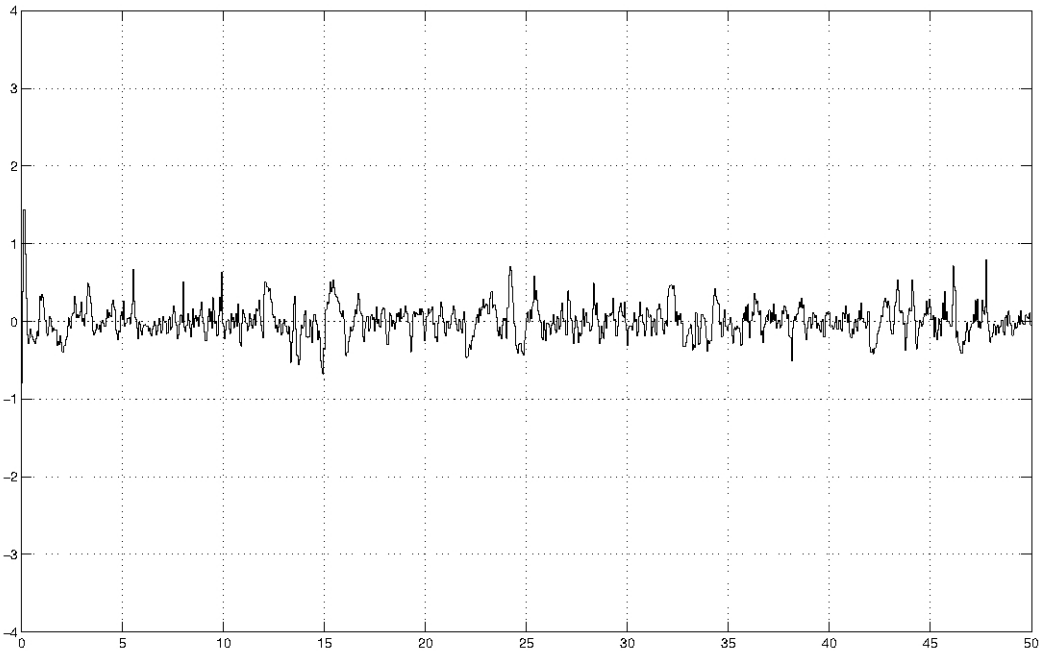
positionsavvikelse från referens och minskad *slew rate*.

4 Resultat

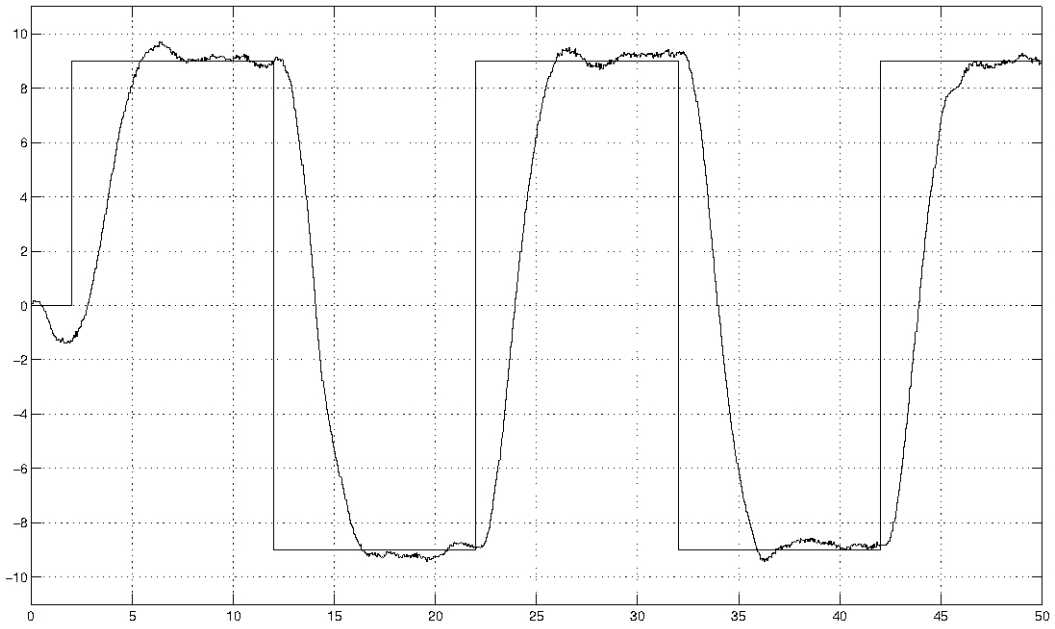
Projektet resulterade i ett reglersystem som klarar att reglera den aktuella processen för olika storlekar på kulan som ska balanseras på bommen. Det finns utrymme för justering av hur den slutliga regleringen ska te sig, och det handlar då i vanlig ordning framför allt om en avvägning mellan stabilitet och snabbhet. I det här fallet valdes relativt stabila inställningar, men snabbheten blir trots detta acceptabel eftersom dessa inställningar som regel ger små överslängar vid ändrat referensvärde. Detta innebär att en förflyttning från ena änden av bommen till den andra tar ungefär fyra sekunder, vilket kan ses i figur 3, 5 och 7.



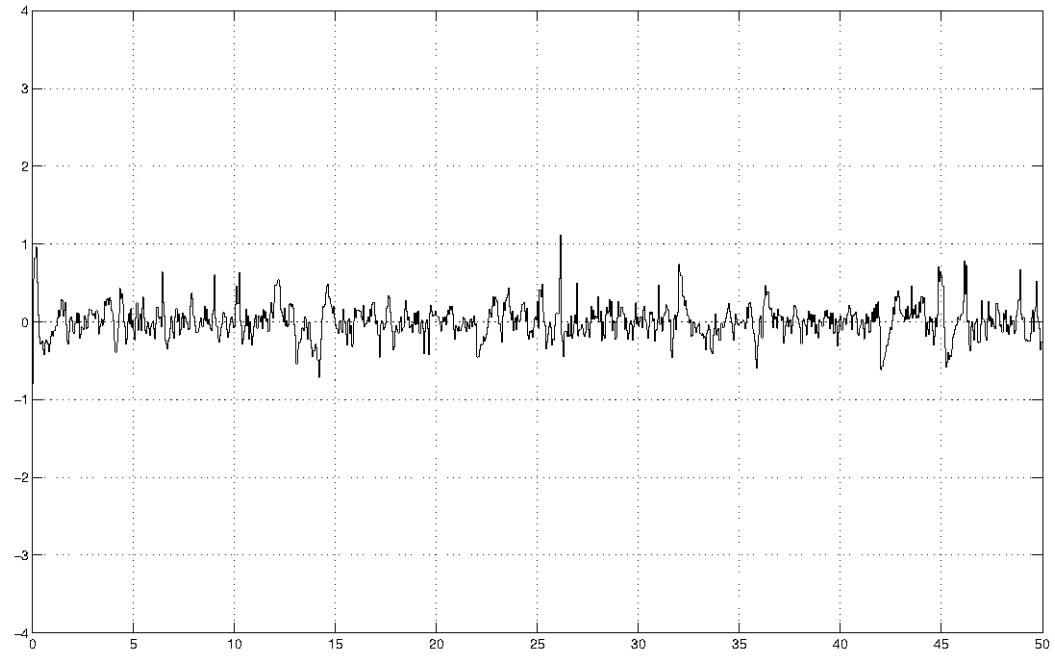
Figur 3: Den minsta kulans referensföljning.



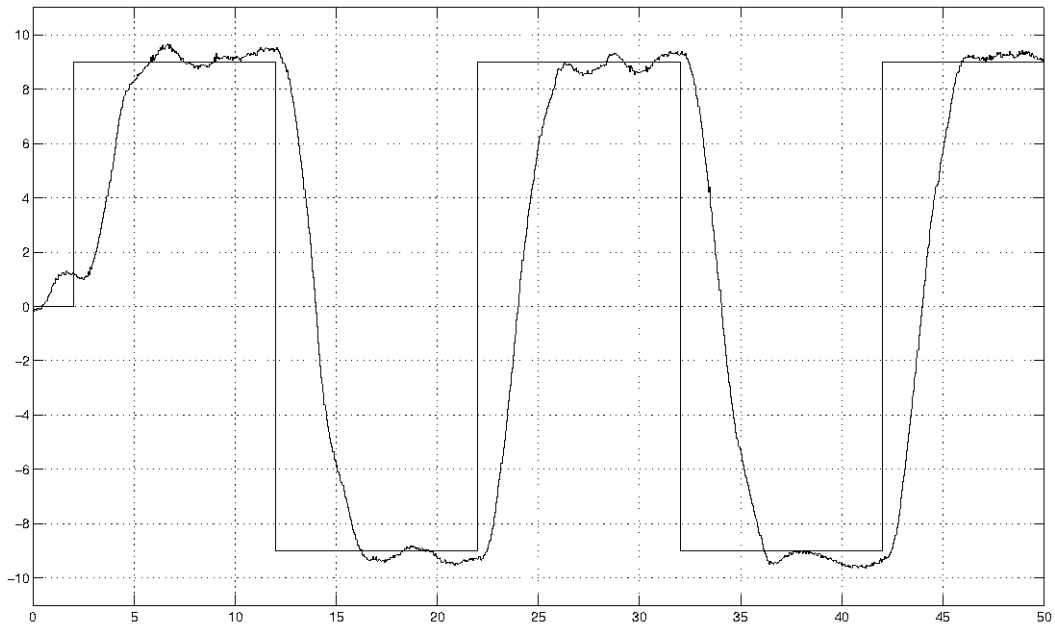
Figur 4: Styrsignal vid reglering med den minsta kulan.



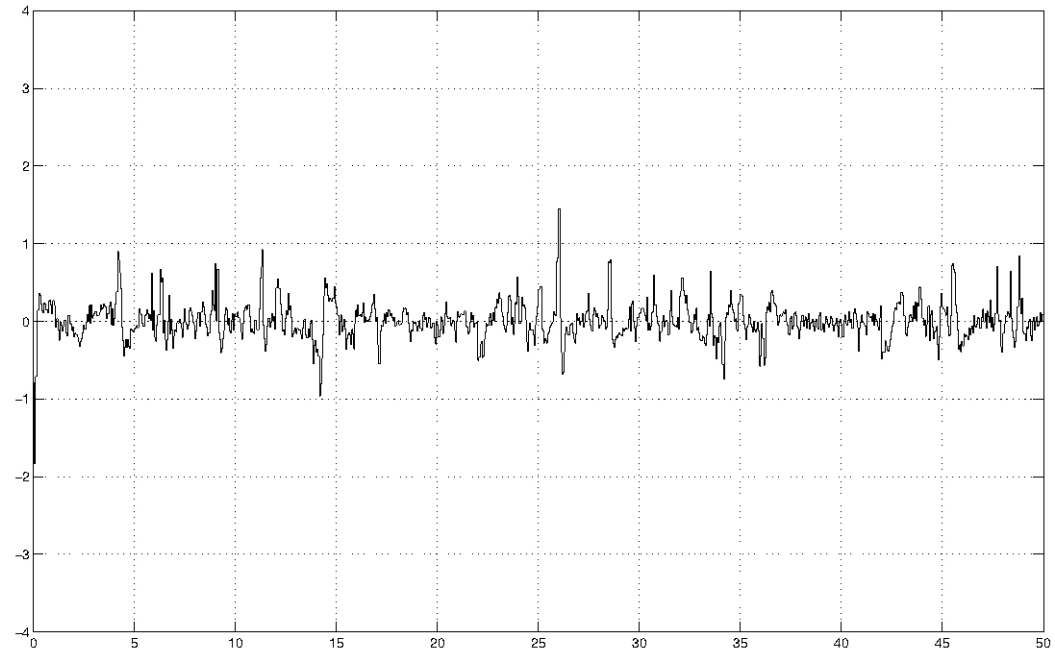
Figur 5: Den mellanstora kulans referensföljning.



Figur 6: Styrsignal vid reglering av den mellanstora kulan.



Figur 7: Den största kulans referensföljning.



Figur 8: Styrsignal vid reglering av den största kulan.

A Optimeringskod för cvxgen

Nedan följer problembeskrivningen till cvxgen:s kodgenerering.

```
dimensions
  m = 1  # inputs.
  n = 3  # states.
  T = 40  # horizon.
end

parameters
  A (n,n)  # dynamics matrix.
  B (n,m)  # transfer matrix.
  C (m,n)  # output matrix.
  Q (m,m) psd  # state cost.
  R (m,m) psd  # input cost.
  ref (m) # position reference.
  x[0] (n)  # initial state.
  u_max nonnegative # amplitude limit.
  y_max nonnegative # position limit
  S nonnegative  # slew rate limit.
end

variables
  x[t] (n), t=1..T+1  # state.
  u[t] (m), t=0..T  # input.
end

minimize
  sum[t=0..T](quad(C*x[t]-ref, Q) + quad(u[t], R))

subject to
  x[t+1] == A*x[t] + B*u[t], t=0..T  # dynamics constraints.
end

constraints
  abs(u[t+1] - u[t]) <= S, t=0..T-1  # slew rate constraint.
  abs(C*x[t]) <= y_max, t=1..T  #position constraint.
  abs(u[t]) <= u_max, t=0..T  # maximum input box constraint.
end
```

B Programkod för Kalmanfiltret

Nedan följer MATLAB-koden för Kalmanfilterblocket i Simulink.

```
function xhat = fcn(u, y)
persistent A B C Qe Qv Qve P xhatt
if isempty(A)
    A = [3 -1.5 0.5; 2 0 0; 0 1 0];
    B = [0.0625 0 0]';
    C = [-0.0103 -0.0205 -0.0051];
    Qe = 0.01;
    Qv = 0.01*eye(3);
    Qve = 0*[1 1 1]';
    P = eye(3);
    xhatt = [1 1 1]';
end

yhatt = C*xhatt;
R = Qe + C*P*C';
K = (A*P*C'+Qve)/(Qe+C*P*C');
P = A*P*A' + Qv - A*P*C'*(1/R)*C*P*A';
xhatt = A*xhatt+B*u+K*(y-yhatt);

xhat = xhatt;
```

C Programkod för anrop av optimering

Nedan följer programkoden för MATLAB-funktionen som kallar på optimeringsalgoritmen som genererats av cvxgen.

```
function u = testfcn(input)
%---Initialization---%
persistent params
if isempty(params)
    params.A = [3 -1.5 0.5; 2 0 0; 0 1 0];
    params.B = [0.0625; 0; 0];
    params.C = [-0.0103 -0.0205 -0.0051];
    params.Q = 20;
    params.R = 1;
    %params.x_0 = ones(3,1);
    params.u_max = 9;
    params.y_max = 10;
    params.S = 0.0001;
end
params.x_0 = input(1:end-1);
params.ref = input(end);
[vars, status] = csolve(params);

u = vars.u_0;
```