ISSN 0280-5316
ISRN LUTFD2/TFRT--5725--SE

Flexibelt linjärservo med inverterad pendel

Pelle Bergkvist
Jörgen Lindgren

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Oktober 2004

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 2004

Document Number
ISRNLUTFD2/TFRT--5725--SE

Author(s)
Pelle Bergkvist and Jörgen Lindgren

Supervisor
Anders Blomdell and Karl-Erik Årzén LTH, Lund

Sponsoring organization

Title and subtitle
Flexibelt linjärservo med inverterad pendel (A Flexible Linear Servo with an Inverted Pendulum)

Abstract
This master thesis is the first step in developing a new laboratory process for the automatic control basic course. The new process should illustrate the benefits of state-space control compared to PID control. A process with several outputs to control is then desired as a single PID controller does not support such a system. The suggested process consists of two carts mounted on a linear rail and connected by a spring, one driven by a DC motor and one trailing along holding a pendulum. The overall goal of the laboratory will be to balance the pendulum upright. This could be done with PID control but with the slight drawback of not being able to control the cart positions. Thus it would not be suitable for a real process of this kind since the rail is not infinitely long.
The state-space controller and observer needed to position the carts arbitrarily while balancing the pendulum are implemented in Matlab and Simulink environment, whereas the I/O interface is handled by an Atmel AVR microcontroller (ATMEGA8). The rail and cart chassis were bought from a local company and modified into modules, enabling the process to change into several shapes. The positioning hardware and driver circuits were developed at the department. The main effort has been to make all these parts work together. Programming the AVR in order to drive the machine, communicate with Simulink and handle position encoding has demanded some hard work. The report deals with issues considering simple D/A Pulse Width Modulation and positioning with state machine with interpolation for extended resolution. The process of modeling this system is also treated together with the basic control theory needed for accurate performance. This master thesis has resulted in a price worthy, fully functional and controllable LFJCIP-process for educational purpose.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
Swedish

Number of pages
58

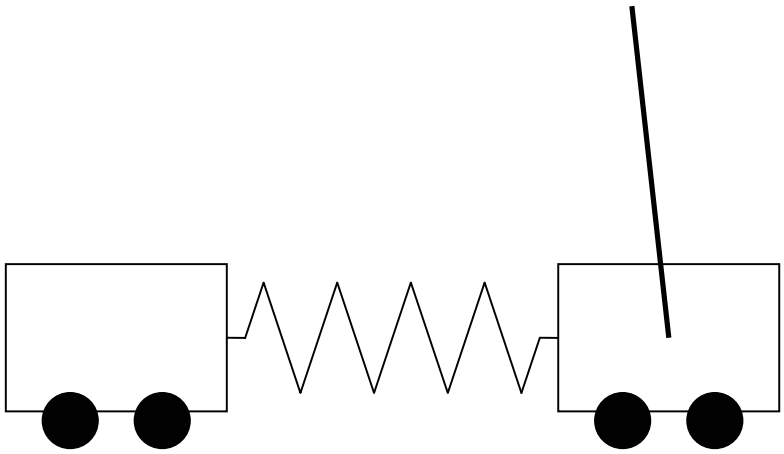
Recipient's notes

Security classification

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet
Department of Automatic Control

Examensarbete i reglerteknik, VT-04

Flexibelt linjärservo med inverterad pendel



Namn:	Pelle Bergkvist E98	pellebk@yahoo.se
	Jörgen Lindgren M99	cim99jl1@m.lth.se
Handledare:	Anders Blomdell	anders.blomdell@control.lth.se
	Karl-Erik Årzén	karlerik@control.lth.se

1 Inledning 2
1.1 Bakgrund 2
1.2 Laboration AK lab 3 2
1.3 Disposition 4
2 Processen 6
2.1 Linjärenhet 6
2.2 Vagnar 7
2.3 Pendel 7
2.4 Kravspecifikation 7
3 Motorer 8
3.1 Stegmotor 8
3.2 Likströmsmotor 8
4 Givare 10
4.1 Linjärgivare 10
4.2 Vinkelgivare 12
5 Datorsystem 13
6 Modell 14
6.1 Kraftanalys och systemekvationer 14
6.2 Tillståndsrepresentation 16
6.3 Processvarianter 17
7 Regulatordesign 18
7.1 Överföringsfunktion 18
7.2 Tillståndsåterkoppling 21
7.3 Stationär förstärkning 21
7.4 Observerare 22
7.5 Simulering 22
7.6 Extrafunktioner 22
8 Arbetsgång och resultat 24
9 Referenser 30
Appendix A Matlab m-fil 31
Appendix B Simulink 34
Appendix C AVR c-kod 37
Appendix D sfunxy7.m 44

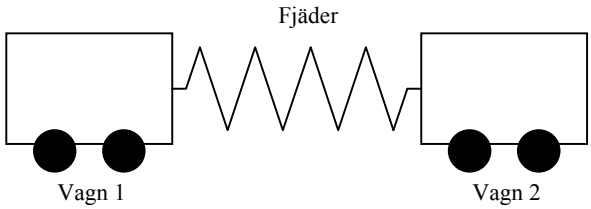
1 Inledning

1.1 Bakgrund

Kurser i reglerteknik är en obligatorisk del i utbildningen till civilingenjör inom programmen teknisk fysik (F), elektroteknik (E), datorteknik (D), maskinteknik (M), industriell ekonomi (I), teknisk matematik (Pi), informations- och kommunikationsteknik (C) och ekosystemteknik (W). Institutionen för reglerteknik på Lunds tekniska högskola inrättades 1965 och ger nu ett femtontal olika kurser i till exempel digital reglering, systemidentifiering och adaptiv reglering. Grundläggande kunskaper i reglerteknik erhålls i den allmänna kursen som alla studenter läser. Kursen innehåller bland annat moment i beräkning av processmodeller, överföringsfunktioner, stabilitetsanalys, Bode- och Nyquistdiagram, PID-regulator, tillståndsåterkoppling och Kalmanfilter. En viktig del av kursen är de tre obligatoriska laborationerna. Den första laborationen syftar till att ge studenten en känsla för hur enkla reglerkretsar uppför sig. I den andra laborationen får eleverna prova att göra enkla modeller för en reglerkrets och att ur modellen bestämma en regulator. I den tredje laborationen skall studenterna lösa ett lite svårare reglerproblem där de ser nyttan med att ha lite mer komplicerade regulatorer. Examensarbetet har gått ut på att skapa en ny process till den avslutande laborationen i denna reglertekniks allmänna kurs.

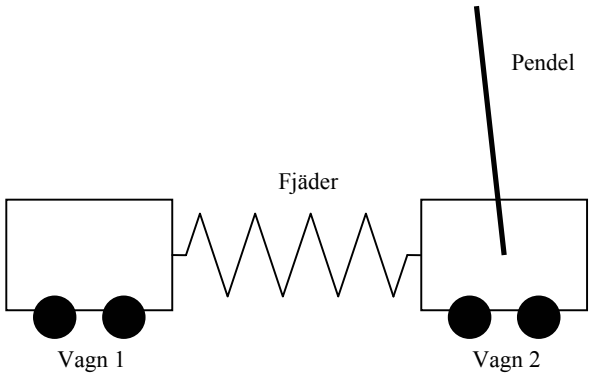
1.2 Laboration AK lab 3

Syftet med laborationen är att belysa tillståndsåterkopplingens fördelar gentemot PID-regulatorn. Den gamla processen består av två vagnar och en fjäder (figur 1.1). Vagn 1 drivs av en DC-motor och vagn 2 hänger efter i fjädern. Det är bara positionen på vagn 1 som är mätbar och regleringen går ut på att positionera vagn 2 med hjälp av tillståndsåterkoppling och observerare.



Figur 1.1 Gammal laborationsuppställning

En PID-regulator provas först och för att detta överhuvudtaget ska lyckas styrs endast vagn 1, varpå vagn 2 antas följa efter och inta rätt position i stationärt läge. Därefter implementeras en tillståndsåterkopplad regulator med observerare för positionen av vagn 2 och uppförandet förbättras avsevärt. Nackdelen med den gamla processen är att den bara kan röra sig ett par centimeter i sidled och dessutom är den, om än svår, fullt styrbar med en PID-regulator. Eftersom processen är stabil i stationäritet är det endast specifikationerna i labmanualen, bland annat om stigtid och stegsvar, som gör att en tillståndsåterkopplad regulator är att föredra. Det är därför önskvärt med en process som inte går att styra med en konventionell PID-regulator, utan kräver tillståndsåterkoppling för att överhuvudtaget gå att reglera. Den nya processen bygger till stora delar på den gamla, med den skillnaden att en inverterad pendel ska balanseras på vagn 2 och slaglängden är nu över en meter (figur 1.2).



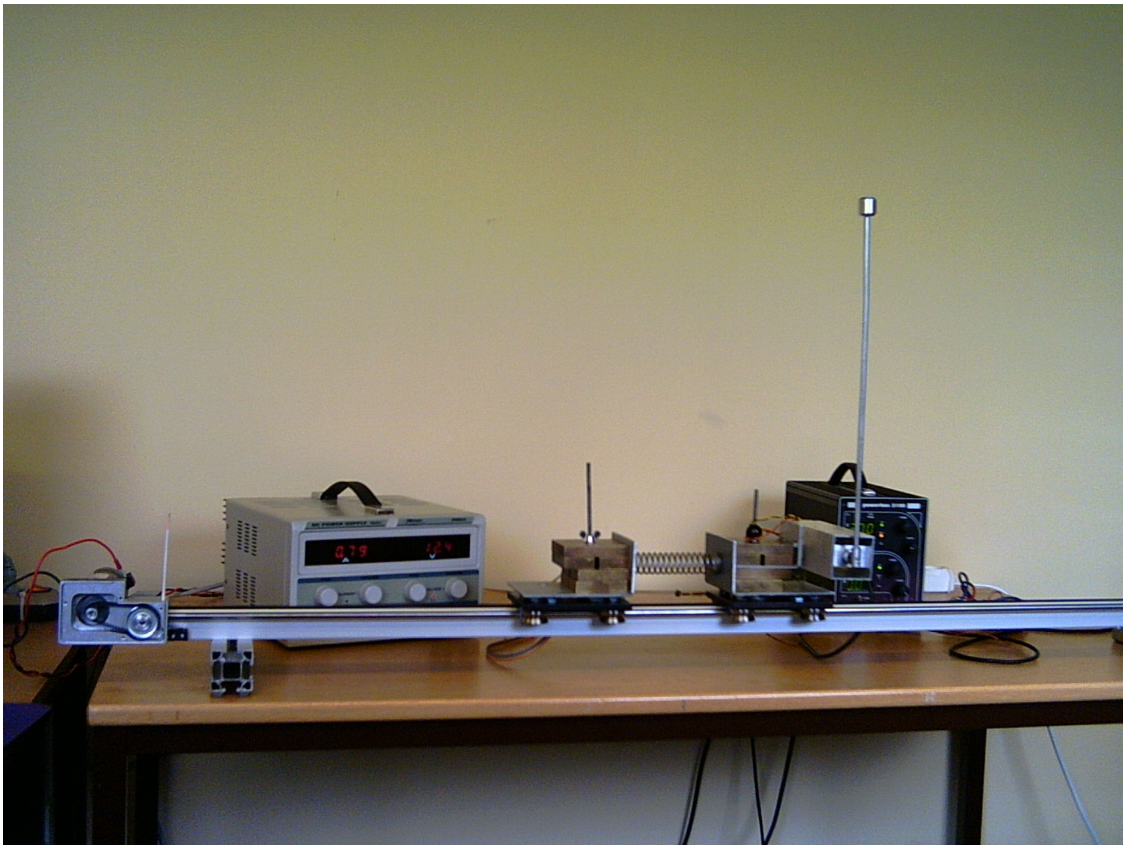
Figur 1.2 Ny laborationsuppställning

Då processen har tre utsignaler men endast en insignal, blir eventuell reglering med en PID-regulator väldigt begränsad. Två tänkbara alternativ är att antingen styrs pendeln till att balansera i sitt instabila läge (scenario 1) eller så positioneras vagnarna (scenario 2).

Scenario 1: Pendeln balanseras upprätt men vagnarna kan inte positioneras och kör omkring tills de slår i någon av laborationsuppställningens kanter.

Scenario 2: Pendeln får hänga i sitt stabila neråtläge och endast vagnarna positioneras. Detta alternativ ger ingen fördel gentemot den gamla processen.

De nya specifikationerna som omöjliggör PID-reglering består således i att positionera vagnarna med pendeln i upprätt läge. Ur lärandesynpunkt kommer syftet med den nya processen vara samma som innan. Fördelarna är att den är mycket visuellt stimulerande och ger tydliga tecken vid god reglering, samt att den till skillnad från den gamla processen är helt omöjlig att reglera med endast en PID-regulator. Detta kan ge studenterna god insikt i mer komplicerad regulatordesign och ytterligare motivation till fördjupning i fler av institutionens kurser. Vidare är processen konstruerad i moduler som tillåter byten av vagnar och pendel så att andra varianter av processer kan köras. I figur 1.3 visas den nya laborationsuppställningen.



Figur 1.3 Bild på nya laborationsuppställningen

1.3 Disposition

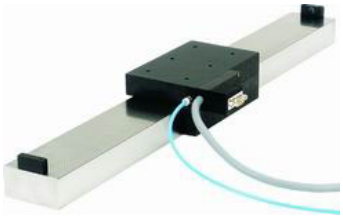
I kapitel 2 beskrivs processen och dess ingående delar. Kapitel 3 beskriver motorerna som vi använt, hur de styrs och till viss del dess begränsningar. Kapitel 4 behandlar givare och beskriver hur dessa är uppbyggda. Vi tar även upp noggrannhet och linjäritet. I kapitel 5 följer en beskrivning av datorsystemet och signalvägarna. I kapitel 6 förklaras

hur vi kommer fram till en modell av processen och hur vi går tillväga för att beskriva den på tillståndsrepresentation. Det diskuteras även vilka ändringar som behövs vid enkla ändringar av processen. I kapitel 7 går vi igenom regulatordesignen och varför vi har valt att reglera med hjälp av tillståndsåterkoppling. Kapitel 8 beskriver arbetsgång, vad som har krävt lite extra jobb och hur det har gått. Här redovisar vi alla resultat med både grafer och beskrivande text. I kapitel 9 finns en referenslista. Därefter följer Appendix A som visar Matlabkod för konstruktion av vår regulator. I Appendix B finns de väsentliga Simulinkblocken. I Appendix C finns C-koden till AVRen och slutligen i Appendix D finns s-funktionen för att grafiskt visa simuleringen av processen.

2 Processen

2.1 Linjärenhet

Från början var tanken att använda en linjärmotor (figur 2.1).



Figur 2.1 Linjärmotor

Detta hade varit den enklaste lösningen eftersom motorn utgörs av löparen som rör sig längs statorelementet. Med en extra dummy-löpare hade laborationsuppställningen i princip varit klar. Linjärmotorn har inga överföringsförluster på grund av mekaniskt glapp i transmissioner eller friktion eftersom löparen glider på ett luftlager. Till detta krävs en pålitlig tryckluftskompressor med hög kapacitet. Ett bortfall av luftlagringen under drift leder till haveri. Linjärmotorn är mycket stark och precis med hastigheter upp till 2 m/s vilket kräver stora försiktighetsåtgärder. Kringutrustning och inte minst motorn hade krävt stora investeringar. Därför bokades ett besök på Solectro [1] i Lomma för att se vad annat som fanns att erbjuda på marknaden. De billigare alternativen i form av lineärgejdrar med kulmutterskruv alternativt remdrift demonstrerades,
(figur 2.2).

Figur 2.2 Till vänster en kulmutterskruv och till höger två remdrifter

En linjärgejd med remdrift visade sig vara det bästa alternativet med tanke på möjligheterna att bygga maskinen i moduler och framförallt var priset rätt.
En 1500 mm lång kuggremsdrift modell ZF 1 med tillhörande lastvagnar och en stegmotor modell MS160 beställdes.

2.2 Vagnar

Vagnarna spänns in glappfritt mot skenan med hjälp av excenterskruvar i hjulen som sitter fast med livstidssmorda kullager. Valfritt antal masselement kan monteras ovanpå vardera vagns lastflak genom att massorna hängs över gängstången och sedan spänns fast med vingmutter. Vid experiment med bara en vagn kopplas vagn 2 loss och flyttas till änden av gejden. På vagn 1 sitter en optisk inkrementell linjärgivare som ger positionen på vagn 1 relativt gejden. En annan linjärgivare sitter mellan vagn 1 och vagn 2 och ger avståndet mellan dessa. Mer om dessa följer i avsnittet om givare.

2.3 Pendel

Pendeln är också byggd som en modul och är tänkt att fästas på valfri vagn. Den bygger på en envarvig vridpotentiometer med extremt låg friktion och sitter fast i ett litet pendelhus som i sin tur monteras på en vagn. På den utstickande axeltappen sitter sedan pendelarmen med en liten vikt längst ut. Vikten går enkelt att byta och flyttas upp eller ner längs pendelarmen.

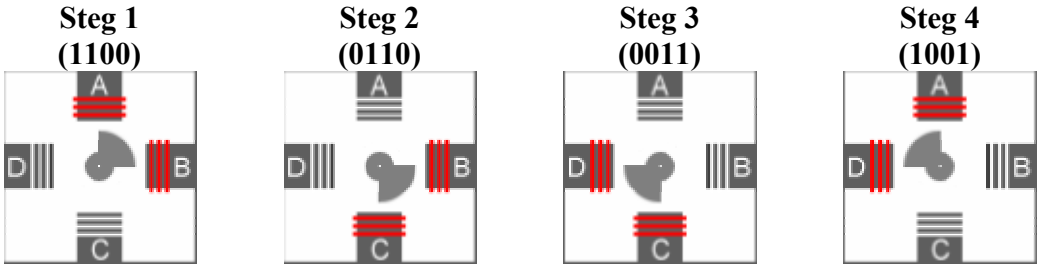
2.4 Kravspecifikation

Det övergripande målet med regleringen är att pendeln ska balansera i upprätt läge under tiden som vagnarna förflyttas till önskad position på gejden. Någon specifikation för regleringen är inte fastställd än, men ett rimligt krav är att vagnarna vid stationär reglering inte avviker från börvärdet med mer än 0.10 m.

3 Motorer

3.1 Stegmotor

En vanlig likströmsmaskin har lindningar som automatiskt kopplas in och ur av borst som ligger an mot kommutatorn. När motorn är inkopplad kör den själv med en hastighet som är proportionell mot spänningen och lasten den driver. En stegmotor har däremot ingen kommutator. Istället kommer det fem eller sex kablar ut ur motorn, en för varje lindning (vanligen fyra), och en eller två jordsladdar. Lindningarna måste kopplas in en efter en i rätt sekvens [2] för att få motorn att snurra. För att få maximalt vridmoment kopplas lindningarna oftast in parvis (figur 3.1).

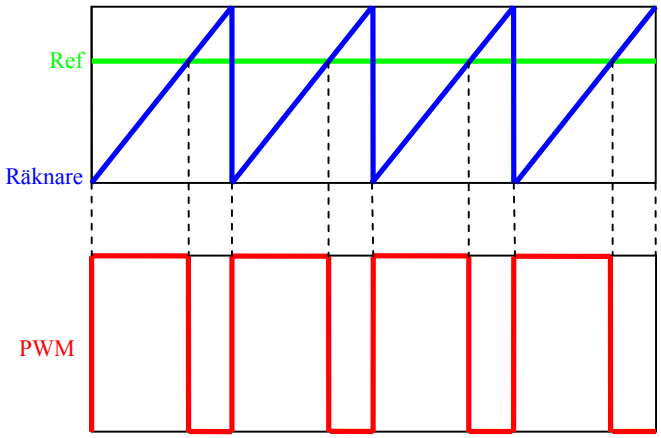


Figur 3.1 Stegmotordrift

Varje steg vrider motoraxeln en bråkdel av ett varv (en grad eller två). Cykeln om fyra steg måste upprepas ungefär 50 gånger för ett fullt varv (inte bara en som visas i figuren). Om alla fyra lindningarna är urkopplade är motorn frikopplad. Annars är den alltid låst i sin aktuella position. Om lasten på en stegmotor blir för stor eller om stegsekvensen klockas för snabbt kommer motorn att hoppa över steg. Efter lite testkörningar kom vi fram till att stegmotorn inte skulle räcka till och blev därför tvingade att byta till en likströmsmotor.

3.2 Likströmsmotor

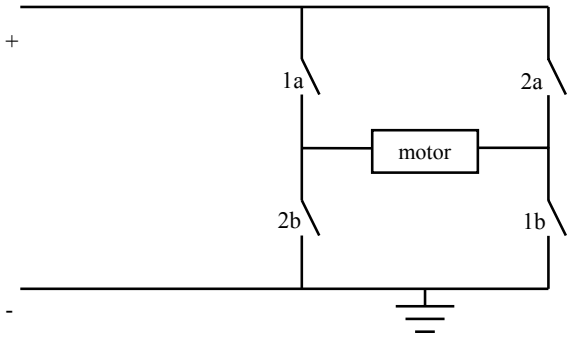
För att få likströmsmotorn att gå med önskad hastighet måste drivspänningen läggas ut i lagom portioner. Detta sker genom en sorts D/A-omvandling som kallas PWM [3] (Pulse Width Modulation) (figur 3.2).



Figur 3.2 Pulsbreddsmodulering

Drivningen delas in i perioder som motsvarar ner till tusendelar av en sekund. Full spänning läggs ut i pulser med en längd som motsvarar delen av maxspänningen som eftersträvas. Resten av perioden matas motorn ej. En räknare startas varje ny period och jämförs med referensvärdet. När de två är lika kopplas motorn ur och väntar på att räknaren ska nå maxvärdet och en ny period startar. På så sätt får man ett medelvärde som motsvarar referensvärdet. Eftersom pulserna är täta och motorn har induktiva element som fungerar som lågpassfilter blir hastigheten jämn.

Stegmotorn är enkel att reversera när väl styrelektroniken fungerar. Det är bara att ändra riktning på stegcykeln. Likströmsmaskinen behöver ingen avancerad drivlogik för att snurra, men väl en krets som vänder polariteten på spänningen för att driva motorn åt andra hållet. H-bryggan är den vanligaste drivkretsen för att klara av detta (figur 3.3).



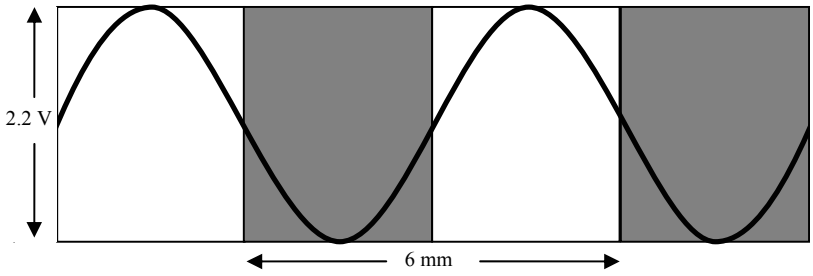
Figur 3.3 H-brygga

Som synes beror motorns riktning på hur strömbrytarna i bryggan ställs. 1a tillsammans med 1b ger positiv drivspänning och driver motorn framåt. 2a och 2b ger negativ drivspänning och driver motorn bakåt.

4 Givare

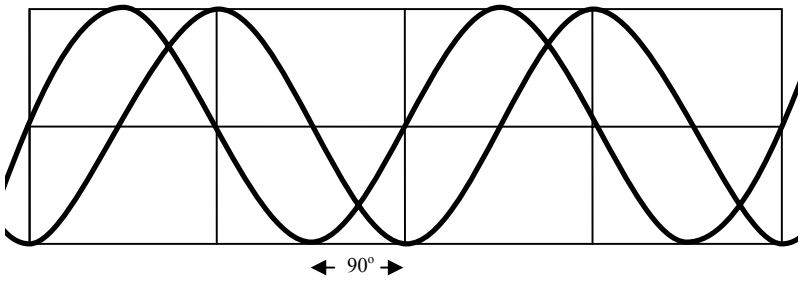
4.1 Linjärgivare

För att bestämma positionen på vagnarna finns det ett antal alternativ. De som har varit aktuella är en avkodare direkt på motoraxeln eller någon form av linjärgivare. Vid användning av stegmotor kan dessutom en pulsräknare enkelt implementeras. Nackdelen med en pulsräknare i kombination med stegmotor är att man inte kan kontrollera att steget verkligen utfördes. Det är därför både lätt och troligt att det givna läget avviker från det faktiska. Det går dock att minimera risken för detta genom att begränsa hastigheten på stegmotorn vilket givetvis inte är önskvärt. Dessutom är det ganska bra om det går att byta motor utan att behöva tänka på hur positionerna ska återges. Istället beslutades att linjärgivare skulle användas. Det är en ganska enkel form av linjärgivare som består av ett svart-vitt randigt papper samt två läshuvud. Vart läshuvud ger en sinusliknande utsignal mellan 0,3 och 2,5 V (figur 4.1).



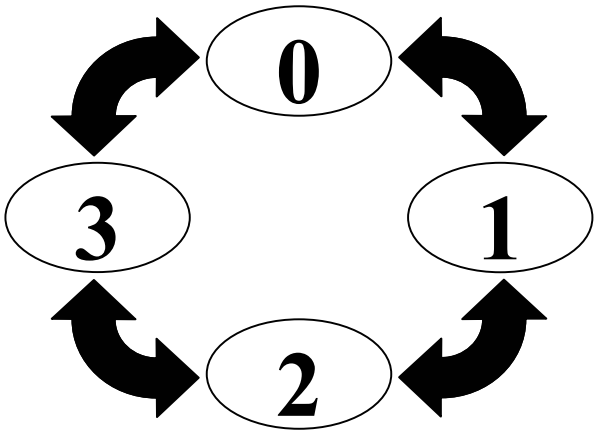
Figur 4.1 Analogsignal från läshuvud

Ränderna är 3 mm breda och läshuvudena är fästa med 1,5 mm inbördes avstånd vilket ger två sinusformiga vågor varav en är fasförskjuten med 90 grader (figur 4.2).



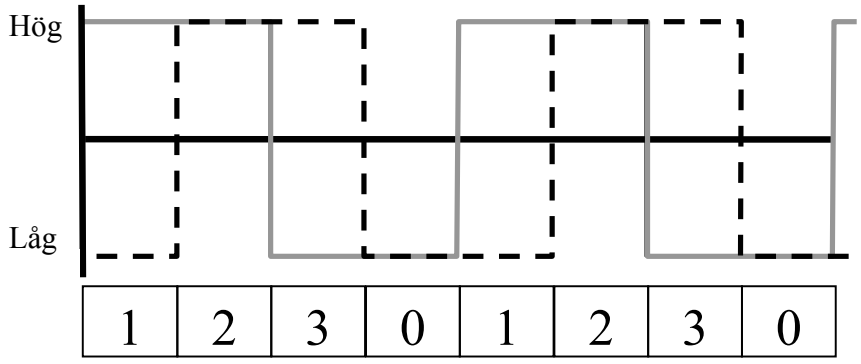
Figur 4.2 Utsignal från linjärgivare

Genom att diskretisera utsignalen får vi 4 tillstånd (figur 4.3) och kan då enkelt bestämma riktningen genom att se på nuvarande och föregående tillstånd. Detta görs genom att ta differensen mellan tillstånden där +1 eller -3 (från 3 till 0) är medurs och motsvarar en rörelse i positiv riktning och där -1 eller +3 (från 0 till 3) är moturs och motsvarar alltså en rörelse i negativ riktning. Är differensen noll är inte förflyttelsen tillräcklig för att tillståndsmaskinen ska byta tillstånd. Här är det viktigt att använda tillräckligt snabb sampling så att skillnaden inte blir ±2 då det skulle vara omöjligt att säga något om vilken riktning vagnen färdas samt vilken position den har.



Figur 4.3 Tillståndsmaskin

Nedan visas ett exempel av den diskreta signalen med respektive tillstånd då vagnen färdas i positiv riktning (figur 4.4).



Figur 4.4 Diskretiserad utsignal samt tillhörande tillstånd

Linjärgivaren matas med en spänning på 5 V och ger en utspänning på ca 2,5 V när läshuvudet befinner sig rakt ovanför vitt fält och ca 0.3 volt vid svart fält. Maxspänningen är lätt att justera med hjälp av två potentiometrer. A/D-omvandlaren ger ett värde enligt

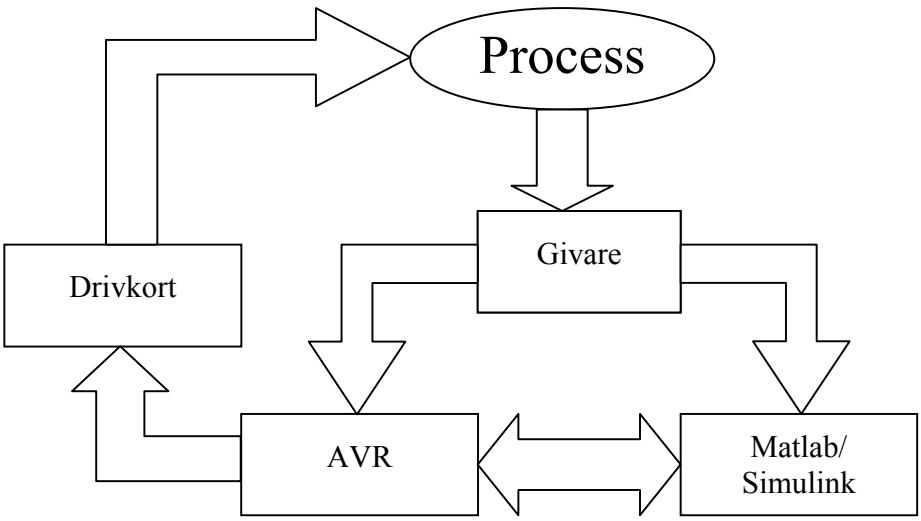
följande formel: $V_{ut} = \frac{V_{in} \cdot 1024}{V_{ref}}$. 1024 kommer från att omvandlaren har 10 bitar vilket är mer än tillräckligt för vår applikation. V_{ref} går att ställa in till 5 V, 2,56 V eller valfri extern spänning. Vi har valt 2,56 V för att matcha utsignalen på bästa sätt och därmed inte förlora någon upplösning i onödan. Med hjälp av det som hittills har beskrivits fås en noggrannhet på 1,5 mm. För att höja noggrannheten ytterliggare, tittar vi även på den analoga signalen. Då fås en uppskattning på var i ett tillstånd vagnen befinner sig, Δ. Den sinusliknande vågen linjäriseras och med hjälp av den räta linjens ekvation $y = kx + m$ fås i vårt fall $\Delta = \frac{V_{in} - m}{k}$. Värdet på k och m beror på tillståndet. Detta gör att noggrannheten blir ett par tiondels millimetrar.

4.2 Vinkelgivare

För att det ska vara möjligt att styra pendeln är det nödvändigt att veta dess vinkel. Därför är pendeln monterad på en vridpotentiometer. Pendeln kan snurra kontinuerligt, men potentiometern har ett dödläge på ungefär 15 grader i skarven mellan dess maximala och minimala resistans. Givaren får samma dödläge och därför är det viktigt att placera skarven på ett ställe där det stör reglering och andra funktioner på ett minimalt sätt, vilket den gör i horisontellt läge. I övrigt är vinkeln mellan pendelarmen och pendelhuset linjärt proportionell mot resistansen i potentiometern. För att få en utsignal matas potentiometern från ett drivkort med 30 V. Värdet på absolutvinkeln representeras av en signal mellan 0 och 10 V som sänds till Simulink, där den omräknas för att ge en vinkel på $\pm\pi$.

5 Datorsystem

I den gamla laborationen används Simulink på en Linuxdator för att styra och ändra på regleringen. Detta är ett grafiskt gränssnitt som är lätt att förstå och gör det väldigt enkelt att ändra parametrar i en vald regulator. Därför är det önskvärt att gränssnittet behålls och kringutrustningen får anpassas till detta istället för tvärtom.
Matlab kan inte kommunicera med högre hastighet än 300Hz på serieporten. Framför allt linjärgivarna behöver betydligt högre hastighet än så för att fungera ordentligt och inte tappa positioner. Detta innebär att en extern microprocessor behövs för uppgiften. Till detta används en Atmel ATMega 8 som har inbyggd A/D-omvandlare. På samma kort som AVRen sitter det en H-brygga som används till styrningen av DC-motorn. Se figur 5.1 för blockdiagram.



Figur 5.1 Blockdiagram av datorsystem

Det finns fyra funktioner för AVRen:

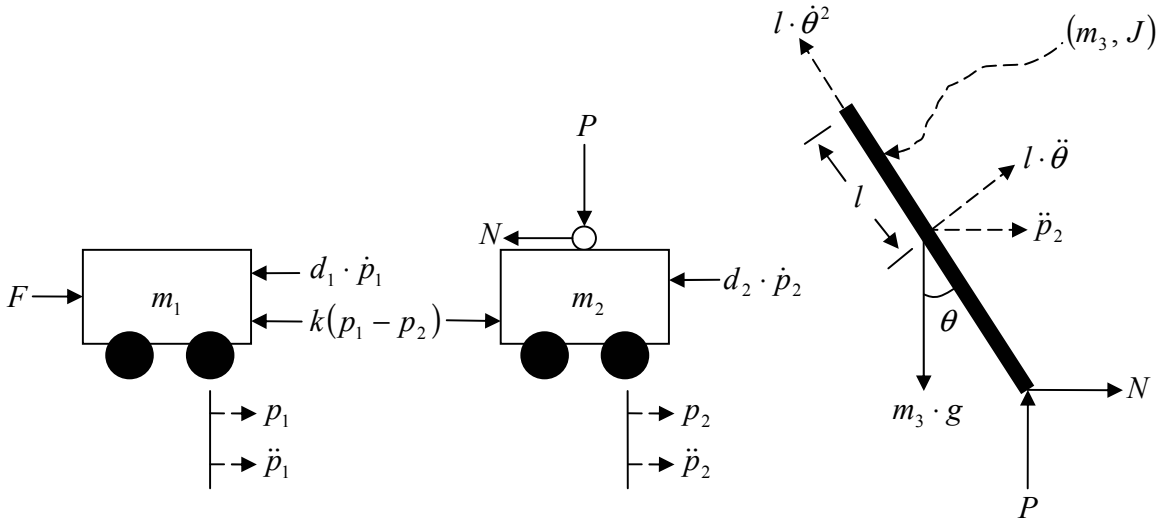
- Huvudprogram som initierar och definierar olika värden.
- Avbrottsstyrd A/D-omvandling av linjärgivarnas värde som uppdaterar vagnarnas position.
- Interpolation som höjer noggrannheten av vagnarnas position.
- Avbrottsstyrd pulsbreddsmodulering av utsignalen som styr motorerna.

AVRen får ett referensvärde från Simulink som används till pulsbreddsmodulationen för att få önskad utsignal. Interpolationen görs endast i samband med att positionerna skickas till Simulink för att inte belasta processorn i onödan.

6 Modell

6.1 Kraftanalys och systemekvationer

En bra början till att förstå hur ett system fungerar är att göra en friläggning så att alla påverkande krafter enkelt kan urskiljas. Lämpliga snitt att frilägga är sådana som har en tydlig utsignal som går att återkoppla. I det här fallet är det tydligt att det blir vagnarna var för sig och pendeln som utgör frikropparna eftersom det är deras positioner som kommer att finnas tillgängliga. I figur 6.1 finns det frilagda systemet.



	Benämning	Uppmätt värde	Enhet
F	motorkraft	-	N
p_1	position vagn 1	-	m
m_1	massa vagn 1	3,075	kg
d_1	dämpning vagn 1	40	N/(m*s)
k	fjäderkonstant	687,44	N/m
p_2	position vagn 2	-	m
m_2	massa vagn 2	3,455	kg
d_2	dämpning vagn 2	5	N/(m*s)
P	vertikal infästningskraft	-	N
N	horisontell infästningskraft	-	N
Θ	pendelns vinkel	-	rad
m_3	pendelns massa	0,04	kg
l	pendelns halva längd	0,25	m
J	pendelns tröghetsmoment	0,0009	kg*m^2
g	gravitationskonstant	9,82	m/s^2

Figur 6.1 Frilagt system.

Tidsderivator av positionerna (hastigheter och accelerationer) betecknas med prickar ovanför variablerna. Summering av horisontella krafter ger följande tre ekvationer:

$$\left(\overrightarrow{Massa_1}\right): \quad m_1 \cdot \ddot{p}_1 = -k \cdot (p_1 - p_2) - d_1 \cdot \dot{p}_1 + F \tag{1}$$

$$\left(\overrightarrow{Massa_2}\right): \quad m_2 \cdot \ddot{p}_2 = k \cdot (p_1 - p_2) - d_2 \cdot \dot{p}_2 - N \tag{2}$$

$$\left(\overrightarrow{Pendel}\right): \quad m_3 \cdot \ddot{p}_2 = m_3 \cdot l \cdot \dot{\theta}^2 \cdot \sin(\theta) - m_3 \cdot l \cdot \ddot{\theta} \cdot \cos(\theta) + N \tag{3}$$

För att eliminera den okända kraften *N* kombineras ekvation (2) och (3).

$$((2)+(3)) \quad \Rightarrow \quad (m_2 + m_3) \cdot \ddot{p}_2 + d_2 \cdot \dot{p}_2 - k \cdot (p_1 - p_2) = -m_3 \cdot l \cdot \left(\ddot{\theta} \cdot \cos(\theta) - \dot{\theta}^2 \cdot \sin(\theta)\right)$$

Det går utmärkt att ställa upp ekvationer även för vertikala krafter men de ger ingen ytterligare information.
Om de mot pendeln vinkelräta krafterna summeras fås en tredje rörelseekvation:

$$(\perp Pendel): \quad P \cdot \sin(\theta) + N \cdot \cos(\theta) - m_3 \cdot g \cdot \sin(\theta) = m_3 \cdot l \cdot \ddot{\theta} + m_3 \cdot \ddot{p}_2 \cdot \cos(\theta) \tag{4}$$

För att få bort *P*- och *N*-termerna i ekvation (4) sammanräknas momenten runt pendelns tyngdpunkt:

$$(\circlearrowleft Pendel): \quad -P \cdot l \cdot \sin(\theta) + N \cdot l \cdot \cos(\theta) = J \cdot \ddot{\theta} \tag{5}$$

Sedan kombineras ekvationerna enligt följande:

$$((4)+(5)) \quad \Rightarrow \quad \left(J + m_3 \cdot l^2\right) \cdot \ddot{\theta} + m_3 \cdot g \cdot l \cdot \sin(\theta) = -m_3 \cdot l \cdot \ddot{p}_2 \cdot \cos(\theta)$$

Kraften *F* kommer från en DC-motor och en enkel modell kan ställas upp för att ta hänsyn till motorns dynamik:

$$(Motor): \quad F = \beta \cdot u - \alpha \cdot \dot{p}_1 \tag{6}$$

Detta innebär att styrsignalen *u* införs i systemet och ekvation (1) ändras till:

$$((1)+(6)) \quad \Rightarrow \quad m_1 \cdot \ddot{p}_1 = -k \cdot (p_1 - p_2) - \dot{p}_1 \cdot (d_1 + \alpha) + \beta \cdot u$$

Sammanfattningsvis en översikt av de tre nödvändiga dynamiska systemekvationerna:

$$m_1 \cdot \ddot{p}_1 = -k \cdot (p_1 - p_2) - \dot{p}_1 \cdot (d_1 + \alpha) + \beta \cdot u \tag{7}$$

$$(m_2 + m_3) \cdot \ddot{p}_2 + d_2 \cdot \dot{p}_2 - k \cdot (p_1 - p_2) = -m_3 \cdot l \cdot \left(\ddot{\theta} \cdot \cos(\theta) - \dot{\theta}^2 \sin(\theta)\right) \tag{8}$$

$$\left(J + m_3 \cdot l^2\right) \cdot \ddot{\theta} + m_3 \cdot g \cdot l \cdot \sin(\theta) = -m_3 \cdot l \cdot \ddot{p}_2 \cdot \cos(\theta) \tag{9}$$

6.2 Tillståndsrepresentation

För att konstruera en vettig regulator linjäriseras modellen runt det instabila upprätta jämviktsläget $\theta = \pi$. Då blir $\cos(\theta) \approx -1, \quad \sin(\theta) \approx -\theta$ och $\dot{\theta}^2 \approx 0$. Ekvation (8) och (9) ändras enligt följande:

$$(8) \quad \Rightarrow \quad (m_2 + m_3) \cdot \ddot{p}_2 + d_2 \cdot \dot{p}_2 - k \cdot (p_1 - p_2) = m_3 \cdot l \cdot \ddot{\theta} \tag{10}$$

$$(9) \quad \Rightarrow \quad \left(J + m_3 \cdot l^2\right) \cdot \ddot{\theta} - m_3 \cdot g \cdot l \cdot \theta = m_3 \cdot l \cdot \ddot{p}_2 \tag{11}$$

För att lösa ut accelerationerna kombineras ekvationerna och det linjäriserade systemet kan till slut beskrivas som:

$$(7) \quad \Rightarrow \quad \left\{ \ddot{p}_1 = \frac{-k \cdot (p_1 - p_2) - \dot{p}_1 \cdot (d_1 + \alpha) + \beta \cdot u}{m_1} \right.$$

$$((10)+(11)) \quad \Rightarrow \quad \begin{cases} \ddot{p}_2 = \dfrac{\left(J + m_3 \cdot l^2\right) \cdot \left(k(p_1 - p_2) - d_2 \cdot \dot{p}_2\right) + (m_3 \cdot l)^2 \cdot g \cdot \theta}{J \cdot (m_2 + m_3) + m_2 \cdot m_3 \cdot l^2} \\ \ddot{\theta} = \dfrac{(m_3 \cdot l) \cdot (k \cdot (p_1 - p_2) - d_2 \cdot \dot{p}_2) + m_3 \cdot l \cdot g \cdot (m_2 + m_3) \cdot \theta}{J \cdot (m_2 + m_3) + m_2 \cdot m_3 \cdot l^2} \end{cases}$$

En tillståndsrepresentation med tillståndsvektorn $x = \begin{bmatrix} p_1 & \dot{p}_1 & p_2 & \dot{p}_2 & \theta & \dot{\theta} \end{bmatrix}^T$ och utsignalen $y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^T$ ges av:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t)$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{k}{m_1} & -\frac{d_1+\alpha}{m_1} & \frac{k}{m_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{a \cdot k}{c} & 0 & -\frac{a \cdot k}{c} & -\frac{a \cdot d_2}{c} & \frac{b^2 \cdot g}{c} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{b \cdot k}{c} & 0 & -\frac{b \cdot k}{c} & -\frac{b \cdot d_2}{c} & \frac{b \cdot g \cdot (m_2+m_3)}{c} & 0 \end{bmatrix} \cdot x(t) + \begin{bmatrix} 0 \\ \frac{\beta}{m_1} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cdot u(t)$$

där $a = J + m_3 \cdot l^2$, $b = m_3 \cdot l$ och $c = J \cdot (m_2 + m_3) + m_2 \cdot m_3 \cdot l^2$.

$$y(t) = C \cdot x(t) = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \cdot x(t) = \begin{bmatrix} k_{y1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{y2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{y3} & 0 \end{bmatrix} \cdot x(t)$$

6.3 Processvarianter

Genom att sätta pendeln på vagn 1 istället och låta vagn 2 fungera som störkälla, eller bara använda en vagn med pendel kan flera olika processvarianter byggas.
Ingen modellering av andra varianter har undersökts då de görs enligt samma koncept som ovan. Nyckeln till framgång med metoden ligger i att först frilägga systemet korrekt och därefter identifiera inverkande krafter. Sedan löses accelerationerna ut explicit och kraftekvationerna innehåller då elementen i systemmatrisen A. Som exempel kan nämnas att ett minustecken framför sista raden i den modellerade A-matrisen gör att processen ändras från att balansera pendeln upprätt till att dämpa den nedåt. Detta kommer att tillämpas senare i avsnittet om extrafunktioner.

7 Regulatordesign

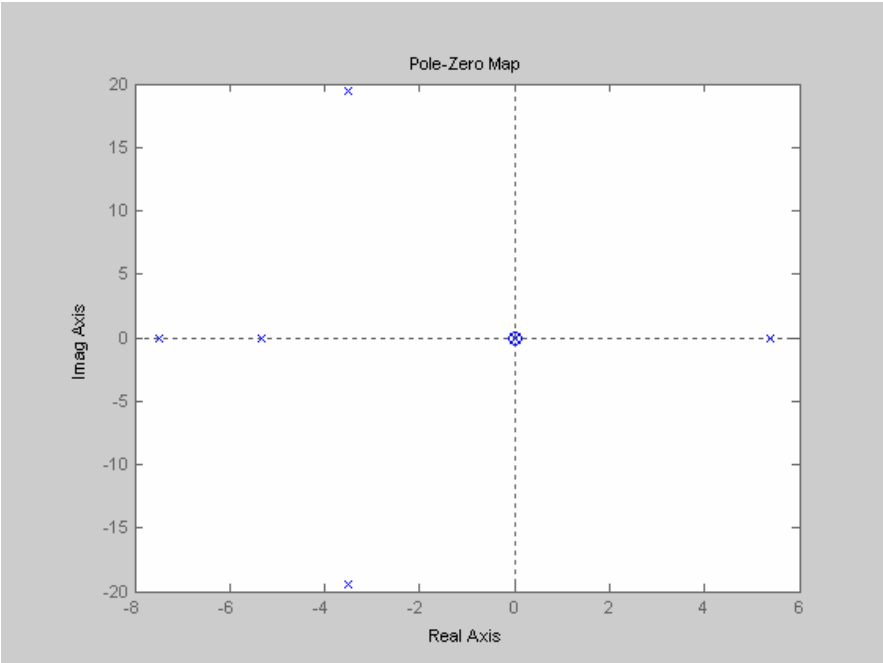
Målet med regleringen är att kontrollera de båda vagnarnas positioner och pendelns vinkel. Ett naturligt första val av regulator är en PID-regulator. En PID regulator hade visserligen varit tillräcklig för att hålla pendeln upprätt, men då hade inte vagnarna kunnat positioneras. Det finns tekniker för att låta PID-regulatorer jobba ihop och på så sätt klara flera utsignaler, till exempel genom kaskadkoppling. Det bästa valet är dock att tillståndsåterkoppla processen, speciellt när det finns god kunskap om fysiken bakom systemets rörelseekvationer. Till detta har Matlab och Simulink använts.

7.1 Överföringsfunktion

För att få en överskådlig blick av processens beteende är det lämpligt att ta fram en överföringsfunktion. Det finns en känd formel för att göra detta ur tillståndsrepresentationen, nämligen $C \cdot (sI - A)^{-1} \cdot B$. Nedan visas överföringsfunktionen från insignal till pendelns position med de värden som finns i figur 6.1.

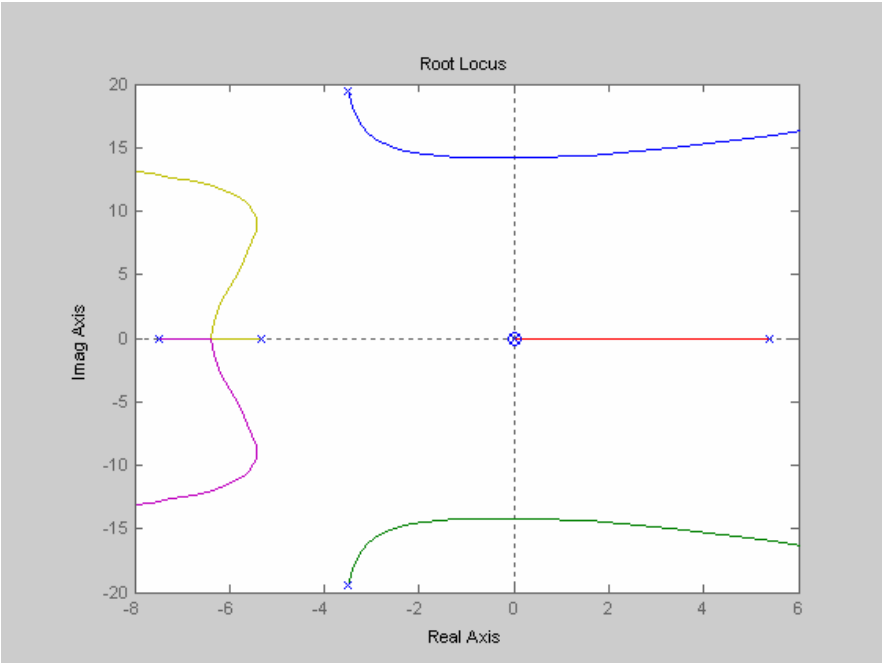
$$G(s) = \frac{189{,}7 \cdot s^2}{s^6 + 14{,}45 \cdot s^5 + 411{,}6 \cdot s^4 + 2483 \cdot s^3 - 12770 \cdot s^2 - 83760 \cdot s}$$

Överföringsfunktionen är av sjätte graden och systemets sex poler definieras av A-matrisens egenvärden, som är rötterna till överföringsfunktionens nämnare. Genom att använda Matlabs kommando *pzmap*, fås en överblick var poler och nollställe är placerade, se figur 7.1. Här finns 6 poler, en i ostabilt område, en i origo, ytterliggare två på den reella axeln samt 2 konjugerade. Det finns även ett dubbelt nollställe i origo.



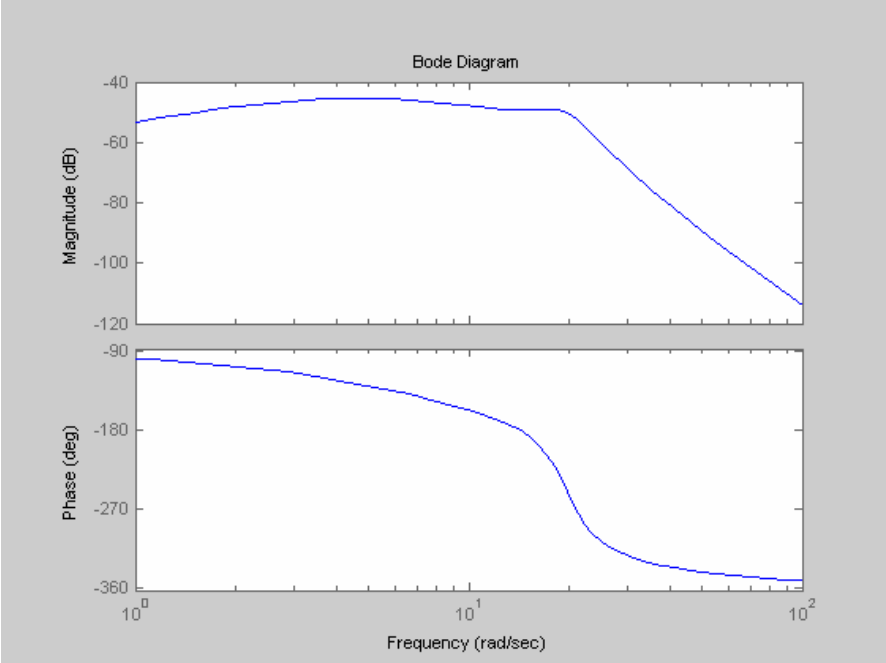
Figur 7.1 Pol- nollställe-diagram

Polen i höger halvplan tillhör pendelns upprätta jämviktsläge och är processens svåraste del att reglera. Genom att använda kommandot *rlocus* kan man se rotorten, figur 7.2.



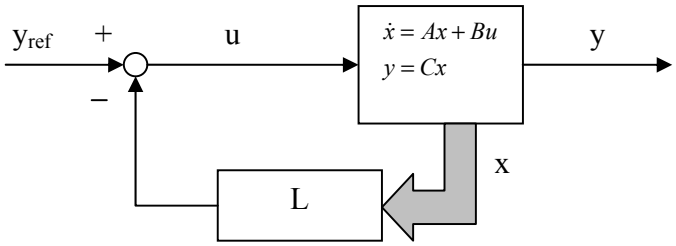
Figur 7.2 Rotort

Här syns vad som händer när återkopplingsförstärkningen ökas. För hög förstärkning kommer att leda ut ytterligare två nollställen på oscillerande och instabilt område. I figur 7.3 finns bodediagrammet för processen.



Figur 7.3 Bodediagram

7.2 Tillståndsåterkoppling



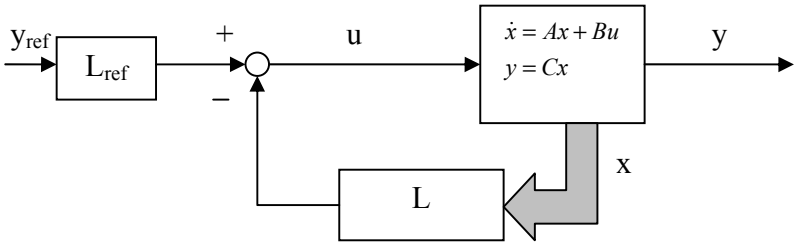
Figur 7.4 Tillståndsåterkoppling

För att göra regleringen hanterbar återkopplas alla sex tillstånd. Detta tillsammans med att systemet är styrbart (styrbarhetsmatrisen $[B \;\; AB \;\; A^2B \;\; A^3B \;\; A^4B \;\; A^5B]$ har full rang) [4] gör att det återkopplade systemets poler kan placeras godtyckligt. Det gäller att hitta vektorn L som bestämmer styrlagen för återkopplingen (figur 7.4). Detta kan göras på flera sätt. Om de önskade polerna för det slutna systemet är kända kan funktionen *place* användas. En annan möjlighet är att använda funktionen *lqr* (linear quadratic regulator), som ger en optimal regulator. Funktionen utgår från två parametrar, R och Q, som bestämmer straffet på styrsignalen och den relativa prioriteten mellan tillstånden. Det enklaste är att anta att R=1 och $Q=C^T*C$. Metoden tillåter alltså styrning av alla positioner och är ganska enkel att använda eftersom regulatorn ställs in till önskat uppförande genom att ändra elementen i Q-matrisen.

7.3 Stationär förstärkning

Det stationära felet kan elimineras genom att räkna ut den stationära förstärkningen av tillstånden och sedan multiplicera referensvärdet med dess invers Lref. (figur 7.5)

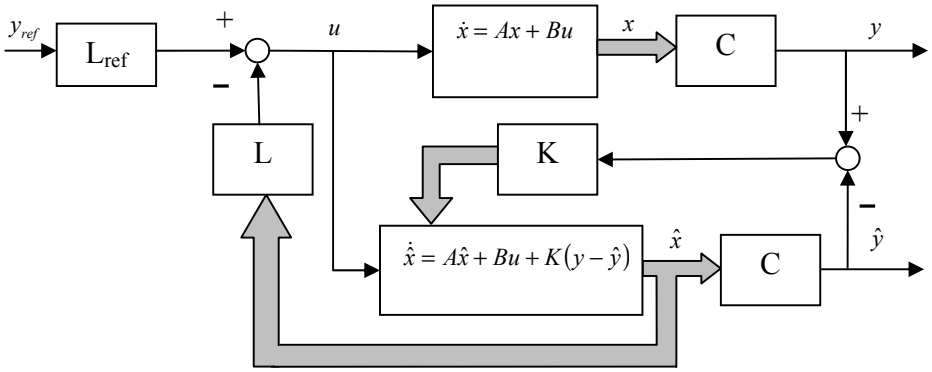
$$Lref = \frac{1}{C \cdot ((-A + B \cdot L)/B)}.$$



Figur 7.5 Tillståndsåterkoppling med kompensering för stationärt fel

7.4 Observerare

Eftersom bara tre tillstånd är mätbara från en direkt utsignal används en observerare för att beräkna övriga. Det krävs då att observerbarhetsmatrisen [$C\ \ CA\ \ CA^2\ \ CA^3\ \ CA^4\ \ CA^5$] har full rang [4]. Observeraren bygger på principen att en modell av processen får samma insignal som den verkliga processen. Sedan jämförs utsignalerna från modellen och processen där skillnaden används för att uppdatera modellens tillstånd (figur 7.6).



Figur 7.6 Tillståndsåterkoppling med observerare och kompensering för stationärt fel

7.5 Simulering

Efter modellens och regulatorns färdigställande är det lämpligt att testa dessa genom simulering, t.ex. med hjälp av Simulink. Det görs antingen då den verkliga processen saknas eller för att testa olika reglerstrategier utan risk att skada hårdvaran. Man ska komma ihåg att det är enklare att få en simulering att fungera än motsvarande hårdvara, eftersom det är svårt att modellera alla störningar och andra felkällor som finns i verkligheten. Däremot kan det generella uppförandet observeras och olika regulatorer jämföras med varandra. Som regel är det ett krav att simuleringarna ska fungera för att det ska lyckas i praktiken.

7.6 Extrafunktioner

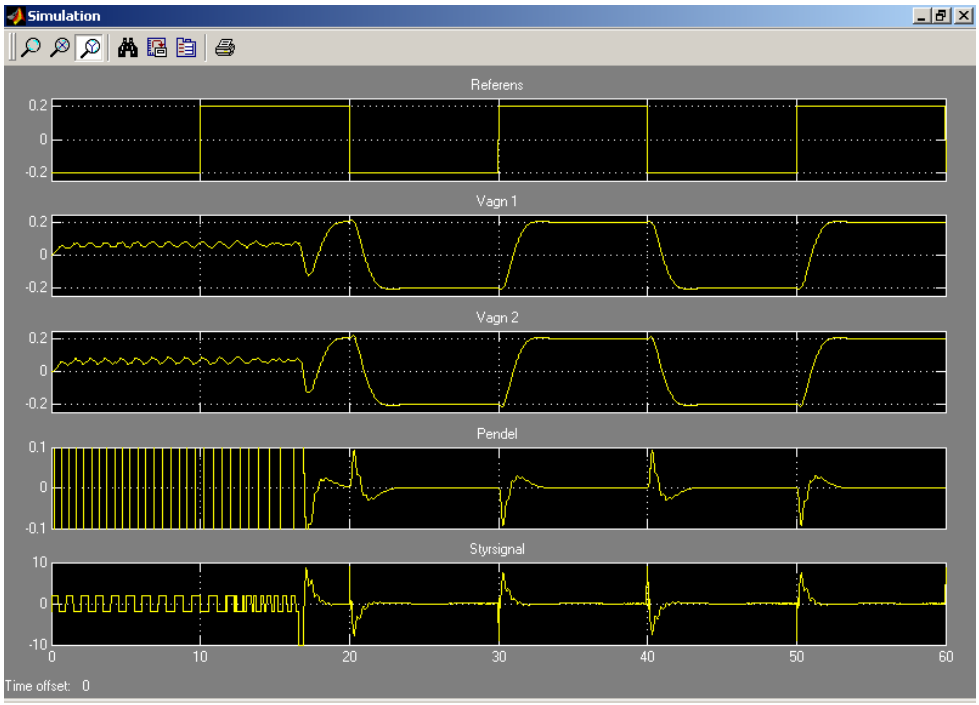
1. För att kunna styra processen helt och hållet från datorn är det lämpligt att ha en funktion som svingar upp pendeln till upprätt läge för att där byta till den regulator som ska stabilisera processen. I princip går det ut på att med så små rörelser som möjligt få maximal hastighet på pendeln. För att hastigheten inte ska bli för hög, så att pendeln inte går att fånga av regulatorn, styrs den med en begränsad signal runt det stabila

jämviktsläget (hängande neråt). När sedan pendeln passerar jämviktsläget ges den fart genom att styra vagnen i motsatt riktning mot vad pendeln kom från. På detta sätt gungas den fram och tillbaka och ges extra fart var gång den passerar nolläget tills att den är så nära det upprätta läget att regulatorn kan ta över.

2. En annan funktion när man använder en pendel är att inte låta den gunga obehindrat. Till det kan man använda en dämpare som hela tiden försöker få pendeln att hänga rakt ner. Detta går att åstadkomma på ett par sätt, en är att på ren känsla styra pendeln tvärtom mot vad tidigare beskrevs vid uppsvingsrörelsen, men då utnyttjas inte systemets dynamik överhuvudtaget. En bättre variant är att göra en modell även för detta läge. Skillnaden mellan en sådan modell och en modell som ska balansera pendeln uppåt är marginell. Det är endast tecknet på sista raden i A-matrisen som ska bytas. Detta eftersom $\cos(\theta)$ respektive $\sin(\theta)$ nu blir 1 respektive θ vid linjärisering runt det stabila jämviktsläget.

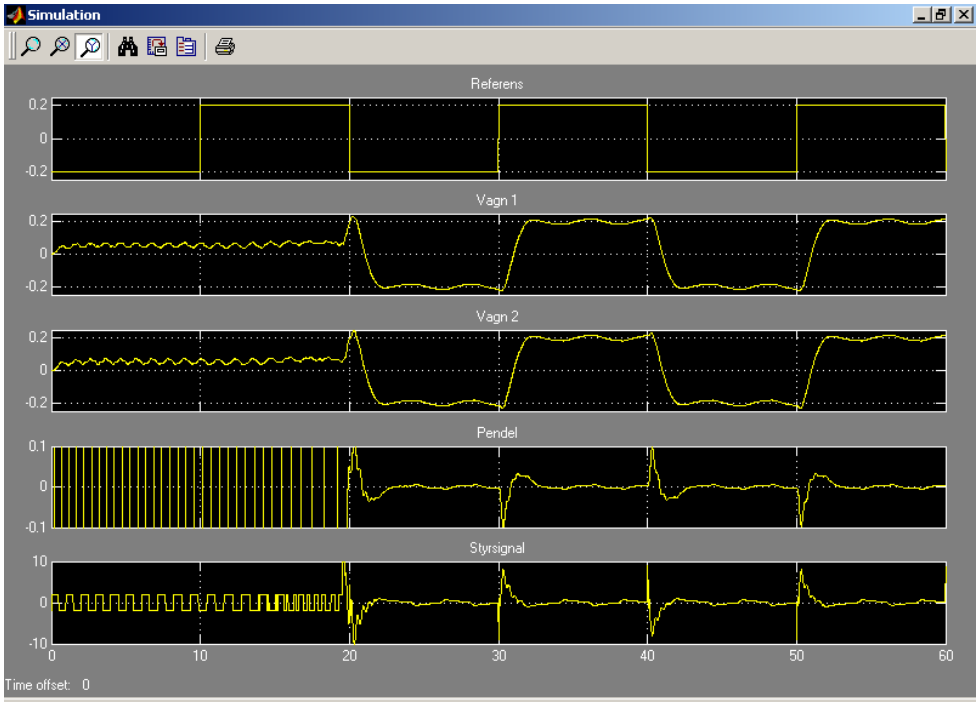
8 Arbetsgång och resultat

Under projektets gång har hela tiden förbättringar observerats och målet närmat sig med jämna steg. De första veckorna ägnades åt att hitta lämpligt material, färdigställa en modell och designa en regulator. Modellen och regulatorn testades sedan i Simulink och det var inga problem att få modellen att fungera enligt önskemål, då ingen hänsyn till friktion eller begränsningar i motorn togs (se figur 8.1). I början av simuleringen används funktionen för att svinga upp pendeln och efter ungefär 16 sekunder tar den stabiliserande regulatorn över.



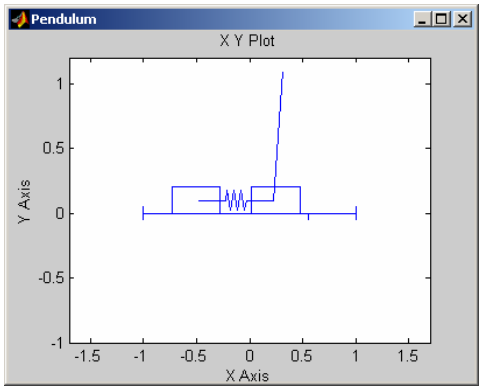
Figur 8.1 Simulering

För att komma närmre sanningen modellerades friktion med hjälp av ett dödband i styrsignalen. Resultatet syns i figur 8.2.

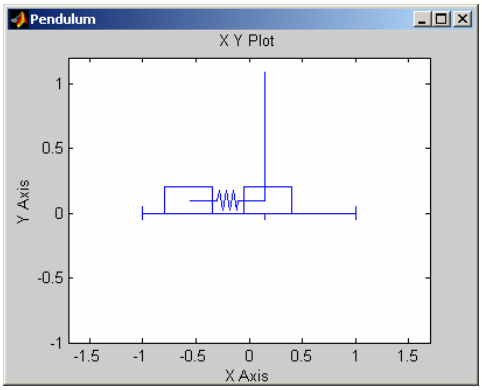


Figur 8.2 Simulering med enkel friktionsmodell

I figur 8.2 syns tydligt att vagnarna har en liten oscillering, som tillsammans med andra faktorer blir än tydligare i verkligheten. Mer om detta fenomen och dess orsak diskuteras senare i kapitlet. Vidare skapades en funktion som animerade händelseförloppet med vagnarna, pendeln och referensvärde för att få en bättre visuell effekt i Simulink än de figurer som visas ovan. Till detta användes *sfunxy*-funktionen som redan finns i Matlab, normalt för att rita xy-grafer. Denna har modifierats till att rita upp 7 noder istället för 2 och är därför omdöpt till sfunxy7 (Appendix D). I figur 8.3 och 8.4 visas utseendet av denna under körning.

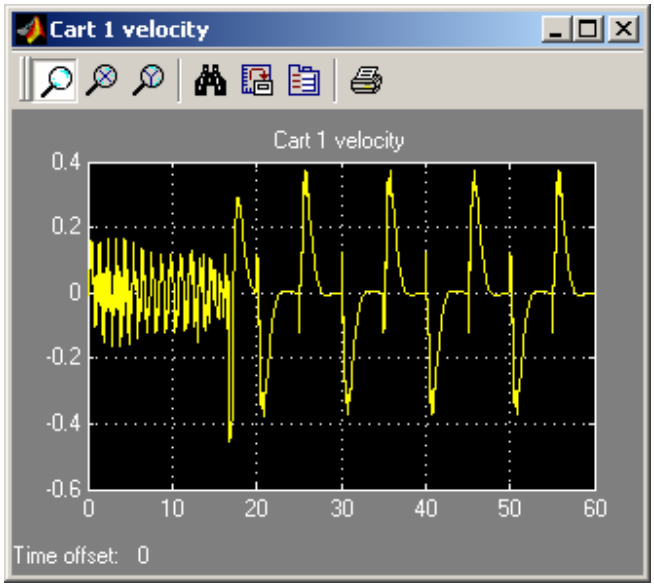


Figur 8.3 Vagnarna på väg mot referensvärdet



Figur 8.4 Vagnarna vid referensvärdet

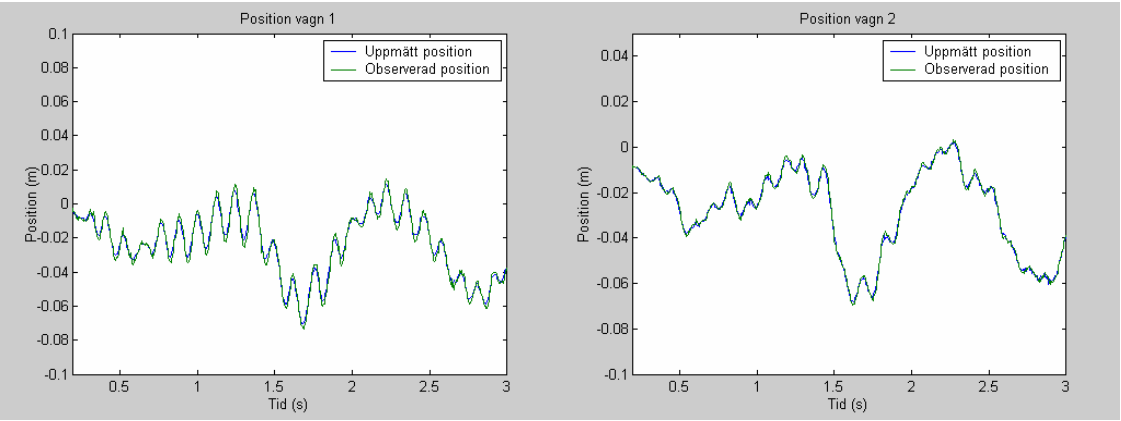
De första problemen kom istället när vi fick den verkliga processen och skulle börja styra stegmotorn. Denna är mer komplicerad än likströmsmotorn, där pålagd spänning direkt ger en kontinuerlig drift. Det krävs spänningspulser i rätt ordning och inte i för snabb takt så att motorn missar en puls. För att kunna köra stegmotorn i så hög hastighet som möjligt implementerades styrningen i en AVR (ATMEGA8) som arbetade direkt mot ett drivsteg, och skötte kommunikationen med Matlab via en seriell port. Nu kunde stegmotorn köras men trots mycket intrimning och modifiering av drivkort kom hastigheten inte upp till mer än 15 cm/s. Detta var inte tillräckligt för att balansera pendeln och nederlaget styrktes av datorsimuleringar där hastigheter på runt 40 cm/s krävdes (se figur 8.5).



Figur 8.5 Hastighetsgraf under simulering

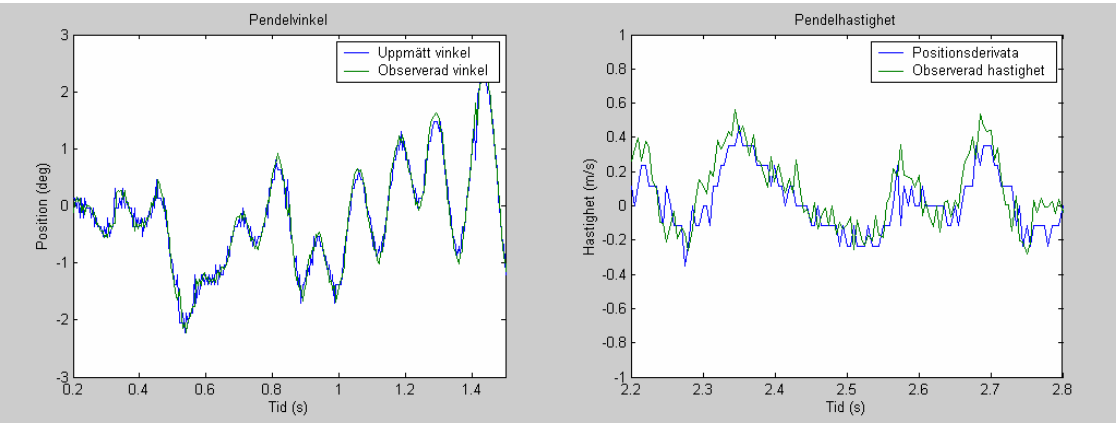
Lösningen blev att övergå till en DC-motor som styrs med hjälp av PWM och en H-brygga. Detta gav genast mycket bra resultat.

En observerare, så kallat Kalmanfilter, implementerades och gjordes cirka 5 gånger snabbare än regulatorpolerna varpå tillståndsåterkopplingen fungerade utmärkt. Eftersom en riktig identifiering av alla processparametrarna inte är utförd är det sannolikt att det är Kalmanfiltret som gör regleringen möjlig. I figur 8.6 till 8.9 syns tydligt att observeraren följer de faktiska värdena vilket leder till robust reglering. I figur 8.10 visas referensföljning av vagn 2, alltså den vagnen som pendeln står på.



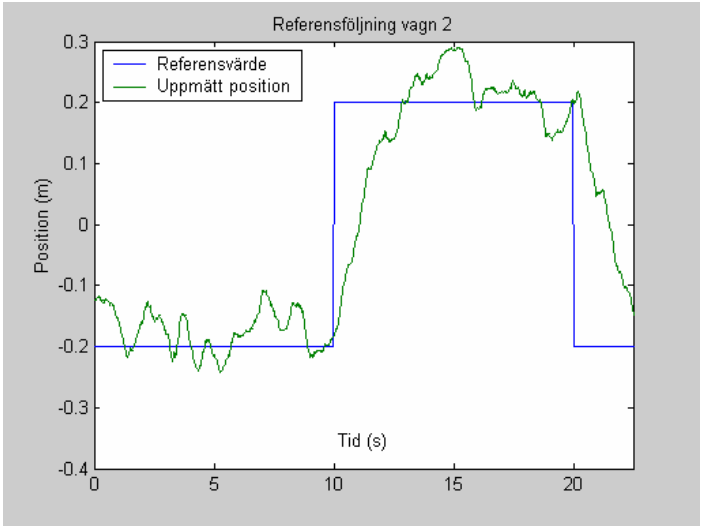
Figur 8.6 Position vagn 1

Figur 8.7 Position vagn 2



Figur 8.8 Pendelvinkel

Figur 8.9 Pendelhastighet



Figur 8.10 Referensföljning vagn 2

I figur 8.7 kan man se att pendeln verkar oscillera med en konstant frekvens då den balanserar upprätt, vilket troligen beror på framförallt 3 faktorer. Den första och största

härstammar från friktion på den vagn där pendeln är monterad, vilket man kunde se under simuleringen. Den andra faktorn är att hela uppställningen sitter fast på ett bord som inte är idealt. Bordet har ganska långa ben som inte stöttas på något sätt och är därför väldigt svajigt då processen körs. Det står dessutom på hjul och kan därför röra på sig. Den sistnämnda rörelsen är dock väldigt liten. Den tredje och sista faktorn är att remmen är något flexibel och ospänd vilket leder till att man inte alltid får ut den rörelsen på vagn 1 som önskas. Detta kan dock till viss del avhjälpas genom att spänna remmen.
Linjärgivarna som är relativt enkla och väldigt billiga fungerar helt utan problem. I och med att microprocessorn håller reda på positionerna kontinuerligt kan man starta och stoppa simuleringar i Matlab utan att nollställa processen. Vid nästa körning ges ändå korrekt position. Vi valde att göra en enkel interpolering i AVRen, vilket ger ganska hög precision. Om det önskas bättre precision vid senare tillfälle är det möjligt att göra interpoleringen i Matlab eller Simulink, då med hjälp av sinus eller cosinus funktioner. Detta tror vi skulle ge en extremt god noggrannhet, men då det inte är något som behövs i detta skede, har vi valt att inte undersöka saken.
På slutet av projektet lades funktioner till för att dämpa pendeln samt för att svinga upp den. Funktioner gjordes även för att växla mellan de olika uppgifterna, dämpning, balansering och uppsvingning. Detta innebär att man kan starta med pendeln hängande neråt och med en knapptryckning få den att automatiskt svinga upp och ställa sig i upprätt läge. Vidare kan man byta reglering och då dämpas pendeln till nedåthängande på kortast möjliga tid för att sedan köra till önskat läge utan att pendeln dinglar mer än nödvändigt. Dessa funktioner fungerar väldigt bra, men någon optimering har inte gjorts på funktionen som svingar upp pendeln och ibland tar det därför några försök innan regulatorn lyckas fånga den.
Även några säkerhetsbrytare har lagts till i mjukvaran för att inte riskera haveri vid okontrollerade körningar av processen. Till exempel är nu körning i närheten av ändlägena på gejden inte möjlig och skulle fjädern sträckas ut eller tryckas ihop för mycket avbryts simuleringen.
Målet med examensarbetet anses som uppnått eftersom en fungerande prototyp till den eventuella framtida laborationsprocessen är färdigställd. Motsvarande process går att köpa färdig av Quanser Consulting Inc [5]. som tillverkar olika laborationsuppställningar.

Priset på denna uppställning är 8755 $. Vi har uppskattat att kostnaden för framställningen av vår process ligger under 20000 kr per exemplar.

9 Referenser

[1] SOLECTRO (2004). Kontaktperson: Mattias Tetzlaff
http://www.solectro.se

[2] Douglas W. Jones (2004). http://www.cs.uiowa.edu/~jones/step/, University of Iowa

[3] Alaküla, Mats (2002). ”Power Electronic Control”, IEA/LTH, Lund

[4] Hägglund, Tore (2001). ”REGLERTEKNIK AK Föreläsningar”, Institutionen för reglerteknik/LTH, Lund.

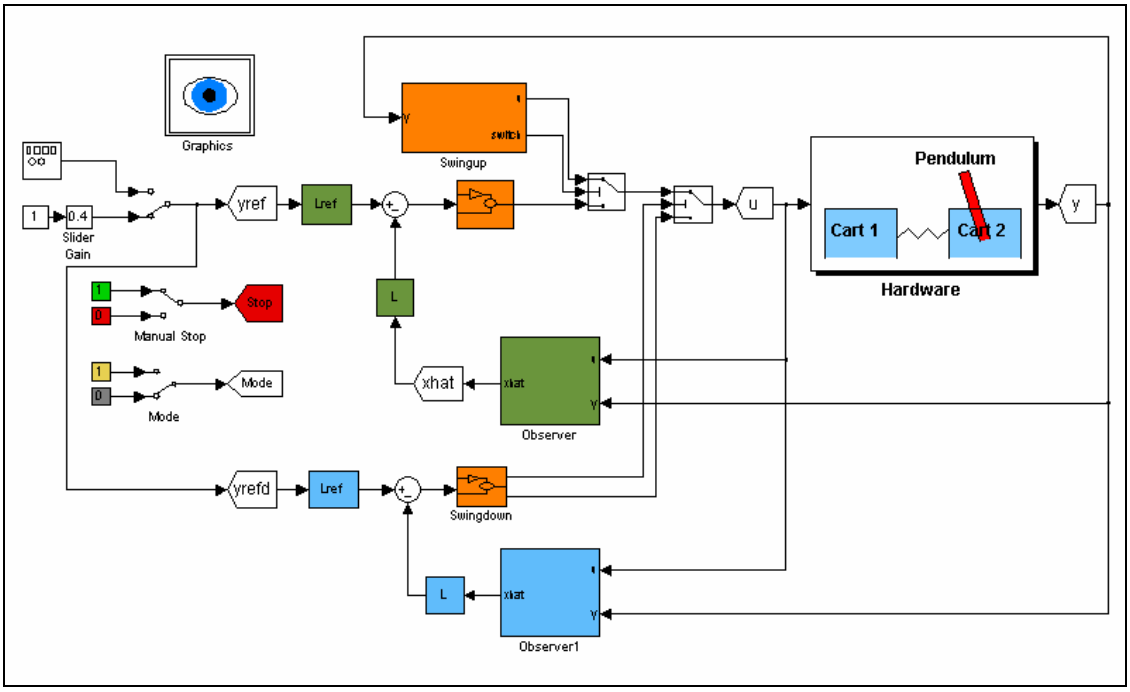
[5] QUANSER INNOVATE. EDUCATE. (2004). Kontaktperson: Andrew Romhanyi
http://www.quanser.com/

Appendix A Matlab m-fil

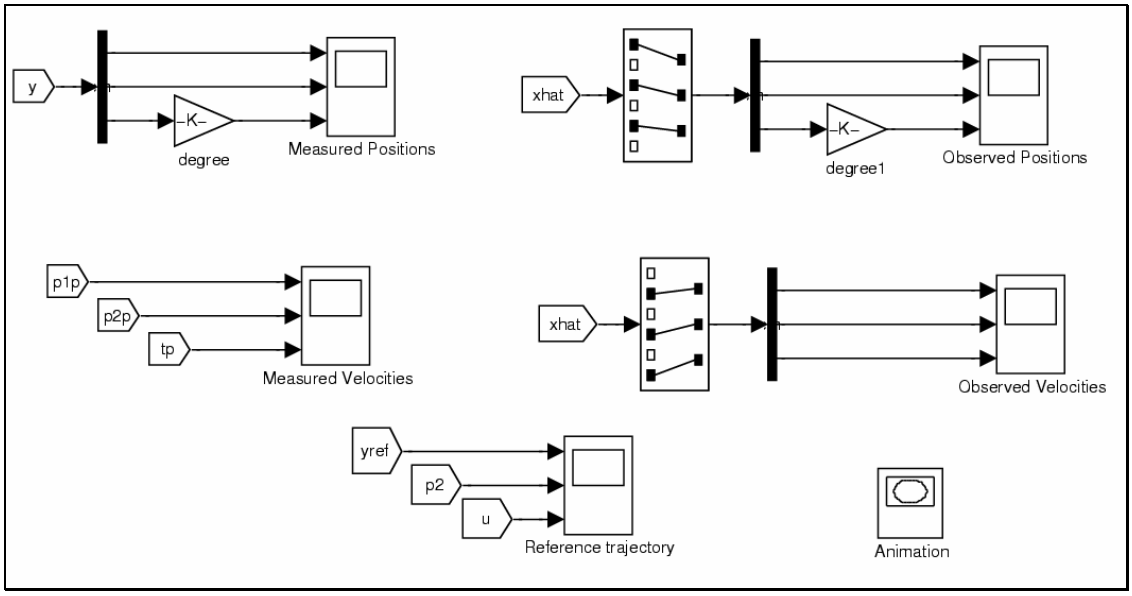
```
%========================================================================
%
% Initiering av Simulink
% Skapad 20040215 av Jörgen Lindgren och Pelle Bergkvist
% Senast ändrad 20040621
%
%========================================================================

%========================================================================
%
% Processkonstanter
%
%========================================================================

g = 9.82;                                       % Gravitaionskonstanten m/s^2
h = 1/200;                                      % Samplings intervall
m1 = 3.075;                                     % Massa Vagn1 kg
d1 = 40;                                        % Dämpning Vagn1 N/(m*s)
ky1 = 1;                                        % Positionssignal Vagn1 V/m
m2 = 3.455;                                     % Massa Vagn2 kg
d2 = 2;                                         % Dämpning Vagn2 N/(m*s)
ky2 = 1;                                        % Positionssignal Vagn2 V/m
k = 687.44;                                     % Fjäderkonstant N/m
m3 = 0.04;                                      % Massa pendel kg
l = 0.25;                                       % Pendelns halva längd m
w0 = 0.7819;                                    % Pendelns egensvängningsfrekvens (Hz)
J = 0.00009;                                    % Pendelns tröghetsmoment kg*m^2
ky3 = 1;                                        % Vinkelsignal pendel V/rad
offsettheta = 7.35550878482150;                 % Pendelns övre offset
onsettheta = 2.11340521237351;                  % Pendelns undre offset
gaintheta = (offsettheta - onsettheta)/pi;      % Pendelns vinkelförstärkning
km = 2;                                         % Kontrollsignal motor V/N

%========================================================================
%
% Linear Flexible Joint Cart with Inverted Pedulum (LFJCIP) model
%
%========================================================================

a = J+m3*l^2;
b = m3*l;
c = a*(m2+m3)-b^2;

Ac = [0 1 0 0 0 0;
    -k/m1 -d1/m1 k/m1 0 0 0;
     0 0 0 1 0 0;
     k*a/c 0 -k*a/c -d2*a/c b^2*g/c 0;
     0 0 0 0 0 1;
     k*b/c 0 -k*b/c -d2*b/c b*g*(m2+m3)/c 0];
Bc = [0 km/m1 0 0 0 0]';
Cc1 = [ky1 0 0 0 0 0];
```

```
Cc2 = [0 0 ky2 0 0 0];
Cc3 = [0 0 0 0 ky3 0];
Cc  = [Cc1;Cc2;Cc3];
Dc = [0 0 0]';

%=============================================================================
%
% Linear Quadratic Regulator design
%
%=============================================================================

q11 = 10;                                   %
q33 = 100;                                  % Viktning av positioner
q55 = 500;                                  %
Q = [q11 0 0 0 0 0;
     0 0 0 0 0 0;
     0 0 q33 0 0 0;
     0 0 0 0 0 0;
     0 0 0 0 q55 0;
     0 0 0 0 0 0];
R = 1;                                      % Straff styrsignal
L = lqr(Ac,Bc,Q,R);
Acreg = [(Ac-Bc*L)];
Bcreg = [Bc];
Ccreg = [Cc];
Dcreg = [Dc];

%=============================================================================
%
% Stationär förstärkning
%
%=============================================================================

Lref = 1/(Cc1*((-Ac+Bc*L)\Bc));             % Stationär förstärkningskonstant

%=============================================================================
%
% Observerardesign
%
%=============================================================================

p = polybutt(6,100,50);                     % Polplacering
K = (place(Ac',Cc',roots(p)))';
Ace = [(Ac-Bc*L) Bc*L;
       zeros(size(Ac)) Ac-K*Cc];
Bce = [Bc*Lref;
       zeros(size(Bc))];
Cce = [Cc zeros(size(Cc))];
Dce = [0 0 0]';
```

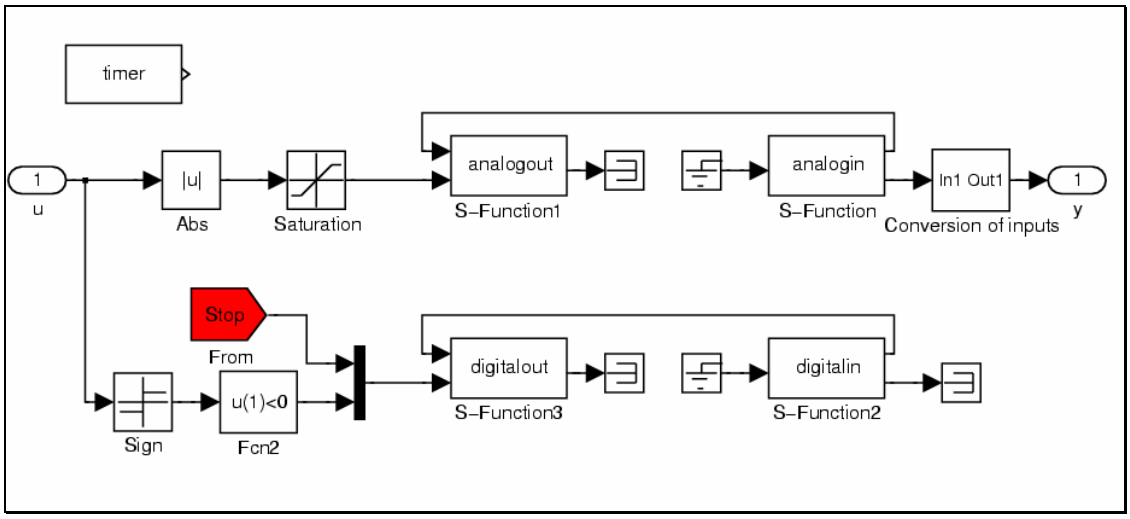
Appendix B Simulink



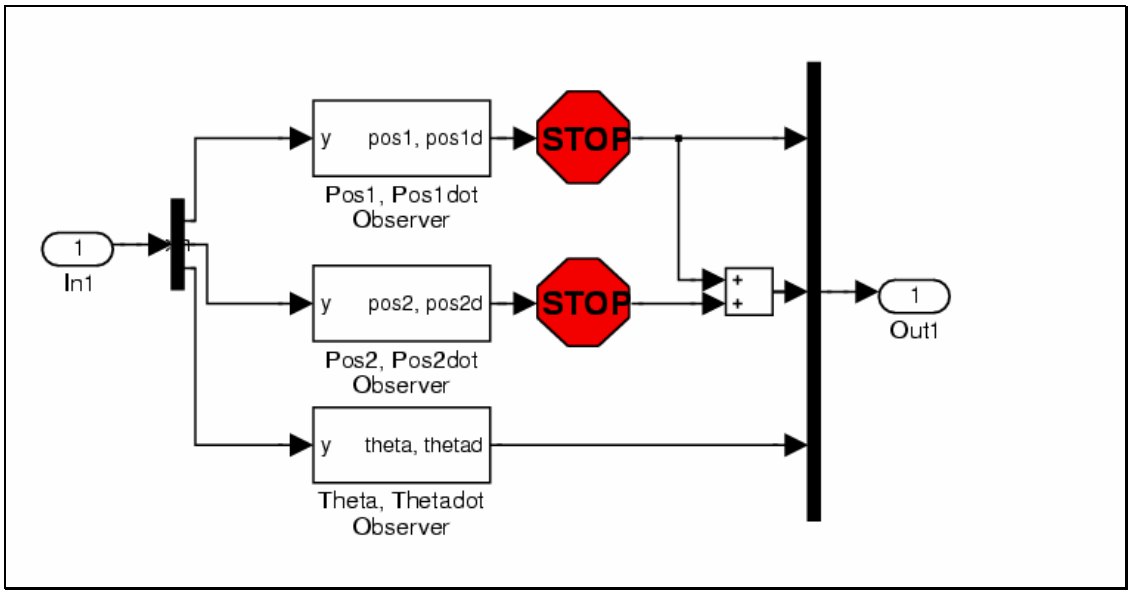
Huvuddiagrammet i Simulink



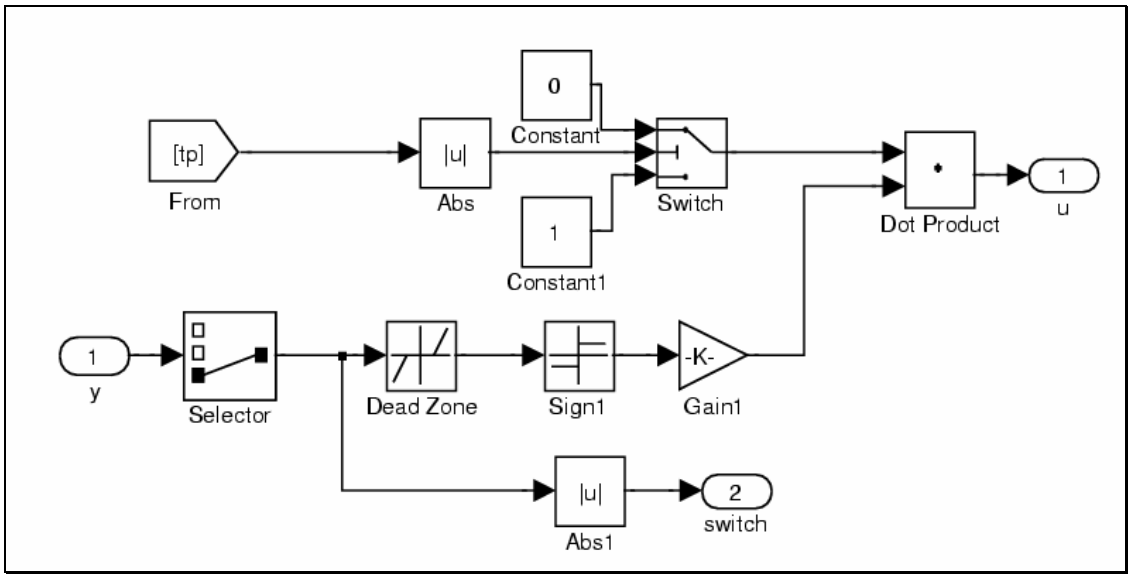
Graphics-blocket, innehållande all grafik



Hårdvarublocket, innehållande all kommunikation med kringutrustning



Conversion of inputs-blocket med mjukvarustopp



Swingup-blocket

Appendix C AVR c-kod

```
#include "avr/interrupt.h"
#include "avr/signal.h"
#include "avr/io.h"

#define CONF_DIG_IN(channel) (0x20 | (channel)&0x1f)
#define CONF_DIG_OUT(channel) (0x40 | (channel)&0x1f)

#define CONF_DIGITAL_IN(chan, config) \
  putchannel(31, (0x20|(chan&0x1f)|(config&0xffffff00)))
#define CONF_DIGITAL_OUT(chan, config) \
  putchannel(31, (0x40|(chan&0x1f)|(config&0xffffff00)))
#define CONF_ANALOG_IN(chan, config) \
  putchannel(31, (0x60|(chan&0x1f)|(config&0xffffff00)))
#define CONF_ANALOG_OUT(chan, config) \
  putchannel(31, (0x80|(chan&0x1f)|(config&0xffffff00)))
#define CONF_END() putchannel(31,0)
#define CONF_RESOLUTION(bits) (((bits)<<10)|0x000)
#define CONF_MIN(value) ((value)|0x100)
#define CONF_MAX(value) ((value)|0x200)
#define CONF_NEGATIVE_VOLT(volt) (((long)(volt)<<14)|0x2000)
#define CONF_POSITIVE_VOLT(volt) ((long)(volt)<<14)
#define CONF_NEGATIVE_MILLIVOLT(millivolt) (((long)(millivolt)<<14)|0x2400)
#define CONF_POSITIVE_MILLIVOLT(millivolt) ((long)(millivolt)<<14|0x400)

//interupt vectors

#define SIG_UART_RECV           _VECTOR(11)
#define SIG_UART_DATA           _VECTOR(12)
#define SIG_UART_TRANS          _VECTOR(13)
#define SIG_ADC                 _VECTOR(14)
#define SIG_OVERFLOW2           _VECTOR(4)
#define SIG_OUTPUT_COMPARE2     _VECTOR(3)

static volatile unsigned long serial_value;
static volatile unsigned char serial_length;
static volatile unsigned char serial_readbits;
static volatile unsigned char serial_readchannels;
static volatile unsigned char serial_readconfig;
static volatile unsigned int control;
static volatile unsigned int direction;
static volatile unsigned long position1;
static volatile unsigned long position2;
static volatile unsigned long A1;
static volatile unsigned long Aold1;
static volatile unsigned long B1;
static volatile unsigned long Bold1;
static volatile unsigned long A2;
static volatile unsigned long Aold2;
static volatile unsigned long B2;
static volatile unsigned long Bold2;
static volatile unsigned int flag;
```

```
static volatile unsigned int state1;
static volatile unsigned int stateOld1;
static volatile unsigned int state2;
static volatile unsigned int stateOld2;
static volatile unsigned int tempState1;
static volatile unsigned int tempState2;
static volatile signed int dir;
static volatile unsigned int HIGH1;
static volatile unsigned int LOW1;
static volatile unsigned int HIGH2;
static volatile unsigned int LOW2;

/*
 * ADConversion
 */
SIGNAL(SIG_ADC)
{
  unsigned char channel = ADMUX & 0x0f;
  unsigned char low = ADCL;
  unsigned char high = ADCH;
  unsigned long value = (high<<8) | low;

  switch (channel) {
    case 0: {
      channel = 1;
      Aold1 = A1;
      A1 = value;
      flag = 0;
    } break;
    case 1: {
      channel = 4;
      Aold2 = A2;
      A2 = value;
      flag = 1;
    } break;
    case 4: {
      channel = 5;
      Bold1 = B1;
      B1 = value;
      flag = 0;
    } break;
    case 5: {
      channel = 0;
      Bold2 = B2;
      B2 = value;
      flag = 1;
    } break;
    default: {
      channel = 0;
    } break;
  }

  /*
   * Statemachine
```

```
   */
  if(flag==0){
    tempState1 = state1;
    if(A1<LOW1 && B1<LOW1)
      state1 = 0;
    else if(A1<LOW1 && B1>HIGH1)
      state1 = 1;
    else if(A1>HIGH1 && B1>HIGH1)
      state1 = 2;
    else if(A1>HIGH1 && B1<LOW1)
      state1 = 3;
    if(tempState1 != state1){
      stateOld1 = tempState1;
      dir = state1-stateOld1;
      if(dir == -3 || dir == 1){
        position1 += 100;
      }else {
        position1 -= 100;
      }
    }
  } else {
    tempState2 = state2;
    if(A2<LOW2 && B2<LOW2)
      state2 = 0;
    else if(A2<LOW2 && B2>HIGH2)
      state2 = 1;
    else if(A2>HIGH2 && B2>HIGH2)
      state2 = 2;
    else if(A2>HIGH2 && B2<LOW2)
      state2 = 3;
    if(tempState2 != state2){
      stateOld2 = tempState2;
      dir = state2-stateOld2;
      if(dir == -3 || dir == 1){
        position2 -= 100;
      }else {
        position2 += 100;
      }
    }
  }
  ADMUX =  0x40 | channel; // Vcc Vref, right adjust
  ADCSR =  0xcf;           // Enable ADC interrupts, Clock/128

}


/*
 * serial link configuration
 */
typedef enum { cmd_clear_bit, cmd_set_bit,
               cmd_read_bit, cmd_read_chan } command;

SIGNAL(SIG_UART_RECV)
{
```

```
  char ch = UDR;
  serial_length++;
  if ((ch & 0x80) == 0x80) {
    serial_value = (serial_value << 7) | (ch & 0x7f);
  } else {
    serial_value = (serial_value << 2) | (ch &0x60) >> 5;
    unsigned char chan = ch & 0x1f;
    if (serial_length == 1 ) {
      unsigned char cmd = (serial_value & 0x03);
      switch(cmd){
        case cmd_clear_bit: {
          switch (chan) {
            case 0: { control = 0; } break;
            case 1: { direction = 0; PORTB = (PORTB & 0xef);} break;
          }
        } break;
        case cmd_set_bit: {
          switch (chan) {
            case 0: { control = 1; } break;
            case 1: { direction = 1; PORTB = (PORTB | 0x10);} break;
          }
        } break;
        case cmd_read_bit: {
         if (chan <= 7) { serial_readbits |= (1<<chan);}
        } break;
        case cmd_read_chan: {
          if (chan <= 7) { serial_readchannels |= (1<<chan); }
          if (chan == 31) { serial_readconfig = 1; }
        } break;
      }
    }
    else {
      switch (chan) {
        case 0: {
           OCR2 = serial_value;
        } break;
      }
    }
    serial_value = 0;
    serial_length = 0;
  }
}

static void putchar(unsigned char ch)
{
  while ((UCSRA & 0x20) == 0) {};
  UDR = ch;
}

void putbit(unsigned char channel, unsigned char value)
{
  if (value) {
    putchar(0x20 | channel);
  } else {
```

```
    putchar(0x00 | channel);
  }
}

void putchannel(unsigned char channel, unsigned long value)
{
  if (value >= (1L<<30)) { putchar(0x80 | ((value >> 30) & 0x03)); }
  if (value >= (1L<<23)) { putchar(0x80 | ((value >> 23) & 0x7f)); }
  if (value >= (1L<<16)) { putchar(0x80 | ((value >> 16) & 0x7f)); }
  if (value >= (1L<< 9)) { putchar(0x80 | ((value >> 9) & 0x7f)); }
  putchar(0x80 | ((value >> 2) & 0x7f));
  putchar(((value << 5) & 0x60) | channel);
}

/*
 * Timer interrupts for PWM
 */
SIGNAL(SIG_OVERFLOW2)
{
    PORTB = (PORTB | 0x02);
    PORTD = (PORTD & 0x7f);
}
SIGNAL(SIG_OUTPUT_COMPARE2)
{
    PORTB = (PORTB & 0xfd);
    PORTD = (PORTD | 0x80);
}

/*
 * Interpolation within states
 */
long calcDelta1(long a, int state) {
  if(state == 0)
    return (a-675)/3;
  else if(state == 1)
    return (375-a)/3;
  else if(state == 2)
    return (675-a)/3;
  else
    return (a-975)/3;
}
long calcDelta2(long a, int state) {
  if(state == 0)
    return 2*(a-500)/7;
  else if(state == 1)
    return 2*(150-a)/7;
  else if(state == 2)
    return 2*(500-a)/7;
  else
    return 2*(a-850)/7;
}

/*
 * Main
```

```
 */
int main()
{
  serial_value = 0;
  serial_length = 0;
  serial_readbits = 0;
  serial_readconfig = 0;
  serial_readchannels = 0;

  position1 = 400000;
  position2 = 400000;
  control = 0;
  Aold1 = 600;
  Bold1 = 600;
  stateOld1 = 0;
  stateOld2 = 0;
  HIGH1 = 700;
  LOW1 = 650;
  HIGH2 = 525;
  LOW2 = 475;

  DDRB = 0x37;          // PortB, 0-5 output , 3 input
  DDRD = 0xf4;          // PortD, 2, 4-7 output
  PORTC = 0x00;         // PortC, no pull up
  DDRC = 0x00;          // PortC, input
  UCSRA = 0x00;         // USART:
  UCSRB = 0x98;         // USART: RxIntEnable|RxEnable|TxEnable
  UCSRC = 0x86;         // USART: 8bit, no parity
  UBRRH = 0;            // USART: 38400 @ 14.7456MHz
  UBRRL = 23;           // USART: 38400 @ 14.7456MHz
  ADMUX = 0x40;         // Vcc Vref, right adjust
  ADCSR = 0xcf;         // Enable ADC, Clock/128
  TCCR0 = 0x03;         // Timer0, Clock / 64
  TCCR2 = 0x6f;         // Timer2, Fast PWM, Clock / 64
  TIMSK = 0xc0;         // Enable Timer2 interrupts
  SREG = 0x80;          // Global interrupt enable
  OCR2 = 0x80;          // PWM enable

  while (1) {
    unsigned char bits, channels, config;
    SREG = 0x00;    // Global interrupt disable
    bits = serial_readbits;
    serial_readbits = 0;
    channels = serial_readchannels;
    serial_readchannels = 0;
    config = serial_readconfig;
    serial_readconfig = 0;
    SREG = 0x80;    // Global interrupt enable
    if (control == 0){
      OCR2 = 0x00;  // PWM disable
    }
    if (channels & 0x01) { putchannel(0, position1-calcDelta1(A1,state1)); }
    if (channels & 0x02) { putchannel(1, position2+calcDelta2(A2,state2)); }
    if (config) {
```

```
      CONF_DIGITAL_OUT(0, CONF_RESOLUTION(1));                  // ON/OFF
      CONF_DIGITAL_OUT(1, CONF_RESOLUTION(1));                  // Direction

      CONF_ANALOG_IN(0, CONF_RESOLUTION(10));                   // Position Cart 2
      CONF_ANALOG_IN(0, CONF_MIN(CONF_POSITIVE_VOLT(0)));
      CONF_ANALOG_IN(0, CONF_MAX(CONF_POSITIVE_VOLT(10)));

      CONF_ANALOG_IN(1, CONF_RESOLUTION(10));                   // Position Cart 1
      CONF_ANALOG_IN(1, CONF_MIN(CONF_POSITIVE_VOLT(0)));
      CONF_ANALOG_IN(1, CONF_MAX(CONF_POSITIVE_VOLT(10)));

      CONF_ANALOG_OUT(0, CONF_RESOLUTION(8));                   // Control signal
      CONF_ANALOG_OUT(0, CONF_MIN(CONF_POSITIVE_VOLT(0)));
      CONF_ANALOG_OUT(0, CONF_MAX(CONF_POSITIVE_VOLT(12)));

      CONF_END();
    }
  }
}
```

Appendix D sfunxy7.m

```
function [sys, x0, str, ts] = sfunxy4(t,x,u,flag,ax,varargin)
%SFUNXY S-function that acts as an X-Y scope using MATLAB plotting functions.
%   This M-file is designed to be used in a Simulink S-function block.
%   It draws a line from the previous input point, which is stored using
%   discrete states, and the current point.  It then stores the current
%   point for use in the next invocation.
%
%   See also SFUNXYS, LORENZS.

%   Copyright (c) 1990-97 by The MathWorks, Inc.
%   $Revision: 1.30 $
%   Andrew Grace 5-30-91.
%   Revised Wes Wang 4-28-93, 8-17-93, 12-15-93
%   Revised Craig Santos 10-28-96

switch flag

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,ts] = mdlInitializeSizes(ax,varargin{:});
    SetBlockCallbacks(gcbh);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2
    sys = mdlUpdate(t,x,u,flag,ax,varargin{:});

  %%%%%%%%%
  % Start %
  %%%%%%%%%
  case 'Start'
    LocalBlockStartFcn

  %%%%%%%%
  % Stop %
  %%%%%%%%
  case 'Stop'
    LocalBlockStopFcn
```

```
  %%%%%%%%%%%%%%
  % NameChange %
  %%%%%%%%%%%%%%
  case 'NameChange'
    LocalBlockNameChangeFcn

  %%%%%%%%%%%%%%%%%%%%%%%%
  % CopyBlock, LoadBlock %
  %%%%%%%%%%%%%%%%%%%%%%%%
  case { 'CopyBlock', 'LoadBlock' }
    LocalBlockLoadCopyFcn

  %%%%%%%%%%%%%%%
  % DeleteBlock %
  %%%%%%%%%%%%%%%
  case 'DeleteBlock'
    LocalBlockDeleteFcn

  %%%%%%%%%%%%%%%%
  % DeleteFigure %
  %%%%%%%%%%%%%%%%
  case 'DeleteFigure'
    LocalFigureDeleteFcn

  %%%%%%%%%%%%%%%%
  % Unused flags %
  %%%%%%%%%%%%%%%%
  case { 3, 9 }
    sys = [];

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    if ischar(flag),
      errmsg=sprintf('Unhandled flag: ''%s''', flag);
    else
      errmsg=sprintf('Unhandled flag: %d', flag);
    end

    error(errmsg);

end

% end sfunxy

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,ts] = mdlInitializeSizes(ax,varargin)
```

```
if length (ax)~=4
  error(['Axes limits must be defined.'])
end

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
sizes.NumInputs      = 7;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0 = [];

%
% initialize the array of sample times, note that in earlier
% versions of this scope, a sample time was not one of the input
% arguments, the varargs checks for this and if not present, assigns
% the sample time to -1 (inherited)
%
if ~isempty(varargin) > 0
  ts = [varargin{1} 0];
else
  ts = [-1 0];
end

% end mdlInitializeSizes

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u,flag,ax,varargin)

%
% always return empty, there are no states...
%
sys = [];

%
% Locate the figure window associated with this block.  If it's not a valid
% handle (it may have been closed by the user), then return.
%
FigHandle=GetSfunXYFigure(gcbh);
if ~ishandle(FigHandle),
   return
end
```

```
%
% Get UserData of the figure.
%
ud = get(FigHandle,'UserData');

delta = (u(3)-u(1)+0.2)/10;

x_data = [1 1 1 u(7)+0.35 u(7)+0.35 u(7)+0.35 -1 -1 -1 -1 u(1)-0.6 u(1)-0.6 u(1)-0.15
u(1)-0.15 u(1)-0.15 u(1)-0.35 u(1)-0.10 u(1)-0.10+delta u(1)-0.10+3*delta u(1)-
0.10+5*delta u(1)-0.10+7*delta u(1)-0.10+9*delta u(3)+0.10 u(3)+0.15 u(3)+0.15 u(3)+0.6
u(3)+0.6 u(3)+0.15 u(3)+0.15 u(3)+0.35 u(3)+0.35+u(5)];
y_data = [-0.05 0.05 0 0 -0.05 0 0 -0.05 0.05 0 0 0.2 0.2 0 0.1 0.1 0.1 0.175 0.025 0.175
0.025 0.175 0.1 0.1 0.2 0.2 0 0 0.1 0.1 0.1+u(6)];

% plot the input lines
set(ud.XYAxes, ...
    'Visible','on',...
    'Xlim', ax(1:2),...
    'Ylim', ax(3:4));
set(ud.XYLine,...
    'Xdata',x_data,...
    'Ydata',y_data,...
        'Erasemode','background',...
    'LineStyle','-');
set(ud.XYTitle,'String','X Y Plot');
set(FigHandle,'Color',get(FigHandle,'Color'))
drawnow


% end mdlUpdate

%
%=============================================================================
% LocalBlockStartFcn
% Function that is called when the simulation starts.  Initialize the
% XY Graph scope figure.
%=============================================================================
%
function LocalBlockStartFcn

%
% get the figure associated with this block, create a figure if it doesn't
% exist
%
FigHandle = GetSfunXYFigure(gcbh);
if ~ishandle(FigHandle),
  FigHandle = CreateSfunXYFigure;
end

ud = get(FigHandle,'UserData');
set(ud.XYLine,'Erasemode','xor');
set(ud.XYLine,'XData',[],'YData',[]);
set(ud.XYLine,'XData',0,'YData',0,'Erasemode','xor');
ud.XData = [];
```

```
ud.YData = [];
set(FigHandle,'UserData',ud);

% end LocalBlockStartFcn

%
%=============================================================================
% LocalBlockStopFcn
% At the end of the simulation, set the line's X and Y data to contain
% the complete set of points that were acquire during the simulation.
% Recall that during the simulation, the lines are only small segments from
% the last time step to the current one.
%=============================================================================
%
function LocalBlockStopFcn

FigHandle=GetSfunXYFigure(gcbh);
if ishandle(FigHandle),
  %
  % Get UserData of the figure.
  %
  ud = get(FigHandle,'UserData');
  set(ud.XYLine,...
      'Xdata',ud.XData,...
      'Ydata',ud.YData,...
      'LineStyle','-');

end

% end LocalBlockStopFcn

%
%=============================================================================
% LocalBlockNameChangeFcn
% Function that handles name changes on the Graph scope block.
%=============================================================================
%
function LocalBlockNameChangeFcn

%
% get the figure associated with this block, if it's valid, change
% the name of the figure
%
FigHandle = GetSfunXYFigure(gcbh);
if ishandle(FigHandle),
  set(FigHandle,'Name',get_param(gcbh,'Name'));
end

% end LocalBlockNameChangeFcn

%
%=============================================================================
% LocalBlockLoadCopyFcn
% This is the XYGraph block's LoadFcn and CopyFcn.  Initialize the block's
```

```
% UserData such that a figure is not associated with the block.
%=============================================================================
%
function LocalBlockLoadCopyFcn

SetSfunXYFigure(gcbh,-1);

% end LocalBlockLoadCopyFcn

%
%=============================================================================
% LocalBlockDeleteFcn
% This is the XY Graph block'DeleteFcn.  Delete the block's figure window,
% if present, upon deletion of the block.
%=============================================================================
%
function LocalBlockDeleteFcn

%
% Get the figure handle associated with the block, if it exists, delete
% the figure.
%
FigHandle=GetSfunXYFigure(gcbh);
if ishandle(FigHandle),
  delete(FigHandle);
  SetSfunXYFigure(gcbh,-1);
end

% end LocalBlockDeleteFcn

%
%=============================================================================
% LocalFigureDeleteFcn
% This is the XY Graph figure window's DeleteFcn.  The figure window is
% being deleted, update the XY Graph block's UserData to reflect the change.
%=============================================================================
%
function LocalFigureDeleteFcn

%
% Get the block associated with this figure and set it's figure to -1
%
ud=get(gcbf,'UserData');
SetSfunXYFigure(ud.Block,-1)

% end LocalFigureDeleteFcn

%
%=============================================================================
% GetSfunXYFigure
% Retrieves the figure window associated with this S-function XY Graph block
% from the block's parent subsystem's UserData.
%=============================================================================
%
```

```
function FigHandle=GetSfunXYFigure(block)

if strcmp(get_param(block,'BlockType'),'S-Function'),
  block=get_param(block,'Parent');
end

FigHandle=get_param(block,'UserData');
if isempty(FigHandle),
  FigHandle=-1;
end

% end GetSfunXYFigure

%
%=============================================================================
% SetSfunXYFigure
% Stores the figure window associated with this S-function XY Graph block
% in the block's parent subsystem's UserData.
%=============================================================================
%
function SetSfunXYFigure(block,FigHandle)

if strcmp(get_param(bdroot,'BlockDiagramType'),'model'),
  if strcmp(get_param(block,'BlockType'),'S-Function'),
    block=get_param(block,'Parent');
  end

  set_param(block,'UserData',FigHandle);
end

% end SetSfunXYFigure

%
%=============================================================================
% CreateSfunXYFigure
% Creates the figure window associated with this S-function XY Graph block.
%=============================================================================
%
function FigHandle=CreateSfunXYFigure

%
% the figure doesn't exist, create one
%
FigHandle = figure('Units',          'pixel',...
                   'Position',       [100 100 400 300],...
                   'Name',           get_param(gcbh,'Name'),...
                   'NumberTitle',    'off',...
                   'IntegerHandle',  'off',...
                   'DeleteFcn',      'sfunxy6([],[],[],''DeleteFigure'')');
%
% store the block's handle in the figure's UserData
%
ud.Block=gcbh;
```

```
%
% create various objects in the figure
%
ud.XYAxes   = axes;
ud.XYLine   = plot(0,0,'EraseMode','xor');
ud.XYXlabel = xlabel('X Axis');
ud.XYYlabel = ylabel('Y Axis');
ud.XYTitle  = get(ud.XYAxes,'Title');
ud.XData    = [];
ud.YData    = [];
set(ud.XYAxes,'Visible','off');

%
% Associate the figure with the block, and set the figure's UserData.
%
SetSfunXYFigure(gcbh,FigHandle);
set(FigHandle,'HandleVisibility','callback','UserData',ud);

% end CreateSfunXYFigure

%
%=============================================================================
% SetBlockCallbacks
% This sets the callbacks of the block if it is not a reference.
%=============================================================================
%
function SetBlockCallbacks(block)

%
% the actual source of the block is the parent subsystem
%
block=get_param(block,'Parent');

%
% if the block isn't linked, issue a warning, and then set the callbacks
% for the block so that it has the proper operation
%
if strcmp(get_param(block,'LinkStatus'),'none'),
  warnmsg=sprintf(['The XY Graph scope ''%s'' should be replaced with a ' ...
                   'new version from the Simulink block library'],...
                   block);
  warning(warnmsg);

  callbacks={
    'CopyFcn',       'sfunxy([],[],[],''CopyBlock'')' ;
    'DeleteFcn',     'sfunxy([],[],[],''DeleteBlock'')' ;
    'LoadFcn',       'sfunxy([],[],[],''LoadBlock'')' ;
    'StartFcn',      'sfunxy([],[],[],''Start'')' ;
    'StopFcn'        'sfunxy([],[],[],''Stop'')'
    'NameChangeFcn', 'sfunxy([],[],[],''NameChange'')' ;
  };

  for i=1:length(callbacks),
    if ~strcmp(get_param(block,callbacks{i,1}),callbacks{i,2}),
```

```
      set_param(block,callbacks{i,1},callbacks{i,2})
    end
  end
end

% end SetBlockCallbacks
```