Kontinuierliche Simulation

325.040 - Projekt 47 - *Sommersemester 2016*

Fabian Wedenik - 1426866
Alexander Wimmer - 1328958
Felix Hochwallner - 1328839
Oskar Fürnhammer - 1329133

E325 Institut für Mechanik und Mechatronik

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Vorwort

Sehr geehrte Damen und Herren, liebe Leser und Leserinnen!

Das vorliegende Protkoll wurde im Rahmen der Vorlesung und Übung *Kontinuierliche Simluation (325.040/325.041)* verfasst und beschäftigt sich mit der Implementierung einer einfachen Regelung eines mechanischen Doppelpendels, sowohl in MATLAB, als auch in MalpeSim.
Dadurch soll unter anderem ein Vergleich zwischen klassischer textuelle Programmierung und grafischer, blockorientierter Modellierung gezogen werden. Betreut wurde das Projekt der Gruppe 47 von Fabian Germ.

Viel Spaß beim Lesen wünschen

2

Aufgabenstellung

Sowohl mit MATLAB als auch MapleSim soll ein mechanisches Modell eines geregelten Doppelpendels realisiert werden. Dabei soll unter anderem ein Vergleich zwischen klassischer textueller Programmierung in MATLAB und grafischer, blockorientierter Modellierung in MapleSim gezogen werden.

Implementieren Sie das Modell mit MATLAB. Führen Sie einen Simulationslauf mit den angegebenen Parametern durch, plotten Sie die Auslenkung x sowie die beiden Winkel θ_1 und θ_2 über der Zeit und interpretieren Sie die Ergebnisse. Berechnen Sie mit MATLAB auch die Eigenwerte. Ist das System stabil? Begründen Sie Ihre Aussage.

Bauen Sie das Modell mit MapleSim auf, testen Sie das Modell mit den angegebenen Parametern und vergleichen Sie die Ergebnisse mit jenen aus der MATLAB-Simulation.

3

Modellbildung

Eine Masse m_m gleitet reibungsfrei auf einer horizontalen Ebene. An der Masse ist ein Stab (m_1, I_1, l_1) über ein reibungsfreies Gelenk befestigt. An seinem anderen Ende ist der Stab $m_1$mit einem weiteren Stab (m_2, I_2, l_2) gelenkig verbunden.

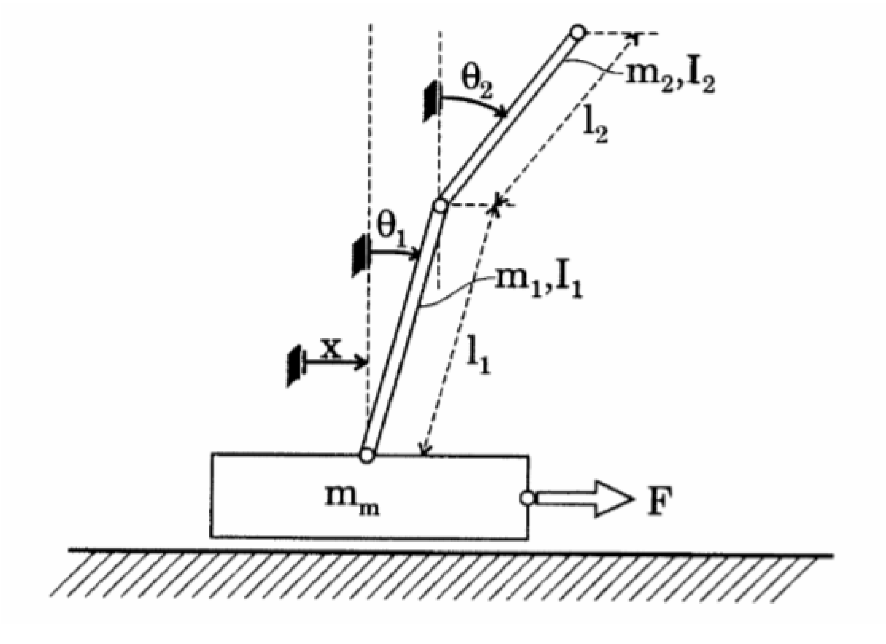


Abbildung 3.1: Mechanisches Modell eines stehenden Doppelpendels

Da wir bei der Berechnung der Matrizen, welche für eine Zustandsraumdarstellung erfoderlich sind, einige Probleme hatten entschlossen wir uns sicherheitshalber mittels Euler-Lagrange-Formalismen auch die Bewegungsgleichungen neu aufzustellen und in MATLAB linearisieren zu lassen. Die Bewegungsgleichung erhalten wir mithilfe der Langrange Gleichung 2.Art.

$$\frac{d}{dt}(\frac{\delta T}{\delta \dot{q}_i}) - \frac{\delta T}{\delta q_i} + \frac{\delta V}{\delta q_i} = 0 \tag{3.1}$$

Dafür werden die kinetische und die potentielle Energie benötigt. Die kinetische Energie setzt sich wiederum aus einem translatorischen und einem rotatorischen Anteil zusammen.

$$T = T_{trans} + T_{rot} \tag{3.2}$$

Um den translatorischen Anteil zu berechnen werden die Geschwindigkeitsvektoren der Körper benötigt.

$$T_{trans} = \frac{1}{2} m \vec{v}^2 \tag{3.3}$$

$$\vec{v} = J_v \dot{\vec{q}} \tag{3.4}$$

Die Jacobi-Matrix J besteht aus den partiellen Ableitungen der Ortsvektoren zu den Schwerpunkten nach den Minimalkoordinaten. Der rotatorische Anteil wird mit Hilfe der Winkelgeschwindigkeitsvektoren der Stäbe und der Trägheitstensoren berechnet.

$$T_{rot} = \frac{1}{2} I_s \vec{\omega}^2 \tag{3.5}$$

Um die Energien in die Lagrange Gleichung 2.Art einsetzen zu können müssen sie partiell Abgeleitet werden (Siehe Gleichgung 3.1). Dies geschieht wiederum mit einer Jacobi-Matrix.
Damit erhalten wir schließlich die Bewgungsgleichungen in folgender Form:

$$M(q)\ddot{q} + f(q, \dot{q}) = 0 \tag{3.6}$$

Diese linearisieren wir nun um die Ruhelage, indem wir die Lagekoordinaten θ_1, θ_2 und x, sowie ihre Ableitung, nullsetzen.

4

Implementierung in MATLAB

MATLAB ist eine numerische Programmiersprache, welche für die schnelle Manipulation und Berechnung von Matrizen entwickelt wurde. Programmiert wird unter Matlab in einer proprietären Programmiersprache, die auf der jeweiligen Maschine interpretiert wird. Die Programmierung erfolgt hierbei textuell.

4.1 Variablendifinition und Modellbildung

Bevor wir unser System simulieren lassen können, müssen wir unser mechanisches (Ersatz-)System in ein digitales Modell übersetzen. Dazu müssen dem Programm einige Parameter übergeben werden.
Zuerst werden Systemvariablen deklariert, sowie die Anzahl der Freiheitsgrade und Körper festgelegt. Außerdem wird ein Minimalkoordinatenvektor mit zugehörigen zeitlichen Ableitungen bestimmt.

```
%---- Ermitteln der Bewegungsgleichungen
%     definieren der Systemvariablen
syms l1 l2 th1 th2 th1_p th2_p th1_pp th2_pp
syms x x_p x_pp mm m1 m2 g I_1 I_2 F xc

frg=3;                                          %Anzahl der
    Freiheitsgrade
n=3;                                            %Anzahl der Koerper

q=[x ; th1 ; th2];                              %Minimalkoordinaten
q_p=[x_p ; th1_p ; th2_p];                      %zeitliche
    Ableitungen
q_pp=[x_pp ; th1_pp ; th2_pp];
```

Außerdem benötigen wir noch die Ortvektoren, sowie diverse Koeffizietenmatrizen um später in die Lagrange'sche Gleichung 2. Art einsetzen zu können.

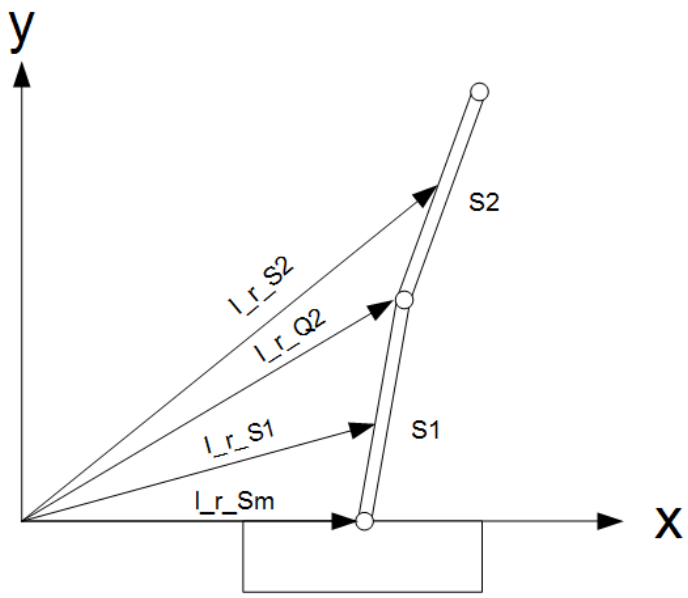


Abbildung 4.1: Ortsvektoren zu den Gelenk- und Schwerpunkten

```
%---- Drehmatrix Stab 1
T_IK1 = [cos(th1) sin(th1) 0;
        -sin(th1) cos(th1) 0;
             0        0    1];
%---- Drehmatrix Stab 2
T_IK2 = [cos(th2) sin(th2) 0;
        -sin(th2) cos(th2) 0;
             0        0    1];

%---- Ortsvektoren
I_r_Sm = [x;0;0];
I_r_S1 = [x+l1/2*sin(th1) ; l1/2*cos(th1) ; 0];
I_r_Q2 = [x+l1*sin(th1) ; l1*cos(th1) ; 0];
K1_r_Q1S1 = [0; l1/2; 0];
K2_r_Q2S2 = [0; l2/2; 0];
I_r_S2 = I_r_Q2 + T_IK2 * K2_r_Q2S2;

%---- Traegheitstensoren in den koerperfesten
    Koordinatensystemen
K1_I_S1 = diag([0 0 I_1]);
K2_I_S2 = diag([0 0 I_2]);

%---- Winkelgeschwindigkeitsvektoren der Staebe
K_om1 = [0 ; 0 ; -th1_p];
```

```
K_om2 = [0 ; 0 ; -th2_p];

%---- JACOBI-Matrizen der Translation
J_Tm = jacobian(I_r_Sm , q);
J_T1 = jacobian(I_r_S1 , q);
J_T2 = jacobian(I_r_S2 , q);

%---- JACOBI-Matrizen der Rotation
J_R1 = jacobian(K_om1, q_p);
J_R2 = jacobian(K_om2, q_p);

%---- Geschwindigkeitsvektoren
I_v_Sm = J_Tm*q_p ;
I_v_S1 = J_T1*q_p ;
I_v_S2 = J_T2*q_p ;

%---- kinetische Energie
T = 1/2*(mm*(I_v_Sm.'*I_v_Sm)+m1*(I_v_S1.'*I_v_S1)+m2*(I_v_S2.'*I_v_S2) ...%Translation
    +K_om1.'*K1_I_S1*K_om1+K_om2.'*K2_I_S2*K_om2);                    %Rotation
T = simplify(T);                                                      %Vereinfachung

%---- potentielle Energie
V=-(m1*I_r_S1.'+m2*I_r_S2.')*[0 ; -g ; 0];

%---- Ableitungen fuer LAGRANGEsche Gleichung 2. Art
dTdv = simplify(jacobian(T,q_p).');          %mit transponieren zu Spaltenvektor gemacht
dTdq = simplify(jacobian(T,q).');
dVdq = simplify(jacobian(V,q).');

%---- Elemente der Bewegungsgleichung M(q)*q_pp + f(q,q_p) = 0
disp('System-Massenmatrix M')
M = simplify(jacobian(dTdv,q_p))
disp('System-Vektorfunktion f')
f = simplify(jacobian(dTdv,q)*q_p+dVdq-dTdq-[F;0;0])

%==========================================================================
%---- Linearisierung um die Gleichgewichtslage:
%      th1 = 0, th2 = 0, x = 0

disp(' ')
disp('Elemente der linearisierten Bewegungsgleichung')
disp('System-Massenmatrix M0')
M0 = subs(M,{th1, th2, x},{0, 0, 0})
f0 = subs(f,{x, th1, th2, x_p, ...
    th1_p, th2_p},{0, 0, 0, 0, 0, 0});
disp('Auslenkungs-proportionaler Anteil')
Q = subs(jacobian(f,q),{x, th1, th2, x_p, ...
    th1_p, th2_p},{0, 0, 0, 0, 0, 0})
disp('Steifigkeitsmatrix K')
K = 1/2*(Q+Q.')
```

```
disp('Matrix der nichtkonservativen Kraefte')
N = 1/2*(Q-Q.')
disp('gesschw.-proportionaler Anteil')
P = subs(jacobian(f,q_p),{x, th1, th2, x_p, ...
    th1_p, th2_p},{0, 0, 0, 0, 0, 0})
disp('Daempfungsmatrix')
D = 1/2*(P+P.')
disp('gyroskopischer Anteil')
G = 1/2*(P-P.')
```

4.2 Zustandsraumdarstellung und Reglerentwurf

Um die Auslegung des LQ-Reglers effizient gestalten zu können transformieren wir unser Problem in den Zustandsraum. Wir berechnen zunächst die benötigten Matrizen. Durch ersetzen der symbolischen Variablen durch ihre Zahlenwerte, kann das System numerisch verarbeitet werden. Mit dem Befehl lqr() lassen sich in Matlab aus den systembeschreibenden Matrizen, die Rückkopplungsparameter für eine LQ-Regelung berechnen. Der Befehl ss() konvertiert unser System nach Festlegung der Ein- und Augänge in den Zustandsraum. Schließlich erfolgt die Simulation mit der gewünschten Simulationsdauer und des konstanten Offsets.

```
%----Erstellen und Simulieren der Zustandsraumdarstellung

A = [zeros(3),eye(3);
    -M0^(-1)*Q, -M0^(-1)*P];
A = double(subs(A,{mm, m1, m2, l1, l2, g, I_1, I_2}, ...
    {0.2, 0.01, 0.01, 0.5, 0.7, 9.81, 2.0833e-04, 4.0833e-04})
        );
A(7,7) = 0;
A(7,1) = -1

B = [zeros(3,1);M0^(-1)*[1;0;0]];
B = double(subs(B,{mm, m1, m2, l1, l2, g, I_1, I_2}, ...
    {0.2, 0.01, 0.01, 0.5, 0.7, 9.81, 2.0833e-04, 4.0833e-04})
        );
B(7,1) = 0
Bxc = [0; 0; 0; 0; 0; 0; 1]

C = [1 0 0 0 0 0 0;
   0 1 0 0 0 0 0;
   0 0 1 0 0 0 0]

D = [0; 0; 0]


Q=eye(7);
r=1;

```

```
%----lqr Regelungsentwurf
k = lqr(A,B,Q,r)

%----neue Zustandsraumsystemmatrizen nach Parameterruekfuehrung
Ac = [(A-B*k)];
Bc = [Bxc];
Cc = [C];
Dc = [D];

states = {'x' 'th1' 'th2' 'x_p' 'th1_p' 'th2_p' 'in'};
inputs = {'F'};
outputs = {'x' 'th1' 'th2'};

sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);

%----definieren des Simulationszeitraums
t = 0:0.01:8;

%----definition des konstanten 0.2m offsets als Input
u =0.2*ones(size(t));

%----Simulation des erstellten Systems ueber gegebene Zeit mit bekanntem
%Input
[y,t,x]=lsim(sys_cl,u,t);
```

Die Form der Zustandsraumdartsellung, mit den Matrizen A und B siehe Abbildung(3.2), sieht folgendermaßen aus:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{4.1}$$

Mit den zugehörigen numerischen Werten ergibt sich:

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & -0.9 & 0.1 & 0 & 0 & 0 & 0\\ 0 & 52.8 & -25.5 & 0 & 0 & 0 & 0\\ 0 & -54.1 & 48.1 & 0 & 0 & 0 & 0\\ -1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\4.9\\-12.7\\3.2\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

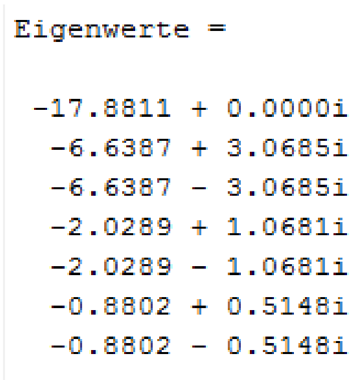
4.3 Simulation und Ausgabe

Um die Ergebnisse besser interpretieren zu können wurden die Position x, sowie die zwei Winkel θ_1 und θ_2 in Abbildung XXXX über die Zeit geplottet. Zudem werden noch die Eigenwerte berechnet und ausgegeben.

```
%----Drei einzelne Diagramme in einem Fenster
figure(1);
ax(1) = subplot(3,1,1);
    plot(ax(1),t,y(:,1),'b');
    title(ax(1),'cart position');     %Titel, Beschriftungen, Kommentare,
    ylim([-0.1,0.25]);                %andere Farben, andere skalierungen,
    grid on                           %da kann man sich noch frei austoben.
ax(2) = subplot(3,1,2);               %relativ einfach verstaendliche
    plot(ax(2),t,y(:,2),'r');         %loesung. Ws nicht Laufzeit optimiert
    title(ax(2),'angle th 1');
    grid on
ax(3) = subplot(3,1,3);
    plot(ax(3),t,y(:,3),'g');
    title(ax(3),'angle th 2');
    grid on

%----Berechnung der Eigenwerte
Eigenwerte = eig(Ac)
```

Wie sich in der Ausgabe erkennen lässt sind die Realteile aller Eigenwerte negativ. Somit ist das betrachtete System stabil!



```
Eigenwerte =

 -17.8811 + 0.0000i
  -6.6387 + 3.0685i
  -6.6387 - 3.0685i
  -2.0289 + 1.0681i
  -2.0289 - 1.0681i
  -0.8802 + 0.5148i
  -0.8802 - 0.5148i
```

Abbildung 4.2: Ausgabe der Eigenwerte

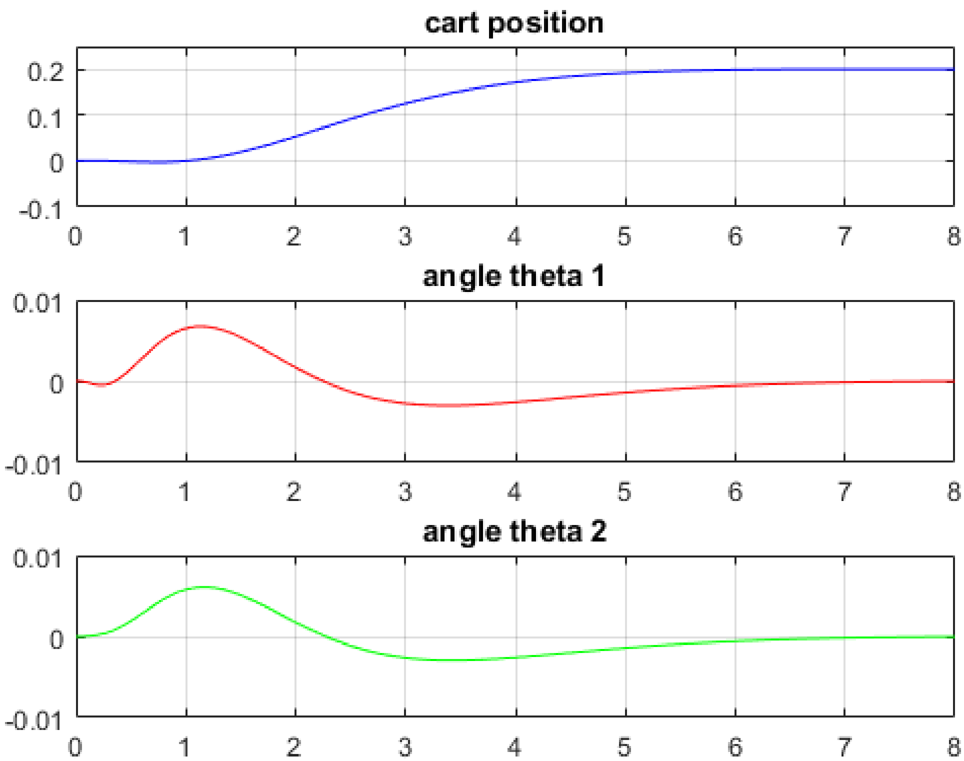


Abbildung 4.3: MATLAB Plot

5

Implementierung in MalpeSim

Nachdem das vorherige Kapitel ausschließlich der Implementierung in MATLAB gewidment wurde, beschäftigt sich dieses Kapitel nun mit der Umsetzung mittels einer *nicht klassischen*, blockorientierten, grafischen Programmierung in MapleSim.
Wie bereits erwähnt funktioniert die Programmierung in MapleSim grafisch. Es wird zuerst das Modell (in unserem Fall das geregelte mechanische Doppelpendel) im MapleSim GUI nachgebildet. Anschließend können Signale direkt an diesem Model abgegriffen und ins System rückgeführt werden. Dadurch lassen sich selbst komplexe dynamische Systeme aus allen Bereichen der Natur- und Ingeneurswissenschaften vergleichsweise einfach modellieren.

5.1 Modellbildung

Da hier keine mathematischen Transformationen mehr nötig waren um das Doppelpendel in MapleSim modellieren zu können wurden die Größen aus der Angabe direkt verwendet. Die Materialparamter sind selbstverständlich die, die auch schon in den anderen Kapiteln verwendet wurden.

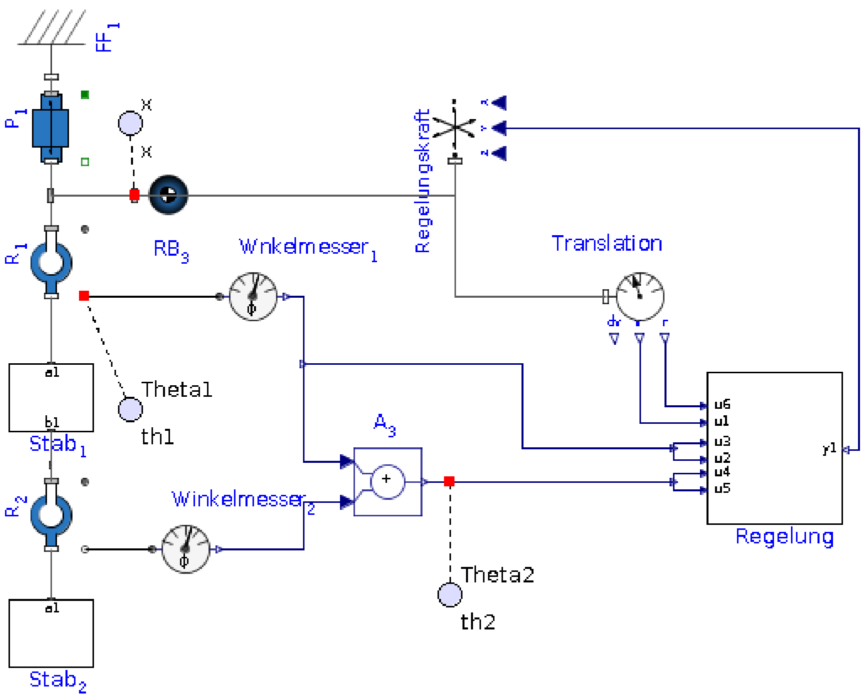


Abbildung 5.1: Modell in MapleSim

Das Pendel wurde aus Komponenten der *Multibody*-Bibliothek aufgebaut. Es besteht aus einem festen Rahmen (*fixed frame*), zwei Drehgelenken (*revolutes*), einem Schlitten (*prismatic*) und den zwei Stäben.
Da innerhalb der Standardbibliotheken keine Kompnenten gab, die die geforderten mechanischen Eigenschaften erfüllten, wurden die Stäbe aus jeweils zwei starren Körpern (*rigid body frames*) und einer Punktmasse (*point mass*) nachgebildet. Die Komponenten der Stäbe wurden aus Gründen der Übersichtlichkeit zu Subsystemen zusammengefasst.

5.2 Regelung

Auch der Bau des Regler ist vergleichweise einfach. Die nötigen Zustandsgrößen θ_1, θ_2 und x können direkt abgegriffen und weiterverwendet werden.
θ_1 wird dabei unverändert dem Regelungssubsystem übergeben. Für θ_2 wird der Winkel, den die beiden Stäbe miteinander einschließen, gemessen und zu θ_1 addiert. Der Abstand des Schlittens vom Koordinatenursprung x kann auch direkt ausgelesen und weitegeben werden. Wie schon bei den Stäben wurde die Regelung zu einem Subsystem zusammengefasst um eine hohe Übersicht gewährleisten zu können. Dieser Regelung möchten wir uns nun widmen.

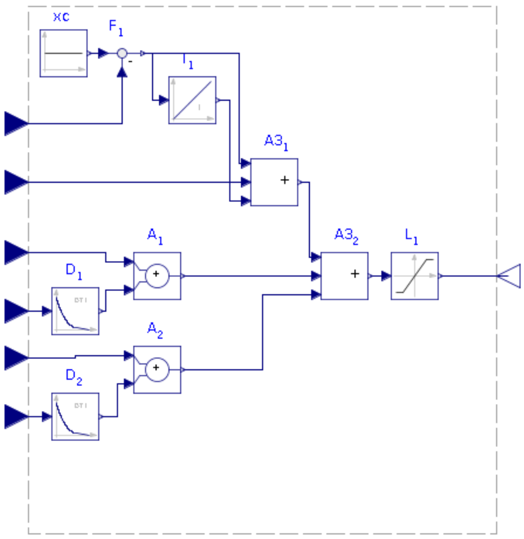


Abbildung 5.2: Subsystem Regelung

Wie der Angabe zu entnehmen ist, bedarf es einer Regelung um das instabile System im Gleichgewicht zu halten. Dabei wird die Kraft als Regelung in Form einer Zustandsrückführung, kombiniert mit einer Positionsregelung der Form

$$e = x_e - x \qquad \dot{e} = -\dot{x} \qquad i_{in} = \int_{t_0}^{t_{end}} edt, \tag{5.1}$$

angesetzt:

$$F = k_x e + k_{\dot{x}}\dot{x} + k_{in}i_{in} + k_{\theta_1}\theta_1 + k_{\theta_2}\theta_2 + k_{\dot{\theta}_1}\dot{\theta}_1 + k_{\dot{\theta}_2}\dot{\theta}_2 \tag{5.2}$$

Nach Gleichung 5.2 ist die Rückstellkraft also eine Linearkombination der Winkel und ihren zeitlichen Ableitungen, sowie einer momentanen Auslenkungsdifferenz des Schlittens vom Sollwerd, mit der zugehörigen zeitlichen Ableitung und dem zeitlichen Integral [über besagte Auslenkung].
Wie sich in Abbildung 5.1 erkennen lässt werden die Winkel jeweils doppelt übergeben. Intern wird dann jeweils ein Signal dieser Signalpaare differenziert. Anschließend werden die Signale mit den zugehörigen Rückkopplungparametern k_{θ_1} und k_{θ_2} beziehungsweise $k_{\dot{\theta}_1}$ und $k_{\dot{\theta}_2}$ in den Addierern A_1 und A_2 aufsummiert.
Ein ähnliches Prodzedere ergibt sich auch für die anderen beiden Eingänge x und $\dot{x}$. Mit der Auslenkung x wird nach den Gleichungen 5.1 die Abweichung e vom Sollzustand x_e berechnet, welche anschließend im Integrationsglied I_1 integriert wird. Auch hier werden die Paramter wieder mit den zugehörigen Koeffizienten k_x, $k_{\dot{x}}$ und k_{in} gewichtet und anschließend in den Addieren $A3_1$ und $A3_2$ summiert.
Da die Rückstellkraft nicht unendlich groß werden darf wurde abschließend noch eine Kraftbegrenzung L_1 hinzugefügt, welche die Rückstellkraft F auf maximal 10 [N] beschränkt.
Damit ist die Regelung abgeschlossen und die Kraft wird am Ausgang $y1$ (Sie-

he Abbildung 5.1) ausgegeben und mit dem Element *applied world force* ins mechanische System rückgeführt.