

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

1	Vorwort	4
2	Aufgabenstellung	5
3	Modellbildung	6
4	Implementierung in MATLAB	8
4.1	Variablendefinition und Modellbildung	8
4.2	Zustandsraumdarstellung und Reglerentwurf	11
4.3	Simulation und Ausgabe	12
5	Implementierung in MalpeSim	15
5.1	Modellbildung	15
5.2	Regelung	16
6	Zusammenfassung	19

Abbildungsverzeichnis

3.1	Mechanisches Modell eines stehenden Doppelpendels	6
4.1	Ortsvektoren zu den Gelenk- und Schwerpunkten	9
4.2	Ausgabe der Eigenwerte	13
4.3	MATLAB Plot	14
5.1	Modell in MapleSim	16
5.2	Subsystem Regelung	17

1

Vorwort

Sehr geehrte Damen und Herren, liebe Leser und Leserinnen!

Das vorliegende Protokoll wurde im Rahmen der Vorlesung und Übung *Kontinuierliche Simulation (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 textueller Programmierung und grafischer, blockorientierter Modellierung gezogen werden. Betreut wurde das Projekt der Gruppe 47 von Fabian Germ.

Viel Spaß beim Lesen!

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.

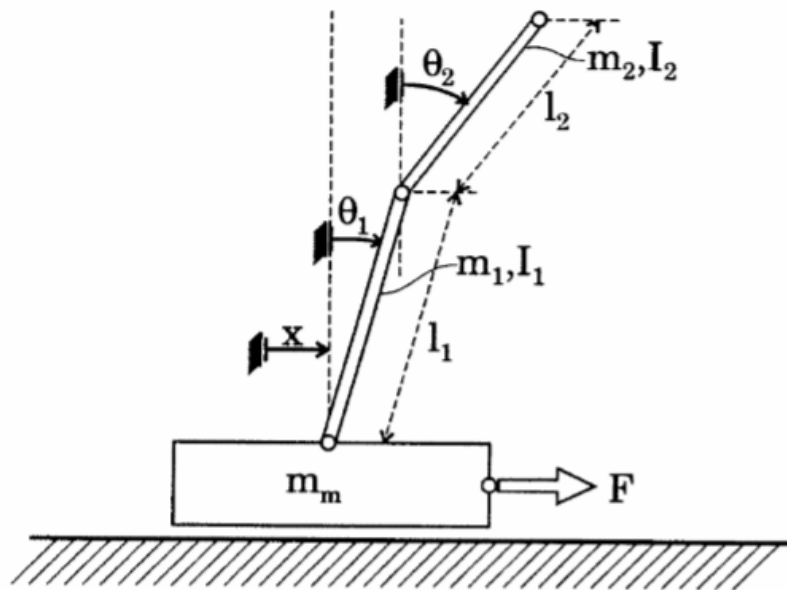


Abbildung 3.1: Mechanisches Modell eines stehenden Doppelpendels

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

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

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

$$T = T_{trans} + T_{rot} \quad (3.2)$$

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

$$T_{trans} = \frac{1}{2}m\vec{v}^2 \quad (3.3)$$

$$\vec{v} = J_v \dot{\vec{q}} \quad (3.4)$$

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.

$$T_{rot} = \frac{1}{2}I_s\vec{\omega}^2 \quad (3.5)$$

Um die Energien in die Lagrange Gleichung 2.Art einsetzen zu können müssen sie partiell Abgeleitet werden (Siehe Gleichung 3.1). Dies geschieht wiederum mit einer Jacobi-Matrix.

Damit erhalten wir schließlich die Bewegungsgleichungen in folgender Form:

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

Die Ruhelage des Systems finden wir, indem man zuerst die Ableitungen der Lagekoordinaten θ_1, θ_2 und x nullsetzt. Anschließend linearisieren wir mithilfe einer Taylorreihenentwicklung um die Ruhelage mit Vernachlässigung aller nichtlinearer Glieder.

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 Variablendefinition 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.

```
1 %---- Ermitteln der Bewegungsgleichungen
2 %      definieren der Systemvariablen
3 syms l1 l2 th1 th2 th1_p th2_p th1_pp th2_pp
4 syms x x_p x_pp mm m1 m2 g l_1 l_2 F xc
5
6 frg=3;                %Anzahl der Freiheitsgrade
7 n=3;                  %Anzahl der Koerper
8
9 q=[x ; th1 ; th2];    %Minimalkoordinaten
10 q_p=[x_p ; th1_p ; th2_p]; %zeitliche Ableitungen
11 q_pp=[x_pp ; th1_pp ; th2_pp];
```


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

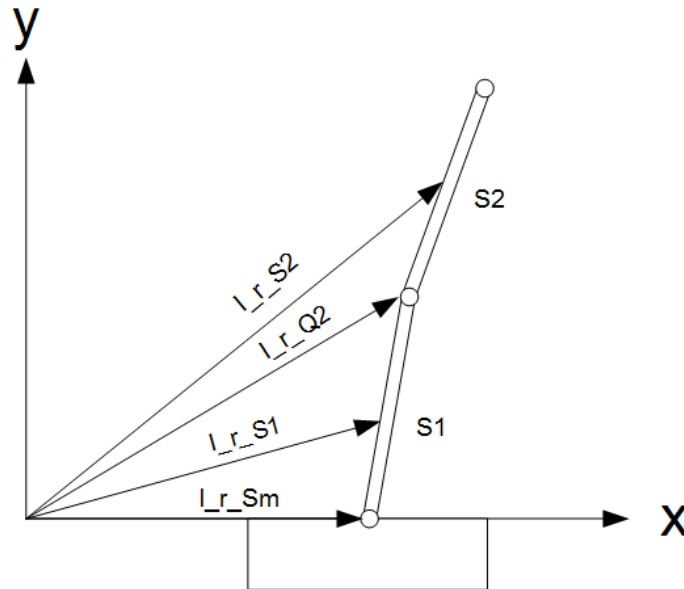


Abbildung 4.1: Ortsvektoren zu den Gelenk- und Schwerpunkten

```

1 %---- Drehmatrix Stab 1
2 T_IK1 = [cos(th1) sin(th1) 0;
3          -sin(th1) cos(th1) 0;
4           0          0      1];
5 %---- Drehmatrix Stab 2
6 T_IK2 = [cos(th2) sin(th2) 0;
7          -sin(th2) cos(th2) 0;
8           0          0      1];
9
10 %---- Ortsvektoren
11 l_r_Sm = [x;0;0];
12 l_r_S1 = [x+l1/2*sin(th1) ; l1/2*cos(th1) ; 0];
13 l_r_Q2 = [x+l1*sin(th1) ; l1*cos(th1) ; 0];
14 K1_r_Q1S1 = [0; l1/2; 0];
15 K2_r_Q2S2 = [0; l2/2; 0];
16 l_r_S2 = l_r_Q2 + T_IK2 * K2_r_Q2S2;
17
18 %---- Traegheitstensoren in den koerperfesten
19      Koordinatensystemen
20 K1_I_S1 = diag([0 0 I_1]);
21 K2_I_S2 = diag([0 0 I_2]);
22
23 %---- Winkelgeschwindigkeitsvektoren der Staebe
24 K_om1 = [0 ; 0 ; -th1_p];

```

```

24 K_om2 = [0 ; 0 ; -th2_p];
25
26 %---- JACOBI-Matrizen der Translation
27 J_Tm = jacobian(I_r_Sm, q);
28 J_T1 = jacobian(I_r_S1, q);
29 J_T2 = jacobian(I_r_S2, q);
30
31 %---- JACOBI-Matrizen der Rotation
32 J_R1 = jacobian(K_om1, q_p);
33 J_R2 = jacobian(K_om2, q_p);
34
35 %---- Geschwindigkeitsvektoren
36 I_v_Sm = J_Tm*q_p ;
37 I_v_S1 = J_T1*q_p ;
38 I_v_S2 = J_T2*q_p ;
39
40 %---- kinetische Energie
41 T = 1/2*(mm*(I_v_Sm.'*I_v_Sm)+m1*(I_v_S1.'*I_v_S1)
42       +m2*(I_v_S2.'*I_v_S2)+K_om1.'*K1_I_S1*K_om1
43       +K_om2.'*K2_I_S2*K_om2);
44 T = simplify(T);
45
46 %---- potentielle Energie
47 V=-(m1*I_r_S1.'+m2*I_r_S2.')*[0 ; -g ; 0];
48
49 %---- Ableitungen fuer LAGRANGEsche Gleichung 2. Art
50 dTdv = simplify(jacobian(T,q_p).');
51 dTdq = simplify(jacobian(T,q).');
52 dVdq = simplify(jacobian(V,q).');
53
54 %---- Elemente der Bewegungsgleichung  $M(q)*q_{pp} + f(q,q_p) = 0$ 
55 M = simplify(jacobian(dTdv,q_p));
56 f = simplify(jacobian(dTdv,q)*q_p+dVdq-dTdq-[F;0;0]);
57
58 %=====
59 %---- Linearisierung um die Gleichgewichtslage:
60 %      th1 = 0, th2 = 0, x = 0
61 M0 = subs(M,{th1, th2, x},{0, 0, 0});
62 f0 = subs(f,{x, th1, th2, x_p, ...
63          th1_p, th2_p},{0, 0, 0, 0, 0, 0});
64
65 %Auslenkungs-proportionaler Anteil
66 Q = subs(jacobian(f,q),{x, th1, th2, x_p, ...
67          th1_p, th2_p},{0, 0, 0, 0, 0, 0});
68
69 %Geschwindigkeits-proportionaler Anteil
70 P = subs(jacobian(f,q_p),{x, th1, th2, x_p, ...
71          th1_p, th2_p},{0, 0, 0, 0, 0, 0});

```

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 Ausgänge in den Zustandsraum. Schließlich erfolgt die Simulation mit der gewünschten Simulationsdauer und des konstanten Offsets.

```

1 %---- Erstellen und Simulieren der Zustandsraumdarstellung
2
3 A = [ zeros(3), eye(3);
4       -M0^(-1)*Q, -M0^(-1)*P ];
5 A = double(subs(A,{mm, m1, m2, l1, l2, g, I_1, I_2}, ...
6             {0.2, 0.01, 0.01, 0.5, 0.7, 9.81, 2.0833e-04, 4.0833e-04}));
7 A(7,7) = 0;
8 A(7,1) = -1
9
10 B = [ zeros(3,1); M0^(-1) * [1;0;0] ];
11 B = double(subs(B,{mm, m1, m2, l1, l2, g, I_1, I_2}, ...
12             {0.2, 0.01, 0.01, 0.5, 0.7, 9.81, 2.0833e-04, 4.0833e-04}));
13 B(7,1) = 0
14 Bxc = [0; 0; 0; 0; 0; 0; 0; 1]
15
16 C = [1 0 0 0 0 0 0;
17       0 1 0 0 0 0 0;
18       0 0 1 0 0 0 0]
19
20 D = [0; 0; 0]
21
22 %Gewichtungsmatrix und Gewichtungsfaktor
23 Q=eye(7);
24 r=1;
25
26 %---- lqr Regelungsentwurf
27 k = lqr(A,B,Q,r)
28
29 %---- neue Zustandsraumsystemmatrizen nach
    Parameterruekfuehrung
30 Ac = [(A-B*k)];
31 Bc = [Bxc];
32 Cc = [C];
33 Dc = [D];
34
35 states = {'x' 'th1' 'th2' 'x_p' 'th1_p' 'th2_p' 'in'};
36 inputs = {'F'};
37 outputs = {'x' 'th1' 'th2'};

```

```

38 sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,
39             'outputname',outputs);
40
41 %----definieren des Simulationszeitraums
42 t = 0:0.01:8;
43
44 %----definition des konstanten 0.2m offsets als Input
45 u =0.2*ones(size(t));
46
47 %----Simulation des erstellten Systems ueber gegebene Zeit mit
    bekanntem
48 %Input
49 [y,t,x]=lsim(sys_cl,u,t);

```

Die Form der Zustandsraumdarstellung, mit den Matrizen A_c und B_c siehe Abbildung(3.2), sieht folgendermaßen aus:

$$\dot{\mathbf{x}} = A_c \mathbf{x} + B_c u \quad (4.1)$$

Mit den zugehörigen numerischen Werten ergibt sich:

$$\begin{bmatrix} \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{x} \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \dot{i}_n \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -13.65 & 161.07 & -255.27 & -16.43 & -1.9 & -41.53 & 4.93 \\ 35.10 & -363.74 & 631.19 & 42.26 & 4.88 & 106.79 & -12.68 \\ -8.36 & 44.56 & -108.24 & -10.06 & -1.16 & -25.43 & 3.02 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ e \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} F$$

$$\begin{bmatrix} x \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ e \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} F$$

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 4.3 über die Zeit geplottet. Zudem werden noch die Eigenwerte berechnet und ausgegeben.

```

1 %----Drei einzelne Diagramme in einem Fenster
2 figure(1);
3 ax(1) = subplot(3,1,1);

```

```

4     plot(ax(1),t,y(:,1),'b');
5     title(ax(1),'cart position');
6     ylim([-0.1,0.25]);
7     grid on
8 ax(2) = subplot(3,1,2);
9     plot(ax(2),t,y(:,2),'r');
10    title(ax(2),'angle th 1');
11    grid on
12 ax(3) = subplot(3,1,3);
13    plot(ax(3),t,y(:,3),'g');
14    title(ax(3),'angle th 2');
15    grid on
16
17 %---Berechnung der Eigenwerte
18 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

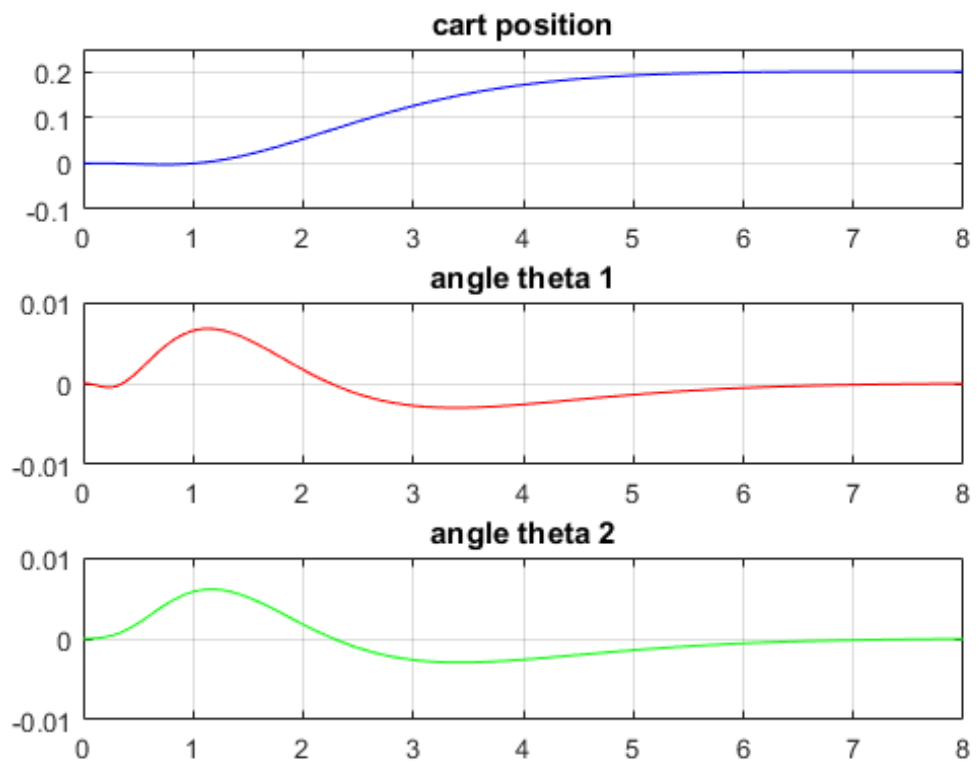


Abbildung 4.3: MATLAB Plot

5

Implementierung in MalpeSim

Nachdem das vorherige Kapitel ausschließlich der Implementierung in MATLAB gewidmet wurde, beschäftigt sich dieses 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 Ingenieurwissenschaften 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 Materialparameter sind selbstverständlich die, die auch schon in den anderen Kapiteln verwendet wurden.

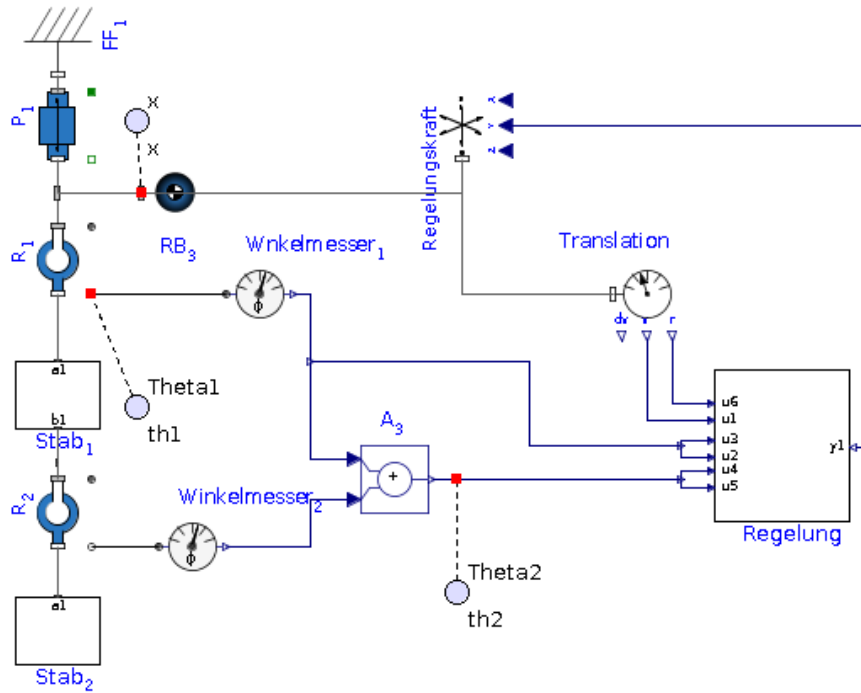


Abbildung 5.1: Modell in MapleSim

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 Komponenten 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 vergleichsweise einfach. Die nötigen Zustandsgrößen θ_1 , θ_2 und x können direkt abgegriffen und weiterverwendet werden.

θ_1 wird dabei unverändert dem Regelungssystem 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 weitergegeben 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.

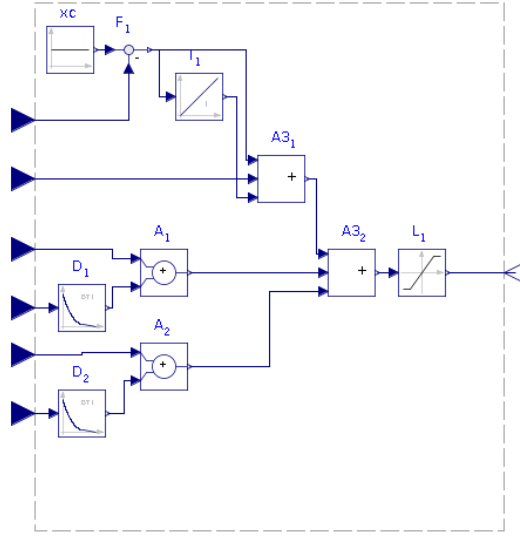


Abbildung 5.2: Subsystem Regelung

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

$$e = x_e - x \quad \dot{e} = -\dot{x} \quad i_{in} = \int_{t_0}^{t_{end}} e dt, \quad (5.1)$$

angesetzt:

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

Nach Gleichung 5.2 ist die Rückstellkraft also eine Linearkombination der Winkel und ihren zeitlichen Ableitungen, sowie einer momentanen Auslenkungsdifferenz des Schlittens vom Sollwert, 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 Signalkopie differenziert. Anschließend werden die Signale mit den zugehörigen Rückkopplungsparametern 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 Prozedere 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 Parameter wieder mit den zugehörigen Koeffizienten k_x , $k_{\dot{x}}$ und k_{in} gewichtet und anschließend in den Addierern $A3_1$ 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 y_1 (Sie-

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

6

Zusammenfassung