

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

Contents

Vorwort	4
Aufgabenstellung	5
Modellbildung	6
Implementierung in MATLAB	8
3.1 Variablendefinition und Modellbildung	8
3.2 Zustandsraumdarstellung und Reglerentwurf	10
3.3 Simulation und Ausgabe	12
Implementierung in MalpeSim	14
4.1 Modellbildung	14
4.2 Regelung	15

List of Figures

2.1	Mechanisches Modell eines stehenden Doppelpendels	6
3.1	MATLAB Plot	13

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 wünschen

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 Simulationsslauf 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.

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.

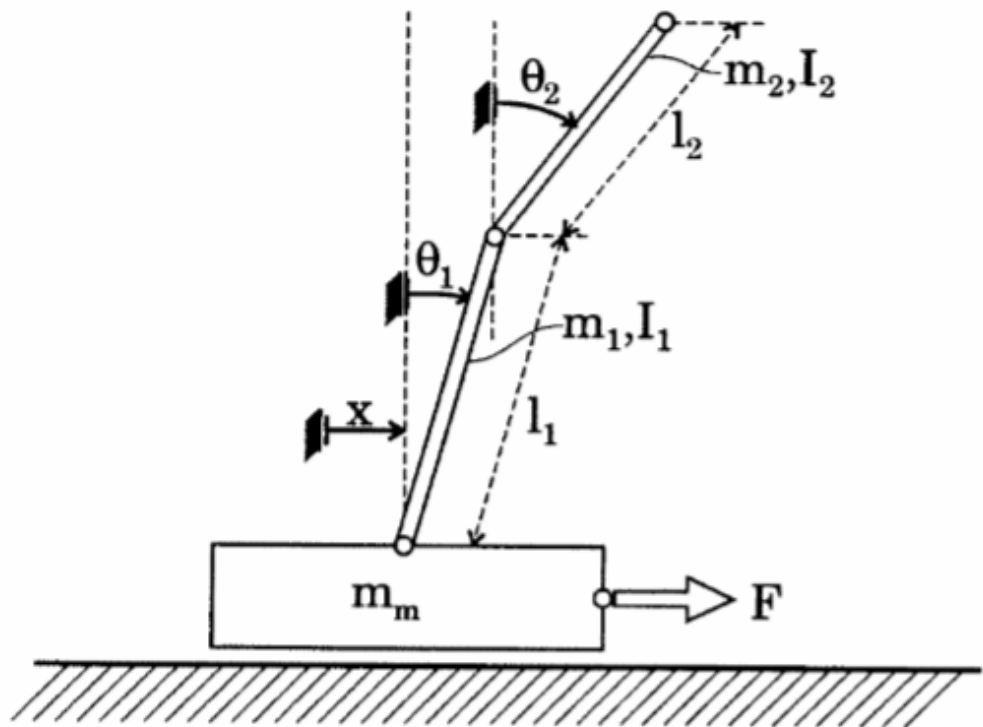


Figure 2.1: Mechanisches Modell eines stehenden Doppelpendels

Da wir bei der Berechnung der Matrizen, welche für eine Zustandsraum-

darstellung erforderlich 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 Lagrange Gleichung 2.Art.

$$\frac{d}{dt}\left(\frac{\delta T}{\delta \dot{q}_i}\right) - \frac{\delta T}{\delta q_i} + \frac{\delta V}{\delta q_i} = 0 \quad (2.1)$$

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

$$T = T_{trans} + T_{rot} \quad (2.2)$$

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

$$T_{trans} = \frac{1}{2} m \vec{v}^2 \quad (2.3)$$

$$\vec{v} = Jv * \dot{q} \quad (2.4)$$

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.

$$T_{rot} = \frac{1}{2} I_s \vec{\omega}^2 \quad (2.5)$$

Um die Energien in die Lagrange Gleichung 2.Art einsetzen zu können müssen sie partiell Abgeleitet werden (Siehe Gleichung 2.1). Dies geschieht wiederum mit einer Jacobi-Matrix.

Damit erhalten wir schließlich die Bewegungsgleichungen in folgender Form:

$$M(q)\ddot{q} + f(q, \dot{q}) = 0 \quad (2.6)$$

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

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.

3.1 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 %Definition der Systemvariablen
2 syms l1 l2 phi_1 phi_2 phi_p1 phi_p2 phi_pp1 phi_pp2
3 syms a a_p a_pp mm m1 m2 g I_1 I_2 F xc
4
5 frg=3; %Anzahl der
        Freiheitsgrade
6 n=3; %Anzahl der Koerper
7
8 q=[a ; phi_1 ; phi_2]; %Minimalkoordinaten
9 q_p=[a_p ; phi_p1 ; phi_p2]; %zeitliche
        Ableitungen
10 q_pp=[a_pp ; phi_pp1 ; phi_pp2];
```


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

```

1 %---- Drehmatrix Stab 1
2 T_IK1 = [cos(phi_1) sin(phi_1) 0;
3          -sin(phi_1) cos(phi_1) 0;
4           0          0        1];
5 %---- Drehmatrix Stab 2
6 T_IK2 = [cos(phi_2) sin(phi_2) 0;
7          -sin(phi_2) cos(phi_2) 0;
8           0          0        1];
9
10 %---- Ortsvektoren
11 I_r_Sm = [a;0;0];
12 I_r_S1 = [a+l1/2*sin(phi_1) ; l1/2*cos(phi_1) ; 0];
13 I_r_Q2 = [a+l1*sin(phi_1) ; l1*cos(phi_1) ; 0];
14 K1_r_Q1S1 = [0; l1/2; 0];
15 K2_r_Q2S2 = [0; l2/2; 0];
16 I_r_S2 = I_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 ; -phi_p1];
25 K_om2 = [0 ; 0 ; -phi_p2];
26
27 %---- JACOBI-Matrizen der Translation
28 J_Tm = jacobian(I_r_Sm, q);
29 J_T1 = jacobian(I_r_S1, q);
30 J_T2 = jacobian(I_r_S2, q);
31
32 %---- JACOBI-Matrizen der Rotation
33 J_R1 = jacobian(K_om1, q-p);
34 J_R2 = jacobian(K_om2, q-p);
35
36 %---- Geschwindigkeitsvektoren
37 I_v_Sm = J_Tm*q-p ;
38 I_v_S1 = J_T1*q-p ;
39 I_v_S2 = J_T2*q-p ;
40
41 %---- kinetische Energie
42 T = 1/2*(mm*(I_v_Sm.'*I_v_Sm)+m1*(I_v_S1.'*I_v_S1)+m2*(I_v_S2
43      .'*I_v_S2) ...%Translation
44      +K_om1.'*K1_I_S1*K_om1+K_om2.'*K2_I_S2*K_om2); %
45      Rotation
46 T = simplify(T); %
47      Vereinfachung
48
49 %---- potentielle Energie
50 V=-(m1*I_r_S1.'+m2*I_r_S2.')*[0 ; -g ; 0];

```

Schließlich werden nach Gleichung (2.1) die Bewegungsgleichungen berechnet und um die Ruhelage linearisiert. Außerdem werden noch die, für eine Zustandsraumdarstellung erforderlichen Matrizen berechnet und ausgegeben.

```

1 %---- Ableitungen fuer LAGRANGEsche Gleichung 2. Art
2 dTdv = simplify(jacobian(T,q-p).');           %mit transponieren
        zu Spaltenvektor gemacht
3 dTdq = simplify(jacobian(T,q).');
4 dVdq = simplify(jacobian(V,q).');
5
6 %---- Elemente der Bewegungsgleichung  $M(q)*\ddot{q} + f(q,\dot{q}) = 0$ 
7 disp('System-Massenmatrix M')
8 M = simplify(jacobian(dTdv,q-p))
9 disp('System-Vektorfunktion f')
10 f = simplify(jacobian(dTdv,q)*q-p+dVdq-dTdq-[F;0;0])
11
12 %=====
13 %---- Linearisierung um die Gleichgewichtslage:
14 %      phi_1 = 0, phi_2 = 0, a = 0
15
16 disp(' ')
17 disp('Elemente der linearisierten Bewegungsgleichung')
18 disp('System-Massenmatrix M0')
19 M0 = subs(M,{phi_1, phi_2, a},{0, 0, 0})
20 f0 = subs(f,{a, phi_1, phi_2, a_p, ...
21          phi_p1, phi_p2},{0, 0, 0, 0, 0, 0});
22 disp('Auslenkungs-proportionaler Anteil')
23 Q = subs(jacobian(f,q),{a, phi_1, phi_2, a_p, ...
24          phi_p1, phi_p2},{0, 0, 0, 0, 0, 0})
25 disp('Steifigkeitsmatrix K')
26 K = 1/2*(Q+Q.')
27 disp('Matrix der nichtkonservativen Kraefte')
28 N = 1/2*(Q-Q.')
29 disp('gesschw.-proportionaler Anteil')
30 P = subs(jacobian(f,q-p),{a, phi_1, phi_2, a_p, ...
31          phi_p1, phi_p2},{0, 0, 0, 0, 0, 0})
32 disp('Daempfungsmatrix')
33 D = 1/2*(P+P.')
34 disp('gyroskopischer Anteil')
35 G = 1/2*(P-P.')

```

3.2 Zustandsraumdarstellung und Reglerentwurf

Um die Auslegung des LQ-Reglers effizient gestalten zu können transformieren wir unser Problem in den Zustandsraum und berechnen die Regelungskonstante.

EKLÄRUNG Zustandsraum LQR parameterrueckfuehrung

XXXXXX
XXXXXX

```

1 %=====
2 %---- Erstellen und Simulieren der Zustandsraumdarstellung
3 syms x th1 th2 x_p th1_p th2_p
4 syms x_pp th1_pp th2_pp
5
6 y = [q.', q-p.'].';
7 y-p = [q-p.', x_pp , th1_pp , th2_pp].';
8
9 A = [zeros(3), eye(3);
10      -M0^(-1)*Q, -M0^(-1)*P];
11 A = double(subs(A, {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 );
14 A(7,7) = 0;
15 A(7,1) = -1
16
17 B = [zeros(3,1); M0^(-1)*[1;0;0]];
18 B = double(subs(B, {mm, m1, m2, l1, l2, g, I_1, I_2}, ...
19               {0.2, 0.01, 0.01, 0.5, 0.7, 9.81, 2.0833e-04, 4.0833e-04}))
20 );
21 B(7,1) = 0
22 Bxc = [0; 0; 0; 0; 0; 0; 0; 1]
23
24 C = [1 0 0 0 0 0 0;
25       0 1 0 0 0 0 0;
26       0 0 1 0 0 0 0]
27
28 D = [0; 0; 0]
29
30 Q=eye(7);
31 r=1;
32 %----lqr Regelungsentwurf
33 k = lqr(A,B,Q,r)
34
35 %----neue Zustandsraumsystemmatrizen nach
36 Parameterruekfuehrung
37 Ac = [(A-B*k)];
38 Bc = [Bxc];
39 Cc = [C];
40 Dc = [D];
41
42 states = {'x' 'th1' 'th2' 'x_p' 'th1_p' 'th2_p' 'in'};
43 inputs = {'F'};
44 outputs = {'x' 'th1' 'th2'};
45 sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,
46             'outputname',outputs);

```

```

46 %----definieren des Simulationszeitraums
47 t = 0:0.01:8;
48
49 %----definition des konstanten 0.2m offsets als Input
50 u = 0.2*ones(size(t));
51
52 %----Simulation des erstellten Systems ueber gegebene Zeit mit
    bekanntem
53 %Input
54 [y,t,x]=lsim(sys_cl,u,t);
55

```

3.3 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.

```

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'); %Titel, Beschriftungen,
        Kommentare,
6     ylim([-0.1,0.25]); %andere Farben, andere
        skalierungen,
7     grid on %da kann man sich noch
        frei austoben.
8 ax(2) = subplot(3,1,2); %relativ einfach
        verstaendliche
9     plot(ax(2),t,y(:,2),'r'); %loesung. Ws nicht
        Laufzeit optimiert
10    title(ax(2),'angle theta 1');
11    grid on
12 ax(3) = subplot(3,1,3);
13     plot(ax(3),t,y(:,3),'g');
14     title(ax(3),'angle theta 2');
15     grid on
16
17 %----Berechnung der Eigenwerte
18 Eigenwerte = eig(Ac)
19 disp('Das System ist stabil, da der Realteil aller Eigenwerte
20     negativ ist!')

```

Wie sich im MATLAB Code schon erkennen lässt sind die Realteile aller Eigenwerte negativ. Somit ist das betrachtete System stabil!

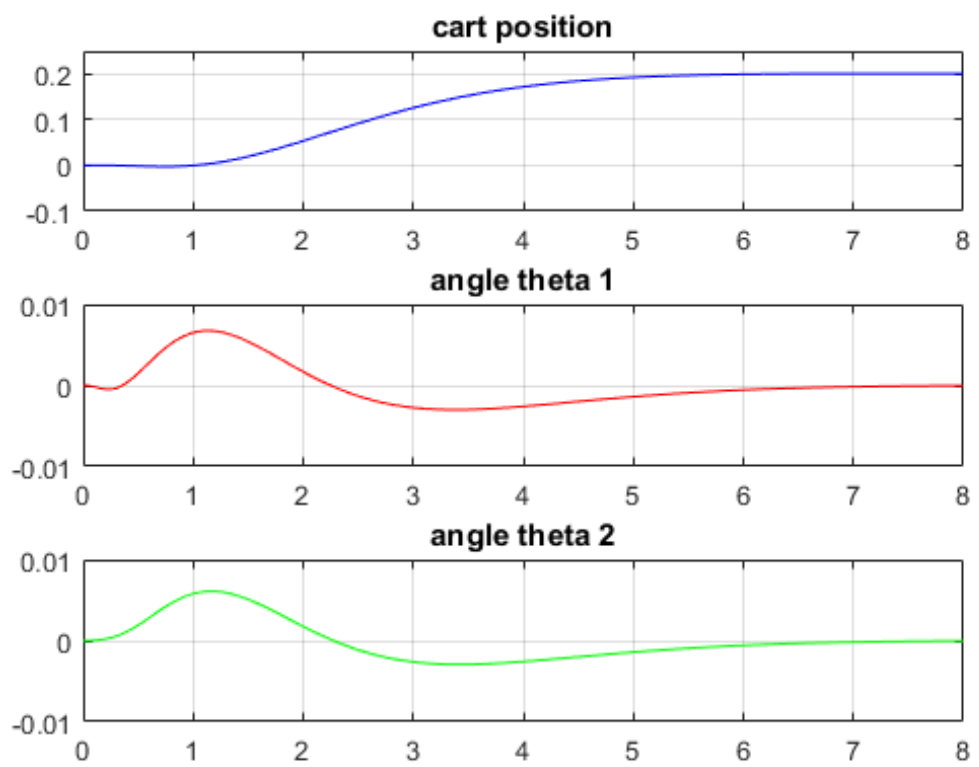


Figure 3.1: MATLAB Plot

Implementierung in MapleSim

Nachdem das vorherige Kapitel ausschließlich der Implementierung in MATLAB gewidmet 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 Ingenieurwissenschaften vergleichsweise einfach modellieren.

4.1 Modellbildung

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

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 (Masserverteilung und somit Schwerpunkt und Trägheitstensor) 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.

abbildung mech model MapleSim x

x
x
x
x
x
x
x

x

eventuel Abbildung subsystem stab

4.2 Regelung

Auch der Bau des Regler ist vergleichsweise einfach. Die nötigen (Zustands-)Größen θ_1 , θ_2 und x können direkt ausgelesen, abgegriffen und weiterverwendet werden.

θ_1 kann direkt ausgelesen werden und wird 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 subsystem regelung x

x
x
x
x
x
x
x

Die Rückstellkraft ...

eingänge u1 - u6 und ausgang y labeln?
simulation?
eigenwertberechnung?

abschließende worte? (vergleich, pros-cons, erfahrung, etc.)