Versuch I, Modellbildung und Simulation eines Doppelpendel-Systems

Andreas Jentsch, Ali Kerem Sacakli

Praktikumsbericht – Praktikum Matlab/Simulink II

TECHNISCHE UNIVERSITÄT DARMSTADT

REGELUNGSTECHNIK UND MECHATRONIK rtm

1.1 Modellbildung

Im Praktikum Matlab/Simulink II werden weiterführende Konzepte der Regelungstechnik basierend auf einem Doppelpendel-System erarbeitet. In diesem Versuch geht es um die Modellierung des Doppelpendelsystems durch verschiedene Modellierungsmöglichkeiten.

Der erarbeite Code für die symbolische Lösung der Bewegungsgleichungen für das Doppelpendel-System sieht wie folgt aus:

Listing 1.1: Code des Doppelpendel-Modells

```
clear all
clc
%% Variablen
% Parameter
syms m1 l1 J1 m2 l2 J2 Rp1 Rp2 g t positive;

% Koordinaten
syms phi1 dphi1 ddphi1 phi2 dphi2 ddphi2 real;

% Momente
syms M real;


%% Zu ersetzende Größen

% Reibungen
MR1 = Rp1 * dphi1;
MR2 = Rp2 * (dphi2 - dphi1);

% Trägheintsmomente
J1 = 1 / 12 * m1 * l1^2;
J2 = 1 / 12 * m2 * l2^2;

% Pendelkoordinaten
x1 = l1 / 2 * sin(phi1);
dx1 = l1 / 2 * dphi1 * cos(phi1);

y1 = -l1 / 2 * cos(phi1);
dy1 = l1 / 2 * dphi1 * sin(phi1);

x2 =  2*x1 + l2 / 2 * sin(phi2);
```

```
dx2 = 2*dx1 + l2 / 2 * dphi2 * cos(phi2);

y2 = 2*y1 - l2 / 2 * cos(phi2);
dy2 = 2*dy1 + l2 / 2 * dphi2 * sin(phi2);

%% Mechanik

T1 = 1/2 * m1 * (dx1^2 + dy1^2) + 1/2 * J1 * dphi1^2;
T2 = 1/2 * m2 * (dx2^2 + dy2^2) + 1/2 * J2 * dphi2^2;

T = T1 + T2;

U = g * (m1 * y1 + m2 * y2);

%Generalisierte Kräfte

Qphi1 = M - MR1;
Qphi2 = -MR2;

%Lagrangegleichung

L = T-U;

%% Herleitung der Ableitung nach generalisierter Koordinate

% dL/dphi1
L_phi1 = jacobian(L, phi1);

% dL/ddphi1
L_dphi1 = jacobian(L, dphi1);

% dL/dhpi2
L_phi2 = jacobian(L, phi2);

% dL/ddphi2
L_dphi2 = jacobian(L, dphi2);

% d(L_dphi1)/dt und d(L_dphi2)/dt

% Variablen ohne t durch Variablen mit t ersetzen
```

```
L_dphi1_t = subs(L_dphi1,{phi1,dphi1,phi2,dphi2},...
    {'phi1(t)', 'dphi1(t)', 'phi2(t)', 'dphi2(t)'});

L_dphi2_t = subs(L_dphi2,{phi1,dphi1,phi2,dphi2},...
    {'phi1(t)', 'dphi1(t)', 'phi2(t)', 'dphi2(t)'});

%Berechnung der Zeitableitung
dL_dphi1_t = diff(L_dphi1_t, t);
dL_dphi2_t = diff(L_dphi2_t, t);

%Variablen mit t
Var_t = {'phi1(t)', 'dphi1(t)', 'diff(phi1(t),t)', 'diff(dphi1(t),t)',...
        'phi2(t)', 'dphi2(t)', 'diff(phi2(t),t)', 'diff(dphi2(t),t)'};

%Variablen ohne t
Var_ot = {phi1, dphi1, dphi1, ddphi1, phi2, dphi2, dphi2, ddphi2};

dL_dphi1_t = subs(dL_dphi1_t, Var_t, Var_ot);
dL_dphi2_t = subs(dL_dphi2_t, Var_t, Var_ot);


%% Berechnung der LAGRANGEschen Gleichungen

Sol = solve([dL_dphi1_t - L_phi1 == Qphi1, dL_dphi2_t - L_phi2 == Qphi2],...
         [ddphi1, ddphi2]);
Sol.ddphi1=simplify(Sol.ddphi1);
Sol.ddphi2=simplify(Sol.ddphi2);
```

1.2 Simulation

Das im vorherigen Kapitel dargestellte Modell wird mit folgenden drei Modellierungsmöglichkeiten implementiert:

- Fcn-Blocks,
- MATLAB-Function-Blocks
- einer M-File S-Function

Im folgenden Abschnitt wird für jede Implementierung das Simulink-Schaltbild mit dem zugehörigen Code zur Simulation vorgestellt.

1.2.1 Fcn-Blocks

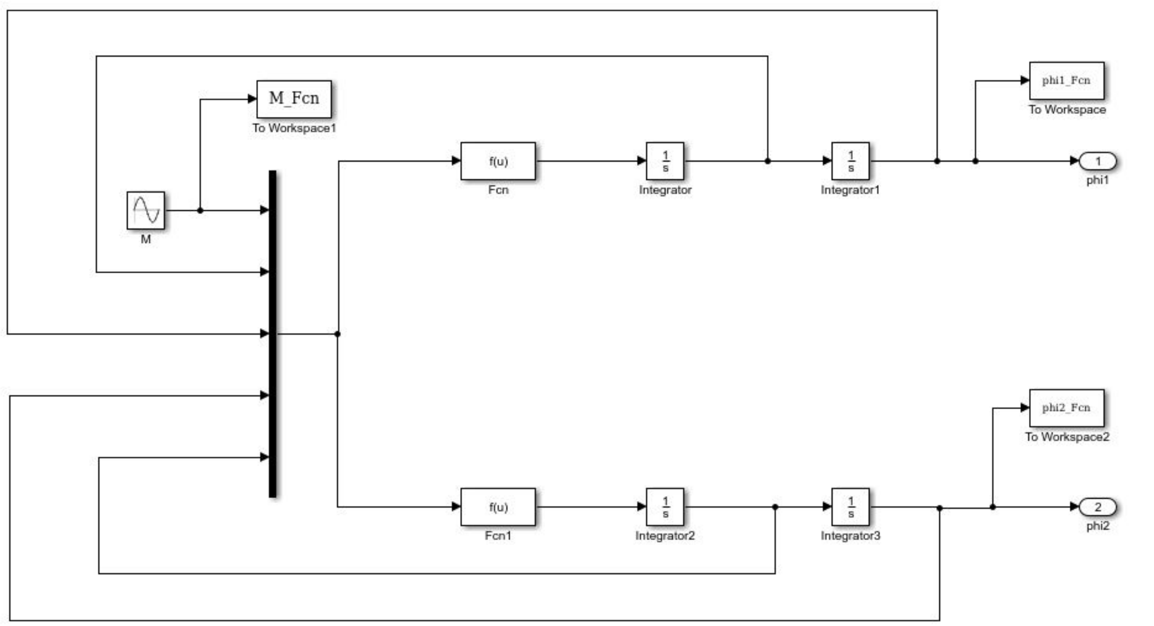

Figure 1.1: Implementierung mit Fcn-Blöcken

Dabei lautet die Funktion von **Fcn**:

$-(6\ast(4\ast Rp1\ast u(2)\ast l2-4\ast u(1)\ast l2+6\ast Rp2\ast u(2)\ast l1\ast cos(u(3)-u(4))-6\ast Rp2\ast u(5)\ast l1\ast cos(u(3)-u(4))+(3\ast g\ast l1\ast l2\ast m2\ast sin(u(3)-2\ast u(4)))/2+(3\ast u(2)^2\ast l1^2\ast l2\ast m2\ast sin(2\ast u(3)-2\ast u(4)))/2+2\ast u(5)^2\ast l1\ast l2^2\ast m2\ast sin(u(3)-u(4))+2\ast g\ast l1\ast l2\ast m1\ast sin(u(3))+(5\ast g\ast l1\ast l2\ast m2\ast sin(u(3)))/2))/(l1^2\ast l2\ast(8\ast m1+15\ast m2-9\ast m2\ast cos(2\ast u(3)-2\ast u(4))))$

und für den **Fcn1** Block lautet die Funktion:

$(6\ast(4\ast Rp2\ast u(2)\ast l1\ast m1-6\ast u(1)\ast l2\ast m2\ast cos(u(3)-u(4))+12\ast Rp2\ast u(2)\ast l1\ast m2-4\ast Rp2\ast u(5)\ast l1\ast m1-12\ast Rp2\ast u(5)\ast l1\ast m2+6\ast u(2)^2\ast l1^2\ast l2\ast m2^2\ast sin(u(3)-u(4))-$

$$3*g*l1*l2*m2^2*sin(u(4))+(3*u(5)^2*l1*l2^2*m2^2*sin(2*u(3)-2*u(4)))/2+3*g*l1*l2*m2^2*sin(2*u(3)-u(4))+6*Rp1*u(2)*l2*m2*cos(u(3)-u(4))+2*u(2)^2*l1^2*l2*m1*m2*sin(u(3)-u(4))-(g*l1*l2*m1*m2*sin(u(4)))/2+(3*g*l1*l2*m1*m2*sin(2*u(3)-u(4)))/2))/(l1*l2^2*m2*(8*m1+15*m2-9*m2*cos(2*u(3)-2*u(4))))$$

1.2.2 Matlab-Function-Block

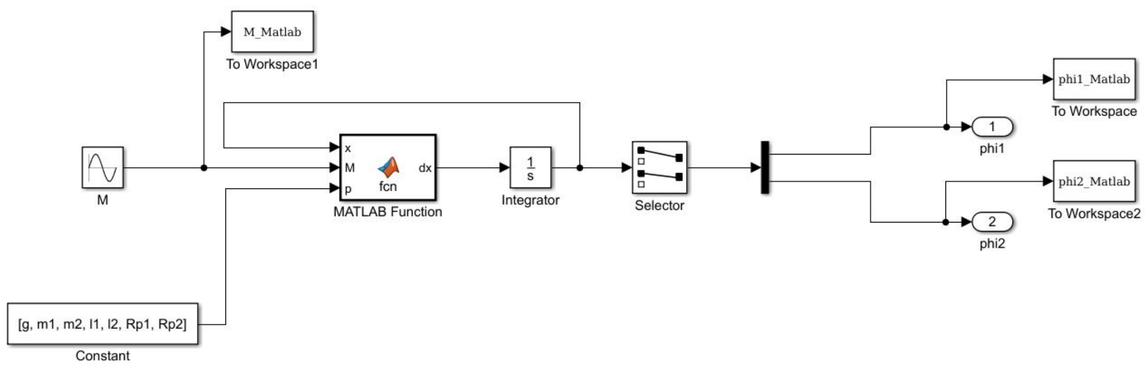

Figure 1.2: Implementierung mit einem Matlab-Function-Block

Der Code des Matlab-Fcn-Blocks ist wie folgt:

Listing 1.2: Code für das Matlab-Function-Block

```
function dx = fcn(x,M,p)
%#codegen

% Zustände definieren
phi1  = x(1);
dphi1 = x(2);
phi2  = x(3);
dphi2 = x(4);

% Parameter definieren
g = p(1);
m1 = p(2);
m2 = p(3);
l1 = p(4);
l2 = p(5);
Rp1 = p(6);
Rp2 = p(7);
```

```
% Bewegungsgleichungen des Modells
ddphi1 = -(6*(4*Rp1*dphi1*l2 - 4*M*l2 + 6*Rp2*dphi1*l1*cos(phi1 - phi2) - 6*Rp2*dphi2*l1*cos(phi1 - phi2) + (3*g*l1*l2*m2*sin(phi1 - 2*phi2))/2 + (3*dphi1^2*l1^2*l2*m2*sin(2*phi1 - 2*phi2))/2 + 2*dphi2^2*l1*l2^2*m2*sin(phi1 - phi2) + 2*g*l1*l2*m1*sin(phi1) + (5*g*l1*l2*m2*sin(phi1))/2))/(l1^2*l2*(8*m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));

ddphi2 = (6*(4*Rp2*dphi1*l1*m1 - 6*M*l2*m2*cos(phi1 - phi2) + 12*Rp2*dphi1*l1*m2 - 4*Rp2*dphi2*l1*m1 - 12*Rp2*dphi2*l1*m2 + 6*dphi1^2*l1^2*l2*m2^2*sin(phi1 - phi2) - 3*g*l1*l2*m2^2*sin(phi2) + (3*dphi2^2*l1*l2^2*m2^2*sin(2*phi1 - 2*phi2))/2 + 3*g*l1*l2*m2^2*sin(2*phi1 - phi2) + 6*Rp1*dphi1*l2*m2*cos(phi1 - phi2) + 2*dphi1^2*l1^2*l2*m1*m2*sin(phi1 - phi2) - (g*l1*l2*m1*m2*sin(phi2))/2 + (3*g*l1*l2*m1*m2*sin(2*phi1 - phi2))/2))/(l1*l2^2*m2*(8*m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));

% Zustandsraummodell
dx = [dphi1; ddphi1; dphi2; ddphi2];
```

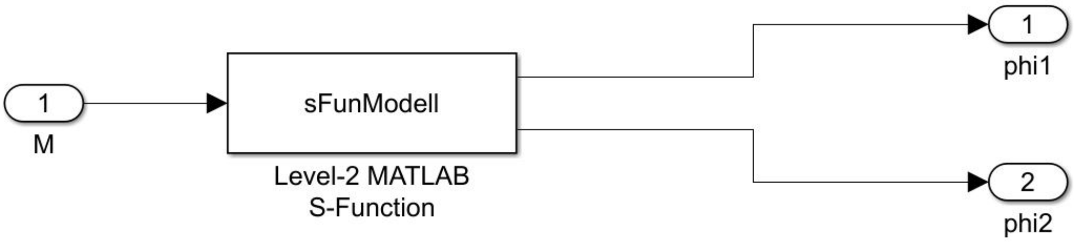

Figure 1.3: Implementierung mit einem Matlab-Function-Block

Die Implementierung des Codes für die S-Function ist wie folgt:

Listing 1.3: Code für das Matlab-Function-Block

```
function sFunModell(block)

    setup(block);

end


% ***************************************************************
% Initialisierung
% ***************************************************************
function setup(block)

    % Anzahl der Ein-/Ausgänge
    block.NumInputPorts  = 1;
    block.NumOutputPorts = 2;

    % Eigenschaften des Eingangs
    block.InputPort(1).Dimensions    = 1;
    block.InputPort(1).DatatypeID    = 0;  % double
    block.InputPort(1).Complexity    = 'Real';
    block.InputPort(1).DirectFeedthrough = false;
    block.InputPort(1).SamplingMode = 'Sample';

    % Eigenschaften des 1. Ausgangs
```

```
    block.OutputPort(1).Dimensions  = 1;
    block.OutputPort(1).DatatypeID  = 0; % double
    block.OutputPort(1).Complexity  = 'Real';
    block.OutputPort(1).SamplingMode = 'Sample';

    % Eigenschaften des 2. Ausgangs
    block.OutputPort(2).Dimensions  = 1;
    block.OutputPort(2).DatatypeID  = 0; % double
    block.OutputPort(2).Complexity  = 'Real';
    block.OutputPort(2).SamplingMode = 'Sample';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Anzahl der Zustände
    block.NumContStates = 4;

    % Anzahl der Parameter
    block.NumDialogPrms = 7;


    % Abtastzeit definieren -> zeitkontinuierlich
    block.SampleTimes = [0 0];



    % weitere Methoden registrieren
    block.RegBlockMethod('InitializeConditions', →
       ↪@InitializeConditions);
    block.RegBlockMethod('Outputs', @Outputs);
    block.RegBlockMethod('Derivatives', @Derivatives);
    block.RegBlockMethod('Terminate', @Terminate);

end


% ***************************************************************
% Anfangsbedingungen setzen
% ***************************************************************
function InitializeConditions(block)
```

```
    block.ContStates.Data = [0, 0, 0, 0];

end


% ****************************************************************
% Ausgänge berechnen
% ****************************************************************
function Outputs(block)

    % Zustände auslesen
    x       = block.ContStates.Data;

    block.OutputPort(1).Data = x(1);
    block.OutputPort(2).Data = x(3);

end


% ****************************************************************
% Ableitungen berechnen
% ****************************************************************
function Derivatives(block)

    % Parameter auslesen
    g  = block.DialogPrm(1).Data;
    m1  = block.DialogPrm(2).Data;
    m2  = block.DialogPrm(3).Data;
    l1  = block.DialogPrm(4).Data;
    l2  = block.DialogPrm(5).Data;
    Rp1  = block.DialogPrm(6).Data;
    Rp2  = block.DialogPrm(7).Data;

    % Zustände auslesen
    x       = block.ContStates.Data;
    phi1    = x(1);
    dphi1   = x(2);
    phi2    = x(3);
    dphi2   = x(4);
```

```
    % Eingang auslesen
    M = block.InputPort(1).Data(1);



    % Ableitungen berechnen
    ddphi1 = -(6*(4*Rp1*dphi1*l2 - 4*M*l2 + 6*Rp2*dphi1*l1*cos(phi1 - phi2) - 6*Rp2*dphi2*l1*cos(phi1 - phi2) + (3*g*l1*l2*m2*sin(phi1 - 2*phi2))/2 + (3*dphi1^2*l1^2*l2*m2*sin(2*phi1 - 2*phi2))/2 + 2*dphi2^2*l1*l2^2*m2*sin(phi1 - phi2) + 2*g*l1*l2*m1*sin(phi1) + (5*g*l1*l2*m2*sin(phi1))/2))/(l1^2*l2*(8*m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));



    ddphi2 = (6*(4*Rp2*dphi1*l1*m1 - 6*M*l2*m2*cos(phi1 - phi2) + 12*Rp2*dphi1*l1*m2 - 4*Rp2*dphi2*l1*m1 - 12*Rp2*dphi2*l1*m2 + 6*dphi1^2*l1^2*l2*m2^2*sin(phi1 - phi2) - 3*g*l1*l2*m2^2*sin(phi2) + (3*dphi2^2*l1*l2^2*m2^2*sin(2*phi1 - 2*phi2))/2 + 3*g*l1*l2*m2^2*sin(2*phi1 - phi2) + 6*Rp1*dphi1*l2*m2*cos(phi1 - phi2) + 2*dphi1^2*l1^2*l2*m1*m2*sin(phi1 - phi2) - (g*l1*l2*m1*m2*sin(phi2))/2 + (3*g*l1*l2*m1*m2*sin(2*phi1 - phi2))/2))/(l1*l2^2*m2*(8*m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));



    % Ableitungen zuweisen
    block.Derivatives.Data = [dphi1; ddphi1; dphi2; ddphi2];

end


% ****************************************************************
% Aufräumen (wenn nötig)
% ****************************************************************
function Terminate(block)
end
```

1.3 Vergleich der Modellierungsmöglichkeiten

Die in den vorangegangen Abschnitten vorgestellten Modelle werden mit einem Anregungssignal

$$M(t) = 0.1 sin(t)$$

und der Anfangsbedingung ($\varphi_1(t=0) = \varphi_2(t=0) = 0\,\mathrm{rad}$) simuliert. Dabei werden die Winkel φ_1 und φ_2 in Abhängigkeit vom Anregungssignal zum einen für eine Simulationsdauer von 10s und 2s dargestellt.

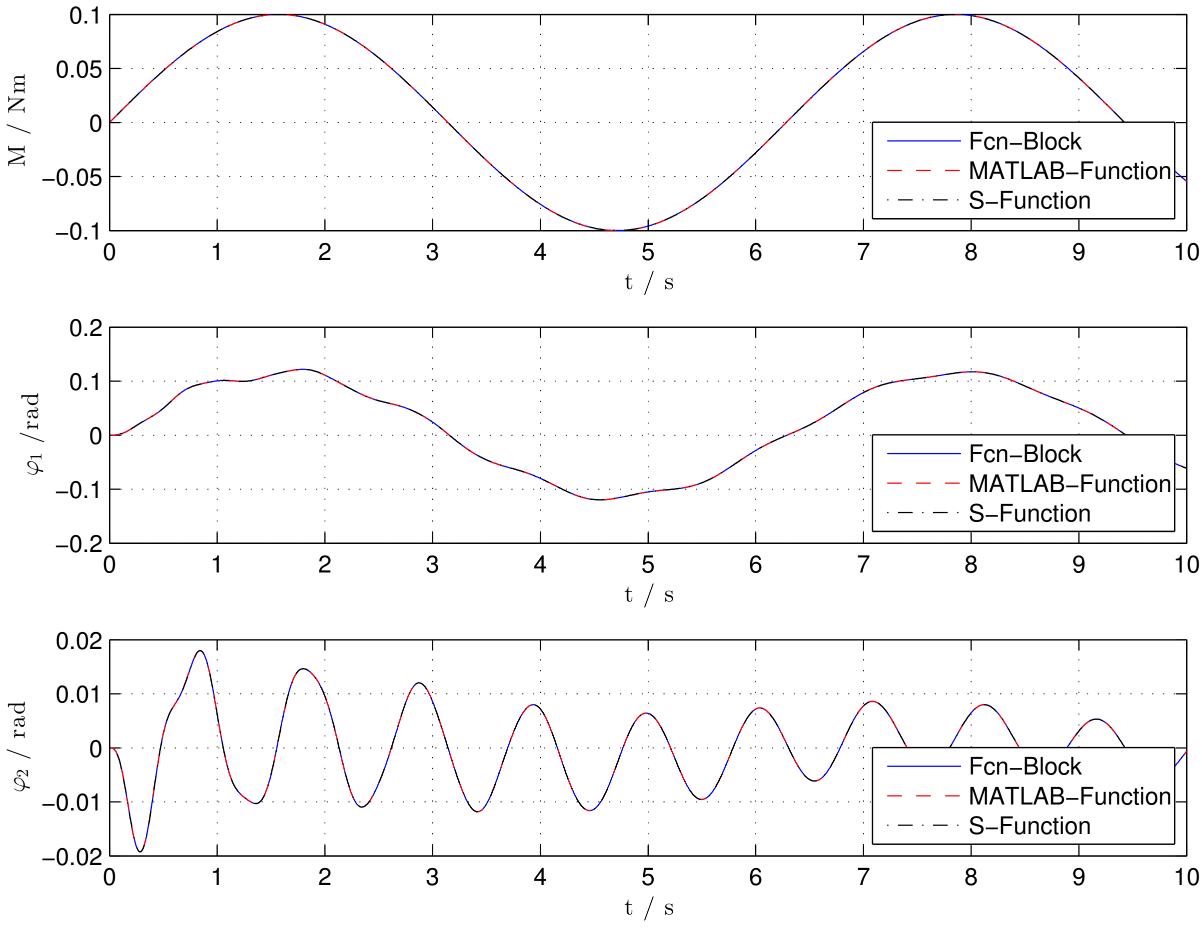

Figure 1.4: Vergleich der Modellierungsmöglichkeiten bei einer Simulationsdauer von 10s

Alle drei behandelten Modellierungsmöglichkeiten sind für die nichtlineare Modellierung geeignet. Bei gleicher Simulationseinstellung sind daher identische Ergebnisse zu erwarten. Wie die Abbildungen zeigen, erfüllen die Ergebnisse diese Erwartung.

Der Verlauf der Winkel in Abhängigkeit des Anregungssignals ist plausibel. Für den Winkel φ_1 ist ein ähnlicher Verlauf wie das Anregungsmoment, nur mit einer kleiner Verzögerung durch die Trägheitsmomente, zu erwarten, da das Anregungsmoment direkt auf die Drehachse in φ_1 wirkt und somit einen ähnlichen Verlauf erzwingt. Der Winkel φ_2 ist dagegen frei gelagert und

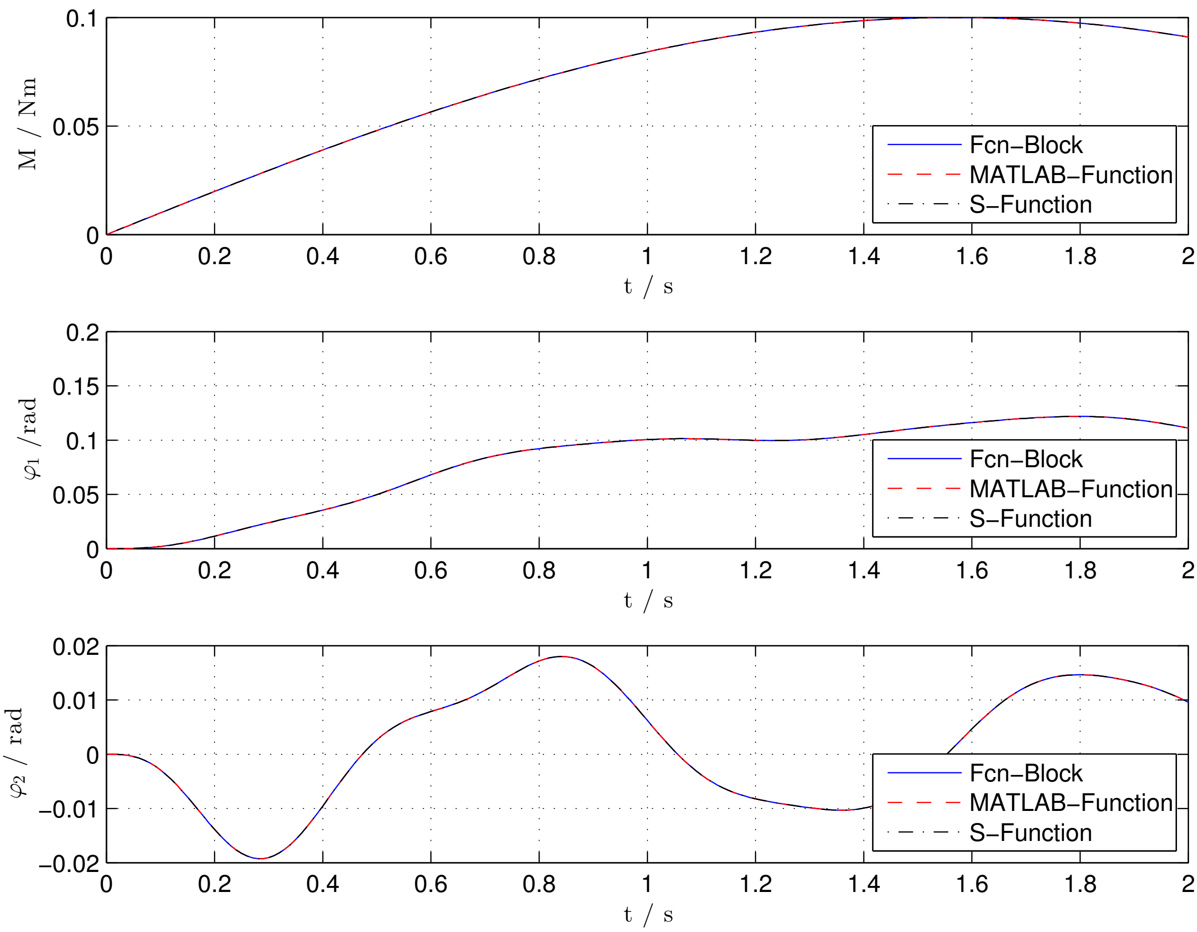

Figure 1.5: Vergleich der Modellierungsmöglichkeiten bei einer Simulationsdauer von 2s

ist nur durch ein Reibmoment begrenzt. Folglich ist anstatt eines zur Anregung synchronen Verlaufs ein frei schwingendes Verhalten zu erwarten. Das heißt, dass der zweite Pendel auch in entgegengesetzter Richtung schwingen kann. Dies ist in der Darstellung der Ergebnisse für eine Simulationsdauer von 2s besonders am Anfang zu erkennen. Durch die Trägheit schwingt der zweite Pendel anfänglich in die negative φ_2-Richtung. Des weiteren sind beide Winkelverläufe nicht perfekt sinusförmig, sondern leicht wellig. Dies ist darauf zurückzuführen, dass das freie schwingen des zweiten Pendels Kräfte auf das erste Pendel aufprägt und damit den sinusverlauf stört.