

# Versuch IV:

# Beobachterentwurf -

# Benutzeroberflächen

Andreas Jentsch, Ali Kerem Sacakli

Praktikumsbericht – Praktikum Matlab/Simulink II

27. Juni 2017



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

REGELUNGSTECHNIK  
UND MECHATRONIK

**rtm**



---

## 4.8 Verhalten des Regelkreises mit Beobachter

---

Nachfolgend sind Verläufe der theoretisch messbaren Zustandsgrößen, sowie der Stellgröße abgebildet, um den Einfluss von verschiedenen Beobachtereigenwerten darzustellen. Dabei sind die vom Beobachter geschätzten Zustände zusammen mit den realwerten abgebildet.

Die Eigenwerte des geschlossenen Regelkreises liegen bei,

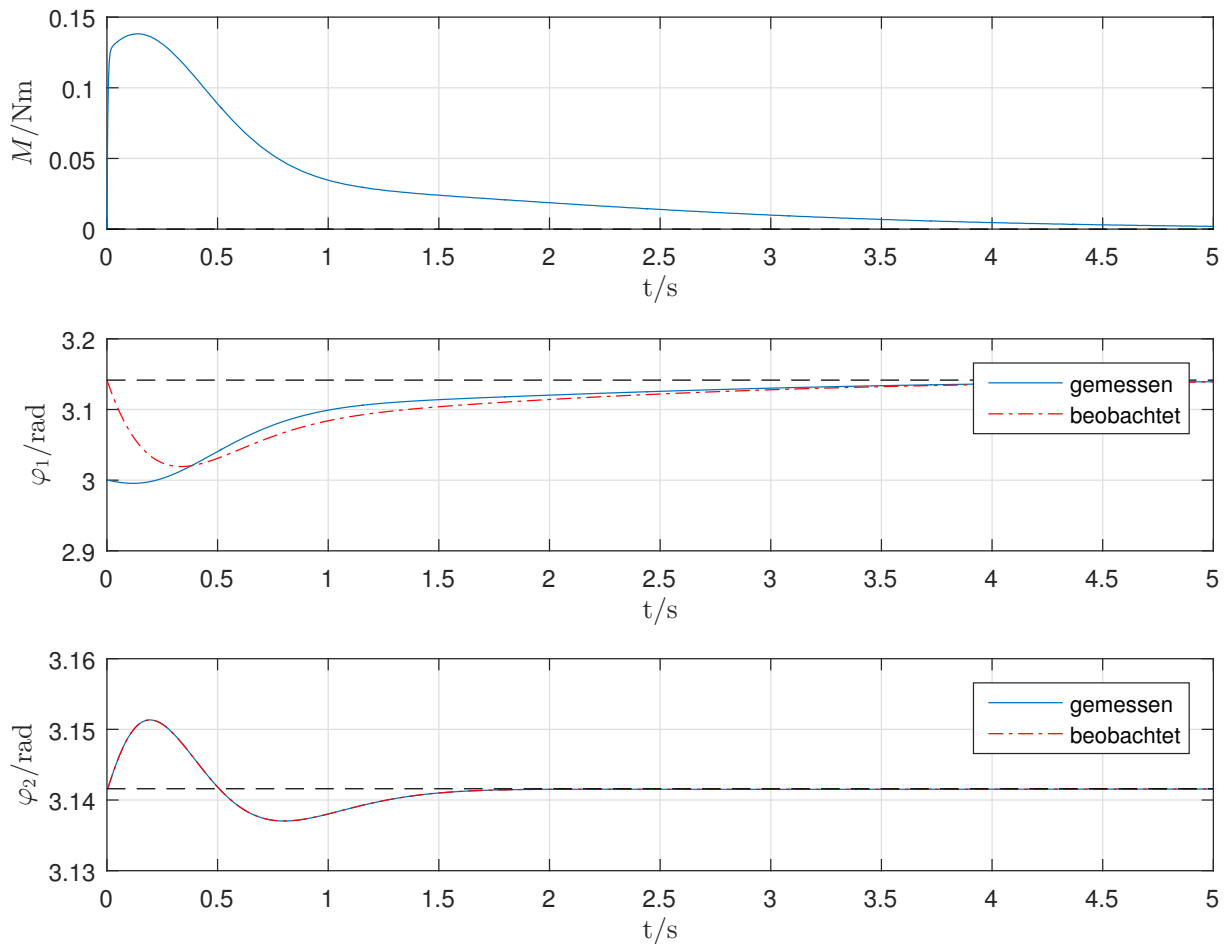
$$\lambda = \begin{bmatrix} -1,3152 & -5,6606 - 3,0304i & -5,6606 + 3,0304i & -258,5879 \end{bmatrix}$$

Der Arbeitspunkt, sowie die Startwerte des Systems und des Beobachters liegen bei

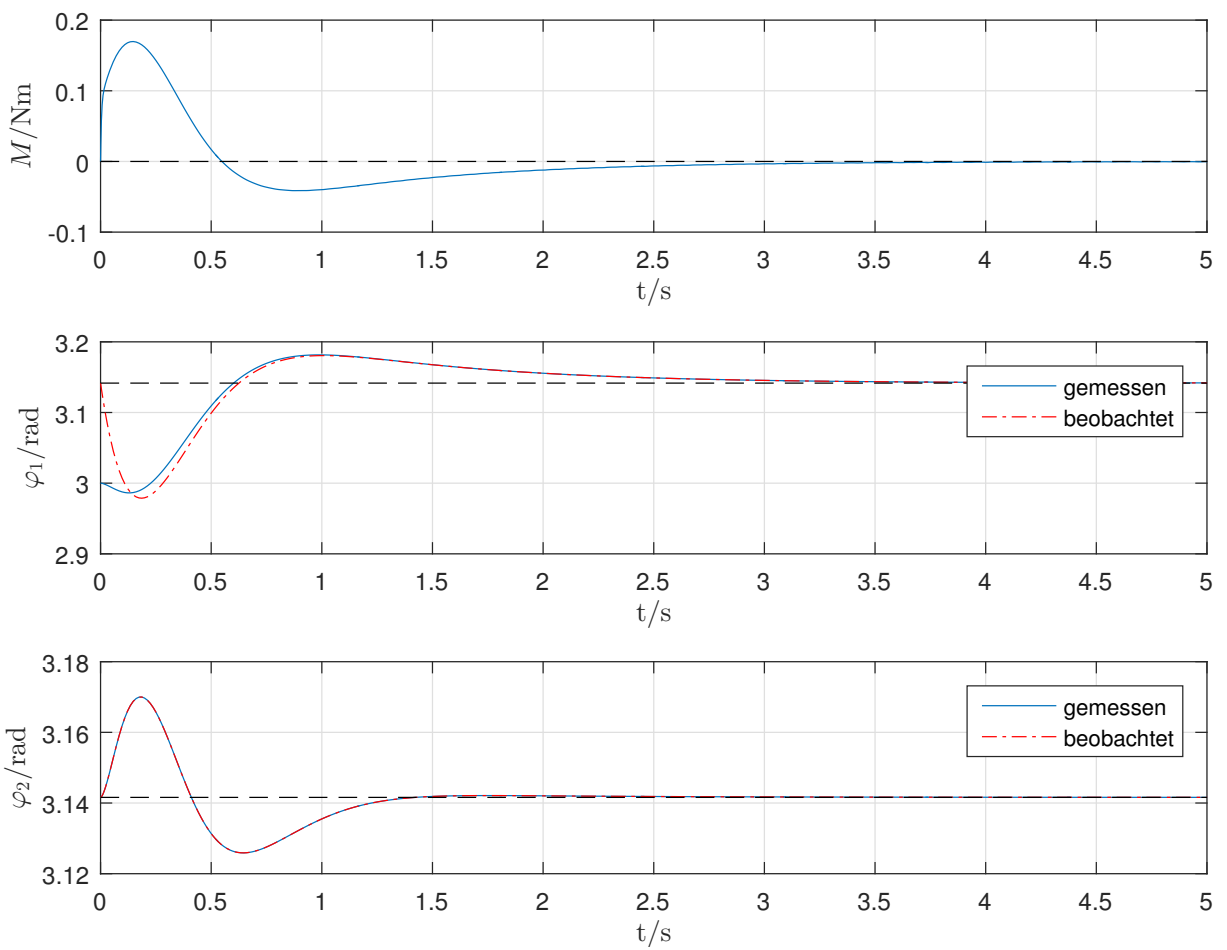
$$\begin{aligned} \mathbf{x}_{AP} &= \begin{bmatrix} \pi & 0 & \pi & 0 \end{bmatrix} \\ \mathbf{x}_0^T &= \begin{bmatrix} 3 & 0 & \pi & 0 \end{bmatrix} \\ \mathbf{x}_{obs,0}^T &= \begin{bmatrix} \pi & 0 & \pi & 0 \end{bmatrix} \end{aligned}$$

In Figure 4.1 bis 4.4 werden zunehmend schneller ausgelegte Beobachter verwendet. Beim langsamsten Beobachter in Figure 4.1 ist neben der langsamen Ausregelung des Schätzfehlers eine niedrige maximale Stellgröße festzustellen, das System regelt den Winkel  $\varphi_1$  infolgedessen sehr langsam aus. Hier liegen die dominanten Eigenwerte des Beobachters noch rechts des dominanten Systemeigenwertes. Die übrigen Abbildungen zeigen den folgenden Trend: Je weiter links der Systemeigenwert die Beobachtereigenwerte liegen, desto schneller wird der Beobachtungsfehler ausgeregelt. Damit steigt auch die Maximale Stellgröße und das System wird schneller in den Arbeitspunkt überführt.

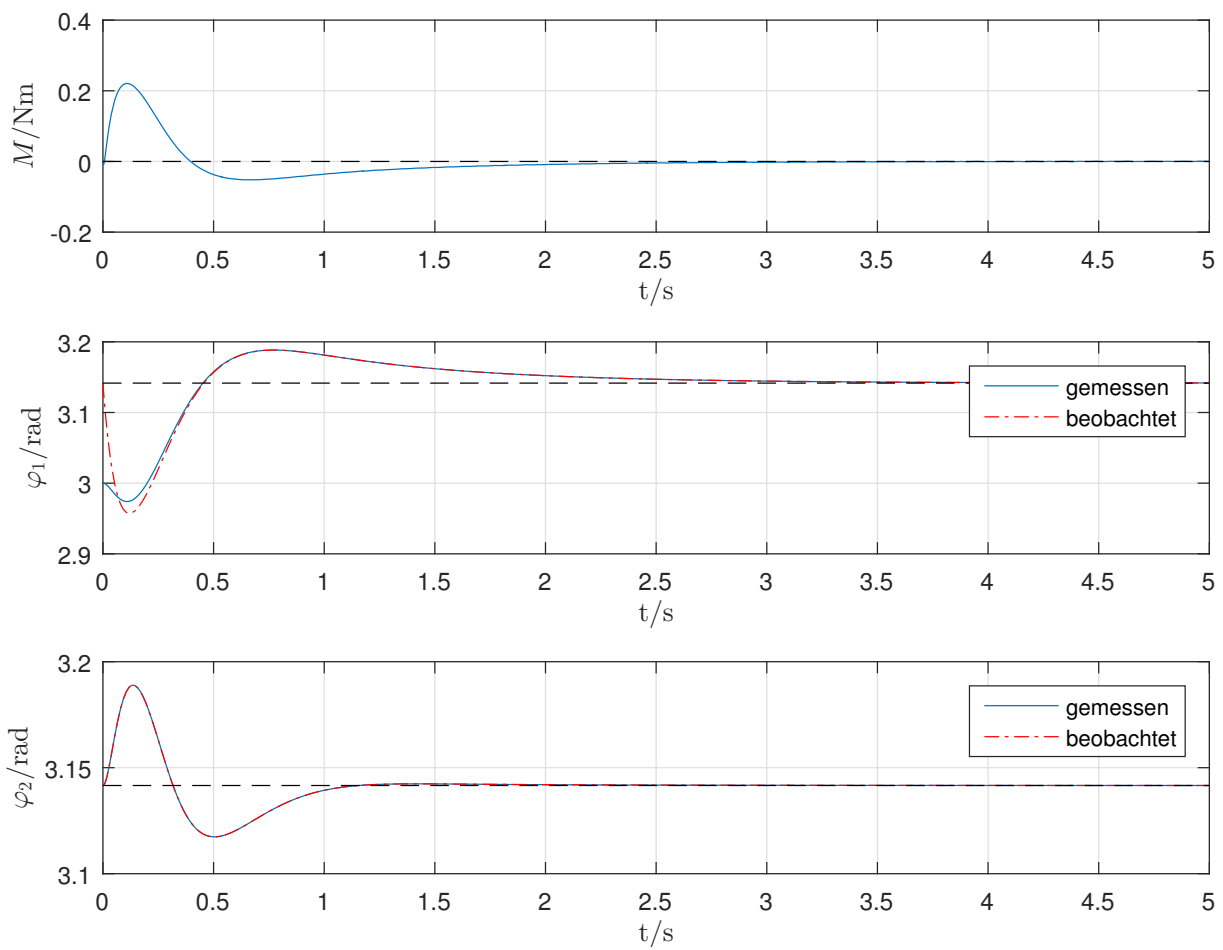
Bei den Werten für die gemessenen Zustände  $x_1$  und  $x_3$ , welche den Winkeln  $\varphi_1$  beziehungsweise  $\varphi_2$  entsprechen fällt auf, dass kein Schätzfehler entsteht, wenn die Anfangswerte von Beobachter und System übereinstimmen. Dies ist bei allen Verläufen für  $\varphi_2$  zu sehen. Um den anfänglichen Schätzfehler bei  $\varphi_1$  zu minimieren, könnten die Anfangswerte des Beobachters an die des Systems angepasst werden,  $\mathbf{x}_{obs,0} = \mathbf{x}_0$ .



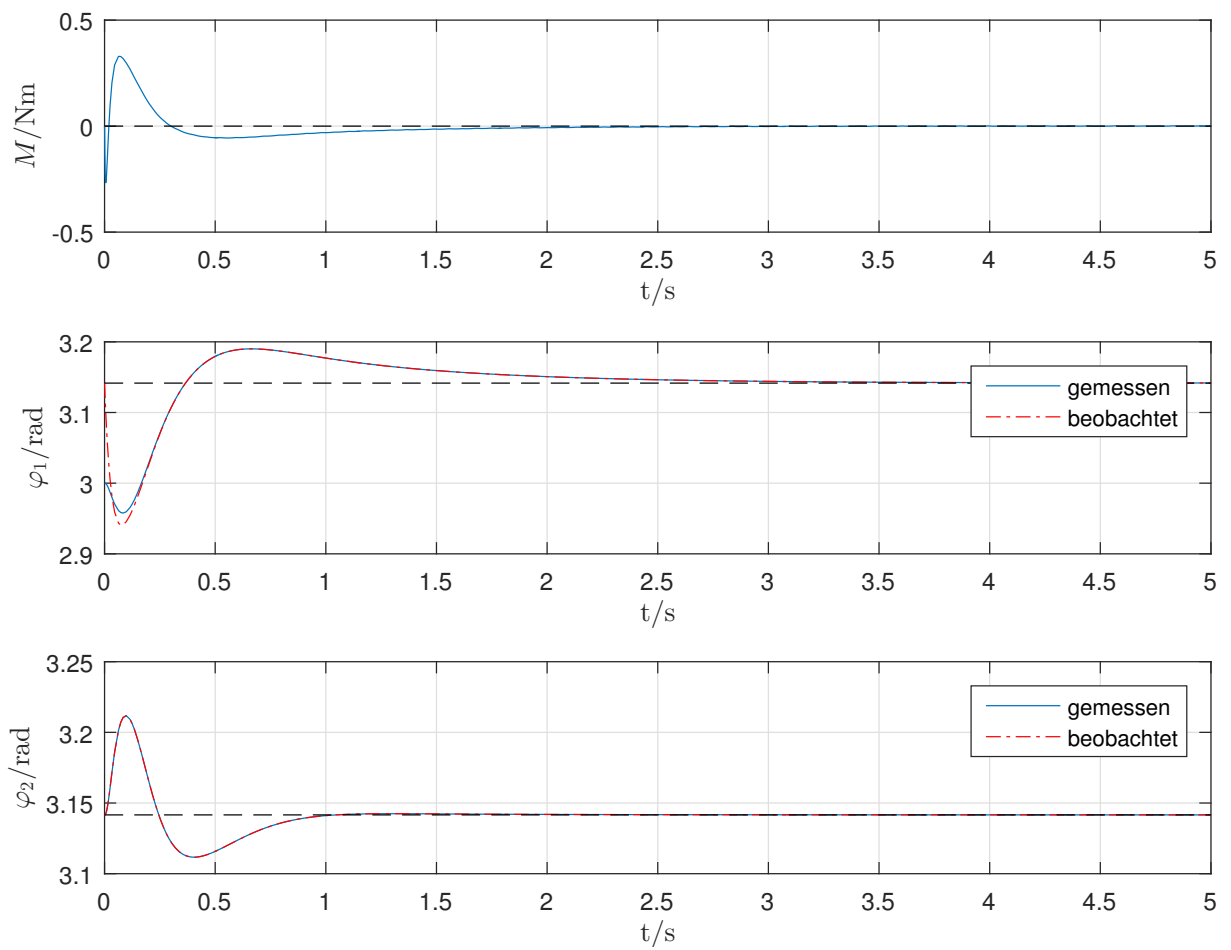
**Figure 4.1:** Systemverhalten mit Beobachtereigenwerten bei: [-1 -1 -5 -5]



**Figure 4.2:** Systemverhalten mit Beobachtereigenwerten bei:  $[-5 \ -5 \ -10 \ -10]$



**Figure 4.3:** Systemverhalten mit Beobachtereigenwerten bei: [-10 -10 -20 -20]



**Figure 4.4:** Systemverhalten mit Beobachtereigenwerten bei: [-20 -20 -40 -40]

## 4.9 GUI Entwurf

Unter Ausnutzung der schon erstellten Funktionen soll in diesem Versuch das Modell durch eine grafische Benutzeroberfläche (Figure 4.5) und einen Luenberger-Beobachter erweitert werden. Die wichtigsten Callback-Funktionen sind in Listing 4.1 bis 4.3 aufgeführt.

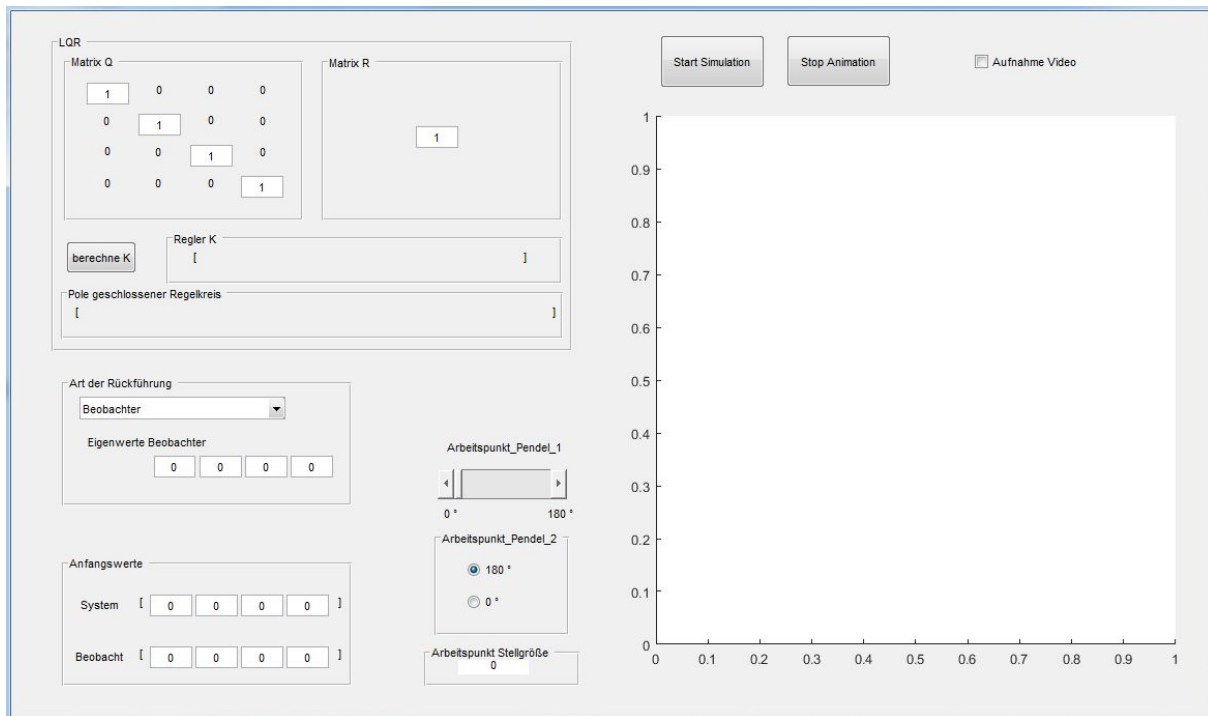


Figure 4.5: Finale grafische Benutzeroberfläche

Listing 4.1: Quellcode der Callback-Funktion zur Reglerberechnung

```
1 % --- Executes on button press in berechneK.
function berechneK_Callback(hObject, eventdata, handles)
    % hObject    handle to berechneK (see GCBO)
    % eventdata  reserved - to be defined in a future version of →
    % ←MATLAB
    % handles    structure with handles and user data (see GUIDATA)

6
    % Struktur mit den Handles aller Objekte der GUI erzeugen
    h = guihandles();

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

11 % Auslesen der Matrix Q
    q11 = str2num(get(h.Q11, 'String'));
    q22 = str2num(get(h.Q22, 'String'));
    q33 = str2num(get(h.Q33, 'String'));
```



```

q44 = str2num(get(h.Q44, 'String'));

16
Q = diag([q11 q22 q33 q44]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 % Auslesen von R
R = str2num(get(h.R, 'String'));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 % Auslesen des Arbeitspunkts
% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
% Ggf. an eigene Codierung des Arbeitspunktes anpassen!
% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
AP = [0 0 0 0];
31 value1 = get(h.slider_AP, 'Value');
    AP(1) = value1*pi;
value2 = get(h.AP_2_1, 'Value');
if (value2 == 1)
    AP(3) = pi;
36 else % (value == 0)
    AP(3) = 0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[f_m, h_m] = nonlinear_model();

41
[A, B, C, D, M_AP] = linearisierung(f_m, h_m, AP);

stObs = getappdata(h.figure1, 'stObs');
stObs.A = A;
46 stObs.B = B;
stObs.C = C;
setappdata(h.figure1, 'stObs', stObs);

[K, poleRK] = berechneLQR(A, B, Q, R);
51 % Anzeigen des Vektors 'K' im Textfeld 'reglerK'

set(h.reglerK, 'String', num2str(K));
set(h.poleRK, 'String', num2str(poleRK));

```

```

56     set(h.M_AP, 'String', num2str(M_AP));
% end function berechneK_Callback

```

---

**Listing 4.2:** Quellcode Callback-Funktion zum Start der Simulation

---

```

% --- Executes on button press in startSim.
function startSim_Callback(hObject, eventdata, handles)
3 % hObject    handle to startSim (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Initialisiert variable zum stoppen der Animation
8 global stopAnimation;
stopAnimation = false;

h = guihandles();
cla(h.axes1);

13 % Startwerte aus der GUI auslesen
x0(1,1) = str2num(get(h.x01, 'String'));
x0(2,1) = str2num(get(h.x02, 'String'));
x0(3,1) = str2num(get(h.x03, 'String'));
18 x0(4,1) = str2num(get(h.x04, 'String'));

% Arbeitspunkt aus der GUI auslesen
AP = [0 0 0 0];
value1 = get(h.slider_AP, 'Value');
23 AP(1) = value1*pi;
value2 = get(h.AP_2_1, 'Value');
if (value2 == 1)
    AP(3) = pi;
else % (value == 0)
28 AP(3) = 0;
end
M_AP = str2num(get(h.M_AP, 'String'));

% Regler aus der GUI auslesen
33 K = str2num(get(h.reglerK, 'String'));
stPendel = ladePendel();

```

---

```

% Reglerpole aus der GUI auslesen
stObs = getappdata(h.figure1, 'stObs');
38 stObs.pole(1) = str2num(get(h.lam_b_1, 'String'));
stObs.pole(2) = str2num(get(h.lam_b_2, 'String'));
stObs.pole(3) = str2num(get(h.lam_b_3, 'String'));
stObs.pole(4) = str2num(get(h.lam_b_4, 'String'));

43 % Reglerstartwerte aus der GUI auslesen
stObs.x0(1) = str2num(get(h.x01b, 'String'));
stObs.x0(2) = str2num(get(h.x02b, 'String'));
stObs.x0(3) = str2num(get(h.x03b, 'String'));
stObs.x0(4) = str2num(get(h.x04b, 'String'));

48 % Fragt ab ob mit oder ohne Beobachter
logic = get(h.popupmenu2, 'Value');
if logic == 1
    stObs.switch = true;
53 else
    stObs.switch = false;
end

% Beobachter berechnen
58 stObs.L = berechneBeobachter(stObs.A, stObs.C, stObs.pole);
setappdata(h.figure1, 'stObs', stObs);

% Simulation des Modells
[vT, mX, mXobs, u] = runPendel(stPendel, AP, K, x0, M_AP, stObs);

63 % Variablen zum plotten in den Base Workspace schreiben
assignin('base', 'vT', vT);
assignin('base', 'mX', mX);
assignin('base', 'mXobs', mXobs);
68 assignin('base', 'u', u);
assignin('base', 'M_AP', M_AP);
assignin('base', 'x0', x0);

% Abfragen ob die animation aufgezeichnet werden soll
73 if get(h.aufnahme, 'Value') == 1
    record = true;
else

```

---

```

    record = false;
end

```

78

```

%Animation des Pendels
animierePendel(vT,mX,stPendel,h.axes1,record);

```

---



---

**Listing 4.3:** Quellcode der Callback-Funktion zum Stoppen der Animation

---

```

% --- Executes on button press in stopAnimation.
function stopAnimation_Callback(hObject, eventdata, handles)
% hObject    handle to stopAnimation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
5 % handles    structure with handles and user data (see GUIDATA)
global stopAnimation;
stopAnimation = true;

```

---



---

## 4.10 Beobachterentwurf

---

Zur Berechnung der Beobachter-Matrix L soll die Funktion berechneBeobachter implementiert werden.

---

**Listing 4.4:** Quellcode der Funktion berechneBeobachter

---

```

function L = berechneBeobachter(A, C, poleBeobachter)
MB = obsv(A,C);
3
if rank(MB)==length(A)
L = place(A', C', poleBeobachter)';
else
disp('System_nicht_vollstndig_beobachtbar');
8 end

end

```

---

Fr die Erweiterung soll zudem die Funktion runPendel erweitert werden:

---

**Listing 4.5:** Quellcode der Funktion runPendel

---

```

function [ vT, mX, mXobs, u ] = runPendel( stPendel, AP, K, x0, M_AP→
    ←, stObs )

vT = 'error';

```

---

---

```
4 mX = 'error';
   mXobs = [];
   u = [];

   if ~isempty(stObs)
9       stObs.switch = true;
       else
           stObs.switch = false;
           stObs.A = eye(4);
           stObs.B = [0;1;0;1];
14          stObs.C = [1 0 0 0; 0 0 1 0];
           stObs.L = stObs.C';
           stObs.x0 = [0 0 0 0];
       end

19 Tend = 10;
   stOptions = simset( 'SrcWorkspace', 'current');
   sim('Modell_V4', Tend, stOptions);

   vT = mZustand.Time;
24 mX = mZustand.Data;
   mXobs = mBeobacht.Data;
   u = vInput.Data;

end
```

---

Das Zugrundeliegende Simulink Modell mit Beobachter ist in Figure 4.6 zu sehen.

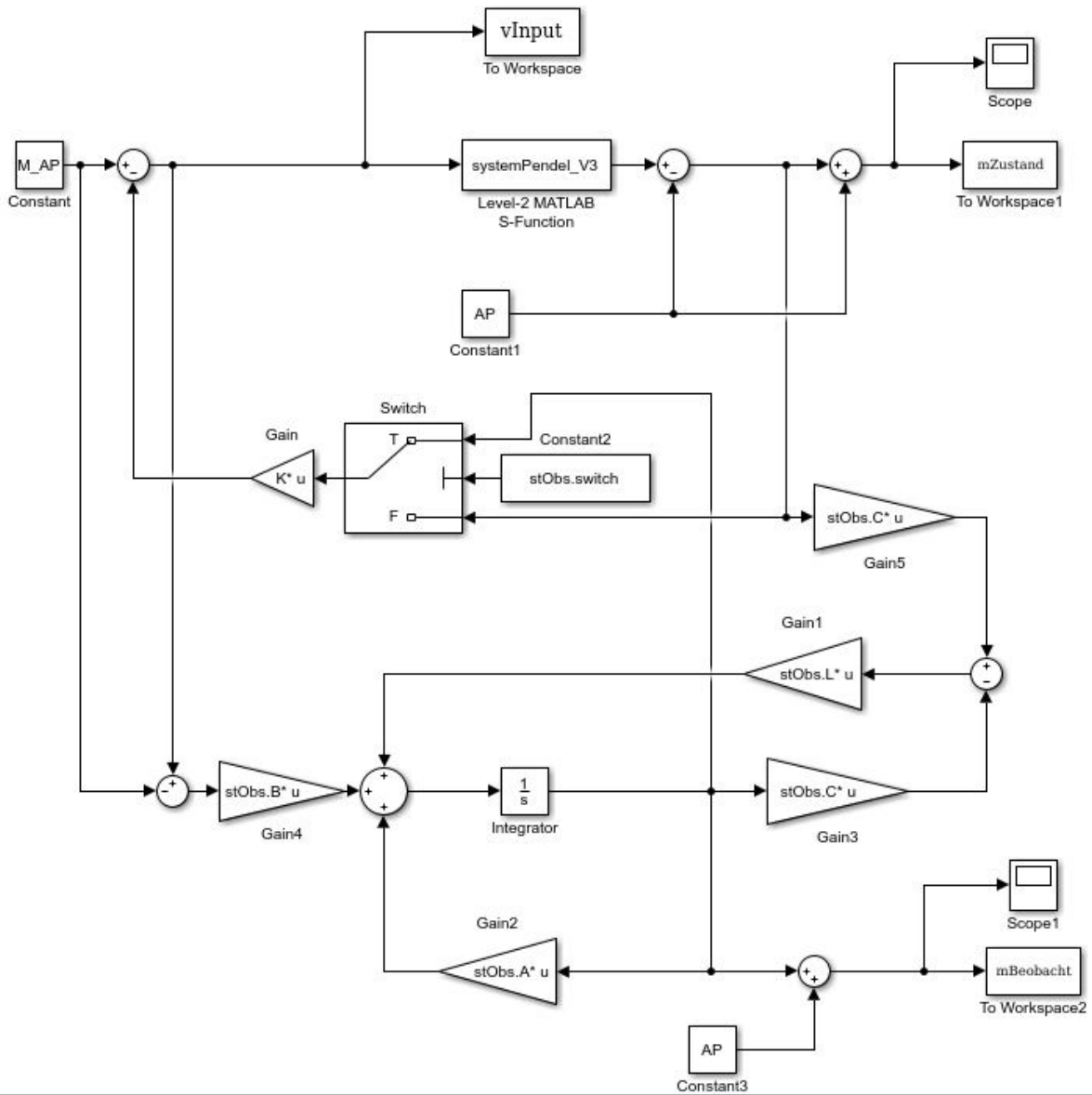


Figure 4.6: Simulink-Modell mit Luenberger-Beobachter