Regelung mechatronischer Systeme

1. Übung, ausgegeben von: Frank Woittennek
zuletzt geändert am: 8. Mai 2022

Es wird das in Abb. 1 skizzierte Helikopter-Rack betrachtet. Dieses besteht aus einem im Abstand d außerhalb des Schwerpunktes drehbar gelagerten Stab, an dessen Enden je ein Propeller befestigt ist. Der Propeller im Abstand d_ϵ zum Drehpunkt soll dabei eine Kraft F_ϵ zur Auslenkung des Elevationswinkels ϵ einprägen, der Propeller im Abstand d_α eine Kraft F_α zur Beeinflussung des Azimut-Winkels α.

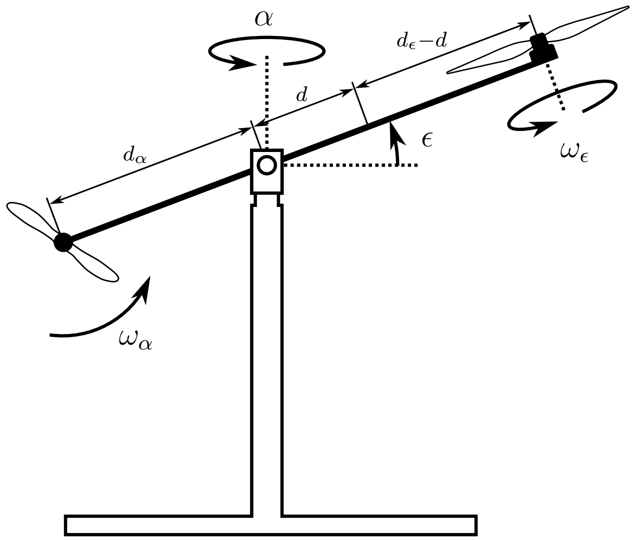

Abb. 1 – Helikopter-Rack

Zur Modellierung des Systems wird der Euler-Lagrange-Formalismus verwendet. Mit den Tabelle 1 zu entnehmenden Parametern lauten die potentielle und die kinetische Energie des Helikopter-Racks

$$
\begin{aligned}
V &= V_{\max} \sin(\epsilon) \\
T &= \frac{J_3}{2}\dot{\epsilon}^2 + \frac{J_1 + J_2\cos^2(\epsilon)}{2}\dot{\alpha}^2 + \frac{J_4}{2}(\cos(\epsilon)\dot{\alpha} + \omega_1)^2 + \frac{J_5}{2}(\dot{\epsilon} + \omega_\alpha)^2.
\end{aligned}
$$

Führt man die verallgemeinerten Koordinaten in $\boldsymbol{q} = (\alpha, \epsilon)$ und die zugehörigen verallgemeinerten Impulse in $\boldsymbol{p} = (p_\alpha, p_\epsilon)^T$ gemäß

$$p_\alpha = \frac{\partial T}{\partial \alpha} = (J_1 + \bar{J}_2\cos^2(\epsilon))\dot{\alpha} + J_4\cos(\epsilon)\omega_\epsilon \tag{1a}$$

$$p_\epsilon = \frac{\partial T}{\partial \epsilon} = \bar{J}_3\dot{\epsilon} + J_5\omega_\alpha \tag{1b}$$

ein, wobei die Abkürzungen $\bar{J}_3 = J_3 + J_5$ und $\bar{J}_2 = J_2 + J_4$ verwendet werden, so können die Bewegungsgleichungen in der Form

$$\dot{p}_\alpha = d_\alpha\cos(\epsilon)F_\alpha - D_\alpha \tag{2a}$$

$$\dot{p}_\epsilon = -V_{\max}\cos(\epsilon) - \tfrac{1}{2}\bar{J}_2\sin(2\epsilon)\dot{\alpha}^2 - J_4\,\omega_\epsilon\sin(\epsilon)\dot{\alpha} + d_\epsilon F_\epsilon - D_\epsilon \tag{2b}$$

angeschrieben werden. Darin lauten die aerodynamischen Rotorkräfte

$$F_\epsilon = s_\epsilon|\omega_\epsilon|\omega_\epsilon, \quad F_\alpha = s_\alpha|\omega_\alpha|\omega_\alpha \tag{3}$$

und die durch Reibung verursachten Drehmomente

$$D_\epsilon = c_\epsilon\dot{\epsilon}, \quad D_\alpha = c_\alpha\dot{\alpha}. \tag{4}$$

Der Azimut-Winkel α ist in den nachfolgenden Überlegungen nicht mehr von Interesse, sondern lediglich die zugehörige Winkelgeschwindigkeit. Als Zustand wird deshalb der Vektor

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \epsilon \\ p_\alpha \\ p_\epsilon \end{pmatrix}$$

Parameter	Wert	Beschreibung
$V_{\max} = mgd$	0.024	maximale potentielle Energie
J_1	0.002	Hauptträgheitsmoment Vertikalwelle um Vertikale
J_2	0.012	Hauptträgheitsmoment Ausleger um horizontale Normale durch Drehpunkt
J_3	0.013	Hauptträgheitsmoment Ausleger um (geneigte) Vertikale durch Drehpunkt
$J_4,\ J_5$	10^{-5}	Hauptträgheitsmomente der Rotoren um ihre Rotationsachse
$c_\epsilon,\ c_\alpha$	0.001	Reibkoeffizienten
$d_\alpha,\ s_\epsilon$	$8 \cdot 10^{-7}$	Schub-Kenngrößen der Propeller
$d_\alpha,\ d_\epsilon$	0.272	Abstände der Rotoren vom Drehpunkt
m		Masse des Auslegers
g		Erdbeschleunigung
d		Abstand des Massenmittelpunktes des Auslegers vom Drehpunkt

Tabelle 1 – Parameter des Helikopter-Racks

verwendet. Der Ausgang ist durch den Elevationswinkel und die Azimut-Winkelgeschwindigkeit gegeben:

$$\boldsymbol{y} = (\epsilon, \dot{\alpha})$$

Die Winkelgeschwindigkeiten ω_α und ω_ϵ der Rotoren werden als Eingangsgrößen des Systems betrachtet

$$\boldsymbol{u} = (\omega_\alpha, \omega_\epsilon)^T.$$

Aufgabe 1 (Kontinuierliche Simulationsmodelle)

a) Vervollständigen Sie das in den Methoden `model` und `output` der Klasse `Heli` in der Datei `libheli.py` definierte Simulationsmodell derart, dass die Ableitungen der Zustandsgrößen bzw. der Systemausgang zurückgegeben werden. Die Methode `output` sollte dabei auch mit vektoriellen Argumenten der Länge N für die Zeit und entsprechenden matrixwertigen Argumenten der Dimension $3 \times N$ für den Zustand umgehen können.

   Prüfen Sie die Korrektheit ihres Modells anhand zur Verfügung gestellten Testdaten.

b) Bestimmen Sie alle Ruhelagen $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ zu einem gegebenen konstantem Ausgang $\bar{\boldsymbol{y}}$. In diesen Regimes rotiert der Ausleger mit konstanter Geschwindigkeit $\bar{\dot{\alpha}} = \bar{y}_2$ um die Vertikale während er im Winkel $\bar{y}_1 = \bar{\epsilon}$ geneigt ist.

   Vervollständigen Sie die Methode `equilibrium` der Klasse `Heli` in der Datei `libheli.py` derart, dass die entsprechende Ruhelage zurückgegeben wird.

   Prüfen Sie die Korrektheit Ihrer Lösung numerisch durch Einsetzen der in in Tabelle 2 abgebildeten Ruhelagen in die Modellgleichungen.

c) Simulieren Sie Ihr Modell für kleine Auslenkungen aus den in Tabelle 2 angegebenen Ruhelagen. Nutzen Sie dazu Abschnitt 1.1.2. des zur Verfügung gestellten Notebooks.

d) Linearisieren Sie die Zustandsdifferentialgleichungen um eine beliebige Ruhelage.

   Nutzen Sie Ihre Ergebnisse, um die Methode `linearize` der Klasse `Heli` in der Datei `libheli.py` derart zu vervollständigen, dass in Abhängigkeit der übergebenen Werte für $\bar{\boldsymbol{y}}$ die konstanten Systemmatrizen A, B, C und D der linearisierten Zustandsdarstellung

$$\dot{\tilde{\boldsymbol{x}}}(t) = A\tilde{\boldsymbol{x}}(t) + B\tilde{\boldsymbol{u}}(t), \quad \tilde{y} = C\tilde{\boldsymbol{x}}(t) + D\tilde{\boldsymbol{u}}(t)$$

   für die Kleinsignalgrößen

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \bar{\boldsymbol{x}}, \quad \tilde{\boldsymbol{y}} = \boldsymbol{y} - \bar{\boldsymbol{y}}, \quad \tilde{\boldsymbol{u}} = \boldsymbol{u}$$

   zurückgegeben werden.

   Prüfen Sie das Ergebnis der Linearisierung mit Hilfe der Methode `verify_linearization` der Klasse `Heli` (Abschnitt 1.2.2 des Notebooks).

e) Berechnen Sie (numerisch) die Eigenwerte der Systemmatrizen des linearisierten Systems. Interpretieren Sie die Simulationsergebnisse aus Teilaufgabe c) im Hinblick auf diese Eigenwerte (mündlich).

	Ruhelage 1	Ruhelage 2	Ruhelage 3	Ruhelage 4
$\bar{\epsilon}$	$-\frac{1}{18}\pi$	$\frac{1}{18}\pi$	$\frac{1}{18}\pi$	0
$\bar{\alpha}$	0.5π	0.25π	10π	0.375π

Tabelle 2 – Zu untersuchende Ruhelagen.

Aufgabe 2 (Kontinuierlicher Reglerentwurf)

a) Prüfen Sie die Steuerbarkeit des linearisierten Systems in den Ruhelagen 1 und 2 mit dem Kalmanschen Kriterium. Geben Sie die möglichen Kronecker-Indizes an. Nutzen Sie dazu die bereits vorgegebene Funktion `controllability_matrix` aus der Datei `libmimo.py`.

b) Zum Zwecke des Steuerungsentwurfs soll das System auf Regelungsnormalform transformiert und (numerisch) die Parametrierung aller Systemgrößen durch den flachen Ausgang $\boldsymbol{\eta}$ bestimmt werden. Dazu steht die Klasse `ContinuousFlatnessBasedTrajectory` zur Verfügung.

   Eine Instanz dieser Klasse wird durch den Aufruf der Methode `rest_to_rest_trajectory` erzeugt. Diese gehört zur Klasse `ContinuousLinearizedSystem`.

   Vervollständigen Sie im Konstruktor der Klasse `ContinuousFlatnessBasedTrajectory` insbesondere die Vorschrift für die Umrechnung zwischen den stationären Werten `eta_a` und `eta_b` des flachen Ausgang und `ya` und `yb` des Systemausgangs.

   Vervollständigen Sie weiterhin die Vorschriften zur Berechnung des Zustands aus der Trajektorie des flachen Ausgangs in der Methode `state` der selben Klasse. Orientieren Sie sich gegebenenfalls an der Methode `input` mit der der Eingang berechnet wird.

   Machen Sie sich auch mit dem übrigen Teil des in der Klasse implementierten Codes vertraut und versuchen Sie die Algorithmen nachzuvollziehen.

c) Wählen Sie geeignete Kronecker-Indizes und planen für das um Ruhelage 4 linearisierte System flachheitsbasiert, also auf Basis der Regelungsnormalform, eine Trajektorie, die das System in der Zeit $T = 2$ aus der Ruhelage 1 in die Ruhelage 2 überführt.

   Simulieren Sie dieses Szenario mit dem in der linearisierten und dem nichtlinearen Modell (Abschnitt 2.1 des Notebooks).

d) Berechnen Sie für das um die Ruhelage 4 linearisierte System mit der Ackermannformel für Mehrgrößensysteme eine Zustandsrückführung $\tilde{\boldsymbol{u}} = -K\tilde{\boldsymbol{x}}$. Wählen Sie dazu die Eigenwerte des geschlossenen Regelkreises derart, das sowohl die Startruhelage 1 als auch die Zielruhelage 2 der oben berechneten Überführung stabilisiert werden.

   Der Aufruf der bereits vorimplementierten Ackermannformel erfolgt über die Methode `acker` der Klasse `LinearizedSystem`, von der auch `ContinuousLinearizedSystem` abgeleitet ist.

   Simulieren Sie den Übergang von Ruhelage 1 in Ruhelage 2 anhand des um Ruhelage 4 linearisierten und des nichtlinearen Modell (Abschnitt 2.2 des Notebooks).

Aufgabe 3 (Diskreter Reglerentwurf)

Das erhaltene Regelgesetz soll digital implementiert werden (Abschnitt 3 des Notebooks).

a) Simulieren Sie das System mit dem zeitkontinuierlich entworfenen Regelgesetz in einem Abtastszenario für die Abtastzeiten 0.01 s, 0.2 s, 0.45 s. Nutzen Sie dazu die bereits vorbereiteten Funktionen des Notebooks.

b) Vervollständigen Sie in der Definition der Klasse `ContinuousLinearizedSystem` die Methode `discretize` derart, dass das zugehörige zeitdiskrete System der Klasse `DiscreteLinearizedSystem` instanziert wird.

   Bestimmen Sie im Anschluss für das diskretisierte System eine Zustandsrückführung sowie eine Steuerung. Wiederholen Sie dazu die für den zeitkontinuierlichen Fall vorgegebenen Entwurfsschritte und nutzen Sie die bereits vorbereiteten Funktionen und Klassen. Legen Sie dazu die Eigenwerte des geschlossenen Kreises nach $z_i = e^{T_a z_i}$ wobei $s_1, \ldots, s_3$ den Wunscheigenwerten des kontinuierlichen Regelkreises entsprechen. Simulieren Sie im Anschluss Ihre Steuerungs- und Regelungsansätze mit den Abtastzeiten 0.1 s und 0.5 s.

Neben den im Notebook zu dokumentierenden Ansätzen und durchzuführenden Berechnungen sind auch hier zusätzlich einige Methoden in den Klassen der Datei `libheli.py` zu ergänzen. Dazu sind insbesondere Methoden der Klasse `DiscreteFlatnessBasedTrajectory` derart anzupassen, dass sie für eine zeitdiskrete Trajektorienplanung genutzt werden können. Orientieren Sie sich bei Ihren Arbeiten an den entsprechenden Methoden der Klasse `ContinuousFlatnessBasedTrajectory`.

Eine Instanz der Klasse `DiscreteFlatnessBasedTrajectory` wird durch den Aufruf der Methode `rest_to_rest_trajecory` der Klasse `DiscreteLinearizedSystem` erzeugt.