4 Regelung eines Roboters

4.1 Einleitung und Organisation Übung 4

In der dritten Übung wurden die Grundlagen der kinematischen und dynamischen Modellierung und Simulation eines Roboters behandelt. Ein Ziel dieser Übung ist es, mit dem 7-Achs-Roboter KUKA LBR iiwa 14 R820 eine Figur auf einem Whiteboard mithilfe eines Stiftes zu zeichnen. Eine schematische Darstellung dieser Aufgabe zeigt Abbildung 4.1. Für diese Aufgabe werden Simulations- und Visualisierungsmodelle des Roboters zur Verfügung gestellt und die Experimente können auch am praktischer Versuchsaufbau am Institut ausgeführt werden. In den nachfolgenden Abschnitten werden die einzelnen Komponenten, die für das Erreichen des Ziels notwendig sind, erarbeitet und schließlich zusammengesetzt.

Zunächst werden in Abschnitt 4.2 das vollständige Robotermodell, die Koordinatensysteme und Parameter vorgestellt. Anschließend wird die Trajektorie für die Bewegung der Stiftspitze in Abschnitt 4.3 berechnet und vom Arbeitsraum in den Konfigurationsraum mithilfe der inversen Kinematik umgerechnet. In Abschnitt 4.4 werden unterschiedliche Regelungsstrategien entworfen und deren Genauigkeit und Fehlerverhalten anhand der berechneten Trajektorie verglichen.

Schließlich enthalten die Abschnitte 4.5 und 4.6 noch weitere Aufgaben für die freiwillige Vertiefung der Robotik-Grundlagen. In diesen Abschnitten wird der Einfluss von Sensorrauschen und das Verhalten von Bewegungen im Nullraum untersucht.

Studieren Sie zur Vorbereitung mindestens folgende Kapitel aus den Skripten:

- Skriptum zur VU Automatisierung (WS 2020/21) [1]
  - Kapitel 6.1 – 6.3
  - Kapitel 8.1 – 8.2
- Skriptum zur VU Fachvertiefung: Automatisierungs- und Regelungstechnik (WS 2020/21) [2]
  - Kapitel 2, 3 und 4, vollständig

Alle Dateien, die zum Bearbeiten dieser Übung benötigt werden, finden Sie in `uebung4.zip` auf der Homepage der Lehrveranstaltung. Bitte verändern Sie die Ordnerstruktur nicht und arbeiten Sie in den folgenden vier Dateien: `UE4_Regelung.slx` (nachfolgend mit SIMULINK-Datei bezeichnet), `Parameter.m` (nachfolgend Parameterdatei genannt), `Pfad.m` und `Zeitparametrierung.m`.

Bei Fragen oder Anregungen zu dieser Übung wenden Sie sich bitte an

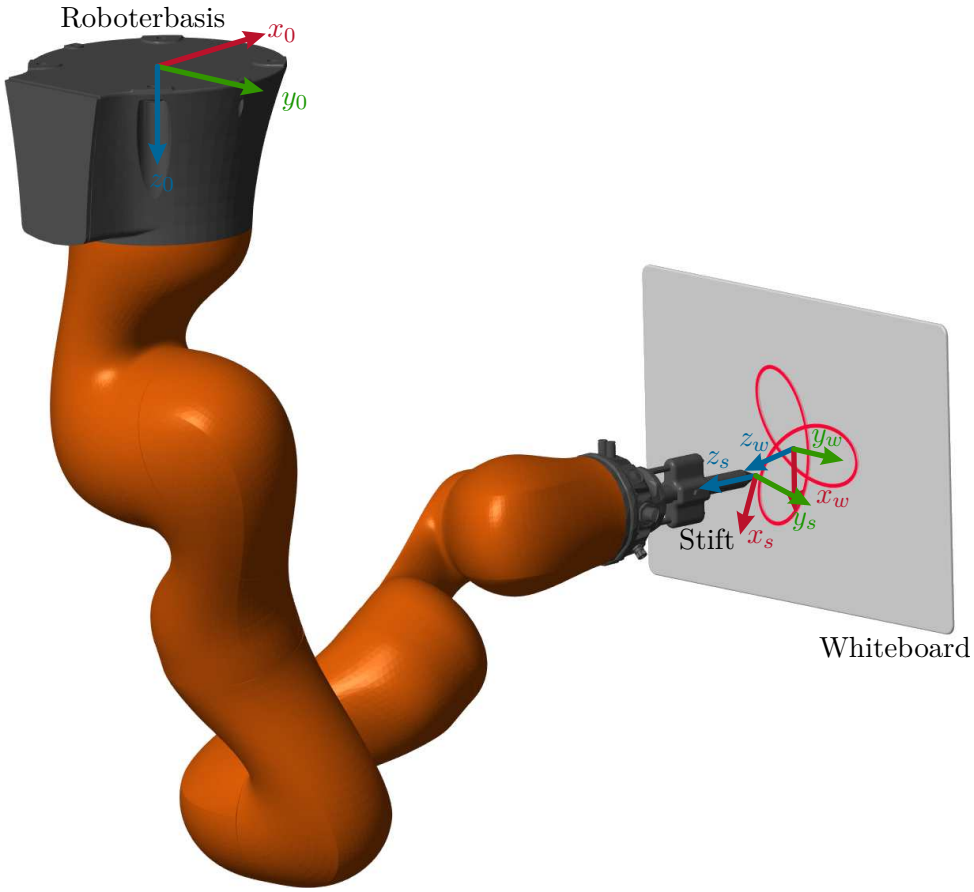

Abbildung 4.1: Visualisierung des Laborversuchs der Übung 4. Mit dem Roboter KUKA LBR iiwa 14 R820 wird eine Figur auf ein Whiteboard gezeichnet.

- Christoph Unger <unger@acin.tuwien.ac.at> oder
- Lucia Lauxmann <lauxmann@acin.tuwien.ac.at>.

4.2 Modell des 7-Achs-Roboters

In Abbildung 4.2 ist der Roboter KUKA LBR iiwa 14 R820 mit den zugehörigen Koordinatensystemen dargestellt. Darin bezeichnet $(0_0x_0y_0z_0)$ das Koordinatensystem der Roboterbasis, $(0_sx_sy_sz_s)$ das Stift-Koordinatensystem und $(0_wx_wy_wz_w)$ den Ursprung und die Lage des Whiteboards. Die Modellierung der Kinematik und der Dynamik des 7-Achs-Roboters wurde analog zur vorigen Übung durchgeführt. Es ist damit das Robotermodell in der Form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{4.1}$$

gegeben, wobei die Massenmatrix $\mathbf{M}(\mathbf{q})$, die Coriolis-Matrix $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ und der Gravitationsvektor $\mathbf{g}(\mathbf{q})$ als MATLAB-Funktionen zur Verfügung stehen. Zusätzlich werden die homogene Transformation $\mathbf{H}_0^s(\mathbf{q})$ des Stift-Koordinatensystems $(0_sx_sy_sz_s)$ in Bezug auf das

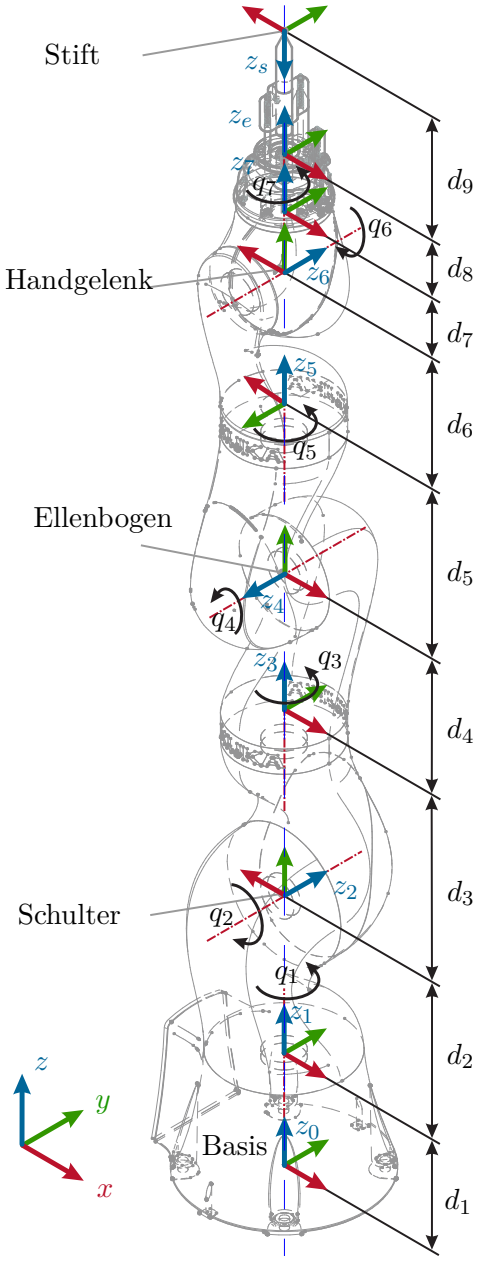

Abbildung 4.2: Schematische Darstellung und Koordinatensysteme des KUKA LBR iiwa 14 R820.

Roboterbasis-Koordinatensystem ($0_0x_0y_0z_0$), die geometrische Manipulator Jacobi-Matrix $\mathbf{J}_0^s(\mathbf{q})$ sowie deren zeitliche Ableitung $\dot{\mathbf{J}}_0^s(\mathbf{q})$ bereitgestellt. Eine Liste aller mathematischen Symbole und deren Bezeichner in der MATLAB-Implementierung findet sich in Tabelle 4.1.

4.3 Trajektorienplanung

Für den Kontaktpunkt des Stiftes auf dem Whiteboard wird eine Trajektorie benötigt, die als Sollvorgabe für die Roboterbewegung dient und zumindest zweifach stetig differenzierbar ist. In den nachfolgenden Unterkapiteln wird ein Trajektoriengenerator entwickelt, die Koordinatentransformation vom Whiteboard-Koordinatensystem zum Roboter-Basis-Koordinatensystem berechnet und die inverse Kinematik numerisch gelöst.

In der bereitgestellten SIMULINK-Datei sind die Subsysteme *Trajektorienplanung* (grün), *Regelung* (orange) und *Strecke und Visualisierung* (cyan) enthalten. Beim Ausführen der SIMULINK-Datei wird automatisch die Parameterdatei aufgerufen, um alle benötigten Parameter zu laden. Für die nachfolgenden Aufgaben wird nur das Subsystem *Trajektorienplanung* benötigt.

Hinweis: In der SIMULINK-Datei ist eine Visualisierung des Roboters implementiert, um Ihnen die Interpretation der Roboterbewegung zu erleichtern. Für eine kürzere Simulationszeit können Sie den Block *Visualisierung* im Subsystem *Strecke und Visualisierung* auskommentieren.

4.3.1 Trajektoriengenerator im Whiteboard-Koordinatensystem

Das Ziel der Trajektoriengenerierung ist die mathematische Beschreibung der Trajektorie der Stiftspitze, ausgedrückt im Whiteboard-Koordinatensystem ($0_wx_wy_wz_w$) durch eine gewünschte Position $\mathbf{p}(t)$ als Funktion der Zeit t. Die Trajektorie $\mathbf{p}(t) = \mathbf{x}(s(t))$ wird durch den Pfad $\mathbf{x}(s)$ und die Zeitparametrierung $s(t)$ zusammengesetzt.

Aufgabe 4.1 (Zeitparametrierung). Entwerfen Sie eine Zeitparametrierung $s(t)$ als Polynom 3. Ordnung mit den Randbedingungen

$$s(0) = 0 \qquad s(T) = 2\pi \tag{4.2}$$
$$\dot{s}(0) = 0 \qquad \dot{s}(T) = 0\ . \tag{4.3}$$

1. Berechnen Sie die Koeffizienten des Polynoms in MAPLE.
2. Implementieren Sie die Zeitparametrierung $s(t)$ sowie die beiden Zeitableitungen $\dot{s}(t)$ und $\ddot{s}(t)$ in der vorgegebenen MATLAB-Datei `Zeitparametrierung.m`. Verwenden Sie dazu die Korrespondenzen aus Tabelle 4.1.

Als Pfad für die Roboterbewegung wird ein sogenanntes *Hypotrochoid* verwendet. Ein Beispiel dafür ist als rote Linie in Abbildung 4.3 gezeigt. Hypotrochoide beschreiben eine Abrollbewegung (Berührpunkt A in Abbildung 4.3), bei der sich ein innerer Kreis (gelb) mit dem Radius r innerhalb eines äußeren Kreises (grau) mit dem Radius R bewegt. Der Pfad wird durch einen Punkt relativ zu dem inneren Kreis mit dem Abstand d beschrieben.

Symbol	MATLAB-Implementierung	Symbol	MATLAB-Implementierung
t	`t`	$\mathbf{q}_0$	`q_0`
T	`par.T`	$\dot{\mathbf{q}}_0$	`q_0_p`
r	`par.r`	$\ddot{\mathbf{q}}_0$	`q_0_pp`
R	`par.R`	$\mathbf{q}_d$	`q_d`
d	`par.d`	$\dot{\mathbf{q}}_d$	`q_d_p`
$\boldsymbol{\tau}$	`tau`	$\ddot{\mathbf{q}}_d$	`q_d_pp`
m_{ext}	`m_ext`	s	`s`
$\mathbf{H}_0^w$	`par.H_0_w`	$\dot{s}$	`s_p`
$\mathbf{x}$	`x`	$\ddot{s}$	`s_pp`
$\mathbf{x}'$	`x_s`	$\mathbf{p}$	`p`
$\mathbf{x}''$	`x_ss`	$\dot{\mathbf{p}}$	`p_p`
k_0	`parreg.ik.k_0`	$\ddot{\mathbf{p}}$	`p_pp`
$\mathbf{K}_0$	`parreg.ik.K_0`	$\mathbf{x}_d$	`x_d`
$\mathbf{K}_1$	`parreg.ik.K_1`	$\dot{\mathbf{x}}_d$	`x_d_p`
$\bar{\mathbf{q}}$	`param_robot.q_mean`	$\ddot{\mathbf{x}}_d$	`x_d_pp`
$\mathbf{q}_{min}$	`param_robot.q_min`	$\mathbf{e}_q$	`e_q`
$\mathbf{q}_{max}$	`param_robot.q_max`	$\dot{\mathbf{e}}_q$	`e_q_p`
		$\int \mathbf{e}_q \, \mathrm{d}t$	`e_q_I`
$\mathbf{q}$	`robot_sensors.q`, `robot_model.q`		
$\dot{\mathbf{q}}$	`robot_sensors.q_p`, `robot_model.q_p`		
$\ddot{\mathbf{q}}$	`robot_sensors.q_pp`, `robot_model.q_pp`		
$\mathbf{M}(\mathbf{q})$	`robot_model.M` `massenmatrix(q,param_robot)`		
$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$	`robot_model.C` `coriolismatrix(q,q_p,param_robot)`		
$\mathbf{g}(\mathbf{q})$	`robot_model.g` `gravitationsvektor(q,param_robot)`		
$\mathbf{J}_0^s(\mathbf{q})$	`robot_model.J` `jacobimatrix_geometrisch_stift(q,param_robot)`		
$\dot{\mathbf{J}}_0^s(\mathbf{q},\dot{\mathbf{q}})$	`robot_model.J_p` `jacobimatrix_geometrisch_stift_p(q,q_p,param_robot)`		
$\mathbf{H}_0^s(\mathbf{q})$	`robot_model.H` `homogene_transformation_stift(q,param_robot)`		

Tabelle 4.1: Mathematische Symbole und die zugehörigen Bezeichner in der MATLAB-Implementierung.

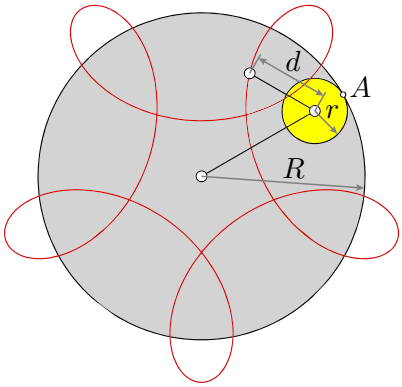

Abbildung 4.3: Beispiel eines Hypotrochoids $R = 5r$, $d = \frac{20}{9}r$ [3].

Somit kann durch die drei Parameter R, r und d eine Vielzahl unterschiedlicher Pfade generiert werden. Eine interaktive Visualisierung dieser Kurven ist in [4] zu finden. Die mathematische Beschreibung eines Hypotrochoids $\mathbf{x}(s)$ in der xy-Ebene lautet

$$\mathbf{x}(s) = \begin{bmatrix} (R-r)\sin(s) - d\sin\left(\frac{R-r}{r}s\right) \\ (R-r)\cos(s) + d\cos\left(\frac{R-r}{r}s\right) \\ 0 \end{bmatrix} \tag{4.4}$$

mit der Pfadposition $s \in [0, 2\pi]$.

Aufgabe 4.2 (Trajektoriengenerator). Berechnen und implementieren Sie den Trajektoriengenerator anhand folgender Punkte:

1. Berechnen Sie die Ableitungen $\mathbf{x}'(s)$ und $\mathbf{x}''(s)$ bezüglich des Pfadparameters s.
2. Implementieren Sie $\mathbf{x}(s)$, $\mathbf{x}'(s)$ und $\mathbf{x}''(s)$ in der vorgegebenen MATLAB-Datei `Pfad.m`. Verwenden Sie dazu die Korrespondenzen aus Tabelle 4.1.
3. Berechnen Sie die Trajektorie $\mathbf{p}(t)$, $\dot{\mathbf{p}}(t)$ und $\ddot{\mathbf{p}}(t)$ aufbauend auf der Zeitparametrierung $s(t)$ aus Aufgabe 4.1 und dem Pfad $\mathbf{x}(s)$ sowie deren Ableitungen. Implementieren Sie Ihre Lösung im Subsystem *Trajektoriengenerator* in der vorgegebenen SIMULINK-Datei.
4. Testen und visualisieren Sie die Trajektorie $\mathbf{p}(t)$ mithilfe des Blocks *XY Graph* in SIMULINK.
5. Experimentieren Sie mit den Parametern. Für welche Wahl der Parameter beschreibt die Trajektorie einen Kreis? Für welche Wahl der Parameter beschreibt die Trajektorie eine Linie auf der y_w-Achse?

> **Hinweis:** Das Whiteboard ist 420 mm breit und 297 mm hoch. Beachten Sie diese Maße, damit Ihre Trajektorie das Whiteboard nicht verlässt.

4.3.2 Trajektorie im Roboterbasis-Koordinatensystem

Um die Trajektorie mit dem Stift am Endeffektor des Roboters abzufahren, muss die Trajektorie im nächsten Schritt im Roboter-Basis-Koordinatensystem mithilfe einer homogenen Transformation $\mathbf{H}_0^w$ dargestellt werden. Die Anordnung der Koordinatensysteme sind in Abbildung 4.1 dargestellt. Aus dieser Abbildung können Sie Verdrehung der Koordinatensysteme herauslesen. Der Abstand vom Roboterbasis-Koordinatensystem zum Whiteboard-Koordinatensystem ist durch die Längen $l_x = 0.8\,\text{m}$ entlang der x_0-Achse, $l_y = 0\,\text{m}$ entlang y_0 und $l_z = 0.5\,\text{m}$ in Richtung der z_0-Achse gegeben.

Zusätzlich zu den Positionen werden für die Beschreibung der Pose auch die Orientierungen des Stifts im dreidimensionalen Raum benötigt. Die Parametrierung der Orientierung soll dabei in Form von Euler-Winkeln mit der selben Orientierung wie das Whiteboard-Koordinatensystems gewählt werden.

Aufgabe 4.3 (Koordinatentransformation). Gehen Sie bei der Bearbeitung der Aufgabe folgendermaßen vor:

1. Bestimmen Sie die homogene Transformation $\mathbf{H}_0^w$, welche die Position und Orientierung des Whiteboard-Koordinatensystems $(0_w x_w y_w z_w)$ in Bezug auf das Roboterbasis-Koordinatensystem $0_0 x_0 y_0 z_0)$ beschreibt. Tragen Sie die Matrix $\mathbf{H}_0^w$ in der Parameterdatei als `par.H_0_w` ein.

2. Die homogene Transformation $\mathbf{H}_w^t(t)$ beschreibt nun die gewünschte Position und Orientierung der Stiftspitze in Bezug auf das Whiteboard-Koordinatensystem. Bestimmen Sie $\mathbf{H}_w^t(t)$ unter Verwendung der Trajektorie $\mathbf{p}(t)$, sowie die zeitlichen Ableitungen $\dot{\mathbf{H}}_w^t(t)$ und $\ddot{\mathbf{H}}_w^t(t)$.

   > **Hinweis:** Überlegen Sie sich, welche Einträge in $\mathbf{H}_w^t(t)$ konstant sind und beim Ableiten wegfallen.

3. Berechnen Sie die homogene Transformation $\mathbf{H}_0^t(t)$ und stellen Sie damit die Trajektorie in Bezug auf das Roboterbasis-Koordinatensystem $(0_0 x_0 y_0 z_0)$ dar. Bestimmen Sie weiters die zeitlichen Ableitungen $\dot{\mathbf{H}}_0^t(t)$ und $\ddot{\mathbf{H}}_0^t(t)$.

4. Berechnen Sie die gewünschte Pose $\mathbf{x}_d$ sowie die beiden Ableitungen $\dot{\mathbf{x}}_d$ und $\ddot{\mathbf{x}}_d$ des Stiftes aus der homogenen Transformation $\mathbf{H}_0^t(t)$ und deren Ableitungen. Verwenden Sie für die Parametrierung der Orientierungen die *klassischen* Euler-Winkel.

5. Implementieren Sie Ihre Gleichungen in der SIMULINK-Datei. Erstellen Sie dazu eine *Matlab Function* im Subsystem *Trajektorienplanung* und verwenden Sie dafür den Funktionskopf

`function [x_d, x_d_p, x_d_pp] = fcn(p, p_p, p_pp, par)`.

6. Testen und visualisieren Sie die Trajektorie $\mathbf{x}_d(t)$ in SIMULINK.

4.3.3 Inverse Kinematik

Mithilfe der inversen Kinematik wird in diesem Abschnitt aus dem zeitlichen Verlauf der Trajektorie im Arbeitsraum der zugehörige zeitliche Verlauf im Konfigurationsraum berechnet. Für viele Regelungskonzepte werden aber nicht nur die Position und Geschwindigkeit im Konfigurationsraum benötigt, sondern auch die Beschleunigungen. Daher soll im Folgenden die differenzielle inverse Kinematik zweiter Ordnung für den kinematisch redundanten 7-Achs-Roboter implementiert werden.

Aufgabe 4.4 (Differenzielle inverse Kinematik). Berechnen Sie die benötigten Ausdrücke für die differenzielle inverse Kinematik zweiter Ordnung für den 7-Achs-Roboter und implementieren Sie diese als *Matlab Function* im Subsystem *Trajektorienplanung* in der SIMULINK-Datei. Verwenden Sie dazu die Korrespondenzen aus Tabelle 4.1. Bearbeiten Sie dazu folgende Punkte:

1. Berechnen Sie die Pose des Stiftes $\mathbf{x}_e$ sowie die analytische Manipulator Jacobi-Matrix des Stiftes $\mathbf{J}_A(\mathbf{q})$ unter Verwendung der bereitgestellten Funktion $\mathbf{H}_0^s(\mathbf{q})$ und der bereitgestellten geometrischen Manipulator Jacobi-Matrix $\mathbf{J}_0^s(\mathbf{q})$.

2. Berechnen Sie die zeitliche Ableitung der Pose des Stiftes $\dot{\mathbf{x}}_e$ und der analytischen Manipulator Jacobi-Matrix $\dot{\mathbf{J}}_A(\mathbf{q}, \dot{\mathbf{q}})$ mithilfe der bereitgestellten Funktion $\dot{\mathbf{J}}_0^s(\mathbf{q}, \dot{\mathbf{q}})$.

3. Aufbauend auf den berechneten Ausdrücken, implementieren Sie nun die differenzielle inverse Kinematik zweiter Ordnung für kinematisch redundante Roboter, um aus der vorgegebenen Trajektorie im Arbeitsraum $(\mathbf{x}_d(t), \dot{\mathbf{x}}_d(t), \ddot{\mathbf{x}}_d(t))$ die gewünschte Trajektorie im Konfigurationsraum $(\mathbf{q}_d(t), \dot{\mathbf{q}}_d(t), \ddot{\mathbf{q}}_d(t))$ zu berechnen.

   Hinweis: Die Anfangswerte der Gelenkswinkel $\mathbf{q}_0$, -geschwindigkeiten $\dot{\mathbf{q}}_0$ und -beschleunigungen $\ddot{\mathbf{q}}_0$ werden in der Parameterdatei automatisch berechnet.

4. Stabilisieren Sie die Bewegung des Roboters im Nullraum mithilfe des Kriteriums, siehe [2]

$$w(\mathbf{q}) = -\frac{1}{2n} \sum_{i=1}^{n} \left(\frac{q_i - \bar{q}_i}{q_{i,max} - q_{i,min}} \right)^2 \tag{4.5}$$

5. Wählen Sie einen geeigneten Wert für k_0 und geeignete Diagonalmatrizen $\mathbf{K}_1$ und $\mathbf{K}_0$. Testen Sie Ihre Wahl mit unterschiedlichen Trajektorien.

> **Hinweis:** Für die visuelle Darstellung verbinden Sie die berechnete Trajektorie ($\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$) mit den *Outport*-Blöcken `q_d`, `q_d_p`, `q_d_pp`, indem Sie die bereits verbundenen *Constant*-Blöcke löschen. In Ihrem MATLAB-Fenster sollte nun im *Mechanics Explorer* die Visualisierung des Roboters und Ihrer implementierten Trajektorie sichtbar sein.

4.4 Roboterregelungen

Mithilfe von verschiedenen Reglern soll der Stift am Endeffektor des Roboters die berechnete Trajektorie aus Kapitel 4.3 nachfahren. In diesem Abschnitt sollen eine Kaskadenregelung, ein PD-Regler und ein Computed-Torque-Regler implementiert und in der Simulation getestet werden. Dabei sind die Stellgrößen für den Regler die Drehmomente an den Gelenken des Roboters $\boldsymbol{\tau}$ (siehe SIMULINK-Datei im Subsystem *Regelung*) und die Eingänge des Reglers sind die gewünschten und tatsächlichen Gelenkswinkel $\mathbf{q}_d$ bzw. $\mathbf{q}$, sowie die Gelenksgeschwindigkeiten und -beschleunigungen. Durch die Sensorik im KUKA LBR iiwa 14 R820 sind die tatsächlichen Gelenkswinkel, -geschwindigkeiten und -beschleunigungen mit Rauschen behaftet. In der Simulation wird bandbreitenlimitiertes weißes Rauschen verwendet, um diesen Effekt abzubilden. Beachten Sie bei der Reglerimplementierung die maximal zulässigen Drehmomente des Roboters

$$\boldsymbol{\tau}_{max} = \begin{bmatrix} 320\,\mathrm{N\,m} & 320\,\mathrm{N\,m} & 176\,\mathrm{N\,m} & 176\,\mathrm{N\,m} & 110\,\mathrm{N\,m} & 40\,\mathrm{N\,m} & 40\,\mathrm{N\,m} \end{bmatrix}^{\mathrm{T}}. \tag{4.6}$$

Die Zustände, die wichtigsten Funktionen und Matrizen des 7-Achs-Roboters werden in den Bussen `robot_sensors` und `robot_model` zur Verfügung gestellt, siehe Tabelle 4.1.

> **Hinweis:** Schalten Sie für den Entwurf aller Regelungen das Rauschen ein, indem Sie die Variable `Rauschen` in Ihrer SIMULINK-Datei von 0 auf 1 setzen.

4.4.1 Kaskadenregelung

Eine einfache Möglichkeit der Regelung eines Roboters besteht in der dezentralen Regelung der einzelnen Achsen (siehe Fachvertiefungsskript [2] Kapitel 4.1.2). Hierbei ist wenig Wissen über das Robotermodell notwendig, da die Kopplungen der einzelnen Roboterglieder zueinander als Störungen interpretiert werden. Als Entwurfsmodell wird das entkoppelte Modell des Roboters

$$\ddot{\mathbf{q}} = \begin{bmatrix} 0.521 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5.193 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.188 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.882 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.024 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.016 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.001 \end{bmatrix}^{-1} \boldsymbol{\tau} \tag{4.7}$$

verwendet. Die Einträge der Diagonalmatrix in Gleichung (4.7) sind Abschätzungen der maximalen Massenträgheitsmomente für jede Achse.

In diesem Kapitel soll die dezentrale Regelung mit einem hochdynamischen unterlagerten Geschwindigkeitsregler und einem überlagerten PI-Positionsregler realisiert werden. Die Implementierung erfolgt dabei in zwei Schritten. Für die unterlagerte Geschwindigkeitsregelung soll ein Zustandsregler verwendet werden. Der Zustandsregler wird durch die Gleichungen

$$\mathbf{x}_{I,k+1} = \mathbf{x}_{I,k} + (\dot{\mathbf{q}}_{d,k} - \dot{\mathbf{q}}_k) \ , \qquad \mathbf{x}_{I,0} = \mathbf{0} \tag{4.8a}$$

$$\boldsymbol{\tau}_k = \mathbf{K}_x \dot{\mathbf{q}}_k + \mathbf{K}_I \mathbf{x}_{I,k} \ , \tag{4.8b}$$

beschrieben. Die Zustände $\mathbf{x}$, die gleichzeitig die Ausgänge $\mathbf{y}$ des unterlagerten Regelkreises darstellen, sind die tatsächlichen Gelenksgeschwindigkeiten $\mathbf{x} = \mathbf{y} = \dot{\mathbf{q}}$.

Aufgabe 4.5 (Geschwindigkeitsregler). Bearbeiten Sie für den Entwurf und die Implementierung des Geschwindigkeitsreglers folgende Punkte:

1. Entwerfen Sie einen Zustandsregler für die Regelung der Gelenksgeschwindigkeiten mit MATLAB in der Parameterdatei. Dimensionieren Sie die Regelparameter so, dass der geschlossene Regelkreis die Pole $p_j = \exp(\lambda_j T_a)$, $j = 1, \ldots, 14$, mit $\mathrm{Re}(\lambda_j) < 0$ besitzt. Hierbei können die zwei gewählten Pole einer Achse für alle Achsen übernommen werden. Diskretisieren Sie dazu das Modell mit einer Abtastzeit T_a =125 µs. Verwenden Sie für die diagonalen Parametermatrizen des Reglers die Struktur `parreg.kr.kx_gr` als Proportionalverstärkung bzw. `parreg.kr.ki_gr` als Integralverstärkung, die Sie in der Parameterdatei wählen.

   Hinweis: Beachten Sie, dass die Geschwindigkeitsregler aller Roboterachsen nur dann entkoppelt sind, wenn Diagonalmatrizen für $\mathbf{K}_x$ und $\mathbf{K}_I$ in Gleichung (4.8) gewählt werden. Verwenden Sie Pole-Placement für die Wahl der Reglermatrizen.

2. Erstellen Sie in der SIMULINK-Datei im Subsystem *Regelung* eine *Matlab Function* mit der Abtastzeit T_a =125 µs. Verwenden Sie als Funktionskopf folgende Zeile

   ```
   function [tau, e_q_p] = fcn(q_d_p, robot_model, parreg) .
   ```

   Implementieren Sie in dieser *Matlab Function* den unterlagerten Geschwindigkeitsregler nach Gleichung (4.8). Verbinden Sie den Ausgang des Reglers mit dem Ausgang `tau` des Subsystems *Regelung*, um die berechneten Drehmomente auf die Strecke aufzuschalten. Aktivieren Sie die Vorgabe von Drehmomenten als Eingänge des Modells, indem Sie in Ihrer SIMULINK-Datei im Feld `Ansteuerung` die Option `tau` wählen.

3. Testen Sie den Geschwindigkeitsregler, indem die Startkonfiguration des Robo-

ters zu Beginn der Simulation mit einer gewünschten Gelenksgeschwindigkeit von $\dot{\mathbf{q}}_d = \mathbf{0}$ gehalten wird. Schalten Sie außerdem einen Sprung von $\dot{q}_d = 1\,\mathrm{rad/s}$ in der Startkonfiguration auf die gewünschte Gelenksgeschwindigkeit separat für jede Achse auf und untersuchen Sie das Einschwingverhalten des Regelfehlers $\dot{\mathbf{e}}_q = \dot{\mathbf{q}} - \dot{\mathbf{q}}_d$. Wählen Sie die Pole des Reglers so, dass die Anstiegszeit der Gelenksgeschwindigkeit $\dot{\mathbf{q}}$ weniger als 50 ms beträgt und die maximalen Drehmomente des Roboters aus Gleichung (4.6) nicht überschritten werden.

Durch eine geeignete Wahl der Eigenwerte des unterlagerten Geschwindigkeitsreglers kann der Roboter durch das kinematische Robotermodell

$$\dot{\mathbf{q}} = \mathbf{u} \tag{4.9}$$

beschrieben werden. Für den überlagerten Positionsregler soll ein PI-Regler der Form

$$\mathbf{u} = -\mathbf{K}_p(\mathbf{q} - \mathbf{q}_d) - \mathbf{K}_i \int (\mathbf{q} - \mathbf{q}_d)\,\mathrm{d}t \tag{4.10}$$

entworfen werden, um auf die gewünschten Gelenkswinkel $\mathbf{q}_d$ zu regeln.

Aufgabe 4.6 (Positionsregler). Bearbeiten Sie für den Entwurf und die Implementierung des Positionsreglers folgende Punkte:

1. Entwerfen Sie einen PI-Regler nach Gleichung (4.10) für die Regelung der Gelenkswinkel mit MATLAB in der Parameterdatei. Achten Sie bei der Dimensionierung des Reglers auf die Wahl der Eigenwerte im Vergleich zum Geschwindigkeitsregler von Gleichung (4.8). Verwenden Sie für die diagonalen Parametermatrizen des Reglers die Struktur `parreg.kr.kp_pr` als Proportionalverstärkung bzw. `parreg.kr.ki_pr` als Integralverstärkung, die Sie in der Parameterdatei wählen.

   Hinweis: Verwenden Sie Pole-Placement für die Wahl der Reglermatrizen, siehe Kapitel 4.1.4 im Vorlesungsskriptum [2]. Hierbei kann $\dot{\mathbf{q}}_d = \mathbf{0}$ angenommen werden.

2. Erstellen Sie für die Implementierung in der SIMULINK-Datei im Subsystem *Regelung* eine weitere *Matlab Function* mit der Abtastzeit $T_a = 125\,\mu\mathrm{s}$. Verwenden Sie als Funktionskopf folgende Zeile

   ```
   function [q_d_p, e_q] = fcn(e_q_I, q_d, robot_model, parreg).
   ```

   Kombinieren Sie den Positionsregler mit Ihrem Geschwindigkeitsregler.

3. Testen Sie Ihre gesamte Kaskadenregelung mit der in den vorigen Aufgaben entworfenen Trajektorie. Überprüfen Sie Ihre Reglerparameter außerdem, indem Sie ausgehend von der Startkonfiguration des Roboters einen Sprung in der Höhe von $q_d = 0.1\,\mathrm{rad}$ auf den gewünschten Gelenkswinkel für jede Achse separat aufschalten. Simulieren Sie das Einschwingverhalten des Regelfehlers $\mathbf{e}_q = \mathbf{q} - \mathbf{q}_d$. Wählen Sie die Reglerparameter so, dass die Anstiegszeit der

Sprungantworten der Gelenkswinkel $\mathbf{q}$ weniger als 500 ms beträgt und die maximalen Drehmomente des Roboters aus Gleichung (4.6) nicht überschritten werden.

4.4.2 PD-Regelung mit Gravitationskompensation

Eine einfache *zentrale* Regelungsstrategie in Gelenkskoordinaten für den Roboter ist die PD-Regelung mit Gravitationskompensation

$$\boldsymbol{\tau} = \mathbf{g}(\mathbf{q}) - \mathbf{K}_d \dot{\mathbf{e}}_q - \mathbf{K}_p \mathbf{e}_q \ , \tag{4.11}$$

welche Sie im Rahmen der Vorlesung kennengelernt haben (siehe Kapitel 4.1.3 des Vorlesungsskriptums [2]).

Aufgabe 4.7 (PD-Regelung mit Gravitationskompensation). Bearbeiten Sie folgende Punkte für den Entwurf und die Implementierung der PD-Regelung mit Gravitationskompensation:

1. Erstellen Sie in der SIMULINK-Datei im Subsystem *Regelung* eine weitere *Matlab Function* mit der Abtastzeit $T_a = 125\,\mu s$. Implementieren Sie in dieser `Matlab Function` das PD-Regelgesetz nach Gleichung (4.11). Benutzen Sie als Funktionskopf folgende Zeile

   ```
   function [tau, e_q, e_q_p] = fcn(q_d, q_d_p, robot_model, par-
   reg).
   ```

   Verwenden Sie für die Parametermatrizen des Reglers die Struktur `parreg.pd.kp` bzw. `parreg.pd.kd`, die Sie in der Parameterdatei hinzufügen.

   Hinweis: Zum Umschalten zwischen den verschiedenen Ausgängen Ihrer Regler mit dem Outport-Block `tau` des Subsystems *Regelung* können Sie Umschaltblöcke wie z. B. `Manual Switch` oder `Multiport Switch` verwenden.

2. Wählen Sie geeignete Parameter für die Diagonalmatrizen $\mathbf{K}_p$ und $\mathbf{K}_d$ und testen Sie Ihre Wahl für die entworfene Trajektorie. Simulieren Sie den geschlossenen Regelkreis mit einem Anfangsregelfehler von $e_{q,i} = q_i - q_{d,i} = \Delta q_d = 0.1\,\text{rad}$ von der Startkonfiguration für jede Achse i separat und untersuchen Sie das Einschwingverhalten des Regelfehlers $\mathbf{e}_q = \mathbf{q} - \mathbf{q}_d$. Wählen Sie die Reglerparameter so, dass die Anstiegszeit der Sprungantworten der Gelenkswinkel $\mathbf{q}$ weniger als 500 ms betragen soll und die maximalen Drehmomente des Roboters aus Gleichung (4.6) nicht überschritten werden. Interpretieren Sie Ihre Ergebnisse!

4.4.3 Computed-Torque-Regelung

Mit der Computed-Torque-Regelung in der Form

$$\boldsymbol{\tau} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_d(t) - \mathbf{K}_d\dot{\mathbf{e}}_q - \mathbf{K}_p\mathbf{e}_q) \tag{4.12}$$

kann die Bewegung des Roboters in Gelenkskoordinaten auf einer gewünschten Solltrajektorie ($\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$) stabilisiert werden (siehe Kapitel 4.1.4 im Vorlesungsskriptum [2]).

Aufgabe 4.8 (Computed-Torque-Regelung). Bearbeiten Sie für die Computed-Torque-Regelung folgende Punkte:

1. Erstellen Sie in der SIMULINK-Datei im Subsystem *Regelung* eine weitere *Matlab Function* mit der Abtastzeit $T_a = 125\,\mu s$. Implementieren Sie den Computed-Torque-Regler nach Gleichung (4.12). Benutzen Sie als Funktionskopf folgende Zeile

   `function [tau, e_q, e_q_p] = fcn(q_d, q_d_p, q_d_pp, robot_model, parreg)`.

   Verwenden Sie für die Parametermatrizen des Reglers die Struktur `parreg.ct.kp` bzw. `parreg.ct.kd`, die Sie in der Parameterdatei wählen.

2. Wählen Sie geeignete Eigenwerte zur Berechnung der Parameter für die Diagonalmatrizen $\mathbf{K}_p$ und $\mathbf{K}_d$ und testen Sie Ihre Wahl für die entworfene Trajektorie. Simulieren Sie den geschlossenen Regelkreis mit einem Anfangsregelfehler von $e_{q,i} = q_i - q_{d,i} = \Delta q_d = 0.1\,\text{rad}$ von der Startkonfiguration für jede Achse i separat und untersuchen Sie das Einschwingverhalten des Regelfehlers $\mathbf{e}_q = \mathbf{q}_d - \mathbf{q}$. Wählen Sie die Reglerparameter so, dass die Anstiegszeit der Sprungantworten der Gelenkswinkel $\mathbf{q}$ weniger als 500 ms betragen soll und die maximalen Drehmomente des Roboters aus Gleichung (4.6) nicht überschritten werden. Interpretieren Sie Ihre Ergebnisse!

4.4.4 Reglervergleich

Anhand von vier unterschiedlichen Szenarien sollen nachfolgend die Regelungsstrategien evaluiert und gegenübergestellt werden. Diese vier Szenarien sind die Stabilisierung eines Arbeitspunkts, das Folgen einer langsamen Trajektorie, das Folgen einer schnellen Trajektorie und das Hinzufügen einer für die Regelung unbekannten Last am Endeffektor. Für die Beurteilung der Trajektoriengeschwindigkeit sind die Geschwindigkeitslimits des Roboters für die einzelnen Achsen mit

$$\dot{\mathbf{q}}_{max} = \begin{bmatrix} 85\,°/s & 85\,°/s & 100\,°/s & 75\,°/s & 130\,°/s & 135\,°/s & 135\,°/s \end{bmatrix}^{\mathrm{T}} \tag{4.13}$$

einzuhalten.

Aufgabe 4.9 (Reglervergleich). Vergleichen Sie Ihre Implementierungen der Kaskadenregelung, der PD-Regelung und der Computed-Torque-Regelung in den vier folgenden Szenarien. Betrachten Sie dabei insbesondere den absoluten Fehler im Konfigurationsraum $\|\mathbf{q} - \mathbf{q}_d\|_2$ und im Arbeitsraum $\|\mathbf{x}_e - \mathbf{x}_d\|_2$.

Hinweis: Verwenden Sie den SIMULINK *Data Inspector*, um mehrere Simulationsläufe miteinander zu vergleichen. Speichern Sie weiters die relevanten Ansichten des *Data Inspectors* ab, um während der Abgabegespräche Zeit zu sparen.

1. **Arbeitspunkte:** Legen Sie dazu $t = 0$ am Eingang des Blocks *Zeitparametrierung* im Subsystem *Trajektorienplanung* an. Wechseln Sie zwischen Ihren Reglern und beurteilen Sie das Ergebnis.

2. **Langsame Trajektorie:** Verwenden Sie *Clock* als Eingang t des Blocks *Zeitparametrierung*. Stellen Sie sicher, dass für eine langsame Trajektorie die Gelenksgeschwindigkeiten 20 % der maximalen Geschwindigkeitslimits $\dot{\mathbf{q}}_{max}$ nicht überschreiten. Wechseln Sie zwischen Ihren Regelungsstrategien und beurteilen Sie das Ergebnis für alle implementierten Regelungen.

   Hinweis: Sie können die Geschwindigkeiten der Trajektorie über die Simulationsdauer `par.T`$= T$, über den Endwert der Zeitparametrierung $s(T)$ oder über die Form der Trajektorie variieren.

3. **Schnelle Trajektorie:** Stellen Sie sicher, dass für eine schnelle Trajektorie die Gelenksgeschwindigkeiten 40 % der maximalen Geschwindigkeitslimits $\dot{\mathbf{q}}_{max}$ nicht überschreiten. Wechseln Sie zwischen Ihren Regelungsstrategien und beurteilen Sie das Ergebnis für alle implementierten Regelungen.

4. **Unbekannte Last:** Stellen Sie in der SIMULINK-Datei die Variable `Lastmasse` von 0 kg auf 1 kg, um an dem letzten Gelenk ein externes Gewicht zu simulieren. Verwenden Sie weiters die Einstellungen der ersten Teilaufgabe „Arbeitspunkt". Wechseln Sie zwischen Ihren Regelungsstrategien und beurteilen Sie das Ergebnis für alle implementierten Regelungen.

4.5 Einfluss von Sensoren

Hinweis: Dieser Abschnitt ist eine **optionale** Aufgabe für interessierte Studierende.

In diesem Abschnitt wird die Sensorik des Roboters näher betrachtet. Der Roboter KUKA LBR iiwa 14 R820 verwendet Sinus-Cosinus-Encoder für die präzise Messung der Gelenkswinkel in jeder Achse. Aufgrund des Messprinzips ist das Positionssignal mit Rauschen behaftet. In der Robotersteuerung muss zusätzlich das Positionssignal abgeleitet werden, damit die Gelenksgeschwindigkeiten und -beschleunigungen für die Regelungen zur Verfügung stehen.

4.5.1 Differenzierfilter

Um das Positionssignal zu filtern und die zugehörigen Ableitungen zu berechnen kann ein Zustandsvariablenfilter verwendet werden. Das Zustandsvariablenfilter für eine einzelne Achse hat dabei die Form

$$\dot{\boldsymbol{\kappa}} = \mathbf{A}\boldsymbol{\kappa} + \mathbf{b}u \ , \qquad \boldsymbol{\kappa}(0) = \boldsymbol{\kappa}_0 \ , \tag{4.14}$$

wobei $\boldsymbol{\kappa} = \begin{bmatrix} \hat{q} & \dot{\hat{q}} & \ddot{\hat{q}} \end{bmatrix}^{\mathrm{T}}$ die gefilterten Werte und $u = q$ das ungefilterte Signal darstellt. Die Matrizen von Gleichung (4.14) werden dabei zu

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ a_0 \end{bmatrix} \tag{4.15}$$

gewählt, wodurch die Parameter durch Vorgabe der Eigenwerte bestimmt werden können.

Aufgabe 4.10 (Differenzierfilter). Bearbeiten Sie folgende Punkte für den Entwurf des Zustandsvariablenfilters:

1. Entwerfen Sie ein Zustandsvariablenfilter nach Gleichung (4.14), das das Positionssignal filtert und die ersten beiden zeitlichen Ableitungen berechnet. Führen Sie diesen Entwurf in der Parameter-Datei durch.
2. Implementieren Sie Ihr Filter in SIMULINK im Subsystem *Regelung* und berechnen Sie damit die gefilterten Signale $\hat{\mathbf{q}}$, $\dot{\hat{\mathbf{q}}}$ und $\ddot{\hat{\mathbf{q}}}$ aus `robot_sensors.q`. Implementieren Sie das Filter als *zeitdiskretes System* mit der Abtastzeit $T_a = 125\,\mu\mathrm{s}$.
3. Wählen Sie geeignete Eigenwerte für das Zustandsvariablenfilter. Testen Sie anschließend Ihr Filter auf Plausibilität und für unterschiedliche Eigenwerte.
4. Was ist bei der Wahl der Eigenwerte zu beachten?
5. Welche Auswirkungen hat das Differenzierfilter auf das Verhalten des geschlossenen Regelkreises mit Computed-Torque-Regelung?

4.6 Trajektorienplanung im Nullraum

Hinweis: Dieser Abschnitt ist eine **optionale** Aufgabe für interessierte Studierende.

Durch die kinematische Redundanz hat der Roboter einen zusätzlichen Freiheitsgrad zum Erreichen einer Pose im Raum. In folgender Aufgabe soll eine Regelung für diesen Freiheitsgrad implementiert werden. Ein möglicher Regler für den Nullraum lautet

$$\ddot{\mathbf{q}}_{n,d} = -\mathbf{K}_{d,n}\dot{\mathbf{q}} - \mathbf{K}_{p,n}(\mathbf{q} - \mathbf{q}_{d,n}) \ , \tag{4.16}$$

mit den Diagonalmatrizen $\mathbf{K}_{p,n}$ und $\mathbf{K}_{d,n}$ für den Proportional- bzw. Differentialanteil und den gewünschten Gelenkswinkeln im Nullraum $\mathbf{q}_{d,n}$.

Aufgabe 4.11 (Trajektorienplanung im Nullraum). Bearbeiten Sie folgende Punkte für die Implementierung der Trajektorienplanung im Nullraum:

1. Adaptieren Sie die Gleichung der differenziellen inversen Kinematik aus Aufgabe 4.4 indem Sie die Gleichung (4.16) des beschriebenen Nullraumreglers anstelle der partiellen Ableitung von (4.5) für $\ddot{\mathbf{q}}_{n,d}$ verwenden. Fügen Sie dafür einen zusätzlichen Eingang für die Vorgabe der gewünschten Nullraumbewegung in Ihrer Implementierung der inversen Kinematik hinzu. Verwenden Sie für die diagonalen Reglermatrizen die Parameterstruktur `parreg.nr.kp` bzw. `parreg.nr.kd`.

2. Wählen Sie geeignete Eigenwerte für die Berechnung der Diagonalmatrizen $\mathbf{K}_{p,n}$ und $\mathbf{K}_{d,n}$ und führen Sie die Berechnung in der Parameter-Datei durch.

3. Drehen Sie den Ellenbogen des Roboters von der Ausgangsposition um 180° und stabilisieren Sie ihn dort. Testen Sie Ihre Trajektorienplanung im Nullraum und stellen Sie mit den Reglermatrizen, bei einem Sprung von der Anfangsposition zur Endposition des Ellenbogens, eine Anstiegszeit von 1 s ein. Überprüfen Sie, ob die maximalen Drehmomente des Roboters aus Gleichung (4.6) eingehalten werden. Überlegen Sie sich, mit welchen Achsen Sie diese Nullraumbewegung vorgeben können. Sie können den Einfluss der einzelnen Achsen über die Einträge in den Reglermatrizen verändern.

4.7 Literatur

[1] A. Kugi, *Skriptum zur VU Automatisierung (WS 2020/2021)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2020.

[2] A. Kugi und W. Kemmetmüller, *Skriptum zur VU Fachvertiefung Automatisierungs- und Regelungstechnik (WS 2020/2021)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2020.

[3] J. Erven und D. Schwägerl, *Mathematik für Ingenieure*, 11. Aufl. Oldenbourg Verlag, 2011.

[4] desmos, *Visualisierung eines Hypotrochoids*, https://www.desmos.com/calculator/6eoux5udvr, 2021.