Computational Engineering und Robotik

Prof. Jan Peters, J. Carvalho, P. Klink, P. Liu, S. Stark

Sommersemester 2020

4. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

Hinweise zu dieser Übung

- Für die Teilnahme an der Übung ist eine Anmeldung beim **Moodle** notwendig. Dort sind auf der Kursseite zusätzliche Informationen zur Veranstaltung und die Regelungen zur Anrechnung der Übungsleistungen in die Endnote aufgeführt:

  `https://moodle.tu-darmstadt.de/course/view.php?id=20214`

- Von dieser Übung werden die **1. Aufgabe** und die **Programmieraufgabe** bewertet. Für die Bearbeitung der Programmieraufgabe lädst du den letzten Code vom *git repository* der im Moodle verlinkt ist.
- **Abgabe der schriftlichen Lösungen** zu den Übungsaufgaben bis **Montag, den 25.05.2019 um 11:59 Uhr** per Datei-Upload im Moodle.
- **Abgabe der Programmieraufgabe** bis **Montag, den 25.05.2019 um 11:59 Uhr** per Datei-Upload im Moodle.
- Lade im Moodle die Lösungen hoch als: **UebungXX.pdf** (z.B, für die Übung 04, Uebung04.pdf); **taskX.py** (z.B., für die zweite Programmieraufgabe task2.py). Gib deine Gruppenummer sowie die Namen und Matrikelnummern aller Beteiligten am Anfang der Datei an.
- Inhaltliche Fragen zu dieser Übung werden in den Tutoren-Sprechstunden und dem Frage- und Antwort-Forum im Moodle beantwortet.
- Stellt Fragen zu dieser Übung bitte im Moodle-Forum oder in den Sprechstunden.

1 Inverses Pendel (5 Punkte)

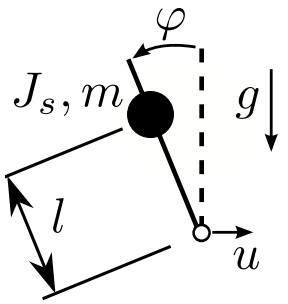

Abbildung 1: Modell des inversen Pendels

Das inverse Pendel ist in Abb. 1 dargestellt. Das Verhalten des Pendels lässt sich durch die Bewegungsgleichung

$$(J_s + ml^2)\ddot{\varphi} = ml(u\cos(\varphi) + g\sin(\varphi)) - c\dot{\varphi} \tag{1}$$

modellieren.

Hierbei bezeichnet l den Abstand des Schwerpunkts der Masse m des Pendels zur Drehachse. Die Trägheit des Pendels ist mit J_s, die viskose Reibkonstante mit c und die Gravitationsbeschleunigung mit g bezeichnet. Der Drehwinkel des Pendels ist mit φ angegeben. Zudem kann das Pendel mittels der horizontalen Beschleunigung u seiner Drehachse gesteuert werden.

1) Forme nun die Bewegungsgleichung in ein System erster Ordnung der Form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u)$ um.
2) Zunächst sollen die Ruhelagen des Pendels betrachtet werden.
   i) Bestimme dazu die Ruhelagen $\varphi_{s,i}$ des Pendels unter der Annahme einer Stellgröße $u_s = 0$.
   ii) Beschreibe (mit Worten) welchen Pendelkonfigurationen die erhaltenen Ruhelagen entsprechen.
3) Das in 1) erhaltene System ist nichtlinear. Da nichtlineare Systeme oft komplex zu regeln sind, soll das System nun um den Punkt $\boldsymbol{x_0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ linearisiert werden. Gib das linearisierte System um $\boldsymbol{x_0}$ an. Nimm dazu wieder an, dass keine Steuerung anliegt ($u_0 = 0$).
4) Abschließend soll ein konkretes System mit $l = 2.5\,\text{m}$, $J_s = 0.05\,\text{Nms}^2$, $m = 1.5\,\text{kg}$, $g = 9.81\,\text{m/s}^2$ und $c = 0.01\,\text{Nms}$ betrachtet werden. Bestimme die Eigenwerte der in 3) berechneten Systemdynamik mit den gegebenen Werten. Runde gegebenenfalls auf vier Nachkommastellen. Welche Aussage über die Stabilität kannst du anhand dieser Eigenwerte treffen? Warum kannst du mit den Eigenwerten etwas über die Stabilität aussagen?

Programmieraufgabe P2 Simulationsstudie 3D-Drucker (8 Punkte)

Im Rahmen der Simulationsstudie für den 3D-Drucker soll nun dessen dynamisches Verhalten genauer untersucht werden. Dafür kann das Modell des 3D-Druckers, wie in Abb. 2 dargestellt, vereinfacht werden. Die Kraftübertragung soll nur entlang der jeweiligen Motorachsen betrachtet werden. Kopplungseffekte durch Drehmomente zwischen den Achsen werden vernachlässigt. Die von den Motoren M_i aufgebrachten Kräfte wirken positiv in Richtung der jeweils positiven Gelenkpositionen q_i. Die Massen und Dämpfungskonstanten sind mit m_i und d_i gegeben. Die Gravitationsbeschleunigung kann zu $g = 9.81\text{m/s}^2$ angenommen werden. Zur Realisierung sollen verschiedene Funktionen in der Datei **`tasks/task2.py`** in Python implementiert werden. Bitte beachte die Hinweise zur Programmieraufgabe und die Kommentare neben den Funktionen in der Datei **`tasks/task2.py`**.

1. Stelle zunächst die Bewegungsgleichungen für den 3D-Drucker auf. Implementiere eine Funktion

   `force = inverse_dynamics(ddq, dq, link_masses, joint_damping, gravity_acc)`,

   welche aus den übergebenen Verläufen der Gelenkgeschwindigkeiten `dq` und -beschleunigungen `ddq` die jeweiligen Motorkraftverläufe über die Zeit berechnet. In `link_masses`, `joint_damping` und `gravity_acc` werden die Modellparameter übergeben. Die Ausgangsvariable `force` enthält die jeweils in den Gelenken aufgebrachten Motorkräfte.

   Typischerweise beschreibt man die Dynamikgleichung des Roboters mit

   $$\boldsymbol{f} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}),$$

   wobei $\boldsymbol{f}$ die Kräfte, $\boldsymbol{q}$ die verallgemeinerten Gelenkkoordinaten, $\dot{\boldsymbol{q}}$ die Gelenkgeschwindigkeiten, $\ddot{\boldsymbol{q}}$ die Gelenkbeschleunigungen, $\boldsymbol{M}$ die Massenmatrix, $\boldsymbol{C}$ die Zentrifugal- und Corioliskräfte und $\boldsymbol{G}(\boldsymbol{q})$ die Gravitationskraft sind. Hier verwenden wir ein vereinfachtes Modell der Dynamik des Roboters. Die Dynamikgleichung für das erste Gelenk lautet beispielsweise

   $$f_1 = (m_2 + m_3 + m_4)\ddot{q_1} + d_1\dot{q_1} + (m_2 + m_3 + m_4)g.$$

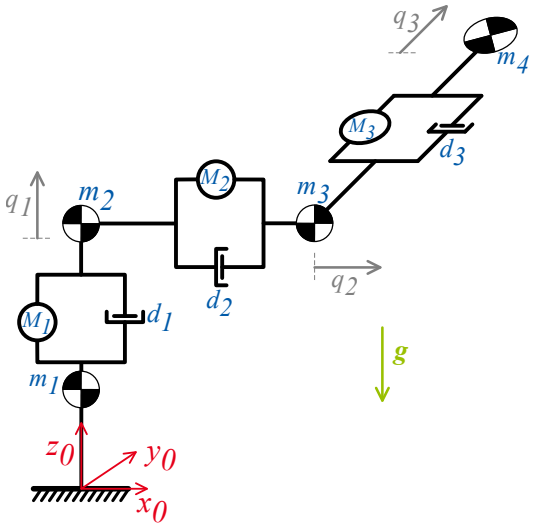

Abbildung 2: Vereinfachtes Modell des 3D-Druckers

Bitte beachte die **Richtungen** der Kräfte!

2. Da tatsächlich oft nur die Gelenkposition, nicht aber ihre Ableitungen, direkt gemessen werden kann, soll nun der Einfluss der Ableitungsmethode sowie der Güte des Messsignals beispielhaft für den Rückwärtsdifferenzenquotienten untersucht werden. Zu diesem Zweck soll der nachstehende Gelenkpositionsverlauf mit der Amplitude a, der Kreisfrequenz ω und der Zeit t betrachtet werden:

$$\boldsymbol{q}(a, \omega, t) = a \cdot \begin{pmatrix} 1 - \cos(\omega t) \\ 1 - \cos(\omega t) \\ 1 - \cos(\omega t) \end{pmatrix}. \tag{2}$$

   Implementiere den beschriebenen Vergleich wie folgt.

   - Vervollständige in einem ersten Schritt die Funktion

     `dq, ddq = cyclic_analytical(sim_time_vect, amplitude, omega)`,

     welche aus den übergebenen Werten für den Simulationszeitverlauf `sim_time_vect`, die Amplitude `amplitude` und die Kreisfrequenz `omega` des Gelenkpositionsverlaufs die erste und zweite analytische Ableitung `dq` und `ddq` berechnet.

   - Schreibe nun die Funktion

     `dq, ddq = cyclic_numerical(q, dt)`,

     welche aus dem übergebenen Gelenkpositionsverlauf `q`, und benötigte Zeitschrittweite `dt`, die erste und zweite numerische Ableitung `dq` und `ddq` mit Hilfe des Rückwärtsdifferenzenquotienten berechnet. Gehe davon aus, dass die Geschwindigkeit und Beschleunigung zum Zeitpunkt $t = 0$ in allen Achsen Null beträgt. Führe die Berechnung des Rückwärtsdifferenzenquotienten in der von dir zu implementierenden Funktion

     `dx = backward_dq(x, dt)`

     durch. Hierbei repräsentiert `x` eine diskret dargestellte Funktion, `dx` ihre Ableitung und `dt` den Zeitschritt.

3. Abschließend soll die Kenntnis über die Inversdynamik des 3D-Druckers eingesetzt werden, um dessen Antriebe auszulegen. Hierfür soll die physikalische Maximalgeschwindigkeit der einzelnen Motoren optimiert werden. **Zur Vereinfachung wird angenommen, dass die Gelenke konstant mit dieser Maximalgeschwindigkeit betrieben werden.** Die Betrachtung soll für den ebenfalls konstanten Sollwert $q_{\text{desired}} = (0.2, 0.2, 0.2)^T$ durchgeführt werden.

- Implementiere als Erstes die Funktion

  `xs = newton(x0, iters, eps, f, df)`,

  welche eine Nullstelle $\boldsymbol{x_s}$ (`xs`) der Funktion `f` ausgehend vom Startvektor $\boldsymbol{x}^{(0)}$ (`x0`) iterativ mit dem Newton-Verfahren in seiner Standard-Basis-Variante ohne Schrittweitensteuerung berechnet. Die Iteration soll abgebrochen werden, wenn $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_2 <$ `eps` oder die maximale Anzahl von Iterationen `iters` erreicht ist. Die genauen Größen und Beschreibungen des Parameters findest du im Code als Kommentar.

  Du musst in dieser Aufgabe die einfache Version des Newton-Verfahren Algorithmus selbst implementieren. Für echte Anwendungen gibt es bereits sehr gute Implementierungen für Python, z.B. `scipy.optimize.newton`.

- Die Maximalgeschwindigkeit ist optimal, wenn die Kostenfunktion

$$\begin{aligned} J &= \sum_{i=1}^{3} J_i(v_{\max}) \\ &= \sum_{i=1}^{3} E_{i_{\text{mech}}}^2 - \beta_i \frac{1}{t} \\ &= \sum_{i=1}^{3} (f_i \cdot q_i)^2 - \beta_i \frac{\dot{q}_i}{q_i} \end{aligned}$$

  minimal wird. Hierbei entspricht E_{mech} der mechanischen Energie, q und $\dot{q}$ der Position sowie Geschwindigkeit und β einem Gewichtungsparameter für jedes Gelenk i. Ersetze die Motorkräfte f durch die zugehörige in Aufgabenteil 1. ermittelte Gleichung.

  Die optimale Geschwindigkeit für die Gelenke i kann durch Lösen der Gleichung $v_{i_{\max}} = \arg\min_{\dot{q}_i} J$ gefunden werden. Implementiere $\frac{\mathrm{d}J}{\mathrm{d}\dot{q}_i}$ (Jacobian Matrix) und die Ableitung $\frac{\mathrm{d}^2 J}{\mathrm{d}\dot{q}_i \mathrm{d}\dot{q}_j}$ (Hessian Matrix), welche zur Optimierung dem Newton-Verfahren zu übergeben sind, in der Funktion

  `vmax_opt = optimize_vmax(q_target, beta, vmax0, iters, eps, link_masses, joint_damping, gravity_acc)`.

  Diese Funktion gibt anhand der bereits beschriebenen Eingangsvariablen die optimalen Maximalgeschwindigkeiten `vmax_opt` in den Gelenken aus.

Hinweise zur Programmieraufgabe

- Für jede Programmierenaufgabe, lade das git *repository*, das im Moodle verlinkt ist, und ziehe die neuesten Updates mit: `git pull`.
- Auf der Startseite der Veranstaltung im Moodle ist eine kurze Einführung in Python und CoppeliaSim verlinkt, welche die wichtigsten Grundlagen zur Verwendung der Programme erläutert.
- Folge der Anleitung zur Installation und Ausführung von Programmen im git *repository*.
- Für die Aufgabe brauchst du nur die Dateien im Ordner *tasks* zu verändern.
- Zur Überprüfung deiner Lösung der einzelnen Teilaufgaben findest du im Ordner `tests` öffentliche Testfälle. Diese können durch den Aufruf des Befehls `make test2` (für task2) im Hauptordner ausgeführt werden. Zur Bewertung werden neben diesen Tests **weitere** Testfälle überprüft!
- Neben den mitgelieferten Testfällen kannst du die Simulation mit `make p2` (für task2) starten. Weitere Informationen findest du im git *repository*.
- Für die Abgabe der Programmieraufgabe lade **nur die zu bearbeitenden Dateien** im Moodle hoch, die sich im Ordner `tasks` befinden.
- Gib als Kommentar am Anfang jeder Datei im Ordner `tasks` die Namen und Matrikelnummern aller deiner Gruppenmitglieder an.

- **Es ist ausreichend, wenn EIN Gruppenmitglied die Lösung einsendet. Alle angegebenen Gruppenmitglieder einer Abgabe bekommen dann dieselbe Bewertung.**

Hinweis zu wissenschaftlichem Arbeiten

Es ist nicht gestattet, Lösungen anderer Personen als die der Gruppenmitglieder als Lösung der Aufgabe abzugeben. Des Weiteren müssen alle zur Lösungsfindung verwendeten, darüber hinausgehenden, relevanten Quellen explizit angegeben werden. Dem widersprechendes Handeln ist Plagiarismus und ist ein ernster Verstoß gegen die Grundlagen des wissenschaftlichen Arbeitens, das ernsthafte Konsequenzen bis hin zur Exmatrikulation haben kann.