Computational Engineering und Robotik

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

Sommersemester 2020

TECHNISCHE UNIVERSITÄT DARMSTADT

6. Übung

Hinweise zu dieser Übung

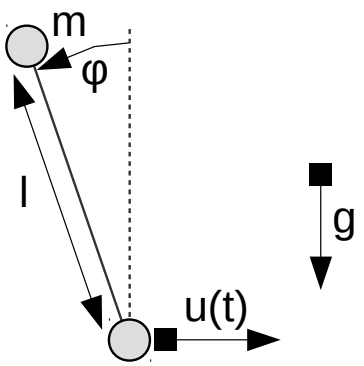
- Für die Teilnahme an der Übung ist eine Anmeldung beim **Moodle** notwendig. Dort sind auf der Kursseite zusätzliche Informationen zur Veranstaltung und die Regelungen zur Anrechnung der Übungsleistungen in die Endnote aufgeführt:

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

- Von dieser Übung werden die **1. Aufgabe** und die **Programmieraufgabe** bewertet. Lade zur Bearbeitung der Programmieraufgabe den aktuellen Code aus dem *git repository* herunter, welches im Moodle verlinkt ist.
- **Abgabe der schriftlichen Lösungen** zu den Übungsaufgaben bis **Montag, den 15.06.2020 um 12:00 Uhr** per Datei-Upload im Moodle.
- **Abgabe der Programmieraufgabe** bis **Montag, den 15.06.2020 um 12:00 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.
- Begründe alle deine Antworten. Falls du einen Scan deiner Antworten hochlädst, stelle sicher, dass sie lesbar sind.
- Stelle Fragen zu dieser Übung bitte im Moodle-Forum oder in den Sprechstunden.

1 PD-Regelung eines inversen Pendels (8 Punkte)

Das inverse Pendel (siehe Abbildung) wird in der Regelungstechnik häufig als Fallbeispiel genutzt. Eine Punktmasse $m = 1\ \mathrm{kg}$ wird an einem Stab der festen Länge $l = 1\ \mathrm{m}$ drehbar um den Winkel φ gelagert. Das Lager ist reibungsbehaftet ($c = 0.01\ \mathrm{Nms/rad}$) und kann über die Motorbeschleunigung u translatorisch bewegt werden, um das Pendel auszubalancieren. Das System unterliegt der Gravitationsbeschleunigung $g = 9.81\ \mathrm{m/s^2}$. Insgesamt lässt sich das Verhalten des inversen Pendels durch folgende Differentialgleichung beschreiben:



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

Das ungeregelte System weist eine instabile Ruhelage bei $\varphi_0 = 0\ \mathrm{rad}$ auf. Es soll nun das Balancieren der Punktmasse in dieser Position senkrecht über dem Lager mittels eines PD-Positionsreglers untersucht werden. Unter der Annahme geringer Auslenkungen ergibt sich die approximierte lineare Bewegungsgleichung:

$$ml^2\ddot{\varphi} = ml \cdot (u + g\varphi) - c\dot{\varphi}.$$

a) Seien $k_p = 14.81$ und $k_v = 1.99$ die Reglerparameter. Ist das geregelte System dann stabil oder instabil? Ist es unterkritisch, kritisch oder überkritisch gedämpft?

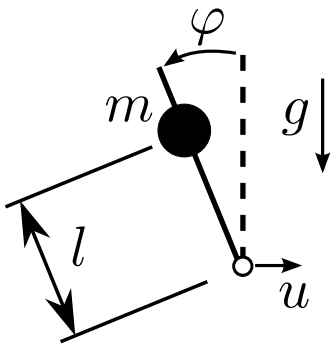
b) Bestimme eine kritische Dämpfung mit $k_v = 1.99$.

c) Bestimme analytisch die Lösung $\varphi(t)$ der linear approximierten Bewegungsgleichung für das Anfangswertproblem $\varphi(0) = 0\,\text{rad}$, $\dot{\varphi}(0) = 1\,\text{rad/s}$ mit dem PD-Regler aus Teilaufgabe a). Falls benötigt, darfst du die Eigenvektoren mit einem numerischen Werkzeug berechnen.

d) Bestimme aus der analytischen Lösung aus c) den Zeitpunkt $t_{\ddot{u},max} > 0$ und den Betrag $\Delta\varphi_{max}$ der maximalen Überschwingweite des Pendels (von negativ bis positiv). Runde das Endergebnis auf 4 Nachkommastellen.

2 Zustandsregelung eines inversen Pendels

Die nebenstehende Abbildung zeigt erneut das inverse Pendel. Die Punktmasse m soll durch eine geeignete Wahl der Motorbeschleunigung u senkrecht stehend balanciert werden. Dabei ist die Länge l des Stabes, auf welchem die Punktmasse balanciert wird, variabel. Zum Balancieren soll ein Zustandsregler eingesetzt werden.
Der Zustandsvektor sei $\boldsymbol{x} = [x_1\ x_2]^T$ mit $x_1 = \varphi$, $x_2 = \dot{\varphi}$, die Stellgröße u, die Systemmatrix $\boldsymbol{A}$ und der Steuervektor $\boldsymbol{b}$. Die um den Arbeitspunkt $\boldsymbol{x_0} = (0, 0)^T$ und $u_0 = 0$ linearisierte Systemgleichung sei allgemein gegeben mit:



$$\boldsymbol{\Delta\dot{x}} = \mathbf{A} \cdot \boldsymbol{\Delta x} + \mathbf{b} \cdot \Delta u.$$

Der Ansatz des Zustandsreglers mit Vorfilter sei mit

$$u = -\boldsymbol{k}^T\boldsymbol{x} + \boldsymbol{v}w$$

gegeben. Dabei bezeichne $\boldsymbol{k}^T$ den Reglervektor mit den Verstärkungsfaktoren, $\boldsymbol{v}$ den Vorfilter und w die Führungsgröße.

a) Stelle zunächst die geregelte Systemgleichung im Zustandsraum in der Form $\boldsymbol{\Delta\dot{x}} = \widetilde{\mathbf{A}}\boldsymbol{x} + \widetilde{\mathbf{b}}w$ auf.

b) Was muss für $\boldsymbol{k}$ gelten, damit das geregelte System stabil ist? Nutze zur Berechnung die Zahlenwerte $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 14.0143 & -0.0571 \end{pmatrix}$ und $\mathbf{b} = \begin{pmatrix} 0 \\ 1.4286 \end{pmatrix}$. Runde auf vier Nachkommastellen genau.

c) Gemessen werden soll nun die Position des Pendels, wobei das Messgerät den entsprechenden Zustand mit dem Faktor 2 verstärkt. Gib die Ausgangsgleichung in der Form $y = \mathbf{c}^T\boldsymbol{x}$ an.

d) Stelle nun die Gleichung für die bleibende Regelabweichung $e_\infty = w - y_\infty$ auf. Hierbei entspricht y_∞ dem stationären Endwert, also der Gleichgewichtslösung, für die Ausgangsgröße y. Diese tritt auf, wenn sich auch die Zustandsgröße $\boldsymbol{x}$ im Gleichgewicht befindet. Wie muss der Vorfilter $\boldsymbol{v}$ gewählt werden, damit keine Regelabweichung verbleibt? Nimm $\boldsymbol{k} = (100, 1)^T$ an.

Programmieraufgabe P3 Simulationsstudie 3D-Drucker (5 Punkte)

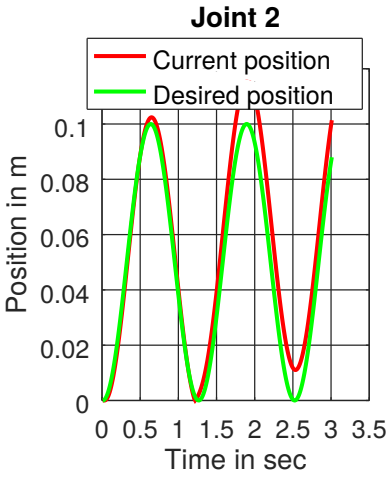


Abbildung 1: Abweichungen zwischen Soll- und Ist-Position bei sinusförmiger Schwingung

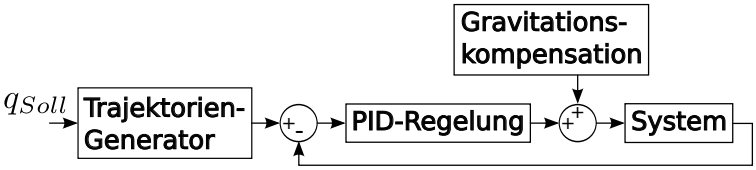


Abbildung 2: Blockschaltbild des im Rahmen dieser Programmieraufgabe zu implementierenden Reglers und Trajektoriengenerators

Wie in der zweiten Programmieraufgabe ersichtlich, gibt es Abweichungen zwischen dem für die Inversdynamik aufgestellten und dem durch CoppeliaSim simulierten Modell (siehe Abb. 1). Dies gilt in den meisten Fällen auch für den Vergleich zwischen theoretischem Modell und realem System. In dieser Programmieraufgabe möchten wir deswegen die bisherigen Kenntnisse über das Modell, ergänzt um eine Regelung, nutzen, um die Abweichung zwischen Soll- und Ist-Verhalten zu verringern. Das zu implementierende Schema ist im Blockschaltbild in Abb. 2 dargestellt.
Zur Realisierung sollen verschiedene Funktionen in der Datei **`tasks/task3.py`** in Python implementiert werden. Bitte beachte die Hinweise zur Programmieraufgabe und die Kommentare neben den Funktionen in der Datei **`tasks/task3.py`**.

Der erste im Rahmen der Simulation ausgegebene Plot visualisiert das Ergebnis des Aufgabenteils a), der zweite das Ergebnis des Aufgabenteils b), der dritte das Ergebnis des Aufgabenteils c).

1. Implementiere zunächst die allgemeine Form eines PID-Reglers ohne Vorsteuerung in der Funktion

   `torques = compute_torques(ctrl_params, q, q_desired, dq, dq_desired)`

   Nutze dafür die jeweils als Spaltenvektoren in der Struktur `ctrl_params` übergebenen Elemente `q`, `dq`, `q_desired` und `q_desired` als Ist-Position, Ist-Geschwindigkeit, Soll-Position und Soll-Geschwindigkeit und lege den berechneten Reglerausgang im Spaltenvektor `u` ab. Berechne dabei das Integral des Fehlers mit Hilfe der Trapezregel.

   Variiere anschließend die für den Aufgabenteil a) angegebenen Verstärkungsfaktoren für die P-, I- und D-Anteile des Reglers in der Funktion

   `kp, ki, kd = setup_coffecient()`

   und verdeutliche dir deren Einfluss auf die Trajektorie, welche der 3D-Drucker tatsächlich zurücklegt.

2. In einem zweiten Schritt soll analog zu Abb. 2 die Regelung nun durch eine Vorsteuerung im Sinne einer modellbasierten Kompensation erweitert und verbessert werden. Da die zweite Ableitung der gemessenen Position in der Realität in der Regel durch starkes Rauschen überlagert ist und um den Aufwand der Identifikation eines geeigneten Reibmodells zu umgehen, soll nur die Gravitation kompensiert werden. Nutze die in der zweiten Programmieraufgabe implementierte Funktion **`inverse_dynamics`** um den Kompensationsterm zu berechnen. Mit dem Befehl **`from tasks.task2 import inverse_dynamics`** kannst du die Funktion importieren.

   Ergänze die Implementierung dieser Teilaufgabe in der für Aufgabenteil a) bereits bearbeiteten Funktion

   `torques = compute_torques(ctrl_params, q, q_desired, dq, dq_desired)`

Ersatzweise kann statt einer eigenen Implementierung der Funktion **`inverse_dynamics`** aus der Vorlage **`tasks/invdyn_template.py`** verwendet werden.

3. Für reale Robotersysteme werden in der Regel glatte Trajektorien anstatt der bislang betrachteten Sprungfunktionen als Solltrajektorien vorgegeben. Dies soll nun beispielhaft für den Fall implementiert werden, dass
   - der Soll-Geschwindigkeitsverlauf trapezförmig angenommen werden kann und
   - der zu erreichende Zielpunkt folglich immer weit genug von der Ausgangsposition entfernt ist, sodass die Maximalgeschwindigkeit aus dem Stand erreicht und von ihr wieder auf Null abgebremst werden kann sowie
   - die zum Erreichen von t_1 und t_2 sowie der Endposition benötigte Zeit ohne Rest durch den Zeitschritt `dt` teilbar ist.

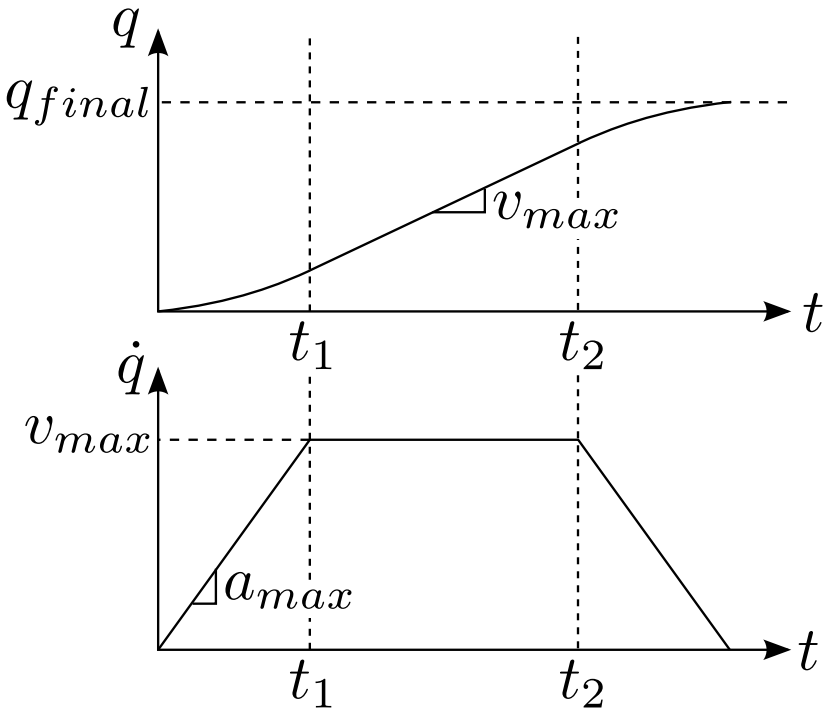


Abbildung 3: Qualitative Darstellung des Trajektorienverlaufs für Position (q) und Geschwindigkeit ($\dot{q}$)

Dies ist qualitativ in Abb. 3 illustriert. Implementiere nun eine Funktion

`q_desired, qd_desired, time = trajectory_generation(q_final, joint_num, vmax, amax, dt)`

welche für den übergebenen Endwert der Position `q_final` die beschriebenen, analytisch berechneten Positions- und Geschwindigkeitstrajektorien `q_desired` und `qd_desired` sowie den zugehörigen Vektor der Zeit `time` $= [0, t_1 + t_2]$ ausgibt. Dabei sollen lediglich die Trajektorien des Gelenks mit der in `joint_num` übergebenen Nummer berechnet werden.

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 test3` (für task3) im Hauptordner ausgeführt werden. Zur Bewertung werden neben diesen Tests **weitere** Testfälle überprüft!
- Neben den mitgelieferten Testfällen kannst du die Simulation mit `make p3` (für task3) 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.