Computational Engineering und Robotik

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

Sommersemester 2019

10. 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 wird nur die **Programmieraufgabe** bewertet. Lade zur Bearbeitung der Programmieraufgabe den aktuellen Code aus dem *git repository* herunter, welches im Moodle verlinkt ist.
- **Abgabe der Programmieraufgabe** bis **Montag, den 13.07.2019 um 12:00 Uhr** per Datei-Upload im Moodle.
- Begründe alle deine Antworten. Falls du einen Scan deiner Antworten hochlädst, stelle sicher, dass sie lesbar sind.
- 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.
- Stelle Fragen zu dieser Übung bitte im Moodle-Forum oder in den Sprechstunden.

Programmieraufgabe P5 Motormodell für den 3D-Drucker (5 Punkte)

Die für den Bau des 3D-Druckers angeschafften Motoren waren ein echtes Schnäppchen, aber leider sind auch die darin auftretenden Reibungseffekte nicht zu vernachlässigen. Um diese Effekte auch in der 3D-Drucker-Simulation abzubilden, wurde dem Simulations-Framework für diese Programmieraufgabe ein Motormodell hinzugefügt.
Analog zu den letzten Programmierübungen wird vereinfachend angenommen, dass ein Motor statt einer Spannung eine Soll-Kraft u_{soll} als Steuereingang erhält. Aufgrund der Reibung bringt er aber effektiv nur eine Ist-Kraft $u_{\text{ist}} \leq u_{\text{soll}}$ auf.

Um dem entgegenzuwirken, soll in den folgenden Teilaufgaben eine Kompensation der Reibungseffekte implementiert werden. Dazu wird eine Modellfunktion benötigt, die das Verhalten der in der Simulation verwendeten Motoren approximiert. Die Parameter dieser Modellfunktion sollen zunächst anhand von Messdaten identifiziert werden. Auf Basis des identifizierten Motormodells kann dann die Kompensation der Reibungseffekte erfolgen.

1. Implementiere zuerst ein Verfahren zur Minimierung der kleinsten Quadrate der Abweichungen einer Modellfunktion von gegebenen Messdaten. Vervollständige dazu die Vorlage der Funktion

   ```
   L, dL_dp, p, err_final, success = nonlinear_least_squares(motor_func, pStart, grid, data, approx_model)
   ```

- Eingabewerte von **`nonlinear_least_squares`** sind eine Modellfunktion des Motors **`motor_func`**, eine erste Schätzung der Parameter **`pStart`**, ein Array von Auswertungspunkten **`grid`**, die Messdaten **`data`** der Motorkraft an den Gitterpunkten in **`grid`** sowie eine Beschreibung des Systemmodells **`approx_model`**, welche zur Approximation der Messwerte verwendet werden soll.
- Rückgabewerte von **`nonlinear_least_squares`** sind ein Funktions-Handle der Fehlerschätzung **`L`**, ein Funktions-Handle der ersten Ableitung **`dL_dp`**, die optimierten Modellparameter **`p`**, der Wert des Gütekriteriums der Optimierung **`err_final`** sowie der Status des nichtlinearen Lösers beim Abbruch, **`success`**.
- Als Modellfunktion des Motors **`motor_func`** dient die Methode

  ```
  u, dp = motor_model_function(grid, approx_model, param)
  ```

  Diese bekommt als Eingabe die Auswertungspunkte **`grid`**, die Modellspezifikation **`approx_model`** sowie eine Liste von Parametern **`param`**. Die Rückgabewerte der Funktion sind die Auswertung **`u`** der Modellfunktion an den gegebenen Punkten sowie die Ableitung **`dp`** der Modellfunktion an diesen Punkten nach den Parametern.
- Eine Zeile der Matrix **`grid`** hat die Form **`[uDesired, q, dq]`**, wobei **`uDesired`** die Soll-Kraft u_{soll}, **`q`** die Motorposition und **`dq`** die Motorgeschwindigkeit bezeichnet. Die Anzahl der Zeilen entspricht der Anzahl an Auswertungspunkten und damit gleichzeitig der Länge des Vektors **`data`**.
- Definiere ein Funktions-Handle **`L(p)`** innerhalb der Funktion **`nonlinear_least_squares`**, welches eine Fehlerschätzung zwischen Motormodell und Messdaten gemäß dem Kleinste-Quadrate-Kriterium

  $$L = \frac{1}{2}\sum_{i=1}^{N}(u_i - \hat{u}_i)^T(u_i - \hat{u}_i)$$

  für einen Parameter-Vektor **`p`** berechnet. Hierbei ist u_i der approximierte Wert von **`motor_func`** und $\hat{u}_i$ sind die Messdaten vom i-ten Punkt.
- Definiere außerdem ein Funktions-Handle **`dL_dp(p)`**, welches die erste Ableitung des Fehlerschätzers **nach allen Parametern** für einen Parameter-Vektor **`p`** berechnet.
- Der Eingabewert **`model_func`** soll unverändert an **`motor_model_function`** weitergegeben werden.
- Die Optimierung kann mittels **`scipy`** erfolgen. Nutz dafür die Optimierungsmethode SLSQP mit der Toleranz 0.0001 und maximalen Iterationen 2000. Weitere Informationen findest du auf dieser Website

  https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

2. Nun soll eine Modellfunktion M des Motors implementiert werden. Eine Auswertung der Motorkräfte an den verschiedenen Datenpunkten legt nahe, dass der Zusammenhang zwischen Soll-Kraft und Ist-Kraft linear ist. Die Steigung dieser linearen Funktion hängt jedoch von der aktuellen Motorposition und Motorgeschwindigkeit ab. Wir nehmen vereinfachend an, dass die Reibungseinflüsse von Position und Geschwindigkeit unabhängig voneinander sind und modellieren daher

   $$u_{\text{ist}} \approx M(u_{\text{soll}}, q, \dot{q}, \boldsymbol{p_1}, \boldsymbol{p_2}) := \mu_1(q, \boldsymbol{p_1}) \cdot u_{\text{soll}} + \mu_2(\dot{q}, \boldsymbol{p_2}) \cdot u_{\text{soll}} \tag{1}$$

   mit Motorposition q, Motorgeschwindigkeit $\dot{q}$ und den Hilfsfunktionen μ_1 und μ_2, die von den Parametern in $\boldsymbol{p_1}$ bzw. $\boldsymbol{p_2}$ abhängen.

   Implementiere die Funktion

   ```
   u, dudp = motor_model_function(grid, approx_model, param)
   ```

   - Das dictionary **`approx_model`** liefert das Funktionshandle **`u1`** und **`u2`**, die Namen der aufzurufenden Hilfsfunktionen, also der Typ der Funktionen μ_1 und μ_2. Dies ist in beiden Fällen „**`polynomN`**“. Der Parameter **`approx_model`** liefert auch die Anzahl an von der Polynomfunktion benötigten Parameter **`n1, n2`**. Diese Angabe legt die Ordnung des Polynoms eindeutig fest. Die Parameter sind in der Vorlage entpackt.

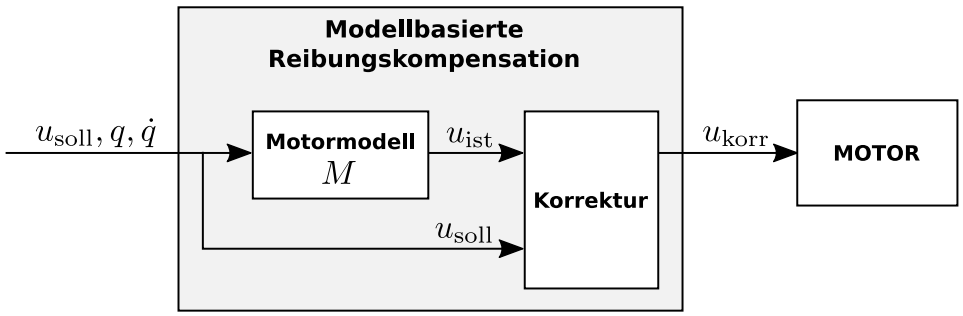

Abbildung 1: Schematische Darstellung der modellbasierten Reibungskompensation.

3. Implementiere die Hilfsfunktion **`fx, dfdp = polynomN(x, param, n)`**, welche ein Polynom der Form

$$f(x) = p_1 + p_2 x + p_3 x^2 + \cdots + p_{n+1} x^n$$

auswertet.

- Der Eingangsvektor **`x`** enthalte alle Stellen x, an denen das Polynom ausgewertet werden soll, der Vektor **`param`** enthalte die Koeffizienten $p_1, p_2, \ldots, p_{n+1}$ und **`n`** sei der Grad n des Polynoms. Der Ausgangsvektor **`fx`** enthalte die Auswertungen des Polynoms an allen Stellen x und die Matrix **`dfdp`** enthalte zeilenweise für alle Stellen x die Ableitung des Polynoms nach den Parametern $p_1, p_2, \ldots, p_{n+1}$.
- *Hinweis:* Verwende **`cumprod`** von **`numpy`**, um den Feature-Vektor $(1\ x\ x^2\ \cdots x^n)^T$ zu generieren.
- Verwende die Methode **`make plot53 n1=[Länge des Parameters p1] n2=[Länge des Parameters p2]`**, um die Parameterschätzung mit der in 1) implementierten **`nonlinear_least_squares`** Funktion auszuführen und das „reale“ Motormodell zusammen mit der in **`motor_model_function`** berechneten Approximation zu plotten. Teste verschiedene Ordnungen der Polynome und untersuche jeweils die Auswirkung auf den Fehler.

4. Schreibe nun eine Kompensation der Reibungseffekte im Motor, die auf der in b) fertiggestellten Modellfunktion basiert.

- Verwende die Vorlage

  `u = modelbased_friction_compensation(approx_model, param, u, q, dq)`

  Eingaben dieser Funktion sind eine Modellspezifikation **`approx_model`**, eine List von Parametern **`param`**, eine Sollkraft **`u`**, die aktuelle Motorposition **`q`** und die aktuelle Motorgeschwindigkeit **`dq`**. Zurückgegeben wird eine korrigierte Motorkraft.
- Durch die im Motor auftretenden Reibungskräfte wirkt statt der Sollkraft u_{soll} nur eine Motorkraft u_{ist}, also $u_{\text{ist}} = \textbf{MOTOR}(u_{\text{soll}})$. Die Funktion **`modelbased_friction_compensation`** soll aus den gegebenen Eingaben eine Korrekturkraft u_{korr} berechnen, so dass

$$u_{\text{soll}} \approx \textbf{MOTOR}\,(u_{\text{korr}})$$

  gilt. Dabei stehe **MOTOR** für das Verhalten des simulierten Motors, welches durch die in 2) identifizierte Modellfunktion M aus (1) approximiert werden kann. Als Basis für die Berechnung von u_{korr} dienen also die Zusammenhänge

$$u_{\text{ist}} \approx M(u_{\text{soll}}) \quad \text{und} \quad u_{\text{soll}} \approx M(u_{\text{korr}}).$$

  Die Funktionsweise der Reibungskompensation ist in Abbildung 1 schematisch dargestellt.
- Verwende die Funktion **`motor_model_function`** als Implementierung der Modellgleichung M.

5. Zur Evaluation der implementierten Reibungskompensation werden drei Simulationen durchgeführt, in denen der 3D-Drucker jeweils dieselbe Figur druckt:

   a) Simulation wie in den vorangegangenen Programmieraufgaben ohne Motor-Modellierung in **`make simu54a`**,

b) Simulation mit Berücksichtigung der Motoren (inklusive auftretender Reibungseffekte) in **`make simu54b`**,

c) Simulation mit Berücksichtigung der Motoren und der hier implementierten modellbasierten Reibungskompensation **`make simu54c`**.

Die 1. Simulation dient als Referenzlösung um die beiden Varianten mit Motor-Berücksichtigung daran messen und vergleichen zu können. In allen drei Fällen wird der Zustandsverlauf des Druckermodells im Ordner **`run/`** gespeichert. Dabei enthält **`q51.csv`** die Aufzeichnung **`Q1`** der ersten, **`q52.csv`** die Aufzeichnung **`Q2`** der zweiten und **`q53.csv`** die Aufzeichnung **`Q3`** der dritten Simulation.
Zum Vergleich der Abweichungen $\delta_1 := ||\mathbf{Q1} - \mathbf{Q2}||$ und $\delta_2 := ||\mathbf{Q1} - \mathbf{Q3}||$ soll der durchschnittliche Fehler zusammen mit dem Mittelwert und der Standardabweichung geplottet werden.

- Simulationsdaten werden in temporären Dateien gespeichert, um die Daten nach erfolgreicher Simulation zu zeichnen. Verwende den Befehl **`make plot55`**
- Vervollständige die Funktion **`plot_model_differences(time_scale, Q1, Q2, Q3)`**, welche als Eingabe die oben beschriebenen Aufzeichnungen **`Q1, Q2, Q3`** und die Zeitskala **`time_scale`** erhält.
- Implementiere darin die Berechnung der Abweichung zwischen **`Q1`** und **`Q2`** und plotte den Fehlerverlauf in einen ersten Subplot (von zwei Subplots).
- Berechne den Mittelwert und die Standardabweichung für diese Fehlerwerte und füge diese als gestrichelte Linien in roter Farbe zu obigem Plot hinzu. Die Standardabweichung soll sowohl oberhalb als auch unterhalb des Mittelwerts aufgetragen werden.
- Wiederhole dieses Verfahren für die Abweichung zwischen **`Q1`** und **`Q3`** und plotte Fehlerverlauf, Mittelwert und Standardabweichung in einen zweiten Subplot.
- Denke daran, eine Achsenbeschriftung und Legende hinzuzufügen.

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 test4` (für task4) im Hauptordner ausgeführt werden. Zur Bewertung werden neben diesen Tests **weitere** Testfälle überprüft!
- Neben den mitgelieferten Testfällen kannst du die Simulation mit `make p5` (für task5) 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.