Computational Engineering und Robotik

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2022, Homework 2

Prof. J. Peters, K. Ploeger, K.Hansel, D. Palenicek, F. Al-Hafez und T. Schneider

Total points: 15

Abgabefrist: 11:59, Montag, 13 Juni 2022

2.1 Physikalische Simulation eines Roboterarms im Freilauf [15 Points]

In der letzten Übung haben wir uns angeschaut, wie wir die Gelenkwinkel unseres Roboters berechnen müssen, um zu beliebigen Zielpositionen fahren zu können. Allerdings haben wir dabei die Physik völlig außer Acht gelassen und einfach angenommen, dass der Roboter in der Lage ist, mit der exakten gewünschten Geschwindigkeit zu den berechneten Gelenkwinkeln zu fahren. In der realen Welt können wir die physikalischen Gesetze jedoch nicht einfach vernachlässigen, sondern müssen diese sorgfältig in das Design unseres Roboters und dessen Reglers einfließen lassen. Folglich wollen wir uns in dieser Übung damit beschäftigen, das dynamische Verhalten unseres Roboters mittels physikalischer Simulation zu untersuchen.

Dazu werden wir verschiedene Integrationsverfahren implementieren und sehen, dass die Wahl dieser einen signifikanten Einfluss auf die Qualität unserer Simulation haben kann. Wir werden zunächst davon ausgehen, dass der Roboter nicht angetrieben ist und es in dem System auch keine Reibung gibt, da dies die Gleichungen etwas vereinfacht. Da die Simulationsmethode sich dadurch nicht ändert, ist es jedoch nicht aufwendig, Reibung und Antrieb später hinzuzufügen. Dies wird z.B. in der nächsten Übung geschehen, in der wir uns ansehen, wie wir die Motoren regeln müssen, um ein gewünschtes Verhalten zu erzielen.

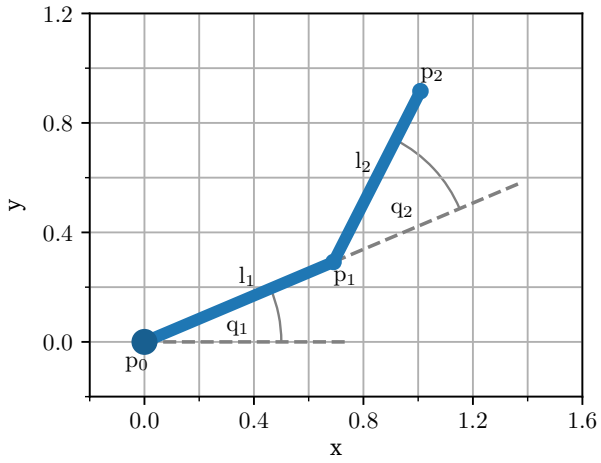

Figure 1: Visualisierung des Roboterarms. p_0 ist die Basis des Arms, p_1 das Ellenbogengelenk und p_2 der Endeffektor. Die Massen des Ellenbogengelenks p_1 und des Endeffektors p_2 sind mit m_1, bzw. m_2 gegeben.

Aufsetzen der Programmierumgebung
Falls du das nicht bereits schon gemacht hast, folge den Anweisungen auf Übungsblatt 1, um eine Programmierumgebung aufzusetzen und zu aktivieren. Installiere anschließend die benötigten Pakete mittels

```
pip install /path/to/cer_pex2_lib
```

und starte wie gewohnt deinen *Jupyter*-Server. Es gelten die selben Hinweise wie auf Übungsblatt 1.

1. **Implementierung der Bewegungsgleichung des Roboters [2 Points]**

   Wie in der letzten Übung schauen wir uns einen zweigelenkigen Roboterarm, wie in Abb. 1 dargestellt, an. Der einzige Unterschied ist, dass wir diesmal zusätzlich Massen m_1 und m_2 mit p_1 und p_2 assoziieren.
   Die Bewegungsgleichungen[1] dieses Roboters sind gegeben durch

$$\ddot{q}_1 = \frac{\frac{1}{2}\dot{q}_1^2 l_1 m_2 \sin(2q_2) + m_2 (\dot{q}_1 + \dot{q}_2)^2 l_2 \sin(q_2) - g\left((m_1 + m_2)\cos(q_1) - m_2 \cos(q_1 + q_2)\cos(q_2)\right)}{l_1 \left(m_1 + m_2 \sin(q_2)^2\right)} \tag{1}$$

$$\ddot{q}_2 = -\ddot{q}_1 \left(1 + \frac{l_1}{l_2}\cos(q_2)\right) - \dot{q}_1^2 \frac{l_1}{l_2}\sin(q_2) - \frac{g}{l_2}\cos(q_1 + q_2) \tag{2}$$

   Wandle diese Gleichungen zunächst in eine DGL erster Ordnung um und implementiere anschließend

```
dx = forward_dynamics(x, l, m, g)
```

   die, gegeben des Roboterzustands $\boldsymbol{x}$, dessen zeitliche Ableitung $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ zurückgibt. Um Kompatibilität mit unserem Testframework zu wahren, strukturiere den Roboterzustand bitte wie folgt: $\boldsymbol{x} = (q_1, q_2, \dot{q}_1, \dot{q}_2)^T$.

2. **Implementierung des expliziten Euler-Verfahrens [2 Points]**

   Implementiere das explizite Euler-Verfahren

```
z = expl_euler(f, x0, dt, n)
```

   welches die zu integrierende Funktion $f\colon \mathbb{R}^m \to \mathbb{R}^m$, den Startwert $x_0 \in \mathbb{R}^m$, den Zeitschritt $\delta t \in \mathbb{N}$ und die Anzahl durchzuführender Schritte $n \in \mathbb{N}$ als Argumente erhält. Zurückgegeben wird eine Matrix $z \in \mathbb{R}^{n \times m}$, bei der jede Zeile ein Zwischenschritt des Verfahrens ist. Die erste Zeile von z ist immer der Startwert x_0.
   Um dieses und die folgenden Integrationsverfahren zu testen, stellen wir dir 6 nominale Trajektorien zur Verfügung, mit denen wir die Ausgabe deines Verfahrens dann vergleichen. Die verschiedenen Trajektorien unterscheiden sich in ihren Startwerten, aber auch in den Parametern des Roboters, auf dem sie erstellt wurden. Falls du dir die Trajektorien ansehen willst, findest du sie unter `cer_pex2_lib/cer_pex2/trajectories.py`.
   Der Vergleich deines Verfahrens mit der nominalen Trajektorie erfolgt punktweise, d.h. für jeden Zeitschritt t berechnen wir

$$e_t = \|z_{t,0:2} - z^*_{t,0:2}\|$$

   wobei z_t die Ausgabe deiner Implementierung ist und z^*_t der Nominalwert für diesen Zeitpunkt ist.
   Anschließend berechnen wir den Durchschnitt dieser Fehler für die gesamte Trajektorie und prüfen ob dieser unter einem verfahrenspezifischen Grenzwert liegt.
   Für einen visuellen Vergleich rendern wir in jedem Test sowohl deine Trajektorie als auch die nominale Trajektorie (als dunkler Schatten im Hintergrund). Somit kannst du sehen, wie nah deine Implementierung an der nominalen Trajektorie dran ist.

[1] Wenn du interessiert bist, kannst die Herleitung dieser Gleichungen auf `https://de.wikipedia.org/wiki/Doppelpendel` nachvollziehen. Beachte jedoch das hier eine leicht andere Definition der Gelenkwinkel verwendet wird.

3. **Implementierung des Vorwärtsdifferenzenquotienten [2 Points]**

   Implementiere den Vorwärtsdifferenzenquotienten in der Funktion

   ```
   df_x = forward_dq(f, x, d)
   ```

   welche als Eingabe eine Funktion $f\colon \mathbb{R}^{m_1} \to \mathbb{R}^{m_2}$, einen Punkt $x \in \mathbb{R}^{m_1}$ und eine Schrittweite d bekommt. Zurück gibt die Funktion eine Schätzung der Jacobi-Matrix $J_f(\boldsymbol{x})$ von f and der Stelle $\boldsymbol{x}$, die mittels des Vorwärtsdifferenzenquotienten berechnet wurde.

4. **Implementierung des impliziten Euler-Verfahrens [3 Points]**

   Implementiere das implizite Euler-Verfahren in der Funktion

   ```
   z = impl_euler(f, x0, dt, n)
   ```

   Da diese Methode das Lösen nichtlinearer Gleichungssysteme erfordert, verwende bitte das Newton-Verfahren, das von uns im *Jupyter*-Notebook vorgegeben wurde. Die Jacobi-Matrix kann durch die Finite-Differenzen-Methode in der Funktion `foward_dq` berechnet werden.

5. **Implementierung des Heun-Verfahrens [3 Points]**

   Implementiere das Heun-Verfahren in der Funktion

   ```
   z = heun(f, x0, dt, n)
   ```

6. **Implementierung des Runge-Kutta-Verfahrens vierter Ordnung [3 Points]**

   Implementiere das Runge-Kutta-Verfahren vierter Ordnung (RK4) in der Funktion

   ```
   z = rk4(f, x0, dt, n)
   ```

Hinweis zu wissenschaftlichem Arbeiten

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Mit der Abgabe einer Lösung für eine schriftliche Aufgabe oder eine Programmieraufgabe bestätigen Sie, dass Sie/Ihre Gruppe die alleinigen Autoren des gesamten Materials sind. Falls die Verwendung von Fremdmaterial gestattet ist, so müssen Quellen korrekt zitiert werden.

Weiterführende Informationen finden Sie auf der Internetseite des Fachbereichs Informatik:

`https://www.informatik.tu-darmstadt.de/studium_fb20/im_studium/studienbuero/plagiarismus/index.de.jsp`

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.