ROME1 - Praktikum 3
Modellierung von Roboter in Python

Das Ziel dieser Übung ist es, die Mathematik für die Robotik kennenzulernen. Dazu werden Orientierungen von Koordinatensystemen mit verschiedenen Methoden berechnet, Roboter mit serieller Kinematik modelliert, die Vorwärts- und Rückwärtskinematik berechnet sowie verschiedene Arten der Bewegungsplanung simuliert und untersucht.

1 Installation der Programmierumgebung

Diese Übung wird mit der Programmiersprache Python durchgeführt. Dazu muss auf dem PC eine Entwicklungsumgebung für Python installiert werden, wie beispielsweise diese hier:

```
https://www.python.org/downloads/
```

Weiter wird eine Toolbox mit Funktionen und Klassen für die Modellierung von Robotern benötigt. Eine populäre Toolbox für Python ist die Robotics Toolbox von Peter Corke. Diese Toolbox wird in einem Terminal des Betriebsystems wie folgt installiert:

```
% pip3 install roboticstoolbox-python
```

Mit dieser Toolbox wird auch die Spatial Math Toolbox for Python installiert, welche Basisfunktionen für die räumliche Geometrie zur Verfügung stellt.

2 Orientierungen und Transformationen

Diese Aufgabe ist eine Einführung in die Robotics Toolbox for Python und zeigt, wie Orientierungen und Transformationen beschrieben und berechnet werden können. Dazu muss zuerst die Spatial Math Toolbox in der Entwicklungsumgebung importiert werden:

```
>>> import spatialmath as sm
```

2.1 Beschreibung von Orientierungen

Orientierungen können mit Rotationsmatrizen, Eulerwinkel, Drehvektoren oder Quaternionen definiert werden. Die Robotics Toolbox erlaubt, die einzelnen Definitionen von Rotationen ineinander umzurechnen.
Mit der folgenden Funktion können Eulerwinkel, die in der Z-Y-Z Notation gegeben sind, in eine Rotationsmatrix umgerechnet werden:

```
>>> r = sm.base.eul2r(0.2, 0.3, 0.5)
>>> print(r)
```

Mit der `print()` Funktion wird diese Rotationsmatrix als 3x3 Matrix in der Konsole angezeigt.

Diese Rotationsmatrix kann dann in Quaternionen umgerechnet werden, und zwar wie folgt:

```
>>> q = sm.UnitQuaternion(r)
>>> print(q)
```

Damit wird ein Quaternion Objekt kreiert, und dem Konstruktor die Rotationsmatrix übergeben. Dieses Objekt bietet Funktionen an, mit denen die Orientierung wieder in andere Darstellungen umgerechnet werden kann, zum Beispiel in Eulerwinkel:

```
>>> e = q.eul()
>>> print(e)
```

Sind dies wieder dieselben Eulerwinkel, mit denen vorhin zuerst die Rotationsmatrix berechnet wurde?
Wiederholen Sie diese Berechnungen noch einmal, aber mit den folgenden Eulerwinkel:

```
>>> r = sm.base.eul2r(0.2, -0.3, 0.5)
>>> print(r)
>>> q = sm.UnitQuaternion(r)
>>> e = q.eul()
>>> print(e)
```

Welche Eulerwinkel erhalten Sie jetzt zurück? Sind diese korrekt?
Berechnen Sie aus diesen Eulerwinkel noch einmal eine Rotationsmatrix, und vergleichen Sie diese mit der Rotationsmatrix `r` oben.

```
>>> r = sm.base.eul2r(e)
```

Ist dies dieselbe Matrix?

2.2 Homogene Transformationen

Homogene Transformationen werden in der Robotics Toolbox mit der Klasse `SE3` dargestellt, welche eine 4x4 Matrix beinhalten. Diese Klasse bietet verschiedene Möglichkeiten, homogene Transformationen zu erstellen und damit zu rechnen.

Definieren Sie nun zuerst eine homogene Transformationsmatrix, die nur aus einer Verschiebung besteht. Das geht mit dem folgenden Konstruktor:

```
>>> T0 = sm.SE3(0.7, 0.0, 0.2)
```

Definieren Sie nun eine Transformationsmatrix mit einer Rotationsmatrix, die durch die Z-Y-Z Eulerwinkel mit den Werten 0.2, 0.3 und 0.5 [rad] gegeben ist:

```
>>> T1 = sm.SE3.Rz(0.2)*sm.SE3.Ry(0.3)*sm.SE3.Rz(0.5)
```

Kombinieren Sie diese beiden Transformationen durch eine Matrixmultiplikation:

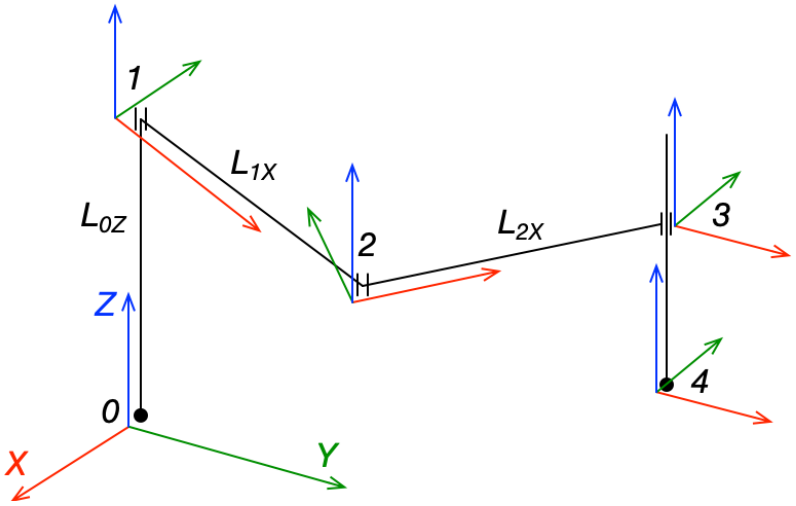
```
>>> T = T0*T1
>>> T
```

Erkennen Sie in dieser Transformation die Rotationsmatrix aus der vorherigen Aufgabe 2.1, sowie den Verschiebungsvektor?

3 Simulation eines SCARA Roboters

3.1 Modellierung des SCARA Roboters

Die Robotics Toolbox for Python erlaubt, serielle kinematische Ketten mit Körpern und Gelenken zu modellieren. Und damit lassen sich typische Industrieroboter wie SCARA Roboter oder Knickarm-roboter modellieren.
Es soll nun ein kinematisches Modell des folgenden SCARA Roboters erstellt werden (siehe Figur 3.1).



Figur 3.1: Kinematisches Modell eines SCARA Roboters

Dieser Roboter besteht aus 3 vertikal ausgerichteten Drehgelenken und aus einem vertikalen Schubgelenk. Das erste Gelenk 1 hat einen Offset von L_{0Z} in vertikaler Richtung bezüglich dem Basissystem *0*. Die folgenden Gelenke *2* und *3* haben einen Offset von L_{1X} und L_{2X} in jeweils lokaler x-Richtung.
Damit kann die Kinematik dieses Roboters mit den folgenden Denavit-Hartenberg Parametern beschrieben werden.

Gelenk	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	q_1	L_{0Z}
2	0	L_{1X}	q_2	0
3	0	L_{2X}	q_3	0
4	0	0	0	q_4

Tabelle 3.1: Denavit-Hartenberg Parameter des SCARA Roboters

Die numerischen Werte der Kinematikparameter sind:

L_{0Z} = 1.0 m
L_{1X} = 0.8 m
L_{2X} = 0.8 m

Wenn die Kinematik eines Roboters mit Denavit-Hartenberg Parametern gegeben ist, kann ein Modell mit der Klasse `DHRobot` erstellt werden. Um diese und weitere Klassen der Robotics Toolbox verwenden zu können, muss diese zuerst importiert werden.

```
>>> import roboticstoolbox as rtb
```

Dann kann mit der folgenden Funktion ein Modell eines SCARA Roboters erstellt werden. Es wird ein Robotermodell aus vier Transformationen kreiert, deren Denavit-Hartenberg Parameter angegeben werden.

```
>>> scara = rtb.robot.DHRobot(
        [
            rtb.robot.RevoluteMDH(d=1.0),
            rtb.robot.RevoluteMDH(a=0.8),
            rtb.robot.RevoluteMDH(a=0.8),
            rtb.robot.PrismaticMDH(qlim=[-1.0, 0.0]),
        ], name="SCARA")
```

Beim letzten Gelenk, dem vertikalen Schubgelenk, muss noch eine untere und obere Limite der Gelenkwinkel angegeben werden, damit später die Kinematik korrekt gerechnet werden kann.

Das `scara` Objekt kann dann in der Python Konsole wieder als DH-Tabelle dargestellt werden:

```
>>> scara
```

Damit lässt sich überprüfen, ob das Modell des Roboters korrekt parametrisiert wurde. Der Roboter kann auch für gegebene Gelenkwinkel grafisch dargestellt werden:

```
>>> scara.plot([1.2, -1.2, -0.4, -0.3])
>>> scara.plot([0.2, 1.2, 0.2, -0.5])
```

3.2 Berechnung der Vorwärts- und Rückwärtskinematik

Mit dem vorhin erstellten Modell des SCARA Roboters kann man nun die Vorwärtskinematik für gegebene Gelenkwinkel oder die Rückwärtskinematik für eine gegebene Lage des Werkzeuges im Raum berechnen.
Die Vorwärtskinematik ist durch die Transformation vom Basissystem zum Werkzeugsystem gegeben. Diese kann wie folgt für gegebene Gelenkwinkel berechnet werden:

```
>>> T = scara.fkine([0.2, 1.2, 0.2, -0.5])
>>> T
```

Damit erhält man eine homogene Transformationsmatrix `T`, welche die Lage des Tool Center Points (TCP) im Basissystems des Roboters beschreibt.
Um die Rückwärtskinematik zu berechnen, muss die Lage des Tool Center Points im Basissystem vorgegeben werden. Damit werden die Gelenkwinkel des Roboters berechnet, die zu dieser Lage des TCP führen.

```
>>> pos = scara.ikine_LM(T, q0=[0,1,0,0], mask=[1,1,1,0,0,1])
```

Die Rückwärtskinematik wird mit einem numerischen, iterativen Algorithmus mit Hilfe der Vorwärtskinematik berechnet. Dazu kann noch ein Array q_0 mit ersten Schätzwerten der Gelenkwinkel angegeben werden. Und mit einem weiteren Argument wird eine Maske definiert, die angibt, welche der 6 Kartesischen Grössen für die Berechnung der Rückwärtskinematik relevant sind. Bei unserem SCARA Roboter sind dies die 3 Kartesischen Positionen, sowie die Orientierung um die Z-Achse.
Das Ergebnis dieser Berechnung ist ein Objekt, das mehrere Metainformationen zur Berechnung der Rückwärtskinematik enthält, wie zum Beispiel die Anzahl numerischer Iterationen. Die resultierenden Gelenkwinkel erhält man wie folgt:

```
>>> pos.q
```

Wie Sie sehen, sind dies wieder die Gelenkwinkel, mit denen vorhin die Vorwärtskinematik berechnet wurde.
Berechnen Sie noch einmal die Rückwärtskinematik, aber nun mit den folgenden Schätzwerten q_0 der Gelenkwinkel:

```
>>> pos = scara.ikine_LM(T, q0=[0,-1,0,0], mask=[1,1,1,0,0,1])
>>> pos.q
```

Erhalten Sie dieselbe Lösung wie vorhin? Wenn nicht, wieso nicht?
Ist diese Lösung korrekt? Um dies zu prüfen können Sie mit dieser Lösung wieder die Vorwärtskinematik berechnen:

```
>>> T = scara.fkine(pos.q)
>>> T
```

Erhalten Sie damit wieder dieselbe Transformation, also dieselbe Kartesische Lage des Werkzeuges wie mit den ursprünglichen Gelenkwinkel, für die Sie die Vorwärtskinematik berechnet haben?

Stellen Sie den Roboter für beide Lösungen der Gelenkwinkel grafisch dar, und vergleichen Sie diese beiden Konfigurationen.

```
>>> scara.plot([0.2, 1.2, 0.2, -0.5])
>>> scara.plot(pos.q)
```

Dies sind die beiden möglichen Lösungen, die es bei der Berechnung der Rückwärtskinematik für SCARA Roboter geben kann. Mit der Wahl der Schätzwerte q_0 kann beeinflusst werden, welche der Lösungen gefunden wird.

3.3 Punkt-zu-Punkt Bewegung des Roboters in Gelenkkoordinaten

Die Robotics Toolbox for Python bietet auch die Möglichkeit, Bewegungen des Roboters zu simulieren. Definieren Sie dazu 2 homogene Transformationsmatrizen, welche die Anfangs- und die Endlage des Werkzeuges beschreiben.

```
>>> T1 = sm.SE3([[0, -1, 0, 0.8],
                 [1,  0, 0, 0.2],
                 [0,  0, 1, 0.7],
                 [0,  0, 0, 1]])

>>> T2 = sm.SE3([[0, -1, 0, -0.6],
                 [1,  0, 0, 0.6],
                 [0,  0, 1, 0.3],
                 [0,  0, 0, 1]])
```

Für diese beiden Transformationsmatrizen muss nun die Rückwärtskinematik berechnet werden. Damit erhält man die Gelenkwinkel für die Anfangs- und die Endlage des Werkzeuges.

```
>>> pos1 = scara.ikine_LM(T1, q0=[0,1,0,0], mask=[1,1,1,0,0,1])
>>> pos2 = scara.ikine_LM(T2, q0=[0,1,0,0], mask=[1,1,1,0,0,1])
```

Zwischen diesen beiden Lagen müssen nun Wegpunkte mit einer Trajektorie berechnet werden. Dazu wird zuerst ein Array mit Zeitpunkten für die Bewegung generiert:

```
>>> import numpy as np
>>> t = np.arange(0, 2, 0.02)
```

Damit erhalten wir ein Array mit 100 Zeitpunkten für die Bewegung.
Nun kann eine Trajektorie für alle Gelenkwinkel berechnet werden:

```
>>> trajectory = rtb.tools.mtraj(rtb.tools.trapezoidal,
                                 pos1.q, pos2.q, t)
```

Mit dem Parameter `trapezoidal` wird die Form der Trajektorie festgelegt. Hier wird eine Trajektorie mit einem trapezförmigen Geschwindigkeitsverlauf gewählt.
Diese Trajektorie kann anschliessend mit einem Plot des Roboters wie folgt visualisiert werden:

```
>>> scara.plot(trajectory.q)
```

Die Trajektorie kann aber auch in einem 2D Plot in Funktion der Zeit dargestellt werden:

```
>>> rtb.xplot(t, trajectory.q)
```

Damit werden die 4 Gelenkwinkel in Funktion der Zeit geplottet. Es ist aber auch möglich, die Gelenkgeschwindigkeiten in Funktion der Zeit darzustellen:

```
>>> rtb.xplot(t, trajectory.qd)
```

Damit sind die Geschwindigkeitstrapeze der 4 Gelenke gut sichtbar.
Wenn man die Trajektorie in Kartesischen Koordinaten visualisieren möchte, dann muss man die Gelenkwinkel der Trajektorie zuerst mit der Vorwärtskinematik in Kartesische Koordinaten umrechnen. Das geht wie folgt:

```
>>> T = scara.fkine(trajectory.q)
```

Damit erhält man ein Array von Transformationsmatrizen. Die Translationen, das heisst, die Positionen dieser Transformationsmatrizen können dann wie folgt geplottet werden:

```
>>> rtb.xplot(t, T.t, labels="x y z")
```

Mit diesen Trajektorien bewegt sich der Roboter nicht direkt von der Start- zur Zielposition, sondern in einem grossen Bogen, weil die Gelenkwinkel, und nicht die Kartesischen Koordinaten interpoliert werden.

3.4 Punkt-zu-Punkt Bewegung des Roboters in Kartesischen Koordinaten

Nun soll eine Bewegung des Roboters mit Interpolation in Kartesischen Koordinaten berechnet werden. Dazu wird zuerst eine Trajektorie in Kartesischen Koordinaten geplant. Diese Bewegung entspricht in etwa dem RAPID Befehl `MoveL`.
Wir verwenden wiederum dieselben Kartesischen Start- und Endlagen, also `T1` und `T2`, wie vorhin.

```
>>> T = rtb.tools.ctraj(T1, T2, t)
```

Damit wird ein Array mit Kartesischen Lagen `T` entlang der Kartesischen Trajektorie für die Zeitpunkte in der Liste `t` berechnet.

Um diese Bewegung in einer Animation darzustellen, müssen für die Kartesischen Lagen zuerst die Gelenkwinkel berechnet werden, und zwar wie folgt:

```
>>> trajectory = scara.ikine_LM(T, q0=[0,1,0,0], mask=[1,1,1,0,0,1])
```

Dann können diese Lagen wieder geplottet werden:

```
>>> scara.plot(trajectory.q)
```

Damit entsteht eine lineare Bewegung des Werkzeuges in Kartesischen Koordinaten. Plotten Sie schliesslich noch einmal die Gelenkwinkel in Funktion der Zeit für diese Bewegung:

```
>>> rtb.xplot(t, trajectory.q)
```

Man erkennt hier, dass sich einzelne Gelenke kurzzeitig sehr schnell bewegen müssen. Das war auch schon in der Animation vorhin zu sehen.

4 Simulation eines ABB IRB 1100 Roboters

4.1 Modell des ABB IRB 1100 Roboters

Die Robotics Toolbox for Python bietet die Möglichkeit, mit Robotermodellen zu arbeiten, die im Unified Robot Description Format (URDF) gegeben sind. Dieses Format ist ein XML Dokument, in dem die Links und Gelenke des Roboters definiert sind. Neben den kinematischen Parametern können auch dynamische Parameter definiert werden, wie z.B. Massen und die Lage der Massenmittelpunkte für einzelne Links des Roboters. Und schliesslich können auch grafische Darstellungen der einzelnen Links eines Roboters in diesem Format beschrieben werden.

Wir haben für diese Übung ein Modell des ABB IRB 1100 Roboters im URDF Format gegeben. Bevor dieses Modell geladen werden kann, muss der aktive Pfad in der Python Umgebung unter Umständen noch gesetzt werden.

```
>>> import os
>>> os.getcwd()
```

Damit wird das aktuelle Verzeichnis angezeigt. Es sollte das Verzeichnis sein, in dem sich auch das Verzeichnis mit dem IRB 1100 Modell sowie dem Python File names 'IRB1100.py' befindet. Andernfalls kann es wie folgt gesetzt werden:

```
>>> os.chdir("/path/to/working/directory")
```

Das Modell kann dann wie folgt geladen werden:

```
>>> from IRB1100 import IRB1100
>>> irb1100 = IRB1100()
>>> print(irb1100)
```

Damit erhalten wir ein geometrisches Modell des IRB1100 Roboters. Mit dessen `plot()` Methode kann es auch für gegebene Gelenkwinkel grafisch dargestellt werden:

```
>>> irb1100.plot(irb1100.qr)
```

Damit wird ein detailliertes Modell des ABB IRB 1100 Roboters in seiner Heimposition dargestellt. Die einzelnen Links des Roboters sind dabei als STL Files gegeben.

Mit diesem Robotermodell können nun wieder dieselben Funktionen verwendet werden, wie mit dem Modell des SCARA Roboters in der Aufgabe 3.

4.2 Berechnung der Vorwärts- und Rückwärtskinematik

Die Vorwärtskinematik kann man für gegebene Gelenkwinkel wie folgt berechnen:

```
>>> T = irb1100.fkine([0, np.pi/4, 0, 0, np.pi/4, 0])
>>> T
```

Die Rückwärtskinematik kann für eine gegebene Kartesische Lage mit einem numerischen, iterativen Algorithmus berechnet werden.

```
>>> T = sm.SE3([ [-1, 0,  0,  0.4],
                 [ 0, 1,  0, -0.3],
                 [ 0, 0, -1,  0.2],
                 [ 0, 0,  0,  1]])

>>> pos = irb1100.ikine_LM(T, q0=irb1100.qr)
>>> irb1100.plot(pos.q)
```

Als Startwerte für die iterative Suche der Gelenkwinkel wird hier die Heimposition des Roboters angegeben. Schliesslich wird der Roboter in dieser Lage grafisch dargestellt.

Berechnen Sie nun die Rückwärtskinematik mit anderen Startwerten für die Gelenkwinkel, wie zum Beispiel mit den folgenden:

```
>>> pos = irb1100.ikine_LM(T, q0=[0,0,0,np.pi,-np.pi/4,-np.pi])
>>> irb1100.plot(pos.q)
```

Oder mit diesen hier:

```
>>> pos = irb1100.ikine_LM(T, q0=[2,-1,-2,0,-np.pi/4,0])
>>> irb1100.plot(pos.q)
```

Vergleichen Sie diese beiden letzten Lösungen mit der ersten Lösung der Rückwärtskinematik. Erkennen Sie die Unterschiede?

4.3 Jacobi-Matrix

Mit dem Modell des ABB IRB 1100 Roboters kann für eine gegebene Lage, das heisst, für eine gegebene Gelenkwinkel-Konfiguration die Jacobi-Matrix numerisch berechnet werden. Mit dieser Jacobi-Matrix können Gelenkgeschwindigkeiten des Roboters in Kartesische Geschwindigkeiten des Werkzeugs umgerechnet werden.

```
>>> jacobian = irb1100.jacob0([0, np.pi/4, 0, 0, np.pi/4, 0])
>>> print(jacobian)
```

Berechnen Sie dann die Determinante dieser Jacobi-Matrix, um die Beweglichkeit des Roboters in Kartesischen Koordinaten zu beurteilen. Dazu kann die `det()` Funktion der `numpy` Bibliothek verwendet werden.

```
>>> np.linalg.det(jacobian)
```

Berechnen Sie nun die Determinanten der Jacobi-Matrizen für die Roboter Konfiguration, bei der alle Gelenkwinkel 0 sind, sowie für die Konfiguration, die wir als Heimposition verwenden.

```
>>> jacobian0 = irb1100.jacob0([0, 0, 0, 0, 0, 0])
>>> jacobian1 = irb1100.jacob0([0, 0, 0, 0, np.pi/4, 0])
```

Weshalb wählen wir als Heimposition die Gelenkwinkel [0, 0, 0, 0, pi/4, 0]?

4.4 Punkt-zu-Punkt Bewegung des Roboters

Mit den bisherigen Aufgaben haben Sie alle Funktionen der Robotics Toolbox for Python kennengelernt, um eine Simulation einer einfachen Pick-and-Place Anwendung zu erstellen.

Schreiben Sie ein Python Skript für eine Simulation, bei der sich der Roboter zwischen den folgenden Lagen hin und her bewegt.

```
>>> T1 = sm.SE3([[-1, 0,  0,  0.4],
                 [ 0, 1,  0, -0.3],
                 [ 0, 0, -1,  0.1],
                 [ 0, 0,  0,  1]])

>>> T2 = sm.SE3([[-1, 0,  0,  0.4],
                 [ 0, 1,  0, -0.3],
                 [ 0, 0, -1,  0.4],
                 [ 0, 0,  0,  1]])

>>> T3 = sm.SE3([[-1, 0,  0, 0.4],
                 [ 0, 1,  0, 0.3],
                 [ 0, 0, -1, 0.4],
                 [ 0, 0,  0, 1]])

>>> T4 = sm.SE3([[-1, 0,  0, 0.4],
                 [ 0, 1,  0, 0.3],
                 [ 0, 0, -1, 0.1],
                 [ 0, 0,  0, 1]])
```

Berechnen Sie Trajektorien zwischen diesen Lagepunkten, und zwar Trajektorien in Kartesischen Koordinaten zwischen den Lagen `T1` und `T2`, sowie zwischen `T3` und `T4`, und eine Trajektorie in Gelenkkoordinaten zwischen den Lagen `T2` und `T3`.
Dies geschieht mit den folgenden Funktionen der Robotics Toolbox:

```
>>> t = np.arange(0, 2, 0.001)

>>> pos2 = irb1100.ikine_LM(T2, q0=irb1100.qr)
>>> pos3 = irb1100.ikine_LM(T3, q0=irb1100.qr)

>>> trajectory1 = irb1100.ikine_LM(rtb.tools.ctraj(T1, T2, t),
                                   q0=irb1100.qr)
>>> trajectory2 = rtb.tools.mtraj(rtb.tools.trapezoidal, pos2.q,
                                  pos3.q, t)
>>> trajectory3 = irb1100.ikine_LM(rtb.tools.ctraj(T3, T4, t),
                                   q0=irb1100.qr)
```

Danach kann man die Listen von Gelenkkoordinaten der einzelnen Trajektorien zusammenfügen:

```
>>> traj=np.concatenate((trajectory1.q,trajectory2.q,trajectory3.q))
```

Und schliesslich kann man die Bewegung des Roboters entlang dieser Trajektorie grafisch darstellen. Am einfachsten kopieren Sie den folgenden Kode in ein Python Skript und führen dieses aus:

```
from swift import Swift

env = Swift()
env.launch(realtime=True)
env.add(irb1100)

for i in range (0, traj.shape[0]):
    irb1100.q = traj[i]
    env.step(0.001)
```

Damit wird eine Pick-and-Place Bewegung des ABB IRB 1100 Roboters simuliert.

5 Steuern des ABB IRB 1100 Roboters mit Python

5.1 Inbetriebnahme der Python Steuerung

Nun soll anstelle eines Simulationsmodells der echte ABB IRB 1100 Roboter in eine gewünschte Lage bewegt werden. Dazu wird der Roboter mit einem TCP/IP Protokoll über Ethernet von einem Python Programm gesteuert.
Damit das geht, muss auf der Robotersteuerung das Programm 'MainModule.mod', welches diesem Praktikum beiliegt, gestartet werden. Dieses Roboterprogramm startet einen TCP/IP Server, der Zeichenketten von einem PC empfängt, diese Zeichenketten verarbeitet und ein entsprechendes RAPID Kommando ausführt.

Das Gegenstück in Python ist eine Klasse mit verschiedenen Funktionen, die unter dem Namen 'OmniCore.py' ebenfalls diesem Praktikum beiliegt. Stellen Sie sicher, dass diese Klasse im aktiven Pfad der Python Umgebung liegt.
Von dieser Klasse 'OmniCore' muss nun ein Objekt erstellt werden, welches die verschiedenen Funktionen anbietet. Mit dem Kreieren des Objektes muss noch die IP-Adresse der Robotersteuerung angegeben werden, mit der via Ethernet kommuniziert werden soll. Wählen Sie dazu die IP-Adresse des Service-Ports der Steuerung, so wie im folgenden Beispiel:

```
>>> from OmniCore import OmniCore
>>> robot = OmniCore("192.168.125.1")
```

Damit wird eine TCP/IP Verbindung mit der Robotersteuerung aufgebaut, über die verschiedene Kommandos an den Roboter gesendet werden können. Um nun den Roboter in eine gewünschte Lage zu bewegen kann die folgende Funktion der 'OmniCore' Klasse aufgerufen werden:

```
>>> robot.moveAbsJ([30, 30, -20, 45, 90, -45])
```

Dieser Funktion muss ein Array mit den 6 gewünschten Gelenkwinkeln übergeben werden. Diese Gelenkwinkel werden hier in der Einheit Grad übergeben. Wenn also die Winkel zum Beispiel mit der `ikine_LM()` Funktion des Robotermodells aus der Aufgabe 4 berechnet werden, müssen die Werte noch von rad in Grad umgerechnet werden!

Der Roboter kann mit der folgenden Funktion auch in seine Null-Lage bewegt werden, also in die Lage, in der alle Gelenkwinkel 0 sind.

```
>>> robot.moveAbsJ([0, 0, 0, 0, 0, 0])
```

Um den Roboter wieder in seine Parkposition zu bewegen, kann die Funktion `home` der 'OmniCore' Klasse aufgerufen werden.

```
>>> robot.home()
```

Die Kommunikation kann wieder abgebrochen werden, indem das Objekt `robot` gelöscht wird.

```
>>> del robot
```

Dabei wird eine `delete` Funktion der 'OmniCore' Klasse aufgerufen, welche der Robotersteuerung zuerst mitteilt, dass die TCP/IP Verbindung geschlossen werden soll, und die Verbindung auf der Seite des PCs dann auch schliesst.

5.2 Roboterprogramm in Python

Schreiben Sie nun ein Python Skript für eine Applikation, bei der sich der Roboter zwischen den Lagen `T1`, `T2`, `T3` und `T4` bewegt, analog zur Aufgabe 4.4 mit dem Simulationsmodell.

Diese Lagen müssen zuerst mit der `ikine_LM()` Funktion des Robotermodells aus der Aufgabe 4.2 in die Gelenkwinkel umgerechnet werden. Diese Gelenkwinkel können dann zuerst in Winkelgrad umgerechnet, und dann der Funktion `moveAbsJ()` des `robot` Objektes übergeben werden, welche den echten Roboter in diese Lage bewegt.

6 Modelle von weiteren Robotern

6.1 Modell eines Franka Emika Panda Roboters

Die Robotics Toolbox for Python beinhaltet viele verschiedene Robotermodelle, die direkt geladen werden können. Eines dieser Robotermodelle ist der 'Panda' von Franka Emika. Es wird wie folgt geladen und angezeigt:

```
>>> panda = rtb.models.URDF.Panda()
>>> panda.plot(q=[0.1, 0.2, 0.3, -1.8, -0.2, 2.5, 0.7])
```

Wie Sie sehen, ist dies ein kollaborativer Industrieroboter mit 7 Gelenken.

6.2 Modell eines Universal Robots UR10 Roboters

Die Robotics Toolbox beinhaltet auch mehrere Robotermodelle von Universal Robots. Ein Modell des UR10 kann zum Beispiel wie folgt geladen und grafisch dargestellt werden:

```
>>> ur10 = rtb.models.URDF.UR10()
>>> ur10.plot([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
```

Dies ist die Nulllage von UR Robotern. Eine Konfiguration der Gelenkwinkel, die für typische Aufgaben besser geeignet ist, ist die folgende:

```
>>> ur10.plot([0.0, -1.0, 2.0, -1.0-np.pi/2, -np.pi/2, 0.0])
```