

Ausgewählte Kapitel der Robotik

Abschlussprojekt zur Vorlesung

Gruppe 4

Dozent: Prof. Andreas Hoch
Betreuer: B.Eng. Fabian Finkbeiner

Autoren: Semyon Kondratev 207612
Lisa-Franziska Schäfer 199318

Inhaltsverzeichnis

| | |
|---|-----------|
| 1 Einleitung | 1 |
| 2 Modellbildung des UR10 | 2 |
| 2.1 Download und Konvertierung der step-Dateien | 2 |
| 2.2 Modellierung in Simscape | 2 |
| 2.3 Integration eines Greifersystems und einer Roboter-Halterung | 3 |
| 2.4 Festlegung der Roboterkoordinatensysteme nach Denavit-Hartenberg-Konvention | 4 |
| 3 Darstellung einer Bewegungssequenz | 4 |
| 4 Ausgabe von Drehmomentverläufen | 6 |
| 5 Vergleichsrechnung mit Newton-Euler-Verfahren | 7 |
| 5.1 Abschätzung von Massen und Trägheiten. Ermittlung von Rotationsmatzen, p- und s-Vektoren | 8 |
| 5.2 Berechnung mit Newton-Euler-Verfahren | 9 |
| 5.2.1 „ <i>compute_torques_with_newton-euler.m</i> “ | 9 |
| 5.2.2 „ <i>function compute_kinematics()</i> “ | 10 |
| 5.2.3 „ <i>function compute_forces_and_torques()</i> “ | 11 |
| Literaturverzeichnis | 13 |

1 Einleitung

Im Rahmen des Abschlussprojekts zur Vorlesung Ausgewählte Kapitel der Robotik sollen verschiedene Aufgaben rund um den Roboter UR10 der Firma Universal Robots bearbeitet werden. Zunächst soll das Modell des Roboters in Matlab Simscape nachgebildet werden. Dazu können die step-Dateien des Roboters verwendet werden, welche auf einigen Webseiten als kostenloser Download zur Verfügung stehen. Die Massen und Trägheiten der Armteile bzw. Gelenke sollen dabei abgeschätzt werden. Als Nächstes soll die zugewiesene Bewegungssequenz aus dem Beispielvideo dargestellt werden. Die Bewegung umfasst dabei eine etwa drei Sekunden lange „*Point to Point*“ Bewegung des UR10. Eine weitere Aufgabe ist, die Drehmomentverläufe der Antriebe des Modells in Simscape auszugeben. Als Letztes sollen die von der Simulation berechneten Drehmomente mittels einer Vergleichsrechnung mit Newton-Euler-Verfahren überprüft werden.

Beim UR10 handelt es sich um einen 6-Achs Roboterarm, welcher in der Abbildung 1 zu sehen ist. Sein Aufbau ist an einen menschlichen Arm angelehnt, wodurch er nahezu jede Position in seinem Arbeitsraum erreichen kann. Er ist der größte der UR-Familie und kann Lasten von bis zu 10 kg bei einem Arbeitsradius von bis zu 1300 mm bewegen. Dies ermöglicht ihm ein breites Anwendungsspektrum in der Maschinenbestückung, Palettierung und Verpackung.[Uni]



Abbildung 1: UR10 der Firma Universal Robots, Quelle: https://www.universal-robots.com/3d/images/slider/ur10/small_images/rendersd_00009.jpg

2 Modellbildung des UR10

Für die erste Aufgabe soll der Roboter in der Matlab Simulationssoftware Simscape modelliert werden. Anders als in der Aufgabenstellung angegeben, existiert jedoch nur eine einzelne step-Datei über den gesamten Roboter als Download, welche alle Armteile enthält.

2.1 Download und Konvertierung der step-Dateien

Die step-Datei des UR10 wird auf diversen Webseiten kostenlos zum Download angeboten. Für das Projekt wurde die Datei der Firma „SG-Automatisierungstechnik GmbH“¹ verwendet. Die step-Datei lässt sich jedoch nicht direkt in Matlab einlesen, sondern muss mittels des „Simscape Multibody Link Plugin“ konvertiert werden. Dabei handelt es sich um ein Zusatzprogramm, welches in bestimmten CAD Anwendungen installiert werden kann und den Export von XML- und Geometriedateien ermöglicht [Mat]. Als CAD Anwendung für die Konvertierung wurde SolidWorks gewählt. Der Dateiexport aus SolidWorks erzeugte eine XML-Datei, eine Geometriedatei „UR10_DataFile.m“ sowie acht step-Dateien der verschiedenen Armteile und der Basis.

2.2 Modellierung in Simscape

Nun kann die XML-Datei mittels dem Befehl „*sm_import()*“ in Matlab Simscape eingelesen und mit den step-Dateien der Roboterbauteile verknüpft werden. Nach einer kurzen Sichtung des Modells stellte sich heraus, dass die Transformationen zwischen den Bauteilen zwar korrekt, jedoch die interne Verknüpfung des Blockdiagramms nicht stimmig war. Beim Einfügen von Aktuatoren bewegte sich ausschließlich das angesteuerte Armteil, während der Rest der kinematischen Kette statisch in der Ausgangsposition verblieb.

Aus diesem Grund wurde der Import einer URDF-Datei des Roboters versucht, was ebenso über den Befehl „*sm_import()*“ möglich ist. Die verwendete Datei kann aus dem Github-Repository von „Positronics Lab“², einer Forschungsorganisation der George Washington Universität, kostenlos heruntergeladen werden. Mit Einfügen der step-Dateien der Armteile erhält man ein funktionierendes Robotermmodell des UR10. Der originale Output nach Einlesen der URDF-Datei ist in der Abbildung 2 zu sehen. Die Massen und Trägheiten der Armteile wurden aus der zuvor erhaltenen Geometriedatei entnommen.

¹<https://www.sg-automation.at/1904594/Downloads-UR10>

²https://github.com/PositronicsLab/reveal_packages/blob/master/industrial_arm/scenario/models/urdf/ur10/ur10.urdf

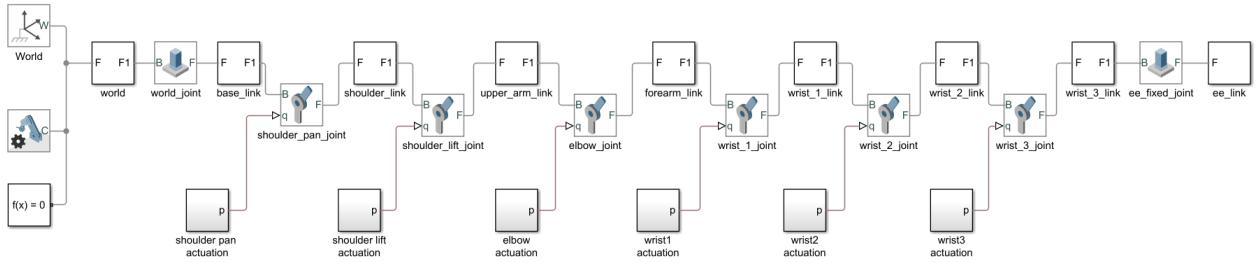


Abbildung 2: Originales Blockdiagramm des UR10 aus der URDF-Datei

2.3 Integration eines Greifersystems und einer Roboter-Halterung

Um dem gezeigten Roboter aus dem Beispielvideo möglichst nahe zu kommen, wurde zusätzlich eine Halterung und ein Greifersystem konstruiert. Die Halterung besteht aus einem quaderförmigen Block, bei dem eine Kante nach oben verschoben ist. Dadurch entsteht eine schiefe Ebene, auf der das RobotermodeLL wie im Video montiert werden kann.

Das Greifersystem ist aus mehreren Teilen zusammengebaut, einer Werkzeughalterung, zwei Greifern und einem Zylinderstift. Die Werkzeughalterung ist ein Prisma mit trapezförmiger Grundfläche. Der Zylinderstift, welcher im Beispielvideo zum Betätigen eines Tasters verwendet wird, ist an einer geraden Fläche an der Haltung befestigt. Für die beiden seitlich angebrachten Greifer wurde die step-Datei eines möglichst ähnlich aussehenden Greifers von der Website „TraceParts“³ verwendet. Das fertige RobotermodeLL mit Halterung und Greifersystem ist in Abbildung 3 dargestellt.

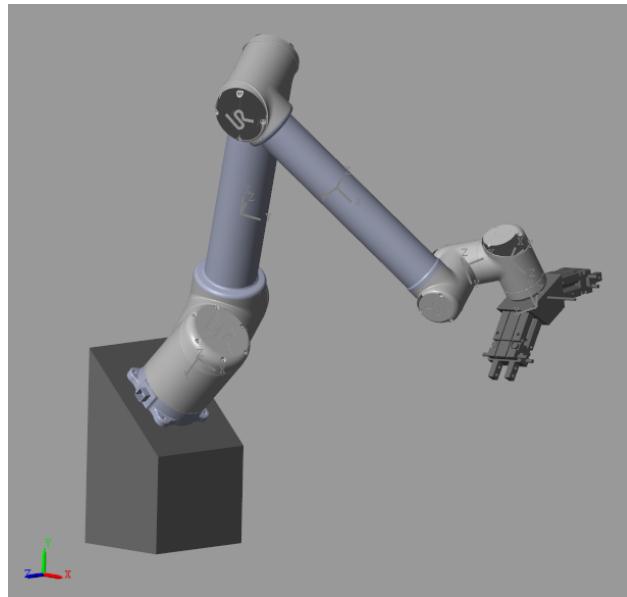


Abbildung 3: UR10 mit Halterung und Greifersystem

³<https://www.traceparts.com/de/product/apore-2backenparallelgreifer?CatalogPath=APORE%3AAPORE.040.010.010&Product=10-28092012-071699>

2.4 Festlegung der Roboterkoordinatensysteme nach Denavit-Hartenberg-Konvention

Für die spätere Vergleichsrechnung mit dem Newton-Euler-Verfahren ist es von Vorteil die Koordinatensysteme der Gelenkachsen des Roboters einheitlich zu definieren. Hierzu bietet sich die Festlegung nach Denavit-Hartenberg-Konvention an. Die nachfolgende Abbildung 4 zeigt die Koordinatensysteme, wie sie auch im Blockdiagramm definiert sind. Der TCP wurde dabei an die Spitze eines Greifers gelegt.

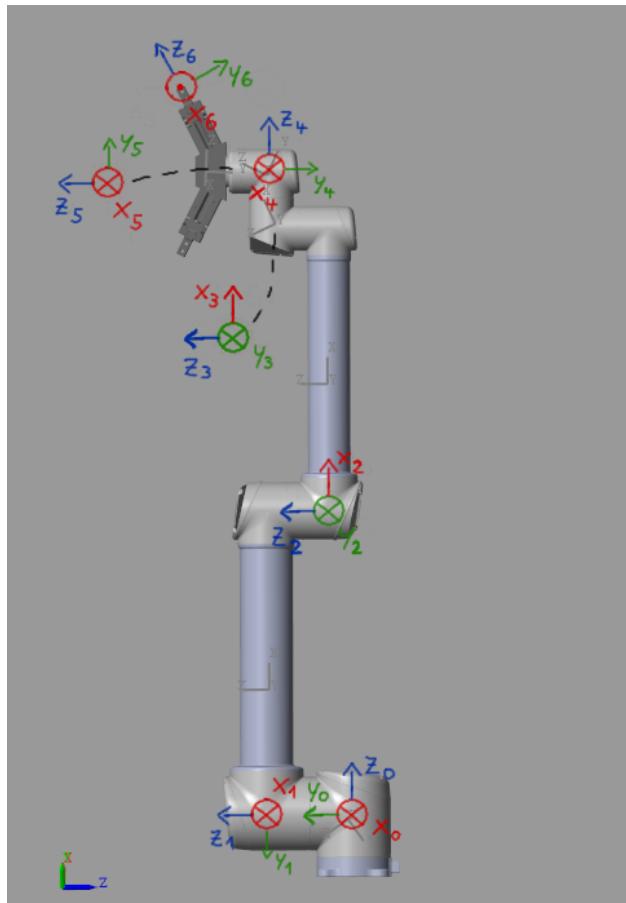


Abbildung 4: Koordinatensysteme der Gelenke und des TCP nach Denavit-Hartenberg-Konvention

3 Darstellung einer Bewegungssequenz

Die geforderte PTP Bewegung des Roboters soll die Zeitstempel 1:05 bis 1:08 im Beispielvideo umfassen. Die daraus resultierende Start- und Endposition ist in den Abbildungen 5a und 5b zusehen. Die Startposition ist mittig an der Türe, die Endposition an der Steuerung auf der rechten Seite. Nach Erreichen der Endposition betätigt der Roboter einen Taster mit seinem



(a) Startposition

(b) Endposition

Abbildung 5: Start- und Endposition der PTP Bewegung,
Quelle: <https://www.youtube.com/watch?v=ugxDTtmykE4>

am Greifer montierten Zylinderstift und startet dadurch die Werkstückbearbeitung in der Maschine. Diese Tasterbetätigung ist in der Bewegungssequenz ebenfalls umgesetzt.

Die Bewegung ist mittels separater Ansteuerung jeder Gelenkachse realisiert. Dazu wird ein Block namens „*Polynomial Trajectory*“ verwendet, welcher Trajektorien durch gegebene Weg- und Zeitpunkte generiert. Der verwendete Block ist in Abbildung 6 auf der linken Seite zu sehen. In unserem Fall handelt es sich bei den Wegpunkten um Winkelstellungen des Gelenks. Daher ist nach dem Trajektoriengenerator ein „*Simulink-PS-Converter*“ eingefügt, der das resultierende einheitenlose Signal des Generators in ein physikalisches mit der Einheit Rad umwandelt.

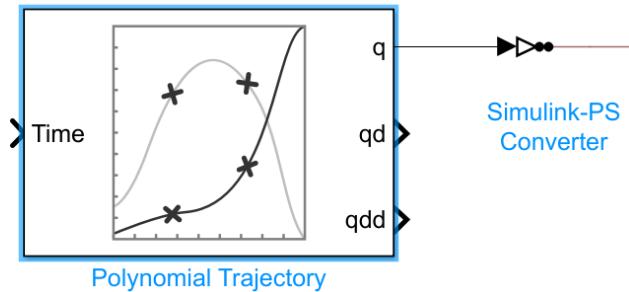


Abbildung 6: Blockdiagramm des Aktuators des ersten Gelenks

Für die erste Gelenkachse sind die folgenden Zahlenwerte für die Weg- und Zeitpunkte eingetragen:

$$\text{Wegpunkte : } [0, 0.61, 0.61, 0.52, 0.52, 0.576, 0.576, 0.52, 0.52, 0]$$

$$\text{Zeitpunkte : } [0, 1, 2, 5, 5.5, 6, 6.5, 7, 7.5, 10]$$

Im „*data_inspector*“ von Matlab Simulink kann das erzeugte Signal des Trajektoriengenerators eingesehen werden. Die Abbildung 7 zeigt beispielhaft wieder die erste Gelenkachse. Zum

erleichterten Verständnis sind die verschiedenen Positionen des Roboters und die Betätigung des Tasters im Signal markiert.

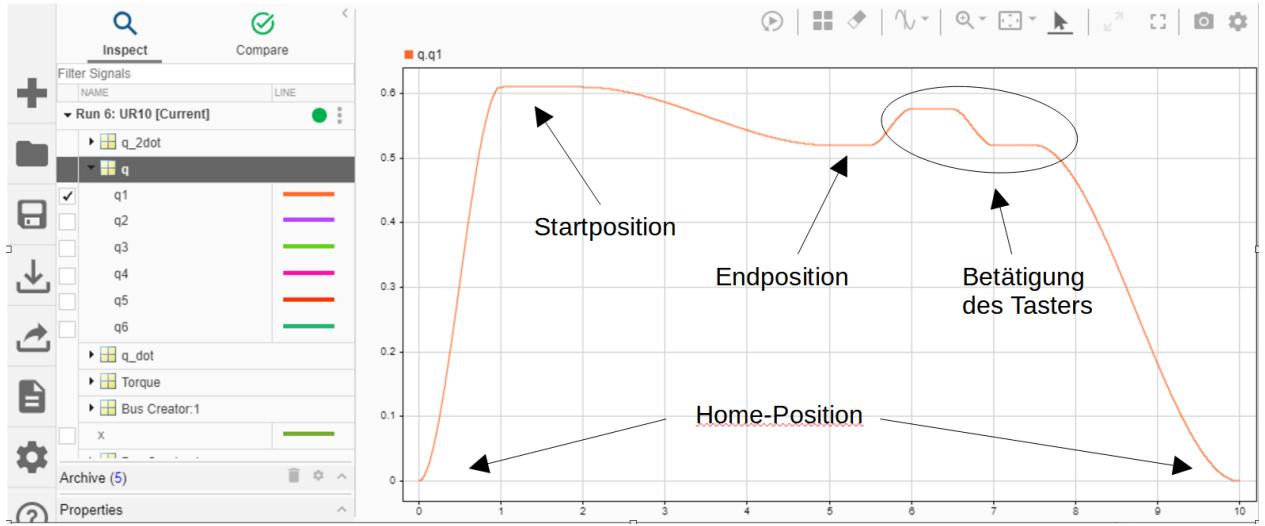
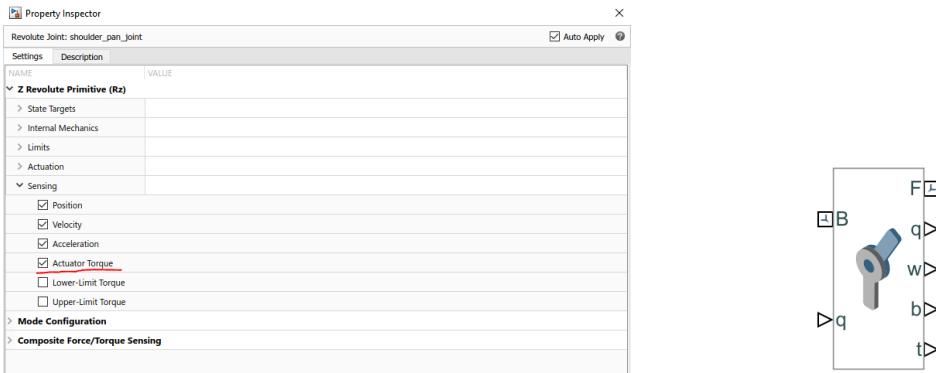


Abbildung 7: Winkelverlauf über Zeit des ersten Gelenks

4 Ausgabe von Drehmomentverläufen

Damit die Drehmomente der Antriebe in der Simulation ausgegeben werden können, muss die „*Actuator Torque*“-Variable des jeweiligen „*Revolute Joint*“ aktiviert werden. Die Variable ist im Bereich „*Sensing*“ zu finden, wie Abbildung 8a zeigt. Auf Abbildung 8b ist der entsprechende Simscape Block mit aktiverter „*Actuator Torque*“-Variable zu sehen.



(a) Property Inspector des „*Revolute Joint*“-Blocks mit „*Actuator Torque*“-Checkbox (b) Revolute Joint Simscape Block mit aktivierter „*Actuator Torque*“-Variable

Abbildung 8: „Property Inspector“ des „*Revolute Joint*“ und Simscape Block

Danach werden die Ausgaben der Drehmomente aller Gelenke mit dem „*BusCreator*“ verbunden, welcher in Abbildung 9 auf der linken Seite dargestellt ist. Die Daten werden dann durch die „*Log Signals*“-Funktion von MATLAB eingeloggt.

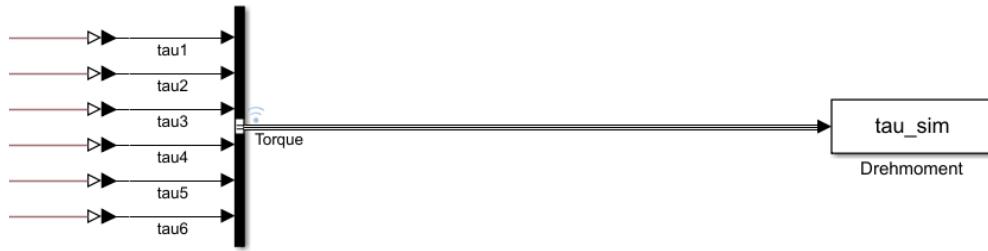


Abbildung 9: Torque Bus Line and Log Sign

Die Ergebnisse können im „*Data Inspector*“ (Abbildung 10) eingesehen werden.

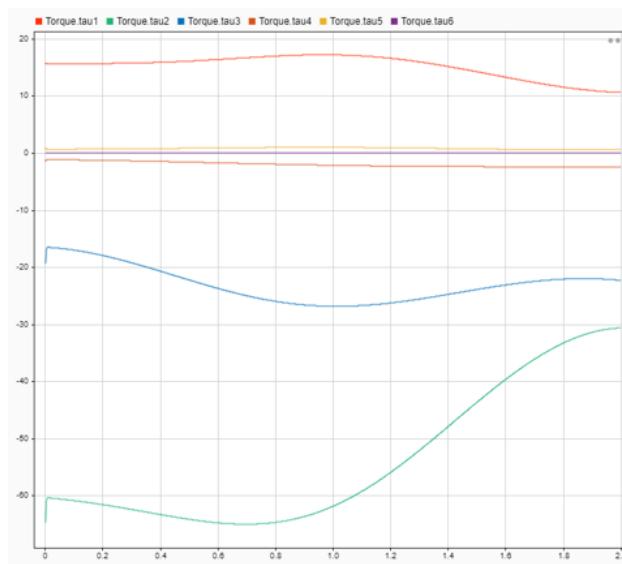


Abbildung 10: Ausgabe der Drehmomente im „*Data Inspector*“

5 Vergleichsrechnung mit Newton-Euler-Verfahren

Die Rechnung wird für debugging-Ziele und eine leichtere Simulation (weniger Rechenintensiv) nur nach der Simulation durchgeführt. Deswegen sollen die Daten aus der Simulation in den MATLAB-Workspace übertragen werden. Für diese Zwecke kann der „*To Workspace*“ MATLAB-Block verwendet werden, welcher in Abbildung 11 dargestellt ist. Die Daten in dieser Arbeit werden für jeden Zeitpunkt der Simulation gespeichert.

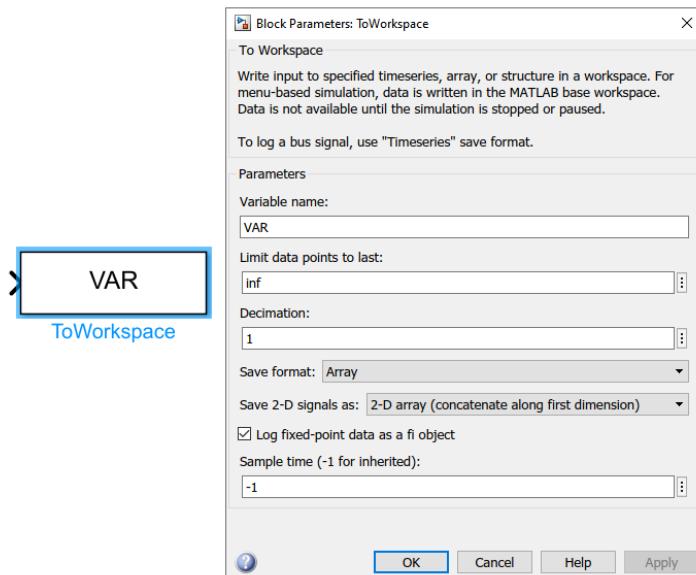


Abbildung 11: „To Workspace“-Block von Matlab

5.1 Abschätzung von Massen und Trägheiten. Ermittlung von Rotationsmatzen, p- und s-Vektoren

Für die Schätzung der Massen, Trägheiten und s-Vektoren aus der Simulation, steht in Simulink der „Inertia Sensor“ zur Verfügung. Für die Rotationsmatrizen sowie p-Vektoren wird ein „Transformation Sensor“ verwendet. Die Sensoren werden in einem Subsystem eingeschlossen. Der linke Port des Subsystems entspricht dem Koordinatensystem i, der rechte Port dem Koordinatensystem i+1. Das Innere des Subsystems ist in der nachfolgenden Abbildung 12 zu sehen.

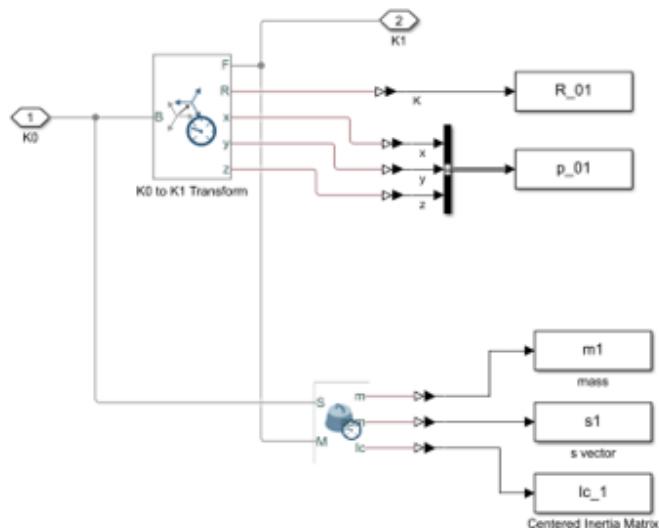


Abbildung 12: Beispiel des Sensorensubsystems

5.2 Berechnung mit Newton-Euler-Verfahren

5.2.1 „compute_torques_with_newton-euler.m“

Dieses Skript wird nach der Simulation ausgeführt (StopFnc-Callback in Simulink). Es führt manche Manipulationen mit den Daten durch, damit sie leichter zu verarbeiten sind.

```
1 if size(q,1) > size(q,2)
2 q = q';
3 q_dot = q_dot';
4 q_2dot = q_2dot';
5 tau_sim = tau_sim';
6 end
7
8 sim_steps = length(q(1,:));
9 n_joints = length(q(:,1));
10
11 omega = zeros(3, n_joints, sim_steps);
12 omega_dot = zeros(3, n_joints, sim_steps);
13 v_dot = zeros(3, n_joints, sim_steps);
14 vs_dot = zeros(3, n_joints, sim_steps);
15 tau = zeros(n_joints, sim_steps);
16
17 % concatenate data from simulation for easier iteration
18 R = cat(4, R_01, R_12, R_23, R_34, R_45, R_56);
19 p = cat(3, p_01', p_12', p_23', p_34', p_45', p_56');
20 s = cat(3, s1', s2', s3', s4', s5', s6');
21 Ic = cat(4, Ic_1, Ic_2, Ic_3, Ic_4, Ic_5, Ic_6);
22 m = cat(1, m1, m2, m3, m4, m5, m6);
23 m = m';
24
25 for i = 1:sim_steps
26 [omega(:,:,i), omega_dot(:,:,i), v_dot(:,:,i), vs_dot(:,:,i)] = ...
27 compute_kinematics(q(:,:,i), q_dot(:,:,i), q_2dot(:,:,i), R(:,:,i,:), ...
28 R_W0(:,:,i), p(:,:,i,:));
29 tau(:,:,i) = compute_forces_and_torques(omega(:,:,i), omega_dot(:,:,i), ...
30 vs_dot(:,:,i), m, Ic(:,:,i,:), p(:,:,i,:), s(:,:,i,:), R(:,:,i,:));
31 end
32
33 ts = timeseries(tau, tout, 'Name', 'tau');
```

5.2.2 „function compute_kinematics()“

Die Funktion in Abbildung 13 repräsentiert die ersten Schritte des Newton-Euler-Verfahrens - „kinematische Berechnung“.

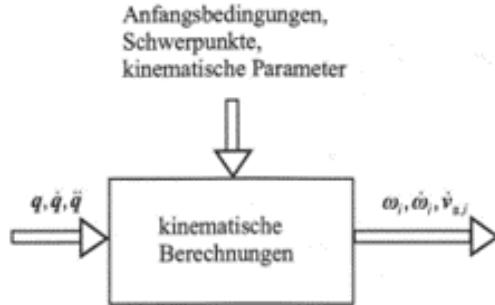


Abbildung 13: Beispiel des Sensorensubsystems, Quelle: Roboterdynamik, VL 11

Die Funktion berechnet die Werte der Geschwindigkeit und Beschleunigungen für K1 mit folgenden Anfangsbedingungen:

$$\omega_0 = 0; \quad \dot{\omega}_0 = 0; \quad \nu = 0; \quad \dot{\nu}_0 = -g$$

Danach berechnet sie die Werte der Geschwindigkeiten und Beschleunigungen von K2 bis K6 iterativ.

```

1 function [omega, omega_dot, v_dot, vs_dot] = compute_kinematics(q, q_dot,
2   q_2dot, R, R_W0, p, s)
3 z = [0 0 1]';
4 n_joints = length(q);
5 %Anfangsbedingungen
6 omega = zeros(3, 6);
7 omega(:,1) = q_dot(1)*z;
8 omega_dot = zeros(3, 6);
9 omega_dot(:,1) = q_2dot(1)*z;
10
11
12 % Beschleunigung des KSs.
13 v_dot(:,1) = inv(R(:,:,1,1)) * (inv(R_W0) * [0 -9.81 0]' ) + ...
14 cross(omega_dot(:, 1), p(:,1,1)) + ...
15 cross(omega(:,1), cross(omega(:,1), p(:,1,1)));
16
17 % Beschleunigung des Schwerpunkts.
18 vs_dot(:,1) = v_dot(:,1) + cross(omega_dot(:, 1), s(:, 1, 1)) + ...
19 cross(omega(:, 1), cross(omega(:,1), s(:, 1, 1)));
20
21
  
```

```

22 for i = 1:n_joints-1
23 invR = inv(R(:, :, 1, i));
24
25 omega(:, i+1) = invR * (omega(:, i) + q_dot(i+1)*z);
26 omega_dot(:, i+1) = invR * (omega_dot(:, i) + ...
27 (z * q_2dot(i+1) + cross(omega(:, i), q_dot(i+1) * z)));
28
29 v_dot(:, i+1) = invR * v_dot(:, i) + ...
30 cross(omega_dot(:, i+1), p(:, 1, i+1)) + ...
31 cross(omega(:, i+1), cross(omega(:, i+1), p(:, 1, i+1)));
32 vs_dot(:, i+1) = v_dot(:, i+1) + ...
33 cross(omega_dot(:, i+1), s(:, 1, i+1)) + ...
34 cross(omega(:, i+1), cross(omega(:, i+1), s(:, 1, i+1)));
35 end
36 end

```

Folgende Gleichungen werden in der Funktion verwendet (aus Roboterdynamik, VL 11):

$$\boldsymbol{\omega}_{i+1} = {}_{i+1}^i A (\boldsymbol{\omega}_i + \boldsymbol{h}_{i+1} \cdot \boldsymbol{z} \cdot \dot{\boldsymbol{q}}_{i+1})$$

$$\dot{\boldsymbol{\omega}}_{i+1} = {}_{i+1}^i A [\dot{\boldsymbol{\omega}}_i + \boldsymbol{h}_{i+1} \cdot (\boldsymbol{z} \cdot \ddot{\boldsymbol{q}}_{i+1} + \boldsymbol{\omega}_i \times \boldsymbol{z} \cdot \dot{\boldsymbol{q}}_{i+1})],$$

$$\dot{\boldsymbol{v}}_{i+1} = {}_{i+1}^i A \cdot \dot{\boldsymbol{v}}_i + \dot{\boldsymbol{\omega}}_{i+1} \times \boldsymbol{p}_{i+1} + \boldsymbol{\omega}_{i+1} \times (\boldsymbol{\omega}_{i+1} \times \boldsymbol{p}_{i+1})$$

$$\dot{\boldsymbol{v}}_{s,i+1} = \dot{\boldsymbol{v}}_{i+1} + \dot{\boldsymbol{\omega}}_{i+1} \times \boldsymbol{s}_{i+1} + \boldsymbol{\omega}_{i+1} \times (\boldsymbol{\omega}_{i+1} \times \boldsymbol{s}_{i+1})$$

5.2.3 „function compute_forces_and_torques()“

Die Funktion in Abbildung 14 repräsentiert den zweiten Schritt des Newton-Euler-Verfahren - „Berechnungen der Kräfte und Drehmomente“.

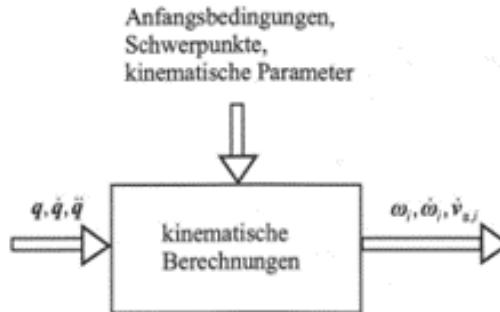


Abbildung 14: Berechnungen der Kräfte und Drehmomente, Quelle: Roboterdynamik, VL 11

```

1 function [tau] = compute_forces_and_torques(omega, omega_dot, vs_dot, m, Ic, p
2 , s, R)
3 z = [0 0 1]';
4 n_joints = length(omega(1,:));
5 % auf n_joints+1 stehen Anfangskraefte- und Drehmomente
6
7 f = zeros(3, n_joints+1);
8 n = zeros(3, n_joints+1);
9 tau = zeros(1, n_joints);
10 R(:,:,1,7) = eye(3,3);
11
12 for i = n_joints:-1:1
13 F(:,i) = m(i) * vs_dot(:, i);
14 f(:,i) = R(:,:,1,i+1) * f(:,i+1) + F(:,i);
15 N(:,i) = Ic(:,:,1,i) * omega_dot(:, i) + cross(omega(:,i), Ic(:,:,1,i) * omega
16 (:,i));
17 n(:,i) = n(:,i+1) + cross(p(:,1,i)+s(:,i), F(:,i)) + cross(p(:,i), f(:,i+1)) +
18 N(:,i);
19 tau(i) = n(:,i)' * inv(R(:,:,1,i)) * z;
20 end
21
22 end

```

Folgende Gleichungen werden in der Funktion verwendet (aus Roboterdynamik, VL 11):

$$\text{Kraft an einem Schwerpunkt: } \mathbf{F}_i = \mathbf{m}_i \cdot \dot{\mathbf{v}}_{s,i}$$

$$\text{Gelenkkraft: } \mathbf{f}_i = {}^{i+1}_i \mathbf{A} \cdot \mathbf{f}_{i+1} + \mathbf{F}_i,$$

$$\text{Wirkendendes Drehmoment: } \mathbf{N}_i = \mathbf{I}_{SP,i} \cdot \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\mathbf{I}_{SP,i} \cdot \boldsymbol{\omega}_i) + h_i \cdot \mathbf{F}_{D,i} \cdot {}^{i-1}_i \mathbf{A} \cdot \mathbf{z} \cdot \dot{q}_i,$$

$$\text{Drehmomente im Gelenk: } \mathbf{n}_i = {}^{i+1}_i \mathbf{A} \cdot [\mathbf{n}_{i+1} + ({}^{i+1}_i \mathbf{A} \cdot \mathbf{p}_i) \times \mathbf{f}_{i+1}] + (\mathbf{p}_i + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{N}_i,$$

$$\text{Drehmomente im Gelenk: } \boldsymbol{\tau}_i = [h_i \cdot \mathbf{n}_i^T + (1-h_i) \cdot \mathbf{f}_i^T] \cdot {}^{i-1}_i \mathbf{A} \cdot \mathbf{z},$$

Literaturverzeichnis

- [Mat] MATHWORKS: *Install the Simscape Multibody Link Plugin - MATLAB & Simulink* - MathWorks Deutschland. <https://de.mathworks.com/help/physmod/smlink/ug/installing-and-linking-simmechanics-link-software.html>, Abruf: 21.02.2022
- [Uni] UNIVERSAL ROBOTS (GERMANY) GMBH: *Mit Cobots Ihre Produktion automatisieren / Universal Robots*. <https://www.universal-robots.com/de/produkte/>, Abruf: 21.02.2022