UR Script

UR Roboter können mit der Python-ähnlichen Sprache UR Script programmiert werden

Fahrbefehle:

```
movej
movel
```

Beispiel:

```
movej([0, -1.57079632, 0, -1.57079632, 0, 0])

movej([d2r(0), d2r(-90), d2r(0), d2r(-90), d2r(0), d2r(0)])
```


UR Script Fahrbefehl

movej(*q*, *a*=1.4, *v*=1.05, *t*=0, *r*=0)

Move to position (linear in joint-space) When using this command, the robot must be at a standstill or come from a movej or movel with a blend. The speed and acceleration parameters controls the trapezoid speed profile of the move. The t parameters can be used in stead to set the time for this move. Time setting has priority over speed and acceleration settings. The blend radius can be set with the r parameters, to avoid the robot stopping at the point. However, if the blend region of this move overlaps with previous or following regions, this move will be skipped, and an 'Overlapping Blends' warning message will be generated.

Parameters

- `q`: joint positions (q can also be specified as a pose, then inverse kinematics is used to calculate the corresponding joint positions)
- `a`: joint acceleration of leading axis (rad/s^2)
- `v`: joint speed of leading axis (rad/s)
- `t`: time (S)
- `r`: blend radius (m)

Ein Punkt im Gelenkwinkelraum wird definiert über []

Ein Punkt im Arbeitsraum wird definiert über p[]

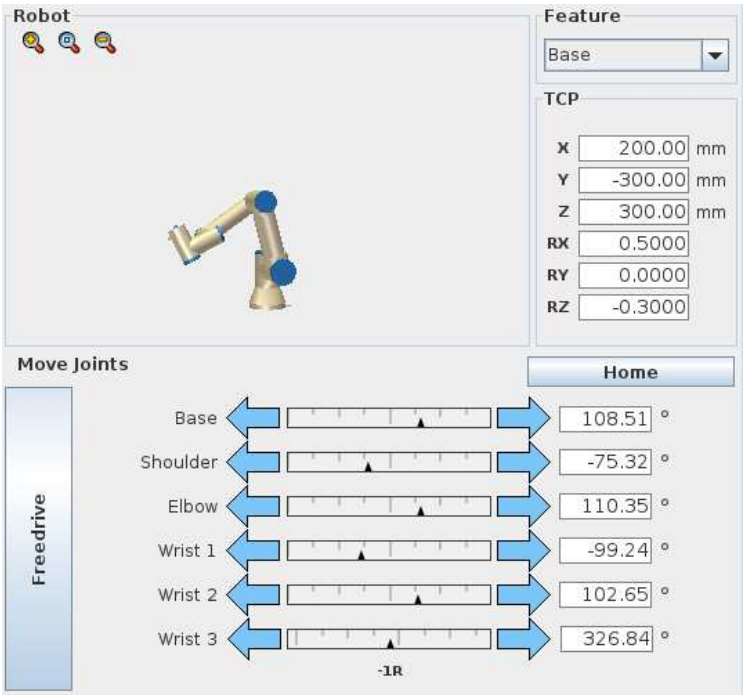
Beispiel:

```
p0 = [d2r(90), d2r(-90), d2r(0), d2r(-90), d2r(0), d2r(0)]

p1 = p[0.200, -0.300, 0.300, 0.5, 0.0, -0.3])
```

p1 entspricht der homogenen Transformationsmatrix:

$$ {}^{0}T_{6} = \begin{bmatrix} 0.95626064 & 0.28328667 & -0.07289894 & 0.2 \\ -0.28328667 & 0.83476241 & -0.47214445 & -0.3 \\ -0.07289894 & 0.47214445 & 0.87850177 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$



`pose_trans(p_from, p_from_to)`

Beispiel:

```
C = pose_trans(A, B)
```

entspricht der Matrixmultiplikation

$$
\begin{aligned}
{}^0T_C &= {}^0T_A\,{}^AT_B \\
&= \begin{bmatrix} {}^0R_A & {}^0t_{0\to A} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^AR_B & {}^At_{A\to B} \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} {}^0R_A\,{}^AR_B & {}^0R_A\,{}^At_{A\to B} + {}^0t_{0\to A} \\ 0 & 1 \end{bmatrix} = {}^0T_C
\end{aligned}
$$

`pose_add(p_1, p_2)`

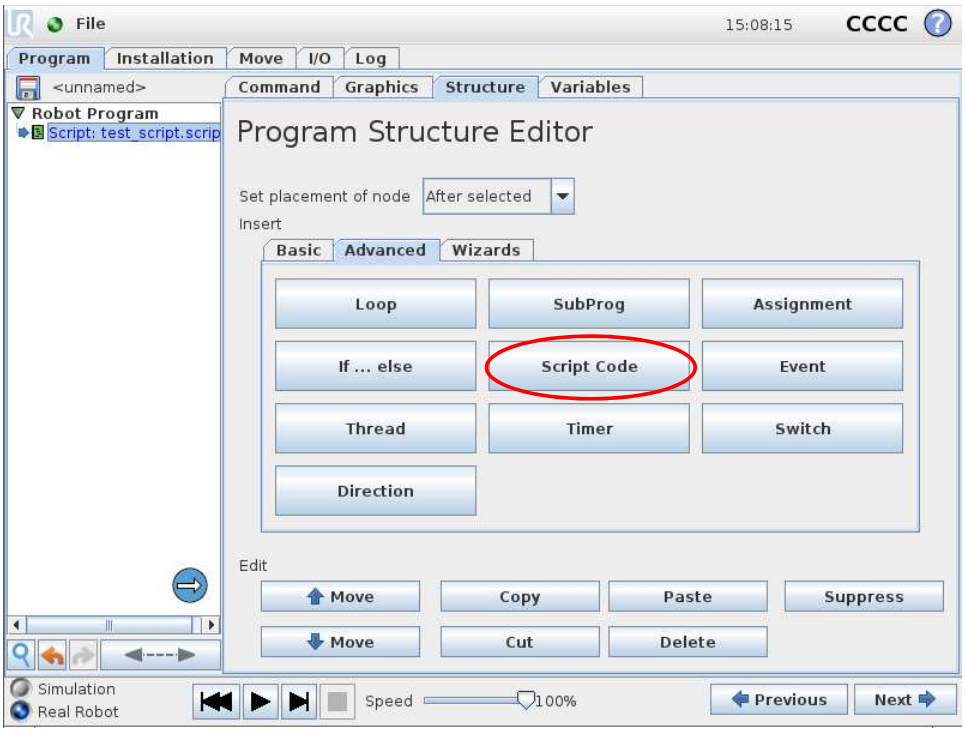
Beispiel:

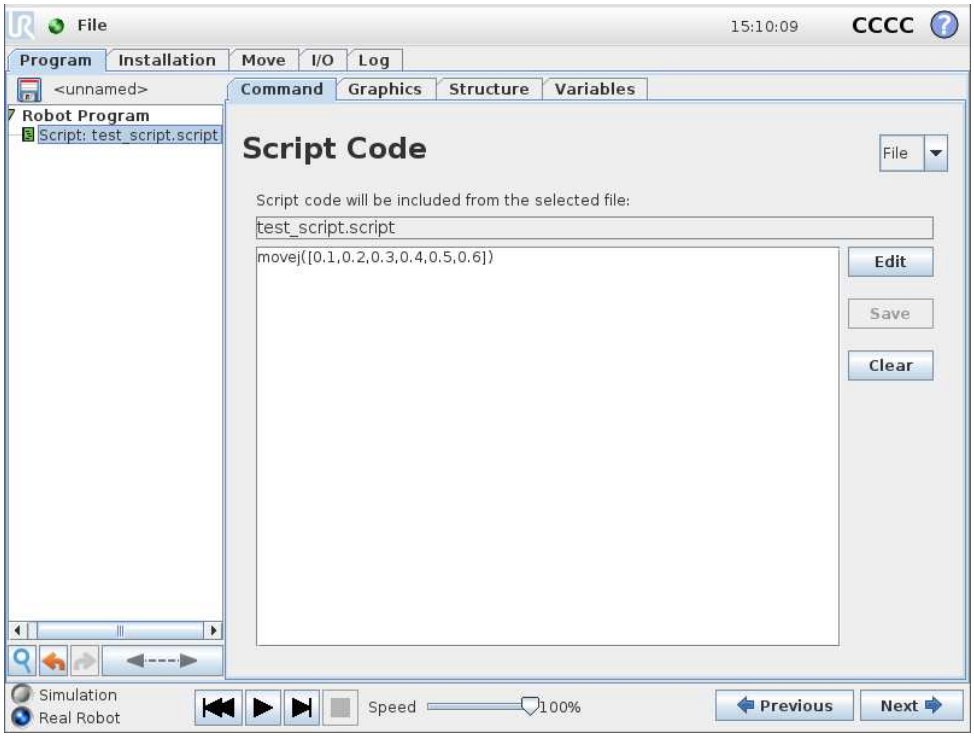
```
C = pose_add(A, B)
```

entspricht der Rechnung

$$
\begin{aligned}
{}^0R_C &= {}^0R_A\,{}^AR_B \\
{}^0t_C &= {}^0t_{0\to A} + {}^0t_{0\to B} \\
&= \begin{bmatrix} {}^0R_A\,{}^AR_B & {}^0t_{0\to A} + {}^0t_{0\to B} \\ 0 & 1 \end{bmatrix} = {}^0T_C
\end{aligned}
$$

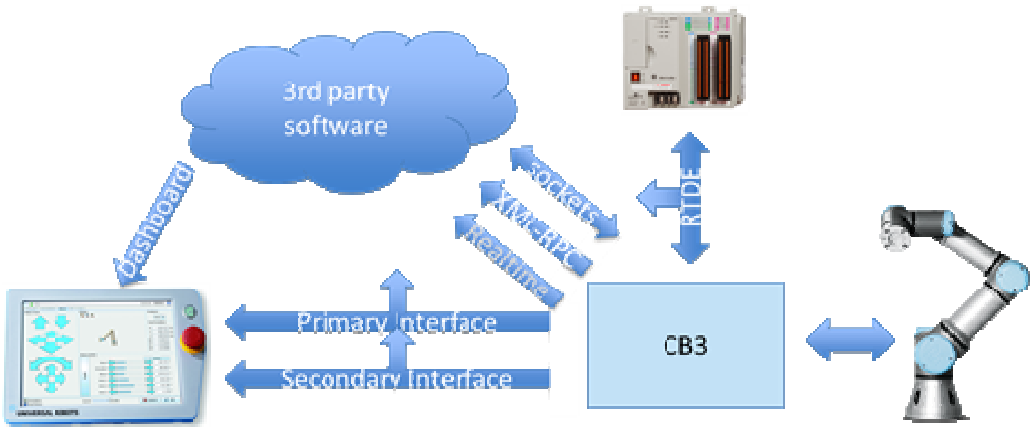
Programmierung über Benutzeroberfläche





Kommunikation über Netzwerk

https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/



Kommunikation über Netzwerk

https://www.universal-robots.com/articles/ur/interface-communication/remote-control-via-tcpip/

e-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	500	500	500
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

CB-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	125	125	125
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

Programmierung über Netzwerk

Die UR Script Programme / Befehle können über Netzwerk an den Roboter geschickt werden, z.B. Secondary Client Interface über Port 30002

Die UR Script Kommandos werden in Klartext übertragen mit Newline am Ende

```
movej([0, -1.57079632, 0, -1.57079632, 0, 0])
```


Python Programm für Netzwerkkommunikation

```
import numpy as np
import socket

HOST = "192.168.232.129"    # robot IP
PORT = 30002                # port

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

command = "movej([0, -1.57079632, 0, -1.57079632, 0, 0])\n"

s.send(command.encode('ascii'))
```

Python Programm für Netzwerkkommunikation

```
import numpy as np
import socket

HOST = "192.168.232.129"    # robot IP
PORT = 30002                # port

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

q1 = 0/180*np.pi
q2 = -90/180*np.pi
q3 = 90/180*np.pi
q4 = 0/180*np.pi
q5 = 90/180*np.pi
q6 = 0/180*np.pi

command = "movej([" + str(q1) +  "," + str(q2) + "," + str(q3)+"," + str(q4)
+"," + str(q5) +"," + str(q6) +"])\n"

s.send(command.encode('ascii'))
```


Roboterdaten empfangen (alt)

Eine Möglichkeit Daten zu empfangen ist über das Real-time Interface am Port 30013

Vorteile: - schnell (125 Hz, also alle 8 ms)
- Datenpaket mit fester Größe (siehe UR-Excel-Datei)

Nachteil: - offiziell „deprecated“ → besser RTDE Interface
- auch nicht benötigte Daten werden gesendet

Paketgröße: in Softwareversion 3.13 1116 Bytes
in Softwareversion 3.15 1220 Bytes

Return to Index (The RealTime interface has been deprecated since version 3.5. It is recommended to use the RTDE Interface, instead)

Meaning	Type	Number of values	Size in bytes	Gnuplot col.	Notes
Message Size	integer	1	4		Total message length in bytes
Time	double	1	8	1	Time elapsed since the controller was started
q target	double	6	48	2 - 7	Target joint positions
qd target	double	6	48	8 - 13	Target joint velocities
qdd target	double	6	48	14 - 19	Target joint accelerations
I target	double	6	48	20 - 25	Target joint currents
M target	double	6	48	26 - 31	Target joint moments (torques)
q actual	double	6	48	32 - 37	Actual joint positions
qd actual	double	6	48	38 - 43	Actual joint velocities
I actual	double	6	48	44 - 49	Actual joint currents
I control	double	6	48	50 - 55	Joint control currents
Tool vector actual	double	6	48	56 - 61	Actual Cartesian coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz is a rotation vector representation of the tool orientation
TCP speed actual	double	6	48	62 - 67	Actual speed of the tool given in Cartesian coordinates
TCP force	double	6	48	68 - 73	Generalised forces in the TCP
Tool vector target	double	6	48	74 - 79	Target Cartesian coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz is a rotation vector representation of the tool orientation
TCP speed target	double	6	48	80 - 85	Target speed of the tool given in Cartesian coordinates
Digital input bits	double	1	8	86	Current state of the digital inputs. NOTE: these are bits encoded as int64_t, e.g. a value of 5 corresponds to bit 0 and bit 2 set high
Motor temperatures	double	6	48	87 - 92	Temperature of each joint in degrees celsius
Controller Timer	double	1	8	93	Controller realtime thread execution time
Test value	double	1	8	94	A value used by Universal Robots software only
Robot Mode	double	1	8	95	Robot mode RobotMo
Joint Modes	double	6	48	96-101	Joint control modes ControlModes
Safety Mode	double	1	8	102	Safety mode SafetyMo
	double	6	48	103 - 108	Used by Universal Robots software only
Tool Accelerometer values	double	3	24	109 - 111	Tool x,y and z accelerometer values (software version 1.7)
	double	6	48	112 - 117	Used by Universal Robots software only
Speed scaling	double	1	8	118	Speed scaling of the trajectory limiter
Linear momentum norm	double	1	8	119	Norm of Cartesian linear momentum
	double	1	8	120	Used by Universal Robots software only
	double	1	8	121	Used by Universal Robots software only
V main	double	1	8	122	Masterboard: Main voltage
V robot	double	1	8	123	Masterboard: Robot voltage (48V)
I robot	double	1	8	124	Masterboard: Robot current
V actual	double	6	48	125 - 130	Actual joint voltages
Digital outputs	double	1	8	131	Digital outputs
Program state	double	1	8	132	Program state
Elbow position	double	3	24	133 - 135	Elbow position
Elbow velocity	double	3	24	136 -138	Elbow velocity
Safety Status	double	1	8	139	Safety status SafetyStat
TOTAL		140	1116		140 values in a 1116 byte package

Python Programm

```
import numpy as np
import socket
import struct

HOST = "192.168.178.83"     # robot IP
PORT = 30013                # port

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

r = s.recv(1116)  # 8 ms

sb = 0      # Start byte
a = struct.unpack('>i', r[sb:sb+4])[0]
print('Message Size: ', a)

sb = 4      # Start byte
t = struct.unpack('!d', r[sb:sb+8])[0]
print('Time: ', t)

sb = 252    # Start byte
d = struct.unpack('!dddddd', r[sb:sb+8*6])
qn = np.array(d)
print('Joints: ', qn/np.pi*180)

sb = 444    # Start byte
d = struct.unpack('!dddddd', r[sb:sb+8*6])
pose = np.array(d)
print('Pose', pose)
```

Anmerkung

```
r = s.recv(1116)
```

Die Funktion fordert vom Roboter ein Datenpaket an. Dieser kann alle 8 ms ein Paket senden.

1. Fall: Empfang ist schneller
→ Roboter sendet erst nach 8 ms das neue Paket, recv ist blockierend

2. Fall: Empfang ist langsamer
→ Roboter puffert intern die Datenpakete in einem FIFO, bei einer neuen Empfangsanfrage wird das erste Paket des Puffers ausgeliefert
Beispiel: alle 2 Sekunden wird recv aufgerufen, die Zeitstempel in den empfangenen Datenpakete unterscheiden sich um 8 ms
(es ist unklar wie groß der Roboter interne FIFO Puffer ist)

Roboterdaten empfangen (neu)

Eine Möglichkeit Daten zu empfangen ist über das RTDE Interface am Port 30004

Vorteile:
- schnell (125 Hz, also alle 8 ms)
- die gewünschten Daten können eingestellt werden
- kann auch für eine Echtzeitregelung verwendet werden

Download der RTDE zip Datei und entpacken

Beispiel zum Aufzeichnen record.py

Beispiel zum Plotten example_plotting.py