ROME1 - Praktikum 5
Sensorgeführte Bewegungsplanung

Das Ziel dieses Praktikums ist es, das Werkzeug am Roboter automatisch wechseln zu können, und mit einem speziellen Werkzeug, einem Distanzsensor, die Bewegung des Roboters entlang einem Objekt dynamisch zu regeln.
Das Roboterprogramm für dieses Praktikum soll in Python geschrieben werden, wobei als Basis für das Arbeiten in Python das RAPID Programm 'MainModule.mod' sowie die Python Klasse 'OmniCore.py' verwendet werden können, welche mit diesem Praktikum gegeben sind. Diese Programme müssen allerdings noch um Funktionen erweitert werden, mit denen der Werkzeugwechsler gesteuert werden kann.

1 Testen des RAPID Programmes und der Python Klasse

Im Verzeichnis dieses Praktikums liegt ein RAPID Programm namens 'MainModule.mod', welches einen TCP/IP Server implementiert und unter anderem Instruktionen für die Bewegung des Roboters mit den Kommandos **MoveAbsJ**, **MoveJ** sowie **MoveL** ausführen kann.
Laden Sie dieses Programm auf die Robotersteuerung, indem Sie es im RobotStudio ins Fenster mit dem aktiven Programm der Steuerung kopieren und anschliessend übernehmen.
Wechseln Sie den Betriebsmodus der Robotersteuerung zu **Auto** und starten Sie dieses Programm.
Die Robotersteuerung wartet nun auf einen Client, der eine TCP/IP Verbindung aufbauen und Instruktionen senden kann.

Kopieren Sie das Dokument 'OmniCore.py', welches diesem Praktikum beiliegt in ein Verzeichnis, in dem Sie auch ihre eigenen Python Skripte für dieses Praktikum ablegen.
Dieses Dokument implementiert eine Klasse mit Funktionen, um eine TCP/IP Verbindung mit der Robotersteuerung aufzubauen und den Roboter mit den oben aufgeführten Kommandos zu bewegen.

Testen Sie das RAPID Programm und die Python Klasse, indem Sie in der Konsole von Python die folgenden Funktionen aufrufen:

```
>>> from OmniCore import OmniCore
>>> robot = OmniCore("192.168.125.1")
>>> robot.moveAbsJ([30, 30, -20, 45, 90, -45])
```

Damit bewegt sich der Roboter in die Lage mit den Gelenkwinkeln in Grad, die mit der Funktion **`moveAbsJ`** als Parameter gegeben sind.
Mit den folgenden Funktionen wird der Roboter schliesslich wieder in die Parkposition bewegt und die Verbindung von Python zur Robotersteuerung geschlossen.

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

Das RAPID Programm und die Python Klasse stellen neu auch die Funktionen **`moveJ`** und **`moveL`** zur Verfügung. Das Kommando **`moveJ`** bewegt den Roboter an eine Kartesische Position mit

Interpolation der Gelenkwinkel, und das Kommando **`moveL`** bewegt den Roboter an eine Position mit Interpolation der Kartesischen Koordinaten.
Diese Kommandos benötigen 5 Parameter, welche die gewünschte Lage des Tools im Raum beschreiben, sowie wie schnell und wie genau sich das Tool zur gewünschten Lage bewegen soll. Die folgende Tabelle 1.1 zeigt das Format und die Beschreibung dieser 5 Parameter.

Parameter	Datentyp	Bemerkung
1	[x, y, z]	Die Position des Tools im Basissystem, gegeben in [mm].
2	$[[r_{11}, r_{12}, r_{13}],$ $[r_{21}, r_{22}, r_{23}],$ $[r_{31}, r_{32}, r_{33}]]$	Eine Rotationsmatrix, welche die Orientierung des Tools im Basissystem beschreibt, oder ...
	$[q_0, q_1, q_2, q_3]$	... ein Array mit den 4 Parametern eines Quaternions.
3	$[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$	Die 6 geschätzten Gelenkwinkel für die gegebene Kartesische Lage, aus denen die Konfiguration des Roboters abgeleitet wird.
4	v	Die Geschwindigkeit des TCP, gegeben in [mm/s].
5	z	Eine Toleranz-Zone, welche definiert, wie nahe sich das Werkzeug zur gewünschen Position bewegen muss, gegeben in [mm].

Tabelle 1.1: Definition der Parameter der Funktionen `moveJ` und `moveL`

Die folgende Python Sequenz bewegt den Roboter an gegebene Positionen und Orientierungen mit Interpolationen der Gelenkwinkel, resp. der Kartesischen Koordinaten. Diese Kommandos dienen als Beispiele für typische Positionen und Orientierungen, die in diesem Praktikum verwendet werden können.

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

>>> robot.moveJ([0, -400, 150], [[0, 1, 0],[1, 0, 0],[0, 0, -1]],
                                [-90, 45, 5, 0, 40, 0], 100, 10)

>>> robot.moveL([0, -400, 50], [[0, 1, 0],[1, 0, 0],[0, 0, -1]],
                               [-90, 60, 0, 0, 30, 0], 100, 0)

>>> robot.moveJ([350, 100, 150], [[-1, 0, 0],[0, 1, 0],[0, 0, -1]],
                                 [15, 40, 15, 0, 35, 15], 100, 10)

>>> robot.moveL([350, 100, 50], [[-1, 0, 0],[0, 1, 0],[0, 0, -1]],
                                [15, 55, 10, 0, 25, 15], 100, 0)

>>> robot.moveL([350, 300, 50], [[-1, 0, 0],[0, 1, 0],[0, 0, -1]],
                                [40, 65, -20, 0, 40, 40], 50, 0)

>>> del robot
```

Zuerst wird ein `OmniCore` Objekt erzeugt, wobei eine TCP/IP Verbindung mit der Robotersteuerung aufgebaut wird. Dann folgt die Sequenz von Kommandos mit Fahrbefehlen, und schliesslich wird das `OmniCore` Objekt wieder gelöscht, und damit die Verbindung zur Robotersteuerung wieder geschlossen.

2 Erweitern der Programme für digitale Ausgänge

Das RAPID Programm und die Python Klasse 'OmniCore.py' sollen mit Funktionen erweitert werden, um das Werkzeug zu wechseln und um den Greifer zu öffnen und zu schliessen. Die Python Klasse soll mit den folgenden 3 Funktionen erweitert werden:

```
def tool(self, state):
  ...

def air(self, state):
  ...

def gripper(self, state):
  ...
```

Die erste Funktion `tool` soll ermöglichen, abhängig vom Parameter `state` das Werkzeug zu lösen, oder umgekehrt das Werkzeug zu greifen. Die zweite Funktion `air` soll die Druckluft zum Werkzeugwechsler freigeben, so dass der Wechsler und der Greifer bedient werden können, oder sperren, so dass nicht unnötig Druckluft entweicht. Die dritte Funktion `gripper` soll abhängig vom Parameter `state` den Greifer öffnen oder schliessen.
Die Variable `state` kann dabei eine numerische Zahl mit den Werten 0 (falsch) oder 1 (wahr) sein.
Diese neuen Python Funktionen müssen im RAPID Programm mit einem erweiterten Kommunikationsprotokoll unterstützt werden.
Die Funktion `tool` muss in RAPID den digitalen Ausgang setzen oder zurücksetzen, mit dem das Werkzeug am Roboter gelöst oder wieder gegriffen werden kann:

```
SetDO DO1_Tool, 1;  ! Werkzeug wird gelöst
SetDO DO1_Tool, 0;  ! Werkzeug wird gegriffen
```

Die Funktion `air` muss in RAPID die beiden digitalen Ausgänge setzen oder zurücksetzen, um die Druckluft für den Werkzeugwechsler freizugeben oder zu sperren. Mit den folgenden Kommandos wird die Druckluft freigegeben:

```
SetDO DO3_Enable_1, 0;  ! Druckluft ist aktiv
SetDO DO4_Enable_2, 1;
```

Um die Druckluft wieder zu sperren, so dass nicht unnötig Luft entweicht, müssen die folgenden Kommandos aufgerufen werden:

```
SetDO DO3_Enable_1, 1;  ! Druckluft ist gesperrt
SetDO DO4_Enable_2, 0;
```

Die Funktion `gripper` muss einen digitalen Ausgang zum schliessen und öffnen des Greifers bedienen. Mit den folgenden Kommandos wird der Greifer geschlossen respektive wieder geöffnet:

```
SetDO DO2_Gripper, 1;  ! Greifer schliessen
SetDO DO2_Gripper, 0;  ! Greifer öffnen
```

3 Automatisches Wechseln des Werkzeuges

Implementieren Sie nun ein Ablaufprogramm, um mit Hilfe der neuen Funktionen der Python Klasse 'OmniCore.py' das Werkzeug mit dem Roboter automatisch zu wechseln. Für dieses Praktikum setzen

wir zwei verschiedene Werkzeuge ein, den klassischen 2-Finger Greifer, sowie einen Distanzsensor. Diese beiden Werkzeuge können in einer Werkzeughalterung auf der linken Seite des Roboters abgelegt werden (siehe Abbildung 3.1).

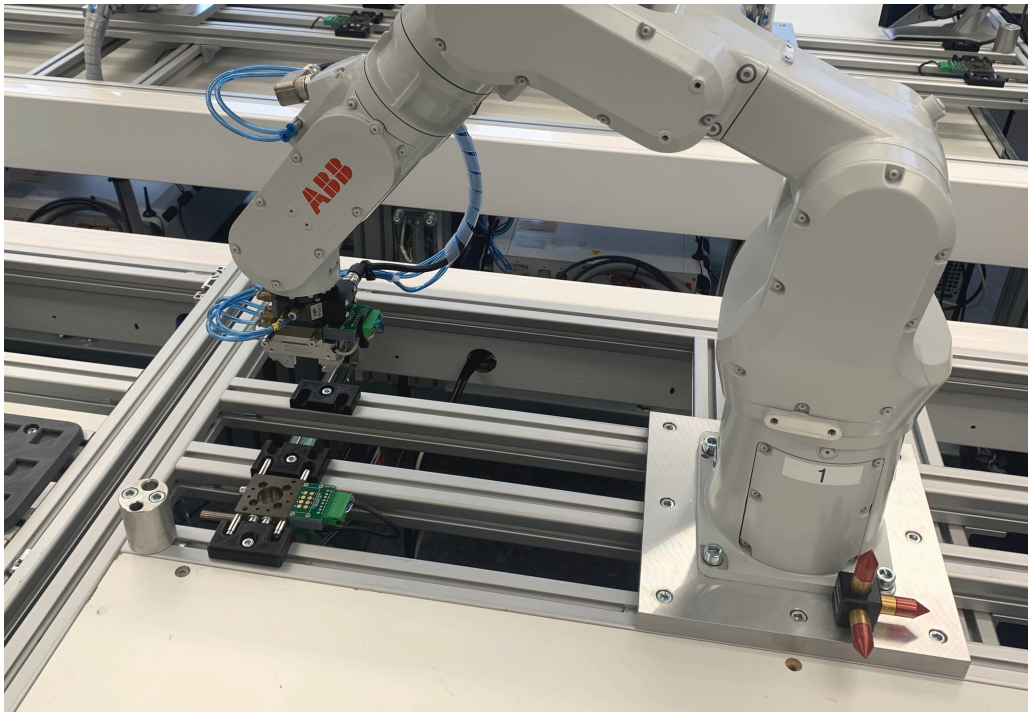


Bild 3.1: Werkzeughalterungen auf der linken Seite des Roboters

Das Ablaufprogramm soll nun das aktive Werkzeug, das sich am Roboter befindet in die leere Position der Werkzeughalterung bewegen, den Werkzeugwechsler lösen, den Roboter mit einer linearen Bewegung vertikal nach oben aus dem Werkzeug bewegen, und schliesslich die Druckluft zum Werkzeugwechsler deaktivieren.
Anschliessend soll das Ablaufprogramm den Roboter über die Position des anderen Werkzeuges in der Halterung bewegen, die Druckluft freigeben, mit einer linearen Bewegung vertikal nach unten ins Werkzeug fahren, und schliesslich das Werkzeug greifen und den Roboter wieder nach oben bewegen.

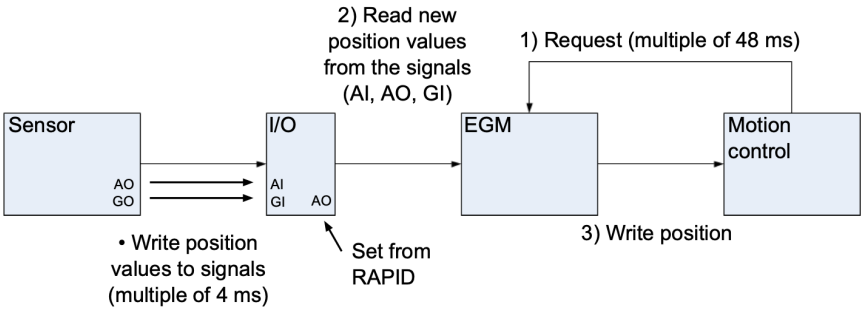
Die Positionen, an denen die Werkzeuge abgelegt oder wieder gegriffen werden, können am einfachsten mit dem Joystick des FlexPendant manuell angefahren werden. In der Funktion **Jog** werden im Display die aktuellen Kartesischen Koordinaten angezeigt. Übernehmen Sie dann einfach diese Kartesischen Koordinaten in Ihr Ablaufprogramm in Python.

Achtung!
Das Einlegen von Werkzeugen in die Werkzeughalterung und das Holen von Werkzeugen von der Werkzeughalterung sind sehr kritische Operationen!
Kleine Flüchtigkeitsfehler können die Werkzeughalterung schnell beschädigen!
Testen Sie Ihr Programm zuerst ohne Werkzeug.

4 Externally Guided Motion (EGM)

Die OmniCore Steuerung von ABB bietet die Möglichkeit, eine Bewegung des Werkzeuges durch externe Sensoren dynamisch zu beeinflussen. Das heisst, ein oder zwei Freiheitsgrade werden nicht statisch geplant, sondern können während der Bewegung des Roboters in Funktion von externen Sensorwerten dynamisch angepasst werden.
Bei einer ABB Steuerung wird diese Funktionalität **Externally Guided Motion** genannt, und dazugehörende RAPID Funktionen mit dem Präfix EGM* bezeichnet.

Ein externer Sensor wird für diese Funktionalität entweder an einen analogen Eingang der OmniCore Robotersteuerung angeschlossen, oder an eine Ethernet Schnittstelle, wobei in diesem Fall ein von ABB definiertes Protokoll basierend auf UDP/IP implementiert werden muss.



1 **Motion control calls EGM.**
2 **The measurement data (y- and z-values) are read from the signals or fetched from the sensor at multiples of about 48 ms.**
3 **EGM calculates the position correction and writes it to motion control. If the UdpUc protocol is used, feedback is sent to the sensor.**

Bild 4.1: Signalfluss für Externally Guided Motion

Damit ein externer Sensor via UDP/IP verwendet werden kann, muss er in der Robotersteuerung noch konfiguriert werden. Dazu kann im RobotStudio das Feld "Konfiguration" aufgeklappt werden.

Zuerst muss im Fenster "Communication" ein neues "UDP Unicast Device" kreiert und wie in der Abbildung 4.2 gezeigt konfiguriert werden. Wichtig ist hier auch die Konfiguration des Namens "egmSensor", weil dieser Name als Referenz in RAPID verwendet wird.

Danach muss im Fenster "External Motion Interface Data" ein EGM Interface konfiguriert werden, wie in der Abbildung 4.3 gezeigt.

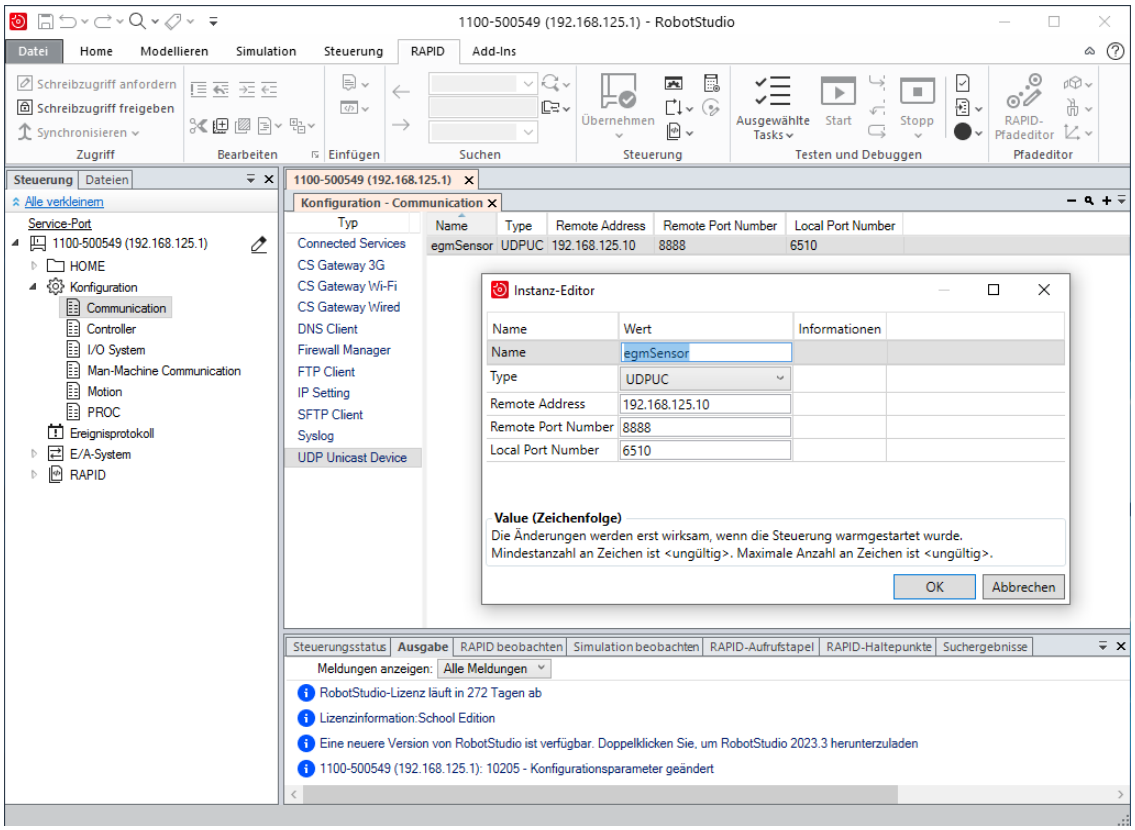


Bild 4.2: Konfiguration eines EGM Sensors

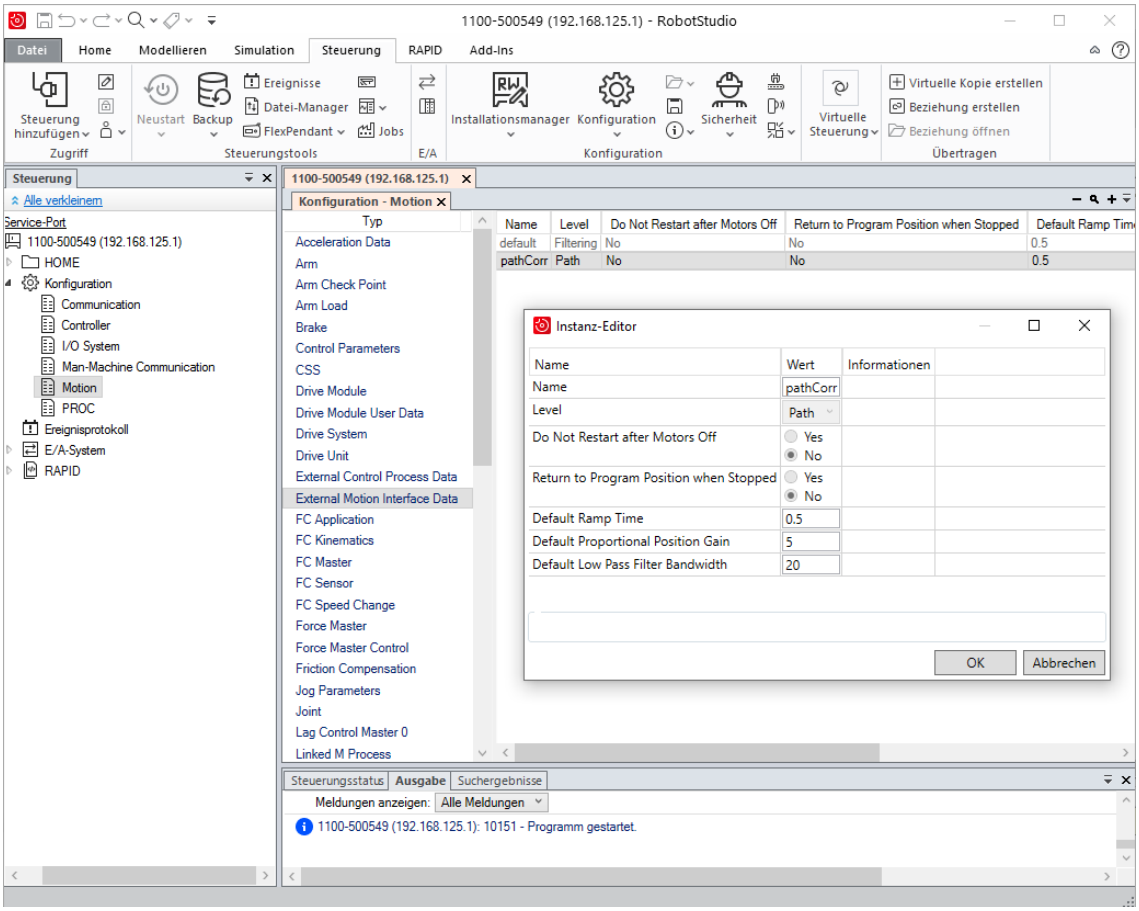


Bild 4.3: Konfiguration der External Motion Interface Daten

Das mit diesem Praktikum gegebene RAPID Programm 'MainModule.mod' beinhaltet bereits die Unterstützung eines Fahrbefehles mit EGM, nämlich **`EGMMoveL`**, also eine lineare Bewegung in Kartesischen Koordinaten, die dynamisch korrigiert werden kann.
In der Python Klasse 'OmniCore.py' gibt es eine dazu passende Funktion **`egmMoveL`**, die wie im folgenden Skript aufgerufen werden kann:

```
>>> from OmniCore import OmniCore
>>> robot = OmniCore("192.168.125.1")
>>> robot.moveJ([350,120,-30], [0,0,1,0], [0,90,0,0,5,0], 100, 0)
>>>
>>> while True:
>>>     robot.egmMoveL([515,-90,42], [[0,0,1],[1,0,0],[0,1,0]],
>>>                                  [0,90,0,0,5,0], 20, 0)
>>>     robot.egmMoveL([515,120,42], [[0,0,1],[1,0,0],[0,1,0]],
>>>                                  [0,90,0,0,5,0], 20, 0)
```

Die Parameter, die in dieser Funktion übergeben werden sind identisch wie bei den Funktionen **`moveL`** oder **`moveJ`**. Die Funktion **`egmMoveL`** korrigiert aber die Position in Richtung des Distanzsensors während der Bewegung.

Der Distanzsensor ist so konfiguriert, dass er einen Messbereich von 100 mm unterstützt. Die EGM Funktion versucht dann, die Distanz des Sensors zu einer Oberfläche in der Mitte dieses Messbereiches zu halten. Programmieren Sie also eine Bewegung dieses Sensors entlang eines Objektes. Verschieben Sie dann das Objekt und beobachten Sie, wie der Roboter die Bewegung korrigiert.