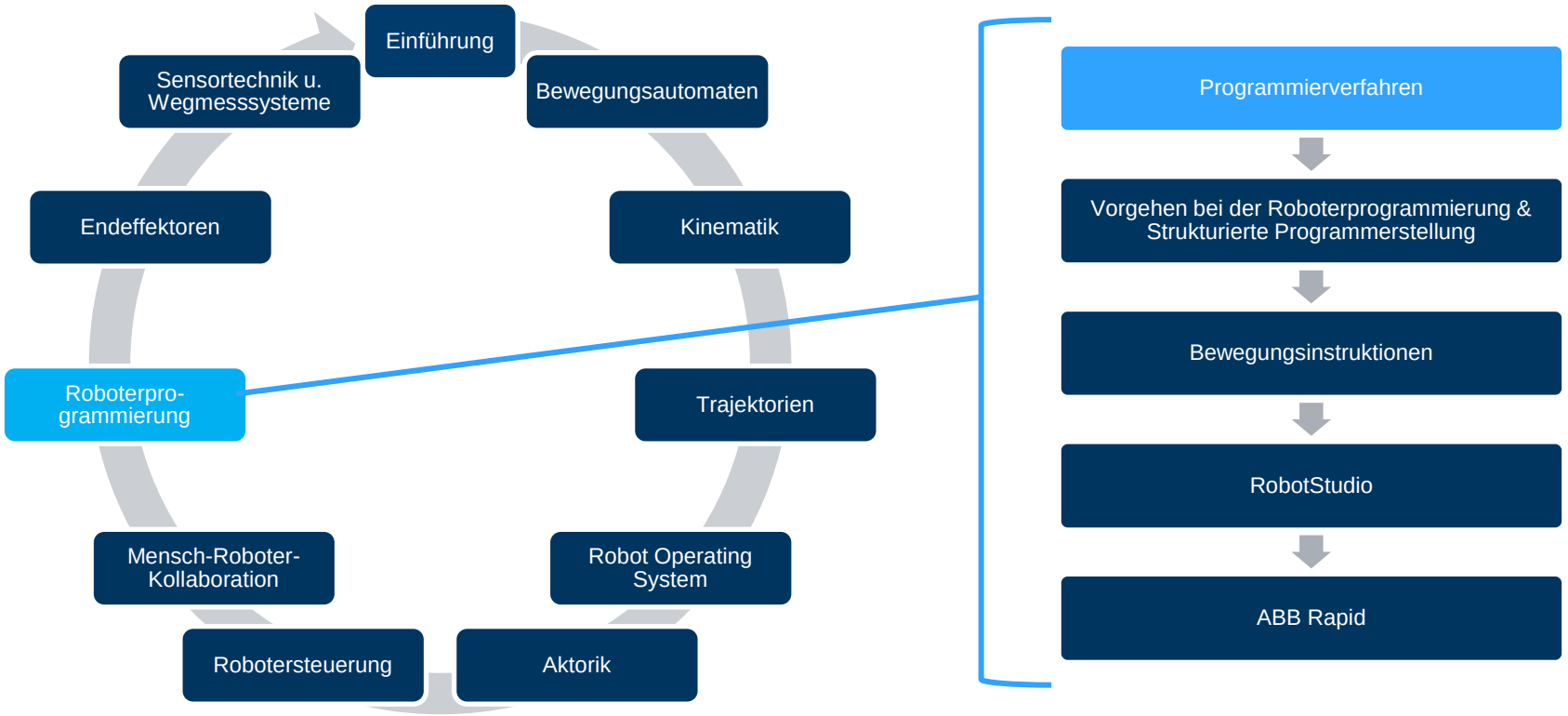
Technische Hochschule Georg Agricola

ROBOTIK
ROBOTERPROGRAMMIERUNG

17.12.2025 Bochum

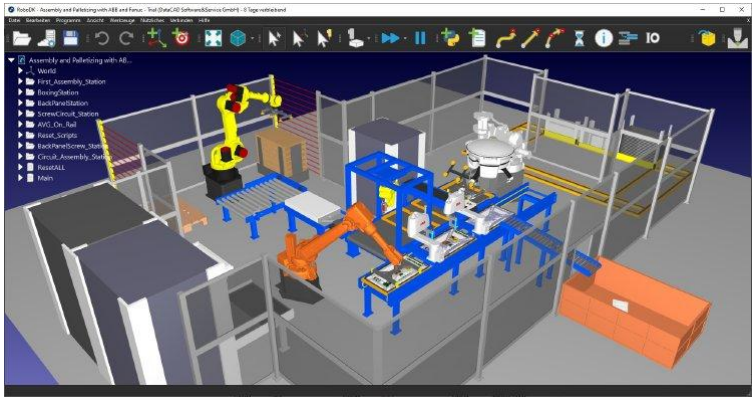
Dr.-Ing. Lars N. Josler

VORLESUNGSINHALTE



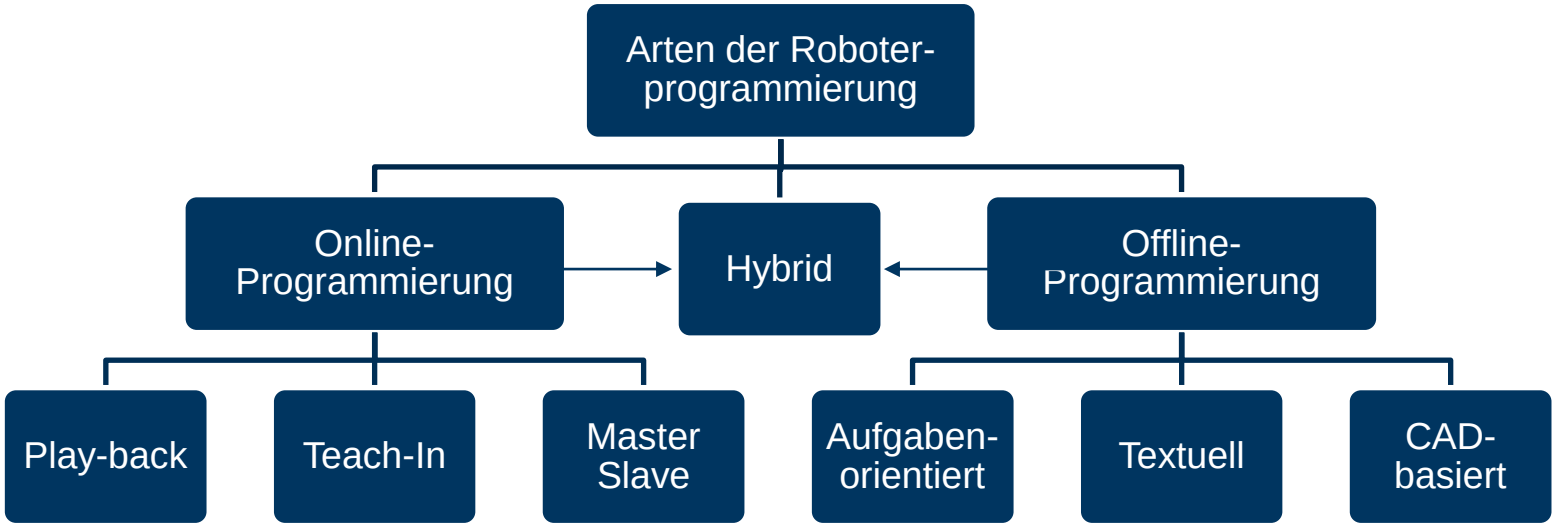
GRUNDLAGEN

- **Möglichkeiten zur Roboterprogrammierung**
  - Offline-Programmierung
  - Online-Programmierung



Quelle: ABB, KUKA

PROGRAMMIERARTEN



ONLINE-PROGRAMMIERUNG

- **Programmierung erfolgt direkt am Einsatzort des Roboters**
- **Manuelle Positionierung des Roboters (mit Hilfsmitteln)**
- **„Programmierung durch Vormachen“**

- **Vorteile**
  - Keine Kenntnisse einer Programmiersprache notwendig

- **Nachteile**
  - Stillstand des Roboters während der Programmierung
  - Hinderung der Produktion

ONLINE-PROGRAMMIERUNG - TEACH-IN

- **Manuelles Führen des Effektors entlang einer Bahn**
- **Führung erfolgt über ein Handbediengerät (Handprogrammiergerät)**

Epson

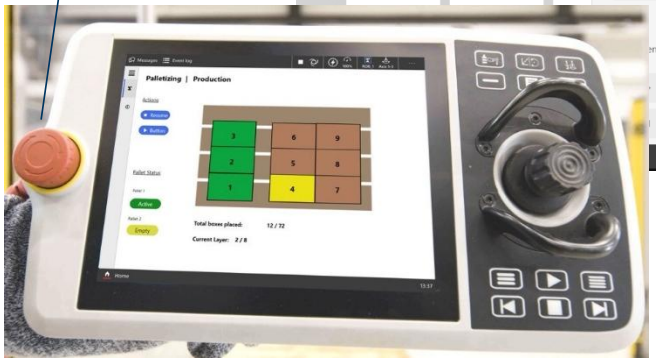
PANELMATE

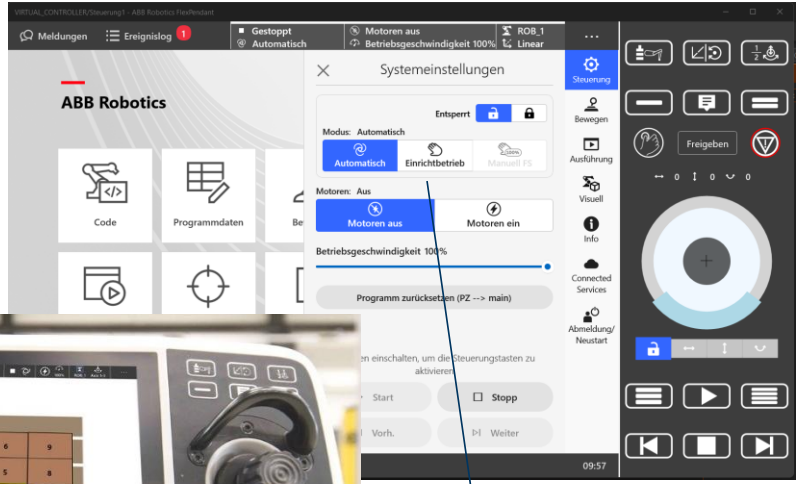
FANUC

KUKA

ABB

TEACH-IN PROGRAMMIERUNG (TPE)





Auswahl
Betriebsart

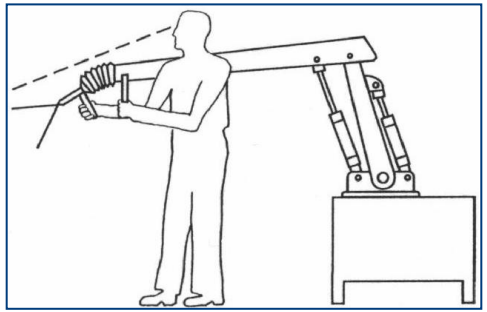
ONLINE-PROGRAMMIERUNG - PLAYBACK

- Manuelles Führen des Effektors entlang einer Bahn
- Speicherung der Gelenkwerte in definierten Zeitintervallen durch die Steuerung
- Führung erfolgt durch einen Griff nahe des Effektors

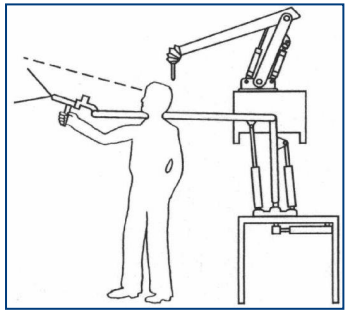
Quelle: KUKA ready2_pilot | KUKA

ONLINE-PROGRAMMIERUNG - MASTER-SLAVE

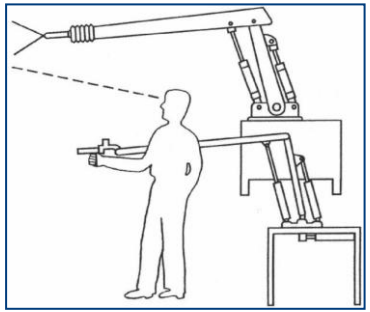
- **Verwandt zu Playback**
- **Bewegungsvorgabe durch einen Bedienarm (Master-Arm)**
- **Ausführung der Bewegung über den Arbeitsarm (Slave-Arm/Industrieroboter)**



Verfahren des Roboters (Playback)



Verfahren des kinematischen Modells



Verfahren des IR über ein Kinematisches Modell

[Quelle: J. Wolfgang Ziegler, FHD, Fachhochschule Düsseldorf]

BEISPIEL MASTER-SLAVE: VINCI SP

What is the da Vinci SP robotic system? - YouTube

VERGLEICH ONLINE-PROGRAMMIERUNG

Kriterien	Teach-In	Playback	Master-Slave
Genauigkeit der Bewegungsfolge	Hoch	Gering	Gering bis Mittel (abhängig vom Eingabegerät)
Harmonischer Bewegungsablauf	Schwierig	Einfach	Sehr einfach (Filterung der Bewegung)
Korrekturmgl. der Bewegung	Sehr gut	Schwierig	Gut (durch Softwareseitige Korrektur)
Optimierbarkeit	Möglich	Nicht möglich	Beschränkt Möglich

OFFLINE-PROGRAMMIERUNG

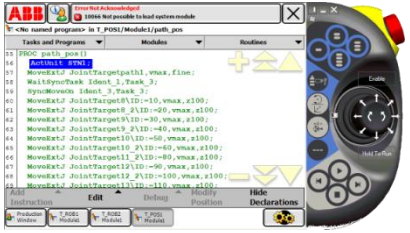
- **Programmerstellung erfolgt computergestützt**
- **Keine Verwendung des Industrieroboters**
- **Roboterbewegungen werden am PC getestet**
- **Übertragung des Programms auf den Roboter**

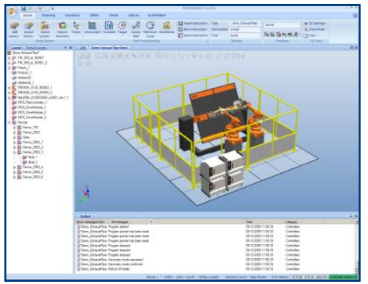
- **Vorteile**
  - Kein Stillstand des Roboters während der Programmierung
  - Keine Hinderung der Produktion

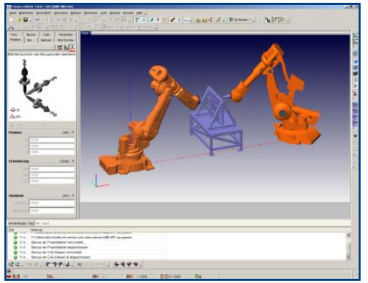
- **Nachteile**
  - Evt. Kenntnisse einer Programmiersprache notwendig

OFFLINE-PROGRAMMIERUNG – ÜBERSICHT

- **Textuell**
  - Programmierung auf dem virtuellen Handbediengerät
  - Programmierung in der Simulationsumgebung
- **CAD-basiert**
  - Erstellung von Bahnen und Roboterprogrammen auf definierten Geometrien
- **Aufgabenorientiert**
  - Automatische Programmgenerierung

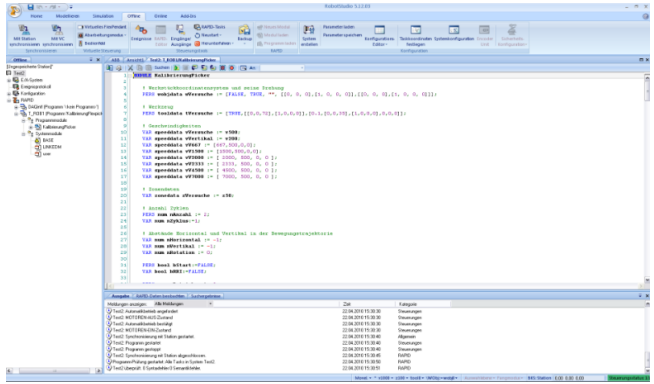






TEXTUELL

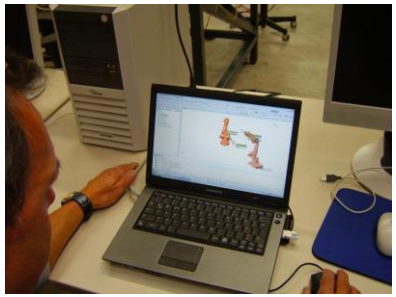
- Einsatz von Programmiersprachen zum Vorgeben der Bewegungsfolge
- Programmerstellung durch textuelle Eingabe von Befehlen
- Unterstützung der Anwender bei der Programmierung durch Roboterhersteller-Systeme (ProgramMaker (ABB), OfficeLite (KUKA))
  - Vermeidung von Schreib- und Syntaxfehlern



- Zusätzliche Möglichkeit:
  - grafische Darstellung des Teach Panels (ABB Virtual Teach Pendant, REIS RobOffice) als Applikation -> Anwählen aller Funktionen, die auch am realen Roboter vorhanden sind

CAD-BASIERT

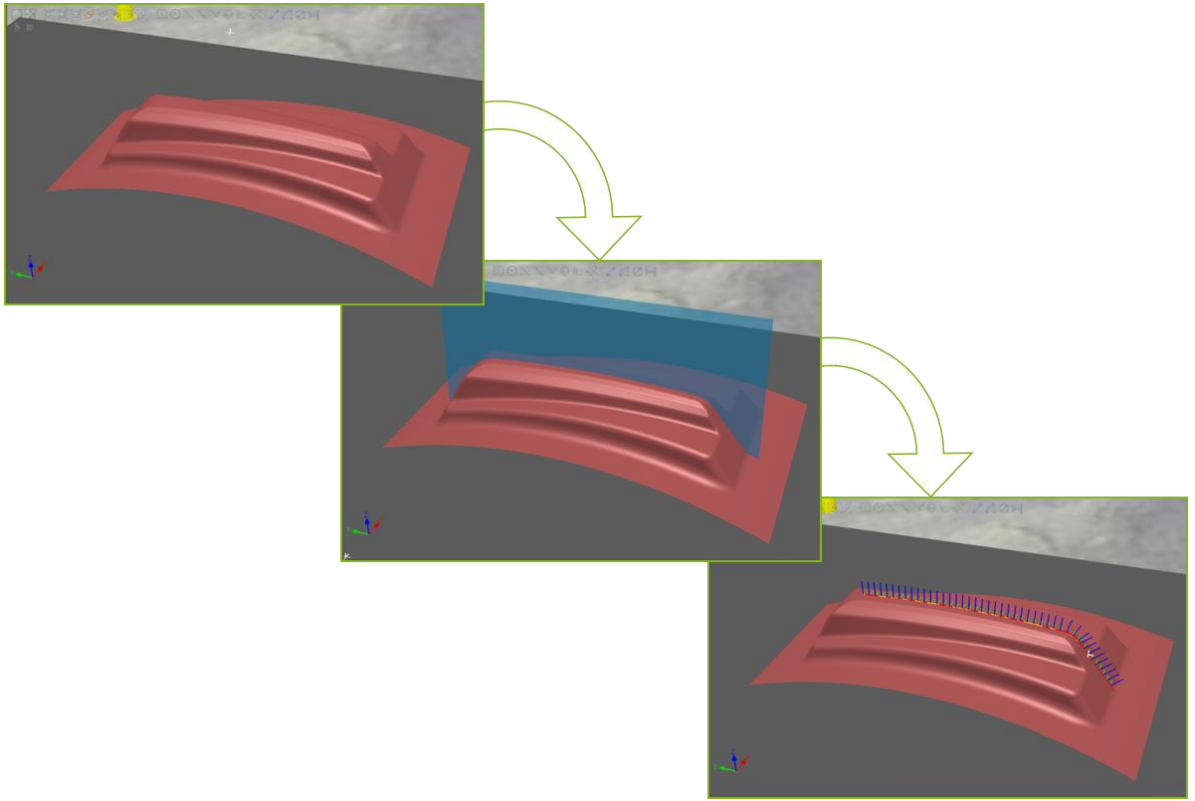
- Nachbildung der kompletten Arbeitszelle in einer CAD-Umgebung
- Anfahren und Abspeichern der Zielstellungen in dieser virtuellen Umgebung oder
- Ausführung eines Roboterprogramms zur Kontrolle, ob Aufgabe in der Simulationsumgebung bewältigt wird
- -> Offline-Programmiersysteme (OLP-Systeme)
  - Universelle Simulationssysteme
  - Simulationssysteme



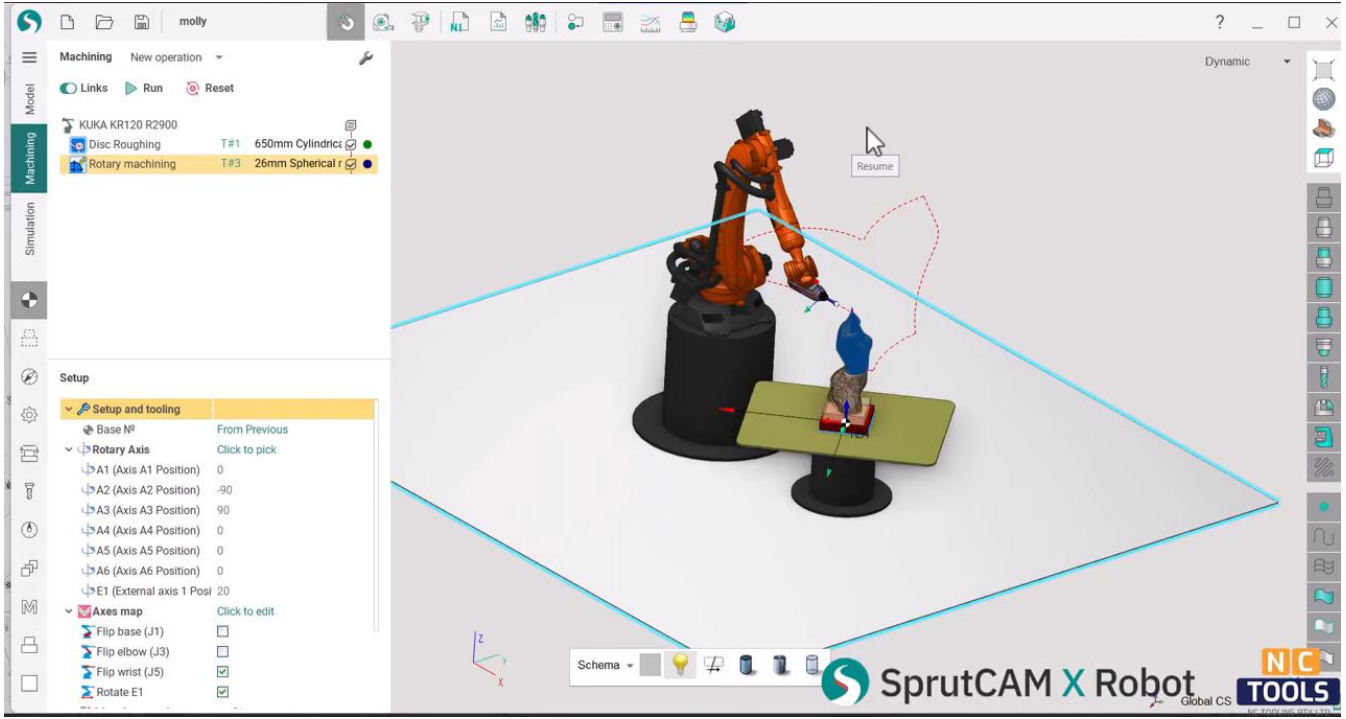
- **Vorteile:**
- Zeit- und Rentabilitätsgewinn
- Höhere Detailqualität
- Vereinfachte Planung

- **Nachteile:**
- Kosten für die grafischen Offline-Programmiersysteme
- Herstellerabhängige Programmiersysteme

OFFLINE-PROGRAMMIERUNG – CAD-BASIERT



BEISPIEL: CAD-BASIERT PROGRAMMIERUNG



Quelle: SprutCAM Robot Offline programming

OFFLINE-PROGRAMMIERUNG – VERGLEICH

	Textuelle Programmierung	CAD-Programmierung
Eingabe von Positionen	nur mittels Teach-in	CAD-basiert
Genauigkeit der Positionen	sehr genau durch gute Wiederholgenauigkeit	abhängig vom Modell und wg. eingeschränkter absoluter Genauigkeit
Zeitaufwand für die Positionseingabe	hoch	niedrig
Zeitaufwand für die Programmstruktur	niedrig	niedrig
Kollisionsüberwachung	an der Anlage	während der Programmierung
Simulation	an der Anlage	möglich

OFFLINE-PROGRAMMIERUNG -EINSATZBEREICHE

- **Planungsphase**
  - Test verschiedener Zellenlayouts mit minimalem Kostenaufwand und Klärung, ob das gefundene Layout den Anforderungen entspricht. Anforderungen können z. B. sein:
    - Erreichbarkeit der Komponenten untereinander
    - Platzbedarf
    - Zykluszeiten
- **Programmierphase**
  - Graphisch gestützte Programmierung
- **Entwicklung von Sondergeräten**
  - Testen von Handhabungssystemen im Entwicklungsstadium

PROGRAMMIERSPRACHEN

- **Überwiegend steuerungsspezifische Sprachen**
- **Programmierung auf der Anwendungsebene und der Steuerungsebene**
- **Die meisten Roboterprogrammiersprachen gehören zum prozeduralen Programmierparadigma**
- **Herstellerabhängige Programmiersprachen**
  - KRL (KUKA Robot Language)
  - KUKA -Programmiersprache
  - RAPID (Robot Application Program Interactive Dialog)
  - ABB-Programmiersprache
  - …
- **Standardisierung der Roboterprogrammiersprache**
  - IRL (Industrial Robot Language), DIN 66312

PROGRAMMIERUNG VON INDUSTRIEROBOTERN OFFLINE

Hersteller	Programmiersprache	Bedienfläche am Touchpanel
Omron	V+	
ABB	RAPID	
Fanuc	Karel	TPE
Kuka	KRL	
Yaskawa Motoman	Inform	
Stäubli	VAL3	
Universal Robots (UR)	script	Polyscope
Epson	SPEL+	
Denso	Pac (RC7) & PacScript (RC8)	
Mitsubishi	MELFA-Basic[1]	

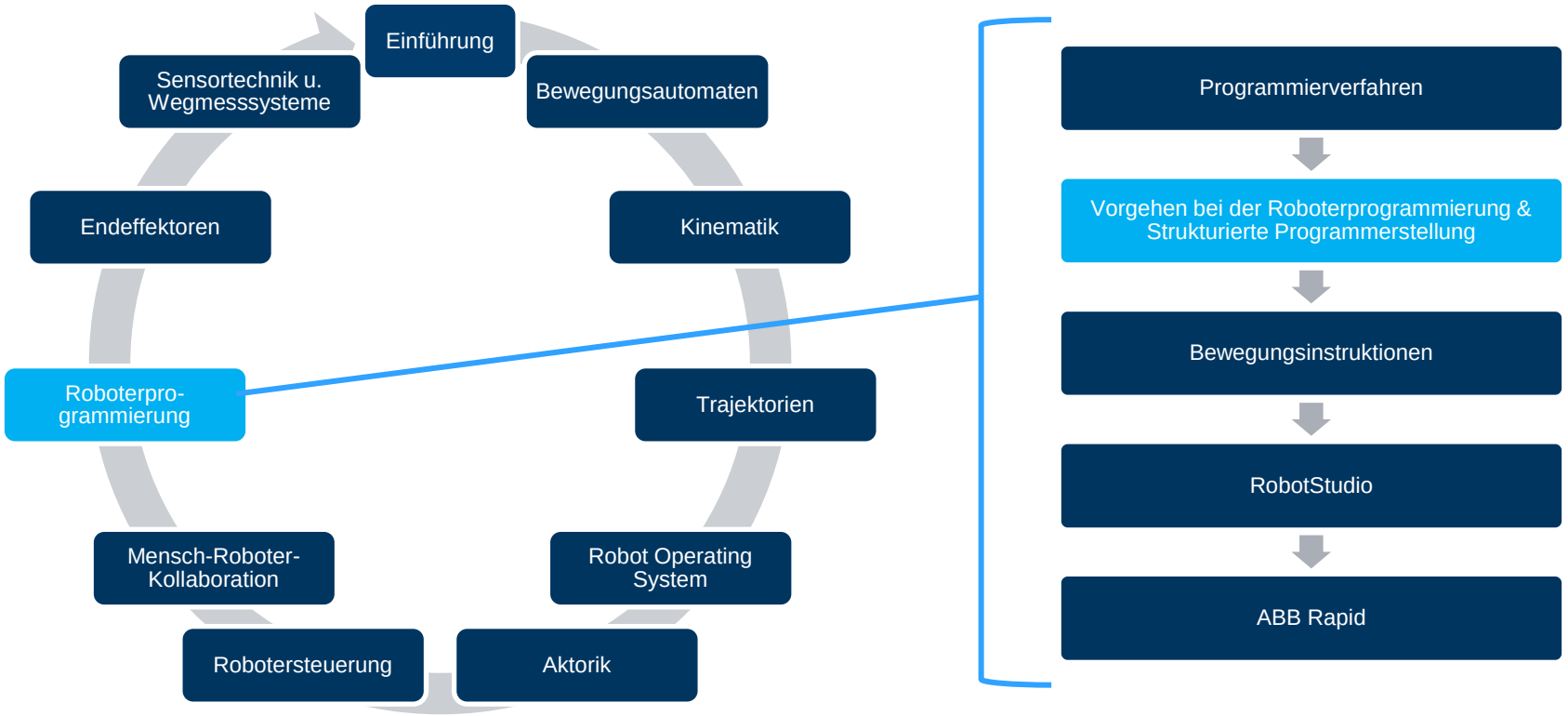
GRUNDELEMENTE VON ROBOTERPROGRAMMIERSPRACHEN

- **Befehle zur ...**
  - Steuerung des Programmablaufes (IF, GOTO, FOR, WHILE , …)
  - Bewegung (PTP, CP-linear, CP-zirkular , …)
  - Bewegungseinstellung (Geschwindigkeit, Werkzeugdaten, …)
  - Digitale und analoge Signale (HIGH, LOW, Ein- u. Ausgabe)
  - Benutzerführung und Kommunikation
  - Mathematik (Grundrechenarten, spezielle Operationen: Matrizen, Vektoren, Transformationen, etc.)
- **Datentypen**
  - Allgemeine Datentypen (Logisch, Ganzzahl, Fließkommazahl, Zeichenkette, etc.)
  - Spezielle Datentypen (Vektor, Koordinatensysteme, etc.)
  - Persistente Variablen
  - Variablenwerte werden zwischen Programmstarts gespeichert
  - Notwendig für die Produktion (Neustart auf vorherigem Zustand)
- **Datenlisten / Positionslisten**

VARIABLEN

- **Variable:**
  - Speicherplatz mit einem Namen, unter dem Daten gespeichert werden. Diese Daten können sich im Programm verändern.
  - Daten sind z.B. Informationen aus der technischen Anlage, die mit der Robotersteuerung bedient und gesteuert wird.
  - Es muss für jede Variable ein bestimmter Datentyp ausgewählt werden, der den zulässigen Inhalt festlegt.
- **Deklaration:**
  - Die Variable wird global im Oberprogramm (z.B. Modul) oder lokal in einem Baustein (z.B. Prozedur) angelegt.
  - Bei der Variablendeklaration wird der Datentyp und der Name festgelegt.
- **Initialisierung:**
  - Am Anfang hat die Variable keinen definierten Wert, dieser muss vor der ersten Verwendung zugewiesen werden.

VORLESUNGSINHALTE



VORGEHENSWEISE BEI DER ROBOTERPROGRAMMIERUNG

- **Erst entwerfen, dann programmieren**
- **Entscheidung über Befehle, Daten, usw.**
- **Algorithmus vor der Programmierung überlegen**

- **Hilfsmittel:**
  - Pseudocode
  - Flussdiagramm
  - UML

PSEUDOCODE

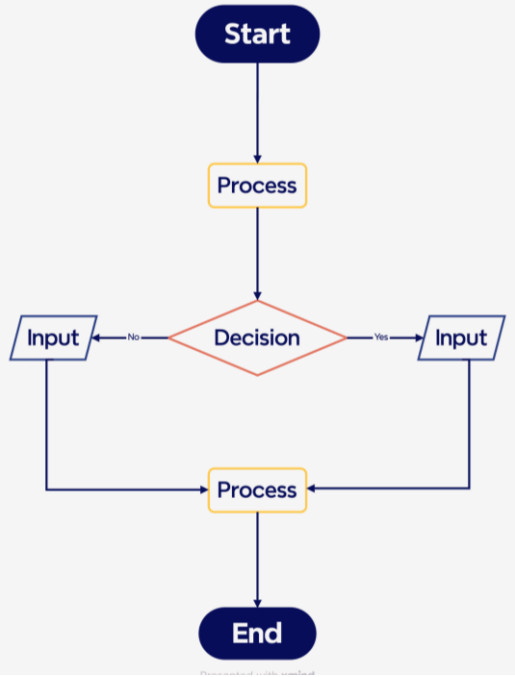
- Vereinfachte Art von Programmiersprache
- Verbale Beschreibung der Aktionen
- Kein Bezug zu der Syntax einer bestimmten Programmiersprache
- Darstellung eines Sachverhaltes

Pseudocode (Beispiel)	RAPID Code (Beispiel)

Konstanten & Definitionen
Home: Definierte Ruheposition
Pick_Point: Greifpunkt
Hauptprogramm: Main
Aktion: Fahre linear (schnell) zur Überfahrtsposition (Pick_Point + 100mm in Z)
WENN Objekt erkannt **DANN**:
Aktion: Greifposition anfahren (linear, fein)
Aktion: Greifer schließen (Simuliert)
SONST:
Ausgabe: "Kein Objekt gefunden, fahre zurück."
ENDE WENN
Aktion: Fahre PTP zur Ruheposition (Home)
	CONST robtarget Home := [...] \WObj:=wobj0;
	CONST robtarget Pick_Point := [...] \WObj:=wobj0;
	PROC Main()
	MoveL Offs(Pick_Point, 0, 0, 100), v1000, z10, tGripper;
	IF diSensor = 1 THEN
	MoveL Pick_Point, v100, z0, tGripper;
	Reset doGripper;
	ELSE
	TPWrite "Kein Objekt gefunden, fahre zurück.";
	ENDIF
	MoveJ Home, v1000, z10, tGripper;
	ENDPROC

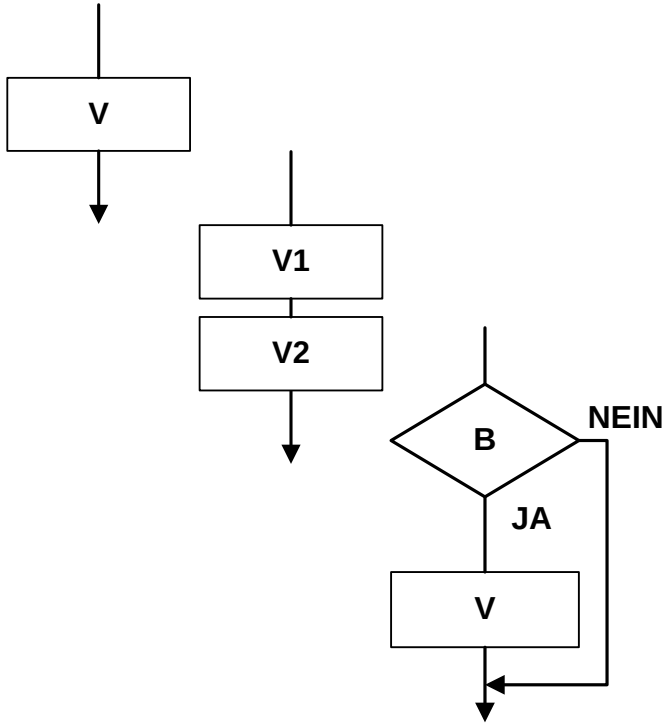
FLUSSDIAGRAMM

- **Standardisierte Darstellungsform für Programm- und Prozesslogik.**
- **Weltweit verständliche Symbole für Start, Aktionen, Entscheidungen und Übergänge.**
- **Besonders geeignet für die Planung, Dokumentation und Kommunikation von Steuerungsabläufen.**
- **Typische Einsatzbereiche:**
  - Programmfluss in Steuerungen (z. B. ROS2-Nodes, SPS-Programme).
  - Fehlerbehandlung und Entscheidungslogik (Sensorwerte → Aktorreaktionen).
  - Visualisierung von Sequenzen und parallelen Abläufen.



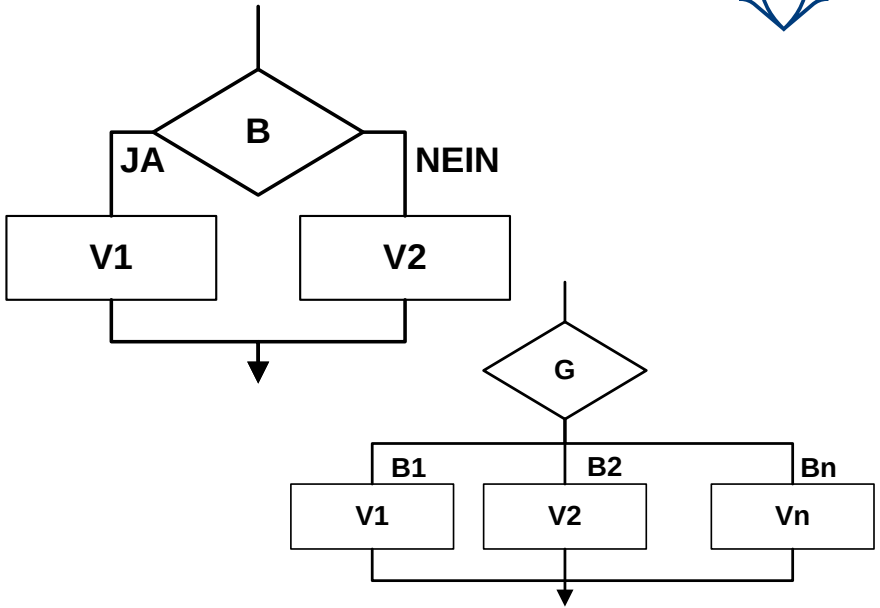
FLUSSDIAGRAMM

- **Programmstruktur - Verarbeitung**
  - Verarbeitung besteht aus einem Block (Programmaktion), der einmal ausgeführt wird
- **Programmstruktur - Folge**
  - zwei oder mehrere Verarbeitungselemente, die jeweils einmal ausgeführt werden
- **Programmstruktur – Auswahl**
  - Bedingte Verarbeitung: Bedingung B bestimmt, ob Verarbeitungsschritt ausgeführt wird oder nicht



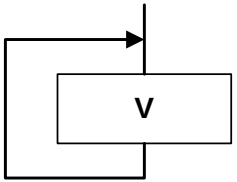
FLUSSDIAGRAMM

- **Programmstruktur – Auswahl**
  - Einfache Alternative: der Bedingung B gibt an, welcher Verarbeitungsschritt ausgeführt wird
  - Mehrfache Alternative: Für jeden Verarbeitungsschritt besteht eine Bedingung; je nachdem, welche Bedingung erfüllt ist, wird der zugehörige Bearbeitungsschritt ausgeführt

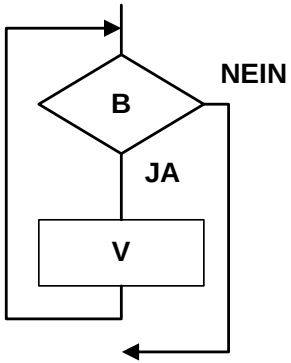


FLUSSDIAGRAMM

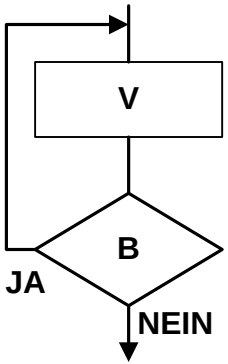
- **Programmstruktur - Wiederholung**
  - Verarbeitungsschritt wird endlos ausgeführt



  - Die Bedingung B bestimmt, ob bzw. wie häufig der Verarbeitungsschritt augeführt wird; da die Prüfung vor der Bearbeitung erfolgt, ist ein Auslassen des Schrittes ebenfalls möglich

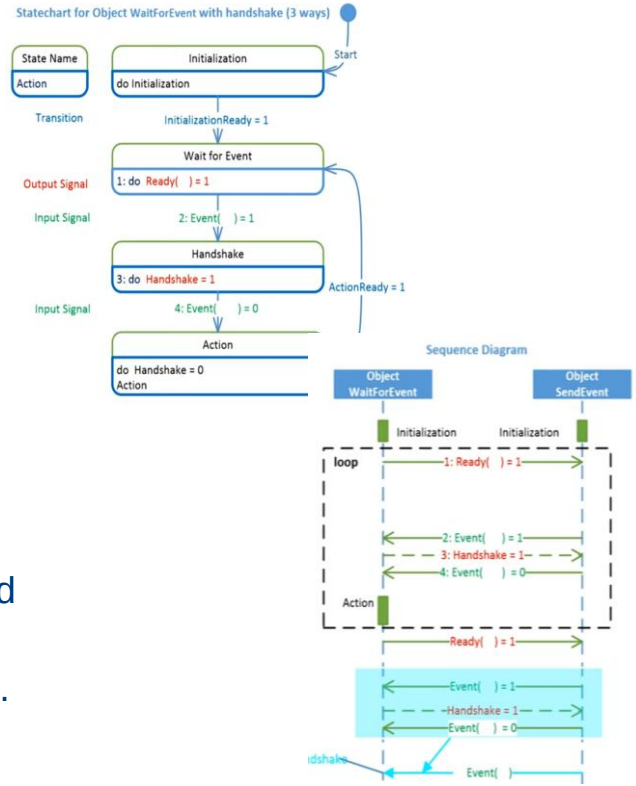


- **Programmstruktur - Wiederholung**
  - nach dem ersten Durchlauf des Verarbeitungsteils wird geprüft, ob bzw. wie oft dieser durchlaufen wird



UNIFIED MODELING LANGUAGE (UML)

- **Internationaler Standard zur Visualisierung von Systemarchitekturen und Abläufen.**
- **Verwendung weltweit verständlicher, eindeutiger Symbole.**
- **Besonders geeignet für komplexe Systeme mit parallelen Prozessen und Zuständen.**
- **Typische Einsatzbereiche:**
  - Aktivitätsdiagramme: Darstellung von Programmabläufen (Logik, Schleifen, Verzweigungen).
  - Zustandsdiagramme: Modellierung von Betriebszuständen und Zustandsübergängen.
  - Sequenzdiagramme: Kommunikation zwischen Objekten (z. B. Sensor → Controller → Aktor).

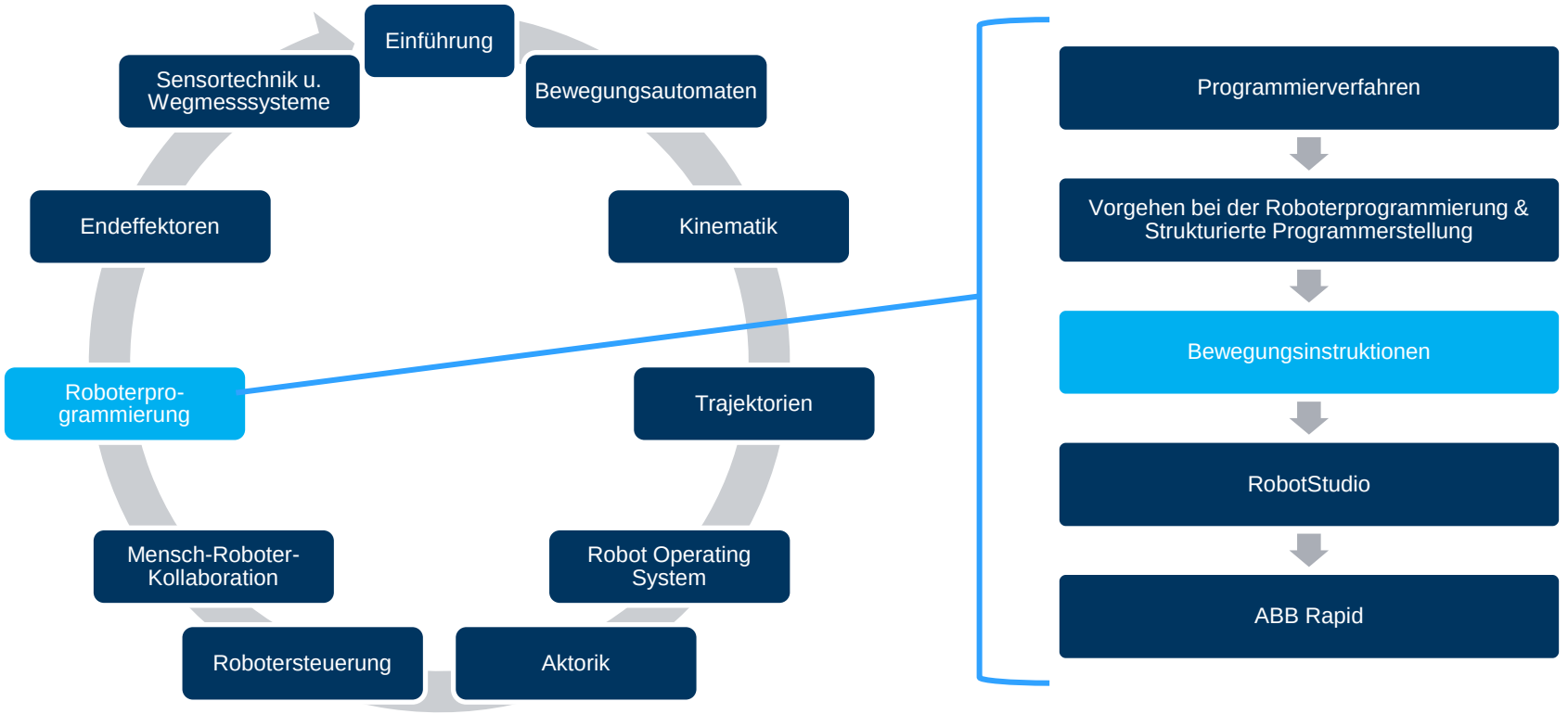


Quelle: FH-SWF

EINSATZBEREICHE DER HILFSMITTEL

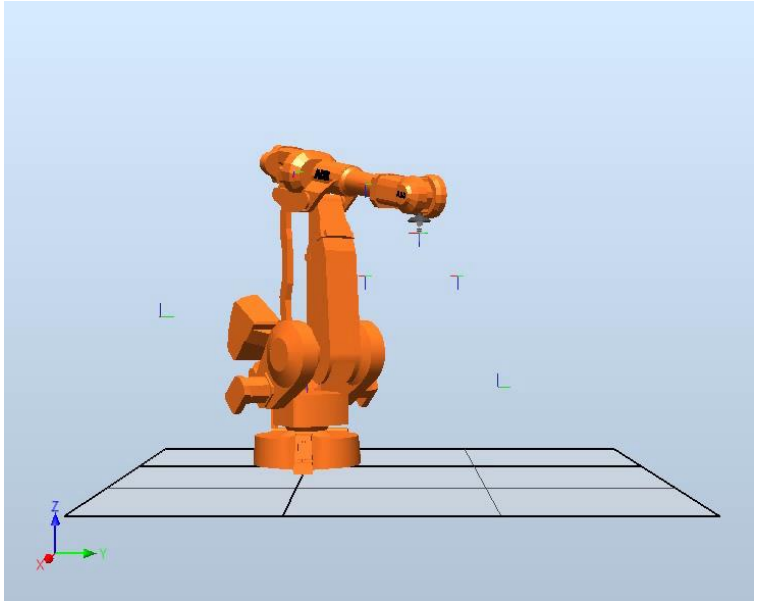
	Pseudocode	Flussdiagramm	UML
Eigenschaft	Textuelle Beschreibung von Algorithmen in natürlicher Sprache, nahe an Programmcode	Grafische Darstellung von Abläufen mit standardisierten Symbolen	Standardisierte Modellierungssprache mit verschiedenen Diagrammtypen
Einsatzbereich	Entwurf und Dokumentation von Algorithmen, schnelle Skizzen für Programmierer	Visualisierung von Prozess- und Programmabläufen, Entscheidungslogik	Modellierung komplexer Systeme, Architekturen und Kommunikation
Vorteile	- Einfach zu erstellen - Verständlich für Programmierer - Flexibel anpassbar	- Intuitiv und visuell - Weltweit verständliche Symbole - Gut für Kommunikation	- Internationaler Standard - Mächtig für komplexe Systeme - Unterstützt parallele Prozesse
Nachteile	- Wenig visuell - Für Laien schwerer verständlich - Keine Normung	- Bei komplexen Systemen schnell unübersichtlich - Weniger formalisiert	- Höhere Lernkurve - Teilweise komplexe Notation - Erfordert mehr Einarbeitung

VORLESUNGSINHALTE



BEISPIEL FÜR PSEUDOCODE

- **Beispiel 1**
- **Roboterbewegung von Target_10 nach Target_30 über Target_20**



PSEUDOCODE

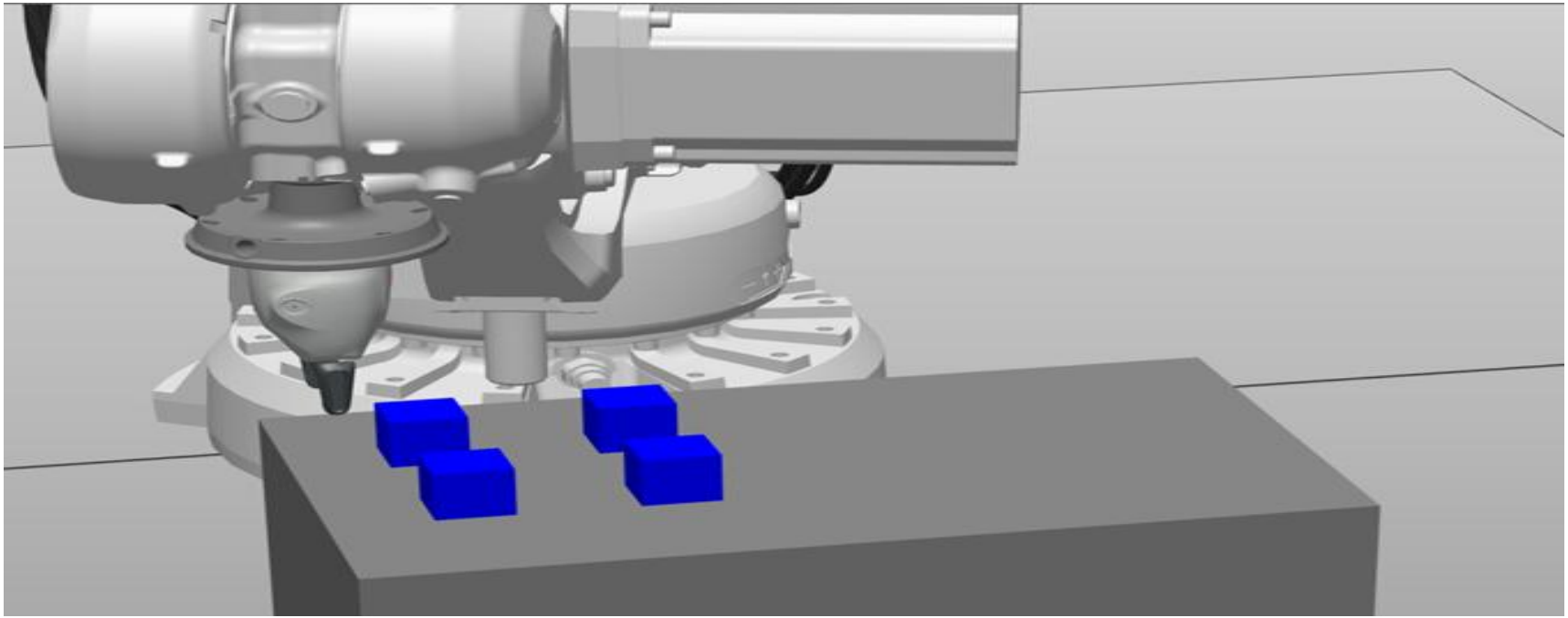
Konstanten & Definitionen

Home: Definierte Ruheposition des Roboters
Target_10: Startpunkt
Target_20: Zwischenpunkt der Bahn
Target_30: Endpunkt

Hauptprogramm: Main

Startposition: Fahre PTP (schnell, fein) zu Position: Target_10
Zwischenposition: Fahre linear (langsam, Überschleifen) zu Position: Punkt B
Endposition: Fahre linear (schnell, Überschleifen) zu Position: Punkt C
Homeposition: Fahre PTP (schnell, fein) zu Position: Punkt D

BEISPIEL PICK AND PLACE



PSEUDOCODE

Konstanten & Definitionen

Home: Definierte Ruheposition des Roboters
Target_10: Startpunkt
Target_20: Ablagepunkt

Hauptprogramm: Main

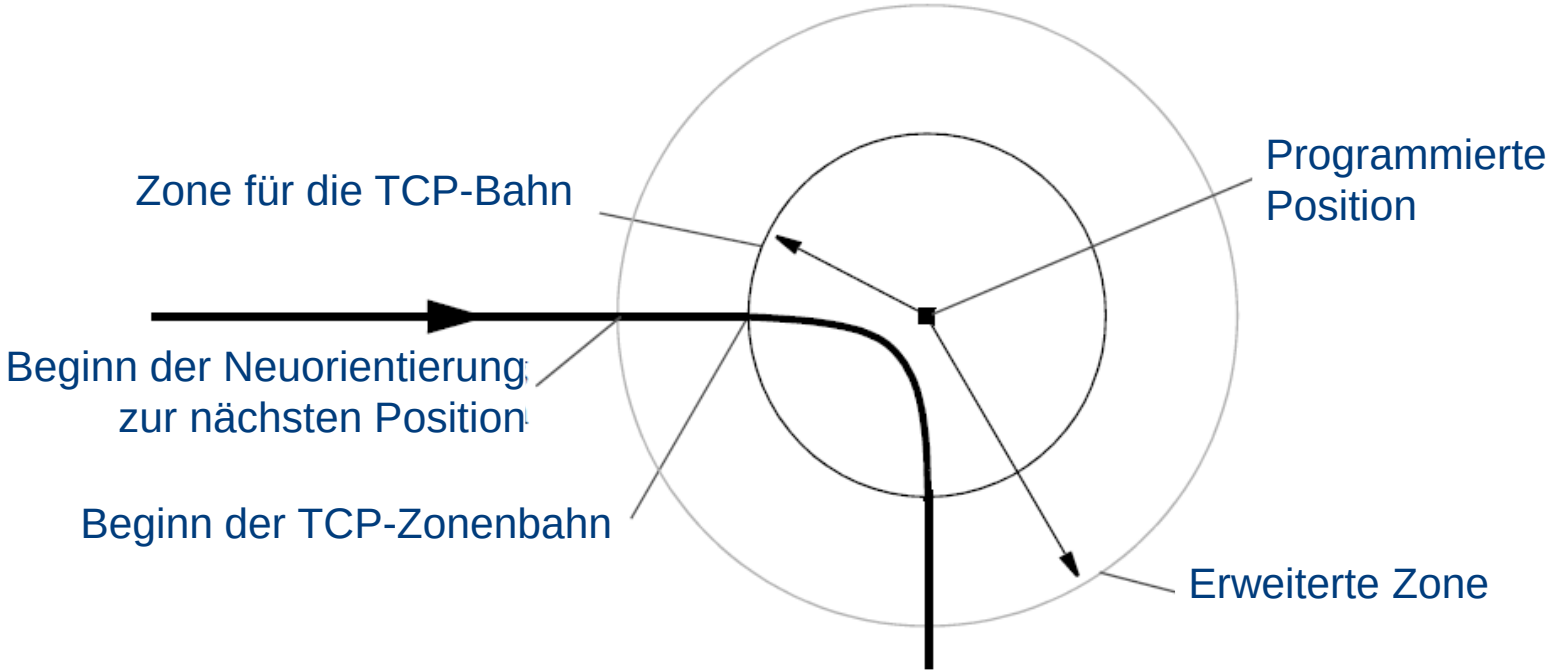
Homeposition: Fahre PTP (schnell, Überschleifen) zu Position: Home
Vorposition: Fahre PTP (schnell, Überschleifen) zu Position: Target_10 + *100mm in Z*
Greifposition: Fahre linear (langsam, fein) zu Position: Target_10
Aktion: Greifer schließen
Warte: 0,3 Sekunden
Anheben: Fahre linear (langsam, Überschleifen) zu Position: Target_10 + *100mm in Z*
Transport: Fahre linear (schnell, Überschleifen) zu Position: Target_20 + *100mm in Z*
Ablageposition: Fahre linear (langsam, fein) zu Position: Target_20
Aktion: Greifer öffnen
Warte: 0,2 Sekunden
Wegfahren: Fahre linear (langsam, Überschleifen) zu Position: Target_20 + *100mm in Z*
Homeposition: Fahre PTP (schnell, Überschleifen) zu Position: Home

BEWEGUNGSINSTRUKTIONEN

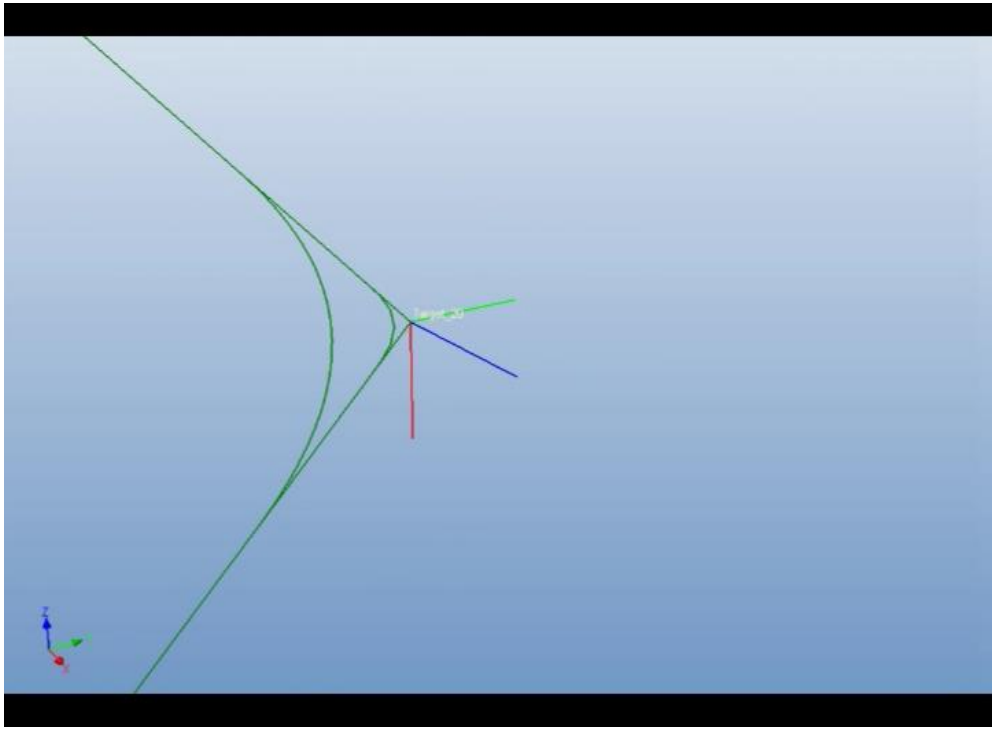
- **Mögliche Bewegungsinstruktionen**
  - Achsbewegungen
  - Linearbewegungen
  - Zirkularbewegungen

- **Notwendige Parameter**
  - Zielpose, Geschwindigkeit, Überschleifradius, Werkzeug

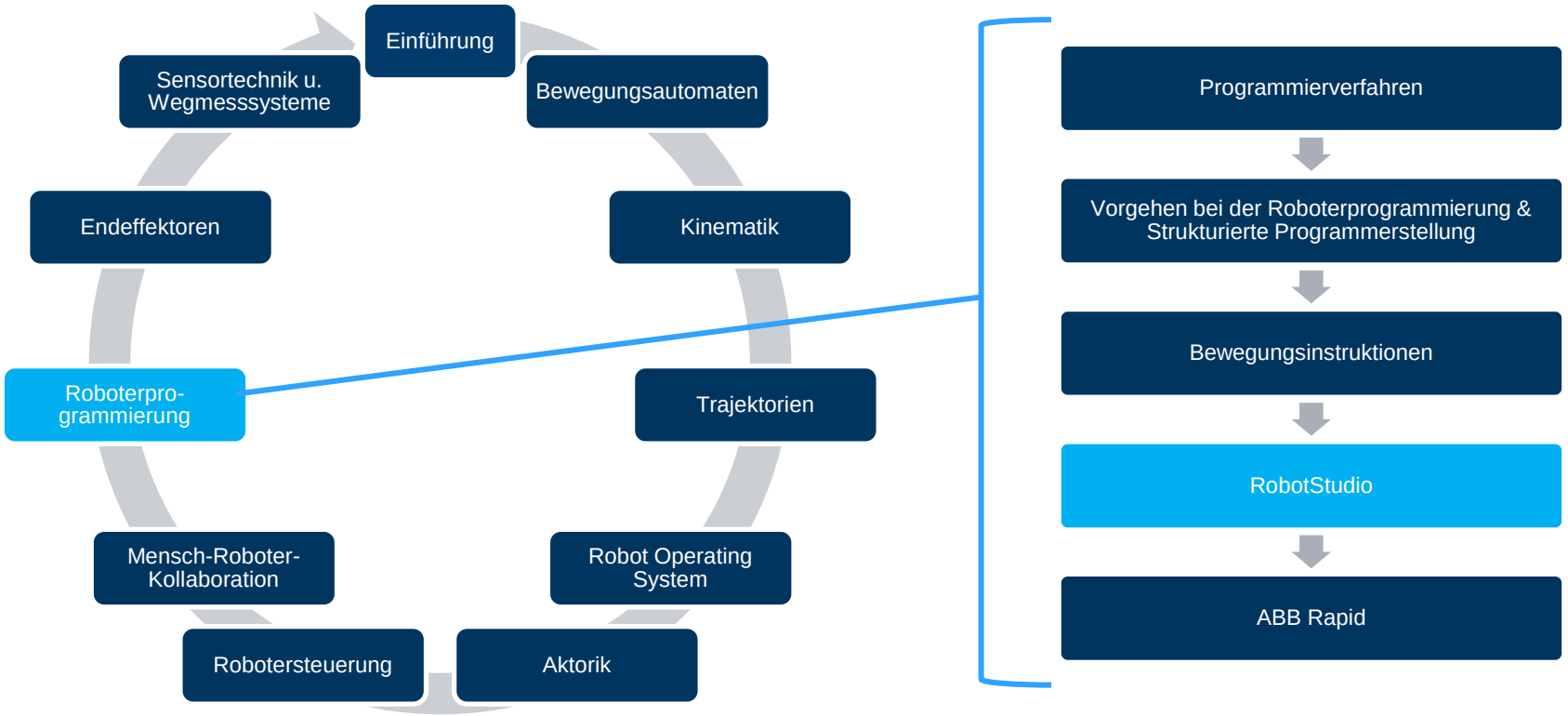
ÜBERSCHLEIFRADIUS



ÜBERSCHLEIFRADIEN - VERGLEICH



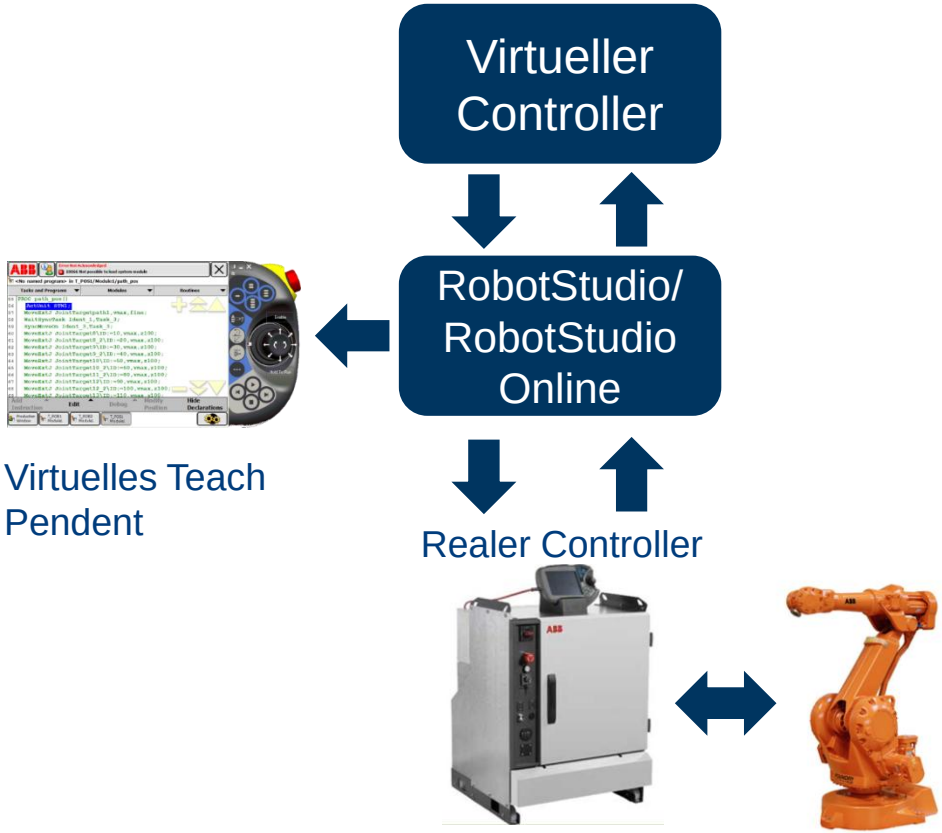
VORLESUNGSINHALTE



ROBOTSTUDIO

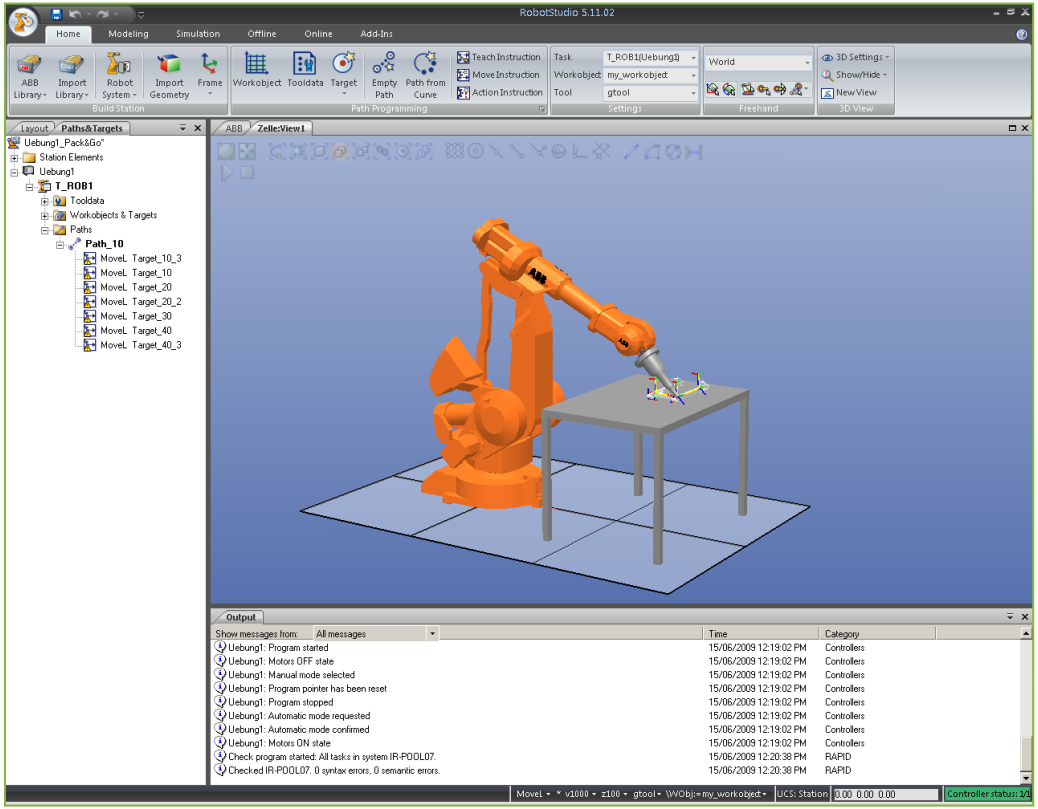
- **Produkt von ABB**
  - Herstellerabhängiges System
- **Unterteilung in verschiedene Betriebsmodi:**
  - “Zellenlayout”
  - Modellierung / Modell (Modeling)
  - Offline-Programmierung (des Roboters)
  - Online-Programmierung
    - Kommunikation mit realem Roboter
    - Download und Upload von Programmen

VERBINDUNG MIT REALEM CONTROLLER

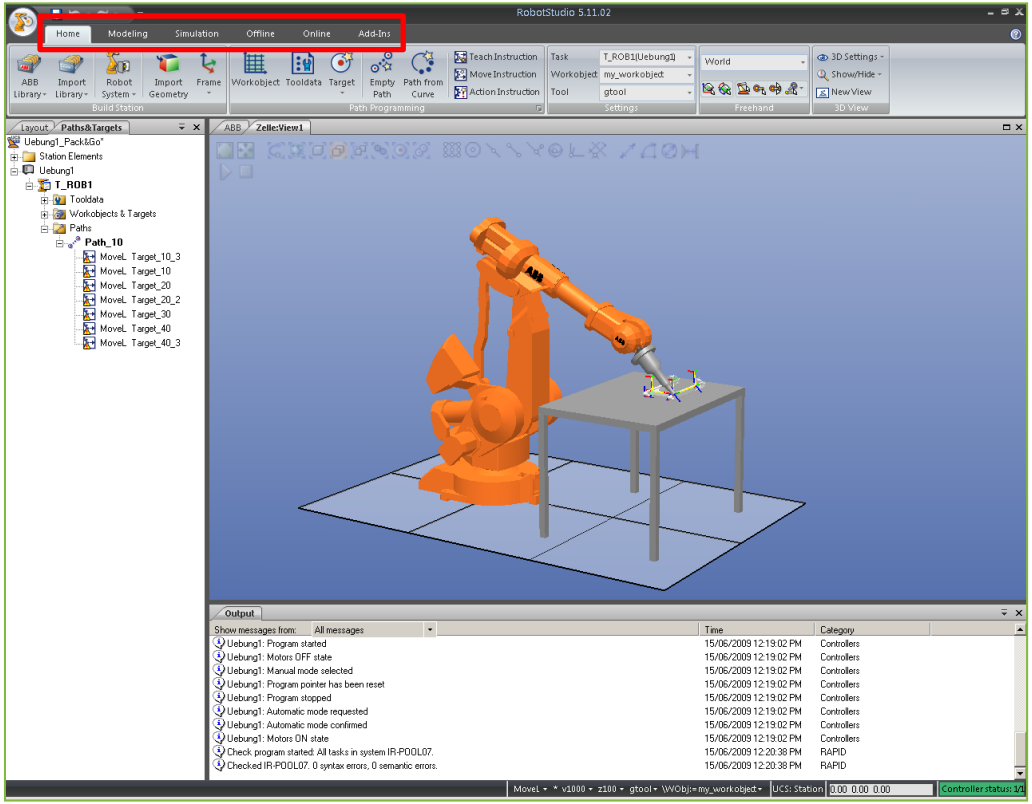


Quelle: ABB

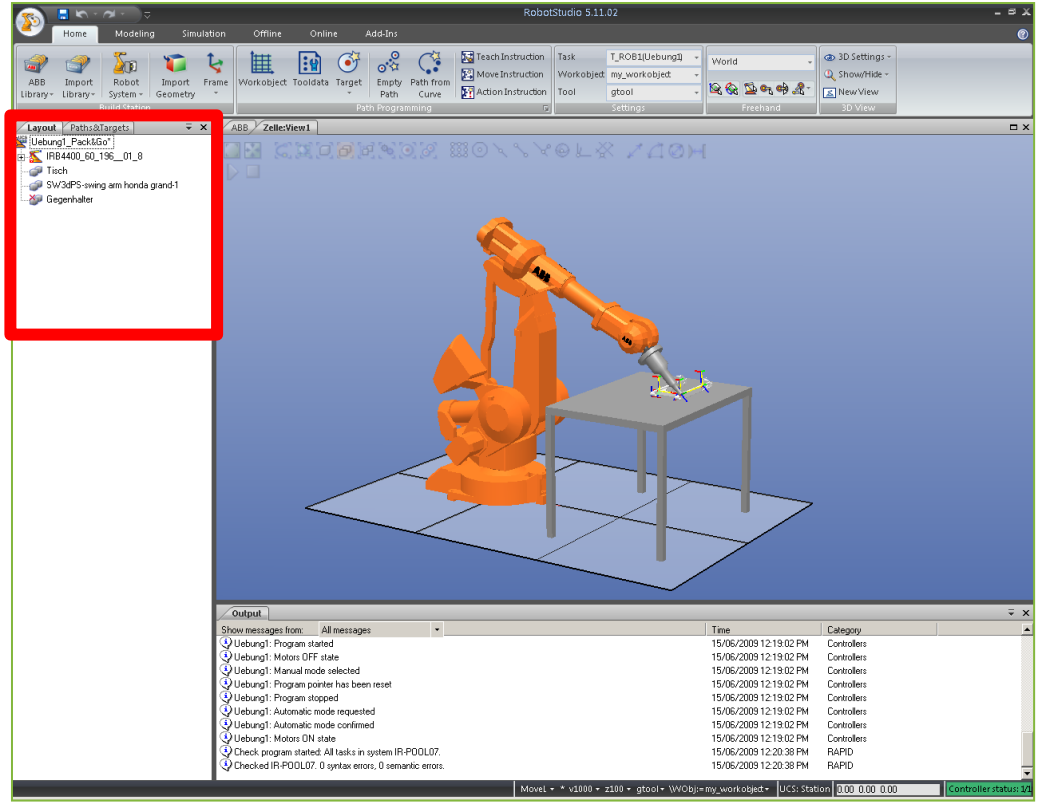
LAYOUTÜBERSICHT – HAUPTFENSTER



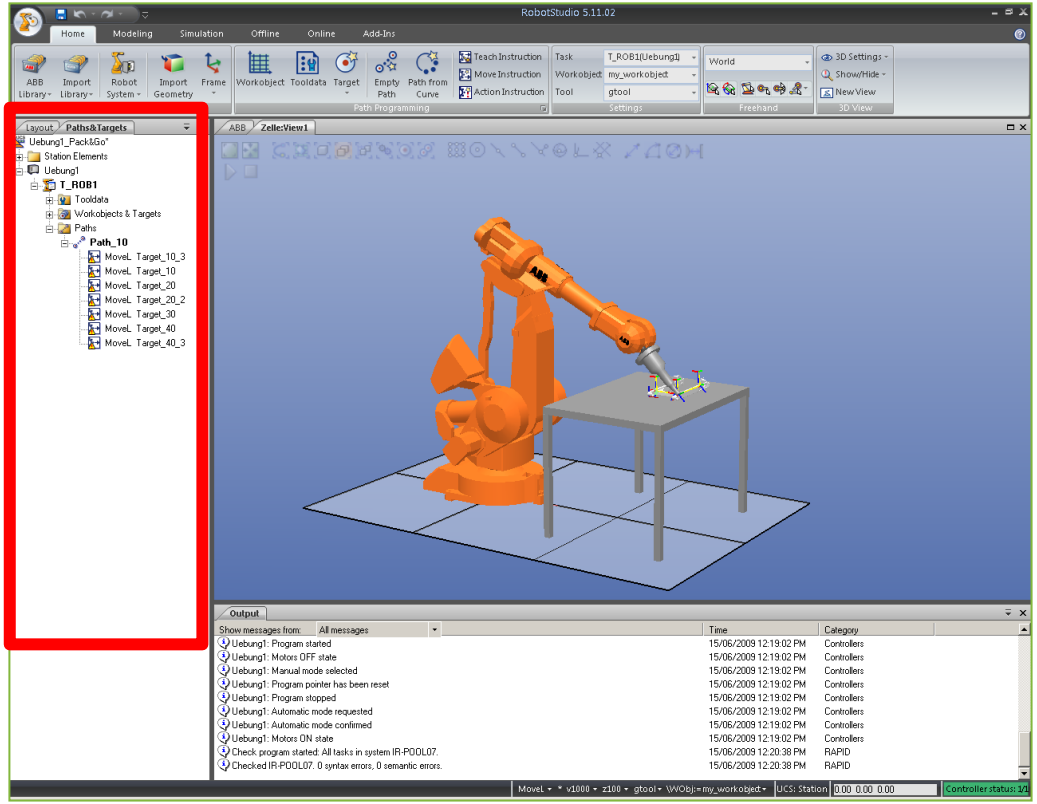
LAYOUTÜBERSICHT – HAUPTMENÜ



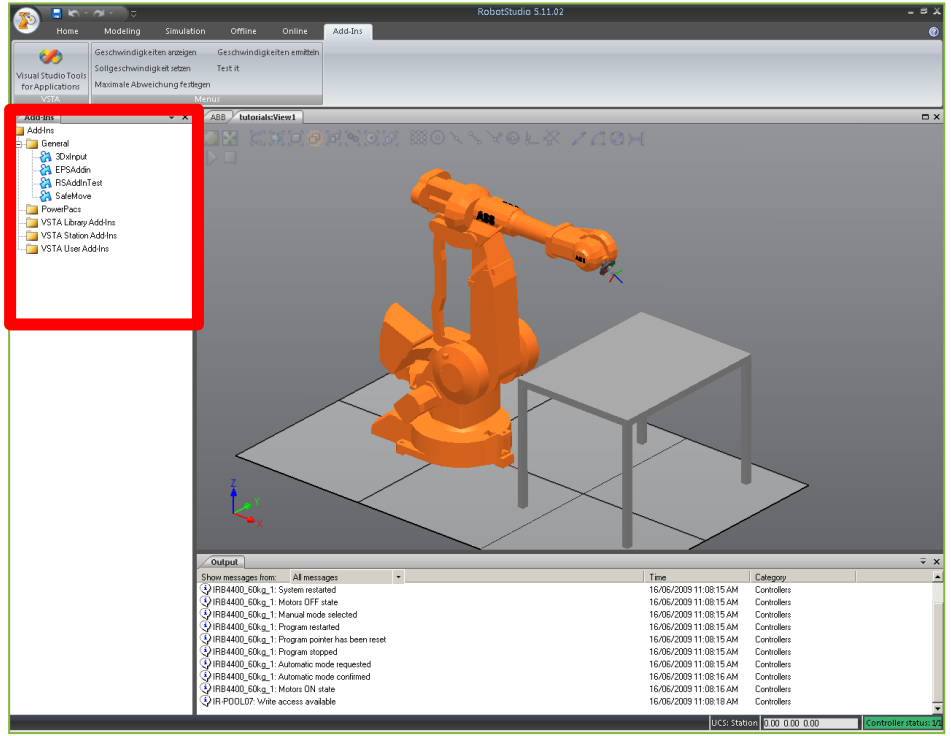
LAYOUTÜBERSICHT – LAYOUT BROWSER



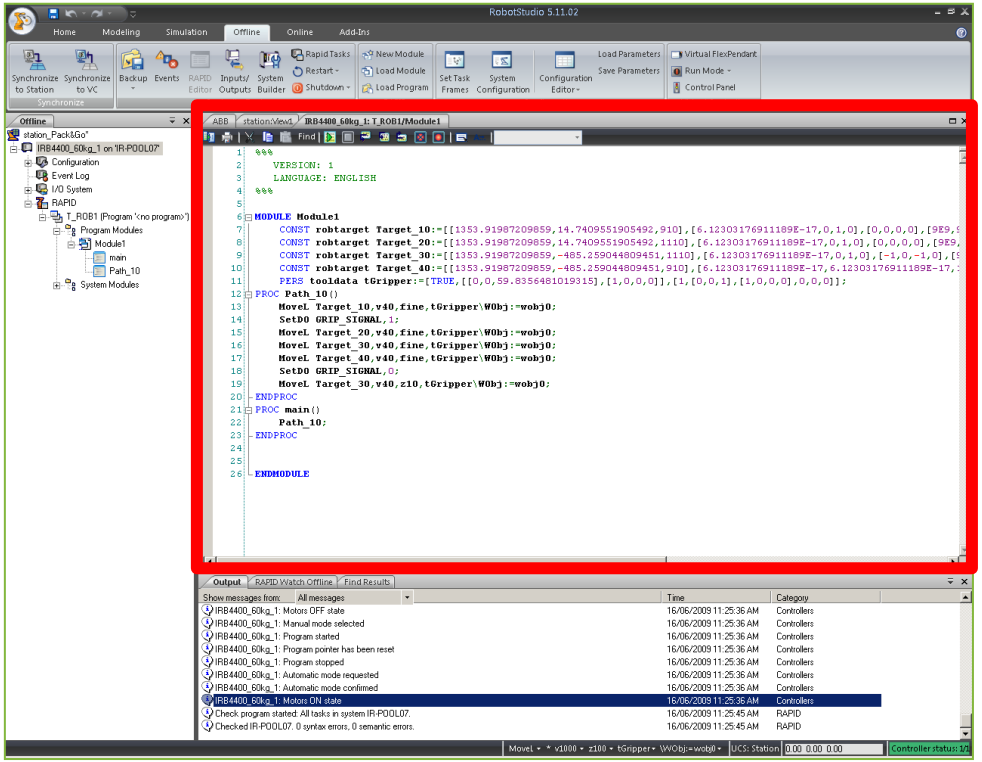
LAYOUTÜBERSICHT – PFADE & POSITIONEN-BROWSER



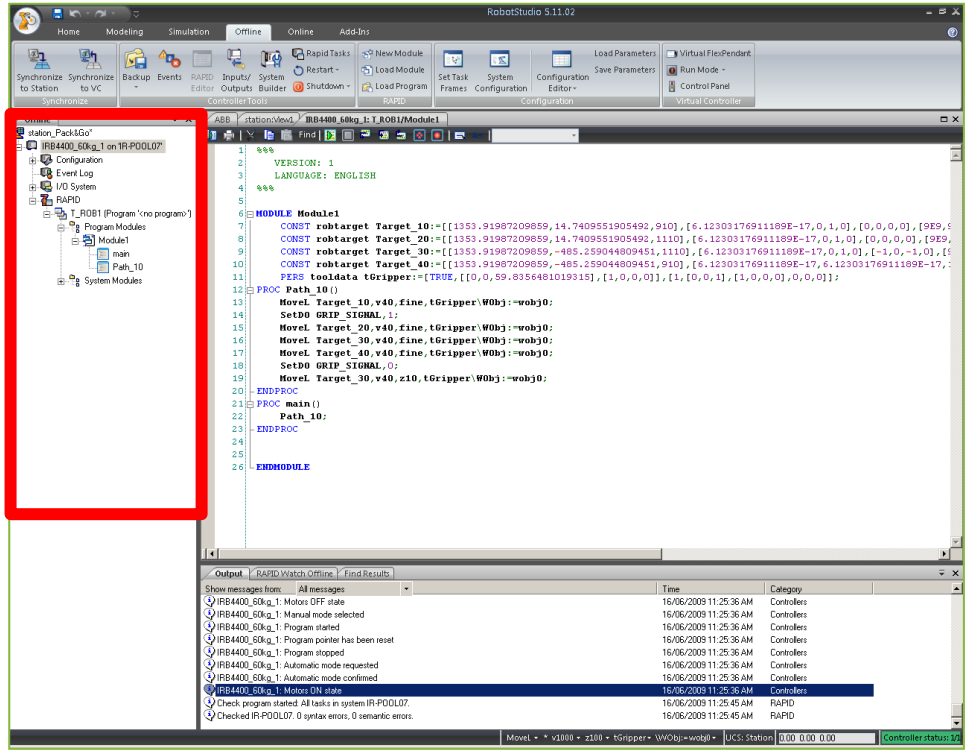
LAYOUTÜBERSICHT – ADDIN-BROWSER



PROGRAMMIERUMGEBUNG

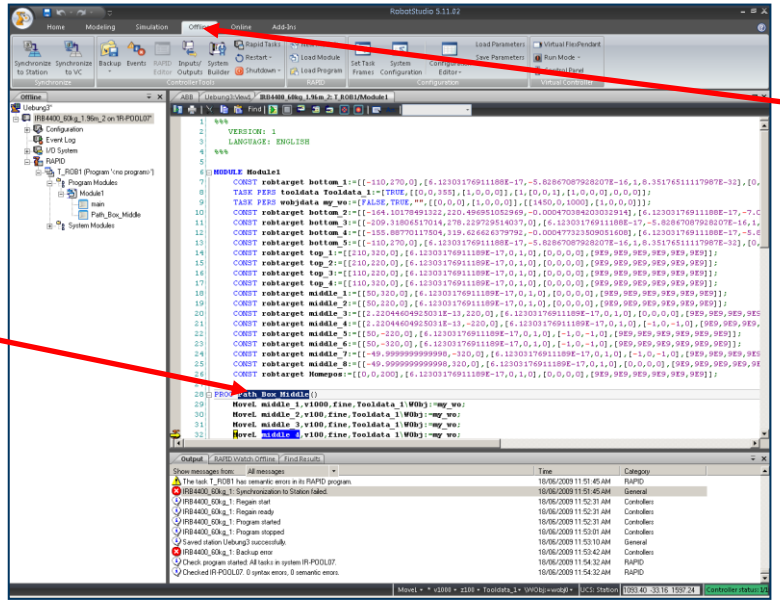


PROGRAMMIERUMGEBUNG



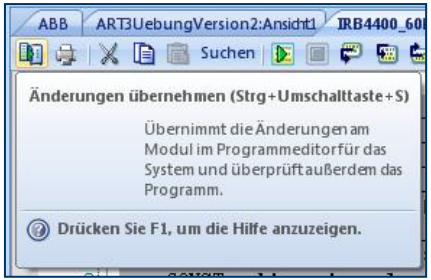
RAPID PROGRAMMIERUNG

- Zum Offline-Tab wechseln
- RAPID → T_ROB1 → Programmmodule → Module1 auswählen



SIMULATION SYNCHRONISIEREN

- **Änderungen am Programmcode übernehmen**



- **Programm mit Station synchronisieren**

- **Simulation starten**

SIMULATIONSFUNKTIONEN

- Verknüpfen / Lösen von Objekten, Werkzeugaustausch, Kollisionserkennung
- Definition von Ereignissen
  - Welche Bedingungen lösen diese Ereignisse aus?
  - Welche Aktion führt das Ereignis aus (z. B. Greifen)?
- Simulation von I/O Signalen
  - Ereignisgesteuert
  - Manuell gesteuert

PROGRAMMIERFUNKTIONEN

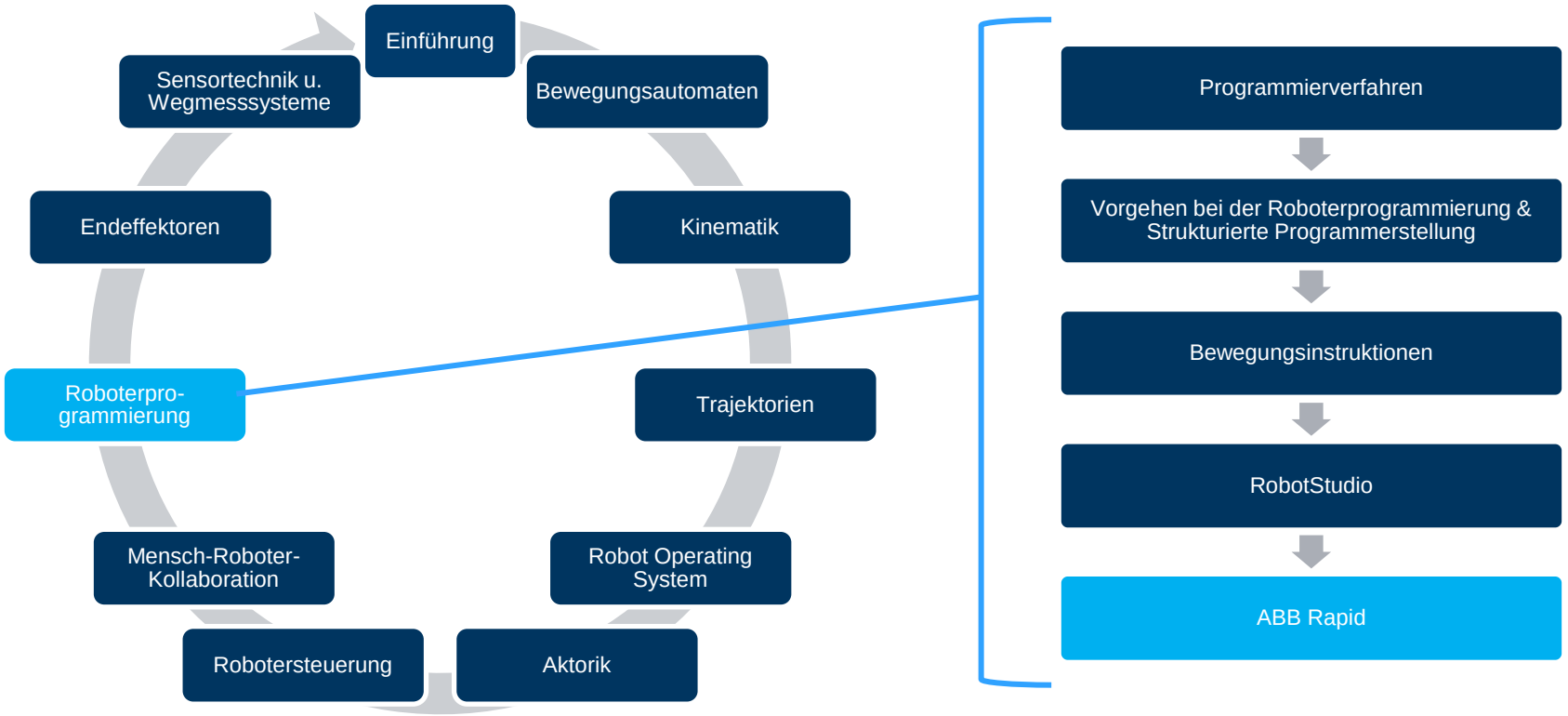
- **Unterstützende Programmfunktionen**
  - Laden von Programmen und Modulen
  - Schreiben und Modifizieren von Bedienerprogrammen
  - Auswahl von Startprozeduren
  - Syntax- und Semantikkontrolle
  - Einrichten von
    - Haltepunkten
    - Programmzeigern
  - …

SIMULATION

- Unterstützende Tests
  - Erreichbarkeit für Layoutoptimierungen
  - Kollision
  - Durchlaufzeit
  - ..

- Austausch von Programmen mit dem Virtual Controller
  - Synchronisation
    - mit dem Virtual Controller
    - mit der Station

VORLESUNGSINHALTE



RAPID – GRUNDLAGEN

- **Programmiersprache der ABB Roboter**
  - Prozedurale Programmierung (vgl. C, Pascal)

- **Struktur eines RAPID-Programms**
  - Module
    - Programmmodule
    - Systemmodule
  - Globale Deklarationen
    - Targets, Werkzeuge, Variablen, etc.
  - Prozeduren
    - Menge an Instruktionen
    - Eine Haupt-Prozedur (main)

RAPID-PROGRAMM

```
MODULE MyModule

        CONST robtarget Target_10:=[…];

        PROC main()
                MyPath;
        ENDPROC

        PROC MyPath()
                MoveL Target_10,…;
        ENDPROC

ENDMODULE
```

Struktur eines RAPID-Programms

ÜBERSICHT

- **Befehlsweise Abarbeitung des Quellcodes**
  - eine Ausführung von unten nach oben (rückwärts) ist möglich
- **Trennung zwischen Befehlen bzw. Abschluss eines Befehls durch ein Semikolon („;“)**
- **überflüssige Zeilenumbrüche, Leerzeichen und Tabulatoren werden ignoriert und ändern das Programmverhalten nicht**
- **Kommentare sind möglich**
  - Kennzeichnung mit einem Ausrufezeichen („!“)
  - Ende des Kommentars mit dem nächsten Zeilenumbruch
- **maximale Länge von Bezeichnern: 16 Stellen**
- **es wird nicht zwischen Groß- und Kleinschreibung unterschieden**

ZUWEISUNGEN

- **Wertezuweisung an Variablen:**
- **durch Zuweisung eines neuen Wertes wird der alte Wert der Variablen überschrieben**
- **Zuweisungsoperator in RAPID ist: „:=“**
- **Achtung: „=“ ist der Vergleichsoperator!**

- **Beispiel:**

```
VAR num zahl;
! korrekt, hier wird der Wert 2 zugewiesen
num := 2;
! falsch, hier wird die Variable num mit 2 verglichen
num = 2;
```

ATOMARE DATENTYPEN

- **interne Basisdarstellung, die nicht weiter unterteilt werden kann**
- **in RAPID:**
  - Zahlen (num)
  - Wahrheitswerte (bool)
  - Zeichenketten (string)
- **komplexere Datentypen können aus diesen atomaren Datentypen aufgebaut werden (record)**

```
RECORD PositionData
        num x;
        num y;
        num z;
        string label;
ENDRECORD
```

```
VAR PositionData pos1;
        pos1.x := 100;
        pos1.y := 200;
        pos1.z := 300;
        pos1.label := "Startpunkt";
```

ATOMARE DATENTYPEN – NUM

- **num wird für numerische Werte wie z. B. Zähler verwendet.**
- **Es können folgende Werte angenommen werden:**
  - eine Ganzzahl, z. B. -5
  - eine Dezimalzahl, z. B. 3,45
- **Exponentialschreibweise ist möglich**
  - 2E3 (= 2 x 10^3 = 2000)
  - 2.5E-2 (= 0,025).

- **Beispiel:**

```
VAR num ganzzahl; ! Deklaration

! ganzzahl wird der Wert 3 zugewiesen.
ganzzahl := 3; ! Initialisierung
```

ATOMARE DATENTYPEN – NUM

- **Unterscheidungen zwischen Gleitkommazahlen und Ganzzahlen**
  - wird einer Variable eine Ganzzahl zugewiesen, so bleibt sie erhalten, so lange sie sich im darstellbaren Wertebereich befindet
  - Addition, Subtraktion, Multiplikation sowie unäres Plus bzw. Minus erhalten die Repräsentation als Ganzzahl
  - Voraussetzung für den Erhalt der Repräsentation als Ganzzahl:
    - das Ergebnis liegt innerhalb des darstellbaren Bereichs
    - alle beteiligten Werte an der Operation werden ebenfalls durch Ganzzahlen repräsentiert
- Beispiel:

```
VAR num zahl, ergebnis;
zahl := 2;
ergebnis := zahl * 2; ! ergebnis ist eine Ganzzahl
ergebnis := zahl * 2.5; ! ergebnis ist eine
Gleitkommazahl
```

ATOMARE DATENTYPEN – STRING

- **string wird für Zeichenfolgen verwendet.**
- **eine Zeichenfolge (string) besteht aus einer Reihe von Zeichen (maximal 80), die von (doppelten) Anführungszeichen (““) umgeben sind**
- **es können die Zeichen 0 – 255 von ISO 8859-1 verwendet werden**
- **es ist möglich ein Anführungszeichen mit einem Backslash zu maskieren (\“)**
- **ein Backslash kann durch einen Backslash maskiert werden (\\)**

- **Beispiel:**

```
VAR string text;
text := "Beginn Schweißen von \"Rohr 1\"";
! in der Variablen text steht: Beginn schweißen von "Rohr 1"
```

ATOMARE DATENTYPEN – BOOL

- **bool repräsentiert logische Werte (TRUE / FALSE)**
- **Beispiel:**

```
VAR bool highvalue;
VAR num reg1;

reg1 := 150;

highvalue := reg1 > 100; ! highvalue = TRUE
highvalue := reg1 < 100; ! highvalue = FALSE
```

VARIABLEN - GÜLTIGKEIT

- **LOCAL: die Variable ist nur innerhalb des aktuellen Moduls sichtbar (kann nicht innerhalb von Routinen verwendet werden)**
- **PERS: die Variable ist eine Persistente → Wert bleibt erhalten, auch wenn das Programm beendet oder neu gestartet wird.**
- **CONST: die Variable ist eine Konstante, der Wert lässt sich nicht ändern**

System-ereignis beein-flusst	Neustart (Warmstart)	Öffnen, Schließen oder neues Programm	Programmstart (PZ auf „main“)	Programmstart (PZ auf Routine)	Programmstart (PZ auf Cursor)	Programmstart (Routinenaufruf)	Programmstart (nach Zyklus)	Programmstart (nach Stopp)
Konstante	Unverändert	Initialisierung	Initialisierung	Initialisierung	Unverändert	Unverändert	Unverändert	Unverändert
Variable	Unverändert	Initialisierung	Initialisierung	Initialisierung	Unverändert	Unverändert	Unverändert	Unverändert
Persistente	Unverändert	Initialisierung / Unverändert	Unverändert	Unverändert	Unverändert	Unverändert	Unverändert	Unverändert

Persistenten ohne Anfangswert werden nur initialisiert, wenn sie noch nicht deklariert sind

DATENTYPEN – TOOLDATA

- **Werkzeugdaten werden verwendet, um die Eigenschaften eines Werkzeugs, z. B. eines Schweißgeräts oder eines Greifers, zu beschreiben.**
- **Komponenten:**
  - Hält der Roboter das Werkzeug oder ist es fest.
  - Das Werkzeug-Koordinatensystem
  - Die Last des Werkzeugs

DATENTYPEN – TOOLDATA

- **Beispiel:**

```
PERS tooldata gripper := [ TRUE, [[97.4, 0, 223.1], [0.924, 0,
0.383 ,0]], [5, [23, 0, 75], [1, 0, 0, 0], 0, 0, 0]];
```

- **Das Werkzeug wird mit den folgenden Werten beschrieben:**
  - Der Roboter hält das Werkzeug.
  - Der TCP befindet sich an Punkt 223.1 mm geradeaus von Achse 6 und 97.4 mm entlang der X-Achse *des* Handgelenk-Koordinatensystems.
  - Die X- und Z-Richtung des Werkzeugs wird 45° um das Handgelenk-Koordinatensystem gedreht. (in Quaternion)
  - Das Werkzeug wiegt 5 kg.
  - Der Schwerpunkt befindet sich am Punkt 75 mm geradeaus von Achse 6 und 23 mm entlang der x-Achse des Handgelenk-Koordinatensystems.
  - Die Last kann als Punktlast betrachtet werden, d. h. ohne Trägheitsmoment.

DATENTYPEN – TOOLDATA

- **Werkzeugdaten beeinflussen die Roboterbewegungen wie folgt:**
  - Werkzeugarbeitspunkt (TCP) verweist auf einen Punkt, der die Bedingungen für die angegebene Bahn und Geschwindigkeitsleistung erfüllt. Wenn das Werkzeug neuorientiert wird, folgt nur dieser Punkt der gewünschten Bahn in der programmierten Geschwindigkeit.
  - Wenn ein stationäres Werkzeug verwendet wird, beziehen sich programmierte Geschwindigkeit und Bahn auf das Werkobjekt, das der Roboter hält.
  - Programmierte Positionen verweisen auf die Position des aktuellen TCP und die Orientierung in Relation zum Werkzeug-Koordinatensystem.

DATENTYPEN – TOOLDATA

- **Die Angabe falscher Lastdaten für das Werkzeug kann folgende Konsequenzen haben:**
  1. Wenn die angegebene Last den Wert der tatsächlichen Last überschreitet:
     - Der Roboter nutzt nicht seine maximale Kapazität.
     - Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.
  2. Wenn die angegebene Last den Wert der tatsächlichen Last unterschreitet:
     - Die mechanische Struktur kann überlastet werden.
     - Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.

DATENTYPEN – ROBTARGET

Merkmale

- **kombiniert mehrere Basiswerte (num) zu einem Datensatz.**
- **Wird genutzt, um Zielpunkte für Bewegungen (MoveJ, MoveL, …) zu definieren.**
- **Kann als CONST festgelegt werden → unveränderlich im Programm.**

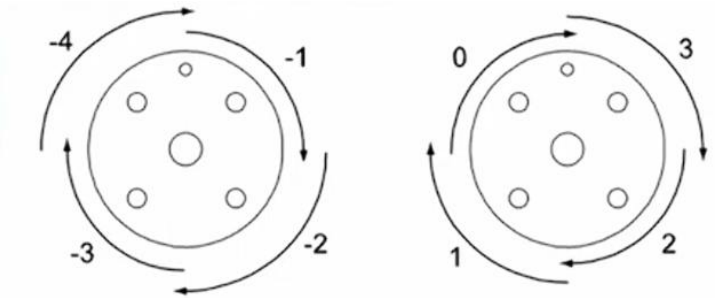
Aufbau eines robtarget

- **Positionsvektor [X,Y,Z]**
  - Kartesische Koordinaten des TCP (Tool Center Point) im Raum
- **Orientierung [q1,q2,q3,q4]**
  - Quaternion zur Beschreibung der Lage des Werkzeugs
- **Konfiguration [cf1,cf4,cf6,cfX]**
  - Informationen über Gelenkstellung und Roboterkonfiguration (z. B. Ellbogen oben/unten, Handgelenkstellung)
- **Externe Achsen [eax_a, eax_b, eax_c, eax_d, eax_e, eax_f]**
  - Werte für zusätzliche Achsen (z. B. Positionierer, Linearachsen) → 9E+09 bedeutet: nicht definiert / nicht relevant

```
CONST robtarget Home:=
[[0.000021,-0.000012419,199.999981547],
[-0.000000015,1,-0.000000057,-0.000000014],
[-1,-1,1,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

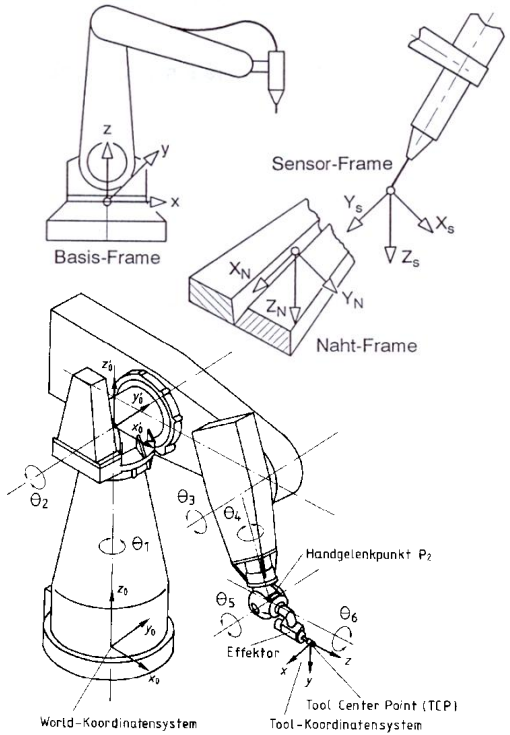
DATENTYPEN – ROBTARGET - KONFIGURATION

- **Eindeutige Festlegung der Gelenkstellung bei gleicher TCP-Pose**
- **Sichert reproduzierbare Bewegungen und vermeidet „Wrist-Flips“**
- **Konfiguration [cf1,cf4,cf6,cfX]**
  - cf1 → Ellbogenstellung (oben/unten)
  - cf4 → Handgelenksstellung (innen/außen)
  - cf6 → Orientierung der letzten Achse (z. B. Drehung des Flansches)
  - cfx → Zusatzinformation für Sonderfälle



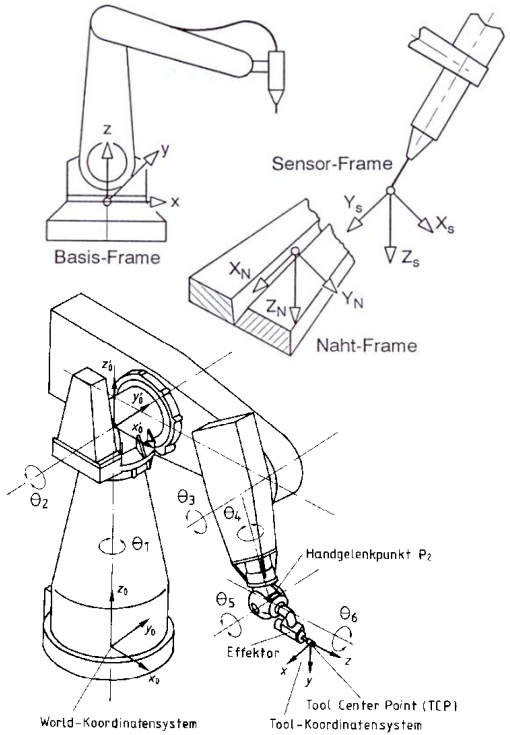
KOORDINATENSYSTEME IN DER ROBOTIK

- **Weltkoordinatensystem**
  - die Basis für alle anderen Koordinatensysteme, fixiert
- **Basiskoordinatensystem**
  - im Sockel des Roboters, relativ zum Weltkoordinatensystem definiert (oft identisch)
- **Benutzerkoordinatensystem**
  - in definiertem Punkt der Werkobjekte, oft relativ zum Weltkoordinatensystem



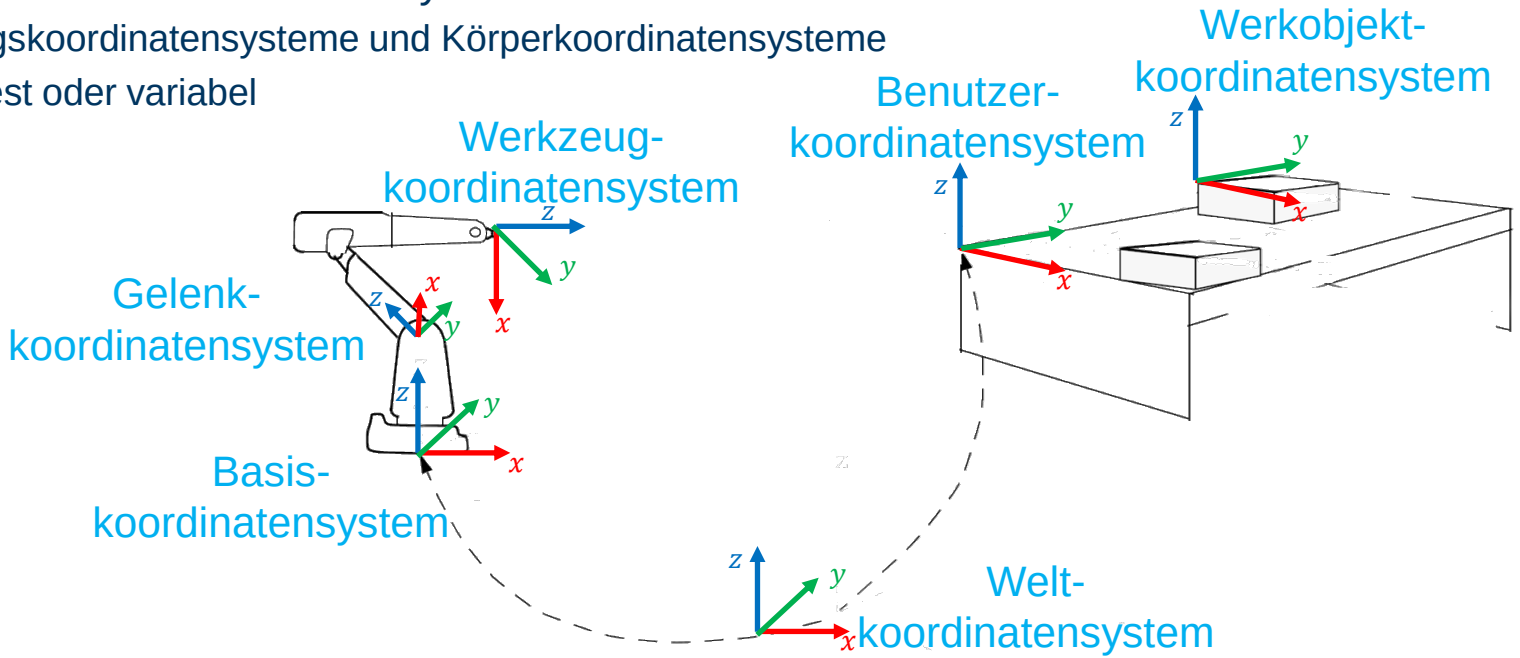
KOORDINATENSYSTEME IN DER ROBOTIK

- **Werkobjektkoordinatensystem**
  - im Werkobjekt, oft relativ zum Benutzerkoordinatensystem
- **Gelenkkoordinatensystem**
  - jedes Gelenk des Roboters besitzt ein Koordinatensystem, das relativ zu anderen Koordina-tensystemen ist
- **Werkzeugkoordinatensystem**
  - wird mit dem Roboter mitbewegt, relativ zum TCP (Tool Center Point)



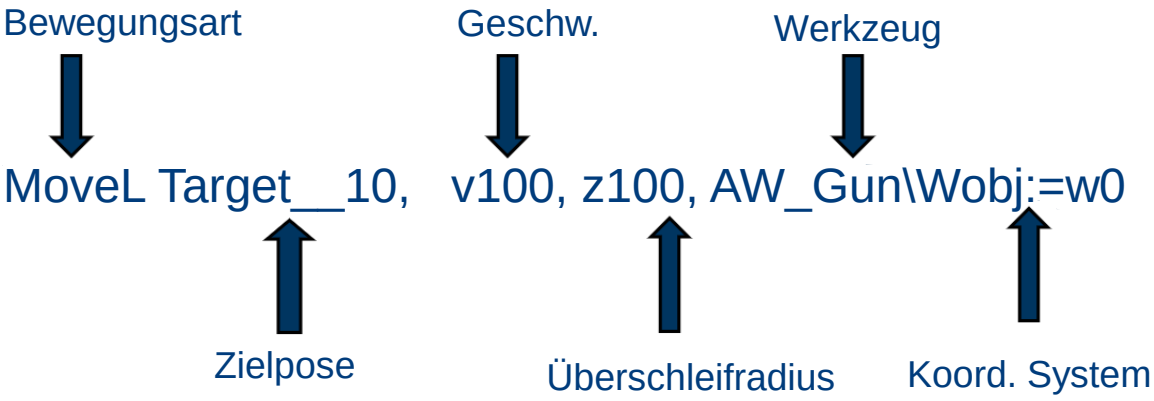
ROBOTERKOORDINATENSYSTEME

- Roboter beschreiben **Posen (Position und Orientierung)** im Arbeitsbereich mit verschiedenen Koordinatensystemen
  - Bezugskoordinatensysteme und Körperkoordinatensysteme
  - Ortsfest oder variabel



BEWEGUNGSINSTRUKTIONEN

- **Mögliche Bewegungsinstruktionen**
  - MoveJ – Achsbewegungen
  - MoveL – Linearbewegungen
  - MoveC – Zirkularbewegungen
- **Notwendige Parameter**
  - Zielpose, Geschwindigkeit, Überschleifradius, Werkzeug



INSTRUKTIONEN - MOVEJ

- **MoveJ: Point-to-Point Bewegung (vereinfachte Syntax)**

```
MoveJ [\Conc] ToPoint [\ID] Speed
[\V] | [\T] Zone [\Z] [\Inpos] Tool
[\WObj]
```

- **Beispiel: Deklaration**

```
MoveJ p1, vmax, z30, tool2;
MoveJ *, vmax \T:=5, fine, grip3;
```

- **Parameter**
  - \Conc (optional)
  - ToPoint ← robtarget
  - \ID (optional)
  - Speed ← speeddata
  - \V (optional)
  - \T (optional)
  - Zone ← Zonedata
  - \Z (optional)
  - \Inpos (optional)
  - Tool ← Tooldata
  - \WObj (optional) ← Workobject

INSTRUKTIONEN - MOVEL

- **MoveL : Continuous Path (Linear)**

```
MoveL [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] [\Inpos] Tool
[\WObj] [\Corr]
```

  - Parameter:
    - Analog zur MoveJ Instruktion
    - \Corr - Korrekturdaten
- **Beispiel: Deklaration**

```
MoveL p1, v1000, z30, tool2;
MoveL start, v2000, z40, grip3\WObj:=fixture;
```

  - Wobj: Koordinatensystem auf das sich die Roboterposition bezieht (Datentyp: wobjdata)

INSTRUKTIONEN – MOVEC

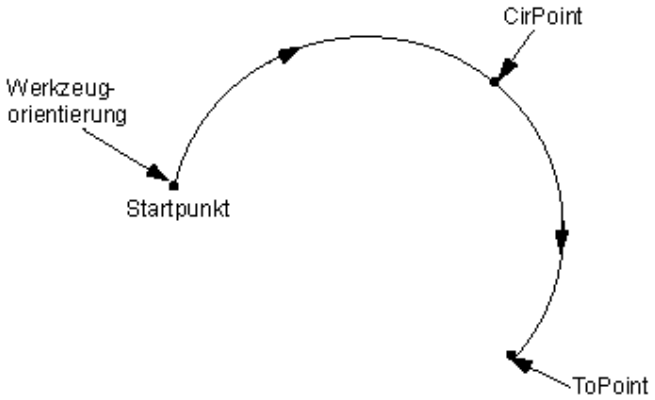
- **MoveC: Continuous Path (Zirkular)**

```
MoveC [\Conc] CirPoint ToPoint [\ID] Speed [\V] | [\T] Zone [\Z]
[\Inpos] Tool [\WObj] [\Corr]
```

  - Parameter:
    - Analog zur MoveL Instruktion
    - CirPoint- Bogenpunkt des Roboters

  - Beispiel: Deklaration

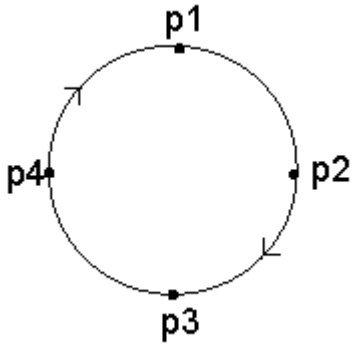
```
MoveC p1, p2, v500, z30, tool2;
```



INSTRUKTIONEN – MOVEC

- **MoveC: Continuous Path (Zirkular)**
  - Beispiel: vollständiger Kreis durch zwei MoveC-Instruktionen

```
MoveL p1, v500, fine, tool1;
MoveC p2, p3, v500, z20, tool1;
MoveC p4, p1, v500, fine, tool1;
```



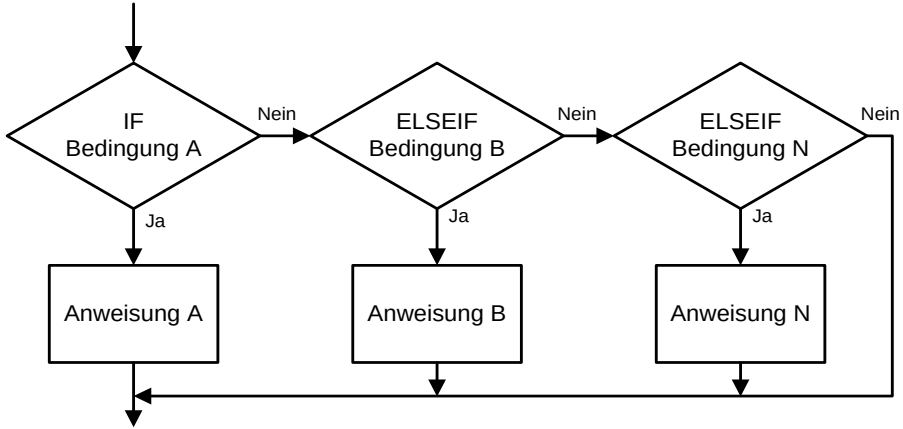
INSTRUKTIONEN – OFFSETS (OFFS)

- **Offsets sind Verschiebungen relativ zu einem bestehenden Zielpunkt (robtarget).**
- **Sie ändern die Position oder Orientierung des TCP ohne neuen Punkt zu teachen.**
- **Arten von Offsets:**
  - Translational (X, Y, Z): Verschiebung in Raumrichtungen.
  - Rotational (Orientierung): Änderung der Lage des Werkzeugs.
  - Kombiniert: Position + Orientierung.
- **Vorteile:**
  - Flexibel: gleiche Basisposition mehrfach nutzbar.
  - Effizient: weniger Teachpunkte nötig.
  - Präzise: kleine Anpassungen ohne neue Koordinaten.
- **Syntax-Beispiel:**
  - rapidMoveL Offs(Target_10, 50, 0, 0), v100, z10, tool1\WObj:=Workobject;
  - Bewegung 50 mm in X-Richtung relativ zu Target_10.

STATEMENTS – IF

- **IF-Statement**
  - Beispiel:

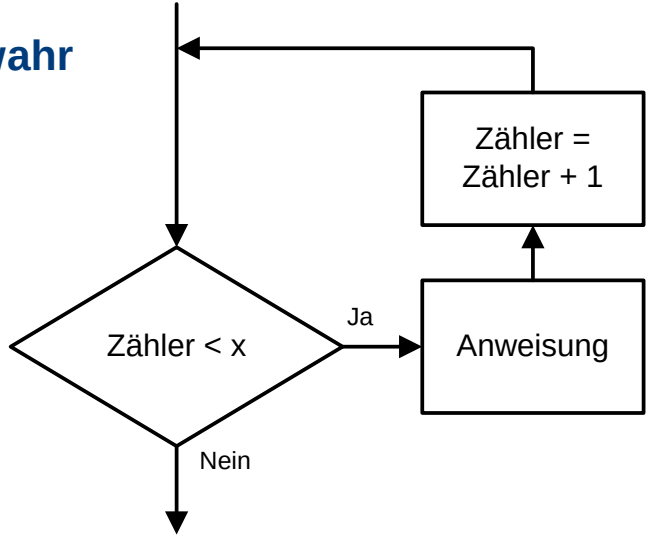
```
IF counter > 100 THEN
    counter := 100;
ELSEIF counter < 0 THEN
    counter := 0;
ELSE
    counter := counter + 1;
ENDIF
```



STATEMENTS – WHILE

- **ein Programmblock wird so lange durchlaufen, wie die Kontrollstruktur wahr ergibt**
  - Beispiel:

```
WHILE a < b DO
  ...
  a := a + 1;
ENDWHILE
```

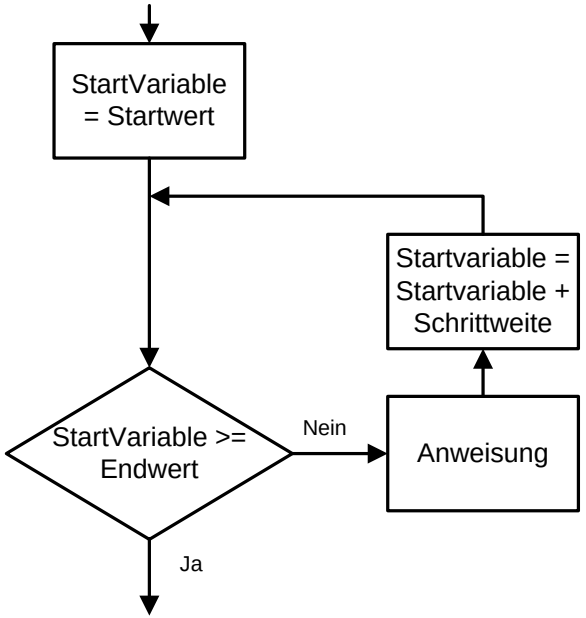


STATEMENTS – FOR

- **führt einen Instruktionblock solange aus, wie die Kontrollstruktur wahr ergibt**
- **dabei wird eine Variable vom gegebenen Anfangswert zum Endwert inkrementiert oder dekrementiert**

  - Beispiel:

```
FOR i FROM 1 TO 10 STEP 1 DO
        x := x + i;
ENDFOR
```



ZEITFUNKTION

- **Einfache Zeitfunktionen:**
  - Die Programmabarbeitung wird für eine vorgegebene Wartezeit unterbrochen

- **Signalabhängige Zeitfunktionen:**
  - Die Programmabarbeitung wird unterbrochen bevor ein Ereignis zu einer wahren logischen Verknüpfung führt

SCHALTFUNKTIONEN

- **Einfache Schaltfunktion:**
  - Schaltfunktionen werden verwendet, um den Wert eines digitalen Ausgangssignals zu ändern.

- **Einfache Impulsfunktion**
  - Ein digitaler Ausgang wird für eine definierte Zeit gesetzt bzw. Zurückgesetzt.

- **Bahnabhängige Schaltfunktion**
  - Bahnabhängige Schaltfunktionen werden verwendet, um Ausgangssignale an festen Positionen zu setzen, während der Roboter Bewegungen durchführt.

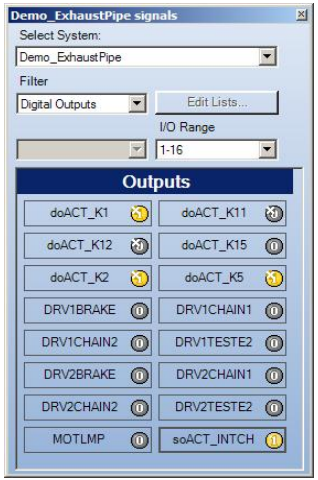
RAPID SIGNALE

- **Signal: Nachricht, beschrieben durch physikalische Größe**
- **Unterscheidung zw. Eingabe und Ausgabe (I/O bzw. E/A)**
  - Ausgehende Signale (Ausgang)
  - Eingehende Signale (Eingang)
- **Unterscheidung zw. diskreten und kontinuierlichen Größen**
  - Diskret: HIGH bzw. LOW (Digitale Größe)
  - Kontinuierlich: beliebiger Wert (Analoge Größe)
- **Verwendung**
  - Definition, Zuweisung
  - Nutzung für den Programmfluss
  - Werkzeugsteuerung
  - Sensorabfragen
  - …

RAPID SIGNALE

- **Realität: I/O-Module mit 8, 16, … Ein- oder Ausgängen**

- **Simulation: virtuelle Abbildung der I/O-Module**



Quelle Moeller, thermokon

RAPID SIGNALE

- **Anwendung Digitaler Signale**
  - Setzen von Zuständen
    - Wert eines digitalen Ausgangssignals setzen auf 1=HIGH oder 0=LOW
    - Invertieren des Wertes eines digitalen Ausgangssignals
    - Generieren eines Impulses auf einem digitalen Ausgangssignal
  - Abfragen von Zuständen
    - Direktes Lesen eines digitalen Eingangs: (IF di1 = 1 THEN ...)
    - Auf das Setzen oder Rücksetzen eines digitalen Eingangssignals warten
    - Testen, ob ein digitales Eingangssignal gesetzt ist.
  - Hauptaufgabe:
    - Kommunikation der Roboter (-steuerungen) sowie Peripherie untereinander

RAPID – SIGNALE (DIGITAL)

- **Beispiel**
  - Abfrage eines Sensors
  - Bewegungen
  - Greiferansteuerung

```
PROC GripPartIfAvailable()
        IF Detector = 1 THEN
                MoveL p20, v100, fine, TCP1;
                SetDO CloseGripper, 1;
                MoveL p10, v100, fine, TCP1;
        ELSE
                ! no part available
        ENDIF
ENDPROC
```

INSTRUKTIONEN - SET / RESET

- **Set**
  - Aktiviert ein Signal oder eine Funktion.
  - Beispiel: Set doGripperClose;
  - Bedeutet: Digitalausgang (DO) wird gesetzt = „Greifer zu“.
- **Reset**
  - Deaktiviert ein Signal oder eine Funktion.
  - Typisch: Greifer öffnen → Werkstück freigeben.
  - Beispiel: Reset doGripperClose;
  - Bedeutet: Digitalausgang wird zurückgesetzt = „Greifer auf“

INSTRUKTIONEN – WAITTIME

- **Programmbefehl, der den Ablauf für eine bestimmte Zeitspanne anhält.**
- **Syntax: WaitTime <Sekunden>;**
- **Funktion:**
  - Erzwingt eine Pause im Programmfluss.
  - Roboter bleibt in aktueller Position stehen.
  - Zeit wird in Sekunden angegeben (Ganzzahl oder Fließkommazahl).
- **Beispiel:**

```
MoveL Target_10, v100, z10, tool1;
Set doGripperClose;
WaitTime 0.2;   -- Pause für 0,2 Sekunden
```

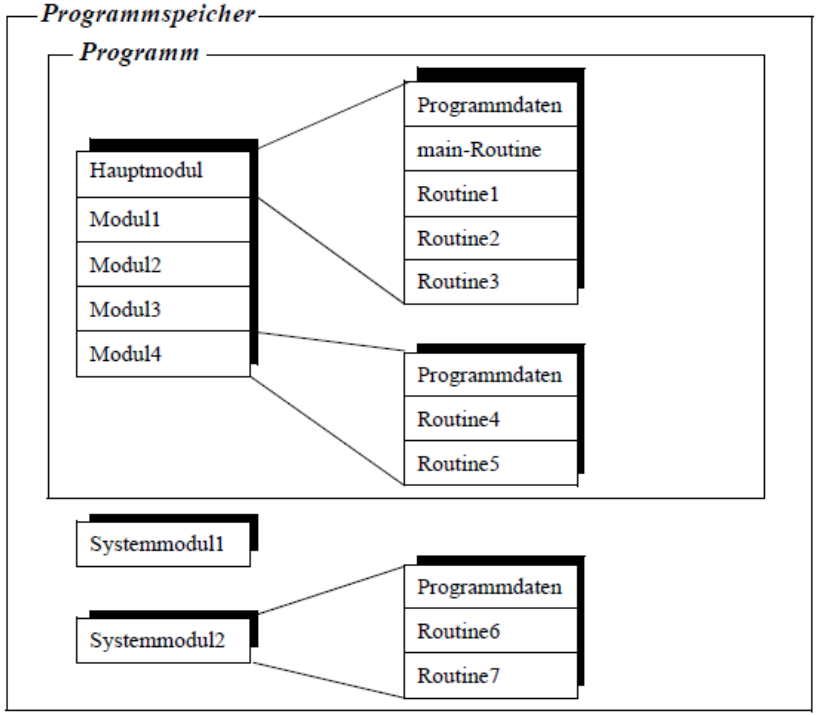
- **Einsatzgebiete:**
  - Warten auf Prozessschritte (z. B. Greifer schließen, Kleber aushärten).
  - Synchronisation mit externen Geräten.
  - Kurze Pausen zur Sicherheit oder Beobachtung.

MODULE

- Ein RAPID-Programm kann aus einem oder mehreren Modulen bestehen. Jedes Modul kann eine oder mehrere Prozeduren enthalten.
- Einfache Programme verwenden oft nur ein Modul.
- In komplexeren Programmierumgebungen können einige Standardprozeduren, die von vielen verschiedenen Programmen verwendet werden, in ein eigenes Modul eingefügt werden.

PROGRAMMSTRUKTUR

- Gliederung der Programmstruktur in unterschiedliche Teilbereiche
- Start der Abarbeitung in der main-Routine



BEISPIEL - MODULE

```
MODULE MainModule
        VAR num x_index;
        VAR num y_index;

        draw square x_index, y_index; ! Ohne Klammern
ENDMODULE

Module figures_moduls
        PROC draw_square(Num x_rel, Num y_rel)
        …
        ENDPROC
        PROC draw_triangle(Num a_rel, Num b_rel, Num c_rel)
        …
        ENDPROC
        PROC draw_circle(Num d_rel)
        …
        ENDPROC
ENDMODULE
```

ROUTINEN

- **In RAPID gibt es drei Arten von Routinen**
  - Prozeduren
  - Funktionen
  - Interrupt-Routinen

- **eine Routine ist global sichtbar**
- **Ausnahme: Bei der Definition wird das Schlüsselwort LOCAL verwendet:**
  - Sichtbarkeit der Routine beschränkt sich auf das umschließende Modul
  - lokale Routinen verdecken innerhalb ihres Gültigkeitsbereichs globale Routinen mit dem selben Namen

- **Der Name einer Routine darf nicht identisch mit dem Namen, Daten oder Datentypen innerhalb seiner Gültigkeit sein**

ROUTINEN – PARAMETER

- **Parameterliste gibt die Argumente der Routine an, die bei einem Aufruf angegeben werden müssen**
- **RAPID kennt vier Unterschiedliche Arten von Parametern:**
  - ohne weitere Kennzeichnung: es erfolgt eine Kopie des Arguments (call by value)
  - INOUT: das Argument muss eine Variable oder Persistente sein, die durch die Routine geändert werden kann (call by reference)
  - VAR: das Argument muss eine Variable sein, die durch die Routine geändert werden kann (call by reference)
  - PERS: Argument muss eine Persistente sein, die durch die Routine geändert werden kann (call by reference)
- **Beispiel:**

```
PROC routine (num eins, INOUT zwei, VAR drei, PERS vier)
  ! Quellcode der Routine
ENDPROC
```

ROUTINEN – AUFRUF UND RÜCKKEHR

- **Verwendung:**
  - Routinen können anstelle einer Instruktion benutzt werden
  - Prozeduren liefern keinen Rückgabewert
- **Aufruf:**
  - es muss der Name der Routine benutzt werden
  - alle Argumente müssen in der korrekten Reihenfolge benutzt werden
- **Rückkehr:**
  - eine Routine wird implizit beim Erreichen des Schlüsselworts ENDPROC verlassen
  - alternativ ist ein explizites Verlassen mittels des Schlüsselworts RETURN möglich

FUNKTIONEN

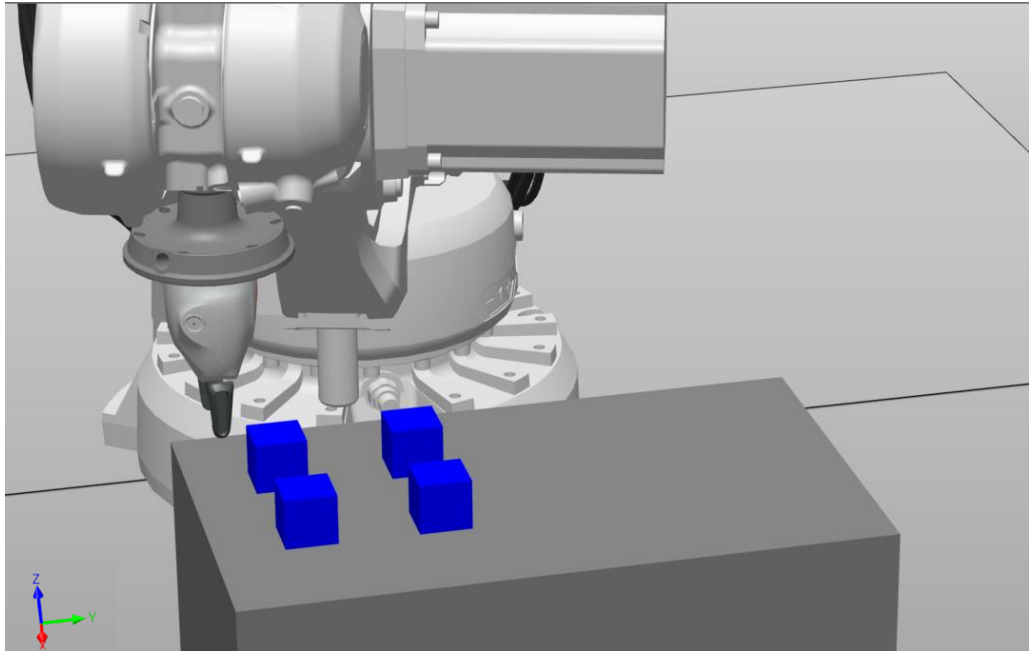
- **Funktionen berechnen einen Wert und liefern ihn zurück**
- **die Rückkehr aus der Funktion muss mittels RETURN erfolgen**
- **Verhalten ansonsten ähnlich wie Prozeduren**

- **Beispiel:**

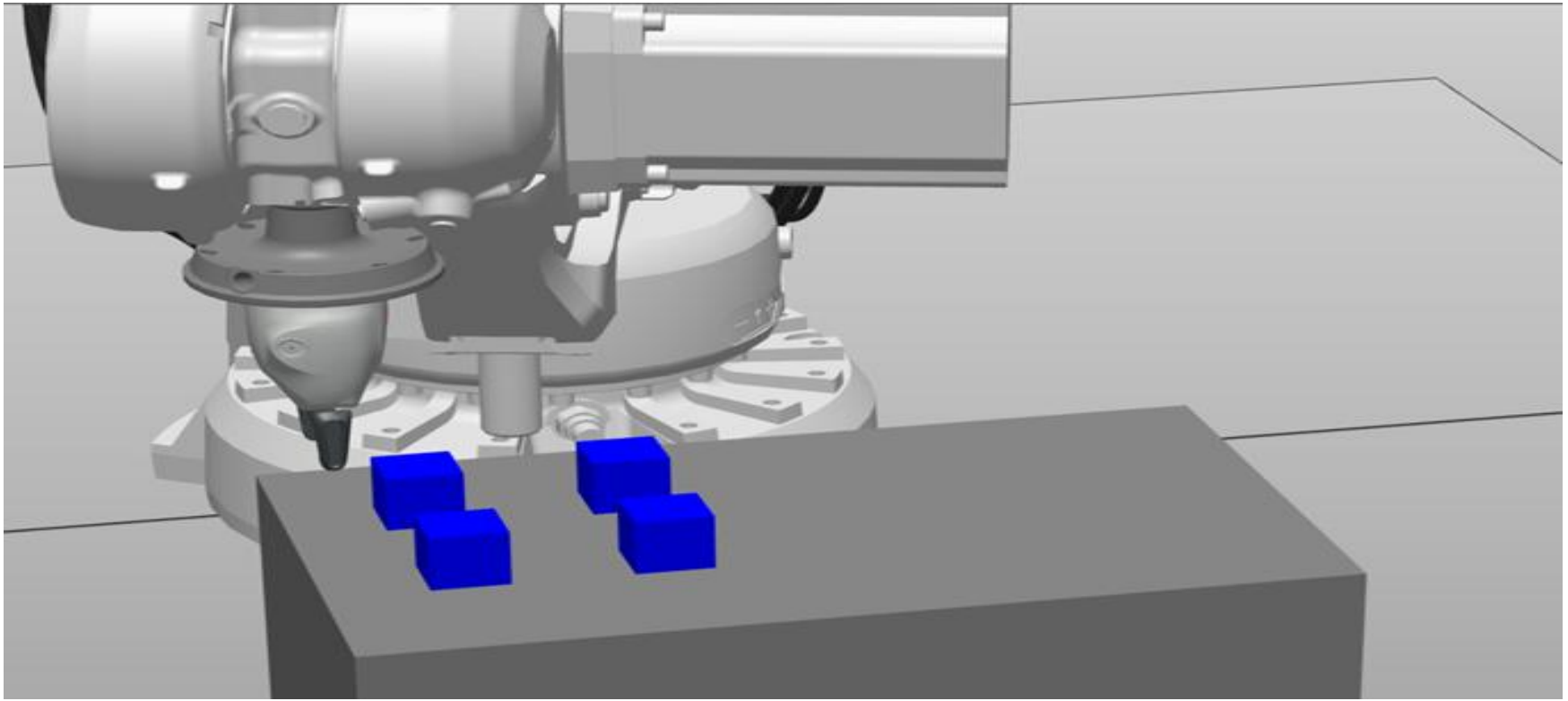
```
FUNC num veclength(pos vector)
        VAR num length;
        length := sqrt(pos.x * pos.x + pos.y * pos.y + pos.z * pos.z);
        RETURN length;
ENDFUNC
```

ÜBUNG

- **Übung - Sortier- und Suchaufgabe mit ABB RAPID:**



LÖSUNG - AUFGABE 1



LÖSUNG - AUFGABE 1