Arbeitspaket 5: Skills

Einführungstext

**Kompetenzen von Skills**

Für die Definition eines Skills wird nicht die Funktion des Gesamtsystem so weit wie möglich in Teilfunktionalitäten heruntergebrochen, sondern die Funktionalitäten der Komponenten im System. Dabei betrachtet man jeden Komponenten-Typ, so weit wie möglich, einzeln. Der Vorteil dieser Definition ist, dass für neue oder andere Systeme dieselben Skills verwendet werden können. Ein Roboter hat in jedem System die gleichen Grundfunktionalitäten und somit Skills. Der Skill ist Anwendungsunabhängig. Die aus den Skills erstellten Sequenzen bilden die Funktion der Anwendung ab. Damit unterscheidet sich die Definierung der Skills von vergleichbaren Projekten wie z.B. der bereits durchgeführten Master-Thesis auf Basis von ROS (Ref). Dort haben Skills Funktionalitäten von mehreren Komponenten ineinander kombiniert, wodurch der Skill stärker an das spezifische System gebunden war.

**Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung**

Die Kompetenzen eines Skills lassen sich in vier Bereiche aufteilen:

|  |  |
| --- | --- |
| Zuweisung: | Der Skills ist für die Zuweisung einer Komponente zuständig. Es muss definiert werden können, welche Komponente den Skill ausführt. |

|  |  |
| --- | --- |
| Umsetzung: | Die definierte Grundfunktion muss innerhalb des Skills umgesetzt werden. Der Skill muss mittels Parameter-Inputs flexibel eingesetzt werden können. |

|  |  |
| --- | --- |
| Verarbeitung: | Die Informationen der Grundfunktion wird so ausgegeben, dass das Anlagenobjekt damit arbeiten und die reale Komponente ansteuern kann. |

|  |  |
| --- | --- |
| Auswertung: | Die momentane Situation der Komponente wird überwacht und ausgewertet. Der Skill kann auf bestimmte Situationen reagieren. |

**Definition von Anwendungs-Skills**

Um die benötigten Skills für die Anwendung zu definieren, werden im ersten Schritt die allgemeinen Arbeitsschritte basierend auf dem mechanischen Aufbau (siehe Verweis) festgelegt. Dabei wird von der Ausgangssituation ausgegangen, dass alle Teile in ihren Lagerpositionen abgelegt sind, der Roboter sich in der Home-Position befindet und alle Komponenten eingeschaltet sowie betriebsbereit sind.

|  |  |  |
| --- | --- | --- |
| Schritt | Aktivität | Komponente |
| 1 | Position von Platte 1 in Lagerung erkennen | Kamerasystem |
| 2 | Platte 1 an Montageposition bringen und mittels L-Stück ausrichten | Roboter, Greifer, Kraftsensor |
| 3 | Position von Platte 2 in Lagerung erkennen | Kamerasystem |
| 4 | Platte 2 an Montageposition bringen und mit Platte 1 zusammenführen | Roboter, Greifer, Kraftsensor |
| 5 | Position der Befestigungslöcher in den Platten ermitteln | Kamerasystem |
| 6 | Position von Befestigungsblech in Lagerung erkennen | Kamerasystem |
| 7 | Befestigungsblech an Montageposition bringen | Roboter, Greifer, Kraftsensor |
| 8 | Position von Stift 1 in Lagerung erkennen | Kamerasystem |
| 9 | Stift 1 an korrekte Position bringen | Roboter, Greifer |
| 10 | Befestigungsblech mit Platte 1 verbinden, durch Eindrücken von Stift 1 | Roboter, Kraftsensor |
| 11 | Wiederholen von Schritt 8 – 10 für Stift 2 bis 3 |  |

Eine detaillierte Auflistung der Arbeitsschritte wird im Anhang beigefügt. Aus diesem lässt sich erkennen, dass sich diverse Schritte mit kleinen Anpassungen wiederholen. Diese sich wiederholenden Arbeitsschritte definierten die Skills. Ein entscheidender Aspekt dabei ist, dass der Roboter und der Kraftsensor als separate Komponenten betrachtet werden. Der Kraftsensor erweitert die Fähigkeiten des Roboters zwar und damit dessen Skills, jedoch kann der Roboter auch ohne Kraftsensor betrieben werden. Der Kraftsensor wird als eigene Objektklasse abgebildet, jedoch besitzt dieser keinen eigenen Skill. Folgende Skills wurden definiert, welche den Prozess abdecken.

|  |  |  |
| --- | --- | --- |
| Komponente | Skill | Bemerkung |
| Kamerasystem: | - Bild aufnehmen |  |
| *(Kamera + Vision)* | - Objekt erkennen |  |
|  | - Greifposition ermitteln |  |
| Roboter: | - Position anfahren | Die Zielposition wird angegeben |
|  | - Kontrolliert bewegen | Die Bewegung wird in Echtzeit vorgegeben und mit Sensor überwacht |
| Greifer: | - Backenposition anfahren | Der Skill kann eine bestimmte Position anfahren. Der Greifer kann dadurch geöffnet oder geschlossen werden. |
| *(mit Sensoren)* |

Die definierten Skills werden innerhalb der Umsetzung (Kapitelverweis) detaillierter beschrieben.

**Definierung der Skill-Struktur**

Alle Skills sollten nach einer einheitlichen Struktur aufgebaut sein und lediglich um die prozessspezifischen Funktionen ergänzt werden. Ein zentraler Bestandteil dieser Grundstruktur sind die Schnittstellen, die ein Skill mindestens benötigt, um innerhalb der Systemstruktur funktionsfähig zu sein. Dazu zählen nicht nur Eingangs- und Ausgangsvariablen, sondern auch Eigenschaften und Methoden, da Informationen ebenfalls über diese übertragen werden können. Ein Skill muss über die definierten Schnittstellen (siehe Kapitel XXX) sowohl mit dem System, mit dem Objekt und innerhalb des Prozessmodells interagieren können.

Innerhalb der Entwicklung wurden mehrere Iterationen von Skill-Strukturen geplant, umgesetzt und getestet, um die Best möglichste Struktur zu finden. Innerhalb dieser Dokumentation, werden nicht alle Iterationsschritte erklärt, sondern nur die daraus folgenden Erkenntnisse.

Die Schnittstellen eines Skills wurden in drei Kategorien aufgeteilt, welche das allgemeine Verständnis einfacher machen sollen.

**Steuerungselemente:**

Die Steuerungselemente sind für die Bedienung des Skills angedacht. Hier wird der Skill gestartet, gestoppt oder resettet. Zusätzlich werden auch Informationen über den Zustand des Skills angegeben.

**Betriebselemente:**

Die Betriebselemente sind für den allgemeinen Betrieb des Skills angedacht, welche nicht mit dem spezifischen Prozess zu tun haben. Dies ist zum Beispiel die Schnittstelle zum System, welche eine Aussage über Systemzustand macht.

**Prozesselemente:**

Die Prozesselemente sind spezifische Informationen, welche der Skill für die Ausführung benötigt und dann auch an das Objekt weitergibt.

Zu Beginn wurden die Steuerungselemente auf Basis des PLC-Standards aufgebaut. Der Skill wurde dabei von einer BOOL-Variable (bExecute) gestartet und hat diverse Informationen als Ausgangsvariablen ausgegeben (bDone, bBusy, bLimit, bError und iErrorID). Die Ausgangsvariablen konnten als Transition-Bedingungen für Abläufe verwendet werden. Auf den ersten Blick ist dies eine einfache und sinnvolle Steuerung des Skills. Jedoch zeigte sich bei ersten Versuchen, dass diese Implementierung diverse Probleme mit sich bringen kann. Die Umsetzung eines bExecute-Triggers ist aufwändig und muss gut durchdacht werden, da der Skill nur einmal ausgeführt werden soll. Wenn der Skill beendet wurde und die bExecute-Variable immer noch betätigt ist, soll der Skill nicht ein zweites Mal gestartet werden. Der Trigger muss entsprechend so umgesetzt werden, dass dieser nur auf eine steigende Flanke reagiert. Das Management der Ausgangsvariablen ist geknüpft an die verschiedenen Zustände innerhalb des Skills. Es muss genau bestimmt werden, welcher Zustand einen Einfluss auf die Ausgänge hat. Durch die hohe Anzahl an Ausgangsvariablen, kann man hierbei schnell den Überblick verlieren. Zusätzlich werden viele der Informationen, welche über die Ausgangsvariablen dargestellt werden, auch über den Zustand dargestellt werden. Die Konsequenz daraus ist, dass die Steuerungselemente mit Methoden und Eigenschaften umgesetzt werden.

|  |  |  |  |
| --- | --- | --- | --- |
| Art | Bezeichnung | Typ | Beschreibung |
| Methode | M\_Start | BOOL | Methode zum Starten des Skills |
| Methode | M\_Stop | BOOL | Methode zum Stoppen des Skills |
| Methode | M\_Reset | BOOL | Methode zum Resetten des Skills |
| Eigenschaft | P\_State (GET) | eSkillState | Eigenschaft zum Abfragen des aktuellen Zustandes |

Die Eigenschaft wird mit einem benutzerdefinieren Datentyp umgesetzt, welche die Zustände des Skills abbildet. Somit ist immer klar, in welchem Zustand sich der Skill im Moment befindet.

|  |  |
| --- | --- |
| Listen-Nummer | eSkillState |
| 0 | *BEREIT* |
| 1 | *LAUFEND* |
| 2 | *ABGESCHLOSSEN* |
| 3 | *FEHLER* |
| 4 | *LIMITE* |
| 5 | *ERREICHT* |

Auch bei den Betriebselementen gab es im Verlauf der Entwicklung Veränderungen. Zu Beginn enthielten diese alle Variablen, die für den Betrieb des Skills erforderlich waren, und dienten als Schnittstelle zwischen System und Objekt. Relevante Informationen für den Prozess wurden dabei über Ausgangsvariablen an das Objekt übergeben, welches diese wiederum als Eingangsvariablen entgegennahm.

Wenn jedoch zwei Skills auf dasselbe Objekt zugreifen sollen (nicht gleichzeitig) und beide mit dessen Eingangsvariablen verbunden sind, überschreibt stets einer der beiden Skills den Wert des anderen. Da es durchaus vorkommt, dass mehrere Skills mit einem Objekt interagieren (ebenfalls nicht gleichzeitig), muss die Interaktion zwischen Skill und Objekt so gestaltet sein, dass sie unabhängig von anderen Skills funktioniert.

Dies lässt sich durch die Instanziierung eines Interfaces erreichen. In der objektorientierten Programmierung wird ein Interface genutzt, um vorzugeben, welche Methoden und Eigenschaften ein Funktionsbaustein zwingend besitzen muss. Durch das Instanziieren eines Interfaces innerhalb eines Funktionsbausteins können dessen Methoden und Eigenschaften verwendet werden. Wenn das Interface als Eingangsvariable instanziiert wird, kann es mit einem Objekt verbunden werden, das dasselbe Interface implementiert. Beim Ausführen einer Methode innerhalb des Funktionsbausteins wird dadurch die entsprechende Methode im verknüpften Objekt aufgerufen. Gleiches gilt für die Eigenschaften. Entsprechend kann auch der Zustand des Objektes über diese Schnittstelle abgefragt werden.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Somit wurden die Betriebselemente wie folgt definiert:

|  |  |  |  |
| --- | --- | --- | --- |
| Art | Bezeichnung | Typ | Beschreibung |
| Eingangsvariabel | eSysCommand | eSystemCommand | Befehlsvariabel von System zu Skill |
| Eingangsvariabel | eSysState | eSystemStatus | Informationen über System (Systemparameter) |
| Eingangsvariabel | fbObjekt | Objekt Interface | Verknüpfung zu Objekt |
| Ausgangsvariabel | iErrorID | INT | Information um welchen Fehler es sich handelt |

Da auch die Übergabe der Objektparameter an das Objekt über die Interface-Verknüpfung erfolgt (mittels einer Eigenschaft), wird für jede Art von Objekttyp ein eigenes Interface definiert. Die Steuerungselemente für ein Objekt sind jedoch für alle Objekttypen einheitlich, ähnlich wie bei den Skills. Dazu gehören die Methoden M\_Start, M\_Stop, M\_Rest sowie die Eigenschaft P\_State (ObjectState).

Die folgenden benutzerdefinierten Datentypen sind für die Betriebselemente von Bedeutung:

|  |  |  |  |
| --- | --- | --- | --- |
| Listen-Nummer | eSystemCommand | ObjectState | eSystemState |
| 0 | *KEINE* | *AUS* | *AUS* |
| 1 | *AUSSCHALTEN* | *BEREIT* | *BEREIT* |
| 2 | *EINSCHALTEN* | *MANUELL* | *LAUFEND* |
| 3 | *STOPPEN* | *LAUFEND* | *GESTOPPT* |
| 4 | *RESETTEN* | *ABGESCHLOSSEN\_INTERN* | *FEHLER* |
| 5 | */* | *ABGESCHLOSSEN\_EXTERN* | */* |
| 6 | */* | *GESTOPPT* | */* |
| 7 | */* | *FEHLER* | */* |

Als letzte Kategorie umfasst ein Skill die Prozesselemente. Die Prozesseigenschaften dienen dazu, dem Skill prozessrelevante Informationen bereitzustellen. Die vorhandenen Prozessmethoden werden vom Skill intern genutzt, beispielsweise zur Datenverarbeitung oder -auswertung.

Ähnlich wie bei einem Objekt wird auch ein Skill mithilfe eines Interfaces aufgebaut, das die zugehörigen Methoden und Eigenschaften definiert. Über die Interface-Verknüpfung lässt sich ein Skill zudem einfach in eine Sequenz oder einen Arbeitsplan integrieren.

Die Grundstruktur eines Skills lässt sich anhand des folgenden Schemas zusammenfassen. Dieses Schema veranschaulicht auch die Interaktion des Skills über die verschiedenen Schnittstellen im System.

Ein Bild, das Text, Screenshot, parallel, Diagramm enthält.

Automatisch generierte Beschreibung

**Konfiguration eines Skills**

Nicht jeder Skill benötigt alle Zustände. Damit der Skill so übersichtlich wie möglich bleit, sollen auch nur die Zustände verwendet werden, welche benötigt werden. Ein Skill kann drei Konfigurationen einnehmen.

**Ein Bild, das Text, Screenshot, Diagramm, Plan enthält.

Automatisch generierte Beschreibung**

|  |  |
| --- | --- |
| **Konfiguration 1:** | Stellt die Grundkonfiguration dar. Jeder Skill muss diese Zustände besitzen. Hierbei handelt es sich um Skills, welche nur durch das Objekt abgeschlossen werden, z.B. eine einfache Punkt-Zu-Punkt-Bewegung des Roboters. |
| **Konfiguration 2:** | Bei dieser Konfiguration kommt der LIMIT-Zustand dazu. Dieser wird benötigt, wenn es eine Limit-Bedingung gibt, z.B. einen Grenzwert für die Kraft oder eine maximale Zeitdauer. |
| **Konfiguration 3:** | Bei der letzten Konfiguration wird der ERREICHT-Zustand ergänzt. Dieser gibt an, ob ein definiertes Ziel erreicht wurde, dies kann z.B. eine Kraft sein. |

**Beispiel der Interaktion zwischen Skill und Objekt**

Anhand eines Beispiels soll die Interaktion zwischen einem Skill und den zugehörigen Objekten erläutert werden. Der Skill **«**Skill\_P2P» ermöglicht es einem Roboter, zu einem definierten Punkt zu fahren. Falls der Roboter dabei mit einem Hindernis kollidiert, wird der Roboter durch den Skill gestoppt. Hindernisse werden dabei mithilfe eines Kraftsensors erkannt.

Der Skill wird über Eingangsvariablen mit den entsprechenden Objekten verbunden, was durch eine blaue Verbindungslinie dargestellt wird. Über die Sequenz werden verschiedene Informationen an den Skill übergeben. Dazu zählen:

* Die aktuelle Position des Roboters,
* Seine Geschwindigkeit und Beschleunigung,
* Die Art der Bewegung bzw. Positionierung,
* Sowie die Kraftgrenze, die ein Hindernis definiert.

Der Start des Skills erfolgt durch die Start-Methode, die von der Sequenz ausgelöst wird. Der Skill selbst initiiert über die Start-Methode des Aktor-Objekts den Bewegungsprozess und liest parallel dazu mithilfe der Messdaten-Eigenschaft die Daten des Sensor-Objekts aus. Die Objekte haben dabei ausschliesslich der Aufgabe, ihre spezifischen Funktionen auszuführen, die durch ihre jeweilige Komponente definiert sind. Sie enthalten keine Logik in Bezug auf das Gesamtsystem – diese liegt vollständig im Skill.

Der Skill analysiert die übermittelten Daten und reagiert entsprechend darauf. Im geschilderten Beispiel stoppt der Skill den Aktor, sobald die gemessenen Daten die vorgegebenen Schwellenwerte überschreiten.

Ein Bild, das Text, Screenshot, Schrift, parallel enthält.

Automatisch generierte Beschreibung

**Umsetzung in TwinCAT**

**Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte BeschreibungAllgemeine Struktur**

Die Programmierung der Skills soll so einfach und übersichtlich gehalten werden wie möglich. Dafür mit den bereits angesprochenen Interfaces gearbeitet, aber auch mit dem Vererben von Funktionsbausteinen. Über eine Vererbung können Methoden und Eigenschaften (inkl. Ihrer Funktionalität), sowie Variablendeklaration (Eingangs-, Ausgangs- und Lokalvariablen) von einem Funktionsbaustein übernommen werden. Dadurch kann ein Basis-Funktionsbaustein erstellt werden, welcher Methoden, Eigenschaften und Variablen zur Verfügung stellt, welche für alle Skills verwendet werden. Diese Elemente werden bei jeder Vererbung separat zur Verfügung gestellt. Mehrere Skills greifen nicht auf die selben Elementen zu und beeinflussen sich somit nicht.

Da jedoch die Steuerungselemente auf das instanziierte Objekt zugreifen sollen (Skizze), können diese nicht in einem Basis-Funktionsblock abgebildet und vererbt werden. Für die Objekte gibt es verschiedene Interfaces, je nach benötigten Daten für den Prozess. Entsprechend werden in unterschiedlichen Skills auch unterschiedliche Objekt-Interfaces benötigt. Dadurch kann das Objekt nicht bereits im Basis-Funktionsbaustein instanziiert werden und die Methoden und Eigenschaften können nicht darauf zugreifen. Aus diesem Grund, müssen «M\_Start», «M\_Stop», «M\_Reset» und «P\_State» im Skill selber implementiert werden. Die Vererbung wird in diesem Fall nur für Variablen genutzt, welche für alle Skills gleich bleiben.

In TwinCat wird eine Vererbung über den Zusatz «EXTENDS» durchgeführt. Ein Interface kann mit dem Zusatz «IMPLEMENTS» zugewiesen werden.

Für den Basis-Funktionsbaustein «FB\_Basis\_Skill» wurden die Variablen wie folgt definiert:

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

**Aufbau des Grund-Skills**

Alle Skills sollten nach einem einheitlichen Grundmuster aufgebaut werden. Dabei wird ein Skill als Funktionsblock mit strukturiertem Text realisiert. Die Verwendung von strukturiertem Text bietet hohe Flexibilität bei der Umsetzung.

Jeder Skill hat folgende drei Steuerungsmethoden:

|  |  |  |
| --- | --- | --- |
| M\_Start | M\_Stop | M\_Reset |
|  |  |  |

Innerhalb der Methode M\_Start wird in einem ersten Schritt geprüft, ob sich das System im korrekten Zustand befindet. Danach wird geprüft, dass sich der Skill im korrekte Zustand befindet. Da die Methode von aussen (zum Beispiel durch die Sequenz) ausgelöst wird, darf der Skill nur im BEREIT-Zustand bestartet werden. Als letzter Schritt werden die Parameter überprüft, falls diese Auflagen erfüllen müssen. Falls alle Bedingungen erfüllt werden, werden die Prozessparameter (in diesem Fall wird eine Roboterposition) übergeben und die Start-Methode des Objektes wird ausgelöst. Die Stop- und Reset-Methode lösen die entsprechenden Methoden beim Objekt aus.

Der Aufbau eines Skills gliedert sich in drei Hauptbereiche: Fehlerüberwachung, Aufruf interner Steuerungs-methoden und Zustandsmanagement. Im Bereich des Aufrufs interner Steuerungsmethoden wird überprüft, ob das System vorgibt, den Skill zu stoppen oder zurückzusetzen. In diesen Fällen wird entweder die Methode M\_Stop oder M\_Reset ausgelöst. Zudem wird geprüft, ob der Skill ein definiertes Limit oder einen Zielwert erreicht. Auch hier führt dies zur Ausführung der Methode M\_Stop.

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

Unter Zustandsmanagement werden die, in Kapitel XXX definierten, Zustände des Skills definiert und umgesetzt. Für die Umsetzung wurde mit einer CASE-Anwendung gearbeitet. Die meisten Zustände sind für alle Skills identisch, jedoch beim Zustand «LAUFEND» gibt es kleinere Unterschiede.

|  |  |
| --- | --- |
| Zustand 0 – BEREIT: | Innerhalb des Zustandes prüft der Skill den Objektzustand. Falls das Objekt durch M\_Start gestartet wurde und sich nun im Zustand LAUFEND befindet, so wechselt auch der Skill auf den Zustand LAUFEND. Der Skill darf jedoch nur dann auf den Objektzustand reagieren, falls der Skill den Start des Objektes auch ausgelöst hat, da unterumständen mehrere Skills auf ein Objekt zugreifen. |

|  |  |
| --- | --- |
| Zustand 1 – LAUFEND: | Der Skill kann auf verschiedene Arten gestoppt werden, welche innerhalb des Zustandes überprüft werden. Das System kann den Skill stoppen, welcher anschliessend direkt in den Zustand BEREIT wechselt. Falls der Objektzustand meldet, dass das Objekt den Prozess abgeschlossen hat *(zum Beispiel bei einer Punkt-zu-Punkt-Bewegung eines Roboters)*, wechselt der Skill in den Zustand ABGESCHLOSSEN. Falls eine Limit- oder Erreicht-Bedingung erfüllt wurde, wird das Objekt durch den Skill gestoppt und wechselt in den entsprechenden Zustand.  Während der Zustands LAUFEND können aber auch interne Methoden aufgerufen werden *(zum Beispiel für das Auswerten von Messwerten)* |

|  |  |
| --- | --- |
| Zustand 2 – ABGESCHLOSSEN: | Der Zustand gibt den Befehl zum resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand BEREIT befindet, kehrt auch der Skill in den Zustand BEREIT zurück. |

|  |  |
| --- | --- |
| Zustand 3 – FEHLER | Im Fehlerzustand wartet der Skill, dass dieser zur das System resettet wird. |

|  |  |
| --- | --- |
| Zustand 4 – LIMIT | Der Zustand gibt den Befehl zum resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand BEREIT befindet, kehrt auch der Skill in den Zustand BEREIT zurück. |

|  |  |
| --- | --- |
| Zustand 5 – ERREICHT | Der Zustand gibt den Befehl zum resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand BEREIT befindet, kehrt auch der Skill in den Zustand BEREIT zurück. |

**Erarbeitete Skills**

Zum Stand der Dokumentation wurden drei Skills umgesetzt, welche sich auf den Roboter (2 Skills) und den Greifer (1 Skill) beziehen.

Bemerkung:

Aus Zeitgründen wurde während der Erarbeitung entschieden, dass das Kamerasystem noch nicht in den Prozess integriert wird. Der Schwerpunkt dieser Arbeit liegt auf der Software-Struktur und wie die verschiedenen Elemente miteinander interagieren. Auch ohne Kamerasystem konnte die Struktur definiert, aufgebaut und getestet werden. Die Einarbeitung in das Vision-Tool von TwinCat ist ein zeitintensiver Prozess, mit wenig Mehrwert für das Ziel dieser Arbeit. Im Rahmen einer weiterführender Arbeit, wäre die Integration des Kamerasystem aber ein relevanter Punkt.

Skill 1 (Roboter): Skill\_Position\_Anfahren

Mit dem Skill kann eine Position vom Roboter angefahren werden. Die Bewegung wird selbständig vom Objekt abgeschlossen. Es ist nicht vorgesehen, dass der Skill die Bewegung stoppt. Damit der Skill ausgeführt werden kann, benötigt er folgende Prozessparameter, welche als Eigenschaften (SET) dem Skill übergeben werden:

|  |  |  |
| --- | --- | --- |
| Bezeichnung | Datentyp | Beschreibung |
| P\_RobotPosition | Array[0..5] of LREAL | Gibt die kartesischen Koordinaten oder Gelenkwinkel an, je nachdem welcher Bewegungstyp ausgewählt wurde. |
| P\_RobotSpeed | LREAL | Gibt an wie schnell sich der Roboter bewegen soll. |
| P\_RobotAcceleration | LREAL | Gibt an mit welcher Beschleunigung der Roboter arbeiten soll. |
| P\_RobotMoveTyp | eRobPosTyp | Gibt den Bewegungstyp des Roboters an. |
| P\_RobotPosTyp | eRobMoveTyp | Gibt den Positionstyp der angegebenen Position an. |

Für den Bewegungstyp und Positionstyp wurden benutzerdefinierte Listen-Datentypen angelegt. Eine detaillierte Erklärung ist bei der Beschreibung des Roboter-Objektes (Kapitel XXX) vorhanden. Diese sind wie folgt definiert:

|  |  |  |
| --- | --- | --- |
| Listen-Nummer | eRobPosTyp | eRobMoveTyp |
| 0 | *Absolut* | *LinearToolSpace* |
| 1 | *Relativ* | *LinearJointSpace* |
| 2 | */* | *CircularToolSpace* |
| 3 | */* | *BlendCircularMoveLinear* |

Skill 2 (Roboter): Skill\_Position\_Anfahren\_Erweitert

Ist ähnlich wie Skill 1, jedoch kann der Skill noch auf eine Kraft reagieren. Es ist möglich einen Grenzwert für die Kraft und deren Richtung anzugeben. Der Skill stoppt das Objekt und somit die Bewegung, falls dieser Grenzwert überschritten wurde. Damit ist es möglich Hindernisse zu erkennen, um zum Beispiel einen Anschlag zu definieren. Damit der Skill ausgeführt werden kann, benötigt er folgende Prozessparameter, welche als Eigenschaften (SET) dem Skill übergeben werden:

|  |  |  |
| --- | --- | --- |
| Bezeichnung | Datentyp | Beschreibung |
| P\_RobotPosition | Array [0..5] of LREAL | Gibt die kartesischen Koordinaten oder Gelenkwinkel an, je nachdem welcher Bewegungstyp ausgewählt wurde. |
| P\_RobotSpeed | LREAL | Gibt an wie schnell sich der Roboter bewegen soll. |
| P\_RobotAcceleration | LREAL | Gibt an mit welcher Beschleunigung der Roboter arbeiten soll. |
| P\_RobotMoveTyp | eRobPosTyp | Gibt den Bewegungstyp des Roboters an. |
| P\_RobotPosTyp | eRobMoveTyp | Gibt den Positionstyp der angegebenen Position an. |
| P\_Erreicht |  | Gibt den Grenzwert für die entsprechenden Richtungen an |

Skill 3 (Greifer): Skill\_GreiferPos\_Anfahren

Der letzte Skill bezieht sich auf den Greifer. Der Skill kann in Verbindung mit einem Greifer wegwendet werden, welche eine definierte Position anfahren kann. Die Bewegung wird selbständig vom Objekt abgeschlossen. Damit der Skill ausgeführt werden kann, benötigt er folgende Prozessparameter, welche als Eigenschaften (SET) dem Skill übergeben werden:

|  |  |  |
| --- | --- | --- |
| Bezeichnung | Datentyp | Beschreibung |
| P\_GreifBreite | LREAL | Gibt an welche Breite die Greifbacken anfahren sollen. |
| P\_GreifGeschwindigkeit | LREAL | Gibt an wie schnell sich die Greifbacken bewegen sollen. |
| P\_GreifKraft | LREAL | Gibt an mit welcher Kraft die Greifbacken zudrücken sollen. |

**Momentaner Stand**

Die Skills wurden derzeit so entwickelt, dass die definierte Anwendung erfolgreich umgesetzt werden kann. Dabei lag der Fokus nicht darauf, bereits jede mögliche Situation abzudecken. Viel Zeit und Aufwand wurden in die Konzeption und das Testen der Skill-Struktur investiert. Diese Struktur bildet das stabile und zuverlässige Fundament, auf dem die Skills aufbauen. Sobald dieses Fundament steht, können die Funktionen des Skills schrittweise erweitert und optimiert werden.

Als potenzielle Erweiterung könnten zusätzliche Einstellungsmöglichkeiten integriert werden. Beim Greifer könnte beispielsweise ein Offset angegeben werden, der die Dicke der Backenaufsätze berücksichtigt. Dadurch ließe sich die angefahrene Breite der Greifbacken an unterschiedliche Aufsätze anpassen.