Arbeitspaket 6: Anlagenmodell

Die Objekte stellen die Komponenten in der Anlage dar und sollen die Funktionalitäten dieser abbilden. Ein Objekt wird immer nur für sich selbst betrachtet und besitzt möglichst keine Abhängigkeiten von anderen Objekten *(Kann in gewisse Situation durchaus Sinn machen)*. Man baut alle Komponenten der Anlage aus und betrachte die Fähigkeiten dieser. Eine erstelle Objektklasse kann anschliessend für die Instanziierung eines Objektes verwendet werden. Dies ermöglicht das Erstellen von mehreren Objekten auf Basis einer Objektklasse.

**Definierung der Objekt-Struktur**

Wie bereits für die Skills wird auch für die Objekte eine Grundstruktur festgelegt, die als Basis für den Aufbau aller Objekte dient. Diese einheitliche Struktur erleichtert die Standardisierung der Interaktionen innerhalb der Software. Dabei sollen die Objektklassen möglichst objektorientiert gestaltet werden, sodass die Funktionalität der Objekte in klar abgegrenzten Methoden abgebildet wird. Zum Ausführen einer Funktionalität reicht es daher, die entsprechende Methode aufzurufen.

Die Objekte wurden zusammen mit dem Skills iterativ erarbeitet. Änderungen bei der Skill-Struktur hatten auch einen Einfluss auf die Objekt-Struktur.

Die Schnittstellen eines Objektes wurden in 4 Kategorien aufgeteilt, welche ähnlich wie beim Skill definiert wurden und entsprechend das allgemeine Verständnis einfacher machen sollen.

**Objektvariablen:**

Die Objektvariablen umfassen die Eingangsvariablen (Objektparameter) und die Ausgangsvariablen (Anlagenparameter), welche für den Betrieb des Objektes benötigt werden. Dabei kann es sich zum Beispiel um eine IP-Adresse handeln, wenn das Objekt über eine TCP/IP-Schnittstelle angesprochen wird. Falls das Objekt direkt mit E/A-Klemmen interagieren muss, kann dies über die Anlagenparameter gemacht werden.

**Steuerungselemente:**

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

**Betriebselemente:**

Die Betriebselemente sind für den allgemeinen Betrieb des Objekts 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 das Objekt für die Ausführung benötigt.

Die Steuerungs-, Betriebs- und Prozesselemente erfüllen somit die gleichen Aufgaben wie beim Skill. Zwischen der Struktur von Objekt und Skill gibt es somit Parallelen, welche die Verständlich- und Übersichtlichkeit des Gesamtsoftware verbessern. Die Steuerungselemente umfassen die Gleichen Methoden und Eigenschaften wie die Skills *(Umsetzung dieser unterscheidet sich jedoch)*.

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

Die Eigenschaft wird mit einem benutzerdefinieren Datentyp umgesetzt, welche die Zustände des Objektes abbildet. Somit ist immer klar, in welchem Zustand sich das Objekt im Moment befindet. Dabei unterscheidet man zwischen Aktor und Sensor.

|  |  |  |
| --- | --- | --- |
| Listen-Nummer | eObjectAktorState | eObjectSensorState |
| 0 | *AUS* | *AUS* |
| 1 | *BEREIT* | *LAUFEND* |
| 2 | *MANUELL* | *FEHLER* |
| 3 | *LAUFEND* | */* |
| 4 | *ABGESCHLOSSEN\_INTERN* | */* |
| 5 | *ABGESCHLOSSEN\_EXTERN* | */* |
| 6 | *GESTOPPT* | */* |
| 7 | *FEHLER* | */* |

In Kapitel XXX (Struktur) wurden die Zustände eines Objekts definiert. Dies stellte einen wichtigen Schritt dar, um die allgemeine Interaktion zwischen System, Skill und Objekt festzulegen. Basierend auf diesen Zuständen wurden die finalen Zustände eines Objekts spezifiziert.

Ein Aktor-Objekt erweitert diese Zustände um einen zusätzlichen Zustand für den manuellen Betrieb. Ein Sensor-Objekt hingegen kann deutlich einfacher gestaltet werden: Sobald das System eingeschaltet wird, ist der Sensor aktiv. Dieser Zustand wird nur durch das Ausschalten des Systems oder das Auftreten eines Fehlers verlassen.

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

Automatisch generierte Beschreibung

Auch in der Struktur unterscheiden sich Aktor- und Sensor-Objekte geringfügig. Sensor-Objekte verfügen über keine Steuerungsmethoden wie Start, Stop oder Reset. Stattdessen werden diese Funktionen vom System über die Betriebsvariablen gesteuert.

Beide Objekt-Typen verwenden jedoch dieselben Betriebsvariablen:

|  |  |  |  |
| --- | --- | --- | --- |
| Art | Bezeichnung | Typ | Beschreibung |
| Eingangsvariabel | eSysCommand | eSystemCommand | Befehlsvariabel von System zu Objekt |
| Eingangsvariabel | eSysState | eSystemStatus | Informationen über System (Systemparameter) |
| Eingangsvariabel | iMode | INT | Momentaner Modus von Objekt (Manuell / Automatik) |
| Ausgangsvariabel | iErrorID | INT | Information um welchen Fehler es sich handelt |

Wie in Kapitel XXX beschrieben, verfügt ein Objekt über ein Interface, das die Steuerungselemente und Prozesseigenschaften definiert. Dieses Interface hängt von den erforderlichen Prozessparametern bzw. Prozessinformationen sowie davon ab, ob es sich um einen Aktor oder einen Sensor handelt.

Die Prozessinformationen umfassen beispielsweise Messwerte eines Sensors oder die aktuelle Position eines Roboters.

Die Grundstruktur eines Objekts wird im folgenden Schema dargestellt. Es zeigt übersichtlich, wie das Objekt über die verschiedenen Schnittstellen im System interagiert.

|  |  |
| --- | --- |
|  |  |

**Umsetzung in TwinCat**

**Allgemeine Struktur**

Die Programmierung der Objekte soll, ähnlich wie bei den Skills, möglichst einfach und übersichtlich gestaltet sein. Die Grundstruktur ist dabei identisch zu der der Skills. Ein Interface legt fest, welche Methoden und Eigenschaften ein Objekt zwingend besitzen muss.

Die gemeinsamen Steuerungselemente und Betriebsvariablen werden durch Vererbung implementiert, sodass sie für alle Objekte einheitlich verfügbar sind. Unterschiede ergeben sich zwischen Aktoren und Sensoren: Während ein Aktor in der Regel umfangreicher definiert werden muss, kann ein Sensor deutlich einfacher und schlanker gestaltet werden.

**Ein Bild, das Text, Screenshot, Design enthält.

Automatisch generierte Beschreibung**

Im Basis-Funktionsbaustein «FB\_Basis\_Objekt\_Aktor» wurden die Betriebs-, Management- und Zustandsvariablen festgelegt. Auch hier zeigt sich die Parallele zum Aufbau des Skill-Basis-Funktionsbausteins, wodurch eine einheitliche Struktur gewährleistet wird.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Der Basis-Funktionsbaustein «FB\_Basis\_Objekt\_Sensor» verwendet dieselben Variablen wie der Aktor-Baustein. Allerdings wird für die Variable iState der benutzerdefinierte Datentyp «eObjectSensorState» verwendet.

Da ein Sensor im Vergleich zu einem Aktor weniger Zustände besitzt, sind bei den Zustandsvariablen deutlich weniger Transitionen abzubilden.

Der Basis-Funktionsbaustein des Aktor-Objekt implementiert auch die Steuerungsmethoden der Objekte. Die Methoden wurden wie folgt umgesetzt:

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

Die Methode M\_Start überprüft zunächst den Status des Systems und anschliessend den Status des Objekts. Beide müssen sich im BEREIT-Zustand befinden, damit die Start-Variable aktiviert werden kann. Die Aktivierung der Variablen erfolgt nur, wenn sowohl das System als auch das Objekt die erforderlichen Voraussetzungen erfüllen.

Die Stop-Methode kann ausschliesslich im LAUFEND-Zustand des Objekts ausgeführt werden. Für die Reset-Methode gibt es hingegen keine Bedingungen; sie kann unabhängig vom aktuellen Zustand des Systems oder des Objekts verwendet werden.

**Definierung des Grund-Objektes**

Das Grundobjekt definiert den Aufbau des Objektes, nachdem sich alle Objekte richten. Der Aufbau gliedert sich in fünf Hauptbereiche: Instanziierungen, Input-Command-Verwaltung, Interner Methodenaufruf, Datenerfassung und Zustandsmanagement.

Instanziierungen:

Der Bereich wird genutzt, um notwendige, im Objekt-Funktionsbaustein instanziierte, TwinCat-Funktionsbausteine aufzurufen. Diese werden nicht in Methoden aufgerufen, da es dabei zu Problemen kommen kann. Während der Entwicklung wurde die Erfahrung gemacht, dass beim Arbeiten mit Kommunikationsbausteinen für TCP/IP-Schnittstellen in einer Methode, Fehler auftreten können. Die Kommunikation ist nicht robust und es kann zu Fehlern kommen. Wenn der Kommunikationsbaustein aber im Objektbaustein selbst aufgerufen wird und die Methode nur auf das Eingangssignal einwirkt, ist die Funktionalität stabiler und reproduzierbarer. Bei der detaillierten Beschreibung der umgesetzten Objekte wird gezeigt, wie Funktionsbausteine aufgerufen werden.

Input-Command-Verwaltung:

Die Input-Command-Verwaltung analysiert über die Variabel «eSysCommand», ob das System einen Befehl vorgibt. Das System kann das Objekt einschalten, ausschalten, stoppen und resetten. Gewisse Befehle können nur ausgeführt werden, wenn sich das Objekt in einem bestimmen Zustand befindet.

Interner Methodenaufruf:

Wie der Name beschreibt, werden Methoden aufgerufen, welche nicht von aussen ausgeführt werden. Dabei kann es sich um Methoden handeln, welche zum Beispiel ausgeführt werden müssen, dass das Objekt eingeschalten wird und von «AUS» zu «BEREIT» wechseln kann.

Datenerfassung:

Der Bereich Datenerfassung wird der Prozess durchgeführt, welcher für die Erfassung von notwenigen Daten notwendig ist. Dazu gehört der Aufbau der Verbindung, das Auslesen von Daten und das Verarbeiten der Daten, so dass das Objekt damit arbeiten kann. Dies ist erforderlich, da manche Objekte bestimmte Informationen für Funktionen benötigen. Damit zum Beispiel ein Roboter-Objekt beurteilen kann, ob die Bewegung abgeschlossen wurde und dadurch in den Zustand «ABGESCHLOSSEN\_INTERN» wechseln kann, muss die aktuelle Position mit der gewünschten Position verglichen werden. Die aktuelle Position wird hierbei unter Datenerfassung erfasst. Es geht grundsätzlich um Daten, die während dem Betrieb des Objektes ausgewertet werden müssen.

Zustandsmanagement:

Wie bei den Skills werden unter Zustandsmanagement die Zustände des Objektes umgesetzt. Für die Umsetzung wurde auch hier mit einer CASE-Anwendung gearbeitet.

|  |  |
| --- | --- |
| Zustand 0 – AUS: | Der Zustand prüft nur, ob das Objekt durch das System eingeschalten wird. Sobald die entsprechende Variabel aktiviert ist, wechselt das Objekt in den Zustand «BEREIT» |

|  |  |
| --- | --- |
| Zustand 1 – BEREIT: | Das Objekt kann entweder gestartet, wieder ausgeschalten oder in den Manuell-Modus geschalten werden. Beim Start des Prozesses durch einen Skill, wird die entsprechende interne Methode ausgeführt und der Zustand wechselt zu «LAUFEND». Das Ausschalten kann nur durch das System gemacht werden. |

|  |  |
| --- | --- |
| Zustand 2 – MANUELL: | Der Zustand implementier alle Funktionalität, welche das Objekt hat, so, dass diese manuell verwendet werden können. Dies kann zum Beispiel das manuelle Anfahren von Punkten mit dem Roboter sein. |

|  |  |
| --- | --- |
| Zustand 3 – LAUFEND: | Ein Objekt kann auf 2 Arten gestoppt werden, durch externen Einfluss oder das Objekt beendet den Prozess von selbst. Beides muss im Zustand überwacht werden. Beim Stoppen durch den Skill, wechselt das Objekt in den Zustand «ABGESCHLOSSEN\_EXTERN». Falls das Objekt selbst den Prozess stoppt, wechselt dieses in den Zustand «ABGESCHLOSSEN\_INTERN». Auch das System kann das Objekt stoppen, in diesem Fall wechselt das Objekt in den Zustand «GESTOPPT», da dies ein Stoppen ausserhalb des ordentlichen Prozesses darstellt. Wie der Stop-Prozess innerhalb des Zustandes umgesetzt wird, kann von Objekt zu Objekt unterschiedlich sein. |

|  |  |
| --- | --- |
| Zustand 4 – ABGESCHLOSSEN\_INTERN: | Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT». |

|  |  |
| --- | --- |
| Zustand 5 – ABGESCHLOSSEN\_EXTERN: | Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT». |

|  |  |
| --- | --- |
| Zustand 6 – GESTOPPT: | Im Zustand wird auf den Reset-Befehl des Systems gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT». |

|  |  |
| --- | --- |
| Zustand 7 – FEHLER: | Im Zustand wird auf den Reset-Befehl des Systems gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT». |

**Erarbeitete Objekte**

**Objektklasse für UR5 (FB\_Roboter):**

**Ein Bild, das Text, Karte Menü, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung**Die Objektklasse implementiert das Interface «I\_Objekt\_Roboter», welches die Steuerungselemente des Objektklasse vorgibt. Als Objekt-Eingangsvariablen benötigt die Objektklasse eine IP-Adresse und zwei Port-Angaben. Die genaue Definition dieser Angaben wird in den folgenden Seiten erklärt. Es gibt keine Objekt-Ausgangsvariablen, welche mit EA-Klemmen verbunden werden müssten. Die Verbindung zum Roboter wird über TwinCat-Funktionsbausteine durchgeführt, welche gleichzeitig auch die direkte Schnittstelle darstellen.

Die Steuerungs- und Betriebsvariablen bleiben unverändert zum Basis-Funktionsbaustein.

Als Prozessmethoden wurden M\_Verbinden, M\_Trennen, M\_BewegungZuPosition und M\_BewegungStoppen definiert. Diese Methoden decken die Grundfunktionen der Schnittstelle zum Roboter und dessen Funktionalität ab, um die definierte Anwendung umsetzen zu können. Damit weitere Funktionalitäten abgebildet werden könnten, müssten weitere Methoden hinzugefügt werden. Die Prozesseigenschaften bestehen aus den Prozess-parametern, die für eine Punkt-zu-Punkt-Bewegung des Roboters benötigt werden.

Wie bereits in Kapitel XXX angesprochen, besitzt der UR5 eine TCP/IP-Schnittstelle. Unterschiedliche Ports haben dabei unterschiedliche Funktionen.

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

Automatisch generierte Beschreibung

Bemerkung:

Beim eingesetzten Controller handelt es sich um einen CB2. Diverse Funktionalitäten der Schnittstellen oder zur Verfügung gestellte Dokumentationen richten sich an Controller ab Version CB3. Dies hat während der Entwicklung zu grossen Problemen geführt und viel Zeit in Anspruch genommen, vor allem bezüglich des Real-Time-Interfaces. Dieses wird in dieser Form nicht mehr unterstützt und wurde durch RTDE (Real-time Data Exchange) ersetzt, welche deutlich umfangreichere Funktionen bietet. Mittels Referenzprogrammen und alten Dokumentation konnte aber auch mit dem Real-Time-Interface gearbeitet werden.

Über die Secundary-Schnittstelle (Port 30002) kann dem Roboter Befehle in der UR-eigenen UR-Script-Programmiersprache übergeben werden. Alle möglichen Befehle lassen sich aus der entsprechenden Dokumentation herauslesen (Anhang).

Über die Real-Time-Schnittstelle (Port 30003) kann ich Informationen über den Roboter in Echtzeit erhalten. Dabei erhält man ein Datenpaket von 812 Byte, bei welchem jede Position für eine bestimmte Information steht.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bezeichnung | UR-Datentyp | Datentyp TwinCat | Byte-Grösse | Beschreibung |
| messageSize | Integer | LREAL | 4 | Gesamtlänger der Nachricht in Bytes |
| timestamp | double | LREAL | 8 | Zeitdauer seit Controller gestartet wurde |
| qTarget | double | LREAL | 48 | Zielposition der Gelenke |
| qdTarget | double | LREAL | 48 | Zielgeschwindigkeit der Gelenke |
| qddTarget | double | LREAL | 48 | Zielbeschleunigung der Gelenke |
| iTarget | double | LREAL | 48 | Zielstrom der Gelenke |
| mTarget | double | LREAL | 48 | Zielmoment der Gelenke |
| qActual | double | LREAL | 48 | Aktuelle Position der Gelenke |
| qdActual | double | LREAL | 48 | Aktuelle Geschwindigkeit der Gelenke |
| iActual | double | LREAL | 48 | Aktueller Strom der Gelenke |
| toolAccelerometerValues | double | LREAL | 24 | Tool-Beschleunigungswerte (x, y und z) |
| unused\_1 | / | / | 120 | / |
| tcpForce | double | LREAL | 48 | Kräfte beim TCP |
| toolVector | double | LREAL | 48 | Zielkoordinaten des Tools |
| tcpSpeed | double | LREAL | 48 | Zielgeschwindigkeit des Tools |
| digitalInputBits | double | LREAL | 8 | Aktueller Zustand der digitalen Eingänge |
| unused\_2 | / | / | 120 | / |

Das korrekte Auslesen des Datenpaketes ist für den fehlerfreien Betrieb des Roboters essenziell. Eine falsche Interpretierung der Daten kann zu unerwarteten Verhalten des Roboters führen. Der Schnittstelle muss auch genug Zeit gelassen werden, dass das komplette Datenpaket von 812 Byte verwendet werden kann.

In TwinCat wurde für die Kommunikation mit dem Roboter das Paket TF6310 (TwinCAT 3 | TCP/IP) verwendet. Das Paket stellt verschiedene Funktionsbausteine zur Verfügung, welche für die Kommunikation über TCP/IP benötigt werden. Der UR5 ist dabei der Server und das TwinCat der Client. Folgende vier Bausteine sind relevant für die Kommunikation mit dem Roboter:

|  |  |
| --- | --- |
| FB\_SocketConnect | FB\_SocketClose |
|  |  |
| FB\_SocketSend | FB\_SocketReceive |
|  |  |

Für eine detaillierte Beschreibung des TF6310-Pakets kann entsprechende Dokumentation angeschaut werden.

FB\_SocketConnect ist für die Verbindung mit dem Server zuständig. Folgende zwei Informationen werden für die Verbindung zum UR5 benötigt:

|  |  |  |
| --- | --- | --- |
| sRemoteHoste: | *192.168.1.100* | IP-Adresse des Roboters |
| sRemotePort: | *30002* | Verbindung mit Secundary-Schnittstelle |
|  | *30003* | Verbindung mir Real-Time-Schnittstelle |

Die Variable «sSrvNetId» kann mit einem Leerstring versehen werden, da der TwinCAT TCP/IP Connection Server auf dem lokalen Rechner läuft. Sobald der Funktionsbaustein über «bExecute» gestartet wird und eine Verbindung zum TCP/IP-Server erfolgreich aufgebaut werden konnte, wird ein TCP/IP-Verbindungshandle erstellt. Dieser wird über «hSocket» zur Verfügung gestellt. Über dieses Handle können Daten an einen Socket gesendet oder empfangen werden. Mit FB\_SocketClose kann die Verbindung zum Kommunikationssocket geschlossen werden.

Zum Senden und Empfangen von Daten werden FB\_SocketSend und FB\_SockerReceive verwendet. Bede Funktionsbausteine benötigen den definierten TCP/IP-Verbindungshandle, die Anzahl der zu sendenden Daten in Bytes («cbLen») und die Pointer-Adresse des Sendepuffers («pSrc» / «pDest»).

Die UR5-Objektklasse hat grundsätzlich folgenden Aufgaben:

* Aufbau und Verwaltung der Verbindung zum UR5-Roboter über TCP/IP
* Entgegennahme und Vorbereitung der Prozessdaten
* Senden des entsprechenden Befehls an den Roboter, damit eine Bewegung durchgeführt wird
* Senden des entsprechenden Befehls an den Roboter, dass eine Bewegung gestoppt werden kann
* Entgegennahme, Verarbeitung und Auswertung der vom Roboter gesendeten Daten

Für den Aufbau und Verwaltung der Kommunikation, werden alle relevanten Funktionsbausteine im Bereich «Instanziierungen» der Objektklasse aufgerufen und mit den allgemeinen Variablen verknüpft. Variablen wie die zu senden Daten werden erst in der entsprechenden Methode zugewiesen.

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

Automatisch generierte Beschreibung

Über die Methoden «M\_Verbinden» und «M\_Trennen» werden die entsprechenden Trigger der Funktionsbausteine aktiviert und dieses auszulösen.

Die Prozessdaten, welche durch den Skill zur Verfügung gestellt werden, werden über die Eigenschaft «P\_Prozessparameter» an das Objekt übergeben. Die Daten werden mittels einer benutzerdefinierten Struktur, mit folgender Definition, übergeben:

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Die Prozessdaten werden innerhalb der Methode «M\_BewegungZuPosition» analysiert, in die korrekte Form gebracht und versendet. Die Methode prüft in einem ersten Schritt, um welche Positionsart es sich handelt. Es kann zwischen absolut und relativ unterschieden werden. Je nach Positionsart werden die Koordinaten anders ausgelegt. Der nächste Schritt ist die Bestimmung der Bewegung. Jede Bewegungsart hat einen definierten Befehl in der URScript-Porgrammiersprache. Am Ende kann der Befehl-String zusammengesetzt werden. Ein Befehl-String besteht in der Regel aus dem entsprechenden Befehls-Wort, der Position, der Geschwindigkeit und Beschleunigung.

Beispiel für einen solchen Befehl-String:

|  |  |  |
| --- | --- | --- |
| *movel(p[x,y,z,rx,ry,rz], a, v)* | *movel :* | Lineare Bewegung im kartesischen Koordinatensystem |
|  | *P[x,y,z,rx,ry,rz] :* | Informationen über anzufahrenden Punkt [m & rad] |
| *a :* | Beschleunigung [m/s^2] |
| *v :* | Geschwindigkeit [m/s] |

Wichtig:

Jeder URSript-Befehl muss mit «\n» beendet werden. Der String kann aber nicht einfach mit diesen Angaben ergänzt werden. Bei der Umwandlung von einem String in einen Byte-Array wird diese Endung «falsch» übersetzt (mit dem Wert 42). Für die korrekte Interpretation des Befehls durch den Kontroller muss am Ende aber eine 10 stehen. Diese muss dem Byte-Array angehängt werden. Wahrscheinlich ist dies auf unterschiedliche Interpretation von Low- und High-Byte zurückzuführen (die Reihenfolge der Bytes wird dabei vertauscht und führt zu falschen Resultaten). Dies hat zu diversen Problemen geführt in Zusammenhang mit der TCP/IP-Schnittstelle.

In folgendem Code-Snippet wird das Byte-Array mit dem Wert 10 ergänzt und anschliessend über den TCP/IP-Funktionsbaustein «FB\_SocketSend» an den Server verwendet.

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

Automatisch generierte Beschreibung

Mit der Methode M\_BewegungStoppen wird grundsätzlich dasselbe gemacht. Nur der Befehl-String ändert sich auf den entsprechenden Befehl.

Im Bereich «Datenerfassung» wird kontinuierlich der aktuelle Wert der Gelenkposition und Geschwindigkeit erfasst, wie auch die TCP-Position. Die Erfassung arbeitet dabei fortlaufend vier Schritte ab:

Schritt 1: Verbindungsaufbau zu Real-Time-Schnittstelle des Roboters

Schritt 2: Empfangen des Datenpakets (812 Bytes)

Schritt 3: Verarbeitung der Rohdaten

Schritt 4: Trennung der Verbindung zur Real-Time-Schnittstelle

Erfahrungen haben gezeigt, dass die Schnittstelle immer wieder geschlossen und neu geöffnet werden muss, damit Daten verlässlich ausgelesen werden konnten. Wird dies nicht gemacht, so werden die Daten nur das erste Mal empfangen und behalten anschliessend diese Werte.

Ein Positions- oder Geschwindigkeitswert setzen sich jeweils aus 8 Bytes zusammen. Diese können über eine UNION-Datenstruktur zu einem LREAL-Wert zusammengefügt werden. Da TwinCat die Low- und High-Bytes anders interpretiert als der Roboter-Kontroller, muss die Reihenfolge des Arrays in einem ersten Schritt umgedreht werden. Anschliessend können diese mittels einer UNION-Datenstruktur umgewandelt werden.

Diese Werte werden im Zustand «LAUFEND» verwendet, um zu prüfen, ob der Roboter die gewünschte Position bereits erreicht hat. Falls die Differenz zwischen der aktuellen und gewünschten Position einen definierten Grenzwert unterschreitet, gilt die Position als erreicht.

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

Automatisch generierte Beschreibung

Beim Stoppen der Bewegung kann über die aktuelle Geschwindigkeit ermittelt werden, ob der Roboter angehalten hat und somit die Bewegung abgeschlossen wurde.

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

Automatisch generierte BeschreibungObjektklasse für HEX-E (FB\_Kraftsensor):**

Die Objektklasse implementiert das Interface «I\_Objekt\_Kraftsensor» und gehört zur Kategorie der Sensor-Objekte. Daher enthält sie keine Steuerungsmethoden, sondern lediglich eine Steuerungseigenschaft. Für den Betrieb der Objektklasse sind eine IP-Adresse und eine Port-Nummer als Eingabevariablen erforderlich. Analog zur Roboterobjektklasse werden keine Ausgangsvariablen definiert. Die Kommunikation mit dem Sensor erfolgt ebenfalls über TwinCat-Funktionsbausteine.

Die Steuerungs- und Betriebsvariablen bleiben unverändert gegenüber dem Basis-Funktionsbaustein.Die Objektklasse benötigt keine Prozessmethoden. Die Prozesseigenschaft stellt die Messwerte des Sensors zur Verfügung.

Wie auch bereits in Kapitel XXX angesprochen, besitzt der definierte Kraftsensor eine TCP/IP-Schnittstelle. Die Schnittstelle kann verwendet werden um direkt die Sensor-Rohdaten auszulesen. Um sich mit dem Sensor zu verbinden, muss als Port-Nummer 49151 verwendet werden.

Der Sensor kann Anfragebefehle (20 Bytes) entgegennehmen und Datenpakete (16 Bytes) senden. Dabei ist der Aufbau des Befehls und des Datenpaketes genau vorgegeben. Damit die Daten des Sensors ausgelesen werden können, muss in einem ersten Schritt ein Startbefehl gesendet werden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bezeichnung | Sensor-Datentyp | Datentyp TwinCat | Byte-Grösse | Wert des Befehls |
| Command | UINT8 | UINT | 1 | Der Wert muss 0 sein |
| Reserved | UINT8 | UINT | 19 | Alle 19 Werte müssen 0 sein |

Der Sensor stellt anschliessen das Datenpaket mit folgender Struktur zur Verfügung:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bezeichnung | Sensor-Datentyp | Datentyp TwinCat | Byte-Grösse | Beschreibung |
| Header | UINT16 | UINT | 2 | Ein fixer Wert «0x1234» |
| Status | UINT16 | UINT | 2 | Statuswort des Sensors |
| Fx | INT16 | INT | 2 | Kraftwert in X-Richtung |
| Fy | INT16 | INT | 2 | Kraftwert in Y-Richtung |
| Fz | INT16 | INT | 2 | Kraftwert in Z-Richtung |
| Tx | INT16 | INT | 2 | Momentwert um X-Achse |
| Ty | INT16 | INT | 2 | Momentwert um Y-Achse |
| Tz | INT16 | INT | 2 | Momentwert um Z-Achse |

Die Rohmessdaten müssen nun in Newton und Newton/Meter umgewandelt werden. Dies kann mit folgenden Formeln gemacht werden:

|  |  |
| --- | --- |
| Umrechnung der Kraft in Newton | Umrechnung des Moments in Newton/Meter |
|  |  |

Alle Faktoren werden durch den Sensor zur Verfügung gestellt, müssen jedoch über eine Anfrage beantragt werden. Mit folgendem Befehl werden die Faktoren durch den Sensor gesendet:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bezeichnung | Sensor-Datentyp | Datentyp TwinCat | Byte-Grösse | Wert des Befehls |
| Command | UINT8 | UINT | 1 | Der Wert muss 1 sein |
| Reserved | UINT8 | UINT | 19 | Alle 19 Werte müssen 0 sein |

Der Sensor schickt darauffolgend ein Datenpaket mit 24 Byte mit folgender Struktur:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bezeichnung | Sensor-Datentyp | Datentyp TwinCat | Byte-Grösse | Beschreibung |
| Header | UINT16 | UINT | 2 | Ein fixer Wert «0x1234» |
| Unit\_Force | UINT8 | USINT | 1 | Gibt an, welche Einheit mit Faktoren ermittelt wird |
| Unit\_Torque | UINT8 | USINT | 1 | Gibt an, welche Einheit mit Faktoren ermittelt wird |
| CPF | UINT32 | UDINT | 4 | Zählwerte pro Kraftwert (Counts per Force value) |
| CPT | UINT32 | UDINT | 4 | Zählwerte pro Momentwert (Counts per Torque value) |
| ScaleFactorFx | UINT16 | UINT | 2 | Skalier-Faktor für Kraft in X-Richtung |
| ScaleFactorFy | UINT16 | UINT | 2 | Skalier-Faktor für Kraft in Y-Richtung |
| ScaleFactorFz | UINT16 | UINT | 2 | Skalier-Faktor für Kraft in Z-Richtung |
| ScaleFactorTx | UINT16 | UINT | 2 | Skalier-Faktor für Moment um X-Achse |
| ScaleFactorTy | UINT16 | UINT | 2 | Skalier-Faktor für Moment um Y-Achse |
| ScaleFactorTz | UINT16 | UINT | 2 | Skalier-Faktor für Moment um Z-Achse |

Folgende Faktoren werden für die Umrechnung in Newton und Netwon/Meter verwendet:

|  |  |  |  |
| --- | --- | --- | --- |
| *CPF =* | 10000 | *ScaleFactorFx =* | 200 |
| *CPT =* | 10000 | *ScaleFactorFy =* | 200 |
|  |  | *ScaleFactorFz =* | 200 |
|  |  | *ScaleFactorTx =* | 100 |
|  |  | *ScaleFactorTy =* | 100 |
|  |  | *ScaleFactorTz =* | 65 |

In TwinCat wird wieder mit dem Paket TF6310 (TwinCAT 3 | TCP/IP) gearbeitet, um die Kommunikation zum Sensor aufzubauen. Der Sensor stellt hierbei wieder den Server dar. Entsprechend können die gleichen vier Funktionsbausteine wie beim Roboter verwendet werden.

Für die Verbindung zum Server mittels FB\_SocketConnect werden folgende zwei Informationen benötigt:

|  |  |  |
| --- | --- | --- |
| sRemoteHoste: | *192.168.1.10* | IP-Adresse des Sensors |
| sRemotePort: | *49151* | Port für TCP-Schnittstelle |
|  |

Die allgemeine Verwendung der Kommunikationsbausteine bleibt identisch zum Roboter, wie auch der Aufbau der Objektklasse.

Die Kraftsensor-Objektklasse hat grundsätzlich folgenden Aufgaben:

* Aufbau und Verwaltung der Verbindung zum Sensor über TCP/IP
* Entgegennahme, Verarbeitung und Auswertung der vom Roboter gesendeten Daten

Die Objektklasse wertet grundsätzlich die Sensordaten aus, wandelt die Rohdaten um und stellt diese über die Prozesseigenschaft zur Verfügung. Der Prozess der Datenauslesung ist dabei wieder in 4 Schritte aufgeteilt:

Schritt 1: Verbindungsaufbau zu Real-Time-Schnittstelle des Roboters

Schritt 2: Empfangen des Datenpakets (16 Bytes)

Schritt 3: Verarbeitung der Rohdaten

Schritt 4: Trennung der Verbindung zur Real-Time-Schnittstelle

Auch hier musste berücksichtigt werden, dass die Low- und High-Bytes von TwinCat anders interpretiert, als vom Server vorgesehen. Da der Messwert nur aus zwei Byte besteht, kann dies relativ simpel gemacht werden. Das Prinzip ist wie folgt:

Ein Bild, das Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Der über die Kommunikationsschnittstelle erhaltene INT-Wert wird mittels eines UNION-Datentyps in eine Array mit zwei Byte umgewandelt. Die Reihenfolge der Bytes wird umgedreht, damit das Array anschliessend mit einem weiteren UNION-Datentyp, in eine INT-Variable zurückgewandelt wird. Der nun korrekte Rohwert kann jetzt für die Umwandlung in den entsprechende Einheit verwendet werden. In TwinCat sieht dies folgendermassen aus:

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

Automatisch generierte Beschreibung

Die umgewandelten Daten können anschliessend über die Prozesseigenschaft «P\_Messdaten» zur Verfügung gestellt werden. Für den Datentyp der Eigenschaft wurde die Struktur «sKraftvariablen» angegeben. Diese benutzerdefinierte Struktur ist wie folgt definiert:

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

Automatisch generierte Beschreibung

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

Automatisch generierte BeschreibungObjektklasse für 2F-85 (FB\_SmartGreifer):**

Die letzte Objektklasse wird mit dem Interface «I\_Objekt SmartGreifer» implementiert. Da es sich beim Objekt wieder um einen Aktor handelt, werden durch das Interface auch Steuerungsmethoden vorgegeben. Die Objektklasse besitzt keine Objektvariablen. Die Kommunikation zum Greifer wird über TwinCat-Funktionsbausteine aufgebaut, welche direkt auf EA-Klemmen zugreifen.

Die Steuerungs- und Betriebsvariablen bleiben unverändert zum Basis-Funktionsbaustein.

Als Prozessmethoden wurden M\_Aktivieren, M\_Resetten, M\_StatusLesen und M\_Position\_Anfahren definiert. Diese Funktionen decken die Grundfunktionalitäten des Greifers ab. Durch die Methoden kann eine bestimmte Backenposition angefahren werden und überprüft werden, ob diese Position erreicht wurde, oder wann der Greifer in ein Hindernis gefahren ist.

Mit den Prozesseigenschaften werden die prozessrelevanten Parameter für den Greifer definiert. Über P\_AnschlagPos wird die Backenposition zur Verfügung gestellt, bei welcher der Greifer in eine Hindernis gefahren ist.

Die Kommunikation zum Greifer wird über ein Modbus RTU Protokoll realisiert. Innerhalb der Dokumentation wird nicht weiter auf die Spezifikation dieses Protokolls eingegangen. Bei der Kommunikation handelt es sich um eine Master-Slave-Kommunikation. Der Greifer stellt dabei den Slave dar. Die wichtigsten Eigenschaften, welche die Verbindung zum Greifer einhalten muss, sind folgende:

Anschlussschnittstelle: RS-485

Baud-Rate: 115200 bps

Data-Bits: 8 bits

Stop-Bit: 1 Bit

Parität: Keine

Slave-ID: 9

Packetgrösse: 2 Byte (16 bit)

Die Modbus RTU Schnittstelle des Greifers unterstützt verschiedene Funktionalitäten, welche unterschiedliche Register darstellen, auf welche man zugreifen kann. Die Register haben unterschiedliche Anfragebefehle und Antworten. Um die Kommunikationsschnittstelle korrekt anwenden zu können, ist es wichtig diese Funktionalitäten und Ihre Befehle und Antworten zu verstehen. Der genaue Aufbau der Nachricht und Antwort kann aus der Dokumentation des Greifers entnommen werden.

Als Beispiel wird der Befehl für das komplette schliessen des Greifers bei 100% Geschwindigkeit und Kraft gezeigt:

|  |  |  |
| --- | --- | --- |
| Array-Nr. | Bits (Hex) | Beschreibung |
| 0 | 09 | Slave-ID |
| 1 | 10 | Funktionscode 16 (Preset Multiple Registers) |
| 2 | 03 E8 | Adresse des ersten Registers |
| 3 |
| 4 | 00 03 | Anzahl der zu beschreibenden Register |
| 5 |
| 6 | 06 | Anzahl der Daten-Bytes (3 Register x 2 Bytes = 6 Bytes) |
| 7 | 09 00 | Wert auf Register schreiben (Aktivieren von Greifer) |
| 8 |
| 9 | 00 FF | Wert auf Register schreiben (Position für komplett geschlossenen Greifer angeben 255/255) |
| 10 |
| 11 | FF FF | Wert auf Register Schreiben (100% Geschwindigkeit und Kraft) |
| 12 |
| 13 | 42 29 | Zyklische Redundanzprüfung (CRC) |
| 14 |

Die Interaktion mit dem Greifer über die Herstellereigene Software funktionierte einwandfrei. Der Greifer wurde über einen RS-485-USB-Adapter am PC angeschlossen. Der Greifer-Port wurde sofort erkannt und konnte verwendet werden. Auch über Programme wie „Modbustester“ konnte der Greifer verbunden und getestet werden. Die Integration des Greifers in TwinCat via Modbus RTU war jedoch eine aufwändige und zeitintensive Arbeit. TwinCat stellt grundsätzlich ein Paket für die Kommunikation mit Modbus RTU zur Verfügung, TF6255 (TwinCAT 3 | Modbus RTU). Das Paket stellt verschiedene Funktionsbausteine für die Kommunikation zur Verfügung. Einer dieser Funktionsbausteine stellt die Verbindung über eine COM-Port-Schnittstelle auf. Nach endlosen versuchen war es aber nicht möglich eine Verbindung mit dem Greifer aufzubauen. Mit Hilfe einer Software zur Überwachung von Serial-Port-Schnittstellen wurde festgestellt, dass kein Signal TwinCat verlässt. Es wurde kein Weg gefunden eine Verbindung über eine COM-Port-Schnittstelle mittels TwinCat zum Greifer herzustellen.

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

Automatisch generierte BeschreibungAls Alternative wurde versucht sich über eine RS-485-Interface-Karte von Beckhoff mit dem Greifer zu verbinden. Dafür wurde die KL6041 (1-Kanal-Kommunikations\_interface) ausgewählt. Wichtig beim Anschluss der Karte ist, dass der Greifer als «Half-Duplex» angeschlossen werden muss. Dabei wird die Datenleitung für das Sensen und Empfangen verwendet. Im aufgeführten Schema wird dargestellt, wie die Klemme mit den entsprechenden Anschlüssen des Kabels verbunden werden müssen. Über einen Busskoppler (BK1120) kann die Karte über eine EtherCAT-Schnittstelle ausgewertet werden. Es war schlussendlich möglich den Greifer über die KL6041-Karte anzusteuern und Daten auszulesen.

Die Karte muss mit dem Konfigurationstool KS2000 von Beckhoff korrekt konfiguriert werden. Anschliessend kann diese verknüpft und eingesetzt werden.

Bild

Das Paket TF6255 stellt für den Einsatz einer KL6041-Karte einen separaten Funktionsbaustein zur Verfügung.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Der Funktionsbaustein erstellt anhand der Eingangsvariablen den zu sendenden Befehlsframe für die Modbus RTU Kommunikation. Folgende Eingangsvariabel sind relevant für die korrekte Erstellung des Frames:

|  |  |  |
| --- | --- | --- |
| Bezeichnung | Beschreibung | Beispielwert (Tabelle X) |
| UnitID | Slave-ID des Greifers | 09 (Hex) | 9 (Dez) |
| Quantity | Anzahl der zu beschreibenden Register | 03 (Hex) | 3 (Dez) |
| MBAddr | Adresse des ersten Registers | 03E8 (Hex) | 1000 (Dez) |
| cbLenght | Anzahl der zu sendenden Daten in Bytes | 06 (Hex) | 6 (Dez) |

Der folgende Bildausschnitt aus der Methode M\_PositionAnfahren veranschaulicht, wie eine entsprechende Implementierung in TwinCat aussehen kann. Die Slave-ID wurde dabei bereits übergeordnet definiert, da sie für alle Befehle unverändert bleibt.

Ein wichtiger Aspekt bei der Zusammenstellung der Daten für die Register ist die Problematik von High- und Low-Bit. Die Registerdaten sind im Vergleich zur Dokumentation des Greifers vertauscht. Konkret bedeutet dies, dass die Array-Nummern aus dem Beispiel (Tabelle X) [7, 8, 9, 10, 11, 12] in TwinCat zu [8, 7, 10, 9, 12, 11] angepasst werden müssen. Das gleiche Prinzip gilt auch für das Auslesen der Register.

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

Automatisch generierte Beschreibung

Der Funktionscode wird durch die ausgewählte Aktion des Funktionsblocks definiert. Für dieses Beispiel wurde der Funktionscode 16 «Communication.WriteRegs» ausgewählt. Der Basis-Funktionsbaustein, in diesem Fall «Communication» kann nicht in seiner Grundform aufgerufen werden, sondern nur zusammen mit einer Aktion.