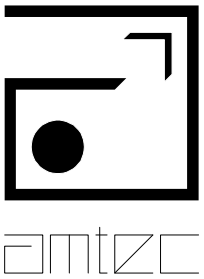

AUTOMATISIERUNGS-
MESS- UND
TESTTECHNOLOGIEN
GMBH

MoRSE-Systemsoftware

Programmierhandbuch

amtec Automatisierungs-, Meß- und Testtechnologien GmbH

MoRSE-Systemsoftware

Programmierhandbuch

Stand: 12.03.02

Schutzgebühr: 150 DM

Referenz: MORSE3D.DOC

amtec
Automatisierungs-, Meß- und Testtechnologien GmbH
Pankstraße 8-10, D - 13127 Berlin (Pankow)
Telefon +49 30 474 99 00, Telefax +49 30 474 99 0 99

Inhaltsverzeichnis

1 MoRSE-Systemsoftware

Die von der amtec GmbH entwickelten intelligenten Antriebsmodule des MoRSE-Systems werden über eine serielle Kommunikationsschnittstelle angesteuert. Das Prinzip der Kommunikation mit den Antriebsmodulen und die dafür verfügbare Systemsoftware werden in dem vorliegenden Programmierhandbuch beschrieben.

In diesem Programmierhandbuch sind alle verfügbaren Funktionsbibliotheken des MoRSE-Systems beschrieben. Bitte beachten Sie, daß nur die Abschnitte für Ihre Anwendung Gültigkeit haben, für die Sie die Funktionsbibliotheken erworben haben.

Die für alle Bibliotheken verwendete, einheitliche *DriveLib*-Programmierschnittstelle stellt eine Treiber-Schnittstelle zur Ansteuerung von intelligenten Positionierantrieben des MoRSE-Systems dar. Auf der Basis dieser Programmierschnittstelle entwickelte Applikationen erlauben den Einsatz aller Funktionsbibliotheken des MoRSE-Systems.

Application Program Interface = API

Diese Programmierschnittstelle (im folgenden API genannt) ist von der konkreten Implementierung unabhängig, das heißt daß unterschiedliche Treibervarianten über die gleiche Programmierschnittstelle angesteuert werden.

1.1 Leistungsumfang der MoRSE-Systemsoftware

Im einzelnen bietet das API mit seinen einfach zu handhabenden Funktionen folgende Leistungen:

- Ansteuerung der MoRSE-Antriebsmodule (***DriveLib***)
- Bahnsteuerung von MoRSE-Manipulatoren und Robotern (***RobotLib***)
- Einbeziehung von Kraft-Momenten-Sensoren (***ForceTorqueSensorLib***)
- Bahnsteuerung mit impedanzgeregelten Bewegungsabläufen (***ImpedanceControlLib***)

Die konkreten Treiber des API stehen Ihnen in Form vorcompilierter statischer Funktionsbibliotheken (LIB) zur Einbindung in Ihre Anwendersoftware zur Verfügung.

1.2 Systemvoraussetzungen

Überzeugen Sie sich bitte zunächst davon, daß Ihr Rechnersystem die nötigen Voraussetzungen zur Anwendung der MoRSE-Systemsoftware erfüllt.

- Der verwendete PC muß ein IBM- oder COMPAQ-kompatibler PC mit mindestens einem 386-Prozessor sein.
- Wenn Sie ein Windows-Programm erstellen, muß MS-Windows 3.x im 'Erweiterten 386-Modus' betrieben werden.

1.3 Installation der MoRSE-Treiber

Die für die Anwendung der einzelnen Funktionsbibliotheken notwendigen Dateien (C-Headerdateien, LIB-Dateien) sind in einem Verzeichnis zusammengefaßt. Im Lieferumfang enthalten ist generell auch ein einfaches

Testprogramm, das Ihnen den Einstieg in die Anwendung der MoRSE-Systemsoftware erleichtern soll.

1.4 Typographie des Handbuchs

Zur Hervorhebung und übersichtlicheren Darstellung bestimmter Sachverhalte werden innerhalb dieses Handbuchs verschiedene Schriftarten verwendet:

`reserved`	reservierte Wörter, Namen von Windows-API-Funktionen etc.
`listing`	Programmlistings von Beispielprogrammen, Namen von API-Funktionen, Elemente der Programmiersprache
parameter	Parameter und Rückgabewerte der API-Funktionen.

1.5 Hinweise, Probleme, Hotline

Die MoRSE-Systemsoftware wurde während ihrer Entwicklung umfangreich getestet und liegen in verschiedenen Implementierungen vor.

Wir sind Ihnen für Hinweise und Ratschläge, die sowohl unsere Software als auch die Dokumentation betreffen, sehr dankbar.

Richten Sie Ihre Fragen oder Hinweise bitte an:

amtec
Automatisierungs, Meß- und Testtechnologien GmbH
Neukirchstraße 62, D - 13089 Berlin (Weissensee)
Telefon (ISDN) +49/30/477 093-0, Telefax +49/30/478 76 51

E-Mail: AMTEC@t-online.de

2 Überblick über die MoRSE-Kommunikationsschnittstelle

Der folgende Abschnitt soll Ihnen einen ersten Überblick über die Kommunikationsschnittstelle vermitteln, über die die Antriebsmodule angesteuert werden.

Die MoRSE-Antriebsmodule der amtec GmbH werden mittels einer seriellen Kommunikationsschnittstelle angesteuert, über die alle Kommandos zur Bewegungssteuerung, Parametrierung und Überwachung zwischen den Modulen und der übergeordneten Steuerung ausgetauscht werden.

Die Kommunikationsschnittstelle ist gegenwärtig in Form eines CAN-Feldbusprotokolls (nach ISO/DIS 11898) sowie auf RS-485 realisiert. Durch die modulare Gestaltung der Antriebsmodul-Software können zukünftig auch andere Protokolle realisiert werden. Dazu ist ein Austausch des Kommunikations-Softwaremoduls sowie der im Antriebsmodul eingesetzten Elektronik erforderlich.

2.1 Struktur der Telegramme

Die Kommando-Telegramme sind für beide genannten Schnittstellentypen identisch. Das bedeutet unter anderem:

- Die bei CAN je Telegramm nutzbare Datenbreite zwischen 0 und 8 Byte gilt für alle Kommandos.
- Es lassen sich bis zu 31 verschiedene Antriebsmodule an einem Bus betreiben.
- Das im Header-Teil eines CAN-Datentelegrammes befindliche ID (11 Bit entsprechend Full-CAN-Definition 2.0A) wird sowohl zur Adressierung des Moduls als auch zur (Grob-)Unterteilung nach Telegrammarten verwendet.

Die im ID untergebrachten Telegrammarten umfassen:

`CANID_MODULEACK`	Bestätigung der Kommunikationsanfrage
`CANID_MODULEREQ`	Kommunikationsanfrage zur Prüfung des Vorhandenseins eines Moduls
`CANID_STATE`	Lesen des Modulstatus
`CANID_SETMOTION`	Bewegungskommando
`CANID_CMDACK`	Bestätigung eines Parameter-Kommandos
`CANID_CMDGET`	Kommando zum Lesen von Parametern
`CANID_CMDPUT`	Kommando zum Setzen von Parametern
`CANID_CMDALL`	wie `CANID_CMDPUT`, jedoch an alle Module gleichzeitig gerichtet

Im folgenden soll das auf RS-485 verwendete Telegramm näher beschrieben werden. Die folgende in der Programmiersprache C definierte Struktur wird für die Kommunikation verwendet:

```
typedef struct
{ UInt8 start;          Startcode (99H)
  UInt16 code;          wie CAN-ID
  UInt8 crc;            Prüfsumme
  UInt8 pBytes[ 8 ];    Nutzdaten für Parameter
} Telegram;
```

Das Strukturelement *start* ist der bei allen Kommandos verwendete Startcode mit dem Wert 99H. Das Element *code* dient der Adressierung des Moduls, der Angabe der Telegrammart sowie der Länge des Nutzdatenbereichs. Dieses Element ist damit identisch mit dem ID beim CAN-Datenpaket. Im Element *crc* wird eine Prüfsumme untergebracht, die aus Gründen der Verarbeitbarkeit auch auf leistungsschwachen Rechnern durch eine Exclusiv-Oder-Verknüpfung aller übrigen Telegramm-Bytes gebildet wird. Unmittelbar danach folgen die je nach der im Element *code* genannten Telegrammart erforderlichen 0 ... 8 Datenbytes im Element *pBytes*.

2.2 Ablauf der Kommunikation

Die Aufnahme der Kommunikation mit den Antriebsmodulen beginnt mit der zyklischen Aussendung von `CANID_MODULEREQ`-Telegrammen durch die übergeordnete Steuerung. Dabei bezeichnet das Datenbyte die logische Nummer des angefragten Moduls. Falls ein solches Modul existiert, antwortet es mit einem entsprechenden `CANID_MODULEACK`-Telegramm.

Nach der Aufnahme der Kommunikation können prinzipiell alle übrigen Telegramme ausgetauscht werden. Dabei arbeitet die übergeordnete Steuerung als Master, die sich jedes ausgesendete Telegramm vom angesprochenen Modul bestätigen läßt. Einzige Ausnahme bildet hier der Typ `CANID_CMDALL`, der alle am Bus befindlichen Module gleichzeitig anspricht.

Die möglichen Telegramm-Paare sind im folgenden dargestellt:

Telegramm-Paare

Anfrage der Steuerung	Antwort vom Modul
`CANID_MODULEREQ` + log. Nummer	`CANID_MODULEACK` + log. Nummer
`CANID_STATE`	`CANID_STATE` + Modulstatuswort
`CANID_SETMOTION` + Bewegungs-Kdo.	`CANID_SETMOTION`
`CANID_CMDGET` + Kdo.	`CANID_CMDACK` + Kdo. + Daten
`CANID_CMDPUT` + Kdo. + Daten	`CANID_CMDACK` + Kdo.
`CANID_CMDALL` + Kdo. + Daten	`<keine Antwort>`

2.3 Aufbau der Parameter-Kommandos

Die festgelegten Kommandos zum Lesen und Setzen von Parametern werden mit `CANID_CMDxxx`-Telegrammen übertragen. Die dazu definierten Konstanten werden im ersten Datenbyte untergebracht. Die übergeordnete Steuerung sendet beim Lesen von Parametern ein `CANID_CMDGET`-Telegramm und beim Setzen von Parametern ein `CANID_CMDPUT`-Telegramm. In beiden Fällen antwortet das Modul mit einem `CANID_CMDACK`-Telegramm.

Alle derzeit implementierten Kommandos zum Lesen und Setzen von Parametern sind weiter unten dargestellt.

Parameter lesen

Beim Lesen von Parametern sendet die übergeordnete Steuerung ein `CANID_CMDGET`-Telegramm, in dem sich im Datenbyte die Kommando-Konstante befindet. Die angeforderten Daten werden vom Modul durch ein `CANID_CMDACK`-Telegramm zurückgesendet.

Folgendes Beispiel soll den Ablauf verdeutlichen:

- Übergeordnete Steuerung sendet:
  `CANID_CMDGET + CMDCAN_POS`
- Modul bestätigt mit:
  `CANID_CMDACK + CMDCAN_POS + <aktuelle Position in Geberimpulsen [Int32]>`

Die übergeordnete Steuerung erhält mit der Antwort des Moduls die aktuelle Position in Geberimpulsen (Inkremente).

Parameter setzen

Beim Setzen von Parametern werden im `CANID_CMDPUT`-Telegramm die Daten untergebracht. Das Modul quittiert dieses Kommando lediglich durch ein (leeres) `CANID_CMDACK`-Telegramm.

Als Beispiel soll die maximal zulässige Geschwindigkeit bei Bewegungskommandos an das Modul übertragen werden, wobei ein Wert in Gleitkommadarstellung (je nach Bewegungsform des Moduls in **m/s** oder **rad/s**) benutzt wird:

- Übergeordnete Steuerung sendet:
  `CANID_CMDPUT + CMDCAN_FMAXVEL + <maximale Geschwindigkeit [Float]>`
- Modul bestätigt mit:
  `CANID_CMDACK + CMDCAN_FMAXVEL`

Parameter in allen Antriebsmodulen gleichzeitig setzen

Darüberhinaus existiert eine Sonderform zum Setzen von Parametern oder Auslösen von Aktionen, bei der alle Module gleichzeitig angesprochen werden. Hierbei erfolgt keine Bestätigung durch die Module. Dieses Kommando kann beispielsweise für die gleichzeitige Ausführung eines Befehls verwendet werden.

Im folgenden Beispiel sollen alle angeschlossenen Module ihren Referenzpunkt anfahren:

- Übergeordnete Steuerung sendet:
  `CANID_CMDALL + CMDCAN_DOSYNC`
- Module bestätigen **nicht** das Telegramm.

Datenformate

Zu beachten ist, daß alle Zahlenwerte in sogenanntem *Little-Endian* -Format übertragen werden, wie dies beispielsweise für x86-Prozessoren, Transputer oder C166 üblich ist. Das bedeutet, daß das *least significant byte* zuerst gesendet wird.

Gleitkomma-Zahlenwerte werden in 32-Bit entsprechend IEEE-754 dargestellt.

Einheiten

Geberimpulse oder Gleitkomma in SI

Für alle Kommandos, bei denen Bewegungsparameter übertragen werden, einschließlich des `CANID_SETMOTION`-Telegrammes zur Auslösung von Bewegungen, existieren Varianten für Geberimpulse oder Gleitkommawerte.

Falls Kommandos mit Gleitkommadarstellung verwendet werden, ist bei der Programmierung von Bewegungen und der Einstellung von Bewegungsparametern zu beachten, daß alle Angaben in SI-Einheiten erfolgen, um weiterführende Berechnungen zu vereinfachen.

Die verwendeten Einheiten sind:

Einheiten

Bedeutung	Linearantrieb	Drehantrieb
Position	**m**	**rad**
Geschwindigkeit	**m/s**	**rad/s**
Beschleunigung	**m/s²**	**rad/s²**
Motorstrom	**A**	**A**
Temperatur	**K**	**K**

Übertragungsparameter

Die Übertragungsparameter bei Verwendung des RS485-Protokolls sind:

8 Datenbit, 2 Stopbit, keine Parität.

Achtung! Immer mit 9600 bit/s anfangen!

Bitte beachten Sie, daß die Antriebsmodule immer zuerst mit 9600 bit/s angesprochen werden müssen. Andere Übertragungsraten lassen sich später mit Hilfe eines `CANID_CMDALL`-Telegrammes mit `CMDCAN_CHANGEBAUDRATE` einstellen.

Die Übertragungsrate ist in weiten Bereichen veränderbar. Die möglichen Übertragungsraten bei der Kommunikation mit den Antriebsmodulen sind 9600, 19200, 38400, 57600, 115200, 144000 und 288000 bit/s. Andere Werte werden entsprechend begrenzt.

2.4 Zusammenfassung der Telegrammtypen

Folgende Telegrammtypen existieren für die Kommunikation zwischen den Antriebsmodulen und der übergeordneten Steuerung:

Telegramme

Telegramm	Konstante	Datenbytes
`CANID_MODULEACK`	`0x0020`	`/* (UInt8)moduleNumber */`
`CANID_MODULEREQ`	`0x0040`	`/* (UInt8)moduleNumber */`
`CANID_STATE`	`0x0060`	`/* (UInt32)state */`

`CANID_SETMOTION`	`0x0080`	`/* (UInt8)motionMode */`
		`/* (Float/Int32)targetValue */`
		`/* (UInt16)stepWidth (nur STEP-Mode) */`

`CANID_CMDACK`	`0x00a0`	`/* (UInt8)cmd (0..7)data */`
`CANID_CMDGET`	`0x00c0`	`/* (UInt8)cmd */`
`CANID_CMDPUT`	`0x00e0`	`/* (UInt8)cmd (0..7)data */`
`CANID_CMDALL`	`0x0100`	`/* (UInt8)cmd (0..7)data */`

2.5 Zusammenfassung der Kommandos

Im folgenden werden alle Kommandos dargestellt, die bei `CANID_CMDGET`-, `CANID_CMDPUT`- oder `CANID_CMDALL`-Telegrammen benutzt werden können.

Folgende Steuerkommandos können nur in `CANID_CMDPUT`-Telegrammen verwendet werden (`CMDCAN_CHANGEBAUDRATE` nur mit `CANID_CMDALL`):

Kommando	Konstante	Datenbytes
`CMDCAN_DORESET`	`0x00`	`/* ./. */`
`CMDCAN_DOSYNC`	`0x01`	`/* ./. */`
`CMDCAN_DOHALT`	`0x02`	`/* ./. */`
`CMDCAN_DOFREEDRIVE`	`0x03`	`/* ./. */`
`CMDCAN_DOINTERNAL`	`0x04`	`/* ./. */`
`CMDCAN_DOCOMMRESET`	`0x05`	`/* ./. */`
`CMDCAN_CHANGEBAUDRATE`	`0x50`	`/* (UInt32)baudRate */`

Mit den folgenden Kommandos können Bewegungsparameter gelesen oder gesetzt werden (in Geberimpulsen):

Kommando	Konstante	Datenbytes
`CMDCAN_POS`	`0x10`	`/* (Int32)theTargetPos */`
`CMDCAN_VEL`	`0x11`	`/* (Int32)theTargetVel */`
`CMDCAN_MAXACC`	`0x20`	`/* (Int32)theMaxAcc */`
`CMDCAN_MAXVEL`	`0x21`	`/* (Int32)theMaxVel */`
`CMDCAN_ACTACC`	`0x22`	`/* (Int32)theActualAcc */`
`CMDCAN_ACTVEL`	`0x23`	`/* (Int32)theActualVel */`
`CMDCAN_POSLIMIT`	`0x24`	`/* (Int32)thePosLimit */`
`CMDCAN_NEGLIMIT`	`0x25`	`/* (Int32)theNegLimit */`

```
CMDCAN_REGCOEFFS        0x26        /* (Int16)theRegCoeff0 */
                                    /* (Int16)theRegCoeff1 */
                                    /* (Int16)theRegCoeff2 */
```

Die folgenden Parameter betreffen jeweils die aktuellen und maximalen Werte für Schleppfehler, Stromaufnahme und Innentemperatur (Schleppfehler in Geberimpulsen, Analogwerte in 10-Bit-A/D-Wandlerausgang):

Kommando	Konstante	Datenbytes
CMDCAN_TOW	0x30	/* (Int32)theActualTow */
CMDCAN_MAXTOW	0x31	/* (Int32)theMaxTow */
CMDCAN_CURRENT	0x32	/* (Int32)theCurrent */
CMDCAN_MAXCURRENT	0x33	/* (Int32)theMaxCurrent */
CMDCAN_TEMPERATURE	0x34	/* (Int32)theTemperature */
CMDCAN_MAXTEMPERATURE	0x35	/* (Int32)theMaxTemperature */

Die hier genannten Standardwerte (nur lesbar) sind im Modul fest vorgegebene Einstellungen. Der angegebene Fahrbereich sowie die Grenzwerte (CMDCAN_DEFMAX...) dürfen nur weiter eingeschränkt werden (Geberimpulse soweit anwendbar):

Kommando	Konstante	Datenbytes
CMDCAN_DEFMAXACC	0x40	/* (Int32)theMaxAcc */
CMDCAN_DEFMAXVEL	0x41	/* (Int32)theMaxVel */
CMDCAN_DEFPOSLIMIT	0x42	/* (Int32)thePosLimit */
CMDCAN_DEFNEGLIMIT	0x43	/* (Int32)theNegLimit */
CMDCAN_DEFREGCOEFFS	0x44	/* (Int16)theRegCoeff0 */
		/* (Int16)theRegCoeff1 */
		/* (Int16)theRegCoeff2 */
CMDCAN_DEFMAXCURRENT	0x45	/* (Int32)theMaxCurrent */
CMDCAN_DEFMAXTOW	0x46	/* (Int32)theMaxTow */
CMDCAN_DEFMAXTEMPERATURE		0x47 /* (Int32)theMaxTemperature */
CMDCAN_DEFINCRPERUNIT	0x48	/* (Float)incrPerUnit */
CMDCAN_DEFMODULETYPE	0x49	/* (UInt8)moduleType */
CMDCAN_DEFROMVERSION	0x4a	/* (UInt16)romVersion */
CMDCAN_DEFSERIALNUMBER	0x4b	/* (UInt32)serialNumber */
CMDCAN_DEFPRODUCTNUMBER	0x4c	/* (Char[7])pProductNumber */

Folgende Bewegungsparameter werden in Gleitkommadarstellung übertragen:

Kommando	Konstante	Datenbytes
CMDCAN_FPOS	0x60	/* (Float)theTargetPos */
CMDCAN_FVEL	0x61	/* (Float)theTargetVel */
CMDCAN_FMAXACC	0x70	/* (Float)theMaxAcc */
CMDCAN_FMAXVEL	0x71	/* (Float)theMaxVel */
CMDCAN_FACTACC	0x72	/* (Float)theActualAcc */
CMDCAN_FACTVEL	0x73	/* (Float)theActualVel */
CMDCAN_FPOSLIMIT	0x74	/* (Float)thePosLimit */
CMDCAN_FNEGLIMIT	0x75	/* (Float)theNegLimit */

Es folgen die aktuellen Parameter in Gleitkommadarstellung:

Kommando	Konstante	Datenbytes
CMDCAN_FTOW	0x80	/* (Float)theActualTow */
CMDCAN_FMAXTOW	0x81	/* (Float)theMaxTow */
CMDCAN_FCURRENT	0x82	/* (Float)theCurrent */
CMDCAN_FMAXCURRENT	0x83	/* (Float)theMaxCurrent */
CMDCAN_FTEMPERATURE	0x84	/* (Float)theTemperature */
CMDCAN_FMAXTEMPERATURE	0x85	/* (Float)theMaxTemperature */

Die Standardwerte (nur lesbar) in Gleitkommadarstellung:

Kommando	Konstante	Datenbytes
CMDCAN_FDEFMAXACC	0x90	/* (Float)theMaxAcc */
CMDCAN_FDEFMAXVEL	0x91	/* (Float)theMaxVel */
CMDCAN_FDEFPOSLIMIT	0x92	/* (Float)thePosLimit */

```
CMDCAN_FDEFNEGLIMIT     0x93        /* (Float)theNegLimit */
CMDCAN_FDEFMAXCURRENT   0x95        /* (Float)theMaxCurrent */
CMDCAN_FDEFMAXTOW       0x96        /* (Float)theMaxTow */
CMDCAN_FDEFMAXTEMPERATURE           0x97/* (Float)theMaxTemperature
*/
```

3 Grundlagen der Programmierung mit MoRSE-Systembibliotheken

Die MoRSE-Systembibliotheken (CLD-Treiber) lassen sich – unabhängig von der unterstützten Thematik – nach einem bestimmten Schema anwenden. Wie bereits eingangs geschildert, ist die Schnittstellendefinition unabhängig von der konkreten Implementierung. Darüberhinaus sind einige allgemeine Funktionen bei allen Schnittstellendefinitionen zu finden.

Im folgenden werden die grundlegenden Mechanismen der verschiedenen allgemeinen Dienste von CLD-Treibern beschrieben.

3.1 Aufbau der Systembibliotheken

CLD = CronoLog Device

CLD-Treiber sind Software-Bibliotheken zur Ansteuerung von Hardware-Schnittstellen. Die Programmierschnittstelle aller festgelegten Thematiken ist grundsätzlich unabhängig von der konkreten Implementierung. CLD-Treiber werden in der Regel in Form von statischen Bibliotheken (LIB) bereitgestellt. Dadurch ist es möglich, in einer einmal erstellten Anwendung mit einem anderen Treiber zu arbeiten, sofern dieser Treiber die gleiche Thematik unterstützt.

Fehlermeldungen

Da die konkrete Implementierung eines Treibers während der Programmierung nicht bekannt ist oder sich bei Verwendung eines anderen Treibers ändert, lassen sich nicht alle möglichen Fehlerzustände von vornherein bestimmen. Die Funktion `CLD_errorMsg` wandelt einen von einer Treiber-Funktion gelieferten Fehlercode in einen Textstring um. Dabei kann der Klartext in der Regel in mehreren Sprachen zurückgeliefert werden. Die Auswertung des Fehlertextes und mögliche Darstellung auf dem Bildschirm, Speicherung in einer Datei oder ähnlichem bleiben dem Programmierer der Anwendung überlassen.

Format von CLD-Funktionen

Der Funktionsprototyp aller CLD-Funktionen ist ähnlich aufgebaut:

- Funktionsname
  Der Name jeder Funktion – ausgenommen wenige allgemeine Funktionen, die in jedem Treiber vorhanden sind – beinhaltet die Thematik, der diese Funktion zuzuordnen ist.
- Parameter und gelieferte Daten
  Von der Funktion zu liefernde Daten werden über Referenzparameter an das Anwenderprogramm übermittelt. Dadurch können Funktionen auch mehrere Informationen gleichzeitig bereitstellen.
- Fehlercodes
  Der eigentliche Rückgabewert der Funktion stellt einen Fehlercode dar. Dadurch läßt sich der Erfolg des Funktionsaufrufs oder die genaue Ursache eines Fehlers feststellen.

3.2 Programmierung mit CLD-Treibern

In diesem Abschnitt werden allgemein im Umgang mit CLD-Treibern häufig benötigte Datentypen und Strukturen erläutert. Die Handhabung von CLD-Treibern – speziell das Laden eines Treibers und die Vorgehensweise bei Fehlerzuständen – werden beschrieben.

Allgemeines

CLD-Treiber werden in der Regel in Form einer LIB ausgeliefert. Die Treiber können auch in Form von Quellcode einer höheren Programmiersprache zur Verfügung stehen, in diesem Fall treffen dennoch alle folgenden Aussagen im wesentlichen zu.

Folgende Dateien werden im Zusammenhang mit der Programmierung benötigt:

- die Funktionsbibliothek (LIB) des Treibers bzw. die benötigten Quellcode-Dateien,
- C-*Header*-Dateien mit allgemeinen Definitionen, Konstanten und Funktionsprototypen.

Da meist C- oder C++-Programmierumgebungen benutzt werden, bezieht sich die Beschreibung der Datentypen und Strukturen auf die Programmiersprache C..

Dateien

Wie bereits oben genannt, werden neben der eigentlichen Treiber-LIB die C-Header-Dateien zur Programmentwicklung benötigt.

Diese Header-Dateien (*include*-Dateien) enthalten folgende Informationen:

- Definition der Standard-Datentypen
- Definition der verwendeten Strukturen
- Definition von Konstanten und Fehlercodes
- Liste der Funktionsprototypen

Die Header-Dateien beschreiben in der Regel keine Eigenschaften des konkreten Treibers, sondern die für mehrere Implementierungen gültige Schnittstelle. Daher werden für die konkreten Versionen von Treibern keine weiteren Informationen benötigt.

Programmierkonventionen

Zu den im folgenden genannten Konventionen zur Programmierung gehören das verwendete Speichermodell, die Datentypen und die Definition der Fehlercodes.

Speichermodell

Das im Zusammenhang mit CLD-Treibern unter DOS/Windows zu verwendende Speichermodell ist LARGE. Das bedeutet, daß Adressen von Daten und Code segmentiert zu verwenden sind. Falls in der Anwendung ein anderes Speichermodell verwendet werden soll, sind einige Korrekturen an Header-Dateien erforderlich. So muß vor allem bei jeder Verwendung von Zeigervariablen dafür gesorgt werden, daß `far`-Adressen übergeben werden.

Datentypen

Die bei den Treiber-Funktionen verwendeten Standard-Datentypen sind:

Datentypen

Datentyp	Bedeutung
`Void`	void
`Char, Int8, Bool`	8 Bit ganzzahlig mit Vorzeichen
`UChar, Byte, UInt8`	8 Bit ganzzahlig ohne Vorzeichen
`Short, Int16`	16 Bit ganzzahlig mit Vorzeichen
`UShort, SWord, UInt16`	16 Bit ganzzahlig ohne Vorzeichen
`Int, Int32`	32 Bit ganzzahlig mit Vorzeichen
`UInt, Word, UInt32`	32 Bit ganzzahlig ohne Vorzeichen
`Float`	32 Bit Gleitkomma
`Double`	64 Bit Gleitkomma
`LongDouble`	80 Bit Gleitkomma

Alle oben genannten Datentypen stellen lediglich speziell vereinbarte Namen für die in der Regel bereits in der Programmierumgebung vorhandenen Standarddatentypen dar. Diese Vereinbarungen befinden sich in der Datei _TYPES.H. Darüberhinaus sind noch weitere Datentypen in dieser Datei vereinbart, die eine spezielle Verwendung bezeichnen.

Der von allen CLD-Funktionen gelieferte Rückgabewert ist vom Typ:

`CLDRet`	32 Bit ganzzahlig ohne Vorzeichen – wird verwendet, um den Erfolg eines Funktionsaufrufes anzuzeigen.

Zur Auswahl des vom Treiber anzusprechenden Gerätes ist `CLDID` zu verwenden:

`CLDID`	16 Bit ganzzahlig mit Vorzeichen – bezeichnet das von der Funktion zu verwendende Gerät eindeutig.

Rückgabewert von Funktionen

CLD_OK == erfolgreicher Funktionsaufruf

Der eigentliche Rückgabewert von Funktionen zeigt den Erfolg des Funktionsaufrufes an. Zu diesem Zweck sind spezielle Konstanten in den CLDxxxxx-Dateien definiert worden, die den Fehlercode bezeichnen.

Die für alle CLD-Treiber geltenden Rückgabewerte sind:

Rückgabewerte

Rückgabewert	Bedeutung
`CLD_OK`	Erfolg – kein Fehler
`CLDERR_LIBRARYNOTFOUND`	Initialisierungsfehler – Treiber konnte nicht initialisiert werden
`CLDERR_LIBRARYNOTLOADED`	fehlerhafter Aufruf einer Anwendungsfunktion – System-DLL ist noch nicht geladen worden
`CLDERR_UNKNOWNFUNCTION`	die gewünschte Funktion ist nicht bekannt
`CLDERR_HARDWARENOTINSTALLED`	die vom Treiber anzusprechende Hardware ist nicht installiert oder konnte nicht initialisiert werden
`CLDERR_UNKNOWNERROR`	unbekannte Fehlerursache
`CLDERR_BADPARAM`	falsche Parameter wurden an die Funktion übergeben
`CLDERR_ALREADYUSED`	das angeforderte Gerät wird bereits von einer anderen Anwendung benutzt

- ☐ `CLDERR_TOOMANYDEVICES` es werden bereits alle vom Treiber oder von der Hardware unterstützten Geräte benutzt
- ☐ `CLDERR_BADINDEX` falscher Geräteindex – es wurde versucht, ein nicht vorhandenes oder noch nicht initialisiertes Gerät zu benutzen
- ☐ `CLDERR_FUNCTIONNOTAVAILABLE` aufgerufene Funktion ist nicht verfügbar
- ☐ `CLDERR_BADDEVICEVERSION` Funktion kann vom angeschlossenen Gerät nicht ausgeführt werden, da falsche Version
- ☐ `CLDERR_INITIALIZATIONERROR` Fehler bei der Initialisierung
- ☐ `CLDERR_DRIVERNOTINSTALLED` ein weiterer benötigter Treiber ist noch nicht installiert worden

Der speziellen Thematik zugeordnete Fehlercodes sind im Referenzteil des vorliegenden Handbuches zu jeder Funktion erklärt.

Vorgehensweise bei Fehlern

Falls eine CLD-Funktion einen Rückgabewert ungleich 0 liefert, wird damit ein Fehler signalisiert. Der Rückgabewert stellt eine 32-Bit-Konstante dar, die zur Fehleranalyse ausgewertet werden kann.

4 Grundlagen der DriveLib-Schnittstelle

In diesem Abschnitt wird die Programmier-Schnittstelle *DriveLib* zur Ansteuerung von intelligenten Antriebsmodulen beschrieben. Dazu zählen Funktionen zur Initialisierung der Schnittstelle zu den Antrieben, Funktionen zur Parametrierung der Antriebe und Bewegungsfunktionen.

4.1 Allgemeines

Die Funktionen der Programmierschnittstelle beziehen sich auf zwei Begriffe eines realen Handhabungssystems – zum einen das gesamte Handhabungssystem (*Manipulator*), bestehend aus mehreren Antriebsmodulen – zum anderen ein einzelner Antrieb (*Drive*).

Bei der Ansteuerung der Module wird zuerst die Struktur des realen Handhabungssystems ermittelt, danach können die einzelnen Module angesprochen werden.

Bewegungsart der Module

Alle Angaben erfolgen in SI-Einheiten (m oder rad)

Die modulspezifischen Funktionen zur Parametrierung und Bewegung erhalten in der Regel Werte im Gleitkommaformat (32 Bit), die die Längenangaben (bei Linearmodulen) oder Winkelangaben (bei Drehmodulen) in Einheiten des SI-Systems darstellen. Dadurch wird der Anwender von der sonst erforderlichen Umrechnung in sogenannte Inkremente (Geberimpulse der Antriebseinheiten) entlastet.

4.2 Vorbereitung

Der folgende Abschnitt beschreibt die erforderlichen Maßnahmen, um die Treiber zu initialisieren sowie den Manipulator und seine Antriebsmodule für Handhabungsaufgaben vorzubereiten.

Initialisierung der CronoCom

Für das Versenden und Empfangen von Nachrichten zwischen den intelligenten Antriebsmodulen und dem PC ist die PC-Einsteckkarte CronoCom verantwortlich. Sie muß zu Beginn der Arbeiten initialisiert und nach Abschluß entsprechend deinitialisiert werden. Dazu dienen die Funktionen `CLDXtrnl_init` und `CLDXtrnl_exit`. Mit dem zu übergebenden Initialisierungsstring erhält die Schnittstelle die Information, auf welcher Basisadresse die CronoCom zu finden ist, welcher Interrupt eingestellt wurde und auf welche Startadresse der Dual-Port-RAM-Speicher zu konfigurieren ist. Das folgende Beispiel illustriert die Verwendung der beiden Befehle:

```
Char* pInitString = "300,11,b0000";    // Basisadresse 300H
                                       // Interrupt 11
                                       // DPR-Startadresse b0000H
CLDXtrnlID xtrnlID;

CLDRet retVal = CLDXtrnl_init( &xtrnlID, pInitString );
```

```
if( retVal != CLD_OK )                // Initialisierung
fehlerhaft?
  ...                                 //   ja - Fehlerauswertung
else
{ ...                                 // weitere Funktionen
}
CLDXtrnl_exit( xtrnlID );             // letzter Funktionsaufruf...
```

Initialisierung der Buskommunikation

Nachdem die PC-Einsteckkarte CronoCom initialisiert wurde, dient der Funktionsaufruf `CLDDrive_initManip` dazu, die Schnittstelle zur Ansteuerung von Antriebsmodulen zu initialisieren.

Das Format des Initialisierungsstrings der konkreten Implementierung finden Sie im Anhang!

Dabei wird die Kommunikationsstrecke zu den Antriebsmodulen vorbereitet, wozu der Initialisierungsstring abhängig von der konkreten Implementierung verwendet wird. Außerdem wird die Zahl der angeschlossenen Antriebsmodule ermittelt. Bitte beachten Sie folgende Zuordnung:

- RS-485 Buskommunikation: `pInitString = "288000"`
- CAN-Buskommunikation: `pInitString = "CAN:1,250000"`

Folgendes Beispiel zeigt die Initialisierung:

```
Short numDrives;
Char* pInitString = "CAN:1,250000";
CLDRet retVal = CLDDrive_initManip( &numDrives, pInitString );
if( retVal != CLD_OK )                // Initialisierung
fehlerhaft?
  ...                                 //   ja - Fehlerauswertung
else
{ ...                                 // weitere Funktionen
}
```

Funktionen werden über ID angesprochen

Falls hierbei kein Fehlercode zurückgeliefert wird, enthält die Variable '`numDrives`' die Anzahl der festgestellten Antriebsmodule. In allen weiteren Funktionen wird das zu bearbeitende Antriebsmodul über ein ID (eine eindeutige Identifikations-Nummer) angesprochen. Dabei ist die Nummer des ersten Antriebsmoduls 0, die des zweiten 1 usw.

Initialisierung der Module

Die beim oben gezeigten Beispiel erhaltene Zahl der angeschlossenen Module wird benötigt, um die Kommunikationsverbindung zu jedem angeschlossenen Antriebsmodul herzustellen:

```
Signed i;
for( i = 0; i < numDrives; ++i )
{ CLDRet retVal = CLDDrive_initModule ( i );
  if( retVal != CLD_OK )
    ...                               // fehlerhaft
  else
  { ...                               // weitere Funktionen
  }
```

Modultyp erfragen

Zur Feststellung der Bewegungsform des jeweiligen Antriebsmoduls und anderer wesentlicher Eigenschaften dient die Funktion

`CLDDrive_getConfigDrive`. Diese Funktion füllt eine Struktur mit den entsprechenden Werten. Eines der Elemente dieser Struktur (`type`) stellt den Typ des Moduls dar und entscheidet über die Interpretation der Funktionsparameter. Folgende Modultypen sind definiert:

- `ROTARY_DRIVE` — Antriebsmodul ist ein Drehmodul, die Parameter werden in der Einheit **`rad`** (Radiant) angegeben.
- `LINEAR_DRIVE` — Antriebsmodul ist ein Linearmodul, die Parameter werden in der Einheit **`m`** (Meter) angegeben.

Beispiel:

```
ConfigDrive configDrive;
CLDRet retVal = CLDDrive_getConfigDrive ( moduleNumber,
&configDrive );
if( configDrive.type == LINEAR_DRIVE )
  ...                                  // Angaben in Meter
else if( configDrive.type == ROTARY_DRIVE )
  ...                                  // Angaben in Radiant
...                                    // weitere Funktionen
```

Die einzelnen Elemente der Struktur sind ausführlich im Referenzteil zur Funktion `CLDDrive_getConfigDrive` auf Seite 76 beschrieben.

4.3 Programmierung von Bewegungen

Im folgenden werden die Einheiten der Bewegungsparameter sowie die Bedeutung einzelner Funktionskategorien erläutert. Zur Verwendung der Funktionen siehe auch Abschnitt 8, *Referenz* .

Einheiten

Bei der Programmierung von Bewegungen und der Einstellung von Bewegungsparametern ist zu beachten, daß alle Angaben in SI-Einheiten erfolgen, um weiterführende Berechnungen zu vereinfachen.

Die verwendeten Einheiten sind:

Einheiten

Bedeutung	Linearantrieb	Drehantrieb
Position	**`m`**	**`rad`**
Geschwindigkeit	**`m/s`**	**`rad/s`**
Beschleunigung	**`m/s²`**	**`rad/s²`**
Motorstrom	**`A`**	**`A`**
Temperatur	**`K`**	**`K`**

Verfahrbereich

Das Betriebssystem des Antriebsmoduls schränkt normalerweise den möglichen Verfahrbereich ein. Dieser Verfahrbereich bezeichnet den Abstand zwischen den im Modul eingestellten Softwareendlagen.

Dynamik

Bei allen Bewegungen des Antriebs werden voreingestellte oder von der übergeordneten Steuerung vorgegebene Werte für Fahrgeschwindigkeit und -beschleunigung verwendet. Dabei werden zwei auch als *Dynamik* bezeich-

nete Wertepaare benutzt. Die *aktuelle* Dynamik wird bei allen von der übergeordneten Steuerung (dem PC-Anwenderprogramm) ausgelösten Bewegungen verwendet. Die *maximale* Dynamik dient dem Modul-Betriebssystem zur Begrenzung aller Bewegungen. Diese Grenzwerte sind standardmäßig im Modul voreingestellt, sie können jedoch von der übergeordneten Steuerung weiter eingeschränkt werden.

Reglerkoeffizienten

Die zur Positionierung verwendeten Parameter des im Betriebssystem der Antriebsmodule implementierten PID-Reglers lassen sich mit Hilfe der Funktion `CLDDrive_setRegCoeff` verändern.

Die rekursive Beziehung des PID-Regelalgorithmus ist:

$$u_k = u_{k-1} + q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}$$

mit den Parametern

- ☐ regCoeff0 $q_0 = K(1 + \frac{T_0}{2\,T_I} + \frac{T_D}{T_0})$
- ☐ regCoeff1 $q_1 = -K(1 + 2\,\frac{T_D}{T_0} - \frac{T_0}{2\,T_I})$
- ☐ regCoeff2 $q_2 = K\,\frac{T_D}{T_0}$.

Status des Antriebsmoduls abfragen

Zur Erkennung von im Antriebsmodul aufgetretenen Fehlern und zur Erfassung des Modulstatus wird die Funktion `CLDDrive_getStatus` verwendet. Der Status des Antriebsmoduls ist ein vorzeichenloser 32-Bit-Wert, dessen niederwertiger Teil (16 Bit) den Zustand des Antriebs signalisiert. Bei vom Betriebssystem des Moduls festgestellten Fehlern wird im höherwertigen 16-Bit-Teil der Fehlerzustand beschrieben.

Die einzelnen Bits des Statuswortes können über binäre UND-Verknüpfungen extrahiert werden.

Bits im Statuswort

Bit im Statuswort	Bedeutung
☐ `ST_ENDPOS`	Endpunkt der Rampenbewegung erreicht.
☐ `ST_HALTED`	Antriebsmodul angehalten und im Sicherheitszustand.
☐ `ST_SYNCHRONIZED`	Antriebsmodul ist synchronisiert.
☐ `ST_WARN_BEYONDSOFTLIMIT`	Vom Bewegungskommando geforderte Zielposition weist nach außerhalb des erlaubten Bereichs und wird auf innerhalb der Softwareendlagen begrenzt.
☐ `ST_BEYONDSOFTLIMIT`	Antriebsmodul befindet sich außerhalb der Softwareendlagen.
☐ `ST_BEYONDHARDLIMIT`	Antriebsmodul befindet sich außerhalb der Hardwareendlagen.
☐ `ST_ERROR`	Antriebsmodul hat Fehler festgestellt und befindet sich im Sicherheitszustand.

☐ ST_ERR_SYNCHRONIZE	Fehler beim Anfahren des Referenzpunktes oder RESET-Kommando ohne vorheriges Synchronisieren des Moduls.
☐ ST_ERR_FREEDRIVE	Fehler beim Freifahren des Moduls.
☐ ST_ERR_HIGHCURRENT	Überstrom in der Endstufe des Antriebsmoduls.
☐ ST_ERR_GATEVOLTAGE	Betriebsspannung der Endstufe außerhalb des zulässigen Bereiches.
☐ ST_ERR_TEMPERATURE	Überschreitung des Temperatur-Grenzwertes im Modul.
☐ ST_ERR_TOW	Überschreitung des Schleppfehler-Grenzwertes.
☐ ST_ERR_INTERNAL	Interner Fehler des Antriebsmoduls.
☐ ST_ERR_COMMERROR	Antriebsmodul hat Protokollfehler der Kommunikationsstrecke festgestellt.
☐ ST_ERR_CRCERROR	Antriebsmodul hat Prüfsummenfehler eines Kommunikationstelegrammes festgestellt.
☐ ST_ERR_FATALERROR	Fataler Fehler – Endstufen wurden abgeschaltet (z.B. zu hoher Schleppfehler oder Endstufe defekt).
☐ ST_ERR_HARDMAXPOS	Positive (maximale) Hardwareendlage überschritten.
☐ ST_ERR_HARDMINPOS	Negative (minimale) Hardwareendlage überschritten.
☐ ST_ERR_SOFTMAXPOS	Positive (maximale) Softwareendlage überschritten.
☐ ST_ERR_SOFTMINPOS	Negative (minimale) Softwareendlage überschritten.

Darüberhinaus existieren für die Feststellung der Beendigung von langandauernden Vorgängen weitere, einfach zu handhabende Funktionen:

Funktion	Bool-Wert liefert TRUE, wenn ...
☐ CLDDrive_querySyncEnd	Referenzpunkt erreicht
☐ CLDDrive_queryEndPos	Endpunkt der Rampenbewegung erreicht (nach CLDDrive_moveRamp) oder Stillstand des Antriebs (nach CLDDrive_haltModule oder bei Fehler)
☐ CLDDrive_queryFreeDrive	Modul wurde aus Software- und Hardwareendlagen freigefahren (nach dem Aufruf von CLDDrive_freeDriveModule)

Diese Funktionen liefern einen Wert vom Typ `Bool`, der das Ende des jeweiligen Vorgangs anzeigt.

4.4 Dateien

#include DRVAPI.H genügt!

Zur Vereinfachung der Programmierung genügt es, wenn Sie die Datei `DRVAPI.H` mittels einer `#include`-Anweisung einfügen – diese Datei liest selbständig alle anderen benötigten Header-Dateien ein.

☐ DRVAPI.H –	Vereinbarungen für die *CLDDrive*-Thematik
☐ DRVAPI.LIB –	Statische Funktionsbibliothek für die Antriebssteuerung zur Einbindung ins Anwenderprogramm

Darüberhinaus sind möglicherweise noch andere Dateien erforderlich, die der Lieferung beigefügt sind.

Die Include-Dateien müssen sich in einem dem Projekt zugänglichen Verzeichnis befinden.

5 Grundlagen der RobotLib-Schnittstelle

Im vorliegenden Abschnitt wird beschrieben, wie mehrere zu einem Robotersystem verbundene Antriebsmodule angesteuert werden. Die CLDRobot-Schnittstelle erlaubt die Ansteuerung von bis zu 32 Roboterstrukturen von einer Steuerung.

5.1 Allgemeines

Programmierung in Weltkoordinaten

Bei der Steuerung von Antriebsmodulen, die zu einem Robotersystem verbunden wurden, sind die auszuführenden Bewegungen möglichst in Koordinaten des räumlichen Bezugssystems zu programmieren.

Die bei der gesteuerten Ausführung von Roboterbewegungen erforderlichen Berechnungsschritte sind folgende:

- Eine **Ablaufsteuerung**, vorgegeben durch ein Anwenderprogramm, generiert die einzelnen Programmschritte. Diese Programmschritte stellen einzelne Bewegungskommandos dar, die die Geometrie und Dynamik der Bewegung beschreiben.
- Die übergebenen Bewegungsbefehle werden von der **Bahnsteuerung** interpretiert, indem zyklisch – mit einem festen Zeittakt – Raumpunkte berechnet werden. Diese Raumpunkte liegen in Koordinaten des räumlichen Bezugssystems vor (Weltkoordinaten).
- Ein optionaler Berechnungsvorgang zur **Einbeziehung von Sensorinformationen** korrigiert die bereits ermittelten Raumpunkte so, daß die vom Programm vorgegebenen Bahnbewegungen durch Sensoren dynamisch den veränderten Umgebungsbedingungen angepaßt werden können.
- Die anschließende Stufe wandelt mit Hilfe der sogenannten Inversen **Koordinatentransformation** die Weltkoordinaten der Raumpunkte in Koordinaten der einzelnen Roboterachsen um. Die Algorithmen sind spezifisch für die jeweils vorliegende kinematische Struktur des Roboters.
- Alle **Antriebsmodule** des Roboters fahren nun geregelt die ermittelten Positionen (auch als Achs- oder Gelenkkoordinaten bezeichnet) an, wodurch die Gesamtstruktur des Roboters die gewünschte Bewegung ausführt.

Während bei der Steuerung der Antriebsmodule über die CLDDrive-Programmierschnittstelle jeder einzelne Antrieb positioniert werden muß, sind zur Steuerung eines Roboters vom Hostrechner aus Bahnkommandos zu senden – die Robotersteuerung übernimmt die Positionierung der einzelnen Antriebsmodule.

CLDRobot-API für Bahnsteuerung

Dazu dient die im vorliegenden Abschnitt beschriebene CLDRobot-Programmierschnittstelle. Sie ermöglicht es dem Anwendungsprogrammierer, den Roboter als einheitliches System zu betrachten und mit Bahnbefehlen komplexe Bewegungen auszuführen.

5.2 Robotermodus

Manipulator « Roboter

Die Ansteuerung der Module des MoRSE-Systems erfolgt über einen Feldbus, an den insgesamt bis zu 31 Module angeschlossen sein können. Alle über diesen Bus angeschlossenen Antriebe bilden den *Manipulator*. Die einzelnen Antriebe eines *Roboters* sind dagegen ausgewählte, in besonderer

Weise mechanisch miteinander verbundene Module.

Es könnten durchaus auch mehrere Roboterstrukturen in einem Manipulator vorhanden, das heißt an einem Bus angeschlossen sein. Aus diesem Grund ist zur Initialisierung des Robotermodus mitzuteilen, welche Module des Manipulators eine Roboterstruktur bilden.

Merkmale des Robotermodus

Im Ablauf des Anwendungsprogramms muß deshalb nach der Aufnahme der Kommunikation mit den Antriebsmodulen und einer eventuell erforderlichen Initialisierung die Initialisierung des Robotermodus erfolgen.

Die Zuschaltung dieser Betriebsart zieht keinerlei Einschränkungen für den Betrieb der Antriebsmodule mit der CLDDrive-Schnittstelle nach sich. Roboterbefehle, wie z.B. Bewegungsbefehle (`CLDRobot_move...`) können im Anwenderprogramm beliebig durch Drive-Kommandos ergänzt werden.

Der Robotermodus bietet für die klassischen Roboterstrukturen vorgefertigte Transformationsroutinen an. Ebenso sind kundenspezifische Transformationen implementiert, die die Bahnsteuerung unkonventioneller Robotersysteme erlauben.

Um dem modularen Charakter des MoRSE-Systems gerecht zu werden, wurden in die Roboterschnittstelle Funktionen integriert, mit deren Hilfe der Anwender eine Anpassung an sein konkretes System erreichen kann.

Benötigte Informationen

Bahnsteuerung benötigt folgende Informationen:

Die Stufen der Bahnsteuerung und der Koordinatentransformation benötigen für die zyklische Berechnung der einzelnen Gelenkkoordinaten entsprechend der vorgegebenen Bahnbefehle folgende Informationen:

- die vom Anwender aufgebaute Roboterkonfiguration (Robotertyp),
- die Zahl der Antriebe im Robotersystem und ihre physische Adresse im Bussystem,
- die Abmessungen der einzelnen Antriebe und ihre Lage zueinander,

Die Informationen sind in einer Initialisierungsphase vom Hostrechner zu übergeben. Diese Phase wird im folgenden als Initialisierung des Roboters bezeichnet.

Funktionen zur Initialisierung:

Zur Initialisierung eines oder mehrerer Roboter sind folgende Funktionen erforderlich:

☐ `CLDRobot_enterRobMode`	Initialisierung des Robotermodus
☐ `CLDRobot_initRob`	Initialisierung eines Roboters mit Angabe der Zahl der Antriebe und der physischen Adresse im Bussystem, sowie der ausgewählten Konfiguration

Die Verwendung dieser Funktionen wird im folgenden Abschnitten beschrieben.

Initialisierung

Vor der Initialisierung eines oder mehrerer Roboter ist es erforderlich, den Robotermodus zu initialisieren. Dies wird durch Aufruf der Funktion `CLDRobot_enterRobMode`. Bei Ausführung dieser Funktion werden intern Datenbereiche angelegt, die zur Verwaltung der Roboter notwendig sind. Im Sinne einer Kontrolle liefert die Funktion die Anzahl aller am Bus

angeschlossenen Antriebsmodule. Es ist unbedingt erforderlich, daß das MoRSE-Antriebssystem vor Aufruf des Robotermodus durch die CLDDrive-Schnittstelle ordnungsgemäß initialisiert wurde (durch Aufruf von `CLDXtrnl_init` sowie `CLDDrive_initManip` und `CLDDrive_initModule`).

Achtung! Vor Aufruf der Initialisierungsfunktion für den Robotermodus sollte auf jeden Fall die Initialisierung der CLDDrive-Schnittstelle, wie im entsprechenden Abschnitt beschrieben, erfolgt sein.

Der Aufruf der Funktion `CLDRobot_enterRobMode` ist im Anwenderprogramm nur einmal erforderlich. Nach ihrer Ausführung können vom Anwender bis zu 32 Roboter mit Hilfe von `CLDRobot_initRob` eingerichtet werden.

Wie bereits in der CLDDrive-Schnittstelle und anderen CLD-Thematiken angewendet, erhält die Funktion `CLDRobot_initRob` zur Initialisierung eine Zeichenkette mit den erforderlichen Angaben und liefert bei erfolgreicher Ausführung der Funktion einen eindeutigen Bezeichner für den Roboter.

Der Prototyp der Initialisierungsfunktion lautet:

```
CLDRet CLDRobot_initRob( CLDRobotID* pRobotId, Char* pInitString,
RobotType robType, const char* fileName );
```

Format des Initialisierungsstrings Der Initialisierungsstring, der im Parameter *pInitString* übergeben wird, besitzt folgendes Format:

<*Zahl der Module*>**[**<*1. Modul*>**,**<*2. Modul*>**,**. . .**]**<*Name*>

Die Reihenfolge der Nummern der Antriebsmodule gibt gleichzeitig den prinzipiellen Aufbau des Roboters an, das heißt welche Antriebe zum Beispiel Fuß, Oberarm, Unterarm usw. bilden. Dabei wird mit dem vom Greifer oder Werkzeug am weitesten entfernten Modul begonnen. Der Parameter *Name* im InitString wird in der den Roboter beschreibenden Struktur abgelegt und kann vom Anwender später auch abgefragt werden. Er hat für die Initialisierung keine Bedeutung.

Um beispielsweise die Antriebsmodule mit den Nummern 0, 2, 4, 5 und 9 als Bestandteil einer Roboterstruktur zu definieren, wobei das Fußmodul die Nummer 2 besitzt, müßte der Initialisierungsstring lauten:

```
"5[2,0,4,5,9]JumpingJack"
```

Folgendes Beispiel verdeutlicht den Vorgang der Initialisierung:

```
CLDRobotID robotId;
Char* pInitString = "5[2,0,4,5,9]JumpingJack";
RobotType robType = ELBOW5;
Char* pFileName = "elbow5.txt"
CLDRet retVal = CLDRobot_initRob( &robotId, pInitString, robType,
pFileName );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

robotId Der Wert in *robotId* stellt einen eindeutigen Bezeichner für den angemeldeten Roboter dar und wird bei allen nachfolgenden Funktionsaufrufen verwendet.

Mit dem Dateinamen verweist der Anwender auf eine Konfigurationsdatei für das defnierte Robotersystem. Die Angaben in der Datei entnehmen Sie bitte dem Abschnitt Aufbau der Konfigurationsdateien auf Seite 40.

Damit ist die Anmeldung des ersten Roboters abgeschlossen. Von diesem Zeitpunkt an können alle weiteren CLDRobot-Kommandos auf diesen Roboter, eindeutig beschrieben durch sein *robotID*, angewendet werden.

Entspricht der vom Anwender aufgebaute Roboter nicht der voreingestellten Konfiguration (siehe betreffenden Abschnitt), so kann die konkrete geometrische Lage angepaßt werden. Lesen Sie dazu den Abschnitt Anpassung der Robotergeometrie auf Seite 39.

Referenzpunktfahrt

Die CLDRobot-Schnittstelle verfügt über einen Befehl zum Referieren des durch das *robotID* gekennzeichneten Roboters. Dieses Kommando kann optional vom Anwender gewählt werden, wenn er die Referenzpunktfahrt der einzelnen Antriebe nicht durch die Befehle der CLDDrive-Schnittstelle ausführen möchte.

Mit dem Aufruf der Funktion `CLDRobot_homeRob` kann der Anwender durch einen InitString zusätzlich angeben, in welcher Reihenfolge die einzelnen Achsen referiert (synchronisiert) werden sollen. Im folgenden Beispiel wird ein 5-Achser in der Reihenfolge Ellenbogen(2), HandKippen(3), HandDrehen(4), Oberarm(1), Schulter(0) referiert:

```
Char* pInitString = "2,3,4,1,0";
CLDRet retVal = CLDRobot_homeRob( robotId, pInitString );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // weiter
```

Es ist zu beachten, daß sich der Initstring hier nur auf die Reihenfolge der Referenzpunktfahrt der einzelnen Module bezieht. Es handelt sich nicht um die physischen Adressen (*driveID*).

Beenden des Robotermodus

Vor dem Verlassen des Robotermodus, müssen zunächst alle eingerichteten Roboter deinitialisiert werden. Dazu wird die Funktion `CLDRobot_exitRob` gerufen.

```
// alle Roboter deinitialisieren...
for( i = 0; i < robCount; i++ )
  CLDRobot_exitRob( i );
CLDRobot_exitRobMode();              // Robotermodus verlassen
```

Wenn ein Roboter deinitialisiert wurde, ist das zuvor als Kennung benutzte *robotId* nicht mehr definiert. Es können demzufolge keine weiteren `CLDRobot_...`-Funktionen für die mit *robotId* vereinbarte Roboterstruktur gerufen werden.

Im Anschluß kann der Robotermodus mit `CLDRobot_exitRobMode` verlassen werden. Die zuvor belegten Datenbereiche für die Verwaltung der Roboter werden so wieder ordnungsgemäß freigegeben.

5.3 Bewegungsbefehle

Im folgenden Abschnitt wird die Behandlung von Bewegungsbefehlen im Anwenderprogramm diskutiert. Voraussetzung für die Auslösung von Roboterbewegungen ist eine erfolgreiche Initialisierung des Robotermodus und mindestens eines Roboters.

Die Bewegungsbefehle der CLDRobot-Schnittstelle teilen sich in zwei Kategorien:

- Bewegungsbefehle für Roboter unter Beschreibung des Zielpunktes in Achskoordinaten (`CLDRobot_goTo`),
- Bewegungsbefehle für Roboter unter Beschreibung des Zielpunktes in Weltkoordinaten (`CLDRobot_movePTP`, `CLDRobot_moveLIN`, `CLDRobot_moveCIRC`).

Die Bewegungsbefehle werden an den durch das *robotID* bestimmten Roboter gesendet. Als weitere Parameter erhalten die Bewegungsbefehle eine Angabe über den Zielpunkt der Bewegung sowie die für die Ausführung der Bewegung vorgesehene Zeit.

Die Zielpunkte für die Bewegung werden in einer Struktur zusammengefaßt. Den Kategorien für Bewegungsbefehle entsprechend existieren Strukturen zur Beschreibung eines Raumpunktes in Achskoordinaten sowie in Weltkoordinaten. Der folgende Abschnitt gibt Auskunft über die Verwendung.

Beschreibung von Raumpunkten

Für die Beschreibung von Raumpunkten sind entsprechend der Unterteilung Achs- / Weltkoordinaten zwei Strukturen vorgesehen. Die Struktur `RobotCoord` dient zur Beschreibung eines Raumpunktes in Achskoordinaten (häufig werden auch die Begriffe Gelenkkoordinaten oder Roboterkoordinaten verwendet). Zur Beschreibung von Weltkoordinaten (Koordinaten des Bezugssystems) ist die Struktur `WorldCoord` vorgesehen.

Die Struktur RobotCoord

Die Beschreibung eines Raumpunktes in Achskoordinaten ist von der Anzahl der Robotergelenke abhängig. Die Struktur `RobotCoord` wird aus diesem Grund dynamisch behandelt. Sie enthält ein Feld `V` von `size` Elementen. Jedes Element stellt eine Achsposition dar.

Die Struktur `RobotCoord` ist in der Datei `DEFROBOT.H` wie folgt definiert:

Struktur RobotCoord

Element	Datentyp	Bedeutung
`size`	`UInt16`	Anzahl der zur Positionsbeschreibung vorgesehenen Roboterachsen
`V`	`Double*`	Feld von size Elementen zur Beschreibung der Achsposition [rad] bzw. [m]

Das folgende Beispiel soll die Verwendung der Struktur `RobotCoord` verdeutlichen:

```
RobotCoord robCoord;
int drvCount = 5;                    // Roboter mit 5 Achsen
Float time = 2.0;                    // 2 Sekunden für Bewegung

robCoord.V = (Double*)calloc( drvCount, sizeof( Double ) );
robCoord.V[ 0 ] = ...;               // Angabe in rad bzw. m
robCoord.V[ 1 ] = ...;               // Angabe in rad bzw. m
robCoord.V[ 2 ] = ...;               // Angabe in rad bzw. m
robCoord.V[ 3 ] = ...;               // Angabe in rad bzw. m
robCoord.V[ 4 ] = ...;               // Angabe in rad bzw. m

CLDRobot_goTo( robotID, &robCoord, &time );
free( robCoord.V );                  // Freigeben des Speichers
```

Die dynamische Behandlung der Achskoordinaten bedingt die explizite Belegung des notwendigen Speichers durch den Anwender und seine Freigabe, wenn die Daten nicht mehr genutzt werden.

Die Struktur WorldCoord

Die für die Bahnbewegungen anzugebenden Roboterpositionen sind definiert als Position und Orientierung des Greifer- oder Werkzeugendpunktes relativ zum Bezugskoordinatensystem.

Zur Beschreibung der Roboterpositionen in Weltkoordinaten dient die Struktur `WorldCoord`, die in der Datei `DEFROBOT.H` wie folgt definiert ist:

Struktur WorldCoord

Element	Datentyp	Bedeutung
☐ `x`	`Float`	Translation entlang der x-Achse [m]
☐ `y`	`Float`	Translation entlang der y-Achse [m]
☐ `z`	`Float`	Translation entlang der z-Achse [m]
☐ `a`	`Float`	Rotation um die x-Achse [rad]
☐ `b`	`Float`	Rotation um die y-Achse [rad]
☐ `c`	`Float`	Rotation um die z-Achse [rad]

Die Verschiebungen und Drehungen beziehen sich auf das eingestellte Bezugskoordinatensystem (siehe Abschnitt Anpassung des Bezugskoordinatensystems).

Das folgende Beispiel soll die Verwendung der Struktur `WorldCoord` verdeutlichen:

```
WorldCoord worldCoord;
Float time = 2.0;                    // 2 Sekunden für Bewegung

worldCoord.x = ...;                  // Angabe in m
worldCoord.y = ...;                  // Angabe in m
worldCoord.z = ...;                  // Angabe in m
worldCoord.a = ...;                  // Angabe in rad
worldCoord.b = ...;                  // Angabe in rad
worldCoord.c = ...;                  // Angabe in rad

CLDRobot_movePTP( robotID, &worldCoord, &time );
```

Weil die Struktur zur Beschreibung von Weltkoordinaten fest definiert ist, ist eine dynamische Speicherzuweisung nicht erforderlich.

Zeitsynchrone Bewegung in Achskoordinaten

Die Funktion `CLDRobot_goTo` erlaubt das Verfahren eines zuvor definierten Robotors unter Angabe eines Zielpunktes in Achskoordinaten. Vom Anwender wird die Dauer der Bewegung in Sekunden festgelegt. Das System ermittelt nun selbständig die für jede Achse optimale Geschwindigkeit und steuert sie so an, daß alle Antriebe zum gleichen Zeitpunkt ihren Zielpunkt erreichen.

Das folgende Beispiel verdeutlicht die Anwendung:

```
CLDRet retVal;
Float time = 2.0;                    // 2 Sekunden für Bewegung

retVal = CLDRobot_goTo( robotID, &robCoord, &time );
if( retVal != CLD_OK )
```

```
  ...                                  // Fehlerauswertung
else
  ...                                  // weiter
```

Diese Funktion ist nicht dazu geeignet, den Roboter entlang einer vordefinierten Bahn zu verfahren.

Zeitsynchrone Bewegung in Weltkoordinaten

Die Funktion `CLDRobot_movePTP` ermöglicht das Verfahren eines zuvor definierten Robotors unter Angabe eines Zielpunktes in Weltkoordinaten. Vom Anwender wird die Dauer der Bewegung in Sekunden festgelegt. Das System ermittelt nun selbständig die Achskoordinaten sowie die optimale Geschwindigkeit für jede Achse und steuert sie so an, daß alle Antriebe zum gleichen Zeitpunkt ihren Zielpunkt, angegeben in Weltkoordinaten erreichen.

```
CLDRet retVal;
WorldCoord target = {...};
Float time = 2.0;                      // 2 Sekunden für Bewegung

retVal = CLDRobot_movePTP( robotID, &target, &time );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // weiter
```

Diese Funktion ist nicht dazu geeignet, den Roboter entlang einer vordefinierten Bahn zu verfahren.

Linearbewegung

Der Befehl `CLDRobot_moveLIN` ist ein Bahnbewegungskommando. Er erlaubt das Verfahren eines zuvor definierten Robotors unter Angabe eines Zielpunktes in Weltkoordinaten entlang einer Geraden, die vom aktuellen Ausgangspunkt zum bezeichneten Zielpunkt verläuft. Vom Anwender wird die Dauer der Bewegung in Sekunden festgelegt.

```
CLDRet retVal;
WorldCoord target = {...};             // Zielpunkt
Float time = 2.0;                      // 2 Sekunden für Bewegung

retVal = CLDRobot_moveLIN( robotID, &target, &time );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // weiter
```

Der vorgegebene Bezugspunkt fährt entlang einer Geraden mit einem trapezförmigen Geschwindigkeitsprofil von Ruhe in Ruhe.

Kreisbewegung

Der Befehl `CLDRobot_moveCIRC` ist gleichfalls ein Bahnbewegungskommando. Er erlaubt das Verfahren eines zuvor definierten Robotors unter Angabe eines Zielpunktes in Weltkoordinaten entlang eines Kreisbogens, der vom aktuellen Ausgangspunkt zum bezeichneten Zielpunkt verläuft. Die Lage

des befahrenen Kreisbogens im Raum wird durch drei Punkte (Ausgangspunkt, Zwischenpunkt, Zielpunkt) definiert. Die Funktion erhält aus diesem Grund vom Anwender die Information über die Lage des Zielpunktes sowie des Zwischenpunktes in Weltkoordinaten. Gleichzeitig wird vom Anwender die Dauer der Bewegung in Sekunden festgelegt.

```
CLDRet retVal;
WorldCoord target = {...};            // Zielpunkt
WorldCoord through = {...};           // Kreismittelpunkt
Float time = 2.0;                     // 2 Sekunden für Bewegung

retVal = CLDRobot_moveCIRC( robotID, &through, &target, &time );
if( retVal != CLD_OK )
  ...                                 // Fehlerauswertung
else
  ...                                 // weiter
```

Der vorgegebene Bezugspunkt fährt entlang eines Kreisbogens mit einem trapezförmigen Geschwindigkeitsprofil von Ruhe in Ruhe.

5.4 Stop- und Notstop-Funktionen

Für den Praxisbetrieb ist es unerläßlich, daß die Bewegung des Roboters unterbrochen, bzw. eine Nothalt-Funktion aufgerufen werden kann.

Die CLDRobot-Schnittstelle stellt diese Funktionen mit `CLDRobot_stopRob` und `CLDRobot_haltRob` zur Verfügung. Beide Kommandos lösen einen unmittelbaren Halt der Roboterbewegung aus. Im Unterschied zu `CLDRobot_stopRob` versetzt `CLDRobot_haltRob` den Roboter in einen Zustand, in dem er ohne explizites Reset keine weiteren Bewegungsbefehle annimmt.

Der Aufruf von `CLDRobot_haltRob` ist demzufolge nur dann erforderlich, wenn der Programmierer eine besondere Bestätigung des Anwenders erwünscht, um im Programm fortzufahren.

5.5 Überwachung

Aktuelle Position in Weltkoordinaten

Mit Hilfe der Funktion `CLDRobot_getToolCoord` läßt sich jederzeit feststellen, welche Position in Weltkoordinaten der durch sein *robotID* bestimmte Roboter augenblicklich einnimmt. Die Lage wird in Weltkoordinaten beschrieben. Aus diesem Grund kommt die Struktur WorldCoord zur Anwendung.

Aktuelle Position in Achskoordinaten

Mit Hilfe der Funktion `CLDRobot_getRobCoord` kann der Anwender jederzeit feststellen, welche Position der durch sein *robotID* bestimmte Roboter augenblicklich hat. Die Position wird in Achskoordinaten beschrieben. Aus diesem Grund kommt die Struktur RobotCoord zur Anwendung.

Aktuelle Achsgeschwindigkeiten

Die Funktion `CLDRobot_getRobVel` liefert die aktuellen Achsgeschwindigkeiten des Roboters. Die ermittelten Daten werden in Form der Struktur RobotCoord übergeben.

Aktuelle Achsdynamik

Die Achsdynamik wird durch die aktuelle Geschwindigkeit und Beschleunigung der Roboterachsen charakterisiert. Durch Aufruf der Furnktion `CLDRobot_getActDynRobot` erhält der Anwender diese Information unter Verwendung der Struktur RobotCoord.

Aktuelle Achsschleppfehler

Der Schleppfehler pro Roboterachse läßt sich mit `CLDRobot_getTow` erfragen. Dazu findet die Struktur RobotCoord Anwendung.

Aktuelle Stromaufnahme der Achsen

Die aktuelle Stromaufnahme der Roboterachsen kann zu Steuerungszwecken vom Anwender als Eingangsgröße für einen überlagerten Regelkreis herangezogen werden. Mit Hilfe der Funktion `CLDRobot_getCurrent` werden die Daten in Form der Struktur RobotCoord übergeben.

Aktuelle Achstemperaturen

Die Erfassung der Achstemperaturen ist im Sinne der Systemüberwachung sinnvoll. Mit `CLDRobot_getTemperature`erhält der Anwender die Information, übergeben mit der Struktur RobotCoord.

5.6 Grenzwerte

Im folgenden Abschnitt wird erläutert, wie im Anwenderprogramm auf die Grenzwerte der Bewegungsparameter Einfluß genommen werden kann. Dies bezieht sich auf den Bewegungsbereich und die Bewegungsdynamik der einzelnen Roboterachsen sowie auf die Bewegungsdynamik des gesamten Robotersystem, bezogen auf den Tool Centre Point.

Werden die Grenzwerte der Bewegungsparameter beeinflußt, so können Geschwindigkeiten und Beschleunigungen des Robotersystems limitiert werden, ebenso wie der Arbeitsbereich.

Endlagen der Roboterachsen

Bei einer aus MoRSE-Antrieben zusammengesetzten Roboterstruktur wird es sich sehr wahrscheinlich als notwendig erweisen, den Bewegungsbereich der einzelnen Achsen zu limitieren. Unter Zuhilfenahme der Funktion `CLDRobot_setDriveRanges` ist die Beschränkung des Bewegungsbereiches aller Roboterachsen möglich. Die Funktion `CLDRobot_getDriveRanges` dient der Abfrage der aktuell eingestellten Limits.

Die Funktionen basieren auf den entsprechenden in der CLDDrive-Schnittstelle beschriebenen. Gleichzeitig werden auch die für die Robotersteuerung notwendigen Datenbereiche ergänzt bzw. überschrieben.

Bewegungsdynamik der Roboterachsen

Wie bereits unter der CLDDrive-Schnittstelle, können die einzelnen Roboterachsen in ihrer Bewegungsdynamik (Maximal-Geschwindigkeiten und -Beschleunigungen) beschränkt werden. Die Funktion `CLDRobot_setMaxDynDrives` realisiert diese Aufgabe und paßt gleichzeitig die Datenbereiche für die Robotersteuerung an. Mit `CLDRobot_getMaxDynDrives` kann der Anwender die aktuellen Einstellungen erfragen.

Bewegungsdynamik des Roboters

Die vom Tool Center Point ausgeführte Bahnbewegung (bei Ansteuerung durch `CLDRobot_moveLIN` oder `CLDRobot_moveCIRC`) unterliegt einem trapezförmigen Geschwindigkeitsprofil (beschleunigte, gleichförmige und verzögerte Bewegung). Die Eckparameter dieser Bahndynamik werden durch die Maximal-Geschwindigkeiten und -Beschleunigungen beschrieben. Sie können vom Anwender unter Aufruf von `CLDRobot_setMaxDynRobot` gesetzt bzw. durch `CLDRobot_getMaxDynRobot` abgefragt werden.

Schleppfehler der Roboterachsen

Weil jeder Antrieb über eine eigene und unabhängige Regelung verfügt, kann der Anwender die Regelgüte beeinflussen. Dazu wird der Schleppfehler pro Achse in engere oder weitere Grenzen gesetzt. Dies wird durch die Funktion `CLDRobot_setMaxTow` erreicht. Mit `CLDRobot_getMaxTow` kann der eingestellte Wert abgefragt werden.

Eine Minimierung des achsweisen Schleppfehlers erhöht die Bahntreue des Robotersystems. Unter Umständen muß jedoch vom Anwender eine Begrenzung der Roboterdynamik vorgesehen werden, weil die Antriebe bei Überschreitung ihrer maximalen Schleppfehlervorgabe selbständig anhalten und einen Fehlerstatus anzeigen.

Maximale Stromaufnahme der Roboterachsen

Eine Limitierung der maximalen Stromaufnahme pro Achse wird durch Aufruf der Funktion `CLDRobot_setMaxCurrent` erreicht. Mit `CLDRobot_getMaxCurrent` kann der eingestellte Wert abgefragt werden. Wird die Vorgabe im Betrieb überschritten, so hält der entsprechende Antrieb augenblicklich an und zeigt einen Fehlerstatus.

Maximale Temperatur der Roboterachsen

Die maximale Innentemperatur der Roboterachsen wird durch `CLDRobot_setMaxTemperature` erreicht. Die Funktion `CLDRobot_getMaxTemperature` realisiert die Abfrage des eingestellten Wertes.

Strombegrenzung für Achsregelung zuschalten

Die Achsregelung kann durch eine überlagerte Strombegrenzung erweitert werden. Wurde die Strombegrenzung aktiviert, so kann der Anwender wie gewohnt Bewegungsbefehle auslösen. Erreicht ein Antrieb während der

Bewegung sein Limit, wird keine weitere Beschleunigung zugelassen. Der Antrieb bewegt sich unter der vorgegebenen Stromaufnahme zu seinem Zielpunkt. Die Einstellung erfolgt mit der Funktion `CLDRobot_setLimCurrent`.

Grenzwerte für Kraftschwellen vorgeben

Ist ein Kraft-Momenten-Sensor im System installiert, kann der Anwender eine Vorgabe für Maximalkräfte einführen. Wird bei der Ausführung von Roboterbewegungen eine Überschreitung der anwenderdefinierten Kraftschwellen registriert, so führt dies zum sofortigen Abbruch der Bewegung und Stillstand des Roboters. Die Einstellung erfolgt mit der Funktion `CLDRobot_setForceLimits`. Entsprechend gestattet die Funktion `CLDRobot_getForceLimits` die Abfrage der aktuellen Einstellung.

5.7 Voreingestellte Roboterkonfigurationen

Der Datentyp `RobotType` bestimmt die Auswahl eine Roboterkonfiguration durch den Anwender bei Aufruf der Funktion `CLDRobot_initRob`. Im folgenden Abschnitt werden die in der CLDRobot-Schnittstelle voreingestellten Roboterkonfigurationen erläutert.

Folgende Konfigurationen sind verfügbar:

- ELBOW3
- ELBOW5
- ELBOW6
- DASA_RIGHT
- DASA_LEFT

Die Roboterkonfigurationen können in ihrer konkreten Geometrie mit Hilfe der Funktionen zur Anpassung der Robotergeometrie beeinflußt werden. Lesen Sie dazu den entsprechenden Abschnitt im Handbuch.

5.8 Anpassung der Bezugssysteme

Die internen Bezugssyteme der angemeldeten Roboter sind konfigurationsabhängig. Sie sind entsprechend der voreingestellten Konfiguration nach Anmeldung gesetzt. Lesen Sie dazu den Abschnitt Voreingestellte Roboterkonfigurationen auf Seite 36.

Im allgemeinen bezieht sich der Ursprung des Basiskoordinatensystems, in dem der Roboter definiert ist, auf den Roboterfuß. Das Toolkoordinaten-system mit dem Tool Centre Point als Urprung liegt in den voreingestellten Konfigurationen üblicherweise in der Handwurzel.

Die Bezugssysteme können vom Anwender an die konkreten Gegebenheiten angepaßt werden.

Die Struktur Frame

Zur Veränderung der Lage eines Bezugspunktes im Raum, wird eine Verschiebung (Translation) und Verdrehung (Rotation) gegenüber seiner ursprünglichen Lage ermittelt. Dazu dient eine Matrix mit 3 Zeilen und 4 Spalten. Sie wird in der CLDRobot-Schnittstelle mit Hilfe der Struktur `Frame` abgebildet.

Die Struktur `Frame` ist in der Datei `DEFROBOT.H` wie folgt definiert:

Struktur Frame

	Element	Datentyp	Bedeutung
☐	`row`	`Short`	Anzahl der Zeilen
☐	`col`	`Short`	Anzahl der Spalten
☐	`type`	`Short`	Typ
☐	`V`	`Double*`	Feld von der Größe row*col zur Darstellung der Matrix

Änderung des Basiskoordinatensystems

Mit der Funktion `CLDRobot_setBaseFrame` kann der Anwender das voreingestellte Basiskoordinatensystem an die speziellen Gegebenheiten anpassen. `CLDRobot_getBaseFrame` dient zur Abfrage des aktuell definierten Basiskoordinatensystems. Das folgende Beispiel demonstriert den Gebrauch der beiden Funktionen:

```
Frame baseFrame;

CLDRobot_getBaseFrame( robotId, &baseFrame );
baseFrame.V[ 0 ] = ...;              // Translation und Rotation
...
baseFrame.V[ 11 ] = ...;

CLDRobot_setBaseFrame( robotId, &baseFrame );
```

Alle weiteren Angaben für Bahnbewegungen beziehen sich auf das neu definierte Basiskoordinatensystem.

Änderung des Toolkoordinatensystems

Mit der Funktion `CLDRobot_setToolFrame ist es möglich,` den Tool Centre Point neu zu definieren. Im allgemeinen befindet er sich in den voreingestellten Konfigurationen in der Handwurzel des Roboterarms (Schnittstelle Handgelenk/Greifer). Die Funktion `CLDRobot_getToolFrame` dient zur Abfrage des aktuell definierten Toolkoordinatensystems. Das folgende Beispiel demonstriert den Gebrauch der beiden Funktionen:

```
Frame toolFrame;

CLDRobot_getToolFrame( robotId, &toolFrame );
toolFrame.V[ 3 ] = ...;              // Translation in x [m]
toolFrame.V[ 7 ] = ...;              // Translation in y [m]
toolFrame.V[ 11 ] = ...;             // Translation in z [m]

CLDRobot_setToolFrame( robotId, &toolFrame );
```

Alle weiteren Angaben für Bahnbewegungen beziehen sich auf das neu definierte Toolkoordinatensystem.

Definition/Änderung des Taskkoordinatensystems

Mit der Funktion `CLDRobot_setTaskFrame` ist es möglich, ein aufgabenorientiertes Bezugssystem zu definieren. Die Funktion `CLDRobot_getTaskFrame` dient zur Abfrage des aktuell definierten Taskkoordinatensystems. Das gewählte Bezugssystem kann als Typ übergeben werden. Folgende Konfigurationen sind verfügbar:

- ☐ BASE_FRAME

- TOOL_FRAME
- USER_DEFINED

Das folgende Beispiel demonstriert den Gebrauch der beiden Funktionen:

```
Frame taskFrame;
FrameTypeType type;

CLDRobot_getTaskFrame( robotId, &type, &taskFrame );
taskFrame.V[ 3 ] = ...;              // Translation in x [m]
taskFrame.V[ 7 ] = ...;              // Translation in y [m]
taskFrame.V[ 11 ] = ...;             // Translation in z [m]

CLDRobot_setTaskFrame( robotId, type, &taskFrame );
```

Um Bewegungen mit Positionsvorgaben im Taskframe auszulösen, ist der entsprechende Parameter beim Aufruf von Bewegungsfunktionen zu übergeben.

Einbeziehung von Kraft-Momenten-Sensoren

Mit der Funktion `CLDRobot_setFTSensorData` ist es möglich, der Roboterkonfiguration einen Kraft-Momenten-Sensor zuzuteilen und eine Paramtrierung der Transformationsdaten vorzunehmen. Für den Betrieb des Sensors an der Roboterstruktur müssen die folgenden Voraussetzungen erfüllt sein:

- erfolgreiche Initialisierung des Sensors (lesen Sie dazu den Abschnitt Grundlagen der ForceTorqueSensor-Schnittstelle auf Seite 42.
- ein gültiges Handle (ID) für den Sensor.
- ein gültiges Roboter-ID.
- Abstand des TCP zum Meßmittelpunkt des Sensors.
- Rotation des Sensors gegenüber dem TCP.
- Kenntnis des maximalen Meßbereichs des Sensors.

Mit der Funktion `CLDRobot_getFTSensorData` kann der Anwender die aktuellen Einstellungen abfragen. Das folgende Beispiel verdeutlicht die Anwendung der Funktionen:

```
Float height;
Float phi;
FTVector limits;

CLDRobot_getFTSensorData( robotId, &height, &phi, limits );
limits[ 0 ] = ...;                   // Meßbereich in Fx [N]
...
limits[ 5 ] = ...;                   // Meßbereich in Mz [Nm]
height = ...;
phi = ...;

CLDRobot_setFTSensorData( robotId, height, phi, limits );
```

5.9 Anpassung der Robotergeometrie

Beschreibung der Denavit-Hartenberg-Parameter

Zur Beschreibung der Greiferendstellung werden körperfeste kartesische Rechtskoordinatensysteme als Zwischenkoordinatensysteme S_i in jedes Gelenk gelegt und die Übergänge zum jeweiligen folgenden Koordinatensystem S_{i+1} ermittelt.

Die Transformationsvorschriften, die den Übergang des Koordinatensystems S_i in das Koordinatensystem S_{i+1} beschreiben, ergeben sich durch die folgenden Operationen:

- Rotation mit dem Winkel $phi0_i$ um die Achse z_i
- Translation mit der Länge b_i entlang der Achse z_i
- Translation mit der Länge $d0_i$ entlang der Achse x_{i+1}
- Rotation mit dem Winkel $teta_i$ um die Achse x_{i+1}.

Die Größen phi0, b, d0 und teta sind die sogenannten Denavit-Hartenberg-Parameter, die die Länge und Orientierung des jeweiligen Gelenkes charakterisieren. Die z_i-Achse des Koordinatensystems S_i kennzeichnet die Bewegungsachse des i-ten Gelenkes. Ihre Richtung wird durch die positive Bewegungsrichtung des i-ten Gelenkes festgelegt. Die x_i -Achse ist durch die gemeinsame Normale von z_{i-1} nach z_i definiert.

- `b` — Abstand des Schnittpunktes von Achse z_i und Achse x_{i+1} und Koordinatenursprung U_i (Translation entlang der Achse z_i)
- `d0` — Länge der gemeinsamen Normalen der Achsen z_i und z_{i+1} (Translation entlang der Achse $x_{neu} = x_{i+1}$)
- `teta` — Winkel zwischen der Achse z_i und der um die Achse x_{i+1} im mathematisch positiven Sinne gedrehten Achse z_{i+1} (Rotation um die Achse $x_{neu} = x_{i+1}$)
- `phi0` — Winkel zwischen der Achse x_i und der um die Achse z_i im mathematisch positiven Sinne gedrehten Achse x_{i+1} (Rotation um die Achse z_i)
- `joint` — Bewegungsart des Antriebs

Das Element `joint` dient der Information, in welcher Art das Antriebsmodul bewegt wird. Dazu wird eine Konstante vom Typ `JointType` verwendet (ebenfalls in der Datei `HDR\DEF\DEFROBOT.H` definiert):

- `TRANS_X_JOINT` — Translationsbewegung, Bewegungsachse ist die x-Achse
- `TRANS_Z_JOINT` — Translationsbewegung, Bewegungsachse ist die z-Achse
- `ROT_PHI_JOINT` — Rotationsbewegung, Bewegungsachse ist die z-Achse

Aufbau der Konfigurationsdateien

5.10 Einheiten

Wie bereits bei den Funktionen der CLDDrive-Thematik angewendet, werden alle Parameter für die Programmierung von Bewegungen in Einheiten des SI-Systems dargestellt, um weiterführende Berechnungen zu vereinfachen.

Die bei der Steuerung des Roboters verwendeten Einheiten sind:

Einheiten

Physikalische Größe	Einheit
Länge	**m**
Winkel	**rad**
Bahngeschwindigkeit	**m/s**
Bahnbeschleunigung	**m/s²**

5.11 Dateien

Zur Verwendung der Funktionen der CLDRobot-Thematik sind folgende Dateien erforderlich:

- `DRVAPI.H` – Vereinbarungen für die *CLDDrive*-Thematik
- `ROBAPI.H` – Vereinbarungen für die *CLDRobot*-Thematik
- `FTSAPI.H` – Vereinbarungen für die *CLDFTS*-Thematik
- `DRVAPI.LIB` – Statische Funktionsbibliothek für die Antriebssteuerung zur Einbindung ins Anwenderprogramm
- `ROBAPI.LIB` – Statische Funktionsbibliothek für Roboterfunktionen zur Einbindung ins Anwenderprogramm
- `FTSAPI.LIB` – Statische Funktionsbibliothek zur Einbindung von Kraft-Momenten-Sensoren ins Anwenderprogramm

Darüberhinaus sind möglicherweise noch andere Dateien erforderlich, die der Lieferung beigefügt sind.

#include ROBAPI.H genügt!

Zur Vereinfachung der Programmierung genügt es, wenn Sie die Datei `ROBAPI.H` mittels einer `#include`-Anweisung einfügen – diese Datei liest selbständig alle anderen benötigten Header-Dateien ein.

Die Include-Dateien müssen sich in einem dem Projekt zugänglichen Verzeichnis befinden.

6 Grundlagen der ForceTorque-Sensor-Schnittstelle

Im vorliegenden Abschnitt wird beschrieben, wie Kraft-Momenten-Sensorik in die Robotersteuerung eingebunden wird. Die CLDForceTorqueSensor-Schnittstelle erlaubt die Abfrage von bis zu 32 Kraft-Momentensensoren von einer Steuerung.

6.1 Allgemeines

Erweiterung der Robotersteuerung

Bei der Robotersteuerung in räumlichen Koordinatensystemen ist die Einbeziehung von Sensordaten für viele Aufgaben hilfreich. Mit einem Kraft-Momentensensor ist die objektive Erfassung der auf den Roboter wirkenden Kräfte möglich. Auf diesem Wege können die Trajektorien der Roboterbewegung dynamisch und in Abhängigkeit der wirkenden Kräfte korrigiert werden.

Für die Erfassung und Verrechnung der Sensordaten in der Robotersteuerung ist folgender Ablauf erforderlich:

- Erfassung der Rohdaten des Sensors. In Abhängigkeit davon, in welcher Form die Daten vom Sensor geliefert werden, können an dieser Stelle Analogsignale oder bereits digitaliserte Werte vorliegen. Bei intelligenten Sensoren (mit integriertem Prozessor) sind die Rohdaten normalerweise schon vorverrechnet.
- Verrechnung der Rohdaten und Gewinnung der einzelnen Komponenten. Wenn die Rohdaten im Host-PC verrechnet werden, so gehen dazu die Kalibrierdaten des Sensors in die Rechnung ein (Kalibriertabelle und Offset).

An den Hostrechner können unterschiedliche Sensoren angeschlossen werden. Es sind Schnittstellen für die Erfassung analoger und digitalisierter Rohdaten implementiert.

CLDForceTorque-Sensor-API für Datenerfassung

Dazu dient die im vorliegenden Abschnitt beschriebene CLDForceTorqueSensor-Programmierschnittstelle. Sie ermöglicht es dem Anwendungsprogrammierer, Kraft-Momenten-Sensorik in die Steuerung einzubeziehen.

6.2 Initialisierung

Zu Beginn des Betriebes eines oder mehrerer Kraft-Momentensensoren ist eine Initialisierung erforderlich. Dies geschieht durch Aufruf der Funktion `CLDFTS_initFTSensor`. Im Ergebnis der Funktion erhält der Anwender einen eindeutigen Bezeichner (Handle) für den initialisierten Sensor. Scheitert die Initialisierung, so gibt die Auswertung des Rückgabewertes der Funktion Aufschluß über einen eventuellen Fehler.

Der Prototyp der Initialisierungsfunktion lautet:

```
HFTS CLDFTS_initFTSensor( const char* pInitString, const char*
pCalFile );
```

Format des Initialisierungsstrings

Der Initialisierungsstring, der im Parameter *pInitString* übergeben wird, ist für die unterschiedlichen Sensortypen spezifisch. Im allgemeinen hat er folgendes Format:

<*Schnittstelle*>,<*Typ*>

Es werden zwei Typen von Sensoren unterschieden (herstellerabhängig):

Typ 0: Sensorschnittstelle mit digitalisierten Daten und paralleler Datenübertragung (IMI_FTS).

Typ 1: Sensorschnittstelle mit analoger Datenübertragung zum Hostrechner und Digitalisierung im PC (AMTEC_FTS).

Für die Initialisierung eines Sensors vom Typ 0 (IMI_FTS) beispielsweise lautet der Initialisierungsstring (vorausgesetzt die Interfacekarte hat die Basisadresse 300H):

```
"300,0"
```

Folgendes Beispiel verdeutlicht den Vorgang der Initialisierung:

```
HFTS hFTS;
Char* pInitString = "300,0";
Char* pFileName = "file";
hFTS = CLDFTS_initFTSensor( pInitString, pFileName );
if( !hFTS )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

Handle hFTS

Der Wert in *hFTS* stellt einen eindeutigen Bezeichner für den angemeldeten Kraft-Momentensensor dar und wird bei allen nachfolgenden Funktionsaufrufen verwendet.

Der in *pFileName* übergebene Dateiname dient zum Einlesen der sensorspezifischen Kalibrier- und Offsetdaten. Beachten Sie, daß in dieser Angabe keine Erweiterung erwartet wird. Die Initialisierungsfunktion sucht automatisch im aktuellen Verzeichnis nach den eindeutig bezeichneten Dateien. Die Dateiendungen (*.cal - Kalibriertabelle und *.off - Offsetdaten) werden automatisch angefügt.

Damit ist die Anmeldung des ersten Kraft-Momentensensors abgeschlossen. Von diesem Zeitpunkt an können alle weiteren CLDFTS-Kommandos auf diesen Sensor, eindeutig beschrieben durch sein *Handle*, angewendet werden.

Deinitialisierung

Zur Beedigung der Arbeit mit den Kraft-Momentensensoren wird die Funktion `CLDFTS_exitFTSensor aufgerufen.`

6.3 Parametrierung und Abfrage

Der Typ FTVector

Bei Abfrage der erfaßten Sensordaten mit der Funktion `CLDFTS_getFTVector` wird das Ergebnis in einer Variable vom Typ *FTVector* hinterlegt.

Typ FTVector

	Datentyp	Bedeutung
□	`Float`	Feld von der Größe 6 zur Darstellung der Sensorwerte

Der Typ FTMatrix

Werden vom Anwender die Kalibrierdaten gesetzt oder abgefragt, benötigt er dazu eine Variable vom Typ FTMatrix.

Typ FTMatrix

	Datentyp	Bedeutung

- `Float` Feld von der Größe 6 x 6 zur Darstellung der Kalibriertabelle

Kalibriertabelle

Mit der Funktion `CLDFTS_getCalMatrix` kann der Anwender die aktuell eingestellten Kalibrierdaten abfragen. Sie werden bei der Initialisierung automatisch eingelesen. Im Falle einer Anpassung der Daten während der Laufzeit ist dies durch die Funktion `CLDFTS_setCalMatrix` möglich.

Offsetdaten

Mit der Funktion `CLDFTS_getCalOffset` kann der Anwender die aktuell eingestellten Offsetdaten abfragen. Sie werden bei der Initialisierung automatisch eingelesen. Im Falle einer Anpassung der Daten während der Laufzeit ist dies durch die Funktion `CLDFTS_setCalOffset` möglich.

Abfrage der aktuellen Werte

Mit der Funktion `CLDFTS_getFTVector` kann der Anwender die aktuellen Daten des Kraft-Momentensensors abfragen.

Folgendes Beispiel verdeutlicht diesen Vorgang:

```
CLDRet retVal;
FTVector forces;
retVal = CLDFTS_getFTVector( hFTS, forces );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

Nullpunktabgleich (Tara)

Mit der Funktion `CLDFTS_getCurOffset` kann der Anwender einen Nullpunktabgleich vornehmen. Nach Aufruf der Funktion nimmt der Sensor die aktuell anliegenden Daten zu Null an.

Schwellwerte

Mit der integrierten Schwellwertfunktion ist eine Desensibilisierung des Sensors im unteren Meßbereich möglich. Dies kann sich bei hohem Rauschanteil bzw. im Falle von Schwingungen des Meßkopfs verursacht durch die Bewegung des Roboters notwendig machen. In diesem Fall übergibt der Anwender mit der Funktion `CLDFTS_setFTThreshold` ein Feld mit den neuen Schwellwerten. Die aktuell eingestellten Schwellwerte können mit `CLDFTS_getFTThreshold` erfragt werden.

Folgendes Beispiel zeigt die Schwellwertvorgabe:

```
CLDRet retVal;
FTVector newThres = { 1.0, 1.5, 1.0, 0.4, 0.2, 0.1 };
retVal = CLDFTS_setFTThreshold( hFTS, newThres );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

6.4 Einheiten

Wie bereits bei den Funktionen der übrigen Thematiken angewendet, werden alle Parameter für die Programmierung in Einheiten des SI-Systems dargestellt, um weiterführende Berechnungen zu vereinfachen.

Die bei der ForceTorqueSensor-Schnittstelle verwendeten Einheiten sind:

Einheiten

Physikalische Größe	Einheit
□ Kraft (F_x, F_y, F_z)	**N**
□ Moment (M_x, M_y, M_z)	**Nm**

6.5 Dateien

Zur Verwendung der Funktionen der CLDForceTorqueSensor-Thematik ist folgende *include*-Datei erforderlich:

- □ `FTSAPI.H` – Vereinbarungen für die *CLDFTS*-Thematik
- □ `FTSAPI.LIB` – Statische Funktionsbibliothe zur Einbindung ins Anwenderprogramm

Darüberhinaus sind möglicherweise noch andere Dateien erforderlich, die der Lieferung beigefügt sind.

#include FTSAPI.H genügt!

Zur Vereinfachung der Programmierung genügt es, wenn Sie die Datei `FTSAPI.H` mittels einer `#include`-Anweisung einfügen – diese Datei liest selbständig alle anderen benötigten Header-Dateien ein.

Die Include-Dateien müssen sich in einem dem Projekt zugänglichen Verzeichnis befinden.

7 Grundlagen der ImpedanceControl-Schnittstelle

Im vorliegenden Abschnitt wird beschrieben, wie die Robotersteuerung unter Einbindung der Kraft-Momenten-Sensorik erweitert werden kann. Die ImpedanceControl-Schnittstelle erlaubt die impedanzgeregelte Bewegung von bis zu 32 Robotern von einer Steuerung.

7.1 Allgemeines

Erweiterung der Robotersteuerung

Impedanzgeregelte Robotersteuerung erlaubt die Ausführung von Bewegungen im kartesischen Raum unter Einbeziehung von Sensordaten. Auf diesem Wege können die Trajektorien der Roboterbewegung dynamisch und in Abhängigkeit der wirkenden Kräfte korrigiert werden.

Für die impedanzgeregelte Bewegung ist folgender Ablauf erforderlich:

- Erfassung der Sensordaten (Kräfte und Momente in 6 Komponenten).
- Transformation der Sensordaten in das vom Anwender bestimmte Bezugssystem.
- Regelkreis mit 6 Größen zur Regelung der Kraftvorgaben und Erzeugung von Korrekturwerten für die Positionsvorgaben an die Robotersteuerung.

ImpedanceControl-API

Dazu dient die im vorliegenden Abschnitt beschriebene ImpedanceControl-Programmierschnittstelle. Sie ermöglicht dem Anwender die Vorgabe von Roboterbewegungen, die unter Einbeziehung der Kraft-Momenten-Sensorik dynamische Korrekturen der Trajektorien gestatten.

7.2 Voraussetzungen

Die ImpedanceControl-API basiert auf der Roboter-Programmierschnittstelle und der ForceTorqueSensorAPI. Vor Aufruf von Befehlen der Impedanzschnittstelle sollte durch den Anwender sichergestellt worden sein, daß die genannten Schnittstellen ordnungsgemäß initialisiert wurden.

Die Impedanzregelung ist von der Roboterstruktur unabhängig. Für ihren Betrieb ist keine gesonderte Initialisierung erforderlich.

Achtung!

Vor Inbetriebnahme der Impedanzregelung müssen mindestens ein Roboter und ein zugehöriger Kraft-Momenten-Sensor initialisiert worden sein!

Die Typen PosVector und SelVector

Zur Vorgabe von Zielpositionen im Zusammenhang mit den Funktionen zur impedanzgeregelten Bewegung (`CLDImp_iConMoveTo, CLDImp_iConMoveRel`) wird eine Variable vom Typ *PosVector* verwendet.

Typ PosVector

Datentyp	Bedeutung
`Float`	Feld von der Größe 6 zur Darstellung der Zielposition

Die nachgiebigen (selektierten) Freiheitsgrade werden in einer Variable vom Typ *SelVector* übergeben.

Typ SelVector

Datentyp	Bedeutung
`UInt8`	Feld von der Größe 6 zur Darstellung des Selektionsvektors

7.3 Parametrierung der Impedanzregelung

Vorgabe der Zykluszeiten für die Interpolation

Mit der Funktion `CLDImp_setIConUpdate` kann der Anwender die Zykluszeit für die Interpolation der Bahnbewegung setzen. Die Impedanzregelung hat eine Bandbreite von 10 Hz .. 100 Hz. Das bestimmt die Grenzwerte für die Zykluszeit der Interpolation (min. 10 ms, max 100 ms). Bitte beachten Sie, daß durch den steuernden Hostrechner ebenfalls Grenzen für die Zykluszeiten gegeben werden.

Vorgabe der Reglerparameter

Für die Impedanzregelung ist ein Kraftregler mit 6 unabhängigen Dimensionen und maximal 32 Regelschleifen realisiert (für max. 32 Roboter an einer Steuerung). Die Reglerstruktur ist PID mit Stellwertbegrenzung und Anti-Wind-Up sowie gewichtetem Eingang des D-Anteils in die Regelung. Mit der Funktion `CLDImp_setForceConParams` kann der Anwender Einfluß auf die Regelungsdaten nehmen. Die Funktionsargumente haben folgende Bedeutung:

- Kp: Verstärkungsfaktor in 6 Dimensionen (Float).
- Ki: Integralfaktor in 6 Dimensionen (Float).
- Ki: Differentialfaktor in 6 Dimensionen (Float).
- minOutput: Minimaler Stellwert in 6 Dimensionen (Float).
- maxOutput: Maximaler Stellwert in 6 Dimensionen (Float).
- minIntLimit: Minimale Integrationsgrenze in 6 Dimensionen (Float).
- maxIntLimit: Maximale Integrationsgrenze in 6 Dimensionen (Float).
- diffUpdate: Zyklus für die Verrechnung des D-Anteils (Int).

Der Zyklus für die Verrechnung des D-Anteils ist als Vielfaches des eingestellten Abtastzyklus des Reglers anzugeben.

Die Funktion `CLDImp_getForceConParams` erlaubt die Abfrage der aktuell eingestellten Reglerdaten. Das folgende Beispiel verdeutlicht die Verwendung von `CLDImp_setForceConParams`.

Auswahl vorkonfigurierter Reglereinstellungen

Die ImpedanceControl-Thematik beinhaltet sinnvolle voreingestellte Daten für die einzelnen impedanzgeführten Bewegungsarten des Roboters. Der Anwender kann aus den voreingestellten Datensätzen die für die Applikation geeigneten auswählen. Dazu findet die Funktion `CLDImp_selectParamSet` Verwendung. Es sind folgende Konfigurationen des Reglers verfügbar (der jeweilige Name verdeutlicht die bevorzugten Applikationen):

- Bewegung im Kontakt (CONTACT)
- Annäherung (APPROACH)
- Ausweichende Bewegung (FOLLOW)

Für die Auswahl eines kraftausweichenden Impedanzreglers beispielsweise ist folgender Aufruf sinnvoll:

```
CLDRet retVal;
ParamSet parSet = FOLLOW;
```

```
retVal = CLDImp_selectParamSet( robId, parSet );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

7.4 Bewegungsbefehle

Impedanzgeregelte Bewegung mit Zielpunktvorgabe

Mit der Funktion `CLDImp_iConMoveTo` kann der Anwender eine impedanzgeregelte Bewegung mit Zielpunktvorgabe initiieren. Folgende Randbedingungen sind dabei zu beachten.

Vor der Bewegung wird ein Bezugssystem gewählt, in dem die Zielpunktvorgabe geschieht (lesen Sie dazu den Abschnitt Definition/Änderung des Taskkoordinatensystems auf Seite 38). Es stehen drei Möglichkeiten zur Auswahl:

- Basiskoordinatensystem (BASE_FRAME)
- Toolkoordinatensystem (TOOL_FRAME)
- Benutzerdefiniertes Taskkoordinatensystem (USER_DEFINED)

Die Zielpunktvorgabe kann absolut oder relativ erfolgen (bei relativer Zielpunktvorgabe wird der Befehl `CLDImp_iConMoveRel` angewendet). Sie beinhaltet eine kartesische Position im angegebenen Bezugssystem. Gleichzeitig wird ein Selektionsvektor übergeben, der die nachgiebigen (impedanzgeregelten) Freiheitsgrade bestimmt.

Die Bewegung erfolgt zum angegebenen Zielpunkt. Während der Ausführung der Bewegung ist der Impedanzregler bestrebt, die in den selektierten (nachgiebigen) Freiheitsgraden auftretenden Kräfte zu Null zu regeln. Auf diese Weise kann die Bewegung von der ursprünglichen Bahn abweichen. Lediglich in den nicht selektierten Freiheitsgraden ist die Bewegung des Roboters nicht nachgiebig.

Die zur Bewegung gewünschte Bahngeschwindigkeit wird als Parameter übergeben.

Für die Ausführung der Bewegung ist der 6-Größen-Regler zu parametrieren. Die Einstellung der Reglerparameter bewirkt unterschiedliche Verhaltensweisen der Steuerung. Die ImpedanceControl-Schnittstelle bietet die Möglichkeit, voreingestellte Regler zu verwenden. Zur Angabe, ob voreingestellte oder anwenderdefinierte Reglerparameter Verwendung finden sollen, dient die letzte Angabe im Funktionsaufruf.

```
CLDRet retVal;
FTVector maxForces = {...};          //Kräfte und Momente
PosVector target = {...};            //kart. Zielposition
SelVector sel = {...};               //Selektionsvektor
Float vel = 0.01;                    //Bahngeschw. in m/s
Bool par = FALSE;                    //interne Reglerparam.

retVal = CLDImp_iConMoveTo( robId, hFTS, vel, target, sel, par );
if( retVal != CLD_OK )
  ...                                // Fehlerauswertung
else
```

```
  ...                                  // OK, weiter
```

Die Ausführung der Bewegung endet zu unterschiedlichen Bedingungen. Als erstes Kriterium für die Beendigung der Bewegung dient die Zielpunktvorgabe des Anwenders. Ist der Zielpunkt nicht erreichbar, kann ein Abbruch der Bewegung aus folgenden Gründen ausgelöst werden:

- die Zielvorgabe überschreitet den Arbeitsbereich des Roboters
- eine vom Anwender vorgegebene maximale Kraftschwelle wurde überschritten (lesen Sie dazu den Abschnitt Grenzwerte für Kraftschwellen vorgeben auf Seite 36)
- es ist ein unvorhergesehener Fehler aufgetreten.

Impedanzgeregelte Annäherung mit Zielkraftvorgabe

Mit der Funktion `CLDImp_approach` kann der Anwender eine Bewegung zur Annäherung des Roboters an ein Objekt initiieren. Die Bewegung des Roboters verläuft in Richtung der im Zielkraftvektor festgelegten Vorgaben.

Die Bewegung bricht ab, wenn die gemessenen Istdaten mit den Zielvorgaben übereinstimmen. Die Übereinstimmung wird mit einem Toleranzband durch den Anwender vorgegeben. Bitte beachten Sie die Dynamik der Kraftmessung bei der Angabe des Toleranzbandes.

```
CLDRet retVal;
FTVector target = {...};
FTVector tol = {...};
Bool par = FALSE;

retVal = CLDImp_approach( robId, hFTS, target, tol, par );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

Nachgiebigkeit in allen Freiheitsgraden

Die Funktion `CLDImp_followForce` definiert eine Nachgiebigkeit in allen Freiheitsgraden. Der Aufruf dieser Funktion ruft nur dann eine Bewegung des Roboters hervor, wenn auf das System von außen Kräfte einwirken. Der Roboter weicht diesen Kräften aus.

Diese Funktion gestattet die Lösung folgender und ähnlicher Applikationen:

- Positionierung durch manuelle Führung am TCP
- Impedanzgeregeltes Greifen.

Das Abbruchkriterium für die Ausführung der Funktion wird aus diesem Grund durch die Anwendung formuliert. Ein einmaliges Aufrufen von `CLDImp_followForce` löst entsprechend den einwirkenden Kräften eine geringfügige Bewegung aus. Erst ein zyklischer Aufruf der Funktion führt zur gesamten kraftausweichenden Bewegung des Roboters.

Das folgende Beispiel illustriert die Anwendung der Funktion `CLDImp_followForce` im Zusammenhang mit einer Greifbewegung. Der Roboter ist solange impedanzgeregelt wie die Bewegung des Greifers anhält:

```
CLDRet retVal;
```

```
Float vel = ...;
Word state;
Bool par = FALSE;

CLDFTS_getCurOffset( hFTS );
CLDImp_selectParamSet( robID, FOLLOW );
CLDDrive_moveVel( gripID, vel );
do
{ retVal = CLDImp_followForce( robID, hFTS, par );
  if( retVal != CLD_OK )
    break;
  CLDDrive_getStatus( gripID, &state )
} while( state && ST_MOVING );

...
```

Abfrage der transformierten Sensordaten

Mit der Funktion `CLDFTS_getTransfSensorData` kann der Anwender die aktuellen Daten des Kraft-Momentensensors transformiert in das gewählte Bezugssystem sowie unter Berücksichtigung der Hebelwirkung zwischen TCP und Sensor erfragen.

Folgendes Beispiel verdeutlicht diesen Vorgang:

```
CLDRet retVal;
FTVector forces = {...};

retVal = CLDImp_getTransfSensorData( robID, hFTS, forces );
if( retVal != CLD_OK )
  ...                                  // Fehlerauswertung
else
  ...                                  // OK, weiter
```

7.5 Einheiten

Wie bereits bei den Funktionen der übrigen Thematiken angewendet, werden alle Parameter für die Programmierung in Einheiten des SI-Systems dargestellt, um weiterführende Berechnungen zu vereinfachen.

Die bei der ForceTorqueSensor-Schnittstelle verwendeten Einheiten sind:

Einheiten

Physikalische Größe	Einheit
□ Länge	**m**
□ Winkel	**rad**
□ Bahngeschwindigkeit	**m/s**
□ Kraft (F_x, F_y, F_z)	**N**
□ Moment (M_x, M_y, M_z)	**Nm**

7.6 Dateien

Zur Verwendung der Funktionen der ImpedanceControl-Thematik ist folgende *include*-Datei erforderlich:

- □ `DRVAPI.H` – Vereinbarungen für die *CLDDrive*-Thematik
- □ `ROBAPI.H` – Vereinbarungen für die *CLDRobot*-Thematik
- □ `FTSAPI.H` – Vereinbarungen für die *CLDFTS*-Thematik

- `IMPAPI.H` – Vereinbarungen für die *CLDImp*-Thematik
- `DRVAPI.LIB` – Statische Funktionsbibliothek für die Antriebssteuerung zur Einbindung ins Anwenderprogramm
- `ROBAPI.LIB` – Statische Funktionsbibliothek für Roboter-Funktionen
- `FTSAPI.LIB` – Statische Funktionsbibliothek für Sensor-Funktionen
- `IMPAPI.LIB` – Statische Funktionsbibliothek zur Einbindung ins Anwenderprogramm

Darüberhinaus sind möglicherweise noch andere Dateien erforderlich, die der Lieferung beigefügt sind.

#include IMPAPI.H genügt!

Zur Vereinfachung der Programmierung genügt es, wenn Sie die Datei `FTSAPI.H` mittels einer `#include`-Anweisung einfügen – diese Datei liest selbständig alle anderen benötigten Header-Dateien ein.

Die Include-Dateien müssen sich in einem dem Projekt zugänglichen Verzeichnis befinden.

8 Referenz

Im folgenden Referenzteil werden alle Funktionen erklärt. Ihre Parameter und Rückgabewerte sowie die Bedeutung des Funktionsaufrufes werden beschrieben.

8.1 Übersicht

Initialisierungs- und Verwaltungsfunktionen Seite 53

- ☐ `CLD_initDll` Initialisierung der Treiber-DLL
- ☐ `CLD_exitDll` Beenden der Treiber-DLL
- ☐ `CLD_errorMsg` Generierung einer Fehlermeldung
- ☐ `CLDDrive_initManip` Initialisierung des Manipulators
- ☐ `CLDDrive_initModule` Initialisierung eines Antriebsmoduls
- ☐ `CLDDrive_exitManip` Freigeben des Manipulators

Kommandofunktionen Seite 57

- ☐ `CLDDrive_syncModule` Referenzpunkt anfahren
- ☐ `CLDDrive_haltModule` Antriebsmodul anhalten
- ☐ `CLDDrive_resetModule` Rücksetzen des Modulstatus
- ☐ `CLDDrive_freeDrvModule` Antriebsmodul aus Endlagen freifahren
- ☐ `CLDDrive_commResetModle` Kommunikationsverbindung zurücksetzen
- ☐ `CLDDrive_commResetManip` Kommunikationsverbindung zurücksetzen

Statusfunktionen Seite 61

- ☐ `CLDDrive_getStatus` allgemeine Funktion zur Statusabfrage
- ☐ `CLDDrive_querySyncEnd` Referenzpunkt erreicht?
- ☐ `CLDDrive_queryEndPos` Endpunkt der Bewegung erreicht?
- ☐ `CLDDrive_queryFreeDrive` Antriebsmodul aus Endlagen freigefahren?

Bewegungsfunktionen Seite 63

- ☐ `CLDDrive_moveRamp` Bewegungskommando im Rampenmodus
- ☐ `CLDDrive_movePos` Bewegungskommando im Positionsmodus
- ☐ `CLDDrive_moveStep` Bewegungskommando im Schrittmodus
- ☐ `CLDDrive_moveVel` Bewegungskommando im Geschwindigkeits-modus

Abfragen des aktuellen Modulzustands Seite 67

- ☐ `CLDDrive_getPos` aktuelle Position abfragen
- ☐ `CLDDrive_getVel` aktuelle Geschwindigkeit abfragen
- ☐ `CLDDrive_getTow` aktuellen Schleppfehler abfragen
- ☐ `CLDDrive_getCurrent` aktuelle Stromaufnahme abfragen
- ☐ `CLDDrive_getTemperature` aktuelle Innentemperatur des Moduls abfragen

- ☐ `CLDRobot_setMaxDynRobot` Grenzwerte für Roboterdynamik setzen
- ☐ `CLDRobot_getMaxDynRobot` Grenzwerte für Roboterdynamik abfragen
- ☐ `CLDRobot_setDriveRanges` Grenzwerte für Bewegungsbereiche der Achsen setzen
- ☐ `CLDRobot_getDriveRanges` Grenzwerte für Bewegungsbereiche der Achsen abfragen
- ☐ `CLDRobot_setMaxDynDrives` Grenzwerte für Bewegungsdynamik der Achsen setzen
- ☐ `CLDRobot_getMaxDynDrives` Grenzwerte für Bewegungsdynamik der Achsen abfragen
- ☐ `CLDRobot_setMaxTow` Grenzwerte für maximale Schleppfehler der Achsen setzen
- ☐ `CLDRobot_getMaxTow` Grenzwerte für maximale Schleppfehler der Achsen abfragen
- ☐ `CLDRobot_setMaxCurrent` Grenzwerte für maximale Stromaufnahme der Achsen setzen
- ☐ `CLDRobot_getMaxCurrent` Grenzwerte für maximale Stromaufnahme der Achsen abfragen
- ☐ `CLDRobot_setMaxTemperature` Grenzwerte für maximale Motortemperatur der Achsen setzen
- ☐ `CLDRobot_getMaxTemperature` Grenzwerte für maximale Motortemperatur der Achsen abfragen
- ☐ `CLDRobot_setLimCurrent` Strombegrenzung für die Achsregelung zuschalten
- ☐ `CLDRobot_getRobCoord` Abfrage der aktuellen Position in Achskoordinaten
- ☐ `CLDRobot_getToolCoord` Abfrage der aktuellen Position in Weltkoordinaten
- ☐ `CLDRobot_getRobVel` Überwachung der Robotergeschwindigkeit
- ☐ `CLDRobot_getActDynRobot` Überwachung der aktuellen Roboterdynamik
- ☐ `CLDRobot_getTow` Überwachung des Schleppfehlers pro Achse
- ☐ `CLDRobot_getCurrent` Überwachung der Stromaufnahme pro Achse
- ☐ `CLDRobot_getTemperature` Überwachung der Achstemperaturen

ForceTorqueSensor-Funktionen Seite 90

- ☐ `CLDFTS_initFTSensor` Initialisierung des Sensors
- ☐ `CLDFTS_exitFTSensor` Deinitialisierung des Sensors
- ☐ `CLDFTS_getFTThreshold` Meßschwelle lesen
- ☐ `CLDFTS_setFTThreshold` Meßschwelle setzen
- ☐ `CLDFTS_getCalMatrix` Kalibriertabelle erfragen
- ☐ `CLDFTS_setCalMatrix` Kalibriertabelle anpassen
- ☐ `CLDFTS_getCalOffset` Offset-Grundeinstellung erfragen
- ☐ `CLDFTS_setCalOffset` Offset-Grundeinstellung anpassen
- ☐ `CLDFTS_getFTVector` Aktuelle Sensordaten erfassen
- ☐ `CLDFTS_getCurOffset` Nullpunktabgleich (Tara)

ImpedanceControl-Funktionen Seite 92

- ☐ `CLDImp_setIConUpdate` Setzen der Zykluszeit für die Interpolation

- ☐ `CLDImp_getForceConParams` Abfrage der eingestellten Reglerdaten
- ☐ `CLDImp_setForceConParams` Reglerdaten anpassen
- ☐ `CLDImp_selectParamSet` Auswahl voreingestellter Reglerdaten
- ☐ `CLDImp_iConMoveTo` Impedanzgeregelte Bewegung mit absoluter Zielpunktvorgabe
- ☐ `CLDImp_iConMoveRel` Impedanzgeregelte Bewegung mit relativer Zielpunktvorgabe
- ☐ `CLDImp_approach` Impedanzgeregelte Annäherungsbewegung
- ☐ `CLDImp_followForce` Kraftausweichende Roboterbewegung
- ☐ `CLDImp_getTransfSensorData` Erfassung transformierter Sensordaten

8.2 Initialisierungs- und Verwaltungsfunktionen

Die im folgenden beschriebenen Funktionen dienen der Initialisierung der CronoCom und des angeschlossenen Manipulators mit seinen Antriebsmodulen.

CLDXtrnl_init

Aufgabe Initialisierung der CronoCom

Prototyp
```
CLDRet CLDXtrnl_init( CLDXtrnlID* pXtrnlID, Char* pInitString );
```

Beschreibung Diese Funktion initialisiert die CronoCom anhand der in der Zeichenkette *pInitString* übergebenen Parameter. Der Aufbau des Initialisierungsstrings hängt von der Installation der PC-Karte CronoCom ab (Basisadresse, Interrrupt, Dual-Port-RAM).

Rückgabewert

`CLD_OK`	CronoCom konnte erfolgreich initialisiert werden.
`CLDERR_LIBRARYNOTFOUND`	System-Dll konnte vom Betriebssystem nicht gefunden werden (gilt nur unter MS-Windows).
`CLDERR_BADPARAM`	Falsche Parameter im Initialisierungsstring oder falsch formatiert (siehe entsprechenden Abschnitt im Anhang).

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDXrtnl_exit

Aufgabe Deinitialisierung der CronoCom

Prototyp
```
CLDRet CLD_exitDll( CLDXtrnlID xtrnlID );
```

Beschreibung Diese Funktion gibt die von der CronoCom belegten Ressourcen (Basisadresse, Interrupt, Dual-Port-RAM) frei.

Rückgabewert

`CLD_OK`	CronoCom wurde erfolgreich freigegeben.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDDrive_initManip

Aufgabe Initialisierung des Handhabungssystems

Prototyp
```
CLDRet CLDDrive_initManip( Short* pNumDrives, const Char* pInitString
);
```

Beschreibung Diese Funktion initialisiert das Handhabungssystem anhand der in der Zeichenkette *pInitString* übergebenen Parameter. Der Aufbau des Initialisierungsstrings hängt von der konkreten Treiber-Implementierung ab (siehe entsprechenden Abschnitt im Anhang).

Über den Zeiger *pNumDrives* wird die Zahl der erkannten Antriebsmodule an das Anwenderprogramm zurückgeliefert. Bei allen weiteren Funktionsaufrufen wird das angesprochene Antriebsmodul mit Hilfe einer laufenden Nummer *driveId* identifiziert, wobei die Vorschrift $0 \leq driveId < (*pNumDrives)$ gilt.

Diese Funktion sollte unmittelbar nach dem Laden der Treiber-DLL aufgerufen werden.

Rückgabewert

`CLD_OK`	Handhabungssystem konnte erfolgreich initialisiert werden.
`CLDERR_DRIVE_BADANSWER`	Handhabungssystem wurde nicht korrekt erkannt.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDDrive_initModule

Aufgabe Vorbereitung eines einzelnen Antriebsmoduls

Prototyp
```
CLDRet CLDDrive_initModule( CLDDriveID driveId );
```

Beschreibung Diese Funktion bereitet das Antriebsmodul, das durch *driveId* spezifiziert wird, auf den Betrieb vor. Diese Funktion muß nach `CLDDrive_initManip` und vor der ersten Verwendung einer anderen antriebsmodul-spezifischen Funktion ausgeführt werden.

Rückgabewert

`CLD_OK`	Antriebsmodul wurde vorbereitet.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch Abschnitt *Initialisierung der Module* auf Seite 21

CLDDrive_exitManip

Aufgabe Beendigung der Kommunikation mit dem Handhabungssystem

Prototyp
```
CLDRet CLDDrive_exitManip( Void );
```

Beschreibung Nach der Verwendung der Manipulator-Funktionen sollte vor dem Freigeben der DLL durch das Betriebssystem diese Funktion aufgerufen werden. Dadurch wird sichergestellt, daß nachfolgende Initialisierungen erfolgreich ausgeführt werden können.

Beispiel Im folgenden Beispiel werden die Handhabungsstruktur und die Treiber-DLL freigegeben.

```
CLDRet retVal = CLDDrive_exitManip();
if( retVal != CLD_OK )                // Fehlerauswertung
  ...
else
  FreeLibrary ( hCLDDriveDLL );
```

Rückgabewert

`CLD_OK`	Handhabungssystem wurde für andere Benutzung freigegeben.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

8.3 Kommandofunktionen

Die folgenden Funktionen stellen komplexe Kommandos dar, die das Antriebsmodul selbständig ausführt.

CLDDrive_syncModule

Aufgabe Referenzpunkt anfahren

Prototyp
```
CLDRet CLDDrive_syncModule( CLDDriveID driveId );
```

Beschreibung Mit Aufruf dieser Funktion fährt das Antriebsmodul *driveId* automatisch seinen Referenzpunkt an.

Vor dem ersten Fahrkommando muß das Antriebsmodul synchronisiert werden, das heißt es muß seinen Referenzpunkt anfahren, um einen absoluten Bezug zum umgebenden Koordinatensystem zu erhalten.

Die Funktion `CLDDrive_syncModule` startet den Vorgang des selbständigen Anfahrens des Referenzpunktes. Mit Hilfe der Funktion `CLDDrive_querySyncEnd` kann vom Anwenderprogramm aus der Abschluß des Vorgangs festgestellt werden.

Nach dem erfolgreichen Anfahren des Referenzpunktes werden Bewegungskommandos erst wirksam, wenn der Sicherheitszustand im Modul aufgehoben wurde (siehe `CLDDrive_resetModule`).

Rückgabewert

`CLD_OK`	Anfahren des Referenzpunktes konnte erfolgreich begonnen werden.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Beispiel 1 Das folgende vereinfachte Beispiel demonstriert das Anfahren des Referenzpunktes unter Nutzung einer Programmschleife.

Das Anfahren des Referenzpunktes kann mehrere Sekunden dauern!

Da das Anfahren des Referenzpunktes mehrere Sekunden dauern kann, beachten Sie bitte, daß Anwenderprogramme bei kooperativem Multitasking (zum Beispiel unter Windows 3.x) während dieses Vorgangs das gesamte System blockieren würden! Ebenso hätte der Anwender keine Kontrolle über den Vorgang.

```
CLDRet retVal = CLDDrive_syncModule( i );
if( retVal == CLD_OK )
{ Bool sync = FALSE;
  ...
  while( (retVal = CLDDrive_querySyncEnd( i, &sync )) == CLD_OK &&
!sync );
  ...
```

Die Endlosschleife wird abgebrochen, wenn ein Fehler auftritt oder der Referenzpunkt erreicht wurde.

Beispiel 2 Folgendes Beispiel zeigt das Anfahren des Referenzpunktes mit Hilfe von WM_TIMER-Nachrichten unter Windows.

```
CLDRet retVal;
Bool sync;
```

```
...
case WM_COMMAND:
  ...                                   // Referenzpunkt anfahren
  retVal = CLDDrive_syncModule( i );
  ...
  break;
case WM_TIMER:
  ...                                   // Referenzpunkt erreicht?
  sync = FALSE;
  retVal = CLDDrive_querySyncEnd( i, &sync );
  if( retVal != CLD_OK )
    ...                                 // FEHLER!
  else if( sync )
    ...                                 // OK, Modul synchronisiert.
  else
    ...                                 // Vorgang dauert noch an ...
```

Die Nachricht WM_TIMER sollte solange den Vorgang überwachen – beispielsweise im zeitlichen Abstand von 200 Millisekunden – bis ein Fehler festgestellt oder der Referenzpunkt erreicht wurde. Durch diese Trennung des Vorgangs in *Starten* und *zyklisches Überprüfen* wird das System nicht unnötig lange blockiert.

Siehe auch CLDDrive_querySyncEnd, CLDDrive_resetModule

CLDDrive_haltModule

Aufgabe Antriebsmodul anhalten

Prototyp

```
CLDRet CLDDrive_haltModule( CLDDriveID driveId );
```

Beschreibung Diese Funktion bewirkt, daß das Antriebsmodul *driveId* automatisch anhält.

Nach der Ausführung dieser Funktion geht das Antriebsmodul automatisch in einen Sicherheitszustand über. In diesem Zustand werden vom Modul keine Bewegungskommandos angenommen. Zur Aufhebung dieses Zustands und muß das Kommando `CLDDrive_resetModule` gesendet werden.

Die Funktion `CLDDrive_haltModule` startet den Vorgang des selbständigen Anhaltens des Antriebsmoduls. Mit Hilfe der Funktion `CLDDrive_queryEndPos` kann vom Anwenderprogramm aus der Stillstand des Moduls festgestellt werden.

Rückgabewert

`CLD_OK`	Anhalten des Moduls konnte erfolgreich begonnen werden.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_queryEndPos, CLDDrive_resetModule

CLDDrive_resetModule

Aufgabe Rücksetzen des Modulstatus

Prototyp

```
CLDRet CLDDrive_resetModule( CLDDriveID driveId );
```

Beschreibung Diese Funktion bewirkt, daß das Antriebsmodul *driveId* den Sicherheitszustand aufhebt und damit Bewegungskommandos erlaubt werden.

Folgende Ereignisse führen zur Einnahme des Sicherheitszustands im Antriebsmodul:

- Einschalten des Moduls,
- Feststellung eines Fehlers,
- Auslösung eines HALT-Kommandos durch das Anwenderprogramm (durch Aufruf der Funktion `CLDDrive_haltModule`).

Erst nach Ausführung der Funktion `CLDDrive_resetModule` werden vom Antriebsmodul Bewegungskommandos angenommen.

Die Funktion `CLDDrive_resetModule` scheitert, wenn die Bedingungen, die zu einem Fehler führten, noch vorliegen oder wenn nach dem Einschalten noch nicht der Referenzpunkt des Moduls angefahren wurde. Mit Hilfe der Funktion `CLDDrive_getStatus` kann vom Anwenderprogramm aus der Status des Moduls festgestellt werden.

Rückgabewert

`CLD_OK`	RESET-Kommando wurde erfolgreich an das Modul gesendet.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_syncModule, CLDDrive_haltModule, CLDDrive_getStatus

CLDDrive_freeDriveModule

Aufgabe Antriebsmodul aus Endlagen freifahren

Prototyp

```
CLDRet CLDDrive_freeDriveModule( CLDDriveID driveId );
```

Beschreibung Durch Ausführung dieser Funktion fährt das Antriebsmodul *driveId* automatisch aus einer Software- oder Hardwareendlage in den erlaubten Bereich.

Im normalen Betrieb – sofern sich die programmierten Softwareendlagen innerhalb der Hardwareendlagen befinden – kann das Antriebsmodul diesen erlaubten Bereich nicht überschreiten. Falls eine Bewegung nach außerhalb der Softwareendlagen ausgeführt werden soll, begrenzt das Modul diese Bewegung selbständig. (Ein solcher Zustand wird im Statuswort mit dem Bit `ST_WARN_BEYONDSOFTLIMIT` angezeigt.)

In bestimmten Situationen kann sich jedoch das Modul außerhalb des erlaubten Bereiches befinden. Hier führt die Funktion `CLDDrive_freeDriveModule` zum Beginn des automatischen Ausfahrens aus Endlagen. Mit Hilfe der Funktion `CLDDrive_queryFreeDrive` kann der Abschluß des Vorgangs geprüft werden. Da das Freifahren mehrere Sekunden dauern kann, ist die Vorgehensweise – um das System nicht unnötig lange zu blockieren – analog dem beim Anfahren des Referenzpunktes (siehe `CLDDrive_syncModule`).

Der Vorgang ist abgeschlossen, wenn das Modul aus Hardware- und Softwareendlagen herausgefahren ist oder ein Fehler auftritt.

Nach dem Freifahren muß vor dem ersten Bewegungskommando der Sicherheitszustand im Modul aufgehoben werden (siehe `CLDDrive_resetModule`).

Rückgabewert

`CLD_OK`	Freifahren des Moduls aus Soft- und Hardwareendlagen konnte erfolgreich begonnen werden.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getStatus, CLDDrive_queryFreeDrive, CLDDrive_resetModule

CLDDrive_commResetModule

Aufgabe Kommunikationsverbindung zurücksetzen

Prototyp
```
CLDRet CLDDrive_commResetModule( CLDDriveID driveId );
```

Beschreibung Die Funktion setzt die Kommunikationsverbindung zum Antriebsmodul zurück. Dies ist erforderlich, wenn ein bereits initialisiertes Handhabungssystem nochmals initialisiert werden soll, ohne vorher die Stromversorgung auszuschalten. Die Funktion ist für alle angeschlossenen Module auszuführen.

Achtung! Ab der Treiberversion 2.13 ist dieser Funktionsaufruf nicht mehr erforderlich, sondern es kann die Funktion `CLDDrive_commResetManip` verwendet werden (siehe folgende Funktionsbeschreibung).

Rückgabewert

`CLD_OK`	Kommunikationsverbindung wurde zurückgesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_initManip, CLDDrive_commResetManip

CLDDrive_commResetManip

Aufgabe Kommunikationsverbindung zum gesamten Manipulator zurücksetzen

Prototyp
```
CLDRet CLDDrive_commResetManip( Void );
```

Beschreibung Die Funktion setzt die Kommunikationsverbindung zu allen Antriebsmodulen zurück. Dies ist erforderlich, wenn ein bereits initialisiertes Handhabungssystem nochmals initialisiert werden soll, ohne vorher die Stromversorgung auszuschalten.

Achtung! Ab der Treiberversion 2.14 ist dieser Funktionsaufruf nicht mehr erforderlich, wenn die Verbindung zum Manipulator mit der Funktion `CLDDrive_initManip` hergestellt wird. Ein eventuell vorher schon verbundener Manipulator muß jedoch zuerst mit der Funktion `CLDDrive_exitManip` getrennt worden sein.

Rückgabewert

`CLD_OK`	Kommunikationsverbindung wurde zurückgesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.

	CLDERR_DRIVE_RECEIVE	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_initManip, CLDDrive_exitManip

8.4 Statusfunktionen

Die im folgenden genannten Funktionen beschreiben den Status des Antriebsmoduls.

CLDDrive_getStatus

Aufgabe Allgemeine Funktion zur Statusabfrage

Prototyp
```
CLDRet CLDDrive_getStatus( CLDDriveID driveId, Word* pStat );
```

Beschreibung Die Funktion fordert das Statuswort des Antriebsmoduls *driveId* ab. Dies ist ein 32-Bit-Wort, das in der Variablen, auf die der Zeiger *pStat* zeigt, abgelegt wird. Jedes Bit in diesem Statuswort hat eine bestimmte Bedeutung, die weiter unten erklärt wird.

Hinweis: Für die häufig benötigten Abfragen zum Erreichen des Referenzpunktes, der Rampenbewegungen und zum Freifahren existieren vereinfachte Statusfunktionen (`CLDDrive_querySyncEnd`, ...).

Das Statuswort ist im Kapitel 4.3 im Abschnitt *Status des Antriebsmoduls abfragen* auf Seite 23 beschrieben.

Rückgabewert

`CLD_OK`	Status des Moduls wurde abgefordert.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_querySyncEnd, CLDDrive_queryEndPos, CLDDrive_queryFreeDrive

CLDDrive_querySyncEnd

Aufgabe Prüfung, ob Referenzpunkt erreicht

Prototyp
```
CLDRet CLDDrive_querySyncEnd( CLDDriveID driveId, Bool* pBool );
```

Beschreibung Die Funktion vereinfacht die Prüfung auf Erreichen des Referenzpunktes. Sie ist nur sinnvoll nach Ausführung der Funktion `CLDDrive_syncModule`. Das Ergebnis der Prüfung (`TRUE` == Referenzpunkt erreicht) wird über *pBool* zurückgeliefert.

Rückgabewert

`CLD_OK`	Prüfung ist erfolgt – in der Variablen, auf die der Zeiger *pBool* zeigt, befindet sich das Ergebnis der Prüfung.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

	CLDERR_DRIVE_MODULEERROR	Fehler im Antriebsmodul während des Anfahrens des Referenzpunktes.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Beispiel Ein ausführliches Beispiel ist zur Funktion `CLDDrive_syncModule` zu finden.

Siehe auch CLDDrive_syncModule

CLDDrive_queryEndPos

Aufgabe Prüfung, ob Endpunkt der Bewegung erreicht

Prototyp
```
CLDRet CLDDrive_queryEndPos( CLDDriveID driveId, Bool* pBool );
```

Beschreibung Die Funktion vereinfacht die Prüfung auf Erreichen des Endpunktes einer Rampenbewegung. Sie ist nur sinnvoll nach Ausführung der Funktion `CLDDrive_moveRamp`. Darüberhinaus wird nach Auslösung des HALT-Kommandos oder nach Auftreten eines Fehlers im Moduls angezeigt, ob das Modul seinen Stillstand erreicht hat. Das Ergebnis der Prüfung (`TRUE` == Endpunkt erreicht) wird über *pBool* zurückgeliefert.

Rückgabewert

`CLD_OK`	Prüfung ist erfolgt – in der Variablen, auf die der Zeiger *pBool* zeigt, befindet sich das Ergebnis der Prüfung.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.
`CLDERR_DRIVE_MODULEERROR`	Fehler im Antriebsmodul während des Anfahrens des Referenzpunktes.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Beispiel Ein ausführliches Beispiel ist zur Funktion `CLDDrive_moveRamp` zu finden.

Siehe auch CLDDrive_moveRamp

CLDDrive_queryFreeDrive

Aufgabe Prüfung, ob Antriebsmodul aus Endlagen freigefahren

Prototyp
```
CLDRet CLDDrive_queryFreeDrive( CLDDriveID driveId, Bool* pBool );
```

Beschreibung Die Funktion vereinfacht die Prüfung auf Freifahren aus Endlagen. Sie ist nur sinnvoll nach Ausführung der Funktion `CLDDrive_freeDriveModule`. Das Ergebnis der Prüfung (`TRUE` == aus Endlagen freigefahren) wird über *pBool* zurückgeliefert.

Rückgabewert

`CLD_OK`	Prüfung ist erfolgt – in der Variablen, auf die der Zeiger *pBool* zeigt, befindet sich das Ergebnis der Prüfung.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.
`CLDERR_DRIVE_MODULEERROR`	Fehler im Antriebsmodul während des Freifahrens aus Software- und Hardware-endlagen.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_freeDriveModule

8.5 Bewegungsfunktionen

Die folgenden Funktionen lösen eine Bewegung des Antriebsmoduls aus.

CLDDrive_moveRamp

Aufgabe Bewegungskommando im Rampenmodus

Prototyp

```
CLDRet CLDDrive_moveRamp( CLDDriveID driveId, Float pos, Float vel,
Float acc );
```

Beschreibung Die Funktion löst eine Bewegung des Antriebsmoduls *driveId* im Rampenmodus aus. Dazu wird die Zielposition *pos* mit der maximalen Geschwindigkeit *vel* selbständig angefahren. Die Beschleunigung erfolgt mit dem in *acc* angegebenen Wert. Falls sich das Modul zum Zeitpunkt des Funktionsaufrufes bereits in einer Bewegung befand, wird mit der angegebenen Dynamik unmittelbar die neue Zielposition angefahren.

Die angegebenen Werte für Geschwindigkeit und Beschleunigung setzen dauerhaft die im Modul eingestellten Werte der aktuellen Dynamik, indem intern die Funktion `CLDDrive_setActDyn` aufgerufen wird.

Die Zielposition *pos* bezeichnet den Abstand zum Referenzpunkt. Die Werte *pos*, *vel* und *acc* werden in Einheiten des SI-Systems angegeben (siehe *Einheiten*). Falls höhere als die aktuell zulässigen Maximalwerte angegeben werden, werden diese automatisch auf die Maximalwerte begrenzt (Funktion `CLDDrive_getMaxDyn`).

Die Funktion schlägt fehl, wenn

- das Modul noch nicht seinen Referenzpunkt angefahren hatte und zurückgesetzt wurde (siehe `CLDDrive_syncModule` und `CLDDrive_resetModule`),
- sich das Modul nach einem Fehler im Sicherheitszustand befindet (siehe `CLDDrive_getStatus`),
- sich die geforderte Zielposition außerhalb der Softwareendlagen befindet.

Das bedeutet, daß die Funktion scheinbar korrekt ausgeführt wurde, sich das Antriebsmodul jedoch nicht bis zur Zielposition bewegt. Aus diesem Grund ist es ratsam, vor Ausführung der Funktion den Modulstatus zu überprüfen (Funktion `CLDDrive_getStatus`). Bei folgendem Bitmuster ist die Funktion korrekt ausführbar, das heißt die Bewegung wird (zumindest bis zur Endlagenposition) stattfinden:

Bit im Statuswort	erforderlicher Zustand
☐ `ST_HALTED`	darf nicht gesetzt sein
☐ `ST_SYNCHRONIZED`	muß gesetzt sein
☐ `ST_WARN_BEYONDSOFTLIMIT`	falls gesetzt, befindet sich die geforderte Zielposition außerhalb der Softwareendlagen, Bewegung wird begrenzt (siehe Text)
☐ `ST_ERROR`	darf nicht gesetzt sein

Das Modul wird an den Endlagen angehalten

Falls sich die geforderte Zielposition außerhalb der Softwareendlagen befindet, wird das Modul entsprechend der Funktionsparameter bewegt, soweit es die Softwareendlagen zulassen. Unmittelbar vor Erreichen der Softwareendlagen wird das Modul mit der programmierten Maximalbeschleunigung abgebremst, so daß die vorgegebene Endlage nicht überschritten wird. Dieser Zustand wird mit dem Bit `ST_WARN_BEYONDSOFTLIMIT` im Statuswort angezeigt, das mit Hilfe der Funktion `CLDDrive_getStatus`

abgefordert werden kann. Das Modul verharrt an dieser Position, bis eine programmierte Bewegung wieder nach innerhalb der Endlagenbegrenzung weist.

Rückgabewert

`CLD_OK`	Bewegung wurde ausgelöst.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Beispiel

Im folgenden Beispiel soll ein Linearmodul im Rampenmodus mit einer maximalen Geschwindigkeit von 45 mm/s bis zur absoluten Position 120 mm bewegt werden. Die Beschleunigung beträgt 0,1 m/s². Um das System während der Bewegung nicht zu blockieren, wird mit Windows-Nachrichten gearbeitet.

```
Float pos = 0.120, vel = 0.045, acc = 0.100;
Bool ready;
...
case WM_COMMAND:
  ...                                   // Zielposition anfahren
  retVal = CLDDrive_moveRamp( driveId, pos, vel, acc );
  ...
  break;
case WM_TIMER:
  ...                                   // Endposition erreicht?
  ready = FALSE;
  retVal = CLDDrive_queryEndPos( driveId, &ready );
  if( retVal != CLD_OK )
    ...                                 // FEHLER!
  else if( ready )
    ...                                 // OK, Zielposition erreicht.
  else
    ...                                 // Vorgang dauert noch an
oder
                                        //  prüfen, ob Fehler oder
                                        //  Softwareendlage ...
```

Siehe auch CLDDrive_queryEndPos, CLDDrive_move...

CLDDrive_movePos

Aufgabe Bewegungskommando im Positionsmodus

Prototyp

```
CLDRet CLDDrive_movePos( CLDDriveID driveId, Float pos );
```

Beschreibung Die Funktion löst eine Bewegung des Antriebsmoduls *driveId* im Positionsmodus aus. Dazu wird die Zielposition *pos* nach dem Aufruf der Funktion **sofort** an den im Betriebssystem des Antriebsmoduls arbeitenden Lageregler übergeben. Diese Betriebsart erfordert daher die Kenntnis der aktuellen Position und der vom Modul beherrschbaren Geschwindigkeits- und Beschleunigungswerte.

Diese Funktion dient der Generierung von beliebigen Bewegungsprofilen, beispielsweise für die Bahnsteuerung komplexer Handhabungssysteme. Zu beachten ist, daß nach der Ausführung eines Bewegungskommandos im Positionsmodus zyklisch in kurzen Abständen weitere Positionskommandos an das Antriebsmodul gesendet werden müssen. Ansonsten würde das Modul alternierend um die angegebene Position schwingen, bis es zum Stillstand kommt. Andere Möglichkeiten, diesen Modus zu verlassen

bestehen im Senden eines Bewegungskommandos in einem anderen Modus oder im Auslösen von HALT.

Der zeitliche Abstand von Positionskommandos sollte je nach Typ des Antriebsmoduls 5...10 ms nicht überschreiten. Aus diesem Grund eignet sich der Modus nicht für Kommunikationsstrecken mit geringer Übertragungsrate. In diesem Fall sollte die Funktion `CLDDrive_moveStep` verwendet werden.

Die unter `CLDDrive_moveRamp` vorgestellten Informationen über das Fehlschlagen der Funktion, über den erforderlichen Modulstatus und zum Verhalten vor Softwareendlagen gelten auch hier.

Rückgabewert

`CLD_OK`	Bewegung wurde ausgelöst.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_move...

CLDDrive_moveStep

Aufgabe Bewegungskommando im Schrittmodus

Prototyp

```
CLDRet CLDDrive_moveStep( CLDDriveID driveId, Float pos, UShort step
);
```

Beschreibung Die Funktion bewegt das Antriebsmodul *driveId* im Schrittmodus. Dabei wird versucht, die angegebene Zielposition *pos* innerhalb von *step* Millisekunden anzufahren. Die berechnete Geschwindigkeit ist

v = (*pos* - <aktuelle Position>) / *step*.

Die im Antriebsmodul integrierte Bewegungssteuerung nimmt selbständig eine eventuell erforderliche Beschleunigung auf die errechnete Geschwindigkeit vor. Hierfür werden die vorgegebenen maximalen Dynamik-Werte verwendet. Sofern die maximalen Dynamikwerte nicht überschritten werden, sorgt die Bewegungssteuerung dafür, daß die Zielposition zum vorgegebenen Zeitpunkt erreicht wird, selbst wenn der Wert der resultierenden Geschwindigkeit kurzzeitig höher als der der berechneten ist.

Falls spätestens bis Ablauf von *step* Millisekunden kein neues Schrittkommando oder ein Kommando einer anderen Betriebsart gesendet wurde, hält das Modul selbständig mit der aktuellen (Brems-) Beschleunigung an.

Die unter `CLDDrive_moveRamp` vorgestellten Informationen über das Fehlschlagen der Funktion, über den erforderlichen Modulstatus und zum Verhalten vor Softwareendlagen gelten auch hier.

Rückgabewert

`CLD_OK`	Bewegung wurde ausgelöst.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_move...

CLDDrive_moveVel

Aufgabe Bewegungskommando im Geschwindigkeitsmodus

Prototyp
```
CLDRet CLDDrive_moveVel( CLDDriveID driveId, Float vel );
```

Beschreibung Das Antriebsmodul *driveId* wird in den Geschwindigkeitsmodus überführt. Das Modul beschleunigt mit der maximal zulässigen Beschleunigung selbständig auf die neue Geschwindigkeit *vel*. Die angegebene Geschwindigkeit *vel* darf nicht größer als die programmierte maximale Geschwindigkeit sein, ansonsten wird *vel* begrenzt.

Die Geschwindigkeit *vel* wird je nach Modultyp in **m/s** oder **rad/s** angegeben.

Das Modul bewegt sich kontinuierlich mit der angegebenen Geschwindigkeit, bis ein anderes Bewegungskommando oder ein HALT-Kommando gesendet wird oder bis die Softwareendlagen erreicht werden.

Die unter `CLDDrive_moveRamp` vorgestellten Informationen über das Fehlschlagen der Funktion, über den erforderlichen Modulstatus und zum Verhalten vor Softwareendlagen gelten auch hier.

Rückgabewert

`CLD_OK`	Bewegung wurde ausgelöst.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_move...

8.6 Abfragen des aktuellen Modulzustands

Mit Hilfe den folgenden Funktionen ist es möglich, den aktuellen Status des Antriebsmoduls abzufragen.

CLDDrive_getPos

Aufgabe Holt die aktuelle Position des Moduls

Prototyp
```
CLDRet CLDDrive_getPos( CLDDriveID driveId, Float* pPos );
```

Beschreibung Die aktuelle Position des Antriebsmoduls *driveId* wird über den Zeiger *pPos* übergeben. Die aktuelle Position ist die von der im Betriebssystem des Moduls integrierten Bewegungssteuerung berechnete Position entsprechend des letzten Bewegungskommandos. Zur Ermittlung der vom Geber des Antriebsmoduls gemessenen Ist-Position ist der Schleppfehler (siehe `CLDDrive_getTow`) abzuziehen.

Hinweis: Der zulässige Bereich der Modul erreichbaren Positionen ist mit Hilfe der Funktion `CLDDrive_setRange` programmierbar.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Position wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getVel, CLDDrive_getTow

CLDDrive_getVel

Aufgabe Holt die aktuelle Geschwindigkeit des Moduls

Prototyp
```
CLDRet CLDDrive_getVel( CLDDriveID driveId, Float* pVel );
```

Beschreibung Die aktuelle Geschwindigkeit des Antriebsmoduls *driveId* wird über den Zeiger *pVel* übergeben. Dies ist die aus den Vorgaben der Bewegungssteuerung resultierende Soll-Geschwindigkeit des Moduls. Die Ist-Geschwindigkeit ist aus mehreren Ist-Positionen zu bestimmen (siehe `CLDDrive_getPos`).

Hinweis: Die zulässige Maximal-Geschwindigkeit des Moduls ist mit Hilfe der Funktion `CLDDrive_setMaxDyn` programmierbar.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**.

Rückgabewert

`CLD_OK`	Aktuelle Geschwindigkeit wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getPos

CLDDrive_getTow

Aufgabe Holt den aktuellen Schleppfehler des Moduls

Prototyp
```
CLDRet CLDDrive_getTow( CLDDriveID driveId, Float* pTow );
```

Beschreibung Der aktuelle Schleppfehler des Antriebsmoduls *driveId* wird über den Zeiger *pTow* übergeben. Der Schleppfehler ist die Differenz zwischen der von der Bewegungssteuerung berechneten Soll-Position und der vom Geber des Moduls ermittelten Ist-Position.

Hinweis: Der zulässige maximale Schleppfehler des Moduls ist mit Hilfe der Funktion `CLDDrive_setMaxTow` programmierbar.

Die Einheit des Schleppfehlers ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Aktueller Schleppfehler wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getPos, CLDDrive_setMaxTow

CLDDrive_getCurrent

Aufgabe Holt die aktuelle Stromaufnahme der Endstufe des Moduls

Prototyp
```
CLDRet CLDDrive_getCurrent( CLDDriveID driveId, Float* pCurrent );
```

Beschreibung Die aktuelle Stromaufnahme der Endstufe des Antriebsmoduls *driveId* wird über den Zeiger *pCurrent* übergeben.

Hinweis: Die zulässige maximale Stromaufnahme der Endstufe ist mit Hilfe der Funktion `CLDDrive_setMaxCurrent` programmierbar.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

`CLD_OK`	Aktuelle Stromaufnahme wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxCurrent

CLDDrive_getTemperature

Aufgabe Holt die aktuelle Innentemperatur des Moduls

Prototyp
```
CLDRet CLDDrive_getTemperature( CLDDriveID driveId, Float* pTemp );
```

Beschreibung Die aktuelle Innentemperatur des Antriebsmoduls *driveId* wird über den Zeiger *pTemp* übergeben.

Hinweis: Die zulässige maximale Innentemperatur des Moduls ist mit Hilfe der Funktion `CLDDrive_setMaxTemperature` programmierbar.

Die Einheit der Temperatur ist Kelvin (**K**).

Rückgabewert

`CLD_OK`	Aktuelle Innentemperatur wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxTemperature

8.7 Abfragen und Verändern von Grenzwerten

Die folgenden Funktionen ermöglichen, Grenzwerte des Antriebsmoduls zu programmieren und deren aktuellen Einstellungen abzufragen. In der Regel bewirkt eine Überschreitung dieser Grenzwerte den Übergang in den Sicherheitszustand des Moduls, das heißt das Modul hält mit der maximalen (Brems-) Beschleunigung an und vermerkt den festgestellten Fehler im Statuswort (siehe `CLDDrive_getStatus`). Weitere Bewegungskommandos werden erst nach einem RESET des Moduls (Funktion `CLDDrive_resetModule`) angenommen, jedoch nur, wenn die verursachende Bedingung nicht mehr vorliegt.

CLDDrive_getMaxDyn

Aufgabe Holt die maximale Dynamik des Moduls

Prototyp
```
CLDRet CLDDrive_getMaxDyn( CLDDriveID driveId, Float* pVel, Float*
pAcc );
```

Beschreibung Die maximale Geschwindigkeit des Antriebsmoduls *driveId* wird über den Zeiger *pVel* übergeben. Die maximale Beschleunigung wird über den Zeiger *pAcc* übergeben.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**, die Einheit der Beschleunigung ist **m/s²** oder **rad/s²**.

Rückgabewert

`CLD_OK`	Maximale Dynamik wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxDyn

CLDDrive_setMaxDyn

Aufgabe Setzt die maximale Dynamik des Moduls

Prototyp
```
CLDRet CLDDrive_setMaxDyn( CLDDriveID driveId, Float vel, Float acc );
```

Beschreibung Die maximale Geschwindigkeit des Antriebsmoduls *driveId* wird in *vel* übergeben. Die maximale Beschleunigung wird in *acc* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung! Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**, die Einheit der Beschleunigung ist **m/s²** oder **rad/s²**.

Rückgabewert

`CLD_OK`	Maximale Dynamik wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getMaxDyn

CLDDrive_getActDyn

Aufgabe Holt die aktuelle Dynamik des Moduls

Prototyp
```
CLDRet CLDDrive_getActDyn( CLDDriveID driveId, Float* pVel, Float*
pAcc );
```

Beschreibung Die aktuelle Geschwindigkeit des Antriebsmoduls *driveId* wird über den Zeiger *pVel* übergeben. Die aktuelle Beschleunigung wird über den Zeiger *pAcc* übergeben. Die aktuelle Dynamik bezeichnet die von der Bewegungssteuerung verwendeten Werte für Rampenbewegungen.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**, die Einheit der Beschleunigung ist **m/s²** oder **rad/s²**.

Rückgabewert

`CLD_OK`	Aktuelle Dynamik wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.

`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setActDyn, CLDDrive_setMaxDyn

CLDDrive_setActDyn

Aufgabe Setzt die aktuelle Dynamik des Moduls

Prototyp
```
CLDRet CLDDrive_setActDyn( CLDDriveID driveId, Float vel, Float acc );
```

Beschreibung Die aktuelle Geschwindigkeit des Antriebsmoduls *driveId* wird in *vel* übergeben. Die aktuelle Beschleunigung wird in *acc* übergeben. Die aktuelle Dynamik bezeichnet die von der Bewegungssteuerung verwendeten Werte für Rampenbewegungen.

Da die Funktion zur Bewegung im Rampenmodus (`CLDDrive_moveRamp`) die aktuelle Dynamik neu setzt, ist der Aufruf dieser Funktion nur während einer bereits laufenden Rampenbewegung sinnvoll. Es lassen sich dadurch für Beschleunigung und Verzögerung innerhalb eines Rampenprofils getrennte Werte verwenden.

Achtung! Die angegebenen Werte dürfen die im Modul eingestellten Maximal-Werte (siehe `CLDDrive_getMaxDyn`) nicht überschreiten. Höhere Werte werden auf die eingestellten Werte begrenzt.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**, die Einheit der Beschleunigung ist **m/s²** oder **rad/s²**.

Rückgabewert

`CLD_OK`	Aktuelle Dynamik wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getMaxDyn, CLDDrive_getActDyn

CLDDrive_getRange

Aufgabe Holt den zulässigen Bewegungsbereich des Moduls

Prototyp
```
CLDRet CLDDrive_getRange( CLDDriveID driveId, Float* pPosLimit, Float*
pNegLimit );
```

Beschreibung Die maximal zulässige Position in positiver Richtung des Antriebsmoduls *driveId* wird über den Zeiger *pPosLimit* übergeben. Die maximal zulässige Position in negativer Richtung wird über den Zeiger *pNegLimit* übergeben.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Zulässiger Bewegungsbereich wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setRange

CLDDrive_setRange

Aufgabe Setzt den zulässigen Bewegungsbereich des Moduls

Prototyp
```
CLDRet CLDDrive_setRange( CLDDriveID driveId, Float posLimit, Float
negLimit );
```

Beschreibung Die maximal zulässige Position in positiver Richtung des Antriebsmoduls *driveId* wird in *posLimit* übergeben. Die maximal zulässige Position in negativer Richtung wird in *negLimit* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung! Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Zulässiger Bewegungsbereich wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getRange

CLDDrive_getMaxTow

Aufgabe Holt den maximal zulässigen Schleppfehler des Moduls

Prototyp
```
CLDRet CLDDrive_getMaxTow( CLDDriveID driveId, Float* pTow );
```

Beschreibung Der maximal zulässige Schleppfehler des Antriebsmoduls *driveId* wird über den Zeiger *pTow* übergeben. Falls der aktuelle Schleppfehler den Wert in *pTow* überschreitet, wird ein Fehler signalisiert.

Die Einheit des Schleppfehlers ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Maximaler Schleppfehler wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxTow

CLDDrive_setMaxTow

Aufgabe Setzt den maximal zulässigen Schleppfehler des Moduls

Prototyp
```
CLDRet CLDDrive_setMaxTow( CLDDriveID driveId, Float tow );
```

Beschreibung Der maximal zulässige Schleppfehler des Antriebsmoduls *driveId* wird in *tow* übergeben. Falls der aktuelle Schleppfehler den angegebenen Wert in *tow* überschreitet, wird ein Fehler signalisiert.

Achtung! Der angegebene Wert darf den im Modul voreingestellten Wert nicht überschreiten. Höhere Werte werden auf den voreingestellten Wert begrenzt.

Die Einheit des Schleppfehlers ist je nach Modultyp **m** oder **rad**.

Rückgabewert

`CLD_OK`	Maximaler Schleppfehler wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getMaxTow

CLDDrive_getMaxCurrent

Aufgabe Holt die maximal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
CLDRet CLDDrive_getMaxCurrent( CLDDriveID driveId, Float* pMaxCurrent
);
```

Beschreibung Die maximal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *driveId* wird über den Zeiger *pMaxCurrent* übergeben. Falls die aktuelle Stromaufnahme den Wert in *pMaxCurrent* überschreitet, wird ein Fehler signalisiert.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

`CLD_OK`	Maximale Stromaufnahme wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxCurrent

CLDDrive_setMaxCurrent

Aufgabe Setzt die maximal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
CLDRet CLDDrive_setMaxCurrent( CLDDriveID driveId, Float maxCurrent );
```

Beschreibung Die maximal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *driveId* wird in *maxCurrent* übergeben. Falls die aktuelle Stromaufnahme den angegebenen Wert in *maxCurrent* überschreitet, wird ein Fehler signalisiert.

Achtung! Der angegebene Wert darf den im Modul voreingestellten Wert nicht überschreiten. Höhere Werte werden auf den voreingestellten Wert begrenzt.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

`CLD_OK`	Maximale Stromaufnahme wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getMaxCurrent

CLDDrive_getMaxTemperature

Aufgabe Holt die maximal zulässige Innentemperatur des Moduls

Prototyp

```
CLDRet CLLDrive_getMaxTemperature( CLLDriveID driveId, Float*
pMaxTemperature );
```

Beschreibung Die maximal zulässige Innentemperatur des Antriebsmoduls *driveId* wird über den Zeiger *pMaxTemperature* übergeben. Falls die aktuelle Innentemperatur den Wert in *pMaxTemperature* überschreitet, wird ein Fehler signalisiert.

Die Einheit der Innentemperatur ist Kelvin (**K**).

Rückgabewert

`CLD_OK`	Maximale Innentemperatur wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setMaxTemperature

CLDDrive_setMaxTemperature

Aufgabe Setzt die maximal zulässige Innentemperatur des Moduls

Prototyp

```
CLDRet CLLDrive_setMaxTemperature( CLLDriveID driveId, Float
maxTemperature );
```

Beschreibung Die maximal zulässige Innentemperatur des Antriebsmoduls *driveId* wird in *maxTemperature* übergeben. Falls die aktuelle Innentemperatur den angegebenen Wert in *maxTemperature* überschreitet, wird ein Fehler signalisiert.

Achtung! Der angegebene Wert darf den im Modul voreingestellten Wert nicht überschreiten. Höhere Werte werden auf den voreingestellten Wert begrenzt.

Die Einheit der Innentemperatur ist Kelvin (**K**).

Rückgabewert

`CLD_OK`	Maximale Innentemperatur wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getMaxTemperature

CLDDrive_getRegCoeff

Aufgabe Holt die aktuell eingestellten Reglerkoeffizienten

Prototyp

```
CLDRet CLLDrive_getRegCoeff( CLLDriveID driveId, Int16* pCoeff0,
Int16* pCoeff1, Int16* pCoeff2 );
```

Beschreibung Die drei Koeffizienten des modulinternen PID-Lagereglers werden abgefordert.

Rückgabewert

`CLD_OK`	Reglerkoeffizienten wurden geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setRegCoeff

CLDDrive_setRegCoeff

Aufgabe Setzt die aktuellen Reglerkoeffizienten des PID-Reglers

Prototyp
```
CLDRet CLDDrive_setRegCoeff( CLDDriveID driveId, Int16 coeff0, Int16
coeff1, Int16 coeff2 );
```

Beschreibung Die drei Koeffizienten des modulinternen PID-Lagereglers werden übergeben. Bei allen folgenden Bewegungskommandos werden diese neuen Parameter verwendet.

Achtung! Es erfolgt keine Überprüfung auf sinnvolle Koeffizienten! Falls das Modul in den Sicherheitszustand übergeht (bei Fehlern oder einem Aufruf der Funktion `CLDDrive_haltModule`), werden die voreingestellten Koeffizienten verwendet.

Die Ermittlung der Koeffizienten erfolgt in der im Abschnitt *Reglerkoeffizienten* auf Seite 23 beschriebenen Weise.

Rückgabewert

`CLD_OK`	Reglerkoeffizienten wurden gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_getRegCoeff

8.8 Abfragen von Vorgabewerten und sonstige Funktionen

Die im vorhergehenden Abschnitt dargestellten Funktionen dienen zum Abfragen und Setzen von Vorgabewerten in den Antriebsmodulen, deren Überschreitung in der Regel einen Fehlerzustand bewirkt. In diesem Fall geht, wie beschrieben, das Modul in den Sicherheitszustand über. Die im folgenden genannte Funktion `CLDDrive_getConfigDrive` dient zum Abfragen der im Modul voreingestellten Maximalwerte, die nicht überschritten werden dürfen.

Wenn in den im vorhergehenden Abschnitt gezeigten Funktionen höhere als die voreingestellten Werte programmiert werden sollen, werden die neuen Werte in jedem Fall auf die voreingestellten begrenzt.

CLDDrive_getConfigDrive

Aufgabe Liefert die im Antriebsmodul voreingestellten Werte in einer Struktur

Prototyp
```
CLDRet CLDDrive_getConfigDrive( CLDDriveID driveId, ConfigDrive*
pDrive );
```

Beschreibung Über den Zeiger *pDrive*, der auf eine Struktur `ConfigDrive` (beschrieben in der Datei `def\defdrive.h`) zeigen muß, werden die im Antriebsmodul voreingestellten Werte zurückgeliefert.

Die Struktur vom Typ `ConfigDrive`, auf die der Zeiger *pDrive* weist, enthält nach dem Funktionsaufruf in ihren Elementen alle wesentlichen voreingestellten Eigenschaften des Antriebsmoduls *driveId*. Die Elemente der Struktur sind:

ConfigDrive-Struktur

	Element	Typ	Einheit	Bedeutung
☐	`posLimit`	`Float`	**m** oder **rad**	positive Softwareendlage
☐	`negLimit`	`Float`	**m** oder **rad**	negative Softwareendlage
☐	`maxVel`	`Float`	**m/s** oder **rad/s**	maximale Geschwindigkeit
☐	`maxAcc`	`Float`	**m/s²** oder **rad/s²**	maximale Beschleunigung
☐	`maxTow`	`Float`	**m** oder **rad**	maximal zulässiger Schleppfehler
☐	`maxCurrent`	`Float`	**A**	maximal zulässige Stromaufnahme
☐	`maxTemperature`	`Float`	**K**	maximal zulässige Innentemperatur
☐	`incrPerUnit`	`Float`	1	interner Umrechnungswert
☐	`defRegCoeff0`	`Int16`	siehe Text	voreingestellter Reglerkoeffizient
☐	`defRegCoeff1`	`Int16`	siehe Text	voreingestellter Reglerkoeffizient
☐	`defRegCoeff2`	`Int16`	siehe Text	voreingestellter Reglerkoeffizient
☐	`romVersion`	`UInt16`	siehe Text	ROM-Version
☐	`serNum`	`UInt16`	siehe Text	Seriennummer des Moduls
☐	`type`	`UInt8`	siehe Text	Modultyp
☐	`pProdNum[8]`	`UInt8`	siehe Text	Zeichenkette zur Modulbeschreibung

Die mit *defRegCoeff0..2* bezeichneten Strukturelemente enthalten die Koeffizienten des in den Antriebsmodulen integrierten Lagereglers. Die Berechnung der Koeffizienten ist im Abschnitt *Reglerkoeffizienten* auf Seite 23 beschrieben.

Im Element *romVersion* ist die Versionsnummer des Modul-Betriebssystems enthalten, wobei gilt:

`0x2037` == Version 2.03.7

Das Element `serNum` enthält die Seriennummer des Moduls, die für Identifikationszwecke verwendet werden kann. Das Element `pProdNum` enthält eine Kurzbezeichnung des Modultyps, beispielsweise die Zeichenkette `15.311`.

Das Element *type* bezeichnet den Typ des Antriebsmoduls, für den in der Datei `def\defdrive.h` folgende Konstanten definiert sind:

- ☐ `ROTARY_DRIVE` Drehmodul, Einheiten gelten in **rad**, **rad/s** bzw. **rad/s²**
- ☐ `LINEAR_DRIVE` Linearmodul, Einheiten gelten in **m**, **m/s** bzw. **m/s²**

Rückgabewert

`CLD_OK`	Struktur `ConfigDrive` wurde geholt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern*.

Siehe auch CLDDrive_setMaxTemperature

CLDDrive_changeBaudRate

Aufgabe Verändert die Übertragungsrate der Kommunikation mit den Antriebsmodulen

Prototyp

```
CLDRet CLDDrive_changeBaudRate( Word baudRate );
```

Beschreibung Zur Kommunikation mit den Antriebsmodulen wird bei allen folgenden Datentelegrammen die vorgegebene Übertragungsrate *baudRate* verwendet. Dazu wird allen angeschlossenen Modulen die neue Übertragungsrate mitgeteilt und eine Wartezeit eingefügt, um nachfolgende Funktionsaufrufe schon sicher mit der neuen Übertragungsrate auszuführen.

Bei Aufruf der Funktion `CLDDrive_exitManip` werden die Module wieder auf die Standardrate zurückgestellt.

Achtung! `CLDDrive_changeBaudRate` ist von der Implementierung abhängig.

Der Aufruf dieser Funktion ist stark von der jeweiligen Implementierung und der gewählten Kommunikationsverbindung abhängig! Bestimmte Feldbusprotokolle lassen eine Änderung der Übertragungsrate nicht zu, ebenso können an dieser Stelle keine möglichen Werte angegeben werden. Die Antriebsmodule mit RS-485-Kommunikationsverbindung sind nach dem Einschalten in der Regel auf 9600 Baud eingestellt.

Beachten Sie hierzu bitte die Beschreibungen im Anhang.

Rückgabewert

`CLD_OK`	Reglerkoeffizienten wurden gesetzt.
`CLDERR_BADPARAM`	Falsch angegebene Übertragungsrate.
`CLDERR_BADDEVICEVERSION`	Mindestens ein Antriebsmodul des Manipulators ist nicht in der Lage, die geforderte Änderung der Übertragungsrate zu vollziehen (Modulversion mindestens 2.03.2).
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch zusätzliche Dokumentation

8.9 Digital-I/O-Funktionen

Im folgenden werden die Funktionen zur Verwaltung der Digital-Einheit der Antriebsmodule beschrieben.

CLDDrive_configDigitalIO

Aufgabe Konfiguriert die Digital-Ein-/-Ausgabe

Prototyp
```
CLDRet CLDDrive_configDigitalIO( CLDDriveID driveId, UInt32 mask );
```

Beschreibung Die Digital-Ausgabe-Einheit des Antriebsmoduls *driveId* wird entsprechend dem Parameter *mask* konfiguriert. Es muß sichergestellt sein, daß das angegebene Modul mit einer Digital-I/O-Einheit bestückt ist. Diese Funktion muß gerufen werden, bevor die Digital-Funktionen verwendet werden können.

Der Parameter *mask* legt fest, welche Anschlüsse als Eingang bzw. Ausgang verwendet werden sollen. Dabei entspricht das Bit an Position 0 dem Digital-Anschluß 0 usw. (Bit = 0 bedeutet Eingang, Bit = 1 bedeutet Ausgang).

Zur Zeit muß für *mask* der Wert 0xF0 angegeben werden. Demzufolge sind die Anschlüsse 0..3 als Eingang, die Anschlüsse 4..7 als Ausgang geschaltet.

Beispiel In folgendem Beispiel soll die Digital-Ein-/-Ausgabe konfiguriert werden, um danach die Ausgabe zu setzen und den Zustand der Eingänge anzufragen.

Konfiguration der Digital-Einheit:

```
UInt32 mask = 0xF0;                    // 0..3 = IN, 4..7 = OUT
```

```
UInt32 output = 0x50;                 // 4=HI, 5=LO, 6=HI, 7=LO
UInt32 input;
retVal = CLDDrive_configDigitalIO( id, mask );
```

Setzen der Ausgänge:

```
retVal = CLDDrive_setDigitalOutput( id, output );
```

Abfragen der Eingänge:

```
retVal = CLDDrive_getDigitalInput( id, &input );
input &= ~mask;
```

Rückgabewert

`CLD_OK`	Digital-I/O-Einheit wurde konfiguriert.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setDigitalOutput, CLDDrive_getDigitalInput

CLDDrive_setDigitalOutput

Aufgabe Setzt die Digital-Ausgänge

Prototyp

```
CLDRet CLDDrive_setDigitalOutput( CLDDriveID driveId, UInt32 out );
```

Beschreibung Nach der Konfiguration des Anschlusses werden mit Hilfe dieser Funktion die Digital-Ausgänge gesetzt. Der im Parameter *out* angegebene Wert wird intern mit der bei der Konfiguration angegebenen Maske UND-verknüpft und an die Ausgänge geschaltet.

Alle anderen als die Bits 4..7 werden momentan ignoriert und sollten auf den Wert 0 gesetzt sein.

Beispiel siehe CLDDrive_configDigitalIO

Rückgabewert

`CLD_OK`	Digital-Ausgänge wurde gesetzt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_configDigitalIO, CLDDrive_getDigitalInput

CLDDrive_getDigitalInput

Aufgabe Liefert die Digital-Eingänge

Prototyp

```
CLDRet CLDDrive_getDigitalInput( CLDDriveID driveId, UInt32* pIn );
```

Beschreibung Nach Aufruf dieser Funktion befindet sich in der 32-Bit-Variablen, auf die 'pIn' zeigt, der Zustand der Digital-Eingänge. Zusätzlich wird der aktuelle Zustand der Digital-Ausgänge in dieser Variablen abgelegt. Deshalb muß der ermittelte Wert mit dem Komplement der bei der Konfiguration angegebenen Ein-/-Ausgangs-Maske UND-verknüpft werden, um ausschließlich den Zustand der Eingänge zu erhalten.

Momentan stellen die Bits 0..3 den Zustand der Eingänge dar, die Bits 4..7 spiegeln den Zustand der Ausgänge wieder, alle anderen Bits sind auf 0 gesetzt.

Beispiel siehe CLDDrive_configDigitalIO

Rückgabewert

`CLD_OK`	Digital-Eingänge wurden abgefragt.
`CLDERR_BADPARAM`	Antriebsmodul *driveId* nicht vorhanden.
`CLDERR_DRIVE_BADANSWER`	Antriebsmodul hat nicht korrekt reagiert.
`CLDERR_DRIVE_RECEIVE`	Fehler bei der Kommunikation mit dem Antriebsmodul.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch CLDDrive_setDigitalOutput, CLDDrive_configDigitalIO

8.10 Roboter-Funktionen

Die folgenden Funktionen ermöglichen bei Vorhandensein der CLDRobot-Schnittstelle die Definition einer Roboterstruktur und die Auslösung von Funktionen zur Bewegung des Roboters.

CLDRobot_enterRobMode

Aufgabe Initialisierung des Robotermodus

Prototyp
```
CLDRet CLDRobot_enterRobMode( Short* pNumDrives );
```

Beschreibung Die Initialisierung des Robotermodus ist zu Beginn der Arbeit mit CLDRobot-Funktionen notwendig. Die Datenbereiche für maximal 32 Roboter werden angelegt. Bei Ausführung der Funktion liefert sie die Anzahl aller am Bus angeschlossenen Module.

Achtung! Diese Funktion muß vor der ersten Verwendung einer anderen roboterspezifischen Funktion ausgeführt werden.

Rückgabewert

`CLD_OK`	Robotermodus wurde vorbereitet.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_exitRobMode

Aufgabe Verlassen des Robotermodus

Prototyp
```
CLDRet CLDRobot_exitRobMode( void );
```

Beschreibung Zum Ende der Arbeit mit der CLDRobot-Schnittstelle werden die durch Aufruf der Funktion die angelegten Datenbereiche freigegeben.

Rückgabewert

`CLD_OK`	Robotermodus wurde freigegeben.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_initRob

Aufgabe Initialisierung eines Roboters

Prototyp
```
CLDRet CLDRobot_initRob( CLDRobotID* pRobotId, const Char*
pInitString, RobotType robType, const char* pFileName );
```

Beschreibung Diese Funktion bereitet einen Roboter vor. Dazu wird ein Initialisierungsstring übergeben. Der vereinbarte Roboter kann bei allen weiteren Funktionen mittels eines eindeutigen Bezeichners angesprochen werden, der über den Zeiger *pRobotId* zurückgeliefert wird. Entsprechend der mit *robType* übergebenen Auswahl, wird ein Roboter mit der voreingestellten Konfiguration definiert.

Achtung! Diese Funktion muß vor der ersten Verwendung einer anderen roboterspezifischen Funktion ausgeführt werden.

Rückgabewert

`CLD_OK`	Roboter wurde vorbereitet.
`CLDERR_BADPARAM`	Fehlerhafte Initialisierung.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

Siehe auch Abschnitt *Initialisierung* (Seite 28)

CLDRobot_exitRob

Aufgabe Beenden des Robotermodus

Prototyp
```
CLDRet CLDRobot_exitRob( CLDRobotID robotId );
```

Beschreibung Diese Funktion beendet den Robotermodus. Dazu wird der Bezeichner *robotId* übergeben. Nach Aufruf der Funktion ist dieser Bezeichner nicht mehr definiert.

Rückgabewert

`CLD_OK`	Robotermodus wurde beendet.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_homeRob

Aufgabe Referenzpunktfahrt eines Roboters

Prototyp
```
CLDRet CLDRobot_homeRob(CLDRobotID robotId, const Char* pInitString);
```

Beschreibung Diese Funktion initiiert eine Referenzpunktfahrt des durch sein *robotId* bezeichneten Roboters. Mit dem übergebenen *Initstring* wird die Reihenfolge der Referenzpunktfahrt der einzelnen zum Roboter gehörenden Antriebe bestimmt. Die Reihenfolge bezieht sich stets auf einen Roboter und beginnt mit 0. Die Funktion erzwingt nur dann die Referenzpunkt eines Antriebs, wenn es durch sein Statuswort darauf hinweist, daß es nicht synchronisiert ist.

Rückgabewert

`CLD_OK`	Referenzpunktfahrt erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_stopRob

Aufgabe Sofortiges Anhalten der Roboterbewegung

Prototyp
```
CLDRet CLDRobot_stopRob( CLDRobotID robotId );
```

Beschreibung Diese Funktion löst einen sofortigen Stop der Bewegung des Roboters aus. Dazu wird der Bezeichner *robotId* übergeben.

Rückgabewert

`CLD_OK`	Roboter steht.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_haltRob

Aufgabe Not-Stop des Roboters

Prototyp
```
CLDRet CLDRobot_haltRob( CLDRobotID robotId );
```

Beschreibung Diese Funktion löst einen sofortigen Halt des Roboters aus. Dazu wird der Bezeichner *robotId* übergeben. Nach Aufruf der Funktion sind die Antriebe im Halt-Zustand und müssen vor Ausführung weiterer Befehle explizit rückgesetzt werden.

Rückgabewert

`CLD_OK`	Roboter steht.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_goTo

Aufgabe Zeitsynchrones Verfahren in Achskoordinaten

Prototyp
```
CLDRet CLDRobot_goTo( CLDRobotID robotId, RobotCoord* pRobotCoord,
Float* time );
```

Beschreibung DieseFunktion dient zum zeitsynchronen Verfahren der Roboterachsen. Der Roboter wird durch *robotId* bestimmt. Die Zielposition wird in Achskoordinaten angegeben. Die Zeitvorgabe erfolgt mit *time.*

Rückgabewert

`CLD_OK`	Bewegung erfogreich ausgeführt.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_movePTP

Aufgabe Zeitsynchrones Verfahren in Weltkoordinaten

Prototyp
```
CLDRet CLDRobot_movePTP( CLDRobotID robotId, WorldCoord* pWorldCoord,
Float* time );
```

Beschreibung DieseFunktion dient zum zeitsynchronen Verfahren der Roboterachsen. Der Roboter wird durch *robotId* bestimmt. Die Zielposition wird in Weltkoordinaten angegeben. Die Zeitvorgabe erfolgt mit *time.*

Rückgabewert

`CLD_OK`	Bewegung erfolgreich ausgeführt.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_moveLIN

Aufgabe Räumliche Bewegung entlang einer Geraden

Prototyp
```
CLDRet CLDRobot_moveLIN( CLDRobotID robotId, WorldCoord* pWorldCoord,
Float* time );
```

Beschreibung Diese Funktion ermöglicht eine bahngesteuerte Bewegung des durch *robotId* bezeichneten Roboters. Es wird eine Bahn in Form einer Geraden vom Ausgangspunkt zum in Weltkoordinaten angegebenen Zielpunkt in *time* Sekunden verfahren.

Rückgabewert

`CLD_OK`	Bewegung erfolgreich ausgeführt.

`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_moveCIRC

Aufgabe Räumliche Bewegung entlang eines Kreisbogens

Prototyp
```
CLDRet CLDRobot_moveCIRC( CLDRobotID robotId, WorldCoord* pWCoord1,
WorldCoord* pWCoord2, Float* time );
```

Beschreibung Der mit *robotId* bezeichnete Roboter verfährt einen Kreisbogen vom Ausgangspunkt über einen Zwischenpunkt (*pWCoord1*) zum Zielpunkt (*pWCoord2*) in *time* Sekunden. Die Beschreibung erfolgt in Weltkoordinaten.

Rückgabewert

`CLD_OK`	Bewegung erfolgreich beendet.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setBaseFrame

Aufgabe Anpassung des Basiskoordinatensystems

Prototyp
```
CLDRet CLDRobot_setBaseFrame( CLDRobotID robotId, Frame* pFrame );
```

Beschreibung Für den durch *robotId* bezeichneten Roboter gilt nach Ausführung der Funktion das durch *pFrame* beschriebene Basiskoordinatensystem.

Rückgabewert

`CLD_OK`	Das neue Basiskoordinatensystem wurde übernommen.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getBaseFrame

Aufgabe Abfrage des eingestellten Basiskoordinatensystems

Prototyp
```
CLDRet CLDRobot_getBaseFrame( CLDRobotID robotId, Frame* pFrame );
```

Beschreibung Diese Funktion fragt das eingestellte Basiskoordinatensystem ab. Dazu wird der Bezeichner *robotId* übergeben. Die Daten werden in der Struktur pFrame übergeben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setToolFrame

Aufgabe Anpassung des Toolkoordinatensystems

Prototyp
```
CLDRet CLDRobot_setToolFrame( CLDRobotID robotId, Frame* pFrame );
```

Beschreibung Diese Funktion setzt ein benutzerdefiniertes Toolkoordinatensystem. Dazu wird der Bezeichner *robotId* übergeben. Die durch die Struktur *pFrame* beschriebene neue Lage wird vom System übernommen und gilt künftig für die Ausführung von Bahnbefehlen.

Rückgabewert

`CLD_OK`	Neues Toolkoordinatensystem wurde übernommen.

CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getToolFrame

Aufgabe Abfrage des eingestellten Toolkoordinatensystems

Prototyp
```
CLDRet CLDRobot_getToolFrame( CLDRobotID robotId, Frame* pFrame );
```

Beschreibung Diese Funktion fragt das eingestellte Toolkoordinatensystem ab. Dazu werden der Bezeichner *robotId* sowie eine Struktur *pFrame* übergeben. Nach Aufruf der Funktion liegen die aktuellen Daten in *pFrame* vor.

Rückgabewert

CLD_OK	Funktionsaufruf erfolgreich.
CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setMaxDynRobot

Aufgabe Grenzwerte für Roboterdynamik setzen

Prototyp
```
CLDRet CLDRobot_setMaxDynRobot( CLDRobotID robotId, Float* pVeloc,
Float* pAccel );
```

Beschreibung Diese Funktion begrenzt die Bahngeschwindigkeiten und Beschleunigungen des bezeichneten Roboters.

Rückgabewert

CLD_OK	Roboterdynamik erfolgreich begrenzt.
CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getMaxDynRobot

Aufgabe Grenzwerte für Roboterdynamik abfragen

Prototyp
```
CLDRet CLDRobot_getMaxDynRobot( CLDRobotID robotId, Float* pVeloc,
Float* pAccel );
```

Beschreibung Diese Funktion fragt die aktuellen Einstellungen für die Bahndynamik des Roboters ab. Dazu wird der Bezeichner *robotId* übergeben. Die Werte liegen nach Ausführung der Funktion in *pVeloc* und *pAccel* vor.

Rückgabewert

CLD_OK	Funktionsaufruf erfolgreich.
CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setDriveRanges

Aufgabe Grenzwerte für Bewegungsbereiche der Achsen setzen

Prototyp
```
CLDRet CLDRobot_setDriveRanges( CLDRobotID robotId, RobotCoord*
pPosMin, RobotCoord* pPosMax);
```

Beschreibung Diese Funktion setzt neue Bewegungsbereiche für die Roboterachsen. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden in den Strukturen *pPosMin* und *pPosMax* übergeben.

Rückgabewert

CLD_OK	Neue Grenzwerte gesetzt.
CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getDriveRanges

Aufgabe Grenzwerte für Bewegungsbereiche der Achsen abfragen

Prototyp
```
CLDRet CLDRobot_getDriveRanges( CLDRobotID robotId, RobotCoord*
pPosMin, RobotCoord* pPosMax);
```

Beschreibung Diese Funktion fragt die Bewegungsbereiche für die Roboterachsen ab. Dazu wird der Bezeichner *robotId* übergeben. Die eingestellten Grenzwerte werden in den Strukturen *pPosMin* und *pPosMax* übergeben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setMaxDynDrives

Aufgabe Grenzwerte für Bewegungsdynamik der Achsen setzen

Prototyp
```
CLDRet CLDRobot_setMaxDynDrives( CLDRobotID robotId, RobotCoord*
pRobotVeloc, RobotCoord* pRobotAccel );
```

Beschreibung Diese Funktion setzt neue Dynamikparameter für die Roboterachsen. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden in den Strukturen *pRobotVeloc* und *pRobotAccel* beschrieben.

Rückgabewert

`CLD_OK`	Neue Grenzwerte wurden gesetzt.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getMaxDynDrives

Aufgabe Grenzwerte für Bewegungsdynamik der Achsen abfragen

Prototyp
```
CLDRet CLDRobot_getMaxDynDrives( CLDRobotID robotId, RobotCoord*
pRobotVeloc, RobotCoord* pRobotAccel );
```

Beschreibung Diese Funktion fragt die eingestellten Bewegungsbereiche der Roboterachsen ab. Dazu wird der Bezeichner *robotId* übergeben. Die Grenzwerte werden in den Strukturen *pRobotVeloc* und *pRobotAccel* übergeben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfogreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setMaxTow

Aufgabe Grenzwerte für maximale Schleppfehler der Achsen setzen

Prototyp
```
CLDRet CLDRobot_setMaxTow( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion setzt neue Maximalwerte für die Schleppfehler der Roboterachsen. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden in der Struktur *pR* übergeben.

Rückgabewert

`CLD_OK`	Neue Grenzwerte wurden gesetzt.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getMaxTow

Aufgabe Grenzwerte für maximale Schleppfehler der Achsen abfragen

Prototyp
```
CLDRet CLDRobot_getMaxTow( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion erfragt die eingestellten Maximalwerte für die Schleppfehler der Roboterachsen. Dazu wird der Bezeichner *robotId* übergeben. Die eingestellten Grenzwerte werden in der Struktur *pR* übergeben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setMaxCurrent

Aufgabe Grenzwerte für maximale Stromaufnahme der Achsen setzen

Prototyp
```
CLDRet CLDRobot_setMaxCurrent( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion setzt neue Grenzwerte für die maximale Stromaufnahme der Antriebe. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden durch die Struktur *pR* beschrieben.

Rückgabewert

`CLD_OK`	Neue Grenzwerte wurden übernommen.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getMaxCurrent

Aufgabe Grenzwerte für maximale Stromaufnahme der Achsen abfragen

Prototyp
```
CLDRet CLDRobot_getMaxCurrent( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion dient der Abfrage der Grenzwerte für die maximale Stromaufnahme der Antriebe. Dazu wird der Bezeichner *robotId* übergeben. Die eingestellten Grenzwerte werden in der Struktur *pR* beschrieben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setMaxTemperature

Aufgabe Grenzwerte für maximale Motortemperatur der Achsen setzen

Prototyp
```
CLDRet CLDRobot_setMaxTemperature(CLDRobotID robotID, RobotCoord* pR);
```

Beschreibung Diese Funktion setzt neue Grenzwerte für die maximale Motortemperatur der Antriebe. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden durch die Struktur *pR* beschrieben.

Rückgabewert

`CLD_OK`	Neue Grenzwerte wurden übernommen.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getMaxTemperature

Aufgabe Grenzwerte für maximale Motortemperatur der Achsen abfragen

Prototyp `CLDRet CLDRobot_getMaxTemperature(CLDRobotID robotID, RobotCoord* pR);`

Beschreibung Diese Funktion erfragt die eingestellten Grenzwerte für die maximale Motortemperatur der Antriebe. Dazu wird der Bezeichner *robotId* übergeben. Die neuen Grenzwerte werden durch die Struktur *pR* beschrieben.

Rückgabewert

`CLD_OK`	Funktionsaufruf erfolgreich.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_setLimCurrent

Aufgabe Strombegrenzung für die Achsregelung zuschalten

Prototyp `CLDRet CLDRobot_setLimCurrent( CLDRobotID robotID, RobotCoord* pR );`

Beschreibung Diese Funktion schaltet die Strombegrenzung für die Achsregelung mit einem vorgebbaren Grenzwert zu. Dazu werden der Bezeichner *robotId* sowie eine Struktur vom Typ RobotCoord mit den Anwenderdaten übergeben.

Rückgabewert

`CLD_OK`	Daten übernommen.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getRobCoord

Aufgabe Abfrage der aktuellen Position in Achskoordinaten

Prototyp
```
CLDRet CLDRobot_getRobCoord( CLDRobotID robotId, RobotCoord*
pRobotCoord );
```

Beschreibung Diese Funktion dient zur Überwachung der Roboterposition in Achskoordinaten. Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ RobotCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getToolCoord

Aufgabe Abfrage der aktuellen Position in Weltkoordinaten

Prototyp
```
CLDRet CLDRobot_getToolCoord( CLDRobotID robotId, WorldCoord*
pWorldCoord );
```

Beschreibung Diese Funktion dient zur Überwachung der Roboterposition in Weltkoordinaten. Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ WorldCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getRobVel

Aufgabe Überwachung der Robotergeschwindigkeit

Prototyp
```
CLDRet CLDRobot_getRobVel( CLDRobotID robotId, RobotCoord*
pRobotVeloc);
```

Beschreibung Diese Funktion dient zur Überwachung der Achsgeschwindigkeiten des Roboters. Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ RobotCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getActDynRobot

Aufgabe Überwachung der aktuellen Roboterdynamik

Prototyp
```
CLDRet CLDRobot_getActDynRobot( CLDRobotID robotId, RobotCoord*
pRobotVeloc, RobotCoord* pRobotAccel );
```

Beschreibung Diese Funktion dient zur Überwachung der Achsdynamik des Roboters Dazu werden das entsprechende *robotId* sowie Referenzen auf Strukturen vom Typ RobotCoord sowohl für die Achsgeschwindigkeiten als auch die Achsbeschleunigungen übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getTow

Aufgabe Überwachung des Schleppfehlers pro Achse

Prototyp
```
CLDRet CLDRobot_getTow( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion dient zur Überwachung der Schleppfehler der Roboterachsen. Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ RobotCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.
`CLDERR_BADPARAM`	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getCurrent

Aufgabe Überwachung der Stromaufnahme pro Achse

Prototyp
```
CLDRet CLDRobot_getCurrent( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion dient zur Überwachung der Stromaufnahme der Roboterachsen. Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ RobotCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

`CLD_OK`	Daten wurden aktualisiert.

CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDRobot_getTemperature

Aufgabe Überwachung der Achstemperaturen

Prototyp
```
CLDRet CLDRobot_getTemperature( CLDRobotID robotID, RobotCoord* pR );
```

Beschreibung Diese Funktion dient zur Überwachung der Achstemperaturen des Roboters Dazu werden das entsprechende *robotId* sowie die Referenz auf eine Struktur vom Typ RobotCoord übergeben. Die Elemente der Struktur werden durch Aufruf der Funktion aktualisiert.

Rückgabewert

CLD_OK	Daten wurden aktualisiert.
CLDERR_BADPARAM	Roboter *robotId* nicht vorhanden.

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

8.11 ForceTorqueSensor-Funktionen

Die folgenden Funktionen ermöglichen bei Vorhandensein eines Kraft-Momenten-Sensors die Einbindung der gemessenen Daten in die Robotersteuerung.

CLDFTS_initFTSensor

Aufgabe Initialisierung des Sensors

Prototyp
```
HFTS CLDFTS_initFTSensor( const Char* pInitString, const Char*
pCalFile );
```

Beschreibung Diese Funktion initialisiert den angeschlossenen Kraft-Momenten-Sensor.

Rückgabewert

CLD_OK

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_exitFTSensor

Aufgabe Deinitialiserung des Sensors

Prototyp
```
CLDRet CLDFTS_exitFTSensor( HFTS hFTS );
```

Beschreibung Diese Funktion gibt die durch die Anmeldung des Kraft-Momenten-Sensors belegten Ressorucen frei.

Rückgabewert

CLD_OK

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_getFTThreshold

Aufgabe Meßschwelle lesen

Prototyp
```
CLDRet CLDFTS_getFTThreshold( HFTS hFTS, FTVector vector );
```

Beschreibung Mit dieser Funktion werden die aktuell eingestellten Schwellwerte für die Erfassung der Sensordaten abgefragt.

Rückgabewert

CLD_OK

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_setFTThreshold

Aufgabe Meßschwelle setzen

Prototyp `CLDRet CLDFTS_setFTThreshold( HFTS hFTS, FTVector vector );`

Beschreibung Diese Funktion legt die Schwellwerte für die Messung der Kräfte und Momente fest.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_getCalMatrix

Aufgabe Kalibriertabelle erfragen

Prototyp `CLDRet CLDFTS_getCalMatrix( HFTS hFTS, FTMatrix matrix );`

Beschreibung Diese Funktion liefert die eingestellten Daten der Kalibriertabelle.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_setCalMatrix

Aufgabe Kalibriertabelle anpassen

Prototyp `CLDRet CLDFTS_setCalMatrix( HFTS hFTS, FTMatrix matrix );`

Beschreibung Mit dieser Funktion können die Daten der Kalibriertabelle aktualisiert werden.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_getCalOffset

Aufgabe Offset-Grundeinstellung erfragen

Prototyp `CLDRet CLDFTS_getCalOffset( HFTS hFTS, FTVector offset );`

Beschreibung Diese Funktion gibt Auskunft über die eingestellten Offset-Daten des Sensors.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_setCalOffset

Aufgabe Offset-Grundeinstellung anpassen

Prototyp `CLDRet CLDFTS_setCalOffset( HFTS hFTS, FTVector vector );`

Beschreibung Mit dieser Funktion können die Offset-Daten des aktiven Sensors angepaßt werden.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_getFTVector

Aufgabe Aktuelle Sensordaten erfassen

Prototyp `CLDRet CLDFTS_getFTVector( HFTS hFTS, FTVector vector );`

Beschreibung Mit dieser Funktion werden die aktuellen Meßdaten eingelesen.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDFTS_getCurOffset

Aufgabe Nullpunktabgleich (Tara)

Prototyp `CLDRet CLDFTS_getCurOffset( HFTS hFTS );`

Beschreibung Diese Funktion nimmt eine Messung der aktuellen Meßdaten vor und hinterlegt sie intern als Offset. Bis zum nächsten Aufruf der Funktion werden diese Daten bei jeder Messung mit `CLDFTS_getFTVector` vom Meßergebnis subtrahiert.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

8.12 ImpedanceControl-Funktionen

Mit den Funktionen der ImpedanceControl-Schnittstelle sind Bewegungen des Roboters unter Einbeziehung der Sensordaten zur Korrektur der vorgegebenen Trajektorie möglich.

CLDImp_setIConUpdate

Aufgabe Setzen der Zykluszeit für die Interpolation

Prototyp
```
CLDRet CLDImp_setIConUpdate( CLDRobotID robotID, unsigned int update
);
```

Beschreibung Diese Funktion setzt die Interpolationszeit für die impedanzgeregelten Bewegungen des Roboters. Gültige Werte liegen zwischen 10 ms und 100 ms.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_getForceConParams

Aufgabe Abfrage der eingestellten Reglerdaten

Prototyp
```
CLDRet CLDImp_getForceConParams( CLDRobotID robotId, FTVector theKP,
  FTVector theKI, FTVector theKD, FTVector theMinDOut, FTVector
theMaxDOut,
  FTVector theMinIntLimit, FTVector theMaxIntLimit, int theDiffUpdate
);
```

Beschreibung Zur Abfrage der eingestellten Reglerdaten wird diese Funktion aufgerufen.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_setForceConParams

Aufgabe Reglerdaten anpassen

Prototyp
```
CLDRet CLDImp_setForceConParams(   CLDRobotID robotId, FTVector newKP,
```

```
  FTVector newKI, FTVector newKD, FTVector newMinDOut, FTVector
newMaxDOut,
  FTVector newMinIntLimit, FTVector newMaxIntLimit, int newDiffUpdate
);
```

Beschreibung Mit dieser Funktion können anwenderdefinierte Reglerparameter gesetzt werden.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_selectParamSet

Aufgabe Auswahl voreingestellter Reglerdaten

Prototyp
```
CLDRet CLDImp_selectParamSet;
```

Beschreibung Mit dieser Funktion können voreingestellte Reglerdaten gesetzt werden.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_iConMoveTo

Aufgabe Impedanzgeregelte Bewegung mit absoluter Zielpunktvorgabe

Prototyp
```
CLDRet CLDImp_iConMoveTo;
```

Beschreibung Der Roboter erhält vom Anwender eine Zielpunktvorgabe in Weltkoordinaten. Die Bewegung erfolgt zielgerichtet auf die Vorgabe mit Nachgiebigkeit in den ausgewählten Freiheitsgraden. Die Bewegung endet im Normalfall mit dem Erreichen der Zielvorgabe in den nicht selektierten Freiheitsgraden.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_iConMoveRel

Aufgabe Impedanzgeregelte Bewegung mit relativer Zielpunktvorgabe

Prototyp
```
CLDRet CLDImp_iConMoveRel;
```

Beschreibung Der Roboter erhält vom Anwender eine relative Zielpunktvorgabe. Die absolute Zielposition wird vom System automatisch ermittelt. Intern wird der Befehl `CLDImp_iConMoveTo` ausgeführt.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_approach

Aufgabe Impedanzgeregelte Annäherungsbewegung

Prototyp
```
CLDRet CLDImp_approach;
```

Beschreibung Der Roboter erhält als Zielvorgabe eine auszuübende Sollkraft in den nicht selektierten Freiheitsgraden. Die Bewegung endet wenn der Regler den Istwert auf den Sollwert eingeregelt hat.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_followForce

Aufgabe Kraftausweichende Roboterbewegung

Prototyp
```
CLDRet CLDImp_followForce;
```

Beschreibung Der Roboter ist in allen Freiheitsgraden nachgiebig, solange die Funktion zyklisch aufgerufen wird.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

CLDImp_getTransfSensorData

Aufgabe Erfassung transformierter Sensordaten

Prototyp
```
CLDRet CLDImp_getTransfSensorData;
```

Beschreibung Es werden die transformierten Meßwerte der auf den TCP einwirkenden Kräfte und Momente ermittelt.

Rückgabewert `CLD_OK`

Bei anderen Rückgabewerten siehe *Vorgehensweise bei Fehlern* .

9 Index

#

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

Ü

V

W

Z