Robotik

AdaIR
Handbuch

Prof. Dr.-Ing. Günter Hommel
Fachgebiet PDV und Robotik
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

Inhaltsverzeichnis

Seite

1 Einleitung

AdaIR (Ada für Industrieroboter) ist ein Programmiersystem, das eine einheitliche Benutzerschnittstelle zur Steuerung unterschiedlicher Industrieroboter zur Verfügung stellt. Ausgangspunkt des AdaIR-Systems ist die REMROC-Schnittstelle [Hom 87], [Krü 88]. AdaIR stellt dem Benutzer eine Bibliothek von Diensten zur Verfügung. Diese Dienste kommunizieren mit der Steuereinheit des Roboters und können von einem in Ada geschriebenen Benutzerprogramm aufgerufen werden. Die wesentlichen Funktionen des AdaIR-Systems sind:

- Bewegung des Roboters im Punkt-zu-Punkt Modus und im Teach-Modus,
- Steuerung des Effektors,
- Ermittlung der aktuellen Stellung des Roboters,
- Behandlung von Fehlern und Unterbrechungen,
- Ansteuerung von peripheren Geräten und Sensorik.

Die Stellung des Effektors kann sowohl durch homogene 4x4-Matrizen als auch durch Tupel von Gelenkvariablen beschrieben werden.

Bei der Spezifikation des Systems wurde von folgenden Anforderungen ausgegangen:

- Einfachheit der funktionellen Beschreibung sowie leichte Installierbarkeit;
- Portabilität sowohl im Hinblick auf verschiedene Rechner als auch im Hinblick auf verschiedene Roboter;
- Flexibilität und universeller Funktionsumfang.

Zur Vereinfachung der Verwendung verschiedener Roboter erwies es sich als sinnvoll, alle kinematischen Berechnungen in einem eigenen Modul zusammenzufassen. Dieses Modul ist eine Bibliothek von Ada-Prozeduren und ist bei Verwendung von Robotern mit unterschiedlicher Kinematik auszutauschen.

2 Grundstruktur des Systems

AdaIR wurde mit Blick auf höchstmögliche Portabilität bezüglich der verwendeten Rechner und Robotertypen konzipiert. Bei Wechsel des Rechners oder des Roboters sowie bei Austausch oder Hinzufügen peripherer Komponenten soll seitens der Software nur der Austausch der jeweils betroffenen Module notwendig werden. Das führt zu dem in Bild 2.1 gezeigten grundsätzlichen Aufbau von AdaIR.

Der Roboter und die ihm direkt zugeordneten peripheren Einheiten (Sensoren und Aktoren im Effektor, binäre Eingangstore der Robotersteuerung) werden ausschließlich vom Systemprozeß ROBOTER verwaltet. Dieser transformiert die Meldungen vom Roboter in ein, für den weiter oben liegenden Prozeß CONTROL geeignetes, Datenformat und nimmt von diesem Anforderungen zur Steuerung des Roboters entgegen. Die Leistungen und Dienste des Roboters und seiner direkt zugeordneten Peripherie werden in Form von Rendezvous-Eintrittspunkten zur Verfügung gestellt.

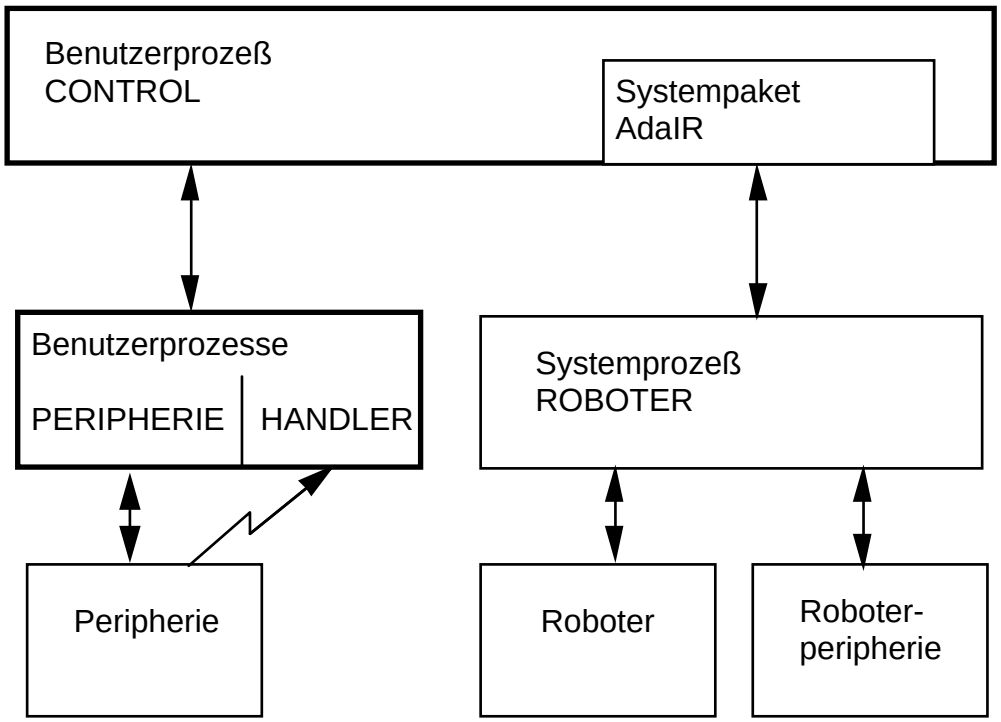


Bild 2.1: Struktur des AdaIR-Systems

Für jede zusätzliche periphere Einheit, die einbezogen werden soll, ist vom Benutzer ein eigener Prozeß bereitzustellen. Ein Prozeß dieser Art ist der ebenfalls in Bild 2.1 dargestellte Prozeß PERIPHERIE. Die Leistungen und Dienste der peripheren Einheit werden in Form von Rendezvous-Eintrittspunkten zur Verfügung gestellt. Soll die periphere Einheit asynchrone Unterbrechungen auslösen können, ist ein weiterer Prozeß HANDLER zur Behandlung bereitzustellen. Für die Synchronisation und Kommunikation mit dem Benutzerprogramm werden Rendezvous-Eintrittspunkte zur Verfügung gestellt.

Das eigentliche Benutzerprogramm zur Steuerung des Roboters bildet den Prozeß CONTROL. Dieser Prozeß kann die im Paket AdaIR zusammengefaßten Objekte benutzen. Dazu gehören Dienste zur Steuerung der Bewegung des Roboters, Datentypen zur Beschreibung des Roboterzustandes, Dienste zur Einstellung und Abfrage der dem Roboter direkt zugeordneten Peripherie, sowie zur Behandlung von Ausnahmen. Die für den Roboter spezifischen geometrischen Größen sind in einem gesonderten Paket KINEMATICS zusammengefaßt, das von AdaIR importiert wird.

Um die Peripherieansteuerung auf Maschinenebene, die Unterbrechungs- und Ausnahmebehandlung sowie die Unterstützung nebenläufiger Prozesse im Interesse einer hohen Portabilität soweit wie möglich ohne Betriebssystemaufrufe abwickeln zu können, wurde als Implementierungssprache Ada gewählt. AdaIR wurde als Ada-Paket realisiert; dem Benutzer stehen daher weiterhin sämtliche Möglichkeiten der Sprache Ada offen.

3 Die Datentypen in AdaIR

Das Paket AdaIR stellt die folgenden vordefinierten Typen zur Verfügung (siehe Anhang A für eine vollständige Spezifikation des Pakets) :

```
AxesVector is
    array (1..MaxJoint) of FLOAT;
```

Beschreibung der Gelenkstellungen als Tupel der Größe MaxJoint. Winkelangaben erfolgen im Bogenmaß, Längenangaben in Millimetern

```
Frame is
    array (1..3, 1..4) of FLOAT;
```

Beschreibung der Stellung mittels einer 3x4-Matrix,bestehend aus Orientierungsteil und Positionsvektor.

```
Motion is
    (JointInterpolated, StraightLine);
```

Bewegungsart.

```
ArmConfig is
    (ArmForward, ArmBackward,
    NoArmConfig);
```

Armgelenkkonfiguration.

```
ElbowConfig is
    (ElbowAbove, ElbowBelow,
    NoElbowConfig);
```

Armgelenkkonfiguration.

```
HandConfig is
    (HandFlip, HandNoFlip,
    NoHandConfig);
```

Handkonfiguration.

```
ToolDescriptor is record
    Trans : Frame;
    ForceMin,
    ForceMax,
    CurrForce : ToolForce;
    DistMin,
    DistMax,
    CurrDist : ToolDistance;
    end record;
```

Beschreibung des Effektors (statische und aktuelle Werte).

```
subtype ErrorCode is NATURAL      Fehlernummer.
    range 0..2000;

subtype PathNo is POSITIVE        Bahnnummer.
    range 1..65000;

subtype PosNo is POSITIVE         Positionsnummer.
    range 1..65000;

subtype ToolForce is FLOAT        Effektorkraft.
    range 0.0..500;

subtype ToolDistance is FLOAT     Effektorabstand.
    range 0.0..100.0;
```

Ein Objekt vom Typ Frame wird intern wie eine homogene 4x4-Matrix behandelt. Deshalb sind die üblichen Operationen zur Verknüpfung von Matrizen erlaubt.

4 Die AdaIR-Dienste

Die AdaIR-Dienste lassen sich in folgende sechs Funktionsgruppen gliedern:

- Initialisierung,
- Steuerung des Effektors,
- Bewegung des Roboters (Stellungsbeschreibung durch Winkelangabe),
- Bewegung des Roboters (Stellungsbeschreibung durch 4x4-Matrizen),
- Definition und Abfahren von Bahnen,

Die Dienste arbeiten synchron, d.h. das Benutzerprogramm wartet nach Aufruf eines solchen Dienstes die Ausführung durch den Roboter bzw. durch den Steuerrechner ab.

Die Beschreibungen der einzelnen Dienste finden sich in alphabetischer Reihenfolge im Anhang A. Im folgenden wird eine Übersicht über die wichtigsten Eigenschaften der einzelnen Klassen von Diensten gegeben.

4.1 Dienste zur Initialisierung

Die Initialisierungsdienste umfassen Befehle zum

- Anfahren der Nullstellung des Roboters
- Kalibrieren der Roboterhardware, Initialisierung der Software
- Setzen und Auslesen von Werten zur Bewegungsbeschreibung.

Der Dienst InitRobot kalibriert den Roboter und fährt ihn in die Nullstellung; diese sind durch die folgenden 4x4-Matrizen gegeben:

RM-501:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 844.6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

RV-M1:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 889.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Die Größen in der Positionsspalte werden in mm angegeben. Durch die Initialisierung wird die Bewegungsart JointInterpolated eingestellt; Bewegungen werden mit der halben Maximalgeschwindigkeit ausgeführt.

Der Dienst EndRobot fährt den Roboter in seine durch die Hardware definierte Ausgangsstellung ("Nest-Position") zurück. Alle bis dahin gesetzten Einstellungen sind verloren.

Zum Kalibrieren während des Betriebs dient der Dienst CalibrateRobot. Der Roboter sollte während des Betriebs etwa alle 30 min neu kalibriert werden, um eine gleichbleibende Positioniergenauigkeit zu gewährleisten.

Die Bewegung des Roboters zwischen zwei Stellungen wird durch drei Parameter beschrieben, die vom Benutzer beeinflußt werden können:

- Geschwindigkeit : SetVelocity,
- Bewegungsart: SetMoveMode,
- Vorzugskonfiguration: SetRobotConfig, ValueRobotConfig.

Die Geschwindigkeitsangabe erfolgt dabei als Prozentwert; der sich auf die Maximalgeschwindigkeit bezieht. Die Dienste zum Setzen und Abfragen der Vorzugskonfiguration sind für den RM-501 und den RV-M1 nicht implementiert.

4.2 Dienste zur Steuerung des Effektors

Es wird davon ausgegangen, daß am Roboter ein Effektor (im Regelfall ein Greifer) montiert ist, der über Sensorik zur Ermittlung der Effektorkraft und -öffnungsweite verfügt. Der Benutzer kann in diesem Fall den Effektor über die Angabe der aufzubringenden Effektorkraft bzw. der zu erreichenden Endstellung steuern. Dazu dient der Dienst MoveTool. Sollte die Sensorik fehlen, so werden entsprechende Standardwerte eingesetzt.

Das kinematische Modell des Roboters ist nur bis zur Montagefläche des Effektors fest vorgegeben. Der Übergang von der Montagefläche zum Tool-Center-Point wird durch eine homogene Transformation beschrieben. Tauscht der Benutzer den Effektor aus, so muß auch die Effektorbeschreibung geändert werden. Dafür steht der Dienst SetTool zur Verfügung.

Nach der Initialisierung ist der Standardeffektor des RM-501 definiert. Der Übergang vom fünften Gelenk zum Tool-Center-Point ist bei diesem Effektor durch die folgende Transformation gegeben :

RM-501:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 149.6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

RV-M1:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 107.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Der Tool-Center-Point liegt in die Spitze des Effektors in der Mitte zwischen den Greiferbacken.

4.3 Bewegungsdienste

Um größtmögliche Flexibilität bei der Umsetzung einer gegebenen Stellungsbeschreibung in die Parameter von AdaIR-Bewegungsdiensten zu erreichen, werden zwei Klassen von Bewegungsdiensten zur Verfügung gestellt. Bei der ersten Klasse werden die Zielstellungen in Form von Gelenkvariablen angegeben, bei der zweiten als homogene 4x4-Matrizen.

4.3.1 Gelenkvariable

Es stehen zwei Dienste zur Bewegung des Roboters über die Angabe der anzufahrenden Gelenkvariablenwerte zur Verfügung : MoveAxesAbsolute und MoveAxesRelative. Beiden Diensten wird als Parameter ein Vektor vom Typ AxesVector übergeben. Die Indizes der Vektorkomponenten entsprechen den Nummern der Gelenke in Bild 4.1. Winkelwerte werden im Bogenmaß angeben, wobei Vielfache von 2π automatisch subtrahiert werden; Strecken werden in mm angegeben.

Nach Ausführung des Dienstes MoveAxesAbsolute befinden sich alle Gelenke des Roboters auf dem für das jeweilige Gelenk übergebenen absoluten Gelenkvariable. Als Nullstellung (alle Winkel 0) gilt der gerade, nach oben ausgestreckte Arm. Diese Nullstellung darf nicht mit der optimalen Nullage zur Bestimmung der D-H-Matrizen verwechselt werden. Bei der Positionierung mit dem Dienst MoveAxesRelative bezieht sich die Angabe der Winkelwerte auf die vor Aufruf des Dienstes eingenommene Stellung des Roboters. Der absolute Wert einer Gelenkvariable nach Ausführung dieses Dienstes berechnet sich als Summe des absoluten Wertes vor dem Aufruf und dem als Parameter übergebenen Wert.

Kann ein Winkelwert in einer Drehrichtung nicht angefahren werden, so versucht die Steuerung, den Wert in der anderen Richtung zu erreichen.

Es existiert ferner eine Funktion zum Auslesen der Gelenkvariablen (ValueAxes). Bei der Interpretation der von ihr gelieferten Werte ist zu berücksichtigen, daß die Auflösung der Gelenkvariablen lediglich $1/40^{o}$ (entspricht etwa dem Winkel 0.000436 im Bogenmaß) beträgt. Dadurch kann der Roboter die angegebenen Winkel nur in Ausnahmefällen genau anfahren. Werden nach Anfahren einer Stellung die eingestellten Gelenkwerte wieder ausgelesen, so ergeben sich in der Regel leicht abweichende Werte.

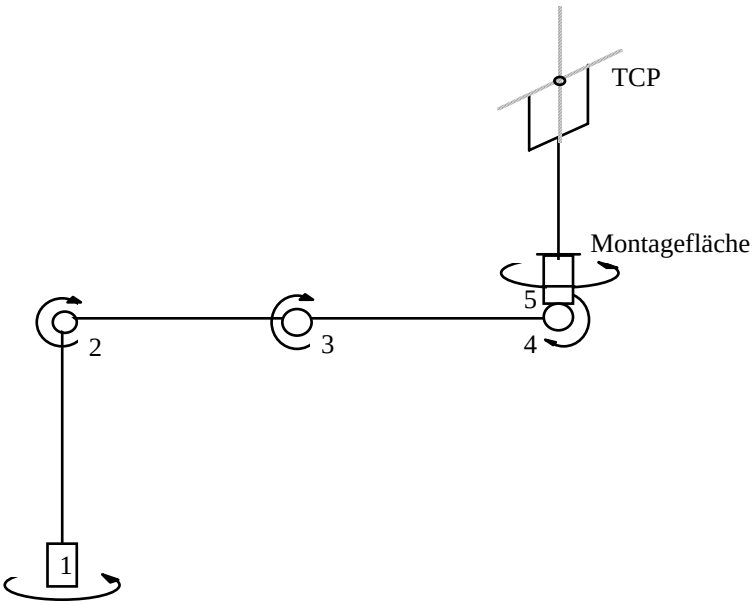


Bild 4.1 : Gelenknummern mit positiven Drehrichtungen

4.3.2 Homogene 4x4-Matrizen

Zur Beschreibung der Stellung des Effektors im dreidimensionalen Raum dienen homogene 4x4-Matrizen (in der englischen Literatur auch als Frames bezeichnet) :

$$\begin{bmatrix} O_{11} & O_{12} & O_{13} & T_X \\ O_{21} & O_{22} & O_{23} & T_Y \\ O_{31} & O_{32} & O_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} = F$$

$O_{n,m}$ ($1 \le m, n \le 3$) beschreibt die Orientierung des Effektors, T_k ($1 \le k \le 3$) seine Position in Millimetern bezogen auf den Tool-Center-Point. Da die unterste Zeile der 4x4-Matrix immer gleich ist, wird sie in den entsprechenden Datenstrukturen weggelassen. Zu beachten ist, daß es sich auch bei den vom System benutzten 3x4-Matrizen mathematisch gesehen nach wie vor um 4x4-Matrizen handelt.

Die Steuerung über die Angabe von 4x4-Matrizen erfolgt absolut (MoveFrameAbsolute) oder relativ (MoveFrameRelative). Die bei der absoluten Positionierung anzugebende 4x4-Matrix bezieht sich auf das Basiskoordinatensystem des Roboters.

Eine relative 4x4-Matrix ^{REL}OBJ beschreibt die Stellung des Effektors in bezug auf ein vorher zu definierendes (Objekt-)Koordinatensystem ^{BKS}REL. Letzteres ist vom Benutzer bezüglich des Basiskoordinatensystems mit Hilfe des Dienstes SetBase anzugeben. Für die Stellung des Objekts im Raum bezüglich BKS gilt :

$$^{BKS}\text{REL} * {}^{REL}\text{OBJ} = {}^{BKS}\text{OBJ}.$$

Nach der Initialisierung durch den Dienst InitRobot fällt das durch SetBase definierte Koordinatensystem des Objekts mit dem Basiskoordinatensystem des Roboters zusammen. Bis zu einer Neudefinition von REL ist die absolute Steuerung mit der relativen identisch.

4.4 Dienste zum Abfahren von Bahnen

Für bestimmte Aufgaben ist es notwendig, den Roboter nicht beliebig von der Start- zur Zielstellung, sondern über Zwischenstellungen kontinuierlich zu bewegen. Start-, Ziel- und Zwischenstellungen, geordnet in der Reihenfolge ihrer Anfahrt, ergeben eine Bahn, zu deren Verwaltung ein Satz von Diensten existiert. Innerhalb des Systems werden Bahnen als Listen geführt, denen dynamisch Speicherplatz zugewiesen wird.

Den zeitlichen Ablauf des Abfahrens einer Bahn verdeutlicht Bild 4.2. Eine Bahn wird mit Hilfe des Dienstes CreatePath erzeugt, wobei als Parameter die Bahnnummer angegeben wird. Als Bahnnummern sind ganze Zahlen im Wertebereich von PathNo zugelassen.

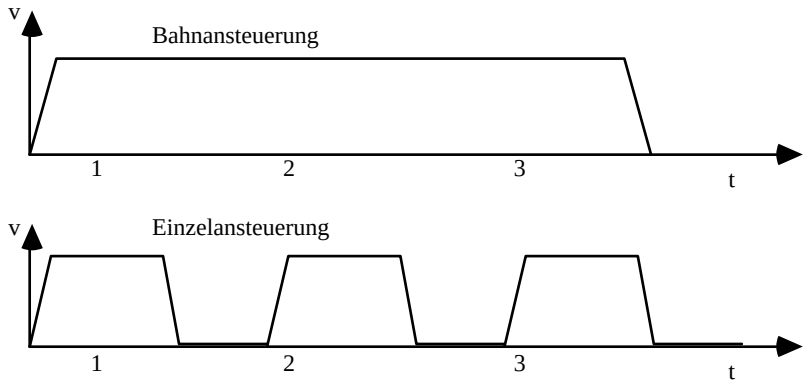


Bild 4.2: Bahn- und Einzelansteuerung

Eine Bahn wird als einfach verkettete Liste der nacheinander anzufahrenden Stellungen repräsentiert. Auf dieser Liste sind die folgenden Operationen definiert:

- Einfügen: InsertPosition
- Löschen: DeletePosition
- Auslesen von Positionen: ValuePosition
- Abfahren der Bahn durch den Roboter: MovePath

Die Operationen *Einfügen* und *Löschen* sind in der bei verketteten Listen üblichen Weise definiert. Erster Parameter ist grundsätzlich die Bahnnummer. Der Parameter PositionNumber ist der Index der entsprechenden Stellung innerhalb der Liste; der Wert des letzten Feldelements ist stets gleich Null. Bei Ausführung des Dienstes InsertPosition wird die neue Stellung in die Liste vor der mit PositionNumber bezeichneten Stelle eingefügt. Existiert die Positionsnummer schon, schiebt die neue Stellung die Stellungen mit höherer Nummer um eine Position weiter und belegt die angegebene Listenstelle. Sollte die Positionsnummer noch nicht in der Liste sein, wird die Stellung ans Ende der Liste angehängt. Der Stellung wird dann die Nachfolgenummer der jeweils letzten bisherigen Stellung zugewiesen. Beim Löschen (DeletePosition) wird die unter PositionNumber abgelegte Stellung aus der Liste entfernt. Eine gesamte Bahn kann durch den Dienst DeletePath gelöscht werden.

Zusätzlich können mit dem Dienst SavePathSystem alle dem System bekannten Bahnen gespeichert werden. Mit LoadPathSystem können Bahnen aus einer Datei gelesen werden.

Ein Beispiel soll das Zusammenspiel der Dienste verdeutlichen:

```
CreatePath (1);                 -- Bahn 1 wird erzeugt
InsertPosition (1, 1, Frame1);      -- Frame1 auf Position 1 von Bahn1
InsertPosition (1, 1, Frame2);      -- Frame2 jetzt auf Position 1
                                    -- und Frame1 auf Position 2  von Bahn1
InsertPosition (1, 3, Frame3);      -- Frame3 auf Position 3 von Bahn1
InsertPosition (1, 3, Frame4);      -- Frame4 jetzt auf Position 3
                                    -- und Frame3 auf Position 4
DeletePosition (1, 4);              -- Löscht Frame3
InsertPosition (1, 3, Frame3);      -- Frame3 vor Frame4
MovePath(1);                        -- Die Reihenfolge von Bahn1 ist:
                                -- Frame2, Frame1, Frame3, Frame4
```

Bei extremen Richtungsänderungen während der Bewegung auf Bahnen unterliegt der Roboter großen mechanischen Belastungen, die zu seiner Zerstörung führen können. Es ist deshalb zu vermeiden, Bahnen mit großen Richtungsänderungen mit hoher Geschwindigkeit auszuführen.

4.5 Peripherieanbindung

Wegen der Vielfalt der anschließbaren Peripherie ist es nicht möglich, eine allgemeingültige Schnittstelle für Peripherieprozesse zu spezifizieren. Aus Gründen der Einheitlichkeit der Programmierung wird aber eine Struktur für solche Prozesse vorgeschlagen.

Periphere Einheiten, die nicht direkt dem Roboter zugeordnet sind, werden durch Gerätetreiber in Form von Ada-Prozessen in AdaIR-Programme eingebunden. Die Implementierung dieser Prozesse ist geräteabhängig; Deklaration und Implementierung sind vom Benutzer vorzunehmen.

Für jede periphere Einheit muß vom Benutzer ein Prozeß zur Verfügung gestellt werden, der die Leistungen der peripheren Einheit dem Benutzerprozeß CONTROL in Form von Rendezvous-Eintrittspunkten zur Verfügung stellt. Ein Peripherie-Prozeß soll folgende Rendezvous-Eintrittspunkte zur Verfügung stellen:

- Initialize (Initialisierung der Peripherie)
- SetValue (Einstellen eines Wertes auf den Ausgängen)
- GetValue (Lesen eines Wertes von den Eingängen)
- EnableInterrupt (Zulassen von Unterbrechungen durch die Hardware)
- DisableInterrupt (Unterdrücken von Unterbrechungen durch die Hardware)

Für die Behandlung von Unterbrechungen durch die Peripheriehardware muß ein gerätegebundener Prozeß mit dem Eintrittspunkt Interrupt spezifiziert werden. Bei Auftritt einer Unterbrechung wird vom Ada-Laufzeitsystem deren Eintrittspunkt aufgerufen. Die eigentliche Behandlung der Unterbrechung bleibt dem Benutzer überlassen. Für die Kommunikation und Synchronisation mit dem Benutzerprogramm können Rendezvous-Eintrittspunkte zur Verfügung gestellt werden. Unterbrechungen durch die Hardware müssen explizit zugelassen werden und können unterdrückt werden.

5 Ausnahmebehandlung

AdaIR bietet dem Benutzer die Möglichkeit, Routinen zur Ausnahme- bzw. Fehlerbehandlung selbst zu definieren. Dadurch wird es möglich, nach Auftritt eines Fehlers einen definierten Zustand herzustellen und den Programmlauf evtl. weiterzuführen.

Das Paket AdaIR exportiert verschiedene Ausnahmen (Exceptions), die bei der Ausführung von AdaIR-Diensten auftreten können; ihre Namen finden sich in Anhang B. Zur Behandlung der Ausnahmen steht dem Benutzer damit der gesamte Mechanismus der Sprache Ada zur Verfügung: Tritt in einer Programmeinheit eine Ausnahme auf und ist in dieser Einheit keine Behandlungsroutine vorgesehen, so wird die Bearbeitung der Einheit abgebrochen und die Ausnahme pflanzt sich in die umgebende Einheit fort. Findet sich dort eine passende Behandlungsroutine, so wird diese ausgeführt und auch die zugehörige Einheit verlassen. Andernfalls wird so lange in der jeweils umgebenden Programmeinheit gesucht, bis entweder eine passende Routine gefunden wird oder die Ebene der Task erreicht ist. Erst wenn sich auch hier keine Routine findet, wird der Prozeß unter Ausgabe des Namens der Ausnahme abgebrochen. Auf diese Art und Weise ist es möglich, sehr detailliert festzulegen, auf welcher Stufe welche Ausnahmen (mit unterschiedlicher Signifikanz für das Gesamtprogramm) behandelt werden sollen.

Über einen jedem AdaIR-Dienst zugeordneten Befehlscode kann der Benutzer im Falle eines Fehlers den Dienst ermitteln, in dem der Fehler auftrat. Dies geschieht durch Aufruf des Dienstes ValueErrorStatus in der Fehlerbehandlungsroutine.

Zu beachten ist, daß eine Endlosschleife entstehen kann, wenn der Benutzer in seiner Fehlerbehandlungsroutine AdaIR-Dienste aufruft, die fehlerhaft verlaufen. Geeignete Abbruchkriterien sind deshalb zu implementieren.

6 Teach-In

Zur manuellen Steuerung des Roboters ist der Anschluß einer Teach-Box oder die Nutzung einer bildschirmorientierten Software-Teachbox vorgesehen.

6.1 Die Teach-Box

Für den RM-501 steht eine Teach-Box zur Verfügung. Zu ihrer Nutzung muß der Schalter auf der Oberseite der Box auf ON gestellt werden. Am Ende des Teach-In-Vorgangs muß die Teach-Box ausgeschaltet werden (Position OFF). Solange die Box angeschaltet ist, kann die Robotersteuerung keine vom Benutzerprogramm angeforderten Aktionen ausführen, d.h. während des Teach-In-Vorganges darf das Benutzerprogramm keine AdaIR-Dienste aufrufen. Sollte dies dennoch geschehen, wird die Ausnahme RobotBusy ausgelöst.

Zum Teach-In einer Stellung empfiehlt sich deshalb folgender Ablauf :

1. Das Benutzerprogramm gibt auf dem Sichtgerät des Benutzerrechners eine Meldung aus, daß der Punkt erreicht ist, an dem das Teach-In stattfinden kann.
2. Das Benutzerprogramm wartet nun so lange, bis es eine Quittung des Benutzers über den Abschluß des Teach-Vorgangs erhält. Dazu kann das Benutzerprogramm beispielsweise die parameterlose Standardprozedur Get_LineS aufrufen, die das Benutzerprogramm bis zu einer Eingabe von <CR> suspendiert.
3. Der Benutzer führt den Teach-Vorgang aus, d.h. er fährt den Roboter an die gewünschte Position.
4. Der Benutzer bestätigt die Beendigung des Teach-Vorgangs auf der Tastatur des Benutzerrechners (durch Eingabe von <CR>).
5. Das Benutzerprogramm kann nun durch Aufruf der Dienste ValueFrameAbsolute, ValueAxes oder ValueTool (Status läßt sich durch Teach Box nicht ändern) den aktuellen Zustand des Roboters auslesen und in eigene Variablen übertragen.

Über die Teach-Box ist es möglich, Speicherzellen in der Robotersteuerung direkt zu manipulieren. Eine solche Manipulation ist verboten, da diese Zellen auch vom Steuerrechner benutzt werden. Die Teach-Box darf nur dazu benutzt werden, die gewünschte Stellung anzufahren.

6.2 Der Dienst TeachBox

Eine weitaus komfortablere Teach-Box bietet der AdaIR-Dienst TeachBox. Er enthält alle notwendigen Anweisungen für einen fehlerfreien Ablauf. Durch Aufruf des Dienstes entsteht auf dem Bildschirm des Steuerrechners eine Teachbox, mit deren Hilfe Parameter wie beispielsweise die Geschwindigkeit der Roboterbewegungnen oder die Position des Roboters verändert werden können. Analog zur gewöhnlichen Teach-Box können nach Rückkehr in das Benutzerprogramm die aktuellen Roboterparameter durch Aufruf von AdaIR-Diensten ausgelesen werden.

Anhang A Die AdaIR-Dienste

CalibrateRobot

Befehlscode: 40.
Definition:

procedure CalibrateRobot;

Beschreibung:

Kalibrierung des Roboters. Muß der verwendete Robotertyp zur Kalibrierung eine bestimmte Stellung einnehmen, so wird diese angefahren.

Besonderheiten:

Das Anfahren einer Kalibrierungsstellung erfolgt auf direktem Weg, entsprechend der eingestellten Bewegungsart.

CreatePath

Befehlscode: 70.
Definition:

procedure CreatePath (PathNumber : **in** PathNo);

Beschreibung: Erzeugt eine Bahn mit der Ordnungsnummer PathNumber.

DeletePath

Befehlscode: 80.
Definition:

procedure DeletePath (PathNumber : **in** PathNo);

Beschreibung:

Löscht die Bahn mit der Ordungsnummer PathNumber.

DeletePosition

Befehlscode: 90.
Definition:

procedure DeletePosition (PathNumber: **in** PathNo; PositionNumber : **in** PosNo);

Beschreibung:

Löscht die durch PositionNumber spezifizierte Bahnposition in der durch die Ordnungsnummer PathNumber bezeichneten Bahn.

EndRobot

Befehlscode: 110.
Definition:

procedure EndRobot;

Beschreibung:

Fährt den Roboter in seine Haltestellung. Alle zuvor durch Aufruf von AdaIR-Diensten gewonnenen Daten sind verloren.

Besonderheiten:

Die Haltestellung wird auf direktem Weg angefahren, der eingestellten Bewegungsart entsprechend.

InitRobot

Befehlscode: 150.
Definition:

procedure InitRobot;

Beschreibung:

Der Roboter wird kalibriert und in die Nullposition (senkrecht erhobener Arm) gefahren. Bei Installation eines Greifers wird dieser geöffnet, und die Greifkraft wird auf den Wert der Systemkonstanten InitToolForce gesetzt.

Besonderheiten:

Der Dienst InitRobot muß im Benutzerprogramm als erster Dienst aufgerufen werden. Es ist dafür Sorge zu tragen, daß bei der Fahrt in die Nullstellung keine Kollisionen auftreten.

InsertPosition

Befehlscode: 160.
Definition:

procedure InsertPosition (PathNumber : **in** PathNo; PositionNumber : **in** PosNo;
Position : **in** Frame);

Beschreibung:

Die Stellung Frame wird an der Stelle PositionNumber in die durch PathNumber spezifizierte Bahn eingetragen. Ist die PositionNumber schon belegt, werden alle Stellungen mit höherer PositionNumber um eine Stelle verschoben, um Platz für die neue Stellung zu schaffen. Ist die PostionNumber noch nicht vergeben, wird die Stellung ans Ende der Bahn angehängt.

LoadPathSystem

Befehlscode: 170.
Definition:

procedure LoadPathSystem (PathFileName : **in** String);

Beschreibung:

Lädt Stellungsbeschreibungen von einem Sekundärspeichermedium in die interne Bahnverwaltung des AdaIR-Systems.

MoveAxesAbsolute

Befehlscode: 180.
Definition:

procedure MoveAxesAbsolute (Axes : **in** AxesVector;Time: **in** DURATION := 0.0; Control: **in** BOOLEAN := FALSE; RealTime: **in** BOOLEAN := FALSE);

Beschreibung:

Die Gelenke des Roboters werden gemäß den Angaben im Winkelvektor Axes eingestellt. Die optionalen Parameter Time, Control und RealTime werden lediglich zur Synchronisation von Aktionen mehrer Roboter benötigt, im Normalbetrieb werden die Default-Einstellungen verwendet.

Besonderheiten:

Bei Rotationsgelenken spezifiziert das Vorzeichen eines Gelenkwinkelwerts die Drehrichtung, mit der der Winkelwert eingestellt wird. Ist die Ansteuerung des Winkels nicht in der Drehrichtung möglich, die der Benutzer angibt, so versucht die Steuerung, den Winkel in der anderen Richtung anzufahren.

MoveAxesRelative

Befehlscode: 190.
Definition:

procedure MoveAxesRelative (Axes : **in** AxesVector);
Time: **in** DURATION := 0.0;
Control: **in** BOOLEAN := FALSE;
RealTime: **in** BOOLEAN := FALSE);

Beschreibung:

Die Gelenke des Roboters werden gemäß den Angaben im Winkelvektor Axes relativ zur aktuellen Stellung eingestellt. Bezüglich der optionalen Parameter siehe MoveAxesAbsolute.

Besonderheiten:

Bei Rotationsgelenken spezifiziert das Vorzeichen eines Gelenkwinkelwerts die Drehrichtung, mit der der Winkelwert eingestellt wird. Ist die Ansteuerung des Winkels nicht in der Drehrichtung möglich, die der Benutzer angibt, so versucht die Steuerung, den Winkel in der anderen Richtung anzufahren.

MoveFrameAbsolute

Befehlscode: 200.
Definition:

```
procedure MoveFrameAbsolute (Goal :  in Frame);
                Time:  in DURATION := 0.0;
                Control:       in BOOLEAN := FALSE;
                RealTime:     in BOOLEAN := FALSE);
```

Beschreibung:

Steuert den Effektor des Roboters in die Stellung Goal, bezogen auf das Basiskoordinatensystem des Roboters. Bezüglich der optionalen Parameter siehe MoveAxesAbsolute.

Besonderheiten:

Ist die Stellung Goal in verschiedenen Konfigurationen erreichbar, wird die Gelenkeinstellung gewählt, die der mittels SetRobotConfig gewählten Vorzugskonfiguration entspricht.

MoveFrameRelative

Befehlscode: 210.
Definition:

```
procedure MoveFrameRelative (Goal :   in Frame);
                Time:  in DURATION := 0.0;
                Control:       in BOOLEAN := FALSE;
                RealTime:     in BOOLEAN := FALSE);
```

Beschreibung:

Steuert den Roboter in die Stellung Goal, bezogen auf das durch DefineBase definierte Basiskoordinatensystem. Bezüglich der optionalen Parameter siehe MoveAxesAbsolute.

Besonderheiten:

Ist die Stellung Goal in verschiedenen Konfigurationen erreichbar, wird die Gelenkeinstellung gewählt, die der momentanen Vorzugskonfiguration entspricht. Nach Ausführung des Dienstes InitRobot ist das Basiskoordinatensystem des Roboters als Bezug definiert. Die Dienste MoveFrameAbsolute und MoveFrameRelative arbeiten bis zur Neudefinition des Basiskoordinatensystems durch SetBase gleich.

MovePath

Befehlscode: 220.
Definition:

```
procedure MovePath (Pathnumber:       in PathNo;
                Time:  in DURATION := 0.0;
                Control:       in BOOLEAN := FALSE;
                RealTime:     in BOOLEAN := FALSE);
```

Beschreibung:

Fährt die Stellungen der durch PathNumber spezifizierten Bahn ab. Bezüglich der optionalen Parameter siehe MoveAxesAbsolute.

Besonderheiten:

Aufgrund abrupter Richtungsänderungen während der Bahnfahrt kann der Roboter beschädigt werden. Der Benutzer hat dies zu vermeiden.

MoveTool

Befehlscode: 230.

Definition:

procedure MoveTool (Distance: **in** ToolDistance; Force : **in** ToolForce);

Beschreibung:

Schließt oder öffnet die Hand, bis entweder der übergebene Greiferabstand Distance (in Millimetern) oder die vorgegebene Kraft Force (in Newton) erreicht ist. Die Angabe der Weite hat dabei im Falle eines Widerspruches (z.B. kleinerer Öffnungsweite bei kleinerer Kraft) Vorrang.

Besonderheiten:

Fehlt eine Sensorik zur Erfassung der Greifweite, öffnet jede Weitenangabe über 0.0 den Effektor bis zur angegebenen Kraft. Wird der Wert 0.0 übergeben, schließt sich der Effektor bis zur angegebenen Kraft. Verfügt der Effektor über keine Sensorik zur Krafterfassung, wird nur die Angabe über die einzustellende Griffweite ausgewertet. Fehlt jegliche Sensorik, öffnet jede Weitenangabe über 0.0 den Effektor, 0.0 schließt ihn.

SavePathSystem

Befehlscode: 260.

Definition:

procedure SavePathSystem (PathFileName : **in** String);

Beschreibung:

Speichert die in der internen Bahnverwaltung des AdaIR-Systems gehaltenen Stellungsbahnen auf ein Sekundärspeichermedium.

SetBase

Befehlscode: 280.

Definition:

procedure SetBase (Base : **in** Frame);

Beschreibung:

Ändert das Bezugskoordinatensystem der Dienste MoveFrameRelative und ValueFrameRelative vom absoluten Koordinatensystem des Roboters nach Base. Diese Änderung hat nur Auswirkungen auf die Dienste MoveFrameRelative und ValueFrameRelative. Die dem Dienst MoveFrameRelative übergebene 4x4-Matrix wird zunächst mit Base multipliziert. Danach wird die sich hierbei ergebende Stellung angefahren. Entsprechend liefert ValueFrameRelative die 4x4-Matrix, die Base in die aktuelle Effektorstellung transformiert.

Besonderheiten:

Nach der Initialisierung durch den Dienst InitRobot sind die Basiskoordinatensysteme für relative und absolute 4x4-Matrizen identisch. Es handelt sich in beiden Fällen um das Basiskoordinatensystem des Roboters.

SetInterpolation

Befehlscode: 300.
Definition:

```
procedure SetInterpolation (Width : in Interpolation);
```

Beschreibung:

Definiert den minimalen Abstand der Stützpunkte zur Interpolation linearer Roboterbewegungen. Der Stützpunktabstand wird in Millimetern angegeben.

SetMoveMode

Befehlscode: 290.
Definition:

```
procedure SetMoveMode (Kind : in Motion);
```

Beschreibung:

Definiert die Bewegungsart des Roboters.

SetRobotConfig

Befehlscode: 310.
Definition:

```
procedure SetRobotConfig (ArmMode:   in ArmConfig;
                          ElbowMode: in ElbowConfig;
                          HandMode:  in HandConfig);
```

Beschreibung:

Definiert die Konfiguration des Roboters, die bei einer mehrdeutigen Stellungsbeschreibung durch 4x4-Matrizen einzunehmen ist.

SetTool

Befehlscode: 320.
Definition:

```
procedure SetTool (Tool : in ToolDescriptor );
```

Mit:

```
type ToolDescriptor is record
    Trans:                          Frame;
    ForceMin, ForceMax, CurrForce:  ToolForce;
    DistMin, DistMax, CurrDist:     Tooldistance;
end record;
```

Beschreibung:

Definiert die den Effektor beschreibende Datenstruktur. Trans stellt die kinematische Beschreibung des Übergangs von der Montagefläche des Effektors am Roboter zum Tool-Center-Point dar. Verfügt der Effektor über Sensorik zur Erfassung der Kraft in Schließrichtung des Effektors, wird in ForceMin und ForceMax die minimale bzw. maximale Kraft übergeben. Fehlt diese Sensorik, so muß beiden Variablen der Wert 0.0 zugewiesen werden. DistMin und DistMax geben die von einem Sensor im Effektor gemessene minimale und maximale Schließweite des Effektors an. Fehlt dieser Sensor, so muß beiden Variablen der Wert 0.0 zugewiesen werden. Die Angabe der Kräfte erfolgt in Newton, die der Weiten in Millimetern. Die übergebenen aktuellen Werte bezüglich der Greifweite und der Effektorkraft werden ignoriert.

Besonderheiten:

Nach der Initialisierung durch den Dienst InitRobot ist als Effektor der Standardeffektor des jeweiligen Robotertyps definiert.

SetVelocity

Befehlscode: 330.

Definition:

```
procedure SetVelocity (Velocity : in RobotVelocity);
```

Beschreibung:

Legt die Bewegungsgeschwindigkeit des Roboters fest. Die Geschwindigkeit einer Roboterbewegung ist im AdaIR-System durch die Winkelgeschwindigkeit in grad/s definiert, mit der das Gelenk bewegt wird, das den größten Winkel zu fahren hat.

TeachBox

Befehlscode: 350.

Definition:

```
procedure TeachBox (Text:        in STRING := " ";
                    REFRESH:     in BOOLEAN := TRUE;
                    CLEAR:       in BOOLEAN := TRUE);
```

Beschreibung:

Für Roboter mit Gelenkmotorschnittstelle bietet dieser Dienst eine bildschirmunterstützte Software-Teachbox, mit der über die Tastatur der Rechnerkonsole die wichtigsten Teach-Operationen vorgenommen werden können. Über den Parameter Text kann eine Zeichenkette zur Information des Benutzers auf dem Bildschirm ausgegeben werden. Um bei mehrmaliger Benutzung der Teachbox das eventuell unnötige Löschen und Ausgeben der Grafik zu vermeiden, kann die Grafikausgabe zur Darstellung der Software-Teachbox über die Parameter Refresh und Clear benutzerseitig kontrolliert werden. Nach dem Anfahren einer Stellung mit der Teachbox kann die Stellung des Roboters mit Hilfe der Valuexx-Dienste eingelesen werden.

Besonderheiten:

TeachBox besteht aus mehreren AdaIR-Diensten. Durch die Programmiereinheit kann auch der Status des Effektors verändert werden, deshalb sollte nach Aufruf von TeachBox mit Hilfe des Dienstes ValueTool der neue Effektorstatus ausgelesen werden.

ValidFrame

Befehlscode: 370.
Definition:

procedure ValidFrame (VaryFrame : **in out** FRAME);

Beschreibung:

Der Dienst ValidFrame ist nur erforderlich, falls ein Roboter mit weniger als sechs Freiheitsgraden verwendet wird. Für Roboter dieser Art sind bestimmte Effektorpositionen innerhalb des Arbeitsbereiches nicht in jeder beliebigen Orientierung erreichbar. Daher werden nur durch bestimmte homogene 4x4-Matrizen erreichbare Stellungen beschrieben. Der Dienst ValidFrame verändert den Orientierungsteil der übergebenen 4x4-Matrix VaryFrame so, daß die jeweils vorgegebene Stellung optimal angenähert wird. Dabei entsteht eine natürliche und sinnvolle Anpassung des Orientierungsteils.

ValueAxes

Befehlscode: 380.
Definition:

procedure ValueAxes (Axes: **out** AxesVector);

Beschreibung:

Liefert die aktuellen Winkelwerte der Gelenke im Bogenmaß.

ValueErrorStatus

Befehlscode: 390.
Definition:

procedure ValueErrorStatus (Status: **out** ErrorCode);

Beschreibung:

Sobald eine Ausnahme ausgelöst wird, liefert dieser Dienst den Befehlscode des AdaIR-Dienstes, in dem die Ausnahme aufgetreten ist.

ValueFrameAbsolute

Befehlscode: 400.
Definition:

procedure ValueFrameAbsolute (ReturnFrame: **out** Frame);

Beschreibung:

Liest die aktuelle Effektorstellung als 4x4-Matrix bezüglich des Basiskoordinatensystems des Roboters aus.

ValueFrameRelative

Befehlscode: 410.
Definition:

procedure ValueFrameRelative (ReturnFrame: **out** Frame);

Beschreibung:

Liest die aktuelle Effektorstellung bezüglich des Koordinatensystems aus, das der Benutzer durch den Dienst SetBase gesetzt hat

ValuePosition

Befehlscode: 420.
Definition:

```
procedure ValuePosition(PathNumber:   in PathNo;
                        PositionNumber:   in PosNo;
                        ReturnFrame:      out Frame);
```

Beschreibung:

Gibt in ReturnFrame die durch PositionNumber spezifizierte Stellung in der durch PathNumber angegebenen Bahn zurück.

ValueRobotConfig

Befehlscode: 430.
Definition:

```
procedure ValueRobotConfig (ArmMode:      out ArmConfig;
                            ElbowMode:  out ElbowConfig;
                            HandMode:   out HandConfig);
```

Beschreibung:

Liefert die Gelenkkonfiguration der aktuellen Roboterstellung.

ValueTool

Befehlscode: 440.
Definition:

```
procedure ValueTool (Tool: out ToolDescriptor);
```

Beschreibung:

Gibt den momentan gültigen ToolDescriptor aus. Zu beachten ist, daß dieser Deskriptor sowohl die statische Beschreibung des Effektors als auch dessen aktuellen Zustand enthält.

Anhang B Vordefinierte Ausnahmen

Das Paket AdaIR exportiert Ausnahmen für die folgenden Fehler:

Fehler	Klasse	möglicher Fehlergrund
CommandNotImplemented	Sys	Dienst ist nicht implementiert
CommandNotSupported	Sys	Dienst oder aktueller Wert eines Parameters wird durch den installierten Roboter nicht unterstützt
ControlCommandActive	User	Versuch, den Roboter zu deinitialisieren, obwohl nicht alle Kontroll-Dienste terminiert sind
DistanceInvalid	User	Greifweite außerhalb des möglichen Bereichs
FatalCommunicationError	Sys	Verbindung mit dem Roboter ist zusammengebrochen
FatalNetError	Sys	Unbekannter Fehler bei der Netzwerkkommunikation
ForceInvalid	User	Greifkraft außerhalb des möglichen Bereichs
FrameInvalid	User	Frame nicht korrekt
MessageInvalid	User	Es wird eine Nachricht empfangen, die nicht dem Datentyp entspricht, der empfangen werden soll
MessageLost	Sys	Eine Nachricht ging verloren oder wurde nicht korrekt übermittelt
MessageOverflow	User	Eine Nachricht ist zu groß
NoConnection	User	Keine Verbindung aufgebaut
NoRobotInstalled	Sys	Kein Roboter angeschlossen
OutOfReach	User	Versuch, Stellung außerhalb des Arbeitsbereiches anzufahren
PathAlreadyDefined	User	Versuch, eine Bahn mehrmals zu kreieren
PathElementNotDefined	User	Versuch, auf eine nicht gespeicherte Bahnstellung zuzugreifen
PathMemOverflow	Sys	Kein Speicherplatz für neue Bahn oder neues Bahnelement
PathNotDefined	User	Versuch, eine Operation auf einer nicht existenten Bahn durchzuführen
RobotBusy	Sys	*Teach-Box* ist angeschaltet
SystemNotInitialized	User	System wurde nicht durch 'InitRobot' initialisiert
TimeInvalid	User	Es ist unmöglich, einen definierten Zeitauftrag einzuhalten
UserInfoInvalid	User	Zeichenkette, die dem Dienst 'TeachBox' zur Benutzerinformation übergeben wird, ist zu lang oder ungültig
WrongRobotInstalled	User	Es steht nicht der Roboter zur Verfügung, für den das Paket 'ADAIR_ROBOT' instantiiert wurde

Fehler der Klasse *User* treten bei falscher Programmierung durch den Benutzer auf (insbesondere bei fehlerhaften Parametern), Fehler der Klasse *Sys* werden ausgelöst, wenn ein Fehler am Rechner, am Roboter, oder auf der Datenleitung vorliegt.

Die Ausnahmebehandlungsroutinen können unter den oben angegebenen Namen wie in Ada üblich spezifiziert werden.

Im folgenden wird ein Beispiel für ein vollständiges Programm mit Ausnahmebehandlung angegeben. Der Roboter (hier RM-501) soll zwei verschiedene Stellungen anfahren. Die erste ist erreichbar; die zweite liegt zwar innerhalb des Arbeitsraumes, ist jedoch mit der geforderten Orientierung nicht erreichbar. In letzerem Fall soll dafür gesorgt werden, daß die der unerreichbaren Stellung am nächsten gelegene und erreichbare angefahren wird.

```
with AdaIR; use AdaIR;

with SERIAL_IO; use SERIAL_IO;

procedure Demo is

--  Terminalport 2 verwenden:
Term_C: constant Term_T := 2;

F1, F2 : Frame;

STATUS : Error

   procedure MoveAnyway (F : inout Frame) is
   begin
      MoveFrameAbsolute (F);
   exception
         when FrameInvalid =>
                  PutS(Term_C,´Anpassung der Stellung,da unerreichbar´);
                  F := ...  -- Hier Code zur Stellungsanpassung
                  MoveFrameAbsolute (F);

         when others  =>
                  PutS (Term_C,´Fehler , Dienstnummer: ´);
                  ValueErrorStatus (STATUS);
                  PutS (Term_C,STATUS);
                  New_LineS(Term_C);
                  raise; -- Brich alles ab

    end MoveAnyway;
```

```
begin
    -- Erreichbares Frame:
    F1 := ((1.0, 0.0, 0.0, 0.0), (0.0, 1.0, 0.0, 0.0), (0.0, 0.0, 1.0, 830.6));
    -- Nicht erreichbares Frame
    F2 := ((0.0, -1.0, 0.0, 0.0), (1.0, 0.0, 0.0, 0.0), (0.0, 0.0, 1.0, 830.6));
    InitRobot;
    MoveAnyway (F1);
    MoveAnyway (F2); -- Hier Anpassung erforderlich
    EndRobot;
end Demo;
```

Anhang C Peripherieanbindung

Ein Beispiel für ein Paket zur Einbindung einer peripheren Einheit ist im folgenden angegeben. Das Beispiel zeigt ausschnittsweise die Implementierung für eine Parallel-Schnittstelle mit je einem 16 Bit Register als Eingang bzw. als Ausgang sowie einer Unterbrechung. Der Prozeß Parallel stellt die für den Betrieb der peripheren Einheit notwendigen Diente zur Verfügung. Der Prozeß Handler implementiert die Reaktion auf eine aufgetretene Unterbrechung.

```
package Parallelpaket is

type t_16_bit is array (0..15) of boolean;

task Parallel is
   entry  Initialize;
   entry SetValue(Val: in t_16_bit);
   entry GetValue(Val: out t_16_bit);
   entry EnableInterrupt;
   entry DisableInterrupt;
end Parallel;

task Handler is
   entry Interrupt;
   for Interrupt use at 8#340#;
   entry Benachrichtigung(Val: out t_16_bit);
end Handler;

end Parallelpaket;
```

```
package body Parallelpaket is

task body Parallel is
begin
  loop
    select
        accept Initialize;
        -- Initialisierung der Hardwareregister
    or  accept  SetValue(Val: in t_16_bit) do
            --  Zuweisen des Wertes Val an das Ausgabe-Register
        end SetValue;
    or  accept  GetValue(Val: out t_16_bit) do
            Val := -- aktueller Inhalt des Eingabe-Registers
        end GetValue;
    or  accept  EnableInterrupt;
        -- Zulassen von Unterbrechungen durch Manipulation des
            Control/Status-Registers
    or  accept  DisableInterrupt;
        -- Unterbrechungs-Unterdrückung durch Manipulation des
Control/Status-Registers
    end select;
  end loop;
end Parallel;

task body  Handler is
begin
  loop
    accept Interrupt;    -- Warten auf eine Unterbrechung
    -- Implementierung der Reaktion auf eine Unterbrechung.
    accept  Benachrichtigung;
  end  loop;
end Handler;

end Parallelpaket;
```

Mit diesen Implementierungen kann der Benutzerprozeß CONTROL durch die Aufrufe

Parallelpaket.Parallel.EnableInterrupt;
Parallelpaket.Handler.Benachrichtigung;
Parallelpaket.Parallel.DisableInterrupt;

auf das Eintreffen einer Unterbrechung warten. Die Implementierung der Task HANDLER berücksichtigt nicht das Eintreffen unerwarteter Unterbrechungen!

Anhang D Literatur

Heiß, H.

„Die explizite Lösung der kinematischen Gleichung für eine Klasse von Industrierobotern”, Dissertation, Technische Universität Berlin, 1985.

Hommel, G., Knoll, A., Krüger, A., Schweikard, A.

„Bedienhandbuch zum REMROC-System”, Skript zur Lehrveranstaltung Robotik 1, WS 87/88, TU Berlin, Fachbereich Informatik, Institut für Technische Infomatik, 1987.

Hommel, G.

Skript zur Lehrveranstaltung Prozeßdatenverarbeitung und Robotik 1, WS 91/92, TU Berlin, Fachbereich Informatik, Institut für Technische Informatik,

Krüger, A.

„Entwurf und Implementierung eines ausgelagerten Roboterprogrammiersystems”, Diplomarbeit, TU Berlin, Fachbereich Informatik, Institut für Technische Infomatik, 1988.

Paul, R.P.

"Robot Manipulators: Mathematics, Programming and Control", Cambridge (Mass.) and London: MIT Press, 1981.

Schweikard, A., Hommel, G.

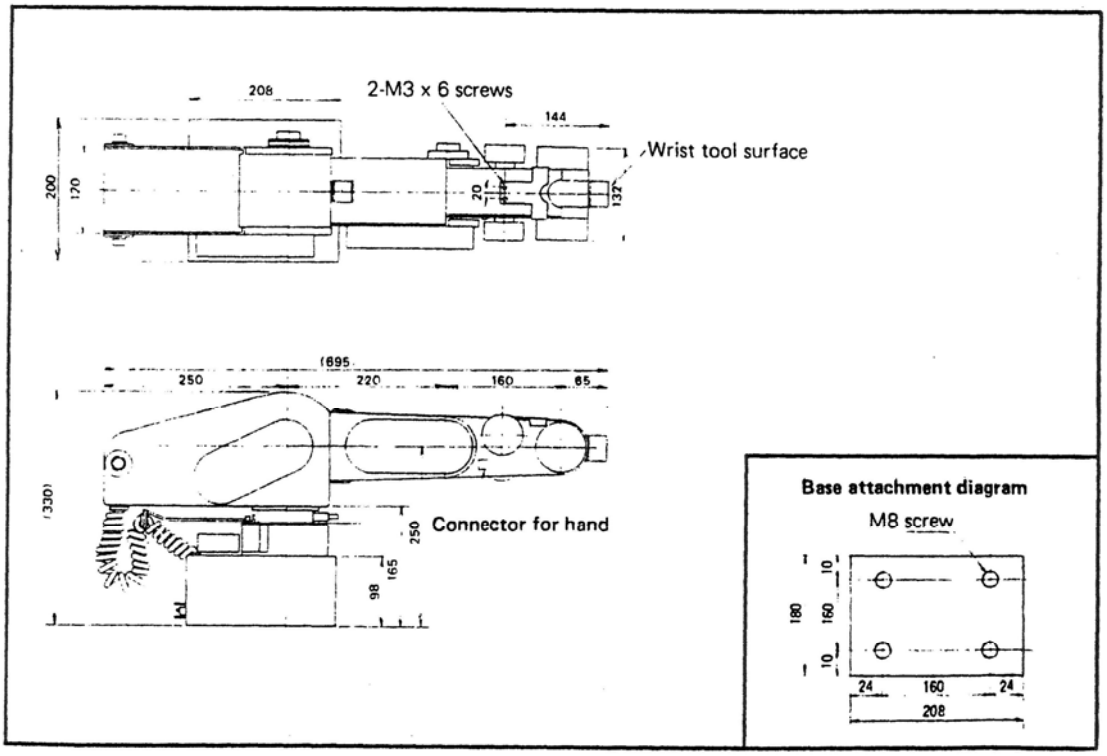
„Berechnung erreichbarer Effektorstellungen für Handhabungsgeräte mit weniger als sechs Freiheitsgraden”. Robotersysteme **4**, S. 177 ... 182 Springer-Verlag: Berlin, Heidelberg, New York, 1988.

Anhang E Beschreibung des RM 501 und RV-M1

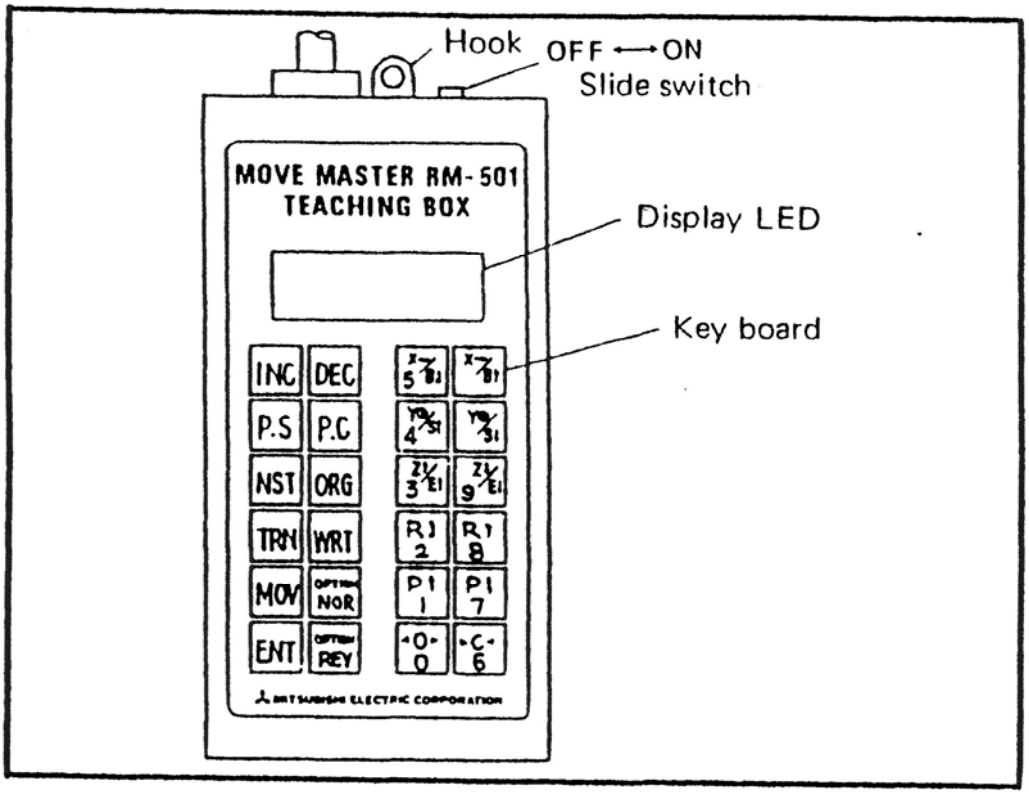
Item		Specification
Structure		Five degrees of freedom Vertical multi-joint type
Range of movement	Waist rotation	300°
	Shoulder rotation	130°
	Elbow rotation	90°
	Wrist pitch	±90°
	Wrist roll	±180°
Permissible handling weight		max. 1.2 kg (includes weight of hand)
Maximum synthesis speed		400 mm/sec (wrist tool surface)
Position repeat accuracy		±0.5 mm (wrist tool surface)
Drive system		Electroservo drive by a DC servomotor
Main unit weight		about 27 kg

Note: The permissible handling weight (1.2 kg) is the value at a point 100 mm from the wrist tool surface.

Outer dimension diagram



Anhang E Beschreibung des RM 501 und RV-M1



Outer appearance of the teaching box

Anhang E Beschreibung des RM 501 und RV-M1

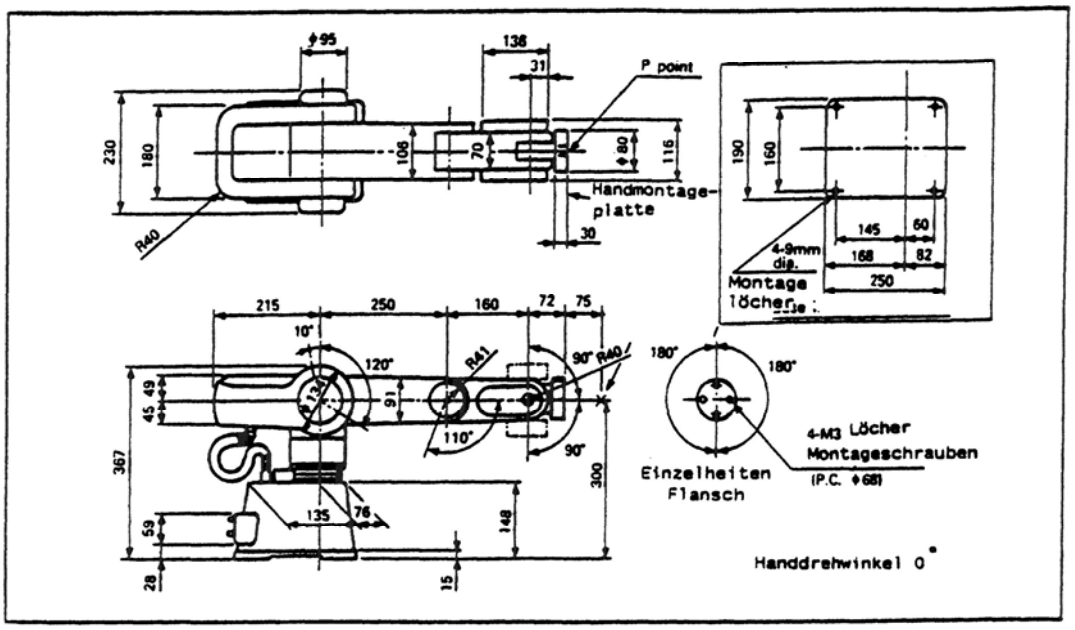
Input key	Function
B - 5	When pushed, the body moves in the clockwise direction.
B ·	When pushed, the body moves in the counterclockwise direction.
S ↑ 4	When pushed, the shoulder moves upwards.
S ↓	When pushed, the shoulder moves downwards.
E · 3	When pushed, the elbow moves upwards.
E ; 9	When pushed, the elbow moved downwards.
R . 2	When pushed, the wrist is turned in the clockwise direction.
R · 8	When pushed, the wrist is turned in the counterclockwise direction.
P · 1	When pushed, the wrist moves upwards.
P ; 7	When pushed, the wrist moves downwards.
◂0▸ 0	When pushed, the hand opens.
▸C◂ 6	When pushed, the hand is closed.
INC	Used to continuously confirm the No. of recorded position after the position No. input into the teaching box.
DEC	Used to continuously confirm the No. of the recorded position before the position No. input into the teaching box.
P.S	Used when the position is decided by teaching and the position is recorded as a No.
P.C	Used when the recording of a portion already recorded is erased.
NST	Used for origin output of the mechanical system.
ORG	Used to return to the origin by control system operation.
TRN	Used for transfer of a ROM converted program to the RAM in the drive unit.
WRT	Used for read-in of the prepared program into the ROM (ROM conversion)
MOV	Used for movement to a recorded position.
ENT	Execution key
OPTION REV	Not used.
OPTION NOR	

Anhang E Beschreibung des RM 501 und RV-M1

Leistungsmerkmale		Spezifikation	Bemerkung
Mechanik		5 Freiheitsgrade	
Bewegungs-raum	Mittelstück (Drehung)	300° (max. 120°/s)	J1 Achse
	Schulter (Drehung)	130° (max. 72°/s)	J2 Achse
	Ellenbogen (Drehung)	110° (max. 109°/s)	J3 Achse
	Handgelenk-Neigungswinkel	± 90° (max. 100°/s)	J4 Achse
	Handdrehwinkel	± 180° (max. 163°/s)	J5 Achse
Armlänge	Oberarm	250 mm	
	Unterarm	160 mm	
Zulässiges Gewicht des zu bewegenden Teils		Max. 1.2 kg (einschließlich der Hand)	Schwerpunkt 75 mm von der Montageplatte
Maximale Bewegungsgeschwindigkeit		1000 mm/s (bezogen auf die Handmontageplatte)	Geschwindigkeit in Punkt P in Abb. 1.3.4
Wiederholgenauigkeit der Positionierung		< 0.3 mm (bezogen auf die Mitte der Handmontageplatte)	Genauigkeit im Punkt P in Abb. 1.3.4
Antrieb		Elektr. Antrieb mit Gleichstrom Servomotoren	
Gewicht des Roboters		ca. 19 kg	
Motorleistung		J1 bis J3 Achsen: 30W J4 + J5 Achse: 11W	

Standardspezifikation

Anhang E Beschreibung des RM 501 und RV-M1



Abmessungen