Reengineering des originalen Maschinenprogramms

Betreuer:

Prof. Dr. Günter Kemnitz
Dipl.-Ing Hossam Addeen Ramadan

Bearbeiter:

Zhao, Dalong 326724

Zhang, Wenyong 333852

12.04.2010 in Clausthal-Zellerfeld

Inhaltverzeichnis

1. Zielstellung:

Die übergeordnete Zielstellung der Projektarbeit war:

Die RoboBasic-Software kann nur unter Windows-XP-Betriebssystem bearbeitet werden. Um das RoboBasic-Programm in allen Betriebssystemen bearbeitet werden zu können, wird eine neue Umgebung in C-Sprache entwickelt. Durch diese Umgebung kann das RoboBasic-Programm nur im Editor beschrieben wird, und allen Befehlen darin können direkt in den Hexcoden, die aus dem Mikrocontroller ausgelesen werden sollen, gewechselt werden.

2. Dokumentation:

a. C

   Herbert Schildt

   C The Complete Reference

   Fourth Edition

b. XML [http://en.wikipedia.org/wiki/XML].

c. Handbuch für Befehlsreferenz RoboBasic.

d. Installationsanleitung der RoboBasic-Software .

e. Compiler und Interpreter [http://web.cs.wpi.edu/~gpollice/cs544-f05/CourseNotes/maps/Class1/Compilervs.Interpreter.html]

3. Grundlagen der Arbeit:

a. Robonova-I

Robonova-I von HiTEC ist eine Art von Roboter mit 16 Gelenken, davon 10 als Beingelenke und als Armgelenke. Er kann durch Infrarot-Fernbedienung gesteuert werden. Die Steuerung wird durch RoboBASIC, eine spezielle BASIC-Programmiersprache implementiert.

b. Entwicklungsumgebung (Robobasic)

RoboBASIC ist eine Programmierumgebung mit spezieller Programmiersprache zur Steuerung von Robotern, deren grundlegende Grammatik auf der Basis von BASIC ist. RoboBASIC präsentiert eine Erweiterung der allgemeinen Grundprogrammiersprache mit speziellen Befehlen zur Steuerung von Robotern.

Diese Umgebung funktioniert nur unter Windows XP Betriebssystem.

c. Zusammenfassung des Befehls für RoboBasic:

Befehle, die der Deklaration/Definition dienen:

DIM:	Variable deklarieren
AS:	Variable bei der Deklaration als Variable definieren
CONST:	Konstante deklarieren
BYTE:	Variable bei der Deklaration als Byte definieren
INTEGER:	Variable bei der Deklaration als Integer definieren

Ablaufsteuerbefehle:

IF:	Beginn einer bedingten Anweisung
THEN:	Nächste Anweisung ausführen, wenn die Bedingung wahr ist
ELSE:	Nächste Anweisung ausführen, wenn die Bedingung falsch ist
ELSEIF:	Beginn einer anderen bedingten Anweisung
ENDIF:	Ende der bedingten Anweisung
FOR:	Beginn einer Wiederholungsanweisung
TO:	Zuweisung des Wiederholungsbereichs einer Wiederholungsanweisung
NEXT:	Ende einer Wiederholungsanweisung
GOTO:	Teilung des Programmablaufs
GOSUB:	Aufruf einer untergeordneten Routine

RETURN: Rückkehr zum Programm aus der untergeordneten Routine
END: Ausführung des Programms beenden
STOP: Ausführung des Programms stoppen
RUN: Programm fortlaufend ausführen
WAIT: Warten, bis das Programm vollständig ausgeführt wurde
DELAY: Programmausführung um einen gewählten Zeitraum verzögern
BREAK: Programmausführung pausieren und in den Fehlerbehebungsmodus wechseln

Digitale Signaleingabe und –ausgabebefehle:
IN: Signal vom Eingangsparametersport lesen
OUT: Signal zum Ausgangsport senden
BYTEIN: Byte-Signal vom Eingangsparametersport der Byte-Einheit lesen
BYTEOUT: Byte-Signal an den Ausgangsport der Byte-Einheit lesen
INKEY: Eingehender Schlüssel vom Eingangsparametersport
STATE: Status des Ausgangsport
PULSE: Impuls-Signal an den Ausgangsport senden
TOGGLE: Status des Ausgangsports zurücksetzen
KEYIN: Analoge Tastenblockeingabe empfangen

Befehle für den Speicher:
PEEK: Daten vom Controller-Arbeitsspeicher lesen
POKE: Daten in den Controller-Arbeitsspeicher schreiben
ROMPEEK: Daten vom externen EEPROM-RAM des Controllers lesen
ROMPOKE: Daten in den externen EEPROM-RAM des Controllers lesen

Befehle für das LCD:
LCDINIT: Initialisieren des LCD-Moduls
CLS: Alle Zeichen im LCD-Modul löschen
LOCATE: Zeichenplatzierung im LCD-Modul bestimmen
PRINT: Buchstaben in LCD-Modul anzeigen
FORMAT: Typ-Format welches auf dem LCD-Modul angezeigt wird einstellen
CSON: Cursor auf dem LCD-Modul anzeigen
CSOFF: Cursor auf dem LCD-Modul verbergen
CONT: Buchstaben-Kontrast auf dem LCD-Modul einstellen
DEC: Dezimale Numerale auf dem LCD ausgeben
HEX: Hexadezimale Numerale auf dem LCD ausgeben
BIN: Binäre Numerale auf dem LCD ausgeben

Auf den Operand bezogene Operationen:
AND: Verwendung des logischen Ausdrucks „und“
OR: Verwendung des logischen Ausdrucks „oder“
MOD: Kalkulationsmodul für arithmetische Operationen

XOR: Verwendung des logischen Ausdrucks „XOR“
NOT: Alle Bits zurücksetzen

Befehle zur Motorsteuerung:
ZERO: Einstellen des 0 Punktes (Standartwinkel) eines Stellmotors
MOTOR: Einschalten des Ausgangsports vom Stellmotor
MOTOROFF: Ausschalten des Ausgangsports vom Stellmotor
MOVE: Steuerung mehrerer Motoren zum selber Zeitpunkt
SPEED: Einstellen der Geschwindigkeit des Stellmotors
ACCEL: Einstellen der Beschleunigung des Stellmotors
DIR: Einstellen der Drehrichtung des Stellmotors
PTP: simultane Operationssteuerung Ein/Aus
SERVO: Steuerung des Stellmotors
PWM: Einstellen der Pulslänge für einen Gleichstrommotor
FASTSERVO: Servomotor mit maximaler Geschwindigkeit betreiben
HIGHSPEED: „Fast-Mode“ des Stellmotors Ein/Aus
MOVEPOS: Motor-Gruppe laut Deklaration von POS bewegen
POS: Einstellen der spezifischen Position des Roboters
FPWM: Ändern der Pulslänge und Frequenz
MOVE24: Alle 24 Stellmotoren zur gleichen Zeit bewegen
INIT: Einstellen der Anfangsbewegungshaltung
MOTORIN: Auslesen der aktuellen Positionswerte des Stellmotors
AIMOTOR: Konfiguration für das Benutzen des AI-Motors
AIMOTORIFF: Abbrechen der Nutzung des AI-Motors
AIMOTORIN: Auslesen der aktuellen Positionswerte des AI-Motors
SETON: Konfiguration zur Nutzung der Konfigurations-Funktion
SETOFF: Abbruch der Konfiguration zur Nutzung der Konfigurations-Funktion
ALLON: Konfigurations-Funktion für alle Stellmotoren
ALLOFF: Abbruch der Konfigurations-Funktion für alle Stellmotoren
GETMOTORSET: Auslesen der aktuellen Positionswerte des Stellmotors und beibehalten der aktuellen Position

Parameter, welche die Motorgruppe zuordnen:
G6A: Servomotoren #0-#5 zu Gruppe A zuordnen
G6B: Servomotoren #6-#11 zu Gruppe B zuordnen
G6C: Servomotoren #12-#17 zu Gruppe C zuordnen
G6D: Servomotoren #18-#23 zu Gruppe D zuordnen
G6E: Servomotoren #24-#29 zu Gruppe E zuordnen
G8A: Servomotoren #0-#7 zu Gruppe A zuordnen
G8B: Servomotoren #8-#15 zu Gruppe B zuordnen
G8C: Servomotoren #16-#23 zu Gruppe C zuordnen
G8D: Servomotoren #24-#31 zu Gruppe D zuordnen
G12: Servomotoren #0-#11 zuordnen
G16: Servomotoren #0-#15 zuordnen

G24: Servomotoren #0-#23 zuordnen
G32: Servomotoren #0-#31 zuordnen

Befehle zur Klangkontrolle:
BEEP: Warnungsgeräusch mit PIEZO erzeugen
SOUND: Frequentierten Sound mit PIEZO erzeugen
PLAY: Ein Lied mit PIEZO abspielen
MUSIK: Musik mit PIEZO abspielen
TEMPO: Rhythmus des Sounds einstellen

Befehle zur externen Kommunikation:
RX: RS-232-Signal durch RX-Port empfangen
TX: RS-232-Signal durch TX-Port senden
MINIIN: Minibus-Signal durch den Mini-Kommunikationsport empfangen
MINIOUT: Minibus-Signal durch den Mini-Kommunikationsport übertragen
ERX: RS-232-Signal durch RX-Port empfangen
ETX: RS-232-Signal durch TX-Port senden

Befehle zur analogen Signalverarbeitung:
AD: Analoges Signal vom AD-Port empfangen
REMOCON: Schlüsselwert vom Infrarot-Controller empfangen
SONAR: Distanz vom Ultraschallwellen-Port empfangen
RCIN: Eingabesignal vom RC-Remote-Controller empfangen
GYRODIR: Konfiguration der Richtung des Gyroskops
GYROSET: Gyroskop einem Servomotor zuordnen
GYROSENSE: Konfiguration der Empfindlichkeit des Gyroskops

Verarbeitungsbefehle:
ON…GOTO: Überspringen bei bestimmtem Wert der Variable

Sonstige Befehle:
RND: Zufallszahl generieren
REMARK: Erzeugen eines Eintrages in Textform

Absichtsbefehle:
$DEVICE: Konfiguration des Controllers zur Nutzung von einem derzeitig laufendem Programm
LIMIT: Beschränken des Bewegungsbereichs eines Stellmotors
[google: Befehl RoboBasicROBOBASIC Befehls-Bedienungsanleitung v2.10 Inhaltsverzeichnis]

d. Interpreter und Compiler

a. Interpreter

Ein Interpreter ist ein Programm, das die Quellecodes von höherer Programmiersprache einliest, analysiert und interpretiert, so dass sie auf allen Computerplattformen laufen kann.

Einzelne Befehle des Quellcodes werden von Interpreter übersetzt und direkt ausgeführt, deshalb müssen einzelne Teile des Programms immer erneut übersetzt werden. Ein Interpreter erzeugt keinen abspeicherbaren Maschinencode, der zu längere Durchführungszeit führen kann.

b. Compiler

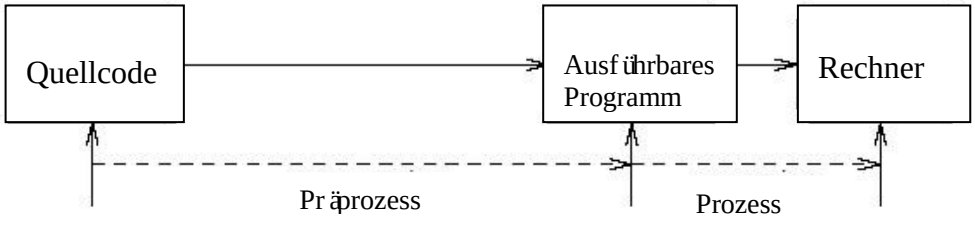
Ein Compiler ist ein Programm, das der Quellcode einer Programmiersprache in entsprechendem Maschinencode übersetzt, der für die Rechner unter bestimmten Betriebssystemen ausführen kann.
Normalerweise funktioniert ein Compiler mit folgendem Arbeitsprozess:

Quellcode -> Präprozessor -> Compiler-> Assembler-> Objektcode-> Programmbinder-> Ausführbares Programm

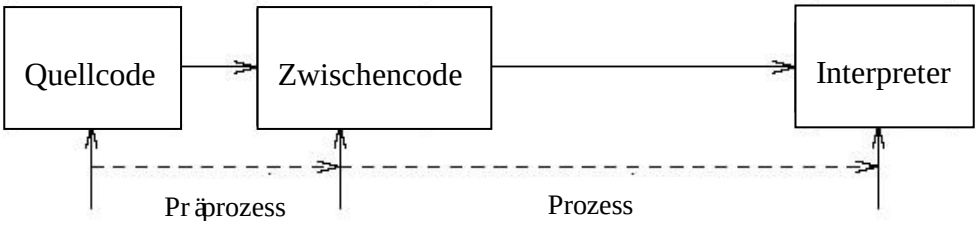
Präprozessor: Ein unabhängiges Programm, das vor der Übersetzung von dem Compiler aufgerufen werden kann. Präprozessor kann die Kommentare löschen, andere Dateien beinhalten und Makro(eine Abstraktion zum Ersetzen von bestimmtem Textmodus) ausführen.
Assembler: Ein Programm, das die Übersetzung eines Computerprogramms(in maschinennaher Assemblersprache geschrieben) in Maschinencode helfen kann.
Objektcode: Binärcode, der von Quellcode durch Compiler und Assembler erzeugt und direkt von CPU erkennt wird.
Programmbinder: Ein Programm, durch das mehrere Objektcodes, die von Compiler oder Assembler erzeugt werden, zu einem ausführbaren Programm verbinden.
Ausführbares Programm: Ein Computerprogramm, das unter bestimmtem Betriebssystem direkt ausgeführt werden kann.

c. Interpreter versus Compiler

Ein Interpreter stellt eine Übersetzung in einigen Formen von Quellcode dar, damit es sofort ausführen und bewerten kann. Die Struktur des Interpreters ist ähnlich wie bei einem Compiler. Die folgende Abbildung zeigt eine Darstellung der Unterschiede.



Compiler



Interpreter

Eigenschaften für Compiler:

- kostet viel Zeit für die Analyse und Bearbeitung eines Programms
- das resultierende ausführbare Programm ist binärer Code
- die Computerhardware interpretiert der resultierende Code
- Programmablauf ist schnell

Eigenschaft für Interpreter:

- kostet relativ wenige Zeit mit der Analyse und Bearbeitung eines Programms
- Der resultierende Code ist eine Art von Zwischencode
- Der resultierende Code ist von einem anderen Programm interpretiert
- Programmablauf ist langsam

4. Arbeitsreihenfolge:

Konkret verfolgt diese Arbeit in die wesentlichen Schritte:

1) Objektcode von Robobasic Programm erkennen

Ein Befehl von RoboBASIC wird zuerst von der Programmierumgebung eingelesen und bearbeitet. Nach der Bearbeitung erzeugt die Umgebung einen entsprechenden Objektcode. Dieser Code wird als eine OBJ-Datei gespeichert, die aus Hexadezimalzahl besteht.
Um der Inhalt der OBJ-Datei gelesen wird zu können, wird die generierte OBJ-Datei von einem Hexeditor eingelesen. Mit Hilfe von Hexeditor wird der konkrete Inhalt als Hexadezimalzahl in einem Fenster dargestellt. Dort sind entsprechende Hexcodes und Kodierungsregeln für bestimmte RoboBASIC-Befehle herauszufinden.

1.1 Standardformat, Anfangs- und Endecodeblock

Manche zusätzlichen Bytecodes werden von RoboBASIC an einigen Stellen verwendet, um einen Bitwert oder 8 Bits ein Byte zu speichern. Viele unterschiedliche Befehle können durch die gleiche Logik mit einem Standardformat bearbeitet werden.
Wenn wir sognannte Flags für die Parameter nach einem Befehl kodieren möchten, die entsprechende Codes der verschiedene Parametertypen der Flags sind wie folgendes:

Typ	Flag
Bytewerte	12
Ganzzahlwerte	13
Bytevariablen	15
Ganzzahlvariablen	16

Danach folgt die Speicheradresse oder Wert. Normalerweise wird die Speicheradresse der ersten Variable mit dem Code 40 kodiert, dann der zweiten 41 usw. Die entsprechenden Codes sind wie folgendes:

Typ	Kodierung
Bytewerte	Wert, 1 Byte
Ganzzahlwerte	LOW- und HIGH-Anteil des Wertes, 2 Bytes
Variablen	Speicheradresse der Variablen, 1 Byte

Die Zahlen 0 und 1 werden speziell behandelt:

Wert	Kodierung
0	10
1	11

Der Anfangscodeblock hat eine spezielle Funktionalität. Viele weitere Informationen der Quellcodedatei werden dadurch beschrieben.

Beschreibung des Anfangscodeblocks in der Programmübertragung:

Byte	Inhalt
k(1<=k<=111, hängt die Anzahl der Variablen, k=0 falls keine Variable vorhanden)	Speicherplatzreservierung für benutzerdefinierte Variablen, die später auftauchen können.
K+1 – k+8	Dateiname des Quelltextes
K+9 – k+11	Datum in der Form Jahr-Monat-Tag
K+12 – k+14	Uhrzeit in der Form Stunden-Minuten Sekunden
K+15- k+16	LOW- und HIGH-Anteil der Länge des Quelltextes (in Bytecodes)

Der Endecodeblock hat ähnliche Funktionalität wie der Anfangscodeblock und beschreibt einige Zusatzinformationen wie folgendes:

Byte	Inhalt
1	C4
2	Anzahl der Codes, der von RoboBasic-Befehle erzeugt hat. Wenn keinen Code erzeugt, dann ist der Bytewert 10. Die Anzahl 1 enspricht Wert 11, usw.
3	00

1.2 Kodierung der Gruppenzahlen

Der Gruppencode zeigt, welcher Servomotor durch den Mikrokontroller gesteuert wird.

Alle verfügbaren Gruppencodes:

Gruppencode	**Servomotoren**	**Kodierung**
G6A	0-5	00 06
G6B	6-11	06 06
G6C	12-17	0C 06
G6D	18-23	12 06
G6E	24-29	18 06
G8A	0-7	00 08
G8B	8-15	08 08
G8C	16-23	10 08
G8D	24-31	18 08
G12	0-11	00 0C
G16	0-15	00 10
G24	0-23	00 18
G32	0-31	00 20

1.3 Kodierung aller Befehle

1.3.1 GYROSET

Der GYROSET-Befehl entscheidet, welche Servomotor in einer Motorgruppe von einem bestimmten Kreisel gesteuert werden soll. [Motor N Gyro] ist die Portnummer, die für jeden Servomotor in der Gruppe verwendet wird.

Der entsprechende Hexcode ist A0

Befehlsstruktur:
GYROSET [Gruppe], [Motor N Gyo]

Beispiel: GYROSET G6A,1,2,1,0,2,0

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 1C
00 47 59 52 4F 53 45 54 2E 0A 02 18 0F 20 1B 1C
00 A0 00 06 01 02 01 00 02 00 C4 19 00
```

1.3.2 GYRODIR

Der GYRODIR-Befehl entscheidet die Richtung einer Servomotorgruppe, wenn ein Kreisel mit der AD Port verbunden ist. Es gibt 4 Nummer von den Kreiseln zur Verfügung stehen. [Motor Direction] ist entweder „0" oder „1". „1" steht für eine zunehmende Servoposition und „0" für eine abnehmende Servoposition.

Der entsprechende Hexcode ist A1

Befehlsstruktur:
GYRODIR [Grupp], [Motorrichtung] …

Beispiel: GYRODIR G6B,1,0,0,1,1,1

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 1C
00 47 59 52 4F 44 49 52 2E 0A 02 18 0F 23 0C 1C
00 A1 06 06 01 00 00 01 01 01 C4 19 00
```

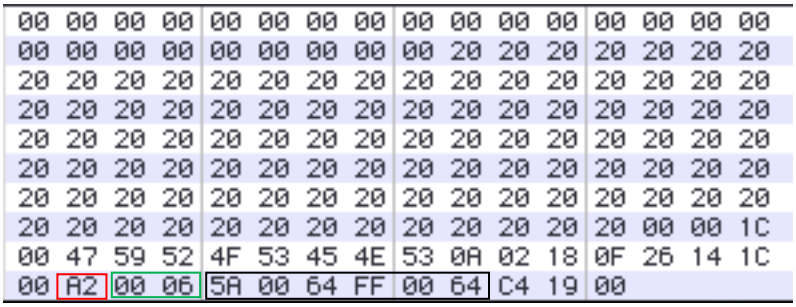
1.3.3 GYROSENSE

Der GYROSENSE-Befehl legt fest, welche Empfindlichkeit des einzelnen Servomotors ist. [Motor N Gyro Sensitivity] steuert die Empfindlichkeit für jeden Servomotor in einer Gruppe durch Zahlen von 0 bis zu 255 oder Konstante. Wenn hier „0“ eingesetzt wird, bedeutet dass die Empfindlichkeit eines Motors nicht geändert wird.

Der entsprechende Hexcode ist A2

Befehlsstruktur:
GYROSENSE [Grupp], [Motor N Gyro Empfindlichkeit] …

Beispiel: GYROSENSE G6A, 90,0,100,255,0,100



1.3.4 MOVE

Der MOVE-Befehl weist an, dass mehrere Servomotoren gleichzeitig zu einem bestimmten Winkeln bewegen sollen. Die Parameter hinter MOVE müssen im Bereich von 10 bis 190 sein. Leere Parameter hinter MOVE sind auch erlaubt. Das bedeutet, dass ein Servomotor nicht bewegt werden soll.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	B0
Byte 2 bis 3	Gruppenbytecode(siehe Kapitel 2)
Byte 4 bis zum Ende des Statements	Jedes Byte zeigt einen Winkel des entsprechenden Servomotors. (Der unbewegten Servomotor wird mit 00 kodiert.)

Oder

Byte 1	B0
Byte 2 bis zum Ende des Statements	Jedes Byte zeigt einen Winkel des entsprechenden Servomotors. (Der

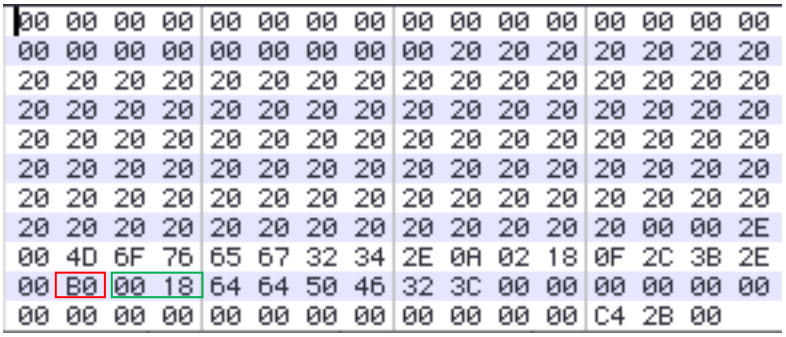
	unbewegten Servomotor wird mit 00 kodiert.)

Befehlsstruktur:

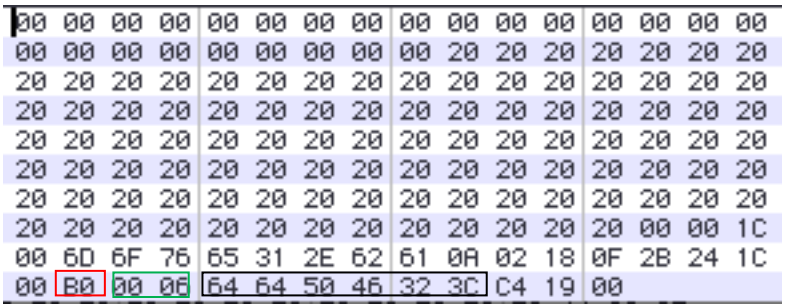
1. MOVE [Gruppencode],[Winkel von ersten Motor],...,[Winkel von letzten Motor]
2. MOVE [Winkel von ersten Motor], … ,[Winkel von sechsten Motor]

Beispiel: 1. MOVE24

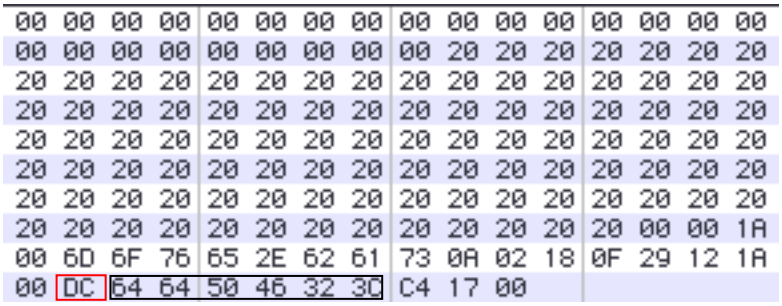
2. MOVE G24



3. MOVE G6A,100,100,80,70,50,60



4. MOVE 100,100,80,70,50,60



1.3.5 POS

Der POS-Befehl ist die Ergänzung zu anderen Befehlen, um die Position des Roboters festzulegen.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	B1
Byte 2 bis 3	Gruppenbytecode
Byte 4 bis zum Ende des Statements	Jedes Byte zeigt einen Winkel von 10 Grad bis 190 Grad des entsprechenden Servomotors.

Befehlsstruktur:
POS [Gruppencode],[Winkel von ersten Motor],...,[Winkel von letzten Motor]

1.3.6 ZERO

Wegen der Fertigungstoleranz der Motoren sind Nullpunkte der Motoren unterschiedlich voneinander, die im Bereich 90 bis 100 sein können. Durch den ZERO-Befehl können diese Fehler vermeiden werden, damit der Nullpunkt eines Servomotors festgelegt werden kann, der als der Standardpunkt für den MOVE-Befehl betrachtet wird.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	D9
Byte 2 bis 3	Gruppenbytecode
Byte 4	Jedes Byte zeigt einen Nullpunkt des jeweiligen Servomotors an.

Oder

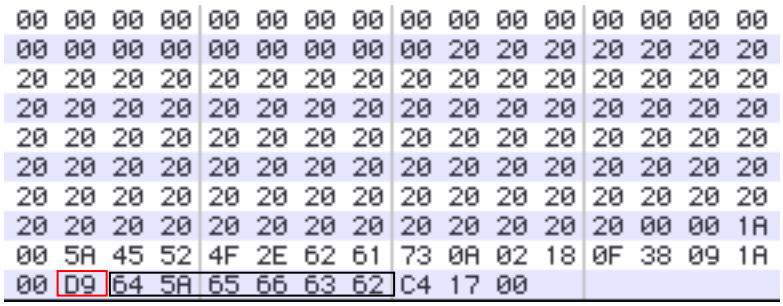
Byte 1	D9
Byte 2	Jedes Byte zeigt einen Nullpunkt

	des jeweiligen Servomotors an.

Befehlsstruktur:

1. ZERO [Gruppencode],[Nullpunkt von erstem Motor], … ,[Nullpunkt von letztem Motor]
2. ZERO [Nullpunkt von Motor 0], … ,[Nullpunkt von Motor 5]

Beispiel: ZERO 100,90,101,102,99,98



1.3.7 DIR

Die Drehrichtung eines Servomotors kann mit DIR-Befehl eingestellt werden. Für eine Rotation nach links wird die Parameter 0 oder leer benutzt, und für eine Rotation nach rechts steht die Parameter 1. Diesen Parametern bauen wie ein String auf, das wird als einer Binärzahl verstehen und besetzt ein Byte. Ein Byte besteht aus 8 Bits, deswegen darf die Parametergruppe nur bis 8 Motoren enthalten.

Ein Beispiel:

Parameter	Binärzahl	Dez	Hex
1,1,0,0,1,0	110010	50	32

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	B4
Byte 2 bis 3	Gruppenbytecode
Byte 4	Hexadezimale Darstellung für die Drehrichtungen in einer Servomotorgruppe

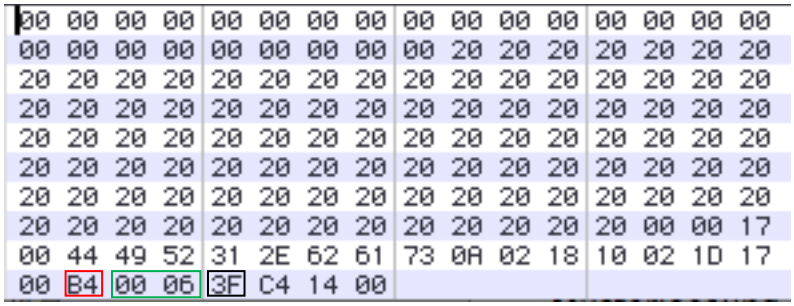
Oder

Byte 1	B4
Byte 2	Hexadezimale Darstellung für die Drehrichtungen in einer Servomotorgruppe

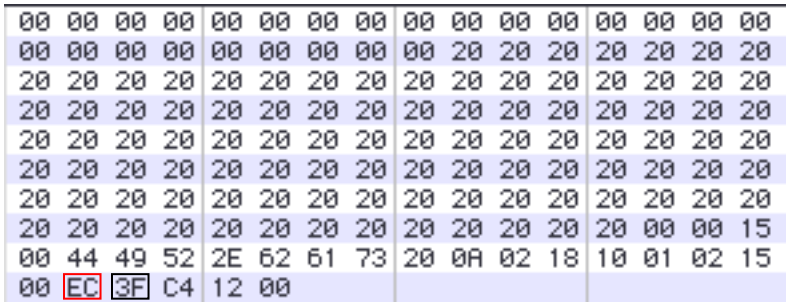
Befehlsstruktur:

1. DIR [Gruppencode],[Winkel von ersten Motor],...,[Winkel von letzten Motor]
2. DIR [Winkel von ersten Motor], … ,[Winkel von letzten Motor]

Beispiel: a. DIR G6A,1,1,1,1,1,1



b. DIR 1,1,1,1,1,1



1.3.8 INIT

Der Befehl INIT gibt einen Initialen Winkel für die Servomotoren einer Gruppe vor. 100 wird als eine Grundpostion der Motoren verwendet.

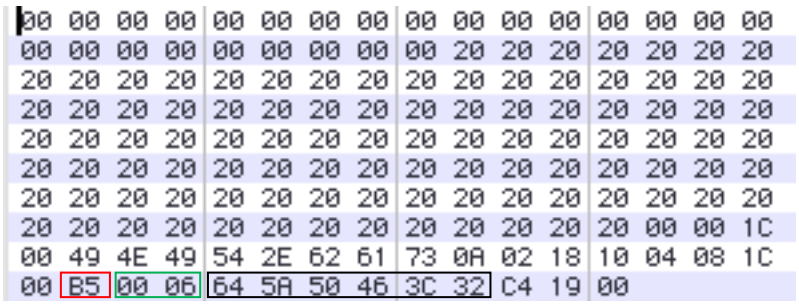
Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	B5
Byte 2 bis 3	Gruppenbytecode
Byte 4 bis zum Ende des Statements	Jedes Byte zeigt einen Winkel des entsprechenden Servomotors. (Der unbewegten Servomotor wird mit 00 kodiert.)

Befehlsstruktur:

INIT [Gruppencode],[Winkel von ersten Motor],...,[Winkel von letzten Motor]

Beispiel: INIT G6A,100,90,80,70,60,50



1.3.9 HIGHSPEED

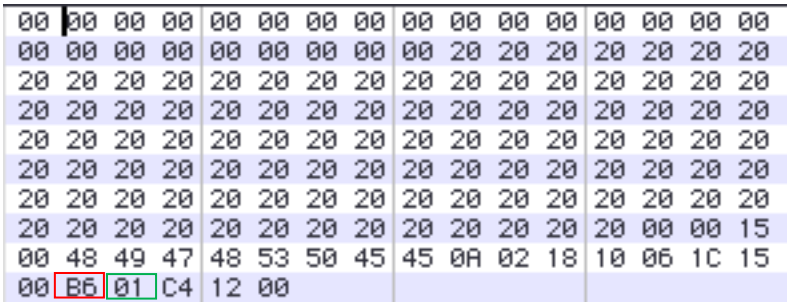
Die Geschwindigkeit des Roboters wird durch den HIGHSPEED-Befehl dreimal erhöht.

Der entsprechende Hexcode besteht aus folgenden Bytes:

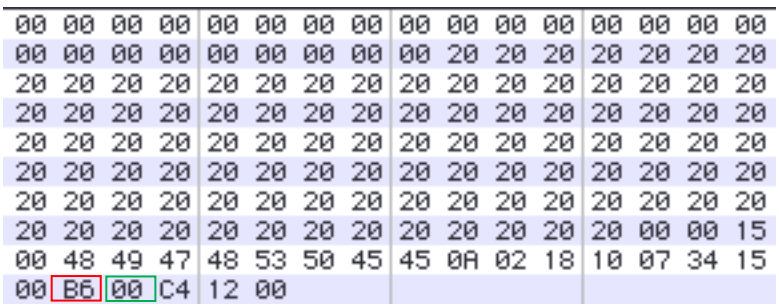
Byte 1	B6
Byte 2(Wenn SETOFF)	00
Byte 2(Wenn SETON)	01

Befehlsstruktur:
HIGHSPEED [SETON | SETOFF]

Beispiel: a. HIGHSPEED SETON



b. HIGHSPEED SETOFF



1.3.10 AIMOTOR

Der AI Motor ist ein Motor mit einer Mikrosteuerchip, die für die Kommunikation mit Mikrokontroller durch RS232 zuständig ist. Der AI Motor steuert Drehungsgrad des Motors und zeigt dem Zustand des Motors. AI Motors können mit Port Nr. 0 bis zu Nr.30 verbunden werden.
[Motor Nr.] stellt einen bestimmten Motor ein. [bestellte Grupp] stellt mehrere Motoren als eine Gruppe ein.
Die Ausführung von diesem Schlüsselwort ist ähnlich wie das Schlüsselwort „MOTOR“. Zahlen, Konstante oder Variable mit dem Typ Byte sind mögliche Alternative für das Parameter [Motor Nr.].

Der entsprechende Hexcode besteht aus folgenden Bytes:

AIMOTOR SETON	B6 03
AIMOTOR SETOFF	B6 02
AIMOTOR INIT	B6 04
AIMOTOR Motornummer	B7 [Bytecode von Bytevariable] [Hexcode von Motornummer]*
AIMOTOR Gruppencode	B7 [Gruppencode]*

Befehlsstruktur:
AIMOTOR [SETON | SETOFF | INIT | Motornummer | Gruppencode]

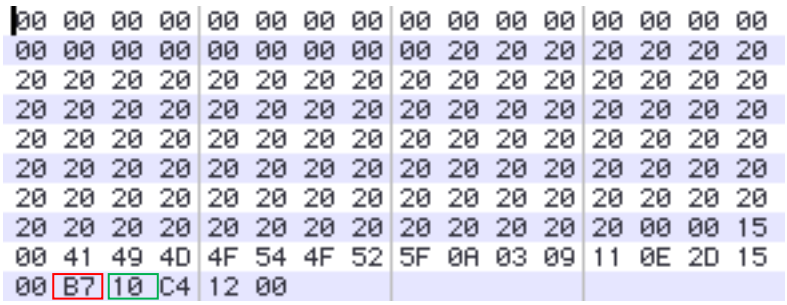
Beispiel: a. AIMOTOR SETON

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 15
00 41 49 2D 53 45 54 4F 4E 0A 03 09 0F 04 2F 15
00 B6 03 C4 12 00
```

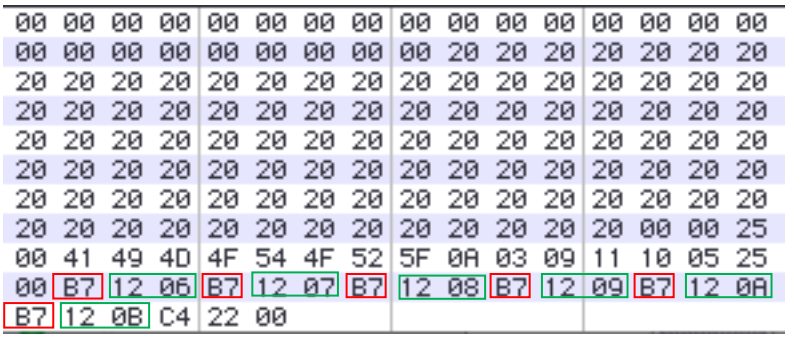
b. AIMOTOR SETOFF

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 15
00 41 49 2D 53 45 54 4F 46 0A 03 09 0F 06 35 15
00 B6 02 C4 12 00
```

c. AIMOTOR 0



d. AIMOTOR G6B



1.3.12 AIMOTOROFF

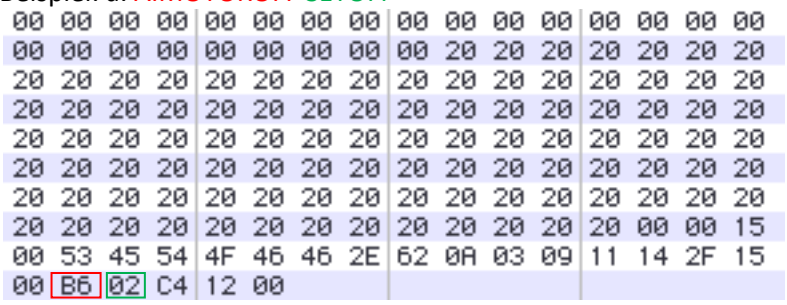
Durch den AIMOTOROFF-Befehl beendet die Durchführung von dem Befehl AIMOTOR.

Der entsprechende Hexcode besteht aus folgenden Bytes:

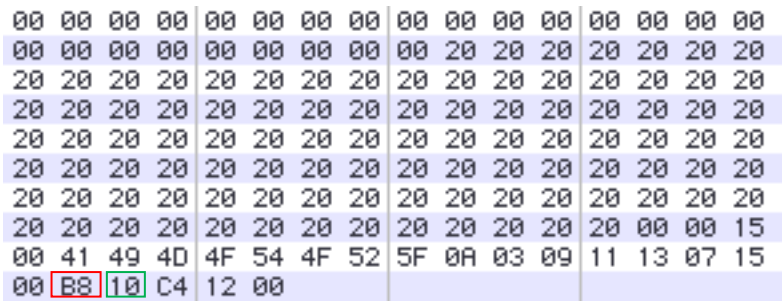
AIMOTOROFF Motornummer	B8 [Bytecode von Bytevariable] [Hexcode von Motornummer]*
AIMOTOROFF Gruppencode	B8 [Gruppencode]*
AIMOTOROFF SETOFF	B6 02

Befehlsstruktur:
AIMOTOROFF [Motornummer | Gruppencode]

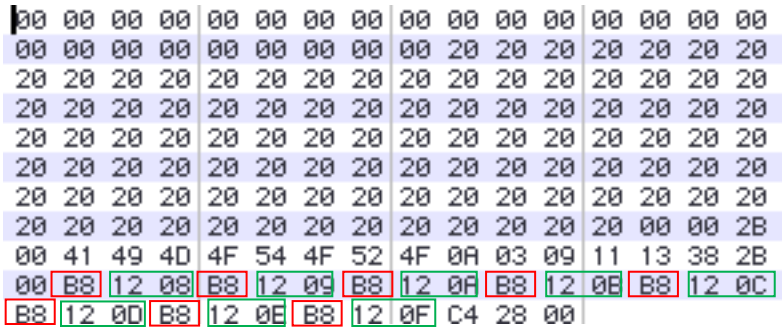
Beispiel: a. AIMOTOROFF SETOFF



b. AIMOTOROFF 0



c. AIMOTOROFF G8B



1.3.13 GETMOTORSET

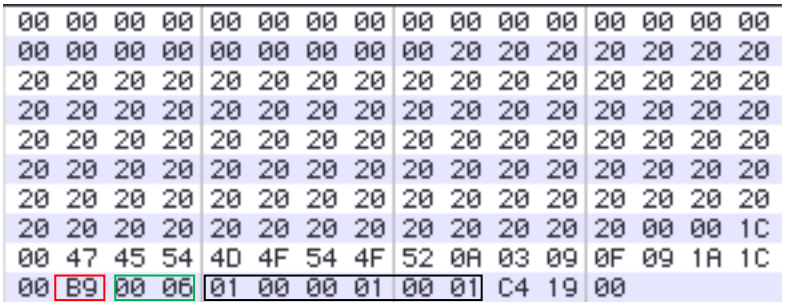
Durch den GETMOTORSET-Befehl liest der **Mikrokontroller** den aktuellen Positionswert eines Servomotors.
Es handelt sich um 2 Möglichkeiten für die Parameter [Motor n der Gruppe]: 0 oder 1. 1 steht für das Auslesen von der aktuellen Wert des ausgewählten Motors. 0 wird verwendet, damit der Motor in die Standardposition von 0 bewegen soll.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	B9*
Byte 2 bis 3	Gruppenbytecode
Byte 4 bis zum Ende des Statements	Wenn Parameter 1 ist: 01 Wenn Parameter 0 ist: 00

Befehlsstruktur:
GETMOTORSET [Gruppencode],[0|1], …

Beispiel: GETMOTORSET G6A,1,0,0,1,0,1



1.3.14 MUSIC

Der MUSIC-Befehl spielt Musik durch Mikrokontroller. Die Parameter [play music line] entspricht aus folgende Tabelle:

Notenfolge	Beschreibung
C	„Do“
D	„Re“
E	„Mi“
F	„Fa“
G	„Sol“
A	„La“
B	„Si“
[	Eine Gruppe von Tönen um das 1,5fache kürzen
]	Eine Gruppe von Tönen um das 1,5fache verlängern
O	Eine Oktave auswählen
M	3 Oktaven auswählen
.	Einen Ton um das 1,5fache verlängern
#, +	Einen Ton erhöhen (#)
$, -	Einen Ton erniedrigen (♭)
P, ,(Rest von a)	Rest von A
<, L	Oktave fallen lassen
>, H	Oktave anheben

Der entsprechende Hexcode ist BA

Befehlsstruktur:
MUSIC “[Notenfolge]”

Beispiel: MUSIC "CDE"

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 1D
00 4D 55 53 49 43 2D 43 44 0A 03 09 0F 0C 26 1D
00 BA 1A 00 18 24 18 26 18 28 00 C4 1A 00
```

1.3.15 TEMPO

Der TEMPO-Befehl legt das Tempo von einem Musikstück fest.

Der entsprechende Hexcode für TEMPO ist BF

Befehlsstruktur:
TEMPO [Wert]

Beispiel: TEMPO 100

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 16
00 74 65 6D 70 6F 31 30 30 0A 03 09 0F 0E 2E 16
00 BF 12 64 C4 13 00
```

1.3.16 IF… THEN

Die Anweisung, die hinter THEN vorkommt, wird ausgeführt, wenn die Bedingung WAHR ist. Andernfalls wird entweder die Bedingung nach ELSEIF oder die Anweisung nach ELSE ausgeführt. Nach der letzten Anweisung soll ein ENDIF kommen.

Der entsprechende Hexcode für IF ist C0
Der entsprechende Hexcode für THEN ist D0
Der entsprechende Hexcode für ENDIF ist C4
Der entsprechende Hexcode für ELSEIF ist C0, sofern die letzte Bedingung nicht erfüllt ist
Der entsprechende Hexcode für ELSE ist D0, sofern die Bedingung davor nicht erfüllt ist

Befehlsstruktur:
IF [Ausdruck] THEN [Anweisung] ELSEIF [Ausdruck] THEN [Anweisung] ELSE [Anweisung] ENDIF

Beispiel:
DIM AUSDRUCK AS BYTE, AUSDRUCK_1 AS BYTE, AUSDRUCK_2 AS BYTE,anweisung_1 AS BYTE,anweisung_2 AS BYTE, anweisung AS BYTE

IF AUSDRUCK=AUSDRUCK_1 THEN
anweisung_1=1
ELSEIF AUSDRUCK=AUSDRUCK_2 THEN
anweisung_2=2

ELSE
anweisung=6
ENDIF

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 06 00 3C
00 41 55 53 44 52 55 43 4B 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 41 55 53 44 52 55 43 4B 5F 31 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 41 00 02 41 55 53 44 52 55 43 4B 5F
32 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 42 00 02 41 4E 57 45 49 53
55 4E 47 5F 31 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 43 00 02 41 4E 57
45 49 53 55 4E 47 5F 32 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 44 00 02
41 4E 57 45 49 53 55 4E 47 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
45 00 02 49 46 2E 62 61 73 20 20 0A 03 09 0F 15
02 3C 00 C0 21 00 15 40 35 15 41 00 D0 15 43 11
00 C4 39 00 C0 33 00 15 40 35 15 42 00 D0 15 44
12 02 00 C4 39 00 D0 15 45 12 06 00 C4 39 00
```

1.3.17 FOR … TO … NEXT …

Die Schlüsselwörter FOR…TO…NEXT bilden eine Schleife. Sie werden immer weiter laufen, sobald die Bedingung hinter FOR erfüllt. Der Zähler, der durch NEXT bezeichnet ist, wird dann weiterzählen.

Der entsprechende Hexcode für FOR ist C1
Der entsprechende Hexcode für TO ist C3
Der entsprechende Hexcode für NEXT ist C2

Befehlsstruktur:
FOR Variable = [Wert | Variable] TO [Wert | Variable]
Anweisungen
NEXT Variable

Beispiel:
```
DIM b AS BYTE
DIM c AS BYTE
DIM d AS BYTE
DIM a AS BYTE
b=2
c=b
d=c+b
FOR a = 0 TO 3
WAIT
NEXT a
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 04 00 39
00 42 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 43 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 41 00 02 44 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 42 00 02 41 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 43 00 02 46 4F 52
2E 62 61 73 20 0A 03 09 0F 16 15 39 00 D0 15 40
12 02 00 D0 15 41 15 40 00 D0 15 42 15 41 21 15
40 00 C1 15 43 10 C3 15 43 12 03 36 00 E9 C2 15
43 29 00 C4 36 00
```

1.3.18 GOTO

Der GOTO-Befehl macht einen Sprung zu einer bestimmten Position des Programms.

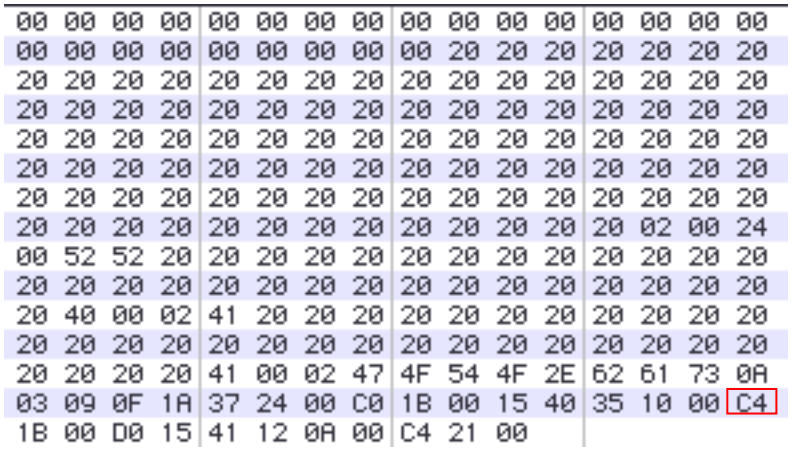
Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte1	C4
Byte2 bis 3	Speicheradresse der Sprungmarke

Befehlsstruktur:
GOTO [Sprungmarke]

Beispiel: DIM RR AS BYTE, A AS BYTE
IF RR = 0 THEN GOTO MAIN1

MAIN1:
A = 10



1.3.19 GOSUB…RETURN

Der Befehl GOSUB…RETURN ruft ein Unterprogramm auf und kehrt danach wieder an der Stelle vor Sprungmarke zurück.

Der entsprechende Hexcode besteht aus folgenden Bytes:
GOSUB:

Byte 1	C5
Byte 2 bis 3	Speicheradresse der Sprungmarke

RETURN:

Byte 1	C6

<u>Befehlsstruktur:</u>
GOSUB [Sprungmarke]
……
[Sprungmarke]: ……
RETURN

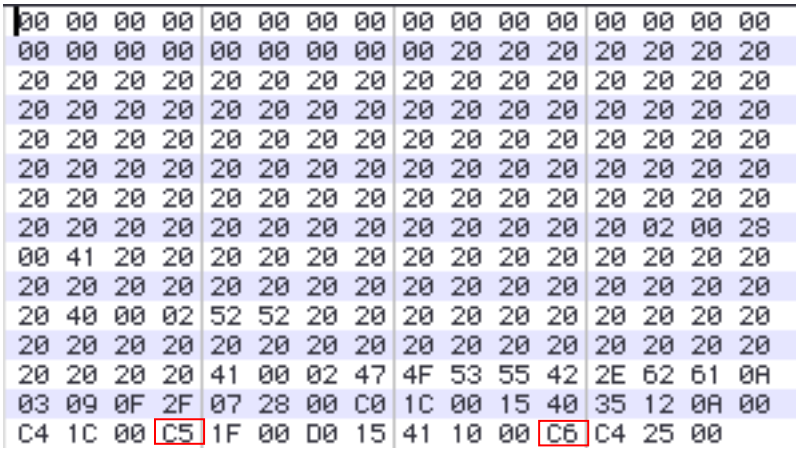
Beispiel: DIM A AS BYTE, RR AS BYTE

IF A=10 THEN GOTO MAIN

MAIN:
GOSUB LABEL

LABEL:
RR = 0

RETURN



1.3.20 ON…GOTO

Der Befehl ON…GOTO vereinfacht den IF-Befehl, falls der Wert der Variable immer um 1 erhöht.

Der entsprechende Hexcode für ON…GOTO ist C7

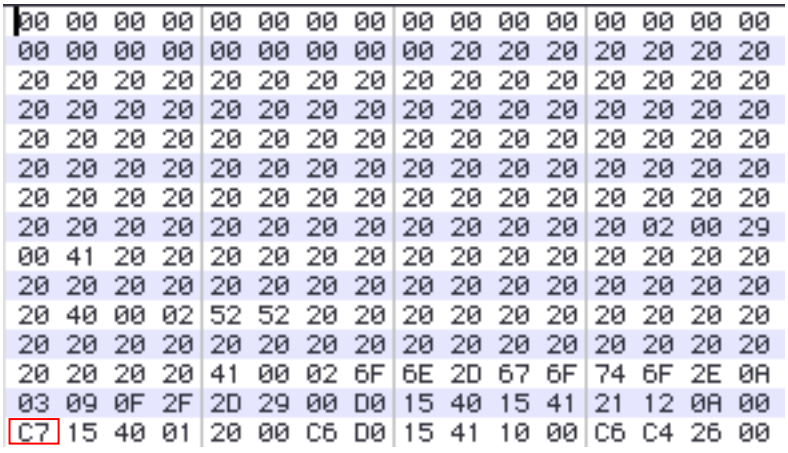
<u>Befehlsstruktur:</u>
ON [Variable] GOTO [Sprungmarke], [Sprungmarke], …

Beispiel: DIM A AS BYTE, RR AS BYTE

A=RR+10

ON A GOTO LABEL
RETURN

LABEL:
RR = 0
RETURN



1.3.21 BREAK

Das Schlüsselwort BREAK macht eine Pause beim Programmdurchführen und wechselt zu Debug-Mode.

Der entsprechende Hexcode ist CB.

Befehlsstruktur:
BREAK

Beispiel: BREAK

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 14
00 62 72 65 61 6B 2E 62 61 0A 03 09 0F 30 12 14
00 CB C4 11 00
```

1.3.22 POKE

Durch den POKE-Befehl wird Daten in den internen Speicher des Roboters geschrieben.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	CC
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
POKE [RAM-Region], [Daten]

Beispiel: POKE &h30, 50

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 18
00 50 4F 4B 45 2E 62 61 73 0A 03 09 0F 31 2C 18
00 CC 12 30 12 32 C4 15 00
```

1.3.23 ROMPOKE

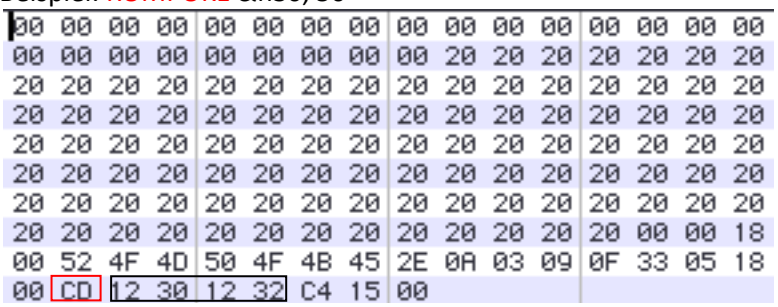
Durch den ROMPOKE-Befehl wird Daten in externen Speicher geschrieben.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	CD
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
ROMPOKE [ROM-Region], [Daten]

Beispiel: ROMPOKE &h30, 50



1.3.24 OUT

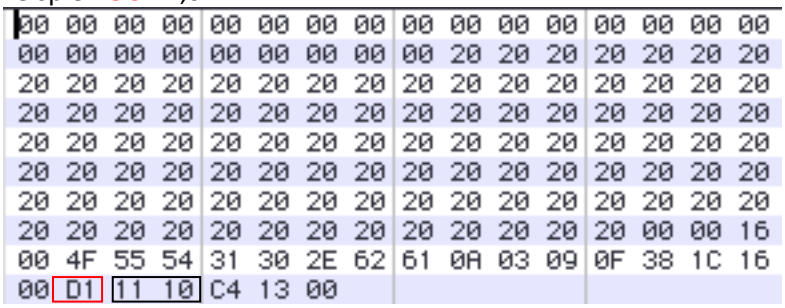
Durch den OUT-Befehl wird ein Signal vom Kontroller gesendet. Der Wert 0 steht für Ausschalten der LED und 1 für Einschalten.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	D1
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
OUT [Portnummer], [Ausgabewert]

Beispiel: OUT 1,0



1.3.25 PULSE

Beim Verwenden von dem PULSE-Befehl wird ein Pulssignal an den Ausgabe-Port gesendet.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	D2
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
PULSE [Portnummer]

Beispiel: PULSE 3

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 16
00 50 55 4C 53 45 33 2E 62 0A 03 09 0F 39 36 16
00 D2 12 03 C4 13 00
```

1.3.26 TOGGLE

Durch den TOGGLE-Befehl wird das Signal am Ausgangsport umgekehrt.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	D3
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
TOGGLE [Portnummer]

Beispiel: TOGGLE 4

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 16
00 54 4F 47 47 4C 45 34 2E 0A 03 09 0F 3B 10 16
00 D3 12 04 C4 13 00
```

1.3.27 DELAY

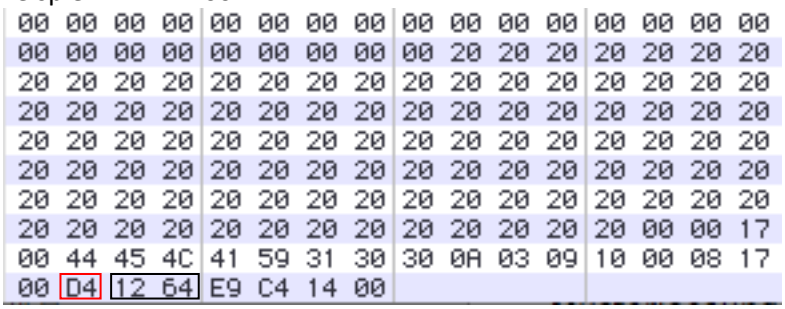
Durch den DELAY-Befehl wird die Ausführung eines Programmes um bestimmte Zeit verzögert.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	D4
Byte 2 bis zum vorletzten Byte	Standardformat für Zahlenwerte oder Variable
Letzte Byte	E9

Befehlsstruktur:
DELAY [Verzögerungsdauer]

Beispiel: DELAY 100



1.3.28 ERX

Empfangen des RS232 Signals von ERX-Port. [Portgeschwindigkeit] steht für Übertragungsgeschwindigkeit der Ports und ist eine Konstant wie folgendes Tabelle:

Nummer	Port Einstellung
2400	2400bps, 8Bit Datei, kein Parität, 1 Stoppbit
4800	4800bps, 8Bit Datei, kein Parität, 1 Stoppbit
9600	9600bps, 8Bit Datei, kein Parität, 1 Stoppbit
14400	14400bps, 8Bit Datei, kein Parität, 1 Stoppbit
19200	19200bps, 8Bit Datei, kein Parität, 1 Stoppbit
28800	28800bps, 8Bit Datei, kein Parität, 1 Stoppbit
38400	38400bps, 8Bit Datei, kein Parität, 1 Stoppbit
57600	57600bps, 8Bit Datei, kein Parität, 1 Stoppbit
76800	76800bps, 8Bit Datei, kein

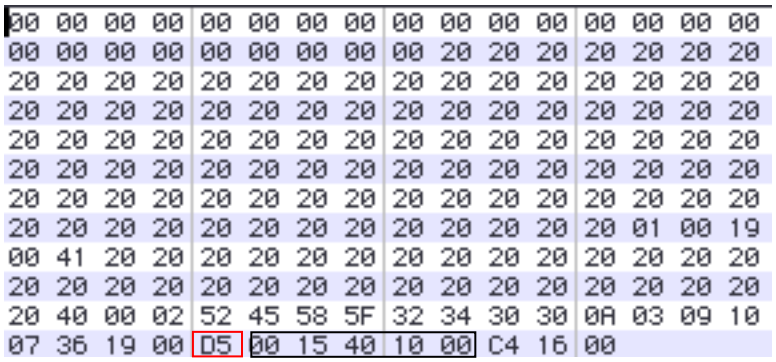
	Parität, 1 Stoppbit
115200	115200bps, 8Bit Datei, kein Parität, 1 Stoppbit
230400	230400bps, 8Bit Datei, kein Parität, 1 Stoppbit

Übliche Variable werden für Speicherung empfangenen Daten und Markierung noch nicht übertragene Daten verwendet.

Der entsprechende Hexcode für ERX ist D5.

Befehlsstruktur:
ERX [Portgeschwindigkeit], [Empfangsvariablen], [Fehlererkennungslabel]

Beispiel: DIM A AS BYTE
LABEL:
ERX 2400,A, LABEL



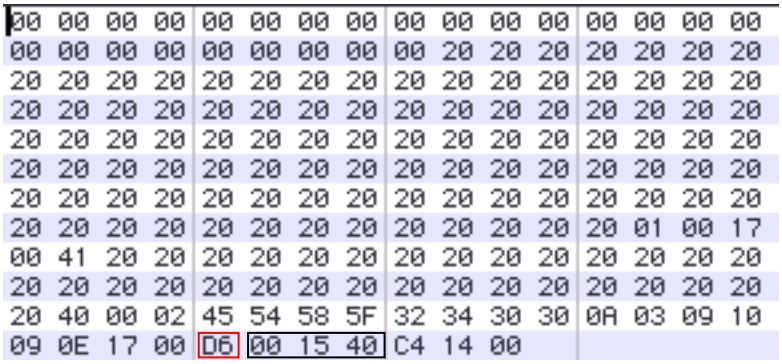
1.3.29 ETX

[Portgeschwindigkeit] ist die Portübertragungsgeschwindigkeit. Siehe obige Tabelle [Daten] kann Zahl, Variable und Konstant sein, die durch ETX-Port übertragen wird.

Der entsprechende Hexcode für ETX ist D6.

Befehlsstruktur:
ETX [Portgeschwindigkeit], [Daten]

Beispiel: DIM A AS BYTE
ETX 2400,A



1.3.30 MOTOR

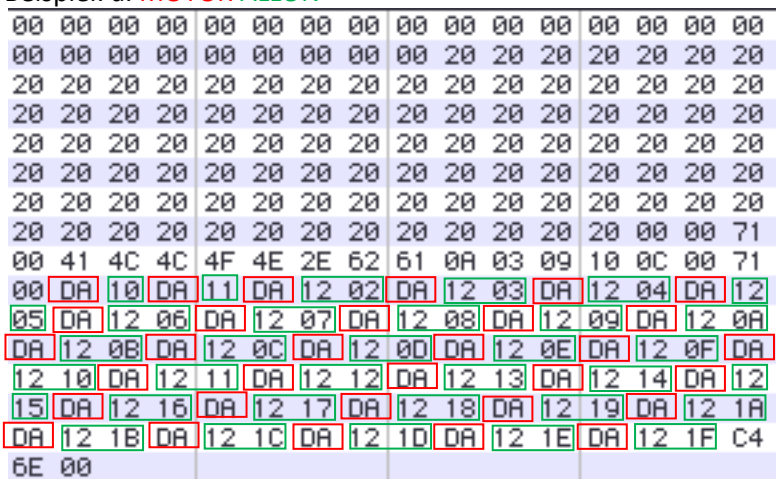
Durch den MOTOR-Befehl wird ein Motor oder eine Motorgruppe eingeschaltet.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	DA
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

<u>Befehlsstruktur:</u>
MOTOR [Motornummer] oder MOTOR [Motornummer] / [Spezielle Gruppe]

Beispiel: a. MOTOR ALLON



b. MOTOR 0

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 15
00 4D 4F 54 4F 52 5F 30 2E 0A 03 09 10 0B 0D 15
00 DA 10 C4 12 00
```

c. MOTOR G6B

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 25
00 4D 4F 54 4F 52 5F 47 36 0A 03 09 10 0C 28 25
00 DA 12 06 DA 12 07 DA 12 08 DA 12 09 DA 12 0A
DA 12 0B C4 22 00
```

d. MOTOR G24

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 59
00 4D 4F 54 4F 52 5F 47 32 0A 03 09 10 0D 20 59
00 DA 10 DA 11 DA 12 02 DA 12 03 DA 12 04 DA 12
05 DA 12 06 DA 12 07 DA 12 08 DA 12 09 DA 12 0A
DA 12 0B DA 12 0C DA 12 0D DA 12 0E DA 12 0F DA
12 10 DA 12 11 DA 12 12 DA 12 13 DA 12 14 DA 12
15 DA 12 16 DA 12 17 C4 56 00
```

1.3.31 MOTOROFF

Durch den MOTOROFF-Befehl wird der Motor ausgeschaltet.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	DB
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Wenn nachfolgende Parameter 0 ist, ist der entsprechende Hexcode für MOTOROFF DA.

Befehlsstruktur:

MOTOROFF [Motornummer] oder MOTOR [Motornummer] / [Spezielle Gruppe]

Beispiel: a. MOTOROFF ALLOFF

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	71
00	41	4C	4C	4F	46	46	2E	62	0A	03	09	10	0F	11	71
00	DB	10	DB	11	DB	12	02	DB	12	03	DB	12	04	DB	12
05	DB	12	06	DB	12	07	DB	12	08	DB	12	09	DB	12	0A
DB	12	0B	DB	12	0C	DB	12	0D	DB	12	0E	DB	12	0F	DB
12	10	DB	12	11	DB	12	12	DB	12	13	DB	12	14	DB	12
15	DB	12	16	DB	12	17	DB	12	18	DB	12	19	DB	12	1A
DB	12	1B	DB	12	1C	DB	12	1D	DB	12	1E	DB	12	1F	C4
6E	00														

b. MOTOROFF 0

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	4D	4F	54	4F	52	4F	46	46	0A	03	09	10	10	0D	15
00	DA	10	C4	12	00										

c. MOTOROFF G8B

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	2B
00	47	38	42	2E	62	61	73	20	0A	03	09	10	10	39	2B
00	DB	12	08	DB	12	09	DB	12	0A	DB	12	0B	DB	12	0C
DB	12	0D	DB	12	0E	DB	12	0F	C4	28	00				

d. MOTOROFF G24

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	59
00	47	32	34	2E	62	61	73	20	0A	03	09	10	0E	1B	59
00	DB	10	DB	11	DB	12	02	DB	12	03	DB	12	04	DB	12
05	DB	12	06	DB	12	07	DB	12	08	DB	12	09	DB	12	0A
DB	12	0B	DB	12	0C	DB	12	0D	DB	12	0E	DB	12	0F	DB
12	10	DB	12	11	DB	12	12	DB	12	13	DB	12	14	DB	12
15	DB	12	16	DB	12	17	C4	56	00						

1.3.32 SPEED

Das Schlüsselwort SPEED legt fest, was für eine Geschwindigkeit der Servomotoren gesetzt werden soll. Der Parameter hinter SPEED darf nicht größer als 5 sein.

Der entsprechende Hexcode besteht aus folgenden Bytes:

Byte 1	DD
Byte 2 bis zum Ende des Blocks	Standardformat für Zahlenwerte oder Variable

Befehlsstruktur:
SPEED [Motorgeschwindigkeit]

Beispiel: SPEED 10

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	16
00	53	50	45	45	44	5F	31	30	0A	03	09	10	11	36	16
00	DD	12	0A	C4	13	00									

1.3.33 PWM

PWM-Port und Servokontroll-Port verbinden mit den MR-C2000 Serie-Kontrollern. Die Servofunktion wird abgebrochen, wenn PWM-Befehle ein Pulsbreitwert ausgegeben ist.

Der entsprechende Hexcode ist DE

Befehlsstruktur:
PWM [Motor Nr.],[Pulsbreitwert] (Motor-Nr. ist von 0 bis 5)

Beispiel: PWM 2,102

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	18
00	50	57	4D	5F	32	5F	31	30	0A	03	09	10	15	21	18
00	DE	12	02	12	66	C4	15	00							

1.3.34 LCDINIT

Durch die Nutzung des LCDINIT-Befehles wird LCD-Modul initialisiert, damit die ungewollten Zeichen verhindert werden.

Der entsprechende Hexcode ist E0

Befehlsstruktur:
LCDINIT

Beispiel: LCDINIT

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	14
00	4C	43	44	49	4E	49	54	2E	0A	03	09	10	16	18	14
00	E0	C4	11	00											

1.3.35 CLS

Durch die Nutzung des CLS-Befehles wird alle Zeichen im LCD-Modul gelöscht.

Der entsprechende Hexcode ist E1

Befehlsstruktur:
CLS

Beispiel: CLS

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 14
00 43 4C 53 2E 62 61 73 20 0A 03 09 10 17 00 14
00 E1 C4 11 00
```

1.3.36 LOCATE

Numerale, Konstanten und Variablen können mit dem LOCATE-Befehl in einem 16x2 LCD-Modul für die Koordinaten x und y genutzt werden.

Der entsprechende Hexcode ist E2.

Befehlsstruktur:
LOCATE [x-Koordinate | Bytewert],[y-Koordinate | Bytewert]

Beispiel: LOCATE 5,0

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 17
00 4C 4F 43 41 54 45 5F 35 0A 03 09 10 17 38 17
00 E2 12 05 10 C4 14 00
```

1.3.37 PRINT

Der PRINT-Befehl wird benutzt, damit ein Zeichen an dem Zeiger ausgegeben wird.

Der entsprechende Hexcode ist E3

Befehlsstruktur:
PRINT “[Inhalt]”

Beispiel: PRINT “ROBOT”

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	1B
00	50	52	49	4E	54	2E	62	61	0A	03	09	10	19	30	1B
00	E3	18	52	4F	42	4F	54	00	C4	18	00				

1.3.38 CSON / CSOFF

Um den Zeiger auf dem LCD-Modul zu zeigen bzw. zu verstecken werden die CSON/CSOFF-Befehle genutzt.

Der entsprechende Hexcode von CSON ist E4
Der entsprechende Hexcode von CSOFF ist E5

Befehlsstruktur:
CSON / CSOFF

Beispiel: CSON
CSOFF

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	43	53	5F	4F	4E	5F	4F	46	0A	03	09	10	20	27	15
00	E4	E5	C4	12	00										

1.3.39 CONT

Die Stärke der Farbe von LCD-Modul kann mit dem CONT-Befehl eingestellt werden. Je größer der [Kontrastwert] ist, desto dunkler werden die Zeichen.

Der entsprechende Hexcode ist E6

Befehlsstruktur:
CONT[Kontrastwert]

Beispiel: CONT 10

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 16
00 43 4F 4E 54 5F 31 30 2E 0A 03 09 10 21 1E 16
00 E6 12 0A C4 13 00
```

1.3.40 BYTEOUT

An einem Port, der durch Bytewert oder Variable genutzt wurden, wird ein Ausgabesignalwert ausgegeben. Für den Signalwert werden den Bytewert zwischen 0-255 oder Bytevariable genutzt.

Der entsprechende Hexcode ist E7

Befehlsstruktur:
BYTEOUT [Byte-Portnummer], [Ausgabewert]

Beispiel: BYTEOUT 1, 10

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 17
00 42 59 54 45 4F 55 54 5F 0A 03 09 10 22 21 17
00 E7 11 12 0A C4 14 00
```

1.3.41 STOP/ RUN

Mit dem Befehl "STOP" kann ein laufendes Programm beendet werden. Mit dem Befehl "RUN" kann eine Programmausführung gestartet werden. Um eine beendende Programmausführung erneut zu initialisieren, kann das Programm mit "RUN" benutzen wird.

Der entsprechende Hexcode von STOP ist EA
Der entsprechende Hexcode von RUN ist EB

Befehlsstruktur:
STOP/ RUN

Beispiel: RUN
STOP

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	16
00	52	2B	53	2E	62	61	73	20	0A	03	09	10	23	0E	16
00	EB	EA	E9	C4	13	00									

1.3.42 PTP

Mit der Verwendung des PTP-Befehles kann sich die Endzeitpunkt der Motoren untereinander von mehreren Bewegungen unterscheiden, damit es bei Robotern zu instabilen Bewegungen kommen kann. Mit SETON wird individuelle Gruppe von Servomotoren aktiviert, mit ALLON werden alle Servomotoren eingestellt.

Der entsprechende Hexcode von PTP SETON ist ED 01
Der entsprechende Hexcode von PTP SETOFF ist ED 00
Der entsprechende Hexcode von PTP ALLON ist B6 06
Der entsprechende Hexcode von PTP ALLOFF ist B6 05

Befehlsstruktur:
PTP [SETON | SETOFF | ALLON | ALLOFF]

Beispiel: a. PTP SETOFF

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	53	45	54	4F	46	46	2E	62	0A	03	09	10	24	20	15
00	ED	00	C4	12	00										

b. PTP SETON

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	53	45	54	4F	4E	2E	62	61	0A	03	09	10	24	08	15
00	ED	01	C4	12	00										

c. PTP ALLOFF

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	41	4C	4C	4F	46	46	2E	62	0A	03	09	10	25	1D	15
00	B6	05	C4	12	00										

d. PTP ALLON

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	15
00	41	4C	4C	4F	4E	2E	62	61	0A	03	09	10	25	03	15
00	B6	06	C4	12	00										

1.3.43 FPWM

Die Veränderung der Frequenz des ausgebenden PWM-Pulses wird bei FPWM-Befehl realisiert.

Der entsprechende Hexcode von FPWM ist EE

<u>Befehlsstruktur:</u>

PWM [Port], [Frequenz], [Geschwindigkeit]

Port: 0-2

Frequenz: kleine Frequenz bis hohe Frequenz (1-5)

Geschwindigkeit: 0-255

Beispiel: FPWM 1, 5, 100

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	00	00	19
00	46	50	57	4D	5F	31	5F	35	0A	03	09	10	26	1C	19
00	EE	11	12	05	12	64	C4	16	00						

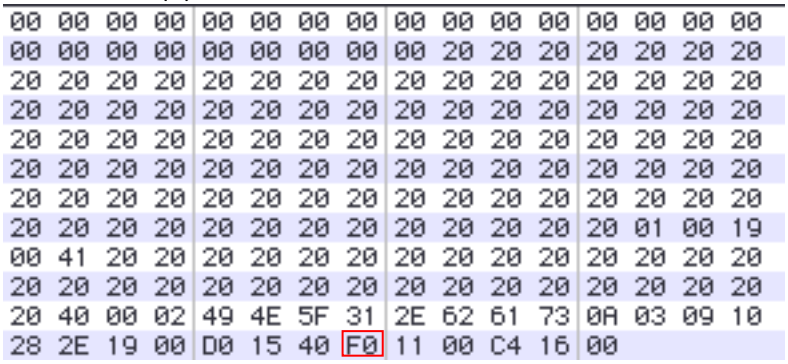
1.3.44 IN

Ein digitaler Wert wird als Variable gespeichert, der durch einen Port eingelesen wird. Die Werte werden in der entsprechenden Variablen mit 0 oder 1 gespeichert.

Der entsprechende Hexcode von IN ist F0

Befehlsstruktur:
Variable = IN([Portnummer])

Beispiel: DIM A AS BYTE
A=IN (1)



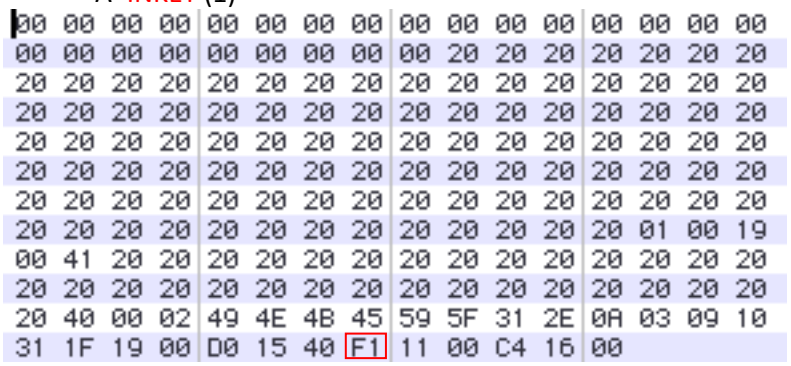
1.3.45 INKEY

Der INKEY-Befehl wird genutzt, um die Eingabe des Wertes von Tasten über einen bestimmten Port zu lesen. Er ist wie eine Schutzfunktion in die Software.

Der entsprechende Hexcode von INKEY ist F1

Befehlsstruktur:
Variable = INKEY([Portnummer])

Beispiel: DIM A AS BYTE
A=INKEY (1)



1.3.46 BYTEIN

Ein digitaler Wert wird als Variable gespeichert, der durch einen Byte-Eingangsparametersport eingelesen wird. Die Werte werden in der entsprechenden Variablen mit 0 oder 1 gespeichert.
Byte-Port 0 entsprechen Ports von einer niedrigeren Ordnung #0 bis #7.
Byte-Port 1 entsprechen Ports von einer niedrigeren Ordnung #8 bis #11

Der entsprechende Hexcode von BYTEIN ist F2

Befehlsstruktur:
Variable = BYTEIN([Byte-Portnummer])

Beispiel: DIM A AS BYTE
A=BYTEIN (1)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 19
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 42 59 54 45 49 4E 5F 31 0A 03 09 10
30 15 19 00 D0 15 40 F2 11 00 C4 16 00
```

1.3.47 AD

Um die Umwandlung eines anlagen Signals in dein digitales Signals zu realisieren, wird der AD-Befehl benutzt. Die AD-Transformation-Ports sind von 0 bis 7 vorhanden.

Der entsprechende Hexcode von AD ist F3

Befehlsstruktur:
Variable = AD ([AD-Port])

Beispiel: DIM A AS BYTE
A=AD (1)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 19
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 41 44 5F 31 2E 62 61 73 0A 03 09 10
33 0E 19 00 D0 15 40 F3 11 00 C4 16 00
```

1.3.48 STATE

Nach dem Senden eines Signals wird mit dem STATE-Befehl den aktuellen Wert an einem Ausgangsport gelesen und in Variable gespeichert.

Der entsprechende Hexcode von STATE ist F6

Befehlsstruktur:
Variable = STATE([Portnummer])

Beispiel: DIM A AS BYTE
A=STATE (1)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 19
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 53 54 41 54 45 5F 31 2E 0A 03 09 10
3B 0A 19 00 D0 15 40 F6 11 00 C4 16 00
```

1.3.49 RND

Durch die Benutzung des RND-Befehls können Zufallszahlen zwischen 0 und 255 generiert.

Der entsprechende Hexcode von BYTEIN ist F8

Befehlsstruktur:
Variable = RND

Beispiel: DIM A AS BYTE
A=RND

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 18
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 52 4E 44 2E 62 61 73 20 0A 03 09 10
3B 30 18 00 D0 15 40 F8 00 C4 15 00
```

1.3.50 PEEK

Die Daten, die vorher an der bekannten Stelle des Roboters gespeichert werden, werden in Variable gespeichert. Im Gegensatz wird dieser Befehl nicht benutzt.

Der entsprechende Hexcode von PEEK ist F9

<u>Befehlsstruktur:</u>
Variable = PEEK([Speicherregion])

Beispiel: DIM A AS BYTE
A=PEEK (32)

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	01	00	1A
00	41	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	40	00	02	50	45	45	4B	5F	33	32	2E	0A	03	09	11
01	36	1A	00	D0	15	40	F9	12	20	00	C4	17	00		

1.3.51 ROMPEEK

Im Gegensatz zu PEEK-Befehl nutzt der ROMPEEK-Befehl den EEPROM zur Speicherung.

Der entsprechende Hexcode von ROMPEEK ist FA

<u>Befehlsstruktur:</u>
Variable = ROMPEEK([Speicherregion])

Beispiel: DIM A AS BYTE
A=ROMPEEK (32)

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	01	00	1A
00	41	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20	40	00	02	52	4F	4D	50	45	45	4B	5F	0A	03	09	11
02	20	1A	00	D0	15	40	FA	12	20	00	C4	17	00		

1.3.52 AIMOTORIN

Den aktuellen Winkel eines AI-Motors wird mit diesem Befehl in der Variable speichert, um den aktuellen Positionswert einzulesen. Für [Motornummer] dürfen Numerale nicht größer als 31 sein, weil es sich nur um insgesamt 32 AI-Motoren handelt.

Der entsprechende Hexcode von AIMOTOR ist FC

Befehlsstruktur:
Variable = AIMOTORIN ([Motornummer])

Beispiel: DIM A AS BYTE
A=AIMOTORIN (3)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 1A
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 41 49 4D 4F 54 4F 52 49 0A 03 09 11
04 36 1A 00 D0 15 40 FC 12 03 00 C4 17 00
```

1.3.53 SONAR

Mit diesem Befehl werden die berechneten Distanzen eines Ultraschallsensors ausgelesen. Die Ein- und Ausgangsports werden 0 bis 11 genutzt.

Port #0	#0 Ultraschallausgangsport
Port #1	#0 UltraschallEingangsparametersport
Port #2	#1 Ultraschallausgangsport
Port #3	#1 UltraschallEingangsparametersport
Port #4	#2 Ultraschallausgangsport
Port #5	#2 UltraschallEingangsparametersport
Port #6	#3 Ultraschallausgangsport
Port #7	#3 UltraschallEingangsparametersport
Port #8	#4 Ultraschallausgangsport
Port #9	#4 UltraschallEingangsparametersport
Port #10	#5 Ultraschallausgangsport
Port #11	#5 UltraschallEingangsparametersport
Port #12	#6 Ultraschallausgangsport
Port #13	#6 UltraschallEingangsparametersport
Port #14	#7 Ultraschallausgangsport
Port #15	#7 UltraschallEingangsparametersport
Port #16	#8 Ultraschallausgangsport
Port #17	#8 UltraschallEingangsparametersport

Port #18	#9 Ultraschallausgangsport
Port #19	#9 UltraschallEingangsparametersport
Port #20	#10 Ultraschallausgangsport
Port #21	#10 UltraschallEingangsparametersport
Port #22	#11 Ultraschallausgangsport
Port #23	#11 UltraschallEingangsparametersport

Der entsprechende Hexcode von SONAR ist FD

Befehlsstruktur:
SONAR([Ultraschallport])

Beispiel: DIM A AS BYTE
A=SONAR (3)

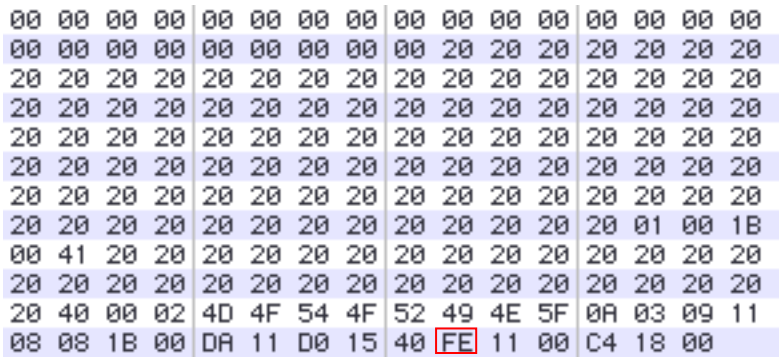
```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 01 00 1A
00 41 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 40 00 02 53 4F 4E 41 52 5F 33 2E 0A 03 09 11
06 32 1A 00 D0 15 40 FD 12 03 00 C4 17 00
```

1.3.54 MOTORIN

Den aktuellen Winkel eines Motors wird mit diesem Befehl in der Variable speichert, um den aktuellen Positionswert einzulesen. Für [Motornummer] dürfen Numerale nicht größer als 31 sein, weil es sich nur um insgesamt 32 Motoren handelt.
Der entsprechende Hexcode von MOTORIN ist FE

Befehlsstruktur:
Variable = MOTORIN([Motornummer])

Beispiel: DIM A AS BYTE
MOTOR 1
A=MOTORIN (1)



1.4 Kodierung der Operatoren

Operator	Bytecode
+	21
-	22
x	23
/	24
MOD	26
AND	27
OR	28
XOR	2A
NOT	EF
<	30
>	31
<=	32
>=	33
!= (ungleich)	34
= (Als Operator)	35
= (Als Wertzuweisung)	D0

2) Herstellung vom XML Datei

Nachdem alle entsprechenden Hex-Codes gefunden worden sind, wird eine Datei als XML-Format mit alle von diesem Format erforderlichen Etiketten erstellt. In dieser Datei werden die Codes durch Nummerierung sortiert und als Datenbank gespeichert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  edited with XMLSpy v2009 sp1 (http://www.altova.com) by ZHAO (TU-Claustha
- <Table>
    <Befehler>Befehler im Hexcode</Befehler>
    <Nr>1</Nr>
    <Code>BREAK</Code>
    <HEX-Code>CB</HEX-Code>
    <Nr>2</Nr>
    <Code>PTP</Code>
    <HEX-Code>ED</HEX-Code>
    <Nr>4</Nr>
    <Code>DIR</Code>
    <HEX-Code>B4</HEX-Code>
    <Nr>5</Nr>
    <Code>GETMOTORSET</Code>
    <HEX-Code>B9</HEX-Code>
    <Nr>6</Nr>
    <Code>SPEED</Code>
    <HEX-Code>DD</HEX-Code>
```

3) Programm: Der Interpreter des Robobasics

1. Sinn und Zweck des Programms

Das in dieser Arbeit entwickelte Programm ist eine Verbesserung und Erweiterung von der originalen RoboBASIC-Umgebung. Mit diesem Programm ist Erstellung und Bearbeitung der RoboBASIC-Instruktionen unter allen Betriebssystemen möglich.

Die Programmierungsfehler werden auch durch dieses Programm spezifiziert, damit man unterscheiden kann, ein Buchstabierfehler oder Syntaxfehler aufgetaucht hat.

Der Programm wurde unter der Umgebung Pelles C Version 6.00.4 for Windows 64 bit entwickelt.

2. Algorithmus

Der Algorithmus von diesem Programm lässt sich durch folgende Abschnitte unterteilen:

-Einlesen:

Am Anfang werden die Quelldatei und vorher erstellte XML-Datei vom Programm eingelesen. Danach werden die Quellcodes von diesen 2 Dateien Wort für Wort in einer Struktur-Array gespeichert und indexiert.

Die Typdefinition von Struktur-Array für Speicherung der Quelldatei besteht aus einem Unterarray, dieses Array speichert ein eingelesenes Wort, das sich durch Leerzeichen oder Komma von anderen Wörtern trennt. Alle eingelesenen Wörter werden in Großbuchstaben gespeichert. Die Typdefinition von Struktur-Array für Speicherung der XML-Datei besteht aus 2 Unterarrays, eins davon speichert einen Befehl und das andere speichert den entsprechenden Hex-Code für den Befehl.

-Überprüfen:

Nach dem Einlesen werden alle gespeicherten Quelltextwörter mit der von XML-Datei erzeugten Datenbank verglichen. Jedes Wort von Quelltext wird Byte für Byte mit den Wörter von erzeugten Datenbank vergleichen. Wenn es für ein Quelltextwort eine Übereinstimmung in dem Datenbankarray gibt, überprüft das Programm üblichen Wörter in der gleichen Zeile, ob restlichen Ausdrücke mit der Grammatik von dem gerade gefundenen Wort übereinstimmt. Wenn nicht, dann wird eine Meldung von Syntaxfehler gezeigt. Dann fängt die weitere Überprüfung in der nächsten Zeile an.

Wenn es für ein Quelltextwort keine Übereinstimmung in dem Datenbankarray gibt, wird es nach folgenden Fällen unterschieden:

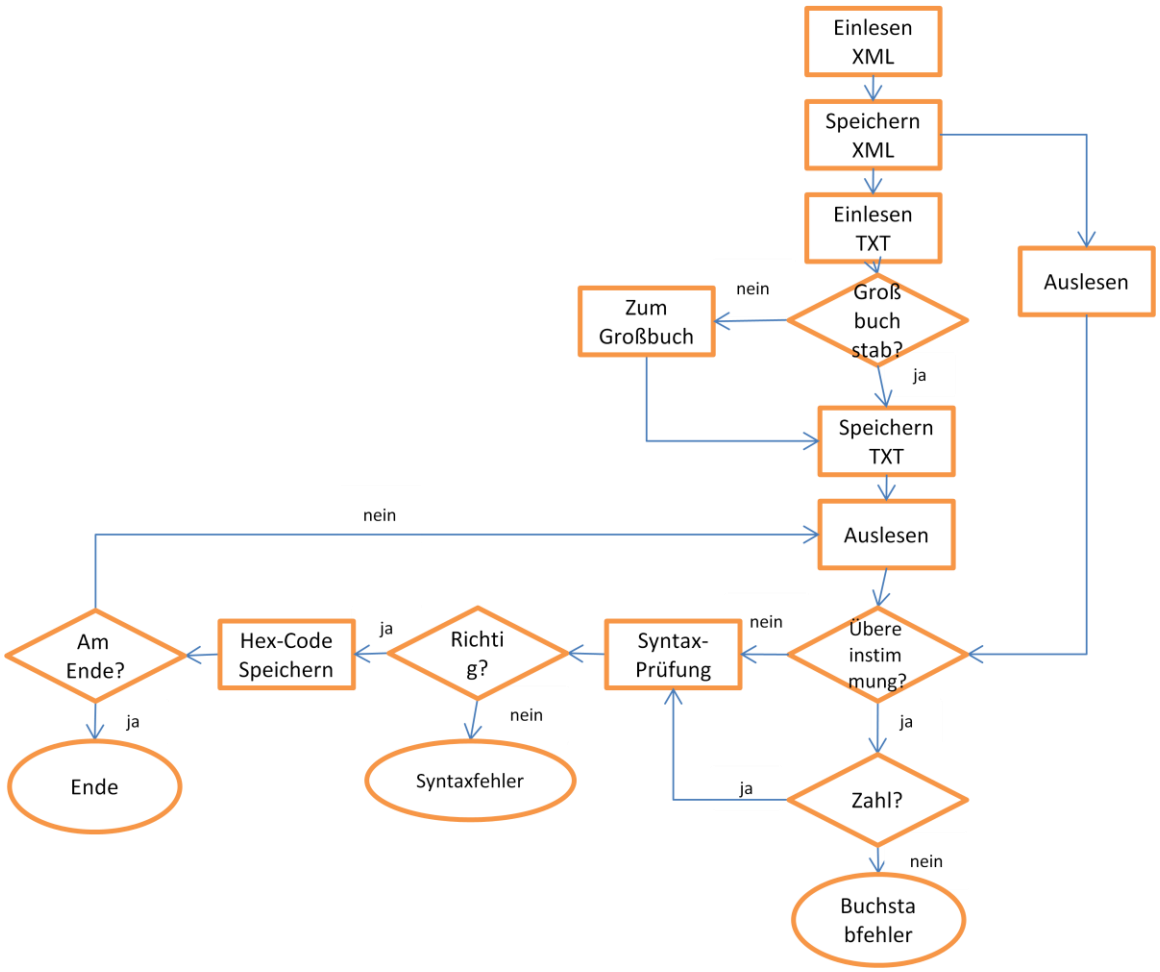
a) Das gerade gelesene Wort ist eine Zahl. Dann wird diese Zahl direkt in eine Hexadezimalzahl konvertiert.

b) Das gerade gelesene Wort ist falsch. Es handelt sich um einen Buchstabierfehler.

-Ausgabe:

Nachdem alle Quelltextwörter richtig eingelesen und überprüft und keine Fehlermeldung gefunden worden ist, gibt das Programm alle entsprechende Hex-Code in einer Textdatei aus, die als Objektcodedatei weiter in dem Mikrokontroller behandelt werden kann.

Das Ablaufdiagramm sieht wie folgendes aus:



3. Das Quellcode

Siehe Beilage.

4. Funktionsbeschreibung

Erklärung der im Programm verwendeten Funktionen:

WriteTextBlockFrom--- Virtualisierung der Speicherraum.

FindInXML--- Konvertierung einer XML-Datai in eine TXT-Datei. Eingangsparametersparameter: strInputXML Ausgang:strDestFile

Sourcelesen--- Einlesen der Quellecodes. Eingangsparameter: strInputTXT

XMLtoTXT--- Einlesen der Datenbank. Eingangsparameter: strInputXML

speed--- Grammatiküberprüfung des Schlüsselwortes Speed. Eingangsparameter: a

GYRODIR--- Grammatiküberprüfung des Schlüsselwortes GYRODIR. Eingangsparameter: a

GYROSENSE--- Grammatiküberprüfung des Schlüsselwortes GYROSENSE. Eingangsparameter: a

byteout--- Grammatiküberprüfung des Schlüsselwortes BYTEOUT. Eingangsparameter: a

fpwm--- Grammatiküberprüfung des Schlüsselwortes FPWM. Eingangsparameter: a

move--- Grammatiküberprüfung des Schlüsselwortes MOVE.
Eingangsparameter: b

aimotor--- Grammatiküberprüfung des Schlüsselwortes AIMOTOR.
Eingangsparameter: b

dth---Konvertierung von Dezimalzahlen in Binärzahlen.
Eingangsparameter: dt

compare--- Durchsuchung in der Datenbank, um Übereinstmmung mit Quelledatei zu finden. Eingangsparameter:strInputXML und strInputTXT

4) Robobasic Erweiterung

Beim RoboBASIC fehlt einige wichtige Programmstruktur, deshalb wird durch folgende Schlüsselwörter erweitert:

1. Erweiterung: Switch

Switch ist eine wichtige Anweisung, um eine Situation von Mehrfachauswahl zu behandeln. Eine Switch-Anweisung kann durch mehrere IF…THEN-Anweisungen gleichwertig realisiert werden.

2. Erweiterung: While

While-Schleife ist eine wichtige Kontrollstruktur, die ein Anweisungsblock solange wiederholt, bis ein eine Laufbedingung gültig ist oder eine Abbruchbedingung erfüllt wird.

Eine While-Schleife kann durch Verwenden einer FOR…NEXT-Schleife gleichwertig realisiert werden.

5. Probleme des Programms und Lösungsvorschlag

1) Offene Probleme

Folgende Probleme bleiben bei unserer Arbeit noch offen:

- Beim erzeugten Code ohne Zusatzinformationen(Dateiname, Datum)
- Es fehlt die Behandlung von benutzerdefinierten Variablen.
- Es fehlt die Behandlung von komplizierten Sprachstrukturen, wie z.B. Unterprogramm, bedingte Anweisung und Schleife.

2) Lösungsvorschlag

- Bei der Behandlung von Variablendefinition:

  Der weitere Programmierer kann eine Referenztabelle erstellen, damit das Programm die entsprechenden Speicheradressen von definierten Variablen nachschlagen kann.

  Zum Beispiel wenn wir folgende Codes vorhanden:
  DIM A AS BYTE
  DIM B AS INTEGER

  Dann eine solche Tabelle kann beispielweise so aussehen:

Variablenname	Speicheradresse	Typ
A	40	02
B	41	03
…	…	---

  Das Programm sieht der Variablenname und führt eine Überprüfung durch. Nachdem es festgelegt hat, dass die Variable kein Schlüsselwort und keine Zahl ist, wird die entsprechende Speicheradresse genommen und für die weitere Bearbeitung zur Verfügung steht.

- Bei der Behandlung von Unterprogramm:

  Die Grundidee ist ähnlich wie bei der Behandlung von Variablen. Wenn das Programm eine Lable findet, merkt es die erschienene Stelle der Lable, damit es zu dieser Stelle zurückkehren kann, sofern es ein Schlüsselwort wie GOSUB findet.

6. Fazit

Durch diese Arbeit wird die originale RoboBASIC-Sprache untersucht. Die entsprechenden Objektcodes werden herausgefunden. Ein neuer Interpreter wird programmiert, damit die Sprachstruktur und die Kompatibilität der Sprache erweitert und verbessert. Nun sollen die Benutzer diese Sprache mit mehr Funktionalitäten unter verschiedene Betriebssysteme zur Steuerung der Robonova-Roboter verwenden können.