



Impressum

Diese Bedienungsanleitung ist eine Publikation der Conrad Electronic GmbH, Klaus-Conrad-Straße 1, D-92240 Hirschau.

Alle Rechte einschließlich Übersetzung vorbehalten. Reproduktionen jeder Art, z. B. Fotokopie, Mikroverfilmung, oder die Erfassung in EDV-Anlagen, bedürfen der schriftlichen Genehmigung des Herausgebers.

**100%
Recy-
cling-
papier.**

Nachdruck, auch auszugsweise, verboten.

Diese Bedienungsanleitung entspricht dem technischen Stand bei Drucklegung. Änderung in Technik und Ausstattung vorbehalten.

**Chlorfrei
gebleicht.**

© Copyright 2003 by Conrad Electronic GmbH. Printed in Germany.

*10-03/HK

Version 10/03



Robot System Robby RP5 ROBOT (CCRP5)

Best.-Nr. 19 03 33

Important! Read Thoroughly!

Please completely read this manual before operating the C-CONTROL ROBOT PROJECT 5 (CCRP5) or attached devices! It will guide you to correct usage and avoid possible dangers.

Conrad Electronics does not assume responsibility for any damage or loss resulting from disregarding the directions in this manual.



Contents

- Important! Read Thoroughly!1
- Contents2
- Preface4
 - Warranty4
 - Service4
- Product Description5
 - Conventional Application5
 - Safety Notes5
 - Features5
 - Chassis and Drive:5
 - Sensors:6
 - Computer:6
 - Operating Conditions6
- Handling8
 - General Handling Notes7
 - Electrostatic Discharge7
 - Power Supply7
 - Writing a Program for the Robot7
 - Programming the Robot8
- Setup8
 - Software Installation8
 - Supplying Power8
 - Connecting the Robot to the PC8
 - Uploading programs to the Robot9
 - A first functionality test9
 - Charging the Batteries9
- Programming the CCRP510
 - INTRODUCTION EINFÜHRUNGEN_CCBASIC for C-Control Beginners10
 - Examples on the CD10
 - NTRODUCTION_CCRP5 for Advanced C-Control Users10
 - Examples on the CD10
- Extended System Resources11
 - POWER SWITCHES11
 - ACS12
 - IRCOMM12
 - NAV14
 - LED CONTROL14
 - DRIVE CONTROL14
- Operation at 12 Mhz (overclocking)16
 - General15
 - Restrictions15
- C-Control I/O Resources15
 - Used C-Control I/O Resources15
 - Free C-Control I/O Resources16

The CCBASIC Programming Language	16
Realtime Clock	16
User Bytes	17
Digital Ports	17
Analog Ports	17
Active Antenna	17
Frequency In/Outputs.....	18
Interrupt Input.....	18
Expansion Connector	18
Programming the C-Control Computer.....	18
The CC-BASIC Introduction Examples	18
What is a Program?	18
Basics of the CCBASIC Programming Language	19
General Topics	19
Identifiers	19
Variables and Constants.....	19
Labels	19
Terms	19
Operands und Operators	20
Functions	20
Assignments	20
Instructions.....	20
Instructions for Program Flow Control	20
Compiler Instructions	20
Defining Symbolic Constants	20
Defining Variables.....	22
Defining Digital Ports	22
Defining Analog Ports.....	22
Mathematical and Logical Operators	22
Precedence of Operators and Function Calls	23
Program Flow Control Statements	23
Serial Interface Communication	26
File Functions	28
Port Commands	29
Defining and Using Data Tables	30
Accessing the Real Time Clock	31
Internal Timer, Sound Generation, Frequency Measurement	31
Embedding Machine Language Programs	32
Troubleshooting.....	32
Programs can not be loaded	32
A program was loaded, but it doesn't start	33
The Roboter executes uncontrolled RESETs during operation	33
Mysterious disturbances in interrupt mode	33
sensors read wrong or no values	33
Operating Elements, Display Elements and Connectors.....	34
Expansion Connector 1	34
Expansion Connector 2	35
Technical Data	35

Preface

Thank you for your decision to purchase the CCRP5. This mobile Robot is equipped with a programmable microcomputer that allows you to determine basic reactions and behaviour to external stimuli. CCRP5 has been designed with the ambition to meet our customers' high demands in quality and functionality.

Conrad Electronic GmbH
D-92240 Hirschau

Warranty

Please completely read this manual before operating the CCRP5 or attached devices! It will guide you for correct usage and avoid possible dangers. Conrad Electronics does not assume responsibility for any damage or loss resulting from disregarding the directions in this manual.

Limitations in Warranty and Liability

The core of the robot is a C-Control/BASIC microcomputer. The embedded microcontroller firmware (ROM-Mask in the MC68HC05B6) and the accompanying PC software is delivered as is.

Conrad Electronic does not guarantee the features to meet individual demands, nor that the software in the microprocessor and the PC-software to run without interruptions or errors in any case.

The user bears all the risk regarding quality and performance of the device including all of the software.

Conrad Electronic guarantees the functionality of the supplied application examples within the conditions specified in the technical data section. Should the robot or the PC software prove to be defective or inadequate beyond those examples, the customer has to assume all cost of service, repair or corrective measures originating thereof.

The warranty of Conrad Electronic is limited exclusively to the exchange of devices within warranty time at obvious hardware defects, such as mechanical damage, missing or wrong population of electronic components, excluding socketed integrated circuits and jumpers. No liability will be assumed for any damage resulting directly or indirectly from the use of the robot.

Not taken into account are claims resulting from legal regulations regarding product liability.

Conrad Electronic does not assume liability for any damage or loss resulting from the use of the Robot!

Service

Conrad Electronics has a competent team of service technicians solely for the purpose of assisting you. Each request will be handled as quick as possible. Special requests regarding the use will be passed to the CTC's design engineers.

To avoid unnecessary delays, we would like to ask you to make sure you read the manual, the online help of the programming software, the text- and reference files as well as the pages in the internet before you request help. Most likely you will find the solution to your problem there already!

Please direct your requests to the Technical Customer Support (TKB).

In case of questions, consult our technical information service

Germany: Tel. 0180/5 31 21 18 or 09604/40 88 46

Fax 09604/40 88 44

e-mail: tkb@conrad.de

Mon - Fri 8.00 to 18.00

Austria: Tel. 0 72 42/20 30 60 · Fax 0 72 42/20 30 66

e-mail: support@conrad.at

Mon - Thu 8.00 to 17.00

Fri. 8.00 to 14.00

Switzerland: Tel. 0848/80 12 88 · Fax 0848/80 12 89

e-mail: support@conrad.ch

Mon - Fri 8.00 to 12.00, 13.00 to 17.00

Product Description

Conventional Application

This mobile Robot is equipped with a programmable microcomputer that allows you to determine basic reactions and behaviour to external stimuli.

The Robot CCRP5 has been designed as an experimental platform for the hobbyist interested in studying robotics. It demonstrates in a very tangible experiment the influence and effects of software parameters and physical dimensions perceived by the sensors.

Any usage other than the conventional application is not permitted.

Driving with connectet powersupply is not allowed.

Not for childeren under 14 years of age.

The Robot is developed for non commercial use only.

Safety Notes

Please read this chapter extra carefully!

Disregarding the Safety Notes may lead to Danger for Life due to electric shock or fire!

Resulting from the open design the CCRP5 has **pointy edges and sharp corners**. Therefore it may not be used as a toy for children under 8 years of age. Supervise children that are in the room when operating the CCRP5. According to the programming unforeseen reactions and movements may occur.

Due to the caterpillar drive there are dangerous sectors between wheels and tracks. These sectors are largely covered by the wheel wells of the CCRP5 and therefore mostly secure. Still, take care not to get your fingers between wheels and tracks. Supervise children that are in the room. Do not operate the robot if there are small animals in the room.

Caution ! According to the programming of the robot the motors may inadvertently start to run!

On the surface of the main board there are uncovered components and electrical tracks. Never put tools or metal objects on the surface to avoid shorts.

Before operating the robot, any fluid containers such as coffe cups, bottles or flower vases within reach must be removed.

Do not operate the robot on table tops or on surfaces that may cause the robot to drop to the ground. Also consider the climbing ability of the robot in this respect.

Features

CCRP5 is a programmable microcomputer, equipped with numerous sensors and mounted on a caterpillar drive frame. Suitably programmed, the CCRP5 is a fully functional small robot capable of picking up and reacting to environmental stimuli.

Simultaneously the CCRP5 offers an ideal base for extensions and supplements regarding sensors and actors, e.g. for competitions.

Electrical power is supplied by 6 NiCd rechargeable batteries (substitutional, with limitations, by 6 heavy duty alkaline batteries)

Chassis and Drive:

The robot CCRP5 utilizes a caterpillar drive.

The chassis is constructed from two symmetrical trays. Both of the motors and the spur gear transmissions are integrated therein. The wheel axles and drive shafts are supported in sintered bearings.

The drives are industrial grade motors, featuring high lifetime and an excellent efficiency. This will ensure long operating periods with fully charged batteries.

The transmissions consist of a 12 tooth pinion gear on each of the motor shafts, two combined 50/12 tooth gears and one 50 tooth gear on the track drive shaft. Each of the spur gears are module 0,5 and are made from durable polyamide. Two fork light barriers at the lower side of the Robot mainboard embrace the combined gears and enable a precise distance measurement by way of small drill holes. The rubber-made tracks are driven by wheels, 44 mm in diameter. Due to the well-designed drive the robot is capable of passing small obstacles as well as steep gradients.

The middle of the chassis holds the rechargeable batteries respectively a battery holder that receives six AA-sized batteries.

The robot mainboard is fastened with 4 bolts to the upper side of the chassis and does not normally have to be removed when operating with rechargeable batteries.

Two independently controllable electric motors ensure highest mobility of the chassis. The speed and direction of each track is freely controllable.

Sensors:

Various sensors enable the user to program very complex interactions controlled by sensor data and therefore accordingly versatile reactions of the robot.

- 2 direction-discriminating light sensors
- 2 high resolution distance sensors
- one non-contact IR anti-collision system with switchable range
- one efficient IR communication system (transmitter/receiver)
- sound sensor
- contact sensor
- sensor for operating voltage
- sensor for motor current
- sensor for charge current

Computer:

The microcontroller on the CCRP5 is a computer of the C-Control series.

This compact unit features universal capabilities for measuring, controlling and steering as well as serial data communication and data storage.

The core of the computer is an advanced microprocessor that allows programming in the well-known and easy to learn BASIC programming language. Through a few lines of BASIC source code the computer is able to handle tasks like an intelligent alarm system, a complex data acquisition system, a heater controller or as in this case, the "brains" of a small robot.

To communicate with its environment, it has eight analogue inputs, two analogue outputs and sixteen digital port lines randomly usable as in- or outputs.

These resources are partly used by the sensors and the motor drive controller. The remaining resources may be freely used for additional extensions, even as an extension for a higher performance CPU. In this case the microprocessor may be used as a co-processor by employing a special software.

Operating Conditions

The mechanical innards are well-protected from coarse environmental impairments through the closed chassis tray. Note that it is nevertheless not sealed against water or dust. Please use your robot therefore exclusively in dry and clean domestic areas, since foreign objects or excessive dirt, dust and moisture may damage the mechanics.

The temperature range during operation may not exceed 0°C respectively 40°C.

Do not operate the robot in the vicinity of flammable or combustible fluids, gases or dusts.

The robot has not been designed for commercial applications.

Handling

This chapter gives an overview of the handling of the Robot and its accompanying components. Please take the required detailed information e.g. concerning the programming from the following chapters of this manual respectively from the documentation in the sample programs.

General Handling Notes

Electrostatic Discharge

Depending on the structure of the floor surface, the human body as well as the robot may build up electrostatic charge, especially in low humidity. When touching conductive objects, the electricity may discharge through a spark. Electronic components may be destroyed or damaged through electrostatic discharge. To avoid this, you should touch a grounded and conductive object, such as a metal PC case, a water pipe or a radiator before handling the robot. A possible discharge of the robot to a grounded object is not dangerous, but in some cases may lead to program interruptions or uncontrolled reactions of the robot.

Power Supply

All electrical connections to and from the device have to be established before connecting the power supply. Connecting or disconnecting any wires while the robot is powered up may lead to damage or destruction of the microcontroller or attached devices.

The robot has been designed for a DC voltage of 7.2 Volts, generated by six rechargeable AA size NiCd batteries. Use only certified chargers to recharge the batteries.

As a substitute the batteries may be charged with the wall charger available as an accessory part. In this case an optimum charge may not be guaranteed and also there is no overcharge protection.

The robot may be operated substitutionally by 6 heavy duty alkaline batteries. Due to the higher resistance one has to avoid power surges (e.g. when suddenly reversing driving direction) by programming.

CAUTION:

- **Never attach the Robot to an external power supply if there are no NC-batteries inserted.**
- **Never attach the Robot to an external power supply if the ON/OFF switch is in the OFF position or regular batteries are inserted**
- **Never use any other than the original power supply offered as an accessory**

In any of these cases the robot may be damaged or destroyed by overvoltage.

Writing a Program for the Robot

There is a comfortable Integrated Design Environment for the development of application programs for the robot at your convenience. The IDE is equipped with a standard mouse-controllable graphical user interface with drop-down menus and allows the development of source code (Editor), translating into machine language (Compiler) and uploading the C-Control program to the robot (Loader).

The developed BASIC program, determining the actions and reactions of the robot, will be translated into a sequence of command bytes by the compiler. The commands and the related parameters may then be transferred via serial interface to the microcontroller, and stored into the EEPROM memory.

The C-Control operating system allows your application programs to be stored in a very compact pattern, so that in most cases only a few hundred of the 8000 available bytes are used. Thus a large part of the memory stays free to be used for data recording.

To utilize the robot's resources to the maximum, you may use a special machine-language driver as an extension to the operating system. It will be automatically loaded with the first example.

Further BASIC subroutines facilitate the use of this driver and the handling of sometimes complicated parameters.

CAUTION:

Always write this Initialization line as the first step in your programs

```
REV_L=on:REV_R=on:SYS PLM_SLOW
```

These commands initialize the ports for the direction control of the drive and the PLM frequency.

OPERATING THE MOTORS WITHOUT THIS INITIALIZATION LEADS TO CERTAIN DESTRUCTION OF THE MOTOR DRIVE COMPONENTS!

Programming the Robot

The interface connection between PC and Robot is only necessary while uploading the program. When programmed, it may be disconnected before starting the Robot.

The connection to the PC does not have to be removed when operating the Robot and thus may be used to transfer sensor data or the like. In this case the freedom of movement of course is very much limited, so that this mode of operation is mainly only useful for debugging the application software.

After pressing the start button the operating system of the Robot starts reading and interpreting the commands in the memory until it hits the program end command.

Setup

Software Installation

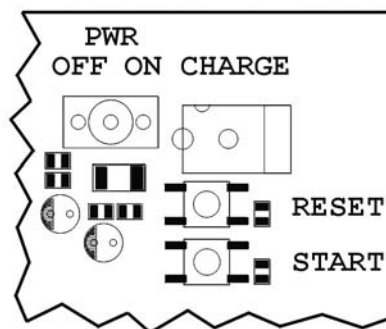
The Robot comes with an Installation CD containing all necessary programs and examples.

Please take the instructions how to set up the software from the file INSTALL.TXT.

After installation you may configure the integrated development environment (IDE) according to your preferences and the assigned interface.

Supplying Power

- Make sure the Robot is not connected to the PC.
- Make sure **and verify** that the ON/OFF switch is in the OFF position (tilted towards the front)

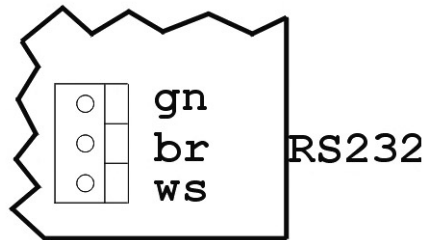


Remove four screws of the main board and cautiously remove the board. Insert 6 NiCd AA batteries into the battery holder. The batteries have to be at least partially charged. Take care to insert the batteries with proper polarity. Now position the main board onto the chassis and fasten it with the four screws.

Please notice the safety instructions on page 38.

Connecting the Robot to the PC

Part of the Robot package is a 9-pin null modem interface cable of approximately 1.5 meters length. Plug it into a free serial interface of your computer. Some older computers have one 9-pin serial interface and one 25-pin serial interface. In case your computer has only the 25-pin interface port available, e.g. because the mouse occupies the 9-pin port, you require an adapter. Now plug the 3-pin end of the interface cord into the header of the Robot and observe the polarity of the wires. The white wire has to point towards the LEDs.



Uploading programs to the Robot

Your programs may be uploaded straight from the Integrated Development Environment (IDE) into the Robot. To do so, the Robot has to be powered up and in RESET-Mode. You will find the relevant item for the upload in the "Development" menu. After the upload is completed, you may start the program by pressing the "Start" button. If your uploaded program activates the drive, you should remove the interface cord before starting the program.

A first functionality test

Switch the Robot's ON/OFF switch to the ON position and press the RESET button on the main board. CCRP5 is now ready to receive the first program, a small functionality test.

Load the program 1_INTRODUCTION_LEDS.BAS in to the IDE. Now choose „BASIC-Compiler“ in the menu "Development". The program will now be compiled (translated in machine language). Watch the "Report"-window – progress and success of the compilation will be listed there. In case the BASIC source code contains errors, you may also find them here. The current example should not contain errors.

After successful compilation choose „in C-ControlUnit übertragen“ in the menu "Development". The commands of the translated program will now be transferred to the C-Control computer, the core of the Robot. Success or failure of the transfer will also be displayed in the "Report" window.

When the upload has been completed successfully, press the START button on the Robot's main board to start the program. You will now see a running light with four LEDs, meaning the Robot is ready for the next step.

CAUTION:

This example loads an important system driver, so do not omit it.

Charging the Batteries

If you are sure you have fully charged batteries, you may leave out this paragraph and return later to read it when it is appropriate.

The Robot is designed to be powered by 7,2 Volts DC, supplied by six NiCd rechargeable batteries. Use only certified chargers to recharge the batteries. As a substitute the batteries may be charged with the wall charger available as an accessory part.

In this case you may use a dedicated program to supervise charging with the accessory wall charger. This program is able to display the charging progress and, to a certain extent, supervise the procedure though there is no overcharge protection and an optimum charge may not be guaranteed.

Recommended Power Supply Conrad Order- No. 51 00 10

Stop the currently running application on the Robot by pressing the RESET button and load the program CHARGE (to be found in the TOOLS directory) into the IDE. Comments and notes at the beginning of the program describe its function.

Now upload the program to the Robot and start it. During charging you have the time and opportunity to read the source code and play with its functionality. If you do not have programming experience with C-Control, you may find a description of the programming language in the chapter PROGRAMMING WITH CCBASIC and some simple program examples on the CD in the directory INTRODUCTION_CCBASIC.

CAUTION:

- **Never attach the Robot to an external power supply if there are no batteries inserted.**
 - **Never attach the Robot to an external power supply if the ON/OFF switch is in the OFF position**
 - **Never use any other than the original power supply offered as an accessory**
- In any of these cases the robot may be damaged or destroyed by overvoltage.**

Programming the CCRP5

PROGRAMMING IN CCBASIC for C-Control Beginners

If you do not have programming experience with C-Control, it would be wise to proceed with the Folder "PROGRAMMING IN CCBASIC" and the accompanying examples.

Examples on the CD

There are 14 examples (001.BAS to 014.BAS) on the CD, that guide you step by step through the programming language CC-BASIC.

GET YOUR ROBOT RUNNING for Advanced C-Control Users

To facilitate the use of the Robot, this Folder contains an introduction that demonstrates and explains the use of the system resources of the CCRP5. These examples always contain the complete code samples required for driving the system resources, because they are needed during system initialization. The relevant code parts of the examples are nevertheless always clearly emphasized and explained.

The last program line needed for loading the system driver is always commented out, since the system driver only needs to be loaded once. This has been done with the first example already.

Examples on the CD

1_INTRODUCTION_LEDS.BAS

This example shows you how to drive the LEDs 1 through 4. The LEDs and the beeper are the only means for output.

1_INTRODUCTION_TOUCHSENSOR_1.BAS

One of the many sensors on the Robot is a touch sensor. It is connected to an A/D converter to distinguish different levels of touching. The output of the sensor is coupled to the LEDs in a bargraph manner to give you a visual impression of the sensor data.

1_INTRODUCTION_TOUCHSENSOR_2.BAS

This example shows how to process this perception and e.g. choose one of four programs.

2_INTRODUCTION_SCHALLSENSOR.BAS

Evaluates sound level and shows it on the LEDs.

3_INTRODUCTION_ACS.BAS

This will explain the subsystem that is part of the Anti Collision System (ACS). The example demonstrates the ACS functionality. Use this program to get an impression of what the ACS can perceive and what not.

3_INTRODUCTION_ACS_SENSITIVITY.BAS

This example demonstrates how to set the ACS sensitivity and how far its range extends.

3_INTRODUCTION_ACS_INTERRUPT.BAS

The ACS may be used in interrupt mode. This example shows how.

4_INTRODUCTION_DRIVE.BAS

The Robot is ready for a first excursion. A demonstration how to drive with ACS support shows the basics of controlling the motors and how collision warnings are transformed to evasive maneuvers.

4_INTRODUCTION_MOTIONSENSOR_ACS.BAS

How to cleverly use the ACS as a motion sensor.

4_INTRODUCTION_DISTANCESENSOR_ACS.BAS

This demonstration shows how to use the ACS to keep a constant distance to another object, to guard it so to speak.

5_INTRODUCTION_LIGHTSENSOR_1 to 3

Three examples of how to use the light sensor, e.g. as a motion sensor or to measure light intensity.

6_INTRODUCTION_DISTANCECOUNTER.BAS

One of the essential prerequisites of orientation in space is the possibility to measure distances.

7_INTRODUCTION_VOLTAGESENSOR.BAS

This example demonstrates the battery voltage sensor. It will give you information regarding its use and explains its importance.

8_INTRODUCTION_CURRENTSENSOR.BAS

9_INTRODUCTION_IR_COMM.BAS

Basics regarding the use of the infrared communication system.

9_INTRODUCTION_IR_COMM_INT.BAS

Basics regarding the use of the infrared communication system with interrupts.

9_INTRODUCTION_REMOTE_CONTROL.BAS

Shows how to use the communication system to control the Robot remotely.

Extended System Resources

One substantial difference of CCRP5 compared to a regular C-Control unit are the extended hardware resources, the SUBSYSTEM.

This is an additional 8-bit microprocessor (with fixed firmware) that handles the function units IRCOMM (infrared communication system), ACS (infrared anti collision system) and NAV (the distance measuring unit, used for navigation).

Compared to a regular C-Control unit, the CCRP5 has an additional 8 bit wide port at disposal available for the user's extensions. It supplies the function units POWERSWITCHES (to switch system voltages) and LED CONTROL (to switch the four LEDs).

The data communication between the C-Control and the SUBSYSTEM is handled via the SUBSYSTEM INTERFACE BUFFER, defined by the identifiers SUBCMD,HBYTE und LBYTE.

A special driver is responsible for data exchange between the C-Control and the SUBSYSTEM. This driver is written in machine language and resides in the EEPROM of the C-Control unit. Since some of the technical aspects are very complicated, you have a series of given BASIC system subroutines to facilitate the use of the SUBSYSTEM.

You may regard these subroutines as fixed part of the operating system, since without them essential functions of the Robot can not be used.

The system subroutines occupy the bytes 1 to 5, the remaining bytes are freely available to the user.

POWER SWITCHES

With the SUBSYSTEM switched off the current consumption is reduced from 35 mA to 12 mA. This function is important in case the Robot should stay inactive for extended periods of time and only use its sensors in short pulses.

Note that with the SUBSYSTEM switched off you can not use those functions any longer, access attempts may lead to errors or even to program crashes.

gosub SUBSYS_PWR_ON
gosub SUBSYS_PWR_OFF

ACS

These routines allow you to comfortably use the ACS functions:

gosub NO_ACS_INT ; Interrupt Mode off
gosub ACS_INT_200 ; Interrupt Mode on
SYS ACS_HI ; ACS HI POWER (Range approx. 60cm) on
SYS ACS_LO ; ACS LO POWER (Range approx. 30cm) on
SYS ACS_MAX ; ACS MAX POWER (Range approx. 100cm) on
SYS COMNAV_STATUS ; Interrogate the ACS/IR-COMM

Calling the COMNAV_STATUS subroutine is necessary whenever you want to get information regarding a collision warning or the reception of an infrared signal.

This routine returns information in byte 5 of SYSTEM_STATUS.

Bit 0 = left ACS Sensor
 Bit 1 = right ACS Sensor
 Bit 3 = IR-COMM receive flag

The bit is HIGH whenever the respective sensor has sensed an event or received data is ready to be read. The identifiers are called ACSL_F, ACSR_F and IR_F.

When ACS and IRCOMM are operated in interrupt mode, these flags hold the information which sensor triggered the interrupt. An interrupt may be reset by reading the status register SYS COMNAV_STATUS.

IRCOMM

Transmit/Receive RC5/REC80 Data

IRCOMM is able to transmit or receive data in both formats. The example demonstrates what the frame looks like for A=30 und C=22.

The RC5 IR Data Frame:

S	S	T	A4	A3	A2	A1	A0	C5	C4	C3	C2	C1	C0
1	1	x	1	1	1	1	0	0	1	0	1	1	0

----- 30 ----- 22 -----

S = Startbit
 T = Toggle (Will be toggled with each press of the remote control button)
 A = Address
 C = Command

The REC80 IR Data Frame:

S	A4	A3	A2	A1	A0	C6	C5	C4	C3	C2	C1	C0
1	1	1	1	1	0	0	0	1	0	1	1	0

----- 30 ----- 22 -----

S = Startbit
 A = Address
 C = Command

The following routines allow to access the IR communication system comfortably.

gosub GET_IRDATA ; supplies received IR Data in HBYTE (A) and LBYTE (C)
 (holds FF if nothing was received)
gosub SEND_IRDATA ; transmits Data in HBYTE (A) and LBYTE (C)
gosub REC80 ; sets the format
gosub REC80_INT ; activates interrupt mode in REC80 format
gosub RC5 ; sets the format
gosub RC5_INT ; activates interrupt mode in RC5 format
SYS COMNAV_STATUS ; queries ACS/IRCOMM (see ACS)

The formats allow following values for address and commands:

LBYTE Byte 1 (value 0..63 - RC5) (value 0...127 - REC 80)
 HBYTE Byte 2 (value 0..63 - RC5) (value 0.....63 - REC 80)

Accessing the IRCOMM, these prerequisites must be met:

- 1) Access may be delayed up to 20ms in case IRCOMM is just receiving or transmitting an infrared signal.
- 2) Any other times, the access will be handled instantaneously and finished within 3ms. During access no signal may be received and thus may be lost.
- 3) IRCOMM may be queried directly by gosub GET_IRDATA. In case the ACS is active, (which means frequent access to the subsystem via SYS COMNAV_STATUS), it is better to evaluate the FLAG in SYSTEM_STATUS (see ACS) and query the received code only when required.
- 4) IRCOMM as well as ACS are capable of generating interrupts. The interrupt will be reset by reading the status register with SYS COMNAV_STATUS. The result will be transferred to SYSTEM_STATUS and the Flags hold the information what triggered the interrupt (see ACS)

Transmitting Telemetry Data RC5

Analyzing the RC5 data format, one discovers that neither the ADDRESS nor the COMMAND has one full byte to be used. Therefore the system routines distribute the telemetry data in such a manner that it is formatted in c0 to a1. To be able to transmit multiple data channels, a2 to a4 hold the channel number, thus up to 16 channels are possible.

The result looks like this:

The modified RC5 IR Data Frame:

S	S	T	A4	A3	A2	A1	A0	C5	C4	C3	C2	C1	C0
1	1	CH3	CH2	CH1	CH0	b7	b6	b5	b4	b3	b2	b1	b0

- SYS SEND_TLM** ; formats and transmits the values in HBYTE and LBYTE
- SYS SEND_SYSSTAT** ; transmits SYSTEMSTATUS (fixed data channel 0)
- SYS SEND_SPEEDL** ; transmits the values of the PLM D/A-converter (fixed data channels 7 and 8)
- SYS SEND_SPEEDR**
- Gosub GET_TLM** ; reads telemetry data from IRCOMM and separates data channel and data

The data channel (0-15) is in the HBYTE whereas the actual measurement data is in the LBYTE (0-255). Calling SYS SEND_TLM formats the data as shown above and transmits them.

You may assign any sensors to the data channels 0 to 15. When processing and evaluating the telemetry data at the receiver side, of course this formatting has to be broken off again. The telemetry program examples supply a subroutine covering this aspect.

Addressed Mode

If you have multiple Robots operating that have to communicate with each other, you may assign an address to identify them within the network. For this purpose both of the formats are modified so that they hold a three-bit transmitter/receiver address. This way up to seven Robots may be identified. The remaining six bits in the frame allow for 64 commands.

The modified RC5 IR Data Frame:

S	S	T	A4	A3	A2	A1	A0	C5	C4	C3	C2	C1	C0
1	1	RX2	RX1	RX0	TX2	TX1	TX0	C5	C4	C3	C2	C1	C0

The modified REC80 IR Data Frame:

S	A4	A3	A2	A1	A0	C6	C5	C4	C3	C2	C1	C0
1	RX2	RX1	RX0	TX2	TX1	TX0	C5	C4	C3	C2	C1	C0

RX is the receiver address
 TX is the transmitter address

When activating this mode of operation, the IRCOMM automatically filters and disregards frames not addressed to itself. Data transferred to the subsystem have to be reformatted. There are a few subroutines to perform this task.

The RX-address is transferred in HBYTE, while commands are transferred in LBYTE. Gosub SEND_ADDRESSED_DATA transmits the data. When receiving a data frame, HBYTE holds the transmitter address and LBYTE contains the command.

Gosub SET_ADDRESS ; set addressed mode LBYTE =Address
gosub GET_ADDRESSED_DATA ; supplies received IR Data in HBYTE (TX) and LBYTE (C)
 (holds FF if nothing was received)
gosub SEND_ADDRESSED_DATA ; transmits data in HBYTE (RX) and LBYTE (C)
gosub REC80 ; sets the format
gosub REC80_INT ; activates interrupt mode in REC80 format
gosub RC5 ; sets the format
gosub RC5_INT ; activates interrupt mode in RC5 format
SYS COMNAV_STATUS ; queries ACS/IRCOMM (see ACS)

NAV

Measuring distances is one essential prerequisite of orientation and navigation.

You may address the NAV system with the following subroutines:

gosub CLR_DISTANCE clear both distance counters
gosub L_DISTANCE query left counter
gosub R_DISTANCE query right counter

After calling the system routine LBYTE and HBYTE contain the counter value, with a resolution of approximately 30mm. Since we are using a 16 bit counter, rollover occurs after approximately 2km.

LED CONTROL

To save you executing complicated bit manipulating operations, the LEDs may be switched by easy-to-handle drivers.

gosub LED1ON
gosub LED1OFF

The same goes for the LEDs 2 to 4.

DRIVE CONTROL

The speed of the Robot is set via the PLM outputs of the C-Control (DA1 und DA2) and does not require a dedicated driver. Setting the direction is quite comfortably managed with system routines, which also keeps the source code short.

SYS REVR ; reverse drive right track
SYS REVL ; reverse drive left track
SYS FWDR ; forward drive right track
SYS FWDL ; forward drive left track
SYS FWD ; forward drive both tracks
SYS REV ; reverse drive both tracks
SYS ROTR ; turn right on the spot
SYS ROTL ; turn left on the spot

CAUTION:

Always write this Initialization line as the first step in your programs

REV_L=on:REV_R=on:SYS PLM_SLOW

These commands initialize the ports for the direction control of the drive and the PLM frequency.

To be on the safe side, also initialize the ports if you do not plan to use the motors.

OPERATING THE MOTORS WITHOUT THIS INITIALIZATION LEADS TO CERTAIN DESTRUCTION OF THE MOTOR DRIVE COMPONENTS!

Operation at 12 Mhz (overclocking)

Optionally the CCRP5 may be operated with 12 Mhz system clock. To use this option, you have to set the jumper 5 before starting your program. Note that for loading a program you have to choose regular 4MHz operation in any case, to get the proper baud rate of the interface.

General

To allow for 12 MHz operation, parts of the system drivers for the SUBSYSTEM have to be slowed considerably, calling for a separate driver. Due to the limited capacity of the controller's EEPROM some of the system routines are no longer available and must be substituted for in BASIC.

The system routines SYS SEND_SPEEDR and SYS SEND_SPEEDL (telemetry data DA1 and DA2 of the motors) are no longer available. Moreover, the start addresses of the system routines change.

Please find some of the aforementioned example programs in the directory OVERCLOCKING. Load example 1_INTRODUCTION_LEDS.BAS first, because it loads the system driver and impressively demonstrates the speed increase.

Restrictions

CCRP5 has been designed and tested with the utmost of care. Though the processor itself runs with 12 MHz, Conrad Electronic can not guarantee correct functionality nor take liability for defects caused by overclocking the system.

Due to the overclocking all commands accessing the internal timer change properties.

The following is a list of the affected commands:

**BAUD
BEEP
PAUSE
FREQ
SECOND
MINUTE
HOUR
DOW
MONTH
YEAR**

All times are reduced to 1/3 of the original, which usually doesn't cause problems to account for when programming. Opposed to the small disadvantages is an enormous increase in processing speed. The execution of a BASIC command now takes only 300µs instead of 1ms.

Hint:

The 12MHz version of the driver (P5DRIV12.S19) is compatible to 4MHz system clock. This means that you may load and test your programs with 4MHz system clock, and switch to 12MHz only after everything runs satisfactorily. Note that accessing IRCOMM, NAV and ACS with 4MHz is approximately 1ms slower than when using the dedicated 4MHz driver (P5DRIV.S19).

C-Control I/O Resources

The C-Control computer supplies a great number of ports that are freely available at the M-Unit and Main-Unit, whereas with PROJECT5 some of those ports are necessary to operate the Robot and are therefore not available or only available in a limited fashion.

Used C-Control I/O Resources

The table shows an overview of the I/O channels that are fixed or switchable. Switchable channels may be used for your own applications only if you remove the relevant jumpers.

PORT P1	COMNAV-INTERFACE/ IOEXPANDER	DATA
PORT P2	COMNAV-INTERFACE /	CLOCK
PORT P3	IOEXPANDER	CLOCK
PORT P4	IOEXPANDER	STROBE
PORT P5	DRIVE CONTROL	DIRECTION A
PORT P6	DRIVE CONTROL	DIRECTION B
DA1	DRIVE CONTROL	SPEED B
DA2	DRIVE CONTROL	SPEED A
AD1	MOTOR CURRENT	
AD2	CHARGE CURRENT	
AD3	BATT VOLTAGE	
AD4	MIC	switchable JP 1
AD5	TOUCHSENSOR	switchable JP 2
AD6	LIGHTSENSOR LEFT	switchable JP 3
AD7	LIGHTSENSOR RIGHT	switchable JP 4
IRQ	SUBSYSTEM INTERRUPT	

Free C-Control I/O Resources

The following table shows an overview of the I/O channels that are not used by the system and are freely usable for extensions or may be used by accessories. These I/O-channels as well as other signals are accessible at the extension ports.

PORT P7
PORT P8
PORT P9
PORT P10
PORT P11
PORT P12
PORT P13
PORT P14
PORT P15
PORT P16
AD8
DCF/FREQ1
FREQ2

The CCBASIC Programming Language

System Resources of the C-Control Computer

By the term system resources all internal units which cannot be derived directly from microcontroller properties but are provided by the on-chip mask programmed operating system, are summarized. How those system resources can be addressed is described in the instruction overview further below.

Realtime Clock

In the background of the operating system there is a 16 bit timer, incremented every 20ms, that is the base of the internal real time clock and that can be read anytime to provide a time reference in BASIC programs. This clock can be synchronized by a DCF77 time receiver.

Time and date received by a DCF77 time receiver is transferred by the operating system into 7 internal RAM locations (divided in year, month, day, day of week, hour, minute, second) and is incremented in 20ms steps between each synchronization. The accuracy of the real time clock between synchronizations depends on

the deviation of the 4MHz crystal from the design frequency (up to 0.01 percent), from tolerances in the mass production and from the temperature.

This corresponds to a deviation of up to 0.36 seconds per hour. After applying power and after a reset the clock starts at the 01.01.97, 00:00:00 o'clock.

The internal date and time register can be read and written to by a BASIC program. Therefore, the clock can be set without using a DCF77 receiver by writing into the memory cells.

For testing purposes or low precision requirements you may omit the DCF77 antenna.

User Bytes

The microcontroller MC68HC05B6 provides 240 bytes of RAM altogether. The C_Control computer occupies most of it for operating system functions (stack, timer, clock, DCF77 frame buffer, interface buffer, buffer for calculations etc.). There are 24 bytes left to use in BASIC programs. CCRP5 requires the bytes 1 to 5 for the extended system resources and are not free to the user's disposal.

The utilization of these user bytes is described further below in the section DEFINE command.

Digital Ports

Using Digital Ports as an Input

Digital ports are used for interrogation of switching states. If a digital port is used as an input, its state is undefined as long as there is no external circuitry attached. To get a defined state, e.g. pullup resistors have to be used. If for example a reed contact is connected to this port, a logical one ("true" or "HIGH") is read from the port in open position, a logical zero ("false" or "LOW") in closed position.

Please note that depending on the external circuitry and the logical value required by your program, the value read on the port may have to be negated (NOT operator, see Instruction Reference).

Using Digital Ports as an Output

When using a digital port as an output, you may drive connected ICs, transistors or low-current LEDs directly. The maximum permissible load current is 10mA. Make sure to always use sufficient current limiting, e.g. a resistor, to avoid destruction of the microcontroller from excessive current.

Within the microcontroller, the definition of a digital port as an output or input is done with the first execution of a user program. After applying power or after executing a reset, all digital ports are initially set to input state and in the case of attached pullup resistors, are in HIGH state.

Analog Ports

A/D-Converter

The C-Control/BASIC control computer provides eight A/D ports and two D/A converters. Before using an A/D input, a reference voltage has to be applied to the reference voltage input of the device. The applied voltage is the effective maximum of the measuring range of the A/D converters and corresponds to a conversion value of 255 (\$FF in hexadecimal notation).

Your Robot has a fixed reference voltage of 2.50 Volts. The lower end of the A/D conversion range is always the ground (sometimes also referred to as "minus") of the supply voltage.

You may attach sensors of any kind to the A/D inputs, as long as they provide an analog output voltage with an amplitude range of 0 to 2.5 Volts. In most cases, active sensors are applied to amplify the signal of the actual sensor element and to accommodate for resolution, linearity and drift requirements.

D/A Converter

Both of the 8 bit D/A converters are implemented as pulse width modulation outputs. Within a fixed time period (modulation interval), divided into 256 slices, the D/A output is high-pulsed for as many slices as corresponds to the 8 bit value specified for the output. The remainder of the slices are low-pulsed. The duration of one slice is 2µs, thus the duration of the complete modulation interval is 512µs (this equates a frequency of 1953Hz).

To demodulate this signal, that is, to obtain a real analog signal, a simple RC network is generally sufficient.

The Robot uses the D/A converters, in this case better referred to as PWM outputs, to switch the drive motor voltage in a highly efficient manner.

Active Antenna

A DCF77 active antenna can be connected to the C-Control/BASIC control computer via a special port at the header 2 (DCF/Freq1).

The antenna has to provide an open collector output switched to ground, that outputs the received signal. Always use shielded cable to attach the antenna, otherwise it will pick up disturbances, especially when using long cables.

Note that unfortunately DCF77 reception is not possible while operating the motors.

Frequency In/Outputs

Measuring Frequency at pin FREQ2

Pin FREQ2 allows measuring the frequency of an applied signal, provided it is a square wave of CMOS/TTL compatible value. The maximum detectable frequency extends up to 32000Hz.

Tonausgabe am BEEP-Pin

The Robot is capable of generating sound by attaching a piezoelectric sound transducer (passive transducer, no internal electronics required) to the BEEP pin. By executing the BEEP command (see below) it outputs a square wave of 5 Volts amplitude.

Interrupt Input

The Interrupt Input IRQ

The IRQ pin on the mainboard of the Robot has a 10k pullup resistor attached and is connected to the IRCOMM/NAV subsystem. On detection of a LOW edge at the IRQ pin, the execution of a BASIC program is interrupted and continued at a user-defined label.

Expansion Connector

All of the usable port lines as well as a few system signals of the control computer are accessible on two twenty-pin headers. The appendix contains a table, covering the association of the pin numbers and the signal names.

Programming the C-Control Computer

CCBASIC is a special BASIC dialect used to program the C-Control BASIC control computer. The syntax is very close to Standard BASIC. A few commands however are different or have extensions to specifically fit the hardware of the control computer.

The CC-BASIC Introduction Examples

There are some simple examples to be found on the CD to guide you step by step through the BASIC commands.

Most of these examples require the use of the "Hyperterminal", to get a connection to the PC. The Hyperterminal is part of your Windows Operating System and is to be found at ->Programs ->Accessories ->Communication. Start the C-CONTROL IDE and load one of the BASIC examples from the directory INTRODUCTION_CCBASIC. Once you are ready to transfer it to the Robot, you need to close the Hyperterminal to release the serial port interface. Configure the Hyperterminal with 9600 Baud, 8n1, to prepare it to display messages from the C_Control computer.

What is a Program?

A program is the description of how to process information that requires a reaction or result. During such a process, a number of output values are calculated from a number of variable or constant input values. Either the output values themselves are the desired information acquisition or they are used as a direct reaction to the input values. Besides actual calculations, a program may contain instructions for hardware access or program flow control.

A BASIC program consists of several lines of so-called source code. Each line contains one or more calculation or flow control instructions. The instruction sequence essentially determines the way of information processing. The execution of operations in the control computer is done sequentially, meaning one after another. A sequence of specific program instructions may also be called an algorithm.

The object of the information processing sequence is called data, it represents the stored information. The C-Control BASIC computer exclusively stores and processes non-fractional numerical data, so-called integer numbers of 1, 8, or 16 bit length. A variable of 8 bit (one byte) can only store positive values of 0 to 255. The set of values of an integer variable of 16 bit (one word) ranges from -32768 to +32767.

Always make sure that a given calculation does not exceed those limits so as not to experience under- or overflows.

$a = 255 + 1$

results in 0, not the expected 256, if a represents just one byte!

$a = -32768 - 1$

Results in 32767, not the expected -32769, if a represents one word!

Basics of the CCBASIC Programming Language

General Topics

A BASIC program consists of several lines of so called source code. One line contains one or more calculations or control statements. Each program line contains one or more instructions, separated by colons (:). Line numbers, as used in older BASIC dialects, are not necessary. However if you specify line numbers, they could be used as jump targets.

10 . . . GOTO 10

The numbers have no influence on the sequence of program operations. If e.g. a line with number 200 is followed by a line with number 100, line 200 is processed before line 100 nevertheless.

Comments may be entered in the source code to explain the written program. They enhance its readability and its service friendliness. A comment in CCBASIC always begins with a single quotation mark (') and declares that the rest of the line does not belong to the program.

a = a + c ' . . . your comment . . .

Identifiers

Identifiers are program elements of alphanumeric characters (A to Z, 0 to 9) that identify objects like variables and constants determined by the programmer.

Label names and so called "keywords" are also identifiers.

No distinction is made between upper and lower case characters. An identifier always begins with a character or an underscore. Spaces within an identifier are not allowed.

Variables and Constants

Variables and Constants are objects of the information processing process.

In CCBASIC both store a numerical value. Whereas the value of a constant is specified once and then remains unchanged, the value of a variable can change unrestricted often during program execution.

Constants in CCBASIC may be specified in decimal, hexadecimal and binary notation.

The syntax of hexadecimal and binary numbers is illustrated here by the number 46 (decimal):

&H2E &B101110

In addition to that, symbolic constants may be declared via DEFINE lines (see below).

Variables are accessed always by their identifiers. This identifier has to be defined in a DEFINE line in the program, before the first assignment of the variable.

Labels

Labels mark special points within the sequence of program operations. Labels are targets or jump operations within an algorithm. In CCBASIC labels are located on the beginning of a line and begin always with a hash sign (#) followed by the identifier without spaces.

The example shows the definition of the label "label1" and the assignment within a jump statement:

#label1 . . . GOTO label1

Terms

A term at once evaluates to a certain value, as variable or constant or by calculation.

Terms are parts of instructions and are placed on the right side of the assignment character (=) when assigning a value to a variable.

a +b (ABS(x) - 13) * 10

Operands und Operators

An operand in its basic form is either a constant, a variable or a function call, but it can also be a composite term of operands and operators in itself again.

Operators identify calculation operations that have to be executed together with surrounding operands. Therefore, there is a defined hierarchical operator order (see Instruction Reference), which determines the sequence of calculations.

Functions

A functions performs a predefined operation – e.g. a calculation – and returns a result value by its call. Most of the functions expect one or more arguments, which are passed in round brackets "()" after the function identifier and are separated by commas. Some functions are called without arguments. In that case, no round brackets are entered.

ABS (x)
MAX(a,b)
RAND
EOF

In CCBASIC, all supported functions are predefined. Their identifiers belong to the keywords. Formulating user defined functions is not provided in CCBASIC.

Assignments

The assignment is the most simple program instruction. After the identifier of a variable whose value should be assigned, the assignment character and a term which determines the value to be assigned, follow. Thus, an assignment corresponds to a simple mathematical formula.

a = 10
b =x -y
c = SQR(a*a + b*b)

Instructions

Beside the normal assignments, instructions are statements to execute program operations by the C-Control/BASIC control computer. Instructions always begin with a keyword. Some instructions expect one or more parameters to exactly specify the program operation to be executed. Those parameters are listed after the instruction identifier followed by a space and are separated by commas (exception PRINT, see Instruction Overview). In contrast to function arguments, the instruction parameters are not surrounded by round brackets!

RANDOMIZE
PAUSE 100
BEEP 440,50,50

Instructions for Program Flow Control

Those instructions allow to control the order of the normally strictly sequentially processed program operations and allow to adapt them to input values of the information processing process. They provide high flexibility in algorithm formulation and they are yet basic prerequisites for solving some application problems. Instructions to control the program flow consist of one or more keywords and may or may not require in a special way further statements.

GOTO label1
IF a > b THEN GOSUBlabel2
FOR i = 0 TO 10 STEP 2
.. _
NEXT

Compiler Instructions

In addition to the program instructions a CCBASIC source code contains compiler instructions, which are used to e.g. define a data block (table), a variable or a constant.

For compiler instructions, the colon rule to separate several instructions in one line is not valid. There must be only one compiler instruction per line.

The DEFINE keyword is a Compiler Instruction.

Defining Symbolic Constants

It is good programming style to use symbolic constants rather than "magic" numbers within the program.

IF x > 1234 THEN GOTO alarm

By binding significant identifiers to constants, the readability of source code is increased. By defining all constants globally, a program becomes easier to maintain. This holds especially true if the same constant is needed several times within the program.

The definition of symbolic constants is done as follows:

DEFINE identifier value

Thus "value" is either a decimal, hexadecimal or binary number. The example above should better read:

DEFINE limit 1234

...

IF x > limit THEN GOTO alarm

Defining Variables

The C-Control/BASIC control computer provides 24 byte memory cells of its internal memory (RAM) for usage in the program. All variables of the BASIC program are stored in this memory area. The 24 bytes can be used also bit by bit or as 16 bit long integers (word), on demand.

In contrast to standard BASIC, in CCBASIC every variable, that is used in the program, has to be defined before its first use. Therefore you have to specify the data type (bit, byte or word) and you can (it's a must for bits!) declare a memory cell number. The user himself must pay attention on unwanted memory overlaps when

assigning memory, because otherwise it could end in mutual variable overwriting. For example bit[18], byte[2] and word[1] each occupy a part of cell 2 in the memory area.

• Defining a Bit Variable:

DEFINE identifier BIT [nr]

For nr values from 1 to 192 (24 bytes with 8 bits each) are permissible.

+ Defining a byte variable with cell number:

DEFINE identifier BYTE [nr]

For nr values from 1 to 24 (24 bytes) are permissible.

+ Defining an integer variable with cell number:

DEFINE identifier WORD [nr]

Fro nr values from 1 to 12 (one word occupies 2 bytes) are permitted.

If you omit the cell number nr in the definition of bytes or words, the compiler does all of the memory partitioning. Take care not to define bytes and words alternately. The following statements

```
DEFINEa BYTE  
DEFINE b WORD  
DEFINE c BYTE  
DEFINE d WORD
```

result in two unused (donated, precious) bytes, between a and b as well as between c and d, because words are always oriented principally to uneven bytes 1,3,5,7, ... etc.

It should better read

```
DEFINE b WORD  
DEFINE d WORD  
DEFINE a BYTE  
DEFINE c BYTE
```

The automatic memory partitioning of variables by the compiler begins at line number 1. The example above (the better one) occupies 6 bytes. If you define further bits, bytes and words and specify the cell number, pay attention on unwanted overlaps once more. An already defined variable identifier must not be defined a second time.

Defining Digital Ports

In CCBASIC, ports and variables are accessed in the same way. Here again, every port that is to be used has to be defined beforehand.

- **Defining a 16 bit Digital Port:**

DEFINE identifier PORT[nr]

- **Defining an 8 bit Digital Port:**

DEFINE identifier BYTEPORT[nr]

For nr permissible values are 1 (ports 1 through 8 as byteport) and 2 (ports 9 to 16).

- **Defining an identifier for common access to all 16 Digital Ports as one 16 bit Port:**

DEFINE identifier WORDPORT[nr]

Permissible value for nr is 1 only.

Defining Analog Ports

CCBASIC accesses ports the same way as variables. Each port that is to be used has to be predefined.

- **Defining one of the 8 A/D Ports:**

DEFINE identifier AD[nr]

Permissible values for nr are 1 to 8.

- **Defining one of the 2 D/A Ports:**

DEFINE identifier DA[nr]

For nr values 1 or 2 are permissible.

Mathematical and Logical Operators

This chapter gives a complete overview of CCBASIC operators, functions and instructions.

I Basic Calculations: + - * /

- **The modulo operator MOD returns the remainder of an integer division:**

a = 10 MOD 3

results in the value 1 for the variable a.

- **Comparison Operators:**

> (greater than), < (less than), >= (greater than or equal), <= (less than or equal), = (equal), <> (not equal)
The result of a compare operation is either -1 or 255 (comparison true) or 0 (comparison false).

a=10 <3

results in the value 0 for the variable a.

- **Logical Operators:**

NOT (negation), AND (AND combination), NAND (AND combination with following negation), OR (OR combination), NOR (OR combination with following negation), XOR (EXCLUSIVE - OR combination). Besides the formulation of conditions (mostly together with compare operations), logical operators can be used for binary byte or word manipulation.

• **Shift Operators:**

SHL (shift left), SHR (shift right) are used in order to bit by bit shift bit patterns in byte or word variables. On the left hand side of the operator the value to shift is located, on the right hand side, the number of bits to shift. Every single left shift meets a multiplication by 2, a right shift meets a division by 2.

a = 10 SHL 3 equates: $a = 10 * 2 * 2 * 2$ and results in the value 80 for the variable a

Mathematical Functions and Instructions

The arguments x and y, according to the function or instruction, always are terms (See definition above).

• **The Root Function SQR(x)**

returns an approximation of the square root of the argument x. Thereby the places after the decimal point are truncated.

• **The Signum Function SGN(x)**

returns 1, if the value of the argument is greater than 0, and returns -1, if the value is less than 0. For x = 0 the result of the SGN function is 0.

• **Die Maximum Function MAX(x,y)**

returns x, if $x > y$ and vice versa.

• **Die Minimum Function MIN(x,y)**

returns x, if $x < y$ and vice versa.

• **The RANDOMIZE x**

command initiates the control computer's internal pseudo random generator by the value of x. The same initiate value always leads to an identical sequence of numbers. The special form RANDOMIZE loads the value of the freewheeling timer into the generator.

• **The Random Function RAND**

returns the next integer value of the pseudo random generator. The random numbers are generated with the multiplicative method and a following modulo division (see a good mathematics book) from the previous value.

Precedence of Operators and Function Calls

When calculating terms by the help of operators and functions, their precedence is of great significance. Partial terms with high hierarchy level operators are calculated before those of a lower hierarchy level (compared to calculation rule: "point precedes line"). If you have operators of the same hierarchy level, calculation is done from left to right. You can take influence on the calculation sequence using parentheses according to general mathematics rules. CCBASICS supports a maximum of 3 parentheses levels. In the interest of program readability, "wild" parenthese expressions should be avoided and complex calculations should be split over several BASIC lines.

The following list shows the operator precedence in CCBASIC:

Hierarchy Level	Operators
9	()
8	Function Calls
7	Negative Sign
6	* / MOD SHL SHR
5	+ -
4	> >= < <= = <>
3	NOT
2	AND NAND
1	OR NOR XOR

Program Flow Control Statements

I Loop

FOR variable = start TO end STEP step

...

NEXT

The FOR loop continues to execute the instructions until NEXT is reached until the value of the variable is equal to the value of the end term. Before the first pass is done, the value of the start term is calculated and assigned to the loop variable. With every loop, the value of the step term is added to the loop variable. In the following form

```
FOR variable = start TO end STEP  
...  
NEXT
```

step always is assumed 1. The values of the end and step term are recalculated with every loop passed. This permits an extended program flow control. FOR loops can be nested. The nesting depth is only limited by the necessary for the loop variables.

```
FOR variable = start TO end1  
  FOR variable = start TO end2  
    FOR variable = start TO end3  
      .....  
    NEXT  
  NEXT  
NEXT
```

Each FOR loop must only loop through its own NEXT statement. The following source code snippet can be compiled and loaded into the control computer, but it will not work as you might expect:

```
FOR v1 = start1 TO end1  
...  
  GOTO anothernext
```

```
...  
NEXT  
FOR v2 = start2 TO end2  
...  
#anothernext  
NEXT
```

Moreover, pay attention on the value range of loop variable and end term!
The following statement

```
DEFINE v BYTE  
FOR v = 1 TO 1000  
...  
NEXT
```

results in an endless loop, because v as byte variable can never reach the value 1000, but rolls over to 0 after 255.

- **Conditional Execution**

```
IF condition THEN statement1  
or  
IF condition THEN statement1 ELSE statement2
```

The IF ... THEN ... ELSE construction enables program flow adaptation to runtime conditions. As condition, any term can be inserted. If its calculation returns a value other than zero, the condition is true and statement1 is executed. If an ELSE branch with a second statement is specified, this statement is executed alternatively, if the calculated term results in a value of zero.

The whole IF ... THEN ... ELSE construction must be placed in one line of source code. Instruction blocks (several instructions) after THEN and ELSE are not permitted.

- **GOTO statement**

```
GOTO label
```

The GOTO statement forces the controller to continue program execution at a predefined location. As jump target a label identifier is specified. The jump target can be located before or after the GOTO statement within the source code.

- **Call and return from a subroutine**

A subroutine call is done with the statement:

GOSUB label

That is, label is the start point of the subroutine. In the so called subroutines, program sections are summarized, which are needed several times during program execution. A subroutine always begins with a label, consists of one or more statements and finally ends with RETURN

After RETURN, program execution continues with the statement found immediately after the GOSUB instruction. Program execution must never reach a RETURN statement without a preceding GOSUB statement. The maximum allowable nesting depth of subroutine calls is four.

```
#mainprog
GOSUB sub1
...
#sub1
GOSUB sub2
...
RETURN
#sub2
GOSUB sub3
...
RETURN
#sub3
GOSUB sub4
...
RETURN
#sub4
...
RETURN
```

- **Value controlled program branch**

ON variable GOTO label0, label1, . ..labeln

or

ON variable GOSUB label0, label1, . ..labeln

Depending on the value of the selector variable, a program branch or a subroutine call to the listed jump targets is performed. If the value of the variable is 0, a branch to label0 is performed, if it is 1, a branch to label1 is performed etc. If the variable value is negative or greater than the number of jump targets listed, program execution continues without branching.

- **Program branching after Interrupt Signal at pin IRQ**

On detection of an Interrupt signal (LOW edge) at the IRQ pin, BASIC program execution will be interrupted after the current instruction is completely executed and will be continued at the predefined

INTERRUPT label

Returning to the previously left BASIC instruction is done with the instruction

RETURN INTERRUPT

The following example toggles a LED with each received interrupt signal:

```

DEFINE led PORT[8]
led = OFF
' define the interrupt routine
INTERRUPT switch_it
' endless loop
#loop
GOTO loop
' interrupt routine
#switch_it
tog led
RETURN INTERRUPT

```

Caution:

Due to an error in the operating system problems may arise when using the instruction "ON (variable) GOSUB (label)" with interrupts. Preferably use "ON (variable) GOTO (label)" instead!

I End of Program

END

If during program execution the control computer reaches the END statement, program execution is terminated. The system remains in an inactive mode. Now, a new user program can be transferred or execution can be restarted by pressing the start button.

• **Program flow delay**

The statement

WAIT conditionterm

interrupts program processing until the calculation of conditionterm returns any value other than zero.

DEFINE key port[9]

```

...
WAIT key

```

In this example the program waits until a high level (logical 1) is detected on digital port 9. The PAUSE instruction interrupts program execution for a certain time. The calculated value of the parameter term affects the pause time as a multiplication factor with a basic unit of 20 milliseconds.

PAUSE term

For example the statement

PAUSE 50

interrupts program execution for about $50 * 20$ milliseconds (= 1 second). The maximum time deviation of the actual pause from the value specified is +20 milliseconds as a matter of principle.

Serial Interface Communication

• **Data Output**

Data output is done as text over the serial interface of the C-Control/BASIC control computer. If e.g. a PC is connected with a terminal program by an interface cable, data output can be displayed there.

PRINT term

outputs the result of the calculation of term.

PRINT "text"

transfers the text within quotation marks.

In both cases a line feed character is appended to the transfer, which causes the terminal program to show the next output in the next line of the screen. The line feed can be suppressed by adding a semicolon to the PRINT

command after the parameter (term or "text").

PRINT term;

Or:

PRINT "text";

Moreover CCBASIC supports several outputs by one PRINT command with the single parameters separated by semi-colon or comma. A comma inserts a tabulator character into the output, which is displayed as a number of spaces on the screen according to the terminal program settings. If two outputs shall follow one after another without space, they have to be separated by a semicolon within the PRINT command.

PRINT "a= ", a
PRINT "a= "; a

A single PRINT command without any parameters outputs just a line feed.

PRINT

• Data Input

The command

INPUT variable

reads an integer value from the serial interface and stores it in a variable for further processing. The value is entered into a terminal program of a PC and after pressing the ENTER key, it is transferred to the C-Control/BASIC control computer over an interface cable. The INPUT command waits until a complete data transfer from the terminal is received. If the INPUT command is called without data transfer from the terminal, the program will wait forever at this point! In this case, only the Reset Key and a subsequent restart of the C-Control/BASIC device will help.

• Byte by byte communication over the serial interface

Whereas PRINT and INPUT send and expect short strings to represent a numerical value, it could be desirable to transfer single bytes over the serial interface. For that reason, CCBASIC provides commands like PUT and GET.

PUT term

sends the calculated value of a term. If necessary, the result is truncated to the byte range (0...255) beforehand..

GET variable

waits for a received byte and then stores the value into the variable specified.

• Further interface commands and functions

As described, INPUT and GET may wait forever for the reception of serial data under certain circumstances. In order to avoid such a pending program, you can check whether or not received data is provided, by calling the status function RXD before every INPUT or GET command. If there is data, the function returns a value of -1. If the interface buffer is empty the function returns a value of 0.

if RXD then GET thebyte

The default transfer rate of the serial interface is 9600 bits per second (baud) for transmitter and receiver. The BAUD command can be used to set other rates, too. CCBASIC has the predefined constants R1200, R2400, R4800, R9600 to specify rates from 1200 to 9600 bits per second.

BAUD R2400

e.g. switches sender and receiver to a rate of 2400 bits per second. Other than the predefined rates are possible, even different rates for transmitter and receiver. The transfer rate of the serial interface is a derivate from the internal clock of the C-Control/BASIC control computer's microprocessor. The byte value to hand over to the BAUD command contains the necessary divider values Nxx.

b7	b6	b5	b4	b3	b2	b1	b0
NP1	NP0	NT2	NT1	NT0	NR2	NR1	NR0

The bits 7 and 6 contain a common divider NP for sender and receiver. NP can adopt the values 1, 3, 4 and 13. The following table shows the necessary settings for NP1 and NP0:

PRESCALER	NP1	NP0
1	0	0
3	0	1
4	1	0
13	1	1

NT (bit 3 to 5) and NR (bit 0 to 2) determine further divider values, separated for transmitter (NT) and receiver (NR), according to the following table:

DIVIDER	NT2 / NR2	NT1/ NR1	NT0 / NR0
1	0	0	0
2	0	0	1
4	0	1	0
8	0	1	1
16	1	0	0
32	1	0	1
64	1	1	0
128	1	1	1

The transmitter's transfer rate calculates according to the following formula:

$$\text{Transmit Rate} = 125000 / (\text{NP} * \text{NT})$$

The receiver's transfer rate calculates according to the following formula:

$$\text{Receive Rate} = 125000 / (\text{NP} * \text{NR}).$$

All other interface parameters (like 8 data bits, no parity bit, 1 stop bit) are fixed and cannot be varied. The prepared handshake channels for the CTS and RTS signals on the C-ControlBASIC computer are disused in the current version and cannot be addressed from CCBASIC programmes.

File Functions

The file functions allow recording measuring values or other data or can be used to store information to reload into the program variables after losing supply voltage. For that purpose the memory area of the EEPROM chip after the user program, which uses the biggest part in most cases, is provided. The memory area is managed as a file which can be accessed for reading and writing after it was opened with the according attribute. The command to open a file reads as follows:

OPEN# FOR WRITE

OPEN# FOR APPEND

OPEN# FOR READ

This is, WRITE means open a file for writing by possibly overwriting old records, APPEND means open for writing by appending new to old records and READ means open for reading records. Only integer values can be saved and read. Each value occupies 2 bytes of the EEPROM. Writing and reading is done using the commands

PRINT# term

where the calculated result of the term is stored, and

INPUT# variable

where variable identifies a defined integer variable in the program. Writing to and reading from the file is done strictly sequential. Therefore, an internal file pointer is held and incremented by 1 after each access. Before any writing action is taken, you should check whether there is enough EEPROM space to store the data. Therefore the FILEFREE function can be called which returns the amount of free memory (in words). The following example illustrates the use of that function:

```

DEFINE a WORD
DEFINE b WORD
DEFINE c WORD
DEFINE blocksize 3
...
IF FILEFREE >= blocksize THEN GOSUB writeblock
...
#writeblock
PRINT# a
PRINT# b
PRINT# c
RETURN

```

Before any reading action is taken, you should check whether there is any further data stored. The according function is EOF ("end of file"). It returns a value of -1, if there is no further data available, otherwise it returns a value of 0. The EOF function framework should be the same as for writing data. According to the example above, the following code would be required:

```

IF NOT EOF THEN GOSUB readblock
...
#readblock
INPUT# a
INPUT# b
INPUT# c
RETURN

```

After accessing a file, it should be closed at once. Only then data is safe from power loss or system reset. Therefore, the command is

CLOSE#

It does not have any parameters.

Port Commands

- **the Toggle command TOG**

The ports of the control computer can be accessed in the same way as variables. To switch on a digital port you write

p=I and **p=O** in order to switch it off.

In order to toggle a port (On -> Off; Off -> On) you may write

p = NOT p

or use the

TOG p

command. The TOG command uses less EEPROM memory and is executed faster than the classical "NOT p" construction. The port variable p must identify only a single digital port, not a byte or a word port.

- **Deactivating a port using DEACT**

As soon as port variable is assigned a value for the first time, the control computer puts the according hardware structures of the processor chip (transistors) into output operation. So there is a current flow from or to the processor, according to the connected circuitry (maximum value of 10mA allowable!). The command

DEACT portvar

deactivates the specified port. That means, the port is switched into a high impedance state and operates as input. The DEACT command can be used for single digital ports or for byte ports.

- **the PULSE command**

The command

PULSE portvar

outputs a pulse lasting a few milliseconds, at the port identified by portvar.

This is useful e.g. to switch external connected edge-triggered logic circuits. If the port is low (=0) before executing the PULSE command, a high pulse (0-1- 0) otherwise a low pulse (1-0-1) is output.

Using the PULSE command, the port variable must identify only one single digital port, not a byte or word port.

Defining and Using Data Tables

In standard BASIC, DATA rows are used to store constant data blocks which then can be accessed sequentially. CCBASIC does not support DATA rows but provides a much more flexible tool to define and access data blocks. Constant data can be stored as tables. Each table is assigned an identifier (tablename) and can contain

any number of entries, assuming there is enough program memory available. Each data entry (Cx) is stored as integer value and therefore needs 2 bytes. With it, data can be listed directly within the source code

TABLE tablename C0 C1 C2 C3 ...

c4 c5 . . . -.

. . Cn

TABEND

or can be imported from an external text file:

TABLE tablename "tabfilename"

Table definitions must be located at the end of the program after the END instruction, because data is stored in the EEPROM memory chip directly after the preceding code bytes. Program execution must never reach any table data, because otherwise data would be interpreted as a BASIC instruction which would lead to system failure in any case.

Table data can be accessed using the command

LOOKTAB tablename,index,variable

tablename identifies a valid table, index can be any term and variable identifies the memory cell to store the result in. The calculated value of the index term must not be negative and can range from 0 to a maximum of N-1, assuming the tables has N entries. If index results in a value of 0, C0 is stored in the variable specified. If index results in a value of 1, C1 is stored in that variable and so on. The following example outputs the contents of a table serially

DEFINE value WORD

DEFINE i BYTE

FOR i = 0 to 3

LOOKTAB mytab,i,value

PRINT "mytab["; i; "]="; value

NEXT

END

TABLE mytab 12 -20 0 1000

TABEND

On the terminal screen should appear

mytab[0]=12

mytab[1]=-20

mytab[2]=0

mytab[3]=1000

Tables prove to be especially useful when transforming A/D values into real physical values. One transformation table then, normally has 256 entries. The A/D value measured is used as table index to determine the physical value.

Accessing the Real Time Clock

In order to read and set the internal real time clock, the following variables are defined:

YEAR year (0...99)

MONTH month (1...12)

DAY day of month (1...31)

DOW day of week (0=sunday...6=saturday)

HOUR hour (0...23)

MINUTE minute (0...59)

SECOND second (0...59)

Please note that the internal clock keeps running during access. Therefore the value of the seconds should always be read first. If that value is 59, you must, after reading the last time information (e.g. year), read it once more and test it for 0. In that case, you must repeat the whole procedure because a new minute has begun

(extreme case: new year's eve having all digits of time and date switched). The year number only is stored as a two place number in the C-Control system.

Note:

Due to an error in the operating system DOW counts up to 7. When running a BASIC program that relies on the DOW date, you may correct this error by testing the DOW register:

```
IF DOW > 6 THEN DOW = 0
```

insert this line before accessing the DOW register.

Internal Timer, Sound Generation, Frequency Measurement

• **Timer**

The internal 20 millisecond timer can be read using the predefined identifier **TIMER**. The timer is freerunning and cannot be set or reset.

• **Sound Output using BEEP**

The C-Control/BASIC control computer is able to output sound as a square wave signal on one of its pins (BEEP pin accords to processor output TCMPI). The command is

BEEP tone, tTone, tPause

The three parameters may be either constants or terms. With it, tone determines the sound pitch with the formula

$$\text{tone} = 250000 / \text{freq [Hz]}$$

The duration of the tone is determined by tTone and tPause. The latter determines the pause between each tone. The unit for time data is 20 milliseconds. The command

BEEP 568, 10, 3

outputs a sound for the time of $10 \times 20 = 200$ milliseconds of about 440 Hz (diapason A) and then waits $3 \times 20 = 60$ milliseconds. If there is a BEEP with no BEEP following, tPause can also be set to 0. If 0 is specified for tTone, a permanent sound is generated. The sound generator switches on the sound and then continues to process the BASIC program. With a tone value of 0, the sound generator can be turned off again.

• **Measuring Frequencies using FREQ**

If there is no active antenna connected to the DCF77 input, it can be used to measure frequency alternatively. The result can be queried with the FREQ function at any time.

x = FREQ

The frequency measurement is based on the pulse count principle with a gate time of one second. Measurement is done permanently in the background, while the BASIC program is processed. The pulses counted every second, exactly correspond to the frequency in Hz.

The maximum detectable frequency extends up to 5 kHz with a measuring error of less than one percent. Above that, the results become more and more inexact.

• Power Save Mode using SLOWMODE

Applications that do not need high processing power, can slow down (1/16) the internal microprocessor clock using the command

SLOWMODE ON

den internen Takt des Mikroprozessors verlangsamen (1/16). In conjunction with the switchable SUBSYSTEM

the power consumption may be reduced significantly.

In case the further execution of the BASIC program requires a higher speed again, the command

SLOWMODE OFF

Reestablishes the default speed.

Attention: Programs that use serial data transfer should not activate SLOWMODE, because the preset transfer rates are reduced together with the processor clock

Embedding Machine Language Programs

The subsequent information is aimed at professional C-Control/BASIC computer users and are not required for regular BASIC programming, since CCRP5 entirely uses the resources provided for machine language programs. As a prerequisite, knowledge of the internal structure of the microcontroller MC68HC05B6 as well as knowledge about assembler programming of this controller is necessary. Furthermore a crossassembler for 68HC05 controllers is required.

There is no doubt that most application tasks may be solved by exclusively using BASIC in a program. Still, in some cases a very special task may require high processing speed or special hardware access. For that purpose there are another 255 bytes in the EEPROM reserved for assembler programmed subroutines. Those routines may be called from BASIC with the command

SYS adr

where adr is a constant determining the jump address, e.g. &HI0I, where the internal EEPROM area starts at. Thus, the assembler code has to be placed at &HI0I via the ORG command. Returning from the assembler routine back to the BASIC program is done via the RTS command. Data exchange between BASIC and assembler routine can be done using the RAM addresses listed in the file SYSADR.INC.

How does the assembler code get into the C-Control/BASIC control computer?

The additional assembler code routines are stored in a separate source code file (e.g. ADDONS.ASM). Then, the assembler is called to generate an object code file in the S19 format (e.g. ADDONS.SI9). Therefore, read the documentation of the assembler provided to you. Your BASIC program, which contains the SYS instruction, must embed the generated code using the command

SYS CODE "ADDONS.SI9"

The SYSCODE command must appear only once within the BASIC program and should be located at the end, even after table definitions.

Troubleshooting

Programs can not be loaded

- In the IDE, you chose the wrong serial port
- Wrong polarity of the interface plug on the Robot
- No or discharged batteries in the battery holder or batteries disconnected
- Robot switched off or fuse blown
- The Jumper is not in the 4MHz position, or missing altogether
- The RESET button has not been depressed

A program was loaded, but it doesn't start

- The START button has not been depressed
- The system driver has not been loaded (see "A first functionality test")
- A permanent, uncontrolled RESET is being executed

The Roboter executes uncontrolled RESETs during operation

- The batteries are discharged or of bad quality (source resistance too high)
- Bad or worn contacts within the battery holder or battery connector
- The notes in "4_EINFÜHRUNG_ANTRIEB.BAS" have not been taken into account

Mysterious disturbances in interrupt mode

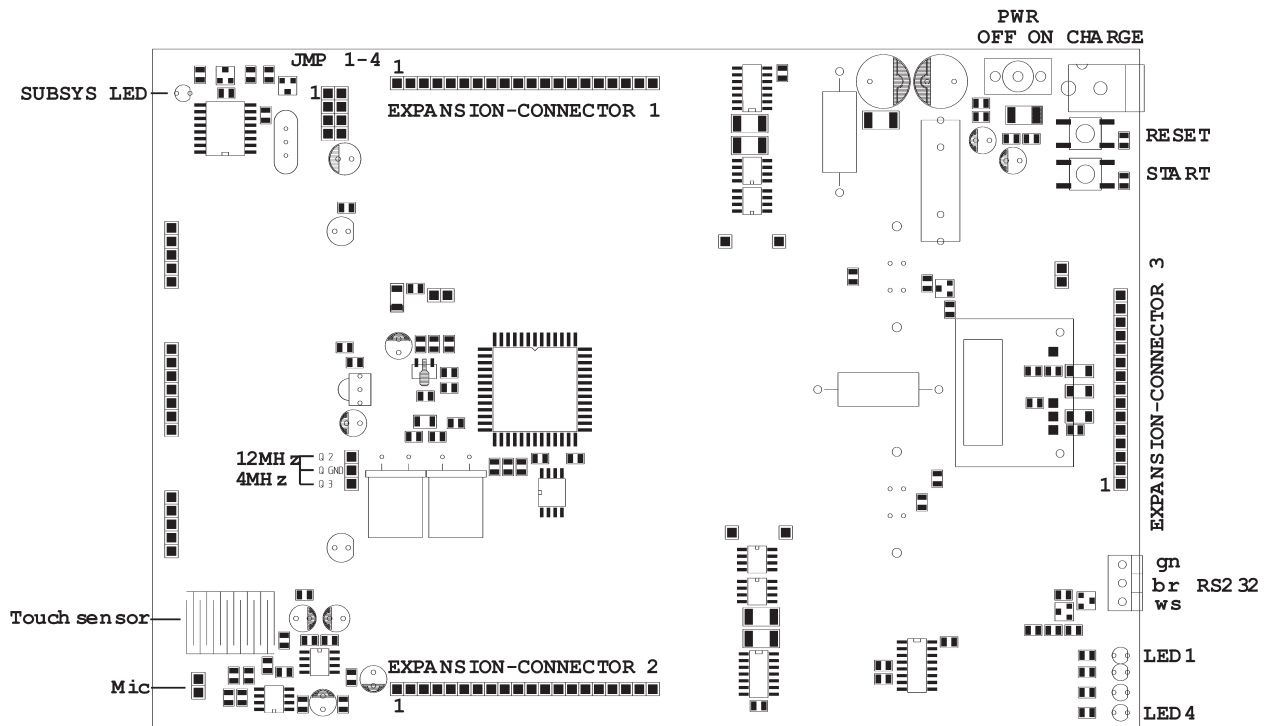
- The notes in "3_EINFÜHRUNG_ACS_INT.BAS" have not been taken into account

Various sensors read wrong or no values

- Jumpers JMP 1 to 4 missing

Operating Elements, Display Elements and Connectors

Expansion Connector 1



GND	1
AD4	2
AD5	3
AD6	4
AD7	5
AD8	6
TXD uC	7
RXD uC	8
PORT P9	9
PORT P10	10
PORT P11	11
PORT P12	12
PORT P13	13
PORT P14	14
PORT P15	15
PORT P16	16
CHARGE JACK -	17
CHARGE JACK +	18
+ BATT	19
+BATT	20

Expansion Connector 2

GND	1
UREF	2
RESET	3
DCF/FREQ1	4
FREQ2	5
START	6
SCL	7
SDA	8
PORT P8	9
PORT P7	10
STROBE	11
SCLOCK	12
DATA	13
FT OUT	14
RS 232 TXD	15
RS 232 RXD	16
AMPL/REF POWER +	17
COMM/NAV POWER +	18
SCLOCK NEG	19
VCC	20

Technical Data

Operating Voltage	5V +- 10%
Current Consumption RX/TX	20/40 mA
Input Impedance Interface (Pull down)	100 kOhm
Input Amplitude Interface	> 3,5 V (H) < 1 V (L)
Output Amplitude Interface	> 4,2 V (H) < 0,4 V (L)
Receiver Sensitivity	-107 dBm
Max. Transmit Power	10 dBm
User-Interface 4Byte Frame	1,5 ms (with C-Control)

Safety Instructions

Keep batteries out of reach of children.

When inserting the batteries make sure to pay attention to the correct polarity.

Do not let batteries lie around openly. There is the risk of batteries being swallowed by children or pets. In such case, seek instant medical care.

Leaking or damaged batteries might cause acid burns when getting into contact with skin, therefore use suitable protective gloves in any such case.

Make sure that batteries are not short-circuited, thrown into fire or recharged. They might explode.

Disposal of Used Batteries/Rechargeable Batteries

You, as end user are under legal obligation to take back all used batteries and rechargeable batteries, **to dispose of them via domestic waste is not permitted.**



Batteries/rechargeable batteries which contain pollutants are marked with the symbols indicated on the side of this manual, they point out the prohibition to dispose of them via domestic waste.

The signs for the relevant heavy metals are: **Cd** = cadmium, **Hg** = mercury, **Pb** = lead.



You can return used batteries and rechargeable batteries free of charge, to supermarkets, dealer's shops or local collection centres where special collection boxes are set up for this purpose.

Doing this, you fulfil your legal obligation and also contribute to environmental protection.

