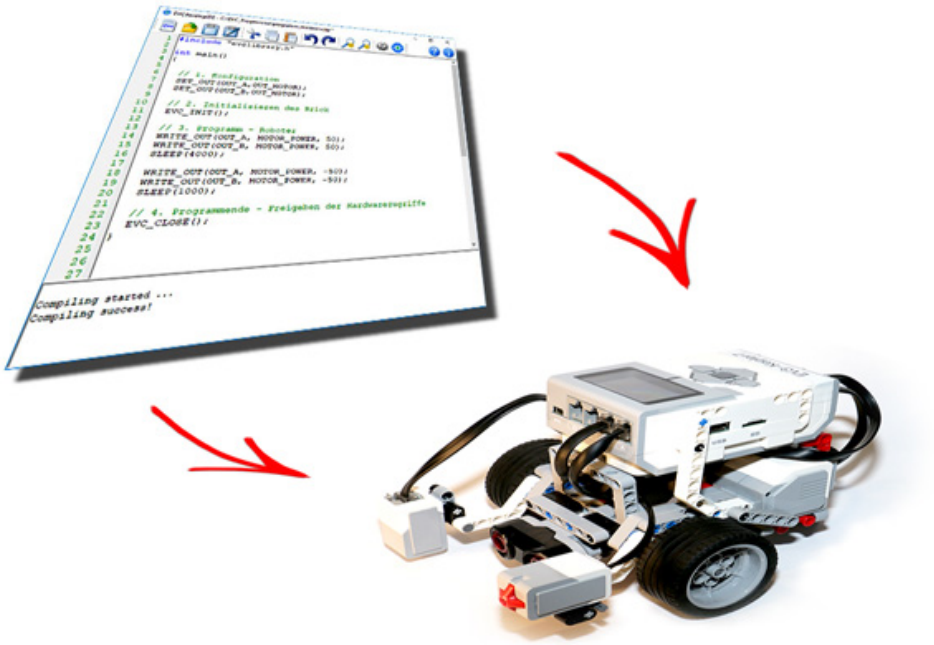
EVC*develop*

The most simple way to program the
LEGO® MINDSTORMS™ EV3 Brick
with C / C++



Leander Brandl

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

1 Einleitung

EVCdevelop ist ein System zur einfachen Softwareentwicklung für den LEGO EV3 Brick und speziell für autonome Roboter, die auf dem EV3 Brick basieren.

EVCdevelop besteht aus der Entwicklungsumgebung ***EVCdevelopIDE***, der Bibliothek ***evclibrary.h***, die es sehr einfach macht, Programme für den EV3 Brick mit C / C++ zu schreiben und umfangreichen Unterrichtsmaterialien, dem ***EVCdevelopTutorial***.

- KEINE Installation von Software!
- KEINE Installation von Treibern!
- KEINE Administrator-Berechtigungen!
- KEIN alternatives Betriebssystem für den EV3 Brick!
- KEINE SD-Karte!

- IDE für Windows 7, Windows 8, Windows 8.1, Windows 10 mit .NET Framework 4.5
- Lego Mindstorms EV3 Brick mit Standard-Firmware (Version 1.03 oder höher)
- Übertragung der Programme über USB

Just write your code,

one-click save-compile-upload

and have fun with your robots!

Wenn Ihnen EVCdevelop gefällt und Sie dieses Projekt unterstützen möchten,

würden wir uns sehr über eine Spende auf das folgende Konto freuen!

Verein Kepler iNNOVATIV - ZVR 338115499

IBAN: AT56 3800 0000 0519 5144 BIC: RZSTAT2G

2 Lizenzhinweise

2.1 EVCdevelopIDE

Die EVCdevelopIDE wird unter der MIT Lizenz veröffentlicht.

2.2 EVCdevelop Tutorial

Das EVCdevelop Tutorial wird unter der folgenden Creative Commons Lizenz veröffentlicht:

Namensnennung - Nicht kommerziell 4.0 International (CC BY-NC 4.0)

Sie dürfen:

Teilen - das Material in jedwedem Format oder Medium vervielfältigen und weiterverbreiten

Bearbeiten - das Material remixen, verändern und darauf aufbauen

Der Lizenzgeber kann diese Freiheiten nicht widerrufen solange Sie sich an die Lizenzbedingungen halten.

Unter folgenden Bedingungen:

Namensnennung - Sie müssen angemessene Urheber- und Rechteangaben machen, einen Link zur Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Diese Angaben dürfen in jeder angemessenen Art und Weise gemacht werden, allerdings nicht so, dass der Eindruck entsteht, der Lizenzgeber unterstütze gerade Sie oder Ihre Nutzung besonders.

Nicht kommerziell - Sie dürfen das Material nicht für kommerzielle Zwecke nutzen.

Keine weiteren Einschränkungen - Sie dürfen keine zusätzlichen Klauseln oder technische Verfahren einsetzen, die anderen rechtlich irgendetwas untersagen, was die Lizenz erlaubt.

Dies ist eine allgemeinverständliche Zusammenfassung der Lizenz (die diese nicht ersetzt):
https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode

2.3 EV3 Framework

Die EVCdevelopIDE verwendet Teile des EV3 Frameworks, das unter der GNU General Public License frei zur Verfügung gestellt wird.

The Ev3 Framework is in no way associated or affiliated with the Lego company. It was and is being developed out of sheer enthousiasme for the Lego Ev3 Mindstorms set. And it is here just so you can enjoy the set even more.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

2.4 GNU GCC Compiler Collection

EVCdevelop verwendet die GNU GCC Compiler Collection, die als freie Software entwickelt wurde. Die folgenden Komponenten sind in dieser EVCdevelop-Distribution enthalten:

GNU Binary Utilities	GNU General Public License 3.0
GNU Compiler	Collection GNU General Public License 3.0
GNU Debugger	GNU General Public License 3.0
GNU C Library	GNU Lesser General Public License 2.1
Linux Kernel	GNU General Public License 2.0
GNU Make	GNU General Public License 2.0
GNU Core Utilities	GNU General Public License 2.0

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

3 LEGO Mindstorms EV3

3.1 Der EV3 Brick

Der EV3 Brick ist eine Art Minicomputer auf dem das Betriebssystem Linux läuft unter dem man eigene Programme ausführen lassen kann. Neben dem integrierten User-Interface (Display, Buttons und LED-Anzeige) besitzt er Anschlüsse für 4 Sensoren und 4 Aktoren, wie auch eine USB- und Bluetooth-Schnittstelle.

Die Bedienung des Bricks, vom Ein- und Ausschalten bis zur Steuerung der Software des Bricks erfolgt über die Buttons unterhalb des Displays.

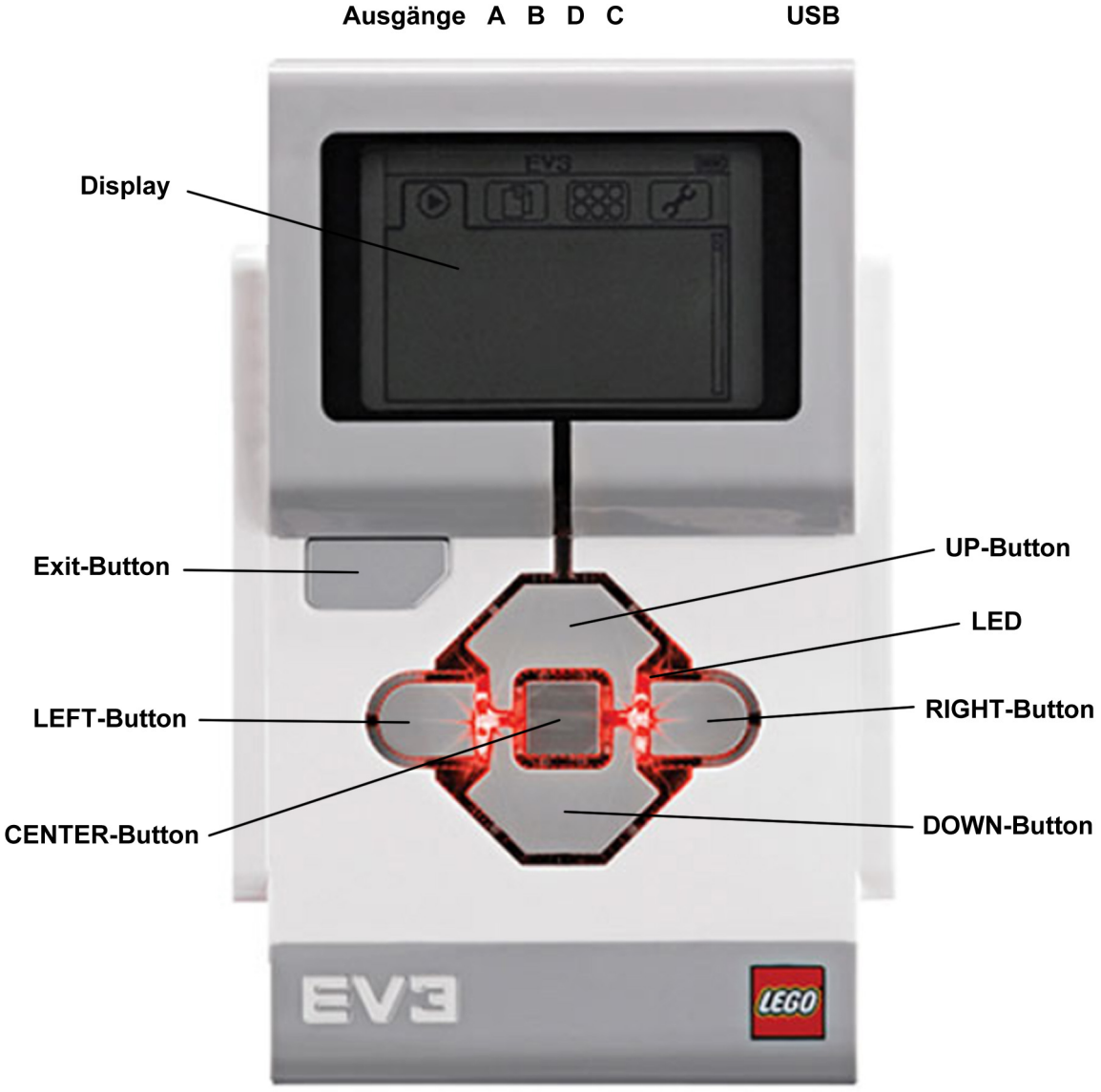


Abbildung 1: Der LEGO EV3 Brick

3.2 Die Sensoren

Im LEGO Mindstorms EV3 Education Basisset sind vier unterschiedliche Typen von Sensoren enthalten: zwei Touch-Sensoren, ein Farb/Licht-Sensor, ein Ultraschall-Sensor und ein Gyro-Sensor.

Der Touch-Sensor kann zum Beispiel eingesetzt werden damit ein Roboter auf die Berührung mit einem Hindernis reagieren kann.

Der Farb/Licht-Sensor liefert Farb- oder Helligkeitsinformationen über die Beschaffenheit einer Oberfläche, über die er bewegt wird.

Der Ultraschall-Sensor dient zum Messen von Entfernungen.

Der Gyro-Sensor kann bei einem Roboter zum Beispiel verwendet werden, um Informationen über die aktuelle Fahrtrichtung zu erhalten.

Abbildung 2: Die LEGO EV3 Sensoren

3.3 Die Motoren

Bei den LEGO EV3 Robotik-Kits sind zwei große Servomotoren und ein mittlerer Servomotor enthalten. Beide Arten von Motoren sind nicht nur Aktoren für Drehbewegungen, sondern beinhalten auch eine Steuer- und Regelelektronik die vielfältige Einsatzmöglichkeiten bietet.

So können die Motoren z. B. konstante Drehgeschwindigkeiten liefern, indem bei etwaigem Widerstand die Antriebskraft automatisch erhöht wird. Weiter lassen sich die tatsächlichen Umdrehungen abfragen oder mit Angabe eines Winkels genaue Positionen anfahren.

Abbildung 3: Die LEGO EV3 Motoren

3.4 LEGO Mindstorms EV3 Education Basisset und Home Edition

Die LEGO Mindstorms EV3 Robotik-Kits sind in zwei Editionen erhältlich: dem Education Basisset und der Home Edition.

Home Edition	**Education Basisset**
2 große Servomotoren	2 große Servomotoren
1 mittlerer Servomotor	1 mittlerer Servomotor
1 Tastsensor	2 Tastsensoren
1 Licht- und Farbsensor	1 Licht- und Farbsensor
1 Infrarot-Entfernungssensor	1 Ultraschall-Entfernungssensor
1 Infrarot Fernsteuerung	1 Gyro-Sensor
	1 Akku (Ladegerät muss separat erworben werden)

Die beiden Sets unterscheiden sich also grundsätzlich darin, dass das Education Basisset zusätzlich einen zweiten Tastsensor, einen Gyro-Sensor und einen Akku enthält.

Der Infrarot-Entfernungssensor der Home Edition dient wie der Ultraschall-Entfernungssensor des Education Basissets zum Messen von Entfernungen, wobei allerdings festgehalten werden muss, dass der Ultraschall-Sensor wesentlich präziser bei der Entfernungsbestimmung ist.

4 Installation und Updates

EVCdevelop kann ohne Installation auf jedem Computer verwendet werden, auf dem Windows 7, Windows 8, Windows 8.1 oder Windows 10 mit .NET framework 4.5 installiert ist.

EVCdevelop steht in zwei ZIP-Datei zum Download zur Verfügung:
EVCdevelop_complete.zip und **EVCdevelop_update.zip**

EVCdevelop_complete.zip
Diese wird bei der ersten Installation von EVCdevelop benötigt und enthält alle benötigten Dateien inklusive des GNU GCC Compilers.

EVCdevelop_update.zip
Zur Installation eines Updates wird in der Folge nur diese Datei benötigt. Diese enthält die aktuelle Version der Entwicklungsumgebung und das aktuelle Tutorial.

4.1 Grundinstallation

Die ZIP-Datei kann an jedem beliebigen Speicherort (lokales Laufwerk, USB-Stick, Netzlaufwerk, ...) entpackt werden und enthält alles, was nötig ist um mit EVCdevelop Programme für den EV3 Brick zu schreiben, zu compilieren und über USB auf den Brick zu übertragen.

Zu beachten!
Der Pfad zu dem Ordner, in dem sich die entpackten Dateien befinden, darf keine Leerzeichen oder Sonderzeichen enthalten!

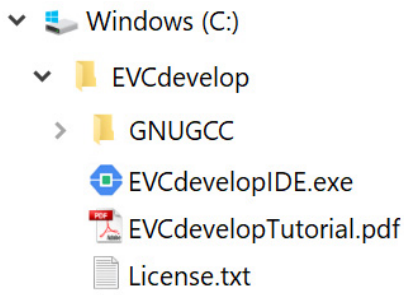


Abbildung 4: Grundinstallation Ordner- und Dateistruktur

4.2 Update

Jedes Update von EVCdevelop beinhaltet nur eine aktuelle Version der Entwicklungsumgebung und des Tutorials.

Es genügt, die ZIP-Datei in den Installationsordner zu entpacken und die beiden Dateien EVCdevelopIDE.exe und EVCdevelopTutorial.pdf zu ersetzen.

4.3 Start der Entwicklungsumgebung EVCdevelopIDE

Der Start der Entwicklungsoberfläche erfolgt durch Aufruf der Anwendung **EVCdevelopIDE.exe** aus dem Installationsverzeichnis.

5 Einführung in die Entwicklungsumgebung EVCdevelopIDE

EVCdevelopIDE ist ein Texteditor mit Funktionen zum Öffnen und Speichern von Dateien, wie auch der Möglichkeit den Quellcode für den EV3 Brick zu compilieren und das fertige Programm auf den Brick zu übertragen.

Die Oberfläche der Entwicklungsumgebung ist bewusst sehr einfach gehalten, um speziell auch Kindern die Möglichkeit zu bieten mit dem Programmieren zu beginnen, ohne von einer unübersichtlichen Programmoberfläche mit unzähligen Menüs, Unterfenstern, Reitern, ... „erschlagen“ zu werden.

Auch wurde die Entwicklung von EVCdevelop darauf ausgelegt, dass ein Programm nur aus einer Datei besteht, die den Code beinhaltet. Somit können zum Beispiel auch mehrere Programme in einem Ordner liegen und man muss sich nicht Gedanken über die Projektverwaltung machen, wie dies in fortgeschrittenen Programmierumgebungen von Nöten ist.

5.1 Die Programmoberfläche

Die Programmoberfläche besteht aus einer Menüleiste, dem Texteditor und einem Statusfenster am unteren Rand.

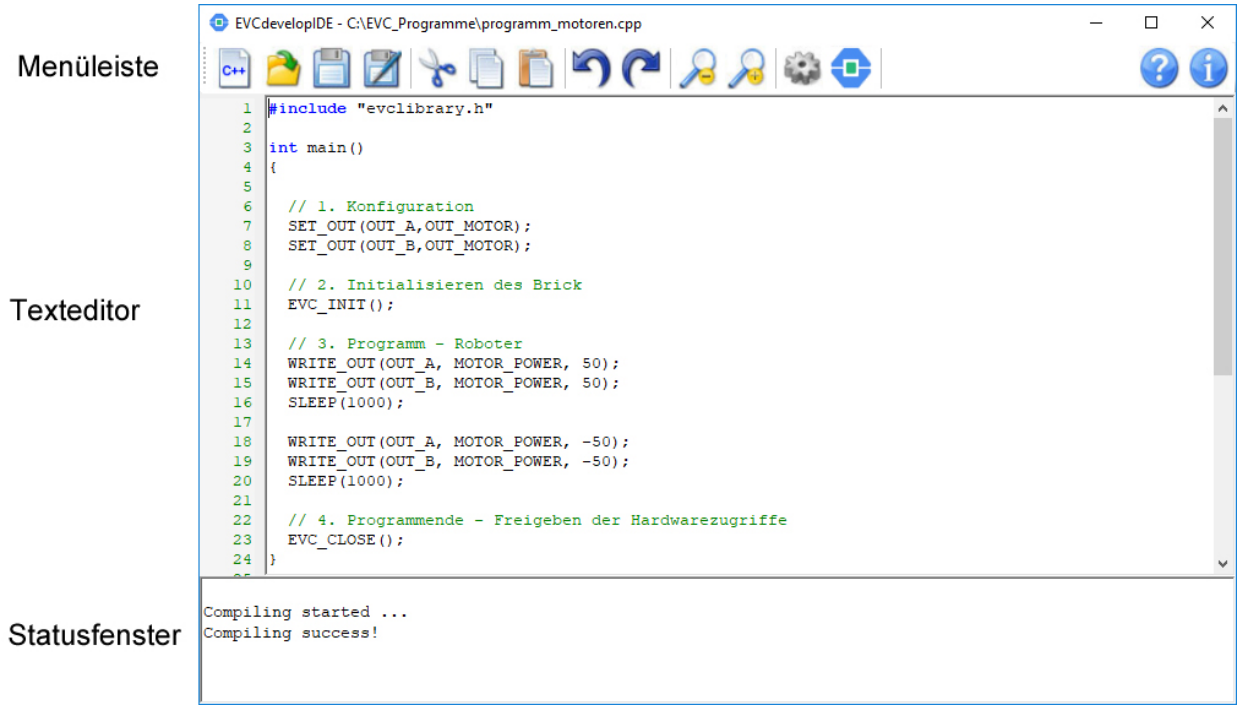


Abbildung 5: Programmoberfläche der EVCdevelopIDE

5.1.1 Die Menüleiste

Die ersten vier Symbole in der Menüleiste dienen der Dateiverwaltung.

New — eine neue, leere Datei wird im Texteditor angelegt

Open — öffnen einer vorhandenen Datei

Save — speichern der aktuell gearbeiteten Datei

Save as — speichern der aktuellen Daten unter einem anderen Namen

Die nächsten drei Symbole dienen der Verwendung der Zwischenablage.

Cut — markierten Text ausschneiden und in der Zwischenablage ablegen

Copy — kopieren des markierten Textes und Ablegen in der Zwischenablage

Paste — einfügen eines Textes aus der Zwischenablage an der aktuellen Position

Mit den beiden Pfeilen lassen sich Schritte bei der Eingabe rückgängig machen oder wiederherstellen.

Undo — Rückgängigmachen der letzten Eingabeschritte

Redo — Wiederherstellen eines rückgängig gemachten Eingabeschritts

Mit nächsten Schaltflächen dienen zum Ändern der Schriftgröße im Texteditor.

Zoom in — vergrößern des Schriftgrades

Zoom out — verkleinern des Schriftgrades

Die beiden letzten Schaltflächen in der Menüleiste auf der linken Seite dienen zum Compilieren und Übertragen des Programms auf den Brick.

Compile Compilieren des Programmcodes

Upload Compilieren und Übertragen des Programms auf den Brick

Rechts befinden sich zwei Schaltflächen zum Aufruf eines Informationsfensters und des Tutorials.

Tutorial Anzeige des PDF-Dokuments mit dem Tutorial

About Anzeige eines Infofensters

5.1.2 Der Texteditor

Im Texteditor wird der Programmcode geschrieben. Um den Code besser lesbar zu machen, werden reservierte Wörter der Programmiersprache C – sogenannte Schlüsselwörter – blau angezeigt. Kommentare – das sind beliebige Informationen im Quellcode – werden mit zwei Schrägstrichen eingeleitet und dann vom Texteditor grün und kursiv dargestellt. Der Texteditor stellt die Funktionen zum Kopieren, Ausschneiden und Einfügen von Text ergänzend zu den Symbolen in der Menüleiste mit den üblichen Tastenkombinationen zur Verfügung:

Strg + C	kopieren
Strg + X	ausschneiden
Strg + V	einfügen

5.1.3 Das Statusfenster

In diesem Fenster werden Informationen angezeigt, wenn der Quellcode compiliert wird. Kann das Programm für den EV3 Brick fehlerfrei erstellt werden, wird dies mit der Meldung „Compiling success!“ angezeigt und das Programm kann auf den EV3 Brick übertragen werden. Liefert der Compiler Fehlermeldungen zurück, so werden diese detailliert im Statusfenster angezeigt.

6 Kurzanleitung - ein erstes Programm

6.1 Start der Entwicklungsoberfläche EVCdevelopIDE

Der Start der Entwicklungsumgebung ***EVCdevelopIDE*** erfolgt durch Aufruf der Anwendung *EVCdevelopIDE.exe* aus dem Verzeichnis in das EVCdevelop entpackt wurde.

z. B. c:\EVCdevelop\EVCdevelopIDE.exe

6.2 Schreiben des Progammcodes

Klick auf 

Nachdem die Entwicklungsumgebung gestartet wurde, kann direkt begonnen werden den Programmcode zu schreiben. Hier ein kurzes Programm, dass einen Roboter mit zwei Motoren für kurze Zeit vorwärts und rückwärts fahren lässt.

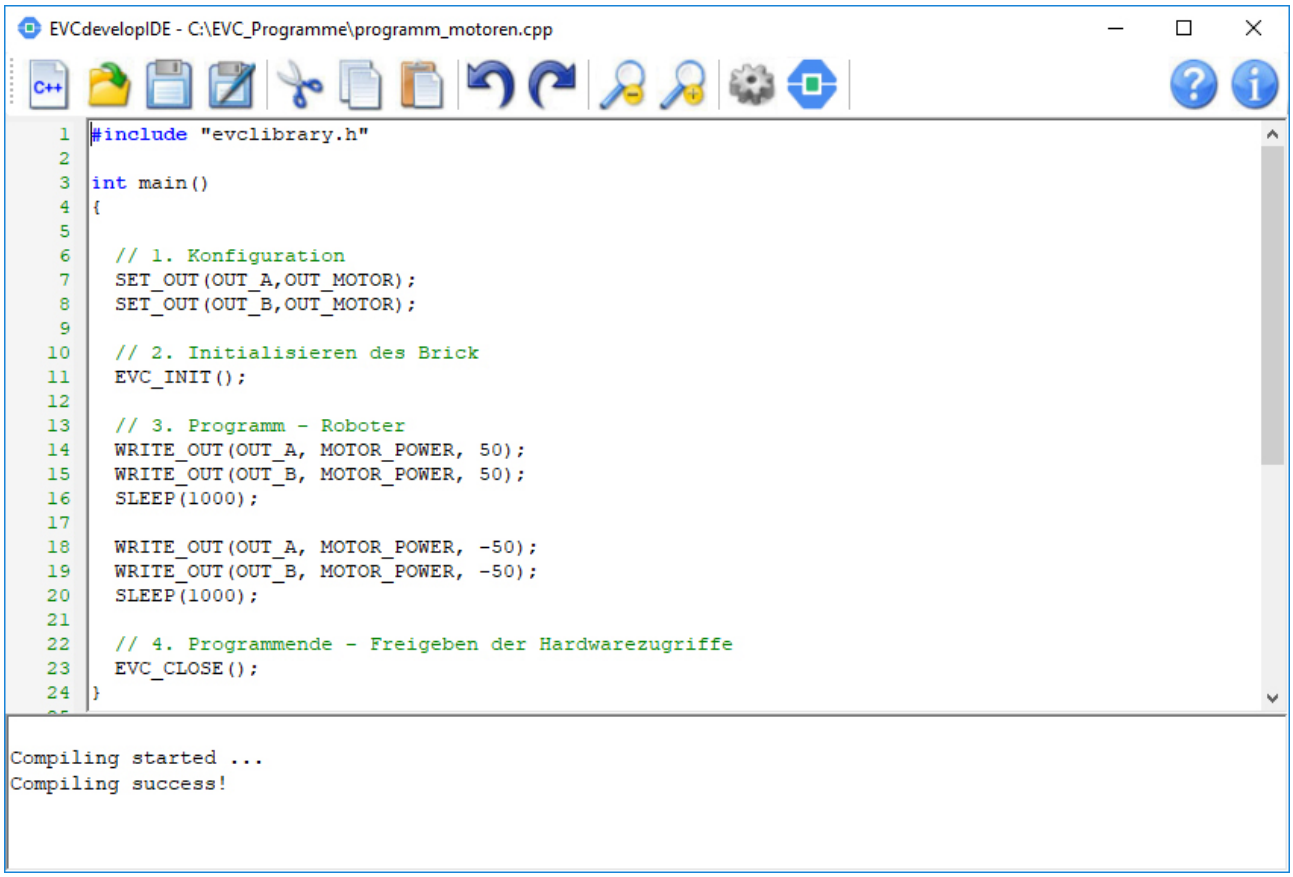


Abbildung 6: Schreiben des Programmcodes in der Entwicklungsoberfläche

6.3 Speichern des Programms

Klick auf  oder 

Bevor aus dem eingegebenen Code das Programm für den Roboter erzeugt werden kann, muss der Programmcode abgespeichert werden.

Als Extension für die Dateinamen muss **.cpp** vergeben werden!

ACHTUNG: Beim Abspeichern der Programme ist unbedingt zu berücksichtigen, dass weder die Dateinamen selbst, noch Ordnernamen Sonderzeichen oder Leerzeichen enthalten dürfen!!!

Das einzige Zeichen zur besseren Strukturierung von Datei- und Ordnernamen ist der Unterstrich.

z. B. c:\MeineEVCProgramme\programm_motoren.cpp

6.4 Compilieren

Klick auf 

Ist der Programmcode abgespeichert, so kann dieser in eine für den EV3 Brick verständliche Sprache übersetzt werden, damit das Programm dann am Brick ausgeführt werden kann. Dieser Vorgang wird als *compilieren* und die dazu verwendete Software als *Compiler* bezeichnet. Dieser befindet sich im Ordner *CSLite* im Installationsverzeichnis.

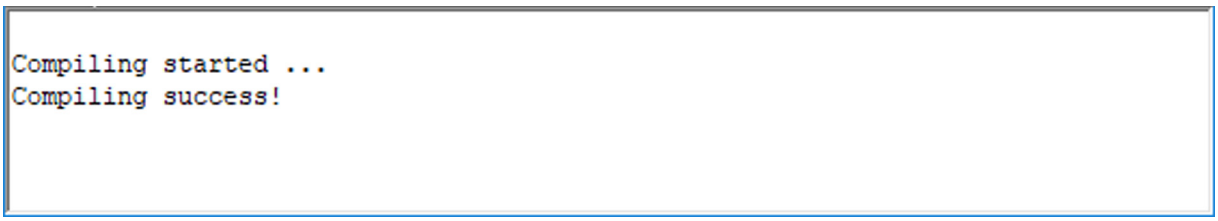


Abbildung 7: Kontrolle eines erfolgreichen Compilierens im Statusfenster

6.5 Übertragen des Programms auf den EV3 Brick

Klick auf 

Zum Übertragen des compilierten Programms auf den EV3 Brick muss dieser mit einem USB-Kabel mit dem Computer verbunden sein. Nach der abgeschlossenen Übertragung ertönt am Brick ein kurzer Signalton.

Durch einen Klick auf diese Schaltfläche werden zunächst die Änderungen im Programmcode gespeichert, dann das Programm compiliert und abschließend auf den Brick übertragen.

6.6 Starten des Programms am EV3 Brick

Hauptmenü  > myapps > EVCdevelopRUN

Nun kann das Programm am EV3 Brick gestartet werden. Dazu wechselt man zum Hauptmenüpunkt Programme und weiter in den Ordner *myapps*.

Dort befindet sich das Programm *EVCdevelopRUN*, mit dem das mit EVCdevelop erstellte Programm gestartet werden kann.

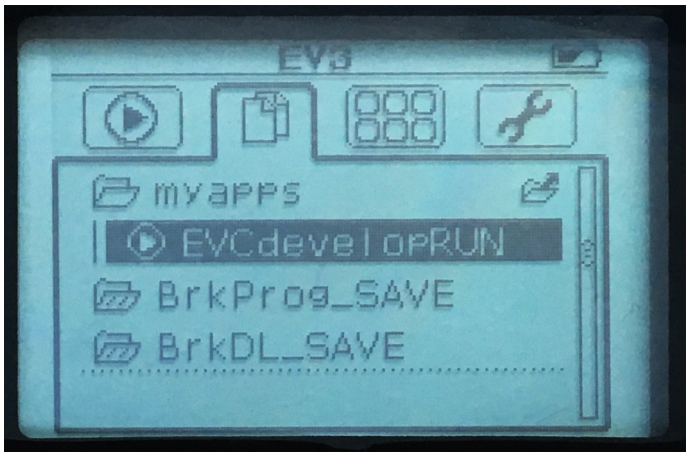


Abbildung 8: Starten des Programms am EV3 Brick

7 Programmbeispiele und Grundlagen der Sprache C

Programme, die man für den EV3 Brick mit EVCdevelop erstellt, werden in der Programmiersprache C geschrieben.

Der Zugriff auf alle Hardware-Komponenten (Sensoren, Motoren, Display, ...) erfolgt über die EVC-Bibliothek. Für alle Komponenten stehen entsprechende Funktionen zur Verfügung, die es sehr einfach möglich machen z. B. Sensorwerte einzulesen, Motoren anzusteuern oder Text am Display auszugeben, ohne sich mit der sehr komplexen Programmierung, die fortgeschrittene C- und Linux-Kenntnisse erfordern, auseinanderzusetzen.

Somit können auch bereits Kinder sehr spielerisch in das Code-basierte Programmieren eines Roboters herangeführt werden und lernen so ganz nebenbei auch noch die Grundlagen der Programmiersprache C.

7.1 Grundstruktur eines EVC-Programms und Ansteuern der Motoren

An den Ausgängen A, B, C und D können die Motoren angeschlossen werden. Bei den Beispielen kommt ein einfaches Robotermodell mit zwei Motoren zum Einsatz, die an den Ausgängen A und B angeschlossen sind.

7.1.1 Grundstruktur eines EVC-Programms

Zu Beginn eines jeden EVC-Programms wird die EVC-Bibliothek eingebunden.

Im Anschluss daran steht in der Hauptfunktion ***`int main()`*** der Programmcode.

Basierend auf der EVC-Bibliothek müssen in der Hauptfunktion die folgenden vier Programmabschnitte enthalten sein:

1. Konfiguration der angeschlossenen Sensoren und Motoren

An dieser Stelle wird festgelegt, an welchen Eingängen Sensoren angeschlossen und an welchen Ausgängen Motoren angeschlossen sind.

2. Initialisierung

Ausgehend von den Anweisungen im Abschnitt 1 wird das Programm mit der Funktion ***`EVC_INIT()`*** so initialisiert, dass es auf die angeschlossenen Sensoren und Motoren zugreifen kann. Weiters werden auch der Zugriff auf das Display, die LED und die Taster am Bedienpanel initialisiert.

3. Steuerung des Roboters

Hier steht nun der eigentliche Programmcode zur Steuerung des Roboters – es können Sensorwerte eingelesen werden und die Motoren angesteuert werden.

4. Programmende

Am Schluss des Programms muss der Zugriff auf die Hardware-Komponenten des EV3 Brick beendet werden. Dies geschieht durch den Aufruf der Funktion ***`EVC_CLOSE()`***.

7.1.2 Programmbeispiel

Wird das Programm gestartet, so fährt der Roboter zunächst vier Sekunden mit einer Leistung von 50 Prozent vorwärts, bleibt anschließend für vier Sekunden stehen und fährt dann mit einer Leistung von 70 Prozent für vier Sekunden rückwärts bis er letztendlich stehenbleibt.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_OUT(OUT_A,OUT_MOTOR);
7    SET_OUT(OUT_B,OUT_MOTOR);
8
9    // 2. Initialisierung des EV3 Brick
10   EVC_INIT();
11
12   // 3. Steuerung des Roboters
13   WRITE_OUT(OUT_A, MOTOR_POWER, 50);
14   WRITE_OUT(OUT_B, MOTOR_POWER, 50);
15   SLEEP(4000);
16   WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_COAST);
17   WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_COAST);
18   SLEEP(4000);
19   WRITE_OUT(OUT_A, MOTOR_POWER, -70);
20   WRITE_OUT(OUT_B, MOTOR_POWER, -70);
21   SLEEP(4000);
22
23   // 4. Programmende
24   EVC_CLOSE();
25   return 0;
26 }
```

Listing 1: Grundstruktur eines EVC Programms und Motorsteuerung

7.1.3 Erklärungen zu diesem Programmbeispiel

Allgemeines

- Ein Programm in der Programmiersprache C besteht grundsätzlich aus der Funktion ***main()***, die aufgerufen und ausgeführt wird, wenn das Programm gestartet wird.
- Jede Anweisung wird in der Programmiersprache C mit einem Strichpunkt beendet.
- Es ist üblich, Anweisungen, die mit geschwungenen Klammern { } zu Blöcken zusammengefasst werden, eingerückt zu schreiben.
- Ergänzende Informationen, die nicht zum Programmcode gehören, werden als Kommentare bezeichnet, hinter zweifachen // geschrieben und in der EVCdevelopIDE grün dargestellt.
- Bei Schlüsselwörtern handelt es sich um vordefinierte reservierte Bezeichner, die eine besondere Bedeutung für den Compiler haben, wie z. B. ***include, int, if, while, break, return, ...*** Diese werden in der EVCdevelopIDE blau hervorgehoben.

Zeile 1: #include "evclibrary.h"

Zu Beginn des Programms wird die EVC-Bibliothek eingebunden.

Zeile 3: int main()

Hier beginnt die Hauptfunktion eines C-Programms. Alle Anweisungen, die zu dieser Funktion gehören, werden in geschwungenen Klammern { } zusammengefasst. Der Programmcode nach einer geschwungenen Klammer auf wird (z. B. zwei Zeichen) eingerückt.

Zeile 6: SET_OUT(OUT_A,OUT_MOTOR);

Mit der Funktion ***SET_OUT(port, value)*** wird festgelegt, an welchem Ausgang ein Motor angeschlossen ist.

Mit dem ersten Parameter ***port*** wird der gewünschte Ausgang angegeben: OUT_A, OUT_B, OUT_C oder OUT_D

Der zweite Parameter ***value*** gibt an, dass an dem entsprechenden Ausgang ein Motor angeschlossen ist: OUT_MOTOR

Zeile 10: EVC_INIT();

Initialisierung des Zugriffs auf die Hardware-Komponenten des EV3 Brick.

Zeile 13: WRITE_OUT(OUT_A, MOTOR_POWER, 50);

Die Funktion ***WRITE_OUT(port, mode, value)*** dient zum Ansteuern der Ausgänge.

Mit dem ersten Parameter wird der gewünschte Ausgang festgelegt, hier OUT_A.

Der zweite Parameter gibt an, in welchem Modus der Motor betrieben werden soll, hier MOTOR_POWER.

In diesem Modus dreht sich ein Motor mit der vorgegebenen Leistung. Erfährt der Motor einen Widerstand (z. B. ein Hindernis oder beim Hinauffahren einer schiefen Ebene), so bleibt die Leistung konstant. Das bedeutet, dass die Geschwindigkeit mit der sich der Motor letztendlich dreht, geringer wird.

Der dritte Parameter steht für die Leistung mit der sich der Motor drehen soll. Die Werte können zwischen -100 und 100 liegen. Das Vorzeichen bestimmt die Drehrichtung, der Wert bezeichnet eine Leistung zwischen 0% und 100%.

Zeile 15: SLEEP(4000);

Steht an einer Stelle im Programmcode die Funktion ***SLEEP()***, so wird die Ausführung des Programms und somit die Abarbeitung der folgenden Anweisungen an dieser Stelle angehalten. Mit dem Parameter ***value*** wird die Zeit in Millisekunden angegeben.

Zeile 16: WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_COAST);

Wird der Funktion ***WRITE_OUT(port, mode, value)*** im zweiten Parameter der Wert MOTOR_STOP übergeben, bleibt der entsprechende Motor stehen.

Mit dem dritten Parameter wird festgelegt, ob der Motor nachläuft (MOTOR_COAST) oder abrupt angehalten werden soll (MOTOR_BRAKE).

Zeile 23: EVC_CLOSE();

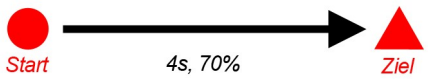
Am Ende des Programms muss der Zugriff auf die Hardware-Komponenten des EV3 Brick freigegeben werden. Beim Aufruf dieser Funktion werden auch die Motoren angehalten, falls sich diese am Ende des Programms noch drehen sollten.

Zeile 25: return 0;

Am Ende der Hauptfunktion ***`int_main()`*** wird in einem C-Programm der Rückgabewert der Funktion mit dem Schlüsselwort return festgelegt.

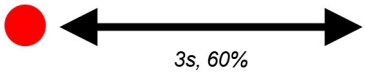
7.1.4 Aufgabe M1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 4 Sekunden mit 70% Motorleistung vorwärts fährt und danach stoppt.



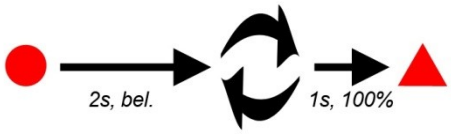
7.1.5 Aufgabe M2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 3 Sekunden mit 60% Motorleistung vorwärts fährt, wieder die gleiche Strecke rückwärts fährt und danach stoppt.



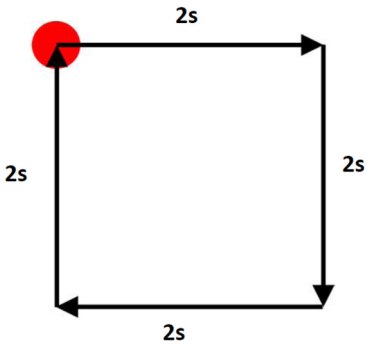
7.1.6 Aufgabe M3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 2 Sekunden mit beliebiger Motorleistung vorwärts fährt, sich auf der Stelle einmal um seine eigene Achse dreht, dann nochmals eine Sekunde mit maximaler Leistung in der ursprünglichen Richtung vorwärts fährt und danach stoppt.



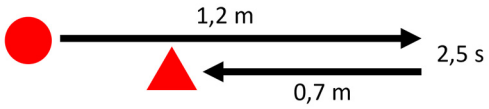
7.1.7 Aufgabe M4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt.



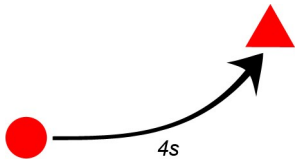
7.1.8 Aufgabe M5

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) genau 1,2 Meter geradeaus fährt, 2,5 Sekunden stehen bleibt und dann 0,7 Meter rückwärts fährt.



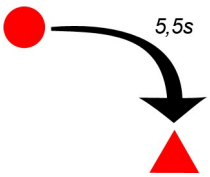
7.1.9 Aufgabe M6

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 4 Sekunden eine leichte Kurve nach links fährt und danach stoppt.



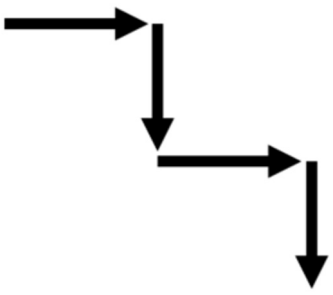
7.1.10 Aufgabe M7

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 5,5 Sekunden eine starke Kurve nach rechts fährt und danach stoppt.



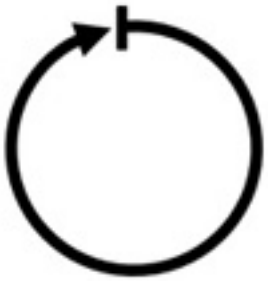
7.1.11 Aufgabe M8

Erstelle ein Programm, mit dem der Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Treppenmuster fährt. Bei den Richtungsänderungen soll er sich immer am Stand drehen.



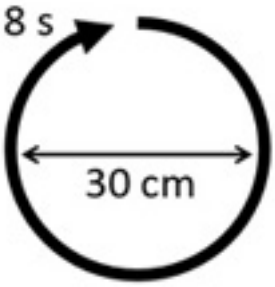
7.1.12 Aufgabe M9

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) einen Kreis fährt und genau an der Ausgangsposition stoppt. Die Motorleistung kann dabei beliebig gewählt werden.



7.1.13 Aufgabe M10

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) einen Kreis mit einem Durchmesser von ca. 30 cm fährt. Dafür soll der Roboter genau 8 Sekunden brauchen.



7.1.14 Aufgabe M11

Erstelle ein Programm mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) den folgenden Bewegungsablauf mit unterschiedlichen Geschwindigkeiten nachfährt. Die Geschwindigkeiten können dabei beliebig gewählt werden:

- beide Motoren für 4 Sekunden vorwärts
- Roboter hält für 2 Sekunden an
- Drehung am Stand für 3 Sekunden
- Roboter fährt für 3 Sekunden rückwärts
- Drehung nur mit einem Rad für 5 Sekunden
- Roboter hält für 4 Sekunden an
- Roboter fährt 2 Sekunden lang rückwärts
- Roboter stoppt

7.2 Ein- und Ausschalten der LED am Bedienpanel

Die LED am Bedienpanel kann in verschiedenen Farben leuchten oder auch blinken.

7.2.1 Programmbeispiel

Nach dem Start des Programms leuchtet die LED rot. Zwei Sekunden später ändert sich die Farbe zu orange und nach weiteren zwei Sekunden zu grün. Nach weiteren zwei Sekunden wird die LED für zwei Sekunden ausgeschaltet bevor das Programm beendet wird.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6
7    // 2. Initialisierung des EV3 Brick
8    EVC_INIT();
9
10   // 3. Steuerung des Roboters
11   WRITE_LED(LED_RED);
12   SLEEP(2000);
13   WRITE_LED(LED_ORANGE);
14   SLEEP(2000);
15   WRITE_LED(LED_GREEN);
16   SLEEP(2000);
17   WRITE_LED(LED_BLACK);
18   SLEEP(2000);
19
20   // 4. Programmende
21   EVC_CLOSE();
22   return 0;
23 }
```

Listing 2: Ein- und Ausschalten der LED am Bedienpanel

7.2.2 Erklärungen zu diesem Programmbeispiel

Zeile 11: WRITE_LED(LED_RED);

Mit dieser Funktion werden die LEDs ein- und ausgeschaltet und die Farbe, wie auch das Leuchtverhalten festgelegt. Mit dem Parameter color wird die gewünschte Farbe festgelegt: LED_BLACK, LED_GREEN, LED_RED, LED_ORANGE, LED_GREEN_FLASH, LED_RED_FLASH, LED_ORANGE_FLASH, LED_GREEN_PULSE, LED_RED_PULSE, LED_ORANGE_PULSE

7.2.3 Aufgabe LED1 - Farbenspiel mit der LED

Erstelle ein Programm das mit der eingebauten LED im Bedienfeld des EV3 Roboters alle möglichen Farben und Anzeigemodi nacheinander jeweils für eine Dauer von 2 Sekunden zeigt. Danach soll das Programm automatisch enden.

7.3 Ausgeben von Text am Display

Mit der EVC-Bibliothek kann am Display Text in 11 Zeilen mit 22 Zeichen pro Zeile ausgegeben werden.

7.3.1 Programmbeispiel

Das folgende Programm gibt am Display in Zeile 1, 5, 8 und 11 zeitversetzt Text aus.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6
7    // 2. Initialisierung des EV3 Brick
8    EVC_INIT();
9
10   // 3. Steuerung des Roboters
11   LCD_DRAW_TEXT(7,1,"EVCdevelop");
12   LCD_DRAW_TEXT(7,5,"EV3::ROBOT");
13   SLEEP(2000);
14   LCD_DRAW_TEXT(9,8,"HELLO");
15   SLEEP(2000);
16   LCD_DRAW_TEXT(7,11,"Good bye!");
17   SLEEP(2000);
18
19   // 4. Programmende
20   EVC_CLOSE();
21   return 0;
22 }
```

Listing 3: Ausgeben von Text am Display

7.3.2 Erklärungen zu diesem Programmbeispiel

Zeile 11: LCD_DRAW_TEXT(7,1,"EVCdevelop");

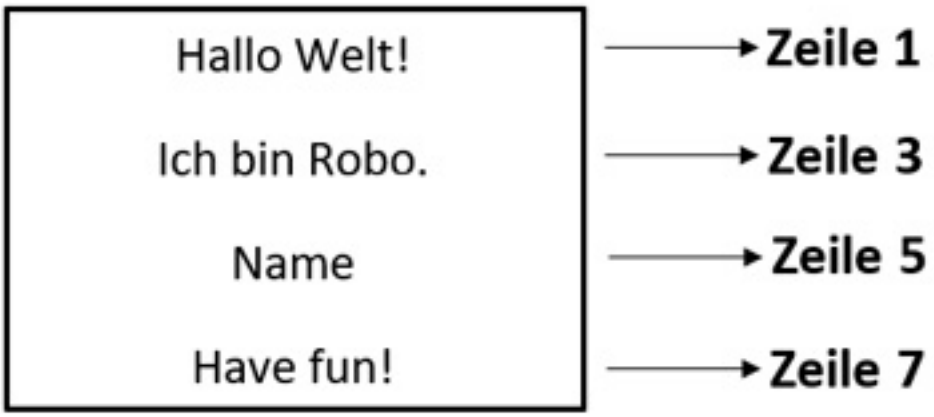
Der Funktion ***LCD_DRAW_TEXT(spalte, zeile, text)*** wird die Position des auszugebenden Textes mit den Werten für die Parameter spalte und zeile übergeben.

Für ***spalte*** kann ein Wert zwischen 1 und 22 gewählt werden, für ***zeile*** ein Wert zwischen 1 und 11.

Der Text wird als dritter Parameter übergeben und muss innerhalb von zweifachen Anführungszeichen stehen.

7.3.3 Aufgabe D1

Erstelle ein Programm, bei dem zentriert in der Mitte des Displays folgender Text ausgegeben wird (setze in der Zeile 5 deinen Namen ein).

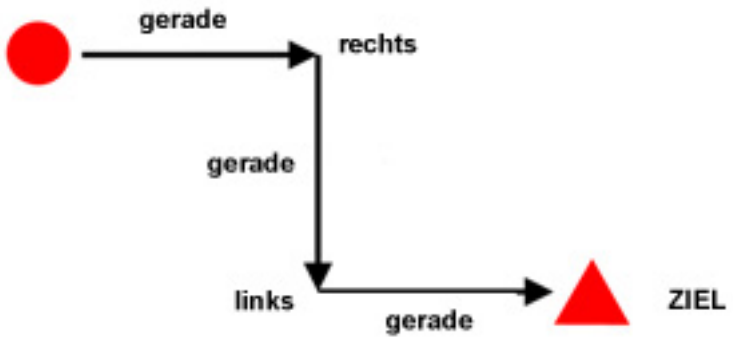


7.3.4 Aufgabe D2

Ergänze das Programm von Aufgabe D1 so, dass eine Zeile nach der anderen in einem zeitlichen Abstand von drei Sekunden erscheint.

7.3.5 Aufgabe D3

Erstelle ein Programm, bei dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) die vorgegebene Figur nachfährt. Dabei soll der aktuelle Bewegungsablauf mit den in der Grafik angegebenen Bezeichnungen am Display angezeigt werden.



7.4 Zeichnen am Display

Am Display können auch Grafiken angezeigt werden. Die EVC-Bibliothek stellt dazu Funktionen zum Zeichnen von Rechtecken und einzelnen Punkten zur Verfügung. Das grafische Koordinatensystem des Displays hat eine Größe von 178 x 128 Pixel. Der Nullpunkt des Koordinatensystem befindet sich in der linken oberen Ecke.

7.4.1 Programmbeispiel

Dieses Programm zeichnet am Display zwei rechteckige Rahmen und innerhalb dieses Bereiches sechs ausgefüllte Rechtecke.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6
7    // 2. Initialisierung des EV3 Brick
8    EVC_INIT();
9
10   // 3. Steuerung des Roboters
11   LCD_DRAW_RECTANGLE(0, 0, 177, 127, 1);
12   LCD_DRAW_RECTANGLE(3, 3, 174, 124, 1);
13   LCD_DRAW_FILLRECTANGLE(20, 20, 60, 60, 1);
14   LCD_DRAW_FILLRECTANGLE(35, 40, 45, 50, 0);
15   LCD_DRAW_FILLRECTANGLE(117, 20, 157, 60, 1);
16   LCD_DRAW_FILLRECTANGLE(132, 40, 142, 50, 0);
17   LCD_DRAW_FILLRECTANGLE(79, 50, 99, 80, 1);
18   LCD_DRAW_FILLRECTANGLE(30, 100, 147, 110, 1);
19   SLEEP(5000);
20
21   // 4. Programmende
22   EVC_CLOSE();
23   return 0;
24 }
```

Listing 4: Zeichnen am Display

7.4.2 Erklärungen zu diesem Programmbeispiel

Zeile 11: LCD_DRAW_RECTANGLE(0, 0, 177, 127, 1);

Die Funktion LCD_DRAW_RECTANGLE(x1, y1, x2, y2, color) zeichnet ein Rechteck. Die Koordinaten x1, y1 beschreiben den linken oberen Eckpunkt, die Koordinaten x2, y2 geben den rechten unteren Eckpunkt des Rechteckes an. Die Werte für die x-Koordinaten können zwischen 0 und 177 liegen, die Werte für die y-Koordinaten zwischen 0 und 127.

Mit dem Parameter color wird die Zeichenfarbe (0 für weiß, 1 für schwarz) festgelegt.

Zeile 13: LCD_DRAW_FILLRECTANGLE(20, 20, 60, 60, 1);

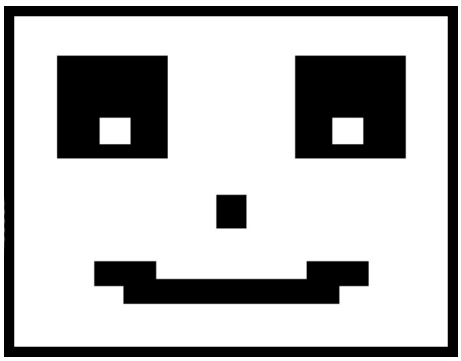
Mit der Funktion LCD_DRAW_FILLRECTANGLE(x1, y1, x2, y2, color) kann ein ausgefülltes Rechteck gezeichnet werden.

7.4.3 Aufgabe Z1

Erstelle ein Programm, das vier ausgefüllte und vier nicht ausgefüllte Rechtecke am Display anzeigt, die sich nicht überschneiden, aber über den gesamten Bildschirmbereich verteilt sein sollen.

7.4.4 Aufgabe Z2

Erstelle ein Programm mit dem das in der Abbildung dargestellte Gesicht am Display gezeichnet wird.



7.4.5 Aufgabe Z3

Erweitere das Gesicht aus Aufgabe Z2 dahingehend, dass sich die Pupillen in den Augen ein paar Mal hin- und herbewegen.

7.5 Variablen und wiederholtes Ausführen von Anweisungen

Oft ist es notwendig, dass Anweisungen mehrmals hintereinander ausgeführt werden. Anstatt die Anweisungen, die mehrmals ausgeführt werden sollen, entsprechend oft im Programmcode anzuführen, kann man dazu Schleifen einsetzen.

In der while-Schleife wird der darin enthaltene Programmcode wiederholt ausgeführt, solange eine zuvor festgelegte Bedingung erfüllt ist. Für die Anzahl der Wiederholungen kommt bei der Verwendung einer while-Schleife meist eine Zählvariable zum Einsatz.

7.5.1 Programmbeispiel

Mit diesem Programm soll der Roboter einen Bewegungsablauf (zwei Sekunden vorwärts fahren und dann zwei Sekunden drehen) vier Mal hintereinander ausführen.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren
  SET_OUT(OUT_A, OUT_MOTOR);
  SET_OUT(OUT_B, OUT_MOTOR);

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  // 3. Steuerung des Roboters
  int Zaehler = 1;
  while (Zaehler<=4)
  {
    WRITE_OUT(OUT_A, MOTOR_POWER, 50);
    WRITE_OUT(OUT_B, MOTOR_POWER, 50);
    SLEEP(2000);
    WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_COAST);
    SLEEP(2000);
    Zaehler = Zaehler + 1;
  }

  // 4. Programmende
  EVC_CLOSE();
  return 0;
}
```

Listing 5: Variablen und wiederholtes Ausführen von Anweisungen

7.5.2 Erklärungen zu diesem Programmbeispiel

Zeile 13: int Zaehler = 1;

Hier wird eine Variable vom Typ Integer definiert und gleichzeitig der Wert 1 zu gewiesen. In Variablen vom Typ Integer können ganzzahlige Werte abgelegt werden.

Die Variable Zaehler dient in diesem Programm dazu, um die Anzahl der Schleifendurchläufe mitzuzählen.

Zeile 14: while (Zaehler<=4) ...

In dieser Zeile wird die Bedingung für die while-Schleife festgelegt. Nach dem Schüsselwort while steht in runden Klammern die Bedingung, wie lange die Schleife läuft.

Der Programmcode, der zwischen den geschwungenen Klammern steht, wird so oft wiederholt ausgeführt, solange der Wert der Variable Zaehler kleiner oder gleich 4 ist.

Zeile 21: Zaehler = Zaehler + 1;

Am Ende der Anweisungen innerhalb der Schleife wird der Wert der Variable Zaehler um eins erhöht. Ist der Wert der Variable Zaehler dann kleiner oder gleich 4, springt das Programm zu Zeile 16 und es werden die Anweisungen innerhalb der Schleife wiederholt ausgeführt.

Ist der Wert der Variable Zaehler größer als 4, wird die Ausführung der Anweisungen in Zeile 23 fortgesetzt.

7.5.3 Die while-Schleife

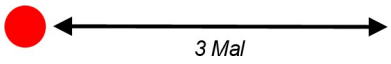
Die Anweisungen in einer while-Schleife werden ausgeführt, solange die Bedingung, die nach dem Schlüsselwort while steht, erfüllt ist.

Alle Anweisungen, die wiederholt ausgeführt werden sollen, sind mit geschwungenen Klammern zusammengefasst.

```
1 while(Bedingung)
2 {
3   Anweisung1;
4   Anweisung2;
5   ...
6 }
```

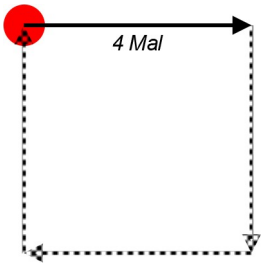
7.5.4 Aufgabe S1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 3 Mal 2 Sekunden mit 85% Motorleistung vorwärts fährt, wieder die gleiche Strecke rückwärts fährt und danach stoppt. Verwende für die mehrfache Ausführung dieses Bewegungsvorgangs eine while-Schleife.



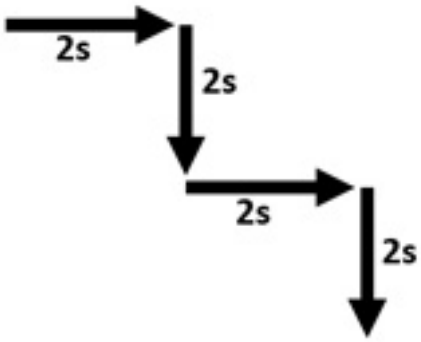
7.5.5 Aufgabe S2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 4 Mal 2 Sekunden geradeaus fährt, sich um 90 Grad nach rechts dreht und somit seine Bahn ein Quadrat beschreibt. Verwende für die vierfache Ausführung des Bewegungsvorgangs *geradeaus - drehen* eine while-Schleife.



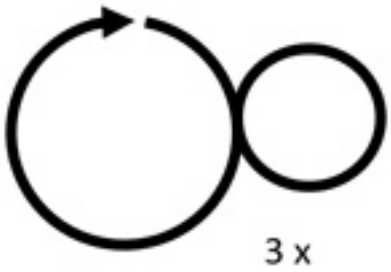
7.5.6 Aufgabe S3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Treppenmuster fährt. Bei den Richtungsänderungen soll er sich immer am Stand drehen. Verwende für sich wiederholende Bewegungsabläufe eine while-Schleife.



7.5.7 Aufgabe S4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) dreimal einen liegenden Achter fährt. Der eine Kreis des Achters soll dabei deutlich größer als der andere Kreis sein. Der Roboter soll dafür zwischen 20 und 25 Sekunden brauchen. Verwende für sich wiederholende Bewegungsabläufe eine while-Schleife.



7.5.8 Aufgabe S5

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 10 Sekunden eine Schlangenlinie fährt und danach stoppt. Sich wiederholende Bewegungsabläufe sollen mit einer while-Schleife realisiert werden.

7.6 Entscheidungsabfrage und Reaktion auf Taster

Meist soll ein Programm nicht nur einmal durchlaufen, sondern solange ausgeführt werden, bis ein bestimmtes Ereignis (z. B. Druck auf einen Taster) eintritt. Dazu verwendet man eine while-Schleife mit einer Bedingung, die immer erfüllt ist. Innerhalb der Schleife muss es dann aber eine Möglichkeit geben, dass die Schleife verlassen wird, damit das Programm beendet werden kann.

7.6.1 Programmbeispiel

Im folgenden Beispiel fährt der Roboter solange vor und zurück bis das Programm durch einen Druck auf den mittleren Taster am EV3 Brick beendet wird.

ACHTUNG: Da ein Schleifendurchlauf aufgrund der beiden Aufrufe der Funktion SLEEP(1000); mindestens 2 Sekunden dauert, muss der Center-Button auch mindestens 2 Sekunden lang gedrückt werden, damit die Bedingung in der if-Abfrage erfüllt ist, wenn diese bei der Ausführung des Programmcodes bzw. eines Schleifendurchlaufs erreicht wird.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren
  SET_OUT(OUT_A,OUT_MOTOR);
  SET_OUT(OUT_B,OUT_MOTOR);

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  // 3. Steuerung des Roboters
  while (1==1)
  {
    WRITE_OUT(OUT_A, MOTOR_POWER, 70);
    WRITE_OUT(OUT_B, MOTOR_POWER, 70);
    SLEEP(1000);
    WRITE_OUT(OUT_A, MOTOR_POWER, -50);
    WRITE_OUT(OUT_B, MOTOR_POWER, -50);
    SLEEP(1000);
    if (READ_BUTTON(BUTTON_CENTER)==1)
    {
      break;
    }
  }

  // 4. Programmende
  EVC_CLOSE();
  return 0;
}
```

Listing 6: Entscheidungsabfrage und Reaktion auf einen Taster

7.6.2 Erklärungen zu diesem Programmbeispiel

Zeile 13: while (1==1) ...

In dieser Zeile beginnt eine while-Schleife mit der Bedingung 1==1. Der Programmcode, der zwischen den geschwungenen Klammern steht, wird solange wiederholt ausgeführt, solange die Bedingung erfüllt ist – also „unendlich“ lange bzw. so lange bis der Akku des EV3 Brick leer ist.

Zeile 21: if (READ_BUTTON(BUTTON_CENTER)==1)

Am Ende der Schleife wird mit einer if-Abfrage ermittelt, ob der mittlere Taster am EV3 Brick gedrückt ist. Dazu wird mit der Funktion READ_BUTTON(BUTTON_CENTER) abgefragt, ob der Taster gedrückt wird. Dies ist dann der Fall, wenn diese Funktion den Wert 1 zurückliefert.

Zeile 23: break;

In die Anweisung ***break*** bewirkt, dass die Schleife verlassen und die Ausführung des Programms in Zeile 26 fortgesetzt wird.

7.6.3 Die if-Abfrage

Eine if-Abfrage entscheidet über den weiteren Programmablauf. Nach dem Schlüsselwort if wird eine Bedingung angegeben, die darüber entscheidet, ob ein Programmcode ausgeführt werden soll oder nicht. Ist die Bedingung erfüllt, werden alle Anweisungen ausgeführt, die mit geschwungenen Klammern zusammengefasst sind.

```
1  if (Bedingung)
2  {
3    Anweisung 1;
4    Anweisung 2;
5    ...
6  }
```

Weiter gibt es auch die Möglichkeit einen Block von Anweisungen festzulegen, die ausgeführt werden sollen, wenn die Bedingung nicht erfüllt ist. Diese stehen nach dem Schlüsselwort else.

```
1  if (Bedingung)
2  {
3    Anweisung 1;
4    Anweisung 2;
5    ...
6  }
7  else
8  {
9    Anweisung 1;
10   Anweisung 2;
11   ...
12 }
```

7.6.4 Vergleichsoperatoren für Bedingungen

In einer Bedingung werden zwei oder mehrere Werte miteinander verglichen. Meist soll überprüft werden ob eine Variable einen bestimmten Wert hat. Für diese Überprüfung können in den Bedingungen einer if-Abfrage und einer while-Schleife folgende Operatoren verwendet werden:

if (a==1) {...}	==	ist gleich	Hat die Variable a den Wert 1?
if (a!=1) {...}	!=	ungleich	Ist der Wert der Variable a ungleich 1?
if (a<4) {...}	<	kleiner	Ist der Wert der Variable a kleiner als 4?
if (a>20) {...}	>	größer	Ist der Wert der Variable a größer als 20?
if (a<=10) {...}	<=	kleiner gleich	Ist der Wert der Variable a kleiner gleich 10?
if (a>=25) {...}	>=	größer gleich	Ist der Wert der Variable a größer gleich 25?

7.6.5 Verknüpfungsoperatoren für Bedingungen

Sollen in einer Bedingung mehrere Bedingungen verknüpft werden, so gibt es dafür die beiden Operatoren && und || für das logische UND und das logische ODER.

Bei einer Verknüpfung mit && - dem logischen UND - ist die Gesamtbedingung erfüllt, wenn beide Bedingungen erfüllt sind.

Bei einer Verknüpfung mit || - dem logischen ODER - gilt die Gesamtbedingung dann als erfüllt, wenn entweder die eine, die andere oder beide Bedingungen erfüllt sind.

if ((a==1)&&(b!=2)) {...}	Hat die Variable a den Wert 1 UND ist der Wert von b ungleich 2?
if ((a==1)||(b==1)) {...}	Hat die Variable a den Wert 1 ODER hat die Variable b den Wert 1?

7.6.6 Aufgabe B1

Erstelle ein Programm, das durch Abfrage der Taster die LED am Display steuert. Wird der linke Taster gedrückt, soll die LED-Anzeige rot leuchten, wird der rechte Taster gedrückt, soll die LED-Anzeige grün leuchten. Wird der mittlere Taster gedrückt, dann soll das Programm beendet werden.

7.7 Grundstruktur mit wiederholter Programmausführung und Exit-Button

Soll Programmcode in einer Endlosschleife wiederholt ausgeführt werden, bis das Programm manuell beendet wird, so kann dies wie im vorigen Abschnitt mit einer Abfrage und Überprüfung eines Tasters gemacht werden.

Damit ein Programm wie gewohnt mit dem Exit-Button (links unter dem Display) beendet werden kann, wurde in EVC eine spezielle Abbruchbedingung für eine while-Schleife implementiert.

7.7.1 Programmbeispiel

Im folgenden Beispiel fährt ein Roboter so lange im Kreis, bis das Programm durch einen Druck auf den Exit-Button am EV3 Brick beendet wird.

ACHTUNG: Wird die Ausführung des Programmcodes innerhalb der while-Schleife mit der Funktion SLEEP(); angehalten, so muss der Exit-Button mindestens so lange gedrückt werden, wie die Ausführung eines Schleifendurchlaufs dauert!

Die Überprüfung der Bedingung der while-Schleife erfolgt jeweils nur nach einem vollständigen Durchlaufen der Schleife.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_OUT(OUT_A,OUT_MOTOR);
7    SET_OUT(OUT_B,OUT_MOTOR);
8
9    // 2. Initialisierung des EV3 Brick
10   EVC_INIT();
11
12   // 3. Steuerung des Roboters
13
14   WRITE_OUT(OUT_A, MOTOR_POWER, 70);
15   WRITE_OUT(OUT_B, MOTOR_POWER, 40);
16
17   while (NOTEXITBUTTON)
18   {
19
20   }
21
22
23   // 4. Programmende
24   EVC_CLOSE();
25   return 0;
26 }
```

Listing 7: wiederholte Programmausführung und Exit-Button

7.7.2 Erklärungen zu diesem Programmbeispiel

Zeile 17: while (NOTEXITBUTTON) ...

In dieser Zeile beginnt eine while-Schleife mit einer Bedingung, die so lange erfüllt ist, bis der Exit-Button am EV3 Brick gedrückt wird.

7.8 Der Touch-Sensor

Der Touch-Sensor ist dafür ausgelegt mechanische Berührungen rückmelden zu können. Der Sensor liefert den Wert 1, wenn dieser gedrückt ist und 0, wenn dieser nicht gedrückt ist.

7.8.1 Programmbeispiel

Das Einlesen von Sensoren erfolgt in der Regel innerhalb einer Schleife. Oft wird der Zustand oder der Wert eines Sensors eingelesen und dann mit einer Entscheidungsabfrage darauf reagiert.

Am Eingang 1 ist ein EV3 Touch-Sensor angeschlossen. Wird der Sensor gedrückt, so leuchtet die LED am EV3 Brick grün, wird der Sensor nicht gedrückt, leuchtet die LED rot.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_IN(IN_1, IN_EV3_TOUCH);
7
8    // 2. Initialisierung des EV3 Brick
9    EVC_INIT();
10
11   // 3. Steuerung des Roboters
12   while (NOTEXITBUTTON)
13   {
14     if (READ_IN(IN_1)==1)
15     {
16       WRITE_LED(LED_GREEN);
17     }
18     else
19     {
20       WRITE_LED(LED_RED);
21     }
22   }
23
24   // 4. Programmende
25   EVC_CLOSE();
26   return 0;
27 }
```

Listing 8: Entscheidungsabfrage und Touch-Sensor

7.8.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_IN(IN_1, IN_EV3_TOUCH);

Mit der Funktion ***SET_IN(port, value)*** wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist.

Der Parameter ***port*** beschreibt die Nummer des Eingangs: IN_1, IN_2, IN_3 oder IN_4.

Mit dem Parameter ***value*** wird der Typ des Sensors festgelegt: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Zeile 14: if (READ_IN(IN_1)==1)

In der Bedingung dieser Entscheidungsabfrage wird die Funktion ***READ_IN(port)*** aufgerufen. Als Rückgabewert liefert diese Funktion den aktuellen Wert des angeschlossenen Sensors. Dieser kann bei einem Touch-Sensor 0 oder 1 sein.

Anschließend wird überprüft, ob der Rückgabewert gleich 1 ist. Das bedeutet bei einem Touch-Sensor, dass dieser gedrückt ist.

Wenn ja, wird die Funktion ***WRITE_LED(LED_GREEN)*** aufgerufen und die LED leuchtet grün.

Ist die Bedingung nicht erfüllt, werden die Anweisungen ausgeführt, die zwischen den geschwungenen Klammern nach dem Schüsselwort else stehen – hier: WRITE_LED(LED_RED)

7.8.3 Aufgabe T1

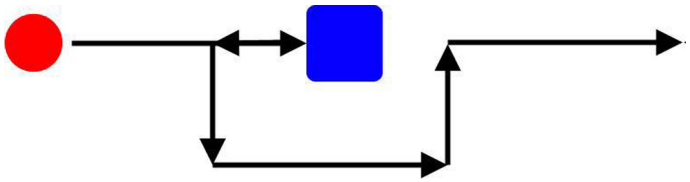
Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) so lange geradeaus fährt, bis er auf ein Hindernis trifft. Dann soll er 1,5 Sekunden gerade zurück fahren und stehen bleiben.

7.8.4 Aufgabe T2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) in einer Endlosschleife immer wieder so lange geradeaus fährt, bis er auf ein Hindernis trifft. Dann soll er 1,5 Sekunden gerade zurück fahren, kurz stehen bleiben und wieder geradeaus fahren, bis er wieder auf das Hindernis trifft. Das Programm soll durch einen Druck auf den Exit-Button beendet werden können.

7.8.5 Aufgabe T3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) gerade auf ein Hindernis zufährt und bei Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll danach seinen Weg wie zuvor fortsetzen.



7.8.6 Aufgabe T4

Ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) soll in einem durch Wände abgeschlossenen Bereich nach einer bestimmten Zeit die gesamte Fläche abfahren (Rasenmäherroboter). Dazu soll der Roboter beim Berühren einer Wand mit dem Tast-Sensor umdrehen und in die entgegengesetzte Richtung fahren. Dabei soll sich der Roboter nicht genau um 180 Grad drehen. Zusätzlich soll es möglich sein, den Roboter durch einen Druck auf den Exit-Button abzustoppen.

7.9 Ausgabe von Zahlen und der Licht-Sensor

Der Farb/Licht-Sensor kann in zwei unterschiedlichen Modi betrieben werden, sodass dieser entweder die Intensität des ausgesandten und reflektierten Lichts misst oder Farben ermittelt.

Ist dieser als Licht-Sensor initialisiert, so liefert dieser Werte zwischen 0% und 100% zurück, was der Intensität des reflektierten Lichts entspricht.

7.9.1 Programmbeispiel

Am Eingang 1 ist ein EV3 Licht-Sensor angeschlossen. Der Sensor liefert je nach Intensität und Farbe des reflektierten Lichts unterschiedliche Werte zurück, die am Display angezeigt werden.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_IN(IN_1, IN_EV3_LIGHT);
7
8    // 2. Initialisierung des EV3 Brick
9    EVC_INIT();
10
11   // 3. Steuerung des Roboters
12   int Wert;
13   LCD_DRAW_TEXT(1, 5, "Light-Sensor:");
14   while (NOTEXITBUTTON)
15   {
16     Wert = READ_IN(IN_1);
17     LCD_DRAW_INT(14, 5, Wert);
18   }
19
20   // 4. Programmende
21   EVC_CLOSE();
22   return 0;
23 }
```

Listing 9: Ausgabe von Zahlen und Licht-Sensor

7.9.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_IN(IN_1, IN_EV3_LIGHT);

Mit der Funktion ***SET_IN(port, value)*** wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist – in diesem Fall ein Farb/Licht-Sensor der im Modus zur Messung der Intensität des reflektierten Lichts betrieben wird.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den Typ des angeschlossenen Sensors: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Zeile 12: int Wert;

Definition der Integer-Variable Wert, in der später der Wert des Licht-Sensors abgelegt wird.

Zeile 13: LCD_DRAW_TEXT(1, 5, “Light-Sensor:“);

In Zeile 5 wird der Text “Light-Sensor:“ angezeigt.

Zeile 16: Wert = READ_IN(IN_1);

Mit der Funktion ***READ_IN(port)*** wird der aktuelle Wert des Sensors ausgelesen und in der Variable Wert abgelegt.

Zeile 17: LCD_DRAW_INT(14, 5, Wert);

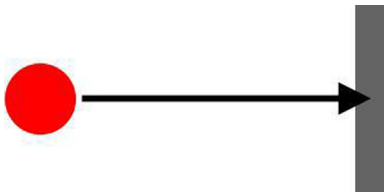
Mit der Funktion ***LCD_DRAW_INT(spalte, zeile, value)*** kann der Wert von Integer-Variablen am Display angezeigt werden. Diese Funktion wandelt dabei den Zahlenwert zunächst in Textzeichen um, die dann letztendlich am Display ausgegeben werden.

Für ***spalte*** kann ein Wert zwischen 1 und 22 gewählt werden, für ***zeile*** ein Wert zwischen 1 und 11.

Als dritter Parameter wird dieser Funktion der Name einer Integer-Variable übergeben.

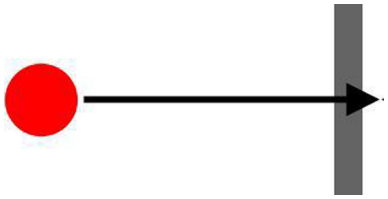
7.9.3 Aufgabe L1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) so lange geradeaus fährt, bis der Lichtsensor auf eine schwarze Linie am Boden trifft. Dort soll der Roboter stehen bleiben. (Grafik Anhang A)



7.9.4 Aufgabe L2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) so lange geradeaus fährt, bis er auf eine schwarze Linie am Boden trifft. Erst wenn der Lichtsensor die Linie überfahren hat und wieder den weißen Untergrund erreicht hat, soll der Roboter stehen bleiben. (Grafik Anhang A)

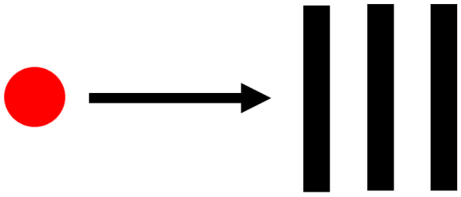


7.9.5 Aufgabe L3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren, einem nach unten gerichteten Licht-Sensor und einem an einer beliebigen Stelle montierten Touch-Sensor), der auf einer strukturierten Oberfläche (z. B. aufgeklebte schwarze Linien auf einer weißen Oberfläche) in Kreisbahnen fährt und der aktuelle Wert des Lichtsensors am Display anzeigt wird. Wird der Touch-Sensor gedrückt, soll der Roboter stehen bleiben. (Grafik Anhang A)

7.9.6 Aufgabe L4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor) auf einer weißen Oberfläche in Richtung von drei parallelen schwarzen Linien fährt. Der Roboter soll beim Überfahren die Linien mitzählen und die Anzahl am Display anzeigen. (Grafik Anhang A)



7.10 Ausgabe von Zahlen und der Farb-Sensor

Ist der Farb/Licht-Sensor so konfiguriert, dass er Farben erkennen kann, liefert dieser Werte zwischen 0 und 7 zurück. Diese stehen für bestimmte Farbbezeichnungen.

7.10.1 Programmbeispiel

An den Eingang 1 ist ein EV3 Farb/Licht-Sensor angeschlossen. Der Sensor ist als Farbsensor konfiguriert und am Display sollen die erkannten Farben angezeigt werden.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_IN(IN_1, IN_EV3_COLOR);
7
8    // 2. Initialisierung des EV3 Brick
9    EVC_INIT();
10
11   // 3. Steuerung des Roboters
12   int Wert;
13   LCD_DRAW_TEXT(1, 5, "Color:");
14   while (NOTEXITBUTTON)
15   {
16     Wert = READ_IN(IN_1);
17     if (Wert==0) LCD_DRAW_TEXT(8, 5, "transparent");
18     if (Wert==1) LCD_DRAW_TEXT(8, 5, "schwarz ");
19     if (Wert==2) LCD_DRAW_TEXT(8, 5, "blau   ");
20     if (Wert==3) LCD_DRAW_TEXT(8, 5, "gruen  ");
21     if (Wert==4) LCD_DRAW_TEXT(8, 5, "gelb   ");
22     if (Wert==5) LCD_DRAW_TEXT(8, 5, "rot    ");
23     if (Wert==6) LCD_DRAW_TEXT(8, 5, "weiss  ");
24     if (Wert==7) LCD_DRAW_TEXT(8, 5, "braun  ");
25   }
26
27   // 4. Programmende
28   EVC_CLOSE();
29   return 0;
30 }
```

Listing 10: Farben und Farb-Sensor

7.10.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_IN(IN_1, IN_EV3_COLOR);

Mit der Funktion SET_IN(port, vlaue) wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist – in diesem Fall ein Farb/Licht-Sensor im Modus Farberkennung.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den Typ des angeschlossenen Sensors: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Zeile 12: int Wert;

Definition der Integer-Variable Wert, in der später der Wert des Farb-Sensors abgelegt wird.

Zeile 13: LCD_DRAW_TEXT(1, 5, "Color:");

In Zeile 5 wird der Text "Color-Sensor:" angezeigt.

Zeile 16: Wert = READ_IN(IN_1);

Mit der Funktion READ_IN(port) wird der aktuelle Wert des Sensors ausgelesen und in der Variable Wert abgelegt.

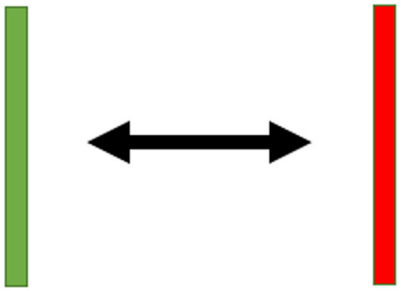
Zeile 17-24: if (Wert==0) LCD_DRAW_TEXT(14, 5, "transparent");

Mit einer if-Abfrage wird der aktuelle Wert des Sensors mit den möglichen Werten verglichen. Jeder Zahl zwischen 0 und 7 ist eine bestimmte Farbbezeichnung zugeordnet. Ausgehend vom aktuellen Wert wird ein Text mit der jeweiligen Farbbezeichnung am Display angezeigt.

Dabei ist zu beachten, dass Zeichen von zuvor angezeigten längeren Wörtern gelöscht werden in dem an diesen Stellen Leerzeichen ausgegeben werden.

7.10.3 Aufgabe F1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Farb/Licht-Sensor) zwischen einer grünen und roten Linie hin- und herfährt. Das Erkennen der jeweiligen Farbe soll dazu führen, dass der Roboter umdreht und in die entgegengesetzte Richtung fährt. Dabei soll der Roboter bei der grünen Linie eine Drehung nach rechts und bei der roten Linie eine Drehung nach links machen (Grafik Anhang A)



7.11 Der Ultraschall-Sensor

Der Ultraschall-Sensor misst Entfernungen mit dem Prinzip der Weg/Zeit-Bestimmung zwischen ausgesandten und reflektierten akustischen Signalen im nicht hörbaren Bereich. Mit dem EV3 Ultraschall-Sensor können Entfernungen zwischen 3 und 150 cm gemessen werden.

7.11.1 Programmbeispiel

An den Eingang 1 ist ein EV3 Ultraschall Sensor angeschlossen. Nach dem Start des Programms fährt der Roboter vorwärts in Richtung einer Wand. Ist die Entfernung zur Wand kleiner als 50 cm, bleibt der Roboter abrupt stehen und das Programm wird beendet.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6    SET_OUT(OUT_A, OUT_MOTOR);
7    SET_OUT(OUT_B, OUT_MOTOR);
8    SET_IN(IN_1, IN_EV3_SONAR);
9
10   // 2. Initialisierung des EV3 Brick
11   EVC_INIT();
12
13   // 3. Steuerung des Roboters
14   WRITE_OUT(OUT_A, MOTOR_POWER, 40);
15   WRITE_OUT(OUT_B, MOTOR_POWER, 40);
16   while (1==1)
17   {
18     if (READ_IN(IN_1)<50)
19     {
20       WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);
21       WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_BRAKE);
22       break;
23     }
24   }
25
26   // 4. Programmende
27   EVC_CLOSE();
28   return 0;
29 }
```

Listing 11: Kollisionsvermeidung mit einem Ultraschall-Sensor

7.11.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_OUT(OUT_A, OUT_MOTOR);

Mit der Funktion ***SET_OUT(port, value)*** wird festgelegt, an welchem Ausgang ein Motor angeschlossen ist.

Mit dem ersten Parameter ***port*** wird der gewünschte Ausgang angegeben: OUT_A, OUT_B, OUT_C oder OUT_D

Der zweite Parameter ***value*** gibt an, dass an dem entsprechenden Ausgang ein Motor angeschlossen ist: OUT_MOTOR

Zeile 8: SET_IN(IN_1, IN_EV3_SONAR);

Mit der Funktion ***SET_IN(port, value)*** wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist – in diesem Fall ein Ultraschall-Sensor.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den Typ des angeschlossenen Sensors: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Zeile 14: WRITE_OUT(OUT_A, MOTOR_POWER, 40);

Die Funktion ***WRITE_OUT(port, mode, value)*** dient zum Ansteuern der Ausgänge.

Mit dem ersten Parameter wird der gewünschte Ausgang festgelegt, hier OUT_A.

Der zweite Parameter gibt an, in welchem Modus der Motor betrieben werden soll: MOTOR_POWER

In diesem Modus dreht sich ein Motor mit einer vorgegebenen Geschwindigkeit. Erfährt der Motor einen Widerstand (z. B. ein Hindernis oder beim Hinauffahren einer schiefen Ebene), so wird dies erkannt und der Motor versucht die Drehgeschwindigkeit konstant zu halten indem er die Leistung je nach Widerstand nachregelt.

Der dritte Parameter steht für die Geschwindigkeit mit der sich der Motor drehen soll. Die Werte können zwischen -100 und 100 liegen. Das Vorzeigen bestimmt die Drehrichtung, der Wert beschreibt eine der möglichen Geschwindigkeiten zwischen 0% und 100%.

Zeile 16: while (1==1) ...

Hier beginnt eine while-Schleife die grundsätzlich so lange läuft, bis das Programm beendet wird.

Zeile 18: if (READ_IN(IN_1)<50) ...

In der Bedingung dieser if-Abfrage wird mit der Funktion READ_IN(port) der aktuelle Wert des Ultraschall-Sensors ausgelesen und überprüft ob dieser Wert kleiner als 50 ist.

Zeile 20: WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);

Wird der Funktion WRITE_OUT(port, mode, value) der Wert MOTOR_STOP übergeben, bleibt der entsprechende Motor stehen.

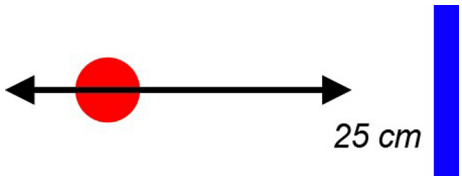
Mit dem dritten Parameter wird festgelegt, ob ein Motor nachläuft (MOTOR_COAST) oder abrupt angehalten werden soll (MOTOR_BRAKE).

Zeile 22: break;

Ist die Bedingung der if-Abfrage erfüllt, werden die beiden Motoren angehalten und auch die while-Schleife verlassen, was in der Folge zum Beenden des Programms führt.

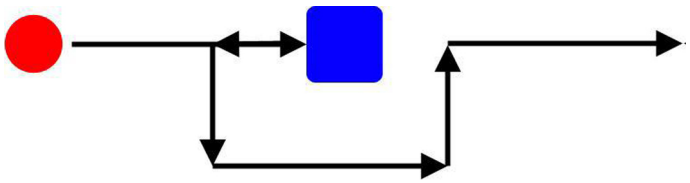
7.11.3 Aufgabe U1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach vorne gerichteten Ultraschall-Sensor an der Vorderseite) so lange geradeaus fährt, bis er sich 25 cm vor einem Objekt entfernt befindet. Dort soll der Roboter für 2 Sekunden stehenbleiben und dann 1 Sekunde zurück fahren.



7.11.4 Aufgabe U2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach vorne gerichteten Ultraschall-Sensor an der Vorderseite) gerade auf ein Hindernis zufährt und 10 cm vor Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll nach dem Hindernis seinen Weg wie zuvor fortsetzen.



7.11.5 Aufgabe U3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem rechtwinkelig zur Seite gerichteten Ultraschall-Sensor) in einem gleichbleibenden Abstand einer Wand entlang fährt. Dazu soll der Abstand mit dem Ultraschallsensor zur Seite gemessen werden.

Grundsätzlich startet der Roboter mit einer konstanten Vorwärtsbewegung. Wird der Abstand zur Wand größer als ein vorgegebener Wert, dann soll der Roboter mit einer leichten Korrektur in Richtung Wand reagieren. Wird der Abstand zu Wand kleiner als ein vorgegebener Wert, dann soll der Roboter sich mit einer leichten Gegenbewegung von der Wand wegbewegen.

7.12 Der Gyro-Sensor

Der Gyro-Sensor ermöglicht es, ausgehend von einem eingebauten Sensor, der Drehbeschleunigungen misst, Drehbewegungen zu ermitteln. Der Sensor liefert die aktuelle Richtung in Grad zurück.

ACHTUNG: Im Gyro-Sensor läuft eine Auto-Kalibrierung ab, sobald der EV3 Brick eingeschaltet wird. In dieser Zeit darf der Gyro-Sensor auf keinen Fall bewegt werden!

Wird der Sensor genau während dieser Kalibrierung bewegt, werden die falschen Werte für den Zustand "still" ermittelt und der Sensor driftet, was man dadurch beobachten kann, dass die vom Sensor gelieferten Werte ohne den Sensor zu bewegen hinauf- oder hinunterzählen.

7.12.1 Programmbeispiel

Am Eingang 1 ist ein EV3 Gyro-Sensor angeschlossen. Der Roboter fährt zunächst für zwei Sekunden vorwärts, dreht sich um 180 Grad und fährt dann wiederum zwei Sekunden vorwärts bis zum ursprünglichen Ausgangspunkt.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren
  SET_OUT(OUT_A, OUT_MOTOR);
  SET_OUT(OUT_B, OUT_MOTOR);
  SET_IN(IN_1, IN_EV3_GYRO);

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  // 3. Steuerung des Roboters
  WRITE_OUT(OUT_A, MOTOR_POWER, 40);
  WRITE_OUT(OUT_B, MOTOR_POWER, 40);
  SLEEP(2000);
  WRITE_OUT(OUT_A, MOTOR_POWER, 20);
  WRITE_OUT(OUT_B, MOTOR_POWER, -20);
  while (1==1)
  {
    if (READ_IN(IN_1)>180)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 40);
      break;
    }
  }
  SLEEP(2000);

  // 4. Programmende
  EVC_CLOSE();
  return 0;
}
```

Listing 12: Umdrehen mit Hilfe eines Gyro-Sensors

7.12.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_OUT(OUT_A, OUT_MOTOR);

Mit der Funktion ***SET_OUT(port, value)*** wird festgelegt, an welchem Ausgang ein Motor angeschlossen ist.

Mit dem ersten Parameter ***port*** wird der gewünschte Ausgang angegeben: OUT_A, OUT_B, OUT_C oder OUT_D

Der zweite Parameter ***value*** gibt an, dass an dem entsprechenden Ausgang ein Motor angeschlossen ist: OUT_MOTOR

Zeile 8: SET_IN(IN_1, IN_EV3_GYRO);

Mit der Funktion ***SET_IN(port, value)*** wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist – in diesem Fall ein Gyro-Sensor.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den Typ des angeschlossenen Sensors: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Zeile 14: WRITE_OUT(OUT_A, MOTOR_POWER, 40);

Die Funktion ***WRITE_OUT(port, mode, value)*** dient zum Ansteuern der Ausgänge.

Mit dem ersten Parameter wird der gewünschte Ausgang festgelegt, hier OUT_A.

Der zweite Parameter gibt an, in welchem Modus der Motor betrieben werden soll: MOTOR_POWER

In diesem Modus dreht sich ein Motor mit einer vorgegebenen Geschwindigkeit. Erfährt der Motor einen Widerstand (z. B. ein Hindernis oder beim Hinauffahren einer schiefen Ebene), so wird dies erkannt und der Motor versucht die Drehgeschwindigkeit konstant zu halten indem er die Leistung je nach Widerstand nachregelt.

Der dritte Parameter steht für die Geschwindigkeit mit der sich der Motor drehen soll. Die Werte können zwischen -100 und 100 liegen. Das Vorzeigen bestimmt die Drehrichtung, der Wert bezeichnet eine der möglichen Geschwindigkeiten zwischen 0% und 100%.

Zeile 16: SLEEP(2000);

Die weitere Ausführung des Programmcodes wird für zwei Sekunden angehalten.

Zeile 19: while (1==1) ...

Hier beginnt eine while-Schleife die grundsätzlich so lange läuft, bis das Programm beendet wird.

Zeile 21: if (READ_IN(IN_1)>180) ...

In der Bedingung dieser if-Abfrage wird mit der Funktion ***READ_IN(port)*** der Wert des Gyro-Sensors ausgelesen und überprüft, ob dieser Wert größer als 180 ist.

Zeile 25: break;

Ist die Bedingung der if-Abfrage erfüllt, wird die while-Schleife verlassen.

Zeile 29: WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);

Wird der Funktion ***`WRITE_OUT(port, mode, value)`*** der Wert MOTOR_STOP übergeben, bleibt der entsprechende Motor stehen.

Mit dem dritten Parameter wird festgelegt, ob ein Motor nachläuft (MOTOR_COAST) oder abrupt angehalten werden soll (MOTOR_BRAKE).

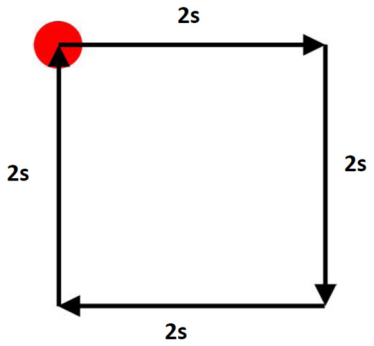
7.12.3 Aufgabe G1

Ein Roboter (Standardroboter mit zwei Motoren und einem Gyro-Sensor) soll zunächst für 2 Sekunden geradeaus fahren. Nach dieser Zeit soll er sich beginnen (langsam!) am Stand zu drehen. Unter Verwendung des Werts des Gyro-Sensors soll sich der Roboter genau 90 Grad drehen, fährt dann wieder für 2 Sekunden geradeaus und bleibt stehen.

Zeige dazu als Hilfe den aktuellen Wert des Gyro-Sensors auf dem Display an!

7.12.4 Aufgabe G2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad und einem Gyro-Sensor) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt. Die sich dabei wiederholenden Bewegungsabläufe sollen mit einer while-Schleife umgesetzt werden und für die exakten Drehungen soll der Gyro-Sensor zum Einsatz kommen.



7.13 Der LineArray-Sensor

Das Sensor-Array der Firma Mindsensors ermöglicht es 8 unterschiedliche Intensitäten von reflektiertem Licht auf einem Untergrund mit nur einem Sensor zu ermitteln.

An den acht Messpunkten erhält man Werte zwischen 0% und 100%, was der Intensität des reflektierten Lichts entspricht.

7.13.1 Programmbeispiel

Am Eingang 1 ist ein Mindsensors LineArray-Sensor angeschlossen. Am Display werden die acht Werte des Sensors ausgegeben.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren
  SET_IN(IN_1,IN_MS_LINEARRAY);

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  int Value1 = 0;
  int Value2 = 0;
  int Value3 = 0;
  int Value4 = 0;
  int Value5 = 0;
  int Value6 = 0;
  int Value7 = 0;
  int Value8 = 0;

  // 3. Steuerung des Roboters

  while (NOTEXITBUTTON)
  {
    Value1 = READ_IN_LINEARRAY(IN_1,1);
    Value2 = READ_IN_LINEARRAY(IN_1,2);
    ...
    Value8 = READ_IN_LINEARRAY(IN_1,8);

    LCD_DRAW_INT(1,1,Value1);
    LCD_DRAW_INT(1,2,Value2);
    ...
    LCD_DRAW_INT(1,8,Value8);
  }

  // 4. Programmende

  EVC_CLOSE();
  return 0;
}
```

Listing 13: Ausgabe von Zahlen und Licht-Sensor

7.13.2 Erklärungen zu diesem Programmbeispiel

Zeile 6: SET_IN(IN_1, IN_MS_LINEARRAY);

Mit der Funktion ***SET_IN(port, value)*** wird festgelegt, welcher Sensor an einem Eingang angeschlossen ist – in diesem Fall ein LineaArray der Firma Mindsensors zur Messung der Intensität von reflektiertem Licht an acht Messpunkten.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den Typ des angeschlossenen Sensors: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO, IN_MS_LINEARRAY

Zeile 11: int Value1 = 0;

Definition der Integer-Variable Value1, in der später der Wert eines Messpunkts des LineArrays abgelegt wird.

Zeile 24: Value1 = READ_IN_LINEARRAY(IN_1,1);

Mit der Funktion ***READ_IN_LINEARRAY(port,channel)*** wird der aktuelle Wert eines Messpunkts des Sensors ausgelesen und in der Variable Value1 abgelegt.

Mit dem ersten Parameter wird der Port festgelegt: IN_1, IN_2, IN_3 oder IN_4

Der zweite Parameter beschreibt den gewünschten Messpunkt des angeschlossenen Sensors: 1, 2, ..., 8

8 Autonome Roboter

Ein Roboter erfüllt seine Aufgaben grundsätzlich durch das Erfassen seiner Umgebung mit Sensoren, der darauffolgenden Verarbeitung von Informationen und der meist der daraus resultierenden Bewegung von mechanischen Bauteilen, sogenannten Aktoren.

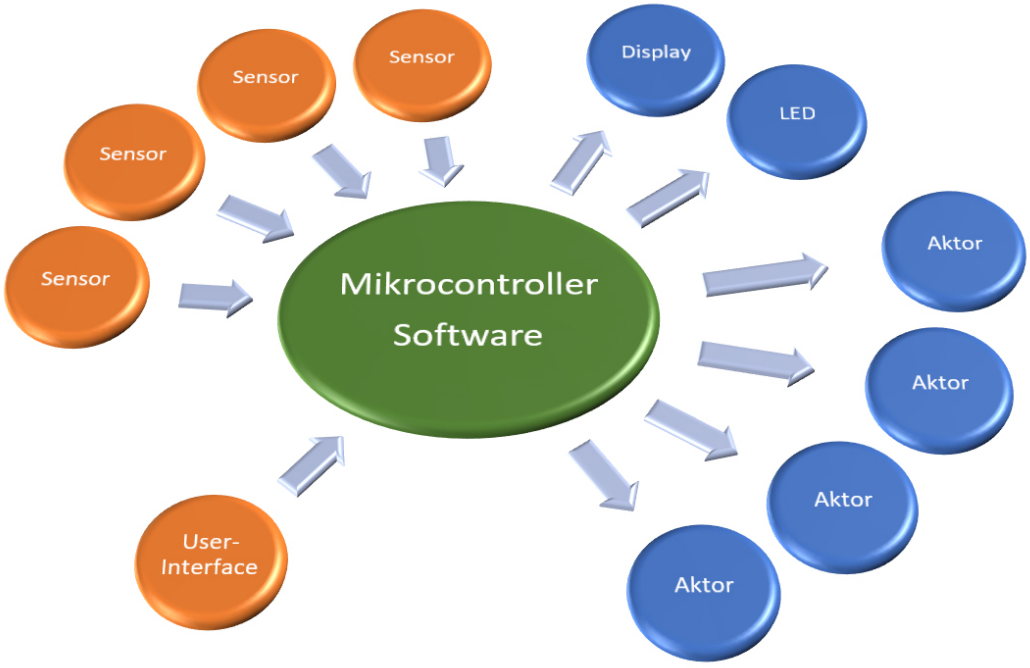


Abbildung 9: Informationsverarbeitung bei dem EV3 Robotik System

Bei einem EV3 Roboter werden die Informationen von den maximal vier Sensoren und dem User-Interface (den Buttons unter dem Display) von einem Mikrocontroller und der Software verarbeitet. Ausgehend von den Informationen der Sensoren und deren Verarbeitung von der Software werden die maximal vier Aktoren (Motoren) angesteuert und auch optische Anzeigen am Display oder der Led generiert.

Autonome Roboter sollen dabei nicht nur vorgegebene Handlungsabläufe reproduzieren können, wie dies zum Beispiel Getränkeautomaten oder Industrieroboter in Produktionsprozessen tun, sondern aufgrund sich ändernder Umgebungsbedingungen mehr oder wenig selbstständig Entscheidungen über weitere Aktionen treffen.

Neben der Anforderung Sensoren zu finden oder zu entwickeln, welche die Umwelt eines Roboters bestmöglich erfassen können, liegt eine große Herausforderung vor allem auch darin die Informationen eines Sensors nicht nur so schnell wie möglich, sondern auch unter Einbeziehung aller anderen vorliegenden Informationen zu verarbeiten.

Daraus können sich auch bereits bei wenigen Sensoren sehr komplexe Aufgabenstellungen ergeben, die bei der Erstellung einer Software berücksichtigt werden müssen, damit der Roboter auf alle möglichen Situationen, die sich aus den Sensor-Informationen ergeben, richtig reagieren kann.

Der nächste Schritt bei der Informationsverarbeitung wäre dann hin zu selbstlernenden Systemen die bei in der Zukunft liegenden Entscheidungsprozessen die Ergebnisse und Auswirkungen von früher getroffenen Entscheidungen mitberücksichtigen und sich somit im Laufe der Zeit immer mehr und mehr verbessern.

8.1 Grundstruktur eines EVC-Programms für autonome Roboter

Damit ein autonomes System optimal auf seine Umwelt und Änderungen von Situationen reagieren kann, genügt es nicht nur ein solches mit einer Vielzahl von Sensoren auszustatten. Vielmehr muss auch die Software, welche die Informationen verarbeitet in der Lage sein, alle diese Daten so schnell wie möglich auszuwerten, um daraus brauchbare und verlässliche Informationen für Entscheidungsprozesse zu erhalten. Dies setzt auch voraus, dass die Ausführung des Programmcodes an keiner Stelle angehalten werden darf.

Es muss gewährleistet sein, dass die Informationen jedes Sensors zu jeder Zeit aktuell zur Verfügung stehen und daraus umgehend Entscheidungen für die nächsten Aktionen getroffen werden - unabhängig davon, welche Aktionen das System gerade ausführt.

8.1.1 Verzicht auf die Verwendung der Funktion SLEEP()

Geht man von einer Grundstruktur aus, in welcher der gesamte Programmcode eines Roboters innerhalb einer while-Schleife wiederholt ausgeführt wird, so kann ein Roboter umso besser auf seine Umwelt reagieren und seine Aufgaben erfüllen, je kürzer ein Durchlauf dieser while-Schleife dauert.

Während der Ausführung des Programmcodes darf dieser innerhalb der while-Schleife also zu keiner Zeit angehalten werden, auch wenn es erforderlich ist, den Roboter zeitgesteuerte Abläufe ausführen zu lassen.

Im folgenden Codebeispiel ist die Grundstruktur eines EVC-Programms für einen autonomen Roboter zu sehen. Damit der Programmcode innerhalb der while-Schleife so schnell wie möglich ausgeführt wird, darf darin die Ausführung auf keinen Fall mit der Funktion ***SLEEP()*** angehalten werden.

Auch muss unbedingt vermieden werden z. B. mit einer weiteren while-Schleife, mit deren Bedingung auf einen bestimmten Wert eines Sensors gewartet wird, die Ausführung des Programmcodes so lange anzuhalten, bis dieses Ereignis eintrifft.

Aktuelle Informationen anderer Sensoren würden in dieser Zeit nicht abgefragt und neue Informationen auch nicht ausgewertet werden!

Grundsätzlich gliedert sich der Programmcode in drei Abschnitte: Einlesen der aktuellen Werte der Sensoren, Informationsverarbeitung und Ansteuern der Aktoren

- **Einlesen der aktuellen Werte der Sensoren**
  Zu Beginn werden die aktuellen Werte aller Sensoren ermittelt.
- **Informationsverarbeitung**
  Ausgehend von den Informationen der Sensoren werden diese analysiert und Entscheidungen für folgende Aktionen getroffen.
- **Ansteuern der Aktoren**
  Basierend auf den nächsten ermittelten Schritten werden die Aktoren angesteuert.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration der angeschlossenen Sensoren und Motoren
6
7    // 2. Initialisierung des EV3 Brick
```

```
8   EVC_INIT();
9
10  // 3. Steuerung des Roboters
11
12  while (NOTEXITBUTTON)
13  {
14    // Einlesen der aktuellen Werte der Sensoren
15
16    // Informationsverarbeitung
17
18    // Ansteuern der Aktoren
19  }
20
21  // 4. Programmende
22  EVC_CLOSE();
23
24  return 0;
25 }
```

Listing 14: Grundstruktur eines EVC-Programms für einen autonomen Roboter

8.1.2 Erklärungen zu diesem Programmbeispiel

Zeile 14-19: Innerhalb dieses Codeblocks muss gewährleistet sein, dass dieser so schnell wie möglich und ohne Unterbrechungen ausgeführt wird. Die Funktion ***SLEEP()*** darf hier nicht verwendet werden.

8.2 Verwendung des Counters

Um dennoch zeitgesteuerte Abläufe implementieren zu können, steht in EVC ein Counter zur Verfügung. Dieser Counter zählt im Hintergrund jede Zehntelsekunde um eins hoch, ohne dass dabei die Ausführung des Programmcodes innerhalb der while-Schleife angehalten wird.

Man kann sich diesen Counter auch wie eine Art Stoppuhr vorstellen, die mit Start des Programms zu laufen beginnt und auf der man stets die vergangene Zeit seit dem Programmstart in Zehntelsekunden ablesen kann.

Der aktuelle Wert des Counters wird in der Integer-Variable ***COUNTER*** zur Verfügung gestellt. Auch ist es möglich den Counter wieder auf 0 zu setzen, indem man der Variable ***COUNTER*** den Wert *0* zuweist.

Im folgenden Beispiel wird der aktuelle Wert des Counters am Display angezeigt. Durch Drücken des Center-Buttons kann der Counter zurückgesetzt werden. Auch kann das Programm mit dem Exit-Button jederzeit ohne Verzögerung beendet werden.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5  // 1. Konfiguration
6
7  // 2. Initialisieren des Brick
8  EVC_INIT();
9
10 // 3. Programm - Roboter
11 LCD_DRAW_TEXT(3,5,"COUNTER: ");
12 while(NOTEXITBUTTON)
13 {
14 LCD_DRAW_INT(15,5,COUNTER);
15 if(READ_BUTTON(BUTTON_CENTER)==1)
16 {
17 COUNTER = 0;
18 }
19 }
20
21 // 4. Programmende - Freigeben der Hardwarezugriffe
22 EVC_CLOSE();
23 return 0;
24 }
```

Listing 15: Anzeige des aktuellen Werts des Counters

8.2.1 Erklärungen zu diesem Programmbeispiel

Zeile 14: Anzeige des Werts der Integer-Variable ***COUNTER*** in Zeile 5 an Position 15. Dieser Wert wird pro Zehntelsekunde automatisch um eins erhöht.

Zeile 17: Wurde der Center-Button gedrückt, so wird der Variable ***COUNTER*** der Wert *0* zugewiesen und somit der Counter zurückgesetzt, womit dieser wieder bei 0 zu zählen beginnt.

8.3 Zeitgesteuerte Bewegungsabläufe unter Verwendung des Counters

Möchte man zeitgesteuerte Bewegungsabläufe realisieren, ohne dass dabei die Ausführung des Programmcodes mit der Funktion ***SLEEP()*** angehalten wird, so kann man dazu den Counter verwenden.

Im diesem Beispiel soll ein Roboter eine Sekunde nach Programmstart mit einer Leistung von 50% für 3 Sekunden vorwärts fahren. Nachdem der Roboter 4 Sekunden lang stehengeblieben ist, soll dieser 4 Sekunden mit einer Leistung von 70% rückwärts fahren bevor das Programm beendet wird.

Um diese zeitabhängige Steuerung der Motoren umzusetzen, werden mit if-Abfragen die entsprechenden Zeitintervalle seit dem Start des Programms abgefragt und darauf basierend die jeweiligen Motorleistungen gesetzt.

Die Anordnung der if-Abfragen muss dabei nicht der Reihenfolge des zeitlichen Ablaufs folgen, sondern könnte auch beliebig durchmischt werden und würde zu dem selben Bewegungsablauf führen.

Dieses Programm kann im Gegensatz zum Einsatz der Funktion ***SLEEP()*** jederzeit mit dem Exit-Button beendet werden, da die Ausführung des Programmcodes an keiner Stelle angehalten wird.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration
  SET_OUT(OUT_A, OUT_MOTOR);
  SET_OUT(OUT_B, OUT_MOTOR);
  // 2. Initialisieren des Brick
  EVC_INIT();
  // 3. Programm - Roboter
  while(NOTEXITBUTTON)
  {
    if (COUNTER==10)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 50);
      WRITE_OUT(OUT_B, MOTOR_POWER, 50);
    }
    if (COUNTER==40)
    {
      WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_COAST);
      WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_COAST);
    }
    if (COUNTER==80)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, -70);
      WRITE_OUT(OUT_B, MOTOR_POWER, -70);
    }
    if (COUNTER==120)
    {
      break;
    }
  }
  // 4. Programmende - Freigeben der Hardwarezugriffe
  EVC_CLOSE();
  return 0;
}
```

Listing 16: Zeitgesteuerter Bewegungsablauf unter Verwendung des Counters

8.3.1 Erklärungen zu diesem Programmbeispiel

Zeile 13-17: Mit dieser if-Abfrage wird überprüft, ob der aktuelle Wert der Variable *COUNTER* gleich *10* ist. Dies bedeutet, dass seit Start des Programms (bei dem der Counter von 0 zu zählen beginnt) eine Sekunde vergangen ist. Zu diesem Zeitpunkt werden die Leistungen für beide Motoren auf 50% gesetzt.

Zeile 28-31: Die Bedingung dieser if-Abfrage ist dann erfüllt, wenn der Wert der Integer-Variable *COUNTER* den Wert *120* hat, also seit dem Programmstart 12 Sekunden vergangen sind. Zu diesem Zeitpunkt wird die while-Schleife mit ***break*** verlassen und das Programm somit beendet.

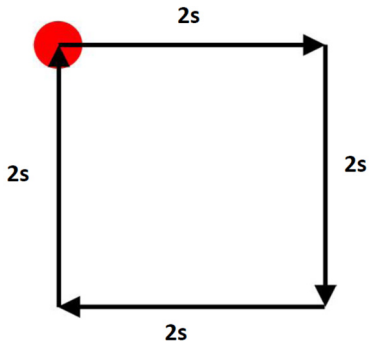
8.3.2 Aufgabe C1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt.

Nachdem das Programm am Roboter gestartet wurde, soll die Fahrt des Roboters aber erst nach 3 Sekunden beginnen. Für die Zeitintervalle soll der Counter zum Einsatz kommen und das Programm automatisch beim Erreichen der Zielposition beendet werden. Während dem Abfahren des Pfades soll es möglich sein, das Programm jederzeit durch Drücken des Exit-Buttons zu beenden.

Diese Aufgabe soll auf zwei Arten gelöst werden:

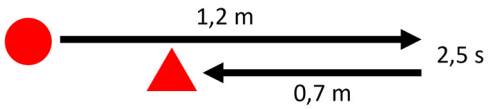
- Der gesamte Bewegungsablauf soll mit einem durchgehenden zeitlichen Verlauf basierend auf den Counter-Werten im Programmcode abgebildet werden.
- Die Bewegung (2s vorwärts und Drehung um 90 Grad), die vier Mal hintereinander gleich abläuft, soll mit einer while-Schleife und einer Zählvariable so implementiert werden, dass der jeweils aktuelle Wert der Zählvariable für die Bestimmung der Zeitintervalle des Counters verwendet wird.



8.3.3 Aufgabe C2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) genau 1,2 Meter geradeaus fährt, 2,5 Sekunden stehen bleibt und dann 0,7 Meter rückwärts fährt.

Nachdem das Programm am Roboter gestartet wurde, soll der Roboter gleich losfahren. Die vorgegebenen Wegstrecken sollen unter Verwendung des Counters gefahren werden, das Programm soll aber nicht beendet werden, wenn der Roboter am Zielpunkt angekommen ist. Das Programm muss jederzeit durch Drücken des Exit-Buttons beendet werden können.



8.4 Exakte Drehung eines Roboters mit dem Gyro-Sensor

Soll ein Roboter eine exakte Drehung um einen bestimmten Winkel durchführen, so kann diese exakter ausgeführt werden, wenn man den Gyro-Sensor verwendet und nicht nur die Motoren mit unterschiedlichen Geschwindigkeiten für eine bestimmte Zeit drehen lässt.

Da sich der Gyro-Sensor nicht zurücksetzen lässt, sondern als Wert immer den absoluten Drehwinkel seit dem Einschalten des EV3 Bricks liefert, muss zu Beginn einer Drehung der aktuelle Winkel bestimmt und in der Folge als Referenz für die gewünschte Drehung verwendet werden.

ACHTUNG: Im Gyro-Sensor läuft eine Auto-Kalibrierung ab, sobald der EV3 Brick eingeschaltet wird. In dieser Zeit darf der Gyro-Sensor auf keinen Fall bewegt werden!

Im folgenden Beispiel wird vor Beginn der Drehung des Roboters um die Hochachse der aktuelle Wert des Gyro-Sensors ermittelt und in der Variable ***GyroStartWert*** abgelegt.

Innerhalb der while-Schleife wird zunächst der aktuelle Wert des Gyro-Sensors eingelesen. Wenn die Drehung genau 90 Grad betragen soll, wird in einer if-Abfrage während der Drehung überprüft, ob der aktuelle Wert gleich dem Startwert plus 90 Grad ist.

Da es sein könnte, dass die Werte des Gyro-Sensors nicht kontinuierlich in 1-Grad-Schritten zurückgeliefert werden, oder sich der Roboter zwischen zwei Abfragen mehr als 1 Grad gedreht hat, sollte man als Vergleichsoperator **>=** und nicht nur **==** verwenden. Somit würde die Bedingung der if-Abfrage auch erfüllt sein wenn der Gyro-Sensor z. B. zunächst den Wert 89 Grad und beim nächsten Auslesen den Wert 91 liefert.

Das Programm kann auch jederzeit mit dem Exit-Button beendet werden, da der Programmcode in der while-Schleife kontinuierlich ausgeführt und an keiner Stelle angehalten wird.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration
  SET_IN(IN_1,IN_EV3_GYRO);
  SET_OUT(OUT_A,OUT_MOTOR);
  SET_OUT(OUT_B,OUT_MOTOR);

  // 2. Initialisieren des Brick
  EVC_INIT();

  // 3. Programm - Roboter
  int GyroStartWert = READ_IN(IN_1);
  int GyroAktuellerWert = 0;

  while(NOTEXITBUTTON)
  {
    GyroAktuellerWert = READ_IN(IN_1);
    WRITE_OUT(OUT_A, MOTOR_POWER, -20);
    WRITE_OUT(OUT_B, MOTOR_POWER, 20);
    if(GyroAktuellerWert>=GyroStartWert+90)
    {
      WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);
      WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_BRAKE);
      break;
    }
  }
```

```
29    // 4. Programmende - Freigeben der Hardwarezugriffe
30    EVC_CLOSE();
31
32    return 0;
33  }
```

Listing 17: Exakte Drehung mit dem Gyro-Sensor ausgehend von einer beliebigen Startrichtung

8.4.1 Erklärungen zu diesem Programmbeispiel

Zeile 14: Hier wird die Integer-Variable ***GyroStartWert*** definiert und dieser dann der aktuelle Wert des Gyro-Sensors bei Programmstart zugewiesen.

Zeile 15: Definition der Integer-Variable ***GyroAktuellerWert***, in der innerhalb der while-Schleife der aktuelle Wert des Gyro-Sensors während der Drehbewegung abgelegt wird.

Zeile 19: Einlesen des aktuellen Werts des Gyro-Sensors und Ablegen in der Variable ***GyroAktuellerWert***.

Zeile 20-21: Drehen der beiden Motoren in entgegengesetzte Richtung mit der Geschwindigkeit 20%.

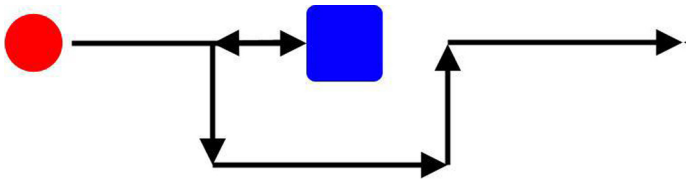
Zeile 22-27: Wenn der aktuelle Wert des Gyro-Sensors größer oder gleich dem zu Programmstart ermittelten Wert des Gyro-Sensors plus 90 Grad ist, werden die beiden Motoren angehalten, mit ***break*** die while-Schleife verlassen und somit das Programm beendet.

8.4.2 Aufgabe GC1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren, einem Touch-Sensor an der Vorderseite und einem Gyro-Sensor) gerade auf ein Hindernis zufährt und bei Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll danach seinen Weg wie zuvor fortsetzen.

Für die Drehungen des Roboters soll der Gyro-Sensor zum Einsatz kommen und dazu der aktuelle Wert des Sensors beim Berühren des Hindernisses als Ausgangswert dienen.

Für die zu fahrenden Strecken beim Ausweichmanöver um das Hindernis soll der Counter verwendet werden.



8.5 Exakte Beibehaltung der Fahrrichtung mit dem Gyro-Sensor

Weist man bei einem Roboter, der von zwei Motoren angetrieben wird, beiden Motoren die gleiche Geschwindigkeit oder die gleiche Leistung zu, so wird der Roboter aufgrund von produktionsbedingten Unterschieden in den Motoren bzw. den darin enthaltenen mechanischen und elektronischen Komponenten über längere Strecken nie genau geradeaus fahren.

Soll ein Roboter seine Fahrtrichtung exakt beibehalten, so kann dazu der Gyro-Sensor verwendet werden. Mit Hilfe dieses Sensors wird laufend die aktuelle Richtung ermittelt und mit der gewünschten Richtung verglichen. Fall es dabei eine Abweichung gibt, wird die Geschwindigkeit der Motoren angepasst um den Roboter wieder in die gewünschte Richtung zurückzudrehen.

In diesem Codebeispiel wird zu Beginn des Programms die aktuelle Richtung ermittelt in die der Roboter fahren soll und in der Variable ***ZielWert*** abgelegt.

In der while-Schleife wird zunächst der aktuelle Wert des Gyro-Sensors eingelesen und der Variable ***AktuellerWert*** zugewiesen. In der Folge wird in if-Abfragen überprüft, ob die aktuelle Richtung links oder rechts von der gewünschten Richtung abweicht. Ist dies der Fall, wird die Geschwindigkeit des jeweiligen Motors um drei erhöht, damit sich der Roboter wieder zur gewünschten Richtung zurückdreht.

Gleicht die aktuelle Richtung der gewünschten Richtung, so bekommen beide Motoren die gleiche Geschwindigkeit zugewiesen.

Dieses Programm kann jederzeit mit einem Druck auf den Exit-Button abgebrochen werden.

8.5.1 Das closed loop - Verfahren

Das Verfahren, das hier zur Anwendung kommt, wird als **closed loop** bzw. als **Regelkreis mit Rückkopplung** bezeichnet.

Die Änderung von gemessenen Werten führt dazu, dass ein Prozess (hier die Ansteuerung von Motoren) geregelt wird. Bei einem ***closed loop*** Verfahren fließen die Änderungen, die sich aus diesem Regelprozess ergeben, bei den nächsten Entscheidungen oder Berechnungen für die folgenden Aktionen direkt mit ein.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration
6    SET_IN(IN_1,IN_EV3_GYRO);
7    SET_OUT(OUT_A,OUT_MOTOR);
8    SET_OUT(OUT_B,OUT_MOTOR);
9
10   // 2. Initialisieren des Brick
11   EVC_INIT();
12
13   // 3. Programm - Roboter
14   int ZielWert = READ_IN(IN_1);
15   int AktuellerWert = 0;
16
17   while(NOTEXITBUTTON)
18   {
19     AktuellerWert = READ_IN(IN_1);
20
```

```
21      if(AktuellerWert>ZielWert)
22      {
23        WRITE_OUT(OUT_A, MOTOR_POWER, 43);
24        WRITE_OUT(OUT_B, MOTOR_POWER, 40);
25      }
26      if(AktuellerWert==ZielWert)
27      {
28        WRITE_OUT(OUT_A, MOTOR_POWER, 40);
29        WRITE_OUT(OUT_B, MOTOR_POWER, 40);
30      }
31      if(AktuellerWert<ZielWert)
32      {
33        WRITE_OUT(OUT_A, MOTOR_POWER, 40);
34        WRITE_OUT(OUT_B, MOTOR_POWER, 43);
35      }
36    }
37
38    // 4. Programmende - Freigeben der Hardwarezugriffe
39    EVC_CLOSE();
40    return 0;
41  }
```

Listing 18: Exaktes Geradeausfahren mit dem Gyro-Sensor ausgehend von einer Startrichtung

8.5.2 Erklärungen zu diesem Programmbeispiel

Zeile 14: Hier wird die Integer-Variable ***ZielWert*** definiert und dieser der aktuelle Wert des Gyro-Sensors zu Programmstart zugewiesen.

Zeile 15: Definition der Integer-Variable ***AktuellerWert*** in der während der Fahrt die aktuelle Richtung des Gyro-Sensors abgelegt wird.

Zeile 19: Kontinuierliches Einlesen des aktuellen Werts des Gyro-Sensors zu Beginn der while-Schleife.

Zeile 21-25: Hier wird überprüft, ob die aktuelle Richtung größer als die Zielrichtung ist, der Roboter also von der gewünschten Richtung aus nach rechts gedreht hat. Ist dies der Fall, wird die Geschwindigkeit des Motors am Ausgang A (Motor auf der rechten Seite) um 3 erhöht, damit der Roboter wieder in Richtung des gewünschten Werts zurückdreht.

Zeile 26-30: Stimmt die aktuelle Richtung genau mit der gewünschten Richtung überein, so werden die Geschwindigkeiten bei beiden Motoren auf den gleichen Wert gesetzt.

Zeile 31-35: Wenn der aktuelle Richtungswert kleiner als der Zielwert ist, so hat sich der Roboter zu weit nach links verdreht und muss durch Erhöhen der Geschwindigkeit des linken Motors wieder nach rechts zurückdrehen.

8.6 Folgen einer Linie mit einem Sensor und einem Schwellenwert

Wenn ein Roboter dem Verlauf einer Linie folgen soll, so muss dieser über einen nach unten gerichteten Sensor verfügen, der Informationen über den befahrenen Untergrund liefert. Dabei kann es sich um Farbinformationen, um Graustufen oder um reine schwarz/weiß-Informationen handeln.

Beim Folgen einer schwarzen Linie auf einem weißen Untergrund bietet es sich an, den Farb/Licht-Sensor als Licht-Sensor zu verwenden. In diesem Modus liefert dieser Werte zwischen 0% und 100% zurück, was der Intensität des reflektierten Lichts entspricht.

Somit kann nicht nur die weiße Oberfläche von der schwarzen Linie unterschieden werden, sondern man erhält auch Werte dazwischen die angeben, wie weit sich der Sensor über der schwarzen Linie bzw. der weißen Fläche befindet.

Soll der Roboter nun einer schwarzen Linie folgen, so muss dieser mit Hilfe eines Sensors Informationen über den weiteren Linienverlauf erhalten. Dies kann aber nicht dadurch erreicht werden, dass sich der Sensor genau über der schwarzen Linie befindet. Das, was mit dem Sensor dedektiert werden kann, ist aber der Übergang von schwarz zu weiß, da sich die Werte in diesem Bereich ändern. Aus diesen Änderungen kann auf den Verlauf der Linie geschlossen und die Bewegung des Roboters ermittelt werden. Ein Roboter fährt genau genommen also nicht entlang einer schwarzen Linie, sondern entlang der Grenze zwischen der schwarzen Linie und der weißen Fläche.

Aus den aktuellen Sensor-Informationen wird die Fahrtrichtung des Roboters bestimmt: Liefert der Sensor die Information, dass sich dieser über der weißen Fläche befindet, muss sich der Roboter drehen und in Richtung der schwarzen Linie fahren; sobald sich der Sensor über der schwarzen Linie befindet, muss sich der Roboter in die Gegenrichtung drehen und wieder von der Linie herunter zur weißen Fläche hin fahren.

Letztendlich bewegt sich der Sensor (und somit auch der Roboter) auf einer Zick-Zack-Bahn bzw. fährt entlang von Schlangenlinien auf der Grenze zwischen schwarz und weiß.

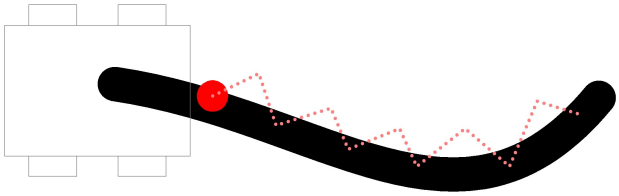


Abbildung 10: Pfad beim Folgen einer Linie mit einem Sensor

Steht nur ein Sensor zur Verfügung, so muss es eine definierte Startrichtung geben, damit der Roboter beim ersten Mal in Richtung der Linie fährt und in der Folge immer auf der gleichen Seite der Linie zwischen schwarz und weiß hin- und herpendelt.

Im folgenden Beispiel werden ausgehend von einem Schwellenwert (die Intensität des reflektieren Lichts, wenn sich der Sensor genau mittig auf der Grenze zwischen schwarz und weiß befindet) die Geschwindigkeiten der Motoren so festgelegt, dass der Roboter entweder eine Links- oder eine Rechtskurve fährt.

Abbildung 11: Ermittlung des Schwellenwerts beim Übergang von schwarz nach weiß

Optimierung bei realen Bedingungen und einem vorgegebenen Streckenverlauf:

- experimentelle Bestimmung der Schwelle basierend auf den tatsächlichen Lichtverhältnissen
- Anpassung der Motorgeschwindigkeiten und der Differenz zwischen beiden an den Kurvenverlauf

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration
6    SET_IN(IN_1,IN_EV3_LIGHT);
7    SET_OUT(OUT_A,OUT_MOTOR);
8    SET_OUT(OUT_B,OUT_MOTOR);
9
10   // 2. Initialisieren des Brick
11   EVC_INIT();
12
13   // 3. Programm - Roboter
14   int Wert = 0;
15
16   while(NOTEXITBUTTON)
17   {
18     Wert = READ_IN(IN_1);
19     LCD_DRAW_INT(2,2,Wert);
20     if(Wert<50)
21     {
22       WRITE_OUT(OUT_A, MOTOR_POWER, 40);
23       WRITE_OUT(OUT_B, MOTOR_POWER, 20);
24     }
25     else
26     {
27       WRITE_OUT(OUT_A, MOTOR_POWER, 20);
28       WRITE_OUT(OUT_B, MOTOR_POWER, 40);
29     }
30   }
31
32   // 4. Programmende - Freigeben der Hardwarezugriffe
33   EVC_CLOSE();
34
35   return 0;
36 }
```

Listing 19: Folgen einer Linie mit einem Sensor und einem Schwellenwert

8.6.1 Erklärungen zu diesem Programmbeispiel

Zeile 18: Einlesen des aktuellen Werts des Licht-Sensors und Ablegen in der Variable ***Wert***.

Zeile 20-29: Befindet sich der Sensor zu einem größeren Teil über der schwarzen Linie (Intensität < 50) wird die Geschwindigkeit der beiden Motoren so gesetzt, dass dieser in der entgegengesetzten Richtung auf die weiße Fläche fährt, aus der er auf die Linie gefahren ist.

Ist die gemessene Intensität größer oder gleich 50, dann befindet sich der Sensor mehr über der weißen Fläche und die beiden Motoren werden so angesteuert, dass der Roboter in Richtung der schwarzen Linie fährt.

8.6.2 Aufgabe S1Linie1

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen.

Die Linie soll grundsätzlich einen geraden Verlauf haben. Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

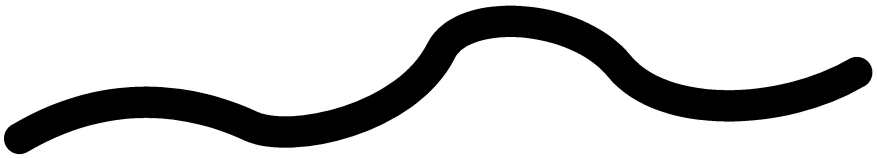
Nun sollen für die herrschenden Lichtverhältnisse der passende Schwellenwert und die Werte für die Geschwindigkeiten so bestimmt werden, dass der Roboter der gesamten Linie bis zum Ende folgen kann, ohne diese zu verlassen.

8.6.3 Aufgabe S1Linie2

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen.

Die Linie soll grundsätzlich einen geraden Verlauf haben, aber Kurven unterschiedlicher Krümmung enthalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Nun sollen für die herrschenden Lichtverhältnisse der passende Schwellenwert und die Werte für die Geschwindigkeiten so bestimmt werden, dass der Roboter dem gesamten Linienverlauf bis zum Ende folgen kann, ohne diesen zu verlassen.

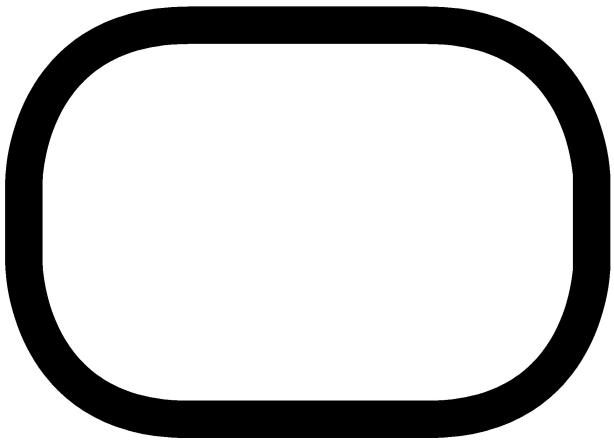


8.6.4 Aufgabe S1Linie3

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll einem Oval gleichen und die Kurven sollen annähernd die gleiche Krümmung aufweisen. Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Ziel ist es, optimale Werte für die Schwelle des Lichtwerts, wie auch für die Grundgeschwindigkeit und die Differenz zwischen beiden Geschwindigkeiten zu ermitteln, sodass der Roboter die vorgegebene Strecke so schnell wie möglich abfährt.

Begleitend dazu soll aber auch berücksichtigt werden, dass der Roboter mit diesen Werten dem Kurvenverlauf zehn Mal hintereinander verlässlich folgen kann, ohne dabei die Linie zu verlassen.



8.6.5 Aufgabe S1Linie4

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll zwei rechte Winkel beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für das Abbiegen bei rechten Winkeln sollen Werte für die Schwelle des Lichtwerts, wie auch für die Geschwindigkeiten ermittelt werden, damit der Roboter sicher und verlässlich die beiden rechten Winkel hintereinander passiert.

Die Aufgabe soll auf zwei Arten gelöst werden, sodass es eine Lösung für das Fahren auf der Innenseite und eine Lösung für das Fahren auf der Außenseite der Linie bzw. der Winkel gibt.

8.7 Folgen einer Linie mit einem Sensor und mehreren Bereichen

Verwendet man nur einen Schwellenwert, so kann ein Roboter einer Linie nur dann verlässlich folgen, wenn alle Kurven (fast) die gleiche Krümmung aufweisen, an welche die Motorgeschwindigkeiten und die Differenz zwischen diesen beiden optimal angepasst wurde.

Bei gleicher Geschwindigkeit und Fahrtrichtung ist der Sensor bei einer Kurve mit größerer Krümmung in der gleichen Zeit weiter von der Linie entfernt als bei einer Kurve mit geringerer Krümmung. Aus diesem Grund muss die Drehung hin in Richtung zur schwarzen Linie bei einer Kurve mit größer Krümmung stärker sein, als bei einer Kurve mit geringer Krümmung.

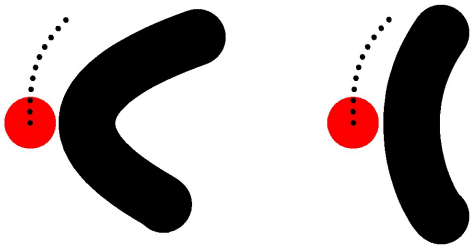


Abbildung 12: Verlieren des Linienverlaufs bei zu starker Krümmung

Damit ein Roboter besser und verlässlicher auf unterschiedliche Krümmungen in einem Kurvenverlauf einer Linie reagieren kann, ist es von Vorteil sich die unterschiedlichen Intensitäten und damit die Informationen über die Entfernung des Sensors von der Grenze zwischen schwarz und weiß zu Nutze zu machen.

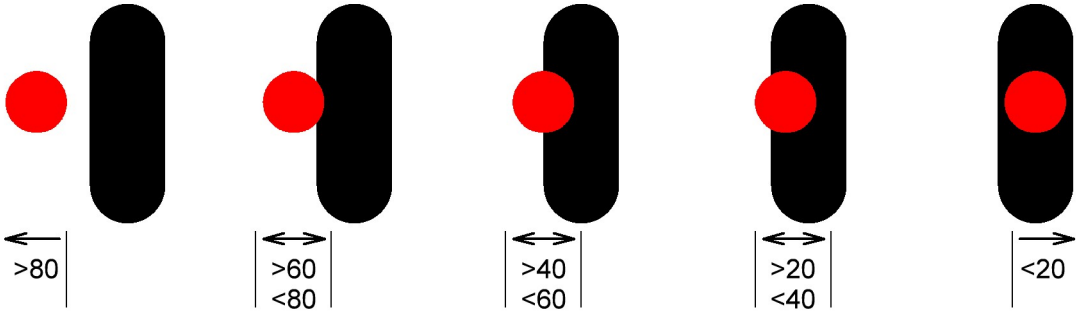


Abbildung 13: Unterschiedliche Intensitäten in Bezug auf die Position des Sensors

Die unterschiedlichen Intensitäten können herangezogen werden um den Roboter je nach Bereich in dem sich die Sensor-Werte befinden entweder steiler oder flacher zur Linie oder zurück zur weißen Fläche fahren zu lassen.

Im folgenden Beispiel werden die möglichen Werte des Sensors in fünf unterschiedliche Bereiche aufgeteilt und davon ausgehend unterschiedlich große Unterschiede für die Geschwindigkeiten der beiden Motoren gesetzt.

Optimierung bei realen Bedingungen und einem vorgegebenen Streckenverlauf:

- experimentelle Bestimmung der Bereiche basierend auf den tatsächlichen Lichtverhältnissen
- Anpassung der Motorgeschwindigkeiten und der Differenzen zwischen beiden an den Kurvenverlauf

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration
  SET_IN(IN_1,IN_EV3_LIGHT);
  SET_OUT(OUT_A,OUT_MOTOR);
  SET_OUT(OUT_B,OUT_MOTOR);

  // 2. Initialisieren des Brick
  EVC_INIT();

  // 3. Programm - Roboter
  int Wert = 0;

  while(NOTEXITBUTTON)
  {
    Wert = READ_IN(IN_1);
    LCD_DRAW_INT(2,2,Wert);

    if(Wert<20)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 20);
    }
    if ((Wert>=20)&&(Wert<=40))
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 30);
    }
    if ((Wert>40)&&(Wert<60))
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 40);
    }
    if ((Wert>=60)&&(Wert<=80))
        {
          WRITE_OUT(OUT_A, MOTOR_POWER, 30);
          WRITE_OUT(OUT_B, MOTOR_POWER, 40);
        }
    if(Wert>80)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 20);
      WRITE_OUT(OUT_B, MOTOR_POWER, 40);
    }
  }

  // 4. Programmende - Freigeben der Hardwarezugriffe
  EVC_CLOSE();
  return 0;
}
```

Listing 20: Optimiertes Folgen einer Linie mit mehreren Bereichen

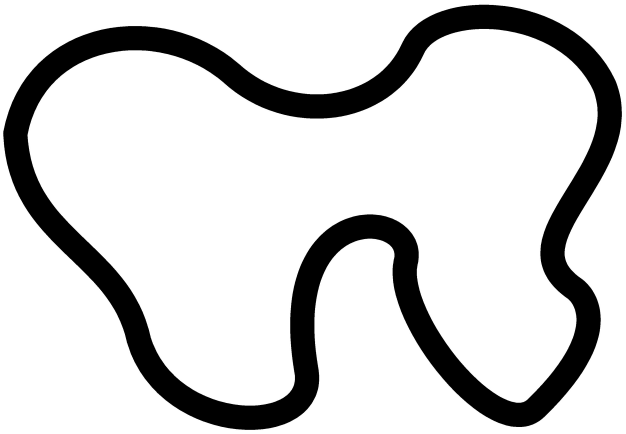
8.7.1 Aufgabe S1Linie5

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Der Linienverlauf soll geschlossen sein und mehrere Kurven mit sehr unterschiedlichen Krümmungen beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für den vorliegenden Kurvenverlauf sollen nun fünf Wertebereiche und dazu optimierte Geschwindigkeiten gefunden werden, damit der Roboter dem Linienverlauf sicher und verlässlich folgen kann.

Dabei soll auch berücksichtigt werden, dass es mit den ermittelten Werten möglich ist den Roboter mehrere Male hintereinander der Linie folgen zu lassen.

Dazu soll ein Programm für das Fahren auf der Innenseite und ein Programm für das Fahren auf der Außenseite erstellt werden.



8.7.2 Aufgabe S1Linie6

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll zwei rechte Winkel beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für das Abbiegen bei den rechten Winkeln sollen Werte für fünf Intervalle der Lichtwerte gefunden werden, damit der Roboter sicher und verlässlich bei beiden rechten Winkeln hintereinander abbiegt.

Die Aufgabe soll auf zwei Arten gelöst werden, sodass es eine Lösung für das Fahren auf der Innenseite und eine Lösung für das Fahren auf der Außenseite der Linie bzw. der Winkel gibt.

8.8 Ablaufsteuerung mit Ereignisbehandlung

Soll ein Roboter oder ein autonomes System unterschiedliche Aufgaben erledigen, so können entsprechende Codesegmente zusammengefasst und diesen ein Status zugewiesen werden. Dieser Status wird in einer Integer-Variable ***Status*** festgelegt, deren Wert für die Steuerung des Programmablaufs ausschlaggebend ist.

Die Informationen der Sensoren werden dazu herangezogen um darüber zu entscheiden, welches Codesegment ausgeführt werden soll. Zur besseren Übersichtlichkeit bietet es sich an, den Programmcode innerhalb der while-Schleife in zwei Bereiche zu unterteilen: Ereignisbehandlung und Aktionen

Im Bereich **Ereignisbehandlung** werden die Sensordaten ausgewertet und aufgrund der jeweiligen Situation der Variable ***Status*** ein Wert zugewiesen.

Erreicht die Ausführung des Programmcodes den Bereich **Aktionen**, so wird nur das Codesegment ausgeführt, das dem aktuellen Wert der Variable ***Status*** entspricht.

Ergänzend bietet es sich an, den jeweiligen Status in dem sich der Roboter gerade befindet am Display anzuzeigen, um während der Ausführung des Programms gut nachvollziehen zu können, welches Codesegment gerade ausgeführt wird.

Bei dieser Strukturierung des Programmcodes ist gewährleistet, dass bereits beim nächsten Durchlauf der while-Schleife auf eine Veränderung der Umgebungssituation bzw. der Sensorinformationen reagiert werden kann. Voraussetzung ist natürlich, dass sich innerhalb der while-Schleife keine Codeteile (z. B. die Funktion ***SLEEP()***) befinden, welche die Ausführung des Programmcodes anhalten!

Mit dem folgenden Programm kann ein Roboter unterschiedliche Bewegungsabläufe (gerade aus, Kurve nach rechts, ...) fahren. Zunächst wird im Bereich **Ereignisbehandlung** abgefragt, ob einer der Buttons am EV3 Brick gedrückt wurde. Davon ausgehend wird der Variable ***Status*** ein Wert zwischen *1* und *5* zugewiesen.

Im Bereich **Aktionen** wird dann die Variable ***Status*** auf ihren Wert hin überprüft und der, dem jeweiligen Status zugeordnete Programmcode ausgeführt.

Da die Variable ***Status*** zu Beginn den Wert *0* zugewiesen bekommt und es für diesen Status kein Codesegment im Bereich **Aktionen** gibt, bleibt der Roboter zunächst stehen, wenn das Programm gestartet wird und beginnt erst zu fahren, wenn einer der Buttons UP, LEFT, RIGHT oder DOWN gedrückt wurde.

Durch den Druck auf einen anderen Button kann die Ausführung des gerade aktiven Programmsegments jederzeit unterbrochen werden und somit auf einen anderen Bewegungsablauf gewechselt werden.

Beendet kann dieses Programm jederzeit mit dem Exit-Button werden.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration
  SET_OUT(OUT_A,OUT_MOTOR);
  SET_OUT(OUT_B,OUT_MOTOR);

  // 2. Initialisieren des Brick
  EVC_INIT();

  // 3. Programm - Roboter
  int Status = 0;
  while(NOTEXITBUTTON)
  {
    // Ereignisbehandlung
    if (READ_BUTTON(BUTTON_UP)==1)  Status = 1;
    if (READ_BUTTON(BUTTON_LEFT)==1) Status = 2;
    if (READ_BUTTON(BUTTON_RIGHT)==1) Status = 3;
    if (READ_BUTTON(BUTTON_DOWN)==1) Status = 4;
    if (READ_BUTTON(BUTTON_CENTER)==1) Status = 5;
    // Aktionen
    if(Status==1)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 40);
    }
    if(Status==2)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 40);
      WRITE_OUT(OUT_B, MOTOR_POWER, 10);
    }
    if(Status==3)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, 10);
      WRITE_OUT(OUT_B, MOTOR_POWER, 40);
    }
    if(Status==4)
    {
      WRITE_OUT(OUT_A, MOTOR_POWER, -40);
      WRITE_OUT(OUT_B, MOTOR_POWER, -40);
    }
    if(Status==5)
    {
      WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);
      WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_BRAKE);
    }
  }

  // 4. Programmende - Freigeben der Hardwarezugriffe
  EVC_CLOSE();
  return 0;
}
```

Listing 21: Ablaufsteuerung mit Buttons

8.8.1 Erklärungen zu diesem Programmbeispiel

Zeile 13: Definition der Variable *Status* welcher zu Beginn der Wert *0* zugewiesen wird.

Zeile 17-21: Im Bereich **Ereignisbehandlung** wird mit den entsprechenden Funktionen ermittelt, ob einer der Buttons am EV3 Brick gedrückt wurde. Wenn ja, dann wird der Variable *Status* einer der Werte zwischen *1* und *5* zugewiesen.

Zeile 23-47: In den if-Abfragen im Bereich **Aktionen** wird überprüft, welchen Wert die Variable *Status* hat. Innerhalb der if-Abfragen befindet sich der Programmcode, der bei einem bestimmten Status ausgeführt werden soll.

Ein weiteres Beispiel wäre das Abfahren eines Pfades (dieses wird dem Status 1 zugeordnet), das aber jederzeit und abrupt abgebrochen werden muss, wenn ein Roboter dabei mit einem Hindernis kollidiert. Dabei würde die Information eines Kollisions-Sensors zur Festlegung eines neuen Status führen, mit dem das Abstoppen aller Motoren verbunden ist und damit der Bewegungsablauf, der in Status 1 festgelegt ist, unterbrochen werden.

Im folgenden Code-Beispiel ist dieses Grundkonzept dargestellt:

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration
  SET_OUT(OUT_A,OUT_MOTOR);
  SET_OUT(OUT_B,OUT_MOTOR);
  SET_IN(IN_1,IN_EV3_TOUCH);

  // 2. Initialisieren des Brick
  EVC_INIT();

  // 3. Programm - Roboter

  int Status = 0;

  while(NOTEXITBUTTON)
  {
    // Ereignisbehandlung

    if(READ_BUTTON(BUTTON_UP)==1)
    {
      Status = 1;
    }

    if(READ_IN(IN_1)==1)
    {
      Status = 2;
    }

    // Aktionen
    if(Status==1)
    {
      // Code fuer den Bewegungsablauf
    }

    if(Status==2)
    {
      // Code fuer die Kollision
    }
  }

  // 4. Programmende - Freigeben der Hardwarezugriffe
  EVC_CLOSE();
  return 0;
}
```

Listing 22: Bewegungsablauf mit Kollisionsabfrage

8.9 Zeitgesteuerter Bewegungsablauf mit Kollisionsabfrage

In diesem Beispiel wird gezeigt, wie zeitgesteuerte Bewegungsabläufe in eine Ablaufsteuerung mit Ereignisbehandlung integriert werden können. Der Roboter, der hier zum Einsatz kommen soll verfügt über zwei Motoren für den Antrieb und einen Touch-Sensor an der Vorderseite um auf das Berühren eines Hindernisses reagieren zu können.

Wichtigstes Ziel bei der Umsetzung ist, dass der Programmcode innerhalb der while-Schleife so schnell wie möglich durchlaufen wird und dieser an keiner Stelle durch die Verwendung der Funktion ***SLEEP()*** oder eine weitere Schleife, die auf das Eintreten eines Ereignisses wartet, angehalten wird.

So ist gewährleistet, dass die if-Abfragen im Bereich **Ereignisbehandlung** ununterbrochen und in möglichst kurzen Zeitabständen hintereinander ausgeführt werden und somit umgehend auf Sensor-Informationen reagiert werden kann.

Mit diesem Programm soll der Roboter durch einen Druck auf den UP-Button einen vorgegebenen, zeitgesteuerten Pfad abfahren und bei einer Kollision jederzeit anhalten. Dazu wird diese Aufgabenstellung in drei Status-Bereiche unterteilt:

Status 0 - Programmstart, keine Aktionen

Wird das Programm gestartet, so ist der Roboter im Status 0 und führt keine Aktionen aus.

Status 1 - zeitgesteuerter Pfad

Durch einen Druck auf den UP-Button am EV3 Brick wird der Status auf 1 und der Counter auf 0 gesetzt. In der Folge wird der Programmcode ausgeführt, der Status 1 zugeordnet ist und den Roboter folgenden Pfad abfahren lässt: 2 Sekunden vorwärts, 1 Sekunde Drehung, 1 Sekunde vorwärts

Status 2 - Kollision

Trifft der Touch-Sensor auf ein Hindernis so liefert dieser den Wert 1 zurück und die Variable Status wird auf 2 gesetzt. In dem dazugehörenden Programmcode werden beide Motoren angehalten und der Roboter bleibt vor dem Hindernis stehen.

Das Abfahren des vorgegebenen Pfades kann jederzeit durch Drücken des UP-Buttons wieder gestartet werden. Auch ist es jederzeit möglich, das Programm mit dem Exit-Button zu beenden.

```
1  #include "evclibrary.h"
2
3  int main()
4  {
5    // 1. Konfiguration
6    SET_OUT(OUT_A,OUT_MOTOR);
7    SET_OUT(OUT_B,OUT_MOTOR);
8    SET_IN(IN_1,IN_EV3_TOUCH);
9
10   // 2. Initialisieren des Brick
11   EVC_INIT();
12
13   // 3. Programm - Roboter
14
```

```
15    int Status = 0;
16
17    while(NOTEXITBUTTON)
18    {
19      // Ereignisbehandlung
20
21      if(READ_BUTTON(BUTTON_UP)==1)
22      {
23        COUNTER = 0;
24        Status = 1;
25      }
26
27      if(READ_IN(IN_1)==1)
28      {
29        Status = 2;
30      }
31
32      // Aktionen
33      if(Status==1)
34      {
35        if (COUNTER>=0)
36        {
37          WRITE_OUT(OUT_A, MOTOR_POWER, 50);
38          WRITE_OUT(OUT_B, MOTOR_POWER, 50);
39        }
40        if (COUNTER>=20)
41        {
42          WRITE_OUT(OUT_A, MOTOR_POWER, 0);
43          WRITE_OUT(OUT_B, MOTOR_POWER, 50);
44        }
45        if (COUNTER>=30)
46        {
47          WRITE_OUT(OUT_A, MOTOR_POWER, 50);
48          WRITE_OUT(OUT_B, MOTOR_POWER, 50);
49        }
50        if (COUNTER>=40)
51        {
52          WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_COAST);
53          WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_COAST);
54        }
55      }
56
57      if(Status==2)
58      {
59        WRITE_OUT(OUT_A, MOTOR_STOP, MOTOR_BRAKE);
60        WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_BRAKE);
61      }
62    }
63
64    // 4. Programmende - Freigeben der Hardwarezugriffe
65    EVC_CLOSE();
66
67    return 0;
68  }
```

Listing 23: Zeitgesteuerter Bewegungsablauf mit Kollisionsabfrage

9 Referenz der EVCdevelop Bibliothek

9.1 Programmstruktur

9.1.1 Programm mit einem Programmdurchlauf

Der Programmcode nach dem Kommentar *// 3. Steuerung des Roboters* wird einmal ausgeführt und das Programm im Anschluss beendet.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  // 3. Steuerung des Roboters

  // 4. Programmende
  EVC_CLOSE();
  return 0;
}
```

9.1.2 Programmstruktur mit wiederholtem Programmdurchlauf und Exit-Button

Die Anweisungen in der while-Schleife (grundsätzlich eine Endlos-Schleife) werden so lange ausgeführt, bis das Programm mit einem Druck auf den Exit-Button am EV3 Brick beendet wird.

```
#include "evclibrary.h"

int main()
{
  // 1. Konfiguration der angeschlossenen Sensoren und Motoren

  // 2. Initialisierung des EV3 Brick
  EVC_INIT();

  // 3. Steuerung des Roboters
  while (NOTEXITBUTTON)
  {
    ...
  }

  // 4. Programmende
  EVC_CLOSE();
  return 0;
}
```

9.1.3 SLEEP(*time*)

Mit dieser Funktion kann Ausführung des Programms für eine bestimmte Zeit angehalten werden. Die Zeit wird dabei in Millisekunden angegeben.

time: ganzzahliger Wert / Millisekunden

Beispiel

```
1 // die weitere Ausfuehrung des Programms wird fuer 2 Sekunden angehalten
2 SLEEP(2000);
```

9.2 Initialisieren und Freigeben der Hardware-Resourcen

9.2.1 EVC_INIT()

Diese Funktion muss aufgerufen werden, bevor die Eingänge des EV3 Brick ausgelesen oder die Ausgänge angesteuert werden können. In dieser Funktion erfolgt die Initialisierung der Sensoreingänge, der Motorausgänge, des LCD-Displays, wie auch der LED und der Tasters am Bedienpanel des EV3 Brick.

9.2.2 EVC_CLOSE()

Damit ein Programm ordnungsgemäß beendet wird, muss diese Funktion am Ende des Programmcodes aufgerufen werden um den Zugriff auf alle Hardware-Ressourcen des EV3 Bricks freizugeben.

9.3 Bedienpanel – Taster und LED

9.3.1 WRITE_LED(*color*)

Mit dieser Funktion kann die LED am Bedienpanel des EV3 Bricks ein- und ausgeschaltet bzw. deren Farbe festgelegt werden.

color: LED_BLACK, LED_GREEN, LED_RED, LED_ORANGE, LED_GREEN_FLASH, LED_RED_FLASH, LED_ORANGE_FLASH, LED_GREEN_PULSE, LED_RED_PULSE, LED_ORANGE_PULSE

Beispiel

```
1 // die Farbe der LED wird auf rot gesetzt
2 WRITE_LED(LED_RED);
```

9.3.2 READ_BUTTON(*button*)

Mit der Funktion READ_BUTTON() kann der Druck auf eine Taste am Bedienpanel des EV3 Bricks abgefragt werden. Mit dem Parameter ***button*** wird der gewünschte Taster festgelegt, der Rückgabewert der Funktion ist 0 oder 1.

button: BUTTON_UP, BUTTON_DOWN, BUTTON_LEFT, BUTTON_RIGHT, BUTTON_CENTER

return: 0, 1

Beispiel

```
1 // mit der linken und rechten Taste wird die LED-Farbe gesetzt
2 if (READ_BUTTON(BUTTON_LEFT)==1) WRITE_LED(LED_RED);
3 if (READ_BUTTON(BUTTON_RIGHT)==1) WRITE_LED(LED_GREEN);
```

9.4 Display

9.4.1 LCD_CLEAR()

Die Funktion LCD_CLEAR() dient zum Löschen des Displays.

9.4.2 LCD_DRAW_FILLRECTANGLE(*x1, y1, x2, y2, color*)

Mit dieser Funktion kann am Display ein ausgefülltes Rechteck gezeichnet werden. Das Koordinatenpaar (x1, y1) gibt den linken oberen Eckpunkt, (x2, y2) den rechten unteren Eckpunkt des Rechtecks an. Mit dem Parameter color wird die Farbe (schwarz oder weiß) angegeben.

x1: 0, ..., 177

y1: 0, ..., 127

x2: 0, ..., 177

y2: 0, ..., 127

color: 0, 1

Beispiel

```
1 // am Display wird ein ausgefuelltes Rechteck in maximaler Groesse gezeichnet
2 LCD_DRAW_FILLRECTANGE(0,0,177,127,1);
```

9.4.3 LCD_DRAW_RECTANGLE(*x1, y1, x2, y2, color*)

Diese Funktion zeichnet ein Rechteck am Display. Das Koordinatenpaar (x1, y1) gibt den linken oberen Eckpunkt, (x2, y2) den rechten unteren Eckpunkt des Rechtecks an. Mit dem Parameter color wird die Farbe (schwarz oder weiß) angegeben.

x1: 0, ..., 177

y1: 0, ..., 127

x2: 0, ..., 177

y2: 0, ..., 127

color: 0, 1

Beispiel

```
1 // am Display wird ein Rechteck gezeichnet
2 LCD_DRAW_FILLRECTANGE(10,10,157,107,1);
```

9.4.4 LCD_DRAW_PIXEL(*x, y, color*)

Mit der Funktion LCD_DRAW_PIXEL() kann ein Pixel am Display ein- und ausgeschaltet werden.

x: 0, ..., 177

y: 0, ..., 127

color: 0, 1

Beispiel

```
1 // das Pixel an der Position (20,10) wird schwarz angezeigt
2 LCD_DRAW_DRAWPIXEL(20,10,1);
```

9.4.5 LCD_DRAW_TEXT(*spalte, zeile, text*)

Zur Ausgabe von Text am Display wird der Funktion LCD_DRAW_TEXT() die gewünschte Position und der auszugebende Text übergeben. In jeder der 11 Zeilen können 22 Zeichen ausgegeben werden.

spalte: 1, ..., 22

zeile: 1, ..., 11

text: "Text Text Text"

Beispiel

```
1 // Ausgabe des Textes Hello EVCdevelop!
2 LCD_DRAW_TEXT(1,1,"Hello EVCdevelop!");
```

9.4.6 LCD_DRAW_INT(*x, y, value*)

Mit dieser Funktion können Zahlenwerte am Display angezeigt werden. Bei diesen Zahlenwerten muss es sich um Integer-Variablen handeln.

x: 0, ..., 177

y: 0, ..., 127

value: Name einer Variable vom Typ int

Beispiel

```
1 // Ausgabe des Inhalts der Variable zahl
2 int zahl = 25;
3 LCD_DRAW_INT(1,1, zahl);
```

9.5 Eingänge und Sensoren

9.5.1 SET_IN(*port, value*)

Mit dieser Funktion wird festgelegt, an welcher Sensor an einem Eingang angeschlossen ist. Diese Funktion muss vor der Funktion EVC_INIT() aufgerufen werden!

Der Parameter port beschreibt die Nummer des Eingangs (IN_1, IN_2, IN_3 oder IN_4). Mit dem Parameter value wird der Typ des Sensors festgelegt.

port: IN_1, IN_2, IN_3, IN_4

value: IN_EV3_TOUCH, IN_EV3_LIGHT, IN_EV3_COLOR, IN_EV3_SONAR, IN_EV3_GYRO

Beispiel

```
1 // an IN-Port 3 ist ein Ultraschallsensor angeschlossen
2 SET_IN(IN_3, IN_EV3_SONAR);
```

9.5.2 READ_IN(*port*)

Mit dieser Funktion kann der aktuelle Wert eines Sensors ausgelesen werden, der mit der Funktion ***SET_IN()*** festgelegt wurde. Der Parameter port beschreibt den gewünschten Eingangsport des EV3 Bricks. Der Rückgabewert steht für den von dem jeweiligen Sensor ermittelten Wert.

port: IN_1, IN_2, IN_3, IN_4

value (IN_EV3_TOUCH): 0, 1

value (IN_EV3_LIGHT): 0, ..., 100

value (IN_EV3_COLOR): 0, ..., 7

value (IN_EV3_SONAR): 3, ..., 150

value (IN_EV3_GYRO): ganze Zahl, Drehwinkel in Grad

IN_EV3_COLOR

Die Rückgabewerte zwischen 0 und 7 stehen für verschiedene Farbwerte:

- 0 transparent (bzw. keine Farbe erkannt)
- 1 schwarz
- 2 blau
- 3 grün
- 4 gelb
- 5 rot
- 6 weiß
- 7 braun

IN_EV3_LIGHT

Die Rückgabewerte zwischen 0 und 100 beschreiben die Intensität des reflektierten Lichts zwischen 0% und 100%.

IN_EV3_GYRO

Ist ein Gyro-Sensor an einen Eingang angeschlossen, so beschreiben die Rückgabewerte der Funktion ***READ_IN()*** den Drehwinkel in Grad seit dem Einschalten des Bricks. Wird der Sensor gegen den Uhrzeigersinn gedreht, so verringert sich dieser Wert, dreht man den Sensor nach rechts, so wird der Wert um den jeweiligen Drehwinkel erhöht.

Beispiel

```
1  // basierend auf dem Wert des Sensors wird die LED-Farbe gesetzt
2  int value = READ_IN(IN_1);
3  if (value>50)
4  {
5    WRITE_LED(LED_RED);
6  }
7  else
8  {
9    WRITE_LED(LED_GREEN);
10 }
```

9.6 Ausgänge und Motoren

9.6.1 SET_OUT (*port, value*)

Mit dieser Funktion wird festgelegt, an welchem Eingang ein Motor angeschlossen ist. Diese Funktion muss vor der Funktion EVC_INIT() aufgerufen werden!

Der Parameter port beschreibt die Nummer des Ausgangs (OUT_A, OUT_B, OUT_C oder OUT_D).

port: OUT_A, OUT_B, OUT_C, OUT_D

value: OUT_MOTOR

Beispiel

```
1 // an OUT-Port B ist ein Motor angeschlossen
2 SET_OUT(OUT_B, OUT_MOTOR);
```

9.6.2 WRITE_OUT(*port, mode, value*)

Mit der Funktion WRITE_OUT() wird ein Motor angesteuert. Mit dem Parameter wird der Anschluss ausgewählt. Im Parameter mode wird der gewünschte Betriebsmodus des Motors festgelegt. Der Wert des Parameters value bezieht sich auf den gewählten Betriebsmodus.

port: OUT_A, OUT_B, OUT_C, OUT_D

mode: MOTOR_POWER, MOTOR_SPEED, MOTOR_STOP, MOTOR_POS, MOTOR_RESET

value (MOTOR_POWER): -100 ... + 100

value (MOTOR_SPEED): -100 ... + 100

value (MOTOR_STOP): MOTOR_COAST, MOTOR_BRAKE

value (MOTOR_POS): -XXX ... + XXX

value (MOTOR_RESET): 0

MOTOR_POWER

Der Motor wird mit einer Leistung zwischen 0% und 100% betrieben. Erfährt der Motor einen Widerstand (z. B. Bergauf-Fahren), so verringert sich die Drehgeschwindigkeit des Motors und wird nicht nachgeregelt (open loop). Das Vorzeichen des im Parameter value festgelegten Werts gibt die Drehrichtung an.

Beispiel

```
1 // der Motor an Port A wird mit einer Leistung von 50% betrieben
2 WRITE_OUT(OUT_A, MOTOR_POWER, 50);
```

MOTOR_SPEED

Wird ein Motor in diesem Modus betrieben, so stellt man mit dem Parameter value die gewünschte Geschwindigkeit ein. In diesem Modus steuert ein Regelkreis die Leistung des Motors und versucht je nach Widerstand die Geschwindigkeit beizubehalten (closed loop). Das Vorzeichen des im Parameter value festgelegten Werts gibt die Drehrichtung an.

Beispiel

```
// der Motor an Port B soll sich mit einer Geschwindigkeit 70% drehen
 WRITE_OUT(OUT_B, MOTOR_SPEED, 70);
```

MOTOR_STOP

Ein Motor lässt sich auf zwei unterschiedliche Arten abbremsen. Im ersten Fall wird die Stromzufuhr zum Motor unterbrochen und der Motor läuft aufgrund der Trägheit nach bis die Drehung zum Stillstand kommt (MOTOR_COAST). Soll ein Motor unverzüglich angehalten werden, so übergibt man den Wert MOTOR_BRAKE. Dabei wird der Motor für eine sehr kurze Zeitspanne in die entgegengesetzte Drehrichtung versetzt, wodurch ein sofortiges Abbremsen erreicht wird.

Beispiel

```
// der Motor an Port B wird gestoppt und laeuft nach
WRITE_OUT(OUT_B, MOTOR_STOP, MOTOR_COAST);

// der Motor an Port C wird unverzueglich abgebremst
WRITE_OUT(OUT_C, MOTOR_STOP, MOTOR_BRAKE);
```

MOTOR_POS

Mit dem Betriebsmodus MOTOR_POS kann ein Motor um einen vorgegebenen Winkel gedreht werden. Im Parameter value wird der Drehwinkel und mit dem Vorzeichen die Drehrichtung festgelegt.

Beispiel

```
// der Motor an Port D soll sich um 90 Grad drehen
WRITE_OUT(OUT_D, MOTOR_POS, 90);
```

MOTOR_RESET

Wir ein Motor im Betriebsmodus MOTOR_POS betrieben, so wird der gewünschte Drehwinkel von der letzten Position gemessen. Mit MOTOR_RESET kann diese Position zurückgesetzt werden.

Beispiel

```
// der Motor an Port A soll sich um 90 Grad drehen
WRITE_OUT(OUT_D, MOTOR_RESET, 0);
```

10 Referenz Programmiersprache C

In der Folge sind grundlegende Sprachelemente der Programmiersprache C zusammengestellt, wie sie in den Programmierbeispielen zu EVCdevelop verwendet werden.

10.1 Programmstruktur

Die Grundstruktur eines C-Programms besteht aus der Funktion main().

```
1 int main()
2 {
3   ...
4   return 0;
5 }
```

10.2 Variablentypen

Zum Speichern von Werten kommen Variablen zum Einsatz. Für ganzzahlige Werte wird meist der Variablentyp int (Integer-Zahlen) verwendet. Zum Ablegen von Dezimalzahlen werden Variablen vom Typ float (Gleitkomma-Zahlen) verwendet.

Die Definition einer Variable erfolgt nach dem Schema: Typ Name

Für den Namen einer Variable dürfen keine Sonderzeichen und keine Leerzeichen verwendet werden. Das einzig erlaubte Zeichen um Variablennamen besser lesbar zu machen ist der Unterstrich.

```
1 int MeineGanzeZahl;
```

Wird eine Variable innerhalb einer Funktion definiert, kann dieser auch schon bei der Definition ein Wert zugewiesen werden.

```
1 float a\_ist\_eine\_Dezimalzahl = 25.7;}}
```

10.3 Rechenoperationen

Die folgenden Operatoren werden in C zum Rechnen mit Variablen und Zahlen verwendet.

=	ist gleich
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Rest einer Division)

Das einfache = wird als zuweisendes „ist gleich“ bezeichnet.

`a = 5;` Der Variable auf der linken Seite wird der Wert auf der rechten Seite zugewiesen.

Beispiel

```
1 int a = 3;
2 int b = 5;
3 int c;
4 c = a + b;
```

Es werden drei Variablen vom Typ int definiert. Den beiden Variablen a und b werden bei der Definition die Werte 3 und 5 zugewiesen.

In der letzten Zeile wird der Variablen c das Ergebnis der Addition von a und b zugewiesen.

10.4 Entscheidungsabfragen

Mit einer Entscheidungsabfrage kann der weitere Ablauf eines Programms gesteuert werden. Zunächst wird eine Bedingung angegeben, die überprüft wird.

Es folgt ein Block mit Anweisungen, die ausgeführt werden sollen, wenn die Bedingung erfüllt ist.

```
1  if (Bedingung)
2  {
3    Anweisung 1;
4    Anweisung 2;
5    ...
6  }
```

Auch ist es möglich, einen weiteren Block von Anweisungen festzulegen, die ausgefüllt werden sollen, wenn die Bedingung nicht erfüllt ist.

```
1  if (Bedingung)
2  {
3    Anweisung 1;
4    Anweisung 2;
5    ...
6  }
7  else
8  {
9    Anweisung 1;
10   Anweisung 2;
11   ...
12 }
```

10.4.1 Vergleichsoperatoren für Bedingungen

In einer Bedingung werden zwei oder mehrere Werte miteinander verglichen. Meist soll überprüft werden, ob eine Variable einen bestimmten Wert hat. Für diese Überprüfung können in den Bedingungen einer if-Abfrage und einer while-Schleife folgende Operatoren verwendet werden:

`if (a==1) {...}`	==	ist gleich	Hat die Variable a den Wert 1?
`if (a!=1) {...}`	!=	ungleich	Ist der Wert der Variable a ungleich 1?
`if (a<4) {...}`	<	kleiner	Ist der Wert der Variable a kleiner als 4?
`if (a>20) {...}`	>	größer	Ist der Wert der Variable a größer als 20?
`if (a<=10) {...}`	<=	kleiner gleich	Ist der Wert der Variable a kleiner gleich 10?
`if (a>=25) {...}`	>=	größer gleich	Ist der Wert der Variable a größer gleich 25?

10.4.2 Verknüpfungsoperatoren für Bedingungen

Sollen in einer Bedingung mehrere Bedingungen verknüpft werden, so gibt es dafür die beiden Operatoren && und || für das logische UND und das logische ODER.

Bei einer Verknüpfung mit && - dem logischen UND - ist die Gesamtbedingung erfüllt, wenn beide Bedingungen erfüllt sind.

Bei einer Verknüpfung mit || - dem logischen ODER - gilt die Gesamtbedingung dann als erfüllt, wenn entweder die eine, die andere oder beide Bedingungen erfüllt sind.

`if ((a==1)&&(b!=2)) {...}`	Hat die Variable a den Wert 1 UND b einen Wert ungleich 2?
`if ((a==1)||(b==1)) {...}`	Hat die Variable a den Wert 1 ODER b den Wert 1?

10.5 Schleifen

Schleifen werden in Programmen verwendet, wenn ein Block von Anweisungen mehrmals hintereinander ausgeführt werden soll.

10.5.1 Die while-Schleife

Die while-Schleife wird verwendet, wenn Anweisungen so lange wiederholt ausgeführt werden sollen, solange eine Bedingung erfüllt ist.

```
1 while (Bedingung)
2 {
3   ...
4 }
```

Für die Bedingung einer while-Schleife gelten die gleichen Vorgaben wie für die if-Abfrage, es sind die selben Vergleichs- und Verknüpfungsoperatoren zu verwenden.

Beispiel

`while (a<30) { ... }`

In der Bedingung wird festgelegt, dass die Anweisungen zwischen den geschwungenen Klammern wiederholt werden, so lange der Wert der Variable a kleiner als 30 ist.

10.5.2 Die for-Schleife

Eine for-Schleife verwendet immer eine Zählvariable, die ausschlaggebend für die Anzahl der Schleifendurchläufe ist.

```
1 for (Bedingungen)
2 {
3   ...
4 }
```

Für die Bedingungen einer for-Schleife müssen drei Angaben gemacht werden.

Beispiel

```
1 for (int i = 1; i<=10; i++)
2 {
3   ...
4 }
```

Bei einer for-Schleife wird zunächst die Zählvariable definiert und dieser ein Startwert zugewiesen: `int i=1;`

Die Laufzeitbedingung der for-Schleife gibt an, dass die Schleife läuft, solange diese Bedingung erfüllt ist: `i<=10;`

Zuletzt wird festgelegt, wie sich der Wert der Zählvariable nach einem Schleifendurchlauf ändern soll: `i++`

Dies ist eine Kurzschreibweise für `i = i + 1`, was bedeutet, dass der Wert der Zählvariable um eins erhöht wird.

10.6 Funktionen

Meist werden Funktionen dazu verwendet ein Codesegment, das öfters ausgeführt werden soll, nur einmal im Quellcode anzuführen. Dieses Codesegment wird in eine Funktion geschrieben und diese Funktion wird dann jedes mal aufgerufen, wenn man dieses Codesegment ausführen möchte.

Auch kommen Funktionen zum Einsatz um umfangreiche Codesegemente an einer Stelle außerhalb des Hauptprogramms auszulagern um dieses übersichtlicher zu gestalten.

10.6.1 Definition und Aufruf einer einfachen Funktion

Im einfachsten Fall wird eine Funktion vor dem Hauptprogramm definiert und dann an einer beliebigen Stelle im Hauptprogramm aufgerufen.

Nach dem Schlüsselwort `void` wird der Name der Funktion festgelegt. Dabei dürfen im Funktionsnamen keine Sonderzeichen oder Leerzeichen verwendet werden.

```
1 void MeineFunktion()
2 {
3 Anweisung 1;
4 Anweisung 2;
5 Anweisung 3;
6 }
```

Listing 24: Definition einer einfachen Funktion

Um den Code, der in einer Funktion zusammengefasst wurde, auszuführen, wird diese durch Angabe des Funktionsnamens aufgerufen.

```
1 ...
2 MeineFunktion();
3 ...
```

Listing 25: Aufruf einer einfachen Funktion

10.6.2 Definition und Aufruf einer Funktion mit Parameterübergabe

Sollen bei der Ausführung einer Funktion zum Beispiel unterschiedliche Werte Berücksichtigung finden, so kann man diese in Form von Parametern an eine Funktion übergeben. Diese stehen innerhalb der Funktion wie Variablen zur Verfügung.

```
1 void MeineFunktion(uint8_t a, uint8_t b)
2 {
3 Anweisung 1 unter Beruecksichtigung des Wertes a;
4 Anweisung 2 unter Beruecksichtigung des Wertes b;
5 Anweisung 3;
6 }
```

Listing 26: Definition einer Funktion mit Parameterübergabe

Wurde eine Funktion mit Parameterübergabe definiert, so muss die Funktion mit der Übergabe von Werten aufgerufen werden. Die Anzahl und die Typen der Werte müssen dabei genau der Festlegung der Parameter in der Definition entsprechen.

```
1 ...
2 MeineFunktion(40, 10);
3 ...
```

Listing 27: Aufruf einer einfachen Funktion

10.6.3 Definition und Aufruf einer Funktion mit Rückgabewert

Eine Funktion kann auch dazu verwendet werden um bei ihrem Aufruf einen Wert zu ermitteln oder berechnen und diesen dann zurückzugeben. Dabei muss bei der Definition der Typ des Rückgabewerts vor dem Funktionsnamen angegeben werden. Durch den Aufruf von **`return xyz;`** wird der Wert **`xyz`** als Rückgabewert der Funktion zurückgegeben.

```
1 uint8_t MeineFunktion()
2 {
3 uint8_t a;
4 Anweisung 1;
5 Anweisung 2 Berechnung eines Werts fuer a;
6 Anweisung 3;
7 return a;
8 }
```

Listing 28: Definition einer Funktion mit Parameterübergabe

Liefert eine Funktion einen Rückgabewert, so steht dieser nach dem Aufruf der Funktion über ihren Namen im Programmcode an dieser Stelle zur Verfügung. Dabei ist zu beachten, dass der Typ des Rückgabewerts der Funktion bei der weiteren Verwendung berücksichtigt wird.

```
1 ...
2 uint8_t b;
3 b = MeineFunktion();
4 ...
```

Listing 29: Aufruf einer einfachen Funktion

10.6.4 Definition und Aufruf einer Funktion mit Parameterübergabe und Rückgabewert

Meist dienen Funktionen dazu um in ihnen umfangreiche Berechnungen auszulagern. Dazu werden Werte an die Funktion übergeben, damit Berechnungen durchgeführt und abschließend das Ergebnis zurückgegeben.

Dazu werden bei der Definition Parameter, wie auch der Typ des Rückgabewerts und die Rückgabe mit **`return`** festgelegt.

```
1 uint8_t MeineFunktion(uint8_t a, uint8_t b)
2 {
3 uint8_t summe;
4 summe = a + b;
5 return summe;
6 }
```

Listing 30: Definition einer Funktion mit Parameterübergabe und Rückgabewert

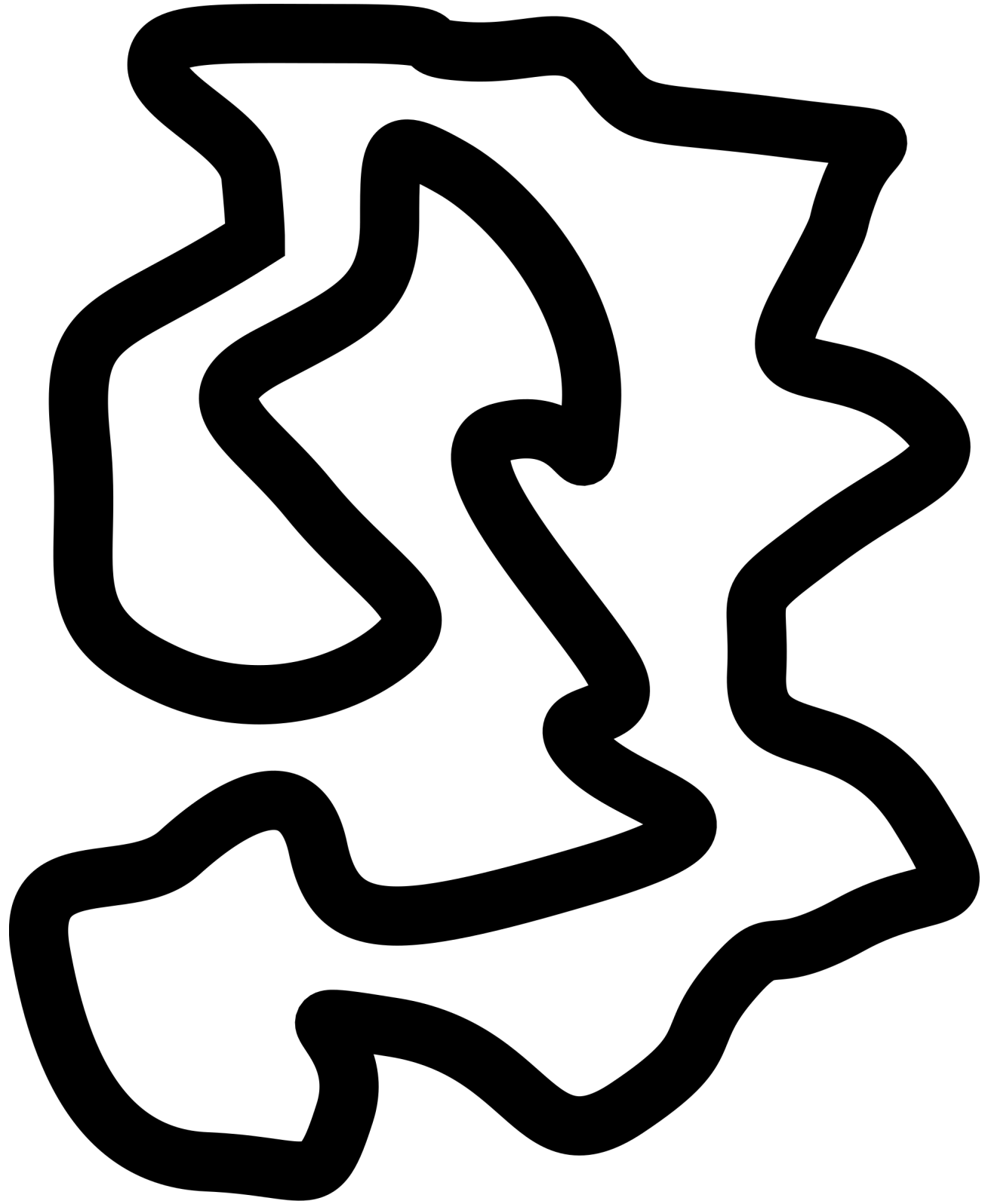
Der Funktion muss beim Aufruf die Anzahl an Werten vom richtigen Typ wie in der Definition festgelegt übergeben werden. Der Rückgabewert steht bei der Ausführung des Programms dann an der Stelle, an der die Funktion aufgerufen wurde zur Verfügung.

```
1 ...
2 uint8_t wert;
3 wert = MeineFunktion(2,3);
4 ...
```

Listing 31: Aufruf einer einfachen Funktion

11 Anhang A Grafiken zum Ausdrucken

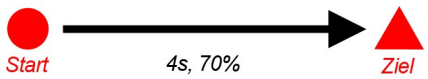
Auf den folgenden Seiten befinden sich einige Grafiken, die ausgedruckt zum Lösen einzelner Aufgaben verwendet werden können.



12 Anhang B Aufgaben Anfänger zum Ausdrucken

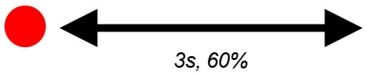
Aufgabe M1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 4 Sekunden mit 70% Motorleistung vorwärts fährt und danach stoppt.



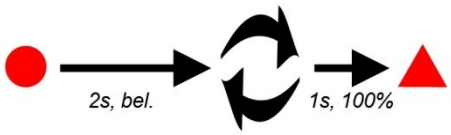
Aufgabe M2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 3 Sekunden mit 60% Motorleistung vorwärts fährt, wieder die gleiche Strecke rückwärts fährt und danach stoppt.



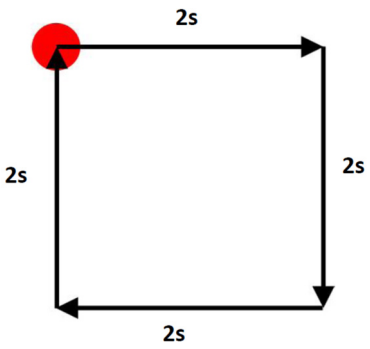
Aufgabe M3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 2 Sekunden mit beliebiger Motorleistung vorwärts fährt, sich auf der Stelle einmal um seine eigene Achse dreht, dann nochmals eine Sekunde mit maximaler Leistung in der ursprünglichen Richtung vorwärts fährt und danach stoppt.



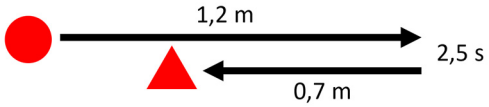
Aufgabe M4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt.



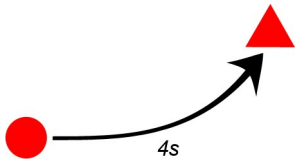
Aufgabe M5

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) genau 1,2 Meter geradeaus fährt, 2,5 Sekunden stehen bleibt und dann 0,7 Meter rückwärts fährt.



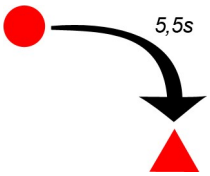
Aufgabe M6

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 4 Sekunden eine leichte Kurve nach links fährt und danach stoppt.



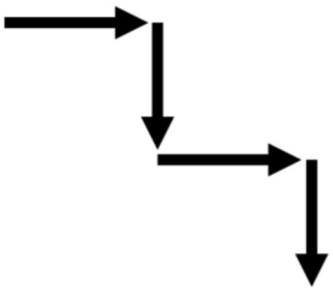
Aufgabe M7

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 5,5 Sekunden eine starke Kurve nach rechts fährt und danach stoppt.



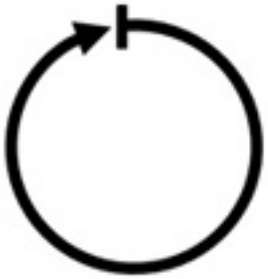
Aufgabe M8

Erstelle ein Programm, mit dem der Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Treppenmuster fährt. Bei den Richtungsänderungen soll er sich immer am Stand drehen.



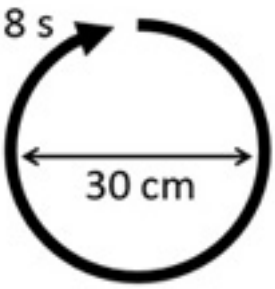
Aufgabe M9

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) einen Kreis fährt und genau an der Ausgangsposition stoppt. Die Motorleistung kann dabei beliebig gewählt werden.



Aufgabe M10

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) einen Kreis mit einem Durchmesser von ca. 30 cm fährt. Dafür soll der Roboter genau 8 Sekunden brauchen.

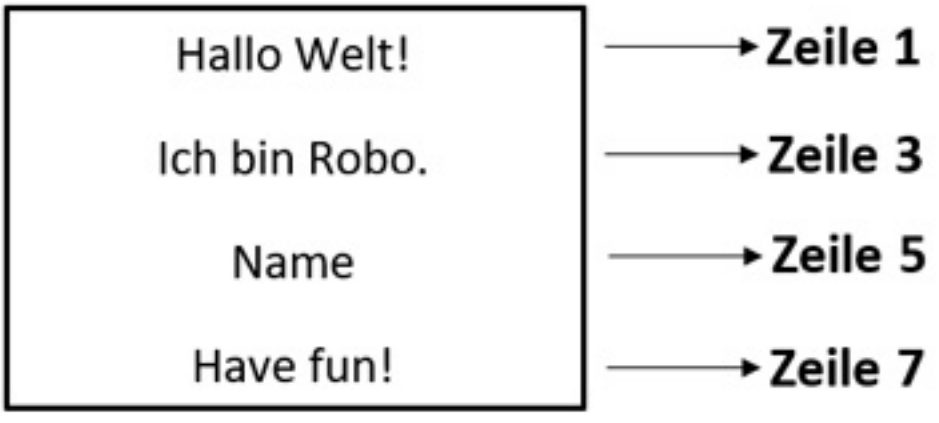


Aufgabe LED1 - Farbenspiel mit der LED

Erstelle ein Programm das mit der eingebauten LED im Bedienfeld des EV3 Roboters alle möglichen Farben und Anzeigemodi nacheinander jeweils für eine Dauer von 2 Sekunden zeigt. Danach soll das Programm automatisch enden.

Aufgabe D1

Erstelle ein Programm, bei dem zentriert in der Mitte des Displays folgender Text ausgegeben wird (setze in der Zeile 5 deinen Namen ein).

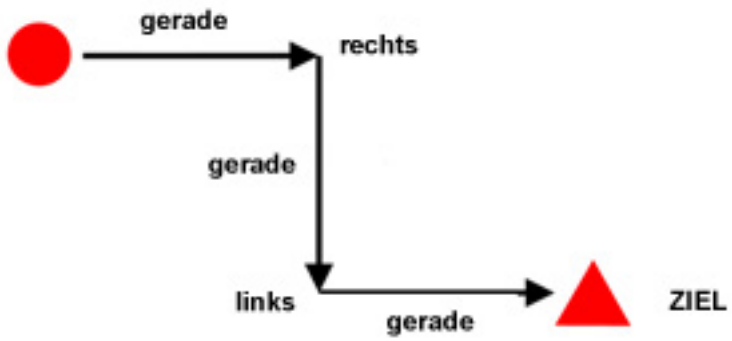


Aufgabe D2

Ergänze das Programm von Aufgabe D1 so, dass eine Zeile nach der anderen in einem zeitlichen Abstand von drei Sekunden erscheint.

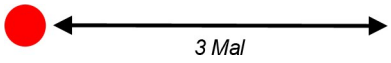
Aufgabe D3

Erstelle ein Programm, bei dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) die vorgegebene Figur nachfährt. Dabei soll der aktuelle Bewegungsablauf mit den in der Grafik angegebenen Bezeichnungen am Display angezeigt werden.



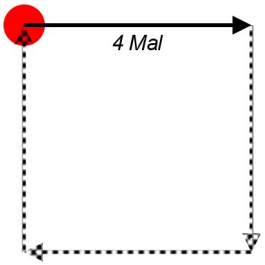
Aufgabe S1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 3 Mal 2 Sekunden mit 85% Motorleistung vorwärts fährt, wieder die gleiche Strecke rückwärts fährt und danach stoppt. Verwende für die mehrfache Ausführung dieses Bewegungsvorgangs eine while-Schleife.



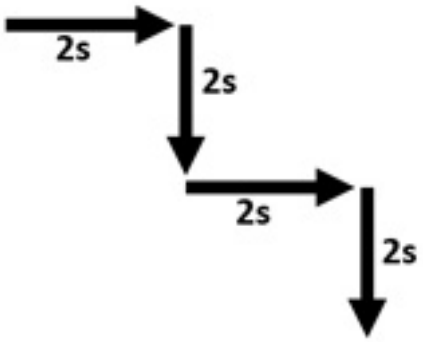
Aufgabe S2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) 4 Mal 2 Sekunden geradeaus fährt, sich um 90 Grad nach rechts dreht und somit seine Bahn ein Quadrat beschreibt. Verwende für die vierfache Ausführung des Bewegungsvorgangs *geradeaus - drehen* eine while-Schleife.



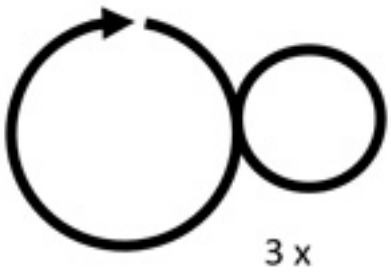
Aufgabe S3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Treppenmuster fährt. Bei den Richtungsänderungen soll er sich immer am Stand drehen. Verwende für sich wiederholende Bewegungsabläufe eine while-Schleife.



Aufgabe S4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) dreimal einen liegenden Achter fährt. Der eine Kreis des Achters soll dabei deutlich größer als der andere Kreis sein. Der Roboter soll dafür zwischen 20 und 25 Sekunden brauchen. Verwende für sich wiederholende Bewegungsabläufe eine while-Schleife.



Aufgabe S5

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) für 10 Sekunden eine Schlangenlinie fährt und danach stoppt. Sich wiederholende Bewegungsabläufe sollen mit einer while-Schleife realisiert werden.

Aufgabe T1

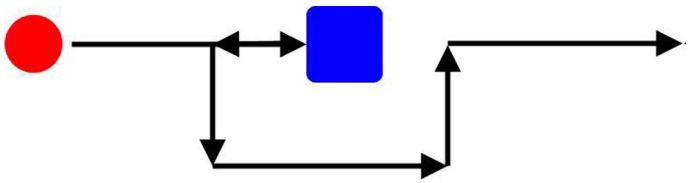
Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) so lange geradeaus fährt, bis er auf ein Hindernis trifft. Dann soll er 1,5 Sekunden gerade zurück fahren und stehen bleiben.

Aufgabe T2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) in einer Endlosschleife immer wieder so lange geradeaus fährt, bis er auf ein Hindernis trifft. Dann soll er 1,5 Sekunden gerade zurück fahren, kurz stehen bleiben und wieder geradeaus fahren, bis er wieder auf das Hindernis trifft. Das Programm soll durch einen Druck auf den Exit-Button beendet werden können.

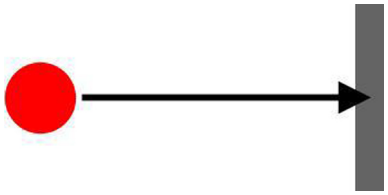
Aufgabe T3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) gerade auf ein Hindernis zufährt und bei Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll danach seinen Weg wie zuvor fortsetzen.



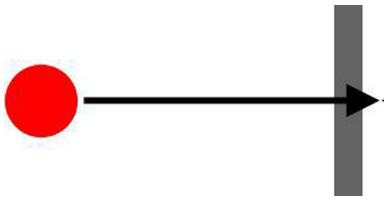
Aufgabe L1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) so lange geradeaus fährt, bis der Lichtsensor auf eine schwarze Linie am Boden trifft. Dort soll der Roboter stehen bleiben.



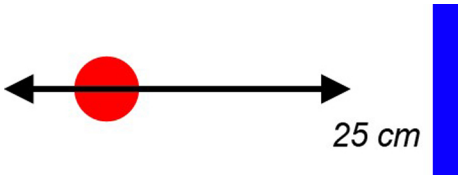
Aufgabe L2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) so lange geradeaus fährt, bis er auf eine schwarze Linie am Boden trifft. Erst wenn der Lichtsensor die Linie überfahren hat und wieder den weißen Untergrund erreicht hat, soll der Roboter stehen bleiben.



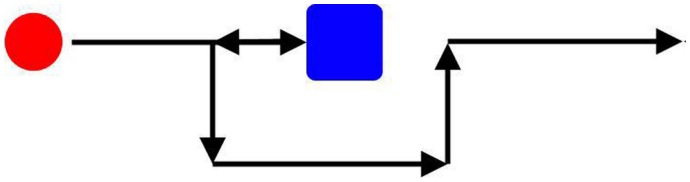
Aufgabe U1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach vorne gerichteten Ultraschall-Sensor an der Vorderseite) so lange geradeaus fährt, bis er sich 25 cm vor einem Objekt entfernt befindet. Dort soll der Roboter für 2 Sekunden stehenbleiben und dann 1 Sekunde zurück fahren.



Aufgabe U2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach vorne gerichteten Ultraschall-Sensor an der Vorderseite) gerade auf ein Hindernis zufährt und 10 cm vor Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll nach dem Hindernis seinen Weg wie zuvor fortsetzen.



13 Anhang C Aufgaben Fortgeschrittene zum Ausdrucken

Aufgabe M11

Erstelle ein Programm mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) den folgenden Bewegungsablauf mit unterschiedlichen Geschwindigkeiten nachfährt. Die Geschwindigkeiten können dabei beliebig gewählt werden:

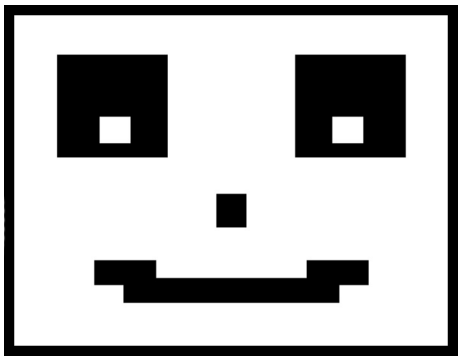
- beide Motoren für 4 Sekunden vorwärts
- Roboter hält für 2 Sekunden an
- Drehung am Stand für 3 Sekunden
- Roboter fährt für 3 Sekunden rückwärts
- Drehung nur mit einem Rad für 5 Sekunden
- Roboter hält für 4 Sekunden an
- Roboter fährt 2 Sekunden lang rückwärts
- Roboter stoppt

Aufgabe Z1

Erstelle ein Programm, das vier ausgefüllte und vier nicht ausgefüllte Rechtecke am Display anzeigt, die sich nicht überschneiden, aber über den gesamten Bildschirmbereich verteilt sein sollen.

Aufgabe Z2

Erstelle ein Programm mit dem das in der Abbildung dargestellte Gesicht am Display gezeichnet wird.



Aufgabe Z3

Erweitere das Gesicht aus Aufgabe Z2 dahingehend, dass sich die Pupillen in den Augen ein paar Mal hin- und herbewegen.

Aufgabe B1

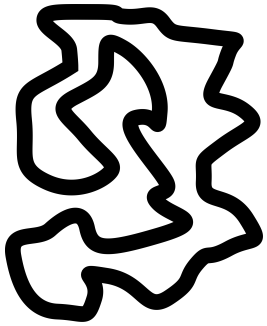
Erstelle ein Programm, das durch Abfrage der Taster die LED am Display steuert. Wird der linke Taster gedrückt, soll die LED-Anzeige rot leuchten, wird der rechte Taster gedrückt, soll die LED-Anzeige grün leuchten. Wird der mittlere Taster gedrückt, dann soll das Programm beendet werden.

Aufgabe T4

Ein Roboter (Standardroboter mit zwei Motoren und einem Touch-Sensor an der Vorderseite) soll in einem durch Wände abgeschlossenen Bereich nach einer bestimmten Zeit die gesamte Fläche abfahren (Rasenmäherroboter). Dazu soll der Roboter beim Berühren einer Wand mit dem Tast-Sensor umdrehen und in die entgegengesetzte Richtung fahren. Dabei soll sich der Roboter nicht genau um 180 Grad drehen. Zusätzlich soll es möglich sein, den Roboter durch einen Druck auf den Exit-Button abzustoppen.

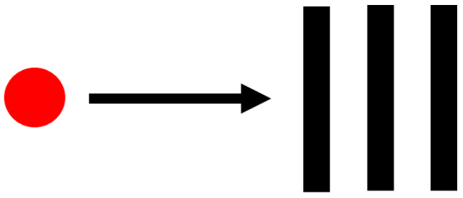
Aufgabe L3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren, einem nach unten gerichteten Licht-Sensor und einem an einer beliebigen Stelle montierten Touch-Sensor), der auf einer strukturierten Oberfläche (z. B. aufgeklebte schwarze Linien auf einer weißen Oberfläche) in Kreisbahnen fährt und der aktuelle Wert des Lichtsensors am Display anzeigt wird. Wird der Touch-Sensor gedrückt, soll der Roboter stehen bleiben.



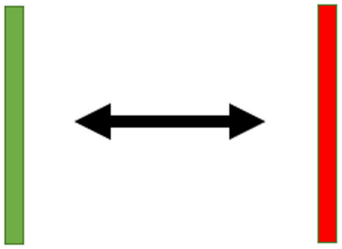
Aufgabe L4

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor) auf einer weißen Oberfläche in Richtung von drei parallelen schwarzen Linien fährt. Der Roboter soll beim Überfahren die Linien mitzählen und die Anzahl am Display anzeigen.



Aufgabe F1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Farb/Licht-Sensor) zwischen einer grünen und roten Linie hin- und herfährt. Das Erkennen der jeweiligen Farbe soll dazu führen, dass der Roboter umdreht und in die entgegengesetzte Richtung fährt. Dabei soll der Roboter bei der grünen Linie eine Drehung nach rechts und bei der roten Linie eine Drehung nach links machen.



Aufgabe U3

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und einem rechtwinkelig zur Seite gerichteten Ultraschall-Sensor) in einem gleichbleibenden Abstand einer Wand entlang fährt. Dazu soll der Abstand mit dem Ultraschallsensor zur Seite gemessen werden.

Grundsätzlich startet der Roboter mit einer konstanten Vorwärtsbewegung. Wird der Abstand zur Wand größer als ein vorgegebener Wert, dann soll der Roboter mit einer leichten Korrektur in Richtung Wand reagieren. Wird der Abstand zu Wand kleiner als ein vorgegebener Wert, dann soll der Roboter sich mit einer leichten Gegenbewegung von der Wand wegbewegen.

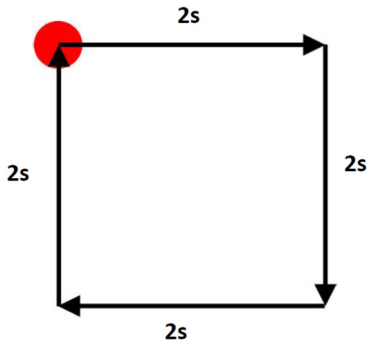
Aufgabe G1

Ein Roboter (Standardroboter mit zwei Motoren und einem Gyro-Sensor) soll zunächst für 2 Sekunden geradeaus fahren. Nach dieser Zeit soll er sich beginnen (langsam!) am Stand zu drehen. Unter Verwendung des Werts des Gyro-Sensors soll sich der Roboter genau 90 Grad drehen, fährt dann wieder für 2 Sekunden geradeaus und bleibt stehen.

Zeige dazu als Hilfe den aktuellen Wert des Gyro-Sensors auf dem Display an!

Aufgabe G2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad und einem Gyro-Sensor) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt. Die sich dabei wiederholenden Bewegungsabläufe sollen mit einer while-Schleife umgesetzt werden und für die exakten Drehungen soll der Gyro-Sensor zum Einsatz kommen.



14 Anhang D Aufgaben Autonome Roboter zum Ausdrucken

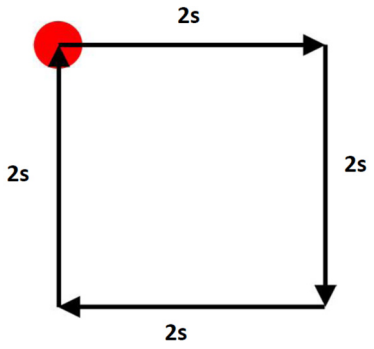
Aufgabe C1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) ein Quadrat fährt: 2 Sekunden geradeaus, um 90 Grad drehen, wieder 2 Sekunden geradeaus usw. bis er wieder an seinem Startpunkt ankommt und stehen bleibt.

Nachdem das Programm am Roboter gestartet wurde, soll die Fahrt des Roboters aber erst nach 3 Sekunden beginnen. Für die Zeitintervalle soll der Counter zum Einsatz kommen und das Programm automatisch beim Erreichen der Zielposition beendet werden. Während dem Abfahren des Pfades soll es möglich sein, das Programm jederzeit durch Drücken des Exit-Buttons zu beenden.

Diese Aufgabe soll auf zwei Arten gelöst werden:

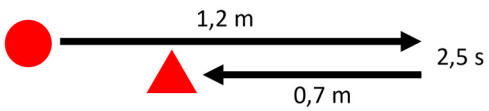
- Der gesamte Bewegungsablauf soll mit einem durchgehenden zeitlichen Verlauf basierend auf den Counter-Werten im Programmcode abgebildet werden.
- Die Bewegung (2s vorwärts und Drehung um 90 Grad), die vier Mal hintereinander gleich abläuft, soll mit einer while-Schleife und einer Zählvariable so implementiert werden, dass der jeweils aktuelle Wert der Zählvariable für die Bestimmung der Zeitintervalle des Counters verwendet wird.



Aufgabe C2

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren und Nachlaufrad) genau 1,2 Meter geradeaus fährt, 2,5 Sekunden stehen bleibt und dann 0,7 Meter rückwärts fährt.

Nachdem das Programm am Roboter gestartet wurde, soll der Roboter gleich losfahren. Die vorgegebenen Wegstrecken sollen unter Verwendung des Counters gefahren werden, das Programm soll aber nicht beendet werden, wenn der Roboter am Zielpunkt angekommen ist. Das Programm muss jederzeit durch Drücken des Exit-Buttons beendet werden können.

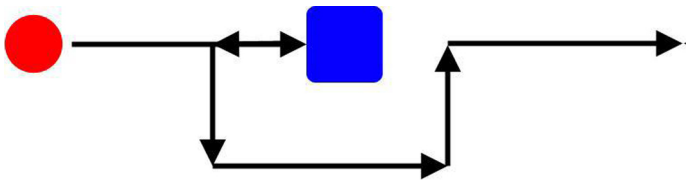


Aufgabe GC1

Erstelle ein Programm, mit dem ein Roboter (Standardroboter mit zwei Motoren, einem Touch-Sensor an der Vorderseite und einem Gyro-Sensor) gerade auf ein Hindernis zufährt und bei Berührung ein Ausweichmanöver startet. Das Hindernis soll dabei rechts oder links umfahren, aber nicht berührt werden. Der Roboter soll danach seinen Weg wie zuvor fortsetzen.

Für die Drehungen des Roboters soll der Gyro-Sensor zum Einsatz kommen und dazu der aktuelle Wert des Sensors beim Berühren des Hindernisses als Ausgangswert dienen.

Für die zu fahrenden Strecken beim Ausweichmanöver um das Hindernis soll der Counter verwendet werden.



Aufgabe S1Linie1

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen.

Die Linie soll grundsätzlich einen geraden Verlauf haben. Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

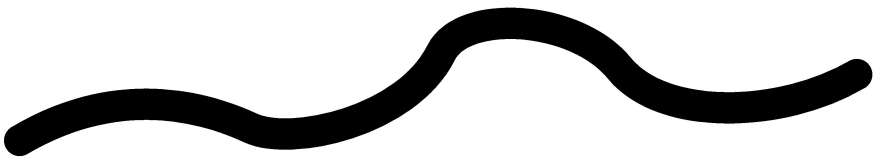
Nun sollen für die herrschenden Lichtverhältnisse der passende Schwellenwert und die Werte für die Geschwindigkeiten so bestimmt werden, dass der Roboter der gesamten Linie bis zum Ende folgen kann, ohne diese zu verlassen.

Aufgabe S1Linie2

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen.

Die Linie soll grundsätzlich einen geraden Verlauf haben, aber Kurven unterschiedlicher Krümmung enthalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Nun sollen für die herrschenden Lichtverhältnisse der passende Schwellenwert und die Werte für die Geschwindigkeiten so bestimmt werden, dass der Roboter dem gesamten Linienverlauf bis zum Ende folgen kann, ohne diesen zu verlassen.

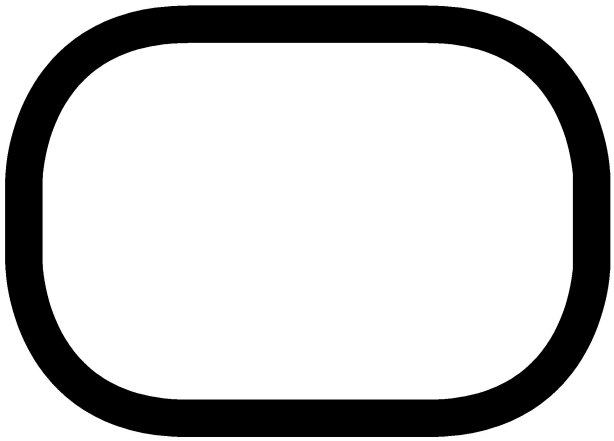


Aufgabe S1Linie3

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll einem Oval gleichen und die Kurven sollen annähernd die gleiche Krümmung aufweisen. Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Ziel ist es, optimale Werte für die Schwelle des Lichtwerts, wie auch für die Grundgeschwindigkeit und die Differenz zwischen beiden Geschwindigkeiten zu ermitteln, sodass der Roboter die vorgegebene Strecke so schnell wie möglich abfährt.

Begleitend dazu soll aber auch berücksichtigt werden, dass der Roboter mit diesen Werten dem Kurvenverlauf zehn Mal hintereinander verlässlich folgen kann, ohne dabei die Linie zu verlassen.



Aufgabe S1Linie4

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll zwei rechte Winkel beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für das Abbiegen bei rechten Winkeln sollen Werte für die Schwelle des Lichtwerts, wie auch für die Geschwindigkeiten ermittelt werden, damit der Roboter sicher und verlässlich die beiden rechten Winkel hintereinander passiert.

Die Aufgabe soll auf zwei Arten gelöst werden, sodass es eine Lösung für das Fahren auf der Innenseite und eine Lösung für das Fahren auf der Außenseite der Linie bzw. der Winkel gibt.

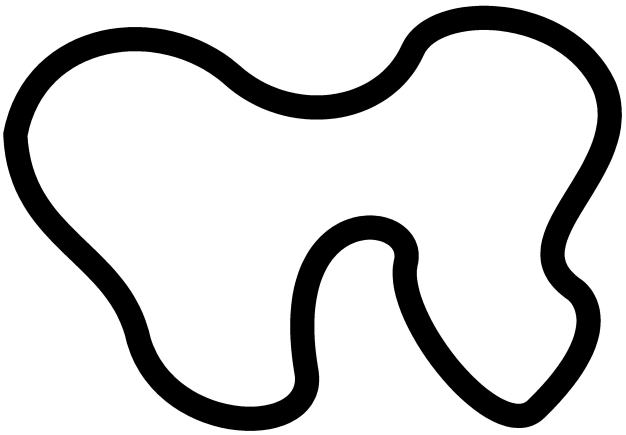
Aufgabe S1Linie5

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Der Linienverlauf soll geschlossen sein und mehrere Kurven mit sehr unterschiedlichen Krümmungen beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für den vorliegenden Kurvenverlauf sollen nun fünf Wertebereiche und dazu optimierte Geschwindigkeiten gefunden werden, damit der Roboter dem Linienverlauf sicher und verlässlich folgen kann.

Dabei soll auch berücksichtigt werden, dass es mit den ermittelten Werten möglich ist den Roboter mehrere Male hintereinander der Linie folgen zu lassen.

Dazu soll ein Programm für das Fahren auf der Innenseite und ein Programm für das Fahren auf der Außenseite erstellt werden.



Aufgabe S1Linie6

Ein Roboter (Standardroboter mit zwei Motoren und einem nach unten gerichteten Licht-Sensor an der Vorderseite) soll einem vorgegebenen Linienverlauf folgen. Die Linie soll zwei rechte Winkel beinhalten (siehe Grafik). Diese Linie könnte z. B. mit einem schwarzen Klebeband (Isolierband) auf einem weißen Untergrund aufgeklebt werden.

Für das Abbiegen bei den rechten Winkeln sollen Werte für fünf Intervalle der Lichtwerte gefunden werden, damit der Roboter sicher und verlässlich bei beiden rechten Winkeln hintereinander abbiegt.

Die Aufgabe soll auf zwei Arten gelöst werden, sodass es eine Lösung für das Fahren auf der Innenseite und eine Lösung für das Fahren auf der Außenseite der Linie bzw. der Winkel gibt.