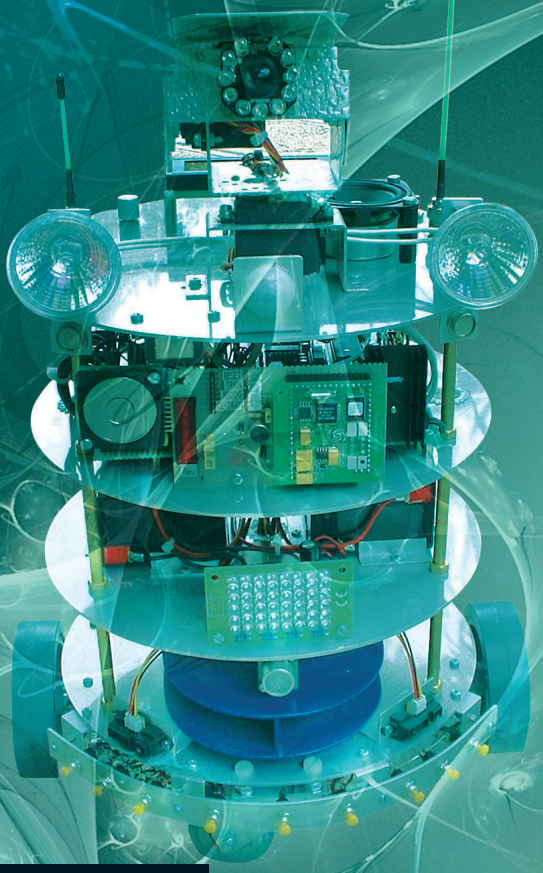
FRANZIS
EXPERIMENTE

Auf CD-ROM

- Kompletter Quellcode zum Buch
- Tools zur einfachen Steuerung von Robotern
- Steuern vom PC aus mit dem PC to Bot Interface
- Datenblätter zu den verwendeten Bauteilen



Ulli Sommer

Roboter selbst bauen

Das große Praxisbuch für Einsteiger und Fortgeschrittene

Auf über 300 Seiten:

- Praktische Bauanleitungen
- Schaltpläne
- Platinenlayouts
- Beispielprogramme

Vorwort

Roboterbauen wird, wie man in den verschiedensten Foren und Fachzeitschriften beobachten kann, immer populärer. Das liegt daran, dass Mikrocontroller und zusätzliche Peripheriebausteine immer günstiger angeboten werden und auch an den Schulen zunehmend in Mikrocontroller, Computer und Robotik unterrichtet wird.

Dieses Buch bietet zahlreiche detaillierte Bauanleitungen mit Schaltplänen und Programmquellcodes. Für den Einsteiger wie auch für den fortgeschrittenen Roboterentwickler dient es als Leitfaden und nützliches Nachschlagewerk. Die Beispielprogramme und Schaltpläne können für eigene Entwicklungen übernommen werden und dienen als Grundbausteine für eigene Ideen und Projekte. Besonderer Wert wurde bei den Anleitungen auf Nachbausicherheit gelegt. Selbstverständlich wurden alle Projekte aufgebaut und ausführlich getestet.

Der Weg zum eigenen Roboter ist für Anfänger mitunter schwer zu bewältigen. Da die Robotik aus einer Kombination von Elektronik, Mechanik und Software besteht, muss der Erbauer handwerkliches Geschick und Grundkenntnisse in den genannten Bereichen für den erfolgreichen Aufbau eines Roboters mitbringen. Man sollte sich jedoch von anfänglichen Misserfolgen nicht abschrecken lassen, denn es macht enorm viel Spaß, seinen selbst gebauten Roboter zum ersten Mal in Aktion zu sehen. Dieses Buch legt den Grundstein für eigene Konstruktionen.

Die Mikrocontroller-Programme dieses Buchs wurden, bis auf den Tischroboter TR-1, mit den Basic-Compiler *Bascom* geschrieben, der auf der Buch-CD als Demoversion enthalten ist. Die einzige Einschränkung der Demoversion ist, dass Programme nur bis 4 KB RAM kompiliert werden können. Alle Funktionen, die in der Vollversion vorhanden sind, werden auch bei der Demoversion bereitgestellt.

Inhaltsverzeichnis

4 PC→Bot-Interface in VB.NET

Auch wenn wir nicht über Tausende von Kilometern hinweg auf dem Mars ein Vehikel steuern und die Bilder von einem anderen Planeten empfangen – interessant ist es allemal, durch die „Augen" des Roboters zu blicken und ihn durch die Sensorwerte und das Videobild ohne direkten Sichtkontakt zu steuern. Unsere Kommandozentrale ist unsere Werkstatt und der fremde Planet unser Garten. Immerhin!

Ein PC→*Bot-Interface* kann man nicht nur zur Steuerung des Antriebs verwenden, sondern auch als Alarmanlage ausbauen oder mit einem Webinterface versehen. Damit könnte man weltweit über einen Internetzugang auf den Roboter zugreifen.

Die nun folgenden Seiten zeigen, wie das Interface aufgebaut ist und welche Befehle man senden und empfangen muss, damit der Roboter reagiert und Messwerte zurücksendet. In Internetforen wird immer wieder die Frage gestellt, wie man das Videobild in ein Formular einbindet – auch diese Frage wird nun beantwortet.

Sie sollten jedoch bereits über Grundkenntnisse in VB.NET verfügen, denn eine komplette Einleitung zur Programmieroberfläche würde den Rahmen dieses Buchs sprengen. Es gibt für Anfänger zahlreiche Foren und Bücher sowie Schulungs-DVDs, die einen guten und leichten Einstieg in das Abenteuer VB.NET bieten.

Nur die wichtigsten Code-Teile des Programms werden folgend beschrieben. Die komplette Software unterliegt der *GNU* (General Public Licence) und darf verändert und weiterentwickelt werden, solange sie ebenfalls der GNU unterliegt. Auf der CD zum Buch sind der Quellcode und eine fertig kompilierte Version des PC→Bot-Interfaces vorhanden.

4.1 Visual Basic.NET

Mit der Einführung von dot.NET ändert sich auch die Philosophie der Anwendungsentwicklung. Das Konzept ist gesamtheitlich neu, auch wenn es teilweise Züge trägt, die an Java erinnern. Natürlich hat sich an der fundamentalen Sprachsyntax nichts geändert. Deklarationen, Schleifen und bedingte Anweisungen gibt es in jeder Sprache. Es ist eigentlich nur die Frage, wie diese Konstrukte in der jeweiligen Sprache eingesetzt werden und welche Grenzen die Sprache vorgibt. Obwohl das objektorien-

tierte Programmieren schon seit vielen Jahren bekannt ist, sind nicht einmal alle professionellen Entwickler in der Lage, auf dieser Basis Programme zu entwickeln. Manche wehren Sie sich sogar heftig gegen die Denkweise in Klassen und Objekten, denn zu tief reichen die Wurzeln der prozeduralen Programmierung.

Mit der dot.NET-Plattform befinden wir uns am Anfang einer neuen Ära in der Programmierung, denn dot.NET läutet eine neue Dimension in der Anwendungsentwicklung ein.

Visual Basic.NET deutet es an: Die Sprache heißt *Visual Basic*. Aber das Suffix .NET (. = dot) kommt einem Versprechen gleich: Visual Basic .NET ist komplett neu und kann nicht mehr mit der Sprache verglichen werden, die bis zur Version 6.0 so viele begeisterte Anhänger gefunden hatte. Wenn Sie bereits Erfahrungen mit einer der alten Versionen gesammelt haben, fangen Sie praktisch fast wieder von vorn an. Der Quellcode ist immer noch offensichtlich als Visual-Basic-Code identifizierbar – aber das ist auch bereits alles. Vergessen Sie die Einschränkungen, denen Sie bisher unterworfen waren, und vergessen Sie das Studium von „Tipps und Tricks“ in den verschiedenen Medien, um eine aus dem üblichen Rahmen fallende Idee zu realisieren. Sie stehen praktisch auf der gleichen Stufe wie ein Kollege, der C++ oder C#, die neue Sprache zur .NET-Plattform, einsetzt – mit allen Konsequenzen, die sich daraus ergeben, wie beispielsweise der intensiven Einarbeitung.

Visual Basic war bis einschließlich Version 6.0 nicht objektorientiert, zumindest nicht im eigentlichen Sinne. Früher rümpften gestandene C++- oder Java-Entwickler bei Diskussionen die Nase. Visual Basic wurde nicht ernst genommen, häufig sogar ein wenig belächelt. Ungerechtfertigt, denn viele alltägliche Problemstellungen bei der Anwendungsentwicklung konnten schnell und effizient gelöst werden.

Ein komplett objektorientierter Ansatz, dieselbe Klassenbibliothek für alle Sprachen und derselbe Compiler Visual Basic.NET ist auch für Sprachumsteiger von C++, Java, Delphi usw. zu einer durchaus gleichwertigen Alternative geworden.

Alle Sprachen, ob C++.NET, C#.NET, Java.NET und VB.NET, die von Microsoft bereitgestellt werden, basieren in Zukunft auf .NET. Damit steht für jede Sprache die gleiche Klassenbibliothek zur Verfügung. Dazu kommt, dass alle Programme, egal mit welcher Sprache sie geschrieben wurden, gleich schnell in der Abarbeitung sind.

Um nun aber starten zu können, muss man die aktuelle Version von Visual Basic.NET von der Microsoft Homepage herunterladen. Der direkte Link dazu ist:

http://msdn.microsoft.com/vstudio/express/downloads/

Microsoft stellt sein geraumer Zeit eine Expressversion mit ein paar Einschränkungen zum freien Download bereit. Man sollte sich jedoch registrieren lassen, da sonst die Version nach 30 Tagen nicht mehr läuft.

Die Einschränkungen in der kostenlosen Expressversion sind für die in diesem Buch beschriebenen Roboteranwendungen vernachlässigbar. Im Gegensatz zu VB6 muss man keine Laufzeitkomponenten mehr mitliefern, da diese bereits in der Framework (Klassenbibliothek) enthalten sind. Das heißt, dass man, alleine mit der erstellten EXE-Datei, das Programm auf jedem Rechner laufen lassen kann, auf dem das Framework installiert ist. Da viele Anwendungen dies mittlerweile benötigen, verwenden wir ein relativ kompaktes Programm von wenigen Kilobytes und es läuft somit auf fast allen Windows-Betriebssystemen, inklusive Vista.

4.2 PC→Bot-Interface – Funktionserklärung

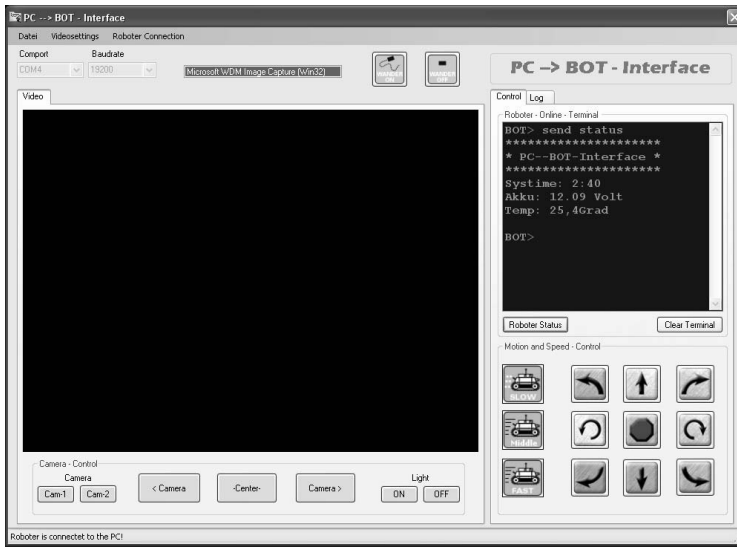


Abb. 4.1: Die Bot-Bedienoberfläche. Wie man sieht: sehr aufgeräumt!

Die Steuerung des Antriebs und der Aktoren übernimmt ein 433-Mhz-Telemetrie-Datentransceiver (z. B. der Firma *Devantech* Ltd., Unit 2B Gilray Road, Diss, Norfolk/England), der an der seriellen Schnittstelle angeschlossen ist. Drücken Sie einen Button in der Bedieneinheit, werden die Befehlsadresse, der Befehl und die Checksumme an die serielle Schnittstelle gesendet und per Funk an den Roboter übertragen. Der AVR-Controller wiederum wertet die Daten aus und weist sie den dafür vorgesehen Sub-Routinen zu (z. B. „Fahre geradeaus“ oder „Drehe dich nach links“).

Damit man sieht, wo der Roboter gerade umherfährt, hat man noch ein Videoframe im Interface, das das über eine TV-Karte eingespeiste Videosignal des Videoempfängers (2,4 GHz) wiedergibt.

Aufbau des Telemetrieprotokolls:

```
Befehlsadresse | Befehl | Checksumme (CRC)
```

Man sendet immer nur drei Byte an den Roboter, mehr wird nicht benötigt. Der Roboter wiederum sendet darauf eine Rückinfo, die sagt, dass der gesendete Befehl am Roboter angekommen ist.

Beispiel: Man sendet „Fahre vorwärts“ und der Roboter gibt, nach Empfang und Ausführung, „Drive forward“ zurück, dass man im Terminal angezeigt bekommt.

Was der Roboter zurückgibt, kann man in der entsprechenden Routine selbst festlegen.

4.3 Die User-Bedienoberfläche

Die Bedienoberfläche besteht aus dem Video Frame, der Kommandoebene, über die man den Roboter steuert, und dem Terminal, über das man Kommandos, Status und alle möglichen Telemetriedaten angezeigt bekommt, die man zuvor im Roboter hinterlegt hat.

Abb. 4.2: Das Videoframe. Hier wird das empfangene Videosignal wiedergegeben

Das Videoframe ist eine Picturebox, die in die Programmieroberfläche bereits integriert ist. Sie dient eigentlich dazu, Bilder anzuzeigen, z. B. für einen Bildbetrachter

oder um Linien zu malen. Indirekt machen wir nichts anderes: Wir kopieren über den Treiber das Videobild in die Picturebox.

Abb. 4.3: Die Menüleiste

In den *Video settings* kann man diverse Einstellungen nach Start des Treibers tätigen. Die Start- und Stopp-*Kameras* dienen dazu, den Treiber bzw. die Videowiedergabe zu starten. Unter *Video Format* legt man die Auflösung sowie die Pixeltiefe und die Komprimierung fest (im beschriebenen Fall 640 x 480). Kann der Videokonverter diese Auflösung nicht bereitstellen, kann man eine kleinere (z. B. 320 x 240) verwenden. Das Bild wird dann gleichmäßig gestreckt, damit es die ganze Box ausfüllt, Nachteil ist, dass das Bild etwas „körnig" wird.

Bei *Video Source* werden die Videoquelle, der Videoeingang und diverse andere Einstellungen, je nach Treiber der TV-Karte, vorgenommen.

Video Overlay on und *off*: Ältere Fernsehkarten (Video-Overlay-Karten) erzeugen das analoge Monitorsignal selbst und mischen es mit dem Signal der Grafikkarte. Nachteilig ist dabei, dass das Signal der Grafikkarte oft leidet, wodurch die Grafik verwischt oder abgedunkelt wird. Neuere TV-Karten hingegen schreiben die Bildinformation direkt in den Bildwiederholspeicher der Grafikkarte. Hier muss man testen, was die TV-Karte unterstützt. Wenn man ein grünes Bild hat, muss man *Overlay* auf „on" schalten. Der Nachteil bei Overlay-Betrieb ist, dass die Screenshot-Funktion nur noch schwarze oder grüne Bilder macht, da das Bild nicht mehr über den Prozessor (CPU) aufbereitet wird.

Über *Roboter connect* stellt man die serielle Verbindung zum Roboter her. Der Bot liefert, sofern er eingeschaltet ist, gleich eine Statusinfo zurück und quittiert damit den Verbindungsaufbau.

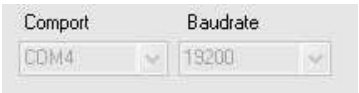


Abb. 4.4: Comport Settings

Unterhalb der Menüzeile befinden sich die *Comport settings*. Mit ihnen legt man fest, welchen Comport (Com-Schnittstelle) man benutzt und auf welche Baudrate er ein-

gestellt werden soll. Unser Interface arbeitet im mitgelieferten Beispiel mit 19.200 Baud, andere Baudraten sind aber auch möglich.

Abb. 4.5: Wandermodus ein und aus

Oben im Formular befinden sich zwei Buttons, mit denen man den Wandermodus ein- und ausschaltet. Der Wandermodus ist im Grunde nur die Wahl zwischen Handsteuerung und autonomem Betrieb. Im Bascom-Code auf Ihrem Roboter ist diese Routine zwar hinterlegt, aber ohne auszuführenden Code. Hier können Sie Ihre eigenen Ideen hineinschreiben (z. B. „Radar Drive“ oder „Finde den besten Weg“).

Abb. 4.6: In dieser Groupbox befinden sich die Kamera-Steuerelemente

Unterhalb des Videoframes befindet sich die Kamerasteuerung *Camera-Control*. Hier kann man auswählen, welche Kamera man verwendet und wohin man sie schwenkt (links, rechts, Mittelstellung). Außerdem kann man hier die Beleuchtung aus- und einschalten.

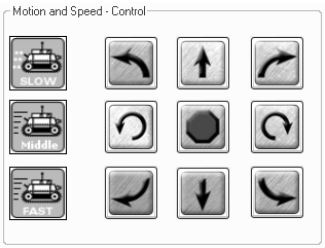


Abb. 4.7: Das Steuerfeld zum Navigieren

Motion-and-Speed-Control dient zum Navigieren des Roboters. Je nachdem, welchen Button man drückt, fährt der Bot geradeaus, dreht sich oder bleibt stehen. Der Clou ist, dass er nur solange fährt, wie die linke Maustaste gedrückt ist. Beim Lösen sendet das Interface sofort ein Stoppkommando und der Bot bleibt stehen. Dieser Trick wird mit der Mouse_up- und Mouse_down-Methode realisiert.

Mit den *Speed*-Buttons legt man die Fahrgeschwindigkeit fest. Beim Einschalten ist die langsamste Geschwindigkeit als Vorwahl definiert.

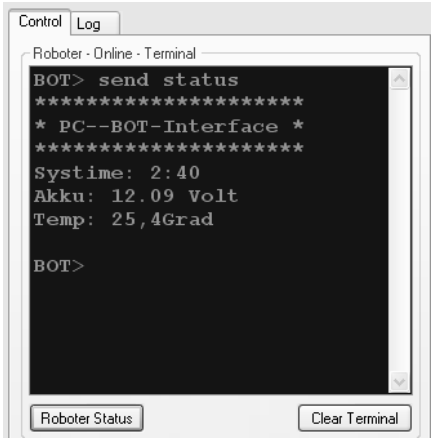


Abb. 4.8: Terminal-Fenster für die Telemetrie-Ausgabe

Das *Terminal-Fenster*, das im schicken „Commodore-C64"-Design gehalten ist, zeigt die Daten des Roboters an. Bei Betätigung des Buttons *Roboter Status* meldet er sich mit PC➔Bot-Interface, der Systemzeit (wie lange er in Minuten und Sekunden seit dem Einschalten in Betrieb ist), seinem Akkuzustand in Volt und der Außentemperatur.

Was man ausgibt, kann man im Code des Roboters selbst festlegen.

Als zweiten TAB haben Sie noch *Log*. Hier wird mitprotokolliert, wann die Verbindung zum Roboter aufgenommen wurde und welche Befehle in dieser Zeit gesendet wurden.

4.4 Das Übertragungsprotokoll

Beim Betätigen der Buttons sendet man einen Befehl an den Roboter, der sich aus 3 Byte zusammensetzt.

Die folgende Befehlstabelle zeigt, welche Befehle welchen Byte-Werten zugeordnet sind. Da der VB.NET mit Unicode arbeitet, kann man 127 Befehle verteilen, die jeweils 127 Zustände bzw. Unterkommandos enthalten dürfen. Würde man einen ASCII-String senden, wäre hier noch viel mehr möglich – aber je mehr Zeichen übertragen werden, desto länger dauert es, bis sie ordnungsgemäß übertragen worden sind. Vor allem kommt es bei längeren Strings vor, dass sich Fehler durch Funkstörungen einschleichen und das Kommando dadurch nicht ausgeführt wird. Durch die Adressvergabe und die Übertragung der Checksumme „CRC" ist man auf der sicheren Seite. Fehlkommandos kommen nicht zustande und die Funkreichweite, in der die Daten noch sicher übertragen werden, steigt.

Das Sendeprotokoll

```
Befehlsadresse | Befehl | Checksumme (CRC)
```

Befehlsadresse	Befehl	Erklärung
1	127	Roboterstatus abfragen
2	10	manueller Betrieb
3	11	autonomer Betrieb
4	1, 2, 3	Geschwindigkeit in drei Stufen
5	127	Antrieb stopp
6	100	vorwärts
7	101	rückwärts
8	102	links drehen auf der Stelle
9	103	rechts drehen auf der Stelle
10	104	links fahren vorwärts
11	105	rechts fahren vorwärts
12	106	links fahren rückwärts
13	107	rechts fahren rückwärts
14	90	Kamera drehen stoppen
15	91	Kamera links drehen
16	92	Kamera rechts drehen
17	93	Kamera Mittelstellung
18	80	Kamera 1
19	81	Kamera 2
20	70	Licht aus
21	71	Licht ein
22	100	Verbindung zum PC hergestellt

4.5 Ein- und Aufbau des Video-Frame in .NET

Um den TV-Kartentreiber anzusprechen, wird der WDM-Treiber (Image Capture Win32) verwendet. Diesem Treiber werden die Befehle über eine API namens „Sendmessage“ gesendet.

Dazu benötigt man folgende Komponenten.

- Picturebox
- API Sendmassage
- API SetWindowPos
- API DestroyWindow
- API apCreateCaptureWindowA
- capGetDriverDescriptionA
- AVIGetStatus
- AVIGetCaps
- AVIFileNames
- AVICallBack
- Buttons: Start Card, Stop Card, Video Format, Video Source
- Konstanten für den Verweis auf das Video-Handling

```
Imports System.Runtime.InteropServices
```

Zunächst muss man die Variablen in der Main-Class anlegen:

```
Public Structure tagCAPSTATUS
  Public uiImageWidth As Long
  Public uiImageHeight As Long
  Public fLiveWindow As Long
  Public fOverlayWindow As Long
  Public fScale As Long
  Public point As Long
  Public fUsingDefaultPalette As Long
  Public fAudioHardware As Long
  Public fCapFileExists As Long
  Public dwCurrentVideoFrame As Long
  Public dwCurrentVideoFramesDropped As Long
  Public dwCurrentWaveSamples As Long
  Public dwCurrentTimeElapsedMS As Long
  Public hPalCurrent As Long
  Public fCapturingNow As Long
  Public dwReturn As Long
  Public wNumVideoAllocated As Long
  Public wNumAudioAllocated As Long
End Structure

Public Structure tagCAPDRIVERCAPS
  Public wDeviceIndex As Long
  Public fHasOverlay As Long
  Public fHasDlgVideoSource As Long
  Public fHasDlgVideoFormat As Long
  Public fHasDlgVideoDisplay As Long
  Public fCaptureInitialized As Long
  Public fDriverSuppliesPalettes As Long
  Public hVideoIn As Long
```

```
 Public hVideoOut As Long
  Public hVideoExtIn As Long
  Public hVideoExtOut As Long
End Structure

Public Structure tagCAPTUREPARAMS
  Public dwRequestMicroSecPerFrame As Long
  Public fMakeUserHitOKToCapture As Long
  Public wPercentDropForError As Long
  Public fYield As Long
  Public dwIndexSize As Long
  Public wChunkGranularity As Long
  Public fUsingDOSMemory As Long
  Public wNumVideoRequested As Long
  Public fCaptureAudio As Long
  Public wNumAudioRequested As Long
  Public vKeyAbort As Long
  Public fAbortLeftMouse As Long
  Public fAbortRightMouse As Long
  Public fLimitEnabled As Long
  Public wTimeLimit As Long
  Public fMCIControl As Long
  Public fStepMCIDevice As Long
  Public dwMCIStartTime As Long
  Public dwMCIStopTime As Long
  Public fStepCaptureAt2x As Long
  Public wStepCaptureAverageFrames As Long
End Structure

'Verweis auf das Video-Handling
Const WM_CAP_start As Short = &H400S
Const wm_cap_driver_connect As Integer = WM_CAP_start + 10
Const WM_CAP_DRIVER_DISCONNECT As Integer = WM_CAP_start + 11
Const WM_CAP_DRIVER_GET_NAME As Integer = WM_CAP_start + 12
Const WM_CAP_DRIVER_GET_VERSION As Integer = WM_CAP_start + 13
Const WM_CAP_DRIVER_GET_CAPS As Integer = WM_CAP_start + 14
Const WM_CAP_FILE_SET_CAPTURE_FILE As Integer = WM_CAP_start + 20
Const WM_CAP_FILE_GET_CAPTURE_FILE As Integer = WM_CAP_start + 21
Const WM_CAP_FILE_SAVE_AS As Integer = WM_CAP_start + 23
Const WM_CAP_FILE_SET_INFOCHUNK As Integer = WM_CAP_start + 24
Const WM_CAP_FILE_SAVE_DIB As Integer = WM_CAP_start + 25
Const WM_CAP_EDIT_COPY As Integer = WM_CAP_start + 30
Const WM_CAP_SET_AUDIOFORMAT As Integer = (WM_CAP_start + 35)
Const WM_CAP_GET_AUDIOFORMAT As Integer = (WM_CAP_start + 36)
Const WM_CAP_DLG_VIDEOFORMAT As Integer = (WM_CAP_start + 41)
Const WM_CAP_DLG_VIDEOSOURCE As Integer = (WM_CAP_start + 42)
Const WM_CAP_DLG_VIDEODISPLAY As Integer = (WM_CAP_start + 43)
Const WM_CAP_GET_VIDEOFORMAT As Integer = (WM_CAP_start + 44)
Const WM_CAP_SET_VIDEOFORMAT As Integer = (WM_CAP_start + 45)
Const WM_CAP_DLG_VIDEOCOMPRESSION As Integer = (WM_CAP_start + 46)
Const wm_cap_set_preview As Integer = (WM_CAP_start + 50)
```

```
Const wm_cap_set_overlay As Integer = (WM_CAP_start + 51)
Const WM_CAP_SET_PREVIEWRATE As Integer = (WM_CAP_start + 52)
Const WM_CAP_SET_SCALE As Integer = (WM_CAP_start + 53)
Const WM_CAP_GET_STATUS As Integer = (WM_CAP_start + 54)
Const WM_CAP_SET_SCROLL As Integer = (WM_CAP_start + 55)
Const WM_CAP_SINGLE_FRAME_OPEN As Integer = (WM_CAP_start + 70)
Const WM_CAP_SINGLE_FRAME_CLOSE As Integer = (WM_CAP_start + 71)
Const WM_CAP_SINGLE_FRAME As Integer = (WM_CAP_start + 72)
Const WM_CAP_GRAB_FRAME As Integer = (WM_CAP_start + 60)
Const WM_CAP_GRAB_FRAME_NOSTOP As Integer = (WM_CAP_start + 61)
Const WM_CAP_SEQUENCE As Integer = (WM_CAP_start + 62)
Const WM_CAP_SEQUENCE_NOFILE As Integer = (WM_CAP_start + 63)
Const WM_CAP_SET_SEQUENCE_SETUP As Integer = (WM_CAP_start + 64)
Const WM_CAP_GET_SEQUENCE_SETUP As Integer = (WM_CAP_start + 65)
Const WM_CAP_STOP As Integer = WM_CAP_start + 68
Const WM_CAP_ABORT As Integer = WM_CAP_start + 69
Const WM_CAP_SET_CALLBACK_ERROR As Integer = (WM_CAP_start + 2)
Const WM_CAP_SET_CALLBACK_STATUS As Integer = (WM_CAP_start + 3)
Const WM_CAP_SET_CALLBACK_YIELD As Integer = (WM_CAP_start + 4)
Const WM_CAP_SET_CALLBACK_FRAME As Integer = (WM_CAP_start + 5)
Const WM_CAP_SET_CALLBACK_VIDEOSTREAM As Integer = (WM_CAP_start +
      6)
Const WM_CAP_SET_CALLBACK_WAVESTREAM As Integer = (WM_CAP_start +
      7)
Const WM_CAP_GET_USER_DATA As Integer = (WM_CAP_start + 8)
Const WM_CAP_SET_USER_DATA As Integer = (WM_CAP_start + 9)
Const WM_CAP_SET_CALLBACK_CAPCONTROL As Integer = (WM_CAP_start +
      85)
Const WS_CHILD As Integer = &H40000000
Const WS_VISIBLE As Integer = &H10000000
Const SWP_NOMOVE As Short = &H2S
Const SWP_NOSIZE As Short = 1
Const SWP_NOZORDER As Short = &H4S
Const HWND_BotTOM As Short = 1
Dim iDevice As Integer = 0 ' Geräte ID
Dim hHwnd As Integer ' Gerätekontext-Handle

'jetzt kommen die APIs
Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
 (ByVal hwnd As Integer, ByVal wMsg As Integer, ByVal wParam As
       Integer, _
 <MarshalAs(UnmanagedType.AsAny)> ByVal lParam As Object) As
       Integer


Declare Function SetWindowPos Lib "user32" Alias "SetWindowPos"
       (ByVal hwnd As Integer, _
 ByVal hWndInsertAfter As Integer, ByVal x As Integer, ByVal y As
       Integer, _
 ByVal cx As Integer, ByVal cy As Integer, ByVal wFlags As Integer)
       As Integer
```

```
Declare Function DestroyWindow Lib "user32" (ByVal hndw As Integer)
As Boolean

Declare Function capCreateCaptureWindowA Lib "avicap32.dll" _
 (ByVal lpszWindowName As String, ByVal dwStyle As Integer, _
  ByVal x As Integer, ByVal y As Integer, ByVal nWidth As Integer, _
  ByVal nHeight As Short, ByVal hWndParent As Integer, _
  ByVal nID As Integer) As Integer

Declare Function capGetDriverDescriptionA Lib "avicap32.dll"
    (ByVal wDriver As Short, _
  ByVal lpszName As String, ByVal cbName As Integer, ByVal lpszVer
    As String, _
  ByVal cbVer As Integer) As Boolean

Public Declare Function AVIGetStatus Lib "user32" Alias
    "SendMessageA" (ByVal hWnd As Long,  ByVal wMsg As Long, ByVal
    wParam As Long, ByRef lParam As tagCAPSTATUS) As Long
Public Declare Function AVIGetCaps Lib "user32" Alias
    "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal
    wParam As Long, ByRef lParam As tagCAPDRIVERCAPS) As Long
Public Declare Function AVIFileNames Lib "user32" Alias
    "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal
    wParam As Long, ByVal lParam As String) As Long
Public Declare Function AVICallBack Lib "user32" Alias
    "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal
    wParam As Long, ByVal lParam As Long) As Long
```

Beim Programmstart muss man zuerst nach den Treibern suchen.

```
Private Sub Main_Load(ByVal sender As Object, ByVal e As
       System.EventArgs) Handles Me.Load
  LoadDeviceList()
  If lstDevices.Items.Count > 0 Then
   StopKameraStripMenuItem.Enabled = False
   lstDevices.SelectedIndex = 0
   StartKameraStripMenuItem.Enabled = True
  Else
   lstDevices.Items.Add("No Capture Device")
   StopKameraStripMenuItem.Enabled = False
  End If

  picCapture.SizeMode = PictureBoxSizeMode.StretchImage
End Sub
```

LoadDeviceList() sucht den Treiber.

```
Private Sub LoadDeviceList()
  Dim strName As String = Space(100)
```

10.13 Der Tischroboter TR-1

Ein ideales Einsteigerprojekt ist der Tischroboter TR-1 von Jürgen Schreier, der dieses Projekt bereitgestellt hat.

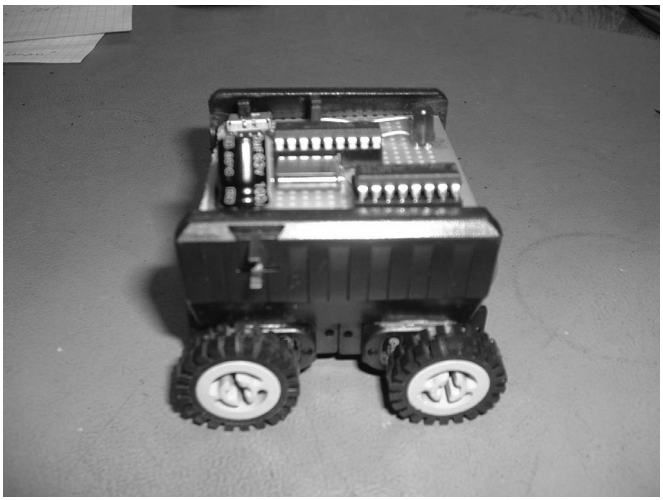


Abb. 10.35: Der fertige Roboter von der Seite

Einen kleineren Roboter kann man nur entwickeln, wenn man einen kleinen Prozessor verwendet, der auf einer kleinen Platine untergebracht werden kann. Hier eignet sich ein PIC16F84A-04, ein kleiner 18-beiniger IC, als Steuerprozessor.

Als Erstes muss man die vier Mini-Servos modifizierten. Dazu zerlegt man die Servos und entfernt den Anschlag im Getriebe, damit sich der Servo, wie ein Getriebemotor, durchgehend drehen kann. Der Istwertgeber (Potenziometer) wird geöffnet, die Widerstandsbahn entfernt und der Anschlag weggebogen, so dass nur noch die Lagerung des Potenziometers aktiv ist. Die Ansteuerplatine wird ebenfalls komplett entfernt. Dann werden noch zwei dünne Drähte mit etwa 0,5 mm² an den Motor angelötet und nach außen geführt. Nun kann man den Servo wieder zusammenbauen und hat einen sehr günstigen kleinen Getriebemotor, der zudem sehr kraftvoll ist!

Danach klebt man mit Zweikomponentenkleber an die Servodrehscheibe ein kleines Lego-Rad. Nachdem der Klebstoff getrocknet ist, klebt man die vier Servos an ein Plastikgehäuse (ausgediente Handyschale) und erhält so einen Allradantrieb.

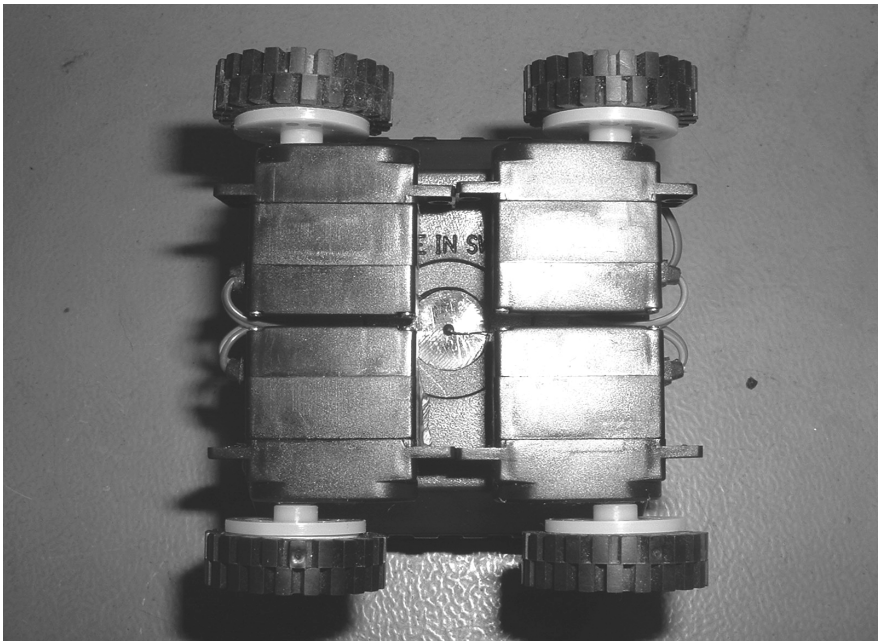


Abb. 10.36: So sieht der fertige Antrieb von unten aus

Ein fahrbares Untergestell ist nun hergestellt. Um das Fahrgestell die ersten Runden drehen zu lassen, könnte man es nun (kabelgebunden) an eine 5-Volt-Gleichspannung anschließen – aber man wird schnell feststellen, dass sich zwar die Räder drehen, das Fahrgestell sich aber nicht vorwärts bewegt. Die verwendeten Materialien sind so leicht, dass die Reifen des Fahrgestells keinerlei Traktion auf die Tischoberfläche ausüben. Die Lösung ist hier ein Mehr an Gewicht durch die Platine, den 9-V-Batterieblock etc. Erst dann kann man testen, was passiert, wenn man das rechte und das linke Reifenpaar gegenläufig anschließt, so dass das Gestell sich wie ein Panzer auf der Stelle dreht.

Es gilt also nun, die Prozessor-Steuerplatine für den Roboter zu konstruieren. Den PIC16F84 bringt man in der Mitte der Platine an. Ferner wird ein 5-V-Stabilisierungs-IC 78T05CT benötigt, um die 9 Volt der Blockbatterie in 5 Volt umzuwandeln. Der PIC braucht noch einen 4-MHz-Quarz, ein Ein-/Aus-Schiebeschalter muss auf die Platine sowie das Wichtigste: ein Motortreiber-IC.

Ideal ist für so einen kleinen Roboter der L293D. Eine rote LED zur Kontrolle kommt ebenfalls noch hinzu. Nachdem die Platine fertiggestellt ist, kann man den neuen Prozessor programmieren. Danach werden die Servos noch mit der Platine verbunden, ein 9-Volt-Batterieklipp installiert und eine Batterie angeschlossen.

Zur Programmierung der kleineren PICs kann man ein PIC-Programmier- und Experimentier-Board von Conrad Electronic verwenden (Bestellnummer: 191020).

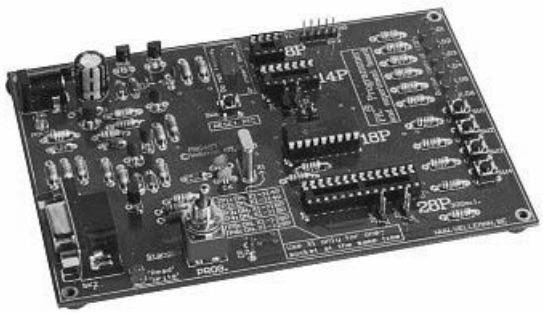


Abb. 10.37: PIC-Programm von der Fa. Vellemark, Bezug über Conrad Electronic

Das Board ist nach ca. 1,5 Stunden zusammengelötet und betriebsbereit.

Bevor man mit der Programmierung beginnt, prüft man die Spannung am IC-Sockel, die den Controller versorgt. Diese Spannung muss 5 V betragen. Nachdem man die Spannungen und die Platinen auf Löt- und Bestückungsfehler kontrolliert hat, kann man mit der Programmierung beginnen.

Um nun das Controllerboard zu testen, lässt man als Erstes die Status-LED blinken.

Den Code kann man mit den Crownhill-PicBasic-Compiler oder einem anderen Free-Compiler wie WinAVR schreiben. Im hier gezeigten Beispiel wird der Crownhill-PicBasic-Compiler verwendet. Funktionieren tut es natürlich auch mit einem anderen.

Blinkt die LED, dürfte soweit alles funktionieren und man kann sich daran wagen, den Roboter fahren zu lassen. Für die ersten „Gehversuche“ lässt man den Roboter einfach geradeaus fahren. Später kann man, wie der folgende Code zeigt, den Roboter ein Quadrat abfahren lassen. Hier sind dem Programmierer keine Grenzen gesetzt.

Der Code des Tischroboters in Proton+

```
Device = 16f84
Xtal = 4

Dim Wert As Word
Dim Wert2 As Word
Dim Flag As Byte
Symbol Motorport = Porta
Symbol Ledport = Portb
```

```
Symbol Led1 = Portb.0        'Alias PORTB to LEDS
Symbol Motorleitung1 = Porta.0
Symbol Motorleitung2 = Porta.1
Symbol Motorleitung3 = Porta.2
Symbol Motorleitung4 = Porta.3

Trisb = %00000000            'Alle Bits Ausgang
Portb = 0
Trisa = %11000000            'Untere 6 Bits Ausgang, Rest Eingang
Porta = 0
Portb = 0
Delayms 2000

Loop:
Toggle Led1

Gosub Vorwaerts
Delayms 2000
Toggle Led1

Gosub Anhalten
Delayms 2000
Toggle Led1

Gosub Neunzig_grad_rechts
Delayms 2000
Toggle Led1

Gosub Anhalten
Delayms 2000
Toggle Led1

Goto Loop

Vorwaerts:
 Motorleitung1 = 0
 Motorleitung2 = 1
 Motorleitung3 = 1
 Motorleitung4 = 0
Return

Rueckwaerts:
 Motorleitung1 = 1
 Motorleitung2 = 0
 Motorleitung3 = 0
 Motorleitung4 = 1
Return

Anhalten:
 Motorleitung1 = 0
 Motorleitung2 = 0
 Motorleitung3 = 0
```

```
 Motorleitung4 = 0
Return

Neunzig_grad_rechts:
 Motorleitung1 = 1
 Motorleitung2 = 0
 Motorleitung3 = 1
 Motorleitung4 = 0
 Delayms 1250

  Gosub Anhalten

Return

Goto Loop
```

Mit dieser Plattform kann nun jeder selbst experimentieren und z. B. Sensoren wie den IS471 (Infrarotsensor) anbringen, um den kleinen Bot-Objekten ausweichen zu lassen. Möglich wäre es auch, unten Sensoren anzubringen, damit er nicht vom Tisch fällt.

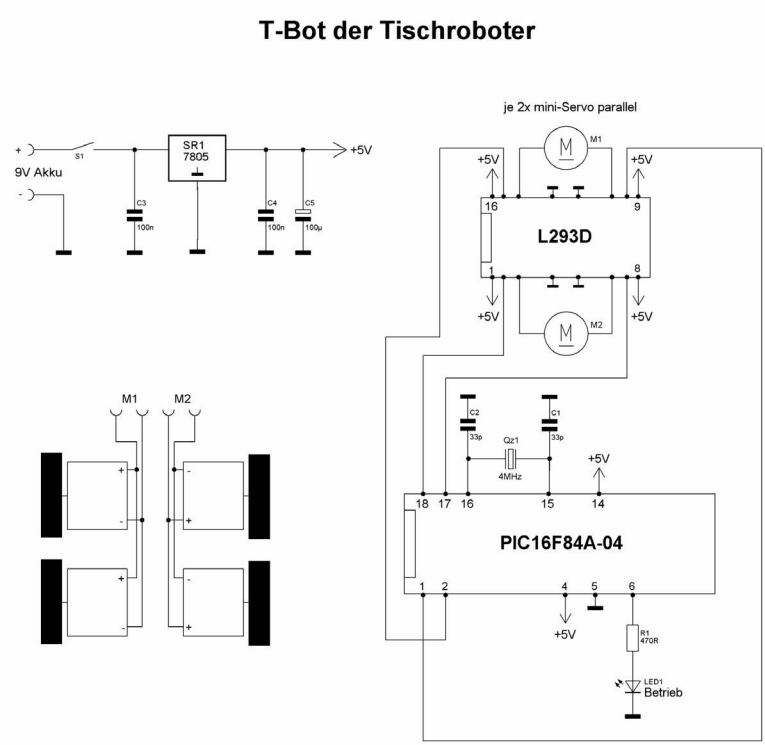


Abb. 10.38: Der Schaltplan des Tischroboters TR-1

Stichwortverzeichnis

Roboter selbst bauen

Das große Praxisbuch für Einsteiger und Fortgeschrittene

Roboter zu bauen wurde in den letzten Jahren immer populärer. Das liegt zum einen daran, dass Mikrocontroller und die zusätzliche Peripherie immer günstiger werden, aber auch an Schulen wird mehr in Richtung Mikrocontroller und Robotik unterrichtet.

Dieses speziell für Praktiker geschriebene Buch bietet zahlreiche detaillierte Bauanleitungen inklusive Quellcode und Schaltplänen für den Einsteiger sowie für den fortgeschrittenen Roboter-Entwickler. Die Beispielprogramme sowie Schaltpläne können für eigene Entwicklungen übernommen werden und dienen somit als Bausteine für eigene Ideen. Besonderen Wert wurde bei den Anleitungen auf Nachbausicherheit gelegt, und selbstverständlich wurden alle Projekte vom Autor selbst aufgebaut und ausführlich getestet. Durch die zahlreichen Beispiele ist es immer ein gutes Nachschlagewerk, das nicht so schnell im Bücherregal landen wird.

Aus dem Inhalt:
Planung und Bau eines Roboters • AVR und BASCOM • PC to BOT Interface über Funksensoren & Aktoren • Fahrregler • Kamerasysteme • Algorithmen • Selbstbauprojekte: GPS-Navigationssystem, Tischroboter, Robo-Control mit Mega32, Cybot-Pimp, Rasenmähroboter Grasshopper Phip, Induktionsschleifensystem, 12-V-Bleigellader, diverse I^2C-Erweiterungen für das Robo-Controlboard, der große Experimentierroboter THX-1

ISBN 978-3-7723-4109-0

EUR 49,95 [D]