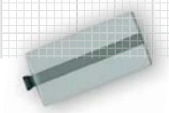
2 Die Programmierung des Arduino

Die als »Sketche« bezeichneten Programme für den Arduino werden in der Arduino-Entwicklungsumgebung (IDE) in einer C/C++-ähnlichen, auf Wiring basierenden Programmiersprache entwickelt. Wir führen kurz in die Besonderheiten dieser Programmierung ein, bevor wir mit der Konstruktion und der Steuerung der ersten Modelle beginnen.



2.1 Grundsätzliches

Die in Java programmierte und aus der am MIT entwickelten Programmierumgebung *Processing* hervorgegangene Entwicklungsumgebung (IDE) des Arduino wird als Teil des Open-Source-Projekts kontinuierlich weiterentwickelt und auf der Arduino-Webseite[1] für die Betriebssysteme Windows, Linux, Linux ARM und Mac OS X zum kostenlosen Download angeboten. Sie umfasst einen Programmeditor und einen auf dem Gnu-C-Compiler basierenden C/C++-Compiler.[2] In dieser IDE wird der Arduino in einer der Programmiersprache C/C++ sehr ähnlichen, aus *Wiring* abgeleiteten Sprache programmiert.[3]

Alle in diesem Buch vorgestellten Modelle können mit den abgedruckten und von der Webseite zum Buch herunterladbaren Sketchen gesteuert werden. Daher ist ein tieferes Verständnis der Programmiersprache für den Nachbau der Modelle nicht erforderlich.

Wer aber die Sketche erweitern oder variieren möchte, sollte einige grundlegende Dinge über die Programmierung des Arduino wissen. Daher möchten wir in diesem Kapitel auf einige Besonderheiten des Arduino etwas näher eingehen, die auf die Tatsache zurückzuführen sind, dass wir es beim Arduino mit einem *Embedded System* mit zahlreichen I/O-Schnittstellen, aber ohne Monitor und Tastatur zu tun haben.

Daraus ergeben sich ein paar spezielle Rahmenbedingungen, die bei der Programmierung des Arduino zu beachten sind. Dazu zählen insbesondere die folgenden fünf Punkte:

- Die Programme (*Sketche*) werden in der Arduino-Entwicklungsumgebung (IDE, Abb. 2–1) auf einem PC oder Laptop entwickelt. Mit einem Mausklick auf »Hochladen« werden sie in einem Schritt kompiliert und, wenn das fehlerfrei gelingt, über eine USB-Verbindung direkt in den Flash-Speicher des Arduino geladen und dort ausgeführt. Dabei werden zuvor auf den Arduino geladene Programme überschrieben. Im Arduino-Speicher ist daher immer genau ein Programm aktiv, das auch nach dem Ausschalten des Arduino erhalten bleibt.

---

1 Download der aktuellen Arduino-IDE: *https://www.arduino.cc/en/Main/Software*

2 Auf Details der Arduino-IDE (Installation, Konfiguration, Bedienung) gehen wir hier nicht weiter ein, denn für versierte Programmierer ist sie weitgehend selbsterklärend und für Einsteiger gibt es bereits hervorragende Bücher und Einführungen. Für einen schnellen Einstieg ist z. B. das »Arduino Special« des Make-Magazins sehr zu empfehlen [7].

3 Auch auf eine Einführung in C/C++ verzichten wir hier: Vielen Lesern wird C bekannt sein, und für alle anderen gibt es im Internet und natürlich auch in Buchform zahlreiche sehr gute Einführungen, von denen wir eine Auswahl auf der Webseite zu diesem Buch verlinkt haben. Speziell für die Programmierung von C auf dem Arduino sind als freie Quellen [9] und [10] zu empfehlen.

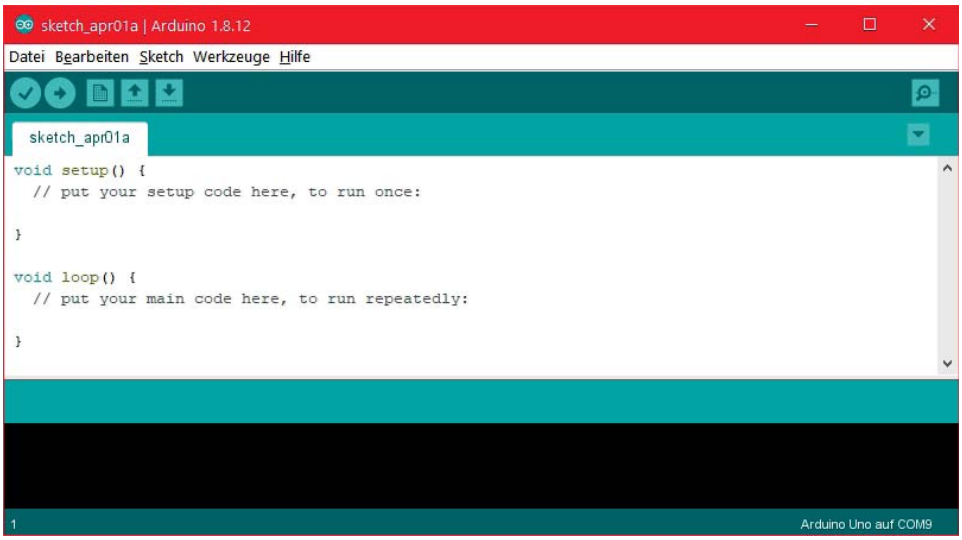
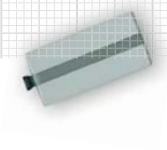


Abb. 2–1 Arduino-IDE mit Default-Sketch

- Ein etwa 0,5 kB großer *Bootloader* im Flash-Speicher des Arduino sorgt dafür, dass das in den Speicher geladene Programm unmittelbar nach der Aktivierung (Anschluss der Stromversorgung oder des USB-Kabels bzw. Abschluss des Uploads) oder nach dem Drücken des Reset-Tasters automatisch auf dem Arduino-Board ausgeführt wird. Will man ein sofortiges Starten (z. B. bei einem autonomen Roboter) verhindern, muss man Roboter und Programm um einen Starttaster erweitern.
- Ein Arduino-Programm kann jederzeit durch das Drücken des Reset-Tasters (rechts oben in Abb. 1–2) auf dem Arduino-Board zum Neustart veranlasst, aber nur durch Unterbrechung der Stromversorgung beendet werden.
- Der sehr begrenzte Hauptspeicher (beim Arduino Uno lediglich 31,5 kByte Flash-Speicher für das Programm und 2 kByte RAM Variablenspeicher) zwingt dazu, mit dem Speicherplatz zu »haushalten«. Dabei hilft der Compiler, der die Speicherbelegung minimiert. Die IDE zeigt nach dem Kompilieren eines Sketches den von Programm und Variablen belegten Speicherplatz an.
- Jedes Arduino-Programm besteht mindestens aus einer `setup()`-Routine, die einmalig zu Programmbeginn ausgeführt wird, und dem Hauptprogramm `loop()`, das anschließend so lange wiederholt wird, bis die Stromversorgung unterbrochen wird (Listing 2–1).



```
void setup() {
  // Initialisierungen (wird zu Programmbeginn einmal durchlaufen)
}

void loop() {
  // Hauptprogramm (wird ad infinitum wiederholt)
}
```

Listing 2–1 Genereller Aufbau eines Arduino-Sketches

Ein Arduino-Programm entspricht in Syntax, Programmstruktur, verwendeten Variablentypen und Operatoren im Wesentlichen der C/C++-Spezifikation, die wir hier als bekannt voraussetzen. Einige wichtige, Arduino-spezifische Eigenschaften und häufig genutzte Befehlsgruppen stellen wir in den folgenden Abschnitten vor.

2.2 Bibliotheken

Für den Arduino gibt es unzählige Codebibliotheken, die (wie in C-Compilern üblich) mit dem Compiler-Kommando `#include` eingebunden werden können und dann nach dem Kompilieren hinzugelinkt werden. Die IDE wird bereits mit einigen Standardbibliotheken ausgeliefert; andere müssen aus dem Internet heruntergeladen und über die Bibliotheksverwaltung als Zip-File in die IDE importiert werden.

```
#include <*.h> // Einbinden einer Bibliothek
```

Bei jedem Start prüft die IDE, ob neue Versionen der installierten Bibliotheken verfügbar sind; das Update kann dann per Mausklick aktiviert werden.

Rund 40 Funktionsbibliotheken finden sich auf Github im »offiziellen« Arduino-Bereich[4]. Die Suche nach »arduino« liefert allein auf Github insgesamt weitere etwa 22.000 Arduino-Code-Repositories. Zu praktisch jedem Sensor oder Aktor, der am Arduino genutzt werden kann, wird inzwischen vom Hersteller eine Arduino-Treiberbibliothek bereitgestellt, oft auf Github, manchmal auch zum Download auf der Webseite des jeweiligen Herstellers. An hilfreichem Programmcode herrscht also kein Mangel. Allein die Auswahl der richtigen – aktuellen, gepflegten und ausgetesteten – Bibliothek oder eines passenden Beispielprogramms für eine bestimmte Anwendung ist manchmal herausfordernd.

Mit der Auswahl unserer Modelle und ihrer Funktionen haben wir uns auf einige zentrale Bibliotheken festgelegt und dabei versucht, deren Anzahl und

4 Quelle für Arduino-Bibliotheken: *https://github.com/arduino-libraries*

Komplexität zu begrenzen. Die folgenden für die Anwendungen und Sketche in diesem Buch wichtigen grundlegenden Arduino-Bibliotheken erläutern wir in den nachfolgenden Abschnitten:

- `SPI.h`: Kommunikation über den SPI-Bus (*Serial Peripheral Interface*)
- `Wire.h`: Kommunikation über das *Two Wire Interface* (TWI, I²C)
- `Servo.h`: Steuerung eines Servomotors
- `Adafruit_MotorShield.h`: Ansteuerung von Gleichstrom- und Schrittmotoren

Außerdem verwenden wir für einzelne Modellvarianten vier weitere externe Bibliotheken, in deren Nutzung wir in den jeweiligen Kapiteln einführen:

- `ArduinoNunchuk.h`: (Fern-)Steuerung über einen Wii-Nunchuk
- `Pixy.h`: Live-Bildauswertung mit der Pixy-Kamera
- `SD.h`: Lesezugriff auf eine SD-Karte (via SD-Leser)
- `Mouse.h`: Maussteuerung des PCs über die USB-Schnittstelle

2.3 Serieller Monitor

Die IDE des Arduino kennt keine Breakpoints oder andere hilfreiche Funktionen zur komfortablen Fehlersuche (*Debugging*). Da der Arduino ohne ergänzende Hardware auch über keine Ausgabeeinheit wie einen Bildschirm verfügt, lassen sich Inputwerte oder Inhalte von Variablen nicht vom Arduino anzeigen.

Um dennoch eine Überwachung und sogar eine manuelle Steuerung des Programmablaufs zu ermöglichen, haben die Entwickler der Arduino-IDE ein Monitorfenster spendiert, an das der Arduino mit einem einfachen seriellen Protokoll über die USB-Verbindung und eine UART-Schnittstelle Daten übertragen kann (Abb. 2–2). Die Übertragungsgeschwindigkeit kann dabei von 300 bis 2.000.000 Baud gewählt werden. In diesem Buch verwenden wir einheitlich eine Baudrate von 115.200 Bit/s; so muss beim Laden eines anderen Sketches die Rate nicht jedes Mal am seriellen Monitor der IDE angepasst werden.

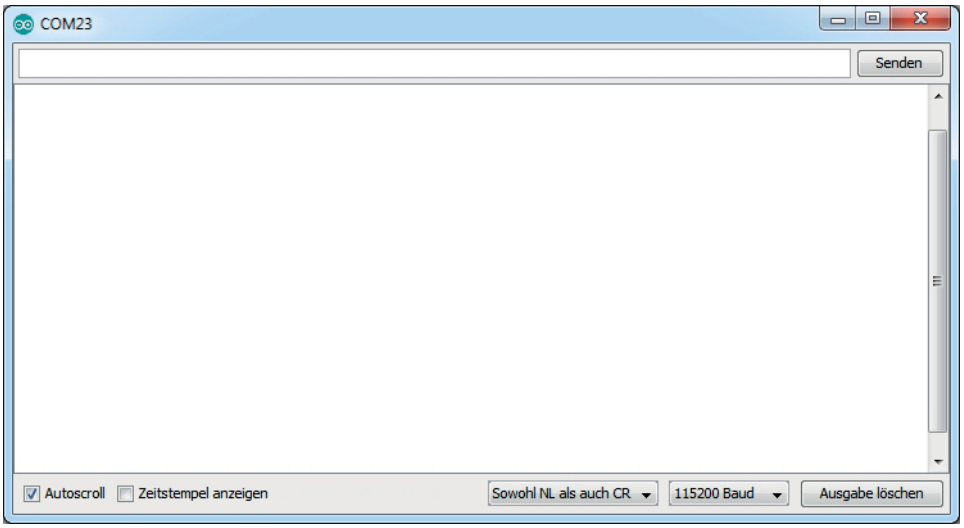


Abb. 2–2 Serieller Monitor der Arduino-IDE

Die empfangenen Daten werden im Monitorfenster der IDE angezeigt (öffnen über das Icon rechts oben in der IDE, Abb. 2–1). Auf demselben Weg können auch Tastatureingaben über das Eingabefenster des Monitors an den Arduino übermittelt werden.

Die Befehle zur Ansteuerung des seriellen Monitors sind integriert und müssen nicht über eine zusätzliche Bibliothek eingebunden werden. Der serielle Monitor muss lediglich mit Angabe der Übertragungsgeschwindigkeit der Daten (Baudrate in Bit/s) aktiviert werden:

```
void Serial.begin(long baud);
```

Dabei kann als Baudrate `baud` einer der folgenden Werte gewählt werden: 300, 1.200, 2.400, 4.800, 9.600, 19.200, 38.400, 57.600, 74.880, 115.200, 230.400, 250.000, 500.000, 1.000.000 oder 2.000.000.[5] Sie muss mit der im Fenster des seriellen Monitors unten rechts ausgewählten Baudrate übereinstimmen, damit die Zeichen korrekt empfangen und dargestellt werden. In den Sketchen in diesem Buch verwenden wir, wie erwähnt, einheitlich die Baudrate 115.200.

Das serielle Protokoll des Monitors arbeitet mit acht Datenbits, keinem Paritäts- und einem Stoppbit. Mit dem erweiterten Kommando

```
void Serial.begin(long baud, int config);
```

5 Grundsätzlich sind auch andere Baudraten möglich; für den seriellen Monitor können in der IDE jedoch nur diese ausgewählt werden.

können auch andere Protokollvarianten gewählt werden, wie z. B. sechs Datenbits, ein Bit für gerade Parität und zwei Stoppbits. Der serielle Monitor der IDE erwartet jedoch die Standardeinstellung.

Die Aktivierung des seriellen Monitors, durch die die Anschlüsse an den Pins D0 und D1 als Empfangs- (RX) und Sendeleitung (TX) belegt werden und daher im Sketch nicht für andere Zwecke genutzt werden dürfen, erfolgt üblicherweise in der `setup()`-Routine. Datenübertragungen werden von den LEDs RX und TX auf dem Arduino-Board angezeigt.

Will man den seriellen Monitor nur temporär aktivieren, lässt er sich mit der folgenden Funktion auch wieder deaktivieren; damit werden zugleich die Pins D0 und D1 für die Nutzung als digitale I/O-Pins freigegeben:

```
void Serial.end();
```

Ausgabe

Die Ausgabe von Daten (eines numerischen Wertes oder eines Textstrings) auf dem seriellen Monitor erfolgt mit folgenden Kommandos:

```
long Serial.print(string text);
long Serial.print(int var, byte type);
long Serial.print(float var, int len);
```

Dabei gibt `type` die Darstellung der Zahl an: `DEC`, `HEX`, `OCT` oder `BIN`. Bei Fließkommazahlen wird mit dem zweiten Parameter `len` die Anzahl der anzuzeigenden Nachkommastellen festgelegt; fehlt die Angabe, werden als voreingestellter Defaultwert zwei Nachkommastellen angezeigt. Die Funktion liefert die Anzahl der geschriebenen Zeichen zurück.

Der folgende Befehl schließt die Ausgabe außerdem mit einem Zeilenumbruch ab:

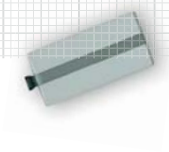
```
long Serial.println(string text);
```

Eingabe

Auch die Übergabe von Zeichen oder Daten an ein Arduino-Programm ist mit dem seriellen Monitor möglich. Dazu werden Zahlen oder eine Zeichenfolge in das Eingabefeld des Monitorfensters eingetragen und abschließend der »Senden«-Knopf (rechts oben) gedrückt.

Der Arduino kann eine Eingabe mit der folgenden Funktion feststellen:

```
int Serial.available();
```



Die Funktion gibt die Zahl der im 64 Byte großen Eingangspuffer vorliegenden Zeichen (`char`) zurück. Aus diesem Puffer können die Daten mit

```
int Serial.read();
```

jeweils einzeln als ASCII-Zeichen ausgelesen werden. Liegt kein Zeichen vor, liefert die Funktion den Wert -1 zurück. Ganzzahlige, durch Leerzeichen getrennte Werte und Fließkommazahlen können alternativ mit den folgenden Funktionen aus dem Puffer gelesen werden:

```
int Serial.parseInt();
float Serial.parseFloat();
```

Dabei wird der jeweils nächste Integerwert bzw. die nächste Fließkommazahl aus dem seriellen Puffer ausgewertet; andere Zeichen (Buchstaben, Leerzeichen) werden übersprungen.

Über einen speziellen Event-Handler lässt sich die Bearbeitung von Eingaben im seriellen Monitor sogar aus dem Hauptprogramm ausgliedern:

```
void serialEvent() {
  // Behandlung von Eingaben über den seriellen Monitor
}
```

Der Event-Handler wird automatisch aufgerufen, wenn eine Eingabe vorliegt; er ersetzt das kontinuierliche Abfragen von `Serial.available()`.

Debugging

Die Nutzung des seriellen Monitors vereinfacht in vielen Fällen die Fehlersuche: Variableninhalte und Sensorwerte können ausgegeben und durchlaufene Programmabschnitte angezeigt werden.

Allerdings kosten die Befehle Laufzeit und knappen Programmspeicher. Wird der serielle Monitor nur für die Fehlersuche benötigt, dann will man – insbesondere bei zeitkritischen Abläufen – die Monitorbefehle aus der endgültigen Version des Programms entfernen.

Das gelingt leicht, wenn man die Funktionsaufrufe mit Compiler-Bedingungen versieht – damit spart man sich später ein manuelles Auskommentieren oder Löschen der Monitorbefehle. Beim folgenden Codebeispiel erfolgt die jeweilige Ausgabe nur, wenn `DEBUG` definiert ist. Wird die einleitende Definition `#define DEBUG` auskommentiert, sind die Aufrufe des seriellen Monitors im kompilierten Programmcode nicht mehr enthalten.

```
#define DEBUG

#ifdef DEBUG
  #define Baud 115200
#endif

void setup() {
#ifdef DEBUG
  Serial.begin(Baud);
#endif
  // setup-Kommandos
}

void loop() {
#ifdef DEBUG
  Serial.println("Debugging");
#endif
  // Programmcode
}
```

Listing 2–2 Compiler-Bedingungen für Debugging-Kommandos

Die Sketche des Buches enthalten diese Codefolge immer dann, wenn der Sketch zeitkritische Programmzweige enthält, die durch die Kontrollausgaben auf dem seriellen Monitor spürbar verlangsamt werden.

2.4 I/O-Ports

Digitale Ports

Die 14 von 0-13 durchnummerierten I/O-Pins (oben in Abb. 1–1) sind digitale Ein- oder Ausgänge, die wir im Folgenden mit D0 bis D13 bezeichnen. Sie werden via Software als Eingang oder Ausgang konfiguriert. Die »Betriebsart« eines digitalen I/O-Pins wird – üblicherweise in der `setup()`-Routine – mit folgendem Befehl festgelegt:

```
void pinMode(byte pin, byte mode);
```

`pin` gibt die Nummer (0-13) des Digitalpins an. Als Betriebsart (`mode`) können `INPUT`, `OUTPUT` oder `INPUT_PULLUP` gewählt werden. `INPUT` ist beim Start des Arduino voreingestellt, daher kann man in der `setup()`-Routine eigentlich auf die Einstellung des `INPUT`-Modus verzichten. Meist macht man es dennoch, damit beim

Blick in den Sketch sofort klar ist, welche Pins zu welchen Zwecken verwendet werden.

Im `INPUT`-Modus liefert der Port entweder `HIGH` (hohes Potenzial, 5 V) oder `LOW` (niedriges Potenzial, 0 V):

```
int digitalRead(byte pin);
```

Wird ein digitaler Pin im `OUTPUT`-Modus verwendet, können mit dem folgenden Befehl ein `HIGH`- oder ein `LOW`-Signal (5 V/ 0 V) ausgegeben werden:

```
void digitalWrite(byte pin, byte value);
```

Im `INPUT_PULLUP`-Modus wird der Eingang intern über einen 20-kΩ-Widerstand mit 5 V verbunden, also auf 5 V »gezogen« – daher auch die Bezeichnung *Pull-up*-Widerstand. Ein angeschlossener Taster liefert dann im offenen Zustand ein definiertes `HIGH`-Signal – anderenfalls wäre der Wert undefiniert und Störströme können das anliegende Signal beeinflussen. Wird der mit GND verbundene Taster geschlossen, liegt ein definiertes `LOW`-Signal an.

Alternativ kann auch ein externer Pull-up-Widerstand verwendet werden. Durch die Nutzung des `INPUT_PULLUP`-Modus spart man sich jedoch in der Schaltung einen Widerstand und den 5-V-Anschluss. Da der Strom bei offenem Input über den Pull-up-Widerstand nach GND abfließt, hat unsere Schaltung unvermeidlich eine Verlustleistung. Damit diese möglichst gering ausfällt, wird ein großer Widerstand als Pull-up verwendet.

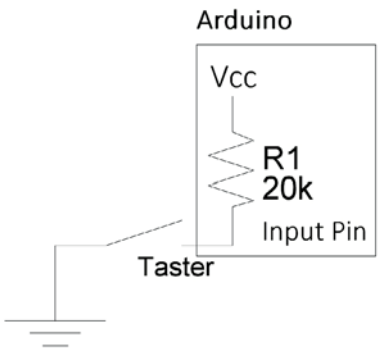


Abb. 2–3 *Interner 20-kΩ-Pull-up-Widerstand R1 für ein definiertes Signal bei offenem Input (z. B. einem geöffneten Taster)*

Die Verlustleistung des internen Pull-up-Widerstands des Arduino liegt damit bei:

$$P = U \cdot I = \frac{U^2}{R} = \frac{25V^2}{20.000\Omega} = 1{,}25\text{mW}$$

Die Funktion `digitalWrite()` ermöglicht außerdem, im `INPUT`-Modus den internen 20-kΩ-*Pull-up*-Widerstand ein- (`HIGH`) und auszuschalten (`LOW`). Das entspricht einer Umschaltung des Pin-Modus von `INPUT` auf `INPUT_PULLUP` (bzw. umgekehrt).

Einigen der digitalen Pins sind spezielle Funktionen zugeordnet. Werden diese Funktionen genutzt, stehen die entsprechenden Pins für andere Aufgaben nicht zur Verfügung:

- Die Pins D0 und D1 sind für die RX/TX-Signale zum seriellen Monitor zuständig und belegt, wenn der serielle Monitor aktiviert ist, um Ausgaben auf ihm oder Eingaben über ihn vorzunehmen (siehe Abschnitt 2.3).

- Die Pins D2 und D3 können mit externen Interrupts belegt werden (siehe Abschnitt 2.5).
- Servomotoren benötigen die PWM-Pins D10 (Servo1) bzw. D9 (Servo2).
- Die Pins D10-D13 sind mit den Signalleitungen des seriellen SPI-Protokolls belegt: SS, MOSI, MISO und SCK (siehe Abschnitt 2.7).
- Schließlich ist Pin D13 direkt mit einer On-Board-LED verbunden, die über das Anlegen eines `HIGH`- oder `LOW`-Signals angesteuert werden kann.

Pin	Funktion	PWM
D0	RX	
D1	TX	
D2	INT0	
D3	INT1	~ (Timer2)
D4		
D5		~ (Timer0)
D6		~ (Timer0)
D7		
D8		
D9	Servo2	~ (Timer1)
D10	Servo1, SS (SPI)	~ (Timer1)
D11	MOSI (SPI)	~ (Timer2)
D12	MISO (SPI)	
D13	SCK (SPI)	

Tab. 2–1 Spezielle Funktionen der digitalen Pins D0 bis D13 (Arduino Uno)

Analoger Output

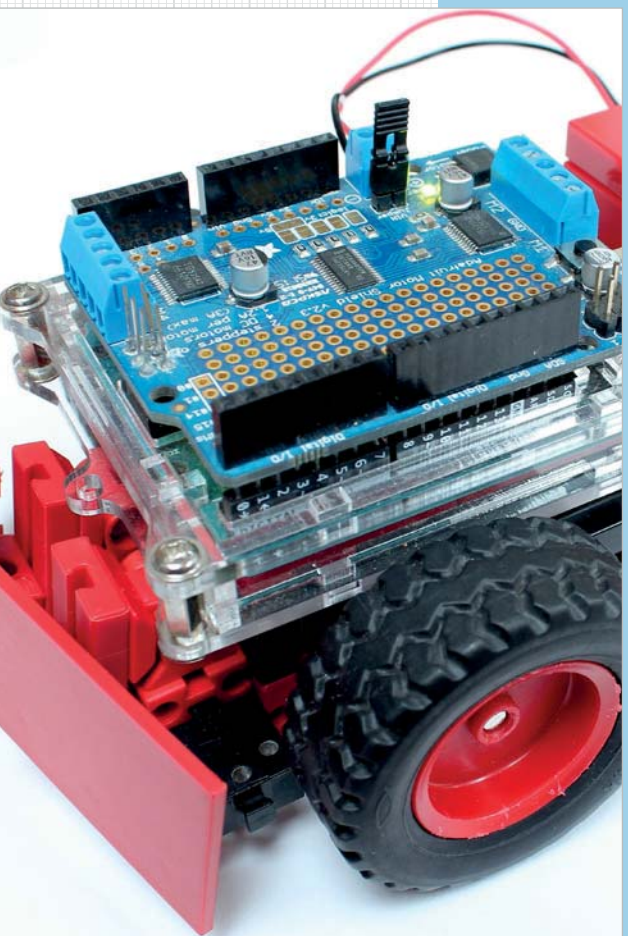
Der ATmega328P verfügt über keinen D/A-Wandler. Dennoch kann über die Digitalpins mit dem folgenden Befehl auch ein analoger 8-Bit-Wert (0-255) ausgegeben werden:

```
void analogWrite(byte pin, int value);
```

Dabei wird der Integerwert in ein *Pulse-Width-Modulation*-Signal (Pulsweitenmodulation, PWM) umgewandelt: Der Ausgang sendet ein »gepulstes« Signal

3 Der Buggy

Eine wesentliche Eigenschaft autonomer Roboter ist deren selbstständige Orientierung mithilfe von Sensoren und die daraus abgeleitete autonome Steuerung – ein Thema, bei dem fischertechnik schon vor fast 40 Jahren die Nase vorn hatte, wie die Geschichte des Buggys beweist. In diesem Kapitel entwickeln wir einen eigenständig fahrenden, Arduino-gesteuerten fischertechnik-Roboter, den wir – als kleine Reminiszenz an den ersten autonomen fischertechnik-Roboter aus dem Jahr 1981 – liebevoll »Buggy« taufen.



3.1 Die Geschichte des Buggys

Das Baukastensystem von fischertechnik war schon immer seiner Zeit voraus – aber bei keinem Thema wird das deutlicher als bei der Entwicklung einfacher autonomer Roboter: Der erste fischertechnik-Buggy wurde im Jahr 1981 – dem »Geburtsjahr« des IBM PC und des Commodore VC 20 – von Mike Bostock und Max Townsend für das britische *Microelectronics Education Program* in Zusammenarbeit mit einem Team der BBC entworfen (Abb. 3–1). Die Firma Economatics aus Sheffield vertrieb ihn ab 1983 für den stolzen Listenpreis von 164,35 £. Der Buggy verfügte über eine Steuerungsplatine, zwei Schrittmotoren, eine Fotozelle, zwei Mini-Taster und einen Barcode-Reader (IR-Sensor mit IR-Diode). Später wurde das Grundmodell des Buggys von Economatics um ein »Pen Kit« erweitert: einen per Elektromagnet absenkbaren Stift, der in der Mitte des Buggys montiert wurde und ihn zu einem »mobilen Plotter« oder »Mal-Roboter« machte.

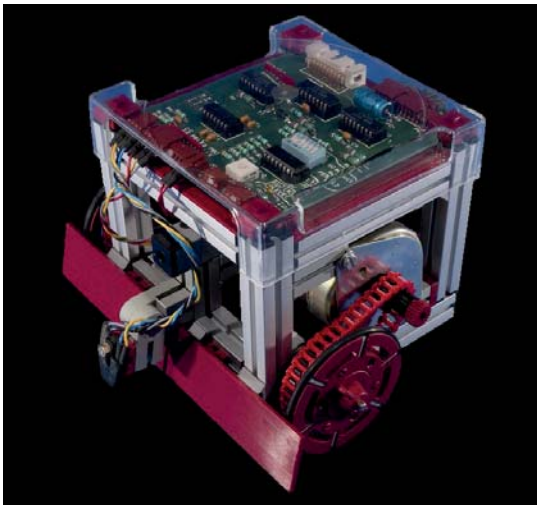


Abb. 3–1 »BBC Buggy« von Economatics (1983)

Die Ansteuerung des Buggys erfolgte über den Parallelport mit acht analogen Datenleitungen, über die die Byte-Sensorwerte an den Hostcomputer übertragen wurden. Der Buggy konnte einer Spur folgen und über die beiden Mini-Taster (»Bumper«) Hindernisse oder Wände erkennen; er konnte sich Sackgassen in einem Labyrinth merken und auf dem Rückweg den kürzesten Weg zum Ausgangspunkt wählen, Barcodes mit Kommandos lesen und Linien zeichnen. Im Januar 1984 reichte Max Townsend die Konstruktion des fischertechnik-Buggys in den USA als Patent ein, das im Oktober 1985 veröffentlicht wurde (Abb. 3–2).

Besonders an britischen Schulen war der kleine Roboter beliebt, da er sich mit dem Homecomputer *BBC Micro* (mit einem 2-MHz-6502-Mikroprozessor) in BASIC programmieren und steuern ließ. Dieser Homecomputer wurde von Acorn im Auftrag der BBC für die Serie »*The Computer Programme*« entwickelt und 1981 in den Schulen eingeführt. Am 28. Februar 1983 hatte der »BBC Buggy« einen Fernsehauftritt in der achten Folge der Serie »*Making the most of the Micro*« (*Everything under control*) [2], und die Zeitschrift »Your Computer« widmete dem Buggy in der Aprilausgabe 1983 die Titelseite und einen Leitartikel (Abb. 3–3) [3].

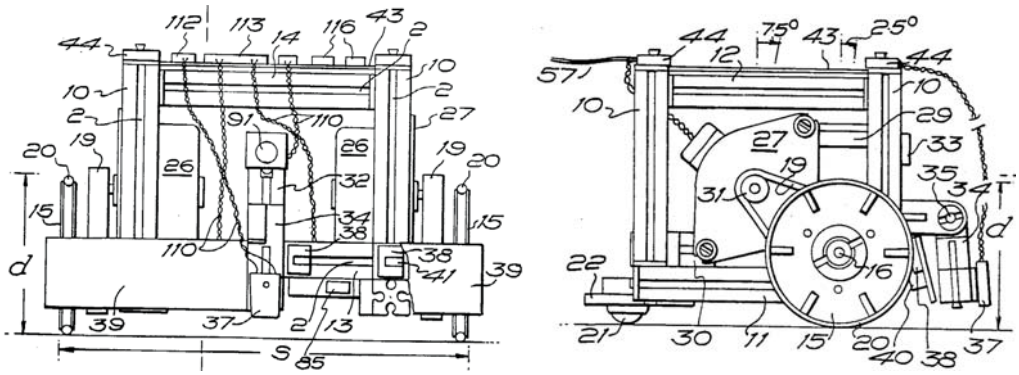


Abb. 3–2 Auszug aus US-Patent No. 4.548.584 (1985) [1]

Abb. 3–3 Titelseite und Leitartikel der Zeitschrift »Your Computer«, Ausgabe 4/1983 [3]

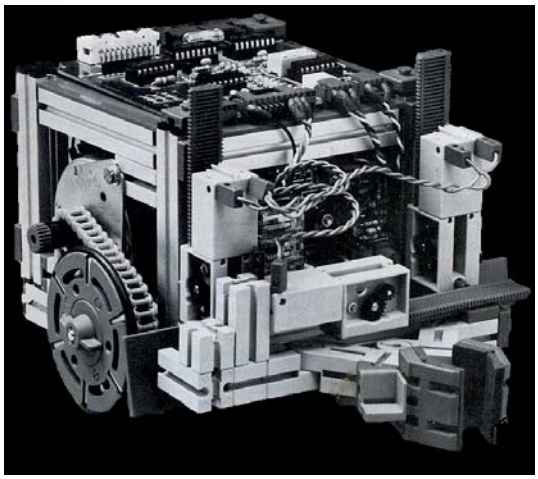


Abb. 3–4 BBC Buggy mit Greifer

Über 5.000 Exemplare des »Ur-Buggys« verkaufte die Firma Economatics allein in Großbritannien. Später gab es auch eine Version, die über das fischertechnik-Interface von anderen Homecomputern wie dem Commodore C64 oder dem Schneider CPC gesteuert werden konnte.

Als Erweiterung bot Economatics für 79 £ ein Ergänzungsset an, das aus einem Greifer bestand, der über drei Mini-Mots mit Hubgetrieben und einer separaten Elektronik steuerbar war. Damit konnte der Buggy gefundene Gegenstände ergreifen und anheben (Abb. 3–4). Ein Strombegrenzer erkannte, wenn der Greifer einen Gegenstand erfasst hatte, und stoppte automatisch den für den Greifer zuständigen Mini-Mot [4].

Heute kann man den BBC Buggy noch im *National Museum of Computing* in Bletchley Park in Funktion bewundern.

Eine weiterentwickelte Version des Buggys, der PIC-Buggy – etwas schlanker, mit S-Motoren und einfachen »Bumpern« –, wurde bis zur Insolvenz der Firma Economatics im Jahr 2008 vertrieben (Abb. 3–5). Anders als der Ur-Buggy funktionierte der PIC-Buggy auch »offline«, d. h. ohne Verbindungskabel zum PC: Die Programmierung der Steuerungsplatine erfolgte über einen EPROM-Brenner [5]. Die Programme wurden dafür in einer der fischertechnik-Einsteigersprache ROBO Pro ähnlichen Programmierumgebung, dem PIC-Logicator, als grafisches Flussdiagramm entwickelt.

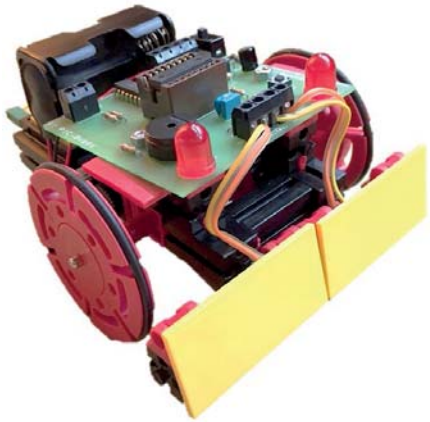
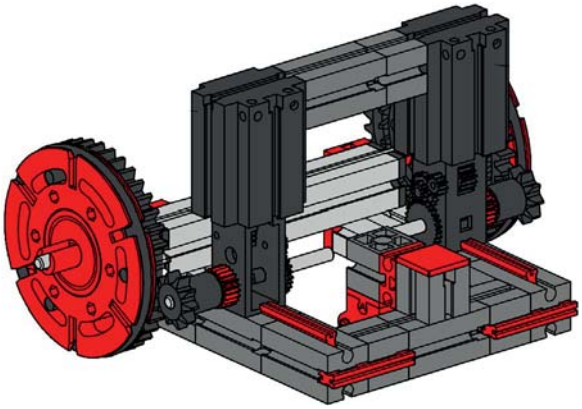


Abb. 3–5 Economatics PIC-Buggy (ca. 2003, Foto: Dirk Uffmann)

Ein diesem PIC-Buggy sehr ähnliches Robotermodell findet sich unter der Bezeichnung »Turtle« (Schildkröte) in vielen fischertechnik-Computing-Kästen, vom Baukasten »Computing Experimental« aus dem Jahr 1987 (mit einer 20:1-Untersetzung, Abb. 3–6) über das »Experimentierbuch Profi Computing« von 1991 [6], das Cornelsen-Schulprogramm »Experimenta Computing« (1994, Abb. 3–7), die Turtle-Bauanleitung aus dem Jahr 1996 mit Impulszahnrädern (100 Impulse je Radumdrehung, Abb. 3–8) bis zum aktuellen Baukasten »Mini Bots«, der im Jahr 2015 erschien (Abb. 3–9).



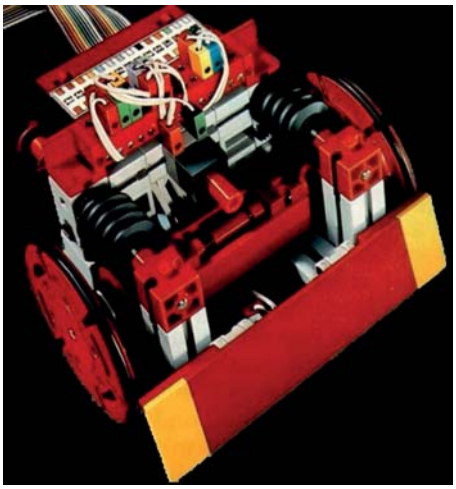


Abb. 3–6, 3–7: Modell »Turtle« aus den Baukästen »Experimenta Computing« (1994), links und »Computing Experimental« (1987), rechts [7]

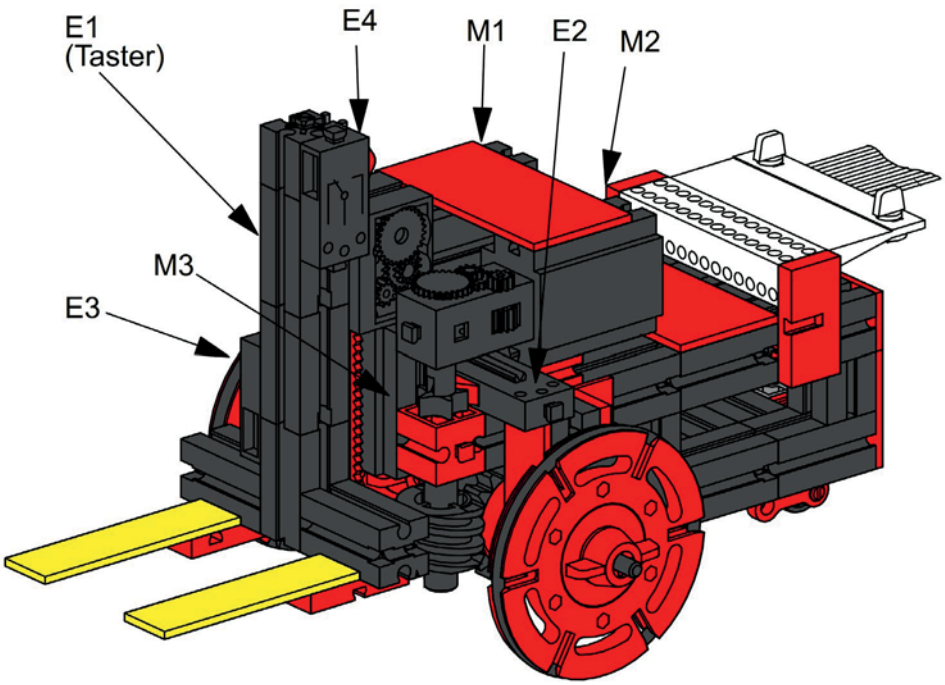


Abb. 3–8 Modell »Turtle« aus einer Bauanleitung des Jahres 1996

Abb. 3–9 Mini Bots (2015, Foto: fischertechnik GmbH)

Das im Folgenden vorgestellte Buggy-Modell, das uns durch dieses Kapitel begleiten wird, greift einige Konstruktionselemente dieser historischen Vorgänger auf. Im Verlauf des Kapitels werden wir unseren Buggy mit verschiedenen Sensoren ausstatten und ihm damit einige autonome Fähigkeiten verleihen, die zum Experimentieren und Weiterentwickeln einladen sollen. Mit dem Arduino wird er so zu einem autonom fahrenden Roboter mit ein wenig »Eigenintelligenz«.

3.2 Das Buggy-Basismodell

Mechanischer Aufbau

Unser kleiner Buggy basiert auf einer ähnlichen Grundkonstruktion wie die zahlreichen historischen Buggys: Zwei Antriebsmotoren und ein Stützrad sorgen für eine hohe Wendigkeit, und zwei »Bumper« an der Frontseite ermöglichen ihm, Hindernisse zu erkennen. Damit er wenig Eigengewicht transportieren muss und leicht nachgebaut werden kann, verwenden wir möglichst wenige Bauteile.

Als Antriebsmotoren nutzen wir – wie beim PIC-Buggy – zwei fischertechnik-S-Motoren (32293), auch als »Minimotor« bezeichnet, mit U-Getriebe (31078) und einer Untersetzung von ca. 56:1. Sie haben mit 0,4 Ncm ein deutlich größeres Drehmoment als die z. B. in den Mini Bots eingesetzten XS-Motoren (0,15 Ncm). Wir verbauen sie so, dass sie zugleich tragende Elemente unserer Fahrzeugkon-

struktion werden: Über die seitlichen Nuten verbinden wir die beiden Motoren mit zwei Federnocken stabil miteinander. In die beiden U-Getriebe setzen wir je eine U-Achse 40 mit Zahnrad Z28 (31064) als Antriebsachse ein (Abb. 3–10).

Abb. 3–10 S-Motoren mit U-Getriebe und U-Achse 40 als Buggy-Chassis

Die beiden vorderen Stoßdämpfer konstruieren wir aus zwei liegend montierten und über einen BS 7,5 verbundenen Mini-Tastern (37783), auf denen je ein BS 7,5, ein Gelenkwürfel (31426/31436) und darauf ein BS 15×30×5 mit Nut und Zapfen sitzen, die die Taster auslösen. In die Nuten der BS 15×30×5 werden je ein Winkelstein 15° geschoben und daran je eine Bauplatte 30×45 mit dem äußersten Zapfen befestigt (Abb. 3–11).

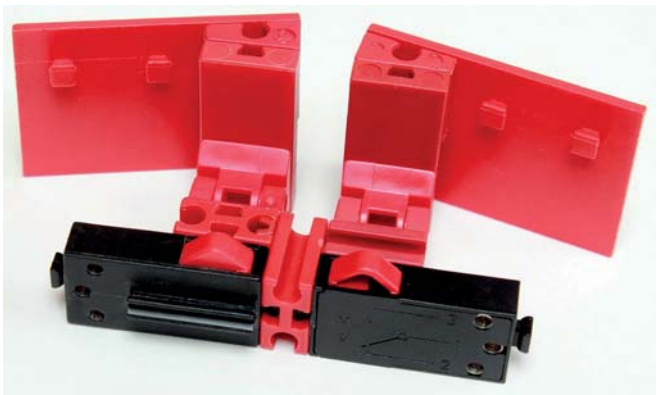


Abb. 3–11 Bumper aus zwei Mini-Tastern mit »hochgeklappten« Bauplatten

Die Bumper-Einheit verbinden wir stabil über das in Abb. 3–12 gezeigte Verbindungselement mit der Antriebseinheit. Zuletzt schieben wir in die oberen Nuten der beiden BS 15 je einen BS 5 und »verriegeln« damit die Verbindung mit den U-Getrieben (siehe Abb. 3–13 und 3–14).

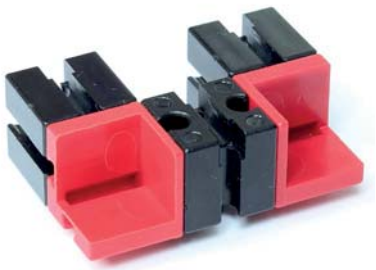


Abb. 3–12 Verbindungselement (zwischen Bumper und Antriebseinheit)

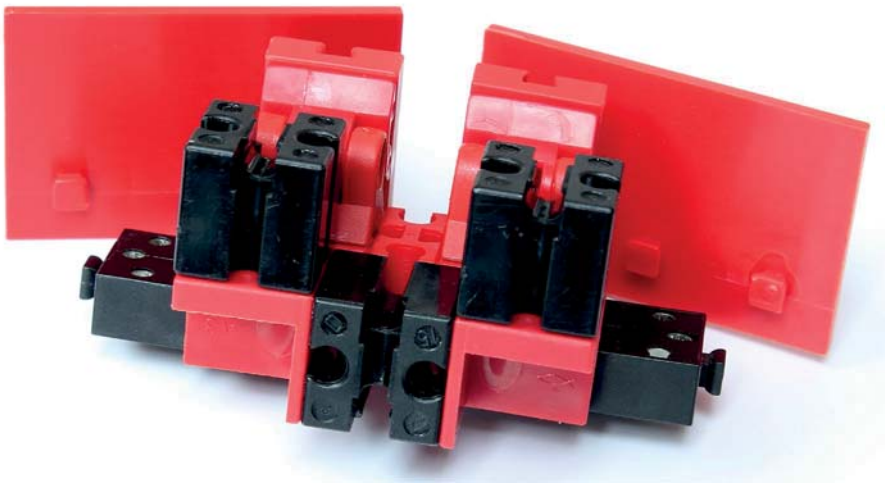


Abb. 3–13 Bumper-Einheit mit Verbindungselement

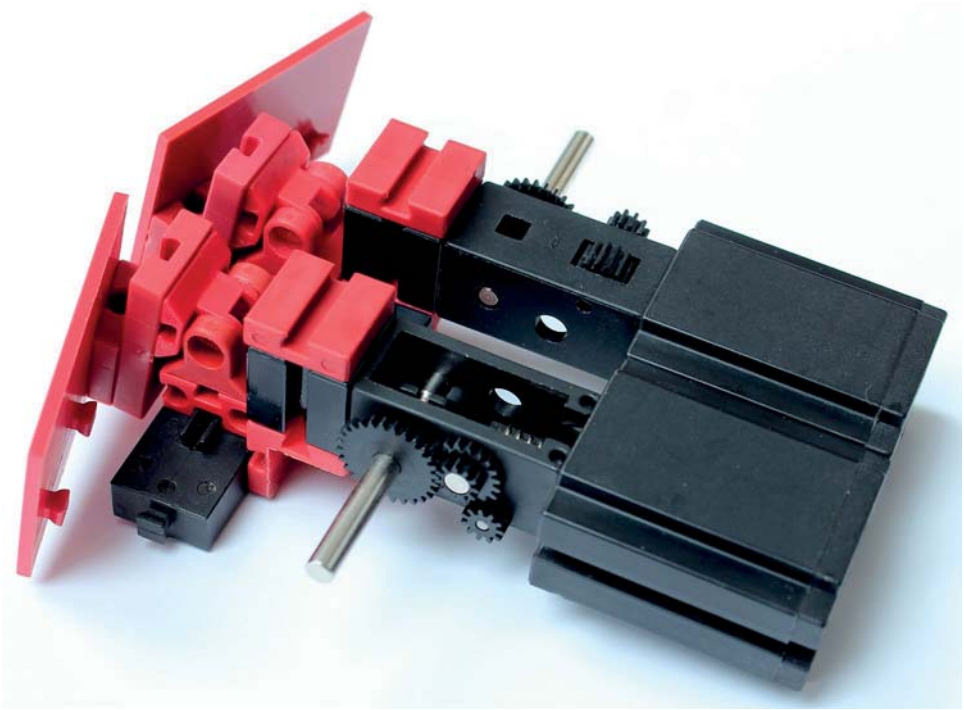


Abb. 3–14 Gesamtansicht der Antriebseinheit mit Bumpern

Abb. 3–15 Bereifung – Traktor- (links) oder einfache Reifen 50 (rechts)

Auf die U-Achsen 40 montieren wir nun die Räder des Buggys. Dafür eignen sich die Felgen 30 (32883), auf die entweder die Reifen 50 (32913) oder die Traktorreifen 50 (106767) bzw. 60 (121661) aufgezogen werden (Abb. 3–15).

Räder mit größerem Durchmesser haben den Charme, dass sie leichter kleinere Hindernisse überfahren können. Wohl auch deshalb wurden in den früheren Buggy- und Turtle-Modellen von fischertechnik Drehscheiben 60 verwendet, bei denen zur Verbesserung der Haftung in die Nuten zwei O-Ringe (Dichtungsringe) 54 × 3 als Bereifung eingesetzt wurden. O-Ringe 54 × 3 (35678) sind noch als fischertechnik-Einzelteil erhältlich, finden sich aber nur in wenigen (ausgelaufenen) Baukästen – daher haben wir unseren Buggy mit einer normalen Fahrzeugbereifung ausgestattet.

Hinten, am Ende der Antriebseinheit, fehlt uns noch ein »Stützrad«, das es dem Buggy erlaubt, wie seine Vorfahren auf der Stelle zu drehen. Dafür gibt es zahlreiche Konstruktionsmöglichkeiten: Der »Klassiker« (Abb. 3–16, oben rechts), der sich auch in vielen fischertechnik-Bauanleitungen findet, ist ein Rad 14 (36573) auf einer Rastachse 20 (31690) in einem Rollenbock (32085). Eine hübsche Variante ist es, das Rad 14 zwischen zwei Lenkhebeln (38473) einzuspannen (Abb. 3–16, oben links).

Wer über einen O-Ring 17 × 4 verfügt, kann damit eine Seilrolle d=21 (35797) bereifen und die Rastachse 20 in zwei Kupplungsstücken (38253) lagern (Abb. 3–16, unten) – sicher die optisch ansprechendste dieser drei Konstruktionsvarianten. Allerdings sollte man bei dieser Variante die beiden Vorderräder mit 60er-Reifen ausstatten, sonst steht der Buggy hinten ein wenig hoch.

Abb. 3–16 Varianten für die Konstruktion des Stützrads

Abb. 3–17 Verbindungselement für Stützrad

Das Stützrad wird mit der Aufnahmeachse (31124) in einen Baustein 15 mit Ansenkung (32321) eingesetzt. Den BS 15 mit Ansenkung stecken wir in die seitliche Nut eines BS 30, der mit einem Verbindungsstück 30 (31061) quer hinter den Motoren angebracht wird (Abb. 3–17).

Richtig montiert dreht sich das Stützrad nun frei unter dem Buggy-Chassis. In die obere Nut des BS 30 schieben wir nun noch den Zapfen eines Bausteins 15 × 30 × 5 mit Nut und Zapfen (35049), sodass dieser über dem BS 15 mit Ansenkung zu liegen kommt (Abb. 3–18). Dessen Nut dient uns als Halterung für einen fischertechnik-Akku. Alternativ können wir daneben einen zweiten Baustein 15 × 30 × 5 mit Nut und Zapfen aufschieben und darauf einen Batteriekasten

(Abb. 3–20). Schließlich befestigen wir an der seitlichen Nut des linken Motors unseres Buggys noch einen Mini-Taster, mit dem wir später das Programm des Buggys starten und z. B. Display-Anzeigen umschalten können (Abb. 3–18).

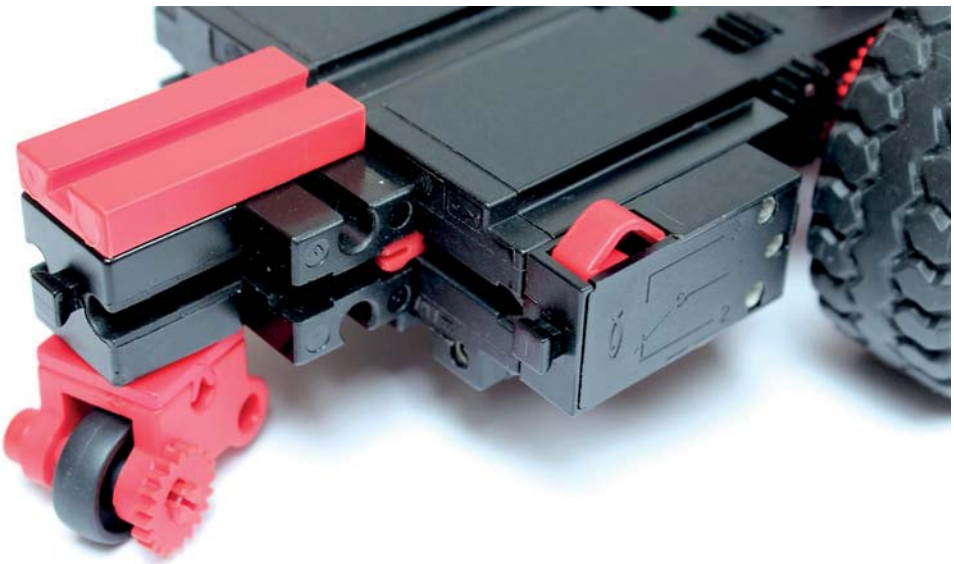


Abb. 3–18 Buggy-Chassis mit Stützrad, Starttaster und Akku-Halterung

Eine fischertechnik-Designer-Datei des Modells findet sich zum Download auf der Website zum Buch. Ausgehend von dieser Grundkonstruktion werden wir im Folgenden für die Ausstattung des Buggys mit dem einen oder anderen Sensor kleine An- oder Umbauten an unserem Buggy vornehmen. Und natürlich sind auch Konstruktionsvarianten zulässig – sie sind sogar ausdrücklich erwünscht.

Akku, Controller und Motorsteuerung

Jetzt müssen wir unseren Buggy noch mit einem Controller und einer Stromversorgung ausstatten. Bevor wir unseren Arduino über den beiden Motoren befestigen, schieben wir in die senkrechte Nut des mittleren BS 15 der Bumper-Halterung eine grüne oder rote em-Verteilerplatte (31327/31328), um darüber später die (knappen) GND-Anschlüsse des Arduino zu verteilen (Abb. 3–20). Alternativ zur em-Verteilerplatte gibt es für alle Pins des Arduino auf dem *Adafruit Motor Shield* je zwei Lötkontakte – wenn wir in die bisher ungenutzten, innen liegenden Kontakte Pinheader löten, können wir weitere GND-Pins dazugewinnen (Abb. 3–19).

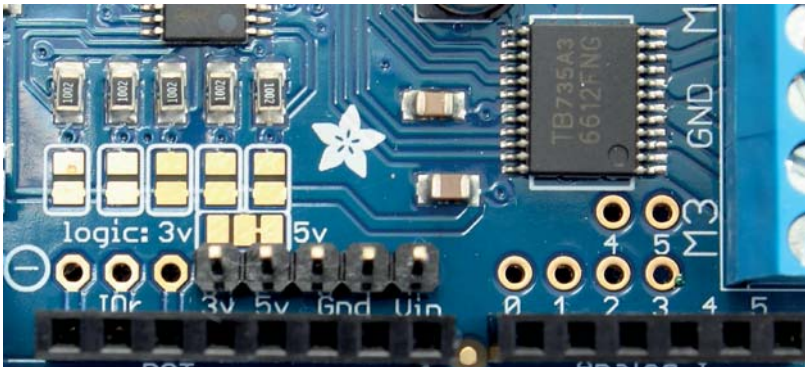


Abb. 3–19 Zusätzliche Pinheader für 3,3 V, 5 V, 2xGND und Vin (9 V)

Als Stromversorgung ergänzen wir einen Batteriekasten (32263) oder einen fischertechnik-Akku, den wir in die Halterung über dem Stützrad einsetzen. Der fischertechnik-Akku hält mit 1500 mAh deutlich länger durch als eine 9-V-Blockbatterie oder ein 9-V-Blockakku – aber mit ihm steigt auch das Gewicht unseres Buggys, denn der Akku wiegt knapp 210 g und verschiebt den Schwerpunkt über das Stützrad, das sich dadurch deutlich schwergängiger dreht. Auch hat er einen sichtlich größeren Platzbedarf (Abb. 3–20 und Abb. 3–21).

Abb. 3–20 Buggy mit Batteriekasten

Abb. 3–21 Buggy mit fischertechnik-Akku

Zur Montage des Arduino schieben wir die beiden äußeren Zapfen einer der beiden am Arduino-Gehäuse angebrachten Bauplatten 30×45 (Abb. 1–12) in die Nuten der beiden BS 5, die auf den BS 15 des Verbindungsstücks zwischen Bumpern und Motorblock aufgesteckt sind. Dabei achten wir darauf, dass der Strom- und der USB-Anschluss des Arduino nach hinten zeigen, damit die Anschlussklemmen für die Motoren später ebenfalls hinten liegen. Abb. 3–22 zeigt die Gesamtansicht unseres kleinen Buggys inklusive aufgestecktem *Motor Shield*.

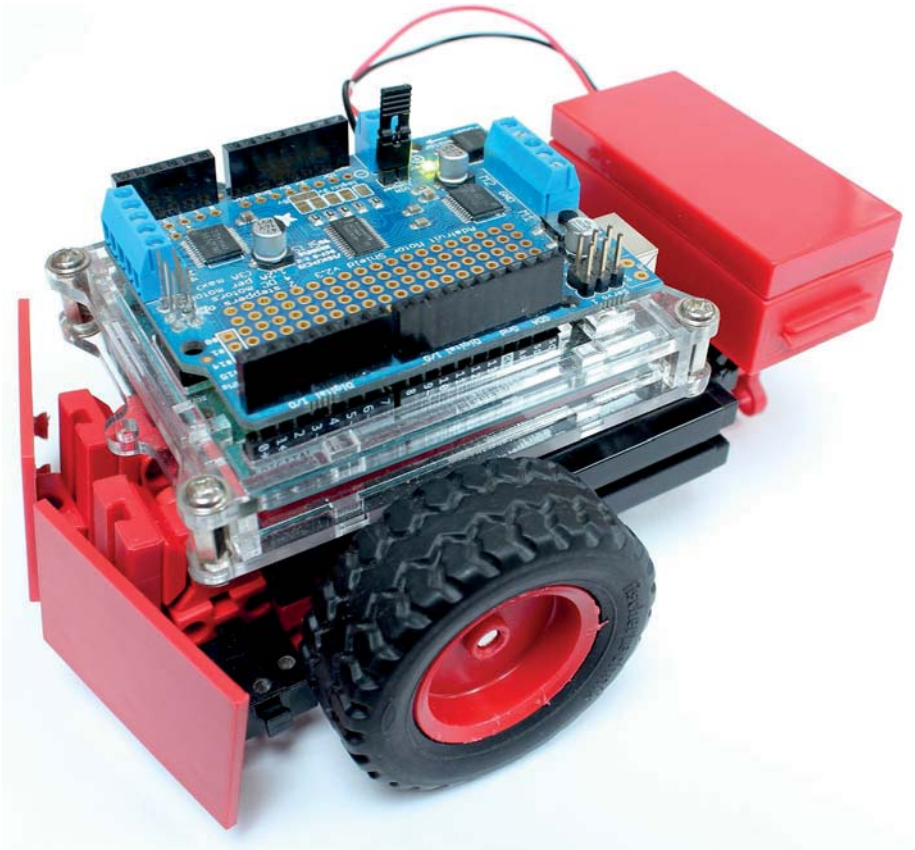


Abb. 3–22 Buggy-Basismodell mit Arduino Uno, Motor Shield und Batteriekasten

Jetzt müssen wir noch die Motoren, den Starttaster und die Stromversorgung an unseren Arduino anschließen. Der Anschluss der beiden S-Motoren ist einfach: Die beiden doppeladrigen Kabel der Motoren werden jeweils mit den mit »M1« (linker Motor) und »M2« (rechter Motor) markierten blauen Schraubklemmen des Motor Shield verbunden. Die korrekte Polung der Anschlüsse überprüfen wir später.

Das Kabel zur Verbindung des Arduino mit der Stromversorgung (fischertechnik-Akku bzw. Batteriekasten) gehört an die mit »(+)« (rotes Kabel) und »(–)« (grünes Kabel) gekennzeichneten Anschlüsse (gelber Kreis in Abb. 3–23). Damit das Motor Shield die Motoren über den externen 9-V-Anschluss versorgt, muss der »Vin-Jumper« (roter Kreis in Abb. 3–23) aufgesteckt werden.

Zur Nutzung des Mini-Tasters verbinden wir dessen mittleren Anschluss mit Pin A0, den offenen Ausgang mit einem GND-Pin. Die komplette Beschaltung zeigt Abb. 3–23.

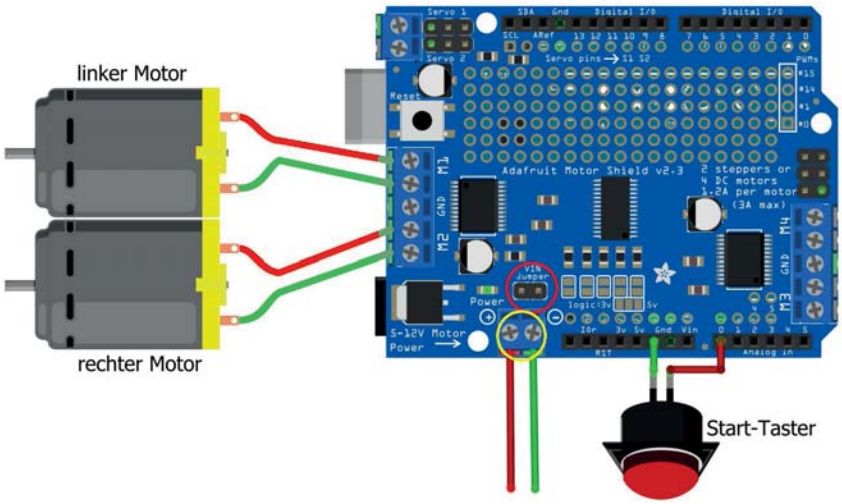


Abb. 3–23 Anschluss von S-Motoren, Starttaster und Stromversorgung (erstellt mit fritzing)

Damit uns der Buggy bei Softwaretests oder der Fehlersuche nicht davonfährt, können wir ihn – wie in einer Werkstatt – mit der in Abb. 3–24 gezeigten Konstruktion »aufbocken«: auf einer roten oder schwarzen V-Grundplatte 45×45 (36593/36596) mit zwei BS 15 (zwei Zapfen). Wer diese kleine Grundplatte nicht in seinem Sortiment hat, kann stattdessen auch eine schwarze Grundplatte 120×60 (35129) verwenden. Dazu schieben wir die Nuten der Motoren auf die beiden freien Zapfen der BS 15 – und schon hängen die Räder des Buggys frei in der Luft.

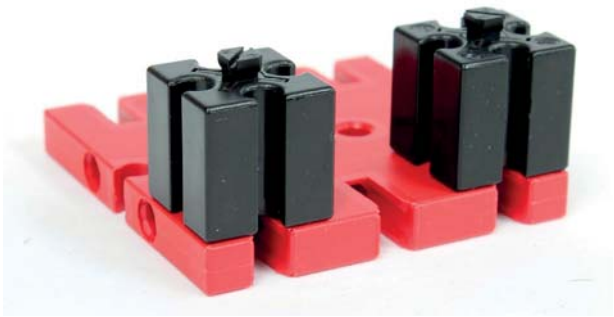
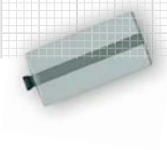


Abb. 3–24 Grundplatte zum »Aufbocken« des Buggys



3.3 Buggy-Steuerung

Die Steuerungsprogramme für unseren Buggy können wir nun auf unserem kleinen »Teststand« mit USB-Verbindung zum PC und einem Netzteil als Stromversorgung testen, bevor wir den Buggy autonom fahren lassen. Für einige der Steuerungen werden wir unseren Buggy ein wenig umkonstruieren, um zusätzliche Sensoren zu befestigen. Soweit verfügbar haben wir dabei fischertechnik-Sensoren verwendet, da sie sich einfacher und eleganter verbauen lassen und sie bei vielen Lesern bereits vorhanden sein dürften. Sie lassen sich aber oft durch ähnliche Sensoren aus dem Modellbau oder der »Maker-Szene« ersetzen. Bezugsquellen für solche alternativen Sensoren haben wir auf der Webseite zum Buch zusammengetragen.

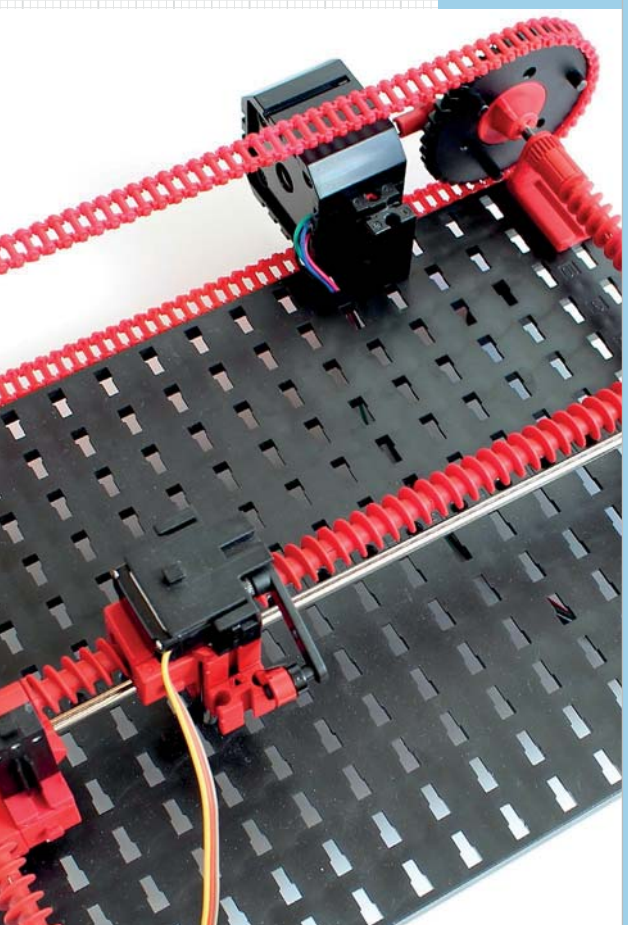
Tanzender Buggy

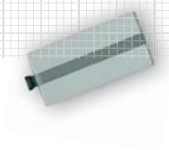
Zunächst möchten wir unseren Buggy einem ersten Fahrtest unterziehen. Der folgende kleine Sketch, mit dem wir den Buggy ein wenig »tanzen« lassen, zeigt, wie die Ansteuerung der Motoren funktioniert. Dabei können wir gleich testen, ob beide Motoren mit der richtigen Polarität angeschlossen sind: Wenn alles stimmt, fährt der Buggy nach dem Drücken des Starttasters in ständiger Wiederholung erst kurz vorwärts, dann rückwärts, dreht dann nach rechts, fährt wiederum vorwärts und rückwärts, dreht dann nach links, fährt vorwärts und rückwärts und sollte nach einer erneuten Rechtsdrehung wieder mehr oder weniger in der ursprünglichen Position stehen.

```
/*
 * "Tanzender Buggy"
 */

 #include <Adafruit_MotorShield.h>

 #define Start A0 // Starttaster
 #define Speed 200 // Motorgeschwindigkeit
 #define Move 1000 // Dauer der Bewegung in ms
 #define Turn 200 // Dauer der Drehbewegung in ms

 Adafruit_MotorShield MShield = Adafruit_MotorShield(0x60);
 Adafruit_DCMotor *MotorL = MShield.getMotor(1); // linker Antriebsmotor
 Adafruit_DCMotor *MotorR = MShield.getMotor(2); // rechter Antriebsmotor

 void setup() { // Festlegung der Motorgeschwindigkeit
```

5 Der Plotter

Ist ein Plotter ein Anachronismus? Gelingen doch mit Laserdruckern Ausdrucke, die zumindest für das bloße Auge hochpräzise wirken. Tatsächlich übertreffen Plotter aber auch heute noch moderne Drucker in Präzision und Auflösung. Häufig werden CAD-Zeichnungen daher mit großen Plottern gedruckt. Auch ist die Plottertechnik Grundlage zahlreicher weiterer, ausgesprochen moderner Geräte: So arbeiten Lasercutter mit denselben Algorithmen und Mechanismen, und ein 3D-Drucker macht im Grunde wenig anderes, als – Schicht für Schicht – eine sehr große Anzahl von 2D-Plots nacheinander auszuführen. In diesem Kapitel stellen wir einen sehr schnellen fischertechnik-Plotter mit einer hochpräzisen Arduino-Ansteuerung vor.





5.1 Hintergrund

Ein Plotter ist ein elektronisches Zeichengerät, das – anders als ein Drucker, bei dem die Ausgabe Zeile für Zeile erfolgt – wie beim »Malen nach Zahlen« seine Ausgabe über eine genaue Ansteuerung der darzustellenden Punkte auf der gesamten Ausgabefläche erzeugt. Plotter eignen sich besonders für Darstellungen zusammenhängender Linien (wie Funktionsgraphen oder technische Zeichnungen) und wurden daher auch als »Kurvenschreiber« bezeichnet. Für Texte hingegen sind Laserdrucker das geeignetere Ausgabegerät, denn das Zeichnen von Buchstaben ist zeitaufwendig.

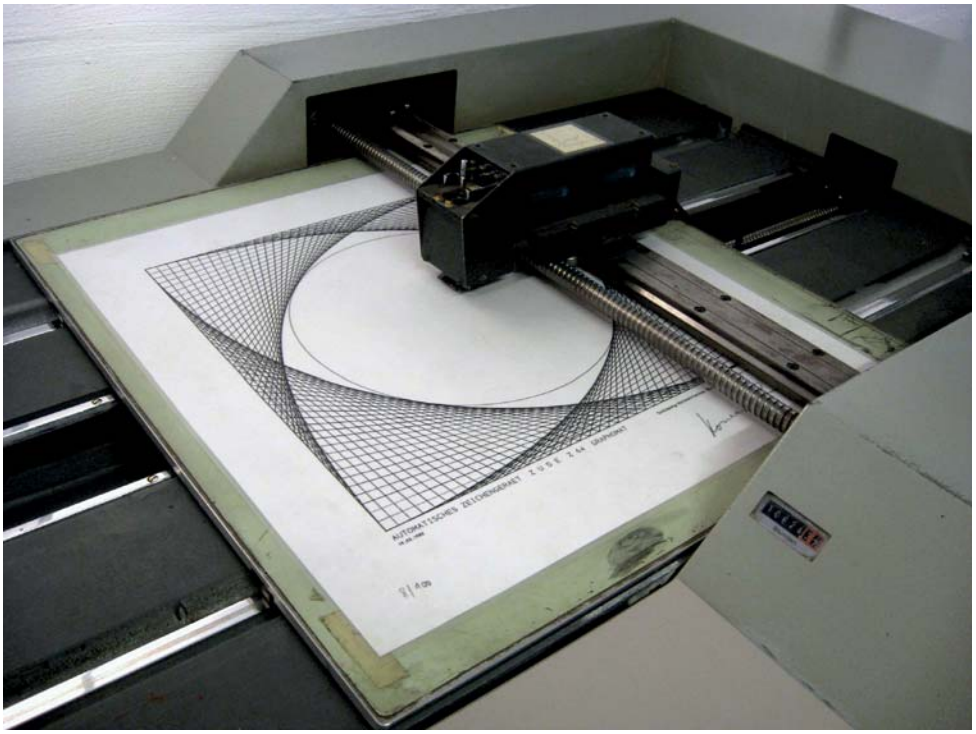


Abb. 5–1 Graphomat Z64 der Zuse AG

Die meisten Plotter sind Flachbettplotter, bei denen das Papier für die Ausgabe flach eingelegt und gegen Verrutschen fixiert wird. Plotter für sehr große Ausgabeformate wie A0 werden oft als Rollenplotter realisiert, bei denen das Papier über eine Rolle gelegt wird und die exakte Positionierung des Stifts durch die horizontale Bewegung des Schreibkopfes und die Drehung dieser Rolle (also den Transport des Papiers) erfolgt.

Plotter haben eine spannende Geschichte: *Konrad Zuse* (1910-1995), der Entwickler des ersten programmierbaren Computers, konstruierte mit seiner 1956 gegründeten Zuse AG auch den weltweit ersten Plotter – den *Graphomat Z64* –, vorgestellt vor fast 60 Jahren auf der Hannovermesse 1961 (Abb. 5–1). Die Zeichengenauigkeit des Geräts lag bei beachtlichen 0,02 mm, die Plot-Geschwindigkeit bei 22,5 mm/sec. Mit einem Gewicht von 1,4 Tonnen und einem Preis von 128.000 DM war das Gerät jedoch eher ungeeignet für das heimische Arbeits- oder Kinderzimmer.

Heute werden Plotter noch immer eingesetzt, wenn es auf eine besonders hohe Präzision der Ausgabe ankommt, wie beispielsweise bei Bauzeichnungen. In jüngster Zeit hat die Plottertechnik zudem mit der Verbreitung von Lasercuttern eine Renaissance erlebt: Das exakte Schneiden von Elementen mithilfe eines hochenergetischen Laserstrahls aus einer Materialplatte ist nichts anderes als der »Plot« einer Zeichnung.

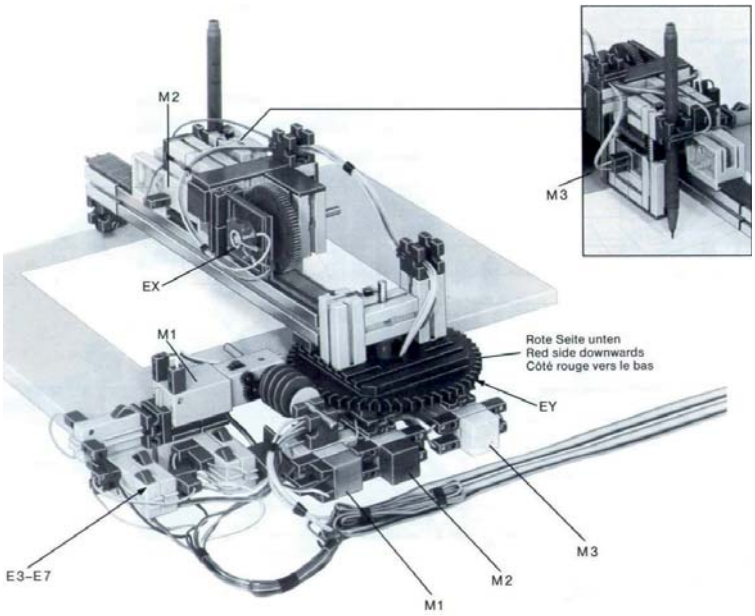


Abb. 5–2 Polarkoordinaten-Plotter aus fischertechnik Computing (30554), 1984 [2]

Ein Plotter ist ein geradezu ideales fischertechnik-Modell: Mit einer geeigneten Mechanik gelingt eine ziemlich genaue Positionierung des Schreibkopfes, und die Controller-Ansteuerung verwendet einige algorithmisch sehr interessante Lösungen. Kein Wunder also, dass sich Plottermodelle wie ein roter Faden durch die Geschichte von fischertechnik ziehen. Ein erster »XY-Schreiber« mit Handsteuerung findet sich schon im Clubheft 4/1977 [1]. Und gleich mit den ers-

ten Computing-Baukästen kamen Plottermodelle auf den Markt. Der legendäre Baukasten fischertechnik Computing aus dem Jahr 1984 (30552) enthielt einen mit Potenziometern gesteuerten Polarkoordinaten-Plotter, der in Basic programmiert wurde (Abb. 5–2) [2].

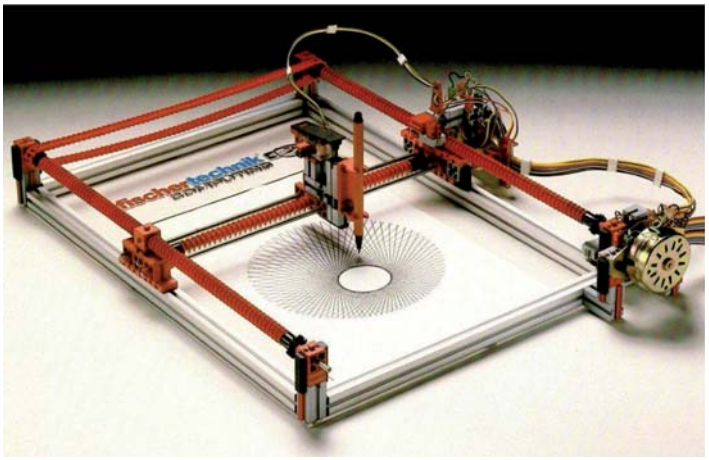


Abb. 5–3 fischertechnik-Baukasten Plotter-Scanner (30571), 1985 [3]

1985 brachte fischertechnik mit dem Baukasten Plotter/Scanner (30571) einen Flachbettplotter mit Schrittmotoren heraus, der ebenfalls in Basic programmiert wurde (Abb. 5–3, [3, 4, 5]) – 24 Jahre nach dem Zuse-Plotter und zu einem 500stel dessen Preises. Es folgten ein Plotter mit einer HP-GL-Steuerung in Basic (in CHIP Special »fischertechnik und Computer«, 1987 [6]) und der Baukasten Profi Computing (30490) mit einem Plotter, der Impulsscheiben zur Stiftabsenkung verwendete, programmiert in Turbo-Pascal und beschrieben im »Experimentierbuch Profi Computing« aus dem Jahr 1991 (36069) [7, 8].

Auch der fischertechnik-Fan-Gemeinde gelangen in den vergangenen Jahren immer wieder beeindruckende Plottermodelle, die Grafiken mit sehr hoher Präzision erzeugen konnten, wie z. B. der hochpräzise Polarkoordinaten-Plotter von David Holtz (Abb. 5–4) [9].

Für unseren Plotter haben wir viele gute Entwurfsideen dieser Plottermodelle aufgegriffen. Dabei haben wir ihn weitestmöglich mit Standardteilen von fischertechnik realisiert und versucht, die Zahl der Bauteile auf ein Minimum zu begrenzen – ohne dadurch die Präzision der Ansteuerung zu verringern.

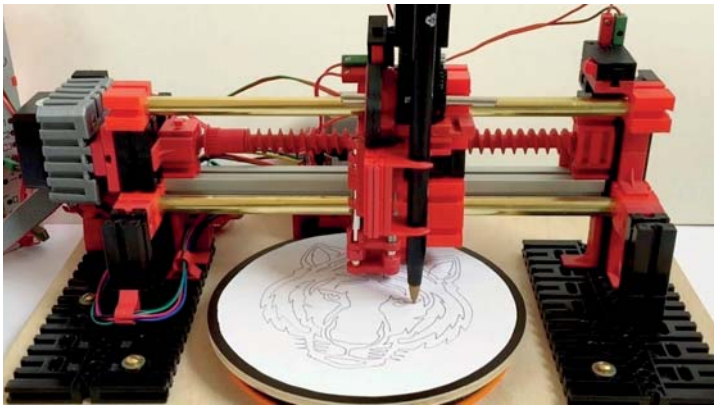


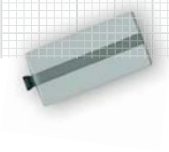
Abb. 5–4 Polarkoordinaten-Plotter von David Holtz (Foto: David Holtz) [9]

Damit der Plotter möglichst leicht nachgebaut werden kann, sollte unsere Konstruktion die folgenden Anforderungen erfüllen:

- Verwendung von ausschließlich originalen fischertechnik-Bauteilen,
- weitestgehender Verzicht auf Spezialteile, die sich nur in wenigen fischertechnik-Materialsammlungen finden (da deren nachträgliche Beschaffung vor allem bei älteren Bauteilen manchmal nicht einfach ist),
- Verwendung einer möglichst geringen Gesamtzahl an Bauteilen (»minimalistische« Realisierung),
- hohe Steifigkeit des Modells und
- hohe Präzision der Ansteuerung.

Einige dieser Anforderungen lassen sich nicht ohne Einschränkung gleichzeitig erfüllen – so erreicht man ein Maximum an Steifigkeit z.T. nur mit Spezialbauteilen wie Aluminiumprofilen oder einer großen Zahl an Bausteinen. Allerdings gilt auch: Jede zusätzliche Komplexität der Konstruktion erhöht die Anzahl der Stellen, an denen verrutschte Bausteine oder lockere Verbindungen die Genauigkeit des Plotters beeinträchtigen können. Daher ist auch bei der Steifigkeit weniger manchmal mehr. Auch das Gewicht spielt eine Rolle: Je leichter – und damit auch leichtgängiger – die beweglichen Teile sind, desto geringer ist die Beeinträchtigung der Genauigkeit des Antriebs. Daher kann sich eine minimalistische Konstruktion auch positiv auf die Präzision auswirken.

Das in diesem Kapitel vorgestellte Modell versucht einen möglichst guten Kompromiss zwischen den oben genannten Zielsetzungen. Dennoch ist unser Modellvorschlag nicht die einzig mögliche Realisierung – wie immer sind mit



fischertechnik Varianten möglich und werden von uns sogar ausdrücklich empfohlen. Denn viele Eigenschaften wie z.B. die Steifigkeit der Konstruktion lassen sich am besten durch Experimentieren nachvollziehen und womöglich gegenüber der von uns vorgeschlagenen Konstruktionsvariante noch weiter verbessern.

5.2 Mechanische Konstruktion

Ein Plotter besteht im Kern aus zwei Mechanismen: der möglichst genauen Positionierung des Schreibkopfes und der Absenkung bzw. Anhebung des Stifts. Für die mechanische Realisierung der Positionierung des Schreibkopfes, also der Bewegung des Stifts in X- und Y-Richtung, gibt es verschiedene Möglichkeiten, darunter ein Schnur- oder Kettenantrieb, die Verwendung von Zahnstangen bzw. einem Hubgetriebe oder einem Schneckengetriebe. Alle diese Mechanismen erreichen nur eine begrenzte Genauigkeit bei der Positionierung: So verringern Fertigungstoleranzen bei den Bauteilen, eine geringe Steifigkeit, die unter Krafteinwirkung zu Verformungen führt, sowie das Spiel der Antriebe die Präzision, die sich theoretisch erreichen lässt.

Von den genannten Antriebsvarianten eignen sich besonders die fischertechnik-Schneckengetriebe für eine möglichst genaue Positionierung, da sie eine sehr einfache und in der Summe vergleichsweise spiel- und reibungsarme, zugleich sehr hoch aufgelöste Ansteuerung ermöglichen. Wahrscheinlich kamen sie aus genau diesem Grund auch schon beim fischertechnik-Plotter aus dem Jahr 1985 zum Einsatz (Abb. 5–3) [3]; auch der Zuse-Plotter positionierte den Schreibkopf mit einem Schneckengetriebe (Abb. 5–1).

Mit einigen konstruktiven Kniffen lässt sich das aus Produktionstoleranzen resultierende und damit unvermeidliche Spiel der Bauteile deutlich verringern, sodass wir mit dem Schneckengetriebe eine vergleichsweise präzise mechanische Ansteuerung erhalten. Das ist für unseren Plotter besonders wichtig, da sich auch kleinste Ungenauigkeiten in der Ansteuerung über den gesamten Plot eines Bildes aufsummieren können.

Grundplatte

Als Grundplatte verwenden wir für unseren Plotter wie im 1991er-fischertechnik-Modell [7] die Experimentierplatte 500. Sie ist in vielen Baukästen enthalten und vereinfacht einen stabilen und maßgetreuen Aufbau durch die präzisen Zapfenlöcher, die ein Verrutschen des Aufbaus (wie bei Nuten möglich) verhindern. Zwar ist die Zeichenfläche damit auf etwas weniger als DIN A5 beschränkt; diese Größe ist jedoch ausreichend, um alle wesentlichen Funktionen eines Plotters zu realisieren. Die Größenbeschränkung ergibt sich auch aus einem weiteren

Grund: Da wir für die Positionierung und stabile Führung des Schreibkopfes Metallachsen verwenden, ist die Größe des Plottermodells ohnehin durch deren maximale Länge von 260 mm begrenzt.

Natürlich ist es möglich, auf der Grundlage unseres Modells einen größeren Plotter zu bauen, z. B. auf einer Grundplatte 1000 (wie im Plotter-Baukasten von 1985 [3]). Dazu benötigt man entweder 500 mm lange Metallachsen, die sich zwar nicht im originalen fischertechnik-Sortiment finden, sich aber leicht selbst anfertigen oder auf Maß beschaffen lassen, oder aber eine andere Antriebskonstruktion – und natürlich insgesamt mehr Bauteile. Hier ist viel Spielraum für eigene Modifikationen unseres Plotters.

Schlitten

Wir beginnen die Beschreibung des Plotteraufbaus mit dem mechanisch sensibelsten Teil: den Getrieben für die Positionierung. Hier ist besondere Sorgfalt vonnöten, damit die Genauigkeit der Ansteuerung möglichst wenig durch Spiel oder Reibung beeinträchtigt wird.

Um den Stift (genauer: den Schreibkopf) an jede Position des Zeichenfeldes bewegen zu können, benötigt man einen »Schlitten«, auf dem der Schreibkopf in der Horizontalen (der x-Achse) über die lange Seite der Experimentierplatte geführt wird. Dazu montieren wir ein Schneckengetriebe aus fünf Schneckenteilen (37926) mit zwei klemmbaren Schnecken (37858) und zugehörigen Zangenmuttern (31915) auf einer 260 mm langen Metallachse. Mit dieser Führung können wir den Schreibkopf fast über die gesamte Längsseite eines DIN-A5-Blattes bewegen. Um später das Spiel des Schneckengetriebes zu verringern, drehen wir nicht eine, sondern zwei Schneckenmuttern (37925) auf das Getriebe. Das Schneckengetriebe muss so montiert werden, dass die Achse auf der einen Seite etwa 0,5 cm weit herausragt. Dabei muss darauf geachtet werden, dass sich beide Schneckenmuttern ohne spürbaren Widerstand an den Schnecken entlangdrehen lassen; die Schneckenteile müssen dafür ggf. zum Schluss ein wenig gegeneinander verdreht werden, sodass zwischen ihnen kein Stück der Achse sichtbar bleibt.

Um die Achse möglichst reibungsarm zu lagern, verwenden wir für das kurze herausstehende Achsenstück eine Gelenkwürfel-Klaue (31436) mit eingeschobener Lagerhülse (36819). Auf der anderen Seite schieben wir die Achse durch einen BS 15 mit Bohrung (32064), da wir hier später noch den Antrieb ergänzen müssen. Der BS 15 mit Bohrung sollte sich möglichst reibungsfrei um die Achse drehen lassen. Am verbleibenden Ende der Metallstange montieren wir hinter dem BS 15 mit Bohrung ein Z40 so, dass die Nabenmutter nach innen und die Zähne des Kronradgetriebes nach außen zeigen (Abb. 5–5).

Abb. 5–5 Konstruktion des Schlittens (x-Achse); darauf wird später der Schreibkopf montiert.

Der Schlitten muss nun senkrecht zur Langseite der Grundplatte (also entlang der y-Achse) über den gesamten Ausgabebereich des Plotters bewegt werden können. Das realisieren wir über zwei weitere identische, an den Schmalseiten der Experimentierplatte zu befestigende Schneckengetriebe aus drei Schneckenteilen und zwei klemmbaren Schnecken auf je einer Metallachse 200. Auch hier lassen wir die Stangen auf einer Seite etwa 0,5 cm herausragen und setzen je zwei Schneckenmuttern auf das Gewinde, die sich ebenfalls ohne merklichen Widerstand drehen lassen sollten (Abb. 5–6).

Abb. 5–6 Schneckengetriebe für die Schlittenpositionierung (2x)

Als Lagerung für die beiden Achsen verwenden wir je zwei Kupplungsstücke (38253), die zwar ein wenig Spiel haben, dafür aber praktisch keine Reibung verursachen. Je eines stecken wir auf einen BS 5, das zweite auf einem BS 15 × 30 × 5 mit Nut und Zapfen.

Damit können wir nun die beiden Schneckengetriebe auf der Grundplatte montieren. Die (in Abb. 5–7 hinteren) Kupplungsstücke werden dabei gerade so weit in die Nut der BS 15 × 30 × 5 geschoben, dass sich die Achsen der Schneckengetriebe leicht drehen, ohne zu verrutschen. Auf die herausstehenden Enden der Achsen setzen wir zwei Z40 so, dass die Zähne des Kronradgetriebes nach außen zeigen; darüber erfolgt später der Antrieb. Die beiden Zahnräder werden

wir noch durch eine Kette verbinden, um die Achsen (und damit die Schneckengetriebe) synchron zu drehen (Abb. 5–7).

Schließlich fügen wir noch den Schlitten ein, indem wir die Lager für die Achse des Schneckengetriebes mit je einem BS 7,5 auf der jeweils hinteren Schneckenmutter der beiden seitlichen Schneckengetriebe fixieren. Dabei ist darauf zu achten, dass der BS 7,5 nicht auf einen der runden Zapfen geschoben wird. Dann werden auch hier die Lager so dicht an die Schnecken geschoben, dass die Achse sich leicht dreht, aber kein Spiel zum Verrutschen hat.

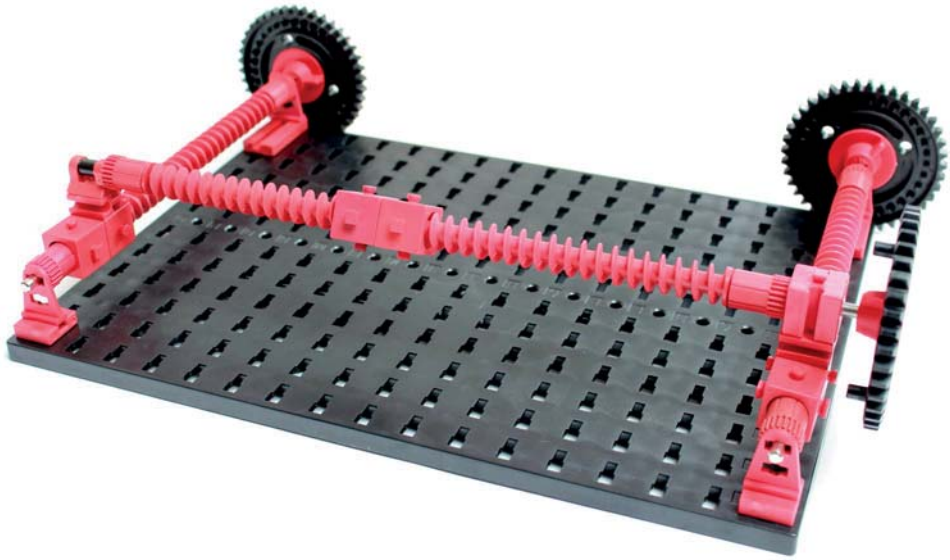


Abb. 5–7 Montage der beiden seitlichen Schneckengetriebe und des Schlittengetriebes auf der Grundplatte 500

Da wir die seitlichen Schneckengetriebe aus Stabilitätsgründen möglichst dicht über der Grundplatte angebracht haben, ragen die beiden Z40 ein wenig nach unten über die Grundplatte hinaus. Deshalb müssen wir den Plotter im Betrieb entweder auf eine mindestens 0,5 cm hohe Unterlage oder an eine Tischkante stellen. Alternativ können wir unter der Grundplatte vier »Füße« befestigen (siehe weiter unten).

Damit ist die für die Positionierung des Schreibkopfes verantwortliche Mechanik fertig. Allerdings haben die Schneckenmuttern auf den Schnecken ein wenig Spiel – sie lassen sich teilweise um mehr als einen Millimeter auf der unbewegten Spindel verschieben. Dieses Spiel lässt sich durch die Verbindung mit einer zweiten, »benachbarten« Schneckenmutter beseitigen [10, 11]. Dazu werden die beiden Schneckenmuttern durch Drehen auf der Spindel jeweils mög-

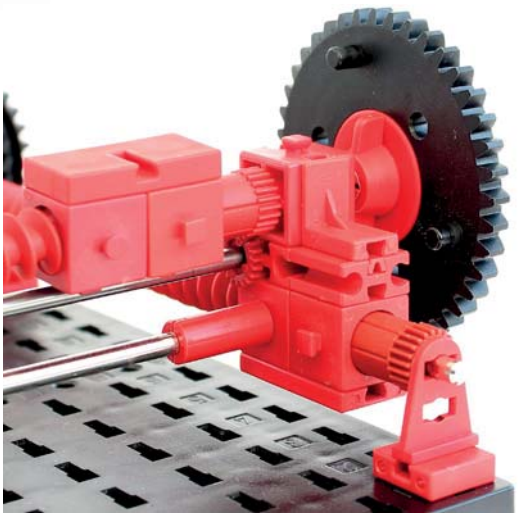


Abb. 5–8 Stabilisierung der Lagerung der Schlittenachse

lichst dicht nebeneinander gebracht, ohne das Schneckengetriebe zu verklemmen. Ist der Abstand zu groß, kann man das meist durch Drehen oder Tauschen der Schneckenmutter korrigieren. Die beiden jeweils benachbarten Schneckenmuttern der seitlichen Schneckengetriebe verbinden wir auf der Unterseite durch jeweils eine Bauplatte 15×30×3,75 mit Nut (32330); die Schneckenmuttern des Schlittens verbinden wir auf der Oberseite durch einen Baustein 15×30×5 mit drei Nuten (38428), sodass die Öffnung der oberen Nut nach »vorne« zeigt (siehe Abb. 5–8).

Damit verkürzen wir zwar die Hubstrecke der Spindeln und verkleinern so die mit dem Schreibkopf erreichbare Zeichenfläche, aber wir verhindern, dass z. B. bei einem Richtungswechsel Drehimpulse vom Spiel der Spindel »verschluckt« werden oder Verzerrungen durch ein Kippen des Schreibkopfes entstehen.

Die (in Abb. 5–7 rechte) Lagerung der Schlittenachse in einem BS 15 mit Bohrung müssen wir zusätzlich stabilisieren, da hier später noch der Antriebsmotor ergänzt wird. Dazu benötigen wir drei weitere Bauteile: einen BS 7,5, einen Federnocken und einen Winkelstein 10×15×15 (38423). Den BS 7,5 schieben wir auf die vordere Schneckenmutter, den Winkelstein mit dem Zapfen von oben in den BS 15 mit Bohrung und den Federnocken von vorne in die Nuten von BS 7,5 und Winkelstein (Abb. 5–8).

Um den Schreibkopf später stabil auf dem Schlitten zu führen, ergänzen wir den Schlitten um zwei weitere Metallachsen:

- Eine Achse 260 wird in die seitliche Nut der BS 7,5 eingeschoben, die die Lager der Schlittenachse führen, und mit zwei Riegelscheiben (36334) gegen Verrutschen gesichert.
- Eine Achse 200 verlängern wir mit zwei roten Klemmkupplungen (31024) und stecken sie auf die beiden nach innen zeigenden runden Zapfen der Schneckenmuttern, auf denen wir die Lager des Schlittens befestigt haben (siehe Abb. 5–6).

Auf ähnliche Weise wurde der Schlitten im 1985er-fischertechnik-Plotter mit zwei Führungsstangen neben der Achse des Schneckengetriebes stabilisiert (Abb. 5–3) [3].

Schreibkopf

Eine besondere Herausforderung stellt die Konstruktion des Schreibkopfes dar. Denn der Stift muss so stabil auf dem Schlitten sitzen, dass er beim Zeichnen kein Spiel hat, die Halterung sich also während des Zeichnens, z. B. bei einem Richtungswechsel, nicht verzieht. Zugleich muss er beweglich montiert sein, damit er sehr schnell auf dem Blatt abgesetzt und wieder angehoben werden kann. Dazu stabilisieren wir zunächst den auf der Spindel montierten Teil des Schreibkopfes, indem wir auf die beiden zur Schlittenachse parallelen Metallachsen einen BS 7,5 (mit der seitlichen Nut) und einen BS 15 mit Bohrung stecken und miteinander verbinden. Den BS 7,5 schieben wir auf den unteren (runden) Zapfen der (in Abb. 5–9 rechten) Schneckenmutter des Schlittens auf. Den beweglichen Teil des Schreibkopfes, an dem wir später den Stift befestigen, führen wir mit zwei Metallachsen 50 mm, die wir durch die seitlichen Nuten zweier BS 7,5 stecken und mit zwei Klemmbuchsen 5 (37679) gegen Verrutschen sichern. Den einen BS 7,5 schieben wir auf den vorderen (Achtung: nicht runden, sondern eckigen!) Zapfen der (in Abb. 5–9 rechten) Schneckenmutter, der andere BS 7,5 bleibt lose und wird die Stifthalterung aufnehmen.

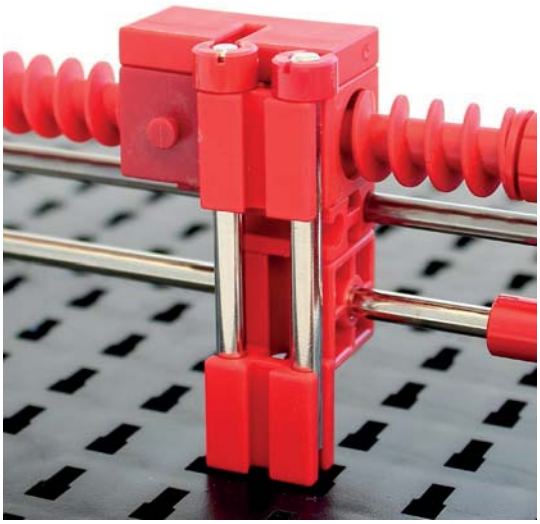


Abb. 5–9 Konstruktion des Schreibkopfes

Für den Mechanismus zum Heben und Senken des Stifts gibt es grundsätzlich drei verschiedene Möglichkeiten, die sich mit fischertechnik realisieren lassen:

- Magnetismus (Elektromagnet), verwendet im fischertechnik-Polarkoordinaten-Plotter und im Plotter/Scanner aus den Jahren 1984/1985 [2, 3]
- Luftdruck (Pneumatik-Kolben)
- Motor (mit Hubgetriebe, Schnecke oder Exzenter/Schaltscheibe) [11]

Besonders schnell schalten der Elektromagnet und ein Pneumatik-Kolben. Die fischertechnik-Elektromagnete, die auch in den ursprünglichen Plottermodellen von 1984 und 1985 (siehe Abb. 5–2, 5–3) verwendet wurden, sind jedoch nur

über eine sehr kleine Distanz (1-2 mm) stark genug und bewältigen daher nur eine geringe Hubstrecke. Ein Pneumatik-Kolben hat zwar sehr viel mehr Kraft, dafür vergrößert er den Schreibkopf erheblich, und wir müssen zusätzlich einen Kompressor für die Druckluftzufuhr vorsehen und ansteuern.

Es bleiben die mit einem Motor angetriebenen Mechanismen. Ein Mini- oder XS-Motor mit Hubgetriebe oder Schnecke funktioniert, ist aber sehr langsam – und damit für einen Plotter weniger geeignet. Anders als die beiden anderen Mechanismen kommen Hubgetriebe und Schnecke ohne Rückstellfeder aus, benötigen dafür aber einen Endlagentaster zur Feststellung des Stiftstatus (oben/unten), der einen zusätzlichen Eingang an unserem Arduino belegt. Die Abb. 5–10 zeigt eine solche Konstruktion mit einem XS-Motor, einer Rastachse mit Schnecke und passender Schneckenmutter (35977, 35973).

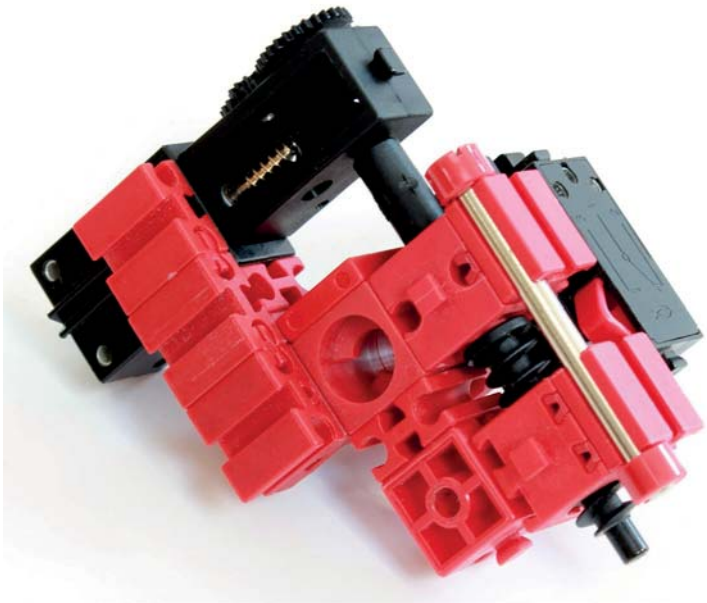


Abb. 5–10 Stiftabsenkung mit XS-Motor und Schnecke sowie Endlagentaster

Schneller und stabiler gegen ein Verrutschen der Bauteile ist ein Motor mit Exzenter, der den Stift auf das Papier drückt. Diese Lösung benötigt eine Rückstellfeder und einen Taster, der vom Exzenter betätigt wird, um den Stiftstatus festzustellen. Als Exzenter eignen sich die Schaltscheiben (37727) (Abb. 5–11) [11].

Die Grundidee für diese Konstruktion findet sich schon im fischertechnik-Plotter von 1991 [7]. Dabei drücken die Schaltscheiben den Stift aufs Papier und eine Rückstellfeder hebt ihn ab, sobald jene ihn freigeben. Ein zugleich von den

Schaltscheiben betätigter Minitaster (37783) dient als Positionsindikator: Ist er gedrückt, ist der Stift unten. Angetrieben werden die Schaltscheiben von einem Minimotor (37488) mit U-Getriebe (31078). Zwar schaltet ein Achsgetriebe etwa um den Faktor 4,6 schneller, allerdings kann es dabei – abhängig von der Stärke der Rückstellfeder – mit der Motorkraft eng werden und die Schaltscheibe bleibt hängen. Mit einem XS-Motor ließe sich der gesamte Schreibkopf noch kompakter konstruieren, aber dann reicht das Drehmoment nicht, um den Stift gegen die Federspannung herunterzudrücken.

Als Rückstellfeder kann man eine kurze Kugelschreiberfeder verwenden, die fischertechnik-Druckfedern 15 × 5 sind dafür zu lang. Eingesetzt auf der Metallstange sorgt die Feder für ein kraftvolles (und damit schnelles) Anheben des Plotterstifts.

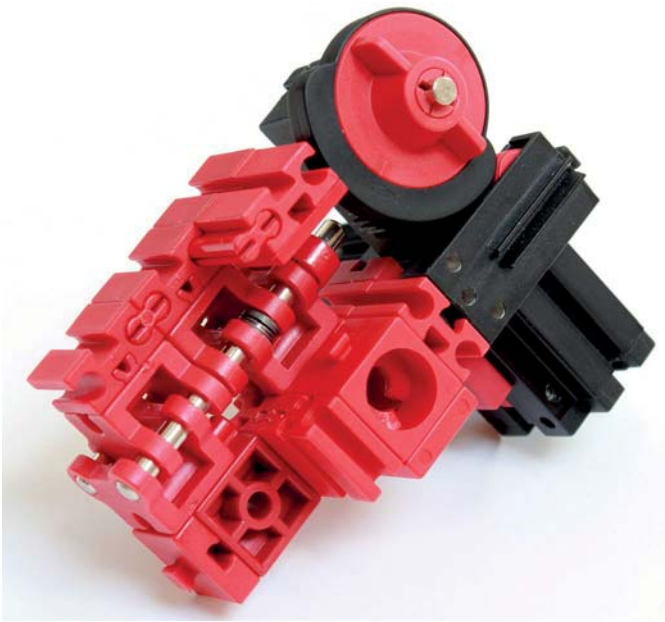


Abb. 5–11 Stiftabsenkung mit Minimotor, Schaltscheiben und Rückstellfeder [11]

Wesentlich schneller und zugleich schlanker (und eleganter) gelingt die Stiftabsenkung unter Verwendung des fischertechnik-Servomotors (132292). Wir können ihn über den Arduino sehr genau positionieren, und seine Reaktionszeiten sind ebenso kurz wie die eines Pneumatik-Kolbens; auch benötigen wir keinen automatischen Rückstellmechanismus. Mit dem TXT Controller funktioniert diese Lösung leider nicht, da er keine Servomotoren ansteuern kann – hier sind wir mit dem Arduino klar im Vorteil.

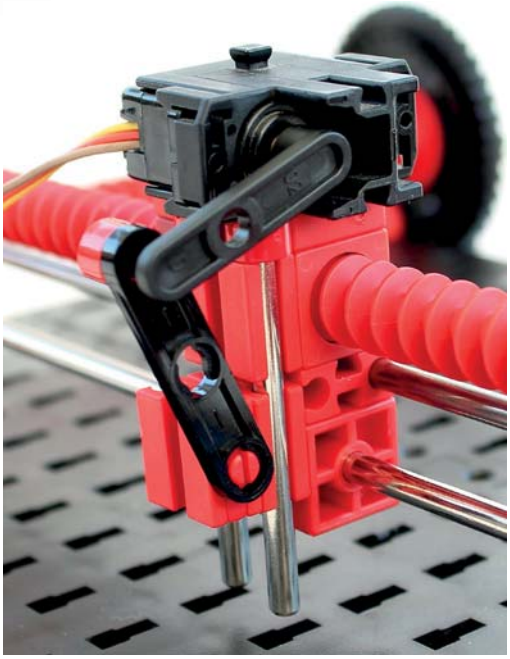


Abb. 5–12 Stiftabsenkung mit Servomotor

Zum Anbau des Servos an den Schreibkopf schieben wir einen Zapfen des Micro-Servo-Adapters (132290) in die obere Nut des Bausteins 15 × 30 × 5 (siehe Abb. 5–9), sodass der Servo seitlich auf dem Schreibkopf zu liegen kommt (Abb. 5–12). Dazu müssen wir die (von vorne betrachtet) linke der beiden Metallachsen 50 kurz entfernen und anschließend die Klemmbuchse wieder aufsetzen. Auf den Hebelarm des Micro-Servohebels LR25 (132004) schieben wir eine Strebe 30, die auf der anderen Seite mit einem Strebenadapter (31848) verbunden wird. Der wiederum wird möglichst fest in die seitliche Nut eines via Federnocken auf dem unteren BS 7,5 befestigten zweiten BS 7,5 eingeschoben. Zuletzt wird der Servohebel auf das Zahnrad der Servoachse gesteckt. Zwischen die Strebe 30 und den Servohebel können wir einen Abstandsring (31597) oder einen Klemmring 0,5 schieben – für die Funktion ist er allerdings nicht erforderlich.

Mit kleinen Servobewegungen können wir nun den unteren, mit dem Hebelarm verbundenen BS 7,5 leicht anheben und absenken. Da der Stift in geringem Abstand über das Blatt »schweben« darf, genügen schon 10-15°, um den Stift deutlich vom Blatt abzuheben – entsprechend schnell wirkt der Mechanismus.

Alle drei vorgestellten Mechanismen zur Stiftabsenkung führen jeweils einen BS 7,5 an einer durch die seitliche Nut gesteckten Metallstange auf und ab (Abb. 5–9). Auf diesem BS 7,5 wird nun die Stifthalterung montiert. Welche Bausteine man als Stifthalterung verwendet, hängt in erster Linie von dem gewählten Stift – genauer: von dessen Schaftdurchmesser – ab. Originale Plotterstifte verwenden Tusche und haben eine Strichstärke von 0,2 bis 0,7 mm – und einen beachtlichen Preis, den man vielleicht lieber in einen fischertechnik-Kasten investiert. Daher lohnt ein Blick auf mögliche Alternativen.

In das in mehreren fischertechnik-Plottermodellen eingesetzte Seilwindengestell 30 (35069) passt z. B. ein ganz klassischer Bleistift. Die »Spangen« der Halterung haben einen Durchmesser von etwa 7 mm, können aber auch etwas dickere Stifte aufnehmen. Der von uns verwendete 15 mm längere »größere Bruder« (31997) hat denselben Spangendurchmesser, reduziert jedoch das Spiel des Stifts und hat

eine deutlich größere Materialstärke, wodurch Verwindungen der Halterung praktisch ausgeschlossen sind. Bleistifte sind die günstigste Stiftvariante, allerdings nutzen sich die Minen schnell ab; bei einem längeren Plot verbreitert sich daher nach und nach die Strichstärke, bis der gesenkte Stift irgendwann kaum noch das Blatt berührt.

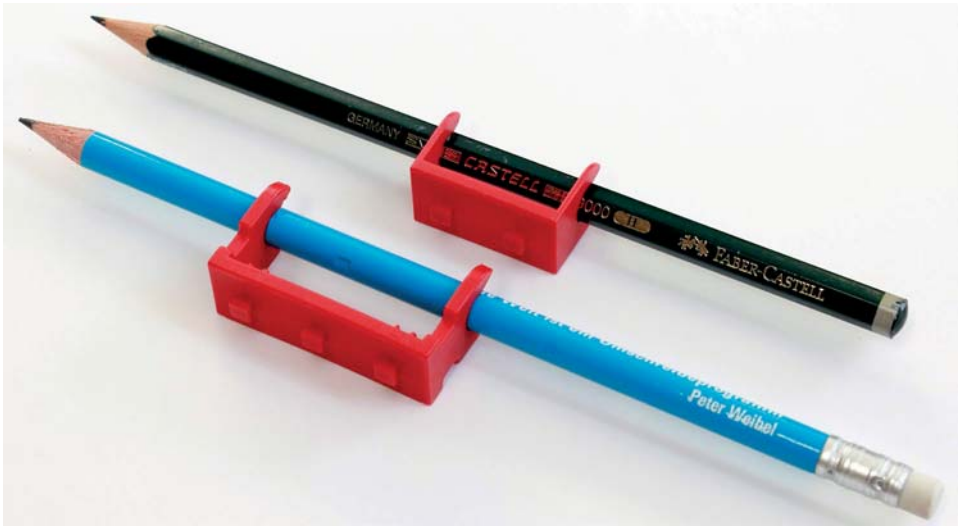


Abb. 5–13 Bleistifte mit Seilwindengestellen als Stifthalterungen

Ballpen-Minen (Strichstärke 0,7 mm) haben diesen Nachteil nicht und der Schaft passt mit 7 mm ebenfalls ins Seilwindengestell. Für einen festen Sitz müssen sie jedoch mit etwas Textilklebeband umwickelt werden. Allerdings können Ballpen-Minen Farbklekse erzeugen, wenn sie warm werden und länger an einem Punkt gehalten oder sehr langsam geführt werden.

Ein Rollenlager (37636) erlaubt das Einklemmen von Stiften verschiedener Durchmesser (ca. 3 und 8 mm) mittels Rastachsen 20 (31690) oder eines Verbindungsstücks 15 (31060). Damit können beispielsweise Fineliner (Strichstärke 0,4 mm) befestigt werden (Abb. 5–14). Diese Befestigungsmethode findet sich schon im Polarkoordinaten-Plotter von 1984 (Abb. 5–2). Ähnlich können in einem Rollenbock (32085) sehr dünne Stifte mit einem Durchmesser von ca. 2,5 mm eingeklemmt werden.

Abb. 5–14 Rollenlager als Halterung für Fineliner

Die besten Plot-Ergebnisse haben wir mit Großraum-Kugelschreiberminen erzielt. Sie lassen sich mit Klemmhülsen (35980) montieren; einen festen Sitz erreicht man auch hier, indem man den dünneren vorderen Abschnitt der Mine mit etwas Textilklebeband umwickelt. Schiebt man sie in einen senkrecht stehenden BS 7,5, wird daraus ein sehr stabiler Schreibkopf (Abb. 5–15).

Abb. 5–15 Großraum-Kugelschreibermine in Klemmhülsen

Konstruktionshinweise

Beim Aufbau des Plotters muss sehr exakt gearbeitet werden, denn selbst kleinste Ungenauigkeiten können eine fehlerhafte Ansteuerung verursachen. Vor allem die Führungen auf den Stangen, sowohl beim Schreibkopf als auch beim Schlitten, sollten möglichst leichtgängig sein; sie dürfen weder bremsen noch verkanten. Bei allen Bausteinen sollten die Zapfen fest in den Nuten sitzen. Bei lockeren Verbindungen sollte man lieber die Bauteile gegen andere, fester sitzende austauschen, damit sich der Plotter im Betrieb nicht verzieht.

Vor größeren Plots sollte zudem der gute »Sitz« des Schlittens auf den Schnecken noch einmal überprüft werden, denn jedes Spiel der Schneckenmuttern kann sich leicht zu deutlich sichtbaren Fehlern im Plot-Ergebnis aufsummieren. Zwischen den Schneckenelementen dürfen auch keine Fugen bestehen, in denen die Schneckenmuttern hängen bleiben oder verkanten können.

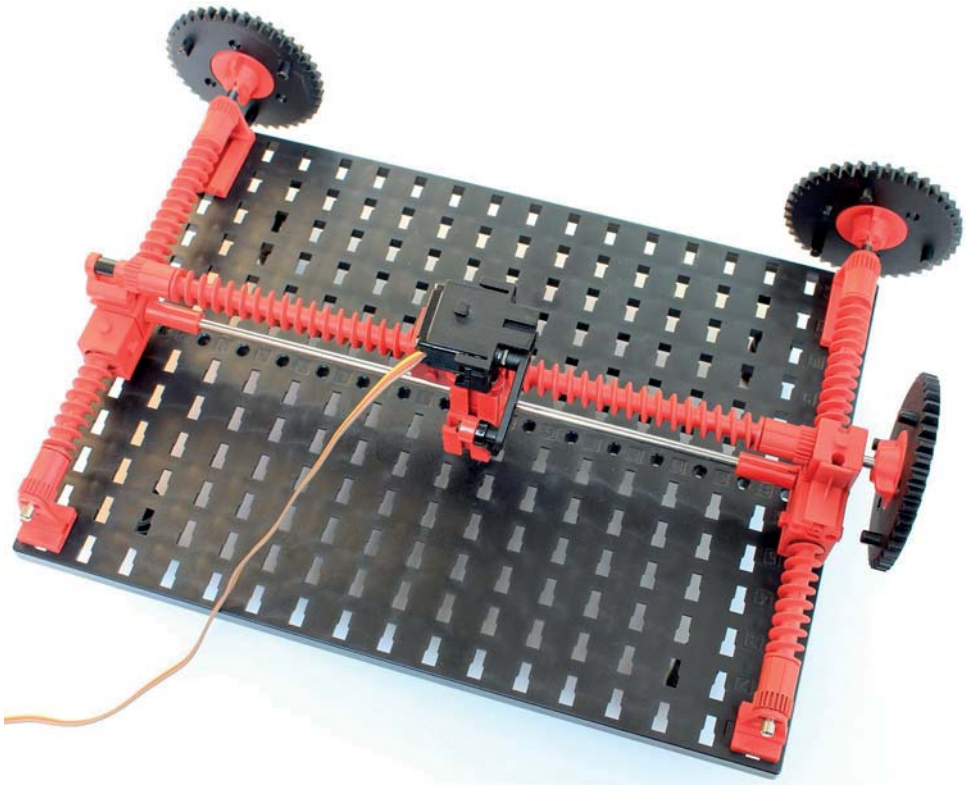
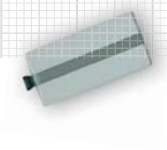


Abb. 5–16 Gesamtansicht des Plottermodells (noch ohne Antrieb)

Damit die nach unten über die Grundplatte hinausragenden Z40 nicht aufsetzen, können wir dem Plotter vier Füße spendieren. Dafür eignen sich Metallachsen 30 (31034) mit aufgesetztem schwarzen Rad 23 (36574) oder die schwarzen Kunststoff-Federn 26 (31760) (Abb. 5–17). Sie lassen sich von unten in dafür vorgesehene Löcher der Grundplatte stecken.

Abb. 5–17 Kunststoff-Federn oder Metallachsen 30 mit Rad 23 als Plotterfüße



5.3 Antrieb

Für den Antrieb benötigen wir Motoren, mit denen wir eine präzise Positionierung des Schreibkopfes vornehmen können. Das gelingt nur, wenn wir die Motorumdrehung in möglichst feiner Auflösung steuern können und die Motoren exakt am gewünschten Punkt stoppen. Zwar lässt sich die Auflösung der Steuerung durch eine entsprechende Untersetzung des Antriebs vergrößern, allerdings verliert man mit jedem Übersetzungsschritt an Präzision, da jedes Getriebe über ein gewisses Spiel verfügt, durch das die Genauigkeit der Ansteuerung beeinträchtigt wird. Und natürlich verlängert sich die Dauer des Plottens entsprechend. Idealerweise verwenden wir also einen Motor, mit dem wir direkt oder über maximal eine Untersetzung die Schneckengetriebe für die Positionierung von Schlitten und Schreibkopf antreiben und dessen Achsenposition wir in möglichst feiner Auflösung bestimmen können.

Nach einigen Experimenten haben wir uns gegen die Verwendung der fischertechnik-Encodermotoren entschieden, denn bei der Ansteuerung kann es vor allem bei sehr kurzen Bewegungen von wenigen Schritten (die bei einem Plotter sehr häufig sind) zum »Überschwingen« kommen, d. h., der Motor dreht – selbst bei einer geringen Motorgeschwindigkeit – gelegentlich über die gewünschte Zahl an Encoderschritten hinaus, bevor er tatsächlich hält. Die sich dadurch aufaddierenden Fehler verfälschen das Plot-Ergebnis erheblich, sofern man nicht die Auflösung der zu plottenden Grafik und damit natürlich auch die Genauigkeit des Resultats verringert. Das passiert auch bei einer Ansteuerung durch den TX/TXT Controller, allerdings sorgt hier der Controller für ein frühzeitiges »Abbremsen« des Motors, sodass die Auswirkungen nicht ganz so stark sind; auch lässt sich das Überschwingen durch eine Verringerung der Motorgeschwindigkeit reduzieren [12]. Der Preis einer geringeren Motorgeschwindigkeit ist jedoch eine lange Laufzeit des Ausgabeprogramms.

Stattdessen setzen wir in unserem Plottermodell zwei bipolare NEMA-14-Schrittmotoren (10 V, 0,5 A je Strang, 200 Schritte/Umdrehung) für die Ansteuerung der x- und der y-Achse ein, mit denen wir den Schreibkopf ganz exakt positionieren können. Schrittmotoren verwenden keine Sensoren zur Positionsbestimmung (wie Encoder, Drehgeber o. Ä.) und sind daher deutlich präziser in der Ansteuerung. Ein Überschwingen kann bei ihnen nur auftreten, wenn die angetriebene Masse ein (externes) Lastmoment erzeugt, das das Haltemoment des Motors übersteigt. Motoren des amerikanischen NEMA-14-Standards (*National Electrical Manufacturers Association*) haben ein definiertes Haltemoment von 0,3 Nm – für unsere Zwecke ist das bei Weitem ausreichend. Im fischertechnik-Baukasten des 3D-Druckers (160842) sind ebenfalls NEMA-14-Schrittmotoren

enthalten, allerdings dürfte sich der nur in wenigen fischertechnik-Sammlungen finden. Inzwischen können die Motoren auch als Einzelteil erworben werden. Sie sollten gleich zusammen mit den Schrittmotorhaltern des 3D-Druckers (160528) beschafft werden, da wir sie mit diesen stabil in einem fischertechnik-Modell verbauen können. Geeignete NEMA-14-Schrittmotoren gibt es jedoch auch im Elektronik-Fachhandel; auf der Webseite zum Buch finden sich entsprechende Bezugsquellen.

Unsere NEMA-14-Schrittmotoren haben einen weiteren wichtigen Vorteil gegenüber den fischertechnik-Encodermotoren: Sie haben eine Auflösung von 200 Schritten je Achsumdrehung; das entspricht 1,8° pro Schritt. Das ist mehr als das Doppelte der Auflösung der fischertechnik-Encodermotoren der ersten Generation (75 Schritte je Achsumdrehung oder 4,8°) und mehr als das Dreifache der Encodermotoren der zweiten, aktuellen Generation (63 1/3 Schritte oder etwa 5,68°).

Auf die einseitig abgeflachte 4 mm-Achse der NEMA-14-Motoren lassen sich mit ein wenig Kraft die roten Z10 der Power-Motoren (159577) aufstecken, die auch als Antriebsritzel für die Schrittmotoren des 3D-Druckers verwendet werden. Die Motoren montieren wir so, dass sie die Z40 der Schlitten- und Schreibkopfachsen über die 32 Zähne des Kronrads antreiben.

Für die beiden seitlichen Schneckengetriebe benötigen wir zur Montage des Schrittmotors lediglich einen BS 30 mit einem Federnocken, dessen Zapfen wir seitlich in eine Nut des Schrittmotorhalters einschieben. Der BS 30 wird so auf die Grundpatte gesteckt, dass das Antriebsritzel in das Kronradgetriebe eines der beiden Z40 eingreift (Abb. 5–18).

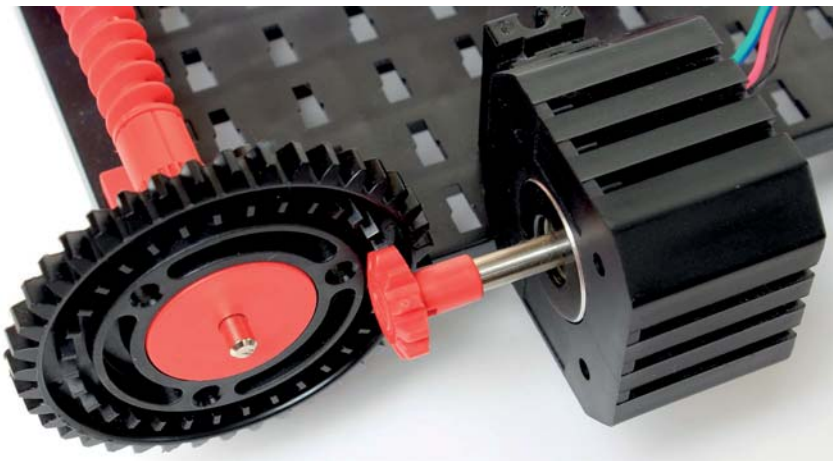


Abb. 5–18 NEMA-14-Schrittmotor mit Ritzel Z10 und Kronrad (Schlittenantrieb)

Nun verbinden wir noch die beiden Z40 mit einer Kette. Diese sollte leicht, aber nicht zu stark gespannt sein: Die beiden Schneckengetriebe müssen sich durch manuelles Ziehen an der Kette weiter leicht drehen lassen. Auf diese Weise kann man das Kettenspiel fast vollständig ausgleichen. Zur Präzision tragen auch die beiden Z40 bei: Im 1985er-Plotter verband fischertechnik zwei Z10er über eine (zudem leicht durchhängende) Kette – das Kettenspiel wirkte sich dort (Abb. 5–3) viermal so stark auf die Schneckengetriebe aus wie in unserem Modell und dürfte bei Richtungswechseln erkennbare Verzerrungen verursacht haben.

Entsprechend verfahren wir beim Schlittenantrieb. Dazu setzen wir auf der rechten Seite des Schlittens einen BS 30 mit je einem BS 5 an jedem Ende auf den roten BS 15 mit Bohrung. Oben ergänzen wir einen BS 7,5, den wir mit einer seitlichen Nut auf den BS 5 stecken. In die mittlere Nut schieben wir einen Federnocken und befestigen den Schrittmotorhalter an dessen Zapfen (Abb. 5–19).

Abb. 5–19 Schrittmotor NEMA-14 mit Ritzel Z10 und Kronrad (Schreibkopfantrieb)

Durch die Untersetzung über das Kronrad des Z40 (32:10) erhöhen wir die Auflösung der beiden Schneckengetriebe um den Faktor 3,2 auf 640 Schritte je (Motor-)Achsumdrehung, also 0,5625° je Motorschritt. Da die fischertechnik-Schneckengetriebe bei jeder Achsumdrehung eine aufsitzende Schneckenmutter um genau 1 cm weiterbewegen, erreicht diese Ansteuerung eine Genauigkeit von 0,015625 mm. Durch die Ansteuerung der Schrittmotoren im Double-Modus verdoppeln wir die Auflösung sogar auf 0,0075 mm (7,5 Mikrometer). Damit ist unser Plotter – zumindest theoretisch – deutlich präziser als die Standard-

auflösung von HP-Plottern mit 1/1.016" = 0,025 mm und übertrifft auch die Auflösung von Zuses Graphomat Z64. In der Praxis erreichen wir diese Genauigkeit allerdings bei Weitem nicht, und zwar aufgrund des Spiels von Schneckengetriebe und Stiftbefestigung und natürlich wegen der Dicke der Schreibmine.

Schließlich müssen wir noch zwei Endlagentaster vorsehen, an denen das Erreichen des »Nullpunkts« (untere linke Ecke) erkannt werden kann (Abb. 5–20, 5–21). Das ist mechanisch nicht sehr präzise (die Taster werden nicht immer an exakt der gleichen Stelle aktiviert), aber an dieser Stelle benötigen wir auch keine hohe Genauigkeit: Der Schreibkopf wird bei der Initialisierung an einen Startpunkt bewegt und dieser Punkt wird als Punkt (0, 0) festgelegt.

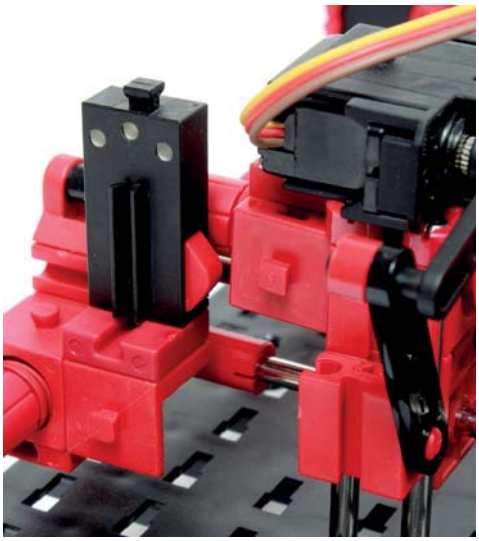


Abb. 5–20 Endlagentaster Schreibkopf

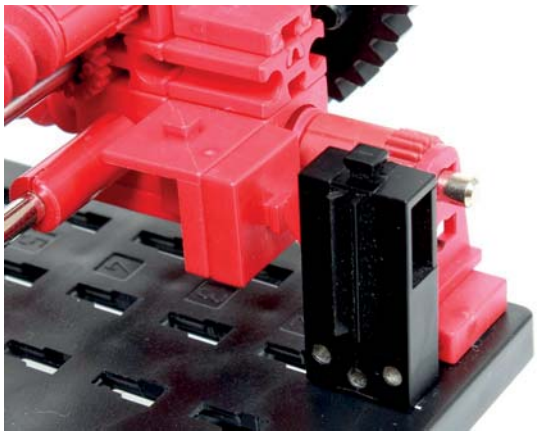


Abb. 5–21 Endlagentaster Schlittenantrieb

Damit ist unser Plotter (Abb. 5–22) mechanisch betriebsbereit. Als Hilfestellung für den Nachbau finden sich auf der Webseite zum Buch eine mit dem Programm »fischertechnik Designer« erstellte Konstruktionsdatei, unterteilt in mehrere Bauphasen, sowie eine Bauteilliste. Was nun noch fehlt, sind der Anschluss aller Sensoren (Taster), Aktoren (Servo- und Schrittmotoren) und eines Speichermediums an den Arduino sowie ein Sketch zum Einlesen einer Vektorgrafik-Datei und zur Ansteuerung des Plotters.

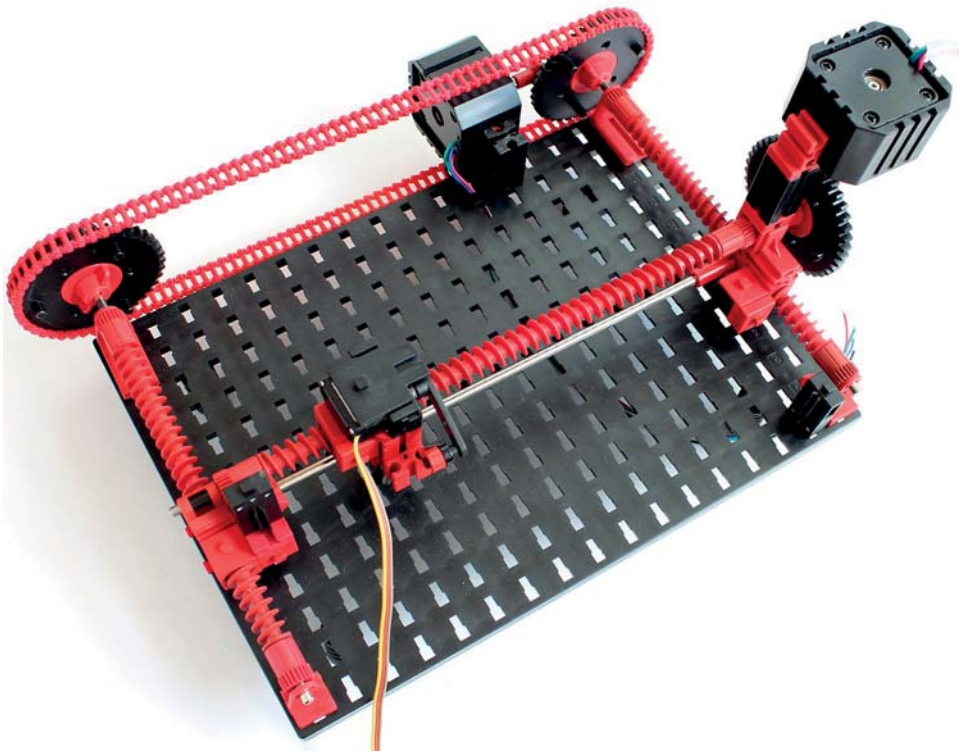


Abb. 5–22 Gesamtansicht des Plotters

5.4 Anschluss an den Arduino

Für den Betrieb des Plotters benötigen wir zunächst unseren Arduino mit einem aufgesteckten Adafruit Motor Shield. Das Motor Shield kennen wir bereits aus den vorausgegangenen Kapiteln, daher gehen wir hier nicht weiter darauf ein. Den Servomotor verbinden wir mit dem Servo-2-Anschluss (oben links auf dem Shield), damit belegen wir zugleich Pin D9.[1]

Die beiden NEMA-14-Schrittmotoren schließen wir an die Motoranschlüsse M1/M2 und M3/M4 an. Dabei wird je eine Spule mit einem Motoranschluss verbunden. Bei unseren Pololu-Schrittmotoren gehören die Kabelpaare in den Farben Rot/Blau und Schwarz/Grün zu jeweils einer Spule (Abb. 5–23). Damit die Drehrichtung der Motoren stimmt, werden das rot/blaue Kabelpaar des x-Achsen-Motors an M1 und das des y-Achsen-Motors an M3 angeschlossen;

1 Der Servo-1-Anschluss belegt Pin D10, den wir für das CS-Signal der SPI-Verbindung zum SD-Kartenleser verwenden; siehe nachfolgender Text.

entsprechend verbinden wir das schwarz/grüne Kabelpaar des x-Achsen-Motors mit M2 und das des y-Achsen-Motors mit M4.

Die Endlagentaster verbinden wir mit den digitalen Pins D4 (Endlage x-Achse) und D5 (Endlage y-Achse) sowie jeweils einem GND-Pin. Pin A0 nutzen wir wie bei unserem Buggy (Kapitel 3) für den Anschluss eines Starttasters, mit dem wir den Sketch und damit den Plot einer Vektorgrafik-Datei starten können. Alle drei Pins werden bei der Initialisierung auf `INPUT_PULLUP` gesetzt. Damit liefern sie bei offenem Taster ein stabiles `HIGH`-Signal.

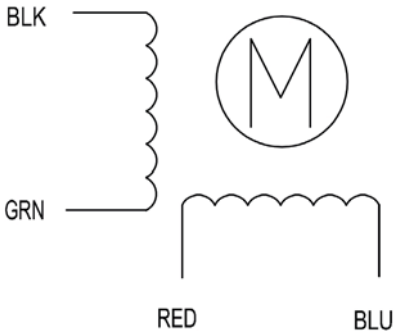


Abb. 5–23 Anschlüsse der NEMA-14-Schrittmotoren

Mit der zusätzlichen Stiftleiste, die wir in Kapitel 4 auf unser Motor Shield gelötet haben, kommen wir mit den verfügbaren GND-Anschlüssen auch ohne eine Verteilerplatte (31327/31328) aus.

Schließlich benötigen wir noch ein Lesegerät, mit dem wir von einem Datenspeicher die zu plottende Vektorgrafik-Datei einlesen können. Wir verwenden dazu einen Micro-SD-Kartenleser[2], der FAT16- oder FAT32-formatierte Micro-SD-Karten (bis 2 GB bzw. 32 GB bei SDHC-Karten, *SD High Capacity*) lesen kann und via SPI (*Serial Peripheral Interface*) angesprochen wird (Abb. 5–24).

Abb. 5–24 Micro-SD-Kartenleser-Board von Adafruit mit SPI-Schnittstelle

Angeschlossen wird der SD-Kartenleser am Arduino an einen GND- und einen Vcc-Pin (5 V) sowie an die digitalen Pins D13 (CLK – *Clock*), D12 (DO – *Data Out*) und D11 (DI – *Data In*). Für den *Chip Select*-Anschluss (CS) verwenden wir Pin D10. Stattdessen kann der Kartenleser auch mit den sechs ICSP-Anschlusspins des Arduino verbunden werden (siehe Kapitel 2 sowie den Anschluss der Pixy-Kamera in Kapitel 3).

Den Anschluss der Schrittmotoren an das Motor Shield, die Verkabelung der Endlagentaster und die Verbindung des Micro-SD-Kartenlesers mit unserem Arduino zeigt die Schemazeichnung in Abb. 5–25.

2 Wir verwenden das Adafruit Micro-SD Card Breakout Board [13]. Bezugsquellen und alternative Micro-SD-Kartenleser haben wir auf der Webseite zum Buch verlinkt.

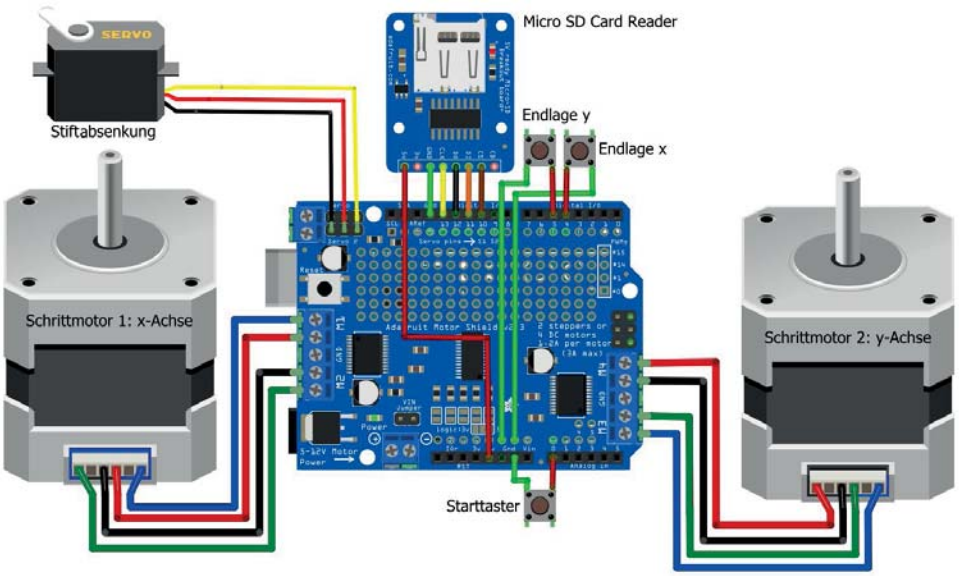


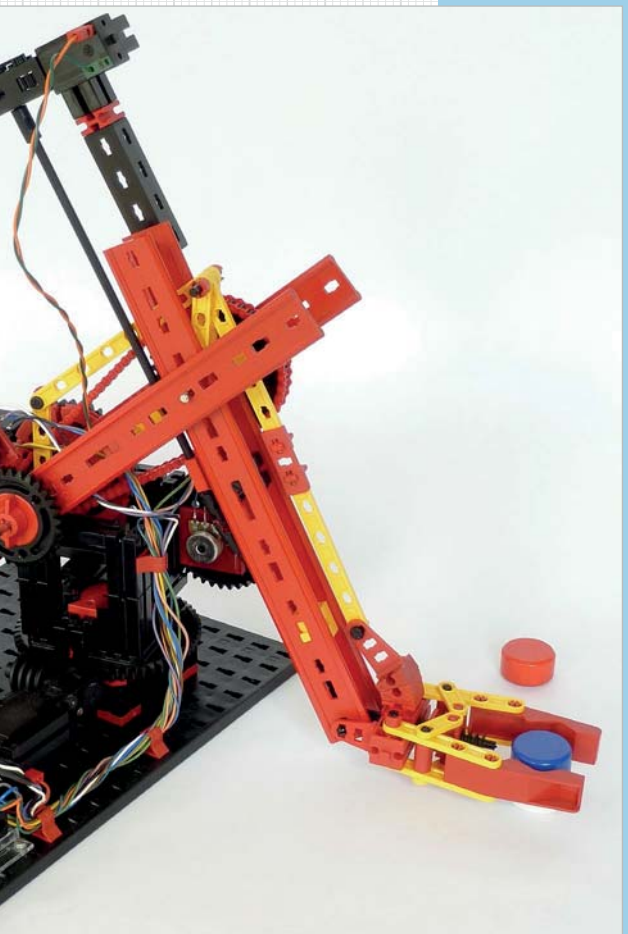
Abb. 5–25 Anschluss der Schrittmotoren, des Servos, des Start- und der Endlagentaster sowie des Micro-SD-Kartenlesers am Arduino

Die Formatierung der Micro-SD-Karte (mit FAT16 oder FAT32) und die Speicherung unserer Grafikdatei nehmen wir an einem Rechner mit SD-Schnittstelle über einen SD-Kartenadapter (Abb. 5–26) vor, der bei einigen Micro-SD-Karten bereits beiliegt.[3]

Abb. 5–26 Micro-SD-Karte von SanDisk mit SD-Adapter

3 Manche Micro-SD-Karten werden bereits formatiert geliefert. Wenn die Karte nicht oder mit dem falschen Format (bspw. NTFS) formatiert ist, gelingt die (Neu-)Formatierung unter Microsoft Windows am einfachsten über die Kommandozeile: Windows-Taste, dann im Suchfenster »CMD« eingeben. Die Formatierung erfolgt mit dem Befehl `format;` die Eingabe von `format /?` liefert eine Beschreibung der benötigten Parameter. Wenn die Micro-SD-Karte unter dem Laufwerksbuchstaben `z:` erreichbar ist, lautet das Kommando `format z: /FS:FAT /V:Plotter /Q` oder `format z: /FS:FAT32 /V:Plotter /Q`. *Achtung:* Beim Formatieren werden alle zuvor auf der Micro-SD-Karte gespeicherten Daten gelöscht!

6 Der Greifer



Roboterarme bestimmen heute das Bild industrieller Fertigungsanlagen. Sie automatisieren Schweißprozesse, das Anbringen von Bauteilen und viele weitere sich wiederholende Tätigkeiten. Dabei arbeiten sie schneller und genauer als menschliche Arbeitskräfte, machen keine Fehler und benötigen keine Pausen. Schon im Jahr 1985 brachte fischertechnik einen Trainingsroboter auf den Markt, der es Heimcomputer-Hobbyisten ermöglichte, einen Einblick in die Welt der Industrieroboter zu erhalten und viel über ihre Mechanik und Programmierung zu erlernen [2].
Unser Greifer ist eine Hommage an diesen Trainingsroboter. Wir haben aber die Sensorik, Mechanik und vor allem die Programmierung vereinfacht. Das einfache und klare Arduino-Konzept zeigt hier seine ganze Stärke. Als Anwendung demonstriert unser Roboter die Lösung des klassischen Knobelspiels »Die Türme von Hanoi«.

6.1 Industrieroboter in Kinderzimmern

Wer schon immer gerne mit fischertechnik gespielt hat, weiß, wie viele kleine und große Entdeckungen man beim Experimentieren und Bau von Modellen machen kann. Tatsächlich haben Konstruktionssysteme ihr Innovationspotenzial in der Geschichte der Robotik noch deutlich eindrucksvoller unter Beweis gestellt. Nach heutigem ISO-8373-Standard ist ein Industrieroboter ein automatisch gesteuerter, frei programmierbarer Manipulator mit drei oder mehr Drehachsen. Laut der englischsprachigen Wikipedia wurde der erste solche Roboter im Jahr 1938 im *Meccano Magazin* vorgestellt (Abb. 6–1). Entwickelt wurde er von 1935 bis 1937 vom damals 21-jährigen Studenten *Griffith P. Taylor*, der ihm den Namen *Gargantua* gab [3, 4, 5]. Der frei programmierbare Ablauf der Bewegungen wurde auf einem Lochstreifen gespeichert. Um einen geordneten Stapel Bausteine zur abgebildeten Struktur aufzutürmen, benötigte Gargantua etwa 50 Minuten.

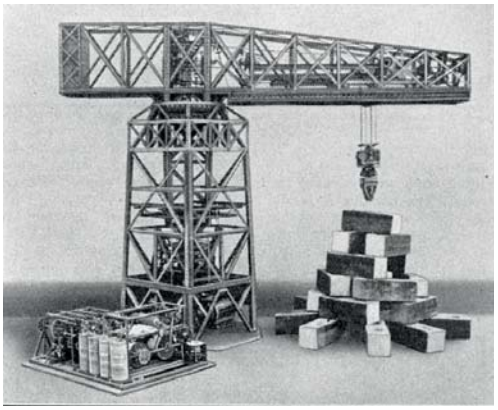


Abb. 6–1 Der Meccano-Roboterkran Gargantua aus dem Jahr 1938 [3]

Das erste Patent auf einen elektronisch gesteuerten Industrieroboter für den tatsächlichen Einsatz in der Industrie bekam *George Devol* (1912-2011) im Jahr 1954. Es dauerte allerdings noch bis zum Jahr 1960, bis das erste Exemplar verkauft wurde. In den folgenden Jahrzehnten verbreiteten sich die Roboter in Fertigungsanlagen immer schneller – vor allem in der Automobilindustrie.

Die ersten Heimcomputer kamen Ende der 70er-Jahre auf den Markt. Sie wurden in der ersten Hälfte der 80er-Jahre erschwinglich und eroberten die häuslichen Wohn- und Kinderzimmer. So kostete beispielsweise der Commodore C64 zu seiner Einführung im Jahr 1983 noch 1495 DM, doch halbierte sich der Preis noch im selben Jahr. Eine unvergleichlich kreative Pionierzeit begann.

Die Firma fischertechnik erkannte früh das Potenzial dieser Entwicklung und brachte schon 1984 einen genialen Computing-Baukasten auf den Markt, der die (elektro-)mechanischen Stärken des fischertechnik-Systems auf ideale Weise mit den neuen Möglichkeiten der Heimcomputer verband und Jugendlichen und Hobbyisten einen fundierten Einblick in die Welt der Automatisierung eröffnete (Abb. 6–2).

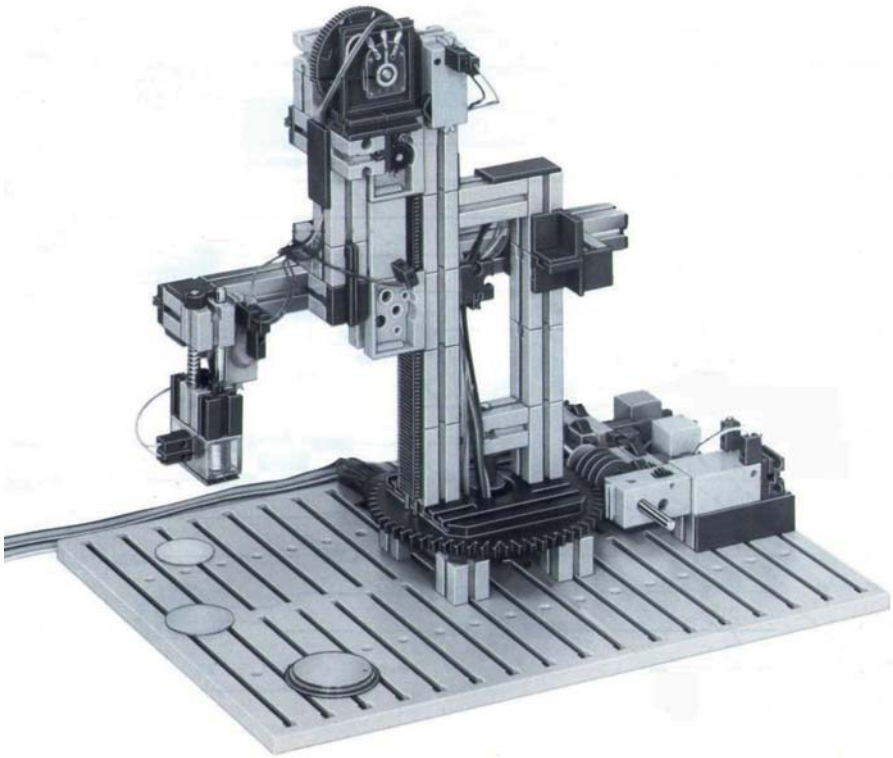


Abb. 6–2 »Die Türme von Hanoi« mit dem fischertechnik-Computing-Set aus dem Jahr 1984 [1]

Der Computing-Baukasten enthielt viele für die damalige Zeit spektakuläre Modelle, unter anderem einen Teach-in-Roboter, ein Grafik-Tablett, einen Polarkoordinaten-Plotter und einen Roboter, der die Lösung des Knobelspiels »Die Türme von Hanoi« vorführt.

Genau diese letzte, klassische Anwendung – »Die Türme von Hanoi« – greifen wir in diesem Kapitel auf. Unser Roboter stapelt allerdings die verschieden großen Scheiben der Türme nicht mit einem Elektromagneten um, sondern mit einer Greifhand.

Unser Greifer (Abb. 6–4) ist eine Hommage an den fischertechnik-Trainingsroboter, der schon ein Jahr später, 1985, dem Computing-Baukasten an die Seite gestellt wurde (Abb. 6–3). Bei beiden Robotern kommen vier Motoren zum Einsatz: drei S-Motoren zum Drehen des Körpers, des Oberarms und des Unterarms und ein kleinerer Mini- bzw. XS-Motor zur Steuerung des Greifers. Die Greifhand wird dabei stets parallel zur Grundfläche gehalten. Dies ist ein sehr guter Kompromiss zwischen Funktionalität und Komplexität.

Abb. 6–3 fischertechnik-Trainingsroboter aus dem Jahr 1985 [2]

Im Gegensatz zum fischertechnik-Trainingsroboter ist unser Greifer in der Lage, verschieden große Werkstücke sicher aufzunehmen und an anderer Stelle wieder abzusetzen. Erst diese Fähigkeit ermöglicht es, die Türme mit einer Greifhand umzustapeln.

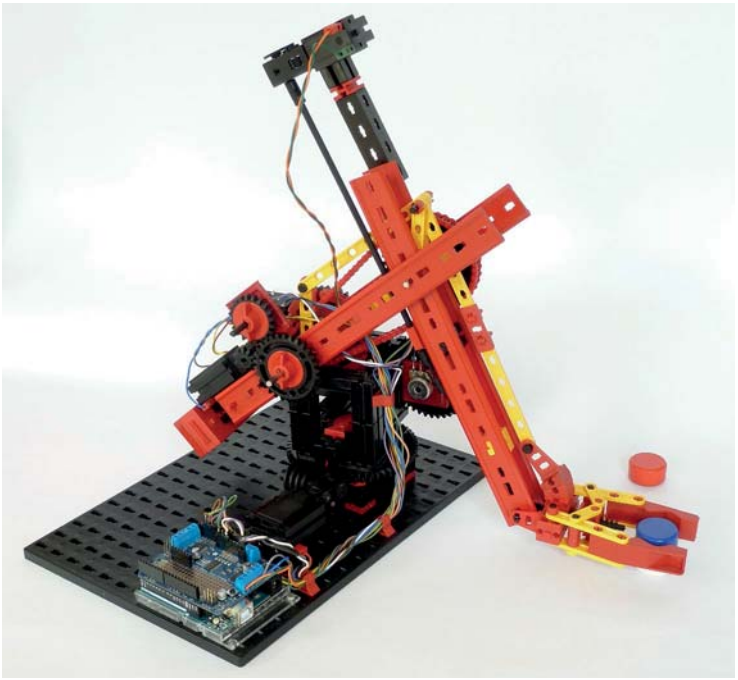


Abb. 6–4 Unser Arduino-gesteuerter Greifer

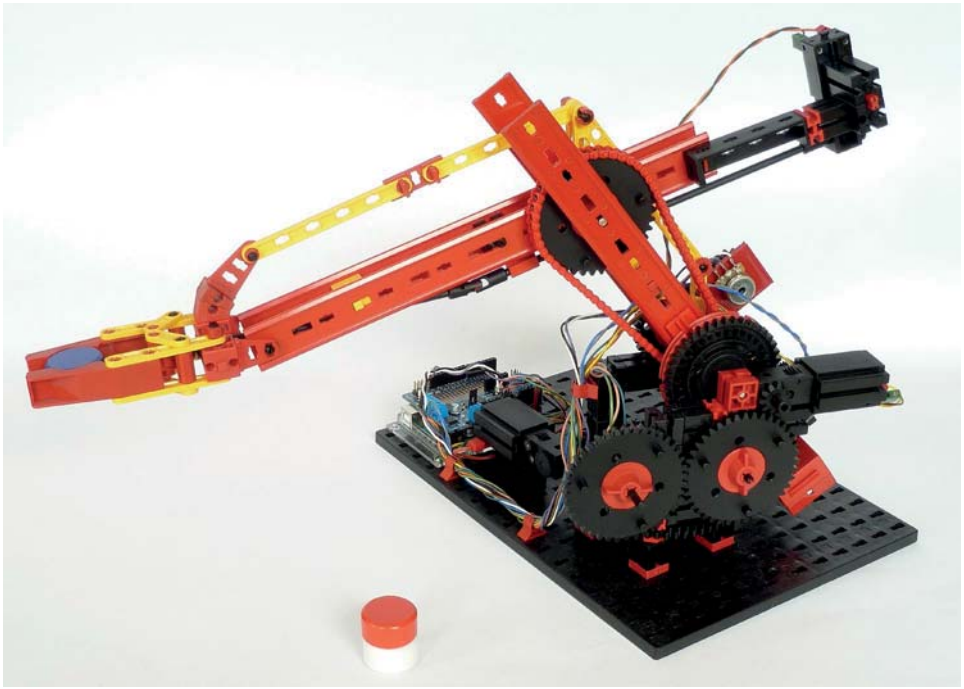
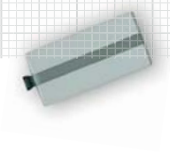


Abb. 6–5 Unser Greifer von der anderen Seite

Wer schon einmal das Innenleben von CD-Playern in den 80er-Jahren mit dem heutiger Player verglichen hat, kennt die Entwicklung bei solchen Produkten hin zu Effizienz, Übersichtlichkeit und Kostenersparnis. Diese allgemeine Entwicklung spiegelt sich auch beim Vergleich zwischen unserem Greifer und dem Trainingsroboter aus dem Jahr 1985 wider: Statt vier Tastern benötigen wir gar keinen, statt teurer Gabellichtschranken setzen wir als Drehgeber günstige Potenziometer ein, statt teurer Aluminiumprofile im Trainingsroboter verwenden wir günstige S-Laufschienen (36333).

Potenziometer kamen schon im Computing-Baukasten aus dem Jahr 1984 zum Einsatz, allerdings sind die heutigen Leitplastik-Potenziometer deutlich langlebiger als die damals verwendeten Kohleschicht-Potenziometer. Schon damals war ein Teach-in-Roboter eines der Hauptmodelle des Baukastens. Allerdings benötigte das Anlernen der Positionen noch eine Vielzahl von Tastern. Wir stellen hier ein einfacheres und schnelleres Verfahren dar, bei dem wir die Motoren auskoppeln und den Roboterarm manuell zu den gewünschten Positionen bewegen.

Die Laufschienen haben gegenüber den Aluminiumprofilen nicht nur den Vorteil, dass sie deutlich weniger kosten, sondern auch dass der Platz zwischen ihnen sinnvoll genutzt werden kann. So können wir den Unterarm unseres Roboters



ausbalancieren, indem wir den Greifer durch einen XS-Motor ansteuern, der auf der anderen Seite des Unterarms als Gegengewicht zum Greifer dient. Die entsprechende Antriebswelle verläuft zum Teil zwischen den Laufschienen. Auch den Oberarm haben wir durch ein Gegengewicht ausbalanciert. Damit wird es möglich, Schulter- und Ellenbogengelenke ohne Schnecken oder Schrauben direkt über U-Getriebe und Ritzel anzutreiben.

Den größten Fortschritt in Richtung Effizienz und Kostenersparnis gibt es jedoch bei der Steuerung. Der fischertechnik-Trainingsroboter wurde von einem der damaligen Homecomputer (Commodore, Schneider, Apple, Atari, Acorn, IBM) und einem fischertechnik-Computing-Interface gesteuert. Unser Greifer wird von einem autarken Arduino mit Motor Shield gesteuert. Der Arduino ist für Steuerungsaufgaben deutlich besser geeignet als die Kombination aus Heimcomputer und Interface, er ist kleiner und billiger als die damaligen Interfaces und lässt sich wesentlich komfortabler programmieren.

Wie schon oben beschrieben wird auch unser Greifer von vier Motoren bewegt. Damit kann er seine Hand überall in seinem Arbeitsbereich hinbewegen, die Ausrichtung der Hand bleibt dabei aber stets parallel zur Grundfläche und zeigt vom Roboter weg nach außen. Die meisten heute in der Industrie eingesetzten Roboterarme können auch die Ausrichtung der Hand frei bestimmen und setzen dafür zusätzliche Motoren ein.

Dass die Greifhand parallel zur Grundfläche bleibt, leistet die deutlich hervortretende, gelbe Verstrebung. Sie besteht aus zwei Parallelogrammen (Abb. 6–6). Das linke Parallelogramm sorgt dafür, dass die kurzen Streben oben im Ellenbogengelenk senkrecht stehen. Das rechte Parallelogramm bewirkt dann, dass der Anknüpfungspunkt der Streben an der Greifhand stets über der Drehachse der Hand liegt. Damit bleibt die Hand horizontal.

Zum Abschluss des Kapitels lassen wir unseren Greifer die Lösung des Spiels »Die Türme von Hanoi« demonstrieren. Dieses Knobelspiel ist zu einem Paradebeispiel geworden, wenn es darum geht, das Prinzip der Rekursion zu erklären. Und genau dafür ist unserer Meinung nach die Kombination aus einer klaren und direkten Umsetzung des Verfahrens in C und der konkreten, sichtbaren Demonstration durch den fischertechnik-Roboter ideal.

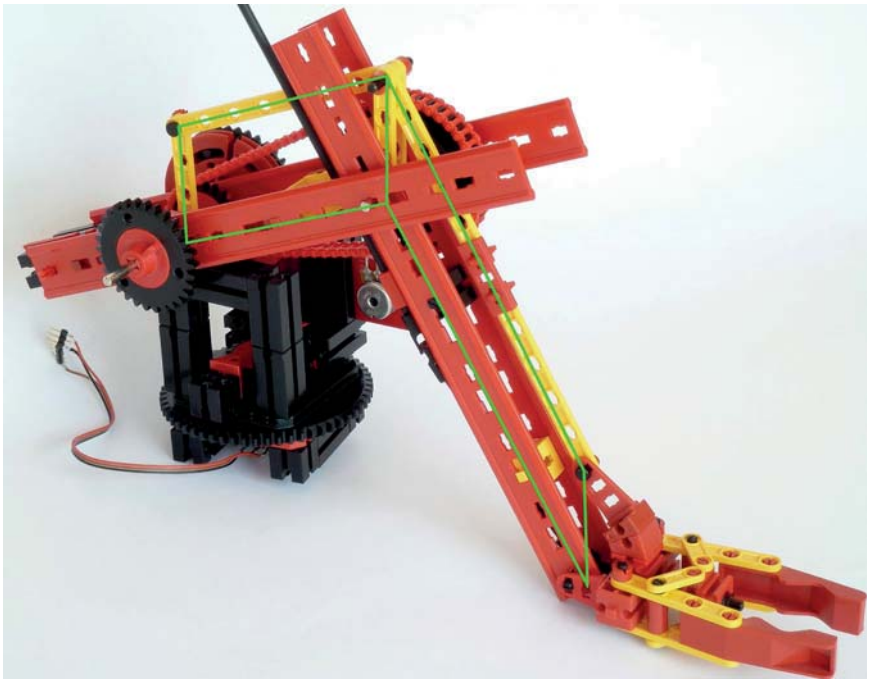


Abb. 6–6 Parallelführung der Greifhand

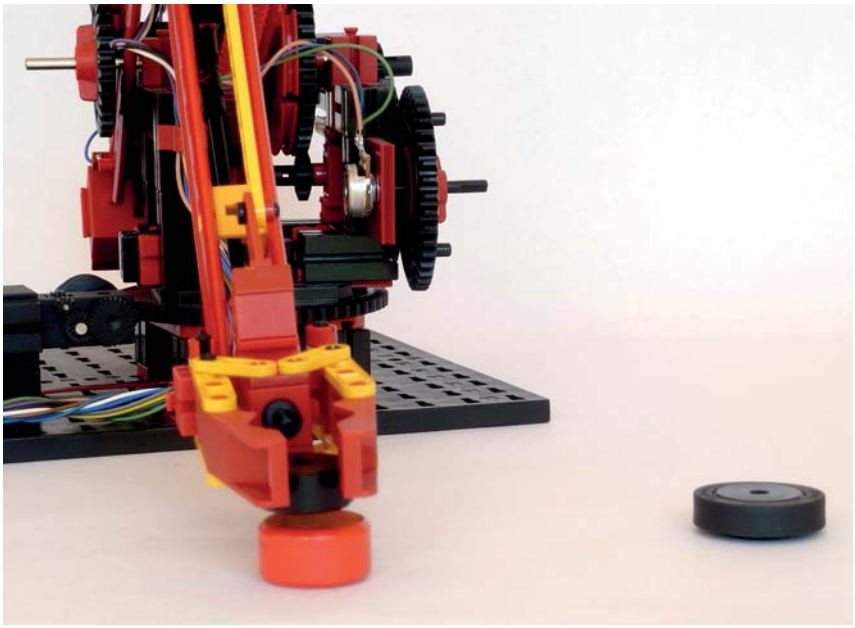


Abb. 6–7 Unser Greifer stapelt den Turm von Hanoi um.

Dank des einfachen Teach-in-Systems, der hohen Positioniergenauigkeit und der Fähigkeit, verschieden große Objekte zu greifen, kann der Roboter schnell für ganz andere Zwecke eingesetzt werden. Sei es, um Werkstücke zwischen Förderbändern zu bewegen, Fahrzeuge zu beladen oder gar Mühle zu spielen. Wie beim originalen fischertechnik-Trainingsroboter gibt es leicht umzusetzende Alternativen zur Greifhand: ein Elektromagnet oder ein Vakuumsauger, die wir beide im nächsten Kapitel vorstellen und einsetzen, oder auch ein einfacher Aufsatz zum Drehen von S-Riegeln. Der Fantasie sind hier keine Grenzen gesetzt.

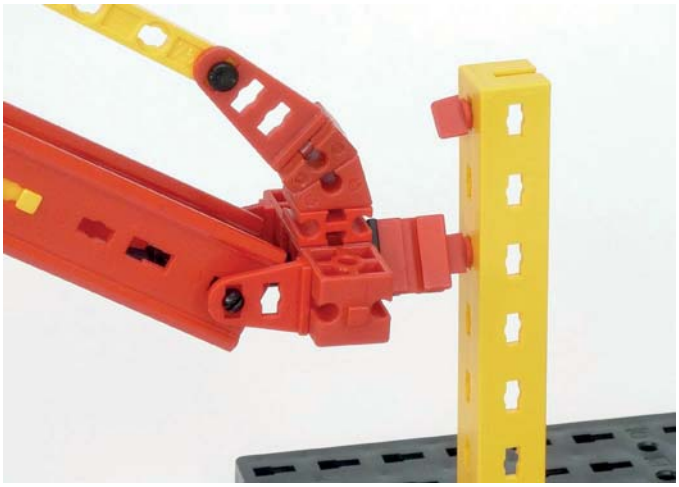


Abb. 6–8 Alternativer Aufsatz zum Schrauben von S-Riegeln

6.2 Steuerung mit Potenziometern

In diesem Abschnitt stellen wir die Methode vor, mit der wir die Gelenke unseres Greifers in diesem Kapitel und des Delta-Roboters im folgenden Kapitel ansteuern. Jede Drehachse ist mit einem Potenziometer ausgestattet. Einer der Außenkontakte wird mit Masse (GND), der andere mit 5 V verbunden. Das Potenziometer fungiert dadurch als Spannungsteiler. Die Spannung am mittleren Kontakt schwankt zwischen 0 und 5 V und ist näherungsweise proportional zum Drehwinkel. Dieser mittlere Kontakt wird mit einem analogen Eingang des Arduino verbunden. In unseren Sketchen können wir die Spannung an diesem Eingang messen. Wir erhalten durch einen Funktionsaufruf einen Wert zwischen 0 und 1023, den wir im Folgenden Potenziometerwert nennen.

Soll das Gelenk eine Bewegung ausführen, so geben wir einen Sollwert vor. Die Drehachse wird mit einem Motor so lange vor- oder rückwärts bewegt, bis

der Sollwert bis auf einen einstellbaren Fehler mit dem aktuellen Potenziometerwert (Istwert) übereinstimmt.

Diese Methode ist naheliegend, einfach und führt zu kurzen, übersichtlichen und verständlichen Programmen.

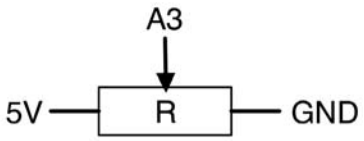


Abb. 6–9 Potenziometer als Spannungsteiler

Aufbau und Motorisierung

Wir verwenden hier die weitverbreiteten linearen Potenziometer PC16BU mit 10 kΩ der Firma OMEG. Die Achsen dieser Potenziometer haben genau wie die fischertechnik-Achsen einen Durchmesser von 4 mm und fügen sich damit problemlos ins fischertechnik-System ein.

Im aktuellen fischertechnik-Sortiment gibt es ebenfalls Potenziometer mit 4-mm-Achsen (32235). Diese originalen Potenziometer können auch verwendet werden; sie sind allerdings deutlich teurer.

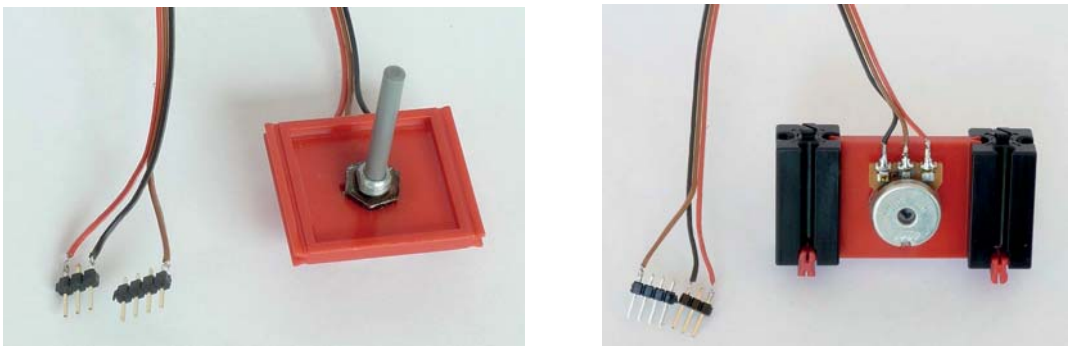


Abb. 6–10 OMEG-Potenziometer im Flachstein (32116)

Dass es Potenziometer im aktuellen fischertechnik-Sortiment gibt, hat den großen Vorteil, dass auch eine gebrauchsfertige Befestigungsmöglichkeit für Potenziometer zur Verfügung steht, nämlich der Flachstein (32116).

Wie in Abb. 6–10 dargestellt, stecken wir das Poti in den Flachstein und schrauben es mit der Mutter fest. Anschließend löten wir ein 14 cm langes, dreiadriges Kabel an die Kontakte des Potenziometers. Die anderen Enden der Adern werden an eine vierpolige und an eine dreipolige Stiftleiste mit 2,54-mm-Raster gelötet (*Male Pin Headers*). In Abb. 6–10 ist der linke Kontakt des Potis über die schwarze Ader mit Masse (GND), der rechte Kontakt über die rote Ader mit 5 V und der mittlere Kontakt über die braune Ader mit Pin A3 verbunden.

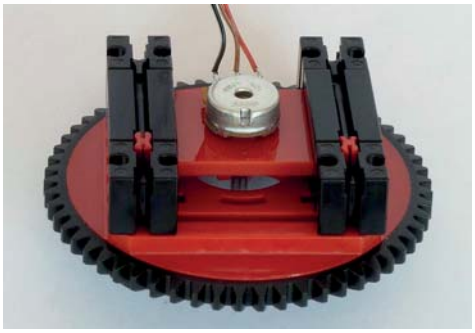


Abb. 6–11 Einbau des Potenziometers in den Drehkranz

In die Eckpositionen der roten Nuten des Drehkranzes schieben wir nun je einen BS 15. Anschließend stecken wir den Flachstein mit dem Potenziometer und den BS 30 ein.

Damit sich die Potiachse starr mit dem schwarzen Drehkranzoberteil dreht, schrauben wir auf ihr ein Zahnrad Z20 mit einer Nabe fest und sichern es durch zwei gegenüberliegende Sperrkeile. Für den Dauereinsatz sollten wir die Sperrkeile etwas stärker als abgebildet befestigen.

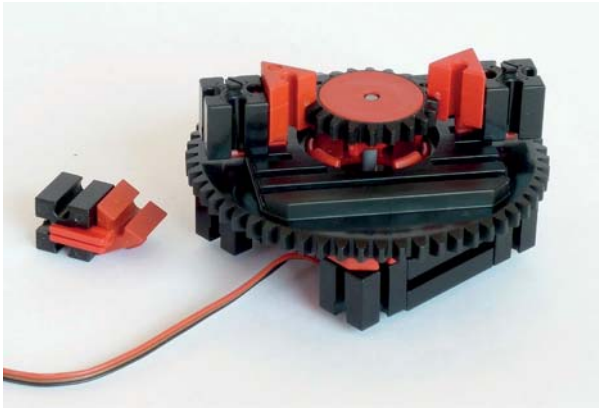


Abb. 6–12 Das drehbare Gelenk mit Potenziometer

Abb. 6–13 Der ein- und auskoppelbare Antrieb des Gelenks

In eine Grundplatte 500 schieben wir nun in die Positionen D3, G3, D6 und G6 je einen roten BS 5 mit zwei Zapfen und in die Positionen K3 und L3 je einen BS 15 mit zwei Zapfen. Auf den vier BS 5 befestigen wir unseren Drehkranz mit dem Potenziometer und auf den zwei BS 15 einen S-Motor mit U-Getriebe und Rastschnecke (35072). Der Motor kann durch Verschieben einfach und schnell ein- und ausgekoppelt werden.

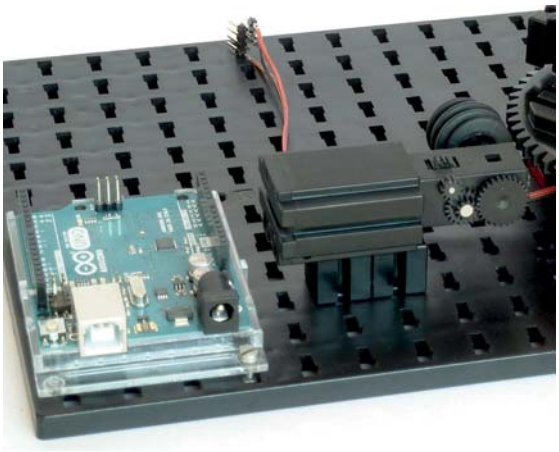


Abb. 6–14 Die Befestigung des Arduino Uno auf der Grundplatte

Anschluss der Elektronik

Ein originaler Arduino Uno R3 wird mit einer Grundplatte aus Plexiglas geliefert, die sich einfach mit einer M3-Schraube und einer passenden Mutter neben dem S-Motor auf der Grundplatte anschrauben lässt. Wer das Sunfounder-Gehäuse verwendet (siehe Kapitel 1), kann die Zapfen der angeklebten Bauplatten 30 × 45 in die Grundplatte schieben.

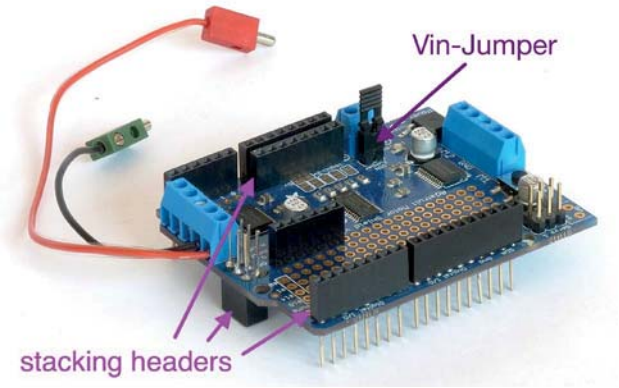


Abb. 6–15 Adafruit Motor Shield v2.3

Das Motor Shield v2.3 von Adafruit sollte man komplett mit sogenannten *Stacking Headers* versehen. Ein passendes Set gibt es in der Regel dort, wo man auch das Motor Shield erwerben kann. Durch diese Stift-/Buchsenleisten kann man Kabel oder weitere Shields ebenso auf das Motor Shield stecken wie auf den Arduino selbst.

Den S-Motor schließen wir an den Motorausgang M4 des Shields an. Dabei muss die Polung beachtet werden. Bei uns ist das rote Kabel 14 cm und das schwarze 7 cm lang. Die Stiftleisten stecken wir so in die Buchsenleisten des Shields, dass die rote Ader mit 5 V, die schwarze mit Masse (GND) und die braune mit Pin A3 verbunden ist.

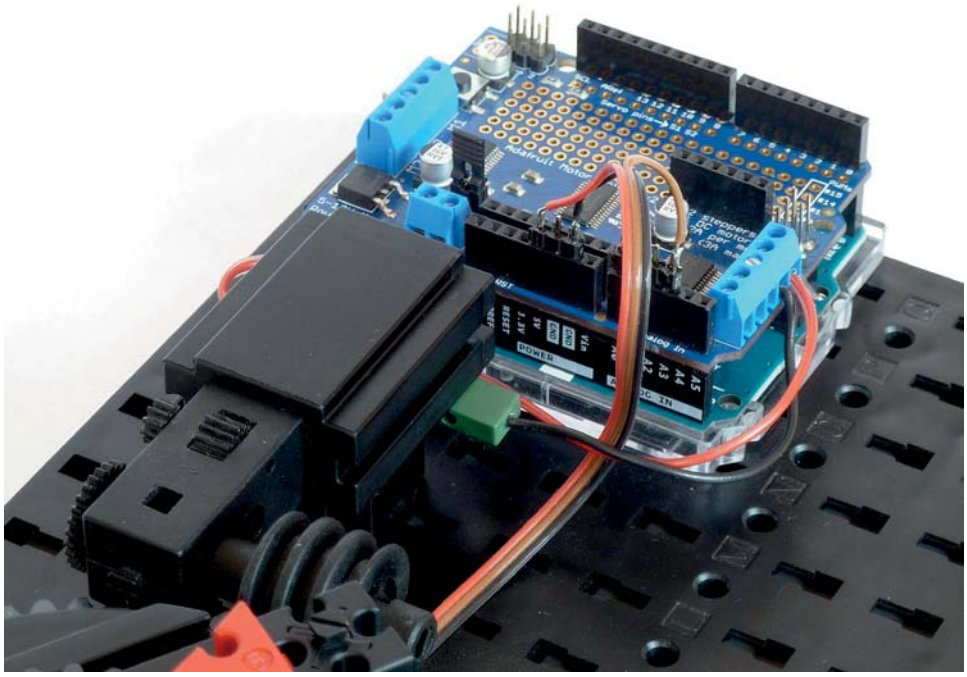


Abb. 6–16 Anschluss des Motors und des Potenziometers

Abb. 6–17 Adapter von 3,5-mm-Klinke auf 5,5-mm-Klinke

Zum Betrieb der Motoren benötigt das Motor Shield eine Gleichspannung von 9 V. Wer ein aktuelles fischertechnik-Steckernetzteil (505287) besitzt, kann dieses über einen Adapter von 3,5-mm-Klinke auf 5,5-mm-Klinke wie in Abb. 6–17 an den Arduino anschließen. Wichtig ist, dass der »Vin-Jumper« auf das Motor Shield gesteckt wird, dann versorgt das Steckernetzteil Arduino und Shield gemeinsam. Der Jumper wird zusammen mit dem Shield geliefert und ist in Abb. 6–15 deutlich zu erkennen. Statt des originalen fischertechnik-Netzteils können natürlich auch andere 9-V-Steckernetzteile verwendet werden. Zu beachten ist, dass an der Klinke innen Plus und außen Minus (GND) anliegt.

Schließlich verbinden wir den Arduino noch über ein USB-Kabel mit dem Computer.

Das Auslesen des Potis

In der Arduino-Entwicklungsumgebung stellen wir unter dem Reiter *Werkzeuge* das verwendete Arduino-Modell und den zugehörigen USB-Port ein. Wir öffnen unseren ersten Sketch *Drehkranz_Poti* von der Webseite zum Buch.

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.println(analogRead(3));
  delay(200);
}
```

Listing 6–1 Sketch Drehkranz_Poti

Die Funktion `setup` wird nur einmalig ausgeführt, und zwar nach dem Hochladen eines Sketches, nach dem Start des seriellen Monitors, nach einem Reset oder direkt nach dem Anlegen der Versorgungsspannung. In ihr wird eine serielle Verbindung zur Datenübertragung zwischen dem Arduino und dem Computer initialisiert. Die Zahl 115200 gibt die Übertragungsrate in Bit/s an.

Die Funktion `loop` dagegen wird anschließend fortlaufend ausgeführt. In ihr wird zunächst mittels `analogRead(3)` der Potenziometerwert gemessen. Das Ergebnis ist ein Wert zwischen 0 und 1023, der durch die Funktion `Serial.println` über die serielle Verbindung zum Computer übermittelt wird. Anschließend wird 200 ms gewartet, und so geht es weiter: Auslesen und Übertragen erfolgen im Wechsel.

Durch Klicken auf den kreisförmigen Button mit dem Pfeil nach rechts in der Arduino-IDE wird der Sketch übersetzt und auf den Arduino hochgeladen.

Um die Messergebnisse sehen zu können, starten wir den seriellen Monitor der Arduino-IDE, indem wir auf den Button mit der Lupe oben rechts im Fenster klicken. Es öffnet sich ein Fenster wie in Abb. 6–18.

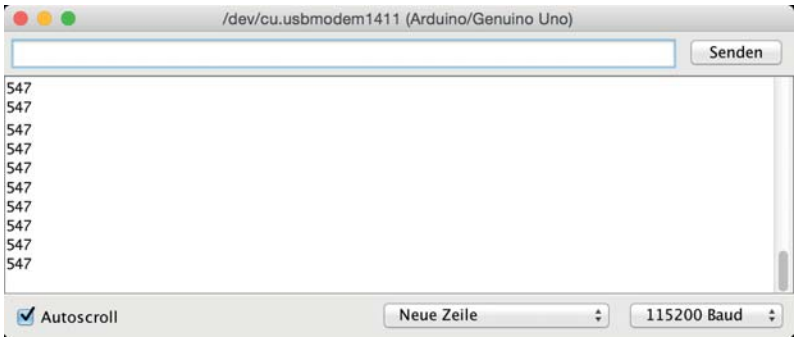


Abb. 6–18 Die gemessenen Potenziometerwerte werden auf dem seriellen Monitor angezeigt.

Drehen wir nun bei ausgekoppeltem Motor das Drehkranzoberteil im Uhrzeigersinn, so werden die angezeigten Werte größer. Sobald sich die Werte dem Maximalwert 1023 nähern, sollte man einmal vorsichtig bis an den Anschlag des Potenziometers weiterdrehen. Jetzt ist eine gute Gelegenheit, den Drehbereich des Oberteils durch Lösen der Sperrkeile anzupassen. Die Potis erlauben einen Drehwinkel von 270°. Drehen wir das Oberteil gegen den Uhrzeigersinn, so werden die Werte wieder kleiner. Auch hier sollte man das Potenziometer einmal vorsichtig bis an den Anschlag drehen.

Die Motorsteuerung

Wir sind jetzt bereit für die Motorsteuerung des Drehkranzes und laden dazu den Sketch *Drehkranz_Motor* von der Webseite zum Buch. Bevor wir ihn erklären, probieren wir ihn erst einmal aus. Dazu übersetzen wir ihn, laden ihn auf den Arduino hoch und starten den seriellen Monitor (Abb. 6–19).

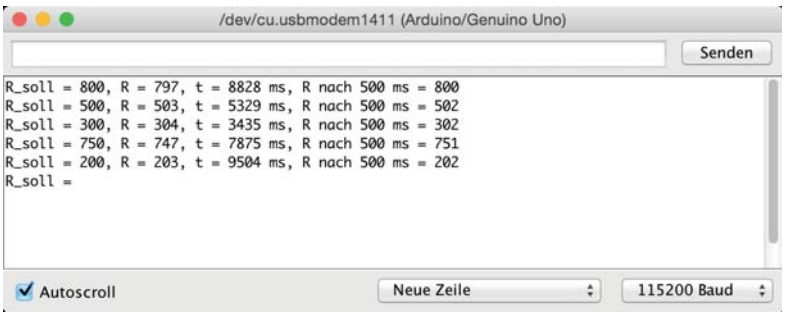


Abb. 6–19 Die Steuerung des Drehkranzes im seriellen Monitor

Nach der Eingabe eines Sollwertes bewegt sich der Drehkranz und kommt dann zum Stoppen. Im seriellen Monitor wird der Potenziometerwert angezeigt, kurz nachdem der Motor gestoppt wurde, sowie die Dauer der Bewegung. Dann wird 500 ms gewartet und der Potenziometerwert noch einmal ausgelesen. Dieser Wert dient zur Kontrolle, ob das unvermeidbare kurze Weiterlaufen des Motors nach dem Abschalten den Drehkranz über den gewünschten Sollwert hinaus dreht. Da wir eine Schnecke zum Antrieb des Drehkranzes verwenden, ist dies nicht der Fall.

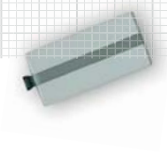
Das Protokoll zeigt, dass der Sollwert durch unseren Sketch bis auf ±2 genau angefahren wird. Da unser Poti einen Drehbereich von 270° besitzt und wir 1024 Potenziometerwerte haben, erhalten wir eine Wiederholgenauigkeit von unter ±1°, die für unsere Zwecke vollkommen ausreichend ist. Wiederholgenauigkeit bedeutet hierbei, dass eine einmal angefahrene Position bei einem erneuten Anfahren mit dem gleichen Sollwert wieder erreicht wird. Es geht hier noch nicht darum, dass wir den gemessenen Potenziometerwert in den Drehwinkel des Drehkranzes umrechnen. Dabei müssten wir die kleine, aber für viele Zwecke nicht vernachlässigbare Nichtlinearität des Potis und des A/D-Wandlers im Arduino berücksichtigen. Darauf werden wir in Kapitel 7 genauer eingehen.

Schauen wir uns nun den Sketch an. In den ersten Zeilen sorgen die `#include`-Anweisungen dafür, dass die Bibliotheken eingebunden werden, die für die Kommunikation zwischen Arduino und Motor Shield benötigt werden:

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>

Adafruit_MotorShield MotorShield = Adafruit_MotorShield();
Adafruit_DCMotor *M = MotorShield.getMotor(4);
```

Listing 6–2 Einbindung der Bibliothek MotorShield

Die dritte und vierte Zeile erzeugen die Objekte `MotorShield` und `M`. Wer mit objektorientierter Programmierung und dem Zeigerkonzept in C nicht vertraut ist, braucht hier keine Angst zu bekommen: Während die beiden Zeilen auf den Neuling durch das häufige Vorkommen des Wortes Motor Shield kryptisch bis unsinnig wirken können, ist die Anwendung der erzeugten Objekte und ihrer Methoden im weiteren Verlauf des Sketches intuitiver. Das Argument 4 des Methodenaufrufs `MotorShield.getMotor(4)` bedeutet, dass der Motor `M` am Motorausgang M4 des Motor Shield angeschlossen ist.



In der einmalig ausgeführten Funktion

```
void setup() {
  Serial.begin(115200);
  MotorShield.begin();
  M->setSpeed(0);
}
```

Listing 6–3 Initialisierung des Motor Shield

werden die serielle Schnittstelle und das Motor Shield initialisiert sowie die Geschwindigkeit des Motors `M` auf 0 gesetzt.

Wir schauen uns jetzt die fortlaufend ausgeführte Funktion `loop()` genauer an:

```
void loop() {
  Serial.print("R_soll = ");
  while (Serial.available() == 0);
  int R_soll = Serial.parseInt();
  Serial.read();
  Serial.print(R_soll);
```

Listing 6–4 Einlesen des Potenziometer-Sollwertes

Im ersten Teil wird zunächst der Potenziometer-Sollwert über den seriellen Monitor eingelesen. Dazu ist es wichtig, dass sich der serielle Monitor im Modus *Neue Zeile* befindet (Abb. 6–18). Was immer man in die Eingabezeile des seriellen Monitors tippt, wird erst in den seriellen Puffer des Computers übertragen, wenn die Eingabetaste gedrückt wurde. Durch die Zeile

```
while (Serial.available() == 0);
```

wartet der Arduino so lange, bis neue Zeichen im Puffer des Computers vorhanden sind. Die Funktion `Serial.parseInt` versucht dann, möglichst viele Zeichen im Puffer in eine ganze Zahl umzuwandeln. Das erste Zeichen, das nicht ins Muster passt, wird als Trennzeichen interpretiert. In unserem Fall wird es das Newline-Zeichen sein, das vom Drücken der Eingabetaste stammt. Dieses Zeichen selbst muss nun noch durch `Serial.read();` aus dem Puffer entfernt werden.

Mit dem nächsten Teil der Funktion `loop()` wird durch den Funktionsaufruf in der mittleren Zeile der Drehkranz gedreht:

```
unsigned long start = millis();
drehe(R_soll);
unsigned long dauer = millis() - start;
```

Listing 6–5 Drehen des Drehkranzes

Das Argument gibt den Sollwert an, der am Ende der Drehung ausgelesen werden soll. Die erste und die letzte Zeile dienen zur Zeitmessung: Die Funktion `millis()` liefert die Millisekunden zurück, die seit dem Start des Sketches vergangen sind. Der Datentyp ist `unsigned long`, also eine vorzeichenlose 32-Bit-Zahl.

Im letzten Teil der Funktion `loop ()` wird der Potenziometerwert angezeigt, kurz nachdem der Motor gestoppt wurde, die Dauer der Bewegung und der Potenziometerwert nach 500 ms:

```
  Serial.print(", R = "); Serial.print(analogRead(3));
  Serial.print(", t = "); Serial.print(dauer);
  Serial.print(" ms, ");
  delay(500);
  Serial.print("R nach 500 ms = "); Serial.println(analogRead(3));
}
```

Listing 6–6 Anzeige der Potenziometerwerte und der Bewegungsdauer

Sehen wir uns jetzt die entscheidende Funktion `drehe()` an:

```
void drehe(int R_soll) {

  int R;
  int toleranz = 3;
  boolean aktiv = true;

  while ( aktiv ) {
    R = analogRead(3);
    if (R > R_soll + toleranz) {
      M->setSpeed(255);
      M->run(BACKWARD);
    }
    else if (R < R_soll - toleranz) {
      M->setSpeed(255);
      M->run(FORWARD);
    }
    else {
      M->setSpeed(0);
      aktiv = false;
    }
  }
}
```

Listing 6–7 Funktion »drehe()«

Diese Funktion dreht den Drehkranz, bis der aktuelle Potenziometerwert bis auf eine einstellbare Toleranz mit dem übergebenen Sollwert übereinstimmt. Dazu werden zu Beginn die benötigten Variablen `R`, `toleranz` und `aktiv` deklariert. Obwohl die Variable `toleranz` vom Sketch nicht verändert wird, haben wir sie nicht als Konstante oder statische Konstante deklariert, da wir uns nicht darauf festlegen wollen, dass sie in erweiterten Varianten der Funktion unveränderbar bleiben muss.

In der `while`-Schleife wird zunächst der aktuelle Potenziometerwert ermittelt und überprüft, ob er größer als der Sollwert plus der Toleranz ist. Ist das der Fall, wird der Motor mit maximaler Geschwindigkeit rückwärts laufen gelassen. Andernfalls wird überprüft, ob der aktuelle Potenziometerwert kleiner als der Sollwert minus der Toleranz ist. Falls ja, wird der Motor mit maximaler Geschwindigkeit 255 vorwärts laufen gelassen. Ansonsten liegt der aktuelle Potenziometerwert im Toleranzintervall um den Sollwert. Der Motor wird dann gebremst und die boolesche Variable `aktiv` wird auf `false` gesetzt. Dadurch wird die Schleife beendet.

Unser Drehkranz mit Potenziometer und der einfachen Funktion `drehe()` kann in vielen verschiedenen Anwendungskontexten eingesetzt werden. Wir verwenden ihn im Folgenden als Basis für unseren Greifer.

6.3 Der Aufbau des Greifers

Der Körper

Der Körper des Roboters wird in der Robotik gelegentlich auch *Karussell* genannt. Der Drehkranz aus dem vorigen Abschnitt bildet die Grundlage. Wir entfernen allerdings zunächst einen der beiden Sperrkeile, den anderen befestigen wir etwas stärker. In den Abb. 6–20 und 6–21 sieht man die ersten beiden Baustufen.

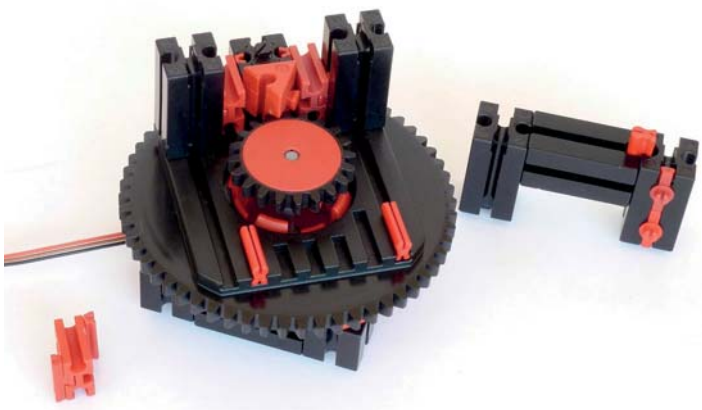


Abb. 6–20 Erste Baustufe des Roboter-Körpers

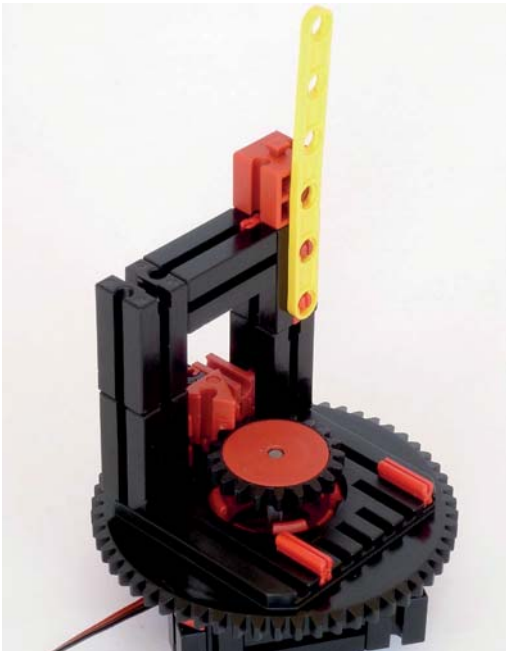


Abb. 6–21 Zweite Baustufe des Roboter-Körpers

Dem verbliebenen Sperrkeil gegenüber liegt ein Block mit den Lagern. Wir bauen ihn aus zwei Teilen zusammen.

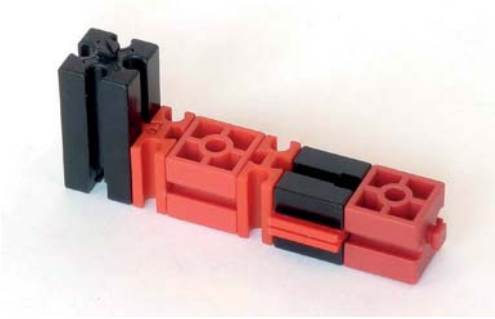


Abb. 6–22 Erster Teil des Lagerblocks

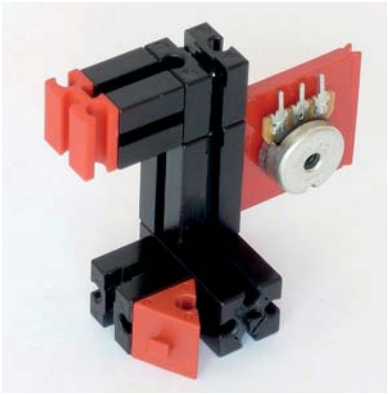


Abb. 6–23 Zweiter Teil des Lagerblocks