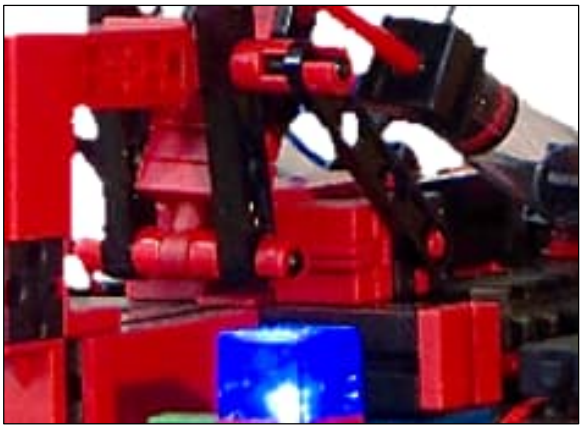
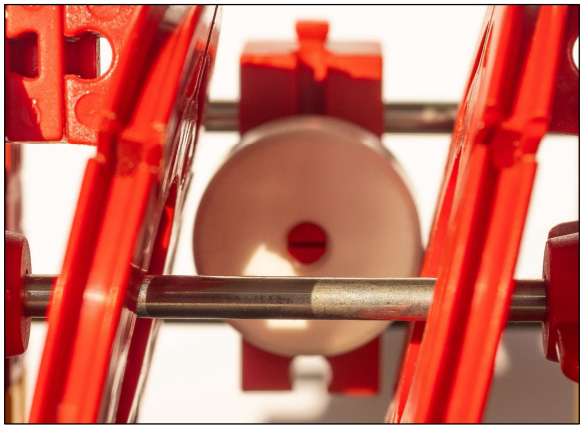
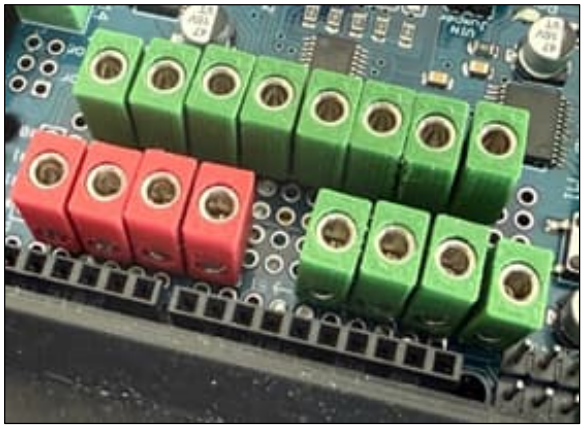
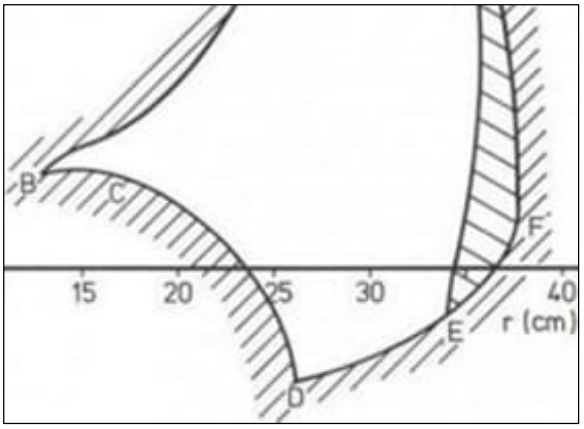
ft:pedia

Heft 2/2025









Herausgegeben von
Dirk Fox und Stefan Falk

ISSN 2192-5879

Editorial

Licht im Tunnel

Dirk Fox, Stefan Falk

Seit 2011 – dem Geburtsjahr der ft:pedia – untersucht das Institut der deutschen Wirtschaft im Auftrag von BDA, BDI, Gesamtmetall und der Initiative „MINT Zukunft schaffen“ die Entwicklung der MINT-Fachkräftelücke und der MINT-Studienabschlüsse in Deutschland und im internationalen Vergleich. Die Ergebnisse werden zweimal jährlich in einem „Frühjahrs-“ und einem „Herbstreport“ veröffentlicht.

Zweimal haben wir bereits in unseren Editorials daraus zitiert („Gefährdete Spezies“, 2/2012; „Es wird ernst“, 1/2022), denn sie belegen einen beängstigenden Trend: Uns gehen die Technik- und Informatik-Experten aus. Ende Mai ist der Frühjahrsreport 2025 erschienen – und zeigt ein Fünkchen Licht im Tunnel. Denn die Zahl der MINT-Akademiker ist in den zehn Jahren bis 2021 um fast 34% gestiegen; besonders groß ist der Zuwachs bei Frauen (+58%) und Zuwanderern (+75%).

Allerdings hat auch die Zahl der MINT-Experten über 54 Jahre überproportional zugenommen: um 63%. Ein großer Teil dieser Gruppe, die geburtenstarken Jahrgänge 1964/65, wird in wenigen Jahren in Rente gehen. Es wird herausfordernd sein, diese Gruppe von über einer Viertelmillion erfahrener MINT-Akademiker zu ersetzen.

Und noch eine Diskrepanz gibt zu denken: Die Zahl der Stellen für MINT-Akademiker hat im gleichen Zeitraum um 55% zugenommen. Trotz aller Erfolge klafft also immer noch eine erhebliche Bedarfslücke.

Die zahlreichen MINT-Initiativen zur Motivation junger Menschen, eine Technik- oder Informatik-Ausbildung zu beginnen, bleiben also wichtig.

Da inzwischen fast jeder dritte Arbeitsplatz in Deutschland ein MINT-Arbeitsplatz ist (Tendenz: steigend), ist jede Stunde mit fischertechnik nicht nur ein Glückserlebnis, sondern eine Investition in die Zukunft. Und wenn jemand einwendet, dass MINT-Arbeitsplätze doch sicher bald durch KIs ersetzt werden, dann setzt doch einmal ChatGPT neben euren Kindern (oder Enkeln) vor die Baukästen und beobachtet gelassen, was passiert.

Beste Grüße,
Euer ft:pedia-Team

P.S.: Am einfachsten erreicht ihr uns unter ftpedia@ftcommunity.de oder über die Rubrik *ft:pedia* im Forum der ft-Community.

	2011	2021	Veränderung in Prozent
MINT-Akademikerinnen und MINT-Akademiker insgesamt	2.366.400	3.165.300	+33,8
davon Frauen	477.300	754.700	+58,1
davon Ältere ab 55 Jahren	448.800	731.800	+63,1
davon Zuwanderinnen und Zuwanderer	368.600	644.900	+75,0

Aus: Institut der deutschen Wirtschaft (IW): MINT-Frühjahrsreport 2025, 21.05.2025

Inhalt

Termine

Was?	Wann?	Wo?
fischertechnik-Convention	20.09.2025	Bad Hersfeld (Martinskirche)
Clubtag De Bilt	18.10.2025	De Bilt, Kultur- und Konferenzzentrum HF Witte

Impressum

http://www.ftpedia.de

Herausgeber: Dirk Fox, Ettlinger Straße 12-14, 76137 Karlsruhe und Stefan Falk, Siemensstraße 20, 76275 Ettlingen

Autoren: Florian Bauer, Simon Bauer, Axel Chobe, Arnoud van Delden, Stefan Falk, Dirk Fox, Fabian Haas, Peter Habermehl, Robert Lippmann, Volker Paelke, Jeroen Regtien.

3D-Modellentwurf

Ein Besuch in den Skunk Works

Peter Habermehl

Skunk Works, Stinktierwerke, so lautet die offiziell-inoffizielle Bezeichnung einer geheimnisumwitterten Abteilung der US-amerikanischen Firma Lockheed Martin zur Entwicklung von exotischen Waffensystemen und Technologien von Übermorgen. Auch im fischertechnik-Universum gibt es solche Geheimfabriken.

Finanziert durch üppige Militärbudgets werden in den Skunk Works von Menschen, deren Intellekt irgendwo zwischen Genie und Wahnsinn anzusiedeln ist, Dinge erfunden, die hochgeheim die Defensivkraft westlicher Armeen stärken, um nur wenige Jahre später in Form von günstigen AliExpress-Produkten auf der ganzen Welt erhältlich zu sein, so wie das chinesische Teflonpfannen-Set. Selbst wenn die fischertechnik-Geheimfabriken deutlich kleiner und für die Verteidigung des Abendlandes völlig unbedeutend sind, entstehen hier Werke jenseits jeglicher Vorstellungskraft. Eine dieser Einrichtungen liegt im fensterlosen Tiefgeschoss eines unscheinbaren

Abb. 1: Riesen-Bonbonniere im Vergleich zu einem echten Baustein 15

Häuschens im Schatten einer brutalistisch-schlichten Sichtbetonkirche in einer Kleinstadt irgendwo in der Mitte Deutschlands. Unbemerkt von Umstehenden werden hier fischertechnik-Innovationen geschaffen, von denen niemals ein Mensch erfahren wird, und das ist auch gut so.

Auf dem Nachhauseweg von einem – natürlich streng geheimen – Einsatz konnte der in der Nut-und-Zapfen-Gemeinschaft nicht ganz unbekannte Geh-Heim-Agent F.X. Fischer einige der nicht mehr so ganz geheimen Produkte besichtigen, mit ihrem Schöpfer sprechen (der eindeutig mehr Wahnsinniger als Genie ist), und einen Blick hinter die sonst streng bewachten Türen der fischertechnik Skunk Works werfen.

Sein erstes Fazit war, dass der Name ob der durch diverse 3D-Drucker bedingten hohen Temperaturen und der fehlenden Belüftung absolut berechtigt ist. Aber er zeigte sich auch begeistert von allem, was er dort sehen durfte. Und er erfuhr, dass ein fischertechnik-Fan, der nicht namentlich genannt werden möchte, die Entwicklung einer repräsentativen Bonbonniere in Auftrag gegeben hat. Diese wurde elegant als vergrößerter Baustein 15 mit abnehmbarem Deckel umgesetzt (Abb. 1).

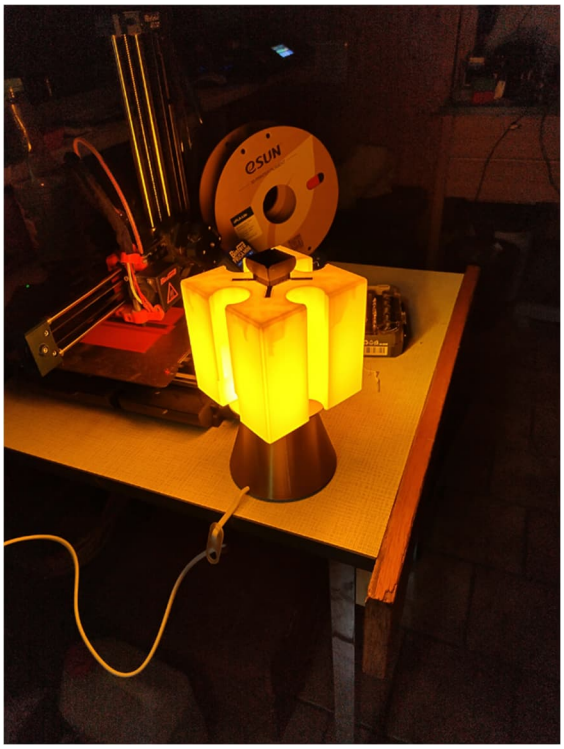


Abb. 2: Riesen-Nachttischleuchte

Fotos der Bonbonniere wurden über das Internet verbreitet, und so trat der berüch-

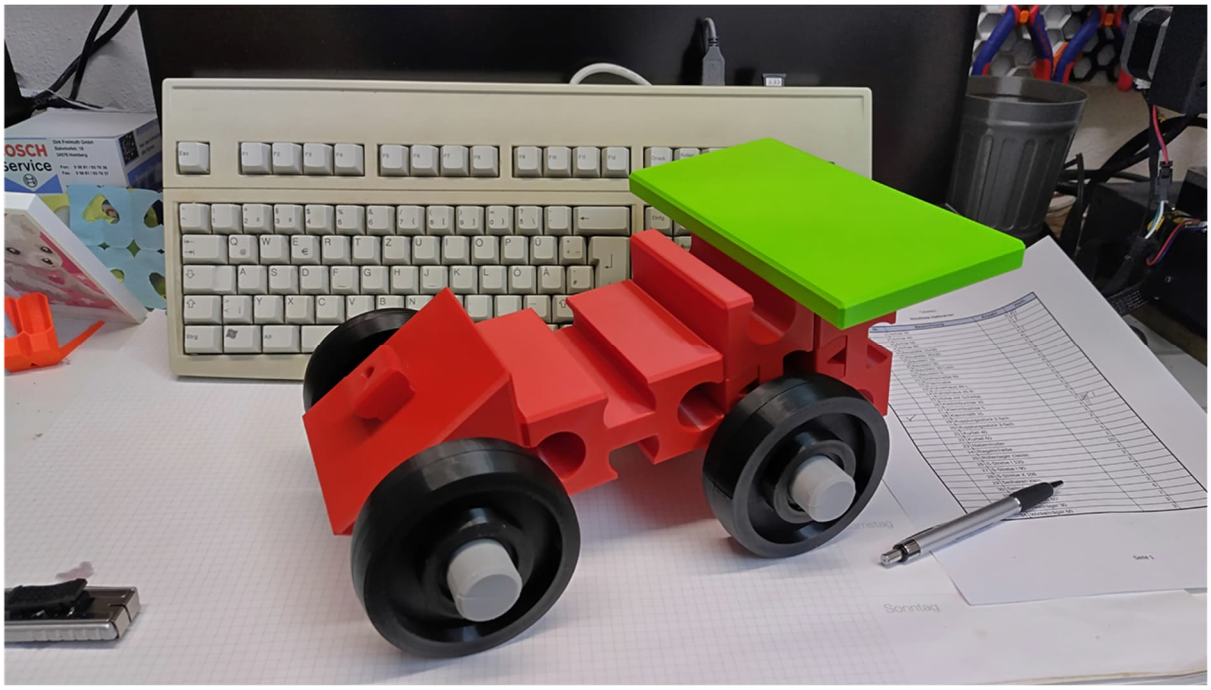


Abb. 3: Riesen-Giveaway

tigte Großmodellbauer Detlef O. auf den Plan, der sich schon lange gewünscht hat, auf Reisen ein exklusives fischertechnik-Artefakt als Manifest seines Hobbies bei sich haben zu können. Er ließ auf Basis der Bonbonniere ein Nachtlicht fertigen (Abb. 2), das ihn fortan in Hotelzimmern an allen fischertechnik-Ausstellungsorten glücklich einschlafen und aufwachen ließ.

In unserem wahnsinnigen Genie aber reifte ein Gedanke. Wenn schon dieser BS15 solches Aufsehen erregt, könnte man dann nicht auch einen BS30 vergrößern? Mit Loch womöglich gar?

Ja! In einer Versuchsreihe, die selbst den seligen Frank N. Furter in Angst und Schrecken versetzt hätte, entstanden riesige Winkelsteine 30 und 60, ein Baustein 7,5 und eine Bauplatte 15×30. Auch schwarze Teile fanden sich bald ein.

Der nimmermüde fischertechnik-Diplomat Chrischan W. aus L., der in diesem Zusammenhang ebenfalls unerkannt bleiben möchte, fügte gedanklich fix das Teilesammelsurium zusammen, und es entstand das wohl bislang größte Mini-Giveaway der fischertechnik Geschichte (siehe Abb. 3). Es ziert nun seinen Greifautomaten, der auf Treffen der fischertechnik-Community immer wieder Groß und Klein begeistert.

Im Stinktierkeller, Verzeihung, im Geheimlabor aber wurde weiter geforscht. Ein leitender Mitarbeiter, von dem allgemein angenommen wurde, er würde, wenn er des Morgens im kleinkarierten Jackett und akkurat faltengebügelten Jeans sein Heim verließ, seinen vorgeblichen Arbeitsplatz im Kellerarchiv des städtischen Bauamts aufsuchen, hatte den Gedanken, seine langjährige, treuselige und ahnungslose Ehefrau zum Valentinstag mit einer prächtigen Grünpflanze und einem dazu passenden Beistelltischchen zu überraschen. Die gelungene Umsetzung dieser Idee wurde in den Skunk Works gemeinhin als großer Hit betrachtet (Abb. 4).

Seit dem Nachmittag des 14.02.2025 haben die Skunk Works übrigens die Stelle des

Abb. 4: Riesen-Blumentisch

Abb. 5: Riesen-Hommage an Artur Fischer

Entwicklungsleiters vakant, da dieser von der Geschenkübergabe aus unbekannten Gründen nicht zurückkehrte. Aber die grundsätzliche Idee war nun mal in der Welt, und so war der Weg zu weiteren dekorativen Einzelstücken nicht weit. Im Umfeld des vorgenannten fischertechnik-Handlungsreisenden Chrischan W. wurden auf dem Fan Club Tag in Waldachtal am 17.05.2025 überdimensionale Flachsteine 30 mit dem Portrait Artur Fischers und einer Widmung zum 60-jährigen fischertechnik-Jubiläum gesichtet.

Auch das Logo der dieses Jahr im September geplanten Convention, die regenbogenbunte Winkelbausteinbrücke mit dem Motto „fischertechnik baut Brücken“ war zu sehen, bewacht von einer grandiosen fischertechnik-Riesenfigur aus den Werkstätten unseres niederländischen fischertechnik-Wizzards Arnoud. In Abb. 5 ist das alles zu sehen.

Und wenn man genau hinschaut – liegen da Klemmbuchsen 5? Eine riesige Achse 30? Und ganz rechts, am Bildrand, in blau, könnte das etwa ein Seilhaken sein? Rätsel über Rätsel...

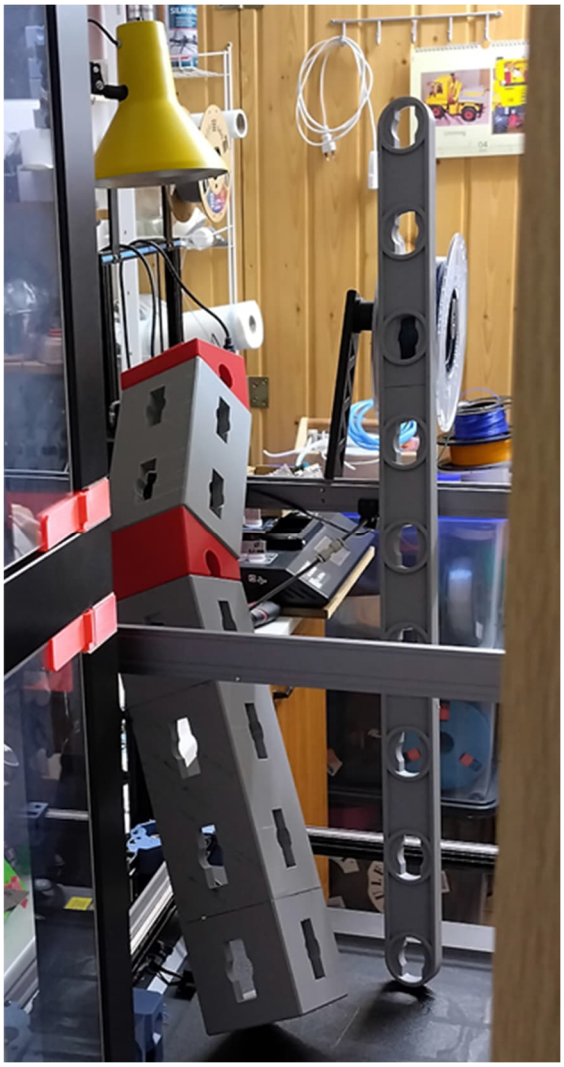


Abb. 6: Riesen-Ausblick

Beim Verlassen der Skunk Works gelang es F.X. dann noch, aus dem Handgelenk mit einer im Boden seiner Schwarzwälder-Kirschwasser-Taschenflasche unsichtbar verborgenen Mikrokamera ein Foto durch den Türspalt des Labors zu schießen – siehe Abb. 6.

Bei der Auswertung der Aufnahme stockte ihm der Atem. War das möglich? Ein Bauteil aus Winkelträger 30 und 60, deutlich über einen halben Meter groß? Und daneben eine Statikstrebe ähnlicher Dimension? Was zum Teufel geht hier vor sich? Sollte fischertechnik zum sechzigsten Jubiläum doch noch eine würdige Ehrung erfahren? Ein Denkmal gar, das in dieser Art zuvor noch nicht gesehen wurde?

Und was hat der zwielichtige Karohemdenträger mit dem MINTronics-Logo auf dem Rücken damit zu tun, der hier überall herumschnüffelt?

Es ist die Rede von einer September-Convention – bis dahin bleibt es wohl spannend.

Modell

Ein Panther in fischertechnik

Volker Paelke

Der Panther des österreichischen Herstellers Rosenbauer ist ein Feuerlöschfahrzeug für den Einsatz auf Flughäfen. Es zeichnet sich durch eine Kombination aus hoher Fahrleistung, großer Löschmittelkapazität und robuster Geländegängigkeit aus. Er ist in verschiedenen Varianten erhältlich mit 4×4-, 6×6- oder 8×8-Antrieb, um möglichst flexibel unterschiedliche Anforderungen und Einsatzszenarien abzudecken.

Das Original: Der Rosenbauer Panther

Eine der auffälligsten Komponenten ist der sogenannte HRET-Löscharm (High Reach Extendable Turret). Dieser Teleskoparm kann ausgefahren werden und ermöglicht das gezielte Eindringen in höhergelegene Flugzeugbereiche, etwa den Rumpf, um Brände direkt an der Quelle zu bekämpfen. Der HRET ist im Original mit einer Kamera und einem Durchstich-Wasserwerfer ausgestattet. Abb. 1 zeigt einen Panther 8×8 mit HRET der Flughafenfeuerwehr Bremen.

Das Modell im Überblick

Die Idee zum Modell entstand aus der Überlegung, die Möglichkeiten des fischertechnik TXT-Controllers in Verbindung mit Pneumatik-Komponenten einmal möglichst umfassend auszuschöpfen. Insbesondere Funktionen wie die USB-Kameraanbindung und die Soundausgabe, die in vielen Projekten oft ungenutzt bleiben, sollten gezielt zum Einsatz kommen.

Als Vorbild diente dabei das Löschfahrzeug Rosenbauer Panther. Aus praktischen Gründen fiel die Wahl auf die 6×6-Version, wobei aktuell nur die vier Hinterräder ange-

Abb. 1: Der Panther 8×8 mit HRET (Bild: Wikipedia © Geogast, zugeschnittene Version, Lizenz: CC BY-SA 4.0)

trieben werden. Das Herzstück des Modells ist der bewegliche HRET-Löscharm, an dessen Spitze die USB-Kamera des TXT montiert ist. Die Kamera liefert als Live-Stream ein aktuelles Videobild aus der Sicht des Fahrzeugs, wobei sich Ausrichtung und Höhe des Arms pneumatisch und motorisch verstellen lassen. Damit können – ähnlich wie beim Original – schwer zugängliche Stellen gezielt angesteuert werden (Abb. 2).

Aufbau

Die Wahl fiel auf einen Maßstab von etwa 1:25. Damit können die fischertechnik-Reifen 50 verwendet werden und das Modell ist groß genug, um die Technik problemlos aufzunehmen, aber noch handhabbar. Die Maße betragen ca. 46 cm Länge (mit ausgeklapptem Bug-Wasserwerfer), 13,5 cm Breite und ca. 15 cm Höhe (ohne HRET). Das Gewicht liegt bei etwa 2,3 kg (ohne Akku).

Das Modell ist weitestgehend modular aufgebaut (Abb. 3):

Die Basis bildet ein Leiterrahmen für das Fahrgestell. Zwei Alu-Profile 225 sorgen für Stabilität. Die beiden Hinterachsen werden über einen Kardan von einem Powermotor 20:1 angetrieben und verfügen über ein Differential.

Abb. 2: Das Modell in Aktion mit ausgefahrenem HRET

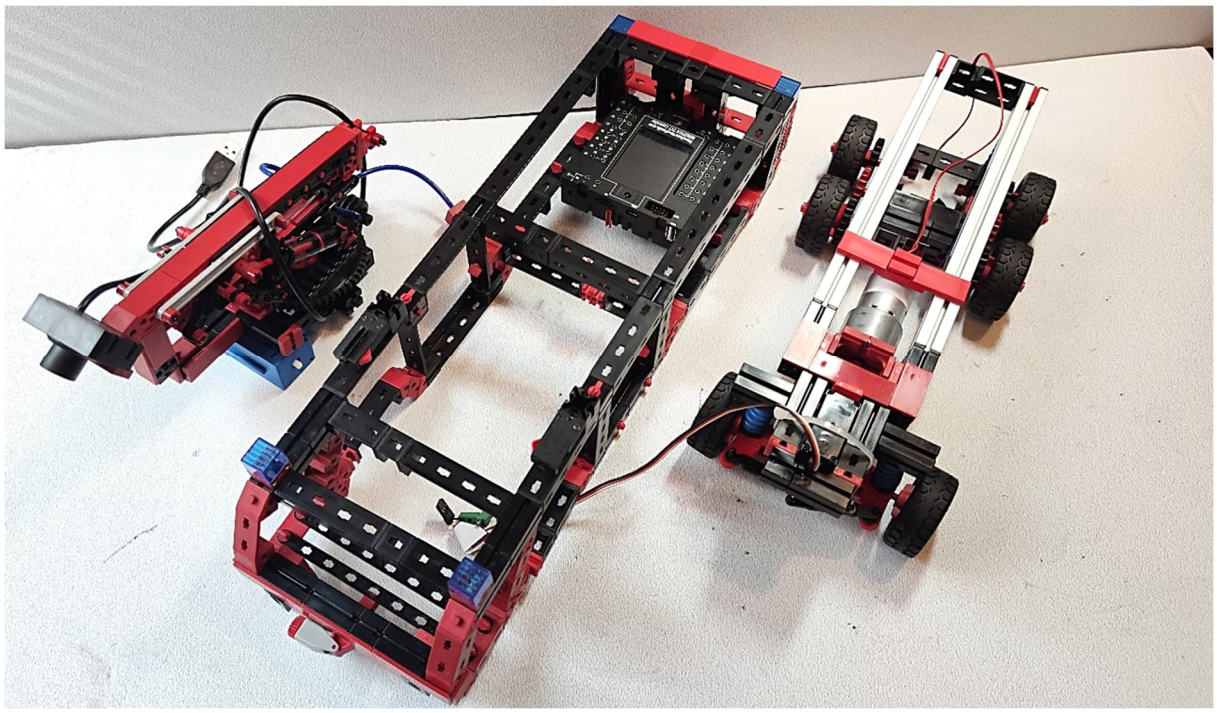


Abb. 3: Modularer Aufbau

Ein Abtrieb nach vorne für einen Allradantrieb ist vorhanden, wird jedoch bislang nicht genutzt. Da der fischertechnik-Rastkardan bei „schwerem Gelände“ an seine Grenzen stößt, wurde er durch einen Stahlkardan aus dem Modellbau ersetzt.

Auf dem Leiterrahmen sitzt die Karosserie mit einer vorbildähnlichen Form, die als Fachwerkrahmen im Wesentlichen aus Statikteilen besteht, um Stabilität bei (relativ) geringem Gewicht zu gewährleisten. In den Rahmen sind die Beleuchtung (Scheinwerfer, Rücklichter, Rückfahrscheinwerfer und Blaulichter vorne und hinten) sowie die Endschalter für den HRET-Arm integriert. Der TXT-Controller und die übrige Elektronik sind mit S-Riegeln und Statikstreben im Rahmen befestigt, was einen einfachen Ein- und Ausbau ermöglicht und gleichzeitig den Rahmen versteift. Der Controller befindet sich im hinteren Teil der Karosserie und ist dort leicht zugänglich.

Der HRET ist ein weiteres Modul, das in den vorderen Teil der Karosserie eingesetzt wird. Die HRET-Pneumatik mit Kompressor und Magnetventilen sowie ein Mini-Motor für das Schwenken des Arms befinden sich in der Karosserie, während der eigentliche Arm auf dem Dach geschwenkt und pneumatisch hoch- und runtergefahren werden kann.

Um einen möglichst vorbildähnlichen Look zu erreichen, wurde das Karosserie-Fachwerk noch mit einer Verkleidung versehen. Die optische Gestaltung orientiert sich an den Panther-Löschfahrzeugen des Flughafens Bremen, die mit ihrer Farbgebung gut zu den gängigen fischertechnik-Farben passen. Allerdings handelt es sich bei den echten Bremer Panthern um die 8×8-Version. Die Paneele der Verkleidung wurden in einem Vektorgrafik-Programm gezeichnet, dann auf Fotopapier gedruckt und mit Strebenadaptern und S-Riegeln an der Karosserie befestigt. Ursprünglich nur als Provisorium gedacht, haben sich die Fotopapier-Paneele als erstaunlich robust und langlebig erwiesen und bereits mehrere Nordconventions überstanden.

Funktionen

Folgende Funktionen sind integriert (Abb. 4):

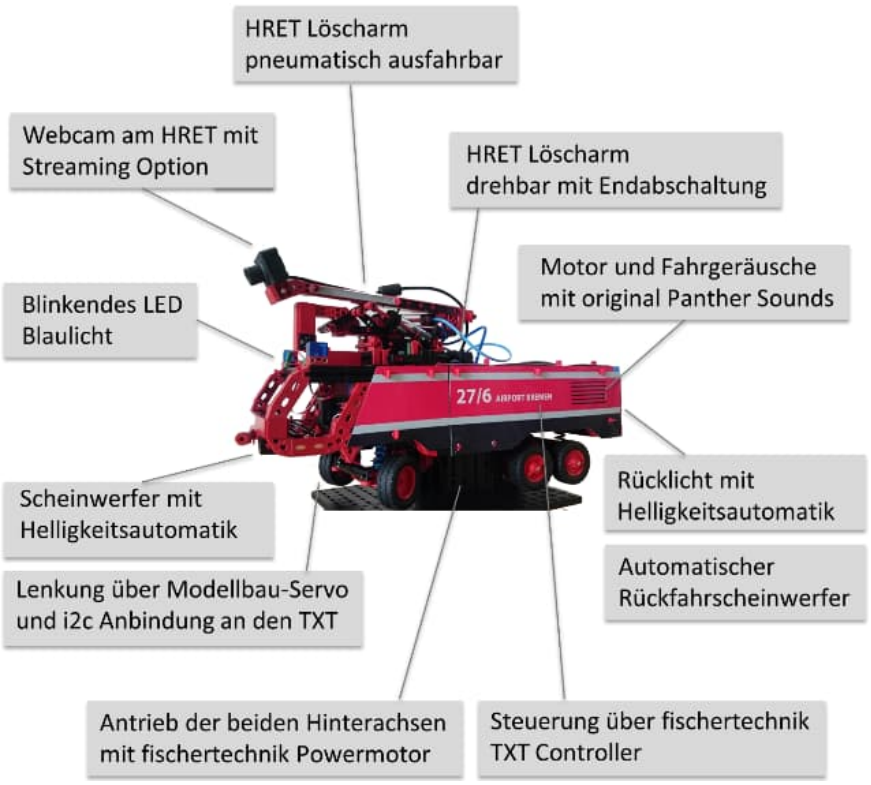


Abb. 4: Übersicht über die Funktionen

- Der HRET-Löscharm kann pneumatisch ein- und ausgefahren werden. Die Ansteuerung erfolgt dabei über zwei 3/2-Wege-Magnetventil, eines für die beiden Pneumatikzylinder des unteren Teils, das andere für den Zylinder des oberen Teils.
- Der HRET-Löscharm kann nach links und rechts geschwenkt werden. Zwei Endlagentaster sorgen dafür, dass der Minimotor beim Erreichen der Endlagen automatisch abschaltet.
- An der Spitze des HRET-Löscharms befindet sich die TXT-USB-Kamera. Beim Start des TXT wird in der in der Community Firmware verfügbare MJPG-Streamer gestartet, der permanent ein Livebild überträgt. So kann der Panther auch aus der „Fahrerperspektive“ ferngesteuert werden.
- Der Antrieb der beiden Hinterachsen erfolgt mit einem 20:1-fischertechnik-Powermotor. Im Kontroll-Interface kann der Motor mit einem Button „gestartet“ werden. Auf dem TXT wird dann ein von einem Original-Panther als WAV-Datei digitalisierter „Start“-Sound abgespielt. Ebenso wird während der Fahrt das Original-Motorengeräusch abgespielt. Diese Sounds können im Kontroll-Interface aber auch deaktiviert werden.
- Beim Rückwärtsfahren werden automatisch die Rückfahrscheinwerfer eingeschaltet.
- Die Blaulichter sind mit blinkenden LEDs realisiert und lassen sich ein- und ausschalten.
- Scheinwerfer und Rücklichter können manuell geschaltet werden. Zusätzlich ist ein Automatikmodus aktivierbar, der über einen lichtempfindlichen Widerstand bei Dunkelheit die Lichter automatisch einschaltet.
- Die Lenkung erfolgt über ein Servo. Da der TXT keinen Servo-Ausgang besitzt, wurde über den am EXT-Port des TXT verfügbaren I²C-Anschluss ein PCA9685-PWM-Servotreiber angeschlossen, der den Servo steuert [4]. Der PCA9685 bietet 16 Servokanäle, sodass noch reichlich Kapazität für Erweiterungen vorhanden ist.

Der TXT nutzt die Community Firmware. Die Funktionalität wird über ROBO Pro gesteuert. Dazu wurde in ROBO Pro ein einfaches Kontroll-Interface realisiert, mit dem der Panther vollständig von einem

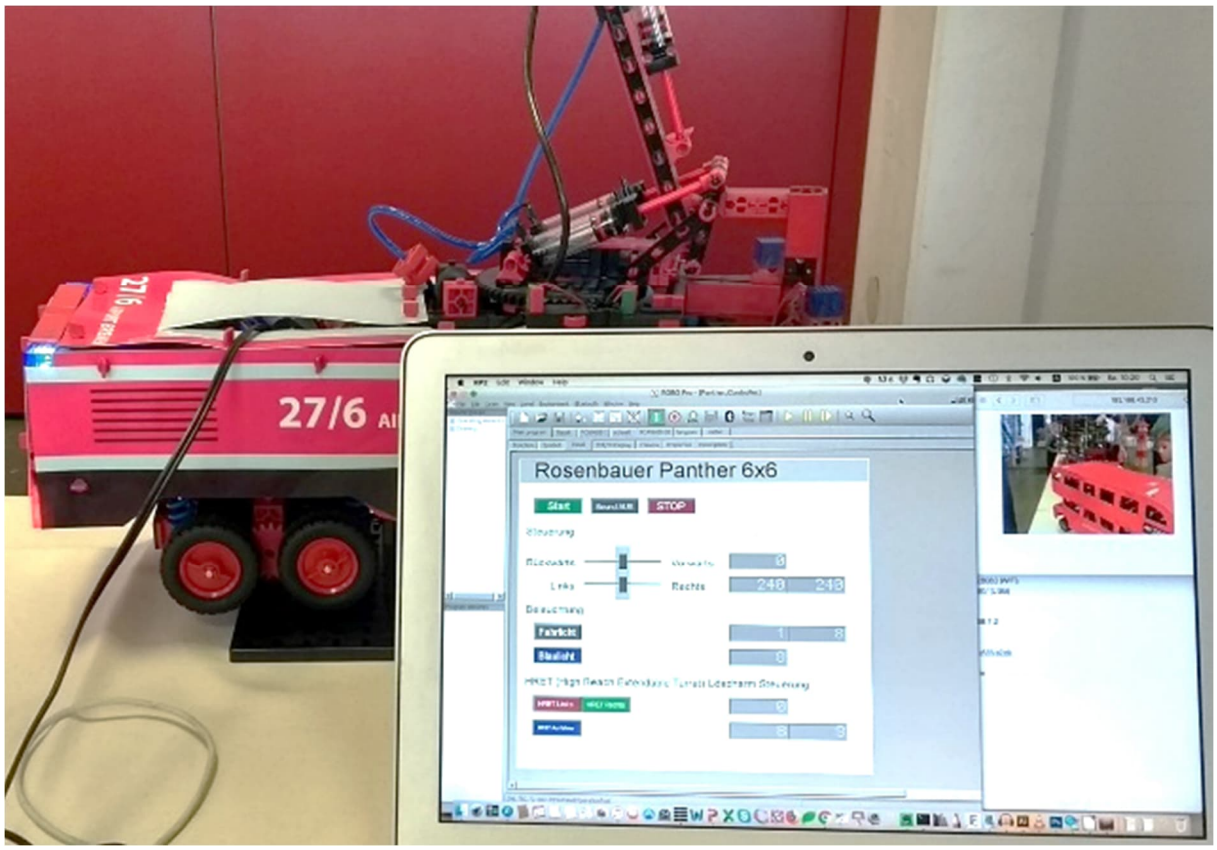


Abb. 5: Kontroll-Interface und Video-Stream

Computer aus ferngesteuert werden kann (Abb. 5).

Das Modell des Panthers 6×6 zeigt, was sich mit dem inzwischen schon historischen fischertechnik-TXT-Controller realisieren lässt. Als mögliche Erweiterungen wären ein Allradantrieb, der Umbau zur 8×8-Version, eine modellspezifische Fernsteuerung, der Austausch der Fotopapier-Verkleidung gegen 3D-gedruckte Teile sowie zusätzliche Funktionen denkbar.

Quellen

[1] Wikipedia: *Rosenbauer Panther*.

[2] Rosenbauer International AG: *PANTHER*. Hersteller-Webseite auf rosenbauer.com.

[3] Volker Paelke: *Rosenbauer Panther 6x6 in fischertechnik.* Auf YouTube, 2027.

[4] Dirk Fox: *I²C mit dem TX(T) – Teil 16: Servo-Driver*. ft:pedia 2/2017, S. 41–47.

Mechanik

Eine schräge Drehscheibe

Fabian Haas, Stefan Falk

Trotz aller elektronischer Steuerung und App-Kontrolle benötigt man weiterhin die Mechanik, um aus der mit einem Motor oder Aktuator erzeugten Bewegung eine Wirkung zu erzielen. Häufig müssen dabei Bewegungsarten ineinander umgesetzt werden: eine Drehung (Rotation) in eine Verschiebung (Translation), eine kontinuierliche Bewegung in eine rhythmische oder stufenweise. Das ist sozusagen das kleine 1×1 der Mechanik. In vielen sozialen Medien lassen sich solche Konstruktionen als CAD-Modelle finden, die der Umsetzung in das fischertechnik Raster harren. Wir nahmen die Herausforderung an, eine Scheibe schräg zur Drehachse zu montieren.

Das Internet bietet die wunderbare Gelegenheit, Menschen und Dinge kennenzulernen, die ansonsten nur nach langem Suchen zugänglich gewesen wären. Besonders die sogenannten sozialen Medien haben sich dem verschrieben und wir nutzen sie, um uns die Welt des Konstruktions- und Ingenieurwesens ganz unverbindlich und neugierig anzuschauen.

Dabei sind die Darstellungen von CAD-Lösungen für mechanische Probleme besonders inspirierend. Unsere Inspirationsquellen waren auf Instagram besonders die User *3D Design Pro* [2], *3d.designer,*

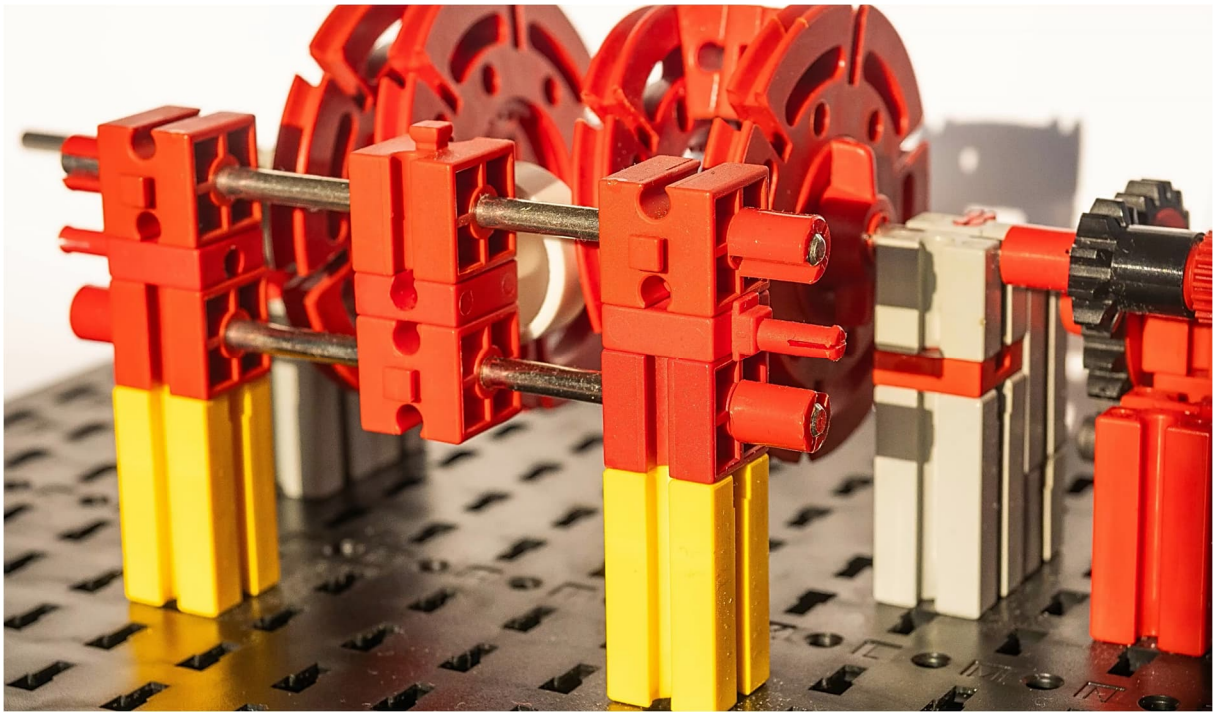


Abb. 1: Das weiße Rad 14 wird von den schrägstehenden Drehscheiben hin und her bewegt

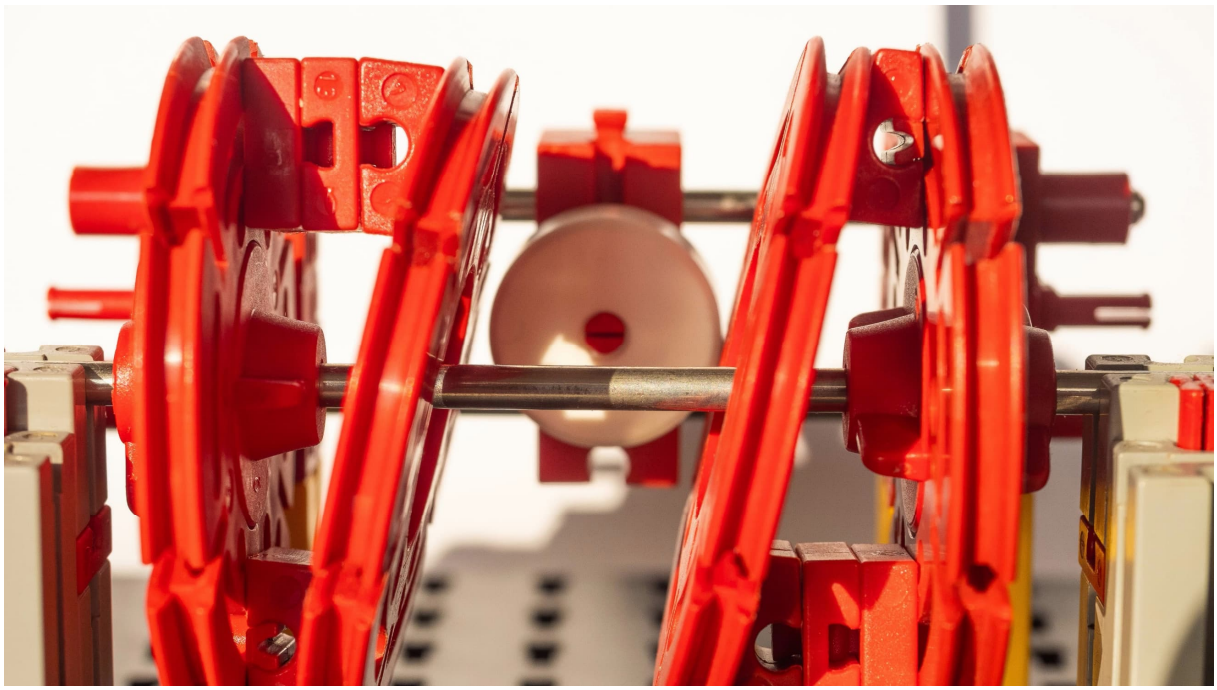


Abb. 2: Blick durch die schräggestellten Drehscheiben auf das zu bewegende Rad 14

engineer.can, cnc_jia_jia oder *CAD Designer* [3]. Weitere finden sich auf YouTube und vielen anderen Plattformen. CAD ist Computer Aided Design, das genutzt wird, um Simulationen von mechanischen Modellen zu erstellen und schon vor dem Bau durchzurechnen. Die Rechenmodelle sind heutzutage extrem ausgefeilt und nicht nur die Kinematik (die Bewegung ohne Kräfte), auch die Kinetik (mit Kräften) lässt sich simulieren, inklusive der Materialeigenschaften. Dabei sind Formen und Größen der Einzelteile in CAD frei definierbar. Nicht so wie bei fischertechnik, das durch die Bausteine bestimmte Maße und Formen (das Raster) vorgibt. Die Kunst des Bauens in fischertechnik ist, wenigstens zum Teil, innerhalb des Rasters einen Mechanismus darstellen und geschmeidig zum Funktionieren zu bringen.

So war es auch bei diesem Modell. Ausgangspunkt waren Darstellungen einer Achse, auf die in einem Winkel (also schräg) eine Scheibe montiert bzw. angeschweißt wurde. Rotiert die Achse, bewegt sich die schräge Scheibe entsprechend, und wenn die schräge Scheibe durch eine Führung läuft, kommt es durch die Rotation zu einer hin- und her-Bewegung der Führung. Das kann zum Beispiel in einem Schlagbohrer genutzt werden, um eben die Schläge zu erzeugen, oder für eine Säge, die sich entsprechend der Auslenkung durch die Scheibe vor und zurück bewegt.

Kann ich das in fischertechnik bauen? Herausforderung angenommen! Achsen und Scheiben gibt es genug, ein Antrieb und eine Führung bekommt man bestimmt hin. Das Kernproblem war: Wie bekommt man eine Scheibe, am besten eine Drehscheibe 60, schräg zur Achse montiert? Eine Nabe mit schrägem Loch ist nicht bekannt. Nach einiger Zeit, Grübeln und dem einen oder anderen gescheiterten Versuch entstand die Lösung: Eine Drehscheibe 60 wird ganz normal (senkrecht) mit einer Nabe auf einer Achse montiert. Eine zweite Drehscheibe 60 wird mit Winkelsteinen 15 auf der ersten angebracht und die notwendige Höhe mit Bausteinen 5 angepasst. Damit sitzt nun eine Drehscheibe schräg zu der Achse. Der Winkel zur Achse lässt sich über die verwendeten Winkelsteine einstellen (wobei wir das nicht durchprobiert haben).

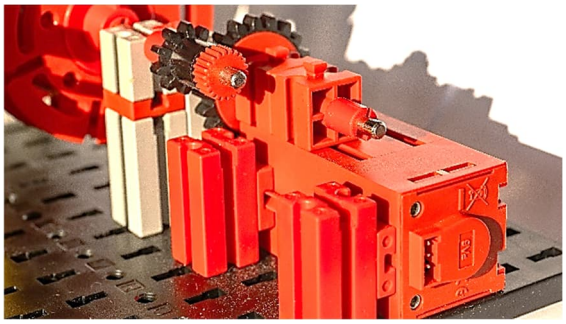


Abb. 3: Blick auf den Antrieb

Ein Problem gelöst, ein Neues aufgetaucht: Durch die Verbindung der schräg montierten Drehscheibe 60 auf eine zweite kann die Drehscheibe 60 nicht in eine Führung aus zwei Rädern eingreifen. Die Winkelsteine würden die Drehung blockieren. Aber warum das Ganze nicht herumdrehen? Warum nicht ein Rad zwischen zwei Scheiben statt zweier Räder vor und hinter einer Drehscheibe 60?

Theoretisch sollte das gehen, und siehe da: Es geht auch in der fischertechnik-Praxis! Verwendet wurde ein Rad 23, das auf einer Radachse zwischen zwei Bausteinen 15 mit Bohrung auf zwei Achsen 120 hin und her läuft. Alles weitere ergibt sich aus den Fotos und dem Video, das das Modell in Funktion zeigt [1].

Statt eines Rads 23 ließe sich irgendein Mechanismus ansetzen, der ein Sägeblatt antreibt oder auf irgendetwas schlägt (was uns hier nicht gereizt hat). Die schräge Montage der Drehscheibe 60 war zunächst das Schlüsselproblem, und hoffentlich lässt sich diese Lösung noch in anderen Situationen verwenden.

Das Modell ist zwar etwas grösser geworden als gedacht, allerdings ging es ja auch um die Umsetzung der Idee und nicht darum, eine konkrete Maschine oder

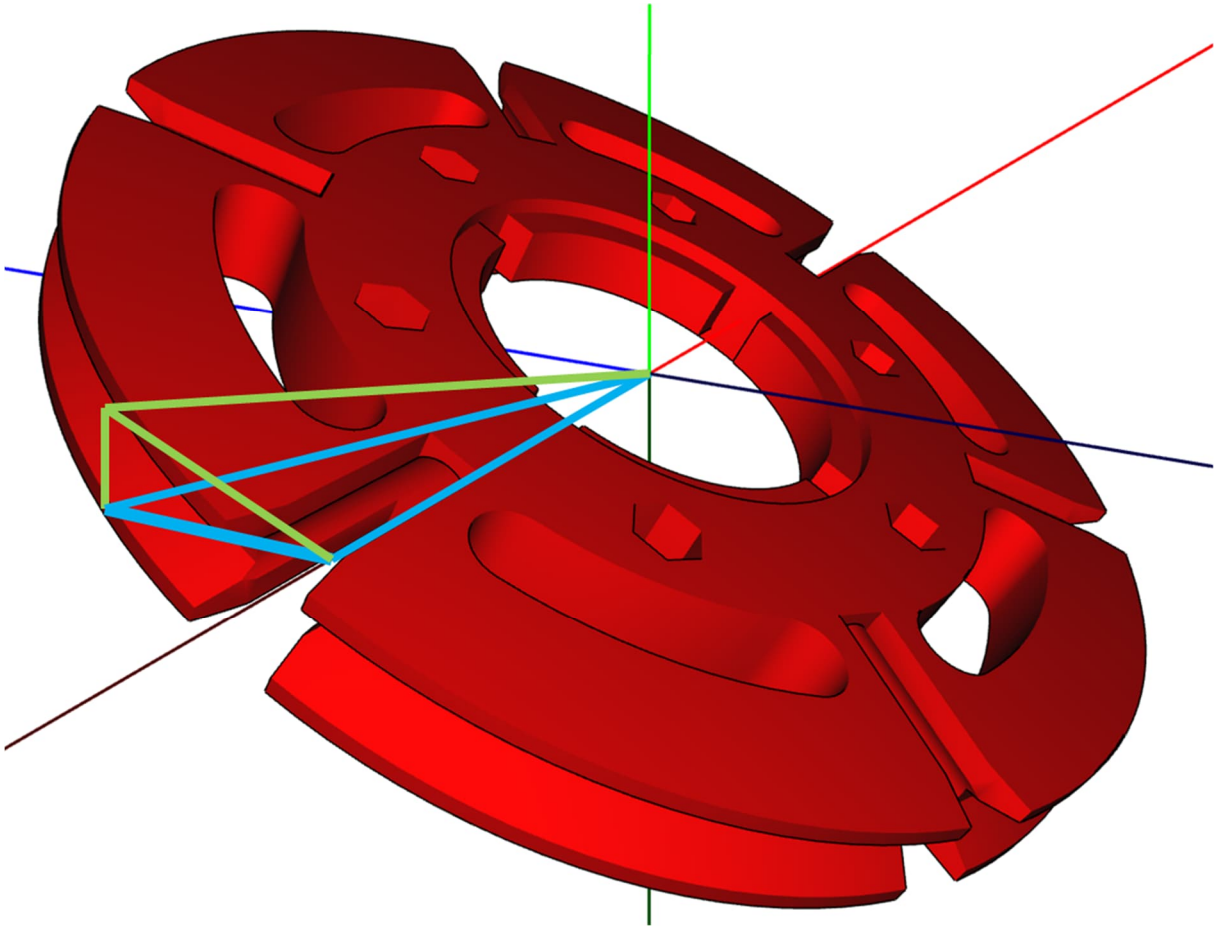


Abb. 4: Die schräge Drehscheibe und die Berechnungsdreiecke

Anwendung zu bauen. Gibt es eine kompaktere Realisierung? Vielleicht findet jemand eine „kleine Lösung“.

Viel Spaß beim Nachbauen und weiterentwickeln!

Die Mathematik dahinter

Nun kann man diese Konstruktion in einer Maschine verwenden und sich an ihrer Funktion erfreuen. Für alle, die es genauer wissen wollen, leiten wir hier genau her, wie sich das hin- und hergeschobene Rad genau bewegt.

Wir nehmen folgende vereinfachende Idealisierung vor:

- Die Nuten der Drehscheibe, die das Rad berühren und zu einem leicht ungleichförmigen Lauf führen, denken wir uns weg. Die Drehscheibe sei „glatt“.
- Wir nehmen an, dass der Mittelpunkt der Drehscheibe auf der Antriebsachse liegt. Die schräge Drehscheibe muss also über die Nuten in den BS5 so justiert werden, dass dies der Fall ist.
- Wir nehmen weiterhin an, dass zwischen der schrägen Drehscheibe und dem geschobenen Rad 14 kein Spiel besteht.

Abb. 4 zeigt die schräge Drehscheibe und ein eingezeichnetes räumlich dreieckiges Gebilde; Abb. 5 zeigt nur das Dreiecksgebilde.

Die Farben in dem Gebilde bedeuten Folgendes:

- Alles blau Dargestellte liegt in der Ebene senkrecht zur Antriebsachse, also in einer Ebene parallel zur nicht schrägstehenden Drehscheibe.
- Alles orange Dargestellte bezieht sich auf die Ebene der schrägen Drehscheibe.
- Grün gesetzte Angaben sind uns bekannt oder gegeben:
  - Der Radius r der Drehscheibe, hier also 30 mm,
  - der Schrägstellungs-Winkel α, mit den genannten Winkelsteinen 15°, also eben diese 15° sowie
  - der Rotationswinkel ω, um den der Antrieb die nicht-schräge Drehscheibe gerade gedreht hat.
- Das rot gesetzte h ist die Höhe, um die das Rad 14 von der Drehscheibe ausgelenkt wird. Das ist die gesuchte Größe in Abhängigkeit von r, α und ω.
- Punkte sind mit Großbuchstaben beschriftet, Strecken bzw. Streckenlängen mit Kleinbuchstaben.

Das Dreieck ABC liegt also in der Ebene parallel zur geradestehenden Drehscheibe. Das Dreieck ABD liegt mit der blauen Kante ebenfalls dort, mit den gelben Linien aber in der Ebene der schrägen Drehscheibe.

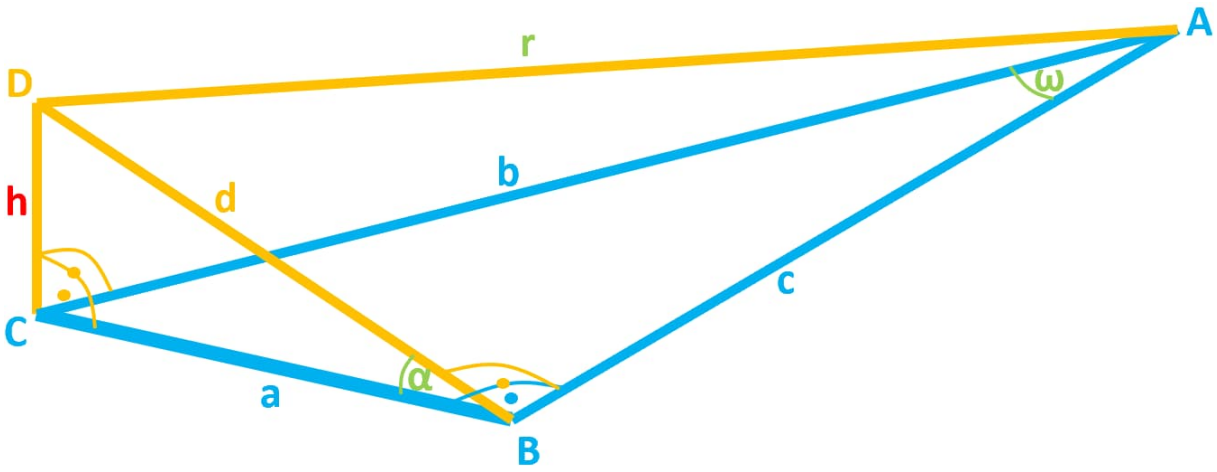


Abb. 5: Die Berechnungsdreiecke

Punkt *C* ist die Projektion des Punktes *D* (an dem das Rad 14 anliegt) auf die Ebene der geraden Drehscheibe. Wir *wählen* die Länge *c* so, dass der Winkel ABC ein rechter Winkel wird. Damit sind auch die orange gezeichneten Winkel ACD, BCD und ABD rechte Winkel.

Wegen der rechten Winkel ergeben sich nach dem Satz des Pythagoras eine ganze Reihe von Gleichungen:

$$a^2 + c^2 + b^2 \tag{1}$$

$$a^2 + h^2 = d^2 \tag{2}$$

$$b^2 + h^2 = r^2 \tag{3}$$

$$a^2 + c^2 = r^2 \tag{4}$$

Tatsächlich genügt uns, wie wir noch sehen werden, Gleichung (4), aber der Vollständigkeit halber seien alle genannt.

Außerdem ergibt sich aus den bekannten Winkeln α und ω:

$$h = a \tan \alpha \tag{5}$$

$$a = c \tan \omega \tag{6}$$

$$a = b \sin \omega \tag{7}$$

Auch davon brauchen wir tatsächlich nicht alle. Jedenfalls können wir nun (7) in (5) einsetzen und erhalten

$$h = b \sin \omega \tan \alpha \tag{8}$$

Gleichung (3) nach *b* aufgelöst ist:

$$b = \sqrt{r^2 - h^2} \tag{9}$$

Dieses *b* können wir in (8) einsetzen und erhalten eine Gleichung, in der nur noch bekannte Größen und das zu bestimmende *h* auftreten:

$$h = \sqrt{r^2 - h^2} \sin \omega \tan \alpha \tag{10}$$

Da *h* sowohl links als auch innerhalb der Wurzel rechts auftritt, lösen wir diese Gleichung schrittweise nach *h* auf:

$$h^2 = (r^2 - h^2) \sin^2 \omega \tan^2 \alpha \tag{11}$$

$$h^2 = r^2 \sin^2 \omega \tan^2 \alpha - h^2 \sin^2 \omega \tan^2 \alpha \tag{12}$$

$$h^2 (1 + \sin^2 \omega \tan^2 \alpha) = r^2 \sin^2 \omega \tan^2 \alpha \tag{13}$$

$$h^2 = \frac{r^2 \sin^2 \omega \tan^2 \alpha}{1 + \sin^2 \omega \tan^2 \alpha} \tag{14}$$

Wir dividieren in Gleichung (14) rechts sowohl den Zähler als auch den Nenner durch $\tan^2 \alpha$:

$$h^2 = \frac{r^2 \sin^2 \omega}{\frac{1}{\tan^2 \alpha} + \sin^2 \omega} \tag{15}$$

Nach Ziehen der Wurzel und Einsetzen von

$$\frac{1}{\tan \alpha} = \cot \alpha \tag{16}$$

resultiert für die gesuchte Auslenkung *h* die finale Gleichung

$$h = \frac{r \sin \omega}{\sqrt{\sin^2 \omega + \cot^2 \alpha}} \tag{17}$$

Wenn wir aus den Quadraten die Wurzel ziehen, bekommen wir streng genommen nur eine Aussage über die Beträge der Größen, nicht über ihre Vorzeichen. Dass *h* sein Vorzeichen aber mit $\sin \omega$ teilt, ergibt sich aus der mechanischen Konstruktion.

Wer das auf einem Taschenrechner oder einem Computer ohne cot-Funktion (Cotangens) berechnen möchte, verwendet also den Kehrwert des tan (Tangens), ohne Gleichung (16) zu benutzen.

h ist also fast proportional zu $r \sin \omega$, einer normalen Sinus-Funktion, aber eben nur fast: Auch im Nenner des Bruches in Gleichung (17) geht der Drehwinkel ω ein. Grafisch sieht es einem Sinus aber trotzdem sehr ähnlich.

Vergleich A: sin²(ω) := 0

Das Diagramm in Abb. 6 zeigt in Grün die echte Funktion von *h* in Abhängigkeit von *r* = 30 mm, α = 15° und ω, während in Rot die Funktion dargestellt ist, wie wenn $\sin^2 \omega$ im Nenner nicht vorkäme und Gleichung (17) sich damit zu

$$h = \frac{r \sin \omega}{\cot \alpha} = r \sin \omega \tan \alpha \qquad (18)$$

vereinfachen würde. Man sieht, dass die originale grüne Linie kleinere Auslenkungen darstellt als die rote aus der vereinfachten Gleichung (18). Es sieht vielleicht so aus, als ob ein konstanter Faktor zwischen beiden Funktionen läge, das ist aber nicht exakt der Fall.

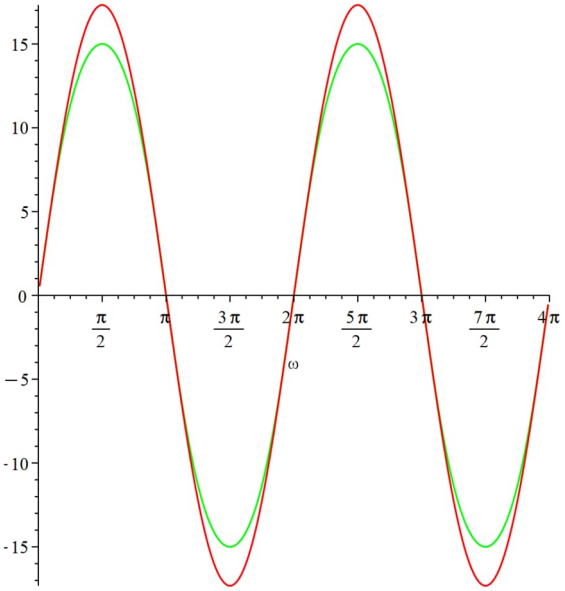


Abb. 6: Resultierender Verlauf echt (grün) gegenüber vereinfacht (rot) mit der Vereinfachung sin²(ω) := 0

Die Auslenkung ω liegt in Abb. 6 auf der waagerechten Achse im Bogenmaß (4π entsprechen 720°, also zwei ganzen Umdrehungen). Die senkrechte Achse gibt die Auslenkung des Rades 14 in mm wieder.

Vergleich B: sin²(ω) := 1

In einem anderen Vergleich nutzen wir, dass $\sin^2(\omega)$ höchstens 1 werden kann. Wir ersetzen es also mal durch 1 und erhalten

$$h = \frac{r \sin \omega}{\sqrt{1 + \cot^2 \alpha}} \qquad (19)$$

Das führt zum Graphen in Abb. 7. Wenn man genau hinschaut, sieht man den leichten Unterschied zwischen der roten und der grünen Linie.

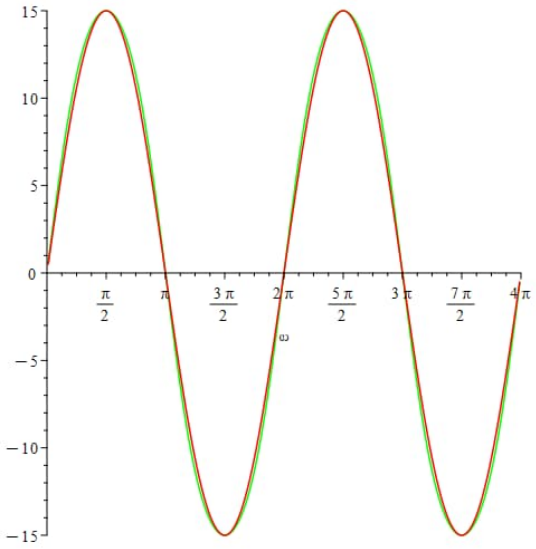


Abb. 7: Resultierender Verlauf echt (grün) gegenüber vereinfacht (rot) mit der Vereinfachung sin²(ω) := 1

Also: Die Mechanik produziert fast, aber nicht exakt einen Sinus. Der Unterschied ist aber recht klein – da wird das Spiel in der Mechanik größeren Einfluss haben.

Konstruktionsvariante ohne Drehscheiben

Noch offen war, eine äquivalente Mechanik ohne Drehschreiben herzustellen. Auch das gelingt, wie Abb. 8, 9 und das Video unter [4] zeigen. Dieser Aufbau ist mathematisch – bis auf Spiel in der Mechanik – identisch zur Konstruktion mit Drehscheiben:

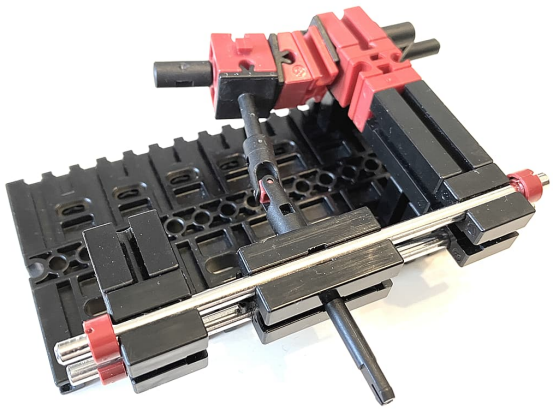


Abb. 8: Ansicht 1 des Kleinmodells

Wir drehen also keine schräge Drehscheibe, sondern eine schrägstehende Achse. Auf

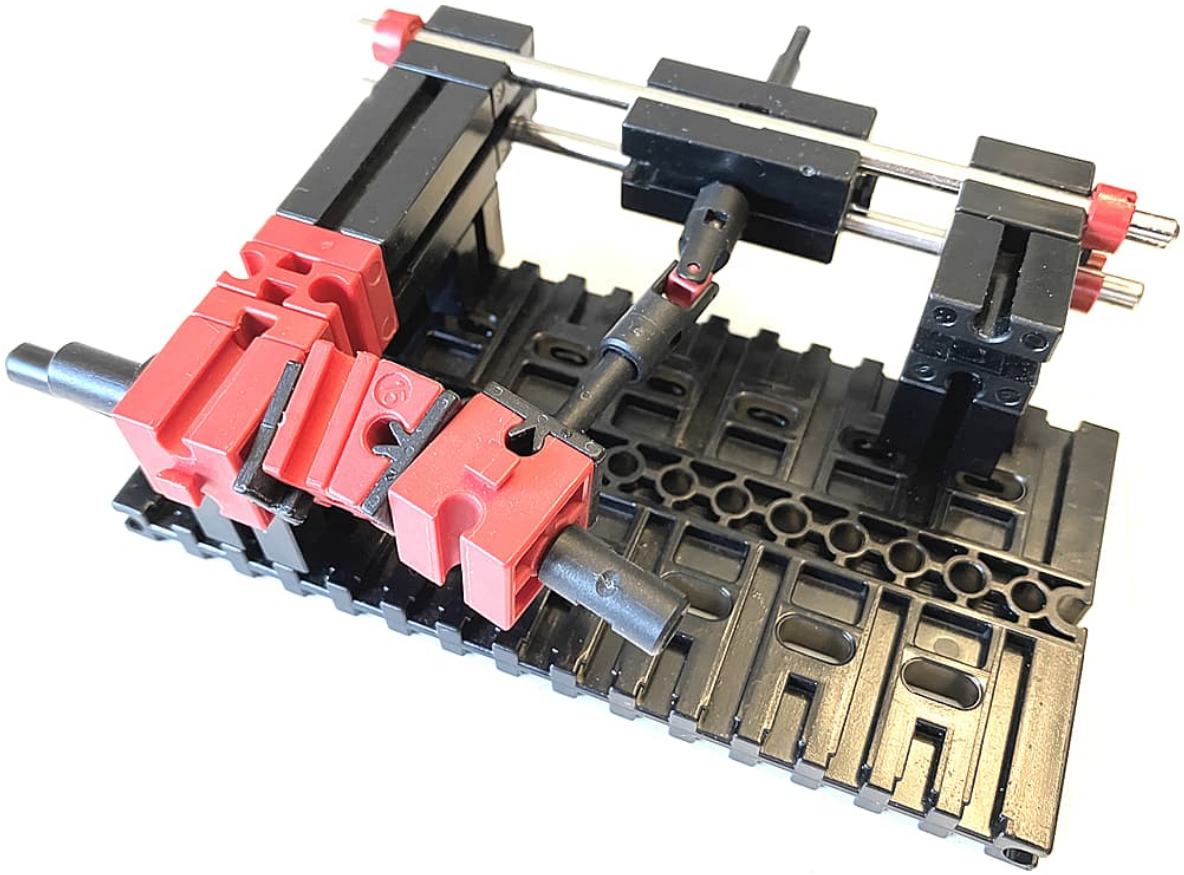


Abb. 9: Ansicht 2 des Kleinmodells

der sitzt drehbar der rote Baustein 15 mit Bohrung, von dem senkrecht die Achse (per Rastaufnahmeachse 22,5 130593) abgeht. Auf dieser sitzt ein Rastkardangelenk, das sozusagen die Funktion der schräggestellten Drehschreibe übernimmt – der Drehpunkt des Kardans entspricht dem Rand der Drehscheibe, auf dem das Rad 23 abrollt.

In der anderen Seite des Kardans steckt eine Rastachse 45, die durch einen BS30 mit Bohrung geht. Dieser wiederum wird mühelos gleitend auf zwei Metallachsen geführt.

Die Mechanik funktioniert leichtgängig, ob mit Kurbel oder auch mit einem Motor. Damit sie in beide Richtungen gleichförmig wirkt, muss der Drehmittelpunkt des schrägen roten BS 15 mittig liegen. Dazu wird der BS 7,5 leicht versetzt mit der Antriebskurbel verbunden.

Quellen

[1] Dr. Fabian Haas: *Rotation in Translation - Rotation to Translation*. Auf YouTube, 2025.

[2] *3D Design pro* auf YouTube.

[3] *CAD Designer* auf YouTube.

[4] Stefan Falk: *Rotation in Translation Kleinmodell*. Auf YouTube, 2025.

Modell

Wolf, Schaf und Kohlkopf (Teil 2)

Arnoud van Delden

Wer kennt es nicht, das klassische „Flussüberquerungsrätsel“ eines Bauern auf dem Weg zum Markt mit einem Wolf, einem Schaf (oder einer Ziege) und einem Kohlkopf? Zum Glück für den Bauern gibt es ein kleines Boot, das er benutzen kann. Er kann aber immer nur einen der drei mit ins Boot nehmen und muss darauf achten, dass der zurückgelassene Wolf nicht das Schaf und das Schaf nicht den Kohlkopf frisst, wenn sie ohne Aufsicht des Bauern zusammen am Ufer zurückbleiben. Die Aufgabe besteht darin, Wolf, Schaf und Kohlkopf auf die andere Seite des Flusses zu bringen, und zwar mit so wenig Flussüberquerungen wie möglich.

Wahrscheinlich hatte der Bauer schon vor mehr als 10 Jahren im Jahr 2015 gelesen, wie Stefan Falk beschrieb, wie er mit seiner fischertechnik-Box und etwas bäuerlichem Hausverstand dieses Problem lösen konnte, noch bevor er sich auf den Weg zum Markt machen würde [1].

Ich vermute, dass es sich um einen Bauern im Grenzgebiet zwischen Deutschland und den Niederlanden handelte. Und dass dieser Bauer seinen Wolf beim Hin- und Herfahren irgendwo am anderen Ufer unbeaufsichtigt gelassen hat. Dabei ist das Tier in Richtung Westen gewandert, denn es kann kaum ein Zufall sein, dass 2015 der erste Wolf (wieder) in den Niederlanden gesichtet wurde. Dieser Wolf kam damals tatsächlich aus Deutschland und legte Hunderte von Kilometern zurück, um in die Niederlande zu gelangen. Ist es nicht schön zu wissen, dass es niederländische fischertechnik-Fans (wie mich) gibt, die gelegentlich Hunderte von Kilometern in die andere Richtung fahren, um fischertechnik-Treffen in Deutschland zu besuchen?

Elektronik ersetzt Elektromechanik

So wurde schon vor langer Zeit demonstriert, wie das Problem auf geniale Weise rein elektromechanisch gelöst werden kann. Die beweglichen Balken in Stefans Modell sollen die Position des Wolfs, des Bauern, des Schafs und des Kohlkopfs darstellen. Eine durchdachte Methode, die rein mechanisch funktioniert. Allerdings ist bei dieser Methode etwas Fantasie gefragt, um die einzelnen Situationen zu durchschauen und das Rätsel zu lösen.

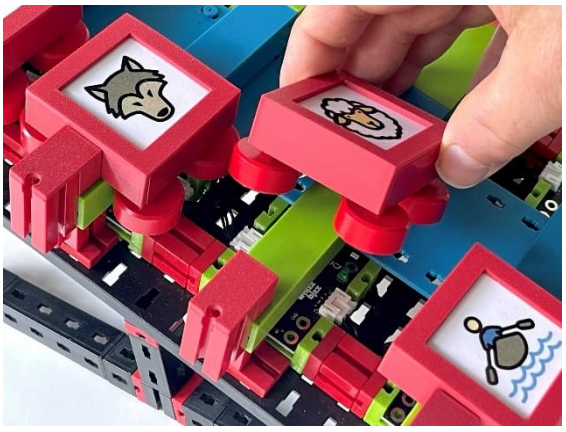


Abb. 1: Jedes Symbol hat seinen eigenen Wagen, der zwischen den Flussufern bewegt werden kann

Ich habe lange darüber nachgedacht, wie man die Bewegungen des Bootes, des Kohls und der Tiere anschaulicher machen könnte. Zum Beispiel mit einem Modell, in dem das Boot und die Tiere tatsächlich über den metaphorischen Fluss bewegt werden

können. Als ich schon fast mit der Arbeit an einer Steuerungslösung mit einem echten Boot begonnen hatte, in dem die Figuren durch den NFC-Chip in ihrem Sockel erkannt werden konnten, beschloss ich zunächst eine technisch einfachere Lösung zu bauen. Hatte Stefan nicht schon vor zehn Jahren mit seiner Lösung den wahren Charme der mechanischen Einfachheit bewiesen?

Doch die Idee von Schiebern oder Elementen, die vom Rätselspieler selbst über den imaginären Fluss geschoben werden können, reizte weiterhin. Wenn jeder Schieber, solange er sich an einem Ufer befindet, einen Taster gemäß Abb. 2 betätigt, bleibt nur noch die Implementierung einer Logik übrig. Und das wäre mit traditionellen Silberlingen durchaus möglich. Man beachte, dass die Verdrahtung der Drucktasten hier bereits der negativen Logik der Silberlinge folgt. Durch Vertauschen von Ruhe- und Arbeitskontakt als Signalquelle am Taster des Bauern wird es bereits logisch invertiert.

Was oder wer wird wann gefressen?

Stefan hat in seinem Artikel bereits ausführlich in Tabellenform erläutert, welche Situationen der Bauer beim Hin- und Herfahren an den verschiedenen Ufern vermeiden sollte. Dass Wölfe eine ernsthafte Bedrohung für unbeaufsichtigte Schafe darstellen, haben leider schon viele Schafhüter nicht nur in den Niederlanden erfahren. Umgekehrt ist der Kohlkopf ohne die Aufsicht des Landwirts natürlich auch nicht sicher vor den Schafen.

Lassen wir uns die zu vermeidenden Situationen logisch zuordnen, wenn sich der Bauer nicht am betreffenden Ufer aufhält. In Abb. 3 ist jedes entsprechende Signal „aktiv“, wenn sich das betreffende Objekt oder Tier am jeweiligen Ufer befindet. Was den Bauern betrifft, ist es vor allem interessant, wenn er sich nicht auf der betreffenden Bank befindet. Daher ist für ihn ein logisch invertiertes Signal $\overline{B}$ sinnvoller, das also genau dann aktiv ist, wenn er sich nicht am Ufer befindet.

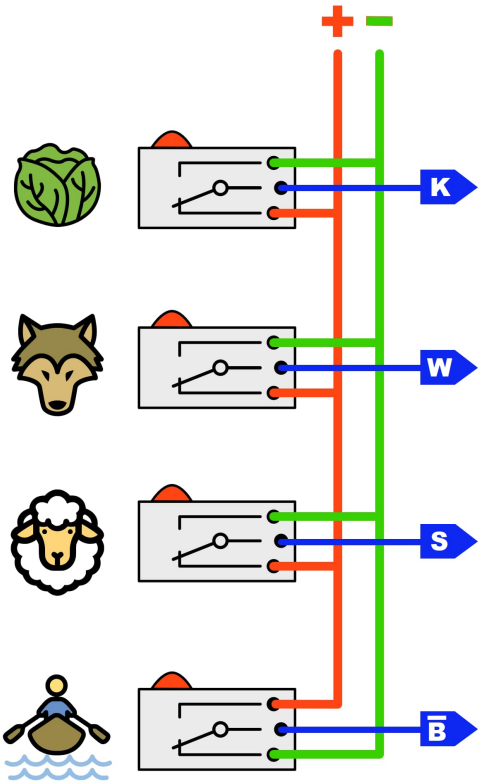


Abb. 2: Die für ein Flussufer erforderlichen Signale sind schnell erstellt

Die beiden logischen UND-Gatter (Abb. 3) erkennen jeweils eine Situation, die vermieden werden muss, wenn der Bauer das jeweilige Flussufer verlässt. Das obere Gatter erkennt die Gefahr, die entsteht, wenn der Kohlkopf und das Schafe unbeaufsichtigt gelassen werden. Das untere erkennt die unerwünschte Situation, in der das unbeaufsichtigte Schaf vom Wolf gefressen werden könnte. Ein ODER-Gatter verknüpft diese beiden Gefahrensituationen, sodass in diesen Fällen ein rotes Warnlicht aufleuchtet. Besteht keine Gefahr, leuchtet die grüne Lampe (= „An diesem Flussufer ist alles ungefährlich“).

Elektronische Logik

Pro Flussufer werden also zwei UND-Ports mit (mindestens) drei Eingängen benötigt;

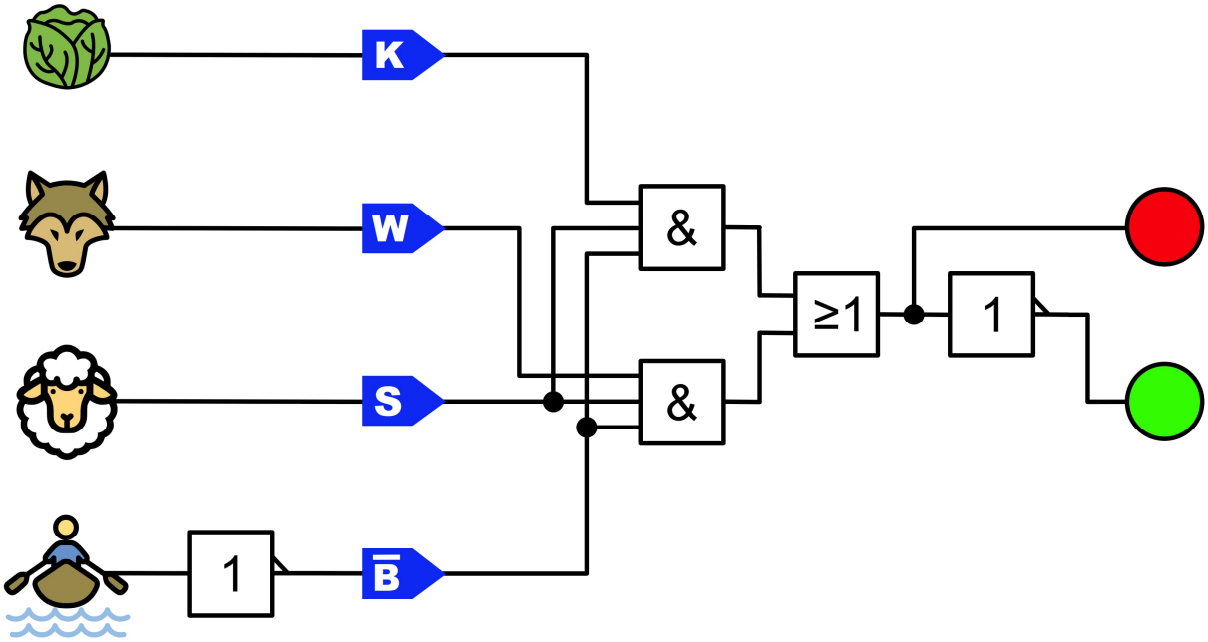


Abb. 3: Logische Darstellung der beiden „Problemsituationen“ an einem Flussufer. Wir brauchen diese Logik für jedes Flussufer, also zweimal.

außerdem ein separater Inverter (oder ein Schaltrelais) zur Ansteuerung der grünen Signallampe. Wer dies mit diskreten Bauelementen aus der traditionellen 7400-IC-Familie lösen will, kann sich z. B. für den 74LS55 entscheiden. Dieser hat auch schon das benötigte ODER-Gatter gleich mit drin. Es muss dann nur noch ein invertiertes Ausgangssignal für die rote LED erzeugt werden. Die grüne kann direkt an den invertierten Ausgang dieses ICs angeschlossen werden.

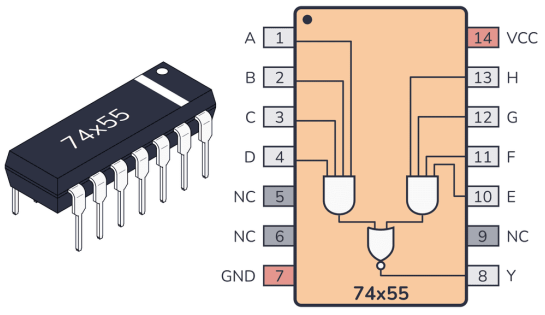


Abb. 4: Der 74(LS)55 aus der Standard-IC-Familie 7400

Aufbau mit Silberlingen

Wer genügend dieser klassischen fischertechnik-Elektronikmodule vorrätig hat (oder, wie ich, nachbaut), löst das Problem natürlich lieber mit „Silberlingen“. Abb. 5 zeigt den Silberlingen-Schaltplan, in dem die Signale K (Kohlkopf), W (Wolf), S (Schaf) und $\overline{B}$ („der Bauer ist weg“) logisch verarbeitet werden.

Die Stromversorgung mit dem Gleichrichtermodul ist nicht gezeichnet. Die oberste Reihe mit drei Logikgattern zeigt die Situation am linken Flussufer. Die untere Reihe zeigt die Situation am rechten Flussufer. Die grüne Lampe leuchtet, solange alles sicher ist, die rote Lampe leuchtet, sobald am betreffenden Ufer eine unerwünschte Situation eintritt.

Sensor-Adaptermodul

Es gibt natürlich viele Möglichkeiten, das Rätsel mit fischertechnik aufzubauen. Wie ich oben erklärt habe, wollte ich die Analogie eines Flusses mit zwei Ufern so gut wie möglich nachbilden. Außerdem dachte ich, es wäre schön, wenn der Kohlkopf und die

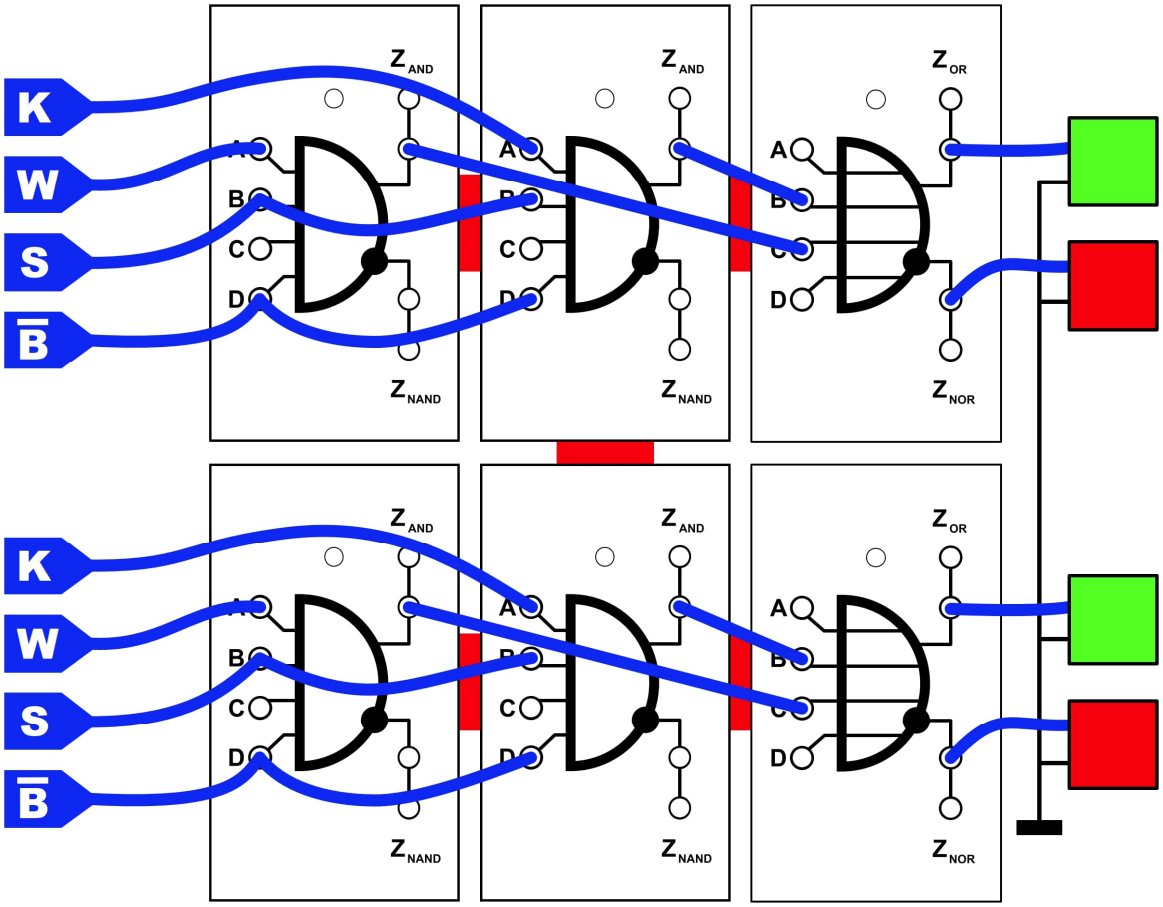


Abb. 5: Zwei identische Reihen von Silberlingen, jede für ein Flussufer

Tiere jeweils wie der Bauer vom Rätselspieler mit seinem eigenen „Boot“ von links nach rechts bewegt werden könnten. Dabei darf das „Boot“ des Bauern jedes Mal gleichzeitig mit höchstens einer anderen „Ladung“ auf die andere Flussseite bewegt werden. Schließlich kann der Bauer jedes Mal nur einen der drei anderen in seinem kleinen Boot auf die andere Seite bringen.

Da ich gerade an meinem Sensor-Adaptermodul [2] gearbeitet hatte, wollte ich die Erkennung berührungslos mit Reed-Kontakten anstelle von Tastern (wie in Abb. 2) durchführen. Zum Anschluss habe ich Modelle mit JST-Steckern verwendet. So konnte auch die notwendige Versorgungsspannung einfach durchgeschleift werden, ohne zu viele fischertechnik-Stecker zu verwenden.

Obwohl es gar nicht notwendig war, hat es mir Spaß gemacht, verschiedene benutzerdefinierte Teile für das Modell zu entwickeln und in 3D zu drucken. In Abb. 7 ist die Konstruktion meiner endgültigen „Karre“ mit dem U-förmigen Bauplattenstein mit Fenster zu sehen, unter das das Symbol des jeweiligen Tieres geklemmt werden kann. Eine Bauplatte mit Magnethalterung für den runden ⌀15×3-mm-Neodym-Magneten ist ebenfalls auf dem Bild zu sehen.

Zur Verdeutlichung ist in Abb. 6 die untere Schiene entfernt, sodass das Adaptermodul mit dem Reedkontakt zum Vorschein kommt. Dieser fällt in einen hohlen (maßgeschneiderten) Baustein, auf den die grünen Bauplatten geschoben werden. Ich habe die verschiedenen Bauplatten und die Hohlbausteine selbst gedruckt, damit ich die blaue Farbe mit den ebenfalls selbst

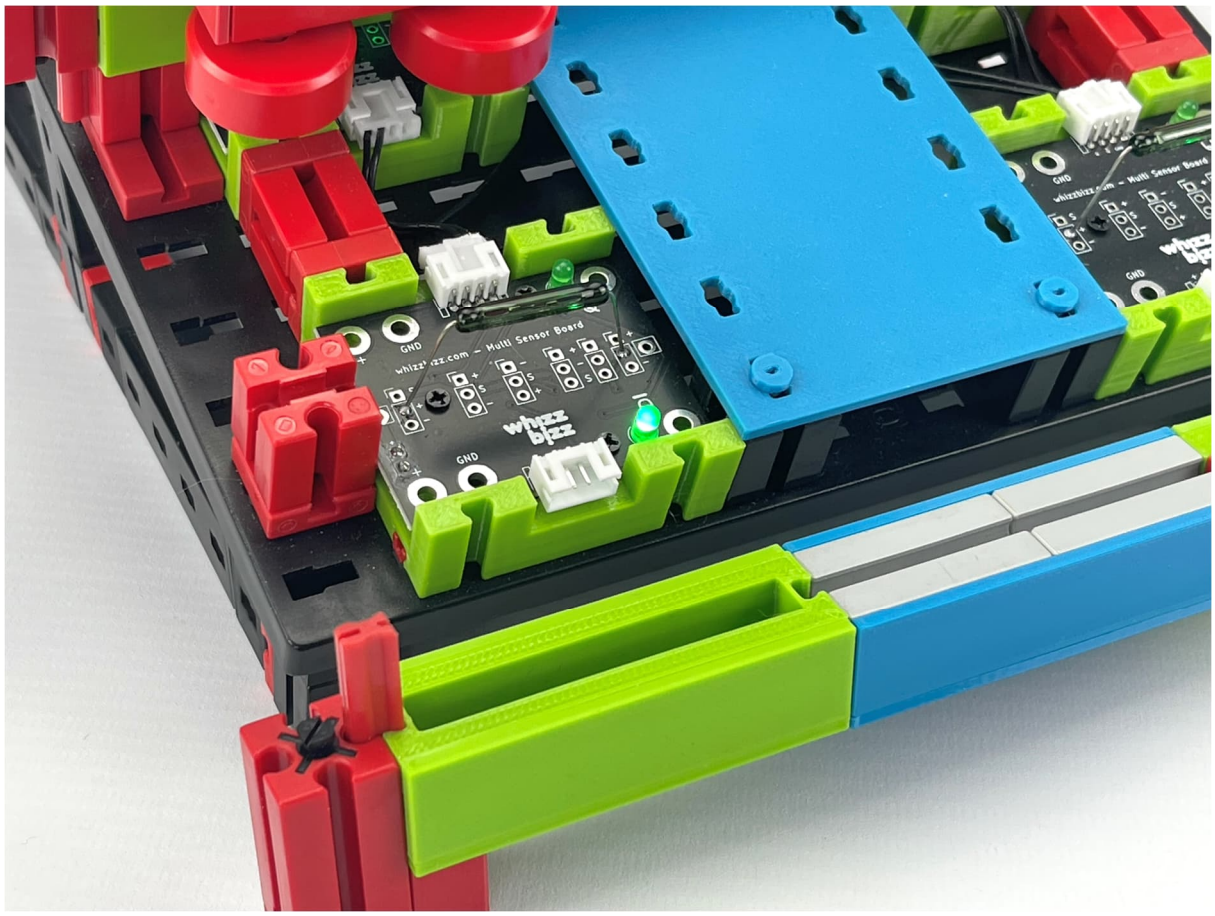


Abb. 6: Das Sensormodul mit Reedkontakten unter der Schiene

gedruckten blauen Statikplatten, die den Fluss darstellen, in Übereinstimmung bringen konnte. Sie sorgen dafür, dass die Wagen reibungslos über den „Fluss“ hin und her bewegt werden können. Natürlich können für diesen Zweck auch die vorhandenen fischertechnik-Bauplatten verwendet werden.

Der Aufbau des Rätsels

Abb. 8 zeigt den Aufbau des Puzzles. Da ich ein Verbindungskabel hergestellt habe, konnte das Modell auf zwei kleinere Grundplatten aufgeteilt werden. Auf der linken Seite ist der „Fluss“, auf dem jedes Mal das Boot mit maximal einem anderen Symbol von einem Ufer zum anderen bewegt werden kann. Rechts befindet sich die Platte mit der Silberling-Logik. Wie man sehen kann, habe ich ein Paar Mini-Ampeln [3] als rote und grüne Warnlichter über jedem ‚Ufer‘ des imaginären Flusses angebracht. Es besteht aber auch die Möglichkeit, fischertechnik-LED-Lichtbausteine direkt an die entsprechenden Ausgänge der Silberlinge anzuschließen.

Abb. 7: Jedes Symbol ein eigener Wagen

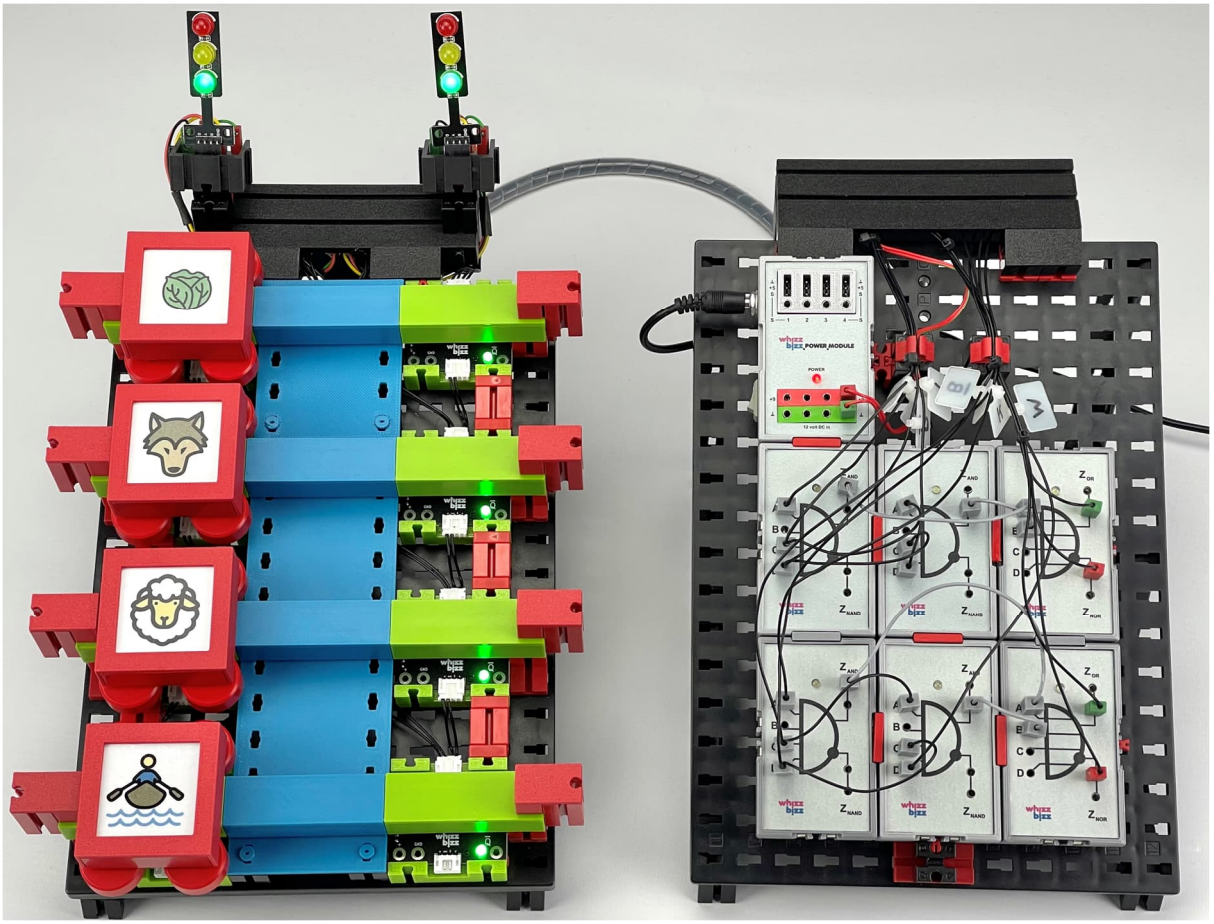


Abb. 8: Das vollständige Rätsel mit „Fluss“ und Silberlingen-Logik

Modular und beim Transport geschützt

Das komplette Modell ließ sich problemlos auf einer großen Grundplatte 1000 (39 cm × 27 cm) unterbringen, aber für den Transport habe ich den „Fluss“ und die Logik mit den Silberlingen auf zwei separaten Bauplatten 500 untergebracht. Da ich noch einige alte 31-polige Stecker herumliegen hatte, entschloss ich mich, ein eigenes Verbindungskabel herzustellen, so dass ein modularer Aufbau möglich wurde. Abb. 11 zeigt das Endergebnis für das „Rätselmodul“ mit den Mini-Ampel-LEDs. Die Kabelverbindung versorgt diesen Teil des Modells mit Strom und überträgt die verschiedenen Ufersignale von Bauer, Kohlkopf, Wolf und Schaf zum Silberling-Modul.

Diese Konstruktion ermöglichte es mir, die beiden Module gestapelt zu transportieren. Beim ersten Mal, als ich dies tat (auf der Zugfahrt zur Südconvention), hatten sich die Bodenplatten gelöst und leichte Schäden verursacht. Daher habe ich die Verankerung aus Standardbausteinen, die ich zunächst dafür verwendet hatte, durch die in Abb. 12 gezeigten selbst entworfenen grauen „Grabensteine“ ersetzt. Seitdem wurde das Modell mehrmals ohne Schäden transportiert.

Abschließende Worte

Der geteilte Aufbau auf zwei Grundplatten, die selbstgedruckten farbigen Bauplatten, die individuelle Gestaltung der „Wagen“, die Verwendung von Ampel-LEDs und die Erkennung mit Sensormodulen anstelle von Tastern sind meine persönlichen Entscheidungen. Nicht jeder wird so weit gehen

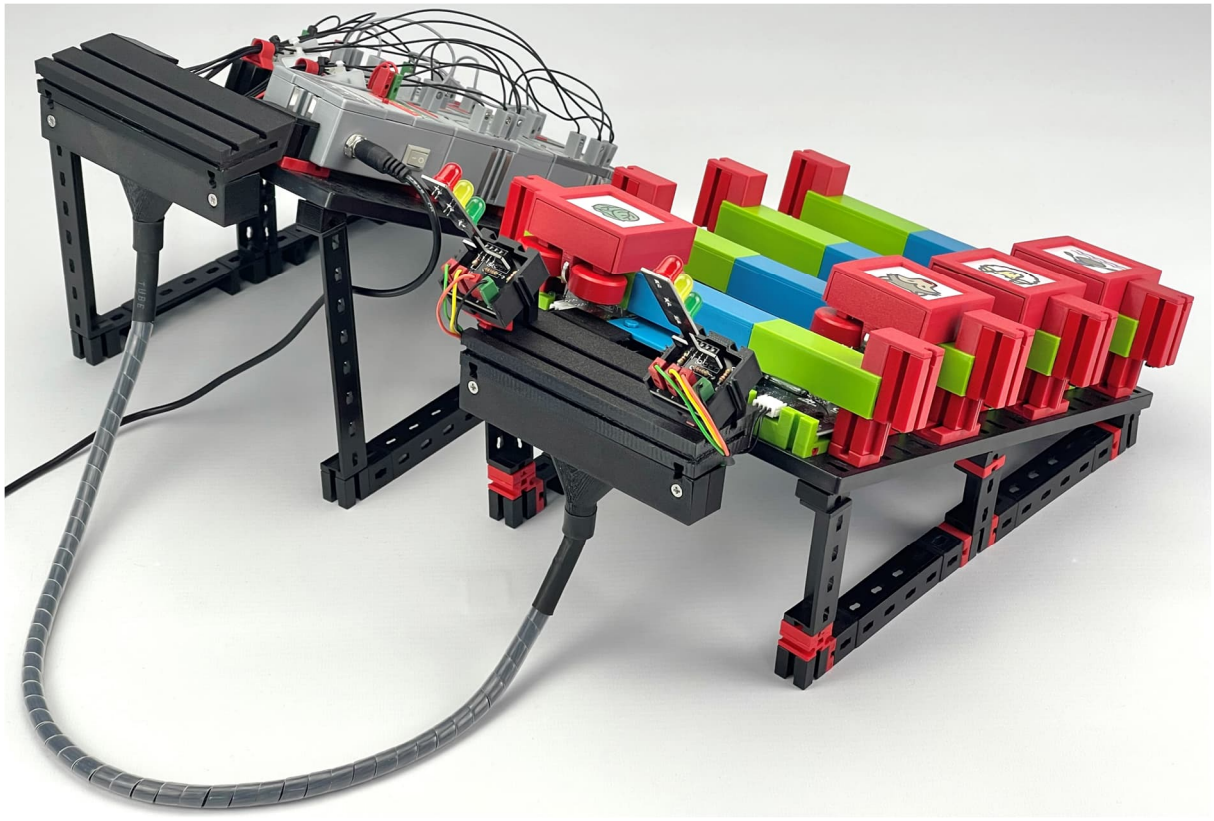


Abb. 9: Ein Blick hinter die Kulissen

können und wollen, um zusätzliche Anpassungen vorzunehmen. Zum Glück lässt sich das Modell auch mit Teilen aus dem Standardsortiment von fischertechnik hervorragend bauen. Ich bin gespannt, mit welchen Konstruktions- und Detektionslösungen andere die Überquerung des „Flusses“ in Zukunft lösen werden.

Zum Schluss noch ein Wort zu den Schafen: Ich bin auf viele Versionen des Rätsels gestoßen, in denen ihre Rolle durch eine Ziege ersetzt wurde. Um diesen Artikel zu einem vollwertigen zweiten Teil der von Stefan begonnenen Serie werden zu lassen, durfte die Ziege im Stall bleiben und ihre Rolle wurde von einem Schaf übernommen. Dem Wolf machte das wenig aus. Aufgrund des U-förmigen Passepartout-Bausteins (siehe Abb. 7) ist es jedoch einfach, eine „regionale Version“ des Puzzles zu erstellen, indem das Bild des Schafs durch das Symbol in Abb. 10 ersetzt wird.

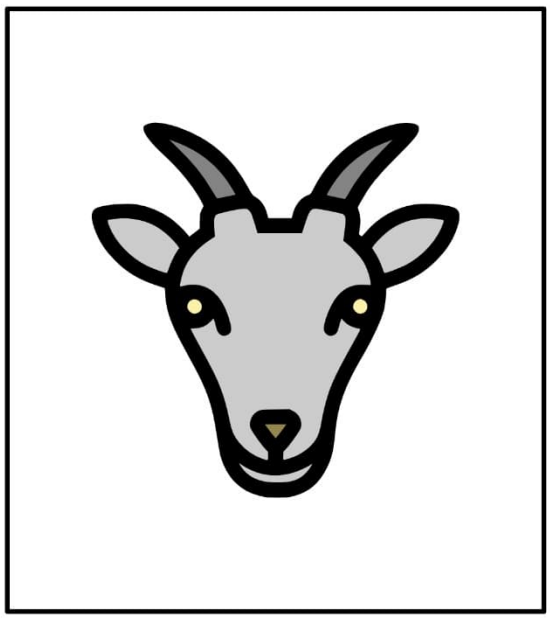


Abb. 10: Die Ziege bestand darauf, dass auch sie in diesem Artikel abgebildet wird!

Dieser modulare Aufbau macht es sogar einfach, eine Version des Rätsels zu erstellen, in der der Bauer einen Fuchs, ein Huhn und einen Sack Mais über den Fluss

bringen muss. Und auch die Version mit einem Fuchs, einer Gans und einem Sack Bohnen wird kein Problem darstellen.

Doch nun muss ich diesen Artikel beenden, denn ich sehe, dass sowohl die Ziege als auch das Schaf mit der Arbeit am Kohlkopf begonnen haben, während der Wolf sich bereits die Lippen leckt. Ich schließe daher mit dem Hinweis, dass weitere Informationen und Details auf der speziellen Projektseite [4] auf meiner Website zu finden sind. Hier gibt es auch einen Link zu einem Video, das das Rätsel in Aktion zeigt.

Quellen

[1] Stefan Falk: *Wolf, Schaf und Kohlkopf*. ft:pedia 1/2015, S. 50–57.

[2] Arnoud van Delden: *Sensor-Adaptermodul*. ft:pedia 1/2025, S. 41–49.

[3] Arnoud van Delden: *Ampelschaltungen*. ft:pedia 3/2024, S. 49–62.

[4] Arnoud van Delden: *The wolf, the sheep and the cabbage*. Seite über das Projekt auf wizzbizz.com.

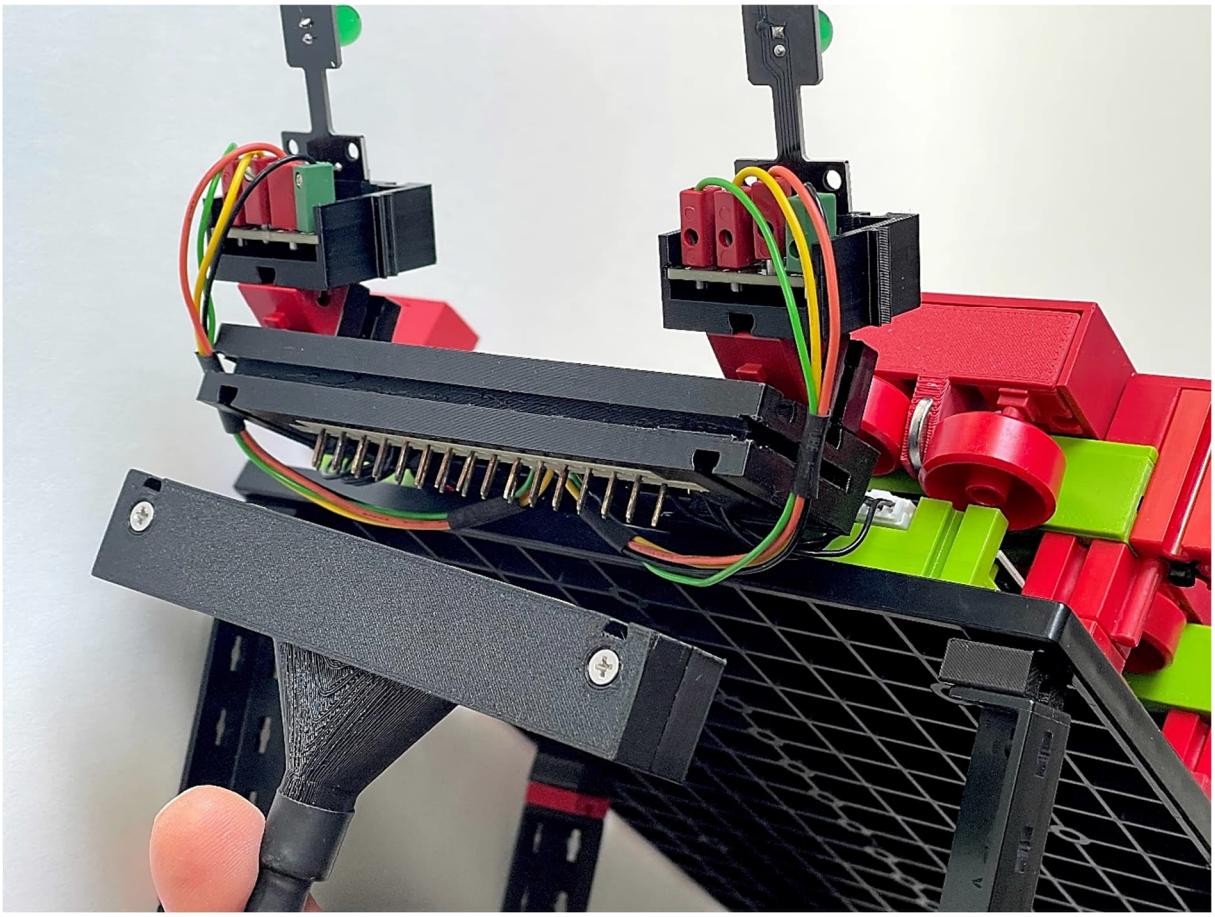


Abb. 11: Praktisch für den Transport: eine Kabelverbindung zwischen dem Fluss-Rätsel und der Silberling-Logik

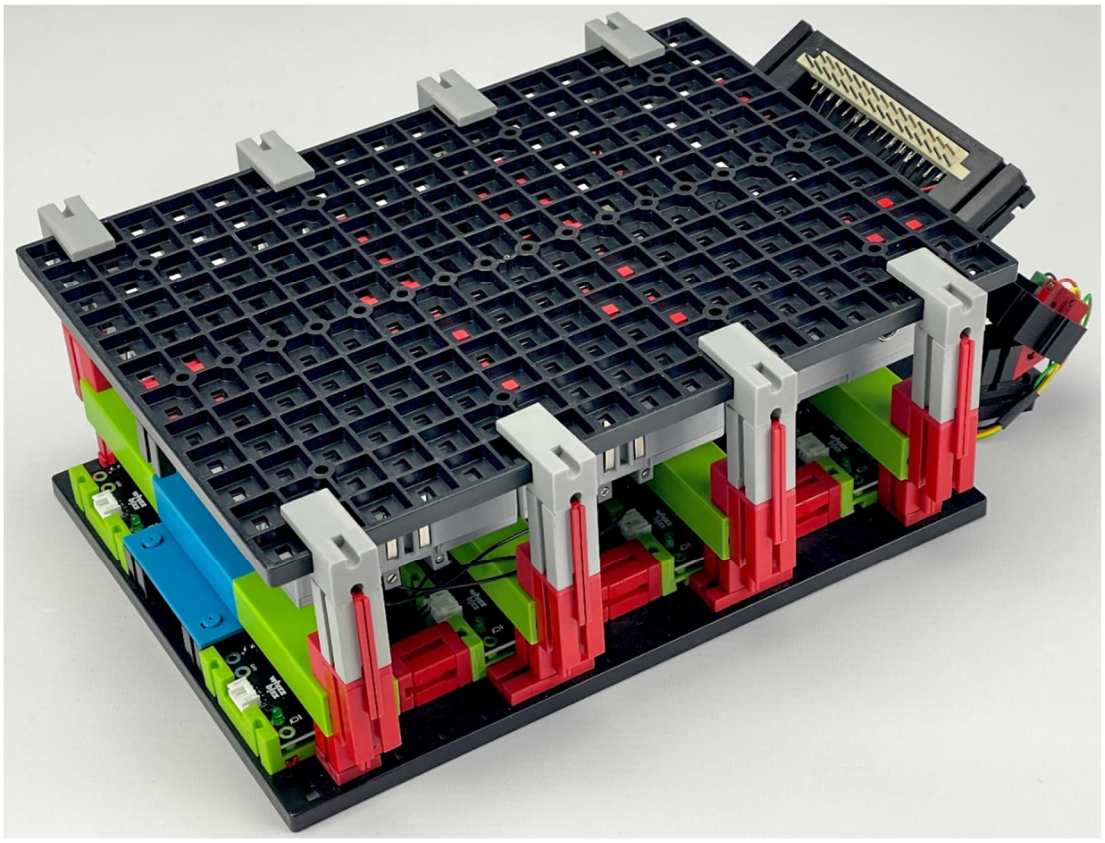


Abb. 12: Gestapelt kann das Modell geschützt transportiert werden

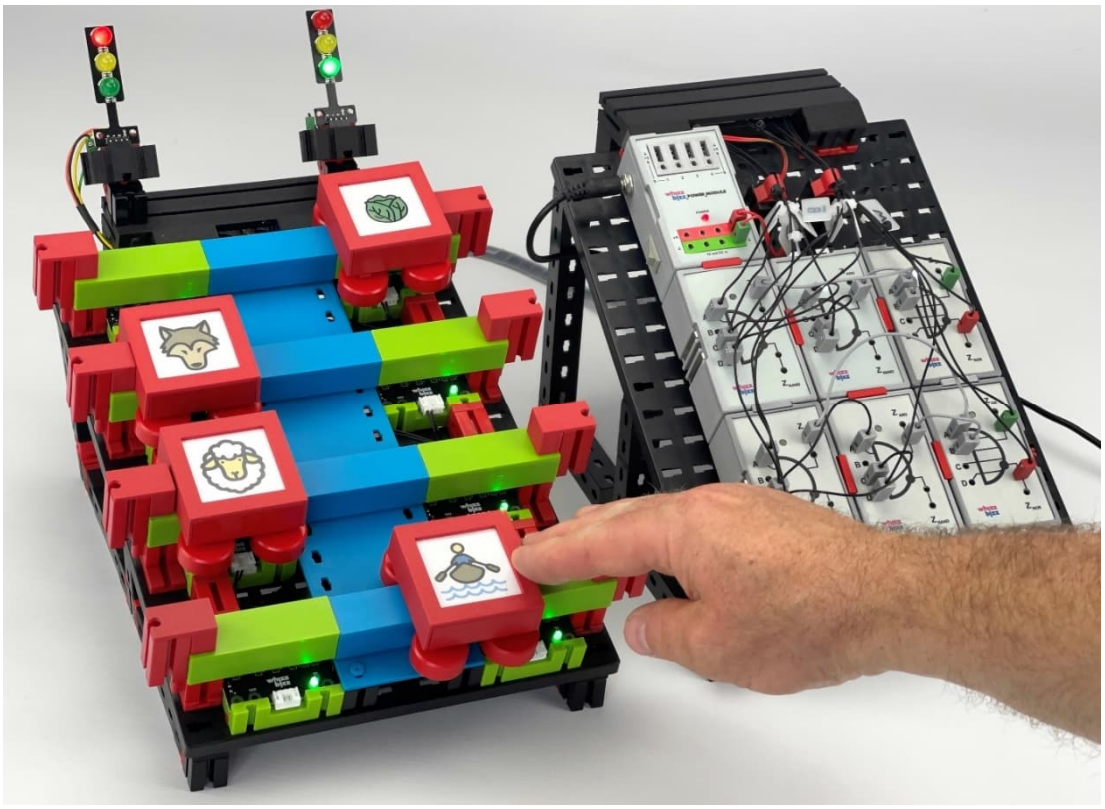


Abb. 13: Bevor der Bauer auf den Markt geht, kann er sein Problem mit einigen Silberlingen lösen

Modell

fischertechnik-Trainingsroboter 1985 Reloaded

Florian Bauer, Simon Bauer

Dieser Beitrag ist in Zusammenarbeit der Autoren entstanden, die aus der Diskussion im fischertechnik-Forum zustande gekommen ist. Wir stellen hier eine Variante des fischertechnik-Trainingsroboters von 1985 mit absoluten Winkelencodern, einem Seilzug-Aktuator und einem Servo-betriebenen Greifer vor.

Einleitung

Der fischertechnik-Trainingsroboter von 1985 (30572) ist ein beliebtes Modell, das zur damaligen Zeit bahnbrechend war. Der Roboter ist ein dreiachsiger Roboter mit zwei Armen auf einer drehbaren Plattform und einem Greifer, der über eine Synchron-Mechanik parallel gehalten wird. Die Positionssteuerung erfolgte optisch mittels Lichtschranken, welche Streifen-Encoder-Trommeln abtasten. Der Roboter konnte über C64 und viele andere Micro-Computer angesteuert werden. Abb. 1 zeigt das klassische Modell.

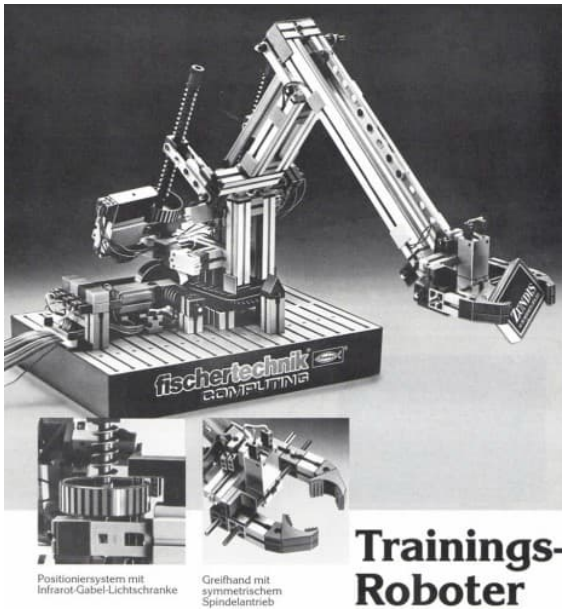


Abb. 1: Bild des originalen fischertechnik-Trainings-Roboter-Modells (aus [2])

Ein Vorteil des Trainingsroboters ist, dass er nicht so komplex ist wie ein sechsarmiger Roboter und daher mit einem moderaten Bauteileaufwand realisiert werden kann. Ein weiterer Vorteil ist, dass die Motoren weit unten liegen, sodass sie nicht so viel Gewicht bewegen müssen, was einerseits den Einsatz von Mini-Motoren (wie beim Ursprungsmodell) und andererseits Motoren von größerem Kaliber erlaubt. Die Motoren dienen dabei auch als Gegengewicht für Lasten, die der Arm aufgreift.

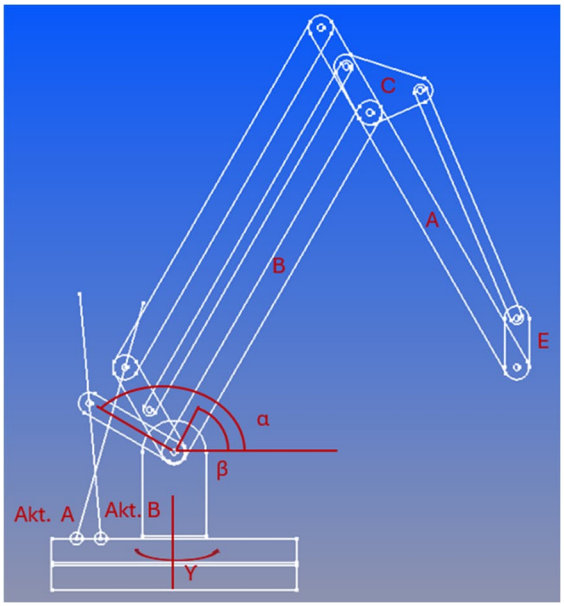


Abb. 2: Prinzipskizze der Kinematik: Akt. A: Aktuator für Arm A, Akt. B: Aktuator für Arm B, E: End-Effektor, C: Dreieck

Seine Kinematik ist sehr einfach. Eine Prinzip-Skizze ist in Abb. 2 zu sehen und eine sehr ausführliche, fast 100-seitige Beschreibung mit Aufbauanleitung und detaillierten Informationen zur Programmierung findet sich in [1].

Trotz oder vielleicht besser wegen seiner Einfachheit ist das Modell sehr gut für den Einstieg in die Robotik geeignet.

Von diesem Modell hat fischertechnik mehrere Nachfolger herausgebracht. Auf jeden Fall hat das Modell in der Vergangenheit zahlreiche Nachbauten und weitere Konstruktionen inspiriert, die mit unterschiedlichen Computern wie Amiga oder Arduino angesteuert wurden – wie z. B. [3, 4, 5, 6, 7, 17], um nur einige zu nennen.

Auch in dem Buch *Roboter mit fischertechnik* [3] ist eine Variante enthalten, die XS-Motoren verwendet und über einen Arduino angesteuert wird.

Das Modell

Die Begeisterung für diesen Roboter hat Simon zu einer neuen Variante inspiriert, die in diesem Beitrag vorgestellt wird.

Anstatt nur eines der Modelle nachzubauen, wollte Simon herausfinden, ob eine eigene Konstruktion trotz begrenzter Verfügbarkeit von Bauteilen möglich ist. Das fischertechnik-Baukastensystem sollte sich als so flexibel erweisen, dass eine alternative Lösung möglich war. In Abb. 3 ist sein Modell abgebildet. Man erkennt den Einsatz zweier XM-Motoren für die Arme und den Arduino MEGA mit dem Motor-Shield. Weiter unten sind Design-Ansichten, in denen die Details des Aufbaus besser erkennbar sind.

So wurden anstatt der im originalen Modell verwendeten Alu-Profile Statik-Träger mit Verstärkungen verwendet. Das „Dreieck“ für die Synchronmechanik des Greifers, wurde durch eine diskrete Konstruktion

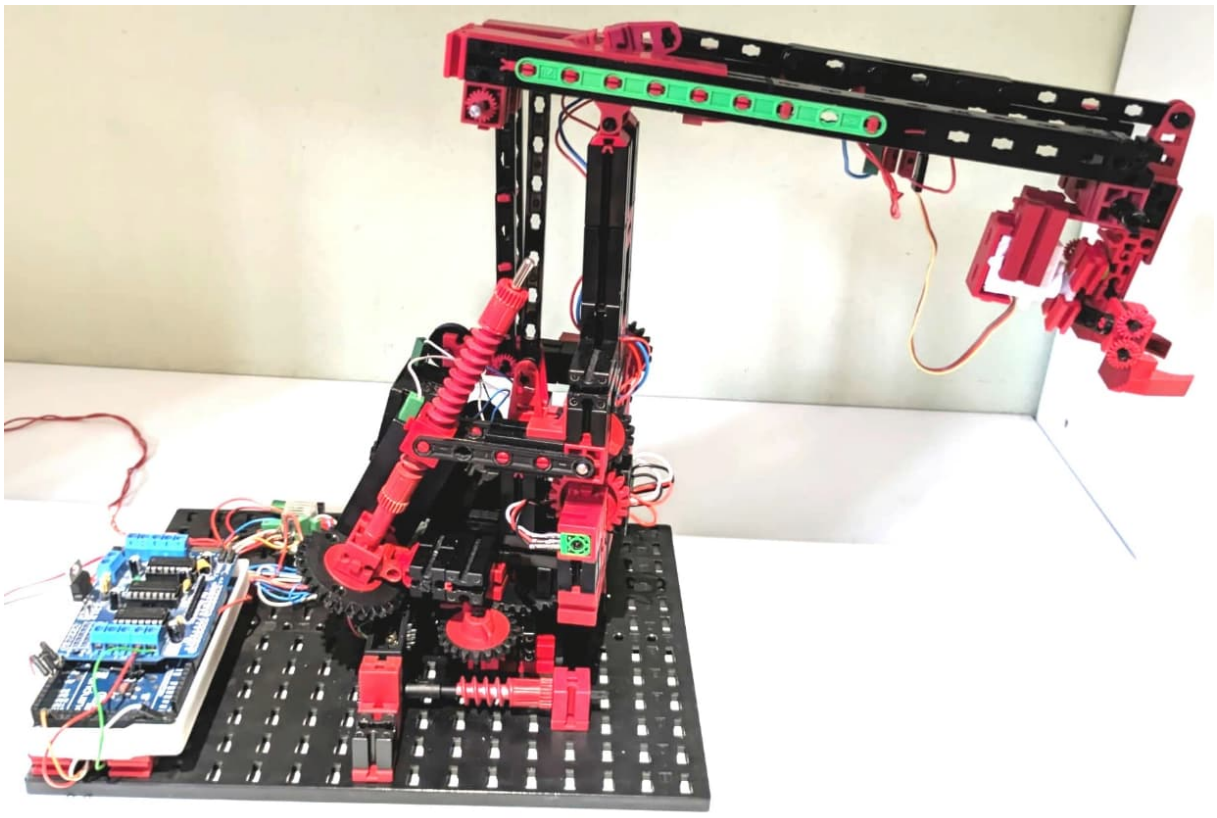


Abb. 3: Yet another Training Robot – Modell eines „alternativen” Trainingsroboters

ersetzt, die sich Simon von einem Modell im ftc-Bilderpool abgeschaut hat.

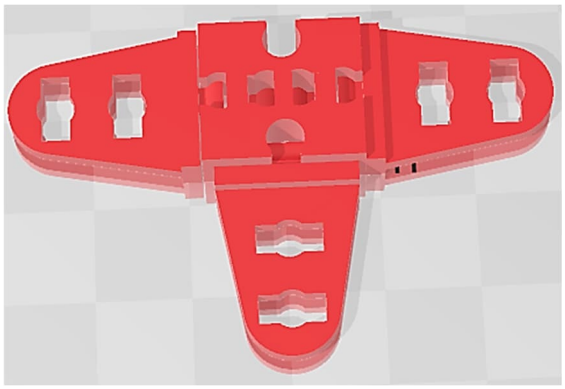


Abb. 4: „Dreieck“ für die Synchron-Mechanik

In Ermangelung eines Drehkranzes (31393) wurde ein Z40 (31022) verwendet, das auf glatten Verblendplatten läuft. In der ersten Version dienten Zahnräder und fischertechnik-Lichtschranken der Impuls-Zählung (siehe Abb. 5, [19]).

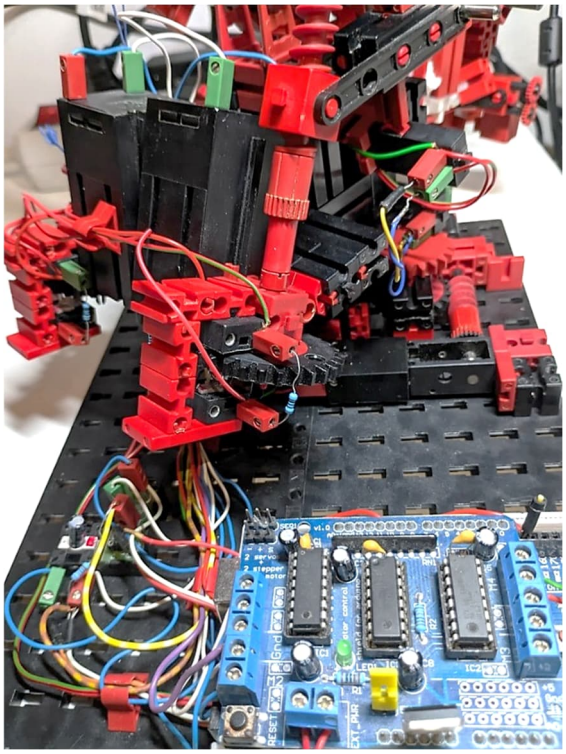


Abb. 5: Impulszählung mittels Lichtschranke und LED. Im Vordergrund zu erkennen: das L293D 4-Kanal-Motor-Shield

Da dies nur eine relative Positionierung erlaubt, mussten Nullpositions-Endschalter installiert werden, die bei einer Referenzfahrt vor dem Betrieb abgefragt werden müssen.

In einer verbesserten Variante wurden die Zahnrad-Encoder durch Potentiometer-Encoder ersetzt. Hierbei wurden Potentiometer aus ausgeschlachteten sg90 / MGXXX-Servos wiederverwendet, die vom Format praktischerweise exakt in die Statikträger 15 passen (siehe Abb. 6). Diese Potentiometer kann man aus Servos gewinnen oder über den Online-Handel beziehen.

Alternativ gibt es formatgleiche Potentiometer, die in Joysticks der PS4- und PS5-Spielkonsolen verbaut sind und die man leichter bekommen kann. Allerdings haben diese Potentiometer nur einen Arbeitsbereich von 90°, im Gegensatz zu den Servo-Potentiometer, die 180° abdecken. Auf [16] finden sich kompakte Halterungen für Standard-Trimmerpotentiometer, welche 270° abdecken.

Den abweichenden Arbeitsbereich solcher Potentiometer kann man leicht durch etwas andere Übersetzungen kompensieren.

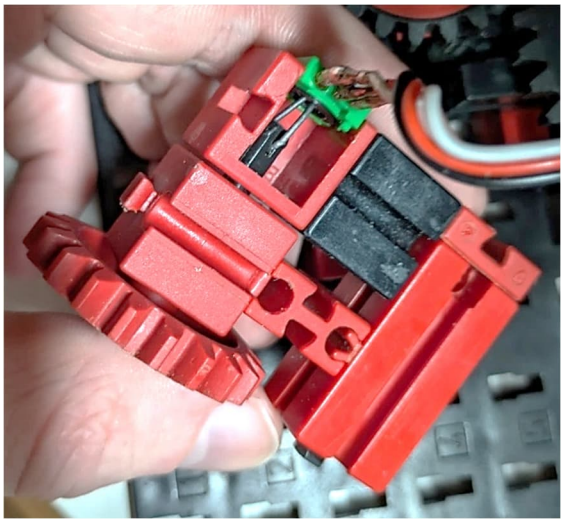


Abb. 6: Spielkonsolen-Sensor im Statikträger 15 mit Antriebs-Kupplung

Mit einem gebogenen Draht, einer „entkleideten“ Büroklammer, kann der Rotor des Potentiometers mit dem Schlitz einer Rastachse verbunden werden. Diese Konstruktion gleicht auch den minimalen Ver-

satz des Potentiometer-Rotors zur Bausteinmitte aus.

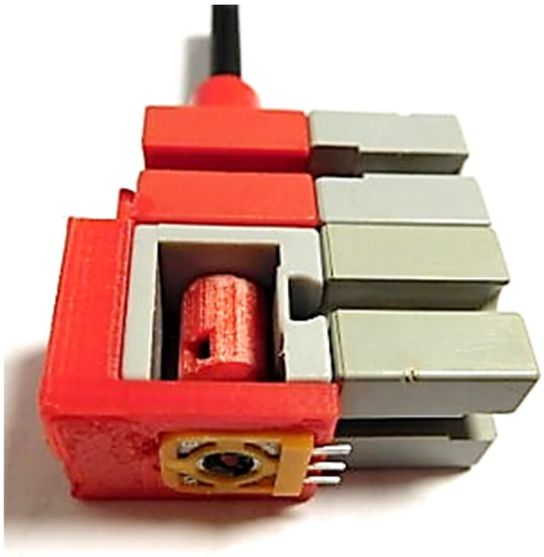


Abb. 7: Alternativer 3D-gedruckter Poti-Halter mit Drive-Bit

Wer Zugang zu einem 3D-Drucker hat, kann sich auch speziell dafür entworfene Teile herstellen, die perfekt zentriert sind. Potentiometer-Halter und „Drive-Bit“ sind in Abb. 7 zu sehen. Die STL-Dateien dazu sind unter [12] verfügbar.

Die Verwendung von Potentiometern als Positionssensoren hat mehrere Vorteile: Erstens erreicht man damit eine absolute Positionsmessung, die Null-Positionsschalter und die Referenzfahrt überflüssig macht. Zweitens ist die Messung kontinuierlich (bis auf die Diskretisierung durch den AD-Wandler) und die Auslesung kann mittels eines einfachen Analog-Reads im Arduino realisiert werden. Sehr vorteilhaft ist, dass die Winkelmessung direkt an den Gelenken erfolgt. Die im originalen Trainingsroboter erforderlichen Umrechnungen von Positionen der Linear-Aktoren in Winkelwerte erübrigt sich. Auf eine spezielle Besonder-

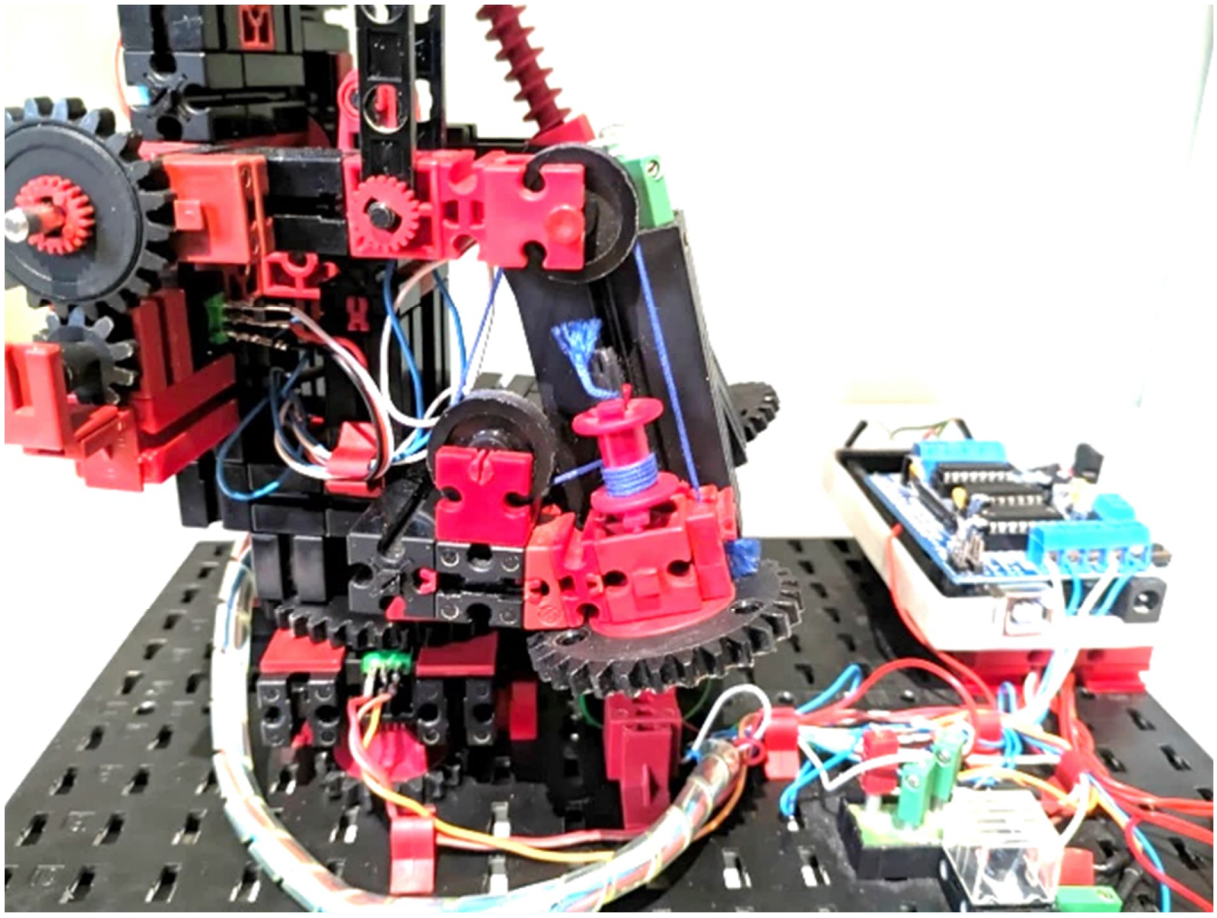


Abb. 8: Seilzug-Mechanismus für den Antrieb ein

heit für den Winkelaufnehmer für den vorderen Arm (Arm A in Abb. 2) sei hier hingewiesen: Da der Winkelbereich für Winkel α sehr gering ist, wird die Drehung des Antriebshebels über eine Zahnradkombination Z20 : Z10 übersetzt (siehe Abb. 8).

Der originale fischertechnik-Trainings-Roboter benötigte zwei schneckenbetriebene Linear-Aktuatoren. Was aber, wenn man nur Teile für einen Aktuator hat? Die naheliegende Lösung wäre sicherlich der Nachkauf der Teile. Aber es gibt auch andere Lösungen: Zum Beispiel kann man den zweiten Arm mittels eines Seilzugmechanismus mit Flaschenzug antreiben. In Abb. 11 kann man diesen Mechanismus sehen, den Simon in seinem Modell verwendet. Der Motor wickelt das Seil auf, um den vorderen Arm zu bewegen. Mit dem Flaschenzug wird einerseits eine Kraftverstärkung und andererseits eine weitere Untersetzung des XM-Motors erreicht. Das Gewicht des vorderen Armes hält das Seil immer gespannt.

Der ursprüngliche Roboter war mit einem speziellen Greifergetriebe mit Links- und Rechts-Gewinde ausgestattet (32342). Dieses Teil wird aber nicht mehr hergestellt und ist nur noch schwer aus zweiter Hand zu beschaffen. Dieser Mechanismus war sehr kompakt und konnte die beiden Greiferzangen parallel komplett schließen. Bei der Konstruktion des hier vorgestellten Greifmechanismus, die mehrere Anläufe erforderte, sollten diese beiden Anforderungen umgesetzt werden.

Dazu wurde ein kleiner Modellbau-Servo verwendet, der den Greifer über ein Gestänge öffnen und schließen kann. Der fertige Greifer ist in Abb. 9 zu sehen und in Abb. 10 die Rückansicht mit Blick auf den Mechanismus.

Der Greifer funktioniert folgendermaßen:

Beim Schließen des Greifers rotiert das Servo-Horn. Dabei entfernen sich die Punkte, an denen die Stangen montiert sind, von den Punkten, an denen die Gabeln des Greifers montiert sind, was zur Folge hat, dass die beweglich montierten Gabeln des Greifers zusammengezogen werden. Beim Öffnen des Greifers bewegen sich die Punkte wieder aufeinander zu, und da die Gabeln beweglich gelagert sind, öffnet der Greifer. Das erlaubt einen sehr kompakten Greifer. Damit die Zangen geführt werden und nicht einfach herumflattern, wurde mittels zweier Rastachsen und einigen Bausteinen 7,5 eine Führung gebaut, die funktional und kompakt ist.

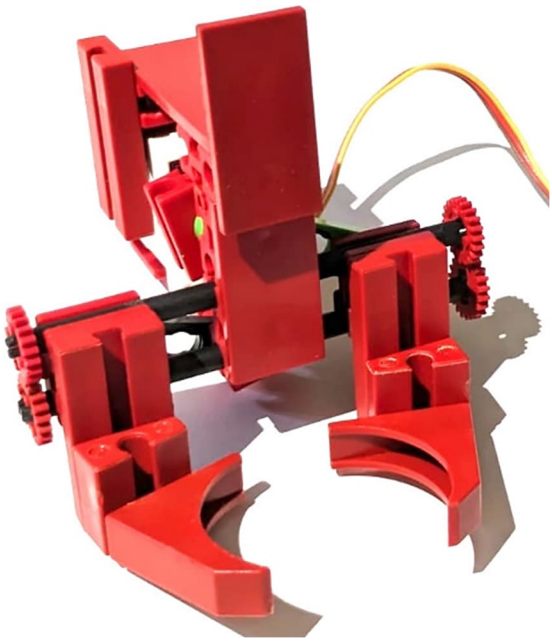


Abb. 9: Servo-betriebener Greifer

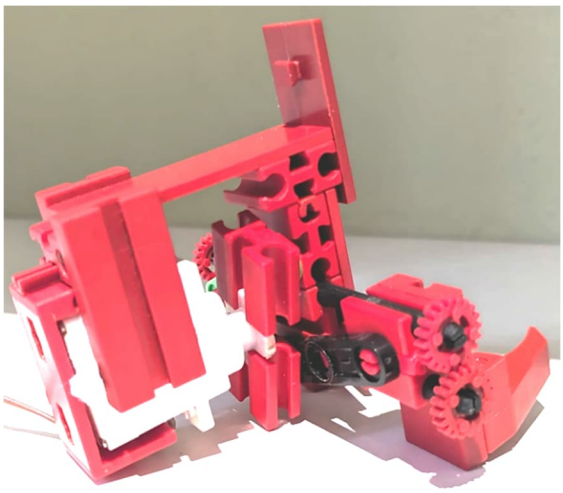


Abb. 10: Servogreifer – Rückansicht

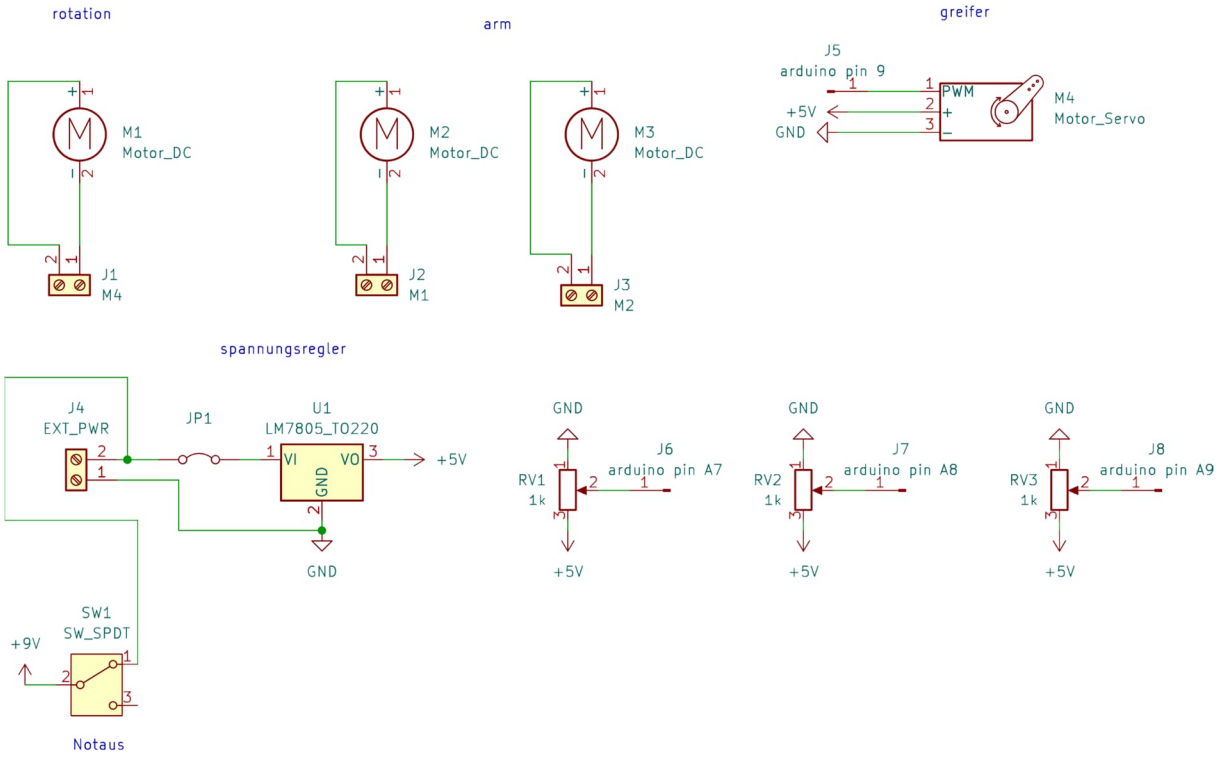


Abb. 11: Schaltplan für Potentiometer, Schalter und Motoren

Elektronik und Steuerung

Der Trainingsroboter wird über einen Arduino mit einem 4-Kanal-L293D-Motortreiber-Shield für den Arduino [9] angesteuert. Der Anschluss-Schaltplan für die Motoren, Potentiometer und Schalter ist in Abb. 11 zu sehen

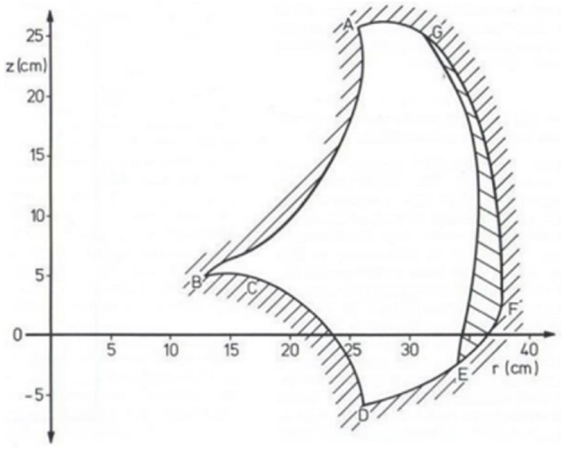


Abb. 12: Arbeitsbereich des Trainingsroboters (aus [1])

Auf dem GitHub-Repository von Simon Bauer [11] findet sich der Arduino-Code für die Ansteuerung. Es ist wichtig zu erwähnen, dass der Arbeitsbereich des Roboters durch seine Geometrie und speziell konstruktive Eigenheiten beschränkt ist. Das bedeutet, dass nicht alle Winkeleinstellungen erlaubt sind. Dies muss bei der Steuerung berücksichtigt werden.

Die Steuerung besteht aus zwei verschiedenen Programm-Teilen:

- Das Backend-Programm, welches im Hintergrund die Motorsteuerung übernimmt: Die Steuerung erfolgt für alle Motoren gleichzeitig.
- Das Frontend-Programm für die Ablaufsteuerung der Bewegungen: Dieses Programm legt fest, auf welche Positionen der Roboter fahren soll. Dieser Programmteil besitzt einerseits einen Automatik-Modus, in dem Bewegungsabläufe fest einprogrammiert sind. Andererseits übernimmt es auch die Interaktion mit dem Benutzer über den seriellen Port (aktuell eingestellt auf 115200 Baud). So hört es auf Eingaben der Arduino-Konsole und gibt perma-

nent die aktuellen Werte der Positions-Potis aus.

Um vom Automatikmodus in den manuellen Steuerungsmodus zu wechseln, so sendet man „s“ oder „k“ über den seriellen Port. Dann stoppen die Motoren, und man kann mit der Tastatur den Roboter steuern.

Die Trennung der Programmteile erlaubt einen einfachen Austausch des Front-Ends.

Die Positionssteuerung erfolgt basierend auf der Vorwärts-Kinematik: Man gibt dem Roboter Stellwerte für die Positions-Encoder vor. Das Motor-Steuerungs-Programm fährt, je nachdem, ob der Zielwert größer oder kleiner als der aktuelle Positionswert ist, in die vorgegebene Richtung. Ist die Position erreicht, wird der jeweilige Motor gestoppt. Vorsicht: Dabei wird derzeit noch nicht auf „verbotene“ Positionen geprüft.

Das Programm, welches sich auf Simons GitHub-Repository [10] befindet, ist eine kleine Demo des Bewegungsumfangs des Roboters.

Design

Das Design für diesen Beitrag wurden diesmal nicht mit dem ft-Designer, sondern mit *Microsoft 3D Builder* erstellt, einem von Microsoft zur Verfügung gestellten Tool zum Zusammenstellen von 3D-Modellen in einem proprietären Format von Microsoft (3MF, das erlaubt auch farbige Modelle, Einzelteile usw.) und im STL-Format. Es besitzt auch eingeschränkte Möglichkeiten, STL-Dateien zu erzeugen. Abb. 13 und 14 zeigen die Seitenansichten des Modells. Die STL-Files und weitere Aufnahmen des Modells finden sich auf Simons GitHub-Repository [11]. Der 3D Builder ist ein ausgezeichnetes Werkzeug, mit dem das Modell untersucht werden kann und sehr hilfreich, um das 3D-Robotermodell virtuell zu erkunden.

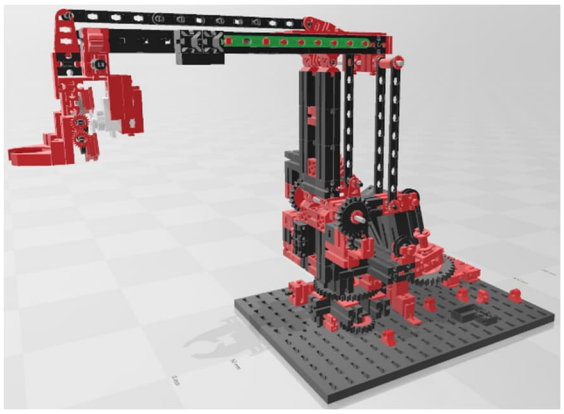


Abb. 13: Visualisierung des Modells im 3D-Designer – Seitenansicht rechts

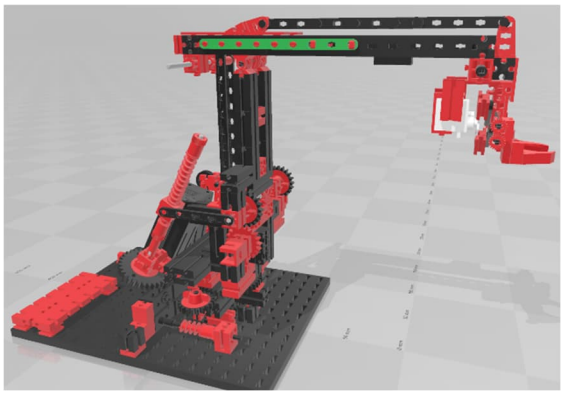


Abb. 14: Visualisierung des Modells im 3D-Designer – Seitenansicht links

Ausblick

Mit Sensoren direkt an den Gelenken könnte der Trainingsroboter ohne Motoren aufgebaut werden und als Eingabegerät für das originale Modell eingesetzt werden, das dann aufgezeichnete Bewegungsabläufe nachfährt. Statt des Greifers könnte man einen Taster vorsehen, um einen Greif-Vorgang aufzuzeichnen.

Der Trainingsroboter eignet sich aber auch zum Ausprobieren anderer Sensoren.

Man könnte eine absolute Positionsmessung auch durch magnetische Positions-Sensoren bewerkstelligen. In Abb. 15 ist ein Positionsgeber mit einem AS5600-Magnet-Encoder zu sehen. Dieser Sensor arbeitet berührungslos und wird über I²C ausgelesen.

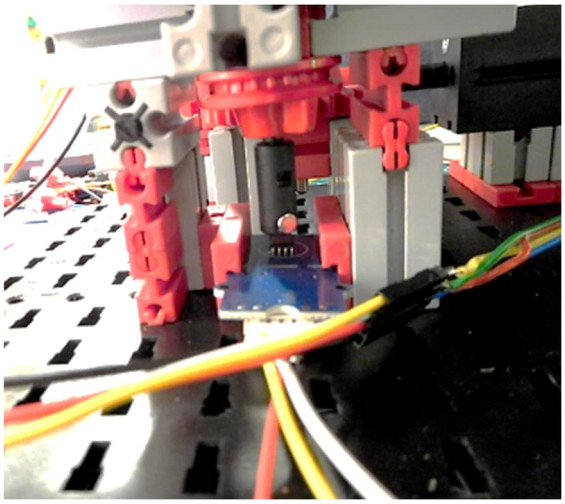


Abb. 15: Positionsgeber mit AS5600 in einer „höher gelegten“ Version des Trainingsroboters

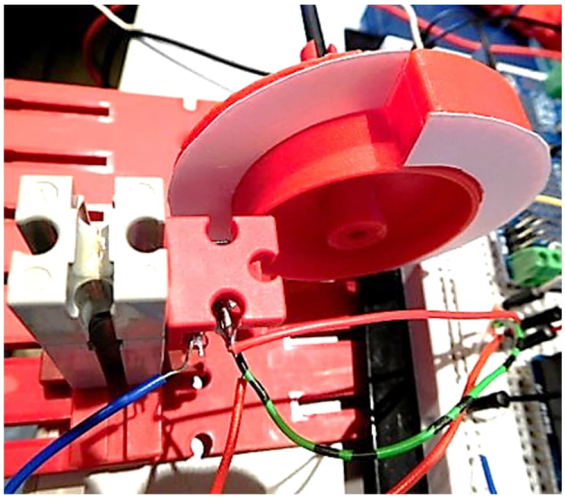


Abb. 16: Alternative Positions-Messung mit CNY70 und Encoderschnecke

Als weitere Möglichkeit könnte die Positionsmessung auch über eine Reflex-Lichtschranke CNY70 realisiert werden, die den Abstand zu einer Encoder-Schnecke mit einer Steigung von 10 mm pro Umdrehung misst (Abb. 16). Da das Mess-Signal nichtlinear vom Abstand abhängt, muss das Sensorsignal kalibriert werden (siehe Abb. 17). Dieser Encoder erfordert aber mehr Platz. Der Vorteil ist, dass er verschleißfrei arbeitet.

Darüber hinaus könnte auch ein anderer End-Effektor, wie anderswo schon gezeigt, ausprobiert werden: zum Beispiel ein Greifer mit Vakuum-Sauger (133028), ein Greifer mit Elektromagnet (31324) – wie bereits als Option des Originalmodells – oder ein Bionischer Gripper (Flex-Greifer) von Festo, den Joe Hanson und Stephan Kallauch für fischertechnik adaptiert haben [13, 20].

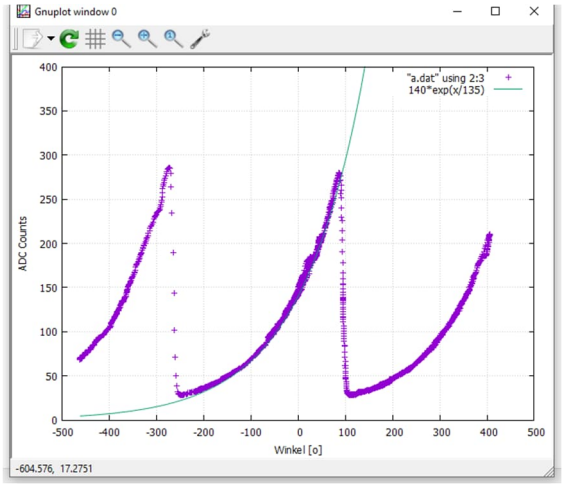


Abb. 17: Nichtlinearer Abhängigkeit des Sensor-Signals vom Winkel

Zusammenfassung

Der Beitrag beschreibt eine weitere Interpretation des fischertechnik-Trainingsroboters von 1985 – man könnte sagen „yet another training robot 1985“. Wir haben vorgestellt, wie man kompakte Servo-Potentiometer integrieren, in Ermangelung einer zweiten Schnecke stattdessen einen Seilzugaktuator nutzen und einen kompakten Servo-betriebenen Greifer bauen kann. Wie dieser Beitrag zeigt, führen manchmal viele Wege zum Ziel.

Fazit: Ein Mangel an Teilen muss nicht unbedingt ein Hindernis darstellen und das fischertechnik-System fördert die Kreativität, alternative Lösungen zu finden.

Die Begeisterung dafür war jedenfalls so ansteckend, dass dieser „Joint-Venture-Beitrag“ zustande gekommen ist. Weitere Aufnahmen des Modells könnt ihr in [14] finden. Die Weiterentwicklung dieses Roboters könnt ihr im fischertechnik-Forum verfolgen. Dort könnt ihr Fragen zum Roboter stellen oder Euch an der Diskussion beteiligen [18].

Quellen

[1] fischertechnik: *Bauanleitung Trainingsroboter*. Auf docs.fischertechnikclub.nl, 1985.

[2] fischertechnik: *Trainings-Roboter.* Produkt-Information auf docs.fischertechnik-club.nl, 1985.

[3] Dirk Fox, Thomas Püttmann: Material zu *fischertechnik-Roboter mit Arduino* (dpunkt.verlag). Auf fischertechnik-roboter-mit-arduino.de.

[4] Marspau: *Training Roboter II*. Auf ft-community.de, 2011.

[5] Andreas Gürten: *TrainingsRoboter von 1985*. Auf ft-community.de, 2008.

[6] PreRetro: *30572 1985 Trainingsroboter Fischertechnik Training Robot Amiga 2000*. Auf YouTube, 2023.

[7] Joe Hanson: *fischertechnik teach-in robot / StepFish demonstration.* Auf YouTube, 2019.

[8] x3picknicker: *Fischertechnik Roboter von 1992 gesteuert durch C64*. Auf YouTube, 2023.

[9] Fouad_Roboticist: *Arduino Servo Motor Control with Motor Driver Shield L293D*. Auf projecthub.arduino.cc, 2019.

[10] Microsoft: *3D Builder.* Auf apps.microsoft.com.

[11] Simon Bauer: *ft-trainingsrobot*. Auf github.com.

[12] Florian Bauer: *Potentiometer Holder and Drive Bit for Fischertechnik*. Auf thingiverse.com, 2025.

[13] Jan Hanson: Antwort auf *Bionischer Gripper von Festo.* Im ftc-Forum, 2025.

[14] Simon Bauer: *XG BC*. Auf flickr.

[15] Let's print 3D: *Download Microsoft 3D Builder for Windows in 2025: Complete Guide*. Auf letsprint3d.net.

[16] Jan Hanson: *fischertechnik angular position encoder / potentiometer*. Auf Thingiverse, 2019.

[17] Jens Lemkamp: *Traingsroboter mit Standardmaterialien.* Auf ftcommunity.de, 2014.

[18] Simon Bauer (XG BC): *1986er (1985er) Trainingsroboter-meine Version*. Foren-Thread auf ftcommunity.de, 2025.

[19] Dirk Fox, Johann Fox: *Impuls-messung mit dem TX(T)*. ft:pedia 2/2017, S. 36–40.

[20] Stephan Kallauch: *Der adaptive Greifer*. ft:pedia 4/2024, S. 9–14.

Computing

Getting the Analog Input to Work on the fischertechnik Universal and CVK Interface

Volker Paelke

Due to documentation mismatches, pin numbering inconsistencies, and quirks in the circuit design it can be tricky to use the analog input on legacy fischertechnik interfaces like the Universal and CVK. The article shows how to avoid these traps…

Introduction

While there are many articles in ft:pedia explaining the function of fischertechnik interfaces, including their use with modern devices like microcontrollers, practical guidance for using the analog input remains limited. A few years ago, I got a CVK interface working with a Schneider CPC, an Atari ST, and an Arduino Pro Micro. I demonstrated it as a controller for the teach-in robot from the original fischertechnik computing kit at several Nordconventions. In all these cases, the analog input relied on the original interface circuitry. Since then, I've been contacted by several users encountering difficulties in getting the analog input to work. In this short article, I aim to share some debugging advice and a minimal test setup to help get the analog input operational.

Background

In addition to 8 digital inputs and 4 motor outputs, the original Fischertechnik interfaces also provides two analog inputs (EX, EY), which are designed to measure variable resistors such as potentiometers or photoresistors in the range of 0 to 5 kΩ. The analog-to-digital conversion (ADC) is handled either by the host computer (as with the Apple II) or by a monostable timer circuit within the interface itself. Each ADC circuit (for EX and EY) uses one half of a NE556 timer, which is functionally equivalent to two 555 timers.

This article focuses on the Universal and CVK interfaces, specifically the ADC functionality using the single (Centronics) DATA-IN line available via the printer port which is used with systems like the Schneider/Amstrad CPC, IBM PC, Atari ST, and Commodore Amiga.

Operation

The monostable timers (monoflops) in the interface are triggered by the control signal TRIGGER-X (or TRIGGER-Y). The time it takes for a capacitor to charge to the threshold voltage depends on the resistance between the EX (or EY) input and +5V. On the Universal and CVK interfaces, the capacitor voltages are routed to the connector as ‘Poti X’ and ‘Poti Y’. These can be directly used for ADC on systems that have internal ADC capabilities. To use the internal circuitry for ADC, it is essential to bridge Poti X to RC-X and Poti Y to RC-Y at the external connector (the original adapter boards include these bridges for systems that use the interface's circuitry).

The NE556 generates a pulse that remains high from the moment it's triggered until the

capacitor reaches the threshold. The outputs from the EX and EY circuits are OR'ed together, and the result is passed through a second OR gate which combines the digital input shift register output and the analog pulse output. This signal then goes through an inverting amplifier and is sent to the Centronics DATA-IN pin. Since only one signal is read at any time there is only one data source and there should be no contention issues. The result is that after a trigger, DATA-IN goes low (!) for a time that depends on the connected resistor, and then returns to high. By measuring this pulse width, the input value can be quantified.

Common Challenges

Pin Numbering Confusion

The pin numbers on the PCB of the interface are mirrored with respect to the connector, which can be misleading as the PCB numbers are used in most documentation (e.g. the 'ft66843_schematic.pdf' on the FTC website). For example, the red wire on the ribbon band cable connects to '20' on the PCB but corresponds to Pin 1 on the connector plug. The image (Fig. 1) shows the pinout at the connector plug at the end of the cable coming from the interface using the standard convention with "1" marked by the red wire and a small triangle on the connector.

Labeling Errors

Possibly due to this mirroring many documents (including some ft:pedia articles) mistakenly swap the labels for RC-X/Poti X/Trigger X and RC-Y/Poti Y/Trigger Y. Fig. 1 also corrects this error.

GND Connection

It is important to ensure that the computer and interface share a common ground.

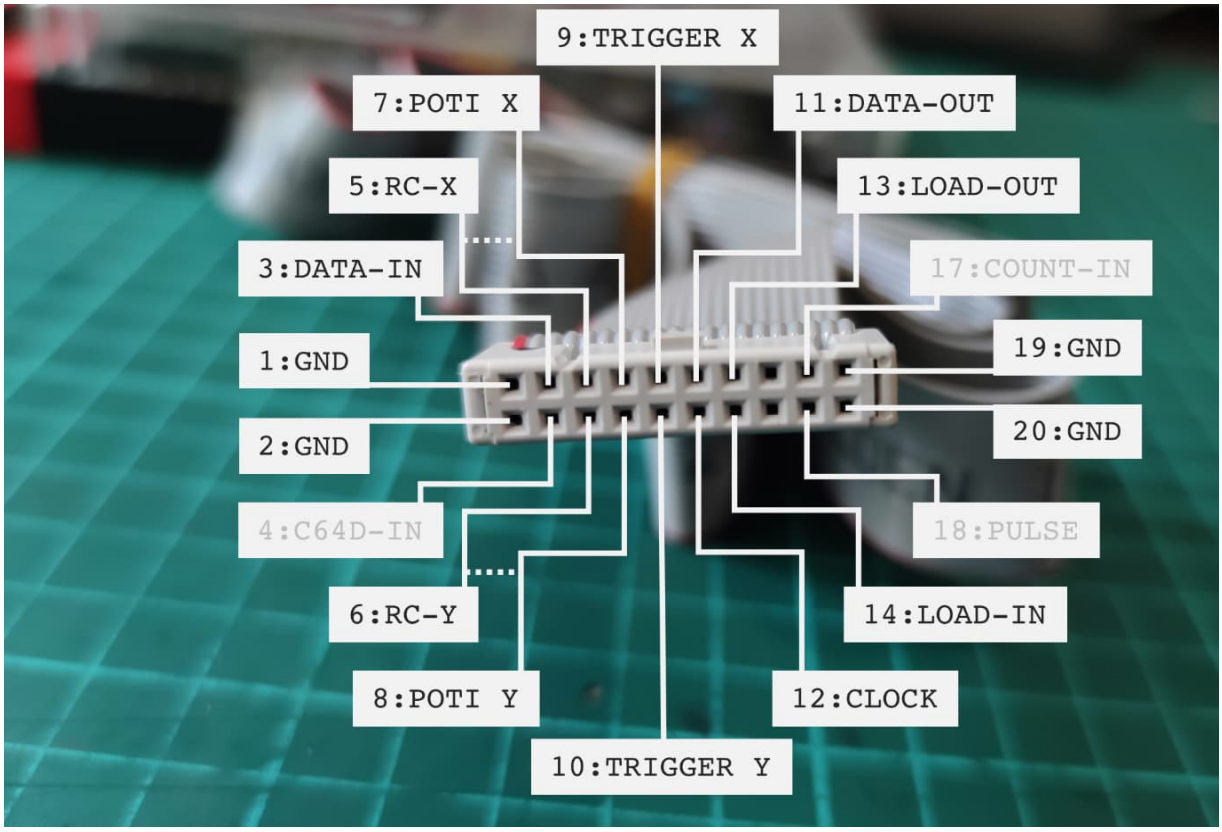


Fig. 1: Pinout at the connector plug of the fischertechnik Universal Interface

Crosstalk in the NE556

Although logically the NE556 contains two separate 555 timers, internal connections can cause crosstalk. If only one input is used, the other should be wired to +5V to prevent interference.

A Simple Test Setup

A 5V microcontroller like an Arduino is a simple way to test the analog input functionality. In the example I used a CVK Interface, an Arduino Mega 2560, and a potentiometer on the EX input.

Wiring

The setup is wired as depicted in Figs. 2-4.

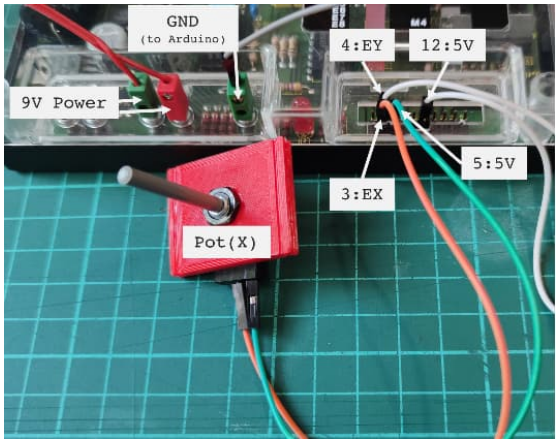


Fig. 2: Wiring at the fischertechnik Universal Interface (Power and Sensor)

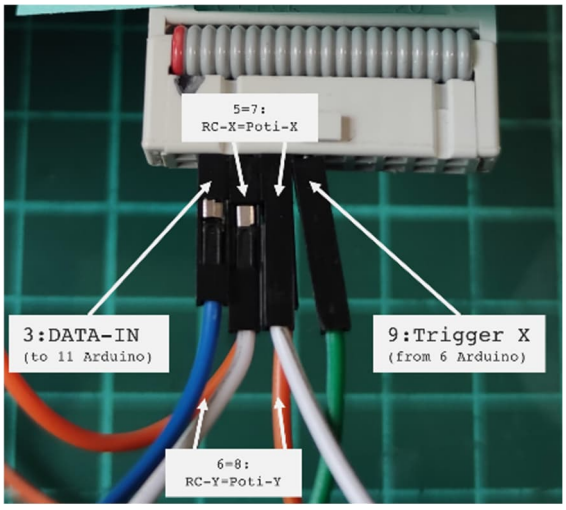


Fig. 3: Wiring the Arduino to the Interface to drive and read the EX input

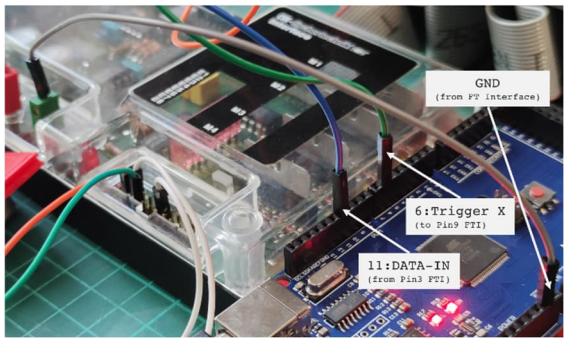


Fig. 4: Connecting the Arduino to the Interface

Interface power supply

- 9V supply to + and GND
- GND connected to Arduino GND

Sensor connection

To minimise the wiring the potentiometer for EX and the +5V connection for EY were directly wired at the model connector on the interface using Dupont jumpers (pin numbers in parentheses). It is, of course, also possible to use the original wireharness.

- EX (03) → potentiometer → +5V (05)
- EY (04) → directly to +5V (12)

Control connection

This numbering refers to the connector plug at the end of the ribbon cable coming from the interface (Pin 1 = red wire = marked with little arrow):

- Pin 3 → DATA-IN (blue wire in Fig. 3 → Arduino Pin 11)
- Pins 5–7 → bridge (white wire in Fig. 3, analog X)
- Pins 6–8 → bridge (orange wire in Fig. 3, analog Y)
- Pin 9 → TRIGGER-X (green wire in Fig. 3 → Arduino Pin 6)

Driver Code

The listing shows a minimal Arduino sketch that reads values from a potentiometer on

the EX input and writes them to the Serial Monitor in the Arduino IDE. Changing the potentiometer should vary the displayed value somewhere between ~ 20 and a few 100, depending on the potentiometer and the speed of the microcontroller. The timeout value in the counting loop might have to be adjusted. The sketch will be available for download on the ft:pedia downloads page.

Required connections:

- GND Arduino – GND FT Interface
- Bridge RC-X – Poti X (5 + 7 on Interface Connector)
- Bridge RC-Y – Poti Y (6 + 8 on Interface Connector)
- Arduino Pin 11 – Pin 3 on Interface Connector = Data-IN
- Arduino Pin 6 – Pin 9 on Interface Connector = Trigger-X

Arduino Pin No = Centronics Pin numbers

Numbering refers to Interface Connector at the end of the cable (mirrored from PCB). Connect both EX and EY (to +5V if not measuring) to avoid crosstalk problems.

```
const int datain = 11;
const int triggerx = 6;
long int testval;

void setup() {
  Serial.begin(9600);
  Serial.println("ANALOG X Test started");
  pinMode(datain, INPUT);
  pinMode(triggerx, OUTPUT);
}

long int get_analogx() {
  long int cnt = 0;
  digitalWrite(triggerx, LOW);
  digitalWrite(triggerx, HIGH);
  while (!digitalRead(datain)) {
    // Pulse Inverted at Centronics Data-IN!
    cnt++;
    if (cnt >= 1000) break;
    // timeout, depends on speed of the controller, adjust
  }
  return (cnt);
}

void loop() {
  testval = get_analogx();
  Serial.println("ANALOGX:");
  Serial.println(testval);
  delay(100);
}
```

Listing 1: Analog Test (Arduino Sketch)

Conclusion

Using the analog input on legacy fischertechnik interfaces like the Universal and CVK can be tricky due to documentation mismatches, pin numbering inconsistencies, and quirks in the circuit design. However, with a correct wiring setup and awareness of the internal operation, it's entirely possible to revive these classic interfaces for use with both vintage systems and modern microcontrollers.

Computing

Kommerzielle Arduino-Shields

Jeroen Regtien

Die Verwendung eines Arduino Uno oder Mega als Controller mit den älteren Parallel- oder Universal-Interfaces (30520) und den seriellen Schnittstellen (30402) als I/O- oder Motorshield wurde in mehreren ft:pedia-Beiträgen beschrieben [1, 2, 3]. Dieser Beitrag stellt drei kommerziell erhältliche I/O-Shields vor, die mit dem Arduino Nano, Uno und Mega verwendet werden können. Dabei handelt es sich um die Didacta Uno und Mega Shields, das ftNano und das Adafruit Motorshield V2.3.

Einleitung

Warum interessieren alternative Schnittstellen oder Shields von Drittanbietern? Schließlich sind die moderneren Controller TXT, RX, TXT 4.0 und der ftDuino leistungsstarke und vielseitige Controller.

Obwohl ich diese auch verwende, bin ich nach wie vor an Alternativen interessiert, die im Bildungsbereich verwendet werden. Das Arduino-Ökosystem hat zudem ein niedriges Einstiegsniveau, eine gut entwickelte Entwicklungsumgebung (IDE), eine große Suite kompatibler Sensoren und Add Ons, eine enorme Nutzerbasis mit informativen Online-Foren – und es entwickelt sich ständig weiter. Ein weiterer Grund ist meine persönliche Abneigung gegen grafische Programmiersprachen. Das sagt wahrscheinlich mehr über mich aus als über diese Programmiermethoden; ich programmiere am liebsten in C, C++, Python und Swift.

Interessanterweise regt fischertechnik mit der neuen Modellreihe „Maker“ auch Bastler zum Einsatz von Mikrocontrollern (Arduino) oder Mikrocomputern (RPi) an.

Arduino

Das Arduino-Ökosystem bedarf kaum einer Einführung; die Eignung und die Spezifikationen wurden in mehreren früheren ft:pedia-Beiträgen beschrieben, darunter [4, 6].

Es gibt jedoch ein paar Dinge, die man beachten sollte. Ein Arduino kann nicht zur direkten Ansteuerung von fischertechnik-Aktuatoren verwendet werden, da es Strom- und Spannungsbegrenzungen gibt. Beim Arduino Uno sind das z. B. 5V und 20mA pro IO-Pin oder 200mA für alle Pins zusammen. Das ist für LEDs noch in Ordnung, doch die fischertechnik-Glühbirnen nehmen ~80-100mA auf und die verschiedenen Motoren ~50-200mA, insbesondere wenn sie stark belastet oder blockiert sind.

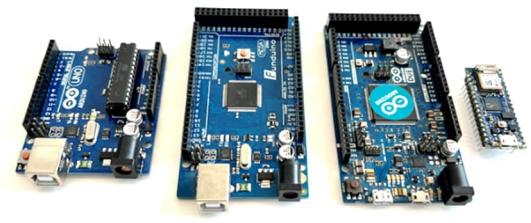


Abb. 1: Arduino Uno, Mega Clone, Arduino Due und Arduino Nano (v.l.n.r.)

Shields

Daher muss zwischen den Ausgangspins des Arduino und den Aktuatoren eine elektronische „Zwischenschutzschicht“ eingesetzt sein. In der Arduino-Sprache

wird diese als „Motorshield“ bezeichnet: Es enthält Motortreiber, H-Brücken oder Relais. Dabei handelt es sich um eine separate Platine mit eingebetteten elektronischen Komponenten, an die der Arduino angeschlossen wird. Dies kann auch durch eine vollständige Integration wie beim ftDuino [5] erfolgen. Separate Shields werden über lange Header-Pins mit dem Arduino verbunden.

In ihrem Buch „Bauen, erleben, begreifen: fischertechnik-Roboter mit Arduino“ [6] haben Dirk Fox und Thomas Püttmann bereits den Einsatz des kommerziell erhältlichen Adafruit Motorshields beschrieben. Darüber hinaus gibt es einige ft:pedia-Beiträge, die auf diesem Buch aufbauen und zusätzliche Steuerungsoptionen vorstellen [7, 8].

Lösungen bietet auch das kroatische Unternehmen Didacta [9] an, das sich auf die Entwicklung, die Produktion und den Vertrieb innovativer Lösungen für Robotik und MINT-Bildung konzentriert. Die meisten der Produkte sind fischertechnik-kompatibel.

Es werden Shields für den Arduino Uno und Mega, den Raspberry Pi und den BBC micro:bit angeboten. Zusätzlich gibt es noch ein Shield für den Arduino Nano, das von Florian Schäffer entwickelt und produziert wurde [10].

Es gibt auch andere Motorshields für den Arduino, wie z. B. das L293D-Shield oder das Arduino Motorshield, aber für fischertechnik-Anwendungen sollten Shields sowohl die Ein- als auch die Ausgänge abdecken und sich nicht nur auf Motorausgänge beschränken.

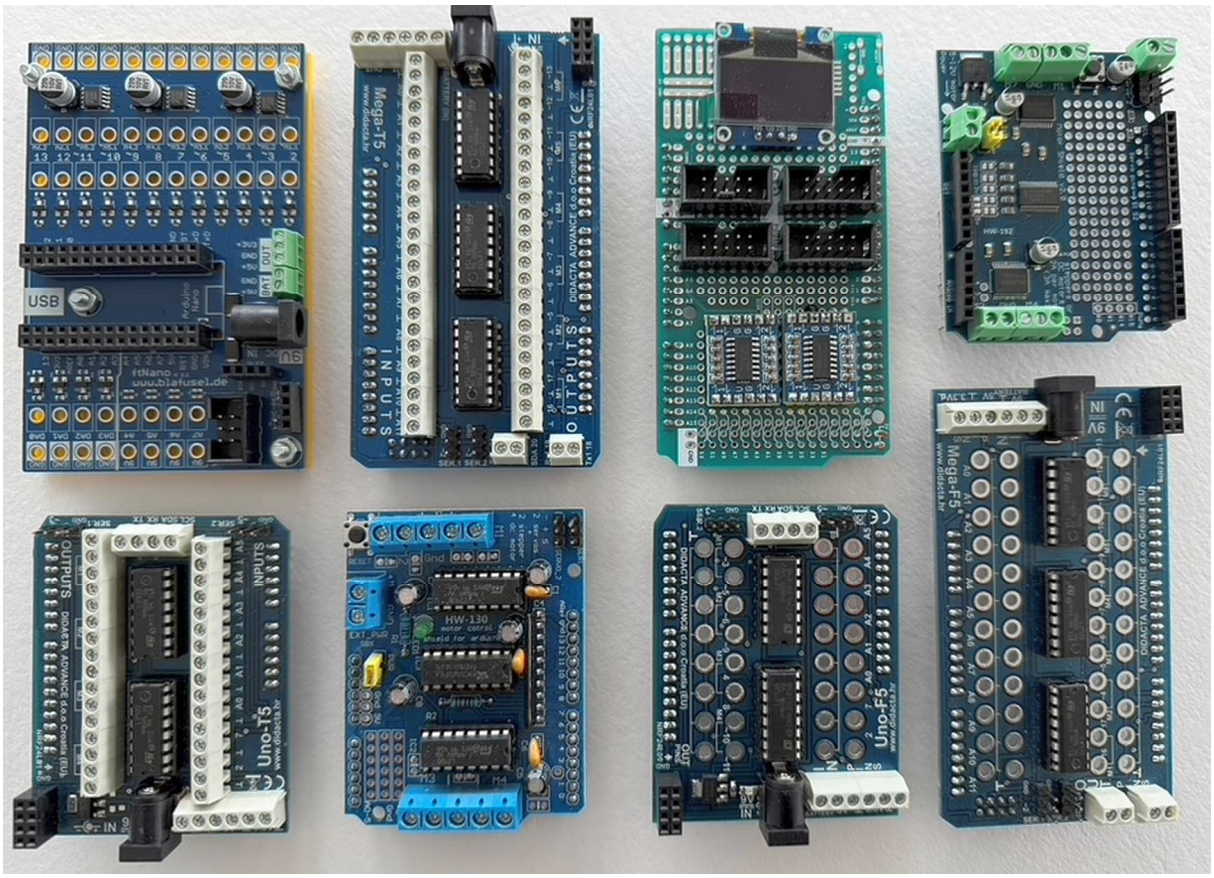


Abb. 2: Acht verschiedene Arduino-Shields (siehe Text)

Abb. 2 zeigt eine Reihe von Shields (im Uhrzeigersinn): ftNano, Didacta Mega Shield mit Schraubklemmen, selbstgebautes Shield mit LP10 Wannensteckern zum Anschluss von vier fischertechnik Intelligent Interfaces, Adafruit Shield Clone, Didacta Mega Shield mit Steckerlöchern, Didacta Uno Shield mit Steckerlöchern, Standard L296D Motortreiber-Shield (nur Motoren) und schließlich ein Didacta Uno Shield mit Schraubklemmen.

ftNano

Der Arduino Uno und Nano haben unterschiedliche Größen, aber ansonsten sehr ähnliche Spezifikationen. Der Nano hat sogar den kleinen Vorteil gegenüber dem Uno, dass er einen viermal größeren SRAM und eine doppelt so hohe Ausgangsstromkapazität hat. Florian Schäffer hat eine Platine [10] entwickelt, die den Arduino Nano mit folgenden Eigenschaften mit der 9V-fischertechnik-Welt verbindet:

- bis zu 16 digitale Ein- und Ausgänge, kompatibel mit 9V, 20mA (2 davon für Interrupts – „*Fast Counting*“)
- bis zu 6 digitale Stromausgänge 9V mit je bis zu 200mA, inkl. Schutz gegen induktive Lasten.
- 4 analoge Eingänge 0-10 V
- Buchsenleiste für OLED-Display
- Buchsenleiste für Bluetooth-Modul HC-05/HC-06 o. ä.
- I²C-Wannenstecker (6-Pin) wie auch beim TX/TXT/TXT 4 Controller
- Buchsenleisten für alle I/O-Pins des Arduino Nano
- Stromversorgung 9V (5-12V) über Netzteil, Batterie, Akku, etc.

Alle Funktionen des Arduino Nano können genutzt werden (I²C, SPI, UART, PWM, etc.). Das Board wird mit einer fischertechnik-kompatiblen Montageplatte und Schrauben geliefert; das einzige Zubehör, das hinzugefügt werden muss, ist ein Arduino Nano (Abb. 3, 4).

Es gibt ein umfangreiches 35-seitiges Handbuch [11] und ein Online-Archiv mit hervorragenden Softwarebeispielen [12]. Das Board kann über die Website bestellt werden und bietet meiner Meinung nach ein hervorragendes Preis-Leistungs-Verhältnis.

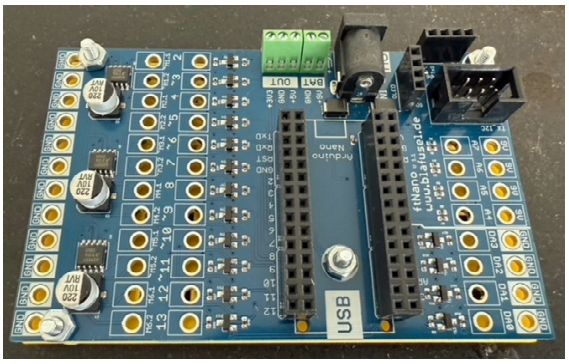


Abb. 3: ftNano

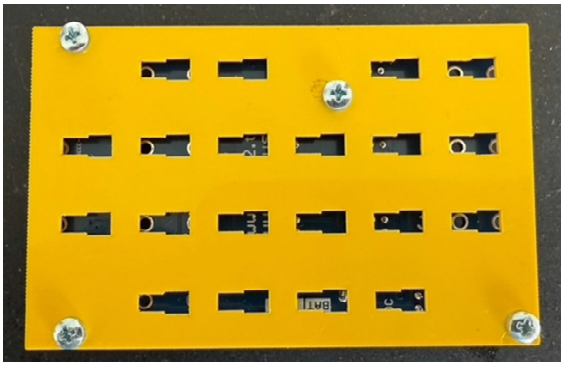


Abb. 4: fischertechnik-kompatible Montageplatte

Durch die offene Arduino-Architektur können alle Arten von Sensoren genutzt und dank der seriellen TX/RX-, SPI- oder I2C-Anschlüsse auch zusätzliche Hardware wie Displays, I/O-Module (wie Bluetooth) oder I2C-Sensoren angeschlossen werden. Das Schöne am ftNano-Board ist, dass keine speziellen Bibliotheken benötigt werden. Alle Arduino-Optionen des Nano können in Kombination mit fischertechnik-Komponenten verwendet werden.

Die Stromversorgung kann über die 7-12V DC-Buchse oder, bei eigenständigen Modellen, über eine 9V-Batterie erfolgen. Der Arduino Nano kann in die Buchsen in der

Mitte des Boards eingesetzt werden; die äußeren Buchsen ermöglichen den Zugang zu jedem der Nano-Pins mittels Dupont-Kabeln.

Bei der Verwendung von analogen oder digitalen Eingängen ist Vorsicht geboten: An den Shield-Anschlüssen dürfen diese bis zu 10V betragen, sind also kompatibel mit der fischertechnik 9V-Welt; beim Zugriff über die direkten Dupont-Stiftleisten sind jedoch 5V das Maximum.

Aufgrund des verwendeten Pegelwandlers können LEDs ohne seriellen Widerstand zwischen einem digitalen Ausgang und einem GND-Pin geschaltet werden. fischertechnik-LEDs mit seriellem Widerstand sollten wie ein Motor angeschlossen werden. Das Board verfügt außerdem über einen dedizierten I2C-Header für ein 0,91-Zoll-OLED-Display mit 128 x 32 Pixeln, wie z. B. das SSD1306, und einen separaten Header für das Bluetooth-Modul HC-05/HC-06 (Abb. 5).

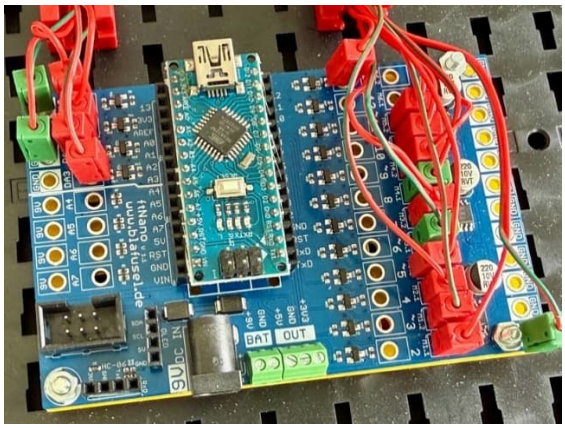


Abb. 5: ftNano mit Nano-Controller und Anschlüssen

Didacta

Die Didacta Shields für Arduino Uno und Mega gibt es in zwei Ausführungen mit je vier Varianten: Es gibt die F5-Platine mit Löchern, in die fischertechnik-Stecker direkt gesteckt werden können, und die T5-Platine mit Schraubklemmen. Sowohl die F5- als auch die T5-Platine können mit zwei verschiedenen Größen für den Stromanschluss erworben werden: entweder 5,2 x 2,1 mm (der ältere „große“ Stecker) oder 3,5 x 1,3 mm (der neuere, kleinere Stecker), wobei letzterer die Steckergröße ist, die fischertechnik für den TX, TXT und TXT 4.0 verwendet.

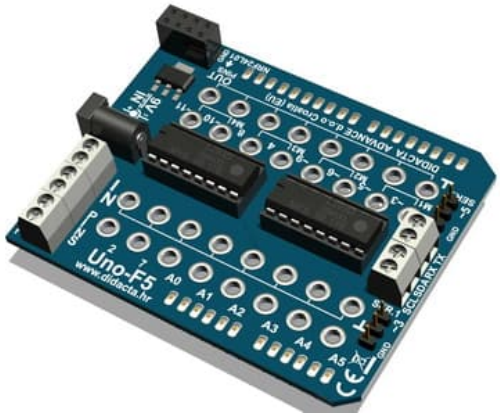


Abb. 6: Uno-F5 Shield, Steckerversion

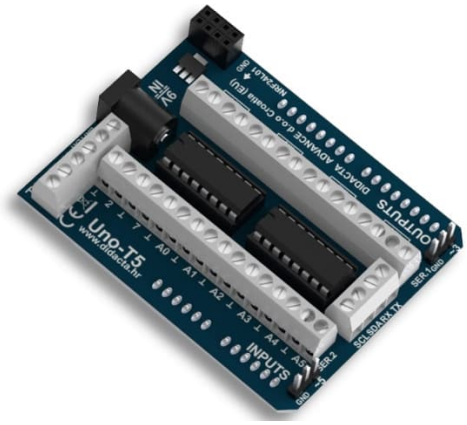


Abb. 7: Uno-T5 Shield mit Klemmen

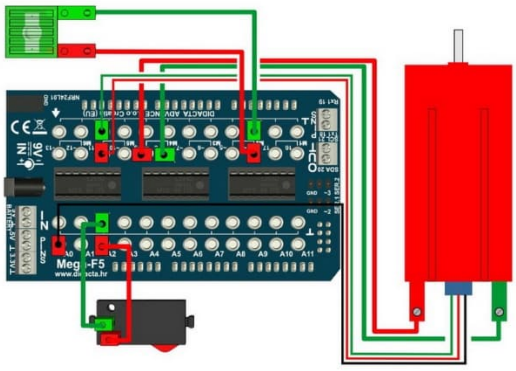


Abb. 8: Das Mega F5-Shield mit angeschlossenem Schalter, LED und Encoder-Motor

Die Shields für den Mega sind länger und haben mehr Anschlüsse; die Hauptunterschiede zwischen den beiden Shield-Typen sind in Tab. 1 zusammengefasst.

Modell	F5, T5	F5, T5
Controller	Uno	Mega
Ausgabe	8x (je 600mA)	12x (je 600mA)
Motoren	4 (3x PWM)	6 (5x PWM)
Eingabe	8, 2x digital, 6x analog	12x analog/ digital
Eingangs-spannung	9V	9V
Servo	1x 5V	2x 5V
Ausgangs-spannung	3,3V und 5V	3,3V und 5V

Tab. 1: Vergleich von Uno und Mega Shield

Wenn Sensoren verwendet werden, die eine bestimmte Spannung wie 3,3 oder 5V benötigen, sollte die Stromversorgung über die 3,3- oder 5-V-Anschlüsse erfolgen. fischertechnik-Sensoren, die mit 9V betrieben werden, werden an einen der digitalen Ausgänge angeschlossen und diese per Software eingeschaltet. In der Online-Programmierschule [13] gibt es viele Beispiele für Sensoren, wie z. B. LIDAR, Ultraschall, IR, Fototransistor, magnetisch, thermisch und Mikrofone.

Die Anzahl der Eingangspins kann durch den Einsatz eines I2C-I/O-Extenders wie den PCF8574 erweitert werden; dann können zahlreiche zusätzliche I2C-fähige Sensoren verwendet werden. Selbstverständlich können auch I2C-Aktuatoren genutzt werden, wie z. B. Displays usw. So können die Shields beispielsweise mit einem NRF24L01-Modul, das mit dem I2C-Bus verbunden ist, Wi-Fi-fähig gemacht werden. Das serielle und das UART-Protokoll können ebenfalls verwendet werden, beispielsweise um zwei Shields miteinander zu verbinden und Signale hin und her zu senden. Beispiele für die Nutzung dieser Funktionalität finden sich auch in der Programmierschule.

Didacta betont, dass es wichtig ist, nur das Didacta Shield mit 9V zu versorgen und nicht sowohl den Arduino als auch das Shield, da dies den 5-V-Spannungsregler am Shield beschädigen kann. Die Stromversorgung sollte mindestens 2,5A liefern; wird nicht genügend Strom zugeführt, funktionieren einige der Motoren nicht wie erwartet. Der Arduino und das Shield können in einem 3D-gedruckten Gehäuse untergebracht werden, das Schutz und Befestigungspunkte bietet [14].

In den Beispielen für Encoder-Motoren werden standardmäßige digitalRead-Operationen verwendet. Wenn man Hardware-Interrupts für die Encoder-Motoren verwenden möchte, werden diese mit den Uno- oder Mega-Interrupt-Pins verbunden. Dabei muss beachtet werden, dass der direkte Zugriff auf diese nur mit max. 5V erfolgen darf: Die dem Encoder zugeführte Spannung sollte daher 5V betragen und nicht die standardmäßigen 9V, die man für TX-, TXT- oder TXT 4.0-Steuerungen verwenden würde. Wie beim ftNano sind nur die spezifizierten Shield-Eingangsanschlüsse geschützt; der Arduino arbeitet weiterhin mit 5V.

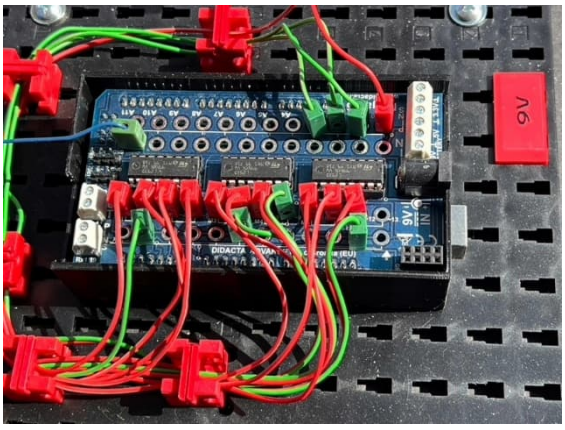


Abb. 9: Das Mega F5-Shield steuert ein Industriemodell

Adafruit V2.3

Das Adafruit Motorshield [15] kann vier Motoren oder zwei Schrittmotoren steuern. Dies geschieht mit mehreren erweiterten Funktionen; Die Treiber sind MOSFETs mit 1,2A pro Kanal (3A Spitze) mit geringeren Spannungsabfällen und Flyback-Dioden zum Schutz vor Spannungsspitzen. Auf dem Shield befindet sich zudem ein PWM-Chip, so dass die Drehzahlregelung über I2C statt wie bei den ftNano- und Didacta über Arduino-PWM-Pins gesteuert werden kann. Die Shield-Steuerung über I2C hat noch weitere Vorteile, da das Shield nicht nur mit dem Arduino Uno, sondern auch in Kombination mit dem Arduino Mega, Due und Leonardo verwendet werden kann. Wenn dem Shield Buchsenleisten hinzugefügt werden, sind auch alle Arduino-Pins zugänglich.

Anders als die ftNano- und Didacta-Shields benötigt das Adafruit-Shield eine Softwarebibliothek, da es I2C-gesteuert ist. Diese Bibliothek wird in der Arduino-Entwicklungsumgebung hinzugefügt und enthält zusätzlich zu den umfangreichen Online-Lernseiten Beispielprogramme [16].

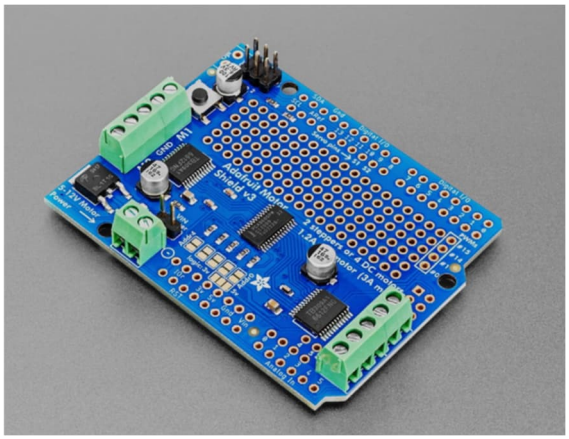


Abb. 10: Adafruit Motorshield V2.3

Die Motorsteuerung erfolgt über die beiden grünen Klemmen; zwei Motoren auf jeder Seite (Abb. 10).

Eine weitere Eigenschaft, die ich sehr mag, sind die freien Lötpunkte im Shield, über die zusätzliche Komponenten hinzugefügt werden können. In einem meiner Shields habe ich LP6- und LP20-Wannenbuchsen (Abb. 11) und in einem zweiten eine Stiftleiste hinzugefügt (Abb. 12). Die Anschlüsse zu den in der Software verwendeten Arduino-Pins habe ich an der Unterseite des Shields verlötet. Mit dem LP20-Stecker ist es nun möglich, ein Flachbandkabel mit dem 28-poligen Verteilerkasten hinzuzufügen – entweder die alte rote Version (75140) oder die modernere grüne Version (36391, 68617). Dadurch ist eine Abwärtskompatibilität zu vielen älteren Modellen gegeben und es lassen sich Ein- und Ausgabe auf kompakte Weise gut kombinieren.

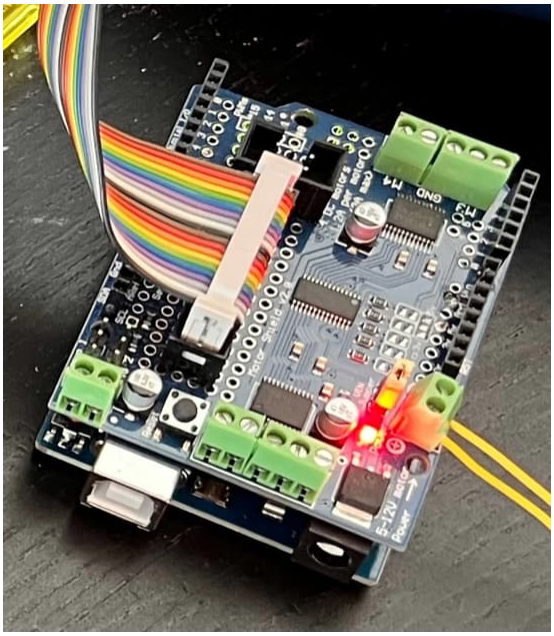


Abb. 11: das Adafruit-Shield mit zusätzlichem LP20-Anschluss, der es mit älteren fischertechnik-Computing-Modellen kompatibel macht

Die zweite Methode besteht darin, Standard-Dupont-Stiftleisten in einem geometrischen Muster hinzuzufügen und dann fischertechnik-Stecker darauf zu montieren. Außerdem führt diese Methode Ein- und Ausgänge transparent und fischertechnik-kompatibel zusammen.

In Abb. 12 ist zu sehen, wo sich die Buchsen am Shield befinden. Im Inneren der grünen Motorklemmen werden 90-Grad-Dupont-Stifte verwendet, um die

Anschlüsse für die Motoren 1-4 zu gewährleisten. Oben an der grünen 2er-Klemme werden 9V angeschlossen. Der einzelne rote Pin beträgt +5V für bestimmte analoge Anwendungen. Die grüne Reihe der Buchsen in der Mitte sind die 8 digitalen Eingänge E1-E8, die Reihe der vier roten Stecker sind vier analoge Eingänge und die vier grünen sind Masse-Pins, die für die analogen oder digitalen Eingänge verwendet werden können.

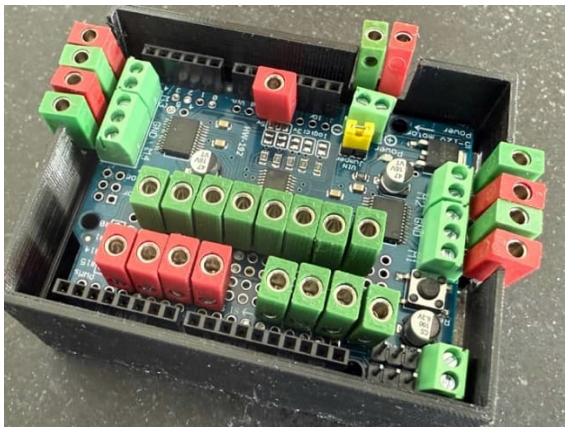


Abb. 12: Adafruit Shield im Gehäuse mit Dupont-Stiftleisten und fischertechnik-Steckerbuchsen

In beiden Varianten müssen die Verbindungen zwischen den hinzugefügten Pins bzw. LP-Steckverbindern und den in der Software verwendeten Arduino-Pins durch Anlöten von Drähten an der Unterseite der Shields hergestellt werden. Ein 3D-gedrucktes Gehäuse schützt den Arduino und das Shield vor Schmutz und unerwünschten Kontakten; die STL-Dateien sind auf Thingiverse zu finden [17].

Warnungen

Die Arbeit mit Shields birgt einige Risiken. Je offener ein Shield oben ist, desto höher ist das Risiko von versehentlichen Verbindungen oder von Wackelkontakten. Dies kann größtenteils durch ein 3D-gedrucktes Gehäuse und die Verwendung von Klemmanschlüssen vermieden werden. Es ist immer ratsam, die Stromversorgung vom Shield zu trennen, wenn Verbindungen hinzugefügt oder entfernt werden.

Jede Shield-Arduino-Kombination hat unterschiedliche Leistungsanforderungen. Bei der Verwendung eines Didacta-Shields sollte nur das Shield mit Strom versorgt werden; der Arduino erhält seinen Strom vom Shield. Im Falle des ftNano gibt es nur ein Netzteil für den Nano, das Shield und die Aktuatoren. Bei der Adafruit-Kombination funktionierte es meiner Erfahrung nach am besten, den Arduino und das Adafruit-Shield mit einem Splitter aus derselben Stromquelle mit Strom zu versorgen. Versorgt man nur das Shield mit Strom, muss der Vin-Jumper auf dem Shield aufgesetzt werden.

Für die I2C-Buchsen gibt es unterschiedliche Konventionen. So verwendet der TXT 4.0 andere Pins für VCC und GND als TX und TXT. Daher ist es wichtig, die Anschlüsse zu überprüfen, um sicherzustellen, dass die I2C-Peripherie korrekt angeschlossen ist.

Software

In der Arduino IDE ist es möglich, für jedes dieser Shields Programme zu schreiben. Angesichts der Pin-Konventionen, die auf dem Shield verwendet werden, funktionieren diese Programme jedoch nur für das jeweilige Shield. Dies ist sehr ineffizient, wenn man ein bestimmtes Modell mit einem anderen Shield oder Mikrocontroller verwenden möchte. Ich habe daher die Software so standardisiert, dass ich diese Modelle mit einem absoluten Minimum an Änderungen in der Software für einen Arduino Uno, Mega oder Nano und entweder mit der parallelen oder seriellen Schnittstelle von fischertechnik oder dem Adafruit, Didacta Shield oder dem ftNano nutzen kann. Diese C++-Softwarebibliothek werde ich im 4. Quartal 2025 auf Github veröffentlichen, sobald ich das umfangreiche Testprogramm mit vielen verschiedenen Konfigurationen und Modellen abgeschlossen habe.

Referenzen

[1] Jens Lemkamp: *Parallel-Interface durch Arduino gesteuert (1-3)*. ft:pedia 1/2014, S. 24–30; ft:pedia 2/2014, S. 36–39; ft:pedia 3/2014, S. 61–65.

[2] René Trapp: *V. I. P. – Ein I2C-nach-Computing-Interface-Umsetzer (Teil 1-5)*, ft:pedia 2/2017, S. 63–73; ft:pedia 3/2017, S. 57–68; ft:pedia 4/2017, S. 36–49; ft:pedia 1/2023, S. 111–122; ft:pedia 2/2023, S. 70–76.

[3] Jeroen Regtien: *Controlling parallel (universal) and serial (intelligent) interfaces with an Arduino*. ft:pedia 4/2023, S. 69–80.

[4] David Holtz: *Alternative Controller (1): Der Arduino*. ft:pedia 2/2016, S. 56–59.

[5] Till Harbaum: *ftDuino – Open-Source trifft Konstruktions-Baukasten*. ft:pedia 1/2018, S. 85–91.

[6] Dirk Fox, Thomas Püttmann: *fischertechnik-Roboter mit Arduino*. dpunkt-Verlag, 2020.

[7] Dirk Fox: *fischertechnik-Roboter mit Arduino (Teil 1): Smartphone-Steuerung über BLE*. ft:pedia 3/2020, S. 93–100.

[8] Arnoud van Delden: *fischertechnik-Roboter mit Arduino (Teil 2): 2,4-GHz-Fernbedienung mit dem PS2-Gamepad*. ft:pedia 4/2020, S. 58–63.

[9] www.didacta.hr

[10] OBD2-Shop: *ftNano (230456)* von Florian Schäffer.

[11] Florin Schäffer: *ftNano Benutzerhandbuch*. Version 1.1, September 2024.

[12] Florian Schäffer: *Beispielprogramme für den ftNano*.

[13] Didacta: *Programmierschule*.

[14] thingiverse: *fischertechnik-Gehäuse für Arduino UNO/MEGA und Didacta Shield*.

[15] Adafruit: *Motor Shield 2.3*

[16] Adafruit: *Lernseiten zum Motorshield*.

[17] thingiverse: *fischertechnik Adafruit Motorshield-Gehäuse*.

Computing

fischertechnik-Modelle mit der Oxocard steuern

Axel Chobe

Die vom Magazin MAKE: promotete Oxocard ist ein Mikrocontroller mit Suchtfaktor – durchdacht und dank der Programmiersprache NanoPy ein hervorragender Einstieg in die Mikrocontrollerprogrammierung. Und man kann fischertechnik-Modelle damit steuern…

Einführung

Die Oxocard ist ein 5 × 4 cm² kleiner Minicomputer des Berner Herstellers Oxon mit einem 240 × 240 Zeichen großen Farbdisplay, Joystick, USB-C-Anschluss, Netzwerkfunktionen und einem Steckplatz für Erweiterungsports.

Kern der Oxocard ist ein ESP32 mit 2 Mbyte PSRAM, 8 Mbyte Flash, WLAN und Bluetooth. Die Oxocard wird mit der Skriptsprache „NanoPy“ über eine browserbasierte Seite via USB, WLAN oder Bluetooth programmiert. Damit lässt sich über die Oxocard mit wenig Aufwand eine leistungsstarke Steuerung z. B. für die Maker Kits von fischertechnik entwickeln.

Die Maker Kits werden seit 2024 angeboten, um fischertechnik-Grundmodelle (ein Fahrzeug mit Servo-Lenkung, ein Mecanum-Wheel-Chassis und ein Roboter-Basismodell) mit beliebigen Mikrocontrollern anzusteuern.

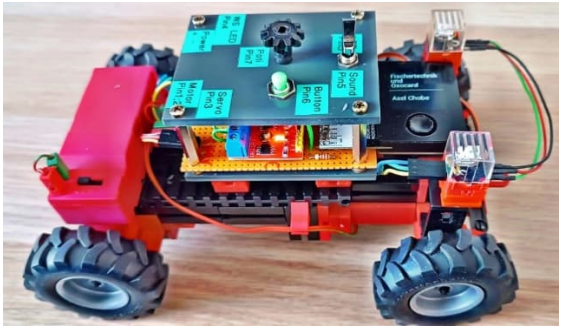


Abb. 1: Maker Kit Car mit Anbauten

Das einfachste, hier benutzte Modell ist das Maker Kit Car (571900). Es besitzt einen Antriebsmotor und für die Lenkung einen Servo. Das Modell habe ich um zwei WS-LED als Scheinwerfer und Blinker sowie eine Hupe ergänzt (Abb. 1).

Mikrocontroller

Die Oxocard besitzt einen universellen Cartridge-Steckplatz, über den fertige oder selbst entwickelte Platinen durch einfaches Einstecken sofort zum Leben erweckt werden können. Um Sensoren auszulesen oder LEDs zu steuern wird ein Erweiterungsboard benötigt, welches an die Oxocard gesteckt wird. Darauf können Bauelemente oder Leitungen aufgesteckt werden. Um eine Schaltung dauerhaft zu nutzen kann eines der vielen Erweiterungsboards erworben werden. Die Pinbelegung der Oxocard umfasst fünf IO-Pins (PWM-fähig), zwei ADC-Eingänge, I2C und SPI.

Abb. 2: Oxocard mit Breadboard-Cartridge

Erweiterungen

Sobald eine Cartridge eingesetzt ist, erkennt die Oxocard die Karte, lädt die Treiber und startet automatisch ein Demoprogramm. Jede Cartridge hat auch einen kleinen Flash-Speicher, auf dem Treiber und Programme gespeichert werden können. Es muss nichts installiert oder konfiguriert werden.

Folgende Karten sind derzeit erhältlich:

- Breadboard-Cartridge: um eigene Schaltungen zu stecken
- Expansion Cartridge: z. B. für einen Übergang auf ein anderes Breadboard oder eine Leiterplatte

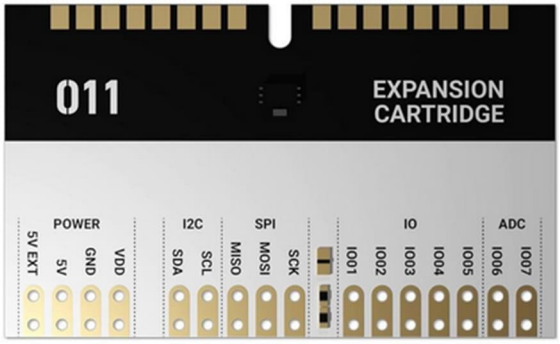


Abb. 3: Expansion Cartridge

- Veroboard Cartridges: um permanent eine Schaltung aufzulöten
- Pixelmatrix Cartridge: stellt eine 5 × 5 WS-LED-Matrix zur Verfügung
- Synthesizer Cartridge: ein 38-stimmiger polyphoner Synthesizer
- TOF-Cartridge (Time of Flight): Distanzmessung mittels Laser
- AIR-Cartridge beinhaltet Sensoren für Gas, CO2 sowie Temperatur und Luftfeuchtigkeit

Software

Die Programmierung erfolgt über eine webbasierte Oberfläche: Man muss keinen Compiler auf dem Rechner installieren, wie sonst bei Mikroprozessorboards üblich. Ein Besuch der Website führt direkt zum Nano-Py-Editor (Abb. 4). Hier kann man fertige Beispiele laden oder auch eigene Programme schreiben (Hinweis: nicht mit Firefox).

Im linken Fenster sind die Informationen über Geräte, Beispielprogramme und die eigenen Programme zu finden. Die Programme können hier auf der Festplatte, der Oxocard oder auf der Cartridge gespeichert werden. Im mittleren Fenster wird der Code geschrieben. Hier kann auch der Debugmodus gestartet werden. Im rechten Fenster werden die Variablenwerte angezeigt (nur im Debugmodus), Konstanten über Regler verändert, im Terminalfenster definierte Ausgaben vom Programm gelesen, sowie eine umfangreiche Hilfe aufgerufen und (wenn vorhanden) ein Tutorial zum Beispielprogramm durchgearbeitet.

Bezug

Die Oxocard kann bei vielen Anbietern erworben werden. Empfehlen kann ich den Kauf über die Zeitschrift Make: Hier bekommt man die Oxocard Connect (Oxocard mit Breadboard-Cartridge) in einem Innovators Kit („Make Edition“) mit 96 Elektronikbauteilen. Enthalten ist ein 60-seitiges Make:-Sonderheft [1], in dem alle grundlegenden Dinge beschrieben werden, um ohne Vorkenntnisse sofort loszulegen.

Motivation

Als Anregung diente mir ein Make:-Artikel von Dirk Fox über das Zusammenspiel von Oxocard und fischertechnik [2]. Darin wird beschrieben, wie die Oxocard um Motortreiber ergänzt werden kann, mit denen die 9V-Motoren von fischertechnik gesteuert werden können.

Nach dem Erwerb der Oxocard wollte ich damit ein Fahrzeug steuern. Gleichzeitig sollte eine Komponente entstehen, mit der es einfach möglich ist, ohne Aufwand Experimente durchzuführen, bei denen auch Ein- und Ausgänge eine Rolle spielen (Abb. 5).

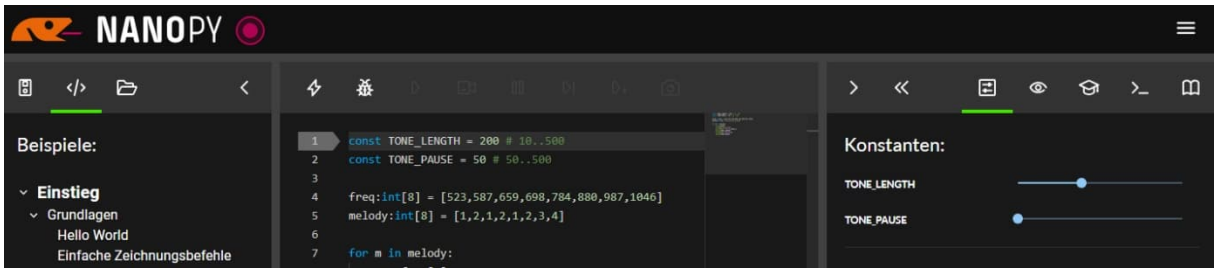


Abb. 4: NanoPy-Editor

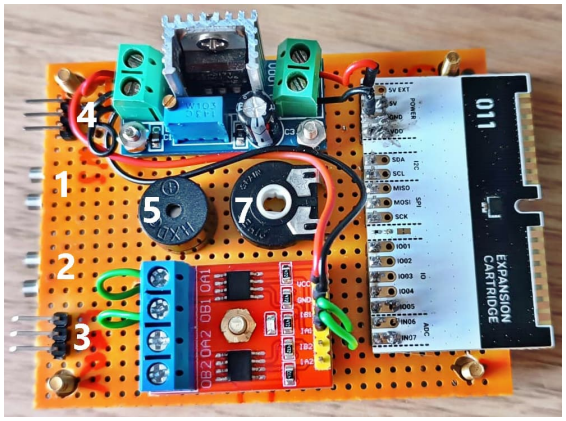


Abb. 5: Platine mit Motortreibern und Spannungswandler

Im Ergebnis können derzeit folgende davon benutzt werden:

- Pin 1 und 2: Motorsteuerung über Treiberplatine
- Pin 3: Servosteuerung
- Pin 4: Ansteuerung von WS-LEDs
- Pin 5: Soundmodul oder LED (über Umschalter wählbar)
- Pin 6: Buttonabfrage
- Pin 7: Potiabfrage

Hardware

Als Grundlage habe ich eine Lochrasterleiterplatte passend zugeschnitten. Das Maß der Breite richtete sich nach der Expansion Cartridge plus den Leiterplatten-Abstandshalter. Die Länge ergab sich aus der Addition der Größe der Expansion Cartridge plus den Schaltungen für Motortreiber und Spannungsregler, sowie den Anschlüssen für Servo, WS-LED, fischertechnik-Motor und Spannungseingang. Der Spannungsregler setz die fischertechnik-Spannung von 9 Volt auf die erforderliche Betriebspannung der Oxocard von 5 Volt um, für die es einen separaten Eingang (5V EXT) gibt; damit entfällt die Stromversorgung über ein USB-C-Anschluss. Die Treiberplatine ermöglicht das Ansteuern von fischertechnik-Motoren, da die Oxocard nur mit 3,3 Volt arbeitet. In der Mitte konnten außerdem noch ein Poti (10kΩ) und ein Buzzer als Hupe aufgelötet werden. Auf der Rückseite der Leiterplatte wurde anschließend alles passend mit Drahtbrücken verlötet.

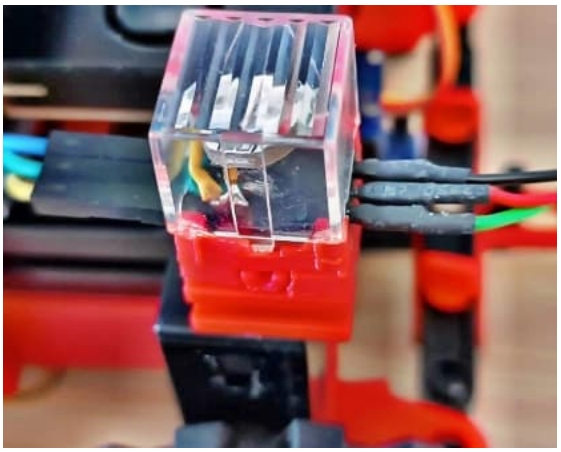


Abb. 6: WS2812b mini Board LEDs

Die im Modell verbauten Gehäuse für die WS-LEDs sind mit dem 3D-Drucker erzeugt. Die Vorlage kann bei thingiverse heruntergeladen werden. Links sind die drei Eingänge (VCC, GND, DataIn) und rechts die drei Ausgänge (VCC, GND, DataOut) zu sehen. Die WS2812b mini Board LEDs können für wenig Geld im Netz gekauft werden.

Auf der Frontplatte aus PVC befindet sich der Umschalter für die LED bzw. dem Buzzer: Bedingt durch die geringe Anzahl

von Ports können nicht beide zugleich angesteuert werden. Des Weiteren sind der Button und die LED zu erkennen. Das Poti auf der Leiterplatte wurde mit einer Rastachse und einem fischertechnik-Zahnrad nach oben geführt (Abb. 7).

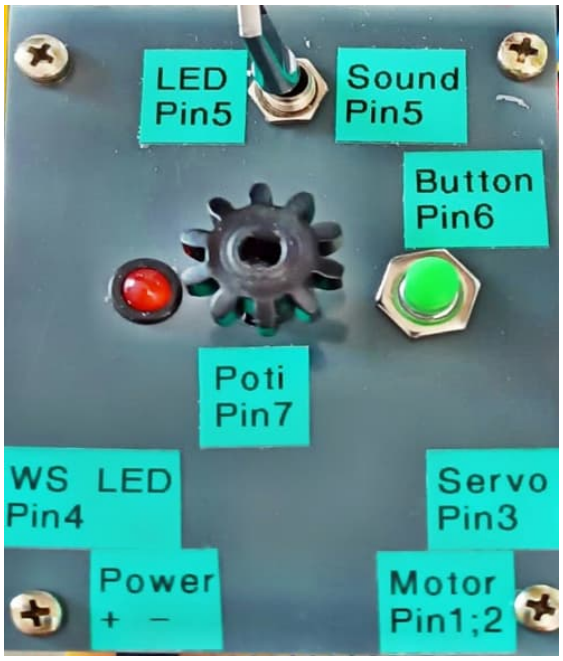


Abb. 7: Frontplatte

Auch die Grundplatte besteht aus PVC. Darauf wurde mit kleinem Abstand die Leiterplatte aufgeschraubt. Um die Kompatibilität zu fischertechnik herzustellen habe ich vier Bausteine 0,5 (37237) von oben angeschraubt, wobei die Zapfen vorher entfernt wurden (Abb. 8).

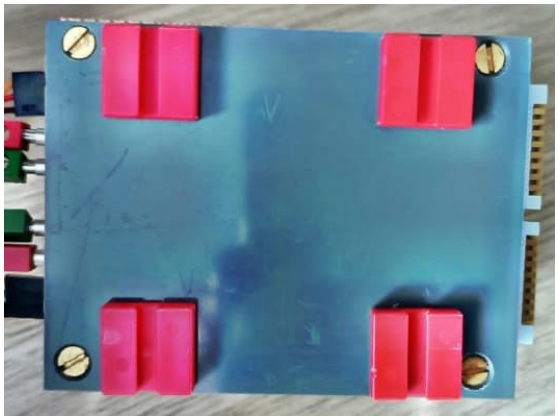


Abb. 8: Grundplatte

Programmbeispiele

Wie erwähnt handelt es sich bei NanoPy um eine Art Python. Durch die gute Einführung des Make:-Heftes, den vielen vorhandenen Beispielen und der ausführlichen Hilfe im Programm ist die Sprache sehr leicht zu erlernen. Im Folgenden stelle ich einige kurze funktionierende Programmelemente für die Ein- und Ausgänge vor:

- Ausgabe von Text auf dem Display

```
textFont(FONT_ROBOTO_32)
drawText(10,10,"FT und Qxocard")
update()
```

- Motorsteuerung

```
initGPIO(C_PIN_01, OUTPUT)
initGPIO(C_PIN_02, OUTPUT)
writePWM(C_PIN_01, 0)
writePWM(C_PIN_02, 4095)
```

- Servolenkung

```
initGPIO(C_PIN_03, OUTPUT)
x = 45 # für x 90 bis -90
dutyCycle = map(x,90,-90,102,512)
writePWM(C_PIN_03, dutyCycle)
```

- WS-LEDs

```
initDigitalLeds(C_PIN_04,2,
C_LED_TYPE_WS2812)
setDigitalLed(0, 255,255,255)
setDigitalLed(1, 255,255,255)
applyDigitalLeds()
```

- LED (an/aus)

```
initGPIO(C_PIN_05, OUTPUT)
writeGPIO(C_PIN_05, 1) # 0 für aus
```

- LED dimmbar

```
initGPIO(C_PIN_05, OUTPUT)
writePWM(C_PIN_05, 2000) # 0-4095
```

- Sound

```
initGPIO(C_PIN_05, OUTPUT)
writePWM(C_PIN_05,4096/2)
delay(200)
writePWM(C_PIN_05,0)
```

- Poti

```
while true:
    initGPIO(C_PIN_07, INPUT)
    print(readADC(C_PIN_07, 100))
    delay(1000)
```

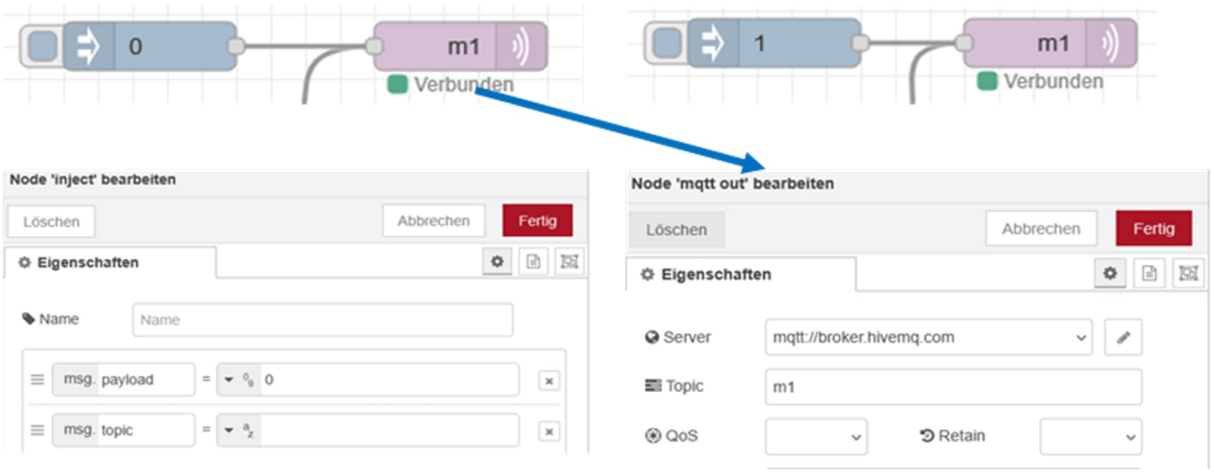


Abb. 9: Konfiguration in Node-RED

Vorschau

Im Weiteren soll auch eine Art Fernbedienung über Node-RED und MQTT realisiert werden. Eine einfache Möglichkeit zum ein- und ausschalten einer LED ist die folgende:

```
initGPIO(C_PIN_05, OUTPUT)
uri = "mqtt://broker.hivemq.com"
username = ""
password = ""
if connectMQTT(uri, username,
password):
    subscribeMQTT("m1")
def onDraw():
    if hasMQTTMessage():
        a = getMQTTData()
writeGPIO(C_PIN_05,strToInt(a))
```

Auf Seiten von Node-RED dann entsprechend (siehe Abb. 9); komfortabler wäre hier das Dashboard.

Ich kann mir auch eine Porterweiterung unter I2C mit dem PCF8574 vorstellen, um größere Projekte zu realisieren.

Fazit

Das Programmieren mit NanoPy ist sehr einfach und macht großen Spaß, gerade weil man sehr schnell zum Erfolg kommt. Im Gegensatz zum Arduino ist keine Softwareinstallation oder Treiberanpassung notwendig.

Ein Beispielvideo des Modells steht auf meiner Webseite zur Verfügung.

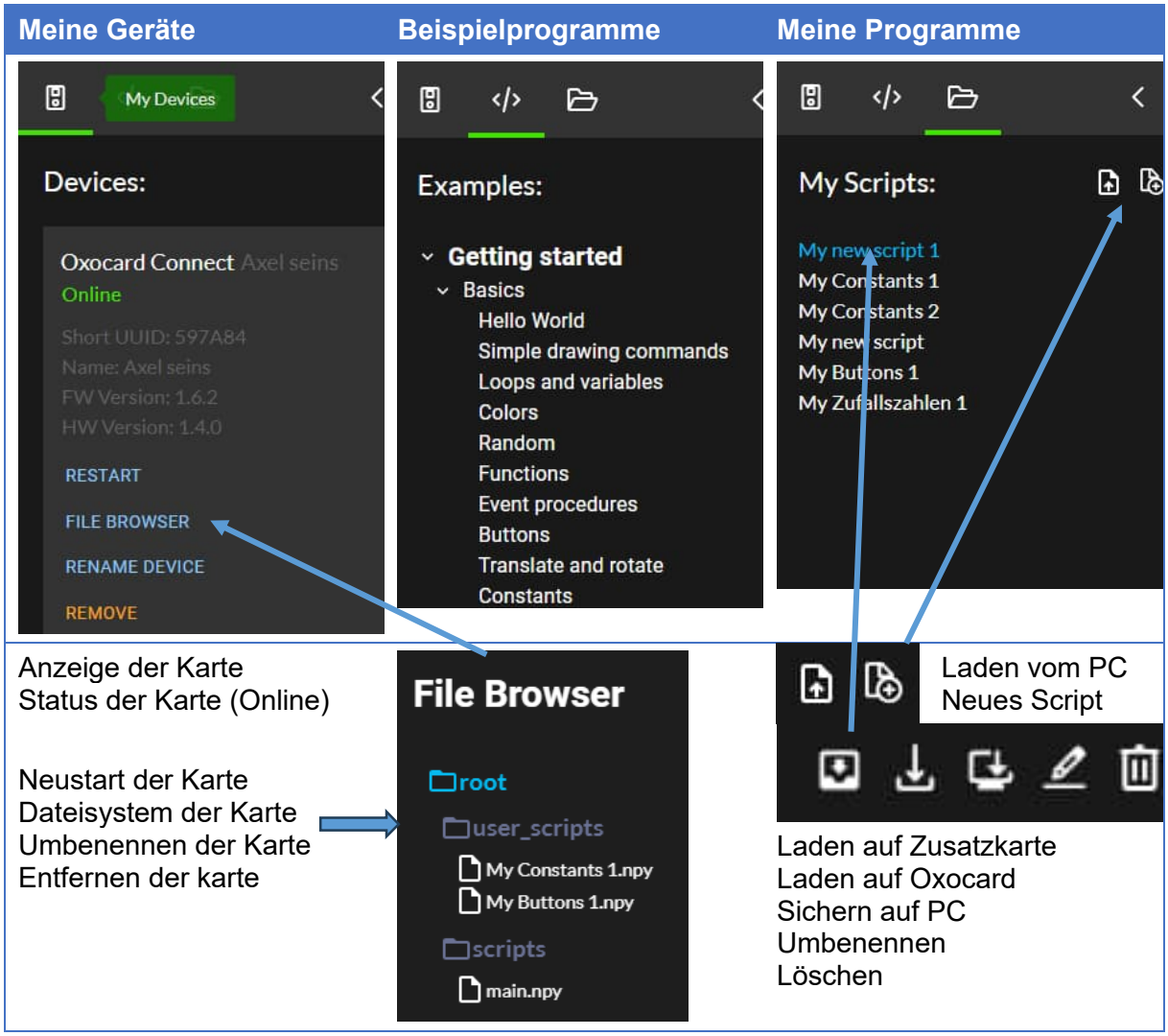
Quellen

[1] Make/Oxon: *Spezial Oxocard*, heise-Verlag, 2025.

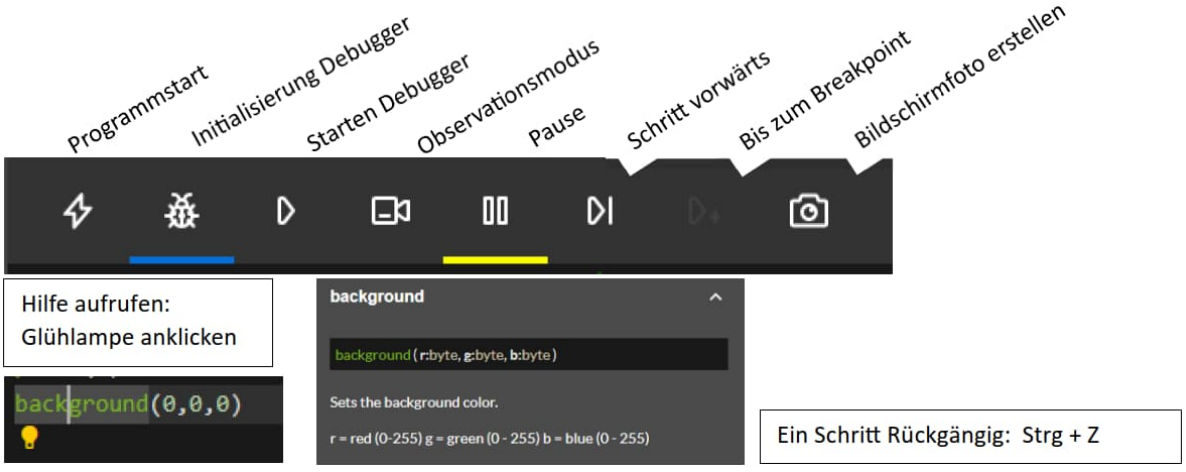
[2] Dirk Fox: *Fischertechnik mit der Oxocard steuern*. Make: 2/2025, heise-Verlag, S. 74-80.

NanoPy IDE

Links

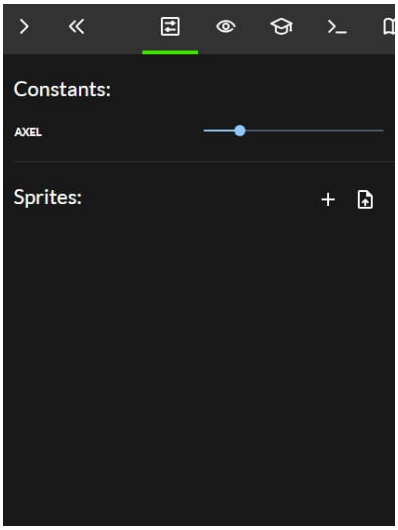


Mitte



Rechts

Konstanten

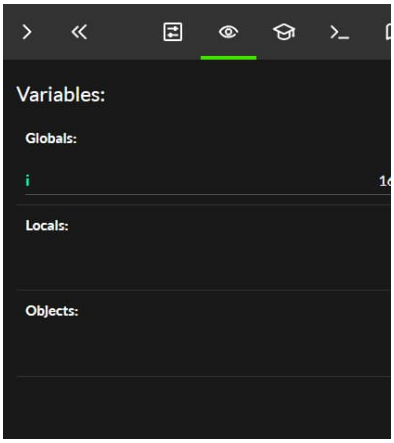


Wird eine Konstante mit einem Wertebereich nach dem Kommentarzeichen festgelegt, erscheint diese im Fenster und kann mit dem Slider verändert werden.

```
const AXEL = 2 # 0 .. 10

drawText(0,0,AXEL)
update()
```

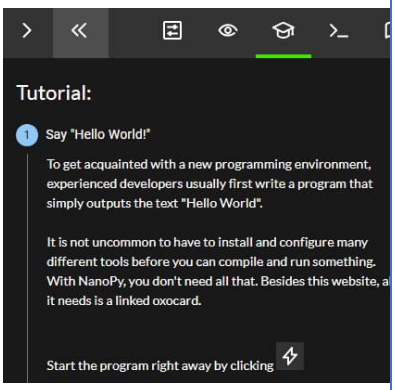
Variablen



Zeigt die Variablen während des Debuggings an

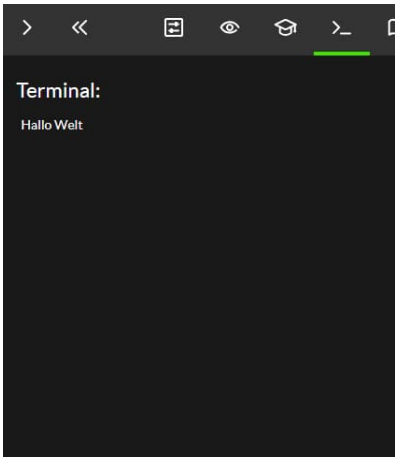
```
1   i = 12
2   while true:
3       print(i)
4       i = i +1
5
```

Tutorial



Zusätzliche Erklärungen eines Programmes. Wird vom Anbieter des Programms erstellt.

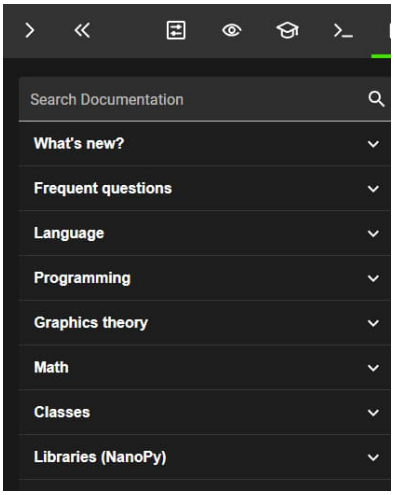
Terminal



Ausgabe von Informationen, die im Programm mit print definiert wurden

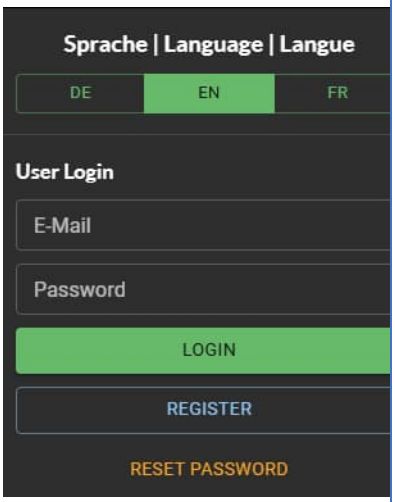
```
print("Hallo Welt")
```

Dokumentation



Hier ist die komplette Dokumentation abgelegt

Menü



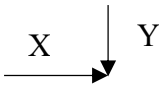
Auswahl der Sprache
Benutzer-Login
Passwort zurücksetzen

NanoPy-Sprachelemente

Variable	`A = true`
	`B = 10`
	`C = 3.14`
	`D = 45.678`
	`E = vector (x=10, y=20)`
		Datentyp bool false/true 0/1
		Datentyp byte 0 bis 255
		Datentyp int -32768 bis 32767
		Datentyp long -2147483648 bis 2147483648
		Vector; x- und y-Parameter in float

Konstanten	`const = FARBE # HUE`
	`const SHOW_TEXT = true # true, false`
	`if SHOW_TEXT:`
	`  drawText(10,10,"HALLO")`
	`update()`
		

Arrays und Listen	`liste = [1, 2, 3, 4]`
	`print(list[2])`
	`print sizeof(list)`
		Erzeugung der Liste
		Abruf des Elementes 2 → 3
		Ausgabe der Listenlänge

Textausgabe auf Bildschirm
 
	`textFont(FONT_ROBOTO_24) oder`
	`textFont(FONT_ROBOTO_BOLD_24)`
	`drawText(10, 10, "Testtext")`
	`update()`
		FONT_ROBOTO_16#24,32, 48,80
		FONT_ROBOTO_BOLD_16 bis 80
		Textposition und Text
		mit update() wird es dargestellt

Texteingabe	`str = textInput("Test", "123")`
	`drawText(10,10,str)`
	`update()`
		öffnet ein Text-Eingabefeld und gibt den Text zurück

Text auf Konsole	`print(„Hallo Welt!“) #Text`
	`print(123) #Zahl`
	`a = 1.23`
	`print(a) #Variable`
		sendet den definierten String-Parameter ans Terminal

Textausrichtung	`textAlignment(alignment:byte)`
	`TEXT_ALIGN_LEFT #links`
	`TEXT_ALIGNE_CENTER #zentriert`
	`TEXT_ALIGN_RIGHT #rechts`
		es kann die Textausrichtung der drawText-Funktion angepasst werden

Ausgabe von Zeichen	`drawChar(10,10,'?')` `update()`	zeichnet ein Zeichen, wichtig → einfache Hochkomma
Ausgabe IP-Adresse	`ipAddress() #=„192.168.178.128“`	gibt die aktuelle IP-Adresse als String zurück
Ausgabe Hardware	`getHardwareType()`	gibt den Typ der Oxocard zurück; Galaxy=0, Artwork=1, Science=2, Sience_Plus=3, Connect=4
Ausgabe Sprache	`getLanguage()`	Gibt die aktuelle System-sprache zurück: C_LANGUAGE_EN; DE; FR

Hintergrundfarbe	`background(r,g,b)`	R,G,B zwischen 0 und 255

Buttonabfrage	`def onDraw()`
	`  b = getButtons()`
	`  if b.up:`
	`  delay(500)`
	`  print("Button hoch"`
		Übergabe d. Abfrage auf „b“
		Wenn b nicht gedrückt:
		Pause zum Entprellen
		up, down, left, right, middle

Pause	`delay(1000)`	Zeitangabe im ms

Einfache Funktion	`def sayHello()`
	`  clear()`
	`  drawText(10,10,"Hallo")`
	`  update()`
	`sayHello()`
		Funktionsdefinition
		Bildschirm löschen, Inhalt in Funktion wird eingerückt
		Textausgabe
		Funktionsaufruf

Ereignisfunktion Funktionen werden automatisch aufgerufen	`def onDraw()`	Aktualisierung des Bildschirms
	`def onClick()` `  print(„Joystick gedrückt")`	Aufruf, wenn eine der Joystick-tasten gedrückt wird

	`print("Text in 3 Sek")`
	`setInterval(3000)`
	`def onTimer():`
	`  print("Text")`
		Ankündigung
		Ruft das TimeEvent auf (in ms), Start TimeEvent
		Funktion im TimeEvent

Endlosschleife	`while true:`
	`  y = random(0,240)`
	`  drawCircle(5,y,50)`
	`  update()`
	`  delay(100)`
		solange wahr → führe aus
		Zufallszahl; Inhalt der Schleife eingerückt
		Kreis m. zufälliger y-Position Zeigen, Pause 0,1 Sek

Zählschleife	`for i in [0..10]:`
	`  drawLine(0,i*20,200,i*20)`
	`  drawLine(i*20,0,i*20,200)`
	`update()`
		i = Zähler und […] Zählerbereich
		Schleife wird 11-mal durchlaufen und zeichnet ein Gitternetz
		Start

Bedingung 1	`test = 1`
	`if test <= 2:`
	`  drawText(10,10,"kleiner 2")`
	`else:`
	`  drawText(10,10,"größer 2")`
	`update()`
		Eine Weiche, mit der der Programmablauf umgelenkt wird
		nach if bzw. else (für Alter-native) wird eingerückt
		elif für Zwischenbedin-gungen

Bedingung 2	`while iI < 5:`
	`  print i`
	`  I = I + !`
		Wirkt wie eine Zählschleife

Runden	`round(24.3) #=24.0`	rundet den Wert n auf den ganzzahligen Wert

Zufallszahl	`x = random(0,240)`	zufälligen Wert zw. min und max
Wertebereich umschreiben	`v = map(50,0,10,0,255) #=127.5`	konvertiert den Wert eines Wertebereichs in einen anderen (Wert 50 von 0-10 in Wert von 0-255)
Bildschirm löschen	`Clear()`	löscht alle Pixel
Strich/Textfarbe festlegen	`stroke(r:byte, g:byte, b:byte)`	setzt die Strich- oder Textfarbe
Strichdicke festlegen	`strokeWeight(w:byte)`	setzt die Strichdicke von 1 bis 10
Füllfarbe festlegen	`fill(0,255,0) #Gelb`	füllt das gezeichnete Objekt
Füllfarbe löschen	`nofill()`	keine Füllfarbe, durchsichtig
Nullpunkt verschieben	`translate(120,120)`	Verschiebt Nullpunkt relativ zur vorherigen Position
Linie zeichnen	`drawLine(0,0,240,240)`	xy=Anfang, x_1y_1=Ende Fenstergröße von 0 bis 240
Rechteck zeichnen 	`drawRectangle(5,5,230,230)`	xy=oben links, x_1y_1=unten rechts
Rechteck mit abgerundeten Ecken zeichnen 	`drawRectangleRounded(5,5,100,100,9)`	zeichnet ein abgerundetes Rechteck, fünfter Wert= Radius
Dreieck zeichnen 	`drawTriangle(5,5,100,5,50,100)`	zeichnet ein Dreieck x_0/y_0, x_1/y_1, x_2/y_2
Viereck zeichnen 	`drawQuadrangle(0,0,150,0,100,200,0,150)`	zeichnet ein Viereck x_0/y_0, x_1/y_1, x_2/y_2, x_3/y_3
Kreis 	`drawCircle(120,120,50)`	Kreis x_0/y_0 mit Radius r
Ellipse zeichnen 	`drawEllipse(100,100,70,90)`	zeichnet eine Ellipse um die Pos. x_0/y_0 mit den Radien r_0 und r_1

Stilfarbe speichern	`fill(255,0,0) #Füllfarbe rot`
	`push() #Zustand speichern`
	`fill(0,255,0) #Füllfarbe grün`
	`drawRectangle(0,0,100,100) #gr. Rechteck`
	`pop() #alte Füllfarbe (fill) laden`
	`drawCircle(50,50,30) #roter Kreis`
	`update() #Ausgabe`
		speichert die vorhergehenden aktuellen Stil-Konfigurationen mit push() und stellt sie mit pop() wieder zur Verfügung

Skaliert Zeichnungselement	`scale(2)`	skaliert alle nachfolgenden Zeichnungsbefehle um den Wert

Objekt der Klasse	`dt:dateTime`	Objekt der Klasse (immer zuerst)
Aktuelle Zeit setzen	`dt.now()`	setzt das dateTime-Objekt gleich der aktuellen Uhrzeit
Anzeige akt. Jahr, Monat, Tag	`getYear() / getMonth()/ getDay()`	gibt aus: Jahr/Monat/Tag
Anzeige Stunde, Minute, Sek.	`getHour() / getMinute() /getSecond()`	gibt aus: Stunde/Minute/ Sekunde
Anzeige des Wochentages	`getWeekDay()`	0=So, 1=Mo, 2=Di, 3=Mi, 4=Do, 5=Fr, 6=Sa
Datum ändern	`setTime(23,59,55) #h,m,s`	überschreibt das Datum
Zeit ändern	`setDate(31,12,1990) #d,m,y`	überschreibt die Zeitangabe

Timer	`setTimer(3000)`
	`def onTimer():`
	`  print("Hallo")`
		ruft das onTimer()-Event nach xx ms auf

Zeitausgabe	`t1 = getSystemTime()`
	`delay(5)`
	`t2 = getSystemTime()`
	`print(t2-t1) #=4704 (kann variieren)`
		gibt die Zeit in Mikrosekunden seit dem Systemstart zurück (z. B. um Zeit zw. Funktion zu messen)

float zu int wandeln	`print toInt(round(1.56)) #=2`	wandelt eine float-Zahl in einen int-Typ um
int zu float wandeln	`toFloat(120) #=120.0`	wandelt eine int-Zahl in eine float-Zahl um
string zu float wandeln	`print strToFloat("120") #=120.0`	wandelt einen String in float um

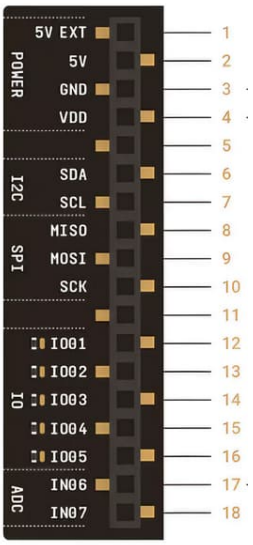
Programm verlassen	`def onDraw():`
	`  delay(10)`
	`  if getButton():`
	`  if returnToMenu():`
	`return`	

Grundoperationen	`+, -, *, /, % (modulo oder Rest)`	für Berechnungen
Vergleichsoperatoren	`==, <, >, =<, =>, <=, !=` `if a <= b:`	Vergleich von Werten o. Zuständen
Verknüpfungen	`and, not, or` `if a = 3 and b = 2:`	Verknüpfungen in Kontrollstrukturen

Binäre Funktionen
bitweise Operation
Bit verschieben
	`&(and), | (or), ^(xor)`
	`print (2 & 3) #=2 oder`
	`print (2 | 3) #=3`
	`<<, >> # Bit links bzw. rechts verschieben`
	`print(4<<2) #=16`
		and 010 or 010
		011 011
		010 011
		Zahlen werden binär betrachtet
		00100 → 10000 #2mal links

MQTT	*(folgt)*	

NanoPy-Befehlsreferenz Oxocard

initGPIO
 
 	
	`initGPIO(pinNr.byte, mode:byte)`
	`C_PIN_01 (20*) #I/O`
	`C_PIN_02 (25*) #I/O`
	`C_PIN_03 (26*) #I/O`
	`C_PIN_04 (32) #I/O`
	`C_PIN_05 (33) #I/O`
	`C_PIN_06 (34*) #I/O`
	`C_PIN_07 (35*) #I/O`
	`C_PIN_MISO (8)`
	`C_PIN_MOSI (5)`
	`C_PIN_SCK (7)`
	`C_PIN_SDA (21*) #I2C`
	`C_PIN_SCL (22*) #I2C`
	`# Ein-/Ausgänge definieren`
	`initGPIO(C_PIN_01, OUTPUT)`
	`#(oder INPUT)`
	`initGPIO(20, OUTPUT)`
		Initialisiert das gewählte GPIO (analog pinMode):
		OUTPUT,
		INPUT,
		INPUT_PULLUP,
		INPUT_PULLDOWN
		
		*Diese GPIOs gibt es nur bei der Oxocard Connect
		
		Pin01 = OUTPUT
		Pin01 = OUTPUT

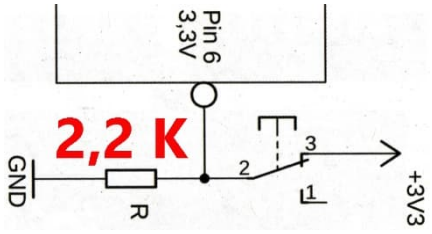
Abfrage Digitalwert	`if readGPIO(BUTTON_PIN):` `  writeGPIO(LED_PIN, 1)`	Abfrage eines vorher als INPUT definierten Pins
Abfrage Analogwert	`readADC(C_PIN_07, 100))`	nur Pin 06 oder 07, 100 mal um den Durchschnitt zu ermitteln
Ausgabe Digitalwert	`writeGPIO(20, 1) #oder` `writeGPIO(C_PIN_01,1)`	20 = verkürzte Angabe von Pin 1 1 = ein, 0 = aus

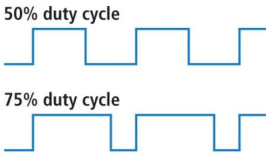
Ausgabe PWM-Signal	`const DUTYCYCLE = 2048`
	`# 0..4096`
	`writePWM(C_PIN_01, DUTYCYCLE)`
		Festlegung des Wertes (0-4096)
		Ausgabe des Wertes an Pin 1

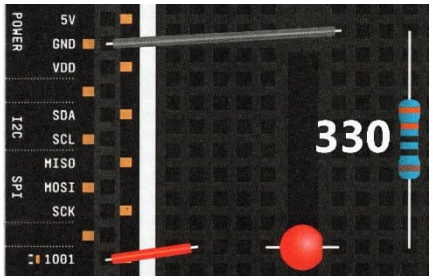
Ausgabe Analogwert	`writeDAC(C_PIN_02, 127)`	gibt auf den Pins _02 oder _03 ein analoges Spannungssignal zwischen 0(0) und 3.3V(255) aus
Frequenz festlegen (kann nur einmal im Programm festgelegt werden)	`setPWMFrequency(Wert)`	Töne = 523 bis 1046 Servo = 50 LED Helligkeit = 20 bis 500

Tonausgabe	`setPWMFrequency(880)`
	`writePWM(C_PIN_05, 4096/2)`
	`delay(1000)`
	`writePWM(C_PIN_05, 0)`
		Frequenz wird festgelegt *
		Ausgabe mit 50% vom DutyCicle

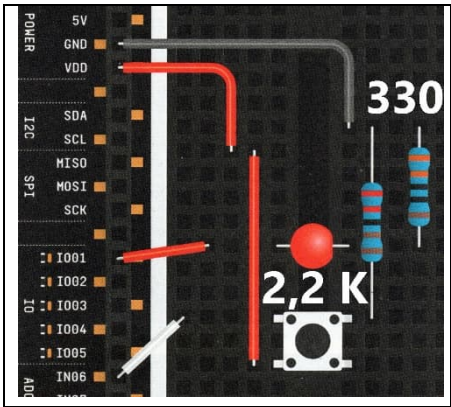
		Tonausgabe (1s) Ausgabe auf 0 (Ausschalten) *C=523,D=587,E=569, F=698 G=784, A=880, H=987,C´=1046
Helligkeit messen	`print(readADC(C_PIN_06, 100))`	Wert zwischen 0 und 4096
Temperatur messen	`adcValue = readADC(C_PIN_06,100)`	Spannung am Spannungsteiler wird abgerufen
Bewegungssensor abfragen	`if readGPIO(C_PIN_05):` `  print("Alarm")`	Wenn Digital-Pin HIGH wird hier Alarm angezeigt

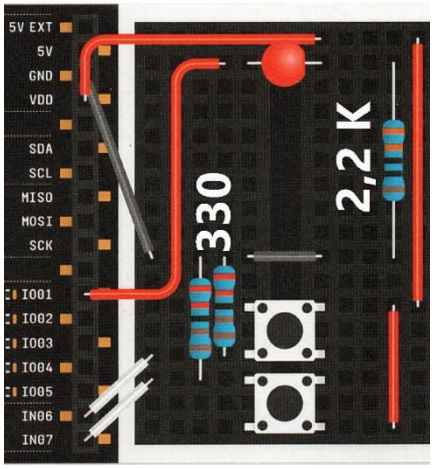
Buttonabfrage
 
	`while true:`
	`  if readGPIO(BUTTON):`
	`  writeGPIO(LED, 1)`
	`  else:`
	`  writeGPIO(LED, 0)`
	`  delay(10)`
		Endlosschleife
		Ist Button gedrückt:
		LED Pin 1 an
		Wenn nicht:
		LED Pin 1 aus
		Pause 10ms

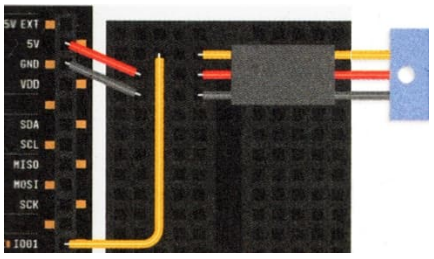
Helligkeit über PWM regeln
 
	`const DUTYCYCLE = 2048`
	`# 0..4096`
	`# const FREQUENCY = 500`
	`# 20..500`
	
	`setPWMFrequency(FREQUENCY)`
	`initGPIO(C_PIN_01, OUTPUT)`
	`writePWM(C_PIN_01, DUTYCYCLE)`
		Tastverhältnis An/Aus bei 500 Hz
		bei Bedarf Änderung der Frequenz
		hier wird neue Frequenz definiert
		Definition Pin1 als Ausgang
		Ausgabe PWM-Signal

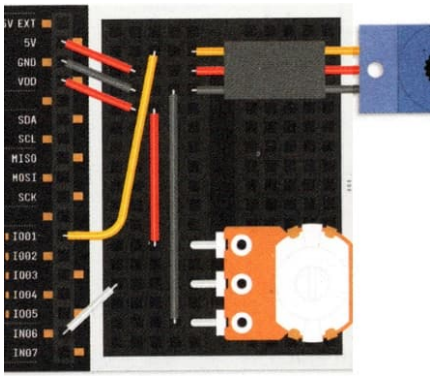
LED als Blinklicht
 
	`const DELAY = 500`
	`# 1..1000`
	`initGPIO(C_PIN_01, OUTPUT)`
	`while true:`
	`  writeGPIO(C_PIN_01, 1)`
	`  delay(DELAY)`
	`  writeGPIO(C_PIN_01, 0)`
	`  delay(DELAY)`
		Konstante zum regeln der Zeit
		Initialisierung Pin 1 als Ausgang
		Endlosschleife
		LED an
		Pause (siehe DELAY)
		LED aus
		Pause (siehe DELAY)

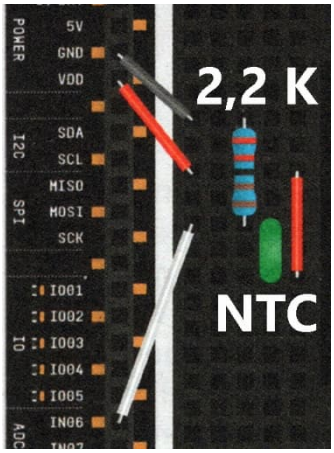
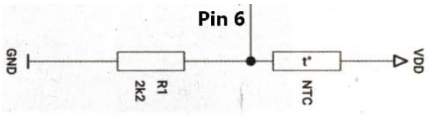
LED mit Taster schalten	`initGPIO(C_LED_01, OUTPUT)`
	`initGPIO(C_LED_06, INPUT)`
	`on = true`
	`while true:`
	`  button = readGPIO(C_PIN_06)`
	`  if button:`
		Pin 1 als Ausgang
		Pin 6 als Eingang
		Variable on wird auf true gesetzt
		Endlosschleife

	`on = not on`
	`delay(500)`
	`writeGPIO(C_PIN_01, on)`
		Einlesen Pin 6
		Button gedrückt:
		Variable wird umgekehrt (true/false)
		Pause (0,5s)
		LED ein- oder ausschalten

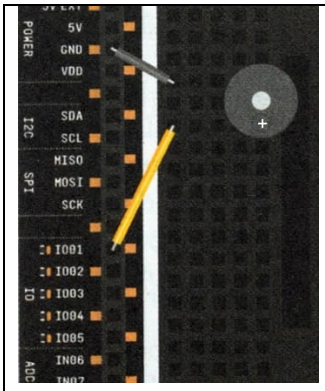
Helligkeitssteuerung mit 2 Tastern
 
	`initGPIO(C_PIN_01, OUTPUT)`
	`initGPIO(C_PIN_06, INPUT)`
	`initGPIO(C_PIN_07, INPUT)`
	`setPWMFrequency(500)`
	`value = 50`
	`while true:`
	`  if readGPIO(C_PIN_06):`
	`  if value<100:`
	`  value=value+1`
	`  if readGPIO(C_PIN_07):`
	`  if value>0`
	`  value=value-1`
	`  pwmValue=map(value,0,100,0,4096)`
	`  writePWM(C_PIN_01, pwmValue)`
	`  delay(10)`
		LED-Pin (01) Ausgang
		Button (06) zum erhöhen
		Button (07) zum reduzieren
		Frequenz 50 Herz
		aktuelle Helligkeit in Prozent
		Endlosschleife
		Button gedrückt:
		Ist Wert unter 100
		Wert um 1 erhöhen
		Button runter gedrückt
		Ist Wert über 0
		Wert um 1 reduzieren
		Wandeln v. Prozent in Wertbereich
		Wert über LED-Pin (01) ausgeben
		Pause 10 ms

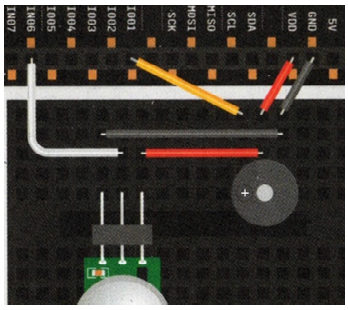
Servosteuerung
 
	`setPWMFrequency(50)`
	`def setAngles(angle):`
	`  dutyCycle=map(angle,90,-90,102,512)`
	`  writePWM(C_PIN_01, dutyCycle)`
	`setAngles(0)`
	`delay(1000)`
	`setAngles(-90)`
	`delay(1000)`
	`setAngles(90)`
		50 Herz für Servosteuerung
		Funktion definieren
		Umwandlung Grad in Impulswert
		Ausgabe auf Pin 1 (Servo)
		Funktionsaufruf: 0°
		Pause (1s)
		Funktionsaufruf: -90°
		Pause (1s)
		Funktionsaufruf: 90°

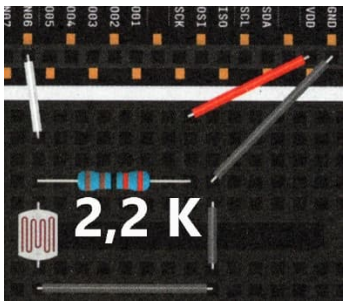
Servo mit Poti
 
	`initGPIO(C_PIN_01, OUTPUT)`
	`initGPIO(C_PIN_07, INPUT)`
	`setPWMFrequency(50)`
	`while true:`
	`  a=readADC(C_PIN_07, 100)`
	`  dutyCycle=map(a,0,4096,102,512)`
	`  writePWM(C_PIN_01, dutyCycle)`
		Servo am Ausgang Pin1
		Poti zur Lenkung auf Pin7
		Frequenz für Servos
		Endlosschleife
		Analogwert von Poti auslesen
		Wandeln von Potiwert in Impulswert
		Ansteuerung des Servos

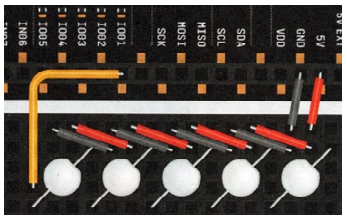
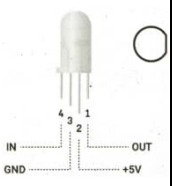
Temperaturmessung
 
 
	`initGPIO(C_PIN_07, INPUT)`
	`const BETA = 4050.0`
	`const T25 = 298.15`
	`const R25 = 10000.0`
	`setPrecision(1)`
	`def drawDisplay(text:byte[]):`
	`  clear()`
	`  textFont(FONT_ROBOTO_48)`
	`  drawTextCentered(120,120,text)`
	`  update()`
	`setInterval(100)`
	
	`def onTimer():`
	`  adcValue = readADC(C_PIN_06,100)`
	`  vr=3.3*adcValue /4095`
	`  vntc=3.3-vr`
	`  rntc=(vntc*2200)/(3.3-vntc)`
	`  tk=1.0/(ln(rntc/R25)/BETA+(1.0/T25))`
	`  t=tk-273.15`
	`  drawDisplay(t)`
		Pin 7 als Analogeingang
		Beta-Wert des NTC (Datenblatt)
		Referenztemperatur
		Ref. Widerstand des NTC bei 25°
		Eine Nachkommastelle ausgeben
		Funktion für Textausgabe
		• Alles löschen
		• Textform
		• Textausgabe mit Position
		• Ausgabe starten
		ruft das onTime()-Event nach ms auf
		Mittelwert aus 100 Messungen
		Spannung messen
		Spannung des NTC
		Widerstand NTC berechnen
		Temperatur in Kelvin berechnen
		Umwandlung in Celsius
		Ausgabe auf Display

Soundausgabe	`const TONE_LENGHT = 200 #10..500` `const TONE_PAUSE = 50 #50..500`	Länge des Tons in ms Länge der Pause Frequenzliste der 5. Oktave

	`freq:int[8] = [523,587,659,698,784,880,987,1046]`
	`melody:int[8] = [1,2,1,2,1,2,3,4]`
	`for m in melody:`
	`  t=freq[m]`
	`  setPWMFrequency(t)`
	`  writePWM(C_PIN_01, 4096/2)`
	`  delay(TONE_LENGHT)`
	`  writePWM(C_PIN_01,0)`
	`  delay(TONE_PAUSE)`
		Liste von Tönen
		Melodieschleife
		Frequenz aus Liste holen
		Frequenz setzen
		DutyCicle von 50%
		Pause Tonlänge
		Ausschalten
		Pause

Bewegungssensor
 
	`initGPIO(C_PIN_06, OUTPUT)`
	`initGPIO(C_PIN_01, INPUT)`
	`def onDraw():`
	`  if readGPIO(C_PIN_06):`
	`  print("Alarm")`
	`  writePWM(C_PIN_01, 4096/2)`
	`  delay(500)`
	`  writePWM(C_PIN_01, 0)`
		Initialisierung Pin 06 (Sensor)
		Initialisierung Pin 01 (Sound)
		Abfrage ob Bewegung erkannt
		Ausgabe “Alarm” auf Terminal
		Ausgabe Alarmton
		Pause (0,5s)
		Alarmton abschalten

Helligkeitsmessung
 
	`setPrecision(2)`
	`def drawDisplay(text:byte[]):`
	`  clear()`
	`  drawTextCentered(120,120, text)`
	`  update()`
	`while true:`
	`  adcValue = readADC(C_PIN_06, 100)`
	`  vldr:float = 3.3 * adcValue /4095`
	`  drawDisplay(vldr)`
		2 Nachkommastellen
		Funktion für Textausgabe
		• Alles löschen
		• Text zentrieren
		• Ausgabe starten
		Dauerschleife
		Mittelwert aus 100 Messungen
		Spannung berechnen
		Ausgabe der Textausgabefunktion

WS-LED
 
	`initDigitalLeds(C_PIN_01,2,C_LED_TYPE_WS2812)`
	`setDigitalLed(0, 255,0,0)`
	`setDigitalLed(1, 255,255,0)`
	`applyDigitalLeds()`
	 
		Initialisierung für 2 LEDs an Pin 01
		RGB-Farbe für LED 1 (0)
		RGB-Farbe für LED 2 (1)
		Ausgabe der LED-Daten (WS2812 B)

Computing

Der ft-RPI-sa – ein moderner BASIC-Controller für fischertechnik (Teil 3)

Robert Lippmann

I2C macht es möglich: Real Time Clock (RTC) und LC-Display am ft-RPI-sa.

Beim nächsten Ton ist es…

Für viele Projekte ist es notwendig, die aktuelle Zeit oder das Datum im Blick zu haben. Deswegen befindet sich auf der ft-RPI-sa Leiterplatte eine Echtzeituhr (RTC), die nicht nur Zeit und Datum liefert, sondern auch zu vorgegebenen Zeiten das Basic Programm beeinflussen oder einfach nur den Raspberry Pi einschalten kann. Darüber hinaus besitzt die RTC 512 Bytes RAM Speicher, der batteriegepuffert über einen längeren Zeitraum erhalten bleibt. Mit einer Genauigkeit von ±3ppm, was auf ein Jahr gerechnet ca. 95 Sekunden entspricht, ist sie zudem sehr genau.

Der Zugriff auf die RTC geschieht in MMBasic mit dem Befehl

```
RTC <Action> <Parameter>
```

wobei `<Action>` und `<Parameter>` folgende Werte annehmen können:

- „`GETTIME`“ um die Uhr auszulesen und die Variablen `DATE$` und `TIME$` entsprechend zu setzen. Dafür sind keine zusätzlichen Parameter notwendig. Diese beiden Variablen werden über Interrupt ständig aktualisiert und dementsprechend reicht es, ein `GETTIME` am Anfang des Programms auszuführen. Allerdings sei erwähnt, dass die Ganggenauigkeit der Variable `TIME$` bei weitem nicht der der RTC entspricht.
- „`SETTIME Jahr, Monat, Tag, Stunde, Minute, Sekunde`“ um die Uhr bzw. das Datum zu setzen. Für die Stunde wird die 24-Stunden-Notation verwendet.

Neben der Bereitstellung der aktuellen Uhrzeit gibt es die Möglichkeit, einen Alarm zu einer bestimmten Uhrzeit auszulösen. Sobald dieser Alarm aktiviert wird, wird je nach Konfiguration der Kurzschlussbrücke JP4 (Abb. 1) der angeschlossene Raspberry Pi eingeschaltet.

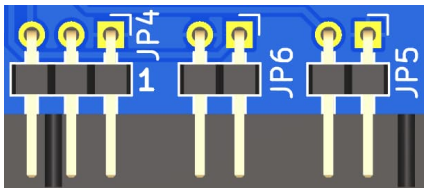


Abb. 1: JP4 für RTC-Konfiguration

Die folgende Tabelle 1 fasst die Konfigurationsmöglichkeiten über JP4 zusammen:

JP4-Brücke	Funktion
1-2	Raspberry Pi immer an
2-3	Raspberry Pi kann über den Taster, die MCU oder einen RTC-Alarm eingeschaltet werden
Ohne	Raspberry Pi immer aus

Tab. 1: RTC-Funktion

- „`SETALARM Tag, Stunde, Minute, Sekunde, Maske`“ um einen Alarm zu setzen. Die Maske schaltet jeden der angegebenen Parameter ein oder aus. Die Maske ist wieder bit-weise kodiert (1 = Ein, 0 = Aus):

Bit #	Dez	Maske für
0	1	Sekunde
1	2	Minute
2	4	Stunde
3	8	Tag

Tab. 2: Alarm-Kodierung

Somit ist es z. B. möglich, mittels

```
RTC SETALARM 1,12,0,0,15
```

an jedem ersten des Monats um Punkt 12:00:00 Uhr einen Alarm auszulösen, da die Maske für jeden Parameter gesetzt ist (1+2+4+8=15), oder mittels

```
RTC SETALARM 1,1,30,27,3
```

jede Stunde und jeden Tag bei 30 Minuten und 27 Sekunden einen Alarm auszulösen, da hier die Maske nur die Minuten und Sekunden berücksichtigt (1+2=3). Für den Tag und die Stunde wird als Dummy eine 1 übergeben, um die Syntax beizubehalten.

Falls ein bereits programmierter Alarm wieder gelöscht werden soll, reicht ein

```
RTC SETALARM OFF
```

- „`CLEARALARM`“ um einen ausgelösten Alarm zu bestätigen bzw. zu löschen. Falls dies nicht geschieht, kann der aufgeweckte Raspberry Pi bzw. dessen Stromversorgung nicht komplett abgeschaltet werden. Hier wird kein weiterer Parameter benötigt.

In MMBasic gibt es eine Systemvariable RTCALARM, die bei einem Alarm auf ‚1‘ gesetzt wird. Somit kann man z. B. ein fischertechnik-Modell zu bestimmten Zeiten aktivieren. Spontan fällt mir dazu das Clubmodell KKSKW '79 aus dem fischertechnik Clubheft 1/79 ein [1].

```
RTC gettime
RTC clearalarm
RTC setalarm 1,7,14,27,7
Do
  Do While (Not RtcAlarm )
  Loop
  RTC clearalarm
  ....
Loop
```

Listing 1: Wecken um 7:14:27 Uhr

- „`SETRAM/GETRAM Adresse, Data`“ um auf das interne, batteriegepufferte RAM der RTC zuzugreifen. Der Bereich der Adresse darf zwischen 0 und 511 liegen und der Bereich für Data zwischen 0 und 255.
- Und schließlich gibt es noch die Möglichkeit, mittels „`SETREG/GETREG Register, Variable`“ auf die internen Register der RTC zuzugreifen, was sich aber erst nach Einlesen in das Datenblatt der RTC empfiehlt.

Eine flexible Schnittstelle

Die IIC, auch I2C oder I^2C genannte Schnittstelle, ist eine serielle Schnittstelle, die ursprünglich dazu gedacht war, Peripheriebausteine auf der gleichen Leiterplatte untereinander zu verbinden, um Daten austauschen zu können. Eigentlich war sie nie dafür ausgelegt, über größere Distanzen Daten zu übertragen, und doch wird sie im Hobbybereich genau dafür verwendet.

Wahrscheinlich sind es das relativ simple Protokoll und die Einfachheit der Anwendung, die dies ermöglichten. So sind auch nur drei Leitungen (inklusive GND) nötig, um zusätzliche Peripherie ansprechen zu können, was den Verdrahtungsaufwand bei einem Modell erheblich vereinfacht. Für weitere Details zu I2C-Protokoll und möglicher Peripherie empfehle ich die schöne Artikelserie von Dirk Fox ab der ft:pedia 3/2012 [2].

Auf dem ft-RPI-sa Controller wird die I2C-Schnittstelle zum einen für die interne Kommunikation der MCU mit der Echtzeituhr sowie der Pull Konfiguration der

Eingänge benutzt, zum anderen für die externen Anschlüsse J5 und J6 (Abb. 2), die die I2C in 3.3V bzw. in 5V der Außenwelt anbieten.

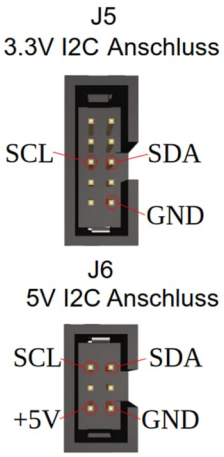


Abb. 2: I2C-Anschlüsse

Falls ein Raspberry Pi verwendet wird, kann dieser die volle Kontrolle über die RTC und die externen Ports übernehmen. Wer Zugriff auf den I2C-Bus bekommt, bestimmen die Kurzschlussbrücken JP2 und JP3 (Abb. 3).

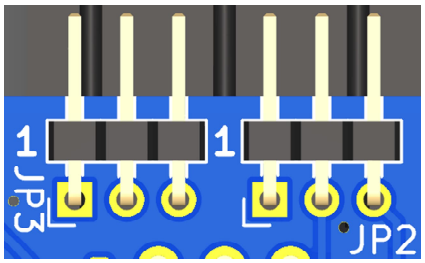


Abb. 3: JP2/JP3 für die I2C-Konfiguration

Sobald sich je eine Kurzschlussbrücke zwischen den Kontakten 1-2 befindet, ist die MCU der Master (der Standard für MMBasic), und je eine Brücke zwischen 2-3 übergibt dem Raspberry Pi die Funktion des Masters.

An dieser Stelle sollte ich darauf hinweisen, dass die bei I2C üblichen Bezeichnungen Master/Slave im Jahre 2021 in Controller/Target umbenannt wurden.

Wie kann nun in MMBasic auf ein über die I2C-Schnittstelle angeschlossenes Gerät zugegriffen werden? Die speziellen Befehle für den Zugriff auf die interne RTC und die Konfiguration der Pull-Widerstände wurden bereits oben bzw. im letzten Beitrag ausführlich besprochen.

Um mit der I2C-Schnittstelle auf andere Peripherieelemente an Port J5 bzw. J6 zugreifen zu können, muss diese erst initialisiert werden mit:

```
I2C OPEN bitrate, timeout
```

Die bitrate kann dabei im Bereich zwischen 10 und 400 liegen, was 10kBit/s bzw. 400kBit/s entspricht. Manche Bitraten lassen sich nicht genau einstellen und werden aufgerundet. Das timeout bestimmt die Zeit in ms, nach der ein Übertragungsversuch abgebrochen wird, wenn kein erfolgreicher Versand bzw. Empfang der Daten stattgefunden hat.

```
I2C WRITE adresse, option, länge,
data [,data...]
```

Dieser Befehl sendet ein oder mehrere Daten an das Target mit der Adresse `<adresse>`. Mit `<option>` kann das I2C-Protokoll beeinflusst werden (Tab. 3).

<option> Wert	Auswirkung
0	Standard-Transfer
1	Kein Stoppbit nach dem Transfer
2	10-Bit-Target-adresse
3	Kombiniere Option 1 & 2

Tab. 3: I2C-Optionen

Bis jetzt ist mir allerdings noch kein Fall untergekommen, bei dem ich Modus 1 verwenden musste, und mangels 10-Bit-Peripherie konnte ich Modus 2 noch nicht testen.

```
I2C READ adresse, option, länge,
buffer
```

Das Kommando entspricht im Wesentlichen dem `WRITE`-Befehl, mit dem Unterschied, dass hier keine Daten- bzw. Variablenkette angegeben werden kann. Für den `WRITE`- und den `READ`-Befehl gibt es eine viel einfachere Möglichkeit, mehrere Datenbytes zu versenden:

- Angabe eines Strings
- Angabe eines eindimensionalen Arrays

Somit kann mittels

```
I2C READ &H6F,0,20,A$
```

der String `A$` mit 20 Datenbytes, die über I2C kommen, gefüllt werden. Gleiches passiert bei Verwendung von

```
I2C READ &H6F,0,20,A()
```

welches das Zahlenarray `A` mit 20 Datenbytes füllt.

MM.I2C Wert	Status
0	Kein Fehler
1	Ein NACK (No ACKnowledge) wurde empfangen oder keine Bestätigung vom Target erhalten
2	Timeout-Fehler

Tab. 4: I2C-Status

Nach einem I2C-Transfer wird der Status in der Systemvariable MM.I2C (Tab. 4) gespeichert. Das letzte I2C-Kommando ist dafür gedacht, die verwendete Schnittstelle nach einem Transfer wieder zu schließen:

```
I2C CLOSE
```

Dies geschieht in der Regel am Programmende.

Am Schluss der I2C-Befehlsbeschreibung möchte ich noch erwähnen, dass die Target-Adressen &H51 und &H20 für die RTC und die Pull-Konfiguration bereits vergeben sind.

Wie sieht es nun aus, wenn man unter MMBasic eine I2C-Peripherie ansteuern will? Dazu ein praktisches Beispiel mit einem LC-Display, so wie ich es auch z. B.

Exkurs: I2C-Adressierung

7- und 8-bit-Adressierung

Die Standard-I2C-Adressen verwenden 7-bit-Adressierung (ohne R/W-Bit), wobei MMBasic je nach Transferrichtung das RW-Bit entsprechend anpasst.

Einige Halbleiterhersteller geben eine 8-bit-Adresse an, die das R/W-Bit beinhaltet. Das kann festgestellt werden, wenn unterschiedliche Bereiche für Schreib- bzw. Lesebereiche angegeben werden. In diesem Fall dürfen nur die oberen 7 Bits als Adresse verwendet werden.

Beispiel: Ist als zu lesende Adresse 0x9b und als zu schreibende 0x9a angegeben, nimmt man die oberen 7 Bits als eigentliche Adresse, die somit zu 0x4d wird. Einfacher ist es, den 8-bit-Wert einmal um eine Bit-Position nach rechts zu schieben, was einem Teilen durch 2 gleichkommt: Aus 0x9b = 0b10011011 wird, um eins verschoben, 0b01001101 = 0x4d.

Eine andere Methode der Identifikation besteht darin, den Wert der Adresse zu überprüfen. Dieser sollte innerhalb des Bereiches von 0x08 bis 0x77 liegen, andernfalls hat der Hersteller eine 8-bit-Adresse angegeben.

Anmerkung: Adressen in den Bereichen 0b0000xxx (0x0-0x7) sowie 0b1111xxx (0x78-0x7f) sind reserviert.

10-bit-Adressierung

Die 10-bit-Adressierung ist kompatibel zur 7-bit-Adressierung, um es zu ermöglichen, beide Varianten auf einem Bus zu nutzen. Bei der 10-bit-Adressierung werden immer zwei Adressbytes verwendet, wobei das erste ein spezielles Muster (0b11110xx aus dem reservierten Bereich) enthält, um anzuzeigen, dass 10-bit-Adressierung verwendet wird. Dies macht MMBasic automatisch, sobald die entsprechende Option genutzt wird. 10-bit-Adressen können im Bereich von 0x0 bis 0x3ff liegen.

für meinen Farbsortier-Roboter verwendet habe [3].

Das hier verwendete Display ist ein 4 x 16 Zeichen großes Standarddisplay (Abb. 4) mit HD44780 Controller, das mittels einer kleinen Zusatzelektronik (Abb. 5) I2C-tauglich gemacht wird [4].

Abb. 4: LC-Display

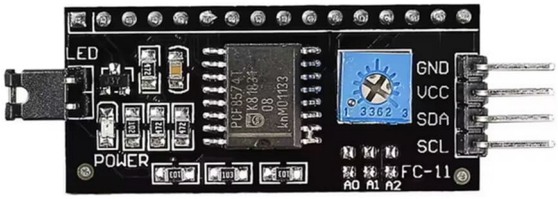


Abb. 5: I2C-Adapter

Der I2C-Adapter wird auf die 16polige Leiste des Displays gesteckt und somit I2C-tauglich. Um die Konvertierung von I2C auf das Display kümmert sich ein PCF8574T, der nichts anderes ist als ein 8-bit I/O Expander mit I2C-Schnittstelle.

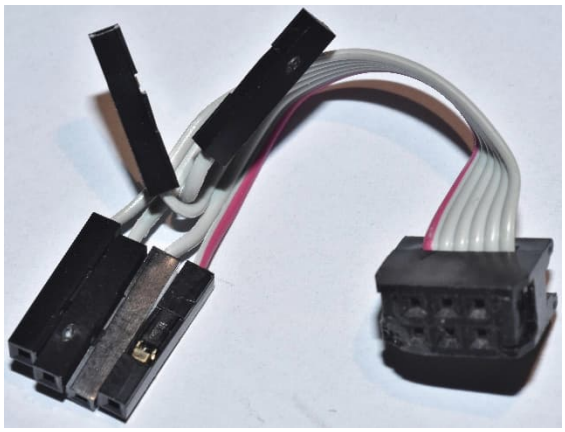


Abb. 6: I2C-Kabel

Für die Verbindung zum ft-RPI-sa verwenden wir ein 6poliges Kabel mit aufgequetschter Pfostenbuchse, passend für J6 (5V I2C), am anderen Ende die vier benötigten Leitungen mit Einzelkontakten zum Anschluss an den I2C-Adapter (Abb. 6).

Nun müssen wir noch wissen, welcher Port-Pin des PCF8574T welchen Pin des LC-Displays ansteuert. Da hilft ein Blick in den Schaltplan des Adapters (Abb. 7).

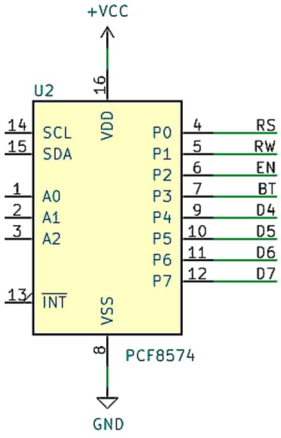


Abb. 7: PCF8574T-Pinbelegung

Auf die Bedeutung der einzelnen Signale gehe ich hier nicht ein, da dies Dirk schon hinreichend getan hat [4].

Der erste Teil des Programms beinhaltet die Variablendeklaration sowie das Hauptprogramm mit den benötigten Funktionsaufrufen:

```
LCDI2cAddr = &H27 ' PCF8574T I2C
Addr
RSCmask = &B00000000 ' Command
mask
RSDmask = &B00000001 ' Data mask
ENmask  = &B00000100 ' Enable mask
BLmask  = &B00001000 ' Backlight
mask
I2C close ' Just to avoid conflict
I2C OPEN 100,100 ' Open
I2C@100kBit/s
InitLCD ' Initialize LC Display
DisplayText 1, 2, „Welcome to the“
DisplayText 2, 2, „ft-RPI-sa Ctrl“
DisplayText 3, 1, „Greetings to
the“
DisplayText 4, 3, „ft community“
End
```

Listing 2: LCD -Hauptprogramm

Für den Zugriff auf das LC-Display fangen wir in der Hierarchie der Unterprogramme ganz unten an:

```
Sub SendByte POut
  POut = POut Or ENmask
  I2C WRITE LCDI2CAddr, 0, 1, POut
  POut = POut Xor ENmask
  I2C WRITE LCDI2CAddr, 0, 1, POut
End
```

Listing 3: 1-Byte Übertragung

Bei dem übergebenen Byte POut, handelt es sich um den Inhalt, den der PCF8574T an seinen Ports ausgibt:

Bit	7	6	5	4	3	2	1	0
Pin	D7	D6	D5	D4	BT	EN	RW	RS

Tab. 5: PCF8574T-Portpinbelegung

Um nun ein Byte in den Displaycontroller zu schreiben (Abb. 8) muss, während dieses anliegt, am EN-Eingang des LCDs ein Puls erzeugt werden. Dies geschieht dadurch, dass wir die ENmask mittels OR setzen, danach das Byte übertragen (Zeitpunkt 1) und anschließend, jetzt ENmask mittels XOR wieder gelöscht, nochmals an den PCF übertragen (Zeitpunkt 2). Die Übernahme der Daten geschieht mit der fallenden Flanke an EN. Der Zustand der anderen drei Portpins RS, RW und BT wird von der übergebenen Variable POut übernommen, aber darauf komme ich gleich zu sprechen.

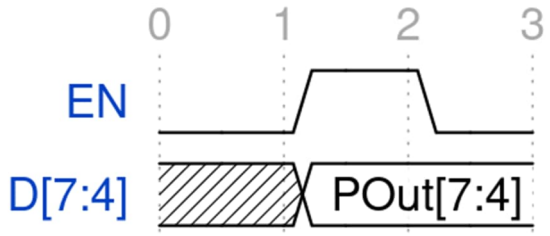


Abb. 8: PCF8574T EN/Daten-Pins

Was bereits in Tab. 5 und auch in Abb. 8 auffällt, ist die Tatsache, dass wir nur vier Datenbits für die Übertragung zur Verfügung haben. Das muss kein Nachteil sein, da in diesem Fall deutlich weniger Leitungen und weniger Hardware nötig ist, allerdings wird für die Übertragung der Daten doppelt so viel Zeit benötigt. Dies hat auch Einfluss auf die Unterroutinen, die die Daten und Kommandos an den Controller übertragen:

```
Sub SendCmdByte aByte
  SendByte ((aByte And &HF0) Or
  RSCmask Or BLmask)
  SendByte (((aByte And &H0F) <<
  4) Or RSCmask Or BLmask)
End Sub

Sub SendDataByte aByte
  SendByte ((aByte And &HF0) Or
  RSDmask Or BLmask)
  SendByte (((aByte And &H0F) <<
  4) Or RSDmask Or BLmask)
End Sub
```

Listing 4: Kommando- und Datenübertragung

Beide Unterprogramme splitten das übergebene Datenbyte in zwei Nibble [5] auf und übertragen beide hintereinander (Abb. 9). Das Aufsplitten wird im ersten Schritt durch das herausUNDen der vier oberen Bits mit &HF0 aus dem übergebenen Datenbyte erreicht. Im zweiten Schritt wird durch herausUNDen der vier unteren Bits durch &H0F und anschließend mittels „<< 4“ um vier Bitpositionen auf die höherwertigen Bits 7 bis 4 verschoben. Nibble 1 beinhaltet somit die Datenbits [7:4] und Nibble 2 die Datenbits [3:0] des zu übertragenen Bytes.

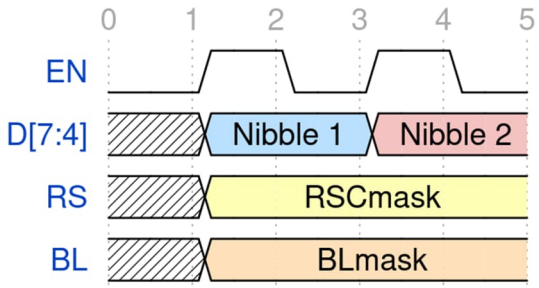


Abb. 9: PCF8574T-Byte-Transfer

Die beiden Unterprogramme unterscheiden sich nur im Wert des RS-Bits, das die Unterscheidung zwischen Kommando (= 0) und Daten (= 1) trifft. Das RW-Bit ist immer auf „0“ gesetzt, da wir auf das Display nur schreiben wollen. Auf das Lesen des „Busy“-Flags, das anzeigt, ob der HD44780 Controller beschäftigt ist, kann verzichtet werden, da die Übertragung eines Kom-

mando- oder Datenbytes immer länger dauert als dessen Ausführung. Allerdings kann die Ausführung eines „Clear Display“- oder „Return Home“-Kommandos bis zu 64ms benötigen. Daher ist es ratsam, eine kleine Pause nach deren Übermittlung einzufügen (Listing 5). Das BL-Bit setzen wir hier immer auf „1“, damit uns, falls wir ein Display mit Hintergrundbeleuchtung haben, ein Licht aufgeht.

Um das LC-Display zu initialisieren müssen zuerst bestimme Werte in die entsprechenden Register des Displaycontrollers geschrieben werden (Listing 5). Wie diese Werte zu interpretieren sind, hat Dirk in seinem Beitrag [4] bereits hinreichend beschrieben. Details dazu sind auch im Datenblatt des Displaycontrollers [6] zu finden.

```
Sub InitLCD
  SendCmdByte &B00110011
  SendCmdByte &B00110010
  SendCmdByte &B00101000
  SendCmdByte &B00001000
  ' Clear Display
  SendCmdByte &B00000001
  Pause 60 ' 60ms Delay required
  SendCmdByte &B00000110
  SendCmdByte &B00001100
End Sub
```

Listing 5: LCD-Initialisierung

Zuerst muss nach dem Einschalten der Displaycontroller von einem 8-Bit- auf einen 4-Bit-Betrieb umgeschaltet werden. Da jedes übergebene Byte in zwei Hälften zerlegt und diese nacheinander übertragen werden, sieht die Übertragung aus Listing 5 wie in Abb. 10 gezeigt aus.

Dem einen oder anderen mag es schon aufgefallen sein, aber wir programmieren unseren Controller auf ein 2-zeiliges Display, obwohl wir doch ein 4-zeiliges verwenden. Der HD44780 war ursprünglich dafür ausgelegt, maximal zwei Zeilen à 40 Zeichen darzustellen. Das hindert aber niemanden daran, auch vier Zeilen à 20 Zeichen mit diesem Controller darzustellen.

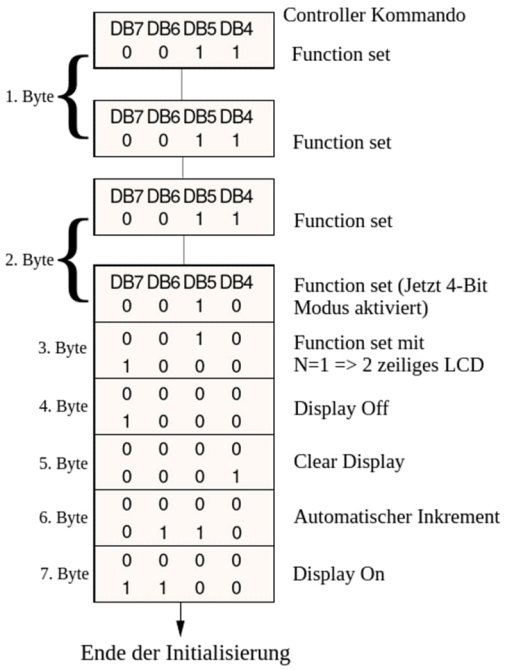


Abb. 10: HD44780 Initialisierung

Allerdings verteilt sich der Speicher für die einzelnen Zeilen je nach Konfiguration (Tab. 6), die es im Programm zu berücksichtigen gilt:

Displaytyp	Anfangs - Endadresse (HEX)			
	1.Zeile	2.Zeile	3.Zeile	4.Zeile
1x8	$00-$07			
1x16	$00-$0F			
1x16(8+8)	$00-$07			
	$40-$47			
1x20	$00-$13			
1x40	$00-$27			
2x8	$00-$07	$40-$47		
2x12	$00-$0B	$40-$4B		
2x16	$00-$0F	$40-$4F		
2x20	$00-$13	$40-$53		
2x24	$00-$17	$40-$57		
2x40	$00-$27	$40-$67		
4x16	$00-$0F	$40-$4F	$10-$1F	$50-$5F
4x20	$00-$13	$40-$53	$14-$27	$54-$67

Tab. 6: LCD-Speicherzuordnung

Jetzt fehlt uns nur noch ein geeignetes Unterprogramm, um Texte auf unserem Display zu platzieren. Mit der Anpassung an den verwendeten Speicherbereich sieht unser letztes benötigte Unterprogramm so aus:

```
Sub DisplayText row, col, text$
Local I, l1h
l1h = &B10000000
Select Case row
  Case 1
  SendCmdByte(l1h + col-1)
  Case 2
```

```
  SendCmdByte l1h + &H40 + col-1)
  Case 3
  SendCmdByte(l1h + &H10 + col-1)
  Case 4
  SendCmdByte(l1h + &H50 + col-1)
End Select
  For I = 1 To Len(text$)
    SendDataByte Asc(Mid$(text$,
I, 1))
  Next I
End Sub
```

Listing 6: LCD-Textausgabe

Der initiale Wert von l1h entspricht dem Kommando „`Set DDRAM address`“ des HD44780, und abhängig von der selektierten Zeile „row“ wird die gewählten Spalte „col“ und der entsprechende Speicheroffset dazu addiert. Die „-1“ ist persönlicher Natur, da bei mir die Spalten bei 1 beginnen und nicht bei 0. Zu guter Letzt wird jedes Zeichen des übergebenen Strings in den angewählten Speicherbereich des LCD-Controllers übertragen.

Als Ergebnis finden wir unseren Text somit mittels I2C übertragen auf unserem LC-Display (Abb. 11).

Abb. 11: LC-Display mit Text

Der LCD-Controller besitzt noch weitere Funktionen und Kommandos, wie das Erstellen eigener Zeichen oder Verschieben des Displayinhaltes, auf die ich hier nicht weiter eingehen möchte. Wer die Unterprogrammfunktionen erweitern will, dem kann ich das Datenblatt des HD44780 empfehlen [6].

Der Hardware-Nachbau

Nach viel Theorie und Beschreibung möchte ich noch ein klein wenig auf den Nachbau des ft-RPI-sa eingehen.

Eines vorab: Das ist kein Projekt für Anfänger, da doch viele kleine Bauteile verbaut und die Abstände der Anschlüsse bei den Halbleitern mit 0.5mm sehr klein sind. Löterfahrungen und eine entsprechend gut ausgestattete Werkstatt sind hier durchaus hilfreich.

Alle für einen Nachbau benötigten Daten, Infos und Unterlagen kann man in Kürze auf github finden [7]. Darunter sind auch die nötigen Daten zur Fertigung der Leiterplatten, die sogenannten Gerber-Files. Gerber ist ein standardisiertes Format, das jede Firma, die sich mit Leiterplattenfertigung beschäftigt, akzeptiert. Die Leiterplatte selbst ist eine vierlagige Variante in der Größe des Raspberry PIs, 85×56mm^2.

Eine vierlagige Variante ist heutzutage nichts Exotisches mehr und relativ günstig herzustellen. Ich habe fünf dieser Platinen für rund 15 Euro fertigen und liefern lassen. Dazu kommt noch das I/O-Board, das aber mit nur zwei Lagen und einer Abmessung von 67×52mm^2 deutlich simpler gestaltet und somit auch günstiger ist. Einige Leiterplattenfirmen bieten auch einen Bestückungsservice an, mit dem man sich das Auflöten der Bauteile erspart, jedoch ist das doch für einen Bastler gerade der Reiz an der Sache.

Letztendlich muss jeder für sich selbst entscheiden, ob er sich dieser Herausforderung stellt oder nicht.

Es ist wenig sinnvoll, hier eine detaillierte „How-To“-Anleitung zu erstellen, da dies weit am Thema fischertechnik vorbei geht und sehr speziell ist. Wenn ihr Fragen zum Nachbau des ft-RPI-sa habt oder den einen oder anderen Tipp zu bestimmten Aspekten des Nachbaus braucht, könnt ihr mich gerne kontaktieren.

Die Soft- bzw. Firmware

Die gesamte Entwicklung der ft-RPI-sa Firmware geschieht mittels der schon in Teil 1 [8] erwähnten NXP Software „S32 Design Studio“ oder auch einfach nur S32DS [9]. Diese basiert auf Eclipse [10] und ist für Windows und Linux kostenlos erhältlich. Wenn das Projekt einmal in S32DS importiert wurde, muss die „Build Configuration“ auf die verwendete MCU eingestellt werden. Dies kann entweder eine MCU des Typs S32K142, S32K144 oder S32K146 sein. Danach wird durch Drücken von „Strg+B“ der Compilevorgang gestartet. Ich habe auf github auch fertig compilierte Versionen für jede MCU bereitgestellt.

Sobald die Hardware steht, muss die generierte Software oder auch Firmware in den Mikrocontroller geladen werden. Dafür wird ein kleines Gerät benötigt, auch Programmer (oder Debugger) genannt, das den PC mit dem Mikrocontroller verbindet. Dieser Programmer muss auch von der Entwicklungsumgebung auf dem PC unterstützt werden. Leider ist die Mehrheit dieser Programmer recht teuer (> 300 €) und scheidet damit für den Hobbygebrauch aus. Es gibt jedoch auch eine preisgünstige, von S32DS unabhängige Variante, und die heißt USBDM [11]. Das ist ebenfalls ein Open-Source-Projekt. Basierend auf diesem Projekt sind fertige Varianten (Abb. 12) günstig (ca. 8 €) im Internet erhältlich.

Abb. 12: USBDM Programmer

Als Verbindung zum ft-RPI-sa werden nur drei Leitungen benötigt: SWCLK, SWDIO und GND. Nach Starten des „ARM_Flash Programmer“, der Teil der USBDM-Software ist, meldet sich dieser mit folgendem Fenster:

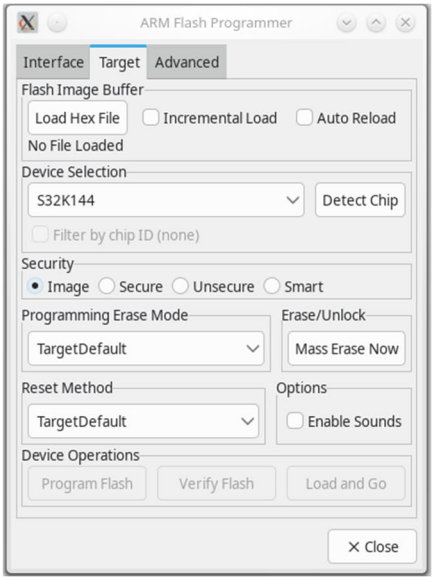


Abb. 13: USBDM ARM Flash Programmer

Nun muss das Device entsprechend der MCU, die auf dem ft-RPI-sa aufgelötet ist, ausgewählt werden und sollte dem entsprechen, was vorhin unter S32DS gewählt wurde. S32DS kann bei einem Compilevorgang eine sogenannte S-Record-Datei (.srec) ausgeben [12]. Diese entspricht einem Standard, der seinerzeit von Motorola definiert wurde, und beinhaltet das binäre Programm in hexadezimaler Form mit zusätzlicher Prüfsumme. Diese S-Record-Datei wird durch Selektieren des „Load Hex File“ Buttons in den Programmer geladen. Alle anderen Einstellungen werden nicht verändert. Durch Anwählen von „Program Flash“ landet unsere Firmware im Flashspeicher der MCU und kann danach durch Drücken des RESET Buttons auf der ft-RPI-sa Platine neu gestartet werden.

Sobald die Firmware mittels USBDM-Software erfolgreich auf den Mikrocontroller übertragen wurde, kann der USBDM auch schon wieder hinten in der Schublade

verschwinden, denn die Firmware bietet eine eigene, weniger aufwändige Methode, die Firmware zu programmieren.

Updates installieren

In der Firmware des ft-RPI-sa ist ein sogenannter Bootloader programmiert. Dieser wird, wenn die Hardware entsprechend konfiguriert ist, anstatt des MMBasic gestartet, und danach kann dieser eine neue Version der Firmware über das Terminal empfangen und im Speicher der MCU ablegen.

Um diesen Bootloader zu starten, muss auf die Anschlüsse JP5 und JP6 (Abb. 1) je eine Kurzschlussbrücke gesteckt werden. Sobald beide Brücken gesteckt sind und der ft-RPI-sa eingeschaltet wird, meldet er sich mit dem Bootloader:

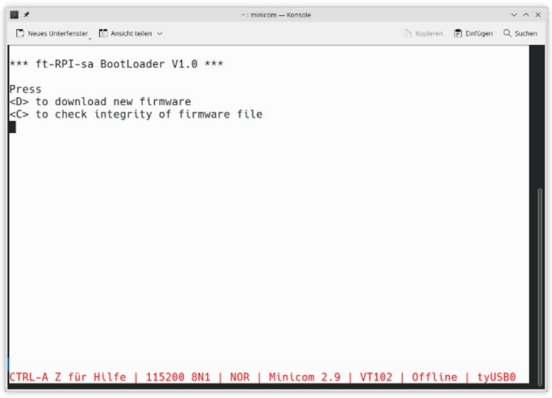


Abb. 14: Bootloader prompt

Es empfiehlt sich jede neue S-Record Version zuvor erst einmal mit „C“ zu prüfen. Wenn dies erfolgreich war, kann die neue Version mit „D“ auf den ft-RPI-sa übertragen werden. Falls aus irgendeinem Grund, wie z. B. einem Stromausfall, Computerdefekt oder dem Laden einer falschen Firmware der Updateprozess fehlgeschlagen ist, muss wieder der USBDM ran und das Dilemma korrigieren. Also nicht zu weit hinten in der Schublade verstecken…

Quellen

[1] fischertechnik: *Clubmodell KKSKW '97*, Clubheft 1/79 (ft-Datenbank.de).

[2] Dirk Fox: *I²C mit TX und Robo Pro – Teil 1: Grundlagen*. ft:pedia 3/2012, S. 32–37.

[3] Youtube: *ft-RPI-sa*, Demovideo.

[4] Dirk Fox: *I²C mit dem TX – Teil 9: LC-Displays*. ft:pedia 1/2014, S. 47–57.

[5] Wikipedia: *Nibble*.

[6] Wikipedia: *HD44780*.

[7] Github: *Projektseite des ft-RPI-sa*.

[8] Robert Lippmann: *Der ft-RPI-sa – ein moderner BASIC-Controller für fischertechnik (1)*. ft:pedia 4/2024, S. 82–87.

[9] NXP: *S32 Design Studio*.

[10] Wikipedia: *Eclipse*.

[11] NXP: *USBDM*, v4.12, Programmer und Debugger für NXP Mikrocontroller, sourceforge.net.

[12] Wikipedia: *S-Record*.

fischertechnik

Modellschau

Mechanik – Statik – Robotik

www.ftcommunity.de

Email: convention@ftcommunity.de

20. September 2025, 10:00-17:00 Uhr

- Eintritt frei -

Martinskirche, Schlippental

Bad Hersfeld