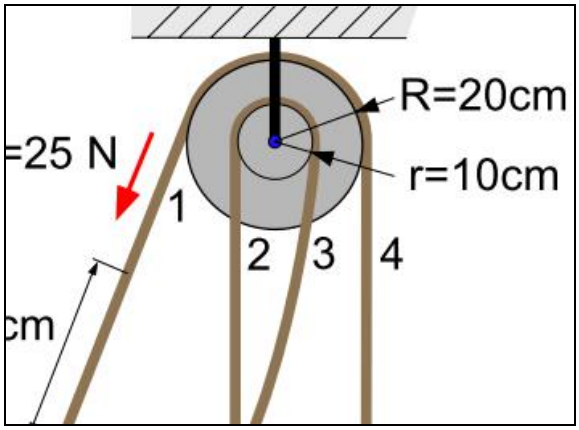
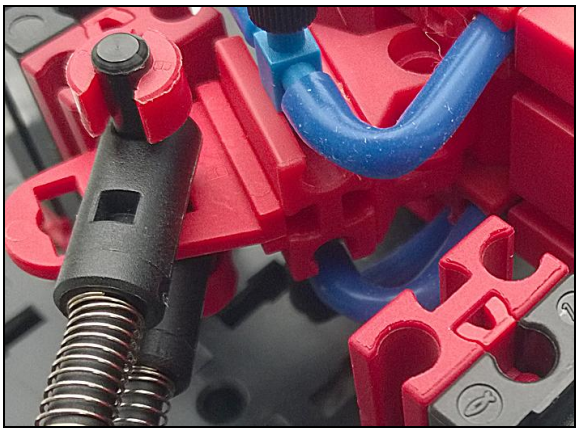
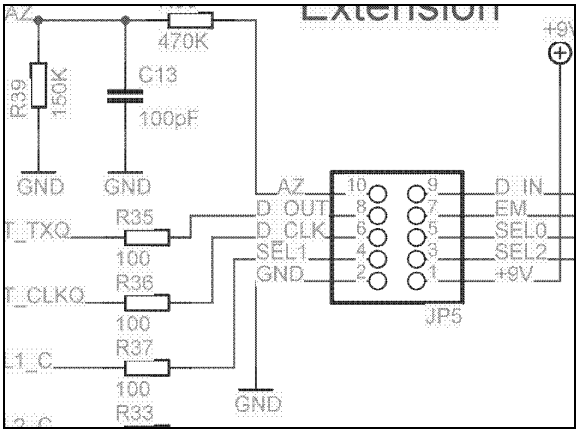
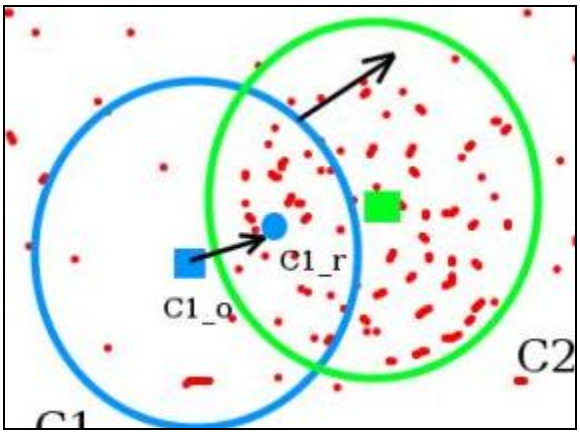
ft:pedia

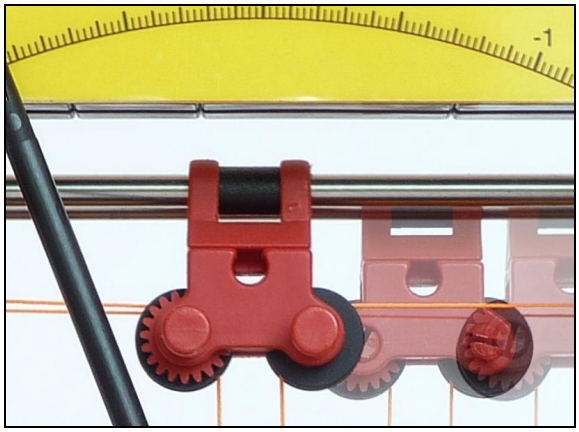
Heft 2/2014











Herausgegeben von
Dirk Fox und Stefan Falk

ISSN 2192-5879

Editorial

Zeitbombe

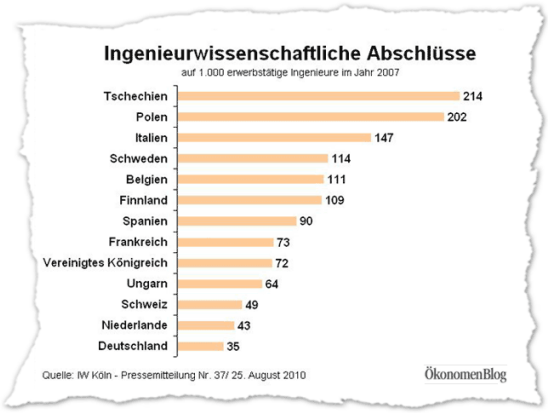
Dirk Fox, Stefan Falk

Die Klage der Industrie über den drohenden – oder gar bereits akuten – ‚Fachkräftemangel' erobert seit Jahren mit steter Regelmäßigkeit und bevorzugt in nachrichtenarmen Zeiten die Schlagzeilen. Kann damit mehr gemeint sein als die wenig überraschende Einsicht, dass ein Geburtenrückgang in der Regel weniger Kinder zur Folge hat – und weniger Kinder zu weniger Fachkräften werden? Oder steckt der auch nicht ganz unerwartete Arbeitgeberreflex dahinter, die Verschiebung von Angebot und Nachfrage zu ihren Ungunsten lautstark zu betrauern?

Wir sind der Sache einmal nachgegangen – und haben ein paar Zahlen aufgestöbert, die zu denken geben sollten.

Die deutsche Exportquote – der Wert aller exportierten Güter im Verhältnis zum Bruttoinlandsprodukt (BIP) – ist von 33 % im Jahr 2000 auf 52 % in 2012 geradezu explodiert. Unser Wohlstand hängt damit unmittelbar an weltweit wettbewerbsfähigen Exportgütern – die von Facharbeitern gefertigt und von deutschen Ingenieuren entwickelt wurden.

Diese Ingenieure werden aber immer älter. Die meisten sind derzeit 45-55 Jahre alt, also in ihren besten Jahren – und werden in den kommenden 10-15 Jahren ihre aktive Berufszeit beenden. Danach sieht es trübe aus – denn im internationalen Vergleich belegt Deutschland beim MINT-Nachwuchs hintere Plätze (Abb. 1). Zwar wählten 2013 immerhin 20% aller Studienanfänger ein ingenieurwissenschaftliches Fach; im Schnitt schaffen aber nur zwei Drittel davon einen Abschluss.



Obwohl Studien belegen, dass das Interesse an Technik maßgeblich von einem spielerischen Umgang mit Technik in der Kindheit (technisches Spielzeug, basteln und reparieren) gefördert wird, leisten wir uns inzwischen weitgehend von echter Technik freie Kinderzimmer und Schulen.

Den Preis für diese Vernachlässigung des einzigen Rohstoffs, den wir in Deutschland haben – technisches ‚Genie' – werden wir in etwa 20 Jahren zahlen. Dann ist es für ein Umsteuern aber zu spät.

Einer, der das schon vor knapp 50 Jahren erkannt hat, ist am 17.06.2014 in Berlin mit dem European Inventor Award des Europäischen Patentamts für sein Lebenswerk ausgezeichnet worden: Artur Fischer. Der Presse war das keine große Erwähnung wert. Nicht einmal der Homepage der fischer-Werke.

Ihm sei diese Ausgabe gewidmet.

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.

Inhalt

Termine

Was?	Wann?	Wo?
Maker Faire 2014	05.-06.07.2014	Hannover
FanClub-Tag	27.07.2014	Waldachtal
Inspiration Modellbau	20.-21.09.2014	Mainz
fischertechnik Convention 2014	27.09.2014	Erbes-Büdesheim

Hinweis

Seit Anfang Juni 2014 ist ein RoboPro-Update auf Version 3.2.6 verfügbar (lauffähig unter Windows 8.x).

Impressum

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

Autoren: Marco Ahlers (funmca), Erik Andresen, Stefan Falk (steffalk), Dirk Fox (Dirk Fox), Johann Fox, Andreas Gail, Jens Lemkamp (datjens), Thomas Püttmann (geometer), Andreas Tacke (TST), Dirk Uffmann (uffi), Ad van der Weiden (Ad2).

Mechanik

Flaschenzug

Dirk Fox

Das Problem kennt ihr zweifellos auch: Ein Motor soll eine Last hochziehen – aber nichts tut sich, weil er zu „schwach auf der Brust“ ist. Was tun? Da lohnt ein Blick in die Technikgeschichte – denn für dieses Problem hatten schon unsere Vorfahren vor über 2.500 Jahren eine wirksame mechanische Lösung.

Die Entwicklung einer Mechanik, mit der es gelang, die Kraftwirkung zu vergrößern, war eine der wichtigsten Entdeckungen in einer Zeit, in der als Kraftquelle nur die Muskelkraft von Tieren oder Menschen zur Verfügung stand. Neben dem Hebelgesetz, das bereits in der Antike von dem griechischen Mathematiker und Physiker *Archimedes von Syrakus* (287-212 v. Chr.) aufgestellt wurde und die Grundlage der Mechanik bildet, war es vor allem die Erfindung des Flaschenzugs, die insbesondere die Bautechnik revolutionierte. Mit dessen Hilfe gelangen vor allem in der Renaissance architektonische Meisterleistungen, die noch heute beeindrucken.

Die Funktion eines Flaschenzugs – dessen Bezeichnung übrigens nichts mit Gefäßen für Flüssigkeiten zu tun hat, sondern von den Rollenhalterungen stammt, die dieselbe Bezeichnung trugen – ist schnell erklärt.

Die für eine bestimmte Hubarbeit – das Anheben eines bestimmten Gewichts um eine definierte Höhe – erforderliche Kraft lässt sich über die Länge des zu überwindenden „Hubweges“ steuern, da die Hubarbeit sich als Produkt aus Kraft und Weg bestimmt: Mit einem längeren Hubweg benötigt man weniger Kraft für dieselbe Hubarbeit. Ein Flaschenzug verlängert nun künstlich den Hubweg, genauer: die Länge des für die Leistung der Hubarbeit aufzuwickelnden Zugseils. Damit ist weniger Kraft für die Hubarbeit erforderlich. Der Preis, den man für diese „Kraftverstärkung“ zahlt: man muss länger ziehen oder kurbeln.

Faktorenflaschenzug

Wenn wir heute von einem Flaschenzug sprechen, meinen wir in der Regel einen Faktorenflaschenzug, der die Seillänge durch „Schlingen“ künstlich verlängert.

Abb. 1: Flaschenzug (aus: Brockhaus' Kleines Konversations-Lexikon, 5. Aufl., Bd. 1, S. 587)

Schon ein einfacher Flaschenzug mit einer Schlinge verdoppelt die Länge des Zugseils und halbiert damit die benötigte Kraft: Ein Mensch, der maximal 50 kg

bewegen kann, kann mit einem solchen Flaschenzug bis zu 100 kg Last anheben.

In der Antike wurden Flaschenzüge von Griechen und Römern in einfachen Kränen eingesetzt. Der römische Ingenieur *Marcus Vitruvius Pollo* (ca. 80-15 v. Chr.) beschrieb den zu seiner Zeit verbreiteten *Trispastos*, einen einfachen Kran mit Drei-Rollen-Flaschenzug, der die Kraft des Bedieners mit einem zusätzlichen Hebel insgesamt etwa verzwölffachte (Abb. 2).

Abb. 2: Römischer Trispastos
(Quelle: Eric Gaba, Wikipedia [1])

Solche Flaschenzüge waren wahrscheinlich schon seit etwa 750 v. Chr. bekannt und kamen auf Baustellen und im Theater zum Einsatz. Dabei sorgte der einfache Flaschenzug für eine Verdreifachung der Kraftwirkung; der Hebel an der Winde bewirkte eine zusätzliche Vervierfachung der Kraft des Bedieners.

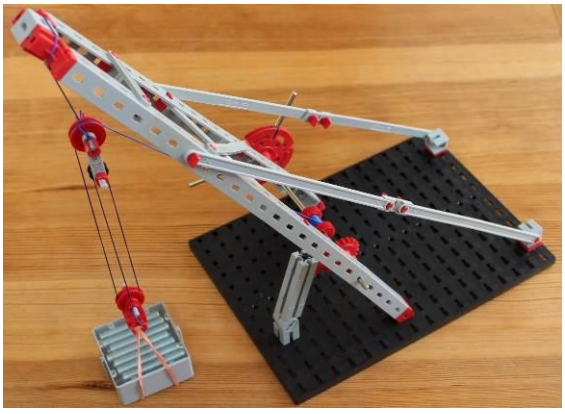


Abb. 3: fischertechnik-Modell eines Trispastos

Die beeindruckende Wirkung eines solchen „Kraftverstärkers“ lässt sich durch einen Nachbau des Trispastos in fischertechnik demonstrieren. Abb. 3 zeigt ein Modell, das etwa denselben Verstärkungsfaktor besitzt. Problemlos kann diese einfache Konstruktion eine mit kurzen Gewindestangen gefüllte Kiste hochziehen. Die Krafteinsparung lässt sich mit weiteren ‚Schlingen’ verstärken: Die für die Hubarbeit benötigte Kraft F_Z sinkt bei n Seilwegen auf ein n-tel der Gewichtskraft der Last F_L:

$$F_Z = \frac{F_L}{n} \qquad (1)$$

Daher liegt es nahe, die von einem Kran oder einer Seilwinde leistbare Hubarbeit zu vergrößern, indem man den Flaschenzug um weitere Rollen ergänzt.

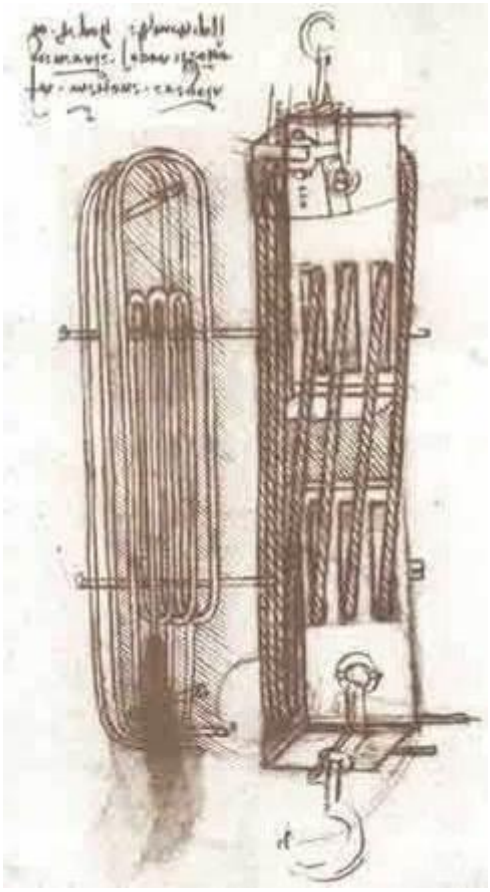


Abb. 4: Flaschenzug von Leonardo Da Vinci

Dafür gibt es im Wesentlichen zwei Ansätze: die Anordnung der Rollen nebenein-

ander (horizontal) und die Anordnung übereinander (vertikal). Mit letzterer lässt sich der Flaschenzug schlanker realisieren; dafür reduziert sich konstruktionsbedingt die maximale Hubhöhe, da ein Teil für die vertikale Anordnung der Rollen benötigt wird. Eine kompakte Konstruktion eines sowohl horizontalen als auch vertikalen Flaschenzugs mit 12 Rollen, bei der die Rollen sowohl nebeneinander als auch übereinander angeordnet sind, ist von dem Universalgenie der Renaissance, *Leonardo Da Vinci* (1452-1519) überliefert (Abb. 4).

Flaschenzüge aus fischertechnik finden sich schon in der Anleitung zum Grundkasten (S. 20) aus dem Jahr 1966 (Abb. 5).

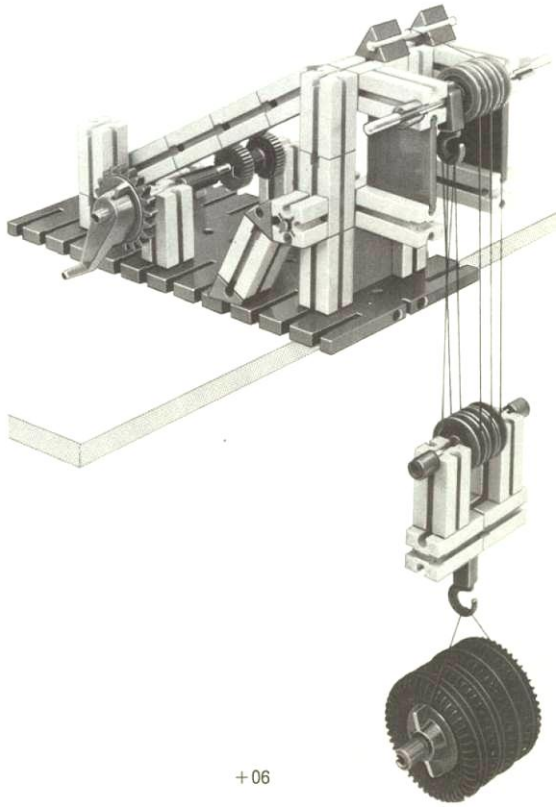


Abb. 5: ft-Flaschenzug von 1966 (aus: Bauanleitung Grundkasten)

Im hobby1 Band 1 wurde dem Flaschenzug 1972 ein eigenes Kapitel gewidmet [2]. Wirken die ersten Flaschenzüge noch etwas plump und eher wie grobe Funktionsmodelle, gelingt unter Verwendung von Statik-Komponenten wie z. B. den S-Laschen (36327) eine deutlich elegantere Konstruktion (linke Variante in Abb. 6) – zu finden z. B. in der Anleitung zum Aufbau-Statikkasten 50S/3 aus dem Jahr 1975. Auch die Kreuzknotenplatten (36322) aus den frühen Statikkästen von 1970 erlauben eine ansprechende Konstruktion wie die zweite Variante von rechts in Abb. 6, zu finden z. B. in [3] (S. 18, 20, 49).

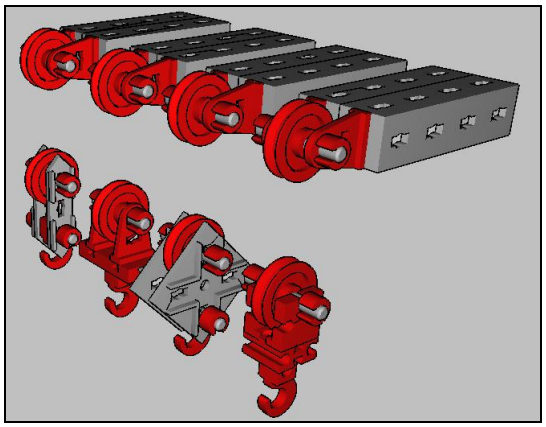


Abb. 6: Konstruktionsvarianten einfacher fischertechnik-Flaschenzüge

Das Rollenlager (37636) aus dem Modellkasten Starlifters von 1990 eignet sich ebenfalls zur Konstruktion eines Flaschenzugs, siehe die rechte Variante in Abb. 6.

Dasselbe gilt für die Kupplungsstücke (38253) aus dem Universal-Baukasten von 1997, verwendet in der zweiten Variante von links in Abb. 6; siehe auch die Bauanleitung des Universal-Baukastens (Abb. 7).

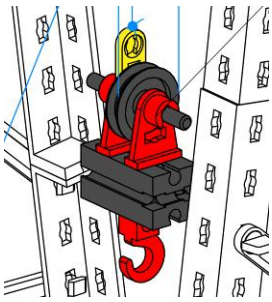


Abb. 7: Flaschenzug (aus: Bauanleitung Universal-Baukasten)

Bei allen Varianten wird das Seilende jeweils mit einer der Klemmbuchsen an der oberen Achse befestigt, über die untere Rolle geführt und von dort über die obere Umlenkrolle zu einer Seilwinde.

Will man mehr als eine Verdoppelung der Kraftwirkung erreichen, benötigt man einen Flaschenzug mit weiteren Führungsrollen. Auch mit fischertechnik lassen sich die zusätzlichen Rollen sowohl vertikal als auch horizontal anordnen. Abb. 8 zeigt vier Realisierungsalternativen für *n*-fache Flaschenzüge.

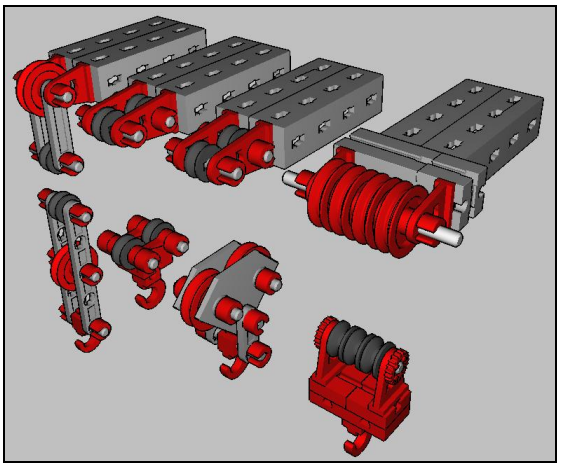


Abb. 8: Realisierungsalternativen für n-fache Flaschenzüge

Die drei linken Varianten sind auf jeweils vier Rollen beschränkt; damit lässt sich die Kraftwirkung vervierfachen. In der Anleitung zum Teleskop-Mobilkran von 1983 (S. 45) findet sich eine vertikale Konstruktion.

Abb. 9: 7-facher Flaschenzug (aus: Abenteuer-Bau-Buch [4])

Ein Flaschenzug mit horizontal angeordneten Rollen wird im hobby2 Band 4 (S. 10) vorgestellt. Er verwendet Statikstreben, um Haken und Rollenachse miteinander zu verbinden [3]. Die rechte Variante in Abb. 8 erreicht eine Verachtfachung der Kraftwirkung; sie kann zudem leicht um weitere Führungsrollen erweitert werden. Sie findet sich z. B. im Abenteuer-Bau-Buch (S. 50 ff.) aus dem Jahr 1985 [4] (Abb. 9).

Potenzflaschenzug

Eine andere Flaschenzugkonstruktion ist der Potenzflaschenzug (Abb. 10). Dabei wird mit jeder zusätzlichen Rolle ein weiteres Zugseil eingeführt und damit die erforderliche Kraft halbiert.

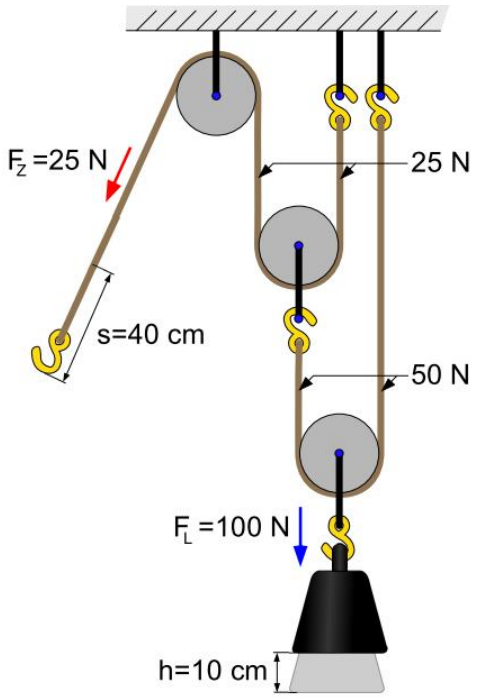


Abb. 10: Potenzflaschenzug (Quelle: Wikipedia [1])

Die für die Hubarbeit erforderliche Zugkraft liegt bei *n* ‚losen' Rollen also bei einem 2^n-tel der Gewichtskraft der Last:

$$F_Z = \frac{F_L}{2^n} \qquad (2)$$

Die Wirkung ist größer als bei einem Faktorenflaschenzug: mit fünf losen Rollen erreicht man eine Verstärkung von 32. Dennoch findet man diesen Flaschenzug-Typ eher selten.

Differenzialflaschenzug

Ein dritter Flaschenzug-Typ ist der Differenzialflaschenzug. Er besteht aus zwei auf einer Achse fest miteinander verbundenen Rollen mit unterschiedlichem Radius und einer dritten, losen Rolle. Die Seilführung erfolgt wie bei einem Faktorenflaschenzug, allerdings werden die Seilenden miteinander verbunden (Abb. 11).

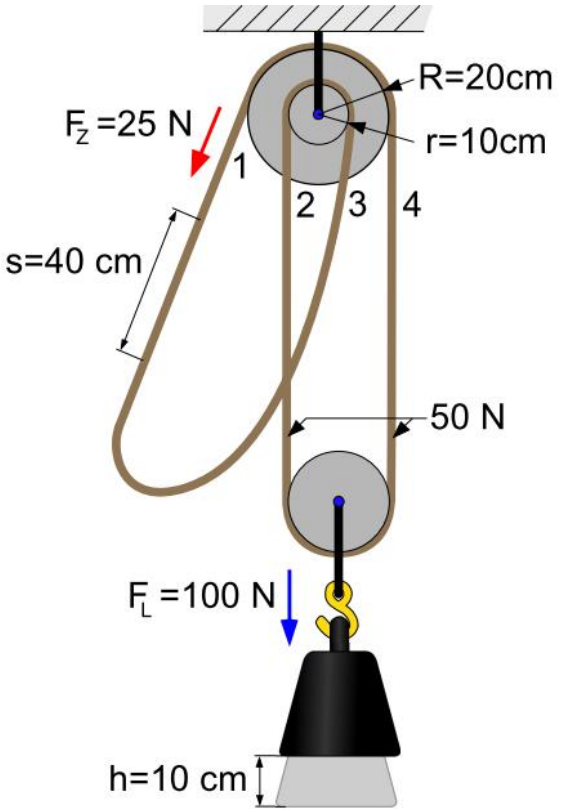


Abb. 11: Differenzialflaschenzug (Quelle: Wikipedia [1])

Die Kraftverstärkung berechnet sich hier aus der Differenz der Radien der beiden Rollen. Denn während über die kleinere Rolle mit Radius r je Umdrehung eine Seillänge von $2\pi \cdot r$ abrollt, verkürzt sich die Seilschlinge zugleich um die an der großen Rolle mit Radius R aufgerollte Seillänge $2\pi \cdot R$. Die erforderliche Zugkraft berechnet sich daher aus der Last mit:

$$F_Z = \frac{F_L \cdot (R - r)}{2 \cdot R} \qquad (3)$$

Ist der Radius R wie in Abb. 11 doppelt so groß wie r, genügt ein Viertel der Kraft für die nötige Hubarbeit. Da der Seilzug auf den Rollen nicht durchrutschen darf, werden meist eine Kette statt des Seils und statt der Rollen Zahnräder verwendet. Ein fischertechnik-Modell eines Differenzialflaschenzugs mit Kette findet sich in hobby2 Band 4 auf S. 19 [3] (Abb. 12).

Abb. 12: Differenzialflaschenzug (aus: hobby2 Band 4 [3])

Nach demselben Prinzip arbeitet die **Differenzialwinde**. Sie kommt allerdings ohne Rollen (bzw. Zahnräder) aus. Stattdessen verwendet sie zwei Winden mit unterschiedlichem Durchmesser, auf denen die beiden Enden des Zugseils in einander entgegen gesetzter Richtung aufgewickelt

sind. Beim Drehen wird das eine Ende des Zugseils auf einer der Winden ab- und das andere auf der anderen Winde aufgewickelt – wegen des unterschiedlichen Durchmessers mit unterschiedlichen Seillängen (Abb. 13).

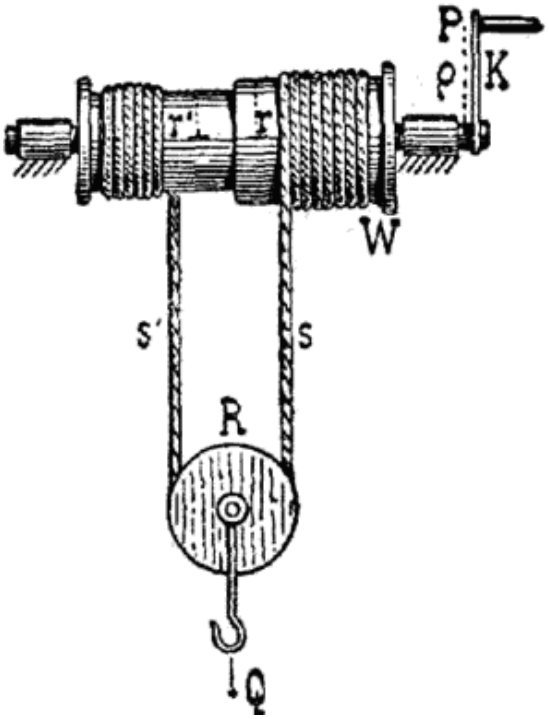


Abb. 13: Prinzip der Differenzialwinde

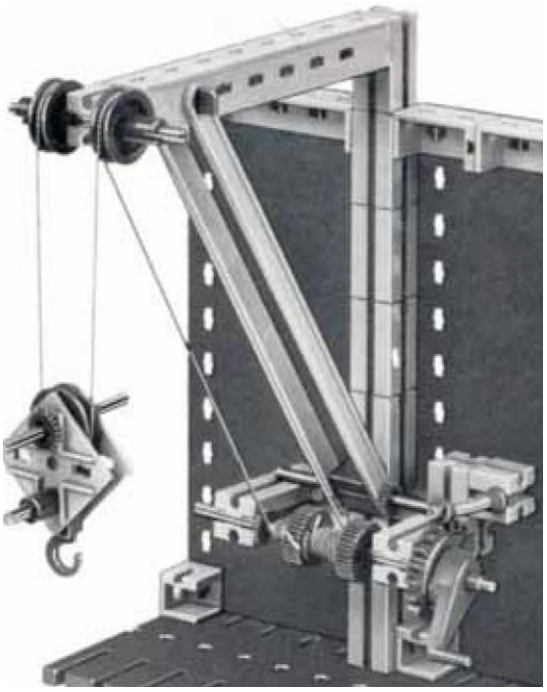


Abb. 14: Differenzialwinde (aus: hobby2 Band 4 [3])

Das Verhältnis der Zugkraft zur Gewichtskraft der Last berechnet sich wie in Formel (3) für den Differenzialflaschenzug. Ein Konstruktionsbeispiel für eine solche Differenzialwinde findet sich z. B. im hobby2 Band 4, S. 18 (Abb. 14).

Das Prinzip der Differenzialwinde wird auch für eine ebenfalls bereits in antiken Kränen verwendeten, heute als **Wellrad** bezeichneten Konstruktion genutzt. Dabei werden zwei Räder mit unterschiedlichem Durchmesser auf derselben Achse montiert. Um das größere Rad wird ein langes Zugseil gewickelt, um das kleinere das die Last haltende Seil bspw. eines Krans. Die Verstärkung der Zugkraft berechnet sich hier unmittelbar aus dem Verhältnis der beiden Winden-Radien zueinander:

$$F_Z = \frac{F_L \cdot r}{R} \qquad (4)$$

Solche Wellräder wurden bereits in der Antike in Kränen mit Flaschenzügen kombiniert (Abb. 15).

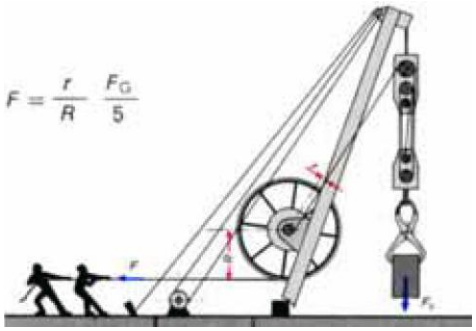


Abb. 15: Historischer Kran mit Wellrad (aus: hobby2 Band 4 [3])

Ersetzt man im Modell des Trispastos die Winde durch ein Speichenrad, wird das Funktionsprinzip anschaulich (Abb. 16).

In römischen (und später mittelalterlichen) Kränen wurde statt eines Speichenrads ein Tretrad verwendet, in dem Sklaven oder Arbeiter die Hubarbeit nicht mehr nur mit Armkraft, sondern durch Einsatz ihres gesamten Körpergewichts verrichteten.

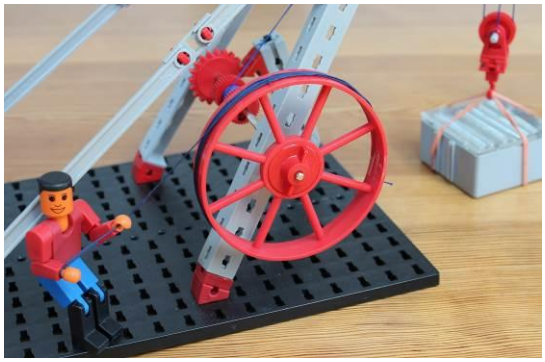


Abb. 16: Trispastos mit Wellrad

Von Kränen mit Tretrad gibt es zahlreiche Nachbauten, z. B. im Limesmuseum bei Aalen (Abb. 17) oder im Technoseum in Mannheim.

Abb. 17: Römischer Baukran mit Tretrad (Limesmuseum bei Aalen)

Ein Modell mit Tretrad findet sich auch im hobby2 Band 4, S. 10 [3] (Abb. 18).

Abb. 18: Historischer Kran mit Laufrad (aus: hobby2 Band 4 [3])

Fazit

Flaschenzüge, Differentialwinden und Wellräder sind einfache mechanische Maschinen, mit denen eine Zugkraft verstärkt oder auch abgeschwächt werden kann. Bei Motorantrieben sind sie eine Alternative zu Über- oder Untersetzungen durch Getriebe.

Flaschenzüge haben zudem einen positiven Nebeneffekt: Sie stabilisieren das Zugseil, indem sie Verdrillungen erschweren: Ein Objekt lässt sich damit gerade nach oben ziehen. Je mehr Seilstränge, desto widerstandsfähiger ist ein Flaschenzug gegen Torsion.

Beliebig große Gewichte lassen sich mit einem Flaschenzug allerdings nicht heben – die Hubarbeit geht irgendwann nicht mehr spurlos am Material vorüber. Spätestens, wenn am Antrieb (Schnecke, Zahnrad) Abrieb entsteht, sollte man das Gewicht reduzieren.

Das Zugseil hingegen wird entlastet, da auf den einzelnen Seilstrang nur ein Bruchteil der Gewichtskraft des zu hebenden Gegenstands wirkt. So kann man mit einem Flaschenzug auch sehr schwere Gegenstände mit einem dünnen Seil anheben.

Quellen

[1] Wikipedia: *Flaschenzug*.

[2] Artur Fischer: *Der Flaschenzug*. In: fischertechnik hobby, Experimente und Modelle, hobby1 Band 1, Fischer-Werke, 1972, S. 40-43.

[3] Artur Fischer: *Krane*. fischertechnik hobby, Experimente und Modelle, hobby2 Band 4, Fischer-Werke, 1975.

[4] Artur Fischer: *Das Abenteuer-Bau-Buch*. Fischer-Werke, 1985.

[5] Heribert Keh: *Der Flaschenzug*. Unterrichtshilfe Technik (u-t). Fischer-Werke, 1980.

Tipps & Tricks

Abluftdrosselung mit dem Pneumatik-Handventil

Stefan Falk

„Richtiges Drosseln ist Abluft-Drosseln“, wurde in der „Druckluftsteuerungen“-Artikelserie in der ft:pedia beschrieben. Mit dem aktuellen Pneumatik-Handventil erscheint das leichter gesagt als getan. Deshalb gibt es hier einen einfachen Tipp, mit dem die Abluft dieses Drehschieberventils doch noch gedrosselt werden kann.

In [1] wird beschrieben, wie und warum man für ein einstellbar langsames Verfahren eines Pneumatikzylinders dessen Abluft und nicht etwa seine Zuluft drosselt: Nur so erreicht man eine möglichst ruckelfreie, gleichmäßige Bewegung des Zylinders.

Nun verfügt das aktuelle Pneumatik-Handventil über keinen Schlauch-gerechten Anschluss für seinen Abluftausgang. Die typischen Bagger-Modelle der aktuellen Pneumatik-Kästen lassen ihre Baggerarme deshalb unschön und völlig unrealistisch mehr auf und ab schlagen, anstatt eine majestätisch langsame Bewegung wie bei einem echten Bagger mit Hydraulikzylindern darzustellen. Zudem gibt es die kompakten Pneumatik-Drosseln aus der fischertechnik Ur-Pneumatik auch nicht mehr im aktuellen Programm.

Wie die Abbildung zeigt, gibt es aber doch einen Weg: Ein Pneumatik-T-Stück passt nämlich sehr wohl in die Abluftöffnung des Handventils. Wenn man es ganz kompakt mag, steckt man darauf einfach direkt zwei P-Stopfen. Der Trick ist nun, einen der Stopfen nicht ganz fest aufzustecken. Die Abluft soll ja immer noch ins Freie strömen können, nur eben gedrosselt. Auf diese Weise erreicht man mit wenig Platz- und Materialbedarf eine wunderbar langsame, gleichförmige Bewegung des Zylinders. Man kann zwar für die beiden Bewegungsrichtungen nicht verschieden stark drosseln, aber eine Verbesserung stellt das allemal dar.

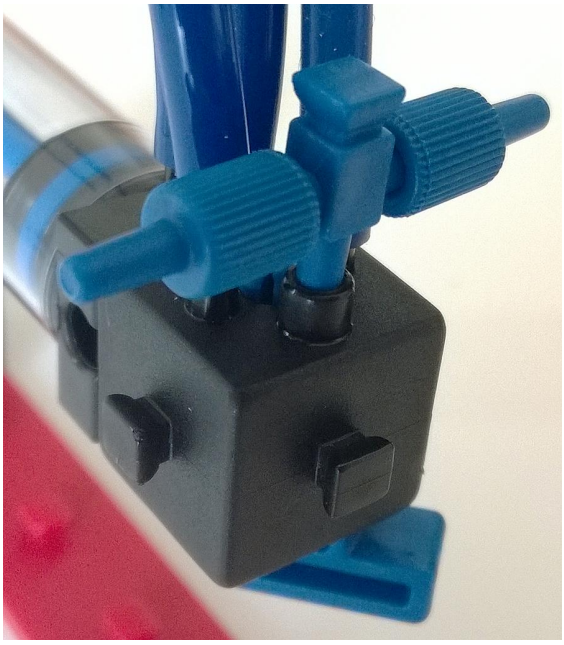


Abb. 1: T-Stück mit zwei P-Stopfen auf dem Abluftausgang des Handventils

Quelle

[1] Falk, Stefan: *Druckluftsteuerungen (Teil 1)*, ft:pedia 1/2014, S. 58-72

Tipps & Tricks

ft-Spezialteile made by TST (Teil 8)

Andreas Tacke

In einer lockeren Reihe stellt TST einige von ihm entwickelte Spezialteile vor, die so manche Lücke beim Bauen mit fischertechnik schließen. Im heutigen Beitrag geht es um Fahrzeugtechnik, genauer gesagt um eine Modifikation des Differentialgetriebes mit stabilen Metallachsen.

Wer kennt sie noch, die guten alten Differentialgetriebe aus den Anfängen von fischertechnik, speziell die Getriebe 31500 (Abb. 1) aus den 80er Jahren [1] sowie das 31043 (Abb. 2, auch in rot [2]) aus den Anfängen von fischertechnik? Beide hatten Metallachsen, die beim Fahrzeugbau für eine stabile Achskonstruktion sorgten.

Abb. 1: Differentialgetriebe 31500

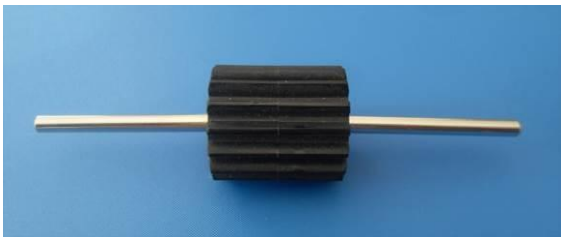


Abb. 2: Differentialgetriebe 31043

Bei den heutigen Differentialgetrieben kommen die Rastachsen aus Kunststoff zum Einsatz, die es ja in verschiedenen Längen gibt. Das bringt natürlich beim Bauen Flexibilität, und alle Achslängen liegen im ft-Raster. Das ist ein klarer Vorteil gegenüber den alten Getrieben, denn dort sind die Längen fix.

Nachdem ich ein größeres Modell mit Kunststoffachsen gebaut hatte, musste ich allerdings feststellen, dass die Achsen aus Kunststoff die Last nicht tragen konnten. Also was tun?

Schön wäre es, bei den neuen Getrieben auch Metallachsen einsetzen zu können. Da bleibt nur: Selbermachen …

Dazu nahm ich mir Rundstahl aus V2A mit 4 mm Durchmesser. An einem Ende fräste ich zwei Flächen an, genau wie bei den Rastachsen. Anschließend wurde in dem Bereich der Flächen noch eine Rändelung aufgebracht, damit die Zahnräder 31413 festen Halt auf der Achse bekommen. Gefertigt habe ich mir diese Achsen in 45, 60, 75 und 90 mm Länge; das reicht eigentlich für den Fahrzeugbau aus (siehe Abb. 3).

Abb. 3: Achsen aus V2A mit Zahnrad 31413

So erhielt ich ein Differentialgetriebe mit Metallachsen, welches auch höhere Achs-

lasten zulässt, ähnlich wie die Getriebe 31500 aus den 80er Jahren (Abb. 4).

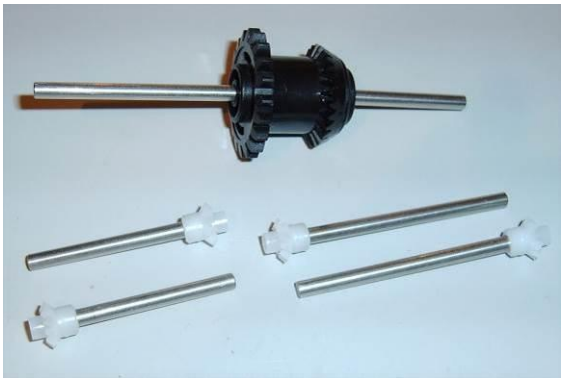


Abb. 4: Differentialkäfig mit Metallachsen

Bei dieser Variante muss man allerdings das Getriebe vorher mit den Achsen zusammenbauen, da sich die Achsen nicht mehr nachträglich von außen einschieben lassen (Abb. 5).

Abb. 5: Zusammenbau

Die Vorteile überwiegen allerdings, wie ich meine. Auch die Realisierung einer Zwillingsbereifung ist damit in stabiler Weise möglich (Abb. 6).

Abb. 6: Aufbau mit Zwillingsbereifung, links mit 45 mm- und rechts mit 60 mm-Achse

Und wieder mal zeigt sich, wie aus einem aktuellen Konstruktionsproblem und ein paar kreativen Gedanken ein Spezialteil entsteht, das perfekt zu fischertechnik passt…

Referenzen

[1] Andreas Tacke: *ft-Spezialteile made by TST (Teil 1)*. ft:pedia 2/2012, S. 24-25.

[2] Stefan Falk: *Perlentauchen (Teil 2)*. ft:pedia 4/2012, S. 14-21.

Optik

Einstieg in Experimente mit Lasern

Andreas Gail

Laser gewinnen im Bereich der Technik, aber auch in der Medizin, immer weiter an Bedeutung. Längst sind sie nicht mehr Science-Fiction, sondern im täglichen Leben angekommen. Angefangen bei der Scanner-Kasse im Supermarkt, dem CD Spieler zu Hause oder auch als Entfernungsmesser für den Heimwerker. Höchste Zeit also, diese noch immer etwas futuristisch anmutende Technik etwas näher zu betrachten. Beginnen wollen wir mit dem Selbstbau einer Lasereinheit; anschließend folgen zwei Anwendungsbeispiele.

Sicherheit

Die verfügbaren Laser werden in Gefährlichkeitsklassen von 1 bis 4 eingeteilt, dabei sind die Laser der Laserklasse 4 die gefährlichsten. Daher werden Experimente nur mit Lasern der Klasse 1 empfohlen.

Trotzdem sollte niemals in das Laserlicht direkt oder über Spiegel hineingeblickt werden. Das gilt besonders bei Verwendung von Linsen. Sicherheitshinweise der Hersteller sind zu beachten.

Punkt und Linie

Für die später folgenden Experimente werden zwei verschiedene Laser eingesetzt. Dieses sind einmal Laser mit einer punktförmigen Lichtaussendung und einmal mit einer linienförmigen. Hierbei werden Laser für einen Gleichspannungsbereich von 3 bis 12 V mit einer Leistung von 2 bis 5 mW verwendet (Conrad #816296 und #816304). Hinsichtlich der spezifizierten Betriebsspannung erscheint alles bestens geeignet, um direkt in die Experimente einzusteigen.

Doch zwei Dinge sollten unbedingt bei der Verwendung berücksichtigt werden, um die Laser nicht gleich zu zerstören. Zum einen muss unbedingt auf die richtige Polung geachtet werden. Weiterhin sind die Laser empfindlich hinsichtlich elektrostatischer Aufladung.

Abb.1: Warnhinweis zum Schutz vor elektrostatischer Aufladung

Gehäuse und Schutzschaltung

Die beiden oben genannten Probleme soll die folgende Schutzschaltung möglichst weitgehend lösen (Abb. 2).

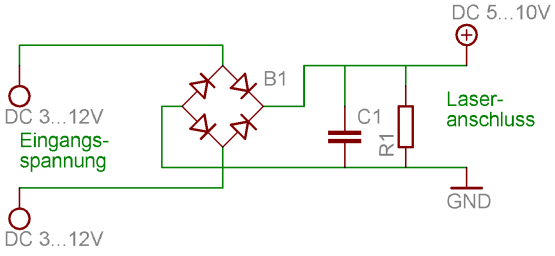


Abb. 2: Schutzschaltung für Laser

Die ggf. falsche Polung wird hierbei über einen Brückengleichrichter kompensiert. Den Schutz vor der elektrostatischen Aufladung übernehmen ein Kondensator und ein Widerstand. Kondensatoren haben

dabei die Eigenschaft, kurzzeitig auftretende Spannungsdifferenzen auszugleichen. Betrachtet man den fertigen Aufbau, ergibt sich folgendes:

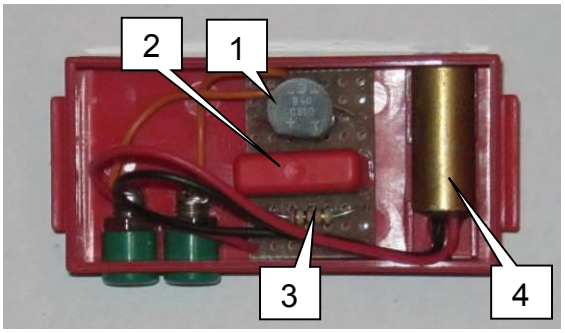


Abb. 3: Laser und Schutzschaltung, eingebaut in ein ft-Batteriekasten mit Brückengleichrichter (1), Kondensator 0,1 µF (2), Widerstand 10 kΩ (3) und Laser (4)

Der vollständige Aufbau wurde mit dem Gefahrenhinweis des Laserherstellers versehen (Abb. 4).

Abb. 4: Vollständiger Aufbau der Lasereinheit

Linienlaser

Fischertechnik hat uns in der Vergangenheit in den Optik-Kästen einige Linsen bereitgestellt, wie nachfolgend gezeigt. Diese sind allerdings nicht beschriftet, und so läßt sich nicht erkennen, welche Brennweite sie haben. Mit der im Folgenden beschriebenen Methode können die Brennweiten experimentell bestimmt werden.

Abb. 5: Fischertechnik Linsen unterschiedlicher Brennweite (siehe auch [1])

Zur Messung der Brennweite mithilfe des zuvor aufgebauten Linienlasers und des nachfolgend gezeigten Versuchsaufbaus kann die Messung beginnen.

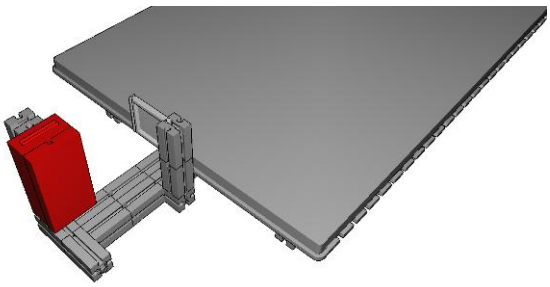


Abb. 6: Messanordnung (von oben)

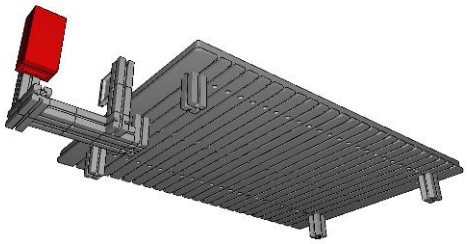


Abb. 7: Messanordnung (von unten)

Abb. 8: Linienlasereinheit mit Spannungsquelle und zu vermessender Linse

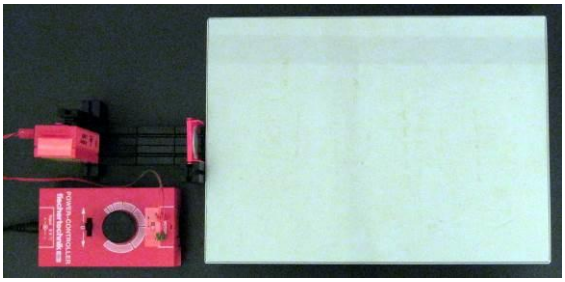


Abb. 9: Draufsicht der Messanordnung

Wird die Lasereinheit parallel zur Linsenfläche verschoben ergeben sich für drei aufeinander folgende Einstellungen die Bilder in Abb. 10. Da bei dem Versuchsaufbau mit einem vertikal ausgerichteten Linienlaser gearbeitet wird, wird auf der ebenen Bauplatte ein sichtbarer linienförmiger Laserverlauf möglich. Es bietet sich an, auf der ebenen Bauplattenseite ein Blatt Papier mit Klebestreifen zu befestigen. Hierbei sollte das Papier möglichst frei von Wellen aufgeklebt werden, um später eine durchgängige Linie zu erhalten.

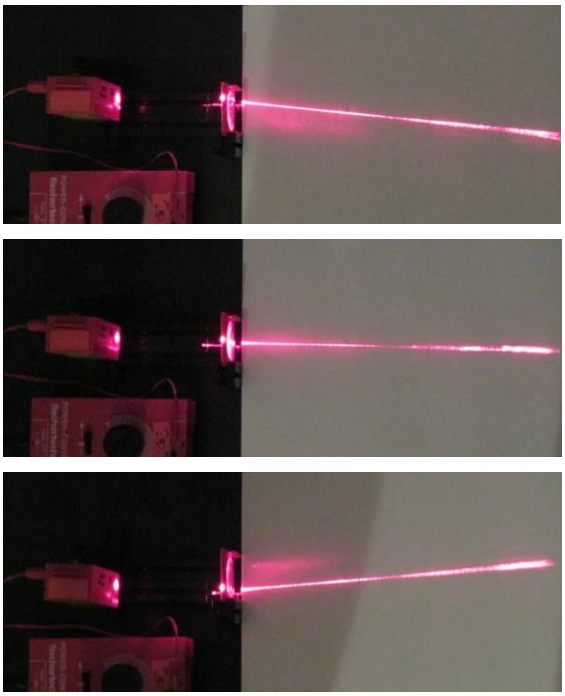


Abb. 10: Vermessung einer Linse mit drei Einstellungen zur Bestimmung der Brennweite

Wird bei jeder dieser drei Einstellungen mit einem Bleistift die jeweilige Laserlinie nachgezogen zeigt sich, dass alle Bleistiftlinien durch einen Punkt verlaufen: den Brennpunkt der Linse. Der Abstand von der Linse zum Brennpunkt wird Brennweite genannt.

Es drängt sich zum Schluss die Frage auf, was man mit der gewonnen Erkenntnis anfangen kann. Eine Antwort könnte sein, auf diese Weise einen Fotowiderstand zielgerichteter einsetzen zu können, indem vor den Fotowiderstand eine oben gezeigte Linse im richtigen Abstand vorgeschaltet wird.

Punktlaser

Eine einfache klassische Anwendung für einen Punktlaser stellt eine Lichtschranke dar. Bei einem Laser lassen sich jedoch deutlich weitere Strecken überbrücken. Mit dem gezeigten Prinzipaufbau konnten mühelos mehrere Meter überbrückt werden.

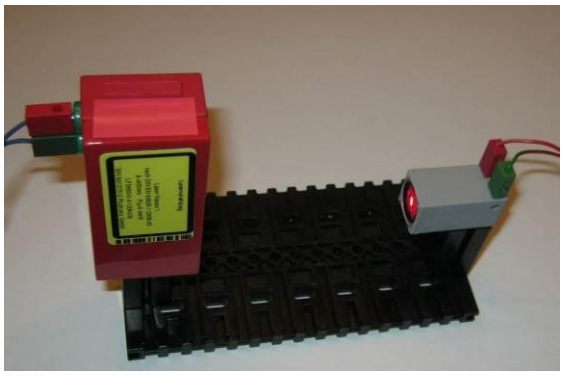


Abb. 11: Aufbau einer Laserlichtschranke (gezeigt sind nur wenige Zentimeter; mehrere Meter sind jedoch mühelos erreichbar, gegebenenfalls unter Nutzung einer Störlichtkappe)

Weitere Anwendungen

Mit einem solchen einfachen Laser lassen sich zahlreiche weitere Anwendungen realisieren. Prinzipiell vorstellbar ist der Aufbau einer groben Entfernungsabschätzung; über ein Feld von Sensoren ließe sich der Geradeauslauf eines ansonsten frei beweglichen Fahrzeugs regeln. Man darf gespannt sein, was sich die Leserschaft alles einfallen lassen wird …

Hinweis zur Gehäusefertigung

Das oben gezeigte Batteriegehäuse zeigt sich sehr widerspenstig bei der Bearbeitung. Das Hineinbohren von Löchern ist nur schwer und unschön möglich. Eine einfache und schnelle Methode ist die Verwendung eines Locheisens (Abb. 12).

Einen Holzklotz auf der Innenseite des Gehäuses plan unterlegen, ein gezielter kräftiger Schlag – und das Loch ist ausgestanzt.

Abb. 12: Locheisen zur Einbringung von Löchern in ein Batteriegehäuse

Referenzen

[1] Stefan Falk: *Perlentauchen (Teil 4)*. ft:pedia 2/2013, S. 18-30.

Modell

Mini-Modelle (Teil 2): Panzer

Johann Fox

In der ft:pedia 4/2013 wurde von René Trapp als erstes Mini-Modell im GiveAway-Format ein Gabelstapler vorgestellt. Als nächstes GiveAway folgt hier ein Minipanzer.

Im Bilderpool der ft-community gibt es ganze Sammlungen von Panzern. Sie variieren in Größe, Bauweise und technischen Details. Aber es gab noch nie einen Panzer, der nur aus 13 Bauteilen besteht – bis jetzt. In Abbildung 1 ist er als Gesamtansicht abgebildet.

Abb. 1: Der Minipanzer

Es gibt den Minipanzer in der Ausführung mit Rädern und mit „Ketten“, die hier durch Gummi-Bänder ersetzt wurden (siehe Abb. 2).

Abb. 2: Modellvariante mit „Ketten“

Die Konstruktion ist sehr simpel: Im Wesentlichen besteht der Minipanzer aus der drehbar gelagerten Kanone, den Hinterrädern, dem Verbindungsstück und den drehbar gelagerten Vorderrädern (Abb. 3). Diese sind meist durch einen S-Riegel blockiert.

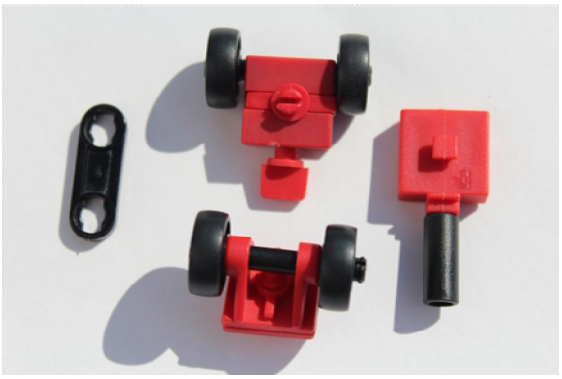


Abb. 3: Minipanzer Konstruktion

In Abb. 4 sieht man alle 13 Einzelteile nebeneinander liegen.

Abb. 4: Einzelteile

Zum Schluss kann man den Minipanzer nochmal von unten betrachten (Abb. 5).

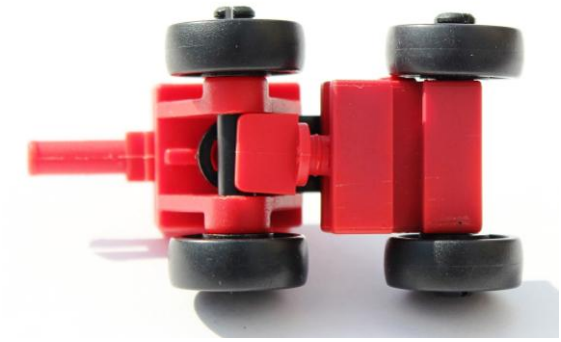


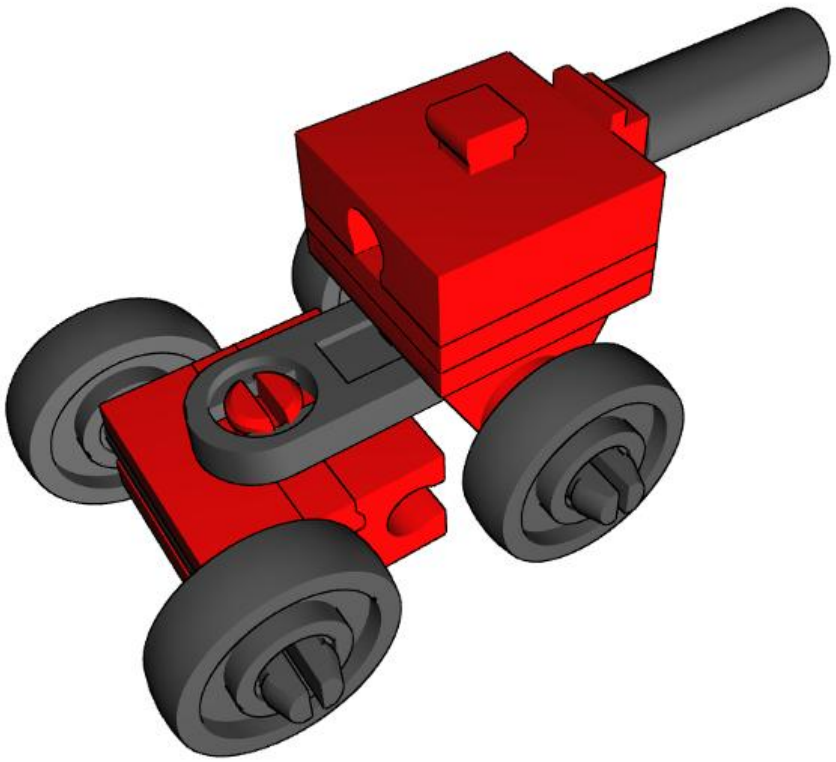
Abb. 5: Minipanzer von unten

Wer nun zum Nachbauen angeregt ist, dem darf natürlich die Einzelteilliste nicht fehlen:

Stück	ft-Nr.	Bezeichnung
1	31771	Lagerstück 1 rot
1	31772	Lagerstück 2 rot
1	31848	Strebenadapter rot
1	36819	Lagerhülse schwarz
1	36914	I-Strebe 15 schwarz
4	36573	Rad 14 schwarz
2	36919	V-Achse 4*28
1	36586	Radachse rot
1	37468	Baustein 7,5 rot
1	36323	S-Riegel 4 mm rot

Referenzen

[1] Trapp, René: *Mini-Modelle (Teil 1): Gabelstabler*. ft:pedia 4/2013, S. 4-5



Pneumatik

Druckluftsteuerungen (Teil 2)

Stefan Falk

In der letzten Ausgabe haben wir die „Schlauch-Logik“ eingeführt, um auch mit aktuell produzierten Teilen von fischertechnik pneumatische Steuerungen herstellen zu können. Diese Reise setzen wir fort, um weitere mit den Ur-Pneumatik-Teilen machbare Steuerschaltungen auch mit heute noch hergestellten Teilen zu realisieren.

Der Trick, einen Schlauch durch simples Abknicken zu einem 2/2-Wege-Ventil (zwei Anschlüsse, zwei Schaltstellungen) umzubauen, hat uns das „pneumatische Relais“, ein 4/2-Wegeventil z. B. zur Umschaltung eines Zylinders durch eine einzige Steuerleitung beschert (siehe ft:pedia 2014-1 [1]):

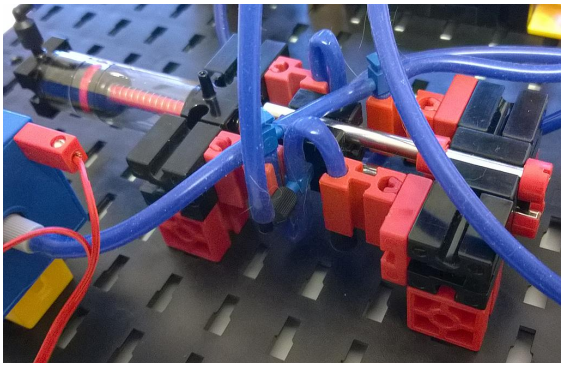


Abb. 1: 4/2-Wegeventil aus ft:pedia 2014-1

Dieses Ventil hat einige angenehme Eigenschaften:

- Das wichtigste ist natürlich, dass man mit aktuell hergestellten und lieferbaren Pneumatik-Teilen auskommt.
- Die Schaltzeit ist – dafür, dass wir ja keine Betätiger und Festo-Ventile [2] mehr zur Verfügung haben, sondern nur Zylinder – brauchbar kurz, weil der bewegliche Teil nur um wenige Millimeter verschoben werden muss.
- Anders als bei den Festo-Ventilen können wir alle vier Anschlüsse des 4/2-Wege-Ventils beliebig verwenden. Bei den Festo-Ventilen ist der Abluftausgang kaum für etwas anderes brauchbar. Hier hingegen könnten wir direkt an die Abluftausgänge eine Drossel setzen, um ein langsames, gleichmäßiges Verfahren von Zylindern zu erreichen.

Allerdings gibt es bei einem solchen, aus der Not geborenen Aufbau, natürlich auch Nachteile:

- Das Ventil baut viel größer als die nur zwei oder drei BS15 große Kombination aus Betätigern und Festo-Ventilen.
- Das Selbstbauventil benötigt durch das größere zu beaufschlagende Volumen des Zylinders mehr Druckluft für den Schaltvorgang als ein Betätiger.
- Es ist schwergängiger als die Kombination Betätiger/Festo-Ventil.
- Das Selbstbauventil springt nicht um, sondern wird langsam „umgedrückt“. Es gibt oft einen Zwischenzustand, in dem Druckluft unerwünscht ins Freie entweicht, weil beide Schläuche eines „Umschaltkontakts“ halb offen sind.

Dieses letzten Nachteils wollen wir uns in dieser Ausgabe annehmen. Der bewirkt nämlich auch, dass wir die Druckluft zum Ansteuern des Ventils immer schlagartig zuführen oder ablassen müssen. Wir können also keine Drossel in die Steuerzuleitung einfügen, um das Ventil mit einer Zeitverzögerung schalten zu lassen. Es würde viel zu langsam umschalten, und der dabei in der Zwischenstellung entstehende Druckverlust kann den Stillstand eines ganzen Modells verursachen.

Springkontakt

Wir brauchen also auch bei langsamer Betätigung ein schlagartiges Umschalten des Ventils. Wer einen der älteren Betätiger besitzt, kann selber beobachten, dass der darin enthaltene Gummibalg tatsächlich schlagartig ein oder aus fährt. Mangels Gummibalg müssen wir uns aber eine andere Möglichkeit suchen.

Betrachten wir dazu die Mechanik im Inneren der elektrischen fischertechnik-Taster. Die ist im aktuellen Minitaster identisch mit dem Ur-Taster, aber in letzterem sieht man auch von außen sehr schön die Funktionsweise [3]:

Abb. 2: Taster unbetätigt

Wie die Abbildungen zeigen, wird der zentrale Umschaltkontakt über ein kleines, gebogenes, federndes Element mit der äußeren Betätigung verbunden. Sobald der Taster weit genug eingedrückt wird, schnappt der Kontakt schlagartig um; beim wieder Loslassen genauso. So ist der Taster immer in einem definierten Zustand, und der eigentliche Umschaltvorgang geht sehr schnell, auch wenn der Taster noch so langsam gedrückt oder losgelassen wird.

Ähnlich können wir auch unser selbstgebautes Pneumatikventil schlagartig umschalten lassen, wie wir gleich sehen werden. Das wiederum ermöglicht es uns, Zeitschaltungen unter Verwendung von Drosseln herzustellen, die einen Zylinder nur ganz langsam verfahren und dennoch ab einem Grenzpunkt ein 4/2-Wege-Ventil augenblicklich umschalten lassen.

Abb. 3: Taster betätigt

Der pneumatische Wimpel

Passend zur gerade laufenden Fußball-Weltmeisterschaft wollen wir den in Abb. 4 und 5 zu sehenden rein pneumatisch gesteuerten Wimpel-Winker bauen. Vielleicht wollt ihr damit Eure Mannschaft (und nach der WM vielleicht euren eigenen Verein) bewerben. ☺

Natürlich kommt es dabei auf die inneren Werte an, auf die Technik dahinter. Das Modell funktioniert so:

1. Der an einem drehbar gelagerten Arm aufgehängte Wimpel wird hochgehoben – und zwar zackig!
2. Einige Sekunden Wartezeit sollen vergehen.
3. Der Wimpel wird herunter gelassen.
4. Wiederum verstreicht etwas Zeit.
5. Dieses Spiel wiederholt sich endlos.

All das funktioniert rein pneumatisch gesteuert. Das einzige elektrische Bauteil der Maschine ist der Kompressor.

Abb. 4: Der Wimpel ist hochgezogen

Überblick

Abb. 6 zeigt – von der Rückseite aus gesehen – einen Überblick über die auf der Bauplatte 500 montierten Technik:

- Am oberen Bildrand sieht man zwei hintereinander angebrachte Pneumatikzylinder 60. Der linke verfährt den rechten, und der rechte verschiebt über einen BS15 mit Bohrung eine Kette, an der letztlich der Hebel für den Wimpel hängt. Durch die beiden so miteinander verbundenen Zylinder bekommen wir mehr nützlichen zurückgelegten Weg für die Kette. Zur Führung genügen vier direkt auf die Bauplatte 500 aufgesetzte Verkleidungsplatten 15 · 75 (achtet darauf, etwas Spiel zu lassen, damit die Verschiebung leichtgängig funktioniert).

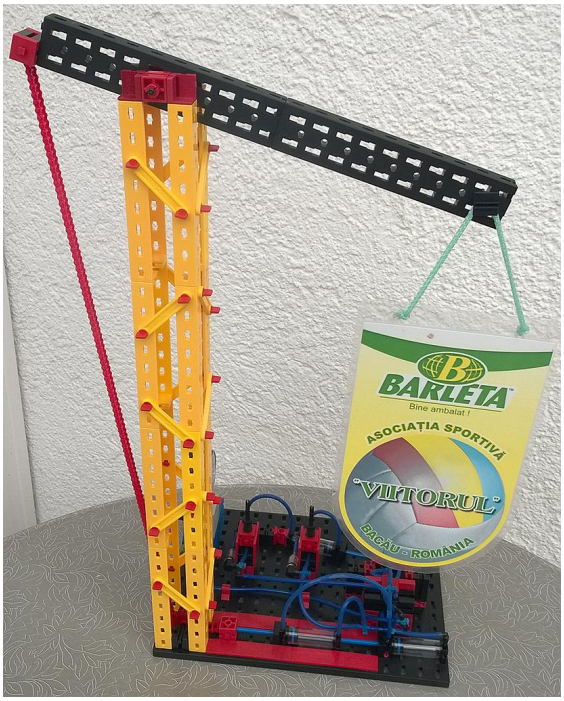


Abb. 5: Der Wimpel ist herunter gelassen

- Diese beiden Zylinder sind parallel an einem 4/2-Wege-Ventil angeschlossen. Das ist mit einer einfachen Kipphebelkonstruktion realisiert. Es funktioniert wieder durch abwechselndes Abknicken bzw. Offenlassen von Schläuchen [1].
- Das Ventil wird nun aber – anders als in Abb. 1 – nicht direkt von einem steuernden Pneumatikzylinder betätigt. Einen Steuerzylinder gibt es zwar (unten im Bild), aber der verdreht nur einen Hebel. Auf diesem Hebel sitzt

eine Federmechanik, und erst die bewegt Kipphebel des Ventils. Die Mechanik ist so gestaltet, dass sie den Kipphebel des Ventils immer vom Zylinder-Hebel weg drückt.

- Für jede Seite des Steuerzylinders existiert je eine Kombination aus Drossel und parallel geschaltetem Rückschlagventil. Die bewirken unabhängig voneinander einstellbare Wartezeiten vor dem Hochziehen bzw. Ablassen des Wimpels, indem sie die Abluft der gerade nicht mit Druckluft beaufschlagten Zylinderseite nur gedrosselt heraus lassen (das ist in [1] ausführlich beschrieben). Bei Teilemangel funktioniert das Modell auch nur mit einer oder sogar ganz ohne Drosseln, nur gibt es dann eben nicht die einstellbaren Wartezeiten.
- Der Steuerzylinder hängt – nur eben über die Drosselstrecken – ebenfalls parallel zu den Ketten-Zylindern geschaltet am 4/2-Wege-Ventil. Und zwar so, dass er das Ventil immer in die Stellung schieben will, die es gerade nicht hat. Daraus ergibt sich ein endloser Wechsel der Ventil- und Zylinderstellungen – der Wimpel wechselt ständig zwischen der oberen und unteren Position. Wir haben einen pneumatischen „Oszillator“ vor uns [4].

Abb. 8 zeigt dasselbe Motiv, nur mit der anderen Ventil- und Zylinderstellung.

Das Schaltbild in Abb. 7 zeigt die Wirkungsweise der Pneumatik-Schaltung. Dargestellt ist die Stellung mit eingefahrenen Zylindern, also angehobenem Wimpel. Die Druckluft des Kompressors geht durch das 4/2-Wege-Ventil zu allen drei Zylindern. Die beiden Ketten-Zylinder werden eingefahren, und der Steuerzylinder will ausfahren. Das parallel zur rechten Drossel geschaltete Rückschlagventil lässt die Luft ungehindert in den Steuerzylinder einströmen. Die Abluft der linken Zylinderseite muss jedoch durch die linke Drossel, weil das dortige Rückschlagventil in Abluftrichtung blockiert. Mit der linken Drossel kann man also ein-

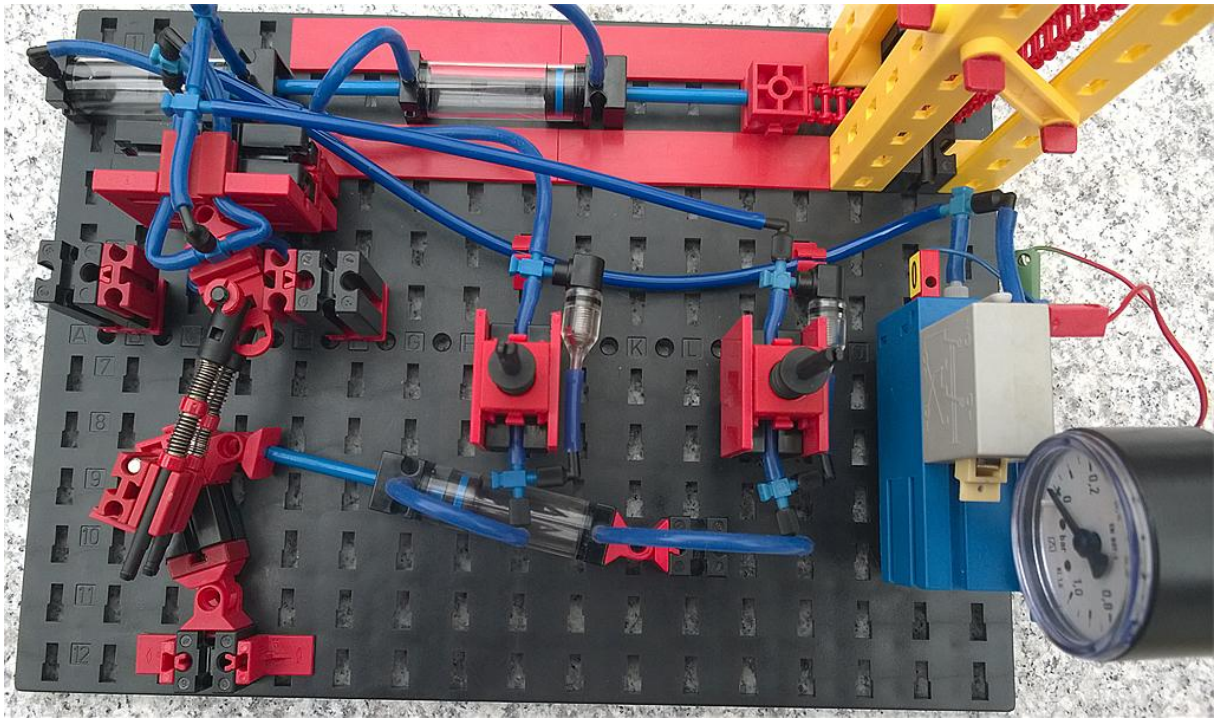


Abb. 6: Überblick über die Technik – alle Kolben sind ausgefahren, der Wimpel ist unten

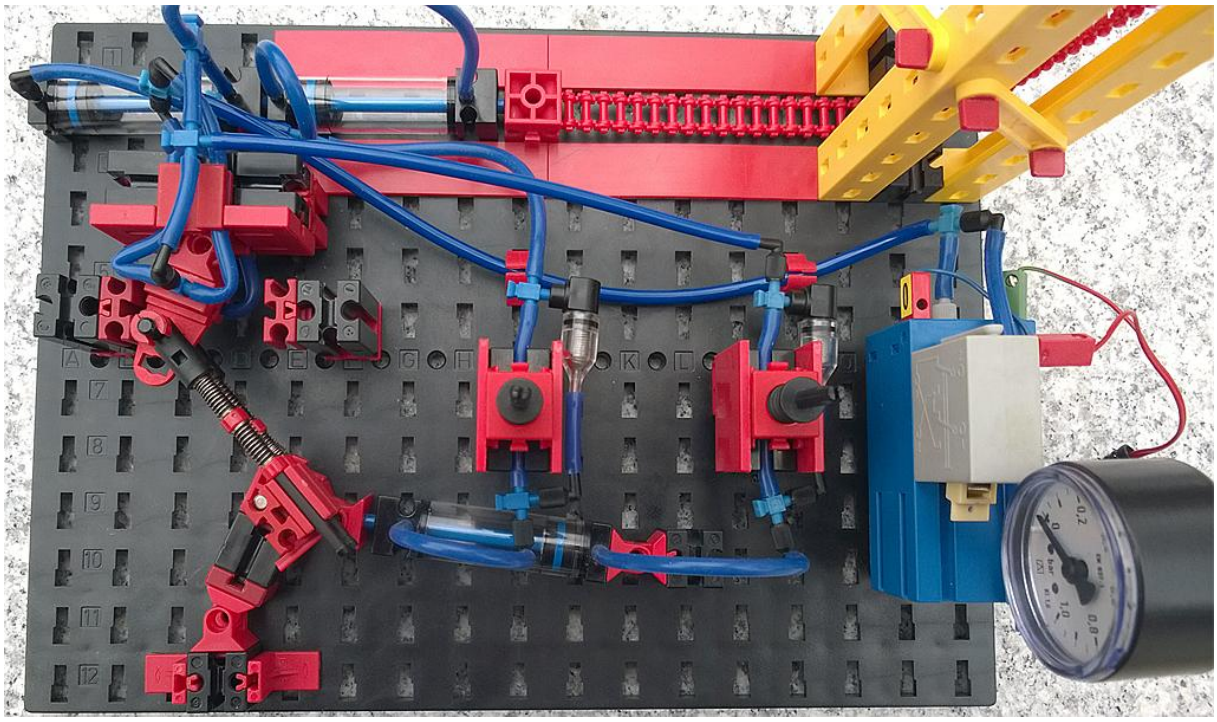


Abb. 8: Überblick über die Technik – alle Kolben sind eingefahren, der Wimpel ist oben

stellen, wie schnell der Steuerzylinder ausfährt und somit, wie lange die Maschine wartet, bevor sie den Wimpel absenkt.

Sobald das Wegeventil umgeschaltet ist (durch die Federmechanik geschieht das schlagartig bei einer bestimmten Zylinderstellung), werden die beiden Ketten-Zylinder ausfahren, Kette nachlassen und so den Wimpel absenken. Gleichzeitig strömt Druckluft in die linke Seite des Steuerzylinders; die in der rechten noch befindliche kann aber nur gedämpft durch die rechte Drossel entweichen (deren Rückschlagventil blockiert ebenfalls in Abluftrichtung). Mit der rechten Drossel lässt sich also einstellen, wie lange der Wimpel unten bleibt, bevor er wieder angehoben wird.

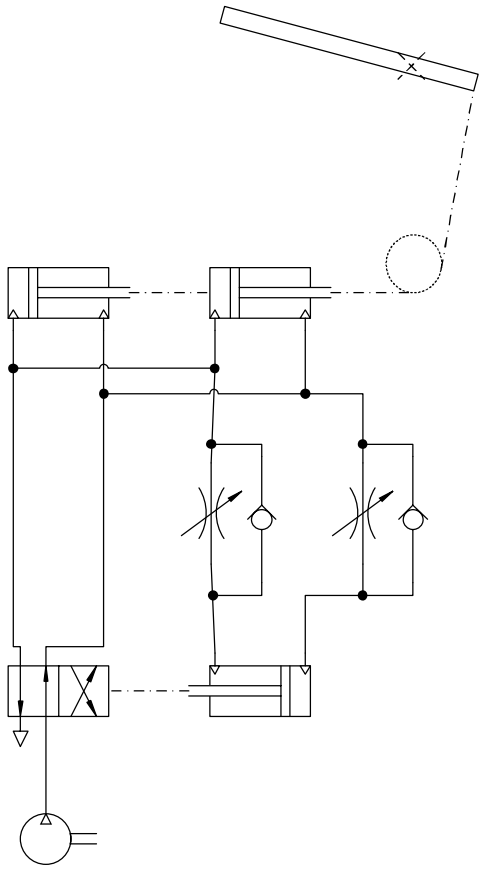


Abb. 7: Schaltbild

Zum Bau des Modells

Fangen wir mit den einfachen Dingen an: Die Kette läuft unten einfach um ein Umlenk-Zahnrad:

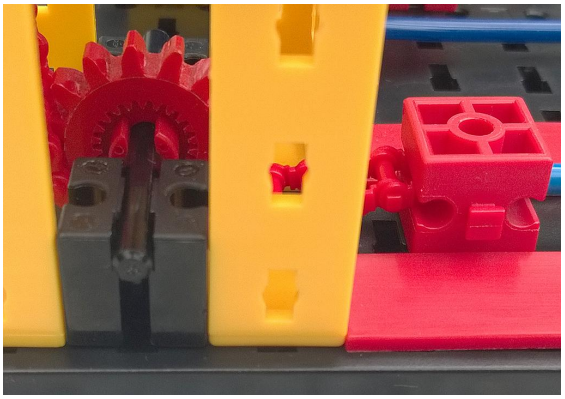


Abb. 9: Umlenkzahnrad

Der waagerechte Balken ist oben einfach durch eine simple Achse drehbar gelagert:

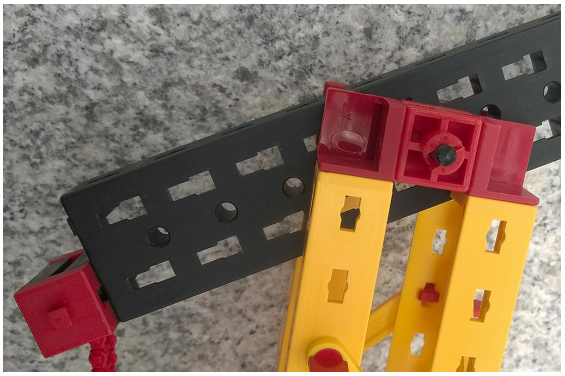


Abb. 10: Lagerung des Balkens und Befestigung der Kette

Abb. 11 zeigt einen Überblick über die komplizierteren Teile der Steuerung:

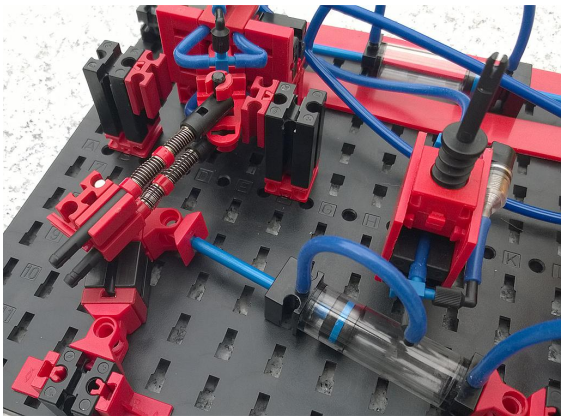
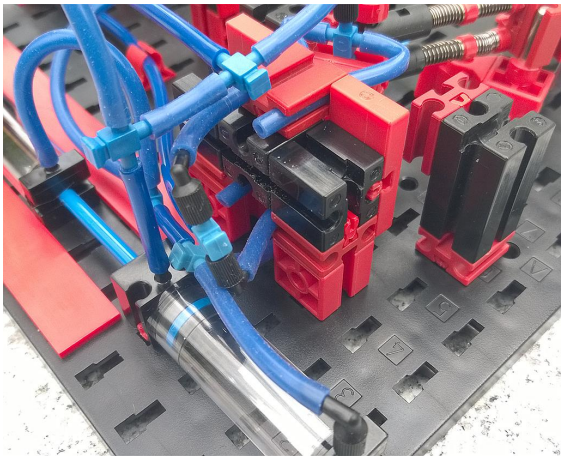


Abb. 11: Das Technikzentrum

Für den Nachbau der Maschine sollten die folgenden Hinweise nützlich sein:

Das 4/2-Wege-Ventil

Abb. 12 zeigt zwei Blickwinkel auf die Halterung des Ventils: Der zentrale BS15 wird beiderseits von zwei BS15 im Abstand von 22,5 mm von der Grundplatte (BS15 plus BS7,5) stramm gehalten. Der restliche Aufbau ist auch auf der Unterseite des mittleren BS15 genauso wie hier auf der Oberseite sichtbar: Mittels eines Federnockens sitzen oben und unten je ein BS7,5, in deren Nuten die Zu- und Abluftschläuche gehalten werden. Diese BS7,5 sind 5 mm nach vorne versetzt. Das ergibt sich übrigens einfach, wenn der Federnocken mit seiner langen Feder in den BS7,5 eingesetzt und sein Nocken bis zum Anschlag am Gelenkstein – siehe Abb. 13/14 – vorgeschoben wird.



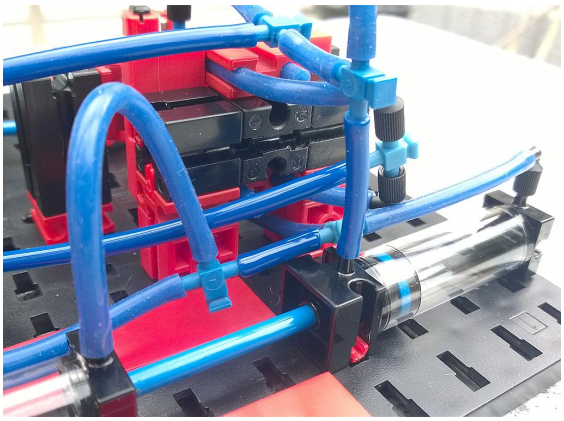


Abb. 12: Ventilhalterung mit Abluftschläuchen

Auf der Gelenkseite sitzen, wiederum per Federnocken angebracht, links und rechts je ein Baustein 5 · 15 · 30. Die sorgen dafür, dass die Schläuche nicht etwa seitlich aus den BS7,5 heraus können.

In Abb. 13 und 14 sieht man auch den auf dem zentralen BS5 aufgesetzten Gelenkstein.

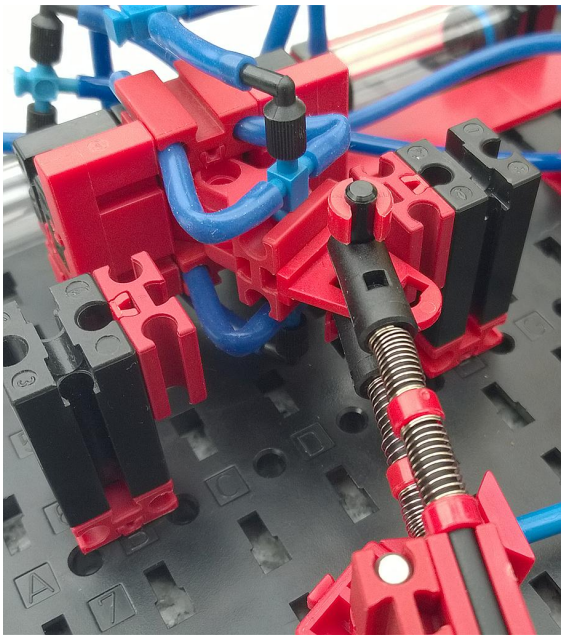


Abb. 13: Das Ventil bei abgesenktem Wimpel

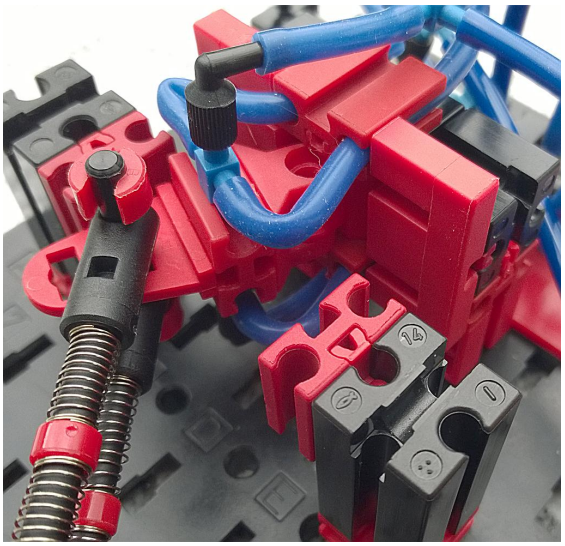


Abb. 14: Das Ventil bei angehobenem Wimpel

Auch auf diesem sitzt ein BS7,5, der oben und unten je ein Pneumatik-T-Stück trägt. Wie man in Abb. 15 sieht, gehen oben wie unten die beiden Schläuche der jeweiligen waagerechten BS7,5 an die beiden seitlichen Anschlüsse des T-Stücks, während die mittleren Anschlüsse über je ein Pneumatik-Winkelstück nach vorne herausgeführt werden.

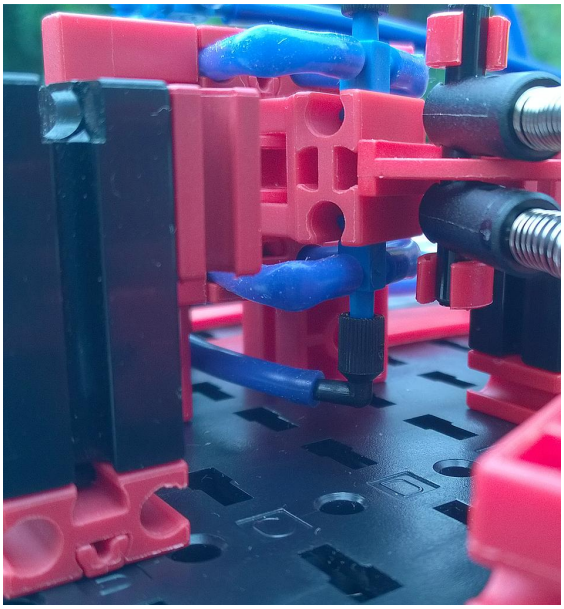


Abb. 15: Verschlauchung unter dem Ventil

Die insgesamt sechs Anschlüsse des Ventils werden wie folgt beschaltet:

- Wie Abb. 12 zeigt, werden der – von vorne gesehen – obere linke und untere rechte Schlauch der waagerechten BS7,5 mit dem Kompressor verbunden.
- Der obere rechte und untere linke Anschluss ist jeweils ein Abluftausgang. Diese Schläuche führen einfach ins Freie.
- Die zentralen Anschlüsse der beiden T-Stücke werden an die Ketten-Zylinder angeschlossen, und zwar der des unteren T-Stücks an die Seite zum Ausfahren der Zylinder, die des oberen T-Stücks an die Seite zum Einfahren. Beide Ketten-Zylinder teilen sich also dieselben Ausgänge. Außerdem gehen diese Anschlüsse auch an die beiden Drosseln.

Durch Kippen des Gelenksteins nach links bzw. rechts werden nun diejenigen Schlauchstückchen abgeknickt und damit

verschlossen, auf deren Seite der Gelenkstein gekippt wurde. Auf der anderen Seite können sich die Schläuche „frei entfalten“ und lassen dadurch genügend Druckluft durchströmen. Für Details der Wirkungsweise dieser Schlauchtechnik sei auf [1] verwiesen.

Anstatt also, wie es das Schaltzeichen eines 4/2-Wege-Ventils (Abb. 7) darstellt, einen beweglichen Teil zwischen festen Anschlüssen hin und her zu bewegen, steuern wir das Verbinden der zentralen Anschlüsse der beiden T-Stücke wahlweise mit Druckluft oder mit Abluft durch Zuklemmen der gerade nicht gewünschten Verbindung. Durch die Zufuhr der Druckluft auf zwei verschiedenen Seiten des Ventils erreichen wir es also, immer genau einen der beiden Zentralausgänge mit Druckluft und den jeweils anderen mit Abluft zu verbinden. Soweit funktioniert diese Ventilbauart genauso wie die in [1] vorgestellte und in Abb. 1 nochmal gezeigte.

Schwellwertschalter mit Springkontakt

Wir haben aber noch die Aufgabe, unser Ventil schlagartig umzuschalten, damit wir immer einen definierten Zustand und keinen ungewollten Druckluftverlust haben. Wie in Abb. 13 bis 15 gut sichtbar, sitzen auf dem Kipphebel über eine S-Kupplung 15 2 zwei Rastadapter, in denen je eine Rastachse feststeckt. Auf diesen sind Druckfedern aufgeschoben.

Im Modellbeispiel sind die originalen fischertechnik-Druckfedern verwendet, wie sie im älteren hobby-2-Baukasten enthalten waren und sich wieder in jüngeren Kästen finden. Davon benötigt man, um genügend Federspannung zu bekommen, vier Stück; je zwei pro Rastachse, hintereinander gesteckt und mit einem kleinen Abstandsring davor geschützt, sich ineinander zu verhaken.

Andere Druckfedern mit gut 4 mm Innendurchmesser werden auch funktionieren, wie man sie z. B. in manchen Kugelschreibern findet. Je nach Federstärke müsst ihr etwas mit der verfügbaren Länge der Achsen experimentieren, um einen sauberen Umschaltvorgang zu erzielen.

Abb. 16 zeigt den Hebel, der vom Steuerzylinder umgelegt wird:

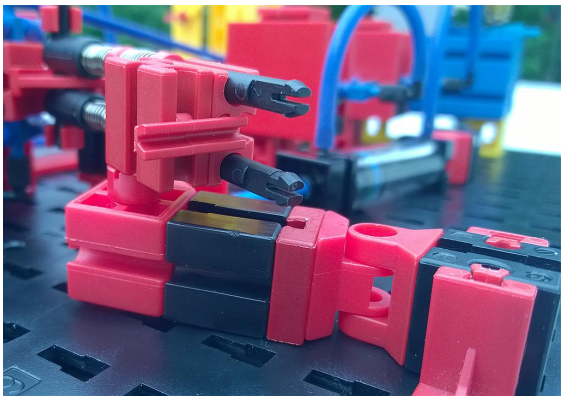


Abb. 16: Steuerhebel

Er ist stabil mit der Bauplatte verbunden und endet in einem BS15 mit Bohrung. In diesem steckt eine Metallachse 30 mit einem mittig angebrachten Klemmring. Die beiden Rastachsen laufen durch einen BS7,5, der auf einem um 90 ° verdrehten zweiten BS7,5 per Federnocken verbunden ist. Letzterer sitzt auf der senkrecht hochragenden Metallachse. Ein quer angesetzter Federnocken sichert den BS7,5 mit den Rastachsen dagegen, durch die Federkraft herausgedrückt zu werden.

Beim Bau stellte es sich als praktisch heraus, den Steuerzylinder auf der Bauplattenseite zu lösen, alle Federn und Abstandsringe auf die Rastachsen zu stecken und den Steuerhebel mitsamt den aufgesetzten BS7,5 fertig zusammengebaut auf der Bauplatte zu befestigen. Auf diese Weise kann man den Hebel z. B. ganz nach rechts kippen und komfortabel auf die Rastachsen schieben, bevor der Steuerzylinder wieder befestigt wird.

Die Funktionsweise ist nun so: Wird der Hebel auf eine der beiden Seiten gedreht,

erreichen die Rastachsen mit den Federn einen Punkt, ab dem Sie die Kippmechanik des eigentlichen Ventils von sich weg drücken. Das geschieht schlagartig, und erst wenn man den Hebel wieder fast ganz in die andere Richtung dreht, wird das Ventil – wiederum schlagartig – in seine vorherige Stellung zurück geschaltet.

Wichtig sind noch die beiden seitlichen Begrenzungen, gebildet aus je einem erhöht angebrachten BS30 mit einem darauf sitzenden BS7,5. Ohne die würde der Kipphebel des Ventils zu weit weg gedrückt werden können. Das hätte zur Folge, dass der Schaltpunkt fürs Zurückschalten des Ventils im nächsten Arbeitsgang nicht mehr erreicht werden könnte.

Steuerzylinder, Drosseln und Rückschlagventile

Der Steuerzylinder, der am Hebel schiebt und zieht, ist einfach über einen weiteren Winkelstein verbunden (siehe Abb. 6 und 11) und sitzt mit seinem anderen Ende ganz einfach mittels eines letzten Winkelsteins und eines BS15 auf der Bauplatte.

Seine beiden Druckluftleitungen gehen durch je eine Parallelschaltung aus einer selbstgebauten Drossel und eines Rückschlagventils (siehe [1]):

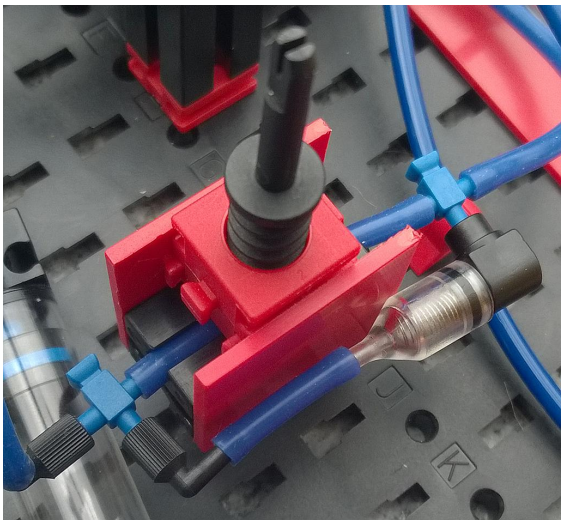


Abb. 17: Parallelschaltung von Drossel und Rückschlagventil

Das Rückschlagventil ist nichts anderes als das beim ft-Selbstbaukompressor (siehe [2]) mitgelieferte. Es lässt Druckluft nur „in Pfeilrichtung“ (man betrachte die Form dieses Bauteils in Abb. 17) durch. In diese Richtung kann die Luft also recht ungestört strömen. Nur in Sperrrichtung der Drossel muss sie sich durch den durch die Einstellschraube verengten Schlauchquerschnitt quälen und kann so nur langsam abströmen. Das Schaltzeichen des Rückschlagventils (Abb. 7) deutet an, wie es tatsächlich realisiert ist: Eine Kugel wird durch eine im Schaltzeichen nicht enthaltene, aber in Abb. 17 gut sichtbare Druckfeder gegen einen Ausgang gepresst. Nur von dort kommende Druckluft kann die Kugel gegen die Federkraft wegdrücken uns durch das Ventil strömen; in der anderen Richtung ist der Durchfluss blockiert. Zum Dämpfen eines Zylinders blockiert man die Abluft, nicht die Zuluft [1].

Zur Not kann man, wie schon gesagt, auch auf die Rückschlagventile oder sogar die ganzen Drosseln verzichten. Man verliert dadurch die einstellbare Wartezeit zwischen dem Auf und Ab des Wimpels, aber abgesehen davon wird das Modell auch so funktionieren.

Der Anschluss des Steuerzylinders ist ansonsten ganz einfach: Er muss, wenn die Verschlauchung genau wie beschrieben und nicht „vertauscht“ erfolgte, zusammen mit den Ketten-Zylindern aus- und einfahren, wird also bis auf die zusätzlichen Drosseleinheiten einfach parallel zu den beiden anderen Zylindern geschaltet.

Justage

Ein schrittweiser Aufbau und Test des Modells sei wärmstens empfohlen, um alles zuverlässig zur Funktion zu bringen:

1. Nachdem die Ketten-Zylinder und der ganze Turm fertig gebaut sind, prüft, ob die Verschiebung der Kette leichtgängig geht. Weitet ggf. die seitlichen Führungen etwas, indem die äußeren flachen

Platten eine Spur zum Rand der Bauplatte gezogen werden. Achtet darauf, dass sich weder der bewegliche Zylinder noch der BS15, in dem die Kette hängt, irgendwo verhaken.

2. Baut das 4/2-Wege-Ventil und schließt es so an, dass ihr durch manuelles Umlegen seines Kipphebels die Ketten-Zylinder zuverlässig ein- und ausfahren lassen könnt. Alleine das macht schon Spaß! Schließt bis dahin die Drosseln und den Steuerzylinder noch nicht an. Zieht die seitlichen Schläuche des Ventils nach vorne heraus und lasst sie durch Kippen des Hebels wieder einziehen. Das ergibt genau die richtige Länge, um das Zuklemmen bzw. Öffnen sauber zu realisieren.
3. Erst wenn das Ventil gut funktioniert, schließt den Steuerzylinder an. Lasst die Drosseln noch ganz geöffnet (Stellschraube ganz herausdrehen). Die Maschine muss jetzt bei eingeschalteter Druckluftversorgung alle Zylinder parallel hin und her und damit den Wimpel ständig auf und ab bewegen.
4. Dreht die Drosselventile nach Wunsch zu, um die jeweilige Wartezeit einzustellen. Man kann mehrere Sekunden erreichen, wenn man die Drosseln fast ganz schließt. In diesem Zustand unterscheidet ein Hauch einer kleinen Drehung zwischen „Stillstand“ und „noch zu schnell“. Hier ist also Fingerspitzengefühl gefragt. Wer kitzelt die längste zuverlässig funktionierende Wartezeit heraus?

Im Youtube-Video des laufenden Modells könnt ihr nachschauen, wenn etwas unklar geblieben sein sollte.

Wie geht’s weiter?

Wer einen oder zwei der fischertechnik Druckluftttanks besitzt, kann diese mit einem weiteren T-Stück zwischen die Drosseln und den Steuerzylinder zuschalten (alle anderen Anschlüsse sind natürlich mit einem P-Stopfen zu verschließen). Auf diese Weise können noch viel längere Wartezeiten (bis in den Minutenbereich!) eingestellt werden.

Im nächsten Beitrag dieser Serie widmen wir uns einem anderen Nachteil des bisherigen Ventils: „Es ist schwergängiger als die Kombination Betätiger/Festo-Ventil.“ Wir werden eine überraschende, frappierend einfache Anordnung vorstellen, die mit ganz wenigen aktuellen Bauteilen ein Druckluftsignal mit ganz wenig Kraft und wenig Schaltweg zu steuern vermag. Bleibt uns also treu!

Quellen

[1] Falk, Stefan: *Druckluftsteuerungen (Teil 1),* ft:pedia 1/2014, S. 58-72.

[2] Falk, Stefan: *Perlentauchen (Teil 5)* (Pneumatik). ft:pedia 4/2013, S. 6-15.

[3] Falk, Stefan: *Perlentauchen (Teil 3)* (Elektromechanik), ft:pedia 1/2013, S. 22-31.

[4] Wikipedia: *Oszillator*.

Computing

Nutzung des Universal-Interfaces 30520 als Port-Erweiterung an einem Mikrocontroller

Dirk Uffmann

In der Ausgabe 1/2014 der ft:pedia [1] hat Jens Lemkamp in seinem Beitrag gezeigt, wie man an einem Arduino-Board das alte ft-Parallel-Interface betreiben kann. Das funktioniert auch an einem beliebigen AVR-Mikrocontroller mit fünf freien I/Os, z. B. mit dem Board, das ich euch ebenfalls in der letzten Ausgabe der ft:pedia vorgestellt habe [2]. Oder mit einem in das Gehäuse des ft-Interfaces eingebauten Mini-Board von 30x40 mm, das sich leicht auf einer Lochrasterplatine aufbauen lässt. In diesem Beitrag verrate ich euch einen Trick, wie man mit Übertragungsraten von bis zu 800 kbit/s an dem alten Universal-Interface 30520 zeitlich parallel die Eingänge abfragen und die Motoren steuern – und sogar insgesamt zwei Interfaces für 16 Eingänge und acht Motoren betreiben kann.

Hintergrund

An dieser Stelle vermeide ich Wiederholungen und verweise für eine Einleitung in das Thema auf den Beitrag von Jens Lemkamp [1]. Das alte Universal-Interface 30520, das für den Betrieb an der parallelen Druckerschnittstelle LPT am IBM-PC gedacht war, hat einige interessante Eigenschaften, die sich noch heute nutzen lassen.

Abb. 1: Geöffnetes ft-Universal-Interface 30520 für die parallele PC-Schnittstelle

Funktionalität

Die Steuerung der Motoren und das Auslesen der Eingänge erfolgt im ft-Universal-Interface über Schieberegister, die mit mehr als 1 MHz getaktet werden können. Damit ist ein schneller Datenaustausch mit diesem Interface möglich. Diese Schieberegister sind zudem kaskadierbar und über den Erweiterungsport des Universal-Interfaces herausgeschleift, so dass mehrere Interfaces kaskadiert werden können. Zur Steuerung werden am Mikrocontroller nur fünf digitale I/O-Pins benötigt.

Die Eingänge E1-E8 am Interface sind mit 1 kOhm Pull-Down-Widerständen ausgelegt, was sie wenig anfällig für Störungen macht. Zudem sind die Motortreiber-ICs TLE4201 kräftig dimensioniert und können laut Datenblatt bis zu 2,5 A schalten. Also ein rundum recht robustes und zuverlässiges System.

Folgende Funktionalität des Universal-Interfaces wird durch diesen Beitrag am Mikrocontroller unterstützt:

- ein oder zwei Interfaces (im Master- und Slave-Betrieb)
- acht (bzw. 16) digitale Eingänge
- vier (bzw. acht) Motoren

Auf die analogen Eingänge Ex, Ey wird bewusst verzichtet, da dafür zwei weitere Pins am AVR benötigt würden (für die Signale TriggerX und TriggerY) und die Performance des AD-Wandlers am AVR viel mehr bietet als das Universal- Interface.

Eine Überlegung war noch, die Eingänge Ex und Ey direkt, also ohne die Timerschaltung mit dem NE556 an den AD-Wandler des Mikrocontrollers herauszuführen, um das RC-Tiefpass-Filter mit 470 Ohm und 470 nF zu nutzen (3 dB-Grenzfrequenz von 720 Hz für das Analogsignal). Dies wäre an den Pins 13 und 14 des 20-poligen, eingelöteten Flachbandsteckers im Interface möglich. Die Idee habe ich aber wegen der zu erwartenden Störungen durch die parallel verlaufenden Leitungen für die Kommunikation mit dem Interface verworfen.

Anschluss an das Mikrocontrollerboard

Um die volle Performance des Interfaces auszuschöpfen, schließen wir es an seinem intern eingelöteten Flachbandkabelstecker an. Dazu muss das Interface zunächst geöffnet werden. Man muss nur vier kleine Schrauben herausdrehen und den Klarsichtdeckel abnehmen. Dann löst man vorsichtig mit einem kleinen Schraubenzieher die Verriegelung für das Flachbandkabel an dem eingelöteten Stecker (Abb. 1 unten links) und nimmt das Oberteil ab.

Nun löst man das Flachbandkabel vorsichtig von den kleinen Messerkontakten. Neben dem Stecker sieht man auf der Platine die Anschlussnummern des Steckers von 1-20. Von diesen Pins bzw. Messerkontakten müssen wir nur sechs Stück anschließen und tun dies mit einem eigenen 6-poligen Flachbandkabel.

Abb. 2 zeigt die Pinzuordnung. Die Pinreihenfolge ist hier so zum AVR PORT zugeordnet, dass der interne Pfostenstecker im ft-Interface gut angeschlossen werden kann (passende Reihenfolge auf dem Flachbandkabel).

```
MEGA1284P(20-pol.)-----FTUI(20-pol.)
-----------------------------------
PA3-(Pin 10)---------LoadIn-(Pin 07)
PA4-(Pin 08)--------LoadOut-(Pin 08)
PA5-(Pin 06)----------Clock-(Pin 09)
PA6-(Pin 04)--------DataOut-(Pin 10)
PA7-(Pin 02)--------DataIn*-(Pin 17)
GND-(Pin 01)------------GND-(Pin 19)
```

Abb. 2: Pinzuordnung zwischen ATMEGA1284P und Universal-Interface

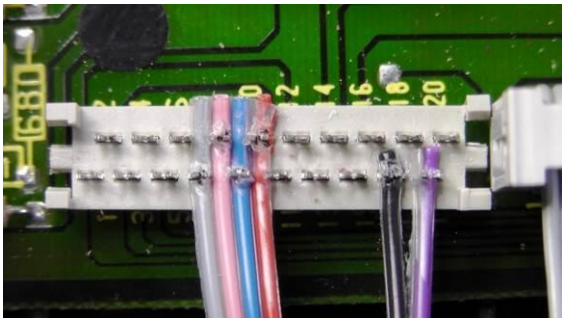


Abb. 3: Auflegen des sechspoligen Flachbandkabels am internen Flachbandkabelstecker des Interfaces

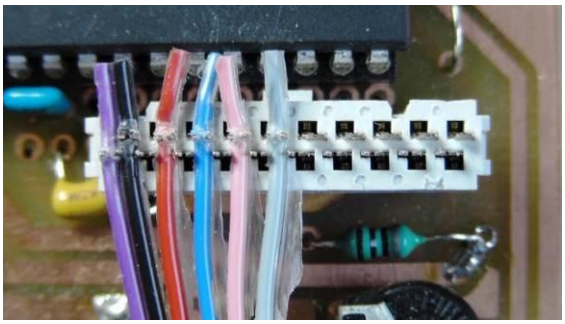


Abb. 4: Auflegen des sechspoligen Flachbandkabels am PORT A des Mikrocontrollers

Hier nun der Trick für hohe Datenraten: Eigentlich liegt DataIn auf Pin 18. Der Ausgang des Schieberegisters HCF4014 für die Eingänge E1-E8 wird auf dem Weg zu Pin 18 über ein Oder-Gatter HCF4071 mit dem Ausgangssignal der Timer NE556 der beiden Analogeingänge verknüpft und dann über einen Treibertransistor BC548 invertiert.

Da wir die Timersignale aber gar nicht brauchen und diese sogar stören, wenn die

Triggersignale dafür nicht auf high gelegt werden und die analogen Eingänge nicht über den Flachbandstecker auf die Timer durchgeschleift werden, verwenden wir lieber den direkten Ausgang des Schieberegisters an Pin 17. Hier hat das Ausgangssignal auch noch eine geringere Verzögerungszeit zur positiven Clockflanke, und daher können wir dann mit einer deutlich höheren Übertragungsrate arbeiten.

Ansteuerung des Interfaces: Motoren

Die Ansteuerung des Interfaces erfolgt nach einem einfachen Prinzip. Um die Motoren zu steuern, werden die Schieberegister HCF4094 mit einem seriellen Eingang und parallelen Ausgängen, die über ein Speicherregister die Motortreiber ansteuern, mit Daten beschrieben. Die Daten werden vom Mikrocontroller Bit-weise an DataOut bereitgestellt und bei der positiven Clockflanke in die erste Registerzelle geschrieben bzw. von dort in die folgenden Registerzellen weiter geschoben. Zuerst werden die zwei Bits für die höchste Motornummer übertragen, zuletzt die für Motor 1. Am Ende der Übertragung wird durch ein high-Signal auf der Leitung LoadOut der Inhalt des Schieberegisters in das Speicherregister übertragen und wird erst damit an den Motortreibern wirksam.

Abb. 5 zeigt die Signale für das Schreiben des Bytes 0b00000010, um bei einem angeschlossenen Interface den Motor 1 (mit bestimmter Drehrichtung: Ausgang gelb = V+, Ausgang orange = V-) einzuschalten. Die Übertragung läuft mit einer Datenrate von ca. 470 kbit/s, damit werden nur etwa 18 µs für ein Byte benötigt. Sind zwei Interfaces angeschlossen, werden 16 Bits übertragen. Abb. 6 zeigt die Signale für das Schreiben des Wortes 0b00001010 00000000 bzw. 0x0A00, um bei zwei angeschlossenen Interfaces die Motoren 5 (gelb = V+, orange = V-) und 6 (blau = V+, grün = V-) einzuschalten.

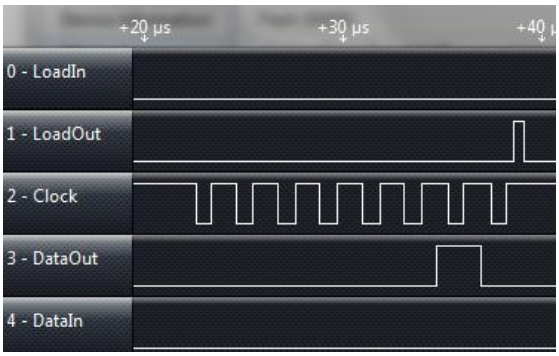


Abb. 5: Senden des Bytes 0x02 zum Einschalten von Motor 1

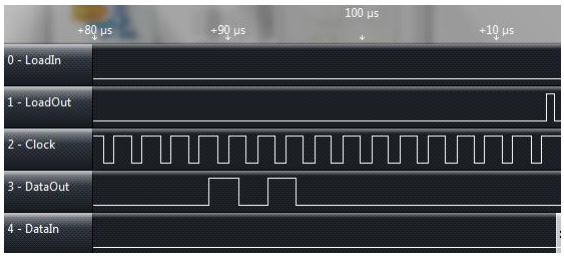


Abb. 6: Senden des Wortes 0x0A00 zum Einschalten der Motoren 5 und 6 mit gleicher Drehrichtung (am zweiten Interface)

Ansteuerung des Interfaces: Eingänge auslesen

Um die Eingänge auszulesen, muss LoadOut auf *low* gesetzt sein, da sonst die Motoren durch die im Schieberegister HCF4094 bewegten Bits umgeschaltet würden. Durch ein *high* auf LoadIn während einer positiven Clockflanke werden die Eingangswerte E1-E8 in das Schieberegister HCF4014 übernommen und das dem Eingang E8 zugeordnete Bit im ersten (Master)-Interface steht auf DataIn zum Einlesen in den Mikrocontroller bereit. LoadIn muss nun wieder auf *low* gezogen werden, wodurch das Schieberegister wieder von den Eingängen entkoppelt wird. Mit weiteren Takten werden die restlichen Bits eingelesen.

Abb. 8 zeigt das Einlesen des Daten-Bytes 0b00000100 oder 0x04. Hier war der Eingang E3 auf *high*.

Eine Besonderheit ergibt sich bei der Verwendung eines zweiten Interfaces. Die Bits für die Eingänge E16 bis E8 werden erst

nach den Bits für die Eingänge E8 bis E1 übertragen, also zuerst die Eingänge 8, 7, … bis 1 und dann die Eingänge 16, 15, ... bis 8. Abb. 9 zeigt die Signale beim Einlesen eines Wortes.

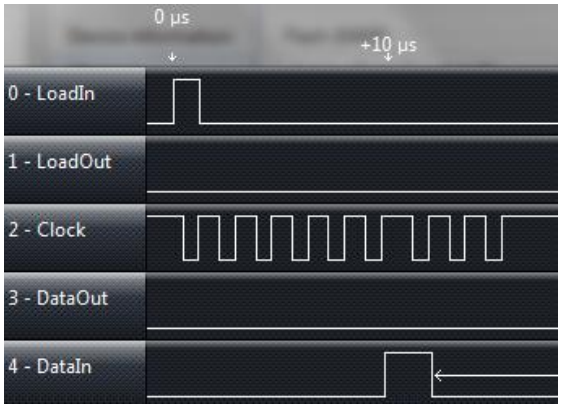


Abb. 8: Einlesen des Bytes 0x04. Hier war der Eingang E3 auf high.

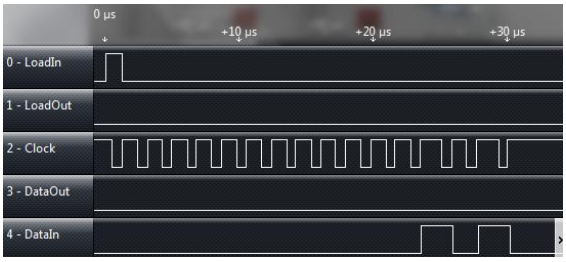


Abb. 9: Einlesen des Wortes 0x000A. Zuerst die Bits vom Master-Interface, danach die vom Slave. Eingänge E12 und E10 waren auf high.

Parallele Ansteuerung

Doch das Beste kommt zum Schluss: nachdem das Senden und Empfangen jeweils für sich so gut und schnell funktionierte, kam ich auf die Idee, das Senden und Empfangen zeitlich parallel abzuwickeln, wie beim SPI. Es sind ja zwei Datenleitungen vorhanden, warum also nicht in einer Taktfolge parallel sowohl Senden als auch Empfangen und beide Register updaten? Damit kommt man auf die Hälfte der Übertragungszeit.

Eine weitere Steigerung der Taktrate war möglich, indem das Auswerten von DataIn auf die *low*-Phase der Clock verlegt wurde (Einsparen der Wartezeit während der *high*-Phase). Abb. 10 zeigt die Signale für den Betrieb mit einem Interface. Hier wurden die Eingänge 1 und 4 auf *high* gesetzt und die Motoren 1 und 2 mit unterschiedlicher Drehrichtung eingeschaltet. Innerhalb von 10 µs ist ein Update des Eingangs- und Ausgangsstatus erfolgt. Die Datenrate liegt bei 800 kbit/s (in beiden Datenflussrichtungen).

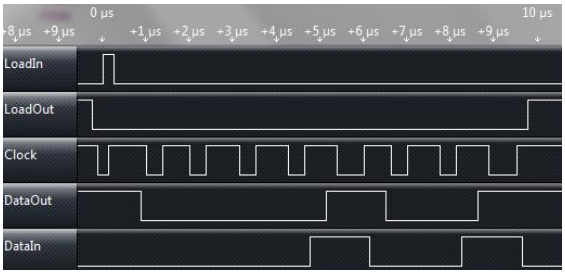


Abb. 10: Zeitlich paralleles Empfangen des Bytes 0x09 (E4 & E1 high) und Senden des Bytes 0x09 (Motor 1 & 2 mit unterschiedlicher Drehrichtung einschalten).

Die Signale für den Betrieb von zwei Interfaces zeigt Abb. 11. Hier werden ca. 28 µs für die 16 Bit benötigt. In diesem Fall wird etwas mehr Zeit pro Bit benötigt, da 16-Bit-Variablen verarbeitet werden müssen. Die Datenrate liegt dann bei 570 kbit/s.

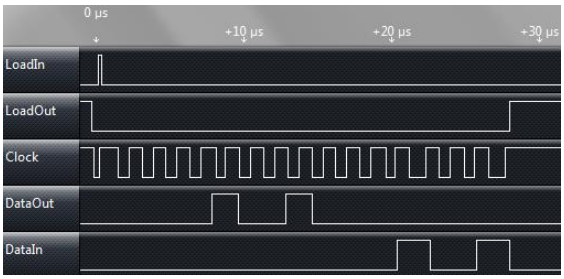


Abb. 11: Zeitlich paralleles Empfangen des Wortes 0x0009 (E9 & E12 high) und Senden des Wortes 0x0900 (Motor 5 & 6 anschalten)

Den Code für diese C-Funktion zeigt Abb. 12. Zusammen mit einem kleinen Programm zum Testen von vier Motoren und acht Tastern nimmt diese kleine Interface-Library weniger als 600 Bytes im Flash ein. Ich habe den Code mit zwei angeschlossenen Interfaces ausgiebig getestet und keine Fehler mehr gefunden. Die benötigten Funktionen für den Programmcode des Mikrocontrollers und den Code

zum Testen steht in einem Downloadpaket zur Verfügung.

```
/*****************************************************************************
      FTUI_update überträgt das motorword und liest parallel das inputword ein.
      Diese Funktion muss mindestens alle 0,3 Sekunden aufgerufen werden, um den
      Watchdog im FT-Interface zurückzusetzen (durch das Clocksignal).
      Bei einem Systemtakt von 18,432 MHz braucht die Routine mit einem Interface
      nur ca. 10 µs, mit zweien 28 µs.
*****************************************************************************/
void FTUI_update (void)
{
#if slave
      uint16_t mask_motor=(1<<15);
      uint16_t mask_input=(1<<7);
#else
      uint8_t mask_motor=(1<<7);
      uint8_t mask_input=(1<<7);
#endif
//    Zuerst wird am CMOS4094 das Speicherregister für die Motorausgänge vom
//    Schieberegister entkoppelt für den Transfer des motorwords
      FTUI_Port_Write &= ~(1<<FTUI_LoadOut);
      FTUI_Port_Write &= ~(1<<FTUI_Clock); //Clock low-Phase einleiten
//    Zunächst werden alle Eingänge als 0 angenommen.
      inputword = 0;
//    CMOS4014: Nun werden die Eingangszustände in das Schieberegister übernommen.
//    Dazu muß LoadIn gesetzt werden und eine positive Clockflanke erzeugt werden.
      FTUI_Port_Write |= (1<<FTUI_LoadIn);
      FTUI_Port_Write |= (1<<FTUI_Clock); //Clock high-Phase einleiten
//    mit dieser positiven Clockflanke steht das erste Bit auf der
//    DataIn-Leitung an, jedoch mit einer Verzögerung. Daher wird
//    es erst in der nächsten Clock-Low-Phase eingelesen
//    CMOS4014: Dann werden die Eingänge wieder vom Schieberegister entkoppelt
      FTUI_Port_Write &= ~(1<<FTUI_LoadIn);
//    Dann werden die Bits 7 bis 0 (ein FT-Interface) bzw. 15 bis 0 (zwei
//    FT-Interfaces) gesendet (motorword) und parallel die Bits 7 bis 0 und
//    dann 15 bis 8 (zwei FT-Interfaces) eingelesen (inputword)
      while (mask_motor)
      {
//          Wenn das jeweilige Sende-Bit high sein soll, dann wird DataOut = 1
//          gesetzt, sonst DataOut = 0
            if(motorword & mask_motor) FTUI_Port_Write |= (1<<FTUI_DataOut);
            else FTUI_Port_Write &= ~(1<<FTUI_DataOut);
            FTUI_Port_Write &= ~(1<<FTUI_Clock); //Clock low-Phase einleiten
            mask_motor = (mask_motor>>1); //Schiebe das Maskierungsbit nach rechts
//          Nun wird das jeweilige Bit im Datenstream auf der Leitung DataIn
//          daraufhin getestet, ob der zugehörige Eingang "high" ist,
//          und dann wird inputword an der jeweilige Bitposition auf 1 korrigiert
            if (FTUI_Port_Read & (1<<FTUI_DataIn)) inputword |= mask_input;
            mask_input = (mask_input>>1); //Maskierungsbit nach rechts schieben
#if slave
            if (mask_input==0) mask_input = (1<<15);//Maskierungsbit auf Pos. 15
#endif
            FTUI_Port_Write |= (1<<FTUI_Clock); //Clock high-Phase einleiten
      }
//    CMOS4094: Durchschalten der Daten aus dem Schieberegister ins
//    Speicherregister für die Motorausgänge
      FTUI_Port_Write |= (1<<FTUI_LoadOut);
}
```

Abb. 12: C-Code für die Funktion FTUI_update() für zeitlich paralleles Empfangen und Senden

Zum Schluss noch zwei weitere Ideen:

- Durch den Einbau eines kleinen Mikrocontroller-Boards mit Infrarot-Empfänger in das Gehäuse des Interfaces ließe sich eine offline-fähige Steuereinheit oder ein IR-Control-Set für vier Motoren daraus machen.

So entfiele das externe Flachbandkabel zum Steuerrechner. Der Einbau ist von außen kaum zu erkennen, so dass euch ein erstes Erstaunen von ft-Fans sicher sein dürfte, wie ihr es geschafft habt, das ft-Interface offline-fähig zu machen.

Abb. 13 zeigt, wie das Board der Größe 30x40 mm in das Universal-Interface eingebaut werden kann. Die 5 V-Versorgung wird mit dem schwarzen Kabel zugeführt, das an einem Draht des blauen Kondensators (47 nF) angelötet ist.

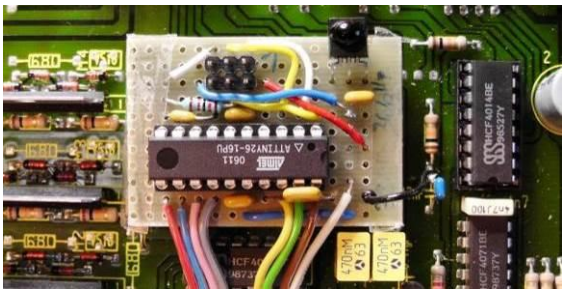


Abb. 13: In das Interface eingebautes Board

Das kleine Board fügt sich gut in den vorhandenen Platz ein. Die Unterseite ist mit Pappe isoliert, die mit Tesafilm am Rand des Boards festgeklebt wurde (Abb. 14).

Abb. 14: Erweitertes Universal-Interface

Hier nutze ich auch die analogen Eingänge Ex und Ey an Pin 13 & 14 des Interfaces, um sie direkt den analogen Eingängen PA1 & 2 des ATTiny26 zuzuführen.

Der Infrarot-Empfänger liegt im Sichtfenster neben der LED, während der Rest des Boards weitgehend hinter dem schwarzen Aufkleber verdeckt ist. Dadurch ist die Modifikation kaum zu erkennen (Abb 15).

Abb. 15: Universal-Interface mit Abdeckung

- Die schnelle Datenrate ließe sich nutzen, um eine PWM-Funktionalität für die Motoren zu implementieren. Eine einfache Variante wären z. B. drei zusätzliche PWM-Stufen mit 25, 50 und 75 % Motorleistung. Dafür müssten nur viermal in einem zeitlich fest definierten Ablauf periodisch (entspricht der PWM-Periode bzw. PWM-Frequenz) die Bits in der Variable „motorword“ umgeschaltet werden. Zu Beginn des Ablaufes werden alle Bits entsprechend „motorword“ gesetzt, nach einem Viertel der Zeit werden die Motoren mit 25 % PWM wieder abgeschaltet, nach der Hälfte die mit 50 % usw.

Ich wünsche euch viel Freude damit und vor allem mit der Wiederbelebung eurer alten Universal-Interfaces, die durchaus noch nützlich sein können. Nun sind eure weiteren Ideen dazu gefragt.

Quellen

[1] Jens Lemkamp: *Parallel-Interface durch Arduino gesteuert (1)*. ft:pedia 1/2014, S. 24-30.

[2] Dirk Uffmann: *ft-Modellsteuerung mit selbst gebautem Mikrocontroller-Board*. ft:pedia 1/2014, S. 39-46.

Computing

ft-Interface durch Arduino gesteuert (2)

Jens Lemkamp

1981 brachte fischertechnik die Elektronik 30253 heraus – als Nachfolger der guten alten Silberlinge. Viele Fans haben noch Bestände im Schrank, auch kann man noch gebrauchte Module günstig erwerben. Unser kleines Projekt zeigt, wie man diese Bausteine mit dem Arduino verknüpfen kann: Wir erzeugen damit Töne, steuern Motoren und Lampen.

Die Elektronik

In diesem zweiten Teil konzentrieren wir uns auf den Anschluss der Elektronik an den Arduino.

Zum Arduino selbst will ich nicht viel erklären, sondern verweise auf den Artikel in der ftpedia 1/2014 [1], in dem ich erläutert habe, wie sich mit dem Arduino ein ‚altes' ft-Computing-Interface ansteuern lässt. Ferner ist die Arduino-Hompage sehr ergiebig [2, 3].

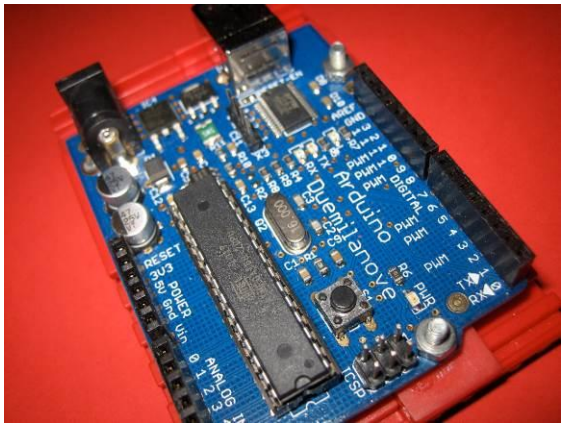


Abb. 1: Arduino Duemilanove

Um schnell in die Materie einzusteigen, starten wir mit einem einfachen Experiment.

Ansteuerung einer Lampe

Zunächst benötigen wir Adapterkabel, um von den Pins der fischertechnik-Elektronik an den Arduino ‚andocken' zu können (Abb. 2). Davon sollte man sich einige Exemplare anfertigen, am besten in verschiedenen Farben und Längen. Für den Anfang benötigen wir nur je ein rotes und ein blaues Kabel.

Abb. 2: Adapterkabel

Weiterhin benötigen wir für den ersten Versuch die folgende Teile:

- ein Baustein IC-Spannungsversorgung
- ein Baustein Leistungsstufe (LST)
- ein Arduino mit USB-Kabel
- eine Lampe
- ein Netzgerät (oder den ft-Akku)
- ein ft-Interface

Wir verbinden nun die +5 V- und Masse-Schienen der beiden Bausteine und versorgen die IC-Spannungsversorgung nach Anleitung mit Strom.

Den Anschluss ‚B1' der Leistungsstufe verbinden wir mit dem Arduino-Pin 13, und den Masse-Anschluss (GND) des Arduinos mit 0 V der ft-Elektronik (GND liegt direkt neben Pin 13). Die Pins bitte nicht verwechseln; eine Verpolung kann den sofortigen und endgültigen Halbleitertod des Arduinos bedeuten. Deswegen stellen wir alle Verbindungen immer im spannungslosen Zustand her, kontrollieren diese am besten zwei Mal und schalten dann erst die Spannungsversorgung ein.

Zwischen den Anschluss ‚C1' der Elektronik und +5-10 V schalten wir eine Lampe. Der Regler ‚P1' darf auf Rechtsanschlag gedreht werden.

Software

Nun laden wir das Programm ‚Blink' in den Arduino. Aus der Arduino-Bibliothek kann es einfach aufgerufen werden (*File – Examples – 1. Basics – Blink*). Je nach Softwareversion kann der Verzeichnispfad abweichen, aber eigentlich sollte man ‚Blink' finden können. Zur Sicherheit ist das Programm hier nochmal abgedruckt; es darf frei verwendet werden:

```
/*
  Blink
  Turns on an LED on for one second, then
  off for one second, repeatedly.

  This example code is in the public
  domain.
 */

void setup() {
  // initialize the digital pin as an
     output:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000);            // wait for a second
  digitalWrite(13, LOW); // set the LED off
  delay(1000);            // wait for a second
}
```

Das Programm macht nichts anderes, als zunächst den Anschluss Pin 13 als Ausgang zu definieren und anschließend in einer Endlosschleife diesen Ausgang zunächst auf +5 V (*high*) zu setzen, dann eine Sekunde Pause einzufügen, darauf den Pin 13 wieder auf 0 V (*low*) zu schalten und erneut eine Sekunde Pause einlegen.

Der Ausgang Pin 13 des Arduino ist ja mit dem ‚B1'-Eingang unserer LST verbunden und schaltet bei +5 V via Transistor die Lampe ein und bei 0 V wieder aus. Wem es zu hell ist, der kann das Poti ‚P1' nach links verdrehen und damit den Strom durch die Lampe herunter regeln (siehe Anleitung der Elektronik).

Man könnte jetzt den ‚B2' unserer LST mit einem anderen Ausgang des Arduino verbinden, die beiden Brücken ‚BR' der Elektronik mit 0 V verbinden, eine weitere Lampe oder einen Motor anschließen und dann entsprechend eine zweite Funktion mit dem Arduino steuern.

In Verbindung mit dem ft-Parallel-Interface ergeben sich neue Möglichkeiten: vier Kanäle links/rechts steuerbar auf dem Interface [1] und jetzt zusätzlich zwei weitere Ein-/Aus-Schaltfunktionen z. B. für Signalleuchten. Natürlich lässt sich auch ein Elektro-Magnet oder ein Magnetventil anschliessen – oder auch ein Lautsprecher.

Ansteuerung eines Lautsprechers

In den Versuchen des Anleitungsbuchs der Elektronik wird beschrieben, wie man Töne erzeugen kann, in dem man statt der Lampe einen Lautsprecher an den Ausgang der Elektronik anschliesst.

Natürlich geht das auch hier. Dem Arduino ist es prinzipiell natürlich egal, was hinten dranhängt. Wenn wir das gleiche Programm ‚Blink' mit einem angeschlossenen Lautsprecher starten, wird dieser in einem Takt von einer Sekunde aus- und wieder eingeschaltet. Das macht sich dann in

einem deutlichen Knacken bemerkbar; evtl. hört man im Hintergrund noch ein leichtes Brummen, das durch die nicht ganz saubere Gleichspannung der Spannungsversorgung hervorgerufen wird.

Abb. 3: Elektronik mit ft-Lautsprecher (siehe [4], S. 27-28)

Die Maßeinheit für die Anzahl der Ein- und Ausschaltvorgänge pro Sekunde heißt übrigens ‚Hertz' (abgekürzt: Hz), benannt nach dem deutschen Physiker *Heinrich Hertz*, (1857-1894), Formelzeichen ‚*f*'.

Man sagt also: die Frequenz *f* beträgt 1 Hz, wenn die Zeit vom Einschalten bis zum nächsten Einschalten genau eine Sekunde beträgt. Da wir eine Sekunde einschalten, dann noch eine Sekunde Pause machen, beträgt die Zeit aber zwei Sekunden für einen vollen Zyklus. Man spricht dabei auch von einer Periode. Unsere Frequenz ist damit nur halb so groß und beträgt 0,5 Hz.

Wenn wir die Zeiten in den Delays für Ein- und Auschalten halbieren, also auf 500 Einheiten (Millisekunden) ändern, erhalten wir am Ausgang eine Frequenz von 1 Hz. Die Frequenz wird vom internen Takt des Arduinos bestimmt und ist aus bestimmten technischen Gründen am Ausgang nicht ganz genau 1 Hz, aber das soll uns zunächst nicht stören.

Bei diesen vergleichsweise niedrigen Frequenzen hören wir aber etwas anderes, denn der Lautsprecher knackt einmal beim Einschalten und nochmal beim Ausschalten. Damit hören wir die doppelte Impulszahl – beim Vergleich mit der internen LED des Arduino, die immer mit Pin 13 verbunden ist, kann man das gut beobachten.

Zum Vergleich: Die Frequenz unseres Stromnetzes, also die Anzahl der Polaritätswechsel unserer Spannung aus der Steckdose, beträgt 50 Hz, d. h. die Netzspannung wechselt ihre Polarität 50 Mal pro Sekunde. Mit dem grauen alten fischertechnik-Travo konnte man diese Frequenz am Wechseltromanschluss abgreifen und z. B. als Takt einer 50-Hz-Uhr verwenden [5].

In unserem Experiment würde sich das als Brummen im Lautsprecher bemerkbar machen. Aber *Achtung*: Nie den Lautsprecher an eine Steckdose anschließen, das ist lebensgefährlich – und der Lautsprecher ist auch nicht für eine Netzspannung von 230 V ausgelegt.

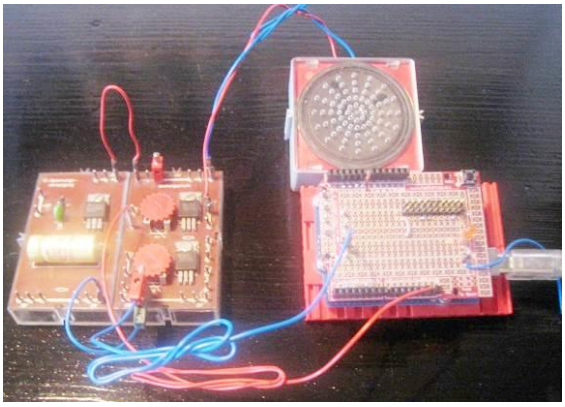


Abb. 4: Anschluss von Lautsprecher und Elektronik an den Arduino

Wenn wir die Delays auf jeweils ‚50' setzen, wird das Knacken langsam zum Dauerton, und bei ‚10' hören wir wir nur noch ein Brummen. Die 2x10 ms sind kein Zufall, sondern entsprechen 50 Hz – eine 50stel Sekunde sind 0,02 s (= 20 ms). Anders ausgedrückt: Der Kehrwert der Frequenz ist die sogenannte Periodendauer – Ausschaltzeit plus Abschaltzeit.

Setzen wir das Delay auf 1 ms, erhalten wir eine Frequenz, die 10 mal so hoch ist, also 500 Hz. Das ist die höchste Frequenz, die wir auf diesem Wege erzeugen können

– der so erzeugte Ton entspricht etwa der Note „b“ (siehe Frequenztabelle).

Der Arduino kann aber noch schneller und hat dafür einen Befehl, der uns in die Lage versetzt, die Frequenz, die wir hören möchten, direkt anzugeben. Hier das komplette Programm:

```
void setup() {
}
void loop() {
  tone(13, 1000, 100);
  delay(1000);
}
```

Der Befehl „tone“ benötigt die folgenden drei Parameter (in der Klammer):

- **13** ist der Ausgangspin, also der Pin, an dem unser Lautsprecherbaustein über die LST angeschlossen wurde. Es geht auch jeder ander Pin des Arduinos
- **1000** ist die Frequenz (in Hz)
- **100** ist die Dauer in ms, für die dieser Ton ausgegeben wird – hier also wird der Ton nach 100 ms deaktiviert

Wir machen eine Sekunde Pause delay(1000) und die Schleife beginnt von vorn. Ich habe die Frequenz 1.000 Hz (= 1 kHz) nicht willkürlich gewählt: Dieser Ton war früher der Testton der Fernsehanstalten, der während der Sendepause bzw. des Testbildes ausgestrahlt wurde.

Für die jüngeren Leser: Es gab einmal eine Zeit, da wurden nur drei Fernsehprogramme ausgestrahlt, und nicht mal rund um die Uhr. Irgendwann in der Nacht war „Sendeschluss“ – und ab da gab es bis morgens das Testbild mit dem 1-kHz-Testton.

Heute verwenden diesen Ton oft noch Tontechniker zum Einmessen von Systemen und Lautstärkepegeln.

Quellen

[1] Jens Lemkamp: *Parallel-Interface durch Arduino gesteuert (1)*. ft:pedia 1/2014, S. 24-30.

[2] http://www.arduino.cc

[3] http://www.fritzing.org

[4] Stefan Falk: *Perlentauchen (4)*. ft:pedia 2/2013, S. 18-30.

[5] Dirk Fox: *Der Elektromotor*. ft:pedia 3/2013, S. 4-8.

Computing

Von Kameras, Himbeeren und schwarzen Hundeknochen

Erik Andresen

Über fünf Jahre nach der Einführung des TX-Controllers ist es an der Zeit, ft-Modelle mit Kameras auszustatten. Für die Umsetzung eignen sich preisgünstige ARM-Boards wie der Raspberry Pi oder das Beaglebone Black am Robo-Interface. Als Kamera kann dabei jede mit Linux kompatible USB-Webcam verwendet werden. Die hier vorgestellten Bibliotheken OpenCV und GStreamer helfen bei der Auswertung und Visualisierung der Kamerabilder.

Moderne Einplatinencomputer, darunter der Raspberry Pi von der gleichnamigen britischen Foundation und das Beaglebone Black (BBB) von Texas Instruments, sind für weniger als 50 € zu haben. Über deren USB-Host-Schnittstelle lassen sich diverse Peripherie-Geräte per Plug'n'Play anschließen. Wie sich die Boards mit einem fischertechnik-Modell verbinden lassen, wird in diesem Beitrag erklärt. Als Basismodell eignet sich dafür der Robo Explorer (Abb. 1), an dem dafür folgende USB-Geräte angeschlossen werden:

- W-Lan-Stick zur drahtlosen Kommunikation mit dem Modell und einer gegenüber dem ROBO RF Data Link höheren Funkreichweite und
- USB-Webcam als neuer Sensor für den Roboter.

Da mit dem Interface die Anzahl der USB-Peripheriegeräte (drei) höher ist als die Anzahl der USB-Host-Ports von Raspberry Pi und BBB, wird ein passiver USB-Hub mit vier Ports zwischengeschaltet.

Im ersten Abschnitt dieses Beitrags wird das Kamerabild über W-Lan übertragen. Im zweiten wird dem erweiterten Robo Explorer beigebracht, Objekte mit dem schnellen Camshift-Algorithmus zu verfolgen. Als Programmiersprache wird Python verwendet.

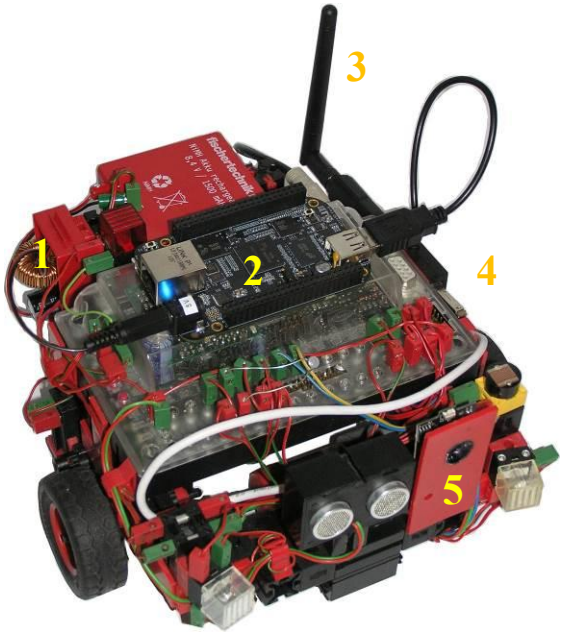


Abb. 1: Mit Beaglebone Black und USB-Webcam getunter Robo Explorer: 5 V Spannungsregler (1), Beaglebone Black (2), WLan-USB-Stick (3), USB-Hub (4) und USB-Webcam (5)

Als Kamera kann jede mit Linux kompatible USB-Webcam [6] verwendet werden, auf die über die Video4Linux-(V4L-)API zugegriffen wird. Hat die USB-Webcam zusätzlich ein eingebautes Mikrofon, lässt

sich außerdem eine Sprachsteuerung z. B. mit PocketSphinx [5] realisieren.

Ein funktionierendes Raspbian Wheezy auf dem Raspberry Pi bzw. Debian Wheezy auf dem Beaglebone Black wird vorausgesetzt. Installationsanleitungen dafür sind im Internet genügend zu finden. Generell kann auch jeder andere Einplatinencomputer mit USB-Host-Schnittstelle oder jede andere Linux-Distribution verwendet werden; dabei kann es jedoch im Detail zu Abweichungen von den folgenden Beschreibungen kommen.

Da auf den ARM-Boards Linux als Betriebssystem zum Einsatz kommt, wird die *libroboint* [1] zur Kommunikation mit den Interfaces verwendet; die Installation ist in [2] beschrieben. Als Interface zwischen Einplatinencomputer und Modell können damit das ROBO Interface, die Robo I/O Extension sowie das Intelligent Interface verwendet werden; der TX Controller wird nicht unterstützt. Für die Verwendung des Intelligent Interfaces ist zusätzlich ein USB-RS232 Adapter (z. B. Ftdi) erforderlich.

Will man die ARM-Boards mit einem mobilen Modell verwenden, muss die Spannungsversorgung über einen Akku erfolgen. Beide Boards benötigen eine 5 V Spannungsversorgung, der Raspberry Pi in Form eines Micro-USB-Steckers (Abb. 2), der BBB als DC-Stecker mit 2,1 mm Stift- (+5 V) und 5,5 mm Außendurchmesser (Masse). Zur Spannungsversorgung des Raspberry Pi eignen sich USB-Battery-Packs, wie sie zum Aufladen von Smartphone-Akkus angeboten werden. Um diese mit dem BBB zu verwenden, muss ein Adapter beschafft werden, der die Spannung vom USB-Stecker auf den DC-Stecker führt.

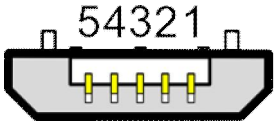


Abb. 2: Pinbelegung Micro-USB-Stecker zur Spannungsversorgung des Raspberry Pi: Pin 1 ist +5 V, Pin 5 ist Masse. (Bild: Wikimedia Foundation)

Wer die Spannungsversorgung des Interfaces mitverwenden möchte, braucht einen Spannungsregler, der die ft-üblichen 9 V auf 5 V herunter regelt. Als Spannungsregler eignet sich z. B. der LM2576, der mit 3 A mehr Strom liefern kann als der beliebte 7805. Der LM2576 benötigt dafür jedoch mindestens eine Eingangsspannung von 7 V, die der ft-Akku aber mit einer Nennspannung von 7,2 V bis auf die letzten zehn Minuten liefert. Die dazu benötigten Bauteile (Tabelle 1) werden dafür wie in Abb. 3 dargestellt verlötet.

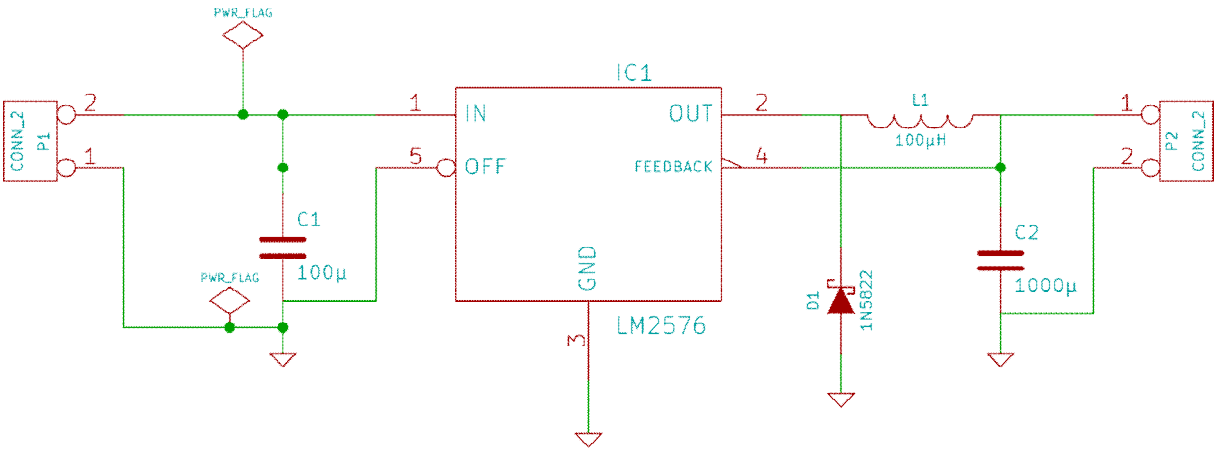


Abb. 3: Schaltungsbeispiel Spannungserzeugung für Einplatinencomputer: An P1 wird eine ca. 9 V große Eingangsspannung eingespeist und durch den LM2576 Spannungsregler in eine 5 V Spannung umgewandelt, die an P2 ausgegeben wird

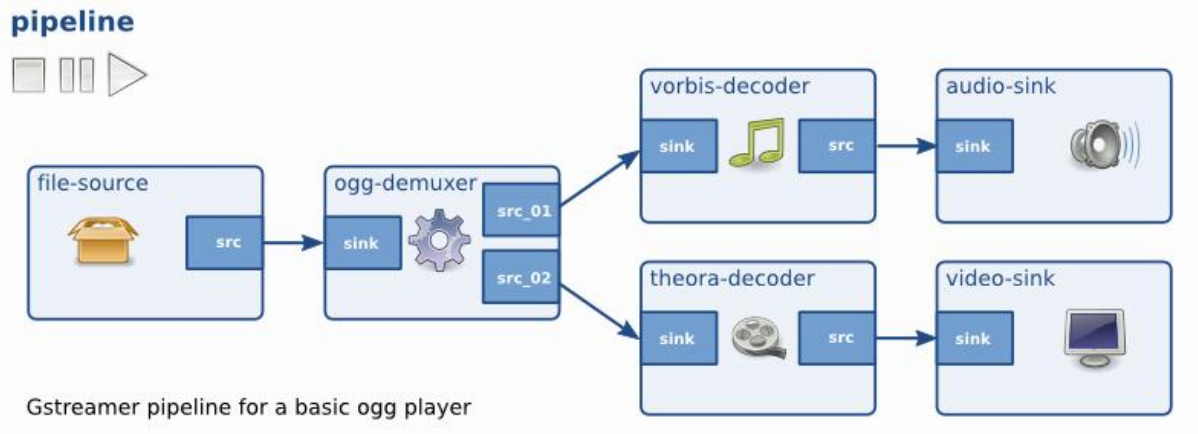


Abb. 4: GStreamer-Pipeline zum Senden der Kamerabilder als MJPEG über Netzwerk

Nr.	Anzahl	Wert
C1	1	100 µF
C2	1	1000 µF
D1	1	1N5822
IC1	1	LM2576
L1	1	100 µH (bis 3 A)

Tabelle 1: Stückliste für LM2576 Schaltungsbeispiel

Vorsicht: Ein falscher Zusammenbau kann den Einplatinencomputer zerstören!

Streaming von Videos mit dem Framework GStreamer

Für die Visualisierung der Kamerabilder auf einem entfernten PC wird das freie Multimedia-Framework GStreamer (GST) [9] verwendet. GStreamer gründet auf dem Konzept einer Video-Pipeline und wird unter Linux-PCs und einigen eingebetteten Systemen, wie den Smartphones Jolla und Nokia N900, für die Wiedergabe von Multimedia-Inhalten eingesetzt.

GStreamer ist modular aufgebaut: Jedes Element in der Pipeline führt einen Arbeitsschritt aus, z. B. das Lesen einer Datei, die Dekompression von Daten oder die Ausgabe von Daten auf einen Bildschirm. Durch die Verkettung von Elementen zu einer Pipeline (Abb. 4) werden komplexe Aufgaben wie die Wiedergabe von Multimedia-Inhalten durchgeführt. Verbunden werden die Elemente über ihre Pads (Kontaktstellen). Ein Pad ist entweder Quelle (*source*, kurz *src*) oder Senke (*sink*). Über die Quelle werden Daten in das Element eingelesen und über die Senke ausgegeben. Elemente verfügen über ein Pad, mehrere Pads oder kein Pad eines Typs. Ein Pad definiert die Daten, mit denen es arbeitet, über dessen *Capabilities* (Eigenschaften, kurz: *Caps*), mit denen beim Aufbau einer GStreamer-Pipeline auf Fehler, wie die versehentliche Anbindung einer Audio-Quelle an eine Videoausgabe,

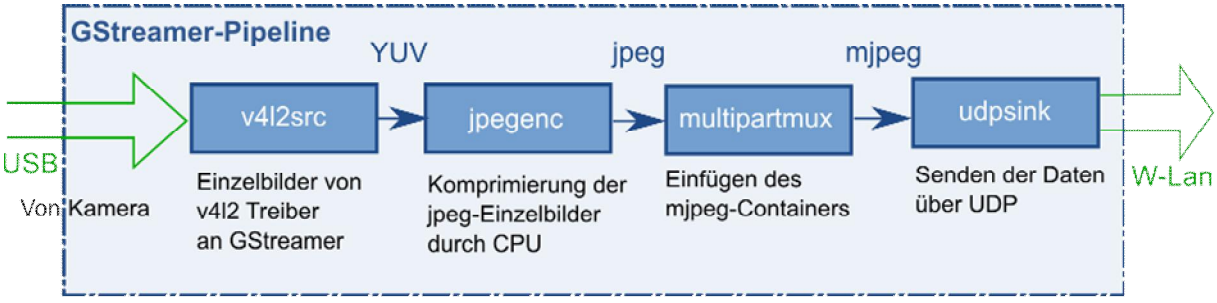


Abb. 5: GStreamer-Pipeline zur Wiedergabe von Audio- und Videodateien, die mit Ogg (theora und vorbis) komprimiert wurden [9].

```
$ sudo apt-get update
$ sudo apt-get install gstreamer-tools gstreamer0.10-plugins-base \
  gstreamer0.10-plugins-good gstreamer0.10-plugins-bad
```

Listing 1: Installation von GStreamer

```
$ gst-launch v4l2src ! "video/x-raw-yuv,width=320,height=240" \
  ! jpegenc ! multipartmux ! udpsink host=192.168.1.2 port=4951
```

Listing 2: Mit dem gst-launch Befehl werden mit der Linux-Shell die Kamerabilder als komprimiertes MJPEG an den PC mit der IP 192.168.1.2 gesendet und können dort mit dem VLC über die Adresse udp://@:4951 abgespielt werden

geprüft wird. Die Capabilities eines MPEG4-Dekodier-Elementes sind z. B. die Entgegennahme von komprimierten MPEG4-Videodaten am Quell-Pad und die Ausgabe des dekomprimierten Videos an der Senke. Das in der Programmiersprache C geschriebene GStreamer wird mit den Kommandos aus Listing 1 installiert.

In der hier zum Streaming verwendeten Pipeline (Abb. 5) werden die Kamerabilder als komprimiertes Motion-JPEG-Video (MJPEG) über einen per USB angeschlossenen W-LAN-Adapter an einen PC gesendet. Das Komprimieren der Daten in MJPEG und Übertragen der Videodaten über Netzwerk wird durch die Elemente *jpegenc, multipartmux* und *udpsink* durchgeführt. Das resultierende Video wird per UDP an einen Empfänger-PC auf Port 4951 (Port 4951 UDP in Firewall öffnen) übertragen und mit einem Videoplayer wie *vlc* [8] über die Adresse *udp://@:4951* abgespielt.

Zur Ausführung der Pipeline wird der *gst-launch* Befehl verwendet (Listing 2), mit dem sich GStreamer-Pipelines ohne Schreiben von Quelltext ausführen lassen. Die einzelnen Elemente der Pipeline werden durch ein Ausrufezeichen getrennt. Um die CPU nicht unnötig stark zu belasten, wird das Quellelement *v4l2src* mit *video/x-raw-yuv, width=320, height=240* angewiesen, ein nur 320 × 240 Pixel großes Bild (YUV-Farbraum) zu senden. Eine Liste der verfügbaren Auflösungen lässt sich mit dem Befehl *v4l2-ctl --list-formats-ext* (Paket *v4l-utils*) erfragen.

Fortgeschrittene können auf dem Raspberry Pi mit dem *omxh264enc*-Element die Videobilder mit H.264 in Hardware komprimiert über einen RTSP-Server übertragen lassen. Dafür ist jedoch eine neue GStreamer-Version erforderlich [7, 3].

Um GStreamer und OpenCV mit dem Raspberry-Pi-Kamera-Modul anstatt einer USB-Webcam zu verwenden, muss das Kernel-Modul *bcm2835-v4l2* geladen werden. Bei Tests (im Mai 2014) funktionierte dieses Modul trotz aktueller Firmware und Kernel mit den Bibliotheken jedoch nicht.

Objektverfolgung mit OpenCV

Im nächsten Schritt soll die Kamera zur Objektverfolgung eingesetzt werden. Dazu wird der *Camshift*-Algorithmus aus OpenCV [10] verwendet.

OpenCV ist eine geschwindigkeitsoptimierte Bibliothek für die Bildverarbeitung auf Prozessoren. Die ursprünglich von Intel entwickelte Bibliothek steht heute unter einer Open-Source-Lizenz und wird von der Firma Willow Garage gepflegt, die für das Robot Operating System (ROS) bekannt ist. OpenCV verfügt über Schnittstellen zu mehreren Programmiersprachen, darunter C, C++, Python und Java und läuft auf mehreren Betriebssystemen wie Windows, Linux, Mac OS und Android. In OpenCV sind u. a. Algorithmen zur Mustererkennung, Gesichtserkennung und Objekterkennung in Bildern implementiert.

Der Camshift-Algorithmus

Zur Objektverfolgung wird der in OpenCV implementierte Camshift-Algorithmus (Continuously Adaptive Mean Shift), eine Erweiterung des Mean-Shift-Algorithmus, verwendet [4]. Beim Mean-Shift-Algorithmus wird von dem zu verfolgenden Objekt eine Liste mit allen im Bild vorkommenden Farben (Genauer: Histogramm) erstellt. In einem iterativen Verfahren wird dann in einem Suchfenster der Schwerpunkt der Verteilung der Farben aus dieser Liste berechnet und das Suchfenster in Richtung dieses Schwerpunktes verschoben (Abb. 6). Der Algorithmus bricht ab, sobald der Schwerpunkt des Suchfensters mit dem Schwerpunkt der Farbverteilung übereinstimmt.

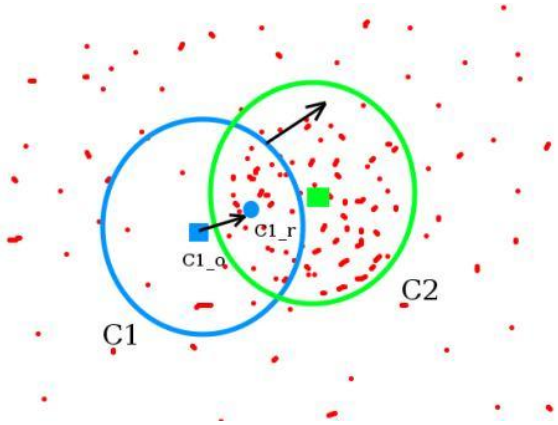


Abb. 6: Mean-Shift-Algorithmus: Im Suchfenster C1 wird der Schwerpunkt der Farbwerte berechnet und in Richtung des neuen Schwerpunktes als C2 verschoben (Bild: OpenCV.org)

Nachteil des Mean-Shift-Algorithmus ist die statische Größe des Suchfensters, weswegen Größenänderungen, z. B. bei einem sich nähernden Objekt, nicht berücksichtigt werden können. Beim Camshift-Algorithmus wird zusätzlich die Verteilung der Pixel betrachtet und damit die Größe und Ausrichtung des Objektes mit einbezogen. Das Ergebnis des Algorithmus ist eine Ellipse.

Objektverfolgung mit Robo Explorer

Zur Anwendung von Camshift wird zuerst OpenCV installiert und das Camshift-Beispielprogramm *camshift.py* in das aktuelle Anwendungsverzeichnis extrahiert (Listing 3).

Abb. 7: OpenCVs Camshift hat den Ball im Bild lokalisiert

Das Beispielprogramm enthält den für den Camshift-Algorithmus notwendigen Python-Programmcode. Wer mag kann das Programm ausführen und etwas mit dem Algorithmus herumspielen. Gestartet wird Camshift im OpenCV-Fenster durch Klicken und Ziehen eines Rechtecks mit der Maus um das zu verfolgende Objekt herum. Das zu verfolgende Objekt (z. B. ein Ball) sollte sich dabei farblich vom Hintergrund abheben. Je nach Kamera kann eine unterschiedlich lange Latenz des Kamerabildes bis zur Darstellung auf dem Monitor entstehen. OpenCV wird nun das Objekt mit einer Ellipse markieren (Abb. 7). Ein einfacher Klick mit der Maus beendet den Algorithmus. Die Interaktion

```
$ sudo apt-get install python-opencv opencv-doc
$ gzip -d /usr/share/doc/opencv-doc/examples/python/camshift.py.gz -c \
  > camshift.py
```

Listing 3: Installation vom OpenCV und Extraktion des Camshift-Beispielprogramms.

mit dem Raspberry Pi oder BBB kann entweder über einen angeschlossenen Monitor mit Tastatur und Maus oder mittels SSH X-Forwarding erfolgen.

Damit der Robo Explorer automatisch einem Objekt folgt, wird im nächsten Schritt das Beispielprogramm in den eigenen Programmcode (Listing 6) eingebunden:

- Zuerst wird die Bildgröße auf 320 × 240 Pixel gesetzt, um die von Camshift benötigte Rechenzeit der CPU zu reduzieren, und das Camshift-Beispielprogramm in einen eigenen Thread gestartet.
- Das Ergebnis des Camshift-Beispielprogrammes (Ellipse *track_window*) wird periodisch abgefragt und der Funktion *follow()* übergeben.
- Die ausbaufähige Funktion *follow()* nimmt als Argumente das Ergebnis des Camshift-Algorithmus als Koordinaten mit Breite und Höhe der Ellipse entgegen und berechnet damit die Sollgeschwindigkeit für den Antrieb:
  - Ist das zu verfolgende Objekt kleiner als 60 Pixel, fährt der Roboter mit halber Kraft vorwärts.
  - Wandert das Objekt aus der Bildmitte heraus, wird der Roboter entsprechend gedreht.
- Die [2] entnommene Methode *setSpeed()* steuert die Geschwindigkeit des Robo Explorers.

Wer keine Möglichkeit für die grafische Ausgabe und Benutzereingaben hat, kann den Camshift-Algorithmus mit dem Code aus Listing 4 starten, der das zu verfolgende Objekt in einem 40 × 40 Pixel großen Bereich in der Bildmitte erwartet. In der Datei *camshift.py* sollten dazu jedoch zusätzlich die Zeilen mit *cv.NamedWindow(), cv.ShowImage()* und *cv.SetMouseCallback()* auskommentiert werden.

Wer Probleme hat, die Bildmitte zu finden, kann die GStreamer-Pipeline aus Listing 2 mit dem *rsvgoverlay*-Element erweitern und damit eine Zielmarkierung mittels SVG-Vektorgrafik einzeichnen (Listing 5).

```
1   self.camshift_demo.drag_start = (140, 100)
2   self.camshift_demo.selection = (140, 100, 40, 40)
3   sleep(1) # wait at least one frame
4   self.camshift_demo.drag_start = None
5   self.camshift_demo.track_window = (140, 100, 40, 40)
```

Listing 4: Programmcode zum Start des Camshift-Algorithmus ohne Maus

```
$ gst-launch v4l2src ! "video/x-raw-yuv,width=320,height=240" \
  ! ffmpegcolorspace ! rsvgoverlay \
  data='<svg xmlns="http://www.w3.org/2000/svg" version="1.1">'\
  '<rect x="140" y="100" width="40" height="40" stroke="red"'\
  ' stroke-width="3" fill="none"/></svg>' \
  ! ffmpegcolorspace ! jpegenc ! multipartmux \
  ! udpsink host=192.168.1.2 port=4951
```

Listing 5: Senden der Kamerabilder mit eingezeichneter Zielmarkierung in der Bildmitte.

Zusammenfassung

Ziel dieses Beitrags war es, euch einen Einblick in die Möglichkeiten zu vermitteln, die einem die Integration von ARM-Boards wie dem Raspberry Pi oder dem Beaglebone Black in das eigene Modell bietet. Die Spannungsversorgung der Boards erfolgt dabei entweder über einen USB Battery Pack oder einen Spannungsregler wie den LM2576. Zur Visualisierung der Kamerabilder eignet sich das Framework GStreamer, das die Bilder mit der hier vorgestellten Pipeline als MJPEG über W-Lan an einen entfernten PC überträgt. Als Einstieg in die Bildverarbeitung wurde der Camshift-Algorithmus aus der OpenCV-Bibliothek vorgestellt, mit dem ein Modell wie der Robo Explorer automatisch einem Objekt folgen kann.

Wer tiefer in das Gebiet einsteigen möchte, kann versuchen, beide Verfahren zu kombinieren, also das Resultat des Camshift-Algorithmus als Bild über W-Lan zu übertragen. Dazu reicht eine per *gst-launch* gestartete GStreamer-Pipeline allerdings nicht mehr aus.

Literatur

[1] Andresen, Erik: *ROBO Interface Bibliothek für Unix-artige Systeme*.

[2] Andresen, Erik: *The fischertechnik Interface for the Rest of us*. In: ft:pedia 2/2012 (2012), S. 32-38.

[3] Bock, Matthias: *Hardware-accelerated video playback on the Raspberry Pi*.

[4] Bradski, Gary R.: *Computer Vision Face Tracking For Use in a Perceptual User Interface*. Forschungsbericht, Intel Corporation 1998.

[5] Carnegie Mellon University: *Using PocketSphinx with GStreamer and Python*.

[6] Embedded Linux Wiki: *RPi compatible USB Webcams*.

[7] gkiagia: *gst-omx now on 1.0 Branch*.

[8] VideoLAN-Team: *VLC media player*.

[9] Walthinsen, Erik u. a.: *Multimedia-Framework Gstreamer*.

[10] Willow Garage: *OpenCV: Open Source Computer Vision Library*.

```
#!/usr/bin/env python
# -*- coding: iso-8859-15 -*-

import thread
import cv2.cv as cv
from time import sleep
from camshift import *
from robointerface import *

WIDTH = 320
HEIGHT = 240

def setSpeed(self, l, r):
    self.speed = (l, r)
    if l > 0:
        self.SetMotor(1, 'l', l)
    elif l < 0:
        # left reverse
        l*=-1
        self.SetMotor(1, 'r', l)
    else:
        self.SetMotor(1, 's', l)

    if r > 0:
        self.SetMotor(2, 'l', r)
    elif r < 0:
        # right reverse
        r*=-1
        self.SetMotor(2, 'r', r)
    else:
        self.SetMotor(2, 's', r)
RoboInterface.setSpeed = setSpeed

def follow(ri, target_x, target_y, target_width, target_height):
    left_speed = 0; right_speed = 0
    if target_width < 60:
        left_speed = 3; right_speed = 3
    diff_x = target_x - WIDTH/2
    if diff_x < -20:
        right_speed+=1
    elif diff_x > 20:
        left_speed+=1
    print "x=%d, width=%d, speed=(%d, %d)"\
            % (target_x, target_width, left_speed, right_speed)
    ri.setSpeed(left_speed, right_speed)


if __name__ == "__main__":
    demo = CamShiftDemo()
    cv.SetCaptureProperty(demo.capture, cv.CV_CAP_PROP_FRAME_WIDTH, WIDTH)
    cv.SetCaptureProperty(demo.capture, cv.CV_CAP_PROP_FRAME_HEIGHT, HEIGHT)
    thread.start_new_thread(demo.run, ())

    # Robo Interface
    ri = RoboInterface()
    # Intelligent Interface
    # ri = RoboInterface(serialDevice="/dev/ttyUSB0",
    #        SerialType=RoboInterface.FT_INTELLIGENT_IF)
    ri.setSpeed(0, 0)

    while True:
        # follow only when all elements of track_window are > 0
        if demo.track_window and all(i > 0 for i in demo.track_window):
            follow(ri, *demo.track_window)
        else:
            ri.setSpeed(0, 0)
        sleep(0.1)
```

Listing 6: Python Programm für den Robo Explorer zur Objektverfolgung mit OpenCV (siehe Text)

Computing

Schau‘ mir in die Augen, Kleiner! Kamera am TX-Controller

Marco Ahlers

Wer nicht auf den neuen fischertechnik-Controller warten möchte, der kann auch dem TX-Controller sehen und sprechen beibringen: Dazu braucht ihr wenig mehr als ein Arduino-Board, einen Raspberry Pi und eine handelsübliche Webcam.

Einführung

In Ausgabe 1/2014 der ft:pedia habe ich vorgestellt, wie ein Arduino an den TX-Controller angeschlossen werden kann, um über das I²C-Protokoll Sensorenwerte im Arduino zu berechnen und zu filtern [1].

Heute gehe ich einen Schritt weiter. Mit einem an den Arduino angeschlossenen Raspberry Pi und einer wiederum dort angeschlossenen Standard-Webcam soll ein Roboter farbige Bälle suchen und finden, sich dorthin bewegen und seinen Kurs in Abständen korrigieren. Außerdem soll er Statusmeldungen laut sprechen und so über die eigenen Erkenntnisse informieren.

Dem Roboter möchten wir Befehle über Tafeln mit Barcodes erteilen, die wir vor die Kamera halten. Durch das Lesen des Barcode-Befehls soll der Roboter dann wissen, dass er losrollen soll, um nach einem roten Ball zu suchen. Mit einem Barcode soll auch der Shutdown eingeleitet werden.

Materialbedarf

Für unseren gehorsamen seh- und sprachbegabten Roboter benötigen wir:

- einen TX-Controller,
- ein Arduino Uno Breadboard, Steckverbindungen,
- optional mehrere LED-Lichtbausteine zur besseren Ausleuchtung bei Dunkelheit,
- einen Raspberry Pi, im Folgenden kurz RPi genannt, inkl. SD Card, Maus, Tastatur und Monitoranschluss (HDMI-an-DVI-Kabel),
- eine Webcam (z. B. Logitech C525 HD-Webcam, muss mit Raspberry Pi Raspbian kompatibel sein [6]),
- ein Emic 2 Text2Speech-Modul (für die Sprachausgabe),
- einen Mini-Aktiv-Lautsprecher, z. B. Boombox (optional) und
- ein USB Kabel.

Seit der letzten ft:pedia kann der TX-Controller Befehle an den Arduino über die I²C-Schnittstelle weitergegeben, um nach Bedarf Sensorwerte vom Arduino zurück zu erhalten [1].

Diesmal erhält wieder zunächst der Arduino Befehle und arbeitet dann entweder selbst Aufgaben ab oder gibt seinerseits Befehle an den Raspberry Pi weiter, um z. B. ein Bild mit der angeschlossenen Webcam zu machen, auszuwerten und Positionsdaten an den Arduino zurück zu liefern. Der Arduino gibt diese Daten dann wiederum aufbereitet an den TX-

Controller weiter. Je nach Kommando des TX-Controllers wird vom Arduino der Sprachchip des Emic-Moduls aktiviert, um bestimmte Sätze oder Zahlenwerte zu sprechen.

Die hier dargestellten Methoden sollen euch als Beispiel und Anregung für eure eigenen Projekte dienen. Deswegen sind auch keine konkreten Bauanleitungen vorhanden, und das ‚Such-finde-und-fahr-da-mal-hin-RoboPro-Programm' ist auch nicht optimiert. Denn fischertechnik soll ja vor allem die eigene Kreativität fördern.

Bau des Roboters

Dieser Beitrag soll das Grundwissen dafür vermitteln, wie ein Roboter seine Umwelt erkennen kann. Der Bau des Roboters unterliegt daher ganz allein eurer Phantasie. Letztlich basiert er aber im wesentlichen auf Modellen wie dem ‚Rasenmäher' von fischertechnik. Es wird allerdings mehr Platz für die zusätzlichen Bauteile (den Arduino, den Raspberry Pi, die Kamera usw.) benötigt. Daher habe ich den Roboter auf die beiden Grundplatten gebaut, die beim ‚Robo-Flipper' benutzt werden.

Die von mir verwendete Kamera ist eine Logitec Webcam auf einer selbst gebauten Kameradrehmechanik; Abb. 1 zeigt sie aus der Nähe.

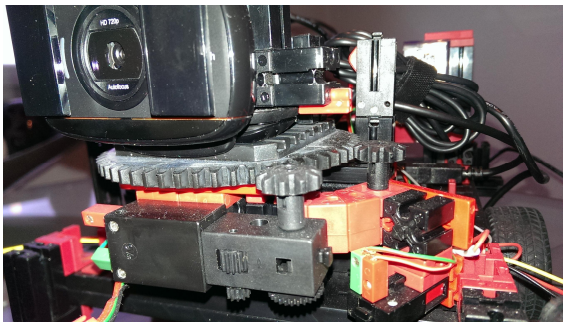


Abb. 1: Drehmechanik der Webcam

Die Drehmechanik besteht aus einem Drehkranz, dem Antriebsmotor mit Getriebe sowie einem Taster mit 4er-Impulsgeber. Motor und Impulsgeber sind direkt unterhalb des Drehkranzes montiert.

Zusätzlich habe ich einen stärkeren Akku benutzt, ein 10 Ah-Smartphone-Akkupack. Er kann den TX-Controller über eine Buchse mit 9 V versorgen und über den USB-Anschluss den Raspberry Pi. Der Arduino wird in diesem Fall über die USB-Schnittstelle vom Raspberry Pi versorgt.

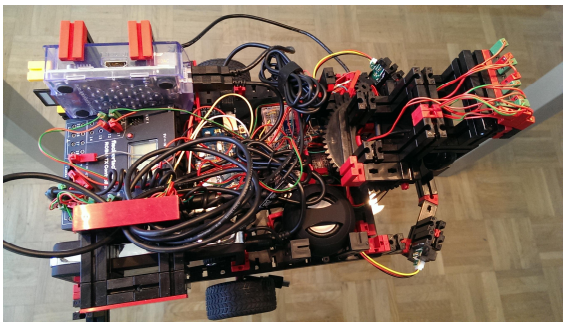


Abb. 2: Roboter von oben

In Abb. 2 gut zu sehen ist auch der runde Aktivlautsprecher (rechts unten). Links unten erkennt man den Akkupack, dahinter den TX-Controller, der Raspberry Pi und in der Mitte (unter den Kabeln) den Arduino. Gerade noch zu erhaschen ist auf dem Steckbrett der Emic-2-Text2Speech-Chip; rechts befindet sich die Kamera-Drehmechanik aus Abb. 1.

Verdrahtung

Die Verbindungen sind recht einfach: Der Arduino wird via USB an den Raspberry Pi angeschlossen, ebenso die Webcam. Der Arduino wird wie in der vorigen Ausgabe beschrieben per I²C an den TX-Controller angeschlossen. Weitere Sensoren sind zunächst nicht nötig.

Die Stromversorgung könnt ihr frei wählen. Ich habe, wie oben beschrieben, einen leistungsstarken Power Pack für die mobile Aufladung von Smartphones benutzt. Dieser hat sowohl USB-Anschlüsse als auch Plugs, die ihr mit dem 9 V Anschlusskabel des fischertechnik-Trafos mit dem TX-Controller-Eingang

(9 V) verbinden könnt. Er hält ziemlich lange.

Der Emic-2-Sprachchip

Eine prima Sache mit einer erstaunlichen Sprachqualität ist der Emic-2-Sprachchip. Er ist im einschlägigen Elektronikfachhandel erhältlich. Zwar ist er nicht unbedingt notwendig, aber er erleichtert das Überwachen des Programmablaufes erheblich. Der Roboter hält uns damit via Sprachausgabe über erfolgte Programmschritte auf dem Laufenden.

Bei dem vorliegenden Programmablauf ist der Anschluss von Sensoren zunächst nicht notwendig, daher kommt der Arduino mit ein paar Drahtverbindungen zum Sprachchip aus (Abb. 3).

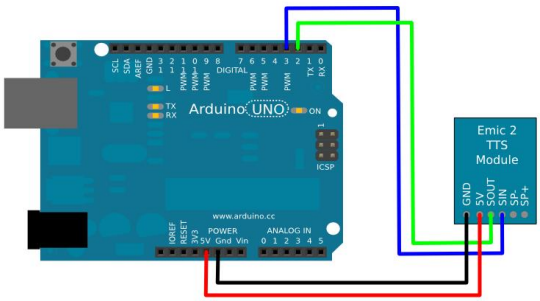


Abb. 3: Anschluss des Emic 2

Bei der I²C-Übergabe der entsprechenden ‚Sprechbefehle' vom TX-Controller wird hier übrigens nicht nur das Register benutzt, sondern auch der Dateninhalt des nachfolgend gesendeten Bytes. In RoboPro macht das das Unterprogramm „Speak": Es bekommt die Variable jeweils je nach gewünschtem Satz mitgeliefert. Abhängig von deren Wert wird der entsprechende Satz aus dem Array `expressions[]` herausgesucht. Der erste Satz „Running Startup..." hat den Index 0, der zweite 1 usw.

```
//Definition der vom Emic 2
Sprachchip zu sprechenden Saetze

char* expressions[]={
  "Running Startup Procedures now.
Please wait until I confirm
complete startup.",
  "Startup is completed
successfully.",
  "Oooooh! I do not see any space
ahead. I go backwards.",
  "YEAH! I found a ball. I turn to
his direction now.",
  "I lost the Ball.",
  "I try to detect a Barcode
now.",
  "I can't see any Barcode.",
  "Barcode detected."
  "Please help me. I can't move.",
  "I will scan now the front
horizon for balls. Please wait
until I confirm full
scan.","Horizon Scan completed.",
  "I will move now forward until
pathway is blocked.",
"Pathway is blocked.",
"Serial Flushed",
"Raspberry Pie active.",
"Raspberry Pie is not
responding.",
"Nothing to report"};
```

Listing 1: Sätze für den Emic-2-Sprachchip

Der Barcodeleser

Ist auf einem von der Webcam aufgenommenen Bild ein Barcode erkennbar, so soll dieser ausgewertet werden. Das Programm für den Arduino und den Raspberry Pi enthält den nötigen Programmcode dafür; in diesem Beitrag gehe ich nicht weiter darauf ein. Ihr könnt den Barcodeleser dafür nutzen, dem Roboter Befehle zu erteilen, indem ihr einen Barcode vor die Linse haltet. In einem Dropbox-Ordner habe ich euch einige Barcodes zum Ausdrucken vorbereitet.

Abb. 4: QR-Code des Shutdown-Befehls

Mit derselben Technik könntet ihr an einer Wand aufgehängte Barcodes nutzen, um dem Roboter die Orientierung zu erleichtern.

Programmierung des Raspberry Pi

Den Raspberry Pi müsst ihr für die Programmierung zunächst mit Tastatur, Maus und Monitor versehen. Dazu habe ich ein HDMI-zu-DVI-Kabel an meinen Monitor angesteckt und kann so mittels Taste am Monitor zwischen PC und Raspberry Pi hin und her schalten. Fertig programmiert läuft der Raspberry Pi dann aber unabhängig von Tastatur und Monitor auf dem Roboter.

Ich habe alle im Folgenden vorgestellten Programme in einer Dropbox für euch zum Download bereitgestellt.

Autostart des Raspberry Pi Programmes

Damit nach dem Start des Raspberry Pi das gewünschte Programm automatisch startet, muss es im Autostartordner abgelegt werden. Gebt dazu in das Adressfeld des Dateimanagers des Raspberry Pi ein: `/home/pi/.config`

Falls kein Autostart-Ordner vorhanden ist, erstellt dort einen. Danach öffnet ihr den Idle- oder Leafpad-Editor und erfasst das folgende Programm. Es wird dann mit der Endung `.desktop` im Autostart Ordner gespeichert:

```
[Desktop Entry]
Encoding=UTF-8
Type=Application
Name=Jimmy Brain Autostart
Exec=sudo python
/home/pi/jimmy_brain_rpi_140401.py
StartupNotify=false
Terminal=false
Hidden=false
```

Listing 2: Autostart-Skript

Unter `Exec=` steht der Ort und der Name des Programms. Beides müsst ihr ändern, wenn ihr das Programm nicht im Hauptordner pi gespeichert habt.

Shutdown des Raspberry Pi

Es ist wichtig für die Datenintegrität auf der Speicherkarte des Raspberry Pi, dass er ordentlich heruntergefahren wird. Dies habe ich durch die Shutdownsequenz realisiert, vom TX-Controller über einen Befehl via Arduino an den Raspberry Pi weitergegeben wird:

```
#Shutdown des Rpi, wenn 3 gesendet
wurde:
  if (input=="3"):
    ser.write("3$")
    command =
      "/usr/bin/sudo/sbin/shutdown
      -h now"
    import subprocess
    process =
      subprocess.Popen(command.
      split(),
      stdout=subprocess.PIPE)
    output =
      process.communicate()[0]
    print output
```

Listing 3: Shutdown-Sequenz für den Rpi

Kommunikation Raspberry Pi und Arduino

Der Rpi ist mit dem Arduino über den USB-Anschluss verbunden. Die Daten werden dabei über die serielle Schnittstelle Byte für Byte übertragen. Will man also z. B. das Wort „TEST“ schicken, so wird nicht das Wort in einem Rutsch, sondern „T“ dann „E“, dann“S“ und zum Schluss „T“ übermittelt.

Der Datenempfänger möchte aber nun das ganze Wort als String weiterverarbeiten. Dazu muss das Wort aus den Buchstaben wieder zusammengebaut werden. Dafür wiederum ist es nötig, dass der Datenempfänger weiß, wann das Wort beendet ist. Der Datensender muss daher entweder die Wortlänge mitteilen oder ein Schlusszeichen senden. In meinem Programm sende ich ein Dollarzeichen („$“) als Schluss-

signal, da dies in keinem meiner Wörter verwendet wird.

Der Sender sendet also: „T“, „E“, „S“, „T“, „$“. Der Empfänger hängt die gesendeten Buchstaben in einem String aneinander, bis das „$“ Zeichen kommt. Ein String ist ein Array aus Zeichen, jedes mit einem Index versehen. Dies wird im Arduino Programm genutzt, um mit `inCount` den jeweiligen Index zu bestimmen.

Beispiel: Dem Arduino soll mit dem Code „255“ mitgeteilt werden, dass kein Ball gefunden wurde:

```
#wenn kein Blob gefunden wurde,
sende 255$ an die serielle
Schnittstelle
    else:
        ser.write("255$")
```

Im Arduino startet das aufnehmende Programm mit der Entgegennahme des Befehls vom TX-Controller, dass jetzt die Ballposition ermittelt werden soll:

```
// Hier folgt die Abfrage der
seriellen Schnittstelle, um die
Variable POS des Balls zu erhalten
// POS=255 bedeutet kein Ball
(Blob) erkannt.
// Wenn Ballposition gewünscht,
wird der Befhel 5 vom Tx-
Controller gesendet
  if(read_register == 0x05){
    int POS=read_ball();
    // hier wird die Funktion für
die Variable „read ball“
aufgerufen
    // emicSerial.print('S');
    // emicSerial.print(POS);
    // emicSerial.print('\n');
    Wire.write(POS);
    // respond with message of 10
bytes
  }
```

Listing 4: Abfragen der seriellen Schnittstelle

Hier folgt der Teil des Arduino-Programms, das die Daten vom Raspberry Pi auswertet:

```
// Definition des „Ende-Zeichens“
($=ASCII 36):
#define INTERMINATOR 36

// Lesen der x-Koordinate eines
gefundenen Balls vom Raspberry Pi#
int read_ball()
// Funktion zur Bestimmung der
Variable der Ballposition
{
  inCount=0;   // Der Indexzähler
wird auf Null zurückgesetzt
  while (Serial.available()>0){
  inString[inCount]=Serial.read();
   // Schleife, die das jeweils
nächste an der seriellen Schnitt-
stelle anstehende Zeichen liest
und dem aktuellen „inCount“ Index
in den String einfügt.

if(inString[inCount]==INTERMINATOR
) break;  // falls das gelesene
zeichen „$“ ist, wird die Schleife
unterbrochen.
  InCount++; // der Index wird um
1 hochgezählt
  }
// aus dem String wird eine ganze
Zahl gemacht
  inString[inCount]=0;
  int szahl=atoi(inString);

  return(szahl); // der berechnete
Wert wird an die Variable „read
Ball“ zurückgegeben
}
```

Listing 5: Auswertung der empfangenen Daten

Programmablauf

Der Roboter – ich habe ihn ‚Jimmy‘ getauft – soll nach dem Start nicht herumfahren, sondern nur von seinem jeweiligen Standort aus den vorderen Horizont, also 180° durch einen Kameraschwenk in 12 Schritten nach dem Ball absuchen. Findet er keinen Ball, so könnte man ihn um 180° drehen (nicht im RoboPro-Programm enthalten) und die Suche wiederholen. Findet er den Ball, soll er die Kamera wieder nach vorn richten, sich in Richtung des Balles drehen, ein Stück vorwärts fahren, die Richtung neu justieren und erneut etwas nach vorne auf den Ball zu fahren. Die Infrarotsensoren und den Ultraschallsensor benötigen wir dafür nicht.

Das Programm, das im Roboter abläuft, muss dazu die folgenden Schritte ausführen:

- Nach dem Anschalten des Roboters und dem Starten des RoboPro-Programms prüft der Roboter, ob sein System vollständig hochgefahren ist.
- Der Arduino meldet nach seinem eigenen Start „Jimmy is Ready!“ über den Lautsprecher.
- Danach prüft der Roboter, ob der Raspberry Pi hochgefahren ist und das Programm `Jimmy Brain rpi.py` bereits läuft. Dies wird durch das Senden des Befehls ‚2‘ an den Raspberry Pi erreicht.
- Läuft das Programm, sendet der Raspberry Pi ‚2$‘ zurück. und der Arduino spricht über den Lautsprecher „Raspberry Pi active“.
- Der Roboter wartet jetzt auf die Barcodes, damit er weiß, was er machen soll. Nachdem er den „Start“-Barcode vor der Kamera erkannt hat, geht es weiter.
- Das RoboPro-Programm lässt nun die Kamera um 90° nach rechts drehen, um dann in 12 Schritten um 180° nach links zu schwenken. Bei jeder Position wird die LED-Lichtleiste angeschaltet, ein Foto gemacht, die Lichtleiste wieder ausgeschaltet und versucht, einen roten Ball im Bild zu entdecken. Dazu sendet RoboPro einen Befehl ‚5‘ an den Arduino, dieser wiederum sendet den Befehl weiter an den Rpi. Dort erfolgt dann der Programmablauf zu „Find Blob“.
- Wurde vom Rpi ein roter Ball, also ein Blob gefunden, meldet der Rpi an den Arduino die *x*-Position der Mitte des Blobs. Der Arduino gibt den Wert an den TX-Controller weiter. Gleichzeitig meldet der Arduino über den Sprachchip „I found a Ball!“
- RoboPro berechnet aus den *x*-Koordinaten des Balls im geschossenen Foto die nötige Drehweite und Richtung des Roboters und startet die Motoren. Nachdem der Roboter zum Ball hin ausgerichtet wurde, wird erneut der Befehl ‚5‘ zur Aufnahme eines Bildes über den Arduino an den Rpi gesendet und mit Hilfe des zurückgelieferten *x*-Wertes die Richtung des Roboters korrigiert.
- Der Roboter fährt jetzt ein Stück in Richtung des Balls, hält an und schießt erneut ein Foto, liefert den *x*-Wert zurück, korrigiert die Richtung, rollt weiter und hält dann an.

Das soll dann zunächst mal genügen. Wenn ihr den Roboter ausschalten wollt, dann haltet den Shutdown-Barcode vor die Kamera.

SimpleCV

Eins vorweg: Es gehören ein wenig Einarbeitung und viele Stunden des Lernens dazu, will man einem Roboter das Sehen beibringen. Schnellere Erfolge verspricht hier sicherlich das angekündigte Discovery Set von fischertechnik. Aber es macht Spaß, sich durch die Materie zu wühlen und die Erfolgserlebnisse sind großartig, wenn die Kamera den Ball erkannt hat und der Roboter „I found the Ball!“ meldet.

Zunächst muss der Raspberry Pi zum Laufen gebracht werden. Allen Neueinsteigern empfehle ich dazu das Buch „Raspberry Pi für Einsteiger“ [2]. Der Raspberry Pi sollte das Betriebssystem Raspbian funktionsfähig installiert haben. Es ist wichtig, dass ihr euch zunächst etwas mit dem Raspberry Pi beschäftigt und euch ein wenig damit auskennt – das setze ich hier voraus, denn dieser Beitrag kann das nicht leisten (mehr z. B. in [3]).

Das Sehen selbst wird durch die OpenCV-Bibliotheken erreicht, unterstützt durch die SimpleCV-Erweiterung, die einige Tasks aus OpenCV in einfach nutzbaren Befehlen

zusammenfasst. CV steht für *Computer Vision* [4]. Damit ist die Auswertung und Bearbeitung von Bilddateien gemeint. Dabei können z. B. Farbkanäle isoliert und Bildausschnitte so gewählt werden, wie man dies von Bildbearbeitungssoftware her kennt.

Darüber hinaus sind aber auch Funktionen enthalten, um z. B. zusammenhängende Farbflächen zu finden, deren (x, y)-Position im Bild zu bestimmen und die Größe zu messen. Dies werden wir nutzen, um einen Ball anhand seiner Farbe zu finden und dabei jeweils nur den größten farbigen Fleck im Bild berücksichtigen.

Da dies eine Menge Rechenaufwand benötigt und ein richtiger Computer dazu erforderlich ist, kann der Arduino dies nicht alleine leisten. Deswegen nutzen wir den Raspberry Pi dafür.

Für die Installieren von SimpleCV auf dem Rpi gibt es eine gute Anleitung. Zusätzlich benötigt ihr noch eine Barcode-Bibliothek; die installiert ihr Euch mit der folgenden Befehlsfolge:

```
sudo apt-get install zbar-tools
```

Und nicht zuletzt wird auch ein QR-Code-Generator benötigt, den ihr euch frei aussuchen könnt – geeignet ist z. B. dieser hier.

Bildauswertung

Ganz wichtig bei unserem Vorhaben ist das Wissen darüber, an welcher Koordinate in einem Bild sich ein bestimmtes Objekt befindet.

Ein Pixel kann sich dabei in einem zweidimensionalen Koordinatensystem auf einer bestimmten (x, y)-Position befinden. Als Punkt (0, 0) legen wir die linke obere Ecke fest. Abb. 5 zeigt ein Beispiel für ein 8×8-Pixel-Bild. Angenommen, der darin erkennbare ‚Kreis' sei der gefundene Ball auf dem Bild. Die rot gekennzeichnete Mitte, genauer gesagt das mittlere Pixel des „Blobs", hat die Koordinate (3, 3). Der x-Wert 3 wird an den Roboter gemeldet und dieser müsste jetzt ganz leicht nach links schwenken, um den Ball in die Mitte vor sich zu bekommen.

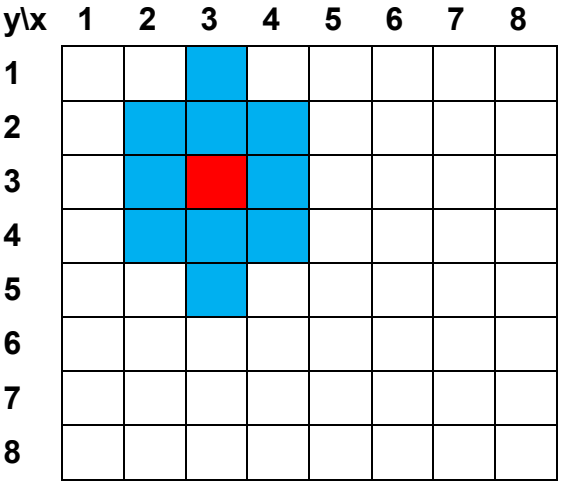


Abb. 5: 8×8-Pixelbild mit Objekt

In Wirklichkeit haben die Bilder, die wir benutzten, eine x-Auflösung von 160 Pixeln – aus Performancegründen nicht mehr, aber das genügt für die Orientierung des Roboters vollkommen.

Die folgenden SimpleCV-Funktionen werden für das Programm benutzt:

- `img=get.image`: nimmt ein Bild mit der Kamera auf.

Abb. 6: Bild der Kamera

- `hue.distance()`: gibt ein Bild wieder, das nur noch Graustufen enthält. Pixel, die der gesuchten Farbe am nächsten liegen, werden schwarz, die am weitesten entfernten weiß; dazwischen in Graustufen dargestellt.

Abb. 7: Graustufenbild

- `Binarize()`: Reduziert das Graustufenbild auf schwarz/weiß.

Abb. 8: Schwarz-Weiß-Bild des Blob

- `find blob`: Ein Blob ist ein Fleck, also genau genommen eine zusammenhängende Fläche einer annähernd gleichen Farbe. In unserem Beispiel ist der Blob die nach `Binarize()` übrig gebliebene weiße Fläche.
- `Sort area`: Wenn ein solcher Fleck gefunden wurde, soll nur der größte berücksichtigt werden. Damit wird störendes „Rauschen“ ausgeblendet, wie kleinere Flecken der gleichen Farbe, die aber zu klein sind, um ein Ball sein zu können.
- `coordinates`: Ergibt die Koordinaten der Mitte des Flecks, also ob sich der Ball rechts, links oder genau auf Kurs des Roboters bzw. der gegenwärtigen Kameraposition befindet.

Blobs

Blobs sind Flecken in einem Bild, die eine zusammenhängende Farbe haben. Um sie zu finden wird zunächst das Bild um alle anderen Farben reduziert und alle Farbtöne der gesuchten Farbe innerhalb einer bestimmten, einstellbaren Toleranzgrenze vom restlichen Bild separiert.

Dabei ist es wichtig genau zu definieren, welche Farbe der Roboter suchen soll. Dies geschieht ganz einfach durch die Angabe der RGB-Werte. In unserem Beispiel wird noch der HUE-Wert berechnet; das macht das Ergebnis unempfindlicher gegen schlechte Lichtverhältnisse:

```
blue_distancepre =
img.hueDistance((190, 50, 50))
```

Mit den RGB-Werten müsst ihr ein wenig experimentieren, vielleicht nehmt ihr erst einmal ein Foto eures Suchobjektes auf und bestimmt dann die RGB-Werte mit einem Bildbearbeitungsprogramm (Pipettentool).

Das RoboPro-Programm steht für euch ebenfalls in einem Dropbox-File zum Download bereit; dort finden sich auch zwei Videos des Roboters.

Und jetzt wünsche ich euch viel Spaß beim Tüfteln, Ausprobieren und dem Liebhaben eures neuen Freundes.

Referenzen

[1] Marco Ahlers: *Arduino mit dem TX verbinden*. ft:pedia 1/2014, S. 31-38.

[2] Matt Richardson, Shawn Wallace: *Raspberry Pi für Einsteiger*, O'Reilly Verlag, 2013

[3] Michael Weigend: *Raspberry Pi programmieren mit Python*, mitp Verlag, 2013

[4] Kurt Demaagd, Anthony Oliver, Nathan Oostendorp, Katherine Scott: *Practical Computer Vision with SimpleCV*, O'Reilly Verlag, 2012

[5] Fabian Kainka: *Handbuch*, Arduino Starterpaket, Franzis-Verlag, 2013.

[6] Embedded Linux Wiki: *RPi compatible USB Webcams*.

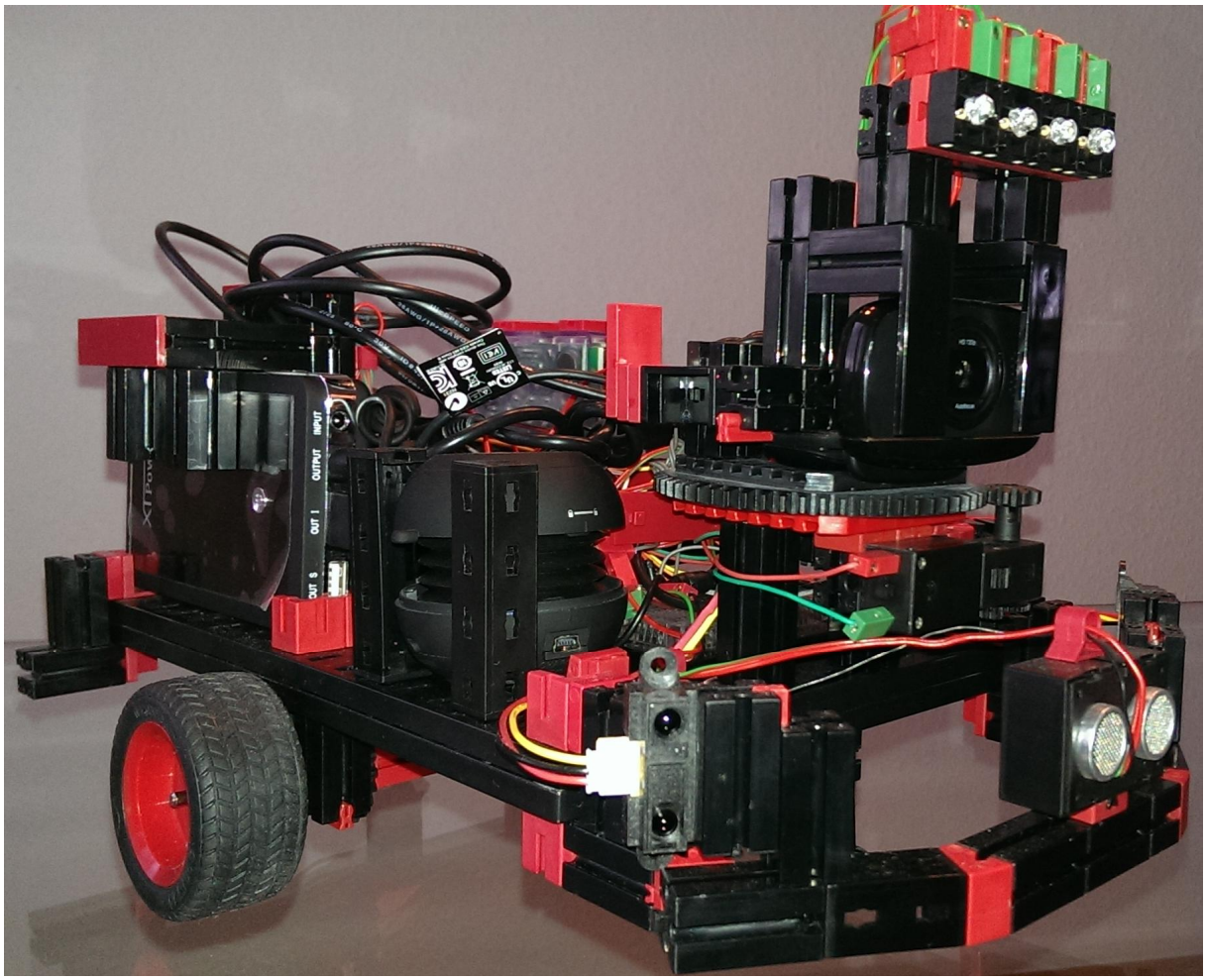


Abb. 9: Roboter mit TX-Controller, LED-Leiste, Webcam, Battery Pack, Raspberry Pi und Arduino

Computing

I²C mit dem TX – Teil 10: Kompass-Sensoren

Dirk Fox

Inzwischen haben wir einige I²C-Sensoren vorgestellt, die sich direkt an den TX anschließen und mit Robo Pro nutzen lassen – und spannende Einsatzmöglichkeiten eröffnen. Ein echter „Klassiker" fehlte bisher in der Reihe: der Kompass-Sensor.

Hintergrund

Für viele Anwendungen der Robotik wird eine möglichst genaue Richtungsangabe benötigt, damit sich beispielsweise ein Roboter in unbekanntem Gelände oder zwischen Hindernissen hindurch in Richtung eines Ziels bewegen kann. Zwar liefert auch ein GPS-Sensor, wie der, den wir in Teil 6 unserer I²C-Serie vorgestellt haben [1], Richtungsinformationen – jedoch sind die nicht besonders genau. Mit einem guten Kompass-Sensor lässt sich die GPS-Navigation erheblich verbessern – dazu mehr in einem eigenen Beitrag.

Die heute verbreiteten Kompass-Sensoren bestimmen die Bewegungsrichtung wie eine Magnetnadel anhand des Magnetfelds der Erde. Sie liefern die Abweichung von der korrekten Nord-Süd-Ausrichtung in der Regel entsprechend der klassischen Kompassrose als einen Wert zwischen 0 und 360°.

Bei der Nutzung von Kompass-Sensoren ist zu beachten, dass sie sich wie die Kompassnadel nicht an den geographischen, sondern an den magnetischen Polen ausrichten (*Deklination* [2]). Außerdem kann die Richtungsangabe durch Magnetfelder oder metallene Gegenstände in Sensornähe verfälscht sein – das sollte man bei der Konstruktion eines Modells berücksichtigen. Bei einigen Sensoren kann man das durch eine Kalibrierung ausgleichen.

Inzwischen bieten verschiedene Hersteller Kompass-Sensoren mit I²C-Schnittstelle an. Im Robotik-Modellbau sind Sensoren von Honeywell und Devantech sehr verbreitet, daher stellen wir im Folgenden zwei Kompass-Sensoren dieser beiden Hersteller vor.

HMC6352

Die amerikanische Firma Honeywell bietet zahlreiche Kompass-Sensoren für industrielle Anwendungen. Einige dieser Sensoren werden u. a. von Sparkfun anschlussfertig als 5 V-I²C-Modul geliefert, wie der 2-Achs-Kompass-Sensor HMC6352. Bestellen kann man ihn für 30-35 € z. B. bei Watterott oder EXP (Abb. 1).

Abb. 1: Kompass-Sensor HMC6352 [3]

Der Sensor verträgt Betriebsspannungen zwischen 2,7 und 5,2 V. Das ermöglicht

einen direkten Anschluss an den TX ohne Level-Shifting. Der Sensor hat die folgenden Leistungsmerkmale:

- Stromverbrauch: 2-10 mA (bei 5 V)
- Update-Rate: 1-20 Hz (variabel)
- Auflösung: 0,5°
- Genauigkeit: 2,5° (mittlerer quadratischer Fehler)
- Mittelwertbildung: 0 bis 16 Messungen (einstellbar, voreingesellt: 4)
- Übertragung: *I²C Standard Mode* (100 kHz)
- Adressbereich: 0x05 bis 0x7B (voreingestellt: 0x21)

Die voreingestellte I²C-Adresse des Sensors ist 0x21 (7 bit). Die Platine hat die Maße 1,5 x 1,5 cm und passt damit sogar genau ins ft-Raster. Die Reihenfolge der Pins des Sensor-Boards entspricht der des TX-EXT 2-Anschlusses (Abb. 2).

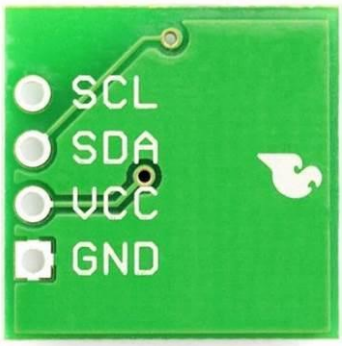


Abb. 2: Pin-Belegung des Sparkfun-Boards

Der in ft:pedia 4/2013 [4] vorgestellte „Universal-Adapter“ kann daher mit dem Board ohne eine Veränderung der Kontaktreihenfolge verbunden werden.

Betriebsarten

Der HMC6352 unterstützt drei Betriebsarten (*Operational Modes*), die im *Operational Mode Register* (RAM-Register 0x74, Bit 0 und Bit 1) eingestellt werden.

Zwischen den Betriebsarten kann im laufenden Betrieb gewechselt werden, indem der Wert des Registers geändert wird.

Beim *Power Up* wird ein voreingestellter Wert aus dem EEPROM (s. u.) ausgelesen und verwendet. Die Betriebsarten sind:

- der *Standby Mode* (*Operational Mode* = 0, Voreinstellung), bei dem Kompass-Messungen nach einem *Get-Data*-Kommando (s. u.) durchgeführt werden;
- der *Query Mode* (*Operational Mode* = 1), bei dem nur ein initiales *Get-Data*-Kommando erforderlich ist, damit nach jedem Auslesen der Daten eine neue Messung veranlasst wird, dadurch sinkt die Antwortzeit erheblich; und
- der *Continuous Mode* (*Operational Mode* = 2), bei dem der Sensor entsprechend der ebenfalls im *Operational Mode Register* (s. u.) konfigurierten Messrate die Richtungsbestimmung durchführt und den Messwert bei einem Lesebefehl automatisch bereitstellt – ohne vorausgegangenes *Get-Data*-Kommando.

In Bit 5 und Bit 6 des Operational *Mode Register* wird die Messfrequenz des *Continuous Mode* gewählt:

- 1 Hz = 0,
- 5 Hz = 1,
- 10 Hz = 2 oder
- 20 Hz = 3.

Durch das Setzen von Bit 4 kann ein periodischer Reset eingestellt werden: Dann wird der Sensor regelmäßig nach wenigen Minuten zurückgesetzt, um Fehler, die z. B. auf Temperaturänderungen oder die Erwärmung des Sensors im Betrieb zurückzuführen sind, zu korrigieren.

Tabelle 1 fasst die im *Operational Mode Register* möglichen Einstellungen zusammen.

0x74	Betriebsart
0x00	*Standby Mode*
0x01	*Query Mode*
0x02	*Continuous Mode, 1 Hz*
0x12	*… with periodic reset*
0x22	*Continuous Mode, 5 Hz*
0x32	*… with periodic reset*
0x42	*Continuous Mode, 10 Hz*
0x52	*… with periodic reset*
0x62	*Continuous Mode, 20 Hz*
0x72	*… with periodic reset*

Tab. 1: Operational Mode Register 0x74

Ausgabeformate

Das Ergebnis einer Messung kann der Kompass-Sensor in fünf unterschiedlichen Formaten ausgeben. Das Output-Format ist im *Output Data Mode Register* (RAM-Register 0x4E) festgelegt (Tab. 2):

0x4E	Ausgabeformat
0x00	*Heading Mode*
0x01	*Raw Magnetometer X Mode*
0x02	*Raw Magnetometer Y Mode*
0x03	*Magnetometer X Mode*
0x04	*Magnetometer Y Mode*

Tab. 2: Output Mode Register 0x4E

- im *Heading Mode* wird die Abweichung des Kompass-Sensors von einer exakten Nord-Süd-Ausrichtung als Zehntel-Grad-Wert (0-3599) geliefert (12 Bit in einem 16-bit-Wert; Voreinstellung);
- im *Raw Magnetometer X* bzw. *Y Mode* wird die Ausgabe des AD-Wandlers als 10-bit-Zweierkomplement geliefert;
- im *Magnetometer X* bzw. *Y Mode* wird die Ausgabe des AD-Wandlers mit dem Kalibrierungs-Offset versehen und skaliert, bevor das Ergebnis ausgegeben wird.

Nach jedem *Power Up* ist der Kompass-Sensor im *Heading Mode*; mit einem RAM-Schreibbefehl (s. u.) kann der Mode im Betrieb gewechselt werden.

Kommandos

Befehle an den Sensor werden nicht in ein Befehlsregister geschrieben, sondern direkt mit einem Schreibbefehl an den Sensor übermittelt. Die folgenden Kommandos sind im Datenblatt dokumentiert [3]:

Wert	Kommando
0x41	*Get Data: Calculate Heading*
0x43	*Enter User Calibration Mode*
0x45	*Exit User Calibration Mode*
0x47	*Write to RAM Register*
0x4C	*Safe Op mode to EEPROM*
0x4F	*Update Bridge Offset*
0x53	*Enter Sleep Mode (sleep)*
0x57	*Exit Sleep Mode (wakeup)*
0x67	*Read from RAM Register*
0x72	*Read from EEPROM*
0x77	*Write to EEPROM*

Tab. 3: Kommandos des HMC6352

Richtungsbestimmung (Heading)

Im *Standby Mode* (s. o.) veranlasst das Kommando 0x41 den Sensor, die Abweichung von einer exakten Nord-Süd-Ausrichtung zu bestimmen und – nach ca. 6 ms – als 16-bit-Wert zurückzuliefern. In welchem Format das Messergebnis geliefert wird, hängt von der Wahl im *Output Mode Register* ab.

Im *Query Mode* und im *Continuous Mode* kann der Mess-Befehl entfallen und der Zwei-Byte-Wert direkt ausgelesen werden (Abb. 3). Da die Messung automatisch direkt nach einem Auslesekommando bzw.

in einem festen Takt vorgenommen wird, muss nicht auf das Messergebnis gewartet werden; das beschleunigt die Abfrage.

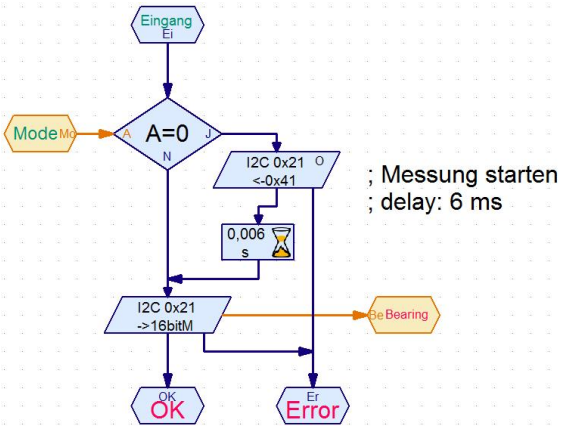


Abb. 3: RoboPro-Befehl zum Auslesen des Kompass-Sensors HMC6352

Auf der Platine ist die korrekte Nord-Süd-Ausrichtung des Sensors durch ein „N“ und einen Pfeil gekennzeichnet (Abb. 1).

Kalibierung

Der Sensor wird bereits kalibriert geliefert und muss normalerweise nicht erneut kalibriert werden. Sollte doch eine Kalibrierung erforderlich sein, um z. B. in einem speziellen Modell störende (feste) Magnetfelder oder Metallgegenstände in Sensornähe bei der Richtungsbestimmung zu berücksichtigen, kann mit dem Befehl 0x43 der *User Calibration Mode* gestartet werden.

Anschließend muss der Sensor mit gleichmäßiger Geschwindigkeit – möglichst zweimal innerhalb von 20 Sekunden – um 360° gedreht werden. Nach frühestens sechs Sekunden und spätestens drei Minuten sollte der Kalibrierungs-Modus mit dem Befehl 0x45 wieder verlassen werden. Daraufhin werden die Magnetometer-Offsets und die Skalierungsfaktoren aktualisiert. Die Magnetometer-Offsets werden in die EEPROM-Register 0x01 bis 0x04 geschrieben und beim nächsten *Power Up* ins RAM kopiert. Die Berechnung selbst benötigt etwa 14 ms.

Stromspar-Modus

Mit dem Befehl 0x53 kann der Sensor in einen Strom sparenden „Schlafmodus“ versetzt werden. Dieser Schlaf-Modus ist *non-operational*, d. h. vor der Aktivierung einer Betriebsart muss der Sensor mit dem Befehl 0x57 erst wieder aufgeweckt werden (0,1 ms Verzögerung).

Set/Reset

Die Set/Reset-Funktion berechnet Offset-Werte zum Ausgleich von thermischen Änderungen oder (permanenten) störenden Magnetfeldern, die als *Bridge Offsets* gespeichert und bei der Bestimmung der Richtung zu den *X/Y Offsets* hinzuaddiert werden.

Die Funktion kann über Bit 4 des *Operational Mode Register* automatisch (regelmäßig nach einigen Minuten) aktiviert oder über den Befehl 0x4F (*Update Bridge Offsets*) direkt aufgerufen werden.

EEPROM-Register

Der Sensor verfügt über 9 Byte-Register im EEPROM, in denen Voreinstellungen wie z. B. die I²C-Adresse, die Default-Betriebsart oder die Versionsnummer der Firmware persistent gespeichert sind (Tab. 4). Diese Werte werden bei *Power Up* in den RAM-Bereich des Sensors kopiert.

Register	Inhalt
0x00	*I²C Slave Address (8 bit, default: 0x42)*
0x01-02	*X Offset*
0x03-04	*Y Offset*
0x05	*Time Delay (0-255 ms; default: 0x01)*
0x06	*Summed Measurements (0-16; default: 0x04)*
0x07	*Software Version*

Register	Inhalt
0x08	*Operational Mode (default: 0x00)*

Tab. 4: EEPROM-Register

Die Register des EEPROMs können mit dem Kommando 0x77 direkt überschrieben werden: Auf den Schreibbefehl müssen die Adresse des EEPROM-Registers (0x00-0x08, Tab. 4) und der Bytewert folgen, der in dieses Register geschrieben werden soll. Geänderte Parameter in den EEPROM-Registern werden allerdings erst beim nächsten *Power Up* aktiv.

I²C-Adresse

Die voreingestellte 7-bit-I²C-Adresse 0x21 (als 8-bit-Wert 0x42) des Sensors, die in Register 0x00 des EEPROMs gespeichert ist, kann mit dem Befehl 0x77 gefolgt von der EEPROM-Adresse 0x00 und der gewünschten Sensor-Adresse (8-bit-Wert!) durch eine I²C-Adresse von 0x05 bis 0x7B (7-bit) ersetzt werden. Die Adressänderung wird beim nächsten *Power Up* wirksam.

Damit können theoretisch 119 HMC6352 ohne Multiplexer [5] an demselben Bus getrennt adressiert werden. In jedem Fall lassen sich durch diese große Zahl einstellbarer Adressen Kollisionen mit anderen I²C-Sensoren auf dem Bus ausschließen.

Hat man die Adresse geändert, sollte man sie sich tunlichst notieren – sonst muss man, wenn man sie vergessen hat, alle möglichen Adressen ausprobieren. Das ist zumindest mit RoboPro sehr mühsam, da die Adresse nicht als Parameter an einen Lesebefehl übergeben werden kann.

X-/Y-Offset

Die X- und Y-Offsets sind Korrekturwerte, die bei der Kalibrierung bestimmt und zu den vom Kompass-Sensor gelieferten Roh-Daten addiert werden. Sie werden beim *Power Up* ins RAM kopiert. Häufig sind beide Werte = 0.

Time Delay

Mit dem *Time Delay* Register 0x05 des EEPROMs kann eine Verzögerung zwischen Messbefehl und -ausführung in ms vorgegeben werden (1-255). Der werkseitig voreingestellte Wert 0x01 steht für „keine Verzögerung“. Wird der Wert verändert, ist ein *Power Up* des Sensors erforderlich, damit die Einstellung aktiv wird.

Mittelwertbildung

Um Schwankungen bei den Messwerten abzuschwächen wird nach der werksseitigen Voreinstellung der Ausgabewert als Mittelwert aus den letzten vier Messwerten gebildet.

Im Register 0x06 (*Summed Measurements*) kann die Zahl der für die Mittelwertberechnung verwendeten Messwerte von 0x00 (keine Mittelwertbildung) bis 0x10 (Mittelwert aus 16 Messwerten) vorgegeben werden. Eine Änderung dieser Einstellung wird erst beim nächsten *Power Up* aktiv.

Firmware-Version

Aus Register 0x07 kann die aktuelle Firmware-Version des Sensors ausgelesen werden. Mein Sensor liefert den Wert 6.

Robo Pro-Treiber

Für die Kompass-Sensoren HMC5843 und HMC6352 von Honeywell sind bereits Treiber in der RoboPro-Bibliothek enthalten. Der von Rei Vilo entwickelte Treiber HMC6352 beschränkt sich auf das Auslesen der Kompass-Richtung und verwendet – vermutlich irrtümlich – statt der voreingestellten I²C-Adresse des Sensors (0x21) die Adresse 0x18; er funktioniert daher erst nach einer Adressänderung. Ein Treiber, der den gesamten Befehlsumfang des HMC6352 zugänglich macht, kann im Download-Bereich der ft:c heruntergeladen werden.

Wer den Sensor in C/C++ nutzen möchte, dem sei die Bibliothek in [6] empfohlen.

CMPS10

Hauptnachteil des HMC6352 ist der starke Fehler bei Neigung: Schon eine Abweichung von einem Grad von einer exakt horizontalen Lage kann einen Richtungsfehler von bis zu zwei Grad verursachen [3]. Daher eignet er sich weniger für ein mobiles Modell wie bspw. ein Fahrzeug.

Neuere Kompass-Sensoren sind zusätzlich mit Beschleunigungssensoren ausgestattet, die die Neigung des Sensors in X- und Y-Richtung bestimmen und neigungsbedingte Fehler automatisch korrigieren. Wertet man die Neigungsdaten aus, eröffnen sich zudem weitere Einsatzmöglichkeiten.

Einer dieser besonders leistungsfähigen Sensoren ist der CMPS10 von Devantech, erhältlich z. B. bei EXP, NoDNA, Manu-Systems oder roboter-teile.de für etwa 30 € (Abb. 3).

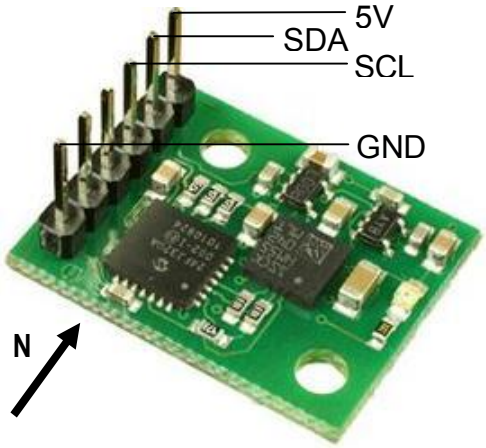


Abb. 3: Kompass-Sensor CMPS10 [7]

Seine Leistungsmerkmale:

- Stromverbrauch: dauerhaft 25 mA (kein ‚Stromspar-Mode')
- Update-Rate: 75 Hz (fest)
- Auflösung: 0,1°
- Genauigkeit: 0,5 %
- Mittelwertbildung: 45 Messungen (fest)
- Übertragung: *I²C Fast Mode* (400 kHz)

Auch dieser Sensor verträgt eine Betriebsspannung von 3,3 bis 5 V und lässt sich direkt an den EXT2-Ausgang des TX anschließen.

Die Platine des CMPS10 liegt mit 2,4 x 1,8 cm leider nicht im ft-Raster; auch sind die beiden Löcher in der Platine zu schmal z. B. für eine Rastachse 20 (31690) oder einen Verbindungsstopfen (32316). Immerhin lässt er sich in zwei Winkelträger 30 (36299) ‚einpacken'.

Ansprechbar ist der Sensor unter der voreingestellten I²C-Adresse 0x60. Über das Command-Register können alternativ die I²C-Adressen 0x61-0x67 eingestellt werden. Die eingestellte Adresse wird beim Einschalten der Stromversorgung durch ein langes und – entsprechend der letzten Stelle der Adresse – *n* kurze Signale der roten LED angezeigt. Bei diesem Sensor darf man eine geänderte Adresse daher getrost auch mal vergessen.

Register

Der Sensor verfügt über 23 Byte-Register, die via I²C-Kommando ausgelesen werden können [7].

Register	Inhalt
0x00	*Firmware Version*
0x01	*Compass Bearing (0…255)*
0x02-03	*Compass Bearing, 16 bit (0…359,9°)*
0x04	*Pitch Angle (-85…85)*
0x05	*Roll Angle (-85…85)*
0x06-09	*Unused*
0x0A-0F	*Roh-Werte Magnetometer (X-/Y-/Z-Achse, je 16 bit)*
0x10-15	*Roh-Werte Accelerometer (X-/Y-/Z-Achse, je 16 bit)*
0x16	*Command Register*

Tab. 3: Sensor-Register

Firmware Version

Der von mir verwendete Sensor liefert bei Auslesen des Registers 0x00 die Firmware-Version 8 zurück.

Compass Bearing

Der Kompass-Sensor liefert die Abweichung von einer exakten Nord-Süd-Ausrichtung des Sensors in den Registern 0x02 und 0x03 als (positiven) 16 bit-Wert in Zehntelgrad (0 bis 3599), also auf eine Nachkommastelle genau zurück. Dem Register 0x01 kann man eine gröbere Auflösung entnehmen: Der 8-bit-Wert ist ein Näherungswert, der die 360° in 256 Stufen misst. Aus ihm gewinnt man die Gradzahl durch Umrechnung.

Die beiden Werte werden als Mittelwert aus 45 Messungen bei einer Messfrequenz von 75 Hz gebildet. Damit werden einzelne Messfehler herausgemittelt. Der Preis: Eine Veränderung der Sensorlage wird erst nach 0,64 sec wirksam, der Sensor reagiert also relativ langsam.

Pitch und Roll

Die Neigung in X- (*Pitch Angle*) und Y-Richtung (*Roll Angle*) in Grad liefert der Sensor als vorzeichenbehafteten Byte-Wert zurück. Dabei wird ein Bereich von -85° bis +85° abgedeckt. Die Werte können den Registern 0x04 und 0x05 entnommen werden.

Rohdaten

Die Rohdaten des Magnetometers in X-, Y- und Z-Richtung sowie die des 3D-Beschleunigungssensors können als 16-bit-Werte aus den Registern 0x0A-0x0F bzw. 0x10 bis 0x15 ausgelesen werden.

Der Zugriff auf die Rohdaten ermöglicht – unter Inkaufnahme von Mess-Ausreißern – eine schnellere Reaktion auf Richtungsänderungen, da die Werte bei einer Messfrequenz von 75 Hz alle 13,3 ms aktualisiert werden. Außerdem erlauben sie die Nutzung des integrierten Beschleunigungssensors – aber davon mehr in einem separaten ft:pedia-Beitrag.

Kommandos

Da die Messwerte in den Registern des Sensors abgelegt und jederzeit ausgelesen werden können, benötigt der CMS10 kein Mess-Kommando. Die Befehle, die in das *Command Register* geschrieben werden können, betreffen daher nur die Konfiguration des Sensors. Tabelle 4 gibt eine Übersicht der Kommandos.

Wert	Kommando
0x20	*Reset to Default Offset Values*
0xA0	*Change I²C Address*
0xF0	*Enter Calibration Mode*
0xF5	*Store Offset*

Tab. 4: Kommandos

Rücksetzung des Sensors

Um den Sensor auf die werksseitige Kalibrierung zurückzusetzen, wird die folgende Befehlsfolge (mit anschließend je 100 ms Pause) an das Command-Register geschickt:

0x20, 0x2A, 0x60

Änderung der Adresse

Das Befehlsregister 0x16 erlaubt die Einstellung der Sensor-Adresse. Um die Adresse zu wechseln wird die folgende Kommando-Folge (mit anschließend je 100 ms Pause) an das Register 0x16 geschickt:

0xA0, 0xAA, 0xA5

gefolgt von der gewünschten Adresse (0x60 bis 0x67).

Kalibrierung des Sensors

Der CMPS10 wird bereits kalibriert ausgeliefert, sodass eine Kalibrierung des Sensors in der Regel nicht erforderlich sein

dürfte. Wird er stationär in der Nähe eines störenden Magnetfelds betrieben, kann eine Kalibrierung dennoch sinnvoll sein.

Dazu sind zunächst die Himmelsrichtungen exakt zu bestimmen. Dann wird der Sensor nach Norden ausgerichtet. Mit dem Kommando 0xF0 im Command-Register 0x16 wird nun der Kalibrierungs-Modus eingestellt. Der Befehl 0xF5 speichert die Ausrichtung als ‚Norden' (bestätigt durch ein Aufleuchten der roten LED).

Dann wird der Sensor um 90° im Uhrzeigersinn gedreht und erneut der Befehl 0xF5 ins Command-Register geschrieben. Das wird zwei weitere Male wiederholt, dann ist der Sensor neu kalibriert.

Robo Pro-Treiber

Für den Vorgänger des CMPS10, den CMPS09, gibt es in der Robo Pro-Bibliothek einen schönen Treiber – der auch mit dem befehlskompatiblen CMPS10 funktioniert.

Es fehlen lediglich Unterprogramme zur Adressänderung und Kalibrierung – beides Funktionen, die man nur in Ausnahmefällen benötigt, z. B. bei einer Adresskollision auf dem I²C-Bus oder einem störenden Magnetfeld bei einem stationären Modell. Diese Funktionen sind bei Bedarf schnell ergänzt.

Fazit

Der Honeywell-Sensor ist die richtige Wahl für ein nicht mobiles Modell, bei dem man mit dem verfügbaren Strom haushalten muss: Mit 1 mA hat er einen wesentlich geringeren Stromverbrauch als der Devantech-Sensor und kann sogar in einen „Sleep"-Modus versetzt werden. Außerdem verhindert er durch die quasi freie Adresswahl jeden möglichen I²C-Adresskonflikt, und die Platine passt auch noch perfekt ins ft-Raster.

Dafür unterstützt der Devantech-Sensor mit dem *I²C Fast Mode* eine viermal höhere Übertragungsgeschwindigkeit, hat dank der Beschleunigungssensoren eine um den Faktor fünf höhere Auflösung, arbeitet auch bei nicht exakt horizontaler Ausrichtung korrekt und liefert neben der Richtung auch Neigungsinformationen – das kann einen separaten Beschleunigungssensor einsparen. Die Messrate von 75 Hz ist fast viermal so hoch wie die des Honeywell-Sensors – bei etwa gleichem Preis.

Eine schöne Anwendung des Sensors findet ihr in einem folgenden Beitrag zum Thema „Navigation", der zeigt, wie man aus GPS-Koordinaten die Kompass-Richtung gewinnt und so z. B. einen autonomen Roboter zu einem Zielpunkt steuern kann.

Quellen

[1] Dirk Fox: *I²C mit dem TX – Teil 6: GPS-Sensor*. ft:pedia 3/2013, S. 54-62.

[2] Wikipedia: *Deklination*.

[3] Honeywell: *2-Axis Compass with Algorithms HMC6352*. Datasheet, 2009.

[4] Dirk Fox: *I²C mit dem TX – Teil 7: Real Time Clock (RTC)*. ft:pedia 4/2013, S. 28-34.

[5] Georg Stiegler: *I²C mit dem TX – Teil 5: Multiplexer*. ft:pedia 2/2013, S. 50-52.

[6] Aaron Berk: *HMC6352 Class Reference*. mbed.org, 2010.

[7] Devantech: *CMPS10 – Tilt Compensated Compass Module*.

Computing

TX Bridge

Ad van der Weiden

The TX is not backward compatible with the Robo-Interface so the extension modules cannot be used as slaves of the TX. To expand the number of inputs or outputs on a TX you need to buy a full TX while you may still have a few extension modules which will provide you with 4 motor outputs, 8 digital inputs and an analog input. Looking for a solution is a logical step, but fischertechnik has not revealed a lot of information about the interfaces. Luckily, Thomas Kaiser (thkais) has done a lot of hard work on this.

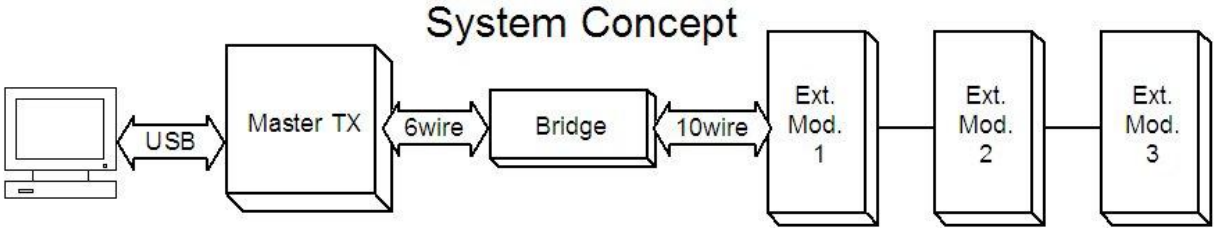


Fig. 1: System Concept

The idea

As fig. 1 shows, my solution is a bridging unit which translates between the Robo TX interface and the Robo I/O extension modules. Fig. 2 shows the finished setup:

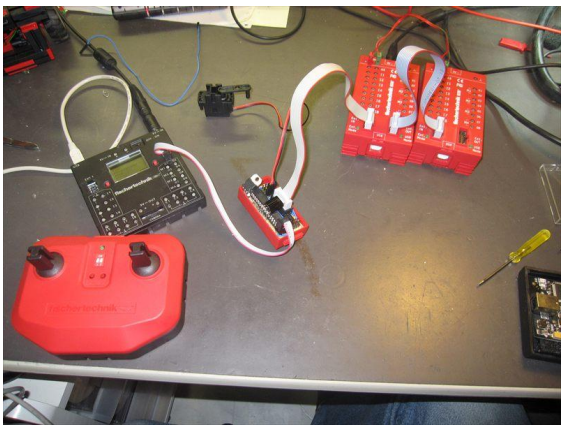


Fig. 2: Test setup showing the concept

Background

On his website, thkais has both analysed the Robo Interface extension protocol [1] as well as provided a circuit to emulate the slave end of the connection. The hardware on the master end was already analysed by Werner Hobelbrecht (hobelbrecht, see fig. 3).

This shows that all extension outputs are open collector and that the inputs are connected via resistors to the processor. The next problem is of course the TX Controller, here fischertechnik has provided the pin out of the connectors and the basics of the X1 protocol a zip package on their computing web page [2]. Again, thkais has made an in-depth analysis of the protocol in an unpublished document concerning the actual content of the messages.

Hardware

The hardware of the bridge is extremely simple and as minimal as possible. For the processor I selected the Atmel ATMega88 [3] which is cheap, easily to obtain, available in DIP and has a free IDE

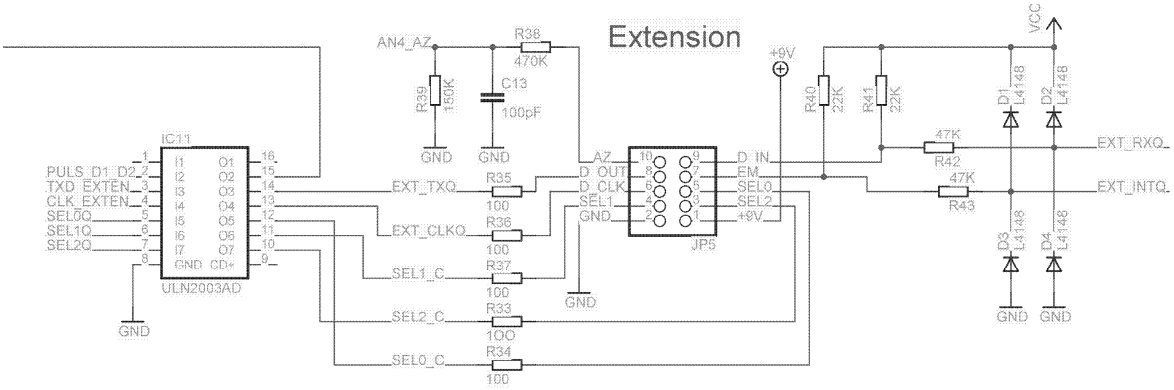


Fig. 3: Drawing by Werner Hobelbrecht

(AtmelStudio). There is also an easy upgrade to the ’168 and ’328 which is now popular as Arduino chip. The clock frequency must be 14,745,600Hz because the USART requires 16x the baud rate of 921,600 Baud. The main problem is in fact the selection of the supply voltage. On the TX side 5V is preferable while on the Ext. side 3.3V is preferable. 3.3V is however too low (according to the specification) for the required clock. I have however built both versions and both work. In this article I will concentrate on the 3.3V version.

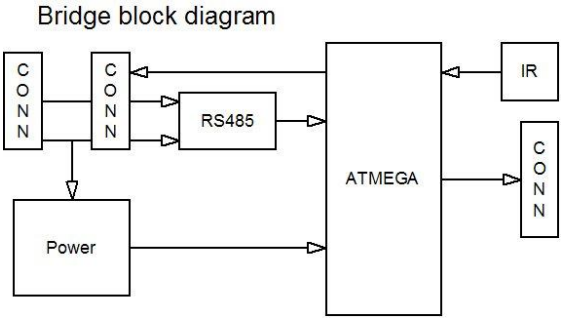


Fig. 4: Block diagram of the bridge module

The 3.3V is derived directly from the 5V as is available on the right hand extension connector of the TX. The 5V is further used to power the RS485 [4] line driver/receiver. As line driver/receiver I selected the cheap SN75176 [5] or equivalent. The receiver requires a bias network [6] because when the TX enters receive mode, the bus is temporarily in an undefined state which causes receive errors. The bias resistors force the bus to a high state when not driven. When comparing the datasheets of the driver/receiver and the microcontroller, you can see that the voltages are compatible and I had no problems in my prototype. To be on the safe side, you could add a pull-up on the TxD line and a series resistor or diode in the RxD line.

On the Ext. side of the bridge no special interfacing is necessary, I only added an extra pull-up on the Acknowledge line.

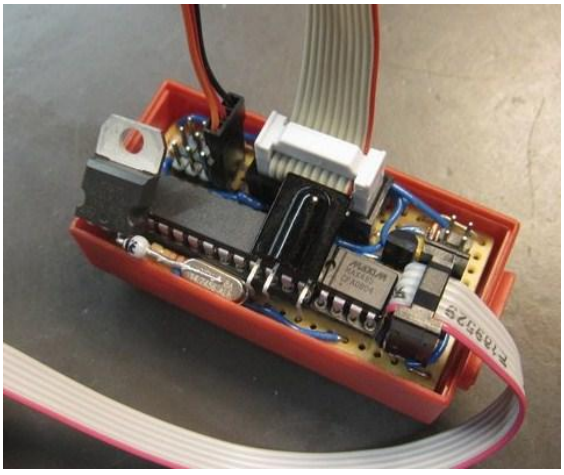


Fig. 5: Prototype

Optional hardware

Another feature missing from the TX that was available on the Robo Interface is a Control Set decoder. The Robo Interface could decode RC5 of the old Control Set. The new Control Set uses a different coding scheme that was published once on the old forum but removed by ft ☹. Now is

a good time to add this feature at the cost of only an IR detector [7].

Other features we could add are an I²C interface (now not so important anymore), outputs for servo motors (more difficult than it looks) or analogue inputs (e.g. for Sharp distance sensors [8]).

The schematic shown in this article has only support for the Control Set and I²C (with level shifting to 5V [9]).

X1 Protocol

As already stated above, the X1 protocol is specified on the ft download site. Not defined are the commands, payloads and message sequences of the protocol. The mystery fields in the messages are the command codes and the payloads. The X1 protocol is essentially the same for USB communication, Bluetooth PC-TX communication and RS485 communication. The protocol is a request-reply protocol (with the exception of some Bluetooth messages (not on RS485)). Requests have low integer command codes, so far I have seen codes 1, 2 and 5-20. Not all commands are used in TX-TX communication though. The replies always have a command code of 100 + the request code.

The commands used on the RS485 connection are shown in table 1.

After power-up the master sends a *get_info* command to the first slave, the slave answers with a *TA_INFO* structure which contains a.o. the name of the slave, the slave replies with a new odd session ID. The master then sends a configuration command with a *TA_CONFIG* structure which describes the configuration of the inputs and outputs of the slave and a session ID of zero. These payload structures are declared in *ROBO_TX_FW 1_24.h* which can be downloaded from the ft site [2]. The slave replies with a new session ID. The master repeats the configuration command with new session ID, the slave now replies with the same session ID.

request	reply	meaning	payload
2	102	output	TA_OUTPUT
5	105	config	TA_CONFIG
6	106	get_info	none
8	108	display	DISPLAY_MSG
9	109	name	C string
	102	input	TA_INPUT
	105	ack	none
	106	info	TA_INFO
	108	ack	none
	109	ack	none

Table 1: Commands on RS485 connections

From this point on the master starts continuously polling all slaves it has found by sending output commands and receiving input replies. At regular intervals the master sends *get_info* commands to see if new slaves have appeared on the bus. The two other commands allow changing the name of the slave and displaying a message on the display of a slave TX. As the bridge does not have a display, we can use this command to set some parameters. The *name* command is implemented but the name is stored in the bridge only, not in the actual extension.

Command mapping

It is important to understand that the bridge replies on behalf of the extension units and emulates a TX slave for each extension unit. Besides this, the slave can also answer on its own behalf which is useful for advanced functions like IR receiver or servo motors. Depending on how many extension units are actually connected, the bridge will answer to "get_info" with "ROBO I/O Ext 1", 2, 3 etc. The bridge will present itself as "ROBO-TX Bridge". When e.g. 3 extension units are connected,

the master will show: Ext.1, 2, 3, 8 where '8' represents the bridge.

Fig. 6: Two I/O Extension modules (1, 2) and the bridge itself (8)

When the bridge receives a configuration command for a particular extension unit, it stores this data in a data structure so it will know what data to return later. For the extension units it makes no difference whether they are used as 8 half-bridges or 4 full-bridges, so the motor configuration is ignored.

Input	Digital	Analog
1	I1	AX
2	I2	AZ
3	I3	AV
4	I4	I4
5	I5	I5
6	I6	I6
7	I7	I7
8	I8	I8

Table 2: Input mapping

The inputs are more problematic because a TX has 8 universal inputs whereas an extension has 8 digital inputs and 3 analogue inputs (AX, AZ and AV). When input 1 is configured as analogue it will return AX, otherwise I1, when input 2 is configured as analogue it will return AZ, otherwise I2, and similarly input 3 will return AV. Inputs I4-I8 are always digital.For the bridge itself a different mapping was chosen:

input	voltage	distance	resistance
1	IR 1	IR 1	I2C 1
2	IR 2	IR 2	I2C 2
3	IR 3	IR 3	I2C 3
4	IR 4	IR 4	I2C 4
5	IR On	Dist 1	I2C 5
6	IR Off	Dist 2	I2C 6
7	IR freq	Dist 3	I2C 7
8	IR rec	Dist 4	I2C 8

Table 3: Input mapping as used in the bridge

The I^2C data is now obsolete because the TX I^2C interface became useable in the meantime.

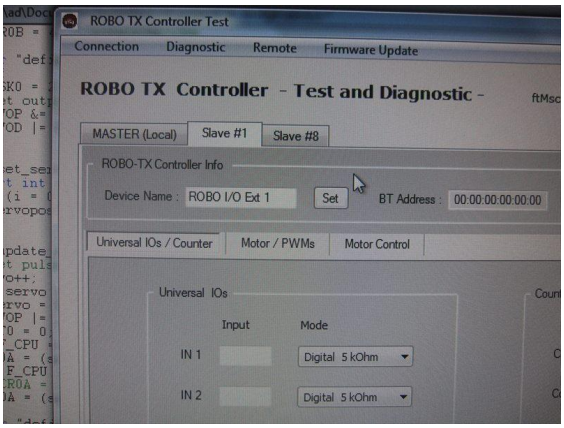


Fig. 7: The Slave is named ROBO I/O Ext 1

The Software

The software is written completely in "C" and has more or less the following architecture:

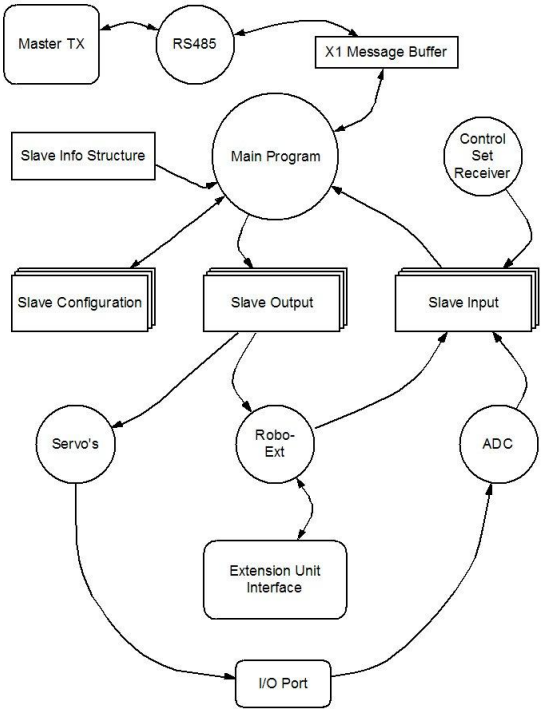


Fig. 8: Dataflow Diagram [10]

The system makes extensive use of interrupts [11] to keep up with all events. The most important interrupt is the USART RX interrupt because of the extremely high baud rate.

The interrupts are:

IRQ	Use
USART_RX	RS485 receive
USART_UDRE	RS485 transmit
TIMER2_COMPA	100us system timer
TWI	I2C
ADC	Distance sensors
EE_READY	Writing to EEPROM
TIMER0_COMPA	Servo pulse
INT0	Ext.Mod Ack
SPI	Ext.Mod Data
TIMER1_CAPT	ControlSet pulse
TIMER1_OVF	ControlSet overflow

Table 3: Interrupts

The main program loop is responsible for parsing the X1 messages and sending replies.

Program structure

Like most embedded programs, the TX bridge program has the following well known structure:

```
//#includes
//#defines
//declarations and definitions

void main()
{ init_RS485();
  //other initialisations
  sei(); //enable interrupts
  for(;;) //do forever...
  { if (tx_dest < nrofext ||
        tx_dest == thisbridge)
    { switch (msg.hdr.cmd)
      { case 2: copy_to_ext(idx);
                send_102(idx);
                break;
        case 5: store_config(idx);
                send_105(idx);
                break;
        case 6: send_106(idx);
                break;
      }//switch
    }//if
  }
}
```

Listing 1: General program structure

The *init_RS485* function initialises the USART and puts the transceiver in receive mode. When an X1 message is received, *tx_dest* gets the index value of the extension the message is addressed to. If the message is for one of the extensions that we emulate, we parse the message header to identify the command. In the above snippet we check for commands 2, 5 and 6 as shown in table 1 and send the appropriate response. The receiver interrupt routine populates the X1 message buffer.

```
ISR(USART_RX_vect)
{ data = UDR0;
  switch (state)
  { case 0: //idle
            if (data == 0x02)
                  state = 1;
            break;
```

```
    case 1: //stx
            if (data == 0x55)
                  state = 2;
            else
                  state = 0;
            break;
    case 2://length H
            msg.hdr.bytesL = data;
                  state = 3;
            break;
    case 3://length L, low byte
            msg.hdr.bytesH = data;
            wp = message_begin;
            if (msg.hdr.bytes >
                    sizeof(msg)-7)
                  state = 0;
            else
                  state = 4;
            break;
    case 4://store
            *wp++ = data;
            if (wp >= message_end)
            { tx_message =
                  msg.hdr.rec - 3;
              state = 0;
            }
            break;
  }
}
```

Listing 2: Interrupt service routine

The receiver interrupt service routine is implemented as a state machine with 5 states. In the idle state (0) it waits for the STX character (0x02) to arrive, in states 2 and 3 it determines the size of the message which is required in state 4 to determine the end of the message (in reality 'message_end' is computed based on *mg.hdr.bytes*).

The program is too large to go into every detail, if you are interested please have a look at the source code as posted on the ftc website [12].

Boot loader

As it may happen that the firmware needs to be updated and not all users have a suitable programmer, an attempt was made to integrate a boot loader. Often the serial port is used for this purpose, but in our case the UART is connected to RS485. Not everybody has an RS485 adapter and the TX cannot easily be used as such because of software limitations[1]. Another option would be a software USB interface like V-USB, but this was not foreseen on the PCB. Therefore, the I2C interface was selected as physical interface (connects to TX via same ribbon cable). The boot loader software itself is based on Atmel application notes AVR109 and AVR311 (for the TWI slave function). On the host side, a RoboPro program takes care of sending the data to the I2C interface. The data is sent from a list which is loaded from a CSV file. An external program is needed to convert Intel HEX files to CSV files, but of course the firmware can also be distributed as CSV files.

Conclusions

With enough effort and co-operation it is possible to interface to the proprietary ft controllers. Besides the prototype in the picture I have built two other versions. One operates at 5V, is built on a single sided PCB and is primarily intended to control stepper motors via I^2C. Another version operates at 3.3V, is built on a double sided PCB, and has mounting holes, level shifters for 5V I^2C but no direct connection for servos or distance sensors. The servo feature is problematic anyway, caused by the resolution of the 8 bit counter and by the interrupt latency. This results in quite a bit of jitter in the ft servo. Because I consider the servos as a useful feature (ft components that can otherwise only be used with the Control Set), we can look into another solution that uses the PWM outputs directly. If more people want to build a bridge it may also be worthwhile to provide a kind of self-programming feature for updates.

[1] In principle, the TX display command allows sending data to the bridge over RS485. There is however no back channel to verify the data and the boot loader would need an X1 parser.

References

[1] Thomas Kaiser, *Übertragungsprotokoll RoboInterface -> Extension-Modul*.

[2] MSC Vertriebs GmbH, *Programming the ROBO TX Controller, Part 1: PC programming*. In: PC Programming Robo TX Controller Version 1.5, Aachen, 11/24/2012

[3] Atmel Corporation, *8-bit Atmel Microcontroller with 4/8/16K Bytes In-System Programmable Flash*. San Jose, 2011

[4] Wikipedia, *RS-485*.

[5] Texas Instruments, *SN75176A Differential Bus Transceiver*. Dallas, May 1995

[6] Wikipedia, *RS485 Bias Termination*.

[7] Vishay Semiconductors, *IR Receiver Modules for Remote Control Systems*. 2/5/2014

[8] SHARP Corporation, *GP2Y0A21YK Optoelectronic Device*. Osaka, 2005.

[9] NXP, *Level shifting techniques in I2C-bus design*. 6/18/2007.

[10] Wikipedia, *Data flow diagram*.

[11] Wikipedia, *Interrupt*.

[12] Ad van der Weiden, *TX Bridge firmware source code and Eagle files*. 3/7/2014.

Modell

Detail Engineering (2) – Ansteuerung von Leistungsmotoren

Andreas Gail

Im Rahmen des Baus eines Robotermodells wurde eine Reihe von ganz unterschiedlichen Detaillösungen erarbeitet, die durchaus Lösungsansätze bei diversen anderen Bauprojekten sein könnten. Im zweiten Beitrag der Serie wird der Antrieb vorgestellt.

Aufgabenstellung

Der immerhin etwa 5 kg schwere Roboter gemäß Abb. 1 sollte mithilfe von vier Leistungsmotoren vom Typ Power-Motor 20:1 angetrieben und mit dem Robo TX Controller (RTXC) automatisiert werden.

Abb. 1: Roboter, angetrieben mit vier Power-Motoren 1:20

Beim RTXC gibt es vier Motorenausgänge mit einer Belastbarkeit von jeweils 250 mA. Die früheren Power-Motoren wurden jedoch alle auf 1,1 A Stromaufnahme ausgelegt, also deutlich zu viel für den RTXC. Sicher aus diesem Grund wurde das Motorenprogramm geändert; alle aktuellen Motoren haben geringere Stromaufnahme und Leistung, was teilweise durch fest verbaute Getriebesätze kompensiert werden soll.

Volle Beweglichkeit

Jeder der vier Leistungsmotoren treibt dabei ein einzelnes Rad an. Zwei Motoren und Räder einer Seite bilden einen gemeinsamen Radsatz. Dieser Aufbau (siehe Abb. 2) erlaubt eine vielfältige Bewegungsfähigkeit des Roboters.

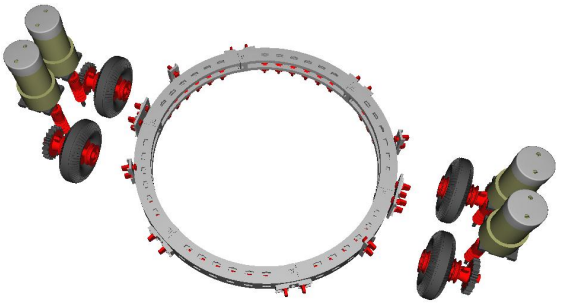


Abb. 2: Antriebskonzept des Roboters

Eine Rotation auf der Hochachse wird beispielsweise erreicht, indem ein Radsatz vorwärts und der andere rückwärts angetrieben wird. Ein Radsatz kann weiterhin mit zwei unterschiedlichen Geschwindigkeiten betrieben werden, indem die beiden Motoren eines Radsatzes entweder parallel oder in Reihe verschaltet werden.

Lösung

Letztendlich müssen die vier Motoren vom RTXC aus angesteuert werden. Um stets

sicher zu sein, die max. Stromaufnahme des RTXC nicht zu überschreiten, kommen Relais zum Einsatz. Leider gehören die nicht mehr zum aktuellen fischertechnik-Sortiment, so dass ein Selbstbau erfolgte (Abb. 3).

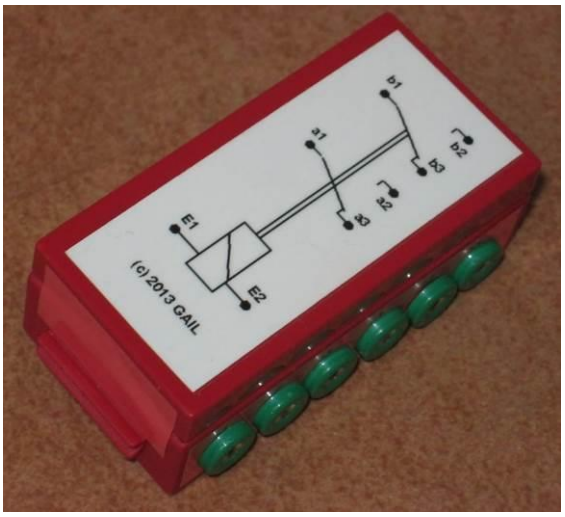


Abb. 3: Selbstbau-Relais

Im Gehäuse verbaut ist ein Signalrelais vom Typ Axicom, tyco Electronics, C93428, 5-14620000-7, mit einer Ansteuerspannung von 3,9 V bis 14,0 V (Conrad #504 314).

Wie nachfolgend gezeigt, werden für die Ansteuerung der vier Motoren insgesamt sechs Relais verwendet, d. h. für jeden Radsatz drei Relais. Zugehörig werden auf dem RTXC auch sechs digitale Ausgänge benötigt. Die drei Relais eines Radsatzes haben die folgenden Funktionen, die in der angegebenen Reihenfolgen verschaltet werden:

- an / aus
- Richtungseinstellung
- Geschwindigkeitseinstellung

Relaisschaltung an/aus

Der Relaiseingang E1 wird von einem einzelnen Ausgang des RTXC angesteuert. Wenn das der Fall ist, wird durch das Relais die Verbindung a1-a2 hergestellt, sodass der angeschlossene Motor zu laufen beginnen würde (Abb. 4).

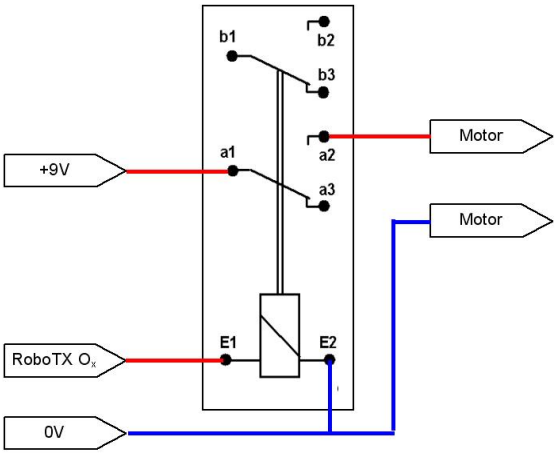


Abb. 4: Relaisschaltung an/aus

Relaisschaltung Richtungseinstellung

Der Relaiseingang E1 wird von einem einzelnen Ausgang des RTXC angesteuert. Der Motor läuft dabei immer, bei Schaltvorgängen des Relais kommt es zur Drehrichtungsänderung des Motors (Abb. 5).

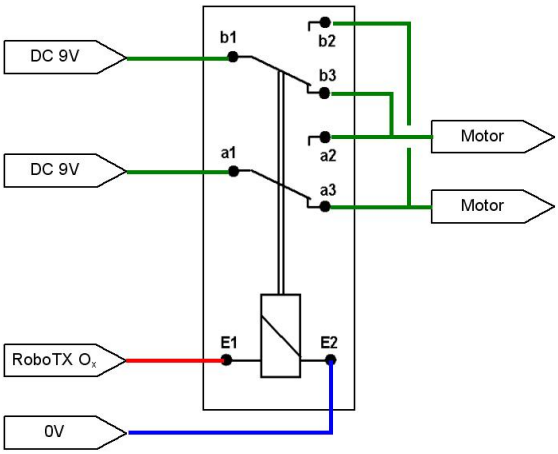


Abb. 5: Relaisschaltung Richtungseinstellung

Relaisschaltung Geschwindigkeitseinstellung

Der Relaiseingang E1 wird von einem einzelnen Ausgang des RTXC angesteuert. Die Motoren laufen dabei immer, bei Schaltvorgängen des Relais kommt es zur Parallelschaltung dieser Motoren oder zur Reihenschaltung (Abb. 6).

Bei Reihenschaltung laufen die Motoren langsam, bei Parallelschaltung schnell. Auf diese Weise ist es möglich, eine Geschwindigkeitsumschaltung zu erreichen. Dabei ist jedoch zu beachten, dass bei der Reihenschaltung im Fall des Blockierens eines Motors der andere Motor schneller läuft.

In der ft Community wurde übrigens der Effekt der Reihenschaltung von Motoren bei einer anderen Anwendung (Flash) von Jürgen Landstuhl (chevyfahrer) beschrieben.

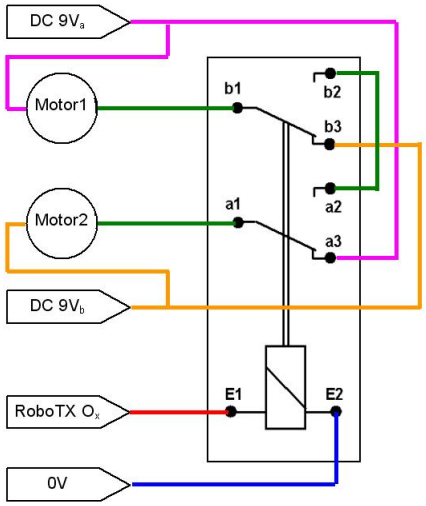


Abb. 6: Relaisschaltung Geschwindigkeitseinstellung

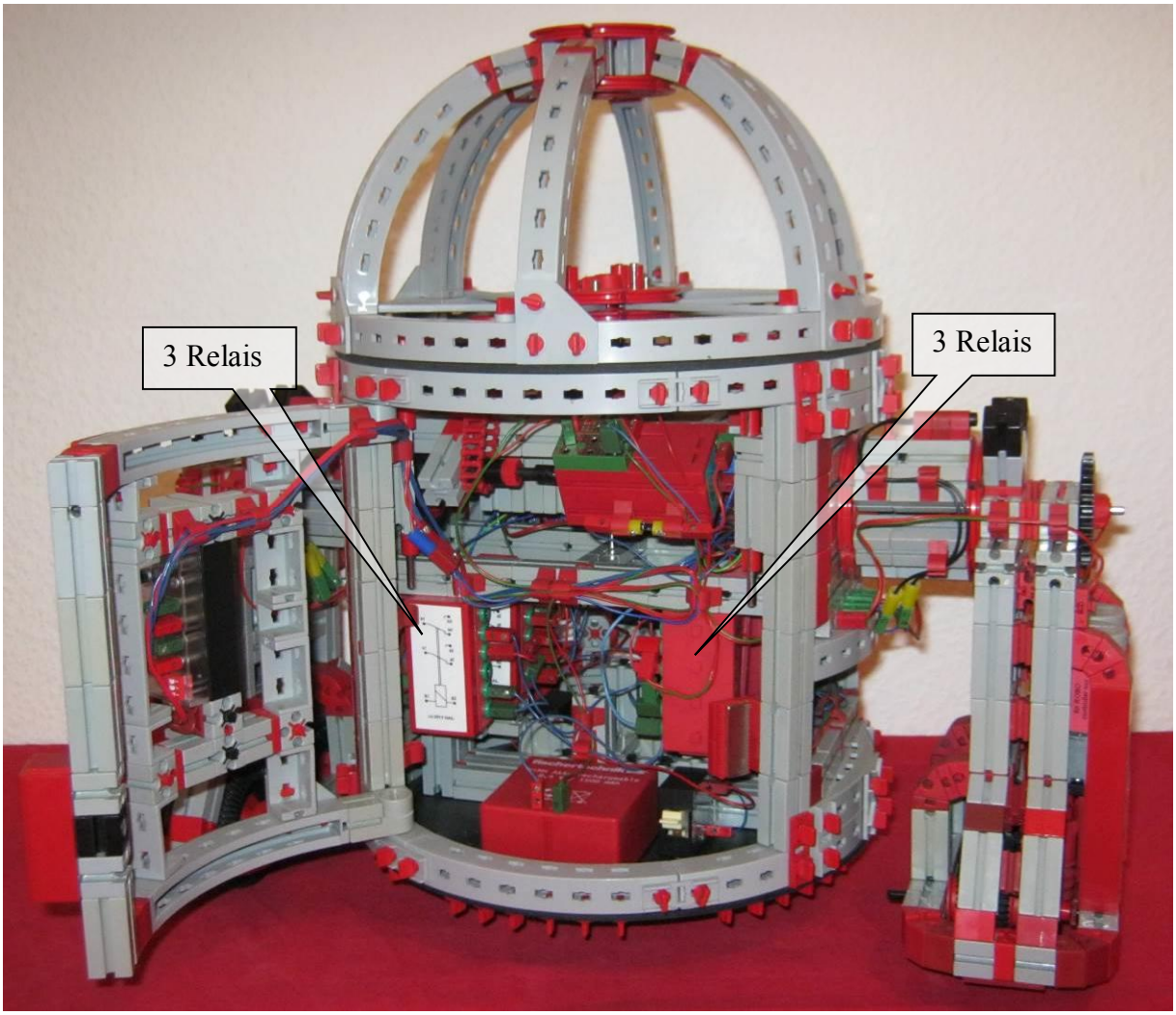


Abb. 7: Einbauposition der sechs Relais

Abb. 8: Radsatz mit zwei Motoren (vorne rechts)

Analogcomputer

Der Seilcomputer Kelvin

Thomas Püttmann

Mit wenigen Bauteilen und auf einer Grundplatte entsteht ein fischertechnik-Seilcomputer, der lineare Gleichungssysteme löst und die Optik und Haptik klassischer analoger Messgeräte besitzt. Er eignet sich hervorragend als Lernspielzeug, weil er das zentrale mathematische Konzept der linearen Gleichungssysteme anschaulich und begreifbar macht.

Vorgeschichte

Lineare Gleichungssysteme sind von zentraler Bedeutung in der Mathematik wie in den Natur- und Ingenieurwissenschaften. Egal, ob es um die Konstruktion großer Brücken, um komplexe Schaltkreise oder um Computertomographie geht: Bei der mathematischen Modellierung von Objekten und Prozessen müssen häufig lineare Gleichungssysteme gelöst werden. Solche Systeme von Hand zu lösen ist aufwendig und zeitraubend – vor allem, wenn sie viele Gleichungen und Unbekannte beinhalten.

So ist es nicht verwunderlich, dass viele Wegbereiter des Computerzeitalters Maschinen zur Lösung linearer Gleichungssysteme entwickeln wollten [1]. Schon im Jahr 1878 schlug William Thomson, der spätere Lord Kelvin, eine Apparatur aus Wippen und Seilzügen zur Lösung linearer Gleichungssysteme vor [2]. Erst in den 30er Jahren des 20. Jahrhunderts wurde Thomsons Konzept – mit Veränderungen – in größerem Maßstab (Systeme mit neun Gleichungen für neun Unbekannte) am renommierten Massachusetts Institute of Technology (MIT) in Vannevar Bushs Arbeitsgruppe von John Wilbur realisiert [3]. Unterstützt wurde der Bau von der Nähmaschinenfirma Singer. Der Simultaneous Calculator wog nahezu eine Tonne, enthielt fast 1.000 kugelgelagerte Seilrollen und fast 200 m Stahlband. Seine Fertigstellung wurde in der Presse mit Schlagzeilen wie ‚*Robot Mathematician Solves Nine Simultaneous Equations*‘ bekannt gegeben.

Durch den Siegeszug programmgesteuerter elektronischer Rechner fanden der Simultaneous Calculator wie Kelvins Entwurf überhaupt allerdings keine weitere Verbreitung. Der Simultaneous Calculator verschwand sogar auf unbekannte Weise, eventuell war er zuletzt in den 80er Jahren im damaligen IBM Gebäude (590 Madison Avenue) in New York ausgestellt. Erhalten ist jedoch ein japanischer Nachbau aus dem Jahr 1944, der im National Museum of Nature and Science in Tokyo zu sehen ist.

Lineare Gleichungssysteme

Um den Aufbau und die Bedienung des Seilcomputers Kelvin zu erklären, ist es sinnvoll, zuerst ein einfaches Beispiel für ein lineares Gleichungssystem anzugeben:

$$\begin{aligned} 4\,x + (-2)\,y &= 0 \\ -4\,x + 4\,y &= 2 \end{aligned}$$

Die roten Zahlen nennt man Koeffizienten, die blauen Inhomogenitäten. Sie sind vorgegeben. Gesucht sind die schwarz gesetzten Unbekannten x und y – allerdings nicht unabhängig von einander, sondern auf

einmal. Deshalb ist es wichtig, hier genauer zu formulieren: Eine Lösung des Systems ist ein Punkt (x, y), so dass beide Gleichungen erfüllt sind, wenn man die Koordinaten einsetzt. In unserem Beispiel ist der Punkt $(x, y) = (½, 1)$ eine Lösung, denn

$$4 \cdot ½ + (-2) = 0$$
$$-4 \cdot ½ + 4 = 2$$

Gesucht sind alle möglichen Lösungen, also die sogenannte Lösungsmenge. Wie man die Lösungsmenge rechnerisch bestimmt, lernt man in der Schule. Für das Verständnis wichtiger ist die zeichnerische Methode. Jede Gleichung stellt eine Bedingung dar, die einen eigentlich in der Ebene vollkommen freien Punkt auf eine Gerade einsperrt. Die Lösungsmenge des Systems ist die Schnittmenge der beiden Geraden. In unserem Beispiel schneiden sich die beiden Geraden genau in dem Punkt $(x, y) = (½, 1)$, siehe Abb. 1. Es gibt also nur diese Lösung.

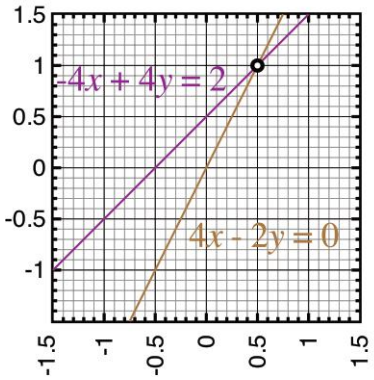


Abb. 1: Lösungsmenge eines Gleichungssystems als Schnittmenge der Lösungsmengen der einzelnen Gleichungen

Der Seilcomputer Kelvin

Kelvin kann in verschiedenen Größen gebaut werden. Die einfachste Variante für Gleichungssysteme mit zwei Gleichungen und zwei Unbekannten passt auf eine Grundplatte. Sehr gut gefällt mir auch die Variante für vier Gleichungen und vier Unbekannte, die auf zwei Grundplatten passt. Kelvins Aufbau versteht man am besten anhand des gerade angegebenen Beispiels und Abb. 2. Zu jeder Gleichung gehört ein Seil: Das hintere Seil zur ersten Gleichung, das vordere Seil zur zweiten Gleichung. Die Koeffizienten werden eingestellt, indem die Radhalter in den Nuten der Wippen entsprechend verschoben werden: Links hinten auf +4, rechts hinten auf −2, links vorne auf −4 und rechts vorne auf +4.

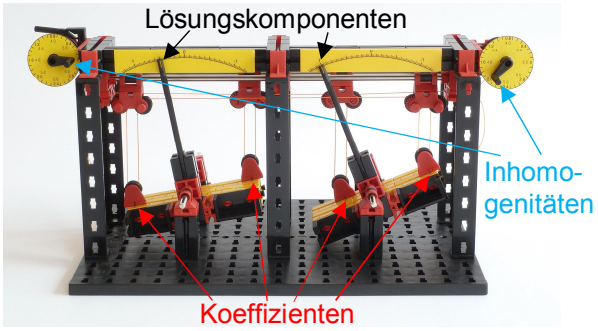


Abb. 2: Mechanisierung des Gleichungssystems
$4x - 2y = 0,\ -4x + 4y = 2$

Die Inhomogenitäten werden über die Seilwinden eingestellt: Die linke Seilwinde gehört zum hinteren Seil, damit zur ersten Gleichung und bleibt auf 0 stehen, die rechte Seilwinde gehört zum vorderen Seil, damit zur zweiten Gleichung und wird auf +2 gedreht. Die Unbekannte x entspricht der Neigung der linken Wippe, die Unbekannte y entspricht der Neigung der rechten Wippe. Abgelesen werden können die Lösungskomponenten auf den mit Skalen versehenen gelben Bauplatten 6×1.

Bedienung

Kelvin hat eine lange Entwicklungsgeschichte hinter sich und unterscheidet sich sowohl von Thomsons ursprünglichem Konzept als auch von Wilburs modifizierter Umsetzung in mehreren Details und in einem entscheidenden Punkt: Er löst das eingestellte Gleichungssystem nicht permanent, sondern der Benutzer führt den Lösungsvorgang aktiv durch. Dazu drückt der Benutzer mit den Fingerkuppen so auf die Ablagefächer an den Wippen, dass bei-

de Seile stramm sind. Das Verfahren erfordert etwas Fingerspitzengefühl, mit etwas Übung kann man damit aber sehr genaue Ergebnisse erzielen. Der Vorteil des aktiven Lösens ist zum einen, dass ein Verklemmen vermieden wird. Dies wird aus der Theorie weiter unten verständlich. Zum anderen bekommt das Lösen eines Gleichungssystems so ein spielerisches Element und Kelvin wird zu einem hochwertigen Lernspielzeug. Der verkoppelte Einbezug der Sinne Sehen, Fühlen und Hören (Sperrklinke an den Seilwinden) beim Lösen ermöglicht ein intensives, schnelles und nachhaltiges Lernerlebnis.

Der genaue Lösungsvorgang ist wie folgt:

- Die beiden Inhomogenitäten werden auf 0 gedreht.
- Die Koeffizienten werden eingestellt. Dabei werden die Schlitten auf den Achsen so mitbewegt, dass die Seilabschnitte zwischen den Koeffizientenseilrollen und den Schlitten vertikal verlaufen. Zum Teil erfolgt das automatisch.
- Es wird ein Nullabgleich durchgeführt (Abb. 3). Dazu wird mit den Fingerkuppen so auf die Ablagefächer an den Wippen gedrückt, dass beide Zeiger genau 0 anzeigen. Der Benutzer merkt sich, wie fest er dazu auf welche Ablageflächen drücken musste.

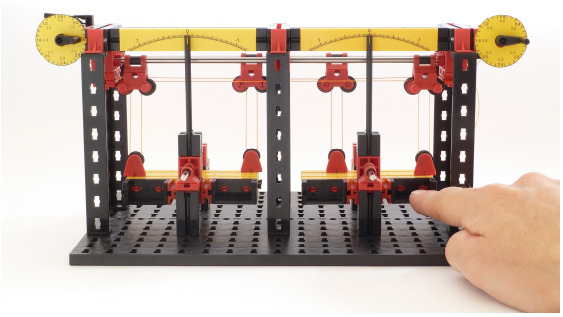


Abb. 3: Nullabgleich

- Die Inhomogenitäten werden auf ihre eigentlichen Werte gedreht. Dabei sind die Wippen grob von Hand so nachzuführen, dass die Seile nicht ganz stramm sind, aber auch nicht so weit durchhängen, dass sie von den Rollen rutschen. Die Schlitten über den Koeffizientenseilrollen werden so mitbewegt, dass die Seile vertikal verlaufen.
- Der Benutzer löst das System, indem er auf genau die gleiche Art und Weise auf die Ablageflächen drückt wie beim Nullabgleich (Abb. 4).

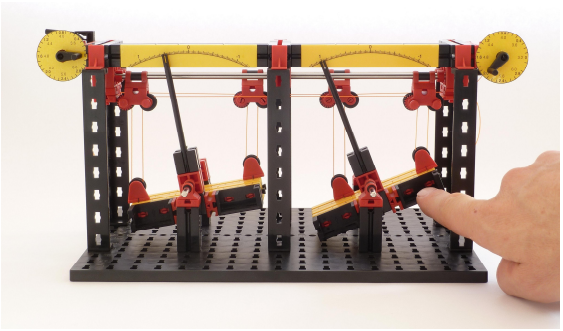


Abb. 4: Lösung des Gleichungssystems mit eingestellten Inhomogenitäten

Wenn man bei gleichen oder leicht veränderten Koeffizienten das Gleichungssystem noch einmal mit anderen Inhomogenitäten lösen möchte, können die ersten drei Schritte natürlich entfallen. Wird im zweiten Schritt zu Koeffizienten mit anderen Vorzeichen übergegangen, so ist es sinnvoll, die Inhomogenitäten zunächst nicht auf 0, sondern auf −0,1 zurück zu drehen. Dadurch hängt das Seil ein wenig durch, wenn der Radhalter die Nullposition passiert, und das Verstellen wird deutlich einfacher. Erst nach der Einstellung der Koeffizienten werden die Inhomogenitäten auf 0 gedreht.

Das Verfahren über den Nullabgleich ist einfach und liefert mit etwas Übung sehr gute Ergebnisse. Das eigentliche Ziel im fünften Schritt ist es, so zu drücken, dass beide Seile gleichzeitig gespannt sind. Wenn man die Wippen und Zeiger durch Drücken bewegt, so ist im Normalfall ein Seil gespannt und eines hängt durch. Man kann versuchen, den genauen Punkt, an dem beide Seile gespannt sind, optisch und durch Anlegen eines Fingers an die Seile

zu bestimmen. Meistens ist diese Methode jedoch aufwendiger und nicht genauer als das vorgeschlagene Verfahren über den Nullabgleich. Bei schwierig zu lösenden Systemen können beide Methoden kombiniert werden.

Die einzelnen Koeffizienten können in einem Bereich von −4 bis +4 eingestellt werden, die Inhomogenitäten je nach Koeffizientenwerten maximal in einem Bereich von −8 bis 8. Natürlich liegen die Koeffizienten und Inhomogenitäten praktisch auftretender linearer Gleichungssysteme in der Regel nicht in diesen Intervallen. Dann multipliziert man die einzelnen Gleichungen jeweils so mit einer Konstanten, dass der betragsmäßig größte Koeffizient ±4 wird. Dabei ändert sich die Lösungsmenge nicht. Die Inhomogenitäten kann man nach einem ersten Lösungsversuch noch anpassen. Z. B. kann man alle Inhomogenitäten gleichzeitig mit dem Faktor 10 multiplizieren, wenn die Zeiger nur sehr wenig ausgeschlagen haben. Man erhält dann in einem neuen Versuch Lösungen für $10\,x$ und $10\,y$, d. h. man muss die jetzt angezeigten Lösungskomponenten durch 10 dividieren, um die tatsächlichen Lösungskomponenten zu erhalten.

Durch Mehrfachanwendung kann man mit Kelvin theoretisch jedes eindeutig lösbare Gleichungssystem mit beliebiger Genauigkeit lösen. Wenn man das Ergebnis des ersten Lösungsversuchs in die linken Seiten der Gleichungen einsetzt und mit den vorgegebenen Inhomogenitäten vergleicht, erhält man den Fehler. Diesen Fehler kann man nun als neue Inhomogenitäten einstellen und erhält so im zweiten Schritt eine Korrektur für das erste Ergebnis. Für das korrigierte Ergebnis kann man dann wieder durch Einsetzen einen neuen Fehler berechnen, der wieder eine neue Korrektur liefert. Diese Selbstverbesserung kann man so oft wiederholen, bis die Genauigkeit den eigenen Anforderungen entspricht.

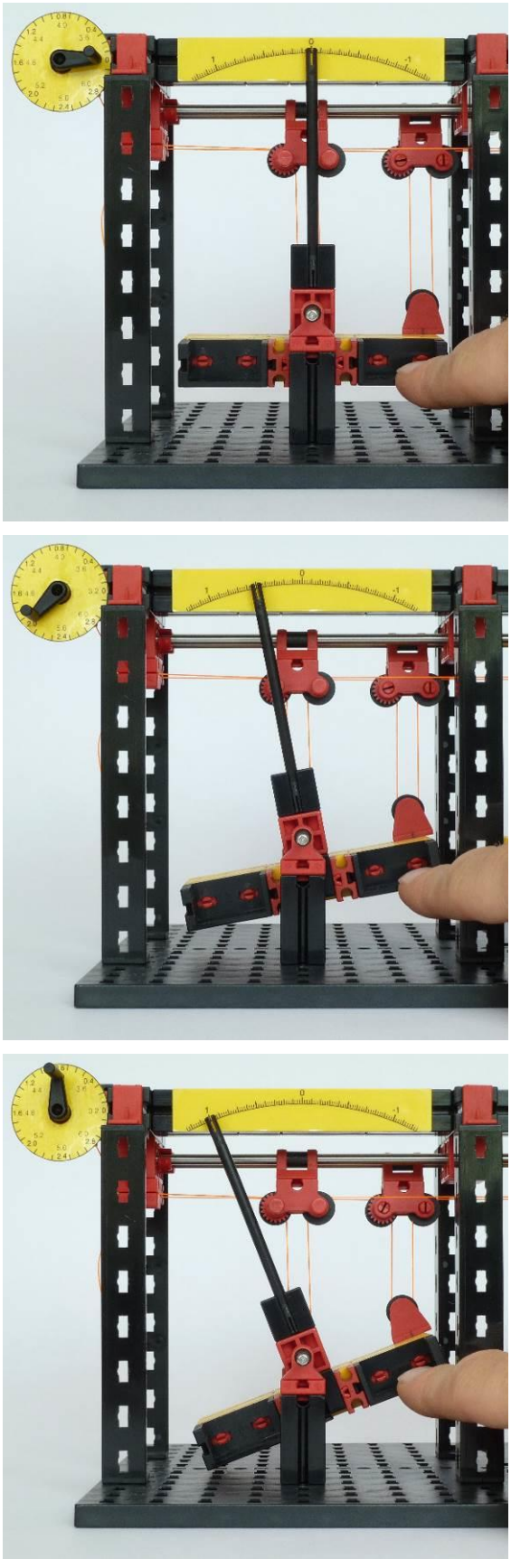


Abb. 5: Lösung der Gleichung $4\,x = b$

Funktionsweise

Um zu verstehen, wie Kelvin funktioniert, beginnen wir bei den Inhomogenitätsskalen. Diese zeigen die Länge des aufgekurbelten Seils an. Mit jedem Klick der Sperrklinke beim Drehen gegen den Uhrzeigersinn wird die Anzeige um 0,1 erhöht und eine konstante Länge Seil aufgekurbelt. Wir betrachten nun die lineare Gleichung

$$4\,x = b$$

Wir geben diese in Kelvin ein, indem wir den hinteren Radhalter auf der linken Wippe auf +4, alle anderen Radhalter auf 0 verschieben. Die rechte Wippe kann nun frei bewegt werden (die Unbekannte y kann beliebige Werte annehmen) und ist im Folgenden nicht von Bedeutung. Wir betrachten daher nur Kelvins rechte Hälfte.

Die Ergebnisskala ist nun so angelegt, dass die Lösung $x = b/4$ der Gleichung $4x = b$ angezeigt wird, wenn die Inhomogenität b eingestellt ist und mit dem Finger auf die rechte Ablagefläche der linken Wippe gedrückt wird, siehe Abb. 5.

Was passiert nun, wenn man den Koeffizienten z. B. von +4 auf +2 ändert? Nehmen wir an, wir ändern die Inhomogenität gleichzeitig von b auf $b/2$, dann ändert sich die Position der Wippe nicht. Die Höhe der Seilrolle über der horizontalen Ebene durch die Drehachse der Wippe ist nämlich immer gleich der Hälfte der Länge des aufgekurbelten Seils, siehe Abb. 6. Bei der Gleichung $2x = b/2$ ist nur halb so viel Seil aufgekurbelt, das wird aber dadurch kompensiert, dass die Koeffizientenseilrolle nur halb so weit von der Drehachse entfernt ist. Somit wird auch bei der Gleichung $2x = b/2$ die korrekte Lösung $x = b/4$ angezeigt. Das gleiche Argument kann man für alle anderen Koeffizienten analog durchführen. Damit ist klar, dass Kelvin alle Gleichungen der Form $ax = b$ korrekt löst.

Wenn nun der hintere Radhalter auf der linken Wippe auf +4 und der hintere Radhalter auf der rechten Wippe auf −2 ausgelenkt werden, zeigt Kelvin Lösungen der Gleichung $4\,x - 2\,y = b$ an, denn die Summe der halben Höhen der Seilrollen über der horizontalen Ebene durch die Drehachsen der Wippen ist gleich der Länge des aufgekurbelten Seils, wenn das Seil stramm ist.

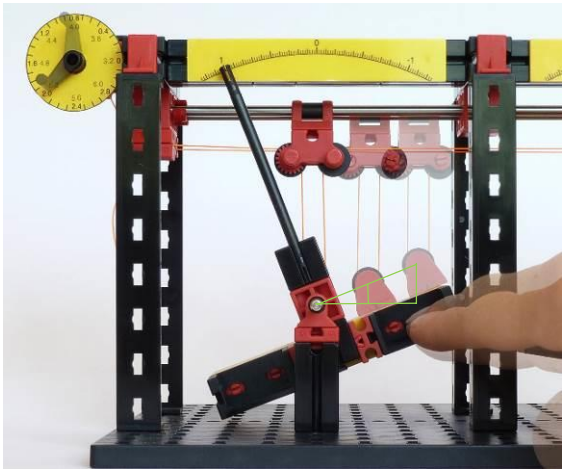


Abb. 6: Beim Übergang von 4x = 4 auf 2x = 2 ändert sich die Position der Wippe nicht

Stellt man auf dem vorderen Teil der Wippen eine zweite Gleichung ein, so wird das System aus beiden Gleichungen durch Kelvin gelöst, wenn beide Seile gleichzeitig stramm sind.

Kelvin löst übrigens immer ein Ungleichungssystem, ob die Seile gespannt sind oder durchhängen. Im Anfangsbeispiel löst er das System

$$4\,x - 2\,y \geq 0$$
$$-4\,x + 4\,y \geq 2$$

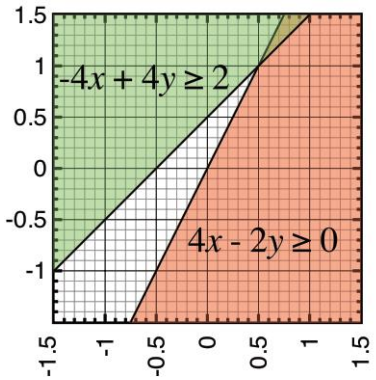


Abb. 7: Lösungsmenge (Ungleichungssystem)

Der Lösungssektor des Ungleichungssystems ist die Schnittmenge der beiden farbigen Halbebenen (Abb. 7). Von seiner Gestalt und Lage hängt ab, auf welche der Auflagefelder zur Lösung des Gleichungssystems gedrückt werden muss. In unserem Beispiel haben wir die Möglichkeit, die Variable x zu verkleinern, d. h. auf die rechte Auflagefläche der linken Wippe zu drücken, die Variable y wird sich dann automatisch verkleinern, sobald wir den linken Rand des Lösungssektors erreicht haben.

Die zweite Möglichkeit ist, die Variable y zu verkleinern, d. h. wie in Abb. 3 gezeigt auf die rechte Auflagefelder der rechten Wippe zu drücken. Dann verkleinert sich die Variable x automatisch, sobald wir den unteren Rand des Lösungssektors erreicht haben. Schließlich kann man mit etwas Gefühl gleichzeitig auf die rechten Auflagenfelder beider Wippen drücken. Dadurch kann man vermeiden, auf den Rändern der Lösungssektoren entlang zu wandern, wo immer ein Seil stramm und damit die Reibung groß ist.

In William Thomsons ursprünglichem Konzept wurden permanent stramme Seile durch das Auflegen konstanter Gewichte auf die Auflagefelder erzwungen, bei Wilburs Simultaneous Calculator durch eine nach unten hin verdoppelte Seilführung. Beide Methoden erhöhen den Anspruch an die Reibungsfreiheit der Seilrollen, an die Qualität der Lager und an die Starrheit des Rahmens unnötig und eignen sich für ein Baukastensystem wie fischertechnik nicht. Ein früher Prototyp meines Seilcomputers arbeitete mit Gewichten und bestätigte diese Aussage. Bei Systemen mit kleinem Öffnungswinkel des Lösungssektors verklemmte der Mechanismus regelmäßig.

Bauanleitung

In der folgenden Anleitung machen wir von den üblichen Bezeichnungen der fischertechnik-Bauteile Gebrauch, wie sie z. B. in der Einzelteilübersicht der Firma Knobloch zu finden sind. Die für Kelvin benötigten Teile sind in der Datei Teile.txt aufgelistet.

Skalen und Markierungen

Zur Herstellung der Skalen druckt man die Datei Kelvin.pdf auf eine transparente Klebefolie (z. B. Sattleford). Alternativ kann man die Skalen natürlich auch preisgünstiger auf Papier oder auf Overheadfolie drucken und mit doppelseitigem Klebeband befestigen. Beim Druckprozess darauf achten, dass keine unbeabsichtigte Verkleinerung oder Vergrößerung stattfindet (Größe 100%).

Die beiden Ergebnisskalen werden ausgeschnitten und jeweils zentriert auf eine gelbe Bauplatte 6×1 geklebt (Abb. 8).

Abb. 8: Aufgeklebte Ergebnisskala

Die vier Koeffizientenskalen werden ausgeschnitten und jeweils auf drei in Längsrichtung zusammengefügte gelbe Bausteine 30 geklebt (Abb. 9).

Die Zentrierung kann man daran erkennen, dass die ±1, 5-Teilstriche exakt über der kleinen Nut zwischen den Bausteinen 30 liegen. Damit die Radhalter in der Längsnut noch frei verschoben werden können, sollte man die Koeffizientenskalen nicht zu dicht an diese Nut kleben.

Abb. 9: Aufgeklebte Koeffizientenskala

Die zwei Inhomogenitätsskalen werden mit einer runden Schere ausgeschnitten. Falls Klebefolie verwendet wird, werden sie zuvor auf gelben Karton geklebt (Abb. 10).

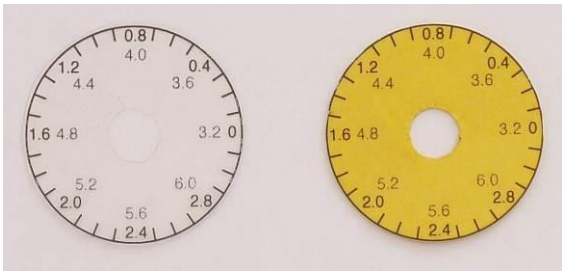


Abb. 10: Inhomogenitätsskalen

Die vier Radhalter haben eine klar erkennbare Mittellinie. Diese wird mit einem feinen schwarzen Stift im unterem Sockelbereich möglichst gerade nachgezeichnet, siehe Abb. 11. Die Markierung wird zum Einstellen der Koeffizienten benötigt.

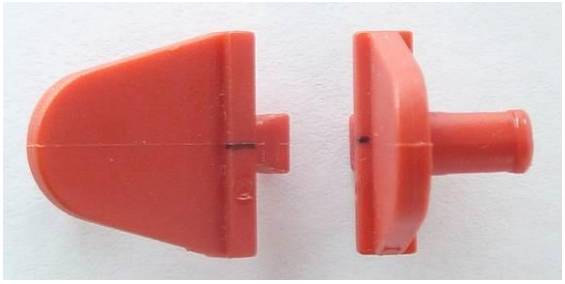


Abb. 11: Markierung der Radhalter

Wippen

Der Zusammenbau der Wippen ergibt sich aus Abb. 12 und 13.

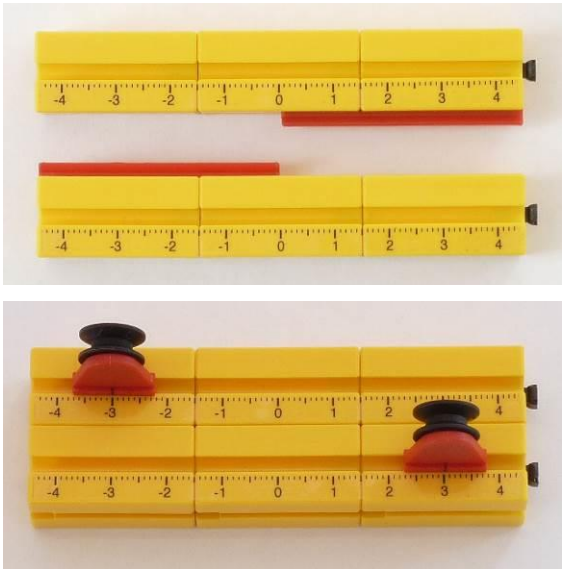


Abb. 12: Zusammenbau der Koeffizientenblöcke

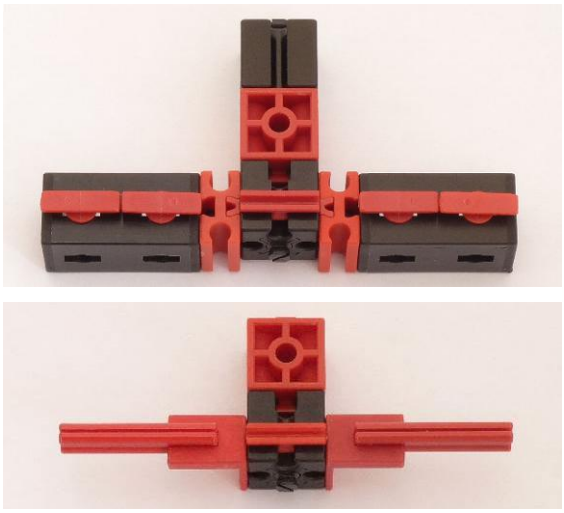


Abb. 13: Zusammenbau von Zeiger- und Lagerblock

Zeigerblock und hinterer Lagerblock sollten sorgfältig zentriert werden. Die Zentrierung des Zeigerblocks erkennt man daran, dass die Verbindungsstellen der gelben Bausteine mit den Kanten der Bausteine 7,5 übereinstimmen (Abb. 14).

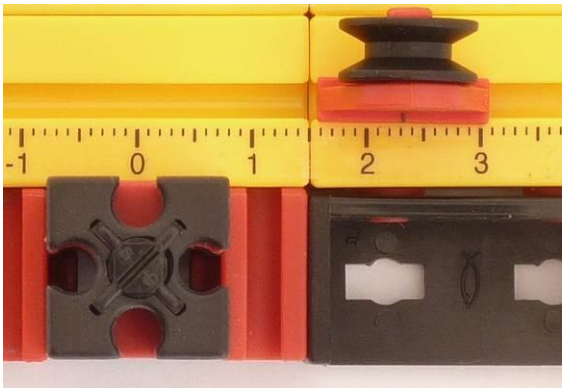


Abb. 14: Zentrierung des Zeigerblocks

Beim hinteren Lagerblock kann man einen losen Baustein 7,5 zur Hilfe nehmen, um die Zentrierung zu überprüfen. Wichtig ist außerdem die korrekte Höhe der Bausteine 15 mit Bohrung. Deren Bohrungen müssen in einer Ebene mit den Achsen der Radhalter liegen. Um das zu erreichen, wird der Koeffizientenblock auf eine ebenen Tischplatte gelegt und fest aufgedrückt, so dass die Unterkante der Zapfen der unteren Bausteine 15 plan mit den Bausteinen 30 und der Tischplatte abschließen (Abb. 15).

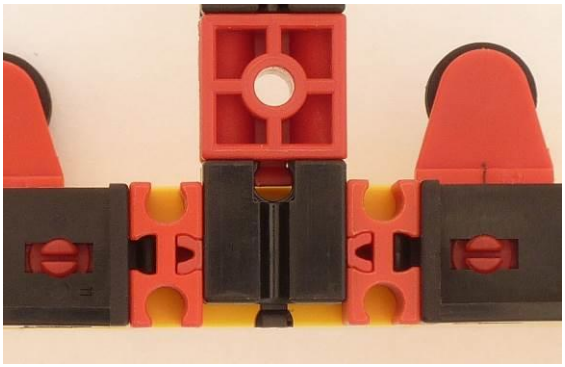


Abb. 15: Plan abschließender Zapfen

Montieren der Wippen

Die vier Wipphalter werden gemäß Abb. 16 zusammengebaut. Die Lagerhülse wird in eine Richtung etwa 1 mm aus der Gelenkwürfelklaue herausgeschoben, um einen großflächigen Kontakt zwischen Wippe und Wipphaltern vollständig auszuschließen.

Abb. 16: Wipphalter

Anschließend werden die Wipphalter an den Positionen E4, M4, E9 und M9 in die Grundplatte eingeschoben, allerdings nicht bis zum Anschlag, sondern nur so weit, dass sie fest sitzen (die Einschubtiefe wird im Verlauf der Konstruktion angepasst).

Die beiden Wippen werden zwischen je zwei Wipphalter montiert, indem man die Achsen 50 lose durch die Lagerhülsen in Bohrungen der Bausteine 15 mit Bohrung einschiebt.

Abb. 17: Fertige Wippe

Die Wippen (Abb. 17) müssen ohne nennenswerte Reibung und mit nur wenig Spiel möglichst gut pendeln können (Abb. 18).

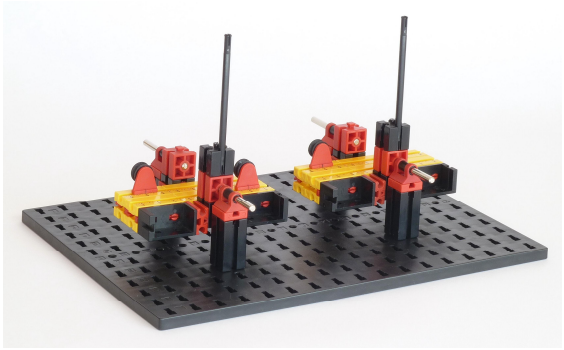


Abb. 18: Grundplatte mit Wippen

Oberteil

Zunächst werden die beiden Skalenträger zusammengebaut und die Ergebnisskala eingeschoben und zentriert (Abb. 19).

Abb. 19: Skalenträger

Dabei kann ein Baustein 7,5 als Hilfsmittel benutzt werden. Als zweites werden die Achsenträger zusammengebaut (Abb. 20).

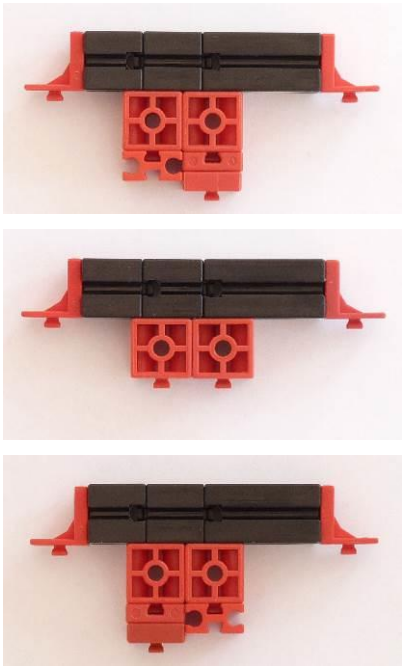


Abb. 20: Achsenträger links, mitte und rechts

Die genaue Position der Bausteine 15 mit Bohrung wird später eingestellt. Die Skalenträger werden nun zunächst 7,5 mm in die Achsenträger eingeschoben (Abb. 21). Die Einschubtiefe muss eventuell später noch einmal leicht korrigiert werden, um ein optimales, reibungsloses Vorbeistreichen der Zeiger an der Skala zu gewährleisten.

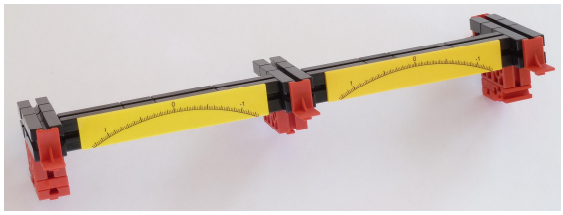


Abb. 21: Oberteil ohne Achsen

Als nächstes werden die Schlitten gemäß Abb. 22 zusammengebaut. Damit die Schlitten problemlos aneinander vorbei gleiten können, ist es erforderlich, die Rastachsen 20 nur bündig in die Rollenböcke einzuschieben. Wenn möglich, sollten die Rastachsen 20 durch die grauen Kunststoffachsen 15 ersetzt werden, die allerdings nur noch gebraucht erhältlich sind (Abb. 23).

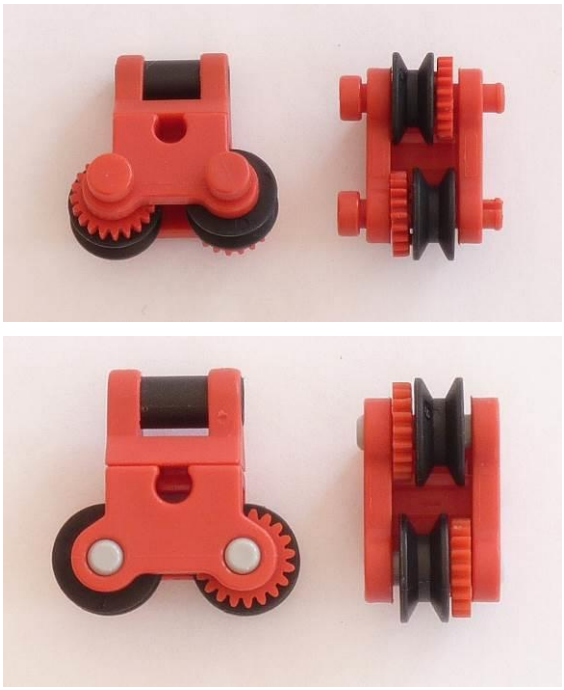


Abb. 22: Alternative Schlittenkonstruktionen

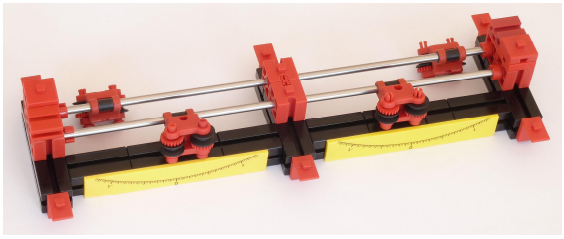


Abb. 23: Oberteil mit Achsen und Schlitten

Die Achsen 125 werden durch die Bausteine 15 mit Bohrung in den Achsenträgern von außen in das Oberteil zunächst nur teilweise eingeschoben. Dann werden die Schlitten auf die Achsen gesteckt und die Achsen weiter eingeschoben, bis sich jeweils zwei in den beiden mittleren Bausteinen 15 mit Bohrung treffen. Anschließend werden die Achsen mit Klemmringen an der Innenseite der äußeren Bausteine 15 mit Bohrung fixiert.

Die Winkelträger 120 werden wie abgebildet in die Positionen A4, I4, Q4, A9, I9 und Q9 der Grundplatte eingeschoben. Das Oberteil wird angebracht. Beim Bewegen der Wippen sollten die Zeiger direkt vor der Skala vorbeigleiten, ohne diese zu berühren. Gegebenenfalls müssen die Skalenträger etwas weiter nach hinten gescho-

ben oder nach vorne gezogen werden. Die Bausteine 15 mit Bohrung unter den Achsenträgern werden nun so positioniert, dass sich die Achsen 125 senkrecht über den Koeffizientenseilrollen befinden.

Seile

Kelvin funktioniert mit originalen fischertechnik-Seilen, allerdings wird die Bedienung deutlich vereinfacht, wenn handelsüblicher Leinen- oder Ramiezwirn verwendet wird, der einen wesentlich höheren Elastizitätsmodul besitzt. Die beiden Seile sollten 90 cm lang sein. Jedes Seil wird mit Schlingen an einer Seiltrommel befestigt. Das lose Ende wird von oben durch die mittlere Bohrung, anschließend von unten durch die seitliche Bohrung gesteckt und möglichst festgezogen (Abb. 24).

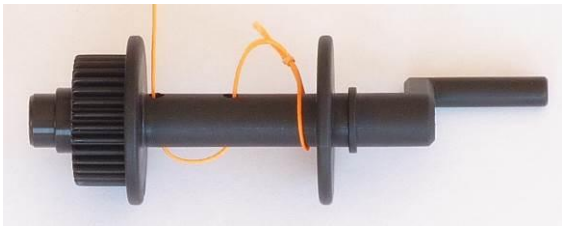


Abb. 24: Seilbefestigung

Die Seiltrommeln werden in die Seilwindengestelle gesteckt. Die Kurbel der linken Seilwinde wird bei aktiver Sperrklinke so gedreht, dass sie in Richtung der Zapfen des Seilwindengestells, die Kurbel der rechten Seilwinde dagegen so, dass sie von den Zapfen weg zeigt. Die Inhomogenitätsskalen werden mit einem kleinen Stück doppelseitigem Klebeband so an den Seilwinden befestigt, dass die Kurbeln sich in Nullposition befinden (Abb. 25).

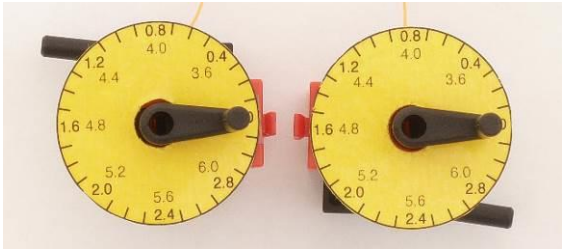


Abb. 25: Winden mit aufgeklebten Skalen

Die Positionierung sollte möglichst exakt erfolgen. Beim Drehen der Trommel mit aktivierter Sperrklinke sollte die Kurbel in den Rastpositionen genau auf die Teilstriche der Skala zeigen. Die beiden Seilwinden werden wie in Abbildung 26 gezeigt am Oberteil befestigt.

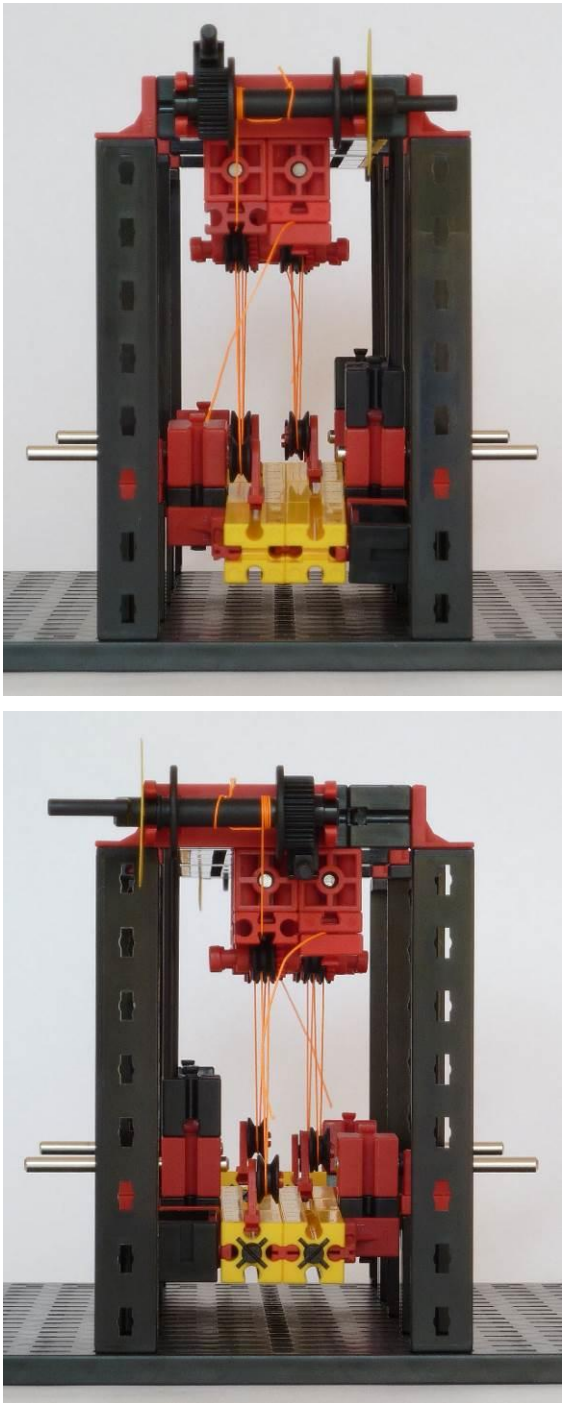


Abb. 26: Befestigung der Seilwinden – Ansicht von links (oben) und rechts (unten)

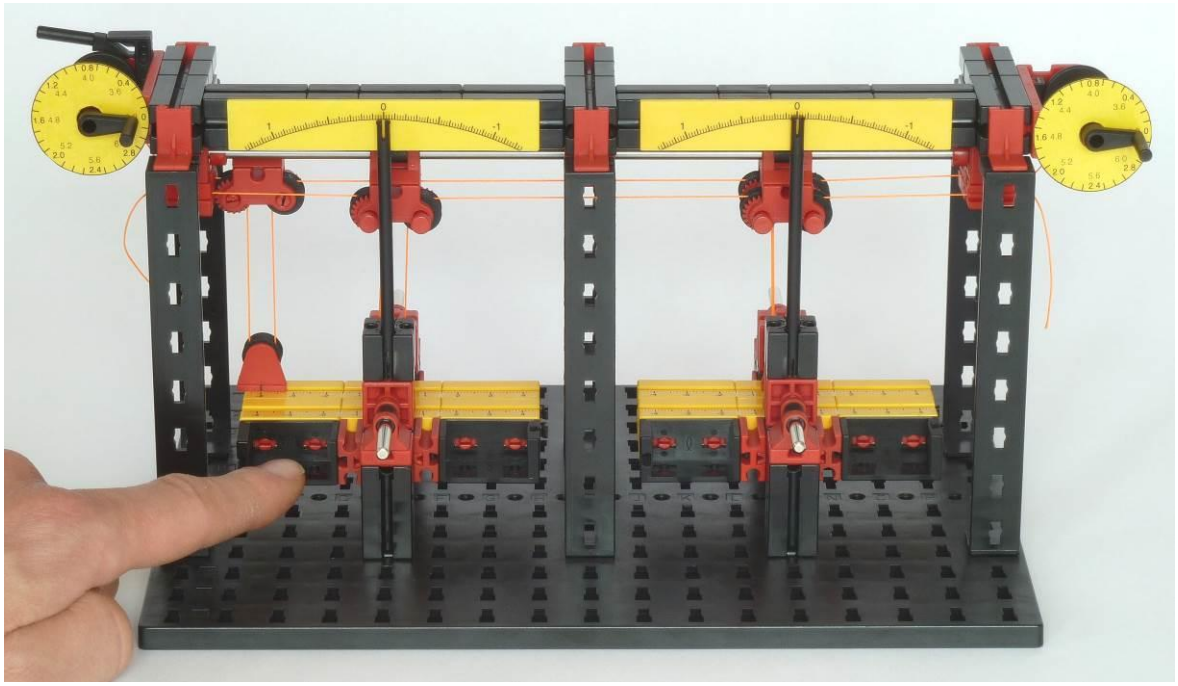


Abb. 27: Justierung von Kelvin

Seilführung

Die vier Radhalter werden in den Nuten des Koeffizientenblocks alle auf Position 0 geschoben. Das hintere Seil wird nun durch die Nut des Bausteins 7,5 am linken Achsenträger über die Rollen der hinteren Schlitten und die hinteren Rollen der Wippen zur rechten Seite geführt, siehe Abb. 27.

Die beiden hinteren Schlitten werden senkrecht über die Radhalter geschoben. Anschließend wird bei stramm, aber nicht zu stramm gezogenem Seil die Kurbel der linken Winde exakt drei volle Umdrehungen gegen den Uhrzeigersinn gedreht. Die Windungen auf der Seiltrommel müssen direkt nebeneinander auf der Seiltrommel liegen. Dann wird der untere Baustein 5 am rechten Achsenträger entfernt, das immer noch stramm gehaltene Seil zweimal um den Zapfen des festen oberen Bausteins 5 gewickelt und der untere Baustein 5 seitlich wieder aufgeschoben (siehe Abb. 26). Analog wird das vordere Seil von rechts nach links geführt und am linken Achsenträger befestigt.

Justierung

Für ein gutes Funktionieren des Computers sind korrekte Seilspannungen wesentlich. Die Spannung eines Seils kann eingestellt werden, indem die beiden Bausteine 5, an denen es befestigt ist, gemeinsam gegenüber den Bausteinen 15 mit Bohrung in Seilrichtung verschoben werden. Die Einstellung erfordert etwas Fingerspitzengefühl. Wesentlich ist es, spüren zu können, wann ein Seil stramm gezogen ist. Das ist der Fall, wenn sich trotz leichter Änderung der Zugkraft das Seil, die Wippen und damit die Zeiger nicht bewegen.

Zur Justierung wird der hintere linke Koeffizient auf -4 gestellt und der entsprechende Schlitten senkrecht über der Seilrolle positioniert. Die linke Wippe wird links nach unten gedrückt, bis das Seil stramm ist. Bei richtig eingestellter Seilspannung zeigt der Zeiger den Wert 0, ansonsten muss korrigiert werden (siehe Abb. 27). Danach wird der hintere linke Koeffizient auf +4 gestellt.

Die linke Wippe wird jetzt rechts mit dem gleichen Druck wie zuvor auf der anderen

Seite nach unten gedrückt. Der Zeiger sollte ohne weitere Einstellung wieder den Wert 0 anzeigen. Ist das nicht der Fall, liegt eine Asymmetrie vor. Zu überprüfen ist dann der gesamte Aufbau (z. B. können die als Zeiger verwendeten V-Achsen nicht ganz gerade sein). Eine Anpassung kann notfalls erfolgen, indem die Erebnisskalen in den Nuten etwas nach links oder rechts verschoben werden.

Anschließend wird der linke hintere Koeffizient auf 0 zurückgestellt. Das Verfahren wird für den rechten hinteren Koeffizienten wiederholt. Hier sollte die Seilspannung eigentlich schon passen. Ist das nicht der Fall, muss zwischen links und rechts ein Kompromiss gefunden werden. Bei einer Differenz von mehr als einem kleinen Teilstrich (also 0,025) auf der Ergebnisskala sollte der gesamte Aufbau nach Fehlerursachen untersucht werden.

Abschließend wird für das vordere Seil analog vorgegangen. Kelvin ist nun fertig zum Einsatz.

Aufgaben

Wer Kelvin nachgebaut hat, möchte vielleicht gleich noch weitere Gleichungssysteme lösen. Am besten ist es natürlich, frei zu experimentieren.

Für alle diejenigen, die ein schnelles Erfolgserlebnis haben möchten, sind hier noch drei weitere Beispiele angegeben, bei denen Koeffizienten und Inhomogenitäten sofort zu Kelvin passen:

$$\begin{aligned} -1\,x + (-4)\,y &= 0.4 \\ 3\,x + 4\,y &= 1.6 \end{aligned}$$

$$\begin{aligned} 4\,x + (-3)\,y &= 0 \\ -4\,x + (-3)\,y &= 6 \end{aligned}$$

$$\begin{aligned} -4\,x + (-2)\,y &= 1.2 \\ -1\,x + (-4)\,y &= 3.8 \end{aligned}$$

Ob man gute Lösungen gefunden hat, kann man leicht durch Einsetzen überprüfen.

Quellen

[1] Petzold, H., *Wilhelm Cauer and his mathematical device,* in: Exposing Electronics, Michigan State University Press 2003, S. 45–73.

[2] Thomson, W., *On a machine for the solution of simultaneous linear equations*, Proc. R. Soc. 28 (1878–1879), 111–113.

[3] Wilbur, J., *The mechanical solution of simultaneous equations*, J. Franklin Inst. 222 (1936), 715–724.

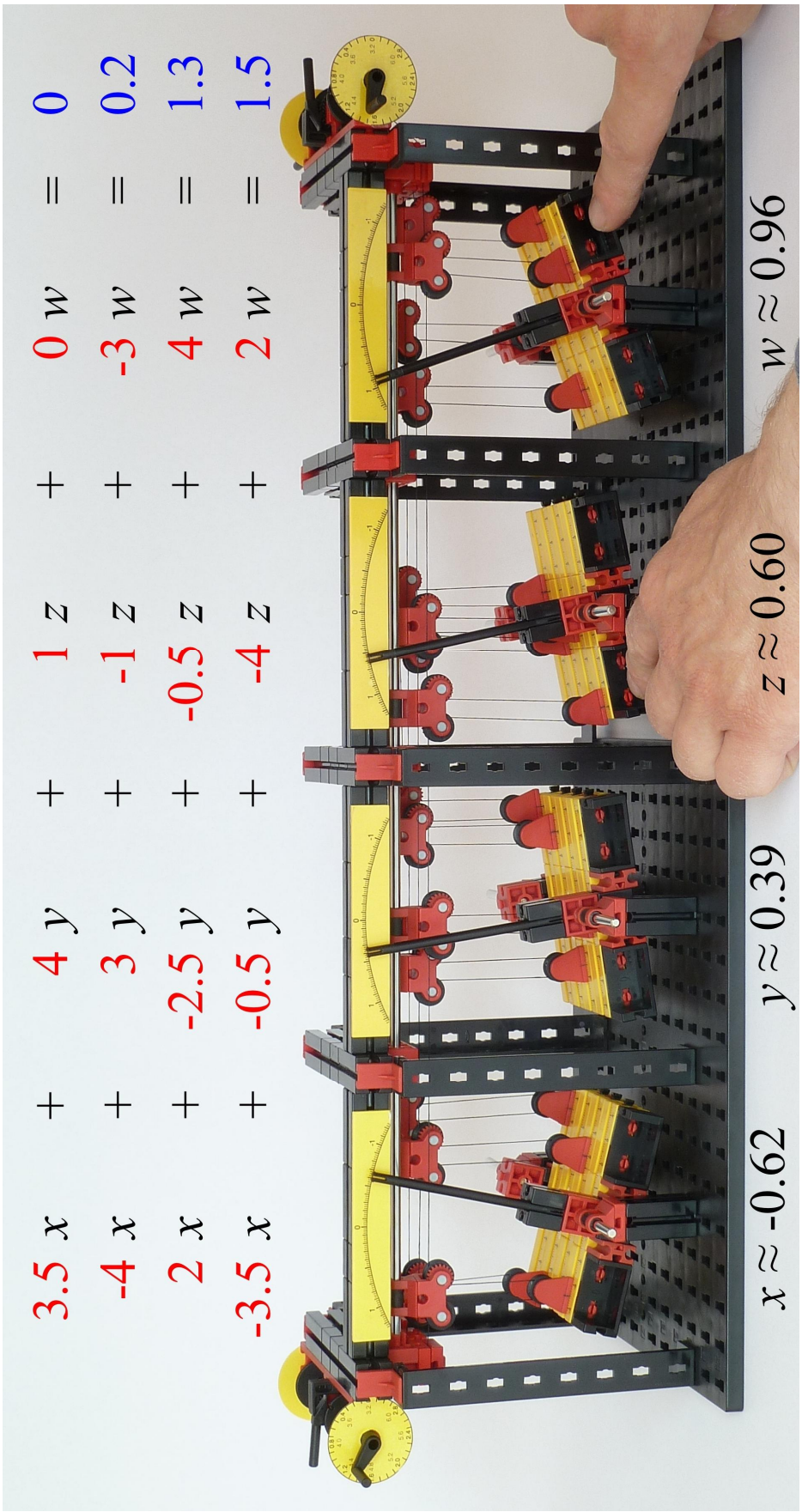


Abb. 28: 4×4-Kelvin zur Lösung von Gleichungssystemen mit vier Unbekannten