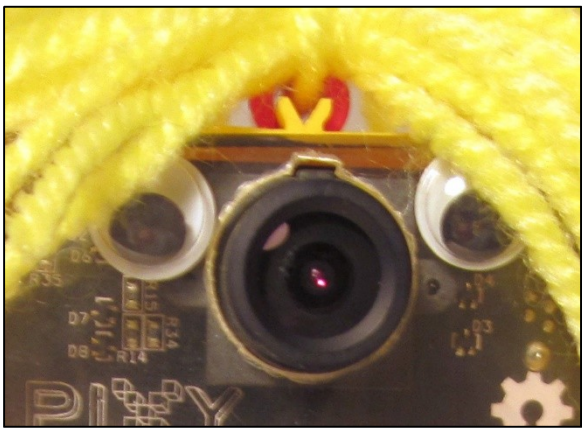
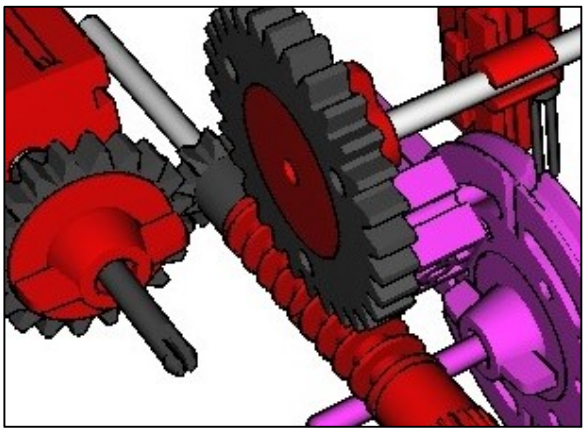
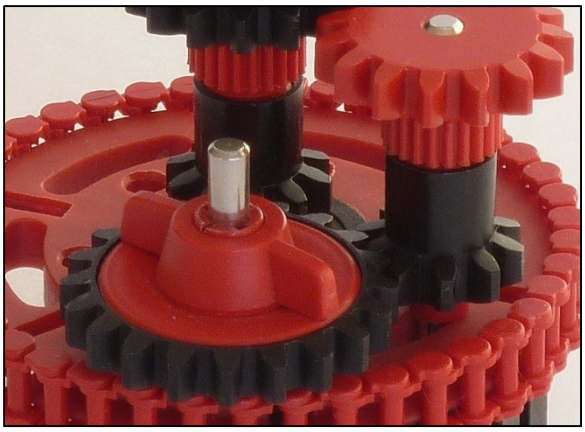
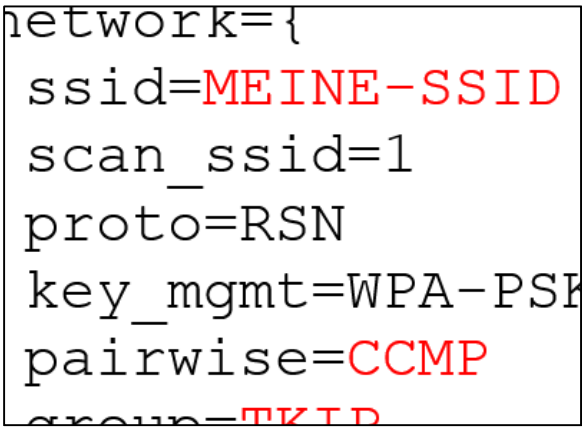
ft:pedia

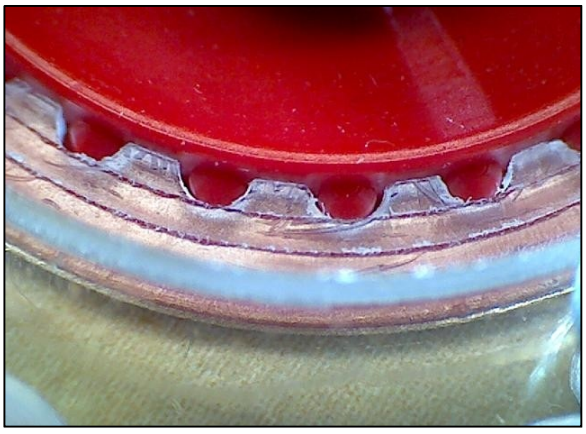
Heft 4/2014











Herausgegeben von
Dirk Fox und Stefan Falk

ISSN 2192-5879

Editorial

Durchbruch

Dirk Fox, Stefan Falk

Wie haben wir das herbeigesehnt! fischertechnik hat es geschafft – seit über 30 Tagen steht der Baukasten „Dynamic“ in der Spielzeug-Top-100-Liste von Amazon; ganze vier Kästen haben es in die Top500 geschafft. Im Windschatten der Auszeichnung des großen Bruders, des Baukastens „Dynamic XL“ als „Spielzeug des Jahres 2014“ wird fischertechnik also am 24.12. dieses Jahres die deutschen Kinderzimmer erobern.

Das ist weder Glück noch Zufall, sondern das Ergebnis einer ausgefeilten und konsequenten Produktpolitik. Den Innovations-Volltreffer der Flexschienen hat fischertechnik geschickt genutzt: die drei Kugelbahn-Kästen „Rolling Action“, „Dynamic“ und „Dynamic XL“ decken den gesamten Preiskorridor ab, den Eltern und Großeltern in ein Weihnachtsgeschenk zu investieren bereit sind – und bieten fischertechnik-Einsteigern zudem die Aussicht, später mit Ergänzungskästen die Konstruktionsmöglichkeiten zu erweitern. Alle drei Kästen sind zudem thematisch wenig erklärungsbedürftig, nicht pädagogisch überfrachtet und dank der Auszeichnung ohnehin über jeden Zweifel erhaben – die perfekte „Einstiegsdroge“ also.

Die hohe Kunst wird jetzt darin bestehen, den Erstkäufern eines Kugelbahn-Kastens einen Zugang zum „Universum fischertechnik“ zu eröffnen. Eine wichtige Rolle werden dabei Modelle spielen, die sich im Wesentlichen mit den Bauteilen eines „Dynamic“ oder „Dynamic XL“ konstruieren lassen, aber über die Kugelbahnen hinausgehen. Aber auch mechanische Anbauten für die Kugelbahnen, die man mit den Bauteilen aus anderen Kästen wie dem „Mechanic & Static“, einer „Creative Box“ oder „Pneumatic 3“ realisieren kann, dürften unterstützend wirken.

Mit der Veröffentlichung von Modellanleitungen für größere Kugelbahnen aus „Dynamic“ und „Rolling Action“ bzw. „Dynamic XL“ hat fischertechnik (wenn auch etwas versteckt auf der Webseite) bereits einen Schritt in diese Richtung getan. Jetzt sollten weitere Gratis-Modellanleitungen folgen, damit die frisch entfachte Lust auf fischertechnik nicht in einer Kinderzimmerecke neben einem verstaubten Lego-Modell endet.

Aber auch ihr seid gefragt: Modelle wie die fantastischen Kugelbahnen, die u. a. auf der Convention 2014 zu bewundern waren, und ganz neue Ideen für den Einsatz der Flexschienen sind geeignet, den Kreis der fischertechnik-Fans wirksam zu vergrößern.

Denn eines wissen wir ja aus eigener Erfahrung: Wer einmal Blut geleckt hat, kommt von fischertechnik nicht mehr los.

Jetzt wünschen wir euch aber erstmal eine wunderschöne Weihnachtspause – mit viel Muße für ungestörte fischertechnik-Stunden.

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?
Clubdag in Veghel	07.02.2015	Veghel (NL)

Hinweis

Weil so etwas einfach gefehlt hat, gibt es jetzt den fischertechnik-Blog: http://www.fischertechnik-blog.de

Impressum

http://www.ftcommunity.de/ftpedia

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

Autoren: Christian Andersch (Carrera124), Gerhard Birkenstock (gggb), Stefan Falk (steffalk), Dirk Fox (Dirk Fox), Johann Fox, Andreas Gail, Werner Hasselberg, Raphael Jacob (ski7777), Thoamas Püttmann (geometer), Andreas Tacke (TST), René Trapp, Dirk Uffmann (Uffi), Dirk Wölffel (DirkW).

ft-Geschichte

fischertechnik im Spielwarenkatalog (1982-88)

Christian Andersch

Kaum zu glauben, aber es gab eine Zeit, in der man sich nur über so genannte ‚Kataloge' über das Spielwarenangebot eines Herstellers informieren konnte. Und die gab es nicht etwa zum Download, sondern ausgedruckt auf Papier in so genannten ‚Spielwarenläden' ... und waren manchmal sogar vergriffen.

Der klassische Spielwarenladen stirbt zunehmend aus, zu erdrückend ist die Online-Konkurrenz. Vor dem Siegeszug des Internetshoppings hingegen waren Spielwarenhändler und Kaufhäuser mit Spielwarenabteilung nahezu die einzigen Bezugsquellen für fischertechnik. Sowohl für die Waren an sich, als auch für Informationen.

Üblicherweise lagen im Herbst jeden Jahres gedruckte Kataloge zur kostenlosen Mitnahme aus.

Abb. 1: Katalog Idee & Spiel (1982)

Die junge Kundschaft bediente sich und nicht wenige der angepriesenen Spielzeuge dürften sich auf den Wunschzetteln fürs kommende Weihnachtsfest wiedergefunden haben.

Insbesondere die fischertechnik-Modellbaukästen mit den 1982 eingeführten vollbeweglichen Figuren wurden darin ansprechend in Szene gesetzt (siehe Abb. 1 bis 4).

Abb. 2: Katalog Idee & Spiel (1983)

Beeindruckend sind die Preise und Preissteigerungen in jenen Jahren. War der Baukasten „Start 100“ im Jahr 1982 noch für 69 DM zu haben (Idee & Spiel, Abb. 1), so stieg der Preis im 1983er Katalog bereits auf 79,90 DM (Abb. 2).

Abb. 3: Idee & Spiel (1984)

Abb. 4: Spielzeugring (1984)

Ähnlich bei der Hebebühne: 1982 noch für 39 DM erhältlich (Vedes, Abb. 5), aber 1984 bereits mit 45 DM ausgezeichnet (Abb. 6). Das entspricht einer Preiserhöhung um 14-15% – allerdings lag die Inflationsrate 1982 auch bei 5,3% und 1983 immerhin noch bei 3,3%.

Offenbar spürte auch Lego die Konkurrenz (oder witterte die Chance) und konterte die Hydraulik-Fahrzeuge von fischertechnik (Abb. 1, 2, 4, 5) erstmals mit einem Lego Technic-Fahrzeug mit Pneumatik (Abb. 4).

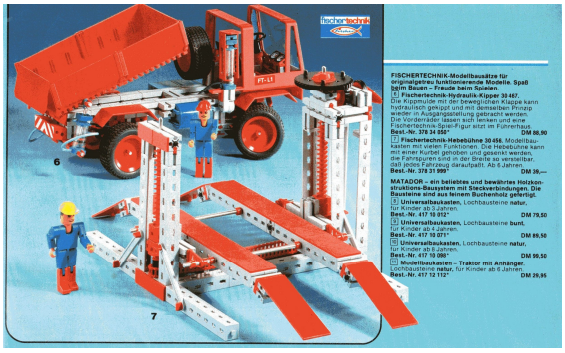


Abb. 5: Vedes (1982)

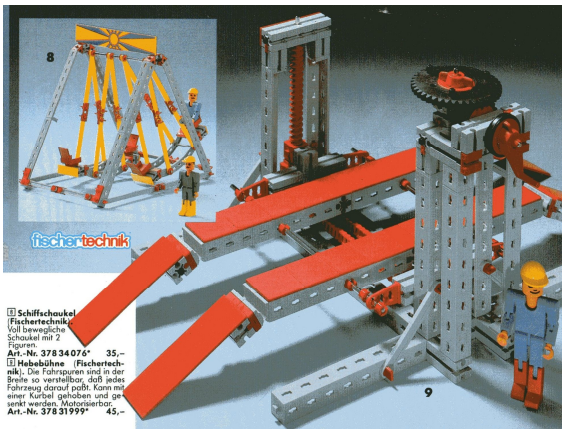


Abb. 6: Vedes (1984)

1985 fand das neue Flaggschiff, der Computing-Baukasten, Einzug in die Spielwarenkataloge: bei Vedes für 199 DM, bei Idee & Spiel ohne Preisangabe (Abb. 7, 8).

Auch die Antwort auf Legos Pneumatik, den von fischertechnik zusammen mit Festo entwickelten Pneumatik-Kasten sowie den Kompressor-Anhänger, die beide heute nur noch zu Sammlerpreisen auf dem Gebrauchtmarkt erhältlich sind und von ihren neuen Besitzern wie Schätze gehütet werden, findet man in diesem Jahr erstmals in einem Spielwarenkatalog (Abb. 8).

Abb. 7: Vedes (1985)

In den Katalogen von 1986 bis 1988 findet sich praktisch kein fischertechnik-Bildmaterial mehr. Höchstwahrscheinlich ist das darauf zurückzuführen, dass in diesen Jahren kaum „echte“ Neuheiten erschienen sind, sondern nur Ergänzungs- und Service-Kästen.

Abb. 8: Idee & Spiel (1985)

Referenzen

[1] Christian Andersch: *Die ‚neue fischertechnik‘ – 1989-1994*. ft:pedia 3/2014, S. 4-10.

Modell

Mini-Modelle (Teil 5): Traktor

René Trapp

Tuning für die „Straßenwalze“.

Als Grundmodell für unseren kleinen Traktor dient der fischertechnik-Bausatz „Straßenwalze“ [1], auch als „BiFi-Traktor“ erschienen [2]. Noch zwei unscheinbare Teile (Abb. 1) dazu und schon hat das kleine Modell einen fetten Auspuff und ein Cockpit (Abb. 2).

Abb. 1: Die Straßenwalze und das Tuning-Kit

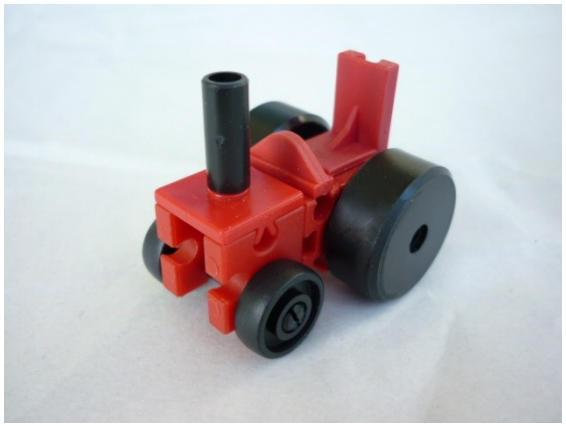


Abb. 2: Der TÜV kann kommen

Und hier noch die komplette Stückliste:

Stück	ft-Nr.	Bezeichnung
1	37636	Rollenlager
1	32064	Baustein 15 mit Bohrung 4
1	38423	Winkelstein 10x15x15
1	37468	Baustein 7,5
1	31124	Radachse mit Platte
2	36573	Rad 14
1	36919	V-Achse 28
1	38413	Kunststoffachse 30
2	36574	Rad 23, schwarz
1	36819	Lagerhülse
1	31602	Kufe

Quellen

[1] Bauanleitung 104596: Giveaway „Straßenwalze“

[2] Bauanleitung 39070: Giveaway „BiFi Spiel & Spaß mit fischertechnik“

Modell

Mini-Modelle (Teil 6): Bagger

Johann Fox

Und noch ein Mini-Modell: Diesmal für die kleine Sandkiste…

Leider passt der Mini-Bagger nicht mehr in die „Unter-20-Teile-Kategorie“, hat aber trotzdem den Charakter eines Mini-Modells. Vom Maßstab her ist er aber etwas größer als zum Beispiel der Mini-Panzer [4]. Hier sieht man den Bagger in der Gesamtansicht.

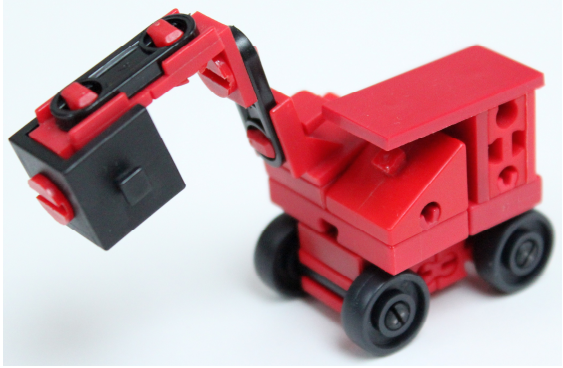


Abb. 1: Mini-Bagger Gesamtansicht

Alternativ kann man die Räder 14 auch durch Seilrollen ersetzten. Damit wird der Bagger zu einem Schienenbagger.

Der Arm des Baggers besteht im Wesentlichen aus schwarzen I-Streben 15, Winkellaschen und Verschlussriegeln. Dadurch wirkt er schlanker als mit S-Riegeln. Die Schaufel bildet ein Winkelträger 15 mit zwei Zapfen.

Leider kann man den Arm weder pneumatisch ansteuern noch drehen oder schwenken.

Abb. 3: Baggerarm mit Schaufel

Das überdachte Führerhaus ist mit dem Arm auf eine Art Plattform gebaut (Abb. 4).

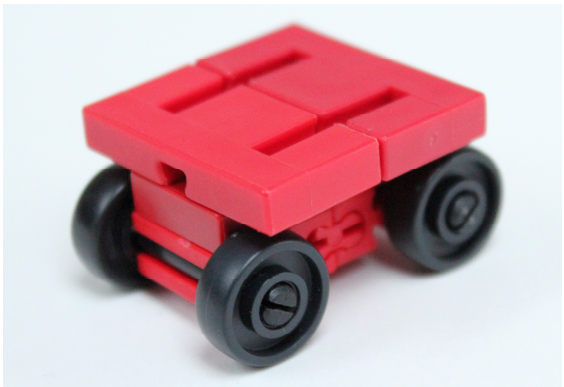


Abb. 4: Plattform von oben

Unten an der Plattform (Bauplatte 15x30x5 mit drei Nuten) wird das Fahrgestell mit zwei Bausteinen 5 mit zwei Zapfen angebracht.

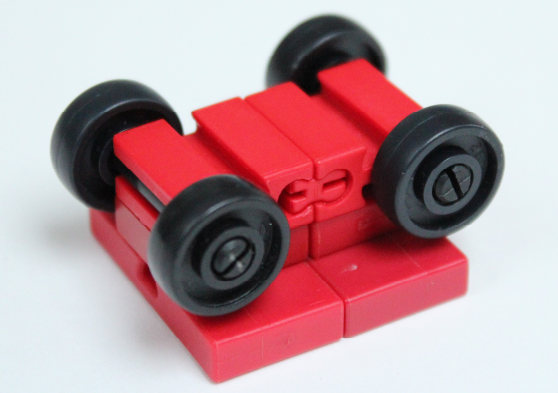


Abb. 5: Befestigung des Fahrgestells

Wer den Bagger nachbauen will, dem empfehle ich, mit dem Arm zu beginnen (Abb. 3), dann die Vorderachse mit Führerstand (Abb. 6) und zum Schluss die hintere Achse mit Aufbau (Abb. 7) zu bauen und danach alle Teile zusammen zu stecken.

Abb. 6: Vordere Achse mit Führerstand

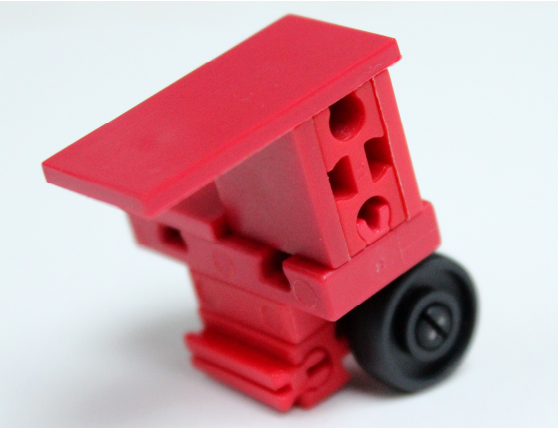


Abb. 7: Hintere Achse mit Aufbau

Zum Schluss noch die Einzelteilliste:

Stück	ft-Nr.	Bezeichnung
1	31012	Winkelstein 30 Grad rechtwinklig
1	31060	Verbindungsstück 15
2	31670	Winkellasche
1	31674	Adapterlasche
1	31981	Winkelstein 15 Grad
2	31982	Federnocken
1	36323	S-Riegel 4 rot
4	36573	Rad 14 schwarz
2	36914	I-Strebe 15 schwarz
2	36919	V-Achse 4*28
1	36950	Winkelträger 15 mit 2 Zapfen schwarz
4	37232	Verschlussriegel
2	37238	Baustein 5 mit 2 Zapfen
3	37468	Baustein 7,5
1	38241	Bauplatte 15x30 rot
3	38246	Bauplatte 15x15 rot
2	38428	Bauplatte 15x30x5 mit 3 Nuten

Bisher erschienen:

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

[2] Johann Fox: *Minimodelle (Teil 2): Panzer*. ft:pedia 2/2014, S. 18-19.

[3] René Trapp: *Minimodelle (Teil 3): Scheinwerfer*. ft:pedia 3/2014, S. 11.

[4] Johann Fox: *Minimodelle (Teil 4): Panzer*. ft:pedia 3/2014, S. 12-13.

Tipps & Tricks

ft-Spezialteile made by TST (Teil 10)

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 das Impulsrad – und eine modifizierte Version.

Vorneweg erst einmal etwas Grundlegendes. Wofür benötigt man eigentlich ein Impulsrad?

Man benötigt es, wie der Name schon sagt, zum Ermitteln von Impulsen. Es wird in der Hauptsache in Verbindung mit dem Robo Interface oder dem Robo TX Controller eingesetzt, um über die Zahl der Umdrehungen einer Achse beispielsweise die Strecke zu bestimmen, die ein Reifen zurückgelegt hat, die Verschiebung einer Schneckenmutter auf einer Schnecke oder die von einem Hubgetriebe bewältigte Distanz [1].

Es gibt zwei verschiedene Ausführungen des Impulsrads: eines mit vier Schaltnocken (Z4, 37157) und ein älteres mit fünf (Z5, 35995). Beide sind standardmäßig für das Rastachsensystem konzipiert (Abb. 1).

Abb. 1: Impulsräder Z5 und Z4

Hat man ein Modell gebaut, bei dem z. B. mit einem Motor eine Bewegungsachse angetrieben wird, kann über das Impulsrad und einen Taster die Achsbewegung abgefragt werden. Diese kann dann im Interface resp. im Controller über den Zähleingang verarbeitet werden.

Dabei lassen sich verschiedene Varianten einstellen. So kann man über die ermittelten Impulse z. B. ein Roboter um eine festgelegte Strecke bewegen. Je nachdem, ob man einen Schnecken- oder ein Kettenantrieb benutzt, muss man dann noch rechnerisch ermitteln, wie viele Impulse für die gewünschte Strecke benötigt werden. Abb. 2 zeigt die Ansteuerung in Robo Pro.

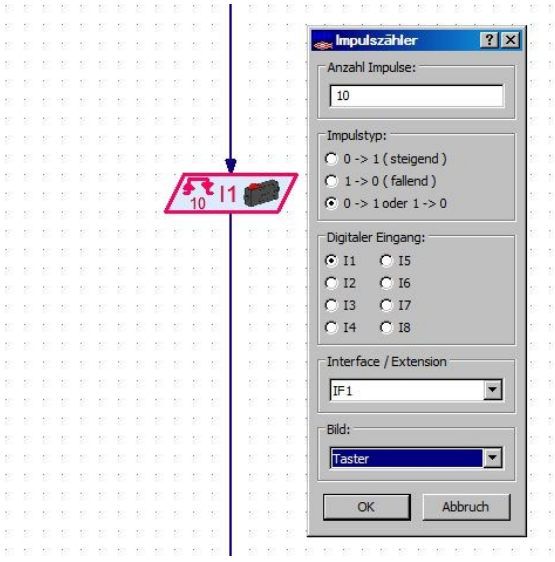


Abb. 2: Impulszähler in Robo Pro

Für Abbildung 3 habe ich die typische Verwendung eines Impulsrads mit vier Nocken (37157) und einem Mini-Taster (37783) nachgestellt.

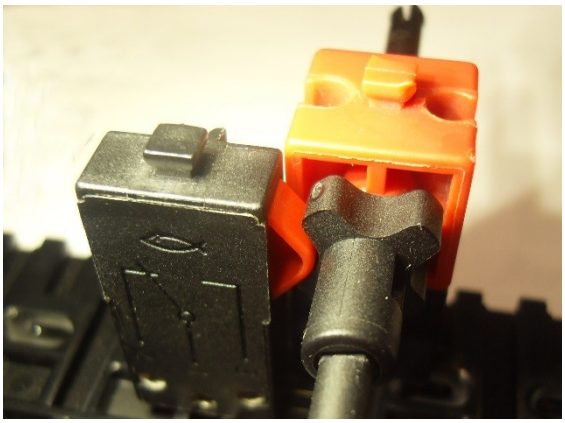


Abb. 3: Einbau des Impulsrads Z4 (37157) mit Mini-Taster (397783)

Wie oben beschrieben sind die Impulsräder für das Rastachsensystem konzipiert. Wie aber soll man mit diesen Impulsrädern bei der Verwendung von Metallachsen arbeiten?

Um auch diese Lücke zu schließen, habe ich ein Impulsrad Z4 (37157) so modifiziert, dass sich dieses mit Metallachsen verwenden lässt.

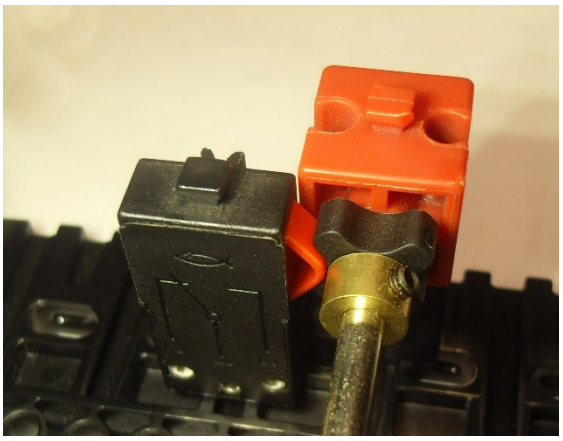


Abb. 4: Einbau des modifiziertem Impulsrads Z4 (37157) mit Mini-Taster (37783)

Dazu wurde das Impulsrad aufgebohrt und mit einer Messingnabe (mit einer 4 mm Bohrung) versehen. Mit der M3-Madenschraube lässt es sich nun auf der Metallachse fixieren.

So kann man nun auch bei der Verwendung von Metallachsen die Standard-Impulszählung der fischertechnik-Controller verwenden.

Fazit: Was nicht passt, wird eben passend gemacht …

Referenzen

[1] Andreas Tacke: *ft-Spezialteile made by TST (Teil 5). Hubgetriebe mit Impulsrad*. ft:pedia 3/2013, S. 36-37.

Getriebe

Das Differentialgetriebe

Thomas Püttmann

Differentialgetriebe fanden und finden in vielen Gebieten Anwendung. Wie solche Getriebe genau funktionieren, wird mit Hilfe von fischertechnik-Modellen in diesem Beitrag erklärt. Dabei kann man insbesondere das Konzept des Bezugssystemwechsels, das in Mathematik, Naturwissenschaften und Technik eine große Bedeutung besitzt, anwendungsorientiert erlernen.

Geschichte

Im Jahr 1900 stießen Schwammtaucher vor der griechischen Insel Antikythera auf ein Schiffswrack aus der Zeit um 100 v. Chr. Viele Artefakte wurden in den folgenden Monaten aus diesem Schiff geborgen. Eines dieser Fundstücke war ein unscheinbarer Metallklumpen, dessen große wissenschafts- und technologiegeschichtliche Bedeutung der Allgemeinheit erst viele Jahrzehnte später bekannt wurde. Es handelte sich um einen unvollständigen, erstaunlich komplexen Mechanismus mit mindestens 27, leider nur bruchstückhaft erhaltenen Zahnrädern, der unter anderem zur Vorhersage des Sonnen- und Mondlaufs sowie von Finsternissen und Mondphasen diente.

Die vorhandenen Fragmente zeigen eindeutig, dass im antiken Griechenland epizyklische Bewegungen – das sind Bewegungen, bei denen Kreise auf Kreisen abrollen (siehe Abb. 1) – nicht nur zur theoretischen Modellierung von Himmelskörperbahnen benutzt wurden, sondern auch in mechanischen Getrieben. Sie legen nahe, dass Stirnrad-Differentialgetriebe wie in Abbildung 1 damals bereits bekannt waren.

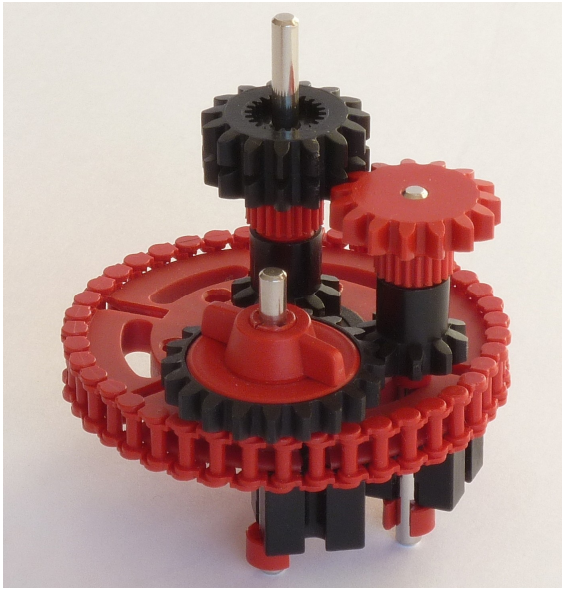


Abb. 1: Differentialgetriebe, wie es bis auf die Anzahl der Zähne von Derek de Solla Price als Teil des Antikythera-Mechanismus postuliert wurde [1]. In den aktuellen Modellen des Mechanismus findet es sich nicht mehr in exakt dieser Form.

Ein Differentialgetriebe war höchst wahrscheinlich auch zentraler Bestandteil des Kompasswagens von Ma Jun (ca. 200-265), aber diese Vermutung basiert auf einer schriftlichen Überlieferung. Tatsächliche Fundstücke aus dem antiken China gibt es nicht – vor allem, weil häufig eine Dynastie die Errungenschaften der Vorgängerdynastie umfangreich vernichtete.

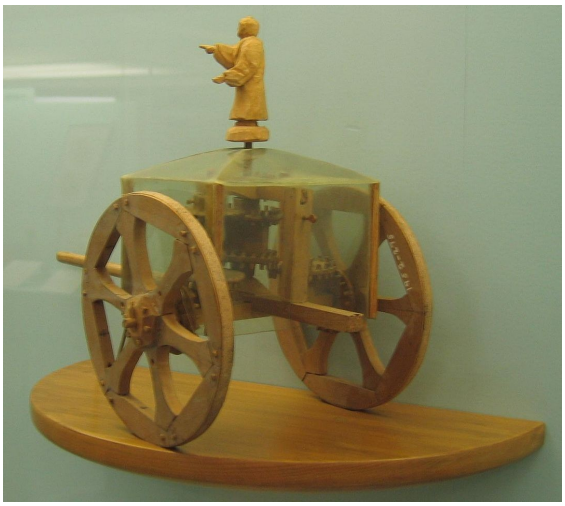


Abb. 2: Hypothetisches Modell eines Kompasswagens nach George H. Lanchaster, Science Museum London (Foto: Andy Dingley CC BY 3.0)

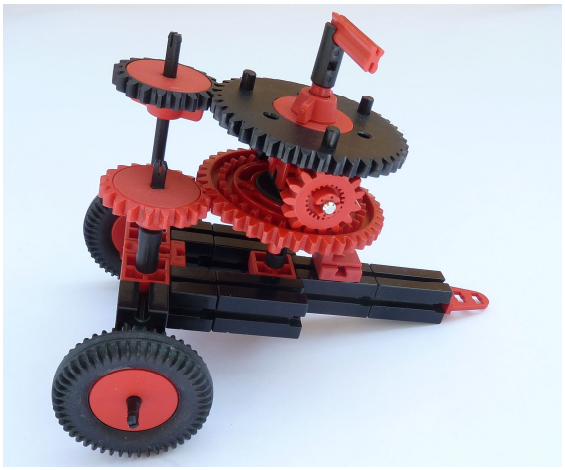


Abb. 3: Lanchaster-Modell aus fischertechnik

Die erste gesicherte Verwendung eines Differentialgetriebes erfolgte in einer Uhr. Noch im 18. Jahrhundert hatte die wahre Sonnenzeit – also die Zeit, die Sonnenuhren anzeigen – einen hohen Stellenwert. Mechanische Uhren zeigen die mittlere Sonnenzeit an, bei der die Stunden immer gleich lang dauern. Um die wahre Sonnenzeit daraus zu ermitteln, gab es häufig Korrekturtabellen auf Uhren. Der Londoner Uhrmacher Joseph Williamson setzte um das Jahr 1720 ein Differentialgetriebe ein, um die Korrektur mechanisch durchzuführen [2].

Ein gelernter Uhrmacher war es auch, der auf die Idee kam, Differentialgetriebe beim Antrieb von Fahrzeugen einzusetzen. Im Jahre 1748 präsentierte der unter anderem für seine Automaten bekannte Erfinder Jacques de Vaucanson (1709-1782) dem französischen König Ludwig XV. ein durch ein großes Uhrwerk angetriebenes Automobil mit Allradantrieb und Differentialgetriebe. Patentiert wurde das Differentialgetriebe für dampfgetriebene Fahrzeuge im Jahr 1827 durch einen weiteren Uhrmacher, nämlich Onésiphore Pecqueur (1792–1852). Die Verwendung des Differentialgetriebes in Fahrzeugen wird hervorragend in dem Video *Around the Corner* aus dem Jahr 1937 erklärt. In Fahrzeugen werden Differentialgetriebe zum ersten Mal als Verteilgetriebe benutzt, d. h. nur einer der drei An-/Abtriebe wird angetrieben; wie sich die Drehbewegung auf die beiden Abtriebe verteilt, wird durch die Kurvenfahrt bestimmt.

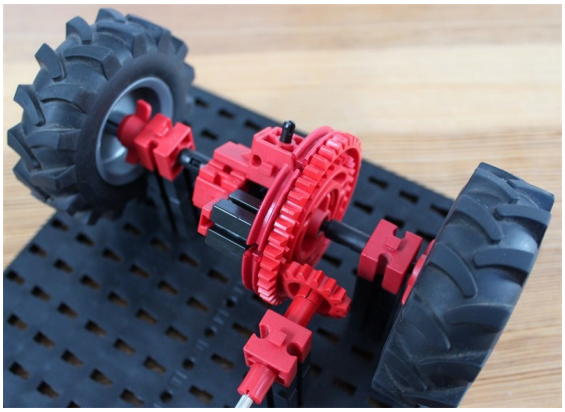


Abb. 4: fischertechnik-Modell eines Fahrzeugantriebs von Dirk Fox

Im ausgehenden 19. und in der ersten Hälfte des 20. Jahrhunderts fanden Differentialgetriebe ausgiebige Anwendung in mechanischen Analogrechnern. Wenn man von der militärischen Ausrichtung absieht, ist der Ausbildungsfilm *Basic Mechanisms in Fire Control Computers* der US Navy aus dem Jahr 1953 sehr lehrreich und sehenswert.

Beispiele

Zunächst schauen wir uns einige Beispiele an. Abbildung 5 zeigt das ursprüngliche fischertechnik-Differentialgetriebe aus der grau-roten Epoche und das aktuelle Differentialgetriebe. Beide arbeiten mit kleinen Kegelzahnrädern und verbergen den eigentlichen Mechanismus in einem geschlossenen Käfig. Sie eignen sich daher nicht gut, um zu verstehen, wie ein solches Getriebe eigentlich funktioniert.

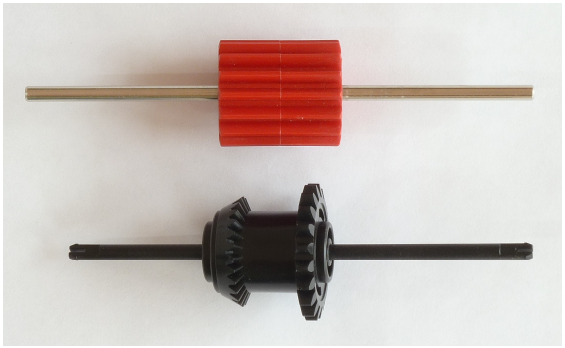


Abb. 5: Fertige Differentialgetriebe im fischertechnik-System

Ein sehr schönes Differentialgetriebe mit einem einsehbaren Differentialkäfig hat Dirk Fox gebaut (siehe Abb. 4 und 6). Allerdings kommen auch hier kleine Kegelzahnräder zum Einsatz. Das Differentialgetriebe in Abbildung 7 habe ich speziell mit dem Ziel entwickelt, die Funktion möglichst gut erklären zu können. Es eignet sich besonders für den Einsatz in Kompasswagen und Analogcomputern, aber weniger für den Fahrzeugbau.

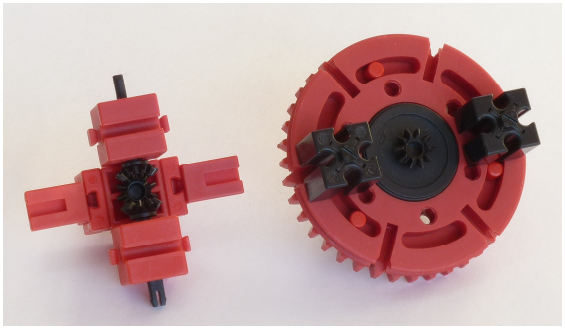


Abb. 6: Differentialgetriebe mit Kegelzahnrädern von Dirk Fox

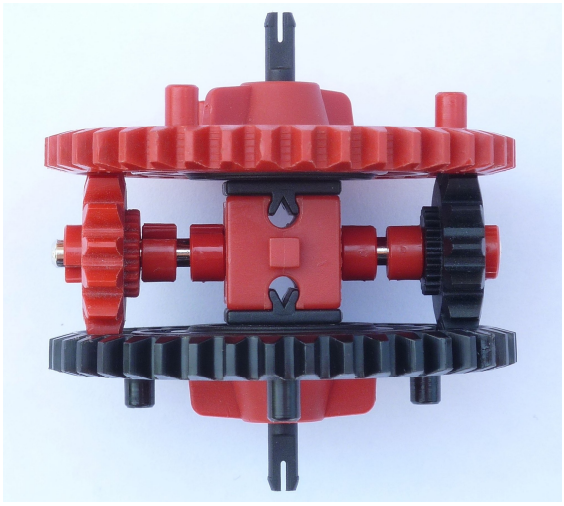


Abb. 7: Differential mit Kronenzahnrädern

Abb. 8 und 9 zeigen weitere Beispiele für Differentialgetriebe, anhand derer ich erklären möchte, welchen einfachen Grundideen man zweckmäßigerweise bei der gezielten Entwicklung solcher Getriebe mit oder ohne fischertechnik folgen sollte. Weitere Beispiele von Stirnradgetrieben finden sich im Bilderpool der ft-community.

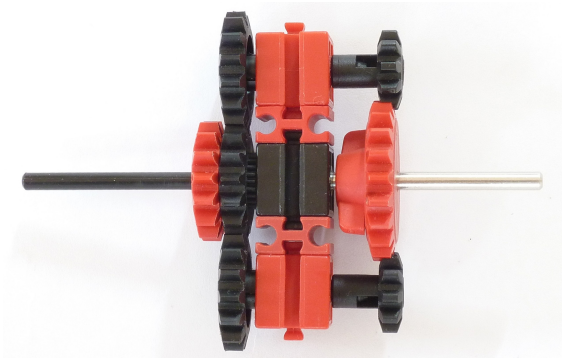


Abb. 8: Stirnrad-Differentialgetriebe für Kompasswagen und mechanische Analogrechner

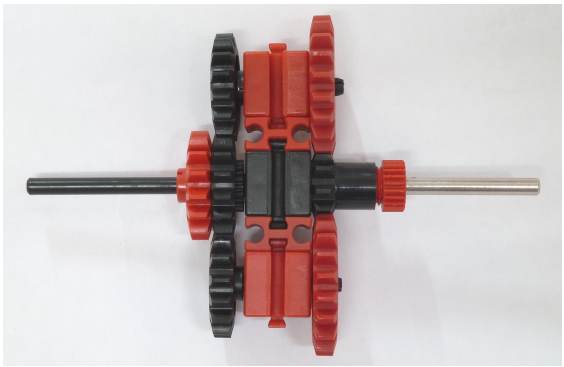


Abb. 9: Stirnrad-Differentialgetriebe, das sich auch für den Einsatz in Fahrzeugen eignet

Experimente

Wer in vollem Umfang verstehen möchte, wie ein Differentialgetriebe funktioniert, sollte das Funktionsmodell aus den Abbildung 7 und 10 nachbauen. Die Drehbewegungen aus oberem Sonnenrad, Steg und unterem Sonnenrad sind verkoppelt, d. h. es können nicht alle drei An-/Abtriebe unabhängig voneinander bewegt werden. Man kann sich zwei aussuchen und in eine gewünschte Position drehen, der dritte dreht sich dann zwangsweise in eine festgelegte Position. Eine solche Verkopplung dreier Drehbewegungen um eine gemeinsame Achse ist typisch für ein Umlaufrädergetriebe. Um die Verkopplung zu verstehen, empfiehlt es sich, am oberen und unteren Sonnenrad einen Zahn weiß zu markieren, und zwar so, dass die beiden Markierungen und der Steg alle mit der Nut des unteren roten Bausteins fluchten (Abb. 10).

Abb. 10: Markierung der Zahnräder

Experiment 1: Wir halten den Steg fest und drehen das untere Sonnenrad. Die Drehung wird durch das Rad am Steg auf das obere Sonnenrad übertragen. Dieses dreht sich von uns aus gesehen genauso weit wie das untere, aber in die entgegengesetzte Richtung. Offensichtlich spielt es keine Rolle, wie groß das Rad am Steg ist.

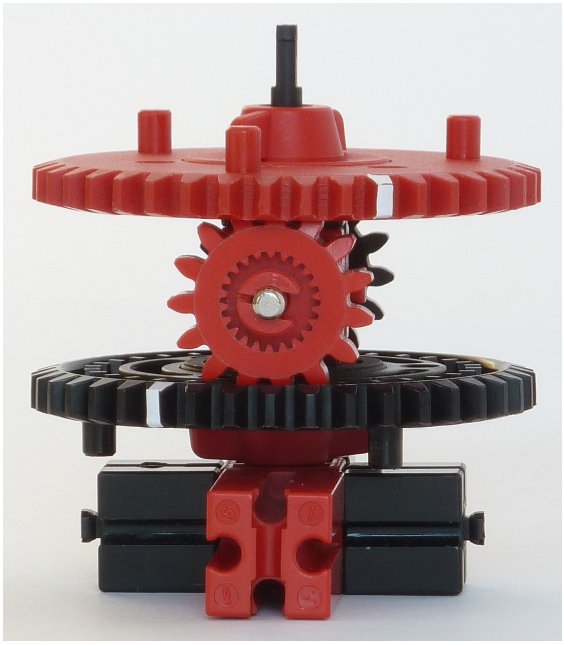


Abb. 11: Fester Steg

Experiment 2: Wir bringen das Modell wieder in seine Ausgangsposition, halten nun statt des Stegs das untere Sonnenrad fest und drehen das obere Sonnenrad. Der Steg dreht sich im gleichen Sinn, aber nur um den halben Winkel. Das ist die zweite wesentliche Beobachtung. Allerdings ist diese Beobachtung nicht so einfach zu erklären wie die erste. Darauf werden wir später noch eingehen.

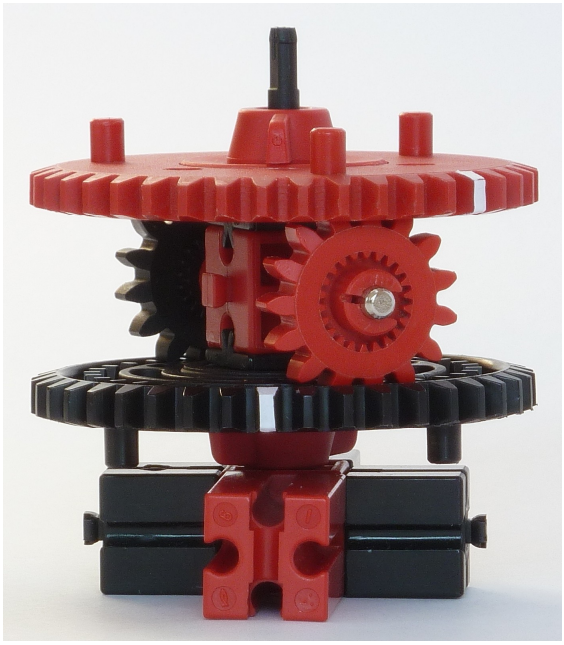


Abb. 12: Festes unteres Sonnenrad

Charakterisierung von Differentialgetrieben

Bei der Entwicklung von Differentialgetrieben ist es extrem nützlich, folgendes Prinzip im Kopf zu haben: Ein Umlaufrädergetriebe ist genau dann ein Differentialgetriebe, wenn es sich beim Festhalten eines frei wählbaren An-/Abtriebs wie in einem der beiden Experimente oben verhält. Anders formuliert: Wir suchen uns einen An-/Abtrieb aus und halten ihn fest. Dann drehen wir einen zweiten An-/Abtrieb um einen bestimmten Winkel. Dreht sich nun der dritte im gleichen Drehsinn halb oder doppelt so weit oder im entgegengesetzten Drehsinn genauso weit, dann ist das Getriebe ein Differentialgetriebe. Natürlich ist es sinnvoll, den festzuhaltenden Antrieb danach auszusuchen, dass man die Verkopplung der beiden anderen An-/Abtriebe möglichst einfach erklären kann. Wir verdeutlichen dieses Prinzip anhand einiger Beispiele.

Als erstes Beispiel betrachten wir das Antikythera-Getriebe in Abb. 1. Halten wir das Z41 (Drehscheibe mit Ketten [3]) fest und drehen an der Metallachse, so drehen sich die beiden Z10 und damit auch das rote Z15 mit dem Klemmring synchron mit. Die beiden schwarzen Z15 drehen sich somit synchron zur Metallachse in die entgegengesetzte Richtung. Daher ist das Getriebe ein Differentialgetriebe.

Als zweites betrachten wir das Getriebe in Abb. 8. Der Steg ist mit der schwarzen Kunststoffachse fest verbunden. Wir halten ihn fest. Wenn wir nun die rechte Metallachse und damit das Z20 um einen bestimmten Winkel drehen, so drehen sich die Z10 doppelt so weit im entgegengesetzten Sinn, ebenso die über die Wellen damit fest verbundenen äußeren Z15. Die beiden inneren Z15, die sich auf der schwarzen Achse frei drehen, bewegen sich damit gleichsinnig zum Z20, aber doppelt so weit. Damit ist auch dieses Stirnradgetriebe ein Differentialgetriebe.

Ähnlich sieht es bei dem Stirnradgetriebe in Abb. 9 aus. Hier drehen sich die beiden inneren Z15 bei festgehaltenem Steg gleichsinnig zum Z20, aber nur halb so weit. Auch hier liegt also ein Differentialgetriebe vor. Der Unterschied zum vorigen Stirnradgetriebe liegt darin, dass jetzt Kunststoffachse und Metallachse gegensinnig drehen, wenn die inneren Z15 festgehalten werden. Damit eignet sich diese Variante grundsätzlich für den Fahrzeugbau: Angetrieben wird das rote Z15, genügend große Räder können direkt auf die beiden – gegebenenfalls verlängerten – Achsen montiert werden.

Mathematische Beschreibung

Wir wollen jetzt die Zustände, die das Differentialgetriebe einnehmen kann, durch eine mathematische Gleichung beschreiben und dabei das Konzept des Bezugssystemwechsels an einem möglichst einfachen und anwendungsorientierten Beispiel vermitteln. Der erste Schritt in Richtung auf eine mathematische Beschreibung besteht darin, die verkoppelten Größen genau zu benennen und möglichst exakt festzulegen, wie sie gemessen werden. Wenn man hier zu wenig Zeit investiert, resultieren daraus häufig nachhaltige Unklarheiten.

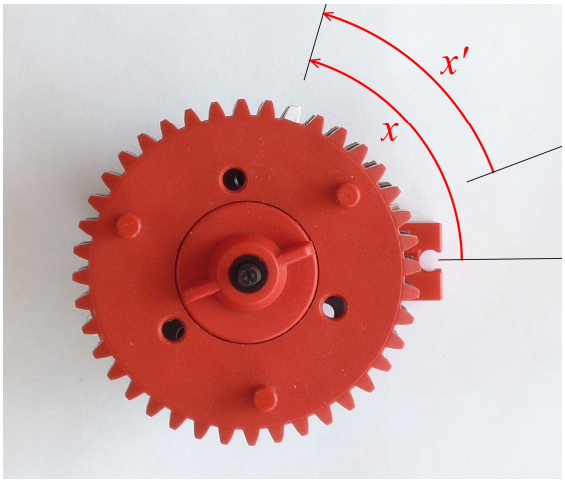


Abb. 13: Winkel x

Die verkoppelten Größen sind die Drehwinkel des Stegs und der beiden Sonnenräder. Bei Winkeln muss man darauf achten, dass man angibt, wogegen man diese misst und

in welchem Drehsinn. Wir messen die Winkel gegen die Halbgerade, die von der zentralen Achse aus durch die Nut des roten Bausteins 30 verläuft, gegen den Uhrzeigersinn positiv, im Uhrzeigersinn negativ, wenn man von oben auf das Modell schaut. Diese drei Winkel x, y und z sind in Abb. 13 und 14 dargestellt.

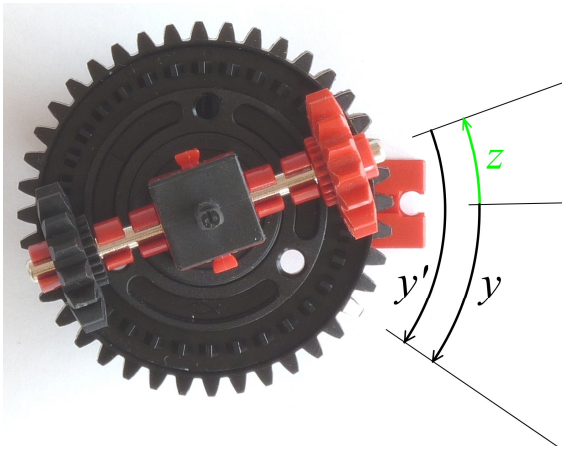


Abb. 14: Winkel y und z

In unserem ersten Experiment oben haben wir den Steg festgehalten. Das wird durch die Gleichung $z = 0$ beschrieben. Wir haben herausgefunden und erklärt, dass in diesem Fall eine Drehung des oberen Sonnenrads eine gleichweite Drehung in entgegengesetztem Sinn bewirkt. Mathematisch präziser lässt sich das jetzt durch die Gleichung $y = -x$ oder äquivalent

$$x + y = 0$$

formulieren. Diese Gleichung beschreibt alle Zustände des Getriebes bei festgehaltenem Steg.

Im zweiten Experiment haben wir beobachtet, dass sich bei festgehaltenem unterem Sonnenrad, also $y = 0$, der Steg halb so weit dreht wie das obere Sonnenrad. Die Gleichung

$$x = 2z$$

beschreibt also alle Zustände des Getriebes bei festgehaltenem unterem Sonnenrad. Diese Gleichung besitzt keine direkte Erklärung. Noch weniger verstehen wir bislang den allgemeinen Fall (Abb. 15).

Abb. 15: Allgemeiner Zustand des Differentialgetriebes

Beobachterwechsel

Wir denken uns einen Beobachter, der sich mit dem Steg dreht und die Winkel immer gegen den Steg misst, statt gegen die Halbgerade durch den roten Baustein. Dieser Beobachter misst die drei Winkel $z' = 0$, x' und y'. Da auch aus seiner Sicht das Rad am Steg die Bewegungen der beiden Sonnenräder verkoppelt, gilt $x' = -y'$ oder äquivalent $x' + y' = 0$.

Die zentrale Frage ist nun, wie die von uns gemessenen Winkel x, y und z mit den Winkeln x' und y' des mitbewegten Beobachters zusammenhängen. Um das zu einzusehen, drehen wir Sonnenräder und Steg gemeinsam um den Winkel $-z$ gegen den Uhrzeigersinn, also um den Winkel z im Uhrzeigersinn. Der Steg befindet sich somit wieder in Ausgangsposition. Wir messen nun die Winkel x' und y', die der mitbewegte Beobachter vorher gemessen hat. Es ergibt sich also $x' = x - z$ und $y' = y - z$.

Einsetzen dieser beiden Formeln in die Gleichung $x' + y' = 0$ liefert [4]

$$x + y - 2z = 0$$

Das ist die Grundgleichung eines Differentialgetriebes. Sie beschreibt alle möglichen Zustände. Anschaulich besagt sie, dass sich die Drehbewegung des Stegs bzw. Käfigs durch Mittelung aus den Drehbewegungen der Sonnenräder ergibt. Man kann ihre Gültigkeit gut in den Abb. 13, 14 und 15 überprüfen.

Mit Hilfe der Grundgleichung können wir nun noch einmal zu unserem zweiten Experiment zurückgehen. Dort hielten wir das untere Sonnenrad fest, was durch die Gleichung $y = 0$ beschrieben wird. Dann werden alle möglichen Zustände des Getriebes durch die Gleichung $x = 2z$ beschrieben. Damit haben wir nun eine mathematische Erklärung unserer Beobachtung.

Planetengetriebe

Die obigen Überlegungen verallgemeinern sich direkt auf allgemeine Planetengetriebe. Wir illustrieren das anhand des Standard-Planetengetriebes mit dem Innenzahnrad (Abb. 16).

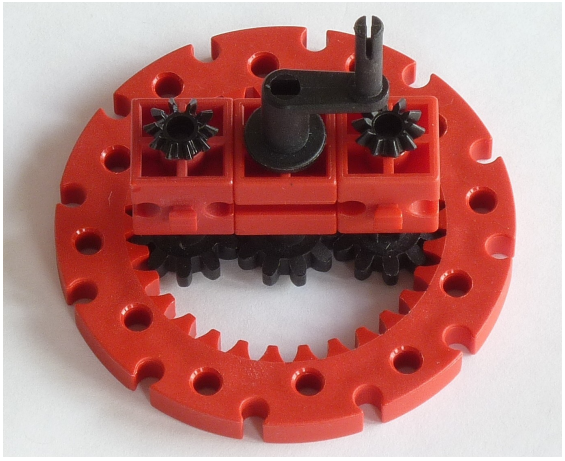


Abb. 16: Planetengetriebe

Auch hier haben wir drei Winkel, den Winkel x des Sonnenrads, den Winkel y des Innenzahnrads und den Winkel z des Stegs, alle drei gemessen gegen eine gemeinsame Halbachse, auf der sie fluchten können. Auch hier können wir das Getriebe am einfachsten verstehen, wenn wir den Steg festhalten. Die Zustände des Getriebes werden dann durch die Gleichung

$$x = -3y$$

beschrieben, da das Zahnverhältnis 10:30 ist und eine Umkehrung des Drehsinns stattfindet. Ein sich mit dem Steg drehender Beobachter misst daher immer Winkel x' und y', die der Gleichung $x' = -3y'$ genügen.

Der Zusammenhang zwischen unseren Winkeln und den vom mitbewegten Beobachter gemessenen Winkels ist wie oben durch $x' = x - z$ und $y' = y - z$ gegeben. Durch Einsetzen erhält man die Gleichung

$$x + 3y - 4z = 0$$

Sie beschreibt alle möglichen Zustände des Getriebes.

Für jedes Planetengetriebe lässt sich analog eine Gleichung der Form

$$n_1 \cdot x + n_2 \cdot y + n_3 \cdot z = 0$$

herleiten. Sie wird *Willis-Gleichung* genannt. Da man stets alle drei An-/Abtriebe synchron drehen kann, liefern alle x, y, z mit $x = y = z$ Lösungen dieser Gleichung. Daraus folgt $n_1 + n_2 + n_3 = 0$.

Ausblick

In der nächsten Ausgabe werden wir uns ausführlich dem Thema Kompasswagen widmen. Dabei werden die hier vorgestellten Getriebe und Grundideen Anwendung finden. In einem weiteren Aufsatz werden wir den Einsatz von Differentialgetrieben in mechanischen Analogrechnern behandeln. Die obigen Überlegungen legen nahe, dass und wie man lineare Gleichungssysteme durch Getriebe lösen kann. Tatsächlich ist diese Idee mehrfach gefunden und auch publiziert, aber nicht überzeugend umgesetzt worden. Zum Einstellen der Koeffizienten benötigt man nämlich präzise einstellbare variable Übersetzungen. Dafür gibt es bislang noch kein geeignetes Konzept. Zum Addieren und Subtrahieren einzelner Größen in komplexen Mechanismen sind Differentialgetriebe allerdings vielfach zweckdienlich eingesetzt worden.

Zum Nachbau

Leider laufen nicht alle schwarzen Z15 (35695) reibungslos auf den Achsen. Hier sollte man leichtgängige Kombinationen aus seinem Teilevorrat aussuchen, um gut funktionierende Getriebe zu erhalten. In den Getrieben, die in den Abbildungen 1, 8 und 9 dargestellt sind, wurden außen jeweils Z15 mit Klemmring (37685) eingesetzt. Diese finden sich leider nicht mehr im aktuellen fischertechnik-Programm. Man kann ein Z15 mit Klemmring allerdings problemlos durch ein freilaufendes Z15 ersetzen, in das man erst einen kegelförmig zusammengerollten Papierstreifen und dann die Achse einsteckt. Der Papierstreifen sollte dazu idealerweise genauso lang sein, wie der Umfang einer 4 mm Achse – also etwa 12 mm. Dadurch wird sichergestellt, dass die Achse zentrisch im Z15 sitzt.

Zum Abschluss geben wir eine Liste der Einzelteile für das Differential mit Kronenzahnrädern aus Abbildung 7 an.

Stück	ft-Nr.	Bezeichnung
2	31022	Z40
2	31058	Nabenmutter rot
2	68535	Freilaufnabe schwarz
2	130593	Rastachse mit Platte
1	32064	Baustein 15 m. Bohrung
1	31032	Metallachse 60
6	37679	Klemmring 5
2	35695	Z15 freilaufend

Referenzen

[1] Derek de Solla Price: *Gears from the Greeks*. Transactions of the American Philosophical Society, Vol. 64, Part 7, 1974.

[2] Henry C. King: *Geared to the Stars. The Evolution of Planetariums, Orreries, and Astronomical Clocks*. University of Torronto Press, Torronto, Buffalo, 1978.

[3] fischertechnik: *Hobby 2, Motor und Getriebe*. Fischer-Werke, Tumlingen, 2/1975.

[4] Thomas Püttmann: *Zahnräder und Übersetzungen (Teil 2)*. ft:pedia 3/2011, S. 25-28.

Getriebe

Uhrwerk mit Z80 und Z100

Gerhard Birkenstock

Vor einigen Monaten stand ich vor dem Problem, eine Untersetzung realisieren zu müssen. Es sollten einige Messreihen recht präzise erfasst werden. Dabei war wichtig, keinen Totweg in der großen Untersetzung zu bekommen. Kleine Zahnräder mit wenig Zähnen hat fischertechnik im Programm – das Problem sind die großen. Aus diesen Überlegungen sind zwei riesige Zahnräder mit 80 und 100 Zähnen entstanden – und ein besonderes Uhrwerk.

Die Zahnräder

Um die großen Zahnräder vernünftig auf die Achsen zu bringen, musste ich mir die Nabenmuttern und Nabenzangen von fischertechnik genauer ansehen. Als vernünftige Maße haben sich dabei die folgenden herausgestellt:

- Die innere freie Öffnung hat einen Durchmesser von 20 mm. Dort sind 24 Nasen eingebracht.
- Jede Nase hat einen Radius von 0,75 mm. Der Mittelpunkt dieser Radien wird auf den 20 mm Durchmesser aufgelegt. Somit ist alle 15° ein Halbkreis zu erstellen.

Um beim Festziehen der Nabenmuttern eine Zentrierung zu bekommen, hat fischertechnik in die Klemmung einen flachen Konus eingebracht. Da ich mit der mir zugänglichen CNC-Maschine keine perfekten Schrägen in der Z-Achse herstellen kann, habe ich mich mit fein abgestuften (h = 0,2 mm) Treppenschritten begnügt. Dies funktioniert sehr gut.

In Abb. 1 kann man die eben beschriebenen Konturen in der CAD-Ansicht erkennen. Die Kreise haben von innen nach außen die folgenden Durchmesser:

- Taschen-Innenaussparung = 16,0 mm
- Durchbruch mit 24 Nasen = 20,0 mm
- Rand 1 = 21,5 mm
- Rand 2 = 23,0 mm
- Rand 3 = 24,5 mm
- Rand 4 = 26,0 mm

Der Rand 4 ist gleichzeitig die Außenkontur für die Nabe.

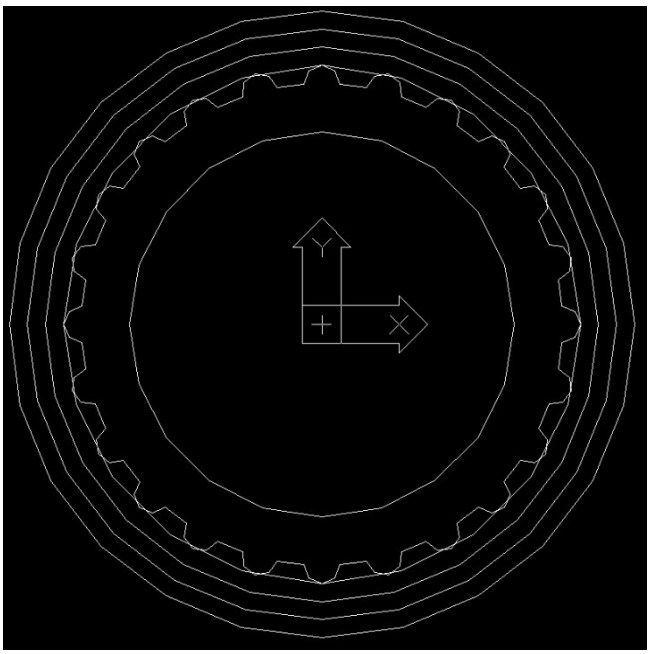


Abb. 1: CAD-Ansicht Konturen

In der folgenden Abb. 2 ist eine Detaildarstellung eingefügt. Man sieht die Stufenringe sehr schön und einen Teil der 24 Nasen in der erstellten Kontur.

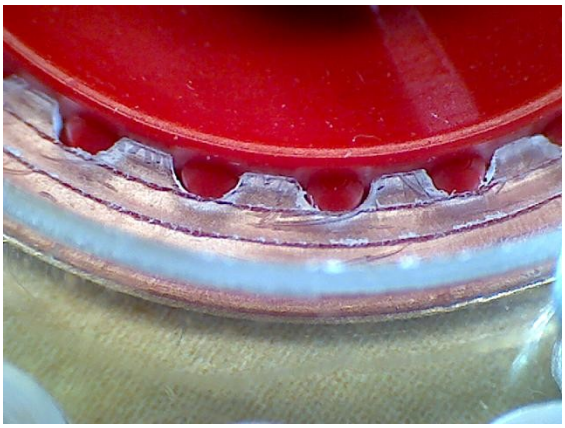


Abb. 2: Detailansicht der Nabe

Zahnkontur per Evolvente

Wie in der ft:pedia 1/2012 schon erläutert wurde [1], werden Zahnräder mit Evolventen beschrieben. Dies ist hier vereinfacht durchgeführt. Einen Zahn habe ich mir mit dem Modul 1 erstellt. Dann wurde 80 bzw. 100 mal kopiert und an der richtigen Winkelstelle wieder abgesetzt. Der letzte Schritt beim CAD-Zeichnen war dann die Verbindung der Zähne im Fußkreis. Die Grundinformationen für die CNC-Weitergabe waren damit erstellt.

Nach der Übersetzung in das CNC-Programm mittels eines 1 mm-Fräsers war dann auch klar, wie lange die Maschine für ein Zahnrad fräsen wird: Es waren zehn Stunden!

Abb. 3: CAD-Ansicht einiger Zähne

Aus Stabilitätsgründen habe ich mir als Werkstoff den Kunststoff *Polycarbonat* ausgesucht. Er ist sehr bruchfest und gut zu bearbeiten. Die berüchtigte Rissbildung bei der Bearbeitung bleibt hier aus.

Das Uhrengetriebe

Zeiger

Eine schöne Anwendung der großen Zahnräder sind die „schwebenden Zeiger“ einer Uhr. Hierzu wählte ich eine besondere Darstellungsart auf dem Zifferblatt: 12 Uhr mittags sollte weiterhin oben sein, die Datumsgrenze, also Mitternacht, aber unten. Der Minutenzeiger sollte sich so verhalten wie gewohnt.

Daraus resultiert jedoch eine Teilung per Getriebe von 24 zu 1.

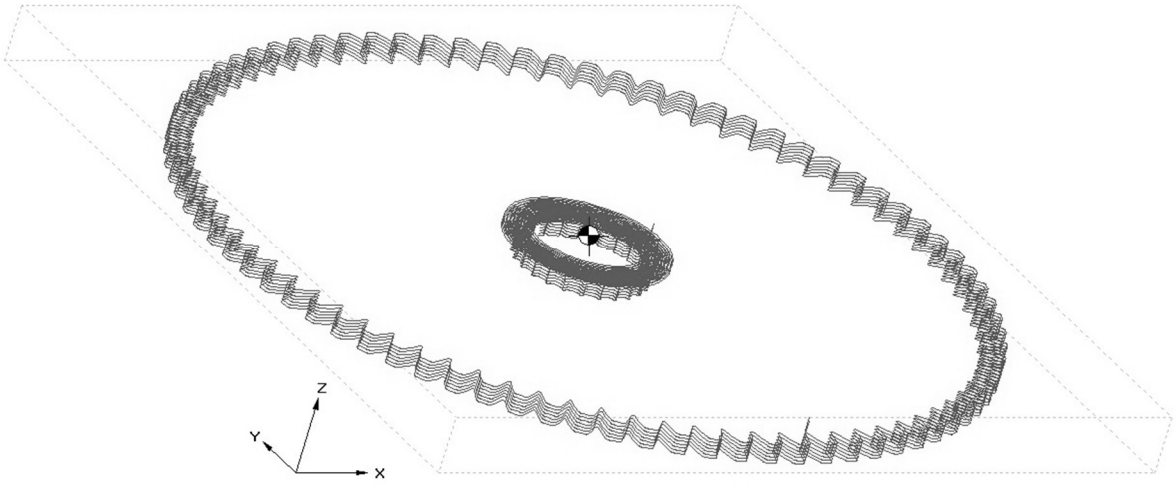


Abb. 4: Darstellung der CNC-Fräskopfstrecke beim Z80 mit Nabendurchbruch

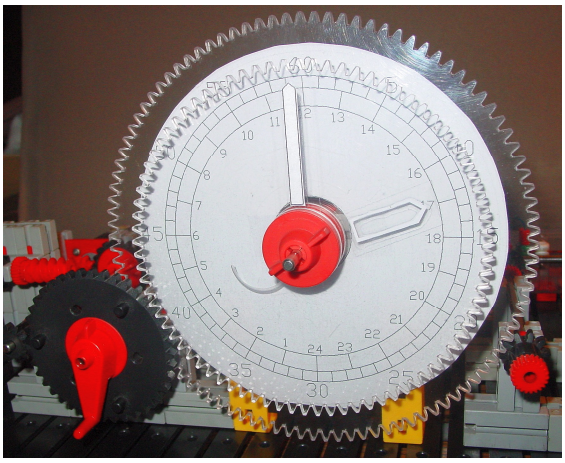


Abb. 5: Zifferblatt

Dafür wählte ich die folgende Zerlegung:

$$24 = \frac{40 \cdot 15 \cdot 32 \cdot \mathbf{100}}{\mathbf{80} \cdot 10 \cdot 10 \cdot 10}$$

Die beiden gegebenen Zahnräder mit 80 und 100 Zähnen habe ich grün hinterlegt. Alle anderen Zahnradkombinationen sollten von fischertechnik sein. Die Zahnräder Z40, Z15 und Z10 sind normale Stirnräder aus dem fischertechnik-Programm. 32 Zähne sind an der Seite eines Z40 (Kronrad) zu finden. Wie aus dem Gesamtbild ersichtlich, konnte ich durch den seitlichen Trieb und einen weiteren Kegeltrieb um das große Zifferblatt herumkommen (Abb. 6).

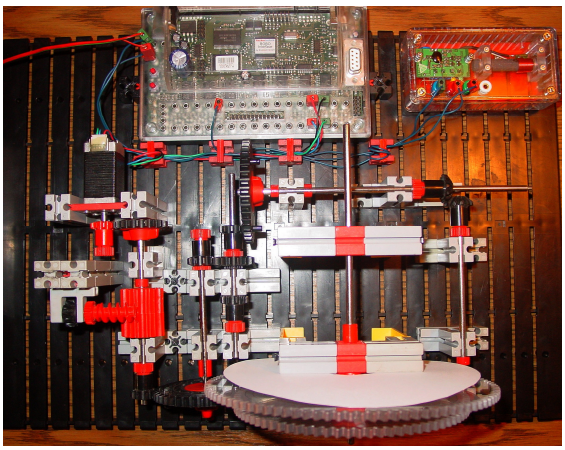


Abb. 6: Gesamtbild der Uhr

Das Ergebnis ist, wie ich finde, eine schöne mechanische Lösung. Um daraus eine 12 h-Version der Uhr zu erstellen, genügt es, das letzte Zahnrad durch ein Z20 zu ersetzen (siehe Abb. 5 unten rechts).

Mechanisches Additionsglied

Um die Uhr während des Betriebs stellen zu können, habe ich ein Differential in den Antriebsstrang eingesetzt. Die Schnecke ist durch die Steigung selbst hemmend. Muss die Uhr gestellt werden, wird nur die Schnecke in die entsprechende Richtung gedreht – ein mechanisches Additionsglied.

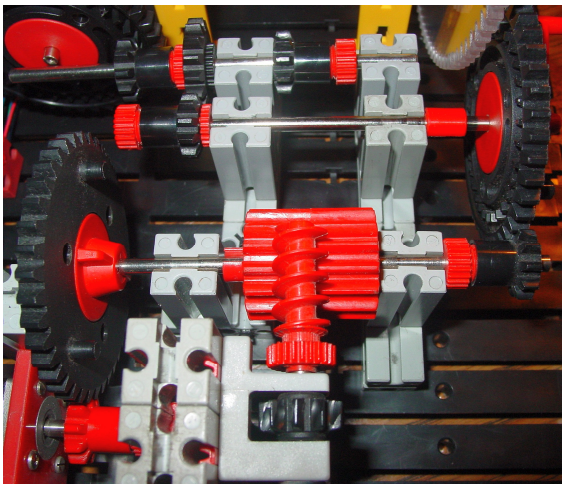


Abb. 7: Differential als mechanisches Additionsglied

Den Antrieb habe ich mit einem kleinen Schrittmotor realisiert. Der Vorteil dieses Schrittmotors ist seine Achse: Sie hat einen Durchmesser von genau 4 mm. Damit sind wir mitten in der fischertechnik-Welt: Die fischertechnik-Zahnräder können direkt darauf geklemmt werden.

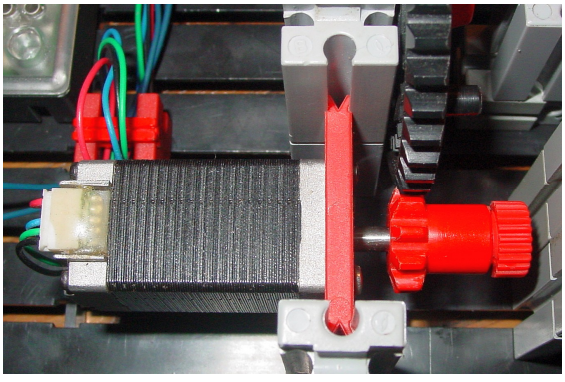


Abb. 8: Schrittmotor

Ein alter Flachstein 30 rot eignet sich bestens als Flansch (Abb. 9).

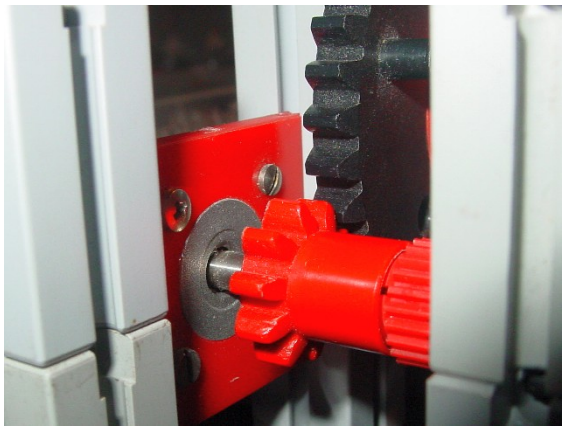


Abb. 9: Motorflansch

DCF77-Modul mit Verpolungsschutz

Um die Uhr mit einem exakten Zeittakt zu versehen, verwende ich ein DCF77-Modul. Das Thema DCF77 ist in der ft:pedia 3/2013 super gut erklärt [2]. Hier gibt es nicht viel hinzuzufügen. Nur eine kleine Ergänzung zum Artikel habe ich noch.

Um den DCF77-Empfängerbaustein vor einer Verpolung zu schützen, habe ich im Schaltplan die Diode D1 eingefügt. Der Typ 1N4148 ist der weltweite Standard.

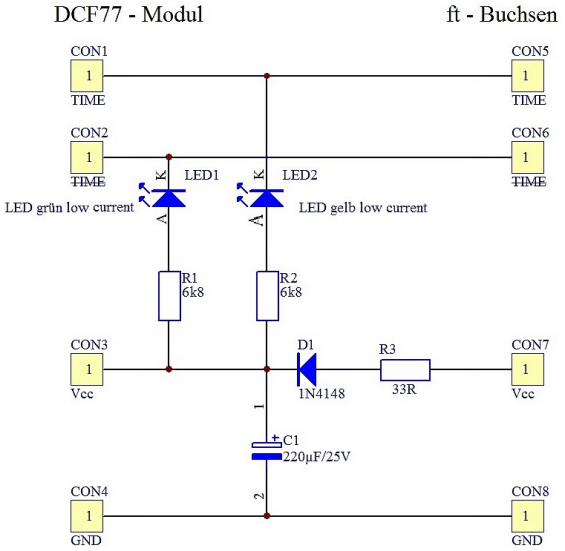


Abb. 10: Schaltplan des DCF77-Empfängers

Zu beachten ist hierbei jedoch, dass durch den geringen Innenwiderstand im Kondensator dieser sich im ersten Moment der Aufladung wie ein Kurzschluss verhält. Zum Schutz der Diode habe ich aus diesem Grund noch einen Widerstand R3 mit 33 Ohm vorgeschaltet. Die an diesem Widerstand abfallende Spannung ist:

$$U = R3 \cdot I_{ges} = 33 \cdot 0{,}012 = 0{,}396\ V$$

Mit 0,4 V ist dies im Normalbetrieb vernachlässigbar. Wird jedoch ein eingeschalteter fischertechnik-Transformator angeschlossen, sieht die Berechnung so aus:

$$\frac{U}{R3} = \frac{15\ V}{33\ Ohm} = 0{,}45\ A = I_{max}$$

Das ist etwas weniger als der maximal zulässige Stoßstrom von 0,5 A. Die Diode ist somit geschützt.

Um die Funktion des Moduls sehen zu können, habe ich zwei LEDs eingesetzt. Da die beiden Ausgänge des DCF77-Moduls im Datenblatt mit 1 mA angegeben sind, können keine 20 mA-Typen verwendet werden. Es sind zwar BC847-Transistoren mit bis zu 100 mA verbaut; diese bekommen jedoch nicht genügend Basisstrom vom Prozessor, um komplett durchsteuern zu können.

Die verwendeten 6k8-Ohm Widerstände stellen einen Strom von 2 mA ein. Dies ist für die *low current* LEDs genau richtig. Und die Transistoren schalten bei diesem Strom auch noch zuverlässig gegen GND durch, um auch den RoboPro-Eingang sicher schalten zu können.

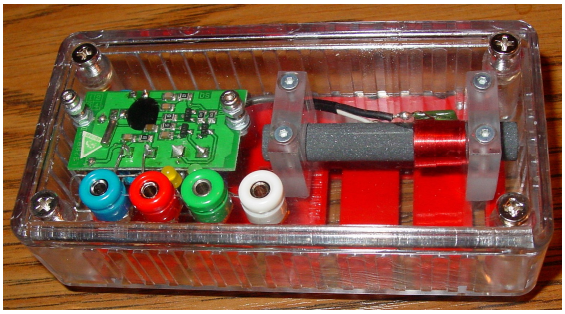


Abb. 11: Ansicht meiner DCF77-Lösung

Unter dem Gehäuse wurde eine alte, halbierte fischertechnik-Grundplatte 90x90 (31002) angeschraubt. Die im Gehäuse eingesetzten Buchsen stammen von Conrad – man muss sie mit der Umschreibung „Miniatur-Laborbuchse“ suchen. Und, lasst

euch nicht irritieren. Es sind 2,5 mm Buchsen, auch wenn dort 2,6 mm steht. Die zum Download bereitgestellten Fertigungszeichnungen beweisen es.

RoboPro-Programm

Für meine Uhr benötige ich nur ein Minutensignal. Da der Zeitzeichenempfänger genau 59 Impulse innerhalb einer Minute sendet, habe ich das genutzt: Einfach bis 59 zählen und die benötigten Schritte für die Bewegung um eine Minute auf den Schrittmotor ausgeben.

Bei meiner Konfiguration ist die Impulszahl leider etwas unglücklich:

Für eine Stunde Laufzeit werden 800 Impulse benötigt.

$$\frac{800}{60} = 13,\overline{3} \text{ Impulse pro Minute}$$

Meine Lösung: Ich gebe zweimal 13 Impulse und einmal 14 Impulse aus. Damit funktioniert es:

$$20 \cdot (13 + 13 + 14) = 800$$

Referenzen

[1] Thomas Püttmann: *Zahnräder und Übersetzungen (Teil 3)*. ft:pedia 1/2012, S. 13-21.

[2] Dirk Fox, Dirk Ottensmeyer: *Bau einer ft-Funkuhr*. ft:pedia 1/2012, S. 4-10.

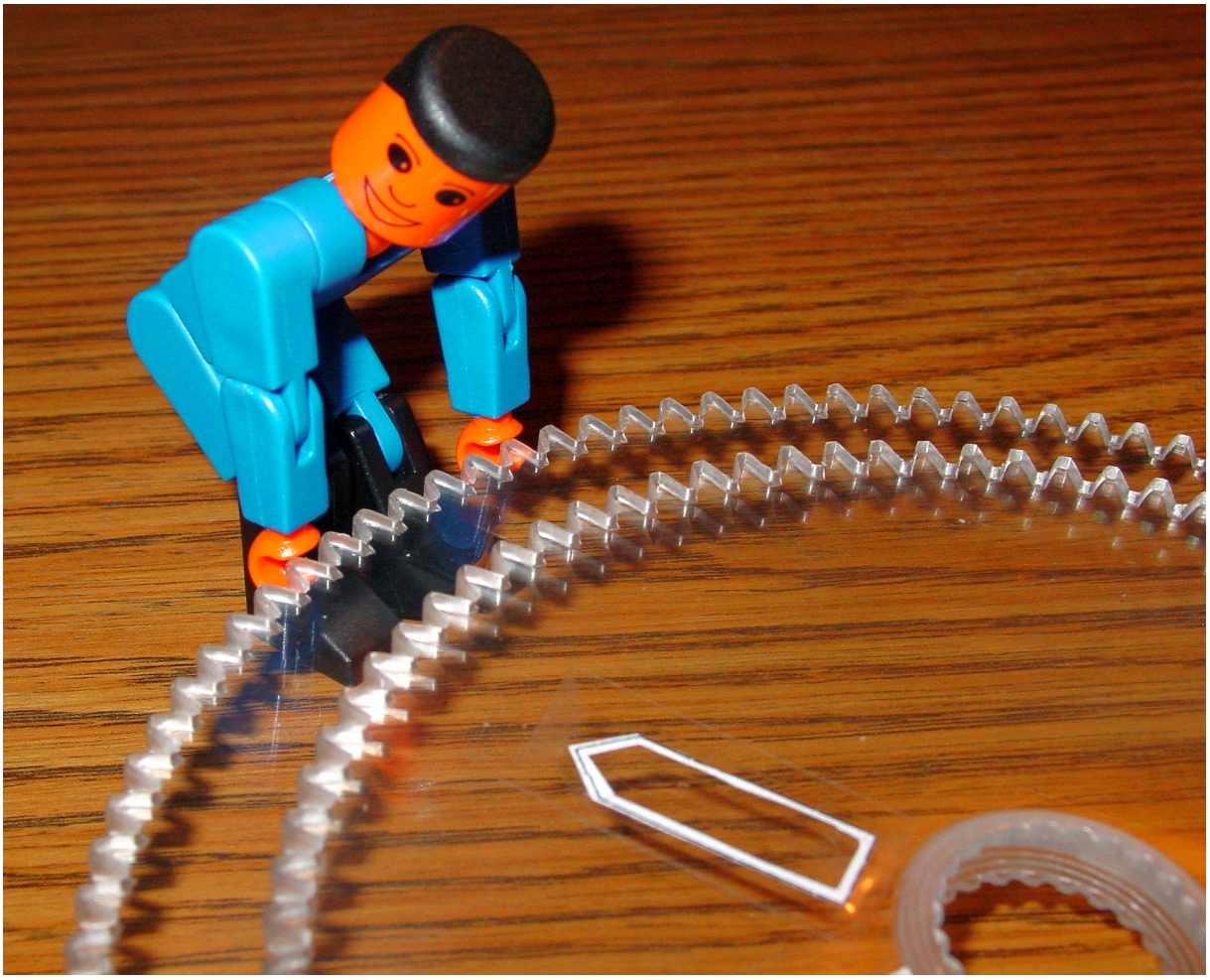


Abb. 12: Z80 und Z100 mit fischertechnik-Nabe und Chef-Mechaniker

Modell

Detail Engineering: Schreiender Wecker

Andreas Gail

Der Schreiende Wecker ist aus der Reihe ‚Drei ???' literaturbekannt. Ist das nun Dichtung oder Wahrheit? In diesem Fall zumindest ist aus Dichtung Wahrheit geworden: Unter Verwendung von drei Schrittmotoren und einer Echtzeituhr habe ich einen solchen Wecker gebaut. Auch dieses Bauprojekt führte zu einer Reihe von Detaillösungen, die sich als Lösungsansätze für diverse andere Bauprojekte eignen können.

Das Gesamtprojekt

Eine Uhr mit Zeigern sollte sich nach dem Einschalten auf die aktuelle Uhrzeit einstellen. Mit einem dritten Zeiger sollte die gewünschte Weckzeit eingestellt werden können, und damit ein solcher Wecker auch seinen Namen verdient, galt es ein schreiendes Wecksignal zu realisieren.

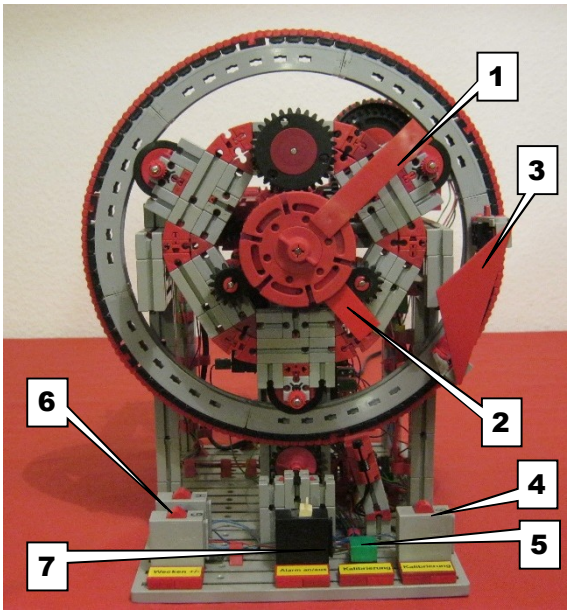


Abb. 1: Schreiender Wecker, Gesamtansicht

Abb. 1 zeigt den Wecker mit Minutenzeiger (1), Stundenzeiger (2), Zeiger der Weckzeit (3), Taster zur manuellen Neukalibrierung der Zeiger (4) und die Kontrollleuchte, die die Kalibrierung anzeigt (5). Die Kalibrierfunktion läuft bei jedem Einschalten des Weckers ab; dabei werden die Zeiger über einzelne Schrittmotoren auf die Zeit der Echtzeituhr eingestellt und es wird ermittelt, auf welche Uhrzeit die Weckfunktion eingestellt wurde.

Die Weckzeit kann über die beiden Taster (6) verstellt werden. Der Schalter (7) dient zum Ein- oder Ausschalten der Weckfunktion. Die gesamte Uhr wird durch einen Robo TX Controller gesteuert.

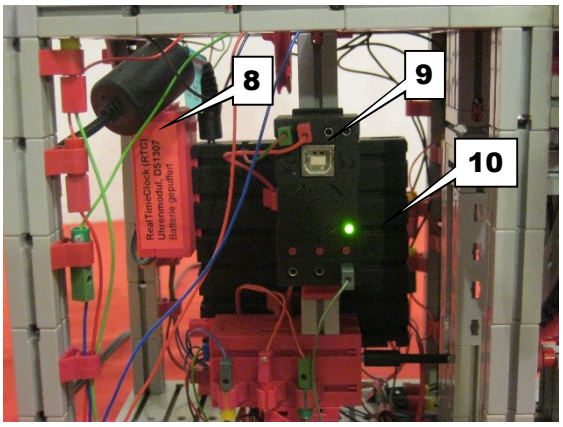


Abb. 2: Schreiender Wecker, Elektronik-Komponenten, Echtzeituhr (8), Sound+Lights-Modul (9), Rückseite Robo TX Controller (10)

Eine Anleitung zur Einbindung der Echtzeituhr findet sich in der ft:pedia 4/2013 [1].

Im Folgenden werden drei Detaillösungen vorgestellt: die Verriegelung zweier gekoppelter mechanischer Systeme, die Einstellung der Uhr auf eine vorgegebene Zeit und der Antrieb des Weckzeit-Zeigers.

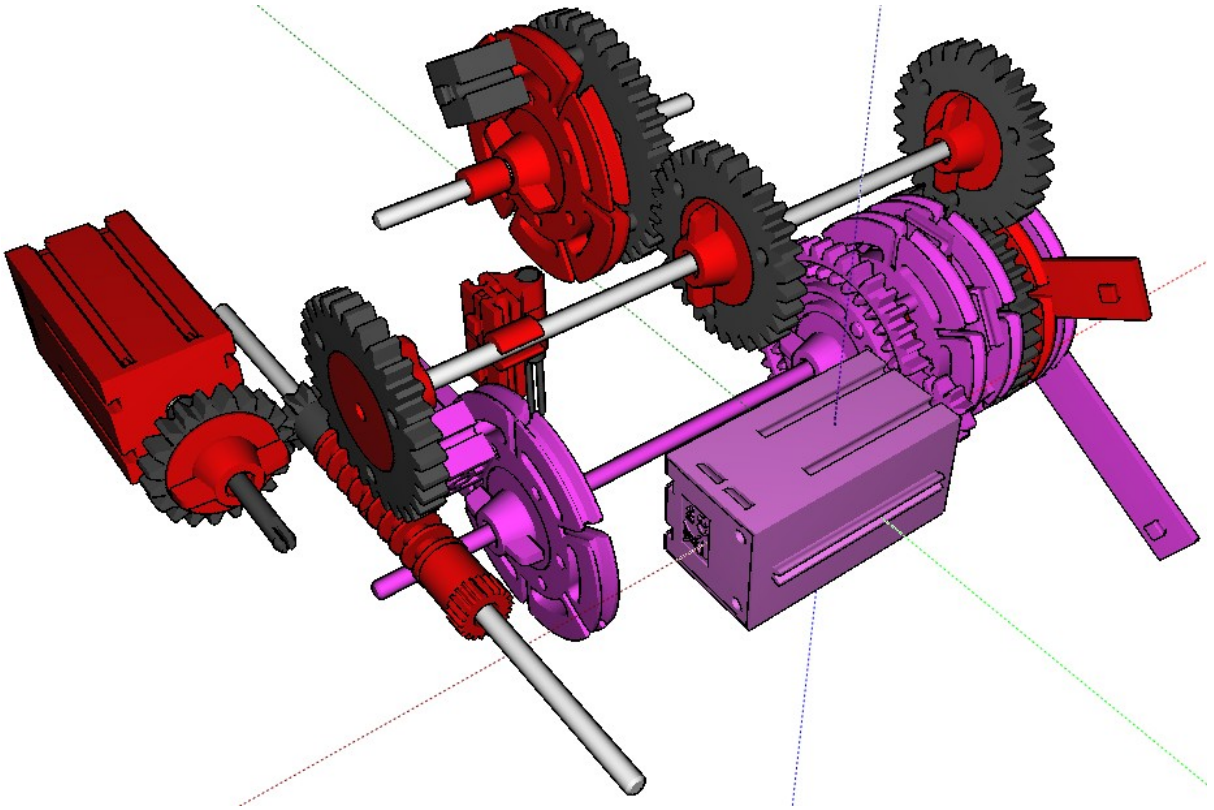


Abb. 3: Getrenntes Antriebsprinzip von Stundenzeiger (rot/schwarz) und Minutenzeiger (violett)

Detail-Lösung A

Verriegelung zweier gekoppelter mechanischer Systeme

Wie eingangs beschrieben werden die beiden Zeiger (Minutenzeiger und Stundenzeiger) getrennt über Schrittmotoren angesteuert. Auf diese Weise kann die aktuelle Uhrzeit leicht eingestellt werden.

Problem: Der schnellere Minutenzeiger hat bei der Erstkonstruktion gelegentlich den eigentlich langsameren Stundenzeiger ‚mitgeschleppt'. Dadurch war die Stundenanzeige stark fehlerhaft.

Lösung: Das Mitschleppen des Stundenzeigers (Abb. 3, rot/schwarz) wird durch die Nutzung eines Schneckengetriebes verriegelt. Eine Antriebsschnecke verlangsamt nicht nur die Drehbewegung [2], sondern hat den hier genutzten Vorteil, dass die motorisch angetriebene Welle nicht in umgekehrter Richtung bewegt werden kann. Wird also der Minutenzeiger angetrieben (Abb. 3, violett), kann der Stundenzeiger nicht mitbewegt werden, weil er durch das Schneckengetriebe blockiert wird.

Detail-Lösung B

Kalibrierung der Zeigerposition und Einstellung der Uhrzeit

Nach dem Einschalten des Weckers sollen die Zeiger auf die aktuelle Uhrzeit eingestellt werden.

Problem: Die aktuelle Zeigerposition ist zunächst nicht bekannt. Folglich weiß die Steuerung nicht, wie weit sie die Zeiger drehen muss, um diese auf die korrekte Position zu stellen.

Lösung: Die Kalibrierung läuft in drei aufeinanderfolgenden Schritten ab.

- Schritt 1: Zunächst wird der Zeiger in einer Richtung gedreht, bis der Reedkontakt den auf der Drehscheibe befindlichen Magnetbaustein detektiert (Abb. 4, violett). Der Motor wird dabei nicht als Schrittmotor angesteuert, sondern einfach nur ein- und bei Reedkontaktmeldung abgeschaltet. Wichtig ist dabei, dass die einmal festgelegte Drehrichtung und die Geschwindigkeit später nicht mehr geändert werden.

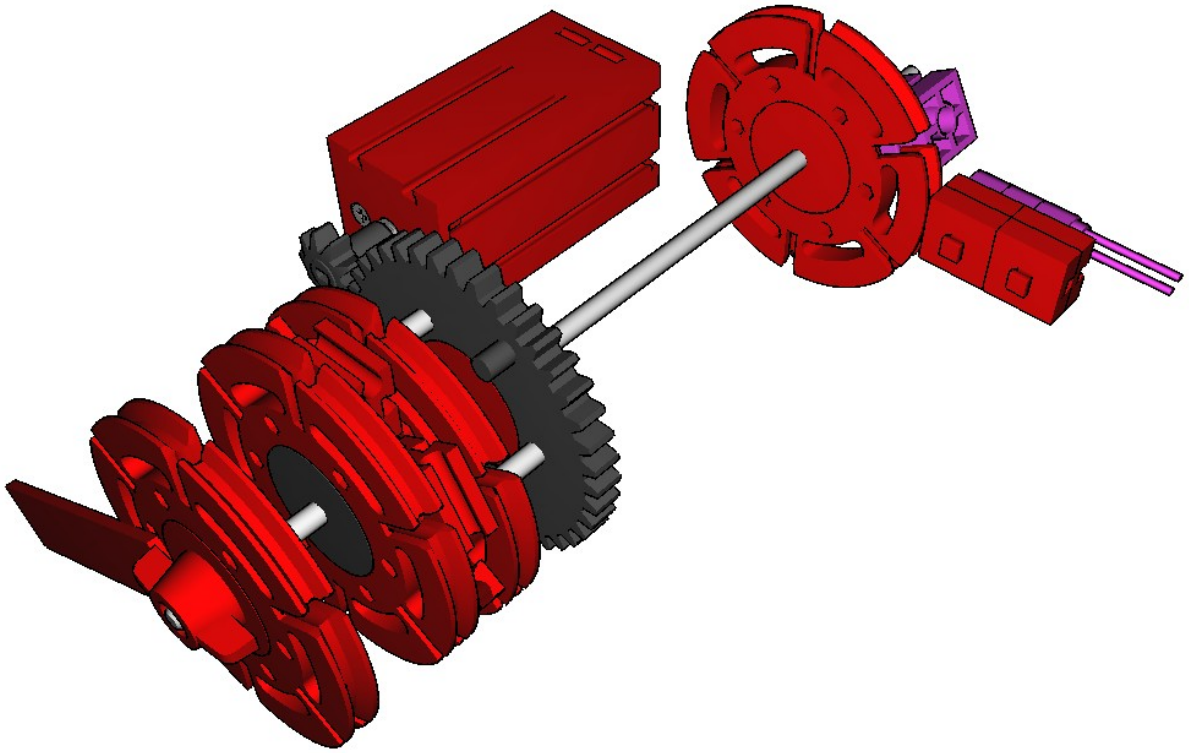


Abb. 4: Antrieb des Stundenzeigers, Reedkontakt zur Positionskalibrierung (violett)

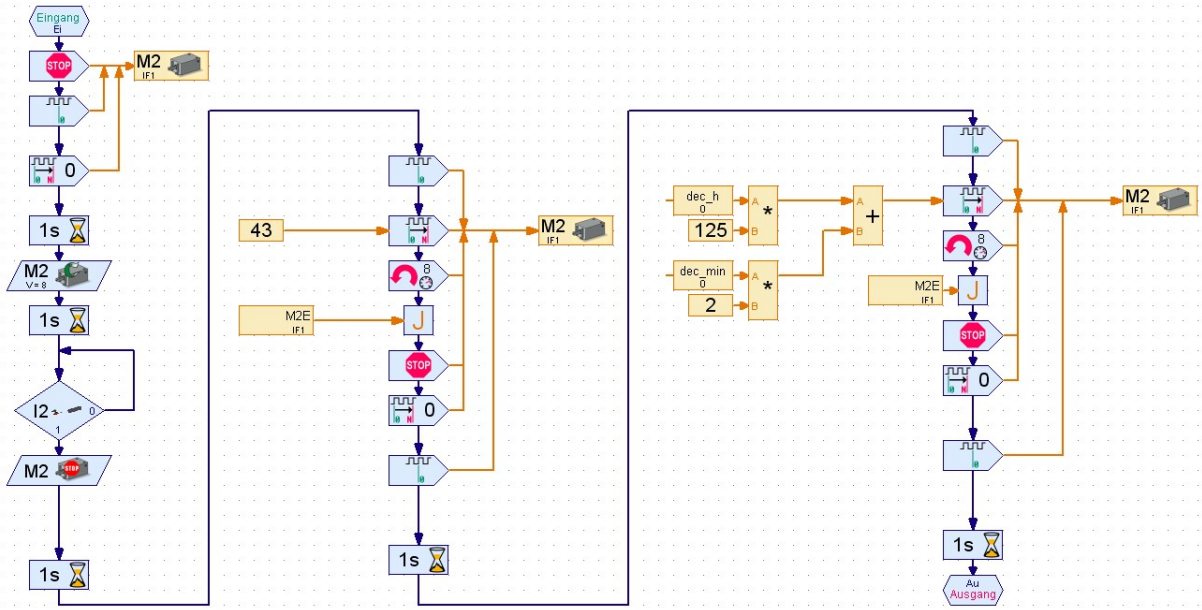


Abb. 5: Robo Pro Unterprogramm zur Kalibrierung der Zeigerposition des Stundenzeigers und Einstellung der aktuellen Zeigerposition auf Basis der aktuellen Uhrzeit, dec_h = Uhrzeit in Stunden, dec_min = Uhrzeit in Minuten, 1.500 Schritte entsprechen 12 Stunden

- Schritt 2: Ausgehend von der Position des Reedkontakts wird der Zeiger auf die 12-Uhr-Position gefahren. Der Verfahrweg wird dabei mithilfe der Schrittmotorfunktionalität durchgeführt. Dazu wurde im Vorfeld die hierzu erforderliche Anzahl an Schritten ermittelt.
- Schritt 3: Entsprechend der aktuellen Uhrzeit und der zugehörigen erforderlichen Zeigerstellung wird berechnet, um wie viele Schritte der Zeiger abschließend gedreht werden muss.

Dieses Verfahren wird nacheinander für den Stunden- und den Minutenzeiger angewendet. Abb. 5 zeigt das zugehörige Programm, die Spalten entsprechen den o.g. Vorgängen.

Detail-Lösung C

Antrieb des ringförmig gelagerten Zeigers der Weckzeit

Es gibt eine Reihe von Veröffentlichungen zum Thema Zahnräder im Internet; die folgende Variante habe ich bislang dort noch nicht gefunden. Im Aufbau sollten möglichst keine Spezialteile gefertigt werden müssen.

Problem: Wie kann der Ring angetrieben werden, und welche Lagerung ist dafür erforderlich?

Lösung: Der Aufbau eines Zahnrings unter Verwendung von fischertechnik-Standardteilen ist möglich, wie in Abb. 7 und 8 gezeigt. Der Knackpunkt hierbei liegt jedoch beim violett markierten Teil. Dieses Teil passt normalerweise nicht in die Riegellöcher der grauen Statikbauteile. Deshalb ist es erforderlich den Zapfen dieses Bauteils an zwei Seiten mit einer Schlüsselfeile so weit zu bearbeiten, bis er hineinpasst. Wenn dann eine 90°-Drehung erfolgt sitzt der Zapfen im grauen Riegelloch fest. Die spätere Demontage vom Statikbauteil ist beschädigungsfrei möglich.

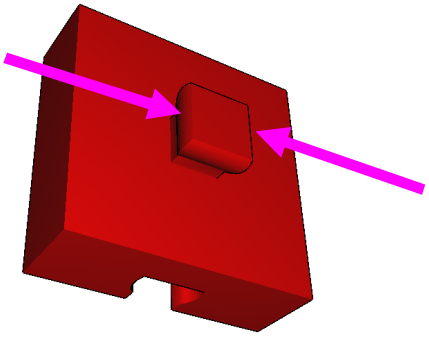


Abb. 6: Bauteilmodifikation an den markierten Zapfenkanten

Schlüsselfeilen gibt es in jedem Baumarkt. Gemäß Abb. 6 die Kanten leicht abfeilen, bis der Zapfen in das gewünschte Riegelloch passt.

Quellen

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

[2] Thomas Püttman: *Zahnräder und Übersetzungen (Teil 2)*. ft:pedia 3/2011, S. 25-28.

[3] Andreas Gail: *Schreiender Wecker*, Youtube

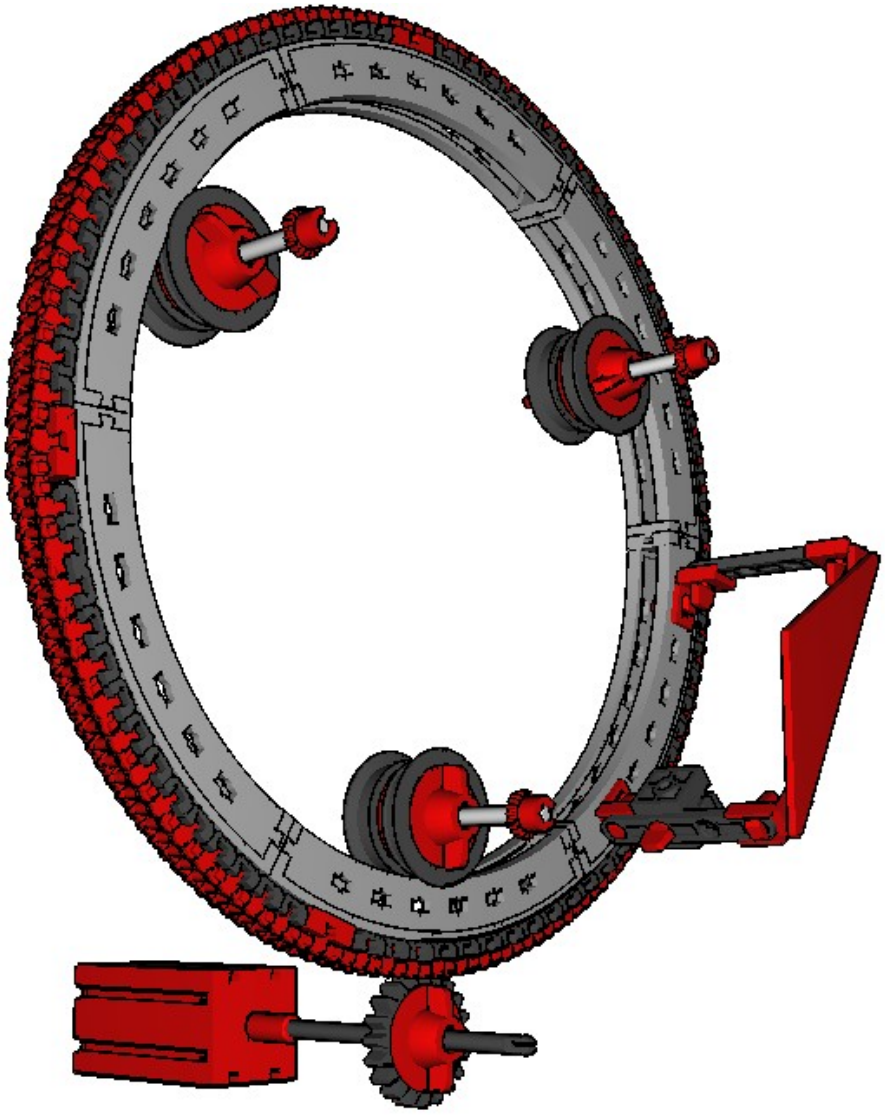


Abb. 7: Gesamtansicht des Zahnrings zur Lagerung des Zeigers der Weckzeit

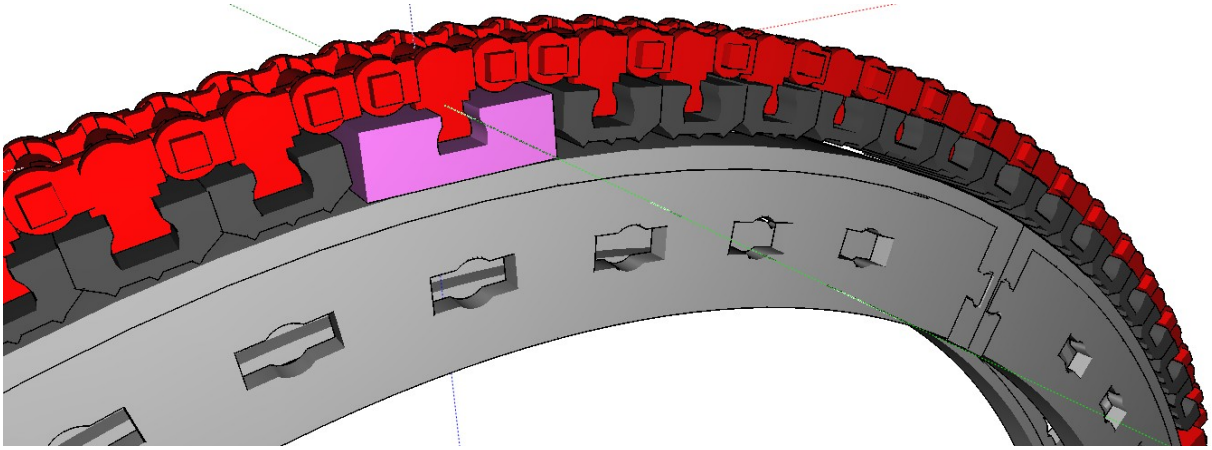


Abb. 8: Detailansicht des Zahnrings aufgebaut mit Standardteilen, violett markiertes Teil muss jedoch modifiziert werden

Elektronik

Vollautomatische Aussichtsplattform

Werner Hasselberg

In Ausgabe 4/2013 präsentierte ich eine vollautomatische Kransteuerung mit dem weiteren Ausblick, damit auch mal einen 3-achsigen Roboter steuern zu können. Die vollautomatische Aussichtsplattform ist dafür nun der nächste Schritt. Sie wird in einem späteren Artikel noch ein wenig verfeinert und schließlich zusammen mit der Schaltung aus ft:pedia 4/2013 (Automatik zur Steuerung eines Kranes) einen Roboter zum Leben erwecken. Doch zunächst wollen wir uns mit der neuen Schaltung beschäftigen. Das Schöne daran ist, dass sie alleinstehend betrieben werden kann und deshalb hervorragend geeignet ist, um ein eigenes Modell wie beispielsweise eine Hebebühne, einen Aufzug oder eben unsere Aussichtsplattform zu steuern. Der große Vorteil: Wir erhalten nicht erst in einer zukünftigen Ausgabe eine vollständige und verwendbare Steuerung, sondern bereits jetzt, und die Schaltung ist für sich allein genommen sicherlich auch besser verständlich.

Was ist also gefordert? Zunächst einmal die Plattform. Im Prinzip arbeitet sie wie ein Lift. Sie fährt nach unten, wartet solange bis alle Gäste eingestiegen sind und fährt dann wieder nach oben. Dort stoppt sie wieder eine gewisse Zeit, so dass auch alle den Ausblick schön genießen können, bevor es schließlich wieder nach unten geht, und so fort.

Ein schönes Beispiel einer solchen Plattform findet sich in den Hobbyheften, wie Abb. 1 zeigt. Hier fehlt nur noch die Motorisierung. Es ist aber unschwer zu erkennen, dass die Plattform über einen einfachen Seilzug nach oben gehoben wird.

Weit interessanter sind dagegen die auf der rechten Seite angebrachten Fotowiderstände in gelben Fassungen. Einer davon links oben, der andere rechts unten mit den dazugehörigen Lampenfassungen gegenüber. Sie bilden das Kernstück der Bühne, denn mit Ihrer Hilfe wird gesteuert, wie weit die Plattform nach oben und unten fahren soll.

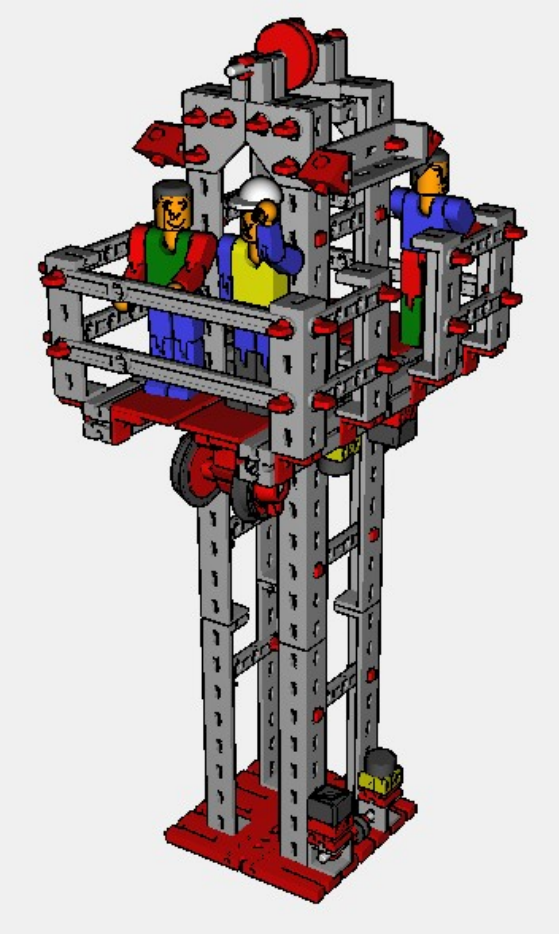


Abb. 1: Aussichtsplattform

Erforderliche Bauteile

- eine Spannungsversorgung aus dem Elektronikbaukasten von 1981
- ein Schwellwertschalter aus dem Elektronikbaukasten von 1981
- eine Leistungsstufe aus dem Elektronikbaukasten von 1981
- zwei Relais aus em3, dem em-5-Ergänzungskasten oder in Selbstbauweise
- beide Bausteine aus dem ec1 Baukasten (Gleichrichter und zugehöriges Relais)
- Grundbaustein aus dem ec2 Baukasten
- Widerstände, Dioden, Elko470, Elektromagnet

Dunkelimpuls speichern

In der ft:pedia 3/2013 wird dieses Prinzip im Artikel „Automatik für weichen Motorstart und -stopp“ bereits angewendet und erläutert. Hier benötigen wir es ebenso: Die beiden Fotowiderstände (kurz FW) werden im Schwellwertschalter (SWS) zwischen dem Br-Pin und der Minusschiene angebracht. Zusätzlich wird an jedem SWS ein 22 kΩ-Widerstand an (+) und EA bzw. EB eingefügt. Sie bilden mit den SWS-Potentiometern und den FW einen so dimensionierten Spannungsteiler, dass sich der SWS vorerst nicht einschaltet. Seine rote LED Leuchte bleibt aus. Sobald ein FW, z. B. der von SWS1, abgedeckt und damit „unterbrochen“ wird, schaltet sich dessen rote LED ein. $\overline{QA}$ wird deshalb zu (-), und das hat zur Folge dass sich der SWS2 ausschaltet, denn der Strom durch den Widerstand von EB fließt jetzt direkt über $\overline{QA}$ ab. Auf diese Weise wird der SWS2 ausgeschaltet, sobald sich der SWS1 einschaltet, und umgekehrt.

Arbeitsweise der Plattform

Betrachten wir als Ausgangspunkt die Plattform am Boden. Sobald der untere FW ausreichend beleuchtet wird (sagen wir, er befindet sich am SWS1), ist SWS1 ausgeschaltet. SWS2 ist demnach jetzt an, und weil (wie in Abb. 2 ersichtlich) die Leistungsstufe 2 (kurz LST2) an $\overline{QB}$ angeschlossen ist, bleibt diese aus. An C2 der LST 2 befindet sich das em3-Relais, das uns als Polwendeschalter dient. Es ist so gepolt dass es im ausgeschalteten Zustand die Plattform nach oben fahren lässt. Sobald sie (je nach Einstellung des SWS-Potentiometers) hoch genug ist, wird der FW von SWS1 zu gering beleuchtet und SWS1 damit eingeschaltet. Das schaltet wie oben erläutert den SWS2 sofort aus und das em3-Relais damit ein, was den Motor schließlich umpolt. Die Plattform fährt jetzt zurück nach unten. Dort angekommen wiederholt sich alles wieder von vorne.

Damit ist fast alles erledigt. Es fehlt aber noch ein wichtiger Punkt, denn unsere Plattform fährt im jetzigen Zustand der Schaltung ohne jegliche Unterbrechung rauf und runter. Das ist etwas bescheiden. Es fehlen noch die Wartezeiten fürs Ein- und Aussteigen. Wir brauchen also nach jedem Polwechsel des Plattformmotors noch eine Pause.

Plattform-Wartezeit

Das ist wieder die große Stunde der guten alten Silberlinge. Im ec2-Begleitheft auf Seite 53 wird eine sogenannte Ein-/Austastverzögerung vorgestellt, die uns jetzt sehr nützlich sein wird. Beim Drücken eines Tasters dauert es eine gewisse Zeit, bis sich das ec1-Relais einschaltet. Wird der Taster losgelassen, dauert es ebenfalls noch eine Weile bis es sich auf das ec1-Relais auswirkt und es wieder abfällt – genau das, was wir brauchen. Diese beiden Zustände verbinden wir einfach mit den beiden Polzuständen des em3-Relais.

So wird mit dem Ein- bzw. Ausschalten des zweiten em3-Relais (in Schaltbild 1 links unten) die jeweilige Verzögerung aktiv. Es schaltet sich aus, sobald die LST1 aus geht. LST1 aus bedeutet, dass ein Polwechsel

stattfindet, LST2 schaltet folglich sich selbst und damit das em3 Relais 1 ein. Der Strom muss dieses Relais durch a1-a3 passieren, damit der Motor wieder laufen kann. Das wird je nach Stellung des ec-Relais verhindert, und dieses wiederum wird vom ec-Grundbaustein gesteuert. Dessen Eingang E1 bekommt durch das em-Relais 2 und einen 22-kΩ-Widerstand Strom, der durch den 470-µF-Kondensator (aus dem Elektronik-Kasten von 1981) zum Gleichrichter aber nur langsam Spannung aufbauen kann. Je nach Potentiometer-Einstellung des Grundbausteins wird das ec-Relais also erst verzögert den umgepolten Strom zum Motor durchlassen. Das von LST1 gesteuerte em-Relais 2 steuert den Grundbaustein also verzögert in die jeweils benötigte Richtung. Nach dieser Zeit kann die Plattform wieder in die jeweils andere Richtung los fahren.

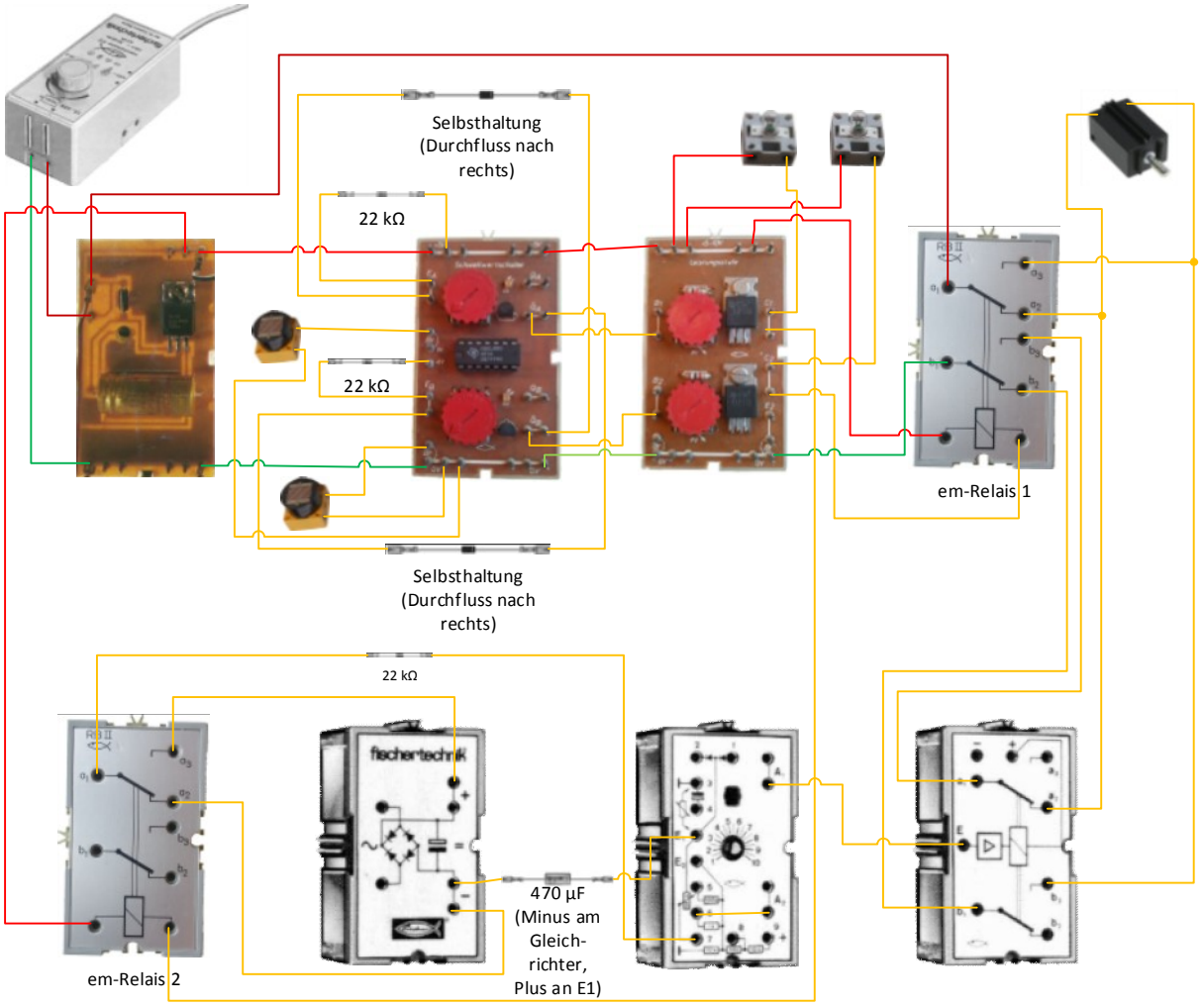
Damit haben wir die nötigen Fahrtunterbrechungen geschaffen. Fertig ist die Schaltung, und wir haben ein schönes, voll funktionsfähiges Deko-Modell für den Fun-Park oder die Bauspielbahn oder alles was die Fantasie erlaubt. Viel Spaß beim Basteln.

Quellen

[1] fischertechnik: *Elektronik*. Fischer-Werke, Tumlingen, 1981.

[2] fischertechnik: *ec2 (Begleitheft)*. Fischer-Werke, Tumlingen, 1975.

[3] fischertechnik: *hobby 1 Band 5* „Aussichtsturm mit Aufzug“. Fischer-Werke, Tumlingen, 1973



Schaltbild 1: Die vollständige Schaltung des Aussichtsturms

Computing

TX-Fernsteuerung mit dem Raspberry Pi

Raphael Jacob

In Ausgabe 3/2014 der ft:pedia wurde vorgestellt, wie man den TX-Controller mit dem IR-Empfänger verbinden kann [1]. Diese Art der Fernsteuerung des TX-Controllers hat jedoch verschiedene Nachteile. In diesem Beitrag wird gezeigt, wie man den TX-Controller über ein Web-Interface steuern kann.

Hintergrund

Für viele fischertechnik-Modelle könnte man eine komplexe Fernsteuerung gut gebrauchen. Weil man mit der IR-Fernbedienung von Fischertechnik nicht viele Funktionen auslösen kann und die Reichweite sehr begrenzt ist, musste eine andere Lösung her. Da ich ein wenig Erfahrung mit Linux habe und einen Raspberry Pi (kurz: RPi) besitze, habe ich diesen über die GPIO-Pins mit dem TX Controller verbunden.

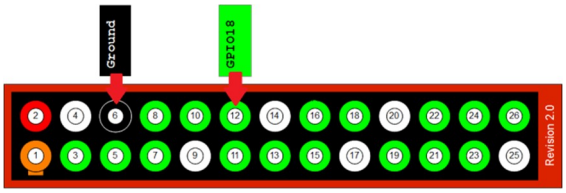


Abb. 1: Pinbelegung Raspberry Pi Rev. 2 [2]

Das Konzept

Mein Plan war, ein kleines Web-Interface zu programmieren (es ist ein sehr einfaches Web-Interface, daher sind individuelle Modifikationen leicht möglich). Darüber werden von einem Python-Skript Binärdaten an den TX-Controller geschickt, die dieser verarbeiten kann.

Der Versuchsaufbau

Abb. 2 zeigt die Verdrahtung der Verbindung des RPi mit dem TX Controller. Dabei wird ein Eingang des TX als unidirektionale Datenleitung verwendet, um binäre Werte seriell zu übertragen.

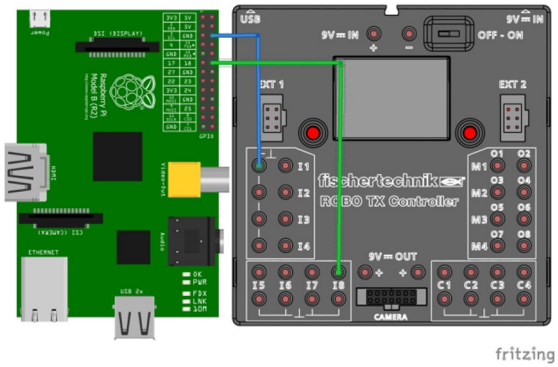


Abb. 2: Die Verdrahtung des RPi mit dem TX Controller [3, 4]

- Die Verbindung zwischen dem GPIO Ground Pin (Pin 6) und dem Ground Pin des TX ist Voraussetzung dafür, dass der TX-Controller und der RPi überhaupt miteinander kommunizieren können (wie beim I²C-Bus).
- Die Verbindung zwischen dem GPIO Pin 18 (Pin 12) und dem I8-Anschluss des TX dient der Datenübertragung vom RPi an den TX Controller.

Zusätzlich benötigen wir noch eine Internetverbindung für den RPi. Am einfachsten ist es, den RPi per Ethernet-Kabel an einen Router oder Switch anzuschließen. Für mobile Modelle wird später eine WLAN-Verbindung konfiguriert.

Die Inbetriebnahme

Zunächst wird die mit dem aktuellen Image bespielte SD-Karte in den RPi eingelegt. Nun wird der Raspberry Pi mit dem Ethernet-Kabel verbunden.

Jetzt verbinden wir den TX-Controller mit dem RPi und dem PC (via USB oder Bluetooth) und stellen die Stromversorgung her: beim RPi mit einem Handyladegerät oder einem Micro-USB-Kabel, das mit dem PC verbunden wird, beim TX mit dem Netzteil oder einem fischertechnik-Akku.

Nun verbinden wir uns mit dem RPi. Dafür laden wir die Software putty herunter und geben folgende Daten ein:

- *Hostname*: raspberrypi oder IP-Adresse
- *Connection type*: ssh
- Open-Button drücken
- *Username*: pi
- *Password*: raspberry

Nach einer erfolgreichen Verbindung sollten wir noch das Betriebsystem des RPi durch folgende Befehle aktualisieren [5]:

- sudo -i (mit finish bestätigen)
- sudo -i
- apt-get update
- apt-get upgrade

Um die Dateneingabe zu vereinfachen, findet ihr im Download-Bereich der ft:c (in der neuen Rubrik „ft:pedia Dateien“) ein Paket mit allen Dateien und Befehlen.

Der Webserver

Als Webserver installieren wir Apache 2. Das gelingt mit dem folgenden Befehl:

- apt-get install apache2 apache2-doc apache2-utils libapache2-mod-php php5 php-pear php5-xcache

Nun kann man unter der IP-Adresse des RPis die Apache-Testseite sehen (Abb. 3). Sollte dies nicht der Fall sein, habt ihr etwas falsch gemacht.

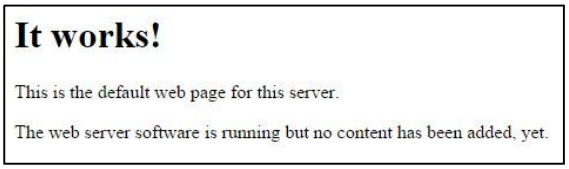


Abb. 3: So sieht die Apache-Testseite aus

Nun möchten wir unsere eigene Webseite erstellen. Auf dieser sollen ein Start- und ein Stopp-Button angezeigt werden, über die das fischertechnikmodell gestartet und gestoppt werden kann. Die Befehle lauten:

- sudo -i
- cd /var/www
- wget https://www.iconfinder.com/icons/61447/download/png/64
- mv 64 start.png
- wget https://www.iconfinder.com/icons/61441/download/png/64
- mv 64 stop.png
- nano index.html

Jetzt sehen wir einen Texteditor, in dem wir zuerst den gesamten Text löschen. Der neue Text lautet:

```
<html>
 <head>
  <title>Fischertechnik</title>
 </head>
 <body>
  <br><br><br>
  <center>
   <a href="start.php"><img
src="start.png"></a>
   <a href="stop.php"><img
src="stop.png"></a>
  </center>
 </body>
</html>
```

Gespeichert wird mit Strg+O und Enter; mit Strg+X verlässt man den Editor.

Ein erneuter Blick auf die Website zeigt die beiden Buttons (Abb. 4).

Abb. 4: So sehen die Symbole aus

Wenn man auf eines der Symbole klickt, bekommt man eine ‚*Error 404 Not Found*'-Meldung, denn noch fehlen die php-Skripte, die beim Anklicken aufgerufen werden sollen. Legen wir nun die php-Skripte an:

- nano start.php

Im Editor geben wir ein:

```
<?php
shell_exec("sudo python
/var/www/start.py");
header ("Location:index.html");
?>
```

Wir speichern das Skript und schließen mit Strg+O, Enter und Strg+X.

Es folgt das zweite php-Skript:

- nano stop.php

```
<?php
shell_exec("sudo python
/var/www/stop.py");
header ("Location:index.html");
?>
```

Wenn man nun erneut auf eines der Symbole auf der Startseite klickt, wird im Hintergrund versucht, eines der beiden Python-Skripte `start.py` bzw. `stop.py` auszuführen – was wieder mit einer Fehlermeldung im Apache-Log endet, und man wird zurück auf die Startseite geleitet. Um keine Fehler hervorzurufen, muss der RPi neu gestartet werden:

- reboot

Die Datenübertagung

Die Daten werden in drei Bits vom RPi zum TX Cotroller übertragen. So sind acht verschiedene Werte (0 bis 7) möglich (Tab. 1). Dieses einfache „Protokoll" lässt sich ausbauen: Wenn ein Bit übertragen wird, sendet der RPi ein High-Signal. Fällt dieses innerhalb von 0,1 Sekunden ab, ist der Wert dieses Bits 0; bleibt das Signal auf High, ist der Wert 1. Erst nach Abfallen der Spannung ist der TX Controller bereit, weitere Bits anzunehmen.

Bit 0	Bit 1	Bit 2	Wert
0	0	0	0
1	0	0	1
0	1	0	2
1	1	0	3
…			
1	1	1	7

Tab. 1: Binäre Kodierung der Daten

Die Software auf der Seite des RPi ist ein Python-Skript, welches von einer php-Datei aufgerufen wird. Wir programmieren nur die Python-Skripte für die Werte 0 und 1. Weitere Skripte lassen sich ohne Probleme ergänzen.

Auf dem RPi werden mit die Skripte den folgenden Befehlen erstellt:

- sudo –i
- cd /var/www
- nano start.py

Der Text lautet:

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO_toTX = 18
GPIO.setup(GPIO_toTX,GPIO.OUT)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
```

```
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
```

Wir speichern den Text ab und schreiben das Skript zum Starten des Modells:

- nano stop.py

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO_toTX = 18
GPIO.setup(GPIO_toTX,GPIO.OUT)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.HIGH)
time.sleep(0.1)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.2)
GPIO.output(GPIO_toTX, GPIO.LOW)
time.sleep(0.1)
```

Damit diese Skripte ausgeführt werden können geben wir ein:

- chmod 777 start.py
- chmod 777 stop.py
- visudo

An das Ende der Datei hängen wir an:

```
www-data ALL=(ALL) NOPASSWD: ALL
```

Die Änderungen werden wirksam, wenn der Raspberry Pi neu gestartet wird. Der Befehl dafür lautet (wie oben):

- reboot

Passwortschutz

Um unsere Webseite vor fremder Benutzung zu schützen, kann ein Passwortschutz eingerichtet werden [6]:

- sudo –i
- nano /etc/apache2/apache2.conf

An das Ende der Datei hängen wir an:

```
<directory /var/www>
AuthType Basic
AuthName "Passwort eingeben:"
AuthUserFile
"/etc/htpasswd/.htpasswd"

# in der naechsten Zeile muss an-
# gegeben werden ob nur ein be-
# stimmter Benutzer oder alle
# Benutzer auf die Datei Zugriff
# haben.
# Bei ‚alle Benutzer‛ heisst die
# Zeile:
# Require valid-user

Require user ft
Order allow,deny
Allow from all
</directory>
```

- mkdir /etc/passwd
- htpasswd -c /etc/.htpasswd NAME-DES-NUTZERS

Passwort eingeben, dann weiter:

- /etc/init.d/apache2 reload
- reboot

Abb. 4 zeigt das Fenster, in dem bei Programmablauf das Passwort abgefragt wird.

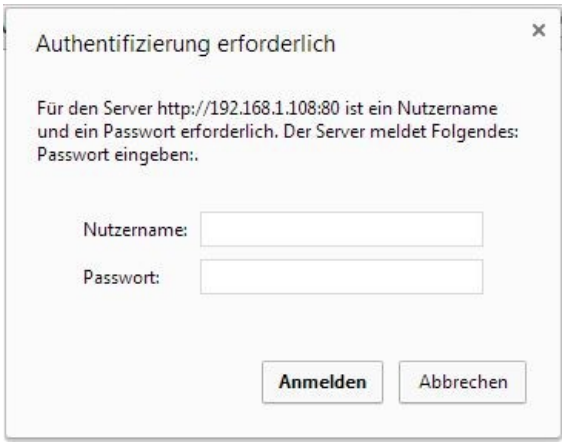


Abb. 4: Passwortabfrage am Beispiel von Google Chrome

W-LAN Verbindung

Damit auch mobile Modelle kabellos gesteuert werden können, benötigen wir einen W-LAN-Adapter. Dieser wird an einen der USB-Anschlüsse gesteckt und wie folgt konfiguriert [7]:

- sudo -i
- nano /etc/network/interfaces

Wir löschen den gesamten Text und schreiben:

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf
/etc/wpa_supplicant/wpa_supplicant
.conf
```

- iwlist scanning

Dort merken wir uns folgende Dinge zu unserem Netzwerk:

- pairwise
- group

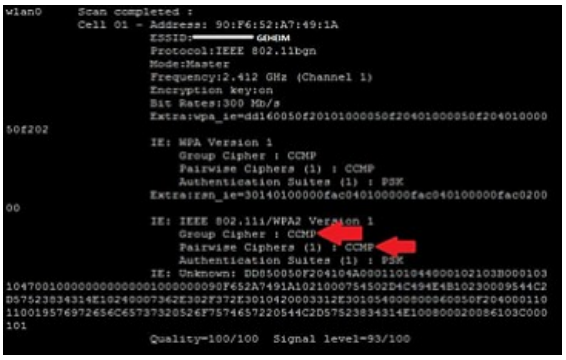


Abb. 5: Die Ausgabe des Befehls mit Markierung der Informationen

Diese Daten werden in die folgende Datei eingetragen:

- nano /etc/wpa_supplicant/ wpa_supplicant.conf

```
ctrl_interface=DIR=/var/run/wpa_su
pplicant GROUP=netdev
update_config=1

network={
 ssid=MEINE-SSID
 scan_ssid=1
 proto=RSN
 key_mgmt=WPA-PSK
 pairwise=CCMP
 group=TKIP
 psk="MeinPasswort"
}
```

Damit sich nach einer abgebrochenen W-LAN-Verbindung der RPi wieder automatisch verbindet, müssen vorab folgende Befehle eingegeben werden. [9]:

- cd /etc/ifplugd/action.d
- mv ifupdown ifupdown.original
- cp /etc/wpa_supplicant/ifupdown.sh /etc/ifplugd/action.d/ifupdown

Schlusswort

Mit dieser Art der Fernsteuerung des TX Controllers lassen sich theoretisch unendlich viele Funktionen steuern, jedoch dürfen das Modell und das Steuergerät (Smartphone, Tablet, PC, …) nur ca. 50 m (im Freifeld) voneinander entfernt sein.

Wie der Raspberry Pi Daten vom TX Controller empfangen und diese in Form einer E-Mail an uns weiterleiten kann, erfahrt ihr in der nächsten Ausgabe der ft:pedia.

Referenzen

[1] Andreas Gail: *Detail Engineering R2D3 (3) IR-Fernbedienung am Robo TX Controller*, ft:pedia 3/2014, S. 51-54

[2] WebRaPi: *Raspberry – Remote installieren*, swek.de

[3] Software für den Schaltplan: fritzing.org

[4] Thomas Dragon: *TX Controller für fritzing* (*fischertechnik Robo TX Controller*), 22.06.2014

[5] Manfred Steger: *Der eigene Webserver mit dem Raspberry Pi*. 22.06.2013

[6] sempervideo: *Alarm-Anlage (Teil 1 von 5)*, Youtube

[7] sempervideo: *WLAN Auto-Reconnect*, Youtube

Computing

Strichcode-Leser am Robo TX Controller (2): Automatisiert mit Microsoft Visual Basic

Andreas Gail

Allgemeines über Strichcodes oder auch Barcodes kann im ersten Teil des Beitrags nachgelesen werden, ebenso der Bau eines Scanners mit Standard-fischertechnik-Teilen. Als Alternative zur RoboPro Software aus der vorherigen Ausgabe der ft:pedia soll im vorliegenden Teil 2 die Automatisierung vollständig mithilfe von Microsoft Visual Basic 2010 oder höher erfolgen.

Warum überhaupt VB?

fischertechnik stellt mit der RoboPro-Software eine einfach zu erlernende, sehr anschauliche Möglichkeit zur Automatisierung bereit. Dem Einsteiger wird so ein anschaulicher Weg aufgezeigt, sinnvolle Abläufe zunächst zu erfassen, festzulegen und schließlich zu programmieren. Aber wo Licht ist, ist auch Schatten. So bietet es durchaus Vorteile, eine Standard-Programmierumgebung zu verwenden, um seine Automatisierungspläne umzusetzen. Insbesondere Basic heißt so (BASIC = *Beginners All System Instruction Set*), weil es bis heute in seinen Grundzügen eine gewisse Einfachheit erhalten konnte. Durch seine konsequente Weiterentwicklung ist Basic z. B. in Form von Microsoft Visual Basic Express (VB) zu einen sehr mächtigen Werkzeug geworden, was für den Hobby Programmierer kaum Wünsche offen lässt.

Obendrein stellt Microsoft diese Express-Version aktuell kostenfrei zur Verfügung. Entscheidend dabei sind jedoch die im Grunde unlimitierten Möglichkeiten (im Online-Betrieb): Viele Dinge, die unter Nutzung von RoboPro viele Seiten umfassender Konfigurationen erfordern, lassen sich unter Basic mit einer überschaubaren Anzahl an Codezeilen abbilden. Es können im Grunde Programme mit unlimitiertem Umfang und einer ebenso unlimitierten Anzahl an Robo TX Controllern (RTXC) verwirklicht werden. Auch die Visualisierung lässt kaum Wünsche offen, einschließlich einer ggf. gewünschten Kopplung mit dem Internet.

Kurzum: VB ist definitiv eine Erweiterung der Möglichkeiten.

Mehraufwand am Anfang

Bei so vielen Vorteilen soll nicht verschwiegen werden, dass erstmal ein gewisser Mehraufwand erforderlich ist, um VB nutzen zu können. So muss die Kommunikation mit dem RTXC erstmal organisiert werden. Auch muss man sich gewisse zusätzliche Gedanken machen, wenn Vorgänge gleichzeitig ablaufen sollen. Ein Grundgerüst steht unter http://www.ftcommunity.de in Form eines vollständigen Visual Basic-Projekts zur Verfügung.

Visualisierung

Abb. 1 zeigt ein Beispiel für eine Visualisierung von Programmabläufen. Hierbei erfolgt eine animierte Anzeige entsprechend der aktuell laufenden Vorgänge. Neben der bildlichen Darstellung des Modells wird rechts das Leseergebnis des Scanners angezeigt, in diesem Fall die Ziffer 4. Im unteren Bereich können alle

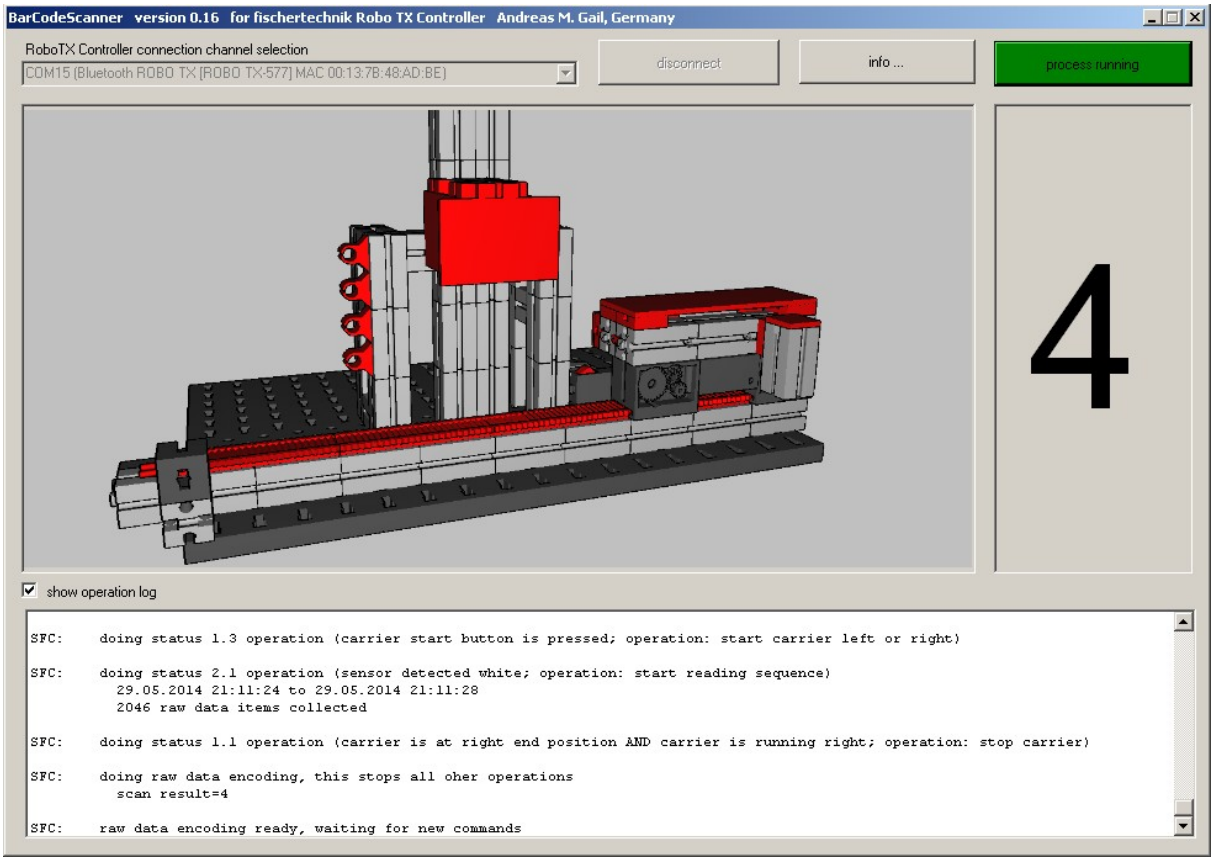


Abb. 1: Strichcode-Leser automatisiert mit Microsoft Visual Basic Express mit animierter Visualisierung des Programmablaufs und Anzeige aktueller Vorgänge (operation log)

aktuell laufenden Vorgänge als textliche Abfolge (*operation log*) verfolgt werden. Die genauen Details der Bedienoberfläche soll hierbei nicht weiter erläutert werden; vielmehr soll beispielhaft aufgezeigt werden, was möglich ist. Ich bin auf weitere Beispiele der Leserschaft gespannt...

Herzstück ftMscLib.dll

Das softwareseitige Herzstück zur Ansteuerung des RTXC ist sicherlich die von fischertechnik bereit gestellte Programmbibliothek *ftMscLib.dll*. Damit wird die Kommunikation zwischen PC und dem RTXC hergestellt. Der RTXC selbst hat dabei nur wenige Aufgaben, wie auch bei der Nutzung von RoboPro im Online-Modus. Online hat der Microcontroller im RTXC nur die Aufgabe, Eingangszustände auszulesen und weiter zu melden, sowie vom PC gewünschte Ausgangszustände einzustellen (remote I/O).

Die Hauptschwierigkeit der Nutzung von VB zusammen mit dem RTXC ist die Einbindung der Programmbibliothek *ftMscLib.dll*. Diese enthält „native code“ („unmanaged“ in .net-Sprechweise) und kann als solche nicht so direkt von der „managed code“-Welt von .net angesteuert werden, wie das bei einer .net-DLL der Fall wäre. Vielmehr müssen alle gewünschten Funktionen oder Prozeduren zuvor entsprechend den jeweils verwendeten Datentypen deklariert werden. Weil das alles andere als trivial ist, wird dies in Abb. 2 gezeigt. Vielen Dank an Carel van Leeuwen (Niederlande) für entscheidende Hilfestellung. Mit den abgebildeten Beispielen lassen sich auch die restlichen fehlenden Funktionen der Programmbibliothek ableiten.

Download

Für den interessierten Leser stehen auf der ft-Community-Website die kompletten VB-Projektdateien zur Verfügung [1].

Was nun?

Dieser Beitrag soll als Anregung dienen und viele weitere Projekte beflügeln. So könnte z. B. ein Erkundungsroboter Bilddaten aus seiner Umgebung an den PC senden, deren Auswertung unter VB programmiert ist. Oder denkt an ein autonomes Gefährt, welches seine aktuelle Position per GPS ermittelt und an den PC sendet, mit gleichzeitiger Abbildung der Position auf Satellitenbildern aus dem Internet, oder...

Gleichzeitig darf man gespannt sein, was fischertechnik der Fangemeinde mit dem angekündigten Robotics TXT Controller an neuen Möglichkeiten anbieten wird.

Quellen

[1] Gail, Andreas: *Software Barcode-Scanner RoboPro/Visual Basich 2010*.

[2] Microsoft Developer Network: *Exemplarische Vorgehensweise: Aufrufen von Windows-APIs (Visual Basic)*, 2013

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

Public Class classFtMscLib

    '*** List of var

    Public Const dllpath As String = "C:\windows\system32\ftMscLib.dll"


    '*** List of declarations
    Public Class FtLibDeclarations

        <DllImport(dllpath, EntryPoint:="ftxGetLibVersionStr",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxGetLibVersionStr(<MarshalAs(UnmanagedType.LPStr)> ByVal Foo2
As StringBuilder, ByVal len As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxGetLibVersion",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxGetLibVersion() As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxIsTransferActiv",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxIsTransferActiv(ByVal fthdl As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="GetAvailableComPorts",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxGetAvailableComPorts(ByVal selectMode As Integer) As
UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="EnumComPorts",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxEnumComPorts(ByVal idx As UInteger,
<MarshalAs(UnmanagedType.LPStr)> ByVal comstr As StringBuilder, ByVal len As UInteger) As
UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxOpenComDevice",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxOpenComDevice(ByVal comStr As String, ByVal bdr As UInteger,
ByRef uint As UInteger) As UInteger
```

```
        End Function

        <DllImport(dllpath, EntryPoint:="ftxOpenComDeviceNr",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxOpenComDeviceNr(ByVal comStr As UInteger, ByVal bdr As
UInteger, ByRef uint As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxInitLib",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxInitLib() As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxIsLibInit",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxIsLibInit() As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxIsHandleValid",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxIsHandleValid(ByVal fthdl As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="GetComStatus",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxGetComStatus(ByVal fthdl As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxCloseLib",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxCloseLib() As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxGetLibErrorString",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxGetLibErrorString(ByVal errCode As UInteger, ByVal typ As
UInteger, <MarshalAs(UnmanagedType.LPStr)> ByVal buff As StringBuilder, ByVal maxLen As
UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxCloseDevice",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxCloseDevice(ByVal fthdl As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxStartTransferArea",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxStartTransferArea(ByVal fthdl As UInteger) As UInteger
        End Function

        <DllImport(dllpath, EntryPoint:="ftxStopTransferArea",
CallingConvention:=CallingConvention.Cdecl)> _
        Public Shared Function ftxStopTransferArea(ByVal fthdl As UInteger) As UInteger
        End Function


'hier weitere Funktionen ergänzen oder unter www.ftcommunity.de laden
'http://ftcommunity.de/data/downloads/software/barcodescannerroboproundvisualbasic2010.zip


    End Class
```

Abb. 2: Deklarationen zur Ansteuerung bzw. Nutzung der ftMscLib.dll (getestet mit Microsoft Visual Basic 2010)

Computing

I²C mit dem TX – Teil 11: Pixy-Kamera (1)

Dirk Wölffel, Dirk Fox

Seit die I²C-Anbindung beim Robo TX Controller funktioniert sind ganz neue Möglichkeiten für fischertechnik-Modelle entstanden. Ein wichtiger Sensor fehlte allerdings noch in der Sammlung: eine Kamera. Die CMUcam5 (kurz: Pixy) ist eine I²C-Kamera, die sich an den Robo TX Controller anschließen lässt. Sie kann Objekte in bis zu sieben verschiedenen Farben erkennen, gibt die Koordinaten des Objekt-Mittelpunkts und sogar dessen Länge und Breite aus. Damit lassen sich Modelle nun um intelligente Bildverarbeitung ergänzen und so z. B. ein schneller Sortierroboter oder sogar ein Cube Solver in RoboPro realisieren.

Hintergrund

Seit dem Erscheinen des TX Controllers im Jahr 2009 findet sich in der Packungsbeilage der Hinweis auf die bald erscheinende Kamera zum Anschluss an die „Camera"-Buchse. Fünf Jahre warteten die TX-Käufer vergeblich – bis die Kamera nun mit dem Nachfolge-Controller TXT ausgeliefert wird. Am TX lässt sich die neue Kamera jedoch nicht betreiben.

Wie sicher einige andere TX-Käufer haben wir in den vergangenen Jahren immer wieder nach einer Kamera Ausschau gehalten, die sich möglicherweise mit Bordmitteln des TX nutzen lässt. Da der TX über keinen schnellen USB-Anschluss verfügt, an den sich externe Geräte anschließen ließen, sind einer solchen Lösung jedoch enge technische Grenzen gesetzt. Denn über die einzige verbleibende Daten-Schnittstelle, den I²C-Bus, lassen sich maximal 400 kBit/s übermitteln – das ist bei weitem zu wenig, um selbst im Sekundentakt erkennbare Bilder zu übermitteln.

Die Pixy CMUcam5

Dirk Wölffel hat schließlich die Lösung gefunden: eine Kamera, die die Bildauswertung in der Hard- und Firmware vornimmt, sodass über die Daten-Verbindung mit dem TX nur noch die Ergebnisse der Analyse übermittelt werden müssen. Eine solche Kamera wurde von Charmed Labs entwickelt, einem *Spin Off* der Carnegie Mellon University (CMU): Die kurz „Pixy" genannte CMUcam5 (Abb. 1) [1].

Abb. 1: Pixy – CMUcam5 (Bild: CMU)

Im Herbst 2013 startete Charmed Labs eine Kickstarter-Kampagne zur Vorfinanzierung der ersten „Charge". Mit Erfolg: 2.800

Interessierte unterstützten das Startup mit 275.000 US$ [2]. Herausgekommen ist eine einfach zu bedienende Kamera als Open-Source Projekt.

Seit Ende April 2014 ist die Kamera nun allgemein käuflich zu erwerben – in Deutschland unter anderem bei Watterott electronic und noDNA für ca. 65 €.

Technische Daten

- *Prozessor*: NXP LPC4330, 204 MHz, Dual Core
- *Image Sensor*: Omnivision OV9715, 1/4", bis zu 1.280 x 800 Pixel
- *Bildbereich*: 75° horizontal, 47° vertikal
- *Stromaufnahme*: 140 mA
- *Spannung*: USB (5V), Power In (6-10V) oder I²C (5V)
- *Speicher:* 264 kB RAM, 1 MB Flash
- *I²C-Adresse*: 0x54 (frei änderbar)

Abb. 2: Pixy-Kamera in Kassette

Die Objekterkennung der Pixy-Kamera identifiziert bis zu 135 Objekte je Frame – bei 50 Frames pro Sekunde. Der Prozessor NXP LPC4330 beherrscht das I²C-Protokoll im *Fast Mode Plus* (bis 1 MBit/s). Wie es der Zufall will, arbeitet der I²C-Controller auch noch mit fischertechnik kompatiblen 5 V und kann daher direkt am TX betrieben werden. Schließlich ist die Pixy mit 54 x 44 mm so kompakt, dass sie in eine fischertechnik-Kassette (32076) mit Klarsichtdeckel (35360) passt (Abb. 2).

Anschlüsse

In Abb. 3 seht ihr die Rückseite der Pixy-Kamera mit den Anschlüssen.

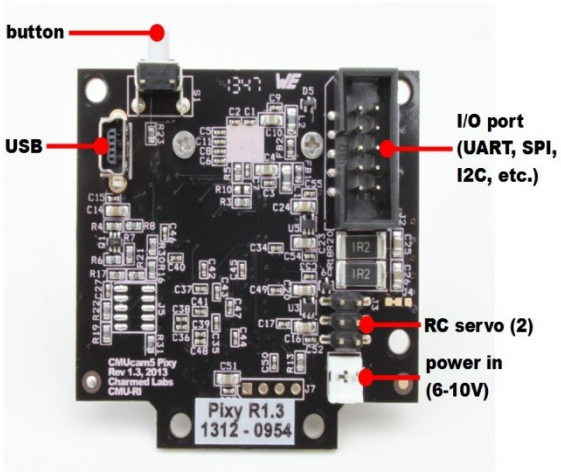


Abb. 3: Rückseite der Pixy (Bild: CMU)

- Button: Knopf für das Anlernen von Objekten und zum Aktualisieren der Firmware
- USB: Mini-USB-Anschlussbuchse
- I/0 Port: zehnpolige Pfostenbuchse für die serielle (UART, SPI, I²C) und analoge Daten-Ausgabe; 5V Betriebsspannung
- RC Servo: Anschluss für zwei Servos (Pan/Tilt)
- Power in: Stromversorgung (6-10V, optional – für unsere Nutzung nicht erforderlich)

Verbindungskabel

Die Pixy-Kamera wird mit einem sechsadrigen Verbindungskabel (SPI) ausgeliefert, auf dem auf einer Seite eine sechspolige Pfostenbuchse sitzt. Leider ist es nur für den Arduino geeignet, da die Belegung der Pfostenbuchse nicht zu der des EXT2-Anschlusses am Robo TX Controller passt.

Daher müssen wir uns mit einer eigenen Adapter-Konstruktion behelfen. Die Pin-Belegung des Wannensteckers zeigt die Abb. 4: 5V an Pin 2, SCL an Pin 5, SDA an Pin 9 und GND an Pin 10.

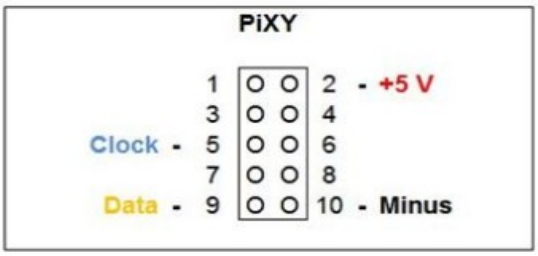


Abb. 4: Anschlussbelegung der Pixy-Kamera

Für die ersten Experimente mit der Pixy bietet sich die Verwendung des „Universaladapters“ aus der ft:pedia 4/2013 [a] an (Abb. 5). Dessen Female-Jumper lassen sich fest auf die Kontaktstifte des zehnpoligen Pixy-Wannensteckers aufstecken.

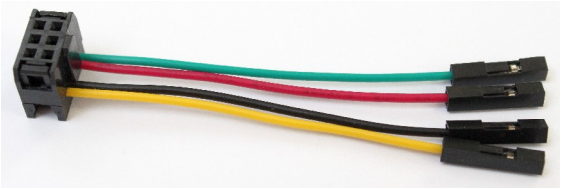


Abb. 5: I²C-Universaladapter [a]: (grün: GND, rot: VCC, schwarz: SDA, gelb: SCL)

Sofern ihr die Kamera nicht am Arduino nutzen wollt, könnt ihr auch die zehnpolige Pfostenbuchse des mitgelieferten Kabels abnehmen und die vier relevanten Kabel in der richtigen Reihenfolge an die Buchse anschließen (Abb. 6). Etwas fummelig, aber ihr müsst weder Kabel noch Buchsen beschaffen.

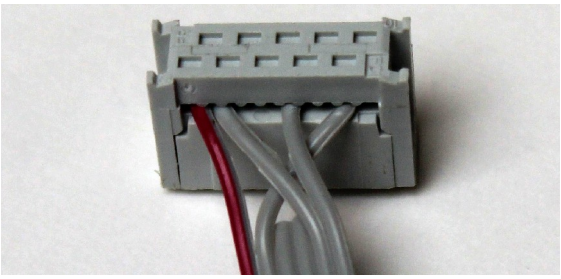


Abb. 6: Modifiziertes Pixy-Anschlusskabel

Doch Vorsicht: die Pixy-Kamera reagiert empfindlich auf einen falschen Anschluss des VCC-Kabels, daher sollte man das Verbindungskabel vor Benutzung lieber noch einmal genau durchmessen.

Für die Verbindung zwischen dem PC und der Pixy-Kamera benötigt ihr außerdem ein ‚Mini-USB auf USB‘-Kabel (wie das, das dem TX beiliegt).

Installation PixyMon und Update

Nach Erhalt der Kamera muss man zunächst die aktuelle Version der Software PixyMon (in der zum PC-Betriebssystem passenden Version) herunterladen und installieren. Unter Windows werden dabei die erforderlichen USB-Treiber automatisch installiert. Anschließend sollte man die Firmware der Pixy-Kamera auf die aktuelle Version (derzeit v1.0.2beta) updaten. Wie das geht, ist Schritt für Schritt und gut verständlich im Pixy-Wiki unter Uploading New Firmware beschrieben und sei daher hier ausgespart. Falls dabei etwas schiefgehen sollte, kann man jederzeit auf die alte Firmware (z. Zt. v0.1.44) zurückwechseln.

Konfiguration PixyMon

Derzeit erlaubt die Firmware nicht, die Kamera-Parameter via I²C-Protokoll einzustellen – dafür muss der USB-Monitor PixyMon gestartet werden.

Hier die für uns relevanten Einstellungen:

- **I²C-Adresse**: 7-bit-I²C-Adresse (Voreinstellung: 0x54).
- **Datenport**: Wahl des Übertragungsprotokolls: 0: SPI, 1: I²C, 2: UART, 3: analog/digital X, 4: analog/digital Y (voreingestellt ist Port 0).
- **maximale Block-Anzahl**: Zahl der je Frame maximal zu identifizierenden Objekte (Voreinstellung: 1.000).

- **maximale Block-Anzahl je Farbsignatur**: Zahl der maximal zu identifizierenden Objekte derselben Farbe (Voreinstellung: 1.000).
- **minimale Objektgröße**: Mindestzahl der Pixel eines zu erkennenden farbigen Objekts (Voreinstellung: 20).
- **Farbmodus**: Erkennung ein- und mehrfarbiger Objekte (Voreinstellung: 1)
- **Kamera-Helligkeit**: Wert zwischen 0 und 288 (Voreinstellung: 90).

Zunächst müsst ihr den Datenport 1 wählen, damit die Pixy via I²C erreichbar ist. Die voreingestellte I²C-Adresse 0x54 kollidiert mit den Adressen des TX-EEPROMS und sollte daher auch gleich geändert werden – in eine Adresse, die auf dem Bus zu keiner Adresskollision mit einem anderen Sensor oder Aktor führt. In unseren Beispielprogrammen zu diesem Beitrag und dem Pixy-Treiber verwenden wir die Adresse 0x14.

Die Maximalzahl der erkennbaren Objekte (Blöcke und Blöcke je Farbsignatur) kann man getrost deutlich reduzieren – so wird es bei fast allen Anwendungen genügen, einige wenige Objekte zu erkennen. Außerdem reicht die Geschwindigkeit des I²C-Protokolls ohnehin nur für die Übermittlung von maximal 50 Blöcken (näheres weiter unten).

Mit der minimalen Objektgröße filtert ihr kleine Farbflächen heraus, die die Pixy erkennt. Soll das Programm nur große Objekte erkennen, kann der Wert auch deutlich größer gewählt werden.

Testbetrieb

Ihr müsst für ausreichend helles Licht sorgen, damit die Objekte auch gut erkannt werden. Dann sucht ihr ein Objekt, am besten in einer kräftigen Farbe, und stellt dies vor die Linse. Jetzt startet ihr den PixyMon. Mit den Einstellungen müsst ihr etwas experimentieren, damit die Pixy-Kamera die Objekte sauber erkennt. Hier die von uns empfohlenen Einstellungen (Abb. 7).

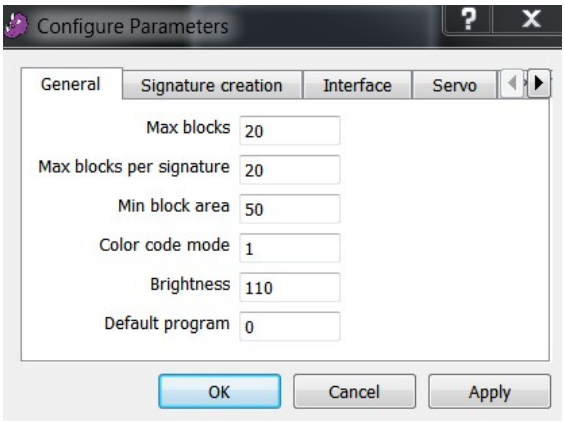


Abb. 7: PixyMon-Einstellungen

Nachdem ihr alle Einstellungen vorgenommen habt, startet ihr das Anlernen von Objektfarben über den PixyMon – das ist einfacher und präziser als über den Knopf an der Pixy-Kamera. Dazu müsst ihr in der Menüleiste oben auf das Bild des „Kochs“ klicken und anschließend, wenn das Objekt im Fenster erscheint, die Linse an der Pixy-Kamera scharf stellen (Abb. 8).

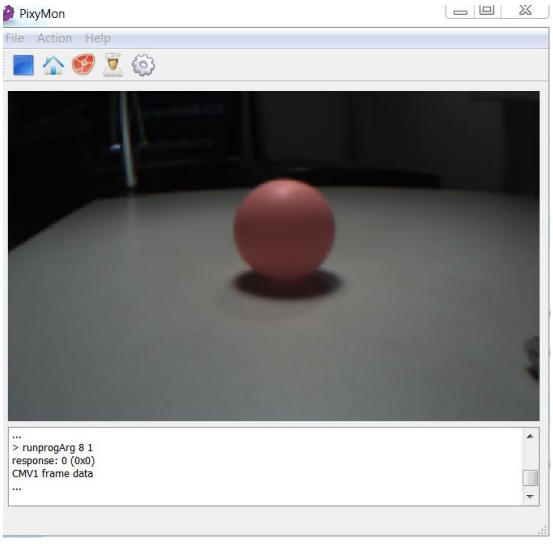


Abb. 8: Objekt vor der Pixy-Kamera

Um eine Objektfarbe anzulernen sind die folgenden Schritte erforderlich:

- Wähle „Action – Set Signature 1“
- Objekt 1 mit der Maus markieren

Jetzt ist die Objektfarbe erlernt. Im Bild wird ein Rahmen um das erkannte Objekt mit dem Text `s=1` angezeigt (Abb. 9).

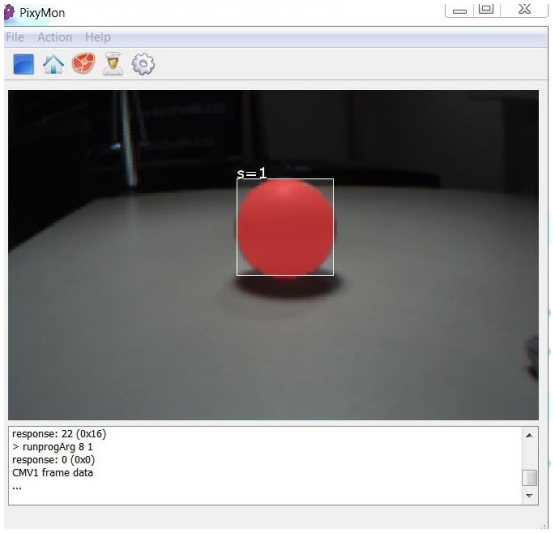


Abb. 9: Objektfarbe 1 (s=1) ist erlernt

Auf dieselbe Weise könnt ihr der Pixy bis zu sieben verschiedene Farb-Signaturen beibringen. Die Pixy-Kamera speichert alle erlernten Farben in ihrem Flash-Speicher; sie gehen daher nicht verloren, wenn ihr die Kamera ausschaltet. Das Löschen von Farbsignaturen erfolgt über das Menu „Action – Clear (all) signature(s)“ in PixyMon.

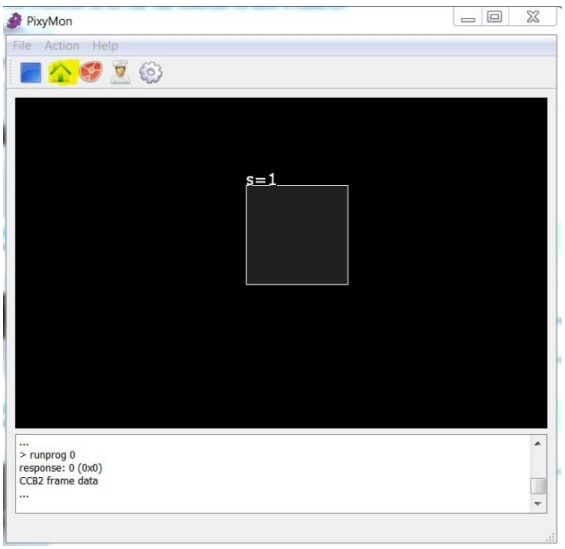


Abb. 10: Objekt-Daten werden über die serielle Schnittstelle übertragen

Sobald ihr im Menü auf den „Home“-Button klickt, werden die Daten der erkannten Objekte in einer der erlernten Farben über den I²C-Anschluss an den Robo TX Controller übertragen (Abb. 10).

Die Pixy zeigt mit einer roten LED unten an der Frontseite der Platine an, dass sie ein Objekt erkannt hat – je näher am Objektiv der Kamera, desto heller leuchtet die LED.

Serielles Daten-Protokoll

Das serielle Protokoll an der I²C-Schnittstelle ist sehr einfach: Die Pixy-Kamera schickt als Slave die Daten aller bei der Bildauswertung identifizierten Objekte nacheinander über den I²C-Bus an den Master. Dabei werden je Objekt die folgenden Daten (Objekt-Datenblock) wortweise im *Little Endian Format* (niederwertiges Byte zuerst) übermittelt. Die Positionsangaben erfolgen in Pixel, bezogen auf eine Bildgröße von 320 x 200:

- *Checksum* (2 Bytes; Summe der Werte der folgenden fünf 16-bit-Worte)
- *Signature* (2 Bytes; Werte 1-7)
- *X-Position* (2 Bytes; Werte 0-319)
- *Y-Position* (2 Bytes; Werte 0-199)
- *Width* (2 Bytes; Werte 1-320)
- *Height* (2 Bytes; Werte 1-200)

Die Übermittlung der Daten eines Objekts wird eingeleitet von zwei *Sync*-Worten (0xaa55). Das ermöglicht eine Byte-Synchronisation beim Empfänger (TX) auf den Beginn eines Wortes und auf den Beginn des Objekt-Datenblocks.

Die identifizierten Objekte werden nach Farbe und Größe geordnet übermittelt (erster Farbcode, dann größtes Objekt zuerst). Sind alle Datenblöcke der Bildauswertung eines Frames komplett übertragen, werden nur noch 0-Bytes geschickt; daran lässt sich der Beginn eines neuen Frames erkennen.

Reicht die Zeit zwischen zwei Frame-Auswertungen (0,02 Sekunden) nicht für die Übermittlung aller Block-Daten der erkannten Objekte, dann beginnt die Pixy mit der Übermittlung der neuen Datenblöcke, wieder beginnend beim größten Block des ersten Farbcodes.

Eine kleine Überschlagsrechnung zeigt, dass es bei sehr vielen Objekten mit der Geschwindigkeit des I²C-Busses am TX eng werden kann:

- Bei 400 kbit/s kann der TX etwa 20.000 16-bit-Worte je Sekunde empfangen, im günstigsten Fall also ca. 2.500 Datenblöcke.[1]
- Bei 50 Frames pro Sekunde können also je Frame bestenfalls 50 Datenblöcke (von 120 möglichen Objekten, die Pixy erkennen kann) übermittelt werden.

Diese Rechnung gilt allerdings nur unter der Annahme, dass der TX keine aufwändigen zusätzlichen Operationen während des Empfangs der Daten vom I²C-Bus durchführt – und auch nur für den Offline-Mode. Da die empfangenen Daten ja noch gespeichert werden müssen, liegt die Zahl der vom TX je Frame empfangenen Datenblöcke in der Praxis deutlich darunter.

Im Online-Mode verlangsamt sich der I²C-Empfang beträchtlich, da die erhaltenen Daten nach jedem Kommando über die USB-Schnittstelle an den PC übertragen werden müssen, bevor das nächste Kommando folgen kann. In unseren Experimenten konnten wir im Online-Mode selten mehr als einen einzigen Block je Frame empfangen und auswerten.

Die Tests zeigten allerdings auch, dass die Übermittlung sehr zuverlässig erfolgt, sodass wir zur Beschleunigung der Übermittlung auf eine Auswertung der Checksumme verzichtet haben.

Der Robo Pro-Treiber

Beim Einlesen der Objekt-Daten der Pixy über die I²C-Schnittstelle ist Geschwindigkeit alles. Daher enthält der Robo Pro-Treiber ein paar Optimierungen:

- Die I²C-Verbindung wird nicht geschlossen, sondern während des Auslesens offen gehalten. Die Synchronisation der von der Pixy empfangenen Byte-Werte auf Wort-Grenzen (`Pixy_ByteSync`) muss daher nur einmal zu Beginn des Programms durchgeführt werden. Anschließend genügt ein `Pixy_BlockSync`.
- Die Funktion `Pixy_GetNextBlock` liefert die Werte des nächsten Datenblocks, ohne die Checksumme zu überprüfen. `Pixy_GetNextBlockChk` liefert die Werte inklusive Checksummen-Prüfung.
- Das Auslesen vieler Blöcke in Folge (z. B. aller Blöcke eines Frames) erledigt `Pixy_ReadDetectedBlocks`: die Funktion schreibt die Daten von bis zu 50 Blöcken in fünf (globale) Listenvariablen (`Signature`, `X`, `Y`, `Width` und `Height`).

Das Hauptprogramm des Treibers zeigt, wie die Datenblöcke aller erkannten Objekte eines Frames in die Listenelemente eingelesen und anschließend nacheinander auf dem TX-Display angezeigt werden können.

Im Online-Mode lässt sich in der Regel nur ein Objekt-Datenblock je Frame einlesen. Um die Daten mehrerer erkannter Objekte pro Frame auszulesen, muss das Programm im Download-Betrieb gestartet werden.

[1] Mit Start-, Start/Stopp- und Stopp-Signal benötigt das I²C-Protokoll (bei Fehler freier Übermittlung) etwa 20 Bits pro Datenwort; ein Objekt-Datenblock besteht (inklusive Sync) aus acht Worten.

Um komplexere Funktionen zu unterstützen, haben wir ein paar Unterfunktionen mit zusätzlicher „Intelligenz“ hinzugefügt:

- Die Funktion `Pixy_GetColor` gibt nur den Farbcode des nächsten gefundenen Objekts zurück. Damit lässt sich ein einzelnes Objekt in einer angelernten Farbe besonders schnell erkennen.
- Die Funktion `Pixy_GetPan` liefert die Richtung, in die die Kamera horizontal geschwenkt werden muss, um den Mittelpunkt eines Objekts in die Mitte des Sichtbereichs der Kamera zu rücken. Der Farbcode des gesuchten Objekts und die tolerierte Abweichung (in Pixel) werden als Parameter übergeben. Die Ausgabe der Funktion kann entweder direkt an einen Motor übergeben oder mit dem RoboPro-Kommando „Warten auf Befehl“ ausgewertet werden.
- Die Funktion `Pixy_GetTilt` gibt die Richtung zurück, in die die Kamera vertikal geneigt werden muss, um den Mittelpunkt eines Objekts in die Mitte des Sichtbereichs der Kamera zu rücken. Der Farbcode des gesuchten Objekts und die tolerierte Abweichung (in Pixel) werden als Parameter übergeben. Auch hier kann die Ausgabe der Funktion entweder direkt an einen Motor übergeben oder mit dem RoboPro-Kommando „Warten auf Befehl“ ausgewertet werden.
- Die Funktion `Pixy_GetDistance` berechnet aus der Größe eines Objekts im Bild (Farbcode und echte Größe werden als Parameter übergeben) den Abstand und gibt diesen als Ergebnis zurück.

Für die Berechnung des Abstands hilft ein wenig Trigonometrie. Abb. 11 zeigt in einer Schemazeichnung den Sichtbereich der Kamera im Schnitt: Der rote Ball ist das erkannte Objekt, dessen Mittelpunktkoordinaten und Maße (Höhe, Breite in Pixeln) von der Pixy zurückgeliefert werden.

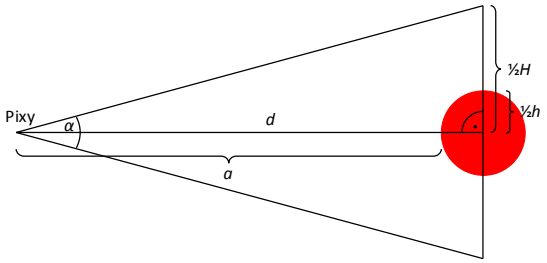


Abb. 11: Schnitt des Kamera-Sichtbereichs

Der Öffnungswinkel α ist bekannt (47°). Daraus lässt sich d wie folgt berechnen:

$$d = \frac{H}{2} \cdot \cot\frac{\alpha}{2}$$

Die Höhe H des Sichtbereichs der Kamera können wir aus der echten Höhe des Objekts und dem Höhenverhältnis von Objekt und Sichtbereich bestimmen:

$$H = h \cdot \frac{H_{Pix}}{h_{Pix}}$$

Dabei ist die Höhe des Bildbereichs H_{Pix} = 200 Pixel, und somit berechnet sich unser d aus der echten Objekthöhe h wie folgt:

$$d = \frac{h \cdot 100}{h_{Pix} \cdot \tan(23{,}5°)}$$

Bei der Bestimmung des Abstands a müssen wir berücksichtigen, dass die Ränder des Objekts (bspw. bei einer Kugel) weiter entfernt sind als der vorderste Punkt. Daher könnt ihr an die Funktion noch die Dicke des Objekts übergeben – bei einer Kugel ist das die Hälfte des Durchmessers:

$$a = d - \frac{h}{2}$$

Die Abstandsberechnung liefert ein ziemlich genaues Ergebnis, wenn die Ränder des Objekts exakt erkannt werden und der Mittelpunkt genau in der Mitte des von der Kamera erfassten Bildbereichs liegt. Die Funktion gibt die Abweichung vom Mittelpunkt in der Horizontalen und Vertikalen (in % des Abstands zum Rand) zurück,

damit ihr feststellen könnt, ob die Kamera korrekt justiert ist.[2]

Den Robo Pro-Treiber der Pixy findet ihr im Download-Bereich der ft:community.

Anwendungsbeispiele

Zum Schluss stellen wir drei Beispielmodelle vor, die ihr mit der Kamera nachbauen könnt.

Pan/Tilt-Modell

Das Pan/Tilt-Modell ersetzt die Dreh- und Neigungs-Plattform, die für die Pixy-Kamera für ca. 35 € angeboten wird und über die beiden Servo-Anschlüsse gesteuert werden kann. Dabei folgt die Kamera einem zuvor angelernten Objekt. Die Steuerung übernehmen zwei Motoren, die die Kamera auf der Plattform drehen und neigen (Abb. 12).

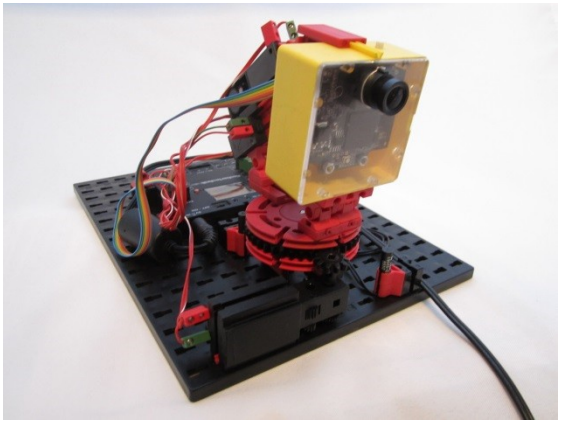


Abb. 12: Pixy I²C Pan/Tilt-Modell

Ein Nachbau ist anhand der Fotos des Modells und mit dem Steuerungsprogramm in der ft:community leicht möglich, und in einem Youtube-Video könnt ihr das Modell in Aktion sehen.

Pixy-Erkundungsroboter

Einen mit der Pixy-Kamera ausgestatteten Erkundungsroboter könnt ihr dazu bringen, einem farbigen Ball zu folgen. So lässt sich das Basismodell aus dem Baukasten ‚Robo TX Training Lab' über die beiden Encoder-Motoren steuern, indem die Links-Rechts-Abweichung aus der X-Koordinate des Objekt-Mittelpunkts abgeleitet wird (Abb. 13). Die Abstandsberechnung ist etwas komplizierter (siehe oben).

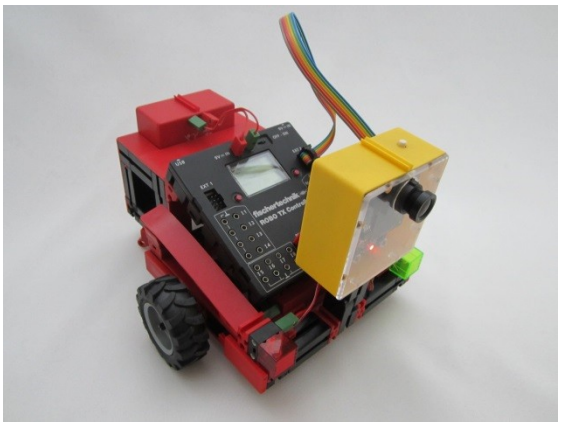


Abb. 13: Erkundungsroboter mit Pixy

Auch hier findet ihr das Modell und die RoboPro-Software zum Download in der ftcommunity, sowie ein Video des Erkundungsroboters auf Youtube.

Pixy-Farberkennung

Die Farberkennung der Pixy funktioniert auch bei Störlicht sehr zuverlässig – und ist damit erheblich besser als der fischertechnik-Farbsensor (128599).

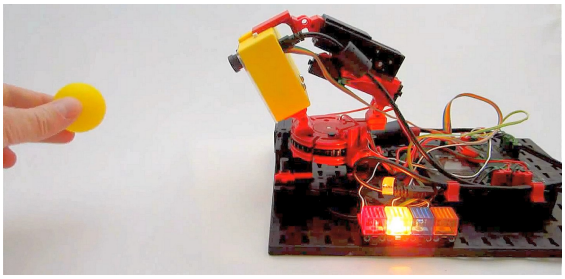


Abb. 14: Farberkennung mit der Pixy

Das dritte kleine Beispielmodell zeigt die Farbe des erkannten Objekts mit einem

[2] Richtig, man kann die Abweichung von der Mittelachse in der Abstandsberechnung berücksichtigen. Um die eine Hälfte von euch nicht mit noch mehr Mathematik zu schocken, überlassen wir das der anderen Hälfte als Fingerübung.

Lämpchen an (Abb. 14). Auch von diesem Modell gibt es ein Youtube-Video.

Fortsetzung folgt…

Eine Funktion der Pixy haben wir Euch vorenthalten: die Erkennung von *Color Codes*. Was es damit auf sich hat und welche Modelle sich mit *Color Codes* realisieren lassen, stellen wir im zweiten Teil dieses Beitrags in der nächsten ft:pedia vor.

Quellen

[1] Charmed Labs: *CMUcam5 Pixy (Wiki)*.

[2] Kickstarter: *Pixy (CMUcam5): a fast, easy-to-use vision sensor*.

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

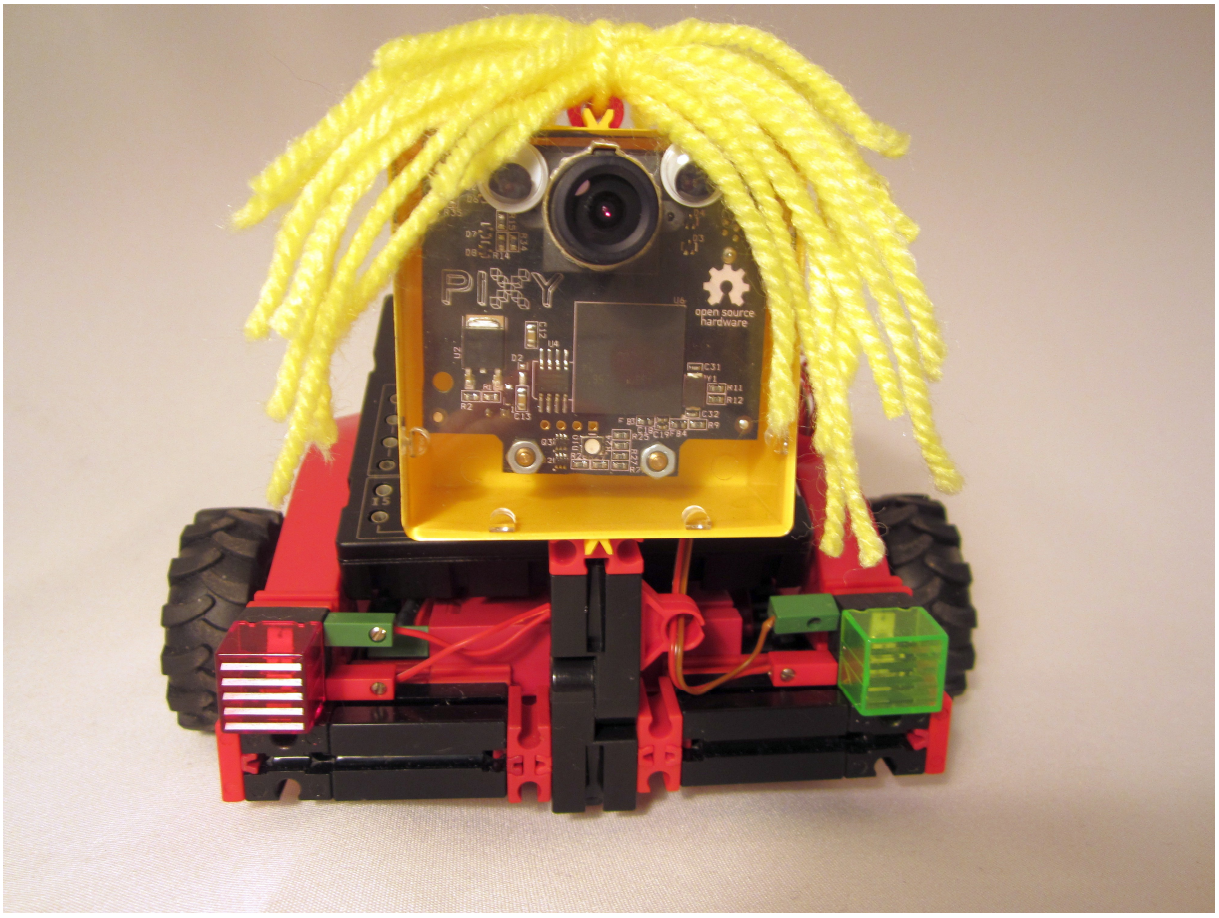


Abb. 15: Pixy-Robot

Computing

Ziffernerkennung über eine CMOS-Kamera am AVR-Controller

Dirk Uffmann

Kameras und Bildverarbeitung in Modellsteuerungen werden immer beliebter. Von fischertechnik gibt es mittlerweile auch eine Kamera für den TXT-Controller. Im vorausgegangenen Beitrag wurde vorgestellt, wie sich eine Pixy CMUcam5 über das I²C-Interface am TX nutzen lässt – und in diesem Beitrag stelle ich euch eine weitere, kostengünstigere Möglichkeit vor: mit einem Arduino-Mega2560-Board und einem Kameramodul lassen sich sogar Ziffern identifizieren, die von einer Vorlage abgelesen werden.

Hintergrund

Ich habe schon lange davon geträumt, in meine Modellsteuerungen eine Kamera zu integrieren und ein bisschen mit Bildverarbeitung zu experimentieren. Bei Internetrecherchen findet man mittlerweile recht häufig Anwendungen zum Nachverfolgen von Farbklecksen (*Color Blob Tracking*). Meistens werden dafür 32-bit-Controller auf ARM-Basis verwendet, was schon einiges an Programmier- und Hardwarekenntnissen erfordert. Es sei denn, man kauft einfach ein fertiges Kameramodul mit bereits programmiertem Controller. Das letztere wollte ich nicht und so habe ich nach einem einfacheren Weg mit AVR-Controllern gesucht, da ich diese bereits kenne. Die Performance ist selbst für einfache Bildverarbeitung knapp bemessen, daher muss man geschickt mit deren Ressourcen umgehen.

Komponenten

Ich begann mit der Suche nach günstigen Komponenten und bin u. a. bei Aliexpress aus China fündig geworden: Ein Ardino-Mega-2560R3-Board für ca. 11 €, ein passendes LCD-2408(V1.1)-Farbdisplay mit Display-Controller ILI9325D in der Größe 2,4 Zoll für ca. 7 €, ein TFT-Shield Lseeduino V1.1 mit Level-Shiftern von 5V auf 3,3V für ca. 7 € und ein Alientek Kameramodul V2.2 mit einer Omnivision OV7670 CMOS-VGA-Kamera inkl. Averlogic FIFO AL422B (Abb. 1) zur Zwischenspeicherung der Bilder für ca. 6 € (bei Abnahme von fünf Stück).

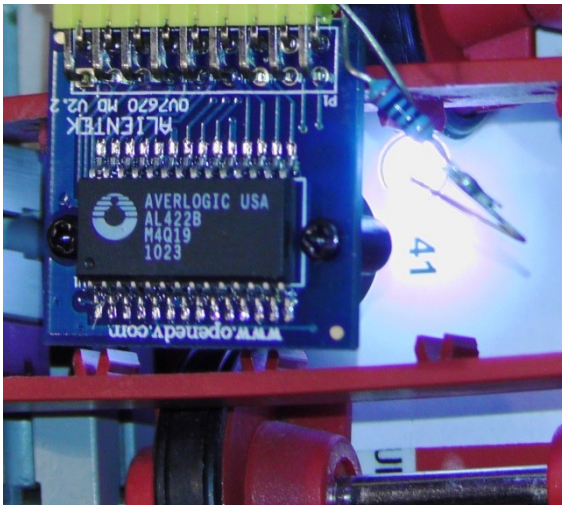


Abb. 1: Kameramodul mit gebastelter LED-Beleuchtung und Objekt (Ziffer 41)

Zum Debuggen nutzte ich noch ein Text-LCD für ca. 5 € am I²C-Bus. Die Beschaffung war versandkostenfrei, die Lieferzeit betrug ca. 3-6 Wochen (zollfrei bis 22 € je Bestellung). Gesamtkosten: ca. 36 €.

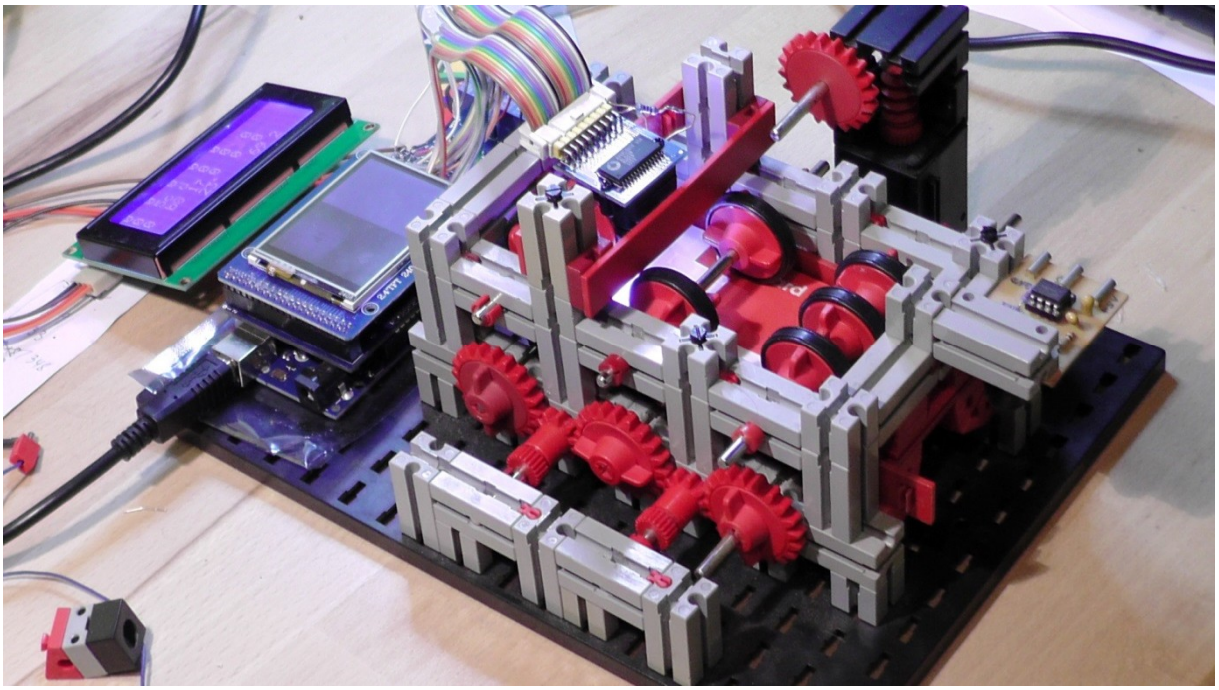


Abb. 2: Modell des Kartenlesers mit Arduino-Mega-Board, TFT-Shield mit angelöteten Kameraleitungen und Display

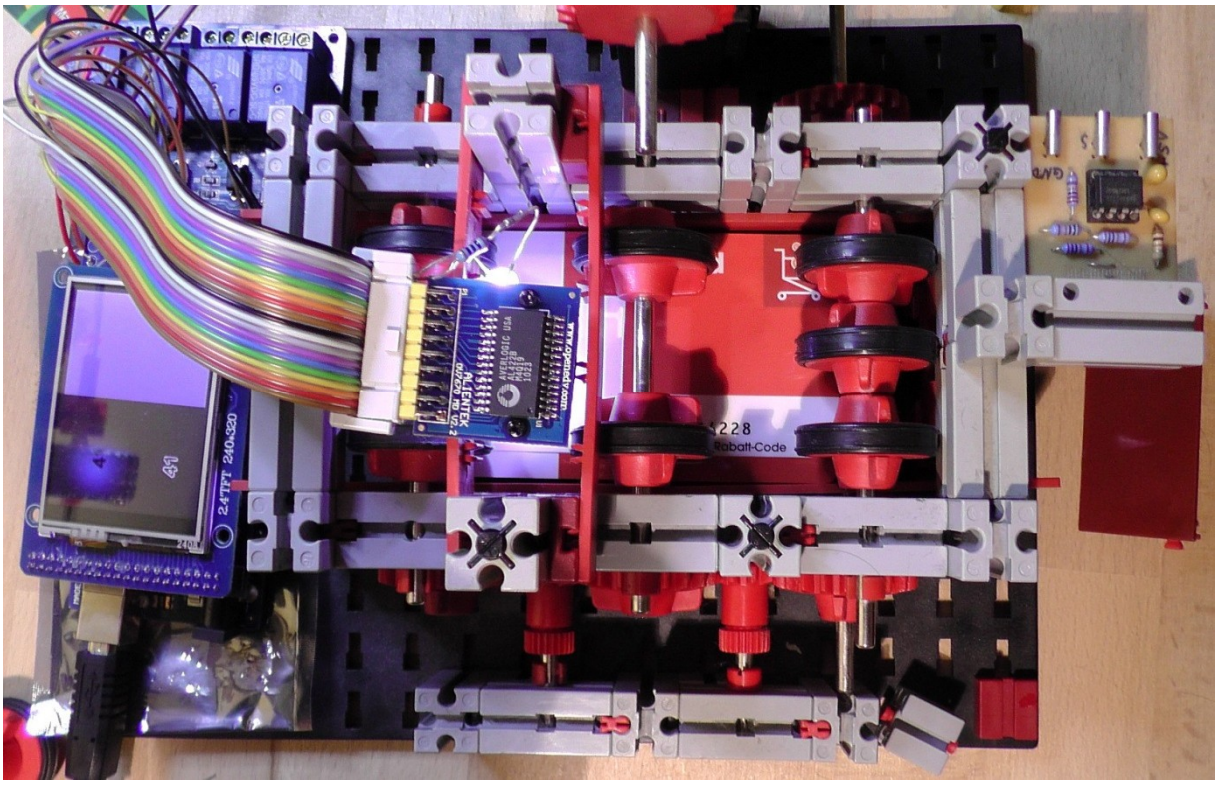


Abb. 3: Modell des Kartenlesers von oben

Hardware-Konzept

Elektrische Anschlüsse und Pin-Zuordnung

Für eine direkte Darstellung von Kamera-Bildern auf dem Display sollten Kamera und LCD im RGB565-Mode betrieben werden und mit den acht Datenleitungen gemeinsam an einen Bus (Port A) des ATMEGA-2560 angeschlossen werden [1]. Dadurch kann das Kamerabild direkt und schnell auf das LCD-Display gebracht werden, ohne es in den Mikrocontroller einlesen zu müssen. Dieses Bild ist dann als Referenz für die weiteren, mit Bildverarbeitung errechneten Bilder sichtbar.

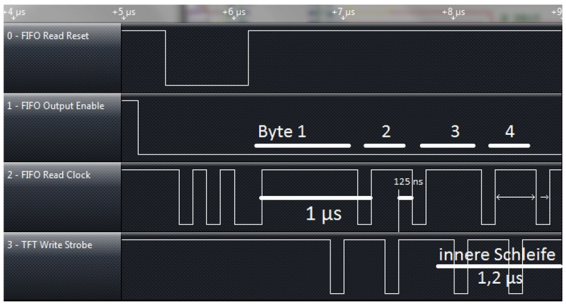


Abb. 4: Timing-Diagramm von FIFO Read Clock und TFT Write Strobe zur direkten Übertragung des RGB565-Bildes ans Display

Der LCD-Controller ILI9325D des Farb-TFT-Displays wird im 8-bit MPU-Modus betrieben (zwei Bytes pro Pixel für RGB565). Die dafür benutzen Datenleitungen sind [D17:D10], die auf [DB15:DB08] am TFT-Shield gelegt sind. Die Datenleitungen [DB15:DB08] des TFT-Shields sind über Level-Shifter (Übergang von 5 V auf 3,3 V) an die Arduino-Pins [D29:D22] am Port A angeschlossen. Hier müssen dann parallel auch die Signale [D7:D0] der Kamera angeschlossen werden (angelötet an den Pins am 36-poligen Verbindungssteckers des TFT-Shields). Dann kann das Kamerabild über die Steuersignale direkt ins Grafik-RAM des LCD-Controllers übertragen werden (Abb. 4).

Über das Signal OE (*active low*) wird der Datenausgang des FIFO auf den Bus geschaltet. Mit OE=*high* ist der FIFO-Datenausgang hochohmig. Das LCD treibt nie auf die Datenleitungen, da dessen Signal RD im Shield auf *high* geklemmt ist. Die Spannungsversorgung der Kamera wird an den Ausgang des 3,3 V-Reglers auf dem TFT-Shield und GND des Arduino-Boards angeschlossen (angelötet am GND-Pin des sechspoligen Versorgungssteckers J24 des TFT-Shields).

Der SCCB-Bus der Kamera wird über einen Software-I²C-Bus mit 3,3 V betrieben (der TWI mit den 10k Pull-up-Widerständen an 5 V ist dafür ungeeignet, da damit die Kamera beschädigt würde, die nur 3,3 V an ihren Pins verträgt). Mit der Library von Peter Fleury, die auf direkten Bit-Zugriffen in den I/O-Registern mit CBI- und SBI-Befehlen basiert, kann nur der IO-Space bis 0x1F adressiert werden. Daher stehen dafür nur die Ports bis einschließlich Port G zur Verfügung. Genutzt werden die Analog-Pins A1 (für SDA) und A0 (für SCL) des Arduino-Boards, entsprechend Port F Pin 1 und 0 des ATMEGA2560. Beide Pins benötigen noch einen Pull-Up-Widerstand von 5 kOhm auf 3,3 V. Dazu wird eine einreihige, achtpolige Steckerleiste ins TFT-Shield eingelötet (J22) und an den Pins A0 und A1 an die SCCB-Kamera-Leitungen und die Pull-up-Widerstände angelötet (4fach Widerstandsnetzwerk mit 4 x 10 kOhm, je zwei parallel geschaltet).

Damit der FIFO zum Schreiben der Bilddaten immer am Bildanfang zurückgesetzt wird, wird das Signal *Write Reset* (WRST, *active low*) immer zeitgleich mit einem VSYNC-Puls ausgelöst. VSYNC kann über ein Register so konfiguriert werden (invertiert), dass es von *high* auf *low* auf *high* wechselt (*active low*). Dann kann VSYNC direkt über eine Verbindungsleitung den *Write Reset* WRST des FIFO ansteuern.

Die Verbindung der beiden FIFO-Pins wird am Arduino-Pin 19 hergestellt. VSYNC wird außerdem im ATMEGA2560 benötigt, um andere Signale wie das FIFO *Write*

Enable (WE) per Software zu steuern. Dies soll über Interrupts erfolgen. Hierfür wird der Interrupt INT2 auf Arduino Pin 19 = Port D Pin 2 am ATMEGA 2560 gewählt.

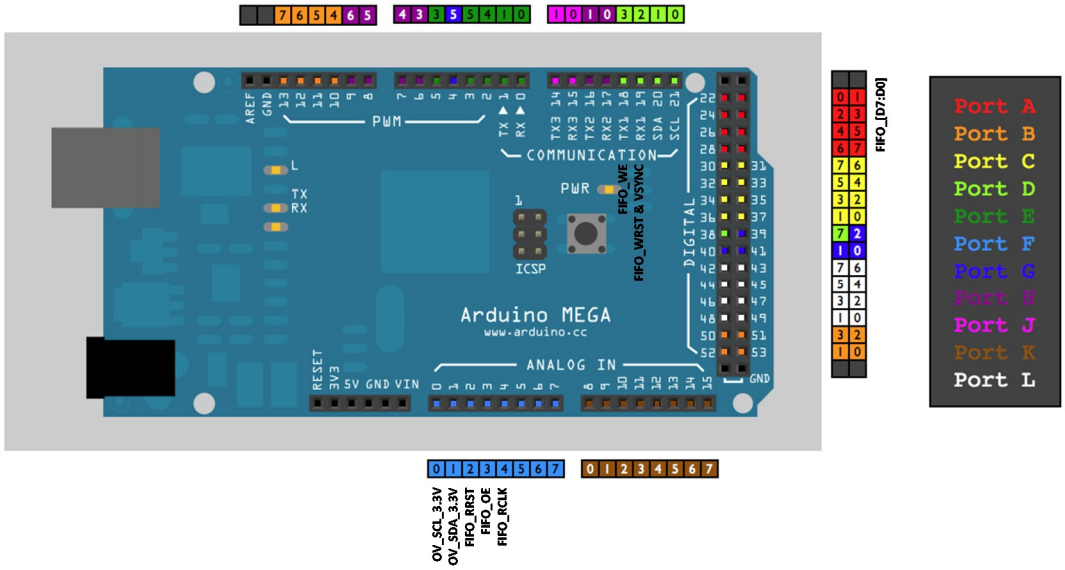


Abb. 5: Elektrische Anschlüsse bzw. Pin-Zuordnung zwischen den Modulen

Alientek OV7670	Arduino	ATMEGA2560	Bemerkungen
01: GND	GND	GND	
02: VCC3.3V	n.c	n.c.	3,3 V Regler auf dem TFT-Shield
03: OV_SCL	A0	Port F0	5k Pullup an 3V3 (Regler auf TFT-Shield)
04: FIFO_WRST 18: OV_VSYNC	19: RX1	Port D2	Beide Kabel auf denselben Pin am Arduino-Board löten, VSYNC in der Konfiguration invertieren
05: OV_SDA	A1	Port F1	5k Pullup an 3V3 (Regler auf TFT-Shield)
06: FIFO_RRST	A2	Port F2	FIFO *Read Reset*
07: FIFO_D0	22	Pin A0	
08: FIFO_OE	A3	Port F3	FIFO *Output Enable* (*active low*)
09: FIFO_D2	24	Pin A2	
10: FIFO_D1	23	Pin A1	
11: FIFO_D4	26	Pin A4	
12: FIFO_D3	25	Pin A3	
13: FIFO_D6	28	Pin A6	
14: FIFO_D5	27	Pin A5	
15: FIFO_RCLK	11	Port F4	die FIFO *Read Clock* treibt mit positiver Flanke neue Daten aus dem FIFO auf den Datenbus raus, diese werden dann mit der positiven Flanke des TFT *Write Strobe* zwei Takte später übernommen.
16: FIFO_D7	29	Pin A7	
17: FIFO_WEN	18: TX1	Port D3	FIFO *Write Enable* (*active low*)
TFT 2.4“	Arduino	ATMEGA2560	Bemerkungen
	38	Port D7	
WS: *Write Strobe*	39	Port G2	
	40	Port G1	
	41	Port G0	

Tab. 1: Elektrische Anschlüsse bzw. Pin-Zuordnung zwischen den Modulen

Durch Modifikation einer Status-Variablen „image_state“ im Hauptprogramm und im Interrupt wird gesteuert, welcher Code jeweils ausgeführt wird, also z. B. ob ein Bild aufgezeichnet werden soll (0), ob gerade ein Bild aufgezeichnet wird (1) oder ob ein vollständiges Bild zur weiteren Verarbeitung im FIFO steht (2), siehe auch [3].

```
ISR (INT2_vect) {
if (image_state == 0) {
// start to store image to FIFO, write enable
   PORTD |= _BV(PD3);
// at next VSync stop image saving
   image_state++; }
else if (image_state == 1) {
// stop to store image, write disable
   PORTD &= ~(_BV(PD3));
// image is ready for reading from FIFO
   image_state++; }
}
```

Tab. 2: Interrupt-Service-Routine, über die die Bildaufnahme von der Kamera gesteuert wird

Taktschema

Die *Read Clock* RCK des FIFO muss im AVR generiert werden. Hierfür wird Port F-Pin 4 = PF4 (Arduino Pin A4) verwendet.

Das LCD-Display wird am Arduino-Pin 39 = Port G Pin 2 mit seiner *Write Clock* versorgt (TFT-Shield). Wichtig ist, dass beide Takte synchron laufen und dass das TFT *Write Strobe* mit seiner positiven Clock-Flanke mindestens 10 ns der positiven FIFO Read-Clock-Flanke hinterherläuft (Setup-Zeit) und die *high*-Phase des FIFO *Read Clocks* dann noch mindestens 15 ns anhält (Hold-Zeit).

Die Bilddaten werden also mit zwei synchronen Clock-Signalen aus dem FIFO der Kamera ausgelesen und ins LCD geschrieben. Damit ergibt sich die in Abb. 5 und Tab. 1 dargestellte Pin-Zuordnung. Um die nötigen Kabel aufzulöten werden zwei einreihige, achtpolige Steckerleisten ins TFT-Shield eingelötet (J21 für Pins 14-21 und J22 für Pins A0-A7).

Bilddaten und Timing

Der 12 MHz-Schwingquarz im Kameramodul wird über die PLL auf 48 MHz vervierfacht und dann über den *Clock Prescaler* wieder auf 24 MHz halbiert. Die Pixelclock am Bildsensor läuft dann mit 24 MHz. Damit ergeben sich folgende Werte für das Frame-Timing:

- tP = 2 · tPCLK (RGB-Format mit 2 Bytes / Pixel) = 83 ns (Zeit für ein Pixel = 2 Bytes)
- tLINE = 784 · tP = 65 µs (Zeit für die Übertragung einer Bildzeile)
- tFRAME = 510 · tLINE = 33 ms → 30 Frames / Seunde

Das gewählte Bildformat ist QQVGA mit 160 x 120 Pixeln (also Teilung des VGA-Bildes in beiden Dimensionen durch vier). Ein Bild ist also 2 x 160 x 120 Bytes = 38.400 Bytes groß. Dafür reicht der Speicherplatz im internen RAM des AVR bei weitem nicht aus. Da der FIFO außerdem mit einem Takt von mindestens 1 MHz ausgelesen werden muss, muss die erste Erfassung und Auswertung bzw. Verdichtung der Bilddaten ins RAM des AVR möglichst rasch pixelweise erfolgen.

Je Byte werden für das Einlesen der Daten ins RAM des AVR mindestens vier Takte à 16 MHz benötigt (Einlesen eines Ports in ein Register = ein Takt, Speichern des Registers im RAM = zwei Takte, Schleifensprung = ein Takt). In diesem Fall kann das Auslesen der Daten aus dem FIFO also mit maximal 4 MHz erfolgen. Für weitere Auswertungen oder Umrechnungen des Bytes stehen maximal 12 weitere Takte zur Verfügung, um die Leserate aus dem FIFO nicht unter 1 MHz sinken zu lassen.

Beleuchtung

Für die Bildverarbeitung ist eine möglichst gleichmäßige Ausleuchtung des Objektes und dessen „Hintergrunds“ wichtig. In meinem Modell habe ich mit einer einfachen LED gearbeitet, brauchte zur Erkennung der Ziffern dann aber einen Laserausdruck der Ziffern auf mattem Papier anstelle der auf den hoch reflektierenden Kreditkarten geprägten silbern glänzenden Ziffern. Besser wäre eine ringförmige LED-Beleuchtung, die das Blickfeld der Kamera unter flachem Winkel ausleuchtet.

Software-Konzept

Die Software wurde mit Atmel-Studio in C geschrieben und mit XLoader auf den AVR geladen (damit umgehe ich die Arduino-

Tools). Einstellungen für die Initialisierung der Kamera und des TFT-Displays habe ich mir im Internet zusammengesucht [2].

Bildformat

Für eine Ziffern-Erkennung kann mit der Helligkeit gearbeitet werden. Dafür bietet sich das YUV-Bildformat an (4:2:2), bei dem das erste der beiden Bytes/Pixel jeweils die Helligkeit repräsentiert. Nachdem das Referenzbild im RGB-Format aufgenommen und direkt auf dem Display dargestellt wurde, wurden die Kamera-Einstellungen auf das YUV (4:2:2)-Format geändert und ein weiteres Bild in den FIFO eingelesen, welches dann pixelweise vom AVR verarbeitet wird (Abb. 6).

Schwellwertfilter

Der Helligkeitswert des Pixels wird mit einem ermittelten Schwellwert verglichen und bei Unterschreitung werden die Pixel mit ihren Pixelkoordinaten in einer Liste gespeichert, die aus den Feldern x[i], y[i] und *region*[i] besteht. Zunächst werden nur die x und y-Koordinaten jedes herausgefilterten Pixels gespeichert und der Index hochgezählt. Die so gesammelten Pixel werden sodann in einer reinen Schwarz-Weiß-Darstellung (keine Grauwerte) auf dem Display dargestellt (Abb. 8).

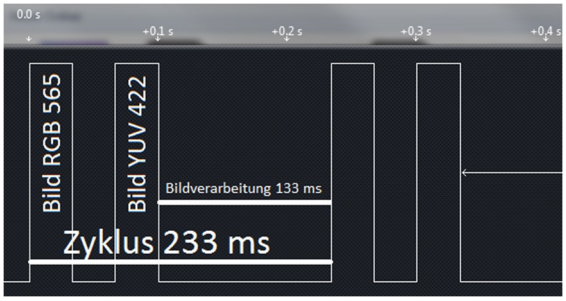


Abb. 6: Timing-Diagramm des gesamten Zyklus' mit Aufnahme und Darstellung von zwei QQVGA-Bildern auf dem Display und der Berechnungen zur Identifikation von zwei Ziffern. Damit werden ca. vier Zyklen pro Sekunde erreicht.

Einfaches Kantenfilter

Eine Bildzeile wird dabei im RAM des AVR gespeichert und laufend nach Auswertung des aktuellen, zwischengespeicherten Pixels mit diesem überschrieben.

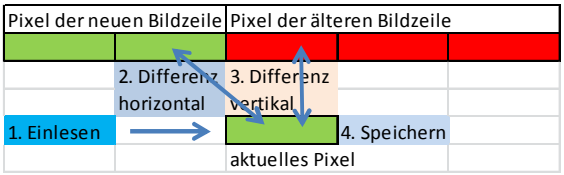


Abb. 7: Ablauf der pixelweisen Bildauswertung beim einfachen Kantenfilter

Dies ermöglicht ein simples Kantenfilter in beiden Bilddimensionen. Über einen Vergleich des temporär in eine Variable eingelesenen Pixels mit dem direkt zuvor eingelesenen Pixel wird die Änderung der Helligkeit in horizontaler Richtung ermittelt, ebenso durch einen Vergleich mit dem positionsgleichen Pixel der vorherigen Bildzeile die Änderung der Helligkeit in vertikaler Richtung.

Übersteigt einer dieser beiden Änderungswerte eine definierte Schwelle, wird dieses Pixel in einer Liste gespeichert, die aus den Feldern x[i], y[i] und *region*[i] besteht. Zunächst werden nur die x- und y-Koordinaten jedes herausgefilterten Pixels gespeichert und der Index hochgezählt. Die so gesammelten Pixel werden sodann in einer reinen Schwarz-Weiß-Darstellung (keine Grauwerte) auf dem Display dargestellt (Abb. 9).

Für die weitere Erkennung von Ziffern habe ich mich jedoch für das Schwellwertfilter anstelle des Kantenfilters entschieden, da es ein weniger komplexes Bild der Ziffer liefert. Der Schwellwertfilter reagiert allerdings anfälliger auf eine ungleichmäßige Ausleuchtung des Objektes.

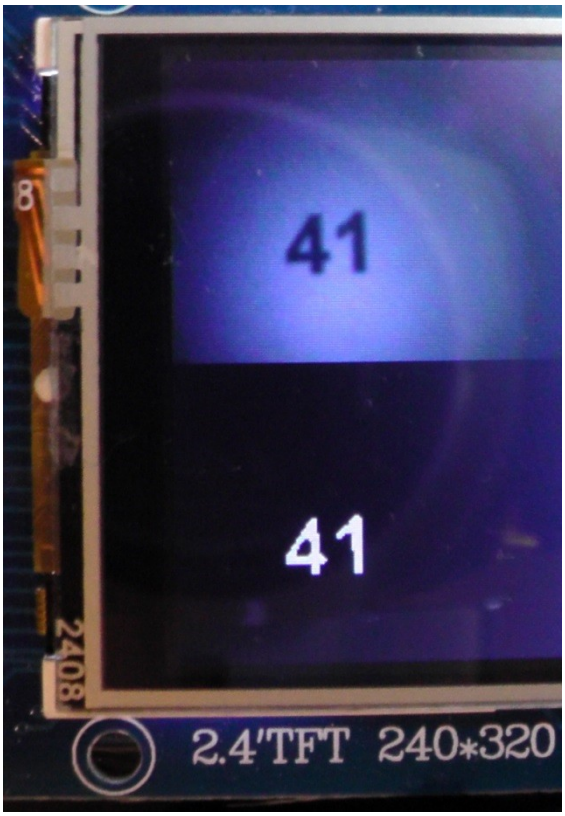


Abb. 8: Ansicht des Originalbildes (oben) und des Bildes nach dem Schwellwertfilter (unten)

Abb. 9: Ansicht des Originalbildes (oben) und des Bildes *nach dem Kantenfilter (unten)*

Segmentierung

Die herausgefilterten Pixel müssen nun den Ziffern zugeordnet werden. Dazu wird die Pixel-Liste bearbeitet. Ich habe mich für folgende Methode entschieden: In einem ersten Durchlauf werden zusammenhängende Pixel in horizontaler Richtung mit einem übereinstimmenden *region*-Wert gekennzeichnet und dieser danach jeweils hochgezählt. Im zweiten Durchlauf werden beginnend mit der ersten Zeile alle Pixel in der nächst höheren Zeile, die in benachbarten oder nahen x-Bereichen zur vorherigen Zeile liegen, dem entsprechenden *region*-Wert der vorherigen Zeile zugeordnet.

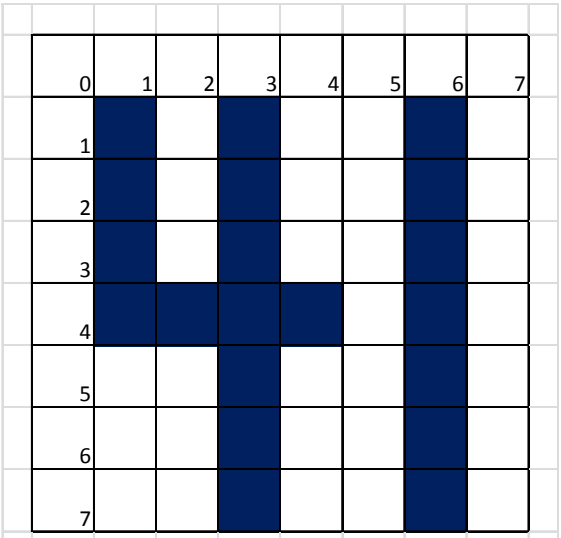
```
i = 2;
label = 1;
region [1] = label;

// 1. Durchlauf zur Segmentierung
do {
   label++;
   if (x[i] == x[i-1] + 1) {
      if (y[i] == y[i-1]) {
         label--;  }
   }
   region[i] = label;
   i++;
} while (i <= filtered_pixel_counter);

// 2. Durchlauf zur Segmentierung
i = 1;
do {
   j=i+1;
   do {
      if ((x[j]>x[i]-5)&&(x[j]< x[i]+5)) {
         region[j] = region[i]; }
      j++;
   } while (j<=filtered_pixel_counter);
   i++;
} while (i<filtered_pixel_counter);
```

Tab. 3: Code-Beispiel zum Segmentieren in zwei Durchläufen

So werden zusammenhängende Bereiche gefunden, die eine übereinstimmende Lage in vertikaler Richtung haben. Diese werden dann demselben *region*-Wert zugeordnet. Das Ergebnis ist eine Liste, in der alle Pixel, die derselben *region* zugeordnet sind, genau eine der Ziffern im Bild repräsentieren (Abb. 10).



			Nach Durchlauf	
			1.	2.
i	x[i]	y[i]	region[i]	region[i]
0	0	0	0	0
1	1	1	1	1
2	3	1	2	1
3	6	1	3	2
4	1	2	4	1
5	3	2	5	1
6	6	2	6	2
7	1	3	7	1
8	3	3	8	1
9	6	3	9	2
10	1	4	10	1
11	2	4	10	1
12	3	4	10	1
13	4	4	10	1
14	6	4	11	2
15	3	5	12	1
16	6	5	13	2
17	3	6	14	1
18	6	6	15	2
19	3	7	16	1
20	6	7	17	2

Abb. 10: Beispiel der Segmentierung an zwei Ziffern

Ziffern-Erkennung

Um nun die Ziffern zu unterscheiden ermittelt man charakteristische Merkmale wie Pixelanzahl oder Schwerpunkt etc. Die Pixelzahl ist allerdings abhängig von der Beleuchtung und der geometrische Schwerpunkt der Zahlen unterscheidet sich nicht so stark. Ich fand folgendes Charakteristikum interessant und habe es angewandt: es wird für jede Ziffer ermittelt, wie viele Übergänge von schwarz nach weiß in allen Bildzeilen und wie viele solcher Übergänge in allen Bildspalten vorhanden sind [4, 5]. Damit sind die Ziffern voneinander unterscheidbar und können zugeordnet werden. In dem simplen Beispiel in Abb. 10 hat die 4 in horizontaler Richtung in Summe zehn Übergänge von schwarz nach weiß und in vertikaler Richtung vier, die 1 hingegen sieben in horizontaler und einen in vertikaler Richtung.

Mit der reinen Summenbildung dieser Übergänge stößt man bei Unterscheidung der Ziffern 6 und 9 allerdings auf ein Problem. Hier wird wegen der Symmetrie der beiden Ziffern zusätzlich eine räumliche Information benötigt. Ich habe daher die Summenbildung aufgeteilt: für die Zeilensummen nach oberer und unterer Hälfte und für die Spaltensummen nach linker und rechter Hälfte der Ziffer.

```
i=filtered_pixel_counter;
do {
   j = region [i];
   if (x[i] < center_x[j]) {
      transition_counter_x_left[j]++;
      if (x[i]==x[i-1]+1) {
         if (y[i]==y[i-1]) {
            transition_counter_x_left[j]--;}
      }
   }
   else
   {
      transition_counter_x_right[j]++;
      if (x[i]==x[i-1]+1) {
         if (y[i]==y[i-1]) {
            transition_counter_x_right[j]--;}
      }
   }
   i--;
} while (i);
```

Tab. 4: Code-Beispiel zum Zählen der Übergänge in Zeilenrichtung mit Unterscheidung nach oberer und unterer Hälfte

So ergibt sich ein gut unterscheidbarer „Fingerabdruck“ für jede Ziffer. Diese „Fingerabdrücke“ oder Datensätze werden einmal ermittelt, im Controller gespeichert und dann mit den jeweils beobachteten Objekten verglichen. Dazu lasse ich je eine Fehlerfunktion für den Vergleich der ermittelten Objektdaten mit den für alle Ziffern

gespeicherten Datensätze (jeweils vier Bytes, siehe Tab. 5) berechnen, indem ich den absoluten Betrag der Differenzen für jedes der vier Bytes aufsummiere.

Der Vergleich, der den geringsten Fehler aufweist, ergibt die Identifikation der Ziffer. Mit den von mir verwendeten Zeichen und justierter Beleuchtung hat das funktioniert.

```
const uint8_t cypher[10][4] PROGMEM = {
// left, right, top,  bottom
   {0x17, 0x14, 0x0B, 0x09},//cypher 0
   {0x03, 0x16, 0x06, 0x01},//cypher 1
   {0x0d, 0x0e, 0x0e, 0x07},//cypher 2
   {0x0c, 0x12, 0x0f, 0x0d},//cypher 3
   {0x0a, 0x14, 0x06, 0x0e},//cypher 4
   {0x14, 0x0b, 0x13, 0x0c},//cypher 5
   {0x17, 0x0e, 0x13, 0x0e},//cypher 6
   {0x0a, 0x0d, 0x11, 0x04},//cypher 7
   {0x16, 0x13, 0x10, 0x10},//cypher 8
   {0x15, 0x12, 0x0b, 0x16} //cypher 9
};
```

Tab. 5: Ermittelte Datensätze („Fingerprints“) für die zehn Ziffern 0 bis 9.

Wenn man den Zeichensatz ändert, muss man die Vergleichsdatensätze möglicherweise neu ermitteln. Wenn die Größe der Ziffern variabel ist, muss man noch eine Funktion zum Skalieren der erkannten Objekte auf eine einheitliche Größe implementieren.

Zuletzt werden die erkannten Ziffern noch anhand ihrer x-Koordinaten in die passende Reihenfolge von links nach rechts sortiert, so wie sie abgelesen wurden [6].

```
i = 1;
do {
   j = i + 1;
   do {
      if (min_x[i] > min_x[j]) {
         fig = identified_fig [i];
         identified_fig[i]=identified_fig[j];
         identified_fig [j] = fig;
         label = min_x [i];
         min_x [i] = min_x [j];
         min_x [j] = label;
      }
      j++;
   } while (j <= region_max);
   i++;
} while (i < region_max);
```

Tab. 6: Code-Beispiel zum Sortieren der erkannten Ziffern. Die x-Position jeder Ziffer wird mit allen höher indizierten verglichen. Wird eine Ziffer mit niedrigerem x gefunden, so werden die beiden Ziffern vertauscht

Der erstellte Code steht zum Download zur Verfügung. Derzeit werden nicht mehr als zwei Ziffern erkannt. Grund dafür ist die Begrenzung auf 255 Pixel (Felder x[i], y[i]), da andernfalls mit einem 16-bit-Index die Schleife für das Lesen aus dem FIFO zu langsam wird.

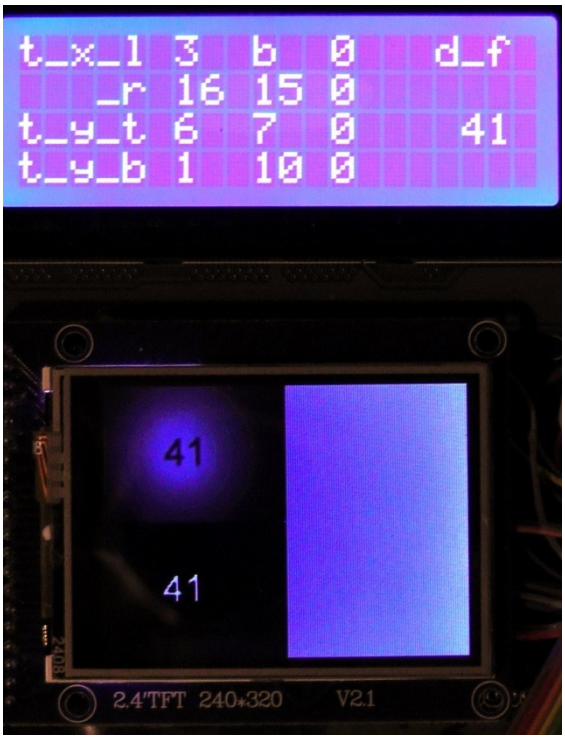


Abb. 11: Die beiden Displays mit der durch das System korrekt erkannten Ziffernfolge 41. Das Text-LCD zeigt auch die für beide Ziffern ermittelten Übergänge an, links für die 1, rechts für die 4

Fazit

Auch mit vergleichsweise einfachen Mitteln ist es möglich, mit Kameras und Bildverarbeitung zu spielen und diese in fischertechnik-Modellsteuerungen zu verwenden. In [7] ist eine weitere Idee zu finden: ein mobiler Roboter, der Türschilder lesen kann.

Quellen

[1] Jenssen, Arndt: *OV7670 + FIFO Camera Control with an AVR ATMEGA-1284P*, GitHub.

[2] ComputerNerd: *Arduino Mega 2560 Code which uses either an ov7670 or an MT9D111 to display an image on a tft screen*, GitHub.

[3] C, Joe: *Einstieg: Mikrocontroller STM32F103/Kameraboard*.

[4] Zimmermann, Lipfert: *Applet zur Zahlen-Erkennung auf der Basis einer Kohonen-Feature-Map*. Fachhochschule Regensburg.

[5] Schedl, Christian; Zachmayer, Andreas: *Postleitzahlenerkennung mit einer Kohonen Feature Map*. Fachhochschule Regensburg.

[6] Meinelt: *Sortieralgorithmen*.

[7] Nast-Kolb, Jens: *Angewandte Texterkennung für mobilen Roboter*. Studienarbeit TU München.

Explorer aus dem fischertechnik Robotics TXT Discovery Set (Bild: fischertechnik GmbH)