embedded projects JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

[PROJECTS]

ARTIKEL-WETTBEWERB: GHOSTCAR-PROJECT

BOARDING

GNUPI

XML & XSLT

AUSGANGSSPANNUNG OHNE VERLUSTE GEFILTERT

„ARDUINO-CAR“

Eine Open-Source Zeitschrift
zum Mitmachen!

Einleitung

Ausgabe 02/2013

Embedded Projects Journal - Ausgabe No. 17

Einleitung

Voilà, Journal Nr. 17.

Es ist wieder eine nette Sammlung an Artikeln geworden. Das Journal startet diesmal gleich zu Beginn mit dem Siegerartikel 2013 vom Mikrocontroller Artikelwettbewerb, der bereits im Journal Nr. 15 mit einem Aufruf an alle Bastler angekündigt wurde [1] [2].

In dieser Ausgabe findet man desweiteren einen Artikel über ein Thema, bei dem viele noch auf der Suche sind. Wie kann man besser geplant in eine Soft- oder Firmware-Entwicklung starten bzw. was für Tools gibt es als Hilfsmittel. Vielen Dank an Klaus für so eine Vorlage.

In den letzten Monaten sind wir außerdem viel herumgereist und waren so auf den LinuxTagen in Chemnitz, Augsburg, Berlin, Graz, Hackerbrücke München. Einige werden noch folgen :) . Wir freuen uns viele nette Leute kennengelernt zu haben. Wer sehen will, was bei uns so alles passiert, kann Ankündigungen oder Berichte zukünftig in unserem neuen Blog[3] verfolgen.

Wir freuen uns ebenfalls gemeinsam mit Elektor eine Kooperation abgeschlossen zu haben. Zusammen möchten wir mehr Menschen innerhalb und außerhalb Deutschlands für die Elektronik begeistern (siehe Marktplatz S. 28).

Benedikt Sauter

und das embedded projects Team

[1] http://journal.embedded-projects.net/

[2] http://www.mikrocontroller.net/topic/artikelwettbewerb-2012-2013

[3] http://blog.embedded-projects.net

Hinweis: Zu jeden Artikel gibt es wieder einen Forumsbeitrag zur Diskussion

http://journal.embedded-projects.net/forum.php (Weiterleitung zu Mikrocontroller.net)

→shop.embedded-projects.net

HARDWARE FOR YOUR PROJECTS – ONLINESHOP

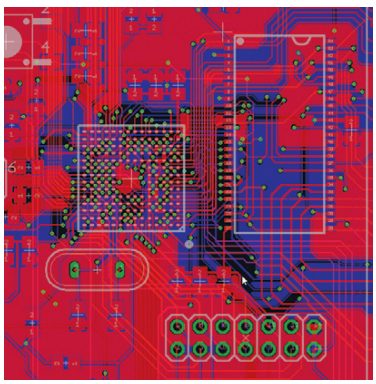
Design your GNUBLIN

Sie suchen ein Board, das fast so wie GNUBLIN ist und brauchen davon nur eine kleine Menge pro Jahr? Wir passen Ihnen auf kurzem Weg die Schaltung an und ergänzen diese nach Ihrem Wunsch. Dank unserer internen Bestückung können wir Ihnen bereits ab kleinen Mengen ähnliche Stückpreise wie bei den GNUBLIN Boards in diesem Shop liefern.

→ **www.gnublin.org/designer**

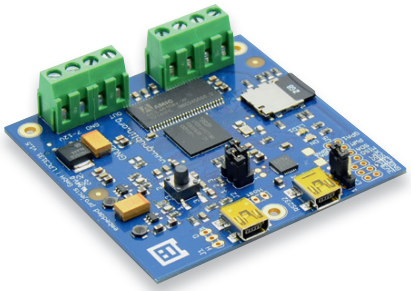
Die einfachste Möglichkeit an ein eigenes embedded GNU/Linux Board zu kommen!
Einfach und genial ist das Produkt „Design your GNUBLIN“. Für alle, die kleine Serien von Platinen mit embedded GNU/Linux benötigen. Basierend auf der Schaltung der Boards der GNUBLIN Familie können wir einfach und schnell kundenspezifische Lösungen erstellen. Innerhalb kürzester Zeit können wir Ihnen die Boards von 1 bis ca. 1000 Stück liefern.

NEU: online GNUBLIN-Designer mit Kalkulator



Wir zeichnen den Schaltplan und das Layout ...

... bestücken mit einem Automaten in Augsburg ...



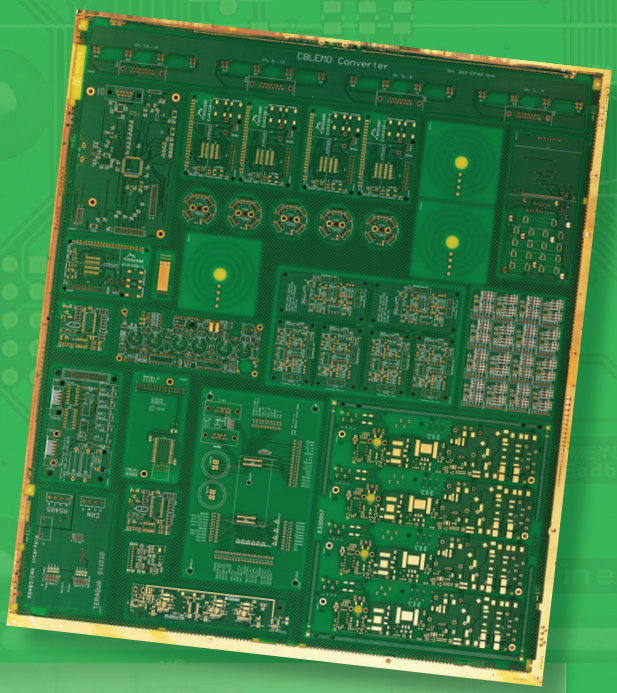
... und nehmen die Schaltung für Sie in Betrieb.

Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0
Fax +49 (0) 821 279599-20
shop@embedded-projects.net

embedded projects GmbH
HARDWARE FOR PROJECTS

EURO CIRCUITS

Die europäische Referenz für PCB Prototypen und Kleinserien



Kostensenkung durch Online-Pooling

- Keine Einrichtkosten
- Keine Mindestbestellwerte - ab der 1. Platine
- Sofort-Bestellung Online - ohne Vorkasse

Zeitgewinn durch neue Online-Daten-Dienste

PCB Visualizer® Sofort Design-Rule-Check

PCB Checker® sofortige Anzeige von DRC Fehlern im Layout

PCB Configurator® kalkulieren und prüfen von Layout-Parametern, Anzeige von Preis-Optionen

PCB proto

spezieller Prototypen-Service für Entwickler, preiswert und schnell

- 1, 2 oder 5 LP in 2, 3, 5 oder 7 Arbeitstagen
- 2 und 4 Lagen
- DRC-geprüft, prof. Ausführung, inkl. 2 x Lötstopplack, 1 x Bestückungsdruck, 150µm Technologie

STANDARD pool

die größte Auswahl an Eurocircuits Pooling Optionen

- FR-4, für bleifreies Löten geeignet, bis 16 Lagen
- Layout-Technologie bis 90µm
- ab 2 AT

RF pool

alle Pooling-Vorzüge mit Hochfrequenz-Material

- Material: Isola IS680 und Rogers 4000 Serie
- 2-4 Lagen, bis 100µm Technologie
- ab 3 AT

IMS pool

Aluminiumkern-Leiterplatten für hohe Wärmeableitung (z.B. LED-Anwendung)

- Leiterplatten mit einlagig isoliertem Metallsubstrat
- weisser/schwarzer Lötstopplack/Bestückungsdruck oder umgekehrt
- ab 3 AT

www.eurocircuits.de

PCB Visualizer® PCB Checker® PCB Configurator®

GhostCarProjekt

Platz 1: Artikelwettbewerb [1]

André F. <and_ref@canathome.de>

GCP - das GhostCarProjekt steht als Projekttitel für die Realisierung eines autonom fahrenden Fahrzeugs auf einer spurgebundenen Spielzeugrennbahn. Es soll ein Auto entwickelt werden, das auf einer Carrerabahn möglichst schnelle Rundenzeiten fährt. Dabei sind Eingriffe in die Strecke oder Steuerbefehle von außen tabu.

Abb. 1: Ghostcar mit seinen Brüdern

Idee und Motivation

Die legendäre Carrera®-Bahn aus Kindheitstagen ist vielen noch ein Begriff. Die Hauptmerkmale sind steckbare Schienenteile mit Führungsschlitz in der Mitte und zwei Spannungsschienen, die die für den Elektromotor nötige Energie liefern. Für manuell gesteuerte Autos wird durch einen Handregler der Schienenstrom (Analogbahn) oder ein PWM-Datenwort (Digitalbahn) vorgegeben und damit die Fahrgeschwindigkeit gesteuert. Für Digitalbahnen gibt es von Carrera unter dem Begriff Ghostcar auch automatisch fahrende Fahrzeuge. Diese fahren mit einer konstanten Geschwindigkeit, die manuell auf die langsamste Stelle auf der Strecke eingerichtet wird. Aber erst wenn das Fahrzeug auch auf Geraden schneller fährt und rechtzeitig vor Kurven abbremst, verhält sich das Ghostcar wie ein realer Gegner. Genau dies soll das Gcp-Fahrzeug (hier dann als Abkürzung für GhostCar Protoyp) leisten können.

Grundprinzip

Der entscheidende Punkt um schnell fahren zu können ist das Wissen über den Streckenverlauf und die Position des Autos auf der Strecke. Hierzu wurde ein Carrera®-Auto mit einem Gyrosensor und einer Reflexkoppler-Lichtschranke an der Hinterachse ausgestattet. Der Gyrosensor misst die Winkelgeschwindkeit (in Milligrad pro Sekunde [mdps]), oder anders formuliert, er liefert einen Wert, der aussagt, wie schnell sich das Fahrzeug um seine Hochachse dreht - im folgenden als GyroZ bezeichnet. Misst man sehr häufig und summiert alle Zahlenwerte auf, so ergibt sich daraus der zurückgelegte Kurvenwinkel (oder mathematisch ausgedrückt: Der überstrichene Winkel ist das Integral der Drehrate). Mit der Reflexkoppler-Lichtschranke, die eine Scheibe mit schwarzen und weißen Teilungsstrichen abtastet, lassen sich (Teil-)Umdrehungen der hinteren Antriebsräder messen. Werden die gezählten „Wheelticks“ durch eine (Tor-)Zeit dividiert, so erhält man daraus die Raddrehzahl (gemessen in RPM). Sofern kein Schlupf an den Rädern auftritt, ist die Fahrzeuggeschwindigkeit proportional zur Raddrehzahl.

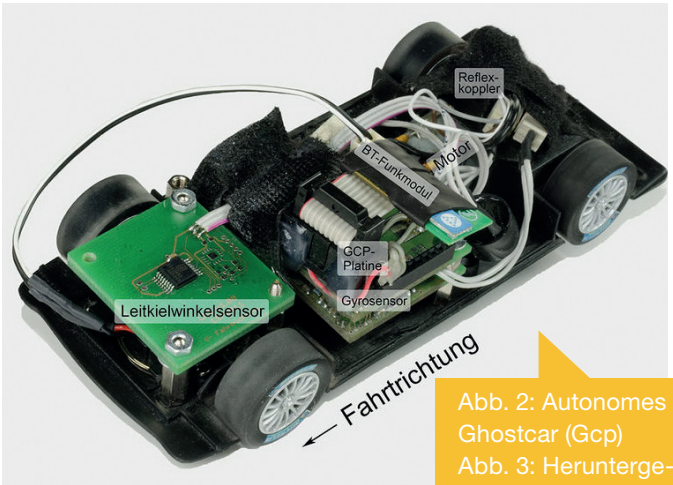


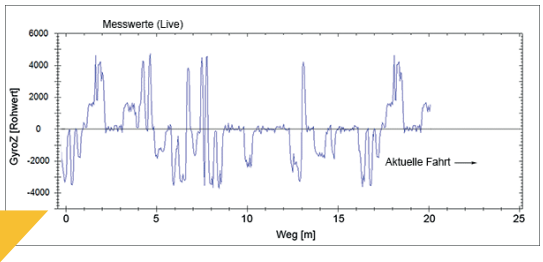
Abb. 2: Autonomes Ghostcar (Gcp)
Abb. 3: Heruntergeklappte Antriebsachse mit Reflexkoppler und Teilungsscheibe

Auswertung der Messwerte

Die nachfolgende Animation zeigt einen aufgezeichneten GyroZ-Verlauf einer kompletten Runde (plus etwas Zugabe):

Diese Strecke besteht aus 14 Rechts- und 9 Linkskurven unterschiedlicher Kurvenradien und -längen. Ein hoher GyroZ-Wert bedeutet (bei konstanter Geschwindigkeit) eine hohe Drehrate und damit einen engen Kurvenradius. Kleinere GyroZ-Werte bedeuten weniger enge Kurven. Die plateauartigen Passagen sind Teilstücke, die eine konstante Drehrate über eine längere Zeit (bzw. Weg) haben, z.B. langgezogene Kurven oder Geraden. Kurze Kurven oder kurze Geraden werden durch kurze Peaks abgebildet. Normiert man den GyroZ-Wert auf eine Geschwindigkeit von z.B. 1000RPM (=> GyroZNorm = Rohwert / Drehzahl * 1000), so erhält man eine von der Momentangeschwindigkeit unabhängige Aussage über den Kurvenradius. Die dargestellten Daten sind bereits umgerechnet auf den zurückgelegten Weg (anstatt den GyroZ-Verlauf über der Zeit darzustellen). Dies hat den Vorteil, dass man so immer den gleichen Kurvenverlauf erhält - egal, ob man langsam oder schnell fährt. Dieser Kurvenverlauf ist also ein charakteristischer Verlauf der Strecke und nicht der momentanen Fahrweise. Somit ist der genaue Verlauf der

Strecke bekannt, es liegt quasi ein Streckenmodell vor - bitte in Gedanken auf Karton ausdrucken.



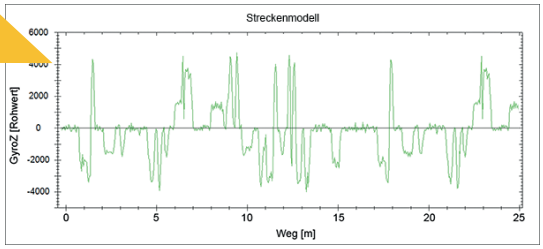


Abb. 3: Positionsermittlung
Abb. 4: Streckenmodell

Stellt man sich jetzt weiter vor, dass das Auto schon einen längeren Weg auf dieser Strecke zurückgelegt hat und sich gerade irgendwo mitten in der Runde befindet, könnte man den aktuellen GyroZ-Verlauf bis hierhin auf ein (ggf. sehr) breites Stück Transparentpapier drucken. Würde man dieses Transparentpapier beliebig über den Karton vom Streckenmodell legen, so könnte man durch hin- und herschieben des Transparentpapiers, beide Kurvenverläufe zur Deckung bringen. Dort wo der rechte Rand des Papiers ist, dort befindet man sich gerade (bezogen auf das Streckenmodell). Um zu wissen ob beschleunigt werden darf oder ob gebremst werden muss, muss man nur auf dem Streckenmodell (Karton) etwas nach rechts schauen (also quasi in die Zukunft blicken) und kann sich so auf den weiteren Streckenverlauf einstellen. Die nachfolgende Animation soll das prinzipielle Vorgehen verdeutlichen (Abb. 3 und Abb. 4).

In der Signalverarbeitung läuft das Verfahren des Übereinanderschiebens unter dem Begriff Autokorrelation. Liegen alle Messwerte gleichzeitig vor, so kann nach vielen Rechenschritten ausgesagt werden, wie die beiden Papiere zueinander geschoben werden müssen, um deckungsgleich zu sein, sprich, wo sich die Momentanposition auf dem Streckenmodell befindet. Allerdings liegt an dieser Stelle auch das Problem: Es sind weder alle Messdaten gleichzeitig(!) vorrätig (mangels RAM), noch steht genügend Rechenzeit zur Verfügung, um alle Verschiebungen durchrechnen zu können.

- Abschätzung der anfallenden Datenmenge: Messrate 1kHz; Rundenlänge 20s => 40kByte pro Runde
- Abschätzung der Rechenzeit: Näherungsweise eine Addition und eine Multiplikation (je 16bittig) pro Millisekunde UND pro Durchgang => 500ns * 20s * 1000[1/s] * 20.000 = 200s.

Um die Position fortlaufend bestimmen zu können, müsste man diese Rechnung mehrmals pro Sekunde abschließen können. So lässt sich dieses Verfahren also nicht umsetzen - es muss effizienter durchgeführt werden.

Übertragung des Grundprinzips in eine ressourcenschonende Implementierung

Dieser Abschnitt gibt nur einen kurzen Überblick über die grundlegenden Verarbeitungsschritte im Fahrzeug. Alle verwendeten Algorithmen und Ideen werden danach genauer unter die Lupe genommen. Das folgende Diagramm soll den Ablauf grob skizzieren:

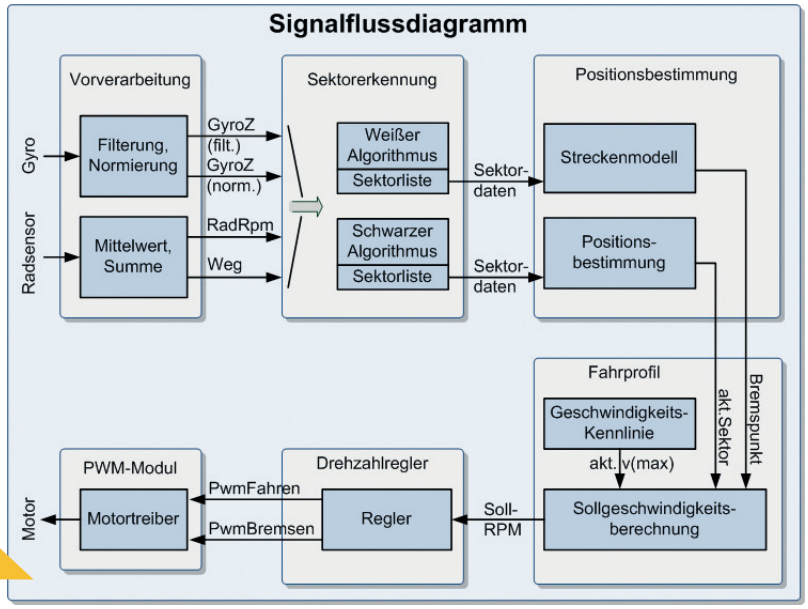


Abb. 5: Signalflussdiagramm

Sektorerkennung: Streckenaufteilung in Sektoren zur Datenverdichtung

Wenn man sich die GyroZ-Verläufe genau ansieht stellt man fest, dass sich immer Abschnitte finden lassen, in denen der GyroZ-Wert für eine gewisse Zeit stabil ist. Eigentlich ist der Verlauf fast eine Rechteckkurve. Das liegt an den Carrera®-Schienenteilen, die konstante Kurvenradien haben (es gibt keine parabelförmigen Kurven o.ä.). Dies lässt sich gut ausnutzen, um Abschnitte bzw. Sektoren zu bilden. Eingeschränkt gilt dies auch für frei verlaufende, „organische“ Bahnverläufe.

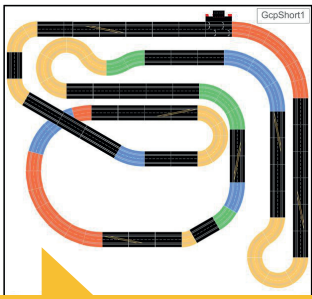


Abb. 6: Streckenverlauf „Gcp1Short“

Ein Sektor ist also ein Streckenabschnitt, dessen GyroZ-Verlauf nahezu konstant ist. Als Sektorparameter reicht es also aus, wenn nur der charakteristische GyroZ-Wert und die Sektorlänge gespeichert wird, um später daraus wieder den eigentlichen GyroZ-Verlauf rekonstruieren zu können. Dies gilt sowohl für Kurven, als auch für Geraden (bei denen der GyroZ-Wert ~0 ist). So lässt sich unsere Beispielstrecke „GcpShort1“ in ca. 31 Sektoren zerlegen, siehe Bild rechts. (Hinweis: Die Farben kennzeichnen den Kurvenradius und sollen die Strecke nicht in einzelne Sektoren teilen). Der benötigte Speicherbedarf ist mit weniger als 400Bytes (=MaxAnz Sektoren * 4Parameter zu je 16Bit; Details später) durchaus µC-tauglich.

Wie lassen sich aus dem kontinuierlichen Messdatenstrom Sektoren bestimmen?

Stellt man sich ein virtuelles horizontales Band um den GyroZ-Verlauf vor und legt fest, dass ein Sektor endet, sobald der GyroZ-Wert (für eine gewisse Zeit) die Bandgrenzen überschreitet, dann erhält man automatisch eine Stückelung in Abschnitte/Sektoren, die als Gemeinsamkeit einen ähnlichen GyroZ-Wert haben. Wenn man die Bandhöhe (also die Höhe des gewählten Bandes) so wählt, dass nicht zu viele Kleinstsektoren oder nichtaussagekräftige Großsektoren entstehen, hat man automatisch eine gute Annäherung

an den tatsächlichen (kontinuierlichen) GyroZ-Verlauf und muss nur ein Bruchteil der anfallenden Daten vorhalten. Aber wie und wann legt man sich auf die Bandmitte fest, um die letztendlich das Band gelegt wird? Die Praxis zeigt, dass sich der GyroZ-Wert nach kurzer Zeit auf sein neues stabiles Plateau einpendelt - und zwar relativ unabhängig von der Lage/Höhe des Plateaus. Man nimmt z.B. 75ms nach Sektorbeginn einfach den GyroZ-Wert heraus, der gerade anliegt und definiert diesen zur aktuellen Bandmitte für den gerade aktiven Sektor. Praktisch verbessert sich die Sektorqualität noch etwas, wenn man nicht den letzten Rohwert nimmt, sondern einen leicht gefilterten GyroZ-Wert heranzieht (PT1-Filter mit 100ms Filterzeit). Ignoriert man dann noch kurze Ausreißer aus dem Band, die z.B. wegen Messfehlern/-schwankungen vorkommen, dann erhält man relativ saubere Sektordaten, die den physikalischen Gegebenheiten bei langsamer Fahrt ziemlich genau entsprechen. Möchte man die Sektordaten dann nutzen, um die eigene Position auf der Strecke zu berechnen, und zeichnet daraus einen GyroZ-Verlauf, so lässt sich theoretisch genauso vorgehen wie oben mit den (quasi)kontinuierlichen Messwerten beschrieben (Stichwort: Autokorrelation; Verschieben der Papiere). Praktisch würde man dann die aktuellen Sektordaten mit vergangenen Sektordaten vergleichen und bei einer erkannten, identischen Folge könnte man auf die aktuelle Position schließen. Doch leider funktioniert dies so nicht ausreichend genau, da die Sektorgrenzen nicht immer am gleichen physikalischen Ort auf der Strecke liegen (also z.B. nicht genau am Kurveneingangspunkt), weil die „Vorgeschichte" des GyroZ-Verlaufs in die Wahl der Bandmitte einfließt und sich somit der Zeitpunkt (und auch Ort) des Bandverlassens verschiebt => Bremspunkt verschiebt sich => Abflug garantiert. Zudem ändern sich die Sektordaten deutlich, wenn sich die Fahrzeuggeschwindigkeit ändert. Es kommt bei höheren Geschwindigkeiten zu Drifts und Überschwingern. Dies führt zu „zufällig" auftauchenden Sektoren, die die Positionsbestimmung zusätzlich erschwert. Überschwinger treten z.B. am Kurvenausgang auf, wenn das Fahrzeug weiter seiner Kurvenbahn folgen möchte, obwohl die Strecke bereits in die Gerade übergegangen ist (Massenträgheit). Das führt dann dazu, dass die Sektordaten suggerieren, dass die Kurve länger erscheint als sie es tatsächlich ist. Der Kurvenausgangspunkt verschiebt sich so vermeintlich. Bei S-Kurven ist die Sache noch extremer. Hier hängen die Sektordaten noch stärker von der Fahrzeuggeschwindigkeit ab. Praxisbeispiel: Unsere Beispielstrecke ist bei langsamer Geschwindigkeit (1.5m/s = 1285RPM) ungefähr 30.4m lang. Bei leicht höherer Geschwindigkeit (2m/s = 1700RPM) ist die vom Fahrzeug gemessene Streckenlänge schon 31m! (Die Reproduzierbarkeit der Streckenlänge bei konstanter Geschwindigkeit beträgt unter 10cm). Die Abweichung kommt durch die angesprochenen Drifts zustande, aber auch durch „Abkürzen" von S-Kurven bei langsamer Fahrt. Alles in allem also keine guten Voraussetzungen für eine schnelle und sichere Fahrt. Dennoch liefern diese Sektordaten einen wichtigen Beitrag zum Streckenmodell und werden deswegen unbedingt benötigt. Die Länge aller Geraden wird ausreichend zuverlässig ermittelt, auch die Länge und Radien der Kurvenstücke ist gut genug, um daraus eine maximal mögliche Fahrgeschwindigkeit abzuleiten. Lediglich die Reproduzierbarkeit bzw. der genaue physikalische Ort der Sektorgrenzen ist verbesserungswürdig.

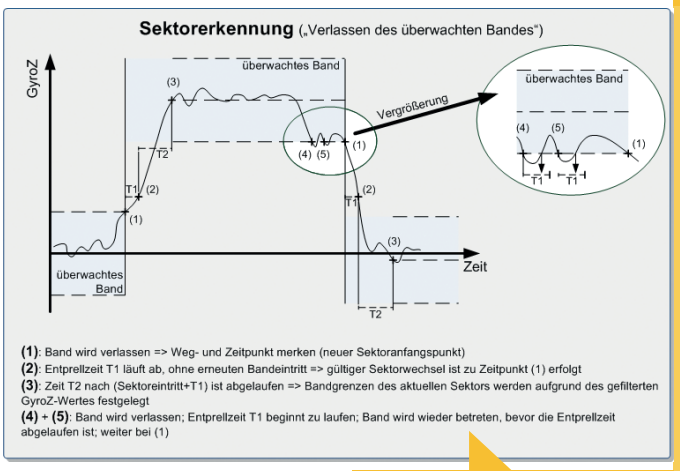


Abb. 7: Sektorerkennung

Was sind weiße/schwarze Sektoren und wie kann die Qualität der Sektordaten verbessert werden?

Um die Reproduzierbarkeit zu erhöhen, muss zusätzlich auf ein weiteres Verfahren zur Ermittlung von Sektorkenngrößen gesetzt werden. Um sprachlich hier nicht die beiden Konzepte zu vermischen, nennen wir die beiden Verfahren zur Sektorerkennung:

Das Konzept „Band verlassen/weiße Sektoren" ist oben bereits beschrieben. Im folgenden wird der schwarze Algorithmus näher betrachtet. Übrigens, die Begriffe sind willkürlich gewählt und sollen rein zur Unterscheidung dienen. Ziel des schwarzen Algorithmus ist es die Sektorgrenzen an einen festen Ort zu binden. Das Auto muss sich darauf verlassen können, dass der Bremspunkt korrekt sitzt. Kurvenradien zu erfassen oder besonders gute Abbildung des GyroZ-Verlaufs sind hier also nicht gefragt. Schaut man sich den GyroZ-Verlauf an einem Kurveneingang an, so fällt auf, dass dieser zunächst nahe 0 liegt und sich anschließend rampenförmig oder schon fast sprungförmig ändert. Kein Wunder, schließlich geht es ja von einer Geraden in eine Kurve über. Zur Erinnerung, die Carrera®-Schienenteile haben keinen weichen stetigen Übergang von Gerade in Kurve, sondern eine „unstetige Stelle". Aber selbst bei weichen Übergängen wäre ein Knick bzw. steiler Anstieg des GyroZ-Wertes vorhanden. Wenn man sich den Punkt merkt, bei dem der GyroZ-Wert ein gedachtes Nullband verlässt (auch hier wieder ein virtuelles Band um die Nulllinie vorstellen), dann sollte dieser Punkt in jeder Runde örtlich genau an der gleichen Stelle liegen. Am Ende einer (langen) Geraden fährt das Auto immer schnurgeradeaus. Um aber nicht Messrauschen oder kurzzeitige Schläge (z.B. durch die Stoßstellen der Schienen) fehlzuinterpretieren, wird verlangt, dass nach Abbiegen auch mindestens eine Kurve von 3° gefahren werden muss, sonst wird der Kurveneinlenkpunkt verworfen. Wurden aber die 3° „angesammelt", so gilt der Einlenkpunkt als Beginn eines neuen Sektors. Dieser Sektor endet erst wieder, wenn der GyroZ-Wert wieder ins Nullband zurückfällt UND es erneut verlässt (dabei natürlich wieder mehr als 3° überfährt).

Dadurch sind die schwarzen Sektoren nicht im geringsten mit den weißen Sektoren vergleichbar. Vereinfacht gesagt, beginnen und enden die schwarzen Sektoren immer an einem Übergang von Gerade in Kurve oder in S-Kurven. Dazwischen können aber beliebig viele andere Kurvenabschnitte liegen. Dies trifft auch für Kurven zu, deren Kurvenradius sich durch Aneinanderreihung von z.B. immer enger werdenden Schienenteilen ändert. Weiße Sektoren würden zu jeden einzelnen Schienenabschnitt einen Sektor liefern, der schwarze Sektor hingegen fasst diese alle zusammen. Speichert man die weißen und schwarzen Sektordaten in zwei voneinander getrennten Listen, so erhält man für die Beispielstrecke folgende (idealisierte!) Tabellen (siehe [1]). Die weißen Sektoren dienen fortan ausschließlich als Streckenmodell. Die schwarzen Sektoren werden zur Synchronisierung/Positionsbestimmung verwendet. So kombinieren sich die Vorteile beider Sektorarten.

Positionsbestimmung/Synchronisierung

Unsere erste Idee (die sich später allerdings als Sackgasse herausstellte), war es, zu Beginn eine Referenzrunde zu fahren. Die weißen Sektoren sollten das Streckenmodell (Referenzsektorliste) bilden. Später sollte dann nur noch ermittelt werden, wo wir uns momentan (bezogen auf die Referenzrunde) befinden. Da sich die Sektoren aber stark in Abhängigkeit der Fahrzeuggeschwindigkeit ändern, war das Streckenmodell schon veraltet, bevor es überhaupt genutzt werden konnte. Außerdem müsste zuverlässig erkannt werden können, wann die Referenzrunde zu Ende sein soll. Auch musste der gerade erfahrene Livesektor eindeutig einem Referenzrundensektor zuordenbar sein, wollte man die Synchronität nicht verlieren. Das Vorgehen hätte dann so ausgesehen: Die Referenzfahrt liegt als komplette Liste vor und beinhaltet alle Sektoren der Runde. Der zuletzt gemeldete Sektor wird auf der Referenzsektorliste gesucht. Es wäre keine Liste mit aktuellen Sektordaten erforderlich gewesen. Nachteil: Würde auch nur eine Synchronisierung fehlschlagen, dann müsste die komplette Referenzfahrt erneut durchgeführt werden, da keine Infos über die letzten aufgelaufenen Sektoren vorlägen.

Als wesentlich geeigneter erwies sich das fortlaufende Aktualisieren nur einer Sektorliste nach dem FIFO-Prinzip. Einzige Bedingung: Die Runde darf nicht mehr Sektoren liefern, wie die Liste aufnehmen kann. Was natürlich die maximale Rundenlänge begrenzt, aber auch sicherstellt, dass immer die aktuellsten Daten vorliegen. Die Fahrt beginnt also mit einer Lernphase, deren einziger Zweck es ist, die Sektorliste zu füllen. Sobald dies erreicht ist, geht die Fahrt nahtlos in die eigentliche Fahrt über und die Positionsbestimmung kann mit ihrer Arbeit beginnen. Ab jetzt wird schnell gefahren. (Derzeit umfasst die Liste 30 schwarze Sektoren, was auch für längere Strecken ausreicht).

Ziel: Es muss ausgehend vom aktuellen Sektor, der Vergleichssektor gefunden werden, der zu dem Streckenabschnitt vor genau einer Runde passt. Kurzum: „Der Vorrundensektor muss gefunden werden“. Die Positionsbestimmung wird immer dann ausgeführt, wenn ein neuer Sektor von der Sektorerkennung angeliefert wird. Wenn die Positionsbestimmung zustandslos arbeitet, d.h. keine Infos aus vorangegangenen Synchronisationen für die aktuelle Rechnung benötigt, dann kann es auch nicht zu unheilbaren Fehlsynchronisationen kommen. Würde man auf dem Wissen aus den SyncRechnungen aus den vorherigen Sektoren aufbauen, so könnte man sich in eine Sackgasse manövrieren, aus der man (algorithmisch) nur schwer wieder heraus kommt. Auf der Suche nach dem Vorrundensektor wird nicht einfach nur die gleiche Abfolge der letzten angefallenen Sektoren in der Sektorliste gesucht - da dies voraussetzen würde, dass die genaue Reihenfolge ohne Lücken und zusätzlichen Sektoren bereits so in der Liste steht. Wäre auch nur ein Sektor (z.B. wegen Überschwingern) hinzugekommen, dann käme der Algorithmus aus dem Tritt. Die Suche muss also robust gegen ausfallende bzw. zusätzliche Sektoren sein. Durch eine Ähnlichkeitssuche/Heuristik wird nicht die Reihenfolge der angefallenen Sektoren bewertet, sondern es wird versucht den Vorrundensektor über unabhängige Wahrscheinlichkeiten zu finden. Und das ist ganz ohne Sonderbehandlung zusätzlicher oder ausgefallener Sektoren möglich.

Heuristik: Wie lässt sich der Vorrundensektor zuverlässig aufspüren?

Getreu dem Motto „Wir können zwar nicht ausrechnen wo wir genau sind, dafür schätzen wir aber recht gut“, setzen wir für die Positionsbestimmung ein heuristisches Verfahren ein. Eine statistische Auswertung der in Frage kommenden Vorrundensektoren soll uns dabei helfen. Alle nachfolgenden Berechnungen finden ausschließlich mit den eigens hierfür eingeführten schwarzen Sektoren statt. Das Verschieben der Transparentpapierverläufe wird ersetzt durch numerisches Suchen von „ähnlichen“ Sektoren. Dabei wird in folgenden Schritten vorgegangen:

1) Ähnlichkeitsmatrix aufstellen
2) Häufigkeitsverteilung ermitteln
3) gewichtete relative Wahrscheinlichkeiten errechnen

Am Ende steht der Sektor mit der höchsten Wahrscheinlichkeit als der gesuchte Vorrundensektor fest.

1) Was ist die Ähnlichkeitsmatrix?

Die Ähnlichkeitsmatrix beschreibt, ob sich zwei Sektoren ähnlich sind. „Ähnlich“ bedeutet, dass sich die Sektordaten bis auf eine geringe Abweichung gleichen. („Eine enge 50cm lange 90° Rechtskurve ist einer anderen 48cm langen engen 94° Rechtskurve ähnlich - aber nicht einer 130cm langen Geraden“). Aufgrund der reinen Zahlenvergleiche kann noch nicht sicher zwischen dem echten Vorrundensektor und einem an anderer Stelle liegenden ähnlichen Sektor unterschieden werden, was hier aber auch noch nicht der Fall sein muss. Hierfür kommt später dann die Gewichtung der so gefundenen Kandidaten ins Spiel.

Bei dieser Ähnlichkeitsbetrachtung wird jeder Sektor der Sektorenliste mit jedem anderen Sektor davor verglichen. Die Ergebnisse kann man sich als Matrix notiert vorstellen.

Die rechts gezeigte Ähnlichkeitsmatrix basiert auf nur sechs Sektoren pro Runde. Das wurde so gewählt, damit die Tabelle hier in der Darstellung noch handlich bleibt. Die Werte sind frei erfunden und dienen nur zur Verifikation des Algorithmus. Die tatsächlich gemessenen schwarzen Sektordaten sind deutlich reproduzierbarer. Die Zahlenreihe ist also ein Extrembeispiel für schlechte Sektordaten. Dennoch soll auch hiermit die Positionsbestimmung noch möglich sein. (Die Zahlenwerte sind: 353, 208, 400, 479, 627, 125, 358, 207, 186, 211, 479, 619, 481, 207, 186, 214, 477, 607, 482, 207, 185, 210, 477, 585, 453, 206; das Ähnlichkeitskriterium hier sei ausschließlich der reine Zahlenwert; Toleranz +-10).

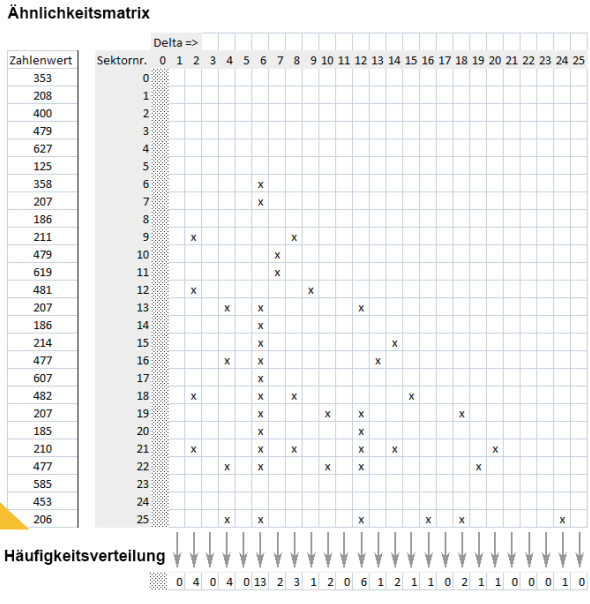


Abb. 8: Ähnlichkeitsmatrix und Häufigkeitsverteilung

Lesebeispiel: Der aktuelle Sektor (hier: 25) wird vergleichen mit allen seinen Vorgängersektoren (also von 0..24). Daraus entsteht dann die unterste Zeile (Zeile 25). Für jeden Sektor, der dem Sektor 25 ähnlich ist, wird ein Kreuz gesetzt. Genauso wird für alle anderen Sektoren/Zeilen verfahren. Nach rechts wird nicht die Vergleichssektornummer aufgetragen, sondern die Anzahl der Zeilen, die die beiden Sektoren in der Liste voneinander trennen (also der „Abstand" bzw. das Delta der Sektornummern). Das Kreuz bei Zeile25/Spalte4 (=Z25S4) bedeutet somit: Sektor 25 ist dem Sektor ähnlich, der einen Abstand/Delta von 4 zu ihm hat - also Sektor 21 (25-4=21). => „Sektor 25 und Sektor 21 sind sich ähnlich". Diese Darstellung hat ihren Vorteil darin, dass sich die Rundenlänge (in Sektoren) mit einem Blick ablesen lässt. In vielen Zeilen ist in Spalte 6 ein Kreuz => eine komplette Runde besteht wahrscheinlich aus 6 Sektoren. Übrigens, uns interessiert ja eigentlich der Vorrundensektor. Wenn wir aber wissen wie groß die (momentane) Rundenlänge ist, dann können wir daraus natürlich auch den Vorrundensektor errechnen.

2) Häufigkeitsverteilung

Die Häufigkeitsverteilung erlaubt eine Aussage darüber, welches die wahrscheinlichste Rundenlänge ist. Sie ist die Summe aller Kreuze in einer Spalte. Für Spalte 6 ergibt sich somit eine Häufigkeit von 13. D.h. 13 der 25 Sektoren sind denen ähnlich, die in der Liste 6 Sektoren davor stehen. Die Häufigkeitsverteilung fließt in die Berechnung der gewichteten relativen Wahrscheinlichkeit ein.

Was hat es mit der gewichteten relativen Wahrscheinlichkeit auf sich?

Da die Ähnlichkeitsmatrix alle Sektoren liefert, die sich ähnlich sind, aber keine Bewertung/Gewichtung für den wahren Vorrundensektor einführen kann, muss diese Information über ein anderes Verfahren ermittelt werden. Die für den Vorrundensektor relevante Zeile in der Ähnlichkeitsmatrix ist immer die letzte Zeile, also hier Zeile 25. Alle mit einem Kreuz versehenen Sektoren sind die Kandidaten, die für den Vorrundensektor in Frage kommen. Im Beispiel rechts sind das die Sektoren 21 (=25-4), 19 (=25-6), 13 (=25-12), 9 (=25-16), 7 (=25-18) und 1 (=25-24). Doch welcher davon ist der richtige? Die Frage lässt sich mit der „gewichteten relativen Wahrscheinlichkeit" beantworten. Diese lässt sich so bestimmen: Die Wahrscheinlichkeit, dass der jeweilige Kandidat dem gesuchte Vorrundensektor entspricht, verhält sich wie die Verteilung der Häufigkeit der Kandidaten (in der betrachteten Spalte) zur Gesamthäufigkeit. Oder anders gesagt: Wenn häufig ein Kreuz in der Spalte für Rundenlänge=6 ermittelt wurde, dann ist die Wahrscheinlichkeit hoch, dass die wahre Rundenlänge tatsächlich 6 beträgt („Mehrheitsentscheid"). Deswegen wird die Rundenlänge=6 auch bei der Positionsbestimmung für den aktuellen Sektor durch die relative gewichtete Wahrscheinlichkeit bevorzugt (=> „höherer Prozentwert"). Der Prozentwert errechnet sich aus dem Wert der Häufigkeitsverteilung * 100% dividiert durch die Sektorhäufigkeitssumme. Die Sektorhäufigkeitssumme ist die Summe aller in dieser Zeile beteiligten Häufigkeiten. Für Zeile 25 wäre dies: 4+13+6+1+2+1 = 27. Für Zeile/Sektornr. 25 und Rundenlänge=4 (also für Vergleichssektornr. 21) erhält man 4*100/27 = 14%. Für eine Rundenlänge=6 ergibt sich ein Zahlenwert von 13*100/27 = 48%. Die restlichen Kandidaten haben eine relative gewichtete Wahrscheinlichkeit von 22%, 3%, 7%, 3%.

Hinweis: Die Matrix (Abb. 9) ist komplett mit Wahrscheinlichkeitswerten ausgefüllt. Für die Bestimmung des Vorrundensektors würde es auch ausreichen, wenn nur die unterste Zeile berechnet/dargestellt wäre.

Der Kandidat mit der höchsten Wahrscheinlichkeit soll als Vorrundensektor anerkannt werden.

In diesem Fall ist dies der Sektor 19 mit Rundenlänge/Delta=6. Ein Blick in die Liste der Zahlenwerte (im Text oben) ergibt: Sektor25 hat einen Zahlenwert von 206, Sektor19 hat einen Zahlenwert von 207 => passt.

Für manche Zeilen lassen sich überhaupt keine Kandidaten finden, d.h. es kann kein Vorrundensektor bestimmt werden. => Infolge darf das Auto nur mit verringerter Geschwindigkeit weiterfahren, bis die Position auf dem Streckenmodell erneut feststeht (=wieder ein Vorrundensektor ermittelt werden kann). Die nächste Gelegenheit hierzu bietet sich dann, wenn der nächste schwarze Sektor gemeldet wird. Aber selbst wenn ein Kandidat mit einem hohen Prozentsatz (oder gar mit 100%) versehen ist, heißt dies nicht zwangsläufig, dass dies auch der wahre Vorrundensektor ist. Tatsächlich könnte es auch sein, dass dieser durch besondere Ereignisse (z.B. Räder drehen für längere Zeit durch) komplett „versteckt" ist und nicht mehr dem wahren Vorrundensektor ähnlich ist (evtl. aber jetzt zufällig einem anderen). Aus den ermittelten Kandidaten deutet die relative gewichtete Wahrscheinlichkeit denjeni-

gen heraus, der es mit der höchsten Wahrscheinlichkeit auch ist. (Nebenbei bemerkt: Bisher hat sich das Gcp praktisch noch nicht einmal verbremst und ist deswegen aus der Kurve geflogen, hat aber öfters keinen Kandidaten finden können und musste deswegen bis zum nächsten Syncpunkt langsam fahren).

Da die Rundenlänge hier ja nur 6 Sektoren beträgt, die Liste aber 26 Sektoren aufnehmen kann, müssten doch eigentlich mehrere Runden in der Liste auftauchen. Wieso synchronisiert sich der Algorithmus nicht versehentlich auf z.B. -12 Sektoren auf? Das könnte theoretisch durchaus passieren. In der Ähnlichkeitsmatrix sieht man auch in Spalte Delta=12, Delta=18 (schon weniger) und Delta=24 (nur sehr theoretisch) ebenfalls Ähnlichkeitshäufungen. Das Problem löst sich quasi von selbst, da die oberen Zeilen weniger Kreuze beitragen können (und wenn überhaupt, dann liegen diese wegen der Dreiecksform in den linken Spalten) und erfahren deswegen auch eine geringere Gewichtung.

Was passiert eigentlich in Zeile 9 oder Zeile 12? Hier vertut sich der Algorithmus komplett und wählt einen falschen Kandidaten. Würde das mit echten Messwerten passieren, dann könnte das Fahrzeug letztendlich von der Strecke fliegen.

Wieso beträgt in Zeile 10 und 11 die angebliche Rundenlänge = 7 Sektoren? Das liegt daran, dass in der Vorrunde (von Sektor 10 und 11), sich ein Sektor in zwei Sektoren aufgeteilt hat (z.B. durch Drift). Die Positionsbestimmung liefert also (auch) in diesem Fall einen korrekten Wert. Die Rundenlänge muss nicht für alle Ewigkeit konstant sein, sondern kann sich dynamisch zur Laufzeit ändern.

Was heißt eigentlich: Zwei Sektoren sind sich ähnlich

Als Kriterien zur Ähnlichkeitsbestimmung dienen Sektorlänge, Sektorwinkel und Sektorlage. Die Begriffe Sektorlänge und Sektorwinkel sind aus obiger Beschreibung schon bekannt. Die Sektorlage ist die Ausrichtung des Sektors in der Ebene (z.B. der Sektor liegt im 270°-Winkel zur Start-Ziel-Geraden). Erst wenn alle Parameter mit ihrem Wert innerhalb eines erlaubten Toleranzbereiches sind, gelten zwei Sektoren als „ähnlich“. Für die Sektorlänge haben sich 10 Wheelticks bewährt (das sind 10*17.5mm, also 17.5cm), der Sektorwinkel muss auf 5° identisch sein und die Lage darf um 60° abweichen. Das liegt im Langzeitdrift des Gyros begründet. Nach einer Drehung (z.B. nach einer Runde) liefert der Gyro nicht genau 360°, sondern weicht um bis zu 30° ab. Für Strecken die mehr als 360° Rundenwinkel haben (z.B. zwei ineinander verschachtelte Schleifen), muss die Lage folglich n*30° tolerieren (bei uns momentan auf 60° parametriert).

Fahrprofil: Jetzt steht der (schwarze) Vorrundensektor fest, was nun?

Der Vorrundensektor in der schwarzen Sektorliste liefert uns den Punkt, auf dem wir uns vor einer Runde befanden. Und da sich die Streckenführung definitionsgemäß nach genau einer Runde wiederholt, darf man ruhigen Gewissens davon ausgehen, dass der Vorrundensektor+1 derjenige Sektor ist, der als nächstes erreicht wird. Doch die schwarzen Sektoren liefern nicht die Infos, die zur Berechnung von Bremspunkt und maximaler Kurvengeschwindigkeit notwendig sind. Ein Wechsel zur weißen Sektorliste ist erforderlich. Das Bindeglied zwischen den Listen sind die absoluten Wegpunkte. Wegpunkte sind die aufaddierten Wheelticks (Reflexkoppler-Pulse) und werden in jeder Fahrt kontinuierlich hochgezählt - sie wiederholen sich nie (vergleichbar mit dem Gesamtkilometerzähler im echten Auto). Da bekannt ist, an welchem Wegpunkt der schwarze Vorrundensektor lag, kann in der Liste der weißen Sektoren derjenige Sektor gesucht werden, in dessen Bereich ebenfalls dieser Wegpunkt fällt (muss ja nicht zwangsläufig auf eine weiße Sektorgrenze fallen). Jetzt ist der Abstand zur nächsten (weißen) Sektorgrenze bekannt und auch der charakteristische GyroZ-Wert (Bandmitte) des nächsten Sektors liegt vor.

Woher weiß das Fahrzeug mit welcher Geschwindigkeit ein bestimmter Kurvenradius gefahren werden kann?

Zunächst einmal hängt die maximale Kurvengeschwindigkeit eines Fahrzeugs von seiner Bodenhaftung (u.a. Magnet im Unterboden, Fahrbahnbelag, Gewichtsverteilung, Reifenart usw.) ab und muss für jedes Fahrzeug einmalig ermittelt werden. Bestimmt man diese Geschwindigkeit exemplarisch für einen oder zwei bestimmte Kurvenradien, so kann man dies auf andere Kurvenradien inter-/extrapolieren. Aus der Physik wissen wir, dass sich die maximale Kurvengeschwindigkeit bei doppeltem Kurvenradius um Faktor Wurzel2 erhöhen darf. (Zentripetalkraft $Fz = m*v^2/r$). Diese Aussage wird durch praktische Beobachtungen gestärkt. Es lässt sich so ein Zusammenhang aufstellen, der in Abhängigkeit des GyroZ-Wertes (Betrag) eine maximal mögliche Drehzahl (=Geschwindigkeit) aufzeigt. Verpackt in einer (Lookup)Tabelle ergibt sich für unser Fahrzeug die rechts abgebildete Kennlinie. Der hyperbelförmige Verlauf wird durch einen unteren Wert begrenzt (hier z.B. 1400RPM). Dieser Wert steht für die Drehzahl, die auch am langsamsten Streckenabschnitt noch problemlos gefahren werden kann. Für Geraden (GyroZ=0) wurde eine Drehzahl von 8000RPM festgelegt. Diese wird mit unserem Fahrzeug auch bei sehr langen Geraden noch nicht erreicht. Übrigens, wenn von „Drehzahl“ die Rede ist, ist damit immer die Drehzahl der Hinterachse gemeint. Die Motordrehzahl ist durch das Getriebe nochmals um den Faktor 3 höher.

Wann muss gebremst werden?

Aus der Momentangeschwindigkeit, und der im nächsten Sektor erlaubten Maximalgeschwindigkeit (ermittelt aus der Kennlinie) lässt sich der Bremsweg berechnen. Passt der Bremsweg gerade noch in den aktuellen Sektor hinein, dann wird‘s Zeit zu bremsen. Das Bremsen wird dadurch erreicht, dass die Solldrehzahl langsam verringert wird (Bremsrampe). Eine sprungförmige Drehzahländerung würde zwar maximal stark verzögern, könnte aber in Kurven auch zu Instabilitäten des Fahrzeugs führen. Wenn sich z.B. eine schnelle Kurve „plötzlich“ zuzieht, dann könnte das Fahrzeug beim abrupten Bremsen ausbrechen. Umgekehrt wird auch nicht sprungförmig beschleunigt, um keine durchdrehenden Räder zu provozieren (kein Magnet im Fahrzeugunterboden verbaut => wenig Bodenhaftung => Vollgas => durchdrehende Räder). Legt man die maximal (gewünschte) Verzögerung, als auch die maximal (gewünschte) Beschleunigung als Einstellparameter aus, so lässt sich der Algorithmus an Fahrzeug und Strecke anpassen. Es wäre auch denkbar, dass diese Parameter in einer Kalibrierfahrt automatisch von der Software ermittelt werden.

Muss überhaupt gebremst werden?

Liefert die Bremswegberechnung quasi einen negativen Bremsweg, d.h. die Drehzahl muss an der Sektorgrenze nicht verringert werden, dann muss geprüft werden, ob nicht sogar beschleunigt werden darf. Beschleunigt werden darf immer dann, wenn die Maximalgeschwindigkeit für den aktuellen Sektor noch nicht erreicht ist. Vor der Sektorgrenze darf noch nicht beschleunigt werden (also nicht wie im echten Auto bereits im Kurvenscheitel), da ja die Schienenteile einen konstanten Radius haben und so-

mit in der ganzen Kurve nur eine (optimale) Geschwindigkeit erlauben. Ein früheres Beschleunigen würde zu übermäßigem Drift führen und letztendlich zu langsamerer Rundenzeit. Nebenbei bemerkt: Für frei gefräste Holzbahnen würde eine zulaufende Kurve durch die Sektorerkennung schon in mehrere Abschnitte aufgeteilt werden und diese Besonderheit würde sich dort nicht ergeben.

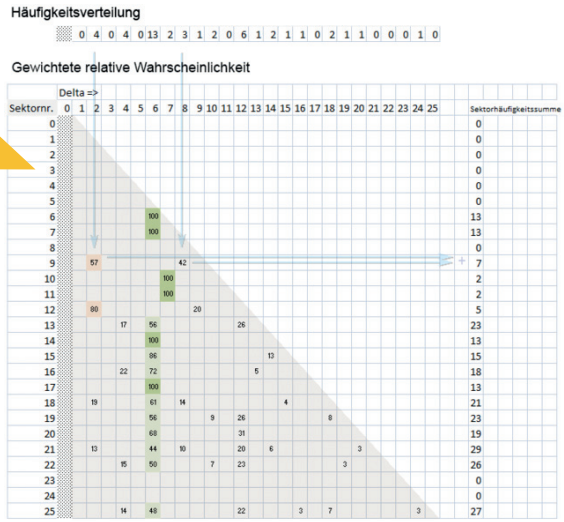


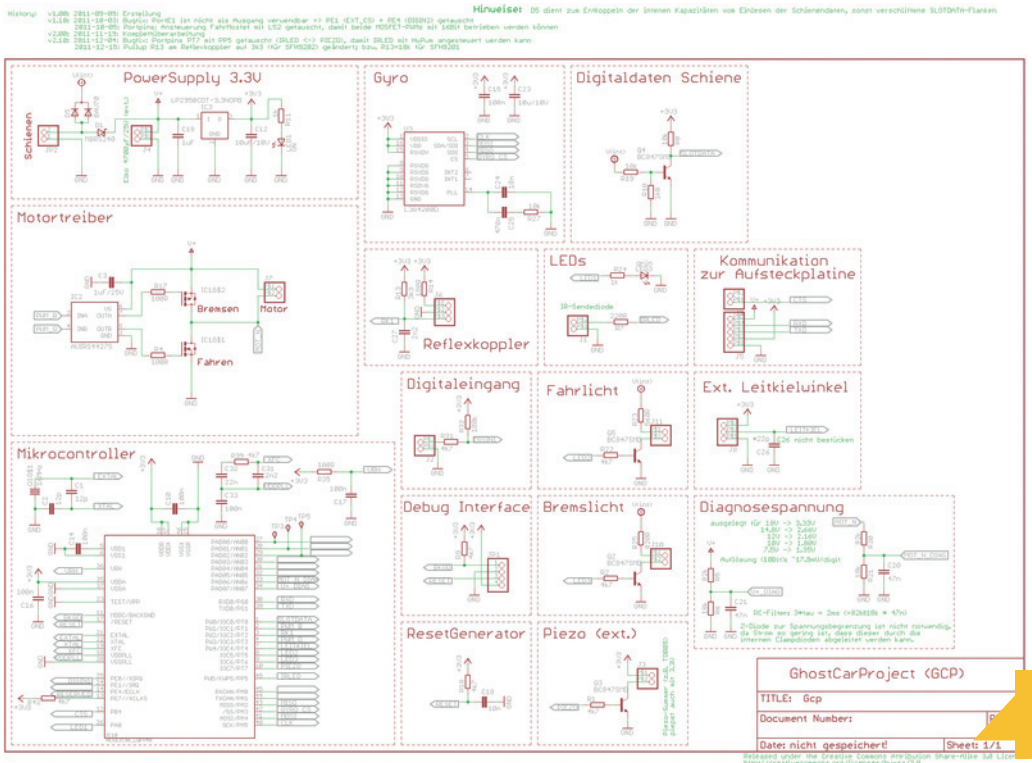
Abb. 9: Häufigkeitsverteilung und gewichtete relative Wahrscheinlichkeit

Drehzahlregler

Die von der Sollgeschwindigkeitsberechnung ermittelte Solldrehzahl („SollRPM“) wird an den Drehzahlregler übergeben, der daraus ermittelt, mit welchen PWM-Tastverhältnissen die Motortreiber-Mosfets angesteuert werden müssen. Es gibt einen Kanal zur Ansteuerung des „Fahr-Mosfets“, der die Schienenspannung auf den Motor schaltet und einen Kanal für den „Brems-Mosfet“ zum aktiven Bremsen, der den Motor kurzschließt. Beide dürfen natürlich nicht gleichzeitig angesteuert werden, da es sonst zu einem harten Kurzschluss kommt. Der Regler ist als PI-Regler ausgelegt. Die Reglerparameter wurden experimentell so ermittelt, dass eine möglichst schnelle Annäherung an die Führungsgröße (SollRpm) gegeben ist, ohne dass der Regler (stark) überschwingt. Die PWM-Kanäle werden mit 25kHz betrieben und lösen das Tastverhältnis in 0.1%-Schritten auf.

Elektronik

Als Spenderfahrzeug für das Gcp musste ein ausrangiertes Auto herhalten, dessen Elektronik durch eine eigene Platine ersetzt wurde. Neben den Sensoreingängen und den Motortreibern erwies sich auf dem Prototypen auch eine Kommunikationsschnittstelle zum PC für Logging/Debug-Aufgaben als zwingend notwendig.



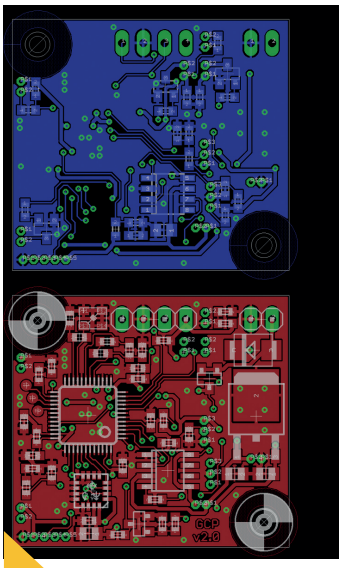


Abb. 10: Layout (Großansicht siehe PDF oder [1]

Abb. 11: Schaltplan (Großansicht siehe PDF oder [1]

Technische Daten:

Sensoren/Aktoren:

- 3-Achsen-Gyrosensor: L3G4200D von ST
- Reflexkoppler-Lichtschranke zur Drehzahlerfassung: SFH9202
- Motortreiber + Mosfets: AUIRS4427S + IRF7343 (zweikanalig zum Beschleunigen und aktiven Bremsen)
- Einlesemöglichkeit der Schienendaten (Carrera® Digital); Spannungsmessung
- IR-Sendediode im Fahrzeugboden zum Schalten der Carrera Weichen
- Ansteuerung der Fahrzeug LEDs (Fahrlicht, Bremslicht)
- Magnet-Winkelencoder am Leitkiel (AS5045; Optional, zur Messung der Leitkielstellung, Driftwinkel usw.)
- (Beschleunigungssensor nicht (mehr) verbaut)

Kommunikation:

- RS232-Kommunikationsinterface über Aufsteckplatine (BT-Modul von Free2Move im transparenten RS232-Modus)
- Piezo-Piepser

Prozessor:

- Freescale HCS12C96 (50MHz); 3.3V; 96k Flash (10k benutzt); 4k RAM (3k benutzt)

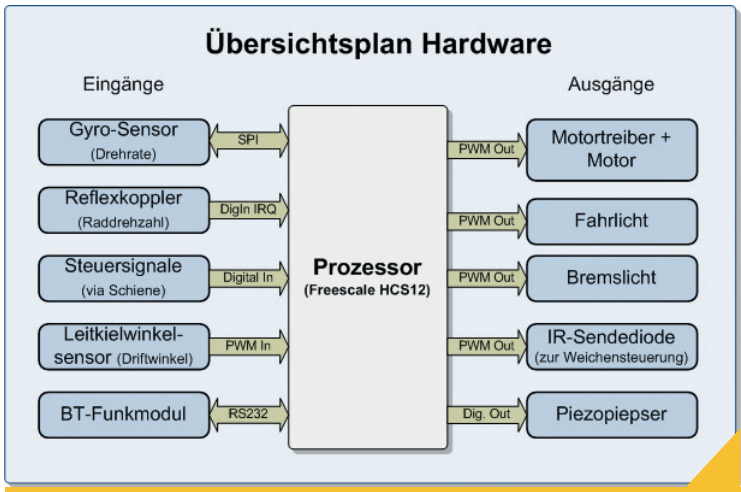


Abb. 12: Übersichtsplan HW

Der Gyrosensor liefert einen Drehratenwert (skaliert in 70 MilliDegrees per Second; mdps) und wird zyklisch jede 1ms ausgelesen.

Zur Drehzahlmessung (und Entfernungsmessung) wurde auf der hinteren Antriebsachse eine Scheibe angebracht, die pro Radumdrehung 8 Flankenwechsel liefert. Da wir momentan nur die High-Low-Flanken auswerten erhalten wir bei einem Radumfang von 70mm eine Auflösung von 17.5mm pro Reflexkoppler-Impuls. Der Impuls wird nicht nur gezählt, sondern der genaue Zeitpunkt wird zusätzlich per InputCapture-Einheit erfasst, um eine deutlich höhere zeitliche Auflösung zu erhalten und somit die Raddrehzahl auch bei „niedrigen" Drehzahlen sauber erfassen zu können. Im relevanten Drehzahlbereich von ca. 1000RPM..8000RPM kommt im Schnitt etwa alle 1..10ms ein Wheeltick (=17.5mm).

Die beiden Beschleunigungs- und Bremsmosfets werden mit einem Mosfet-Treiber angesteuert, da diese weniger Platz auf der sowieso knappen Platinenfläche einnimmt und auch die Dimensionierung der sonst notwendigen diskreten Mosfet-Beschaltung vereinfacht.

Der verwendete HCS12-Prozessor von Freescale wurde gewählt, weil dieser aus anderen Projekten bereits vertraut war und die technischen Daten (Taktfrequenz, Speicher) dem Einsatz nicht im Wege standen, bzw. alles so ausgelegt wurde, dass die Ressourcen passen.

Versuche mit 3-Achsen-Beschleunigungssensoren brachten keine befriedigenden Ergebnisse, weshalb diese (auch aus Platzgründen) nicht verbaut wurden. Das Nutzsignal war in der gleichen Größenordnung wie das Rauschen, woraus sich keine sinnvoll verwertbaren Daten ableiten ließen. Ursache dafür waren vermutlich die starken Motor- und Fahrbahnvibrationen, die sich trotz mechanischer Entkopplung (Moosgummi) nicht ausreichend verbesserten.

Der Leitkielwinkelsensor wurde als Vorhalt eingeplant, um den fahrdynamischen Grenzbereich (wenn das Fahrzeug langsam in den Drift übergeht) besser ausmessen zu können. Der Leitkiel ist eine Führungsschiene im Unterboden des Fahrzeugs, die im Schlitz in der Fahrbahn entlangläuft und zur Spurführung dient. Hierzu wird die Leitkielstellung relativ zum Fahrzeug gemessen. Bei Kurveneinfahrten dreht sich der Leitkiel relativ zum Fahrzeug ein Stück zur Seite. Übersteigt dieser Winkel einen bestimmten Wert, so kann man daran ein Driften des Fahrzeugs ablesen. Durch Montage eines kleinen Magneten auf dem Leitkiel lässt sich dessen Winkel bzw. Verdrehung relativ zum feststehenden MagnetWinkelencoder messen. Die magnetische Winkelauflösung beträgt 12Bit, was ca. 0.1° entspricht - also weit genauer als hier nötig wäre. Alle 4ms wird ein neuer Messwert geliefert. Derzeit messen wir den Leitkielwinkel zwar, verwerten die Daten aber nicht. Grund: Da die Montage relativ aufwändig ist (modifizierter Leitkiel; Befestigungsmöglichkeiten), wollten wir nur dann auf die Daten zurückgreifen, falls eine ausreichend schnelle Fahrt nicht ohne diesen möglich wäre.

Das Einlesen der Carrera® Digitaldaten auf den Schienen ist dazu gedacht, um das Fahrzeug vom Benutzer parametrieren und manuell fahren zu können. Denkbar wäre es z.B. die Fahrgeschwindigkeit künstlich einzuschränken oder den Fahrbeginn vom Benutzer per Handregler starten zu können. Voraussetzung dafür ist ein schnelles und ungepuffertes Einlesen der Schienenspannung. Die Schiene führt bei Digitalbahnen auf der einen Spur Gnd und auf der Anderen einen Pegel von 12-18V je nach Trafo. Zur Kommunikation von der Carrera®-Steuereinheit zum Fahrzeug ist der Gleichspannung ein PWM-Signal überlagert. Eine Datenübertragung vom Fahrzeug zur Steuereinheit ist nicht vorgesehen. Das ungepufferte Einlesen der Schienenspannung steht in entgegengesetzter Anforderung zu einer stabilen Spannungsversorgung für die Elektronik. Durch Lücken in der Schiene („Weichen") kommt es zu Aussetzern von ein paar wenigen Zentimetern bzw. wenigen hundert Millisekunden, die von einem Stützelko überbrückt werden müssen. Die überlagerten Kommunikationssignale sind deshalb (per Diode) von der internen Versorgung entkoppelt.

Grundsätzlich unterstützt das GCP auch Analogbahnen mit einer konstanten Spannungsversorgung (=> „Handregler per Klebeband auf Vollgas fixieren")

Die IR-Sendediode im Fahrzeugboden erlaubt bewusst eingeleitete Spurwechsel, um z.B. jederzeit auf der Ideallinie fahren zu können. Um die anderen Fahrzeuge nicht abzuschießen, würde das aber praktisch auch mehr Rundumsicht benötigen (=> „Abstandssensoren"). Automatische Spurwechsel sind derzeit aber noch nicht implementiert.

Um den Reflexkoppler einzusparen, und so den Nachbau zu erleichtern, bzw. zu verbilligen, haben wir auch die Drehzahl ohne Reflexkoppler getestet („Drehzahlmessung per Gegen-EMK"). Dabei wird ausgenutzt, dass die im Motor induzierte Gegenspannung proportional zu seiner Drehzahl ansteigt. Misst man diese „elektromotorische Kraft" zu einem Zeitpunkt in der der Motor nicht bestromt wird („PWM-Aus"-Phase), so kann man eine Aussage über seine Drehzahl treffen. Praktisch funktioniert dies auch prinzipiell, allerdings war die Messung nicht so genau wie die Drehzahlmessung per Reflexkoppler, was dann den Regler (bzw. unsere Reglerparameter) überforderte. Das Fahrzeug fuhr deutlich „unrunder". Das Motorgeräusch war schrecklich. Deshalb haben wir diese Option bisher auch noch nicht weiter vertieft, obwohl da noch Verbesserungspotential stecken dürfte.

Software

Neben den obligatorischen Komponenten wie Scheduler und Lowlevel-Treibern (InputCapture, Pwm, Adc, Spi) besteht der Kern der Gcp-Software aus den beiden Komponenten Kommunikationsmodul (ApplComm) und allem, was mit der Bestimmung Fahrzeugposition und Ansteuerung des Motors zusammenhängt (ApplLap).

Die oben beschriebenen Konzepte und Ideen sind im Modul ApplLap (Download [1]) implementiert. Der zentrale 1ms-Task, in dem alle zyklisch anfallenden Arbeiten ausgeführt werden, verwaltet die einzelnen Teilkomponenten, wie Sektorerkennung weiß, Sektorerkennung schwarz und Soll-Drehzahlberechnung. Die Teilkomponente Positionsbestimmung ist weniger zeitkri-

tisch, aber relativ rechenzeitintensiv, weshalb diese nicht wiederkehrend alle 1ms gerechnet wird, sondern nur dann, wenn ein neuer potentieller Syncpunkt anfällt (also ein neuer schwarzer Sektor geliefert wird). Die Ausführung findet im IdleTask statt, der immer dann (weiter)gerechnet wird, wenn der 1ms-Task seine Arbeit erledigt hat. Die Positionsbestimmung wird somit ständig von den zyklischen 1ms-Aufgaben unterbrochen. Es ist mit dem aktuellen Stand der Software so, dass der 1ms-Task maximal 820µs Laufzeit verbraucht und für den Idletask pro 1ms nur (minimal) 180µs Rechenzeit übrig bleiben. Letztendlich liefert die Positionsbestimmung dann aber nach spätestens 30ms einen neuen Syncpunkt. (Der zwischenzeitlich verfahrene Weg wird natürlich berücksichtigt).

Das Kommunikationsmodul (Download) implementiert ein kleines Protokoll zur Datenübertragung zum PC. Hierüber werden in verschiedenen Modi (die vom PC vorgegeben werden) folgende Daten vom Fahrzeug an den PC gesendet:

LiveMeasure-Daten: interne Variablen zur (grafischen) Echtzeitanalyse, wie z.B. MomentanDrehzahl, SollDrehzahl, aktueller GyroZ-Wert, Debugvariablen usw.; bis zu 6 Parameter werden (typ.) alle 50ms übertragen.

Fastlog: Echtzeitübertragung aller physikalischen Eingangsgrößen(!)

Weiße und schwarze Sektordaten: Länge, Wegpunkt, Winkel, Lage, Bandmitte zur späteren automatischen oder händischen Auswertung.

SyncTelegramme: Wegpunkt der Synchronisierung und um wie viele Wheelticks („cm“) die errechnete Position korrigiert werden musste

Diagnosedaten, wie z.B. Prozessorauslastung im 1ms-Task/Idle-Task, um Ressourcenprobleme erkennen zu können.

PC-Software zur Auswertung und Simulation

Die empfangenen LiveMeasure-Daten werden von einer dafür entwickelten PC-Software in Echtzeit in einen Graphen eingetragen und helfen somit bei der visuellen Bewertung der Fahrzeugsituation. Hilfreich ist dies z.B. beim Aufspüren von Fahrzeugdrifts am Kurveneingang oder -ausgang (GyroZ-Sprung/Schwinger im Graphen) oder bei der Bewertung der Reglerparameter („Istdrehzahl schwingt über“). Da hierbei nicht alle Messwerte übertragen werden können, reicht diese Analysemethode nur für erste grobe Einschätzungen.

Daneben gibt es aber viele Situationen, die schwieriger zu bewerten sind. Zum einen, weil nicht alle, zur Analyse nötigen Daten, vorliegen (50ms Aktualisierungsrate reicht oft nicht aus), zum anderen, weil die 6 Livemeasure-Kanäle nicht ausreichen.

Die Frage „Wieso wurde die Sektorgrenze nicht gesehen“ lässt sich nicht nur durch Betrachten des GyroZ-Verlaufes beantworten, sondern es ist notwendig den Algorithmus live zu beobachten. Da sich das Fahrzeug aber bewegt und sich damit eine Onboard-Debugmöglichkeit ausschließt, mussten die Daten zur späteren Auswertung aufgezeichnet werden.

Hierzu nimmt der PC die in Echtzeit/live gesendeten physikalischen Messgrößen entgegen (Betriebsart „Fastlog“) und legt diese in einem Logfile ab. Dadurch können auch sehr lange Messfahrten problemlos aufgezeichnet werden. Am Ende werden die physikalischen Eingangsgrößen am PC genauso nachgerechnet, wie diese auch im Fahrzeug berechnet wurden. So ist es möglich zu beliebigen Zeitpunkten und an beliebigen Stellen das Verhalten des Algorithmus zu betrachten/debuggen und die grafische Ausgabe zu analysieren. Die PC-Software enthält hierzu eine Komponente (Targetcode.cs), in dem der (relevante) Microcontrollercode fast 1:1 identisch nachgerechnet wird. Die Dateien lassen sich relativ einfach per Diff-Tool auf gleichem Stand halten.

Um algorithmische Probleme zu lösen, die keinen Echtzeitbezug benötigen, reichen die Daten der Sektortelegramme aus. Für die heuristische Positionsbestimmung z.B. ist eine Simulation umgesetzt, die als Eingangsdaten, rein die gelieferten Sektordaten benötigt. In der Ausgabe der Simulation lassen sich somit einfach die Ähnlichkeitsmatrix, Häufigkeitsverteilung oder gewichtete relative Wahrscheinlichkeit realer Fahrzeugsituationen analysieren.

Auch das Bestimmen der Parameter für die Sektorerkennung (weißer Algorithmus) ließ sich sehr anschaulich über eine PC-Simulation lösen. Hier kann in Echtzeit grafisch beobachtet werden, welchen Einfluss das Verstellen der Parameter auf die Wahl der Sektorgrenzen hat (siehe Video [1]). So ist schnell überprüfbar, ob die Sektordaten sinnvoll die tatsächliche Kurvenform (GyroZ-Verlauf) wiedergeben können.

Fazit: Ohne die umfangreichen Simulationsmöglichkeiten wäre es unmöglich gewesen auch nur ansatzweise ein autonom fahrendes Fahrzeug zu bauen. Der Einsatz des Prozessordebuggers konnte auf die Fälle beschränkt werden, an denen (einfache) elektrische/physikalische Probleme auftaten (z.B. Probleme beim Einlesen der Reflexkoppler-Daten aufgrund von Prellen, bzw. der elektrische Signalform und bei Implementationsfehlern im Kommunikationsprotokoll...). Die restlichen, schwierigeren algorithmischen Implementationsprobleme ließen sich komfortabel in der Simulation austesten.

Links/Literatur/Kontakt

[1] http://www.mikrocontroller.net/articles/GhostCarProjekt

[2] Abbildungen: http://www.mikrocontroller.net/articles/GhostCarProjekt

Aufgrund des begrenzten Umfangs dieses Artikels, müssen leider viele Punkte offen bleiben oder werden nur in einem kurzen Nebensatz angerissen. Sollten noch Fragen auftauchen, bitte melden.

GCP ist ein Freizeitprojekt von and_ref und galoscha. Eine kommerzielle Verwertung ist nicht gestattet. Bei entsprechendem Interesse bitte anfragen. :-)

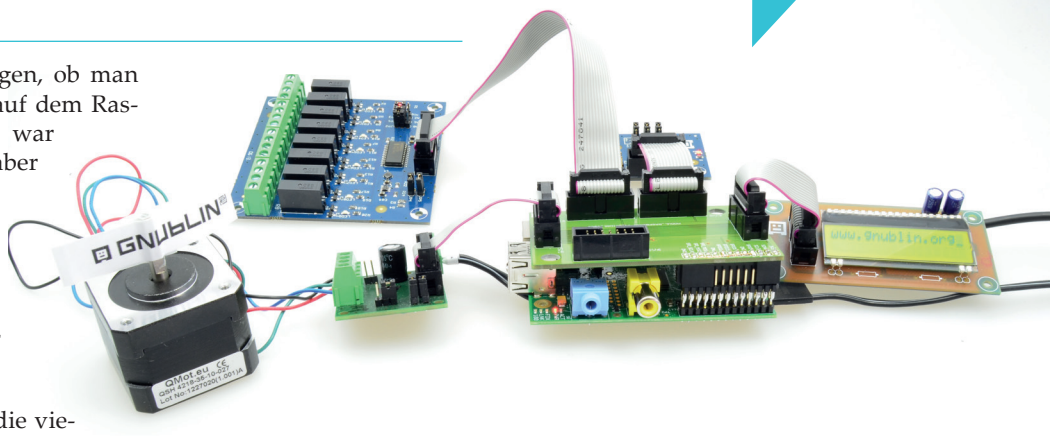
[3] Kontakt: André; and_ref (at) canathome.de

GnuPi

Eine Adapterplatine zur Nutzbarmachung der Gnublin Module auf dem Raspberry Pi

Simon Kunzmann <simon.kunzmann@googlemail.com>

Abb. 1: Raspberry Pi mit verschiedenen Modulen und der Adapterplatine GnuPi



Einleitung

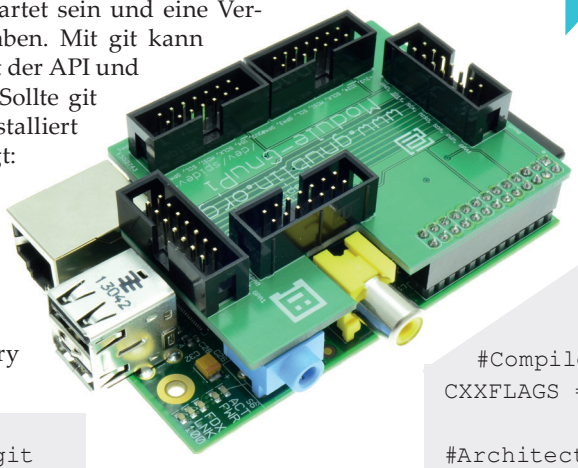
Immer wieder erreichten uns Anfragen, ob man denn unsere Gnublin Module auch auf dem Raspberry Pi verwenden könne. Das war immer schon möglich, erforderte aber etwas Bastelarbeit. Nun haben wir eine Adapterplatine entwickelt, die man einfach auf das Raspberry Pi steckt und schon hat man wie gewohnt 5 Steckerbuchsen auf die man ganz einfach unsere Module per Flachbandkabel verbinden kann.

Ein riesen Vorteil von Gnublin sind die vielen Tools. Zu jedem Modul (auch Selbstbau möglich) gibt es ein kleines Tool auf der Kommandozeile, mit dem man das Modul bereits ansteuern kann. Es passiert das, was man erwartet: der Motor dreht sich, ein Temperaturwert wird ausgegeben, am Display erscheint Text, usw.

Passend zu jedem Modul gibt es in der API Aufrufe um eigene Programme in C und C++ schreiben zu können. Neben der Unterstützung für Gnublin wird und wurde ein Support für das Raspberry Pi umgesetzt.

Vorbereitungen auf dem Raspberry Pi

Abb. 2: Die Adapterplatine GnuPi



Das RaspberryPi muss gestartet sein und eine Verbindung in das Internet haben. Mit git kann man jetzt das Repository mit der API und den Gnublin Tools klonen. Sollte git auf RaspberryPi nicht installiert sein macht man das wie folgt:

```
pi@raspberrypi ~
$ sudo apt-get
install git
```

jetzt kann man das Repository abholen:

```
pi@raspberrypi ~ $ git
clone https://github.com/
embeddedprojects/gnublin-api.git
```

Wechselt in das Verzeichnis:

```
pi@raspberrypi ~ $ cd gnublin-api
```

Bevor man die API übersetzen kann, muss man angeben, das die API auf einem Raspbberry Pi laufen soll. Dazu ändert man in der Datei API-config.mk zwei Zeilen ab:

Die Zeilen 2, 12 und 17 werden auskommentiert, die Zeilen 6, 14 und 18 dafür einkommentiert. Die API-config.mk Datei sieht dann wie folgt aus:

```
#Crosscompiler for
Gnublin
#CXX := arm-linux-gnueabi-g++
#Crosscompiler for Raspberry Pi:
#CXX := arm-linux-gnueabihf-g++
#Compiler for onboard compilation:
CXX := g++

  #Compilerflags:
CXXFLAGS = -Wall

#Architecture for gnublin:
#Architecture = armel
#Architecture for raspberryPi:
Architecture = armhf

#Define which Board you want:
#BOARD := GNUBLIN
BOARD := RASPBERRY_PI

#DO NOT EDIT BEYOND THIS LINE!
BOARDDEF := -DBOARD=$(BOARD)
```

Nun kann man die API übersetzen und die Programme und Beispiele installieren:

```
pi@raspberrypi ~ $ make && sudo make install
```

Damit die Programme auch alle richtig funktionieren, muss man noch die folgenden Treiber laden:

```
pi@raspberrypi ~ $ sudo modprobe spi-bcm2708

pi@raspberrypi ~ $ sudo modprobe i2c-bcm2708

pi@raspberrypi ~ $ sudo modprobe i2c-dev
```

Damit werden die Treiber für den SPI und den I2C Bus geladen. Möchte man die Treiber bei jedem Systemstart automatisch laden, kann man sie auch fest in die Datei /etc/modules eintragen:

```
spi-bcm2708
i2c-bcm2708
i2c-dev
```

Pro Zeile ein Modul.

Nun kann man ganz einfach die Module seiner Wahl anschließen und wie gewohnt die Gnublin Tools nutzen.

Abb. 3:
GnuPi- Video

Nutzung der API in eigenen Programmen

Die Nutzung der API in eigenen Programmen ist denkbar einfach. Man fügt in seinem Programm oben die folgenden beiden Zeilen ein (Reihenfolge beachten!):

```
#define BOARD RASPBERRY _ PI

#include „gnublin.h"
```

Und schon kann man auf die API zugreifen. Für mehr Informationen, Dokumentation und Beispielprogramme siehe:

http://wiki.gnublin.org/index.php/API

Links/Literatur

[1] http://www.gnublin.org

[2] http://wiki.gnublin.org/index.php/API

XML & XSLT: C-Code Generierung einer Finite State Machine

Klaus Weichinger <snaky.1@gmx.at>

Einführung

In der 13ten Ausgabe von Embedded Projects Journal wurden im Artikel „Implementierung einer Finite State Machine“ [1] die Grundlagen zu endlichen Automaten vermittelt und unterschiedliche Implementierungsmöglichkeiten aufgezeigt. Darauf aufbauend zeigt dieser Beitrag, wie mit Hilfe von XML endliche Automaten beschrieben und mit XSLT der zugehörige C-Code automatisch generiert werden können. Als Anwendungsbeispiel wird eine einfache Temperaturregelung modelliert und mit einem Arduino [2] realisiert.

Beispiel: Temperaturregelung

Die Temperaturregelung soll die Temperatur in einem Bereich zwischen 50 und 55°C halten. Fällt die Temperatur unter 50°C wird ein Heizelement eingeschaltet und beim Überschreiten der Temperatur von 55°C wieder ausgeschaltet. Wird die jeweilige Grenze nach 100 Sekunden nicht erreicht, geht die Regelung in einen Fehlerzustand über. Mit zwei Tastern soll die Temperaturregelung ein- und ausgeschaltet werden können. Zwei LEDs zeigen den Gerätezustand und den Fehlerzustand an. Die UML-Darstellung des beschriebenen endlichen Automaten ist in Abb. 1 dargestellt.

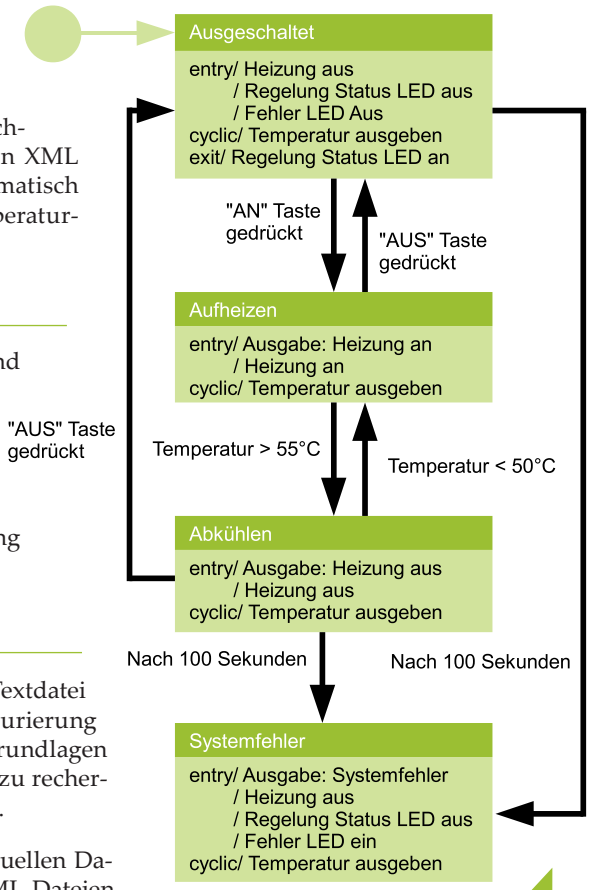


Abb. 1: Modellierung der Temperaturregelung in UML

Modellierung mit XML

Bei XML handelt es sich um eine Auszeichnungssprache, mit der in einer Textdatei Informationen strukturiert gespeichert werden können. Die Art der Strukturierung kann frei gewählt und an die Anwendung optimal angepasst werden. Die Grundlagen zu XML können in [3] nachgelesen werden. Es lohnt sich aber auf jeden Fall zu recherchieren, ob es für die Anwendung bereits eine geeignete Strukturierung gibt.

XML findet in vielen Systemen zur Datenspeicherung Anwendung. Die aktuellen Dateiformate für Dokumente und Tabellenkalkulaten sind meistens mehrere XML-Dateien, welche komprimiert in einer Datei zusammengepackt sind (zusätzlich sind darin Bilder und andere Objekte abgelegt). Weiters findet XML beispielsweise im Graphikformat SVG Anwendung.

Im nächsten Schritt wird das mit UML dargestellte Modell der Temperaturregelung in Form einer XML-Datei dargestellt (siehe Listing 1). Die dafür verwendete Struktur wurde so gewählt, dass alle Informationen kompakt und ohne größere Einschränkungen eingegeben werden können.

Listing 1: statemachine.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="./xsl/statemachine2c.xsl"?>
<statemachine name="temperaturregelung" title="Digitale Temperaturregelung">
  <!-- Definition eines endlichen Automaten einer Temperaturregelung -->
  <input id="An"          type="unsigned char" default="0" description="An Taster"/>
  <input id="Aus"         type="unsigned char" default="0" description="Aus Taster"/>
  <input id="Temperatur" type="signed short"  default="0" description="Temperatur in °C"/>

  <output id="Heizung"   type="unsigned char" default="0" description="0/1...Zeizung Aus/An"/>
  <output id="Status"    type="unsigned char" default="0" description="0/1...LED Aus/An"/>
  <output id="Error"     type="unsigned char" default="0" description="0/1...LED Aus/An"/>

  <startstate id="ausgeschaltet"/>
```

```
<state id="ausgeschaltet" description="Regelung ausgeschaltet">
  <transition nextstate="aufheizen">
    <input id="An" op="neq" value="0"/>
  </transition>
  <entry>
    <output id="Heizung" value="0"/>
    <output id="Status" value="0"/>
    <output id="Error" value="0"/>
  </entry>
  <exit>
    <output id="Status" value="1"/>
  </exit>
  <cyclic>
    <code>Serial.println(sm-&gt;inputs.Temperatur,DEC);</code>
  </cyclic>
</state>

<state id="aufheizen" description="Aufheizen">
  <transition nextstate="ausgeschaltet">
    <input id="Aus" op="neq" value="0"/>
  </transition>
  <transition nextstate="abkühlen">
    <input id="Temperatur" op="gt" value="55"/>
  </transition>
  <transition nextstate="fehler">
    <time op="gt" value="100"/>
  </transition>
  <entry>
    <code>Serial.println(„Heizung An");</code>
    <output id="Heizung" value="1"/>
  </entry>
  <cyclic>
    <code>Serial.println(sm-&gt;inputs.Temperatur,DEC);</code>
  </cyclic>
</state>

<state id="abkühlen" description="Abkühlen">
  <transition nextstate="ausgeschaltet">
    <input id="Aus" op="neq" value="0"/>
  </transition>
  <transition nextstate="aufheizen">
    <input id="Temperatur" op="lt" value="50"/>
  </transition>
  <transition nextstate="fehler">
    <time op="gt" value="100"/>
  </transition>
  <entry>
    <code>Serial.println(„Heizung Aus");</code>
    <output id="Heizung" value="0"/>
  </entry>
  <cyclic>
    <code>Serial.println(sm-&gt;inputs.Temperatur,DEC);</code>
  </cyclic>
</state>

<state id="fehler" description="Systemfehler">
  <entry>
    <code>Serial.println(„Systemfehler");</code>
    <output id="Error" value="1"/>
    <output id="Heizung" value="0"/>
    <output id="Status" value="0"/>
  </entry>
```

Die ersten beiden Zeilen sind XML Deklarationen, gefolgt vom Wurzelelement beschrieben mit den Start- und End-Tags <statemachine>...</statemachine>. Das Wurzelelement besitzt die beiden Attribute name und title, wobei name für die Bezeichnung der generierten C-Struktur verwendet wird. Innerhalb des Wurzelelements befinden sich mehrere Kindelemente, die den Automaten beschreiben. Mit den leeren Elementen (das sind Elemente ohne Kindelemente) <input ..../> und <output .../> werden die Ein- und Ausgänge definiert. Das Attribut id gibt den Ein- und Ausgängen eindeutige Bezeichnungen, die später zur Referenzierung benötigt werden.

Die Elemente <state>...</state> beschreiben die unterschiedlichen Zustände. Der Startzustand wird durch das leere Element <startstate .../> festgelegt. Ein Zustand wird nun durch Transitionen, Entry-Aktionen, Cyclic-Aktionen und Exit-Aktionen definiert. Eine Transition besteht aus einer Bedingung (z.B. <input id="Aus" op="neq" value="0"/> entspricht Eingang „AUS"!=0) und einer Angabe des nächsten Zustandes, in den übergegangen wird, wenn die Transitionsbedingung zutrifft. Eine Bedingung kann auch aus logischen UND und ODER Operationen zusammengesetzt sein (Beispiel: <and><input id="x" op="equ" value="0"/><or><input id="y" op="equ" value="0"/><input id="y" op="equ" value="3"/></or></and> entspricht in C der Bedingung (x==0 && (y==0 || y==3))). Die Aktionen bestehen aus Wertzuweisungen zu den Ausgängen bzw. direktem C-Code für eigene Funktionsaufrufe.

Es besteht auch die Möglichkeit, die Gültigkeit der Struktur und der eingegebenen Daten eines XML-Dokuments mit Hilfe von DTD (Document Type Definition) oder einem XML Schema zu überprüfen, welche für den jeweiligen XML-Dokumenttyp erstellt werden müssen (siehe dazu [4] und [5]).

```
    <cyclic>
      <code>Serial.println(sm-&gt;inputs.Temperatur,DEC);</code>
    </cyclic>
  </state>
</statemachine>
```

Transformation in C-Code

Die XML Darstellung der Temperaturregelung ist nun vollständig, jedoch kann die Zielplattform – der Arduino – diese nicht ausführen. XSLT [6] wird hier nun dazu verwendet, die XML Darstellung in einen kompilierbaren C-Code für den Arduino zu transformieren. Der Ablauf der Transformation ist in Abb. 2 dargestellt.

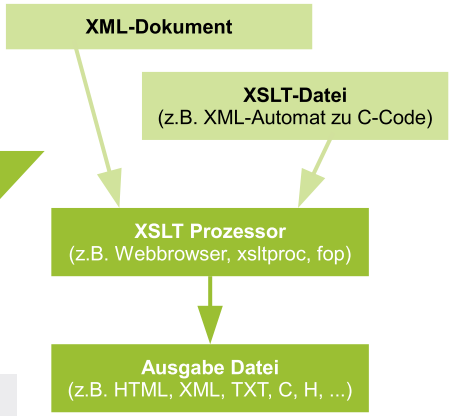


Abb. 2: Transformation einer XML-Datei mit XSLT

Die einfachste Möglichkeit XSLT anzuwenden, ist die Verwendung eines aktuellen Webbrowsers. Diese sind meistens mit einem XSLT-Prozessor ausgestattet. Wichtig dabei ist, in der XML-Datei zu vermerken, welche XSLT-Datei verwendet werden soll (siehe Zeile 2 in Listing 1). Öffnet man nun die XML-Datei mit dem Browser, wird mit der XSLT-Datei die Transformation durchgeführt und das Ergebnis vom Browser dargestellt.

Listing 2: statemachine2c.xsl; Einige Auszüge, die vollständige Datei umfasst 192 Zeilen.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" versi-
on="1.0">
  <xsl:template match="/statemachine">
    <html>
      <head>
        <title>State-Machine - <xsl:value-of select="@title"/></title>
      </head>
      <body>
        <h2>C-Code</h2>
        <p style="font-family:'Courier New', Arial;">
          #ifndef _ <xsl:value-of select="@name"/> _ H<br/>
          #define _ <xsl:value-of select="@name"/> _ H<br/>

          <!-- === Code removed here =============-->
          void <xsl:value-of select="@name"/> _ Init(struct <xsl:value-
of
select="@name"/> *sm)<br/>
          {<br/>
              sm-&gt;state=0;<br/>
              sm-&gt;nextstate=0;<br/>
              sm-&gt;statetimer=0;<br/>
              sm-&gt;timertick=0;<br/>
            <br/>
            <xsl:for-each select="input">
                sm-&gt;inputs.<xsl:value-of select="@
id"/>=<xsl:value-of select="@default"/>;<br/>
            </xsl:for-each>
            <br/>
            <xsl:for-each select="output">
                sm-&gt;outputs.<xsl:value-of select="@
id"/>=<xsl:value-of select="@default"/>;<br/>
            </xsl:for-each>
          }<br/>
          <!-- === Code removed here =============-->
          #endif
        </p>
```

Wie in Listing 2 ersichtlich ist, ist die XSLT-Datei selbst wieder eine XML-Datei. Die Elemente <xsl:....>...</xsl:....> sind dabei die vom XSLT-Prozessor umgesetzten Anweisungen. Alle anderen Zeichen sind Bestandteil des Ausgabedokuments. Zu beachten ist, dass für die Ausgabe anstatt „<“, „>“, „&“ die Entitäten „<“, „>“, „&“ verwendet werden müssen. Die Entität   fügt gezielt Leerzeichen ein, welche normalerweise von XML und HTML zu einem Leerzeichen zusammengepackt werden. Mit diesem Trick werden die Einrückungen für den Source-Code realisiert, worunter die Lesbarkeit der XSLT-Datei leidet.

Bei der Transformation wird zuerst mit <xsl:template match="/statemachine"> ... </xsl:template> das Wurzelelement des XML-Dokuments ausgewählt und die darin definierte Transformation angewandt. Mit <xsl:value-of select="@title"/> wird der Titel des endlichen Automaten ausgegeben. „@title“ selektiert das Attribut title des aktuellen Elements und ist ein Sprachelement von XPath [7]. XPath ermöglicht es, durch die Elemente und

```
      </body>
    </html>
  </xsl:template>

  <!-- === Code removed here ============-->
  <xsl:template match="output" mode="entryexit">
    <xsl:te
xt>          
    </xsl:text>
    <xsl:text>sm-&gt;outputs.</xsl:text>
    <xsl:value-of select="@id"/>
    <xsl:text>=</xsl:text>
    <xsl:value-of select="@value"/>
    <xsl:text>;</xsl:text>
    <br/>
  </xsl:template>

  <!-- === Code removed here ============-->
  <xsl:template match="input|time" mode="transition">
    <xsl:text>(</xsl:text>
    <xsl:if test="name()='input'">
      <xsl:text>sm-&gt;inputs.</xsl:text>
    </xsl:if>
    <xsl:if test="name()='time'">
      <xsl:text>sm-&gt;statetimer</xsl:text>
    </xsl:if>
    <xsl:value-of select="@id"/>
    <xsl:choose>
      <xsl:when test="@op='gt'"><xsl:text>&gt;</xsl:text></
xsl:when>
      <xsl:when test="@op='lt'"><xsl:text>&lt;</xsl:text></
xsl:when>
      <xsl:when test="@op='equ'"><xsl:text>==</xsl:text></
xsl:when>
      <xsl:when test="@op='neq'"><xsl:text>!=</xsl:text></
xsl:when>
    </xsl:choose>
    <xsl:value-of select="@value"/>
    <xsl:text>)</xsl:text>
  </xsl:template>
  <!-- === Code removed here ============-->
</xsl:stylesheet>
```

Attribute eines XML-Dokuments zu navigieren. Um durch die Definitionen der Eingänge zu iterieren wird das Konstrukt <xsl:for-each select="input">....</xsl:for-each> verwendet. Innerhalb der Entry/Cyclic/Exit Aktionselemente können durch den Aufruf <xsl:apply-templates mode="entryexit"/> andere definierte Templates angewandt werden. Am Ende von Listing 2 sind exemplarisch die Templates für einige Transitionsbedingungen und Ausgangsaktionen angeführt.

Das Resultat der Transformation mit dem Webbrowser ist in Abb. 3 abgebildet. Mit einem Kommandozeilen XSLT-Prozessor (z.B. xsltproc [8]) kann die Übersetzung automatisiert in den Build-Prozess eines Projektes eingebunden werden.

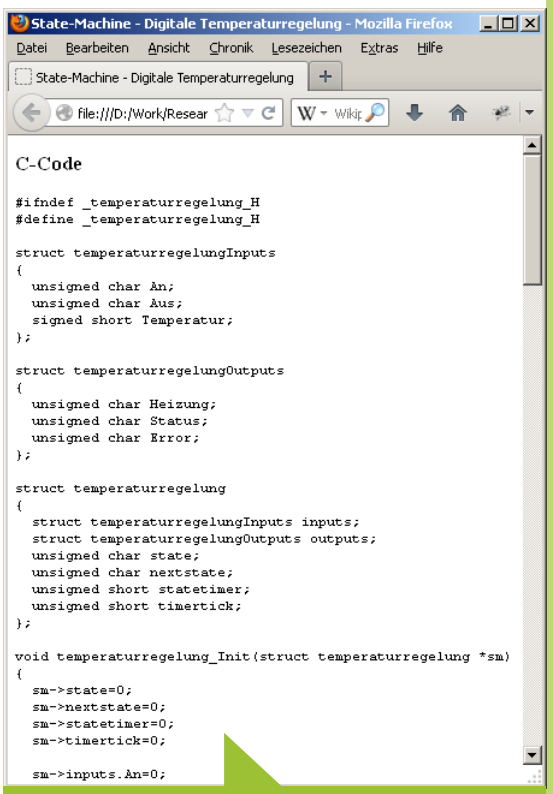


Abb. 3: XSLT Ausgabe im Webbrowser

Zielplattform Arduino

Im letzten Schritt wird die Temperaturregelung mit einem Arduino getestet. Ein RC-Glied wird zur Emulation der Heizung verwendet. Die Kapazitätsspannung entspricht der Temperatur (0... 102.3°C), und der Widerstand realisiert die Heizung und Abkühlung. Der Aufbau mit dem Arduino ist in Abb. 4 abgebildet. Der generierte C-Code für den endlichen Automaten verwendet als Schnittstelle zur Hauptanwendung folgende C-Struktur und Funktionen:

- struct temperaturregelung sm; ... Instanzierung eines endlichen Automation in Form einer C-Struktur
- temperaturregelung_Init(&sm); ... Initialisierung der C-Struktur
- temperaturregelung_Tick(&sm); ... Gibt einen Takt vor (z.B. jede Sekunde). Dieser Takt wird für die Zeitmessung in einem Zustand verwendet. Außerdem werden die Cyclic-Aktionen in diesem Takt aufgerufen, wenn keine Transition ausgelöst wurde.
- temperaturregelung_Process(&sm); ... Arbeitet den Automaten ab. Überprüfung der Transitionen. Anfallende Aktionen werden ausgeführt.

Die Ein- und Ausgänge des Automaten sind in der C-Struktur sm unter sm.inputs.<inputname> und sm.outputs.<outputname> zugänglich.

Die korrekte Anwendung dieser Funktion ist in Listing 3 demonstriert. Über die serielle Schnittstelle werden die Temperatur und Statusmeldungen ausgegeben. Eine Messung der Temperatur (Kondensatorspannung) und des Heizungssignals (A1) wird in Abb. 5 gezeigt.

Listing 3: Einbindung des Automaten im Arduino Framework

```
// Generierten C-Code hier einfügen

struct temperaturregelung sm;
#define AnPin 7
#define AnGNDPin 6
#define AusPin 2
#define AusGNDPin 3
#define TemperaturPin A0

#define HeizungPin A1
#define StatusPin 13
#define ErrorPin 10
#define ErrorGNDPin 11

void setup() {
  Serial.begin(9600);
  temperaturregelung_Init(&sm);
  pinMode(AnPin,INPUT_PULLUP);
  pinMode(AusPin,INPUT_PULLUP);
  digitalWrite(AnGNDPin,LOW);
  digitalWrite(AusGNDPin,LOW);
  pinMode(AnGNDPin,OUTPUT);
  pinMode(AusGNDPin,OUTPUT);

  digitalWrite(HeizungPin,LOW);
  digitalWrite(StatusPin,LOW);
  digitalWrite(ErrorPin,LOW);
  digitalWrite(ErrorGNDPin,LOW);

  pinMode(HeizungPin,OUTPUT);
  pinMode(StatusPin,OUTPUT);
  pinMode(ErrorPin,OUTPUT);
  pinMode(ErrorGNDPin,OUTPUT);
}

void loop() {
  static unsigned long lasttime=0;
  static unsigned long time=0;
  static unsigned char anFilter=0;
  static unsigned char ausFilter=0;
  // Taster entprellen
  anFilter<<=1;
  if (digitalRead(AnPin)==LOW) anFilter|=1;
  ausFilter<<=1;
  if (digitalRead(AusPin)==LOW) ausFil-
ter|=1;
```

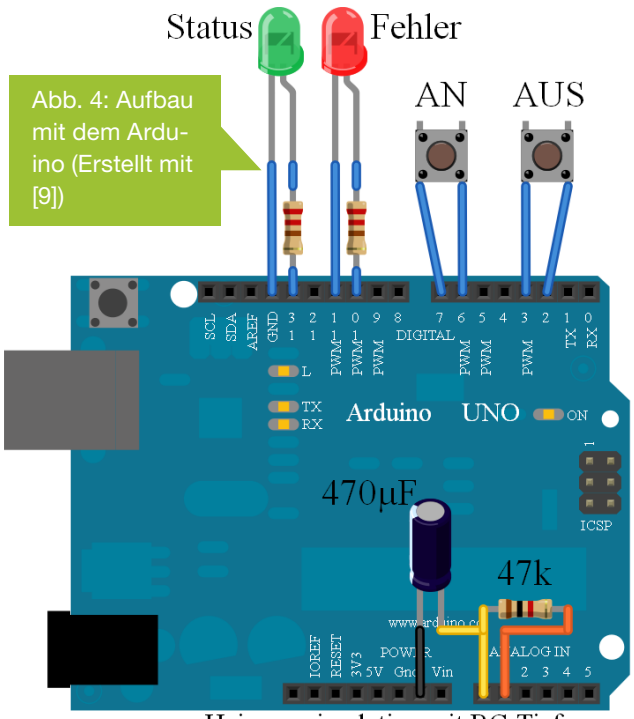


Abb. 4: Aufbau mit dem Arduino (Erstellt mit [9])

Heizungssimulation mit RC-Tiefpass

```
  // Taste wurde gedrückt Ereignis erkennen
  // (steigende Flanke)
  sm.inputs.An=((anFilter&0b00011111)==
0b00001111)?1:0;
  sm.inputs.Aus=((ausFilter&0b00011111)==
0b00001111)?1:0;

  // Temperatur messen
  sm.inputs.Temperatur=analogRead(TemperaturP
in)/10;

  time=millis();
  if (time>lasttime+1000uL)
  {
    temperaturregelung_Tick(&sm);
    lasttime=time;
  }

  temperaturregelung_Process(&sm);

  digitalWrite(HeizungPin,(sm.outputs.
Heizung>0)?HIGH:LOW);
  digitalWrite(StatusPin,(sm.outputs.
Status>0)?HIGH:LOW);
  digitalWrite(ErrorPin,(sm.outputs.
Error>0)?HIGH:LOW);
}
```

Weitere Ideen

Eine Verbesserung und Erweiterung der XML Darstellung endlicher Automaten und des Code-Generators ist natürlich jederzeit möglich. Beispielsweise können Transitionen um Aktionen ergänzt, verschachtelte und parallel laufende Zustandsautomaten ermöglicht werden. Es ist naheliegend, anhand der XML Darstellung nicht nur C-Code, sondern auch beispielsweise eine auf HTML basierte Dokumentation des endlichen Automaten automatisch generieren zu lassen, um somit Dokumentation und Code ohne Mehraufwand immer am aktuellen Stand zu halten.

Denkbar wäre es auch, mit XSLT eine HTML-Datei mit eingebettetem JavaScript zu erzeugen, welche die direkte Simulation des endlichen Automaten im Webbrowsers ermöglicht.

Zum Denkanstoß noch weitere Möglichkeiten um XML im Embedded-Bereich einzusetzen:

- Ein µC gibt z.B. Statusinformationen oder LOG-Aufzeichnungen direkt im XML Format aus. So kann eine XSL Transformation dazu verwendet werden um diese Daten visuell aufbereitet am PC darstellen zu lassen.
- Parametersätze können übersichtlich in XML dargestellt werden, und bei jedem Kompiliervorgang werden automatisch die entsprechenden #define ... erzeugt.

- KiCad [10] verwendet beispielsweise xsltproc [8], um Stücklisten zu generieren. Das Programm bietet auch die Möglichkeit eigene Transformationen anzuwenden. Die von KiCad zur Verfügung gestellte XML-Datei beinhaltet dabei bei weitem mehr als nur Stücklisteninformationen.
- Neben C-Code und Dokumentation kann auch das Diagramm des endlichen Automation mit Hilfe von PlantUML [11] erstellt werden. PlantUML bietet einen Online-Service an, womit die Integration in eine Web-Seite mit Hilfe von JavaScript kein Problem ist. Das Ergebnis könnte wie in Abb. 6 aussehen.

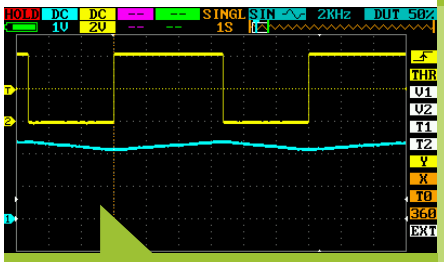


Abb. 5: Messung mit dem Oszilloskop (CH1...Kapazitätsspannung/ Temperatur, CH2...Heizung)

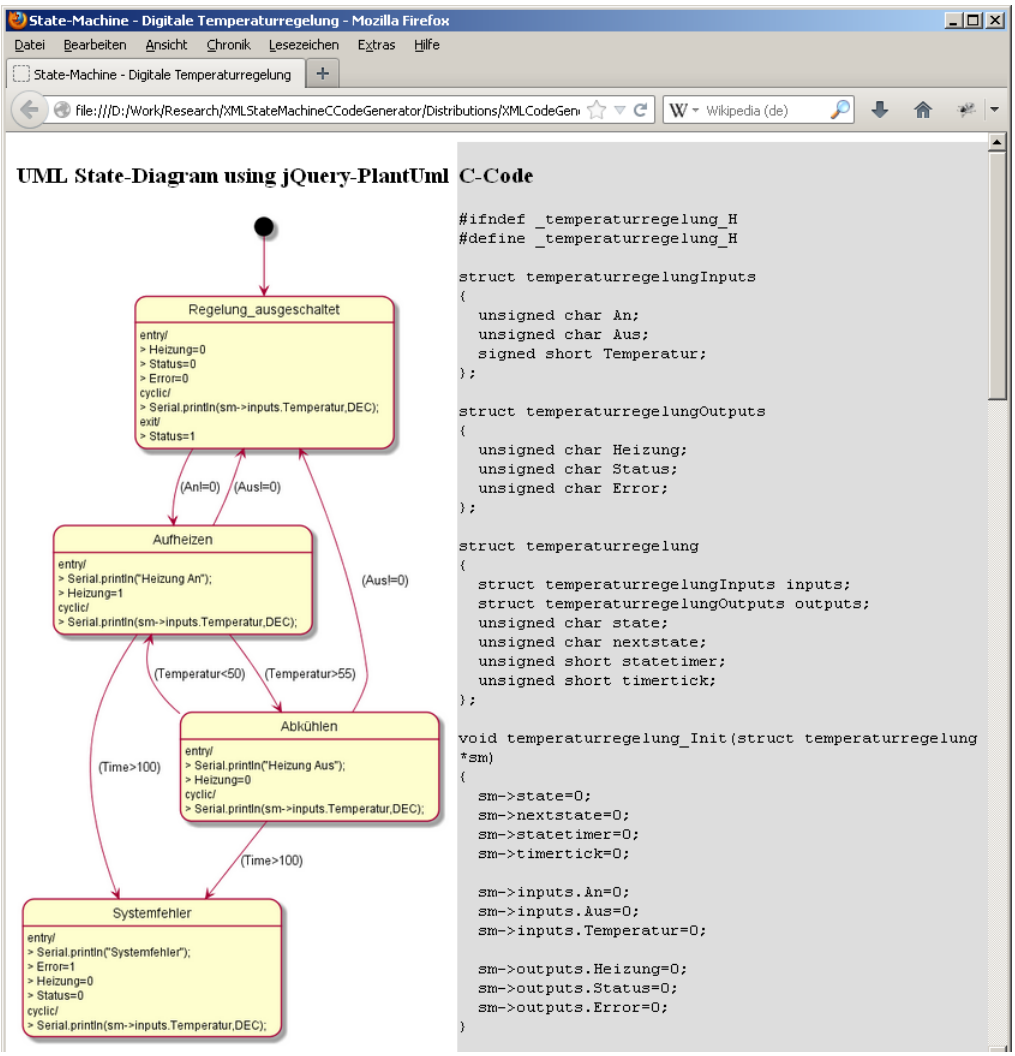


Abb. 6: UML State Diagramm des endlichen Automaten mit Hilfe von XML, XSLT und PlantUML automatisch erzeugt

Schlusswort

Dieser Artikel hat das Zusammenspiel von XML und XSLT anhand eines endlichen Automaten demonstriert und wurde dazu verwendet den zugehörigen C-Code automatisch zu generieren. Die Vorteile automatischer Code-Generierung liegen darin, dass bei häufig wiederkehrenden gleich strukturierten Aufgaben die Fehler des C-Codes dramatisch reduziert und somit dessen Qualität gesteigert werden können (z.B: nie wieder vergessene break-Anweisungen in einem switch-case Konstrukt).

XSLT ist für einen C-Programmierer anfangs ungewohnt. So gibt es zwar beispielsweise Variablen, welche aber nur einmal gesetzt und nie mehr geändert werden können. Rekursion wird gerne verwendet, um kompliziertere Aufgaben lösen zu können. Auch zählt die Mathematik, abgesehen von den vier Grundrechnungsarten, nicht zu den Stärken von XSLT. XSLT kann dafür bei komplizierten Abhängigkeiten von Daten, ähnlich wie bei rationalen Datenbanken, mit Hilfe von XPath seine Stärke demonstrieren.

Download / Literatur / Links

Die Dateien zu diesem Artikel können unter http://sourceforge.net/projects/bioe/files/others/XMLCodeGenerationTutorial/ als ZIP-Datei mit der Bezeichnung XMLCodeGenerationTutorial_13.05.zip heruntergeladen werden. Es ist auch geplant, dass auf der Projekteseite http://bioe.sourceforge.net ein Artikel zu diesem Thema erstellt wird.

[1] Embedded Projects Journal - Implementierung einer Finite State Machine; Philip Kälin; Issue 13 - July 2012

[2] Internet, Arduino-Framework Web-Page (Stand 30.04.2013) - http://www.arduino.cc/

[3] Internet, XML Tutorial (Stand 30.04.2013) - http://www.w3schools.com/xml

[4] Internet, DTD Tutorial (Stand 30.04.2013) - http://www.w3schools.com/dtd/

[5] Internet, XML Schema Tutorial (Stand 30.04.2013) - http://www.w3schools.com/schema/

[6] Internet, XSLT Tutorial (Stand 30.04.2013) - http://www.w3schools.com/xsl/

[7] Internet, XPath Tutorial (Stand 30.04.2013) - http://www.w3schools.com/xpath

[8] Internet, Command Line XSLT Processor (Stand 30.04.2013) - http://xmlsoft.org/XSLT/xsltproc.html

[9] Internet, Fritzing-Framework Web-Page (Stand 30.04.2013) - http://fritzing.org/

[10] Internet, KiCad EDA Software Web-Page (Stand 30.04.2013) - http://www.kicad-pcb.org

[11] Internet, PlantUML State-Diagram Web-Page (Stand 30.04.2013) - http://plantuml.sourceforge.net/state.html

Lesen Sie die neue Elektor ein Jahr lang in der ultimativen GOLD-Mitgliedschaft und profitieren Sie von allen Premium-Vorteilen!

Die Elektor-GOLD-Jahresmitgliedschaft bietet Ihnen folgende Leistungen/Vorteile:

- Sie erhalten **10 Elektor-Hefte** (8 Einzelhefte + 2 Doppelausgaben Januar/Februar und Juli/August) pünktlich und zuverlässig frei Haus.
- **Extra:** Jedes Heft steht Ihnen außerdem als PDF zum sofortigen Download unter www.elektor-magazine.de (für PC/Notebook) oder via App (für Tablet) bereit.
- **Neu & Exklusiv:** Sie erhalten alle 2 Wochen per E-Mail ein neues Extra-Schaltungsprojekt (frisch aus dem Elektor-Labor).
- **Neu & Exklusiv:** Wir gewähren Ihnen bei jeder Online-Bestellung 10% Rabatt auf alle unsere Webshop-Produkte – dauerhaft!
- **Neu & Exklusiv:** Der Online-Zugang zum neuen Community-Bereich www.elektor-labs.com bietet Ihnen zusätzliche Bauprojekte und Schaltungsideen.
- **Extra:** Die neue Elektor-Jahrgangs-DVD (Wert: 27,50 €) ist bereits im Mitgliedsbeitrag inbegriffen. Diese DVD schicken wir Ihnen sofort nach Erscheinen automatisch zu.
- **Extra:** Top-Wunschprämie (im Wert von 30 €) gibts als Dankeschön GRATIS obendrauf!

elektor GOLD membership

UMWELTSCHONEND – GÜNSTIG – GREEN

Möchten Sie Elektor lieber im elektronischen Format beziehen? Dann ist die neue GREEN-Mitgliedschaft ideal für Sie! Die GREEN-Mitgliedschaft bietet (abgesehen von den 10 Printausgaben) alle Leistungen und Vorteile der GOLD-Mitgliedschaft.

elektor GREEN membership

Jetzt Mitglied werden unter www.elektor.de/mitglied!

Ausgangsspannung ohne Verluste gefiltert

Stefan Klein <Stefan.Klein@we-online.de>

Problemstellung: Schaltregler weisen ausgangsseitig eine Restwelligkeit in ihrer Ausgangsspannung auf, welche nachfolgende Baugruppen stören und zu elektromagnetischen Beeinflussungen führen können. Zur Störunterdrückung werden daher häufig Ausgangsfilter verwendet, welche unter Umständen Einfluss auf die Regelschleife ausüben können. Um Verluste in der Ausgangsleistung zu verhindern, kann dabei eine Kompensation der Regelschleife erforderlich werden.

Einleitung

Gleichgültig welche Schaltreglertopologie im Einsatz ist, der Ausgangsstrom verursacht durch den parasitären Serienwiderstand ESR und der parasitären Induktivität ESL des Ausgangskondensators eine unerwünschte Restwelligkeit. In Abhängigkeit des gewählten Kondensatortypen entsteht so eine relativ große Restwelligkeit und weist unterschiedliche Kurvenformen auf. So zeigt beispielsweise ein gängiger Elektrolytkondensator, je nach Ausgangsleistung des Schaltreglers, eine Rippel-Spannung bis zu einigen hundert Millivolt. Bei der Wahl eines Keramikkondensators kann die Rippel-Spannung nur noch einige zehn Millivolt betragen.

Hohe Restwelligkeit ist unerwünscht und kann nachfolgende elektronische Baugruppen stören. Speziell analoge und HF-Schaltkreise erfordern eine stabile, geglättete und saubere Versorgungsspannung. Nicht zu vernachlässigen ist allerdings auch der hochfrequente Anteil an harmonischen Oberschwingungen der Ausgangsspannung, welche zur erhöhten elektromagnetischen Störaussendung führen kann. Ein Ausgangsfilter kann an dieser Stelle die Restwelligkeit reduzieren und hochfrequente Anteile herausfiltern.

Um die Restwelligkeit zu reduzieren, wird in der Praxis üblicherweise auf LC-Tiefpassfilter zurückgegriffen. Ist eine besonders saubere Ausgangsspannung erforderlich, wird der LC- Tiefpassfilter um einen nachfolgenden Tiefpass aus einem Ferrit und einem Kondensator erweitert. Abbildung 1 zeigt einen solchen Zweistufen Ausgangsfilter, welcher kostengünstig beispielweise mit einer Spule WE-PD2 und einem SMD-Ferrit wie z. B. WE-MPSB [2] realisiert werden kann.

LFilter und CFilter1 fungieren als Tiefpassfilter, welche die Taktfrequenz des Schaltreglers ausfiltern und seine harmonischen Oberschwingungen dämpfen. Weitere hochfrequente Anteile der Ausgangsspannung des Schaltreglers werden durch den SMD-Ferrit in Wärme umgewandelt und zusammen mit Cfilter2 in ihrer Amplitude gedämpft. Solch ein einfacher Ausgangsfilter reduziert die Restwelligkeit auf wenige Millivolt und ermöglicht sogar die Versorgung von funkbetriebenen Schaltungen.

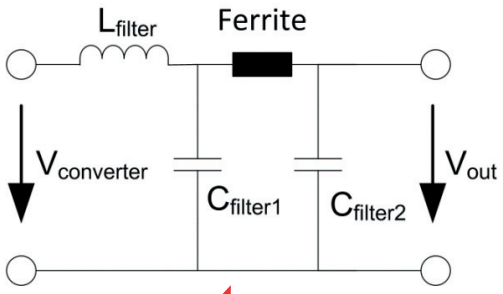


Abb. 1: Zweistufen Ausgangsfilter

Hauptteil

Ab einer gewissen Ausgangsleistung des Schaltreglers führt der Ausgangsfilter zu großen DC-Verlusten der Ausgangsleistung und somit zur Reduzierung des Wirkungsgrads des Schaltreglers. Der Gleichstromwiderstand RDC der Spulen und Ferrite erzeugt nun einen signifikanten Spannungsabfall über dem Ausgangsfilter und bewirkt dadurch eine Reduzierung der resultierenden Ausgangsspannung. Je nach Abhängigkeit der Bauform der Spule, kann der RDC zwischen wenigen Milliohm bis einigen Ohm betragen und ist daher bei hohen Ausgangsströmen nicht vernachlässigbar. Selbst ein SMD-Hochstromferrit kann einen RDC von bis zu 0,04Ω aufweisen.

Zur Ermittlung der Ist-Spannung wird bei Schaltreglern die Ausgangsspannung über einen Spannungsteiler abgegriffen und an den Feedback-Anschluss des Schaltregler-IC's geführt. Um Verluste der Ausgangsspannung durch einen Ausgangsfilter zu reduzieren, besteht die Möglichkeit den Ausgangsfilter in die Regelschleife zu implementieren, indem der Ist-Wert nach dem Ausgangsfilter ermittelt wird. Abbildung 2 zeigt die schematische Anordnung dieses Verfahrens.

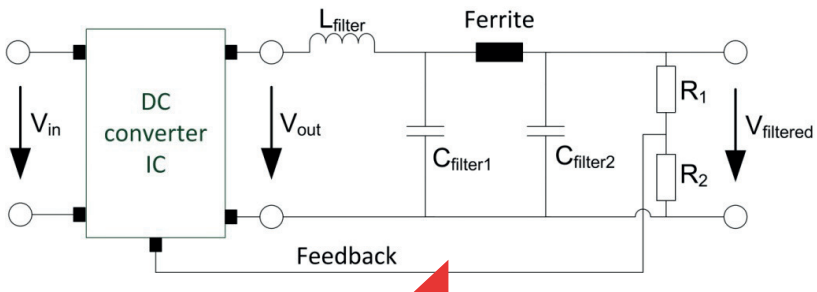


Abb. 2: Implementierung des Ausgangsfilters in den Regelkreis

Zu Abb. 2: Implementierung des Ausgangsfilters in den Regelkreis; Filterspule, Ferrit und die Filterkondensatoren führen jedoch zu einer unerwünschten Phasenverschiebung, wodurch die Stabilität der Regelschleife gestört wird. Diese unerwünschte Phasenverschiebung führt zur Verkleinerung der Amplituden- und Phasenreserve. Im Extremfall führt es zu Instabilität und die Ausgangsspannung neigt zum Schwingen. Um die Stabilität zu gewährleisten, wird in der Praxis eine Amplitudenreserve von >12dB und eine Phasenreserve von >45° verlangt, damit der Regelkreis bei Erregung nicht zum Schwingen neigt. Als dynamisch stabil gilt der Regelkreis, wenn die Schleifenverstärkung auf 0dB fällt, bevor die zugehörige Phasenverschiebung den Wert von -180° erreicht hat. Dabei soll der Amplitudengang der Schleifenverstärkung den Schnitt der X-Achse, also bei 0dB mit 20dB/Dekade durchlaufen.

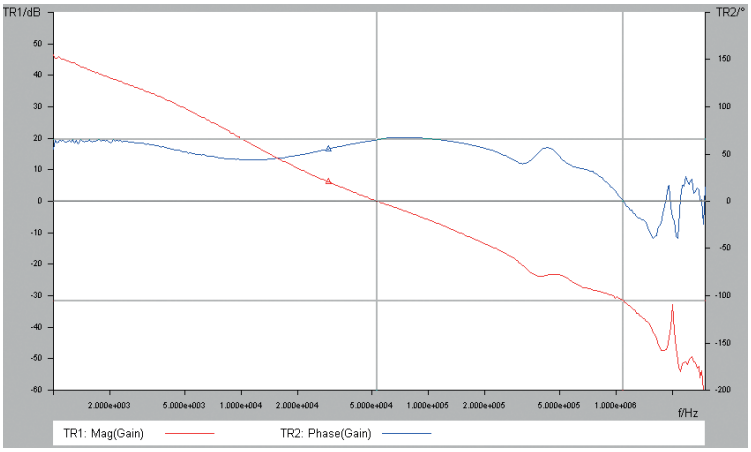


Abb. 3: Bode-Plot eines stabil-geregelten Schaltreglers

Abbildung 3 zeigt ein Bode-Plot eines stabil-geregelten Buck-Konverters. Dieses Beispiel zeigt eine Amplitudenreserve von 32dB und eine Phasenreserve von 56°.

Sind die Stabilitätskriterien eines Schaltreglers mit einem Ausgangsfilter nicht erfüllt, so ist eine Kompensation der Regelschleife erforderlich, um eine stabile Ausgangsspannung zu gewährleisten. Die Stabilität der Regelstrecke übt somit Einfluss auf die Stabilität der Ausgangsspannung aus. Bei einer Spannungsschwankung am Eingang des Schaltreglers soll die Ausgangsspannung stabil bleiben. Analog hierzu soll bei einem schlagartigen Abfall oder Anstieg des Ausgangsstroms, die Ausgangsspannung schnell nachgeregelt werden. Hierbei ist die Rede vom Transient Response. Abbildung 4 zeigt den Transient Response eines Stabil-geregelten Schaltreglers (gelbe Kurve) bei einer Ausgangsspannung von 5V und einem schlagartigen Lastwechsel von 0A auf 1A (grüne Kurve).

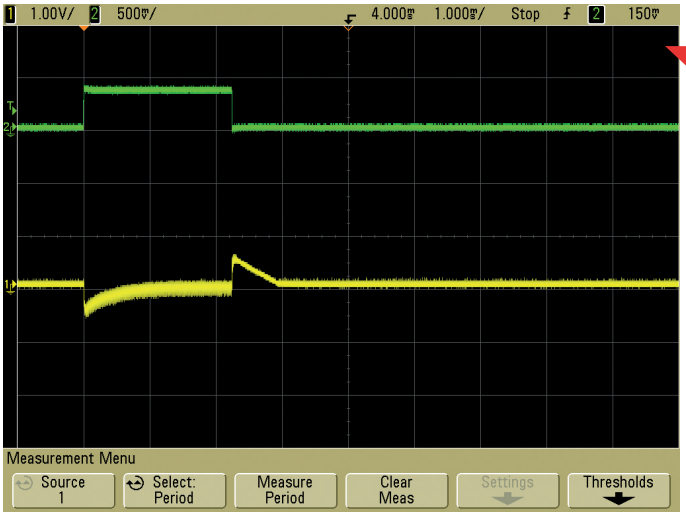


Abb. 4: Transient Response eines stabil-geregelten Schaltreglers

Eine schlagartige Laständerung soll zu einer schnellen Sprungantwort des Regelkreises führen und die Ausgangsspannung zeitig auf seinen Soll-Wert regeln. Dabei soll die Sprungantwort keine zu große Amplitude in der Ausgangsspannung bewirken, da sonst nachfolgende Baugruppen bei einer Spannungsüberhöhung zerstört werden könnten. Idealerweise soll die Ausgangsspannung nach der Sprungantwort zeitig auf den Soll-Wert geregelt werden, ohne dabei ein Überschwingen zu erzeugen oder gar zu klingeln. Ein Klingeln während der Ausgleichsphase wäre somit auf eine Instabilität des Schaltreglers zurückzuführen. Ist eine schnelle Sprungantwort und eine zeitige Ausgleichsphase gegeben, so kann der Schaltregler als stabil-geregelt gelten.

Resümee und Abschluss

Wird der Ausgangsfilter in die Regelstrecke implementiert entsteht eine Regelstrecke 2. Ordnung. Der Schaltregler muss daher mit einem höheren Integrationsanteil betrieben werden, wodurch die Regelstrecke gedämpft und somit langsamer wird. Eine umständliche Kompensation der Regelstrecke wird nun erforderlich. Von dem Verfahren, der Implementierung des Ausgangsfilters in die Regelstrecke, wird daher abgeraten. Ein Abgriff der Ausgangsspannung des Schaltreglers sollte unmittelbar am Ausgangskondensator des Schaltreglers erfolgen, somit vor einem Ausgangsfilter. Um DC-Verluste des Ausgangsfilters zu reduzieren wird empfohlen Filterspulen und Ferrite mit möglichstem kleinem RDC zu wählen.

Links / Kontakt

[1] Stefan Klein, Application Engineer, Stefan.Klein@we-online.de

[2] Würth Elektronik eiSos GmbH & Co. KG : http://www.we-online.de

Willkommen an Bord

Die Würth Elektronik Gruppe fertigt und vertreibt Leiterplatten, elektronische Bauelemente und intelligente Systeme. Würth Elektronik ist mit 6000 Mitarbeitern in knapp 50 Ländern eines der erfolgreichsten Unternehmen innerhalb der Würth Gruppe.

WÜRTH ELEKTRONIK

Erfrischung gefällig? Wasser marsch!

Bachelorprogramm

Technischer Vertrieb | Produktentwicklung | Qualitätsmanagement

Direkteinsteig

Produktentwicklung | Forschung und Entwicklung
Qualitätsmanagement | Technischer Vertrieb

Masterprogramm

Produktentwicklung

Direkteinstieg

Produktentwicklung | Qualitätsmanagement | Verwaltung
Informationstechnologie | Forschung und Entwicklung

www.we-online.de/karriere

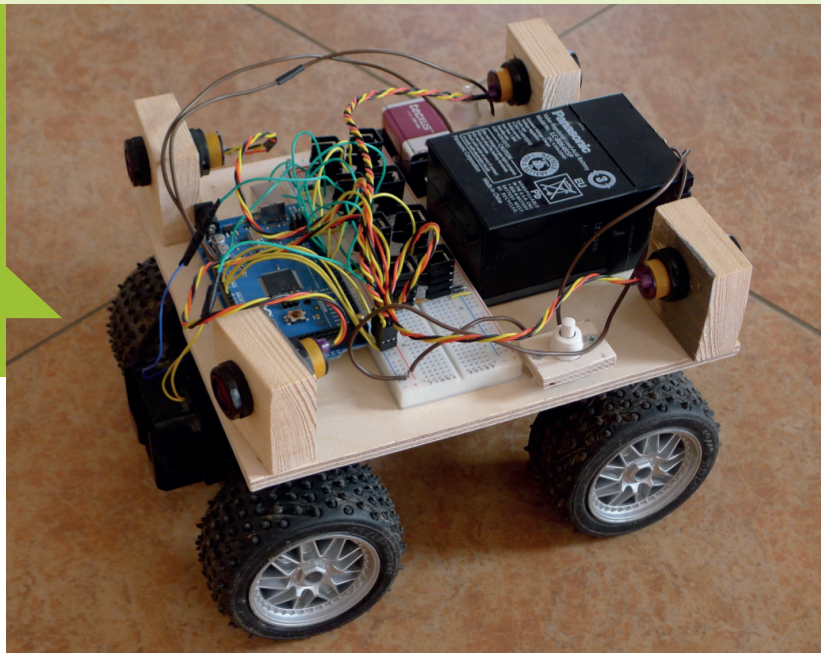
more than you expect

„Arduino Car“ –

Mein erstes embedded Projekt

Julian Sarcher <julian.sarcher@googlemail.com>

Was macht man mit einem alten RC-Car aus Kindertagen? Ausschlachten und zu einem „autonomen Roboter“ umbauen war mein Plan. Die Zeit dafür schien reif zu sein. Ich habe gerade mein erstes Semester Technische Informatik an der Hochschule Augsburg hinter mir und ein paar freie Wochen zum Basteln vor mir. Einige Tage und Nächte später und bestückt mit 4 Abstandssensoren, einem Arduino Board und etwas Code schnurrt jetzt das RC-Car (Abb. 1 rechts) durchs ganze Haus. Dies ist eine kleine Anleitung für alle die noch fahrbare Restbestände im Keller oder Dachboden bunkern.



Aufbau

Das RC-Car habe ich bis auf Motor und Lenkung ausgeschlachtet. Ein aufgeschraubtes Sperrholzbrett dient als Halterung für Arduino, Breadboard, Sensoren und Stromversorgung. Die Sensoren sind drehbar montiert, so dass genügend Raum für Experimente bezüglich der Erkennung von nahenden Wänden bleibt.

Einlesen der Sensoren

Um Wände, welche dem Roboter bei seiner autonomen Fahrt gefährlich werden könnten, zu erkennen, habe ich vier Photoelectric Switch Sensoren verwendet. Die vier Sensoren sind an den Ecken des Roboters angebracht. Dabei sind zwei nach vorne und zwei nach hinten gerichtet. Die Ansteuerung der Sensoren ist simpel. Rotes Kabel an Plus, schwarzes Kabel an GND und die gelbe Steuerleitung gibt als Ausgangssignal HIGH zurück, wenn kein Hindernis erkannt wird und LOW, wenn der Switch Sensor anschlägt. Letzteres ist bei den verwendeten Sensoren gut an der aufleuchtenden LED zu erkennen.

Ansteuerung von Antrieb und Lenkung

Gesteuert werden müssen ein einfacher DC Motor für den Antrieb und ein Weiterer für die Lenkung. Dies geschieht für beide über je eine H-Bridge. Um eine H-Bridge zu realisieren kann auf fertige IC's zurückgegriffen werden oder man baut sich die Schaltung, wie in meinem Fall, selbst auf. Die aufgebaute H-Bridge besteht aus zwei UF630 N-Kanal MOSFET's und zwei IRF9630 PKanal MOSFET's. Nicht zu vergessen ist je eine Freilaufdiode, welche bei einer induktiven Last stets zu verwenden ist und den MOSFET beim Abschalten der Last schützt. Zur Steuerung wird ein ATMEGA2560 auf einem Arduino Board verwendet (Abb. 2).

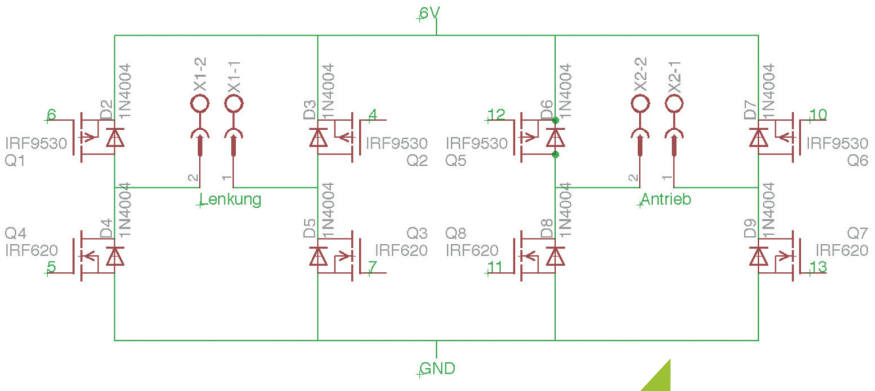


Abb. 2: Aufbau der beiden H-Brücken für Antrieb und Lenkung

Stromversorgung

Die optimale Eingangspannung für das Arduino Board liegt zwischen 7V und 12V. Die anzusteuernden DC-Motoren benötigen eine Betriebsspannung von 6V. Somit habe ich mich entschieden, die vergleichsweise kleinen Ströme für das Board, von den Motoren, welche unter Last bei einer Spannung von 6V bis zu 1,5A ziehen können, zu trennen. Die Stromversorgung für das Board und somit für den ATMEGA2560 und Bereitstellung der Gate Spannung geschieht über einen 9V Block. Die Stromversorgung der Motoren und Sensoren wird über einen 6V Blei-Vlies Akku gewährleistet. Kennt man sich jedoch mit dem Laden von Blei-Vlies Akkumulatoren nicht sonderlich aus, sollte man die Finger davon lassen und lieber vier 1,5V Mono Akkumulatoren in Reihe schalten.

Ansteuerungslogik

Die Ansteuerungslogik erfolgt über die Signale der Sensoren, welche die Zustände bestimmen. Ein Zustand ist zum Beispiel: Geradeaus fahren. Die Zustände werden in einer Funktion sensorRead() bestimmt, welche die Sensoren ausließt und die Ansteuerlogik kennt. Bei der Erstellung der Ansteuerlogiktabelle ist dabei darauf zu achten, dass jede mögliche Variation der Zustände der Sensoren abgedeckt ist. Bei vier Switch Sensoren gibt es also $2^4 = 16$ Möglichkeiten. Eine Möglichkeit ist, dass gar kein Sensor etwas detektiert, dann soll das Auto geradeaus fahren:

```
case 1 : // Vorwärts
      digitalWrite(forwardPin1, LOW);
      digitalWrite(forwardPin2, HIGH);
      while (sensorRead() == 1) delay(20);
      digitalWrite(forwardPin1, HIGH);
      digitalWrite(forwardPin2, LOW);
      break;
```

Im Code oben werden mit digitalWrite() die MOSFET's, welche für das vorwärts fahren gedacht sind, geschalten. Mit LOW bzw. HIGH am Gate des P-Kanal bzw. N-Kanal MOSFET's werden diese leitend und der Antriebsmotor dreht sich, umgekehrt sperren diese und der Antriebsmotor dreht nicht mehr. Er fährt also so lange vorwärts, bis ein anderer Zustand erreicht wird. Nähert sich das Auto einer Wand, kann man es zum Beispiel so programmieren, dass wenn ein Signal vorne rechts anliegt, soll es so lange rechts rückwärts fahren, bis ein anderer Zustand bestimmt wird und danach noch eine Sekunde weiter, aber nur wenn kein weiteres Hindernis hinter ihm ist. Jetzt ranschiert das Auto ein-zweimal hin und her und die Kurve ist geschafft und es geht weiter bis zum nächsten Hindernis.

```
case 6 : // Rückwärts und nach rechts
      Serial.print(„case 6“);
      digitalWrite(rightPin1, LOW);
      digitalWrite(rightPin2, HIGH);
      digitalWrite(backwardPin1, LOW);
      digitalWrite(backwardPin2, HIGH);
      while (sensorRead() == 6) {
            delay(20);
      }
      turnAround();
      digitalWrite(rightPin1, HIGH);
      digitalWrite(rightPin2, LOW);
      digitalWrite(backwardPin1, HIGH);
      digitalWrite(backwardPin2, LOW);
      break;

void turnAround() {
   time = millis();
   while((millis() - time < 1000) && sensor-
Value3 > 500 && sensorValue4 > 500) {
      delay(20);
      sensorRead();
   }
}
```

Sind alle 16 möglichen Sensorkombinationen nach obigem Schema programmiert, kann das Arduino Car zu seiner ersten Testfahrt ausgesetzt werden. Je nachdem wie schnell das Auto unterwegs ist, muss man vor Hindernissen früher oder später den Rückwärtsgang einlegen. Unterschiedliche Anstellwinkel der Sensoren führen ebenfalls zu einem veränderten Fahrverhalten.

Fazit

Man kann ohne große Probleme aus einem ausrangierten RC-Car selbst einen funktionsfähigen „autonomen Roboter“ bauen. Spaß bereitet zudem auch die weitere Programmierung des Roboters, da die Möglichkeiten dank des Mikrocontrollers nahezu unbegrenzt sind. So kann man die Funktionsweise des Roboters immer weiter verfeinern, indem man ihm zum Beispiel ein Gedächtnis über die letzten vergangenen Zustände verschafft. Weitere Informationen über Motoransteuerung und den Link zum Video finden Sie in den unten genannten Links und Quellen.

Links und Quellen

[1] Bilder vom „Arduino Car“: http://www.sarcher.de/Julian/Arduino-Car/Bilder.pdf

[2] Video zum „Arduino Car“: http://youtu.be/mB5lH24j4TE

[3] Motoransteuerung: http://www.netzmafia.de/skripten/rechnerperipherie/ein_ausgabe.pdf

Marktplatz / Neuigkeiten

Die Ecke für Neuigkeiten und verschiedene Produkte

Open-Source Funknetzwerk für Heimautomatisierung & Co

Das Merkur Board ist ein Arduino und Contiki-OS kompatibles Mikrocontrollerboard mit integriertem 2.4Ghz IEEE802.15.4 Funksender. Der Arduino-Bootloader ist vorprogrammiert. Um Mini-Funkmodule auch als Mensch in den Griff zu bekommen haben wir dieses Board geschaffen. Der Funkteil unterstützt alle auf 802.15.4 basierenden Funkprotokolle wie 6LoWPAN, Zigbee und RF4ce.

http://www.osdomotics.com/

Rechenkraft.net e.V. - Mitarbeit gesucht

Engagierte Studenten für interessante Projekte aus den Bereichen Informatik, Marketing, Design, Medien und Kommunikation gesucht! Wir sind ein gemeinnütziger Verein, der sich mit Distributed Computing beschäftigt.

Warum Distributed Computing?

Mit Distributed Computing (Verteiltes Rechnen) kannst du Dich mit deinem Computer an Forschungsarbeiten z.B. in der Klimaforschung, Astronomie, Medikamentensuche, uvm. beteiligen.

Wie kannst Du helfen?

Um an der spannenden Welt des Distributed Computing teilzunehmen und die ungenutzte Rechenleistung seines Computers für wissenschaftliche Zwecke einzusetzen, kann man Software installieren, die im Hintergrund wissenschaftliche Berechnungen durchführt. Es gibt auch Projekte, bei denen - ähnlich wie bei einem Computerspiel - die wissenschaftlichen Analysen von Dir selbst durchgeführt werden können. Wir haben inzwischen selbst zwei eigene Projekte am Laufen und suchen Interssierte, die auf Freiwilligenbasis dabei helfen wollen, unseren Verein auszubauen und unsere Projekte weiterzuentwickeln.

Weitere Infos:
http://www.rechenkraft.net/

Kooperation mit Elektor International Media BV

Wir freuen uns gemeinsam mit Elektor eine Kooperation abgeschlossen zu haben. Gemeinsam möchten wir mehr Menschen innerhalb und außerhalb Deutschlands für die Elektronik begeistern.

Die embedded projects GmbH (www.embedded-projects.net) entwickelt und produziert mit einem jungen Team aus Informatikern und Ingenieuren seit 2009 eingebettete Mess-, Steuer- und Regellösungen – vollständig webbasiert und kundenspezifisch. Das junge Unternehmen steht für ein innovatives, zukunftsweisendes – dabei praxisnahes und bezahlbares – Produktportfolio innerhalb der Linux-Welt. Auch die Verantwortlichen der Publikation Elektor (www.elektor.de) – als europaweit führende Elektronik-Fachzeitschrift in acht Sprachen und mehr als 50 Ländern erhältlich – wurden schnell auf das große Potenzial der Augsburger IT-Schmiede aufmerksam. So arbeitete man bereits in der Vergangenheit immer wieder mit dem embedded projects-Team redaktionell zusammen und bot einzelne Produkte im eigenen Online-Shop an. Um diese lose Zusammenarbeit zukünftig auf eine feste Grundlage zu stellen und weiter auszubauen, unterzeichneten nun die beiden Geschäftsführer Benedikt Sauter und Don Akkermans einen entsprechenden Kooperationsvertrag.

Ebenso wird Elektor – im Zuge eines kontinuierlichen Informationsaustausches – regelmäßig über neue Projekte und Entwicklungen des Augsburger Unternehmens und der um sie entstandenen Community berichten.

Des Weiteren ist die Entwicklung sowie der weltweite Direkt- oder Reseller-Vertrieb gemeinsam erarbeiteter Produkte geplant. Um auch Kindern und Jugendlichen das Thema spielerisch näher zu bringen, ist hierbei auch die Verwirklichung entsprechender Projekte konzertiertes Bestreben. Sämtliche neuen Technologien sind dann zukünftig auch über den Elektor-Webshop international beziehbar. Darüber hinaus werden gemeinsam veranstaltete Lehrgänge, Workshops, Seminare und Webinare zu unterschiedlichen Embedded Linux-Thematiken stattfinden.

Wir freuen uns auf eine gemeinsame Zukunft!

Interesse an einer Anzeige?

info@embedded-projects.net

embedded - projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

Falls Du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe deine Idee an:

journal@embedded-projects.net

Regelmäßig

Die Zeitschrift wird über mehrere Kanäle verteilt. Der erste Kanal ist der Download als PDF - Datei. Alle Ausgaben sind auf der Internetseite [1] verfügbar. Diejenigen, die lieber eine Papierversion erhalten möchten, können den zweiten Kanal wählen. Man kann sich dort auf einer Internetseite [2] in eine Liste für die gesponserten Abos oder ein Spendenabo eintragen. Beim Erscheinen einer neuen Ausgabe wird dank Sponsorengeldern an jeden auf der Liste eine Ausgabe des aktuellen Journal versendet. Falls man den Versandtermin verpasst hat, kann man das Heft auch über einen Online - Shop [2] beziehen.

[1] Internetseite (Anmeldeformular gesponserte Abos): http://journal.embedded-projects.net

[2] Online - Shop für Journal:
http://www.embedded-projects.net

Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821 / 2795990 oder sendet eine E-Mail an die oben genannte Adresse.

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

◯ Ich möchte jede zukünftige Ausgabe erhalten

◯ Wir möchten als Hochschule / Ausbildungsbetrieb jede weitere Ausgabe bekommen. Bitte gewünschte Anzahl der Hefte pro Ausgabe ankreuzen. 5 10

◯ Ich möchten im embedded projects Journal werben oder eine Stellenanzeige aufgeben. Bitte schicken Sie mir Infomaterial, Preisliste etc. zu.

Impressum

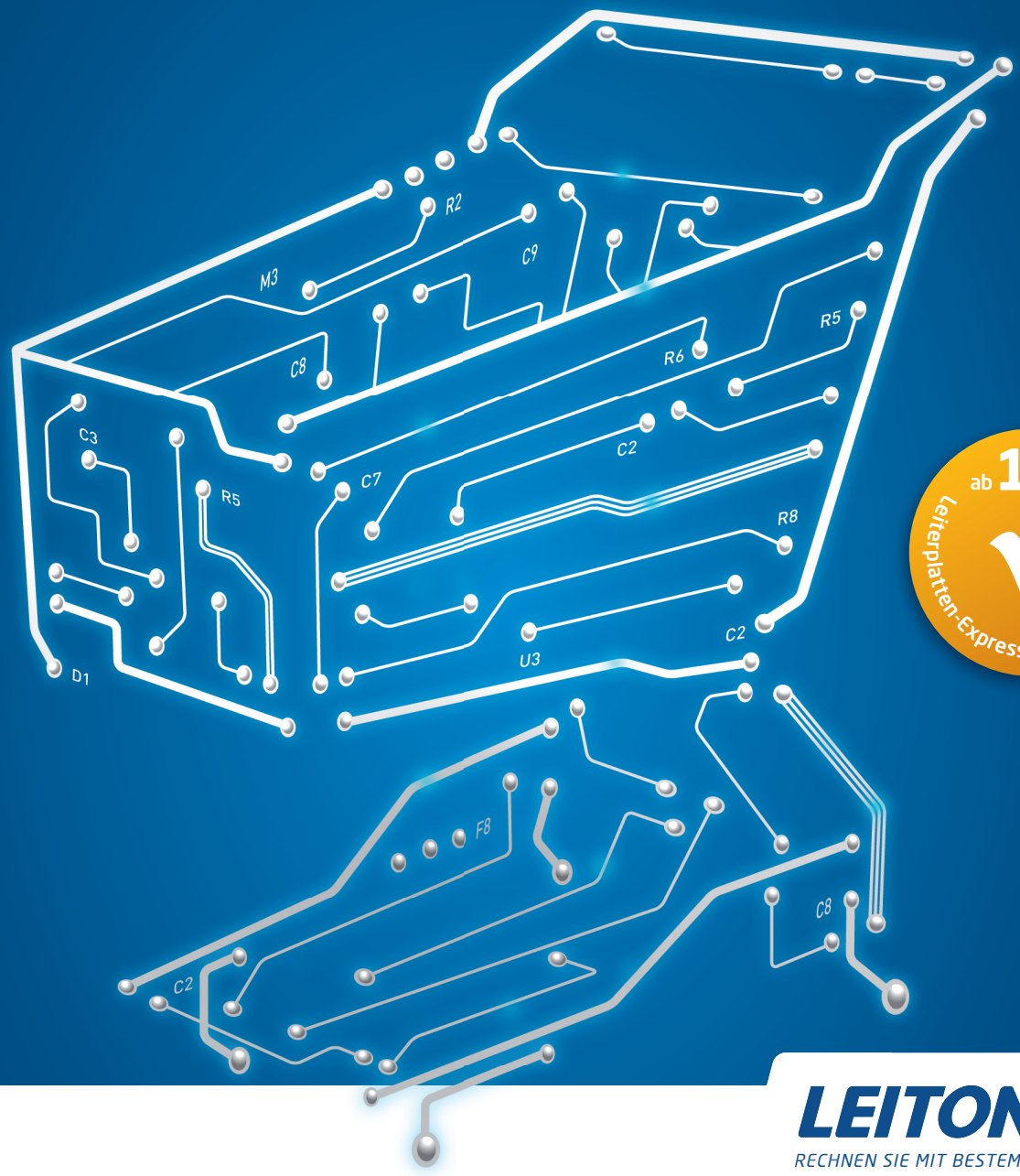
embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Telefon: +49(0)821 / 279599-0
Telefax: +49(0)821 / 279599-20

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print
Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelbild: Edith Högl

EINKAUFSSPASS AUCH FÜR MÄNNER.

GROSSES PRODUKTSPEKTRUM ONLINE KALKULIERBAR.



ab 12 h Leiterplatten-Express! Pünktlich oder gratis!

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Bei LeitOn Leiterplatten online kalkulieren und kaufen spart Nerven, Zeit und Geld. Stark: **Die Online-Kalkulation gilt bei uns auch für Aluminium- und flexible Leiterplatten.** Dazu kommt das große Produktspektrum: Multilayer mit bis zu 18 Lagen, FLEX sowie ALU-Leiterplatten, SMD-Schablonen, Starrflex-Leiterplatten und und und. Ein Plus an Service bietet der **Leiterplatten-Expressdienst** von LeitOn. Unser Versprechen: Sollten wir bei einem Expressdienst den vereinbarten Übergabetermin an den Versender nicht einhalten, sind die Platinen für Sie gratis! Sie möchten mehr darüber wissen? Wir bieten persönliche Beratung am Telefon, einen kompetenten Außendienst und Angebote auch per E-Mail in Windeseile. Sie können bei LeitOn immer mit bestem Service rechnen.

LeitOn GmbH | **www.leiton.de** | **kontakt@leiton.de** | Info-Hotline **+49 (0)30 701 73 49 0**