c't magazin für computer technik

Mit einer Drehzahlregelung fährt der c't-Bot geradeaus und mit konstanter Geschwindigkeit

Steuermann

Praxis & Tipps | Praxis .. Uhr
Daniel Bachfeld

Vertrauen ist gut, Kontrolle ist besser: Ohne Regelung fährt der c't-Bot aufs Geratewohl durch die Gegend. Eine einfache Drehzahlregelung hält den Roboter in der Spur und die Drehzahl konstant.

Bislang führt der c't-Bot die ihm vorgegebenen Verhalten ohne eine echte Kontrolle aus. Selbst eine simple Geradeausfahrt kann bei gleichen PWM-Werten aufgrund der Fertigungstoleranz der Motoren in einer Kurvenfahrt resultieren. Aber nicht nur der Gleichlauf will kontrolliert werden. Durch sinkende Akkuspannung, variierendes Gewicht, Neigung und Beschaffenheit des Untergrunds ist es schwer vorauszusagen, mit welcher Geschwindigkeit der Roboter bei seinen Erkundungsfahrten das Wohnzimmer unsicher macht. Daher reicht es nicht, den Motoren einfach nur Stellwerte zu liefern, die Elektronik muss prüfen, ob sich auch die gewünschten Ergebnisse einstellen.

So muss der Prozessor ständig messen, ob die aktuellen Motorstellwerte zu den gewünschten Geschwindigkeiten der Räder führen und gegebenenfalls nachstellen. Hinkt die Drehzahl etwa hinterher, muss der Motor etwas schneller laufen. Diesen Kreis aus Messen und Nachstellen nennt man landläufig auch Regelung. Leider wird fälschlicherweise der Begriff Steuerung oftmals synonym verwendet - einer Steuerung fehlt aber immer die zum Schließen des Kreises notwendige Rückkopplung.

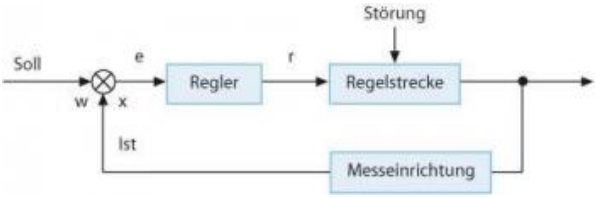
An und aus

Die simpelste Art der Regelung ist die On-off-Regelung, die beim Überschreiten eines Ist-Wertes einfach den Motor abschaltet. Sinkt der Ist-Wert (meist X genannt) unter den Soll-Wert (W), so springt der Motor wieder an. Um ein Flattern der Regelung zu verhindern, definiert man unterschiedliche Anschalt- und Abschaltepunkte (Hysterese). Für viele Systeme, etwa einfache Heizungsregelungen, reicht diese Art der Regelung völlig aus. Für eine Drehzahlsteuerung ist sie jedoch gänzlich ungeeignet, da sich mit ihr Fehler nicht wirklich gut eliminieren lassen, eine bestimmte Abweichung bleibt immer. Zudem würde der c't-Bot einen ziemlichen Eiertanz aufführen.

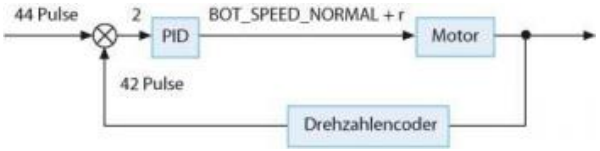
Statt der unstetigen On-off-Regelung setzen Entwickler deshalb auf Proportionalregler in Verknüpfung mit Integral- und Differenzialreglern. Beim Proportionalregler, auch P-Regler genannt,

stehen der Fehler e und die Stellgröße in einem linearen Zusammenhang, nämlich über den Verstärkungsfaktor K_P. Damit lässt sich einstellen, wie schnell und stark der Regler der Abweichung entgegenwirkt: $R_P(t) = K_P \cdot e(t)$. Zu hohe Werte von K_P lassen den Regler allerdings überschwingen oder bringen ihn in einen instabilen Zustand. Nicht selten fängt er dann auch kontinuierlich zu schwingen an. Leider verbleibt auch beim Proportionalregler immer ein gewisser Fehler. Zudem ist er vergleichsweise langsam.

Für eine schnellere Regelung, ohne gleich instabil zu werden, sorgt der PD-Regler. Hier differenziert zusätzlich ein D-Regler (Differenzial) die Regelabweichung und lässt das Ergebnis in den Stellwert mit einfließen. Wie stark er das tut, bestimmt der Wert K_D: $R_D(t) = K_D \cdot de(t)/dt$. So sorgt der PD-Regler für eine schnellere Reaktion auf Abweichungen vom Sollwert. Leider schafft er es aber ebenfalls nicht, die Abweichung vollständig zu eliminieren.



Ein Regelkreis ermittelt aus dem Soll- und Ist-Wert die Regelabweichung und berechnet daraus den neuen Stellwert.

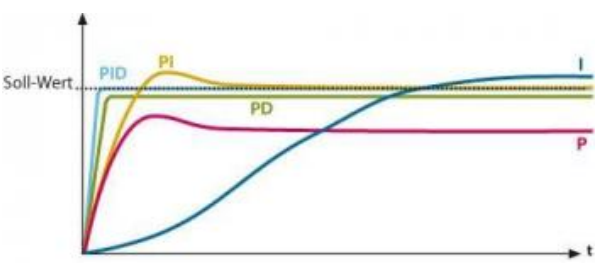


Der digitale Regelkreis des c't-Bot besteht aus Mikrocontroller, Motorsteuerung, Motor und Drehzahlencodern.

Um dies zu erreichen, fügt man dem PD-Regler eine weitere Komponente hinzu: einen integrierenden Regler. Der summiert die Regelabweichungen über die Zeit, multipliziert sie mit dem Faktor K_I und addiert sie zu den Werten des PD-Reglers. Je länger eine Abweichung vom Ist- zum Sollwert andauert, desto größer wird der Anteil des I-Reglers im resultierenden Stellwert: $R_I(t) = K_I \cdot e(t) \cdot dt$. Zusammengebaut erhält man dann einen kontinuierlichen PID-Regler: $R(t) = K_P \cdot e(t) + K_D \cdot de(t)/dt + K_I \cdot \int e(t) \cdot dt$. Je nach Zielsystem reicht aber manchmal ein PD- oder PI-Regler aus, wenn die vorhandene Regelstrecke bereits I- oder D-Eigenschaften aufweist, etwa ein Motor durch seine differenzierende Induktivität. Im Falle des c't-Bot trifft dies aber nicht zu.

Ganz diskret

In der analogen Welt bauen Entwickler PID-Regler meist mit Operationsverstärkern auf, die zeitkontinuierlich arbeiten, also analog integrieren, differenzieren und multiplizieren. Beim c't-Bot stehen die Signale aber nur zeitdiskret zur Verfügung, beispielsweise als Impulse oder Spannung pro Zeiteinheit. Zudem muss man die erforderlichen Funktionen für die Differentiation und Integration per Algorithmus nachbilden. Insbesondere die Integration stellt einen Mikrocontroller vor ein Problem. Deshalb muss man das Integral durch numerische Methoden ausdrücken, etwa durch eine endliche Summe.



Durch Anwendung der Trapezregel und diverser Substituierungen [1] bleibt eine Formel übrig, die sich recht einfach in Software gießen lässt: $R_N = R_{N-1} + K_P \cdot (e_N - e_{N-1}) + K_I \cdot (e_N - e_{N-1})/2 + K_D \cdot (e_N - 2 \cdot e_{N-1} + e_{N-2})$. Der Index N bezeichnet hierbei den aktuellen Wert, N-1 den Wert der vorangegangenen Messung beziehungsweise

Der PID-Regler regelt am genauesten und schnellsten auf den gewünschten Soll-Wert ein.

Berechnung und so weiter.

Um nun die Drehzahlen beziehungsweise die Geschwindigkeit der Motoren des c't-Bot zu regeln, berechnet der Stellalgorithmus zu bestimmten Zeitpunkten aus den gegebenen Ist-Werten einen neuen Stell-Wert und steuert damit den Motor an. Damit die Regelung nicht zu langsam ist, darf der Zeitraum zwischen den Neuberechnungen nicht zu groß sein. Zu klein sollte man ihn allerdings auch nicht wählen, sonst wird die Regelung zu ungenau. Als Abtastzeitraum haben wir für den c't-Bot in unserer Beispielimplementierung 330 Millisekunden gewählt, die wir aus dem Interrupt von Timer2 ableiten (179 µs · 1860) [2]. Dann wird aus dem seit dem letzten Abtastzeitpunkt gezählten Radencoderpulsen und den Sollwerten die Regelabweichung ermittelt und ein neuer Stellwert unabhängig für jeden Motor berechnet.

Ruf mich auf!

Neben der Wahl der Parameter stellt sich noch die Frage, wie man die Regelung implementiert. Sie als Verhalten in den c't-Bot einzufügen erscheint zwar auf der ersten Blick elegant, allerdings werden dann die berechneten Stellwerte von den anderen Verhalten überschrieben. Besser ist es, den Regler direkt in der Motorsteuerung zu implementieren.

```
void speed_control (int16 left, int16 right){
    int16 rmp,lmp,desired_left, desired_right, err_l,err_r; int16 reg_l, reg_r;
    if (clock_motor_control >1860)
     {
        ...
        lmp = sensEncL - tmpL;
        rmp = sensEncR - tmpR;
        tmpR = sensEncR;
        tmpL = sensEncL;
        ...
        err_l = left - lmp;
        reg_l = reg_l_old + KP* (err_l - err_l_old);
        reg_l = reg_l + KI * (err_l + err_l_old)/2;
        reg_l = reg_l + KD * (err_l - 2 * err_l_old + err_l_old2);
        reg_l_old = reg_l;
        err_l_old2 = err_l_old;
        err_l_old = err_l;
        ...
        clock_motor_control = 0;
      }
  bot_motor(reg_l_old, reg_r_old);
}
```

Damit die Regelung zum Zuge kommt, wird in motor_set() aus motor.c statt bot_motor() die Funktion speed_control() aufgerufen. Die globale Variable clock_motor_control wird im Modul timer-low.c in der Interrupt-Routine SIG_OUTPUT_COMPARE2 alle 179 µs hochgezählt [2].

Mit welcher Geschwindigkeit der c't-Bot fahren soll, wird bislang nur über das Tastverhältnis der PWM vorgegeben. Allerdings ist das keine Geschwindigkeit im eigentlichen Sinne, sondern nur ein

Krücke, mit der aus verschiedenen PWM-Werten verschiedene Geschwindigkeiten resultieren. So ist BOT_SPEED_NORMAL nur eine willkürliche Festlegung für eine „gefühlte“ normale Geschwindigkeit. Zukünftig geben wir keine PWM-Werte mehr vor, sondern die gewünschte Geschwindigkeit. Welche die richtige ist, kann man entweder schätzen oder messen. Je nach Akkuspannung liegt sie ohne Regelung bei BOT_SPEED_NORMAL im Freilauf zwischen 40 und 50 Pulse pro 330 Millisekunden. Bei BOT_SPEED_SLOW kommt man auf Werte zwischen 23 und 28. Daraus leiten wir die Geschwindigkeit in mm/s ab, etwa so: 25 · 1/0.33s · d · π · 60 = 225.

```
#define BOT_SPEED_NORMAL 396;
#define BOT_SPEED_SLOW   225;
```

Im Regler berechnet man daraus die richtige Drehzahl, indem man die Geschwindigkeit durch 9 teilt (left/9). Je nachdem, ob der c't-Bot aufgebockt im Freilauf betrieben wird oder sein eigenes Gewicht übers Parkett wuchtet, variieren die Drehzahlen bei fehlender Regelung zu dem vorgegebenen Stellwert - ohne Last dreht es sich halt schneller. Mit Regelung arbeitet der c't-Bot völlig lastunabhängig und hält immer die vorgegebene Geschwindigkeit ein: genau das war der Sinn der Übung.

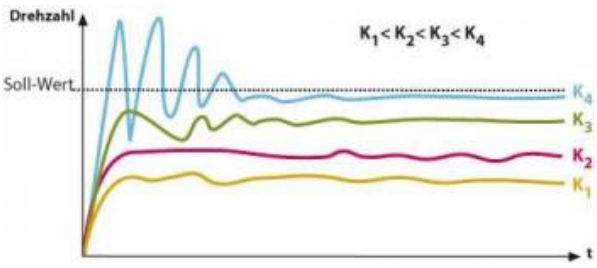
Zu beachten ist allerdings, dass bei sinkender Versorgungsspannung ebenfalls die Umdrehungsgeschwindigkeit sinkt, selbst wenn die Pulsbreitenmodulation konstant bleibt. Daher sollte der Programmierer beispielsweise die für BOT_SPEED_NORMAL definierte Drehzahl nicht zu hoch wählen, damit sie auch mit weniger Akkupower noch erreicht werden kann. Im Versuch lieferten die Radencoder des Motors bei vollen Akkus in einer Drittel Sekunde ungefähr 52 Pulse.

Eiertanz

Einer der schwierigsten Schritte bei der Implementierung des Reglers ist das Auswählen der richtigen PID-Parameter. Fuhr der c't-Bot ohne Regelung im schlimmsten Falle nur eine Kurve statt einer Geraden, so kann er mit einer falsch dimensionierten Regelung völlig außer Kontrolle geraten - das sieht zwar lustig aus, fährt er aber dabei gegen ein Hindernis, kann er durchaus Schaden nehmen oder welchen verursachen. Theoretisch lassen sich die richtigen Parameter anhand der Kennwerte der gesamten Regelstrecke, also Prozessor, Motortreiber, Motor und Radencoder, bestimmen. Dazu ist aber einiges an messtechnischem Aufwand notwendig. Da Entwickler faule Menschen sind, versuchen sie üblicherweise, sich den richtigen Werten durch Trial and Error anzunähern.

Hier ist es wichtig zu wissen, dass die PID-Parameter des Roboters im aufgebockten Zustand von denen einer Lastfahrt abweichen. Wenn die Regelung im Leerlauf schwingt, muss sie dieses Verhalten unter Last nicht unbedingt auch zeigen. Des Weiteren hängen die PID-Parameter auch von der Akkuspannung ab. Je höher sie ist, desto mehr Kraft hat der Motor und desto weniger P-Anteil ist unter Umständen notwendig.

Bei der häufig angewendeten Methode nach Ziegler-Nichols betreibt man den Regelkreis an seinen Stabilitätsgrenzen, um die Werte zu ermitteln. Dazu werden K_D und K_I zunächst auf 0 gesetzt und K_P so lange erhöht, bis der Regelkreis nach einer Störung dauerhaft zu schwingen beginnt oder zumindest sehr lange Regelversuche zu beobachten sind (K_{Pkrit}). Nun wird die Periodendauer T_{Krit} des schwingenden P-Reglers gemessen und daraus die so genannte Vorhaltezeit, die Nachlaufzeit und daraus K_D sowie K_I berechnet.



Mit den falschen Parametern schadet die P-Regelung mehr, als dass sie nützt. Im Extremfall fängt sie sogar an zu schwingen.

In der Praxis zeigt sich jedoch, dass T_{Krit} gar nicht so einfach zu messen ist. Etwas ungenauer, aber weitaus weniger aufwendig kommt man zum Ziel, wenn man K_P auf $K_{Pkrit}/2$ setzt und nun K_D allmählich erhöht, bis sich eine einigermaßen schnelle Antwort auf eine Störung einstellt. Durch das Drehen an der K_I-Schraube lässt sich nun die Geschwindigkeit und Genauigkeit des Regelkreises weiter verbessern. Fängt der Kreis wieder an zu schwingen, so muss K_I halbiert werden. In unseren Experimenten haben für unseren c't-Bot bei BOT_SPEED_NORMAL die Werte

```
#define KP 6
#define KI 5
#define KP 1
```

gute Resultate erbracht. Leider zeigt sich, dass diese Werte für andere Geschwindigkeiten nicht gut passen. Für BOT_SPEED_SLOW passen sie gar nicht - der Roboter regelt die Geschwindigkeit nicht aus und hoppelt mehr, als dass er fährt. Für niedrige Drehzahlen muss man deshalb die Werte neu ermitteln, bei BOT_SPEED_SLOW haben $K_P = 2$, $K_I = 2$ und $K_D = 0$ leidlich funktioniert. Grundsätzlich haben diskrete Drehzahlregler aber immer Probleme bei sinkenden Drehzahlen, da einfach weniger Daten sprich Pulse zum genauen Regeln zur Verfügung stehen. Theoretisch ließe sich dies durch eine Verlängerung des Abtastzeitraums lösen, damit sinkt aber wieder die Gechwindigkeit der Regelung.

Damit der Regler bei wechselnder Geschwindigkeit zur Laufzeit die richtigen PID-Parameter zugewiesen bekommt, ist es sinnvoller, sie für jede Geschwindigkeit einzeln zur Laufzeit zuzuweisen, statt per #define festzulegen.

```
if (left==BOT_SPEED_NORMAL){
  Kp = 7;
  Ki = 5;
  Kd = 1;
}
...
if (left==BOT_SPEED_SLOW){
  Kp = 2;
  Ki = 2;
  Kd = 0;
}
```

Hier liegt auch der Vorteil mikroprozessorgesteuerter Regler: Adaptive Regelungen lassen sich viel leichter implementieren als mit Operationsverstärkern.

Um die Stellwerte des Reglers im vernünftigen Rahmen bleiben zu lassen, werden sie per Software begrenzt:

```
#define MoMAX 255
```

```
#define MoMIN 0
...
if (reg_l > MoMAX) reg_l = MoMAX;
if (reg_l < MoMIN) reg_l = MoMIN;
...
```

Andernfalls würde durch den integrierenden Regler bei größeren oder längeren Störungen ein zu hoher Wert entstehen. Zudem ist es sinnlos, den Motor mit Werten anzusteuern, die er ohnehin nicht verarbeiten kann: schneller als Höchstgeschwindigkeit geht einfach nicht.

Interoperabel

Um in ersten Versuchen das Regelungsverhalten und die Geradeausfahrt zu testen, genügt es, nur mit dem Verhalten bot_base() zu arbeiten, alle anderen Verhaltensweisen in bot-logik.c sollten erst einmal auskommentiert oder deaktiviert (deactivateBehavior) werden. Fährt der c't-Bot wie gewollt nun geradeaus statt in Kurven, so kann man weitere Verhalten dazuschalten. Hier stellt sich die Frage, ob und wie man etwa Ausweichmanöver durch bot_avoid_col() in die Regelung einbindet. Dafür muss sie einen Änderungswunsch sofort entgegennehmen und nicht erst 330 ms verstreichen lassen. Dazu wird die Regelung um die Variablen

```
last_left = left;
last_right = right;
```

ergänzt, in denen der Regler sich die letzten Geschwindigkeitswünsche merkt. Durch eine Erweiterung der if-Abfrage regelt der Stellalgorithmus dann sofort, wenn sich einer der Wünsche ändert :

```
if ((last_left != left) || (clock_motor_control >1860))
```

Bei kurzen Ausweichermanövern mit hohen Geschwindigkeitssprüngen hat die Regelung aber keine Chance, den gewünschten Wert schnell genug einzustellen, da die Zahl der Encoderpulse nicht ausreicht, um etwas Vernünftiges zu berechnen. Eine Lösung wäre, die gewünschte Geschwindigkeit einfach ungeregelt weiterzugeben, indem man etwa

```
reg_l=left;
```

setzt und nach Wegfall des Änderungswunsches wieder die Regelung greifen lässt. Allerdings kann der Regler schwer voraussehen, ob die Änderung nun kurzfristig ist oder länger andauert, was beispielsweise bei Kurvenfahrten der Fall sein kann. So kann es vorteilhafter sein, den Regler arbeiten zu lassen. Was für die eigene Anwendung am besten passt, muss der c't-Bot-Besitzer im Experiment herausfinden - dafür ist der c't-Bot ja gedacht.

Eine weitere Schwäche des hier vorgestellten einfachen Modells ist die Trennung der Regler für den linken und rechten Motor. In aufwendigere Regelungen für Roboterantriebe fließen in jeden Regler zusätzlich die Abweichungen des jeweils anderen Motors mit ein. Diese Querkopplung ermöglicht einen echten Gleichlauf beider Motoren. Des Weiteren lassen sich darüber auch Offsets für Kurvenfahrten definieren. Aber auch mit der hier gezeigten Lösung fährt der Roboter prima geradeaus. Zudem kann der Anwender sie um fehlende Funktionen selbst erweitern.

Wer seinen c't-Bot mit einem LC-Display ausgestattet hat, kann die Arbeit der Regelung auf

einfache Weise mitverfolgen und so die Parameter leichter anpassen. Dazu muss der Anwender in ct-bot.c den Aufruf von display() auskommentieren und seine eigene Anzeige in speed_control () einsetzen, beispielsweise so:

```
display_cursor(1,1);
display_printf("L=%d R=%d Cl:%d",lmp , rmp, clock_motor_control);
```

Ausblick

Mit unserer Drehzahlregelung allein, deren Code mit Erscheinen des Hefts auch im CVS verfügbar ist, kann man noch keine wirklich komplexen Aufgaben wie das Anfahren von Zielen zufriedenstellend erledigen. Dazu bedarf es einer Positionsregelung, die aber eine Schicht über der Drehzahlregelung arbeitet. Diese stellt sicher, dass der c't-Bot sich da befindet, wo er sein soll, und trotz seiner Massenträgheit nicht über das Ziel hinausschießt. Sie sorgt auch für ein geschmeidigeres Anfahren und Bremsen des Roboters, indem sie über Rampenfunktionen die Geschwindigkeit allmählich erhöht oder verringert, statt die Motoren sofort abzustellen oder mit maximalen Werten anzusteuern. Das Gleiche gilt auch für Drehungen auf der Stelle. Für diese Aufgaben bietet sich der Maussensor an, der unabhängig von den Drehzahldaten der Radencoder Informationen über die tatsächliche Bewegung liefert. In einem der folgenden Artikel zum c't-Bot wollen wir auf das Theman Positionsregelung näher eingehen. Bis dahin lohnt sich zwischendurch auch ein Blick auf die Projektseite und Mailingliste des Roboters [4]. (**dab[1]**)

Literatur

[1] Thomas Bräunl, Embedded Robotics, Springer Verlag, 2003

[2] Benjamin Benz, Nervensystem, Programmierung des c't-Bot von der Pike auf, c't 3/06, S. 264

[3] **Vishay Datenblatt, Application of Optical Reflex Sensors[2]**

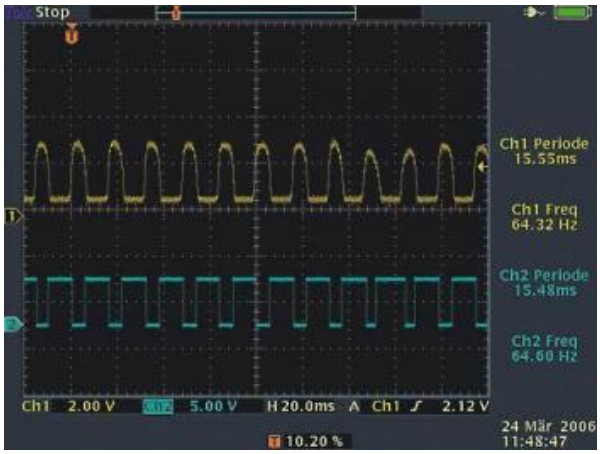
[4] **c't-Bot-Projekt[3]**

Fehlerquelle Radencoder

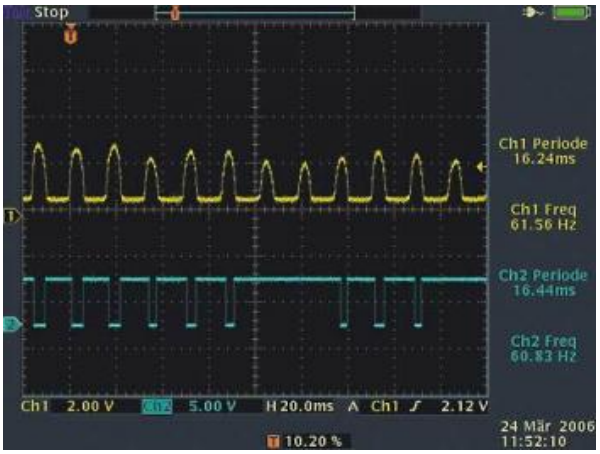
Die Radencoder des c't-Bot arbeiten leider nicht immer zuverlässig: Unter Umständen fallen einige Pulse unter den Tisch. In der Folge zählt der Prozessor zu wenig Pulse pro Radumdrehung, sodass sich die genaue Anfahrt einer Position oder der Motorgleichlauf nur schwer realisieren lassen. Als Ursache für das Problem kommen mehrere Fehlerquellen in Frage: schlechte Qualität der Encoderscheibe, Montagefehler, zu geringe Ansteuerspannung des Schmitt-Triggers, Klebstoffreste und Streulicht.

Die größte Quelle für unbefriedigende Ergebnisse der Radsensoren sind die Encoderscheiben. Während das Licht des sichtbaren Spektrums von einem schwarzen Feld noch ausreichend geschluckt wird, um einen Hell-Dunkel-Wechsel im Sensor zu erzeugen, gilt dies mitnichten für das Infrarotspektrum des Reflexkopplers CNY70. Hier ist Schwarz nicht immer schwarz, sodass selbst ein tiefdunkles Feld den Infrarotlichtstrahl fast vollständig reflektieren kann - wie ein weißes Feld. Der Puls des Sensors bleibt dann aus.

Der Hersteller des CNY70, Vishay Telefunken, gibt in seinem Datenblatt das Reflexionsvermögen verschiedener Tinten, Toner und Stifte bei 950 nm an [3]. Erstaunlicherweise reflektiert etwa ein



Ein richtig eingestellter Radencoder erzeugt aus jedem Hell-Dunkel-Wechsel einen analogen Puls, den der Schmitt-Trigger in ein Rechtecksignal umwandelt.



Die Signale des CNY70 sind teilweise zu schwach, sodass der Schmitt-Trigger nicht auslöst und ein Viertel der Impulse verloren geht.

schwarzer Stabilo-Stift fast 80 Prozent des Lichts, während ein mit Edding geschwärztes Feld nur zehn Prozent zurückwirft. Ähnlich schwankend sind auch die Ergebnisse ausgedruckter Vorlagen. In vielen Fällen verbessern sich die Resultate, wenn man beim Ausdruck der Scheiben mögliche Economy-Modes im Druckertreiber ausschaltet und in den Einstellungen die Druckdichte manuell auf „Dunkel“ setzt.

Neue Versionen des c't-Bots werden bereits mit gefräster Encoderscheibe ausgeliefert. Besitzer älterer Versionen erhalten sie von Segor gegen Einsendung eines frankierten Rückumschlages. (Bild: Volker Dirks)

Wer auch damit keine zuverlässigen Impulse aus den invertierenden Schmitt-Triggern des IC3 herausbekommt, sollte die Vorteilerwiderstände R17 und R19 sowie R18 und R20 anpassen. Hier hilft nur probieren. Die weitaus besten Ergebnisse lieferten in unseren Tests gefräste Encoderscheiben aus Kunststoff, die auf die Innenseite der Räder geklebt werden. Mit dem Kleber sollte man allerdings sparsam umgehen, damit er nicht unter den Scheiben herausläuft. Mancher getrocknete Klebstoffklecks absorbiert Infrarotlicht nämlich nahezu vollständig. In einer gefrästen Aussparung hat er dann den gleichen Effekt wie der schwarze Kunststoff.

Mitunter bringt auch Streulicht mit hohem Infrarotanteil die Radencoder aus dem Takt, insbesondere wenn der Abstand der Sensoren zu den Scheiben zu groß ist. Dann erkennen die Sensoren die Übergänge zwischen hell und dunkel schlechter, was sogar zum „Prellen“ führen kann: Statt eines Pulses erzeugt ein CNY70 mehrere. Um dieses programmtechnisch zu unterdrücken, dient in sensor-low.c der Wert ENC_ENTPRELL. Auch hier hilft wieder nur, mit verschiedenen Größen zu experimentieren, bis man für seinen c't-Bot das beste Ergebnis gefunden hat. Erfahrungsgemäß liegt ENC_ENTPRELL zwischen 4 und 12.

URL dieses Artikels:

`http://www.heise.de/-290434`

Links in diesem Artikel:

[1] mailto:dab@ct.de

[2] http://www.vishay.com/docs/80107/80107.pdf

[3] https://www.heise.de/ct/artikel/c-t-Bot-und-c-t-Sim-284119.html