Nichtlineare Regelung mit einem Raspberry PI

Automatisierung des Versuchsaufbaus „Ball auf Rad“ mit einem Raspberry PI und offener Hard- und Software

S.-O. Lindert, C. Höfler, K. Schlacher OVE

Die kommerziell erhältlichen Automatisierungssysteme sind sehr kostspielig. Hinzu kommen meist Lizenzgebühren für Software zur Simulation und Automatisierung. In diesem Artikel wird die Lösung einer anspruchsvollen Automatisierungsaufgabe, die Balancierung eines Balles auf einem rotierenden Rad, durch offene Software und preiswerte Hardware beschrieben. Zum Einsatz kommen der Einplatinencomputer Raspberry PI, das offene Hardware-Automatizationssystem BIOE und offene Software.

Schlüsselwörter: Raspberry PI; BIOE; Ball on Wheel; nichtlineare Regelung; objektorientierte Programmierung; open-source

Nonlinear control with a Raspberry PI. Automation of the experiment setup „ball on wheel“ with a Raspberry PI and open source hard- and software.

The commercially available automation systems are quite expensive. Additionally, there are license fees for the simulation and automation software. In this article a challenging automation task, namely balancing a ball on a spinning wheel, is solved solely by open source software and inexpensive hardware. The single-board-computer Raspberry PI, the open hardware automation system BIOE and open source software are employed.

Keywords: Raspberry PI; nonlinear control; object-oriented programming; open-source

Eingegangen am 9. Februar 2017, angenommen am 28. März 2018, online publiziert am 24. April 2018
© The Author(s) 2018

1. Einleitung

Der folgende Artikel möchte demonstrieren, wie mit preiswerten und frei verfügbaren Mitteln moderne Automatisierungsaufgaben gelöst werden können.

Zunächst wird in Abschn. 2 der Versuchsaufbau „Ball auf Rad“ kurz vorgestellt. Seine physikalischen Eigenschaften werden soweit dargestellt, wie es für den anschließenden Reglerentwurf erforderlich ist. Der Reglerentwurf selber wird für den vorliegenden Aufbau gezeigt, ohne auf theoretische Hintergründe näher einzugehen.

In Abschn. 3 wird gezeigt, wie ein Raspberry PI, ein ursprünglich als Lernhilfe für Schüler konzipierter Einplatinencomputer, als echtzeitfähiger digitaler Prozessrechner verwendet werden kann. Dabei wird im Sinne von Industrie 4.0 nur der echtzeitkritische geschlossene Regelkreis am Versuch selber realisiert, während das Bedienen und Beobachten über das Internet geschieht. Hierfür kann man sich mit einem beliebigen Browser einloggen. Es kommt ausschließlich open-source Software zum Einsatz.

Im vierten und letzten Abschnitt wird die ebenfalls offene Simulationsumgebung erläutert und Testergebnisse von dem Versuchsaufbau gezeigt. Ein Video, das den funktionierende Versuch zeigt, wurde ebenfalls aufgenommen und ist unter <https://youtu.be/CJSD-gsHqU> zu sehen.

2. Modellbildung und Reglerentwurf

Der Versuchsaufbau ist ein international verbreitetes Labormodell und auf Englisch als „Ball on Wheel“ bekannt. Zunächst sollen der physikalische und mathematische Hintergrund kurz skizziert werden, beginnend mit der Modellbildung.

2.1 Modellbildung

Ein Ball mit einer Masse m_B und einem Radius r_B liegt auf einem Rad. Dieses Rad mit Radius r_W und dem Trägheitsmoment J_W wird von einem Gleichstrommotor angetrieben. Ziel des Aufbaus ist es, den Ball durch die Drehung des Rades auf dem Rad zu balancieren. Der Winkel des Rades wird mit φ_W und der des Balles mit φ_B (siehe Skizze Abb. 1) bezeichnet. Zunächst sollen beide Winkel einbezogen werden. Allerdings wird vom Rad lediglich die Winkelgeschwindigkeit $\dot{\varphi}_W$ gemessen, so dass der Winkel φ_W später weggelassen wird. Mit dem Trägheitsmoment J_W des Rades sowie dem Trägheits-

Lindert, Sven-Olaf, Institut für Regelungstechnik und Prozessautomatisierung, Johannes Kepler Universität Linz, Altenberger Straße 69, 4040 Linz, Österreich (E-Mail: Sven-Olaf_Lindert@gmx.de); **Höfler, Christian**, Institut für Regelungstechnik und Prozessautomatisierung, Johannes Kepler Universität Linz, Altenberger Straße 69, 4040 Linz, Österreich; **Schlacher, Kurt**, Institut für Regelungstechnik und Prozessautomatisierung, Johannes Kepler Universität Linz, Altenberger Straße 69, 4040 Linz, Österreich

Abb. 1. Versuchsaufbau „Ball auf Rad“ mit physikalischer Skizze

moment J_B und der Masse m_B des Balles erhält man die kinetische Energie des Gesamtsystems

$$E_{\text{kin}} = \frac{1}{2} \left(J_W \dot{\varphi}_W^2 + J_B \frac{r_W^2}{r_B^2} (\dot{\varphi}_B - \dot{\varphi}_W)^2 + m_B (r_W + r_B)^2 \dot{\varphi}_B^2 \right).$$

Weiters erhält man mit der Schwerkraft g die potentielle Energie

$$E_{\text{pot}} = g m_B (r_W + r_B) \cos \varphi_B.$$

Aus dem Lagrange Formalismus erhält man die Bewegungsgleichungen

$$\begin{bmatrix} J_W + J_B \frac{r_W^2}{r_B^2} & -J_B \frac{r_W^2}{r_B^2} \\ -J_B \frac{r_W^2}{r_B^2} & J_B \frac{r_W^2}{r_B^2} + m_B (r_W + r_B)^2 \end{bmatrix} \begin{bmatrix} \ddot{\varphi}_W \\ \ddot{\varphi}_B \end{bmatrix} - \begin{bmatrix} 0 \\ g m_B (r_W + r_B) \sin \varphi_B \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \quad (1)$$

mit den verallgemeinerten Kräften $\mathbf{Q} = [Q_1 \quad Q_2]^T$. Die Kräfte setzen sich zusammen aus einem Teil vom Motor und einem Teil aus der Reibung im System. Der wirkt über ein Getriebe mit der Übersetzung i_G auf das Rad. Da es sich um einen Gleichstrommotor handelt, ist sein Moment M_M gegeben durch:

$$M_M = \frac{i_G k_M}{R_A} (u_A - i_G k_M \dot{\varphi}_W) \quad (2)$$

mit der Motorkonstanten k_M , dem Ankerwiderstand R_A und der Ankerspannung u_A als Stellgröße. Die Induktivität des Motors ist gegenüber dem mechanischen Aufbau vernachlässigbar klein. Die wesentlichen Reibbeiträge entstehen durch das Drehen des Rades im Lager

$$R_W = k_W^c \text{sign} \dot{\varphi}_W + k_W^v \dot{\varphi}_W$$

und durch das Rollen des Balles auf dem Rad

$$R_B = k_B^c \text{sign} (\dot{\varphi}_B - \dot{\varphi}_W) + k_B^v (\dot{\varphi}_B - \dot{\varphi}_W),$$

jeweils mit einem Coulombschen und einem viskosen Anteil. Zusammengefasst sind die Kräfte dann

Tab. 1. Zahlenwerte der Parameter und des Systems (4)

m_B	0.2046	kg	k_M	0.041	$\frac{\text{Nm}}{\text{A}}$
r_B	0.068	m	R_A	2.96	Ω
r_W	0.2663	m	k_W^c	0.04994	N m
J_W	0.1288	kg m ²	k_W^v	0.005540	N m s
J_B	0.001058	kg m ²	k_B^c	0.002979	N m
i_G	14		k_B^v	0.0003945	N m s
a_{21}	2.014				
a_{22}	-0.8446				
a_{23}	0.001669				
a_{31}	18.00				
a_{32}	-0.3413				
a_{33}	-0.009400				
b_2	1.402				
b_3	0.5820				

$$Q_1 = M_M - R_W + R_B \quad (3a)$$

$$Q_2 = -R_B. \quad (3b)$$

Die Zahlenwerte der Parameter sind in Tab. 1 zu finden.

Sowohl für den Reglerentwurf als auch für die Trajektorienplanung wird eine kompakte Darstellung als eingangsaffines System, im Englischen Affine Input System, verwendet. In den Modellgleichungen taucht der Winkel φ_W nicht auf. Da er auch nicht gemessen wird, kann man ihn weglassen und führt $\omega_W = \dot{\varphi}_W$ ein. Setzt man in die Gl. (1), (2) und (3a), (3b) die Parameterwerte aus Tab. 1 ein, so erhält man mit dem Zustand

$$\mathbf{x} = [\varphi_B \quad \omega_W \quad \dot{\varphi}_B]^T$$

das eingangsaffine System

$$\begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \\ \ddot{\varphi}_B \end{bmatrix} = \underbrace{\begin{bmatrix} \dot{\varphi}_B \\ a_{21} \sin \varphi_B + a_{22} \omega_W + a_{23} \dot{\varphi}_B + C_W \\ a_{31} \sin \varphi_B + a_{32} \omega_W + a_{33} \dot{\varphi}_B + C_B \end{bmatrix}}_{\mathbf{f}(\mathbf{x})} + \underbrace{\begin{bmatrix} 0 \\ b_2 \\ b_3 \end{bmatrix}}_{\mathbf{g}} u_A. \quad (4)$$

Die Werte sind rechts in Tab. 1 zu finden. Die Konstanten C_W und C_B entstehen durch die Coulombsche Reibung und hängen nur von den Vorzeichen $\text{sign}(\omega_W)$ und $\text{sign}(\dot{\varphi}_B - \omega_W)$ ab, die sich während des Betriebes zumeist nicht ändern. Das rechtfertigt die Berücksichtigung als Konstante. Im Einzelnen sind es

$$C_W = -0.3611 \text{sign}(\omega_W) + 0.01260 \text{sign}(\dot{\varphi}_B - \omega_W)$$

$$C_B = -0.1499 \text{sign}(\omega_W) - 0.07098 \text{sign}(\dot{\varphi}_B - \omega_W).$$

2.2 Reduzierter Beobachter

Da verschiedene Zustandsregler zum Einsatz kommen sollen, muss der gesamte Zustand messbar sein. Die Größen ω_W und φ_B werden messtechnisch erfasst. Die dritte $\dot{\varphi}_B$ muss beobachtet werden. Sieht man von den Konstanten ab, so fällt bei der Betrachtung von Gl. (4) auf, dass die nichtlinearen Terme einzig aus gemessenen Größen bestehen und die zu beobachtende Größe linear auftritt. Das System hat die Struktur

$$\begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \end{bmatrix} = \mathbf{A}_y \dot{\varphi}_B + \mathbf{f}_y(\varphi_B, \omega_W, u_A)$$

$$\ddot{\varphi}_B = \mathbf{A}_\varphi \dot{\varphi}_B + \mathbf{f}_\varphi(\varphi_B, \omega_W, u_A).$$

Der Zustand $\hat{\mathbf{z}}$ des reduzierten Beobachters und die Beobachterverstärkung werden analog zum linearen Fall eingeführt

$$\mathbf{z} = \dot{\varphi}_B + \mathbf{L} \begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \end{bmatrix}, \quad \hat{\mathbf{z}} = \dot{\varphi}_B + \mathbf{L} \begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \end{bmatrix}$$

$$\mathbf{z} - \hat{\mathbf{z}} = \dot{\varphi}_B - \dot{\varphi}_B = \mathbf{e}.$$

Die Beobachtergleichungen lauten dann

$$\dot{\hat{\mathbf{z}}} = \mathbf{A}_\varphi \dot{\varphi}_B + \mathbf{f}_\varphi(\varphi_B, \omega_W, u_A) + \mathbf{L}(\mathbf{A}_y \dot{\varphi}_B + \mathbf{f}_y(\varphi_B, \omega_W, u_A))$$

und für den „wahren“ Wert

$$\dot{\mathbf{z}} = \mathbf{A}_\varphi \dot{\varphi}_B + \mathbf{f}_\varphi(\varphi_B, \omega_W, u_A) + \mathbf{L}(\mathbf{A}_y \dot{\varphi}_B + \mathbf{f}_y(\varphi_B, \omega_W, u_A)).$$

Zieht man beide voneinander ab, erhält man die lineare Fehlerdynamik

$$\dot{\mathbf{z}} - \dot{\hat{\mathbf{z}}} = \dot{\mathbf{e}} = (\mathbf{A}_\varphi + \mathbf{L}\mathbf{A}_y)\mathbf{e}.$$

2.3 Reglerentwurf

Zwei unterschiedliche Zustandsregler sollen realisiert werden, zum ersten ein einfacher linearer Zustandsregler, zum zweiten eine Steuerung und Regelung, die die Eigenschaft der Flachheit nutzt.

2.3.1 Linearer Regler mit I-Anteil

Bei der näheren Betrachtung von Gl. (4) fällt auf, dass die einzigen nichtlinearen Terme die Konstanten der trockenen Reibung und $\sin \varphi_B$ sind. Wenn der Ball also in der Nähe der oberen Ruhelage $\varphi_B \approx 0$ bleibt, so wird ein linearer Regler gute Ergebnisse bringen. Zur Kompensation der trockenen Reibung wird ein I-Anteil angefügt. Das linearisierte System lautet

$$\begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \\ \ddot{\varphi}_B \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \varphi_B \\ \omega_W \\ \dot{\varphi}_B \end{bmatrix} + \begin{bmatrix} 0 \\ b_2 \\ b_3 \end{bmatrix} u_A. \quad (5)$$

Der Zustandsregler wird um den I-Anteil erweitert

$$\dot{x}_I = \omega_W - \omega_W^{\text{sol}},$$

so dass das Gesamtsystem

$$\begin{bmatrix} \dot{\varphi}_B \\ \dot{\omega}_W \\ \ddot{\varphi}_B \\ \dot{x}_I \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \varphi_B \\ \omega_W \\ \dot{\varphi}_B \\ x_I \end{bmatrix} + \begin{bmatrix} 0 \\ b_2 \\ b_3 \\ 0 \end{bmatrix} u_A + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} \omega_W^{\text{sol}}$$

ergibt. Zur Berechnung der Rückführung $u_A = -\mathbf{k}^T \mathbf{x}$ wird die LQR-Methode (siehe z.B. [7]) verwendet mit den Wichtungen

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad \text{und} \quad R = 1$$

und man erhält

$$\mathbf{k}^T = \begin{bmatrix} 99.74 & -2.809 & 23.79 & -1.414 \end{bmatrix}.$$

2.3.2 Flachheitsbasierte Trajektorienplanung

Man kann leicht zeigen, dass das AI-System (4) flach ist. Die Eigenschaft der Flachheit bedeutet, dass eine fiktive Größe y existiert, aus deren Verlauf $y(t)$ der Verlauf sämtlicher anderen Größen $\mathbf{x}$ und u_A berechnet werden kann, ohne eine Differentialgleichung lösen zu müssen. Dies erleichtert die Planung von Trajektorien erheblich. Eine Einführung in diese Theorie findet sich in [2] oder [4]. Zur Ermittlung des flachen Ausgangs y wird die nichtlineare Regelungsnormalform genutzt, siehe [5, 6] oder [2]. Glücklicherweise ist die Distribution

$$D = [\mathbf{g}, [\mathbf{f}, \mathbf{g}]] = \begin{bmatrix} 0 & b_3 \\ b_2 & -1.186 \\ b_3 & -0.4840 \end{bmatrix}$$

konstant. Dadurch ist lediglich zu prüfen, ob sie Maximalrang besitzt. Der Rang der folgenden Distribution

$$[\mathbf{g}, [\mathbf{f}, \mathbf{g}], [\mathbf{f}, [\mathbf{f}, \mathbf{g}]]] = \begin{bmatrix} 0 & b_3 & -0.4840 \\ b_2 & -1.186 & 1.003 + 1.172 \cos \varphi_B \\ b_3 & -0.4840 & 0.4094 + 10.48 \cos \varphi_B \end{bmatrix}$$

ist voll für relevante Winkel¹ φ_B , wie man durch Berechnen der Determinante leicht zeigen kann. Der flache Ausgang ergibt sich dann aus dem Links-Annihilator zu D als

$$y = \begin{bmatrix} 0.01423 & 0.4150 & -1 \end{bmatrix} \mathbf{x}.$$

Damit kann man einen neuen Zustand einführen

$$\mathbf{z} = \begin{bmatrix} y & \dot{y} & \ddot{y} \end{bmatrix}^T$$

mit der Transformation $\mathbf{z} = \Phi(\mathbf{x})$, die ausgeschrieben

$$z_1 = y = 0.01423 \varphi_B + 0.4150 \omega_W - \dot{\varphi}_B \quad (6a)$$

$$z_2 = \dot{y} = -17.17 \sin \varphi_B - 0.01009 \omega_W + 0.02432 \dot{\varphi}_B + 0.4150 C_W - C_B \quad (6b)$$

¹Rangverlust tritt erst bei Winkeln $|\varphi_B| > \frac{\pi}{2}$ auf, bei denen der Ball vom Rad fällt.

$$z_3 = \ddot{y} = -17.17 \dot{\varphi}_B \cos \varphi_B + 0.4174 \sin \varphi_B + 0.0002454 \omega_W - 0.0002454 \dot{\varphi}_B - 0.01009 C_W + 0.02432 C_B \quad (6c)$$

lautet. Auf das Ausschreiben der inversen Transformation $\mathbf{x} = \Phi^{-1}(\mathbf{z})$ wird aus Platzgründen verzichtet. Man geht so vor, dass zunächst die Größen $\dot{\varphi}_B$ und ω_W , die lediglich linear auftreten, eliminiert werden. Die verbleibende trigonometrische Gleichung in φ_B ist numerisch robust lösbar, womit dann wiederum $\dot{\varphi}_B$ und ω_W berechnet werden können. Die Werte der Konstanten C_W und C_B sind praktisch immer bekannt. Wendet man diese Transformation auf Differentialgleichung (4) an, so erscheint sie in der gewünschten Form

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ \alpha(\mathbf{z}) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \beta(\mathbf{z}) \end{bmatrix} u_A. \quad (7)$$

Hiermit kann man offensichtlich aus der Kenntnis von $y(t) = z_1(t)$ durch Differenzieren und Anwenden von Φ^{-1} sämtliche Größen berechnen.

2.3.3 Reglerentwurf mittels exakter Linearisierung

Die Darstellung (7) kann auch genutzt werden, um einen Trajektorienfolgeregler mit linearer Fehlerdynamik zu entwerfen. Der gewünschte Verlauf $y^d(t) = z_1^d(t)$ ist mit ausreichend vielen Ableitungen bekannt. Aus der Kenntnis des Zustandes $\mathbf{x}$ kann man mit der Transformation $\Phi(\mathbf{x})$ den Istzustand $\mathbf{z}$ bestimmen. Nun fordert man, dass der Fehler $(y - y^d)$ einer stabilen Differentialgleichung gehorcht

$$0 = \ddot{y} - \ddot{y}^d + c_2(\ddot{y} - \ddot{y}^d) + c_1(\dot{y} - \dot{y}^d) + c_0(y - y^d)$$

und mit Gl. (7)

$$0 = \alpha(\mathbf{z}) + \beta(\mathbf{z}) u_A - \ddot{y}^d + c_2(z_3 - \ddot{y}^d) + c_1(z_2 - \dot{y}^d) + c_0(z_1 - y^d).$$

Löst man obige Gleichung nach der Stellgröße u_A auf, erhält man die stabilisierende linearisierende Rückführung

$$\mathbf{z} = \Phi(\mathbf{x}) \quad (8a)$$

$$u_A = \frac{1}{\beta(\mathbf{z})} (\ddot{y}^d - \alpha(\mathbf{z}) - c_2(z_3 - \ddot{y}^d) - c_1(z_2 - \dot{y}^d) - c_0(z_1 - y^d)). \quad (8b)$$

Darin enthalten ist selbstverständlich auch die Möglichkeit, einen konstanten Wert ω_W^{sol} einzustellen. Dann ist nur $y^d(t) = \ddot{y}^d$ zu berechnen, denn sämtliche Ableitungen sind Null. Die Gl. (6a), (6b) (6c) wird zu

$$\ddot{y}^d = 0.01423 \varphi_B^{\text{sol}} + 0.4150 \omega_W^{\text{sol}}$$

$$0 = -17.17 \sin \varphi_B^{\text{sol}} - 0.01009 \omega_W^{\text{sol}} + 0.4150 C_W - C_B$$

und die Rückführung bekommt die in der Literatur verbreitetere Form

$$u_A = -\frac{1}{\beta(\mathbf{z})} (\alpha(\mathbf{z}) + c_2 z_3 + c_1 z_2 + c_0(z_1 - y^d)).$$

3. Implementierung mit offener Hard- und Software

Ein Anliegen dieses Artikels ist es zu zeigen, dass anspruchsvolle Regelungskonzepte nicht notwendigerweise teure Hard- oder Softwarepakete benötigen. Als echtzeitfähiger Prozessrechner wurde ein Raspberry PI Modell B verwendet, der über seine GPIO (General Purpose Input Output) Pins mit der Messhardware kommunizierte. Zum Bedienen und Beobachten kam das RTXML, das in [11] vorgestellt wurde, zum Einsatz. Als digitales Ein- und Ausgabesystem wurde BIOE [10] in einer für den Raspberry PI adaptierten Fassung und als

Betriebssystem wurde Xenomai [1], eine auf Linux basierende Echtzeitumgebung, verwendet. Dadurch hat man die gesamte Linux Umgebung für Diagnose- und Verwaltungszwecke zur Verfügung.

Zum Antrieb des Rades wurde ein Motor-Getriebe-Paket, bestehend aus einem Motor der Serie 3557 K024 CR und einem Getriebe der Serie 38/2, der Firma *Faulhaber* verwendet. Die Winkelgeschwindigkeit ω_W wurde durch einen Digital-Inkrementalgeber mit anschließender Torzeitmessung erfasst. Der Winkel φ_B des Balles wurde indirekt über einen Lasersensor vom Typ OADM 2014470/S14C der Firma *Baumer Electric* gemessen.

3.1 LPT-Adaptierung des Raspberry PI für BIOE-Anschluss

BIOE wurde ursprünglich für eine Verwendung mit einem echtzeitfähigen Standard-PC entwickelt. Daher verwendet es dessen parallele Schnittstelle (LPT) zur Übertragung. So musste der Raspberry PI mit einer LPT-Schnittstelle versehen werden. Dies wurde mit einem speziellen LPT-Treiber, der auf dem Beispiel [8] basiert, erreicht. Mit diesem Treiber ist neben der geforderten minimalen IO-Zeit und der Zuverlässigkeit beim Setzen einzelner GPIOs auch das zeitlich parallele Setzen und Lesen mehrerer GPIOs möglich. Mit dem damit entwickelten Treiber ergibt sich sogar eine etwas bessere IO-Zeit als am PC in der Regel mit einer echten LPT-Schnittstelle zu erwarten wäre. Die IO-Zeit am Raspberry PI beträgt etwa 30 μs . Noch kürzere Zeiten sind vom Raspberry PI her möglich. Jedoch sind die Mikroprozessoren der BIOE-Module dafür nicht geeignet.

Zur Adaption der Hardware wurde der LPT-Adapter geplant und gefertigt, welcher dazu dient, den Logik-Pegel für High von 5 V des BIOE auf die 3.3 V des Raspberry PI wandeln zu können. Eine Trägerplatine nimmt drei kleine Logik-Konverter-Platinen auf, welche ihrerseits je vier bidirektionale IO-Leitungen umsetzen können. Diese werden unter anderem für Entwicklerarbeiten am Raspberry PI angeboten.

3.2 BIOE-Erweiterung

Das bisher veröffentlichte BIOE [10] bietet die Möglichkeit pro Modul bis zu vier Spannungen zwischen 0 und 5 V, einzulesen. Zur Ausgabe analoger Signale existiert lediglich eine Puls-Weiten-Modulation (PWM). Wegen des offenen Charakters ist das BIOE jedoch gut erweiterbar. Es wurde eine Platine gefertigt, die mit verschiedenen Pin-kompatiblen 12-bit ADC-Chips bestückt werden kann. Hier kamen ein HI-674A der Firma *Intersil* und ein ADS7774 von *Texas Instruments* zum Einsatz. Weiters wurde eine Platine gefertigt, die einen 12-bit DAC, den DAC811 von *Texas Instruments*, an das BIOE anschließt. Dadurch können mehrere bipolare Spannungen mit wählbaren Bereichen eingelesen und ausgegeben werden. Die jeweils nötigen Versorgungsspannungen werden mit einem integrierten DC/DC-Wandler aus der +5 V Versorgung gewonnen.

Die BIOE-Firmware wurde modifiziert, um eine neue DTN (Device-Type-Number) für den Betrieb mit der ADC-Platine und der DAC-Platine bereit zu stellen.

3.3 Software-Anpassungen

Zusammen mit BIOE [10] wird das Internet-basierte Bedien- und Beobachtungspaket RTXML angeboten. Dieses ist als Erweiterung für Scilab [9] und Scicoslab verfügbar. Über die in Scilab und Scicoslab enthaltene graphische Programmierungsumgebung Scicos, kann damit C-Code und dessen direkt ausführbares Programm generiert werden. Sobald es gestartet wird, bietet es ein über HTTP bereitgestelltes Web-Interface an, welches dem Benutzer erlaubt, Parameter des im Programm enthaltenen Echtzeittasks zu ändern und Messdaten von jenem im Web-Browser graphisch darzustellen.

Tab. 2. Ergebnis der Echtzeitmessung über vier Stunden

Zeit	Mittelwert	Max	Min	Stichprobenvarianz
Jitter	19.84 μs	55.85 μs	10.47 μs	1.969 μs
Einlesezeit	112.8 μs	172.3 μs	102.9 μs	3.034 μs
Echtzeitkritische Rechenzeit	41.17 μs	1818 μs	1.96 μs	5.098 μs
Ausgabezeit	33.22 μs	90.78 μs	29.34 μs	1.815 μs
Echtzeitkritische Gesamtzeit	187.2 μs	1968 μs	137.8 μs	6.807 μs
Sonstige Rechenzeit	18.77 μs	147.9 μs	14.26 μs	4.738 μs

Abb. 2. Histogramm des Jitters

Im Zuge der in diesem Artikel vorgestellten Adaptierung und Erweiterung von BIOE wurde auch eine Modifikation von RTXML realisiert, welche die Verwendung von RTXML in der Programmierung mit C oder C++ möglich macht. Die spezielle Schnittstelle, die bei der Code-Generierung in Scilab verwendet wird, wurde dabei durch eine für den Programmierer lesbare ersetzt.

Als Betriebssystem wird Xenomai [1] für den Raspberry PI verwendet. Eine Verwendung von RT PREEMPT und Standard-Linux funktioniert mit wenigen Anpassungen ebenso.

Um die Echtzeitfähigkeit zu testen, wurde ein Ablauf ähnlich zu dem im Abschn. 4.3 vier Stunden lang durchgeführt. Dabei wurden während der Ausführung über Xenomai-POSIX Zeitpunkte der CLOCK_MONOTONIC² gemessen. Das Ergebnis findet man in Tab. 2. Die Verteilung des Jitters, also die Zeit um die sich der Start des Programmes gegenüber dem geplanten Start verzögert, ist in Abb. 2 als Histogramm dargestellt. Hervorzuheben ist, dass der Jitter im Mittelwert lediglich 19.84 μs mit geringen Schwankungen beträgt, so dass auch wesentlich kürzere Abtastzeiten, als die hier verwendeten 10 ms möglich sind. Die echtzeitkritische Rechenzeit ist jene Zeit, die von der Bereitstellung der Messwerte bis zur Bereitstellung der Stellgröße für den jeweiligen Abtastschritt verstreicht. Die echtzeitkritische Gesamtzeit ist die Zeit, die vom Start des Programmes bis zur Ausgabe der neuen Stellgröße verstreicht, also die Summe aus Eingabe-, Ausgabe- und Rechenzeit. Der recht hohe Maximalwert zeigt an, dass es durch betriebssysteminterne Abläufe sehr vereinzelt auch zu deutlich längeren Rechenzeiten kommen kann, die jedoch die Stabilität der Regelung nicht merklich beeinflussen. In Abb. 3 sieht man das Histogramm der echtzeitkritischen Gesamtzeit.

²Eine Uhr, die nicht gestellt werden kann und die die monoton verstrichene Zeit seit einem nicht definierten Anfangszeitpunkt misst.

3.4 Objektorientierte Programmierung

Um eine gute Modularisierung und Wiederverwendbarkeit zu erreichen, wird der Regler objektorientiert in C++ realisiert. Dadurch können in der gegebenen Struktur (Abb. 4) unterschiedliche Regler leicht ausgetauscht werden. Dabei müssen selbstverständlich nicht immer sämtliche Klassen vorhanden sein und bei den Klassen müssen nicht sämtliche Eigenschaften ausgeprägt werden. Beispielsweise hat der Trajektorienfolgeregler (8a), (8b) keine internen Zustände und der lineare Regler mit I-Anteil benötigt keinen $x^d(t)$ -Generator. Man beachte, dass der $y^d(t)$ -Generator keinerlei Information über die spezifische Regelstrecke hat, bzw. benötigt. Er lässt sich daher für andere Aufgaben wiederverwenden. In der vorliegenden Arbeit wird die Solltrajektorie durch abschnittsweise definierte Polynome dargestellt, bei denen an den Übergangsstellen die erforderliche Glattheit garantiert wird. Weil die Darstellung der Trajektorien eine private Eigenschaft ist, können ohne Schwierigkeit auch andere Darstellungen implementiert werden. Ein Algorithmus zur Planung von Übergängen zwischen Ruhelagen y_{start}^d und y_{end}^d gehört zur Grundausstattung.

4. Testergebnisse

Der Aufbau wurde mit den beiden vorgestellten Regelungskonzepten getestet. Zuvor jedoch mussten die Algorithmen an einem Simulationsmodell validiert werden.

4.1 Simulationsumgebung

Auch die Simulationsumgebung wurde ausschließlich mit offener Software eingerichtet, nämlich mit GNU-Octave [3] und dessen C++ Schnittstelle. Zunächst wurden die Bewegungsgleichungen (1) um einige bewusst nicht modellierte Eigenschaften erweitert, um realitätsnahe Störungen in der Simulation zu erhalten. Speziell wurden die Exzentrizität des Balles, denn der Ball ist nicht ausgewuchtet, und ein feineres Reibmodell hinzugenommen. Dieses Modell wurde für den Befehl `/sode` von GNU-Octave als zeitkontinuierliches Modell implementiert. In einer Schleife über die Abtastzeitpunkte wurden die

Abb. 3. Histogramm der echtzeitkritischen Gesamtzeit

Abb. 4. Generelles Objektschema des Reglers

simulierten Messwerte über das *External Code Interface* (Oct-Files) an die in Abschn. 3.4 skizzierten Regler übergeben und die berechnete Stellgröße für den neuen Abtastschritt entgegengenommen. Zu bedenken ist dabei nur, dass die Objekte nach jedem Aufruf zerstört werden. Daher müssen deren Eigenschaften während der Simulation *static* deklariert werden, was zur Folge hat, dass jeweils nur eine Reglerkonfiguration gleichzeitig getestet werden kann. Von Abtastzeitpunkt zu Abtastzeitpunkt wird dann wieder zeitkontinuierlich simuliert, weshalb man von einer hybriden Simulation sprechen kann.

Die validierten Regler werden in der Folge an der realen Strecke mit einer Abtastzeit von 10 ms getestet.

4.2 Test: Linearer Regler mit I-Anteil

Zwei Anschläge an dem Aufbau verhindern, dass der Ball im Falle des Versagens der Regelung herunter fällt. Zu Beginn des Tests liegt der Ball an einem dieser Anschläge. Das Testergebnis findet man in Abb. 5. Mit Einschalten des Reglers wird der Ball in die Mitte geholt und bei einer festen Winkelgeschwindigkeit ω_M^{soll} balanciert. Ein Sollwertsprung demonstriert die Robustheit. Dabei versucht der Regler das System in der Ruhelage zu halten, was natürlich bei Beschleunigung speziell für den Winkel φ_B nicht gelten kann. Die genaue Gestalt der Abweichungen bleibt bei diesem Regler dem Zufall überlassen. Man beachte, dass auch der Vorzeichenwechsel in ω_W ,

bei dem auch die trockene Reibung das Vorzeichen wechselt, kein Problem ist. Der I-Anteil kompensiert die geänderte Reibung wieder.

Falls keine höheren Ansprüche an das Verhalten bei Sollwertänderungen gestellt werden, ist dieser Regler gut genug, denn die Ruhelage wird robust stabilisiert.

4.3 Test: Flachheitsbasierter Trajektorienfolgeregler

Die Trajektorienplanung soll in etwa die gleiche Aufgabe lösen, wie zuvor der lineare Regler. Der Ball soll von einem Anschlag in die Mitte geholt und dort bei einer festen Geschwindigkeit ω_W^{soll} balanciert werden. Danach soll er auf eine andere feste Sollgeschwindigkeit wechseln. Die Wechsel sollen möglichst schnell geschehen. Dabei sind der Schnelligkeit jedoch Grenzen gesetzt, denn der Ball soll sich nicht zu weit aus der Mitte bewegen. Schaut man sich Gl. (6b) näher an, so fällt auf, dass $\dot{y}$ überwiegend von $\sin \varphi_B$ abhängt. Wenn man also $z_2 = \dot{y}$ beschränkt, wird sich der Ball nicht zu weit heraus bewegen. Die Solltrajektorie y^d wird daher eine Rampe mit der Steilheit $-17.17 \sin(10^\circ)$, die an den beiden Enden durch Übergangspolynome so in eine Konstante $\bar{y}^d$ überführt wird, dass die Trajektorie zweimal differenzierbar bleibt und die Spannung u_A nicht über ± 24 V geht. Durch die genauere Vorausplanung kann der Regler schwächer mit

$$c_0 = 16.42, \quad c_1 = 2.650, \quad c_2 = 9.972$$

Abb. 5. Testergebnis des linearen Reglers mit I-Anteil

Abb. 6. Testergebnis des flachheitsbasierten Trajektorienfolgereglers

eingestellt werden. Das Ergebnis sieht man in Abb. 6. Den geplanten Trajektorien wird gut gefolgt. Durch die Planung ist man in der Lage, die Grenzen des Systems auszunutzen und gleichzeitig einzuhalten.

5. Zusammenfassung

Die vorgestellte Reglerimplementierung ist einer gängigen, kommerziell erhältlichen Implementierung sowohl in der Hardware als auch in den Algorithmen im Ergebnis gleichwertig. Die Flexibilität, die man zum Experimentieren benötigt, ist voll gegeben. Man muss allerdings einräumen, dass Aufbau und Programmierung vertiefte Kenntnisse der jeweiligen Techniken verlangen. Dem gegenüber steht jedoch eine erhebliche Reduktion der Kosten, was diese Vorgehensweise vor allem für Ausbildungseinrichtungen interessant machen dürfte.

Danksagung

Open access funding provided by Johannes Kepler University Linz. Diese Arbeit wurde im Rahmen eines K2 Projektes in der Linz Center of Mechatronics GmbH durchgeführt. Die K2 Projekte werden aus Mitteln des österreichischen COMET-K2 Programms finanziert. Die COMET K2 Projekte im LCM werden unterstützt von der Österreichischen Bundesregierung, dem Land Oberösterreich, der Johannes Kepler Universität und allen wissenschaftlichen Partnern, die Teil des K2-COMET Konsortiums innerhalb der Linz Center of Mechatronics GmbH sind.

Autoren

Sven-Olaf Lindert

Jahrgang 1974, war bis zum 28.02.2017 Assistent am Institut für Regelungstechnik und Prozessautomatisierung der Johannes Kepler Universität Linz. Hauptarbeitsgebiete: Modellierung und Regelung von linearen und nicht-linearen konzentriert-parametrischen Systemen im Hinblick auf Industrieanwendungen.

Christian Höfler

Jahrgang 1982, ist Techniker am Institut für Regelungstechnik und Prozessautomatisierung der Johannes Kepler Universität Linz. Hauptarbeitsgebiete: Automatisierungssysteme mit offener Hard- und Software.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Literatur

1. Xenomai (2016). <http://www.xenomai.org/>.
2. Adamy, J. (2014): Nichtlineare Systeme und Regelungen, 2. Aufl. Berlin: Springer Vieweg. <https://doi.org/10.1007/978-3-642-45013-6>. ISBN 987-3-642-45012-9.
3. Eaton, J. W., et al. (2016): GNU octave, version 4.0.3. <https://www.gnu.org/software/octave/>.
4. Fliess, M., Levine, J., Martin, P., Rouchon, P. (1995): Flatness and defect of nonlinear systems: introductory theory and examples. Int. J. Control, 61(6), 1327–1361. <https://doi.org/10.1080/00207179508921959>.
5. Isidori, A. (1989): Nonlinear control systems. 2. Aufl. Berlin: Springer.
6. Jakubczyk, B., Respondek, W. (1980): On linearization of control systems. Bull. Acad. Pol. Sci., Sér. Sci. Math. Astron. Phys., 28, 517–522.
7. Lunze, J. (2013): Regelungstechnik 2. Berlin: Springer Verlag.
8. van de Maele, P. J. (2013): Low level programming of the Raspberry Pi in C. <http://www.pieter-jan.com/>.
9. Scilab Enterprises: Scilab (2016). <http://www.scilab.org/>.
10. Weichinger, K. (2011): Basic input output elements. <http://bioe.sourceforge.net/>.
11. Weichinger, K. (2016): A nonlinear model-based control realized with an open framework for educational purposes. <http://static.lwn.net/images/conf/rtlws-2011/proc/Weichinger.pdf>.

Kurt Schlacher

Jahrgang 1955, ist Institutsvorstand am Institut für Regelungstechnik und Prozessautomatisierung der Johannes Kepler Universität Linz. Hauptarbeitsgebiete: Modellierung und Regelung von nichtlinearen konzentriert- und verteilt-parametrischen Systemen im Hinblick auf Industrieanwendungen und unter Verwendung von differentialgeometrischen und Computer-Algebra-basierten Methoden.