Strampelnder Tux

Christoph Reichenbach, Christian Rüdiger, Markus Weimer

25. Mai 2003

Zusammenfassung

Embedded Systems auf Linux-Basis sind aktuell eines der Boom-Gebiete des Open Source Systems. Linux findet sich z.B. in der Steuerung von Industrieanlagen, auf kleinst-Webservern oder auch in jüngster Zeit auf PDAs. Eben diese Geräte, die Persönlichen Digitalen Assistenten, finden seit einiger Zeit eine immer grössere Verbreitung. Im Rahmen des Projekts Läufer wurden an der TU Darmstadt diese beiden Bereiche kleiner, eingebetteter Systeme zusammengeführt: Die Steuerung eines innovativen Fahrzeugs erfolgt durch einen Linux-PDA.

Bei diesem Fahrzeug handelt es sich um ein von seinen Entwicklern „Läufer" getauftes Tandem, das sich durch die Vielzahl an technischen Neuerungen auszeichnet, deren Entwicklung sich zwei Zielen unterordnet: Reisetauglichkeit und Fahrspaß.

Entwickelt wird das Fahrzeug von Studenten der TU Darmstadt, TU München sowie der FH Darmstadt in einem sehr interdisziplinären Team. Beteiligt sind neben Maschinenbauern verschiedener Spezialisierungen auch (Wirtschafts-)Informatiker, Textil- und Industrie-Designer. Ziel des Projektes ist die Entwicklung eines muskelgetriebenen *Reise*fahrzeugs für zwei Personen. Der Schwerpunkt liegt hier auf der Reisetauglichkeit, und nicht auf Dingen wie der theoretisch erreichbaren Maximalgeschwindigkeit. Neben der technischen Herausforderung sind das Lernen in anderen Strukturen sowie die intensive Zusammenarbeit mit Partnern aus Industrie und Forschung Triebfedern für das Projekt.

Ein wichtiger Schritt, die Ziele Fahrspaß und Reisetauglichkeit zu erreichen, ist die Gewichtsreduktion. So ist der Läufer komplett aus Aluminium und Kohlefaser gefertigt, was sein Gesamtgewicht auf ca. $50kg$ reduziert. Der Läufer verfügt im Gegensatz zu vielen anderen muskelgetriebenen Fahrzeugen über eine Verkleidung. Diese dient zum einen

der Verbesserung der Aerodynamik, zum anderen aber auch als Regenschutz. Dies zusammen mit dem Antriebsstrang, auf den weiter unten eingegangen werden soll, ermöglicht es zwei normal trainierten Menschen mit dem Fahrzeug eine Durchschnittsgeschwindigkeit von ca. $40km/h$ zu erreichen. Dieser errechnete Wert bezieht sich allerdings nicht etwa auf eine besonders ebene Route, sondern auf eine Strecke im Odenwald rund um Darmstadt, also im Mittelgebirge.

Insbesondere im Bereich der Mechatronik, also der Verbindung von Mechanik mit moderner Informationstechnologie, geht der Läufer nicht nur für Fahrrad-Verh ältnisse neue Wege.

Mechatronische Komponenten im Läufer

Mechatronische Systeme am Läufer sind der leistungsverzweigte Hybridantrieb, das Energiespeichersystem, die intelligente Beleuchtung sowie weitere funktionsrelevante Systeme. Aber das System ist auch für weitere Komponenten offen, die dann sehr einfach in die bestehende Infrastruktur integriert werden können. Im Projekt diskutiert wurden die Erweiterung um ein GPS oder ein eigenes Display für den Beifahrer.

Für die Elektronik dieser mechatronischen Komponenten wurde in enger Kooperation mit der Firma ce.tron[?] eine Platine mit vielfältigen Anschlussmöglichkeiten entwickelt. Zu diesen Möglichkeiten gehören beispielsweise Anschlüsse für Motoren und Sensorik. Die Steuerung der Anschlüsse erfolgt durch einen PIC-kompatiblen Microcontroller. Die mittels dieser Platine gesteuerten mechatronischen Geräte kommunizieren dann über einen CANBus[?][1] mit der zentralen Steuereinheit, also im Falle des Läufers dem Linux-PDA.

Scheckkarten mit Funktion

Soll nun eine neue Komponente in das System integriert werden, so enthält die Platine bereits viele fertig implementierte Funktionen um Eingänge abzugreifen, auszuwerten und an den PDA weiterzuschicken. Das Rad muß also nicht jedes mal neu erfunden werden. Mindestens die Kommunikation mit anderen wird mit bereits auf der Platine implementierten Funktionen abgewickelt. Der Entwickler muß dazu nur die von ihm benötigten unter den vorhandenen Routinen aufrufen.

Auf dieser untersten Ebene der Mechatronik sollen jedoch nur die nötigsten Entscheidungen getroffen werden. Das Zusammenspiel der Kompo-

[1] Controller Area Network

nenten wir über den PDA kontolliert und ausgewertet. Ein Beispiel ist die Platine mit deren Hilfe die Fahrzeugstütze des Läufers implementiert ist. Sie schaltet den Motor zum Stütze Ausfahren ab, wenn die Sensorik durch Beobachten von Stromspitzen am Motor einen Widerstand registriert. Ob die Stütze jedoch überhaupt ausgefahren werden sollwird im PDA entschieden und als Befehl über den CANbus an diese Platine gesendet.

All diese Komponenten des Läufers möchten die Fahrer nun zentral und vor allem intuitiv steuern, um z.B. die Geschwindigkeit abhängig vom eigenen Erschöpfungsgrad und dem der Akkus einzustellen. Aber nicht nur dies, es müssen auch Fahrzeugdaten aufbereitet werden. Diese Informationen werden aus den Fahrzeugkomponenten via CANBus gewonnen, und für den Fahrer in der Situation angemessen präsentiert. Ein einfaches Beispiel für solche Fahrdaten ist die aktuelle Geschwindigkeit, die aus dem Antriebsstrang gewonnen wird. Die Anforderungen an das Fahrerinformationssystem des Läufers sind also:

Kommunikation mit dem CANBus Um die Informationen von den verschiedenen Geräten zu erhalten und diesen Befehle zu schicken.

Rechenleistung Um die Daten zu verarbeiten und die Berechnung komplexer Zusammenhänge nicht in Assembler auf den Platinen durchführen zu müssen.

Display Um die Informationen an den Fahrer weiterzugeben. Diese Informationen sind z.B. bekannte Größen wie die aktuelle Geschwindigkeit, aber auch Läufer-Spezifika wie der Ladestand der Batterien.

Über Rechenleistung und Display verfügen heutige PDAs in mehr als ausreichendem Masse. Aus diesem Grund lag die Entscheidung nahe, einen solchen als „Bordcomputer" einzusetzen. Nachdem erste Versuche mit einem Palm IIIx sehr ermutigend waren, wurde das Team in der Ansicht bestätigt, diesen Weg weiter zu verfolgen. Dies geschah entgegen der verbreiteten Meinung, Consumer Elektronik habe im Embedded-Bereich nichts verloren.

Im Laufe der Entwicklung kam allerdings im Team der Wunsch nach einer flexibleren Lösung als PalmOS auf. Da PalmOS, das Betriebssystem der Palm-PDAs, in seinen aktuellen Versionen nicht über Multitasking verfügt, hätte jede Form von Erweiterung des Läufers um neue Komponenten Änderungen am Fahrprogramm erfordert. Schließlich kann man ohne Multitasking nicht einfach eine weitere Software installieren, die ebenfalls während der Fahrt zur Ausführung kommen soll. Eine Abänderung der sowieso schon laufenden Fahrsoftware sollte aber aus Sicherheitsgründen

vermieden werden. Da das Projekt aber seiner Struktur nach offen für neue Entwicklungen bleiben will, kam ein ausschließen zukünftig hinzukommender Funktionalität nicht in Frage. So kam die Forderung nach einem Multitaskingfähigen PDA auf. Zur Wahl standen die üblichen Verdächtigen: Also WindowsCE/PocketPC und seit neuestem auch Linux.

Das MechatronikTeam in Darmstadt entschied sich f ür Linux, da es im Bereich der Programmierung schon über mehr und tiefergehende Erfahrungen mit Linux als mit WindowsCE verfügte. Ein weiterer Vorteil von Linux auf dem PDA ist, daß die darauf laufenden Anwendungen in der Regel Sourcecodekompatibel zu LinuxPCs bleiben. Man kann also eine Anwendung am PC entwickeln und testen, um sie dann später durch einfaches Neuübersetzen in eine PDAAnwendung zu verwandeln. Dies ist ein nicht zu unterschätzender Vorteil, gerade vor dem Hintergrund der verteilten Produktentwicklung im Projekt. Bei dieser werden Teile der Mechatronik an der TU München entwickelt, wo bei dieser Auslegung der Mechatronik kein PDA für die Entwicklung nötig ist.

Diese Methode kommt insbesondere auch ohne Emulation des PDA auf dem PC aus. Dies stellt sicher, daß die Anwendung auf dem PC eine genauso direkte Verbindung zu den Schnittstellen hat wie später auf dem PDA, ohne daß dem eine eventuelle Emulation des PDA eine weitere Hürde in den Weg stellt.

Im Projekt wurde zunächst ein mittels der LinuxDistribution Familiar und der graphischen Oberfläche OPIE [**?**] auf Linux umgestellter Compaq IPaq H3660 verwendet. *SHARP!?!*

Programmiert werden beide PDAs in C++ unter Nutzung der Klassenbibliothek QT des norwegischen Herstellers Trolltech. Diese Bibliothek liegt z.B. auch dem erfolgreichen UNIXDesktop KDE zu Grunde und ermöglicht es, Software für Windows, *ix, MacOS und nun auch LinuxPDAs zu entwickeln. Die dabei enstehende Software ist bei gutem Programmierstil sogar durch einfaches Neuübersetzen von einer auf die andere Plattform portabel. So wurde z.B. im Projekt Läufer Software für den PDA auf PCs und Workstations entwickelt, bevor der erste PDA zur Verfügung stand. Bei der Gestaltung der Oberfläche des Programms muß man dann allerdings die Einschränkungen des PDAs hinsichtlich Texteingabe und DisplayAufl ösung berücksichtigen.

1 Die Verbindung zwischen PDA und Mechatronik

All dies sind interessante Experimente für sich– die Platine und ihre Programmierung wie der PDA als Bedienelement– doch damit das bunte Bild der Bedienelemente nicht blosse Façade bleibt und die Mechatronik vom Willen des Fahrers erfahren kann, ist eine Kommunikationsschicht nötig.

1.1 Kommunikation

Für das Projekt Läufer war von vorneherein die Entscheidung für ein verkabeltes System gefallen. Auf Protokollseite wird hierbei auf CAN (Controller Area Network, siehe [?]) gesetzt, ein verhältnismäßig einfaches und verbreitetes Kommunikationsprotokoll, das unter anderem von BMW und Mercedes zur Datenübertragung innerhalb von Fahrzeugen eingesetzt wird.

Ein unmittelbarer Vorteil der Wahl eines bereits verbreiteten Protokolls ist das Vorhandensein existierender Hardware-Implementierungen, von denen eine [– FIXME: der sowieso-Chip –] hier die Aufgabe eines Kommunikationsteilnehmers wahrimmt.

Das CAN-Protokoll gew ährleistet Korrektheit der übertragenen Daten, erzwingt Arbitrierung der auf dem Bus gesandten Kommunikation und erlaubt theoretisch 2048 logische und eine beliebige Anzahl physikalischer Kommunikationsteilnehmer.

1.1.1 Ergänzungsprotokolle

Zwei weitere Probleme jedoch löst CAN nicht: Übertragungssicherheit, also die Gewährleistung, daß der Sender dem Erfolgen einer Übertragung erfährt, ob diese beim Empfänger ankam, und die Überprüfung des Vorhandenseins des vorgesehenen Bus-Masters, ein insbesondere durch die problemlose Entfernbarkeit des PDAs relevantes Problem.

Die Lösungen erfolgten auf höherer Protokollebene, die in Form des eigens entworfenen LLO-Protokolles (L äufer Layer One) entstand. Zwar existiert mit CAN/Open (siehe [?]) bereits ein etabliertes und umfangreiches Standardprotokoll zur Kommunikation auf dem CAN-Bus, das jedoch für eine Implementierung mit der gewählten Hardware und in dem gegebenen Rahmen zu umfangreich gewesen wäre.

1.2 Die LLO- und LLZ-Protokolle

LLO setzt direkt auf dem CAN-Protokoll auf und kodiert seine Operationen in Teilen des CAN-Adressfeldes, so daß sich die Anzahl der adressierbaren Geräte auf 32 vermindert. Zwei dieser Adressen sind wiederum reserviert, eine für den Bus-Master, eine f ür Broadcasts. Die Notwendigkeiten des Läufers fordern zur Zeit keinen größeren Adressraum[2]. Mittels eines regelmäßigen Keep-Alive-Signals stellt LLO sicher, daß die Sklaven im Bus einen Ausfall des Bus-Masters bemerken; die andere noch fehlende Anforderung, die Korrektheit von Übertragungen, wird allerdings durch ein zweites Protokoll erfüllt: LLZ (Läufer Layer Zero), konrastierend zu den üblichen Konventionen benannt, liefert dies und auch eine allgemeine Zustandsverwaltung der Busteilnehmer.

1.2.1 Ausgewählte Eingeweide

Dabei werden vom Protokoll einige Zustände gefordert, insbesondere ein stabiler Einzelbetriebszustand, in dem das angesteuerte Gerät eingehende Nachrichten ignoriert. Dieser wird eingenommen, wenn, zum Beispiel durch mehrfaches Ausbleiben eines Keep-Alive-Signals vom Busmaster, das Gerät davon ausgehen muß, daß es nicht mehr unter der Aufsicht des PDAs steht und eine Fehlfunktion vorliegt. Wie genau dieser sichere Betriebszustand tatsächlich aussieht, hängt vom Gerätetyp ab; der Motor beispielsweise nimmt in diesem Fall die Drehzahl langsam zurück, eine vorhandene Warnblink-Anlage w ürde sich automatisch aktivieren.

Zur Sicherstellung der korrekten Übertragung einer Botschaft werden (auf LLO-Ebene) vierbittige Sequenzz ähler verwendet, die sich nach jeder erfolgreichen Botschaft um eins inkreminieren. Da die Inhalte der Zähler in LLO-Botschaften einkodiert werden, kann der Empf änger bei Erhalt einer Nachricht seine Synchronität zum Bus-Master mit einem hohen Grad an Sicherheit überprüfen.

1.3 Aus dem Tagebuch einer Nachricht

Begleiten wir nun einmal eine konkrete Nachricht auf ihrer Reise auf dem Bus. Eine typische Botschaft wäre eine Aufforderung an einen Scheibenwischer (`SW`), sich selbst einzuschalten (`ON`) und die Scheibe mit einer bestimmten Geschwindigkeit (`FR`) zu putzen.

Beobachten wir nun die Reise dieser Botschaft.

[2] Durch geeignete Versionnierung wurde jedoch spätere Erweiterbarkeit sichergestellt

Master	Seq	Seq	Client	
ON FR				Rohform
REQ:2 ON FR				LLZ-Encodierung
STX:SW[ON FR]	0	0		LLO-Encodierung
STX:SW[ON FR] →	0	0		LLO-Versand
	1	0	↛	Übertragungs-fehler!
STX:SW[X] →	1	0		Neuer Versand
	2	0	→STX:SW[X](1)	Seq-Fehler
	2	0	←ERR:EINSQ(0)	Fehlermeldung
ERR:EINSQ(0)←	0	0		Reset
STX:SW[ON FR] →	0	0		Neuversand
	1	1	→STX:SW[ON FR]	Übertragung erfolgreich!
	1	1	←ACK(0)	Bestätigung der Übertragung...
ACK(0) ←	1	1		...an den Master
	1	1	STX:SW[ON FR]	Dekodierung nach LLZ
			REQ:2 ON FR	Übertragung erfolgreich!
			ON FR	Auswertung und...
			RES:1 ON	...Konstruktion der Antwort

Im obigen Beispiel sehen wir– neben den Schritten der Kodierung und Dekodierung des Paketes– auch die Behandlung eines Fehlerfalles: Durch z.B. eine temporaere Leitungsstoerung ging ein Paket verloren. Auf der LLZ-Ebene des Masters wird über versandte Pakete Protokoll gefuehrt, so daß das Nichtantworten auf eine LLZ-Anfrage registriert und durch einem Neuversand behandelt werden kann; das erfordert natürlich eine bestimmte Form von Botschaften, nämlich solche, die Zustände setzen, anstatt sie zu modifizieren. Unter Umständen können jedoch schon vorher auf LLO- bzw. LLZ-Ebene Fehlübertragungen erkannt werden, da beide Ebenen Bestätigungen der versandten Nachrichten erwarten. Infolge dessen, daß diese jedoch nicht unmittelbar in Konsequenz der Übertragung geschehen müssen, ist es möglich, daß davor schon eine oder mehrere weitere Botschaften übertragen wurden; dies wird durch Vergleich der Sequenznummern von Client und Master, die in jeder LLO-Übertragung einkodiert sind, erreicht.

In diesem hier illustrierten Fall fand eben diese letztbeschriebene Erkennung statt. Der zurückgesandte Fehlercode, `EINSQ(0)`, beschrieb zum Einen die Art des Fehlers, eben gerade die unpassende Sequenznummer, war zum Anderen jedoch auch mit der clientseitig verwendeten Nummer parametrisiert– diese Information war somit ausreichend, das letzte fehlende Paket neu zu senden und die Übertragung an der Fehlerstelle neu aufzunehmen.

Fazit

Im Rahmen des Projekts Läufer ist ein Mechatronik-Framework entstanden, dass explizit von Informatikern für Ingenieure entwickelt wurde. Dies ermöglicht es den Ingenieuren im Projekt, ohne grossen Aufwand die Software für mechatronische Komponenten zu entwickeln. Die Eignung für diesen Einsatz wurde im Rahmen eines Workshops für Studenten und Wissenschaftler der TU München unter Beweiss gestellt. Im Rahmen dieses Workshops war es den Teilnehmern möglich, nach drei Tagen eigene mechatronische Komponenten zu entwickeln.

Das Projekt hat gezeigt, dass der Einsatz von Consumer Elektronik in Bereichen, in denen normalerweise Spezialentwicklungen zum Einsatz kommen, möglich und auch sinnvoll ist. Ermöglicht wurde dies nicht zuletzt durch die Verbreitung von Linux auf nahezu allen relevanten Rechnerarchitekturen.