Eine leistungsfähige Hard- und Softwarearchitektur für kognitive Funktionen in Fahrzeugen

Matthias Goebl*, Florian Rattei*, Stephan Neumaier* und Georg Färber†

Zusammenfassung: Zukünftige Fahrerassistenzsysteme erfordern eine hohe Verlässlichkeit der Wahrnehmung, die oftmals nur durch Einbeziehung verschiedener Informationsquellen erreicht werden kann. Dies verlangt eine enge Vernetzung der Funktionen und Mehrfachnutzung von Sensoren. Für die Versuchsträger des SFBs „Kognitive Automobile“ wurde eine echtzeitfähige Hard- und Softwarearchitektur entwickelt, die eine schnelle Verbindung kognitiver Funktionen erlaubt und ihnen einen ganzheitlichen Zugriff auf das im Fahrzeug gesammelte Umgebungswissen bietet. Ausgewählte Anwendungen veranschaulichen die Ergebnisse.

Schlüsselwörter: Systemarchitektur, Fahrerassistenz, Umfeldwahrnehmung, Realzeitsystem

1 Einleitung

Heutige Fahrerassistenzsysteme (FAS) wie ACC und LDW werden bevorzugt als separate, geschlossene Steuergeräte (SG) aufgebaut, die ihren primären Sensor wie RADAR oder eine Kamera integriert haben. Die Architektur des SG ist kostenminimal und auf die konkrete Funktion optimiert. Die informationstechnische Integration geschieht durch Verbinden mit einem der vorhandenen Fahrzeugbusse wie CAN. Dies hat den Vorteil, dass die FAS-Funktionen als optionale Sonderausstattung verkauft werden können und den Grundpreis des Basismodells nicht in die Höhe treiben. Allerdings zeigt sich, dass Funktionen zunehmend voneinander abhängig sind, so dass die Wahl einer höherwertigen Funktion die Hinzunahme einer Anzahl von Basisfunktionen erfordert.

In zukünftigen FAS wird der Trend zu einer engen Vernetzung der Funktionen weiter zunehmen: Gerade sicherheitsrelevante Funktionen erfordern eine hohe Verlässlichkeit der Wahrnehmung, die oftmals nur durch Einbeziehung verschiedener Informationsquellen erreicht werden kann. Die bestmögliche Erkennungsleistung ergäbe die Fusion der Information aller verbauten Sensoren, jeweils ausgewertet mit funktionsspezifischen Algorithmen. Dies setzt jedoch die Verfügbarkeit aller Informationen einschließlich der Rohdaten aller Sensoren voraus, was in der heutigen Fahrzeugarchitektur nicht vorgesehen ist. Eine funktionsunabhängige Architektur für alle Fahrerassistenzfunktionen mit den dazugehörigen Schnittstellen kann dies leisten und erlaubt sogar das Nachrüsten neuer Funktionen.

*Matthias Goebl, Florian Rattei und Stephan Neumaier sind wissenschaftliche Mitarbeiter am Lehrstuhl für Realzeit-Computersysteme (RCS), Technische Universität München, (e-mail: {goebl,rattei,neumaier}@rcs.ei.tum.de).

†Georg Färber ist Ordinarius des Lehrstuhls für Realzeit-Computersysteme (RCS), Technische Universität München (e-mail: faerber@rcs.ei.tum.de).

Der Sonderforschungsbereich/Transregio 28 „Kognitive Automobile“ [9, 10] stellt ähnliche Anforderungen an eine Architektur. Ein kognitives Automobil muss basierend auf der Wahrnehmung seiner Umgebung seine eigene Situation verstehen und sich angemessen verhalten. Dies wird eindrucksvoll durch autonomes Fahren demonstriert. Die Forschungsergebnisse, wie das erreichte Umgebungsverständnis, nutzen jedoch auch mittelfristig aktiven Fahrerassistenzsystemen. Die Software der Fahrzeuge setzt sich aus vielen Einzelmodulen zur Kognition zusammen, die zur Demonstration in ein Gesamtsystem integriert werden müssen. Dabei müssen zudem enge Zeitanforderungen eingehalten werden.

Dieser Herausforderung wird im vorliegenden Beitrag mit einer aufeinander abgestimmten Hard- und Softwarearchitektur begegnet. Sie trägt dazu bei, das im Fahrzeug gesammelte Wissen einschließlich aller Rohdaten effizient zu verwalten und zu verteilen, so dass alle Funktionen maximalen Nutzen daraus ziehen können. Es wird dabei kein bestimmter Sensor oder Fusionsansatz bevorzugt, so dass verschiedenste Algorithmen im Forschungsbereich erprobt und die erfolgreichsten kombiniert werden können.

2 Hardwarearchitektur

Die zugrundeliegende Hardwarearchitektur muss den Softwaremodulen der genannten Funktionen zum einen die erforderliche Rechenleistung zur Verfügung stellen und zum anderen ihre enge Kopplung ermöglichen. Bei Verwendung vernetzter Rechner können zwar beliebige Leistungsstufen realisiert werden, bei intensiver Kommunikation von Modulen über Rechnergrenzen hinweg können sich jedoch die enstehenden Kommunikationstotzeiten zum beschränkenden Faktor entwickeln. Zudem müssen Daten dafür serialisiert und notwendigerweise priorisiert werden. Für das Ziel eine enge Kooperation zu fördern, darf kein nennenswerter Kommunikationsaufwand entstehen, um im Fahrzeug verfügbares Wissen abzurufen. Eine Partitionierung in mehrere Rechnersysteme erhöht zudem die Gefahr der Entstehung von Inselsystemen, welche die Entwicklung neuer übergreifender Funktionen erschweren.

Daher bildet den Kern des Entwicklungssystems ein zentraler Multicore-Multiprozessor-Rechner, auf dem alle Wahrnehmungsmodule ausgeführt werden. In [2] wurden moderne Multiprozessorarchitekturen verglichen und der AMD Opteron als die am besten skalierende x86-Architektur ausgewählt, in der weitere Prozessoren ohne spezieller Chipsatzunterstützung hinzugefügt werden können. Die Opteron-Prozessoren haben jeweils einen eigenen Speichercontroller und sind untereinander über HyperTransport verbunden mit einer nominellen Bandbreite von $3.2 \cdot 10^9 Byte/s$ je Verbindung und Richtung[1]. Die Latenz für einen Speicherzugriff[2] liegt deutlich unter 1μs.

Diese Konfiguration hat klare Vorteile zu einem Rechnernetzwerk: Schon bei unbelastetem Ethernet liegen dort die Latenzzeiten in einer Größenordnung von 100μs bei einer nominalen Bandbreite von maximal $10 \cdot 10^9 Bit/s$. Zudem ist für jeden Datenaustausch zusätzlicher Rechenaufwand auf beiden Seiten einer Kommunikationsbeziehung notwen-

[1] Eigene Messungen zeigen einen nutzbaren Speicherdurchsatz von $3.11 \cdot 10^9 Byte/s$ für den eigenen Speicher und $2.62 \cdot 10^9 Byte/s$ für den entfernten Speicher auf einem Dual-AMD Opteron 275HE (2.2GHz, DDR-400 RAM) bei der Ausführung einer 32-bit linearen Read-Modify-Write-Schleife über einen Speicherbereich von $8 \cdot 10^9 Bytes$.

[2] In [6] wurde die Latenz mit $110ns$ gemessen und in [11] mit $330ns + 130ns \cdot (n-1)$ für n aufeinanderfolgende Zugriffe bestimmt, bei leicht unterschiedlichen Taktfrequenzen

dig. Hier beschränken sich die hardwarebedingten Kommunikationszeiten auf die geringe Latenzzeit für den Zugriff auf den gemeinsamen Hauptspeicher, so dass in Kombination mit einer effizienten Softwareimplementierung umfangreicher Datenaustausch in Echtzeit möglich ist.

Dieses „Cluster-in-a-Box"-System benötigt zudem Infrastrukturkomponenten wie Festplatten oder Konsole nur einmal. Der Leistungsbedarf liegt bei ca. 350W aus dem Bordnetz, wovon 160W auf das Motherboard mit Prozessoren und Speicher entfallen. Unter Verwendung kommender Generationen von Multicore-Prozessoren ist vorstellbar, diese Entwicklungsplattform bei Beschränkung auf ausgewählte Funktionen auf Steuergerätegröße zu schrumpfen.

3 Softwarearchitektur

Die eingesetzte Softwarearchitektur muss auf die Hardware abgestimmt sein. Insbesondere die Interprozesskommunikation muss schnell, effizient und ressourcenschonend verlaufen. Verfügbare Lösungen wie CORBA weisen oftmals einen beachtlichen Overhead auf und sind nicht gleichzeitig für große Datenmengen wie Videodatenströme und hohe Reglerdatenraten im kHz-Bereich geeignet.

Daher wurde die „Realzeitdatenbasis für kognitive Automobile" (KogMo-RTDB) als zentrales Kommunikationsframework für alle kognitiven Funktionen entwickelt. Der zugrundeliegende Ansatz ist datenzentrisch: Jedes Softwaremodul kann Daten veröffentlichen und sie so anderen Modulen verfügbar machen. Dies ergibt eine ganzheitliche Sicht auf alle im Fahrzeug vorhandenen Daten und das damit repräsentierte Umgebungswissen.

Im Gegensatz zu den oftmals unbekannten maximalen Ausführungszeiten t_{WCET} der in der Wahrnehmung eingesetzten Algorithmen ist der Zeitaufwand für den Zugriff auf die hinterlegten Daten in der KogMo-RTDB echtzeitfähig und vorhersagbar: Selbst bei einem stark ausgelasteten System wurde eine maximale Zugriffszeit von $62\mu s$ gemessen [3]. Für alle Daten muss zudem eine minimale zeitliche Verweildauer spezifiziert werden, die von der KogMo-RTDB durch ein System von Ringpuffern sichergestellt wird. So kann zum einem garantiert werden, dass beispielsweise Module höherer Wahrnehmungsebenen, die sich im Allgemeinen durch lange Zykluszeiten auszeichnen, auf Daten von Modulen mit kurzen Zykluszeiten problemlos zugreifen können. Zum anderen verhindert die KogMo-RTDB auch, dass schnelle Wahrnehmungs- und Regelungsprozesse, die ihre Daten über die RTDB austauschen, durch langsamere Module blockiert werden.

Die kleinste von der RTDB verwaltete Einheit ist ein RTDB-Objekt, ein definierter Container, der eine Menge zusammengehöriger Daten aufnimmt. Dazu kann bei Bedarf auch ein C++-Objekt bereitgestellt werden. Jeder Objektdefinition wird eine eindeutige Typ-ID zugeordnet. Das Anlegen eines Objektes in der RTDB erfordert zudem einen beliebigen Namen und die Objekt-ID des Vaterobjekts, sollte der Aufbau eines Szenenbaums nach [1] gewünscht sein. Die Publikation von RTDB-Objekte dient als offene Schnittstelle zwischen allen Modulen. Beispiele von RTDB-Objekten sind:

- *Rohdaten von Sensoren:* GPS/IMU-Position, Videobilder, Kameraparameter (Kalibrierung, Verschlusszeiten), LIDAR-Messwerte, Positionsdaten einer aktiven Kameraplattform, Fahrzeugdaten (Geschwindigkeit, Lenkwinkel, Raddrehzahlen), ...
- *Ergebnisse der Umfeldwahrnehmung:* Fahrspurschätzwerte, verfolgte Fahrzeuge, Hindernisse, Visualisierungselemente, Flussvektoren im Bild, ...

- *Ergebnisse höherer Wahrnehmungsebenen:* klassifizierte Objekte, bewertetes Fahrzeugumfeld, analysierte Situation, eigene Verhaltensentscheidung, ...
- *Steuergrößen:* Lenkrate, Beschleunigung, Fahrkorridor, Blickrichtungswunsch, ...

Durch die konsequente Nutzung von Zeitstempeln ist zudem eine konsistente Sicht auf alle Daten gewährleistet: Bei jeder Aktualisierung eines Objekts wird dieses mit einem *Commit-Timestamp* t_c versehen, der den Zeitpunkt der Übergabe der Daten an die RTDB markiert. Zusätzlich muss der *Data-Timestamp* t_d auf den Zeitpunkt gesetzt werden, zu dem die Daten entstanden, beispielsweise dem mittleren Belichtungszeitpunkt. Bei der Weiterverarbeitung des Bildes muss t_d in die Ergebnisobjekte wie z.B. das Fahrspurobjekt übernommen werden. Anhand von t_c und t_d lässt sich in der Historie der Daten navigieren, beispielsweise um fehlende Werte zu interpolieren. Für das autonome Fahren lässt sich bei Ausfall oder Zeitüberschreitung eines Wahrnehmungsmoduls anhand der letzten verfügbaren Umfelddaten eine gelenkte Notbremsung einleiten.

3.1 Leistungsfähigkeit

Um harten Echtzeitbedingungen zu genügen ist die KogMo-RTDB rein hauptspeicherbasiert. Sie ist in der Lage, sowohl große Datenobjekte wie Kamerabilder mehreren Prozessen zu liefern, als auch gleichzeitig Fahrzeugdaten mit einer Frequenz von 1 kHz weiterzuleiten. Die effiziente Implementierung erlaubt aktuell[3] 120481 Aktualisierungs- und 217391 Abfrageoperationen pro Sekunde, abhängig von der Objektgröße und der Zugriffsmethode [4]. Die Interprozesskommunikationszeit, gemessen zwischen der Aktualisierung eines Objektes durch einen ersten Prozess, dem Aufwachen eines zweiten Prozesses und der Abfrage des Objektes beträgt durchschnittlich $29.6\mu s$ [3].

Alle in der KogMo-RTDB hinterlegten Daten können einschließlich ihres zeitlichen Verlaufs aufgezeichnet werden und beispielsweise im Labor wieder in eine laufende Datenbank eingespielt werden. Die Aufzeichnung beinhaltet auch das Anlegen und Löschen von Objekten zur Laufzeit, es werden auch dynamische Objekte wie erkannte Hindernisse berücksichtigt. Um mit aufgezeichneten Daten zu arbeiten, müssen die Verarbeitungsmodule nicht modifiziert werden. So können reproduzierbare Tests zur Algorithmenbewertung und -entwicklung durchgeführt werden. Tab. 1 enthält einen Auszug der Objektliste einer Aufzeichnung wie sie auf dem beschriebenen Entwicklungssystem auf einer „Raptor“ SATA-Festplatte von Western Digital mitprotokolliert wurde.

4 Anwendungen und Ergebnisse

Abb. 1 zeigt eine Modulkonfiguration des kognitiven Automobils, mit der auf dem Versuchsträger, einem Audi Q7, das Spurhalten mit bewegter Weitwinkelkamera demonstriert wurde. Der Videodatenstrom wird von einem dedizierten Modul *I/O Videodaten* in Einzelbildern in der RTDB abgelegt. Auf der verwendeten Hardwarearchitektur ist dies problemlos mit 4 Kameras gleichzeitig möglich. Die Rohdaten der Bilder stehen so allen Modulen zur Verfügung; es ist auch möglich, vorverarbeitete Bilddaten zusätzlich wieder in die RTDB zu stellen.

[3]Gemessen auf einem AMD Opteron 275HE bei einer Objektgröße von 152 Bytes und ohne direktem Pointerzugriff

Tabelle 1: Objekte in einer RTDB-Aufzeichnung mit resultierenden Bandbreiten

Objekttyp	Größe	Frequenz	Anzahl	Bandbreite
Fahrzeugstatus	100 Bytes	250 Hz	1	0.024 MB/s
Fahrzeugkommando	96 Bytes	25 Hz	1	0.002 MB/s
Plattformstatus	60 Bytes	10 Hz	1	0.0005 MB/s
Plattformkommando	68 Bytes	2 Hz	1	0.0001 MB/s
Kamerakalibrierung	76 Bytes	10 Hz	1	0.0007 MB/s
Kamerabild (640x480)	307248 Bytes	33 Hz	4	9.669 MB/s
Erkannte Fahrspur	184 Bytes	33 Hz	1	0.006 MB/s
Spurvisualisierung	388 Bytes	33 Hz	1	0.012 MB/s
IMU/GPS-Position	328 Bytes	10 Hz	1	0.003 MB/s
2D-LIDAR (Sick)	3008 Bytes	37.5 Hz	0	0.108 MB/s
3D-LIDAR (Velodyne)	1280124 Bytes	10 Hz	0	12.208 MB/s
Prozessstatus	76 Bytes	nach Funktion		
Gesamte Bandbreite				≈38.73 MB/s

Der *Fahrspurerkenner* [7] verfolgt die Fahrspur im Bild unter Verwendung eines Klothoidenmodells. Er bezieht sich dabei auch auf Fahrzeugdaten wie die Eigengeschwindigkeit. Diese liefert ein unterlagerter Fahrzeugregler auf einer dSpace Autobox über CAN und wird von dem Interfacemodul *I/O Fahrzeug* mit genauem Zeitstempel in der RTDB als *Fzg.-Status* hinterlegt.

Die *Fahrzeugkontrolle* errechnet aus den Parametern im Objekt *Fahrspur* zusammen mit den aktuellen Fahrzeugdaten aus *Fzg.-Status* die erforderlichen Steuergrößen, die sie in das Objekt *Fzg.kommando* schreibt. Das Modul *I/O Fahrzeug* verfolgt jede Änderung von *Fzg.kommando* und sendet die neuen Vorgaben zur Autobox.

4.1 Einbindung einer bewegten Kameraplattform

Für die Erkennung entfernter Objekte und und einen weiten Sichtbereich verfügt der Versuchsträger verfügt über eine aktive Kameraplattform [12]. Sie besteht aus zwei unabhängig gierbaren Weitwinkelkameras und einer nick- und gierbaren Telekamera. Ein Softwaremodul *I/O Kameraplattform* sorgt für eine transparente Anbindung: Die periodisch über CAN gemeldeten Achsposition werden in einem RTDB-Objekt publiziert, für eine Weitwinkelkamera beispielsweise in einem der Kamera zugeordneten Objekt *Ist-Gierwinkel*. So kann der *Fahrspurerkenner* den aktuellen Kameragierwinkel einfach aus der RTDB beziehen. Da Videobilder eine systematische Verzögerung aufweisen, muss bei der Abfrage der Datenzeitstempel t_d des jeweiligen Bildes verwendet werden, um den *Ist-Gierwinkel* aus der RTDB zu erhalten, der zum Zeitpunkt der Bildentstehung aktuell war.

Die Nutzung der in der RTDB enthaltenen Kurzzeithistorie aller Objekte verdeutlicht folgendes Beispiel: Da die Videobilder während eines Kameraschwenks verschmieren können, ist es sinnvoll, diese für die kurze Zeit der Bewegung nicht zu verwenden. Dazu müssen keine eigenen Schnittstellen definiert werden, vielmehr ist der Aufwand minimal, wenn die vorhandene RTDB-Historie der Kamerabewegungsdaten genutzt wird. Dazu wird unter Verwendung der gespeicherten *Ist-Gierwinkel* $\psi_{ist}(t)$ eine Größe *Kamera-*

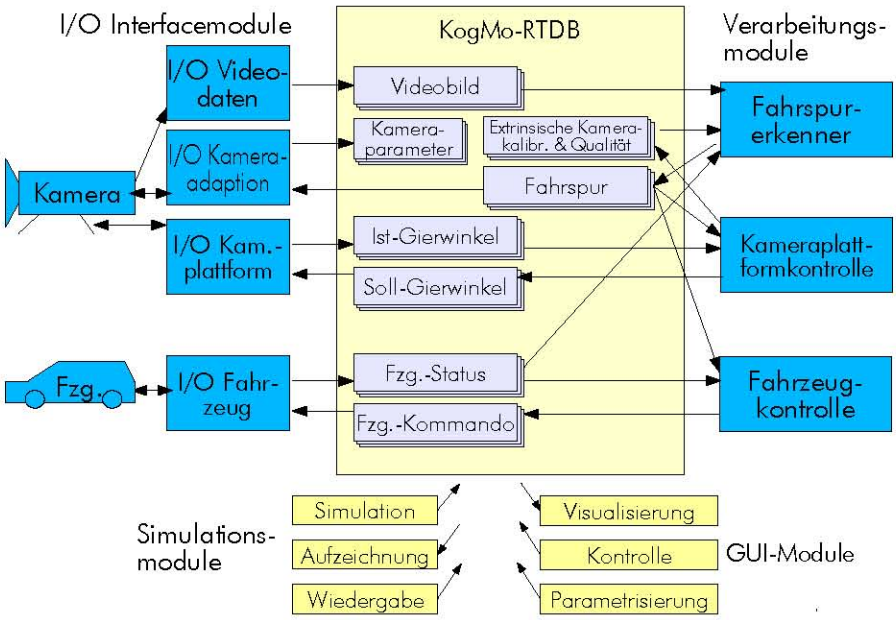

Abbildung 1: KogMo-RTDB mit Objekten und Datenfluß der Module

Ruhezeit $t_{idle}(t_0)$ zum Zeitpunkt t_0 mit

$$t_{idle}(t_0) = t_0 - t_{-n} \text{ wenn } \psi_{ist}(t_0) = \psi_{ist}(t_{-1}) = \ldots = \psi_{ist}(t_{-n}) \neq \psi_{ist}(t_{-n-1})$$

definiert. Diese kann zu jedem Zeitpunkt t nur aus den Daten in der RTDB berechnet werden. Daraus wird ein Qualitätsmaß $q(t)$ abgeleitet und einschließlich einer Beruhigungzeit von 50ms exemplarisch definiert zu

$$q(t) = \begin{cases} 0.95 & \text{für } t_{idle}(t) >= 50\text{ms} \\ 0.05 & \text{sonst} \end{cases}$$

Unter Verwendung von $q(t)$ wurde der Fahrspurerkenner so erweitert, dass er minderwertige Bilder mit z.B. $q(t) < 0.5$ nicht auswertet, aber die vergangene Zeit bei der Mitführung seiner Modelle berücksichtigt. Abb. 4.1 zeigt den Verlauf der genannten Größen beim Gieren einer Weitwinkelkamera von 0 auf 10 Grad.

4.2 Erweiterung um eine adaptive Belichtungssteuerung

Das folgende Beispiel zeigt, wie einfach in der bestehenden Architektur neue Funktionalitäten unter Verwendung vorhandener Daten hinzugefügt werden können: Zusätzlich zur Fahrspur liefert der *Fahrspurerkenner* ein Objekt, das zur Visualisierung Grafikprimitive mit ihren Positionen im Videobild enthält.

Die Positionen der Spurmarkierungen werden nun von einem Modul *I/O Kameraadaption* genutzt, das die Belichtungsregelung der Kameras so parametrisiert, dass nur für die Erkennung relevante Bereiche im Bild in die Regelung eingehen [8]. Die resultierenden Einstellungen und Belichtungswerte der Kamera werden anschließend in einem eigenen Objekt *Kameraparameter* publiziert und könnten wie oben gezeigt ebenfalls in ein Qualitätsmaß eingehen.

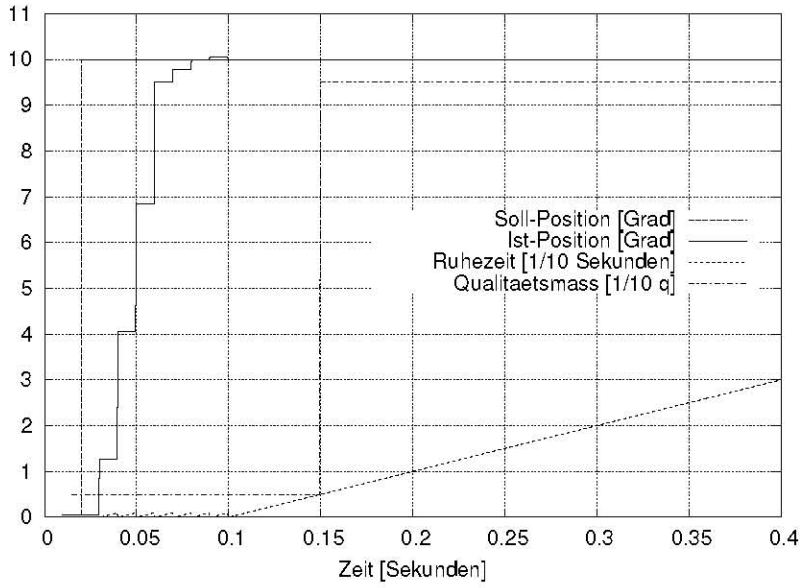

Abbildung 2: Gierbewegung einer Weitwinkelkamera

5 Sicherheit und Stabilität

Ein softwaregesteuertes bewegtes Fahrzeug stellt eine große Gefahrenquelle dar. Daher erhält die Stabilität und Robustheit der Softwarearchitektur einen hohen Stellenwert für die Sicherheit. Für alle Software auf dem Fahrzeugrechner wurde eine einheitliche Laufzeitumgebung festgelegt, die in identischer Form auch auf einem Laborrechner läuft. Module können so erst in der Simulation getestet werden, bevor sie im Fahrzeug betrieben werden.

Aus Sicherheitsgründen werden alle Module im Userspace des Betriebssystems Linux ausgeführt. Sie stehen damit unter dem Speicherschutz des Betriebssystems. Um ausgewählten Modulen harte Realzeitbedingungen zu garantieren, wird die Realzeiterweiterung Xenomai eingesetzt. Die KogMo-RTDB sorgt dafür, dass Realzeit- und nicht-Realzeitmodule blockierungsfrei kommunizieren. Realzeitmodule dürfen keine Schnittstellen des Standardbetriebssystems nutzen. Daher sollte bei RTDB-Modulen die graphische Benutzeroberfläche (GUI) vom eigentlichen Rechenprozess getrennt werden und wie in Abb. 1 gezeigt über die RTDB verbunden werden. Dies hat den weiteren Vorteil, dass beim Abspielen aufgezeichneter RTDB-Datenströme die Visualisierungsmodule ohne Modifikation eingesetzt werden können.

6 Fazit

Die erstellte Hard- und Softwarearchitektur wird auf den Versuchsträgern des SFBs erfolgreich eingesetzt. Sie ist leistungsfähig genug, alle anfallenden Daten einschließlich der Sensorenrohdaten gleichzeitig zu verarbeiten. Die offenen Schnittstellen ermöglichen die einfache Erweiterung um neue Wahrnehmungsmodule und deren enge Vernetzung. Der Einsatz im Fahrzeug des Teams AnnieWAY [5], das beim *DARPA Urban Challenge*, einem Wettbewerb für autonomes innerstädtisches Fahren, bis ins Finale kam, zeigte, dass die entwickelte Architektur auch für LIDAR-basierte Wahrnehmung einsetzbar ist.

7 Danksagung

Die Autoren danken der Deutschen Forschungsgemeinschaft (DFG) für die Förderung des Sonderforschungsbereich/Transregio 28 „Kognitive Automobile“ und allen Projektpartnern für die gute Zusammenarbeit.

Literatur

[1] Ernst D. Dickmanns. *Dynamic Vision for Perception and Control of Motion.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[2] Matthias Goebl, Sebastian Drössler, and Georg Färber. Systemplattform für videobasierte Fahrerassistenzsysteme. In *Autonome Mobile Systeme 2005*, pages 187–193. Springer-Verlag, 2006.

[3] Matthias Goebl and Georg Färber. A Real-Time-capable Hard- and Software Architecture for Joint Image and Knowledge Processing in Cognitive Automobiles. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 734–740, 2007.

[4] Matthias Goebl and Georg Färber. Eine realzeitfähige Softwarearchitektur für kognitive Automobile. In *Autonome Mobile Systeme 2007*, pages 198–204. Springer-Verlag, 2007.

[5] Soeren Kammel, Benjamin Pitzer, Stefan Vacek, Joachim Schroeder, Christian Frese, Moritz Werling, and Matthias Goebl. DARPA Urban Challenge: Team AnnieWAY - Technical System Description, 2007.

[6] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The Opteron processor for multiprocessor servers. In *IEEE Micro*, volume 23, pages 66–76, 2003.

[7] Stephan Neumaier, Philipp Harms, and Georg Färber. Videobasierte Umfelderfassung zur Fahrerassistenz. In *4. Workshop Fahrerassistenzsysteme*, Löwenstein, October 2006.

[8] Florian Rattei, Matthias Goebl, and Georg Färber. Beitrag zur Robustheitssteigerung videobasierter Fahrerassistenzsysteme durch frühe Rückkopplungen zur Sensorebene. In *Bildverarbeitung in der Mess- und Automatisierungstechnik.* VDI-Berichte, VDI Verlag, Düsseldorf, 2007.

[9] Sonderforschungsbereich Transregio 28. http://www.kognimobil.org.

[10] Christoph Stiller, Georg Färber, and Sören Kammel. Cooperative Cognitive Automobiles. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 215–220, 2007.

[11] Jürgen Stohr. *Auswirkungen der Peripherieanbindung auf das Realzeitverhalten PC-basierter Multiprozessorsysteme.* PhD thesis, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, March 2006.

[12] Thao Dang, C. Hoffmann, and C. Stiller. Self-calibration for Active Automotive Stereo Vision. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 364–369, 2006.