Anbindung des Robot Operating Systems an Speicher-programmierbare Steuerungen

Dipl.-Inform. Felix Meßmer, Fraunhofer Institut für Produktionstechnik und Automatisierung Stuttgart, Abteilung Roboter- und Assistenzsysteme

Connecting the Robot Operating System and programmable logic controllers

Current robotic research offers plenty of innovative technologies and approaches – e.g. for navigation, manipulation or sensor data processing – that allow the automation industry to develop new applications, to master complex challenges and to open new markets.
This tutorial shows a possible approach to connect the two domains, thus enabling communication between programmable logic controllers and the linux-based Robot Operating System.

PLC, ROS, communication, robotics

1. Einleitung und Motivation

Speicherprogrammierbare Steuerungen sind nach wie vor Standard in der Automatisierungstechnik und im industriellen Umfeld weit verbreitet. Sie erfüllen unter anderem Aufgaben bei der Ablaufsteuerung fertigungs- oder verfahrenstechnischer Prozesse, bei der Auswertung von Sensordaten beispielsweise in der Qualitäts-sicherung oder auch bei der Steuerung von Fahrerlosen Transportsystemen (FTS) und Industrierobotern.
Neue Anwendungsgebiete und steigende Anforderungen in der Industrie stellen jedoch heutige Automatisierungssysteme vor immer anspruchsvollere Aufgaben und komplexere Herausforderungen. So geht beispielsweise der Trend weg von Industrierobotern, welche ihre fest einprogrammierten Routinen in eigens dafür vorgesehenen Roboterzellen verrichten. Vielmehr kommt es im Zuge immer variantenreicherer und flexiblerer Produktionsabläufe und -umgebungen zu verstärkter Kooperation zwischen Mensch und Maschine in gemeinsam genutzten Bereichen. Auch mobile Manipulation und flexiblere Programmierungsmethoden gewinnen für die Industrie immer mehr an Bedeutung.
In eben diesen Bereichen verfügt die Roboterforschung – insbesondere der Bereich der Servicerobotik – bereits über eine Vielzahl fortschrittlicher und zukunftsweisender Technologien und Ansätze. Heutige Serviceroboter und Forschungsplattformen – wie etwa Care-O-bot 3® [1] – kombinieren bereits seit geraumer Zeit komplexe Algorithmen

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

aus den Bereichen Navigation, Manipulation und Bildverarbeitung, um komplexe und anspruchsvolle Aufgaben zu erfüllen.
So nutzt etwa Care-O-bot 3® Module aus dem Bereich Navigation, um sich ohne künstliche Hilfsmittel, die in der Umgebung angebracht werden müssten, nur mithilfe von Odometriedaten und Informationen aus Laserscannern in unbekannten, dynamischen Umgebungen fortzubewegen. Durch intelligente Algorithmen aus dem Bereich der Bildverarbeitung gelingt es ihm, verschiedene Objekte in seiner Umgebung zu detektieren, wiederzuerkennen und zu lokalisieren. Diese Objekte können dann mit einem integrierten Manipulator gegriffen und transportiert werden, wobei die Bewegungen des Manipulators unter Berücksichtigung verschiedener Rahmenbedingung wie etwa der aktuell vorherrschenden Hindernissituation im Voraus berechnet und anschließend ausgeführt werden.
Diese Module haben sich im Bereich der Servicerobotik durchaus bewährt. Die Portierung, Integration und Anwendung solcher Technologien und Ansätze in der Automatisierungstechnik wird bisher allerdings durch die in den jeweiligen Bereichen zum Einsatz kommende unterschiedliche Hardware, Software und Kommunikationsweise erschwert. So kommen etwa in der Industrie zumeist windows-basierte SPSen oder eingebettete Systeme zum Einsatz, während die Roboterforschung zumeist linuxbasierte PCs nutzt. Auch gibt es in der Industrie viele herstellerspezifische und standardisierte Kommunikationsprotokolle.
Eine Anwendung von Technologien und Ansätzen aus der Roboterforschung würde es jedoch der Industrie ermöglichen, gänzlich neue Anwendungsgebiete zu erschließen, komplexe Aufgaben zu bewältigen und daraus möglicherweise entscheidende Wettbewerbsvorteile zu ziehen.

Am Beispiel einer Steuerung für ein FTS zeigt dieser Vortrag, wie bisherige SPS-Systeme einfach und modular um Funktionen aus der Roboterforschung erweitert werden können, ohne dabei größere Anpassungen an bereits vorhandenen Systemen vornehmen zu müssen. In Abschnitt 2 wird zunächst das Anwendungsbeispiel dargestellt und die Problemstellung erläutert. Abschnitt 3 stellt das Robot Operating System (ROS) [2, 3] sowie die verwendete Kommunikationskomponente im Detail vor. Auf dieser Grundlage zeigt Abschnitt 4 wie die Anbindung für das konkrete Anwendungsbeispiel umgesetzt wurde. Abschließend fasst Abschnitt 5 die Ergebnisse zusammen und gibt einen Ausblick auf zukünftige Weiterentwicklungen.

2. Anwendungsbeispiel

Als ein konkretes Beispiel für die Anbindung des Robot Operating Systems (ROS) an Speicherprogrammierbare Steuerungen dient in diesem Vortrag die Steuerung eines FTS, welches um die Möglichkeit der Freien Navigation [4, 5] erweitert werden soll. Fahrerlose Transportsysteme werden inzwischen häufig im industriellen Umfeld – insbesondere in der Intralogistik – eingesetzt, um den Materialfluss und Teilenachschub zu automatisieren. Solche Systeme nutzen zur Orientierung zumeist noch künstlich angebrachte Hilfsmittel wie etwa optische Leitlinien, Induktionsdrähte, Magnetbänder oder Reflektormarken. Im Zuge immer variantenreicherer Produktionsabläufe, kürzerer Produktlebenszyklen und der sich daraus ergebenden ständig ändernden Produktionsumgebung, sind insbesondere kleine und mittelständische Unternehmen (KMU) darauf angewiesen, ihre Produktion schnell, günstig und flexibel umkonfigurieren zu können. Bei einer solchen Umstrukturierung bedeutet jedoch die Anpassung der künstlichen angebrachten Hilfsmittel an die neue Umgebung einen erheblichen Mehraufwand.

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Deshalb soll nun die bisherige Steuerung des FTS um die Möglichkeit der Freien Navigation erweitert werden. Freie Navigation bedeutet hierbei, dass das Verfahren des FTS nicht mehr gebunden ist an eben solche künstlichen Hilfsmittel. Vielmehr erfolgt die Lokalisierung und Navigation des FTS mithilfe der Odometrie des Fahrzeugs sowie Informationen aus Sensorik wie etwa Laserscannern, welche den Regelkreis mit der Umgebung schließen, um Odometriefehler (Schlupf) auszugleichen und zu korrigieren. Diese Verfahren haben sich in der Roboterforschung bewährt und werden häufig eingesetzt, um Roboterplattformen das Navigieren in dynamischen, unstrukturierten oder unbekannten Umgebungen zu ermöglichen.
Das verwendete FTS wurde bisher von einer SPS gesteuert und entlang optischer Leitlinien durch den Produktionsbereich bewegt. Ein solches Setup ist fehleranfällig in Bereichen, in denen die optischen Leitlinien starker Abnutzung und Verschmutzung unterliegen (bspw. in Bereichen, welche zusammen mit Gabelstaplern genutzt werden). In einem solchen Fall kann das FTS von seiner Route abkommen und gegebenenfalls seine Fahrt nicht fortsetzen. In diesen Bereichen soll nun die Freie Navigation verwendet werden, um die Abhängigkeit von den Leitlinien aufzulösen.
Fährt das FTS entlang einer optischen Leitlinie, wird mithilfe einer zum Boden gerichteten Kamera die Abweichung von der Leitlinie detektiert und als Steuersignal an die SPS übergeben. Diese setzt dieses Steuersignal gemäß dem implementierten Steuerungsalgorithmus in entsprechende (korrigierende) Motorbefehle um. Um weiterhin beide Möglichkeiten (also Fahren entlang optischer Leitlinie sowie Freie Navigation) nutzen zu können und nur geringfügig Änderungen am bisherigen System durchführen zu müssen, soll das leitlinien-gestützte Steuerungssignal von der Freien Navigation emuliert werden, das heißt: in Bereichen, in denen nun frei navigiert werden soll, wird das Steuersignal aus der Abweichung von einer „virtuellen“ optischen Leitlinie aus der Lokalisierungskomponenten der Freien Navigation berechnet und an die Steuerung geschickt. Diese kann nun mit demselben Steuerungsalgorithmus das Steuersignal in Motorbefehle umsetzen.

3. Konzepte

Wie bereits erwähnt, wird für die Funktionalität der Freien Navigation das Robot Operating System (ROS) genutzt. Für die Kommunikation zwischen ROS und SPS wird ebenfalls eine Komponente aus ROS verwendet – die sogenannte rosbridge [6]. Beides wird in diesem Abschnitt eingeführt und im Detail erläutert.

3.1 Das Robot Operating System

Das Robot Operating System (ROS) wurde im Rahmen des STAIR Projekts [7] der Stanford University und dem Personal Robots Program [8] von Willow Garage entwickelt. Es handelt sich dabei nicht um ein Betriebssystem im eigentlichen Sinn. Vielmehr handelt es sich um ein Framework, welches auf das zugrunde liegende Host-Betriebssystem (zumeist eine Linux-Distribution) aufsetzt und eine einheitliche und strukturierte Kommunikationsschicht sowie Entwicklungsumgebung bereitstellt.

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Inzwischen hat sich ROS quasi als Standard in der Servicerobotik etabliert und wird von einer großen Community aus dem Bereich der Robotik weltweit genutzt, unterstützt und weiterentwickelt. ROS besteht ausschließlich aus freier Software und jeglicher Quellcode ist öffentlich zugänglich (Lizenzmodelle BSD, GPL, LGPL).
Das Hauptziel von ROS ist dabei eine einheitliche Infrastruktur zu schaffen, die es ermöglicht, Algorithmen unabhängig von der Hardware eines Robotersystems auch auf verschiedenen anderen Plattformen verwenden zu können. Dadurch soll die Kompatibilität und Interoperabilität unterschiedlichster Komponenten gewährleistet werden und so der Austausch von Technologien und Quellcode zwischen Entwicklern ermöglicht und gefördert werden.
So bietet etwa das von ROS bereitgestellte, integrierte Build-System viele Werkzeuge für ein einheitliches, effizientes Package Management, welches eine einfache Struktur vorgibt, wie Code innerhalb ROS organisiert ist. Darüber hinaus wird mit dem Build-System die Code-Generierung vereinfacht und eine automatische Dokumentation unterstützt. Neben der Abstraktion der verwendeten Hardware unterstützt ROS auch die Verwendung verschiedener Programmiersprachen (C/C++, Python, LISP, Java).
Durch die weite Verbreitung von ROS bietet das Framework inzwischen einen beachtlichen Funktionsumfang, welcher von low-level Treibern für verschiedene Sensoren und Aktoren über die Einbindung standardisierter Bibliotheken wie etwa OpenCV [9] oder OMPL [10] bis hin zu high-level Funktionen für komplexe Problemstellungen wie beispielsweise Navigation, Bewegungsplanung und Bildverarbeitung reicht. Darüber hinaus beinhaltet ROS viele Tools, die das Arbeiten mit komplexen Robotersystemen erleichtern. Insbesondere sind hier verschiedene Visualisierungstools, Debug-Werkzeuge und Simulationsumgebungen zu nennen.
Um zu verstehen auf welche Art eine SPS mit ROS kommunizieren und Daten austauschen kann, soll im Folgenden näher auf das Kommunikationskonzept, das in ROS zum Einsatz kommt, eingegangen werden. Dieses nutzt ein TCP/IP basiertes Protokoll zum Austausch von Nachrichten zwischen verschiedenen Komponenten und Prozessen – sogenannten Knoten - innerhalb eines zumeist verteilten Robotersystems (multi-process, multi-host). Ein solcher Knoten erfüllt dabei eine spezielle Aufgabe innerhalb des komplexen Systems (bspw. das Filtern von Sensordaten). Um seine Aufgabe erfüllen zu können ist dieser Knoten mit verschiedenen anderen Knoten verbunden und tauscht mit diesen Informationen und Daten über entsprechende Nachrichten aus. Beispielsweise erhält er die ungefilterten Kameradaten von einem Knoten, der als Treiber für die Kamera fungiert. Im Gegenzug übergibt der Filterknoten die aufbereiteten Daten an einen Knoten, welcher diese weiterverarbeitet (bspw. Segmentieren geometrischer Strukturen).
Für den Austausch von Informationen und Daten zwischen den Knoten gibt es unter ROS zwei Möglichkeiten. Zum einen gibt es eine asynchrone, uni-direktionale Kommunikation, bei der Knoten Daten über einen mit einem Bezeichner versehenen Kanal senden. Der Bezeichner eines solchen Kanals – das sogenannte Topic – setzt sich dabei aus einem passenden Namensraum und einem Namen zu einer global eindeutigen Kennung (globally unique identifier, GUID) zusammen. Die Kommunikation über Topics entspricht dem Publisher/Subscriber-Prinzip. Hierbei können mehrere Knoten Nachrichten über den gleichen Kanal schicken. Genauso können verschiedene Knoten Nachrichten des gleichen Topics empfangen und weiterverarbeiten. Durch dieses Publisher/Subscriber-Prinzip sind die Prozesse voneinander entkoppelt.

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Neben der Kommunikation über Topics, gibt es auch eine synchrone, bi-direktionale Kommunikation – sogenannte Services. Dabei kommunizieren jeweils nur zwei Knoten direkt miteinander. Bei der Kommunikation über Services übernimmt einer der Knoten die Rolle des Service-Servers, der eine bestimmte Funktionalität bereitstellt. Der zweite Knoten möchte diese Funktion nutzen und übernimmt die Rolle des Clients. Diese Art der Kommunikation entspricht also in etwa einem Funktionsaufruf. Der Client-Knoten initiiert die Kommunikation, indem er eine Nachricht (Request) an den Server-Knoten schickt, welche die entsprechenden Parameter enthält. Der Server-Knoten bearbeitet den Request und sendet nach Bearbeitung eine entsprechende Response-Nachricht mit dem Ergebnis zurück. Bis zum Eintreffen der Response-Nachricht ist der aufrufende Knoten blockiert.
Beide Kommunikationsarten von ROS nutzen das gleiche Format für den Aufbau gesendeter Nachrichten. Dieser Aufbau wird mithilfe einer message definition festgelegt. Diese Definition beschreibt die interne Struktur der Nachricht durch sogenannte typed fields. Dabei setzt sich ein solches typed field aus dem Namen des Feldes und dem geforderten Datentyp zusammen. Der geforderte Datentyp kann dabei aus primitiven Standardtypen – wie etwa Bool, Integer oder String – bestehen. Ebenso können solche Standardtypen aber auch zu Arrays zusammengesetzt oder zu beliebig komplexen Nachrichtenstrukturen verschachtelt werden. Diese message definitions werden dann in einfachen Textdateien abgelegt (siehe Bild 1) und zur Serialisierung und Deserialisierung sowie zur Interpretation der Nachrichten genutzt.

```
# Geschwindigkeit des Fahrzeugkoordsystems in x-Richtung
# m per sec
float32 v_x

# Geschwindigkeit des Fahrzeugkoordsystems in y-Richtung
# m per sec
float32 v_y

# Rotation des Fahrzeugkoordsystems um die z-Achse
# rad per sec
float32 w
```

Bild 1: Eine message definition für den Austausch von Odometriedaten

3.2 Die Kommunikationskomponente rosbridge

Mit dem Packet rosbridge bietet ROS eine abstrahierende Schnittstelle, die es ermöglicht, ROS-Funktionalität auch ROS-fremden Systemen zugänglich zu machen. Rosbridge wurde ursprünglich für die Bereiche Mensch-Roboter-Interaktion sowie Telepräsenz und Teleoperation entwickelt. Dabei wurden web-basierte Benutzerinterfaces genutzt, um über die rosbridge mit dem ROS-System des Roboters zu kommunizieren. Es sollte so auch Entwicklern, welche selbst nicht über ein reales Robotersystem verfügen, ermöglicht werden, ihre Technologien auf verschiedenen Robotersystemen zu testen, indem das reale Robotersystem remote von einem Web-Browser aus gesteuert wird (Teleoperation) und die Bewegungen und die Umgebung überwacht (Telepräsenz) werden können. So konnten nun Interface-Designer oder

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Anwendungs-programmierer Oberflächen und Verhaltensweisen für Roboter entwickeln ohne ein tieferes Verständnis für die komplexen Algorithmen zu haben.
Die Kommunikation zwischen dem Web-Browser (bzw. dem ROS-fremden System) und dem Robotersystem (ROS) geschieht bei der rosbridge über einfache Sockets. Es werden sowohl Web-Sockets (HTML5) als auch reine TCP-Sockets (POSIX IP) unterstützt. Standardmäßig wird dafür Port 9090 genutzt.
Die Nachrichten, welche über den Socket geschickt werden, sind reine ASCII Zeichenketten, die dem JSON-Format entsprechen. JSON (JavaScript Object Notation, [11]) ist ein von JavaScript abgeleiteter, text-basierter Standard zum Austausch von Daten. Dieser wurde als low-overhead Alternative zu XML entwickelt und wird zum Serialisieren und Versenden strukturierter Daten über Netzwerkverbindungen verwendet. Trotz der Verbindung zu JavaScript ist dieses Nachrichten-Format weitestgehend programmiersprachenunabhängig und Parser bzw. Interpreter sind für viele Sprachen verfügbar.
Einzige Voraussetzung für die Kommunikation mit ROS über rosbridge ist somit die Unterstützung einer Kommunikation über Sockets sowie die Möglichkeit zum Parsen bzw. Verarbeiten von ASCII-Zeichenketten. Dies ist bei SPSen durch entsprechende Standard-Funktionsblöcke gegeben.
ROS-fremde Prozesse erhalten so unter Verwendung der rosbridge Zugriff auf die Kommunikationsschicht des ROS-Systems (Topics, Services). Dadurch ist es Ihnen möglich angebotene Topics abzuhören (subscriben) oder selbst Nachrichten auf Topics zu schreiben (publishen). Auch der Aufruf von Services ist möglich. Somit können auch ROS-fremde Prozesse die volle Funktionalität von ROS und der unter ROS entwickelten Module nutzen.
Im Folgenden soll nun konkret dargestellt werden, wie entsprechende ROS-Nachrichten für das Versenden über die Socket-Verbindung der rosbridge aufbereitet und repräsentiert werden. Die rosbridge bildet die ROS-Nachrichten auf entsprechende JSON-Objekte ab. Diese JSON-Objekte enthalten dann die kodierte ROS-Nachricht, erweitert um zusätzliche Felder, welche für das korrekte Adressieren des Empfängers sowie für die Interpretation der codierten ROS-Nachricht wichtig sind. Das resultierende JSON-Objekt umfasst demnach die folgenden Felder:

- receiver GUID (globally unique identifier)
- msg: die in JSON abgebildete ROS-Nachricht
- type: der Typ der gesendeten ROS-Nachricht

Dabei ist das receiver Feld unbedingt erforderlich, die Felder msg und type hingegen optional (jedoch meist vorhanden). In dem Feld receiver wird der gewünschte Empfänger der Nachricht mithilfe eines global eindeutigen Bezeichners angegeben. Dies entspricht auf ROS-Seite zumeist einem Topic oder Service. Das Feld msg enthält die eigentlichen Nutzdaten. Hier wird die zu sendende ROS-Nachricht kodiert. In dem Feld type wird zusätzlich der Typ der ROS-Nachricht mit angegeben. Mithilfe der entsprechenden message definiton kann dann der Empfänger die Nachricht parsen und interpretieren.
Zusätzlich wird dem JSON-Objekt noch ein Null-Byte (“\x00“) vorangestellt bzw. ein Eins-Byte (“\xff“) angehängt, um den Anfang und das Ende einer Nachricht zu markieren. Anschließend wird diese ASCII-Zeichenkette auf den Socket geschrieben.

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Sowohl für eingehende als auch für ausgehende Nachrichten wird dieses Nachrichten-Format verwendet. Auf ROS-Seite gewährleistet die rosbridge das korrekte Weiterleiten der Nachrichten an den adressierten Empfänger-Knoten. Auf Seiten der SPS muss mithilfe der Standard-Funktionsblöcke die Nachricht geparst und gemäß dem receiver- und dem type-Feld die Nachricht aufbereitet und weiterverarbeitet werden.

4. Umsetzung

Für das in Abschnitt 2 beschriebene Anwendungsbeispiel ergeben sich somit folgende Aspekte bei der Umsetzung der Anbindung von ROS an die SPS des FTS.
Die linux-basierte ROS-Umgebung ist auf einem Standard Industrie-PC (IPC) installiert, welcher neben der SPS zusätzlich im FTS untergebracht wird. Dies unterstützt den modularen Gedanken und erlaubt es die komplexen und rechenintensiven Algorithmen für die Freie Navigation auf eine leistungsfähige Hardware auszulagern. Somit kann die SPS weiterhin ihre Steuerungsaufgaben wahrnehmen. Auch die bisherigen Sicherheitsfunktionen bleiben Teil der SPS. Die Verbindung zwischen IPC und SPS wird mit einer Standard-Ethernet-Verbindung realisiert.
Wie bereits erwähnt müssen keine größeren Änderungen an dem eigentlichen Steuerungsalgorithmus der SPS vorgenommen werden. Lediglich das Umschalten zwischen Navigation entlang einer optischen Leitlinie und Freier Navigation muss hinzugefügt werden. Das Umschalten wird in dem vorgestellten Fall durch RFID-Tags gelöst, welche den Beginn bzw. das Ende eines Abschnitts markieren, in dem das FTS mithilfe der Freien Navigation gesteuert werden soll. Für die Kommunikation mit dem IPC und der rosbridge werden auf SPS-Seite Standard-Funktionsblöcke genutzt, um die entsprechende Socket-Verbindung auf- bzw. abzubauen, sowie ASCII-Zeichenketten zu lesen und zu schreiben. Der größte Programmieraufwand auf Seiten der SPS fällt auf die Funktionsblöcke, welche entsprechende Nachrichten zusammenstellen bzw. empfangene Nachrichten parsen und weiterverarbeiten. Aufgrund der klar definierten Nachrichten-Struktur, lässt sich allerdings auch dies leicht bewältigen.
Der Informationsaustausch zwischen SPS und IPC umfasst nun folgende Nachrichten:

- Odometriedaten: SPS → IPC
- RFID-Signal: SPS → IPC zum Starten der Freien Navigation
- Steuerungssignal: IPC → SPS das emulierte Kamerasignal

Die Messdatenausgänge der Sicherheits-Laserscanner können direkt mit dem IPC verbunden werden.
Der IPC verfügt über ein Linux-Betriebssystem (Ubuntu) und hat neben den ROS-Basis-Komponenten auch alle benötigten Pakete für die Freie Navigation sowie die rosbridge installiert. Diese Pakete werden beim Starten des FTS automatisch gestartet. Der IPC übernimmt die Rolle des rosbridge-Servers und wartet auf eingehende Verbindungen des rosbridge-Client, in diesem Fall der SPS. Nach dem Hochfahren aller Komponenten initiiert nun die SPS eine Socket-Verbindung zu dem IPC über die rosbridge. Zusätzlich fordert rosbridge einen Handshake zur Initialisierung der Socket-Verbindung. Dies geschieht durch das Senden der ASCII-Zeichenkette

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

„raw\r\n\r\n“

Nach dem erfolgreichen Verbindungsaufbau, kann sich die SPS nun für Topics der ROS-Umgebung auf dem IPC registrieren (subscriben), die für die Steuerung relevant sind. Dies betrifft lediglich das emulierte Kamerasignal der Freien Navigation. Im Vorfeld wurde hierfür ein entsprechender Bezeichner definiert, unter dem das Steuerungssignal abgerufen werden kann (hier: /ipa/cpp_data) sowie der dafür verwendete Datentyp (hier: ipa_msgs/cpp_data). Durch senden der ASCII-Zeichenkette

'\x00{"receiver":"/rosbridge/subscribe","msg":["/ipa/cpp_data",-1,"ipa_msgs/cpp_data"]}\xff'

werden nun alle Nachrichten die auf das Topic “/ipa/cpp_data“ gepublished werden auch über die rosbridge an die SPS übertragen.
Im Gegenzug kann die SPS Odometriedaten (siehe Bild 1) an die Freie Navigation senden, indem sie die ASCII-Zeichenkette

'\x00{"receiver":"/odometry", "msg":{"v_x": **X**, “v_y”: **Y**, “w”: **W**}, "type":"ipa_msgs/odometry"}\xff'

auf den Socket schreibt, wobei **X** bzw **Y** den aktuellen Wert für die Geschwindigkeit in x- bzw. y-Richtung beschreibt und **W** die aktuelle Rotationsgeschwindigkeit um die z-Achse. Diese Nachricht kann dann von der Freien Navigation unter dem Topic “/odometry“ abgehört werden.

5. Zusammenfassung und Ausblick

Das hier vorgestellte konkrete Anwendungsbeispiel zeigt, wie die Anbindung der beiden Bereiche Automatisierungstechnik und Roboterforschung sowie der in den jeweiligen Bereichen genutzten Infrastruktur (Windows, SPS bzw. Linux, ROS) gelingen kann.
Die Kommunikation und der Austausch von Daten ist über die ROS-Komponente rosbridge einfach zu realisieren. Dadurch lassen sich bisherige Systeme in der Automatisierungstechnik im industriellen Umfeld leicht um fortschrittliche Technologien aus dem Bereich der Roboterforschung erweitern. Dies erlaubt die Erschließung neuer Anwendungsgebiete und die Bearbeitung komplexer Aufgabenstellungen.
An dieser Stelle ist anzumerken, dass diese Anbindung nicht nur auf die Erweiterung einer SPS um die Freie Navigation für das Steuern eines FTS beschränkt ist. Vielmehr ist diese Anbindung vielseitig anwendbar und es wurde gezeigt, dass die beiden Bereiche Automatisierungstechnik und Roboterforschung generell miteinander kombiniert werden können und sich so gegenseitig ergänzen.
So sind beispielsweise auch die Erfahrungen aus der Roboterforschung im Bereich der Mobilen Manipulation für die Industrie interessant, wenn es etwa darum geht, Material nicht nur zu transportieren, sondern möglicherweise auch direkt manipulative Schritte durchzuführen oder Maschinen zu bestücken. Auch hierfür gibt es unter ROS verschiedene Module für das Planen und Ausführen kollisionsfreier Bewegungen von Robotern und Manipulatoren in dynamischen Umgebungen. Dies führt ebenfalls zu einer höheren Flexibilität in der Produktion, da so möglicherweise aufwändiges Einteachen von Bewegungsabläufen reduziert werden kann. Neben der Mobilen Manipulation bietet ROS auch im Bereich der Sensordatenverarbeitung und Bildverarbeitung fortschrittliche

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.

Technologien und Ansätze, welche beispielsweise in der Qualitätssicherung zum Einsatz kommen könnten.

Darüber hinaus sollen zukünftig weitere Kommunikationsmöglichkeiten geschaffen werden, die eine einfache und nahtlose Integration von Technologien aus der Roboterforschung oder möglicherweise auch die Nutzung von Serviceroboter-Plattformen im industriellen Umfeld ermöglichen. Dabei sollen weitestgehend Verfahren zum Einsatz kommen, die in der Industrie bereits eingesetzt werden. Beispielsweise existieren erste Ansätze für eine Anbindung von ROS an OPC/OPC-UA [12].

6. Referenzen

[1] Reiser, Ulrich; Connette, Christian P.; Fischer, Jan; Kubacki, Jens; Bubeck, Alexander; Weisshardt, Florian; Jacobs, Theo; Parlitz, Christopher; Hägele, Martin; Verl, Alexander: "Care-O-bot® 3 - Creating a product vision for service robot applications by integrating design and technology", In: Institute of Electrical and Electronics Engineers: IROS 2009 : The 2009 IEEE/RSJ International Conference on Robots and Intelligent Systems, St. Louis, MO, USA, Oct. 11-15, 2009. Piscataway, NJ : IEEE, 2009, S. 1992-1998.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng: "ROS: an open-source Robot Operating System", In: International Conference on Robotics and Automation, ser. Open-Source Software workshop, 2009.

[3] The ROS website. [Online]. Available: www.ros.org

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, ser. Intelligent robotics and autonomous agents. The MIT Press, Aug. 2005.

[5] SRI International. The Karto website. [Online]. Available: www.kartorobotics.com

[6] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. Jenkins, "Rosbridge: ROS for non-ROS users," in Proceedings of the 15th International Symposium on Robotics Research, 2011.

[7] M. Quigley, E. Berger, and A. Y. Ng, "STAIR: Hardware and Software Architecture," in AAAI 2007 Robotics Workshop, Vancouver, B.C, August, 2007.

[8] K. Wyobek, E. Berger, H. V. der Loos, and K. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," in Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), 2008.

[9] G. Bradski and A. Kaehler, "Learning OpenCV", Sep. 2008.

[10] I. A. Sucan, M. Moll, and L. Kavraki. (2010). The Open Motion Planning Library (OMPL), [Online]. Available: http://ompl.kavrakilab.org

[11] Crockford, D. JSON (JavaScript Object Notation). [Online], Available: http://www.json.org/.

[12] W. Mahnke, S.-H. Leitner, M. Damm: OPC Unified Architecture, ISBN: 978-3-540-68898-3, New York: Springer, 2009.

erschienen in: sps ipc drives 2012: 27. - 29. November 2012, Nürnberg; Tagungsband. Berlin; Offenbach : VDE Verlag, 2012, S. 15-23.