Vorversion des Beitrags, der verröffentlicht wird bei den 19. Fachgesprächen Autonome Mobile Systeme AMS 2005, 8. und 9. Dezember 2005, Stuttgart

Architektur und Komponenten für ein heterogenes Team kooperierender, autonomer humanoider Roboter

Jutta Kiener, Sebastian Petters, Dirk Thomas, Martin Friedmann und Oskar von Stryk

Fachgebiet Simulation und Systemoptimierung, Fachbereich Informatik,
Technische Universität Darmstadt, D-64289 Darmstadt,
{kiener,petters,dthomas,friedmann,stryk}@sim.tu-darmstadt.de,
WWW homepage: http://www.sim.informatik.tu-darmstadt.de

Zusammenfassung. Für ein kooperierendes Team autonomer, humanoider Roboter, das derzeit aus insgesamt vier unterschiedlichen, ca. 37 - 68 cm großen Robotertypen besteht, werden eine plattformübergreifende, modulare Softwarearchitektur sowie plattformübergreifende und individuelle Module zur Sensordatenverarbeitung, Planung und Bewegungssteuerung entwickelt. Das entwickelte funktionale Framework ermöglicht die Kommunikation der Softwaremodule, d.h. Algorithmen für die unterschiedlichen Aufgaben innerhalb der Architektur untereinander, sowie die Kommunikation per WLAN zwischen verschiedenen Rechnern und Robotern. Als Anwendungsszenario für die Teamkooperation in einer dynamischen und strukturierten Umgebung wird Roboterfußball untersucht. Die entwickelten Methoden wurden im Juli 2005 von den Darmstadt Dribblers beim RoboCup in Osaka bei der Premiere von Teamspielen in der Humanoid Robot League eingesetzt. Daneben werden Kooperationsszenarien von heterogenen Robotersystemen bestehend aus vierbeinigen und humanoiden Robotern untersucht.

1 Einleitung

Motivation und Zielsetzung. Kooperierende, autonome Mehrrobotersysteme werden derzeit für unterschiedliche Anwendungen, z.B. den kooperativen Transport einer Last durch mehrere fahrende oder fliegende Roboter, die Überwachung und Aufklärung eines Katastrophengebiets oder beim Roboterfußball in einer besonders dynamischen Umgebung untersucht.

Zur effektiven, flexiblen und robusten Entwicklung eines heterogenen Mehrroboterteams werden neben *Modulen* zur Lösung der grundlegenden Fragen der Sensordatenverarbeitung, Planung und Bewegungssteuerung *Softwarearchitekturen* sowie ein effizientes *Framework* benötigt. Anforderungen sind dabei *Modularität* für die Integration plattformunterschiedlicher oder konkurrierender Module zur Sensor-, insbesondere Kameradatenverarbeitung, Lokalisierung, Bewegungssteuerung, Verhaltenssteuerung sowie *Flexibilität* zur Anpassung an wechselnde Hardware wie unterschiedliche Prozessoren, Kamerasysteme oder Bewegungsapparate. Ebenso unterstützt werden muss die *Kommunikation* zwischen einzelnen Modulen und verschiedenen Robotern und Rechnern in unterschiedlichen Phasen von Entwicklung und der Betrieb der Mehrrobotersysteme unter Berücksichtigung der besonderen Anforderungen laufender, zwei- und

vierbeiniger Roboter, die u.a. in der besonderen Schwierigkeit und Vielfalt der Bewegungsmöglichkeiten sowie der Auge-Bein Koordination liegen.

Stand der Forschung. In den letzten Jahren wurden wesentliche Fortschritte bei humanoiden Robotern erzielt. Dennoch enthält die robuste und schnelle Fortbewegung beim zweibeinigen Laufen sowie die autonome Navigation mit Auge-Bein-Koordination noch viele ungelöste Fragen. Die Herausforderungen beim Fußballspielen mit autonomen humanoiden Robotern liegen unter anderem in der Ausführung möglichst schneller, zielorientierter und situationsabhängiger Bewegungen unter Berücksichtigung von Bewegungsstabilität und Echtzeitanforderungen.

Bisherige Ansätze für Roboterarchitekturen für zielorientiert kooperierende Mehrrobotersysteme erfüllen die vorstehend ausgeführten Anforderungen für die hier betrachteten Humanoid-Roboter nur bedingt. Beispielsweise ist das für rollende Mehrrobotersysteme entwickelte auf CORBA basierende Miro [1] für leistungsfähige Mehrprozessorsysteme optimiert. Auf stromsparenden, leistungsschwächeren Ein- oder Mehrprozessorsystemen wie auf den hier betrachteten Humanoid-Robotern bringt die Verwendung von CORBA jedoch einen Effizienzverlust mit sich, da nur wenige Vorteile dieser Middleware ausgenutzt werden können.

Saphira [2,3] ist die Softwareumgebung für die verbreiteten, radgetriebenen Pioneer-Roboter und ermöglicht eine direkte Anwendungsprogrammierung. Diese ist nicht transparent und im Quelltext nicht zugänglich, so dass die Integration nicht vom Hersteller vorgesehener Sensor- oder Aktuator-Hardware nicht auf einfachem Wege möglich ist. Auch ermöglicht es keine genaue Zeitstempelung von Sensordaten. CLARAty [4] ist als ein Framework für generische und wiederverwendbare Roboterkomponenten entwickelt worden, das zwar auf unterschiedliche Plattformen angepaßt werden kann, aber kooperative Mehrrobotersysteme nicht direkt unterstützt. Darüber hinaus sind sämtliche dieser Architekturen für radgetriebene und nicht für die besonderen Anforderungen bei laufenden, vier- oder zweibeinigen Robotern ausgerichtet.

Ansätze für funktionale Architekturen bei autonomen, humanoiden Robotern wurden erst in jüngster Zeit vorgestellt, ohne jedoch die Lösung komplexer Aufgaben durch Teamkooperation zu berücksichtigen. Beispielsweise die für den japanischen HRP2 Humanoid-Roboter jüngst vorgestellte Systemsoftware [5] ist nicht für den kompletten Austausch einzelner Module, eine Portierung auf andere Humanoid-Roboter-Plattformen oder zur Roboterkooperation vorgesehen. Eine modulare, verteilte Steuerungsarchitektur für einen humanoiden Oberkörper auf Rädern zur Anwendung bei der Mensch-Roboter Kooperation wird in [6] beschrieben, wobei Aspekte der zielorientierten Mehrroboterkooperation und der Bewegungen auf zwei Beinen keine Rolle spielen.

Die in der Four-Legged Robot League des RoboCup im GermanTeam (HU Berlin, U Bremen, TU Darmstadt, U Dortmund) entwickelte Architektur [7], die in aktueller Version [8] im Oktober 2004 veröffentlicht und seither von mindestens einem Dutzend Teams weltweit eingesetzt wird, berücksichtigt einen Teil der genannten Anforderungen für ein homogenes Team vierbeiniger Roboter. Diese enthält jedoch fix verbundene Schnittstellen zwischen den einzelnen Modulen, so dass die Struktur nicht flexibel geändert werden kann, beispielsweise für das Testen oder den Austausch neuer Module mit modifizierten Ein- und Ausgaben. Zudem ist die Anwendung nicht auf anderen Plattformen als dem vierbeinigen Roboter getestet, die graphischen Oberflächen zum

Debuggen der Anwendung sind aufgrund von Windows-spezifischen Bibliotheken nur auf diesem Betriebssystem lauffähig.

Das hier vorgestellte, neu entwickelte plattformunabhängige Framework *RoboFrame* bietet effiziente Mechanismen zum Datenaustausch auf Ein- oder Mehrprozessorsystemen, wie sie auf Humanoid-Robotern eingesetzt werden, sowie einfach zu benutzende Schnittstellen für den Austausch zwischen den Modulen. Zum Debuggen wird eine erweiterbare und plattformunabhängige graphische Benutzeroberfläche verwendet.

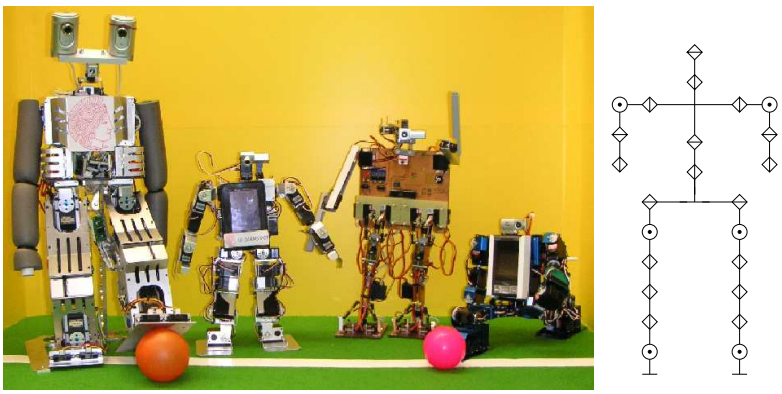

Abb. 1. Die vier untersuchten, ca. 37-68 cm großen Humanoid-Roboter-Prototypen (links) und kinematische Struktur von Mr. DD (rechts).

2 Humanoide Roboterplattformen

Derzeit wird an vier unterschiedlichen Prototypen von Humanoid-Robotern geforscht. Der Bewegungsapparat des 68 cm große Roboters Mr. DD (24 DoF, ganz links in Abb. 1) ist ein von der japanischen Firma iXs hergestelltes Unikat. Die Bewegungsapparate der beiden kleineren, ca. 37 cm großen Roboter Mr. DD junior 1 (zweiter von links) und 2 (ganz rechts) mit je 21 DoF sind als Bausätze KHR1 und YDH der japanischen Firma Kondo verfügbar, die jeweils um ein Drehgelenk in den Beinen erweitert wurden. Der vierte Prototyp (dritter von links in Abb. 1) wurde in Zusammenarbeit mit der deutschen Firma PM-Solutions entwickelt. Alle Roboter sind aus Servomotoren aufgebaut und besitzen jeweils sechs nicht-redundante Bewegungsfreiheitsgrade pro Bein, jedoch in unterschiedlichen kinematischen Anordnungen. Die Bewegungsapparate wurden jeweils um ein Stereo- bzw. Monokamerasystem, um WLAN sowie Berechnungskapazitäten durch einen Pocket PC erweitert, wobei Mr. DD über ein Controller-Board mit 64 Bit MIPS Prozessor verfügt. Darüber hinaus wurden teilweise weitere Sensoren wie Gyroskop im Oberkörper, Beschleunigungssensor und Bodenkontaktsensoren in den Füßen integriert. Weitere Informationen zum Aufbau der Roboter stehen auf der Team-Homepage www.dribblers.de zur Verfügung.

3 Softwarearchitektur

Die der Architektur zugrunde liegende objektorientierte Software ist modular aufgebaut. Sie gliedert sich in die einzelnen Module für die verschiedenen Aufgaben innerhalb der Bildverarbeitung, Objekterkennung, Lokalisierung, Bewegungs- und Verhaltenssteuerung und ein Framework, das die Kommunikation zwischen den Modulen abwickelt. Da das Framework nicht nur bei unterschiedlichen Humanoid-Robotern, sondern mittelfristig auch bei heterogenen Teams von Humanoid-Robotern mit vierbeinigen oder radgetriebenen Robotern eingesetzt werden soll, muss es plattformunabhängig, d.h. auch unabhängig von spezifischer Hardware oder Betriebssystemen, gehalten werden. Unterstützte Betriebssysteme sind Linux, Unix, Windows und Windows CE. Auch für spätere Weiterentwicklungen muss die Erweiterbarkeit des Frameworks gegeben sein.

Das Framework stellt die Interprozess-Kommunikation zwischen den Modulen, die in Threads zusammengefaßt oder auf mehrere Threads verteilt werden können, her. Die Threads können dabei asynchron auf einem Rechner oder verteilt auf mehreren Rechnern ausgeführt werden. Für den zeitoptimierten Datenaustausch sind zwei verschiedene Kommunikationsmechanismen installiert: SharedMemory für große auf einem Rechner vorgehaltene Daten sowie Nachrichten in Ringpuffern für den allgemeinen Datenaustausch auf einem Rechner sowie zwischen verschiedenen Rechnern, die mit einer exakten Zeitstempelung versehen werden. Zur Kommunikation kann je nach Anwendungsfall TCP als zuverlässiges verbindungsorientiertes oder UDP als minimales, verbindungsloses Protokoll verwendet werden. Für eine Roboter-Roboter-Kommunikation wird in der Regel UDP gewählt, um auch für die Inter-Robot-Kommunikation die Echtzeitanforderungen erfüllen zu können. Die zentralen Betriebssystemfunktionen wie Synchronisation und Netzwerkzugriff sind gekapselt und durch eine Abstraktionsschicht plattformübergreifend verfügbar.

Die Architektur selbst wird über die Applikation aufgebaut. Dort werden benutzerdefiniert die in einem oder mehreren Threads verwendeten Module angegeben, ggf. mit unterschiedlichen Taktraten für jeden Thread. Da der Datenaustausch pufferbasiert oder mittels SharedMemory erfolgt, müssen keine festen Schnittstellen zum Datenaustausch in den einzelnen Modulen angegeben werden. Der zeitliche Ablauf der Module regelt sich über das Vorhandensein der von den Modulen angeforderten Daten in den jeweiligen Puffern. Über einen Log-Mechanismus können Modul-Informationen zum textbasierten Debugging auf die Konsole oder in Log-Dateien ausgegeben werden.

Eine graphische Benutzeroberfläche (GUI) innerhalb des Frameworks ermöglicht über eine zuverlässiges TCP-Verbindung zusätzlich die schnelle Auswertung der programmierten Algorithmen sowie deren graphisches Debugging und das Arbeiten mit aufgezeichnete Nachrichten. Für Module können eigene Visualisierungsroutinen implementiert und in die GUI eingebunden werden. Desweiteren sind Standardmechanismen wie zum Beispiel das Aufbauen einer Netzwerkverbindung in der GUI steuerbar.

4 Module

Die einzelnen Software-Module für Sensorik, Planung und Aktorik können nicht nur in einem hierachischen, deliberativen Ablauf aufgerufen werden, sondern es sind belie-

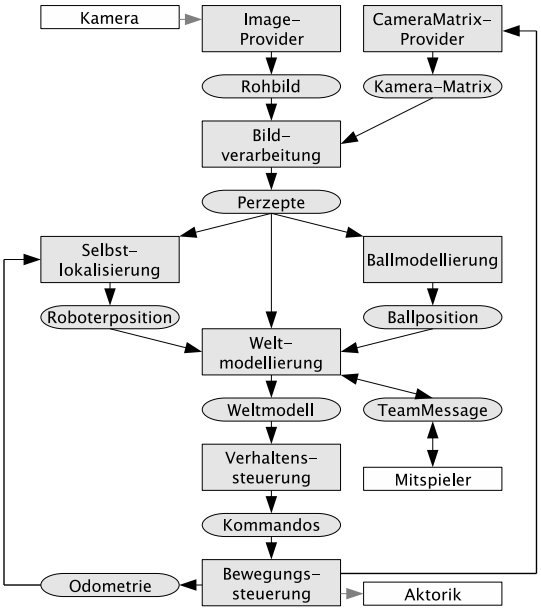

Abb. 2. Übersicht über die verwendeten Module (Rechtecke) und ausgetauschte Daten (Ellipsen) der realisierten Architektur für die Anwendung Roboterfußball.

bige Steuerungsstrukturen implementierbar. Sensorwerte und deren weiterverarbeitete Werte stehen in den oben beschriebenen Ringpuffern oder im SharedMemory bereit und werden erst auf Anfrage von anderen Software-Modulen zur Verfügung gestellt. Die Daten sind dabei von Frameworkseite automatisch mit dem zur Generierungszeit erstellten Zeitstempel versehen. Durch das dedizierte Anfragen der Daten durch die Module wird unnötiges Generieren und Umkopieren der Daten vermieden. Es stehen immer die aktuellsten Werte in einer zuvor definierten Anzahl zur Verfügung. Das Weiterleiten der Daten im Puffer zwischen einzelnen Modulen, Threads oder Rechnern wird komplett vom Framework übernommen. Somit beschränkt sich die Datenabfrage und -ausgabe innerhalb der Module auf jeweils einen Funktionsaufruf, wodurch eine hohe Benutzerfreundlichkeit und Lesbarkeit des Code garantiert wird.

Die einzelnen Software-Module gliedern sich im Wesentlichen in die Bereiche Bildverarbeitung, Verhalten, Sensordatenaufnahme und Generierung einer Laufbewegung (vgl. Abb. 2). Die nachfolgend beschriebenen Module kommen auf den Humanoid-Robotern zum Einsatz.

Sensor-Aktuator-Schnittstelle. Die externen (Kamera, Abstandssensoren) und internen (Winkelencoder, Gyroskop, Beschleunigungssensor) Sensordaten werden über eine USB-, SDIO- oder serielle Schnittstelle eingelesen und mit einem eindeutigen Zeitstempel versehen. Die berechneten Gelenkwinkel werden über eine USB- oder serielle Schnittstelle an die Motoren übergeben.

Bewegungen. Die Art der Laufbewegung (nach vorne, zur Seite, rückwärts) wird zur Laufzeit durch die Verhaltenssteuerung festgelegt und derzeit online mit Hilfe einer roboterspezifischen inversen Beinkinematik als statisch stabile Bewegung bestimmt. Neben den Laufbewegungen sind Sonderbewegungen wie auf den Boden legen, Aufstehen aus der Bauch- oder Rückenlage sowie Schussbewegungen und Übergänge zwischen Elementarbewegungen auswählbar.

Bildverarbeitung. In der Humanoid League des RoboCup gibt es ein definiertes Spielfeld. Daher können aufgrund der Größe bekannter Objekte wie der Tore und des Balls auch mit einem Monokamerasystem Abstände berechnet werden. Aufbauend auf einer Farbsegmentierung werden derzeit die auftretenden Objekte wie Tor, Ball, Feldlinien, Mitspieler und Gegner klassifiziert und lokalisiert.

Selbstlokalisierung. Im Unterschied zur Four-Legged Robot League stehen keine Landmarken als zusätzliche Orientierungshilfen zur Verfügung. Die Selbstlokalisierung muß daher allein auf Basis der erkannten Objekte (Tore, Linien) und ggf. auf über WLAN kommunizierten Informationen der Mitspieler beruhen. Zu diesem Zweck wird derzeit ein kombiniertes Markov-Partikel-Verfahren entwickelt.

Weltmodellierung. Die Umwelt des Roboters wird anhand der erkannten Objekte (Tore, Linien, Ball, Roboter) sowie der über WLAN kommunizierten Informationen der Mitspieler modelliert. Dabei wird unterschieden, ob eine bekannte Umgebung (wie beim RoboCup) anhand der lokalisierten Objekte bestimmt wird oder ob eine unbekannte Umwelt in einer potentiellen anderen Anwendung exploriert wird.

Verhalten. Zur Verhaltenssteuerung beim Roboterfußball werden hierarchische Zustandsautomaten zur Steuerung von Kopf-, Bein-, Armbewegungen sowie von komplexeren Verhaltensweisen von Torwart und Feldspielern entwickelt, wie sie bereits im GermanTeam [8,9] in der Four-Legged Robot League erfolgreich eingesetzt wurden.

5 Ergebnisse und Ausblick

Die entwickelten Humanoid-Roboter, Module und Architektur werden u.a. im Rahmen der Wettbewerbe des RoboCup evaluiert.

RoboCup 2004. Mr. DD war der erste Humanoid-Roboter einer deutschen Forschungseinrichtung, der in der Humanoid League in der Disziplin Penalty Kick teilgenommen hat.

RoboCup German Open 2005. Mr. DD und Mr. DD jr. 1 führten gemeinsam mit dem Freiburger Team Penalty Kick Demonstrationen durch. Das Torwartverhalten ermöglichte das Fallen zur Seite, beim Stürmerverhalten konnte der Ball lokalisiert, Schritte und Positionierung zum Ball geplant und mit einem Schuß aufs Tor abgeschlossen werden.

RoboCup 2005 Osaka. An den erstmalig durchgeführten 2-2 Spielen sowie am Elfmeterschießen konnte erfolgreich teilgenommen werden. Darüber hinaus fand eine Demonstration eines gemischten Teams autonomer vierbeiniger Roboter (Aibos) und humanoider Roboter statt.

Kooperatives Verhalten. Durch kooperatives Verhalten von homogenen oder heterogenen Robotern kann ein ausfallsichereres Gesamtsystem erzeugt werden, da Sensorausfälle auf einem Robotersysteme durch Daten der anderen Roboter kompensiert werden können. Die entsprechende Verhaltenssteuerungsarchitektur ist für die Kommunikation der Roboter untereinander so ausgelegt, dass die von den anderen Robotern verschickten Daten in die eigenen Module eingelesen werden können, bei einem Übertragungsausfall der anderen Roboter die Module innerhalb der Architektur aber nicht auf die Fremddaten für das Erzeugen eines Verhaltens angewiesen ist. Auf Modulebene kann der Zugriff auf die Fremddaten benutzerfreundlich analog wie auf die robotereigenen Pufferdaten erfolgen. Ein kooperatives Verhalten wird für zwei verschiedene Konstellationen demonstriert, zwischen zwei Humanoid-Robotern und einem vierbeinigen und einem Humanoid-Roboter.

In einem vereinfachten RoboCup-Szenario befinden sich zwei Humanoid-Roboter mit einem Ball auf einem Spielfeld. Position und Orientierung der Roboter ist vorgegeben. Die Roboter tauschen untereinander aus, an welcher Stelle der Ball von ihnen bzgl. eines jeweils festen Roboterkoordinatensystems gesehen worden ist. Die von jedem Roboter erkannte Position wird mit der vom anderen Roboter verglichen, aus beiden zusammen wird die endgültige Ballposition berechnet. Fällt ein Kamerasystem bei einem der beiden Humanoid-Roboter aus, so kann die Ballposition trotzdem noch aus den Daten des anderen Robotersystems rekonstruiert werden, wodurch einen robusteres Gesamtsystem erreicht wird. Weiter soll die Position der Roboter beliebig in einer definierten Umgebung sein. Mittels Landmarken können die Roboter ihre eigene Position und Orientierung innerhalb der Umgebung bestimmen. Ausgetauscht werden dann die Daten des robotereigenen Weltmodells zur Verbesserung der jeweiligen Weltmodelle.

Als zweites Szenario wird die Kooperation zweier heterogener Robotersysteme, eines vierbeinigen AIBO ERS-7 Roboters mit 64 Bit MIPS-Prozessor unter Aperios/-OPEN-R, auf dem eine entsprechende Verhaltenssteuerung auf Basis des entwickelten Frameworks implementiert wurde, und des Humanoid-Roboters Mr. DD jr. 2, gezeigt. Der vierbeinige Roboter erkennt mit seinem Kamerasystem einen aus Sicht des Humanoid-Roboters verdeckten Ball und übermittelt diesem die Koordinaten des Balls. Daraufhin gibt der Humanoid-Roboter durch Heben des jeweiligen Arms die Lage des vom vierbeinigen Roboters erkannten Balls (links, rechts) zu erkennen. Liegt kein Ball im Sichtfeld des vierbeinigen Roboters, reagiert der Humanoid-Roboter nicht, vgl. Abb. 3.

Ausblick. Während die genannten Szenarien für sich allein auch mit erheblich weniger Architekturaufwand direkt realisierbar sind, dienen diese als erste, erfolgreiche Tests für die beschriebenen Entwicklungen und damit als Ausgangsbasis für die Untersuchung komplexerer Mehrroboterkooperationen. Bis zum RoboCup 2006 soll aufbauend auf einer auf dem Framework aufsetzenden Steuerungsarchitektur sowie einer Weiterentwicklung der verwendeten Module und Hardware ein erfolgreiches Team humanoider Roboter für 3-3 Spiele entwickelt werden. Für Anwendungen heterogener Roboterteams außerhalb des Roboterfußballs werden außer humanoiden und vierbeinigen Robotern auch radgetriebene Systeme wie ein Pioneer 2DX integriert.

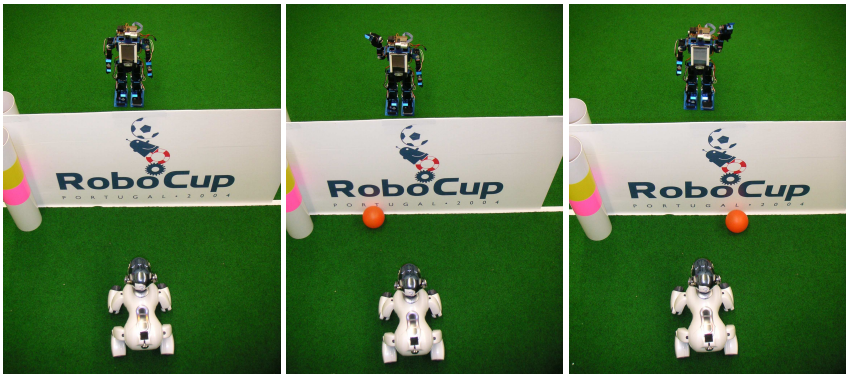

Abb. 3. Der hinter einem Sichtschutz stehende Humanoid-Roboter reagiert auf die vom vierbeinigen Roboter gesendeten Nachrichten bzgl. der erkannten Ballposition. Links: Kein Ball erkennbar, keine Reaktion. Mitte/Rechts: Ball links bzw. rechts, Humanoid-Roboter winkt mit rechtem bzw. linkem Arm.

Literaturverzeichnis

1. H. Utz, S. Sablatnög, S. Enderle, G.K. Kraetzschmar: Miro – middleware for mobile robot applications. *IEEE Trans. on Robotics and Automation*, Vol. 18, 2002, 493-497.
2. K. Konolidge: Saphira robot control architecture version 8.1.0. *SRI International*, April, 2002.
3. A. Orebäck, H.I. Christensen: Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14, 2003, 33-49.
4. I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, W.S. Kim: CLARAty: An architecture for reusable robotic software. *SPIE Aerosense Conference*, Orlando, Florida, April 2003.
5. K. Okada, T. Ogura, A. Haneda, D. Kousaka, H. Nakai, M. Inaba, H. Inoue: Integrated system software for HRP2 humanoid. *Proc. 2004 IEEE ICRA*, New Orleans, April, 2004, 3207-3212.
6. D.N. Ly, K. Regenstein, T. Asfour, R. Dillmann: A modular and distributed embedded architecture for humanoid robots. *Proc. IEEE/RSJ IROS*, Japan, Sep. 28 - Oct. 2, 2004.
7. T. Röfer: An architecture for a national RoboCup team. In G.A. Kaminka, P.U. Lima, R. Rojas (eds.): *RoboCup 2002: Robot Soccer World Cup VI*, Lecture Notes in Artificial Intelligence, Springer, 2003, 417-425.
8. GermanTeam. http://www.germanteam.org
9. M. Lötzsch, J. Bach, H.-D. Burkhard, M. Jüngel: Designing agent behavior with the extensible agent behavior specification language XABSL. In *7th International RoboCup Symposium 2003 (Robot World Cup Soccer Games and Conferences)*, Padovy, Italy, Lecture Notes in Artificial Intelligence, Springer, 2004, to appear.