Robocloud: prototypische Implementierung einer netzbasierten Robotikanwendung

Maximilian Strehse

Brandenburg University of Applied Sciences
strehse@fh-brandenburg.de

Zusammenfassung

Dieses Paper beschreibt die prototypische Realisierung einer netzbasierten Robotikanwendung am Beispiel von Robocloud. Zu Beginn werden ausgewählte bestehende Standards aus dem Bereich der netzbasierten Robotik kurz erläutert. Danach erfolgt eine Übersicht wie Robocloud umgesetzt wurde. Nach der Überprüfung der Anwendung auf ihre Eignung als alternative Implementierung für die Steuerung eines autonomen Segelbootes folgt ein kurzer Ausblick, wie Robocloud später erweitert und für andere Anwendungen im Bereich der Robotik angepasst werden kann. Das Projekt Robocloud entstand im Kontext der Lehrveranstaltung autonomes Segelboot des KI Labors der FH Brandenburg.

I. Einleitung

Unter Cloud-Computing wird die Bereitstellung von Hardware und Infrastruktur (IaaS), Plattformen (PaaS) und Anwendungen (SaaS) über das Internet verstanden [5, 11], welches in Netzwerk Diagrammen allgemein als Wolke (engl. Cloud) gekennzeichnet wird. Im Allgemeinen zeichnen sich Clouddienste durch ihre geringe Ausfallzeit, ihre Fähigkeit zu skalieren und die Fähigkeit innerhalb kürzester Zeit hohe Rechenleistung bereitstellen zu können, aus. Dies macht sie vor allem für Bereiche interessant, in welchen nicht vorhersehbar ist, wann rechenintensive Prozesse stattfinden oder aufgrund der geringen Leistung der angebundenen Systeme solche Berechnungen nicht in absehbarer Zeit durchgeführt werden können [3, 10].

In der Regel ist die Leistungsfähigkeit von autonomen Systemen durch deren Größe begrenzt [2, 6]. Dies ist vor allem im Bereich der Robotik der Fall, wo je nach Anwendungsgebiet die Größe der Systeme starren Vorgaben unterliegt. Zudem verbraucht leistungsstarke Hardware auch entsprechend viel Strom, was für einen Dauereinsatz des Systems ein Problem darstellen kann [5]. Der PR2 von Willow Garage, welcher zur Zeit als Forschungsplattform für das Projekt ROS (Robot Operating System) verwendet wird, verbraucht beispielsweise allein für die 2 verbauten Computer 500 Watt Strom, während die Aktoren gerade einmal 160 Watt verbraucht. Im Falle vom PR2 würde das Abschalten eines Computers die Akkulaufzeit nahezu verdoppeln.

Ein weiterer Punkt sind die Kosten für die Anschaffung [2] der Computer und die damit verbundene Kühlung, welche sich auf den Gesamtpreis des Roboters niederschlagen. Der große Vorteil heterogener Systeme im Vergleich zu autonomen Systemen ist, dass man rechenintensive Prozesse der Roboter dorthin verlagert, wo Strom im Überfluss vorhanden ist, Kühlung kein Problem darstellt und nahezu unbegrenzter Platz für die Hardware existiert. Zudem bietet die gemeinsame Nutzung einer Informationsdatenbank die Möglichkeit des gemeinsamen Lernens. aller am System beteiligten Clients. Durch die Verbindung aller Clients untereinander, mit der Cloud als Message Broaker, wird es möglich, Umgebungsinformationen gemeinsam zu sammeln,

gemeinsam Aufgaben auszuführen, welche von einem einzelnen Roboter nicht bewältigt werden könnten, oder auch Aufgaben wesentlich schneller auszuführen als ein einzelner Roboter[2, 5, 11].

Die Technologien, welche für die Umsetzung einer solchen Anwendung notwendig sind, sind inzwischen für jeden kostengünstig zugänglich. Micro PC's wie der Raspberry Pi sind schon für 20-30 Euro zu haben und auch Smartphones sind nicht zuletzt aufgrund ihrer zahlreichen Sensoren und Datenübertragungskanäle als Plattform für Roboter geeignet [3, 13]. Viele große Firmen stellen ihre Rechenkapazitäten für die private Verwendung preiswert zur Verfügung. Hierzu gehören z.B. die Amazon Webservices (E2C Cloud), Google (Google Code and App Engine), IBM (Eclipse and IBM Cloud) und Microsoft (Visual Studio and Azure) [5, 9, 14]. Projekte wie CloudStack, Open Nebula und Nimbus ermöglichen auf einfache Weise eine eigene Cloud Infrastruktur aufzubauen [15][ToDo Cloudstack Quellen]. Dies macht es auch für Privatpersonen und kleine Organisationen möglich ohne großen Aufwand in die Welt der vernetzten Robotik sowie des Cloud-Computing einzusteigen.

I. Ziel

Das Ziel dieser Arbeit ist es, auf Basis von CloudStack[1] und dem Zend Framework 2[2] eine Anwendung zu schaffen, welche es zum Einen möglich macht, Roboter über den Webbrowser zu überwachen sowie ihnen Missionen zuzuweisen und zum Anderen, Robotern den Aufbau einer gemeinsamen Wissensbasis zu ermöglichen. Prototypisch soll die Anwendung in einer Service orientierten Architektur (SOA) mit einer Beispielimplementierung eines einfachen Robotes als Client umgesetzt werden. Der Client soll durch die Anwendung auf der CloudStack Infrastruktur (Server) (M2C) [3] Befehle für die Ausführung der ihm zugewiesenen Mission erhalten, ohne selber die konkrete Abarbeitung der Mission zu überwachen.

II. Aufgabenstellung

Die Realisierung von Robocloud soll auf Basis von CloudStack und dem Zend Framework 2 erfolgen. Hierbei soll auf die Einhaltung bestehender Standards im Bereich der Webentwicklung, wie die konsequente Nutzung des MVC Design Pattern, einer umfassenden Modularisierung von Funktionen sowie der aspektorientierten Programmierung, geachtet werden. Durch den Einsatz aspektorientierter Programmierung soll eine Architektur geschaffen werden, welches eine einfache Erweiterung oder Anpassung des Systems ermöglicht.

Nach der Fertigstellung des Grundsystems soll aufbauend auf diesem prototypisch die selbstständige Abarbeitung einer Mission, welche aus mehreren einfachen Koordinaten besteht, durch das System und einen angemeldeten Client erfolgen. Die Abarbeitung soll hier ausdrücklich nur auf einfachste Weise realisiert werden und ohne den Einsatz erweiterter Wegplanung oder Filteralgorithmen erfolgen. Viel mehr steht die Evaluierung der Leistungsfähigkeit des Systems als heterogenes System im Vordergrund.

III. Abgrenzung

Das System soll ausschließlich prototypischen Charakter haben und nicht die Implementierung erweiterter Algorithmen aus dem Bereich der Robotik zeigen. Wie und in welchem Umfang die gesammelten Informationen für die weitere Auswertung und Nutzung für die Missionsplanung genutzt werden können, soll nur erwähnt werden. Konkrete Umsetzungen oder die Planung einer Umsetzung gehört nicht zum Umfang dieser Arbeit. Genau so ist die Realisierung der grafischen Oberfläche für den Webbrowser nicht Teil der Arbeit. Es wird angenommen, dass diese bereits vorhanden ist.

[1] http://cloudstack.apache.org
[2] http://zendframework.com
[3] M2C: Maschine to Cloud

II. cloudgestützte Robotic

Der Bereich der cloudgestützten Robotik beschäftigt sich mit der Anbindung von Robotern an Cloud-Ressourcen und der Bereitstellung von Robotern als Ressource innerhalb der Cloud [5, 6, 9]. Die Kommunikation mit anderen Ressourcen erfolgt hierbei entweder M2C (Machine-to-cloud) in einer Client / Server Architektur oder M2M (Mashine-to-machine) als Peer2Peer Netzwerk [2, 3, 6]. Grundlegend unterscheidet sich die cloudgestützte Robotik von der netzbasierten Robotik darin, dass externe Resourcen für die Abarbeitung von Programmcode verwendet werden, auf deren Verwaltung und Bereitstellung der Nutzer in der Regel keinen Einfluss halt. Zudem dient die Cloud als ein zentraler Punkt, mit welchem alle an das System gekoppelten Roboter kommunizieren. Im Gegensatz zu dezentralen Netzwerken bietet dies den Vorteil, dass die Netzwerklast weitaus geringer ist und die Rechenleistung nun nicht mehr an die Leistung der einzelnen Roboter gekoppelt ist, da weitere Ressourcen jederzeit in der Cloud zur Verfügung gestellt werden können [6] und weitere Kapazitäten jederzeit bereitgestellt werden können.

Weitere Vorteile bei der Nutzung von Cloudresourcen sind, auf Seiten der Roboter, die nahezu unbegrenzte Leistungsfähigkeit in Hinblick auf die zur Verfügung stehende Rechenleistung, eine Vereinfachung der Kommunikation der Roboter untereinander, die gemeinsame Datenbasis aller Roboter und ein geringerer Kostenaufwand für die Roboter, da keine leistungsstarke Hardware und entsprechend auch keine extrem leistungsstarken Akkus notwendig sind [1, 2, 4].

Der große Nachteil ist hingegen die Abhängigkeit von externen Ressourcen, wodurch ein Roboter, welcher stark abhängig von diesen ist, bei Störungen seiner Verbindung zur Cloud handlungsunfähig wird. Hier lassen sich zwar in vielen Fällen Ausweichroutinen implementieren, diese beinhalten aber aufgrund der geringen Leistung der Systeme meist nur das Herstellen eines sicheren Zustandes.

Aktuelle Anwendungen im Bereich der cloudgestützten Robotik sind zum Beispiel DAvinCi und RoboEarth. DAvinCi erleichtert Robotern den Zugriff und das Erstellen von 3D Umgebungsmodellen, um die eigene Lokalisierung in unbekannten Umgebungen zu verbessern, welche bereits von anderen Robotern besucht wurden [5, 14]. RoboEarth ist ein Internet für Roboter, in welchem Informationen und Verhaltensmuster gespeichert werden können [3, 14]. Der Zugriff auf diese ist für jeden angemeldeten Roboter möglich, wodurch dieser auch mit unbekannten Objekten interagieren kann, selbst wenn es sich um den ersten Kontakt mit einem Objekt handelt. Voraussetzung hierfür ist lediglich, dass dieses Verhalten bereits von einem anderen Roboter oder einem Menschen in das System eingepflegt wurde [3, 7, 8].

III. Robocloud Architektur

Im Folgendem wird der Begriff „Client" für ein an Robocloud angemeldetes Robotersystem und „Server" für die Robocloud Webanwendung verwendet. Die Begriffe Client und Server sind hier im Kontext einer Webanwendung zu sehen.

Die Architektur von Robocloud wurde von Anfang an auf maximale Skalierbarkeit ausgelegt, um zukünftig eine große Anzahl von Clients bedienen zu können. Zudem sollte sie sich einfach in bereits bestehende Infrastrukturen integrieren können. Als Basis für die private Cloud kommt bei Robocloud Cloudstack zum Einsatz. Als Anwendungsserver kommen mehrere kleine Webserverinstanzen mit dem Lighttpd[4] Webserver und einem PHP Interpreter zum Einsatz. Alle VMs haben Zugriff auf einen gemeinsamen Memcache und die MongoDB Datenbank, welche für die Persistierung der Daten zum Einsatz kommt. Die MongoDB ermöglicht zudem die Erstellung eines Geo-Index, welcher für den Datastore

[4] http://www.lighttpd.net/

von Robocloud benötigt wird, um Abfragen anhand von Standortdaten zu ermöglichen.

Da die Verbindungen der Clients auf das Robocloud System per HTTP erfolgen, welches zustandslos ist, werden Sessiondaten, Missionsdaten und Informationen über die angemeldeten Roboter mit Hilfe des Memcache zwischengespeichert. Dies macht diese auch über alle Anwendungsserverinstanzen erreichbar. Die Anwendungsserver, der Memcache und die MongoDB sind über eine private Netzwerkbrücke miteinander verbunden um einen direkten Zugriff von Außerhalb zu verhindern.

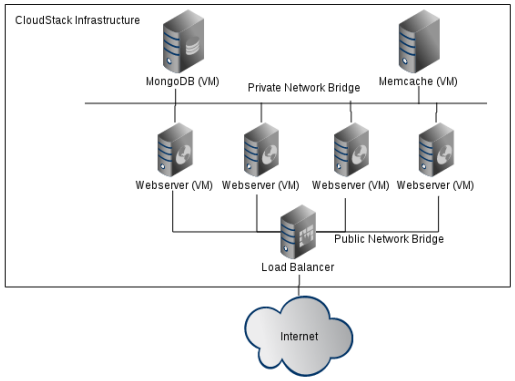

Abbildung 1: *Virtualisierung mit Cloudstack*

Robocloud an sich basiert auf dem Zend Framework 2 und nutzt Doctrine MongoDB ODM für den Zugriff auf die Datenbank. Das Zend Framework 2 ist modular aufgebaut und bietet somit die Möglichkeit der Modularisierung von Funktionen wie z.B. den Algorithmen für die Sensordatenverarbeitung. Die Schnittstellen, die Robocloud für den Zugriff von Clients anbietet, sind in JSON codiert um möglichst viele Clients unterstützen zu können.

Die Roboclound Anwendung an sich beinhaltet im Wesentlichen ein Modul mit folgenden Controllerklassen für die einzelnen Bereiche der Anwendung:

- MeasureController: Bietet Schnittstelle für die Übertragung der Messwerte und verarbeitet diese
- MissionController: Erstellen, Bearbeiten, Löschen und Zuweisen von Missionen zu Robotern
- RobotController: Erstellen, Bearbeiten und Löschen von Robotern
- StorageController: Schnittstellen für den privaten und öffentlichen Speicher

Die Verarbeitung der Sensordaten findet im Wesentlichen innerhalb des MeasureControllers statt, welcher über den EventManager des Zend Framework ein Ereigniss auslöst, auf welches andere Module lauschen können. Andere Module haben somit die Möglichkeit auf eintreffende Sensorwerte der Clients zu reagieren und die Rückgabe an den Client anzureichern. Diese Vorgehensweise hat den Vorteil, dass für die Erweiterung des Systems der Anwendungskern nicht angefasst werden muss. Zudem lassen sich hierdurch Fehler leichter lokalisieren.

Die Kommunikation zwischen dem Roboter und Robocloud ist aufgrund der Nutzung von HTTP für die Verbindung zum Einen zustandslos und zum Anderen hart synchronisiert. Bei der Anfrage übergibt der Roboter die aktuellen Messwerte seiner Sensoren sowie zusätzliche Informationen über den Systemstatus. Die Anfrage wird dann von Robocloud empfangen, die Messwerte werden, wenn gewünscht, gespeichert und eine Antwort wird auf Basis der Sensorinformationen, der aktuell laufenden Mission sowie weiteren Informationen generiert. Wenn alle Informationen zusammengetragen sind und alle Berechnungen beendet wurden, wird die Antwort an den Roboter zurück gesendet. Die in der Antwort enthaltenen Informationen werden vom Roboter nun in Aktorbefehle umgesetzt und die Verbindung wird geschlossen. Nach einem vom Roboter definierten Zeitintervall wird dann erneut eine Anfrage an das System gestellt. Die Länge des Zeitintervalls ist abhängig davon, wie groß der Anteil der Berechnungen ist, der vom Roboter selbstständig durchgeführt wird.

Grundsätzliche kann an dieser Stelle bei den Robotern zwischen Thin Clients und Fat Cli-

ents unterschieden werden. Thin Clients, also Roboter, welche wenig eigene Berechnungen anstellen, werden an Robocloud Anfragen im Intervall von 0.5 - 2 Sekunden senden. Fat Clients, also Roboter, welche nach dem Erhalt einer Mission diese selbstständig abarbeiten können, werden hingegen nur zu Beginn der Mission eine Anfrage an Robocloud senden, um einerseits den Start der Mission zu quittieren und andererseits die Missionsdaten zu laden.

Da aufgrund der technischen Gegebenheiten ausgehend von Robocloud keine Verbindung zum Roboter aufgebaut werden kann, können aktuell keine Ereigniss von Robocloud ausgehend an einen Roboter im System getriggert werden. Dieses Problem lässt sich jedoch durch die Verbindung von Socketverbindungen beheben. Im Falle der prototypischen Implementierung werden diese allerdings erst einmal keine Rolle spielen.

Für den Programmablauf ergeben sich auf einem einfachen Roboter, welcher als Thin Client umgesetzt wurde, 2 Programmschleifen. Die erste baut im Abstand von 0.5 Sekunden eine Verbindung zu Robocloud auf und übergibt die aktuelle Position und den Kurs. Aus der Antwort von Robocloud wird dann der Kurs, welcher für das Erreichen des nächsten Wegpunktes gefahren werden muss, lokal gespeichert. Die 2. Programmschleife schaut im Abstand von 0.1 Sekunden, welcher Kurs gehalten werden soll, vergleicht diesen mit dem aktuellen und generiert Befehle, welche dann an die Aktorik gesendet werden, um den vorgegebenen Kurs zu halten. Zudem speichert er gleichzeitig alle Sensorwerte lokal ab, damit diese bei der nächsten Verbindung zu Robocloud übergeben werden können.

IV. Schnittstellen

Robocloud bietet angemeldeten Robotern eine ausgewählte Zahl von Methoden, welche für das gemeinsame Arbeiten der Roboter unerlässlich sind.

- `/measure/send`
  Erlaubt das Senden von Sensordaten an Robocloud und gibt Informationen über die aktuell laufende Mission, Kursinformationen für den nächsten Wegpunkt, sowie Informationen über Roboter und Objekte in der Umgebung zurück.
- `/mission/save`
  Eine neue Mission speichern oder eine bestehende updaten
- `/mission/delete/[:id]`
  Löscht eine Mission anhand ihrer ID
- `/mission/get-json/[:id]`
  Gibt Informationen über eine Mission anhand ihrer ID zurück
- `/mission/assign?r=ROBOT_ID&id=MISSION_ID`
  Erlaubt das Zuweisen einer Mission zu einem Roboter
- `/mission/cancel?r=ROBOT_ID`
  Breche die aktuell laufende Mission ab
- `/data/set`
  Füge eine neue Informationen zu der globalen Datenbank hinzu oder update eine bestehende (setzt Berechtigungen vorraus)
- `/data/get/[:id]`
  Frage eine Information anhand ihrer ID ab
- `/data/delete/[:id]`
  Löscht eine Information aus dem Speicher (setzt Berechtigungen vorraus)
- `/data/geo?type=TYPElat=LATITUDE&lng=LONGITUDE&distance=DISTANCE`
  Abfrage einer Information anhand ihrer Geo-Koordinaten
- `/data/type?type=TYPE&page=PAGE`
  Abfrage aller Informationen eines Types

V. gemeinsames lernen

Um die Möglichkeit des gemeinsamen Lernens zu schaffen, verfügt Robocloud über einen Datastore, welcher neben normalen Key /

Value Einträgen auch das Ablegen von Geo-Daten erlaubt. Hierdurch können zum Beispiel Missionen, Objektdaten, Bewegungsmuster und Maps hinterlegt werden, auf die alle an das System angemeldeten Roboter zugreifen können. Die Ablage der Daten erfolgt in Form eines Key / Value Stores mit optionalen Geo-Index.

Der Geo-Index hat im Bezug auf die Robotik einen besonderen Stellenwert, da er es Robotern ermöglicht eine dynamische Karte der Umgebung zu erstellen [1]. Damit veraltete Daten nicht mit einbezogen werden, kann jedem Datensatz eine Ablaufzeit hinzugefügt werden. Nach dem Ablaufen der Lebenszeit des Eintrages wird dieser dann aus dem Datastore gelöscht.

Ein Eintrag innerhalb des Datastore sieht wie folgt aus:

- id: string, unique
- type: string, required
- geo: data, optional
- lifetime: int
- value: data

Der optionale Eintrag ”geo” ermöglicht die Angabe von x und y Koordinaten, anhand welcher später Umgebungsabfragen an das System gestellt werden. Die Felder ”id”, ”type” und ”geo” sind zusätzlich voll indiziert, wodurch Abfragen schnell bearbeitet werden können. Alle Abfragen werden zusätzlich gecached, um die Anzahl der Abfragen an den Datastore zu minimieren.

VI. Performance

Das System erreichte unter Laborbedingungen auf einem V-Server mit 2GB RAM über GSM eine durchschnittliche Antwortzeit von 208ms vom Verbindungsaufbau bis zum Schließen der Verbindung, welches einer möglichen Abfragefrequenz von ca. 4-5 Anfragen pro Sekunde entspricht. Berechnungen, welche in Echtzeit erfolgen müssen, eignen sich somit nicht für die Auslagerung auf Cloud-Infrastruktur. Berechnungen, die weniger als 3 mal pro Sekunde ausgeführt werden müssen, können hingegen gefahrlos ausgelagert werden. Ausgenommen hiervon sind systemkritische Prozesse, wie z.B. Backupprozesse, das Logging oder in den meisten Anwendungsgebieten direkte Aktorbefehle.

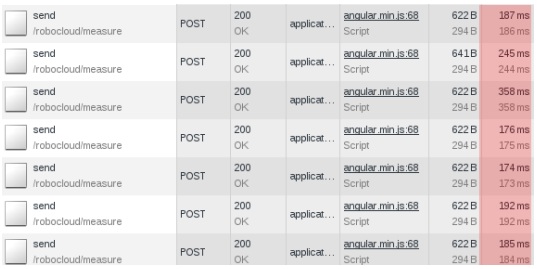

Abbildung 2: *Antwortzeiten von Robocloud über GSM*

Robocloud bietet mit seinen API's vorwiegend Methoden an, die in den meisten Anwendungsfällen in die Cloud ausgelagert werden. Direkte Aktorbefehle und andere zeitkritische Prozesse müssen vom Roboter in der Standartkonfiguration von Robocloud selbst ausgeführt werden. Da es sich hierbei in den meisten Anwendungsbereichen um einfachste Berechnungen handelt, kann man in diesem Zusammenhang weiterhin von Thin Clients reden.

Da Robocloud komplett Modular aufgebaut ist, ist es allerdings auch jederzeit möglich zeitkritische Prozesse in der Cloud auszulagern. Ob dies im Einzelfall sinnvoll ist, hängt in erster Linie von der Bandbreite, der Latenz und in einigen Fällen auch der Sicherheit der Verbindung zur Cloud ab.

VII. autonomes Segelboot Beispielimplementierung

Eine Beispielimplementierung eines Robocloud-Clients in Form eines autonomen Segelbootes wurde mit dem Raspberry Pi und der BrickPi Erweiterung durchgeführt. Beim Raspberry Pi handelt es sich um einen Kleinstkomputer mit ARM v6 32Bit Single Core CPU mit 700MHz, 512MB RAM (Modell B) sowie

2 USB Steckplätzen und Pins für die Erweiterung mit Zusatzmodulen. Der BrickPi ist ein entsprechendes Zusatzmodul, welches das Anschließen von Lego Mindstorm Sensoren und Aktoren an den Raspberry Pi ermöglicht. Das Segelboot verfügt über UMTS Anbindung an das Internet, einen Kompass, einem GPS Sensor, einem Servo Controller, sowie über Servos für die Ansteuerung des Ruders und der Segelwinde.

- Raspberry Pi (Modell B) http://www.raspberrypi.org/
- BrickPi http://www.dexterindustries.com/BrickPi/
- dGPS http://www.dexterindustries.com/dGPS.html
- dCompass http://www.dexterindustries.com/dCompass.html
- NXTServo-v3 8 channel ESC Servo Controller
- Huawei E173 Surfstick
- 8GB SD Karte
- EasyAcc 8400mAh

Die technische Komponenten des Systems ergeben zusammen genommen einen Anschaffungspreis von ca. 300 Euro (Stand Anfang 2014). Hinzu kommen die Kosten für das Modell und die Modellbauservos. Bei der Beispielimplementierung werden die Abarbeitung der Mission, sowie die Berechnung des Kurses zum Zielpunkt in der Robocloud durchgeführt. Der Roboter errechnet ausschließlich, welche Ruderbefehle notwendig sind, um den Zielpunkt zu erreichen.

VIII. Ergebnis und offene Fragen

Da Sensordaten an Robocloud aktuell noch unverschlüsselt übertragen werden, eignet es sich noch nicht für den Einsatz in sicherheitskritischen Bereichen. Die Sicherheit kann jedoch durch die Nutzung eines SSL Zertifikates sowie einer Verschlüsselung der Verbindung erhöht werden. Ein weiteres Sicherheitsproblem kann sich zudem durch die zentralisierte Speicherung der Daten ergeben. Durch die Kapselung der Datenbankserver und des Cache in einem privaten Netzwerk, wird dies oberflächlich verhindert, ist das System jedoch kompromittiert, kann der komplette Datenbestand ausgelesen werden und Rückschlüsse auf die Position und die Aufgabe eines Roboters sind möglich. Das System sollte also entsprechen den Standarts für sichere Webanwendungen abgesichert werden [2].

Ein Lasttest mit mehreren Robotern konnten aus Kosten und Zeitgründen leider auch nicht durchgeführt werden. Da die Infrastruktur jedoch skaliert, ist davon auszugehen, dass die Bedienung mehrerer Clients möglich ist. Dies muss aber noch evaluiert werden.

Literatur

[1] K. Kamei, et al., ”Cloud Networked Robotics”, IEEE Network, May 2012

[2] G. Hu, et al., ”Cloud Robotics: Architecture, Challenges and Applications”, IEEE Network, May 2012

[3] D. Lorencik and P. Sincak, ”Cloud Robotics: Current trends and possible use as a service”, IEEE, 2013

[4] L. Turnbull and B. Samanta, ”Cloud Robotics: Formation Control of a Multi Robot System Utilizing Cloud Infrastructure”, IEEE, 2013

[5] R. Arumugam, et al., ”DAvinCi: A Cloud Computing Framework for Service Robots”, 2010 IEEE International Conference on Robotics and Automation, May 3-8, 2010

[6] B. Dhiyanesh, "Dynamic Resource Allocation for Machine to Cloud Communications Robotics Cloud", IEEE 2012

[7] B. Kehoe, et al., "Cloud-Based Robot Grasping with the Google Object Recognition Engine"

[8] E. Hidago-Pe-na, et al., "Learning from the Web: Recognition Method Based on Object Appearance from Internet Images", IEEE 2013

[9] Y. Chen, et al., "Robot as a Service in Cloud Computing", IEEE 2010

[10] F. Ren, et al., "Robotics Cloud and Robotics School", IEEE 2011

[11] R. Doriya, et al., "Robotic Services in Cloud Computing Paradigm", International Symposium on Cloud and Services Computing, 2012

[12] W. Adiprawita, et al., "Service Oriented Architecture in Robotic as a Platform for Cloud Robotic (Case Study : Human Gesture Based Teleoperation for Upper Part of Humanoid Robot)", Institut Teknologi Bandung Ganesha 10, Bandung, Indonesia

[13] R. Tsuchiya, et al., "Simulation Environment based on Smartphones for Cloud Computing Robots", IEEE 2012

[14] L.Wang, et al., "Towards Cloud Robotic System: A Case Study of Online Colocalization for Fair Resource Competence", Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics December 11-14, 2012, Guangzhou, China 2012

[15] L. Wang, et al., "Towards Real-time Multi-Sensor Information Retrieval in Cloud Robotic System", 2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI) September 13-15, 2012. Hamburg, Germany

[16] J. Furrer, et al., "UNR-PF: An Open-Source Platform for Cloud Networked Robotic Services", 2012 IEEE/SICE International Symposium on System Integration (SII) Kyushu University, Fukuoka, Japan December 16-18, 2012