Erzeugung und inkrementelle Erweiterung großer 3D-Karten

Malte kl. PIENING, Benedikt SCHUMACHER, Marcel WIEGAND, Thomas WIEMANN, Felix IGELBRINK, Sebastian PÜTZ und Joachim HERTZBERG

Zusammenfassung

Durch die steigende Verfügbarkeit hochaufgelöster 3D-Daten gewinnt der Einsatz dreidimensionaler Karten für die Navigation mobiler Roboter an Bedeutung. Als Repräsentation bieten sich dafür Dreiecksnetze an, welche in der Regel offline aus den 3D-Daten generiert werden. Um Robotern auch eine Erkundung der Umgebung mit Hilfe solcher Karten zu ermöglichen, wird ein Verfahren zur inkrementellen online Erweiterung einer vorhandenen Karte benötigt. Die Funktionalität zur Erzeugung von solchen Karten auf City-Scale wurde in den vergangenen Jahren an der Universität Osnabrück entwickelt und als Open-Source-Software im Las Vegas Reconstruction Toolkit 2 (LVR2) zur Verfügung gestellt[1]. Dabei werden einzelne 3D-Punktwolken zunächst registriert und zur besseren Skalierbarkeit in kleineren Teilbereichen (Chunks) verarbeitet.

In diesem Beitrag präsentieren wir eine Erweiterung dieser Bibliothek, die es erlaubt, bereits erstellte Karten nachträglich online mit neuen Daten zu ergänzen. Der Ansatz besteht darin, nach der Registrierung neuer 3D-Punktwolken jeweils die neu erfassten Bereiche in die interne TSDF-Darstellung neu einzufügen bzw. bereits vorhandene Bereiche zu aktualisieren. Diese TSDF-Volumina dienen dann als Grundlage einer inkrementellen Polygonalisierung mittels Marching Cubes, um ein topologisch korrektes Dreiecksnetz als erweiterte Karte zu generieren.

1 Einleitung

Bei der autonomen Exploration von Umgebungen plant ein Roboter auf dem ihm aktuell bekannten Bereich seiner Umgebung einen Weg zu einer Position, von der aus weitere Teile der Umgebung aufgenommen werden können. Die dort aufgenommenen Daten müssen anschließend aufbereitet und in die bestehende Umgebungsrepräsentation integriert werden. Für die Navigation der Roboter bieten sich aufgrund des geringen Speicherbedarfs und der Zahl an existierenden Navigations-Algorithmen Dreiecksnetze an (PÜTZ ET AL. 2016). Dazu ist in unserem Szenario vor der Navigation eine Rekonstruktion der Oberfläche nötig aus den hochaufgelösten Punktwolken des 3D-Laserscanners notwendig.

[1] www.las-vegas.uni-osnabrueck.de

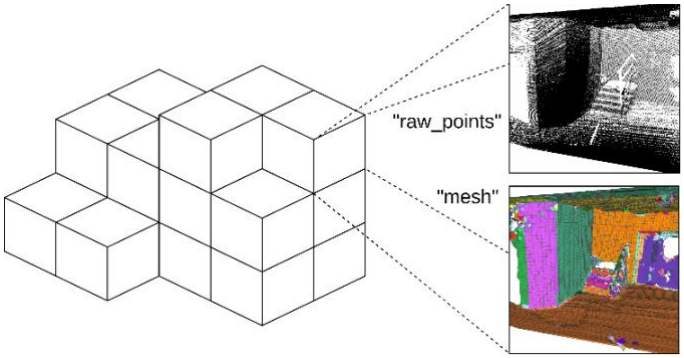

Abb. 1: Aufbau einer in Chunks aufgeteilten 3D-Karte

Eine solche Oberflächenrekonstruktion aus 3D-Punktwolken für große Umgebungen ist bereits im LVR2 implementiert worden (WIEMANN & MITSCHKE 2017). Bei diesem Ansatz wird die 3D-Punktwolke in Teilbereiche aufgeteilt, welche anschließend unabhängig voneinander rekonstruiert und anschließend zusammengefügt werden. Da dieser Ansatz das inkrementelle Einfügen neuer Daten jedoch nicht unterstützt, stellen wir in diesem Beitrag eine Erweiterung des bereits bestehenden Ansatzes vor.

Für die inkrementellen Erweiterung der 3D-Karte wird neben der Rekonstruktion auch eine Datenstruktur benötigt, welche sowohl große 3D-Karten verwalten kann als auch das inkrementelle Einfügen neuer Teilbereiche erlaubt. Außerdem muss diese Kartenrepräsentation die Annotation der enthaltenen geometrischen Daten mit Attributen unterstützen, um die für die Fahrplanung benötigten Kostenkarten erzeugen zu können (PÜTZ ET AL. 2016). Da die Navigation zudem nicht die gesamte Karte, sondern lediglich die befahrbaren Bereiche benötigt, sollte die Kartenrepräsentation zudem auch Teilbereichsanfragen unterstützen, die ein konsistentes Dreiecksnetz des angeforderten Bereichs liefern. Im folgenden Abschnitt stellen wir zur Lösung dieses Problems den Einsatz des von SOOHEE ET AL. (2012) vorgestellten hashbasierten 3D-Virtual-Grids in unserem Anwendungskontext vor, um die globale 3D-Karte in viele kleinere Teilkarten aufzuteilen.

2 Chunking Datenstruktur

Durch die hohe Auflösung der im Prozess der Exploration mit einem terrestrischen Laserscanner erzeugten Karten wird der Speicherbedarf dieser Karten schnell zu hoch, um die Karte im Arbeitsspeicher herkömmlicher Computer und somit auch vieler Roboter zu verarbeiten. Als Lösung dieses Problems teilen wir die aufgenommene Karte in viele einzelne Teilkarten, sogenannte Chunks, auf, die anschließend in Bereichsanfragen wieder vereint werden können, um die gewünschten Berechnungen auszuführen. Dabei ist die Kantenlänge eines Chunks in allen Dimensionen gleich und initial für einen Datensatz konfigurierbar.

Jeder Chunk besteht aus beliebig vielen Darstellungen, die entweder Dreiecksnetze oder Punktwolken des jeweiligen Chunks repräsentieren. Innerhalb eines Chunks habe diese Ebenen eindeutige Bezeichnungen. So besteht der in Abbildung 1 dargestellte Chunk aus den Ebenen *raw_points* für die ursprünglich aufgenommene Punktwolke und *mesh* für die daraus resultierte Oberflächenrekonstruktion. Diese Struktur ermöglicht es, rekonstruierte Oberflächen in verschiedenen Qualitätsstufen abzulegen und so ein Level-of-Detail Rendering, bei dem je nach Entfernung zum sichtbaren Bereich eine entsprechende Qualitätsstufe angezeigt wird, umzusetzen. Zudem können die in den Ebenen der Chunks abgelegten Punktwolken und Dreiecksnetze mit Attributen versehen werden. So wird es möglich, unter Betrachtung des Navigationsszenarios für Roboter auf den 3D-Karten, Analysen zur Befahrbarkeit eines Dreiecksnetzes durchzuführen und die Kartendaten entsprechend zu annotieren, bevor die Karte zur Fahrtplanung an den Roboter übertragen wird.

Die Chunks werden einerseits persistent auf dem Sekundärspeicher des Rechners abgelegt und in einem Cache im Hauptspeicher des Rechners vorgehalten. Die persistente Speicherung erfolgt im HDF5 Format, einem Datenformat zur effizienten Serialisierung n-dimensionaler Tensoren.

Der Cache sorgt neben der Speicherung auf dem Sekundärspeicher für einen schnellen Zugriff auf zuletzt verwendete Chunks. Dabei kommt ein hashbasiertes 3D-Virtual-Grid in Kombination mit einem LRU Cache zu Einsatz. Ein solcher LRU Cache entfernt beim Hinzufügen eines neuen Elements das am längsten nicht verwendete Element aus dem Cache, sofern die maximale Anzahl an Elementen im Cache erreicht ist. Das im Cache eingesetzte 3D-Virtrual-Grid enthält die Chunk-Daten während der LRU Cache die Verwaltung der geladenen Elemente übernimmt. Dieses 3D-Virtual-Grid stellt die zuvor genannte Struktur der Chunks in der Karte dar, indem mittels einer Map von dem Hashwert eines Chunkindexes auf den dazugehörigen Chunk abgebildet wird. Der Vorteil bei dieser Repräsentation ist der alleinige Speicherverbrauch für geladene Chunks und die konstante Zugriffszeit auf die jeweiligen Chunks. Die eingesetzte Hashfunktion ist in der Formel 1 abgebildet. Dabei befindet sich die Position eines Chunks in Chunk-Koordinaten. So hätte der Chunk mit der Position *(1, 1, 1)* seine dem Koordinatenursprung nächste Ecke im Weltkoordinatensystem an der Position *(Chunkgröße, Chunkgröße, Chunkgröße)*. Der Parameter *minChunks* gibt den minimalen Index der Boundingbox des gechunkten Bereichs, ebenfalls in Chunk-Koordinaten, an. Dieser wird als Offset von der Chunkposition abgezogen, um alle Chunk-Koordinaten in den positiven Bereich zu verschieben, um Kollisionen von Hashwerten im durch die Boundingbox definierten Bereich der Chunk-Positionen zu verhindern. Anschließend werden die resultierenden positiven Chunk-Positionen entsprechend der Hashfunktion aus WIEMANN & MITSCHKE (2017) auf Skalare abgebildet. Dieser Ansatz der Hashfunktion hat vor allem den Vorteil, dass bei bekannter Reichweite der Chunk-Positionen ein Zugriff auf die Nachbarn eines Chunks über einen konstanten Offset in der Hashfunktion möglich ist. Dennoch müssen bei einer Veränderung der Boundingbox der Chunks die Hashes alle gecachten Chunks im 3D-Virtual-Grid aktualisiert werden.

$$h(\overrightarrow{pos}) = \left(\overrightarrow{pos} - \overrightarrow{minChunks}\right) \begin{pmatrix} chunksY \cdot chunksZ \\ chunksZ \\ 1 \end{pmatrix} \quad \textbf{1}$$

Ein konsistentes Mesh wird auf die entsprechenden Chunks aufgeteilt, indem alle Faces mit den zugehörigen Vertices den entsprechenden Chunks zugeordnet werden. Der Zuordnung von Face zu Chunk wird bestimmt, indem der Schwerpunkt des Faces betrachtet und das Face dem Chunk, der den Schwerpunkt enthält, zugeordnet wird. Da jeder Chunk ein konsistentes Mesh enthalten soll, um auch chunkweise die enthaltenen Daten verarbeiten zu können, kommt es dabei zur Duplizierung von Vertices zwischen benachbarten Chunks. Da nicht vorausgesetzt werden kann, dass die Kanten der Faces genau an den Grenzen der Chunks liegen, wird ein Überlappen der Boundingboxen aneinander liegender Chunks zugelassen. Um diesen Bereich jedoch möglichst klein zu halten, werden die Kanten, die zu weit in benachbarte Chunks hineinragen, aufgeteilt. Dieser Prozess ist in den Abbildungen 2 und 3 dargestellt. Dabei ist einerseits das aufzuteilende Dreiecksnetz mit den Chunkgrenzen in der Abbildung 2 und das Ergebnis des Aufteilens in der Abbildung 3 visualisiert. Die in blau markierten Kanten in der Abbildung 3 stellen die durch das Aufteilen der zu großen Kanten neu hinzugefügten Kanten dar.

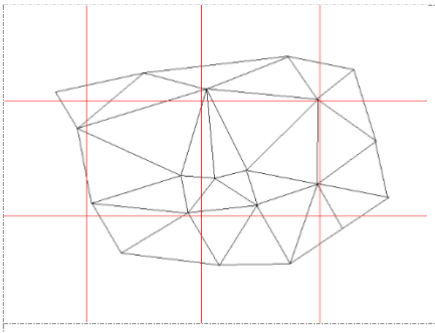

Abb. 2:
Aufzuteilendes Dreiecksnetz (schwarz) mit Grenzen der Chunks (rot)

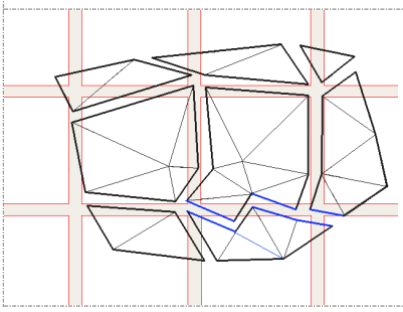

Abb. 3:
Aufgeteiltes Dreiecksnetz (schwarz) mit ergänzten Kanten für Auflösung zu großer Überlappungen (blau) und Chunkgrenzen (rot)

Um die Dreiecksnetze einzelner Chunks performant zusammenfügen zu können, werden im Prozess des Aufteilens die duplizierten Vertices an den Anfang der Vertexlisten der jeweiligen Chunks geschrieben. Zudem wird auch die Anzahl der duplizierten Vertices gespeichert. Diese Information wird anschließend beim Zusammenfügen der Chunks verwendet, um die Suche nach Duplikaten einzuschränken und so den Prozess zu beschleunigen.

3 Inkrementelles Einfügen in die Rekonstruktion

Bei der inkrementellen Erweiterung von 3D-Karten sollen nur die Teile der 3D-Punktwolke neu polygonalisiert werden, die entweder Daten eines neu einzufügenden Bereichs oder frühere Scan-Daten , bei denen sich durch die erneute Registrierung Posen signifikant verändert haben, beinhalten. Diese Herangehensweise soll, insbesondere bei großen Datensätzen, die Bearbeitungszeit der Daten erheblich verringern, da nicht pro neu aufgenommenem Laserscan die vollständige 3D-Punktwolke verarbeitet wird. Bisher wurden große 3D-Karten mit einem *k*-d-Baum in balancierte Zellen unterteilt (WIEMANN & MITSCHKE 2017). Da sich k-d-Bäume aber nur schwer in beliebige Richtung erweitern lassen, baut das hier beschriebene Verfahren auf der zuvor genannten Chunking-Datenstruktur auf.

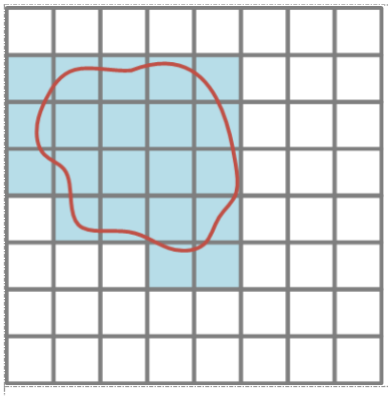

Abb. 4:
Initiale 3D-Punktwolke (rot), die in Chunks (blau) unterteilt wird.

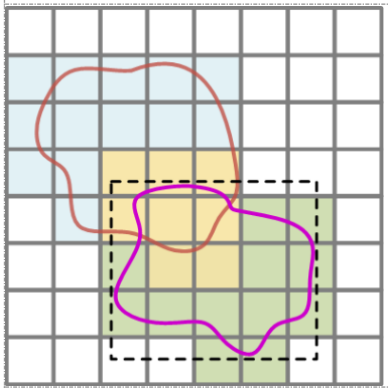

Abb. 5:
Violett: Inkrementell eingefügter Laserscan
Gelb: Chunks im Überlappungsbereich
Grün: neu erzeugte Chunks
Gestrichelte Linie: Boundingbox des Laserscans

Für die initiale Rekonstruktion der Oberfläche wird die 3D-Punktwolke gemäß der oben genannten Chunking-Datenstruktur in einzelne Zellen fester Größe aufgeteilt (vgl. Abb. 4). Die Randbereiche der Chunks überlappen sich dabei. Anschließend wird für jeden Chunk parallelisiert ein Voxelgrid generiert und pro Voxel die TSDF-Values berechnet (HOPPE ET AL. 1992). Das resultierende Voxelgrid wird als Ebene mit den dazugehörigen TSDF-Values in der Chunking-Datenstruktur abgelegt.

Die Information darüber, welche Scandaten neu zu rekonstruieren sind, folgt aus der Registrierung. Nach Abschluss der Registrierung wird für alle zu rekonstruierenden Daten die Vereinigung der Boundingboxen berechnet. Anhand dieser Information werden danach die korrespondierenden Zellen im 3D-Virtual-Grid bestimmt (vgl. Abb. 5). Die Rekonstruktion der betroffenen Zellen erfolgt parallel. Das Voxelgrid wird anschließend wieder in die Chunking-Datenstruktur eingefügt. Sollten dabei bereits einzelne Chunks existieren, werden diese mit den neu berechneten Werten aktualisiert.

Da die Befahrbarkeitsanalyse von PÜTZ ET AL. (2016) Dreiecksnetze als Eingangsdaten erfordert, wird aus den zuvor gewonnenen TSDF-Values mittels Marching Cubes die entsprechende Oberfläche erzeugt (LORENSEN & CLINE 1987). Für die Befahrbarkeit werden auf der Oberflächenrepräsentation, bzw. den Teilbereichen Höhendifferenzen und die Geländeunebenheit berechnet. Abschließend wird der aktualisierte Bereich des Dreiecksnetzes und die Befahrbarkeitsdaten entsprechend in Chunks unterteilt, in die Datenstruktur und zuletzt in eine HDF5-Datei eingefügt.

4 Experimentelle Analyse des Verfahrens

4.1 Analyse der Ressourcennutzung der Rekonstruktion

Um die Ressourcennutzung unseres Ansatzes experimentell zu überprüfen haben wir aus einem Datensatz mit 12 Scans eines Riegl VZ400i zunächst die ersten 11 Scans rekonstruiert und anschließend den verbleibenden Scan in die im ersten Schritt entstandene Karte eingefügt. Die Scans enthielten jeweils ca. 18 Millionen Punkte, wodurch der gesamte Speicherverbrauch etwa 12 GB betrug. Dabei haben wir die Auslastung des Arbeitsspeichers über die verstrichene Zeit der Rekonstruktion aufgezeichnet und in Abbildung 6 dargestellt. Die Rekonstruktion erfolgte dabei auf einem Prozessor des Typs Ryzen 7 1700 ohne Grafikkartenbeschleunigung.

Die Abbildungen zeigen die Arbeitsspeicherauslastung im Verlauf der Rekonstruktionen. Dabei ist zu erkennen, dass die Rekonstruktion in beiden Szenarien eine etwa gleichbleibende Arbeitsspeicherauslastung vorweisen. So benötigt die Initialrekonstruktion mit 11 Scans im Maximum etwa 8750MB des Arbeitsspeichers während die Erweiterung um den Scan 12 im Maximum etwa 9200MB benötigt. Zudem ist erkennbar, dass die Arbeitsspeicherauslastung bei der Initialrekonstruktion stärker schwankt als die Rekonstruktion der inkrementell erweiterten Karte. Dies ist vor allem auf die Anzahl der zu rekonstruierenden Chunks zurückzuführen, die im ersten Szenario wesentlich höher ausfällt als bei der Erweiterung der Karte.
Außerdem ist anzumerken, dass die Rekonstruktion im Prozess der Erweiterung in unserem Experiment etwa die Hälfte der Zeit der Initialrekonstruktion in Anspruch nimmt. Diese Ergebnisse zeigen, dass unser Ansatz zur inkrementellen Erweiterung von 3D Karten funktioniert und so nur von der Erweiterung betroffene Bereiche neu rekonstruiert werden.

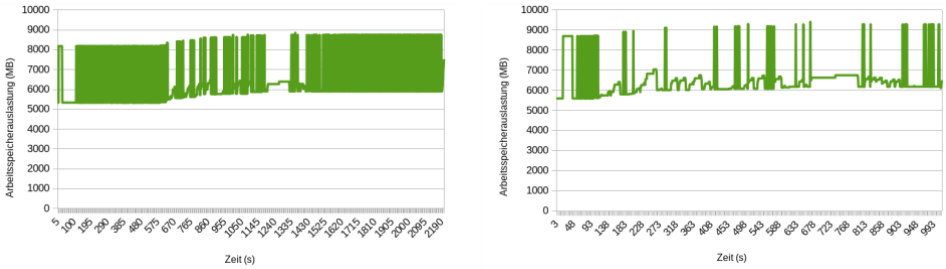

Abb. 6: Arbeitsspeicherauslastung bei Initialrekonstruktion mit 11 Scans (links) sowie bei Erweiterung der Initialrekonstruktion um einen weiteren Scan (rechts).

4.2 Vergleich mit *k*d-Baum-Verfahren

Tabelle 1: Laufzeitanalyse bei ca. 72,5 bzw. 76 Mio. Punkten (GPU-beschleunigt)

Aufteilungsansatz	**Laufzeit**	
	5 Scans	**5 + 1 Scans**
*k*d-Baum	01:04:40	1:17:51
Chunking Datenstruktur	01:52:59	0:11:48

Im direkten Vergleich zwischen den Aufteilungsverfahren der 3D-Punktwolke in Tabelle 1 fällt auf, dass die Laufzeit für die initiale Rekonstruktion eines großen Datensatzes mittels der in Absatz 2 erläuterten Chunking-Datenstruktur deutlich langsamer ist als die ursprüngliche Herangehensweise. Grund dafür ist vor allem, dass der *k*d-Baum Chunks verschiedener Volumina, aber mit ähnlicher Anzahl an Punkten generiert. Bei der schnellen Aufteilung mittels Chunking-Datenstruktur variiert die Anzahl der Punkte innerhalb eines Chunks stark. Sollen jedoch neue Daten zu einer existierenden 3D-Karte hinzugefügt werden ist die Rekonstruktion basierend auf der Chunking-Datenstruktur deutlich effizienter. Bei Verwendung des bisherigen Ansatzes muss der *k*d-Baum neu generiert und die 3D-Punktwolke vollständig neu polygonalisiert werden, was zu einer deutlich höheren Laufzeit führt.

5 Zusammenfassung und Ausblick

Die Chunking Datenstruktur ermöglicht zusammen mit dem vorgestellten Rekonstruktionsverfahren die inkrementelle Erstellung von 3D-Karten während der Exploration von großen Arealen mit einem mobilen Roboter. Außerdem hat das präsentierte Verfahren im Gegensatz zu dem Ansatz von WIEMANN & MITSCHKE (2017) den Vorteil, dass bestehende Datensätze aktualisiert und um neue Daten ergänzt werden können. Die experimentellen Untersuchungen zeigen, dass unser Verfahren bei der initialen Erstellung

eine größere Laufzeit aufweist. Jedoch kann in Szenarien, bei denen die 3D-Karte erweitert werden muss, eine geringe Laufzeit als bei WIEMANN & MITSCHKE (2017) erzielt werden. Durch die Chunking Datenstruktur können außerdem einzelne Teilbereiche der Karte abgefragt werden. So können z.B. alle Teilbereiche abgefragt werden, welche zur Pfadplanung einer Roboter Navigation benötigt werden, anstatt die gesamte Karte zu laden.

Das vorgestellte Verfahren bietet auch Potential für Erweiterungen. So könnte z.B. ein Prozess zum dynamischen Laden einzelner Chunks für die Roboternavigation entwickelt werden. Ebenfalls denkbar wäre eine "On-demand Polygonnetz Generierung" wodurch sich die Zeit des "Pre-Processings" verringern würde und das Dreiecksnetz individuell auf den jeweiligen Anwendungszweck angepasst werden könnte. Neben den Erweiterungen gibt es auch Ansätze zur Leistungssteigerung, indem die gespeicherten TSDF-Values der zu aktualisierenden Bereiche, mit denen aus der neuen Punktwolke ermittelten TSDF-Daten gemittelt werden, anstatt die Bereiche komplett neu zu rekonstruieren.

Literatur

Wiemann, T & Mitschke, I (2017): *Massiv parallelisierte Oberflächenrekonstruktion aus großvolumigen und hochaufgelösten 3D-Punktwolken.* In: Luhmann/Müller (eds.): Photogrammetrie, Laserscanning, Optische 3D-Messtechnik. Beiträge der Oldenburger 3DTage 2017, Wichmann Verlag, 2017

Pütz, S & Wiemann, T & Sprickerhof, J & Hertzberg, J (2016): *3D Navigation Mesh Generation for Path Planning in Uneven Terrain*. Proc. IAV 2016 -- 9th IFAC Symposium on Intelligent Autonomous Vehicles, Leipzig, June 29-July 1st, 2016

Hoppe, H & DeRose, T & Duchamp, T & McDonald, J & Stuetzle, W (1992): *Surface Reconstruction from Unorganized Points.*SIGGRAPH Comput. Graph. 26 (1992), Juli, Nr. 2, 71–78

Lorensen, W. E & Cline, H. E (1987): *Marching cubes: A high resolution 3D surface construction algorithm.* ACM SIGGRAPH, 1987

Soohee, H & Sangmin, K & Jae Hoon, Jung & Changjae, K & Kiyun, Y & Joon, H (2012): *Development of a hashing-based data structure for the fast retrieval of 3D terrestrial laser scanned data.* Computers & Geosciences, 2012

Bitte geben Sie auf einer separaten Seite zu allen genannten Autoren folgende Informationen:

Wiemann, Thomas, Dr.
Universität Osnabrück, Institut für Informatik
twiemann@uni-osnabrueck.de

kl. Piening, Malte
Universität Osnabrück, Institut für Informatik
mklpiening@uni-osnabrueck.de

Schumacher, Benedikt
Universität Osnabrück, Institut für Informatik
bschumacher@uni-osnabrueck.de

Wiegand, Marcel
Universität Osnabrück, Institut für Informatik
mawiegand@uni-osnabrueck.de

Igelbrink, Felix
Universität Osnabrück, Institut für Informatik
felix.igelbrink@uni-osnabrueck.de

Pütz, Sebastian
Universität Osnabrück, Institut für Informatik
spuetz@uni-osnabrueck.de

Hertzberg, Joachim, Prof. Dr.
Universität Osnabrück, Institut für Informatik
DFKI-Labor Niedersachsen – Planbasierte Robotersteuerung
joachim.hertzberg@uni-osnabrueck.de

Diese Angaben werden im Autorenverzeichnis am Ende des Tagungsbandes aufgeführt.