Energieeffizientes FPGA-basiertes TSDF-SLAM für Mobile Mapping

Thomas WIEMANN, Marc EISOLDT, Marcel FLOTTMANN, Julian GAAL,
Marc ROTHMANN, Marco TASSEMEIER, Mario PORRMANN

Zusammenfassung

Rekonfigurierbare System-on-Chips (SoCs) mit FPGA-Beschleunigern kombinieren die Eigenschaften von klassischen CPU-basierten Systemen mit der Möglichkeit, Algorithmen in Hardware zu beschleunigen. Im Gegensatz zu klassischen GPUs arbeiten die FPGA-Beschleuniger dabei sehr energieeffizient. Gerade für Anwendungen wie Mobile Mapping mit UAVs oder anderen sich bewegenden Systemen ist mit Hinblick auf eine möglichst lange Laufzeit der Stromverbrauch ein wesentlicher limitierender Faktor. In diesem Beitrag präsentieren wir ein in sich geschlossenes System mit LiDAR und IMU, dass die Daten eines Ouster OS1-128 LiDARs mit 20 Hz in eine TSDF-Karte einträgt. Durch diese Datenfusion in Echtzeit weist das System einen sehr geringen inkrementellen Fehler bei der Lokalisierung auf. Die TSDF-Darstellung erlaubt es zudem, eine polygonale Darstellung der Umgebung in Echtzeit zu erzeugen. Die Nutzung des SoCs mit FPGA reduziert den Energieverbrauch dabei um den Faktor 18 verglichen mit einer klassischen CPU-Implementierung.

1 Einleitung

Beim Laserscannen auf mobilen Plattformen ist es wünschenswert, dass die eingehenden Daten direkt auf dem System verarbeitet werden, um die Qualität der Ergebnisse bereits während des Scanprozesses beurteilen zu können. Dafür ist eine Integration der Messdaten in Echtzeit erforderlich. Leider ist die verfügbare Rechenleistung auf mobilen Geräten in der Regel gering. Auch das durch die Akkus vorgegebene Energiebudget ist begrenzt. Selbst moderne Computer wie Laptops können bei hoher Last im Idealfall nur einige wenige Stunden betrieben werden. Eine Alternative zu solchen CPU-basierten System sind System-on-Chips (SoCs) mit integrierten FPGAs, welche die Beschleunigung des Algorithmus mithilfe von rekonfigurierbarer Hardware ermöglichen. FPAGs arbeiten äußerst energieeffizient und lassen sich flexibel auf die zu lösende Problemstellung anpassen. Sie sind daher ein ideales Mittel, um Algorithmen auf mobilen Plattformen zu implementieren.

Eine zentrale Herausforderung bei der Programmierung von FPGAs ist, dass eine umfangreiche Auseinandersetzung mit der verwendeten Systemarchitektur unvermeidbar ist, wenn die maximale Performanz erzielt werden soll. Zudem sind solche Beschleuniger vor allem für parallele Algorithmen geeignet. Daher werden in unserer Implementierung nur die Programmteile auf den FPGA ausgelagert, die sich gut parallelisieren lassen. Alle anderen Systemkomponenten laufen auf der CPU mit einem angepassten Linux-Basissystem. Dieses Basissystem stellt u.a. die Schnittstellen und Treiber zu den verwendeten Sensoren sowie

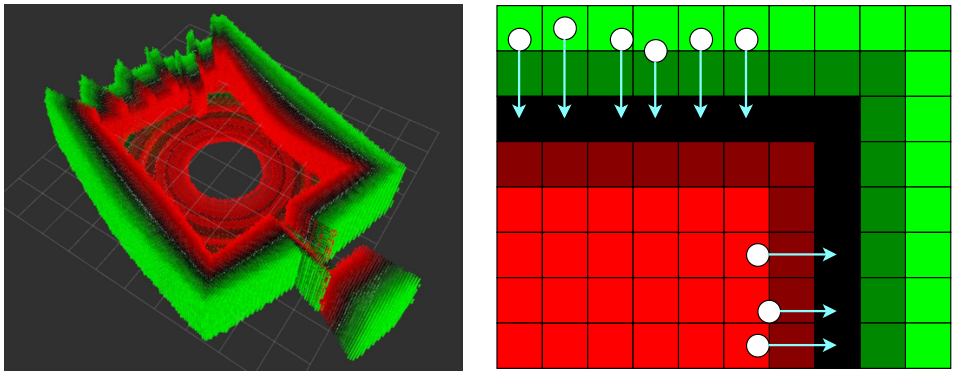

Abb. 1: Visualisierung einer TSDF-Repräsentation (links) und das Prinzip der Registrierung von Scanpunkten gegen eine gegebene TSDF-Karte.

eine für SoCs angepasste Basisversion des Robot Operating Systems (ROS) namens ReconfROS zur Verfügung (EISOLDT ET AL. 2021).

Im Folgenden beschreiben wir den teilweise in Hardware umgesetzten TSDF-SLAM-Algorithmus und den grundlegenden Aufbau unserer entwickelten SLAM-Box. Anschließend evaluieren wir das System anhand von selbsterstellten Datensätzen und vergleichen die Lokalisierungsqualität mit bekannten Referenzdatensätzen aus der mobilen Robotik.

2 TSDF-SLAM

TSDF-basiertes SLAM wird in der Robotik schon seit einigen Jahren vor allem für die Rekonstruktion von Oberflächen anhand von Tiefenkameras verwendet (IZADI ET AL., 2011). Als Kartenrepräsentation dient dabei ein sogenannteS *Truncated Signed Distance Field*. Das Prinzip dieser Datenstruktur ist in Abbildung 1 links skizziert. Der zu erfassende Raum wird in ein dreidimensionales Voxel-Gitter unterteilt. Für jeden Eckpunkt eines Voxels wird der relative Abstand zur nächsten Oberfläche abgespeichert. Positive Werte sind in grün dargestellt, negative Werte in Rot. Die Vorzeichen ergeben sich dabei aus der Orientierung der Flächennormalen. Die zu repräsentierende Oberfläche zeichnet sich durch einen TSDF-Wert von 0 aus (dargestellt in schwarz). Abstände, die eine Entfernung von mehr als ϵ haben, werden als Freiraum angesehen. Diese Anteile des Volumens werden durch einen speziell reservierten Wert repräsentiert. Der Vorteil einer solchen Darstellung ist, dass sie sich auf GPUs hervorragend verarbeiten lassen. Zudem ist die Erzeugung eines Dreiecksnetzes aus dieser impliziten Flächenrepräsentation mittels Marching Cubes (LORENSEN & CLINE, 1987) leicht umzusetzen. Der Nachteil ist, dass die Repräsentation viel Arbeitsspeicher benötigt. Daher konnten erste Algorithmen wie KinectFusion (IZADI ET AL., 2011) nur ein sehr kleines Volumen repräsentieren. Spätere Erweiterungen wie Kintinous (WHELAN ET AL., 2012) erlaubten durch einen Ringbuffer auch größere Areale zu kartieren. Eine weitere Besonderheit dieser Algorithmen ist, dass sie für Tiefenbilder entwickelt wurden. Eine direkte Übertragung auf LiDAR-Daten ist schwer möglich, da diese in der Regel nicht räumlich strukturiert vorliegen. Der in unserem SLAM-System implementierte Point-

to-TSDF-Algorithmus basiert daher auf dem von CANELHAS ET AL. (2013) vorgestelltem Verfahren in Kombination mit dem in WIEMANN UND MITSCHKE (2020) vorgestellten Ansatz zur Segmentierung von TSDFs. Unser Verfahren erlaubt somit auch die Verarbeitung von unstrukturierten Umgebungsmessungen. Die Registrierung wird dabei als Optimierungsproblem formuliert:

$$\xi^* = \text{argmin}_\xi \sum_i^{|P|} \|D(T(\xi) \cdot p_i)\|_2^2 \tag{1}$$

Dabei ist

$$\xi^T = [\omega_1 \quad \omega_2 \quad \omega_3 \quad v_1 \quad v_2 \quad v_3] \tag{2}$$

die Beschreibung des aktuellen Zustands mit Winkelgeschwindigkeit ω und Lineargeschwindigkeit v. $T(\xi)$ ist die Transformationsmatrix, die aus dem aktuellen Bewegungszustand ξ abgeleitet und auf die Messpunkte p_i angewendet wird. Die Funktion D gibt den TSDF-Wert für einen Raumpunkt zurück. Entsprechend ist (1) dann minimal, wenn alle Punkte p_i auf einer Nullstelle des TSDF-Feldes liegen. Zur Lösung wird eine Taylorreihenentwicklung 1.Ordnung um $\xi = 0$ durchgeführt:

$$D(T(\xi) \cdot p_i) \approx D(T(\xi) \cdot p_i)|_{\xi=0} + (\nabla_\xi D(T(\xi) \cdot p_i)|_{\xi=0})^T \cdot \xi \tag{3}$$

Unter der Annahme, dass die Bewegung zwischen zwei Scans gegen Null geht (hohe Scanfrequenz) ergibt sich aus Gleichung (3):

$$D(T(\xi) \cdot p_i) \approx D(p_i) + (\nabla_\xi D(p_i))^T \cdot \xi \tag{4}$$

Mit der Jacobimatrix $J(p_i) = \nabla_\xi D(p_i)$ lässt sich Gleichung (4) vereinfacht schreiben als

$$D(T(\xi) \cdot p_i) \approx D(p_i) + J(p_i) \cdot \xi \tag{5}$$

Durch Anwendung der Kettenregel und Auflösung der partiellen Ableitungen ergibt sich approximativ

$$J(p) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{\delta D(p)}{\delta x} \\ \frac{\delta D(p)}{\delta y} \\ \frac{\delta D(p)}{\delta z} \end{bmatrix} \tag{6}$$

Eingesetzt in die Taylorreihenentwicklung, kann die Zielfunktion wie folgt umgeschrieben werden:

$$\xi^* = min_\xi \sum_i^{|P|} \xi^T \cdot J(p_i) \cdot J(p_i)^T \cdot \xi + \xi^T \cdot J(p_i) \cdot D(p_i) + D(p_i)^2 \tag{7}$$

Im nächsten Schritt wird diese Gleichung nach ξ abgeleitet und die Nullstellen bestimmt, um das Minimum der Funktion zu erhalten:

$$0 = \sum_i^{|P|} J(p_i) \cdot J(p_i)^T \cdot \xi^* + \sum_i^{|P|} J(p_i) \cdot D(p_i) \tag{8}$$

Mit $H = \sum_i^{|P|} J(p_i) \cdot J(p_i)^T$ und $g = \sum_i^{|P|} J(p_i) \cdot D(p_i)$ ergibt sich vereinfacht:

$$\xi^* = -H^{-1} \cdot \mathrm{g} \tag{9}$$

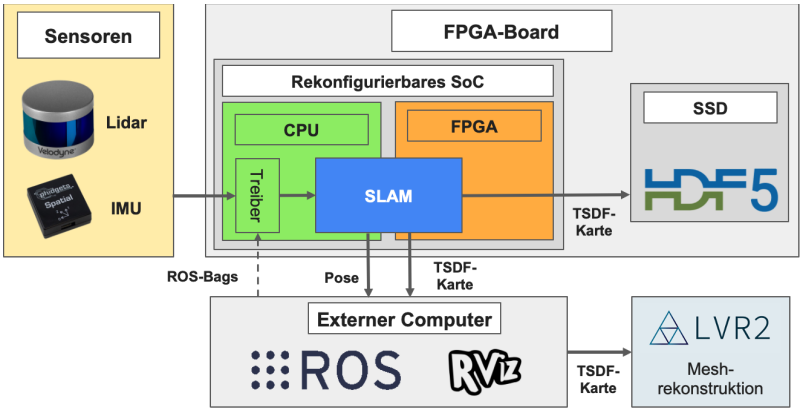

Abb. 2: Überblick über die Systemarchitektur der SLAM-Box, Integration der Sensoren und Kommunikation mit externen Komponenten.

Analog zum ICP-Algorithmus werden diese Berechnungen nach der Transformation der Punkte wiederholt, bis keine Änderungen in der Transformation mehr zu beobachten sind oder eine vorgegebene maximale Anzahl von Iterationen erreicht ist. Veranschaulicht wird die Funktionsweise im rechten Bild von Abbildung 1. Die weißen Scanpunkte werden entlang des Gradienten im TSDF-Feld in Richtung der Nullstellen verschoben.

3 Systemdesign

Der grobe Aufbau unseres SLAM-Systems ist in Abbildung 2 dargestellt. Das FPGA-Board besteht aus dem SoC mit ARM-CPU und FPGA sowie einer SSD, auf der die erzeugten Karten zwischengespeichert werden. Das SoC stellt das Linux-Betriebssystem zur Verfügung. Darüber können über die entsprechenden Schnittstellen und Treiber die externen Sensoren angesprochen werden. Des Weiteren ist eine auf dieses System angepasste Version des Robot Operating Systems (ROS) namens ReconfROS auf dem Board installiert (EISOLDT ET AL., 2021). Dieses stellt die Software-Schnittstellen zur Verfügung, mit denen die geschätzte Pose sowie die erzeugten Karten, z.B. zur Visualisierung mit RViz, an externe Computer weitergegeben werden können. Optional lassen sich die TSDF-Karten nach dem Scanprozess mit der LVR2-Bibliothek polygonalisieren. Der eigentliche SLAM-Algorithmus ist so implementiert, dass die Schritte, die sich nicht gut parallelisieren lassen, auf der CPU laufen. Die Datenintensiven Operationen laufen parallelisiert auf dem FPGA.

Details zur eigentlichen Implementierung sind in Abbildung 3 dargestellt. Im Wesentlichen laufen auf dem FPGA zwei Kernel: einer ist dafür zuständig, die Karte zu aktualisieren, der andere setzt die Registrierung der 3D-Punkte gegen die TSDF-Repräsentation um. Die Scanpunkte, die lokale Karte sowie eine temporäre Kopie der lokalen Karte werden im DRAM abgelegt. Die Kernel im FPGA können darauf über sogenannte Memory-Ports zugreifen. Die Kernel selbst laufen dabei in mehreren Instanzen (Processing Units) parallel.

Der Registrierungskernel besteht dabei aus mehreren Komponenten. Zunächst werden die Messpunkte, gemäß der aktuellen Poseschätzung, transformiert. Mit Hilfe der TSDF-

Repräsentation in der lokalen Karte und Gleichung (8) werden für die einzelnen Punkte die partiellen Einträge der Matrizen H und g berechnet. Diese Einträge werden dann zu den entsprechenden Gesamtmatrizen kombiniert, sodass mittels Gleichung (9) die Lösung ξ^* für das Optimierungsproblem berechnet werden kann. Mittels $T(\xi^*)$ kann aus ξ^* die dazugehörige Transformationsmatrix bestimmt werden, mit der die Punkte in das globale Koordinatensystem transformiert werden.

Nach der Berechnung der Transformation muss die gewonnene Information in die Karte eingetragen werden. Dazu gibt es den Map-Update-Kernel. Dieser projiziert die transformierten Punkte in das TSDF-Volumen. Dies geschieht mit einem Ray-Marching-Ansatz. Dabei werden entlang der Strahlen ausgehend von der Pose des Scanners die Elemente im Volumen als frei markiert, bis die gemessene Entfernung erreicht ist. Im Bereich ϵ um diesen Oberflächenwert werden die TSDF-Einträge mit den passenden Entfernungen aktualisiert. Da es bei diesem Verfahren aufgrund von Sensorrauschen und Registrierungsfehlern zu Inkonsistenzen kommen kann, werden die TSDF fortlaufend durch einen gleitenden Mittelwert interpoliert (TSDF-Interpolation). Die Interpolierten Werte werden in einer temporären Karte abgelegt. Sobald die Sensorpose einen vordefinierten Offset überschreitet, wird die temporäre Karte in die lokale Karte übertragen (Map Update).

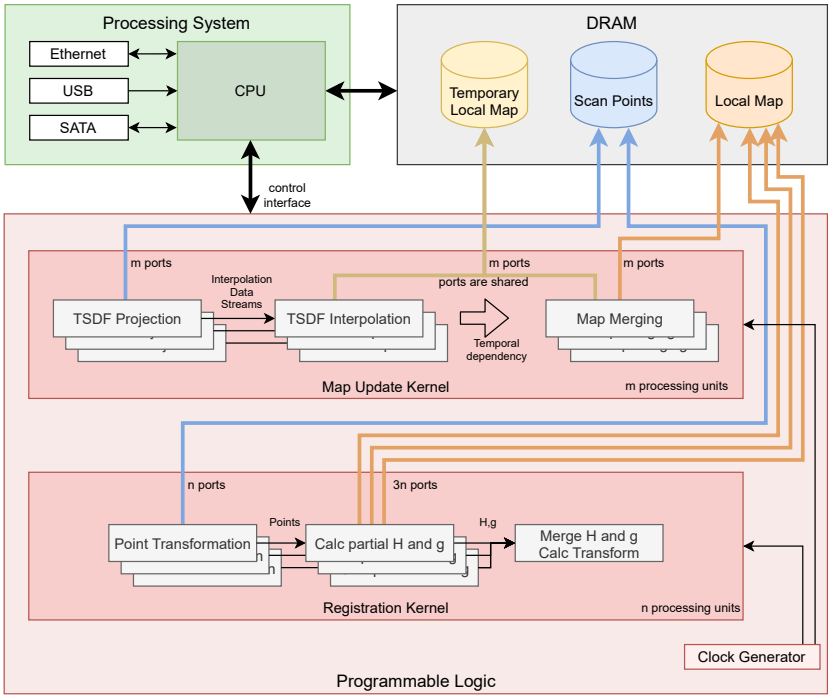

Abb. 3: Ablauf der Registrierung und des Kartenupdates. Der Registrierungskernel und der Map-Update-Kernel laufen parallel. Der Datenaustausch geschieht ober globales DRAM und dedizierte Memory-Ports.

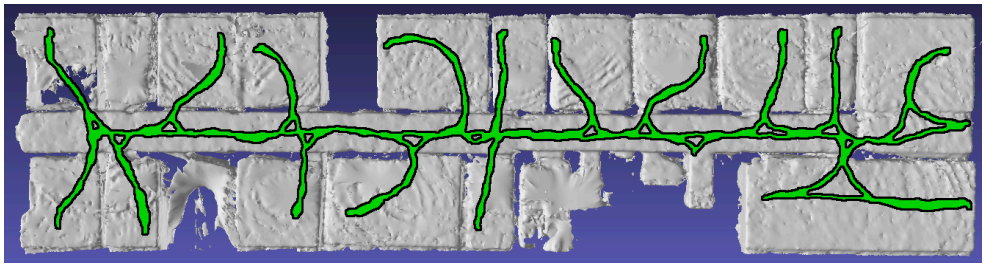

Abb. 4: Ein mit Hilfe des Systems erzeugtes Dreiecksnetz. Die geschätzte Trajektorie ist in grün dargestellt.

4 Evaluation

Wir haben die Genauigkeit und den Energieverbrauch unseres Systems mithilfe eigener Messungen und evaluiert und anhand von Referenzdatensätzen mit anderen Verfahren verglichen. Als Referenzen haben wir den in der Robotik weit verbreiteten KITTI-Datensatz sowie LeGoLOAM (SHAN & ENGLOT, 2018) verwendet. Die Auswertungen erfolgen hier im Wesentlichen qualitativ, da präzise Referenzen nicht zur Verfügung standen. In einem ersten Experiment wurde ein komplettes Stockwerk unseres Institutsgebäudes vermessen. Zur Abschätzung der Qualität der Lokalisierung wurde die Startpose markiert und am Ende der Messung wieder eingenommen. Die Gesamtlänge der Trajektorie betrug 275 m. Die Abweichung nach dem Schleifenschluss am Ursprung betrug dabei 7,5 cm bei einer eingestellten Voxelgröße von 6 cm in der TSDF-Karte. Zum Vergleich mit den KITTY- und LeGoLOAM-Datensätzen wurden die bereitgestellten ROS-Bags mit den aufgenommenen Sensordaten in das System eingespielt. Die Ergebnisse sind in Abbildung 5 zusammengefasst. Dort zeigen sich nur geringe Unterschiede zu den Referenzen. Diese sind vor allem darauf zurückzuführen, dass bei den KITTI-Daten keine Odometrieschätzungen vorhanden waren und die Fahrgeschwindigkeit in beiden Fällen deutlich höher war als in unseren Experimenten, sodass sich ein relativ großer initialer Poseversatz zwischen zwei Messungen ergab.

Der Verlauf der Leistungsaufnahme unseres Systems mit fortschreitender Zeit ist in Abbildung gezeigt. Die mittleren Laufzeiten pro Scan und der dazugehörige Energieverbrauch sind in Tabelle 1 zusammengefasst. Im Verlauf von Abbildung 6 erkennt man deutlich die periodische Integration von neuen Laserscans. Der Map-Update-Prozess wird nicht regelmäßig angestoßen, sondern nur dann, wenn der erkannte Poseversatz einen vorgegebenen Wert erreicht. Man kann deutlich erkennen, dass dieser Prozess deutlich mehr Energie benötigt als das Einfügen einzelner Scans. In Tabelle 1 wurden die mittleren Werte unseres Systems mit einem Minicomputer Intel NUC6iKYK verglichen. Dieser wird z.B. zur Ansteuerung unserer mobilen Roboter verwendet und verfügt über 6 Rechenkerne, 32 GB RAM und NVIDIA GeForce GTX 1050 Grafikkarte. Man kann deutlich sehen, dass auf dieser Plattform die Daten nicht in Echtzeit verarbeitet werden können. . Die verwendeten

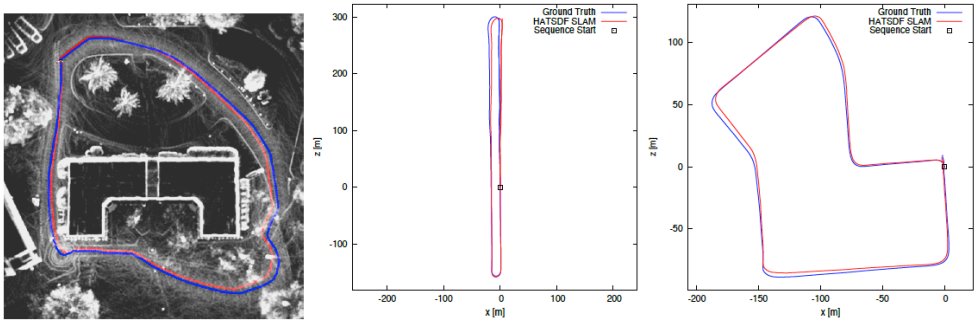

Abb. 5: Vergleich unseres SLAM-Systems mit LeGoLOAM (links) und den KITTI-Datensätzen 6 und 7 (Mitte bzw. rechts).

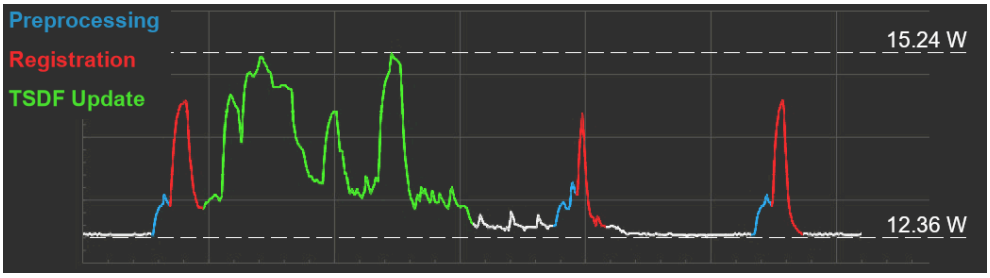

Abb. 6: Leistungsaufnahme der einzelnen Verarbeitungsschritte.

LiDAR-Sensoren arbeiten mit einer Frequenz von 20 Hz, was eine Integrationszeit von unter 50 ms erfordert, um schritthaltend zu sein. Dies wird auf dem Intel NUC nicht annähernd erreicht. Im Mittel benötigt unsere Plattform 28 ms zur Integration neuer Daten, entsprechend wurde mit der vorgelegten Implementation das Ziel, die Daten in Echtzeit verarbeiten zu können, erreicht. Der Energieverbrauch betrug dabei nur 0,52 J pro Scan, verglichen mit 9,40 J auf dem Intel NUC. Unser System arbeitet also ca. 18-mal energieeffizienter als der PC.

5 Zusammenfassung und Ausblick

In diesem Beitrag haben wir ein System vorgestellt, das es erlaubt, Daten von LiDAR-Sensoren auf mobilen Systemen in Echtzeit zu verarbeiten. Das System ist komplett unabhängig und stellt Schnittstellen zum Robot Operating System zur Verfügung, was eine einfache Integration in bestehende Systeme erlaubt. Die vorgelegte Evaluation hat gezeigt, dass der benötigte Energiebedarf in Vergleich zu klassischen CPU-basierten System deutlich reduziert ist. Die Positionierungsgenauigkeit entspricht dabei dem Stand der Forschung im Bereich der mobilen Robotik. Die Möglichkeit, aus der internen TSDF-Repräsentation automatisch Dreiecksnetze erzeugen zu können, eröffnet für mobile Systeme viele neue Forschungsansätze. Da jetzt großräumig Dreiecksnetze von großen Umgebungen in kurzer Zeit erstellt werden können, besteht jetzt die Möglichkeit neue Verfahren zur Lokalisierung

Tabelle 1: Mittlere Laufzeit pro Scan und Energieverbrauch

Plattform	Laufzeit [s]	Leistung [W]	Energie pro Scan [J]
Intel NUC	0,279	34,0	9,40
SLAM Box	0,028	13,8	0,52

und Pfadplanung in solchen Repräsentationen zu entwickeln. Erste Ansätze dazu finden sich z.B. in PÜTZ ET AL. (2019). Lokalisierung in 6D ist vor allen für UAVs interessant, Pfadplanung auf beliebigen Oberflächen wird in allen Bereichen der autonomen Robotik benötigt. Neben der Entwicklung solcher Algorithmen für mobile Roboter sollen in zukünftigen Arbeiten weitere Daten in die erstellten Karten integriert werden. Denkbar ist z.B. die Integration von RGB-Daten, um farblich texturierte Meshes erstellen zu können.

Literatur

Eisoldt, M. & Flottmann, M. & Gaal, J. & Hinderink, S. & Vana, J. & Rothmann, M. & Tassemeier, M. & Wiemann, Th. & Porrmann, M. (2021) *ReconfROS: Running ROS on Reconfigurable SoCs*. 2021 Workshop on Drone Systems Engineering (DroneSE 2021). ACM Digital Library, 2021

Canelhas, D.R. & Stoyanov, T. & Lilienthal, A. J (2013). *SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images*, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013

Izadi, S. & Kim, D. & Hilliges, O. & Molyneaux, D. & Newcombe, R.& Kohli, P. & Fitzgibbon, A (2011). *Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera.* In Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM, 2011

Whelan, T.& Kaess, M. & Fallon, M. & Johannsson, H. & Leonard, J. & McDonald, J. (2012). *Kintinuous: Spatially extended kinectfusion*. MIT Library, 2012

Lorensen, W. E., & Cline, H. E. (1987). *Marching cubes: A high resolution 3D surface construction algorithm*. ACM Siggraph Computer Graphics, 21(4), 163-169.

Wiemann, Th. & Mitschke, I. (2018). *Verteilte Oberflächenrekonstruktion aus 3D-Punktwolken*. avn – Allgemeine Vermessungsnachrichten, 125(3), 53 -62, Wichman Verlag, 2018

Shan, T. & Englot, B. (2018): *LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain*, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018

Pütz, S. & Wiemann, T. & Kleine Piening, M. & Hertzberg, J (2021). *Continuous Shortest Paths Vector Field Navigation on 3D Triangular Meshes for Mobile Robots*. 2021 IEEE International Conference on Robotics and Automation (ICRA 2021), IEEE 2021

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

Name, Vorname
Firma, Ort *oder* Hochschule, Institut
E-Mail-Adresse

Diese Angaben werden im Autorenverzeichnis am Ende des Tagungsbandes aufgeführt.

Wiemann, Thomas, Universität Osnabrück, Institut für Informatik, twiemann@uos.de

Eisoldt, Marc, Universität Osnabrück, Institut für Informatik, meisoldt@uos.de

Flottmann, Marcel, Universität Osnabrück, Institut für Informatik, mflottmann@uos.de

Gaal, Julian, Universität Osnabrück, Institut für Informatik, gjulian@uos.de

Rothmann, Marc, Universität Osnabrück, Institut für Informatik, mrothmann@uos.de

Tassemeier, Marco, Universität Osnabrück, Institut für Informatik, mtassemeier@uos.de

Porrmann, Mario, Universität Osnabrück, Institut für Informatik, mporrmann@uos.de