Echtzeit-Fingertracking in Unity 3D durch 3D Convolutional Neural Networks in einem Multi-Depth-Camera-Setup

Tobias Picker* und Bastian Dewitz*, Christian Geiger*, Frank Steinicke†

* Hochschule Düsseldorf
Münsterstr. 156
40476 Düsseldorf

†Universität Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg

Abstract: In diesem Work-in-Progress-Paper wird ein in Entwicklung befindliches System für Fingertracking in Echtzeit für stationäre virtuelle Umgebungen vorgestellt, das auf der Kombination mehrerer Tiefensensoren und 3D Convolutional Neural Networks basiert und ohne weitere Instrumentierung der Hände auskommt. Die ersten Ergebnisse zur Geschwindigkeit des Systems und Genauigkeit der Rekonstruktion sind vielversprechend und das vorgestellte System wird weiterentwickelt.

Keywords: Handtracking, Machine-Learning, Neuronale Netze, Tiefenkameras

1 Einleitung

Handtracking und handbasierte Schnittstellen sind vielversprechende Konzepte zur Interaktion mit virtuellen Umgebungen und Computersystemen im Allgemeinen. Technisch wurden in den letzten Jahren große Fortschritte bei der Erkennung nicht instrumentierter Hände erzielt, vor allem durch die Nutzung künstlicher neuronaler Netze in bildbasierten Ansätzen. Eine weite Verbreitung von handbasierter Interaktion scheint daher in den nächsten Jahren sehr wahrscheinlich.

In diesem Work-in-Progress-Shortpaper wird ein System vorgestellt, das zur Rekonstruktion von Handposen in Echtzeit mehrere Tiefensensoren einsetzt, die anhand eines kombinierten 3D-Voxel-Volumens durch ein 3D Convolutional Neural Network (3DCNN) die Positionen von Fingergelenken berechnen soll. Hierbei wird von einen Spezialfall von handbasierter Interaktion ausgegangen: Stationäre Systeme, die zur Interaktion an einem vorbereiteten Arbeitsplatz eingesetzt werden können, beispielsweise im chirurgischen oder industriellen Kontext in Verbindung mit Virtual- oder Augmented-Reality-Geräten. In solchen Anwendungsfällen soll das System eine Interaktion mit digitalen Inhalten ohne die Nutzung weiterer Eingabegeräte wie Controller ermöglichen. Durch die Verwendung mehrerer Sensoren in einem festen Aufbau sollen ferner verschiedene Probleme wie Verdeckungen oder die Begrenzung des Interaktionsraums durch das Sichtfeld eines einzelnen Sensors minimiert werden.

Im Folgenden werden Details der Implementierung präsentiert und es wird auf Ergebnisse

einer ersten Machbarkeitsabschätzung eingegangen.

2 Verwandte Arbeiten

Die handbasierte Interaktion mit Computersystemen ist seit den 80er Jahren Bestandteil der Mensch-Computer-Interaktionsforschung. Es wurden verschiedenste Systeme vorgestellt, die zumeist Datenhandschuhe verwenden oder kamerabasiert sind. Insbesondere im Bereich der kamerabasierten Systeme, die nicht instrumentierte Hände erfassen, konnten in den letzten Jahren durch Machine Learning große Fortschritte erzielt werden, sodass Echtzeitfingertracking sowohl über RGB-Kameras (z. B. [ZBV+20, MBS+18, BTG+12]) als auch über Tiefensensoren (z. B. [TST+15, OL17, GLYT17, GCWY18]) möglich ist und für spezielle Anwendungsfälle wie Mid-Air-Gestenerkennung genutzt werden kann. [OL17] nutzt ein 2D Convolutional Neural Network, das auf Basis des Tiefenbilds die 3D-Gelenkpositionen ausgibt. [GLYT16] erweitert diesen Ansatz um zusätzliche Eingabedaten, indem die Tiefendaten auf drei orthogonale Flächen projiziert werden, sodass sich drei unterschiedliche Ansichten ergeben. Aufbauend auf diesem Prinzip wandelt [GLYT17] die Tiefendaten in ein Voxel-Raster um, sodass ein 3D CNN verwendet werden kann. Eine weitere Abwandlung [GCWY18] nutzt ein PointNet, eine spezielle Form des 3D CNNs, das direkt die 3D-Rohdaten der Tiefenkamera in Form einer Punktwolke verarbeiten kann. Dies erspart die Vorverarbeitung und ermöglicht eine effizientere Nutzung der Daten. Gute Erfolge erzielen auch Mali et al. [MAE+20], die das Tiefenbild einer einzelnen Kamera in Voxel konvertieren.

Generell lässt sich feststellen, das viele Ansätze die Rekonstruktion von Mid-Air-Handposen aus einzelnen Kamerabildern fokussieren und hierbei durchschnittliche Fehler von ca. 1 bis 2 cm erhalten [OL17], was in vielen Anwendungsfällen als ausreichend betrachtet werden kann. Multi-Kamerasysteme sind selten vertreten und Spezialfälle wie die Interaktion mit Objekten oder zwischen zwei Händen werden bisher kaum untersucht.

3 Implementierung

Zur Überprüfung des Konzepts wurde ein prototypisches System in Unity 3D, einer im Forschungsbereich der Mensch-Technik-Interaktion beliebten Entwicklungsumgebung, in Kombination mit einem Tensorflow-Machine-Learning-Ansatz entwickelt. Im Vorfeld hatte es

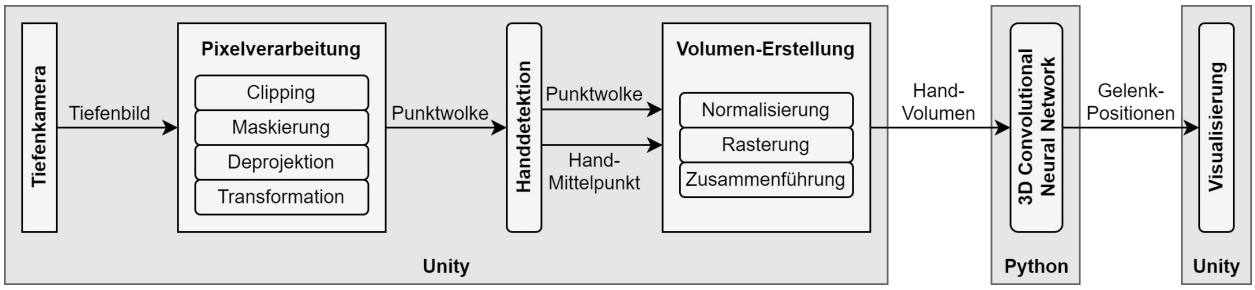
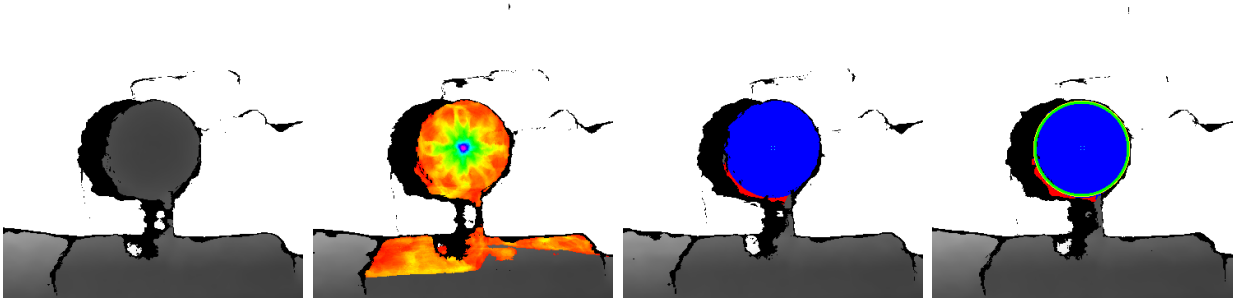


Abbildung 1: Pipeline der prototypischen Anwendung für eine Kamera und eine Hand.



(a) Tiefenbild. (b) Surface Detektor. (c) Region Grow. (d) Regression.

Abbildung 2: Ablauf der Pixelverarbeitung zur Kugelerkennung.

Bedenken gegeben, ob dies eine geeignete Entwicklungsumgebung darstellt, da eine große Anzahl an Berechnungen durchgeführt werden muss und unklar war, ob dies in Echtzeit möglich ist. In dieser Iteration der Entwicklung wurden daher die Eignung von Unity 3D und insbesondere die Nutzbarkeit für Bildverarbeitung und komplexe Berechnungen durch Parallelisierung im *Job System* und Optimierung mit dem *Burst Compiler* gezielt geprüft. Hierdurch sollte eine Abschätzung ermöglicht werden, inwieweit Entwicklungen dieser Art innerhalb von Unity 3D möglich sind, ohne auf weitere, spezialisierte Entwicklungsumgebungen zur Vorverarbeitung zurückgreifen zu müssen. Da Unity 3D zum Zeitpunkt der Erstellung der Arbeit keine flexible und performante Lösung für die Ausführung von Tensorflow-Graphen bietet, wird die Berechnung des neuronalen Netzes ausschließlich in Python ausgeführt. Als Schnittstelle zwischen Unity- und Python-Anwendung wird daher die Bibliothek ZeroMQ genutzt, die auf einem TCP-Sockel basiert, der auf geringe Latenzen und eine hohe Durchsatzrate ausgelegt ist. Ein Aufbau des gesamten Systems ist unter Abb. 1 aufgeführt.

Um die vorgestellte Pipeline ausführen zu können, müssen zuerst in einem Kalibrierungsschritt die räumliche Anordnung der Tiefenkameras bestimmt und eine Maskierung des Hintergrundes durchgeführt werden. Nach dieser Kalibrierung werden die Daten kontinuierlich für das 3DCNN vorverarbeitet und dorthin gestreamt.

3.1 Kalibrierung der Tiefenkameras

Für die Maskierungen als Grundlage der weiteren Berechnungen werden die Rohdaten der einzelnen Tiefenkameras über einen festen Zeitraum gesammelt und gemittelt, sodass das Grundrauschen der Sensoren gefiltert werden kann. Für die Berechnung der Transformationsmatrizen wird ein zusätzlicher Kalibriervorgang nötig. Dazu wird eine Kamera als Master bestimmt, die den Ursprung des gemeinsamen Koordinatensystems festlegt. Die Transformation eines Punktes einer Slave-Kamera S_i in das Master-System M kann mit der Gleichung $T_i \cdot S_i = M$ beschrieben werden, wobei T_i der gesuchten Transformationsmatrix entspricht. Die Ermittlung von T_i wird durch die Anwendung einer Singular-Value-Decomposition auf eine Menge von im Kalibrierungsschritt erzeugten Punktpaaren durchgeführt. Hierdurch kann auf komplexere Algorithmen wie den Iterative-Closest-Point-Algorithmus zur Bestimmung der relativen Transformation zwischen beiden Koordinatensystemen verzichtet werden.

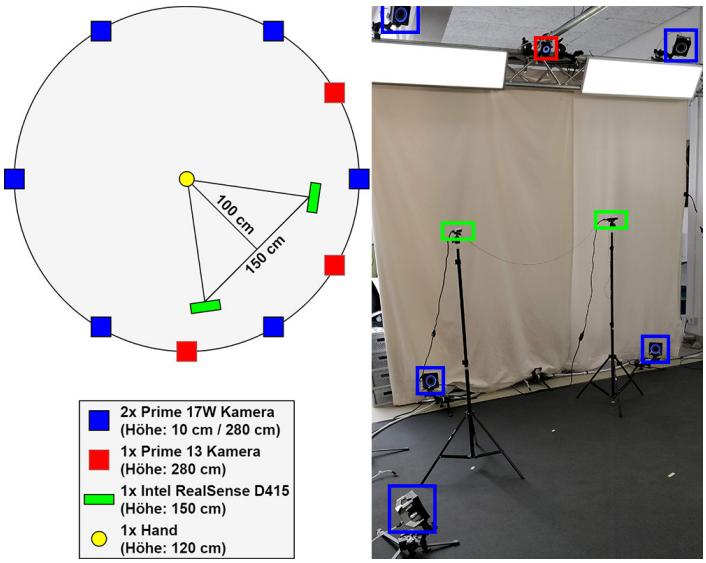


Abbildung 3: Versuchsaufbau und schematischer kreisförmiger Aufbau der Anwendung zur Erfassung von Trackingdaten.

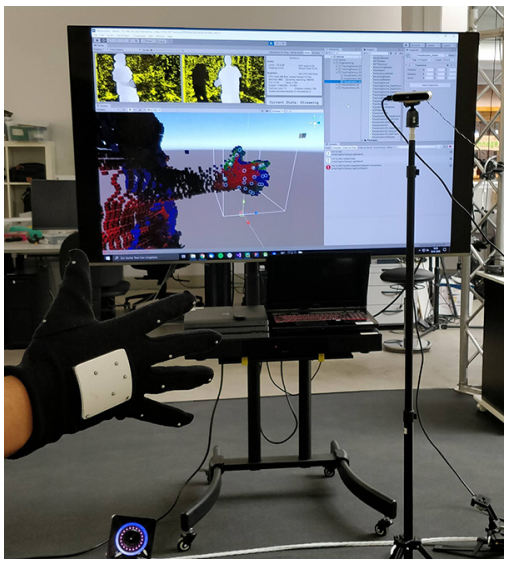


Abbildung 4: Erfassung der Trainingsdaten (Tiefendaten und Referenz-Gelenkpositionen).

Zur Generierung von einander zugeordneten Punktpaaren („Wanding") wird eine Styroporkugel mit bekanntem Radius durch das zu kalibrierende Volumen bewegt, die in den jeweiligen Tiefendaten der Kameras sichtbar ist (Abb. 2a). Anfangs wird jeder Pixel durch einen lokalen Operator auf seine Wahrscheinlichkeit hin, der Mittelpunkt auf einer Kugeloberfläche zu sein, untersucht. Auf vier Achsen (horizontal, vertikal, 2 x diagonal) werden die Pixel innerhalb des Kugelradius überprüft und, sofern die Distanz zwischen gemessener und theoretischer Position auf der Oberfläche einer Kugel mit dem gesuchten Radius innerhalb eines Schwellenwerts liegt, einer Kugel zugeordnet. Bei genügend zugeordneten Pixeln gilt der betrachtete Pixel als potentieller Mittelpunkt einer Kugel (violette Farbe in Abb. 2b). Im nächsten Schritt werden anhand dieser Mittelpunktkandidaten Segmente gebildet (Abb. 2c), denen angrenzende Pixel zugeordnet werden (blau), sofern die gemessene und berechnete Position nicht zu stark von der gesuchten Kugel abweicht (rot). Anhand einer sphärischen Regression können durch die zugeordneten Pixelkoordinaten der exakte Mittelpunkt und der zugehörige Radius bestimmt werden. Der Radius der berechneten Kugel und die Standardabweichung der Regression bestimmen als Konfidenzwerte die finale Kugel (Abb. 2d). Bei einer ausreichenden Menge von Punktpaaren – 15 bis 20 Stück haben sich in diesem System als praxistauglich erwiesen – wird das Wanding beendet.

Anhand eines im Vorhinein definierten kubischen Volumens zur Detektion der Hand werden aus jeder Punktwolke die innen liegenden Punkte bestimmt und durch eine Rasterung in ein Voxel-Raster überführt. Dazu werden alle Punkte, die nach Differenzbildung zur Maske übrig bleiben, anhand der Ausmaße des Volumens normalisiert und mit der gewählten Auflösung auf die für das neuronale Netz benötigte Größe skaliert. Die so entstehenden Koordinaten werden für alle Punktwolken in ein gemeinsames Boolean Array zusammengeführt, das in seinen Dimensionen den definierten Eingangswerten des 3DCNNs entspricht.

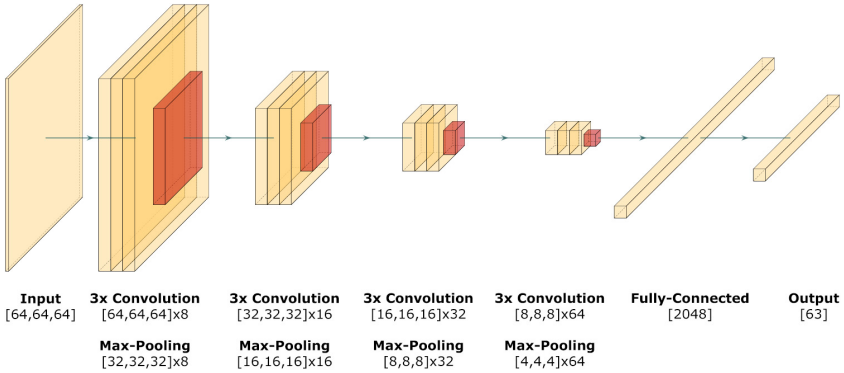


Abbildung 5: Graph des neuronalen Netzes.

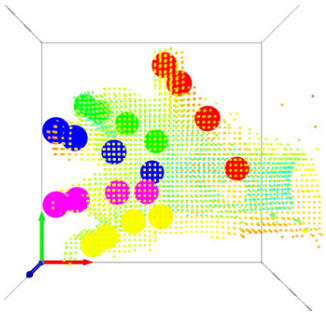


Abbildung 6: Ein Trainingsdatensatz.

3.2 Training des neuronales Netze

Für das Training des 3DCNNs wird die *tf.data*-Pipeline von Tensorflow im *Mixed Precision Modus* genutzt. Als Eingabedaten wird ein vierdimensionaler Boolean Tensor der Form $[64, 64, 64, 1]$ genutzt und für die Ausgabedaten ein eindimensionaler float32-Tensor der Form [63], der den 21 Gelenkpositionen mit jeweils drei Komponenten (x, y, z) entspricht (Abb. 5). Zur automatisierten Bestimmung der besten Hyperparameter wie Anzahl der Convolution-Schichten, Größe des Filterkernels, und Menge an Neuronen in den Fully-Connected-Schichten des 3DCNNs wurde die Bibliothek *Keras Tuner* verwendet.

Für die gleichzeitige Erzeugung der Referenzdaten der Gelenkpositionen und Tiefenbilder der Hände als Trainingsdaten (siehe Abb. 6) wird ein Optitrack-System mit Handschuh genutzt, das Gelenkpositionen mit hoher Genauigkeit und geringer Latenz in Echtzeit liefert (Abb. 4). Die Tiefendaten werden von zwei synchronisierten Intel RealSense D415 mit einem Abstand von 150 *cm* bei einer Höhe von 150 *cm* aufgenommen (Abb. 3). Um die Generalisierung des 3D Convolutional Neural Networks weiter zu verbessern und ein Overfitting zu verhindern, werden die Trainingsdaten zur Laufzeit des Trainings zufällig gemischt und augmentiert. Insgesamt wurden 22 937 Datensätze generiert und zufällig in 70 Prozent Trainings- und jeweils 15 Prozent Validierungs- und Testdatensätze eingeteilt. Die Trainingsdaten bestehen aus kontinuierlichen Handbewegungen der rechten Hand einer einzelnen Person und beinhalten verschiedene Handrotationen, Greif- und Spreiz-Gesten, Bewegungen einzelner sowie die Interaktion mehrerer Finger. Als Grundlage dient eine klassische CNN-Architektur, bestehend aus vier Convolution-Schichten mit jeweils 8, 16, 32 und 64 Filtern der Größe $[3, 3, 3]$, vier Max-Pooling-Schichten mit einem Skalierungsfaktor von $0, 5$ und zwei Fully-Connected-Schichten mit 2048 und 63 Neuronen (siehe Abbildung 5). Zur Aktivierung werden *ReLU*-Funktionen genutzt sowie ein *Adam Optimizer* zum Training, während zur Bewertung der neuronalen Netze eine *Mean-Absolute-Error*-Funktion verwendet wird, die die mittlere durchschnittliche Abweichung der Ausgabewerte des neuronalen Netzes von den Referenzpositionen der Fingergelenke beschreibt.

4 Test

4.1 Performance

Zur Evaluation der Performance der prototypischen Pipeline wurden die Ausführungszeiten der einzelnen Abschnitte gemessen: Das *(1) Abrufen der Tiefendaten* enthält die Anfrage nach neuen Daten des Tiefensensors und die Konvertierung des Rohdatenstroms. Die Tiefendaten werden in der *(2) 2D-Datenverarbeitung* geclippt, maskiert und an die *(3) 3D-Datenverarbeitung* weitergegeben. Hier werden die Daten deprojiziert und in ein gemeinsames Koordinatensystem transformiert. Der Abschnitt der *(4) Rasterung* enthält die Normalisierung und Konvertierung in ein Voxel-Array, das in der *(5) Zusammenführung* mit den Daten der anderen Tiefenkameras zum Hand-Volumen vereint wird. Das *(6) Machine-Learning* beinhaltet die Übertragungen zwischen Unity und Python mit den zugehörigen Datentyp-Umwandlungen sowie die Ausführung des neuronalen Netzes in Tensorflow. Die Messungen wurden auf einem Rechner mit einem Quad-Core-Prozessor Intel Xeon E3-1230v2 mit einer Grundtaktfrequenz von 3,30 GHz, 16 GB Arbeitsspeicher und einer Nvidia Geforce GTX 1070 mit 2432 CUDA Cores, einer Basistaktung von 1607 MHz und 8 GB Arbeitsspeicher durchgeführt.

Pipeline-Schritt	**2x D415**	**3x D415**
(1) Abrufen der Tiefendaten	0,8–2,2 ms	0,8–2,7 ms
(2) Bildverarbeitung (2D)	0,9–2,7 ms	0,8–2,7 ms
(3) Datenverarbeitung (3D)	0,9–2,9 ms	0,9–2,6 ms
(4) Rasterung	0,9–2,8 ms	0,9–2,8 ms
(5) Zusammenführung	10,8–19,1 ms	16,5–32,7 ms
(6) Machine-Learning	27,6–50,0 ms	28,6–36,4 ms
Insgesamt	41,6–79,7 ms	48,5–79,9 ms

Tabelle 1: Minimale und maximale Berechnungszeiten für zwei und für drei Tiefenkameras (RealSense D415).

4.2 Fingertracking

Bei der subjektiven Beobachtung der vom neuronalen Netz gelieferten Gelenkpositionen im Echtzeiteinsatz mit unbekannten Daten konnte festgestellt werden, dass in einigen Fällen die Gelenkpositionen sehr gut anhand des 3D-Volumens rekonstruiert werden konnten, andere Fälle erzeugten jedoch keine sinnvollen Ergebnisse (Abb. 7). Hierbei scheint es sich hierbei um eine Overfitting zu handeln, da Grundhaltungen der Hand, wie sie auch in den Traingsdaten auftauchten, generell gut erkannt wurden, seltene Bewegungen in den Datensätzen wie die Opposition von Fingern jedoch größere Schwierigkeiten bereiteten. Die Ergebnisse entsprechen der Erwartung, da die Anzahl an Trainingsdaten in dieser Iteration noch sehr niedrig ist und vor allem die Machbarkeit der Pipeline analysiert werden sollte.

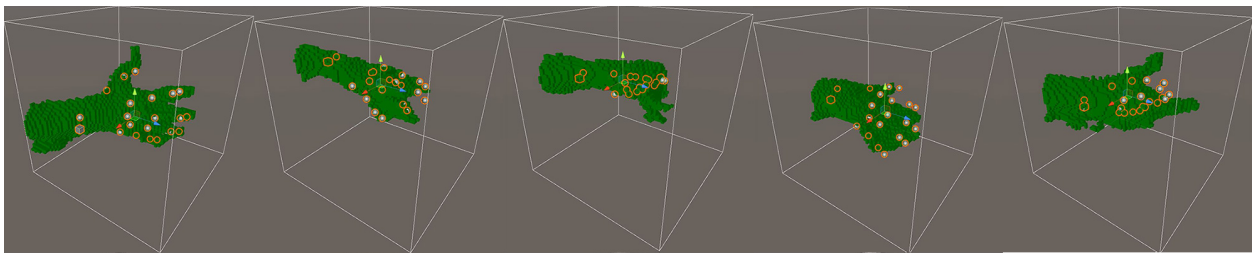


Abbildung 7: Ausgegebene Gelenkpositionen des neuronalen Netzes für eingegebene Voxel.

5 Diskussion

Die Messungen zeigen, dass die angestrebte Echtzeitfähgkeit des Systems unter Einschränkungen erreicht werden konnte. Die gesamte Laufzeit der Pipeline von ca. 40 bis 80 ms ist ausreichend für viele Interaktionsfälle und die Darstellung von Händen. Eine weitere räumliche Skalierbarkeit durch das Hinzufügen weiterer Tiefenkameras scheint möglich und ist vor allem vom verwendeten System und weiteren Optimierungen abhängig. Weiter ist aus Tab. 1 ersichtlich, dass die Verwendung des Job-Systems und des Burst-Compilers die Parallelisierung wichtiger Arbeitsschritte in Unity erlaubt (im Vergleich zur vorherigen Implementierung ohne Parallelisierung der Arbeitsschritte unter Verwendung identischer Algorithmen konnte eine Beschleunigung der Vorverarbeitung um den Faktor 20 erzielt werden) und dass die Anbindung und Berechnungen der externen Machine-Learning-Umgebung den Bottleneck der Anwendung darstellen. Das neuronale Netz in seiner aktuellen Form zeigt, dass das System grundsätzlich dazu in der Lage ist, anhand der 3D-Voxel die Gelenkpositionen in Echtzeit zu bestimmen, auch wenn in diesem Fall ein Overfitting nicht ausgeschlossen werden kann. Eine Änderung der zugrunde liegenden Struktur, beispielsweise durch Residual Neural Networks oder komplexere Netzwerke wie in [MAE+20] präsentiert, könnte die Genauigkeit der Erkennung jedoch weiter steigern.

6 Zusammenfassung und Ausblick

Insgesamt erscheint der gewählte Ansatz für ein Handtrackingsystem vielversprechend und die grundsätzliche Machbarkeit konnte erfolgreich demonstriert werden. Eine Weiterentwicklung des Systems zur Performance-Optimierung sowie die Generierung zusätzlicher Trainingsdaten werden in der nächsten Entwicklungsstufe angestrebt. Insbesondere sind ein objektiver Vergleich zu anderen Handtrackingsystemen sowie eine Fehleranalyse geplant, um eine Aussage über die Vorteile eines Multi-Tiefenkamera-Setups treffen zu können.

Anmerkungen

Diese Forschungsarbeit wurde vom Bundesministerium für Bildung und Forschung (BMBF) im Projekt „Interaktive körpernahe Produktionstechnik 4.0 – iKPT4.0"(Projektnummer 13FH022IX6) gefördert.

Literatur

[BTG+12] Luca Ballan, Aparna Taneja, Jürgen Gall, Luc Van Gool, and Marc Pollefeys. Motion capture of hands in action using discriminative salient points. In *European Conference on Computer Vision*, pages 640–653. Springer, 2012.

[GCWY18] Liuhao Ge, Yujun Cai, Junwu Weng, and Junsong Yuan. Hand pointnet: 3d hand pose estimation using point sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8417–8426, 2018.

[GLYT16] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. Robust 3d hand pose estimation in single depth images: from single-view cnn to multi-view cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3593–3601, 2016.

[GLYT17] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. 3d convolutional neural networks for efficient and robust hand pose estimation from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1991–2000, 2017.

[MAE+20] Jameel Malik, Ibrahim Abdelaziz, Ahmed Elhayek, Soshi Shimada, Sk Aziz Ali, Vladislav Golyanik, Christian Theobalt, and Didier Stricker. Handvoxnet: Deep voxel-based network for 3d hand shape and pose estimation from a single depth map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7113–7122, 2020.

[MBS+18] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Ganerated hands for real-time 3d hand tracking from monocular rgb. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 49–59, 2018.

[OL17] Markus Oberweger and Vincent Lepetit. Deepprior++: Improving fast and accurate 3d hand pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 585–594, 2017.

[TST+15] Andrea Tagliasacchi, Matthias Schröder, Anastasia Tkach, Sofien Bouaziz, Mario Botsch, and Mark Pauly. Robust articulated-icp for real-time hand tracking. In *Computer Graphics Forum*, volume 34, pages 101–114. Wiley Online Library, 2015.

[ZBV+20] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.