

Praktikum 2 Computer Vision: Laptopsteuerung mit Handgesten

Hochschule Darmstadt, SS 2024

Prof. Dr. Elke Hergenröther

Autoren: Anh Vu (1129004), Hannah Schult (1129010)

ABSTRACT

Im vorliegenden Paper werden zwei Ansätze zur robusten und fehlerarmen Echtzeiterkennung von Handgesten untersucht, um die Mensch-Computer-Interaktion zu verbessern. Ziel ist es, zwei Verfahren zu entwickeln, welche menschliche Handgesten mittels Kameras effizient erkennen. Dabei wird zunächst ein eigenständig entwickelter Ansatz (Ansatz A) vorgestellt. Darauf folgt die Implementierung eines vorgefertigten Frameworks MediaPipe (Ansatz B). Durch den Vergleich dieser beiden Methoden sollen die Stärken und Schwächen jedes Ansatzes aufgezeigt und Ansätze zur weiteren Optimierung der Handgestenerkennung erörtert werden. Zum Schluss wird deutlich, dass der Einsatz von dem vorgefertigten Framework sinnvoll ist, da dieses den selbst entwickelten Ansatz in Robustheit und Genauigkeit übertrifft.

1 EINLEITUNG

Die Interaktion zwischen Mensch und Computer hat sich in den letzten Jahren rasant entwickelt und ist aus unserem Alltag nicht mehr wegzudenken. Insbesondere die Handgestenerkennung spielt eine wichtige Rolle, um intuitive Schnittstellen zu ermöglichen, die natürliche Bewegungen in computergestützte Befehle umsetzen können. Aus diesem Grund hat sich die Mensch-Computer-Interaktion (engl. human-computer interaction (HCI)) in den letzten Jahren zu einem wichtigen Forschungsgebiet entwickelt (vgl. [1], S. 1).

Die Handgestenerkennung wurde beispielsweise bereits durch tragbare Sensoren wie Datenhandschuhe realisiert, die physische Bewegungen erfassen. Diese Lösungen liefern präzise Ergebnisse, sind jedoch oft kostenintensiv und unhandlich in der Bedienung (vgl. [2]).

Eine zunehmend populäre und kostengünstigere Methode ist die Handgestenerkennung mittels Computer Vision, bei der Kameradaten und maschinelle Lernalgorithmen zur Erkennung und Interpretation von Handgesten genutzt werden. Jedoch muss in diesem Bereich weiter geforscht werden, um die Erkenntnisse in der Handgestenerkennung voranzutreiben. Daher werden in der vorliegenden Arbeit zwei Ansätze zur Handgestenerkennung untersucht. Zuerst wird

ein eigenständig entwickelter Ansatz entwickelt und auf Herausforderungen geprüft. Anschließend werden diese Herausforderungen mit einem vorgefertigten Framework (MediaPipe) verglichen.

Ziel dieser Arbeit ist es somit diese zwei Ansätze zur Handgestenerkennung zu vergleichen und Herausforderungen im Bereich der Handgestenerkennung aufzuzeigen.

2 GRUNDLAGEN / RELATED WORK

Ein bekanntes Framework im Bereich Computer Vision ist MediaPipe, ein von Google entwickeltes Open-Source-Framework. Es ermöglicht Entwicklern, komplexe Verarbeitungspipelines für visuelle Daten zu erstellen. MediaPipe bietet vorgefertigte Komponenten zur Echtzeitanalyse von Handgesten und anderen visuellen Informationen. Es nutzt ein modulares Kalkulatoren-System, das Entwicklern erlaubt, Anwendungen schnell zu prototypisieren und systematisch zu verbessern. Zudem ist es plattformübergreifend und kann auf mobilen sowie stationären Geräten eingesetzt werden. MediaPipe wurde erstmals im Paper „MediaPipe: A Framework for Building Perception Pipelines“ (vgl. [3]) vorgestellt. Das Paper wurde auf der CVPR (Conference on Computer Vision and Pattern Recognition) im Jahr 2020 präsentiert.

Zusätzlich gibt es zahlreiche wissenschaftliche Arbeiten, die auf ähnliche Techniken zur Handgestenerkennung setzen. Oudah et al. (2020) bieten einen umfassenden Überblick über verschiedene Ansätze und heben die Rolle maschinellen Lernens hervor (vgl. [2]). Just (2006) zeigte bereits früh die Möglichkeiten auf, Handgesten als Eingabemethode in der HCI zu nutzen (vgl. [1]). Shefali Parihar (2023) verglich mehrere Deep-Learning-Modelle zur Handgestenerkennung und unterstrich die Bedeutung von gut strukturierten Datensätzen und leistungsfähigen Algorithmen (vgl. [4]).

Die Erarbeitung und Analyse von Herausforderungen spielt ebenfalls eine zentrale Rolle in der Forschung, da sie Einblicke in die Schwächen und Grenzen aktueller Technologien und Methoden liefert. Das Verständnis dieser Herausforderungen ist entscheidend, um bestehende Systeme zu verbessern und neue, innovative Ansätze zu entwickeln.

3 KONZEPT

Diese Arbeit nähert sich der Handgestenerkennung in zwei verschiedenen Ansätzen an. Im ersten Ansatz (Ansatz A) wird die Handgestenerkennung vollständig selbst entwickelt und implementiert.

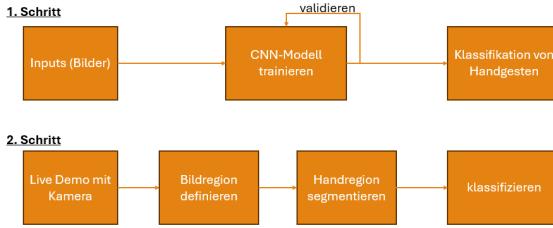


Abbildung 1: Konzept Ansatz A

Wie in Abbildung 1 beschrieben ist, wurde ein Convolutional-Neural-Network (CNN)-Modell mit einem geeigneten Datensatz trainiert. Das trainierte Modell wird anschließend genutzt, um die erlernten Handgesten in Echtzeit innerhalb einer vordefinierten Bildregion der Videoaufnahme der Laptopkamera zu erkennen.

Im zweiten Ansatz (Ansatz B) wird das vorgefertigte MediaPipe-Framework zur Realisierung der Handgestenerkennung verwendet (siehe Abbildung 2).



Abbildung 2: Konzept Ansatz B

Dieser Ansatz nutzt die in MediaPipe integrierten Algorithmen zur schnellen und effizienten Erkennung von Handgesten. Somit findet kein eigener Trainingsprozess statt. Es muss lediglich das MediaPipe-Hand-Modul initialisiert werden, um alle Funktionalitäten von MediaPipe nutzen zu können. Anschließend erfolgt der Zugriff auf die Laptopkamera, wobei das Videobild manuell vorverarbeitet wird. Außerdem wird die Python-Bibliothek PyAutoGUI genutzt, um bestimmte Handgesten mit GUI-Aufgaben (Graphical User Interface) zu verknüpfen. Sobald eine festgelegte Handgeste erkannt wird, erfolgt automatisch die Ausführung der entsprechenden Aufgabe.

Das Ziel von Ansatz A ist es, alle Schritte eigenständig zu entwickeln sowie zu implementieren. Somit wird die ganze Anwendung selbst aufgebaut. Anschließend soll der Ansatz B demonstrieren, wie eine optimal implementierte Handgestensteuerung aussehen kann. Ansatz B basiert dabei auf einem leistungsfähigeren Modell, welches mit mehr Ressourcen und einer längeren Trainingsphase entwickelt wurde. Dieser Vergleich ermöglicht es, die Unterschiede in der Effizienz und Benutzerfreundlichkeit zwischen einem selbst entwickelten System und einem vorgefertigten, professionell trainierten Modell aufzuzeigen.

4 UMSETZUNG

Im ersten Ansatz wird die Handgestenerkennung durch ein CNN-Modell realisiert. Dafür wird zuerst ein passender Datensatz benötigt. Auf Kaggle existiert solch ein frei verfügbarer Datensatz [5] mit 10.321 Bildern, welche in Schwarz-Weißer Segmentierung vorliegen. Er ist geeignet, da sowohl von der rechten als auch der linken Hand jeweils Bilder aus verschiedenen Winkeln und in verschiedenen Positionen in hoher Anzahl vorliegen. Zudem ist der Datensatz bereits gelabelt. In dieser Arbeit wurden sechs Arten von Handgesten berücksichtigt:

- „blank“ (keine Hand im definierten Bildbereich),
- „fist“,
- „five“,
- „ok“,
- „thumbsdown“,
- „thumbsup“ (siehe Abbildung 3).

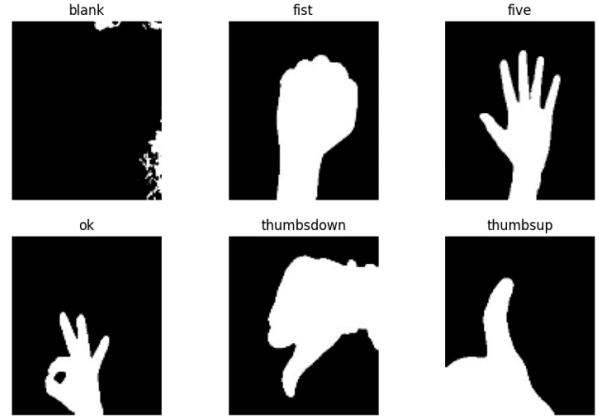


Abbildung 3: Handgesten aus dem Kaggle-Datensatz

Dabei werden insgesamt 60% der Bilder für das Training, 20% der Bilder für die Validierung und 20% der Bilder als Testdaten genutzt.

Mit diesem Datensatz wurde dann das CNN trainiert. In Abbildung 4 ist die Struktur des CNN-Modells dargestellt.

Layer (type)	output shape	Param #
conv2d (Conv2D)	(None, 224, 224, 32)	320
batch_normalization (Batch Normalization)	(None, 224, 224, 32)	128
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 64)	18496
batch_normalization_1 (Batch Normalization)	(None, 112, 112, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 64)	0
flatten (Flatten)	(None, 200704)	0
dense (Dense)	(None, 128)	25690240
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 6)	774

Abbildung 4: CNN-Modell zur Erkennung von Handgesten

Hierbei werden zwei Faltungsschichten genutzt, bevor die Fully-Connected-Layer kommen. Die Eingangsschicht verwendet Convolutional Layer, um Merkmale aus den Bildern zu extrahieren. Batch Normalization normalisiert die Werte der Gewichte aller Schichten während des Trainings, was einen gleichmäßigen Fluss des Gradienten (Fehlers) durch das Netz ermöglicht. Nach jeder Conv2D-Schicht wird Max Pooling eingesetzt, um die räumliche Dimension der Feature Maps zu reduzieren und die Invarianz gegenüber Skalierungs- und Translationsvarianzen zu erhöhen.

Die Convolutional Layers extrahieren Merkmale aus den Bildern, während die Fully Connected Layers die extrahierten Merkmale klassifizieren. Dropout wird verwendet, um das Überanpassen an die Trainingsdaten zu verringern, indem während des Trainings zufällig Neuronen ausgeschaltet werden. Zudem wird Early Stopping angewendet, um Overfitting zu vermeiden.

Da die Bilder im Datensatz in segmentierter Form vorliegen, muss die Anwendung ebenso segmentiere Bilder erzeugen. Deswegen wird im Kameraausschnitt ein Bildbereich definiert, in dem sich die Hand befinden soll. Dieser Bildausschnitt wird dann binarisiert und auf dem binarisierten Bild wird die Handgeste mittels des CNN-Modells klassifiziert. So passt sich die Anwendung den Trainingsdaten an.

Außerdem werden alle weiteren Rahmenbedingungen möglichst einfach gehalten, um eine höhere Robustheit des selbst trainierten Modells zu erreichen. Es wird ein hoher Kontrast zwischen der Hand und einem einfarbigen Hintergrund vorausgesetzt (bspw. die Hand vor einer weißen Wand), keine Bereiche der Hand dürfen verdeckt werden und die Hand muss sich in dem definierten Bereich befinden. Diese Bedingungen sind notwendig, damit die Anwendung mit dem selbst trainierten Modell funktioniert.

Im zweiten Ansatz wird ein vortrainiertes Modell (MediaPipe) verwendet, um die Leistung und Robustheit der Handgestenerkennung zu verbessern. MediaPipe definiert 21 key points der Hand. Mit diesen Punkten können verschiedene Handgesten definiert werden (siehe Abbildung 5).

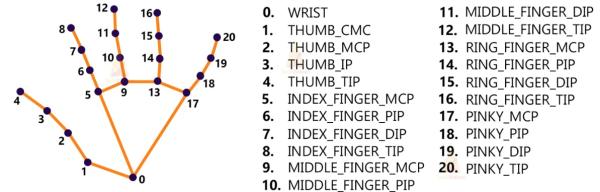


Abbildung 5: MediaPipe Hand-Modul

Die Handgesten werden definiert, indem einzelne Fingerzustände geprüft werden. Beispielsweise wird geprüft, ob der Zeigefinger gestreckt ist, indem man schaut ob sich der Zeigefinger-Tip (= die Spitze des Zeigefingers) über dem Zeigefinger-Dip (= das obere erste Gelenk des Zeigefingers) befindet. Diesen Zustand speichert man dann als „index_finger_up“ ab. So können mehrere Fingerzustände definiert werden. Mit mehreren Fingerzuständen kann anschließend eine Handgeste definiert werden: Um also ein „Peace-Zeichen“ zu erkennen kann abgefragt werden, ob der Zeigefinger und der Mittelfinger gestreckt sind. Mit der Bibliothek pyautogui kann dann eine Aktion gestartet werden, sobald eine Handgeste erkannt wird. Beispielsweise: „Wenn ein Peace-Zeichen (der Hand) erkannt wird, dann vergrößere das Bild.“. PyAutoGUI wird hierbei genutzt, um das Betätigen der ctrl und + Tasten einer Tastatur zu imitieren, was die Tastenkombination für das Bildvergrößern ist.

Somit können Tastenkombinationen mit Handgesten aktiviert werden und der Laptop kann auf diese Weise gesteuert werden, ohne ihn physisch zu nutzen.

Nachdem beide Ansätze entwickelt bzw. implementiert wurden, wird Ansatz A auf Robustheit und Genauigkeit in einem Live-Test geprüft. Die Herausforderungen, die dabei auftreten, werden anschließend auch bei Ansatz B geprüft, um die Ansätze so zu vergleichen.

5 ERGEBNISSE

Die Implementierung und Umsetzung beider Ansätze verursachten keine größeren Probleme. Zu Beginn war die Verknüpfung mit der Laptopkamera falsch, sodass das Kamerabild spiegelverkehrt war. Dies erschwerte zu Beginn die Live-Tests. Jedoch konnte das Kamerabild gedreht werden, sodass die Nutzung vereinfacht wurde.

In Abbildung 6 wird deutlich, dass das CNN-Modell (aus Ansatz A) eine hohe Genauigkeit in der Handgestenklassifizierung auf den Trainings- und Validation-Bildern des Datensatzes aufweist.

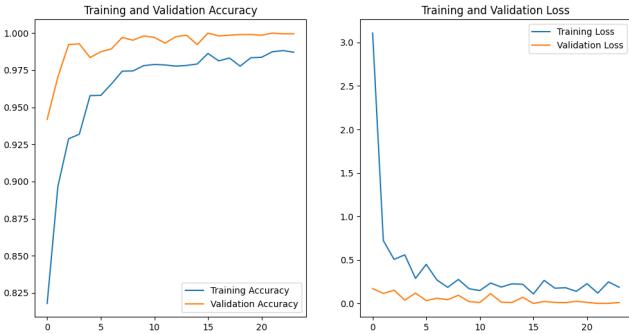


Abbildung 6: Evaluation des CNN-Modells an den Trainings- und Validierungsdaten

Diese hohe Genauigkeit war im Live-Test nicht vorhanden und es wurden weitere Herausforderungen erarbeitet.

Eine wesentliche Herausforderung bei Ansatz A waren die Lichtbedingungen und der Hintergrund. Trotz vereinfachter Rahmenbedingungen bleibt das Modell stark von den Umgebungslichtverhältnissen und dem Hintergrund abhängig, was zu inkonsistenten Erkennungsergebnissen führte. So funktioniert dieses Verfahren manchmal präzise, zeigt jedoch in anderen Situationen eine verringerte Genauigkeit.

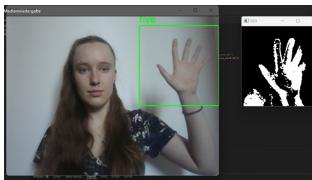


Abbildung 7: Schattenwurf an der Wand

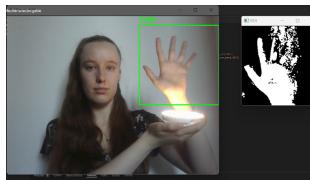


Abbildung 8: Veränderung der Lichtverhältnisse

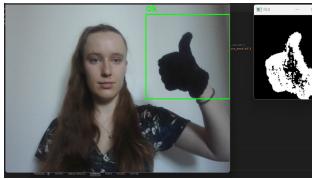


Abbildung 9: Erhöhung des Kontrastes durch schwarze Handschuhe

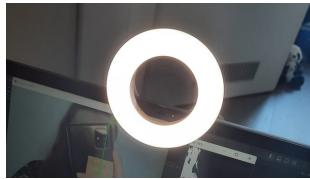


Abbildung 10: Veränderung der Lichtverhältnisse durch Ringlicht am Laptop

In Abbildung 7 ist ersichtlich, dass der Schattenwurf an der Wand die Qualität des segmentierten Bildes beeinträchtigt. Es wurde versucht die Lichtverhältnisse durch den Einsatz eines Ringlichts am Laptop sowie durch manuelle Beleuchtung zu optimieren (Abbildung 8, Abbildung 10). Allerdings verbesserte sich das segmentierte Bild dadurch ebenfalls nicht. Auch der Einsatz dunkler Handschuhe zur Kontrasterhöhung führte zu keiner signifikanten Verbesserung der Bildsegmentierung (Abbildung 9).

Dieses Bildrauschen in den segmentierten Aufnahmen

führte mehrmals zu Fehlklassifikationen während der Live-Tests. Aufgrund der ungenauen Klassifikationen wurde darauf verzichtet die Klassifikation einer Handgeste mit der Steuerung einer Aktion am Laptop zu verknüpfen. Eine mögliche Lösungsstrategie für die ungenauen Klassifikationen in der zukünftigen Forschung könnte darin bestehen, das CNN-Modell nicht mit segmentierten Bildern, sondern mit unbearbeiteten Fotoaufnahmen zu trainieren. Dieser Ansatz könnte die Robustheit des Systems gegenüber variierenden Lichtverhältnissen und Hintergründen erhöhen und somit die Erkennungsgenauigkeit verbessern. In diesem Ansatz wurde sich aber für die segmentierten Bilder entschieden, um die Komplexität der Implementierung einer Objekterkennung zu vermeiden.

Die genannten Herausforderungen wurden anschließend im Ansatz B überprüft. Dabei war festzustellen, dass diese Herausforderungen kein Problem im Ansatz B darstellen, da das MediaPipe Modell sehr robust und fehlerfrei im Live-Test performt.

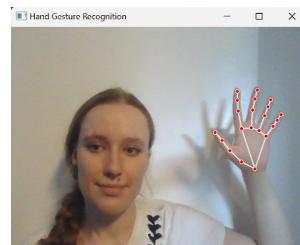


Abbildung 11: Schattenwurf an der Wand

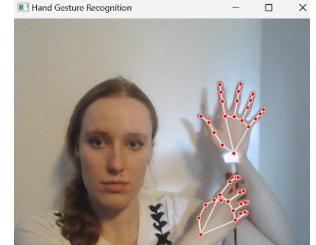


Abbildung 12: Veränderung der Lichtverhältnisse

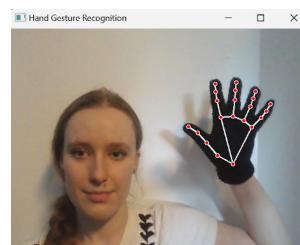


Abbildung 13: Erhöhung des Kontrastes durch schwarze Handschuhe



Abbildung 14: Verdecken von Bereichen der Handfläche

In Abbildung 11 ist erkennbar, dass der Schattenwurf zu keinen Problemen mit der Handerkennung führt. Auch bei veränderten Lichtbedingungen (Abbildung 12), beim Tragen des Handschuhs (Abbildung 13) oder wenn Teile der Hand verdeckt werden (Abbildung 14) funktioniert die Handerkennung sehr robust und fehlerfrei. Somit ist der Ansatz B deutlich robuster und zuverlässiger unter verschiedenen Umgebungsbedingungen im Vergleich zu Ansatz A.

Die größte Herausforderung in Ansatz B war die präzise Steuerung von Aktionen am Laptop durch die Handgesten. Zu Beginn wurde immer wenn das Peace-Zeichen zu sehen

war, ein endloses Zoomen ausgelöst. Dieses hielt an, solange die Geste erkannt wurde. Dadurch entstand die Situation, dass Aktionen zwar ausgeführt wurden, allerdings war eine exakte Steuerung des Laptops nicht möglich. Um dieses Problem zu beheben, wurde die Variable `last_action` eingeführt. Diese Variable bewirkt, dass Aktionen nun bei Erkennung der Handgeste genau einmal durchgeführt werden und die Handgeste in der Variable abgespeichert wird. Die Aktion wird erst dann erneut durchgeführt, wenn die aktuelle Handgeste ungleich jener Handgeste ist, die in der Variable abgespeichert wurde. Auf diese Weise kann der Laptop gesteuert werden.

Nachdem diese Herausforderung gelöst wurde, erzielt die Anwendung aus Ansatz B fehlerfreie Ergebnisse. Die definierten Gesten wurden immer direkt korrekt klassifiziert und die dazugehörigen Aktionen wurden sofort ausgeführt.

Werden nun Ansatz A (komplett selbst entwickelt) und Ansatz B (Nutzung von MediaPipe) verglichen, fällt auf, dass Ansatz B alle Probleme von Ansatz A gelöst hat und fehlerfrei funktioniert.

6 ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurden zwei unterschiedliche Ansätze zur Handgestenerkennung untersucht und miteinander verglichen. Ansatz A basiert auf einem selbst entwickelten Convolutional Neural Network (CNN), das mithilfe eines spezifischen Datensatzes trainiert wurde. Ansatz B nutzt das vorgefertigte MediaPipe-Framework, um eine effizientere und robustere Erkennung von Handgesten zu gewährleisten.

Die Ergebnisse zeigen, dass der selbst entwickelte Ansatz A in einer kontrollierten Umgebung gute Resultate erzielen kann. Dennoch war die Erkennung in Echtzeit aufgrund der hohen Abhängigkeit von Lichtbedingungen und Hintergründen inkonsistent. Diese Schwächen unterstreichen die Herausforderungen bei der Entwicklung eigenständiger Gestenerkennungssysteme, insbesondere im Hinblick auf die Robustheit und die Anpassungsfähigkeit an unterschiedliche Umgebungen.

Im Gegensatz dazu konnte Ansatz B, der auf einem vortrainierten Modell basiert, durchweg stabile und verlässliche Ergebnisse liefern. Die Kombination von MediaPipe und der Python-Bibliothek PyAutoGUI ermöglichte eine präzise Erkennung und unmittelbare Ausführung von Gestenbefehlen, was die Benutzerfreundlichkeit deutlich erhöhte.

Der Vergleich beider Ansätze verdeutlicht die Vorteile der Nutzung vortrainierter Modelle in der Praxis. Insbesondere für Anwendungen, die eine hohe Zuverlässigkeit und schnelle Implementierung erfordern. Die höhere Fehlerquote bei dem selbst entwickelten Ansatz zeigt, dass es immer in Erwägung gezogen werden sollte, vorhandene Ergebnisse und Frameworks zu integrieren. So können die besten Erkenntnisse und maßgeschneiderte Lösungen entwickelt werden.

Für zukünftige Arbeiten könnten die gewonnenen Erkenntnisse zur Verbesserung der Robustheit des selbst entwickelten Modells genutzt werden. Eine mögliche Weiterent-

wicklung könnte darin bestehen, das CNN-Modell mit unsegmentierten und variantenreicheren Trainingsdaten zu trainieren, um die Abhängigkeit von Lichtverhältnissen und Hintergrundbedingungen zu reduzieren. Darüber hinaus könnten fortschrittlichere Bildvorverarbeitungsmethoden und Algorithmen zur Rauschunterdrückung eingesetzt werden, um die Erkennungsgenauigkeit in Echtzeit weiter zu erhöhen.

Zudem sollten zukünftig die Integration von MediaPipe mit anderen maschinellen Lerntechniken untersucht werden, um komplexere und vielfältigere Handgesten zu erkennen. Dies könnte insbesondere für Anwendungen in der virtuellen Realität, im Gaming oder in der assistiven Technologie von Vorteil sein, da in diesem Gebiet natürliche und flüssige Mensch-Computer-Interaktion von entscheidender Bedeutung ist.

Insgesamt zeigt diese Arbeit das Potenzial und die Herausforderungen der Handgestenerkennung auf und bietet eine solide Grundlage für weiterführende Forschungsarbeiten in diesem dynamischen und praxisrelevanten Feld.

LITERATUR

- [1] A. Just, “Two-Handed Gestures for Human-Computer Interaction,” *IDIAP Research Institute*, p. 1, 1 2006. [Online]. Available: <http://publications.idiap.ch/downloads/reports/2006/just-idiap-rr-06-73.pdf>
- [2] M. Oudah, A. Al-Naji, and J. Chahl, “Hand gesture recognition based on computer vision: A review of techniques,” *Journal of Imaging*, vol. 6, no. 8, 2020. [Online]. Available: <https://www.mdpi.com/2313-433X/6/8/73>
- [3] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee *et al.*, “Mediapipe: A framework for building perception pipelines,” *arXiv preprint arXiv:1906.08172*, 2019.
- [4] N. S. Shefali Parihar and P. Thakore, “Hand gesture recognition: a review,” pp. 471–483. [Online]. Available: https://doi.org/10.1007/978-981-99-3611-3_39
- [5] A. Sharma, “hand_gesture_recog_dataset,” 3 2020. [Online]. Available: <https://www.kaggle.com/datasets/sarjit07/hand-gesture-recog-dataset/data>