I believe I can fly V2.0 – Detektion und Klassifikation von NUI-Flugbefehlen anhand einzelner Farbkamera-Bilder

Studienarbeit

des Studiengangs Informatik
Studienrichtung Informationstechnik/Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Henri Kohlberg

20.05.2019

Matrikelnummer	6214814
Kurs	TINF16B1
Ausbildungsfirma	SAP SE, Walldorf
Betreuer	Prof. Dr. Marcus Strand
Bearbeitungszeitraum	01.10.2018 – 20.05.2019

Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29.09.2015)

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema „I believe I can fly V2.0 – Detektion und Klassifikation von NUI-Flugbefehlen anhand einzelner Farbkamera-Bilder“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum Unterschrift

Vorwort

Der Traum vom Fliegen ist genauso alt wie die Menschheit an sich. Die Welt einmal aus Sicht eines Vogels zu erleben ist das Ziel vieler Menschen. Doch mit der heutigen Technik nähern wir uns diesem Ziel Tag für Tag. Drohnen erobern seit einigen Jahren den freien Luftraum und können mittlerweile von jedermann ohne spezielle Ausbildung oder Lizenz gekauft werden. Auch Techniken der virtuellen Realität sind am Markt angekommen und entwickeln sich nach und nach zu einer ausgereiften Technik. Durch das Aufsetzen einer Videobrille wird der Nutzer in völlig neue Welten versetzt ohne sich auch nur von der Stelle zu bewegen. Werden beide Techniken miteinander kombiniert, ist der Traum von Fliegen schon in greifbarer Nähe.

Diese Studienarbeit befasst sich mit der Entwicklung eines Systems zur Steuerung einer Drohne durch NUI-Flugbefehle auf Basis einer einfachen 2D-Webcam und ist Teil einer Reihe von weiteren Studienarbeiten mit dem Titel „I believe I can fly“. Durch die Ausführung von definierten körperlichen Gesten ist ein Benutzer in der Lage, eine Drohne in beliebiger simulierter Umgebung fliegen zu lassen und das Bild einer an der Drohne befestigten Kamera auf seiner Videobrille zu sehen. Durch die passende Wahl der Gesten und das Anzeigen des Bildes der Kamera entsteht bei Benutzung das Gefühl des Fliegens, obwohl sich die Person zu jedem Zeitpunkt am Boden befindet.

Abstract

The dream of flying is as old as the humanity itself. To see the world from a bird's view is the goal of many people. But with today's technology the solution to this problem is within reach. For a couple of years drones conquer the market rapidly and nowadays everybody can buy a drone without needing special permissions or licensing. Other techniques such as virtual reality are developing fast to be a mature technology and can be found more and more in people's homes. By putting VR goggles onto their head, people get shifted into different worlds without leaving their trusted and comfortable environment. Combining these two technologies allows people to life the dream of flying.

This research project deals with the development of a drone controller using NUI flight instructions based on a 2D webcam and is part of a series of projects with the title "I believe I can fly". The users can control a simulated drone by performing well-defined gestures in front of a webcam and seeing the drones view inside their worn video goggles. The feeling of flying is created using natural gestures in combination with the view of the cameras image although the users stay all the time on ground.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface, dt. Programmierschnittstelle
COCO	Common Objects in Context
ECCV	European Conference on Computer Vision
FPS	Frames per Second, dt. Bilder pro Sekunde
GPU	Graphics processing unit, dt. Grafikprozessor
ICCV	International Conference on Computer Vision
NUI	Natural User Interface, dt. Natürliche Benutzeroberfläche
SDK	Software Development Kit
SPA	Sense Plan Act
UDP	User Datagram Protocol
UE4	Unreal Engine 4
VR	Virtual Reality, dt. virtuelle Realität

Abbildungsverzeichnis

Typographische Hinweise

- Abkürzungen sind bei ihrer erstmaligen Erscheinung *kursiv* dargestellt. Nähere Informationen sind dem Fließtext bzw. die genaue Bedeutung dem Abkürzungsverzeichnis zu entnehmen.
- Quellcode und technische Namen sind durch `rote Schrift mit grauem Hintergrund` oder eine umgebende Textbox markiert.
- Aus Gründen der Lesbarkeit wird im Text die männliche Form gewählt, nichtsdestotrotz beziehen sich die Angaben auf alle Geschlechter.

1. Einleitung

Diese Studienarbeit ist Teil einer Reihe von Studienarbeiten, die sich mit dem Projekt "I believe I can fly" auseinandersetzen. Bei diesem Projekt geht es im Allgemeinen um die Steuerung einer Drohne mithilfe eines Natural User Interfaces (*NUI*), welches die Körperbewegungen des Piloten detektiert und in Fluganweisungen für eine Drohne klassifiziert. Ziel ist es, dem Piloten ein möglichst realistisches Fluggefühl zu vermitteln und eine hohe Immersion zu erreichen. Dazu trägt der Pilot während des Fliegens eine Videobrille, welche wiederrum einen Livestream einer an der Drohne befestigten Kamera anzeigt. Die Kamera ist dabei so ausgerichtet, dass sie aus Sicht der Drohne stets nach vorne zeigt und dem Piloten die Welt so zeigt, als würde er selbst in der Drohne sitzen.

Eine solche Technik kann für die unterschiedlichsten Einsatzzwecke verwendet werden. Dabei spielt es keine Rolle, ob diese Steuerung für die visuelle Inspektion eines schwer zu erreichenden Ortes wie z.B. einer Hochspannungsleitung oder zu Unterhaltungszwecken von Heimanwendern verwendet wird. Ein wichtiger Aspekt ist in allen Anwendungsfällen eine intuitive Steuerung und ein realistisches Gefühl für das Fliegen, sodass die Drohne präzise gesteuert werden kann.

Die erste Version des Projektes entstand im Rahmen einer Studienarbeit 2015 und setzt die Grundanforderungen bereits funktionierend um. Zum Einsatz kommt hierbei ein Sensor der Xbox Kinect, welcher mithilfe eines 3D-fähigen Infrarot-Sensors den Benutzer und seine Körperbewegungen erkennen kann. Die erkannten Körperbewegungen werden anschließend in Befehle für eine Drohne umgerechnet und ermöglichen eine zuverlässige Steuerung. (1)

Eine weitere Studienarbeit befasste sich mit einem intelligenten Assistenzsystem, welches den Piloten bei der Navigation durch Engstellen unterstützen soll. Aufgrund von technischen Schwierigkeiten kommt diese Arbeit jedoch zu keinem positiven Ergebnis, sodass ein solche Assistenzsystem noch in den frühen Kinderschuhen steckt. (2)

1.1. Problemstellung

Die erste Version von "I believe I can fly" funktioniert zwar recht gut und ermöglicht dem Benutzer ein realistisches Gefühl des Fliegens, basiert jedoch auf Hardware, welche möglicherweise nicht jedem Benutzer zur Verfügung steht. Die Produktion des bereits erwähnten Kinect-Sensors wurde von Microsoft offiziell eingestellt und ist somit

nicht mehr für Neukunden verfügbar. (3) Selbst wenn ein Nutzer Zugriff auf einen solchen Sensor hätte, müsste er diesen bei jeder Verwendung der Drohne benutzen. Auch die Verwendung eines 3D-Sensors eines anderen Herstellers kann dieses Problem nicht umgehen.

Eine mögliche Lösung liegt in der Verwendung einer handelsüblichen 2D-Webcam, welche heute in fast allen mobilen Endgeräten wie Laptops oder Smartphones verbaut ist. Diese Kameras sind günstig in der Herstellung und liefern schon seit einigen Jahren die benötigte Qualität des aufgenommenen Bildes (siehe Kapitel 1.2.1 Hardware). Das Problem bei der Verwendung einer 2D-Informationsquelle liegt an der fehlenden dritten Dimension, welche für eine vollständige 3D-Steuerung eigentlich benötigt werden würde. Dieses Problem wiederum kann durch verschiedene Verfahren gelöst werden, welche in Kapitel 4 Gestenerkennung verglichen werden.

Neben dem bisher benötigten Sensor wird für die erste Version des Projektes zum Fliegen auch immer eine reale Drohne benötigt. Neben der Drohne muss aber auch eine für den Flugbetrieb geeignete Umgebung vorhanden sein. Im Freien spielen oftmals das Wetter sowie gesetzliche Bestimmungen für den Aufstieg einer Drohne eine große Rolle, in geschlossenen Räumen ist meistens der vorhandene Platz der limitierende Faktor. Vor Allem für Trainingszwecke ist dies aber eher hinderlich. Ziel dieser Studienarbeit ist also auch die Integration eines Simulators, welcher die Drohne vollständig ersetzt. Der Benutzer sieht dann an Stelle der Realität aus Sicht der Drohne ein gerendertes Bild eines Simulators, welches möglichst realistisch aussehen soll um weiterhin das Gefühl des Fliegens vermitteln zu können. Mögliche Lösungen werden in Kapitel 5.1 Steuerung des Simulators behandelt.

1.2. Technische Voraussetzungen

Von Beginn des Projektes an sind klare Anforderungen an sowohl die verwendete Software als auch die verwendete Hardware gegeben. Dieses Voraussetzungen ermöglichen eine flexible Laufzeitumgebung und heben Limitierungen wie ein bestimmtes Betriebssystem von Anfang an auf.

1.2.1. Hardware

Die Aufnahme der Bilder, Erkennung der Gesten als auch die Simulation bzw. Steuerung der Drohne sollen nach Möglichkeit auf einem einzigen Gerät stattfinden. Um eine gewisse Mobilität zu erreichen, bietet sich für diese Zwecke ein leistungsstarker Laptop an, welcher eine handelsübliche Webcam eingebaut hat.

Die Webcam dient der Aufnahme einzelner Bilder des steuernden Benutzers und sollte daher von guter Qualität sein. Die Qualität wird unter anderem durch die Bilder pro Sekunde (*FPS*) als auch die zur Verfügung stehende Auflösung der Bilder bestimmt. Ist einer dieser beiden Parameter zu gering, leidet die Qualität der Steuerung darunter. Zu wenige FPS führen zu einer sehr trägen Steuerung und eine zu geringe Auflösung kann unter Umständen zu einer fehlschlagenden Gestenerkennung führen. Beide Konsequenzen beinträchtigen unmittelbar die Immersion des Benutzers und sollten daher vermieden werden. (4)

Sowohl die Erkennung der Gesten als auch die Simulation der Drohne benötigen eine hohe verfügbare Rechenleistung. Da die Berechnungen gleichzeitig ausgeführt und die entsprechenden Algorithmen parallelisiert werden können, ist vor Allem eine leistungsstarke Grafikkarte (*GPU*) mit einer großen Anzahl an Rechenkernen notwendig. Die verschiedenen Hersteller von Grafikkarten haben unterschiedliche Architekturen mit verschiedenen Eigenschaften, welche sich unterschiedlich gut für solche Berechnungen nutzen lassen. Dies sollte bei der Auswahl des Computers unbedingt beachtet werden. Daher empfiehlt es sich, auf eine moderne Plattform zu setzten, welche die neusten Features unterstützt, da sich in den letzten Jahren einiges in diesem Bereich entwickelt hat. (5) Der zur Verfügung stehende Laptop hat einen Prozessor mit vier Kernen der Firma Intel und eine NVIDIA Grafikkarte der Reihe GTX 1060 verbaut, welche gut für solche Einsatzzwecke geeignet ist und ausreichend Leistung liefern sollte.

Um eine Drohne über einen Rechner ansteuern zu können, muss die Drohne diese Art der Kommunikation logischerweise ebenfalls unterstützen. Auf dem Markt sind einige dieser Drohnen zu finden, jedoch ist die eigene Software-Entwicklung bei einem speziellen Modell sehr gut möglich, nämlich der *AR.Drone* des Herstellers Parrot. Der Hersteller stellt Entwicklern ein Software Development Kit (*SDK*) zur Verfügung, welches die Steuerung der Drohne für Programme über ein Application Programming Interface (*API*) ermöglicht. (6) Dieser Quadrocopter wurde ebenfalls in den beiden vorangegangen Studienarbeiten verwendet und wird deshalb auch im Rahmen dieser Studienarbeit verwendet werden.

1.2.2. Robot Operating System

Software sollte nach Möglichkeit immer modular aufgebaut werden und optimalerweise plattformunabhängig funktionieren. Dies hat den Vorteil, dass Quellcode nur einmal

geschrieben werden muss und anschließend wiederverwendet werden kann, ohne eine Anpassung für die entsprechende Plattform vornehmen zu müssen. In der Robotik gibt es für diesen Zweck das Robot Operating System (*ROS*), welches typischerweise auf Linux aufsetzt und Entwicklern eine Sammlung verschiedener Bibliotheken zur Verfügung stellt. Die notwendige Infrastruktur für modulare Software wird ebenfalls bereitgestellt und ermöglicht durch die Kompatibilität zu Programmiersprachen wie C, C++ und Python ein breites Spektrum an Anwendungen. (7)

ROS wird als Open Source Projekt gepflegt und kann somit theoretisch von jeder Person erweitert werden. Verschiedene Linux-Distributionen werden hierbei offiziell als Betriebssystem unterstützt, jedoch gibt es auch die Möglichkeit, ROS auf Windows und MacOS zu betreiben. Diese Varianten werden offiziell nicht unterstützt oder getestet, jedoch durch die Community vieler Entwickler unterstützt.

Die Modularität wird in ROS durch Pakete umgesetzt. Ein Paket hat im Idealfall eine übergeordnete Aufgabe und besteht intern aus weiteren Nodes. Ein Node übernimmt genau eine Aufgabe und kommuniziert mit anderen Nodes über Topics, um z.B. komplexere Algorithmen umsetzten zu können und die inhaltliche Zuordnung möglichst gering zu halten. Topics sind spezielle Kommunikationssysteme, welche Nachrichten von Nodes entgegennehmen und an alle anderen Nodes weiterreichen. Weitere Nodes können auf bestimmte Nachrichten in einem Topic warten und beim Empfangen der Nachricht ihr Verhalten anpassen. Dieses System ist in der Informatik weitverbreitet und wird in der ereignisgesteuerten Programmierung auch als Bussystem bezeichnet. (8)

Dieses Konzept soll auch in dieser Studienarbeit verwendet werden. So kann ein Node die Erkennung der Person übernehmen, ein weiterer Node wiederrum übernimmt die Erkennung der Gesten usw. Somit ist die Software modular aufgebaut und so können, falls alle Schnittstellenspezifikationen eingehalten werden, einzelne Module problemlos ausgetauscht werden (siehe 2.1 Architektur).

1.2.3. Simulation einer Drohne

Unter gewissen Umständen ist es nicht immer möglich, eine reale Drohne fliegen lassen zu können. Deshalb kann es sinnvoll sein, die Drohne lediglich in einem Simulator fliegen zu lassen. Das Bild der echten Kamera, welches der Benutzer normalerweise in seiner Videobrille sehen würde, wird bei diesem Aufbau durch ein

virtuelles Bild ersetzt und erlaubt dem Nutzer eine realistische Immersion. Wichtig an dieser Stelle ist die möglichst realistische Grafik der Simulation, sodass nicht das Gefühl einer simulierten Umgebung entsteht. Diese Möglichkeit ist vor Allem während der Entwicklung oder als Training für neue Nutzer sinnvoll und senkt dabei gleichzeitig das Risiko, dass durch unkontrolliertes Fliegen oder einen Fehler ein tatsächlicher Schaden entsteht. Für die Simulation stehen softwaretechnisch mehrere Möglichkeiten zur Auswahl, welche in Kapitel 2.2 Simulator genauer analysiert und verglichen werden.

2. Software Komponenten

Wie bereits erläutert, unterstützt ROS verschiedene Programmiersprachen. Bei der Wahl der Programmiersprache müssen verschiedene Kriterien beachtet werden, welche eine Programmiersprache für ein solches Vorhaben qualifiziert:

1. Parallelität: Die Programmiersprache sollte Nebenläufigkeit in Form von Threads nativ unterstützen. Dies ist bei mehreren parallelen Aufgaben wie der Personenerkennung, Positionserkennung und Steuerung einer Drohne unabdingbar und erhöht die potenzielle Leistung einer Anwendung. Müsste ein Node immer auf die Fertigstellung der vorherigen Nodes warten, würden immer mehrere Nodes untätig sein und nur auf Eingabewerte anderer Nodes warten. Können die Nodes jedoch ihre Berechnungen gleichzeitig auf mehreren Prozessorkernen verteilt ausführen, steigt die potentielle Leistung stark an. Deshalb sollten die Nodes auch so unabhängig wie möglich voneinander sein und nicht stark voneinander abhängen. In der Praxis ist dies leider nicht immer möglich, da die Positionserkennung ohne die Koordinaten der einzelnen Körperteile kein Ergebnis berechnen kann.
2. Unterstützung von ROS: Da die Verwendung von ROS verlangt ist, ist diese Bedingung natürlich nicht zu vernachlässigen. ROS unterstützt von offizieller Seite aus nur eine geringe Auswahl an Programmiersprachen, jedoch wird durch die Community eine Unterstützung vieler weitere Programmiersprachen gewährleistet. Bei diesen Community-Implementierungen der ROS Unterstützung darf jedoch nicht dieselbe Qualität an Zuverlässigkeit und Funktionalität wie bei den offiziell unterstützen Programmiersprachen erwartet werden. An diesen Erweiterungen arbeiten oftmals keine großen Teams, sondern lediglich eine kleine Anzahl von Entwicklern.
3. Unterstützung weiterer Bibliotheken: In der heutigen Softwareentwicklung ist die ständige Wiedererfindung des Rads bei weitem nicht mehr nötig. Stattdessen kommen Bibliotheken zum Einsatz, welche eine gewisse Aufgabe standardisiert übernehmen. Dies spart in der Entwicklung hohen Entwicklungsaufwand, da ein Entwickler bereits gelöste Probleme nicht erneut lösen muss. Verschiedene Programmiersprachen haben eine unterschiedlich gute Verfügbarkeit von Bibliotheken für verschiedene Einsatzzwecke. So eignet sich z.B. Python hervorragend für wissenschaftliche Berechnungen,

währenddessen C# perfekt für die clientseitige Entwicklung geeignet ist. Beide Sprachen bieten eine Vielzahl von verschiedenen Bibliotheken, jedoch sind beide Sprachen für vollkommen unterschiedliche Anwendungszwecke entwickelt worden.

Jeder Programmierer hat seine eigenen Vorlieben bei der Wahl der Programmiersprache, basierend auf dem Anwendungszweck, gemachten Erfahrungen und den eigenen Fähigkeiten für die jeweilige Programmiersprache. Durch die Verwendung von ROS bieten sich die nativen Integrationen in C++ und Python sehr gut an. Letzten Endes wird im Rahmen dieses Projektes Python verwendet werden, da Python alle Anforderungen für dieses Projekt erfüllt und auch offiziell von ROS unterstützt wird. Mit Ausblick auf die Integration eines Simulators ist Python auch von Vorteil, mehr dazu in Kapitel 5.1 Steuerung des Simulators.

2.1. Architektur

Das Ziel dieser Softwarearchitektur ist es, die Software möglichst modular aufzubauen, wobei jedes Modul (ROS Node) eine feste Aufgabe übernimmt. Ein Überblick über die verschiedenen Nodes kann Abbildung 1 entnommen werden.

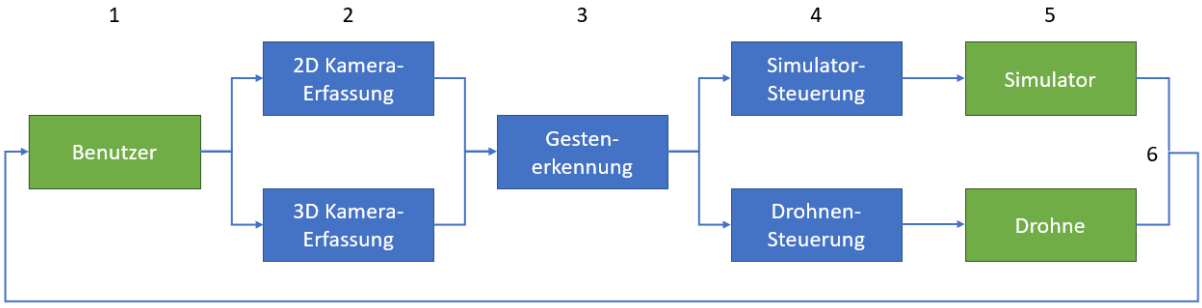

Abbildung 1: ROS Node Architektur

Grüne Ressourcen sind nicht direkter Teil der Software, werden aber zwingend für die erfolgreiche Ausführung benötigt. Dies ist zum einen der Benutzer an sich, welcher die Gesten ausführt und somit die Kontrolle über die Steuerung übernimmt. Zum anderen gehört die Ausgabe nicht direkt zur Software, da sowohl die Drohne als auch der Simulator lediglich angesteuert werden und die Kontrolle der Umsetzung dieser Ansteuerung nicht in Händen dieses Projektes liegt. Blaue Ressourcen sind direkter Teil der Software und werden im Rahmen dieser Arbeit entwickelt werden. Jedes Rechteck entspricht dabei genau einem Node, welcher die jeweils ihm zugeordnete Aufgabe übernimmt.

Die Verarbeitung der Signale ist in mehrere Teilschritte unterteilt:

(1) Der Benutzer führt verschiedene Gesten aus und liefert dem System somit verschiedene Eingabeparameter, welche anschließend vom System in konkrete Steuerungsbefehle umgewandelt werden. Die Gesten werden vom Projekt fest definiert und können in beliebiger Reihenfolge nacheinander ausgeführt werden. Jede Geste entspricht dabei einem konkreten Steuerungsbefehl, welche wiederum miteinander kombiniert werden können. Die Steuerung wird in Kapitel 4 ausgearbeitet.

(2) Im zweiten Schritt werden die Gesten des Benutzers aus dem ersten Schritt durch verschiedene Kamera-Technologien erkannt und zur Weiterverarbeitung aufbereitet. Bei der Wahrnehmung des Benutzers soll immer die bestmögliche verfügbare Technik verwendet werden. Ist am System eine 3D fähige Kamera wie die Xbox Kinect oder Intel RealSense angeschlossen, soll diese Kamera verwendet werden. Ist hingegen nur eine 2D fähige Webcam verfügbar, soll diese verwendet werden. Je besser die Kamera ist, desto besser können die Gesten anschließend erkannt werden. Die Nodes des zweiten Schritts ermitteln ebenfalls aus dem Kamerabild die Koordinaten der einzelnen Körperteile, welche anschließend im nächsten Schritt verarbeitet werden.

(3) In der Positionserkennung werden die erhaltenen Koordinaten der einzelnen Körperteile zu Gesten zusammengesetzt. Dazu kommt ein Fuzzy-Controller zum Einsatz, welcher verschiedene Eingabeparameter zu mehreren Ausgabeparametern transformiert. Die Verarbeitung der Parameter ist in Kapitel 4 beschrieben. Als Ausgabe erzeugt dieser Schritt allgemeine Flugbefehle, welche für alle Ausgabemöglichkeiten gültig sind.

(4) Im vierten Schritt werden die allgemeinen Flugbefehle in konkrete Steuerungsbefehle für die jeweils gewählte Ausgabe umgewandelt. Jede Ausgabemöglichkeit wird dabei über zur Verfügung gestellte APIs der verschiedenen Umgebungen realisiert. Der Simulator unterscheidet sich dabei grundlegend von der Steuerung einer echten Drohne, weshalb diese konkreten Steuerungen jeweils als eigene Nodes implementiert werden. Auf diese Weise können neue Ausgabemöglichkeiten relativ einfach an das System angebunden werden, ohne bestehende Strukturen verändern zu müssen.

(5) Dieser Schritt stellt die Steuerung der gewählten Ausgabeoption dar. Aufgabe dieses Schrittes ist es, ein Bild aufzunehmen bzw. zu erzeugen und es dem Benutzer zur Verfügung zu stellen. Im Falle eines künstlichen Bildes aus dem Simulator sollte dies so realistisch wie möglich sein, denn nur so kann dem Benutzer ein Gefühl des Fliegens vermittelt werden. Bei einer echten Drohne ist das Bild aus Sicht der Drohne gerade nach vorne zu bevorzugen.

(6) Im letzten Schritt wird das aufgenommene bzw. erzeugte Bild aus Sicht der Drohne an den Benutzer zurückgegeben. Dazu trägt der Benutzer eine Videobrille, welche ihm genau diese Bilder anzeigt. Anschließend beginnt der gesamt Steuerungszyklus von vorne.

Für die Kommunikation zwischen den einzelnen Nodes wird das Nachrichtensystem von ROS verwendet. Jeder Node hat die Möglichkeit, Nachrichten auf sogenannten Topics zu versenden und fremde Nachrichten zu empfangen. Hat ein Node die zu erledigende Arbeit beendet, wird das Ergebnis in der Regel auf einem definierten Topic an alle Nodes gesendet. Der nächste Node der Verarbeitungskette hat verschiedene Topics abonniert und kann bei Eingang einer Nachricht wiederrum mit seiner Arbeit beginnen.

Das Format einer Nachricht ist genaustens definiert und lässt keinerlei Abweichungen zu. Dies ermöglicht zuverlässige Schnittstellen zwischen mehreren Nodes und sichert die Kommunikation gleichzeitig ab, sodass bezüglich der Formatierung keine unerwarteten Nachrichten beim Empfänger eintreffen können. Jede Nachricht kann rekursiv aus weiteren Nachrichten bestehen und besteht auf unterster Ebene immer aus elementaren Datentypen (Nummern, Zeichenketten, booleschen Wahrheitswerten) bzw. einer Auflistung dieser. (9)

Der Vorteil von Nachrichten besteht darin, dass die Nodes in unterschiedlichen Programmiersprachen programmiert sein können und trotzdem ohne Probleme und eindeutig miteinander kommunizieren können. Ohne ein solch zentrales System ist die Kommunikation über mehrere Programmiersprachen hinweg nicht ohne weiteres möglich. Weiterhin können Nodes unter der Bedingung, dass sie das festgelegte Nachrichtenformat einhalten, beliebig ausgetauscht werden und die Komplettlösung ist von den Teilproblemen losgelöst, die jeweils von einem Node gelöst werden.

Wie der Verarbeitungskette zu entnehmen ist, hat die Verarbeitung die Form eines Kreislaufs. In der Robotik ist ein sehr ähnliches Vorgehen unter dem Namen Sense

Plan Act (*SPA*) bekannt. Für den Vergleich zu diesem Projekt ist die Software als Roboter anzusehen. Im ersten Schritt *Sense* nimmt ein Roboter seine Umgebung wahr. Dazu dienen ihm verschiedene Sensoren wie Kameras, Infrarotsensoren usw. Im Beispiel des Projektes orientiert sich das Programm als Ganzes anhand der Gesten des Benutzers, welche durch eine Kamera aufgenommen werden (Abbildung 1, Schritt 1-2). Anschließend plant der Roboter sein weiteres Vorgehen in der *Plan* Phase. Auch die Steuerungssoftware wandelt die wahrgenommenen Gesten in konkrete Befehle um und plant somit die nächsten auszuführenden Flugbefehle (Abbildung 1, Schritt 3). Im letzten und vorerst abschließenden Schritt *Act* führt der Roboter die geplante Aktion aus. Auch die Software verhält sich vergleichbar und steuert im letzten Schritt den Simulator bzw. die Drohne an (Abbildung 1, Schritt 4-5). Da nach der einmaligen Ausführung der Ablauf nicht beendet ist, beginnt der Kreislauf wieder von vorne. Der Roboter nimmt die Umgebung also erneut wahr, plant die nächste Aktion und führt diese aus. Die Software gibt dem Benutzer das aufgenommene Bild aus, woraufhin dieser mit einer entsprechenden Geste reagiert, welche anschließend erneut wahrgenommen und verarbeitet wird.

2.2. Simulator

Neben der Gestensteuerung soll ebenfalls die Möglichkeit zur Simulation einer Drohne entwickelt werden. Der Simulator kommt dann zum Einsatz, wenn eine echte Drohne nicht zur Verfügung steht oder es schlichtweg nicht möglich ist, eine echte Drohne unter den Gegebenheiten fliegen zu lassen. Ursachen für diese Umstände könnten Platzmangel, schlechtes Wetter, große Menschenmengen, unerfahrene Piloten und vieles mehr sein. Deshalb bietet es sich an, die Drohne an einem Computer in einer simulierten Umgebung fliegen zu lassen und das Risiko eines Schadens so gering wie möglich zu halten.

Eine realistische Flugsimulation ist eine gewaltige Aufgabe. Neben physikalischen Berechnungen, der grafischen Benutzerausgabe oder der Programmierschnittstelle gibt es noch viele weitere zu bewältigende Problemstellungen. Deshalb ist eine Eigenentwicklung für den Rahmen dieser Studienarbeit zu aufwändig und liegt auch nicht im Fokus dieser Arbeit. Stattdessen soll ein bereits bestehender Simulator zu Einsatz kommen, welcher diese Probleme bereits gelöst hat.

Bei der Wahl eines passenden Simulators gibt es, wie bei der Wahl der Programmiersprache auch, einige Dinge zu beachten:

1. Steuerbarkeit über Schnittstelle: Viele Simulatoren lassen sich nicht über eine Programmierschnittstelle ansteuern. Stattdessen können diese Simulatoren lediglich über Maus und Tastatur oder eine Modellbau-Fernsteuerung steuern. Eine Umrechnung der erkannten Flugbefehle in Tastenkombinationen ist zwar prinzipiell denkbar, jedoch längst nicht so präzise und steuerbar wie eine Anbindung über eine Schnittstelle. Dies liegt an einer typischen Eigenschaft einer Tastatur: Eine Taste ist entweder gedrückt oder nicht. Tasten können nicht nur zur Hälfte gedrückt werden. Wenn das Ziel eine dynamische Steuerung mit unterschiedlich starken Steuerungsbefehlen (vergleiche einen schnellen mit einem langsamen Vorwärtsflug) ist, ist diese Gegebenheit ein großes Hindernis. Eine Schnittstelle sollte also auf jeden Fall vorhanden sein und eine feingranulare Steuerung ermöglichen.
2. Realistische Grafik: Ziel des Projektes ist es, dem Benutzer ein möglichst realistisches Gefühl des Fliegens zu vermitteln. Dafür ist eine gute Grafik in Form von hochauflösenden Texturen notwendig, die dem Benutzer in seiner Videobrille angezeigt werden kann. Ebenso sollten in dem gewählten Simulator realistische Umgebungen verfügbar sein, welche dem Benutzer vertraut vorkommen. Dies könnte eine einfach Wohnsiedlung oder ein kleiner Ausschnitt einer Stadt sein, jedoch sollte sich der Benutzer an alltäglichen Dingen orientieren können.
3. Performance: Da alle Schritte von der Personenerkennung bis zur Bildanzeige auf einem Computer ablaufen sollen, muss der Simulator auch mit geringer zur Verfügung stehender Performance auskommen. Flüssige Bildraten sind für das Immersionsgefühl erforderlich und sollten nicht unter eine gewisse Grenze fallen. Deshalb ist ein sparsamer Umgang mit Ressourcen wie der verfügbaren Rechenleistung und dem Speicherverbrauch von Vorteil.
4. Plattformunabhängigkeit: Das gesamte Projekt soll nach Möglichkeit unabhängig von dem darunter liegenden Betriebssystem laufen können. In Folge dessen muss auch der Simulator auf gängigen Betriebssystemen wie Windows, Linux und MacOS funktionieren.

Für eine frühere Version dieser Studienarbeit wurde eine Implementierung von Gazebo für ROS verwendet, die den Namen „tum_simulator“ trägt. Die Implementierung wurde an der Technischen Universität München geschrieben und

emuliert die Steuerung der AR.Drone 1.0 und 2.0. Diese Emulation hat den Vorteil, dass bei der Ansteuerung nicht zwischen einer echten oder simulierten Drohne unterschieden werden muss, da der Emulator die gleiche Schnittstelle wie sein reales Vorbild nutzt. (10) Das Problem an dieser Implementierung liegt an der nicht ausreichenden Grafik-Qualität, welche subjektiv betrachtet kein realistisches Gefühl des Fliegens vermitteln kann (siehe Abbildung 2).

Abbildung 2: tum_simulator

Eine weitere Kategorie der Simulatoren sind die für Unterhaltungszwecke geschaffenen Simulatoren. Insbesondere Simulatoren für Drohnenrennen sind für ihre realistischen Physik-Simulationen bekannt und speziell auf Drohnen ausgelegt. Verbreite Programme sind unter Anderem Liftoff, Velocidrone oder der DRL Sim. So werden für den DRL Sim alle im Simulator verfügbaren Bauteile auf ihre Aerodynamik untersucht um anschließend im Simulator möglichst realitätsnah simuliert zu werden. (11) Auch die Grafik dieser Simulatoren ist sehr gut, da die Simulatoren für Unterhaltungszwecke gebaut worden sind und die Performance auf einem typischen Gaming-PC bei weitem ausreicht. Jedoch gibt es zwei grundlegende Probleme, welche diese Simulatoren für diesen Einsatzzweck unbrauchbar machen: Zum einen erlauben diese Simulatoren oftmals nicht das Bauen von eignen Welten und die bereits vorhandenen Welten sind stark auf Rennen optimiert. Es könnten also keine eigenen Szenarios geladen werden, welche dem Benutzer in der Realität weiter helfen. Zum anderen ist die fehlende Schnittstelle für den programmatischen Zugriff ein großes

Problem. Keiner der verfügbaren Simulatoren erlaubt den Zugriff auf die Drohne über eine API und ist somit für diese Studienarbeit unbrauchbar.

Seit der Verwendung des Simulators „tum_simulator“ ist einige Zeit vergangen und Microsoft hat in der Zwischenzeit AirSim weiterentwickelt und eine Verbindung zu ROS ermöglicht. AirSim ist ein Fahrzeug-Simulator, der für das Trainieren von autonomen Fahrzeugen entwickelt wird. Technisch gesehen basiert AirSim auf der Unreal Engine 4 (*UE4*) und ist als Plugin für jede beliebige Welt verfügbar. So kann ein Entwickler eine ganz neue Welt in der Unreal Engine entwickeln und AirSim als Plugin hinzufügen; dasselbe gilt auch für bereits bestehende Welten. Microsoft selbst stellt ebenfalls einige Welten zur Verfügung, die frei genutzt werden können. (12)

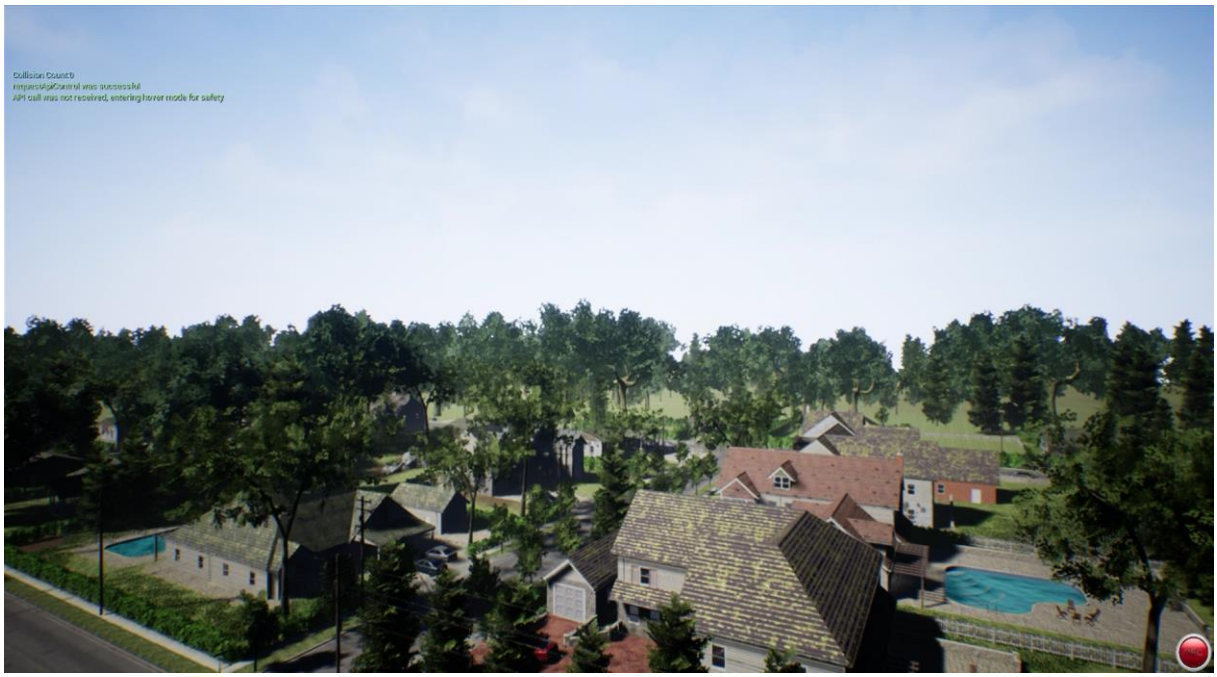

Abbildung 3: AirSim Simulator

Durch die Verwendung der Unreal Engine sind die erzeugten Bilder von hoher Qualität, erfordern aber auch eine gewisse Rechenleistung. AirSim erreicht auf dem vorliegenden Laptop bei alleinigem Betrieb in etwa 60 FPS. Die Ansteuerung erfolgt über eine API, die sowohl für C++ als auch Python verfügbar ist. Ebenfalls ist die Unreal Engine auf allen benötigten Plattformen verfügbar und erfüllt somit alle definierten Anforderungen. Aus diesem Grund wird für diese Studienarbeit AirSim basierend auf der Unreal Engine verwendet werden.

3. Personenerkennung

Der erste Schritt der Verarbeitungskette (Abbildung 1) ist die Personenerkennung, welche ein Bild des Benutzers aufnimmt und die Koordinaten der einzelnen Körperteile bestimmen soll. Obwohl das Problem sowohl für 2D- als auch 3D-Bilder besteht, wird nachfolgend nur auf die Verarbeitung von 2D-Bildern eingegangen. Die Verarbeitung von 3D-Bildern ist vorerst nur eine geplante Erweiterung (siehe Kapitel 6).

3.1. Problemstellung

Das Problem der Personenerkennung besteht aus mehreren Teilproblemen, welche für ein gutes Ergebnis zuverlässig gelöst werden müssen. Zu Beginn muss unterschieden werden, welche Bildquelle vorliegt. Die Grundidee des Projektes ist die Reduktion der benötigten Hardware, um eine Drohne fliegen zu lassen. In früheren Versionen wurde ein Xbox Kinect Sensor verwendet, welcher gekauft, aufgebaut und eingerichtet werden muss. Außerdem muss dieser Sensor bei jeder Verwendung der Software vorhanden sein. Dies bedeutet zusätzlichen Aufwand und eine weitere mögliche Fehlerquelle. Stattdessen soll es nun möglich sein, eine einfache Webcam, wie sie in fast jedem Laptop verbaut ist, als Bildquelle zu verwenden. Die Webcam gibt jedoch nur 2D-Bilder aus und ist daher auf den ersten Blick nur weniger gut für eine 3D-Steuerung zu gebrauchen.

Ein weiteres Problem ist die eigentliche Erkennung der Person auf einem Bild. Die Objekterkennung ist im Bereich der Bildverarbeitung ein alt bekanntes Problem, für welches es verschiedene Lösungsansätze mit unterschiedlichen Ergebnissen gibt. Grundsätzlich kann das Problem mit verschiedenen Algorithmen der Bildverarbeitung angegangen werden. Die Extraktion von konkreten Koordinaten mehrerer Objekte stellt sich jedoch schon als schwierig heraus. Als Alternative wurden vor allem in den letzten Jahren neuronale Netze immer weiter entwickelt und leistungsstärker. Sind neuronale Netze richtig trainiert, können sie ein Problem zuverlässig und in kurzer Zeit lösen.

Unabhängig von der Wahl der Lösung, gibt es grundlegend zwei verschiedene Ansätze zur Problemlösung: Objekterkennung und Objektverfolgung. Durch die Kamera ist immer eine Folge von mehreren Bildern gegeben. Bei der Objekterkennung wird jedes einzelne Bild auf das gesuchte Objekt gescannt und bei dem darauffolgenden Bild werden sämtliche Ergebnisse des vorherigen Bildes ignoriert. Objektverfolgung macht sich den benötigten Rechenaufwand des vorherigen Bildes zu

Nutze und sucht auf einem neuen Bild nur nach Veränderungen im Vergleich zum vorherigen Bild. Ziel der Verfolgung ist es, eine Positionsveränderung eines erkannten Objektes für jedes Bild berechnen zu können. (13) Dies erlaubt eine performante Auswertung der Bilder, macht die einzelnen Bilder jedoch voneinander abhängig und benötigt im Allgemeinen mehrere Bilder für eine erfolgreiche Auswertung. Objekterkennung benötigt im Endeffekt etwas mehr Rechenaufwand, kann dafür aber auch bei starken Szenen- und Beleuchtungswechseln die gesuchten Objekte zuverlässig erkennen.

3.2. Digitale Bildverarbeitung

Die digitale Bildverarbeitung ist ein weites Fachgebiet der Informatik und beschäftigt sich mit der Auswertung des Inhaltes von Bildern. In vielen Anwendungsbereichen wird Bildverarbeitung für die unterschiedlichsten Einsatzzwecke verwendet und dient unter Anderem der Qualitätskontrolle in Produktionsreihen, medizinischen Untersuchungen oder der Verkehrsüberwachung. Dazu werden in der Regel fünf verschiedene Schritte für ein Bild angewendet. (14 S. 2-3)

1. Bilderfassung: Während der Bilderfassung wird ein natürliches Bild mit einer Kamera aufgenommen und in ein elektronisches Format umgewandelt. Die aufzunehmende Szene sollte für den Einsatzzweck optimal beleuchtet sein, um ein bestmögliches Ergebnis erzielen zu können. Entstehen in diesem Schritt bereits qualitativ schlechte Bilder, erschwert dies die weitere Verarbeitung ungemein und impliziert zwangsweise ein schlechteres Ergebnis.
2. Vorverarbeitung: Die aufgenommenen Bilder müssen vor der eigentlichen Verarbeitung wie einer Objekterkennung vorverarbeitet werden. So können z.B. die Kanten eines Bildes extrahiert oder die Farbwerte auf Graustufen reduziert werden. Je nachdem was mit dem Bild erreicht werden soll, müssen in diesem Schritt verschiedene Algorithmen angewendet werden.
3. Segmentation: Die Segmentation zerteilt das Gesamtbild in kleinere Teilbereiche, welche gemeinsame Kriterien erfüllen und im Optimalfall alle vom gleichen Objekt des aufgenommenen Bildes stammen. Oftmals wird aber auch das gleiche Objekt in mehrere Segmente unterteilt, wenn z.B. eine Hälfte des Objektes im Schatten liegt und die andere Hälfte direkt von einer Lichtquelle angestrahlt wird. Auf diese Weise soll auch der Vordergrund vom Hintergrund des Bildes getrennt werden.

4. Merkmalsextraktion: Das Ziel der Merkmalsextraktion ist die Beschreibung von Segmenten und die Zusammenführung mehrerer Segmente, falls diese ähnliche Merkmale haben. So ist auch die Erkennung von komplexeren Objekten und bei schwierigen Lichtverhältnissen möglich.
5. Klassifikation: Der letzte Schritt der Bildverarbeitung ist die Klassifikation der gesammelten Informationen. In diesem Schritt werden die zu ermittelnden Daten aus dem Bild gezogen, welche anschließend weiter verarbeitet werden können.

Nicht alle Schritte müssen zwingend für jedes Bild durchlaufen werden, jedoch basieren viele Verfahren auf genau diesen Schritten und benötigen in späteren Schritten definierte Gegebenheiten. Der Einsatz dieser Verfahren kann dabei variieren und unterschiedlich kombiniert werden.

3.2.1. Hintergrundextraktion

Zur Lösung des Personenerkennungs-Problems ist es ein erster Versuch, die Person über eine Hintergrundextraktion zu erkennen. Dieser Schritt ist typisch für die Bildverarbeitung und soll den Fokus der nächsten Schritte auf den Vordergrund legen, in welchem typischerweise die interessanten Objekte zu finden sind. Ist der „normale“ Hintergrund bekannt, kann der bekannte Hintergrund einfach von dem aktuell aufgenommenen Bild abgezogen werden und der Vordergrund ist in voller Farbe verfügbar. Jedoch ist der Hintergrund nicht immer bekannt, da die Kamera auch an einem anderen Ort platziert werden kann und auch sich bewegende Objekte im Hintergrund zu finden sind. Außerdem müssen die Lichtverhältnisse für diese Methode immer die gleichen sein, um eine regelmäßige und gleichmäßige Beleuchtung zu erzeugen. Dieses Verfahren bietet sich also nicht für realistische und sich ändernde Umgebungen an.

Stattdessen wird der Vordergrund über die Differenz mehrerer Bilder über einen Zeitraum hinweg berechnet. Ändert sich ein Objekt in einem Bild über eine gewisse Zeit nicht, kann dieses als Hintergrundobjekt mit geringer Relevanz für die Szene angesehen werden. Der sich bewegende Teil spielt offenbar eine wichtigere Rolle und scheint somit im Vordergrund zu stehen. (15) Das Problem dieser Methode ist, dass sich der Vordergrund nur durch eine Bewegung vom Hintergrund trennt. Steht eine Person still oder zumindest teilweise still, werden diese still gehaltenen Körperteile möglicherweise nicht vom Algorithmus erkannt.

3.2.2. Implementierung mit OpenCV

Eine weit verbreitete und umfangreiche Bibliothek für Bildverarbeitung heißt OpenCV und ist für C++, Python und Java verfügbar. OpenCV ist ein Open Source Produkt und wird immer wieder um neue Verfahren der Bildverarbeitung erweitert. Durch die Festlegung auf die Programmiersprache Python für dieses Projekt eignet sich auch OpenCV mit der Python-Schnittstelle gut für diese Einsatzzwecke. Für die Hintergrundextraktion sind in OpenCV verschiedene Verfahren implementiert, welche jeweils ihre Vor- und Nachteile haben.

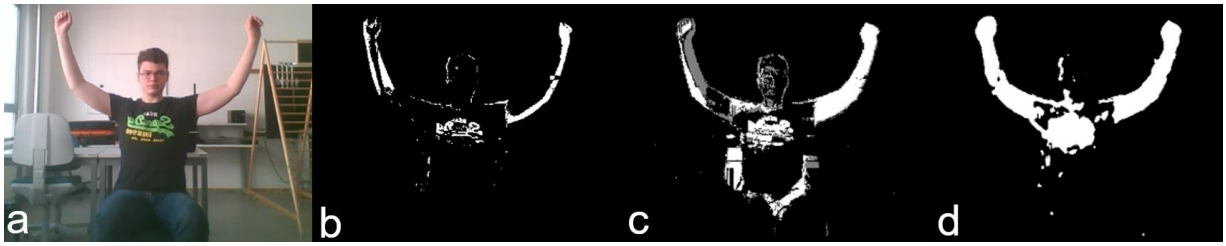

Abbildung 4: OpenCV - Background Subtraction Algorithmen a) Original
b) BackgroundSubtractorMOG c) BackgroundSubtractorMOG2 d) BackgroundSubtractorGMG

Die ersten beiden Algorithmen tragen den Namen *BackgroundSubtractorMOG* bzw. *BackgroundSubtractorMOG2* und basieren auf einem Segmentierungs-Algorithmus mit Verwendung einer Gaußschen Mischverteilung für den Hintergrund, veröffentlicht in mehreren Papers in den Jahren 2001 bzw. 2004/-05. (16) (17) (18) Der zweite Algorithmus ist dabei eine Weiterentwicklung des ersten, welcher auch mit wechselnder Beleuchtung und Schatten gut umgehen kann. Bei beiden Algorithmen werden Pixel mit verschiedenen Wahrscheinlichkeiten in Abhängigkeit zu der Sichtbarkeitsdauer über mehrere Bilder hinweg gewichtet und modelliert. Pixel werden dabei anhand ihrer Farbe identifiziert und verglichen. Je höher die Wahrscheinlichkeit aufgrund der Gewichtung ist, desto eher ist der Pixel Teil des Hintergrunds und wird als dieser identifiziert. Schatten eines Objektes werden ebenfalls erkannt und in dem erzeugten Bild grau dargestellt. (15) (Abbildung 4.c)

Der dritte verfügbare Algorithmus trägt den Namen *BackgroundSubtractorGMG* und stammt ursprünglich aus einer interaktiven Kunstaustellung, welche die Position der Besucher verfolgte und dementsprechend Töne abspielte. (19) Neben einer statistischen Hintergrunderkennung verwendet dieser Algorithmus für jeden Pixel eine Segmentation nach Bayer. Dazu werden die ersten verfügbaren Bilder für eine Hintergrundmodellierung verwendet, welche nach und nach angepasst wird. Neu erkannte Objekte werden stärker als ältere Objekte gewichtet und das resultierende

Bild wird anschließend durch Opening und Closing, zwei Verfahren zur Entfernung von Rauschen, gefiltert. (15)

Wie in Abbildung 4 zu erkennen ist, funktioniert die Extraktion des Vordergrunds unabhängig von der Wahl des Algorithmus stellenweise recht gut, an anderen Stellen jedoch nicht so gut. Das Problem liegt in der Funktionsweise der Algorithmen, die nur in einem bewegten Bild funktionieren. Wird der Algorithmus mit demselben Bild wieder und wieder versorgt und es ist keine Veränderung in den Bildern zu sehen, wird der Vordergrund nicht mehr vom Hintergrund getrennt und die Person nicht mehr erkannt. Bewegt sich die Person zu schnell, sind auf dem erzeugten Bild Schlieren zu sehen (Abbildung 5), welche nicht mehr eindeutig zu einer Person zuordenbar sind.

Abbildung 5: BackgroundSubtractorMOG2 - Schlieren bei schnellen Bewegungen

Angenommen diese Vordergrundextraktion wäre erfolgreich gewesen, wäre immer noch die Positionserkennung der einzelnen Körperteile ein zu lösendes Problem. Dazu müssten für alle Körperteile Kriterien definiert werden, welche die Erkennung dieser Körperteile auf einem Kantenbild zulässt. Bei vielen verschiedenen Benutzern, die unterschiedlich groß sind, aus unterschiedlichen Blickwinkeln aufgenommen werden und unterschiedliche Körperproportionen haben, ist dies auf einem Kantenbild eher schwierig.

3.2.3. Performance

Ein weiteres großes Problem dieser Algorithmen ist, dass sich diese nur schwer parallelisieren lassen. Zusätzlich muss der Algorithmus immer auf neue Bilder der Kamera warten bevor er mit der Verarbeitung eines neuen Bildes beginnen kann. In einem Computer muss für diese Art von Berechnungen ein Kern des Prozessors verwendet werden, welcher eine deutlich höhere Single-Core-Performance als ein

Kern eines Grafikprozessors hat, von welchen es wiederrum eine große Anzahl auf einer Grafikkarte gibt. Die Verwendung eines solchen Kerns verringert die Durchlaufzeit des Algorithmus und erlaubt die schnellere Verarbeitung des nächsten Bildes. Die Anzahl der Prozessorkerne ist jedoch stark beschränkt und diese werden auch von vielen weiteren Anwendungen verwendet. Lastet schon zu Beginn der Verarbeitungskette ein Prozess einen Großteil der verfügbaren Rechenleistung des Prozessors aus, steht dem Rest umso weniger Rechenkapazität zur Verfügung. Der Simulator wird aber auch einen großen Teil der verfügbaren Rechenkapazität ausnutzen müssen, um realistische Simulationen und flüssige Bildraten erzeugen zu können. Dementsprechend überwiegen an dieser Stelle die Nachteile der Personenerkennung durch einen Hintergrundabzug und es muss eine weitere Methode für die Personenerkennung gefunden werden.

3.3. Neuronale Netze

In Diskussionen um Artificial Intelligence und Big Data fällt auch immer ein weiteres Stichwort: Neuronale Netze. Diese für Computer neue Art des Rechnens scheint auf magische Art und Weise zu funktionieren und zuverlässige Entscheidungen zu treffen, obwohl kein Entwickler die Entscheidungsfindung vollkommen nachvollziehen kann. Das Ziel hinter der Entwicklung von neuronalen Netzen ist die Nachempfindung des menschlichen Gehirns zur Entscheidungsfindung für Computer. So sollen von einem Computer möglichst menschliche Entscheidungen getroffen werden, die auf vorher gemachten Erfahrungen basieren. Außerdem sind diese Neuronen-Netze performant und können komplexe Gleichungssysteme lösen, ähnlich wie ein Mensch das aus dem Bauch heraus auch kann. Diese Art des Rechnens kann in vielen Gebieten eingesetzt werden, die bisher nur von Menschen bearbeitet werden konnten. Dazu zählen unter anderem Hinderniserkennung im Straßenverkehr, Klassifikation von Bildern oder das Wiedererkennen von gesuchten Objekten.

Neuronale Netze bestehen, wie es der Name bereits vermuten lässt, aus vielen Neuronen, die zu einem Netz zusammengesponnen sind. Ein Neuron, auch Unit genannt, nimmt Informationen aus der Umwelt oder von anderen Neuronen auf, trifft eine Entscheidung und gibt diese an verbundene Neuronen weiter. Mehrere Neuronen sind durch unterschiedlich stark gewichtete Kanten miteinander verbunden. Je höher die Gewichtung einer Kante ist, desto höher ist der Einfluss des sendenden Neurons auf das durch diese Kante verbundene und empfangende Neuron. Dabei wird das Ergebnis, auch Aktivitätslevel genannt, mit dem Kantengewicht multipliziert und als

Eingabe für das empfangende Neuron verwendet.

$$input_{ij} = a_j \, w_{ij}$$

a_j = Aktivitätslevel der sendenden Unit j
w_{ij} = Gewicht zwischen der sendenden (j) und der empfangenden (i) Unit

Wenn ein neuronales Netz „lernt“, werden letzten Endes nur die Gewichte der Kanten verändert. Um ein neuronales Netz trainieren zu können, müssen Lernregeln definiert werden, welche die Gewichtungsänderung beschreiben. Außerdem trainiert ein Netz schneller, wenn es die eigene Ausgabe mit einer gültigen Ausgabe vergleichen kann. Je besser die errechnete Ausgabe mit dem richtigen Ergebnis übereinstimmt, desto näher ist das neuronale Netz an einer optimalen Entscheidungsfindung und behält die aktuelle Gewichtung der Kanten bei. (20)

Die Verbindungen der einzelnen Neuronen ist für jedes neuronale Netz ein entscheidender Faktor. Je nachdem, wie die einzelnen Neuronen miteinander verbunden sind, kann das Netz zu unterschiedlichen Ergebnissen kommen. Prinzipiell werden Neuronen in einzelne Schichten, auch Layer genannt, unterteilt. Ein neuronales Netz besteht in der Regel aus mindestens drei Schichten:

1. **Input Layer**: Diese Schicht ist die erste Station jeder Information, die das neuronale Netz durchlaufen soll. Neuronen dieser Schicht nehmen die Eingabeparameter entgegen, gewichten diese mit ihren eigenen Gewichten und schicken den gewichteten Parameter an jedes Neuron der nächsten Schicht weiter.
2. **Hidden Layer**: Diese Zwischenschichten haben einen großen Einfluss auf die Verarbeitung der Informationen und sind im Normalfall der größte Bestandteil eines neuronalen Netzes. Die Anzahl der Hidden Layer ist dabei variabel und kann je nach Einsatzzweck angepasst werden. Mit steigender Anzahl der Layer steigt gleichermaßen der benötigte Rechenaufwand für einen Durchlauf des Netzes. Je mehr Hidden Layer in einem Netz vorhanden sind, desto tiefer wird das Netz bezeichnet. Die Neuronen der letzten Hidden Layer sind jeweils mit allen Neuronen der allerletzten Schicht, der Output Layer, verbunden.
3. **Output Layer**: Diese Schicht ist die letzte Schicht eines neuronalen Netzes. Jede berechnete Information wird dieser Schicht entnommen, sie dient also als Schnittstelle zur Außenwelt.

Über die Jahre hinweg wurden viele verschiedene Arten von neuronalen Netzten entwickelt und getestet. Die erste Version eines neuronalen Netzes sind die Perzeptoren. Ein Perzeptor besteht in seiner grundlegendsten Form aus nur einem einzigen Neuron, welches eine anpassbare Gewichtung und einen Schwellwert besitzt. Die heute meist genutzten Netze sind sogenannte Feedforward Netze. Informationen durchlaufen das Netz genau ein einziges Mal und jedes Neuron ist nur mit Neuronen der nächsten Schicht verbunden. Eine weitere Art von neuronalen Netzen sind rekurrente Netze. In diesen Netzen sind durch Rückkopplungen auch Verbindungen zu anderen Schichten als der nächsten Schicht möglich. Ein Neuron kann also mit sich selbst (direkte Rückkopplung), mit einem Neuron der vorherigen Schicht (indirekte Rückkopplung), mit einem Neuron der gleichen Schicht (seitliche Rückkopplung) oder mit jedem anderen Neuronen innerhalb des gesamten Netzes (vollständige Rückkopplung) verbunden sein. Rekurrente Netze kommen meistens bei sequentieller Informationsverarbeitung wie einer Handschrifterkennung oder Spracherkennung zum Einsatz. Eine weitere vorherrschende Art der neuronalen Netze sind die faltenden neuronalen Netze, auch Convolutional Neuronal Network (*CNN*) genannt. Diese Netze werden vorwiegend in der Bild- und Tonverarbeitung eingesetzt und führen in den versteckten Schichten eine Mustererkennung durch, welche das bereits erkannte Muster aus der jeweils vorherigen Schicht verfeinern. (21) Als Ausgabe kann das Netz jedoch nur feststellen, welche Objekte auf dem Bild zu sehen sind und eventuell auch noch grob einordnen, an welcher Stelle sich das Objekt befindet. Um einen genaueren Aufenthaltsort und eine genauere Abgrenzung von Objekten zu erhalten, werden Regional-CNNs (*R-CNN*) verwendet. Für diese Aufgabe wird das CNN mit einem kleineren Bildausschnitt versorgt, welches genauer untersucht werden soll. In diesem Bildausschnitt dominiert das zu erkennende Objekt und kann somit leichter abgegrenzt werden. Im Anschluss werden alle kleineren Bildausschnitte wieder zu einem Gesamtbild zusammengesetzt.

3.3.1. Neuronale Netze für Personenerkennung

Das Trainieren eines neuronalen Netzes ist keine einfache Aufgabe und benötigt neben massenhaft qualitativ hochwertigen Trainingsdaten auch eine Menge Rechenkapazität und Zeit. Deshalb kommt das Training eines eigenen Netzes für dieses Projekt nicht in Frage. Glücklicherweise ist das Problem der Personenerkennung weit verbreitet und spielt in immer mehr Anwendungsfällen eine entscheidende Rolle. Verschiedene Teams haben es sich zur Aufgabe gemacht,

solche Netze zu trainieren. Teilweise stehen diese Projekte im Rahmen eines Open Source Projektes zur freien Nutzung.

Projekte dieser Art entstehen oftmals im Rahmen von jährlichen Wettbewerben, welche es sich zum Ziel setzen, die bestmögliche Erkennung von Personen auf 2D-Bildern zu finden und das dahinter stehende Team dementsprechend zu belohnen. Die entsprechenden Ergebnisse werden anschließend auf großen Konferenzen wie der ICCV oder ECCV vorgestellt und ausgezeichnet. Beispiele für solche Wettbewerbe sind unter anderem die PoseTrack-Challenge (ICCV) oder der COCO Keypoint Detection Task (ECCV). Im Rahmen dieser Wettbewerbe gab es in den vergangenen Jahren viele verschiedene Einreichungen, die sich auf verschiedene Aspekte der Challenges konzentriert haben und dementsprechend unterschiedlich gut abschnitten.

Jede Challenge bietet allen Teilnehmern unterschiedliche Datensätze an, anhand derer die finale Bewertung der Einreichungen erfolgen. Neben der finalen Berwertung können die Datensätze auch für das Training der neuronalen Netze verwendet werden. Die Datensätze bestehen aus zehntausenden Bildern mit den dazugehörigen Keypoints und sind in den verschiedensten Alltagssituationen aufgenommen worden. Auf den Bildern sind zum Beispiel Personen zu sehen, welche Sport treiben, an einem Schreibtisch sitzen oder sozial interagieren. Somit bieten sie eine umfassende und ausreichend große Datenbank für das Training an. Beispiele für diese Datensätze sind der MPII- oder COCO-Datensatz.

R-CNN

R-CNNs haben im Allgemeinen die Aufgabe, zusammenhängende Regionen in einem Bild zu erkennen und diese voneinander zu trennen. Diese maskierten Regionen sind typischerweise unterschiedliche Objekte wie eine Person, ein Fahrrad oder ein weiteres Fahrzeug.

Wie in Abbildung 6 zu sehen ist, verbindet ein maskierendes R-CNN verschiedene Objekte und kann zusammengehörige Bestandteile erkennen. Das Problem bei der Verwendung eines solchen neuronalen Netzes liegt in der fehlenden Positionsangabe der einzelnen Körperteile. Als Ergebnis wird zwar die Region gefunden, in welcher sich die Person mit allen Körperteilen befindet, diese ist jedoch nicht nach Körperteilen aufgetrennt. Ein R-CNN müsste also erweitert werden, damit die Koordinaten der einzelnen Körperteile erkannt werden können. Dies ist theoretisch denkbar, würde jedoch weitere Rechenkapazität benötigen und einen weiteren Schritt in die

Erkennungs-Pipeline hinzufügen. Da es neben R-CNNs auch weitere Lösungen gibt, rückt diese Option vorerst in den Hintergrund.

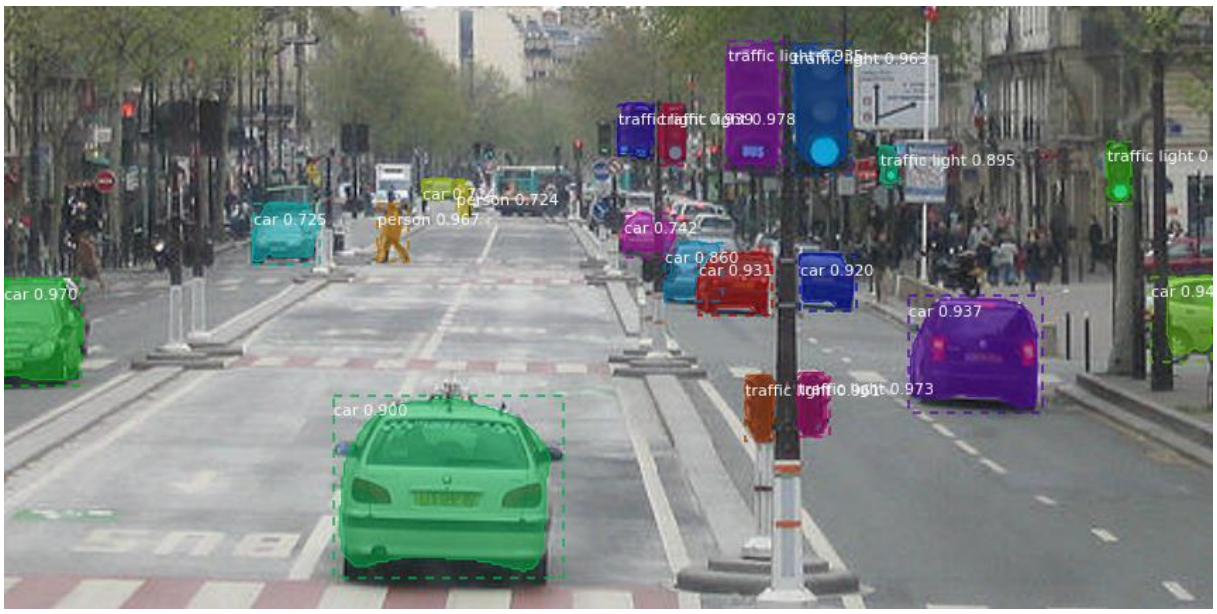

Abbildung 6: Beispiel eines Mask R-CNN im Straßenverkehr (22)

AlphaPose

Bei AlphaPose handelt es sich nicht um ein generelles Konzept wie bei R-CNNs, sondern um eine tatsächliche Implementierung. Das Ziel dieses Projektes ist die Erkennung von mehreren Personen auf 2D-Bildern in Echtzeit und basiert auf dem Paper „Regional Multi-Person Pose Estimation“. Im Inneren durchläuft AlphaPose zwei verschiedene Phasen für jedes zu untersuchende Bild: Im ersten Schritt werden mögliche Bereiche des Bildes gesucht, in denen sich eine Person befinden könnte. In diesen Bereichen werden im zweiten Schritt die konkreten Positionen der einzelnen Körperteile anhand verschiedener Algorithmen berechnet. Wichtig für diesen zweiten Schritt ist eine zuverlässige Erkennung einer Person im ersten Schritt. Sind nur kleine Teile eines Körpers in dem Bildausschnitt abgeschnitten, kann dies die Leistung und Zuverlässigkeit des zweiten Schrittes stark beeinflussen. (23)

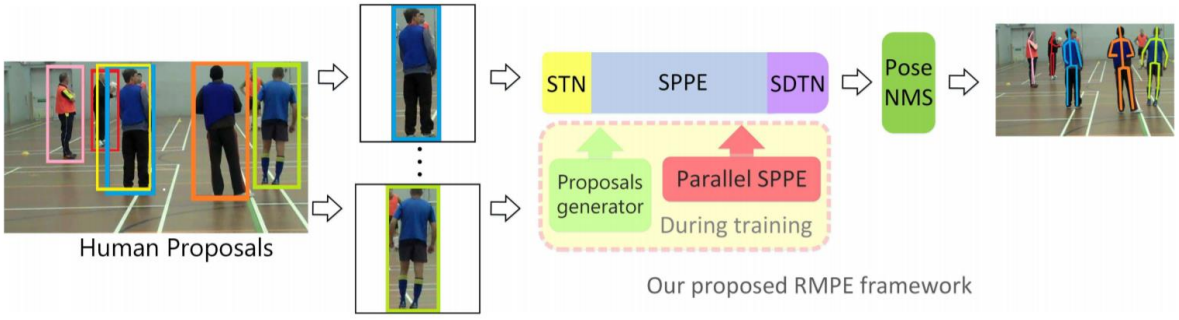

Abbildung 7: Personenerkennung mit AlphaPose (23 S. 3)

AlphaPose kann Personen zuverlässig und performant erkennen, stellt von Entwicklerseite aus jedoch keine öffentlich zugängliche API zur Verfügung. Alle Daten wie Bilder und Keypoints müssen vor der Weiterverarbeitung erst in ein externes Format wie JSON, XML oder Ähnliches umgewandelt, anschließend in dem ROS Modul importiert und wieder konvertiert werden. Um die Pipeline so kurz wie möglich zu halten und die Anzahl der möglichen Fehlerquellen klein zu halten, ist die Verwendung zwar prinzipiell möglich, aber mit höherem Aufwand verbunden.

OpenPose

Ähnlich wie AlphaPose kann auch OpenPose die Keypoints von mehreren Personen zuverlässig erkennen. OpenPose wurde im CMU Perceptual Computing Lab entwickelt, gewann 2016 einige Keypoint-Erkennungs-Challenges und wurde erstmals im April 2017 veröffentlicht. Das Entwicklungsziel ist neben der Erkennung der größeren Körperteile wie dem Oberkörper, Armen, Beinen und dem Kopf auch die Erkennung der Füße, einzelnen Finger, Augen und Ohren. Ebenfalls soll die Verwendung des neuronalen Netzes im Vergleich zu bereits bestehenden Lösungen wie AlphaPose deutlich vereinfacht werden. Dazu muss ein Entwickler nicht die gesamte Pipeline von der Aufnahme des Bildes bis hin zur Verarbeitung der erkannten Person selbst implementieren, sondern kann auf die Basisimplementierung von OpenPose in C++ oder Python zurückgreifen. Des Weiteren unterstützt OpenPose eine Vielzahl von Hardware-Implementierungen wie NVIDIAs CUDA-Kernen oder AMDs OpenCL Schnittstelle und ist somit auf vielen unterschiedlichen Systemkonfigurationen und Betriebssystemen lauffähig. (24 S. 6) OpenPose wird immer wieder um neue Funktionalitäten erweitert, wie an den Patch Notes zu erkennen ist. Dies bedeutet, dass auch in Zukunft immer wieder mit Verbesserungen gerechnet werden kann und das Projekt mit Updates versorgt wird. (25)

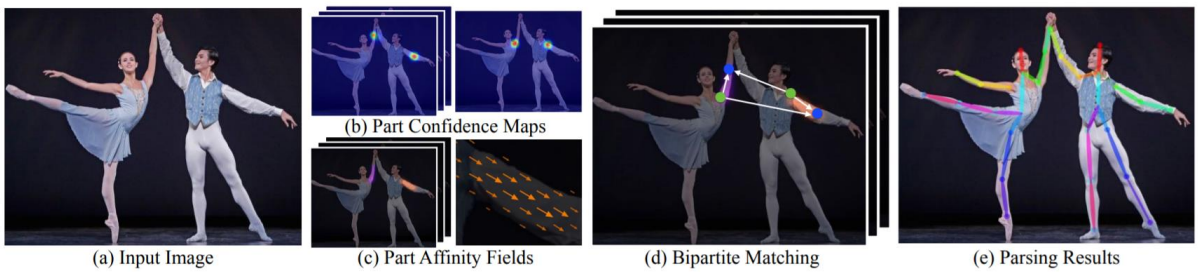

Abbildung 8: Personenerkennung mit OpenPose (24 S. 2)

Im Gegensatz zu AlphaPose verwendet OpenPose keine gesonderte Personenerkennung, welche kleinere Bildausschnitte mit einzelnen Personen zur

Verfügung stellt. Stattdessen setzt der Algorithmus auf eine Kombination aus mehreren Erkennungsschritten, wie in Abbildung 8 zu sehen ist.

Zu Beginn wird das Bild in mehrere Confidence Maps unterteilt. Jede Map konzentriert sich dabei auf ein anderes zu findendes Körperteil und beinhaltet Markierungen, an denen mit einer gewissen Wahrscheinlichkeit genau dieses Körperteil zu finden ist (farbige Markierung in Abbildung 8b). Sollten auf dem Bild mehrere Personen zu sehen sein, kann es dementsprechend vorkommen, dass auch mehrere Stellen in der gleichen Map markiert sind. Die verschiedenen Maps stehen zu diesem Zeitpunkt in noch keinerlei Relation zueinander. Anschließend werden für alle markierten Stellen der verschiedenen Maps Affinity Fields gesucht, die den typischen Verlauf des entsprechenden Körperteils darstellen sollen. Diese Felder zeigen auf, wo sich der Rest des Körperteils befindet. Wurde z.B. ein Ellenbogen erkannt, wird von diesem Ellenbogen aus der restliche Arm mit einer gewissen Wahrscheinlichkeit ermittelt. Sind nun alle verschiedenen Gelenke und die dazugehörigen anschließenden Körperteile erkannt, müssen diese zu einem Skelett zusammengesetzt werden. Dazu kommen mehrere bipartite Abgleiche zum Einsatz, welche die Skelette zuverlässig verbinden. Im Regelfall werden bei dieser Operation keine Skelette von mehreren Personen durchmischt. (24 S. 3-6)

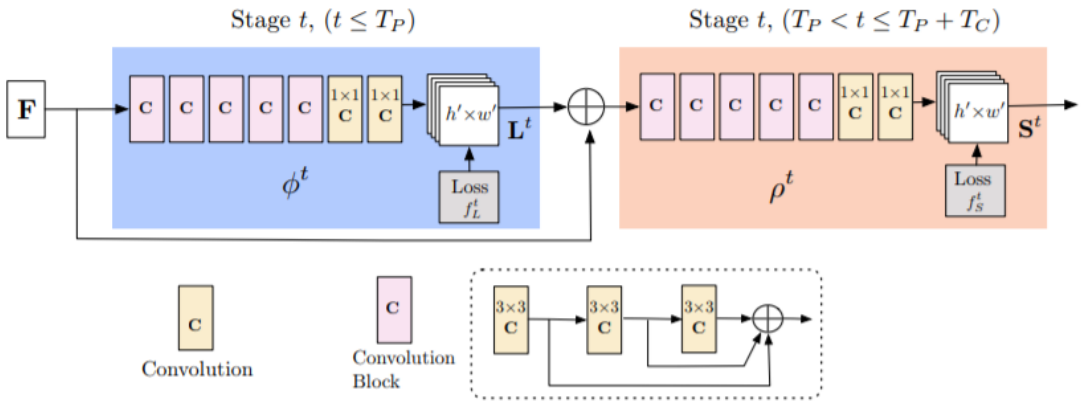

Abbildung 9: OpenPose - Aufbau des neuronalen Netzes für Part Affinity Fields (24 S. 3)

OpenPose basiert auf einem komplexen neuronalen Netz, welches im Inneren aus mehreren Teilnetzen besteht. Im ersten Schritt wird das farbige 2D-Bild an ein Feedforward Netz weitergeleitet, welches die Confidence Maps für die einzelnen Körperteile berechnet. Das in Abbildung 9 gezeigte neuronale Netz ist ein CNN und bestimmt im Anschluss an den ersten Schritt die Part Affinity Fields. Wie zu erkennen ist, bestehen die neuronalen Netze insgesamt aus einer Vielzahl von Neuronen, was sich auf die mögliche Performance auswirkt (siehe Kapitel 3.3.2 Performance).

Neben der ursprünglichen mit dem CMU Panoptic Dataset trainierten Version existieren für OpenPose zwei weitere Netze, welche dieselbe Architektur aber andere Trainingsdaten erhalten haben. Dazu zählen das COCO Dataset, Bestandteil der jährlichen COCO-Challenge, und das MPI Dataset. Die Wahl des Netzes wirkt sich unmittelbar auf die Performance, Anzahl und Genauigkeit der Keypoints aus und darf nicht vernachlässigt werden. Von offizieller Seite wird die Verwendung des eigenen Modells mit 25 Keypoints empfohlen, welches in Abbildung 10 zu sehen ist, da dieses die beste Performance und zuverlässigsten Ergebnisse liefert.

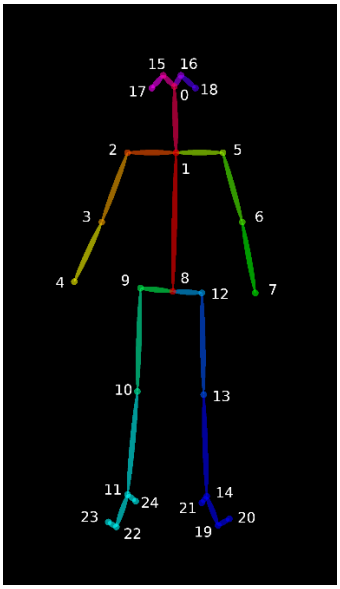

Abbildung 10: OpenPose - Skelett mit 25 Keypoints (25)

3.3.2. Performance

Wie in den Anforderungen definiert ist, sollte die gesamte Anwendung auf einem Computer lauffähig sein. In der Theorie können neuronale Netze auch durchaus alleine von einem Prozessor berechnet werden, jedoch hat kein Prozessor ausreichend Rechenkapazität, um ein neuronales Netz dieser Größe alleine in kurzer Zeit berechnen zu können. Die Kamerabilder sollten so schnell wie möglich verarbeitet und in Steuerungsbefehle umgewandelt werden, damit der Benutzer das Gefühl des Fliegens erhält und nicht mehrere Sekunden auf die Umsetzung seines neuen Befehles warten muss. Für diese Art der parallelen Berechnungen eignen sich GPU-Kerne hervorragend, da diese in einer großen Anzahl in einem Computer verbaut sind. Die verschiedenen Hersteller dieser GPUs, vornehmlich NVIDIA und AMD, haben in den vergangenen Jahren eigene Technologien für genau diese Art der Berechnungen entwickelt. Bei NVIDIA trägt diese Technik den Namen Compute Unified Device Architecture (*CUDA*) und ist sowohl auf Seite der Hardware als auch auf Seite der

Software seit einigen Jahren in den GPUs implementiert. Bei AMD heißt die Schnittstelle OpenCL und verfügt über ähnliche Fähigkeiten.

Bei ersten Tests mit OpenPose und der Verwendung der CPU zur Berechnung der Netze war eine Verzögerung von mehreren Sekunden pro Bild bemerkbar. Dies ist für eine Echtzeit-Anwendung inakzeptabel. Nach vielen Problemen mit der Installation von CUDA-Treibern auf Ubuntu konnte OpenPose jedoch vollkommen überzeugen. Für ein Frame benötigt OpenPose auf einer NVIDIA GTX 1060 im Schnitt 112ms Rechenzeit. Dies bedeutet im Umkehrschluss, dass pro Sekunde in etwa 8,9 Frames verarbeitet werden können. Dabei beträgt die Auslastung der GPU ca. 75%, was für die restlichen zu erledigenden Aufgaben wie das Rendern der Simulation noch etwas Spielraum lässt.

3.4. Implementierung

Nach dem Vergleich der verschiedenen Optionen sticht OpenPose mit all seinen Funktionen und Kompatibilitäten aus der Masse heraus. Das Projekt erfüllt alle Anforderungen (C++ oder Python-API, Verwendung der Grafikkarte, zuverlässige Ergebnisse, ständige Weiterentwicklung) und scheint gut für den geplanten Einsatzzweck zu passen. Als dennoch gute Alternative kommt AlphaPose in Frage, für welches jedoch eine eigene Schnittstelle implementiert werden müsste.

Die eigentliche Implementierung mit OpenPose in die ROS Umgebung stellt sich als einfach heraus. Sobald das Bild der Webcam mit Hilfe von OpenCV aufgenommen ist, wird es im gleichen Format an die Schnittstelle von OpenPose weitergeleitet. Die Schnittstelle gibt neben einer Liste mit allen berechneten Keypoints auch das Ursprungsbild, ergänzt um die Skelette der erkannten Personen, an den Aufrufer zurück. Das Anzeigen des Bildes ermöglicht dem Benutzer nachzuvollziehen, was der Computer sehen kann. Außerdem behält der Benutzer auf diese Weise die räumliche Orientierung, die ihm durch die aufgezogene Videobrille in den meisten Fällen verwehrt bleibt.

Sobald dem ROS Node die Keypoints vorliegen, wird aus der Menge der verfügbaren Personen die Person mit dem geringsten Abstand zur Bildmitte herausgefiltert. Als Referenzpunkt wird für diese Berechnung immer die Mitte der Hüfte (Abbildung 10, Keypoint 8) herangezogen. Dieser Vorgang setzt den Fokus auf die Person in der Bildmitte, falls auf einem Bild mehrere Personen erkannt werden. Dies erlaubt es Zuschauern am Rand des Sichtfeldes oder im Hintergrund der eigentlich fliegenden

Person zu stehen, ohne dass diese Personen die Kontrolle über die Drohne übernehmen können. Ein Problem bleibt dennoch bestehen, sollte eine weitere Person zwischen der Kamera und der fliegenden Person entlanglaufen. Diese Person kann aufgrund der fehlenden Tiefeninformation nicht erkannt und von der eigentlich steuernden Person unterschieden werden und unterbricht deshalb für einen kurzen Moment die Steuerung. Nachdem die Koordinaten gefiltert sind, werden diese auf einem ROS Topic in Form einer Message in einem einheitlichen Format zur Weiterverarbeitung veröffentlicht.

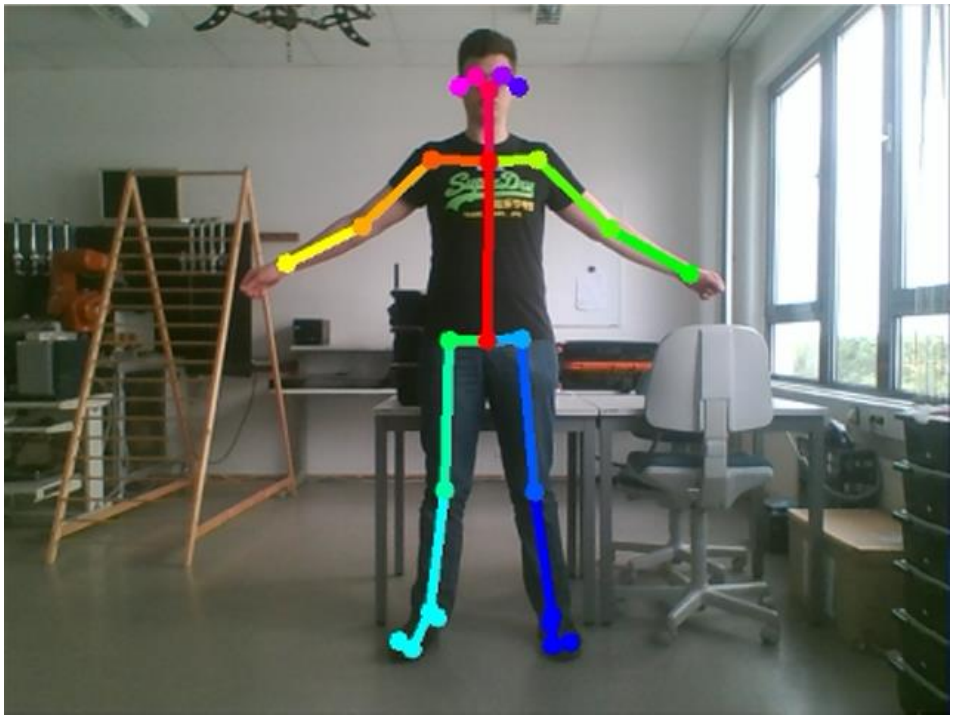

Abbildung 11: Personenerkennung mit OpenPose

4. Gestenerkennung

Im zweiten Schritt der Verarbeitungskette (Abbildung 1) wird die aktuelle Körperhaltung der erkannten Person untersucht und in allgemein gültige Steuerungsbefehle umgewandelt. Diese Befehle sind unabhängig von der vorhandenen Umgebung (echte Drohne oder Simulator) und stellen ein allgemeines Steuerungskonzept dar. Wie auch das Kapitel der Personenerkennung befasst sich dieses Kapitel ausschließlich mit der Auswertung von 2D-Koordinaten; eine Version für 3D-Koordinaten kann analog zu dieser Lösung erstellt werden und nach den gleichen Prinzipien arbeiten (siehe Kapitel 4.5 Implementierung).

4.1. Problemstellung

In der ursprünglichen Version von „I believe I can fly“ basierte die Körperbewegung für einen Flug nach vorne oder hinten auf dem Vor- bzw. Zurücklehnen des Oberkörpers. Diese Bewegung wurde durch die Differenz auf der Z-Achse zwischen Hüfte und Schultern erkannt. (1 S. 104-105) Durch die Verwendung einer Webcam, welche nur 2D-Bilder aufnehmen kann, entfällt jedoch die Tiefeninformation in den Bildern. Deshalb kann aus den aufgenommenen Bildern nicht zuverlässig erkannt werden, ob sich eine Person nach vorne oder hinten beugt bzw. welche Körperteile im Allgemeinen nach vorne oder hinten gestreckt werden. Aus diesem Grund müssen die Körperbewegungen für die einzelnen Steuerungsbefehle angepasst und teilweise auch neu definiert werden.

Die Erkennung der einzelnen Körperstellungen muss in diesem Schritt aus den aufgenommenen Bildern und den daraus berechneten Koordinaten der einzelnen Körperteile erfolgen können. Da die Bilder zu diesem Zeitpunkt bereits ausgewertet wurden, liefern diese für die Berechnung keine weiteren nützlichen Informationen und können ab diesem Schritt vernachlässigt werden. Interessant hingegen sind die Koordinaten der Körperteile, aus welchen sich Körperhaltungen ermitteln lassen.

4.2. Definition der erkennbaren Körperhaltungen

Bevor die Körperhaltung erkannt werden kann, müssen alle zu erkennenden Körperhaltungen definiert werden. Ein Teil der Haltungen kann aus den vorherigen Versionen übernommen werden. Die Steuerung basiert vollkommen auf der Haltung der Arme und ist unabhängig von der Haltung des restlichen Oberkörpers. Dies hängt unter anderem mit der fehlenden Tiefeninformation der Kamera zusammen, wie nachfolgend beschrieben ist.

Position halten

Die Geste, um die Position der Drohne zu halten, ist ein menschliches „T“. Dazu streckt der Pilot beide Arme auf Schulterhöhe mit einem Winkel von 90° seitlich vom Körper ab. Wird diese Geste eingenommen, soll die Drohne auf der Stelle stehen bleiben und sich auch nachfolgend an dieser Stelle halten. Sollte eine externe Kraft wie z.B. ein Windstoß die Drohne leicht bewegen, soll die Drohne aktiv gegen diesen Wind ansteuern und die Position trotzdem halten. Diese Geste ist als Standardgeste zu interpretieren und wird daher auch für die Kalibrierung verwendet. (siehe Kapitel 4.4 Kalibrierung)

Propellerdrehgeschwindigkeit

Die Höhe der Hände in Relation zur Höhe der Schultern bzw. des Nackens lässt die Propeller schneller oder langsamer drehen. Um die Propeller der Drohne schneller drehen zu lassen und somit mehr Auftrieb zu erzeugen, streckt der Pilot beide Arme gerade nach oben. Um die Propeller langsamer drehen zu lassen und somit weniger Auftrieb zu erzeugen, streckt der Pilot beide Arme nach unten. Diese Geste kann mit weiteren Gesten kombiniert werden.

Flug nach vorne

Um nach vorne fliegen zu können, muss der Pilot beide Arme nach vorne strecken. Der Winkel der Arme bei seitlicher Betrachtung spiegelt dabei die zu fliegende Geschwindigkeit wider. Streckt der Pilot seine Arme nach vorne und leicht nach unten, fliegt die Drohne nach vorne und verliert etwas an Flughöhe, verursacht durch die leicht langsamer drehenden Propeller. Streckt der Pilot seine Arme jedoch nach vorne und stark nach oben, fliegt die Drohne schnell nach vorne und gewinnt zusätzlich an Höhe. Werden die Arme auf Schulterhöhe nach vorne gestreckt, fliegt die Drohne ohne eine Änderung der Höhe mit mittlerem Tempo nach vorne.

Der Flug nach hinten wurde bewusst aus der Steuerung entfernt. Ein Pilot sollte immer sehen, wohin die von ihm gesteuerte Drohne fliegt, damit die Drohne nicht aus Versehen rückwärts in ein Hindernis hinein fliegen kann. Dies dient sowohl dem Schutz der Drohne als auch der Umgebung, in der sich die Drohne befindet.

Drehung um die eigene Achse

Um die Drohne zu drehen bzw. seitlich fliegen zu lassen, streckt der Pilot einen Arm nach oben und den anderen Arm nach unten. Wie bei einem Vogel gibt der nach unten gestreckte Arm bzw. Flügel dabei die Bewegungsrichtung an. Ist also der linke Arm

nach unten und der rechte Arm nach oben gestreckt, wirkt sich dies in einer Linkskurve der Drohne aus. Eine Oberkörper-Drehung des Piloten war in bisherigen Versionen für die Yaw-Drehung und der Winkel der Arme für den Winkel der Roll-Achse verantwortlich. Die Drehung des Oberkörpers kann in dieser Version nicht oder nur sehr unzuverlässig wie in der bisherigen Steuerung aus einem 2D-Bild erkannt werden. Das Problem an dieser Stelle ist die fehlende Tiefeninformation, welche eine Achse der Drohne unbenutzbar macht. Welche Achse benachteiligt wird, hängt von der konkreten Implementierung ab. Für diese Version sind die Yaw- und Roll-Achse der Drohne zusammengelegt und können nicht einzeln angesteuert werden. Stattdessen muss die Steuerungslogik mit diesem Problem umgehen und die Position der Armstellung berücksichtigen.

Hält die Drohne ihre Position und ist in keine Richtung geneigt, bewirkt eine Drehung auf der Yaw-Achse eine perfekte Drehung, sodass sich die Drohne auf der Stelle in die gewünschte Richtung dreht. Befindet sich die Drohne jedoch in einem Vorwärtsflug (Pitch-Achse steht nicht senkrecht zur Schwerkraft), bewirkt eine Drehung auf der Yaw-Achse ein ungewünschtes Verhalten und die Drohne dreht sich von der Richtung ihres Moments seitlich weg. Für unerfahrene Piloten kann dies im Extremfall den Absturz der Drohne bedeuten. Stattdessen muss für eine Drehung in einem Vorwärtsflug die Roll-Achse berücksichtigt werden, welche die Drehung auf der Yaw-Achse ausgleicht und das gewollte Verhalten einer einfachen Kurve wiederherstellt. Die Steuerung muss im Endeffekt einen guten Mittelweg zwischen der Stärke der Befehle auf der Roll- und Yaw-Achse finden. Im Optimalfall wird dabei die aktuelle Lage der Drohne berücksichtigt, um den Winkel der Roll-Achse abhängig von dem Winkel der Pitch-Achse zu steuern.

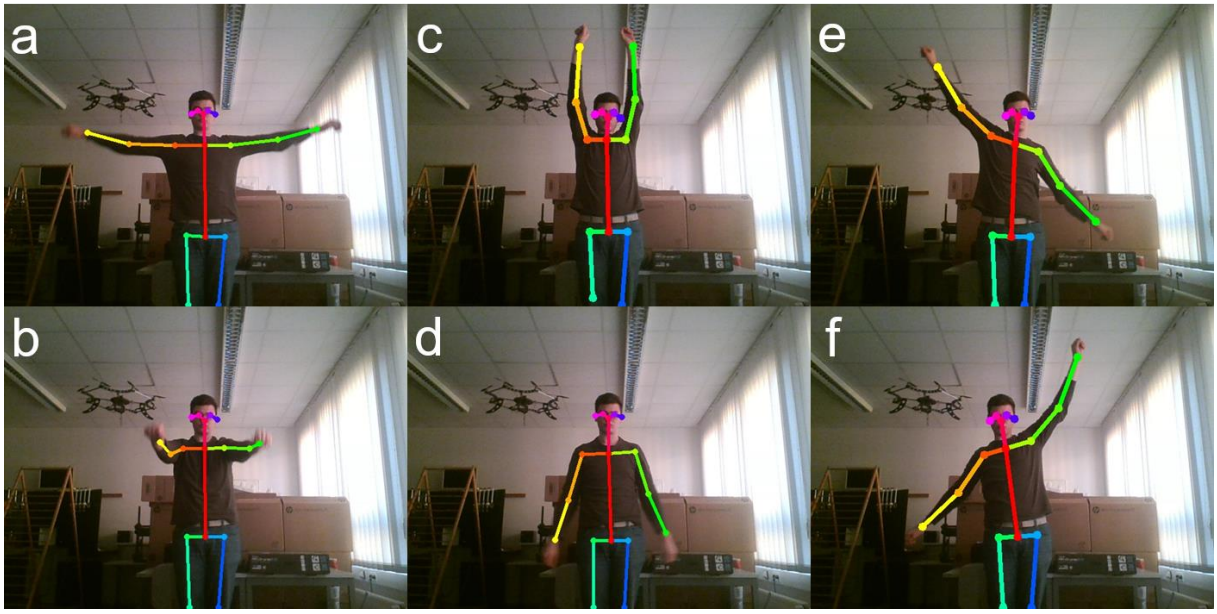

Abbildung 12: Steuerungsgesten a) Position halten b) Flug nach vorne c) Flug nach oben d) Flug nach unten e) Flug nach links f) Flug nach rechts

4.3. Analyse der Körperhaltung

Für die Analyse der Körperhaltung gibt es verschiedene Ansätze, welche wiederrum alle auf den Eigenschaften der vorliegenden Koordinaten beruhen. Diese Eigenschaften können unterschiedlich definiert werden und verschiedenen Zwecken dienen. Wie diese Eigenschaften durch mathematische Parameter ermittelt werden, hängt auch von der Personenerkennung des vorherigen Schrittes ab.

Wird in jedem neuen Bild die Erkennung der Körperstellungen komplett neu berechnet und beruht damit nicht auf der Auswertung vorheriger Bilder, sind die erkannten Körperstellungen als absolute Angaben anzusehen. Wie auch die Aufnahmen sind die Körperstellungen unabhängig von der Bildfolge und können somit ohne weitere Berechnungen in Flugbefehle umgewandelt werden. Werden bei der Auswertung jedoch auch die vorherigen Bilder berücksichtigt und nur die Differenz der Körperstellungen betrachtet, sind die erkannten Körperstellungen relativ anzusehen und können nur in Zusammenhang mit allen vorherigen Bildern zu einer absoluten Körperstellung umgewandelt werden. Dies bedeutet, dass es bei dem Fehlen eines Bildes oder einer zu geringen Aufnahmefrequenz zu ungewollten Differenzen kommen kann, da die Zwischenbilder nicht ausgewertet wurden, was sich wiederrum in ruckartigem Flugverhalten der Drohne bemerkbar macht. Deshalb ist der Einsatz eines absoluten Systems vorzuziehen, welches jedes erhaltene Bild unabhängig von anderen Bildern auswertet und in Flugbefehle umwandelt.

4.3.1. Wahl der Parameter

Unabhängig von der Auswertungsmethodik benötigen alle Auswertungsmethoden Parameter, anhand dieser die Körperstellung erkannt werden kann. Die Anzahl der Parameter beeinflusst die Genauigkeit der erkannten Stellung, hat jedoch auch Einfluss auf die benötige Rechenleistung und Komplexität der Berechnung. Deshalb ist die richtige Wahl der Parameter entscheidend, auf welcher die Berechnungen basieren. Für beide nachfolgenden Berechnungslogiken kommen mehrere Parameter in Frage. Alle Parameter beschreiben dabei jeweils die Beziehung von mehreren Körperteilen zueinander. Würden die Parameter keine Beziehung wiederspiegeln, könnten die Koordinaten direkt zur Erkennung der Geste verwendet werden.

Für die Erkennung aller benötigten Gesten (Abbildung 12) und die Kombination dieser Gesten werden insgesamt drei Parameter benötigt. Jeder Parameter beschreibt eine unterschiedliche Relation von mindestens zwei Körperteilen und setzt den Fokus auf die unterschiedlichen Gesten. Um die Berechnungen unabhängig von der Position der Person im aufgenommenen Bild zu machen, werden die Koordinaten vor der Weiterverarbeitung transformiert, sodass die mittlere Hüfte (Abbildung 10, Keypoint 8) der Ursprung des neuen Koordinatensystems ist. (siehe 4.5 Implementierung)

Winkel der Geraden von Hand zu Hand

Für diesen Parameter wird zwischen den beiden Händen des Piloten eine Gerade gespannt. Der Steigungswinkel dieser Geraden sagt dabei aus, inwieweit sich beide Hände in ihrer Höhe zueinander unterscheiden.

$$steigung = \frac{rechteHandY - linkeHandY}{rechteHandX - linkeHandX}$$

Hält der Pilot seine rechte Hand nach unten und die linke Hand nach oben, hat die Gerade zwischen beiden Händen eine stark negative Steigung. Hält der Pilot hingegen beide Hände auf Schulterhöhe, hat die Gerade eine Steigung von 0° und der Parameter hat den Wert 0,0. Die Steigung wird dabei relativ angegeben und passend skaliert. Eine Steigung von 50% bzw. 45° entspricht dabei einem Wert von 0,5. Die maximale Steigung beträgt theoretisch 100% bzw. 90° und somit einen Wert von 1,0. Der Parameter kann also jeden Wert zwischen -1 und 1 annehmen.

$$relativeSteigung = \frac{steigung}{2}$$

Abstand von Hand zu Hand

Der Abstand zwischen beiden Händen wird ebenfalls wie der Winkel über eine direkte Strecke zwischen beiden Händen berechnet. Die Berechnung an sich ist kein Problem und liefert den Abstand in Pixeln gemessen.

$$abstand = \sqrt{(linkeHandX - rechteHandX)^2 + (linkeHandY - rechteHandY)^2}$$

Der in Pixeln angegebene Wert kann aufgrund unterschiedlicher Bild-Auflösungen nur schwer weiterverarbeitet werden und muss deshalb in Relation zum maximalen Abstand der beiden Hände betrachtet werden. Der maximale Abstand muss berechnet werden, weshalb am Anfang der Benutzung die Armlänge und Schulterbreite kalibriert wird, wie in Kapitel 4.4 beschrieben ist.

$$relativerAbstand = \frac{abstand}{maximalerAbstand}$$

Gemittelte Höhendifferenz zwischen Händen und Schultern

Die gemittelte Höhendifferenz zwischen beiden Händen und beiden Schultern sagt aus, in wie weit beide Arme gleichmäßig nach oben oder unten ausgestreckt sind. Mit diesem Parameter lässt sich das Gleichgewicht zwischen beiden Armen ermitteln.

a) Ist der Wert positiv, sind beide Arme nach oben gestreckt.

b) Ist der Wert in etwa neutral, sind beide Arme auf Schulterhöhe oder gleichmäßig nach oben bzw. unten ausgestreckt.

c) Ist der Wert negativ, sind beide Arme nach unten gestreckt.

Wie auch der Abstand der Hände muss dieser Wert relativiert werden um weiterverwendet werden zu können. Dazu wird die gemittelte Distanz durch die kalibrierte Armlänge eines Armes geteilt und kann somit prozentual verwendet werden.

$$rechteDifferenz = rechteSchulterY - rechteHandY$$

$$linkeDifferenz = linkeSchulterY - linkeHandY$$

$$durschnittlicheDifferenz = \frac{rechteDifferenz + linkeDifferenz}{2}$$

$$relativeDifferenz = \frac{durschnittlicheDifferenz}{kalibrierteArmlänge}$$

4.3.2. Manuelle Auswertung

Ein naiver Ansatz zur Auswertung der festgelegten Parameter ist die manuelle Auswertung. Bei dieser Methode werden die vorliegenden Parameter durch manuell formulierte Abfragen überprüft und in Anweisungen kategorisiert. Als Ausgabe liefert diese Methode einen einzigen Flugbefehl für eine Kombination an Parametern. Die manuelle Auswertung erlaubt also in ihrer einfachsten Form keine Kombination von mehreren Befehlen. Des Weiteren ist die Entwicklung eines solchen Systems sehr aufwendig und wenig flexibel. Anpassungen an den aktuellen Piloten sind zwar theoretisch denkbar, aber schwierig in der Implementierung umzusetzen.

```
WENN steigung >= 0.5 UND relativerAbstand >= 0.75 UND relativeDifferenz == 0:
        FLIEGE LINKS.
WENN steigung <= -0.5 UND relativerAbstand >= 0.75 UND relativeDifferenz == 0:
        FLIEGE RECHTS.

// weitere Befehle abfragen

ANSONSTEN:
        HALTE POSITION.  // Die Standardgeste ist „Position halten“
```

Wie in dem Pseudocode zu sehen ist, müssten die Schwellwerte für jede Geste einzeln definiert werden und würden sich von Pilot zu Pilot unterscheiden. Ebenfalls wird der Code schnell unübersichtlich und mit steigender Komplexität schwieriger in der Wartung. Deshalb eignet sich dieser Ansatz nicht für eine komplexe Steuerung.

4.3.3. Fuzzy-Logik

Die 1965 von Zadeh beschriebene Fuzzy-Logik befasst sich mit der unscharfen Mengenlehre, welche auch unter dem Namen Fuzzy-Set-Theorie bekannt ist. Grundlage dieser Theorie ist, dass ein Wert auch nur teilweise zu einer Menge gehören kann. So gehören Hunde, Katzen oder Pferde ohne Frage in die Klasse der Tiere. Auch andere Lebensformen wie Bakterien weisen Eigenschaften auf, die sie in die Klasse der Tiere einordnen würden. Trotzdem weisen Bakterien auch weitere Eigenschaften auf, durch die sie auch in andere Klassen eingeordnet werden könnten; sie sind also nicht eindeutig klassifizierbar. (26 S. 338) Auch in der Mathematik und Informatik ist die Klassenbildung eine typische Problemstellungen. Ein Objekt, Wert, etc. soll aufgrund seiner Eigenschaften in eine Klasse eingeteilt werden, sodass Folgeentscheidungen basierend auf dieser Einteilung getroffen werden können. Der Vorteil der Fuzzy-Logik ist, dass die Zugehörigkeit zu den verschiedenen Klassen mit

natürlicher Sprache formuliert werden können und nicht durch eine Kaskade manueller Abfragen definiert werden müssen.

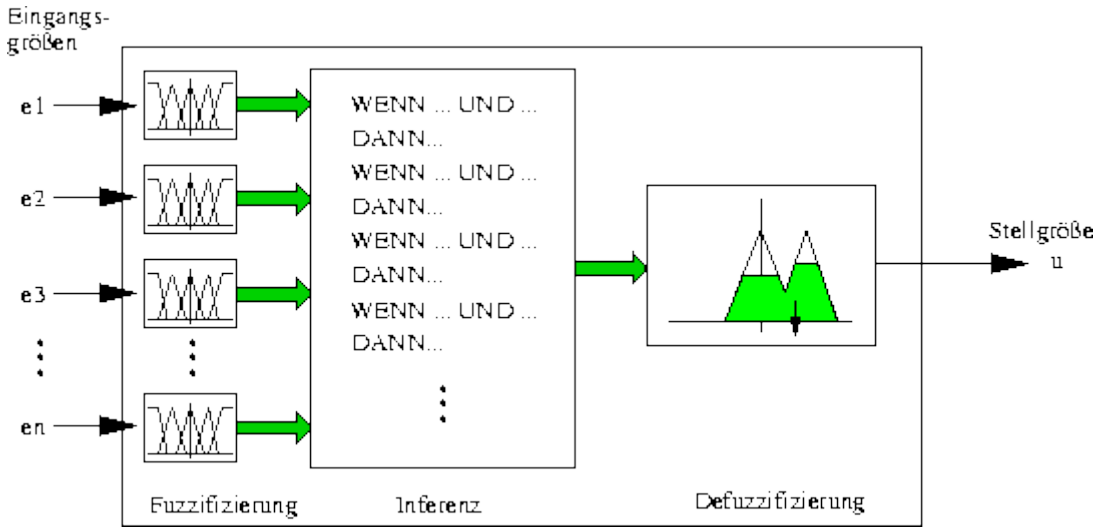

Abbildung 13: Schritte einer Fuzzy-Logik (27)

Eine Fuzzy-Logik besteht aus drei Bestandteilen, welche scharfe Eingabewerte in eine konkrete Anweisung in Abhängigkeiten zu allen Eingabeparametern umwandeln. Im ersten Schritt, der Fuzzifizierung, werden die scharfen Eingabewerte in natürlichsprachliche Werte umgewandelt. Diese natürlichsprachlichen Werte können mit Hilfe einer Inferenzmaschine anhand natürlichsprachlich formulierter Regeln in natürlichsprachliche Handlungsanweisungen umgewandelt werden. Im letzten Schritt, der Defuzzifizierung, werden die Handlungsanweisungen defuzzifiziert und in eine konkrete Handlung umgewandelt, welche ausgeführt werden kann. (27)

Auch in diesem Projekt sollen scharfe Eingabewerte, nämlich die Koordinaten der einzelnen Körperteile, in klare Flugbefehle umgewandelt werden. Damit die Schwellwerte nicht manuell gepflegt werden müssen, wie dies bei der manuellen Auswertung der Fall ist, soll auf natürliche Sprache zurückgegriffen werden. Dies vereinfacht die Programmierung und Wartung der Steuerung um einiges. Deshalb kommt für dieses Projekt eine Fuzzy-Steuerung zum Einsatz.

4.4. Kalibrierung

Bei immer wieder wechselnden Benutzern wird schnell klar, dass jede Person andere Körperproportionen hat. Neben der Größe der Person spielt auch die Länge der Arme eine tragende Rolle bei der Steuerung durch Gesten. Hinzu kommt variabler Abstand zwischen der Kamera und der Person, welcher die Größe der Person in einem aufgenommenen Bild beeinflusst. Diese Tatsachen erfordern in der Realität eine

Kalibrierung des gesamten Systems, sodass sich das System an den aktuellen Benutzer anpassen kann.

Bei der Kalibrierung werden verschiedene Parameter gemessen, welche nachträglich in die Berechnungen miteinfließen können und somit die Steuerungslogik an den Benutzer anpassen. Dazu stellt sich der Benutzer in die „Position halten“-Stellung für drei Sekunden mittig in das Kamera-Bild. In diesen drei Sekunden sollte sich die zu kalibrierende Person so wenig wie möglich bewegen und immer alle benötigten Körperteile sichtbar halten. Dieser Vorgang stellt sicher, dass die Person die Kalibrierung bewusst gestartet hat und nicht nur aus Versehen das Programm gestartet hat. Des Weiteren hat das System so die Möglichkeit, alle Voraussetzungen erneut zu prüfen und die benötigten Parameter zu messen.

Neben der durchschnittlichen Armlänge wird ebenfalls der Abstand beider Schultern zueinander gemessen und gespeichert. Beide Parameter werden für die Berechnung der verschiedenen Steuerungsparameter benötigt (siehe 4.3.1 Wahl der Parameter) und können im Nachgang jederzeit abgerufen werden.

Die Kalibrierung sollte für jeden Benutzer zurückgesetzt werden, damit auch die nachfolgende Person die bestmögliche und genauste Steuerung erreichen kann.

4.5. Implementierung

Bevor die erhaltenen Koordinaten der Körperteile für weitere Berechnungen verwendet werden können, muss die Richtigkeit dieser kontrolliert werden. Das neuronale Netz gibt für jedes Koordinatenpaar an, wie sicher es sich bei deren Bestimmung ist. Es kann also passieren, dass die Sicherheit z.B. einer Schulter nur 10% beträgt. Die Wahrscheinlichkeit einer Fehlerkennung ist an dieser Stelle einfach zu groß, weshalb solche Erkennungen herausgefiltert werden müssen. Andernfalls könnten unklare Klassifizierungen zu ungewollten Flugbefehlen führen, welche möglichweise einen realen Schaden mit sich bringen.

Die Mindestwahrscheinlichkeit, mit der ein Koordinatenpaar erkannt worden sein muss, ist aus praktikablen Gründen auf 70% festgesetzt. Dieser Wert gibt dem neuronalen Netz etwas Spielraum, falls z.B. schlechte Lichtverhältnisse oder schlechte Bedingungen im Allgemeinen herrschen, und gibt trotzdem eine ausreichende Sicherheit. Eine Optimierung an dieser Stelle ist, dass nicht jeder Keypoint mit dieser hohen Wahrscheinlichkeit gefunden werden muss. Körperteile, die nicht direkt an der Steuerung beteiligt sind (Beine, Füße, Kopf), müssen diese Wahrscheinlichkeit nicht

erreichen und dürfen sogar komplett fehlen. Die für die Steuerung benötigten Keypoints sind die mittlere Hüfte, beide Arme (inklusive der Schultern, Ellenbogen und Handgelenke) und der Nacken (siehe Abbildung 10).

Bevor die überprüften Koordinaten zur Berechnung der Körperstellung verwendet werden können, müssen diese noch in ein einheitliches System transformiert werden. Dazu wird die Mitte der Hüfte als Koordinatenursprung der Person interpretiert. Dies erlaubt weiteren Berechnungen in den darauffolgenden Schritten, immer mit relativen Koordinaten und unabhängig von der Auflösung der Kamera zu arbeiten. OpenCV setzt den Ursprung eines Bildes standardmäßig in der linken oberen Ecke.

Für alle Keypoints, mit Ausnahme der Hüfte, wird deshalb folgende Berechnung durchgeführt, welche die Koordinaten anhand der Hüfte neu ausrichtet. Die Hüfte selbst erhält im Anschluss die Koordinaten (0 | 0).

$$neuX = altX - huefteX$$

$$neuY = huefteY - altY$$

Nachdem die Koordinaten geprüft und transformiert wurden, werden die für die Steuerung benötigten Parameter berechnet (siehe 4.3.1 Wahl der Parameter). Die fertig umgewandelten Parameter können nun als Eingabe für den Fuzzy-Controller verwendet werden.

Der Fuzzy-Controller benötigt neben den drei beschriebenen Parametern keinerlei weitere situationsabhängige Informationen. Als Ausgabe liefert der Fuzzy-Controller drei verschiedene Parameter, welche im Anschluss an den nächsten ROS Node, die Drohnen- bzw. Simulator-Steuerung, weitergeben werden.

Forward

- Aufgabe: Beschreibt Neigung auf Pitch-Achse
- Wertebereich: [0, 100] → [keine Neigung, leichte Neigung, stark Neigung]

Turn

- Aufgabe: Beschreibt die seitliche Drehung, Kombination Roll- und Pitch-Achse
- Wertebereich: [-100, 100] → [stark links, keine Drehung, stark rechts]

Throttle

- Aufgabe: Beschreibt die Regulierung der Propellerdrehgeschwindigkeit
- Wertebereich: [-100, 100] → [absenken, Höhe halten, aufsteigen]

Für die Zuteilung der Werte in Klassen werden sowohl die Eingangs- als auch die Ausgangsparameter in mehrere Klassen unterteilt. Parameter mit einem negativen

Wertebereich sind in der Regel in fünf Klassen unterteilt: [stark negativ, negativ, neutral, positiv, stark positiv]; Parameter mit lediglich positivem Wertebereich in vier Klassen: [kein, niedrig, mittel, hoch]. Des Weiteren sind für jeden Ausgabeparameter Regeln definiert, welche die Ausgabeparameter anhand der Eingabeparameter bestimmen. Diese Regeln sind in einer ähnlichen Sprache im Vergleich zur natürlichen Sprache formuliert und unterstützen logische Verknüpfungen wie UND (&), ODER (|) oder NICHT (~). Die Implementierung wurde mit SciKit Fuzzy (*skfuzzy*) vorgenommen, einer Python-Bibliothek für Fuzzy-Logik. Eine Regel für das Absenken und Landen der Drohne hat folgendes Format:

```
rule = Rule(
        (self.hand_distance['none'] | self.hand_distance['low'] |
        self.hand_distance['medium'])
        & self.hand_to_shoulder_distance['vnegative'],
        self.throttle['fast_down'],
        'fast_throttle_down'
)
controller.addrule(rule)
```

Jede Regel besteht aus einer Bedingung, einer Folge und einer optionalen Benennung. Wie in der Regel zu sehen ist, muss der Pilot für die Verlangsamung der Propellerdrehgeschwindigkeit (throttle hat Wert fast_down) beide Hände nah beieinander (keine, geringe oder mittlere hand_distance) halten und die Hände im Vergleich zur Schulter relativ weit unten halten (sehr geringe hand_to_shoulder_distance). Je besser diese Bedingung erfüllt sind, desto eher wird die Kombination der Eingabeparameter als Klasse throttle in der Ausprägung fast_down klassifiziert. Die Kombination mehrerer solcher Regeln erlaubt die genaue Definition aller Körperhaltungen und wandelt diese entsprechend in numerische Parameter um.

Als Ausgabe liefert der Fuzzy-Controller für alle drei Ausgabeparameter einen Zugehörigkeitswert. Kombiniert der Benutzer mehrere Befehle wie z.B. eine Vorwärtsbewegung und eine Linkskurve, werden zwei der drei Ausgabeparameter stärker aktiviert. In dem vorangehenden Beispiel würde der Parameter *Turn* einen leicht negativen Wert (Linkskurve), der Parameter *Forward* einen positiven Wert (Vorwärtsflug) und der Parameter *Throttle* einen neutralen Wert (Höhe halten) zugeordnet bekommen.

Die Ausgabeparameter des Fuzzy-Controllers sind noch zu allgemein und könnten von verschiedenen ROS Nodes unterschiedlich interpretiert werden. Deshalb werden

diese drei Parameter nochmals in eindeutige Befehle transformiert. Mögliche Befehle sind `THROTTLE_UP`, `THROTTLE_DOWN`, `TURN_RIGHT`, `TURN_LEFT`, `FORWARD` und `HOLD`. Diese Befehle werden in Abhängigkeit zu der Ausgabe des Fuzzy-Controllers kombiniert und in einen Gesamtbefehl zusammengesetzt.

Übersteigen die Fuzzy-Parameter einen definierten Schwellwert, werden die entsprechenden Flugbefehle in den Gesamtbefehl mit der passenden Ausprägung übernommen. Erkennt der Fuzzy-Controller z.B. die Vorwärtsgeste, wird der numerische Wert des `FORWARD` Parameters um den Ausgabewert *Forward* des Fuzzy-Controllers erhöht. Da die Werte alle prozentual interpretiert werden, können diese ohne weiteres addiert werden. Auch die anderen Parameter des Fuzzy-Controllers beeinflussen die Ausgabe-Parameter nach fest definierten Regeln. Sollte keine Geste erkannt werden, wird die Ausgabe `HOLD` mit 100% erzeugt. Nimmt ein nachfolgender Node einen beliebigen Wert im Parameter `HOLD` wahr, darf er keinerlei andere Operationen als die Position zu halten ausführen. Dies kann technisch seitens der Gestenerkennung nicht kontrolliert werden, ist aber ein definierter Grundsatz, welcher zwingend eingehalten werden sollte.

Die Kombination der transformierten Anweisungen wird als letzte Aktion auf einem ROS Topic veröffentlicht und kann von den nachfolgenden Nodes verwendet werden.

Die Personenerkennung erreicht auf dem vorliegenden Laptop mit einer verbauten NVIDIA GTX 1060 in etwa 9 FPS (siehe 3.3.2 Performance). Für die Berechnung der Gesten wird kaum Rechenkapazität benötigt, weshalb die Framerate auf den durchschnittlichen FPS gehalten werden kann. Werden dem Benutzer die berechneten Skelette der erkannten Personen angezeigt, scheint das Video aus Sicht des Benutzers zu ruckeln. Dies liegt daran, dass Menschen ca. 25 oder mehr FPS benötigen, bis sie eine Bildfolge als flüssiges Video wahrnehmen. Die erreichten 9 FPS sind aber weniger als die Hälfte der benötigten FPS, weshalb das Video nicht flüssig erscheint. Das Problem kann durch das Ausblenden des Videos oder dem Anzeigen der echten Kameraaufnahmen, welche mit 30 FPS geschossen werden, umgangen werden. In letzterem Fall können die erkannten Skelette als Overlay auf die Aufnahmen der Webcam gelegt aufgelegt werden. Dann ruckeln nur die erkannten Skelette und nicht das gesamte Video, was allerdings auch für Verwirrung sorgen kann.

5. Benutzerausgabe

Der letzte Schritt der Verarbeitungskette (Abbildung 1, Schritte 4-6) besteht aus der Umwandlung der klassifizierten Gesten in konkrete Steuerungsbefehle und das Anzeigen eines Ergebnisses in Form eines aufgenommenen oder generierten Bildes aus Sicht der Drohne an den Benutzer. Dazu werden die allgemein gültigen Fluganweisungen des vorherigen ROS Nodes in plattformabhängige Anweisungen umgewandelt und an die entsprechenden Instanzen gesendet. Wie in den technischen Anforderungen beschrieben, soll eine Drohnensteuerung für die AR.Drone 2.0 des Herstellers Parrot und eine Simulatorsteuerung für AirSim implementiert werden.

5.1. Steuerung des Simulators

Da die Bedingungen für das Fliegen einer echten Drohne nicht immer erfüllt sind, ist die Möglichkeit der Ansteuerung eines Simulators sehr wichtig. Wie bereits evaluiert kommt der Simulator AirSim zum Einsatz, da er sowohl grafisch als auch Schnittstellen-technisch sehr gut für dieses Projekt geeignet ist. AirSim bietet für die Integration in verschiedene Projekte eine C++ und eine Python-Schnittstelle mit ähnlicher Funktionalität an. Da auch das restliche Projekt auf Python basiert, wird auch die Schnittstelle zwischen AirSim und Python verwendet werden.

AirSim hat an sich nur eine einzige Voraussetzung, damit die Schnittstelle funktioniert: Auf dem System muss ein UE4 Projekt ausgeführt werden, welches das AirSim Plugin integriert hat. Ist diese Bedingung erfüllt, kann sich die Python-Schnittstelle automatisch mit der laufenden Instanz des Simulators verbinden. (12)

An dieser Stelle muss die Kalibrierung wie in Kapitel 4.4 beschrieben stattfinden. Dies stellt sicher, dass der Benutzer wirklich den Simulator aktivieren möchte und zum Abflug bereit ist.

Um die Steuerung von AirSim zu aktivieren, muss sich ein Programm bei der Schnittstelle anmelden und die Kontrolle über die simulierte Drohne anfragen, nachdem das UE4-Projekt gestartet wurde. Dies erlaubt eine höhere Sicherheit, da AirSim die Verbindung erst bestätigen muss. So können nicht mehrere Quellen gleichzeitig die Kontrolle übernehmen und sich gegenseitig beeinflussen.

```
client = airsim.MultirotorClient()
if client.confirmConnection():
        // Flug-Anweisungen
else:
        // Verbindung wurde abgelehnt
```

Ist die Verbindung zu AirSim hergestellt und der Benutzer kalibriert, kann die Schnittstelle über verschiedene Befehle angesteuert werden. Bei der Art der Befehle gibt es prinzipiell zwei Unterschiede. Synchrone Befehle werden direkt ausgeführt und erlauben keine Änderung der Befehle, bis der Befehl zu Ende ausgeführt wurde. Dies bedeutet gleichzeitig auch, dass die Kontrolle über die Drohne komplett abgegeben wird und kein Eingriff in die aktuelle Flugbahn möglich ist. Diese Befehle stammen noch aus einer älteren Version von AirSim und sind in neueren Versionen aus Sicherheitsgründen deaktiviert. Stattdessen sind nun asynchrone Befehle möglich, welche bei Erhalt zwar sofort gestartet werden, sich jedoch gegenseitig beeinflussen und abbrechen können. Erhält der Simulator während der Ausführung eines anderen Befehls einen neuen Befehl, verwirft dieser den alten Befehl und führt den neuen Befehl aus. Dies erlaubt eine angemessene Steuerung und z.B. bei der Erkennung eines Hindernisses Eingriff in die aktuelle Flugbahn.

Die Schnittstelle bietet eine Vielzahl von Steuerungsmöglichkeiten an. So können feste Flugbahnen, ein Zielort mit beliebiger Flugbahn oder eine Änderung der Winkel der Drohne als neuer Befehl festgelegt werden. Einige dieser Befehle bieten zusätzlich die Möglichkeit, die Dauer der Ausführung für diesen Befehl festzulegen. Soll die Drohne von A nach B innerhalb von 5 Sekunden fliegen, passt der Simulator die Geschwindigkeit der Drohne passend zu der zu fliegenden Distanz in der Ausführungszeit an. Erreicht den Simulator innerhalb der 5 Sekunden ein neuer Befehl, wird der alte Befehl abgebrochen und der neue Befehl ausgeführt.

Dieses Verhalten kann sich aus Sicht der Sicherheit zu Nutze gemacht werden. Der Node sendet an den Simulator immer nur Befehle, welche von kurzer Dauer sind. Bricht die Verbindung zwischen dem Node und dem Simulator ab, wird die Drohne innerhalb kurzer Zeit in ihr definiertes Standardverhalten zurückkehren, da sie von der Schnittstelle keine neuen Befehle erhält. Würden die Befehle hingegen mit einer sehr langen Ausführungszeit gesendet werden, würde ein Abbruch der Verbindung nicht zum Abbruch des Befehls führen und die Drohne würde unkontrolliert weiterfliegen.

Deshalb ist in dem Node ein Puffer für Befehle eingebaut, welcher immer den aktuellsten Befehl speichert. Wird über das abgehörte ROS Topic ein neuer Flugbefehl von der Gestenerkennung erhalten, wird dieser Befehl in ein spezielles Format für den Simulator konvertiert und in den Puffer geschrieben. In regelmäßigen Abständen wird der Befehl aus dem Puffer gelesen und an die Drohne gesendet. Der Simulator startet

die Ausführung des Befehls mit der gleichen Dauer wie die Befehlsaktualisierung und wartet anschließend auf neue Befehle. Sollte innerhalb eines kleinen Zeitfensters kein weiterer Befehl von der Schnittstelle folgen, geht die Drohne automatisch in den „Position halten“-Modus über und nimmt vorerst keine weiteren Befehle über die Schnittstelle an, bis die Verbindung erneut aufgebaut und von beiden Seiten bestätigt wird.

Die Konvertierung wandelt die erhaltenen allgemeinen Befehle in konkrete Winkelangaben für die Roll-, Pitch- und Yaw-Achse sowie den Throttle-Wert um. Durch die Umwandlung kann auf das spezifische Flugverhalten des Simulators reagiert werden und die Befehle werden in angemessene Werte umgewandelt. Wird ein Befehl für einen Kurvenflug empfangen (Parameter `TURN_LEFT` oder `TURN_RIGHT` sind gesetzt), werden sowohl an dem Roll-Winkel als auch an dem Yaw-Winkel Änderungen vorgenommen, um die Kurve sauber fliegen zu können. Würde nur ein Winkel geändert werden, würde sich die Drohne aus Sicht der Nutzers ungewollt verhalten und die Drohne würde mit der Kamera nicht mehr in die Flugrichtung zeigen (siehe 4.2 Definition der erkennbaren Körperhaltungen, Drehung um die eigene Achse). Ebenfalls kann so die Skalierung der Befehle an die Skalierung der Drohne angepasst werden. Dafür kann eine allgemein gültige Formel verwendet werden, welche einen prozentualen Wert in einem gegebenen Wertebereich in einen anderen Wertebereich übertragt und dabei den prozentualen Wert beibehält.

$$skalierterWert = \frac{(wert - altesMin) * (neuesMax - neuesMin)}{(altesMax - altesMin)} + neuesMin$$

Gegeben sei der konkrete Wert 50 in dem Wertebereich [0, 100]. Der neue Wertebereich hat die Grenzen [0, 200], somit wäre der skalierte Wert 100.

$$skalierterWert = \frac{(50 - 0) * (200 - 0)}{100 - 0} + 0 = \frac{50 * 200}{100} = 100$$

Mit dieser Methode können die Winkelangaben passend zum Simulator an das Flugverhalten angepasst werden, sodass die Steuerung weder über- noch untersteuert. In der konkreten Implementierung werden die Ober- und Untergrenzen der erlaubten Wertebereiche angepasst. Dazu wurden mehrfach verschiedene Grenzwerte auf ihr Flugverhalten getestet und sind nun in die Steuerungslogik implementiert.

Damit der Benutzer auch sieht, wohin die Drohne im Simulator fliegt, muss er natürlich ein Kamerabild angezeigt bekommen. Dazu gibt es zwei verschiedene Möglichkeiten. Die erste und einfachste Möglichkeit ist es, das Bild des Simulators auf die Videobrille, welche wie ein weiterer Monitor funktioniert, zu ziehen. So sieht der Benutzer ohne große Umwege und Verzögerungen, was in dem Simulator vor sich geht. Als Alternative kann die Schnittstelle des Simulators genutzt werden, um Bilder der verschiedenen Kameraperspektiven zu erhalten. Diese Bilder können dann in regelmäßigen Abständen vom Simulator abgefragt werden und dem Benutzer angezeigt werden. Ein Problem an der Stelle ist, dass der Laptop schon mit dem Betreiben des Simulators und der Gestenerkennung leistungstechnisch an seine Grenzen kommt und jede weitere Verzögerung der Bildanzeige zu einem deutlich spürbaren Delay führt. Deshalb ist es am besten, dem Benutzer den Simulator, so wie er ist, zu zeigen und das Bild nicht zusätzlich durch ein ROS Node weiterzuleiten.

Damit der Benutzer nicht das Gleichgewicht verliert und die Orientierung im Raum behalten kann, wird ihm neben dem Bild des Simulators noch das aufgenommene Bild der Kamera gezeigt. Dieses Bild hilft vielen Nutzern weiter, die üblichen Effekte der virtuellen Realität wie Übelkeit oder Orientierungsverlust zu umgehen. Außerdem kann der Benutzer so besser nachvollziehen, was der Computer genau sieht und sich besser im Bild positionieren.

Abbildung 14: Benutzerausgabe bei Verwendung des Simulators

Ein Problem der Ausgabe ist die mangelnde Performance des Systems. Einzeln betrieben reicht die Leistung der Grafikkarte bei weitem aus und die Komponenten können sehr gut betrieben werden. Werden jedoch alle Komponenten gleichzeitig betrieben, ist die verfügbare Leistung einfach zu gering. Der vorher 60 FPS erreichende Simulator hat in einem solchen Szenario deutlich weniger Rechenkapazität der GPU frei und kann daher nur langsamere Bildfolgen mit ca. 20 FPS erzeugen. Dieser Wert liegt knapp unter der Grenze, ab welcher eine Bildfolge als flüssig empfunden wird. Trotzdem kann der Simulator benutzt werden, da sich ein Pilot im Regelfall nach einer kurzen Eingewöhnungszeit an die fehlenden Bilder gewöhnt und sich trotzdem auf das Fliegen konzentrieren kann.

Die geringen Bildraten könnten auf mehrere Arten angehoben werden:

1. Steigerung der Systemleistung: Durch das Hinzufügen einer weiteren GPU oder den Austausch der vorhandenen GPU mit einer stärkeren GPU steht dem gesamten System mehr Rechenkapazität zur Verfügung. Im Falle eines Laptops ist dies allerding weniger praktikabel, da die GPU bei den meisten Laptops fest verschweißt ist und deshalb nicht einzeln ausgetauscht werden kann. Natürlich ist der Austausch der Hardware auch eine Frage des Geldes, welche vom zur Verfügung stehenden Budget abhängt.
2. Senkung der Grafikqualität: Der Simulator läuft in dem konkreten Beispiel mit einer leicht geringeren Auflösung als Full-HD, was 1920x1080 Pixeln entsprechen würde. Durch die Senkung der Anzahl der zu berechnenden Pixel wird weniger Rechenkapazität für ein einzelnes Bild benötigt, weshalb in Folge dessen bei gleicher verfügbarer Rechenkapazität mehr FPS erzeugt werden können. Die Auflösung hat jedoch auch direkten Einfluss auf das Immersionsgefühl des Nutzers, wie bei Verwendung des tum_simulator zu sehen ist (siehe Abbildung 2) und sollte deshalb gut abgewogen werden.
3. Senkung der Steuerungsgenauigkeit: OpenPose ermöglicht es einem nutzenden Programm, die Tiefe des neuronalen Netzes einzustellen. Wird ein flacheres Netz gewählt, wird die Genauigkeit der erkannten Personen gesenkt und im Gegenzug weniger Rechenleistung benötigt. Wir jedoch die Genauigkeit gesenkt, werden Befehle ungenau oder gar nicht erkannt, was nicht im Sinne einer Echtzeitsteuerung ist. Deshalb fällt diese Option für diesen konkreten Anwendungsfall weg.

Nach dem Testen der Steuerung mit mehreren Personen wird klar, dass die Steuerung von verschiedenen Personen unterschiedlich wahrgenommen wird. Fliegt eine Person zum ersten Mal, sind die Gesten oftmals noch nicht verinnerlicht und werden nicht klar ausgeführt. In Folge dessen reagiert die Drohne anders als gewollt oder es wird überhaupt keine Geste vom System wahrgenommen. Abhängig von den Erfahrungen reagieren Piloten anders auf diesen Zustand: Manche probieren einfach weitere Gesten, bis sie das System verstanden haben und es intuitiv wirkt. Andere geben schneller auf und benötigen weitere Hilfestellungen, damit sie im Simulator fliegen können. Die meisten Piloten sind nach einer kurzen Übungsphase in der Lage, sich frei in der simulierten Umgebung zu bewegen. Das Kurvenverhalten wird verstanden und auch die Kombination von mehreren Flugbefehlen funktioniert zufriedenstellend.

Vor Allem die Verwendung der Videobrille sorgt für unterschiedliche Erfahrungen. Für viele Nutzer war dies das erste Mal, dass sie eine Videobrille getragen haben. Erst-Benutzer verlieren ohne ihren optischen Sinn schnell das Gleichgewicht und müssen sich immer wieder neu orientieren, bevor sie den Flug fortsetzen können. Hilfreich dabei ist das anzeigte Bild der Webcam, an welchem sich die Benutzer orientieren können. Erfahrene Benutzer verstehen das System deutlich schneller und benötigen weniger Hilfe, da ihnen das Konzept einer Videobrille bereits vertraut ist.

5.2. Steuerung der Drohne

Für die Steuerung einer AR.Drone sind verschiedene Schnittstellen verfügbar, welche teilweise auch eine Integration in ROS zur Verfügung stellen. Eine weit verbreitete Schnittstelle für ROS heißt ardrone_autonomy und wurde von Mani Monajjemi für die AR.Drone 1.0 und 2.0 entwickelt. (28) Alle Schnittstellen für die AR.Drone funktionieren nur unter der Bedingung, dass der steuernde Computer mit dem gleichen Netzwerk wie die Drohne an sich verbunden ist. Dazu stellt die Drohne ein WLAN-Netzwerk zur Verfügung, in welches sich der steuernde Computer einloggen muss. Sollte dieses Netzwerk nicht ausreichend sein oder sich der Computer mit mehreren Drohnen gleichzeitig verbinden wollen, besteht über eine Telnet-Verbindung mit der Drohne die Möglichkeit, die Drohne manuell mit einem anderen WLAN-Netzwerk zu verbinden. Die Kontrolle über dieses Netzwerk obliegt dann dem Nutzer und kann frei konfiguriert werden. Die eigentliche Kommunikation mit der Drohne findet unabhängig von der Wahl des Netzwerkes über eine UDP-Verbindung statt, über welche sowohl die Steuerungsbefehle als auch Telemetrie-Daten und die Bilder der Kamera versendet werden. (6)

Aus zeitlichen Gründen konnte die Verbindung mit einer AR.Drone nicht für dieses Projekt implementiert werden. Aus architektonischer Sicht sollte die Anbindung der Drohne kein Problem sein. Wie in Abbildung 1 zu sehen ist, wurde die Steuerung in der Architektur berücksichtigt, lediglich nicht implementiert. Der zu implementierende Node empfängt die allgemeinen Flugbefehle der Körperhaltungserkennung und wandelt diese in ein Format passend für die Schnittstelle der Drohne um, analog zu dem Node für die Steuerung des Simulators. Die konvertierten Befehle können im Anschluss über das ardrone_autonomy Package an die Drohne gesendet werden.

6. Zusammenfassung und Ausblick

Im Rahmen der Studienarbeit „I believe I can fly V2.0“ wurde eine Steuerungslogik implementiert, die auf den Bildern einer handelsüblichen 2D-Kamera basiert. Als Ausgabemöglichkeit ist zum Zeitpunkt des Abschlusses eine simulierte Umgebung in AirSim verfügbar, welche gut für Trainings- und Demonstrationszwecke geeignet ist. Die Ansteuerung einer echten Drohne wie der AR.Drone aus dem Hause Parrot ist vorbereitet und kann mit etwas Entwicklungsaufwand an das bestehende System angeschlossen werden. Das Ergebnis dieser Arbeit wurde sowohl am Tag der offenen Tür der DHBW Karlsruhe als auch auf der Hannover Messe 2019 präsentiert und konnte von Besuchern ausprobiert werden.

Im Vergleich zu den vorherigen Studienarbeiten sind die Voraussetzungen an die benötigte Hardware deutlich gesenkt worden. Es kann nicht nur auf die Verfügbarkeit eines 3D-Sensors vollkommen verzichtet werden, sondern auch auf die Verfügbarkeit einer echten Drohne. Die neu entwickelte Gestenerkennung basiert ausschließlich auf einer software-technischen Implementierung und kann aufgrund der breiten Verfügbarkeit von Webcams an vielen Computern ausgeführt werden, vorausgesetzt diese Computer haben die benötigte Rechenkapazität frei. Neben dem Einsparen von Hardware auf Seite der Steuerungseingabe kann auch auf Seite der Ausgabe auf Hardware verzichtet werden. AirSim ermöglicht die Simulation einer Drohne und ist auf vielen Plattformen verfügbar. Somit eignet sich das Projekt hervorragend für den Einsatz zu Trainingszwecken, da in dem Simulator keine Schäden entstehen können. Im Falle eines „Absturzes“ wird die Drohne einfach an ihre Ursprungsposition zurückgesetzt und der Pilot kann weiter fliegen. Ebenfalls kann der Simulator vor dem Kauf einer echten Drohne verwendet werden, um zu entscheiden, ob ein solches System für den benötigten Einsatzzweck in Frage kommt.

Eine geplante Erweiterung ist die optionale Nutzung eines 3D-Sensors, welcher genauere Messwerte für die Koordinaten der einzelnen Körperteile liefern kann. 3D-Sensoren sind in der Lage, auch die Tiefeninformation eines Bildes auszugeben. Mit vorliegenden Tiefeninformationen können sowohl die bestehenden Gesten besser als auch vollkommen neue Gesten erkannt werden, die auf einer Tiefeninformation basieren. Dies ermöglicht eine noch intuitivere Steuerung, wie dies in den ersten Versionen von „I believe I can fly“ der Fall ist. Mögliche 3D-Sensoren sind sowohl Kameras der Intel RealSense Produktserie als auch der bereits verwendete Xbox Kinect Sensor von Microsoft. Beide Sensoren-Systeme bieten Schnittstellen zur freien

Entwicklung an und können den Bedürfnissen nach an die Software angepasst werden.

Weiterhin kann die Steuerung des Simulators verbessert werden. Die Mischung der Roll- und Yaw-Achse könnte je nach Fluglage der Drohne angepasst werden, sodass die Drohne immer an die Situation angepasst fliegt. Befindet sich die Drohne in einem Vorwärtsflug sollte die Drehung auf der Roll-Achse stärker sein als wenn die Drohne ihre Position hält. So kann sich der Nutzer ohne eine Positionsänderung der Drohne umschauen und trotzdem ohne den Verlust der Orientierung eine Kurve fliegen.

Literaturverzeichnis

1. **Benz, Nicolai, Bergen, Marcus von und Vonscheidt, Denis.** *Gestensteuerung eines Flugroboters im AR-Kontext - I believe I can fly.* Karlsruhe : s.n., 2015.
2. **Meise, Christoph und Lenk, Max.** *Entwicklung eines Indoor-Assistenzsystems für Multicopter mit Hilfe von Monokularer Tiefenbildrekonstruktion.* Karlsruhe : s.n., 2017.
3. **Sarkar, Samit.** Microsoft discontinues Xbox One Kinect adapter. *Polygon.* [Online] Vox Media, 03. Januar 2018. [Zitat vom: 01. Februar 2019.] https://www.polygon.com/2018/1/2/16842072/xbox-one-kinect-adapter-out-of-stock-production-ended.
4. **Abrash, Michael.** What VR could, should, and almost certainly will be within two years. [Online] 15. Januar 2014. [Zitat vom: 01. Februar 2019.] http://media.steampowered.com/apps/abrashblog/Abrash%20Dev%20Days%202014.pdf.
5. **NVIDIA Corporation.** Was ist CUDA? [Online] 2019. [Zitat vom: 01. Februar 2019.] https://www.nvidia.de/object/cuda-parallel-computing-de.html.
6. **Parrot Drones SAS.** Parrot for Developers. [Online] 2016. [Zitat vom: 01. Februar 2019.] https://developer.parrot.com/products.html.
7. **Open Robotics Foundation.** ROS. [Online] 01. Februar 2019. [Zitat vom: 01. Februar 2019.] http://www.ros.org.
8. **Bruns, Ralf und Dunkel, Jürgen.** *Event-Driven Architecture. Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse.* Berlin : Springer, 2010. ISBN 978-3-642-02438-2.
9. **Open Robotics Foundation.** Messages - ROS Wiki. [Online] 26. August 2016. [Zitat vom: 04. April 2019.] http://wiki.ros.org/Messages.
10. **Huang, Hongrong und Sturm, Jürgen.** tum_simulator - ROS Wiki. [Online] 17. Februar 2018. [Zitat vom: 04. April 2019.] http://wiki.ros.org/tum_simulator.
11. **Drone Racing League Inc.** DRL SIM 3. [Online] 2019. [Zitat vom: 09. April 2019.] https://thedroneracingleague.com/drl-sim-3/.
12. **Microsoft.** Airsim - Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research. [Online] Microsoft, 04. April 2019. [Zitat vom: 04. April 2019.] https://github.com/Microsoft/AirSim.
13. **Grevelink, Evelyn.** A Closer Look at Object Detection, Recognition and Tracking. *Intel Developer Zone.* [Online] 18. Dezember 2017. [Zitat vom: 11. April 2019.] https://software.intel.com/en-us/articles/a-closer-look-at-object-detection-recognition-and-tracking.
14. **Neumann, Burkhard.** Bildverarbeitung für Einsteiger: Programmbeispiele mit Mathcad. *Bildverarbeitung für Einsteiger: Programmbeispiele mit Mathcad.* Berlin : Springer, 2005.
15. **OpenCV Team.** Background Subtraction. [Online] 11. April 2019. [Zitat vom: 11. April 2019.] https://docs.opencv.org/3.4/db/d5c/tutorial_py_bg_subtraction.html.
16. **KaewTraKulPong, P. und Bowden, R.** *An improved adaptive background mixture model for real-time tracking with shadow detection.* London : Brunel University, 2001.
17. **Zivkovic, Zoran.** *Improved adaptive Gaussian mixture model for background subtraction.* Amsterdam : University of Amsterdam, 2004.
18. **Zivkovic, Zoran und van der Heijden, Ferdinand.** *Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction.* Amsterdam : s.n., 2005.
19. **Godbehere, Andrew B., Matsukawa, Akihiro und Goldberg, Ken.** *Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio*

Art Installation. 2012.
20. **Rey, Günther Daniel und Beck, Fabian.** Neuronale Netze - Eine Einführung. [Online] 11. April 2019. [Zitat vom: 11. April 2019.] http://www.neuronalesnetz.de.
21. **Moeser, Julian.** Künstliche Neuronale Netze - Aufbau & Funktionsweise. [Online] 27. September 2017. [Zitat vom: 24. April 2019.] https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/.
22. **Dhanjal, Sahib Singh.** Mask R-CNN based Pedestrian Detection. [Online] 05. April 2018. [Zitat vom: 24. April 2019.] https://github.com/sahibdhanjal/Mask-RCNN-Pedestrian-Detection.
23. *RMPE: Regional Multi-Person Pose Estimation.* **Fang, Hao-Shu, et al.** 5, China : Shanghai Jiao Tong University, 2016. arXiv:1612.00137 [cs.CV].
24. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.* **Cao, Zhe, et al.** 2018. arXiv:1812.08008v1.
25. **CMU Perceptual Computing Lab.** OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation. *Github.* [Online] 21. April 2019. [Zitat vom: 24. April 2019.] https://github.com/CMU-Perceptual-Computing-Lab/openpose.
26. **Zadeh, Lotfi Asker.** Fuzzy Sets. *Information and Control 8.* California : s.n., 1965.
27. **Müller, Gerhard.** Anwendungen der Fuzzy Logic - Struktur von Fuzzy-Regelungssystemen. [Online] [Zitat vom: 01. Mai 2019.] http://ejb-ressourcen.de/docs/FuzzyLogic/node7.html.
28. **Monajjemi, Mani.** ardrone_autonomy - ROS Wiki. [Online] Open Source Robotics Foundation, 23. April 2015. [Zitat vom: 05. Mai 2019.] http://wiki.ros.org/ardrone_autonomys.