Gestensteuerung eines Flugroboters im AR-Kontext – I believe I can fly

STUDIENARBEIT

für die Prüfung zum

Bachelor of Science

des Studienganges Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Nicolai Benz
Marcus von Bergen
Denis Vonscheidt

Abgabedatum

11.05.2015

Matrikelnummer Nicolai Benz	8180238
Matrikelnummer Marcus von Bergen	7125762
Matrikelnummer Denis Vonscheidt	3108570
Kurs	TINF12B1
Ausbildungsfirma	SAP SE, Walldorf
Gutachter der DHBW	Hans-Jörg Haubner

Selbstständigkeitserklärung

Hiermit erklären wir, dass wir den vorliegenden Praxisbericht selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet haben. Alle wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Karlsruhe, den 29.04.2015

Marcus von Bergen

Nicolai Benz

Denis Vonscheidt

Hinweis: Sollte aus Gründen der Vereinfachung ausschließlich die männliche oder weibliche Form verwendet werden, so schließt dies selbstverständlich alle Geschlechter ein.

Inhaltsverzeichnis

I. Abbildungsverzeichnis 5
II. Tabellenverzeichnis 8
III. Codelistingsverzeichnis 9
IV. Abkürzungsverzeichnis 10
1 Einleitung 11
1.1 Aufgabenstellung 12
1.2 Motivation 13
1.3 Technische Voraussetzungen 14
2 Quadrocopter 15
2.1 Kinematik 16
2.2 Sensorik 24
2.2.1 Gyroskopische-Sensoren 24
2.2.2 Translationssensoren 26
2.2.3 Ultraschallsensoren 26
2.2.4 GPS Sensoren 27
2.2.5 Algorithmen-gestützte Kamerasensoren 28
2.3 AR.Drone 2.0 29
3 Fuzzy-Controller 31
3.1 Grundprinzipien der Fuzzy-Regelung 32
3.1.1 Fuzzifizierung 33
3.1.2 Inferenz 35
3.1.3 Defuzzifizierung 39
3.2 Aufbau des Controllers 42
3.2.1 Definition der Ein- und Ausgangsgrößen 42
3.2.2 Wahl der Fuzzy-Operatoren 45
3.2.3 Regelstrategie und Regelsatz 46

3.2.4 Entwurf der Zugehörigkeitsfunktionen 50
3.3 Auswirkung auf die Flugeigenschaften der Drohne 60
3.4 Alternative Lösungsmöglichkeit 61
4 User Experience 63
4.1 Wie „I believe I can fly“ im Projekt umgesetzt wurde 64
4.2 Microsoft Kinect 71
4.2.1 Entwicklungsrichtlinien 72
4.2.2 Programmieren mit Kinect SDK 1.8 für Windows 76
4.3 Augmented Reality-Brillen 86
4.3.1 Zeiss Cinemizer 87
4.3.2 Oculus Rift 88
5 Technische Realisierung des Projektes 89
5.1 Umsetzung der Drohnenkommunikation 90
5.1.1 Senden von Befehlen 90
5.1.2 Empfangen von Navigationsdaten 93
5.1.3 Konfiguration der Drohne 96
5.1.4 Empfangen des Videostreams 98
5.2 Aufbau der Benutzeroberfläche 100
5.2.1 Fenstermanagement 100
5.2.2 Head-Up-Display 101
5.3 Steuerung der Drohne 103
5.3.1 Kinect 103
5.3.2 Headtracker 106
5.3.3 Fuzzy-Controller 107
5.4 Zusammenspiel der Komponenten 109
6 Fazit und Ausblick 111
V. Literaturverzeichnis 112

I. ABBILDUNGSVERZEICHNIS

Abbildung 1: Zusammenspiel der Komponenten 14

Abbildung 2: Paarweise entgegengesetzt drehende Rotoren 16

Abbildung 3: Addition von Kräften 17

Abbildung 4: Entgegengesetzte Kräfte bei Quadrocoptern 17

Abbildung 5: Roll-Pitch-Yaw Winkel 18

Abbildung 6: Aufsteigen und Sinken eines Quadrocopters 20

Abbildung 7: Neigen eines Quadrocopters 21

Abbildung 8: Rotation eines Quadrocopters 22

Abbildung 9: Rotationsenergie τ 23

Abbildung 10: Mechanisches Gyroskop 25

Abbildung 11: Nutzung des Global Positioning System 28

Abbildung 12: Grundprinzip Fuzzy-Regelung 32

Abbildung 13: Linguistische Variable mit Zugehörigkeitsfunktionen 34

Abbildung 14: Min-Operator - Verknüpfung zweier Fuzzy-Mengen 36

Abbildung 15: Max-Operator – Verknüpfung zweier Fuzzy-Mengen 37

Abbildung 16: Komplement einer Fuzzy-Menge 38

Abbildung 17: Defuzzifizierung nach der Schwerpunktmethode 40

Abbildung 18: Drohnen Fuzzy-Controller - Übersicht der Ein- und Ausgangsgrößen 44

Abbildung 19: Zugehörigkeitsfunktion für „strongForward“ des Eingangs „backward“ 51

Abbildung 20: Zugehörigkeitsfunktion für „mediumForward“ des Eingangs „backward“ 52

Abbildung 21: Zugehörigkeitsfunktion für „zero“ des Eingangs „backward“ 53

Abbildung 22: Zugehörigkeitsfunktion für „mediumBackward“ des Eingangs „backward“ 53

Abbildung 23: Zugehörigkeitsfunktion für „strongBackward“ des Eingangs „backward“ 54

Abbildung 24: Zugehörigkeitsfunktion für „strongForward“ des Ausgangs „backwardSpeed“ 55

Abbildung 25: Zugehörigkeitsfunktion für „mediumForward“ des Ausgangs „backwardSpeed“.....56

Abbildung 26: Zugehörigkeitsfunktion für „zero“ des Ausgangs „backwardSpeed“.....57

Abbildung 27: Zugehörigkeitsfunktion für „mediumBackward“ des Ausgangs „backwardSpeed“.....57

Abbildung 28: Zugehörigkeitsfunktion für „strongBackward“ des Ausgangs „backwardSpeed“.....58

Abbildung 29: Verarbeitung der Eingangssignale (blau gestrichelt) in Stellgrößen für die Drohne (rot) anhand einer logistischen Funktion.....61

Abbildung 30: Die Ansicht des Webcockpits.....65

Abbildung 31: Head-Up-Display eine Helicopters in einem Spiel.....66

Abbildung 32: Das Head-Up-Display der "I believe I can fly"-Anwendung.....67

Abbildung 33: AR-Brillen zwingen den Nutzer zur Ansicht des Cockpits.....68

Abbildung 34: Das Steuerungsmenu der Anwendung.....69

Abbildung 35: Übersicht über die Technologie eines Kinect-Sensors.....71

Abbildung 36: Verschiedene Gesten, die von einer Microsoft Kinect interpretiert werden können....72

Abbildung 37: Handgesten vor dem Körper.....73

Abbildung 38: Zu schnelle Bewegung.....73

Abbildung 39: Verlassen des Aufnahmebereichs.....73

Abbildung 40: Größenunterschiede bei den Nutzern.....74

Abbildung 41: Cursor als visuelles Feedback für die Hand.....75

Abbildung 42: Orientierungshilfen für den Nutzer geben.....75

Abbildung 43: Ein "User Viewer" hilft bei der Orientierung des Nutzers.....75

Abbildung 44: Nach Anschluss eines neuen Kinect-Sensors initialisiert die Anwendung den Sensor...79

Abbildung 45: Ist der Sensor verfügbar ändert sich das Icon vom KinectSensorChooserUI.....79

Abbildung 46: Ein Fenster mit User Viewer mit einem Cursor.....82

Abbildung 47: User Viewer und Cursor passen sich bei Bewegung des Nutzers an.....82

Abbildung 48: Kontinuierliches „Drücken“ der Buttons verhindert versehentliche kurze Betätigung .83

Abbildung 49: Bei Kontinuierlichem Druck wird die Delegate-Methode ausgeführt.....83

Abbildung 50: Der ScrollViewer kann gegriffen und verschoben werden 85
Abbildung 51: Dadurch können weitere Elemente erreicht werden 85
Abbildung 52: Eine Virtual Reality-Brille Zeiss Cinemizer 87
Abbildung 53: Die Virtual Reality-Brille Oculus Rift 88
Abbildung 54: Übersicht der Befehlsklassen für die AR.Drone 92
Abbildung 55: Protokoll zum Empfang der Navigationsdaten 93
Abbildung 56: Aufbau des Navigationsdaten Bytestreams 94
Abbildung 57: Klassenstruktur der Konfigurationssektionen 96
Abbildung 58: Klassendiagramm zum Empfang des Videostreams 98
Abbildung 59: Das Klassendiagramm zur Navigation zwischen den Seiten 100
Abbildung 60: HUD des Drohnencockpits in Visual Studio 101
Abbildung 61: Klassendiagramm für das HUD 102
Abbildung 62: Klassendiagramm des Teilprojektes "Kinect" 103
Abbildung 63: Geste für die Aufwärtsbewegung der Drohne 104
Abbildung 64: Geste für linke Seitwärtsbewegung 104
Abbildung 65: Geste für eine rechte Rotation 104
Abbildung 66: Geste für die Vorwärtsbewegung 104
Abbildung 67: Cinemizer Headtracker 106
Abbildung 68: Klassendiagramm für den Headtracker 107
Abbildung 69: Klassenstruktur des Fuzzy-Controllers 108
Abbildung 70: Die Abhängigkeiten zwischen den einzelnen Teilprojekten 109

II. TABELLENVERZEICHNIS

Tabelle 1: Technische Spezifikationen der AR.Drone2.0 29

Tabelle 2: Zuweisung linguistischer Terme zu linguistischen Variablen 44

Tabelle 3: Mapping der Ein- und Ausgänge für die Regelmodellierung 46

Tabelle 4: Regelübersicht für den Ausgang "backwardSpeed" für Kurvenflug 47

Tabelle 5: Regelübersicht für den Ausgang "sidewardSpeed" für Kurvenflug 48

Tabelle 6: Regelübersicht für den Ausgang "rotationSpeed" für Kurvenflug 48

Tabelle 7: Linguistische Terme der Eingänge mit identischen Zugehörigkeitsfunktionen 51

Tabelle 8: Linguistische Terme der Ausgänge mit identischen Zugehörigkeitsfunktionen 55

Tabelle 9: Übersicht der Drohnenbefehle 91

III. CODELISTINGSVERZEICHNIS

Listing 1: Der Window-Tag mit allen benötigten Namespaces 76

Listing 2: UI-Element zum Anzeigen des Status zur Konnektivität eines Kinect-Sensors 77

Listing 3: Anmeldung einer Delegate-Methode für die Initialisierung eines Kinect-Sensors 78

Listing 4: Erweiterung der SensorChooserOnKinectChanged-Methode.....80

Listing 5: Hinzufügen eines User Viewers im XAML 81

Listing 6: Hinzufügen eines TileButtons und eines CircleButtons im XAML.....82

Listing 7: Implementierung einer Delegate-Methode für die Betätigung eines Buttons 83

Listing 8: Hinzufügen eines ScrollViewers im XAML 84

Listing 9: Dem ScrollViewer werden programmatisch CircleButtons hinzugefügt.....84

IV. ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface
AR	Augmented Reality
FIFO	First In First Out
FIS	Fuzzy Inference System
GPS	Global Positioning System
HDMI	High Definition Multimedia Interface
HUD	Head-Up-Display
OLED	Organic Light Emitting Diode
PaVE	Parrot Video Encapsulation
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
WPF	Window Presentation Foundation
XAML	Extensible Application Markup Language

1 EINLEITUNG

Wer hat als Kind oder als Erwachsener nicht gerne mit ferngesteuerten Autos oder Flugzeugen gespielt? Solche fernsteuerbaren Roboter sind ein Kassenschlager und etliche Modellflugbegeisterte können Stunden auf dem Flugplatz verbringen und ihr Modellflugzeug durch die Luft sausen lassen. Manch einer träumt sogar davon, selbst vom Boden abheben zu können und dasselbe schwerelos-Gefühl zu erleben, wie das Flugzeug, das er steuert. Für viele Menschen ist es ein Traum, einmal im Leben aus der Perspektive eines Vogels die Welt zu sehen.

Mit dem heutigen Stand der Technik ist das möglich. Modellflugzeuge und Quadrocopter haben sich weiterentwickelt und sind bestückt mit Sensoren jeglicher Art. So können Quadrocopter über drahtlose Netzwerke mit einem Computer kommunizieren und über ein technisches Protokoll Steuerungsbefehle entgegennehmen. Als Steuergerät muss dabei nicht zwangsweise eine Tastatur oder ein Controller dienen. Visuelle Sensoren wie die einer Microsoft Kinect ermöglichen es, Personen im Raum zu lokalisieren und ihre Bewegungen zu erfassen. Anhand ihrer Bewegungen lassen sich Steuerungsbefehle für die Drohne ableiten. Dem Benutzer wird es somit ermöglicht, ohne jegliches Steuergerät und nur mit seinen eigenen Gesten, den Quadrocopter zu steuern. Sogenannte „Augmented Reality“ (AR) -Brillen erlauben es, das Videobild des Quadrocopters vor den eigenen Augen zu sehen, als würde der Nutzer selbst aus der Perspektive der Drohne die Welt erblicken.

Mit solch einer technischen Ausrüstung wurde das Projekt „I believe I can fly“ realisiert und die Ergebnisse und Erkenntnisse in diesem Dokument beschrieben. Mit dem Ergebnis des Projektes soll der Traum der Menschen erfüllt werden, auf spielerische Art und Weise die Umgebung aus der Luftperspektive zu betrachten.

1.1 AUFGABENSTELLUNG

„Gestensteuerung eines Flugroboters im AR-Kontext – I believe I can fly“, eine Aufgabenstellung mit viel Platz für Kreativität. Die Aufgabe bestand darin, eine Drohne, genauer einen Quadrocopter, ausschließlich durch Körpergesten zu steuern. Der „Augmented Reality“-Effekt soll durch das Zusammenspiel zwischen dem Videobild des Quadrocopters und der Gestensteuerung des Piloten erzeugt werden. Dabei soll dem Benutzer der Eindruck vermittelt werden, wirklich zu fliegen.

1.2 MOTIVATION

Die Motivation des Projektes ist grundsätzlich spielerischer Natur. Jede Komponente des Projektes wird auch in anderen Szenarien für Spiele oder Entertainment verwendet. Mit einer Microsoft Kinect kann z.B. virtuelles Tennis gespielt werden und AR-Brillen werden heutzutage von sehr vielen Startup-Unternehmen, beispielsweise zur Spielentwicklung, eingesetzt. Modellflugzeuge oder ferngesteuerte Quadrocopter waren auch schon früher ein begehrtes Spielzeug.

Die Idee und Motivation hinter dem Projekt besteht darin, die vorhandenen technischen Spielzeuge miteinander zu verbinden und einen „Showcase“ zu entwickeln, mit welchem der heutige Stand der Technik demonstriert werden kann. Die Duale Hochschule Karlsruhe beispielsweise, kann mit einem solchen Projekt an einem Tag der offenen Tür zeigen, dass die heutige Technik es erlaubt, Flugdrohnen auf intuitive Art und Weise zu bedienen.

1.3 Technische Voraussetzungen

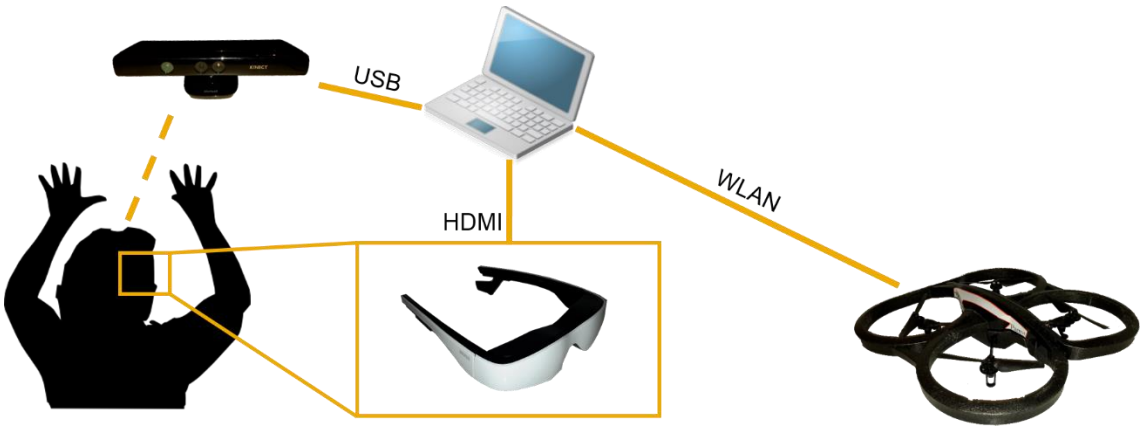


Abbildung 1: Zusammenspiel der Komponenten

Abbildung 1 zeigt die Konfiguration der für das Projekt verwendeten Komponenten. Benötigt wird ein Computer mit einem Windows Betriebssystem ab Version 7, welcher über einen Wireless-LAN-Adapter verfügt. Als Quadrocopter wurde die *Parrot AR.Drone 2.0* verwendet, die ein WLAN-Netzwerk zur Verfügung stellt, zu welchem sich der Computer verbindet. Zur Steuerung der Drohne werden die Gesten des Benutzers über eine Microsoft Kinect aufgezeichnet und über eine USB-Verbindung an den Computer übertragen. Die Darstellung der Benutzeroberfläche kann entweder über den Monitor des Computers erfolgen oder über die *Zeiss Cinemizer Augmented Reality Brille*. Die Brille wird per HDMI an den Computer angeschlossen. Zusätzlich wird ein an der *Zeiss Cinemizer* befestigter Headtracker verwendet, welcher wiederum per USB an den Computer angeschlossen wird. Der Headtracker ist ebenfalls von der Firma Zeiss.

Microsoft stellt für die Kinect ein Software Development Kit (SDK) in den .NET-Sprachen C#, C++ und Visual Basic zur Verfügung. Als Entwicklungsumgebungen standen *Visual Studio* von Microsoft in der Version 2013, sowie *Matlab* von Mathworks zur Verfügung.

2 QUADROCOPTER

Quadrocopter sind Fluggeräte aus der Familie der Helikopter. Sie verfügen über vier auf einer Ebene angeordneter Rotoren oder Propeller, welche senkrecht nach unten wirken und somit für den Auftrieb sorgen.

Wie auch Helikopter sind Quadrocopter durch die nach unten wirkenden Rotoren sehr wendig. Durch die Möglichkeit alle Rotoren individuell ansteuern zu können, sind Quadrocopter in der Lage, sich auf der Stelle zu drehen und in jede beliebige Richtung zu neigen. Um den Quadrocopter in eine bestimmte Richtung fliegen zu lassen muss dieser lediglich in diese Richtung geneigt werden.

Quadrocopter werden heute zumeist im Modellbau als ferngesteuerte oder autonom fliegende Drohnen verwendet.

2.1 KINEMATIK

Die Kinematik beschreibt mathematische und geometrische Methoden zur Beschreibung von Bewegungen von Körpern im Raum.[1] Die beschreibenden Größen bei der Kinematik sind die Position, die Geschwindigkeit und die Beschleunigung.

Die Bewegung an sich wird als Dynamik oder auch als Kinetik bezeichnet.[2] Die Dynamik ist demnach die Bewegung, welche durch die Kinematik formell beschrieben werden kann.

Bei Quadrocoptern wird die Dynamik ausschließlich durch die Kräfte der Rotoren oder Propeller erzeugt. Die Rotoren verdrängen durch die Drehbewegung Luft und erzeugen so eine Schubkraft. Da alle vier Rotoren so ausgerichtet sind, dass sie die Luft senkrecht nach unten verdrängen, wird ein Auftrieb erzeugt. Zur besseren Stabilität drehen die Rotoren bei Quadrocoptern paarweise entgegengesetzt und verhindern so die Eigenrotation. Die nachfolgende *Abbildung 2* visualisiert dieses Verhalten. Sie zeigt einen skizzenhaften Quadrocopter mit paarweise unterschiedlich rotierenden Propellern.

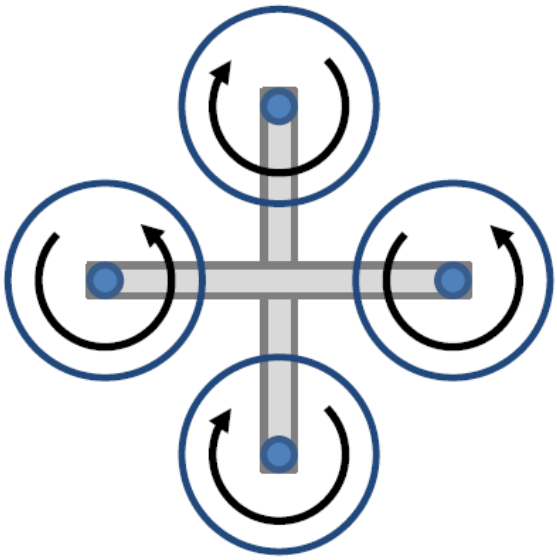


Abbildung 2: Paarweise entgegengesetzt drehende Rotoren[3]

[1] (Popov, 2009, S. 1)
[2] (Popov, 2009, S. 1)
[3] (Sturm, 2014)

Kräfte sind Vektoren und können addiert werden. Aus der Addition aller beteiligten Kräfte ergibt sich die Gesamtkraft als Vektor $F = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

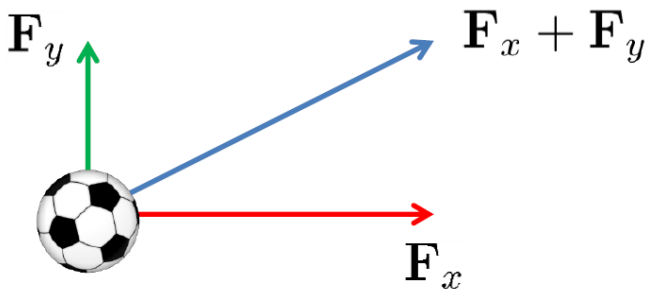


Abbildung 3: Addition von Kräften[4]

Abbildung 3 zeigt am Beispiel eines Balls, auf den die Kräfte F_x und F_y wirken, das Verfahren zur Berechnung der resultierenden Kraft. Die resultierende Kraft wird als blauer Pfeil dargestellt und berechnet sich durch Vektoraddition der Einzelkräfte, die als grüner und roter Pfeil markiert sind.

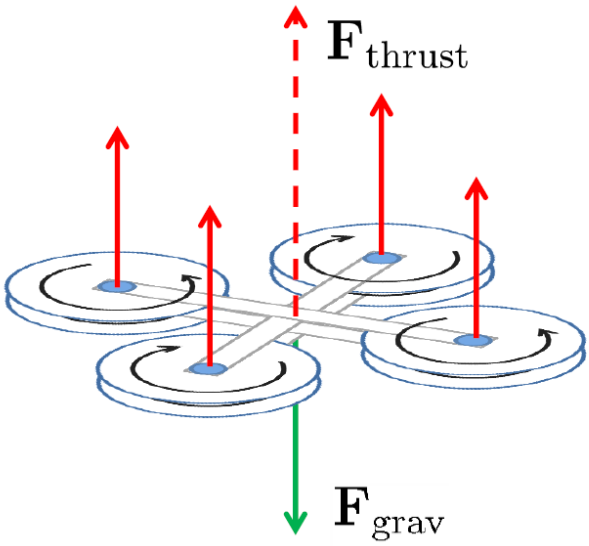


Abbildung 4: Entgegengesetzte Kräfte bei Quadrocoptern[5]

[4] (Sturm, 2014)
[5] (Sturm, 2014)

Werden die Kräfte aller vier Rotoren aus *Abbildung 4* addiert, ergibt sich die Gesamtschubkraft des Quadrocopters.

$$F_{thrust} = F_{thrust\,1} + F_{thrust\,2} + F_{thrust\,3} + F_{thrust\,4}$$[6]

Die der Schubkraft entgegenwirkende Kraft ist die Gravitation. Aus der Addition der Schubkraft und der Gravitationskraft ergibt sich die Gesamtkraft F_{Gesamt}.

$$F_{Gesamt} = F_{thrust} + F_{grav}$$

Wenn der Quadrocopter in eine Richtung geneigt ist und nicht waagerecht in der Luft schwebt, muss bei der Kräfteberechnung eine Rotationsmatrix R berücksichtigt werden, welche die aktuelle Orientierung des Quadrocopters im Raum beschreibt.[7]

Daraus ergibt sich für die Berechnung der Gesamtkraft folgende Formel:

$$F_{Gesamt} = R * F_{thrust} + F_{grav}$$

Die Rotationsmatrix R ergibt sich aus dem Produkt von drei Matrizen, die jeweils die Drehung um eine Achse des Koordinatensystems im Roll-Pitch-Yaw-System beschreiben. In der folgenden *Abbildung 5* wird das Roll-Pitch-Yaw-System visualisiert.

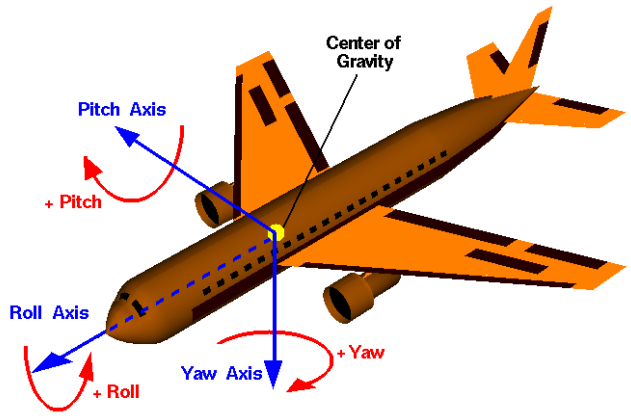


Abbildung 5: Roll-Pitch-Yaw Winkel[8]

[6] (Sturm, 2014)
[7] (Sturm, 2014)
[8] NASA

Die drei Matrizen für die Drehachsen und die daraus resultierende Rotationsmatrix R lauten wie folgt:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$ [9]

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$ [10]

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ [11]

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma)$$ [12]

Wenn der Quadrocopter nicht geneigt ist und für α, β und γ 0° eingesetzt werden, ergeben sich für alle Rotationsmatrizen eine Einheitsmatrix. Durch die Multiplikation dieser ergibt sich für R ebenfalls eine Einheitsmatrix, wodurch R die Gesamtkraft nicht mehr beeinflusst.

Aus der allgemeinen Formel $F = m * a$ lässt sich die Beschleunigung des Quadrocopters berechnen. Hierzu müssen die zugrunde liegenden Kräfte beim Quadrocopter in die umgestellte Formel $a = \frac{F}{m}$ eingesetzt werden. Die sich daraus ergebende Formel lautet:

$$a = \frac{(R * F_{thrust} + F_{grav})}{m}$$ [13]

Mithilfe dieser Information kann die Geschwindigkeit zu einem Zeitpunkt t_i berechnet werden. Zur Berechnung der Geschwindigkeit $v(t_i)$ muss das Integral über die Beschleunigungsfunktion $a(t)$ in den Grenzen t_0 und t_i gebildet und zur Startgeschwindigkeit v_0 addiert werden. Für die Geschwindigkeit gilt somit die Formel:

$$v(t) = v_0 + \int_{t_0}^{t_i} a(t)dt$$

[9] (Strand, 2014)
[10] (Strand, 2014)
[11] (Strand, 2014)
[12] (Strand, 2014)
[13] (Sturm, 2014)

Des Weiteren lässt sich die Schubkraft eines Rotors anhand dessen Rotationsgeschwindigkeit, dessen Radius, sowie der Neigungswinkel der Rotorblätter bestimmen. Der einzige variable Faktor ist die Rotationsgeschwindigkeit. Somit kann die Geschwindigkeit des Quadrocopters, als Vektor, zu einem beliebigen Zeitpunkt anhand der bisherigen Rotationsgeschwindigkeiten der Rotorblätter bestimmt werden. Dies gilt allerdings nur unter der Annahme, dass keine anderen Kräfte, außer der Gravitation, auf den Quadrocopter wirken.

Um den Quadrocopter navigieren zu können, müssen die einzelnen Rotoren unterschiedlich stark angesteuert werden. Je nach Manöver müssen die Rotoren unterschiedlich starke Schubkräfte entwickeln. Soll der Quadrocopter seine Position halten, also in der Luft schweben, muss die resultierende Schubkraft exakt die Gravitationskraft ausgleichen. Zudem müssen alle Rotoren die gleiche Schubkraft entwickeln, damit sich der Quadrocopter in der Waage hält und sich nicht dreht. In Formeln ausgedrückt muss gelten:

I. $$-F_{grav} = F_{thrust} = F_{thrust\,1} + F_{thrust\,2} + F_{thrust\,3} + F_{thrust\,4}$$

II. $$F_{thrust\,1} = F_{thrust\,2} = F_{thrust\,3} = F_{thrust\,4}$$

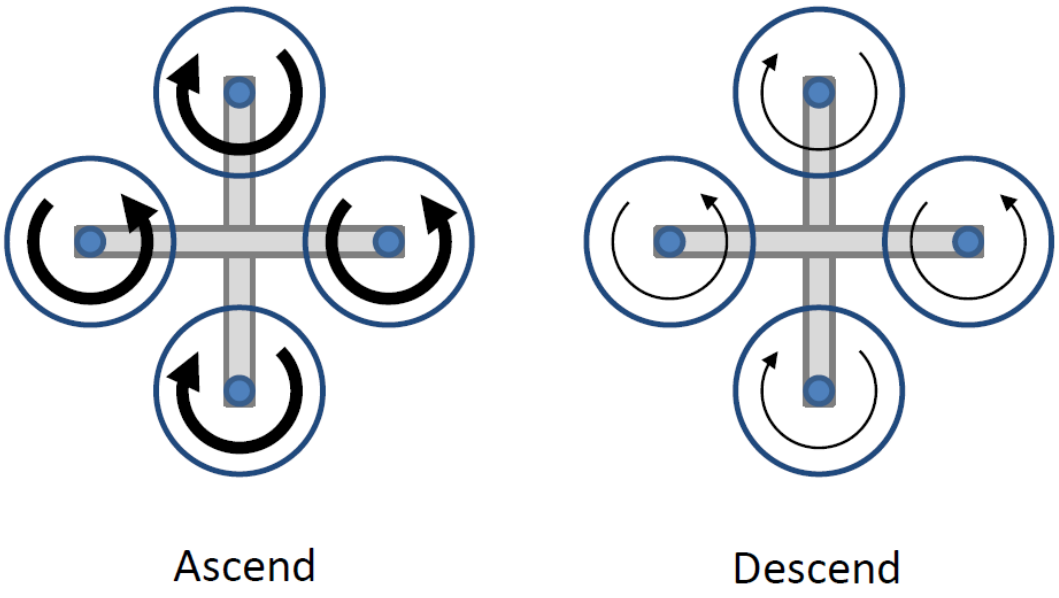


Abbildung 6: Aufsteigen und Sinken eines Quadrocopters[14]

[14] (Sturm, 2014)

Um den Quadrocopter aufsteigen zu lassen, muss die Gesamtschubkraft größer als die Gravitationskraft sein. Dazu müssen die Schubkräfte $F_{thrust\,1}..F_{thrust\,4}$ der einzelnen Rotoren gleichmäßig erhöht werden. Um den Quadrocopter sinken zu lassen muss die Gesamtschubkraft dementsprechend geringer sein als die Gravitationskraft. Dazu müssen die Schubkräfte der Rotoren gleichmäßig verringert werden, bis die Gravitationskraft stärker als die resultierende Schubkraft ist. Die beiden Vorgänge Aufsteigen und Sinken sind in *Abbildung 6* dargestellt.

Um den Quadrocopter in eine Richtung zu neigen und dementsprechend in eine Richtung fliegen zu lassen, muss die Schubkraft des Rotors auf der Seite, in die sich geneigt werden soll, verringert und die Schubkraft des entgegenliegenden Rotors proportional erhöht werden. Währenddessen verändert sich die Schubkraft der anderen beiden Rotoren nicht. Davon ausgegangen, dass sich der Quadrocopter in Richtung vom ersten Rotor Neigen soll, bedeutet das in Formeln ausgedrückt:

I. $F_{thrust\,1} + F_{thrust\,3} = F_{thrust\,2} + F_{thrust\,4}$

II. $F_{thrust\,1} < F_{thrust\,3}$

III. $F_{thrust\,2} = F_{thrust\,4}$

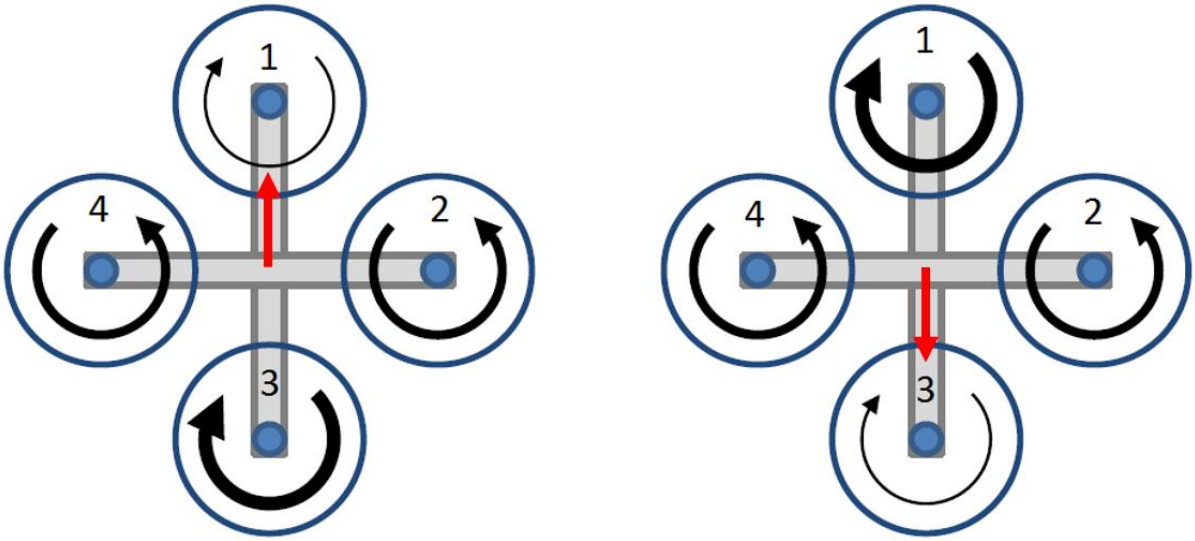


Abbildung 7: Neigen eines Quadrocopters[15]

In *Abbildung 7* ist die Neigung eines Quadrocopters anhand von zwei Beispielen dargestellt. Der linke Quadrocopter neigt sich in Richtung des Rotors 1. Der rechte hingegen neigt sich in Richtung des

[15] (Sturm, 2014)

Rotors 3. Die roten Pfeile symbolisieren die jeweils resultierenden Bewegungsrichtungen der Quadrocopter.

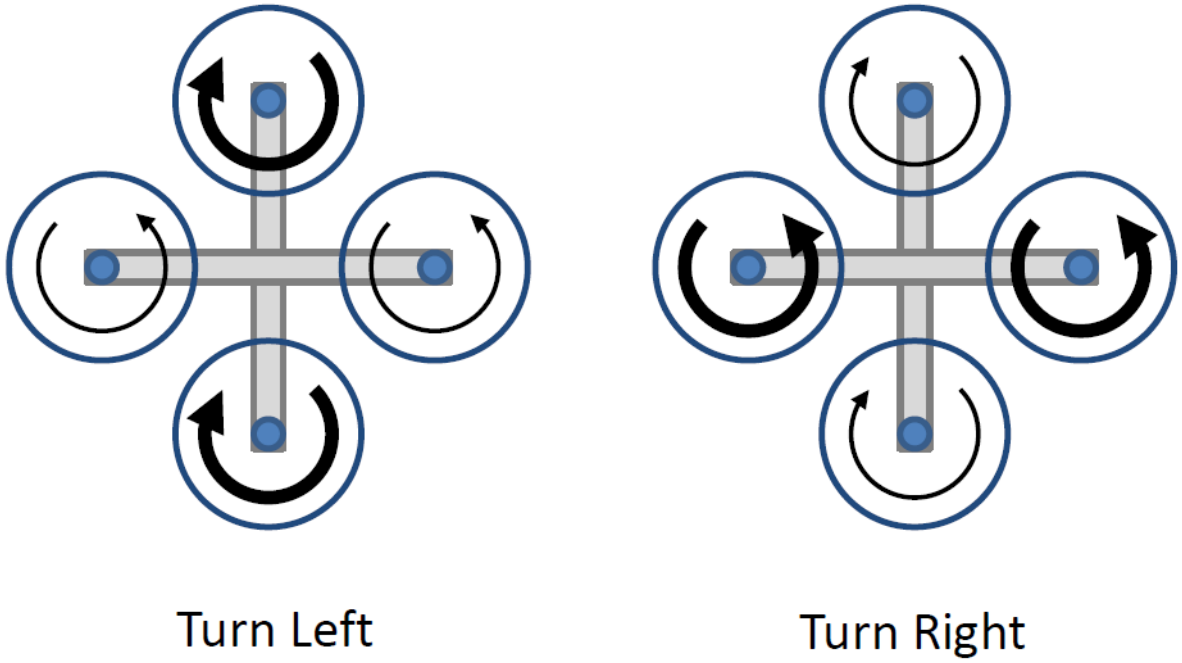


Abbildung 8: Rotation eines Quadrocopters[16]

In *Abbildung 8* ist die Rotation eines Quadrocopters um seinen Mittelpunkt visualisiert. Der linke dreht sich nach links und der rechte Quadrocopter nach rechts. Damit der Quadrocopter sich auf der Stelle dreht müssen die zwei sich gegenüberliegenden Rotoren als Paare betrachtet werden. Bei einem Paar wird dabei die Schubkraft erhöht während bei dem anderen Paar die Schubkraft proportional verringert wird. Dabei ist zu beachten, dass die Gesamtschubkraft konstant bleibt. Je nachdem, in welche Richtung sich der Quadrocopter drehen soll, muss ein anderes Paar seine Schubkraft erhöhen. Es muss zum Beispiel gelten:

I. $F_{thrust\,1} + F_{thrust\,3} > F_{thrust\,2} + F_{thrust\,4}$

II. $F_{thrust\,1} = F_{thrust\,3}$

III. $F_{thrust\,2} = F_{thrust\,4}$

[16] (Sturm, 2014)

Die induzierte Gesamtrotation τ des Quadrocopters ergibt sich aus der Rotationenergie der einzelnen Rotoren. Die Formel zur Berechnung der Gesamtrotation ist folgende:

$$\tau = \tau_1 - \tau_2 + \tau_3 - \tau_4 = \tau_1 + \tau_3 - (\tau_2 + \tau_3)^{17}$$

τ wird dabei über die Kraft F, welche zur Drehung des Rotors aufgewendet wird und dem Radius r des Rotors durch die Formel $\tau = F * r$ berechnet.[18] Durch Ausnutzung der Trägheit des Quadrocopters wird so eine gewisse Drehung des Flugkörpers erreicht. Die Komponenten F, r und τ werden in der *Abbildung 9* dargestellt.

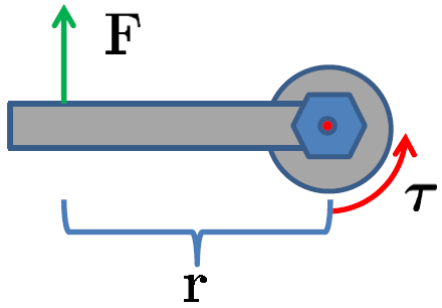


Abbildung 9: Rotationsenergie τ[19]

Durch die Kombination der verschiedenen Navigationsmöglichkeiten und einer gezielten Ansteuerung der Rotoren ist es dem Quadrocopter möglich komplexe und wendige Manöver zu fliegen.

[17] (Sturm, 2014)
[18] (Sturm, 2014)
[19] (Sturm, 2014)

2.2 SENSORIK

Um Aussagen über den aktuellen Status – zum Beispiel Neigung, Flughöhe, Fluggeschwindigkeit – eines Quadrocopters treffen zu können und um diese Erkenntnisse für die Flugstabilität oder die Standorterkennung zu nutzen, sind Quadrocopter meist mit einer Vielzahl von Sensoren ausgestattet. Die Daten der Sensoren werden gesammelt und durch spezielle Algorithmen kombiniert und fusioniert. Des Weiteren können die fusionierten Daten zur Verbesserung des Flugverhaltens der Drohne eingesetzt werden. Beispielsweise kann der Neigungswinkel der Drohne begrenzt werden.

Ausgestattet werden Quadrocopter zumeist mit Inertialsensoren, wie translatorische Sensoren (Beschleunigungssensoren) und gyroskopische Sensoren (Drehratensensoren), Ultraschallsensoren zur Abstands- und Höhenbestimmung und Sensoren für die absolute Positionsbestimmung, wie GPS oder algorithmengestützte Kameras.[20]

2.2.1 Gyroskopische-Sensoren

Gyroskope sind Orientierungssensoren, welche auf einen Fixpunkt ausgerichtet werden. Diesen Fixpunkt behalten sie immer bei, egal in welche Richtung sich das Objekt dreht oder neigt, auf welchem das Gyroskop verbaut wird. Ein Gyroskop ist also ein Instrument zur absoluten Orientierungsbestimmung.[21]

Es wird zwischen zwei Arten von Gyroskopen unterschieden: mechanische Gyroskope und optische Gyroskope.

Mechanische Gyroskope Nutzen die Trägheit eines sich schnell drehenden Rotors. Das Gyroskop wird dabei in sehr hoher Geschwindigkeit um sich selbst gedreht.

[20] (Bristeau, Callou, Vissière, & Petit, 2011, p. 1478)
[21] (Siegwart & Nourbakhsh, 2004, p. 99)

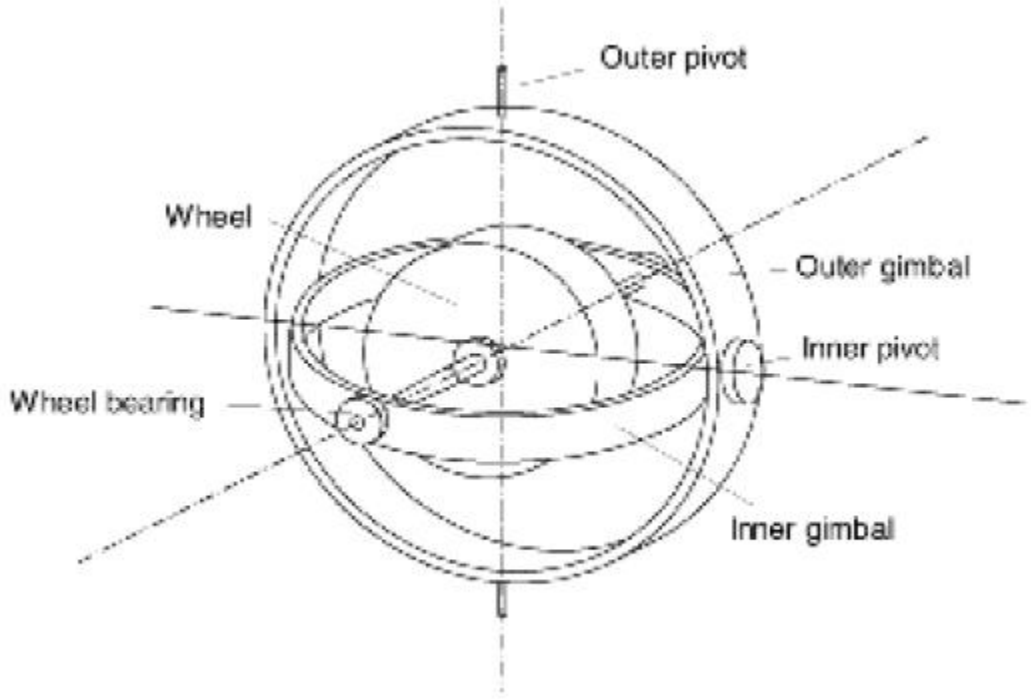


Abbildung 10: Mechanisches Gyroskop[22]

In *Abbildung 10* ist ein dreiachsiges mechanisches Gyroskop zu sehen. Durch die schnelle Rotation bleiben die Achsen des Gyroskops immer konstant gleich, während sich der Quadrocopter in jede Richtung neigen und drehen kann. Damit ist es möglich, die aktuelle Orientierung anhand der Gelenkstellungen des Gyroskops abzulesen.

Optische Gyroskope basieren auf der Konstanz der Lichtgeschwindigkeit. Zwei monochromatische Lichtstrahlen oder Laser werden von derselben Lichtquelle emittiert. Der eine Strahl bewegt sich links herum in einem Kreisel, der andere Rechts herum. Beide werden dabei durch Lichtwellenleiter auf ihrer Bahn geführt. Darauf basierend, dass die Lichtgeschwindigkeit immer konstant ist und sich das optische Gyroskop dreht, erreicht ein Lichtstrahl sein Ziel etwas früher als der andere, weil der zurückzulegende Weg durch die Rotation geringer ist als der zurückzulegende Weg des anderen Lichtstrahls. Die Lichtstrahlen werden in einer hohen Frequenz emittiert. Der Lichtstrahl der gegen die Drehrichtung unterwegs ist hat zwangsläufig eine höhere Frequenz beim erreichen seines Zieles,

[22] (Siegwart & Nourbakhsh, 2004, p. 100)

da die Lichtstrahlen in kürzerem Abstand zueinander ankommen, als auf der anderen Seite. Aus dem Frequenzunterschied ist es möglich die Orientierung zu errechnen.[23]

2.2.2 Translationssensoren

Translationssensoren, auch Beschleunigungssensoren genannt, nutzen die Trägheit der Masse um die momentane Beschleunigung eines Körpers zu ermitteln. Die Masse wird an einem beweglichen Material befestigt, welches fest an einem Körper verankert ist. Wenn der Sensor beschleunigt wird, wird durch die Trägheit der Masse eine Kraft ausgeübt. Diese Kraft kann, je nach Sensor, auf unterschiedliche Art und Weise abgegriffen werden. Das zweite Newtonsche Gesetz lautet *„Die Änderung der Bewegung einer Masse ist der Einwirkung der bewegenden Kraft proportional und geschieht nach der Richtung derjenigen geraden Linie, nach welcher jene Kraft wirkt“*[24].Daraus folgt, dass die auf die Masse ausgeübte Kraft proportional zur Beschleunigung ist. So kann durch die Messung der Kraft eine Aussage zur aktuellen Beschleunigung getroffen werden.[25]

2.2.3 Ultraschallsensoren

Das Grundprinzip von Ultraschallsensoren ist das Aussenden einer Reihe von hochfrequenten Ultraschallwellen und das Messen der Zeit, welche diese benötigen um von einem Hindernis reflektiert zu werden und wieder zum Sensor zurückzukehren. Aus der Zeit t, welche die Ultraschallwellen für den Hin- und Rückweg benötigen und aus der Geschwindigkeit des Schalls c lässt sich die Distanz d zwischen Hindernis und Sensor errechnen. Die Formel für die Berechnung lautet wie folgt:

$$d = \frac{c * t}{2}$$

Die Schallgeschwindigkeit c ist durch die Formel

$$c = \sqrt{\gamma * R * T}$$

mit:

[23] (Siegwart & Nourbakhsh, 2004, p. 101)
[24] (Buchner, 2009, S. 9)
[25] (Buchner, 2009, S. 9)

$\gamma =$ Isentropenexponent (Angabe zur thermischen Raumausdehung)

$R =$ universelle Gaskonstante

$T =$ Temperatur in Grad Kelvin

gegeben. In Luft unter normalen Luftdruck und bei einer Temperatur von 20°C ist die Schallgeschwindigkeit ungefähr $c \approx 343 \frac{m}{s}$.[26]

2.2.4 GPS Sensoren

Global Positioning System (GPS) Sensoren dienen zur absoluten Standortbestimmung eines Objektes. Der Sensor braucht Kontakt zu mindestens 4 GPS Satelliten gleichzeitig, um seine eigene Position zu bestimmen. Alle Satelliten senden regelmäßig und gleichzeitig Informationen bezüglich ihres aktuellen Standortes. Jeder Satellit hat eine andere Entfernung zum Empfänger, was bedeutet, dass die Signale den Sensor nicht gleichzeitig, sondern zeitlich versetzt erreichen. Aus den Positionsdaten der Satelliten und den Zeitunterschieden der ankommenden Signale ist es dem Sensor möglich, seine eigene Position zu bestimmen. GPS Sensoren sind also passive Sensoren und dienen nur als Empfänger für die Signale der Satelliten. Für die Positionsbestimmung würde es reichen, wenn der Sensor gleichzeitig eine Verbindung zu drei GPS Satelliten haben würde. Die Zeitunterschiede beim Empfang der Signale sind allerdings sehr klein, meist im Nanosekundenbereich, sodass der vierte Satellit für einen Zeitabgleich verwendet werden muss, um eine genaueres Ergebnis zu haben.[27]

Um Sicherzugehen, dass alle GPS Satelliten die exakt gleiche Zeit haben und gleichzeitig die Signale aussenden, müssen die Satelliten regelmäßig von Bodenstationen kalibriert werden.[28]

[26] (Siegwart & Nourbakhsh, 2004, p. 105)
[27] (Siegwart & Nourbakhsh, 2004, p. 102f)
[28] (Siegwart & Nourbakhsh, 2004, p. 102)

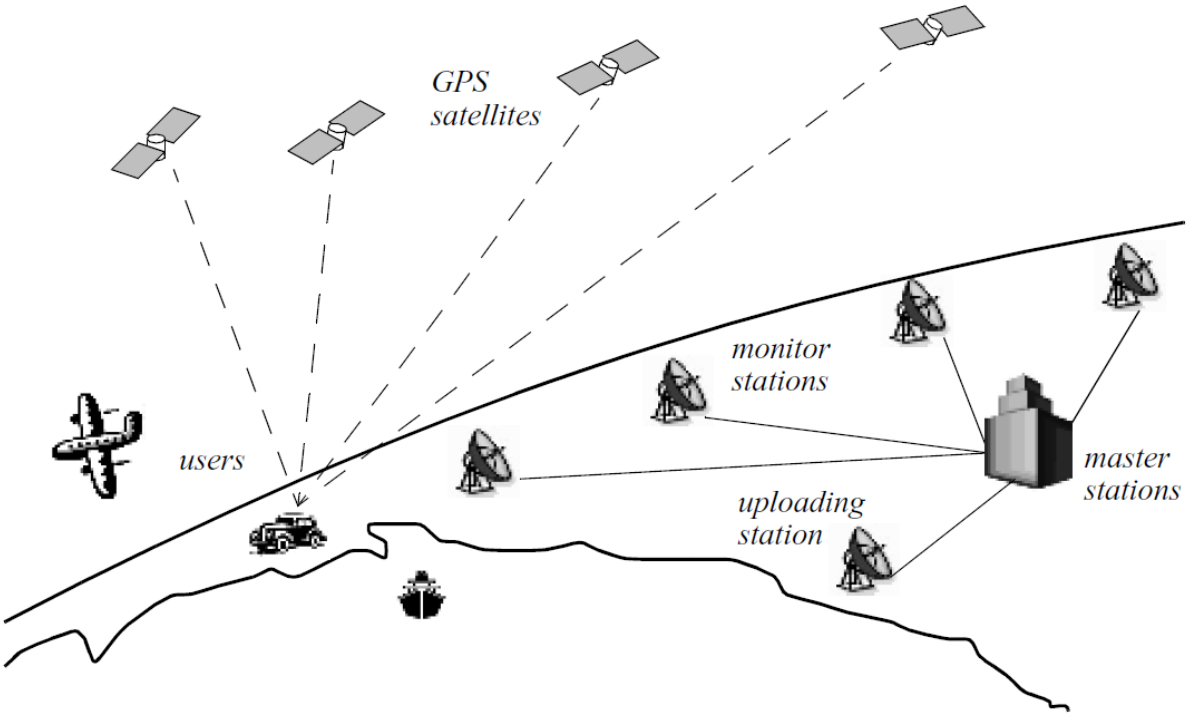


Abbildung 11: Nutzung des Global Positioning System[29]

Abbildung 11 skizziert die Nutzung des GPS-Systems. Dabei sind exemplarisch Satelliten, ein Nutzer, sowie Bodenstationen zur Überwachung der Satelliten aufgeführt.

2.2.5 Algorithmen-gestützte Kamerasensoren

Das Sehen ist der stärkste Sinn des Menschen. Er erlaubt es ihm, sehr viele und sehr genaue Informationen über seine Umwelt zu erlangen. Der visuelle Sinn wird auch für die Sensorik immer wichtiger, da aus visuellen Informationen mittels spezieller Algorithmen viele wichtige Informationen, wie Entfernung und Geschwindigkeit von sich bewegenden Objekten, ermittelt werden können. Kamerasensoren können außerdem dafür genutzt werden, ein genaues Modell der Umwelt aufzubauen. Anhand dieses Modells können sich Roboter beispielsweise gezielter in ihrer Umgebung bewegen und Hindernissen ausweichen.

[29] (Siegwart & Nourbakhsh, 2004, p. 102)

2.3 AR.DRONE 2.0

Die *AR.Drone 2.0* ist ein ferngesteuerter Quadrocopter aus dem Hause des französischen Hardwareherstellers Parrot SA. Die *AR.Drone 2.0* erschien 2012 und bietet gegenüber seinem Vorgänger eine deutlich verbesserte Hardware und verfügt über folgende technische Spezifikationen:

Tabelle 1: Technische Spezifikationen der AR.Drone2.0[30]

Kategorie	Unterkategorie	Spezifikation
Prozessor		1 GHz 32 bit ARM Cortex A8 mit einem 800 MHz Videoprozessor
Arbeitsspeicher		1GB DDR2 Ram mit 200 MHz
Betriebssystem		Linux 2.6.32
Kamera	Vorne	720p mit 30 fps
	Unten	QVGA mit 60 fps
Sensoren	Gyroskop	Dreiachsiges Gyroskop
	Magnetometer	Dreiachsiges Magnetometer
	Beschleunigungssensor	Dreiachsiger Beschleunigungssensor
	Abstandssensor	6 Meter Ultraschall
Konnektivität		Wi-Fi b/g/n
Motoren		4 bürstenlose Innenläufermotoren mit 14,5 Watt und 28.500 Umdrehungen pro Minute

[30] (Parrot SA)

Gesteuert wird die Drohne standardmäßig mithilfe einer Android oder iOS App auf einem geeigneten Smartphone. Zusätzlich stellt Parrot ein SDK zur Verfügung, mit dem Entwickler ihre eigenen Anwendungen zur Steuerung der Drohne programmieren können. Die Drohne fungiert als Access Point und baut ein eigenes WLAN Netzwerk auf, zu dem sich das Gerät mit der Steuerungssoftware verbinden kann. Das WLAN Netzwerk hat eine Reichweite von 50 Metern[31], somit beträgt die Reichweite der Drohne ebenfalls 50 Meter.

[31] (Parrot SA)

3 FUZZY-CONTROLLER

Bei diesem Projekt wurde ein Fuzzy-Controller eingesetzt. Der Controller steuert die Drohnenbewegungen auf Basis der Daten, welche er von der Kinect erhält. Dabei verbessert er das Flugverhalten der Drohne und ermöglicht eine für den Piloten intuitive Steuerung.

Dieses Kapitel beginnt mit einer Einführung in die Fuzzy-Regelung, wobei auf die Funktionsweise eines Fuzzy-Reglers, sowie auf die Möglichkeiten der Steuerung des Regelverhaltens eingegangen wird. Daraufhin wird der Aufbau des Fuzzy-Controllers, welcher für dieses Projekt realisiert wurde, beschrieben. Anschließend wird der Einfluss des Controllers auf das Flugverhalten der Drohne erläutert und evaluiert, ob er den Anforderungen genügte. Zu guter Letzt wird ein alternativer Lösungsansatz beschrieben, der vor der Umsetzung des Fuzzy-Controllers realisiert wurde.

3.1 GRUNDPRINZIPIEN DER FUZZY-REGELUNG

Die Fuzzy-Regelung basiert auf der 1965 von Lotfi Asker Zadeh entwickelten unscharfen Mengenlehre, der sogenannten *Fuzzy-Set-Theorie*. Diese Theorie liefert die nötige Mathematik, um mit Mengen zu rechnen, zu denen ein Wert auch nur teilweise gehören kann.[32] Der Grundgedanke der Fuzzy-Regelung besteht darin, die Steuerung eines Systems durch unscharfe Regeln zu beschreiben, anstatt das System selbst durch komplexe Differenzialgleichungen zu modellieren. Da Regeln für einen Menschen deutlich einfacher zu formulieren sind, ermöglicht es die Fuzzy-Regelung Expertenwissen auf intuitive Weise in eine automatische Prozesssteuerung zu überführen.

Die Fuzzy-Regelung findet durch drei Stufen statt, nämlich Fuzzifizierung, Inferenz und Defuzzifizierung.

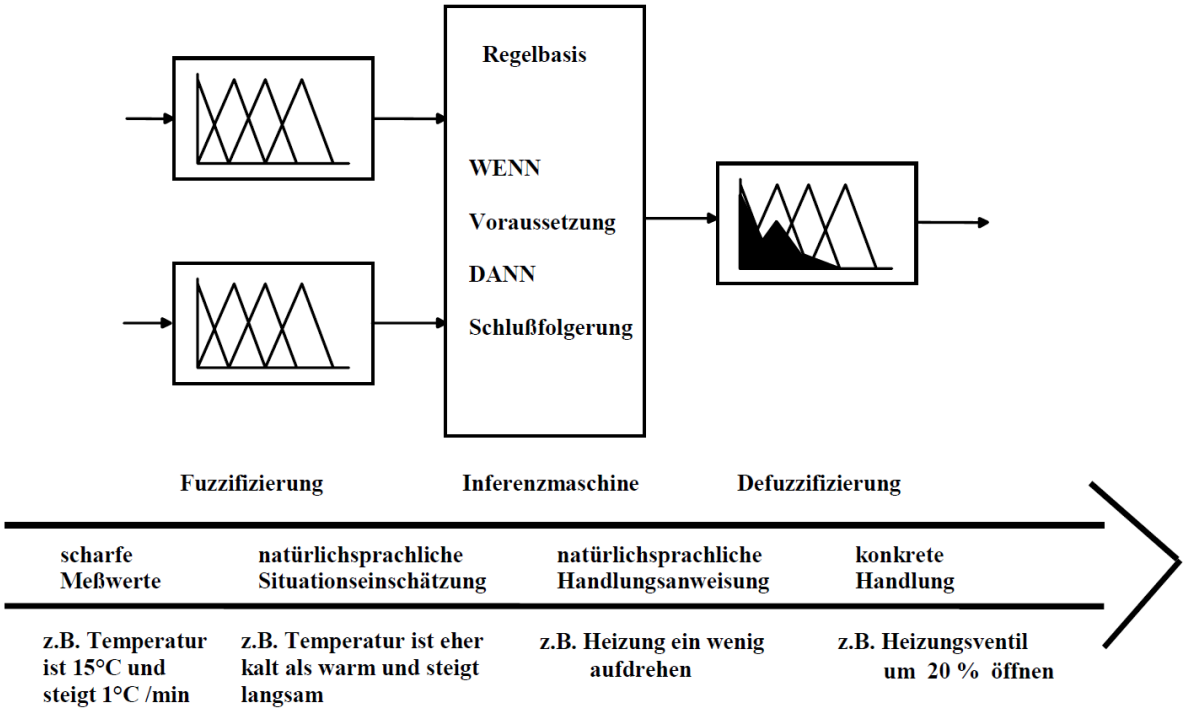


Abbildung 12: Grundprinzip Fuzzy-Regelung[33]

[32] (Zadeh, 1965, p. 338)
[33] (Keller, Video Fuzzy-Control zur Optimierung der thermischen Abfallbehandlung - ein Erfolgsbericht, S. 2)

Abbildung 12 zeigt das Grundprinzip der Fuzzy-Regelung anhand der drei zuvor angesprochenen Stufen am Beispiel einer Heizungssteuerung. Zuerst werden die scharfen Messwerte bestimmt. In diesem Fall die Temperatur (z.B. 15°C) und die Temperaturänderung (z.B. 1°C). Diese Messwerte werden bei der Fuzzifizierung in linguistische Terme umgewandelt. Hier ist die Temperatur eher kalt als warm und steigt langsam. Daraufhin wird mithilfe der Inferenzmaschine die Regelbasis evaluiert und die gültigen Schlussfolgerungen berechnet. In diesem Fall wäre die Schlussfolgerung die Heizung ein wenig aufzudrehen.

Bei der Defuzzifizierung wird diese unscharfe Beschreibung zu einem konkreten Wert bzw. einer konkreten Handlungsanweisung umgeformt. Hier z.B. muss das Heizungsventil um 20% geöffnet werden.

Die eben kurz beschriebenen Stufen, werden im Folgenden im Detail erläutert.

3.1.1 Fuzzifizierung

Bei der Fuzzifizierung werden die scharfen Messwerte in linguistische Terme umgewandelt. Hierfür wird für jeden linguistischen Term eine Zugehörigkeitsfunktion μ definiert. Diese Funktion beschreibt die Zugehörigkeit eines Wertes zu dem linguistischen Term im Bereich von 0 bis 1.
Also:

$$\mu_A: U \rightarrow [0,1] \text{ (}A\text{: Menge)}$$[34]

Die verschiedenen Zugehörigkeitsfunktionen gehören zu einer linguistischen Variable. Diese beschreibt logisch die Eingangsgröße. Mithilfe dieser partiellen Mengenzugehörigkeiten lässt sich aus einem scharfen Wert eine wage Beschreibung in linguistischen Termen formulieren, welche als Grundlage für die Inferenz zur Verfügung steht.

[34] (Keller, Wissenbasierte Systeme - Skript Vorlesung DHBW, 2014, S. 181)

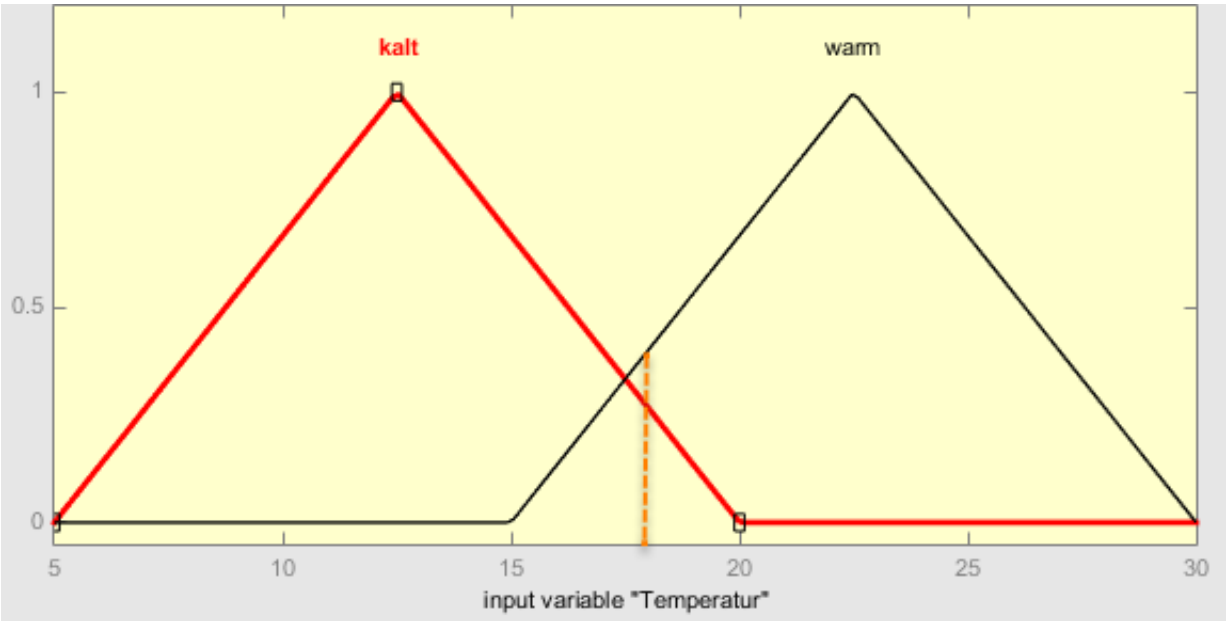


Abbildung 13: Linguistische Variable mit Zugehörigkeitsfunktionen

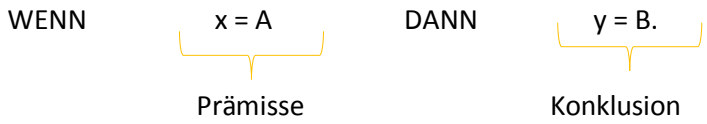
Bei dem zuvor beschriebenen Beispiel der Heizungssteuerung, wäre „Temperatur“ eine linguistische Variable und die dazu gehörenden Zugehörigkeitsfunktionen „Kalt“ und „Warm“. Diese linguistische Variable mit den dazugehörigen Zugehörigkeitsfunktionen ist in *Abbildung 13* zu erkennen. Auf der X-Achse ist die Temperatur in Grad Celsius und auf der Y-Achse die Zugehörigkeit zum linguistischen Term aufgetragen. Die orangene gepunktete Linie zeigt eine Stelle, bei der der Temperaturwert partiell zu beiden linguistischen Termen gehört. Dies ist in der Fuzzy Regelung möglich und sogar gewünscht. Hier ergibt sich durch die Fuzzifizierung für den durch die orange Linie markierten Eingangswert folgende linguistische Beschreibung: „Die Aussage Kalt stimmt zum Grad 0.3 und die Aussage Warm zum Grad 0.4“. [35]

[35] (Keller, Video Fuzzy-Control zur Optimierung der thermischen Abfallbehandlung - ein Erfolgsbericht)

3.1.2 Inferenz

Bei der Inferenz werden aus dem Ergebnis der Fuzzifizierung Schlussfolgerungen gezogen. Dies geschieht auf einer zuvor definierten Regelbasis. Zadeh hat diese Art des Folgerns als *unscharfes Schließen* bezeichnet.[36]

Die Regeln der Regelbasis sind als WENN-DANN-Regeln formuliert. Also in folgender Form:



Die Prämisse ist dabei die Voraussetzung, damit die Regel erfüllt werden kann und die Konklusion das, was aus der Regel folgt.[37]

Eine solche Regel, wird bei der Fuzzy-Implikation folgendermaßen verarbeitet:

„Regel R: *WENN x = A DANN y = B*
Faktum: *x besitzt mehr oder weniger die Eigenschaft A*

Schlussfolgerung: *y besitzt mehr oder weniger die Eigenschaft B“* [38]

In Anlehnung an die normale Implikation wird auch bei der Fuzzy-Implikation die folgende Schreibweise genutzt:

$$A \rightarrow B$$

Es stellt sich die Frage, wie diese Regel verarbeitet wird, da die Bedingung nur teilweise erfüllt sein kann. Mamdani beantwortete diese Frage damit, dass eine Konklusion maximal genauso wahr sein kann wie die Prämisse.[39]

Dies führt zu der Verwendung des Minimum-Operators, welcher das Minimum zweier Werte wählt.

$$A \rightarrow B: \quad \mu_{A \rightarrow B}(x,y) := \min\{\mu_A(x), \mu_B(y)\}$$ [40]

Bezogen auf das Heizungsbeispiel, wäre eine mögliche Regel:

[36] (Zadeh, 1965, pp. 338-353)
[37] (Schulz & Graf, 2008, S. 357)
[38] (Schulz & Graf, 2008, S. 357)
[39] (Schulz & Graf, 2008, S. 358)
[40] (Börcsök, 2000)

WENN Temperatur = Kalt DANN Heizung aufdrehen = Stark

Mit dem Fakt „Temperatur besitzt die Eigenschaft Kalt mit einem Grad von 0.3“ ergibt sich nach Mamdani die Konklusion: „Heizung aufdrehen besitzt die Eigenschaft Stark mit einem Grad von 0.3“.

Zadeh definierte für die üblichen Mengenoperatoren Disjunktion, Konjunktion und Komplement Berechnungsvorschriften, sodass diese Operatoren auch auf Fuzzy-Mengen angewendet werden können. Diese sind die folgenden:

Disjunktion:	$A \cup B$: $\mu_{A \cup B}(x)$	$:= \max\{ \mu_A(x),\ \mu_B(x) \}$
Konjunktion:	$A \cap B$: $\mu_{A \cap B}(x)$	$:= \min\{ \mu_A(x),\ \mu_B(x) \}$
Komplement:	A^c: $\mu_{A^c}(x)$	$:= 1 - \mu_A(\mathrm{x})$ [41]

Diese drei Operatoren finden sich auch in der Logik wieder. Dadurch kann eine Regel mehrere Prämissen haben, da sich diese mithilfe einer Konjunktion (∧) oder einer Disjunktion (∨) verknüpfen lassen. Zudem lassen sie sich auch negieren (¬), was dem Komplement bei Mengen entspricht.

Im Folgenden werden diese drei Operatoren im Detail vorgestellt.

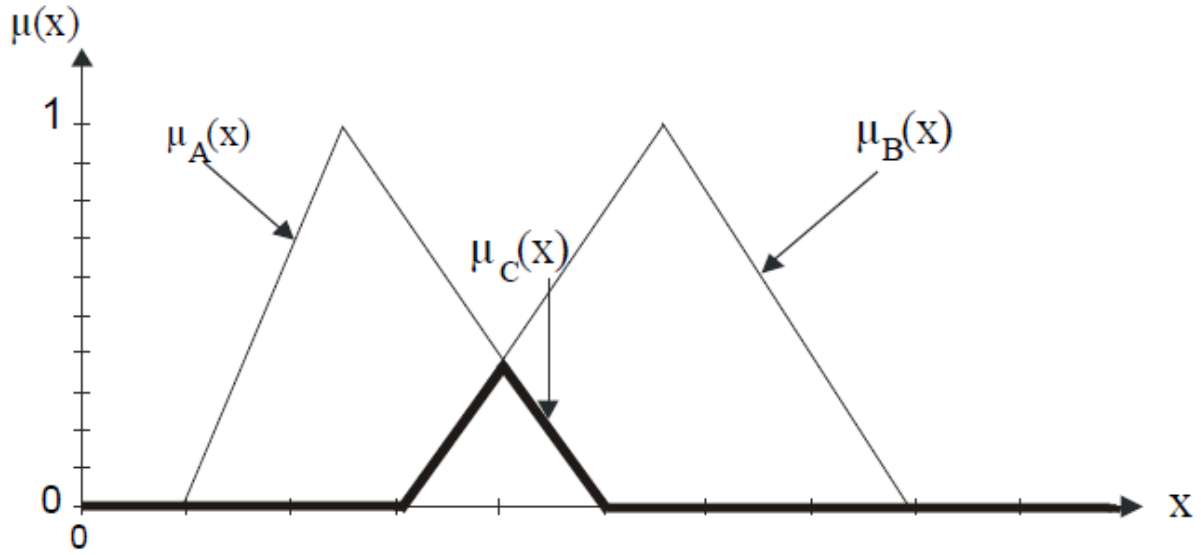


Abbildung 14: Min-Operator - Verknüpfung zweier Fuzzy-Mengen[42]

[41] (Zadeh, 1965)
[42] (Ottens & Jaouad, 2009/2010, S. 9)

In *Abbildung 14* ist die Anwendung des Min-Operators zu sehen. Hier werden die Fuzzy-Menge A und die Fuzzy-Menge B miteinander verknüpft. Zur Menge A gehört die Zugehörigkeitsfunktion μ_A und zur Menge B die Zugehörigkeitsfunktion μ_B . Das Ergebnis ist die Menge C mit der Zugehörigkeitsfunktion μ_c. Bei diesem Operator wird für jeden X-Wert der kleinste Wert der ursprünglichen Zugehörigkeitsfunktionen gewählt.

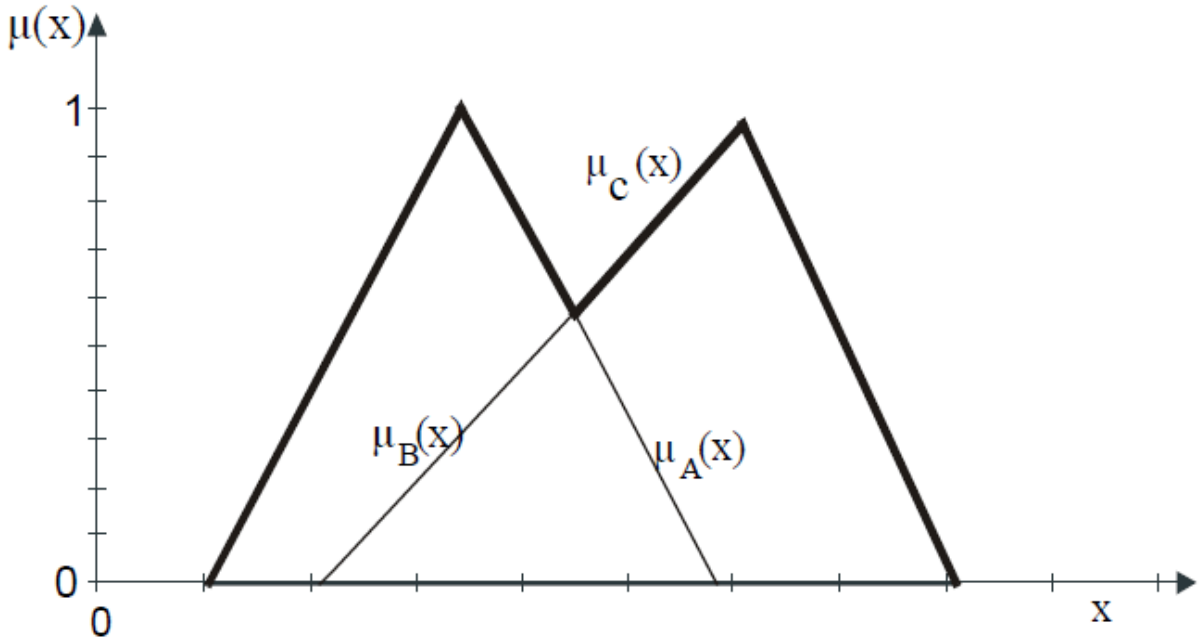


Abbildung 15: Max-Operator – Verknüpfung zweier Fuzzy-Mengen[43]

Abbildung 15 visualisiert den Max-Operator anhand der zwei Fuzzy-Mengen A und B mit ihren Zugehörigkeitsfunktionen μ_A und μ_B, die mit dem Max-Operator zu der Menge C verknüpft werden. Der Max-Operator ist optimistisch, da er für jeden X-Wert den höchsten Wert der Zugehörigkeitsfunktionen der miteinander zu verknüpfenden Fuzzy-Mengen wählt.

[43] (Ottens & Jaouad, 2009/2010, S. 11)

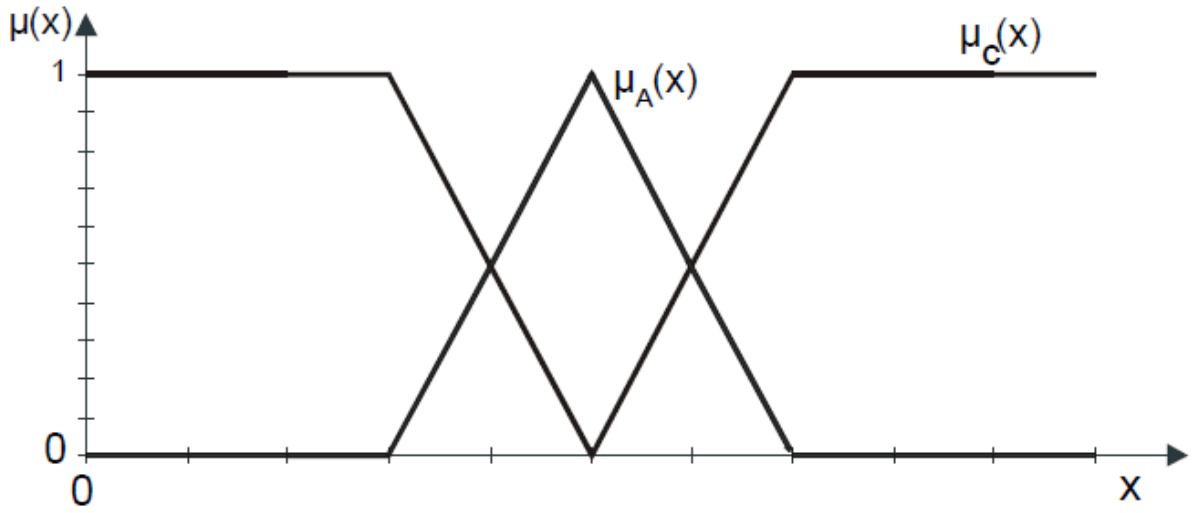


Abbildung 16: Komplement einer Fuzzy-Menge[44]

In *Abbildung 16* wird die Komplementbildung der Fuzzy-Menge A mit der Zugehörigkeitsfunktion μ_A visualisiert. Das resultierende Komplement ist hier die Fuzzy-Menge C mit der Zugehörigkeitsfunktion μ_C. Bei dieser Operation wird μ_A von 1 abgezogen.

Neben den Definitionen von Zadeh gibt es noch alternative Definitionen dieser Operatoren. Jedoch wird darauf nicht weiter eingegangen, da für dieses Projekt die Definitionen von Zadeh genutzt wurden. Zadehs Definitionen wurden eingesetzt, weil sie die gängigsten sind.

Als Ergebnis der Regelauswertung können sich widersprechende Schlussfolgerungen gezogen werden. Beispielsweise kann anhand einer Regel die Schlussfolgerung lauten: „Heizung aufdrehen = Stark mit Grad 0.3 erfüllt“, aber anhand einer anderen Regel: „Heizung aufdrehen = Stark mit Grad 0.5 erfüllt“ gezogen worden sein. Um dieses Problem zu lösen muss ein Operator für die Aggregation festgelegt werden. Übliche Operatoren hierfür sind das Maximum oder die Summe der widersprüchlichen Werte.

Als Ergebnis der Inferenz liegen die Zugehörigkeiten zu den linguistischen Termen der linguistischen Variablen, welche die Ausgangsgrößen beschreiben, vor.

[44] (Ottens & Jaouad, 2009/2010, S. 13)

3.1.3 Defuzzifizierung

Bei der Defuzzifizierung wird die unscharfe Beschreibung, die aus der Inferenz resultiert, wieder in scharfe Werte umgewandelt. Diese Werte sind die Stellgrößen des Systems.

Um diese Aufgabe zu erfüllen, können verschiedene Verfahren eingesetzt werden. Im Folgenden wird lediglich auf das Schwerpunktverfahren eingegangen, da es die besten Ergebnisse liefert und am häufigsten eingesetzt wird. Die Ergebnismenge beschreibt eine Fläche von welcher bei diesem Verfahren der Schwerpunkt gebildet wird. Anschließend wird das Lot an der Stelle des Schwerpunktes gefällt. Daraus ergibt sich folgende Formel:[45]

$$u_s = \frac{\int_{u_{min}}^{u_{max}} u * \mu_B(u) du}{\int_{u_{min}}^{u_{max}} \mu_B(u) du}$$ [46]

u_s : Abszissenwert des Flächenschwerpunktes
u_{min} : Beginn der Zugehörigkeitsfunktion
u_{max} : Ende der Zugehörigkeitsfunktion
B : Ergebnis Fuzzy-Menge der Inferenz (beliebig geformt)
μ_B : Zugehörigkeitsfunktion der Fuzzy-Menge B

Diese Formel ermöglicht die exakte Schwerpunktberechnung einer beliebigen Fläche. Allerdings müssen hierfür zwei Integrale gelöst werden, was ein sehr rechenintensives Unterfangen sein kann. Als Maßnahme kann der Schwerpunkt numerisch angenähert werden. Hierfür werden die Integrale durch gewichtete Summen genähert. Da die sich überlappenden Flächenanteile bei der Rechnung nicht ausgeschlossen werden, entsteht meist ein Fehler. Jedoch ist diese Näherung oft ausreichend.[47]

[45] (Schulz & Graf, 2008, S. 375)
[46] (Ottens & Jaouad, 2009/2010, S. 25)
[47] (Schulz & Graf, 2008, S. 375)

Daraus ergibt sich folgende Näherungsformel:

$$u_s \approx \frac{\sum_{i=1}^{q} u_i * \mu_B(u_i)}{\sum_{i=1}^{q} \mu_B(u_i)}$$ [48]

u_s	: Abszissenwert des Flächenschwerpunktes
q	: Anzahl der Abszissenstützpunkte
B	: Ergebnis Fuzzy-Menge der Inferenz (beliebig geformt)
μ_B	: Zugehörigkeitsfunktion der Fuzzy-Menge B

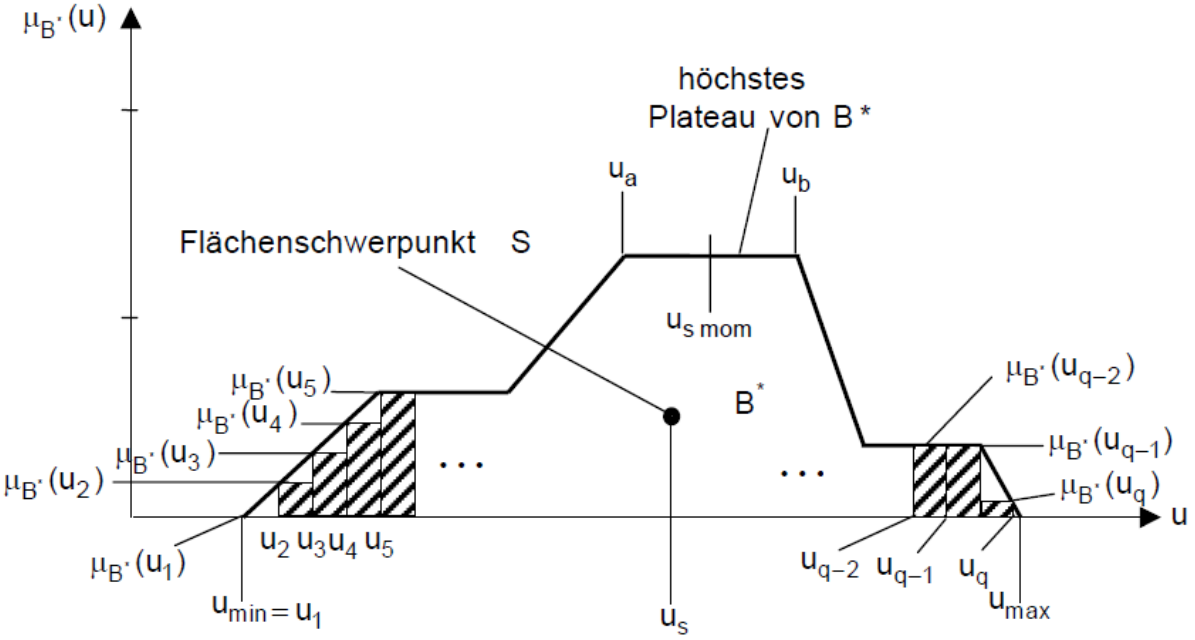


Abbildung 17: Defuzzifizierung nach der Schwerpunktmethode[49]

In *Abbildung 17* wird das Schwerpunktberechnungsverfahren visualisiert. Man sieht die einzelnen Abszissenstützpunkte und die daraus entstehenden Teilflächen, welche das Integral annähern. Des Weiteren ist die Zugehörigkeitsfunktion der aus dem Inferenzschritt resultierenden Fuzzy-Menge B zu erkennen, von der der Flächenschwerpunkt berechnet werden soll. Diese kann jede beliebige Form annehmen.

[48] (Ottens & Jaouad, 2009/2010, S. 25)
[49] (Ottens & Jaouad, 2009/2010, S. 24)

Je kleiner der Abstand $\Delta u = u_i - u_{i-1}$ zwischen den einzelnen Stützpunkten gewählt wird, desto genauer wird das Ergebnis, aber desto höher wird der Aufwand. Dies kann vor allem in Echtzeitsystemen zu Problemen führen.[50]

[50] (Ottens & Jaouad, 2009/2010, S. 25)

3.2 AUFBAU DES CONTROLLERS

Nachdem die Grundlagen der Fuzzy-Regelung erläutert wurden, wird in diesem Kapitel der Aufbau des für dieses Projekt eingesetzten Reglers beschrieben. Hierbei wurde beim Regler kein Feedback der Drohne in die Regelung mit aufgenommen, da die Drohne die Befehle auf abstraktem Niveau annimmt. Diese werden von einem eigenen Regler, welcher sich an Board der Drohne befindet, umgesetzt. Der Fuzzy-Controller in diesem Projekt hat den Zweck, das Zusammenspiel verschiedener Körperbewegungen in passende Steuerungsbefehle für die Drohne umzusetzen. Der Fuzzy-Controller wird eingesetzt, weil er die Möglichkeit eröffnet, die Bewegungstransformation in für Menschen verständliche Regeln zu formulieren.

3.2.1 Definition der Ein- und Ausgangsgrößen

Als erstes müssen die relevanten Ein- und Ausgangsgrößen für den Fuzzy-Controller bestimmt werden. Die Eingangsgrößen müssen so gewählt werden, dass sie die benötigten Informationen der Bewegung des Piloten enthalten. Die folgenden Schilderungen werden sich des Öfteren auf den *Ruhezustand* des Piloten beziehen. Hiermit ist der Zustand gemeint, in dem er frontal zum Kinect-Sensor steht, beide Arme im 90° Winkel von seinem Körper wegstreckt und aufrecht steht.

Alle Eingangsgrößen bewegen sich im Bereich von -1 bis 1. Der Wert 0 beschreibt den Ruhezustand und die beiden Randwerte, 1 und -1, die jeweiligen Maximalzustände der Bewegung. Es gibt vier für die Regelung relevante Größen und damit vier Eingänge.

Es wird mit der ersten relevanten Größe begonnen. Der Pilot kann seine Arme heben oder senken, um die Drohne steigen oder sinken zu lassen. Hierbei ist der interessante Wert der Abstand der Arme zum Ruhezustand. Sind seine Arme komplett am Körper angelegt, so beträgt der Wert -1. Hält er seine Arme senkrecht nach oben, so beträgt der Wert 1. Dieser Eingang wird als „up“ bezeichnet.

Die zweite relevante Größe ist die Drehung des Piloten um seine Z-Achse. Hierbei ist der interessante Wert der Winkel der Arme zwischen der aktuellen Position und dem Ruhezustand. Dreht der Pilot sich nach links bis er sich im 90° Winkel zum Ruhezustand befindet, entspricht dies dem Wert -1. Dreht er sich genausoweit nach rechts, entspricht dies dem Wert 1. Dieser Eingang trägt die Bezeichnung „rotation“.

Die dritte benötigte Größe ist die Neigung des Piloten zur Seite. Neigt er sich nach rechts, so nähert sich der Wert der 1 an. Neigt er sich nach links, so nähert sich der Wert der -1 an. Im Ruhezustand ist der Wert wieder 0. Dieser Eingang wird mit „sideward“ bezeichnet.

Die vierte und damit letzte Größe ist die Neigung nach vorne bzw. nach hinten. Bei der maximalen Neigung nach vorne, wird die -1 erreicht. Bei der maximalen Neigung nach hinten, wird die 1 erreicht. Da der positive Wert bei der Rückwärtsbewegung erreicht wird, trägt dieser Eingang den Namen „backward“.

Auf die Bewegungsmuster des Piloten wird näher in *Kapitel 5.3.1* eingegangen.

Nachdem die relevanten Eingänge besprochen wurden, müssen die benötigten Ausgänge ermittelt werden. Dies sind all jene Größen, welche die Drohne benötigt, um gesteuert zu werden. Es handelt sich ebenfalls um vier Größen. Ebenso haben die Ausgangsgrößen einen Bereich von -1 bis 1. Der Wert ist eine Prozentangabe von der Maximalgeschwindigkeit in der jeweiligen Bewegungsart der Drohne. Das Vorzeichen definiert dabei die Richtung der Bewegung. Ein Wert von 0 bedeutet keine Bewegung der Drohne in dieser Art der Bewegung.

Die erste Stellgröße ist die Geschwindigkeit nach oben oder unten. Ist der Wert positiv, richtet sich die Bewegung nach oben, bei einem negativen Wert nach unten. Dieser Ausgang im Fuzzy-Controller heißt „upSpeed“.

Die zweite Stellgröße, ist die Rotationsgeschwindigkeit um die eigene Achse. Eine Rotation nach rechts entspricht dem positiven Wert und eine Rotation nach links dem negativen. Der Name dieses Ausgangs ist „rotationSpeed“.

Der nächste Ausgang beschreibt die Seitwärtsbewegung der Drohne. Eine Bewegung nach rechts entspricht dem positiven Wert, eine Bewegung nach links dem negativen. Dieser Ausgang heißt „sidwardSpeed“.

Die vierte und letzte Stellgröße ist die Geschwindigkeit nach vorne bzw. hinten. Eine Bewegung nach hinten bedeutet einen positiven Wert und eine Bewegung nach vorne einen negativen. Der Name dieses Ausgangs lautet „backwardSpeed“.

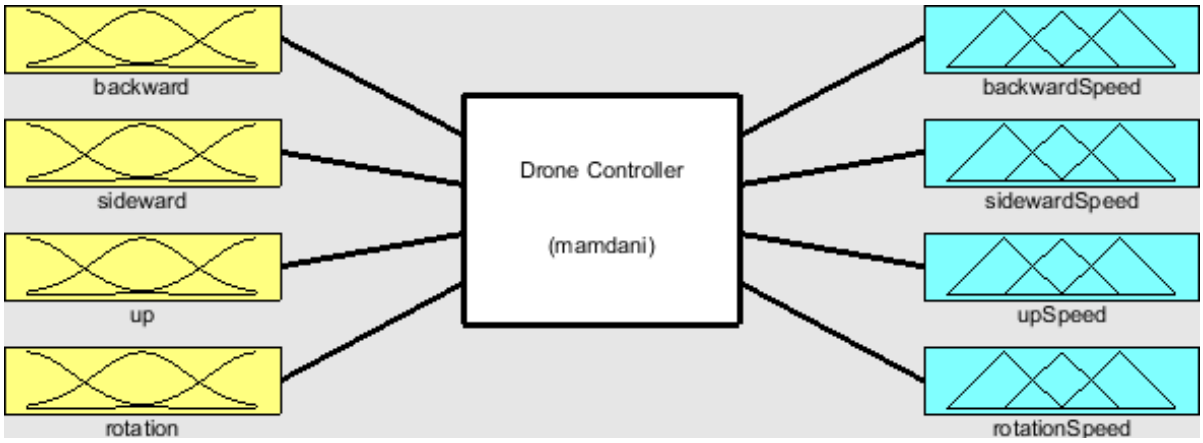


Abbildung 18: Drohnen Fuzzy-Controller - Übersicht der Ein- und Ausgangsgrößen

In *Abbildung 18* ist eine Übersicht über die benötigten Ein- und Ausgangsgrößen zu sehen. Bei den gelben Kästchen auf der linken Seite handelt es sich um die Eingangsgrößen und bei den blauen auf der rechten Seite um die Ausgangsgrößen. Der weiße Kasten in der Mitte visualisiert den eigentlichen Fuzzy-Controller. In dieser Abbildung ist zu erkennen, dass es sich bei den Ein- und Ausgangsgrößen eigentlich um die gleichen Größen handelt. Des Weiteren haben alle Größen einen Wertebereich von -1 bis 1. Dies legt die Überlegung nahe, der Fuzzy-Controller sei unnötig. Allerdings ist dies nicht korrekt, da der Fuzzy-Controller es ermöglicht, komplexe Flugeigenschaften der Drohne durch das Festlegen von Regeln, sowie die Anpassung der Zugehörigkeitsfunktionen auf einfache Art und Weise zu definieren. Im *Kapitel 3.3* wird auf die Auswirkungen des Fuzzy-Controllers auf die Flugeigenschaften der Drohne im Detail eingegangen.

Jeder linguistischen Variablen wurden fünf linguistische Terme zugewiesen. Die Zuweisung der linguistischen Terme zu den linguistischen Variablen und die jeweilige Bedeutung der Terme wird in der nachfolgenden *Tabelle 2* beschrieben.

Tabelle 2: Zuweisung linguistischer Terme zu linguistischen Variablen

Linguistische Variablen	Linguistischer Term	Bedeutung
backward	strongForward	Starke Vorwärtsbewegung
backwardSpeed	mediumForward	Vorwärtsbewegung
	zero	Keine Bewegung nach vorne oder hinten
	mediumBackward	Rückwärtsbewegung

	strongBackward	Starke Rückwärtsbewegung
sideward **sidewardSpeed**	strongLeft	Starke Seitwärtsbewegung nach links
	mediumLeft	Seitwärtsbewegung nach links
	zero	Keine Seitwärtsbewegung
	mediumRight	Seitwärtsbewegung nach rechts
	strongRight	Starke Seitwärtsbewegung nach rechts
up **upSpeed**	strongDown	Starke Bewegung nach unten
	mediumDown	Bewegung nach unten
	zero	Keine Bewegung hoch oder runter
	mediumUp	Bewegung nach oben
	strongUp	Starke Bewegung nach oben
rotation **rotationSpeed**	strongLeft	Starke Rotation nach links
	mediumLeft	Rotation nach links
	zero	Keine Rotationsbewegung
	mediumRight	Rotation nach rechts
	strongRight	Starke Rotation nach rechts

3.2.2 Wahl der Fuzzy-Operatoren

Bei diesem Projekt wurden die Eingangsgrößen nur mit dem UND-Operator verknüpft. Für den UND-Operator wurde, wie von Zadeh definiert, der Min-Operator genutzt. Die Implikation verwendet, wie von Mamdani beschrieben, auch den Min-Operator. Dadurch ist die Schlussfolgerung maximal so

wahr wie die Prämisse mit dem geringsten Erfüllungsgrad. Für die Aggregation der Schlussfolgerungen wird der Summen-Operator genutzt, da er alle Teilergebnisse berücksichtigt. Die Defuzzifizierung findet mit dem Schwerpunktverfahren statt. Dadurch werden stetige Ergebnisse erzielt.

3.2.3 Regelstrategie und Regelsatz

Das Verhalten des Fuzzy-Controllers entscheidet sich stark durch die definierten Regeln. Daher muss zuerst die Regelstrategie, sowie die daraus erwarteten Absichten erläutert werden.

Zuerst wurden die gleichnamigen linguistischen Terme der Eingangsvariablen auf die der Ausgangsvariablen übertragen. Hierfür wurde für jeden linguistischen Term jeder Eingangsvariablen eine Regel angelegt, welche denselben linguistischen Term als Konklusion für die entsprechende Ausgangsvariable besitzt. Die hierfür als zusammengehörend betrachteten Ein- und Ausgänge sind in der nachfolgenden *Tabelle 3* aufgeführt.

Tabelle 3: Mapping der Ein- und Ausgänge für die Regelmodellierung

Name der Eingangsvariablen	Name der Ausgangsvariablen
backward	backwardSpeed
sideward	sidewardSpeed
up	upSpeed
rotation	rotationSpeed

Eine Ausnahme ist die Rückwärtsbewegung, da sich bei Flugversuchen ohne Regler herausgestellt hat, dass es schwierig ist, die Drohne schnell rückwärts fliegen zu lassen. Dies liegt an dem eingeschränkten Winkel, um den sich ein Mensch nach hinten lehnen kann. Aus diesem Grund wurde der linguistische Term *mediumBackward* für die Eingangsvariable *backward* nicht nur nach dem soeben beschriebenen Schema als Regel aufgenommen. Stattdessen wurde für die Prämisse „*backward = mediumBackward*“ die Konklusion „*backwardSpeed = strongBackward*“ zusätzlich festgelegt. Dadurch hat ein leichtes Zurücklehnen einen deutlich größeren Einfluss. Generell wurde

ein Eins-zu-Eins-Mapping der Ein- und Ausgänge durchgeführt, um später durch die Modellierung der Zugehörigkeitsfunktionen genauere Einstellungen vornehmen zu können. Dazu jedoch mehr in *Kapitel 3.2.4*.

Der Fuzzy-Controller soll ermöglichen, dass die Drohne eine Kurve, inklusive Rotation um die eigene Achse in die entsprechende Richtung durchführt, wenn der Pilot sich nur nach vorne und zur Seite lehnt. Der Pilot muss nicht zusätzlich eine Rotationsbewegung durchführen. Dieser Effekt soll nur für den Vorwärtsflug gelten, da es beim Rückwärtsflug zu Verwirrung führen könnte. Zur Realisierung dieses Effektes wurden Regeln mit den Eingängen *backward* und *sideward* als Prämisse und den Ausgängen *backwardSpeed*, *sidewardSpeed* und *rotationSpeed* als Konklusion definiert. Die drei nachfolgenden Tabellen zeigen diese Regeln jeweils für einen der drei relevanten Ausgänge. In *Tabelle 4* ist die Regelübersicht für den Ausgang *backwardSpeed* zu sehen, in *Tabelle 5* die Regelübersicht für den Ausgang *sidewardSpeed* und in *Tabelle 6* die Regelübersicht für den Ausgang *rotationSpeed*. In den Tabellenspalten sind die linguistischen Terme für den Eingang *backward* aufgeführt. In den Tabellenzeilen die linguistischen Terme für den Eingang *sideward*. Die Zeilen und Spalten sind somit die Prämissen für eine Regel. In einer Zelle, die eindeutig durch eine Zeile und Spalte definiert wird, ist die Konklusion für eine Regel zu finden. Damit definiert jede Tabelle alle Regeln, die für den jeweiligen Ausgang im Zusammenhang mit dem Kurvenflug gelten. Benachbarte Zellen, die den gleichen Wert haben, wurden zu einer zusammengefasst und farblich hervorgehoben.

Tabelle 4: Regelübersicht für den Ausgang "backwardSpeed" für Kurvenflug

		backward		
		strongForward	***mediumForward***	***zero***
sideward	***strongLeft***	strongForward	mediumForward	zero
	mediumLeft			
	zero			
	mediumRight			
	strongRight			

Tabelle 5: Regelübersicht für den Ausgang "sidewardSpeed" für Kurvenflug

		backward		
		strongForward	***mediumForward***	***zero***
sideward	***strongLeft***	strongLeft		
	mediumLeft	mediumLeft		
	zero	zero		
	mediumRight	mediumRight		
	strongRight	strongRight		

Tabelle 6: Regelübersicht für den Ausgang "rotationSpeed" für Kurvenflug

		backward		
		strongForward	***mediumForward***	***zero***
sideward	***strongLeft***	strongLeft	strongLeft	-
	mediumLeft	strongLeft	mediumLeft	-
	zero	zero	zero	-
	mediumRight	strongRight	mediumRight	-
	strongRight	strongRight	strongRight	-

Die Spalte *zero* in der *Tabelle 6* enthält keine Werte, da auch beim Rückwärtsflug Zugehörigkeiten größer null für den linguistischen Term *zero* des Eingangs *backward* erreicht werden und dies für niedrige Rückwärtsfluggeschwindigkeiten zu dem Kurveneffekt führen würde. Dieser ist jedoch für Rückwärtsbewegungen nicht gewünscht. Als Ergebnis entstanden die nachfolgenden Regeln:

Vorwärts- und Rückwärtsbewegungen

WENN backward=strongForward DANN backwardSpeed=strongForward
WENN backward=mediumForward DANN backwardSpeed=mediumForward
WENN backward=zero DANN backwardSpeed=zero
WENN backward=mediumBackward DANN backwardSpeed=mediumBackward
WENN backward=mediumBackward DANN backwardSpeed=strongBackward
WENN backward=strongBackward DANN backwardSpeed=strongBackward

Seitwärtbewegungen

WENN sideward=strongLeft DANN sidewardSpeed=strongLeft
WENN sideward=mediumLeft DANN sidewardSpeed=mediumLeft
WENN sideward=zero DANN sidewardSpeed=zero
WENN sideward=mediumRight DANN sidewardSpeed=mediumRight
WENN sideward=strongRight DANN sidewardSpeed=strongRight

Aufwärts- und Abwärtsbewegungen

WENN up=strongDown DANN upSpeed=strongDown
WENN up=mediumDown DANN upSpeed=mediumDown
WENN up=zero DANN upSpeed=zero
WENN up=mediumUp DANN upSpeed=mediumUp
WENN up=strongUp DANN upSpeed=strongUp

Rotationsbewegungen

WENN rotation=strongLeft DANN rotationSpeed=strongLeft
WENN rotation=mediumLeft DANN rotationSpeed=mediumLeft
WENN rotation=zero DANN rotationSpeed=zero
WENN rotation=mediumRight DANN rotationSpeed=mediumRight
WENN rotation=strongRight DANN rotationSpeed=strongRight

Linkskurve

WENN backward=strongForward UND sideward=strongLeft DANN backwardSpeed=strongForward
WENN backward=strongForward UND sideward=strongLeft DANN sidewardSpeed=strongLeft
WENN backward=strongForward UND sideward=strongLeft DANN rotationSpeed=strongLeft

WENN backward=strongForward UND sideward=mediumLeft DANN backwardSpeed=strongForward;
WENN backward=strongForward UND sideward=mediumLeft DANN sidewardSpeed=mediumLeft
WENN backward=strongForward UND sideward=mediumLeft DANN rotationSpeed=strongLeft

WENN backward=mediumForward UND sideward=mediumLeft DANN backwardSpeed=mediumForward
WENN backward=mediumForward UND sideward=mediumLeft DANN sidewardSpeed=mediumLeft
WENN backward=mediumForward UND sideward=mediumLeft DANN rotationSpeed=mediumLeft

WENN backward=mediumForward UND sideward=strongLeft DANN backwardSpeed=mediumForward
WENN backward=mediumForward UND sideward=strongLeft DANN sidewardSpeed=strongLeft
WENN backward=mediumForward UND sideward=strongLeft DANN rotationSpeed=strongLeft

Rechtskurve

WENN backward=strongForward UND sideward=strongRight DANN backwardSpeed=strongForward
WENN backward=strongForward UND sideward=strongRight DANN sidewardSpeed=strongRight
WENN backward=strongForward UND sideward=strongRight DANN rotationSpeed=strongRight

WENN backward=strongForward UND sideward=mediumRight DANN backwardSpeed=strongForward
WENN backward=strongForward UND sideward=mediumRight DANN sidewardSpeed=mediumRight
WENN backward=strongForward UND sideward=mediumRight DANN rotationSpeed=strongRight

WENN backward=mediumForward UND sideward=mediumRight	DANN backwardSpeed=mediumForward
WENN backward=mediumForward UND sideward=mediumRight	DANN sidewardSpeed=mediumRight
WENN backward=mediumForward UND sideward=mediumRight	DANN rotationSpeed=mediumRight
WENN backward=mediumForward UND sideward=strongRight	DANN backwardSpeed=mediumForward
WENN backward=mediumForward UND sideward=strongRight	DANN sidewardSpeed=strongRight
WENN backward=mediumForward UND sideward=strongRight	DANN rotationSpeed=strongRight

Kurve (ohne Richtung)

WENN backward=strongForward UND sideward=zero	DANN backwardSpeed=strongForward
WENN backward=strongForward UND sideward=zero	DANN sidewardSpeed=zero
WENN backward=strongForward UND sideward=zero	DANN rotationSpeed=zero
WENN backward=mediumForward UND sideward=zero	DANN backwardSpeed=mediumForward
WENN backward=mediumForward UND sideward=zero	DANN sidewardSpeed=zero
WENN backward=mediumForward UND sideward=zero	DANN rotationSpeed=zero

3.2.4 Entwurf der Zugehörigkeitsfunktionen

Weitere Faktoren, welche das Verhalten des Fuzzy-Controllers stark beeinflussen, sind die Zugehörigkeitsfunktionen. Diese müssen so gewählt werden, dass sich das Verhalten, das durch die Regeln gewünscht ist, herauskristallisiert. Im Folgenden werden die entstandenen Zugehörigkeitsfunktionen, sowie der Weg dorthin erläutert. Des Weiteren werden das erwartete Verhalten des Quadrocopters und die daraus resultierenden Auswirkungen auf das Design der Zugehörigkeitsfunktionen erläutert.

Um die Modellierung der Zugehörigkeitsfunktionen nicht zu komplex werden zu lassen, wurden für alle Eingänge zunächst die gleichen Kurvenverläufe gewählt und später, falls nötig, konnten die Kurven für einzelne Eingänge angepasst werden. Das gilt auch für die Ausgänge. Somit mussten zunächst nur sechs Zugehörigkeitsfunktionen für die Eingänge und sechs Zugehörigkeitsfunktionen für die Ausgänge modelliert werden. In der Praxis stellte sich heraus, dass keine weiteren Anpassungen an den Zugehörigkeitsfunktionen für einzelne Eingänge vorgenommen werden mussten. Allerdings mussten die initial gewählten Zugehörigkeitsfunktionen angepasst werden, damit die gewünschten Effekte eintreten.

Generell war das Ziel, dass die Drohne auf leichte Bewegungen schwach reagiert, sobald die Bewegungsintensität jedoch zunimmt, die Drohne deutlich stärker auf leichte Veränderungen reagiert. Bei Annäherung an den Maximalwert von 1 bzw. -1 soll die Geschwindigkeitszunahme der Drohne für höhere Eingangswerte wieder abflachen. Die schwache Anfangsreaktion soll dem Piloten ermöglichen, die Drohne bei niedrigen Geschwindigkeiten präzise zu fliegen, z.B. für Landemanöver.

Bei stärkeren Gesten soll ihm ermöglicht werden, zügig an Geschwindigkeit zuzulegen. Die langsame Annäherung an die Maximalgeschwindigkeit soll den Piloten dazu befähigen, die Geschwindigkeit der Drohne auch bei hoher Geschwindigkeit unter Kontrolle zu haben.

Wie bereits angesprochen, sind die Zugehörigkeitsfunktionen für alle Eingänge dieselben. In der *Tabelle 7* werden die linguistischen Terme der Eingänge aufgeführt. Die in einer Zeile stehenden Terme haben dabei die identische Zugehörigkeitsfunktion.

Tabelle 7: Linguistische Terme der Eingänge mit identischen Zugehörigkeitsfunktionen

backward	sideward	up	rotation
strongForward	strongLeft	strongDown	strongLeft
mediumForwad	mediumLeft	mediumDown	mediumLeft
zero	zero	zero	zero
mediumBackward	mediumRight	mediumUp	mediumRight
strongBackward	strongRight	strongUp	strongRight

Nachfolgend werden, exemplarisch für alle Eingänge, die einzelnen Zugehörigkeitsfunktionen für den Eingang *backward* betrachtet. Die jeweils beschriebene Zugehörigkeitsfunktion wird rot markiert.

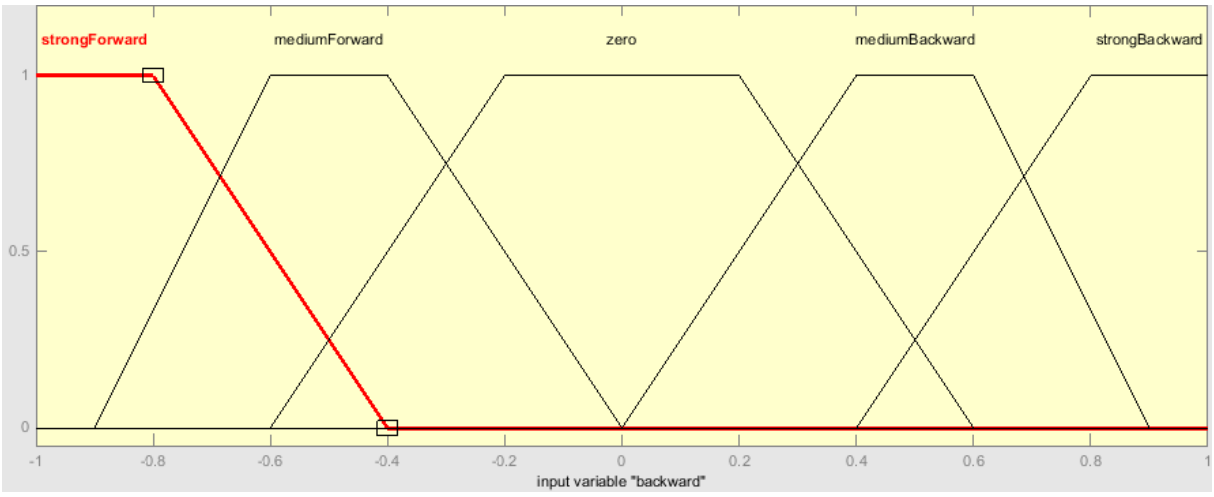


Abbildung 19: Zugehörigkeitsfunktion für „strongForward“ des Eingangs „backward“

In *Abbildung 19* ist die für den linguistischen Term *strongForward* definierte Zugehörigkeitsfunktion zu sehen. Es handelt sich um eine Trapezfunktion. Der Zugehörigkeitsgrad von 1 wird für Eingabewerte von -1 bis -0.8 erreicht. Anschließend nimmt der Grad der Zugehörigkeit stetig ab, bis schließlich bei einem Wert von -0.4 der Grad einen Wert von 0 erreicht und diesen hält bis zu einem Eingabewert von 1.

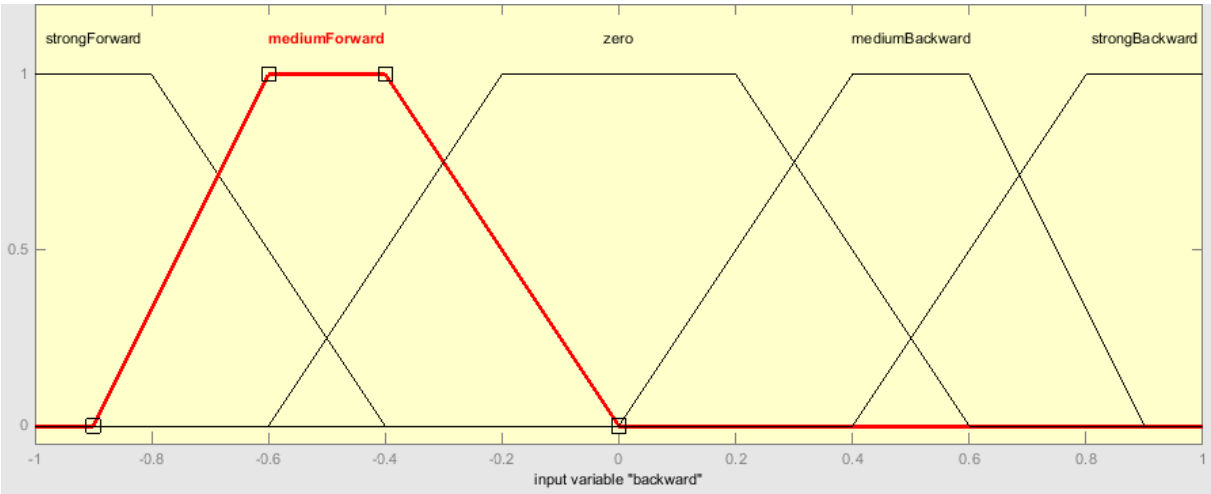


Abbildung 20: Zugehörigkeitsfunktion für „mediumForward“ des Eingangs „backward“

In *Abbildung 20* ist die für den linguistischen Term *mediumForward* definierte Zugehörigkeitsfunktion zu sehen. Auch bei dieser Zugehörigkeitsfunktion handelt es sich um eine Trapezfunktion. Diesmal ist das Trapez vollständig im Definitionsbereich zu erkennen. Diese Funktion hat ab einem Wert von -1 bis -0.9 einen Zugehörigkeitsgrad von 0. Ab einem Eingabewert -0.9 nimmt der Grad zu und erreicht die volle Zugehörigkeit von 1 für den Eingabewert -0.6. Bis zu einem Eingabewert von -0.4 bleibt die volle Zugehörigkeit erhalten. Ab dem Wert von -0.4, nimmt der Zugehörigkeitsgrad stetig ab, bis für den Eingabewert von 0 keine Zugehörigkeit zur Fuzzy-Menge *mediumForward* mehr besteht. Dies gilt auch für den Bereich von 0 bis 1.

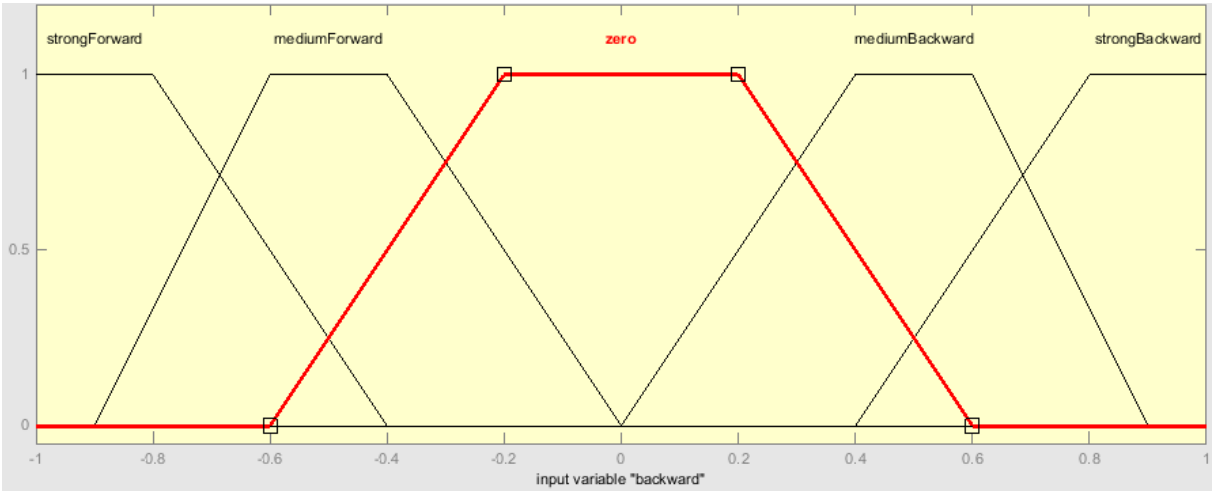


Abbildung 21: Zugehörigkeitsfunktion für „zero“ des Eingangs „backward“

In *Abbildung 21* ist die für den linguistischen Term *zero* definierte Zugehörigkeitsfunktion zu sehen. Auch bei dieser Funktion handelt es sich um eine trapezförmige. Sie hat ab -1 bis -0.6 einen Zugehörigkeitsgrad von 0. Ab dem Eingabewert von -0.6 nimmt der Grad zu und erreicht die volle Zugehörigkeit für den Wert -0.2. Die volle Zugehörigkeit erstreckt sich bis zum Wert von 0.2. Daraufhin nähert sich der Zugehörigkeitsgrad stetig der 0 bis zum Eingabewert von 0.6, wo der Grad von 0 erreicht wird und bis zum Wert 1 so bleibt.

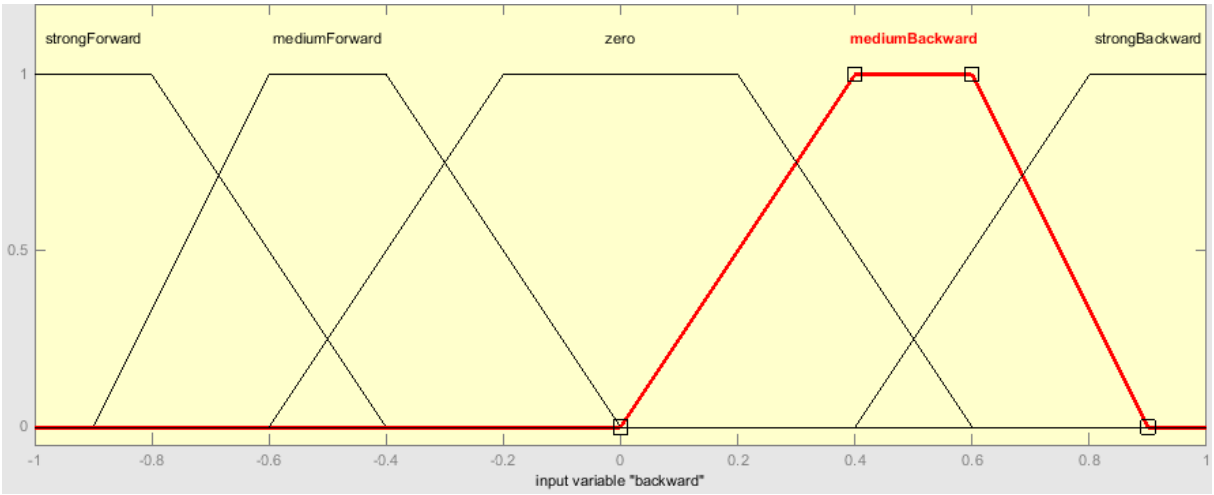


Abbildung 22: Zugehörigkeitsfunktion für „mediumBackward“ des Eingangs „backward“

In *Abbildung 22* ist die für den linguistischen Term *mediumBackward* definierte Zugehörigkeitsfunktion zu sehen. Hierbei handelt es sich wieder um eine Trapezfunktion. Es ist zu erkennen, dass diese Funktion den an der Y-Achse, an der Stelle 0, gespiegelten Kurvenverlauf der Zugehörigkeitsfunktion für die Menge *mediumForward* hat. Das bedeutet, sie hat ab dem Eingabewert -1 bis 0 einen Zugehörigkeitsgrad von 0. Dieser nimmt ab einem Wert von 0 ständig zu

bis er für den Wert 0.4 die volle Zugehörigkeit erreicht. Die volle Zugehörigkeit bleibt bis zu Eingabewerten von 0.6 erhalten und nimmt dann wieder ab bis an der Stelle 0.9 der Wert von 0 erreicht wird. Diese Zugehörigkeit gilt bis zur Stelle 1.

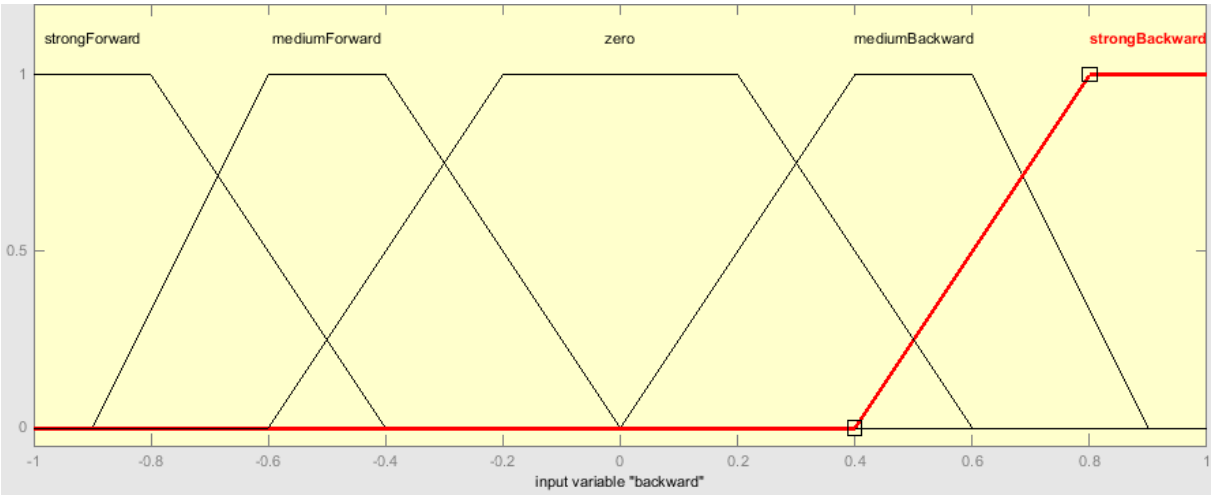


Abbildung 23: Zugehörigkeitsfunktion für „strongBackward“ des Eingangs „backward“

In *Abbildung 23* ist die für den linguistischen Term *strongBackward* definierte Zugehörigkeitsfunktion zu sehen. Auch diese Funktion ist eine Trapezfunktion und ist, wie die Funktion für *mediumBackward*, die an der Y-Achse gespiegelte Version der *strongForward* Funktion. Dadurch ergibt sich folgender Kurvenverlauf: Ab einem Eingabewert von -1 bis 0.4 ist keine Zugehörigkeit vorhanden. Der Grad der Zugehörigkeit nimmt anschließend zu, bis er für den Eingabewert von 0.8 die volle Zugehörigkeit erreicht. Diese bleibt erhalten bis zu einem Wert von 1. Der rechte Teil des Trapezes ist außerhalb des definierten Bereichs und somit nicht mehr relevant.

Diese Funktionen, wurden so gewählt, dass sie das zuvor beschriebene gewünschte Verhalten des Quadrocopters begünstigen. Dabei ist das Gesamtbild der Funktionen an der Y-Achse am X-Wert 0 gespiegelt. Dadurch ist gewährleistet, dass die Bewegung des Piloten in beide Bewegungsrichtungen einer Bewegungsart gleich stark eingeordnet wird. Des Weiteren sind die Zugehörigkeitsfunktionen der Medium-Terme, also *mediumForward* und *mediumBackward*, breiter zum Wert 0 hin aufgestellt. Das bedeutet die Zugehörigkeit in Richtung des Ruhezustandes des Piloten nimmt langsamer ab als in Richtung der Maximalwerte -1 und 1. Dies sorgt dafür, dass die Regeln der Medium-Terme noch mehr Einfluss im Ruhebereich des Piloten haben und für eine langsame Steigerung der Reaktivität der Drohne in diesem Bereich führen. Dieser Effekt wird zusätzlich durch die breit aufgestellte Zugehörigkeitsfunktion der *zero* Fuzzy-Menge verstärkt. Damit bei intensiveren Bewegungen des Piloten die Reaktivität wieder zunimmt, sind die Strong-Terme, also *strongForward* und

strongBackward, nicht so breit aufgestellt, wodurch der Einfluss der Regeln mit diesen Termen ab einem Wert von -0.4 bzw. 0.4 schnell zunimmt.

Nachdem die Zugehörigkeitsfunktionen der Eingänge betrachten wurden, werden im Folgenden die Zugehörigkeitsfunktionen der Ausgänge aufgeführt und erläutert. Wie schon bei den Eingängen, wurden auch hier für alle Ausgänge die gleichen Kurvenverläuft gewählt. Analog zu *Tabelle 7* werden in *Tabelle 8* die linguistischen Terme der Ausgänge mit identischer Zugehörigkeitsfunktion aufgeführt.

Tabelle 8: Linguistische Terme der Ausgänge mit identischen Zugehörigkeitsfunktionen

backwardSpeed	sidewardSpeed	upSpeed	rotationSpeed
strongForward	strongLeft	strongDown	strongLeft
mediumForwad	mediumLeft	mediumDown	mediumLeft
zero	zero	zero	zero
mediumBackward	mediumRight	mediumUp	mediumRight
strongBackward	strongRight	strongUp	strongRight

Nachfolgend werden, exemplarisch für alle Ausgänge, die einzelnen Zugehörigkeitsfunktionen für den Ausgang *backwardSpeed* betrachtet. Die jeweils beschriebene Zugehörigkeitsfunktion wird rot markiert.

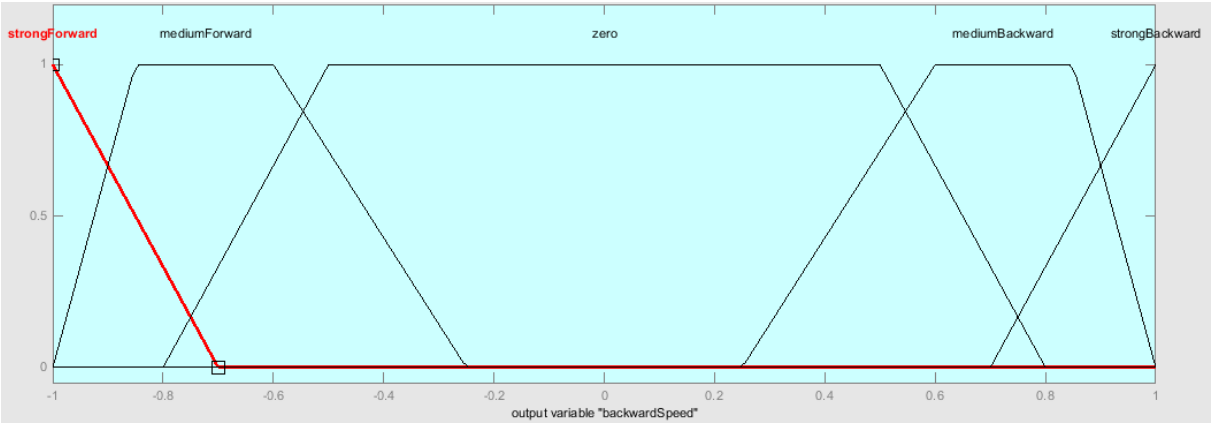


Abbildung 24: Zugehörigkeitsfunktion für „strongForward“ des Ausgangs „backwardSpeed“

In *Abbildung 24* ist die für den linguistischen Term *strongForward* definierte Zugehörigkeitsfunktion zu sehen. Bei dieser Funktion handelt es sich um eine Trapezfunktion, jedoch ist lediglich der rechte Abstieg des Trapezes noch im definierten Wertebereich. Hierdurch hat die Funktion für den Eingabewert -1 den höchsten Grad der Zugehörigkeit. Dieser nimmt kontinuierlich ab bis zum Wert -0.7 an dem keine Zugehörigkeit mehr vorliegt. Für den restlichen Wertebereich von -0.7 bis 1 liegt keine Zugehörigkeit mehr vor.

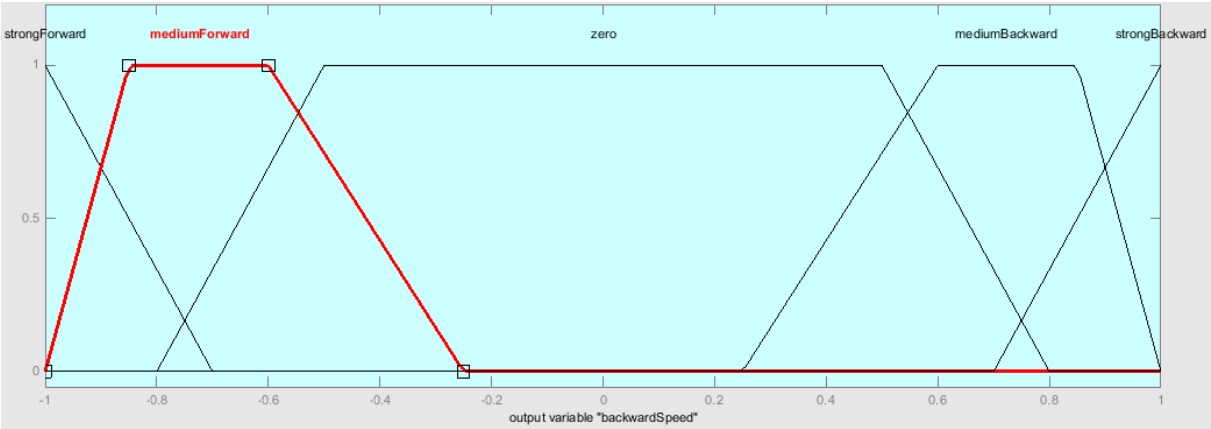


Abbildung 25: Zugehörigkeitsfunktion für „mediumForward“ des Ausgangs „backwardSpeed“

In *Abbildung 25* ist die für den linguistischen Term *mediumForward* definierte Zugehörigkeitsfunktion zu sehen. Bei dieser Funktion handelt es sich ebenfalls um eine Trapezfunktion. Sie beginnt am unteren Rand des Wertebereichs, also bei -1, mit einem Zugehörigkeitsgrad von 0. Ab dort steigt der Zugehörigkeitsgrad bis er am Eingabewert -0.85 die volle Zugehörigkeit von 1 erreicht. Die volle Zugehörigkeit bleibt bis zum Wert -0.6 erhalten. Ab diesem Wert nimmt der Zugehörigkeitsgrad wieder ab bis er für den Eingabewert von -0.25 den Grad von 0 erreicht. Der Grad von 0 bleibt für den restlichen Bereich, also bis zur Stelle 1, erhalten.

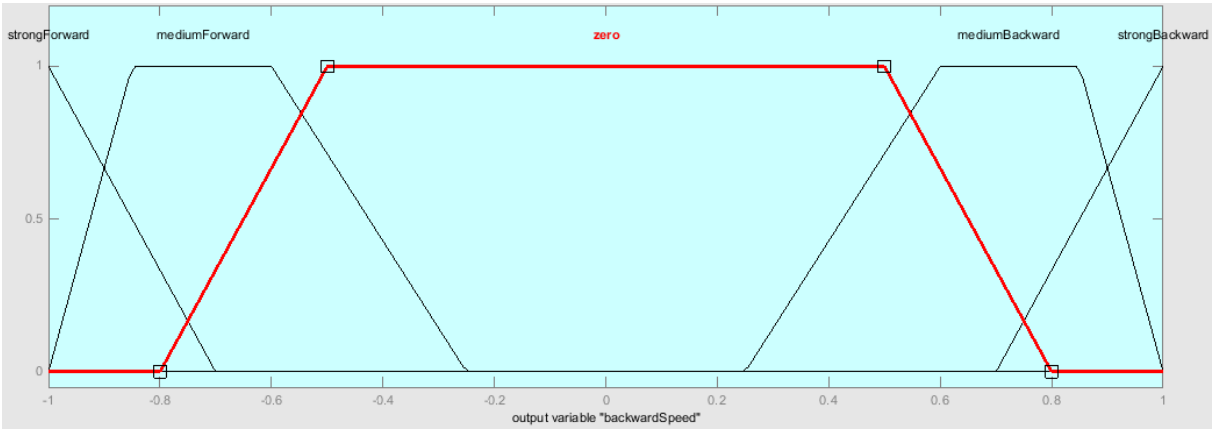


Abbildung 26: Zugehörigkeitsfunktion für „zero“ des Ausgangs „backwardSpeed“

In *Abbildung 26* ist die für den linguistischen Term *zero* definierte Zugehörigkeitsfunktion zu sehen. Auch bei dieser handelt es sich um eine Trapezfunktion. Diese hat einen Zugehörigkeitsgrad von 0 bis zur Stelle -0.8. Ab diesem Wert nimmt der Wert der Funktion stetig zu bis zur vollen Zugehörigkeit von 1 an der Stelle -0.5. Die volle Zugehörigkeit bleibt erhalten bis zur Stelle 0.5 und nimmt ab dort wieder langsam ab bis zur Stelle 0.8. Ab diesem Eingabewert von 0.8 bis zum Wert 1 ist keine Zugehörigkeit mehr vorhanden.

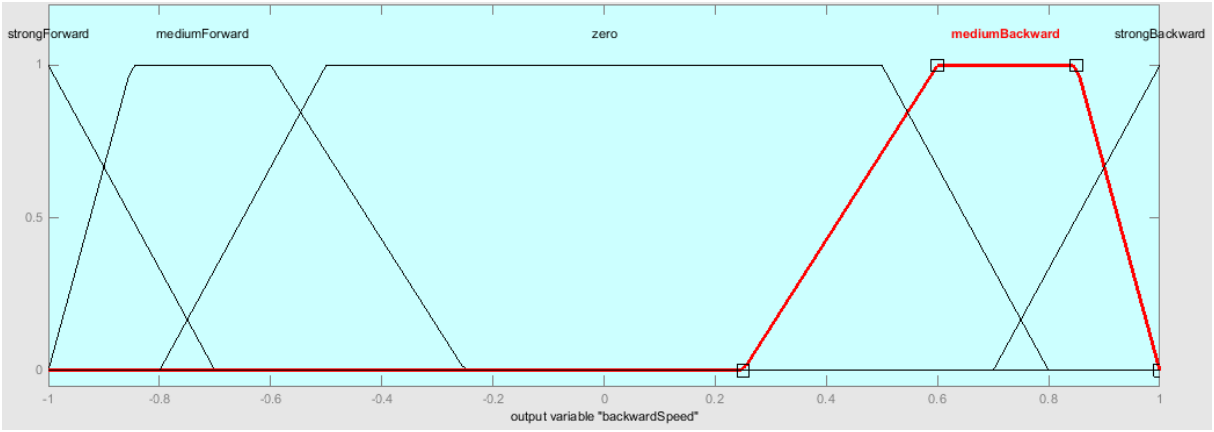


Abbildung 27: Zugehörigkeitsfunktion für „mediumBackward“ des Ausgangs „backwardSpeed“

In *Abbildung 27* ist die für den linguistischen Term *mediumBackward* definierte Zugehörigkeitsfunktion zu sehen. Diese trapezförmige Funktion hat einen Wert von 0 im Bereich von -1 bis 0.25. Ab diesem Eingabewert von 0.25 nimmt der Grad der Zugehörigkeit langsam zu bis die volle Zugehörigkeit an der Stelle 0.6 erreicht wird. Die volle Zugehörigkeit bleibt erhalten bis zur Stelle 0.85. Ab dieser Stelle nimmt der Wert kontinuierlich ab, bis zur Stelle 1, an dem er 0, also keine Zugehörigkeit erreicht. Weiterhin ist zu erwähnen, dass es sich bei dieser Funktion um die an der Y-

Achse, an der Stelle 0 der X-Achse, gespiegelte Zugehörigkeitsfunktion des linguistischen Terms *mediumForward* handelt.

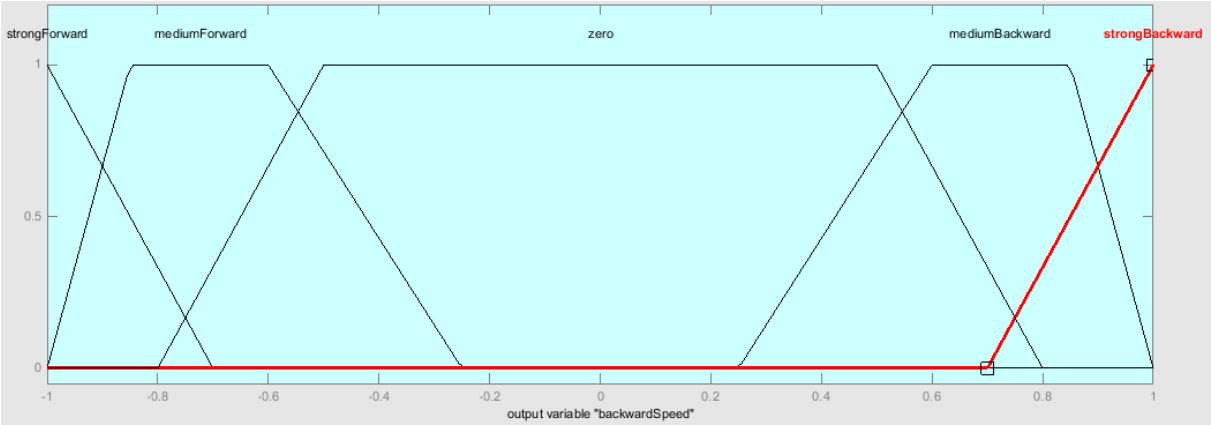


Abbildung 28: Zugehörigkeitsfunktion für „strongBackward“ des Ausgangs „backwardSpeed“

In *Abbildung 28* ist die für den linguistischen Term *strongBackward* definierte Zugehörigkeitsfunktion zu sehen. Bei dieser als Trapenzfunktion definierten Zugehörigkeitsfunktion, befindet sich lediglich der linke Aufstieg des Trapezes im Definitionsbereich. Im Bereich von -1 bis 0.7 hat die Funktion den Wert 0. Ab dem Eingabewert von 0.7 nimmt der Grad der Zugehörigkeit jedoch stetig zu bis zur Stelle 1, an der die volle Zugehörigkeit erreicht wird. Auch bei dieser Funktion handelt es sich um eine Spiegelung an der Y-Achse. Hier wurde die Zugehörigkeitsfunktion des Terms *strongForward* gespiegelt.

Die Zugehörigkeitsfunktionen der Ausgänge wurden so modelliert, dass die, am Anfang des Kapitels beschriebenen gewünschten Flugeigenschaften der Drohne erzielt werden. Hierfür wurde die Drohne wiederholt getestet und die Zugehörigkeitsfunktionen entsprechend der empirischen Daten angepasst. Dennoch lässt sich das Resultat anhand der Zugehörigkeitsfunktionen erklären. Die breite und damit großflächige Zugehörigkeitsfunktion des Terms *zero* sorgt dafür, dass die Reaktivität der Drohne in der Nähe des Ruhebereichs sehr gering ist. Dieser Effekt tritt auf, da es den Flächenschwerpunkt der Ergebnisfläche, selbst für geringe Erfüllungsgrade, stark zur null hin verschiebt. Die Medium-Terme, helfen dabei die Reaktivität in etwas von der Ruhelage weiter entfernten Bereichen zu verstärken, weil sie sich bei den Ausgängen deutlich weiter an den Randbereichen befinden als bei den Eingängen. Das bedeutet, die Einflüsse aus den Regeln der Medium-Terme bringen die Stellgrößen, selbst bei geringer Erfüllung, stärker in die Randbereiche als die Eingangsgröße dies verlangt. Hiergegen steht natürlich immer noch stark der Effekt des großen *zero*-Bereichs, welcher aber mit zunehmender Entfernung zum Ruhepunkt abflacht. Die

Zugehörigkeitsfunktionen der Strong-Terme wiederum sind sehr weit an die Ränder gedrängt und sehr steil. Dies führt dazu, dass auch noch hohe Stellgrößen, nahe der Maximalwerte -1 und 1 erreicht werden können. Das Erreichen der Randwerte ist generell problematisch bei Verwendung des Schwerpunktverfahrens.

3.3 AUSWIRKUNG AUF DIE FLUGEIGENSCHAFTEN DER DROHNE

In *Kapitel 3.2* wurde bereits erläutert, welches Flugverhalten der Drohne durch den Fuzzy-Controller gewünscht war. Zusammengefasst handelt es sich dabei um die folgenden Punkte:

1. Starke Reaktion der Drohne auf Rückwärtsbewegung des Piloten
2. Geringe Reaktivität im niedrigen Geschwindigkeitsbereich der Drohne, also in der Nähe der Ruheposition des Piloten
3. Hohe Reaktivität im mittleren Geschwindigkeitsbereich der Drohne
4. Geringe Reaktivität im hohen Geschwindigkeitsbereich der Drohne
5. Optimierter Kurvenflug, durch automatische Rotation der Drohne bei gleichzeitiger Seitwärts- und Vorwärtsbewegung des Piloten

Alle fünf Punkte konnten mithilfe des Fuzzy-Controllers umgesetzt werden. Der Einsatz des Fuzzy-Controllers war damit ein voller Erfolg.

3.4 Alternative Lösungsmöglichkeit

Bevor der Fuzzy-Controller im Projekt zum Einsatz kam, wurde auf eine naive Lösung des Problems zurückgegriffen. Wie in *Kapitel 3.1* beschrieben, erhält der Fuzzy-Controller Eingangssignale, welche „fuzzifiziert“ werden. Im Falle der Drohnenanwendung handelt es sich bei den Eingangssignalen um vorverarbeitete Daten aus dem Skeleton des Nutzers, das von einer Microsoft Kinect berechnet wird. Wie diese Vorverarbeitung genau aussieht, wird in *Kapitel 5.3.1* erläutert. Diese Eingangssignale werden im Fuzzy-Controller evaluiert und nach der Defuzzifizierung zu Steuerungsbefehlen der Drohne verarbeitet. Der Fuzzy-Controller verfolgt dabei das klassische EVA-Prinzip: Eingabe, Verarbeitung, Ausgabe, wobei die Ausgabe die Steuerungssignale für die Drohne ist.

Die alternative, naive Lösungsmöglichkeit unterscheidet sich nicht sehr stark vom EVA-Prinzip. Genau wie beim Fuzzy-Controller werden bei dieser Lösung vorverarbeitete Daten der Kinect entgegen genommen, in Stellsignale für die Drohne verarbeitet und ausgegeben. Statt hingegen linguistische Terme aufzustellen und über WENN-DANN-Regeln den Einfluss des Eingangswertes auf die Stellgröße zu definieren, wurden lediglich die vorverarbeiteten Eingangswerte mit einer logistischen Funktion verarbeitet.

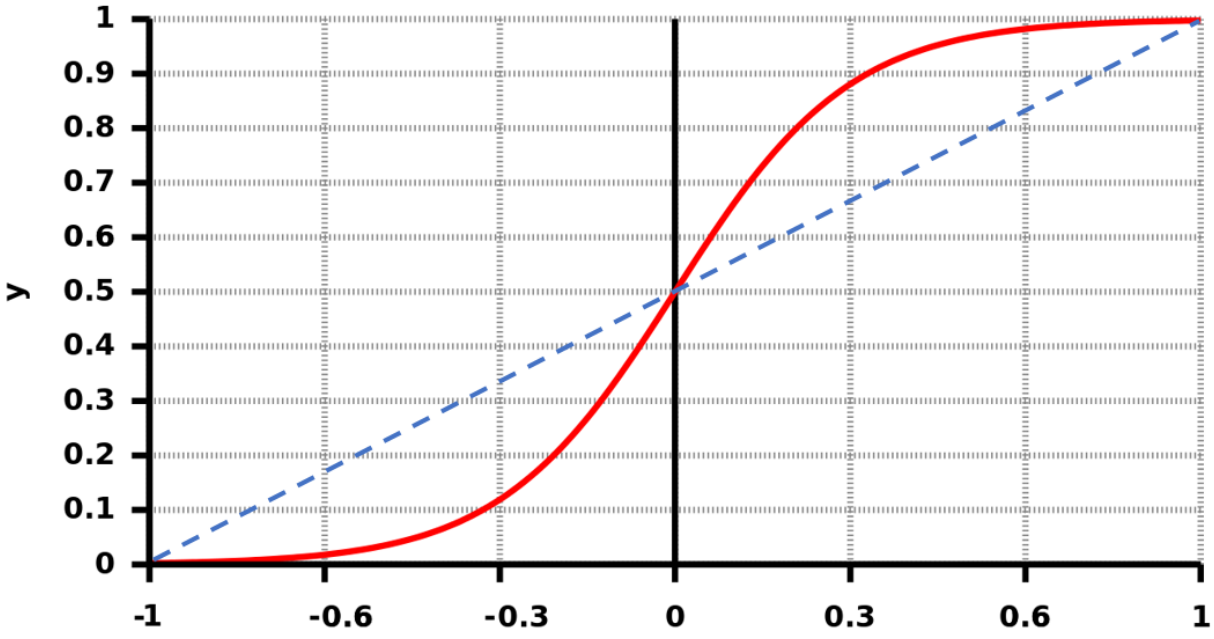


Abbildung 29: Verarbeitung der Eingangssignale (blau gestrichelt) in Stellgrößen für die Drohne (rot) anhand einer logistischen Funktion

Abbildung 29 zeigt, wie die Eingangssignale – in der Grafik blau gestrichelt – in die Steuerungsbefehle für die Drohne – als rot durchzogene Linie dargestellt – verarbeitet werden. Die Funktion, die lineare Eingangssignale in eine logistische Kurve umrechnet lautet folgendermaßen:

$$f(x) = \frac{S\ G}{S + (G - S)\ e^{-iGx}}$$

$S: untere\ Grenze$

$G: obere\ Grenze$

$i: Intensität\ der\ Verstärkung\ (Sensitivität)$

Der große Vorteil dieser Variante gegenüber des Fuzzy-Controllers ist die sehr einfache und schnelle Berechnung der Stellgrößen für die Drohne. Die Berechnung der Werte über die logistische Funktion bietet allerdings nicht die Flexibilität einer Feinjustierung der Stellwerte wie linguistische Terme des Fuzzy-Controllers. So konnten die Punkte 2, 3 und 4 aus Kapitel *„3.3 Auswirkung auf die Flugeigenschaften der Drohne“* erfüllt werden. Auch Punkt 1 „Starke Reaktion der Drohne auf Rückwärtsbewegung des Piloten“ konnte erfüllt werden, indem bei Rückwärtsbewegungen das Ausgabesignal um den Faktor 4 erhöht wurde. In der Praxis war die Drohne sehr anfällig für Störsignale, was dazu geführt hat, dass sie gelegentlich schwankte. Dieses Problem wurde durch eine Mittelwertberechnung der letzten 20 Stellgrößen gelöst. Lediglich für den Kurvenflug – Punkt 5 – hätte ein hoher Aufwand geleistet werden müssen, um ein akzeptables Ergebnis zu erzielen. Daher wurde für die Realisierung dieses Features ein Fuzzy-Controller implementiert.

4 USER EXPERIENCE

Bei der Software, die im Rahmen des Projektes entstanden ist, steht der Endnutzer im Zentrum des Geschehens. Ziel des Projektes war nicht nur eine Ansteuerung der Drohne mittels Gesten, die von Sensoren einer Microsoft Kinect erfasst werden, sondern vielmehr drehte sich die Entwicklung rund um das Thema „I believe I can fly“. Dabei soll der Nutzer das Gefühl vermittelt bekommen, er würde sich physisch in der Drohne befinden und selbst durch die Luft fliegen. „User Experience“, oder kurz UX, beschreibt in der Informationstechnik die Nutzererfahrung einer Software. Je besser die User Experience, desto besser ist die Software bedienbar. Im Rahmen des Projektes wird die Qualität der User Experience mit dem „I believe I can fly“-Gefühl gemessen. Je mehr der Nutzer das Gefühl hat, selbst durch die Luft zu fliegen, desto besser ist die User Experience der Applikation. Dieses Kapitel widmet sich der sukzessiven Realisierung des „I believe I can fly“-Gefühls im Projekt und den Technologien, die zur Umsetzung verwendet werden können.

4.1 WIE „I BELIEVE I CAN FLY“ IM PROJEKT UMGESETZT WURDE

Die Gestensteuerung ist die wichtigste Funktionalität der Software und ist unerlässlich für den Erfolg des Projektes. Wenn die Drohne nicht effizient gesteuert werden kann, hat der Nutzer kein Gefühl der vollständigen Kontrolle über die Drohne. Sobald das Gefühl der Kontrolle vermisst wird, verschwindet auch das Gefühl des aktiven Fliegens. Dem Nutzer kann zwar der Eindruck vermittelt werden, als fliege er in einem Quadrocopter mit, aber nicht als Pilot, was das eigentliche Ziel des Projektes sein sollte.

Aus diesem Grund wurde am Anfang des Projektes das Hauptaugenmerk auf die Kernfunktionalität der Software gelegt, nämlich der Ansteuerung der Drohne mittels Gesten, die von einer Microsoft Kinect aufgenommen werden. Microsoft stellt für die Entwicklung mit einer Microsoft Kinect ein SDK in .NET-Sprachen wie C#, Visual Basic und C++ bereit. Dieses SDK ermöglicht eine einfache Implementierung von „Skeletal Tracking“. Beim „Skeletal Tracking“ werden die menschlichen Freiheitsgrade, wie Schulter, Ellenbogen, Hüfte u.v.m. vom Kinect-Sensor erkannt. Das SDK erlaubt die Anmeldung von Event-Methoden, die zyklisch die Koordinaten dieser Freiheitsgrade, auch Joints genannt, wiedergeben. Diese Koordinaten können genutzt werden, um ein Ansteuerungsmodell der Drohne zu realisieren. Das Ansteuerungsmodell beschreibt, wie die Koordinaten interpretiert werden und in Befehle für die Drohne umgewandelt werden. In *Kapitel 5.3.1* wird auf das Ansteuerungsmodell im Detail eingegangen.

Um die Drohne in Bewegung zu setzen, müssen die Befehle an die Drohne gesendet werden. Die *AR.Drone 2.0* kommuniziert über ein drahtloses Netzwerk, das sie selbst öffnet, sobald sie an einen Akku angeschlossen wird. Die Befehle können daher über eine UDP/IP-Verbindung zwischen einem PC, der an eine Microsoft Kinect angeschlossen ist, und der Flugdrohne übertragen werden. Im *Kapitel 5.1* wird genauer auf die technische Realisierung eingegangen.

In der ersten Version der Software wurden die Elemente „Befehlsermittlung über Skeletal Tracking“ und „Datenverbindung mit der Drohne“ vereint und alle vorhandenen Information über den Bildschirm ausgegeben. Das User Interface (UI) wurde mit HTML5 und JavaScript umgesetzt, da alle Projektmitglieder für diese Technologie das meiste Know-How besaßen und ein schnelles Ergebnis erzielt werden sollte. Die Navigationsbefehle für die Drohne und ein Videobild der Kinect wurden über Websockets an den Browser des Clients übertragen und dort zur Anzeige gebracht. Die erste Version wurde daher auch „Webcockpit“ genannt.

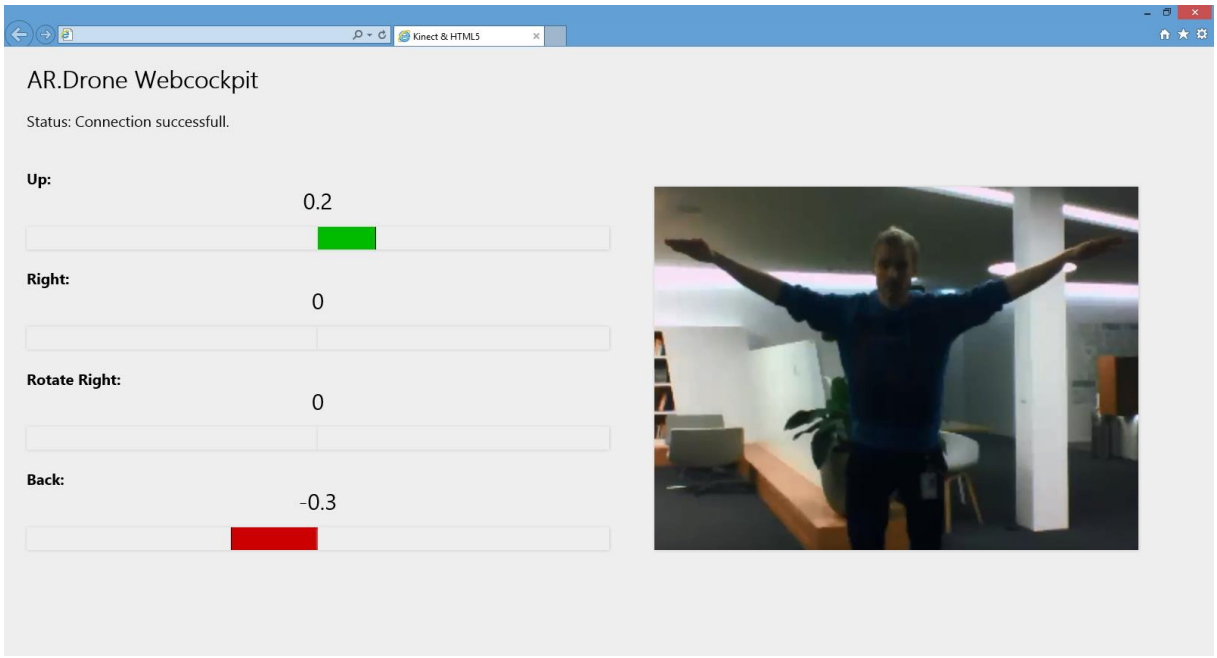


Abbildung 30: Die Ansicht des Webcockpits

Abbildung 30 zeigt die erste Version der „I believe I can fly“-Anwendung, das Webcockpit. Über das Webcockpit bekommt der Benutzer Rückmeldung über die Steuerbefehle für die Drohne und kann interaktiv seine Gesten anpassen, um die Drohne in die gewünschte Richtung zu lenken. Ein „I believe I can fly“-Gefühl kommt bei dieser Version leider nur über die Tatsache zustande, dass die Drohne den Bewegungen des Körpers folgt und der Nutzer so das Gefühl der Kontrolle über die Drohne besitzt. Das Gefühl, sich selbst im Cockpit der Drohne zu befinden, kommt beim Benutzer nicht an. Auch die Hinzunahme des Videobildes der Drohne zum Webcockpit hat die UX nur marginal verbessert. Die Anwendung benötigte daher ein von Grund auf neues Design.

Nachdem in der ersten Version die Kernfunktionalität bereitgestellt wurde, bestand der Fokus der zweiten Version in der Verbesserung des Gefühls, sich in der Drohne zu befinden, anstatt auf dem Boden zu sein. C# bietet mit dem „Window Presentation Foundation“ (WPF)-Framework eine umfangreiche Bibliothek, die es Entwicklern mit einfachen Elementen ermöglicht, authentische Fensteranwendungen zu erschaffen. Da die komplette Backend-Funktionalität bereits mit C# implementiert wurde und das WPF-Framework zur Verfügung steht, wurde entschieden, das Frontend mit C# zu entwickeln. Das „I believe I can fly“-Gefühl soll in der zweiten Version authentisch wirken. Das Gefühl zu erwecken, in einem echten Cockpit zu sitzen, wird erreicht, wenn der Nutzer ein scheinbar echtes Cockpit vor Augen hat. Das User Interface dieser Version orientierte sich daher an sogenannten „Head-Up-Displays“ (HUDs).

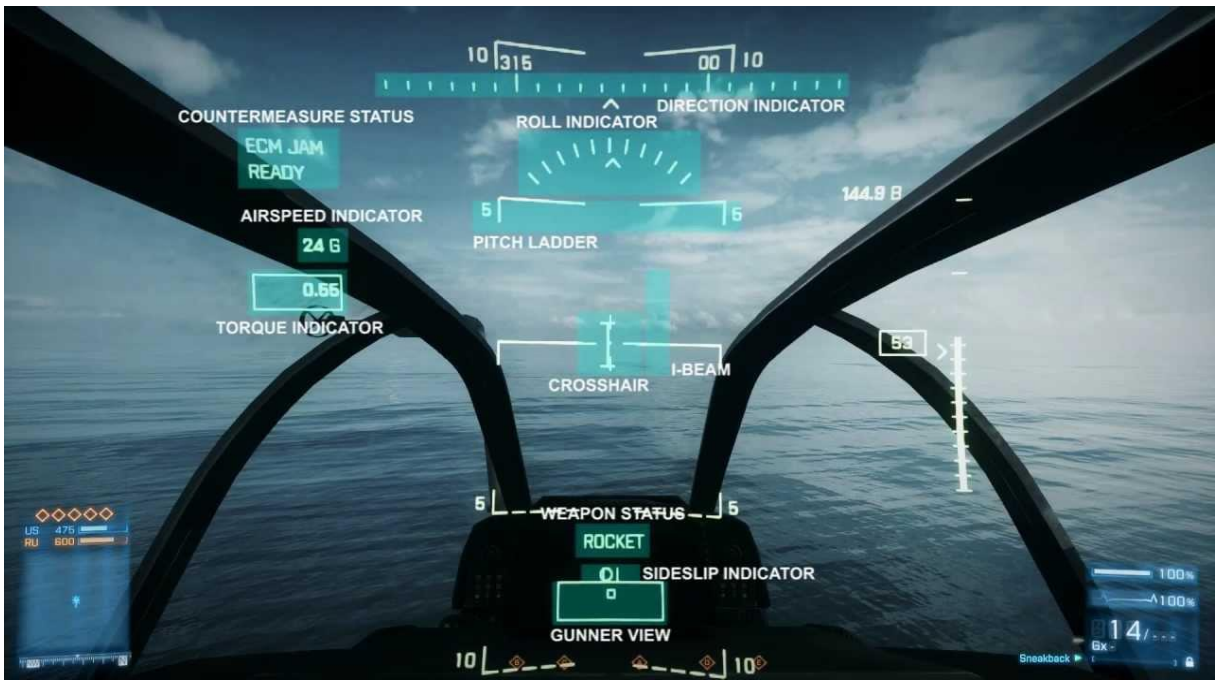


Abbildung 31: Head-Up-Display eine Helicopters in einem Spiel[51]

Ein HUD ist eine Ansicht, die zusätzliche Information, z.B. Steuerelemente, in das Sichtfeld einer Person hineinprojiziert. Ein gutes HUD verschmilzt alle notwendigen Bedien- und Steuerinformationen des Gerätes in das Blickfeld, sodass diese Informationen bei dessen Steuerung förderlich sind und zusätzlich gut aussehen. *Abbildung 31* zeigt ein beispielhaftes Head-up-Display aus dem Spiel *Battlefield 4*. Für die Implementierung mit der *AR.Drone 2.0* sind vor allem die Informationen „Roll“, „Pitch“ und „Yaw“ interessant, da sie die Lage des Quadrocopters im Raum beschreiben. Zusätzlich sind der Ladestand des Akkus und die Signalstärke des WLAN-Signals wissenswerte Informationen, die ebenfalls im HUD untergebracht werden sollten. Beim Design des Head-Up-Displays für die eigene Anwendung wurde das Bild aus *Abbildung 31* zum Vorbild genommen.

[51] Screenshot aus dem Spiel “Battlefield 4” von Electronic Arts

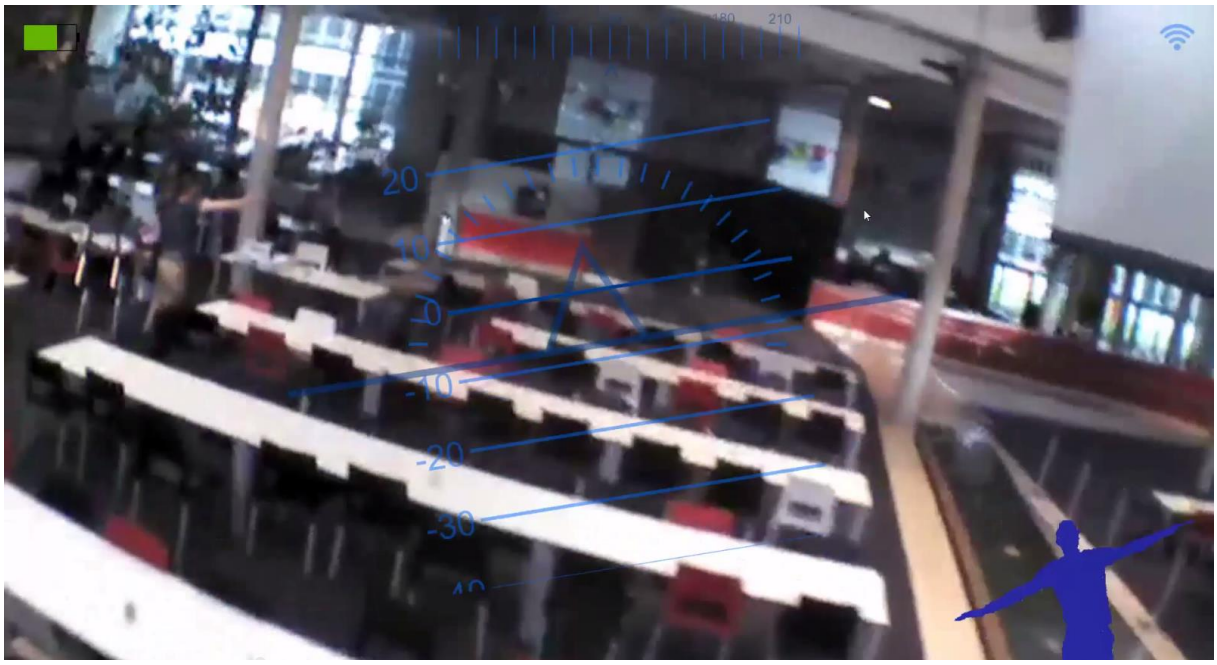


Abbildung 32: Das Head-Up-Display der "I believe I can fly"-Anwendung

Abbildung 32 ist das Resultat der Entwicklung eines eigenen HUDs für den Quadrocopter. Neben der Kontrolle über die Drohne, bekommt er in dieser Version zusätzlich mit einem HUD die Ansicht eines authentischen Cockpits des Quadrocopters angezeigt. In der Praxis zeigt sich allerdings, dass Nutzer nicht auf die Cockpit-Ansicht auf dem Bildschirm achten. Vielmehr richtet sich die Aufmerksamkeit des Nutzers auf die Flugdrohne, die durch die Luft fliegt. Fliegt der Nutzer mit der Drohne eine Kurve, besteht die Gefahr, dass der Nutzer der Drohne hinterherschaut und sich rotiert. Die Rotation wird von der Anwendung wiederum als Steuerbefehl für die Drohne interpretiert, obwohl der Nutzer keine Rotation der Drohne wollte. Richtet der Nutzer hingegen seine volle Aufmerksamkeit auf das HUD, treten solche missverständlichen Steuerbefehle seltener auf. Da die Kamera der Drohne nur nach vorne gerichtet ist, hat der Nutzer nur den vorderen Teil der Drohne im Blick. Er ist dazu geneigt, sich zu der Drohne zuzuwenden, um zu beobachten, was sich neben und hinter der Drohne für mögliche Hindernisse befinden.

Die fliegende Drohne im Raum zog zu viel Aufmerksamkeit an sich, weshalb sich der Nutzer nicht mehr vollständig auf das Head-Up-Display konzentrierte, das zu dem Zweck entworfen wurde das „I believe I can fly“-Gefühl zu verbessern. Es musste daher eine Möglichkeit geschaffen werden, die den Nutzer zwingt die ganze Zeit auf das Cockpit zu achten, um versehentliche Steuerungsbefehle zu vermeiden. An dieser Stelle kommen AR-Brillen ins Spiel. Das HUD, das zuvor auf einem Computermonitor zu sehen war, kann mit AR-Brillen wenige Zentimeter vor dem Auge zur Anzeige

gebracht werden, sodass der Nutzer keine andere Wahl mehr hat, als den Quadrocopter über die Cockpit-Ansicht zu fliegen.

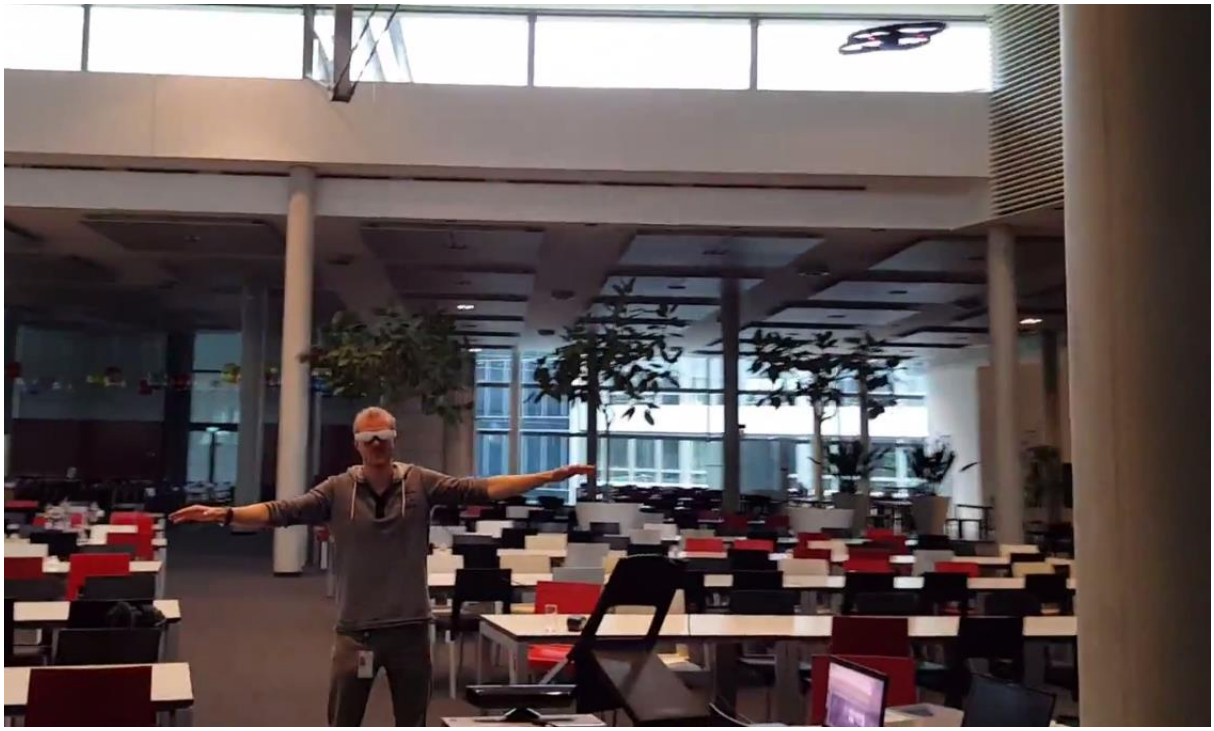


Abbildung 33: AR-Brillen zwingen den Nutzer zur Ansicht des Cockpits

Abbildung 33 zeigt den Flugversuch mit einer AR-Brille. Der Nutzer hat keine Möglichkeit, nach der Drohne zu schauen, da sich die Brille permanent vor den Augen befindet und das Cockpit der Drohne anzeigt. Nach wenigen Minuten Gewöhnungszeit haben sich Nutzer mit AR-Brillen an die Ansicht gewohnt und vermissen es nicht, nach der Drohne Ausschau zu halten. Erst jetzt kommt das eigentliche „I believe I can fly“-Gefühl zustande, da der Nutzer aus seiner Perspektive keinerlei Informationen mehr über die Außenwelt besitzt. Für den Nutzer wirkt es mit diesem Aufbau der Anwendung, dass er sich in der Perspektive der Drohne durch den Raum bewegt.

In dieser Version gab es lediglich einen Aspekt, der die User Experience noch negativ beeinflusste. In den bisherigen Versionen wurde davon ausgegangen, dass alle Komponenten funktionsfähig waren, bevor die Anwendung gestartet wurde. Dazu gehört, dass die Microsoft Kinect mit Strom versorgt und an den Computer angeschlossen, die Drohne mit dem Akku und der PC mit dem Netzwerk der Drohne verbunden wurden. War eines davon nicht der Fall oder war die Drohne nach einem Sturz im „Emergency Mode“, musste die Anwendung neu gestartet werden. Eine weitere Schwierigkeit bestand im Umgang mit verschiedenen Körpergrößen und Körperhaltungen von Nutzern. Damit ein Nutzer mit längeren Armen beim Heben und Senken der Arme denselben Ausschlag bei der Drohne

erreicht, wie ein Nutzer mit kürzeren Armen, muss die Anwendung vor dem Start der Drohne die Armlänge des Nutzers erfassen. Ebenso muss die Körperhaltung bestimmt werden. Für die Vor- und Rückwärtsbewegung beugt sich der Nutzer nach vorne oder nach hinten. Die Anwendung muss wissen, bei welcher Körperhaltung der Nutzer aufrecht steht, sodass die Drohne ebenfalls stehen bleibt, wenn sich der Nutzer im Ruhezustand befindet. Da unterschiedliche Nutzer verschiedene Körperhaltungen aufweisen, muss auch dies kalibriert werden.

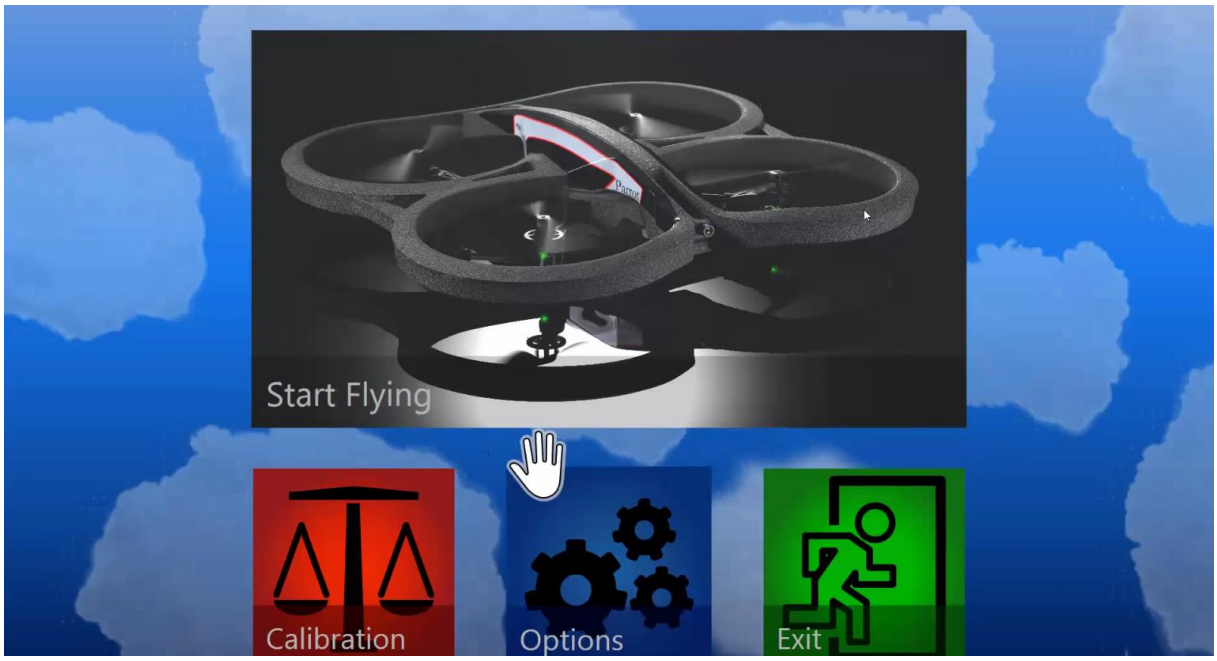


Abbildung 34: Das Steuerungsmenu der Anwendung

Um die Flugansicht nicht mit diesen Informationen zu überladen, wurde in der letzten Version der Anwendung ein Menu konzipiert, über welches die gewünschten Aktionen ausgeführt werden können. Das Menu, welches in *Abbildung 34* zu sehen ist, kann der Nutzer mit Gesten über die Microsoft Kinect steuern. Wie eine Gestensteuerung mit einer Kinect entwickelt werden kann, wird in *Kapitel 4.2.2* genauer beschrieben. Mit dem Klick auf „Start Flying“ kann der Nutzer den „Emergency Mode“ aufheben, sich zur Drohne verbinden und in den Flugmodus mit dem HUD wechseln, um die Drohne zu steuern. Nimmt er die Hände zusammen, wechselt er wieder in das Hauptmenü und die Drohne bleibt auf der aktuellen Stelle stehen. Bei Auswahl der Schaltfläche „Calibration“ kann der Nutzer die oben beschriebene Kalibrierung durchführen. Zusätzlich kann der Nutzer mit Auswahl der Schaltfläche „Options“ die Empfindlichkeit der Drohne einstellen. Zu guter Letzt kann der Nutzer die Anwendung verlassen, wenn er auf die Schaltfläche „Exit“ klickt.

Dementsprechend steht mit der letzten Version der Anwendung einem „I believe I can fly“-Feeling nichts mehr im Weg.

4.2 MICROSOFT KINECT

Für die Aufnahme der Gesten eines Nutzers wird ein visueller Sensor benötigt, der den Nutzer erkennen und die Lage von dessen Gelenken im Raum interpretieren kann. Für dieses Vorhaben eignet sich die Microsoft Kinect. Der Sensor einer Kinect ist in der Lage Tiefen zu erkennen, besitzt eine 1080p Farbkamera, einen Infrarot-Sender und ein Mikrofon-Array, womit sowohl die Sprache, als auch die Bewegung im Raum erkannt werden kann.

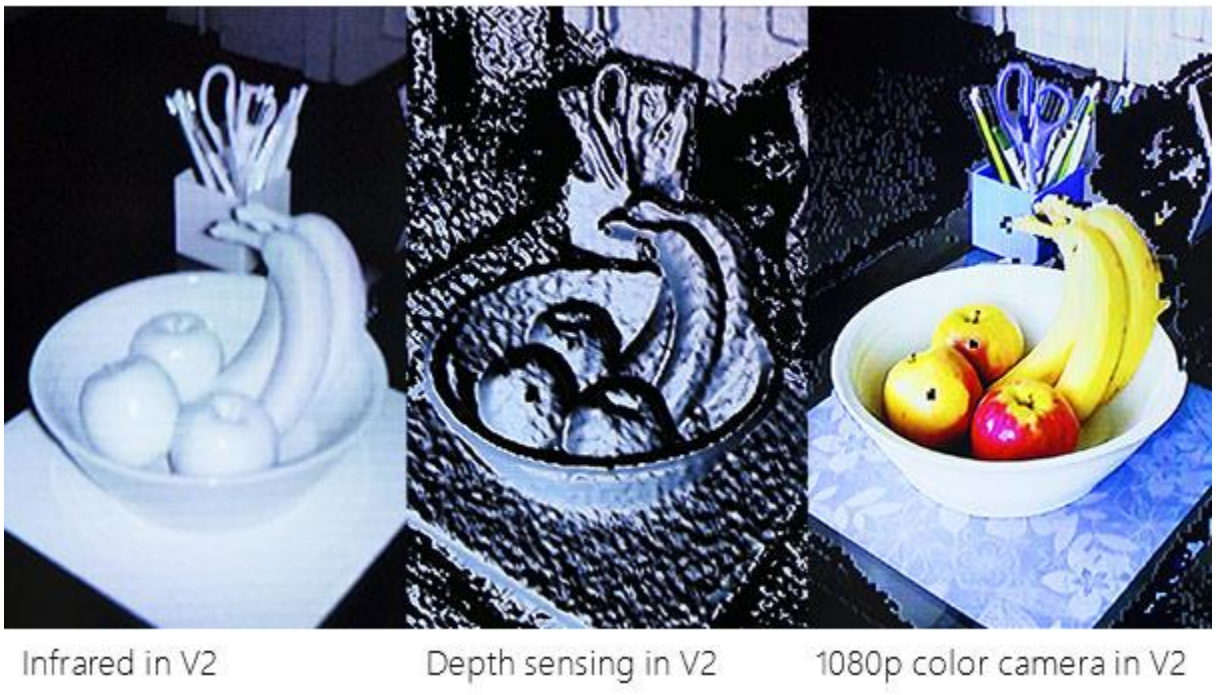


Abbildung 35: Übersicht über die Technologie eines Kinect-Sensors[52]

Abbildung 35 demonstriert wie ein Kinect-Sensor das Spektrum seiner Technologie ausnutzt. Die gebräuchlichste Funktionalität eines Kinect-Sensors ist jedoch die Körpererkennung. Ein Kinect Sensor erlaubt die Lokalisierung von bis zu sechs Personen gleichzeitig. Zwei der sechs Personen können im Raum über 20 Gelenke mit dreidimensionalen Koordinaten verfolgt werden. Diese Technik wird auch „Skeletal Tracking“ genannt. Kinects der Version 2 erlauben sogar „Skeletal Tracking“ mit 25 Gelenken. Neu hinzugekommen sind Gelenke für Fingerspitzen, Daumen und die

[52] (Microsoft Corporation, 2015)

Mitte der Schultern[53]. Dieses Kapitel befasst sich mit den grundlegenden Richtlinien die in Bezug auf User Experience bei der Realisierung des Projektes beachtet werden mussten, um mit Microsoft Kinect zu entwickeln. Im danach folgenden Unterkapitel wird beschrieben, wie eine Kinect mit dem von Microsoft gelieferten SDK angesteuert werden kann.

4.2.1 Entwicklungsrichtlinien

Microsoft stellt mit jeder neuen Version einer Microsoft Kinect ein Handbuch bereit, das Richtlinien zur Entwicklung von User Interfaces enthält, die mit einer Kinect bedient werden können. Da im Rahmen des Projektes eine Microsoft Kinect der Version 1 verwendet wurde, wird in diesem Kapitel das Handbuch für die Version 1 beschrieben.

Kinect-UIs nehmen Eingaben mittels Gesten entgegen. So kann beispielsweise ein Button nicht nur mit einer Maus geklickt, sondern auch mit einer „drückenden“ Handbewegung vor einer Microsoft Kinect betätigt werden. Das Kinect SDK für Windows stellt bekannte UI-Elemente bereit, die nicht nur, wie gewohnt, mit einer Maus bedient werden können. Wie beim Button können all diese UI-Elemente auch über Gesten gesteuert werden.

Abbildung 36: Verschiedene Gesten, die von einer Microsoft Kinect interpretiert werden können[54]

[53] (Microsoft Corporation, 2015)
[54] (Microsoft Corporation, 2013, S. 22)

Die Frau in *Abbildung 36* demonstriert die gängigsten Gesten, über die eine Anwendung mit Kinect-UI-Elementen gesteuert werden kann. Die Geste oben links im Bild zeigt, wie ein Button gedrückt wird. Dafür muss der Nutzer die Hand ausstrecken und die Handfläche entgegen der Kinect bewegen. Eine ebenfalls oft verwendete Geste ist das Greifen. Diese Geste wird hauptsächlich zum Scrollen in Anwendungen verwendet, wie die Frau im unteren linken Teil der Abbildung demonstriert. Manche UI-Elemente erlauben auch ein Zoomen wenn beide Hände, wie im oberen rechten Teil der Abbildung, greifen.

Obwohl die Bedienung von UIs mit Microsoft Kinect sehr bequem sein kann, können technische Barrieren Nutzern Probleme bereiten.

Abbildung 37: Handgesten vor dem Körper[55]

Abbildung 38: Zu schnelle Bewegung[56]

Abbildung 39: Verlassen des Aufnahmebereichs[57]

Die Erkennung und Verfolgung von Gelenken einer Person ist sehr präzise. Dennoch kann der Kinect-Sensor Schwierigkeiten bei Erkennung von Gelenken haben, wenn sie direkt vor den Körper gehalten werden. Um eine zuverlässige Steuerung der Anwendung zu gewährleisten sollte der Nutzer seine Hand seitlich vom Körper bewegen, wie der linke Nutzer in *Abbildung 37* demonstriert. Der rechte Nutzer in dieser Abbildung könnte bei der Bedienung der Anwendung auf Probleme stoßen. Auch bei zu schnellen Bewegungen kann der Kinect-Sensor keine eindeutige Geste interpretieren. Daher sollten Bewegungen, wie der Nutzer in *Abbildung 38* zeigt, vermieden werden. Die größte Barriere zeigt *Abbildung 39*. Sollte sich ein Gelenk des Nutzers außerhalb des Aufnahmebereichs des Kinect-Sensors liegen, ist es der Kinect nicht möglich, diesen Körperteil des Nutzers zu erkennen oder zu

[55] (Microsoft Corporation, 2013, S. 43)
[56] (Microsoft Corporation, 2013, S. 43)
[57] (Microsoft Corporation, 2013, S. 43)

verfolgen. Bei der Anwendung muss daher auf jeden Fall sichergestellt werden, dass sich alle Gelenke, die zur Bedienung der Software benötigt werden, im Aufnahmebereich der Kinect befinden.

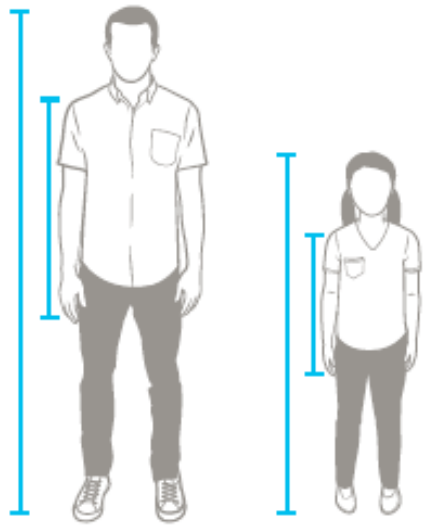


Abbildung 40: Größenunterschiede bei den Nutzern[58]

Bei bestimmten Anwendungsfällen müssen auch unterschiedliche Körpergrößen und Längen der Gliedmaßen von verschiedenen Personen beachtet werden. Wie *Abbildung 40* zeigt, kann es, bezogen auf die Körpergröße, einen erheblichen Unterschied ausmachen, ob eine erwachsene Person oder ein Kind die Anwendung bedient. So muss bei der Drohnen-Anwendung, wie in *Kapitel 4.1* beschrieben, eine Kalibrierung auf Armlänge und Körperhaltung vorgenommen werden, bevor der Nutzer die Drohne fliegt.

Der Nutzer sollte Feedback von der Anwendung über gegebene Aktionsmöglichkeiten und getätigte Aktionen erhalten. Beim Design des Feedbacks muss berücksichtigt werden, was der Nutzer mit der Anwendung erreichen möchte. Im Falle der Drohnenanwendung ist es beispielsweise wichtig, dass der Nutzer weiß, ob er im Aufnahmebereich der Kinect steht oder ob er mit der Hand den richtigen Button im Menü drückt. Dafür benötigt er visuelles Feedback von der Anwendung.

[58] (Microsoft Corporation, 2013, S. 44)

Abbildung 41: Cursor als visuelles Feedback für die Hand[59]

Abbildung 42: Orientierungshilfen für den Nutzer geben[60]

Abhilfe bei der Navigation durch ein Menü schafft der klassische Cursor. *Abbildung 41* zeigt, wie ein Cursor in einem Kinect-UI angezeigt wird. Bewegt der Nutzer seine Hand, folgt der Cursor seinen Bewegungen. Um den Nutzer im Aufnahmebereich der Kinect zu zentrieren, kann ein Rahmen um den Bereich, in dem sich der Nutzer befinden sollte, helfen, sich zu orientieren. *Abbildung 42* zeigt ein möglichen Rahmen und gibt zusätzlich Anweisungen, wie sich der Nutzer platzieren sollte.

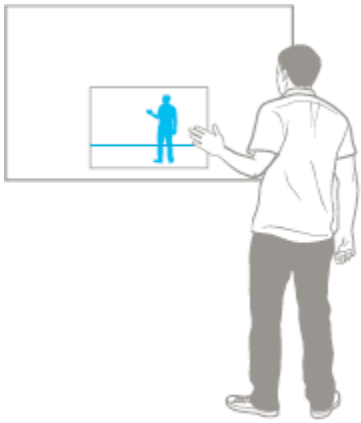


Abbildung 43: Ein "User Viewer" hilft bei der Orientierung des Nutzers[61]

Eine weitere gute Orientierungshilfe ist der sogenannte „User Viewer“. Dabei handelt es sich um die Darstellung des Spielers in Form eines Schattens. Über diesen Schatten, wie er in *Abbildung 43* zu sehen ist, kann der Nutzer schnell erkennen, ob er komplett von der Kinect erfasst wird. Sobald ein

[59] (Microsoft Corporation, 2013, S. 65)
[60] (Microsoft Corporation, 2013, S. 71)
[61] (Microsoft Corporation, 2013, S. 71)

Teil seines Körpers im Schatten abgeschnitten wird, befindet er sich nicht mehr im Aufnahmebereich der Kinect und weiß somit, dass er sich oder die Kinect anders aufstellen muss.

Neben den aufgezeigten Richtlinien behandelt das Handbuch viele weitere Richtlinien, die für spezifische Szenarien angewandt werden können. Darunter fällt zum Beispiel die richtige Handhabung von Spracheingaben mit Microsoft Kinect. Die in diesem Kapitel beschriebenen Richtlinien sind für die Drohnenanwendung hinsichtlich einer guten User Experience unverzichtbar und fanden bei dessen Implementierung große Beachtung.

4.2.2 Programmieren mit Kinect SDK 1.8 für Windows

Das letzte Kapitel handelte von den theoretischen Grundlagen, die von Entwicklern bedacht werden sollten, um gute und interaktive Anwendungen zu implementieren, die mit einer Microsoft Kinect interagieren. Dieses Kapitel geht etwas tiefer auf die praktischen Elemente der Entwicklung für Microsoft Kinect ein. Es wird gezeigt, wie ein Kinect-Sensor initialisiert und wie mit WPF ein klassisches Kinect-UI mit User Viewer, Buttons und Scroll-Containern implementiert werden kann.

Für das Beispielprogramm muss ein C#-WPF-Projekt und die Verweise zu *„Microsoft.Kinect.dll“, „Kinect.Toolkit.dll“, „Kinect.Toolkit.Controls.dll“ und „Kinect.Toolkit.Interaction.dll“* angelegt werden. Außerdem muss entweder die Bibliothek *„KinectInteraction180_32.dll“* oder *„KinectInteraction180_64.dll“* zum Ausführungspfad der Anwendung hinzugefügt werden. Diese Bibliotheken sind im Kinect SDK, sowie im Kinect Developer Toolkit v1.8[62] zu finden.

```
<Window x:Class="GettingStarted.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:k="http://schemas.microsoft.com/kinect/2013"
      Title="Getting Started" Height="500" Width="600">
```

Listing 1: Der Window-Tag mit allen benötigten Namespaces[63]

Um Kinect-UI-Elemente verwenden zu können, muss neben einem Verweise auf die Bibliothek auch ein Namespace ins Extensible Application Markup Language (XAML)-File des Windows eingefügt werden. XAML ist eine auf XML-basierende Beschreibungssprache, die von Microsoft eingeführt wurde, um u.a. Beschreibungselemente für WPF zu definieren. Der Namespace für die Entwicklung

[62] http://www.microsoft.com/en-us/download/details.aspx?id=40276
[63] (Ándras, 2013)

mit Kinect-Elementen lautet `http://schemas.microsoft.com/kinect/2013`. Alle Namespaces der Beispielanwendung lassen sich aus *Listing 1* entnehmen.

```
<Grid>
    <k:KinectSensorChooserUI
         HorizontalAlignment="Center"
         VerticalAlignment="Top"
         Name="sensorChooserUi" />
    <k:KinectRegion Name="kinectRegion">
    </k:KinectRegion>
</Grid>
```

Listing 2: UI-Element zum Anzeigen des Status zur Konnektivität eines Kinect-Sensors[64]

Das erste Kinect-UI-Element dieser Anwendung ist der *SensorChooser*. Dabei handelt es sich um ein Icon, das den Status des Kinect-Sensors anzeigt. Bei jeder Statusänderung ändert der *SensorChooser* gemäß des neuen Status sein Aussehen und zeigt gegebenenfalls auch einen erklärenden Text dazu an. Der Status ändert sich zum Beispiel wenn eine Microsoft Kinect im laufenden Betrieb an den Computer angeschlossen oder abgezogen wird, das Netzteil der Kinect nicht angeschlossen ist oder eine andere Anwendung den Kinect-Sensor blockiert. Insgesamt gibt es zehn verschiedene Status, die der *SensorChooser* annehmen und bei Bedarf auch einen beschreibenden Statustext abrufen kann. Um einen *SensorChooser* zum Fenster hinzuzufügen, muss das in *Listing 2* gezeigte Coding in das Grid-Element vom XAML-File des Windows eingefügt werden. Der Verwendungszweck für *KinectRegion*, wird im weiteren Verlauf des Kapitels erläutert.

[64] (Ándras, 2013)

```
public partial class MainWindow : Window
{
    private KinectSensorChooser sensorChooser;

    public MainWindow()
    {
        InitializeComponent();
        Loaded += OnLoaded;
    }


    private void OnLoaded(object sender,
        RoutedEventArgs routedEventArgs)
    {

        this.sensorChooser = new KinectSensorChooser();
        this.sensorChooser.KinectChanged +=
            SensorChooserOnKinectChanged;

        this.sensorChooserUi.KinectSensorChooser =
            this.sensorChooser;

        this.sensorChooser.Start();
    }

    private void SensorChooserOnKinectChanged(object sender,
        KinectChangedEventArgs args)
    {
        MessageBox.Show(args.NewSensor == null ?
            "No Kinect" :
            args.NewSensor.Status.ToString());
    }
}
```

Listing 3: Anmeldung einer Delegate-Methode für die Initialisierung eines Kinect-Sensors [65]

Die Anwendung kann bei Statusänderungen des Kinect-Sensors Aktionen vornehmen. Dafür muss dem *SensorChooser* lediglich eine Delegate-Methode hinzugefügt werden, die bei Statusänderungen aufgerufen wird. Damit das UI den richtigen Status anzeigt, muss dem Kinect-UI-Element ein Objekt vom Typ *KinectSensorChooser* zugewiesen werden. *Listing 3* verdeutlicht, welche Implementierungsschritte getätigt werden müssen, um einen *SensorChooser* sinngemäß in Betrieb zu nehmen. Mit `this.sensorChooser.Start()` wird der *SensorChooser* aktiviert. In der Delegate-Methode `SensorChooserOnKinectChanged` wird das Verhalten implementiert, das

[65] (Ándras, 2013)

die Anwendung bei einer Statusänderung des Kinect-Sensors aufweisen soll. In diesem Fall wird eine *MessageBox* geöffnet, die den beschreibenden Statustext des *SensorChoosers* beinhaltet.

Abbildung 44: Nach Anschluss eines neuen Kinect-Sensors initialisiert die Anwendung den Sensor[66]

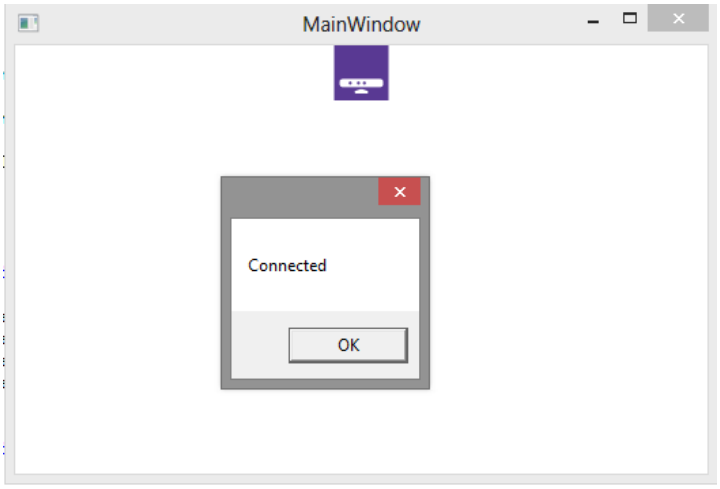


Abbildung 45: Ist der Sensor verfügbar ändert sich das Icon vom KinectSensorChooserUI[67]

Abbildung 44 und *Abbildung 45* zeigen, wie der verwendete *SensorChooser* im User Interface aussieht. Das Aussehen des *SensorChoosers* in *Abbildung 44* ist zu sehen, wenn eine Microsoft Kinect im laufenden Betrieb der Anwendung angeschlossen wird. Der Status des *SensorChoosers* wechselt in den Status „Initialized“. Ist die Kinect betriebsbereit und an einer Steckdose angeschlossen, wechselt der *SensorChooser* in den Status „Connected“. Das Ergebnis im UI wird in *Abbildung 45* dargestellt.

[66] (Ándras, 2013)
[67] (Ándras, 2013)

```
bool error = false;
if (args.OldSensor != null)
{
    try
    {
        args.OldSensor.DepthStream.Range = DepthRange.Default;
        args.OldSensor.SkeletonStream.EnableTrackingInNearRange = false;
        args.OldSensor.DepthStream.Disable();
        args.OldSensor.SkeletonStream.Disable();
    }
    catch (InvalidOperationException)
    {
        // KinectSensor might enter an invalid state while
        // enabling/disabling streams or stream features.
        // E.g.: sensor might be abruptly unplugged.
        error = true;
    }
}

if (args.NewSensor != null)
{
    try
    {
        args.NewSensor.DepthStream.Enable(
             DepthImageFormat.Resolution640x480Fps30);
        args.NewSensor.SkeletonStream.Enable();

        try
        {
            args.NewSensor.DepthStream.Range = DepthRange.Near;
            args.NewSensor.SkeletonStream.EnableTrackingInNearRange = true;
            args.NewSensor.SkeletonStream.TrackingMode =
                 SkeletonTrackingMode.Seated;
        }
        catch (InvalidOperationException)
        {
            // Non Kinect for Windows devices do not support Near mode,
            // so reset back to default mode.
            args.NewSensor.DepthStream.Range = DepthRange.Default;
            args.NewSensor.SkeletonStream.EnableTrackingInNearRange = false;
            error = true;
        }
    }
    catch (InvalidOperationException)
    {
        error = true;
        // KinectSensor might enter an invalid state while
        // enabling/disabling streams or stream features.
        // E.g.: sensor might be abruptly unplugged.
    }
}

if (!error)
    kinectRegion.KinectSensor = args.NewSensor;
```

Listing 4: Erweiterung der SensorChooserOnKinectChanged-Methode[68]

[68] (Ándras, 2013)

Um eine hohe Flexibilität zu erhalten, kann die Delegate-Methode `SensorChooserOnKinectChanged` aus *Listing 3* mit dem Coding aus *Listing 4* befüllt werden. Wird ein Kinect-Sensor im laufenden Betrieb an den Computer angeschlossen, wird der neu angeschlossene Kinect-Sensor initialisiert und im Anschluss dem Attribut *KinectSensor* der *KinectRegion* aus *Listing 2* zugewiesen. Alle Kinect-UI-Elemente die sich innerhalb des Objekts *KinectRegion* befinden, können vom Nutzer mit einem Hand-Cursor ausgewählt werden. Daher muss *KinectRegion* dem Kinect-Sensor zugewiesen werden, von dem sie die Nutzeraktionen entgegennimmt.

Wenn allerdings der Kinect-Sensor im laufenden Betrieb vom Computer abgezogen wird und somit für die Anwendung nicht mehr verfügbar ist, wird er deaktiviert und auf Standardwerte zurückgesetzt.

```
<k:KinectUserViewer
     VerticalAlignment="Top"
     HorizontalAlignment="Center"
     k:KinectRegion.KinectRegion="{Binding ElementName=kinectRegion}"
     Height="100" />
```

Listing 5: Hinzufügen eines User Viewers im XAML [69]

Bevor ein Kinect-UI-Element auf dem Bildschirm angezeigt wird, ist es sinnvoll ein Abbild des Nutzers zu sehen, damit dieser überprüfen kann, ob er im Aufnahmebereich der Kinect steht und die Anwendung problemlos bedienen kann. Dafür dient der sogenannte *UserViewer*. Um diesen im UI anzuzeigen, muss der Programmcode in *Listing 5* zum Main-Grid im XAML-File des Windows hinzugefügt werden. Auch dieser benötigt Informationen des Kinect-Sensors. Deshalb wird der *UserViewer* über ein Binding an das Objekt *kinectRegion* gebunden und erhält darüber die benötigten Informationen.

[69] (Ándras, 2013)

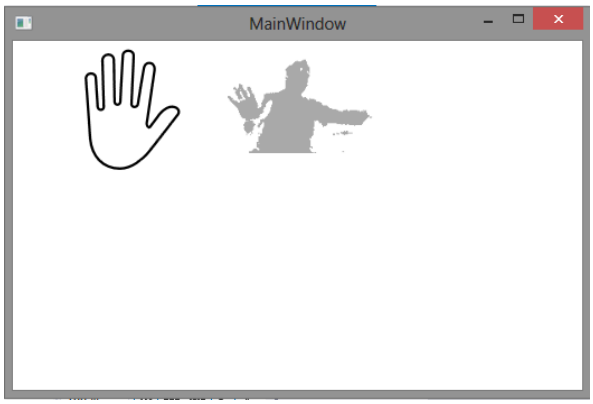


Abbildung 46: Ein Fenster mit User Viewer mit einem Cursor[70]

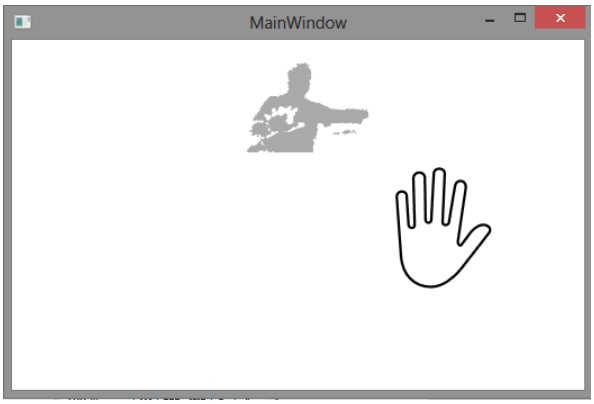


Abbildung 47: User Viewer und Cursor passen sich bei Bewegung des Nutzers an [71]

Wie ein *UserViewer* und eine *KinectRegion* im UI aussieht kann aus *Abbildung 46* und *Abbildung 47* entnommen werden. Der Nutzer, der von der Kinect aufgenommen wird, erscheint als Schatten seiner selbst im *UserViewer*, welcher am oberen Teil des Fensters zu sehen ist. Bewegt der Nutzer seine Hand, folgt ein Cursor in Form einer Hand auf dem Bildschirm seinen Bewegungen. Der Cursor erscheint in den Teilen des Fensters, in denen die *KinectRegion* definiert wurde. In diesem Gebiet können Kinect-UI-Elemente bedient werden, die der Anwendung hinzugefügt wurden.

```
<k:KinectTileButton
     Label="Press me!"
     Click="ButtonOnClick" />

<k:KinectCircleButton
     Label="Circle"
     HorizontalAlignment="Right"
     Height="200"
     Click="ButtonOnClick">Hi
</k:KinectCircleButton>
```

Listing 6: Hinzufügen eines TileButtons und eines CircleButtons im XAML[72]

Nachdem der *UserViewer* und die *KinectRegion* funktionieren, sollten Kinect-UI-Elemente hinzugefügt werden. Dazu muss lediglich zur *KinectRegion* im XAML das Coding in *Listing 6* ergänzt

[70] (Ándras, 2013)
[71] (Ándras, 2013)
[72] (Ándras, 2013)

werden. Der *TileButton* ist eine rechteckige Schaltfläche, die ein Bild und eine Beschreibung beinhalten kann. Ein *CircleButton* hingegen ist eine Schaltfläche in Form eines Kreises, die eine Beschriftung beinhalten kann. Mit der *Click*-Eigenschaft wird dem Button eine Delegate-Methode zugewiesen, die ausgeführt wird, wenn die Schaltfläche betätigt wird.

```
private void ButtonOnClick(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Well done!");
}
```

Listing 7: Implementierung einer Delegate-Methode für die Betätigung eines Buttons[73]

Listing 7 zeigt die Implementierung der Delegate-Methode für die *Click*-Eigenschaft der Buttons. Mit der Ausführung dieser Methode wird eine *MessageBox* geöffnet die „Well done!“ anzeigt.

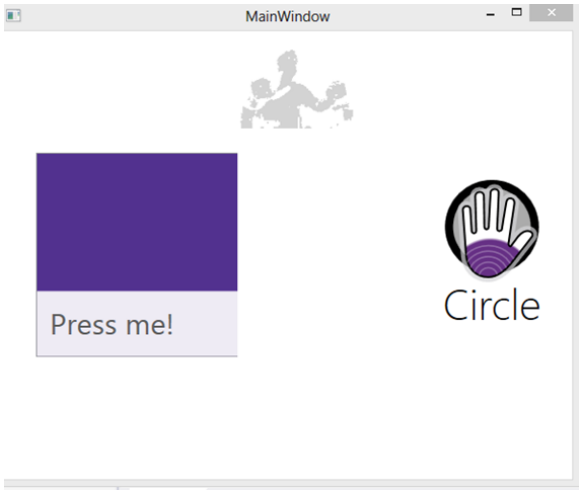


Abbildung 48: Kontinuierliches „Drücken“ der Buttons verhindert versehentliche kurze Betätigung[74]

Abbildung 49: Bei Kontinuierlichem Druck wird die Delegate-Methode ausgeführt[75]

Abbildung 48 und *Abbildung 49* zeigen den Gebrauch der Buttons im UI. Der linke Button ist ein *TileButton* und der rechte ein *CircleButton*. Das Besondere an den Buttons in Kinect-UIs ist die Notwendigkeit eines kontinuierlichen „Drückens“. Betätigt ein Nutzer mit der Kinect einen Button, muss er die Hand für eine gewisse Zeit entgegen der Kinect bewegen, damit die Anwendung den Button als geklickt wertet. Dadurch wird ein versehentliches Drücken des Buttons verhindert. Wird

[73] (Ándras, 2013)
[74] (Ándras, 2013)
[75] (Ándras, 2013)

der Button hingegen mit einem Mauscursor geklickt, reagiert die Anwendung direkt auf den Mausklick.

```
<k:KinectScrollViewer
     VerticalScrollBarVisibility="Disabled"
     HorizontalScrollBarVisibility="Auto"
     VerticalAlignment="Bottom">

          <StackPanel
               Orientation="Horizontal"
               Name="scrollContent" />

</k:KinectScrollViewer>
```

Listing 8: Hinzufügen eines ScrollViewers im XAML[76]

Ein weiteres gängiges Feature, das im Zusammenhang mit Kinect-UIs verwendet wird, ist das Scrollen. Der Nutzer kann scrollen, indem er mit der Hand greift und die geschlossene Hand in Scrollrichtung bewegt. Dieses Feature wird mit dem sogenannten *ScrollViewer* realisiert. Um einen *ScrollViewer* im Beispielprogramm unterzubringen, muss das Coding *Listing 8* in das XAML der *KinectRegion* eingefügt werden. In diesem Fall wird dem *ScrollViewer* ein *StackPanel* hinzugefügt, welches seine Inhalte horizontal anordnet.

```
for (int i = 1; i < 20; i++)
{
   var button = new KinectCircleButton
                    {
                       Content = i,
                       Height = 200
                    };

   int i1 = i;
   button.Click +=
      (o, args) => MessageBox.Show("You clicked button #" + i1);

   scrollContent.Children.Add(button);
}
```

Listing 9: Dem ScrollViewer werden programmatisch CircleButtons hinzugefügt[77]

[76] (Ándras, 2013)
[77] (Ándras, 2013)

Um die Scrollfähigkeit zu demonstrieren werden dem *StackPanel* programmatisch 19 Buttons mit einer eigenen Delegate-Methode, die den Namen des Buttons in einer *MessageBox* ausgibt, hinzugefügt.

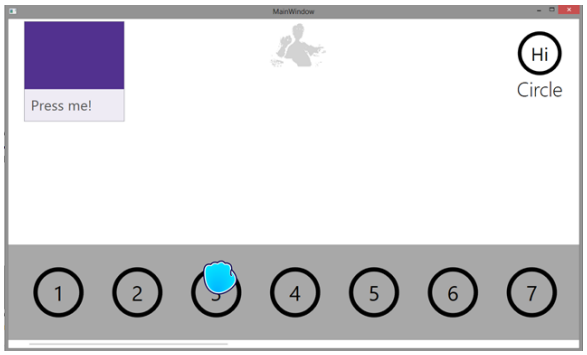


Abbildung 50: Der ScrollViewer kann gegriffen und verschoben werden[78]

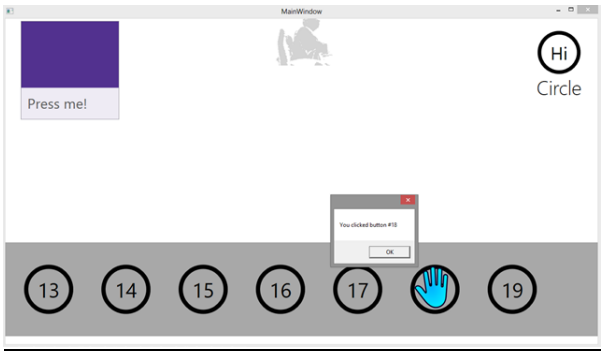


Abbildung 51: Dadurch können weitere Elemente erreicht werden[79]

Der *ScrollViewer* im UI wird in *Abbildung 50* und *Abbildung 51* dargestellt. Möchte der Nutzer, wie in diesem Beispiel den 18. Button drücken muss er zunächst zur Seite scrollen. Dafür greift er mit der Hand die Reihe mit den Buttons und zieht diese mit der greifenden Bewegung von rechts nach links. Hierdurch kommen die restlichen Buttons zum Vorschein, bis der gewünschte Button gedrückt werden kann.

Die beschriebenen Kinect-UI-Elemente reichen aus, um ein großes Spektrum an Funktionalität in einer Anwendung über Interaktion mit Microsoft Kinect abzudecken. Diese Mittel waren die Grundlage, um das Menü und einen Teil des HUDs der Drohnenanwendung zu realisieren.

[78] (Ándras, 2013)
[79] (Ándras, 2013)

4.3 AUGMENTED REALITY-BRILLEN

Unter Augmented Reality wird die Erweiterung der Realitätswahrnehmung mittels technischer Hilfsmittel verstanden. Mit AR ist es z.B. möglich, computergenerierte Informationen in das Blickfeld des Betrachters zu projizieren. Die Projektion erfolgt dabei abhängig vom Kontext, beispielsweise vom momentan betrachteten Objekt in Echtzeit. Dem Betrachter werden im richtigen Moment zusätzliche Informationen zur Verfügung gestellt, zu denen er im Normalfall keinen Zugang hätte. Augmented Reality bedeutet, dass Informationen zusätzlich zu einer realen Szene zur Verfügung gestellt werden. Der Realitätsbezug ist immer gegeben, was den Unterschied zwischen Augmented Reality und Virtual Reality ausmacht, welche oftmals verwechselt werden. Bei Virtual Reality wird dem Betrachter eine computergenerierte Welt gezeigt, die mit der aktuellen Realität nichts zu tun haben muss.[80]

Augmented- und Virtual Reality-Brillen sind eine Möglichkeit, dem Betrachter, Informationen komfortabel in sein Sichtfeld zu projizieren. Sie können in drei Kategorien aufgeteilt werden:

Bei den ***Video-See-Through Head-Mounted Displays*** werden an der Brille zwei kleine Monitore, für jedes Auge einen, montiert, über welche sowohl die Abbildung der realen Szene, als auch die realitätserweiternden Informationen gezeigt werden. Die Abbildungen der realen Szene werden von einer Kamera aufgenommen, beispielsweise von der eines Quadrocopters.

Bei den ***See-Through-Head-Mounted Displays*** ermöglichen gekippte, halb durchlässige Spiegel dem Betrachter den direkten Blick auf die reale Szene. Über den Spiegeln angebrachte kleine Monitore oder Projektoren projizieren die zusätzlichen Informationen in Echtzeit über die Spiegel in das Blickfeld des Betrachters und überlagern das reale Bild.

Die Letzte Kategorie der AR-Brillen bilden die ***Retinal Scanning Brillen***, bei denen die zusätzlichen Informationen über kleine Projektoren direkt auf die Netzhaut des Betrachters projiziert werden. Der Betrachter sieht die reale Szene, welche durch die Projektionen der Projektoren überlagert werden.[81]

[80] (Blümchen, 2002, S. 1)
[81] (Blümchen, 2002, S. 5)

4.3.1 Zeiss Cinemizer

Abbildung 52: Eine Virtual Reality-Brille Zeiss Cinemizer[82]

Abbildung 52 zeigt die Augmented Reality / Virtual Reality Brille *Cinemizer* der Firma Zeiss. Die Zeiss *Cinemizer* Brille gehört zur Kategorie der Video-See-Through Head-Mounted Displays und verfügt über zwei hochauflösende OLED Displays mit einer Auflösung von je 870 x 500 Pixeln, welche zusammen ein Bild ergeben. Die Displays simulieren einen 40 Zoll Monitor bei einer Distanz von zwei Metern. Die *Cinemizer* Brille soll dem Betrachter das Gefühl geben, das Bild auf einer Kinoleinwand zu betrachten. Die Brille wird über HDMI an den Computer oder Fernseher angeschlossen. Sie verfügt über keinerlei Sensorik zum Erfassen von Kopfbewegungen. Dazu stellt Zeiss eine Erweiterung, den Zeiss *Headtracker* zur Verfügung, welcher an der Brille montiert werden kann. Mit ihm können die Kopfbewegungen erfasst und verwendet werden.[83]

[82] (Zeiss, 2015)
[83] (Zeiss, 2015)

4.3.2 Oculus Rift

Abbildung 53: Die Virtual Reality-Brille Oculus Rift[84]

Abbildung 53 zeigt die Virtual Reality-Brille Oculus *Rift* von Oculus. Wie auch die Zeiss *Cinemizer* Brille gehört die Oculus *Rift* zur Kategorie der Video-See-Through Head-Mounted Displays und verfügt über zwei Displays mit einer Auflösung von je 960 x 1080 Pixel. Die Oculus *Rift* hat ein deutlich größeres Sichtfeld und ermöglicht dem Betrachter das Gefühl, direkt in der auf den Displays dargestellten Szene zu sein. Als Grafische Schnittstelle wird HDMI verwendet. Anders als die Zeiss Cinemizer verfügt die Oculus Rift bereits über eingebaute Sensorik. So wurden bei ihr ein Gyroskop, Beschleunigungssensoren und ein Magnetometer verbaut, um die Kopfbewegungen zu erfassen.[85]

[84] Quelle: „Oculus Rift Dev Kit“ von Tmfroehlich - Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Oculus_Rift_Dev_Kit.jpg#/media/File:Oculus_Rift_Dev_Kit.jpg

[85] (Oculus)

5 TECHNISCHE REALISIERUNG DES PROJEKTES

In diesem Kapitel wird ein Blick hinter die Kulissen der bei diesem Projekt entstanden Software geworfen. Es wird aus technischer Sicht die Funktionsweise der Software beschrieben, von der Kommunikation mit der Drohne, über den Aufbau der Benutzeroberfläche, bis hin zur Steuerung der Drohne durch die von der Kinect empfangen Signale. Hierbei werden der Paketaufbau, die Klassenstruktur und das Zusammenspiel der einzelnen Komponenten erläutert. Dieses Projekt wurde unter der Verwendung der Programmiersprache C#, sowie dem .NET Framework 4.0 umgesetzt.

5.1 UMSETZUNG DER DROHNENKOMMUNIKATION

Die Kommunikation mit der Drohne ist einer der zentralen Bestandteile dieses Projektes. Hierfür wurde der offizielle Developerguide der *AR.Drone 2.0* verwendet, welcher von Parrot zur Verfügung gestellt wird. In diesem Guide sind die Funktionalität, sowie die verschiedenen Kommunikationsmöglichkeiten der Drohne beschrieben. Es existiert auch ein offizielles SDK in der Programmiersprache C. Da jedoch, aufgrund der Verwendung der Microsoft Kinect, C# als Programmiersprache Anwendung fand, wurde entschieden die Drohnenkommunikation selbst in C# zu entwickeln. Hierfür wurde ein eigenes Assembly mit dem Namen *„AR.Drone“* angelegt. Auf die gesamte Funktionalität dieses Assemblies lässt sich elegant über die Klasse *„DroneController“* zugreifen. Die einzelnen Funktionen, welche dieses Assembly bereitstellt, werden in den nachfolgenden Unterkapiteln beschrieben.

5.1.1 Senden von Befehlen

Die Drohne erhält alle Befehle über eine UDP-Verbindung auf Port 5556. Bei UDP wird nicht gewährleistet, dass jedes Paket ankommt. Daher müssen die Befehle wiederholt gesendet werden. Nur so kann eine gute User Experience gewährleistet werden, da die Drohne verloren gegangene Befehle nicht ausführen kann.[86]

Bei einem Befehl handelt es sich um eine Zeichenkette, deren Zeichen als 8-bit ASCII Zeichen kodiert sind. Das *„Carriage Return“*-Zeichen dient als Zeilenumbruch. Es hat den Bytewert $0D_h$. Jeder Befehl beginnt mit den drei Zeichen *„AT*“*, also der Bytefolge $41_h, 54_h, 2A_h$. Anschließend kommt der Befehlsname gefolgt von einem Gleichzeichen, einer Sequenznummer und einer Liste von Parametern für diesen Befehl. Die Sequenznummer wird benötigt, damit die Drohne die Reihenfolge der Befehle wiederherstellen und damit alte Befehle ignorieren kann. Die Parameter sind mit einem Komma getrennt und für manche Befehle nicht notwendig. In einem UDP-Packet können auch mehrere Befehle gesendet werden, allerdings muss ein Befehl immer komplett in ein Paket passen und darf eine Länge von 1024 Zeichen nicht überschreiten. Zum Trennen von Befehlen in einem UDP-Paket wird das *„Carriage-Return“* - Zeichen verwendet.[87]

[86] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 19)
[87] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 32)

Tabelle 9: Übersicht der Drohnenbefehle[88]

Befehl	Parameter	Beschreibung
AT* REF	Eingabe	Start / Landung / Notstop
AT*PCMD	flag, roll, pitch, gaz, yaw	Drohnenbewegung
AT*PCMD_MAG	flag, roll, pitch, gaz, yaw, psi, psi-Genauigkeit	Drohnenbewegung (mit absolut Steuerungsunterstützung)
AT*FTRIM	-	Setzt den Referenzwert für die horizontale Lage. Die Drohne darf dafür nicht fliegen
AT*CONFIG	Schlüssel, Wert	Konfiguration der AR.Drone 2.0
AT*CONFIG_IDS	Sitzung, Benutzer, Anwendungsbezeichner	Bezeichner für die AT*CONFIG Befehle
AT*COMWDF	-	Setzt den Kommunikationswatchdog zurück
AT*CALIB	Gerätenummer	Bitte die Drohne das Magnetometer zu kalibrieren. (Drohne muss fliegen)

In *Tabelle 9* ist eine Übersicht über alle verfügbaren Befehle mit ihren jeweiligen Parametern, sowie einer kurzen Beschreibung aufgelistet.

[88] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 34)

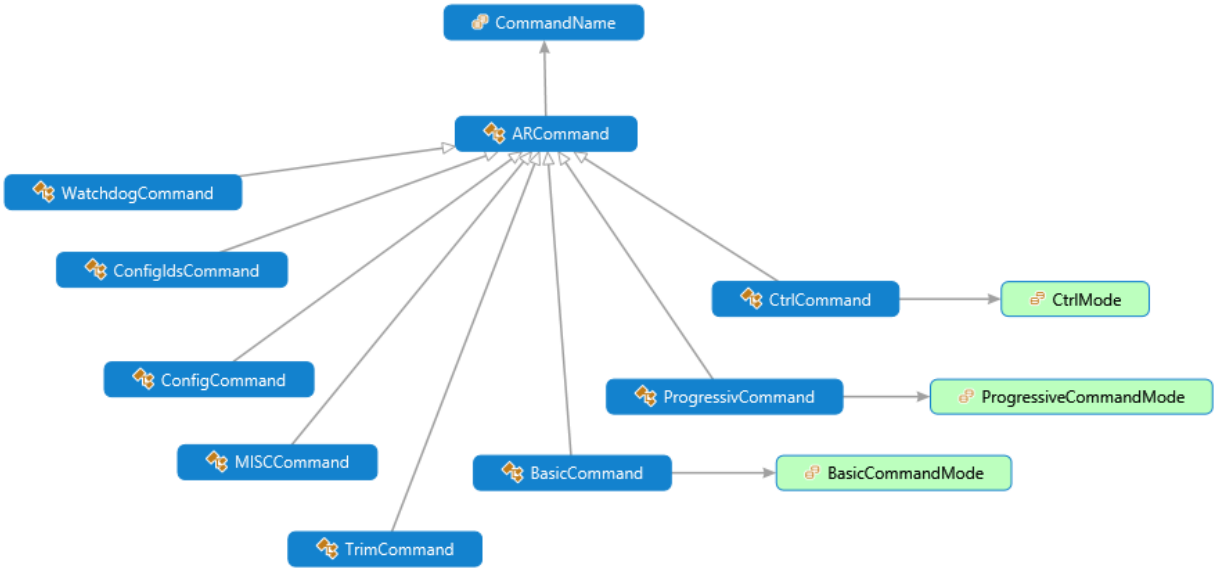


Abbildung 54: Übersicht der Befehlsklassen für die AR.Drone

In unserem Projekt wurde jeder Befehl als eigene Klasse modelliert, welche von der abstrakten Klasse *ARCommand* erbt. All diese Klassen befinden sich im Namespace *AR.Drone.Control.Command*. In *Abbildung 54* ist eine Übersicht der hierfür relevanten Klassen, sowie deren Verbindungen zueinander zu sehen. Bei allen Elementen, außer den grünen, sowie dem mit *CommandName* beschrifteten Element, handelt es sich um Klassen. Alle anderen sind Enums.

Die eben angesprochene abstrakte Oberklasse *ARCommand* erwartet im Konstruktor den Namen des Befehls, sowie die Implementierung einer Methode, welche die Parameter des Befehls als korrekt formatierten String zurückgibt. Die verfügbaren Befehlsnamen sind in dem Enum *CommandName* aufgeführt. Des Weiteren werden Methoden bereitgestellt, um das zu sendende Byte-Array, welches diesen Befehl beschreibt, zu generieren, entweder mit oder ohne Sequenznummer. Die Sequenznummer muss jedoch als Parameter übergeben werden, da diese erst kurz vor dem Senden durch den entsprechenden Thread vergeben werden darf, um keine Duplikate zu erhalten. Für manche Befehle sind zusätzlich im Namespace *AR.Drone.Control.Command.Mode* Enums definiert, welche die möglichen Werte für manche Parameter enthalten. Diese Enums sind in *Abbildung 54* in grün dargestellt und die Klassen, welche diese Enums verwenden, zeigen mit einem Pfeil darauf.

Wie zuvor angesprochen, ist es nötig, dass die Sequenznummer bei 1 beginnt und kontinuierlich ansteigt. Dies erfordert, dass alle Befehle von einer zentralen Stelle aus versendet werden. Diese Stelle ist die Klasse *CommandSender* im Namespace *AR.Drone.Control.* Eine Instanz dieser Klasse ist in der Lage einen eigenen Thread zu starten, welcher Befehle annimmt und diese versendet. Hierfür wird eine Methode bereitgestellt, welche als Parameter ein Objekt vom Typ *ARCommand* erwartet.

Damit der Thread, welcher diese Methode aufruft nicht warten muss bis dieser gesendet wurde, wird dieser in einer blockierenden Queue gespeichert. Es handelt sich um eine First In First Out (FIFO)-Queue. Der Sender-Thread holt einen Befehl aus der Queue, versieht ihn mit der nächsten Sequenznummer und sendet ihn. Ist kein Befehl mehr in der Queue, wartet der Thread solange, bis wieder ein Befehl hinzugefügt worden ist. Dadurch wird gewährleistet, dass nur ein Thread UDP-Pakete sendet und die Sequenznummern korrekt berechnet werden.

5.1.2 Empfangen von Navigationsdaten

Die Navigationsdaten stellen Informationen zum aktuellen Status wie Neigungswinkel, Höhe, Geschwindigkeit, etc. der Parrot *AR.Drone 2.0* bereit. Um an diese Informationen zu gelangen, muss eine UDP-Verbindung über den Port 5554 aufgebaut werden. Alle Entwicklungsartefakte für die Navigationsdaten befinden sich im Namespace *AR.Drone.NavData* oder in einem Sub-Namespace.

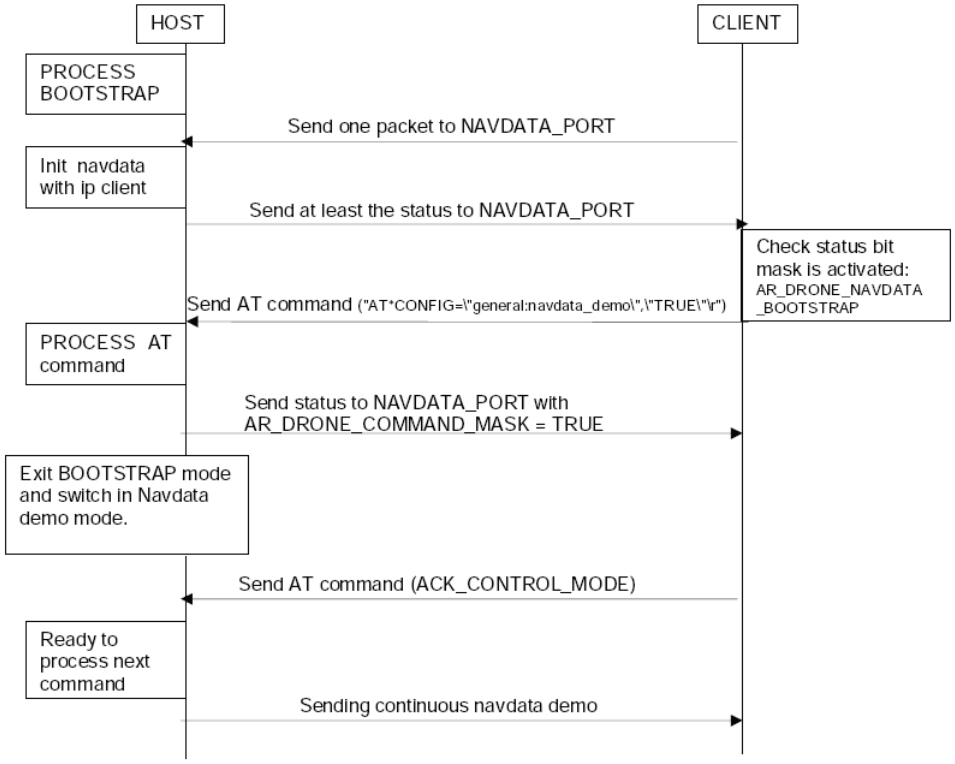


Abbildung 55: Protokoll zum Empfang der Navigationsdaten[89]

[89] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 41)

Abbildung 55 zeigt das Protokoll zum Anfordern der Navigationsdaten von der Drohne. Zunächst muss der Drohne bekannt gemacht werden, dass sich ein Client an den Navigationsdaten-Port verbunden hat. Dies geschieht über das Senden eines Keep-Alive-Signals in Form einer 1. Nach der Antwort der Drohne muss noch festgelegt werden, welche Informationen gesendet werden sollen. Zur Auswahl stehen die „Demo"-Daten, welche nicht alle Informationen enthalten, und die „Nicht-Demo"-Daten, welche alle verfügbaren Navigationsdaten enthalten. Im Projekt waren die „Demo"-Daten nicht ausreichend, weswegen wir die „Nicht-Demo"-Daten verwendet haben. Diese Einstellung ist ein Teil der Konfiguration, weswegen hierfür nicht der Navigationsdaten-Port verwendet wurde, sondern der Command-Port, wie es in *Kapitel 5.1.3 „Konfiguration der Drohne*" beschrieben wird. Wenn über den Navigationsdaten-Port erneut der Status der Drohne empfangen wird, muss ein Aknowledge-Command gesendet werden. Damit weiß die Drohne, dass die Einstellungen, welche geändert wurden, richtig sind. Nachfolgend beginnt die Drohne, regelmäßig Navigationsdaten über den Navigationsdaten-Port zu versenden.

Die Navigationsdaten werden von der Drohne als Bytestream-Pakete versendet, welche von der Anwendung zur Steuerung der Drohne interpretiert werden müssen.[90]

Header	**Drone**	**Sequence**	**Vision**	**Option 1**			…	**Checksum block**		
0x55667788	**state**	**number**	**flag**	id	size	data	…	cks id	size	cks data
32-bit int.	32-bit int.	32-bit int.	32-bit int.	16-bit int.	16-bit int.	… …	… …	16-bit int.	16-bit int.	32-bit int.

Abbildung 56: Aufbau des Navigationsdaten Bytestreams[91]

Abbildung 56 zeigt den Aufbau des Bytestreams, welcher von der Drohne zur Übertragung der Navigationsdaten verwendet wird. Der Bytestream besteht im Groben aus drei Bereichen: dem Header-Bereich, dem Option-Bereich und dem Checksum-Bereich.

Der **Header-Bereich** besteht aus *Header*, *Drone state*, *Sequence Number* und *Vision Flag*. *Header* enthält immer dieselbe Zahl, dient zur Synchronisation zwischen Sender und Empfänger und markiert den Beginn des Navigationsdaten-Bytestreams. *Drone state* besteht aus 32 Flags, welche aktuelle Informationen über den Status enthalten. Welche Flags genau in *Drone state* zu finden sind, kann im Quellcode im Enum *def_ardrone_state_mask_t* nachgesehen werden. *Sequence Number* ist ein

[90] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 39)
[91] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 39)

inkrementierender Zähler, welcher angibt, welcher Bytestream gerade gesendet wird. Da UDP ein verbindungsloses Protokoll ist, muss *Sequence Number* dazu verwendet werden, die Bytestreams in der richtigen Reihenfolge abzuarbeiten. *Vision Flag* enthält Informationen darüber, ob die Drohne momentan ein Ziel, welches mit einem sogenannten Tag versehen ist, anvisiert hat und diesem Ziel folgt. Die Enums zur Interpretation der Flags befinden sich im Namespace *AR.Drone.NavData.Data.States*.

Der **Option-Bereich** enthält die detaillierten Daten und Angaben über den Status der Drohne, wie die Höhe und die Geschwindigkeit. Ein Navigationsdaten-Bytestream kann über mehrere Option-Blöcke verfügen, welche alle mit einer ID beginnen, die angibt, welche Informationen im jeweiligen Option-Block übertragen werden. Eine genaue Liste von möglichen Option-Blöcken findet sich ebenfalls im Quelltext in dem Enum *navdata_tag_t*. Nach der ID folgt die Größe des gesamten Option-Blockes, welche benötigt wird, um zu wissen, wann ein neuer Option-Block oder der Checksum-Bereich anfängt. Nach der Angabe der Größe folgen die eigentlichen Daten. Die Strukturen der einzelnen Option-Blöcke befinden sich im Namespace *AR.Drone.NavData.Data.Options*.

Der **Checksum-Bereich** ist gleich aufgebaut wie ein Option-Block. Er besteht aus einer ID, welche ihn als Checksum-Bereich identifiziert, seiner Größe und der Checksumme an sich. Die Checksumme wird benötigt um die fehlerfreie Übertragung des Bytestreams zu überprüfen. Bei den Navigationsdaten wird auf ein sehr einfaches Verfahren zur Berechnung der Checksumme gesetzt. Der gesamte Bytestream, ohne den Checksum-Bereich wird Byteweise addiert und anschließend mit der übertragenen Checksumme aus dem Checksum-Bereich verglichen. Ist die Summe der Bytes gleich der übertragenen Checksumme, kann der Bytestream weiterverarbeitet werden. Wenn der Checksummentest fehlschlägt, wird der Bytestream nicht weiter verwendet. In dem Fall wird einfach auf den nächsten Bytestream gewartet.

Um den Bytestream richtig zu interpretieren, wurden in Anlehnung an das SDK von Parrot einige Strukturen und Enums in C# angelegt, mit denen es möglich ist, den Bytestream aufzuteilen. Dabei ist es wichtig, dass für jeden Bereich und jede Option eine passende Struktur zur Verfügung steht, bei der zum einen die Byteanzahl und zum anderen die Datentypen stimmen. Anhand der IDs der Option-Blöcke und des Checksum-Blockes, kann ermittelt werden, um welchen Block es sich handelt. Aus den passenden Strukturen kann ein *NavigationPacket*-Objekt befüllt werden, welches zur weiteren Verarbeitung und zur Anzeige der Navigationsdaten auf dem Bildschirm verwendet wird. Die Strukturen zur Interpretation des Bytestreams befinden sich im Namespace *AR.Drone.NavData.Data.*

5.1.3 Konfiguration der Drohne

Das Verhalten der Drohne kann durch einige Parameter beeinflusst werden. Diese Paramater sind die Drohnenkonfigurationswerte. Die Werte lassen sich abrufen, als auch setzen.

Zum Empfangen der Werte muss der AT*CTRL Befehl mit einem Mode-Parameter von 4 (CFG_GET_CONTROL_MODE) gesendet werden. Daraufhin sendet die Drohne auf dem TCP-Port 5559 alle Parameter der Konfiguration. Die Parameter werden als ASCII-Zeichenkette in der Form „ParameterName = ParameterWert“ versendet. [92]

Die Konfiguration ist in verschiedene Sektionen aufgeteilt, so gibt es die *General configuration*-Sektion, welche allgemeine Konfigurationsparameter enthält. Dann gibt es die *Control configuration*-Sektion, welche Eigenschaften enthält, die das Flugverhalten betreffen. Des Weiteren gibt es *Network configuration*, welche Werte über das Netzwerk beinhaltet. Zudem gibt es eine Sektionen für die Konfiguration des Videos, des Höhensensors, der LEDs, des Erkennens von speziellen Tags, des GPS, als auch für die Multikonfiguration.[93]

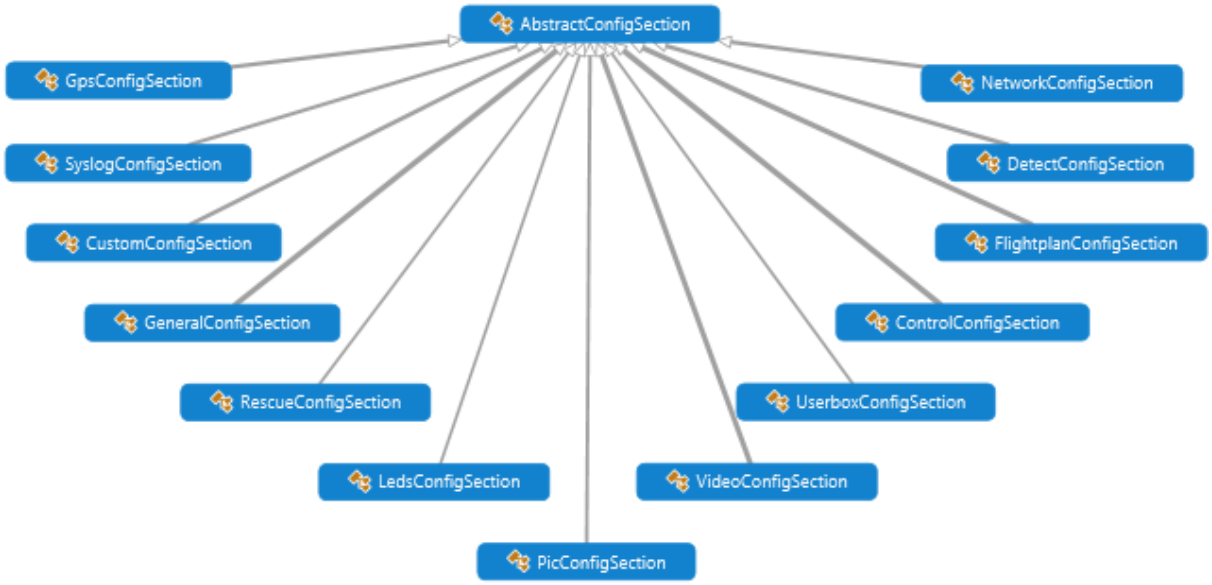


Abbildung 57: Klassenstruktur der Konfigurationssektionen

Jede dieser Sektionen wurde als eigene Klasse modelliert. Alle Klassen erben von der abstrakten Klasse *AbstractConfigSection*. In *Abbildung 57* ist diese Klassenstruktur zu sehen. Im oberen Teil des

[92] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 63)
[93] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 81-94)

Bildes ist die abstrakte Oberklasse und in Kreisform darum befinden sich alle ihre Unterklassen, welche jeweils eine konkrete Sektion beschreiben.

Die Klasse *AbstractConfigSection* hat eine Map als statisches Attribut. Der Schlüsseltyp ist *string* und der Wertetyp ist *Object*. In dieser Map werden alle Konfigurationswerte nach einem Schema abgelegt. Der Sektionsname wird von einem Doppelpunkt und vom Namen des Konfigurationsparameters gefolgt. Der Wert ist vom Typ *Object*, da die Werte der Konfigurationsparameter von verschiedenen Typen sein können, wie z.B. *string* oder *int*.

Zum Hinzufügen eines Konfigurationswertes ist eine Methode vorhanden, welche den Befehl zum Senden eines Konfigurationsbefehls absendet. Dies geschieht mithilfe des „*AT*CONFIG*"-Befehls, mit dem Schlüssel und dem Wert des Konfigurationswertes als Parameter, gefolgt von einem „*AT*CTRL*"-Befehl mit dem Kontrollparameter auf dem Wert 5 (ACK_CONTROL).

Des Weiteren stehen mehrere Methoden zum Auslesen der Werte aus der Map bereit. Die Methoden unterscheiden sich anhand ihres Rückgabetyps. Es gibt eine Methode mit dem Typ s*tring*, eine mit dem Typ *float* und eine mit dem Typ *int*.

Jede der Sektionsklassen hat alle ihre möglichen Parameter als Attribute vom entsprechenden Typ. Des Weiteren werden für jeden Parameter die Getter und Setter so definiert, dass sie die Methoden zum Hinzufügen und Lesen aus der Map der abstrakten Oberklasse nutzen. Damit liegen alle Konfigurationswerte in einer Map, auf welche über verschiedene Klasseninstanzen zugegriffen werden kann.

Um dies zu koordinieren und die Konfigurationsparameter über ein einfaches Interface bereitzustellen, wurde die Klasse *DroneConfiguration* erstellt. Sie hat von jedem Sektionstyp eine Instanz als Attribut. Diese Klasse dient somit als Schnittstelle zur gesamten Konfiguration.

Doch bevor die Konfiguration genutzt werden kann, muss sie von der Drohne geladen werden. Hier kommt die Klasse *ConfigurationReceiver* ins Spiel. Eine Instanz hiervon erzeugt einen neuen Thread, welcher mithilfe einer blockierenden Queue Elemente vom Typ *ConfigurationTask* abarbeitet. Um einen *ConfigurationTask* zu erzeugen werden zwei Methoden bereitgestellt, welche diesen erstellen und direkt zur blockierenden Queue hinzufügen. Eine Methode ruft die gesamte Konfiguration von der Drohne ab und speichert sie anschließend in die zuvor beschriebene Map der abstrakten Konfigurationssektionsklasse. Anschließend sind die Werte über die *DroneConfiguration*-Instanz

verfügbar. Die zweite Methode ruft lediglich die Liste von *Custom Configuration Ids* ab. Diese Methode wird aktuell nicht im Projekt benötigt, ist jedoch vollständigkeitshalber vorhanden.

5.1.4 Empfangen des Videostreams

Zum Empfangen des Videostreams muss eine TCP-Verbindung mit dem Port 5555 zur Drohne aufgebaut werden. Sobald die Verbindung aufgebaut wurde, beginnt die Drohne automatisch mit dem Senden von Parrot Video Encapsulation (PaVE)-Frames. PaVE-Frames sind Datenpakete, die entweder einen ganzen Videoframe enthalten oder einen Teil eines Videoframes. Zusätzlich enthält der PaVE Frame Informationen zu den übertragenen Daten, wie Höhe und Breite in Pixeln, den verwendeten Video Codec und die Größe des PaVE-Frames. Anhand dieser Informationen lässt sich der Payload einerseits zu richtigen Videoframes zusammensetzen, wenn ein Videoframe auf mehrere PaVE-Frames verteilt übertragen wurde, andererseits lässt sich der Videoframe decodieren und anzeigen. Das Decodieren der Videoframes ist nötig, da die Daten komprimiert im H.264-Codec übertragen werden, um die zu übertragenen Daten zu minimieren.[94]

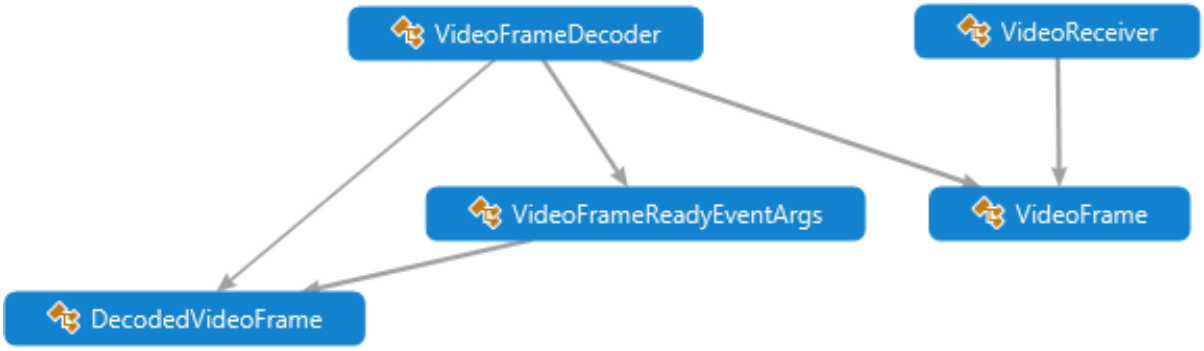


Abbildung 58: Klassendiagramm zum Empfang des Videostreams

Da Videoframes in mehreren PaVE-Frames versendet werden können, ist es sinnvoll das Empfangen und das Dekodieren in jeweils eine eigene Klasse auszulagern. *Abbildung 58* zeigt die Klassen, die zum Empfangen des Videostreams benötigt werden. Instanzen dieser Klassen laufen in einem jeweils eigenen Thread. Die Kommunikation zwischen den Threads wurde mit blockierenden Queues realisiert. Wenn PaVE-Frames von der Instanz der Klasse *VideoReceiver* empfangen werden, werden sie als *VideoFrame*-Objekt zusammengefügt und dann in die *VideoFrameQueue* eingespeist. Die Klasse *VideoFrameDecoder* holt sich die VideoFrames aus der *VideoFrameQueue* und übergibt sie

[94] (Piskorski, Brulez, Eline, & D'Haeyer, 2012, S. 59)

FFmpeg zur decodierung. Sobald ein Frame fertig decodiert ist, wird ein FrameReady-Event gefeuert, um der Benutzeroberfläche mitzuteilen, dass ein neuer Frame zur Anzeige verfügbar ist.

5.2 Aufbau der Benutzeroberfläche

Die Benutzeroberfläche wurde bewusst einfach aufgebaut, damit diese intuitiv vom Nutzer bedient werden kann. Die Hauptbestandteile sind das Hauptmenü, über welches der Nutzer die Anwendung pausieren und Einstellungen vornehmen kann, und das Cockpit der Drohne mit einem Head-Up-Display. In diesem Kapitel wird beschrieben, wie diese beiden Bestandteile im Projekt umgesetzt worden sind.

5.2.1 Fenstermanagement

Das Fenstermanagement ist für eine reibungslose Navigation durch die Anwendung verantwortlich. Mit Ausnahme des Drohnencockpits wurden ausschließlich Kinect-UI-Elemente verwendet, welche in *Kapitel 4.2.2* gezeigt wurden, um eine Interaktionsmöglichkeit mit dem UI und einer Kinect sicherzustellen.

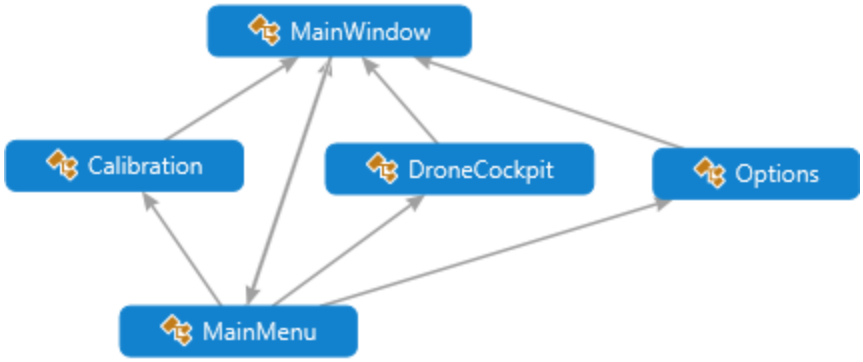


Abbildung 59: Das Klassendiagramm zur Navigation zwischen den Seiten

Das Klassendiagramm aus *Abbildung 59* zeigt den Aufbau des Fenstermanagements im Projekt. *MainWindow* ist der zentrale Bestandteil des Fensters. Diese Klasse beinhaltet ein Objekt vom Typ *Frame.* Ein *Frame* agiert als Container, über welchen zwischen verschiedenen Seiten in WPF – in diesem Projekt sind dies *Calibration, DroneCockpit, Options und MainMenu* – navigiert werden kann. Außerdem beinhaltet die Klasse *MainWindow* eine Instanz des Kinect-Sensors und die Logik zum Austauschen des Sensors im laufenden Betrieb der Anwendung. Um mit der Kinect interagieren zu können, bedienen sich alle Seiten dieser Instanz. Einstiegspunkt der Anwendung ist *MainMenu*. Von *MainMenu* aus kann zu allen anderen Seiten der Anwendung navigiert werden. Von den anderen Seiten kann der Nutzer wiederum zurück zu *MainMenu* navigieren. Die Seite *Calibration* beinhaltet

UI-Elemente zum Starten der Kalibrierung von Armen und Körperhaltung, sowie Elemente, die den Status der Kalibrierung anzeigen. *Options* beinhaltet UI-Elemente, über welche die Sensitivität der Drohne eingestellt werden kann. *DroneCockpit* stellt den Hauptbestandteil der Anwendung dar. Auf dieser Seite kann der Nutzer mit der Drohne interagieren und sie steuern. Sie zeigt das Videobild der Drohne, sowie ein Head-Up-Display mit den Steuerelementen und einem farbigen Schatten des Nutzers an. Das folgende Unterkapitel geht genauer auf die Elemente des HUDs ein.

5.2.2 Head-Up-Display

Zusätzlich zum Videostream der Drohne müssen die Navigationsdaten auf der Benutzeroberfläche dargestellt werden, damit die Informationen zum aktuellen Status der Drohne ersichtlich sind. Auf der Suche nach einer geeigneten Lösung wurde die Implementierung eines Head-Up-Displays gewählt. Ein HUD ist ein Anzeigesystem für Informationen, bei dem der Benutzer seine Blickrichtung nicht ändern muss, da die Informationen in sein Blickfeld projiziert werden.

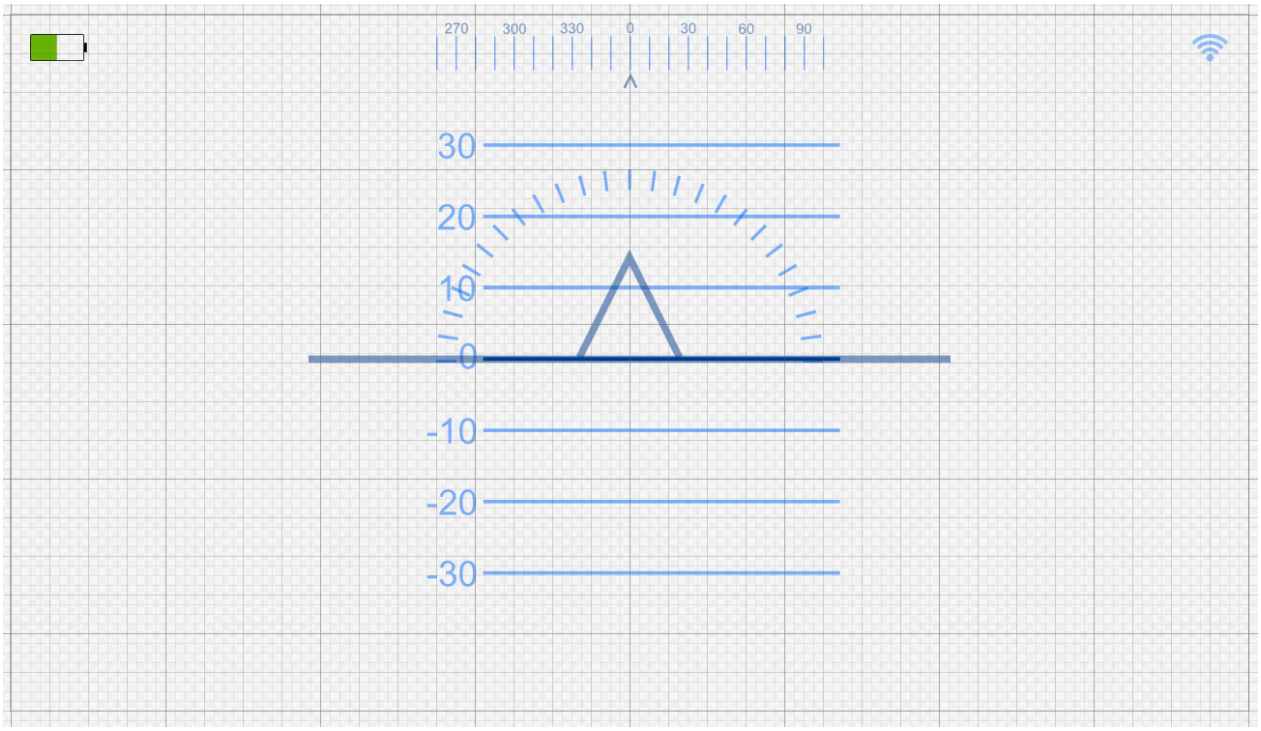


Abbildung 60: HUD des Drohnencockpits in Visual Studio

Abbildung 60 zeigt das entstandene HUD im *Visual Studio Designer*. Im linken oberen Bereich ist eine Anzeige für die Batterieladung zu erkennen, im rechten oberen Bereich wird die Stärke der WLAN-Verbindung zum Client angezeigt und im mittleren oberen Bereich ist die Anzeige für den Yaw-Winkel, also die Drehung um die Z-Achse, abgebildet. Zentral in der Mitte des HUD sind zwei

Anzeigen kombiniert: zum einen stellen der Bogen und der breite waagerechte Balken mit dem Dreieck den Roll-Winkel dar, also die Neigung zur Seite. Zum anderen stellt die Skala mit den nummerierten waagerechten Balken den Pitch-Winkel dar, also die Neigung nach vorne und nach hinten. Während das Programm läuft, wird in der unteren rechten Ecke zusätzlich der Umriss der steuernden Person, welche vom Kinect-Sensor erfasst wird, dargestellt.

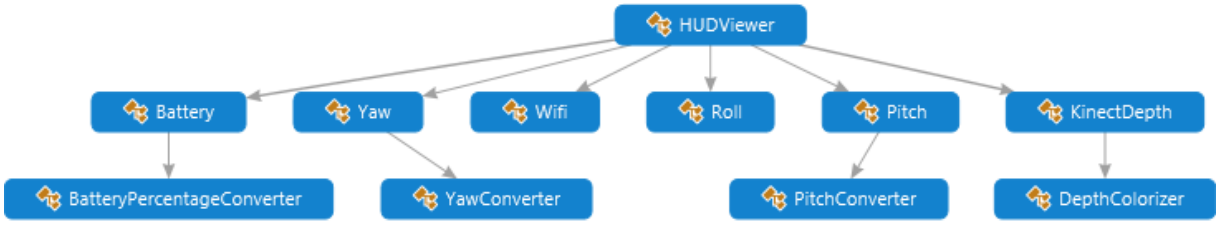


Abbildung 61: Klassendiagramm für das HUD

Abbildung 61 zeigt das Klassendiagramm für das HUD der Drohne. Das HUD ist aus eigenen User-Controls aufgebaut, damit die einzelnen Elemente separat angesteuert werden können. Alle User Controls werden in den *HUDViewer* eingebunden, welcher in der Benutzeroberfläche dargestellt wird. Alle User Controls, außer *KinectDepth,* bekommen ihre Daten direkt aus den Navigationsdaten und werden über PropertyChangeListener aktualisiert. *KinectDepth* bezieht seine Daten direkt von der Kinect und lässt den Umriss der steuernden Person durch den *DepthColorizer* einfärben. Die User Controls *Battery*, *Yaw* und *Pitch* benötigen jeweils einen Converter, um die Informationen aus den Navigationsdaten vor der Darstellung aufzubereiten. Beim *Battery* User Control wird aus einer Prozentzahl ein Balken erstellt, welcher den Rahmen der Batterie, je nach Ladungszustand, füllt. Die Werte für *Yaw* und *Pitch* müssen zur Darstellung in translatorische Werte transformiert werden, da aus einem Drehwinkel eine Verschiebung der jeweiligen Balken auf der Benutzeroberfläche resultieren muss. Das User Control *Roll* benötigt solch einen Converter nicht, da der Winkel der Roll-Navigationsdaten direkt für die Rotation des *Roll* User Controls auf der Benutzeroberfläche verwendet werden kann.

5.3 Steuerung der Drohne

Neben der Kommunikation mit der Drohne ist die effiziente Steuerung der Drohne durch den Nutzer ein weiterer wichtiger Bestandteil der Anwendung. Hierfür muss die Anwendung den Nutzer zunächst erfassen. Eine Microsoft Kinect wurde verwendet, um aus dem Bild des Nutzers ein Skeleton zu erzeugen. Aus diesem wiederum können Steuerungssignale für die Drohne ermittelt werden. Das Teilprojekt, welches die Kinect ansteuert wird in *Kapitel 5.3.1* dokumentiert. Das darauf folgende Unterkapitel handelt vom *Headtracker*, einem Zubehör der AR-Brille Zeiss *Cinemizer*. Mit der Ansteuerung des *Headtrackers* sollte die Bodenkamera der Drohne angesteuert werden. Das letzte Unterkapitel handelt von der Projektstruktur des Fuzzy-Controllers, der zum Einsatz kam, um die Steuerung der Drohne zu optimieren.

5.3.1 Kinect

Im *Kapitel 4.2* wurden generelle Aspekte über Microsoft Kinect beschrieben, wie z.B. die Entwicklungsrichtlinien zur Gestaltung von Anwendungen mit Kinect-UIs und ein Beispielprogramm zur Implementierung dieser. Dieses Kapitel handelt von der praktischen Realisierung des Projektes mit der Kinect und wie das Ansteuerungsmodell aussieht, mit dem der Nutzer über die Kinect die Drohne steuern kann.

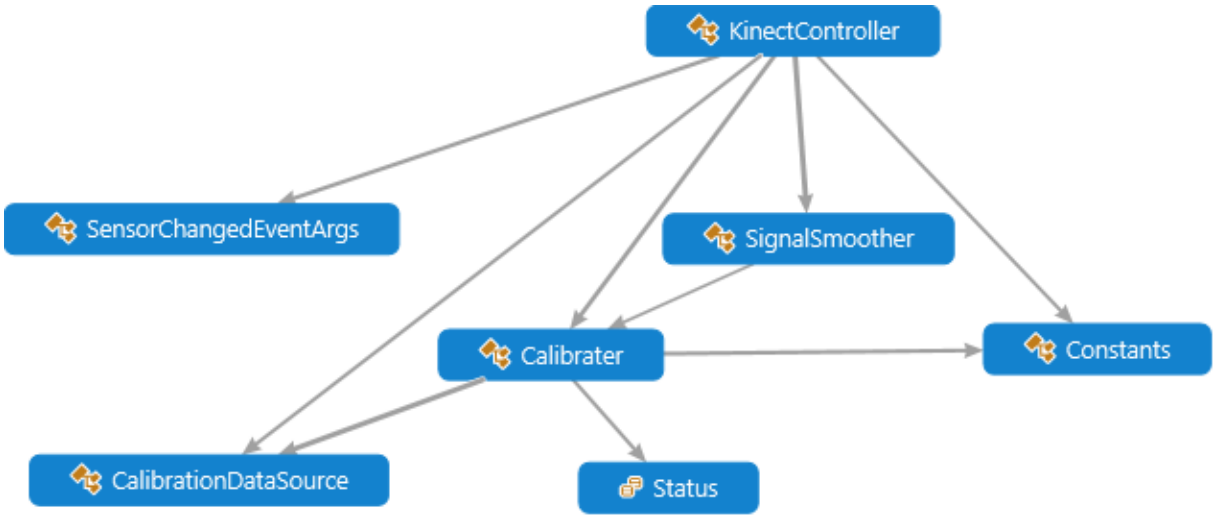


Abbildung 62: Klassendiagramm des Teilprojektes "Kinect"

Die praktische Realisierung des Projektes zeigt das Klassendiagramm in *Abbildung 62*. Der Hauptteil der Logik läuft im *KinectController* ab. In dieser Klasse ist eine Delegate-Methode definiert, die zyklisch aufgerufen wird, wenn neue Information des Skeletons von der Kinect berechnet wurden. Bei den Informationen handelt es sich um die Koordinaten der Gelenke des Nutzers. Aus den Koordinaten werden die Eingangssignale für den Fuzzy-Controller ermittelt.

Abbildung 63: Geste für die Aufwärtsbewegung der Drohne

Abbildung 64: Geste für linke Seitwärtsbewegung

Abbildung 65: Geste für eine rechte Rotation

Abbildung 66: Geste für die Vorwärtsbewegung

Das Signal „up“ berechnet sich aus der Differenz in Y-Richtung zwischen den Schultern und den Händen des Nutzers. Der Nutzer aus *Abbildung 63* lässt daher die Drohne aufsteigen. Der Nutzer aus *Abbildung 64* lässt die Drohne hingegen seitwärts nach links fliegen. Somit wird das Signal „sideward“ aus der Differenz in Y-Richtung zwischen der linken und der rechten Hand berechnet. Das Signal „rotation“ berechnet sich aus der Differenz in Z-Richtung zwischen der rechten und der linken Hand. Daher fliegt der Nutzer aus *Abbildung 65* die Drohne in einer rechten Rotationsbewegung. Der vierte

Nutzer aus *Abbildung 66* steuert die Drohne vorwärts. Das Signal „back“ wird daher durch die Differenz in Z-Richtung zwischen Schultern und Hüfte berechnet.

Die berechneten Eingangssignale werden von Methoden der Klasse *SignalSmoother* zu einer linearen Funktion mit den Grenzen -1 und 1 für die weitere Verarbeitung im Fuzzy-Controller verarbeitet. Anschließend werden die vorverarbeiteten Signale vom *FuzzyController* weiterverarbeitet und die Steuerbefehle an den *DroneController* gesendet. Der *FuzzyController* und *DroneController* stammen aus anderen Teilprojekten, die in den *Kapiteln 5.3.3* und *5.1.1* dokumentiert sind.

Die Klasse *Calibrater* beinhaltet Methoden zum Messen der Armlänge und Körperhaltung des Nutzers. Die gemessenen Daten werden über die Klasse *CalibrationDataSource* an das Frontend kommuniziert. Bei *Status* handelt es sich um ein Enum, das den aktuellen Status der Kalibrierung wiedergibt. *Status* kann die Zustände „Initialized“, „Waiting“ und „Checked“ wiedergeben.

Über die Klasse *SensorChangedEventArgs* wird der neue Sensor für die weitere Verarbeitung des Skeletons übernommen, falls der Kinect-Sensor vom Computer abgezogen und wieder ein Kinect-Sensor angeschlossen wird.

5.3.2 Headtracker

Abbildung 67: Cinemizer Headtracker[95]

Abbildung 67 zeigt den Zeiss *Cinemizer Headtracker*, der eine Erweiterung zur *Cinemizer* Brille von Zeiss darstellt und am Rahmen der Brille befestigt wird. Der *Headtracker* zeichnet mithilfe mehrerer Sensoren die Kopfbewegungen der tragenden Person auf und gibt diese an den Computer weiter. Über ein USB-Kabel mit dem Computer verbunden fungiert das Gerät zunächst wie eine normale Maus. Der Träger der Brille kann mithilfe des *Headtrackers* durch seine Kopfbewegungen die Maus steuern.[96]

Zeiss bietet für den Headtracker ein SDK zum Download an, welches eine Schnittstelle für den Zugriff auf den Headtracker bereitstellt. Mithilfe des SDKs ist es möglich, die Einstellungen des Headtrackers zu ändern und die Bewegungsdaten abzugreifen, um sie für die Anwendungen zu verwenden.

Aus den Rohdaten des Headtrackers wird der Blickwinkel bestimmt. Je nachdem, ob der Blickwinkel nach vorne oder nach unten gerichtet ist, wird die aktive Kamera der Drohne gewechselt. Angenommen der Blickwinkel beträgt 0°, wenn er nach vorne gerichtet ist. Neigt der Benutzer seinen Kopf nach unten, verändert sich der Blickwinkel. Ab einem bestimmten Schwellwert des Blickwinkels,

[95] (Zeiss, 2015)
[96] (Zeiss, 2015)

z.B. ab 45° wird die aktive Kamera auf die Bodenkamera gewechselt. Ändert sich der Blickwinkel wieder nach vorne, wird die aktive Kamera zurück auf die Frontkamera gewechselt. Da das Bewegen der Maus bei der Betrachtung des Kamerabildes der Drohne nicht erwünscht ist, wurde der Headtracker so konfiguriert, dass er die Mausgeschwindigkeit und die Mausbewegung nicht beeinflusst.

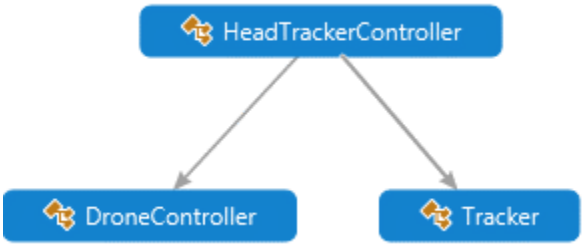


Abbildung 68: Klassendiagramm für den Headtracker

In *Abbildung 68* wird der Klassenzusammenhang für die Kontrolle des *Headtrackers* dargestellt. Die Klasse *Tracker* stammt aus dem SDK von Zeiss und bietet eine Schnittstelle, um auf den *Headtracker* und die Daten des *Headtrackers* zuzugreifen. Eine Instanz des *HeadTrackerController* läuft als eigenständiger Thread und kümmert sich um die Konfiguration des Headtrackers und das Empfangen und Verarbeiten der Rohdaten. Dazu werden Methoden des *Trackers* aufgerufen. *DroneController* stammt zwar nicht aus dem Headtracker Assembly, muss der *HeadTrackerController*-Instanz allerdings bei der Initialisierung übergeben werden. *DroneController* stellt Methoden zur Verfügung, um die Kamera der Drohne zu wechseln. So kann die *HeadTrackerController*-Instanz dafür Sorge tragen, dass die Kamera bei Änderungen des Blickwinkels geändert wird.

Leider stellte sich heraus, dass der Headtracker sehr instabil läuft und die Neigungswinkel trotz mehrfacher Kalibrierung nicht verlässlich sind. Daher kommt es bei der Steuerung mit dem Headtracker öfters zu ungewünschtem Verhalten bezüglich der Umstellung der aktiven Kamera.

5.3.3 Fuzzy-Controller

Der Fuzzy-Controller, wurde mit der *Fuzzy Logic-Toolbox* von *Matlab* entworfen, da diese Möglichkeiten zum Testen der Regeln, sowie der Zugehörigkeitsfunktionen bereitstellt. Des Weiteren lässt sich der entstandene Controller als Fuzzy Inference System (FIS)-Datei speichern. Das Dateiformat ist textbasiert und menschenlesbar.

Für das C# Programm wurde das AForge.NET-Framework[97] genutzt. Um die FIS-Datei im C# Programm nutzen zu können, wurde ein Compiler geschrieben, welcher eine FIS-Datei einliest und zwei C# Klassen generiert. Diese Klassen enthalten den notwendigen Progammcode um den in der FIS-Datei definierten Fuzzy-Controller in C# zu nutzen. Auf diesen Compiler wird jedoch nicht weiter eingegangen.

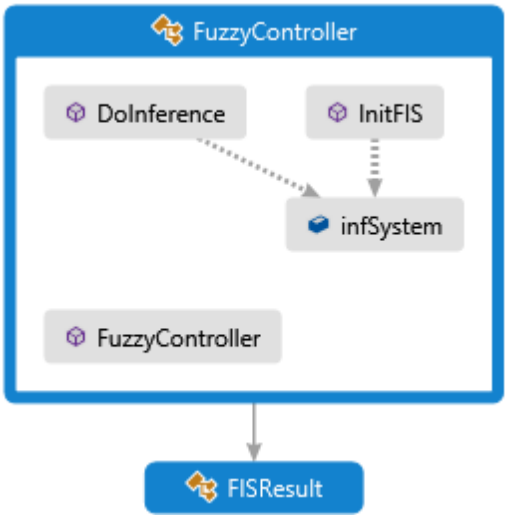


Abbildung 69: Klassenstruktur des Fuzzy-Controllers

In *Abbildung 69* sind die beiden generierten Klassen *FuzzyController* und *FISResult* zu sehen. Auch ist die interne Struktur der Klasse *FuzzyController* dargestellt. Die Klasse stellt zwei Methoden bereit, eine zur Initialisierung und eine zum Berechnen einer Ausgabe für eine Eingabe. Beide Methoden arbeiten mit dem privaten Attribut *infSystem*, welches der eigentliche Fuzzy-Controller darstellt. Der Typ dieses Attributes wird vom AForge.NET Framework bereitgestellt.

Die Methode zur Initialisierung muss nur einmal aufgerufen werden und trägt den Namen *InitFIS*. In dieser Methode werden die Zugehörigkeitsfunktionen definiert und die Regeln angelegt. In der Methode *DoInference* wird die eigentliche Berechnung ausgeführt. Als Eingabeparameter werden die Eingänge des Fuzzy-Controllers erwartet. Die Ausgabe ist vom Typ *FISResult*, die zweite generierte Klasse. Diese dient der Datenhaltung der Ergebnisse, um die Werte für mehrere Ausgänge zurückgeben zu können. Die Klasse enthält für jeden Ausgang des Controllers ein Attribut und stellt diese über Getter-Methoden bereit.

[97] http://www.aforgenet.com/framework/

5.4 Zusammenspiel der Komponenten

Die in den vorherigen Kapiteln dokumentierten Eigenschaften wurden innerhalb des Gesamtprojektes in verschiedene logische Teilprojekte unterteilt. Diese wurden separat voneinander kompiliert. Ein Teilprojekt bildet somit ein logisch gekapseltes Application Programming Interface (API), welches für andere Teilprojekte Funktionalität bereitstellt. Ein Teilprojekt kann auf Funktionen eines anderen Projektes zugreifen, indem Projektabhängigkeiten zwischen den Teilprojekten gepflegt werden. Mit dieser Architektur wird das Software-Design *Seperation of Concerns* umgesetzt. Das bedeutet das Gesamtprojekt wird in logisch voneinander trennbare Stücke zerlegt und vom Rest des Projektes gekapselt. Das Gesamtprojekt wird dadurch überschaubarer und die Teilprojekte beinhalten weniger Dateien.

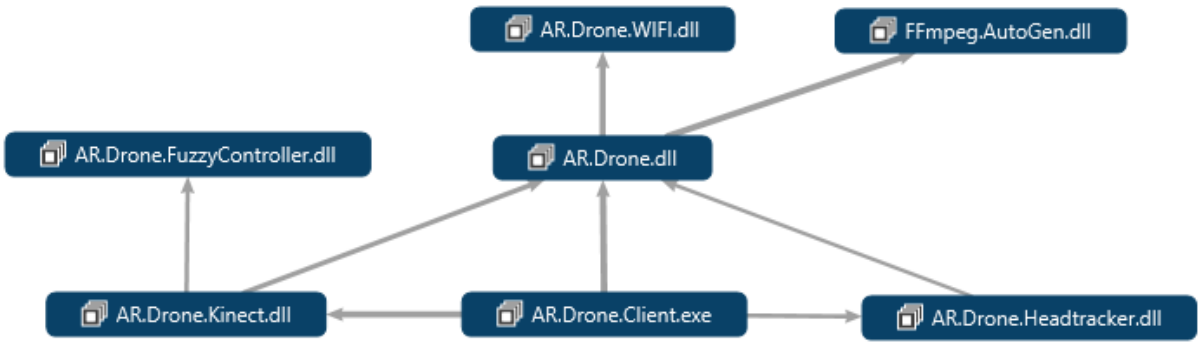


Abbildung 70: Die Abhängigkeiten zwischen den einzelnen Teilprojekten

Abbildung 70 zeigt ein Abhängigkeitsdiagramm für alle Teilprojekte des Gesamtprojektes. Das Herzstück der Anwendung bildet das Projekt *AR.Drone.* In diesem Projekt werden Funktionen zum Verbinden, Steuern und Abfragen von Daten der Drohne bereitgestellt. Funktionen zum Verbinden mit dem WLAN der Drohne und zum Dekodieren des Videostreams wurden wiederum in die Teilprojekte *AR.Drone.WIFI* und *FFmpeg.AutoGen* ausgelagert. *FFmpeg.AutoGen* beinhaltet einen generierten Wrapper für das Videoframework FFmpeg. Dieser Wrapper stammt aus einem Open Source Projekt.[98] Auf die Realisierung dieser Teilprojekte wird in den Unterkapiteln von *Kapitel 5.1* näher eingegangen.

[98] https://github.com/Ruslan-B/FFmpeg.AutoGen

Einen weiteren großen Bestandteil bildet das Teilprojekt *AR.Drone.Kinect.* In diesem Projekt wird die Kommunikation mit einem Kinect-Sensor aufgebaut und mit dem Skeleton des Nutzers Steuerungsbefehle für die Drohne ermittelt. Um die Steuerung der Drohne zu verbessern, werden die Steuerungsbefehle im Fuzzy-Controller, welcher Funktionen des Teilprojektes *AR.Drone.FuzzyController* konsumiert, optimiert. Auf die Realisierung des Teilprojekt *AR.Drone.Kinect* wird im *Kapitel 5.3.1* näher eingegangen. Das Teilprojekt *AR.Drone.FuzzyController* ist in *Kapitel 5.3.3* dokumentiert.

Das Teilprojekt *AR.Drone.Client* beinhaltet sämtliche Logik des Frontends. Hierzu gehören das Fenstermanagement, sowie die Implementierung des Drohnencockpits. Dieses Teilprojekt ist die Repräsentation der Anwendung gegenüber dem Nutzer. Daher ist dieses Teilprojekt ebenfalls das Startprojekt der Anwendung. Die Realisierung dieses Teilprojektes wird in den Unterkapiteln von *Kapitel 5.2* beschrieben.

Weitere Funktionalität bietet die Anwendung mit dem *Headtracker*. Die Logik zur Ansteuerung des *Headtrackers* wurde im Teilprojekt *AR.Drone.Headtracker* implementiert. Ziel der Einbindung des *Headtrackers* war die Ansteuerung der Bodenkamera von der Drohne. Die Realisierung des Teilprojektes ist in *Kapitel 5.3.2* dokumentiert.

6 FAZIT UND AUSBLICK

Das Projekt wurde mit den gesetzten Zielen erreicht. Die Drohne kann mittels Gesten gesteuert werden. Die Steuerung wurde durch den Einsatz eines Fuzzy-Controllers optimiert. Mit Verwendung eines Zeiss *Cinemizers* wird das „I believe I can fly"-Gefühl verbessert.

Leider ist die Einbindung des *Headtrackers* nicht sehr gut gelungen, da er inkonsistente und damit unverlässliche Sensorenwerte liefert, was dazu führte, dass das volle Potential dieser Technik nicht ausgenutzt werden konnte. Die AR-Brille *Oculus Rift* stellt dafür eine gute Alternative dar. Wie in *Kapitel 4.3.2* beschrieben, besitzt die *Oculus Rift* ein eigenes Gyroskop, das programmatisch angesteuert werden kann. Zahlreiche Spielanwendung, wie beispielsweise „Affected – The Cabin"[99] zeigen bereits sehr gute Ergebnisse bei der Verwendung einer *Oculus Rift*.

Neben einer besseren technischen Ausrüstung, kann auch die Software weiterentwickelt werden. Zum aktuellen Zeitpunkt erlaubt die Software eine reine Ansteuerung der Drohne mittels Gesten, die von einer Microsoft Kinect aufgenommen werden. Das „I believe I can fly"-Gefühl könnte beispielsweise durch Minispiele, die in die Software integriert werden, verbessert werden. Der Nutzer würde auf spielerische Art und Weise mit der Drohne fliegen. Eine andere Möglichkeit die Software zu erweitern, wäre die Einbindung von Szenen, die mit dem Kamerabild verschmelzen. So könnte zum Beispiel ein virtueller Fluss am Boden entlang fließen oder holographische Adler um die Drohne herumfliegen. Dadurch würde die Verwendung der Software mit AR-Brillen deutlich imposanter wirken und der AR-Charakter würde noch stärker zur Geltung kommen.

Leider fehlte für die Realisierung dieser Ideen die Zeit. Bibliotheken wie „OpenCV"[100] können bei der Implementierung von visuellen Szenen und Minispielen in Verbindung mit dem Videobild der Drohne unterstützend sein.

[99] http://www.theriftarcade.com/back-this-affected-the-cabin/
[100] http://opencv.org/

V. LITERATURVERZEICHNIS

Ándras. (25. März 2013). *Kinect Interactions with WPF - Part I: Getting Started*. Von dotneteers.net: http://dotneteers.net/blogs/vbandi/archive/2013/03/25/kinect-interactions-with-wpf-part-i-getting-started.aspx abgerufen

Blümchen, K. (2002). *Augmented reality.* Karlsruhe: Universität Karlsruhe.

Börcsök, J. (2000). *Fuzzy Control. Theorie und Industrieeinsatz.* Technik Berlin.

Bristeau, P.-J., Callou, F., Vissière, D., & Petit, N. (2011). *The Navigation and Control Technology Inside the AR.Drone Micro UAV.* Paris; Aubevoye: Centre Automatique et Systèmes, Unité Mathématiques et Systèmes; Parrot S.A.; SYSNAV.

Buchner, S. (2009). *Beschleunigungssensoren.* Koblenz: Universität Koblenz.

Dremel, B., Harl, S., & Kotulla, S. (2008). *Konzeption und Realisierung einer Regelungs- und Steuerungssoftware fur einen Quadcopter.* Friedrich-Alexander Universität Erlangen-Nurnberg, Informatik. Erlangen-Nürnberg: Dremel, Benedikt; Harl, Sebastian; Kotulla, Sebastian.

Fernando, M., Marichal, N., & Jiménez, E. (2014). *Fuzzy Modeling and Control: Theory and Applications.* Paris: Atlantis Press.

Keller, H. (2000). *Maschinelle Intelligenz: Grundlagen, Lernverfahren, Bausteine intelligenter Systeme (Computational Intelligence).* Braunschweig; Wiesbaden: Vieweg.

Keller, H. (2014). Wissenbasierte Systeme - Skript Vorlesung DHBW.

Keller, H. (kein Datum). *Video Fuzzy-Control zur Optimierung der thermischen Abfallbehandlung - ein Erfolgsbericht.*

Mann, H., Schiffelgen, H., & Froriep, R. (2009). *Einführung in die Regelungstechnik: Analoge und digitale Regelung, Fuzzy-Regler, Regler-Realisierung.* München: Hanser.

Microsoft Corporation. (9. September 2013). Human Interface Guidelines v1.8. Redmont, Washington, USA.

Microsoft Corporation. (7. April 2015). *Kinect for Windows*. Von Kinect for Windows features: https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx abgerufen

Oculus. (kein Datum). *Oculus Rift DK2*. Abgerufen am 9. April 2015 von Oculus: https://www.oculus.com/dk2/

Ottens, M., & Jaouad, S. (2009/2010). *Einführung in die Regelungstechnik mit Fuzzy-Logik - Skript der TH Berlin.*

Parrot SA. (kein Datum). *Parrot*. Abgerufen am 17. März 2015 von ardrone2.parrot.com

Piskorski, S., Brulez, N., Eline, P., & D'Haeyer, F. (2012). *AR.Drone Developer Guide.* Paris: Parrot S.A.

Popov, P. D. (2009). *Kinematik und Dynamik.* Berlin: Technische Universität Berlin.

Pounds, P., Mahony, R., & Corke, P. (2006). *Modelling and Control of a Quad-Rotor Robot.* Canberra, Australia; Brisbane, Australia: Australian National University; CSIRO ICT Centre.

Prof. Dr. Stolzenburg, F., Ruh, F., & Schmidsberger, F. (2011). *Multikopter – Mobile Roboter steigen in die Luft.* Wernigerode: Hochschule Harz.

Schulz, G., & Graf, K. (2008). *Regelungstechnik 2: Mehrgrößenregelung, Digitale Regelungstechnik, Fuzzy-Regelung.*

Siegwart, R., & Nourbakhsh, I. (2004). *Introduction to Autonomous Mobile Robots.* Cambridge, Massachusetts: Massachusetts Institute of Technology.

Strand, M. (2014). *Vorlesung Robotik.* Karlsruhe: DHBW Karlsruhe.

Sturm, J. (2014). *Autonomous Navigation for Flying Robots.* München: Technische Universität München.

Zadeh, L. (1965). Fuzzy Sets. Information and Control. Berkeley, California: Department of Electrical Engineering and Electronics Research Laboratory, University of California.

Zeiss. (8. April 2015). *Zeiss Cinemizer OLED Product Information*. Abgerufen am 8. April 2015 von Zeiss: http://www.zeiss.com/cinemizer-oled/en_de/product-information.html