Westfälische Hochschule
University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Entwicklung einer Benutzeroberfläche zur Steuerung einer Drohne mittels manueller Eingabe sowie Bildverarbeitung

Erarbeitet von

Dominik Lauzon
und
Tobias Uhle

Projektarbeit im Modul Robotik

Prof. Jürgen Dunker

Abgabe: xx.xx.2024

Inhaltsverzeichnis

1. Einleitung / Aufgabenstellung

In der heutigen Zeit gewinnen Drohnen in unterschiedlichsten Bereichen zunehmend an Bedeutung. Sie spielen sowohl in der Industrie, im Militär, bei der Überwachung und Inspektion als auch zur privaten Nutzung eine immer größere Rolle. Diese vielfältige Anwendungsmöglichkeit sorgt dafür, dass auch der Leistungsbereich und die Preisspanne der erhältlichen Geräte weit gefächert sind. Jeder Einsatzbereich hat seine eigenen Ansprüche an Funktionen und Handhabbarkeit der Fluggeräte und erfordert unterschiedliche Ausstattungen und teils spezielle Fähigkeiten.

Im Rahmen dieser Projektarbeit soll untersucht werden, welche Anwendungsmöglichkeiten eine speziell zu Lehrzwecken entwickelte Drohne bieten kann. Ziel ist es, eine Softwarelösung für eine Steuerung dieser Drohne über einen Computer zu entwickeln. Es sollen verschiedene Optionen zur Steuerung der Drohne zum Einsatz kommen und die Möglichkeiten und Grenzen der verbauten Technik ergründet werden.

Es wird angestrebt, eine intuitive, leicht bedienbare Benutzeroberfläche zu entwickeln, mittels welcher die Drohne über verschiedene Eingabemethoden manuell steuerbar ist. Sie soll zudem mit Hilfe von Bildverarbeitung verschiedene eigenständige Manöver ausführen und ausgewählte Ziele verfolgen können.

2. Grundlagen

2.1 Programmiergrundlagen

2.1.1 Programmiersprache Python

Die für dieses Projekt wird die Scriptsprache Python in der Version 3.8 verwendet.
Die Sprache wurde im Februar 1991 erstmals veröffentlicht und seitdem bis in die Gegenwart weiterentwickelt. Sie kann auf unterschiedlichsten Betriebssystemen ausgeführt werden und bietet entsprechend eine sehr große Kompatibilität und Verwendbarkeit von in Python geschriebenen Skripten. Zur Ausführung eines Skriptes ist stets ein bereits installierter Interpreter notwendig. Die Skripte werden also nicht kompiliert.
Python kann dabei sowohl objektorientiert als auch prozedural verwendet werden.
Es zeichnet sich vor allem durch mittels Einrückungen definierten Syntax aus. Dabei werden abhängige Befehlsanweisungen, beispielsweise ein „if-else-Statement“ nicht – wie bei den meisten anderen Programmiersprachen üblich – mittels einer Klammer begonnen und beendet.

Klar wird dies an einem typischen „Hello-World“-Beispiel. Hierbei wurde zunächst die Funktion *main* deklariert und danach aufgerufen, um eine dadurch nötige Einrückung darzustellen.

```
def main():
    print(„Hello World“)
main()
```

```
def main():
```

Deklariert die Methode *main* . In der Klammer können Argumente angegeben werden. Der Doppelpunkt zeigt an, dass daraufhin ein entsprechend eine Ebene weiter eingerückter Anweisungsblock folgt. Dieser wird beendet, sobald diese Einrückung endet.

```
print("Hello World")
```

Gibt den als Argument mitgegebenen Text in der Interpreter-Konsole aus.

2.1.2 Grafische Darstellung

Als grafische Oberfläche dient die Python-Bibliothek „Pygame“. Diese Bibliothek ist für das Schreiben von einfachen grafischen Spielen konzipiert und erschien in einer ersten Version im Jahr 2000 von dem einzigen Entwickler „Pete Shinners“. Sie basiert auf der „SDL-Bibliothek“ („Simple DirectMedia Layer“), die für einen grundlegenden Zugang von Audio, Tastatur, Maus, Joystick und grafischer Hardware sorgt. Zwar ist SDL standardmäßig in der Programmiersprache „C“ geschrieben, jedoch wird sie für viele andere Sprachen adaptiert und entsprechend nutzbar gemacht.

Pygame erlaubt eine Erstellung von „Multimedia“-Spielen mit allen nötigen Methoden in der Sprache „Python“ und funktioniert auf nahezu allen Plattformen und Betriebssystemen. Der prinzipielle grafische Aufbau von Pygame besteht aus der Erstellung von Oberflächen, die während der Spiellaufzeit miteinander interagieren und sich dementsprechend verändern können. Durch die Begebenheit der zweidimensionalen Oberflächen eignet sich die Bibliothek naturgemäß nicht für 3D-Anwendungen. In diesem Fall muss eine dedizierte „3D-Engine“ hinzugezogen werden.

Die rudimentären Methoden von Pygame ermöglichen dem Benutzer ein hohes Maß an Freiheit in der Gestaltung seines Spiels. Dem gegenüber steht die Notwendigkeit eines hohen Verständnisses der Bibliothek besonders für das Schreiben fortgeschrittener Spiele.

2.2 Drohnentechnik

Unbemannte Multicopter, umgangssprachlich als „Drohnen“ bezeichnet, sind zumeist kleine Flugobjekte, welche ferngesteuert ähnlich einem Funktionsmodell gesteuert werden können. Je nach Preisstufe und Ausstattung können sie dabei einen erheblich unterschiedlichen Leistungsumfang bieten.

So gibt es Drohnen mit oder ohne Kamera, für den professionellen Bereich sogar mit wechselbaren Objektiven und komplexer Bildstabilisierung.

Die meisten Drohnen besitzen vier Rotoren, mit denen sie ihre komplette Bewegung steuern können. Sollen höhere Lasten transportiert werden oder die Ausfallsicherheit erhöht werden, kann aber auch eine höhere Anzahl an Rotoren zum Einsatz kommen.

Zumeist werden Drohnen für Foto- oder Filmaufnahmen aus der Luft verwendet. Mittlerweile werden sie zudem immer öfter zum Warentransport, in der Landwirtschaft oder auch für das Löschen von Feuern verwendet.
Drohnen werden hauptsächlich elektrisch per Akku betrieben und sind daher in ihrer Flugzeit begrenzt. Wichtige Einflüsse auf diese Zeit sind neben der Akkukapazität das Startgewicht, die vorherrschen Windverhältnisse und die Umgebungstemperatur.
Oftmals besitzen die Fluggeräte ein GPS-Modul, um ihre Position präzise bestimmen zu können und bei Verlust der Funkverbindung zur Fernsteuerung zu einem sicheren Landeplatz zurückzukehren.

Die Bewegung einer Drohne in der Luft geschieht ausschließlich über die Drehzahlsteuerung der Rotoren. Generell drehen sind (bei vier Rotoren) jeweils zwei Motoren in dieselbe Richtung, um das auf das Flugobjekt wirkende Drehmoment aufzuheben.
Die Drehung der Drohne um ihre Hochachse wird durch eine Veränderung der Drehzahl zweier gleichlaufender Rotoren erreicht. Dadurch ändert sich das Gesamtdrehmoment, die Drohne dreht sich auf der Stelle.
Für die Bewegung in eine der horizontalen Richtungen muss die Drohne jeweils etwas gekippt werden, was durch die Anpassung der Drehzahlen aller Rotoren geschieht. Während die in Bewegungsrichtung nach vorne zeigenden Propeller etwas weniger Auftrieb erzeugen, heben die hinteren Rotoren das temporäre Heck der Drohne etwas an und ermöglichen so eine Bewegung in die entsprechende Richtung.
Um sich möglichst präzise in der Luft bewegen zu können, benötigen Drohnen immer Gyroskope. Deren Daten werden ständig vom verbauten Mikrocontroller abgefragt, der dann die Drehzahlen der Rotoren koordiniert.
Einige Modelle sind zusätzlich mit weiteren Sensoren zur Umgebungsbeobachtung ausgestattet. Sie können dann zum Beispiel umliegende Hindernisse erkennen und rechtzeitig Maßnahmen zur Kollisionsvermeidung einleiten.
Seit dem 1. Januar 2021 existieren für die Teilnahme am öffentlichen Luftverkehr mit unbemannten Drohnen detaillierte Regeln in einer europaweit gültigen Verordnung, die unter anderem die Unterscheidung verschiedener Gewichtsklassen regelt und Flugscheine für größere Drohnen verlangt.

2.3 Bildverarbeitung

Die Bildverarbeitung dient dazu, optisch aufgenommene oder digital generierte Bilder zu analysieren und daraus verschiedene Arten von Informationen zu gewinnen. Sie ist zentrale Komponente der in dieser Bachelorarbeit entwickelten Steuerung des Roboters.
Während in Kapitel 3.2.5 näher auf die Möglichkeiten der in diesem Kontext verwendeten Bildverarbeitungs-Software eingegangen wird, soll an dieser Stelle die grundlegende Idee der Bildverarbeitung und der Weg vom optischen Bild zur digitalen Matrix dargestellt werden.

Moderne digitale Kameras besitzen üblicherweise einen CCD[1]-Sensor zur Aufnahme von Fotografien. Diese enthalten rasterförmig angeordnete, auf Licht reagierende Fotodioden, welche je eine Größe zwischen 1,4 µm und 20 µm haben.
Der Sensor erzeugt anhand der an den Fotodioden anliegenden induzierten[2] Spannungen für jede Fotodiode, die sogenannten Pixel, eine elektrische Information über die Lichtintensität an dieser Stelle. Da Fotodioden lediglich die Intensität des Lichts, nicht aber die Farbe messen können, werden sie bei Farbkameras in einem bestimmten Muster[3] mit roten, grünen und blauen Farbfiltern überzogen. Der Bildprozessor innerhalb der Kamera errechnet dann mit Hilfe des Wissens, welches Muster verwendet wurde, das eigentliche farbige Bild.

Das vom Prozessor generierte Bild besitzt eine native[4] maximale Auflösung, die von der Anzahl der Fotodioden auf dem Sensor bestimmt wird. Die Pixel sind dabei in einem rechteckigen Raster angeordnet, welches so die Bildhöhe- und breite definiert. Für jedes Pixel wird der finale Farbwert in der Regel im RGB-Farbraum bestimmt. RGB steht dabei für die drei Grundfarben **R**ot, **G**rün und **B**lau im sogenannten „additiven Farbraum“[5].
Der Wert jeder Farbkomponente umfasst üblicherweise eine Farbtiefe von 8 Bit, bietet also $2^8 = 256$ Abstufungen und gibt die Helligkeit selbiger an. Die Kombination der drei

[1] *charge-coupled device* (dt. ‚ladungsgekoppeltes Bauteil‘)
[2] Eine Fotodiode erzeugt eine dem Lichteinfall entsprechende Spannung
[3] Üblicherweise die sog. Bayer-Matrix
[4] unveränderte
[5] Addiert bzw. mischt die Farbanteile der Grundfarben zu einem finalen Farbwert

Komponenten bestimmt das Erscheinungsbild des jeweiligen Pixels. Ein RGB-Wert von (255, 255, 255) ergibt beispielsweise weiß, (255, 255, 0) resultiert in gelber Farbe.

Mit Hilfe der Bildverarbeitung können diese digitalen Bilder nun beliebig genutzt und manipuliert werden.
Ein grundlegendes Beispiel ist die Veränderung der Bildgröße, bei der mittels verschiedener Methoden die eingehende Pixel-Matrix auf die vorgegebenen Zielmaße umgerechnet wird. Neben der Verkleinerung und Vergrößerung (Abbildung 1) ist auch die Änderung des Seitenverhältnisses möglich.
Weiterführend bietet die Bildverarbeitung Methoden zur Änderung des Farbraumes, etwa um ein Farbbild in ein Monochrombild zu wandeln, sowie einen großen Umfang an sogenannten Filtern, die sich verschiedenste Gesichtspunkte eines digitalen Bildes zunutze machen und dieses somit faktisch unbegrenzt verändern können.

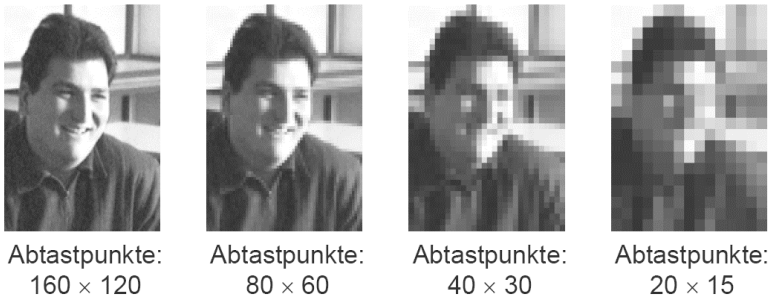

Abbildung 1: Darstellung unterschiedlicher Bildauflösungen

(Quelle dieses Kapitels: *Entwicklung einer Robotersteuerung zur bildverarbeitungsgestützten Objekterkennung auf Basis neuronaler Netze*, Tobias Uhle, 2022, Kapitel 2.3, S. 12ff.)

3. Systemkomponenten

3.1 Drohne

Als Flugobjekt findet die Drohne „Tello“ des Herstellers DJI Verwendung. Hierbei handelt es sich um ein Modell, das hauptsächlich in der Lehre genutzt und verglichen mit professionellen Drohnen entsprechend günstig angeboten wird.
Wie zumeist üblich besitzt die Drohne vier starre Rotoren, welche mittels flexiblen Armen mit der Hauptplatine verbunden sind. Als Zubehör lassen sich Schutzgitter anbringen, die sowohl den Anwender als auch die Drohne vor Schädigungen schützen.
Die Drohne wiegt etwa 80 Gramm, erreicht eine maximale Geschwindigkeit von 28,8 km/h, besitzt eine Reichweite von 100 Metern und kann bis zu 10 Minuten fliegen.
Der Akku ist austauschbar, um einen mehrfachen Einsatz innerhalb kurzer Zeit zu ermöglichen.
In der Front der Drohne ist eine Kamera verbaut, die mittels eines fünf Megapixel messenden Sensors ein Live-Video von maximal 720p Auflösung und 30 Bildern pro Sekunde liefert.
Sie besitzt kein GPS-Modul.

Die Drohne kann entweder mittels einer App des Herstellers über das Handy oder über eine entsprechende Programmbibliothek vom PC aus gesteuert werden. Dafür wird sie jeweils mittels eines eigens erzeugten WLAN-Netzwerks verbunden.

3.2 Software

Das Projekt nutzt einige Klassenbibliotheken von Dritten, welche in den folgenden Unterkapiteln einzeln vorgestellt werden sollen.

3.2.1 PyCharm

PyCharm ist eine Entwicklungsumgebung des Entwicklers JetBrains, die speziell für die Softwareentwicklung mittels Python gedacht ist. Sie bietet Funktionalitäten wie Projekt- und Versionsverwaltung sowie die Verwaltung von in Python installierten Bibliotheken.
In PyCharm können Projekte direkt ausgeführt und debuggt werden, um Fehler lokalisieren und analysieren zu können.
Das Programm ist durch ihre benutzerfreundlichen und leistungsstarken Funktionen sowohl für Anfänger als auch für erfahrene Entwickler geeignet.

Die Versionsverwaltung wird über die weithin bekannte Erweiterung „Git“ realisiert.

3.2.2 djitellopy

Die Klassenbibliothek djitellopy für Python wird von Damià Fuentes Escoté entwickelt und enthält Methoden, um mittels eines Python-Skriptes mit der Tello zu kommunizieren. Sie implementiert alle verfügbaren Tello-Befehle und bietet entsprechend sämtlicher Funktionalitäten, die zur Steuerung und Überwachung der Drohne benötigt werden.
Des Weiteren überwacht die Bibliothek die Verbindung zur Drohne, reagiert auf Programmabstürze, indem sie die Drohne sicher landet und eignet sich sogar dazu, mehrere Drohnen auf einmal zu steuern.

Um die Funktionen nutzen zu können, muss zunächst die Bibliothek mittels

```
from djitellopy import tello
```

importiert werden. Danach ist es zwingend notwendig, ein Objekt der Klasse *tello* zu instanzieren.

```
mytello = tello.tello()
```

Wichtige im Kontext dieser Projektarbeit genutzte Funktionen sind:

- `mytello.connect()`
- `mytello.takeoff()`
- `mytello.land()`
- `mytello.getBattery()`
- `mytello.streamon()`
- `mytello.streamoff()`
- `mytello.getImage()`
- `mytello.send_rc_control()`
- `mytello.move_up()`
- `mytello.move_down()`
- `mytello.rotate_clockwise()`
- `mytello.rotate_counter_clockwise()`
- `mytello.move_forward()`
- `mytello.move_back()`
- `mytello.move_right()`
- `mytello.move_left()`

Der Zweck einer jeden Methode lässt sich aus dem Namen erschließen. Die detaillierte Funktionsweise lässt sich je nach Anwendung im Kapitel „4.1.3 dronecomms“ sowie in der Dokumentation der Bibliothek „djitellopy“ nachlesen.

3.2.3 OpenCV

3.2.3.1 Grundlagen

Das Softwarepaket OpenCV ist eine quelloffene Klassenbibliothek für Python und bietet zahlreiche Algorithmen, um Bilder zur Programmlaufzeit zu manipulieren. Es wurde erstmals 2002 veröffentlicht und seitdem stetig weiterentwickelt.

Beispielsweise lassen sich mittels OpenCV Farbwerte einzelner Pixel auslesen, Bilder wie bei der klassischen Bildbearbeitung editieren (weichzeichnen oder schärfen), umfärben und Farbräume konvertieren. Zudem enthält die Bibliothek Methoden, um etwa Kreise, Quadrate oder Text in Bilder einzufügen.

3.2.4.3 Wichtige Bildoperationen

Im Folgenden werden die in diesem Projekt genutzten Befehle aus OpenCV aufgelistet.

cv2.GaussianBlur	
Übergabeparameter: 3	`src, ksize, sigmaX`
Rückgabeparameter: 1	`dest`
Belegt ein Bild mit einem Gauß'schen Weichzeichnungsfilter. Dabei wird jedes Pixel basierend auf seiner Umgebung der Fläche *size* mit Hilfe der Gauß-Verteilung neu berechnet. Das Bild wird geglättet und mögliche Störpixel[6] oder Bildrauschen[7] reduziert.	
`dest = cv2.GaussianBlur(img, (5, 5), 5)`	

cv2.CvtColor	
Übergabeparameter: 2	`src, converttype`
Rückgabeparameter: 1	`dest`
Konvertiert ein Bild in einen anderen Farbraum. Essenziell sind hierbei vor allem der monochrome und HSV-Farbraum. Letzterer stellt die Pixelwerte als „**H**ue, **S**aturation, **V**alue" dar, also Farbton, Sättigung und Helligkeit. Der *converttype* bestimmt die Art der Konvertierung und folgt dem Schema „Vorher-zu-Nachher".	
`dest = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)`	

cv2.rectangle	
Übergabeparameter: 5	`src, start, end, color, thickness`
Rückgabeparameter: 1	`dest`
Zeichnet ein Rechteck zwischen den Punkten *start* und *end* in das angegebene Bild. Die Liniendicke wird mittels *thickness* angegeben, die Farbe mittels *color*.	
`dest = cv2.rectangle(img, (10, 10), (50, 20), (255,0,0), 2)`	

[6] Fotodioden, die beispielsweise aufgrund einer Fehlfunktion zu helle oder dunkele Farbwerte ausgeben
[7] Körnige, unruhige Struktur, entsteht meist durch variierende Empfindlichkeit einzelner Sensordioden, nicht zu verwechseln mit Rauschen innerhalb einer Datenmenge (vgl. Kapitel 2.4)

cv2.circle	
Übergabeparameter: 5	`src, center, radius, color, thickness`
Rückgabeparameter: 1	`dest`
Zeichnet einen Kreis mit dem Mittelpunkt an den Koordinaten *center* und dem Radius *radius* in das angegebene Bild. Die Liniendicke wird mittels *thickness* angegeben, die Farbe mittels *color*.	
`dest = cv2.circle(img, (20, 50), 10, (255,0,0), 2)`	

cv2.fitEllipse	
Übergabeparameter: 1	`contour`
Rückgabeparameter: 1	`rect`
Zeichnet eine Ellipse um einen Datensatz von 2D-Positionsdaten, wie beispielsweise eine Kontur. Wichtigster Parameter der Ellipse ist in diesem Kontext ihr Winkel, der Aufschluss über die Ausrichtung der längsten Ausdehnung einer Kontur gibt.	
`rect = cv2.fitEllipse(contour)`	

cv2.Ellipse	
Übergabeparameter: 1	`src, rct, color, thickness`
Rückgabeparameter: 1	`dest`
Zeichnet eine Ellipse mit angegebenen Maßen in ein Bild. Die Farbe *color* und Liniendicke *thickness* können bestimmt werden.	
`dest = cv2.ellipse(img,rct,(0,0,255),3)`	

cv2.putText	
Übergabeparameter: 8	`src, text, org, fontFace, fontScale, color, thickness, lineType`
Rückgabeparameter: 1	`dest`
Fügt den angegebenen *text* an der Stelle *org* in das Bild ein. Die Schriftart wird dabei mittels *fontFace* festgelegt, die Schriftgröße über *fontScale*. Farbe (*color*), Liniendicke (*thickness*) und Linientyp (*lineType*) lassen sich ebenfalls definieren.	
`cv2.putText(img, "Hallo", (50,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)`	

cv2.inRange	
Übergabeparameter: 3	`src, lower, upper`
Rückgabeparameter: 1	`mask`
Findet Pixel, deren Werte zwischen *lower* und *upper* liegen und markiert diese nach demselben Schema binär. Die Tupel für die untere und obere Grenze bedienen dabei je nach Aufbau des eingegebenen Bildes (RGB, BGR, HSV) die im jeweiligen Farbraum gültigen Parameter.	
`mask = cv2.inRange(img, (20,20,40), (55,255,230))`	

cv2.findContours	
Übergabeparameter: 3	`mask, mode, method`
Rückgabeparameter: 2	`contours, hierarchy`
Detektiert sämtliche Konturen auf einem Bild. Diese Methode funktioniert am besten mit einem per *inRange* aufgeteilten Bild, in welchem die Konturen durch den Farbwechsel klar abgegrenzt sind.	
`contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)`	

3.2.4 Pygame

3.2.4.1 Beispiel

Um einen Einblick in den Aufbau und die Funktionsweise von Pygame als auch die zugehörigen wichtigsten Methoden zu erhalten, zeigt Abbildung 2 einen typischen Code für ein Pygame-Programm in der simpelsten Erscheinung. Abbildung 3 stellt das grafische Ergebnis dieses Codes dar.

```
# Example file showing a basic pygame "game loop"
import pygame

# pygame setup
pygame.init()
screen = pygame.display.set_mode((1280, 720))
clock = pygame.time.Clock()
running = True

while running:
    # poll for events
    # pygame.QUIT event means the user clicked X to close your window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill the screen with a color to wipe away anything from last frame
    screen.fill("purple")

    # RENDER YOUR GAME HERE

    # flip() the display to put your work on screen
    pygame.display.flip()

    clock.tick(60)  # limits FPS to 60

pygame.quit()
```

Abbildung 2: Beispielcode für eine Pygame-Anwendung (Quelle: https://www.pygame.org/docs/)

Abbildung 3: Ausgabe des in Abbildung 2 gezeigten Codes

Zunächst werden alle Module von Pygame initialisiert mit *pygame.init*. Darauf folgt die Erstellung eines sogenannten „Displays“, welches das sich öffnende Fenster repräsentiert. Die Parameter der Methode *pygame.display.set_mode* sind die Größen des Fensters in Pixel für die Breite und Höhe. Außerdem wird ein „Clock“-Objekt erstellt, das später in diesem Beispiel-Code noch relevant wird. Daraufhin wird eine *running*-Variable erzeugt und mit „True“ initialisiert, um anschließend die „while“-Schleife (auch „Game-Loop“ genannt), in der sich das eigentliche Spiel befindet, zu starten.

Die „for“-Schleife innerhalb der Game-Loop dient dazu sogenannte „Events“ abzufragen und dementsprechend zu handeln. In Pygame werden alle Eingaben des Benutzers „Events“ genannt. Die Eingaben können unter Anderem von der Maus und der Tastatur kommen oder es handelt sich um spezielle Aktionen wie zum Beispiel das Schließen des Fensters. Dabei reiht Pygame eintreffende Events in eine Warteschlange ein. Auf die Events der Warteschlange kann mit Hilfe der Methode *pygame.event.get* zugegriffen werden. Im gleichen Moment des Zugriffs werden alle Events in der Wartschlange entfernt, damit zum einen keine Events unbeabsichtigt doppelt eingelesen werden und zum anderen Platz für neue Events geschaffen werden. Denn die Größe der Warteschlange ist begrenzt, sodass ein wiederholtes Abfragen und Löschen der Warteschlange notwendig ist, um keine Eingaben zu verpassen. In diesem Fall wird nach dem Eventtyp *pygame.QUIT*, der dann erscheint, wenn das Fenster durch das „X“ am oberen rechten Rand geschlossen wird, gefiltert. Die *running*-Variable wird somit auf „False“ gesetzt und Pygame schließt mit der Methode *pygame.quit* in der letzten Zeile.

Um grafische Elemente auf dem Display anzuzeigen, folgen nun wichtige Methodenaufrufe. Das Display wird mit einer Farbe versehen, ausgeführt durch *screen.fill*, die den Hintergrund füllt. Ab diesem Punkt finden nun die spezifischen Grafikelemente des Spiels ihren Platz. Für Grafikelemente gibt es von Pygame bereitgestellte Konstruktoren. So können geometrische Formen wie zum Beispiel Rechtecke oder Kreise, aber auch importierte Bilddateien, erzeugt werden. Jedes dieser Elemente repräsentiert eine Oberfläche, die an einer definierten Position über den Hintergrund gelegt wird. Dabei ist der Ursprung des Koordinatensystems in der oberen linken Ecke des Fensters fest definiert. Es können auch mehrere Oberflächen übereinandergelegt werden, um komplexere Formen zu erzeugen. Anschließend müssen alle Änderungen auf dem Display auch aktualisiert werden, um

überhaupt angezeigt zu werden. Das erledigt die Methode *pygame.display.flip*. Mit dem Aufruf von *clock.tick* kann die Bildwiederholrate begrenzt werden. Diese richtet sich entweder nach der maximal erreichbaren Rate des Monitors oder der Komplexität des Programms selbst.

3.2.4.2 Verwendete Methoden

Um nun ein vollständiges Programm mit grafischen Elementen zu erstellen, bedarf es folgender Methoden, die von Pygame bereitgestellt werden. Dabei wird sich bei der Auflistung nur auf die Methoden und deren Attribute beschränkt, die auch im fertiggestellten Programm Anwendung finden:

pygame.init	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Initialisiert alle Module der Klasse „Pygame“.	

pygame.dispay.set_mode	
Übergabeparameter: 1	`size`
Rückgabeparameter: 1	`Surface`
Erzeugt ein Fenster der entsprechenden Größe. *Size* ist vom Typ (0, 0).	

pygame.display.set_caption	
Übergabeparameter: 1	`title`
Rückgabeparameter: 0	
Definiert den Titel des Fensters.	

pygame.font.SysFont	
Übergabeparameter: 4	`name, size, bold = False, italic = False`
Rückgabeparameter: 1	`Font`
Erzeugt ein Font-Objekt mit beschriebenen Eigenschaften.	

pygame.font.Font.render	
Übergabeparameter: 4	`text, antialias, color, background = None`
Rückgabeparameter: 1	`Surface`
Erstellt aus einem Text ein Surface-Objekt mit beschriebenen Eigenschaften.	

pygame.font.Font.size	
Übergabeparameter: 2	`font, text`
Rückgabeparameter: 2	`width, height`
Gibt die Größe eines Textes mit einer bestimmten Schriftart zurück.	

pygame.image.load	
Übergabeparameter: 1	`filename`
Rückgabeparameter: 1	`Surface`
Lädt eine Bilddatei in Pygame.	

pygame.Surface.fill	
Übergabeparameter: 1	`color`
Rückgabeparameter: 1	`Rect`
Füllt das gesamte Fenster mit einer Hintergrundfarbe.	

pygame.Rect()	
Übergabeparameter: 4	`left, top, width, height`
Rückgabeparameter: 1	`Rect`
Erzeugt ein Rechteck-Objekt mit der beschriebenen Startposition, Breite und Höhe.	

pygame.Surface.get_rect	
Übergabeparameter: 0	
Rückgabeparameter: 2	`width, height`
Gibt die Breite und Höhe eines Rechteck-Objekts zurück.	

pygame.draw.rect	
Übergabeparameter: 3	`surface, color, rect`
Rückgabeparameter: 0	
Zeichnet ein Rechteck-Objekt mit der gewählten Farbe auf eine Oberfläche.	

pygame.Surface.blit	
Übergabeparameter: 2	`source, dest`
Rückgabeparameter: 0	
Zeichnet eine Oberfläche auf eine andere Oberfläche.	

pygame.display.flip	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Aktualisiert alle erzeugten Oberflächen in dem Fenster.	

pygame.event.get	
Übergabeparameter: 0	
Rückgabeparameter: 1	`Eventlist`
Gibt eine Liste mit allen bis dahin erzeugten Events zurück und löscht sie danach.	

pygame.mouse.get_pressed	
Übergabeparameter: 0	
Rückgabeparameter: 3	`button1, button2, button3`
Gibt den Status der drei Maus-Tasten zurück.	

pygame.mouse.get_pos	
Übergabeparameter: 0	
Rückgabeparameter: 2	`x, y`
Gibt die aktuelle Mausposition zurück.	

pygame.Rect.collidepoint	
Übergabeparameter: 2	`x, y`
Rückgabeparameter: 1	`bool`
Gibt an, ob sich ein Punkt innerhalb eines Rechtecks befindet.	

pygame.quit	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Schließt alle zuvor erzeugten Pygame-Module.	

3.2.5 inputs

Die Bibliothek **„**inputs**“** ermöglicht das Erfassen von Eingaben über Peripheriegeräte wie Tastatur, Maus oder Gamecontroller. Sie eignet sich für Anwendungen, die Echtzeit-Benutzereingaben benötigen, wie Spiele, Automatisierungsaufgaben oder Eingabedatenanalyse. Die Bibliothek bietet einfache Methoden, um Tastenanschläge, Mausklicks und Bewegungen sowie Controller-Eingaben abzufangen und zu verarbeiten.

3.2.6 face-library

Mit der Bibliothek „face-library“ lässt sich eine Gesichtserkennung und -wiedererkennung mit wenigen Zeilen Code umsetzen. Die Bibliothek basiert auf OpenCV und nutzt bekannte Algorithmen, um in Echtzeit Gesichter in Bildern oder Videostreams zu erkennen und zu markieren.

3.2.7 pythonping

Das Modul „pythonping“ bietet eine einfache Möglichkeit, Echo-Anfragen (Pings) über Python zu senden. Es dient dazu, die Erreichbarkeit von Netzwerkknoten zu überprüfen und die Latenzzeit (Round-Trip-Time, RTT) zu messen. Das Modul ist einfach gehalten und eignet sich als Diagnosewerkzeug für Netzwerkverbindungen.

4. Implementation / Ergebnisse

In diesem Kapitel sollen nun die Ergebnisse aus Programmierung und Tests erläutert werden.

4.1 Erstellte Python-Skripte

Für dieses Projekt wurden mehrere separate Python-Skripte erstellt, welche im „main“-Skript miteinander verbunden und zur Ausführung gebracht werden.
In den folgenden Kapiteln werden diese Skripte und ihre Funktionen kurz beschrieben:

4.1.1 KeyboardControl

Um die Drohne zunächst per Tastatur zu steuern und damit eine immer verfügbare Möglichkeit zur direkten Kontrolle zu haben, wurde das Skript „KeyboardControl“ entworfen. Dieses verwendet die Bibliothek PyGame, um Tastatureingaben zu erkennen und eine der in Kapitel „4.4.1 Steuerungsschema“ festgelegten Richtline entsprechende Rückgabe an die Hauptroutine zu liefern.

4.1.2 XBoxControl

Da die Drohne auch gesteuert werden soll, ohne direkt am entsprechenden PC zu sitzen, wurde mit dem Skript „XBoxControl“ eine Eingabemethode mittels eines kabellosen Gamecontrollers realisiert. Die darin enthaltene Klasse bedient sich der frei verfügbaren Klassenbibliothek „inputs“ und kann die Eingabe eines entsprechend geeigneten Controllers (siehe beispielhaft Abbildung 4), der das für XBox-Controller übliche Tastenschema besitzt, in das in Kapitel „4.4.1 Steuerungsschema“ festgelegte Schema übersetzen.

Abbildung 4: Logitech F710 als Beispiel eines XBox-kompatiblen Controllers.

4.1.3 dronecomms

Das Skript „dronecomms“ stellt eine Klasse zur Verfügung, welche vollständig für die Kommunikation mit der Drohne verantwortlich ist.
Sie nutzt als Basis die bereits vorgestellte Klasse „djitellopy“, sichert diese jedoch nochmal explizit gegen Fehler ab. Sie implementiert Variablen, die selbstständig die Verbindung zur Drohne, den Flugstatus (Unterscheidung zwischen „in der Luft“ und „am Boden“), sowie den Bewegungsstatus speichern. Damit können bestimmte Methoden der Hauptbibliothek „djitellopy“ nur ausgeführt werden, wenn sich die Drohne in einem entsprechenden Stadium befindet. Dies verhindert, dass das Programm bei einem womöglich ungültigen Befehl einfach abstürzt und die Drohne unkontrollierbar wird.
Im Folgenden werden die implementierten Methoden gelistet und kurz erläutert:

dronecomms::__init__	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Initialisiert ein Objekt der Klasse „dronecomms“.	

dronecomms::connect	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Versucht, eine Verbindung zur Drohne aufzubauen. Ist dies erfolgreich geschehen, gilt die Drohne als verbunden und es können weiterführende Befehle gesendet werden.	

dronecomms::getBattery	
Übergabeparameter: 0	
Rückgabeparameter: 1	`int batteryvalue`
Gibt den Akkustand der Drohne in Prozent zurück.	

dronecomms::streamon	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Aktiviert den Videostream der Drohne.	

dronecomms::streamoff	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Deaktiviert den Videostream der Drohne	

dronecomms::takeoff	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Lässt die Drohne eigenständig abheben. Die Drohne überwacht dabei selbstständig ihre Flughöhe.	

dronecomms::land	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Lässt die Drohne eigenständig landen.	

dronecomms::getspeed	
Übergabeparameter: 1	`string dir`
Rückgabeparameter: 1	`int speed`
Gibt den Wert für die Geschwindigkeit der Drohne in die per Variable *dir* angefragte Richtung (x, y oder z) zurück.	

dronecomms::getacceleration	
Übergabeparameter: 1	`string dir`
Rückgabeparameter: 1	`int acceleration`
Gibt den Wert für die Beschleunigung der Drohne in die per Variable *dir* angefragte Richtung (x, y oder z) zurück.	

dronecomms::getHeight	
Übergabeparameter: 0	
Rückgabeparameter: 1	`height`
Gibt den Wert der Flughöhe der Drohne zurück.	

dronecomms::gettemperature	
Übergabeparameter: 0	
Rückgabeparameter: 1	`int temp`
Gibt die Temperatur der Hauptplatine der Drohne zurück	

dronecomms::getImage	
Übergabeparameter: 0	
Rückgabeparameter: 1	`image img`
Gibt das aktuelle Bild der Drohnenkamera zurück.	

dronecomms::flip	
Übergabeparameter: 0	
Rückgabeparameter: 0	
Lässt die Drohne selbstständig einen Vorwärts-Salto ausführen.	

dronecomms::sendcontrols	
Übergabeparameter: 2	`string mode, string movementtable`
Rückgabeparameter: 0	
Verarbeitet die in *movementtable* mitgelieferen Steuerungsdaten, um sie als Bewegungsbefehle an die Drohne zu senden. Der Modus bestimmt dabei, wie die Gestalt der gewünschten Bewegung ist. *mode == 1* besagt, dass die mitgegebenen Daten eine permanente Richtung mit der entsprechend angegebenen Geschwindigkeit vorgeben. *mode == 2* hingegen besagt, dass sich die Drohne in die mitgegebene Strecke und Richtung in Zentimetern bewegen soll.	

4.1.4 ImageProcessing

Die Bibliothek „ImageProcessing“ ist für alle Bildbearbeitungen und damit zusammenhängende Berechnungen zuständig.
Ihre Funktionen werden allesamt über die Methode *processImage* aufgerufen. Die Mitgabeparameter sind dabei das aktuelle Bild der Drohnenkamera und der Status, welcher die auszuführenden Bildmanipulationen bestimmt.

Je nach ausgewähltem Modus kann diese Methode nun einer der folgenden Berechnungen durchführen:

- Keine Veränderung (Modus 0)
- Ballverfolgung im absoluten Modus (Modus 1)
- Ballverfolgung im relativen Modus (Modus 2)
- Gesichtserkennung inkl. Drohnendrehung (Modus 3)

Die genauen Funktionsweisen der jeweiligen Modi werden in Kapitel 4.5 Bildgesteuerter Betrieb“ näher beschrieben.
Letztendlich führen die Operationen der unterschiedlichen Modi dazu, dass als Rückgabewert das jeweils bearbeitete Bild und eine Tabelle, entsprechend dem Steuerungsschema, zurückgegeben wird. Die Bearbeitungen des Bildes beschränken sich dabei auf die Markierung respektive Umrahmung des verfolgten Objekts und auf eingefügte Texte, die beispielsweise die Koordinaten des Schwerpunkts der erkannten Kontur oder die Drohnenbefehle in die jeweiligen Raumrichtungen.
Das zurückgegebene Bild wird von PyGame zur Anzeige für den Nutzer weiterverwendet, weshalb es bereits innerhalb des ImageProcessing in ein für PyGame kompatibles Format konvertiert wird.
Im Code lässt sich des Weiteren ein Modus 5 finden, welcher allerdings nicht über die Benutzeroberfläche aktivierbar ist. Dieser Modus diente bei der Entwicklung dazu, die minimalen und maximalen Farbwerte des Balls zu ermitteln. Dabei werden die aktuellen des zentralen Bildpixels mit dem bisherigen Minimum sowie Maximum verglichen und entsprechend diese Grenzwerte angepasst. Es lassen sich durch eine händische Bewegung des Balls die passenden farblichen Grenzwerte für die anderen Modi ermitteln.

4.1.5 dashboard

In der Bibliothek „Dashboard“ kommt Pygame zum Einsatz. Das Dashboard öffnet ein Pygame-Fenster, welches für die Darstellung von Bildmaterial der Drohnenkamera als auch von weiteren Informationsdaten der Drohnensensoren zuständig ist. Zusätzlich soll das Fenster interaktive grafische Knöpfe besitzen, die zur Steuerung der Drohne dienen. Zusammenfassend funktioniert das Pygame-Fenster als eine Art „Userinterface“ – also als eine Schnittstelle zwischen Mensch und Drohne – für die Überwachung und Kontrolle.

4.1.6 main

Die übliche „main“ verbindet alle Module miteinander. Damit die Software richtig funktioniert, muss die Datei „main.py“ aufgerufen werden. Alle anderen Module müssen im selben Verzeichnis wie diese abgelegt sein.

Nach dem Start wird zunächst mittels der Bibliothek „pythonping“ ein sogenannter Ping, also eine Verfügbarkeitsanfrage an die der Drohne zugeordnete IP-Adresse 192.168.10.1, ausgeführt. Nur, wenn die Drohne darüber erreichbar ist, werden die eigentlichen Funktionalitäten der Skripte aktiviert. Dadurch wird verhindert, dass das Programm bei nicht verbundener Drohne abstürzt, da es beispielsweise keine Parameter der Drohne abfragen kann.
Die korrekte Abfolge für eine erfolgreiche Drohnenverbindung wird dabei in Kapitel „Das Dashboard-Skript basiert auf einer gleichnamigen Klasse, in der alle Parameter und Methoden zur Erstellung der Benutzeroberfläche untergebracht sind. Für bestimmte komplexere Grafikelemente mit erweiterten Funktionsumfang bietet es sich in Pygame an eine dedizierte Klasse zu bilden. Dies ist der Fall für die interaktiven Knöpfe in dem Userinterface, sodass eine zweite Klasse für die „Buttons“ in demselben Skript vorzufinden ist. Der Vorteil einer Strukturierung als Klasse ist die Übersichtlichkeit des Programms in der Main und das einfache Ansprechen und Manipulieren von Objektinstanzen.

Zur Erläuterung des geschriebenen Codes werden die erstellten Konstanten und Methoden als Funktionseinheit angesehen und deren Funktionsweise zusammengefasst erörtert.

4.2.2 Konstanten

Um alle Grafikobjekte bezüglich Position, Größe und Farbe zu definieren, bedarf es zu Beginn vieler statischer Konstanten. Dazu gehört auch die Bestimmung der Fenstergröße und die allgemeine Textgröße. Außerdem wird eine vertikale und horizontale Distanz erzeugt. Beider dieser Distanzen dienen der Positionierung von Anzeigen und Knöpfen, die sich innerhalb eines weiteren Rahmens befinden (siehe Status, Modus und Steuerung).

4.2.3 __init__()

Die Initialisierungsmethode einer Klasse wird für Code benutzt, der lediglich am Anfang der Initialisierung eines Objekts ein einziges Mal ausgeführt werden muss. So kümmert sich die „__init__“ um die Initialisierung von Pygame selbst und um die Generierung eines sich öffnenden Fensters mit Titel. Zusätzlich wird eine Schriftart aus dem internen Speicher des Computers geladen, mit der später alle Texte ausgestattet werden. Damit die Anzeige des Bildes mit der Tastenbelegung ermöglicht werden kann, wird die dementsprechende PNG-Datei in das Programm geladen.
Es folgen nun die vier verschiedenen Umrandungen in Form von erzeugten Rechtecken für die Bereiche des Status, des Modus und der Steuerung.
Im Anschluss werden die Buttons über die eigenständige Button-Klasse und dessen entwickelten Konstruktor initialisiert. Zu den interaktiven Knöpfen zählen sowohl die Steuerung der vier verschiedenen Modi als auch die beiden Aktionen „Starten/Landen“ und „Flip nach vorne“. Die Button-Klasse wird später näher erläutert.

4.2.4 checkForExit()

Diese Methode überprüft die Event-Warteschlange auf ein *pygame.quit*-Ereignis, welches durch ein Klick auf das rote „X“ des Fensters verursacht wird. Für diesen Fall wird das Pygame-Modul als auch das gesamte Python-Programm geschlossen.

4.2.5 showText(text, x, y)

Der Befehl “showText” erstellt eine einfache Anzeige eines Textes an einer bestimmten Position. Dies dient lediglich der Präsentation von Daten und Informationen und besitzt keine interaktiven Eigenschaften. In Pygame muss für die Darstellung eines Textes ein kleiner Umweg genommen werden, weil von der Bibliothek keine Funktion dafür bereitgestellt wird. Der Text muss zuerst als Oberfläche – also als eine Art Bild – gerendert werden um anschließend angezeigt werden zu können. Für die Umrandung der Texte wird ein Rechteck von der passenden Größe bezüglich des zuvor erstellten Textbildes hinter diesen gelegt.

4.2.6 showPicture(img, x, y)

Dieser Befehl ähnelt der vorherigen Methoden mit dem Unterschied, dass direkt eine mitgegebene Bilddatei dargestellt werden soll. Auch hier wird wieder ein Rahmen um das Bild gezogen.

4.2.7 checkButton(modus)

In dieser Methode werden die Knöpfe auf Interaktion abgefragt und jedem Button eine spezifische Funktion zugeordnet. Aus der „main“ wird der aktuelle Modus zunächst übernommen. Falls nun einer der Modus-Knöpfe gedrückt wird, wird der Modus dementsprechend angepasst. Die Methode *isPressed*, die hier zum Einsatz kommt, ist in der Klasse Button beschrieben.
Im Anschluss folgt die Überprüfung, welcher Modus aktuell aktiviert ist, um dessen Knopf mit einer roten Umrandung erscheinen zu lassen. Auf diese Weise hebt sich der Button von den anderen ab und der Nutzer erkennt sofort, in welchem Modus er sich zurzeit befindet. Im gleichen Schritt werden die Modi noch auf den Kommunikationsstandard übersetzt und schlussendlich an die „main“ als Steuerungsdaten zurückgegeben.

4.2.8 loadAll(img, height, battery, temperature, speedx, speedy, speedz, modus)

Diese Methode wird in der „Spiel-Schleife“ der „main“ wiederholend aufgerufen, sodass hier alle Methoden Platz finden, die für ihre richtige Funktion dauerhaft aufgerufen werden müssen. Auf diese Weise genügt in der „main“ der Aufruf einer einzelnen Funktion, was die Übersichtlichkeit erhöht. Alle Daten, die in dem Fenster erscheinen sollen, werden an diese Methode übergeben. Die Texte und Bilder mit den empfangenen Daten werden mit Hilfe der zuvor erstellen Methoden in das Fenster gezeichnet. Danach werden die Buttons erzeugt.

4.2.9 loadNotConnected()

Diese Funktion kommt zur Anwendung, wenn die Drohne nicht ordnungsgemäß mit dem Computer verbunden ist. Über ein Text wird dem Benutzer mitgeteilt, dass die Drohne nicht gekoppelt ist.

4.2.10 Klasse „Button“

Die Klasse „Button“ erzeugt interaktive Knöpfe, welche vom Benutzer durch Mausklicken verwendet werden können.

4.2.10.1 Konstanten

Wie auch in der Klasse „Dashboard“ werden Konstanten zur Bestimmung der Position und Farbe der Knöpfe festgelegt.

4.2.10.2 __init__(text, x, y)

Der Konstruktor der Klasse erzeugt ein Button-Objekt. Die Knöpfe besitzen eine identische grafische Gestaltung wie die zuvor erklärten Textfelder. Mit der Übergabe der Parameter ist der Text des Knopfes und die Position in dem Fenster definiert. Es gibt eine zusätzliche Variable *pressed*, die mit „False“ initialisiert wird. Sie ist für die folgende Funktion relevant.

4.2.10.4 isPressed()

Diese Methode überprüft einen Knopf, ob er durch einen Mausklick gedrückt wird. Als erstes wird die aktuelle Position der Maus abgefragt und ob die linke Maustaste gedrückt wird. Falls nun die linke Maustaste gedrückt wird, ohne dass sie davor gedrückt wurde, ist ein neuer Mausklick erkannt worden. Mit der zusätzlichen Bedingung, dass sich die Mausposition

dabei in einem Rechteck eines Button-Objekt befindet, gilt dieser Knopf als gedrückt. Es wird ein „True“ an die *checkButtons*-Methode zurückgegeben. In allen anderen Fällen gilt ein Knopf als nicht gedrückt. Die Funktion ist so aufgebaut, dass die Knöpfe dauerhaft abgefragt werden.

4.2.10.3 showButton(screen)

Damit die Buttons in dem Fenster angezeigt werden, gibt es ähnlich wie die „showText“-Funktion eine *showButton*-Methode. Da sich diese Methode in einer anderen Klasse befindet muss zusätzlich das Fenster aus der Dashboard-Klasse als Übergabeparameter mitgegeben werden.

4.3 Verbindungsaufbau“ beschrieben.
Bei erkannter Verbindung zur Drohne wird eine Instanz der „dronecomms“ zur Kommunikation und Befehlsgebung erzeugt.
Unabhängig von der Drohnenverbindung werden des Weiteren eine Instanz des Moduls „XBoxControl“ und des Dashboards erzeugt.

Die nachfolgende while-Schleife ist nun für alle wiederkehrenden Operationen während der Programmlaufzeit zuständig. Zu Beginn wurde über die Variable *FPS* die Anzahl an Durchläufen dieser Schleife pro Sekunde festgelegt, indem am Ende der Schleife der Programmablauf um 1/FPS Sekunden pausiert wird.
Zunächst werden die aktuellen Eingaben von Controller und Tastatur abgefragt.
Ist die Drohne verbunden, wird das aktuelle Kamerabild abgerufen und entsprechend des aktuellen Modus verarbeitet. Rückgabewerte sind das bearbeitete Bild sowie die eventuellen Steuerungsdaten, sofern ein bildgesteuerter Modus aktiviert wurde.
Im nächsten Schritt werden einige Daten der Drohne wie Flughöhe, Temperatur, Geschwindigkeit und Akkuladung abgefragt. Die Abfrage wird nur einmal pro Sekunde durchgeführt, um die Anzahl der Parameterabfragen zu reduzieren und die Reaktionszeit der Drohne somit erhöhen zu können.

Anschließend werden die aktuellen Daten und das Bild ins Dashboard überführt, welches seinerseits ein passendes Bild für die grafische Oberfläche generiert. Das Dashboard kann dabei ebenfalls Steuerungsdaten zurückgeben (siehe Kapitel „4.1.5 dashboard“).

Um den gewünschten Bildsteuerungs-Modus aktivieren zu können, müssen nun die für den jeweiligen Modus zuständigen Ausgabewerte der Steuerungsschemata verglichen werden. Wird mittels einer der Eingabemethoden der Steuerungsmodus geändert, so wird diese Änderung direkt in der Variable *videostatus* gespeichert und in der nächsten Iteration berücksichtigt, also der Bildsteuerungs-Modus entsprechend geändert.

Nachfolgend werden die Eingaben von Tastatur, Dashboard, Bildverarbeitung und Controller verglichen. Es entsteht eine Hierarchie, bei welcher die Tastatur generell das Vorrecht vor allen anderen Befehlsgebern besitzt. Somit wird gewährleistet, dass die Tastatur immer die Steuerung übernehmen kann, falls eine der anderen Methoden die Drohne unkontrollierbar macht.
Konkret werden alle Indices „0“ der jeweiligen gesendeten Steuerungsschemata verglichen, also ob der Sende-Status eine „1“ (= Daten gesendet) enthält. (Für Details siehe Kapitel „4.4.1 Steuerungsschema“). Nur dann wird die Eingabe der jeweiligen Methode überhaupt beachtet. Die hierarchisch erste Eingabe wird zur Steuerung der Drohne für diesen Durchlauf der Hauptschleife verwendet.
Es ist zu beachten, dass die zuvor beschriebenen Abfragen zur Änderung des Bildsteuerungs-Modus schon vor diesem Vergleich der Methoden erfolgen muss, da es sonst zu Steuerungs-Konflikten kommen könnte.
Ist beispielsweise einer der bildgesteuerten Modi aktiviert, würde stets die Eingabe des „ImageProcessing“ verwendet, weshalb es dem Anwender unmöglich würde, den Bildsteuerungs-Modus über den Controller, welcher in der Hierarchie an letzter Stelle steht, wieder zu deaktivieren. Dies kann nur funktionieren, wenn alle Eingaben in Gänze beachtet werden, während es für die eigentliche Drohnensteuerung notwendig ist, sich auf einen Steuerungsbefehl festzulegen.

Abschließend werden die gewählten Steuerungsdaten an die „dronecomms“ übergeben, welche sie in entsprechende Bewegungsbefehle für die Drohne umwandeln und an diese weiterleiten.

Für den Fall, dass es keine erfolgreiche Verbindung zur Drohne gibt, wird eine Fehlermeldung auf dem erzeugten Fenster angezeigt.

4.2 Benutzeroberfläche

Die folgenden Unterkapitel beschreiben die grafische Benutzeroberfläche. Abbildung 5 zeigt dessen allgemeinen Aufbau.

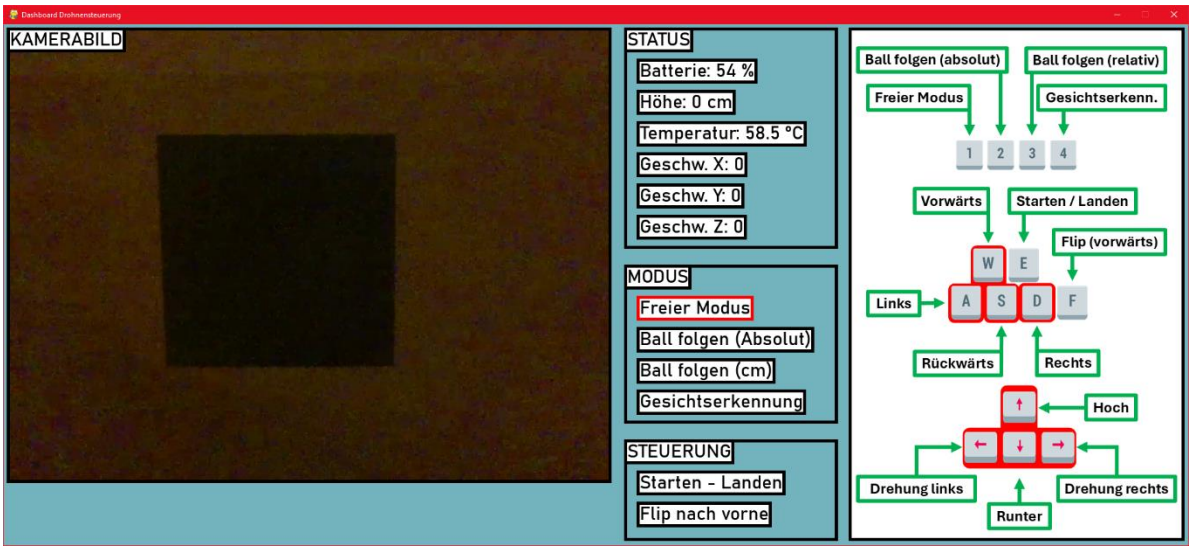

Abbildung 5: Screenshot der Benutzeroberfläche

4.2.1 Allgemeiner Aufbau

Das Dashboard-Skript basiert auf einer gleichnamigen Klasse, in der alle Parameter und Methoden zur Erstellung der Benutzeroberfläche untergebracht sind. Für bestimmte komplexere Grafikelemente mit erweiterten Funktionsumfang bietet es sich in Pygame an eine dedizierte Klasse zu bilden. Dies ist der Fall für die interaktiven Knöpfe in dem Userinterface, sodass eine zweite Klasse für die „Buttons“ in demselben Skript vorzufinden ist. Der Vorteil einer Strukturierung als Klasse ist die Übersichtlichkeit des Programms in der Main und das einfache Ansprechen und Manipulieren von Objektinstanzen.

Zur Erläuterung des geschriebenen Codes werden die erstellten Konstanten und Methoden als Funktionseinheit angesehen und deren Funktionsweise zusammengefasst erörtert.

4.2.2 Konstanten

Um alle Grafikobjekte bezüglich Position, Größe und Farbe zu definieren, bedarf es zu Beginn vieler statischer Konstanten. Dazu gehört auch die Bestimmung der Fenstergröße und

die allgemeine Textgröße. Außerdem wird eine vertikale und horizontale Distanz erzeugt. Beider dieser Distanzen dienen der Positionierung von Anzeigen und Knöpfen, die sich innerhalb eines weiteren Rahmens befinden (siehe Status, Modus und Steuerung).

4.2.3 __init__()

Die Initialisierungsmethode einer Klasse wird für Code benutzt, der lediglich am Anfang der Initialisierung eines Objekts ein einziges Mal ausgeführt werden muss. So kümmert sich die „__init__“ um die Initialisierung von Pygame selbst und um die Generierung eines sich öffnenden Fensters mit Titel. Zusätzlich wird eine Schriftart aus dem internen Speicher des Computers geladen, mit der später alle Texte ausgestattet werden. Damit die Anzeige des Bildes mit der Tastenbelegung ermöglicht werden kann, wird die dementsprechende PNG-Datei in das Programm geladen.
Es folgen nun die vier verschiedenen Umrandungen in Form von erzeugten Rechtecken für die Bereiche des Status, des Modus und der Steuerung.
Im Anschluss werden die Buttons über die eigenständige Button-Klasse und dessen entwickelten Konstruktor initialisiert. Zu den interaktiven Knöpfen zählen sowohl die Steuerung der vier verschiedenen Modi als auch die beiden Aktionen „Starten/Landen“ und „Flip nach vorne“. Die Button-Klasse wird später näher erläutert.

4.2.4 checkForExit()

Diese Methode überprüft die Event-Warteschlange auf ein *pygame.quit*-Ereignis, welches durch ein Klick auf das rote „X“ des Fensters verursacht wird. Für diesen Fall wird das Pygame-Modul als auch das gesamte Python-Programm geschlossen.

4.2.5 showText(text, x, y)

Der Befehl “showText” erstellt eine einfache Anzeige eines Textes an einer bestimmten Position. Dies dient lediglich der Präsentation von Daten und Informationen und besitzt keine interaktiven Eigenschaften. In Pygame muss für die Darstellung eines Textes ein kleiner Umweg genommen werden, weil von der Bibliothek keine Funktion dafür bereitgestellt wird. Der Text muss zuerst als Oberfläche – also als eine Art Bild – gerendert werden um

anschließend angezeigt werden zu können. Für die Umrandung der Texte wird ein Rechteck von der passenden Größe bezüglich des zuvor erstellten Textbildes hinter diesen gelegt.

4.2.6 showPicture(img, x, y)

Dieser Befehl ähnelt der vorherigen Methoden mit dem Unterschied, dass direkt eine mitgegebene Bilddatei dargestellt werden soll. Auch hier wird wieder ein Rahmen um das Bild gezogen.

4.2.7 checkButton(modus)

In dieser Methode werden die Knöpfe auf Interaktion abgefragt und jedem Button eine spezifische Funktion zugeordnet. Aus der „main“ wird der aktuelle Modus zunächst übernommen. Falls nun einer der Modus-Knöpfe gedrückt wird, wird der Modus dementsprechend angepasst. Die Methode *isPressed*, die hier zum Einsatz kommt, ist in der Klasse Button beschrieben.
Im Anschluss folgt die Überprüfung, welcher Modus aktuell aktiviert ist, um dessen Knopf mit einer roten Umrandung erscheinen zu lassen. Auf diese Weise hebt sich der Button von den anderen ab und der Nutzer erkennt sofort, in welchem Modus er sich zurzeit befindet. Im gleichen Schritt werden die Modi noch auf den Kommunikationsstandard übersetzt und schlussendlich an die „main“ als Steuerungsdaten zurückgegeben.

4.2.8 loadAll(img, height, battery, temperature, speedx, speedy, speedz, modus)

Diese Methode wird in der „Spiel-Schleife“ der „main“ wiederholend aufgerufen, sodass hier alle Methoden Platz finden, die für ihre richtige Funktion dauerhaft aufgerufen werden müssen. Auf diese Weise genügt in der „main“ der Aufruf einer einzelnen Funktion, was die Übersichtlichkeit erhöht. Alle Daten, die in dem Fenster erscheinen sollen, werden an diese Methode übergeben. Die Texte und Bilder mit den empfangenen Daten werden mit Hilfe der zuvor erstellen Methoden in das Fenster gezeichnet. Danach werden die Buttons erzeugt.

4.2.9 loadNotConnected()

Diese Funktion kommt zur Anwendung, wenn die Drohne nicht ordnungsgemäß mit dem Computer verbunden ist. Über ein Text wird dem Benutzer mitgeteilt, dass die Drohne nicht gekoppelt ist.

4.2.10 Klasse „Button“

Die Klasse „Button“ erzeugt interaktive Knöpfe, welche vom Benutzer durch Mausklicken verwendet werden können.

4.2.10.1 Konstanten

Wie auch in der Klasse „Dashboard“ werden Konstanten zur Bestimmung der Position und Farbe der Knöpfe festgelegt.

4.2.10.2 __init__(text, x, y)

Der Konstruktor der Klasse erzeugt ein Button-Objekt. Die Knöpfe besitzen eine identische grafische Gestaltung wie die zuvor erklärten Textfelder. Mit der Übergabe der Parameter ist der Text des Knopfes und die Position in dem Fenster definiert. Es gibt eine zusätzliche Variable *pressed*, die mit „False“ initialisiert wird. Sie ist für die folgende Funktion relevant.

4.2.10.4 isPressed()

Diese Methode überprüft einen Knopf, ob er durch einen Mausklick gedrückt wird. Als erstes wird die aktuelle Position der Maus abgefragt und ob die linke Maustaste gedrückt wird. Falls nun die linke Maustaste gedrückt wird, ohne dass sie davor gedrückt wurde, ist ein neuer Mausklick erkannt worden. Mit der zusätzlichen Bedingung, dass sich die Mausposition dabei in einem Rechteck eines Button-Objekt befindet, gilt dieser Knopf als gedrückt. Es wird ein „True“ an die *checkButtons*-Methode zurückgegeben. In allen anderen Fällen gilt ein Knopf als nicht gedrückt. Die Funktion ist so aufgebaut, dass die Knöpfe dauerhaft abgefragt werden.

4.2.10.3 showButton(screen)

Damit die Buttons in dem Fenster angezeigt werden, gibt es ähnlich wie die „showText“-Funktion eine *showButton*-Methode. Da sich diese Methode in einer anderen Klasse befindet muss zusätzlich das Fenster aus der Dashboard-Klasse als Übergabeparameter mitgegeben werden.

4.3 Verbindungsaufbau

Der Verbindungsaufbau zur Drohne sollte einem bestimmten Schema folgen, damit er erfolgreich ist.

Zuerst muss die Drohne eingeschaltet werden. Sobald ihre Status-LED orange blinkt, kann sie über den PC unter den WLAN-Netzwerken gefunden werden und eine Verbindung aufgebaut werden.

Erst danach darf die „main.py“ gestartet werden. Nun sollte die Drohne erkannt werden, was in einer Statusmitteilung in der Konsolen-Ausgabe von PyCharm und einem erfolgreich angezeigten Kamerabild auf der Benutzeroberfläche resultiert.

Sollte dies nicht der Fall sein, behebt sich das Problem zumeist, wenn das Skript noch einmal gestartet wird.

Ist der Verbindungsaufbau auch nach wiederholtem Versuch erfolglos, sollte die Drohne noch einmal aus- und wieder eingeschaltet werden.

4.4 Manuelle Drohnensteuerung

Die Software umfasst zwei Arten der manuellen Steuerung, um die Drohne mittels direkter Kontrolle des Bedieners in der Luft bewegen zu können. Diese werden in den folgenden Kapiteln vorgestellt. Zunächst folgt eine Erklärung des allgemein verwendeten Steuerungsschemas.

4.4.1 Steuerungsschema

Vor der Implementation von Bewegungs-Befehlen für die Drohne war es unerlässlich, eine gemeinsame Basis der Kommunikation zwischen den einzelnen Skripten zu schaffen.
Dementsprechend wurde ein allseits gültiges Steuerungsschema festgelegt, welches von allen befehlsgebenden Funktionen einheitlich erzeugt wird, von den „dronecomms“ stets lesbar ist und von dort aus an die Drohne weitergegeben werden kann.

Das Schema wird als Array von jedem der befehlsgebenden Module „KeyboardControl“, „Dashboard“, „XBoxControl“ und „ImageProcessing“ unabhängig voneinander erzeugt und in der Hauptroutine gesammelt. Für Details zur hierarchischen Verarbeitung siehe Kapitel „4.1.6 main“.
Für den Aufbau des Arrays für das Steuerungsschema ist folgende Reihenfolge der Daten festgelegt:

Index	**Wertebereich**	**Zuständigkeit**
0	0 oder 1	Sende-Status: 0 = keine Daten gesendet, 1 = Daten gesendet
1	1 oder 2	Sende-Modus: 1 = absolute Werte, 2 = relative Werte
2	-100 bis 100	Hoch / Runter
3	-100 bis 100	Drehung
4	-100 bis 100	Vorwärts / Rückwärts
5	-100 bis 100	Rechts / Links
6	0 oder 1	Button 1 (Starten/Landen)
7	0 oder 1	Button 2 (Flip)
8	0 oder 1	Button 3 (Bildsteuerung aus)
9	0 oder 1	Button 4 (Ballverfolgung absoluter Modus)
10	0 oder 1	Button 5 (Ballverfolgung relativer Modus)
11	0 oder 1	Button 6 (Gesichtsverfolgung)

Der Sende-Status wird von einem befehlsgebenden Modul mit dem Wert 1 versehen, sobald Steuerbefehle von ihm gesendet werden. Gibt es dagegen keine Eingaben mittels der

entsprechenden Methode, bleibt der Wert auf 0. Somit wird die Eingabe in der Hauptroutine für diesen Durchlauf nicht beachtet.

Der Sende-Modus bestimmt, welcher Natur die gesendeten Steuerdaten sind. Wird im **absoluten** Modus gesendet, geben die Indices 2 bis 5 **absolute** Geschwindigkeiten an, mit der sich die Drohne in die jeweilige Richtung bewegen soll. Wird der **relative** Modus genutzt, geben diese Indices stattdessen die Entfernung in **Zentimetern** an, die die Drohne in die entsprechende Richtung fliegen soll. In diesem Fall ist der Wertebereich entgegen den Angaben in der Tabelle nicht begrenzt.

Die Indices 6 bis 11 beziehen sich schließlich auf frei definierbare Buttons, die frei zuweisbare Befehle wie das Starten und Landen der Drohne ausführen können.
Bei der Weitergabe an die „dronecomms“ wird der Index 0 ignoriert, da dieser nur für die Reihenfolge innerhalb der Hauptroutine wichtig ist.

4.4.1 Per Tastatur

Die Steuerung der Drohne per Tastatur erfolgt über die Pfeiltasten sowie über das für Spiele übliche „WASD“-Schema. Dabei lässt sich per Pfeiltasten rauf/runter die Höhe der Drohne steuern, die Pfeile nach links und rechts steuern die Rotation. Die Tasten „W“ und „S“ bestimmen die Bewegung vorwärts und rückwärts, die Tasten „A“ und „D“ die Bewegung in seitliche Richtung.
Somit lässt sich das gesamte Bewegungsspektrum der Drohne gleichzeitig mit beiden Händen kontrollieren.
Da die Tasten nur binäre Eingaben beherrschen, wird die Geschwindigkeit auf entweder 0 Prozent oder 100 Prozent gesetzt.
Über die Nummern-Tasten „1“ bis „4“ lassen sich die einzelnen Modi durchwechseln. „1“ deaktiviert dabei den bildgesteuerten Betrieb, während Taste „2“ die Ballverfolgung im absoluten Modus, „3“ die Ballverfolgung im relativen Modus und „4“ die Gesichtserkennung aktiviert. Die Drohne kann über die Taste „E“ gestartet und gelandet werden. „F“ führt einen experimentellen Vorwärtssalto aus.

4.4.2 Per Controller

Der Controller steuert die Hauptbewegungen der Drohne über die beiden Joysticks. Dabei ist – analog zur Tastatursteuerung – der rechte Joystick für die Auf- und Abbewegung sowie die Rotation zuständig, der linke Joystick bestimmt die Bewegung in die horizontalen Richtungen. Da bei dem Controller entgegen der Tastatursteuerung die Auslenkung der Joystick von 0 bis 100 Prozent stufenlos bestimmbar ist, definiert hiermit der Ausschlag der jeweiligen Steuerachse die korrespondierende Bewegungsgeschwindigkeit.
Der Button „A“ wird zum Starten und Landen der Drohne benutzt, wobei die Software selbstständig erkennt, ob die Drohne gerade am Boden oder in der Luft ist. Einzige Ausnahme dieser Erkennung stellt ein möglicher Absturz der Drohne oder durch ein in Kapitel „4.6.3 Softwareseitige Fehler“ beschriebenes Problem dar. In diesem Fall muss der Button „A“ zweimal gedrückt werden.
Der Button „B“ führt einen experimentellen Vorwärtssalto aus.

Mittels des Steuerkreuzes lassen sich schließlich die unterschiedlichen bildgesteuerten Modi, beschrieben in Kapitel „4.5 Bildgesteuerter Betrieb“, aktivieren. Der Button „X“ deaktiviert den bildgesteuerten Betrieb wieder.
Das Steuerkreuz nach oben aktiviert die Ballverfolgung im absoluten Modus, nach links aktiviert die Ballverfolgung im relativen Modus und nach unten letztendlich die Gesichtserkennung.

4.5 Bildgesteuerter Betrieb

Der bildgesteuerte Betrieb der Drohne kann wie in den vorigen Kapiteln beschrieben aktiviert werden und besitzt drei unterschiedliche Modi. Sie alle stützen sich auf die Verarbeitung der Bilddaten der Drohnenkamera und erzeugen daraus jeweils eine Steuerungsanweisung, welche, wie auch bei der manuellen Steuerung, der Drohne übergeben wird.
Im Folgenden werden die einzelnen Modi und ihre Arbeitsweisen beschrieben.

4.5.1 Modus 1: Ballverfolgung mit absoluten Steuerungswerten

Dieser Modus lässt die Drohne einen grünen Ball mit einem Durchmesser von ungefähr acht Zentimetern (Siehe Abbildung 6) fokussieren und in alle drei Raumrichtungen verfolgen.

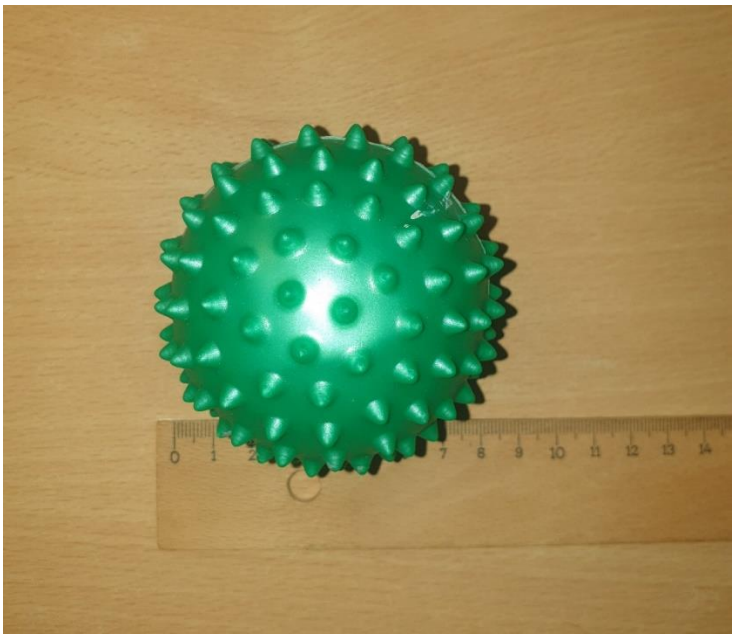

Abbildung 6: verwendeter Ball

Dafür wird das Bild zunächst in den HSV-Farbraum konvertiert und danach mit einem Threshold zwischen HSV(35, 30, 40) und HSV(90, 255, 253) maskiert. Die zwischen diesen Farbwerten erkannten Konturen werden anschließend ausgewertet.
Es wird dabei immer mit der flächenmäßig größten Kontur gearbeitet. Solange diese Kontur ein Polygon mit fünf oder mehr Ecken darstellt und eine Fläche von 500 Quadratpixeln nicht unterschreitet, verarbeitet die Software die Koordinaten der Kontur. Es wird zunächst eine Ellipse um die Kontur gelegt und anschließend deren Position im Bild bestimmt. Bezogen auf den Bildmittelpunkt wird mit diesen Daten nun die horizontale sowie vertikale Differenz des Mittelpunkts der Kontur berechnet.

Mittels der Differenzen werden anschließend die Steuerungsdaten berechnet. Für eine präzise Steuerung über absolute Bewegungsbefehle, also der reinen Angabe einer aktuellen Geschwindigkeit in eine bestimmte Richtung, ist es nötig, dass die vorgegebene Geschwindigkeit mit der Entfernung zum Bildmittelpunkt skaliert wird. Andernfalls würde

die Drohne bei einer nur kleinen Differenz in die entgegengesetzte Bildrichtung über den angezielten Bildmittelpunkt hinausschießen.
Entsprechend wurden vorab feste Skalierungswerte erprobt und diese mittels einer linearen Funktion auf die aktuelle Differenz in horizontale und vertikale Bildrichtung projiziert.
Ein ähnliches Vorgehen mit abweichenden Werten wurde auch für die Entfernung der Drohne zum Ball umgesetzt. Hier wird die Fläche der erkannten Kontur als Referenz benutzt und basierend darauf eine Geschwindigkeit zum Ball hin oder vom Ball weg festgelegt.
Hat der Ball nur einen geringen Abstand vom Bildmittelpunkt, wird der Drohne eine Geschwindigkeit von Null in der entsprechenden Raumrichtung mitgeteilt, was ihre Bewegung in eben dieser Richtung stoppt.

4.5.2 Modus 2: Ballverfolgung mit relativen Steuerungswerten

Der zweite Modus verfolgt dasselbe Ziel wie auch der erste Modus. Er setzt die Differenz zwischen Konturmittelpunkt und Bildmitte allerdings direkt in relative Werte um, gibt der Drohne also präzise Entfernungswerte in Zentimetern, um den Ball zu zentrieren.
Hierbei galt es zu beachten, dass sich der maximale seitliche Abstand des Balls zur Bildmitte (in Zentimetern) mit zunehmender Entfernung des Balls zur Kamera ebenfalls vergrößert.
Um nun den ungefähren realen Abstand zur Mittelachse der Kamera zu berechnen, wurde vorab für zwei unterschiedliche Konturflächen (1000 und 2000 Px^2) für den jeweiligen Abstand des Balls zur Drohne die sichtbare Bildbreite ermittelt. Daraus ergibt sich eine lineare Funktion, mit welcher basierend auf der aktuellen Konturfläche die zunächst in Pixeln bemessene Differenz der Ballkontur zur Ballmitte auf reale Werte umgerechnet werden kann. Diese Differenz kann im letzten Schritt an die „dronecomms“ übergeben werden und die Drohne entsprechend präzise an die gewünschte Stelle bewegt werden.
Es gilt bei diesem Steuerungsmodus zu beachten, dass die Drohne keine Bewegung von weniger als 20 Zentimetern ausführen kann. Weitere Erläuterungen dazu finden sich in Kapitel „4.6.2 Zentimetergenaue Steuerung“.

4.5.3 Modus 3: Statische Verfolgung von Gesichtern

Als weiteres Experiment wurde im dritten Modus eine Verfolgung von Gesichtern umgesetzt, welche die Drohne im Gegensatz zu den ersten beiden Modi allerdings nicht in den drei Raumrichtungen folgen lässt, sondern eine Drehung der Drohne initiiert.

Der Modus nutzt die bereits vorgestellte Bibliothek „face-library“.

Wird er aktiviert, ermittelt die Software mittels dieser Bibliothek eventuelle Gesichter im Kamerabild der Drohne und nutzt das jeweils zuerst erkannte Gesicht. Mittels der Koordinaten des erkannten Umrisses wird der horizontale Abstand des Gesichts zur Bildmitte errechnet.

Durch den vorab ermittelten maximalen Blickwinkel der Kamera, welcher etwa 65 Grad beträgt, lässt sich nun die nötige Drehung berechnen, welche nötig ist, um das Gesicht zu zentrieren.

Die Drohne muss für diesen Modus zunächst auf eine entsprechende Höhe oberhalb des Kopfes des Anwenders gesteuert werden.

4.6 Schwierigkeiten und Probleme

Die Erprobung der Software- und Steuerungsmöglichkeiten hat einige Schwierigkeiten und Probleme in der Umsetzung der Software und Handhabung der Drohne aufgezeigt, welche im Folgenden erläutert werden.

4.6.1 Verbindungsqualität und -geschwindigkeit

Die Verbindung zwischen Drohne und PC stellte sich mitunter als instabil heraus.
Schon bei versuchtem Verbindungsaufbau kann es vorkommen, dass die Drohne unter der für sie üblichen IP-Adresse (192.168.10.1) nicht immer erreichbar ist. Dies lässt sich meist nur beheben, indem das WLAN des PCs deaktiviert und wieder aktiviert wird.
Zudem schaltet sich die Drohne nach einer gewissen Zeit von selbst aus. Es liegt die Vermutung nahe, dass sich das Gerät dadurch vor Überhitzung schützt. Dafür spricht, dass sich die Platine der Drohne im verbundenen, aber nicht fliegenden Zustand auf über 90°C erhitzt. Bei aktiver Kühlung im Flug werden diese Temperaturen nicht erreicht. Entsprechend führen kurze Flugpausen, etwa im Zuge von Code-Anpassungen dazu, dass die Drohne anschließend neu gestartet und verbunden werden muss.
Mitunter ist es in seltenen Fällen auch schon vorgekommen, dass die Drohne während des Fluges kein Videosignal mehr geliefert hat oder automatisch gelandet ist, was eventuell ebenfalls auf eine schlechte Verbindungsqualität zurückzuführen ist, aber nicht abschließend geklärt werden konnte.

Die Verbindungsgeschwindigkeit beeinflusst maßgeblich einerseits die Reaktionsfähigkeit der Drohne auf eingehende Signale, andererseits aber auch die Aktualisierungsrate des Kamerabildes, was sich seinerseits insbesondere im bildgesteuerten Modus wieder auf die Geschwindigkeit der Steuerungsbefehle auswirkt. Zudem weist das gesendete Bild eine nicht unwesentliche Latenz auf, welche eine präzise und zeitgenaue Befehlsgebung zusätzlich erschwert. Letztendlich kann man dadurch nicht von einer verlässlichen optischen Rückmeldung ausgehen, welche die Steuerung träge wirken lässt.

Die Software wurde auf unterschiedlichen Computern getestet, wobei die Verbindungsgeschwindigkeit je nach Gerät stark variierte.

Als Beispiel dient an dieser Stelle der Vergleich zwischen einem PC ohne eigenes WLAN-Modul (die Verbindung wurde über einen USB-WLAN-Stick hergestellt) und einem Microsoft Surface Pro 7.

Dabei war auffällig, dass bereits die Steuerungseingaben über Tastatur und Controller bei den Tests am PC um einige Millisekunden verzögert von der Drohne umgesetzt wurden, während die Reaktionen über das Surface deutlich schneller eintraten. Dies dürfte auf die genannte Verwendung eines USB-WLAN-Sticks am PC zurückzuführen sein, welcher allgemein eine sehr träge Netzwerkkonnektivität aufweist.

Der Vergleich zeigt jedoch auf, dass bereits die Art und Qualität der WLAN-Verbindung eine wichtige Rolle bei der gezielten Steuerung der Drohne spielt und zu Problemen führen kann.

4.6.2 Zentimetergenaue Steuerung

Während die Steuerung der Drohne über absolute Geschwindigkeitsbefehle in die verschiedenen Richtungen problemlos funktioniert, bringt die Steuerung mittels Angabe von Entfernungen zwei essenzielle Probleme auf.

Einerseits kam es während den Tests häufig zum immer gleichen Fehler beim Senden solcher Steuerbefehle („error No valid imu“), der nach einiger Recherche auf eine zu geringe Umgebungsbeleuchtung zurückzuführen zu sein scheint. Auch die Bedienungsanleitung weist auf diese Problematik hin.(Siehe Abbildung 7) Selbst bei einem gut beleuchteten Raum konnte es selten dazu kommen, dass dieser Fehler auftrat, was die Drohne zur sofortigen Landung zwingt. Dies schränkt die Zuverlässigkeit dieser Methode nicht unerheblich ein und macht ein Testen deutlich komplizierter.

f. in Bereichen mit häufigem oder starkem Lichtwechsel;
g. beim Überfliegen sehr dunkler (< 10 Lux) oder heller (> 100.000 Lux) Flächen oder beim Flug in Richtung heller Lichtquellen (z. B. der Sonne entgegen);
h. beim Überfliegen von Flächen ohne deutliche Muster oder Konturen;
i. beim Überfliegen von Flächen mit wiederkehrenden Mustern oder Strukturen (z. B. Fliesen);
j. beim Überfliegen von kleinen oder empfindlichen Gegenständen (z. B. Baumzweige oder Stromleitungen);
k. beim Fliegen bei Geschwindigkeiten von über 18 km/h auf 1 Meter oder niedriger;

- Bei starker Dunkelheit (< 100 Lux) erkennt das optische Positionsbestimmungssystem möglicherweise kein Muster am Boden. Starten Sie das Fluggerät NICHT, wenn eine Warnaufforderung in der Tello App angezeigt wird, die Sie darauf hinweist, dass die Umgebung zu dunkel ist.

Abbildung 7: Auszug aus Bedienungsanleitung, S. 7

Andererseits lassen Drohne respektive Bibliothek „djitellopy“ es nicht zu, Strecken von weniger als 20 Zentimetern zu fliegen. Dadurch wird es unmöglich, die Drohne mittels Kamera zentimetergenau auf ein gewünschtes Objekt auszurichten.
Kombiniert mit der trägen Verbindung lässt sich dieses Verhalten auch nicht simulieren, indem die Drohne beispielsweise in die gewünschte Richtung gesteuert und im passenden Moment gestoppt wird.

Zu den einschränkenden Steuerungsmöglichkeiten kommen physische Probleme der Drohne hinzu. Je nach Umgebung driftet die Drohne im Raum umher, etwa durch einen Luftzug oder durch an Objekten zurückgestoßener Luftströme, welche die Drohne durch ihre Propeller selbst erzeugt hat. Tests haben ergeben, dass sich die Drohne in offener Umgebung, beispielsweise im heimischen Garten, stabiler verhält als in geschlossenen Räumen. Es scheint, als dass die Drohne ihre Eigenbewegung nicht fein genug erkennen und entsprechend reagieren kann.

4.6.3 Softwareseitige Fehler

Als weiteren Punkt ist anzumerken, dass die genutzte Bibliothek „djitellopy“ nur rudimentär dokumentiert ist und im Laufe der Entwicklung immer wieder Fehler aufgetreten sind, welche nur schwer oder gar nicht nachvollziehbar waren. Somit wurde die Drohne öfters zu automatischen Landungen gezwungen, ohne dass ein offensichtlicher Grund dazu bestand.
Nachforschungen und weitere Versuche haben oft gezeigt, dass die Fehler nicht reproduzierbar sind und entsprechend auch nicht weiter analysiert werden konnten.

5. Zusammenfassung und Ausblick

In dieser Projektarbeit wurden verschiedene Methoden und Wege entwickelt, die Drohne DJI „Tello“ über einen PC zu steuern. Dabei wurden mehrere Python-Skripte programmiert, welche in Zusammenarbeit eine benutzerfreundliche Oberfläche für eine einfache Steuerung und Überwachung des Fluggeräts zur Verfügung stellt.

Der Anwender kann zwischen einer manuellen Steuerung mittels Tastatur oder Controller wählen und die Parameter der Drohne sowie ihr Kamerabild mittels einer auf der Bibliothek „Pygame“ basierenden Benutzeroberfläche überwachen. Weiterhin lässt sich die Drohne automatisch mittels dreier bildverarbeitender Modi steuern, von denen zwei einen vorher bestimmten Ball erkennen und zentrieren können und einer dem Gesicht eines Menschen mittels einer Rotation folgen kann.

Mit der Benutzung von „OpenCV“ wurde eine der am weitesten verbreiteten Bibliotheken in der Bildverarbeitung verwendet, welche noch viele weiterführende Methoden beinhaltet, als in diesem Projekt genutzt wurden. Entsprechend bietet ihre Nutzung Potential für weitere Experimente.

Zudem hat sich „Pygame“ als Bibliothek für die Benutzeroberfläche als gut geeignet herausgestellt, da sie sehr flexibel auf die hier benötigten Grafikelemente angepasst werden konnte und zudem einen Lernaspekt zum grundlegenden Thema der Grafikausgabe bot. Dem gegenüber steht allerdings der Fakt, dass durch die rudimentäre Art der Bibliothek der Einstieg in die Programmierung mit selbiger erschwert war. Entsprechend hätten andere, fortgeschrittenere Grafik-Bibliotheken durchaus unkomplizierter zu demselben Ergebnis führen können.

Des Weiteren haben sich einige Probleme in der Handhabung der Drohne herauskristallisiert. Neben einem teils erschwerten Verbindungsaufbau ergaben sich auch bei der zur Steuerung verwendeten Bibliothek „djitellopy“ einzelne Hürden. Es traten vereinzelte Abstürze seitens dieser Bibliothek auf. Zusätzlich wurden auch mehrere Restriktionen beim Senden von Bewegungsbefehlen ersichtlich, was beispielsweise die zentimetergenaue Ausrichtung erschwert hat. Kombiniert mit hoher Latenz in der WLAN-Verbindung kann man festhalten, dass die Drohne für einfache Steuerungen gut geeignet ist, es sich jedoch als problematisch

darstellt, sie mittels komplexerer Berechnungen und Kommunikation punktgenau im Raum zu positionieren.

Zukünftige Experimente auf Basis dieser Arbeit könnten sich auf eine eigens entwickelte Bibliothek konzentrieren, welche die „djitellopy“ ersetzen und eventuell bessere Ergebnisse und eine genauere Steuerung erzielen könnte.
Zudem könnte eine andere Grafikoberfläche genutzt werden, auf welcher die Drohne möglicherweise per Mauszeiger über virtuelle Joysticks steuerbar ist.
Es waren ursprünglich weitere Modi im Zuge der Bildverarbeitung geplant, beispielsweise eine Linienverfolgung mittels Umlenkspiegel und ein Modus, in dem eine Drohne eine andere verfolgt. Da allerdings durch die bereits entwickelte Software schnell klar wurde, dass diese Modi nicht ohne größere Schwierigkeiten durch die genannten Probleme umsetzbar wären, könnte eine Implementation dieser Erweiterungen in einem separaten Forschungsprojekt geschehen.