



Fakultät Elektrotechnik Feinwerktechnik
Informationstechnik

DRONEVERSE

Projektbericht im Studiengang Media Engineering

vorgelegt von

Sina Steinmüller, 3637368

Tobias Schwarz, 3627943

Maximilian Richter, 3642353

Lisa Berbig, 3627969

Anusha Kanagarasa, 3631692

Katharina Hölzl, 3579055

Betreuer: Prof. Dr. (USA) Ralph Lano

© 2024

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

1	Einführung	2
2	Projektbeschreibung	4
2.1	Zusammenfassung des Zwischenstandes im Wintersemester	6
2.1.1	Gestenerkennung	6
2.1.2	Steuerung der Tello Drohne	7
2.1.3	Netzwerkverbindung	7
2.1.4	Kollisionserkennung	8
2.1.5	Quadkopter	9
2.2	Netzwerkverbindung	10
2.2.1	Probleme durch einen zu langsamen Router	10
2.2.2	Statische IP-Adressen	10
2.2.3	Mögliche Vorteile der 5 GHz Frequenz	10
2.2.4	Vorteile der Drohnensteuerung über einen Router	11
2.2.5	Verwendung der Netzwerkverbindung für OSC	11
2.3	Softwarearchitektur	12
2.3.1	Gestenerkennung	16
2.3.1.1	Erkennung von oben und unten durch Regionen	17
2.3.1.2	Vorwärts- und Rückwärtserkennung durch Distanzberechnungen	18
2.3.1.3	Erkennung von rechts und links	19
2.3.1.4	Praxiserfahrung	20
2.3.2	Handyausrichtungserkennung (OSC)	22
2.3.2.1	Erstmaliges Empfangen von OSC Nachrichten	22
2.3.2.2	Empfangen und Verarbeiten der OSC Werte	24
2.3.2.3	Nutzung des iPhones zur Richtungserkennung	26
2.3.3	Keyboard Erkennung	29
2.3.3.1	Initialisierung	30
2.3.3.2	Handling der KeyDown-Events	31
2.3.3.3	Handling der KeyUp-Events	32

2.3.4	Kollisionserkennung	33
2.3.4.1	Notwendigkeit einer Kollisionserkennung	33
2.3.4.2	Differenzbild zur Positionsbestimmung der Drohne	33
2.3.4.3	Erkennung der Drohne in den Randbereichen	34
2.3.4.4	Eingreifen in die Drohnensteuerung	34
2.3.4.5	Problem bei der Integration in der Drohnensteuerung	35
2.3.4.6	Lösungsansätze zur Integration in die Drohnen- steuerung	35
2.3.4.7	Verwendung einer 360° Kamera als 180° Kamera	36
2.3.4.8	Kollision bei der Projektpräsentation	37
2.3.4.9	Fazit zur Kollisionserkennung	38
2.3.5	Tello Dronecontrol	39
2.3.5.1	Initialisierung	40
2.3.5.2	Übergabe-Methode	41
2.3.5.3	Richtungs-Methoden	41
2.3.6	Quadcopter Dronecontrol	43
2.4	Quadcopter Hardware	44
2.4.1	Erste Ideen	44
2.4.2	Das Luftschiff	45
2.4.3	ESP32-Versuche	46
2.4.4	Programmierung des ESP	48
2.4.5	Tastatursteuerung	50
2.4.6	Bau des neuen Luftschiffs	52
2.4.7	Aussicht	54
2.5	Hindernis-Parcours	56
2.5.1	Gestaltung und Umsetzung des Hindernis-Parcours	56
2.5.2	Planung & Konstruktion	56
2.5.3	Konstruktion mit Shapr3D	60
2.5.4	Aufbau des Parcours	61
2.5.5	Aussicht zum Hindernis-Parcours	61
2.5.6	Finale Bilder der Umsetzung	62
2.5.7	Designerstellung der Highscore-Anzeige	63
2.5.8	Entwicklung der Highscore-Anzeige	63
2.5.9	Integration eines Timers basierend auf der Ausgabe eines Sensors	69
2.5.9.1	Herausforderung	69

2.5.9.2	Recherche zu den Sensoren Fototransistor und LDR (Light Dependent Resistor)	69
2.5.9.3	Recherche zu einem Abstandssensor	70
2.5.9.4	Ergebnis der Recherche	71
2.5.9.5	Ultraschallsensor HC-SR04	71
2.5.9.6	Konstruktion des Startblocks	77
2.5.10	Raumgestaltung	80
2.5.10.1	Plakate	82
3	Design Integration	85
3.1	Marketingstrategie	85
3.2	Produktion von Foto- und Videomaterialien	89
3.2.1	Drehbuch	90
3.3	Werbematerial	91
3.4	Webseite	95
4	Corporate Identity	100
4.1	Markenidentität	100
4.1.1	USP (Alleinstellungsmerkmal)	100
4.1.2	Zielgruppe	101
4.2	Das Logo	102
4.2.1	Aussage	102
4.2.2	Größenangabe	104
4.3	Farbsystem	104
4.3.1	Farbverhältnis	104
4.3.2	Farbpalette	105
4.4	Typografie	106
4.4.1	Schrift	106
4.4.2	Verwendung	106
5	Zusammenfassung	107
	Abbildungsverzeichnis	109
	Listings	111
	Bibliographie	113

Kapitel 1

Einführung

(Sina Steinmüller)

Das Projektjahr im 5. und 6. Semester des Studiengangs Media Engineering ist das Herzstück des Studiengangs. Anders als bei anderen Studienfächern dauert diese Projektarbeit ein ganzes Jahr und ist daher unterteilt in zwei Hauptphasen: Die erste Phase endet mit der Zwischenpräsentation nach 4 Monaten, bei der ein Proof-of-Concept präsentiert werden soll. Dieses Proof-of-Concept soll dann in der zweiten Phase zu einem idealerweise funktionierenden Prototypen umgesetzt werden und am Tag der Abschlusspräsentation gezeigt werden. Neben diesem Prototypen soll das Projekt auch bei der Präsentation in der Kürze von 10 Minuten, aussagekräftig beschrieben werden. Als besonderes Highlight des Studiengangs gibt es nach der Zwischenpräsentation auch eine Projektfahrt, bei der man die Gelegenheit hat, 3 Tage am Stück mit dem Team zusammen das Projekt weiterzuentwickeln.

Im Projekt DRONEVERSE wurde die Gelegenheit geboten, die Aufgabenstellung „Interaktive Drohnensteuerung mit Luftballons“ nach eigenen Vorstellungen auszuarbeiten und umzusetzen. Dabei war in der ersten Hälfte des Projekts das Ziel, Forschung zu betreiben und auf Basis dieser Erkenntnisse die Aufgabenstellung für das zweite Semester zu definieren. Erforscht wurden hierfür Fragestellungen wie „Welche Möglichkeiten gibt es, die Drohnensteuerung interaktiv zu gestalten?“, „Kann man mit den Quadkopter-, oder Tello-Drohnen eine interaktive Drohnensteuerung realisieren?“, „Welche Steuerungsmöglichkeiten sind sinnvoll und wie können sie umgesetzt werden?“.

Nach intensiver Forschung und Planung fand die Zwischenpräsentation statt, bei der es von vielen Studierenden, Professoren wie auch anderen Besuchern, Feedback gab, was eine genauere Definition des Projektziels bewirkte. Obwohl es viele Möglichkeiten und Ideen gab, wie das Projekt final aussehen könnte, wurde sich

dafür entschieden, dass es am Tag der Abschlusspräsentation möglich sein sollte, Drohnen mittels Gestensteuerung durch einen Hindernisparcours zu steuern und dabei möglichst viele Punkte zu erzielen. Das ganze Projekt sollte nicht nur langweilig präsentiert, sondern von allen spielbegeisterten und experimentierfreudigen Zuschauern gespielt werden können und Spaß bringen. Ein kleiner Spoiler an dieser Stelle: Dies ist uns gelungen. DRONEVERSE war am Tag der Projektpräsentation das absolute Highlight – Studierende und Professoren steuerten begeistert die Drohnen durch den Hindernisparcours, hatten großen Spaß dabei und konkurrierten um den Highscore. Außerdem fand die Unternehmenskommunikation der Ohm das Projekt so beeindruckend, dass sie uns in ihrem Instagram-Post erwähnten und das Projekt im Ohmjournal veröffentlicht werden soll. Von der Aufgabenstellung des Projekts bis zur tatsächlichen Umsetzung war es ein langer Weg, der in den folgenden Kapiteln näher beschrieben wird.

Kapitel 2

Projektbeschreibung

(Sina Steinmüller)

Im Rahmen des Projekts DRONEVERSE wurde im vergangenen Jahr ein einzigartiges Spiel entwickelt, das zum Ziel hat, Drohnen mit Hilfe verschiedener Steuerungsarten durch einen Hindernis-Parcours zu navigieren und dabei den Highscore zu knacken. Zur Steuerung der Drohnen können Spieler zwischen verschiedenen Optionen wählen: Gestenerkennung (2.3.1), Handyausrichtungserkennung (OSC, 2.3.2) oder Tastatureingabeerkennung (2.3.3). Zusätzlich standen zwei Drohnentypen zur Verfügung: einer DJI Tello-Drohne (2.3.5) oder einem Quadkopter (2.4), der mit Heliumluftballons ausgestattet war.

Nach Auswahl der Steuerungsmethode und Art der Drohne über das User Interface (2.3) startet das Spiel und die Drohne musste nun den Parcours (2.5) durchfliegen, der aus drei Haupthindernissen bestand: Durchfliegen von drei Ringen, das Navigieren in Schlangenlinien um LED-Säulen und ein Salto durch ein Hindernis mit zwei Öffnungen. Während des gesamten Spiels wurde die Zeit gemessen (2.5.9.1), und bei erfolgreicher Landung wurde die Bestzeit (2.5.8) notiert.

Das Projekt DRONEVERSE setzt sich aus drei wesentlichen Teilen zusammen:

1. Forschung und Recherche: Im ersten Semester erfolgte eine umfassende Recherche, die zur genauen Aufgabenstellung führte und das Fundament für das DRONEVERSE-Projekt legte. Diese Phase umfasste die Untersuchung von Erkennungsmethoden, die Wahl der Programmiersprache, die Vor- und Nachteile der verschiedenen Drohnenmodelle und die Entscheidung für den Hindernis-Parcours.

2. Praktische Umsetzung: Im zweiten Teil wurde das Spiel entwickelt, wobei sowohl Hardware- (2.4) als auch Softwarekomponenten (2.3) integriert wurden. Die Software des Projekts ist auf [Github](#) verfügbar.
3. Design und Marketing: Der dritte Teil des Projekts widmete sich der Designintegration (3), einschließlich der Marketingstrategie, des Social Media Marketings sowie der Erstellung von Foto- und Videomaterial (3.2). Weitere Informationen hierzu sind im Kapitel 3.3 zu finden. Alle Medien können in der [FAUbox](#) heruntergeladen werden.

Eine detaillierte Darstellung der zugrunde liegenden Forschung und die wichtigsten Erkenntnisse bis zur Zwischenpräsentation sind im [Zwischenbericht](#) zu finden. Eine Zusammenfassung dieser Ergebnisse folgt im nächsten Kapitel (2.1).

2.1 Zusammenfassung des Zwischenstandes im Wintersemester

(Sina Steinmüller)

Im Zwischenprojektbericht wurde sich intensiv mit der Corporate Identity von DRONEVERSE beschäftigt. Dabei ging es um die Markenidentität, die Definition unserer Zielgruppe und Gestaltungselemente wie Logo, Farben und Typografie. Auch die Marketingstrategien, insbesondere im Bereich Social Media, sowie die Pläne für die Zukunft wurden erläutert.

Auf technischer Seite wurde das Projektmanagement mit Obsidian und die Entwicklung einer interaktiven Drohnensteuerung mithilfe von Gestenerkennung vorangetrieben. Es wurden die Tello-Drohne und ihre technischen Komponenten näher untersucht, verschiedene Steuerungsmöglichkeiten für mehrere Drohnen ausprobiert. Außerdem haben wir uns mit der Vermeidung von Kollisionen befasst. Auch externe Komponenten wie LED-Module und Heliumballons wurden in die Überlegungen miteinbezogen.

Zum Abschluss wurde der aktuelle Stand der Technik in den Bereichen Gestenerkennung und Drohnenschwärme betrachtet, viele Paper gelesen und einbezogen und einen Ausblick auf die nächsten Schritte gegeben, wie z. B. der Hindernisparcours oder das Drohnenmultiplayergame. Auch die Raumgestaltung für unsere Präsentation stand auf der Agenda, damit wir unsere Fortschritte für die Abschlusspräsentation bestmöglich präsentieren konnten.

2.1.1 Gestenerkennung

(Sina Steinmüller)

Für die Zwischenpräsentation lag der Schwerpunkt der interaktiven Steuerung auf dem Testen verschiedener Ansätze und deren Bewertung. Nach sorgfältiger Abwägung wurde die Gestenerkennung anderen Methoden wie Sprach-, Gesichts- und Körpererkennung vorgezogen. Verschiedene Programmieransätze wurden ausprobiert, darunter die Nutzung von C++ in einer virtuellen Ubuntu-Umgebung. Diese erwies sich jedoch als zu langsam. Eine Echtzeit-Gestenerkennung mit React.js und

Tensorflow.js zeigte gute Ergebnisse, scheiterte jedoch an der dauerhaft notwendigen Internetverbindung, die aufgrund der Netzwerksicherheit an der TH Ohm nicht realisierbar war. Nach einigen Rückschlägen fiel schließlich die Entscheidung auf Python und MediaPipe Handpose, was sich als der vielversprechendste Ansatz herausstellte, der letztlich die Grundlage für DRONEVERSE bildete.

2.1.2 Steuerung der Tello Drohne

(Maximilian Richter)

Vorerst wurde die Ansteuerung der Tello Drohne über Methoden, die aus der DJITelloPy-Bibliothek importiert wurden, umgesetzt. Hierbei ist die Drohne nach dem Erkennen eines Steuerungsbefehls um einen Wert $\mathbf{x}$ in die entsprechende Richtung geflogen (z.B.: `drone.move_up(x)` – Drohne fliegt um $\mathbf{x}$ cm nach oben).

Mit dieser Steuerungsmethode ist eine auf Gesten basierte Echtzeitsteuerung, wie wir unter anderem bei der Zwischenpräsentation festgestellt haben, eher schwierig zu bedienen.

Aus diesem Grund wurde für das folgende Semester eine Methode verwendet, bei der vier verschiedene Geschwindigkeiten an die Drohne übergeben werden, welche vorgeben, in welche Richtung die Drohne fliegt.

2.1.3 Netzwerkverbindung

(Lisa Berbig)

Die Verbindung zwischen der Tello Drohne und dem Computer erfolgt normalerweise über ein Netzwerk, das die Drohne selbst aufbaut. Mit diesem kann sich dann mit einem Endgerät verbunden und der Drohne die Steuerungsbefehle übermittelt werden.

Dies führte im letzten Semester zu dem Problem, dass von einem Computer nur eine Drohne auf einmal angesteuert werden konnte, da sich der Computer nicht mit mehreren Netzwerken gleichzeitig verbinden kann.

Um trotzdem mehrere Drohnen gleichzeitig ansteuern zu können, wurde im letzten Semester an verschiedenen Lösungen gearbeitet. Es wurde versucht, eine Verbindung

zwischen der Drohne und einem ESP32 herzustellen, der dann über USB an den Computer angeschlossen werden kann. Des Weiteren gab es Versuche mit dem Promiscuous-Mode, der es ermöglichen sollte, sich mit mehreren Netzwerken gleichzeitig zu verbinden.

Schließlich konnte eine gute Lösung darin gefunden werden, die Tello Edus in ein externes Netzwerk einzubinden. Dies ist zwar leider nur eine Eigenschaft der Edu Version der Tello Drohnen, ermöglicht es aber mit dem Aufruf eines kurzen Python-Programmes, die Drohne in ein Netzwerk einzubinden. Mit diesem Netzwerk bleibt die Drohne auch verbunden, wenn sie aus- und wieder eingeschaltet wird. Wenn die Drohne doch wieder über ihr eigenes Netzwerk gesteuert werden soll, kann sie durch langes Drücken des Einschaltknopfes wieder auf Werkseinstellungen zurückgesetzt werden.

Ende letzten Semesters war es also möglich, Drohnen in ein externes Netzwerk einzubinden, was zunächst über einen kleinen, älteren Router getestet wurde.

2.1.4 Kollisionserkennung

(Lisa Berbig)

Im letzten Semester wurden bereits erste Ideen zur Kollisionserkennung entwickelt. Damals wurde noch mit mehreren gleichzeitig fliegenden Drohnen geplant, die sowohl vor einer Kollision miteinander, also auch einer Kollision mit den Rändern des Flugbereiches, also Wand oder Netz, geschützt werden müssen.

Es wurde geplant, die Position der Drohnen auf der horizontalen Ebene anhand der Farbe des leuchtenden Ballons, der an den Drohnen befestigt werden sollte, zu ermitteln und ihre Höhe auf der vertikalen Achse anhand des Barometers der Drohne zu bestimmen.

Durch das Anfang diesen Semesters erstellte Konzept eines Hindernis-Parcours war es nicht mehr nötig, eine Kollision von zwei Drohnen zu erkennen und zu verhindern, da nur eine Drohne gleichzeitig durch den Parcours fliegen sollte. Anstatt über die Farbe des Ballons wird die Position der Drohne nun durch Differenzbilder bestimmt, da nicht mehr zwischen mehreren Drohnen unterschieden werden muss.

2.1.5 Quadkopter

(Tobias Schwarz)

Bei den Zwischenpräsentationen im Wintersemester konnte ein „Poof of concept“ vorgezeigt werden. Dabei handelte es sich um ein flugfähiges Luftschiff, basierend auf einem Cheerson Quadcopter und mehreren Heliumballons. Die auseinander gebaute Drohne diente als Steuerung des Luftschiffs. Das Helium brachte dem Ganzen den benötigten Auftrieb. Dieser wurde so gewählt, dass das gesamte Konstrukt langsam von selbst zu Boden sinkt, damit das „hochfliegen“ möglichst wenig Leistung benötigt.

Im darauf folgenden Semester war vor allem die Steuerung via Laptop im Fokus, diese wird in Kapitel [2.3.6](#) näher erläutert.

2.2 Netzwerkverbindung

(Lisa Berbig)

Die Verbindung zwischen der Drohne und dem steuernden Computer sollte dieses Semester über ein eigenes Netzwerk erfolgen.

2.2.1 Probleme durch einen zu langsamen Router

Bei dem Router, einem Netgear WNR2000v3, welcher zunächst für Testzwecke verwendet wurde, traten immer wieder Probleme bei der Steuerung der Drohne auf. Die Drohne reagierte nicht auf die Steuerungsbefehle des Computers. Bei der Fehlersuche stellte sich heraus, dass es sich sehr wahrscheinlich um ein Problem mit der Geschwindigkeit des Routers handelt. Dieser ist mit 14 Jahren schon relativ alt und nicht besonders schnell, weshalb es relativ lange dauerte, bis das Steuersignal bei der Drohne ankam. Dies dauerte in manchen Fällen so lange, dass die Drohne wegen fehlendem Steuersignal wieder landete oder gar nicht erst startete.

Dieses Problem konnte schließlich durch einen anderen Router behoben werden. Wir konnten für das Projekt einen ASUS RT-AX55 von der Hochschule ausleihen. Nach erfolgreicher Einrichtung des Routers konnten die Drohnen in das Netzwerk eingebunden und auch zuverlässig darüber gesteuert werden.

2.2.2 Statische IP-Adressen

Im Code werden die Tello Drohnen (2.3.5) dafür über ihre IP-Adresse aufgerufen. Um diese IP-Adresse nicht nach jedem Einschalten der Drohne überprüfen zu müssen, wird den Drohnen im Router je eine statische IP-Adresse zugewiesen.

2.2.3 Mögliche Vorteile der 5 GHz Frequenz

Um eine noch stabilere Verbindung zwischen Computer und Drohne zu gewährleisten, wäre die Verwendung der 5 GHz Frequenz von Vorteil gewesen, welche auch vom ASUS-Router unterstützt wird. Gerade wenn viele Personen mit Endgeräten im Raum sind, wie beispielsweise bei der Präsentation, gibt es viele Geräte, die die

2,4 GHz Frequenz stören. Auf 5 GHz hätten wir das verhindern können, jedoch unterstützen die Tello Drohnen leider nur 2,4 GHz.

2.2.4 Vorteile der Drohnensteuerung über einen Router

Über dieses Netzwerk ist es nun also möglich, mehrere Drohnen auf einmal zu steuern. Für den Parcours, für dessen Umsetzung sich dieses Semester entschieden wurde, ist das zwar nicht unbedingt notwendig, bietet aber trotzdem Vorteile.

Einerseits müssen die Tello Drohnen nicht mehr ihr eigenes Netzwerk aufbauen, was sich zumindest etwas positiv auf die nicht besonders lange Akkulaufzeit der Drohnen auswirkt. Andererseits ist das Netzwerk des Routers auch stärker als das der Drohnen, was auch auf 2,4 GHz für eine etwas stabilere Verbindung sorgt.

2.2.5 Verwendung der Netzwerkverbindung für OSC

Außer für die Steuerung der Drohnen wurde das Netzwerk auch für die Handysteuerung (2.3.2) der Drohne genutzt. Diese übermittelt Daten des im Handy verbauten Gyroskops und sendet dies über OSC an den Computer, wo sie dann für die Drohnensteuerung verarbeitet werden. Um die OSC-Daten an den Computer zu übermitteln, wurde dasselbe Netzwerk, über welches auch die Drohnen gesteuert werden, verwendet.

Dies ermöglichte es, dem Computer, der die Gyroskopdaten empfängt, eine feste IP-Adresse zuzuweisen. Dadurch kann, egal von welchem Handy aus die Drohne gesteuert wird, immer dieselbe IP-Adresse angegeben werden, an welche die Daten dann mit OSC gesendet werden.

2.3 Softwarearchitektur

(Sina Steinmüller)

Die Softwarearchitektur von DRONEVERSE besteht aus mehreren modularen Komponenten, die es ermöglichen, aus drei verschiedenen Erkennungsmethoden und drei verschiedenen Steuerungsmethoden jeweils eine zu wählen. Die erste Erkennungsmethode ist die auf Mediapipe und mp_hands [1] basierende Gestenerkennung, bei der, je nach dem, wo und wie man seine Hand im Webcambild bewegt, eine Richtung ausgegeben wird (2.3.1). Weiterhin gibt es die Handyausrichtungs- oder OSC-Erkennung (2.3.2), bei der man mittels heruntergeladener iPhone App aus dem AppStore [5] auf das Gyroskop im iPhone zugreifen kann und mittels kippen des iPhones die Richtung erkennen kann (2.3.2). Zu guter letzt gibt es die Möglichkeit, die Drohne über die Tastatureingabeerkennung auf dem Laptop zu steuern (2.3.3). All diese Erkennungsmethoden wie auch die Kollisionserkennung (2.3.4) werden im Hauptmodul main.py verarbeitet und schließlich wird die erkannte Richtung, wenn sie zu keiner Kollision führt, an den jeweiligen Drohnencontroller gesendet und die Drohne fliegt in die gewünschte Richtung. Das Zusammenspiel aller Module wird in der folgenden Grafik deutlich.

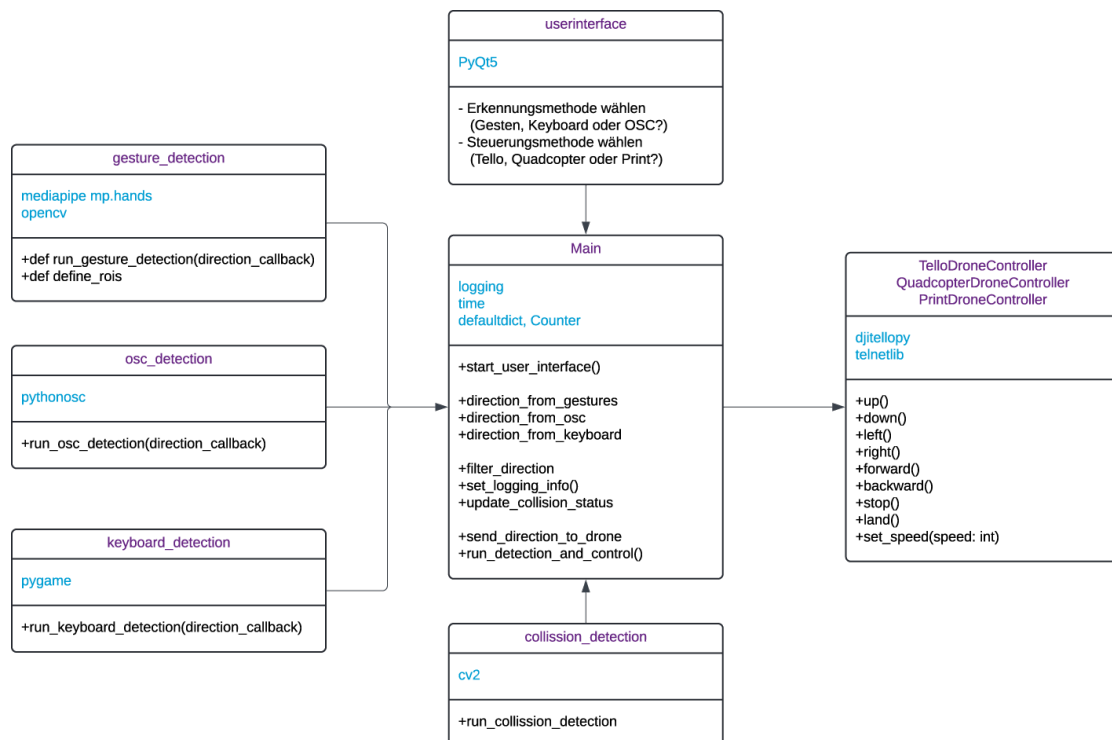


Abbildung 2.1: Softwarearchitektur Abbildung

Um für den Nutzer ein bestmögliches Spielerlebnis zu gewährleisten, wurden alle Auswahlmöglichkeiten in einem User Interface dargestellt, in dem der Spieler über Radiobuttons seine bevorzugte Erkennungs- und Steuerungsmethode auswählen kann.

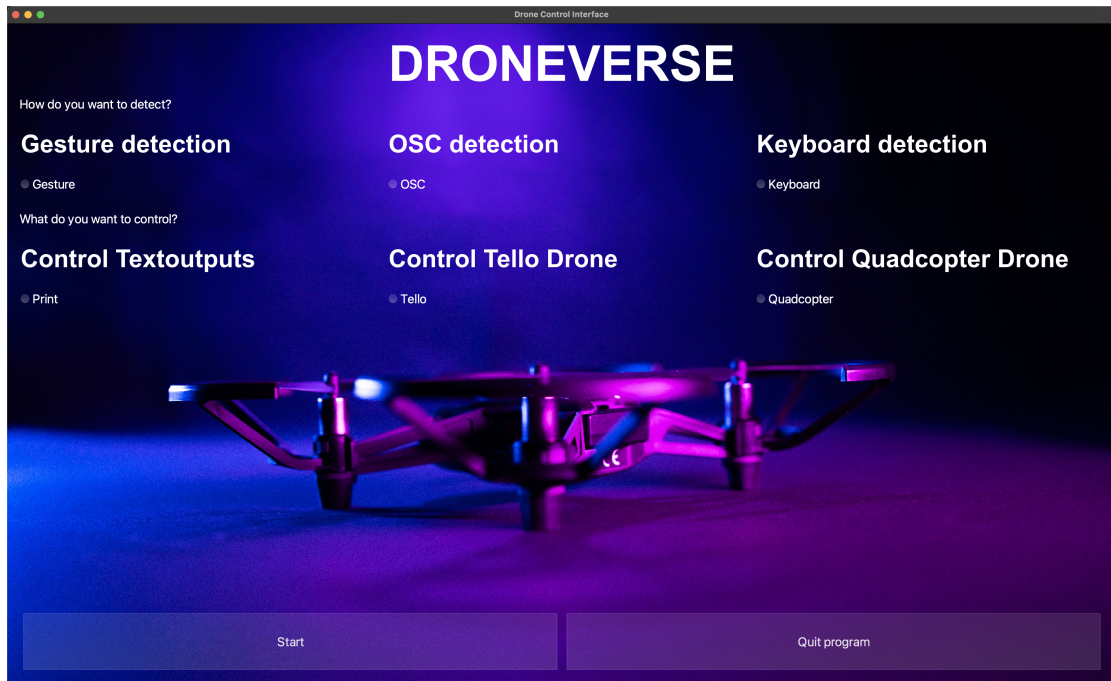


Abbildung 2.2: User Interface

Alle Erkennungsmodule haben den gleichen Aufbau, in dem der sogenannte Callback eine zentrale Rolle spielt. In den Modulen wird die Methode `run_xy_detection` implementiert (wobei xy für Gesten, OSC oder Tastatur steht) die eine Richtung über den `direction_callback` an die `main.py` übergibt. Ein Callback ist eine Funktion, die als Argument an eine andere Funktion übergeben wird, um später im Programmablauf aufgerufen zu werden.

Listing 2.1: Vereinfachter Aufbau aller Detection Module

```
1 def run_xy_detection(direction_callback):
2     # (...)
3     direction_callback(direction)
4
5 if __name__ == "__main__":
6     run_xy_detection(lambda direction: print(f"xy.py {
        direction}"))
```

In der `main.py` werden die zuvor erkannten Richtungen weiterverarbeitet. Zunächst wird ein Filter angewendet, um die Anzahl der Signale, die an die Drohne gesendet werden, zu begrenzen. Dies geschieht durch die *filter_direction-Funktion*, die aus einer Liste von erkannten Richtungen die am häufigsten vorkommende auswählt und diese dann weiterleitet. Die gefilterte Richtung wird anschließend über eine der spezifischen Callback-Funktionen, wie *direction_from_gestures*, *direction_from_osc* oder *direction_from_keyboard*, verarbeitet. Diese Funktionen rufen dann die zentrale Funktion *send_direction_to_drone* auf, die die Steuerbefehle an den gewählten Drohnencontroller übergibt.

In *send_direction_to_drone* werden die Steuerparameter der Drohne initialisiert, bevor überprüft wird, ob eine Kollision droht. Wenn keine Kollision erkannt wird und die Richtung sicher befliegar ist, wird die Drohne in die entsprechende Richtung gesteuert. Die Wahl der Erkennungs- und Steuerungsmethode erfolgt dynamisch zur Laufzeit und wird durch die jeweiligen Aufrufe in der `main.py` gesteuert.

Listing 2.2: Verkürzter Auszug aus der `main.py`

```
1  # (...)
2  from gesturedetection import run_gesture_detection
3  from oscdetection import run_osc_detection
4  from keyboarddetection import run_keyboard_detection
5  # (...)
6
7  def filter_direction(directions, max_size):
8      if len(directions) >= max_size:
9          counter = Counter(directions)
10         most_common_direction = counter.most_common(1)[0][0]
11         directions.clear()
12         return most_common_direction
13     return None
14
15 def direction_from_gestures(direction):
16     # (...)
17     send_direction_to_drone(filtered_direction)
18     # (...)
19
20 def direction_from_osc(direction):
21     # (...)
```

```

22
23 def direction_from_keyboard(direction):
24     # (...)
25
26 def send_direction_to_drone(filtered_direction):
27     global drone_controller
28     # (...)
29
30     if filtered_direction == "left" and not get_collison() ==
31         "left":
32         drone_controller.left()
33     # (...)
34
35 if __name__ == "__main__":
36     # (...)
37     if chosen_control == "tello":
38         drone_controller = TelloDroneController()
39         # (...)
40
41     if chosen_detection == "keyboard":
42         run_keyboard_detection(direction_from_keyboard)
43         # (...)

```

Bevor detaillierter auf die einzelnen Softwaremodule eingegangen wird, sei erwähnt, dass in allen Modulen eine Spielschleife implementiert wurde. Diese Schleife ermöglicht es, während der Ausführung des Programms zwischen verschiedenen Zuständen zu wechseln. Beispielsweise kann man durch Drücken der Taste Q zurück zum Auswahlmenü gelangen und die gewählte Erkennungsmethode oder Steuerung ändern. Um das Programm vollständig zu beenden, muss im Auswahlmenü die Schaltfläche „Quit Program“ ausgewählt werden.

2.3.1 Gestenerkennung

(Sina Steinmüller)

Das Python-Modul *gesture_detection.py* ist für die Erkennung von Handgesten in Echtzeit konzipiert und nutzt dafür die Bibliotheken OpenCV und Mediapipe durch *mp_hands = mp.solutions.hands[1]*. Anschließend wird Mediapipe mitgeteilt, dass es keine statischen Bilder verwenden soll sondern auf die Webcam zugreifen und nur eine Hand detektiert werden soll. Danach wird die Hauptfunktion des Moduls *run_gesture_detection* definiert, welche die tatsächliche Gestenerkennung startet und mittels des bereits erwähnten *direction_callback* die erkannte Richtung an das main-Modul übergibt.

Innerhalb dieser Hauptfunktion wird zunächst einmal das Webcambild abgerufen und ein Querformat Vollbildfenster geöffnet. Damit der Spieler nicht durch sein eigenes Webcambild irritiert wird, wird statt dem Kamerabild ein statisches Hintergrundbild angezeigt mit Informationen zur Steuerung und zu den zu verwendenden Gesten. Lediglich ein Handskelett wird angezeigt, damit der Nutzer sehen kann, wo im Bild die Hand erfasst wurde. Dieses Handskelett besteht aus 20 Gelenkpunkten, jeweils 4 Gelenkpunkte pro Finger. Für die geplante interaktive Drohnensteuerung sind 6 Gesten für 6 Richtungen nötig. Die Richtungen hierbei sind vorwärts, rückwärts, hoch, runter, rechts und links. Auf Drehungen im bzw. gegen den Uhrzeigersinn haben wir bewusst verzichtet, einerseits weil es schon schwierig genug war, 6 Gesten sicher voneinander unterscheiden zu können, andererseits weil sich der Nutzer so auch keine Gedanken zur Orientierung der Drohne im Raum machen muss.

Um mit dem Mediapipe *mp_hands* Modell Gesten erkennen zu können, muss man detektierte Gelenkpunkte in Abhängigkeit zu anderen Gelenkpunkten setzen. Die Herausforderung bei der Umsetzung der Handgestenerkennung liegt in der sicheren Unterscheidung zwischen sechs verschiedenen Gesten, was dadurch bedingt ist, dass jeder Mensch Abweichungen darin hat, wie die Gesten gemacht werden. Daher war es notwendig, mehrere Erkennungsmethoden zu kombinieren, um eine zuverlässige und präzise Erkennung zu gewährleisten. Diese Kombination von Methoden stellt sicher, dass die Gestenerkennung unter verschiedenen Bedingungen konsistent und genau bleibt.

2.3.1.1 Erkennung von oben und unten durch Regionen

Die Erkennung der Richtungen „Oben“ und „Unten“ wird durch die Definition spezifischer Bereiche (Regions of Interest, ROIs) auf dem Bildschirm realisiert. Hierbei werden das obere und untere Viertel des Bildes als entscheidende Zonen festgelegt. Diese Zonen sind durch weiße Linien in den Abbildungen (2.3, 2.4) gekennzeichnet. Wenn die Spitze des Zeigefingers in eine dieser Zonen eintritt, wird die entsprechende Richtung erkannt, unabhängig von der Ausrichtung des Fingers.

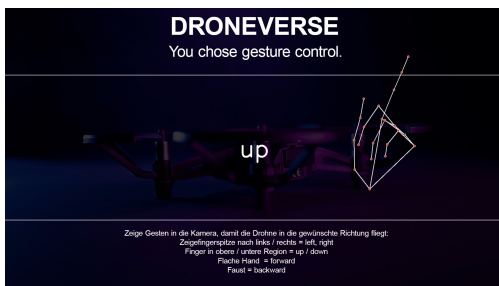


Abbildung 2.3: Geste Hoch

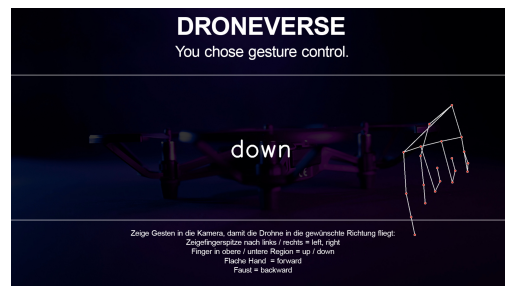


Abbildung 2.4: Geste Runter

Die Bestimmung der Position des Fingers erfolgt durch die y-Koordinate des Gelenkpunktes an der Spitze des Zeigefingers. Befindet sich dieser Punkt oberhalb der oberen weißen Linie, also in der Region „roi_top“, wird die Bewegung nach „Oben“ erkannt. Umgekehrt wird eine Bewegung nach „Unten“ erkannt, wenn der Punkt unterhalb der unteren weißen Linie liegt. Diese einfache Regel sorgt dafür, dass „Oben“ und „Unten“ mit höchster Priorität erkannt werden.

Listing 2.3: Erkennung von oben und unten mittels Regionen

```
1 # Finger oben
2 if finger_y < roi_top:
3     direction = "up"
4
5 # Finger unten
6 elif finger_y > roi_bottom:
7     direction = "down"
```

2.3.1.2 Vorwärts- und Rückwärtserkennung durch Distanzberechnungen

Die Erkennung der Richtungen „Vorwärts“ und „Rückwärts“ basiert auf spezifischen Handgesten, die durch die relative Position und den Abstand zwischen Daumen- und Zeigefingerspitze bestimmt werden. Diese Gesten haben nach der Erkennung von „Oben“ und „Unten“ die nächsthöhere Priorität.

Für die Richtung „Vorwärts“ wird eine Geste verwendet, bei der die Hand flach nach vorne ausgestreckt wird, mit gespreizten Fingern (auch bekannt als „Stopp“-Geste, 2.5). Diese Bewegung symbolisiert, dass sich die Drohne vom Benutzer wegbewegen, also vorwärts fliegen, soll. Um die Richtung „Rückwärts“ zu signalisieren, wird die Hand zur Faust geballt und direkt neben dem Kopf positioniert, was bedeutet, dass die Drohne zum Benutzer zurückkehren soll (2.6)

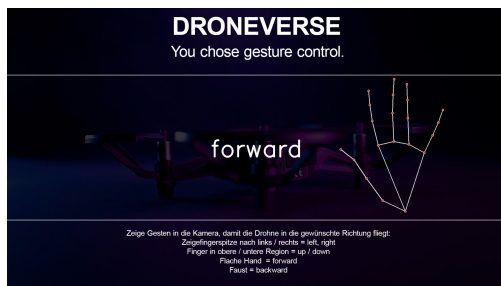


Abbildung 2.5: Geste Vorwärts

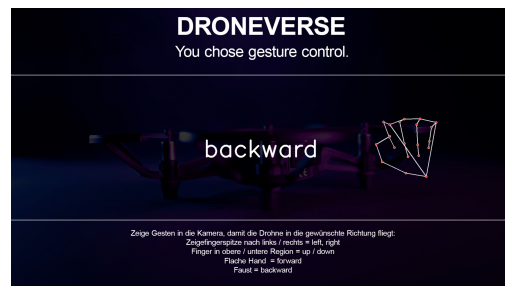


Abbildung 2.6: Geste Rückwärts

Die technische Umsetzung erfolgt durch die Messung der Distanz zwischen der Daumenspitze und der Spitze des Zeigefingers. Ein Schwellwert für diesen Abstand wird verwendet, um zwischen den beiden Bewegungsrichtungen zu unterscheiden. Wenn der gemessene Abstand unterhalb eines definierten Werts (*threshold_backward*) liegt, wird „Rückwärts“ erkannt und wenn der Abstand hingegen größer als ein anderer definierter Schwellenwert (*threshold_forward*) ist, wird „Vorwärts“ erkannt. Um die Unterscheidung optimal zu gewährleisten war es auch nötig, die Hand bei „Vorwärts“ näher zum Bildschirm hin zu bewegen, um die Distanz zwischen Zeigefingerspitze und Daumen zu vergrößern und für „Rückwärts“ weiter vom Bildschirm weg zu halten, um besagte Distanz zu verringern.

Listing 2.4: Erkennung von vorwärts und Rückwärts durch Distanzmessung

```
1 else:
2     thumb_tip = hand_landmarks.landmark[mp_hands.
3         HandLandmark.THUMB_TIP]
```

```

4      # Schwellwerte definieren
5      threshold_backward = 0.06
6      threshold_forward = 0.2
7
8      # Berechne die Distanz zwischen Daumen und Zeigefinger
9      distance = math.sqrt(
10         (index_finger_tip.x - thumb_tip.x) ** 2 + (
11             index_finger_tip.y - thumb_tip.y) ** 2)
12
13     if distance < threshold_backward:
14         direction = "backward"
15     elif distance > threshold_forward:
16         direction = "forward"

```

2.3.1.3 Erkennung von rechts und links

Wenn die Zeigefingerspitze weder in den für „Oben“ und „Unten“ definierten Regionen liegt, noch die Distanz zwischen Daumen- und Zeigefingerspitze außerhalb der festgelegten Schwellenwerte für „Vorwärts“ und „Rückwärts“ ist, bleibt als letzte Möglichkeit die Erkennung der horizontalen Richtungen „Rechts“ und „Links“.

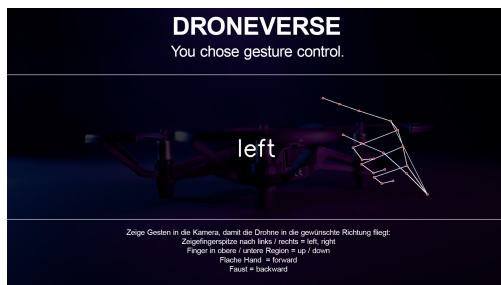


Abbildung 2.7: Geste Links

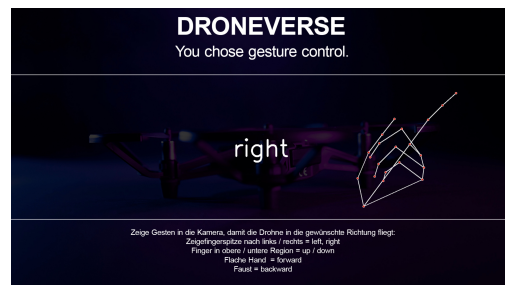


Abbildung 2.8: Geste Rechts

In diesem Fall wird die horizontale Position (X-Koordinate) der Zeigefingerspitze mit der Daumenspitze verglichen. Wenn die Zeigefingerspitze links von der Daumenspitze liegt (2.7), wird die Richtung „Links“ erkannt. Befindet sich die Zeigefingerspitze hingegen rechts von der Daumenspitze (2.8), wird die Richtung „Rechts“ ausgegeben. Diese Logik stellt sicher, dass die Bewegungen nach links und rechts zuverlässig erkannt werden, wenn keine andere vorrangige Bewegung detektiert wurde.

Listing 2.5: Erkennung von rechts und links durch Zeigefingerspitzenposition

```
1 else:
2     if index_finger_tip.x < thumb_tip.x:
3         direction = "left"
4     elif index_finger_tip.x > thumb_tip.x:
5         direction = "right"
6     else:
7         direction = "stop"
```

Wenn keine Richtung erkannt wird, gibt das System die Richtung „Stopp“ aus, was der Drohne signalisiert, dass es keine Richtungsänderung gab. Die Drohne verharret daraufhin in der Luft und bewegt sich nicht weiter. Alle diese Richtungserkennungen werden über die Callback-Funktion an die Hauptanwendung weitergeleitet. Dies geschieht etwa 940 Mal pro Minute, also zwischen 14 und 19 Mal pro Sekunde. Dabei spielt es keine Rolle, welche Geste erkannt wird oder wie oft sich eine Erkennung ändert. Um die Drohne vor einer Überlastung durch zu viele Befehle zu schützen, werden die entsprechenden Gestenerkennungen mit einem Filterfaktor von 15 verarbeitet. Das bedeutet, dass von den 15 Erkennungen pro Sekunde die am häufigsten erkannte Richtung an die Drohne gesendet wird.

Schließlich wird die erkannte Richtung ungefiltert als Text auf dem Bildschirm angezeigt, damit der Spieler eine Rückmeldung erhält und seine Gesten entsprechend anpassen kann, um die Drohne in die gewünschte Richtung zu lenken. Möchte man zurück ins Hauptmenü und eine andere Art der Steuerung wählen, so sorgt das Drücken der Taste „q“ für eine Abbruchbedingung, bei der die Gestenerkennungsschleife unterbrochen wird und man zurück ins Auswahlmenü gelangt.

2.3.1.4 Praxiserfahrung

Bei der Abschlusspräsentation zeigte sich, dass die Gestenerkennung nicht nur wie geplant funktionierte, sondern auch für viel Spaß sorgte. Von den etwa zehn Personen, die die Drohne durch den Hindernisparcours steuerten, konnten alle mindestens ein Hindernis erfolgreich überwinden. Für einige war die Steuerung so intuitiv, dass sie den gesamten Parcours schnell meisterten. Andere hingegen hatten Schwierigkeiten, da sie instinktiv den Daumen statt des Zeigefingers verwenden wollten, was auch nach wiederholten Hinweisen nicht besser wurde. Die Erkennung von Rechts, Links, Oben und Unten funktionierte jedoch einwandfrei. Lediglich der

Schwellwert für Vorwärts und Rückwärts bereitete gelegentlich Probleme, abhängig von der Handgröße der jeweiligen Person und dem Abstand zum Bildschirm. Hier könnte in Zukunft eine Funktion implementiert werden, die die Handgröße der Nutzer erfasst und die Schwellwerte dynamisch anpasst. Nach ein paar Minuten Übung konnten die entsprechenden Gesten jedoch korrekt ausgeführt werden, weshalb diese Anpassung nicht zwingend erforderlich wäre. Ein weiterer Punkt, der auffiel, war, dass die Geste für „Rechts“, also mit dem rechten Zeigefinger nach rechts zu zeigen, auf Dauer sehr anstrengend wurde. Diese Geste könnte überarbeitet werden, wenn man die Drohne häufiger gestengesteuert fliegen möchte. Insgesamt war die Gestenerkennung die bevorzugte Steuerungsmethode, die allen Anwendern viel Spaß bereitete und das Publikum zum Mitfiebern und Helfen anregte.

2.3.2 Handyausrichtungserkennung (OSC)

(Sina Steinmüller)

Für die Steuerung der Drohne mittels eines iPhones wurde die App „Data OSC“ aus dem AppStore [5] verwendet. Diese App ermöglicht es, die Gyroskopdaten des Handys in Echtzeit an den Computer zu übertragen und je nach Bedarf zu verarbeiten. Damit dies funktioniert, müssen sich sowohl das Smartphone als auch der Computer im selben Netzwerk 2.2 befinden und der korrekte Netzwerkport angegeben werden, auf dem der Computer die Daten empfängt. Um dies zu ermöglichen, wird ein Argument-Parser in Python konfiguriert, der zwei optionale Befehlszeilenargumente definiert: `-ip` für die IP-Adresse des Computers und `-port` für den Port, auf dem der OSC-Server auf die Daten hört.

Listing 2.6: Definition der IP Adresse und des Ports

```
1 parser.add_argument(  
2     "--ip",  
3     default="192.168.50.152", # Lisas Router in der TH  
4 )  
5 parser.add_argument(  
6     "--port",  
7     type=int,  
8     default=5005,  
9 )  
10 args = parser.parse_args()
```

2.3.2.1 Erstmaliges Empfangen von OSC Nachrichten

Anschließend wird auf der definierten IP-Adresse und dem dazugehörigen Port der OSC Server initialisiert und die Verarbeitung von OSC-Daten gestartet. Eine Herausforderung hierbei war es herauszufinden, wie der Pfad der gesendeten Daten konkret aussieht, da weder die iPhone App selbst, noch die zugehörige Dokumentation [6] hier Aufschluss boten. Die Abbildung 2.9 zeigt Screenshots der iPhone App.

Anhand des rechten Screenshots 2.9 wird deutlich, dass man durchaus meinen könnte, der Datenpfad lautet lediglich „gyroscope / x“, was jedoch nicht funktionierte.

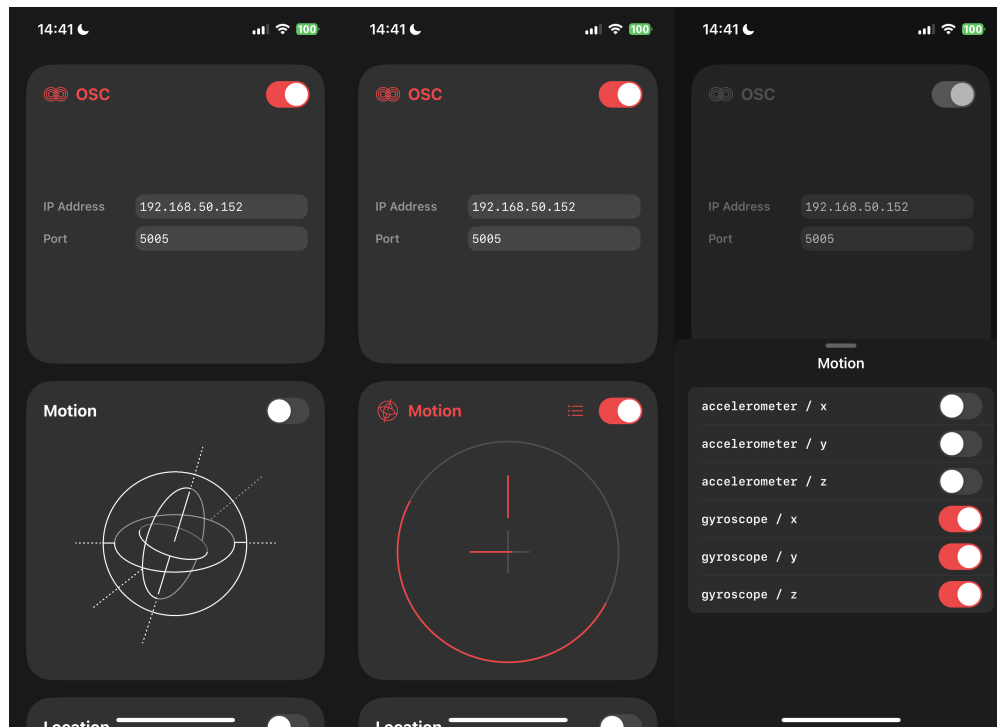


Abbildung 2.9: Screenshots der DataOSC iPhone App

Um den entsprechenden Datenpfad herauszufinden war es nötig, ein Hilfsskript zu schreiben, das alle Daten der entsprechenden IP-Adresse auf dem Port anzeigt.

Listing 2.7: Hilfsskript zum Herausfinden des OSC Datenpfades

```

1 from pythonosc import dispatcher
2 from pythonosc import osc_server
3
4 def print_message(address, *args):
5     print(f"Received message: {address} {args}")
6
7 def main():
8     disp = dispatcher.Dispatcher()
9     disp.set_default_handler(print_message)
10
11     server = osc_server.ThreadingOSCUDPServer(("
12         192.168.178.44", 5005), disp)
13     print("Serving on port 5005")
14
15     server.serve_forever()

```



```

16 if __name__ == "__main__":
17     main()

```

Das Hilfsskript brachte Klarheit, in dem es den Eingang von Daten mit dem Pfad anzeigte, was in Abbildung 2.10 ersichtlich wird. Damit konnten die OSC Daten zwar empfangen werden, für die Verarbeitung und Interpretation jedoch waren noch einige Formatierungen nötig, um herauszufinden, welches Dreh- und Kippverhalten des iPhones welche Änderung der Daten bewirkt.

```

Received message: /data/motion/gyroscope/z (-0.29093995690345764,)
Received message: /data/motion/gyroscope/x (0.47478941082954407,)
Received message: /data/motion/gyroscope/y (-0.010446249507367611,)
Received message: /data/motion/gyroscope/z (-0.20994897186756134,)
Received message: /data/motion/gyroscope/x (0.48533979058265686,)
Received message: /data/motion/gyroscope/y (-0.3719154894351959,)
Received message: /data/motion/gyroscope/z (-0.1077624186873436,)
Received message: /data/motion/gyroscope/x (0.4185570478439331,)
Received message: /data/motion/gyroscope/y (-0.5682477355003357,)

```

Abbildung 2.10: OSC Nachrichten

Listing 2.8: Starten des OSC Servers

```

1 osc_dispatcher = osc_dispatcher_module.Dispatcher()
2 osc_dispatcher.map("/data/motion/gyroscope/x",
   gyro_handler_x)
3 osc_dispatcher.map("/data/motion/gyroscope/y",
   gyro_handler_y)
4 osc_dispatcher.map("/data/motion/gyroscope/z",
   gyro_handler_z)

```

2.3.2.2 Empfangen und Verarbeiten der OSC Werte

Nach dem Herausfinden des richtigen Datenpfades wurde dieser auch gleich in das Python-Skript *oscdetection.py* integriert, welches dafür sorgt, dass diese Gyroskopdaten vom OSC-Server empfangen werden und die Bewegungsrichtung des Smartphones bestimmt werden kann. Der obige Code zeigt die Initialisierung des OSC Servers.

Das Skript liest kontinuierlich die Gyroskopdaten in den drei Achsen (X, Y, Z) und speichert diese in einer Datenhistorie von jeweils 500 Werten. Um die Bewegungsrichtung des Smartphones zu berechnen, werden die Durchschnittswerte der

letzten 500 Gyroskopdaten herangezogen und auf Basis dieser Durchschnittswerte dann die Richtung der Bewegung bestimmt.

In Python ist das ganze folgendermaßen umgesetzt.

Listing 2.9: Sammeln von Gyroskopdaten

```
1 gyro_values = list(zip(gyro_x_values, gyro_y_values,
    gyro_z_values))
```

Die Gyroskopdaten der letzten 500 Bewegungen werden als Tupel in der Liste *gyro_values* gespeichert. Anschließend werden die Summen der Werte jeder Achse (X, Y und Z) berechnet und durch die *HISTORY_SIZE*, also die Anzahl der gesammelten Werte, geteilt, um den Durchschnitt zu erhalten.

Listing 2.10: Ermittlung der Durchschnittswerte

```
1 sum_x, sum_y, sum_z = 0, 0, 0 #2
2 for x, y, z in gyro_values: #3
3     sum_x += x
4     sum_y += y
5     sum_z += z
6 avg_x = sum_x / HISTORY_SIZE #4
7 avg_y = sum_y / HISTORY_SIZE #4
8 avg_z = sum_z / HISTORY_SIZE #4
9 max_value = max(abs(avg_x), abs(avg_y), abs(avg_z))
```

Schließlich findet aufgrund der vorausgehenden Berechnung die Richtungsbestimmung statt: Hierfür prüft die Funktion, welche der drei Achsen (X, Y oder Z) den größten Durchschnittswert hat und vergleicht diesen mit dem Schwellenwert *THRESHOLD*. Durch einen Vergleich der Werte wird nun die gewünschte Richtung ausgegeben.

Listing 2.11: Richtungsbestimmung mittels Schwellwert

```
1
2 if max_value < THRESHOLD:
3     return "stop"
4 if max_value == abs(avg_x):
5     if avg_x > 0:
6         return "right"
7     else:
```

```

8         return "left"
9 elif max_value == abs(avg_y):
10     if avg_y > 0:
11         return "down"
12     else:
13         return "up"
14 elif max_value == abs(avg_z):
15     if avg_z > 0:
16         return "forward"
17     else:
18         return "backward"

```

Um die Richtung auch für den Spieler schön darzustellen, wurde mittels PyGame [4] eine Benutzeroberfläche erstellt, bei der Informationen zur Haltung des iPhones mitgeteilt werden, wie auch die detektierte Richtung dargestellt wird.

Die Richtungsdetektion basiert auf der Ausrichtung und Bewegung des Smartphones. Hierbei muss das Display nach oben zeigen, die Kamera auf der linken und der Homebutton bzw. die Ladebuchsen- und Kopfhöreranschlüsse auf der rechten Seite sein. Verharret das iPhone nun in dieser Haltung, wird als Richtung zunächst „Stopp“ ausgegeben, bis ein gewisser Schwellwert überschritten ist.

2.3.2.3 Nutzung des iPhones zur Richtungserkennung

Die beiden Richtungen vorwärts und rückwärts werden durch das Kippen des Smartphones um die Z-Achse erkannt, wie die Abbildungen 2.11 und 2.12 zeigen.

Eine Drehung um 45° nach rechts oder links führt zu einer Vorwärts- oder Rückwärtsbewegung, wie in den Abbildungen 2.13 und 2.14 gezeigt ist.

Durch das Kippen des Smartphones um die Y-Achse erkennt das Skript Bewegungen nach links oder rechts, wie man in den Abbildungen 2.16 und 2.15 sieht.

Auch hier wurde wie bei der Gestenerkennung (vgl. 2.3.1) eine Spielschleife implementiert, aus der man mit Drücken der Taste Q heraus und zum Hauptmenü zurückkehren kann, um die nächste Steuerungsmethode auszuprobieren.



Abbildung 2.11: OSC Hoch

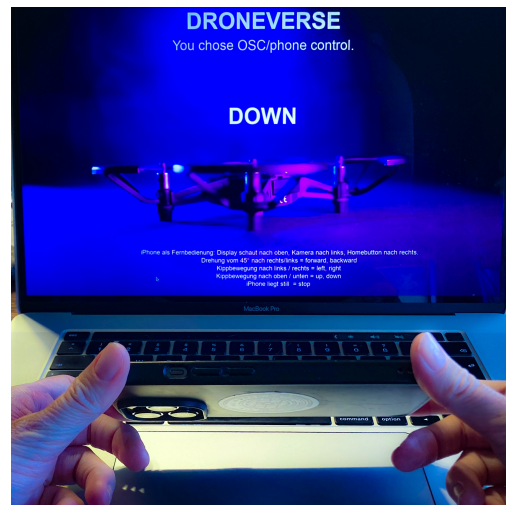


Abbildung 2.12: OSC Runter



Abbildung 2.13: OSC Vorwärts

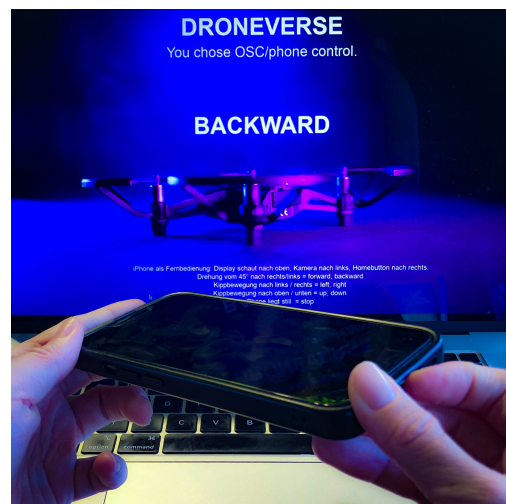


Abbildung 2.14: OSC Rückwärts

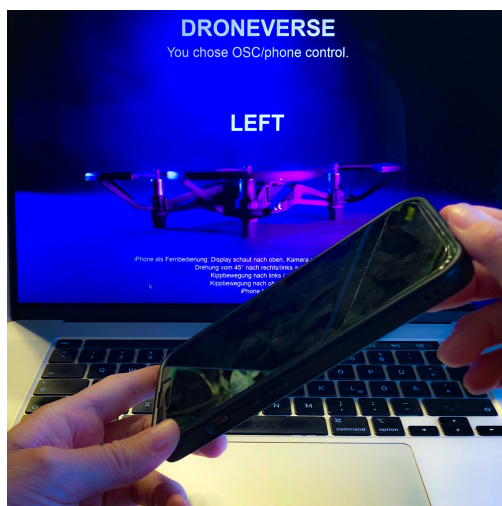


Abbildung 2.15: OSC Links



Abbildung 2.16: OSC Rechts

2.3.3 Keyboard Erkennung

(Maximilian Richter)

Neben der Verwendung von Gesten (2.3.1) und OSC-Signalen (2.3.2) um die Drohne zu steuern, galt es auch eine Steuerung über die Tastatur zu realisieren.

Aufgrund der häufigen Verwendung von „W, A, S, D“ als Steuerungsmethode bei Videospielen, wurde auch bei der Drohnensteuerung von DRONEVERSE darauf zurückgegriffen. Es ergab sich die folgende Tastaturbelegung:

- W: Richtung - Forward
- A: Richtung - Left
- S: Richtung - Backward
- D: Richtung - Right
- Q: Abbruch der Erkennungssoftware
- Pfeiltaste Hoch: Richtung - Up
- Pfeiltaste Runter: Richtung - Down

Weil schon im vorherigen Semester eine unabhängig von anderen Modulen funktionierende Tastatursteuerung umgesetzt wurde, bei der die Bibliothek „PyGame“ [**PyGame**] als Basis diente, entschieden wir uns auch bei der Implementierung in die Softwarearchitektur (2.3) für dieses Open-Source Modul.

Im Vergleich zum vorherigen Semester galt es also, der Steuerungseinheit eine Richtung zu übergeben, anstatt direkt einen Steuerungsbefehl auszuführen.

Hierfür wurde die Methode *run_keyboard_detection(direction_callback)* ins Leben gerufen.

2.3.3.1 Initialisierung

Zunächst wird in dieser Funktion PyGame initialisiert (1) und verschiedene Anpassungen am Design des Spielfensters werden vorgenommen.

Beispielsweise wird der einheitliche DRONEVERSE Hintergrund geladen und skaliert (2).

Es werden die Parameter „running“ und „direction“ initialisiert. Running wird dem Booleschen Wert „true“ (3) zugeordnet, dieser dient als Bedingung für die while-Schleife (Game-Loop). Die Direction wird zu Beginn auf „none“ gesetzt. Außerdem wird das Dictionary „keys_pressed“ definiert. Die Schlüssel repräsentieren die verschiedenen Richtungen und jedem Schlüssel kann der Wert „true“ und „false“ zugeordnet werden.

Vorerst wird der Wert jeder Richtung jedoch auf false gesetzt.

Listing 2.12: Initialisierung Tastaturerkennung

```
1         PyGame.init()           #1
2     screen = PyGame.display.set_mode((1792, 1120))
3     PyGame.display.set_caption("Keyboard Detection")
4     clock = PyGame.time.Clock()
5     background_image = PyGame.image.load("images/DSC01497.
6         jpg")                   #2
7     background_image = PyGame.transform.scale(
8         background_image, (1792, 1120))    #2
9     font_large = PyGame.font.SysFont("Arial", 80, bold=True)
10    font_small = PyGame.font.SysFont("Arial", 48)
11    font_textblock = PyGame.font.SysFont("Arial", 24)
12
13    running = True                #3
14    direction = "none"           #4
15    keys_pressed = {             #5
16        "forward": False,
17        "backward": False,
18        "left": False,
19        "right": False,
20        "up": False,
21        "down": False
22    }
```

2.3.3.2 Handling der KeyDown-Events

Um bei Tastendruck eine Richtung an die Drohne zu übergeben, werden die KeyDown-Events der PyGame Bibliothek verwendet. In der folgenden while-Schleife wird bei KeyDown-Events (Taste wurde gedrückt) die entsprechende Richtungsvariable auf true gesetzt(1), danach wird diese Richtungsvariable als Direction festgelegt(2). Beim Aufruf des „PyGame.QUIT-Events“ (Standardmäßig Taste „Q“) wird die Tastaturerkennung beendet(3). Zum Schluss findet ein Direction-Callback statt, der die Richtung der TelloDroneControl Klasse übergibt(4).

Listing 2.13: Initialisierung Tastaturerkennung

```
1
2     while running:
3         for event in PyGame.event.get():
4             if event.type == PyGame.QUIT:                #3
5                 PyGame.quit()
6                 sys.exit()
7             if event.type == PyGame.KEYDOWN:
8                 if event.key == PyGame.K_w:
9                     keys_pressed["forward"] = True        #1
10                    direction = "forward"                  #2
11                elif event.key == PyGame.K_s:
12                    keys_pressed["backward"] = True        #1
13                    direction = "backward"                  #2
14                    direction_callback(direction)           #4
```


2.3.3.3 Handling der KeyUp-Events

Damit die Drohne nicht endlos weiterfliegt muss in der Tastaturerkennung die stop-Methode der TelloDroneControl (2.3.5) Klasse aufgerufen werden. Hierfür wurden die KeyUp-Events (Taste losgelassen) die logischerweise zwangsläufig nach dem Drücken einer Taste auftreten, verwendet.

Sobald eine Taste losgelassen wurde, wird die entsprechende Richtungsvariable auf „false“ gesetzt (1) und die direction auf „stop“ (2).

Am Ende findet wieder der Direction- Callback statt, der die Richtung an den Drone-Controller weitergibt (3).

Listing 2.14: Initialisierung Tastaturerkennung

```
1         if event.type == PyGame.KEYUP:
2             if event.key == PyGame.K_w:
3                 keys_pressed["forward"] = False           #1
4                 direction = "stop"                        #2
5             elif event.key == PyGame.K_s:
6                 keys_pressed["backward"] = False          #1
7                 direction = "stop"                        #2
8             direction_callback(direction)                  #3
```

2.3.4 Kollisionserkennung

(Lisa Berbig)

2.3.4.1 Notwendigkeit einer Kollisionserkennung

Da die Steuerung einer Drohne über Gesten (2.3.1) für die meisten Menschen neu und ungewohnt ist, wird die Drohne anfangs noch nicht ganz sicher gesteuert. Dadurch kann es leicht passieren, dass die Drohne nicht genauso fliegt wie gewollt und aus dem Flugbereich hinausfliegt. Um zu verhindern, dass die Drohne deshalb außerhalb der Flugzone Schaden anrichten kann, wurde eine Kollisionserkennung in das Projekt integriert. Diese sorgt dafür, dass die Drohne nicht aus dem Bereich des Parcours hinausfliegen kann. Somit soll in unserem Fall eine Kollision mit der Wand oder dem Netz, das die Zuschauer zusätzlich vor der Drohne schützt, verhindert werden.

2.3.4.2 Differenzbild zur Positionsbestimmung der Drohne

Für die ursprüngliche Idee, die Position der Drohne anhand der Farbe ihres Ballons zu erkennen, konnte dieses Semester eine bessere Lösung gefunden werden. Da es mit nur einer Drohne gleichzeitig im Parcours nicht mehr notwendig ist, zwischen mehreren Drohnen unterscheiden zu können, stellte sich die Verwendung von Differenzbildern als eine bessere Lösung heraus.

Die Kollisionserkennung ist, wie auch der Rest des Projekts in Python implementiert und verwendet OpenCV als Bibliothek zur Bildverarbeitung.

Der Parcour wird mit einer Kamera von unten gefilmt und der Videostream in das Python-Programm eingelesen. In einer While-Schleife werden die Frames nacheinander in Graustufen umgewandelt, mit dem vorherigen Frame verglichen, daraus ein Differenzbild erzeugt und dieses schließlich in ein Binärbild umgewandelt. Dadurch entsteht ein schwarzes Bild mit weißen Pixeln an den Stellen, an denen sich etwas im Bild bewegt. Da der Parcour statisch ist und sich nur die Drohne darin bewegt, zeigen die weißen Pixel die Position der Drohne an.

Listing 2.15: Berechnung eines binären Differenzbildes

```
1 gray = cv2.cvtColor(cropped_frame, cv2.COLOR_BGR2GRAY)
2 if old_frame is not None:
3     diff = cv2.absdiff(old_frame, gray)
4     _, bin_diff = cv2.threshold(diff, 100, 255, cv2.
        THRESH_BINARY)
5     cv2.imshow("diff_frame", bin_diff)
```

2.3.4.3 Erkennung der Drohne in den Randbereichen

Anhand der weißen Pixel im Bild kann also nun erkannt werden, wo sich die Drohne befindet. Durch einen Margin kann die Breite des Randbereiches festgelegt werden. Wenn sich nun weiße Pixel in diesem Bereich befinden, wird dies erkannt. Dabei kann auch unterschieden werden, ob sich die Drohne oben, unten, rechts oder links im Bild in einem Gefahrenbereich befindet.

Listing 2.16: Erkennung von weißen Pixeln in den Randbereichen

```
1 top_margin = bin_diff[:margin_size, :]
2 bottom_margin = bin_diff[-margin_size:, :]
3 left_margin = bin_diff[:, :margin_size]
4 right_margin = bin_diff[:, -margin_size:]
5
6 top_has_white = np.any(top_margin == 255)
7 bottom_has_white = np.any(bottom_margin == 255)
8 left_has_white = np.any(left_margin == 255)
9 right_has_white = np.any(right_margin == 255)
```

2.3.4.4 Eingreifen in die Drohnensteuerung

Um nun auch verhindern zu können, dass die Drohne dann auch nicht weiter in die Richtung fliegt, muss die Kollisionserkennung in die Drohnensteuerung eingreifen. Dafür sollte eine Variable an die Drohnensteuerung übergeben werden, die über die Gefahrensituation informiert. Diese Variable sollte mitteilen, wo sich die Drohne in einem Randbereich befindet und die Bewegung der Drohne in diese Richtung blockieren.

2.3.4.5 Problem bei der Integration in der Drohnensteuerung

Es stellte sich jedoch als Problem heraus, dass die Gestenerkennung (2.3.1) und die Kollisionserkennung nicht gleichzeitig auf einem Computer laufen können. Denn beide greifen auf Kameras zu – die Gestenerkennung auf die Webcam und die Kollisionserkennung auf einen über USB angeschlossene Kamera. Es konnte jedoch nicht auf beide Kameras gleichzeitig zugegriffen werden. Die Kollisionserkennung wurde erst dann aktiv, wenn die Gestenerkennung beendet wurde.

2.3.4.6 Lösungsansätze zur Integration in die Drohnensteuerung

Um dieses Problem zu lösen, wurden verschiedene Ansätze verfolgt. Der erste Versuch bestand darin, mithilfe von Threading, Kollisionserkennung und Gestenerkennung parallel laufen zu lassen. Dies funktionierte jedoch aufgrund des Global Interpreter Locks von Python nicht [8]. Dieser verhindert, dass in Python mehrere Threads gleichzeitig ausgeführt werden. Ein weiterer Versuch, das Problem mithilfe von Multiprocessing zu lösen, schlug ebenfalls fehl. Das Betriebssystem scheint die Verwendung von zwei Kameras zu blockieren.

Schließlich wurde sich dafür entschieden, Kollisionserkennung und Drohnensteuerung auf zwei unterschiedlichen Computern laufen zu lassen und die Gefahrenmeldungen der Kollisionserkennung mit OSC über unser Netzwerk an den Computer mit der Drohnensteuerung zu übermitteln.

Hierfür wurden mit `sendCollision.py` und `receiveCollision.py` ein OSC-Client und OSC-Server implementiert, welche die Kollisionsinformationen übermitteln sollen. Befindet sich die Drohne nun zum Beispiel auf der linken Seite des Bildes im Gefahrenbereich, wird `runSendCollision(„left“)` aus `sendCollision.py` aufgerufen, welches nun eine Warnung an den Computer der Drohnensteuerung schickt, mit der Information wo der Gefahrenbereich ist, also in diesem Fall links.

Listing 2.17: Aufruf von `sendCollision.py` in `CollisionDetection.py`

```
1 if top_has_white:
2     print("Gefahrbereich vorne")
3     sendCollision.run_send_collision('forward')
4
5 elif bottom_has_white:
6     print("Gefahrbereich hinten")
```

```

7     sendCollision.run_send_collision('backward')
8
9 elif left_has_white:
10     print("Gefahrbereich links")
11     sendCollision.run_send_collision('left')
12
13 elif right_has_white:
14     print("Gefahrbereich rechts")
15     sendCollision.run_send_collision('right')

```

2.3.4.7 Verwendung einer 360° Kamera als 180° Kamera

Die Kamera für die Kollisionserkennung muss den gesamten Bereich des Parcours aufnehmen können. Um etwas mehr Abstand zwischen der Kamera und dem Parcours zu erzeugen und somit möglichst viel vom Parcours ins Bild zu bekommen, filmt die Kamera diesen von unten. Trotzdem reichte die Brennweite der Handykamera, welche anfangs dafür verwendet wurde, nicht aus.

Um ein weitwinkligeres Bild zu erhalten, das den kompletten Parcours abbildet, wurden Weitwinkellinsen für das Handy verwendet. Damit war schon ein größerer Teil des Parcours im Bild, die Brennweite war jedoch immer noch nicht weitwinklig genug.

Dieses Problem konnte schließlich durch eine 360° Kamera gelöst werden. In diesem Projekt wurde die Ricoh Theta V verwendet. Diese kann per USB mit dem Computer verbunden werden. Wichtig ist hierbei, dass auch ein Kabel verwendet wird, das Daten übertragen kann. Das mitgelieferte Kabel der Kamera konnte das nicht und sorgte deshalb anfangs für Probleme.

Das Bild der 360° Kamera besteht aus einem quadratischen Bild der einen Linse, an dessen Rand rechts und links jeweils die Hälfte des Bildes der anderen Linse angehängt werden (vgl. 2.17). Für die Kollisionserkennung soll die 360° Kamera als 180° Kamera verwendet werden. Dafür genügt es, nur das Bild Linse zu verwenden, die das ganze quadratische Bild aufnimmt, zu verwenden. Die Kamera wurde also mit dieser Linse nach oben zeigend auf den Boden in die Mitte des Parcours gelegt und mit dem Computer verbunden.

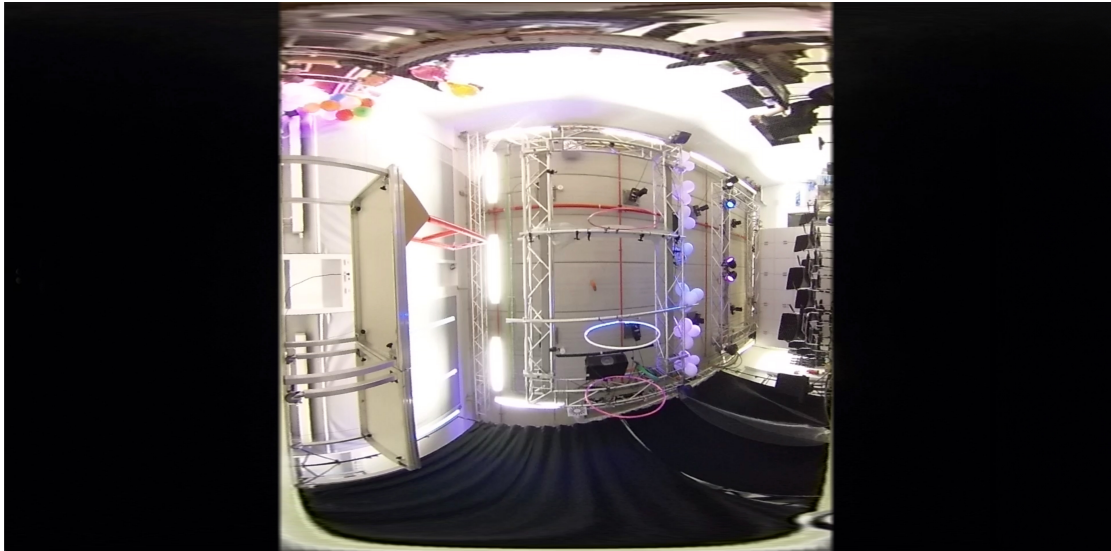


Abbildung 2.17: Bild der 360° Kamera

Im Programmcode wird nun das 360° Bild an den Seiten zugeschnitten, sodass nur noch das quadratische Bild der nach oben zeigenden Linse übrig bleibt. Dadurch erhält man ein 180° Bild, welches den gesamten Raum abbildet.

Da allerdings nur der Parcours und nicht auch die Zuschauer außerhalb davon auf dem Bild der Kamera sein und als Bewegung erkannt werden sollen, wurde das 180° Bild schließlich noch auf den Bereich des Parcours zugeschnitten.

2.3.4.8 Kollision bei der Projektpräsentation

Am Projekttag trat das Problem auf, dass die Drohne nur schwer ihre Position in der Luft halten konnte. Das hat dazu geführt, dass die Drohne mehrmals in das Netz geflogen ist. Das kann durch die Kollisionserkennung jedoch nicht verhindert werden. Die Kollisionserkennung verhindert nur, dass die Drohne, wenn sie zu nahe an einem Rand des Parcours ist, nicht weiter in diese Richtung fliegen kann.

Um die Kollision mit dem Netz am Projekttag zu verhindern, hätte die Drohne aktiv in die entgegengesetzte Richtung gesteuert werden müssen, sobald sie zu nahe am Rand ist. Das wäre jedoch mit den Tello Drohen nicht umsetzbar gewesen, da man dafür wissen müsste, wie schnell die Drohne unvorhergesehen in Richtung des Netzes steuert. Da die Tellos jedoch keine Beschleunigungs- und Richtungssensoren haben, welche so etwas messen könnten, und auch die Latenz, diese Daten zu übermitteln, zu verarbeiten und die entsprechenden Steuerungsbefehle wieder an

die Drohne zu schicken, zu groß wäre, konnte das im Rahmen dieses Projekts leider nicht verhindert werden.

2.3.4.9 Fazit zur Kollisionserkennung

Mit den Differenzbildern konnte eine effiziente Lösung gefunden werden um die Position der Drohne zu bestimmen. Die Verwendung der 360° Kamera als 180° Kamera ermöglichte es, den kompletten Parcours zu überwachen und wird bestimmt auch in weiteren studentischen Projekten Verwendung finden.

Insgesamt konnte die Kollisionserkennung maßgeblich zur Sicherheit bei der Präsentation beitragen. Sie stellte nicht nur sicher, dass das Publikum vor der Drohne geschützt wird, sondern konnte den Besuchern, welche die Drohne steuerten, Angst vor anfänglichen Fehlern nehmen.

2.3.5 Tello Dronecontrol

(Maximilian Richter)

Auch dieses Semester wurde für die Steuerung der Tello Drohne die Bibliothek „DJITelloPy“ von Damià Fuentes Escoté verwendet [2].

Diese wird über den Kommandozeilenbefehl, *pip install djitellopy* installiert.

Wie schon im Zwischenstand des Wintersemesters erwähnt, war es das Ziel eine geschwindigkeitsbasierte Steuerung zu entwickeln.

Daher konnte auf die „move-Methoden“ aus dem letzten Semester nicht mehr zurückgegriffen werden.

Die folgenden Funktionen aus der Tello-Bibliothek wurden für die finale Version von DRONEVERSE verwendet [3].

- `Drone = tello.Tello()`:
Instanziierung der Tello Klasse mit dem Namen „Drone“.
- `Drone.connect()`:
Wechseln in den SDK Modus - muss vor Steuerbefehlen einmalig aufgerufen werden.
- `Drone.takeoff()`:
Die Drohne hebt ab und hält die Position
- `Drone.send_rc_control(x, y, z, t)`
Beschleunigt die Drohne in/um vier Achsen:
 - `x` = Geschwindigkeit nach links|rechts
 - `y` = Geschwindigkeit nach vorne|zurück
 - `z` = Geschwindigkeit nach oben|unten
 - `t` = Geschwindigkeit der Drehung um die Eigene Achse
- `Drone.land()`:
Die Drohne landet.

Um das Steuerungsmodul in die Softwarearchitektur einzugliedern, wurde eine Klasse „TelloDroneController“ erstellt, über dessen Methoden Steuerungsbefehle an die Tello Drohne übergeben werden.

2.3.5.1 Initialisierung

Zunächst wird in der Klasse eine Methode zur Initialisierung der Drohnensteuerung benötigt.

In dieser Methode wird eine Instanz der importierten Tello-Klasse aus der DJITelloPy-Bibliothek erstellt (1).

Anschließend werden die Variablen für die 4 gerichteten Geschwindigkeiten deklariert und auf 0 gesetzt (2), die Drohne wird mit dem Computer, der das Programm verwendet verbunden (3) und bekommt danach den Befehl abzuheben (4).

Außerdem wird noch eine übergeordnete Geschwindigkeitsvariable deklariert (5), welche an die vier gerichteten Geschwindigkeiten übergeben werden soll, um für ein Umstellen der Drohnengeschwindigkeit nur eine Variable ändern zu müssen.

Listing 2.18: Initialisierung TelloDroneControl

```
1 def __init__(self):
2     self.drone = Tello()           #1
3     print("Drone initialized.")
4
5     self.speed_left_right = 0      #2
6     self.speed_up_down = 0
7     self.speed_forward_back = 0
8     self.yaw_speed = 0
9
10    self.drone.connect()           #3
11    print("Drone connected.")
12
13    self.takeoff()                 #4
14
15    self.speed = 30                 #5
```

2.3.5.2 Übergabe-Methode

Für das Senden der Steuerungsbefehle, anhand der 4 gerichteten Geschwindigkeiten wird die oben erwähnte Methode `send_rc_control(x,y,z,t)` benötigt.

Listing 2.19: TelloDroneControl Übergabe-Methode

```
1 def update_movement(self):  
2     self.drone.send_rc_control(  
3         self.speed_left_right,  
4         self.speed_forward_back,  
5         self.speed_up_down,  
6         self.yaw_speed,  
7     )
```

Um potenzielle Verwirrung bei der Steuerung per Gesten (2.3.1) in einem dunklen Raum zu vermeiden entschieden wir uns dafür die Drehung bei den verschiedenen Erkennungsmodulen wegzulassen. In der Steuerung ist die Drehung der Vollständigkeit halber trotzdem implementiert, der Wert wird jedoch nicht verändert.

2.3.5.3 Richtungs-Methoden

Um den verschiedenen Geschwindigkeitswerten Richtungen zuzuordnen, wurden entsprechende Methoden implementiert.

In diesen wird die jeweils betroffene Geschwindigkeit auf den zuvor in der Initialisierung festgelegten Wert der übergeordneten Geschwindigkeitsvariable festgesetzt, wobei ein Umdrehen der Vorzeichen dabei eine Richtungsänderung um 180° auf der zugehörigen Achse bewirkt.

Anschließend wird die Übergabe Methode aufgerufen, woraufhin die Drohne in die entsprechende Richtung fliegt.

Listing 2.20: TelloDroneControl Richtungs-Methoden

```
1  def up(self):
2      self.speed_up_down = self.speed
3      self.update_movement()
4      print("Tello: Up")
5
6  def down(self):
7      self.speed_up_down = -self.speed
8      self.update_movement()
9      print("Tello: Down")
```

Neben den Funktionen für die verschiedenen Richtungen gibt es zusätzlich noch eine stop-Funktion, in der alle Geschwindigkeitswerte auf 0 gesetzt werden. Sie wird an bestimmten Stellen aufgerufen damit die Drohne nicht endlos weiterfliegt:

- Gestenerkennung: Aufruf wenn keine Geste erkannt wird
- Tastaturerkennung: Aufruf beim KeyUp-Event (Taste wird losgelassen)
- OSC-Erkennung: Aufruf wenn kein OSC Steuerungssignal empfangen wird

Außerdem besitzt die Klasse TelloDroneController noch eine Methode zum landen der Drohne, welche aufgerufen wird sobald die Erkennungssoftware beendet wird.

Listing 2.21: TelloDroneControl Lande-Methode

```
1  def land(self):
2      self.drone.land()
3      print("Tello: landing")
```

2.3.6 Quadcopter Dronecontrol

(Tobias Schwarz)

Als weitere Option neben den Tello Drohnen kann man mithilfe des Quadcoptermoduls auch die Luftschiffe steuern. Im letzten Semester war die Steuerung eben jener nur per mitgelieferten Controller möglich, doch nun wurde die Steuerung des ESP mit in die Softwarearchitektur aufgenommen. Hierbei verlief die Implementierung parallel zu den Tello Drohnen, mit einem einzigen Unterschied in der Richtungsmethode und der Verwendung der „Telnetlib“-Bibliothek. Es werden hier per Telnet Verbindung die einzelnen Buchstaben bzw. Befehle an das Luftschiff gesendet. In folgenden Beispiel wird beim Befehl für „rechts“ der String „d“ versendet, bei „vorwärts“ ein „w“ – basierend auf der verbreiteten WASD-Steuerung für Tastatur (2.3.3). Die Verarbeitung der empfangenen Befehle erfolgt auf dem ESP (2.4.3).

Listing 2.22: Telnet Richtungsanweisung

```
1 def right(self):
2     tn.write("d".encode('ascii') + b"\n")
3     print("Quad: Right")
4
5     def forward(self):
6         tn.write("w".encode('ascii') + b"\n")
7         print("Quad: Forward")
```

Die Telnet-Verbindung muss anfangs initialisiert werden. Dabei werden Host-Netzwerk und Port angegeben und damit ein Verbindungsversuch mit dem Telnet-Server auf dem ESP gestartet.

Listing 2.23: Telnet Initialisierung

```
1 import telnetlib
2
3 HOST = "192.168.43.133"
4 PORT = 23
5 print("Connecting to Telnet server...")
6 tn = telnetlib.Telnet(HOST, PORT) # Verbindung zum Telnet-
   Server herstellen
7 if not telnetlib.Telnet: print("Telnet-Verbindung konnte
   nicht hergestellt werden.")
```

2.4 Quadcopter Hardware

(Tobias Schwarz)

Die Quadcopter wurden nicht nur als Steuermethode eingefügt, sondern mussten auch Hardwareseitig umgebaut werden (was bei den Tello Drohnen nicht der Fall war). Die ursprüngliche Aufgabe war es die Quadcopter per Laptop ansteuern zu können. Sollte dies funktionieren, kann gezeigt werden, dass das System zur interaktiven Steuerung auch auf unterschiedlicher Hardware angewandt werden kann.

Im Folgenden werden die Herangehensweise, Probleme, Lösungen und Kompromisse innerhalb des Projektjahres dargestellt.

2.4.1 Erste Ideen

Zu Beginn ging es darum mit den Quadcoptern zu arbeiten. Diese wurden von Prof. Lano gestellt und waren nur 4,5 x 4,5 cm groß. Sie waren lange Zeit die kleinsten Drohnen der Welt, wurden jedoch von der Skeye Pico Drohne (2,2 x 2,2 cm) abgelöst. Die Cheerson-Drohnen sind in verschiedenen Farben erhältlich, wiegen rund 90 Gramm und kosten – je nach Verkäufer – pro Stück ca. 20 €. Die Recherche zeigte, dass es zwei Versionen von ihnen gibt: die einen haben eine rote, die anderen eine blaue Platine, die sich nur im Aufbau und in der Anordnung der einzelnen Komponenten unterscheiden – der Grund hierfür ist bislang nicht bekannt. Auch wenn das zu Beginn nicht wichtig erscheint, so hat sich dieser Unterschied im weiteren Verlauf doch auf einzelne Ideen bzw. Herangehensweisen ausgewirkt. Die Kommunikation zwischen Controller und der Drohne basiert auf dem 2.4 GHz Frequenzband und findet erst nach einem manuellen Verbindungsaufbau statt. Falls (wie es hier der Fall war) mehrere Cheerson Quadcopter der gleichen Bauart vorliegen, ist es egal welchen Controller man zum Steuern verwendet, da vor jedem Start der Drohne die beiden Parteien erst synchronisiert und verbunden werden müssen. Wenn die Drohne verbunden ist, hält der Akku für ca. 3-4 Minuten im Dauerbetrieb, bis er wieder geladen werden muss.

Um die Aufgabenstellung des Steuerns per Laptop anzugehen, war die erste Idee die Drohnen zu flashen, also den internen Speicher zu löschen und einen eigenen Code darauf zu laden. Dies hätte jedoch zu Folge, dass die Drohnen für weitere Projekte

unbrauchbar wären, weshalb dieser Ansatz schnell verworfen wurde. Daraufhin kam die Idee auf, Transceiver-Module zu verwenden, genauer die gut dokumentierten **nRF24** Module. Diese können sehr einfach via Arduino und ähnlichen Plattformen angesprochen werden. Ziel hierbei war es die Verbindung zwischen Controller und Quadcopter zu analysieren, kopieren und dadurch selbstständig nachbauen zu können. Doch schon der erste Schritt, dem Analysieren, ist gescheitert. Das Frequenzband ließ sich zwar auslesen, jedoch konnten keine inhaltlichen Daten daraus entnommen werden, da die Interpretation der einzelnen Bits nicht ohne weitere Hilfe möglich war. Die letzte Möglichkeit wäre Reverse Engineering, bei dem man versucht aus bestehenden Systemen Informationen zu extrahieren oder Rückschlüsse auf einzelne Teile zu ziehen. Das hätte jedoch den (zeitlichen) Rahmen gesprengt.

Es wurde daraufhin beschlossen, zunächst den originalen Controller als Steuerungsinstrument zu verwenden und ein anderes wichtiges Ziel wurde verfolgt: den Bau eines kleinen interaktiven Luftschiffs.

2.4.2 Das Luftschiff



Abbildung 2.18: Luftschiff bei Zwischenpräsentationen

Anfangs gab Prof. Lano verschiedene Inspirationen für das Projekt in Form von Bildern und Videos. Hierbei kam vor allem das Luftschiff (zu engl. Blimp) des Channels „**FliteTest**“ zur Geltung. Dann ging es darum auch etwas Ähnliches zu kreieren: Der Cheerson Quadcopter wurde auseinandergebaut, die Kabel der

Motoren verlängert, um diese neu zu positionieren, ein Gehäuse aus MakerFoam wurde gebaut und das ganze an mehrere Heliumballons gehangen, damit genug Auftrieb entsteht. Der Auftrieb war ein elementarer Bestandteil dieses Luftschiffs: Es sollte nämlich nicht selbstständig nach oben fliegen, sondern von alleine langsam aber stetig zu Boden sinken. Was im ersten Moment vielleicht unlogisch erscheint, ist aber vor dem Hintergrund entstanden, dass man dadurch eine vereinfachte Steuerung hat. Das „hoch“ fliegen konnte man nun durch Einschalten der Rotoren, welche nach oben gerichtet waren, erlangen - das „runter“ fliegen durch einfaches Ausschalten dieser.

Damit bestand ein Proof-of-Concept: Es war ein voll flugfähiges Luftschiff, das nach vorn, hoch und runter fliegen und sich nach links und rechts drehen konnte. Das Ganze war, wie eingangs erwähnt, noch mit dem originalen Controller verwendbar, doch darauf konnte nun weiter aufgebaut werden.

2.4.3 ESP32-Versuche

Nun ging es vorrangig darum eine Verbindung zwischen Drohne und Laptop herzustellen. Die Idee den Funkverkehr zwischen Drohne und Controller zu kopieren und damit selbstständig zu nutzen wurde schon früh verworfen, doch es wurde schnell eine praktikable Lösung gefunden: ein ESP32.

Der ESP32 (folgend auch kurz „ESP“ genannt) der Firma Espressif ist ein Mikrocontroller ähnlich dem Arduino. Er ist der Nachfolger des ESP8266 und ist durch den geringen Stromverbrauch und Kosten, seine große Kompatibilität und eingebauten Systemen wie Wi-Fi und Bluetooth ideal für eine breite Palette an Projekten. Es gibt verschiedenste Ausführungen der Boards, je nach Modell haben sie unterschiedliche Pin-Outs, Größen, Optimierungen und verschiedene (zusätzliche) Bauteile.

Damit der ESP32 eigenständig laufen kann muss dieser zunächst programmiert werden. Ein Vorteil des in diesem Projekt verwendeten ESP32 Modells „WROOM“ ist, dass dieser sehr leicht und klein ist. Doch durch die kleine Bauform muss der Chip auf ein sog. „Burning Fixture Tool“, also eine Platine die jeden der kleinen Kontakte auf GPIO-Header übersetzt und per USB Kabel betrieben und programmiert werden kann. Danach kann dieser aus der Vorrichtung entfernt und beispielsweise per Akku betrieben werden, jedoch müssen alle Kabel dann

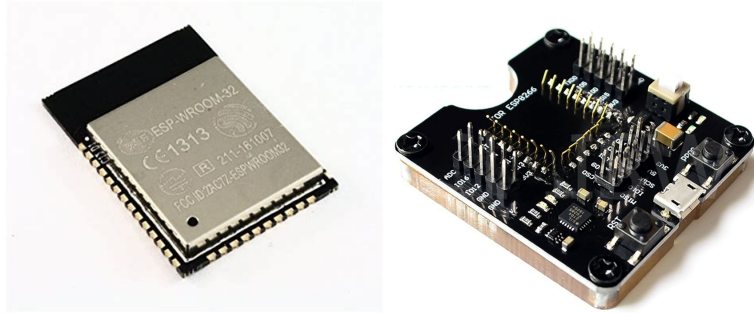


Abbildung 2.19: ESP32 WROOM (links) und Burning Fixture Tool (rechts)

festgelötet werden. Somit muss man zuerst alles programmieren und kann als aller letzten Schritt erst den ESP herausnehmen und verkabeln.

Während der Entwicklung wurde jedoch mit einem standardmäßigen Dev-Kit eines ESP32 getestet. Die unterscheidet sich vom WROOM nur darin, dass der Chip auf einer kleinen Platine mit fest installierten IO-Pins und eingebauten Spannungsregler sitzt, durch welchen man den Chip per USB-Kabel betreiben und programmieren kann. Zusätzlich lassen sich hiermit schnell verschiedenste Schaltungen und Bauteile austauschen und ausprobieren. Nach Beenden der Testphase kann der Code ganz einfach auf den WROOM übertragen und synchron zum „Test-ESP“ verwendet werden.

Um nun die Verbindung zum Laptop herzustellen gab es mehrere Herangehensweisen. Anfangs wurde die generelle technische Verbindung getestet, indem auf dem ESP ein Web-Server eingerichtet wurde. Dabei kam das Beispiel der `<ESPAsyncWebServer.h>` Library zum Einsatz. Es benötigt die Zugangsdaten des zu verwendenden Netzwerks und kann sich darauf hin damit verbinden. Als nächstes wird ein Web-Server mit einer vom Router zugewiesenen IP-Adresse aufgebaut. Im Code kann nun die Internetseite programmiert werden. Hier bestand sie aus einfachen Links, welche je einen bestimmten Befehl triggern. Beispielsweise wurde hier den Link „Turn ON 1“, welcher die LED 1 anschalten soll und dementsprechend den Befehl `„/led1/on“` ausführt, erstellt:

Listing 2.24: HTML Code auf ESP32

```

1  html += "<a href='/led1/on'>Turn ON_1</a>";
2  html += "<br><br>";
3  html += "<a href='/led1/off'>Turn OFF_1</a>";
4  html += "</html>";

```


Im Gegenzug wird im weiteren Verlauf des Codes der Befehl erkannt und schaltet die entsprechenden, vorher definierten, GPIO-Pins an oder aus:

Listing 2.25: Ausführung der HTML Befehle auf ESP

```
1  server.on("/led1/on", HTTP_GET, [](AsyncWebServerRequest *  
    request) {  
2      Serial.println("ESP32 Web Server: New request received:"  
        );  
3      Serial.println("GET /led1/on");  
4      LED_state = HIGH;  
5      digitalWrite(LED_PIN_1, LED_state);  
6      request->send(200, "text/html", getHTML());  
7  });  
8  server.on("/led1/off", HTTP_GET, [](AsyncWebServerRequest  
    *request) {  
9      Serial.println("ESP32 Web Server: New request received:"  
        );  
10     Serial.println("GET /led1/off");  
11     LED_state = LOW;  
12     digitalWrite(LED_PIN_1, LED_state);  
13     request->send(200, "text/html", getHTML());  
14 });
```

Nach Fertigstellung des Codes wird dieser via der ArduinoIDE auf den ESP geschrieben.

Die Herangehensweise mit dem Webserver war zwar schnell erfolgreich, erwies sich aber nach kurzer Zeit als recht ineffektiv. Die Befehle wurden nur dann ausgeführt, wenn der Link angeklickt bzw. die Internetseite mit dem bestimmten Tag aufgerufen wurde. Zusätzlich musste sich die Website dann noch im Hintergrund mit aufbauen, auch wenn sie am Ende nicht mehr angezeigt werden sollte. Das alles hat die erstrebte Echtzeitanwendung verlangsamt und unnötig verkompliziert.

2.4.4 Programmierung des ESP

Eine neue Idee musste her und nach einiger Recherche wurde sich für das etwas veraltete, aber hierfür doch gut passende, Telnet Protokoll entschieden. Luber [7] beschreibt das Protokoll folgendermaßen: „Über eine Client-Server-Verbindung mit

dem Telnet-Protokoll ist der Remote-Zugriff auf einen entfernten Rechner per IP-Netz möglich. Mit textbasierten Kommandos lassen sich auf dem entfernten Rechner Funktionen ausführen. Ein geeigneter Client ist in fast allen Betriebssystemen integriert.“

Eine Client-Server-Verbindung beschreibt also einen Teilnehmer der als Server agiert und auf den Befehl bzw. eine Anfrage eines Clients (der zweite Teilnehmer) wartet. Dabei sendet der Client einfachen Text als String an den Server, welcher diesen interpretiert und je nach Programmierung darauf reagiert. In unserem Fall sollte die Drohne als Server und den Laptop als Client genutzt werden, so kann der Laptop der Drohne Befehle schicken und diese steuern.

Das wurde mit den `<TelnetStream.h>` und `<WiFi.h>` Bibliotheken für die ArduinoIDE bzw. den ESP erzielt. Der grobe Aufbau des finalen Codes sah wie folgt aus: Zuerst wurde eine Verbindung mit dem selbst vorgegebenen Netzwerk aufgebaut, dann startete der Server bzw. der sog. TelnetStream. In einer Loop-Schleife wurde ständig überprüft ob und welche Zeichen beim Server ankamen und bei Übereinstimmung mit den festgelegten Zeichen wurden die bestimmten GPIO-Pins angeschalten, um die Motoren zu starten.

Die festgelegten Befehle waren 'w', 'a', 's', 'd', 'x' zum Steuern und 't' zum Testen mit einfachem Print-Befehl. Bei Aufruf der Steuerbefehle werden alle Motoren gestoppt, dann nur die benötigten eingeschaltet und eine halbe Sekunde verzögert. In der Stop() Funktion werden alle Pins via `digitalWrite(x,LOW)` ausgeschaltet.

Listing 2.26: Erkennung Taste 'W' auf ESP32

```
1  case 'w':  
2      stop();  
3      Serial.println(incomingChar);  
4      digitalWrite(12,HIGH);  
5      digitalWrite(27,HIGH);  
6      TelnetStream.println("Test erfolgreich: w");  
7      delay(500);  
8      break;
```

Die Steuerung basierte auf der weitverbreiteten WASD-Steuerung per Tastatur. Die Taste 'w' steht für „vorne“, also Gas geben und nach vorne fliegen. Die Tasten 'a' und 'd' für „links“ und „rechts“ und 's' steht für „nach oben“, damit konnten alle Tasten eng beieinander und einfach erreichbar für den Piloten sein.

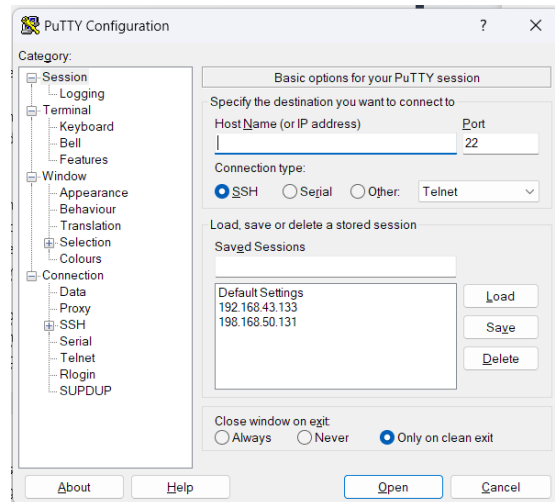


Abbildung 2.20: Screenshot des PuTTY Clients

Nur der ESP-Code alleine reicht nicht, es müssen auch die Befehle des Clients dort ankommen. Dafür kann man sich mit einem Telnet Client auf den ESP-Server verbinden. Die einfachste Methode um das ganze zuerst technisch auszuprobieren war mithilfe des Programms PuTTY.

PuTTY ist ein kostenloser open-source SSH und Telnet Client für Windows. Er wurde ursprünglich von Simon Tatham erstellt und wird nun von Freiwilligen mit unterstützt. Bei erfolgreicher Verbindung kann man wie mit der gewohnten Kommandozeile Befehle bzw. Text an den Telnet Server senden.

Nach erfolgreichen Tests mit PuTTY konnte die Funktionalität des ESP-Codes bewiesen werden, jedoch hatte das Programm noch keine wirklich flüssige Anwendung. Jeder Befehl musste mit der Eingabe-Taste bestätigt werden, was bei einer möglichst schnellen und präzisen Steuerung hinderlich ist. Optimal wäre ein Programm, das jeden Tastenanschlag direkt an den Telnet Server sendet ohne vorher noch eine Bestätigung zu benötigen. Leider war dies mit dem PuTTY Programm nicht möglich, weshalb dafür ein eigenes Python Skript erstellt wurde. Dies hatte zusätzlich den Vorteil, dass es zu späteren Zeitpunkt noch in die Softwarearchitektur mit eingearbeitet werden konnte.

2.4.5 Tastatursteuerung

Zu Beginn der Versuche wurde die einfache Python-Bibliothek „keyboard“ verwendet, diese hatte jedoch nur eine Funktion für die Aktion Drücken UND Loslassen

einer Taste. Es wurde aber je Aktion ein eigener Befehl benötigt, weshalb diese Bibliothek schnell ausschied und verworfen wurde. Doch die Bibliothek „pygame“ hatte alles Nötigen, auch wenn sie weit mehr Funktionen beinhaltete als im Rahmen des Projekts benutzt werden konnte.

Pygame [4] ist eine Sammlung von Python-Modulen, die für die Entwicklung von Spielen konzipiert wurden. Sie basiert auf der leistungsstarken SDL-Bibliothek und ermöglicht die Erstellung von voll funktionsfähigen Spielen und Multimedia-Programmen für Python. Sie zeichnet sich durch eine hohe Portabilität aus und läuft auf nahezu allen Plattformen und Betriebssystemen.

Im Code wird jede Sekunde einmal abgefragt, welche vorher festgelegte Taste gedrückt wird. Falls eine erkannt wird, wird per Telnet Verbindung (tn.write) das entsprechende Zeichen übertragen.

Listing 2.27: pygame Code Ausschnitt

```
1      clock.tick(1)
2      for event in pygame.event.get():
3          if event.type == pygame.QUIT:
4              [...]
5          elif event.type == pygame.KEYDOWN:
6              if event.key == pygame.K_a:
7                  print("left")
8                  tn.write("a".encode('ascii') + b"\n")
9              [...]
10         elif event.type == pygame.KEYUP:
11             tn.write("x".encode('ascii') + b"\n")
12             print("stop")
```

Die Idee war es immer wenn eine Taste gedrückt wurde, soll diese und bei Loslassen ein Stop-Befehl übertragen werden. Die reine Übertragung der Zeichen war schnell eingerichtet und konnte einfach realisiert werden. Jedoch die Menge der übertragenen Zeichen war ein Problem, da diese sowohl Hunderte Male in der Sekunde erkannt wurden, als auch in einer internen Liste gespeichert und nach und nach an den ESP gesendet wurden. Es kam daraufhin zu einer nicht verwendbaren Flut an Befehlen: Ein Zeichenanschlag führte demnach zu extrem langen Ausführen eines Befehls, auch wenn die Taste nicht mehr gedrückt wurde. Um dem entgegenzuwirken wurde die Clock-Rate auf eine Sekunde gesetzt. Das war keine optimale

Lösung, da nun auch eine leichte Verzögerung zwischen den einzelnen Befehlen auftrat, aber um ein Vielfaches besser als die ursprüngliche Version.

2.4.6 Bau des neuen Luftschiffs

Zu Beginn wurde die Schaltung mit dem ESP32 auf einem Breadboard realisiert. Zunächst wurde anstatt der Motoren, die am Ende verbaut werden, nur mit kleinen roten LEDs gearbeitet. Diese werden direkt mit den GPIO-Pins verbunden und dienen somit als einfache optische Indikatoren, ob die gewünschten Pins geschaltet werden oder nicht.

Der Aufbau blieb einige Zeit bestehen, um die verschiedenen Code Versionen zu überprüfen und kleine Fehler, wie vertauschte IO Nummern, zu korrigieren. Nachdem alles funktionierte war der nächste Schritt den ESP nicht mehr per USB Kabel an den Laptop anzuschließen, um den Serial Monitor auszulesen, sondern per Akku. Es standen einige RC-Akkus zur Verfügung. Sie hatten alle eine Spannung von 3.7V aber verschiedene mAh (sie reichten von 160mAh bis 600mAh, was sehr das Gewicht beeinflusste). Die Akkus konnten ohne Probleme mit dem 3.3V durchgeschliffenen Anschluss und Ground des ESP Dev-Kits verbunden werden, da dieser einen eigenen Spannungsregler eingebaut hatte. Das hat zunächst ohne weitere Komplikationen funktioniert und damit konnte das Ganze eigenständig und ohne direkte Verbindung zum Laptop verwendet werden (im späteren Verlauf gab es dabei mit dem ESP WROOM Probleme; vgl. Kapitel [2.4.7](#)).

Im nächsten Schritt wurden die Transistoren eingeführt. Diese agieren als Schalter und können bei einer geringen Spannung an der Basis einen größeren Stromkreis schließen. Der Plan war, dass die GPIO-Pins die Transistoren und am Ende dadurch die Motoren schalten. Nach anfänglichen logischen Schwierigkeiten konnten diese aber schnell eingebaut werden. Auch hier wurde zunächst mit den LEDs gearbeitet, um ein optisches Feedback der geschalteten Transistoren zu bekommen. Hier wurde daraufhin der zweite Akku eingeführt, da der Stromverbrauch durch die Motoren enorm angestiegen ist und man so die Flugdauer verlängern konnte.

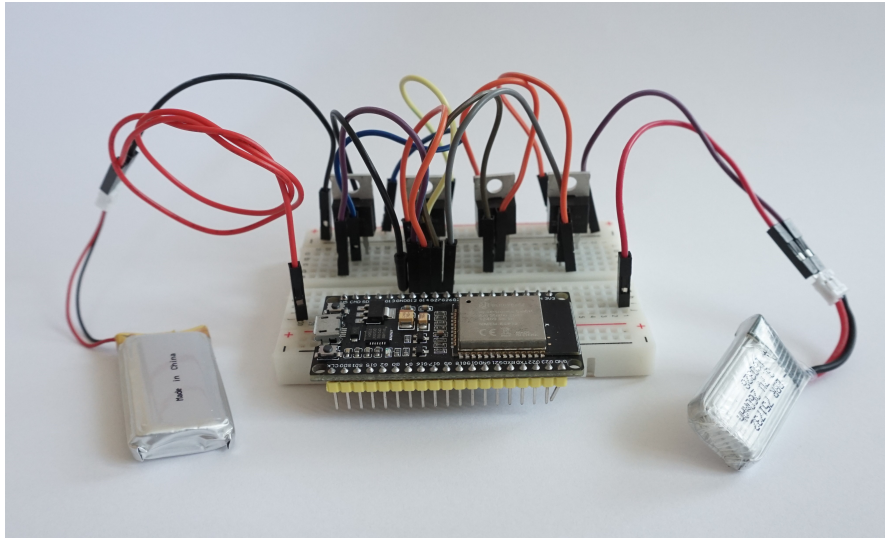


Abbildung 2.21: ESP mit Transistoren und Akkus auf Breadboard

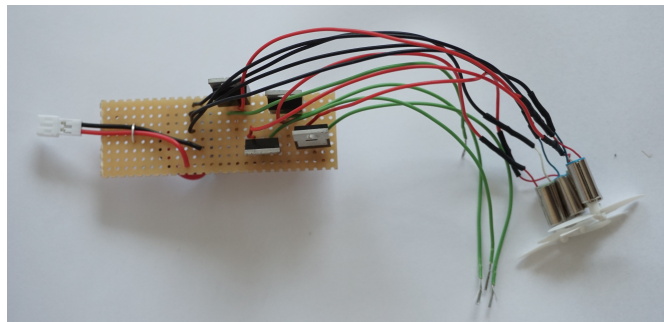


Abbildung 2.22: Lochplatine mit Motoren und grüne Kabel für ESP

Schlussendlich konnten die Motoren des Luftschiffs aus der Zwischenpräsentation verbaut werden. Diese wurden an die Transistoren und den gemeinsamen Ground angeschlossen und konnten nun per Laptop über den Telnet Stream und das Schalten der GPIO-Pins gestartet und gestoppt werden.

Da nun alles funktionierte wurde die gesamte Schaltung auf eine Lochplatine übertragen (vgl. [Abbildung 2.21](#)). Dabei konnte einiges an Gewicht eingespart werden, da das Luftschiff am Ende auch möglichst leicht sein sollte.

Im letzten Schritt wurde alles in der schon vorhandenen Form aus MakerFoam verbaut und konnte mit einigen Ballons final präsentiert werden.

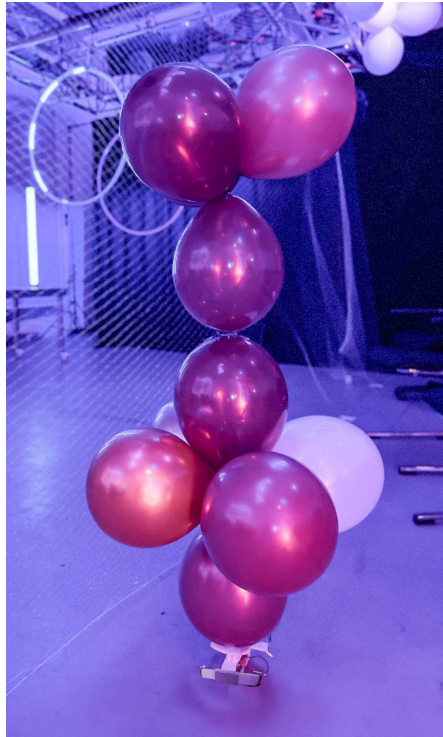


Abbildung 2.23: Luftschiff bei Abschlusspräsentation

2.4.7 Aussicht

Zusammenfassend kann gesagt werden, dass das Projekt der Quadcopter erfolgreich war. Anfangs wurde mit einer fertigen kleinen Drohne begonnen worden und schließlich endete es mit einem komplett selbst gebauten Luftschiff, das an Heliumballons hängt. Falls das Projekt weitergeführt werden sollte, gibt es dennoch noch einige Verbesserungs- und Optimierungsansätze.

Zu aller erst das Gewicht. Durch die zwei Akkus ist extrem viel Gewicht im Gegensatz zum ursprünglichen leichten Quadcopter dazugekommen. Dazu kommen noch die vier Feldeffekttransistoren, die zusammen auch noch einiges an Gewicht hinzufügen. Um dem Ganzen entgegenzuwirken könnte man auf nur einen (ggf. etwas größeren) Akku zurückgreifen, um insgesamt vielleicht etwas weniger mAh zur Vergnügung zu haben aber dennoch genug um lange genug zu fliegen - das müsste man ausprobieren. Die Feldeffekttransistoren können ggf. durch sehr kleine SMD-Bausteine ausgetauscht werden und im gleichen Zug wäre eine extra dafür bestückte Platine mit ebenjenen SMD-Bauteilen eine enorme Gewichtsreduzierung. Es war auch im Raum gestanden eine Platine ätzen und bestücken zu lassen, doch der Aufwand und die Mindeststückzahl für etwas, von dem man nicht weiß ob

es genau in dem Zustand bleibt, war einfach eine zu große Hürde. Für weitere Versuche ist dies aber eine legitime Herangehensweise.

Ein Punkt der theoretisch noch mit zu den Bauteilen gehört ist der ESP WROOM. Dieser sollte kurz vor der Abschlusspräsentation in die Schaltung integriert werden und das ESP Dev-Kit ablösen. Im Burning Fixture Tool hat alles funktioniert (Code, Serial Monitor und Verkabelung). Doch als der ESP im letzten Schritt verlötet werden sollte, hat es nicht mehr funktioniert. Die Kabel waren alle verbunden, es gab keine Kurzschlüsse oder Brücken aber es konnte kein Telnet Server aufgebaut werden, die Vermutung war das intern etwas zu Schden gekommen ist, aber das wurde nicht mehr herausgefunden. Daraufhin wurde kurzfristig auf das ESP Dev-Kit gewechselt, welches jedoch sperriger und schwerer war. Somit besteht hier auch noch Optimierungspotenzial.

Aber sowohl die Bauteile, als auch die Optik kann noch verbessert werden. So war es am Ende des letzten Semesters schon ein Ziel ein Gehäuse für das Luftschiff zu gestalten und bsp. mit dem 3D-Drucker zu drucken. Dieses Ziel ist aber leider zeitlich nicht mehr erreichbar gewesen und hätte dem Luftschiff den letzten optischen Schliff verpasst.

Trotz dieser Punkte war das Quadcopter-Projekt ein voller Erfolg und bietet eine gute Basis für zukünftige Weiterentwicklungen durch andere Studentengruppen oder für neue innovative Ideen.

2.5 Hindernis-Parcours

(Anusha Kanagarasa)

2.5.1 Gestaltung und Umsetzung des Hindernis-Parcours

Im Rahmen dieses Projekts wurde ein interaktiver Parcours entworfen und aufgebaut. Ziel des Projekts war es, einen Raum zu schaffen, in dem Nutzer mittels Gestensteuerung (2.3.1) durch verschiedene Hindernisse navigieren können. Der gesamte Prozess umfasste die Planung, Konstruktion und letztlich den Aufbau des Parcours, wobei besonderes Augenmerk auf die Balance zwischen Herausforderung und Zugänglichkeit für Anfänger gelegt wurde.

2.5.2 Planung & Konstruktion

Zu Beginn des Projekts stand die Inspektion und Vermessung des Raumes. Dies war ein nötiger Schritt, um die räumlichen Gegebenheiten genau zu erfassen und die Dimensionen für die Hindernisse festzulegen. Durch sorgfältiges Messen konnten die zur Verfügung stehende Fläche und die Platzierung der Hindernisse optimal geplant werden. Dabei wurden sowohl die Höhe als auch die Breite des Raumes berücksichtigt, um ein harmonisches und funktionales Design zu gewährleisten. Nach der Vermessung des Raumes begann die Phase der Ideenfindung und Konzeptentwicklung. Verschiedene Entwürfe wurden dabei erstellt, um unterschiedliche Gestaltungsmöglichkeiten zu evaluieren. Hierbei wurde das Zeichentool SketchUp genutzt, um die ersten Entwürfe genau zu visualisieren.

Der erste Entwurf des Parcours wurde mit SketchUp erstellt und enthielt eine Vielzahl von Hürden, wie in der beigefügten Abbildung zu sehen ist. Der Raum mit den Maßen 10,6 m x 10,77 m, bot ausreichend Platz für die verschiedenen Elemente. Die Hindernisse waren jedoch zahlreich und komplex angeordnet, was für Personen ohne Vorerfahrungen in der Gestensteuerung (2.3.1) eine erhebliche Herausforderung darstellen könnte.

Der Entwurf umfasste mehrere Bereiche, in denen die Teilnehmer durch unterschiedlich große und geformte Hindernisse navigieren mussten. Dies beinhaltete unter anderem das Durchfliegen von mehreren Reifen, das Manövrieren durch waagrechte

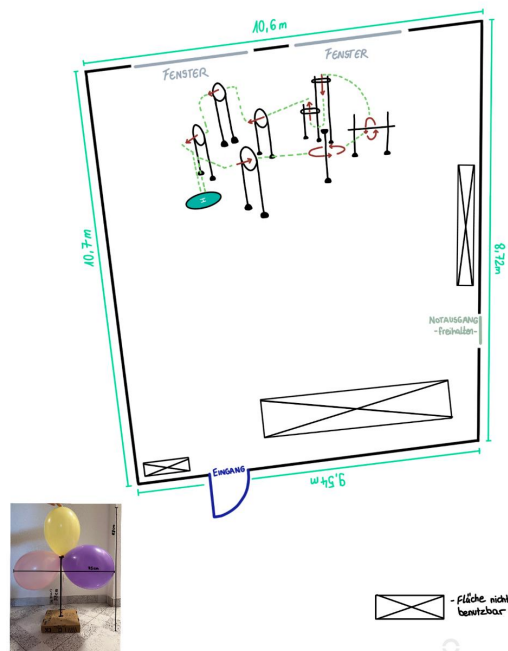


Abbildung 2.24: Erster Entwurf mit SketchUp

Stangen und das Umkreisen von Stangen. Obwohl der Entwurf visuell ansprechend und abwechslungsreich gestaltet war, stellte sich heraus, dass die Vielzahl der Hürden und deren Anordnung zu einer Überforderung der Nutzer führen könnte.

Ein entscheidender Punkt war die Erkenntnis, dass der Parcours aus der Perspektive einer steuernden Person, die keine Erfahrung mit der Gestensteuerung hat, als zu komplex und schwierig empfunden wurde. Die Teilnehmer wären möglicherweise mit der Vielzahl der Aufgaben und der erforderlichen Präzision überfordert gewesen, was den Spaß und die Motivation erheblich beeinträchtigt hätte.

Diese Überlegungen führten zu der Entscheidung, den Entwurf zu überarbeiten und die Hindernisse zu vereinfachen. Der zweite Entwurf fokussierte sich auf einfacher strukturierte und klar definierte Hindernisse, um eine bessere Nutzererfahrung zu ermöglichen. Die neue Gestaltung zielte darauf ab, den Parcours zugänglicher zu machen und sicherzustellen, dass auch Personen ohne Vorerfahrung in der Gestensteuerung den Parcours erfolgreich absolvieren können.

Im zweiten Entwurf wurden die Hindernisse so gewählt und angeordnet, dass sie einerseits eine Herausforderung bieten, andererseits aber auch leicht verständlich und überschaubar sind. Der neue Parcours setzte sich aus drei Hauptkomponenten zusammen:

1. Hindernis 1: Drei Hula-Hoop-Reifen

- Die Hula-Hoop-Reifen wurden stufenweise aufgehängt, sodass die Teilnehmer durch sie hindurchfliegen müssen. Die stufenweise Anordnung der Reifen bot eine interessante visuelle und physische Herausforderung, die jedoch nicht zu komplex war. Durch die unterschiedliche Höhe der Reifen wurde die motorische Geschicklichkeit der Teilnehmer gefordert, ohne sie zu überfordern.

2. Hindernis 2: Drei LED-Säulen

- Die drei LED-Säulen wurden so platziert, dass die Teilnehmer im Slalom durch sie hindurch navigieren müssen. Diese Säulen boten eine klare und definierte Strecke, die visuell ansprechend war und durch die LED-Beleuchtung zusätzlich hervorgehoben wurde. Das Slalom-Hindernis erforderte eine präzise Steuerung, war aber dennoch für Anfänger gut machbar.

3. Hindernis 3: Ein Metallrahmen mit zwei Öffnungen

- Der Metallrahmen mit zwei Öffnungen bot unterschiedliche Schwierigkeitsgrade. Die untere Öffnung war leichter zu durchfahren, während die obere Öffnung eine höhere Präzision erforderte. Diese Option ermöglichte es den Teilnehmern, je nach ihrem Können und Vertrauen, die für sie passende Herausforderung zu wählen. Dies förderte die individuelle Anpassung und erhöhte den Spaßfaktor.

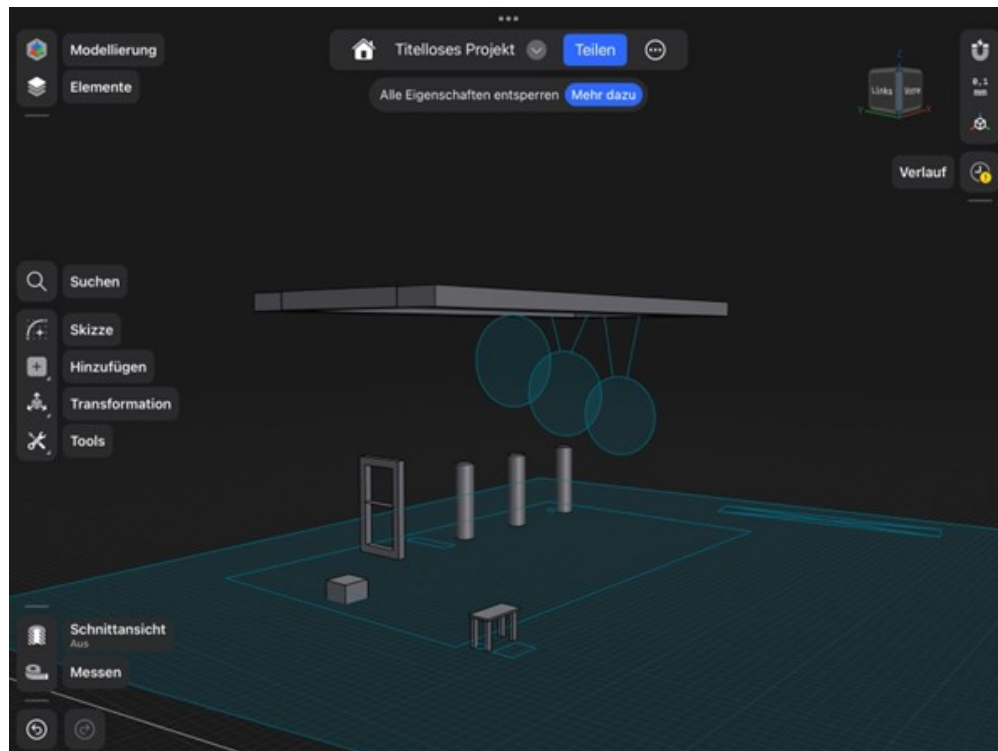


Abbildung 2.25: Zweiter Entwurf des Parcours auf Shapr3D

Der zweite Entwurf war nicht nur funktional, sondern auch ästhetisch ansprechend gestaltet. Die Hindernisse wurden in einer Weise platziert, die den gesamten Raum optimal nutzte und gleichzeitig eine klare und logische Abfolge ermöglichte. Durch die Reduzierung der Anzahl und Komplexität der Hindernisse wurde sichergestellt, dass der Parcours für alle Teilnehmer zugänglich blieb und gleichzeitig eine spannende und herausfordernde Erfahrung bot.

2.5.3 Konstruktion mit Shapr3D

Für die detaillierte Konstruktion des finalen Parcours wurde die App Shapr3D verwendet. Die App erlaubt umfassende 3D-Designs und half bei der Darstellung und Konfiguration der Hindernisse im Raum. Shapr3D ist eine professionelle CAD-Computer-Aided Design-Software-Suite, die für die Verwendung auf Tablets und Computern entwickelt wurde und für ihre leichte Benutzbarkeit und intuitive Bedienbarkeit bekannt.

Shapr3D hat es ermöglicht, komplizierte Geometrien und räumliche Modelle zu schaffen, die in Anwendungen wie Produktdesign, Architektur und Ingenieurwesen von großem Nutzen sein können. Die App ermöglichte es einem Benutzer, genaue 3D-Formen zu erstellen, indem er ihm Werkzeuge zur Verfügung stellte, um die Form und Struktur eines Objekts präzise zu schneiden, zu formen und von verschiedenen Seiten zu betrachten. Dies machte es einfach, Entwürfe iterativ zu verbessern und sicherzustellen, dass die Hindernisse im Parcours sowohl ästhetisch ansprechend als auch funktional waren.

Ein weiterer Vorteil von Shapr3D war die Fähigkeit, Modelle in einer Vielzahl von Dateiformaten zu exportieren, die dann mit anderen CAD-Paketen kompatibel waren oder für die Herstellung eingesetzt werden konnten. Dadurch konnte man den Parcours nicht nur digital entwerfen, sondern auch sicherstellen, dass die Konstruktion den realen Anforderungen entsprach. Insgesamt bot Shapr3D eine leistungsstarke und flexible Plattform, die die präzise und effiziente Gestaltung des Hindernisparcours ermöglichte.

Funktionen und Möglichkeiten:

- Intuitive Bedienung: Dank der benutzerfreundlichen Oberfläche war Shapr3D auch für Einsteiger leicht zu bedienen.
- Präzise Modellierung: Die Software bot Werkzeuge für exakte Konstruktionsarbeiten und Detailanpassungen.
- Integration: Modelle konnten einfach exportiert und in anderen Anwendungen weiterverarbeitet werden.

2.5.4 Aufbau des Parcours

Für die detaillierte Konstruktion des finalen Parcours wurde die App Shapr3D verwendet. Der Aufbau des Parcours im Raum BB.008 wurde schließlich in die Tat umgesetzt, wobei einfache Mittel verwendet wurden. Hula-Hoop-Reifen und Schwimmnudeln, die in einem Laden beschafft wurden, dienten als flexible und günstige Hindernisse. Diese Elemente boten eine gute Balance zwischen Herausforderung und Zugänglichkeit.

Zusätzlich wurden LED-Säulen und Metallrahmen von den Laboringenieuren zur Verfügung gestellt, die dem Parcours visuelle und physische Herausforderungen hinzufügten. Die LED-Säulen wurden strategisch platziert, um die Teilnehmer mit einem Slalom-Parcours herauszufordern, während der Metallrahmen mit seinen unterschiedlichen Öffnungen verschiedene Schwierigkeitsgrade bot.

Zur optischen Aufwertung des Parcours wurden die Reifen mit weißer Sprühfarbe lackiert und mit LED-Streifen dekoriert. Diese Streifen wurden mithilfe der App WLED gesteuert, was eine einfache Farbanpassung und somit eine dynamische Atmosphäre ermöglichte. Die LED-Streifen wurden so konfiguriert, dass sie über IP-Adressen und Ports in der App WLED gesteuert werden konnten. Die Steuerung der LED-Streifen erfolgt auch über das Netzwerk, das durch den ASUS RT-AX55 Router bereitgestellt wird, was eine flexible und zentrale Kontrolle ermöglicht. (2.2)

Die Hindernisse wurden gemäß den Planungen und Entwürfen aufgebaut, wobei auf Stabilität und Sicherheit geachtet wurde. Nach dem Aufbau erfolgten mehrere Testläufe, um die Funktionalität und Benutzerfreundlichkeit des Parcours zu überprüfen. Eventuelle Schwachstellen wurden identifiziert und behoben, um eine optimale Balance zwischen Herausforderung und Benutzerfreundlichkeit zu erreichen.

2.5.5 Aussicht zum Hindernis-Parcours

Das Projekt zur Gestaltung und Umsetzung eines interaktiven Parcours war ein Erfolg. Durch sorgfältige Planung, präzise Konstruktion und kreativen Einsatz von Materialien konnte ein spannender und zugänglicher Parcours geschaffen werden. Die Kombination aus einfachen Hindernissen und ansprechender Beleuchtung sorgte

für eine anregende und unterhaltsame Nutzererfahrung. Zwar hat nicht jeder den gesamten Parcours am Projekttag geschafft, aber dennoch hatten alle viel Spaß dabei.

2.5.6 Finale Bilder der Umsetzung

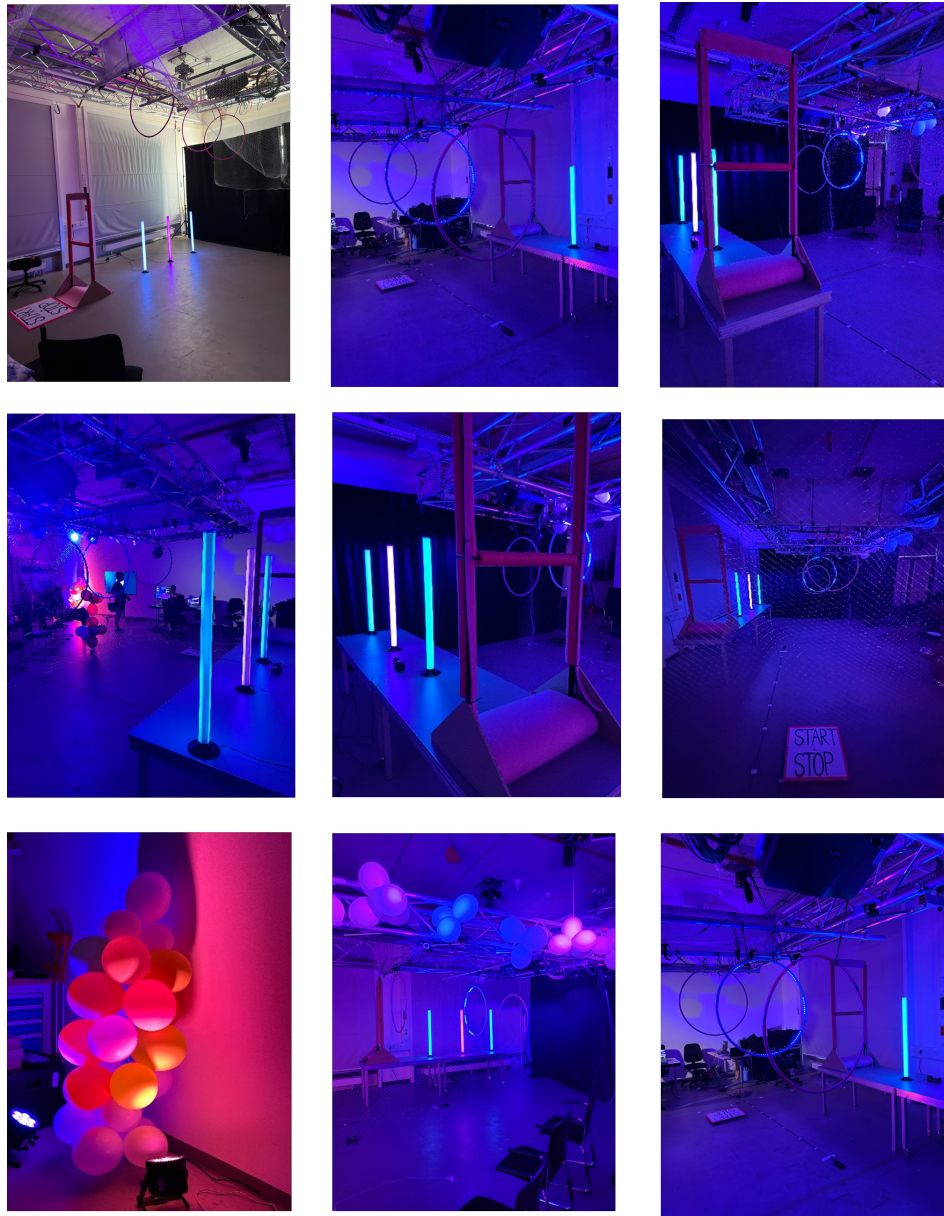


Abbildung 2.26: Finale Umsetzung

2.5.7 Designerstellung der Highscore-Anzeige

(Katharina Hölzl)

Für unser Spiel wurde eine Bildschirmausgabe in Adobe Illustrator gestaltet, die sowohl die Spieler als auch die Zeit anzeigt, in der sie den Hindernis-Parcours durchflogen haben. Das fertige Design umfasst unser Logo, den Markennamen und eine Überschrift, welche die Spieler zum Parcours einlädt. Der Hintergrund ist dynamisch und spielerisch gestaltet, passend zum Charakter des Hindernis-Parcours-Spiels. Das Design erzeugt den Eindruck als wäre ein Spotlight auf unser Logo gerichtet, was den Bezug zwischen Projekt und Spieler verstärkt. Es ist in den Farben unseres Corporate Designs gehalten, was für einen einheitlichen und professionellen Auftritt sorgt.

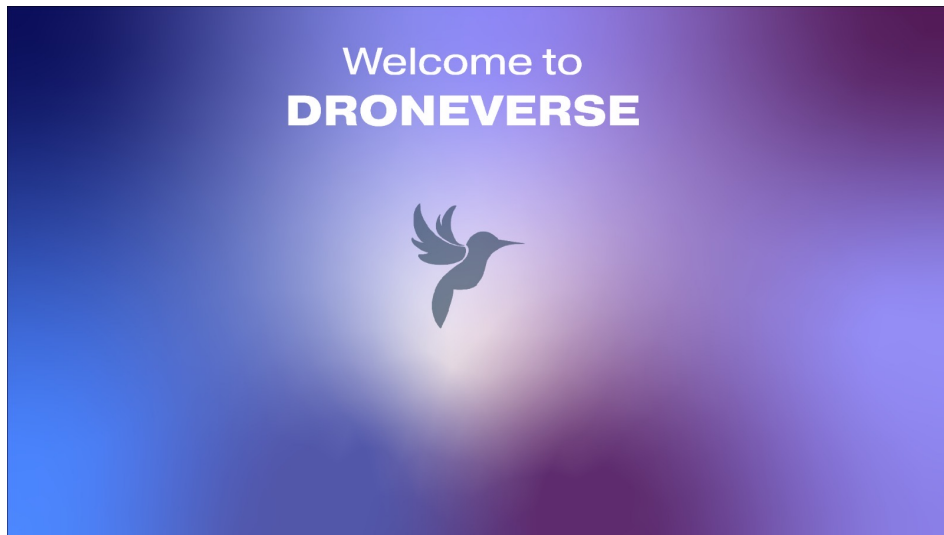


Abbildung 2.27: Hintergrund für das Game-GUI

2.5.8 Entwicklung der Highscore-Anzeige

(Anusha Kanagarasa)

Entwicklung des GUIs für die Top-10-Liste

Im Rahmen des Projekts war es wichtig, eine benutzerfreundliche und visuell ansprechende Benutzeroberfläche (GUI) zu gestalten. Diese sollte es den Benutzern ermöglichen, die Zeitmessung der Drohne zu steuern und die besten Flugzeiten

in einer Top-10-Liste anzuzeigen. Die GUI wurde in Python mithilfe der tkinter-Bibliothek entwickelt und kommuniziert über eine serielle Schnittstelle mit einem Arduino, der die Zeitmessung übernimmt.

Zielsetzung:

Die Hauptziele dieser Implementierung waren:

- Echtzeit-Timer: Ein Timer, der die Flugzeit der Drohne misst und diese in Echtzeit auf der GUI anzeigt.
- Serielle Kommunikation: Eine zuverlässige Datenübertragung zwischen dem Arduino und der GUI, um die gemessenen Zeiten darzustellen.
- Top-10-Liste: Eine automatische Sortierung und Anzeige der zehn besten Zeiten, die von den Benutzern erzielt wurden.
- Benutzerinteraktion: Einfache und intuitive Funktionen zur Steuerung des Timers und zum Speichern der Ergebnisse.

Implementierung der Benutzeroberfläche

1. Gestaltung des Hauptfensters

Die GUI wurde mit tkinter, einer in Python integrierten Bibliothek zur Erstellung von grafischen Benutzeroberflächen, implementiert. Das Hauptfenster wurde so gestaltet, dass es nicht nur funktional, sondern auch ansprechend aussieht.

Listing 2.28: Benutzeroberfläche

```
1
2 # Hauptfenster erstellen
3 root = tk.Tk()
4 root.title("Droneverse Timer")
5
6 # Fenstergröße festlegen
7 root.geometry("1500x800")
8
9 # Bild laden und als Hintergrund anzeigen
10 background_image = PhotoImage(file="bg_new.png")
11 label = tk.Label(root, image=background_image)
12 label.place(x=0, y=0, relwidth=1, relheight=1)
```

```
13 \end{verbatim}
```

Hier wird das Hauptfenster erstellt und ein Hintergrundbild geladen, das den gesamten Bildschirm ausfüllt. Dieses Hintergrundbild wurde passend zu unserem Spiel gestaltet und trägt maßgeblich zur optischen Attraktivität der Benutzeroberfläche bei. Es verleiht der Anwendung eine professionelle und klare visuelle Ästhetik.

2. Anzeige des Timers

Der Timer, der die Flugzeit der Drohne misst, wird innerhalb eines abgerundeten Rechtecks angezeigt, das auf dem Canvas-Element der GUI gezeichnet wird. Dieses Design wurde gewählt, um die Zeitmessung optisch hervorzuheben und ihr einen zentralen Platz auf der Benutzeroberfläche zu geben.

Listing 2.29: Anzeige des Timers

```
1
2 # Zeitlabel innerhalb des Rechtecks hinzufügen
3 time_label = canvas.create_text(
4     (rect_x0 + rect_x1) // 2,
5     (rect_y0 + rect_y1) // 2,
6     text="00:00:00.000",
7     font=("Helvetica", 40, "bold"),
8     fill="#192632"
9 )
```

Das `time_label` zeigt die aktuelle Zeit in einem großen, gut lesbaren Format an. Die Anzeige wird regelmäßig aktualisiert, sobald neue Daten vom Arduino empfangen werden.

3. Serielle Kommunikation mit Arduino

Ein zentrales Element der Implementierung ist die Kommunikation mit dem Arduino über eine serielle Schnittstelle. Hierbei wird die gemessene Zeit vom Arduino an die GUI gesendet und dort angezeigt.

Listing 2.30: Serielle Kommunikation mit Arduino

```
1 # Serielle Verbindung zum Arduino herstellen
2 ser = serial.Serial('COM5', 9600)
3
```

```

4 # Arduino Timer auslesen und aktualisieren
5 def read_from_arduino():
6     if ser and ser.in_waiting > 0:
7         try:
8             line = ser.readline().decode('utf-8').rstrip()
9             canvas.itemconfig(time_label, text=line)
10 except:
11     pass
12 root.after(100, read_from_arduino)

```

Die Funktion `read_from_arduino()` liest kontinuierlich die Daten von der seriellen Schnittstelle und aktualisiert die Timer-Anzeige auf der GUI. Dies ermöglicht eine Echtzeitübertragung der Messwerte.

Implementierung der Top-10-Liste

1. Aufbau und Anzeige der Liste

Die Top-10-Liste wurde implementiert, um die besten Flugzeiten in einer sortierten Reihenfolge anzuzeigen. Die Liste wird dynamisch aktualisiert, wenn neue Zeiten eingegeben werden.

Listing 2.31: Top 10 Liste

```

1 # Top 10 Spielerliste
2 top10_list = []
3
4 def update_top10():
5     for widget in root.place_slaves():
6         if isinstance(widget, tk.Label) and "Top 10 Player"
7         not in widget.cget("text"):
8             widget.destroy()
9     top10_sorted = sorted(top10_list, key=lambda x: x[1])
10    for i, (name, time) in enumerate(top10_sorted[:10]):
11        y_pos = rect_y0 - 20 + (i + 1) * 33
12        tk.Label(root, text=f"{i+1}. {name} {time}", font=
13        ("Helvetica", 20, "bold"), fg="#192632",
14        bg="white").place(x=810, y=y_pos)

```

In dieser Funktion wird zunächst die bestehende Anzeige der Liste gelöscht, um Platz für die aktualisierten Daten zu schaffen. Anschließend wird die Liste nach den Zeiten sortiert, sodass die schnellsten Zeiten oben stehen.

2. Benutzerinteraktion: Namen und Zeiten speichern

Neben der automatischen Anzeige der Top-10-Zeiten ermöglicht die GUI den Benutzern, ihren Namen und die erreichte Zeit in die Liste einzutragen.

Listing 2.32: Speicherung der Namen & Zeiten

```
1 def save_time():
2     name = name_var.get()
3     time_text = canvas.itemcget(time_label, "text")
4     if name and time_text != "00:00:00.000":
5         top10_list.append((name, time_text))
6         update_top10()
```

Mit der Funktion `save_time()` können die Benutzer ihren Namen und die gemessene Zeit in die Top-10-Liste aufnehmen. Dies wird nur dann ausgeführt, wenn ein Name eingegeben wurde und eine gültige Zeit vorliegt.

Steuerung des Timers

Neben der Anzeige der Zeit und der Verwaltung der Top-10-Liste bietet die GUI auch Funktionen zur Steuerung des Timers, einschließlich Starten, Stoppen und Zurücksetzen.

Listing 2.33: Steuerung des Timers

```
1 def start_timer():
2     ser.write(b's') # Signal an Arduino senden, um die
3                     # Messung
4                     # zu starten
5     canvas.itemconfig(time_label, text="00:00:00.000")
6     # Timer zur cksetzen
7
8 def stop_timer():
9     ser.write(b'p') # Signal an Arduino senden, um die
10                    # Messung
11                    # sofort abubrechen
```

```

10
11 def reset_timer():
12     canvas.itemconfig(time_label, text="00:00:00.000")
13
14 \vspace{0.5cm} %

```

Start-Timer: Sendet ein Signal an den Arduino, um die Messung zu starten und setzt den Timer zurück.

Stop-Timer: Stoppt die laufende Zeitmessung.

Reset-Timer: Setzt die Timer-Anzeige auf den Anfangswert zurück.

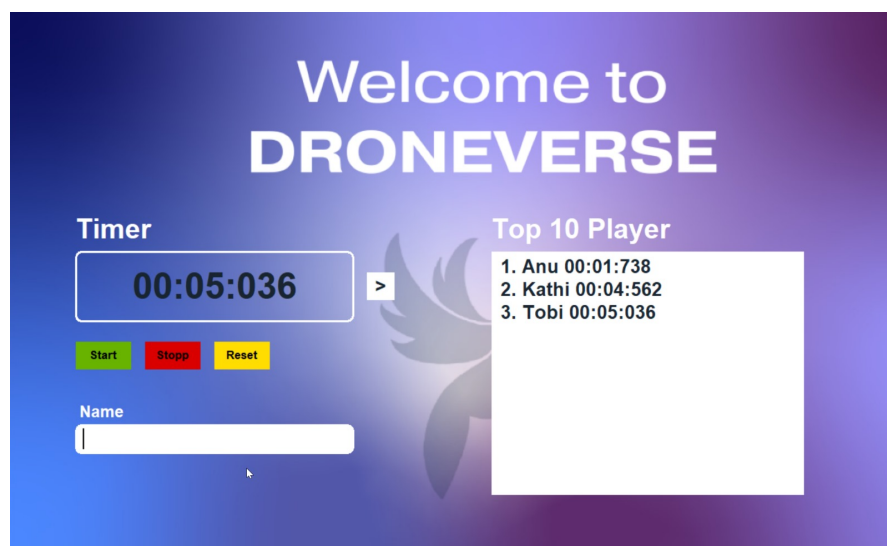


Abbildung 2.28: GUI-Top 10 List

Fazit

Die Entwicklung dieser GUI zur Steuerung und Anzeige der Drohnenflugzeiten erfüllte alle gestellten Anforderungen. Durch die serielle Kommunikation mit dem Arduino wurde eine Echtzeit-Überwachung der Flugzeiten ermöglicht, und die Top-10-Liste bot den Benutzern einen zusätzlichen Anreiz, ihre besten Zeiten zu erreichen und zu speichern. Die intuitive Benutzeroberfläche sorgte dafür, dass die Anwendung einfach zu bedienen war und gleichzeitig die Funktionalität und Ästhetik gewährleistete.

2.5.9 Integration eines Timers basierend auf der Ausgabe eines Sensors

(Katharina Hölzl)

2.5.9.1 Herausforderung

Für den Hindernis-Parcours sollte ein Punktesystem entwickelt werden, das eine Gamification-Komponente integriert. Die Punkte sollten gemessen werden, indem die Drohne bestimmte Hindernisse passiert und dabei Sensoren auslöst, welche die Punkte zählen und auf einen Bildschirm übertragen. Eine logische Lösung wären LDRs (Light Dependent Resistors) oder Fototransistoren, da der Einsatz einer Kamera, welche die Punkte zählt, zusätzlichen Installationsaufwand bedeutete und latenzbehaftet wäre. Das Ziel war es, die Punkte möglichst in Echtzeit zu zählen.

2.5.9.2 Recherche zu den Sensoren Fototransistor und LDR (Light Dependent Resistor)

Es stellte sich die Frage, welcher Sensor am effektivsten für die Messung von Punkten verwendet werden könnte. Folgende Ergebnisse entstanden bei der Recherche zu Fototransistoren und LDR (Light Dependent Resistor):

Fototransistoren bieten eine höhere Empfindlichkeit als Photodioden, da sie gleichzeitig als Verstärker fungieren. Sie werden häufig verwendet, um Licht zu erkennen oder für Lichtübertragungen, wie bei Lichtschranken, Dämmerungsschaltern und Optokopplern. In Fernbedienungen werden hingegen Photodioden verwendet, da Fototransistoren dafür zu langsam sind. Diese Geräte, sowie Lichtschranken und Optokoppler, arbeiten mit Infrarotstrahlen und nicht mit sichtbarem Licht. Ein Fototransistor, reagiert auf Lichtintensität. In einer Lichtschranke bildet er das Gegenstück zur Linsenlampe. Unzufrieden mit dem Ergebnis dieser Recherche und dem Einsatzbereich der beiden Sensoren, wurde auf Anraten von Laboringenieur Daniel Baer im Bereich Abstandssensor weiter recherchiert.

2.5.9.3 Recherche zu einem Abstandssensor

Konkret wurde über den SR04-Ultraschallsensor recherchiert, welcher ein Modul zur Entfernungsmessung mit vielfältigen Anwendungsmöglichkeiten ist. Das Modul sendet einen Ultraschallimpuls aus und misst die Zeit, bis dieser zurückkehrt, um die Entfernung zu berechnen.

Die Vorteile des HC-SR04 bestehen in seiner hohen Genauigkeit und Zuverlässigkeit, das heißt er kann Entfernungen bis zu einer Genauigkeit von 3 mm messen. Der Sensor punktet durch eine einfache Bedienung und einer programmierbaren Verzögerungszeit und einem einstellbaren Messbereich. Er ist kompatibel mit verschiedenen Mikrocontrollern wie dem Raspberry Pi oder dem verwendeten Arduino MKR1000. Zusätzlich ist er ideal durch seine geringe Größe von 45 mm x 20 mm x 15 mm und leichtem Gewicht von 9 Gramm. Der Sensor wird mit 5 V Spannung versorgt, hat eine Stromaufnahme von 15 mA und arbeitet in einem Messbereich von 2 cm bis 400 cm mit einer Frequenz von 40 kHz.

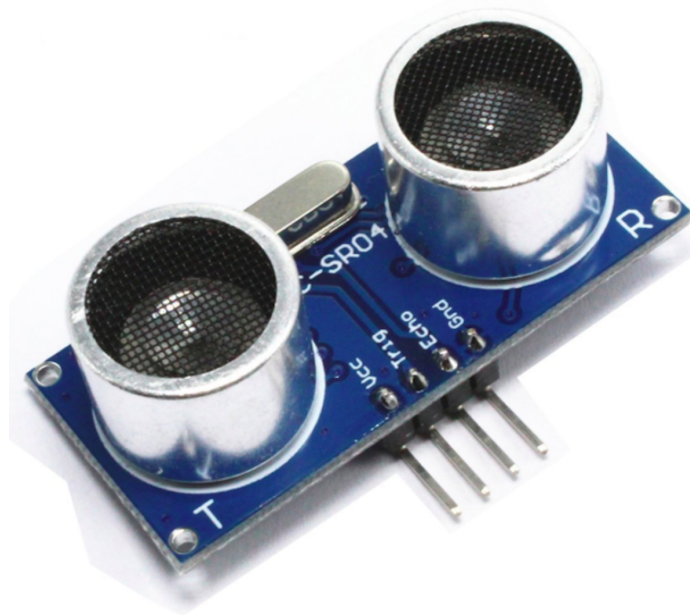


Abbildung 2.29: Ultraschallsensor HC-SR04

2.5.9.4 Ergebnis der Recherche

Aufgrund der durchgeführten Recherche und den Anforderungen wurde entschieden, dass der Abstandssensor HC-SR04 für das Projekt am sinnvollsten ist. Zudem hat sich die Gruppe darauf geeinigt, das Spiel nicht anhand von Punkten zu bewerten, sondern die Einzelzeiten der Spieler zu stoppen. Dadurch werden die Ergebnisse genauer, da bei einem Punktesystem mehrere Spieler dieselbe Punktzahl erreichen könnten. Mit dem zeitbasierten System kann man die Leistung der Spieler besser bewerten und in die Top-10-Wertung einfließen lassen.

2.5.9.5 Ultraschallsensor HC-SR04

Da man sich für den Abstandssensor entschied, wurde von der Hochschule ein Arduino Kit ausgeliehen mit einem SR04-Sensor, in dem Kit befindlich war auch ein Arduino MKR1000: Der Arduino MKR1000 ist ein Mikrocontroller-Board, das speziell für IoT-Anwendungen entwickelt wurde. Das Board bietet eine Vielzahl von Ein- und Ausgängen sowie Analog- und Digitalschnittstellen, die die Steuerung und Überwachung von Sensoren und Aktoren ermöglichen. Mit der kompakten Größe und geringen Stromaufnahme ist der Arduino MKR1000 ideal für das Projekt geeignet. Zudem unterstützt es die Arduino-IDE, was die Programmierung und Integration in bestehende Arduino-Projekte vereinfacht.

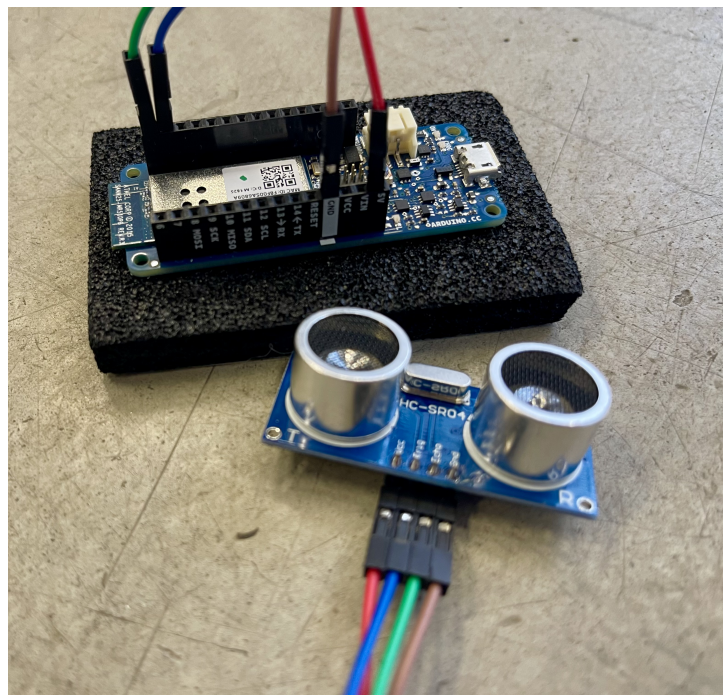


Abbildung 2.30: Der verbundene Sensor mit dem Arduino

Folgender Code wurde verwendet, um den Arduino MKR1000 zu programmieren:

Listing 2.34: Definition von Variablen

```
1 long duration;  
2 int distance;  
3  
4 const int trigPin = 4;  
5 const int echoPin = 5;  
6 const int minDistance = 5; # Minimale Distanz in cm, bei  
    der der Timer gestartet werden soll  
7 const int maxDistance = 45; # Maximale Distanz in cm, bei  
    der der Timer gestoppt werden soll
```

Der Arduino-Code beginnt mit der Definition von Variablen und Konstanten für den Ultraschallsensor. `long duration` speichert die Dauer des Echsignals, `int distance` die berechnete Entfernung, und `trigPin` sowie `echoPin` sind auf die Pins 4 und 5 gesetzt. `minDistance` und `maxDistance` legen den Messbereich des Sensors fest.

Listing 2.35: Definition von Variablen und Konstanten für den Ultraschallsensor

```
1 unsigned long startTime;  
2 unsigned long stopTime; # Variable zur Zeitspeicherung, wenn  
    der Timer gestoppt wird  
3 bool timerRunning = false;  
4 bool measurementStarted = false; # Flag, ob die Messung  
    gestartet wurde  
5 bool initialWaitDone = false; # Flag, um die initiale  
    Wartezeit zu steuern  
6 unsigned long waitStartTime;
```

Der Code beginnt mit der Definition von Variablen und Konstanten für den Ultraschallsensor. `long duration` speichert die Dauer des Echsignals, `int distance` die berechnete Entfernung, und `trigPin` sowie `echoPin` sind auf die Pins 4 und 5 gesetzt. `minDistance` und `maxDistance` legen den Messbereich des Sensors fest.

Listing 2.36: Empfang von Daten der seriellen Schnittstelle

```

1 void loop() {
2   if (Serial.available() > 0) {
3     char input = Serial.read();
4     if (input == 's' && !measurementStarted) {
5       measurementStarted = true;
6       initialWaitDone = false; # Reset initial wait
7     }
8     if (input == 'p' && timerRunning) {
9       stopTimer();
10      measurementStarted = false;
11    }
12  }

```

Die `loop()`-Funktion überprüft, ob Daten über die serielle Schnittstelle empfangen wurden. Bei Empfang des Zeichens “s” wird die Messung gestartet, bei “p” wird der Timer gestoppt, dies garantiert eine richtige Messung im Falle eines Absturzes der Drohnen. Ebenfalls wurde die Eingabe des “p” dazu gedacht im Falle eines Notfalles oder Unfalles mit der Drohne, den Timer sofort zu stoppen. Wenn die Messung aktiv ist und der Timer nicht läuft, wird die Entfernung gemessen, indem der Trigger-Pin kurz auf HIGH gesetzt wird und das Echosignal gemessen wird. Liegt die Entfernung innerhalb des definierten Bereichs von 5 cm bis 45 cm, wird der Timer gestartet.

Listing 2.37: Arduino Code für den Sensor

```

1   if (measurementStarted && !timerRunning) {
2     // Abstand messen
3     digitalWrite(trigPin, LOW);
4     delayMicroseconds(2);
5     digitalWrite(trigPin, HIGH);
6     delayMicroseconds(10);
7     digitalWrite(trigPin, LOW);
8
9     duration = pulseIn(echoPin, HIGH);
10    distance = duration * 0.034 / 2;
11
12    if (distance >= minDistance && distance <= maxDistance)
        {

```

```

13     startTimer();
14     waitStartTime = millis(); # Start the wait time
15 }
16 }
17
18 if (timerRunning) {
19     // Zeit messen und anzeigen
20     printElapsedTime();
21
22     // Initiale 10 Sekunden Wartezeit
23     if (!initialWaitDone && millis() - waitStartTime >=
24         20000) {
25         initialWaitDone = true;
26     }
27
28     #Abstand messen, aber nur nach der initialen Wartezeit
29     if (initialWaitDone) {
30         digitalWrite(trigPin, LOW);
31         delayMicroseconds(2);
32         digitalWrite(trigPin, HIGH);
33         delayMicroseconds(10);
34         digitalWrite(trigPin, LOW);
35
36         duration = pulseIn(echoPin, HIGH);
37         distance = duration * 0.034 / 2;
38
39         if (distance >= minDistance && distance <= maxDistance
40             ) {
41             stopTimer();
42             measurementStarted = false;
43         }
44     }
45
46     delay(100);
47 }

```

Während der Timer läuft, wird die verstrichene Zeit mit `printElapsedTime()` angezeigt. Nach einer Wartezeit von 20 Sekunden wird die Entfernung erneut

gemessen. Wenn sie weiterhin im Bereich liegt, wird der Timer gestoppt und `measurementStarted` auf `false` gesetzt. Dies ermöglicht es, bei der Steuerung der Drohnen, genügend Zeit zum starten zu haben, ohne dass der Timer zu früh stoppt.

Listing 2.38: Timerfunktion

```
1
2 void startTimer() {
3     startTime = millis();
4     timerRunning = true;
5 }
6
7 void stopTimer() {
8     stopTime = millis(); # Speichern der Zeit bei Timer-Stopp
9     timerRunning = false;
10    printElapsedTime(); # Zeit seit dem Timer-Start anzeigen
11 }
12
13 void printElapsedTime() {
14     unsigned long currentTime = timerRunning ? millis() :
15         stopTime;
16     unsigned long elapsedTime = currentTime - startTime;
17
18     unsigned long minutes = (elapsedTime / 60000) % 60;
19     unsigned long seconds = (elapsedTime / 1000) % 60;
20     unsigned long milliseconds = elapsedTime % 1000;
21
22     Serial.print(minutes);
23     Serial.print(":");
24     Serial.print(seconds);
25     Serial.print(".");
26     Serial.println(milliseconds);
27 }
```

Die `startTimer()`-Funktion speichert die Startzeit und setzt `timerRunning()` auf `true`. Die `stopTimer()`-Funktion speichert die Endzeit, setzt `timerRunning` auf `false` und zeigt die verstrichene Zeit an. Die `printElapsedTime()`-Funktion berechnet und gibt die verstrichene Zeit in Minuten, Sekunden und Millisekunden aus.

2.5.9.6 Konstruktion des Startblocks

Um den drohnenbegeisterten Besuchern eine Orientierungshilfe zu geben, wo sie die Drohne starten und landen dürfen, wurde ein Startblock entworfen. Der Startblock besteht aus einer dünnen Spanplatte mit den folgenden Maßen:

- Breite: 44 cm
- Länge: 52,5 cm

Um das Projekt nachhaltiger zu gestalten, wurde die Spanplatte, aus dem Reste-Container der Labor-Ingenieure herausgesucht, sie hatte bereits passende Maße und musste daher nicht weiter zugeschnitten werden. Die Platte wurde zunächst mit in der Mitte durchbohrt, um sie an einem Sockel zu befestigen. Anschließend mit weißem Sprühlack grundiert und Acryl-Farbe beschichtet. Die Kanten wurden mit Neon-Tape abgeklebt und der Schriftzug "Start Stop" in schwarzer Acrylfarbe aufgetragen. Seitlich der Platte wurde der Abstandssensor befestigt, und die Kabel, welche zum Laptop führen, wurden unter der Platte fixiert.



Abbildung 2.31: Spanplatte



Abbildung 2.32: Beschriftung der Platte

2.5.10 Raumgestaltung

(Katharina Hölzl)

Für die Endpräsentation wurde der Projektgruppe der Raum BB.008 zur Verfügung gestellt. Dieser Raum wurde in drei Bereiche unterteilt: Parcours, Zuschauerbereich und technischer Bereich.

Der Raum bietet ausreichend Platz, um viele Zuschauer unterzubringen, unsere Technik aufzubauen und einen großen Parcours zu errichten. Außerdem verfügt der Raum über elektrische Rollläden, welche ihn verdunkeln und die gewünschte Atmosphäre schaffen. Die Atmosphäre erinnert an die Dunkelheit einer Höhle, während die Hindernisse im Parcours kunstvoll durch gezielt platzierte LEDs in Szene gesetzt werden.

Im Eingangsbereich wurde eine kleine Statue aus Ballons gebaut und ebenfalls mit blauen und pinken LED-Par-Scheinwerfern beleuchtet. Weiße Ballons waren an der Decken-Traverse befestigt und wurden ebenfalls mit LED-Par-Scheinwerfern beleuchtet, was einen tollen Effekt ergab. Die restlichen Ballons wurden für die Quadkopter-Konstruktion aufgehoben und verwendet.

Ein großer Bildschirm stand zur Verfügung, auf dem die Präsentation gezeigt wurde und später, während der Vorführung der Drohnen, das Spielinterface abläuft. Das Sicherheitsnetz ist in L-Form an der Traverse befestigt. Am kurzen Ende des Netzes stehen Tische mit Laptops für Gestenerkennung, Sensorsteuerung und die Akkuladestation. Dieses Netz besteht aus transparentem Monofilamentgarn, wodurch es leicht durchsichtig ist und den Blick auf die Drohnenshow kaum beeinträchtigt.

Der Raum war abgedunkelt, sodass die Hindernisse, die teilweise mit LEDs bestückt sind, besser zur Geltung kommen. Das Sicherheitsnetz wurde mit Tape auf dem Boden befestigt, was der Sicherheit dient und einen schnellen Auf- und Abbau garantiert. Schreibtische und Technik, welche nicht aus dem Raum entfernt werden durfte, wurde mit einem schwarzen Vorhangstoff abgehängt.

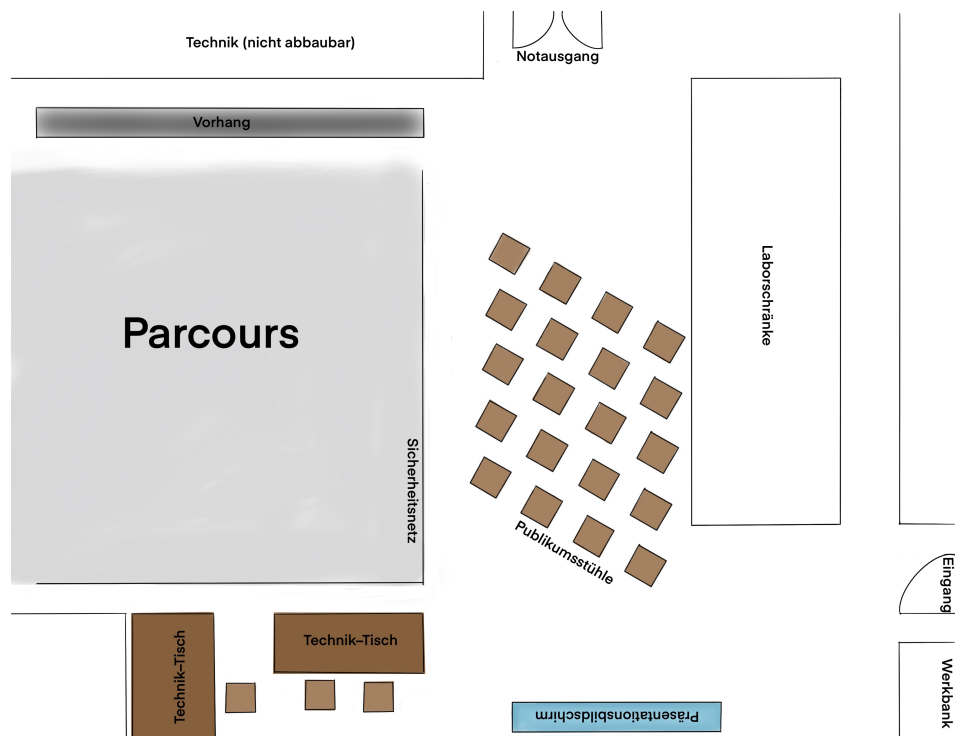


Abbildung 2.33: Skizze der Projektraumgestaltung

2.5.10.1 Plakate

Um den Besuchern des Projekts die Einführung in die Steuerung der Drohnen zu erleichtern, wurden für jede Steuerungsmethode ein Plakate entworfen. Jedes Plakat zeigt die einzelnen Steuerbefehle auf und die dazugehörigen Aktionen, welche erforderlich sind, um die Drohnen zu steuern. Am Präsentationstag wurden die Plakate gut sichtbar ausgelegt.



Abbildung 2.34: Plakat für die Handgestensteuerung [2.3.1](#)

DRONEVERSE

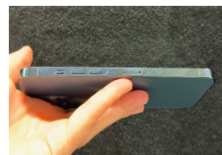
Drohnen-Steuerung per OSC



Grundhaltung



**Smartphone zu einem kippen
= Hoch (Höhe)**



**Smartphone weg kippen
= Runter (Höhe)**



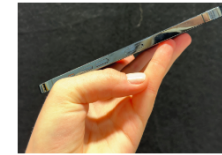
**Smartphone um max. 45° nach
links drehen
= Rückwärts**



**Smartphone um max. 45°
nach rechts drehen
= Vorwärts**



**Smartphone nach rechts
kippen = Rechts**



**Smartphone nach links
kippen = Links**

Abbildung 2.35: Plakat für die OSC Steuerung per Smartphone [2.3.2](#)

DRONEVERSE

**Drohnen-Steuerung
per Tastatur**

W

nach vorne fliegen

A

**nach links
fliegen**

S

nach hinten fliegen

D

**nach rechts
fliegen**



Pfeiltaste nach oben fliegen (Höhe)



Pfeiltaste nach unten fliegen (Höhe)

Q

**Abbruch
des Flugs**

Abbildung 2.36: Plakat für die Tastatursteuerung [2.3.3](#)

Kapitel 3

Design Integration

(Anusha Kanagarasa)

Im Rahmen des Moduls lag der Fokus nicht nur auf der kreativen Gestaltung des Projekts, sondern auch auf der Entwicklung einer umfassenden Marketingstrategie. Dazu gehörte die Erstellung eines kurzen Teaser-Videos, das die wichtigsten Aspekte des Projekts auf ansprechende Weise präsentiert. Ein sorgfältig erstelltes Drehbuch diente als Leitfaden für die Videoproduktion, um sicherzustellen, dass alle wesentlichen Inhalte klar und überzeugend vermittelt werden. Das Drehbuch für das **Teaser-Video** folgt im Anschluss.

3.1 Marketingstrategie

Ziele

1. Interaktivität und Engagement fördern:

Das Ziel ist es, ein Drohnenerlebnis zu schaffen, das die Interaktion der Teilnehmer verstärkt und ihre aktive Einbindung in das Projekt fördert. Dabei soll die Begeisterung der Teilnehmer gesteigert werden. Gleichzeitig wird angestrebt, DRO-NEVERSE als führende Plattform für interaktive Drohnenspiele und Events zu etablieren. Der Fokus liegt auf dem Aufbau einer langfristigen Kundenbindung und der Steigerung der Bekanntheit in der Zielgruppe.

2. Auffällige Werbeflächen schaffen:

Ein weiteres Ziel ist es, durch kreative und innovative Präsentationen neue Partnerschaften zu gewinnen und bestehende Kooperationen zu stärken.

Zielgruppe

- Technikbegeisterte und Drohnenfans.
- Jugendliche und junge Erwachsene, die Spaß an interaktiven Spielen haben.
- Unternehmen, die innovative Werbemöglichkeiten suchen.

Wettbewerbsvorteil

- Einzigartiges Steuerungssystem für Drohnen: Nutzung von Handgesten, Tastatur oder OSC.
- Visuell ansprechende Präsentation: Dekoration des Raums mit bunten Ballons und LED-Lichtern am Parkours.
- Flexible Einsatzmöglichkeiten für verschiedene Events und Werbezwecke.

Branding

Unser Branding basiert auf einer markanten visuellen Identität, die durch die Verwendung von leuchtenden Farben wie Blau und Lila die futuristische und künstlerische Natur des Projekts unterstreicht.

Moderne und klare Schriftarten verstärken das professionelle und innovative Image von DRONEVERSE. Die Konsistenz im visuellen Stil sorgt für Wiedererkennung und stärkt die Marke.

Das Kolibri-Logo ist ein zentrales Element, das durch seine Symbolik für Drohnen und Technologie sofort ins Auge fällt und die Marke unverwechselbar macht.

Posting-Frequenz

Wöchentlich:

- 2-3 reguläre Posts mit:

- Highlights
- Drohnenbildern
- Ankündigungen.

Posting-Timeline:

- Wochenende: Sonntags posten für bessere Reichweite.

Interaktion mit Followern

Antworten auf Kommentare:

- Schnell und freundlich auf Kommentare reagieren, um Engagement zu fördern.

Aufforderung zu Likes und Shares:

- Follower ermutigen, Beiträge zu liken und zu teilen (z.B. „Gefällt dir dieses Drohnenmanöver? Zeig uns dein Like!“).

Engagement mit Hashtags:

- Teilnahme an relevanten Diskussionen und Nutzung von beliebten Hashtags wie #DroneVerse, #DrohnenKunst, #gesture, #steuerung, #fun, #parkour.

Strategien zur Erhöhung der Follower-Zahl

Behind-the-Scenes Inhalte:

- Exklusive Einblicke hinter die Kulissen Ihrer Projekte zeigen, um eine tiefere Verbindung zu Ihren Followern aufzubauen.
 - Beispiel: Videos und Fotos von der Planung und dem Aufbau Ihrer Drohnenparkours teilen.

Hashtags nutzen:

- Relevante und populäre Hashtags verwenden, um die Sichtbarkeit Ihrer Beiträge zu erhöhen und ein breiteres Publikum zu erreichen.
 - Beispiel: Spezifische und trendige Hashtags wie #DroneVerse, #TechArt, #DrohnenLiebhaber nutzen.

Zusammengefasste Marketingstrategie

Unsere Social-Media-Strategie fokussiert sich auf Instagram, da Beiträge direkt zu Facebook geteilt werden können. Dies ermöglicht eine größere Reichweite, da Inhalte auf beiden Plattformen erscheinen. Es wurde ein Geschäftskonto als “Content Creator“ erstellt, um Einblicke in die Statistiken zu erhalten und die Interaktionen zu analysieren.

Mit Stand 12.08.2024 verzeichnen wir 59 Follower, mit 15 geposteten Beiträgen, 1.736 Seitenaufrufen, wovon 60,2 Prozent Follower waren und 39,8 Prozent Nicht-Follower. 90 Nutzerinnen und Nutzer haben mit dem Profil interagiert und es wurden 4 Highlights erstellt in welchen auch diverse Story-Beiträge abgespeichert wurden. Die Vorausproduktion von Content ermöglicht es uns, Beiträge zur optimalen Zeit zu veröffentlichen und eine maximale Reichweite zu erzielen. Tägliche Story-Beiträge sorgen dafür, dass die Marke kontinuierlich im Gedächtnis unserer Follower bleiben.

Durch aktive Interaktion mit anderen Content Creators und relevanten Accounts wird die Präsenz gestärkt und der Instagram-Algorithmus optimal genutzt. Durch qualitativ hochwertige Aufnahmen und gezielte Hashtags wird die Sichtbarkeit der Beiträge erhöht und neue Follower gewonnen.

3.2 Produktion von Foto- und Videomaterialien

(Lisa Berbig)

Um das Projekt DRONEVERSE adäquat nach außen präsentieren zu können, wurden verschiedene **Foto- und Videomaterialien** produziert. Der Großteil davon geschah im Rahmen des Moduls Design Integration. Auch außerhalb der Aufgabenstellungen dieses Moduls wurden ein kurzes Teaser Video und Fotos der Drohne produziert, welche unter anderem für das Pressematerial, Social-Media oder auch das Userinterface des Programms zur Drohnensteuerung verwendet wurden.

Hierbei wurde besonders darauf geachtet, die in Design Integration entwickelte Designsprache bestmöglich umzusetzen. Es wurde mit dunklen Hintergründen gearbeitet, sowie mit farbigem Licht in Blau- und Magentatönen, welche so auch im Designkonzept von DRONEVERSE definiert sind.

Diese hochwertig produzierten Fotos und Videos verleihen dem Projekt ein einheitliches und professionelles Erscheinungsbild. Sie gewähren den Zuschauern einen Einblick in das, was sie beim Besuch von DRONEVERSE erwartet, verraten aber auch noch nicht zu viel, um Neugierde zu erzeugen.

3.2.1 Drehbuch

(Lisa Berbig)

INNEN. DRONEVERSE PARKOUR - [Tageszeit nicht relevant] (Spannende, energetische Musik im Hintergrund. Wenn nicht anders angegeben dunkler/schwarzer Hintergrund, Beleuchtung in magenta und blau)

1 Bild ist schwarz. Die einzelnen LEDs des LED Bands vom Reifen leuchten nacheinander auf. Das Bild faded wieder ins Schwarz. Der Leuchtende Reifen ist im ganzen zu sehen. Vor dem Reifen befindet sich ein Netz. Bild faded wieder zu Black. Kamerabewegung durch die Reifen des Hindernisparkours. Sehr nahe Detailaufnahme des leuchtenden LED Streifens im Reifen. Zwei Personen stehen sich duellierend gegenüber. Nur die Arme und Hände sind zu sehen. PERSON A ist magenta beleuchtet, PERSON B ist blau beleuchtet. Aufnahme der Start/Stopp Plattform, magenta und blau beleuchtet. Nahaufnahme der Drohne auf der Plattform, selbe Beleuchtung und Plattform dreht sich, Kamera bleib auf der Stelle. Drohne steht auf der Plattform, startet und fliegt oben aus dem Bild hinaus.

Bild ist schwarz, Drohne fliegt von unten in das Bild. Drohne ist magenta und blau beleuchtet. Im Hintergrund fliegen Ballons.

2 Nahaufnahme von der Hand von Person A, wie sie eine Faust macht. Die Hand ist magenta beleuchtet. Aufnahme der Drohne vor schwarzem Hintergrund wie sie nach vorne fliegt. Nahaufnahme von der flachen Hand von Person B. Die Hand ist blau beleuchtet. Aufnahme der Drohne vor schwarzem Hintergrund wie sie nach hinten fliegt. Drohne fliegt auch den Reifen des Parkours. Drohne schwebt in der Luft vor Schwarzem Hintergrund, hinter der Drohne fliegen Ballons.

Perspektive der Drohne: Drohne fliegt schwankend durch die Reifen.

3 Schluss-Screen: Droneverse - Informationen zu Ort und Datum der Präsentation

3.3 Werbematerial

(Anusha Kanagarasa)

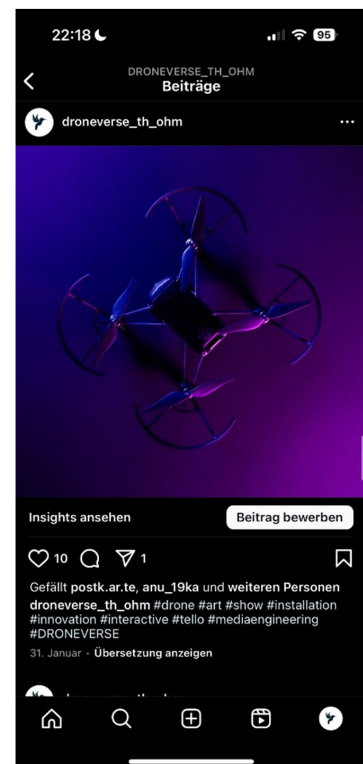
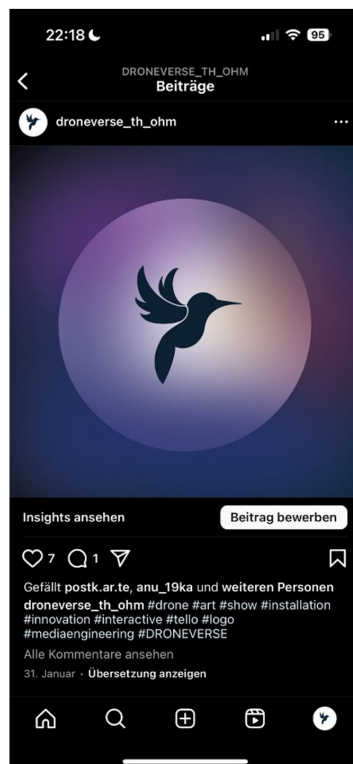
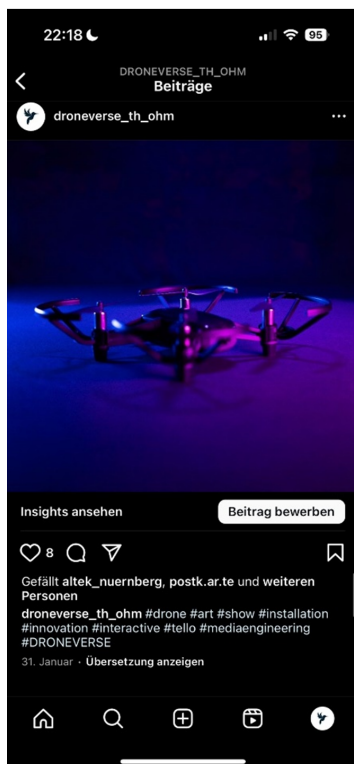
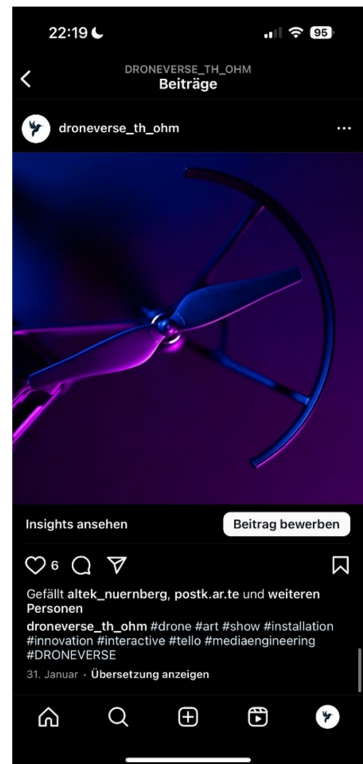
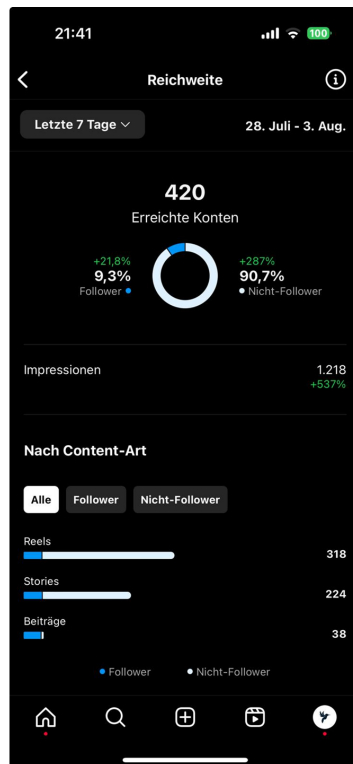
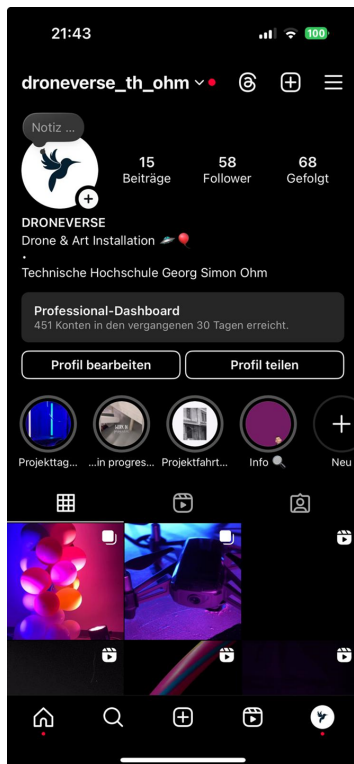
Während des Projekts wurde Instagram als Hauptkanal genutzt, um mit der Außenwelt zu kommunizieren und unsere Follower auf dem Laufenden zu halten.

Durch Instagram-Posts, Stories und Reels wurden die verschiedenen Phasen des Projekts geteilt – von den ersten Ideen bis zu den finalen Ergebnissen. Mit Bildern, Videos und kurzen Updates konnte man die Community direkt in den Entwicklungsprozess einbinden.

Besonders während des Projekttags wurden kontinuierlich Update-Stories gepostet, um neugierige Follower und Interessierte in Echtzeit über die Aktivitäten zu informieren. Diese regelmäßigen Updates gaben Einblicke hinter die Kulissen und sorgten dafür, dass die Community stets auf dem neuesten Stand bleibt.

Der Instagram-Account folgt einem konsistenten visuellen Thema, das durch dunkle Hintergründe und einen lila-blauen Farbton geprägt ist. Diese einheitliche Ästhetik verleiht den Posts einen futuristischen und technologischen Look, der perfekt zu dem Drohnenprojekt passt. Die gezielte Verwendung dieser Farben und Beleuchtung schafft eine einnehmende Atmosphäre, die die Aufmerksamkeit der Betrachter auf sich zieht und gleichzeitig eine professionelle und konsistente Markenidentität vermittelt.

Hier findet man einen Überblick über einige unserer Instagram-Beiträge:



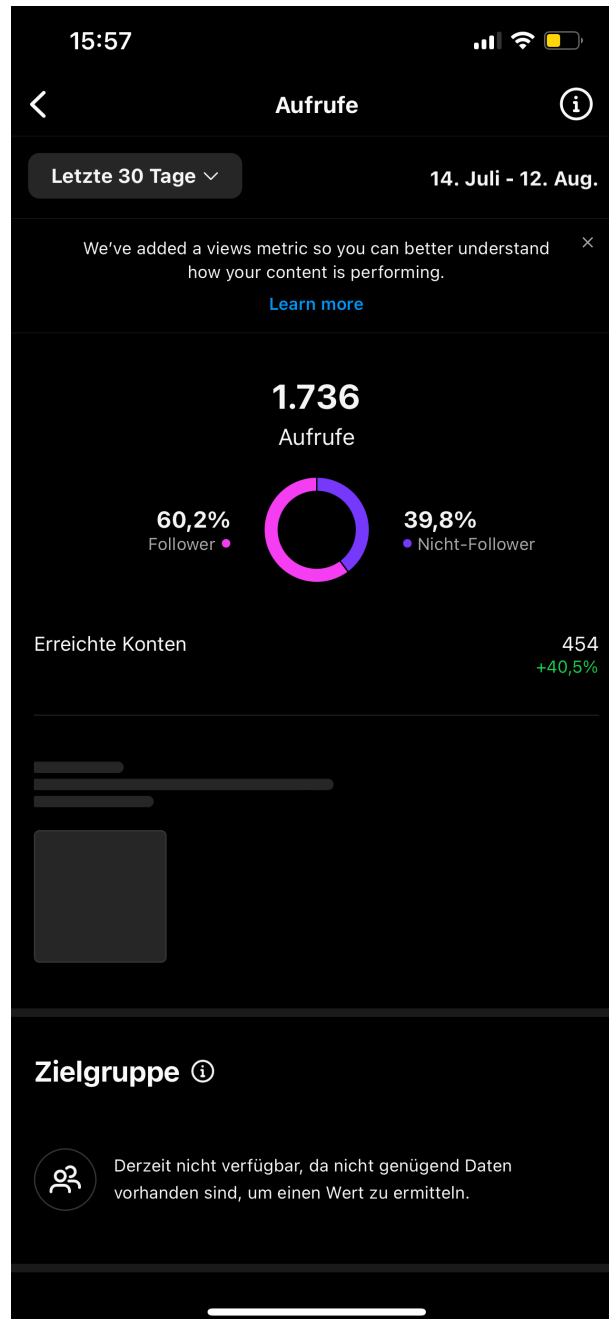


Abbildung 3.1: Statistik der Seitenaufrufe

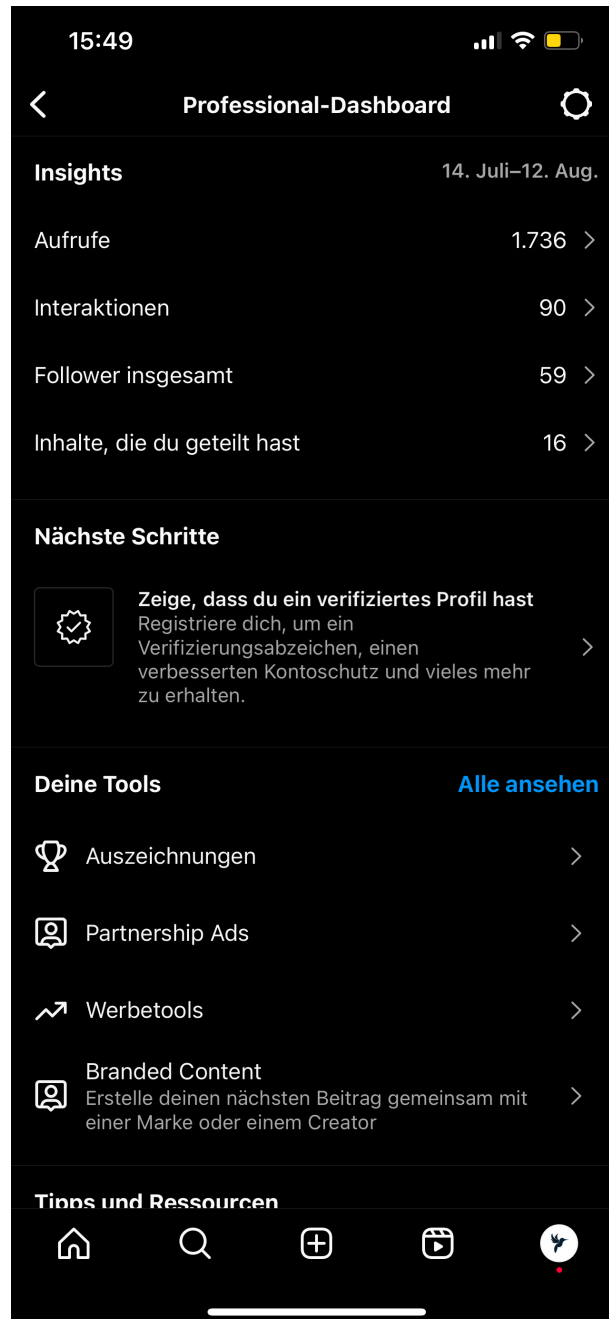


Abbildung 3.2: Instagram Professional-Dashboard

3.4 Webseite

(Katharina Hölzl)

Für die Abgabe dieses Berichtes wurde von den verantwortlichen Professoren auch eine Webseite gefordert, welche einen Einblick in das Projekt gibt. Die Webseite wurde mit Wordpress erstellt und beinhaltet die Hauptseite und das Impressum, sowie Links zu unserem Social Media Kanal auf Instagram. WordPress ist eine kostenlose Plattform zur Erstellung und Verwaltung von Websites und Blogs. Die Webseite beginnt mit einem modernen und schlicht gehaltenem Header, welcher die Besucher herzlichst auf unsere Seite einlädt und Neugierde weckt. Danach wird das Projekt kurz erläutert, wer wir sind und was wir das vergangene Jahr über entwickelt haben. Da kein Kapitel des Projektes ohne das andere existieren würde, hat jedes einen eigenen Kurzbeitrag mit repräsentativem Bild erhalten, in dem verständlich erklärt wird, was im Laufe des Projektes geschehen ist. Abschließend können Besucher der Webseite in der Bildergalerie weitere Eindrücke des Projektes einsehen.

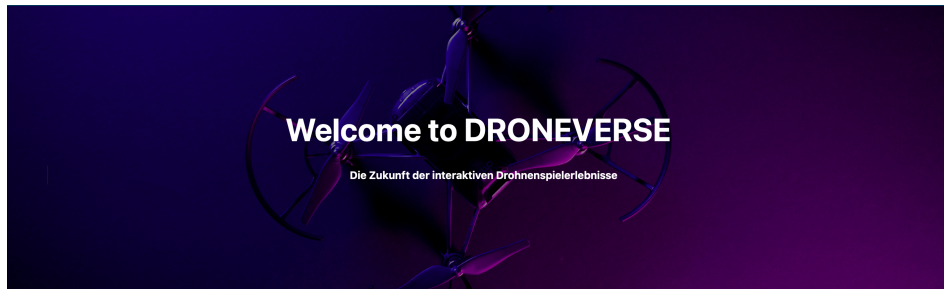


Abbildung 3.3: Welcome to DRONEVERSE



Wer sind wir?

Wir sind DRONEVERSE, ein innovatives Drohnen-Spiel entwickelt von Media Engineering-Studierenden der TH OHM Nürnberg. In diesem Spiel steuern Sie eine Drohne durch einen anspruchsvollen Parcours – sei es durch Gestensteuerung, Steuerung per Smartphone oder Tastatureingaben. Wählen Sie zwischen einer Tello-Drohne und einem von Heliumballons getragenen Quadkopter.

Zu Beginn des Projekts lag der Fokus auf der Forschung zur interaktiven und nutzerfreundlichen Drohnensteuerung. Nach umfassender Analyse und Feedback entwickelten wir das Ziel, eine Drohne per Gestensteuerung sicher durch den Parcours zu navigieren. Die finale Präsentation markierte den Höhepunkt unseres Projekts.

Erleben Sie die Entstehung von DRONEVERSE, vom ersten Konzept bis zur Umsetzung, und entdecken Sie, wie wir Technik mit spannendem Gameplay vereinen konnten.

Abbildung 3.4: Wer sind wir?

Gestenerkennung:

Die Gestenerkennung nutzt ein Python-Modul, das mit OpenCV und Mediapipe Handgesten in Echtzeit erkennt. Das Webcambild wird analysiert, um sechs Richtungen (vorwärts, rückwärts, hoch, runter, rechts, links) zu steuern. Gesten werden durch spezifische Fingerpositionen erkannt. Ein statisches Hintergrundbild ersetzt das Kamerabild, um Ablenkungen zu minimieren.

OSC-Steuerung:

Die Drohnensteuerung per iPhone erfolgt über die App „Data OSC“, die Gyroskopdaten des Smartphones in Echtzeit an den Computer überträgt. Ein Python-Skript liest diese Daten, um die Bewegungsrichtung des Gerätes zu bestimmen. Eine Benutzeroberfläche zeigt die erkannten Richtungen an. Die Steuerung erfolgt durch Kippen des Smartphones, wobei die Ausrichtung entscheidend ist.

Tastatursteuerung:

Zusätzlich wurde eine Tastatursteuerung für die Drohne implementiert. Die Steuerung erfolgt durch KeyDown- und KeyUp-Events, die in einer Schleife überwacht werden. Bei Tastendruck wird die entsprechende Richtung an die Drohne übermittelt, während das Loslassen der Taste die Drohne stoppt. Diese Methode bietet eine präzise und einfache Steuerungsalternative und integriert sich nahtlos in die Software.

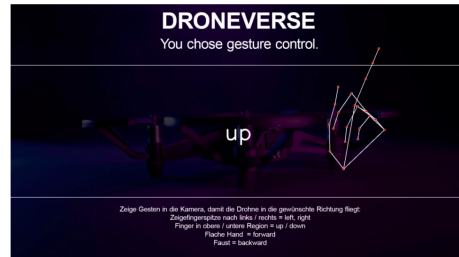


Abbildung 3.5: verschiedene Arten der Drohnensteuerung



Kollisionserkennung

Die Steuerung einer Drohne per Gesten ist oft ungewohnt und kann dazu führen, dass die Drohne den Flugbereich verlässt. Um Schäden zu vermeiden, wurde eine Kollisionserkennung entwickelt, die verhindert, dass die Drohne den Parcoursbereich verlässt. Ein Differenzbild wird zur Positionsbestimmung verwendet. Dabei werden Videostream-Frames einer Kamera unter dem Parcours verglichen, um die Drohnenposition zu ermitteln.

Die Erkennung basiert auf weißen Pixeln in einem binärem Differenzbild, die anzeigen, ob die Drohne einen Randbereich erreicht hat. Bei Gefahr greift das System ein und blockiert die Bewegung in diese Richtung.

Eine 360°-Kamera erfasst den gesamten Parcours, wobei das Bild auf den relevanten Bereich zugeschnitten wurde.

Abbildung 3.6: Kollisionserkennung

Innovative Steuerung von Drohnen und Quadcopter

Für die Tello-Drohne entwickelten wir eine geschwindigkeitsbasierte Steuerungsmethode, die durch eine eigens erstellte Klasse namens „TelloDroneController“ präzise Richtungsanpassungen ermöglicht. Die Initialisierung der Drohne erfolgt über die DJITelloPy-Bibliothek, wobei eine übergeordnete Geschwindigkeitsvariable die Steuerung flexibel gestaltet. Trotz deaktivierter Drehfunktion in dunklen Räumen ist die Steuerung umfassend implementiert, inklusive einer sicheren Landemethode.

Neben der Tello-Drohne haben wir auch zwei Cheerson CX-10 Mini-Quadcopter integriert. Ursprünglich planten wir, eine Funkverbindung mit Transceiver-Modulen, doch entschieden uns zur einheitlichen Implementierung für einen ESP32. Damit gelang schließlich die Verbindung des Quadcopters mit einem Laptop, was eine direkte Steuerung ermöglichte. Durch die Implementierung einer Telnet-Verbindung und die Integration in unsere Projektcodebasis wurde die finale Steuerung erfolgreich getestet und realisiert.

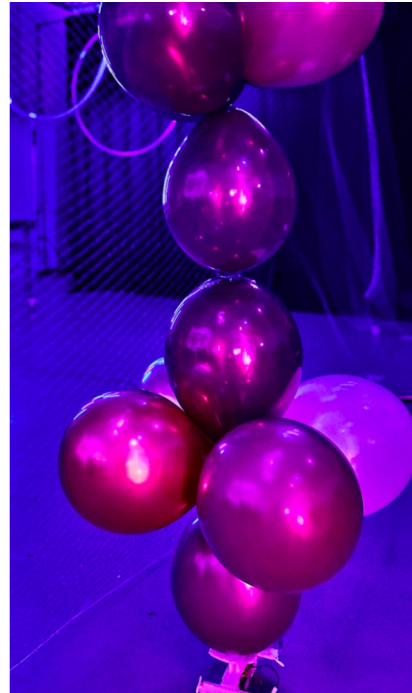


Abbildung 3.7: Innovative Steuerung von Drohnen und Quadcopter



Interaktiver Hindernisparcours: Design & Aufbau

Der interaktive Hindernisparcours ist der Höhepunkt unseres Projekts DRONEVERSE. Von der ersten Idee bis zum Aufbau entstand eine zugängliche Umgebung, die per Gestensteuerung navigierbar ist und die entwickelten Technologien in einer spannenden Praxisanwendung erlebbar macht.

Planung & Konstruktion: Der Raum wurde vermessen und in SketchUp ein erster Entwurf erstellt. Nach einer Überarbeitung entstand eine klare Struktur, die eine gute Balance zwischen Herausforderung und Zugänglichkeit für Anfänger bot.

Konstruktion mit Shapr3D: Mit Shapr3D modellierten wir die Hindernisse präzise und passten sie an die Umgebung an. Die Software erleichterte die Detailplanung und den Export für weitere Anwendungen.

Aufbau des Parcours: Der finale Parcours wurde mit einfachen Materialien wie Hula-Hoop-Reifen und Schwimnudeln im Raum BB008 aufgebaut. LED-Säulen, Metallrahmen und steuerbare LED-Streifen sorgten für visuelle Herausforderungen und stimmungsvolle Beleuchtung.

Abbildung 3.8: Interaktiver Hindernis-Parcours: Design und Aufbau

Sensor und Timer für den Hindernisparcours

Für unseren Hindernisparcours war ein präzises System zur Zeitmessung entscheidend. Nach Überlegungen zwischen einem Punktesystem und einer zeitbasierten Bewertung entschieden wir uns für Letzteres, um die Ergebnisse vergleichbar und exakt zu machen.

Ultraschall-Abstandssensor HC-SR04: Wir wählten den HC-SR04-Sensor aufgrund seiner hohen Messgenauigkeit und seiner einfachen Integration in unser Arduino-System.

Programmierung und Tests: Mit einem Arduino MKR1000 entwickelten wir einen Timer, der die Flugzeit der Drohne automatisch erfasst. Der Timer startet, wenn die Drohne den Startpunkt verlässt, und stoppt, wenn sie das Ziel erreicht, wodurch wir eine präzise Messung sicherstellen.

Aufbau und Befestigung: Der HC-SR04-Sensor wurde auf einer lackierten Spanplatte am Startblock des Parcours montiert, um Stabilität und Genauigkeit zu gewährleisten.

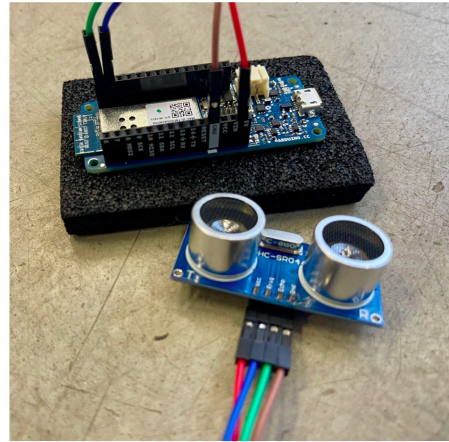
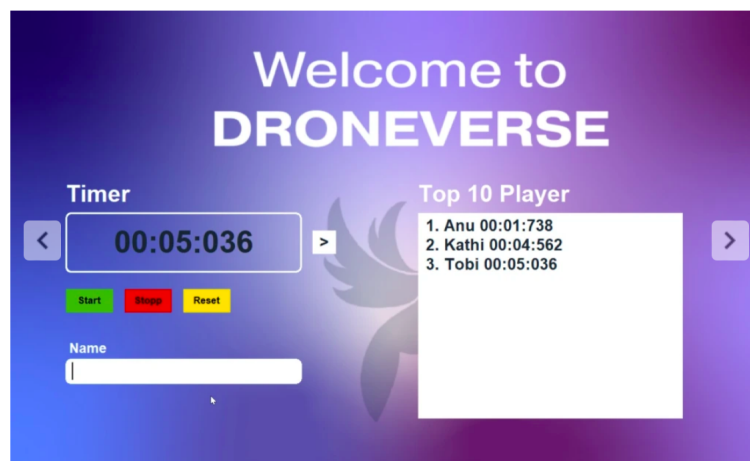


Abbildung 3.9: Sensor und Timer für den Hindernis-Parcours

Impressionen



27 / 28

Abbildung 3.10: Impressionen Slideshow

Kapitel 4

Corporate Identity

(Anusha Kanagarasa)

4.1 Markenidentität

4.1.1 USP (Alleinstellungsmerkmal)

DRONEVERSE hebt sich durch seine innovative Verschmelzung von vielseitigen Steuerungsoptionen und Drohnentypen von anderen Drohnensimulationen ab und bietet ein einzigartiges Spielerlebnis. Im Gegensatz zu herkömmlichen Spielen dieser Art, die oft auf eine einzige Steuerungsmethode und Drohnenkonfiguration beschränkt sind, ermöglicht DRONEVERSE den Spielern, zwischen drei unterschiedlichen Steuerungsoptionen zu wählen: Gestenerkennung, Handyausrichtungserkennung (OSC) und Tastatureingabe. Dies gibt den Nutzern die Flexibilität, die Steuerung der Drohne an ihre persönlichen Vorlieben und Fähigkeiten anzupassen.

Zusätzlich können die Spieler zwischen zwei einzigartigen Drohnentypen wählen: einer präzisen Tellodrohne für detaillierte Steuerung und einem innovativen Quadkopter, der mit Heliumballons ausgestattet ist, um eine besondere Flugdynamik zu bieten. Diese Auswahl an Drohnen und Steuerungsmethoden ergänzt sich in einem anspruchsvollen Hindernisparcours, der speziell entwickelt wurde, um die Fähigkeiten der Spieler auf die Probe zu stellen und gleichzeitig ein fesselndes Gameplay zu bieten. Der Parcours umfasst drei Hauptaufgaben – das Durchfliegen von Ringen, das Navigieren um LED-Säulen und ein Salto durch ein zweigeteiltes Hindernis – und fordert präzises Steuerungsgeschick und strategisches Timing.

Durch diese flexible Kombination von Steuerungsoptionen, Drohrentypen und einem herausfordernden Parcours bietet DRONEVERSE ein besonders anpassungsfähiges und immersives Spielerlebnis, das sowohl Einsteiger als auch erfahrene Drohnenpiloten begeistert. Diese Einzigartigkeit macht DRONEVERSE zu einem herausragenden Beispiel für innovative Drohnensimulation und interaktive Unterhaltung.

4.1.2 Zielgruppe

1. Technologie- und Drohnen-Enthusiasten: Diese Gruppe besteht aus Individuen, die ein starkes Interesse an neuen Technologien und Drohnen haben. Sie sind oft technikaffin, experimentierfreudig und schätzen innovative Ansätze im Bereich der Drohnensimulation und -steuerung. Für sie bietet DRONEVERSE eine spannende Plattform, um ihre Leidenschaft für Drohnen auszuleben und ihre Fähigkeiten in einem simulierten, aber realitätsnahen Umfeld zu testen.

2. Gamer und Simulationsliebhaber: Diese Zielgruppe umfasst Personen, die sich für Computerspiele und Simulationen begeistern. Besonders interessiert sind sie an Spielen, die eine hohe Interaktivität und Realitätsnähe bieten. DRONEVERSE spricht diese Gruppe durch seine anspruchsvollen Hindernisparcours und die Möglichkeit, zwischen verschiedenen Steuerungsoptionen zu wählen, an. Das Spiel bietet eine interessante Alternative zu traditionellen Games und zieht insbesondere diejenigen an, die eine neue Herausforderung suchen.

3. Bildungseinrichtungen und Lernende: Schulen, Universitäten und Bildungseinrichtungen, die sich mit Technologie, Ingenieurwesen oder Informatik befassen, könnten DRONEVERSE als pädagogisches Werkzeug nutzen. Das Spiel bietet praktische Erfahrungen im Umgang mit Drohnen und Steuerungstechnologien, was es zu einem nützlichen Tool für Bildungszwecke macht. Lernende und Studierende können durch das Spiel nicht nur ihre praktischen Fähigkeiten verbessern, sondern auch ihr Wissen über die technische Funktionsweise von Drohnen erweitern.

4.2 Das Logo

4.2.1 Aussage

Name: DRONEVERSE

Logo: modernes, minimalistisches Logo, das Technologie und Kunst symbolisiert.

Farbschema: kühle Farbtöne wie Blau- und Lilatöne, um die Atmosphäre einer mit Luftballons ausgeschmückten Höhle zu betonen.

Das Logo dient als zentrales Element, um die Marke an verschiedenen Berührungspunkten erkennbar zu machen. Es soll einprägsam und wiedererkennbar sein und gezielt die Aufmerksamkeit der Zielgruppe auf sich ziehen.

Motivation hinter dem Kolibri als Wappentier:

- Ähnliche Flugeigenschaften: Der Kolibri symbolisiert die fliegerischen Fähigkeiten, welche auch die Drohne beherrscht. Der Kolibri ist in der Lage, in der Luft zu schweben und blitzschnell durch die Umgebung zu manövrieren. Dies spiegelt die Agilität und das schnelle, präzise Flugverhalten der Drohnen wider.
- Eleganz und Anmut: Der Kolibri verleiht der Wort-Bild-Marke eine natürliche Eleganz und Anmut. Diese Eigenschaften unterstreichen die ästhetische Dimension des Projekts und heben es von rein technischen Darstellungen ab.

Die Wahl des Kolibris als Wappentier verleiht der Marke nicht nur eine visuelle Identität, sondern verbindet auch die fliegerischen Eigenschaften dieses faszinierenden Vogels mit den einzigartigen Merkmalen der Drohnenshow.



DRONEVERSE



DRONEVERSE

Darf nur allein (ohne
Wortmarke) stehen,
wenn DRONEVERSE
anderweitig auf der
Seite genannt ist.

Darf nur nach
Absprache mit dem
Brand Management
alleine stehen.

Abbildung 4.1: Droneverse-Logo

4.2.2 Größenangabe

Das Logo darf nicht kleiner als 20mm verwendet werden. Empfohlene Logohöhe bei verschiedenen DIN-Formaten:

- DIN A5 = Höhe ca. 35 mm
- DIN A4 = Höhe ca. 50 mm
- DIN A3 = Höhe ca. 80 mm
- DIN A2 = Höhe ca. 120 mm usw.

4.3 Farbsystem

4.3.1 Farbverhältnis

Die Farbpalette von DRONEVERSE wurde sorgfältig ausgewählt, um eine einladende und gleichzeitig futuristische Atmosphäre zu schaffen. Dominiert von kühlem Blau- und Violetttönen, erzeugen die Farben eine Aura von Technologie und Innovation, die optimal mit dem Thema des Projekts harmonisiert. Diese Farbtöne symbolisieren die faszinierende Welt der Drohnenshows und Kunstinstallationen, in der sich die Besucher auf eine Reise voller Entdeckungen begeben.

Durch die Ergänzung von tieferen Nuancen entsteht ein visueller Kontrast, der dem Farbschema Tiefe und mehrere Dimensionen verleiht. Diese tiefen Töne dienen dazu, die visuelle Wirkung zu verstärken, während sie gleichzeitig eine elegante und ansprechende Ästhetik bewahren.

Die lebendigen Blautöne in der Palette verleihen dem Gesamtbild eine erfrischende Note. Sie bringen Leben und Bewegung in das Projekt und erwecken die Drohnenshow zum Leben. Diese Farben repräsentieren die kreative und inspirierende Atmosphäre von DRONEVERSE.

Insgesamt schafft die geschickte Kombination dieser Farbtöne eine harmonische und ansprechende visuelle Umgebung, die das Projekt DRONEVERSE in Szene setzt und das Publikum in eine faszinierende Welt voller Technologie, Kunst und Unterhaltung entführt.

4.3.2 Farbpalette

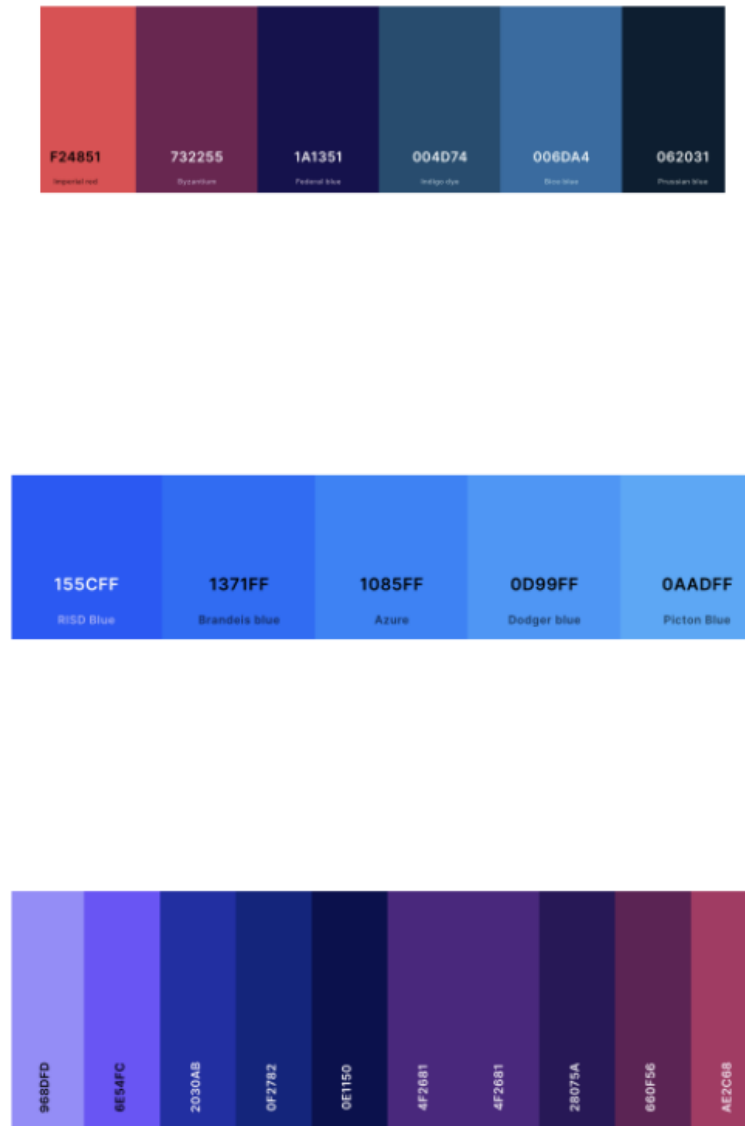


Abbildung 4.2: Corporate Design Farben

4.4 Typografie

4.4.1 Schrift



Abbildung 4.3: Schriftart

4.4.2 Verwendung

Für eine klare und verständliche Kommunikation wurde eine serifenlose Schriftart verwendet mit klar strukturierten Paragraphen. Dies gilt sowohl für die Plakate als auch für die Social-Media-Inhalte, um eine breite Zugänglichkeit sicherzustellen.

für Logo, Plakat, Fyler:

- Pragmatica Extended Bold für Website, Social Media, Fließtext:
- Avenir Light (Fließtext)
- Avenir Medium (Überschriften)
- Avenir Heavy (Hervorhebung wichtiger Informationen)

Kapitel 5

Zusammenfassung

(Sina Steinmüller)

Der lange Weg, der zurückgelegt wurde, um aus dem Namen DRONEVERSE das innovative und einzigartige Spiel zu machen, das bereits das Marketingkonzept im Zwischenbericht versprochen hatte, hat sich definitiv gelohnt. Statt einer gewöhnlichen Abschlusspräsentation konnte ein Erlebnis geschaffen werden, das weit über die ursprünglichen Erwartungen hinausging. Unser Ziel, eine Drohne durch einen Hindernis-Parcours zu steuern, wurde nicht nur erreicht, sondern auch übertroffen.

Dank der stabilen Netzwerkverbindung (2.2) war es möglich, die Tello-Drohne (2.3.5) nahezu nahtlos durch eine andere zu ersetzen, um den Nutzern ununterbrochenen Spielspaß zu bieten. Die anfängliche Sorge der Spieler, die Drohne durch die ungewohnte Gestensteuerung (2.3.1) beschädigen zu können, wurde durch die integrierte Kollisionsvermeidung (2.3.4) erfolgreich gemindert. Mit den beiden zur Verfügung stehenden Drohnen, der Tello und dem Quadcopter (2.3.6), die beide mit derselben Software gesteuert werden konnten, kam keine Langeweile auf. Die beiden weiteren Steuerungsmethoden, die Handysteuering (2.3.2) sowie die Tastatursteuerung (2.3.3), sorgten für noch mehr Spaß und Experimentierfreude.

Zuletzt trugen unser durchdachtes Marketingkonzept (4), das zu der der kreativen Raumgestaltung und des Parcours (2.5) führte, dazu bei, dass das Projekt DRONEVERSE ein stimmiges Gesamtpaket wurde, das die Zuschauer in ein anderes Universum entführte. Der Präsentationsraum war dauerhaft mit Besuchern gefüllt, die alle DRONEVERSE erleben oder spielen wollten. Trotz einiger Ideen, die aufgrund begrenzter Zeit und Ressourcen nicht umgesetzt werden konnten, war dieses Projektjahr eine wertvolle Lernerfahrung für das gesamte Team, bei dem jeder die Möglichkeit hatte, über sich hinaus zu wachsen.

Wir sind glücklich darüber, dass Professor Lano dieses Projekt ins Leben gerufen hat und wir die Chance hatten, dieses so frei und kreativ umsetzen zu können, wie wir es letztlich auch getan haben. Danke!

Abbildungsverzeichnis

2.1	Softwarearchitektur Abbildung	12
2.2	User Interface	13
2.3	Geste Hoch	17
2.4	Geste Runter	17
2.5	Geste Vorwärts	18
2.6	Geste Rückwärts	18
2.7	Geste Links	19
2.8	Geste Rechts	19
2.9	Screenshots der DataOSC iPhone App	23
2.10	OSC Nachrichten	24
2.11	OSC Hoch	27
2.12	OSC Runter	27
2.13	OSC Vorwärts	27
2.14	OSC Rückwärts	27
2.15	OSC Links	28
2.16	OSC Rechts	28
2.17	Bild der 360° Kamera	37
2.18	Luftschiff bei Zwischenpräsentationen	45
2.19	ESP32 WROOM (links) und Burning Fixture Tool (rechts)	47
2.20	Screenshot des PuTTY Clients	50
2.21	ESP mit Transistoren und Akkus auf Breadboard	53
2.22	Lochplatine mit Motoren und grüne Kabel für ESP	53
2.23	Luftschiff bei Abschlusspräsentation	54
2.24	Erster Entwurf mit SketchUp	57
2.25	Zweiter Entwurf des Parcours auf Shapr3D	59
2.26	Finale Umsetzung	62
2.27	Hintergrund für das Game-GUI	63
2.28	GUI-Top 10 List	68
2.29	Ultraschallsensor HC-SR04	70

2.30	Der verbundene Sensor mit dem Arduino	72
2.31	Spanplatte	78
2.32	Beschriftung der Platte	79
2.33	Skizze der Projektraumgestaltung	81
2.34	Plakat für die Handgestensteuerung 2.3.1	82
2.35	Plakat für die OSC Steuerung per Smartphone 2.3.2	83
2.36	Plakat für die Tastatursteuerung 2.3.3	84
3.1	Statistik der Seitenaufrufe	93
3.2	Instagram Professional-Dashboard	94
3.3	Welcome to DRONEVERSE	96
3.4	Wer sind wir?	96
3.5	verschiedene Arten der Drohnensteuerung	97
3.6	Kollisionserkennung	97
3.7	Innovative Steuerung von Drohnen und Quadcopter	98
3.8	Interaktiver Hindernis-Parcours: Design und Aufbau	98
3.9	Sensor und Timer für den Hindernis-Parcours	99
3.10	Impressionen Slideshow	99
4.1	Droneverse-Logo	103
4.2	Corporate Design Farben	105
4.3	Schriftart	106

Listings

2.1	Vereinfachter Aufbau aller Detection Module	13
2.2	Verkürzter Auszug aus der main.py	14
2.3	Erkennung von oben und unten mittels Regionen	17
2.4	Erkennung von vorwärts und Rückwärts durch Distanzmessung	18
2.5	Erkennung von rechts und links durch Zeigefinderspitzenposition	20
2.6	Definition der IP Adresse und des Ports	22
2.7	Hilfsskript zum Herausfinden des OSC Datenpfades	23
2.8	Starten des OSC Servers	24
2.9	Sammeln von Gyroskopdaten	25
2.10	Ermittlung der Durchschnittswerte	25
2.11	Richtungsbestimmung mittels Schwellwert	25
2.12	Initialisierung Tastaturerkennung	30
2.13	Initialisierung Tastaturerkennung	31
2.14	Initialisierung Tastaturerkennung	32
2.15	Berechnung eines binären Differenzbildes	34
2.16	Erkennung von weißen Pixeln in den Randbereichen	34
2.17	Aufruf von sendCollision.py in CollisionDetection.py	35
2.18	Initialisierung TelloDroneControl	40
2.19	TelloDroneControl Übergabe-Methode	41
2.20	TelloDroneControl Richtungs-Methoden	42
2.21	TelloDroneControl Lande-Methode	42
2.22	Telnet Richtungsanweisung	43
2.23	Telnet Initialisierung	43
2.24	HMTL Code auf ESP32	47
2.25	Ausführung der HTML Befehle auf ESP	48
2.26	Erkennung Taste 'W' auf ESP32	49
2.27	pygame Code Ausschnitt	51
2.28	Benutzeroberfläche	64
2.29	Anzeige des Timers	65

2.30	Serielle Kommunikation mit Arduino	65
2.31	Top 10 Liste	66
2.32	Speicherung der Namen & Zeiten	67
2.33	Steuerung des Timers	67
2.34	Definition von Variablen	73
2.35	Definition von Variablen und Konstanten für den Ultraschallsensor . . .	73
2.36	Empfang von Daten der seriellen Schnittstelle	74
2.37	Arduino Code für den Sensor	74
2.38	Timerfunktion	76

Bibliographie

- [1] Google AI. *MediaPipe Hands Dokumentation*. Zuletzt aktualisiert: 2024-05-21. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=de.
- [2] Jakob Löw Damià Fuentes Escoté. *DJITelloPy*. 2018. URL: <https://github.com/damiafuentes/DJITelloPy>.
- [3] Jakob Löw Damià Fuentes Escoté. *DJITelloPy Dokumentation*. 2018. URL: <https://djittelopy.readthedocs.io/en/latest/tello/>.
- [4] René Dudfield. *pygame git*. 2023. URL: <https://github.com/pygame/pygame>.
- [5] Anton Heestand. *Data OSC iPhone App*. 2023. URL: <https://apps.apple.com/de/app/data-osc/id6447833736>.
- [6] Anton Heestand. *DataOSC App Entwicklerwebsite*. Aufgerufen am 13.08.2024. URL: <http://heestand.xyz>.
- [7] Stefan Luber. *Was ist Telnet?* 2018. URL: <https://www.ip-insider.de/was-ist-telnet-a-691219/>.
- [8] *Python Dokumentation*. 2020. URL: <https://wiki.python.org/moin/GlobalInterpreterLock>.

ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen aus dem Internet.

Diejenigen Paragraphen der für mich gültigen Prüfungsordnung, welche etwaige Betrugsversuche betreffen, habe ich zur Kenntnis genommen.

Der Speicherung meiner Projekt-Arbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

M. Richter
L. Berbig
Tobias Schwarz
Katharina Hölzl
Jana Jemmel
A. Managarasa

13.08.2024

(Ort, Datum)

Unterschrift des Verfassers