20FS_IMVS08: Tello Simulator

Implementierung eines Simulators für die Tello Drohne

Fachhochschule Nordwestschweiz
Hochschule für Technik

IP5 im Studiengang Informatik mit Vertiefung in Design und Management an der Fachhochschule Nordwestschweiz
v0.1
Brugg, 6. Oktober 2020

Autoren
Daniel Obrist
Severin Peyer

Auftraggeber
Fachhochschule Nordwestschweiz FHNW
vertreten durch Dieter Holz und Barbara Scheuner
Bahnhofstrasse 6
5210 Windisch

Betreuer
Dr. Dieter Holz
Dr. Barbara Scheuner

Inhaltsverzeichnis

Management Summary

An der Fachhochschule Nordwestschweiz (FHNW) wird im Rahmen des Workshop-Moduls 2 im Studiengang Informatik iCompetence die Drohne «Tello EDU» von Ryze Robotics [1] (nachfolgend: Tello-Drohne) eingesetzt. Bei der Entwicklung von Applikationen zur Steuerung der Drohne steht pro Team mit vier Mitgliedern nur eine Drohne zur Verfügung. Um diesem Flaschenhals entgegen zu wirken und um Missgeschicke mit der echten Drohne vorzubeugen hat die FHNW einen Simulator in Auftrag gegeben, welcher das Verhalten der Drohne nachahmt.

Dabei soll keine Hardware implementiert werden, um nicht eine zusätzliche Abhängigkeit zu erzeugen. Der Simulator soll die gleichen Befehle wie die Tello-Drohne per UDP-Verbindung entgegennehmen und diese möglichst realitätsgetreu in einer einfachen 3D-Welt abbilden.

Der Simulator wurde mit dem JavaFX-Framework (basierend auf Java) umgesetzt. Dieser beinhaltet eine 3D-Szene, in welcher ein quaderförmiger skalierbarer Raum und ein 3D-Modell der Drohne ersichtlich sind. Ebenfalls sind Steuerungselemente für den Simulator, die wichtigsten Parameter für die Drohne sowie einige Parameter für die Verbindung zwischen dem Simulator und dem Steuerungsprogramm vorhanden. Zusätzlich beinhaltet das Layout ein Log, welches das Testen des Codes vereinfacht. Im Hintergrund werden die empfangenen Kommandos und ihre Parameter auf Korrektheit überprüft und das Verhalten fürs User Interface generiert. Die Lösung erlaubt auf einfache Art und Weise eine Steuerungssoftware für die Tello-Drohne zu testen, ohne dabei physische Hilfsmittel zu benötigen.

Der Simulator konnte so weit entwickelt werden, dass wir empfehlen, diesen im Rahmen des Moduls zu verwenden. Es macht Sinn, die einzelnen Kommandos nochmals durchzutesten und das Verhalten der Tello-Drohne und des Simulators zu vergleichen. So kann sichergestellt werden, dass die Drohnensteuerungsapplikation nach dem Testen mit dem Simulator auch die physische Drohne korrekt steuern kann. Mithilfe dieses Berichts kann der Simulator weiterentwickelt und die noch fehelenden Funktionalitäten implementiert werden.

Einleitung

Ausgangslage

Im Workshop-Modul 2 ws2C (Programmieren von Minidrohnen und «intelligente Kleidung») der Fachhochschule Nordwestschweiz (FHNW) wird die Tello EDU von Ryze Robotics [1] (nachfolgend: Tello-Drohne) eingesetzt. Dabei werden in Teams von durchschnittlich vier Mitgliedern Applikationen zur Steuerung der Drohne entwickelt. Jedem Team steht dabei eine Tello-Drohne zur Verfügung.

Problem

Es hat sich gezeigt, dass die Drohne sehr schnell zu einem Engpass führt, da insbesondere im frühen Entwicklungsstadium viel mit der Drohne experimentiert werden muss. Dadurch verzögert sich der Projektfortschritt. Des Weiteren ist es auch bereits zu mehreren Missgeschicken mit der Drohne gekommen. Beispielsweise wurde vergessen ein «stop»/«land»-Signal einzubauen, weswegen die Drohne in eine Wand geflogen und kaputt gegangen ist.

Anforderungen

Um die beiden Probleme zu minimieren, respektive eliminieren, soll ein Simulator entwickelt werden, welcher sich möglichst gleich verhält wie die echte Tello-Drohne. Dieser soll rein softwaretechnisch umgesetzt werden, damit nicht wieder die Abhängigkeit von einem technischen Gerät entsteht. Der Simulator muss die Programmierschnittstelle des Tello SDK 2.0 anbieten. Folgende Anforderungen wurden definiert:

- Der Simulator kann Kommandos des Clients entgegennehmen und eine Response senden
- Die virtuelle Drohne setzt alle Control-, Read- und Set-Commands visuell realitätsgetreu um
- Der Simulator kann einen Status auf einem separaten Port an den Client senden
- Der Simulator kann einen Videostream aus der Sicht der Drohne an den Client senden
- Error-Handling (ungültige Befehle, ungültige Parameter)

Für das Extended Product, haben wurden folgende Kann-Ziele definiert:

- Per UI kann eine Raumgrösse eingegeben werden (LxBxH) und eine Startposition der Drohne (x, y und Start am Boden)
- Der Simulator kann mit mehreren Drohnen (Drohnenschwärme) umgehen

Lösung

Die umgesetzte Lösung für das Problem ist ein einfacher Simulator, basierend auf JavaFX. Die Funktionen sind zu grossen Teilen identisch mit der Tello-Drohne. Die Verbindung und der Datenaustausch zum Simulator ist, analog zur Drohne, über UDP gelöst.

Zielpublikum

Der Simulator dient Studierenden, welche mit der Tello-Drohne arbeiten und die Funktionalitäten der physischen Drohne zuerst softwaretechnisch testen möchten. Im Speziellen wurde er für das Modul ws2C an der FHNW entwickelt. Gemäss der Modulbeschreibung [2] beherrschen die Studierenden die Grundkonzepte der objektorientierten Programmierung.

Hauptteil

Architektur

Die Software-Architektur des Simulators ist geprägt durch JavaFX-Komponenten im Frontend. Die Logik im Backend besteht hauptsächlich aus den UDP-Schnittstellen und dem Handling der Commands. Die nachfolgenden zwei Kapitel geben einen Überblick der Applikations-Struktur und beschreiben die wichtigsten Klassen.

Frontend

Das User-Interface des Simulators ist mit den typischen JavaFX-Bausteinen aufgebaut. Die Benutzerfreundlichkeit wurde bei der Gestaltung nicht speziell beachtet und hat bestimmt noch ein gewisses Verbesserungspotenzial. Der Fokus des Projekts lag jedoch auf den Grundfunktionalitäten, weswegen nicht viele Ressourcen in die Usability investiert wurden.

SimulatorPane

Die SimulatorPane ist die übergeordnete BorderPane, welche alle anderen JavaFX-Nodes enthält. Links befinden sich die SimulatorControls zum Setzen und Beobachten von Simulator- und Drohnen-Parametern. Auf der rechten Seite sind die NetworkControls, welche alle nötigen Informationen zum Verbindungsaufbau mit dem Simulator anzeigen. Unten findet der User eine interaktive LogBox, welche dem Debugging dient. Und in der Mitte befindet sich die Simulator3DScene, in welcher die 3D-Welt und die virtuelle Drohne gerendert werden.

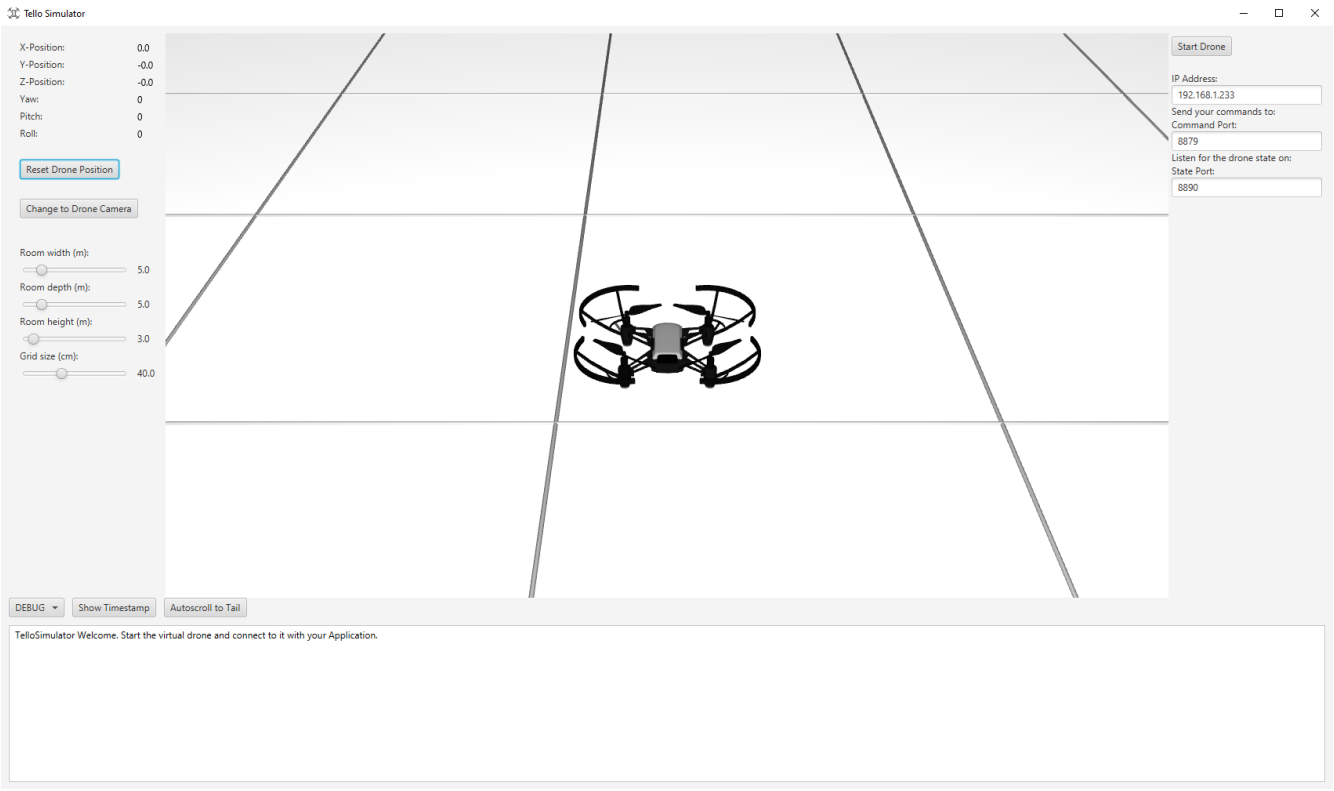

Abbildung 1. Das User Interface zum Stand des finalen Releases mit SimulatorControls (links), NetworkControls (rechts), LogBox (unten) und Simulator3DScene (mitte).

SimulatorControls

Diese Komponente zeigt die wichtigsten Parameter der Drohne an. Dazu gehören die **X-, Y- und Z-Position** sowie die **Yaw-, Pitch- und Roll-Werte**. Zusätzlich steht ein **Reset-Button** zur Verfügung, wodurch alle Werte der virtuellen Drohne zurückgesetzt werden. Mit dem Button darunter kann der User ausserdem zwischen **Simulator- und Drohnenkamera** hin und her wechseln. Ebenfalls befinden sich hier vier Slider zur Konfiguration der Grösse des virtuellen Raums. Ist ein Slider angewählt, können die Werte auch mit den Pfeiltasten eingestellt werden.

NetworkControls

Die NetworkControls auf der rechten Seite beinhalten zuoberst den **Start Drone-Button**, mit welchem die virtuelle Drohne ein- und ausgeschaltet werden kann. Dieser Button repräsentiert in der Funktionsweise den On-Off-Schalter der echten Tello-Drohne. Denn erst nachdem die Tello-Drohne eingeschaltet wurde kann man sich mit ihr verbinden. Analog muss auch die virtuelle Drohne zuerst gestartet werden. Nach dem Betätigen des Start Drone-Buttons baut der Simulator die CommandConnection auf und beginnt auf dem entsprechenden Port Kommandos (Command) zu empfangen. Des Weiteren wird eine entsprechende Nachricht in das Log geschrieben, um dem User zu zeigen, dass die Drohne auf Commands wartet.

Unterhalb des Start-Buttons befinden sich Informationen zum Verbindungsaufbau mit dem Simulator. Im Feld **IP Address** steht bei aktiver Internetverbindung die IP-Adresse des Geräts, auf welchem der Simulator gerade läuft. Wenn keine Internetverbindung besteht, kann der Simulator diese Adresse leider nicht ermitteln. Dann steht in diesem Feld standardmässig die Loopback-Adresse 127.0.0.1.

Im Feld **Command Port** wird die Port-Nummer angezeigt, auf welcher der Simulator seinen DatagramSocket zum Empfangen von Commands erstellt hat. Hierhin müssen also von einem Client-Programm die Commands geschickt werden.

Das nächste Feld **State Port** gibt den Port an, mit welchem sich die StateConnection des Simulators für das Versenden des Drohnen-Status verbindet. Auf diesem Port sollte man also den Drohnen-Status empfangen.

LogBox

Die LogBox an der Unterseite ist ein mächtiges Tool zum Debuggen. Hier werden sozusagen alle Aktivitäten des Simulators geloggt. Die verschiedenen **Log-Level** sind dabei farblich unterschiedlich dargestellt. Das Log-Level lässt sich je nach Bedarf einstellen und die ListView wird entsprechend gefiltert. Ebenfalls lässt sich mit **Show Timestamp** ein Zeitstempel ein- und ausblenden. Der Button **Autoscroll to Tail** scrollt automatisch immer nach unten zum neusten Log-Eintrag.

Simulator3DScene

Im Mittelpunkt des UI steht die Simulator3DScene, auf welcher die ganze 3D-Welt inklusive virtueller Drohne gerendert wird. Hierzu verwendet der Simulator eine JavaFX-SubScene, welche in der umschliessenden BorderPane im Zentrum platziert ist. Neben der 3D-Welt und der Drohne befinden sich zwei Kameras in dem SceneGraph der Subscene: einerseits die vom User kontrollierbare SimulatorCamera sowie die an die Drohne fixierte DroneCamera.

Die **DroneView**-Klasse ist die Repräsentation der Drohne im 3D-Raum als 3D-Modell. Ihr Modell wird mittels einem FXML Source File geladen, inklusive animierten Rotoren. Die Position und Rotation der DroneView sind dabei einseitig an die entsprechenden Properties des DroneModels im Backend gebunden. Das heisst wenn sich im DroneModel was ändert, wird dies durch die DroneView in der 3D-Welt abgebildet.

Damit der User das Verhalten der virtuellen Drohne optimal beobachten kann, lässt sich die SimulatorCamera mit der linken Maustaste drehen. Der Pivot-Punkt, um welchen sich die Kamera dreht, transformiert sich dabei gleichmässig mit der Drohne. So fliegt die Drohne nicht plötzlich aus dem Sichtfeld. Ebenfalls lässt sich die Kamera mit der rechten Maustaste oder durch Drücken des Mausrads nach links und rechts verschieben, um eine andere Perspektive zu erhalten. Die Zoom-Distanz der Kamera lässt sich durch das Scrollen mit dem Mausrad oder dem Touchpad anpassen. Alle diese Manipulationen (Drehen, Verschieben und Zoom) können durch das Halten der Ctrl- beziehungsweise Shift-Taste präzisiert beziehungsweise verstärkt werden.

Tabelle 1. Überblick aller Interaktionsmöglichkeiten mit der Simulaotor3Dscene.

Interaktion	**Beschreibung**
`LeftMouse`	Rotiert die Kamera
`Ctrl` + `LeftMouse`	Rotiert die Kamera langsam
`Shift` + `LeftMouse`	Rotiert die Kamera schnell
`RightMouse`	Bewegt die Kamera nach links/rechts
`Ctrl` + `LeftMouse`	Bewegt die Kamera langsam nach links/rechts
`Shift` + `LeftMouse`	Bewegt die Kamera schnell nach links/rechts
`Scroll`	Zoom erhöhen/verringern
`Ctrl` + `Scroll`	Zoom langsam erhöhen/verringern

Backend

Die Netzwerkschnittstelle und die grundlegenden Logiken des Simulators wurden stets unter Berücksichtigung des Verhalten der echten Tello-Drohne implementiert. Als Ausgangslage diente die offizielle Tello SDK 2.0 User Guide [3] sowie eine Tello-Drohne, welche als Testobjekt verwendet werden konnte. Damit liessen sich Stück für Stück die Logiken der Tello-Drohne rekonstruieren und in den Simulator implementieren. Die aus den Tests mit der Tello-Drohne gewonnenen Erkenntnisse sind ins Kapitel Programmierschnittstelle eingeflossen.

Systembedingt mussten auch einige Spezialfälle berücksichtigt werden. Der Simulator kann zum Beispiel im Gegensatz zur echten Drohne kein eigenes Wireless-Netzwerk aufbauen. Ausserdem muss der Simulator auch auf dem gleichen Gerät laufen können wie das Operator-Programm. Dabei kann es zu Konflikten mit der Port-Belegung kommen. Dies ist unter UDP-Schnittstelle genauer erläutert. Bei der Tello-Drohne hat man diese Probleme nicht, da die Drohne immer alle Ports für sich selbst zur Verfügung hat.

Um die zentralen Datenflüsse der Tello-Drohne abzubilden, implementiert der TelloSimulator zwei Threads, welche parallel zu dem Hauptprogramm laufen: die **CommandConnection** und die **StateConnection**. Die **VideoConnection** als letzter Teil dieser Dreifaltigkeit wurde aus Ressourcengründen nicht umgesetzt.

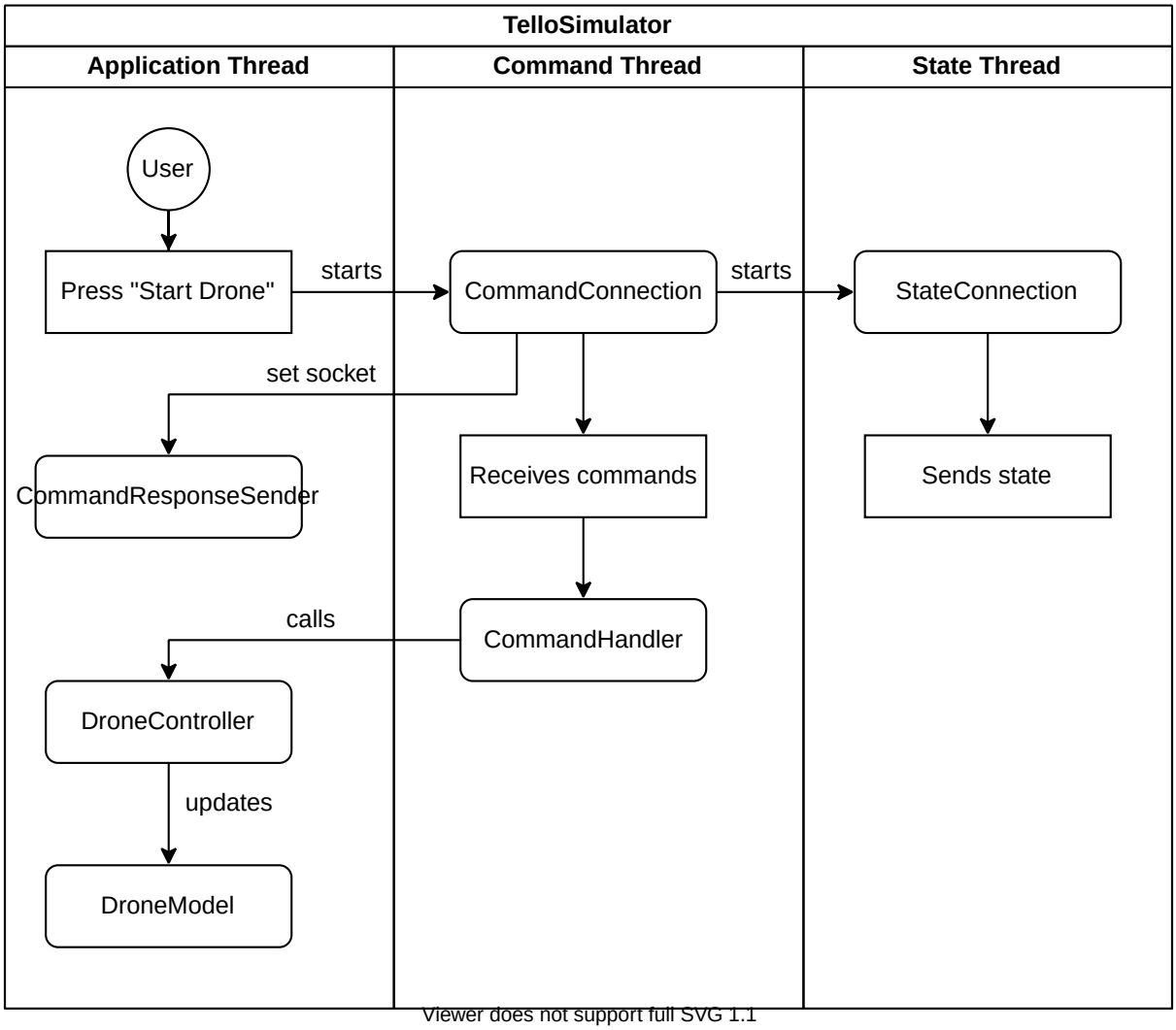

Abbildung 2. Ein grober Überblick, wie das Backend des TelloSimulator bezüglich parallel laufender Threads aufgebaut ist.

CommandConnection

Sobald der Benutzer die virtuelle Drohne einschaltet, wird eine neue Instanz der CommandConnection erstellt und der Thread gestartet. Beim Erstellen des Sockets besteht hierbei eine Eigenheit des Simulators. Der Standard-Port der Tello-Drohne wäre 8889, jedoch wird dieser Port in den meisten Fällen schon durch das ebenfalls lokal laufende Operator-Programm belegt sein. Deshalb bindet sich der Simulator-DatagramSocket der CommandConnection bewusst zum Port 8879 anstatt 8889. Danach empfängt der Thread laufend UDP-Pakete auf diesem Socket.

Nach einem initialen command-Command wird dann wie bei der echten Tello-Drohe der SDK Mode aktiviert. Ab dann ist die Drohne bereit für andere Commands. Gleichzeitig bewirkt dies die Initiierung der StateConnection, welche ab dann regelmässig den Drohnen-Status versendet.

Alle über die CommandConnection empfangenen Nachrichten werden gewrappt als CommandPackage samt Herkunfts-Adresse und Herkunfts-Port an die CommandHandler-Klasse weitergegeben.

StateConnection

Die StateConnection ist ein Stück weniger kompliziert, da sie sich nur mit dem Versenden des Drohnen-Status befassen muss. Nach dem Start durch die CommandConnection schickt die StateConnection asynchron alle 100 ms den Status der Drone im entsprechenden Format an die Addresse, von welcher das erste command-Command empfangen wurde. Da auf dem State-Port nichts empfangen werden muss, verwendet der Simulator hier den gleichen Port wie die Tello-Drohne.

CommandHandler

Die Aufgabe der CommandHandler-Klasse ist es, mit den verschiedenen Commands umzugehen. Der CommandHandler splittet die über die CommandConnection empfangenen Command-Strings auf und extrahiert die enthaltenen Parameter. Anschliessend wird über ein Switch-Statement jedes Command validiert und zu den entsprechenden Methoden im DroneController weitergeleitet. Kann ein Command nicht erfolgreich validiert werden, wird über den CommandResponseSender eine entsprechende Antwort an den Client, auf dem das Operator-Proramm läuft, versendet.

CommandResponseSender

Da der Simulator zu diversen Zeitpunkten und von verschiedensten Klassen aus eine Response schicken können muss, ist der CommandResponseSender als `public final class` implementiert. So ist diese Klasse immer die einzige Quelle aller Responses des Simulators. Die Klasse beinhaltet statische Methoden zum Versenden der Responses. Aufgerufen werden diese z.B. aus dem CommandHandler, wenn ein Command als fehlerhaft validiert wurde, oder auch aus dem DroneController, nachdem die Ausführung vollendet wurde. Versendet werden die Responses über den gleichen DatagramSocket, welcher in der CommandConnection inital erstellt wurde. Dadurch erhält das Operator-Programm die Antworten immer von der Adresse wo das erste command-Command hingeschickt wurde.

DroneController

Diese Klasse steuert die virtuelle Drohne und enthält ihre gesamte Logik. Sie aktualisiert und animiert alle Daten, die in dem DroneModel gespeichert sind, dem sie zugeordnet ist. Die Methoden des DroneControllers führen die Befehle aus, wenn sie vom CommandHandler aufgerufen werden. Ebenfalls sendet der Controller Antworten über den CommandResponseSender an das Operator-Programm, sobald ein bestimmtes Command fertig ausgeführt wurde.

DroneModel

Dies ist die Model-Klasse, welche das Datenmodell der Tello-Drohne repräsentiert. Die Werte des DroneModels werden nur durch die Logik des DroneControllers verändert und im Frontend durch die an seine Properties gebundene Views dargestellt. Dabei dient das DroneModel als **single source of truth** für alle anderen Komponenten, die auf die Parameter der Drohne zugreifen möchten. Dies gewährleistet die Datenintegrität und ermöglicht eine einfachere Skalierbarkeit der Applikation in Zukunft.

Testing

Um während der Entwicklung die zentralen Funktionen des Simulators zu garantieren, wurden einige JUnit-Tests implementiert. Insbesondere der CommandHandler erhielt dabei für jedes Command mehrere Tests, welche die Commands mit validen sowie invaliden Werten aufrufen und überprüfen, ob die korrekten Methoden im DroneController aufgerufen werden. Ebenfalls wurde eine Testmethode für das rc-Command geschrieben, da das Command die zentralste Funktion zur Steuerung der Drohne via Joystick ist.

Interessant war dabei, dass durch das Schreiben der Tests einige Eigenheiten der Commands entdeckt wurden. Dazu gehörte unter anderem die Möglichkeit, Werte mit Kommastellen als Parameter zu übergeben. Diese Spezialfälle waren zuvor nicht aufgefallen, da sie in der Tello SDK gar nicht dokumentiert sind.

UDP-Schnittstelle

Identisch zur Tello-Drohne findet auch beim Simulator die gesamte Kommunikation über das UDP-Netzwerkprotokoll statt. Um den Verbindungsaufbau mit dem Simulator ähnlich wie mit der Tello-Drohne zu gestalten, wurde die Schnittstelle so weit wie möglich gleich gestaltet, wie sie von der Tello-Drohne implementiert wird. Als Grundlage diente hierbei der offizielle Tello SDK 2.0 User Guide [3]. Des Weiteren wurden eigene Tests mit der Tello-Drohne durchgeführt, welche die teilweise lückenhafte Dokumentation im User Guide ergänzen.

Visualisierung der Schnittstellen

In Abbildung 3 ist der Netzwerkaufbau dokumentiert, wenn das Operator-Programm und der Simulator sich auf dem selben Gerät (gleiche IP) befinden. Abbildung 4 hingegen zeigt, wenn sie sich auf unterschiedlichen Geräten (andere IPs) befinden. Die genauen Erklärungen zur Schnittstelle des Simulators und der Tello-Drohne sind in den nachfolgenden zwei Kapiteln zu finden.

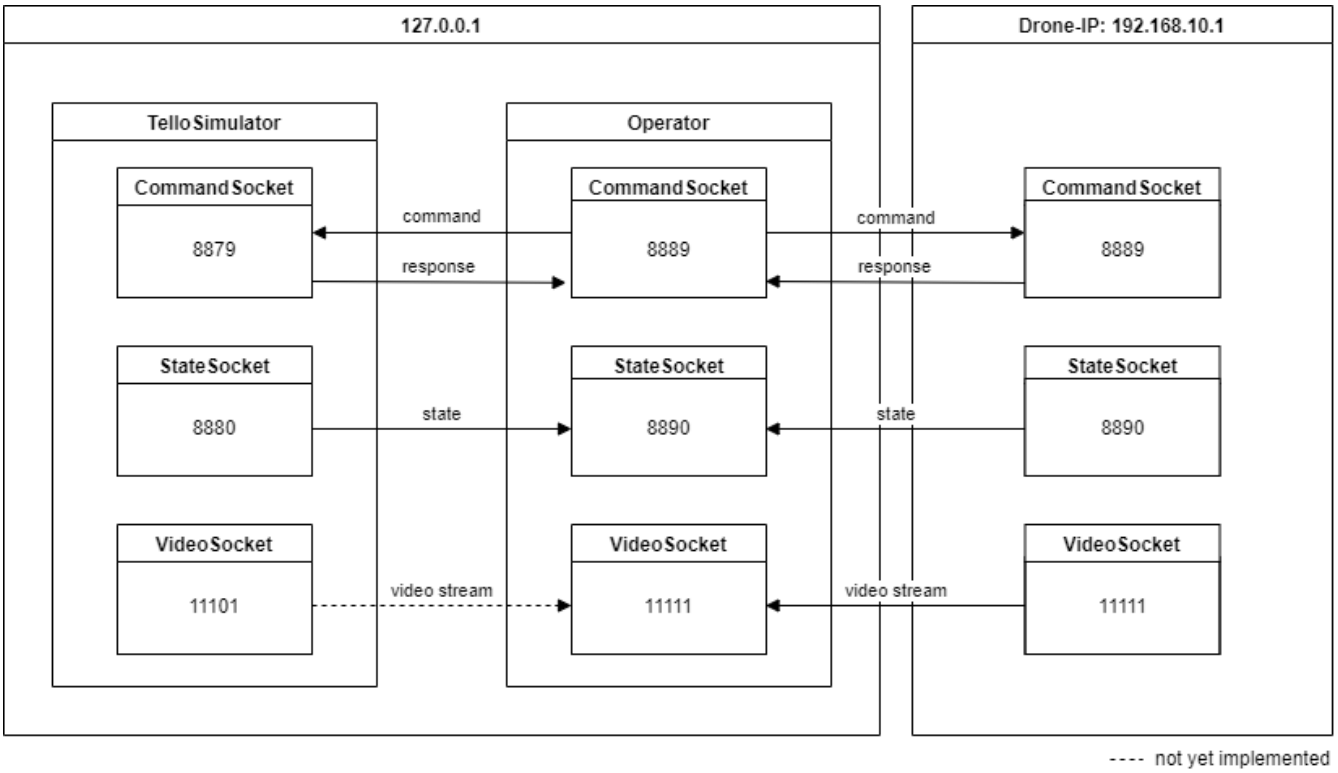

Abbildung 3. Netzwerk-Diagramm mit Simulator, Client-Programm (Operator) und Tello-Drohne (Operator und Simulator haben die gleiche IP).

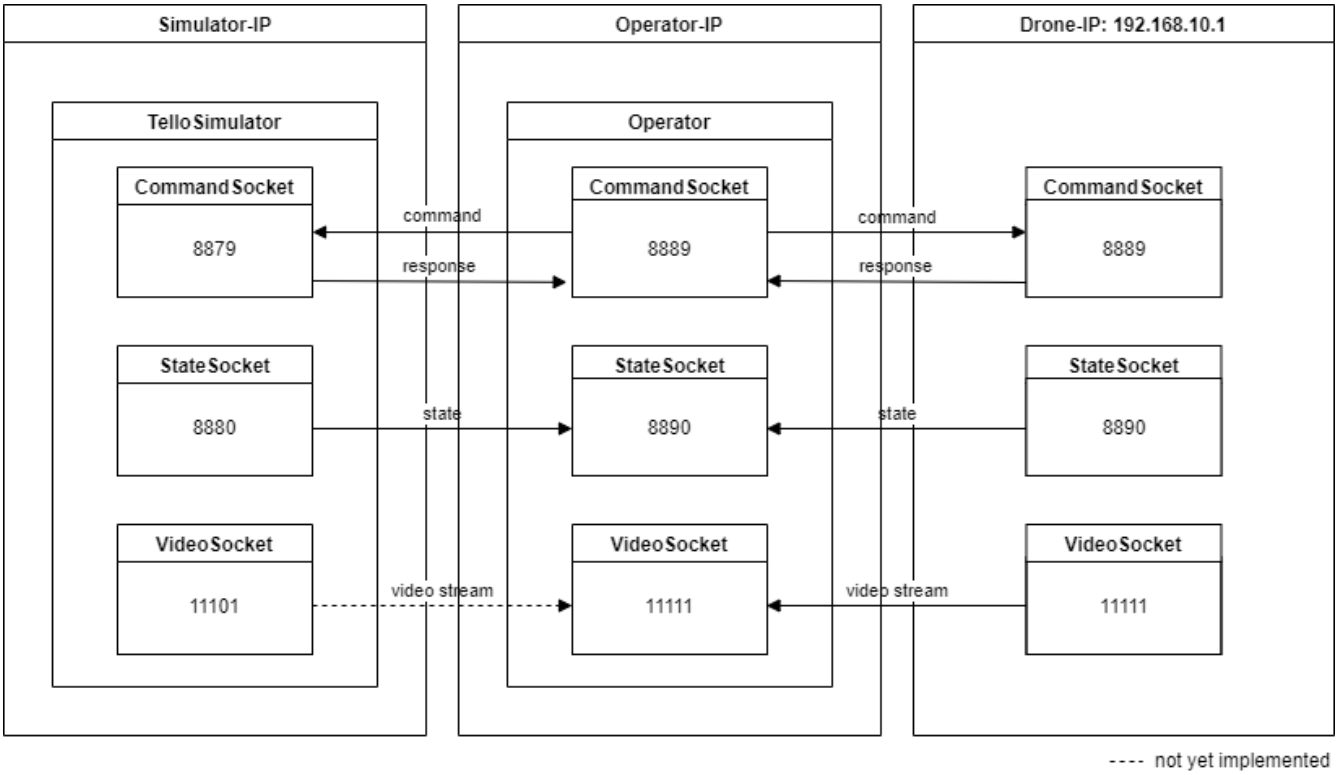

Abbildung 4. Netzwerk-Diagramm mit Simulator, Client-Programm (Operator) und Tello-Drohne (Operator und Simulator haben eine unterschiedliche IP).

Die UDP-Schnittstelle der Tello-Drohne

Die Schnittstelle der Tello-Drohne als eigenständiges Gerät im Netzwerk ist ziemlich unkompliziert und statisch. Commands werden auf der fixen Adresse **192.168.10.1:8889** empfangen sowie an den gleichen Port des Clients versendet. Der State wird auf Port **8890** geschickt, der Videostream auf Port **11111**.

Tabelle 2. Tello-Drone UDP-Schnittstelle für Commands

Verbindung	IP-Adresse der Drohne	Empfängt Pakete auf Socket mit Port	Sendet Pakete statisch an
Command	192.168.10.1	8889	Client-IP:8889
State	192.168.10.1	-	Client-IP:8890
Video	192.168.10.1	-	Client-IP:11111

Die UDP-Schnittstelle des Simulators

Aufgrund der Anforderung, dass der Simulator sowohl vom gleichen Gerät aus als auch von jedem sich im lokalen Netzwerk befindenden Client angesteuert werden können muss, ist seine Schnittstelle leicht unterschiedlich gestaltet.

Denn wenn das Client-Programm sowie der Simulator auf dem gleichen Gerät laufen, führt dies zu Konflikten in der Port-Belegung. Beispielsweise kann der Port 8889 nicht mehr vom Client-Programm verwendet werden, wenn sich der Simulator schon an diesen gebunden hat. Daher bindet der Tello-Simulator seinen DatagramSocket der CommandConnection bewusst zum Port **8879** anstatt 8889. Somit kann der Client weiterhin einen Socket auf 8889 erstellen, wie es von der echten Drohne verlangt wird, ohne vom Simulator blockiert zu werden.

Tabelle 3. Tello-Simulator UDP-Schnittstelle für Commands

Verbindung	IP-Adresse des Simulators	Empfängt Pakete auf Socket mit Port	Sendet Pakete dynamisch an
Command	beliebig | 127.0.0.1	8879	IP und Port aus empfangenen Paket
State	beliebig | 127.0.0.1	-	IP und Port aus empfangenen Paket
Video	beliebig | 127.0.0.1	-	(noch nicht implementiert)

Somit verhält sich die Simulator-Schnittstelle ein wenig dynamischer als diejenige der Drohne (Port von welchem das Operator-Programm sendet ist beliebig). Trotzdem benötigt es Client-seitig nur zwei kleine Anpassungen, um mit dem Simulator zu funktionieren:

1. **Die IP muss von 192.168.10.1 auf die Simulator-IP (ersichtlich im Simulator-UI) eingestellt werden.**
2. **Der UDP-Socket des Clients (gebunden an Port 8889) muss sich in der connect-Methode zu 8879 verbinden anstatt 8889.**

> **Hinweis:** Eine detailliere Anleitung zum Verbindungsaufbau mit dem Simulator inkl. beispielhaftem Java-Code ist im Readme des Projekts zu finden.

Programmierschnittstelle

In diesem Kapitel wird die Programmierschnittstelle des Simulators beschrieben. Dazu gehören alle Commands, welche vom Simulator unterstützt werden. Grundsätzlich basiert diese Dokumentation der Commands auf dem Tello SDK 2.0 [3]. Dank ausführlichen Tests mit der Tello-Drohne beinhaltet die folgende Auflistung aber detailliertere und vollständigere Beschreibungen zu den einzelnen Commands.

Für alle Commands gilt: Unbekannte oder falsch geschriebene Commands liefern die Antwort «unknown command:» gefolgt vom gesendeten String. Commands sind dabei Case-sensitive, d.h. Gross- und Kleinschreibung muss eingehalten werden.

Tabelle 4. Control Commands

Command	Beschreibung	Mögliche Antworten	Simulator
command	**Enter SDK mode**. Die Drohne ist ab jetzt via commands steuerbar. Ein zweites command zu senden gibt zwar «ok» zurück, hat aber keine weiteren Auswirkungen.	ok / error	[]
takeoff	**Auto takeoff**. Startet die Motoren und fliegt 30 cm nach oben. Wird nach dem initialen takeoff ein weiteres takeoff gesendet, wird es ignoriert und die Drohne schickt «error» als Antwort.	ok / error	[]
land	**Auto landing**. Fliegt nach unten bis der Boden erreicht ist und stoppt anschliessend die Motoren. Schickt «ok» nach der Landung.	ok / error	[]
streamon	**Enable video stream**. Startet die Video-Übertragung.	ok / error	
streamoff	**Disable video stream**. Stoppt die Video-Übertragung.	ok / error	
emergency	**Stops motors immediately**. Stoppt die Motoren, fällt auf den Boden. Sendet weiterhin den Status. Hinweis: Funktioniert zu jeder Zeit.	keine Antwort	[]
up x	**Ascend to x cm**. *x = 20-500*. Bewegt sich x cm nach oben. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet	ok / error / out of range	[]
down x	**Descend to x cm**. *x = 20-500* . Bewegt sich x cm nach unten. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet.	ok / error / out of range	[]

Command	Beschreibung	Mögliche Antworten	Simulator
left x	**Fly left for x cm**. *x = 20-500*. Bewegt sich x cm nach links. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet.	ok / error / out of range	[]
right x	**Fly right for x cm**. *x = 20-500*. Bewegt sich x cm nach rechts. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet	ok / error / out of range	[]
forward x	**Fly forward for x cm**. *x = 20-500*. Bewegt sich x cm nach vorne. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet.	ok / error / out of range	[]
back x	**Fly backwards for x cm**. *x = 20-500*. Bewegt sich x cm nach hinten. Nachdem die Drohne sich wieder stabilisiert hat wird die Antwort «ok» versendet.	ok / error / out of range	[]
cw x	**Rotate x degrees clockwise**. ~~*x = 1-360*~~. Dreht die Drohne um x Grad im Uhrzeigersinn um seine Yaw-Achse. Der Range Check 1-360 Grad ist zwar in der SDK dokumentiert, wird von der Drohne sowie dem Simulator aber nicht angewandt. Auch negative Werte sind möglich.	ok / error	[]
ccw x	**Rotate x degrees counterclockwise**. ~~*x = 1-360*~~. Dreht die Drohne um x Grad im Gegenuhrzeigersinn um seine Yaw-Achse. Der Range Check 1-360 Grad ist zwar in der SDK dokumentiert, wird von der Drohne sowie dem Simulator aber nicht angewandt. Auch negative Werte sind möglich.	ok / error	[]
flip x	**Flip in x direction**. *x = left | right | forward | back*. Führt einen Salto in die angegebene Richtung aus.	ok / error / out of range	[]

Command	Beschreibung	Mögliche Antworten	Simulator
go x y z speed	**Fly to x y z at speed (cm/s)**. *x = -500-500, y = -500-500, z = -500-500, speed = 10 - 100*. Fliegt zu den entsprechenden Koordinaten (relativ zur aktuellen Drohnenposition). Achsen: x = Drohnenausrichtung, y = Linker Normalvektor der Drohne, z = Aufwärtsvektor der Drohne. Hinweis: x-, y- und z-Werte können nicht gleichzeitig zwischen -20 - 20 eingestellt werden.	ok / error / out of range	[]
stop	**Hovers in the air**. Unterbricht die Ausführung eines Commands und stoppt die Drohne an ihrer aktuellen Position. Hinweis: Funktioniert zu jeder Zeit.	ok / forced stop / error	[]
curve x1 y1 z1 x2 y2 z2 speed	**Fly at a curve according to the two given coordinates at speed (cm/s)**. *x1, x2 = -500 -500, y1, y2 = -500-500, z1, z2 = -500-500, speed = 10 - 60*. Mit Hilfe der aktuellen Drohnenposition als Punkt (0,0,0) und der beiden gegebenen Punkten (relativ zur aktuellen Drohnenposition) wird ein Kreisbogen im Raum konstruiert. Die Drohne fliegt auf dieser Kurve bis sie am Endpunkt (x2, y2, z2) angelangt ist. Hinweis 1: Wenn x1, y1 und z1 oder x2, y2 und z2 gleichzeitig zwischen -20 und 20 sind, schickt die Drohne die Antwort «out of range». Hinweis 2: Wenn der Bogenradius nicht innerhalb eines Bereichs von 0,5-10 Metern liegt, schickt die Drohne die Antwort «error Radius is too large!».	ok / error / out of range / error Radius is too large!	[]

Command	Beschreibung	Mögliche Antworten	Simulator
go x y z speed mid	**Fly to the x y z coordinates of the Mission Pad at speed (cm/s)**. *mid = m1-m8, x = -500 -500, y = -500-500, z = -500-500, speed = 10 - 100*. Fliegt zu den entsprechenden Koordinaten (relativ zur aktuellen Drohnenposition). Achsen: x = Drohnenausrichtung, y = Linker Normalvektor der Drohne, z = Aufwärtsvektor der Drohne. Hinweis: x-, y- und z-Werte können nicht gleichzeitig zwischen -20 - 20 eingestellt werden.	ok / error / out of range	
curve x1 y1 z1 x2 y2 z2 speed mid	**Fly at a curve according to the two given coordinates of the Mission Pad ID at speed (cm/s)**. *x1, x2 = -500-500, y1, y2 = -500 -500, z1, z2 = -500-500, speed = 10 - 60*. Hinweis 1: Wenn x, y und z gleichzeitig zwischen -20 und 20 sind, schickt die Drohne die Antwort «out of range». Hinweis 2: Wenn der Bogenradius nicht innerhalb eines Bereichs von 0,5-10 Metern liegt, schickt die Drohne die Antwort «error Radius is too large!».	ok / error / out of range / error Radius is too large!	
jump x y z speed yaw mid1 mid2	**Fly to coordinates x, y and z of Mission Pad 1, and recognize coordinates 0, 0, z of Mission Pad 2 and rotate to the yaw value**. *mid = m1-m8, x = -500-500, y = -500 -500, z = -500-500, speed = 10 - 100 (cm/s).* Hinweis 1: Wenn x, y und z gleichzeitig zwischen -20 und 20 sind, schickt die Drohne die Antwort «out of range».	ok / error / out of range	

Tabelle 5. Set Commands

Command	Beschreibung	Mögliche Antwort	Simulator
speed x	**Set speed to x cm/s**. *x = 10-100*. Setzt die Speed-Variable der Drohne auf den entsprechenden Wert.	ok / error	[]
rc a b c d	**Set remote controller control via four channels**. *a = left/right (-100-100), b = forward/backward (-100-100), c = up/down (-100-100), d = yaw (-100-100)*. Setzt die Bewegungsgeschwindigkeiten in cm/s in die entsprechende Richtung. Die Werte sind unabhängig von der gesetzten speed-Variable auf der Drohne. Hinweis 1: Funktioniert zu jeder Zeit und schickt kein ok. Hinweis 2: Wenn während einer Sekunde nur 20 nach vorne gegeben werden, reicht das meistens noch nicht aus, um die Drohne zu bewegen. Es braucht mind. 30 in eine Richtung während zwei Sekunden, um die Drohne überhaupt aus dem Gleichgewicht zu bringen.	out of range	[]
wifi ssid pass	**Set Wi-Fi password**. *ssid = updated Wi-Fi name, pass = updated Wi-Fi password.*	ok / error	[]
mon	**Enable mission pad detection (both forward and downward detection)**.	ok / error	
moff	**Disable mission pad detection**.	ok / error	
mdirection x	**Change mission pad detection mode**. *x = 0/1/2, 0 = Enable downward detection only, 1 = Enable forward detection only, 2 = Enable both forward and downward detection.*	ok / error	
ap ssid pass	**Set the Tello to station mode, and connect to a new access point wit the access point's ssid and password**. *ssid = updated Wi-Fi name, pass = updated Wi-Fi password.*	ok / error	

Tabelle 6. Read Commands

Command	Beschreibung	Mögliche Antwort	Simulator
speed?	**Obtain current speed (cm/s)**. Gibt den aktuell gesetzten Wert der speed-Variable zurück.	x = 10-100 z.B. `100.0\r\n`	[]
battery?	**Obtain current battery percentage**. Gibt den aktuellen Batterieladestand zurück.	x = 0-100 z.B. `76\r\n`	[]
time?	**Obtain current flight time**. Gibt zurück, wie lange die Drohne bereits geflogen ist, seit sie eingeschaltet wurde (in Sekunden). Wenn die Drohne nach der Landung also nicht ausgeschaltet und wieder takeoff gesendet wird, wird die Zeit einfach aufsummiert.	z.B. `24s\r\n`	[]
wifi?	**Obtain Wi-Fi SNR**. Gibt das Wi-Fi Signal-to-Noise Ratio zurück. Hardcoded `90` für den Simulator	z.B. `90\r\n`	[]
sdk?	**Obtain the Tello SDK version**. Gibt die SDK Version zurück. Z.B. (Tello SDK 2.0) = `20` für den Simulator	z.B. `20\r\n`	[]
sn?	**Obtain the Tello serial number**. Gibt die Seriennummer der Drohne zurück.	z.B. `0TQDG3UEDBSP12`	[]

Hinweis: Antworten von Read Commands sowie der Status der Drone enthalten teilweise new line characters `\r\n`.

Weitere Eigenschaften der Tello-API

Durch ausführliche Tests mit der Tello-Drohne konnten ausserdem folgende besondere Eigenschaften der Tello-API beobachtet werden:

- Es gibt Commands die ein «ok» zurückschicken, aber auch **«Fire and Forget»-Commands** wie z.B. das rc-Command, welches zu jedem Zeitpunkt von der Drohne ausgeführt wird und keine Antwort schickt.
- Commands werden von der Drohne **nicht gequeued**. Das heisst das Client-Programm muss entsprechend programmiert werden, damit es der Drohne das nächste Command zum passenden Zeitpunkt schickt. Kommt ein Command während ein vorhergehendes noch nicht

fertig ausgeführt wurde, dann wird die Antwort **«error Not joystick»** geschickt. Ausnahmen hierzu sind die rc-, stop- und emergency-Commands, welche zu jeder Zeit funktionieren.

- Mitgeschickte **Parameter der Commands können nicht weggelassen werden**, müssen also immer vorhanden sein. Sonst wird das Command nicht erkannt.
- Die Tello-Drohne beginnt mit dem Senden des Status erst, nachdem sie **das erste command-Command** erhalten hat.
- Parameter mit **Kommastellen** wie z.B. «forward 35.234234» oder «cw 35.23453» werden von der Drohne ausgeführt.
- Wird nach dem land-Command ein weiteres Control-Command geschickt, welches laufende Motoren benötigt, kommt **«error Motor stop»** als Antwort zurück.
- Mit dem rc-Command gesetzte Werte bleiben **auch nach der Landung noch aktiv**. Eine Landung setzt die rc-Werte also nicht zurück und müssen vor einem erneuten Start idealerweise manuell mit einem erneuten Befehl «rc 0 0 0 0» zurückgesetzt werden.

Usability Testing

Dieses Kapitel beschreibt im Speziellen die Erkenntnisse, welche aus einem Usability Testing gewonnen werden konnten. Des Weiteren wird die Vorgehensweise des Testings kurz erläutert. Das Testkonzept, das Testskript, die Testprotokolle sowie die Auswertung sind im Anhang zu finden. Das Testing wurde auf dem Commit e63f765 durchgeführt, die Ansicht des GUIs wird in Abbildung 5 gezeigt.

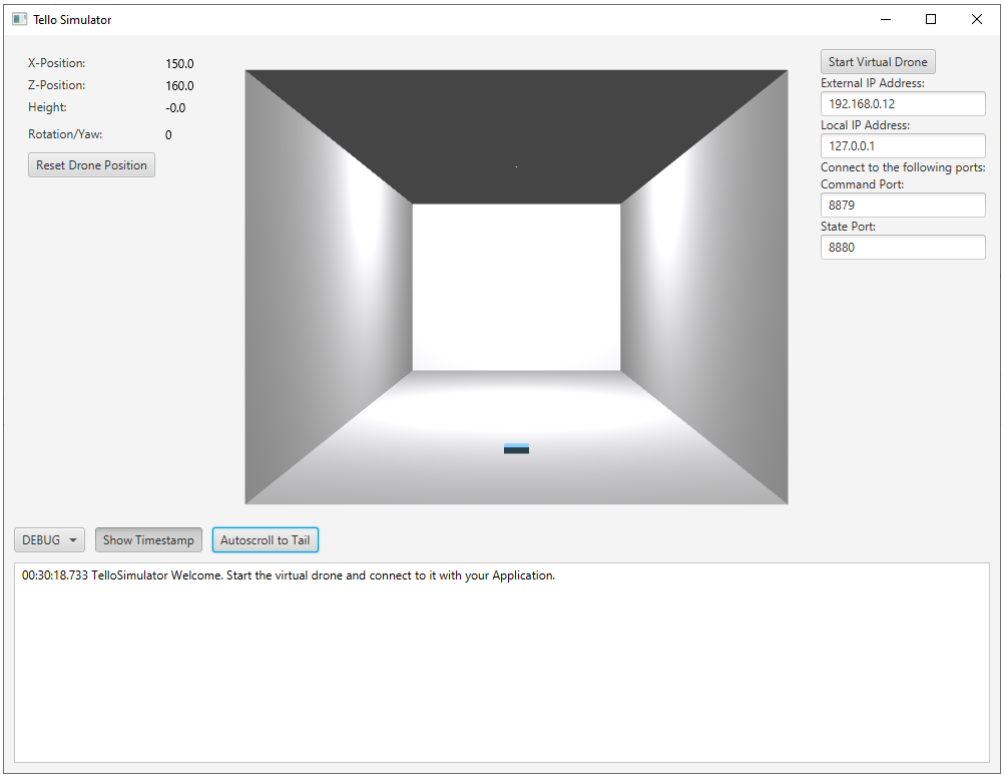

Abbildung 5. GUI zum Stand des Usability Testings.

Vorgehensweise

Um eine unkomplizierte Inbetriebnahme sowie eine einfache Handhabung des Simulators zu gewährleisten, wurde ein Usability Testing durchgeführt.

Das Testing wurde in die folgenden drei Abschnitte unterteilt: «Installation», «Konfiguration des Clients» und «Anwendung TelloSimulator», diese sind im Testkonzept im Anhang etwas genauer beschrieben. Per Videotelefonie wurde das Testing mit vier iCompetence-Studierenden der FHNW (2./3. Semester) durchgeführt. Dabei wurde eine Aufgabenstellung per Chat abgegeben und der Proband versuchte die Aufgabe ohne Hilfe des Moderierenden zu lösen. Der Moderierende konnte bei Problemen eingreifen, der Beobachtende notierte die wichtigsten Verhaltensweisen und Aussagen des Probanden. Zusätzlich wurde das Meeting aufgezeichnet, um wichtige Abschnitte nachgehend noch detaillierter zu dokumentieren.

Anschliessend wurden die wichtigsten Erkenntnisse in die folgenden Kategorien unterteilt: «negative Aussagen / beobachtete Probleme», «positive Aussagen» und «Tipps». Diese wurden nach Thema gruppiert. Pro Thema wurde ein Verbesserungsvorschlag definiert und in das Backlog aufgenommen.

Erkenntnisse

Durch das Usability Testing konnten die folgenden Erkenntnisse gewonnen werden. Dies ist nur ein Auszug der Wichtigsten, genauere Details sind im Anhang zu finden.

Installationsanleitung

- Weniger Beschreibungen, warum etwas gemacht wird, dafür genauere Anweisungen
- Probleme, welche auftreten können, in einen Troubleshooting-Abschnitt extrahieren

Simulator

- Visuelle Orientierung im Raum ist sehr wichtig, um zu sehen, ob die Drohne sich wie gewünscht verhält
  - Schatten einbauen
  - Kameraposition überdenken
  - Drohne als 3D-Model implementieren oder mindestens ein Pfeil auf dem Quader einblenden, damit die Blickrichtung der Drohne klar ist
- Drohne/Simulator muss auch wieder ausgeschaltet werden können
- GUI ist noch nicht sehr strukturiert (schwierig, wichtige Infos zu finden)

Zur Weiterentwicklung des Simulators wurden diese Verbesserungsvorschläge priorisiert und entsprechend in das Backlog übertragen.

Fazit

Erfüllung der Anforderungen

Eine der wohl zentralsten Anforderungen war das **Entgegennehmen von Commands sowie das Versenden von Responses** via UDP. Deshalb wurde bereits zu Projektbeginn der Fokus auf die Entwicklung dieser Schnittstelle gelegt. So konnte bereits vor MVP-Release eine solide Basis für das Backend gebaut werden, auf welcher auch der finale Release noch beruht.

Das eng mit der ersten Anforderung zusammenhängende **Versenden des Status** gestaltete sich dank der guten Vorarbeit mit der CommandConnection entsprechend leicht.

Auch das **Umsetzen aller Control-, Read- und Set-Commands** konnte bis auf die Mission Pad Detection erfüllt werden. Obwohl beim Ausdruck «realitätsgetreu» wohl gewisse Einschränkungen gemacht wurden (keine Simulation von Schwerkraft, Wind und weiteren komplexen Faktoren) verhält sich die virtuelle Drohne grösstenteils wie die echte Tello-Drohne.

Wo der Simulator ebenfalls glänzt, ist das **Error-Handling** von Commands sowie das ganze Logging der Applikation. Mittels Tests mit der echten Tello-Drohne und dem Auslesen des Netzwerk-Verkehrs konnten viele Erkenntnisse bezüglich dem Error-Handling der Drohne gewonnen werden. Diese gingen über die öffentlich zugängliche Tello SDK 2.0 Dokumentation hinaus und halfen dabei, ein möglichst ähnliches Error-Handling zu programmieren. Alle Commands und Parameter werden durch den Simulator validiert und auf das im UI integrierte Log ausgegeben. Dies bietet eine Unterstützung beim Debugging.

Die **Raumgrösse** kann via Slider eingestellt werden. Die **Startposition** kann via Reset-Button zurückgesetzt werden.

Leider erwies sich das **Versenden des Video-Feeds** als komplizierter als erwartet. Es wurde zwar eine funktionsfähige VideoPublisher-Klasse implementiert, welche den Webcam-Feed via UDP überträgt. Die Video-Schnittstelle wurde dabei mittels H.264-Encoding umgesetzt, analog zu derjenigen der Tello-Drohne. Jedoch war es schlussendlich mangels Wissens und Ressourcen nicht möglich, die Funktionalität auf das Versenden der JavaFX-Scene-Snapshots anzuwenden. Aus diesem Grund wurde die Priorität dieser Anforderung nach unten korrigiert und der Task ins Backlog verschoben.

Die Anforderung, mit **Drohnen-Schwärmen** umzugehen, konnte leider ebenfalls noch nicht angegangen werden. Trotzdem wurde bei der Gestaltung der Software-Architektur darauf geachtet, dass die Applikation skalierbar ist. Durch die Entkopplung von CommandHandler, DroneController und DroneModel könnten theoretisch vom CommandHandler aus auch mehrere DroneModels geupdated werden. Auch vonseiten View wäre für die Anzeige mehrerer DroneViews keine grössere Veränderung nötig.

Zusammenfassend ist durch das Projekt ein funktionsfähiger Simulator entstanden, der die Grundanforderungen weitgehend abdeckt.

Wahl der Programmiersprache

Zusammen mit den Betreuenden wurde Anfangs des Projekts entschieden, das JavaFX-Framework für die Applikation zu verwenden. Der Hauptgrund dafür war das Vorwissen der Betreuenden und Studierenden. Der grosse Vorteil dabei war, dass wir sehr schnell mit dem Programmieren loslegen konnten und uns nicht zuerst in eine neue Sprache eindenken mussten. Als Nachteil betrachten wir, dass das Framework nicht wirklich auf 3D ausgelegt ist. Die Vorteile des vorhandenen Wissens überwiegten jedoch schlussendlich, und im Nachhinein würden wir uns wieder dafür entscheiden.

Mögliche Weiterentwicklungen

Obwohl der Simulator in seinem jetzigen Zustand die Grundanforderungen abdeckt, gibt es noch diverse Funktionen, die aus Ressourcengründen im Backlog des Projekts geblieben sind. Dazu gehören die im vorangehenden Abschnitt beschriebenen Funktionen des Video-Streams sowie der Drohnen-Schwärme. Zusätzlich wurden während der Entwicklung und auch im Testing diverse zusätzliche Weiterentwicklungen diskutiert;

Verbesserungen Usability

- **Schatten der Drohne generieren und anzeigen:** Dadurch würde der Abstand zum Boden und die Position der Drohne im Raum noch besser ersichtlich.
- **Simulator als Fullscreen und die Simulator Controls als darüberliegende Ebene darstellen:** So würde der Fokus noch mehr auf der Simulation liegen und der Betrachtende fühlt sich als Teil des 3D-Raums.
- **Simulator Controls als Custom Control gestalten:** Würden die Simulator Controls (x/y/z-Positon, Yaw/Pitch-Roll-Drehung, Reset-Button…) als Custom Control gestaltet, hälfe dies die Bedeutung und die Funktion intuitiver zu erkennen.
- **Log und Network Controls ein- und ausblendbar machen:** Dadurch kann die 3D-Simulation bei Bedarf vergrössert und besser fokussiert werden.
- **Button, um das Log zu leeren:** Dies würde helfen, einfacher zu debuggen, da es übersichtlicher wird.
- **Startposition der Drohne im Raum festlegen:** Trägt zu einer noch realistischeren Funktionsweise des Simulators bei. Dies hilft bei Testings zu erkennen, ob die Drohne beispielsweise in eine Wand fliegt.

Flugverhalten realistischer gestalten

- Ausbalancieren nach einem ausgeführten Command
- Flugverhalten bei Flips (nicht um eigene Achse drehen, sondern als kleiner «Kreis»)
- Beschleunigung und Geschwindigkeit exakter erfassen

Danksagung

Gerne möchten wir uns bei den folgenden Personen bedanken:

Dank einem kurzen Pausengespräch ist es überhaupt erst dazu gekommen, dass Dr. Dieter Holz diese Arbeit im Namen der FHNW eingereicht hat. Wir freuen uns, dass er ein Projekt auf die Beine gestellt hat, welches zu unseren Interessen und Fähigkeiten passt.

Herzlichen Dank an unsere Betreuer*innen, Dr. Barbara Scheuner und Dr. Dieter Holz für die stete Unterstützung und die konstruktiven Gespräche während der Projektarbeit.

Ebenfalls möchten wir uns bei den Probanden des Usability Testings bedanken. Wir konnten dadurch wertvolles Feedback einholen und den Simulator in der zweiten Projektphase erheblich weiterentwickeln.

Referenzen

Literaturverzeichnis

- [1] https://www.ryzerobotics.com/tello-edu
- [2] https://www.fhnw.ch/de/studium/module/9052760
- [3] https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

Abbildungsverzeichnis

Tabellenverzeichnis

Ehrlichkeitserklärung

Hiermit erklären wir, die vorliegende Projektarbeit selbstständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben.

Schlieren, 29.11.2020

Daniel Obrist

Schaffhausen, 29.11.2020

Severin Peyer

Anhang

Appendix A: Installationsanleitung Tello Simulator

Appendix B: Usability-Testkonzept

Appendix C: Usability-Testskript

Appendix D: Testprotokolle

Testskript Testperson 01

Testskript Testperson 02

Appendix E: Auswertung Usability Testing

Auswertung Usability Testing Teil 1 - Installation Simulator

Auswertung Usability Testing Teil 2 - Einrichtung Operator

Auswertung Usability Testing Teil 3 - Anwendung Simulator