HAW HAMBURG

Bachelorarbeit

Leon Chun Wai Yuen

Entwicklung eines digitalen Zwillings für eine Tello-Drohne im MARS-Framework

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Leon Chun Wai Yuen

Entwicklung eines digitalen Zwillings für eine Tello-Drohne im MARS-Framework

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Prof. Dr. Zhen Ru Dai

Eingereicht am: 08.06.2023

Leon Chun Wai Yuen

Thema der Arbeit

Entwicklung eines digitalen Zwillings für eine Tello-Drohne im MARS-Framework

Stichworte

Digitaler Zwilling, Multiagentensystem, MARS-Framework, Quadrokopter, Tello-Drohne, C#

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung eines digitalen Zwilling Prototypen einer Tello-Drohne in einem Multiagentensystem in MARS. Es wird ein Konzept vorgestellt, wie sich ein Digitaler Zwilling (DZ) für individuelle Quadrokoter als Teil eines Multiagentensystems entwickeln lassen. Das Modell integriert die Fluginformationen einer physischen Tello-Drohne, die vom Agenten genutzt werden, um den Zustand virtuell abzubilden. Darüber hinaus besitzt der digitale Zwilling die Fähigkeit zur autonomen Steuerung der Drohne. Abschließend wird das System durch geeignete Testszenarien evaluiert.

Leon Chun Wai Yuen

Title of Thesis

Implementation of a digital twin of a tello drone in the MARS Framework

Keywords

Digital Twin, Multi-agent System, MARS-Framework, Quadcopter, Tello drohne, C#

Abstract

The present thesis focuses on the development of a digital twin prototype of a Tello drone in a multi-agent system. A concept is presented on how to develop a digital twin for individual quadcopters as part of a multi-agent system in the MARS framework. The model integrates the flight information of a physical Tello drone, which is used by the agent to virtually represent its state. In addition, the digital twin has the ability to autonomously control the drone. Finally the system is evaluated through suitable test cases.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

API Application Programming Interface.

DTI Digital Twin Instance.

DTP Digital twin Prototype.

DZ Digitaler Zwilling.

MARS Multi-Agent Research & Simulation.

MAS Multiagentensystem.

PZ Physischer Zwilling.

RRN Record-Repeat Navigation.

SDK Software Development Kit.

UAV Unbemanntes Luftfahrzeug.

Symbolverzeichnis

a_x Beschleunigung in x-Richtung in $\frac{cm}{s^2}$.

a_y Beschleunigung in y-Richtung in $\frac{cm}{s^2}$.

bat Entladungsstand des Akkus in %.

α Ausrichtung des Agenten in MARS.

dir_{frame} Bewegungsrichtung relativ zum Körper.

dir_{frame} Bewegungsrichtung relativ zur Umgebung.

d Die zurückgelegte Distanz der Tello.

h Flughöhe gemessen von der Ausgangsposition in m.

$\bar{m}_s$ Die mittlere Distanz zur Soll-Koordinate eines Befehls.

ϕ Die Drehung um die x_b-Achse.

ψ Die Drehung um die z_b-Achse.

$\hat{d}$ Die skalierte zurückgelegte Distanz der Tello.

v_x Momentangeschwindigkeit in x-Richtung in $\frac{cm}{s}$.

v_y Momentangeschwindigkeit in y-Richtung in $\frac{cm}{s}$.

t_{curr} Messzeitpunkt der aktuellen Flugparameter.

t_{diff} Zeitdifferenz in s.

θ Die Drehung um die y_b-Achse.

t_{prev} Messzeitpunkt der letzten Flugparamter.

vel_x Initialgeschwindigkeit in x-Richtung in $\frac{cm}{s}$.

vel_y Initialgeschwindigkeit in y-Richtung in $\frac{cm}{s}$.

x_b x-Achse des Body Frame vom Quadrokopter.

x_i Die x-Koordinate der des Ist-Koordinatenpunktes.

x_s Die x-Koordinate der des Soll-Koordinatenpunktes.

x_w x-Achse des World Frame.

y_b y-Achse des Body Frame vom Quadrokopter.

y_i Die y-Koordinate der des Ist-Koordinatenpunktes.

y_s Die y-Koordinate der des Soll-Koordinatenpunktes.

y_w y-Achse des World Frame.

z_b z-Achse des Body Frame vom Quadrokopter.

z_w z-Achse des World Frame.

1 Einleitung

1.1 Motivation

In den letzten Jahren haben sich Multikopter aufgrund des rasanten technologischen Fortschritts in allen Aspekten weiterentwickelt und finden heutzutage in unterschiedlichen Domänen Verwendung. Zu den Anwendungen gehören beispielsweise der Transport von Gütern [5], die Überwachung großflächiger Areale [20] oder der Inspektion schwer zugänglicher Objekte [19]. Um diese Anforderungen gerecht zu werden, müssen die Multikopter in der Entwicklung eine Reihe von Test- und Validierungsverfahren durchlaufen. Allerdings sind solche Verfahren sehr kostenintensiv, da reale Prototypen verwendet werden müssen. Daher wurden in den vergangenen Jahren Simulatoren entwickelt [4], [23], um diesen Prozess in den virtuellen Raum zu verlagern. Solche Simulatoren ermöglichen es, das Verhalten von Multikoptern unter verschiedenen Aspekten und Szenarien zu simulieren, um Erkenntnisse über ihr reales Verhalten zu gewinnen. Jedoch sind Simulatoren in ihren Szenarien begrenzt, da für jedes abzubildende Szenario ein Datensatz vorhanden sein muss. Dadurch entstehen Kenntnislücken, die in der Simulation nicht berücksichtigt werden und erst im realen Einsatz auftreten. Eine Möglichkeit, dieses Problem zu lösen, ist der Einsatz von digitalen Zwillingen. Unter diesem Konzept versteht man die Überführung von Informationen aus dem physischen in den virtuellen Raum, um eine realistischere und umfangreichere Darstellung der realen Umgebung zu ermöglichen.

1.2 Zielsetzung

Das Ziel der Bachelorarbeit besteht aus dem Entwurf und der Implementation eines digitalen Zwillings für ein Drohnenmodell vom Herstelle Ryze (Tello) nach dem Paradigma eines Multiagentensystems. Dazu soll eine agentenbasierte Simulation mit dem MARS-Framework aufgesetzt werden, die die Tello und ihre Umgebung als virtuelles Modell

darstellt. Das System soll imstande sein, Zustands- und Verhaltensänderungen der Tello durch einen kontinuierlichen Datenaustausch zu erfassen und widerzuspiegeln. Zusätzlich wird eine Funktion umgesetzt, die es dem digitalen Zwilling erlaubt, eine mit der Tello aufgezeichneten Trajektorie, zu replizieren. Neben dem digitalen Zwilling soll eine Benutzerschnittstelle entwickelt werden, mit dem ein Fernpilot die Tello manuell steuern kann.

1.3 Gliederung der Arbeit

In Kapitel 2 werden die Grundlagen beschrieben, um den Lesern in das Thema dieser Arbeit einzuführen. Hier werden die Besonderheiten des Quadrokopter und des MARS Frameworks beschrieben, die einen wichtigen Bestandteil dieser Arbeit darstellen. In Kapitel 3 werden die verwendenden Hilfsmittel ausführlich beschrieben, sodass ihre Verwendung im System deutlich einzuordnen ist. Kapitel 4 präsentiert eine Liste der funktionalen und nichtfunktionalen Anforderungen, die in der Umsetzung dieser Arbeit beachtet werden. In Kapitel 5 wird die Konzeption dieser Arbeit vorgestellt. Dazu gehört ein Überblick über die Architektur des Drohnensystems, sowie dessen Besonderheiten und Einschränkungen. Insbesondere werden die Entwürfe erläutert, wie die Kommunikation zwischen dem Drohnensystem und der Tello konzipiert werden kann und wie ein digitaler Zwilling für die Tello in einem Multiagentensystem entwickelt wird. Kapitel 6 stellt eine erste Implementation des vorgestellten Konzeptes im MARS-Framework vor. In Kapitel 7 wird das erstellte System mit zwei Versuchen evaluiert. Der erste Versuch untersucht die Abbildungseigenschaften des digitalen Zwillings. Der zweite Versuch untersucht die zusätzlich entwickelte Funktionalität für den digitalen Zwilling. Zum Schluss werden die Ergebnisse aus den Versuchen diskutiert, sowie eine Bewertung der gesamten Arbeit vorgenommen.

2 Theoretischer Hintergrund und Stand der Forschung

In diesem Kapitel wird der theoretische Rahmen für die Arbeit gelegt. Es werden relevante Themen eingeführt und erklärt, die für ein umfassendes Verständnis der Arbeit wichtig sind. Außerdem wird zu den jeweiligen Themen eine kurze Zusammenfassung des technischen Standes gegeben, auf den sich die Arbeit stützt.

2.1 Quadrokopter

Ein Quadrokopter ist ein Unbemanntes Luftfahrzeug (UAV) und stellt eine spezielle Form des Multikopters dar. Sie besitzt vier Rotoren, die in einem Kreuz angeordnet sind. Quadrokopter sind die am häufigsten eingesetzten Multikopter, da sie durch ihre Bauform kompakt und leicht zu steuern sind.

2.1.1 Bauteile

Im Folgenden werden die relevanten Hardwarekomponenten und Sensoren eines Quadrokopters erläutert, um ein grundlegendes Verständnis zu vermitteln. Die Tabelle 2.1 stellt die Hardwarekomponenten vor, die Tabelle Tabelle 2.2 die Sensoren.

Tabelle 2.1: Hardwarekomponenten eines Quadrokopters

Name	**Beschreibung**
Frame	Der Frame ist der physische Körper einer Quadrokopters und hält alle Hardwarekomponenten und Sensoren zusammen
Akkumulator	Der Akkumulator versorgt die elektrischen Bauteile mit Strom
Motoren und Propeller	Ein Quadrokopter besitzt vier Motoren, auf denen je ein Propeller platziert ist. Die Motoren sind in einem Viereck angeordnet, wobei sich jeweils zwei diagonal gegenüberliegende Propeller im Uhrzeigersinn drehen und die anderen beiden Propeller gegen den Uhrzeigersinn
Telemetriemodul	Das Telemetriemodul dient zu Übertragung von Signalen und wird verwendet, um Nachrichten mit der Fernsteuerung auszutauschen. In der Regel werden zur Übertragung Radiofrequenzen von 2,4 GHz oder 5 GHz verwendet.
Flight controller	Der Flight controller führt autonome Aktionen aus, die den manuellen Flug vereinfachen. Zu den Funktionen gehören unter anderem eine Flugstabilisation oder eine Schwebekorrektur.

Tabelle 2.2: Sensoren eines Quadrokopters

Name	**Beschreibung**
Gyroskop	Ein Gyroskop stellt ein Messinstrument dar, das die räumliche Ausrichtung des Körpers einer Drohne erfasst. In einem Quadrokopter sind üblicherweise drei Gyroskope verbaut, die zu den x-, y- und z-Achse des Frames ausgerichtet sind. Das Gyroskop unterstützen den Quadrokopter bei der Orientierung im Raum und der Stabilisierung während des Fluges.
Beschleunigungssensor	Der Beschleunigungssensor erfasst die lineare Beschleunigung entlang einer bestimmten Achse. Bei einem Quadrokopter sind in der Regel drei Beschleunigungssensoren eingebaut, die orthogonal zur x-, y- und z-Achse ausgerichtet sind.
Distanzsensor	Ein Distanzsensor, der an der Unterseite des Frames angebracht wird, hilft bei der Höhenmessung und misst den Abstand zwischen der Drohne und dem darunter liegenden Objekt.
Barometer	Ein Barometer wird zur Messung des Luftdrucks verwendet, um den Höhenbereich zu bestimmen, in dem sich der Quadrokopter befindet.
Inertial Measurement Unit (IMU)	Die inertial Measurement Unit besteht aus mehreren Beschleunigungssensoren und Gyroskopen, die orthogonal zu allen drei Achsen ausgerichtet sind. Sie dient dazu, die Winkel- und lineare Beschleunigung der Drohne zu bestimmen. Die IMU wird vom Quadrokopter verwendet, um sich in der Luft zu stabilisieren und ihre Position zu halten.
Magnetometer	Ein Magnetometer misst das Magnetfeld der Erde. Damit kann der Quadrokopter seine Ausrichtung gegenüber dem Nordpol bestimmen.

2.1.2 Bewegungsmodellierung des Quadrokopters

In diesem Abschnitt wird eine kurze Einführung in die theoretische Grundlage der Bewegungsmodellierung eines Quadrokopters gegeben. Zur Beschreibung der Bewegung müssen zwei Bezugssysteme eingeführt werden, die in Abbildung 2.1 dargestellt sind.

Das erste Bezugssystem wird als World Frame bezeichnet und beschreibt einen geografischen Bezugspunkt, auf den sich ein Körper in seiner Bewegung bezieht. Ausgehend von diesem Bezugspunkt wird ein Koordinatensystem mit den Achsen x_w, y_w und z_w gespannt. Das Bezugssystem ermöglicht es, die Position des Körpers in dem Koordinatensystem zu bestimmen und kann hierbei frei gewählt werden und beispielsweise die Mitte eines Raumes sein.

Das zweite Bezugssystem wird als Body Frame bezeichnet und beschreibt ein Koordinatensystem, das mit einem Körper verbunden ist. Der Body Frame bewegt sich relativ mit dem Körper und beschreibt seine Ausrichtung ([14], S.40).

In einem Quadrokopter wird der Bezugspunkt im Zentrum des Frames (Gehäuse) gesetzt. Relativ zu diesem Bezugssystem besitzt ein Quadrokopter einen sechs dimensionalen Bewegungsraum, der sich aus den drei Achsen des Bezugssystems x_b, y_b und z_b und den drei Rotationsrichtungen Pitch (θ), Roll (ϕ) und Yaw (ψ) zusammensetzt [12].

Mit dieser Einführung wird ein grundlegendes Verständnis vermittelt, um die Beziehung zwischen der Bewegung des Quadrokopters in Bezug auf ein World Frame und seinem Body Frame zu verstehen.

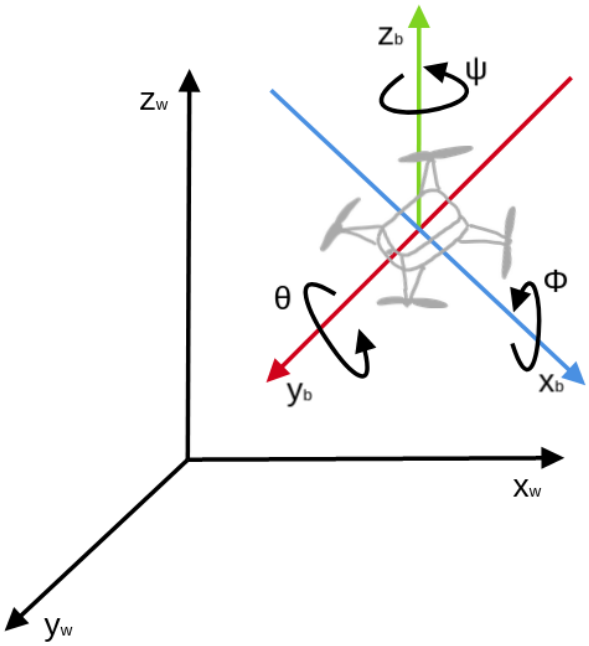


Abbildung 2.1: Bezugssystem und Bewegungsraum eines Quadrokopters

2.2 Digitaler Zwilling

Das Konzept des digitalen Zwillings entstammt einer Präsentation, die im Jahr 2002 von Michael Grieves unter dem Namen „Product Lifecycle Management“ eingeführt wurde [9] und wird seither in unterschiedlichen Domänen angewendet.

Obwohl die Verwendung des DZs sich vielseitig entwickelt hat, ist das Konzept im Laufe der Jahre gleich geblieben und beschreibt grundsätzlich eine „digitale Kopie eines physischen Objektes und dessen Prozesse“([21], S.3).

Bei dem physischen Objekt oder Physischer Zwilling (PZ) kann es sich hierbei um ein einzelnes Produkt handeln, wie zum Beispiel ein Fahrzeug, eine Anlage oder ein Roboter. Dieser kann aber auch ein komplexes System sein, welches aus einem Verbund von Sensoren und Aktoren besteht, wie beispielsweise eine Fabrik, ein Verkehrsnetz oder dem Stromnetz.

In der Literatur wird der PZ häufig als ein zweiteiliges System beschrieben, bestehend aus

- Einer *physischen Entität*, welches das abzubildende Objekt oder das System ist
- Einem *physische Raum*, welcher den realen Raum darstellt, in welchem die physische Entität existiert und von der sie beeinflusst wird

Dementsprechend wird der DZ auch in zwei Teilen aufgeteilt:

- Eine *virtuelle Entität*, welche die virtuelle Repräsentation des physischen Zwillings darstellt und seine Attribute und Prozesse widerspiegelt
- Ein *virtueller Raum* welcher folglich das Gegenstück zum physische Raum ist und seine physischen Eigenschaften abbildet

Beide System sind über eine bidirektionalen Kommunikation verbunden und tauschen in Echtzeit ihre Zustände aus. Dieser Prozess wird als *Twinning* bezeichnet. Dadurch sind sie voneinander nicht zu unterscheiden und agieren als ein System. Die Abbildung 2.2 zeigt die Beziehung zwischen dem digitalen und dem physischen Zwilling.

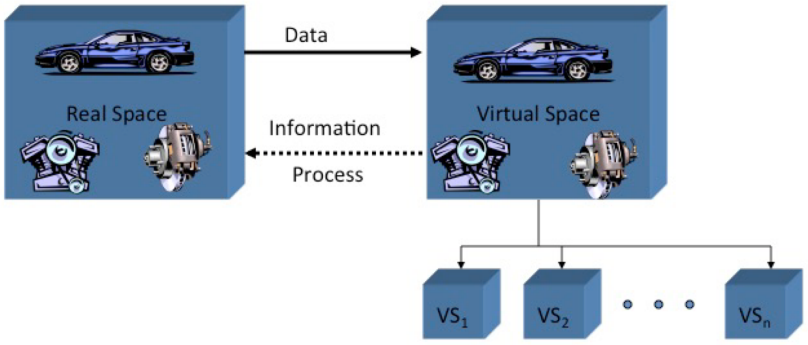


Abbildung 2.2: Informationsaustausch zwischen dem virtuellen und realen Raum [9]

2.2.1 Formen des digitalen Zwillings

Ein DZ kann in unterschiedlichen Lebensabschnitten des physischen Zwillings eingebettet werden [9]. In der Literatur wird unter zwischen unterschiedlichen Lebensphasen des DZ unterschieden

- Digital twin Prototype (DTP): Ein solcher DZ wird vor der Herstellung des physischen Zwillings entwickelt, um Anwendungen und Testfälle zu erproben, bevor sie an einem echten Objekt getestet werden.
- Digital Twin Instance (DTI): Der digital Twin Instance beschreibt ein DZ, welcher mit seinem physischen Zwilling verbunden ist und sich gegenseitig Informationen austauschen.

2.2.2 Anwendungensgebiete

Der DZ findet Anwendung in verschiedenen Bereichen. In der Produktion wird der DZ genutzt, um Produktionsketten zu modellieren, um einzelne Schritte nachzuvollziehen und ihre Effizienz zu überprüfen [16]. In der Städteplanung können DZ in unterschiedlichen Domänen eingesetzt werden, wie zum Beispiel zur Darstellung des Verkehrsaufkommens von Stadtteilen, der Kriminalitätsrate oder der demografischen Verteilung. Durch die Zusammensetzung dieser Informationen kann ein Gesamtbild der Städteentwicklung ent-

stehen und Städteplanern wertvolle Informationen für zukünftige Projekte liefern [16]. Im Gesundheitswesen wird der DZ verwendet, um beispielsweise die Charakteristiken von Medikamenten zu modellieren und ihre Wirkungen zu analysieren und zu verstehen. Dadurch können die individuellen Bedürfnisse des Patienten bei der Planung medizinischer Abläufe besser berücksichtigt werden [16].

2.2.3 Entwicklung digitaler Zwillinge für UAVs

In den vergangenen Jahren wurden verschiedene Architekturen zur Entwicklung von DZs für UAVs präsentiert. Die Ergebnisse wurden in einer Reihe von Forschungsarbeiten publiziert, darunter von Yang et al. [22], Meng et al. [18], Lei et al. [15] und Ji et al. [13]. Eine Gemeinsamkeit der vorgestellten Architekturen besteht darin, dass das System in ein virtuelles Modell des physischen UAVs, in ein kognitives Modell und einer Simulationsplattform unterteilt wird. Das virtuelle Modell enthält eine detaillierte Modellierung der technischen Spezifikationen des UAVS, um das physische Verhalten im virtuellen Raum zu modellieren. Das kognitive Modell umfasst die Funktionen, die den virtuellen Zwilling „intelligent“ machen. Dazu gehören die Ausführung und Überwachung von Aufgaben und die Auswahl des nächsten Ausführungsschritts basierend auf den aktuellen Zustand des virtuellen Modells. Die Simulationsplattform stellt den Raum dar, in der das virtuelle Modell eingesetzt wird und simuliert die Umgebungseigenschaften und dessen Objekte im Raum

2.3 Multiagentensystem

Ein Multiagentensystem (MAS) ist ein Paradigma zur Modellierung für komplexe Systeme, bei dem autonome Software-Agenten eingesetzt werden, um ein komplexes Problem in kleinere Teilaufgaben zu zerlegen [6]. Dabei spielt die spezifische Form und der Anwendungszweck des Systems eine untergeordnete Rolle. Sofern mehrere Agenten im System eingesetzt werden, wird das System als MAS klassifiziert [10]. Trotz der Verwendung eines einzigen Agenten wird dieses System dennoch als MAS betrachtet, da es Funktionen bietet, die nicht nur von einem, sondern auch skalierbar, von mehreren Agenten genutzt werden können.

2.3.1 Agent

Ein Agent wird als ein autonomer Akteur definiert, der innerhalb einer definierten Umgebung lebt. Der Agent hat die Fähigkeit, sowohl seine Umgebung als auch andere Agenten wahrzunehmen und mit ihnen zu interagieren [10]. Dabei besitzt er individuelle Eigenschaften, Verhaltensweisen und Handlungsmöglichkeiten. Der Agent ist zudem in der Lage, aus seinen Interaktionen und vergangenen Erfahrungen zu lernen, indem er Informationen sammelt und neue Erkenntnisse gewinnt [17]. Dieses erworbene Wissen beeinflusst seine zukünftigen Entscheidungen und Handlungen, was zu einer adaptiven Verhaltensweise führt.

2.3.2 MARS-Framework

Das Multi-Agent Research & Simulation (MARS)-Framework ist eine Ansammlung von Werkzeugen zur Entwicklung von umfangreichen MAS [11] in der Programmiersprache C#. Das MARS-Framework ist ein laufendes Projekt der MARS-Group an der Hochschule für angewandte Wissenschaften in Hamburg. Das Hauptziel des MARS-Frameworks besteht darin, die Entwicklung von geografische Bereichen der Erde in agentenbasierten Simulationen zu unterstützen, um neue Erkenntnisse zu gewinnen.

Als Nächstes werden die Konzepte des MARS-Frameworks in vereinfachter Form präsentiert. Für eine ausführliche Einführung und Erläuterung der Funktionalitäten wird die frei zugängliche Dokumentation [1] auf der Webseite der MARS-Group empfohlen.

Ein wichtiges Merkmal des MARS-Frameworks besteht darin, dass Simulationen als tickbasierte Ausführung umgesetzt werden. Der Begriff *tickbasiert* beschreibt ein Konzept, in der die Simulation in eine endliche Anzahl von Schritten (Ticks) unterteilt wird. Ein Tick repräsentiert hierbei einen festen Zeitabschnitt. Dementsprechend erstreckt die Ausführung der gesamten Simulation sich über einen festen Zeitraum hinweg. Innerhalb eines Ticks verändert sich die Simulation in einem zeitlich sinnvollen Verhältnis.

Eine Simulation im MARS-Framework setzt sich zusammen aus den drei Komponenten: Agent, Environment und Layer.

Agent
In einer Simulation repräsentieren *Agenten* autonome Akteure, die sich frei und zielgerichtet in der Umgebung bewegen können. Agenten können mit ihrer Umgebung interagieren und Aktionen ausführen, um ihrem vorgegebenen Ziel schrittweise näherzukommen. Ein Agent setzt sich zusammen aus einem Verhalten, das beschreibt, wie der Agent sich innerhalb der Simulation verhält und seinen Attributen, die das Verhalten auf verschiedenen Arten beeinflussen.

Die Attribute können auf verschiedene Weise konfiguriert werden. Entweder werden die Werte der Attribute fest in der Agentenklasse vorgegeben oder sie können über eine Konfigurationsdatei, hier als *config.json*-Datei bezeichnet, vor dem Beginn der Simulation übergeben werden. Der Agent lädt die Werte für seine Attribute aus der Konfigurationsdatei. Letzteres Verfahren erlaubt eine flexible und schnelle Anpassung der Attribute.

Der Agent implementiert zwei essenzielle Methoden. Die erste Methode, genannt `Init`, wird verwendet, um den Agenten das Layer zuzuweisen, auf dem er lebt. In dieser Methode können außerdem letzte Konfigurationen am Agenten vorgenommen werden. Die zweite Methode ist die *Tick*-Methode, welches das spezifische Verhalten des Agents vorgibt. Führt die Simulation einen neuen Tick aus, wird am Agenten diese Methoden aufgerufen.

Ein Agent besitzt ein Bearing (α), mit der seine Ausrichtung relativ zu seiner Umgebung in Grad angegeben wird. Die Bearing wird in verschiedenen Methoden verwendet, um beispielsweise die Bewegungsrichtung des Agenten vorzugeben. Die Darstellung der Bearing eines Agenten wird durch das Polarkoordinatensystem in Abbildung 2.3 veranschaulicht. Als Standardwert wird der Agent mit einer Bearing von 0,0 initialisiert, was einer Ausrichtung nach Norden entspricht. Eine Drehung im Uhrzeigersinn führt dazu,

dass das Bearing ansteigt und bei einer Gradzahl von 359 endet, um dann wieder bei 0,0 zu beginnen.

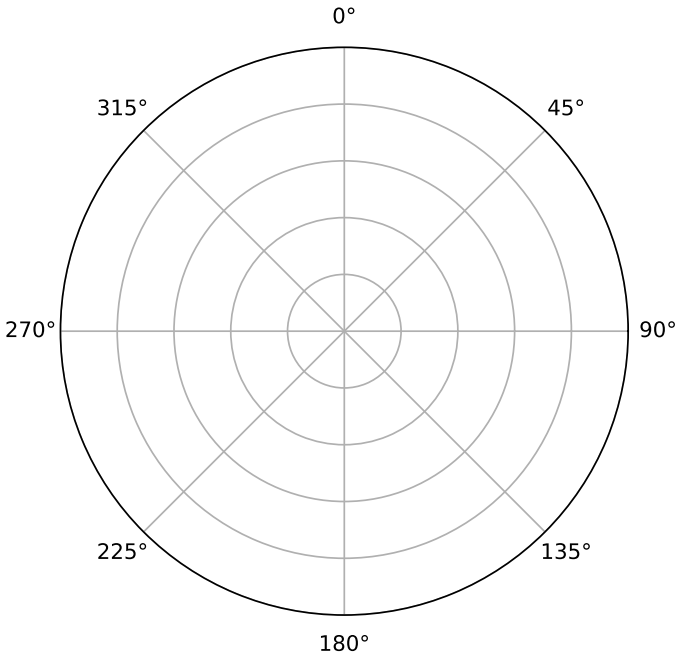


Abbildung 2.3: Darstellung der Bearing im MARS-Framework

Environment
Das *Environment* beschreibt die räumliche Ausdehnung der simulierten Umgebung und legt die Art der Traversierung fest, mit der sich ein Agent innerhalb in der Umgebung bewegen kann. In dem Framework werden vier verschiedene Environment-Typen angeboten:

- SpatialHashEnvironment: Der simulierte Raum wird in einer zwei-dimensionalen Matrix dargestellt. Der Agent kann sich innerhalb des Feldes in der Zeilen und Spalten auf ganzzahlige Koordinatenpunkte bewegen.
- GeoHashEnvironment: Die Umgebung wird in geografische Punkte unterteilt, die in Breiten- und Längengrade angegeben sind. Der Agent ist nicht mehr durch konkrete ganze Koordinatenpunkte beschränkt, sondern kann sich zwischen den Punkten bewegen.

- SpatialGraphEnvironment: Die Umgebung spezifiziert an bestimmten Punkten im Raum Knotenpunkte, die untereinander durch Kanten verbunden sind. Der Agent kann nur zwischen den Knotenpunkten wandern und kann dies ausschließlich über die vorhandenen Kanten tun.
- CollisionEnvironment: Das CollisionEnvironment beschreibt, wie das GeoHashEnvironemnt die Umgebung in Breiten- und Längengraden. Zusätzlich führt dieses Environment das Konzept von Hindernissen ein, die die Bewegung des Agenten einschränken können. Diese Hindernisse können mit verschiedenen Kollisionseigenschaften belegt werden, die den Agenten ermöglichen, auf unterschiedliche Weise mit ihnen zu interagieren.

Layer

Ein *Layer* repräsentiert eine Informationsschicht, in der spezifische Eigenschaft der Umgebung beschrieben werden, die von Agenten wahrgenommen und mit denen sie interagieren können. Diese Eigenschaften können in vielfältiger Form auftreten. Das MARS-Framework erlaubt es gleichzeitig mehrere Layer in einer Simulation zu verwenden, um komplexe Modelle der Umgebungen darzustellen.

3 Technische Ausstattung

In diesem Kapitel werden die Hilfsmittel vorgestellt, die in dieser Arbeit eingesetzt werden. Zunächst wird der verwendete Quadrokopter vorgestellt und seine technischen Spezifikationen kurz erläutert. Im Anschluss erfolgt die Vorstellung des API-Wrappers, welcher die Grundfunktionen der Tello implementiert. Dazu werden seine Funktionalitäten und sein interner Aufbau erklärt.

3.1 Verwendete Drohne

Im Rahmen der Entwicklung des geplanten Drohnensystems ist es erforderlich, eine Drohne auszuwählen, die eine umfangreiche Programmierschnittstelle bereitstellt, eine Vielzahl von integrierten Sensoren besitzt, deren Daten abgerufen werden können und zugleich eine kompakte Größe hat. In vielen Foren wird die Tello des Herstellers Ryze empfohlen, da sie die genannten Anforderungen erfüllt. Insbesondere besteht die Möglichkeit, dieses Modell an der Hochschule auszuleihen, weshalb diese Option gewählt wurde.

Die Tello ist ein Miniquadrokopter mit einem Gewicht von 80 g und besitzt eine Abmessung von 98 mm x 92,5 mm x 41 mm. Aufgrund ihrer geringen Größe wird diese Drohne nach der EU-Drohnenverordnung[8] als Minidrohne eingestuft und erfordert in der Verwendung keine Drohnenlizenz. Die Abbildung 3.1 zeigt ein Porträt der Tello.

Abbildung 3.1: Foto einer Tello

3.1.1 Interne Komponenten

Die Tello ist ausgestattet mit diversen Sensoren, darunter einem Gyroskop, einem Barometer, einem Distanzsensor, einem Temperatursensor und einem Beschleunigungssensor. Der interne Flug Controller verwendet diese Sensoren, um Grundfunktionalitäten bereitzustellen. Außerdem ist eine 720p-Kamera verbaut, über die eine Videoübertragung realisiert werden kann.

3.1.2 Programmierschnittstelle

Für Entwickler, die eine eigene Anwendungen mit der Tello entwickeln wollen, stellt Ryze ein Software Development Kit (SDK) bereit [3]. In diesem SDK wird eine Application Programming Interface (API) angeboten, über der einzelne Befehle versendet, Fluginformationen sowie Videodaten empfangen werden können. Das verwendet Basismodell Tello auf basiert der ersten Version des SDKs und verfügt daher nicht über alle Funktionen.

Kommunikationsarchitektur

Um sich mit der Tello in Verbindung zu setzt, wird das IEEE-802.11-Protokoll verwendet. Nach dem Einschalten der Tello öffnet der Quadrokopter ein öffentliches Netzwerk und erlaubt über eine 2,4GHz Frequenz Signale auszutauschen. Endgeräte können sich daraufhin mit der Drohne verbinden.

Über drei unterschiedliche UDP-Ports können Nachrichten mit der Tello ausgetauscht werden:

- Auf Port 8889 werden Befehle an die Tello gesendet und Statusantworten empfangen.
- Über Port 8890 sendet die Tello in periodischen Abständen seine Fluginformationen aus.
- Über Port 11111 werden die Bilddaten der Drohnenkamera in Echtzeit übertragen.

Steuerbefehle

Die Steuerbefehle werden in Form von Textbefehlen über asynchrone Nachrichtenpakete an die Tello übermittelt. Die meisten Steuerbefehle können ohne Blockierung des Prozesses versendet werden. Es gibt jedoch bestimmte Befehle, auf deren Antwort gewartet wird, die den Ausführungsstatus zurückgibt. Ein Beispiel dafür ist der Befehl `TakeOff`, bei dem der Sender durch den Ausführungsstatus erfährt, ob die Drohne erfolgreich abgehoben oder ein Fehler aufgetreten ist. In der Dokumentation [2] ist eine vollständige Auflistung aller verfügbaren Befehle für die Tello zu finden.

Fluginformationen

Die Tello veröffentlicht in einem Abstand von 100 ms seine Fluginformationen als String über im Unterunterabschnitt 3.1.2 genannten Port. Die Tabelle A.1 im Anhang zeigt alle übertragenen Fluginformationen in einer Liste. Da nicht alle Parameter in den Fluginformationen verwendet werden, sind die wichtigsten Parameter für diese Arbeit in einer weiteren Tabelle 3.1 dargestellt.

Tabelle 3.1: Verwendete Fluginformationen der Tello

Symbol	**Beschreibung**	**Einheit**
ϕ	Neigungswinkel um die x-Achse	$^\circ$
θ	Neigungsinkel um die y-Achse	$^\circ$
ψ	Neigungswinkel um die z-Achse	$^\circ$
vel_x	Fluggeschwindigkeit in x-Achse	$10^{-2}\frac{m}{s}$
vel_y	Fluggeschwindigkeit in y-Achse	$10^{-2}\frac{m}{s}$
h	Flughöhe gemessen vom Ausgangspunkt	$10^{-2}m$
bat	Batteriestand des Akkus	%
a_x	Beschleunigung in x-Achse	$10^{-2}\frac{m}{s^2}$
a_y	Beschleunigung in y-Achse	$10^{-2}\frac{m}{s^2}$

In dem Abschnitt wurden die Rotationswinkel θ, ϕ und ψ eingeführt. Eine Besonderheit der Tello ist, dass diese Rotationswinkel nicht in einem Bereich von 0° bis 360° angegeben werden, sondern, wie in Abbildung 3.2 zu sehen ist, im Bereich von -179° bis 179° angegeben werden. Darüber hinaus gibt es keine Winkelangabe für den gestreckten Winkel. Wenn die Tello eingeschaltet wird, werden alle Rotationswinkel auf 0° initialisiert.

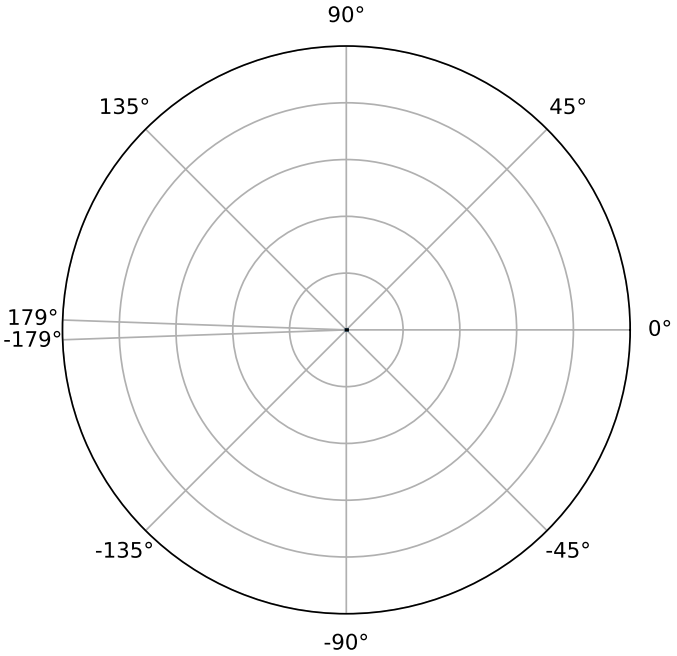


Abbildung 3.2: Darstellung der Yaw der Tello

3.2 C#-Wrapper

Für die Entwicklung des Drohnensystems wird der öffentliche C#-Wrapper von Eloncase [7] verwendet. Die RyzeTelloSDK(Wrapper) nutzt die zweite Version des SDKs und deckt einen Großteil der Funktionen der API ab. Die Verwendung eines Wrappers verkürzt die Entwicklungszeit.

Das Klassendiagramm in Abbildung 3.3 zeigt den Aufbau des Wrappers. Die Hauptklassen der RyzeTelloSDK sind die `TelloClient`, der `StateServer` und der `StateVideoServer`. Diese Klassen greifen auf die einzelnen UDP-Ports zu, aus Abschnitt 3.1.2 genannt sind. Der TelloClient nutzt auf den Port 8889 und implementiert Methoden, um Steuerbefehle entgegenzunehmen und sie zu versenden. Der `StateServer` hört auf Port 8880 und wartet auf neue Fluginformationen. Wenn neue Parameter eintreffen, überführt die Klasse die Parameter in das StateParameter Datenmodell. Der `StateVideoServer` ist verantwortlich für das Empfangen und Anzeigen der Videoübertragung

und hört dementsprechend auf den Port 11111 ab. Diese Klassen werden von der Klasse ConsoleWorker gehalten. Der `ConsoleWorker` implementiert eine Tasteneingabe für die Annahme von Steuerbefehlen und eine Konsolenausgabe für die Darstellung der empfangenen Flugparameter.

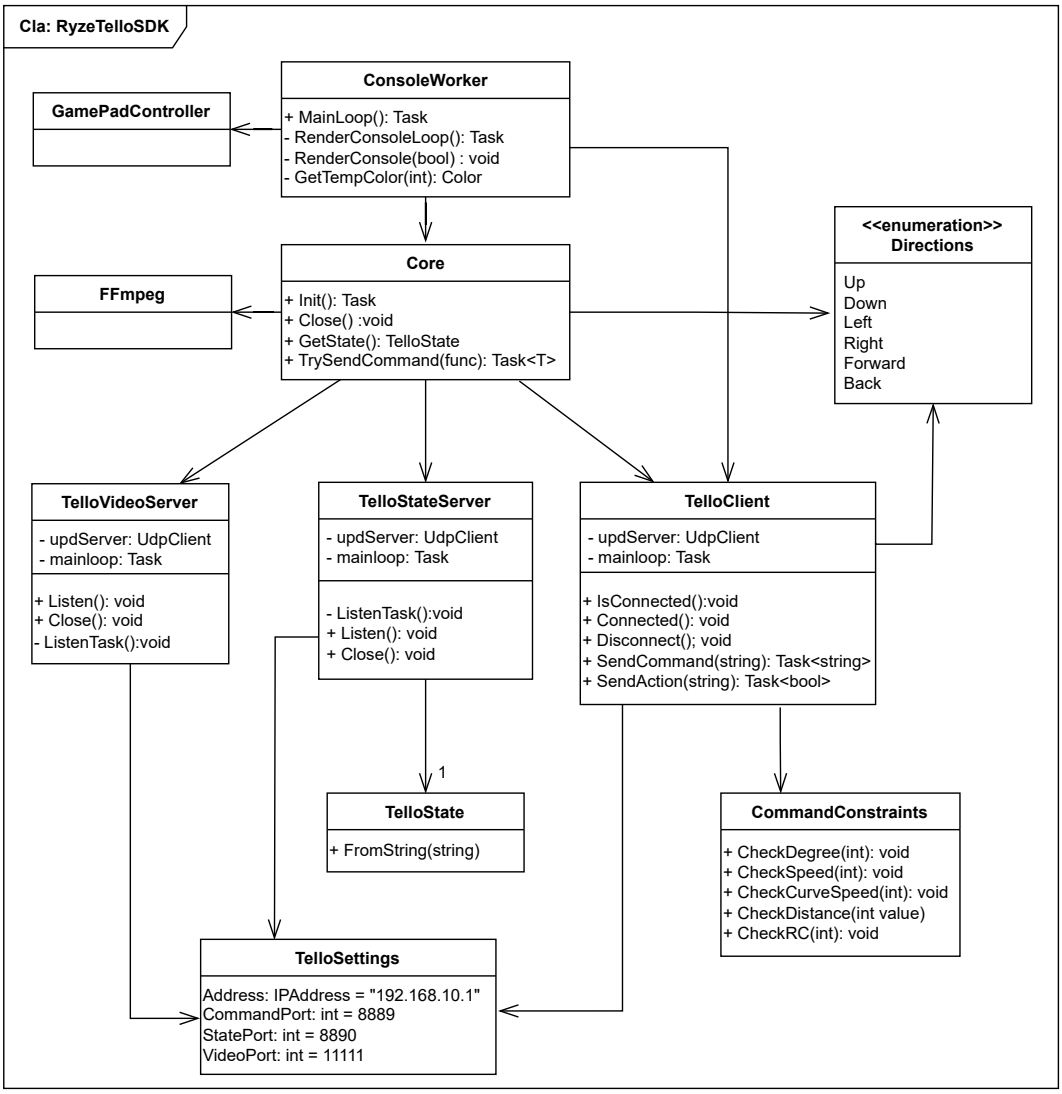


Abbildung 3.3: Klassendiagramm des RyzeTelloSDKs

4 Anforderungsanalyse

In diesem Abschnitt werden die funktionalen und nichtfunktionalen Anforderungen für dieses System präsentiert. Sie dienen der späteren Konzeption und Implementierung als Vorgabe und Verifikation des Systems. Die gestellten Anforderungen stützen sich auf die gesetzten Ziele und die durch die vorgestellten Materialien vorgegebenen Limitationen. Außerdem werden die Anforderungen für die zu entwickelnde zusätzliche Funktionalität präsentiert.

4.1 Funktionale Anforderungen

Tabelle 4.1: Funktionale Anforderungen

ID	**Kategorie**	**Beschreibung**
F01	Digitaler Zwilling	Das Drohnensystem umfasst einen DZ für die Tello.
F02	Digitaler Zwilling	Der DZ empfängt fortlaufend Fluginformationen der Tello.
F03	Digitaler Zwilling	Der DZ bildetet die Trajektorie der Tello ab, sobald neue Fluginformationen empfangen werden.
F04	Digitaler Zwilling	Durch die Analyse der Fluginformationen soll der Flugzustand des Tello ermittelt werden.
F05	Digitaler Zwilling	Über Steuerbefehle kann der DZ die Tello steuern.
F06	Digitaler Zwilling	Vor dem Programmstart können Konfigurationsparameter an den DZ übergeben werden.

F07	Fernsteuerung	Über eine Benutzerschnittstelle kann der Fernpilot die Tello manuell steuern.
F08	Fernsteuerung	Die Benutzerschnittstelle nutzt eine Hardwareperipherie um Befehle entgegenzunehmen.
F09	Fernsteuerung	Die Benutzerschnittstelle stellt Fluginformationen der Tello visuell dar.
F10	Kommunikation	Nachrichten an die Tello werden vor dem Versenden validiert.
F11	Kommunikation	Invalide Nachrichten werden verworfen.
F12	Fehlerbehandlung	Wenn ein blockierender Befehl dreimal unbeantwortet bleibt, wird die Tello angewiesen, eine Landung durchzuführen.
F13	Fehlerbehandlung	Falls das Drohnensystem einen Fehler zurückmeldet, wird die Tello zum Landen befohlen.
F14	Fehlerbehandlung	Falls die Batterie der Tello unter 10 % liegt, wird der Fernpilot auf der Anzeige gewarnt.
F15	Protokollierung	Der DZ zeichnet die Trajektorie der Tello auf und speichert sie in einer Datei ab.
F16	Protokollierung	Aufzeichnungen von Trajektorien können in den DZ geladen werden.
F17	Protokollierung	Befehle, die über die Fernsteuerung eingegeben werden, werden vom DZ erfasst und mit einem Zeitstempel gespeichert.
F18	Operation	Der DZ verfügt über eine Funktion, mit der er aufgezeichnete Trajektorien replizieren kann.
F19	Operation	Während eine Operation ausgeführt wird, überprüft der DZ seinen Zustand gegenüber Validationenswerte.

F20	Operation	Falls eine Operation während der Ausführung gegen seine Validation verstößt, bricht der DZ die Operation ab und bleibt an seiner Position schweben.

4.2 Nichtfunktionale Anforderungen

Tabelle 4.2: Nichtfunktionale Anforderungen

ID	**Kategorie**	**Beschreibung**
NF01	Performanz	Berechnungen müssen kurz und performant sein, damit das System nicht verlangsamt wird.
NF02	Performanz	Das System muss reaktionsfähig bleiben und Prozesse dürfen nicht blockiert werden.
NF03	Performanz	Die System darf im Nachrichtenaustausch mit der Tello die Drohne nicht überlasten.
NF04	Performanz	Die Eingabe von Steuerbefehlen über die Benutzerschnittstelle sollten performant ausgeführt werden.
NF05	Zuverlässigkeit	Befehle an die Drohnen müssen überall konsistent sein.
NF06	Korrenktheit	Der digitale Zwilling achtet während der Datenverarbeitung, dass die Informationen aktuell sind.
NF07	Korrenktheit	Die Methoden zur Abbildung des Drohnenzustands müssen ausreichend präzise und konsistent sein.
NF08	Benutzbarkeit	Die Steuerung muss für den Benutzer intuitiv zu verwenden sein.
NF09	Benutzbarkeit	Die Überwachungsanzeige muss für den Benutzer intuitiv zu verstehen sein.

NF10	Sicherheit	Wenn ein unerwarteter Fehlerzustand eintritt, muss das System in der Lage sein, die Tello in einen sicheren Zustand zu überführen.
NF11	Modularität	Das System soll für Entwickler einfach zu erweitern sein.
NF12	Wartbarkeit	Das System sollte austauschbar sein. Ein Entwickler soll Komponente austauschen können, ohne große Änderungen am System vornehmen zu müssen.
NF13	Skalierbarkeit	Das System soll in der Lage sein mit unterschiedlich vielen Agenten arbeiten zu können.

5 Konzeption

In der Konzeption werden die gestellten funktionalen und nichtfunktionalen Anforderungen aus dem vorherigen Kapitel in einen geeigneten Entwurf überführt. Insbesondere ist es wichtig, eine Architektur zu entwickeln, die die Eigenschaften eines MASs berücksichtigt.

Zuerst werden die Akteure vorgestellt, die mit dem Drohnensystem interagieren. Anschließend folgt eine grobe Einführung in den Systementwurf und eine Erklärung der enthaltenen Komponenten. Im Anschluss werden die einzelnen Komponenten detaillierter beschrieben und erörtert. Schließlich wird eine Funktion vorgestellt, bei der die Komponenten in kooperativer Weise zusammenwirken müssen, um ein gemeinsames Ziel zu erreichen.

5.1 Akteuere

Das Komponentendiagramm in Abbildung 5.1 stellt die Akteure des Systems vor und zeigt ihre Beziehungen zueinander.

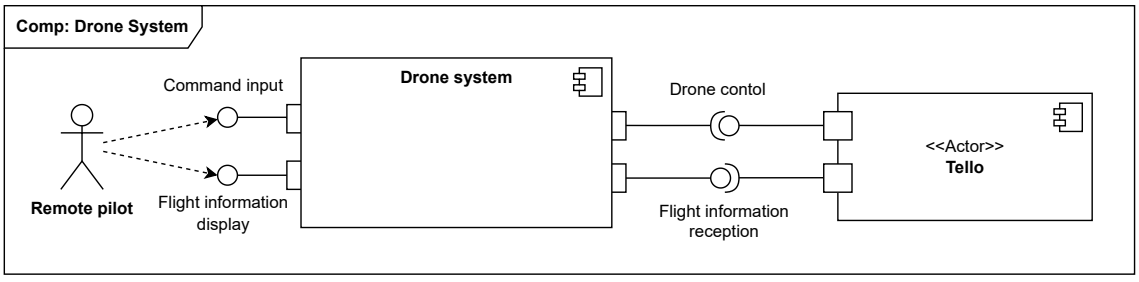


Abbildung 5.1: Akteure des Drohnensystems

Das *Drohnensystem* (Drone system) repräsentiert das zu entwickelnde System und beinhaltet den digitalen Zwilling für die Tello. Es bietet Schnittstellen an, über die Befehle an die Tello gesendet (Drone Contol) oder Fluginformationen empfangen (Flight information reception) werden können. Zusätzlich ermöglicht das Drohnensystem die manuelle Steuerung (Command input) der Tello über eine Eingabeschnittstelle und visualisiert die empfangenen Fluginformationen (Flight information display).

Die *Tello* ist ein eigenständiges passives System, das bis auf wenige Ausnahmen, auf externe Befehle angewiesen ist. Innerhalb des Drohnensystems stellt die Tello den PZ dar, von dem die Eigenschaften und Verhaltensweisen im DZ widergespiegelt werden. Gleichzeitig kann der DZ die Tello durch Befehle steuern, um sie in einen bestimmten Zustand zu überführen.
Die Schnittstelle für das Empfangen der Fluginformationen stellt das Drohnensystem bereit, da die Tello, die Fluginformationen in regelmäßigen Abständen veröffentlicht. Daher muss das Drohnensystem eine Schnittstelle anbieten, damit sie die Fluginformationen abhören kann.

Der *Fernpilot* (Remote pilot) ist ein menschlicher Akteur. Er ist für die Überwachung des Drohnensystems, sowie der Tello zuständig. Über eine visuelle Darstellung der Fluginformationen kann der Fernpilot den Status der Tello überwachen. Über die Eingabeschnittstelle hat der Fernpilot die Möglichkeit, die Tello über manuelle Eingaben zu steuern.

5.2 Grobe Systemarchitektur

Nachdem im vorherigen Abschnitt die Akteure und die Beziehungen zum Drohnensystem vorgestellt wurden, wird nun die grobe Systemarchitektur des Systems vorgestellt. Dabei werden die Aufgaben in logische Komponenten unterteilt und in Beziehung zueinander gesetzt. Eine detaillierte Erläuterung der einzelnen Komponenten folgt in den nächsten Abschnitten.

Das Diagramm in der Abbildung 5.2 veranschaulicht den internen Aufbau des Systems, die aus dem Message Broker, der Fernsteuerung und dem digitalen Zwilling bestehen.

Die *Fernsteuerung* stellt die Funktionen für die Benutzerschnittstelle bereit, auf die der Fernpilot zugreifen kann. Sie bietet neben einer Anzeige für die Fluginformationen, die Schnittstelle, um Befehlseingaben entgegenzunehmen. Diese Eingaben werden von der

Fernsteuerung in konkrete Befehle umgewandelt und an den Message Broker weitergeleitet.

Der *digiale Zwilling* stellt die Komponente dar, in der das virtuelle Modell der Tello und ihrer Umgebung erstellt wird. Außerdem umfasst sie die Funktionen, die für die Abbildung der Daten in das Modell verwendet werden.

Der *Message Broker* fungiert als zentrale Kommunikationsschnittstelle, zu der alle Nachrichten innerhalb des Systems zusammenlaufen. Der Message Broker prüft die Nachrichten und leitet sie an entsprechende Empfänger weiter. Es ist anzumerken, dass der Message Broker eine Schnittstelle bereitstellt, um die Fluginformationen abzurufen, anstatt die Komponenten automatisch zu benachrichtigen, wenn neue Fluginformationen verfügbar sind. Dies dient dazu, den Nachrichtenverkehr zu reduzieren und mögliche Engpässe zu vermeiden.

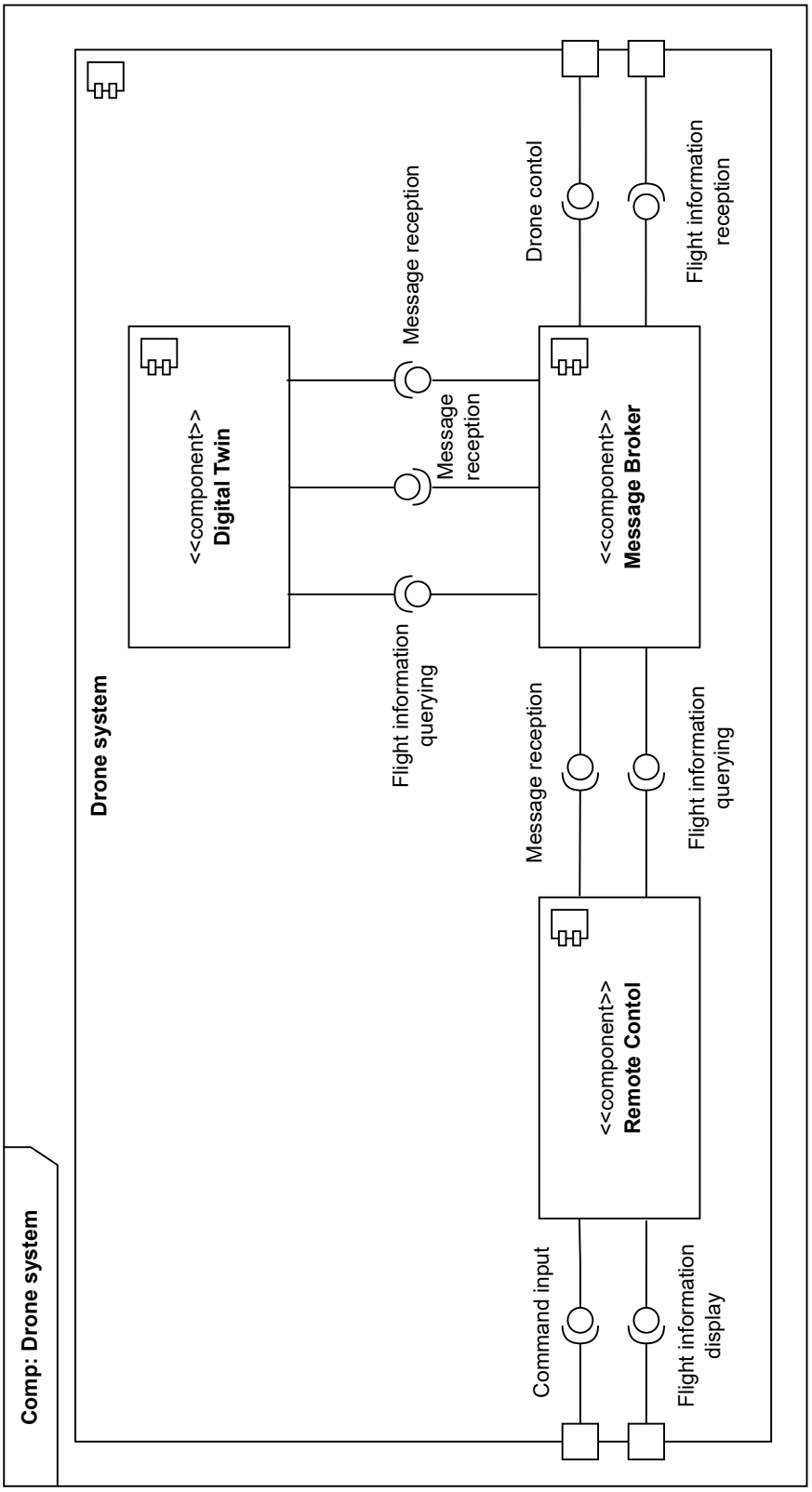


Abbildung 5.2: Komponentendiagramm des Drohnensystems

5.3 Message Broker

Durch die Verwendung eines Nachrichtenbrokers (Message Broker) werden die Komponenten innerhalb des Systems entkoppelt, um ihre Abhängigkeiten zueinander zu minimieren. Dies ermöglicht eine einfachere Modifizierung oder den Austausch einzelner Komponenten, was den Anforderungen NF11 und NF12 der nichtfunktionalen Anforderungen entspricht.

Darüber hinaus verfolgt der Nachrichtenbroker nicht nur das Ziel, die Modularität und Skalierbarkeit zu erhöhen, sondern auch den Anforderungen an die Performanz gerecht zu werden. Eine zentrale Nachrichtenverwaltung ermöglicht das kontrollierte Versenden von Nachrichten. Bei der Analyse der Tello wurde festgestellt, dass sie durch eine Nachrichtenflut nicht mehr reaktionsfähig ist, da sie zunächst alle gesendeten Nachrichten verarbeiten muss. Durch den Einsatz eines Nachrichtenbrokers können Nachrichten in einem geregelten zeitlichen Abstand versendet werden, um eine solche Überlastung zu vermeiden. Darüber hinaus soll der Nachrichtenbroker ungültigen Nachrichten filtern, um das Risiko unerwarteter Verhaltensweisen zu minimieren.

5.3.1 Nachrichtenaustausch innerhalb des Systems

Um zu ermöglichen, dass beliebige Komponenten dem Nachrichtenbroker Nachrichten zuschicken können, wird ein abstraktes Nachrichtenmodell für die interne Kommunikation eingeführt. Das Klassendiagramm in Abbildung 5.3 veranschaulicht den Aufbau einer solchen Nachricht.

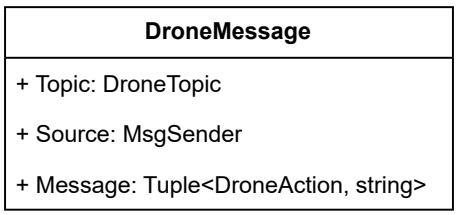


Abbildung 5.3: Klassendiagramm einer DroneMessage

Eine `DroneMessage` soll aus einem Topic, einer Source und einer Message bestehen. Das Topic ermöglicht es dem Empfänger, die für ihn relevanten Nachrichten zu filtern. Über die Source kann der Empfänger den Absender der Nachricht lesen. Diese Infor-

mation ist nützlich, um zu vermeiden, dass ein Sender seine eigene Nachricht empfängt und verarbeitet. Der dritte Teil der `DroneMessage`, die Message, enthält den konkreten Befehl und einen Parameter. Der Befehl kann entweder ein Steuerbefehl sein, der bei der Tello eine Aktion aufruft oder Befehle, um Funktionen innerhalb des Systems auszuführen. Die Darstellung der Nachrichten in einem einheitlichen Nachrichtenmodell ermöglicht die Erweiterung des Gesamtsystems.

Der Nachrichtenbroker übernimmt zusätzlich die Aufgabe der Nachrichtenverteilung innerhalb des Systems, indem ein Modell für den Nachrichtenaustausch eingesetzt wird. In diesem Modell werden Nachrichten nur an die Teilnehmer versendet, für die die entsprechenden Nachrichten relevant sind. Das Verteilen der Nachrichten geschieht über die Filterung der Topics.

Um sicherzustellen, dass sich alle Teilnehmer bei dem gleichen Nachrichtenbroker registrieren, wird dieser als Singleton realisiert.

5.3.2 Nachrichtenaustausch mit der Tello

Wie im Abschnitt 3.1 beschrieben, findet die Interaktion zwischen dem System und der Tello ausschließlich über den Austausch von asynchronen Nachrichten über vorgegebenen UDP-Ports statt. Um zu vermeiden, dass jede Komponente das gleiche Protokoll implementiert, was zu Redundanz und Fehleranfälligkeit führt, soll der Nachrichtenbroker zuständig für den Nachrichtenaustausch mit der Tello sein. Durch das vorher eingeführte Nachrichtenmodell können die Komponenten ihre Befehle dem Nachrichtenbroker übermitteln, woraufhin dieser den konkreten Befehl an die Tello formuliert. Die Möglichkeit den Zugriff auf die Drohne durch eine zentrale Schnittstelle abzulösen, erlaubt es auch, die Schnittstelle zur Tello zu abstrahieren, damit weitere Nachrichtenprotokolle integriert werden und somit verschiedene Drohnenmodelle verwendet werden können.

5.4 Digitaler Zwilling

In diesem Abschnitt wird ein Konzept vorgestellt, wie ein DZ für eine Tello im MAS entwickelt werden kann. Dazu werden zuerst die Merkmale der Architekuren für DZ für UAVs aus dem Grundlagenkapitel und mit den Merkmalen des MAS verglichen. Daraus

soll die Architektur des DZ synthetisiert werden. Daraufhin folgt eine Modellbeschreibung des vorgesehenen MAS.

5.4.1 Herleitung der Architektur

Aus dem dargelegten Forschungsstand im Abschnitt 2.2.3 geht hervor, dass DZ für UAVs aus drei Modellteilen bestehen. Ein virtuelles Modell, das die physische Form und Dynamik des PZ darstellt, ein kognitives Modell, das die Funktionen des virtuellen Modells bereitstellt und ein räumliches Modell, in dem das virtuelle Modell existiert. Unter diesen Aspekten weisen DZ und MAS Gemeinsamkeiten auf. Zum einen gibt es in beiden Systemen das Konzept einer unabhängigen Software-Komponente, die die Eigenschaften eines realen Objektes besitzen. Zum anderen gibt es in beiden Konzepten die Idee eines Raummodells, in der die Eigenschaften der Umgebung abgebildet werden und mit dem das virtuelle Modell interagieren kann.

Diesen beide Merkmale können in eine gemeinsame Architektur überführt werden, in der der Agent in MAS das virtuelle Gegenstück des physischen Objekts darstellt. Analog entspricht die Umgebung des MASs der virtuellen Umgebung des DZs. Das Konzept ist grob in Abbildung 5.4 als Komponentendiagramm dargestellt.

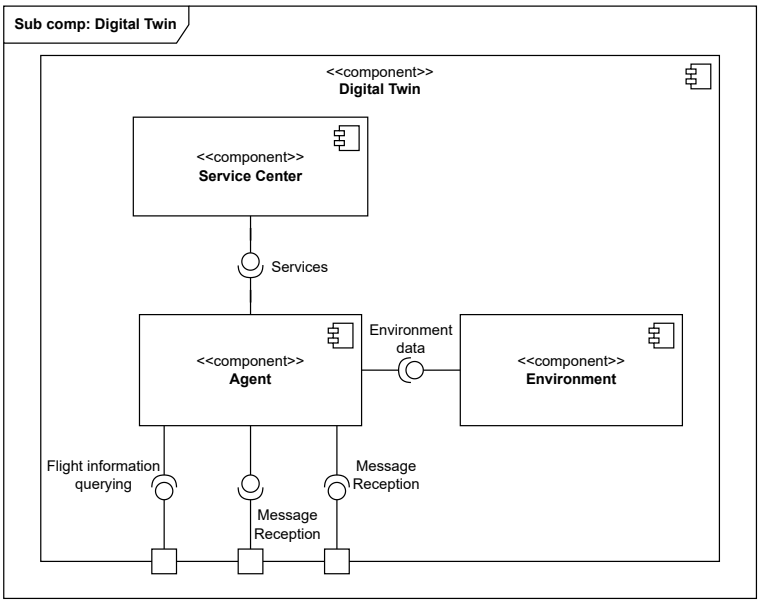


Abbildung 5.4: Komponentendiagramm des digitalen Zwillings

5.4.2 Agent

In diesem System wird ein Agent verwendet werden, der die Tello in der virtuellen Umgebung repräsentiert. Im Fokus dieser Arbeit steht die Nachbildung der Trajektorie der Tello durch den Agenten im virtuellen Raum. Der Agent verwendet die verfügbaren Fluginformationen, um kontinuierlich seine Position anzupassen.

Positionsbestimmung

Eine wichtige Anforderung an den Agenten ist, dass seine Bewegungen nicht nur relativ zum Flugkörper ermittelt werden, sondern auch in Abhängigkeit auf das Bezugssystem, das zu Beginn des Programms durch die Umgebung festgelegt wird. Dies bedeutet, dass der Agent nicht nur die Flugbewegungen entlang der Achse bestimmt, sondern auch die Rotation um die eigene z-Achse abbilden muss. Diese Aufgaben führt der Agenten in seiner Tick-Methode aus. Dazu nutzt der Agent die Fluginformationen, die von der Tello bereitgestellt werden und berechnet aus diesen Daten seinen Standort. Bei jeder neuen Position speichert der Agent seine Koordinaten ab und schreibt sie am Ende der Programmausführung in eine Datei. Da die Tick-Methode iterativ durch MARS aufgerufen wird, ist es dem Agenten nicht möglich, die Fluginformationen selbst abzuhören, da es sonst zur Unterbrechung der Simulation führen würde. Aus diesem Grund soll der Message Broker eine Funktion bereitstellen, bei dem die Fluginformationen abgefragt werden können.

Aktionen des Agenten

Der Agent verfügt über die Möglichkeit, die Tello über vorgegebene Steuerbefehle zu steuern. Allerdings hat der Agent zu Beginn der Programmausführung keine Ziele und führt daher keine Aktionen aus. Das Ziel kann im Laufe des Programmverlaufs jedoch von dem Fernpiloten über die Fernsteuerung übergeben werden. Das Ziel des Agenten ist in Form einer Operation gegeben, auf die in Abschnitt 5.6 weiter eingegangen wird. Durch die Vergabe eines Ziels ändert sich der Agent von einem passiven Akteur in einen aktiven.

Normalerweise werden in MARS Simulationen entwickelt, die einen klar definierten Start- und Endzeitpunkt haben. Dieser ist jedoch in diesem System nicht gegeben, da die Tello

für eine nicht unbestimmte Zeit läuft. Dementsprechend besitzt die Programmausführung kein Ende.

Flugzustandsbestimmung

Das Verhalten der Tello wird anhand von Flugzuständen beschrieben und aus den verfügbaren Fluginformationen abgeleitet. Durch die Bestimmung des Flugzustandes, kann entschieden werden, ob eine Positionsbestimmung notwendig ist. Ein Beispiel ist das Hovering, bei der die Drohne keine Bewegung erfährt und daher keine neue Position bestimmt werden muss. Damit wird die Genauigkeit der Positionsbestimmung verbessert.

5.4.3 Umgebung

Die Umgebung repräsentiert den geografischen Raum, in dem der Agent lebt. In MARS sollte ein Environment gewählt werden, welches die Möglichkeit bietet, Hindernisse zu platzieren, um den realen Raum modellieren zu können. Es ist außerdem erforderlich, dass in dem gewählten Environment die Position aus der zurückgelegten Distanz bestimmt wird. Hindernisse werden in Rahmen dieser Arbeit jedoch nicht eingesetzt. Außerdem werden die Umgebung keine zusätzlichen Layer mit spezifischen Eigenschaften beschrieben, da das Projekt in einem geschlossenen Raum ohne Hindernisse getestet wird.

5.4.4 Service Center

Da die Grundidee der vorgestellten Architektur darin besteht, den Agenten nur als den virtuellen Körper der Tello abzubilden, werden einzelne Funktionalitäten des Agenten in den Service Center ausgelagert. Zu den Funktionalitäten gehören beispielsweise die Berechnungen zur Bestimmung der Position, die Aufzeichnung der Trajektorien oder die Zuordnung der Flugzustände. Dadurch erhöht sich gleichzeitig die Wartbarkeit des Systems.

5.5 Fernsteuerung

Das Komponentendiagramm in Abbildung 5.5 veranschaulicht den Entwurf für die Fernsteuerung. Sie zeigt die Eingabeschnittstelle, mit der der Fernpilot die Tello manuell steuern oder Operationen am DZ aufrufen kann. In dem Cockpit können Fluginformationen abgelesen werden.

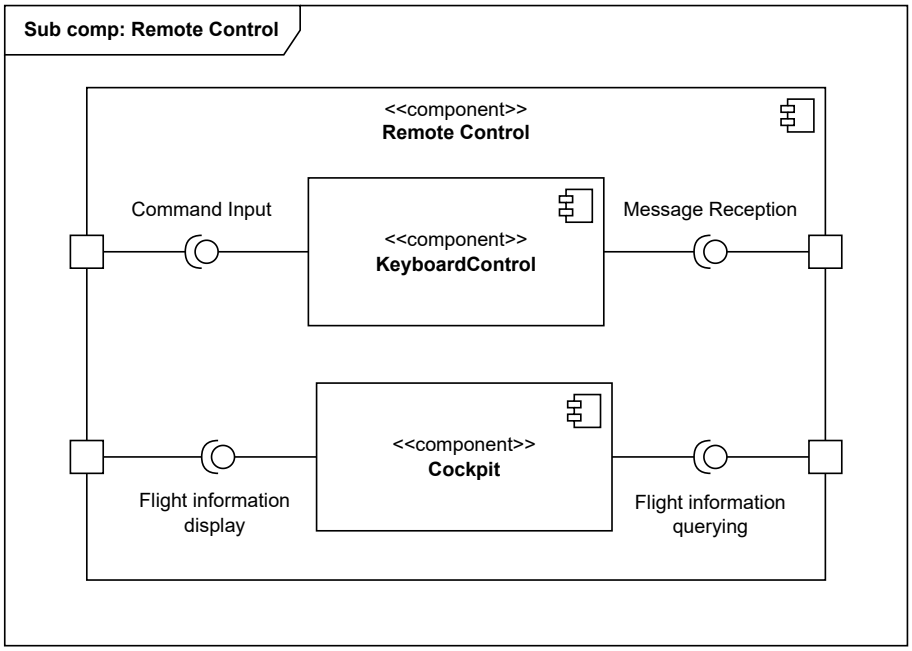


Abbildung 5.5: Komponentendiagramm der Fernsteuerung

Die Fernsteuerung soll über eine Tastaturschnittstelle verfügen, über der die Tello mithilfe von Tasteneingaben gesteuert und Funktionen des DZ aktiviert werden können. Bei der Entwicklung der Eingabeschnittstelle steht die Benutzerfreundlichkeit im Vordergrund. Falls die Tello während des Flugs in eine gefährliche Situation kommen sollte, muss der Fernpilot schnell eingreifen können. Über eine Anzeige soll der Fernpilot die relevanten Fluginformationen einsehen können, um die Interaktion mit der Tello zu erleichtern. Dazu sollen die Flugdaten formatiert dargestellt werden.

5.6 Einführung in Operation

Innerhalb dieser Arbeit wird eine Operation definiert als eine längere Abfolge von Aktionen, die der DZ eigenständig ausführt, um ein bestimmtes Ziel zu erreichen.

Im Zusammenhang mit der Anforderung F18 aus der Anforderungsanalyse 4.1 wird eine Operation entwickelt, bei der der Agent eine aufgezeichnete Trajektorie einlesen und autonom nachstellen soll. Während der Ausführung soll der Agent iterativ seine eigene Position gegenüber einer vorgegebenen Koordinate prüfen, um bei einer Abweichung die Operation zu unterbrechen und so präventiv Kollisionen mit einem Hindernis zu vermeiden.

Diese Operation, die als Record-Repeat Navigation (RRN) bezeichnet wird, vereint die Kernfunktionen der vorher beschriebenen drei Komponenten (Message Broker, Fernsteuerung und Digitaler Zwilling). Die Funktionsweise der Operation wird im Nachfolgenden näher beschrieben.

Die RRN soll sich aus den zwei Phasen Record und Repeat zusammensetzen. In der Record Phase zeichnet das System die Eingaben des Fernpiloten auf, um Befehle an die Tello später replizieren zu können. In der Repeat Phase soll die Aufzeichgung vom Agenten eingelesen werden und bei Aktivierung der Operation durch die Fernsteuerung beginnen, die Flugbahn nachzustellen.

Record-Phase

Die Record-Phase soll mit dem Start des Programms beginnen. In dieser Phase werden alle Befehle, die der Fernpilot über die Fernsteuerung eingibt, vom System aufgezeichnet. Die Befehle werden an den Agenten übermittelt, der zusätzlich zu jedem Befehl seine aktuelle Position und einen Zeitstempel angibt. Diese Position wird in der nächsten Phase zur Validierung verwendet. Die Aufzeichnungen sollen so lange erstellt werden, bis der Fernpilot den Befehl zur Beendigung der Aufzeichnung gibt. Daraufhin werden alle Einträge der Aufzeichnung in einer CSV-Datei geladen und abgespeichert.

Repeat-Phase

Um die Replikation durchführen zu können, muss die Aufzeichnung, ausgewählt und beim Programmstart in den Agenten geladen werden. Die Repeat-Phase beginnt mit der Eingabe eines Befehls über die Tastensteuerung. Daraufhin startet der Agent die Ausführung der Befehle. Bei jedem ausgeführten Befehl berechnet der Agent die Wartezeit für den nächsten Befehl, indem die Differenz zwischen dem Zeitstempel des aktuellen und des nächsten Befehls berechnet wird. Damit ermittelt der Agent, wie lange ein Befehl ausgeführt werden soll und aktiviert den nächsten Befehl, wenn die Wartezeit abgelaufen ist.

Bevor der Agent einen neuen Befehl ausführt, validiert der Agent seine Position mit der vorgegebenen Koordinate des Befehls. Hierbei wird ein Validationsradius verwendet, der den maximalen Abstand zu der vorgegebenen Koordinate angibt. Sollte der Agent außerhalb des zulässigen Gültigkeitsbereichs liegen, wird die Tello umgehend zum Anhalten befohlen und die Operation wird terminiert. Andernfalls führt der Agent die Operation so lange aus, bis alle Befehlsschritte erfolgreich ausgeführt wurden.

6 Implementation

Nachdem im vorherigen Abschnitt die Anforderungen analysiert und in ein Konzept überführt wurden, befasst sich dieses Kapitel mit der technischen Umsetzung des Entwurfes. Dazu werden die wichtigsten Aspekte der vorher eingeführten Komponenten beschrieben.

6.1 Message Broker

In Abbildung 6.1 ist der Message Broker als Klassendiagramm dargestellt. Im Zentrum des Diagramms steht der `TelloMessageBroker` ein und übernimmt die Koordination der Kommunikation im implementierten System. Mithilfe einer Queue werden alle Nachrichten gebündelt gesammelt und verarbeitet. Die Klassen um den `TelloMessageBroker` stellen die wichtigsten Funktionen bereit, die zur Erfüllung der beschriebenen Anforderung im Konzept erforderlich sind und werden in den nächsten Unterkapiteln genauer erläutert.

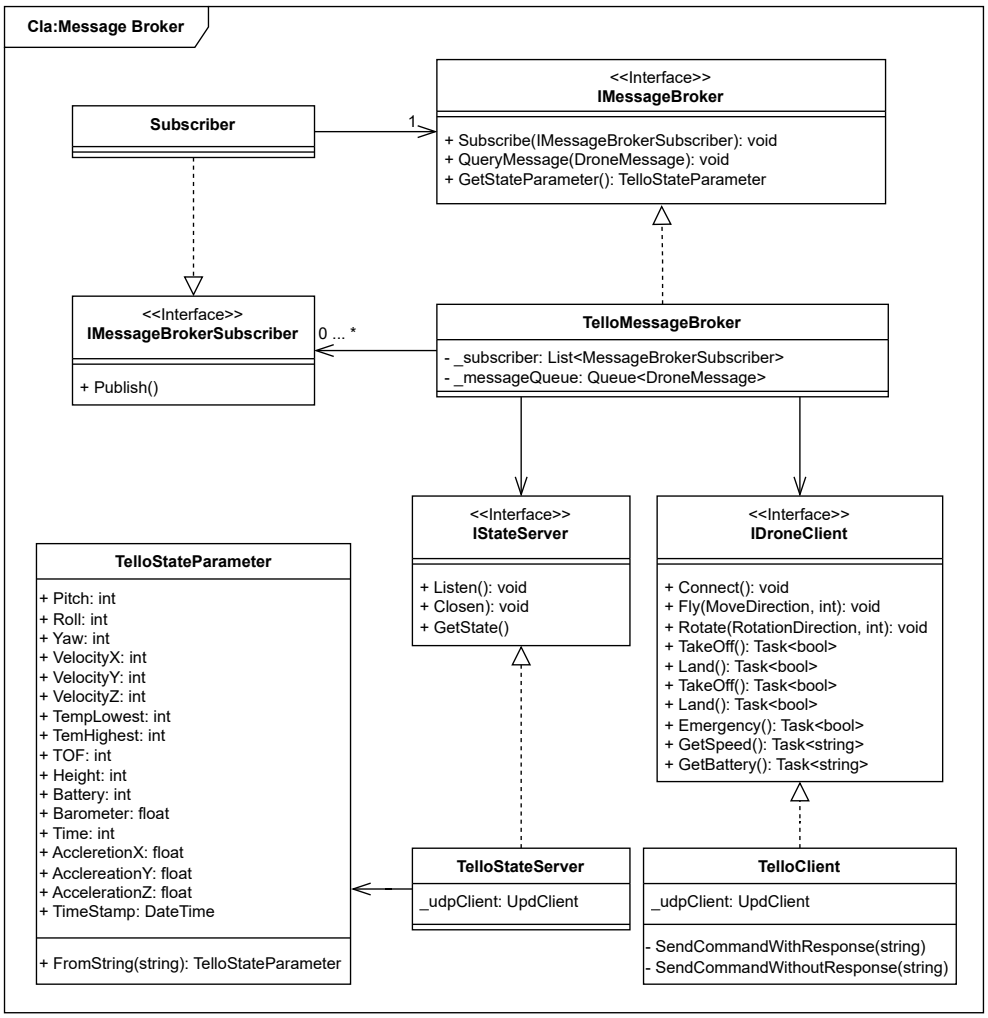


Abbildung 6.1: Klassendiagramm des Message Brokers

6.1.1 Nachrichtenverteilung

Publish-Subscribe

Das Publish-Subscribe ist ein themenbasiertes Modell für den Nachrichtenaustausch, bei dem weder der Sender (Publisher) noch der Empfänger(Subscriber) einander kennen. Beiden verbindet ein Topic, das den Inhalt einer Nachricht beschreibt. Ein Vermittler (Broker) übernimmt die Aufgabe der Verteilung der Nachrichten. Subscriber registrieren sich beim Broker und teilen ihre Topics mit, für die sie Nachrichten empfangen möchten.

Wenn ein Publisher eine Nachricht sendet, werden sie an einen Broker übergeben. Der Broker kategorisiert die Nachricht basierend auf dem Topic und leitet sie dann an die entsprechenden Subscriber weiter.

Das Klassendiagramm in Abbildung 6.2 veranschaulicht das implementierte Konzept.

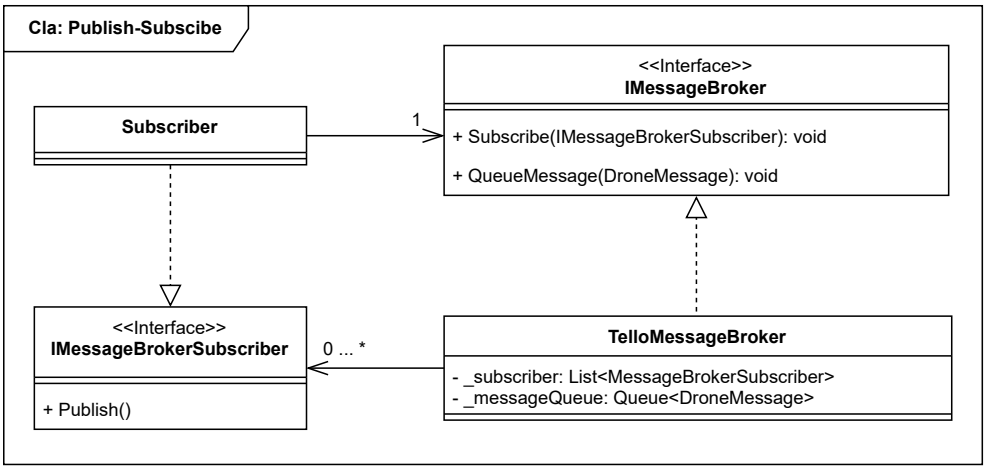


Abbildung 6.2: Klassendiagramm des implementierten Publish-Subscribe-Modells

Das Nachrichtenmuster ist im TelloMessageBroker implementiert, der über das `IMessageBroker`-Interface die Methoden zur Verfügung stellt, damit sich Klassen als Subscriber registrieren können oder Nachrichten als Publisher zu senden. Um sich als Subscriber registrieren zu können, müssen die Klassen das `IMessageBrokerSubscriber`-Interface implementieren. Das Interface definiert die Methode, die der TelloMessageBroker aufruft, um die Nachrichten an die etwaigen Subscriber zu verteilen. Alle Klassen, die eine Instanz auf die TelloMessageBroker halten, können Nachrichten an die Drohne übergeben.

Die Subscriber-Klasse, die oben links in Abbildung 6.2 dargestellt ist, repräsentiert den Agenten und die Tello. Diese Generalisierung wird hier verwendet, um das realisierte Nachrichtenmuster zu veranschaulichen. Im realen System ist die Tello ein geschlossenes System und kann sich nicht beim `TelloMessageBroker` als Subscriber registrieren. Die Filterung der Nachrichten für die Tello, wird daher im `TelloMessageBroker` vorgenommen. Es ist zusätzlich zu beachten, dass sich Subscriber nicht auf bestimmte Topics registrieren können. Diese Entscheidung wurde getroffen, da vorerst alle Nachrichten vom

Agenten empfangen und verarbeitet werden, sodass eine kategorische Zuordnung nicht erforderlich ist.

Nichtsdestotrotz wäre eine vollständige Umsetzung des Publish-Subscribe angebracht gewesen, da weitere Subscriber momentan eine eigene Filterung der Topics vornehmen müssen und dies gegen die Skalierbarkeit des Systems spricht.

6.1.2 Kommunikation mit der Tello

Für die Kommunikation mit der Tello werden die Klassen `TelloStateServer` und TelloClient aus der RyzeSdk verwendet. Das Klassendiagramm 6.3 veranschaulicht, wie die Klassen in die Architektur des Message Brokers integriert sind. Eine wichtige Änderung besteht darin, die wichtigsten Methoden beider Klassen in separate Interfaces, die als `IDroneClient` und `IStateServer` dargestellt sind, auszulagern. Durch diese Änderung werden die Implementationsdetails für die Umsetzung der Nachrichtenbefehle für die Tello hinter abstrakte Methoden verborgen und ermöglicht es diverse Kommunikationsprotokolle einzubinden. Damit ist das System nicht nur auf die Verwendung der Tello beschränkt, sondern kann für diverse Drohnenmodell genutzt werden.

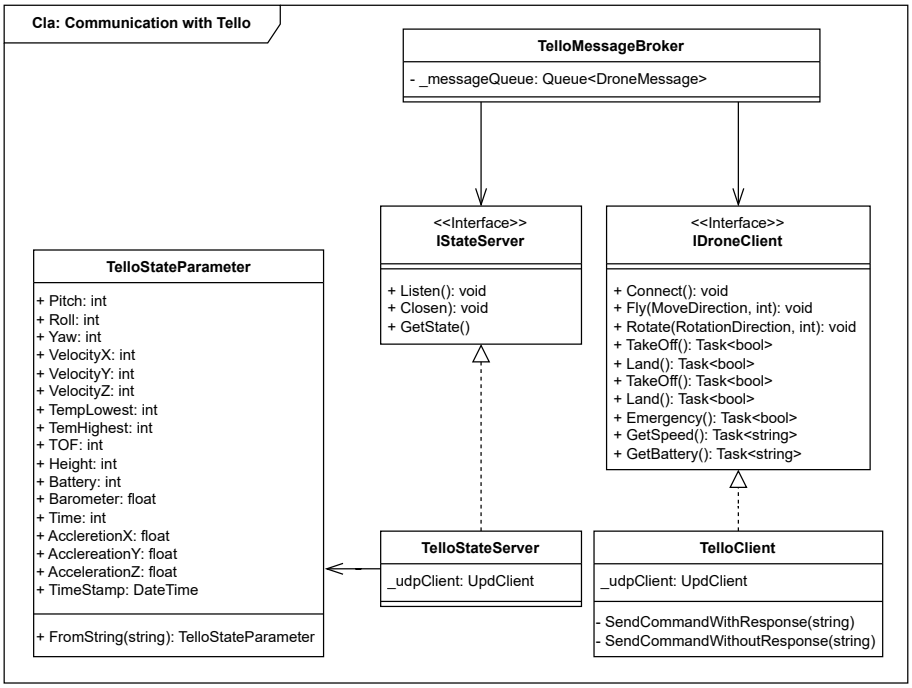


Abbildung 6.3: Klassendiagramm der Schnittstelle zur Tello

Datenakquirierung

Die Fluginformationen werden von der Tello in regelmäßigen Abständen über den Port 8890 veröffentlicht, wie es in 3.1.2 beschrieben ist. Diese Daten werden in der Klasse `TelloStateServer` empfangen und in das Datenmodell `TelloStateParameter` umgewandelt, da die empfangenen Daten unformatiert in einer Zeichenkette übertragen werden. Bei der Umwandlung werden die Daten in Variablen zugeordnet und es wird ein Zeitstempel gesetzt. Der Zeitstempel stellt für einige Funktionen des Systems eine wichtige Information dar, um die Aktualität der Fluginformationen zu überprüfen.

Die umgewandelten Flugdaten werden im `TelloStateServer` temporär gespeichert und können vom `TelloMessageBroker` abgefragt werden. Allerdings werden diese Daten überschrieben, sobald neue Flugdaten empfangen werden. Dies gewährleistet, dass stets aktuelle Daten abgerufen werden. Darüber hinaus wird der Speicherplatz des Rechners effizient genutzt und ein möglicher Speicherüberlauf verhindert.

Befehlsübermittlung an die Tello

Wie in der Einleitung dieses Abschnitts erwähnt, verwendet der `TelloMessageBroker` eine Warteschlange, in der alle eingehenden Nachrichten chronologische zu ihrem Eingangszeitpunkt zugeordnet werden. Der `TelloMessageBroker` holt die nächste Nachricht aus der Warteschlange, sobald die vorherige bearbeitet wurde. Bei der Verarbeitung einer Nachricht überprüft der TelloMessageBroker zuerst das Topic und entscheidet, ob die Nachricht einen Steuerbefehl für die Tello enthält. Wenn dies der Fall ist, wird der spezifische Befehl aus der Message gelesen.

Die Befehle (`DroneAction`) bilden alle grundlegenden Befehle für der Tello und der Operationen im System ab, wie es in Abbildung 6.4 dargestellt ist. Diese Abstraktion ermöglicht eine einheitliche Darstellung der möglichen Aktionen im System. Demzufolge kann der `TelloMessageBroker` die Befehle in einem Switch-Case auf die entspreche Methoden der `IDroneClient` zuweisen und aufrufen.

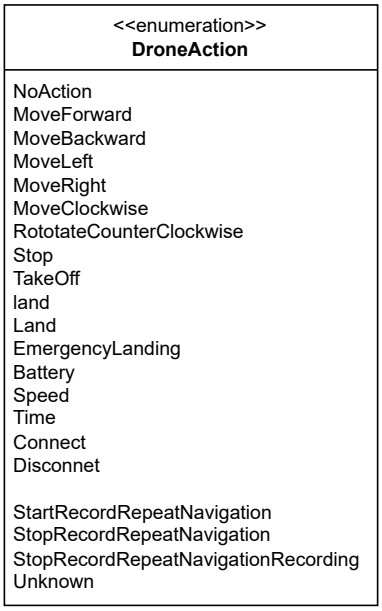


Abbildung 6.4: Klassendiagramm der DroneAction

Der `TelloMessageBroker` sendet jede Nachricht mit einem Zeitabstand von 100 ms, um sicherzustellen, dass keine Nachrichtenengpässe in der Tello entstehen.

Nachrichten, von denen eine Antwort erwartet werden, werden dreimal wiederholt verschickt, um sicherzustellen, dass die Nachricht von der Tello empfangen wird. Dabei wird darauf geachtet, dass nur Befehle wiederholt versendet werden, die keine negativen Auswirkungen auf die Tello haben. Falls zwischen den Wiederholungen eine Antwort empfangen wird, führt der TelloMessageBroker seine Aufgaben fort. Falls nach der dritten Wiederholung keine Antwort erfolgt, wirft der `TelloClient` eine Fehlermeldung zurück, den der TelloMessageBroker als Verbindungsverlust zur Tello interpretiert.

6.2 Digitaler Zwilling

In diesem Abschnitt soll die konkrete Umsetzung den digitalen Zwilling im MARS-Framework vorgestellt werden. Es wird beschrieben, wie das Modell der Tello als Agent umgesetzt wird.

In der Konzeption wurde eine Architektur für den DZ im MARS erarbeitet. In diesem Entwurf wird der DZ in folgende drei Komponenten aufgeteilt: Agent, Layer und Service Center. Sie legt die Vorlage der Implementierung fest und wird im Folgenden aufgegriffen. Das Diagramm zeigt die Klassen des digitalen Zwillings. Die farbliche Markierung der Klassen gibt an, welchen Komponenten die Klasse angehört.

- Blaue Klassen: Die Klassen gehören zur Agenten-Komponente. Sie stellt den Kern des Agenten für das MAS dar.
- Rote Klassen: Die enthaltenen Klassen gehören zur Beschreibung der virtuellen Umgebung.
- Grüne Klassen: Diese Klassen stellen weitere Funktionalitäten und Modelle bereit, die der Agent in seiner Ausführung benötigt.

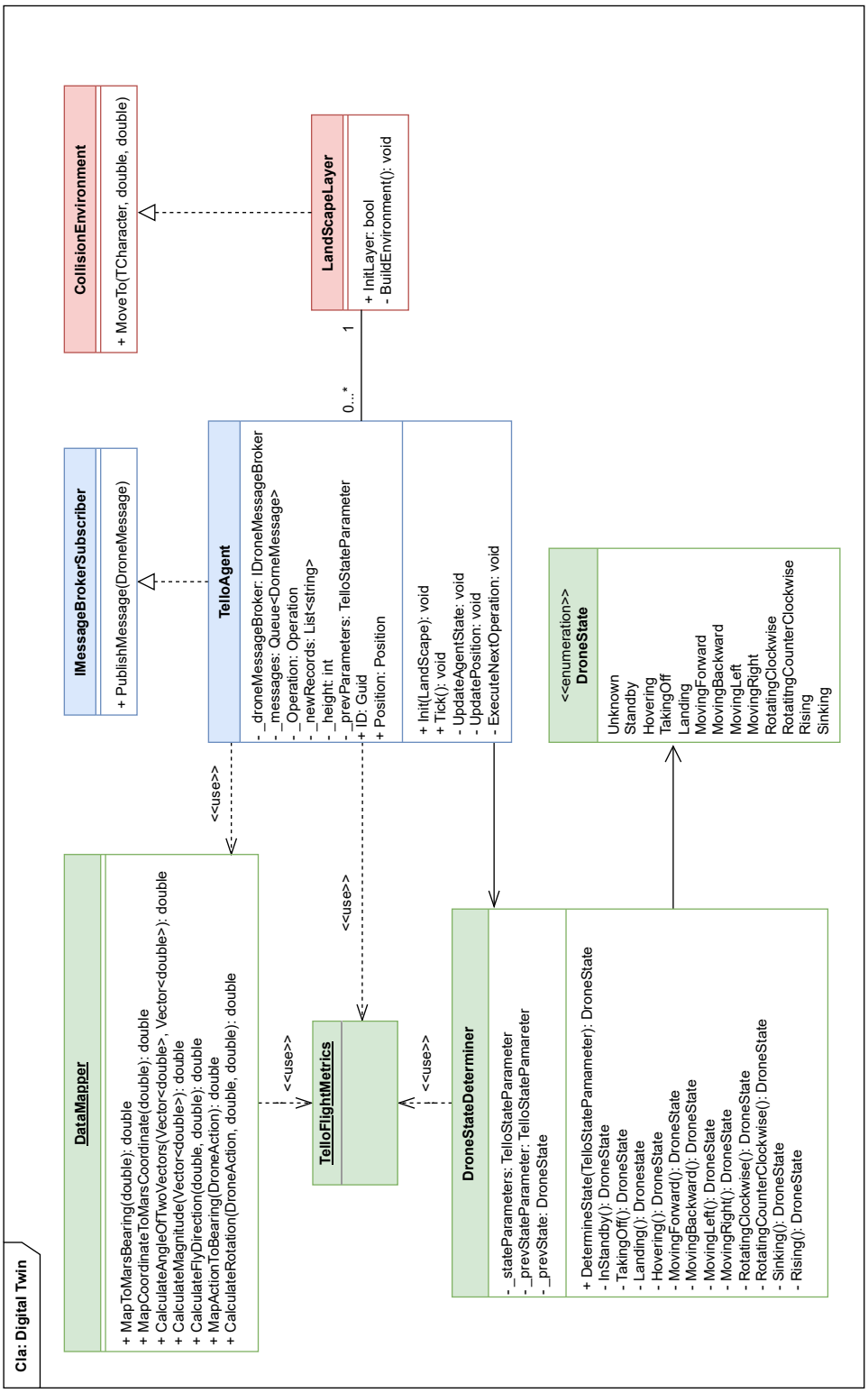


Abbildung 6.5: Klassendiagramm des digitalen Zwillings

6.2.1 Umgebung

Die Umgebung wird von der Klasse `LandScapeLayer` dargestellt, die das `CollisionEnvironment` von MARS implementiert. Das `LandScapeLayer` bildet die Umgebung in einem zwei dimensionalen Raum ab. Die Tello bewegt sich zwar in einem drei dimensionalen Raum, allerdings soll die Position lediglich auf der zwei dimensionalen Ebene bestimmt werden, um die Komplexität der Positionsbestimmung zu vereinfachen. Außerdem kann die Flughöhe der Tello über seine Fluginformationen direkt ausgelesen werden. Durch die Übergabe einer GeoJSON-Datei können Objekte durch Polygone, Linien und Punkte hinzugefügt werden, die zur Darstellung der Raumumgebung genutzt werden können.

Das `CollisionEnvironment` bietet eine `MoveTo()`-Methoden, die mit der Angabe einer Distanz und einer Richtung die neue Position des Agenten bestimmt. Diese Funktion ist ein maßgebliches Kriterium für die Auswahl dieses `Environment`-Types, da die Argumente für diese Methode durch die gegebenen Fluginformationen berechnet werden können.

6.2.2 Service Center

Zugehörig zum Service Center wurden folgende Hilfsklassen implementiert:

1. **TelloFlightMetrics**: Die statische Klasse enthält alle modellspezifischen Informationen zur Tello, die die Drohne in Abmessungen, Gewicht, Flugdauer beschreiben. Diese Informationen ermöglichen eine realistische Modellierung der Tello im virtuellen Raum. Es ist jedoch anzumerken, dass in dieser Arbeit keine Simulation entwickelt wird, weshalb sie im System nicht verwendet werden. Darüber hinaus sind in dieser Klasse Grenzwerte für die Sensordaten definiert, die in den Fluginformationen angegeben sind. Die Grenzwerte entstammen einer analogen Grenzwertanalyse.
2. **DroneStateDeterminer**: Der `DroneStateDeterminer` ist eine Klasse, die als Singleton implementiert ist. Die Klasse ist zuständig den Flugzustand aus den `TelloStateParameter` zu bestimmen. Dazu besitzt sie mehrere private Methoden, in denen einzelne Flugzustände beschrieben sind. Die Flugzustände setzen sich aus der Kombination von Gültigkeitsbereichen zusammen, die aus den Grenzwerten in

der `TelloFlightMetrics` bestimmt werden. Eine Auflistung der identifizierten Flugzustände ist in der Tabelle A.3 gegeben.

3. **DroneState**: Die `DroneState` ist eine Enum-Klasse. Aller Flugzustände sind als Enum in dieser Klasse beschrieben.
4. **DataMapper**: Der DataMapper ist eine statische Helferklasse. Sie beinhaltet Methoden, die der Agent benutzt, um aus den `TelloStateParameter` seine Position zu berechnen oder die Daten in ein MARS abzubilden.

6.2.3 Agent

Der `TelloAgent`, wie im Unterabschnitt 5.4.2 beschrieben, präsentiert die Tello in der virtuellen Umgebung. Seine Aufgabe besteht darin, zusammen mit den Funktionen des `Service Center` den momentanen Zustand der Tello, sowie seine Trajektorie abzubilden. Zudem kann der `TelloAgent` Operationen ausführen.

Alle Schritte, die der `TelloAgent` zur Bewältigung seiner Aufgaben ausführt, sind in seiner Tick-Methode angegeben. Die Schritte sind in Abbildung 6.6 als Aktivitätsdiagramm dargestellt. Das Diagramm zeigt den gesamten Ablauf der Tick-Methode. Um ein besseres Verständnis über die Realisierung der Aufgaben zu bekommen, folgt in den nächsten Abschnitten eine nähere Beschreibung.

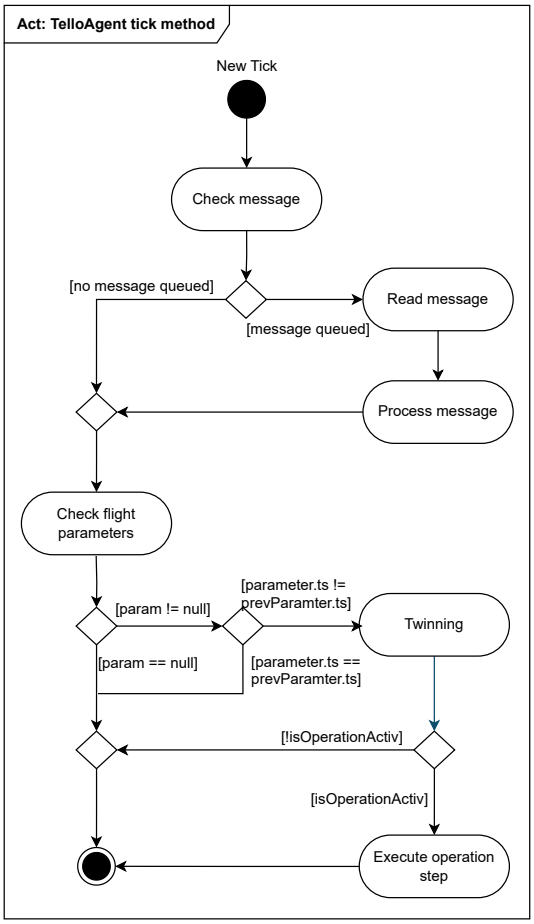


Abbildung 6.6: Aktivitätsdiagramm der Tick-Methode des TelloAgent

Überprüfung der Nachrichtenwarteschlange

Der Agent überprüft zuerst, ob ihm neue Nachrichten zugeschickt wurden ist. Über die Queue erhält Agent den Befehl zur Ausführung seiner Operation oder ein Steuerbefehl, welches er aufzeichnen soll. Bevor eine Nachricht verarbeitet wird, überprüft der Agent den Sender dieser Nachricht. Falls es sich um einen Agenten handelt, wird die Nachricht verworfen. Andernfalls wird die Nachricht verarbeitet.

Twinning

Die Schritte zur Abbildung des Drohnenzustands und der Trajektorie werden in der Abbildung 6.6 unter der Twinning-Aktion zusammengefasst. Wie in Unterabschnitt 6.2.1 beschrieben, müssen im `CollisionEnvironment` die Distanz und die Richtung übergeben werden, um den Agenten auf seine neue Position zu setzen. Das Aktivitätsdiagramm in Abbildung 6.7 zeigt, wie diese Parameter aus den Fluginformationen berechnet werden, sofern neue Fluginformationen zur Verfügung stehen. Dies überprüft der `TelloAgent` über den Zeitstempel, der bei der Übergabe der Daten an den `TelloMessageBroker` gesetzt wird.

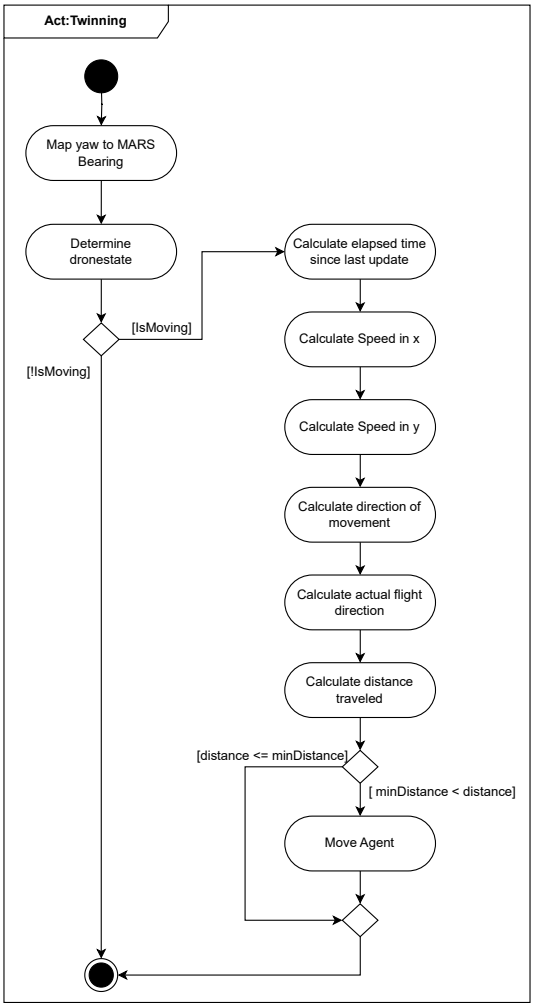


Abbildung 6.7: Aktivitätsdiagramm des Ablaufs des Twinnings

Die theoretische Einführung in 2.3.2 beschreibt, dass ein Agent in einem `CollisionEnvironment` eine Bearing enthält, die die Ausrichtung des Agenten innerhalb der Umgebung darstellt. Im digitalen Zwilling wird die Bearing (α) aus dem Yaw-Wert (ψ) der Fluginformationen abgebildet.

Anschließend wird das Flugverhalten bestimmt, welches vom `DroneStateDeterminer` festgelegt wird. Im Falle einer Bewegung führt der `TelloAgent` eine Berechnung seiner neuen Position durch.

Zuerst wird die Zeitdifferenz t_{diff} aus dem Messzeitpunkt der aktuellen Fluginformationen t_{curr} und der letzten Fluginformationen t_{prev} berechnet.

$$t_{diff} = t_{curr} - t_{prev} \tag{6.1}$$

Im Anschluss werden aus den Fluginformationen die Werte a_x, a_y, vel_x, vel_y verwendet, um die Momentangeschwindigkeiten v_x und v_y zu berechnen.

$$v_x = acc_x \cdot t_{diff} + vel_x \tag{6.2}$$
$$v_y = acc_y \cdot t_{diff} + vel_y \tag{6.3}$$

Aus den Momentangeschwindigkeiten werden Vektoren gebildet, aus denen der Winkel zwischen den beiden Vektoren berechnet wird. Dieser gibt die Flugrichtung dir_{frame} der Tello relativ zu ihrem eigenen Körper an. Der Winkel berechnet sich aus dem Kreuzprodukt.

Um jedoch die tatsächliche Flugrichtung dir_{frame} der Tello zur Umgebung zu bestimmen, muss dir_{frame} mit α des `TelloAgent` verrechnet werden.

$$dir_{world} = dir_{frame} + \alpha \mod 360 \tag{6.4}$$

Schließlich wird noch die Distanz d berechnet, die die Tello im realen Raum zurückgelegt hat.

$$d = |v_x| + |v_y| \tag{6.5}$$

Vor der Übergabe an `MoveTo()` muss d skaliert werden, da die Größe des `CollisionEnvironment` durch die gegebenen Längen- und Breitengrad begrenzt ist. Die skalierte Distanz $\hat{d}$ ergibt sich aus:

$$\hat{d} = \frac{d}{10} \tag{6.6}$$

Im letzten Schritt wird die vorher berechnete Distanz überprüft. Die Distanz wird gegenüber einer Mindestdistanz geprüft, um unnötige kleine Bewegungen zu vermeiden. Ist die Distanz größer, wird diese an `MoveTo()` übergeben und die neue Position wird aufgezeichnet.

Initialisierung des TelloAgent

MARS erlaubt es vor Beginn der Simulation zu Konfigurationsparameter dem Agent übergeben. Diese Informationen werden in einer CSV-Datei geschrieben und enthalten Attribute, die mit den Instanzvariablen des Agenten übereinstimmen. Der `TelloAgent` wird durch die TelloAgent.csv Datei mit Informationen zur Ausgangsposition, Ausrichtung und die Ausgangsfluggeschwindigkeit konfiguriert. Die Ausgangsposition des Agenten sollte basierend auf die reale Position der Drohne in seiner Umgebung gewählt werden. Beispielsweise wäre es nicht sinnvoll den Agenten in die Mitte virtuellen Umgebung zu setzen, wenn die Drohne in der Ecke des Raumes liegt.

6.3 Fernsteuerung

Für die Realisierung der Fernsteuerung werden Methoden aus der `Consoleworker`-Klasse in die Klassen `KeyboardControl` und `FlightDeck` ausgelagert. Damit wird erreicht, dass die Methoden funktional getrennt sind. Die Änderungen sind im Klassendiagramm in Abbildung 6.8 dargestellt.

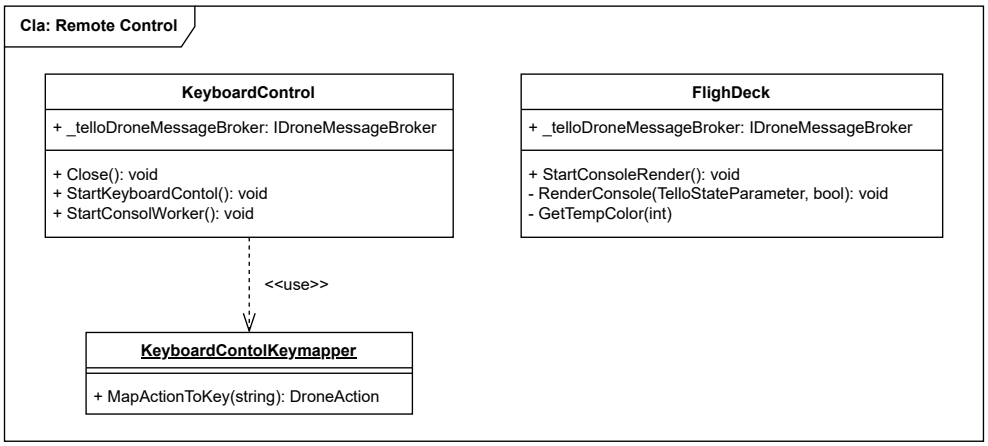


Abbildung 6.8: Klassendiagramm der Fernsteuerung

Die Klasse `KeyboardControl` implementiert die Benutzerschnittstelle und läuft in einem eigenen Thread. Das primäre Ziel dieser Klasse besteht darin, Tastatureingaben einzulesen und sie an den `KeyboardControlKeymapper` weiterzuleiten. Innerhalb dieser Klasse werden die Zeichen auf `DroneActions` abgebildet und an die `KeyboardControl` zurückgegeben. Die `KeyboardContol` nimmt über die Tasteneingaben Zeichen entgegen, die an den `KeyboardControlKeymapper` übergeben werden. Dieser wandelt die Zeichen in die zugehörige `DroneAction` um und gibt diese zurück. Eine vollständige Auflistung der Zuweisungen ist in Tabelle A.2 gegeben. Die `KeyboardControl` wandelt aus der zurückgegebene `DroneAction` in eine `DroneMessage` um und übergibt sie dem `TelloMessageBroker`. Durch die Auslagerung der Tastenzuweisung in eine eigenständige Klasse können unterschiedliche Tastenbelegungen je nach Anforderungen ausgetauscht werden.

Das `Cockpit` erstellt die Anzeige und läuft ebenfalls auf einem separaten Thread. Die Methoden dieser Klasse stammen direkt aus dem `ConsoleWorker`. Eine Änderung der Methoden ist nicht erforderlich, da sie lediglich zur formatierten Darstellung der Flugin-

formationen dienen. Darüber hinaus werden Temperaturen farblich dargestellt, um den Benutzer vor einer Überhitzung der Tello zu warnen. Die Abbildung 6.9 zeigt eine Momentaufnahme der dargestellten Fluginformationen.

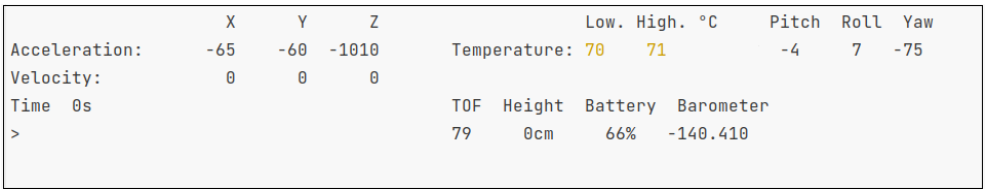


Abbildung 6.9: Momentaufnahme des Cockpits bei einer aktiven Verbindung mit der Tello-Drohne

6.4 Record-Repeat Navigation

In diesem Abschnitt wird die technische Implementation und die Funktionsweise der RRN Operation beschrieben. Das Klassendiagramm in Abbildung 6.10 zeigt die Klassen, die für die Durchführung der RRN zuständig sind.

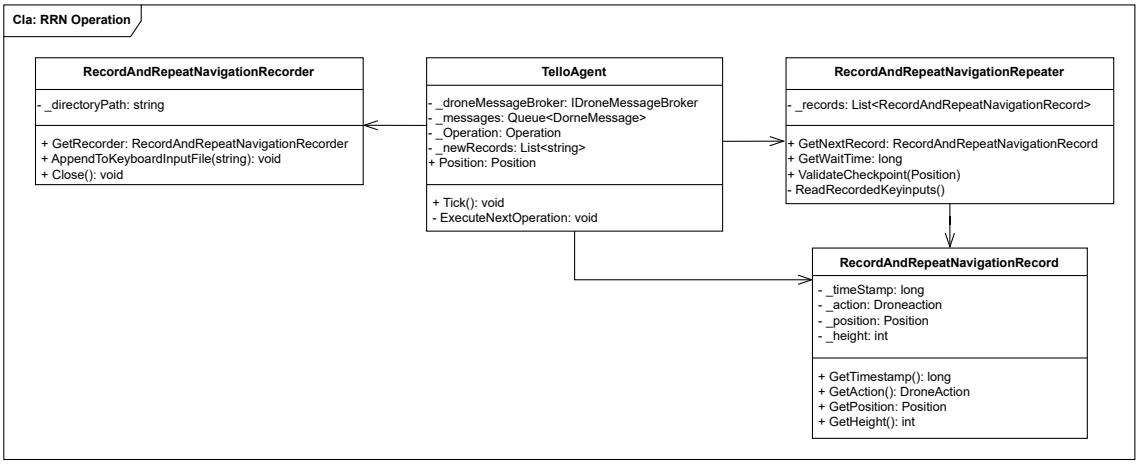


Abbildung 6.10: Klassendiagramm für RRN

Die Klassen haben folgende Aufgaben:

- RecordAndRepeatNavigationRecorder: Die Klasse `RecordAndRepeatNavigationRecorder` enthält einen Dateipfad, in der alle Aufzeichnungen geschrieben

werden. Diese Klasse wird aufgerufen, wenn der `TelloAgent` den Befehl erhält, seine Aufzeichnungen in einer Datei abzuspeichern. Der `TelloAgent` ruft iterativ die `AppendToKeyboardInputFile(string)`-Methode auf und übergibt die einzelnen Aufzeichnungen. Der `RecordAndRepeatNavigationRecorder` erstellt eine neue CSV-Datei und schreibt jeden übergebenen Befehl in eine neue Zeile am Ende der Datei.

- RecordAndRepeatNavigationRecord: `RecordAndRepeatNavigationRecord` ist ein Datenmodell, um die Attribute in der CSV-Datei einem Datentypen zuzuordnen.
- RecordAndRepeatNavigationRepeater: Diese Klasse erhält bei dem Programmstart einen Pfad zur Aufzeichnung, die nachgestellt werden soll. Sie liest aus der Datei und wandelt die Attribute in `RecordAndRepeatNavigationRecord` um. Während der Ausführung der Operation übergibt diese Klasse den `TelloAgent` einzeln die Aufzeichnungen aus. Innerhalb der Klasse kann auch die Wartezeit für den nächsten Ausführungsbefehl und die Position des Agenten validiert werden.

Die Sequenzdiagramme in den Abbildungen 6.11 und 6.12 zeigen die realisierte Operation, mit den in Abschnitt 6.4 eingeführten Klassen und den Ablauf, der in Abschnitt 5.6 beschrieben wurde.

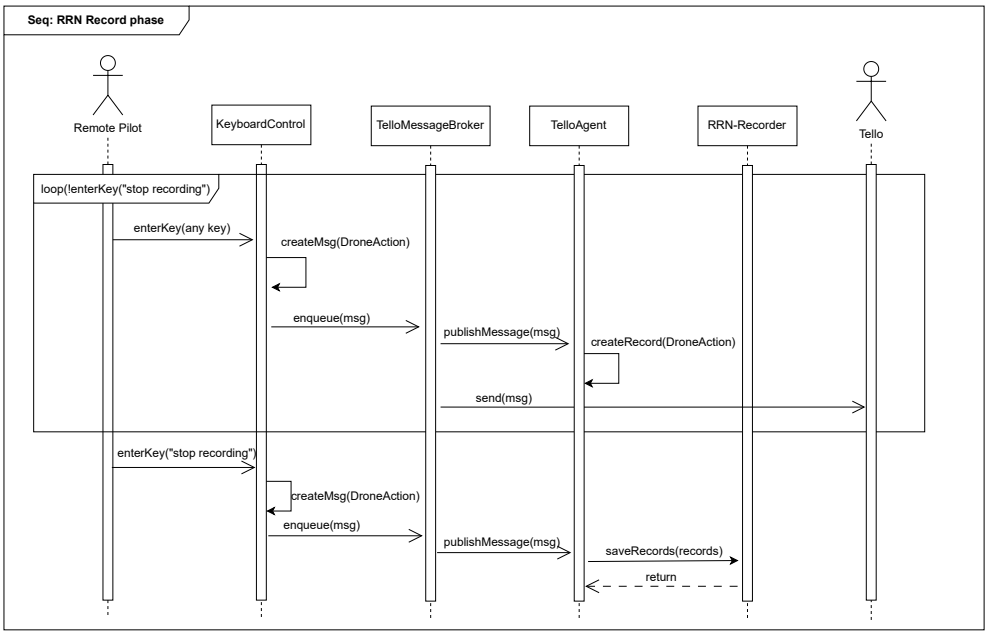


Abbildung 6.11: Sequenzdiagramm der Record-Phase der RNN

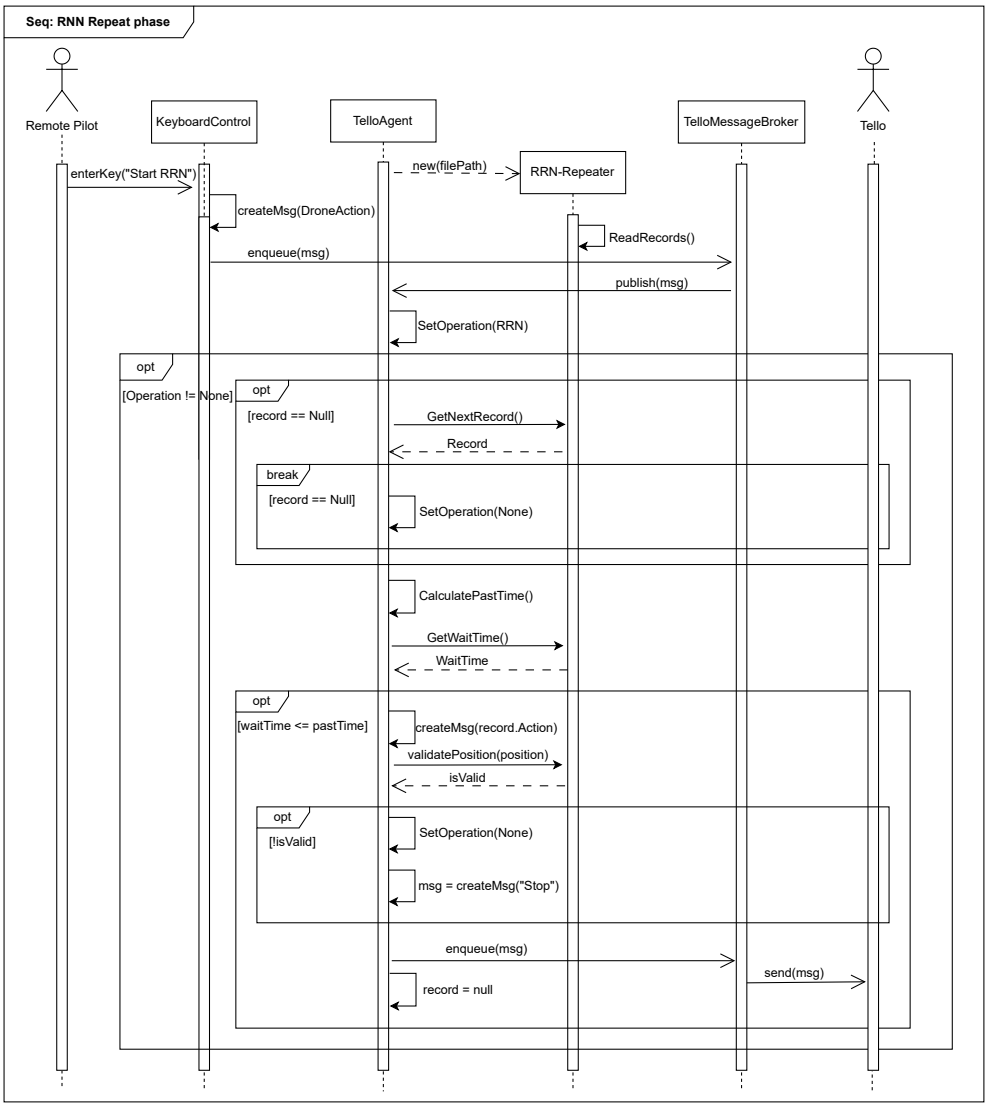


Abbildung 6.12: Sequenzdiagramm der Repeat-Phase der RRN

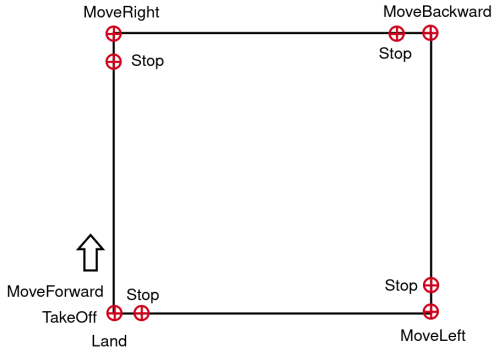
7 Versuch und Evaluation

Einer der wichtigsten Anforderungen an den DZ ist seine Anpassungsfähigkeit an seine sich verändernden Situation. Um diese Eigenschaften zu bewerten, ist es erforderlich, die Interaktionsfähigkeit zwischen dem virtuellen und dem realen Raum zu analysieren. Zu diesem Zweck werden zwei Versuche konzipiert. Der erste Versuch soll die Eigenschaften des Systems, in Hinsicht auf seine Fähigkeit den Zustand der physischen Drohne abzubilden, untersuchen. Der zweite Versuch bewertet die RRN und wird in zwei Teilen aufgeteilt. Im ersten Teil wird eine statistische Auswertung vorgenommen, die die Genauigkeit der Operation untersucht. Im Anschluss folgt eine Validierung der Ergebnisse, indem Validationsradien selektiv gewählt werden.

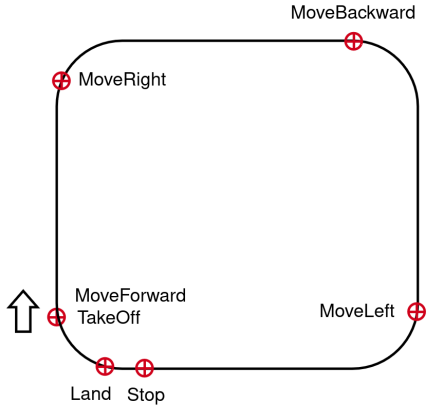
7.1 Untersuchung der Abbildungseigenschaften des Agenten

Der erste Versuch befasst sich mit der Untersuchung der Trajektorieabbildung des Agenten basierend auf den Fluginformationen, die die Tello zur Verfügung stellt. Anhand dieses Versuches soll untersucht werden, ob der Agent in der Lage ist, die ausgeführten Flugbahn korrekt abzubilden. Hierfür werden drei verschiedene Trajektorien vorgestellt, die verschiedene Aspekte der Positionsabbildung veranschaulichen.

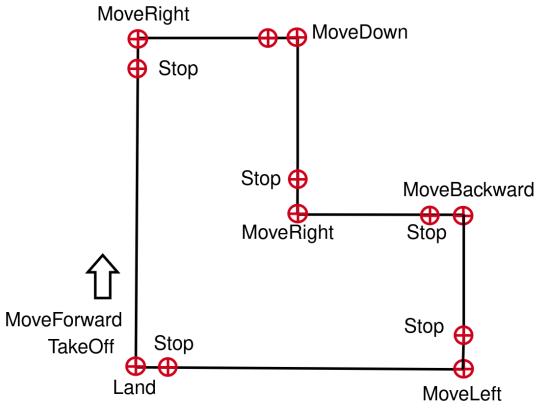
Die Trajektorien, die in diesem Versuch geflogen werden, sind in Abbildung 7.1 veranschaulicht. Die Rechtecktrajektorie demonstriert ein Flugmanöver, anhand grundlegende Aussagen über die Qualität der Trajektorieabbildung getroffen werden. Die Rechtecktrajektorie ohne Zwischenstopps soll zeigen, wie die Abbildung durch abrupte Bewegungen dargestellt wird. Durch die L-Trajektorie soll gezeigt, ob Abweichungen durch in der Abbildung unterschiedlichen Streckenlängen auftreten.



(a) Rechtecktrajektorie mit Zwischenstopps



(b) Rechtecktrajektorie ohne Zwischenstopps



(c) L-Trajektorie

Abbildung 7.1: Theoretische Trajektorien für den ersten Versuch

Durchführung

Alle Trajektorien werden manuell über die Fernsteuerung einmal ausgeführt. Der Versuch findet innerhalb eines geschlossenen Raumes, auf einer Testfläche von circa 2 m x 3 m statt. Auf der Fläche befinden sich keine Hindernisse, die die physische Drohne behindern können. Für die Ausführung wird die Tello mit einer maximalen Fluggeschwindigkeit von 0,5 $\frac{m}{s}$ konfiguriert. Die Aufzeichnung der Flugbahnkoordinaten beginnt mit dem Befehl `Takeoff` und endet mit dem `StopRrNavigationRecording`-Befehl. Während des Fluges werden die Koordinaten des Agenten, wie in **??** beschrieben, aufgezeichnet, sobald die Fluggeschwindigkeit einen bestimmten Grenzwert überschreitet oder wenn ein neuer

Befehl gesendet wird. Wenn die Aufzeichnung endet, werden alle gesammelten Daten in einer .csv-Datei zusammengefasst und abgespeichert.

Ergebnisse

Die erste Trajektion in Abbildung 7.2 zeigt grundsätzlich, dass die Positionen des Agenten aufgezeichnet und dargestellt werden können. Die x-Achse und die y-Achse repräsentieren die Flugrichtung des UAVs, die im Abschnitt 2.1.2 eingeführt wurden. Die Abbildung zeigt eine rechteckförmige Trajektorie, die ihren Startpunkt unten links hat. Nach dem Startbefehl *TakeOff*, bewegt sich der Agent in nördlicher Richtung. Nach 2 m hält die Tello an und fliegt einen Meter in die östliche Richtung, dann in die südliche, bis sie zum Schluss in die westliche Richtung fliegt und in der Nähe des Startpunktes wieder landet. Die Kreuze im Diagramm stellen die Koordinatenpunkte dar, die ein Agent in Laufe der Zeit zurückgelegt hat. Es ist zu sehen, dass die Koordinatenpunkte während der Bremsung enger beieinander liegen und während des Fluges sich weiter auseinander dehnen. Nordwestliche der Trajektion zeigt sie ein unerwarteter Knick.

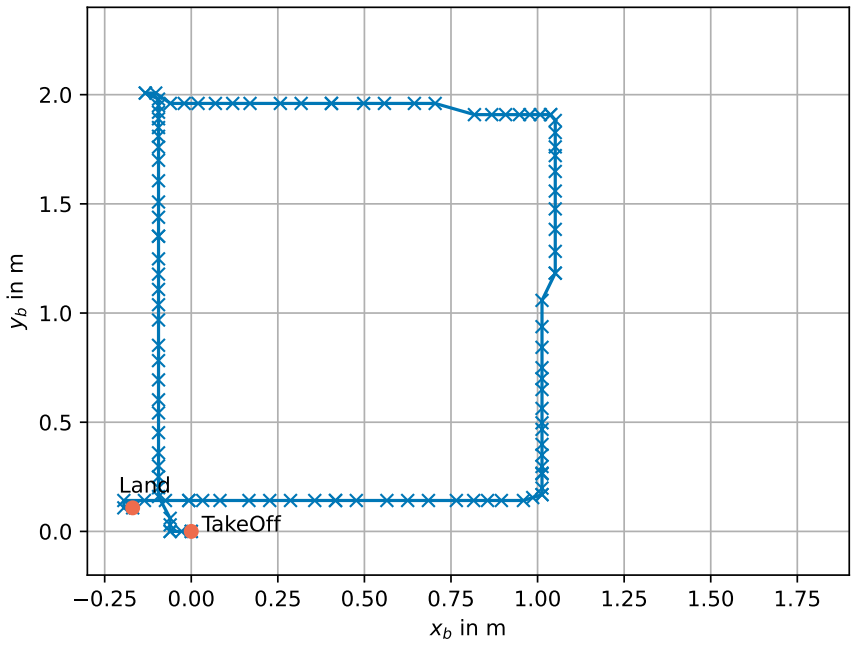


Abbildung 7.2: Geflogene Rechtecktrajektorie mit Zwischenstopps im 2D-Raum

Die Trajektion in der Abbildung 7.3 zeigt eine Rechtecktrajektion ohne Zwischenstopp. Die Abbildung zeigt eine Flugbahn, die der vorherigen Trajektorie sowohl in Richtung als auch Form ähnelt. An den Stellen, an denen eine Richtungsänderung ausgeführt wird, werden die Kurven in einem langen Bogen dargestellt. Es ist jedoch anzumerken, dass die Streckenlänge entlang der x-Achse unverhältnismäßig weit verläuft.

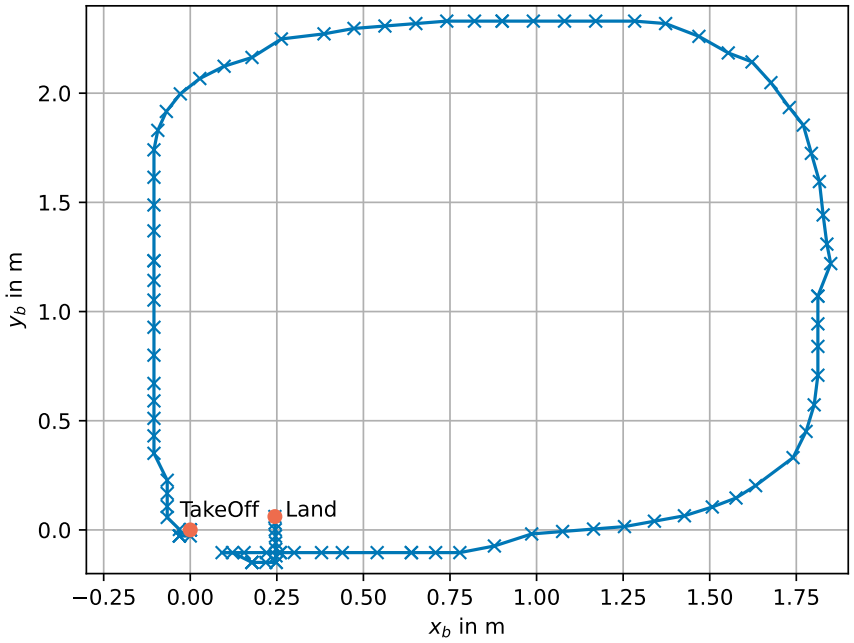


Abbildung 7.3: Geflogene Rechtecktrajektorie ohne Zwischenstopps im 2D-Raum

Die dargestellte Trajektion in Abbildung 7.4 zeigt eine L-förmige Flugbahn, bestehend aus einer Abfolge von kurzen und langen Streckenabschnitten. Es ist an der Abbildung festzustellen, dass die Proportionen von kurzen von langen Strecken richtig dargestellt werden können, weil die Abbildung, dass den Landebefehl auf der gleichen y-Achsen Höhe darstellt, wie der Startbefehl. Es kann außerdem beobachtet werden, dass die Drohne beim ersten Rückwärtsflug leicht übersteuert und sich eigenständig in die gewünschte Flugbahn zurückbegibt.

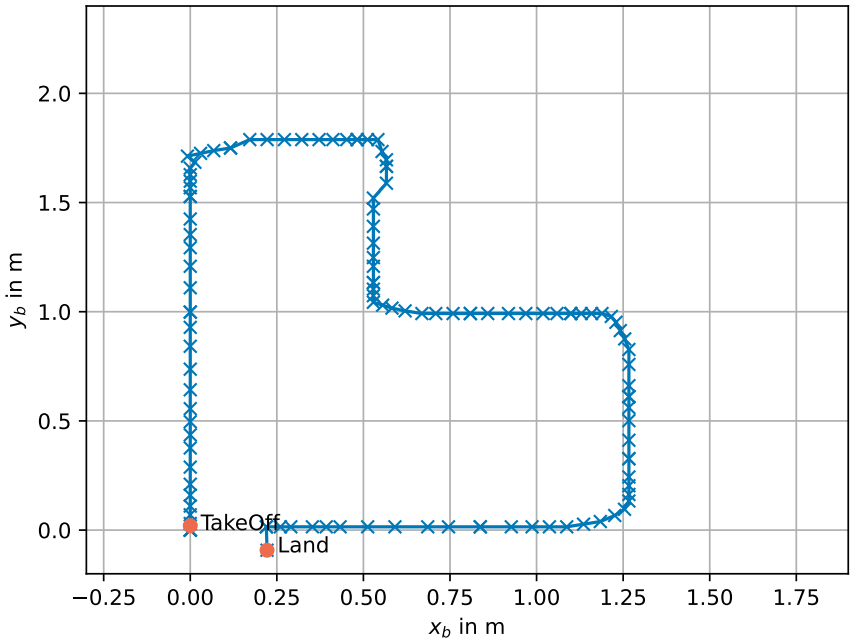


Abbildung 7.4: Geflogene L-Trajektorie im 2D-Raum

7.2 Autonome Navigation einer aufgezeichneten Trajektorie

Im zweiten Versuchs wird eine Analyse der RRN Operation vorgenommen, um zu überprüfen, wie zuverlässig sich diese Operation zur Nachstellung von Trajektorien eignet. Dieser Versuch teilt sich in zwei separate Teilversuche auf.

Im ersten Teilversuch wird die Trajektorie vom Agenten ohne Verwendung eines Validationsradius nachgestellt, um eine statistische Auswertung der nachgestellten Flugbahn zu ermöglichen. Es wird eine Analyse der Genauigkeit der Operation durchgeführt, bei der die mittlere Distanz zwischen den Koordinaten ausgeführten Befehlen und ihren Soll-Koordinaten berechnet wird.

Die mittlere Distanz $\bar{m}_s$ wird als Maß für die Abweichung verwendet und kann mit der folgenden Gleichung berechnet werden:

$$\bar{m}_s = \frac{1}{n} \sum_{i=1}^{n} \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{7.1}$$

Hierbei sind x_i und y_i die x- und y-Koordinaten der Ist-Koordinate, die sich aus der nachgestellten Trajektorie ergeben. x_s und y_s sind die x- und y-Koordinatenpunkte der Soll-Koordinate und sind in der Aufzeichnung der vorgegeben Trajektorie gestellt.

Im zweiten Versuchsteil sollen die Ergebnisse des ersten Teilversuchs überprüft werden, indem mehrere Validationsradien für die maximal erlaubte Abweichung eingesetzt werden. Für die Auswahl der Radien werden die Ergebnisse aus dem ersten Versuch berücksichtigt. Dieser Versuch soll zwei Aspekte überprüfen. Zum einen soll überprüft werden, ob der Agent imstande ist, Überschreitungen des Gültigkeitsbereichs wahrzunehmen und seine Operation entsprechend abzubrechen. Zum anderen soll untersucht werden, ob die ermittelten Durchschnittsabstände der Befehle als Validationsradius verwendet werden können. Dabei wird die Annahme getroffen, dass der Agent tendenziell bei diesen Befehlen die Validation nicht besteht.

7.2.1 Navigation ohne Validation

Um die durchschnittliche Abweichung zu berechnen, soll der Agent die Operation nicht vorzeitig abbrechen. Daher wird die Operation ohne Verwendung eines Validationsradius durchgeführt.

Aufbau
Für diesen Versuch wird die L-Trajektorie, die im ersten Versuch erstellt wurde, als Grundlage für diesen Versuch verwendet. Diese Trajektorie eignet sich aufgrund ihrer unterschiedlichen Bewegungsrichtungen und Flugstreckenlängen. Die Abbildung 7.5 zeigt die L-Trajektion und veranschaulicht die eingegebenen Befehle zur Ausführung dieser Trajektorie.

Dabei ist zu beachten, dass einige Befehle möglicherweise irreführend sein können. Ein Beispiel ist der Fall, in dem der Befehl `MoveRight` gesetzt ist, jedoch die Tello weiterhin nach vorne bewegt wird. Der Grund liegt darin, dass das Setzen der Koordinaten für die

Aufzeichnung latenzfrei ist, wohingegen das Senden des Befehls an die Tello eine gewisse Verzögerung aufweisen kann.

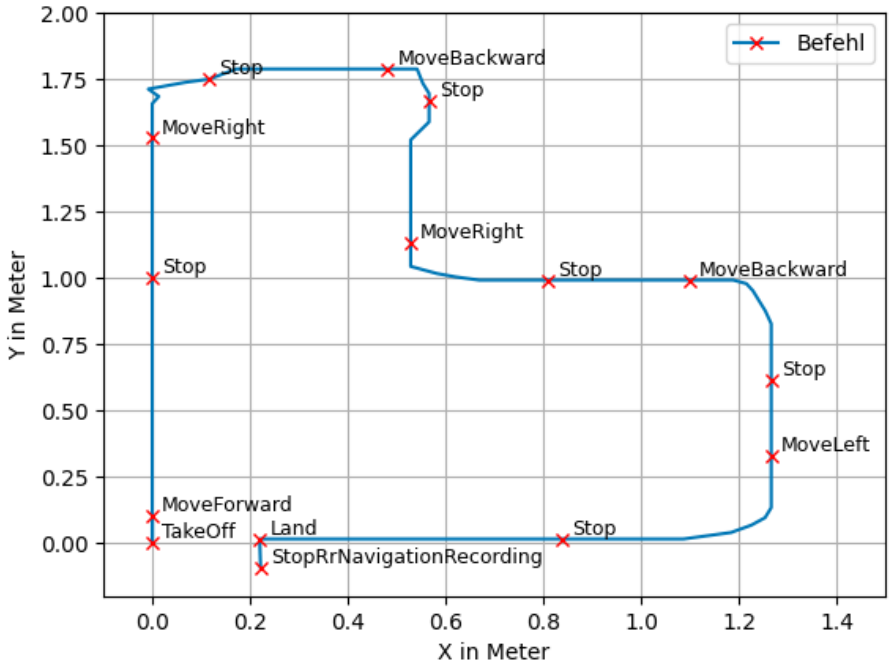


Abbildung 7.5: L-Trajektorie mit den aufgezeichneten Befehlsangaben

Durchführung

Die Operation wird zehnmal wiederholt. Für jede Wiederholung wird die Tello an die gleiche Startposition gesetzt. Es werden für diesen Versuch alle Wiederholungen gezählt, mit Ausnahme derjenigen Wiederholungen, bei denen die Operation durch einer Kollision mit einem Hindernis abbricht.

Ergebnisse

Die Abbildung 7.6 veranschaulicht die zehn Wiederholungen und stellt sie der Vorlage gegenüber. Es ist zu erkennen, dass die wiederholten Trajektorien stark von der erwarteten Trajektorie abweichen. Besonders auffällig sind die Abweichungen zu Beginn der Ausführung, da während des Vorwärtsflugs eine tendenzielle Abweichung nach links festgestellt wird. Darüber hinaus neigen die Wiederholungen dazu, gegen Ende der Vorwärtsbewegung die Soll-Koordinate bei ungefähr (0; 1,8) zu überschreiten. Zusätzlich setzt sich die Abweichung in y-Richtung über die nachfolgenden Befehle fort, wobei festzustellen ist, dass der Agent etwa 0,5 m neben dem erwarteten Zielpunkt landet.

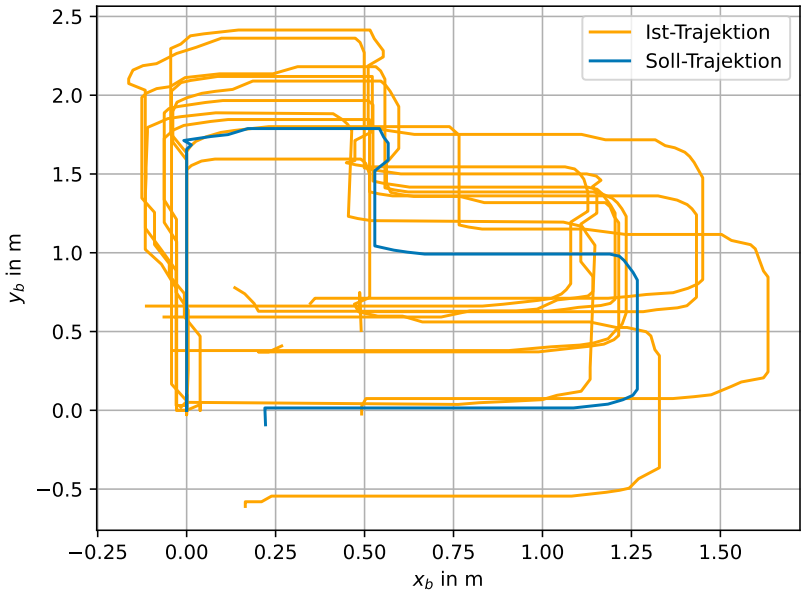


Abbildung 7.6: Wiederholung der Trajektorien ohne Validationsradius

Die Abbildung 7.7 veranschaulicht die Verteilung der Ist-Koordinaten über alle Befehle. Dabei sind die Soll-Koordinaten jedes Befehls sind im Zentrum des Koordinatensystems dargestellt. Diese Grafik setzt sich aus den Abbildungen A.1 bis A.14 zusammen.

Die Grafik deutet darauf hin, dass die Befehle eine signifikante Abweichung in nördlicher Richtung von den Soll-Koordinaten aufweisen. In einigen Fällen ist eine Abweichung von bis zu 0,8 m festzustellen, was jedoch unrealistisch erscheint. Im Gegensatz dazu sind die Abweichungen auf der x-Achse nur halb so groß und liegen unter 0,4 m. Diese Darstellung verdeutlicht, dass die Fehler, die während des Vorwärtsflugs auftreten, sich über die nachfolgenden Befehle hinweg fortsetzen.

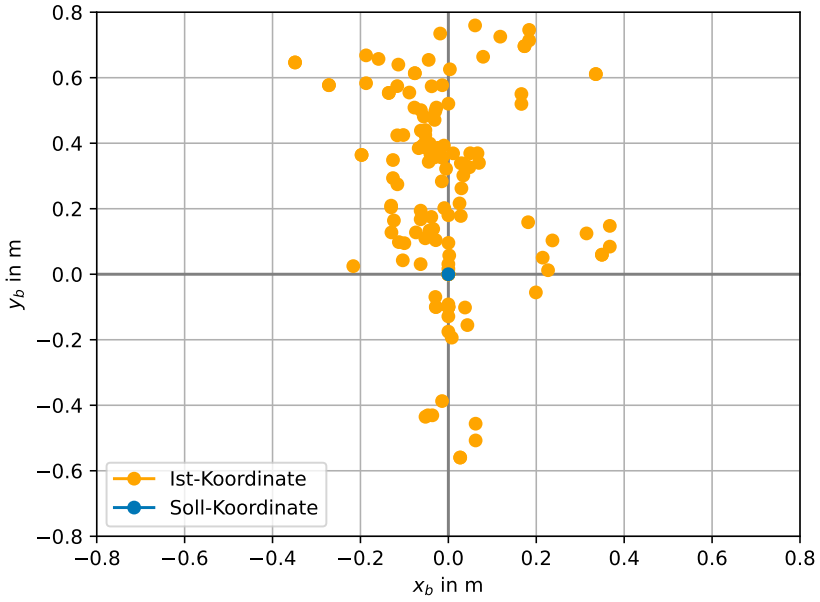


Abbildung 7.7: Darstellung alle Koordinatenpunkte aller Versuche

Die Tabelle 7.1 listet die durchschnittlichen Abweichungen $\bar{m}_s$ der Wiederholungen zu jedem Befehl auf. Die Ergebnisse der Tabelle zeigen, dass die Abstände zu den Soll-Koordinaten nach jedem ausgeführten Befehl in der Regel zunehmen. Eine Ausnahme stellen hierbei die Befehlsnummern 4, 6, 7 und 14 dar, wo die Abstände zu den Soll-Koordinaten zur vorherigen Abweichung kleiner werden. Außerdem ergibt sich aus der Tabelle, dass der Agent im Durchschnitt bis zum letzten Befehl eine Abweichung von etwa 0,52 m von der Soll-Koordinate aufweist.

Tabelle 7.1: Mittlere Abweichungen zwischen den Soll- und Ist-Koordinaten der Befehle

Befehlsnummer	**Befehl**	$\bar{m}_s$ **in m**
1	Takeoff	0
2	MoveForward	0,102
3	Stop	0,2785
4	MoveRight	0,2585
5	Stop	0,3170
6	MoveBackward	0,3026
7	Stop	0,2799
8	MoveRight	0,3797
9	Stop	0,4329
10	MoveBackward	0,4372
11	Stop	0,4925
12	MoveLeft	0,5076
13	Stop	0,5308
14	Land	0,5235

7.2.2 Navigation mit Validation

Im zweiten Teil des Versuchs werden nun die Ergebnisse aus dem ersten Versuchsteil validiert, indem die ermittelten mittleren Abweichungen zur Festlegung von Grenzwerten für der Validierung der RRN verwendet werden.

Aufbau
Im vorherigen Versuch wurde festgestellt, dass die nachgestellten Trajektorien von der Vorlage abweichen. Während die durchschnittliche Abweichung beim 7. Befehl 0,3 m beträgt, wächst sie beim 9. Befehl auf 0,4 m an und erreicht schließlich beim letzten Befehl eine durchschnittliche Abweichung von 0,5 m. Zur Validierung werden die Validationsradien 0,3 m, 0,4 m und 0,5 m eingesetzt. Es wird überprüft, ob die Operation tendenziell bei den entsprechenden Befehlsnummern abgebrochen wird.

Durchführung
Es werden die ersten zehn Ausführungen gezählt, einschließlich unterbrochener Operationen, bei denen die Validierung nicht erfolgreich sind. Allerdings werden wieder die Wiederholungen nicht mitgezählt, bei denen die Operation durch eine Kollision mit einem Hindernis unterbrochen werden.

Ergebnisse
Zuerst werden die Durchläufe des Versuchs unter Verwendung eines Validationsradius von 0,3 m analysiert. Das Balkendiagramm in Abbildung 7.8 zeigt, dass die Hälfte der Wiederholungen die Validierung des dritten Befehls (`Stop`) passieren. Interessanterweise ist außerdem zu sehen, dass ab dem Erreichen des dritten Befehls die Operation die folgenden Schritte mit einer hohen Wahrscheinlichkeit erfolgreich ausführen kann. Das deutet darauf hin, dass die Operationen früher abbrechen als die Durchschnittswerte vermuten lassen.

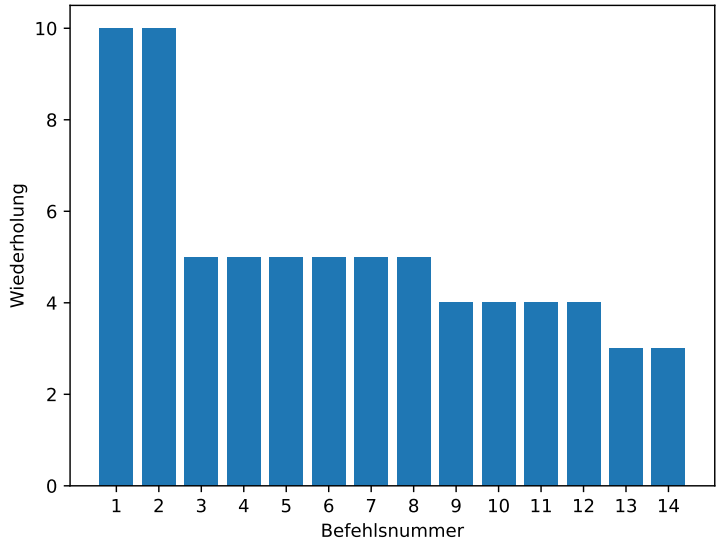


Abbildung 7.8: Ausführung der RRN mit einem Validationsradius von 0,3 m

Die Abbildung 7.9 veranschaulicht die Ausführungen unter Verwendung eines Validationsradius von 0,4 m. Die Ergebnisse legen nahe, dass in einem Drittel der Wiederholungen die gesamte Operation erfolgreich abgeschlossen wird, während zwei Drittel der Operationen vor dem 8. Befehl (`MoveRight`) unterbrochen werden. Diese Ergebnisse deuten erneut darauf hin, dass die Operation früher abgebrochen wird als in den Ergebnissen des ersten Versuchsteils dargestellt.

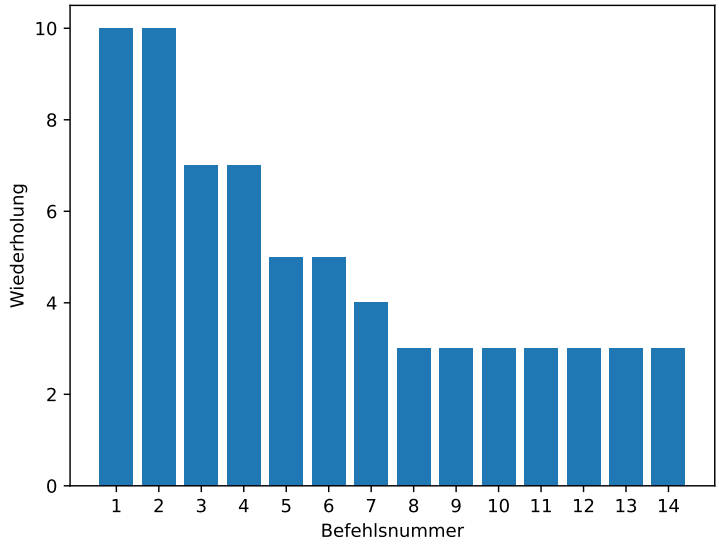


Abbildung 7.9: Ausführung der RRN mit einem Validationsradius von 0,4 m

Zum Schluss werden die Ergebnisse der Durchführungen unter Verwendung eines Validationsradius von 0,5 m untersucht. Das Balkendiagramm in Abbildung 7.10 veranschaulicht die Anzahl der Ausführungen für jeden Befehl. Die Ergebnisse legen nahe, dass von den zehn Wiederholungen nur sechs die Operation erfolgreich abschließen. Die restlichen Ausführungen enden bei dem 7. und dem 10. (`MoveRight`, `Stop` und `MoveLeft`) Befehl. Zusätzlich zeigt das Diagramm, dass nach der Ausführung des 11. Befehls keine weiteren Unterbrechungen mehr stattfinden.

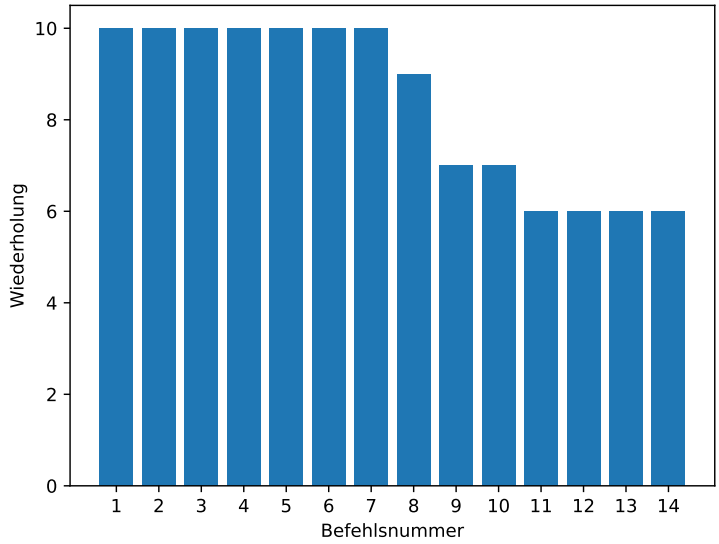


Abbildung 7.10: Ausführung der RRN mit einem Validationsradius von 0,5 m

Insgesamt fällt bei allen drei Untersuchungen auf, dass die Operationen am häufigsten bei den `Stop`-Befehlen unterbrochen werden.

8 Diskussion, Fazit und Ausblick

8.1 Diskussion

8.1.1 Diskussion zu Versuch 1

In Anbetracht der vorliegenden Ergebnisse lässt sich feststellen, dass das Drohnensystem in der Lage ist, Trajektorien der Tello bei verschiedenen Manövern abzubilden. Dies wird durch die ausgewählten Trajektorien deutlich, die mithilfe des DZ aus den aufgezeichneten Informationen dargestellt werden können. Die erfolgreiche Umsetzung der Konzepte zur Erstellung der Umgebung und Berechnung der Positionen bestätigt die gute Auswahl und Umsetzung dieser Konzepte.

Es treten gelegentlich Abbildungsfehler in den Trajektorien auf, die möglicherweise auf ungenaue Flugzustandsbestimmungen zurückzuführen sind. Insbesondere treten diese Fehler auf, wenn die Tello aus einer Flugbewegung heraus anhält. Daraus lässt sich schließen, dass die Schritte zur Abbildung der Position noch optimierungsbedürftig sind, sei es in Bezug auf die Berechnung der Position oder die Berücksichtigung der unterschiedlichen Randbedingungen. Die Ergebnisse zeigen auch, dass die Aufzeichnungen ausreichend Informationen bereitstellen, um den Flugverlauf mithilfe externer Werkzeuge darzustellen. Zudem wurde ein geeignetes Dateiformat ausgewählt, um diese Daten abzuspeichern.

Insgesamt lässt sich feststellen, dass Multiagentensysteme mit dem MARS-Framework effektiv und mit geringem Aufwand entwickelt werden können. Insbesondere für die Modellierung der räumlichen Umgebung stellt das Framework durch eine Vielzahl von verfügbaren `Layers` und `Environments` einen umfangreichen Werkzeugkasten. Allerdings stellte die Entwicklung eines `Agent` eine Herausforderung dar. Die Entwicklung eines Agenten innerhalb einer geschlossenen Simulation ist vergleichsweise einfach. Die Entwicklung eines Agenten, der mit externen Datenquellen interagiert, erfordert jedoch eine sorgfältige Planung Die vorliegende Arbeit dient als Beispiel für diese Herausforderung

und verdeutlicht den Bedarf an weiteren Konzepten zur Vereinfachung der Agentenmodellierung.

8.1.2 Diskussion zu Versuch 2

Der Gesamtversuch des zweiten Teils belegt die Funktionsfähigkeit der implementierten Operation, die nicht nur die Tello abbilden, sondern sie auch autonom steuern kann. Unter Berücksichtigung der Ergebnisse des ersten Teilversuchs ist festzustellen, dass der DZ in der Lage ist, die aufgezeichnete Trajektorie korrekt einzulesen und die Befehle korrekt nachzustellen. Allerdings zeigt sich im Verlauf der Zeit eine zunehmende Abweichung der Flugbahn, was sich an der Tabelle 7.1 veranschaulichen lässt. Diese Abweichung ist auf die Inkonsistenz der Fluggeschwindigkeit der Tello zurückzuführen, da sie mal schneller und mal langsamer fliegt. In der Tabelle 7.1 zeigt sich zudem die Beobachtung, dass der durchschnittliche Abstand bei einigen Befehlen zum Vorgänger verringert ist. Dies resultiert aus einem gegenseitigen Ausgleich der Abweichungen in den verschiedenen Bewegungen.

Im zweiten Teilversuch werden die berechneten durchschnittlichen Abweichungen mithilfe von Validationsradien überprüft. Die Ergebnisse zeigen, dass die Operationen bei jedem ausgewählten Validationsradius tendenziell früher unterbrochen werden, als es in der Auswertung vorgesehen war. Insbesondere ist festzustellen, dass die Operation am häufigsten bei den `Stop`-Befehlen fehlschlägt. Es ist anzunehmen, dass die inkonsistente Fluggeschwindigkeit, auf die zuvor hingewiesen wurde, die Ursache für diesen Fehler ist. Diese Annahme wird dadurch gestützt, dass vor dem `Stop`-Befehlen eine Bewegung stattgefunden hat.

Basierend auf den Ergebnissen der beiden Teilversuche lässt sich abschließend sagen, dass das RRN die gestellten Anforderungen erfüllt und eine Trajektorie nachstellen kann. Allerdings wird die Route nur ungenau abgebildet, was darauf hinweist, dass noch erhebliche Verbesserungen an der Operation vorgenommen werden müssen. Des Weiteren demonstriert dieser Versuch, dass die entwickelten Architekturen die Zusammenarbeit der Komponenten ermöglichen.

8.2 Fazit und Ausblick

Das Hauptziel dieser Arbeit bestand darin, einen DZ für eine Tello zu entwickeln und es als MAS mit dem MARS-Framework umzusetzen. Das entwickelte Drohnensystem präsentiert einen ersten Ansatz, um die Tello oder andere UAVs innerhalb des MARS-Frameworks zu modellieren und mithilfe einer Routine die Trajektorie und den Zustand abzubilden. Zusätzlich stellt das System eine Architektur vor, die eine Schnittstelle für den Datenaustausch mit der Tello integriert und Erweiterungen durch weitere Komponenten ermöglicht. Ein Beispiel ist die entwickelte Fernsteuerung. Durch eine Operation wird demonstriert, dass der DZ nicht nur passiv den Zustand der Tello abbildet, sondern auch aktiv ändern kann. Die Ergebnisse der durchgeführten Versuche legen nahe, dass die entwickelten Funktionalitäten verbessert werden können. Dennoch stellt dieses System einen vielversprechenden Ansatz dar und setzt eine solide Grundlage für zukünftige Forschungsarbeiten.

Die offene Architektur des entwickelten Systems bietet vielfältige Möglichkeiten zur Erweiterung zusätzlicher Funktionalitäten an. Eine dieser Möglichkeiten besteht darin, das System um eine Simulation zu ergänzen. Durch die Integration der vorhandenen Daten über die Tello könnte ein virtuelles Modell entwickelt werden, das der Tello im Flugverhalten nachkommt. Durch das Hinzufügen von Hindernissen und neuen Layer könnte die virtuelle Tello in verschiedenen Szenarien erprobt und untersucht werden. Ein weiterer möglicher Ansatz besteht darin, weitere Tello-Drohnen in das MAS zu integrieren, um kooperative Operationen auszuführen.

Literaturverzeichnis

[1] URL https://www.mars-group.org/docs/category/development. – [abgerufen am 24.05.2023]

[2] *Tello SDK 1.3.0.0.* – URL https://dl-cdn.ryzerobotics.com/downloads/tello/20180910/Tello%20SDK%20Documentation%20EN_1.3.pdf. – [abgerufen am 24.05.2023]

[3] *Tellodownloads.* – URL https://www.ryzerobotics.com/de/tello/downloads. – [abgerufen am 23.05.2023]

[4] Al-Mousa, Amjed ; Sababha, Belal H. ; Al-Madi, Nailah ; Barghouthi, Amro ; Younisse, Remah: UTSim: A framework and simulator for UAV air traffic integration, control, and communication. In: *International Journal of Advanced Robotic Systems* 16 (2019), Nr. 5, S. 1729881419870937. – URL https://doi.org/10.1177/1729881419870937

[5] Chivarov, Nayden ; Chikurtev, Denis ; Stoev, Petko ; Lozanov, Vasil ; Chivarov, Stefan: ROBCO Drone - Service Robot for Transport and Delivery of Grocery Products. In: *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, 2021, S. 1–6

[6] Dorri, Ali ; Kanhere, Salil S. ; Jurdak, Raja: Multi-Agent Systems: A Survey. In: *IEEE Access* 6 (2018), S. 28573–28593

[7] Eloncase: *RyzeTelloSDK.* https://github.com/Eloncase/RyzeTelloSDK. 2023. – [abgerufen am 05.06.2023]

[8] Europäisches Parlament ; Rat der Europäischen Union: *Verordnung (EU) 2019/947 des Europäischen Parlaments und des Rates vom 5. Juni 2019 über die Vorschriften und Verfahren für den Betrieb unbemannter Flugzeuge.* 2019. – URL https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32019R0947&from=EN. – [abgerufen am 07.05.2023]

[9] Grieves, Michael: Origins of the Digital Twin Concept. In: *None* (2016), 08

[10] Görz, Günther (Hrsg.) ; Schmid, Ute (Hrsg.) ; Braun, Tanya (Hrsg.): *Handbuch der Künstlichen Intelligenz*. Berlin, Boston : De Gruyter Oldenbourg, 2021. – URL https://doi.org/10.1515/9783110659948. – ISBN 9783110659948

[11] Hüning, Christian ; Wilmans, Jason ; Feyerabend, Nils ; Clemen, Thomas: MARS - A next-gen multi-agent simulation framework, 04 2014, S. 1–14

[12] Flight dynamics — Concepts, quantities and symbols — Part 2: Motions of the aircraft and the atmosphere relative to the Earth / International Organization for Standardization. Geneva, CH, September 1985. – Standard

[13] Ji, Guang ; Hao, Jian-guo ; Gao, Jia-long ; Lu, Cheng-zhao: Digital Twin Modeling Method for Individual Combat Quadrotor UAV. In: *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, S. 1–4

[14] Julio Alberto Mendoza-Mendoza, Carlos Fernando Aguilar-Ibañez ; Fonseca-Ruiz, Leonardo: *Drones to Go: A Crash Course for Scientists and Makers*. 1. Apress Berkeley, CA, 2021. – ISBN 978-1-4842-6787-5

[15] Lei, Lei ; Shen, Gaoqing ; Zhang, Lijuan ; Li, Zhilin: Toward Intelligent Cooperation of UAV Swarms: When Machine Learning Meets Digital Twin. In: *IEEE Network* 35 (2021), Nr. 1, S. 386–392

[16] Lv, Zhihan ; Guo, Jinkang: Application of Digital Twins in multiple fields. In: *Multimedia Tools and Applications* 81 (2022), S. 26941–26967. – https://link.springer.com/article/10.1007/s11042-022-12536-5

[17] Macal, C.M. ; North, M.J.: Tutorial on agent-based modeling and simulation. In: *Proceedings of the Winter Simulation Conference, 2005.*, 2005, S. 14 pp.–

[18] Meng, Wei ; Yang, Yuanlin ; Zang, Jiayao ; Li, Hongyi ; Lu, Renquan: DTUAV: a novel cloud–based digital twin system for unmanned aerial vehicles. In: *SIMULATION* 99 (2022), 08, S. 003754972211095

[19] Obradović, Ratko ; Vasiljević, Ivana ; Kovačević, Dušan ; Marinković, Zoran ; Farkas, Robert: Drone Aided Inspection during Bridge Construction. In: *2019 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 2019, S. 1–4

[20] Saha, Apurv ; Kumar, Akash ; Sahu, Aishwary K.: FPV drone with GPS used for surveillance in remote areas, 2017, S. 62–67

[21] Segovia, Mariana ; Garcia-Alfaro, Joaquin: Design, Modeling and Implementation of Digital Twins. In: *Sensors* 22 (2022), Nr. 14. – URL https://www.mdpi.com/1424-8220/22/14/5396. – ISSN 1424-8220

[22] Yang, Yuanlin ; Meng, Wei ; Li, Hongyi ; Lu, Renquan ; Fu, Minyue: A Digital Twin Platform for Multi-Rotor UAV. In: *2021 40th Chinese Control Conference (CCC)*, 2021, S. 7909–7913

[23] Yang, Yuanlin ; Meng, Wei ; Zhu, Shiquan: A Digital Twin Simulation Platform for Multi-rotor UAV. In: *2020 7th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*, 2020, S. 591–596

A Anhang

Tabelle A.1: Fluginformationen der Tello

Bescheibung	**Einheit**
Neigungwinkel um die x-Achse	$^\circ$
Neigungwinkel um die y-Achse	$^\circ$
Neigungwinkel um die z-Achse	$^\circ$
Niedrigste gemessene Temperatur	$^\circ C$
Höchste gemessene Temperatur	$^\circ C$
Fluggeschwindigkeit in x-Richtung	$10^{-2}\frac{m}{s}$
Fluggeschwindigkeit in y-Richtung	$10^{-2}\frac{m}{s}$
Fluggeschwindigkeit in z-Richtung	$10^{-2}\frac{m}{s}$
Flughöhe gemessen vom Distanzsensors	$10^{-2}m$
Flughöhe gemessen vom Ausgangspunkt	$10^{-2}m$
Systemlaufzeit	s
Batteriestand des Akkus	%
Beschleunigung in x-Richtung	$10^{-2}\frac{m}{s^2}$
Beschleunigung in y-Richtung	$10^{-2}\frac{m}{s^2}$
Beschleunigung in z-Richtung	$10^{-2}\frac{m}{s^2}$

Tabelle A.2: Tastenbelegung für Tastatursteuerung

Aktion	**Taste**
Vorwärts	W
Rückwärts	S
Links	A
Rechts	D
Stop	Space
Rotieren im Uhrzeigersinn	E
Rotieren gegen den Uhrzeigersinn	Q
Abheben	T
Landen	L
Notstop	P
Batteriestand	B
Record-Repeat Navigation starten	U
Record-Repeat Navigation stoppen	I
Record-Repeat Navigation Aufzeichnung anhalten	Delete

Tabelle A.3: Flugzustände der Drohne

Zustand	**Beschreibung**
Unknown	Nicht definierter Zustand
Standby	Auf den Boden
Hovering	An einem Punkt in der Luft schwebend
TakingOff	Am abheben
Landing	In der Landung
MovingForwards	Fliegend nach vorne
MovingBackwards	Fliegend nach hinten
MovingLeft	Fliegend nach links
MovingRight	Fliegend nach rechts
RotatingClockwise	Rotierend im Uhrzeigersinn
RotatingCounterClockwise	Rotierend gegen den Uhrzeigersinn
Rising	Steigend nach oben
Sinking	Senkend zum Boden

A.1 Ergebnisse des zweiten Versuchs

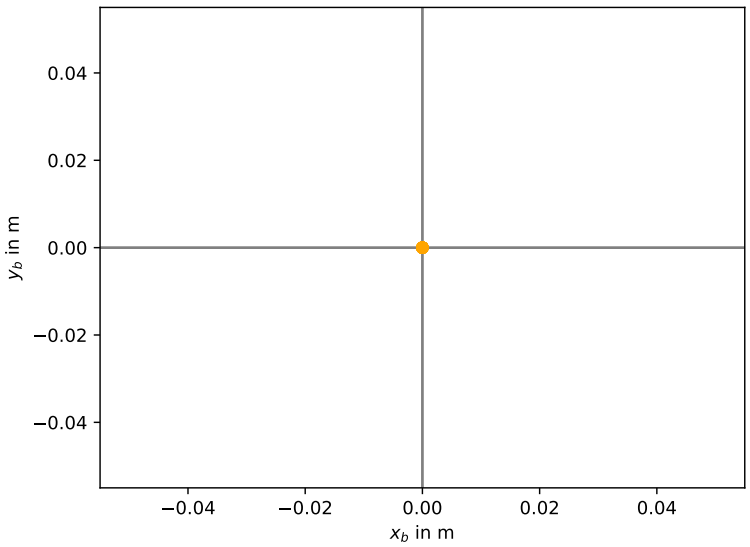


Abbildung A.1: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 1. Befehls

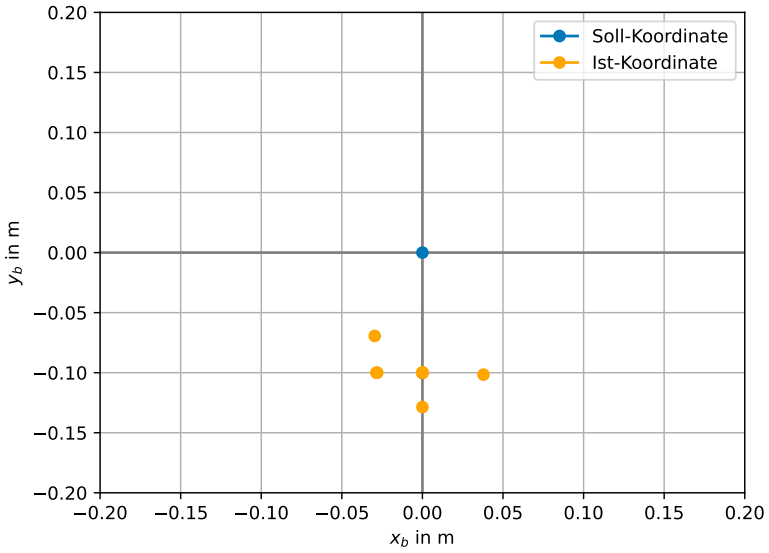


Abbildung A.2: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 2. Befehls

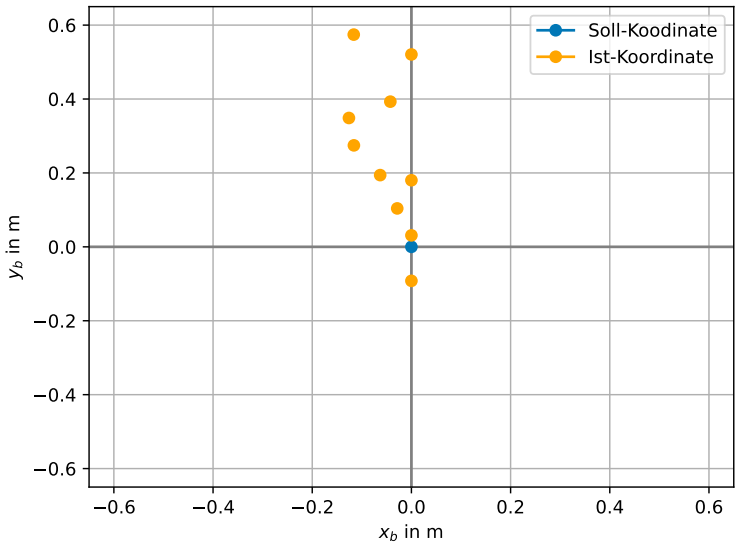


Abbildung A.3: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 3. Befehls

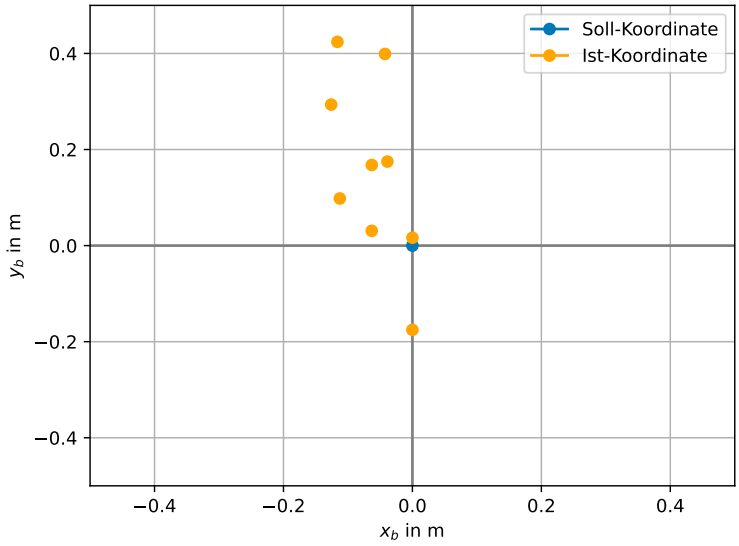


Abbildung A.4: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 4. Befehls

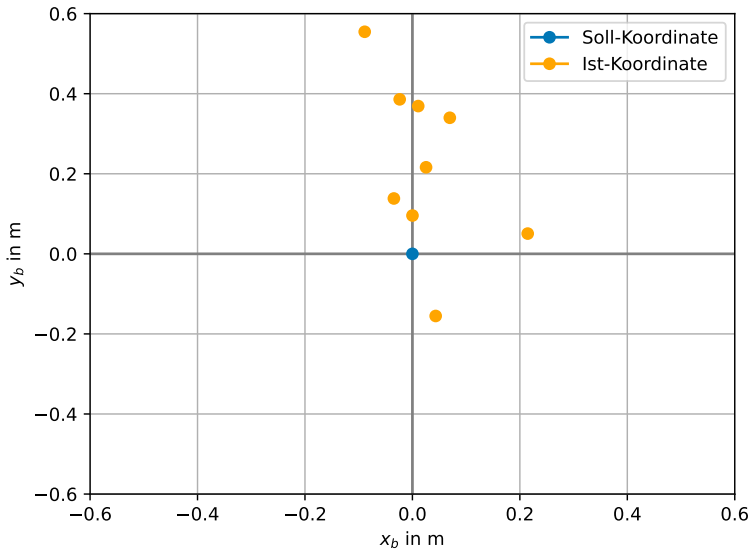


Abbildung A.5: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 5. Befehls

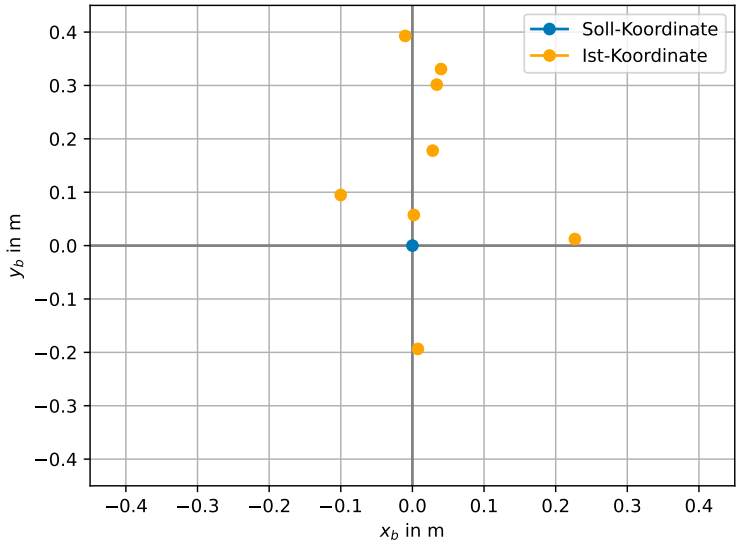


Abbildung A.6: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 6. Befehls

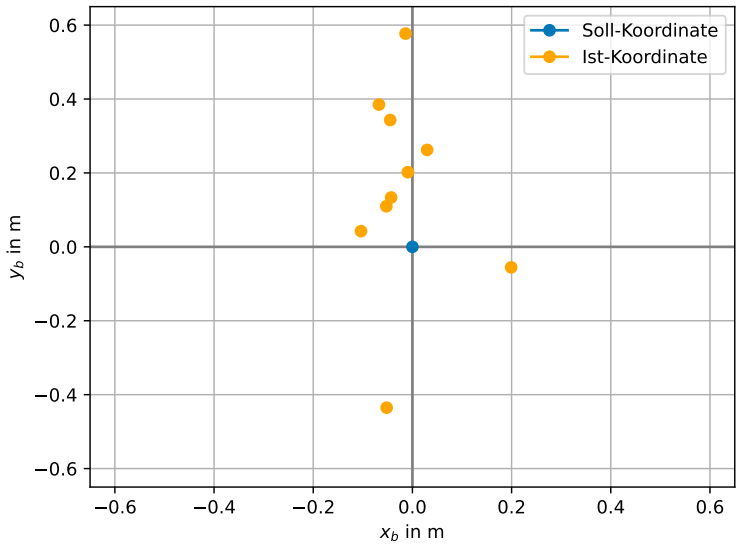


Abbildung A.7: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 7. Befehls

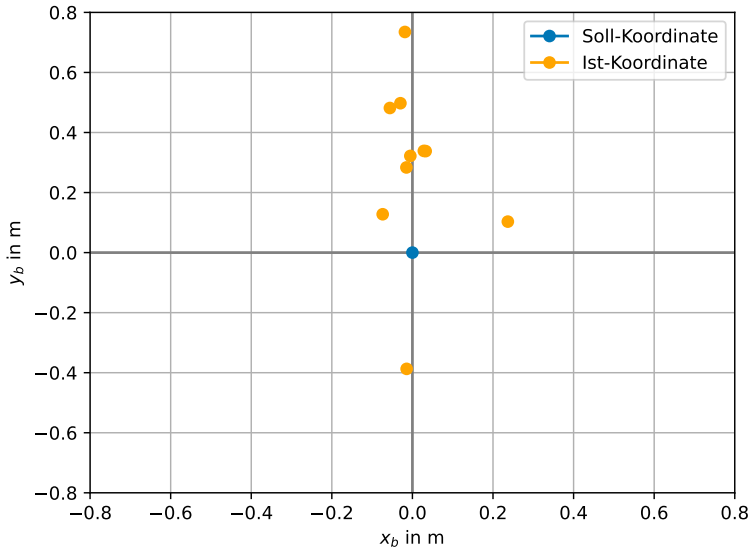


Abbildung A.8: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 8. Befehls

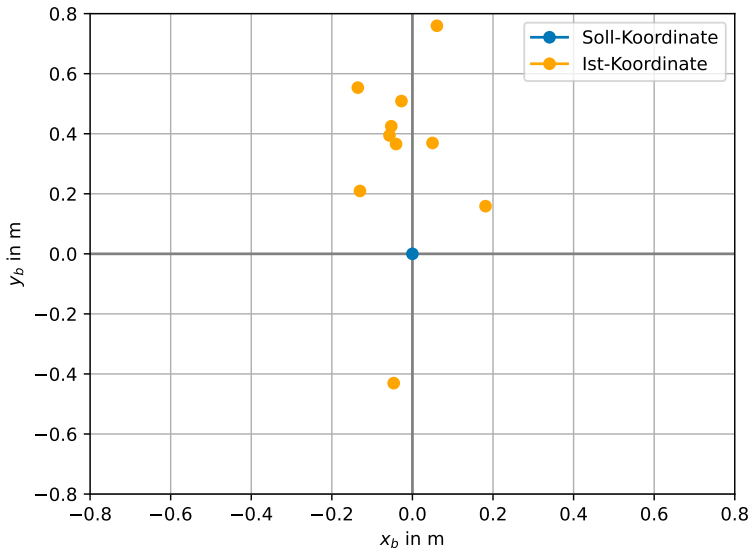


Abbildung A.9: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 9. Befehls

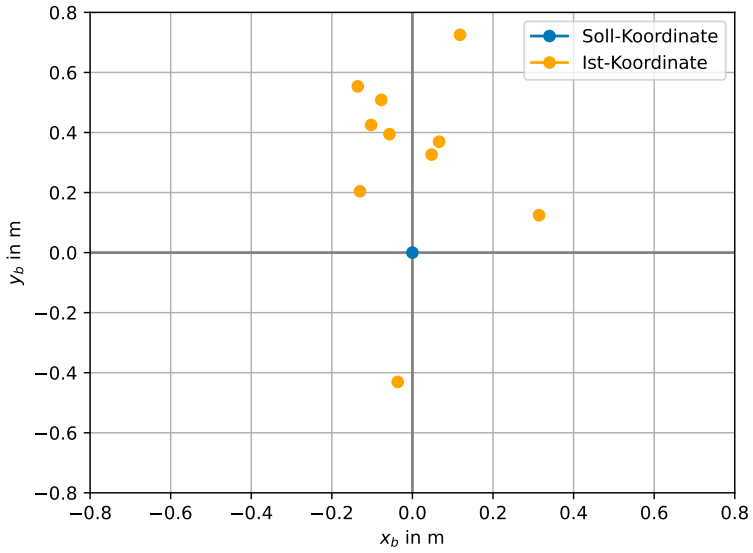


Abbildung A.10: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 10. Befehls

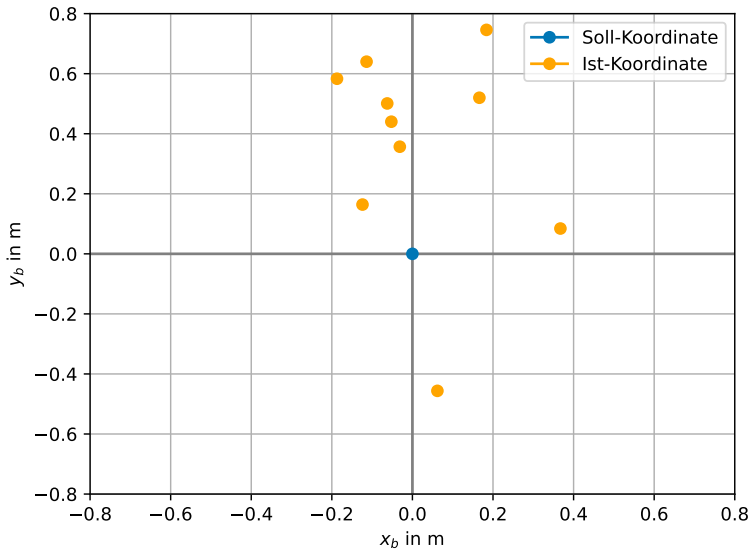


Abbildung A.11: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 11. Befehls

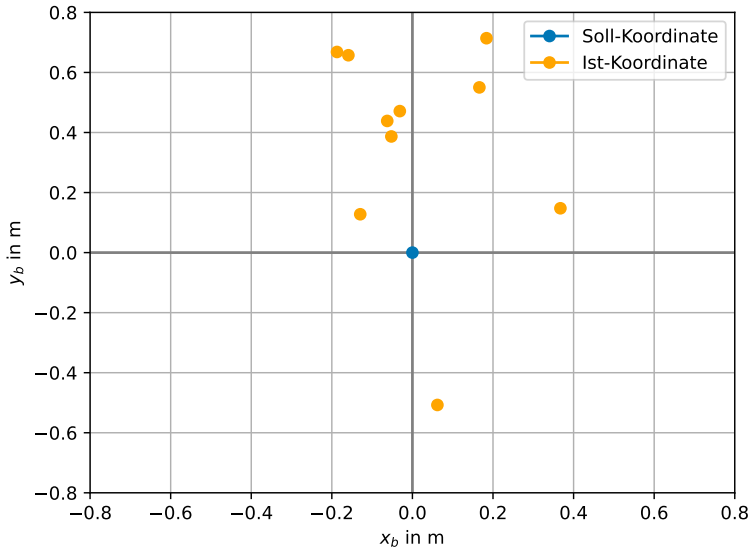


Abbildung A.12: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 12. Befehls

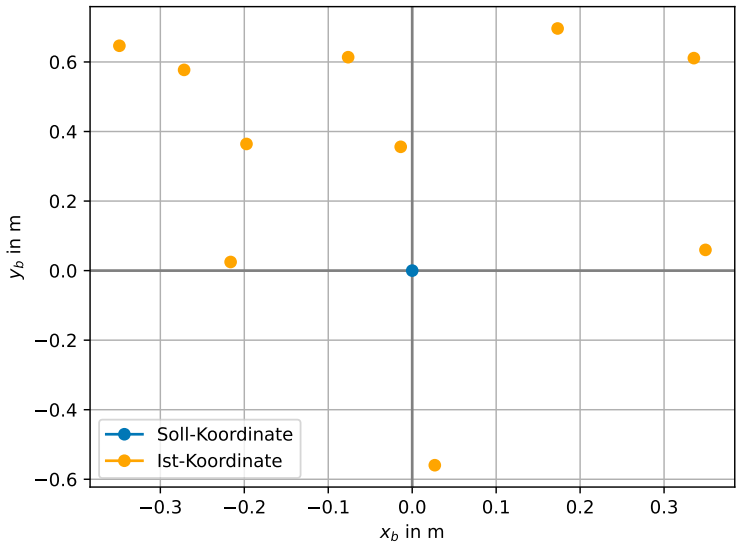


Abbildung A.13: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 13. Befehls

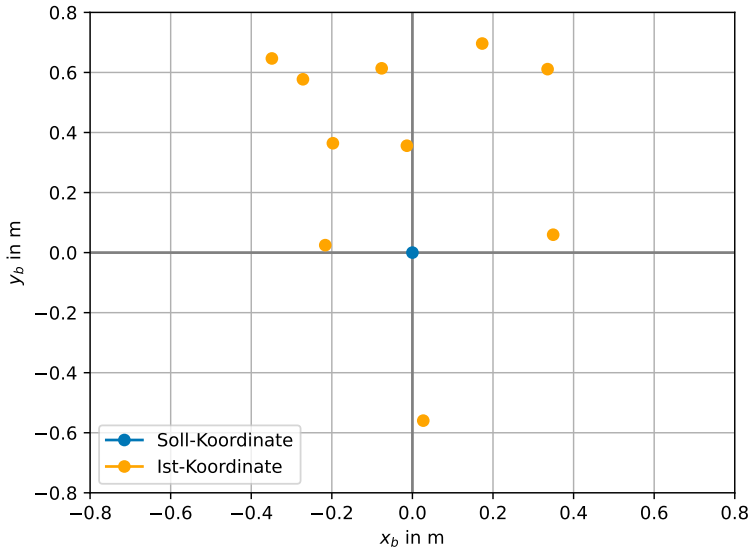


Abbildung A.14: Abweichungen der Ist-Koordinaten von der Soll-Koordinate des 14. Befehls

Glossar

Cockpit Das Cockpit bezeichnet eine Instrumententafel, auf der alle wichtigen Informationen über ein Fluggerät dargestellt werden.

Fernpilot Eine Person, welche ein Luftfahrzeug über eine Fernsteuerung steuert.

IEEE-802.11 Das IEEE-802.11-Protokoll oder im allgemeinen Sprachgebrauch als WLAN bezeichnet ist ein drahtloses Netzwerkprotokoll, um Nachrichten von einem Teilnehmer zu einen anderen Teilnehmer zu verschicken. Der Nachrichtenverkehr findet entweder auf einer Radiofrequenz vonn 2,4 GHz oder auf 5 GHz statt.

Multikopter Ein Multikopter ist ein Fluggerät, das zwei oder mehr Propeller besitzt, die senkrecht angeordnet sind. Sie können senkrecht vom Boden abheben und an einer Position in der Luft schweben.

UDP-Port Eine Kommunikationsschnitte an der UDP-Nachrichten verschickt werden können. Sender wissen über eine Portnummer, ob sie ihre Nachricht an den richtige Kanal verschicken.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original