Hochschule Neubrandenburg
University of Applied Sciences

Hochschule Neubrandenburg

# Prototypische Entwicklung einer Software zur Planung, Visualisierung und Dokumentation UAV-gestützter Gasfreimessungen in Ballastwassertanks von Schiffen unter Verwendung der Unity 3D Game Engine

MASTERTHESIS
zur Erlangung des akademischen Grades
Master of Engineering (M.Eng.)

vorgelegt von *Jupp Otto*

**Geprüft von:**
Prof. Dr.-Ing. Tobias Hillmann
Prof. Dr.-Ing. Sven Brämer

**Vorgelegt am:**
15.05.2019

**urn:**
urn:nbn:de:gbv:519-thesis2018-0868-7

# Kurzfassung

Die zunehmende Digitalisierung des maritimen Wirtschaftssektors erfordert unter anderem neue Lösungen zur Automatisierung von Schiffsbauprozessen, wie zum Beispiel im Bereich des Arbeitsschutzes und der Qualitätssicherung. Im Rahmen dieser beiden Teilbereiche befasst sich die vorliegende Arbeit mit der UAV-gestützten Ermittlung des Gefahrenpotentials durch Gase, der sogenannten Gasfreimessung, in Ballastwassertanks von Schiffen sowie der Objektdokumentation ebendieser. Hierzu wird eine Software konzipiert und prototypisch implementiert, die dem Benutzer eine grafische Oberfläche zur Planung und Visualisierung automatisierter Messflüge zur Verfügung stellt. Dies erfolgt unter Verwendung der Entwicklungsumgebung Unity 3D, welche primär zur Erstellung von 3D Cross-Plattform-Spielen eingesetzt wird. Mit der Bereitstellung von Komponenten und Konzepten zur Verarbeitung und Visualisierung dreidimensionaler Modelle und Geodaten liefert diese Spiele-Engine jedoch auch ideale Voraussetzungen für die hier gegebene Problematik. Die entwickelte Softwarelösung bietet zudem ein Werkzeug zur Berechnung von Funkausleuchtungen im Hochfrequenzbereich sowie zum Importieren und Verwalten von Abstrahlcharakteristiken. Das ausgearbeitete Funkausbreitungsmodell stützt sich auf bereits bekannte Konzepte physikalischer und empirischer Modelle und kombiniert diese zu einem semi-physikalischen Modell. Ein weiterer Teilaspekt der Software ist die Visualisierung der durch das Modell berechneten Empfangsleistungen. Die abschließende Gasfreimessung des Ballastwassertanks erfolgt auf Grundlage der erhobenen Messdaten in Kombination mit einer geostatistischen Simulation. Als Simulationsverfahren wird die konditionierte sequentielle Gauß'sche Simulation genutzt, deren Umsetzung mit der Softwareumgebung *R* erfolgt und durch eine Schnittstelle in die Planungslösung eingebunden wird. Eine Validierung der Simulations- sowie Funkausbreitungsergebnisse findet im Rahmen dieser Arbeit nicht statt und bleibt Bestandteil weitergehender Untersuchungen.

# Abstract

The increasing digitalisation of the maritime economic sector requires, amongst other things, new solutions for the automation of shipbuilding processes, such as in the fields of occupational safety and quality assurance. In the context of those two subjects the present work deals with the UAV-based determination of hazard potentials by gases, the so-called gas based clearance measurement, in ballast water tanks of vessels as well as the object documentation of those. For this purpose a software is designed and prototypically implemented, which provides the user a graphical user interface for planning and visualizing automated measurement flights. This is done by using the development environment Unity 3D, that is primarily known from creating 3D cross-platform games. By providing components and concepts to process and visualize three-dimensional models and geodata, the game engine also features ideal conditions for the given issue. The developed software solution includes a tool to calculate the radio coverage in high frequency ranges as well as to import and manage radiation characteristics. The designed radio propagation model is based on already known concepts of physical and empirical models and combines them to a semi-physical model. A further aspect of the software is the visualization of the reception power calculated by the model. The final gas based clearance measurement of the ballast water tank is performed with the collected measurement data in combination with a geostatistical simulation. Therefore the conditioned sequential gaussian simulation is used as simulation method, which is implemented with the software environment *R* and integrated into the solution via an interface. A validation of the simulation and radio propagation results does not take place within the scope of this work and remains as a part of further investigations.

# Danksagung

An dieser Stelle möchte ich mich bei all denen bedanken, die mich bei meiner Masterarbeit sowohl fachlich als auch persönlich unterstützt haben.

Ich Danke Prof. Dr.-Ing. Tobias Hillmann und Prof. Dr.-Ing. Sven Brämer für die fachliche Beratung und die wertvollen Hinweise, mit denen diese Arbeit an vielen Stellen bereichert werden konnte. Zudem möchte ich Prof. Dr.-Ing. Tobias Hillmann für die Bereitsellung des Themas danken.

Einen besonderen Dank möchte ich Dipl.-Ing. Olaf Keitsch für die enge und ertragreiche Zusammenarbeit innerhalb des Forschungsprojektes GOBIM aussprechen. Er stand mir mit viel Geduld, Interesse und Hilfsbereitschaft zur Seite, wobei die gemeinsamen Debatten und Ideen, diese Masterarbeit maßgeblich geprägt haben. Des Weiteren möchte ich mich bei ihm für die Vermittlung der Masterarbeit sowie für unsere gemeinsame Studienzeit bedanken.

Außerdem danke ich meiner Freundin für ihre Geduld sowie für die hilfreichen Korrekturen, mit denen die Arbeit an vielen Stellen verbessert werden konnte.

Nicht zuletzt danke ich meiner Mutter für ihre finanzielle Unterstützung während der Ausarbeitungszeit dieser Arbeit.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

## Abkürzungsverzeichnis

**API** Application Programming Interface

**AP** Access Point

**BMWi** Bundesministerium für Wirtschaft und Energie

**BSS** Basic Service Set

**CSV** Comma-Separated Values

**DFKI** Deutsches Forschungszentrum für Künstliche Intelligenz

**DP** Durchlaufpunkt

**DSSS** Direct Sequence Spread Spectrum

**DS** Distributed System

**EIRP** Effective Isotropic Radiated Power

**ESS** Extended Service Set

**FHSS** Frequency Hopping Spread Spectrum

**FSL** Free Space Loss

**GOBIM** Gasfreimessung und Objektdokumentation von Ballastwassertanks von Schiffen durch mobile Indoor-UAV gestützte Messplattform

**GO** Geometrische Optik

**IEEE** Institute of Electrical and Electronics Engineers

**iOS** iPhone Operating System

**IP** Interaktionspunkt

**MJPEG** Motion Joint Photographic Experts Group

**OFDM** Orthogonal Frequency Division Modulation

**OSDK** Onboard Software Development Kit

**POI** Point of Interest

**ROT** Robots in Tanks

**RSL** Received Signal Level

**SDK** Software Development Kit

**SGS** Sequentielle Gauß'sche Simulation

**SSID** Service Set Identifier

**TCP** Transmission Control Protocol

**UART** Universal Asynchronous Receiver/Transmitter

**UAV** Unmanned Aerial Vehicle

**UV** Unmanned Vehicle

**WLAN** Wireless Local Area Network

**XML** Extensible Markup Language

**ZIM** Zentrales Innovationsprogramm Mittelstand

# Teil I
# Einleitung

Der maritime Sektor befindet sich, ebenso wie viele andere Wirtschaftszweige, im Prozess der Digitalisierung mit dem Ziel der Industrie 4.0. Hierbei sollen innovative Technologien die Automatisierung von Prozessen ermöglichen und somit zur Effizienzsteigerung in den Wertschöpfungsketten beitragen. Die Bedeutung der Digitalisierung der maritimen Wirtschaft und die damit verbundene Konkurrenzfähigkeit des Industrie- und Exportstandorts Deutschland, lassen sich sowohl durch die „*Gemeinsame Erklärung zur Digitalisierung in der maritimen Wirtschaft*“, als auch durch die „*Maritime Agenda 2025*“ verdeutlichen. Inhalt dieser, von der Bundesregierung und Akteuren aus der Branche, ausgearbeiteten Agenden sind unter anderem die gezielte Stärkung der Forschung und Entwicklung in der maritimen Wirtschaft durch Unterstützung mittelständischer Unternehmen. In diesem Rahmen entstehen zahlreiche Projekte im Kontext verschiedener Bereiche der Automatisierung von Schiffsbauprozessen, zu denen auch der Arbeitsschutz und die Qualitätssicherung zählen.

Ein Beispiel für solch ein Forschungsvorhaben ist das in (Christensen et al. 2011) beschriebene Projekt des DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz) ROT (RObots in Tanks), welches sich mit der Automatisierung von Inspektionen und Wartungen in Ballastwassertanks von Schiffen beschäftigt. Die in dem Tank durchgeführten Arbeiten werden dabei unter Verwendung autonomer schienengelenkter Roboter durchgeführt.

Einen anderen Ansatz zur Ballastwassertankinspektion verfolgt das, durch ZIM (Zentrales Innovationsprogramm Mittelstand) und BMWi (Bundesministerium für Wirtschaft und Energie), geförderte Forschungsprojekt GOBIM. Ziel des Projektes ist es ein System zu entwickeln, das eine UAV (Unmanned Aerial Vehicle)-gestützte Automatisierung von Arbeitsschutz- und Qualitätssicherungsprozessen in Ballastwassertanks ermöglicht. Hierbei sollen Mess- und Dokumentationsflüge durchgeführt werden. Die dabei aufgenommenen Daten dienen zur Bestimmung des Gefahrenpotentials, der im Ballastwassertank befindlichen Gase (Gasfreimessung). Des Weiteren ist die Aufnahme von Bilddaten, zur Objektdokumentation des Tanks, zu realisieren, welche beispielsweise zur Feststellung von Korrosion oder Ablagerungen Verwendung finden. Zur Umsetzung dieser Aufgaben, soll die UAV durch eine Innovative Indoor-Navigationslösung so erweitert werden, dass diese mittels Kollisionserkennung und Auto-Routing-Funktionalität, autark agiert.

Ziel dieser Arbeit ist die Konzipierung und prototypische Implementierung einer Softwarelösung zur Planung, Visualisierung und Auswertung UAV-gestützter Mess- und Dokumentationsflüge. Des Weiteren soll das Programm, ein Werkzeug zur Berechnung der Funkausleuchtung im Hochfrequenzbereich von 2,4 und 5 GHz bereitstellen. Durch die

Berechnung ist sicherzustellen, dass jederzeit eine WLAN- beziehungsweise Funkverbindung zum UAV besteht. Dies soll eine stetige Bildübertragung sowie eine Steuerung der Drohne im Fehlerfall gewährleisten. Im Rahmen der Funkausbreitungsberechnung ist die Entwicklung und softwaretechnische Implementierung eines geeigneten Modells notwendig. Zur Gasfreimessung des Ballastwassertanks muss das Programm eine Möglichkeit bieten die, während der Mission aufgenommenen, Daten auszuwerten. Hierdurch soll eine Entscheidungsgrundlage zur Einschätzung des Gefahrenpotentials innerhalb des Tanks geschaffen werden. Zur Datenauswertung wird eine geostatistische Simulation durchgeführt, welche erhobenen Gasmesswerte als Datengrundlage verwendet. Während Interpolationsverfahren, wie das Kriging, eine Glättung der Ausgangswerte bewirken, streben geostatistische Simulationen eine möglichst genaue Abbildung der räumlichen Variabilität hinsichtlich der Simulationswerte an. Hierdurch wird dem Über- und Unterschätzen von Extremwerten entgegengewirkt. Die softwaretechnische Realisierung des oben beschriebenen Vorhabens soll unter Verwendung einer Game Engine stattfinden. Diese bietet bereits Komponenten und Konzepte zur Verarbeitung und Visualisierung dreidimensionaler Modelle und Geodaten. Hierdurch soll eine Verringerung des Programmieraufwands von Grundfunktionalitäten der 3D-Visualisierung erreicht werden.

Zur Abgrenzung der Zielstellung wird verdeutlicht, dass die zu entwickelnde Software lediglich als Planungs-, Dokumentations- und Auswertungstool für die, von der UAV durchzuführenden Missionen, dient. Das Programm setzt keine Navigations- oder Routing-Funktionalitäten um. Es definiert lediglich Schnittstellen zur Kommunikation zwischen den Softwarelösungen. Eine Validierung der Simulations- sowie Funkausbreitungsergebnisse findet im Rahmen dieser Arbeit nicht statt und bleibt Bestandteil weitergehender Untersuchungen.

Der zweite Teil dieser Arbeit beschäftigt sich mit den Grundlagen, die zur Umsetzung der Planungslösung notwendig sind. Hierzu wird zunächst auf die Softwarearchitektur und Datenstrukturen des *DJI* Onboard SDK eingegangen. Zur Entwicklung eines, im Hochfrequenzbereich operativen, Funkausbreitungsmodells wird die WLAN-Standardfamilie 802.11 beschrieben. Des Weiteren wird ein Überblick der relevanten Antennenparameter und gängigen Funkausbreitungsmodelle gegeben. In den geostatistischen Grundlagen wird der theoretische Hintergrund über die durchzuführende geostatistische Simulation erläutert und die programmtechnische Umsetzung einer konditionierten sequentiellen Gauß'schen Simulation gezeigt. Abschließend erfolgt die Beschreibung relevanter Grundlagen der Unity 3D Game Engine, wobei auf elementare und für die Arbeit relevante Konzepte der Engine eingegangen wird.

Im dritten Teil der Arbeit wird die Analyse, der Entwurf sowie die Implementierung der Planungslösung behandelt. Hierbei werden die funktionalen und nicht-funktionalen Softwareanforderungen definiert und das Gesamtsystem sowie dessen Schnittstellen dargestellt. Des Weiteren erfolgt die Beschreibung der, durch die Planungslösung, automatisierten Prozessabläufe. Abschließend wird das entwickelte Funkausbreitungsmodell

erläutert. Der Implementierungsteil zeigt die softwaretechnische Umsetzung, der funktionalen Anforderungen, auf. Aus Gründen des Umfangs werden hierbei lediglich die Hauptfunktionalitäten der Software betrachtet.

Teil Vier fasst die vorliegende Arbeit zusammen und diskutiert einzelne Aspekte der Implementierung.

# Teil II
# Grundlagen

Der zweite Teil dieser Arbeit beschäftigt sich mit den, für die Umsetzung der Planungslösung relevanten, Grundlagen. Hierzu wird zunächst der Drohnenbegriff definiert und auf die Softwarearchitektur und Datenstrukturen des *DJI* Onboard SDK beschrieben. Zur Entwicklung und Integration eines passenden Funkausbreitungsmodells, in die Softwarelösung, wird die WLAN-Standardfamilie 802.11 beschrieben und ein Überblick relevanter Antennenparameter und Funkausbreitungsmodelle gegeben. Die geostatistischen Grundlagen bilden den theoretischen Hintergrund der durchzuführenden geostatistischen Simulation. Weiter wird die Umsetzung einer konditionierten sequentiellen Gauß'schen Simulation anhand der Programmiersprache und Softwareumgebung *R* vermittelt. Abschließend erfolgt eine Erläuterung relevanter Grundlagen zur Unity Game Engine. Diese beinhalten elementare Konzepte der Engine, wobei speziell auf Funktionen der Physikengine sowie auf Lichtquellen und das 3D-Modellformat *Wavefront OBJ* eingegangen wird.

# 1 Technische Grundlagen

Dieses Kapitel behandelt die, im Rahmen der Arbeit, notwendigen technischen Grundlagen. Hierzu erfolgt zunächst eine Beschreibung des *Onboard Software Development Kit's* der *DJI-Matrice 100*, wobei insbesondere auf die Missionsverwaltung eingegangen wird. Des Weiteren wird die drahtlose Kommunikation über die *Wireless Local Area Network* (WLAN) Technologie behandelt. Hierzu wird die, von dem *Institute of Electrical and Electronic Engineers* (IEEE) entwickelte, Standardfamilie 802.11 näher betrachtet. Abschließend werden die, zur Modellierung und Berechnung der Funkausleuchtung, notwendigen Antennenparameter erläutert.

## 1.1 Unmanned Aerial Vehicle (UAV)

Als *Unmanned Aerial Vehicle* (UAV) werden Fluggeräte bezeichnet die im Stande sind, die ihnen aufgetragenen Missionen zum großen Teil eigenständig durchzuführen. UAV's stellen eine Untergruppe der *Unmanned Vehicle's* (UV's) dar, welche umgangssprachlich auch als Drohnen bezeichnet werden. Ein einheitlicher Klassifizierungsansatz für Drohnen existiert nicht. Eine Möglichkeit der Unterscheidung besteht in der Differenzierung anhand technischer Merkmale. Hierzu zählen beispielsweise Art und Anbringung der Tragflächen sowie die Art des Drohnenantriebes (Christen et al. 2018, S. 37-39).

Die im Rahmen des Forschungsprojekts verwendete Drohne *DJI Matrice 100*, ist ein Multicopter mit vier Rotoren, weshalb bei dieser Drohnenart auch von *quadrotor* oder *quadrocopter* gesprochen wird. Die sich gegenläufig drehenden Rotoren erzeugen dabei den Schub und ermöglichen der Drohne das vertikale Starten und Landen. Die Einsatzmöglichkeiten dieses und ähnlicher Systeme werden unter anderem für In- und Outdoor-Inspektionen, zur Aufnahme von Luftbildern oder zum Agrarmonitoring verwendet (Sa et al. 2018).

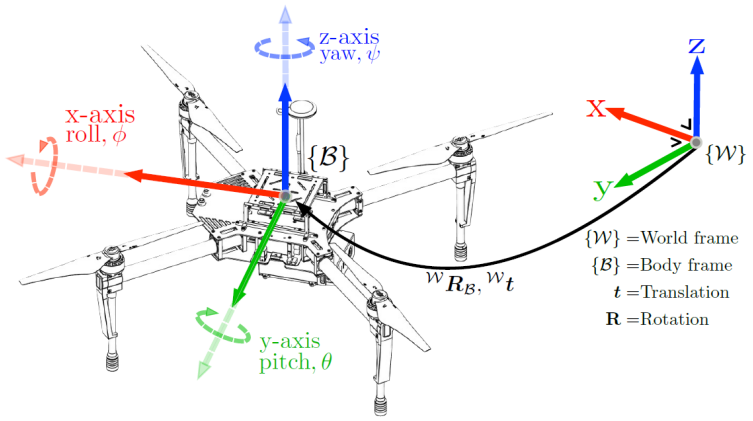



Abbildung 1: *DJI Matrice 100* mit Abbildung des globalen und lokalen Koordinatensystems (Sa et al. 2018).

### 1.1.1 DJI Onboard SDK

*DJI* stellt zur Entwicklung eigener Anwendungen für die *Matrice 100* drei *Software Development Kits* (SDK's) zur Verfügung. Während das *Mobile SDK* zur Realisierung von Android- und iOS-Applikationen dient, wird das *Guidance SDK* dazu verwendet, Anwendungen für das visuelle Sensor- und Positionierungssystem der *Matrice 100* zu erstellen. Das *Onboard SDK* (OSDK) wird zur Entwicklung von Anwendungen genutzt, die zur Steuerung des Flugverhaltens der Drohne dienen und ist somit zur Umsetzung einer autonomen Wegfindung sowie autonomer Flugmanöver einsetzbar. Die Anwendungen werden auf *eingebetteten Systemen* (engl. embedded systems) ausgeführt, welche über eine serielle Verbindung (UART) mit dem Flugcontroller der Drohne kommuniziert (Lu et al. 2017).

Abbildung 2 zeigt eine Übersicht der SDK's und deren Einbindung in verschiedene Entwicklungsumgebungen. Im Folgenden wird näher auf den Aufbau des OSDK's eingegangen, da dieses im Rahmen des Projektes zur Steuerung der Drohne verwendet wird.

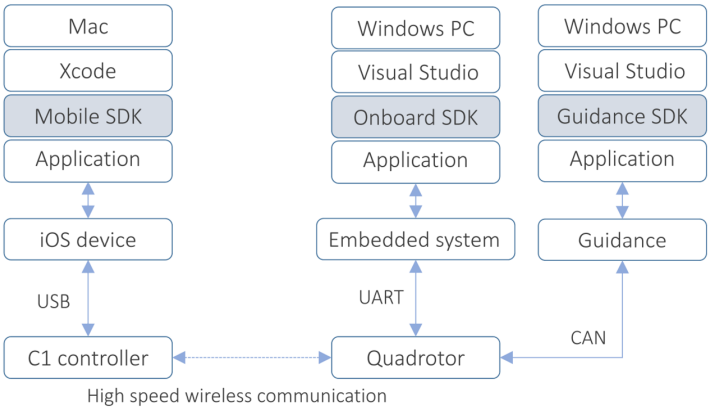



Abbildung 2: Übersicht der SDK's (grau hinterlegt) für die *DJI Matrice 100* (verändert nach Lu et al. 2017).

Abbildung 3 zeigt die vom OSDK bereitgestellten Klassen sowie deren Funktionen innerhalb des Development Kit's. Der Zugriff auf die Funktionalitäten und die Verwaltung des SDK's erfolgt über die Klasse *Vehicle* (DJI 2018f).

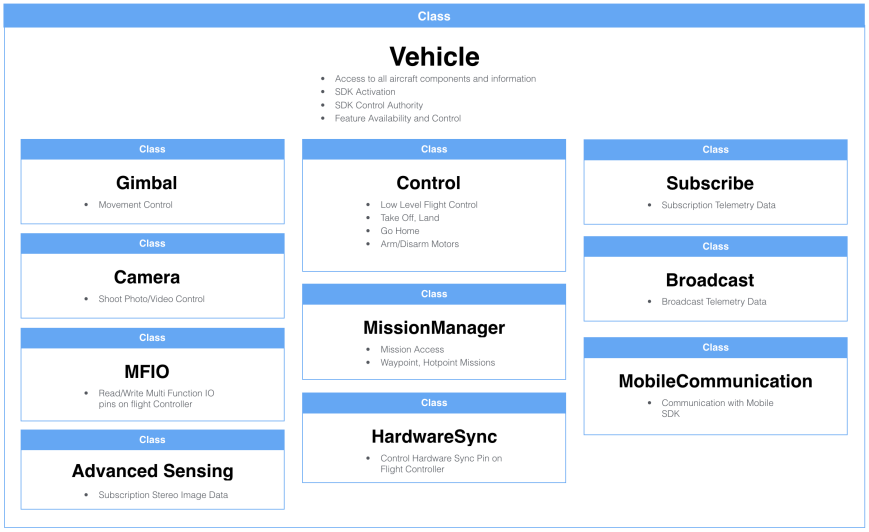



Abbildung 3: Klassenübersicht des Onboard Software Development Kit's (DJI 2018f).

### Missionsverwaltung

Die Missionsverwaltung wird durch die Klasse *MissionManager* realisiert. Das SDK sieht hierbei zwei Missionstypen vor, welche von der Drohne autonom ausgeführt werden (siehe Abbildung 4). Die *Hot Point Mission* lässt die Drohne wiederholt kreisförmig um einen *Point of Interest* (POI) fliegen. Parameter wie die Flughöhe, Geschwindigkeit, Kreisradius sowie die Ausrichtung der Drohne und die Koordinaten des POI werden vor Ausführung der Mission definiert. Die zweite Missionsart ist die sogenannte *Waypoint Mission*. Hierbei fliegt eine Drohne eine Folge vordefinierter Wegpunkte ab. Jedem Wegpunkt können *Aktionen* zugewiesen werden. Diese werden von der Drohne am *Wegpunkt* ausgeführt und beinhalten zum Beispiel die Aufnahme eines Fotos. Die maximale Anzahl von *Wegpunkten* ist im SDK auf 99 beschränkt (DJI 2018e).

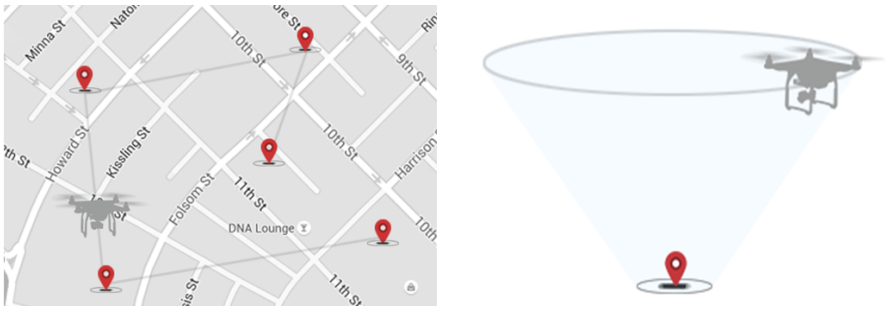



Abbildung 4: Beispielhafte Darstellung einer *Waypoint Mission* (links) und einer *Hot Point Mission* (rechts) (DJI 2018e).

In Abbildung 5 ist die vereinfachte Klassenstruktur der Missionsverwaltung einer *Waypoint Mission* innerhalb des OSDK's dargestellt.

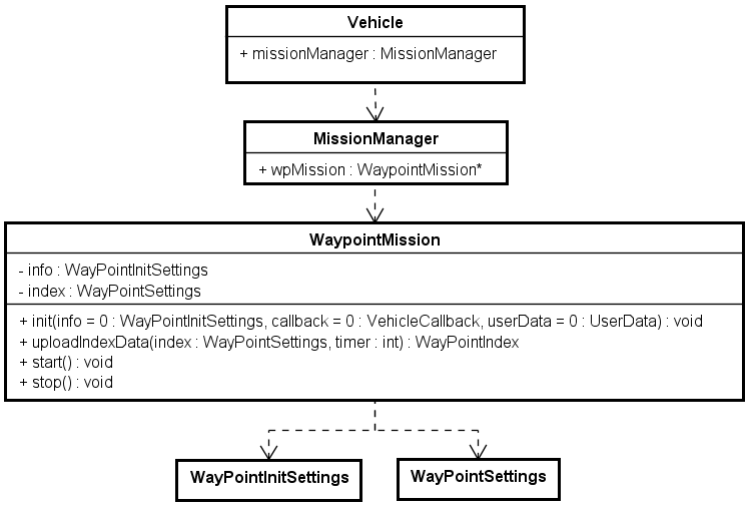



Abbildung 5: Klassendiagramm der Missionsverwaltung im OSDK (erstellt nach DJI 2018a).

Der *MissionManager*, welcher über die Klasse *Vehicle* erreichbar ist, hält die *Waypoint Mission* als Referenz (*wpMission : WaypointMission*). Mit Hilfe dieser Klasse wird eine Mission durch die *init*-Methode initialisiert, außerdem wird die Mission durch diese Klasse gestartet und gestoppt (*start() : void* und *stop() : void*). Zur Initialisierung wird der *init*-Methode eine Referenz auf die Struktur *WayPointInitSettings* übergeben (*info = 0 : WayPointInitSettings*). Diese hält die Missionsparameter der aktuellen *Waypoint Mission*. Abbildung 6 zeigt die Missionsparameter als Klassendiagramm. Eine Erläuterung der einzelnen Parameter ist in (DJI 2018b) zu finden.

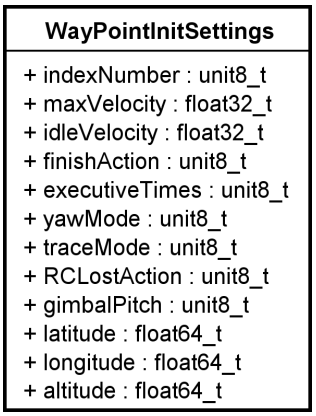



Abbildung 6: Klassendiagramm der Klasse *WayPointInitSettings* (erstellt nach DJI 2018b).

Die Methode *uploadIndexData* der Klasse *WaypointMission* dient dazu, die Wegpunkte in die aktuelle Mission zu übernehmen. Hierzu wird der Methode die Struktur *WayPointSettings* übergeben. Diese enthält alle, zur *Waypoint Mission* dazugehörigen,

Wegpunktparameter sowie die zu einem Wegpunkt dazugehörigen Aktionen. In Abbildung 7 ist die Klasse *WayPointSettings* mit den Wegpunktparametern dargestellt. Die Beschreibung aller Parameter ist (DJI 2018c) zu entnehmen.

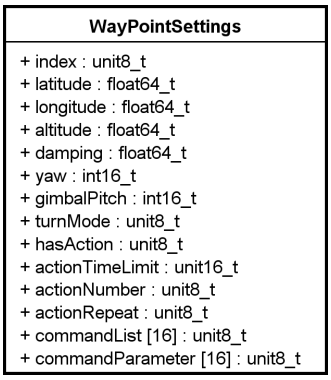



Abbildung 7: Klassendiagramm der Klasse *WayPointSettings* (erstellt nach DJI 2018c).

Die Variablen *commandList* und *commandParameter* der Klasse *WayPointSettings* dienen zur Speicherung der Aktionen sowie derer Parameter als vorzeichenlose 8 Bit-Integerwerte, wobei die Elementanzahl auf 16 beschränkt ist. Tabelle 1 enthält alle möglichen Aktionen sowie eine Beschreibung der dazugehörigen Parameter.

Tabelle 1: Übersicht der möglichen Aktionen der *DJI Matrice 100* (DJI 2018d).

| **Aktionsname** | **Wert** | **Parameter** | **Beschreibung** |
|---|---|---|---|
| WP_ACTION_STAY | 0 | Zeit in ms | In der Luft schweben |
| WP_ACTION_SIMPLE_SHOT | 1 | k.A. | Foto aufnehmen |
| WP_ACTION_VIDEO_START | 2 | k.A. | Video starten |
| WP_ACTION_VIDEO_STOP | 3 | k.A. | Video beenden |
| WP_ACTION_CRAFT_YAW | 4 | 180° bis -180° | yaw-Winkel justieren |
| WP_ACTION_GIMBAL_PITCH | 5 | 0° bis 90° | Gimbal-Winkel justieren |

## 1.2 Wireless LAN

Im Folgenden werden die, für die Berechnung der Funkausbreitung, notwendigen Grundlagen und Modelle betrachtet. Hierzu erfolgt eine Beschreibung der, in der IEEE 802.11 festgelegten, Übertragungsstandards zur drahtlosen Kommunikation über das WLAN. Weiter werden die im Rahmen der Arbeit verwendeten Antennenparameter beschrieben. Diese dienen zur Charakterisierung respektive zur Modellierung von Antennen. Abschließend werden Modelle zur Berechnung der Funkausbreitung vorgestellt, welche die Grundlage, des im Rahmen der Arbeit entwickelten Funkausbreitungsmodells bilden.

### 1.2.1 IEEE 802.11 Standards

Die Standardfamilie IEEE 802.11 spezifiziert die drahtlose Datenübertragung in einem *Wireless Local Area Networks* (WLAN), sowohl infrastrukturell als auch physikalisch und sorgt somit für eine herstellerunabhängige Kompatibilität unter dessen Netzwerkkomponenten. WLAN bezeichnet dabei die kabellose Erweiterung des klassischen *Local Area Network* (LAN). Generell eignet sich WLAN besonders unter Verwendung mobiler Endgeräte welche, innerhalb eines lokal begrenzten Abdeckungsbereichs, keinen festen Standpunkt besitzen. Die Einbindung des Endgerätes in das *WLAN* erfolgt nach IEEE 802.11 durch sog. *Access Points*. Diese dienen als Schnittstelle zwischen dem kabelgebundenem LAN und dem auf Funk basierenden WLAN. Der *Access Point* zusammen mit der dazugehörigen Funkzelle wird *Basic Service Set* (BSS) genannt. Einzelne *BSS* werden durch ein sog. *Distribution System* (DS) verbunden. Die Summe der mit einem *DS* verbundenen *BSS* wird als *Extended Service Set* (ESS) bezeichnet. Zur Identifikation eines *ESS*, wird diesem ein sogenannter *Service Set Identifier* (SSID) zugewiesen. Der oben beschriebene Aufbau wird auch als *Infrastruktur-Modus* bezeichnet (siehe Abbildung 8). Alternativ dazu existiert der *Ad-hoc-Modus*, in dem Clients, direkt ohne Zuhilfenahme eines AP's, miteinander kommunizieren. Findet eine direkte Kommunikation zweier Access Points statt, wird dies *LAN-Kopplung* genannt (Hoff et al. 2005, S. 9-12).

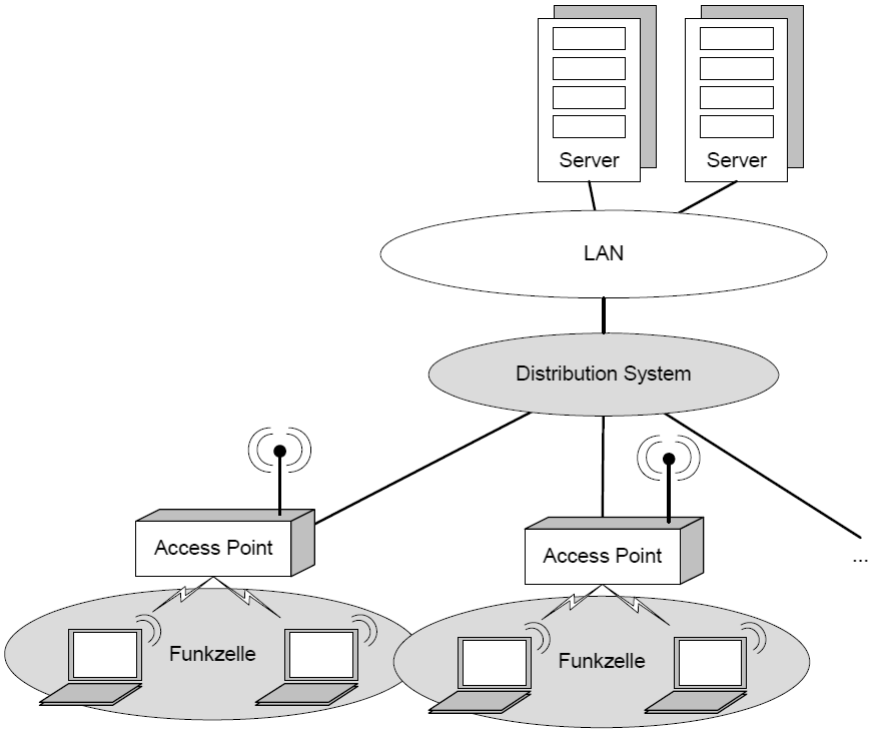



Abbildung 8: Infrastruktur eines WLAN nach IEEE 802.11 (Hoff et al. 2005, S. 11).

Die Substandards (IEEE 802.11b, a, g, h und n) unterscheiden sich sowohl in den zur Datenübertragung genutzten Frequenzbereichen, als auch in der zur Verfügung stehenden Bandbreite. Ein weiterer Unterschied liegt im genutzten Verfahren zur Frequenzspreizung (DSSS, FHSS oder OFDM), da dieses allerdings, im Rahmen dieser Arbeit nicht von Belang ist, wird hierauf nicht weiter eingegangen. Die Standarderweiterungen 802.11b (auch IEEE 802.11 HighRate) und 802.11g sind weitverbreitet und nutzen das 2,4 GHz ISM[1]-Frequenzband (2,400 GHz – 2,4835 GHz) zur Datenübertragung. Diese findet bei der 802.11b-Erweiterung mit einer maximalen Datenrate von 11 Mbps statt. Bei dem Substandard 802.11g beträgt die maximal erreichbare Datenrate 54 Mbps. Die Erweiterungen 802.11a und h sind weitestgehend identisch und nutzen zur Datenübertragung ein Frequenzband im 5 GHz Bereich (5,15 – 5,35 GHz bzw. 5,725 – 5,825 GHz). Die maximal erreichbare Datenrate beträgt dabei 54 Mbps. Der Substandard 802.11h erweitert 802.11a um eine automatische Sendeleistungsregelung. Diese soll die Störungen anderer Funkdienste im 5 GHz-Bereich (z.B. Militärfunk oder Flugnavigationsdienste in Europa) vermeiden (Preiner et al. 2006, S. 8-20). Die jüngste Erweiterung 802.11n, operiert sowohl auf dem 2,4 GHz als auch auf dem 5 GHz Band und bietet eine maximale Datenrate von 600 Mbps (Chen 2012, S. 3). Die oben genannten Kennzahlen sind in Tabelle 2 zusammengefasst.

Tabelle 2: Kennzahlen der WLAN Substandards.

| Erweiterung | Frequenzbereich [GHz] | max. Datenrate [Mbps] |
|---|---|---|
| 802.11b | 2,4 | 11 |
| 802.11a/h | 5 | 54 |
| 802.11g | 2,4 | 54 |
| 802.11n | 2,4/5 | 600 |

### 1.2.2 Antennenparameter

Eine Antenne ist eine elektromechanische Struktur mit Hilfe derer elektromagnetische Wellen ausgesendet bzw. empfangen werden. Hierbei kommt besonders die Richtcharakteristik zu tragen, welche die dreidimensionale Verteilung der abgegebenen Strahlung und somit die *Richtwirkung* der Antenne beschreibt. Weiter sind der *Antennenwirkungsgrad* und die *Direktivität* einer Antenne von großer Bedeutung. Aus diesen Größen lässt sich der *Antennengewinn* berechnen, mit dem sich die tatsächlich, von einer Antenne *effektiv abgestrahlte Sendeleitung* (EIRP) in Richtung des Strahlungsmaximums bestimmen lässt (Heuberger u. Gamm 2017, S. 150).

Die Charakteristik einer Antenne lässt sich durch zahlreiche Parameter beschreiben. Die oben genannten Größen werden im Folgenden näher beschrieben, da diese zur Berechnung der Funkausbreitung relevant sind.

[1] Industrial, Scientific and Medical Band

### Koordinatensystem

Der eine Antenne umgebene Raum wird durch ein sphärisches Koordinatensystem beschrieben. Dieses Koordinatensystem definiert den Azimutal- und Elevationswinkel $\theta$ und $\varphi$ sowie den Abstand zum Koordinatenursprung $r$ (siehe Abbildung 9).

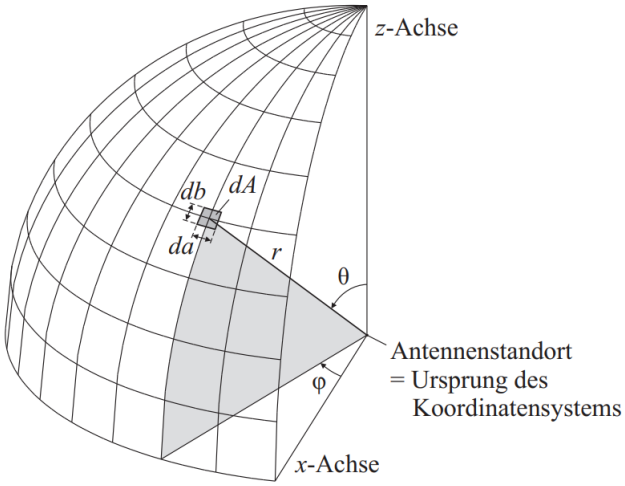



Abbildung 9: Sphärisches Antennenkoordinatensystem im $\mathbb{R}^3$ (Heuberger u. Gamm 2017, S. 152).

Die Kugelkoordinaten lassen sich durch folgende Formeln in ein kartesisches Koordinatensystem $(x, y, z)$ überführen.

$$\begin{aligned} x &= r\ cos\varphi\ sin\theta \\ y &= r\ sin\varphi\ sin\theta \\ z &= r\ cos\vartheta \end{aligned} \qquad \text{bzw. als Vektor} \qquad \vec{v} = r \begin{pmatrix} cos\varphi\ sin\theta \\ sin\varphi\ sin\theta \\ cos\theta \end{pmatrix} \tag{1}$$

Umgekehrt gelten folgende Formeln.

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= \arccos \frac{z}{\sqrt{x^2 + y^2 + z^2}} \end{aligned} \qquad \varphi = \begin{cases} \arctan \frac{y}{x} & \text{für } x > 0 \\ sgn(y)\frac{\pi}{2} & \text{für } x = 0 \\ \arctan \frac{y}{x} + \pi & \text{für } x < 0 \ \text{ und } y \geq 0 \\ \arctan \frac{y}{x} - \pi & \text{für } x < 0 \text{ und } y < 0 \end{cases} \tag{2}$$

### Richtfaktor und Antennengewinn

Eine Antenne die in alle Senderichtungen ($\theta$, $\varphi$) die exakt gleiche Strahlungsstärke aufweist wird als *isotroper Strahler* (*Kugelstrahler*) bezeichnet. Neben dieser Antenne, welche nur in der Theorie vorkommt, weisen real existierende Antennen, eine, in Abhängigkeit der Senderichtung, variierende Strahlungsintensität auf. Durch die Konzentration der Strahlungsintensität entsteht eine Hauptstrahlrichtung, in der die Anten-

ne scheinbar um ein vielfaches der Gesamtleistung sendet (siehe Abbildung 10). Dieser Effekt wird durch den *Richtfaktor* beschrieben, welcher sich wie folgt berechnet.

$$D = \frac{S_r(\vartheta, \varphi)_{max}}{\langle S_r(\vartheta, \varphi) \rangle} \tag{3}$$

Mit

| | |
|---|---|
| $D$ | als dimensionslosen Richtfaktor, |
| $S_r(\vartheta, \varphi)_{max}$ | als die Funktion der maximalen Strahlungsdichte und |
| $\langle S_r(\vartheta, \varphi) \rangle$ | als mittlere Strahlungsdichte bzw. Strahlungsdichte eines Kugelstrahlers. |

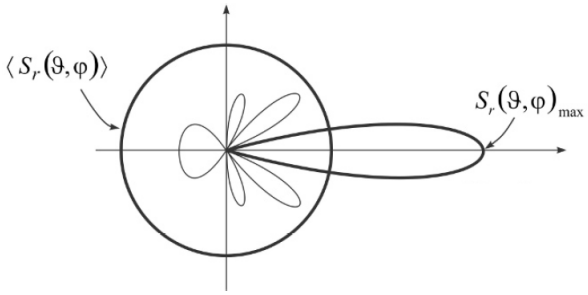



Abbildung 10: Strahlungsdichten einer realen Antenne und eines Kugelstrahlers (verändert nach Kark 2018, S. 252).

Der *Antennenwirkungsgrad* $\eta$ gibt das Verhältnis der abgestrahlten zur zugeführten Leistung wie folgt an

$$\eta = \frac{P_S}{P_S + P_V} \leq 1 \tag{4}$$

wobei,

| | |
|---|---|
| $\eta$ | der dimensionslose Wirkungsgrad, |
| $P_S$ | die abgestrahlte Leistung in mW und |
| $P_V$ | die Verlustleistung in mW ist. |

Das Produkt aus dem *Antennenwirkungsgrad* und dem *Richtfaktor* ist der *Antennengewinn G*. Dieser gibt das Verhältnis zwischen der maximalen Strahlungsdichte einer realen Antenne und der Strahlungsdichte eines isotropen Strahlers an, wobei beiden die gleiche Generatorleistung zur Verfügung steht. Gewöhnlich wird ein isotroper Strahler als Referenzantenne genutzt, wobei die Angabe des Antennengewinns logarithmisch erfolgt. Die daraus resultierende Einheit ist dBi, dabei steht das *"i"* für isotrop (Kark 2018, S. 252-253).

$$G = \eta \; D \qquad bzw. \qquad G_{dBi} = 10 \; lg \; G \tag{5}$$

### Antennendiagramm

Das *Antennendiagramm* dient dazu die Strahlungscharakteristik einer Antenne, in Abhängigkeit der Raumwinkel $\theta$ und $\varphi$, darzustellen. Hierzu werden eine horizontale und eine vertikale Schnittebene, an den Stellen der Strahlungsmaxima, durch die dreidimensionale Antennencharakteristik gelegt (siehe Abbildung 11).

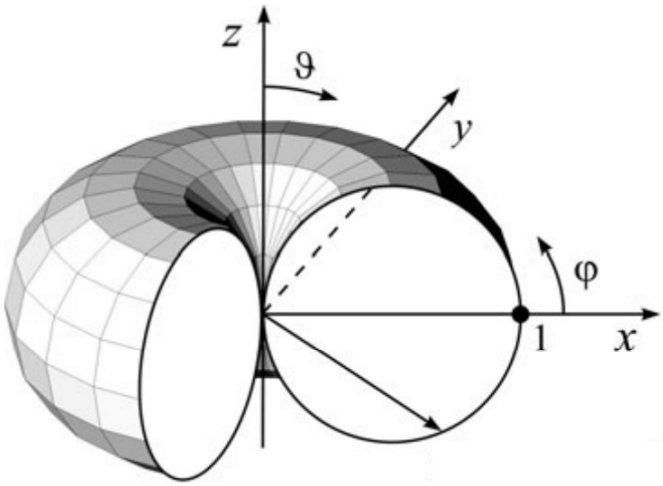



Abbildung 11: Vertikaler Schnitt durch das dreidimensionale Antennendiagramm eines Dipols (verändert nach Kark 2018, S. 249).

Die daraus entstehenden Schnitte werden häufig in Form von Polardiagrammen (siehe Abbildung 12) dargestellt. Diese geben die Strahlungsverteilung, relativ zum Maximalwert der Hauptstrahlrichtung, in linearem oder logarithmischem Maßstab an (Kark 2018, S. 247-251).

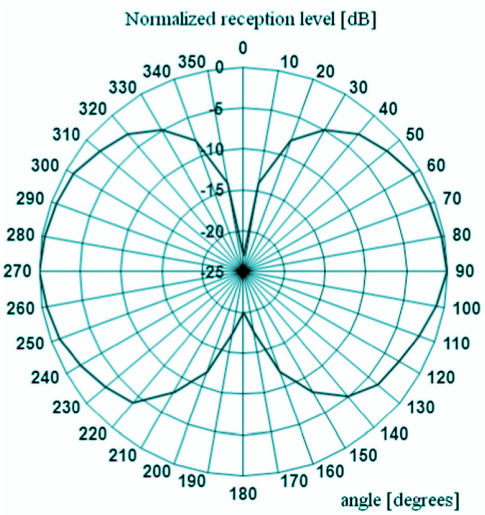



Abbildung 12: Logarithmisches Antennendiagramm in der Vertikalebene (Dolea et al. 2015)

### Effektive Sendeleistung und Empfangsleistung

Die *Equivalent* (oder Effective) *Isotropic Radiated Power* (EIRP) gibt die tatsächlich von der Antenne abgegebene Sendeleistung in dBm an. Die logarithmische Maßeinheit *dBm* bezieht sich hierbei auf 1 mW, was einem Äquivalent von 0 dBm entspricht. Definiert ist der EIRP als die Leistung, die eine isotrop strahlende Antenne abgäbe, würde diese die gleiche Leistung wie in Richtung des maximalen Antennengewinns abstrahlen (Vazquez 2013, S. 11-12). Die Berechnung des EIRP erfolgt nach (Rech 2006, S. 271) unter Verwendung folgender Formel

$$EIRP = P_s - L_k + G_s \tag{6}$$

wobei,

| | |
|---|---|
| $EIRP$ | die äquivalent isotrop abgestrahlte Leistung in dBm, |
| $P_s$ | die Sendeleistung des Gerätes in dBm, |
| $L_k$ | die Dämpfung des Antennenkabels in dB und |
| $G_s$ | der Antennengewinn der Sendeantenne in dBi ist. |

Die maximal zulässige Sendeleistung ist regional festgelegt und darf in Deutschland, bei einer Sendefrequenz von 2,4 GHz, die EIRP von 20 dBm (100 mW) nicht überschreiten. Bei Frequenzen im 5 GHz-Band beträgt dieser Grenzwert 30 dBm (1000 mW) (Rech 2006, S. 271).

Die *Empfangsleistung* oder auch *Received Signal Level* (RSL) ist die, an der Empfangsantenne gemessene, Leistung (Empfangsleistung) und wird nach (Faruque 2014, S. 24) wie folgt berechnet

$$RSL = EIRP - L_p + G_r - L_c \tag{7}$$

wobei,

| | |
|---|---|
| $RSL$ | die Empfangsleistung in dBm, |
| $EIRP$ | die equivalent isotrop abgestrahlte Leistung in dBm, |
| $L_p$ | der Pfadverlust in dB, |
| $G_r$ | der Antennengewinn der Empfangsantenne in dBi und |
| $L_c$ | die Dämpfung der Steckverbindungen der Antennen in dB ist. |

## 1.3 Funkausbreitungsmodelle

Nach (Viol et al. 2012) werden Modelle zur Berechnung der Funkabdeckung grundsätzlich in empirische, semi-empirische und physikalische Modelle unterschieden. In (Fernández 2015, S. 23) wird diese Unterteilung um die theoretischen Modelle ergänzt. Hierbei

beziehen die unterschiedlichen Modellarten die Umgebungsgeometrie verschieden stark in die Berechnung der Funkausbreitung ein. Im Folgenden werden die oben genannten Arten der Funkausbreitungsmodelle näher erläutert.

### 1.3.1 Theoretische Modelle

Besteht eine Sichtverbindung zwischen Sender und Empfänger kann die Dämpfung des Funksignals (Freiraumdämpfung), respektive die Empfangsleistung, nach dem *Freiraummodell* (engl. Free Space Model) ermittelt werden. Hierzu wird angenommen, dass sich die abgestrahlte Energie mit gleicher Intensität omnidirektional ausbreitet. Die Berechnung der Freiraumdämpfung erfolgt nach folgender Formel (Rech 2006, S. 273)

$$FSL = 20\ log\left(\frac{4\ \pi\ d}{\lambda}\right) \tag{8}$$

wobei,

| | |
|---|---|
| $FSL$ | die Freiraumdämpfung in dB, |
| $d$ | die Distanz zwischen Sender und Empfänger in Metern und |
| $\lambda$ | die Wellenlänge in Metern ist. |

Fernández 2015, S. 24 gibt für das *Freiraummodell* eine, um den Antennengewinn der Sende- und Empfangsantenne, erweiterte Formel an. Während in 8 die Dämpfung zwischen Sender und Empfänger berechnet, wird in 9 die Empfangsleistung, in Abhängigkeit zur Distanz $d$ ermittelt. Hierzu wird die Sendeleistung der Antenne $P_t$ mit einbezogen.

$$P_r(d) = \frac{P_t\ G_t\ G_r\ \lambda^2}{(4\ \pi)^2\ d^2\ L} \tag{9}$$

Mit,

| | |
|---|---|
| $P_r(d)$ | als Empfangsstärke in Abhängigkeit der Entfernung $d$ gemessen in dBm, |
| $P_t$ | als Sendeleistung in dBm, |
| $G_t$ | als Antennengewinn des Senders in dBi, |
| $G_r$ | als Antennengewinn des Empfängers in dBi und |
| $L$ | als Ausbreitungsunabhängige Dämpfungen in dB. |

In Tabelle 8 sind Freiraumdämpfungen für verschiedene Entfernungen im 2,4 GHz- und im 5 GHz-Band angegeben.

Tabelle 3: Freiraumdämpfung für verschiedene Distanzen (Rech 2006, S. 274).

| Distanz [m] | FSL 2,4 GHz-Band [dB] | FSL 5 GHz-Band [dB] |
|---|---|---|
| 1 | 40,2 | 47,16 |
| 10 | 60,2 | 67,16 |
| 100 | 80,2 | 87,16 |
| 200 | 86,2 | 93,18 |
| 400 | 92.24 | 99,2 |
| 1000 | 100,2 | 107,16 |

Weitere theoretische Modelle wie das *Diffracting-Screens-Model*, treffen vereinfachte Annahmen durch die Generalisierung der Umgebungsgeometrie. So bildet dieses Modell die Geometrie von Hausreihen auf eine geringe Anzahl von Parametern ab. In die Berechnung der Signaldämpfung fließen hier beispielsweise die Höhe der Gebäude, die Straßenbreite oder die Höhe der Sende- und Empfangsantenne ein (Fernández 2015, S. 25).

### 1.3.2 Empirische Modelle

*Empirische Modelle* beschreiben die Signaldämpfung, zwischen Sender und Empfänger, mit Hilfe empirisch ermittelter Parameter. Diese werden unter Verwendung von Ausgleichungsverfahren aus Messdatenreihen bestimmt (Chaabane 2005, S. 20). Auf Grundlage dieser Parameter werden erweiterte *Freiraummodelle* erstellt, die in Umgebungen eingesetzt werden können, welche den Untersuchungsgebieten der Datenerfassung ähneln (Hussain 2017, S. 48).

Ein Beispiel hierfür bietet das *One-Slope-Model*, welches auf dem *Freiraummodell* basiert. Die Berechnung der Empfangsstärke hängt hierbei, wie beim *Freiraummodell*, von der Distanz zwischen Sender und Empfänger ab. Zusätzlich wird die Signaldämpfung, durch die Umgebung, mit Hilfe des Dämpfungsfaktors $\gamma$ betrachtet. Das Modell wird in (Retscher u. Tatschl 2017) wie folgt beschrieben

$$P(d) = P_0 + 10\ \gamma\ log(d) \tag{10}$$

wobei,

$P_d$ die Signalempfangsstärke in dBm, abhängig von der Entfernung $d$ in m,
$P_0$ die Referenzsignalstärke in einem Meter Entfernung in dBm und
$\gamma$ der Dämpfungsfaktor ist.

In Tabelle 5 werden für $\gamma$ und $P_0$ empirisch ermittelte Werte für verschiedene Indoor-Szenarien im 2,4 und 5 GHz Band angegeben. Aufgrund der Variabilität der Dämpfungsfaktoren kann es jedoch zu sehr großen Fehlern kommen (Retscher u. Tatschl 2017).

Tabelle 4: Parameter des One-Slope-Modells (Retscher u. Tatschl 2017).

| | Frequenz [GHz] | $P_0$ [dB] | $\gamma$ |
|---|---|---|---|
| Bürogebäude | 2,45 | 40,2 | 4,2 |
| Korridore | 2,45 | 40,2 | 1,2 |
| Bürogebäude | 5,25 | 46,8 | 4,6 |

### 1.3.3 Semi-empirische Modelle

Die *semi-empirischen Modelle* erweitern die rein *empirischen Modelle* um den Einbezug der Umgebung des zu untersuchenden Gebietes. Das *Multi-Wall-Model* soll hierbei als Beispiel dienen. Dieses, für den Indoor-Bereich entwickelte Modell, erzielt in einfach aufgebauten Umgebungen bessere Ergebnisse als *empirische Modelle*. Es betrachtet die Dämpfungseigenschaften $D_i$ der Wände, welche das Signal durchläuft. Hierbei wird der direkte Weg des Senders zum Empfänger betrachtet. Abbildung 14 stellt solch ein Szenario schematisch dar.

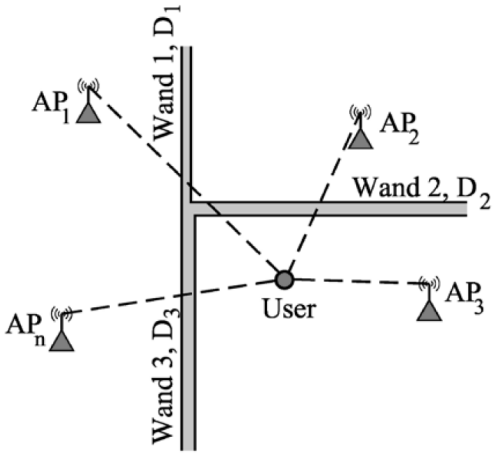



Abbildung 13: Darstellung des Multi-Wall-Modells (Retscher u. Tatschl 2017).

Ungenauigkeiten treten hierbei durch Kleinstrukturen (zum Beispiel schmale Gänge oder Säulen), unterschiedliche Materialien in den Wänden oder durch, in der Stärke variierende, Wände auf (Retscher u. Tatschl 2017). Die, durch die Dämpfung der Wände erweiterte, Empfangsstärkenberechnung erfolgt nach (Retscher u. Tatschl 2017) wie folgt

$$P(d) = P_0 + 10\ \gamma\ log(d) + \sum_{i=1}^{n} D_i \tag{11}$$

wobei,

$P_d$ die Signalempfangsstärke in dBm, Abhängig von der Entfernung $d$ in m,

$P_0$ die Referenzsignalstärke in einem Meter Entfernung in dBm,

$\gamma$ der Dämpfungsfaktor und
$\sum_{i=1}^{n} D_i$ die Summe der Signaldämpfungen durch $n$ Wände in dB ist

Eine Übersicht verschiedener Dämpfungswerte für unterschiedliche Baumaterialien ist in Tabelle 5 gegeben. Bei metallischen Flächen respektive Wänden findet, in einem Sendefrequenzbereich von 30 MHz bis 10 GHz, nahezu keine Durchdringung des Materials statt, was bei ausreichender Wandstärke eine absolute Reflexion der elektromagnetischen Welle zur Folge hat (BSI TR-03209 - 1 2008, S. 15 u. 16).

Tabelle 5: Dämpfungswerte verschiedener Materialien im 2,4 GHz Bereich (Retscher u. Tatschl 2017).

| Material | Dämpfungen $D_i$ [dB] |
|---|---|
| dünne Mauer | 2 - 5 |
| Ziegelwand | 6 - 12 |
| Betonwand | 10 - 20 |
| Doppelverglasung | 25 - 35 |
| Betondecke | 20 - 40 |

### 1.3.4 Physikalische Modelle

In *Physikalischen Modellen* (auch Strahlenoptische Modelle) wird angenommen, dass sich die Ausbreitung hochfrequenter elektromagnetischer Wellen durch die Ausbreitung von Strahlen annähern lässt. Voraussetzung ist, dass die Wellenlänge bedeutend kleiner ist, als die in der Umgebung befindlichen Objekte. Die Grundlage, für die Approximation elektromagnetischer Wellen durch Strahlen, bildet die *Geometrische Optik* (GO). Die klassische GO vernachlässigt hierbei die Welleneigenschaften elektromagnetischer Strahlung. Dies bedeutet, dass die Phasenlage sowie die Polarisation einer Welle nicht in Betracht gezogen und lediglich die Amplitude sowie die Ausbreitungsrichtung modelliert wird. Die klassische GO kann somit zur Modellierung der Ausbreitung reflektierter und gebrochener Strahlen genutzt werden (siehe Abbildung 12) (Bachhuber 2011, S. 95-96).

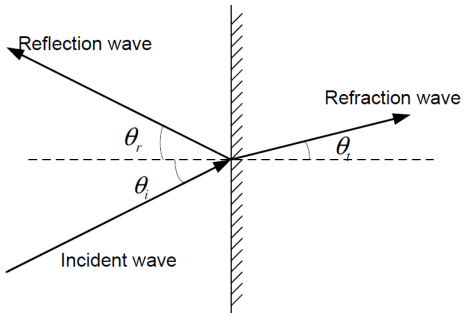



Abbildung 14: Reflexion und Brechung (verändert nach Luo 2013, S.12).

Nach (Luo 2013, S. 11-12) gilt hierbei das Brechungsgesetz von Snellius

$$\theta_r = \theta_i$$
$$\frac{sin(\theta_t)}{sin(\theta_i)} = \frac{n_1}{n_2} \tag{12}$$

wobei,

| | |
|---|---|
| $\theta_i$ | der Eintrittswinkel, |
| $\theta_r$ | der Austrittswinkel, |
| $\theta_t$ | der Brechungswinkel und |
| $n_1$ und $n_2$ | die Brechungszahlen der unterschiedlichen Medien sind. |

Die moderne GO erweitert die Eigenschaften eines Strahls um die Phase und Polarisation. Hierzu wird die *Geometrische-* und die *Einheitliche-Beugungstheorie* (Geometrical / Uniform Theory of Diffraction) verwendet. Diese beschreiben Reflexions-, Transmissions- und Beugungseffekte eines Strahls mit den Welleneigenschaften Phase und Polarisation, für die Interaktion mit *kanonischen Objekten*. Für diese einfachen geometrischen Körper sind Ausbreitungseffekte, wie Reflexion, Transmission und Beugung, parametrisiert lösbar (Bachhuber 2011, S. 96-98).

Da die konkreten Umgebungseigenschaften in die Berechnung der Funkausbreitung mit einfließen, werden die Art von Modellen auch als *Site-Specific*, also standortspezifisch bezeichnet (Chaabane 2005, S. 22). Eine umfangreiche Beschreibung der modernen GO ist in (Geng u. Wiesbeck 1998; Kouyoumjian u. Pathak 1974 und Keller 1962) zu finden. Folgend wird das *Path Launching Model*, als Vertreter der *Physikalischen Modelle*, genauer betrachtet. In der vorliegenden Arbeit wird dieses Modell zur Berechnung der Funkausbreitung genutzt.

### Ray Launching Model

Beim *Ray Launching Model* werden Strahlen von einem Sender aus, mit einer anfänglich definierten Winkelauflösung ($\Delta\varphi$), in die entsprechenden Raumrichtungen ausgesendet (siehe Abbildung 16). Für die Empfangsstärkeberechnung in einem Detektorpixel wird insbesondere die *Freiraumdämpfung* des Strahlenwegs $d$ berücksichtigt (Preiner et al. 2006, S. 31).

Trifft ein Strahl auf ein Objekt, werden an dem Interaktionspunkt (Schnittpunkt mit der Umgebungsgeometrie) Ausbreitungseffekte wie Reflexion und Brechung berechnet. Vom Interaktionspunkt aus werden neue Strahlen erzeugt, welche den reflektierten und gebrochenen Strahl darstellen. Dieser Vorgang wird für die neu erzeugten Strahlen rekursiv wiederholt. Ein Abbruch der Rekursion erfolgt durch die Definition von Abbruchbedingungen. Diese Bedingungen können das Unterschreiten einer bestimmten Empfangsleistung oder eine maximale Anzahl von Interaktionen mit der Umgebung sein.

Zur Simulation von Beugungseffekten an Kanten muss diesen ein Toleranzbereich zugewiesen werden, da das direkte Auftreffen eines Strahles auf eine Kante unwahrscheinlich ist. Trifft ein Strahl auf eine Kante entsteht ein Beugungskegel (siehe Abbildung 15). Dieser erzeugt eine diskrete Anzahl weiterer Strahlen. Der Pfadverlauf zwischen Sender und Empfänger wird durch sogenannte *Detektorflächen* (auch Detektorpixel) nachvollzogen. Anhand des ermittelten Signallaufweges erfolgt die Berechnung der Empfangsleistung an den Positionen der Detektorflächen (Bachhuber 2011, S. 110-111).

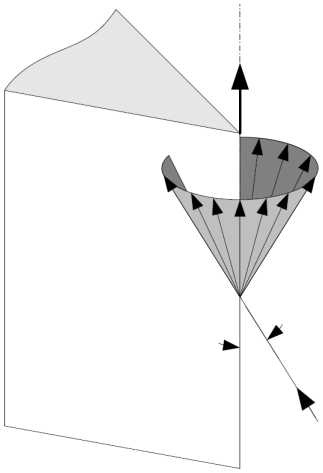


Abbildung 15: Beugungskegel an einer Kante (verändert nach Bachhuber 2011, S. 106).

Nachteilig an diesem Modell ist, dass die ausgesendeten Strahlen in großer Entfernung stark divergieren. Hier kann es bei einer zu niedrigen Winkelauflösung vorkommen, dass Detektorpixel von dem Strahl nicht durchlaufen werden (siehe Abbildung 16). An diesen Stellen ist dann kein Empfangswert ermittelbar. Andere Detektorpixel hingegen werden unter Umständen mehrmals durchlaufen (Bachhuber 2011, S. 111).

Ein Vorteil dieser Methode ist die Berechnung der Empfangsleistung für ein gesamtes Gebiet und nicht nur für einen einzelnen Standort. Die Sendestandorte der Antennen müssen dabei bekannt und fix sein (Preiner et al. 2006, S. 31).

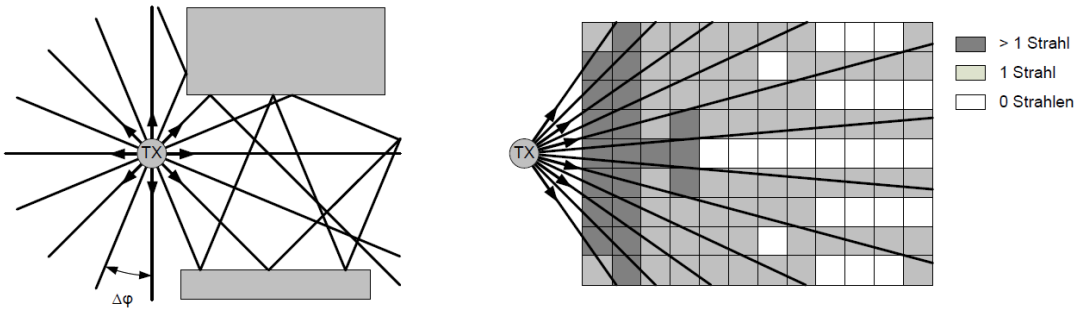



Abbildung 16: Aussenden von Strahlen mit der Winkelauflösung $\Delta\varphi$ (links). Werteabdeckung der Detektorfläche bei divergierenden Strahlen (rechts) (Bachhuber 2011, S. 111).

# 2 Geostatistische Grundlagen

Dieses Kapitel beschäftigt sich mit den, für die Arbeit relevanten, geostatistischen Grundlagen. Hiezu erfolgt eine Beschreibung des Schätz- und Interpolationsverfahren *Kriging*. Dabei wird zunächst das *Variogramm* erläutert, da dieses im *Krigingverfahren* zur Modellierung der räumlichen Variabilität genutzt wird und auch bei dem, später in diesem Kapitel, betrachteten *geostatistischen Simulationsverfahren* zum Tragen kommt. Abschließend erfolgt eine Beschreibung der, in dieser Arbeit verwendeten, Simulationssoftware, welche zur Durchführung der *geostatistischen Simulation* genutzt wird.

## 2.1 Kriging

Das *Krigeverfahren* (kurz *Kriging*) bezeichnet einen Schätzvorgang in der Geostatistik, bei dem die Schätzvarianz eines vorherzusagenden Wertes $Z^*(x_0)$ unter einer gegeben Anzahl von Proben minimiert wird. Das *Kriging* umfasst dabei eine Reihe von Schätzverfahren. So wird bei einem bekannten und konstanten Erwartungswert von *simple Kriging*, bei unbekanntem und lokal schwankendem Erwartungswert von *ordinary Kriging* gesprochen. Das *universal Kriging* nimmt einen Trend im Erwartungswert an und modelliert diesen (Akin u. Siemes 1988, S. 112).

### Variogramm

Das *Variogramm* $2\gamma(h)$ oder auch *Semivariogramm* $\gamma(h)$[2] dient, in der Geostatistik zur Beschreibung der räumlichen Variabilität einer Messgröße und wird im *Krigeverfahren*, zur Berechnung der Wichtungsfaktoren $\lambda_i$ genutzt. Hierzu wird das *Variogramm* zunächst durch ein *experimentelles Variogramm* $\gamma^*(h)$ abgeschätzt (Schafmeister 1999, S. 14-15). Dieses stellt die halben gemittelten quadrierten Differenzen der gemessenen Proben mit gleichen Schrittweiten $h$ (*Lag*) dar und wird wie folgt berechnet (Akin u. Siemes 1988, S. 31 u. 112)

$$\gamma^*(h) = \frac{1}{2\ n(h)} \sum_{i=1}^{n(h)} \{[z(x_i + h) - z(x_i)]^2\} \tag{13}$$

wobei,

| | |
|---|---|
| $\gamma^*(h)$ | der experimentelle Variogrammwert der Schrittweite $h$, |
| $n(h)$ | die Anzahl der Wertepaare für die Schrittweite $h$, |
| $z(x_i)$ | der Wert der gemessenen Probe an der Stelle $x_i$ ist. |

[2] Im Rahmen der Arbeit werden die Begriffe *Variogramm* und *Semivariogramm* synonym verwendet. Die Unterscheidung wird anhand von $2\gamma(h)$ oder $\gamma(h)$ ersichtlich.

Hierbei wird die *intrinsische Hypothese* angenommen. Diese besagt, dass der Variogrammwert $\gamma^*(h)$ nur von der Differenz der Position $x_i + h - x_i$ abhängt, nicht aber von der Position $x_i$ selbst (Dutter 1985, S. 19).

Im Zuge des *Krigeverfahrens* wird das *experimentelle Variogramm* durch ein parametrisierbares *Variogrammmodell* $\gamma(h)$ ersetzt (siehe Abbildung 17), das mit Hilfe der Reichweite $a$ (*Range*) und dem Grenzwert $C$ (*Sill*) beschrieben wird. Nimmt das *experimentelle Variogramm* im Ursprung nicht den Wert 0 an, muss außerdem der *Nugget-Effekt* $C_0$, als weiterer Parameter, einbezogen werden. Dieser ist ein Hinweis auf Mess- oder Analysefehler sowie Anzeichen für eine hohe Variabilität, bezogen auf eine kleinräumige Ausdehnung (Schafmeister 1999, S. 15-16).

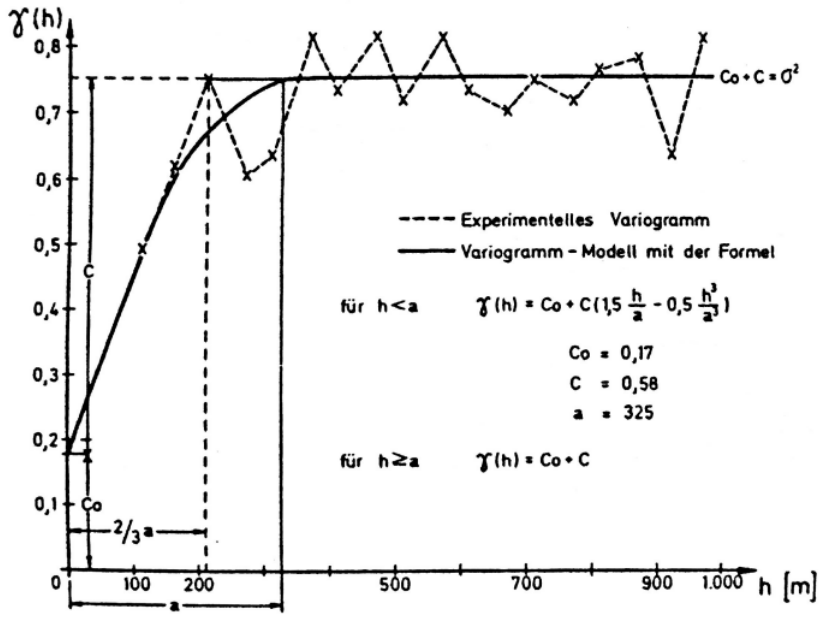



Abbildung 17: Anpassung eines Variogrammmodells durch die aus dem experimentellen Variogramm ermittelten Parameter $C_0$, $a$ und $C$. (Akin u. Siemes 1988, S. 48).

Im Folgenden werden gebräuchliche *Variogrammmodelle* formelmäßig nach (Akin u. Siemes 1988, S. 44 u. 46) beschrieben. Eine grafische Darstellung der Modelle ist Abbildung 18 zu entnehmen.

**Sphärisches Modell**

$$\gamma(h) = \begin{cases} C\left(\frac{3}{2}\,\frac{|h|}{a} - \frac{|h|^3}{2a^3}\right) & \text{für h} \leq \text{a} \\ C & \text{für h} > \text{a} \end{cases} \tag{14}$$

**Exponentielles Modell**

$$\gamma(h) = C\left[1 - \exp\left(\frac{-|h|}{a}\right)\right] \tag{15}$$

**Gauß'sches Modell**

$$\gamma(h) = C\left[1 - \exp\left(\frac{-h^2}{a^2}\right)\right] \tag{16}$$

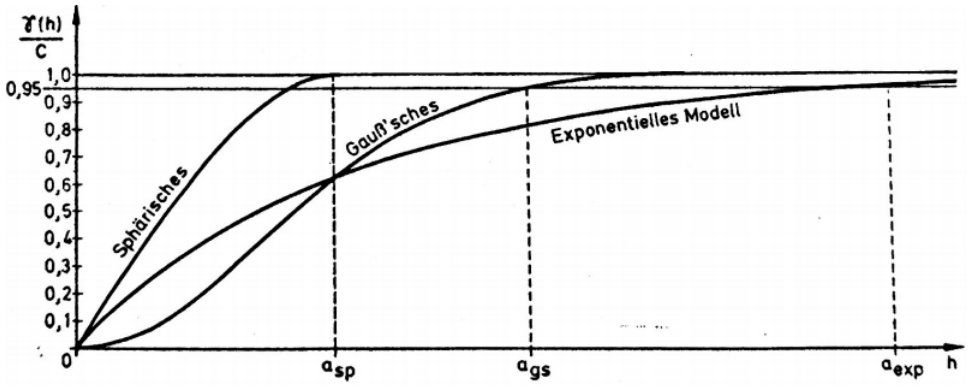



Abbildung 18: Sphärisches, Gauß'sches sowie Exponentielles *Variogrammmodell* (Akin u. Siemes 1988, S. 45).

**Ordinary Kriging**

Beim *OK* ergibt sich der Schätzwert $Z^*(x_0)$ an der Stelle $x_0$ aus der Linearkombination der Gewichte $\lambda_i$, mit den Werten an den bekannten Stellen $Z(x_i)$. Die Berechnung des Schätzwerts erfolgt nach (Akin u. Siemes 1988, S. 120) durch folgende Formel

$$Z^*(x_0) = \sum_{i=1}^{n} \lambda_i Z(x_i) \tag{17}$$

wobei,

| | |
|---|---|
| $Z^*(x_0)$ | der Schätzwert an der Position $x_0$, |
| $Z(x_i)$ | der Wert an der bekannten Stelle $i$ und |
| $\lambda_i$ | der Wichtungsfaktor $i$ für den Wert an der Stelle $Z(x_i)$ ist. |

Die Wichtungsfaktoren $\lambda_i$ werden nach der *Lagrange-Methode* so berechnet, dass die Varianz der Schätzung $\sigma^2$ (auch *Krigevarianz*) minimiert wird. Weiter wird als Nebenbedingung gefordert, dass die Summe der Gewichte 1 beträgt. Durch die Minimierung der Schätzvarianz $\sigma^2$ entsteht folgendes, um den *Lagrange-Multiplikator* $\mu$ erweitertes, Gleichungssystem mit $n + 1$ unbekannten (*Krige-System*). Durch die Einführung des Multiplikators wird die oben genannte Nebenbedingung erfüllt. Die Lösung des *Krige-Systems* ermöglicht die Berechnung der Unbekannten $\lambda_i$ und $\mu$, was die Ermittlung des Schätzwertes $Z^*(x_0)$ sowie der *Krigevarianz* $\sigma^2$ erlaubt (Akin u. Siemes 1988, S. 116-121).

$$\begin{bmatrix} \gamma(x_1x_1) & \gamma(x_1x_2) & \dots & \gamma(x_1x_n) & 1 \\ \gamma(x_2x_1) & \gamma(x_2x_2) & \dots & \gamma(x_2x_n) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(x_nx_1) & \gamma(x_nx_2) & \dots & \gamma(x_nx_n) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ \mu \end{bmatrix} = \begin{bmatrix} \gamma(x_1x_0) \\ \gamma(x_2x_0) \\ \vdots \\ \gamma(x_nx_0) \\ 1 \end{bmatrix} \tag{18}$$

Mit,

| | |
|---|---|
| $\gamma(x_ix_j)$ | als Variogrammwert der Distanz $x_ix_j$, |
| $\lambda_i$ | als Wichtungsfaktoren und |
| $\mu$ | als Lagrange-Multiplikator. |

Die Berechnung der Schätzvarianz erfolgt nach (Akin u. Siemes 1988, S. 120) unter Verwendung folgender Formel

$$\sigma^2(x_0) = \mu + \sum_{i=1}^{n} \lambda_i \gamma(x_ix_0) \tag{19}$$

wobei,

| | |
|---|---|
| $\sigma^2$ | die Varianz der Schätzung, |
| $\mu$ | der Lagrange-Multiplikator, |
| $\gamma(x_ix_0)$ | der Variogrammwert der Strecke $x_ix_0$ und |
| $\lambda_i$ | der Wichtungsfaktor $i$ für den Wert an der Stelle $Z(x_i)$ ist. |

## 2.2 Geostatistische Simulation

Im Gegensatz zum *Krigeverfahren*, ermitteln *geostatistische Simulationsverfahren*[3] nicht die besten Schätzwerte $Z^*(x_i)$ an den Stellen $x_i$, sondern jene Werte, die die räumliche Variabilität möglichst genau wiedergeben. Das Ziel der Simulation ist es somit, dass die Simulationswerte die räumliche Charakteristik, des *Variogramms* der Ausgangsdaten, möglichst genau abbildet. Dies ist bei Interpolationsverfahren wie dem *Kriging* nicht der Fall, da hier eine Glättung der Schätzwerte erfolgt und das *Variogramm* der interpolierten Daten, eine zu geringe Varianz (*Sill*) aufweist (Blöschl 2006).

Abbildung 19 zeigt das Ergebnis einer für eine Interpolation mit dem *Krigeverfahren* sowie einer Simulation, unter Verwendung des selben *Variogrammmodells*. Das, aus den Simulationsdaten erzeugten, *experimentelle Variogramm* gibt den Verlauf des *Variogrammmodells* genauer wieder, als das *experimentelle Variogramm* der *Krigewerte*. Dies führt zu einer realistischeren räumlichen Variabilität der erzeugten Daten (Goovaerts 1999).

[3] Im Weiteren auch kurz als *Simulation* bezeichnet.

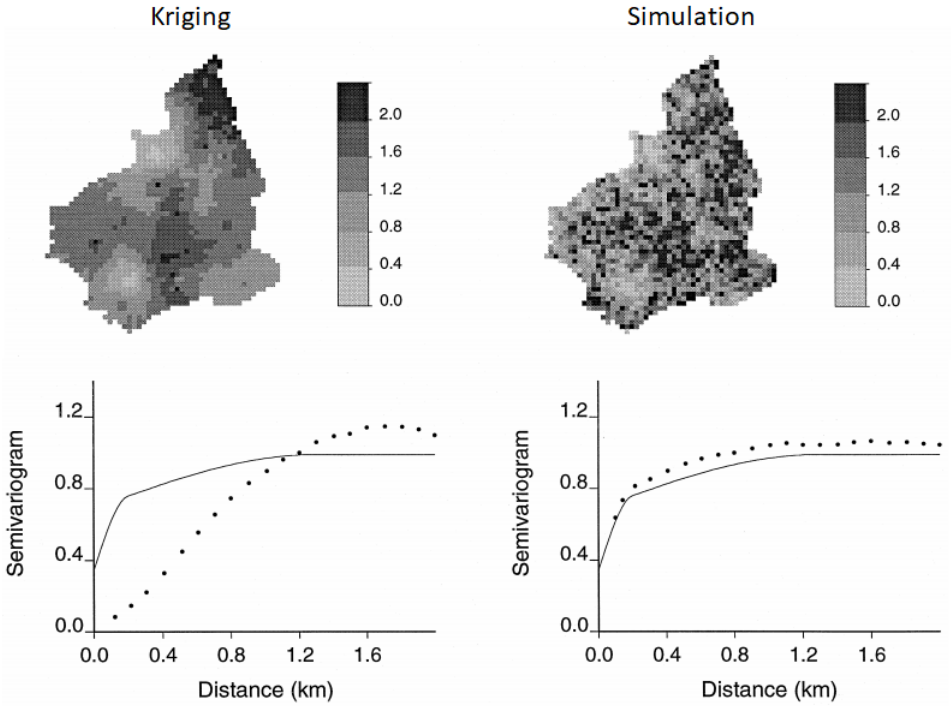



Abbildung 19: Gegenüberstellung der Ergebnisse einer Interpolation mit dem *Krigeverfahren* (links) und einer *geostatistischen Simulation* (rechts). In den *Variogrammen* wird das *Variogrammmodell* als durchgezogene und das *experimentelle Variogramm* (aus den erzeugten Daten) als gepunktete Linie dargestellt (verändert nach Goovaerts 1999).

Das Ergebnis einer *Simulation* wird als *Realisation* bezeichnet und stellt ein alternatives Resultat $z(x)$ in einem räumlichen Zufallsprozess $Z(x)$ dar. Durch eine sogenannte Konditionierung (siehe Abbildung 20) lassen sich die Werte an den Stützstellen in einer *Realisation* erhalten (Gau 2010, S. 25).

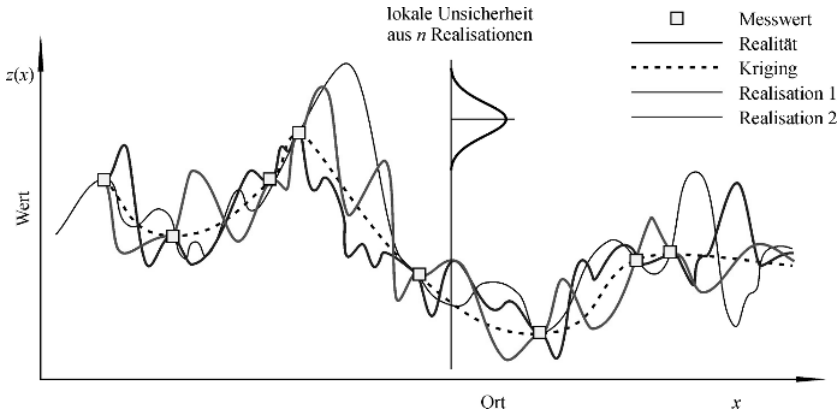



Abbildung 20: Darstellung einer *konditionierten geostatistischen Simulation* mit den *Realisationen* 1 und 2 sowie des *Krigeverfahrens* (Gau 2010, S. 26).

Die häufigst verwendete konditionierte Simulation ist nach (Hengl 2007, S. 49) die *Sequentielle Gaußsche Simulation* (SGS), welche in (Deutsch 2002, S. 164-165) durch folgende Schritte algorithmisch zusammengefasst wird.

1. Transformation der Originaldaten $Z$ in die Standardnormalverteilung.
2. Berechnung des *Schätzwertes* $Z^*(x_0)$ und der *Krigevarianz* $\sigma^2(x_0)$ an der Stelle $x_0$ (siehe Formel 5 und 7).
   Soll das *Variogramm* exakt durch die simulierten Daten abgebildet werden, wird an dieser Stelle das *Simple Kriging* verwendet. Durch die Verwendung des *OK* in diesem Prozessschritt, wird das vorgegebene *Variogramm* weniger exakt durch die Simulation reproduziert, wobei jedoch der lokale Mittelwert stärker einbezogen wird (Pebesma 2014, S. 20). Die Berechnung von $Z^*(x_0)$ und $\sigma^2(x_0)$ im Rahmen des *Simple Kriging*, kann (Deutsch 2002, S.157 - 160) entnommen werden.
3. Erzeugen einer zufälligen Störgröße (Residuum) $R(x_0)$, welcher der Normalverteilung folgt und einen Mittelwert von 0 sowie die Varianz $\sigma^2(x_0)$ aufweist.
4. Das Residuum $R(x_0)$ wird auf den Schätzwert $Z^*(x_0)$ addiert, um so den Simulationswert $Z_s(x_0)$ zu erhalten.

$$Z_s(x_0) = Z^*(x_0) + R(x_0) \tag{20}$$

5. Dem Datensatz $Z$ wird der Simulationswert $Z_s(x_0)$ hinzugefügt.
6. Schritte 2 bis 5 werden für alle zu simulierenden Stellen $x_0$ wiederholt. Der Abarbeitungspfad der Stellen $x_0$ wird zufällig gewählt.
7. Rücktransformation des Datensatzes $Z$.

Soll mehr als eine Realisation $z(x)$ erzeugt werden, wird dies durch unterschiedliche *Seeds* im Rahmen der Residienerzeugung (Schritt 3) sowie der Pfadgenerierung (Schritt 6) realisiert (Deutsch 2002, S. 165).

## 2.3 Simulationssoftware

Der zuvor beschriebene *SGS*-Algorithmus ist in diversen Programmen und Programmbibliotheken für geostatistische Berechnungen implementiert (eine Übersicht gängiger Simulationssoftware ist Hengl 2007, S. 79 zu entnehmen). Hierzu zählt unter anderem die Softwareumgebung und Programmiersprache *R*, welche zur Durchführung statistischer Berechnungen dient. Zur Einbindung geostatistischer Simulationen greift *R* auf das *gstat*-Paket zurück. Dieses wird als Erweiterung in die *R*-Umgebung eingebunden und stellt zahlreiche geostatistische Funktionen bereit. Hierzu zählen unter anderem die Berechnung und Darstellung von *Variogrammen* oder die Durchführung verschiedener *Krigeverfahren* sowie *konditionierter Simulationen* (Hengl 2007, S. 55 u. 67).

Die Ausführung einer *Simulation* erfolgt in *R* durch die *krige*-Funktion des *gstat*-Paktes. Der Funktion wird dabei eine Reihe von Parametern übergeben. Diese sind zum einen der funktionale Zusammenhang des Regressionsmodells (Listing IV, Parameter: *function*) sowie die für die *Simulation* verfügbaren Daten (Listing IV, Parameter: *data*). Der für die *Simulation* verwendete Datenrahmen respektive das Datenraster wird ebenfalls definiert (Listing IV, Parameter: *grid*). Weiter wird das *Variogrammmodell* mit der Funktion *vgm* erzeugt und der *krige*-Funktion übergeben (Listing IV, Parameter: *model*). Die Anpassung des *Variogrammmodells* erfolgt durch die Übergabe des *Sills*, *Nuggets* und der *Range* an die *vgm*-Funktion. Die Anzahl der zu erzeugenden *Simulationen* und der Maximalabstand (Listing IV, Parameter: *maxdist*) sowie die Maximalanzahl (Listing IV, Parameter: *namx*) der, in die Berechnung der Wichtungsfaktoren einzubeziehenden Datenpunkte, sind ebenfalls definierbar (Hengl 2007, S. 31-32).

Listing 1: Beispiel für die Ausführung der *krige*-Funktion in *R* (verändert nach Hengl 2007, S. 31-32).

```
krige(function, data, grid, model = vgm(psill=1, "Gau", range=.2, nugget=0), nsim,
      maxdist, nmax)
```

Abbildung 21 zeigt ein Beispiel verschiedener Realisationen $z(x)$ des selben Zufallsprozesses $Z(x)$. Diese sind durch die *krige*-Funktion, unter Verwendung der *SGS* erzeugt worden und treten dabei alle mit gleicher Wahrscheinlichkeit auf (Hengl 2007, S. 117).

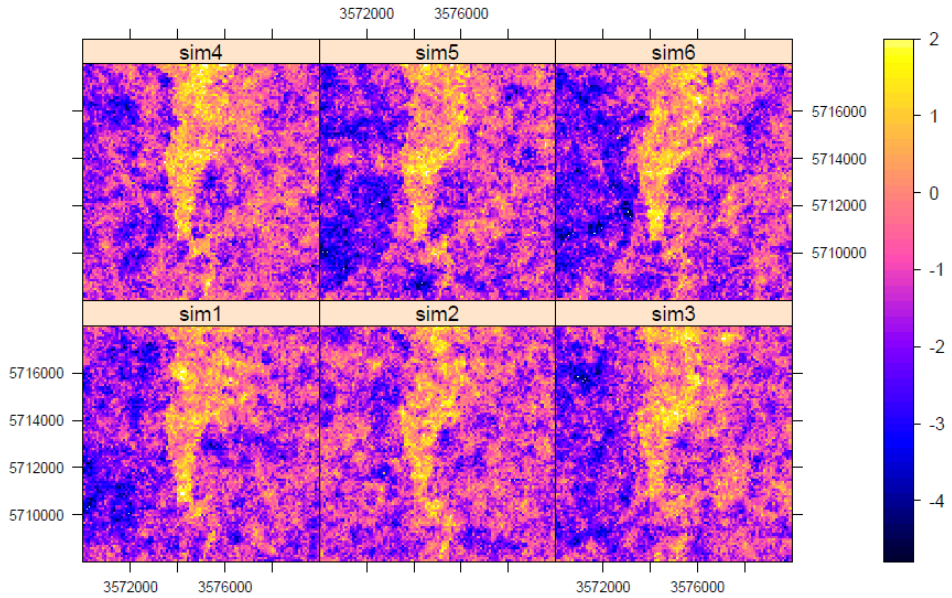



Abbildung 21: Sechs Realisationen, erzeugt mit der *krige*-Funktion in *R* (Hengl 2007, S. 117).

# 3 Unity 3D

*Unity 3D* ist eine von *Unity Technologies* entwickelte cross-platform Game Engine für 2D- und 3D-Spielanwendungen. Diese können sowohl auf Computern als auch auf Konsolen oder mobilen Endgeräten ausgeführt werden (Buyuksaliha et al. 2017). In (Christopoulou u. Xinogalos 2017) erfolgt ein Vergleich gängiger Game Engines hinsichtlich verschiedener Kategorien wie dem generellen Funktionsumfang, der Zugänglichkeit oder der audiovisuellen Möglichkeiten. Hierbei wird gezeigt, dass *Unity 3D* und die *Unreal Engine 4*, in allen Aspekten über einen vergleichbar hohen Funktionsumfang verfügen. Aufgrund schon vorhandener Vorkenntnisse und der damit verbundenen geringeren Einarbeitungszeit, wird Unity 3D für die Umsetzung der Planungslösung verwendet. In diesem Kapitel werden die, für die Arbeit relevanten, Konzepte der Unity Game Engine erläutert.

## 3.1 Szenen

Der Spielinhalt einer in Unity erstellten Anwendung wird in sogenannten *Szenen* (Scenes) dargestellt. Diese enthalten die im Spiel befindlichen Objekte (z.B. 3D-Modelle) und können als abgeschlossenes Level eines Spiels betrachtet werden (Unity 2018l).

Die Positionierung der Spielobjekte innerhalb der Szene erfolgt durch ein linkshändiges globales kartesisches Koordinatensystem sowie durch beliebig viele lokale Koordinatensysteme. Ein lokales Koordinatensystem ist dabei an ein Spielobjekt gebunden und beschreibt durch seine Koordinaten die Position relativ zu dem Objekt (Unity 2018m).

Abbildung 22 zeigt ein Spielobjekt und dessen lokales Koordinatensystem innerhalb des globalen Koordinatensystems.

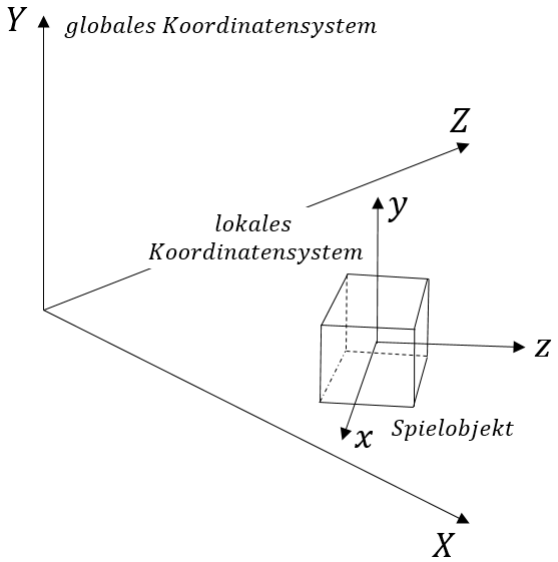



Abbildung 22: Darstellung des globalen Koordinatensystems einer Szene, mit einem darin befindlichen Spielobjekt und dessen lokalen Koordinatensystems.

Der Inhalt einer Szene wird mit Hilfe mindestens einer Kamera dargestellt. Diese definiert den Beobachtungsstandpunkt sowie den zweidimensionalen Bildausschnitt der dreidimensionalen Szene. Der darzustellende Bereich wird durch die nahe und ferne Clippingebene bestimmt. Der durch die Ebenen definierte Sichtkegel (engl. frustum) schließt alle auf dem Monitor darzustellenden Objekte innerhalb der Szene ein. Objekte Außerhalb dieses Kegels werden nicht dargestellt (siehe Abbildung 23). Der Bildausschnitt kann perspektivisch mit, oder orthografisch, ohne Tiefeneffekt dargestellt werden. Die orthografische Darstellung der Szene eignet sich beispielsweise zur Darstellung zweidimensionaler Karten.

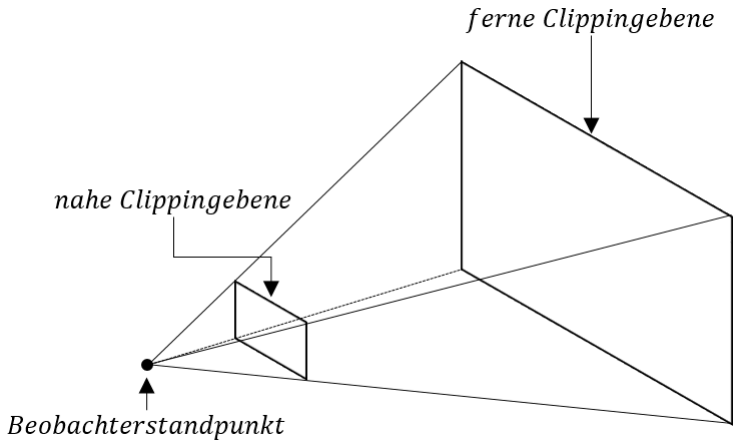



Abbildung 23: Clippingebenen der Kamera.

Das Koordinatensystem der Kamera ist lokal, wobei die Z-Achse in Blickrichtung der Kamera definiert ist (Unity 2018a).

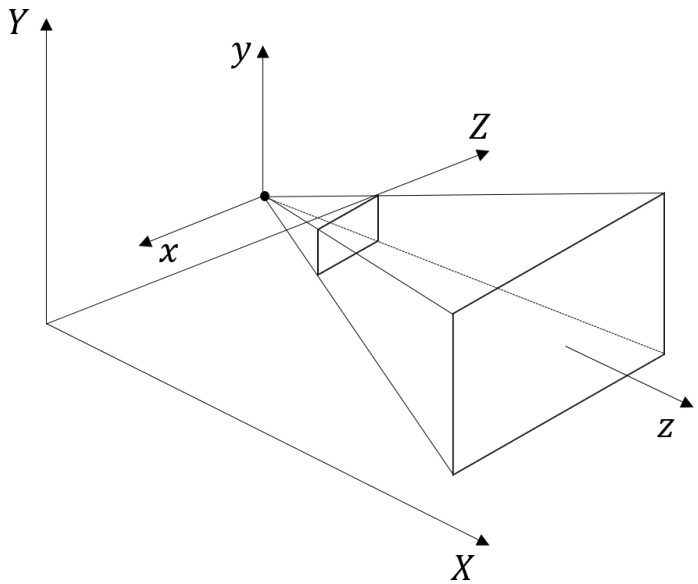



Abbildung 24: Lokales Koordinatensystem der Kamera.

## 3.2 Spielobjekte und Komponenten

Ein *Spielobjekt* (GameObject) ist das allgemeinste in einem Spiel vorkommende Objekt. *GameObjects* dienen als Container-Objekte für *Komponenten* (Components) und weitere Kindobjekte. Komponenten dienen wiederum zur Implementierung des Verhaltens und der Eigenschaften eines *GameObjects*. Auf sie werden die in der Anwendung verfügbaren Funktionalitäten abgebildet (Unity 2018d).

Components werden, in Form von Skripten (Scripts), an das entsprechende *Spielobjekt* angefügt. Nativ erfolgt die Programmierung der Skripte durch die Erstellung von Klassen in der Programmiersprache C#. Alternativ können die Sprachen *JavaScript* und *Boo* verwendet werden. Die Einbindung einer Klasse in die Ausführungsstruktur der Engine erfolgt durch die Vererbung der Elternklasse *MonoBehaviour*. Erbt ein Skript von dieser Elternklasse, stellt die Engine eine Reihe von Methoden zur Verfügung, welche sie an definierten Zeitpunkten zur Laufzeit ausführt. Listing 2 zeigt den grundlegenden Aufbau eines in C# verfassten Skriptes. Die Methode *Start* (siehe Listing 2, Zeile 7) wird vor dem Spielstart aufgerufen und dient zur Initialisierung einer Komponente bzw. der dazugehörigen Attribute. Die *Update*-Methode (siehe Listing 2, Zeile 12) wird periodisch vor jedem Frame aufgerufen und kann beispielsweise Funktionalitäten wie das Bewegen von Spielobjekten oder die Verarbeitung von Nutzereingaben enthalten (Unity 2018c). Eine vollständige Auflistung aller vererbten Methoden mit ihren Aufrufzeitpunkten ist dem Anhang A zu entnehmen.

Listing 2: C#-Skript der Komponente *MainPlayer* in Unity (Unity 2018c).

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainPlayer : MonoBehaviour {
5
6      void Start () {
7
8      }
9
10     void Update () {
11
12     }
13 }
```

Soll ein *Spielobjekt* mit seinen entsprechenden Komponenten und Eigenschaften mehrmals in einer Szene verwendet werden, kann die Speicherung des *GameObjects* als *Prefab* erfolgen. *Prefabs* sind vordefinierte wiederverwendbare Spielobjekte deren Instanziierung zur Laufzeit der Anwendung erfolgt. Wird ein *Prefab* instanziiert, werden diesem bzw. seinen Komponenten Eigenschaften zugewiesen. Diese Eigenschaften können beispielsweise die Position, Rotation und Skalierung des Spielobjekts in der Szene sein

(Unity 2018e). Im Folgenden sollen die für die Arbeit relevanten vordefinierten Komponenten vorgestellt werden, welche Teil des Unity Frameworks sind.

### 3.2.1 Transform

Die *Transform*-Komponente dient zur Speicherung der Position, Rotation und Skalierung eines Spielobjekts innerhalb einer Szene. Weiter wird die Position des Spielobjekts in der Spielobjekthierarchie durch diese Komponente vorgehalten. Ist ein Spielobjekt (Kindobjekt) Teil eines weiteren Spielobjekts (Elternobjekt), werden Änderungen der Position, Rotation und Skalierung des Elternobjektes ebenso auf das Kindobjekt übertragen (Unity 2018m).

### 3.2.2 Lichtquellen

Zur Berechnung der Schattendarstellung dreidimensionaler Modelle werden die Intensität, die Richtung und die Farbe des, auf das Objekt einfallenden, Lichts benötigt. Zur Bereitstellung dieser Informationen existieren in Unity verschiedene Typen von Lichtquellen (Unity 2018f). Unity bietet nach (Unity 2018n) folgende Lichtquellen als Komponenten an:

**Point Lights** - Ein *Point Light* ist an einer Position in der Szene verortet von der es isotrop Licht aussendet. Die Intensität des Lichtes nimmt mit der quadrierten Inverse der Distanz ab (Abstandsquadratsgesetz). Die Entfernung zwischen dem Ort höchster und niedrigster Intensität wird als *Range* bezeichnet.

**Spot Lights** - Diese Lichtkomponente besitzt ebenso wie das *Point Light* eine feste Position und eine Range. Das *Spot Light* definiert einen Öffnungswinkel, durch den das Licht kegelförmig abgestrahlt wird. Die Intensität des Lichts verringert sich innerhalb dieses Kegels nach dem Abstandsquadratsgesetz.

**Directional Lights** - Anders als die vorherigen Lichtkomponenten besitzt das *Directional Light* keine definierte Position innerhalb einer Szene. Es kann als unendlich weit entfernte Lichtquelle betrachtet werden. Alle Objekte in einer Szene werden so mit dem gleichen Einfallswinkel bestrahlt. Es erfolgt keine distanzabhängige Abnahme der Strahlungsintensität.

**Area Lights** - Diese Lichtquelle definiert ein Rechteck von dessen Oberfläche das Licht abgegeben wird, wobei sich die Intensität nach dem Abstandsquadratgesetz verringert.

Die Beleuchtung kann in Unity entweder zur Laufzeit (realtime) berechnet werden oder vorprozessiert (baked) erfolgen. Bei der Echtzeitberechnung hat eine Änderung der Position eines Spielobjekts zur Lichtquelle eine sofortige Neuberechnung der Beleuchtung zur Folge. Ist diese vorprozessiert, werden die Texturen der Spielobjekte mit

den vorberechneten Lichteffekten verrechnet. Eine dynamische Änderung der Beleuchtungsverhältnisse ist somit nicht möglich (Unity 2018f).

### 3.2.3 Physiksimulation

Die in Unity integrierte Physik Engine ermöglicht die Simulation physikalisch korrekten Verhaltens der, in der Szene befindlichen, Spielobjekte. Zu den simulierten physikalischen Kräften zählen unter anderem die Gravitation oder die kinetische Energie, welche durch die Kollision mehrerer Spielobjekte entsteht (Unity 2018j).

Unity stellt eine Reihe vorgefertigter Komponenten zur Verfügung die zur Umsetzung der Physiksimulation genutzt werden. Eine dieser Komponenten ist der *Collider*. Dieser ist unsichtbar und definiert die Form eines Spielobjektes bezüglich der Kollisionsabfrage. Die Form des *Colliders* muss dabei nicht mit der Form des 3D-Modells übereinstimmen. Unity stellt eine Reihe primitiver *Collider*-Komponenten, wie zum Beispiel den Box-*Collider* zur Verfügung. Kollisionen für diese *Collider* sind besonders schnell zu berechnen und können dazu genutzt werden komplexe Geometrien durch einfache geometrische Formen anzunähern (Unity 2018b).

Soll der *Collider* mit der Vermaschung eines Modells übereinstimmen, dieser also die Form des Modells möglichst genau wiedergeben, wird die *MeshCollider*-Komponente verwendet. Diese ermöglicht eine genauere Kollisionserkennung, was allerdings eine höhere Prozessierungszeit zur Folge hat (Unity 2018h).

Soll überprüft werden, ob sich ein *Collider* entlang einer Sichtlinie befindet, kann das sogenannte *Ray Casting* verwendet werden. Dieses Verfahren wird in Unity dazu genutzt, den ersten oder alle *Collider* entlang eines Strahls zu ermitteln. Hierzu wird der koordinatenmäßige Ursprung, der Richtungsvektor und die Länge des auszusendenden Strahls angegeben. Die durch das *Ray Casting* enthaltenen Informationen sind unter anderem die Namen der, von dem Strahl durchlaufenen, Spielobjekte sowie die Koordinaten des Strahleneintritts in den *Collider* eines Spielobjektes (Unity 2018k).

## 3.3 3D-Modelle

Zur Darstellung und Verwaltung von triangulierten 3D-Modellen werden in Unity verschiedene Vermaschungskomponenten verwendet. Der programmiertechnische Zugriff auf eine solche Vermaschung erfolgt über die Klasse *Mesh*. Diese speichert die Knotenpunkte, die Normalen der Knotenpunkte und die Indizes der Polygone eines 3D-Modells als Arrays ab (Unity 2018o).

Die *MeshFilter*-Komponente hält eine Instanz der *Mesh*-Klasse und gibt diese an die *MeshRenderer*-Komponente weiter. Diese stellt das Modell an der Position dar, die in der *Transform*-Komponente vorgehalten wird (Unity 2018i).

Unity verwendet zur Darstellung eines 3D-Modells sogenannte *Materialien* (Materials), die Verweise auf Texturen und Shader enthalten. Sie bestimmen wie die Ober-

fläche eines Modells unter bestimmten Beleuchtungsverhältnissen dargestellt wird (Unity 2018g).

Im Folgenden soll das *Wavefront OBJ*-Format näher beschrieben werden, da es im Rahmen dieser Arbeit dazu genutzt wird, 3D-Modelle in die Unity Game Engine zu importieren.

### Wavefront OBJ

Das von der Firma *Wavefront Technologies* entwickelte OBJ-Format dient zur Speicherung sowie zum Austausch dreidimensionaler Modelldaten. Das Format nutzt die Dateiendung *obj* und beschreibt 3D-Modelle anhand von Linien, Polygonen, Freiform- und Oberflächen. Die Kodierung einer OBJ-Datei kann dabei im ASCII oder im Binärformat erfolgen (Murray u. vanRyper 1996, S. 946-947).

Eine OBJ-Datei beginnt üblicherweise mit einem Kommentar, dieser ist jedoch nicht obligatorisch und kann entfallen. Kommentare werden mit einem Rautezeichen eingeleitet (siehe Listing 3, unter anderem Zeile 1 und 13). Zur Verbesserung der Lesbarkeit können einer OBJ-Datei beliebig viele Leerzeilen eingefügt werden. Alle anderen Zeilen beginnen mit einem Schlüssel, gefolgt von den eigentlichen Daten (Murray u. vanRyper 1996, S. 947). Tabelle 6 gibt eine Übersicht der für die Arbeit relevanten Schlüsselbegriffe.

Tabelle 6: Schlüsselbegriffe des OBJ-Formats (Murray u. vanRyper 1996, S. 947-948).

| **Schlüssel** | **Beschreibung** |
|---|---|
| **Knotendaten** | |
| v | Geometrie-Knoten |
| vt | Textur-Knoten |
| vn | Normale am Knotenpunkt |
| **Elemente** | |
| f | Fläche |
| **Gruppierungen** | |
| g | Gruppenname |
| o | Objektname |
| **Darstellungsattribute** | |
| mtlib | Materialbibliothek |
| usemtl | Materialname |

Üblicherweise wird ein 3D-Modell im OBJ-Format anhand von Polygonflächen beschrieben. Hierzu erfolgt eine koordinatenmäßige Angabe der Polygon-Knotenpunkte unter Verwendung des Schlüssels "$v$" (siehe Listing 3, Zeile 5-12). Eine Polygonfläche wird mit dem Schlüssel "$f$" (siehe Listing 3, unter anderem Zeile 17) ausgewiesen und enthält die Indizes der Polygon-Knotenpunkte. Die direkte Zuweisung von Materialeigenschaften der Polygonflächen ist in der OBJ-Datei nicht möglich. Hierzu wird in der Datei auf eine separate Materialdatei (.mtl) verwiesen. Dies geschieht unter der Ver-

wendung des Schlüsselbegriffs "*mtlib*" (siehe Listing 3, Zeile 3). Die in dieser Datei definierten Materialien legen unter anderem die RGB-Farbwerte sowie die Transparenz der Polygonflächen fest. Referenziert werden Materialien mit dem "*usemtl*"-Schlüssel (siehe Listing 3, unter anderem Zeile 16 und 20). Ein Material gilt dabei so lang für eine Folge von Polygonflächen bis auf ein anderes Material verwiesen wird. Polygonflächen können anhand des Schlüssels "*g*" in Gruppen zusammengefasst werden (siehe Listing 3, unter anderem Zeile 15 und 18), wodurch die Verwaltung und Animation komplexer 3D-Modelle vereinfacht wird (Murray u. vanRyper 1996, S. 949).

Listing 3: Beispiel für einen Kubus mit unteschiedlichen Materialien im Wavefront OBJ-Format (Murray u. vanRyper 1996, S. 950).

```
1  mtllib master.mtl
2
3  v 0.000000 2.000000 2.000000
4  v 0.000000 0.000000 2.000000
5  v 2.000000 0.000000 2.000000
6  v 2.000000 2.000000 2.000000
7  v 0.000000 2.000000 0.000000
8  v 0.000000 0.000000 0.000000
9  v 2.000000 0.000000 0.000000
10 v 2.000000 2.000000 0.000000
11
12 g front
13 usemtl red
14 f 1 2 3 4
15 g back
16 usemtl blue
17 f 8 7 6 5
18 g right
19 usemtl green
20 f 4 3 7 8
21 g top
22 usemtl gold
23 f 5 1 4 8
24 g left
25 usemtl orange
26 f 5 6 2 1
27 g bottom
28 usemt1 purple
29 f 2 6 7 3
```

# Teil III
# Umsetzung

Der dritte Teil der Arbeit enthält die Anforderungsanalyse, den Entwurf sowie die Implementierung der zu entwickelnden Softwarelösung. Hierzu wird zunächst eine Vision der Planungslösung gegeben, anhand derer die Herausarbeitung der funktionalen und nicht-funktionalen Anforderungen an die Planungslösung stattfindet. Die Anforderungen werden dabei textlich beschrieben und mithilfe eines Anwendungsfalls grafisch verdeutlicht. Auf Grundlage der vorher ermittelten Anforderungen wird ein Softwareentwurf des Gesamtsystems vorgestellt, wobei besonders auf dessen Schnittstellen und Prozessabläufe eingegangen wird. Eine der umzusetzenden Anforderungen stellt die Funkausbreitungsberechnung innerhalb des Ballastwassertanks dar. Das hierzu entwickelte Funkausbreitungsmodell wird in diesem Teil beschrieben und sowohl grafisch als auch formelmäßig erläutert. Abschließend wird die softwaretechnische Implementierung der vorher definierten Anforderungen aufgezeigt, wobei im Besonderen auf die entwickelten Klassenstrukturen und Algorithmen eingegangen wird. Aus Gründen des Umfangs werden hierbei lediglich die Hauptfunktionalitäten betrachtet.

# 4 Analyse und Entwurf

In diesem Kapitel wird zunächst eine Vision der zu entwickelnden Planungslösung vorgestellt. Hierzu wird ein grober Funktionsüberblick des Programms gegeben, welcher folgend durch die ermittelten funktionalen und nicht-funktionalen Anforderungen konkretisiert wird. Das zu erstellende System wird im Weiteren durch Architekturentwürfe modelliert, wobei auf die Komponenten, Programmschnittstellen und Datenflüsse der Planungslösung eingegangen wird. Weiter wird der durch die Planungslösung abzubildende Arbeitsprozess der Gasfreimessung beschrieben. Dieser umfasst die Missionsvorbereitung, Missionsdurchführung und Missionsnachbereitung. Abschließend findet eine theoretische Beschreibung des, im Rahmen der Arbeit entwickelten, Funkausbreitungsmodells statt.

## 4.1 Vision

Im Rahmen der Gasfreimessung und Objektdokumentation, muss die zu entwickelnde Softwarelösung Planungsfunktionen für Missionen der *DJI Matrice 100* bereitstellen. Hierzu ist eine Importfunktion notwendig, die das Einlesen eines 3D-Modells des Ballastwassertanks ermöglicht. Das Modell dient als Planungsgrundlage für die auszuführende Mission, wobei die Missionsplanung innerhalb des 3D-Modells erfolgt. Hierbei wird die Flugroute durch das Setzen von Wegpunkten definiert. Den Wegpunkten sollen die vom *DJI*-SDK implementierten Wegpunktaktionen zuweisbar sein (siehe 1.1.1. DJI Onboard SDK - Missionsverwaltung), um so beispielsweise die automatisierte Bilddokumentation des Ballastwassertanks zu ermöglichen. Die Software soll eine Exportfunktion für die Missionsdaten (Flugroute, Aktionen und Missionsparameter) bereitstellen. Die Daten sollen als Datei abgespeichert werden, sodass diese von der Indoor-Navigationslösung importierbar sind. Weiter soll das Starten und Stoppen einer Mission durch die Planungslösung möglich sein.

Die, während der Mission aufgenommenen, Gasmesswerte dienen zur Simulation der Gasverteilung (siehe 2.2. Geostatistische Simulation). Hierzu müssen die Gasmesswerte in das Programm importierbar sein. Anschließend soll das Programm eine geostatistische Simulation durchführen und die Ergebnisse visualisieren.

Aus dokumentarischen Zwecken soll die Softwarelösung während der Mission ein Livebild anzeigen. Hierbei soll eine manuelle Bildaufnahme möglich sein. Zur Bildübertragung sowie zum manuellen Eingreifen in die Flugsteuerung durch einen Piloten im Fehlerfall, ist eine ausreichend gute *WLAN*-Abdeckung der Flugroute notwendig. Um vor dem Missionsstart Abschätzungen über die Empfangsstärke im Ballastwassertank treffen zu können, soll das Programm die Funkausbreitung innerhalb des 3D-Modells berechnen und visualisieren (siehe 1.3 Funkausbreitungsmodelle).

Die Umsetzung der Softwarelösung soll mit der Unity 3D Game Engine erfolgen, welche zahlreiche Konzepte zur Darstellung von 3D-Grafik sowie zum Umgang mit orts-

bezogenen Objekten und deren physikalischen Eigenschaften bietet (siehe 3. Unity 3D). Abbildung 25 zeigt die Anwendungsfälle für die zu entwickelnde Planungslösung und stellt die oben beschriebene Vision funktionell dar.

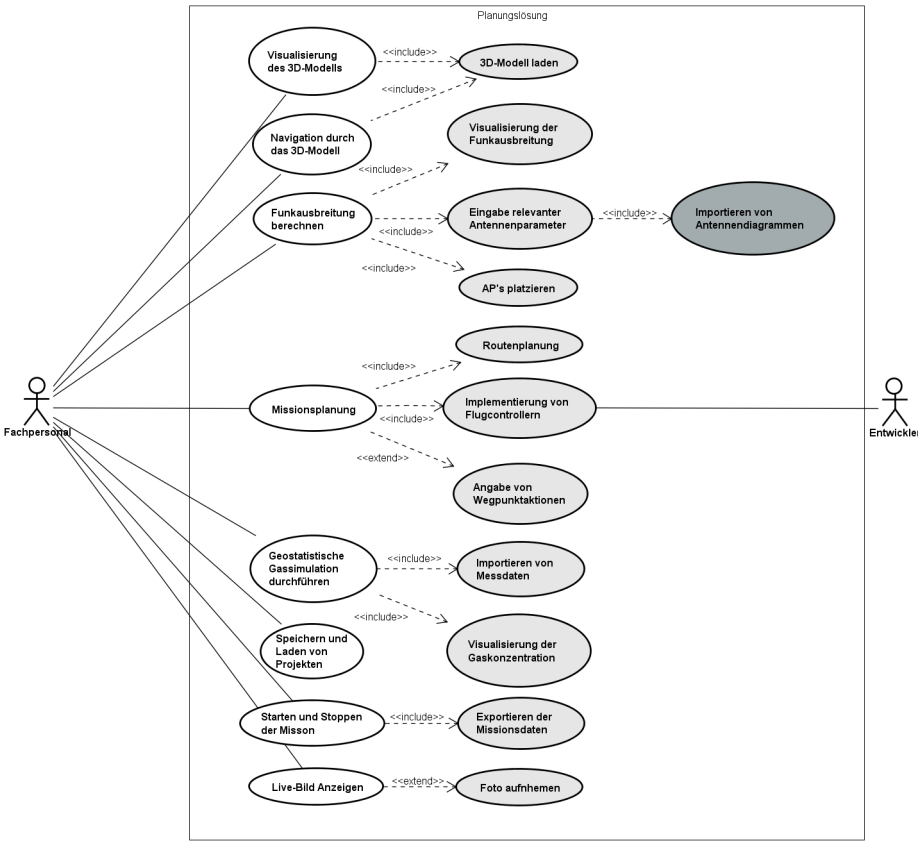



Abbildung 25: Use case Diagramm der Planungslösung.

## 4.2 Anforderungsanalyse

Zur Umsetzung der oben beschriebenen Vision ist eine Spezifizierung der Anforderungen, an die zu entwickelnde Planungslösung, notwendig.

Die Anforderungen werden hierbei in funktionale und nicht-funktionale Anforderungen unterschieden. Funktionale Anforderungen beschreiben dabei die Funktionalitäten eines Systems, die es benötigt um ein fachliches Problem zu lösen. Nicht-funktionale Anforderungen hingegen sind grundlegende Eigenschaften des Systems, die im Architekturentwurf berücksichtigt werden müssen (Balzert 2009, S. 498).

Im Folgenden werden die identifizierten funktionalen und nicht-funktionalen Anforderungen an die Planungslösung aufgelistet und kurz beschrieben.

### 4.2.1 Funktionale Anforderungen

**F01 Import von 3D-Modellen:** Die Softwarelösung ermöglicht den Import von 3D-Modellen im OBJ-Format (siehe 3.3 3D-Modelle - Wavefront OBJ). Bei dem Im-

portvorgang ist die Maßeinheit festlegbar, in dem das Modell konstruiert ist.

**F02 Visualisierung der importierten 3D-Modelle:** Die Softwarelösung visualisiert das importierte 3D-Modell und stellt Funktionen zur Navigation durch das Modell zur Verfügung.

**F03 Navigationsfenster mit Schnittdarstellung:** Zur besseren Navigierbarkeit innerhalb des 3D-Modells soll das Programm ein Navigationsfenster bereitstellen, welches es als Schnitt darstellt.

**F04 Darstellung des 3D-Modells als Drahtgittermodell:** Das importierte 3D-Modell ist als Drahtgittermodell darstellbar.

**F05 Missionsplanung:** Das Programm soll das Planen von Wegpunktmissionen, innerhalb des importierten 3D-Modells, ermöglichen. Die Routenplanung soll durch das Setzen von Wegpunkten erfolgen (siehe 1.1.1. DJI Onboard SDK - Missionsverwaltung).

**F06 Angabe von Wegpunktaktionen:** Den Wegpunkten sollen, die vom *DJI*-SDK definierten, Aktionen zuweisbar sein (siehe Tabelle 1).

**F07 Angabe verschiedener Flugcontroller:** Das Programm soll die Möglichkeit in Betracht ziehen, eine Reihe von Flugcontrollern zu unterstützen. Die prototypische Version der Softwarelösung soll die Unterstützung des *DJI*-Controllers gewährleisten.

**F08 Import von Antennencharakteristiken:** Das Programm stellt eine Funktion bereit, die das Importieren und Verwalten verschiedener Antennencharakteristiken ermöglicht (siehe 1.2.2 Antennenparameter - Antennendiagramm).

**F09 Berechnung der Funkausleuchtung:** Die Softwarelösung ist in der Lage eine Empfangsstärkeberechnung im Hochfrequenzbereich durchzuführen. (siehe 1.3 Funkausbreitungsmodelle). Das hierbei verwendete Funkausbreitungsmodell bezieht das importierte 3D-Modell in die Berechnung ein und soll mehrere AP's sowie deren Sendefrequenz (in GHz) und Antennencharakteristiken berücksichtigen. Zur Visualisierung sollen visuelle Indikatoren bereitgestellt werden, welche Rückschlüsse auf die zu erwartende Empfangsleistung liefern.

**F10 Export der Missionsdaten:** Die Wegpunkte sowie die dazugehörigen Wegpunktaktionen sollen, unter Verwendung einer definierten Schnittstelle, exportierbar sein. Der Missionsparameterexport soll über die selbe Schnittstelle erfolgen (siehe *WayPointSettings* und *WayPointInitSettings* in 1.1.1 DJI Onboard SDK - Missionsverwaltung).

**F11 Starten der Mission:** Die Wegpunktmission soll über das Programm gestartet werden können.

**F12 Stoppen der Mission:** Die Wegpunktmission soll über das Programm gestoppt werden können.

**F13 Visualisierung des Drohnenflugweges:** Die tatsächlich von der Drohne beschriebene Flugroute soll während der Missionsdurchführung visualisiert werden. Die hierzu erforderlichen Positionsdaten werden, über eine definierte Schnittstelle an die Planungslösung, übertragen.

**F14 Anzeige eines Kameralivebildes mit Aufnahmefunktion:** Während der Missionsausführung soll das Bild der Drohnenkamera übertragen und innerhalb der grafischen Oberfläche des Programms zur Anzeige gebracht werden. Weiter soll es möglich sein, Einzelbilder des Kamerabildes abzuspeichern.

**F15 Simulation der Gaskonzentration:** Die Softwarelösung soll mit Hilfe der aufgenommenen Gasmesswerte in der Lage sein eine geostatistische Simulation durchzuführen, um so eine möglichst realistische Gasverteilung zu erhalten. Weiter sollen visuelle Indikatoren bereitgestellt werden, welche die Gaskonzentration und die damit verbundene Gefährdung darstellen.

**F16 Speichern und Laden von Projekten:** Das Programm ermöglicht geplante Missionen mit den planungsrelevanten Elementen Wegpunkte, Wegpunktaktionen und Accsess Points persistent, in einer Projektdatei zu speichern. Der Benutzer soll mit dem Programm im Stande sein, Projektdateien zu laden und deren Bearbeitung fortzuführen.

### 4.2.2 Nicht-funktionale Anforderungen

**N01 Korrektheit:** Die Softwarelösung setzt alle funktionalen Anforderungen wie spezifiziert um.

**N02 Änderbarkeit:** Das Programm soll in Hinblick auf Erweiterungen (zum Beispiel die Implementierung anderer Flugcontroller oder Anpassung des Funkausbreitungsmodells) leicht anpassbar sein.

**N03 Portierbarkeit:** Die Softwarelösung soll durch einen vertretbaren Aufwand auch auf andere Betriebssysteme (zum Beispiel Android) portierbar sein.

**N04 Speicher- und Laufzeiteffizienz:** Das System soll unter dem Testsystem einen fließenden Arbeitsablauf aufweisen. Funktionen wie die Berechnung der Funkausleuchtung oder die Simulation der Gaskonzentration, sollen in vertretbarer Zeit ausgeführt werden können (weniger als 5 Minuten). Das Testsystem ist hierbei wie folgt spezifiziert:

| | |
|---|---|
| Prozessor: | Intel(R) Core(TM) i7-7820HQ CPU @ 2.9 GHz |
| Arbeitsspeicher: | 16 GB 2.400 MHz DDR4 |

Grafikkarte: NVIDIA GeForce 940MX 4 GB

Betriebssystem: Windows 10 Pro (64 Bit)

**N05 Benutzerfreundlichkeit:** Das Programm und dessen Abläufe sowie Funktionalitäten sollen leicht erlernbar sein. Die grafische Benutzeroberfläche ist übersichtlich zu gestalten.

## 4.3 Systementwurf

Das zu erstellende System besteht grundlegend aus den zwei Komponenten *Planungslösung* und Indoor-Navigationslösung. Abbildung 26 stellt die Komponenten und Schnittstellen der Planungs- und Navigationslösung sowie relevante Klassen des OSDK dar (siehe auch 1.1.1 DJI Onboard SDK - Missionsverwaltung beziehungsweise Abbildung 5).

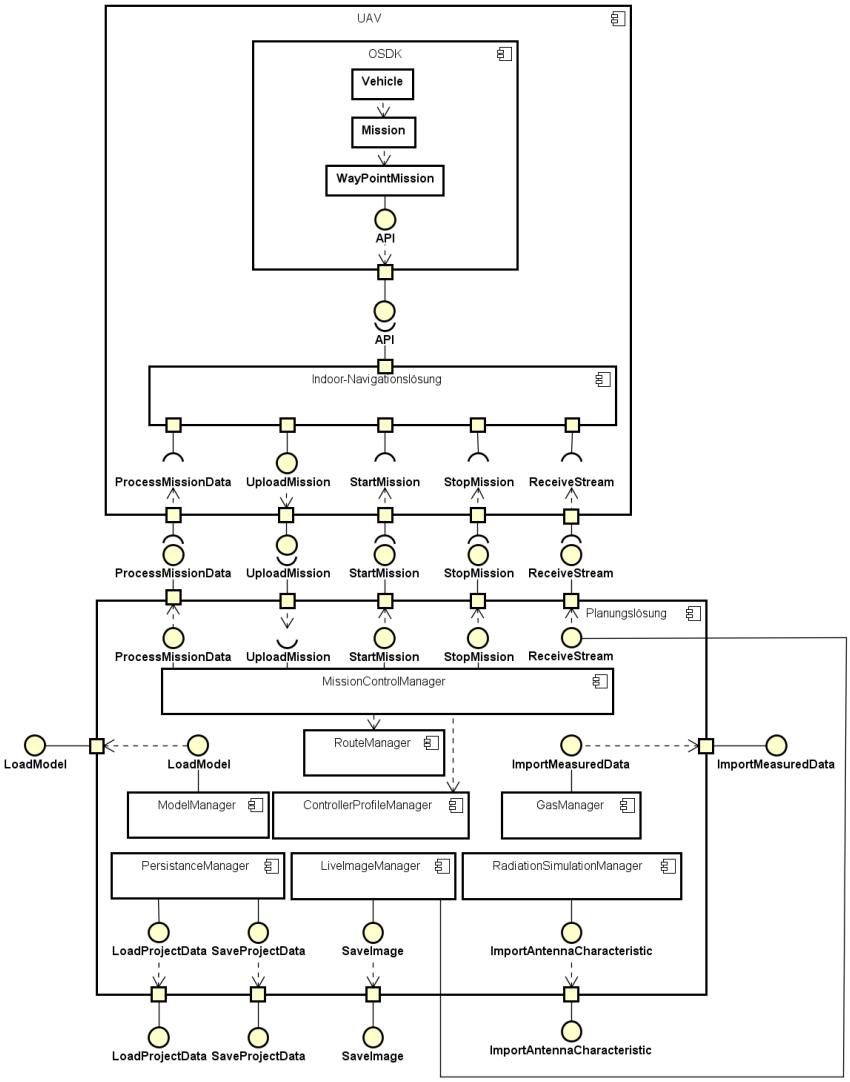



Abbildung 26: Komponentendiagramm des Gesamtsystems.

Das auf der Drohne befindliche *Embedded System* stellt die Navigationslösung bereit. Im Rahmen der Arbeit wird diese Komponente als Black Box betrachtet. Die Navigationslösung stellt der Drohne Wegfindungsfunktionalitäten bereit und wird dazu verwendet die einzelnen Wegpunkte der geplanten Route abzufliegen sowie die entsprechenden Wegpunktaktionen auszulösen. Die in einer Mission definierten Wegpunkte und Aktionen werden von der *Planungslösung* über die Schnittstelle *UploadMission* an die Navigationslösung weitergereicht. Diese nutzt die API des OSDK's zur Initialisierung einer Mission.

Die Kommunikation zwischen der Planungs- und der Navigationslösung findet durch die Komponente *MissionControlManager* statt, welche Teil der Planungskomponente ist. Der *MissionControlManager* kommuniziert über Interfaces mit der Navigationslösung und ermöglicht das Starten (*StartMission*) und Stoppen (*StopMission*) der Mission. Des Weiteren werden die, durch die Navigationslösung berechneten, Positionsdaten der Drohne über die *ProcessMissionData*-Schnittstelle an den *MissionControlManager* weitergereicht.

Weitere Schnittstellen der Planungslösung dienen zum Import des 3D-Modells (*LoadModel*), zum Speichern und Laden von Projekten (*LoadProjectData*/*SaveProjektData*), zum Abspeichern der Livebilder (*SaveImage*) und dem Import von Antennendiagrammen (*ImportAntennaCharacteristic*) sowie dem Import der Gasmesswerte (*ImportMeasuredData*). Im Folgenden werden die in der Abbildung 26 dargestellten Komponenten der Planungslösung, hinsichtlich ihrer Aufgaben erläutert.

**MissionControlManager** - Diese Komponente stellt Schnittstellen zur Kommunikation mit der Navigationslösung bereit. Sie dient zum Export der Missionsdaten sowie dem Starten und Stoppen einer Mission. Der Komponente werden die Positionsdaten der Drohne von der Navigationslösung übermittelt, um so den Flugweg der Drohne während der Missionsausführung anzeigen zu lassen. Die Visualisierung des Flugweges fällt ebenfalls in den Verantwortungsbereich dieser Komponente. Die für den Missionsexport benötigten Daten wie Wegpunktkoordinaten und Wegpunktaktionen erhält der *MissionControlManager* von der Komponente *Route- Manager*. Der *MissionControlManager* bezieht die Missionsparameter vom *ControllerProfileManager*.

**RouteManager** - Der *RouteManager* stellt Funktionen zur Flugroutenerstellung bereit. Die Komponente ist für das Hinzufügen und Löschen von Wegpunkten und Aktionen zuständig. Des Weiteren sind hier Funktionen zum Verschieben eines Wegpunktes innerhalb des 3D-Modells implementiert. Die verfügbaren Aktionsarten und Parameter (zum Beispiel die maximale Anzahl von Wegpunkten oder Aktionen) enthält der *RouteManager* von der Komponente *ControllerProfileManager*.

**ControllerProfileManager** - Diese Komponente dient der Definition der verfügbaren Wegpunktaktionen und Missionsparameter. In dieser Komponente werden die Namen der Wegpunktaktionen definiert und die Werte der Missionsparameter vorgehalten.

**ModelManager** - Diese Komponente dient dem Import des 3D-Modells. Des Weiteren stellt sie Navigationsfunktionen für das Kamera-Spielobjekt in der Szene bereit. Diese werden benötigt, um eine Navigation durch das 3D-Modell zu ermöglichen. Die Schnittdarstellung des 3D-Modells wird hier ebenfalls realisiert.

**RadiationSimulationManager** - In dieser Komponente sind Methoden implementiert, die zur Umsetzung der Funkausbreitungsberechnung dienen. Hierzu ermöglicht der *RadiationSimulationManager* das Setzen von AP's sowie die Angabe relevanter Antennenparameter. Die Komponente realisiert sowohl den Import und die Verwaltung von Antennendiagrammen, als auch die Visualisierung der Empfangsleistung.

**GasManager** - Der *GasManager* dient zur Durchführung der geostatistischen Simulation der Gaskonzentration. Hierzu wird eine Funktion zum Import der Gasmesswerte bereitgestellt.

**LiveImageManager** - Diese Komponente dient zur Anzeige und Speicherung des Livebildes.

**PersistanceManager** - Das Speichern und Laden von persistenten Datenobjekten ist die Aufgabe des *PersistanceManagers*. Zu diesen Datenobjekten zählen die Projektdatei, Flugcontroller samt Missionsparameter sowie die importierten Antennencharakteristiken.

### 4.3.1 Datenschnittstellen

Im Folgenden werden die, vom Gesamtsystem zu implementierenden, Schnittstellen beschrieben.

#### LoadModel

Die Schnittstelle *LoadModel* dient dazu das 3D-Modell des Ballastwassertanks in die Planungslösung zu importieren. Die Schnittstelle wird von der *ModelLoader*-Komponente bereitgestellt, wobei das Modell im OBJ-Format (siehe 3.3 3D-Modelle - Wavefront OBJ) erwartet und von der Schnittstelle in *Mesh*-Komponenten (siehe 3.3 3D-Modelle) überführt wird, welche von der Game Engine als dreidimensionales Modell dargestellt werden. Abbildung 27 bildet den oben beschriebenen Vorgang als Datenflussdiagramm ab.

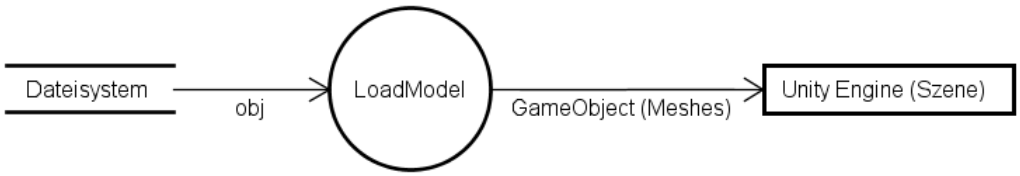



Abbildung 27: Datenflussdiagramm des Modellimports.

### Save/LoadProjectData

Das Speichern und Laden von Projekten wird durch die *Save*- bzw. *LoadProjectData*-Schnittstelle realisiert. Zu speichern sind hierbei zum einen die dreidimensionalen Koordinaten der Wegpunkte sowie die dazugehörigen Wegpunktaktionen, inklusive der Aktionsparameter. Weiter sind die Positionen der AP's, deren zugewiesene Antennendiagramme und die angegebene Sendefrequenz (in GHz) zu speichern. Die Ergebnisse der Funkausbreitungsberechnung sind ebenfalls in die Projektdatei zu übernehmen. Die persistente Speicherung der für das Projekt relevanten Spielobjekte erfolgt durch eine Containerklasse, welche sich innerhalb der *PersistanceManager*-Komponente befindet. Die Containerklasse erfüllt das *Serializable*-Interface und kann somit durch einen Serialisierer/Deserialisierer in Dateiform serialisiert beziehungsweise deserialisiert werden (siehe Abbildung 28).

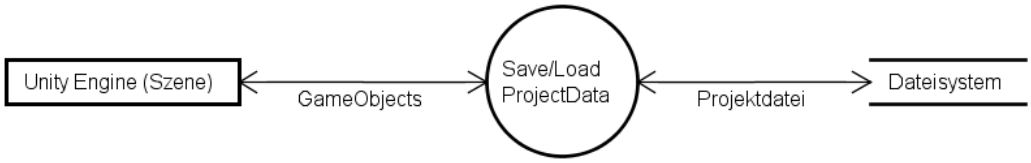



Abbildung 28: Datenflussdiagramm zum Speichern und Laden einer Projektdatei.

### UploadMission

Die Schnittstelle *UploadMission* wird von der Navigationslösung bereitgestellt und dient zum Übertragen der Missionsdaten von der Planungslösung an die Navigationslösung. Die Planungslösung stellt hierzu die Wegpunktdaten (Positionen und Aktionen der Wegpunkte) in Spielobjekten und die Missionsparameter in dem verwendeten Controllerprofil (hier *DJIControllerProfile*) bereit. Die Missionsdaten werden, durch die *ParseMission*-Methode, in Abhängigkeit vom verwendeten Flugcontroller, in das jeweilige Format geparst, im XML-Format abgespeichert (*SaveMission*) und an die Navigationslösung übertragen (*UploadMission*). Abbildung 29 zeigt das Datenflussdiagramm zum Missionsupload.


Abbildung 29: Datenflussdiagramm zum Mission-Upload.

### ImportAntennaCharacteristic

Das Importieren von Antennendiagrammen wird durch die *ImportAntennaCharacteristic*-Schnittstelle umgesetzt. Zur Abbildung der horizontalen und vertikalen Abstrahlungscharakteristik werden EIRP-Werte für diskrete Raumwinkel $\theta$ und $\varphi$ angegeben (siehe 1.2.2 Antennenparameter - Antennendiagramm beziehungsweise 1.2.2 Antennenparameter - Effektive Sendeleistung und Empfangsleistung). Die Raumwinkel sowie die dazugehörigen EIRP-Werte, werden im CSV-Format in die Planungslösung importiert und der *RadiationSimulationManager*-Komponente als Objekt vom Typ *AntennaCharacteristic* übergeben, welches die oben genannten Attribute vorhält (siehe Abbildung 30).

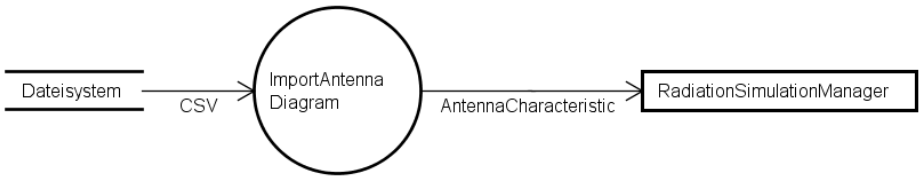



Abbildung 30: Datenflussdiagramm des Antennendiagramm-Imports.

### ImportMeasuredData

Zur Simulation der Gaskonzentration werden der Planungslösung Gasmesswerte bereitgestellt. Die Messwerte werden durch die *ImportMeasuredData*-Schnittstelle, unter Verwendung des CSV-Formats, in die Planungslösung geladen und durch die Angabe der dreidimensionalen Raumposition $(x, y, z)$ sowie durch den Konzentrationswert (in ppm) beschrieben. Die Positionen der Messstellen und die dazugehörigen Konzentrationswerte werden der *GasManager*-Komponente als Liste vierdimensionaler Vektoren übergeben. Abbildung 31 gibt eine Übersicht des Datenflusses im Zuge des Messdaten-Imports.

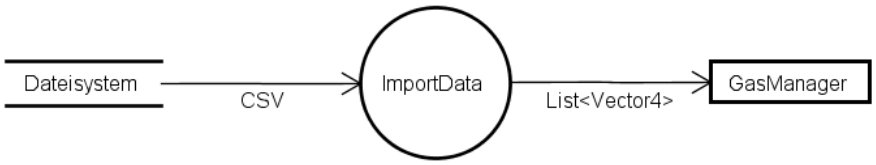



Abbildung 31: Datenflussdiagramm zum Import der Gasmessdaten.

### ProcessMissionData

Das Versenden missionsbezogener Nachrichten von der Navigations- an die Planungslösungs wird über die Schnittstelle *ProcessMissionData* realisiert. Zu den versendeten Daten zählen die dreidimensionale Raumposition der Drohne innerhalb des 3D-Modells $(x, y, z)$ sowie Bestätigungsnachrichten, welche der Planungslösung Aufschluss über den Missionsstatus geben (siehe Abbildung 32).

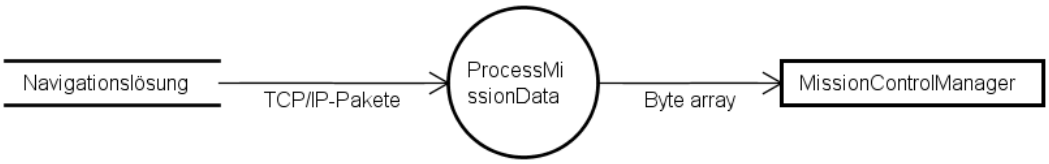



Abbildung 32: Datenflussdiagramm der *ProcessMissionData*-Schnittstelle.

### ReceiveStream und SaveImage

Der Zugriff auf das Livebild der Drohnenkamera erfolgt über den, von der Kamera bereitgestellten, Livestream im *MJPEG* Videocodec. Die Einzelbilder des Streams werden von der *ReceiveStream*-Methode in Byte-Arrays umgewandelt und dem *LiveImageManager*, für die weitere Verarbeitung, zur Verfügung gestellt (siehe Abbildung 33).

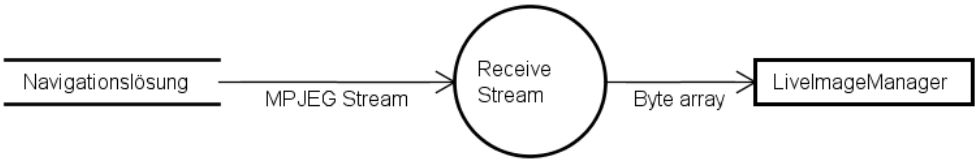



Abbildung 33: Datenflussdiagramm des *ReceiveStream*-Interfaces.

Das im *LiveImageManager* vorgehaltene Byte-Array wird, durch die *SaveImage*-Schnittstelle, in das Dateisystem des Zielcomputers geschrieben. Hierbei wird das Bildformat PNG verwendet (siehe Abbildung 34).

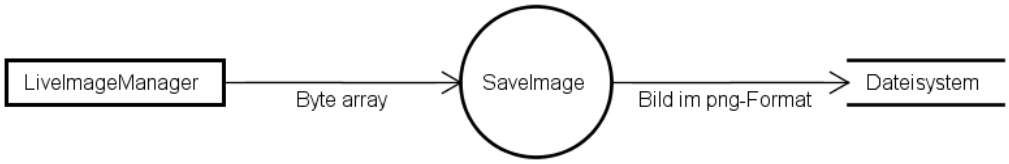



Abbildung 34: Datenflussdiagramm der *SaveImage*-Schnittstelle.

### 4.3.2 Prozessbeschreibung

Während die oben gezeigten Diagramme den strukturellen Aufbau der Planungslösung verdeutlichen, werden im Folgenden die zeitlichen Aspekte des, durch die Planungslösung abzubildenden, Prozessablaufes betrachtet. Abbildung 35 zeigt das Aktivitätsdiagramm des zu automatisierenden Prozesses, wobei zwischen den Rollen *Fachpersonal*, *Planungslösung* und *Navigationslösung* unterschieden wird. Die von den Rollen durchzuführenden Aktivitäten werden weiter in die Teilprozesse der *Missionsvorbereitung*, *Missionsdurchführung* und der *Missionsnachbereitung* beziehungsweise Gasfreimessung unterteilt.

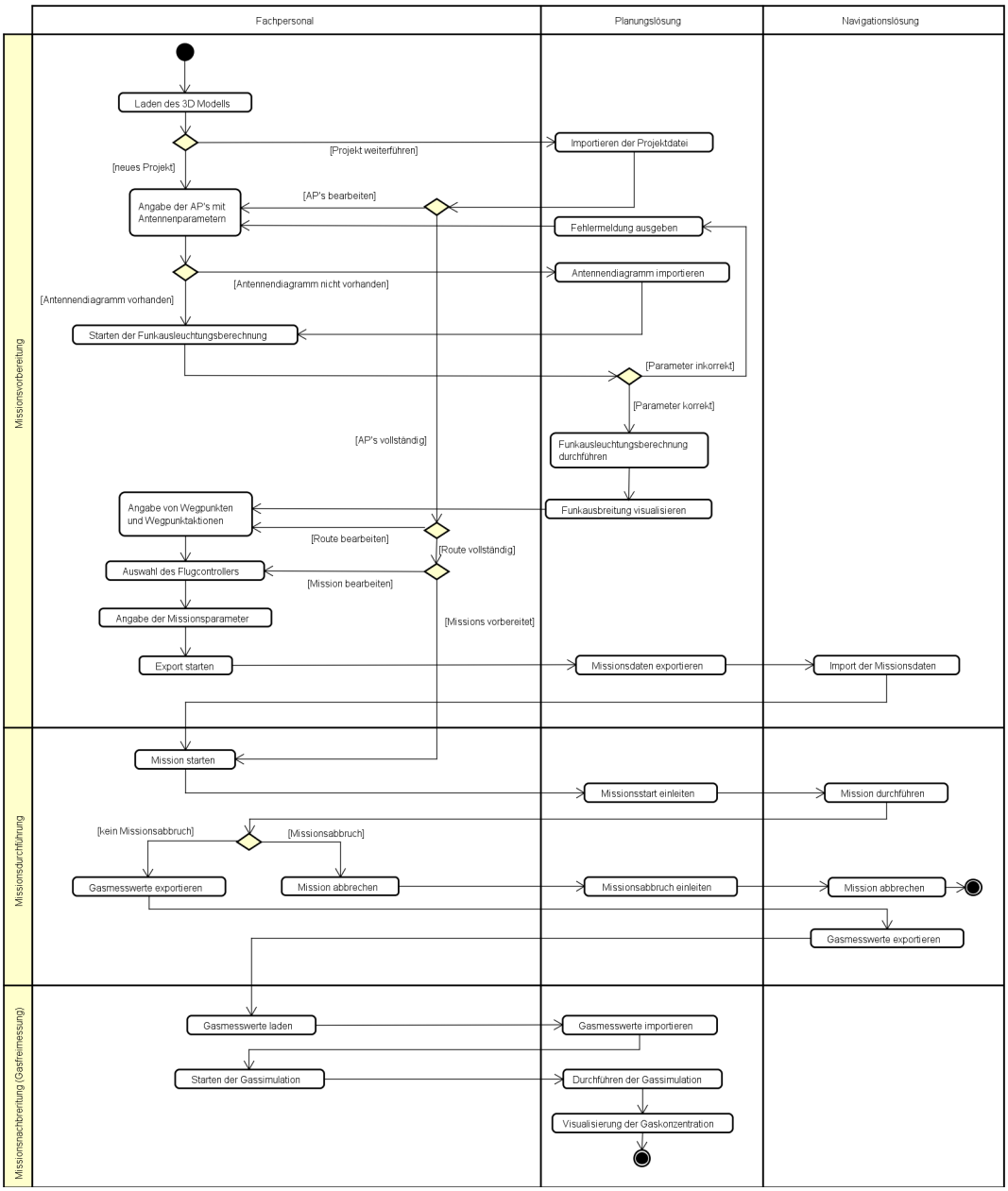



Abbildung 35: Aktivitätsdiagramm einer Missionsdurchführung.

Da das Fachpersonal für die Bedienung der Planungslösung zuständig ist, startet dieses die Missionsvorbereitung durch das Importieren eines 3D-Modells. Hiernach ist es möglich eine vorhandene Mission durch eine Projektdatei zu laden, oder eine neue Mission zu erstellen. Wird eine neue Mission erstellt, erfolgt zunächst die Platzierung der AP's im 3D-Modell sowie die Definition der Antennenparameter Sendefrequenz und Antennendiagramm. Ist das gewünschte Antennendiagramm bereits vorhanden, kann dies dem AP zugewiesen werden, andernfalls wird dieses durch das Fachpersonal importiert. Sind alle Antennenparameter korrekt eingegeben, startet das Fachpersonal die Funkausbreitungsberechnung. Nach Abschluss der Berechnungen wird die Funkausbreitung durch die Planungslösung visualisiert.

Im nächsten Schritt wird die Mission unter Beachtung der Funkausbreitung geplant, wobei die Wegpunkte der zu befliegenden Route innerhalb des 3D-Modells gesetzt werden. Dem Fachpersonal ist es in diesem Schritt möglich, den Wegpunkten Aktionen zuzuweisen. Ist die Route vollständig, gibt das Fachpersonal den für die Mission verwendeten Flugcontroller samt Missionsparameter an und exportiert die Mission mit Hilfe der Planungslösung. Der letzte Schritt, im Rahmen der *Missionsvorbereitung*, ist das Importieren der Missionsdaten in die Navigationslösung.

Das Fachpersonal startet die Mission und somit die Phase der *Missionsdurchführung*. Die Planungslösung signalisiert der Navigationslösung, dass die Mission gestartet werden soll. Kommt es während der Mission zum Fehlerfall oder bricht das Fachpersonal die Mission ab, ist der Prozess beendet. Wird die Mission erfolgreich durchgeführt, können die Gasmesswerte mit Hilfe der Navigationslösung exportiert werden.

Die Phase der *Missionsnachbereitung* beziehungsweise der Gasfreimessung beginnt mit dem Laden der Gasmessdaten in die Planungslösung. Nach dem Import der Messdaten, startet das Fachpersonal die geostatistische Simulation der Gasmesswerte und visualisiert anschließend die Simulationswerte. Dieser Schritt stellt das Ende des Prozesses dar und liefert die Grundlage für die Einschätzung des Gefahrenpotentials der im Ballastwassertank befindlichen Gase.

### 4.3.3 Benutzeroberfläche

Ein Entwurf für die grafische Oberfläche der Planungslösung ist Abbildung 36 zu entnehmen. Da die Planungslösung auch auf mobilen Endgeräten wie Tablets oder Smartphones portierbar sein soll, wird auf ein einfaches Layout mit großen Bedienelementen gesetzt. Dies soll eine zuverlässige und einfache Steuerung mittels Touchpad ermöglichen.

Die Werkzeugleiste ist am oberen Rand des Anwendungsfensters angebracht und stellt die Funktionalitäten des Programms mit Hilfe von Buttons bereit. Diese dienen entweder dazu eine Funktion zu starten, oder dazu ein weiteres Werkzeugfenster anzuzeigen, das weitere Funktionalitäten enthält und/oder zur Konfiguration des Werkzeugs dient.

Am linken Rand des Programmfensters befindet sich die Wegpunkt- beziehungsweise AP-Liste, welche die in der Planungsphase gesetzten Wegpunkte oder AP's beinhaltet.

Die entsprechende Liste wird hierbei entsprechend des Bearbeitungsmodus angezeigt, wobei die Listenelemente auswählbar sind. Darunter befindet sich das Datenpanel. Dieses dient zur Verwaltung von Wegpunktaktionen und Antennenparametern.

Am rechten Rand der Planungslösung werden das Navigationsfenster und das Livebild bereitgestellt. Das Navigationsfenster zeigt das, im Modellbereich dreidimensional dargestellte, Modell als zweidimensionalen Schnitt in der Draufsicht.

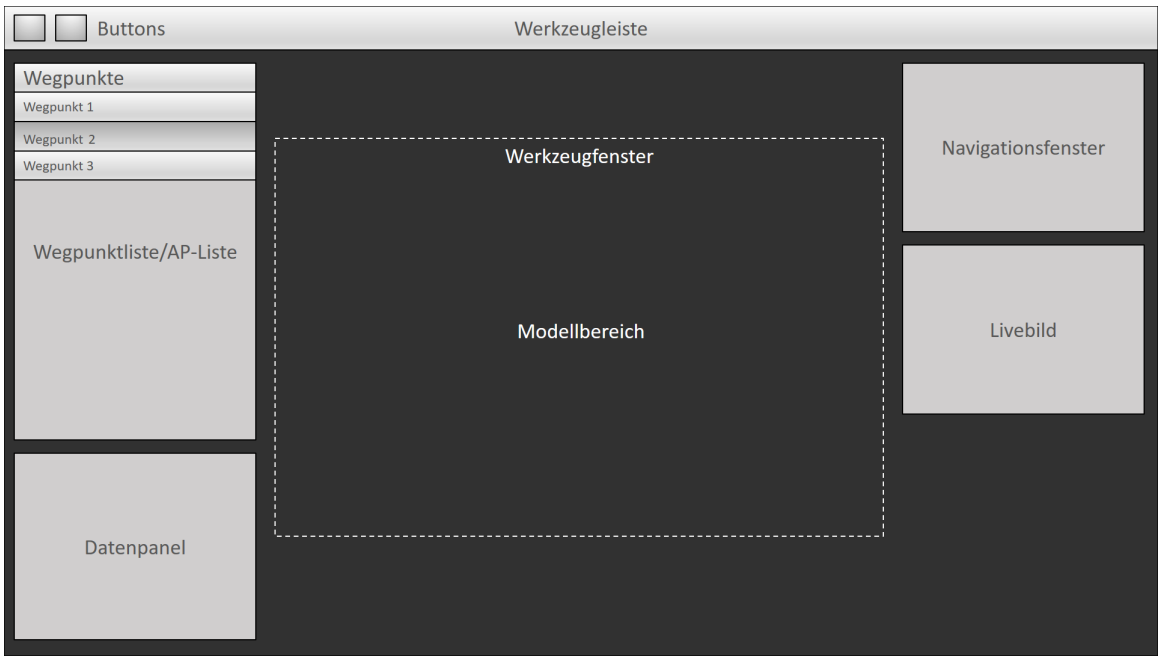



Abbildung 36: Benutzeroberfläche der Planungslösung im Entwurf.

## 4.4 Funkausbreitungsmodell

Im Folgenden wird das im Rahmen der Arbeit entwickelte Funkausbreitungsmodell theoretisch beschrieben. Dieses dient innerhalb des Projekts dazu die Empfangsleistung (siehe 1.2.2 Antennenparameter - Effektive Sendeleistung und Empfangsleistung) im Ballastwassertank zu bestimmen.

In Anbetracht zukünftiger Nutzungen der Planungslösung, soll das Modell nicht ausschließlich in Ballastwassertanks, welche überwiegend aus Stahl bestehen, einsetzbar sein. Hierzu berücksichtigt das entwickelte Modell unterschiedliche Materialeigenschaften des Untersuchungsgebietes (vergleichbar mit den semi-empirischen Modellen in siehe 1.3.3 Semi-empirische Modelle). Weiter soll, wie bei den physikalischen Modellen, die konkrete Geometrie des importierten 3D-Modells in die Berechnung der Empfangsleistung mit einfließen. Da das entwickelte Modell Elemente aus den semi-empirischen und den physikalischen Modellen verwendet, wird es im Folgenden als *semi-physikalisches Modell* bezeichnet.

Das Funkausbreitungsmodell berechnet die Empfangsleistung $RSL_{xyz}$ an regelmäßigen Rasterpunkten im $\mathbb{R}^3$. Die in 1.3.4 Physikalische Modelle - Ray Launching Model erwähnten

Detektorflächen, werden folglich zu dreidimensionalen Detektorzellen, welche den Raum innerhalb des 3D-Modells in dreidimensionale Voxel diskretisieren und zum Feststellen eines Strahlendurchlaufs dienen. Eine Detektorzelle wird durch ihre Position $(x, y, z)$ im Raster sowie durch die Kantenlänge der Zelle $w$ bestimmt (siehe Abbildung 37). Die Diskretisierung des Raumes ermöglicht die Ermittlung des Signallaufweges $d$. Aus diesem lässt sich die Signalempfangsstärke $RSL_{xyz}$, für jede Detektorzelle im Raum, berechnen.

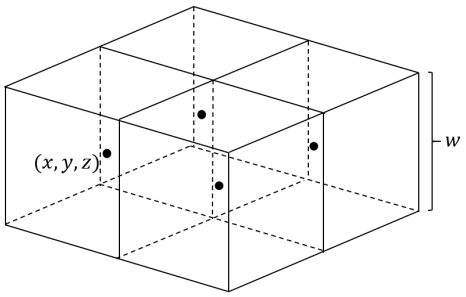



Abbildung 37: Dreidimensionales Detektorzellen-Raster.

Das Modell approximiert die elektromagnetische Welle als geometrischen Strahl und berücksichtigt die Reflexion an Oberflächen nach dem Brechungsgesetz von *Snellius* sowie die Durchdringung von Materialien durch die Dämpfungsfaktoren $D_i$ (siehe 1.3.3 Semi-empirische Modelle und 1.3.4 Physikalische Modelle). Die Strahlen werden hierbei, nach dem *Ray Launching Modell*, in die Umgebung gesendet (siehe 1.3.4 Physikalische Modelle - Ray Launching Model). Die dabei ausgestrahlte Sendeleistung (EIRP) ist, in Bezug auf das Antennendiagramm $C$, abhängig von den Raumwinkeln $\varphi$ und $\theta$. Die Empfangsleistung $RSL_{xyz}$ an der Position einer Detektorzelle berechnet sich aus der Differenz der Sendeleistung in Richtung $\varphi$ und $\theta$ sowie des Pfadverlustes $P_r$ in Abhängigkeit des Signallaufweges $d$. Der Signallaufweg $d$ ergibt sich aus der Summe der Teilwege $d_1$ bis $d_n$. Diese werden durch die Teilstrecken zwischen den Interaktionspunkten (IP) $IP_1$ bis $IP_n$ mit dem 3D-Modell, beziehungsweise den Durchlaufpunkten (DP) $DP_1$ bis $DP_n$ der Detektorzellen definiert. Die Gesamtheit aller Teilwege bildet den Signallaufweg $d$, welcher die Entfernung zwischen dem AP und der entsprechenden Detektorzelle darstellt (siehe Abbildung 38).

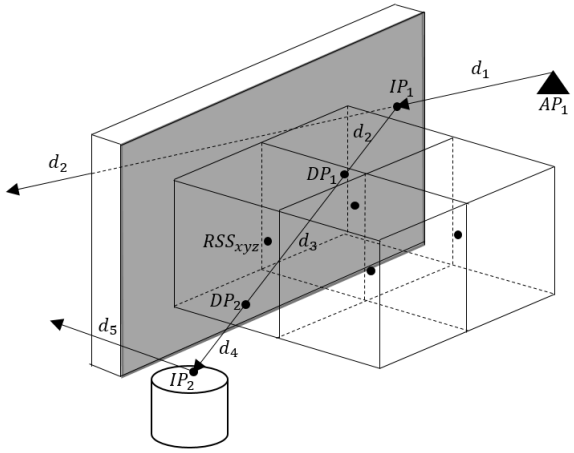



Abbildung 38: Signallaufwege im semi-physikalischen Modell.

Die Berechnung der Empfangsleistung an der Position einer Detektorzelle erfolgt dabei unter Verwendung folgender Formel

$$RSL_{xyz}(\theta, \varphi, d) = max(EIRP(\theta, \varphi) - P_r(d)) \tag{21}$$

wobei,

| | |
|---|---|
| $RSL_{xyz}(\theta, \varphi, d)$ | die maximale Empfangsstärke an der Position x,y und z in dBm, |
| $EIRP(\theta, \varphi)$ | die Sendeleistung in Richtung der Raumwinkel $\theta$ sowie $\varphi$ in dBm und |
| $P_r(d)$ | die Signaldämpfung in dBm in Abhängigkeit des Signallaufwegs in m ist. |

Das Modell berechnet die maximale Empfangsleistung an einer Position. In 21 wird hierzu der Maximalwert der Differenz aus der effektiven Sendeleistung $EIRP(\theta, \varphi)$ und dem Pfadverlust zum Empfänger $P_r(d)$ beziehungsweise zur Detektorzelle ermittelt. Die Berechnung des Raumwinkelabhängigen EIRP erfolgt durch

$$EIRP(\theta, \varphi) = EIRP_v(\theta, \theta_{abs}) + EIRP_h(\varphi, \theta_{abs}) \tag{22}$$

wobei,

| | |
|---|---|
| $EIRP(\theta, \varphi)$ | die Sendeleistung in Richtung der Raumwinkel $\theta$ sowie $\varphi$ in dBm, |
| $EIRP_v(\theta, \theta_{abs})$ | der vertikale Anteil der vertikalen Sendeleistung in dBm, |
| $EIRP_h(\varphi, \theta_{abs})$ | der horizontale Anteil der vertikalen Sendeleistung in dBm sowie |
| $\theta_{abs}$ | der Absolutwert des Raumwinkels $\theta$ in $^\circ$ ist. |

Hierbei werden der horizontale und vertikale Anteil an der Sendeleistung aufaddiert.

Die Berechnung des vertikalen Anteils des EIRP erfolgt durch

$$EIRP_v(\theta, \theta_{abs}) = \begin{cases} \frac{\theta_{abs} \cdot C(\theta)}{90} & \text{für } \theta_{abs} > 90 \\ \frac{(90-\theta_{abs}) \cdot C(\theta)}{90} & \text{für } \theta_{abs} \leq 90 \end{cases} \tag{23}$$

wobei,

| | |
|---|---|
| $EIRP_v(\theta, \theta_{abs})$ | der Vertikalanteil der Sendeleistung in dBm, |
| $\theta_{abs}$ | der Absolutwert des Raumwinkels $\theta$ in °, |
| $C(\theta)$ | die Sendeleistung des vertikalen Antennendiagramms in dBm, bezogen auf den Winkel $\theta$ ist. |

Der horizontale Anteil wird nach folgender Formel berechnet

$$EIRP_h(\varphi, \theta_{abs}) = \begin{cases} \frac{(90-\theta_{abs}) \cdot C(\varphi)}{90} & \text{für } \theta_{abs} > 90 \\ \frac{\theta_{abs} \cdot C(\varphi)}{90} & \text{für } \theta_{abs} \leq 90 \end{cases} \tag{24}$$

wobei,

| | |
|---|---|
| $EIRP_v(\varphi, \theta_{abs})$ | der Horizontalanteil der Sendeleistung in dBm, |
| $\theta_{abs}$ | der Absolutwert des Raumwinkels $\theta$ in °, |
| $C(\varphi)$ | die Sendeleistung des horizontalen Antennendiagramms in dBm, bezogen auf den Winkel $\varphi$ ist. |

Zur Berechnung des Horizontal- und Vertikalanteils stehen die Werte der horizontalen und vertikalen Antennendiagramme $C(\varphi)$ und $C(\theta)$ zur Verfügung. Da diese die dreidimensionale Antennencharakteristik lediglich in den Ebenen des maximalen Antennengewinns abbildet, müssen die restlichen Sendeleistungen, welche nicht direkt auf den Maximalebenen liegen, ermittelt werden. Die Berechnung der horizontalen und vertikalen Anteile dieser Sendeleistungen, in beliebige Raumrichtungen, erfolgt prozentual durch die Verwendung des absoluten Vertikalwinkels $\theta_{abs}$. Beträgt dieser Winkel 0°, setzt sich die Sendeleistung zu 100% aus dem Wert $C(\theta)$ und zu 0% aus dem Wert der horizontalen Charakteristik $C(\varphi)$ zusammen. Besitzt $\theta_{abs}$ den Wert 45°, bildet sich die Sendeleistung zu jeweils 50% aus $C(\theta)$ und $C(\varphi)$. Der Winkel $\theta_{abs}$ kann dabei einen Wert von 0° bis 180° annehmen. Übersteigt der Winkel den Wert von 90°, wird dieser von 90° subtrahiert, um so die Berechnung der Sendeleistung für alle Oktanten zu realisieren.

Die Signaldämpfung $P_r(d)$ wird in Abhängigkeit des Signalweges $d$ in Metern berechnet. Hierbei wird die Freiraumdämpfung sowie die Dämpfungseigenschaften unterschiedlicher Materialien der Umgebung durch die Dämpfungswerte $D_i$ abgebildet. Für $D_i$ können Werte verschiedener Baumaterialien aus der Literatur entnommen werden (siehe Retscher u. Tatschl 2017 oder Rech 2006, S. 350).

Im Falle einer überwiegend metallischen Umgebung, wie es bei Ballastwassertanks der Fall ist, kann auf die Angabe von $D_i$ verzichtet werden, da bei metallischen Oberflächen praktisch keine Durchdringung des Materials, durch die elektromagnetischen Wellen, im vorliegenden Frequenzbereich (2,4 GHz und 5 GHz) stattfindet (siehe 1.3.3 Semi-empirische Modelle).

$$P_r(d) = 20\ log\left(\frac{4\ \pi\ d}{\lambda}\right) + \sum_{i=1}^{n} D_i \tag{25}$$

wobei,

| | |
|---|---|
| $P_r(d)$ | die Signaldämpfung in dBm, abhängig von dem Signallaufweg $d$ in m, |
| $\lambda$ | die Wellenlänge in Metern, |
| $\sum_{i=1}^{n} D_i$ | die Summe der Dämpfungswerte $D_i$ in dBm ist. |

# 5 Implementierung

In diesem Kapitel wird die softwaretechnische Umsetzung der Planungslösung beschrieben. Aus Gründen des Umfangs erfolgt hierbei lediglich eine Beschreibung der Hauptfunktionalitäten. Diese werden durch Klassen- und Sequenzdiagramme grafisch dargestellt und algorithmisch durch Pseudocode erläutert.

## 5.1 Modellimport

Der Modellimport wird in der Planungslösung durch den *ModelManager* realisiert, welcher in Unity *GameObject* umgesetzt ist. Dem Manager ist die *ModelLoader*-Klasse angefügt, welche von der Klasse *MonoBehaviour* erbt und somit, im Unity-Framework, eine *Component* darstellt. Die von dem *ModelLoader* bereitgestellte Funktion *OpenFileDialog* öffnet ein Dateiauswahl-Fenster, in dem das zu importierende 3D-Modell angegeben wird. Die vom Manager implementierte *LoadModel*-Methode realisiert den Dateiimport, des 3D-Modells, in die Unity-Engine. Hierzu nutzt der Manager die Klasse *OBJModelImporter*, welche die Methoden *ImportOBJ* und *GetModelAsGameObject* zur Verfügung stellt.

Die Modelldaten (Knoten, Polygonflächen, Gruppen, etc.) werden von der *ImportOBJ*-Methode ausgelesen, jede Modellgruppe wird hierbei als Instanz der Klasse *Modelpart* gespeichert. Die *GetModelAsGameObject*-Methode dient zur Überführung des externen OBJ-Formats in die Unity internen *Meshes* und gibt das Modell als GameObject zurück (siehe Abbildung 39).

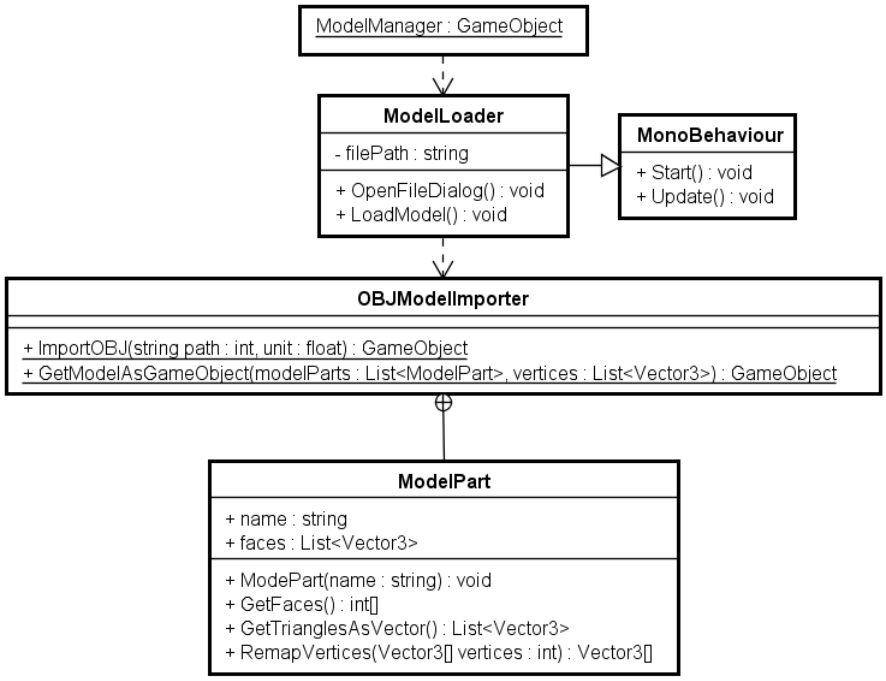



Abbildung 39: Klassendiagramm der *ModelManager*-Komponente.

Abbildung 40 zeigt das Sequenzdiagramm des Importvorgangs. Dieser wird vom Fachpersonal durch den Aufruf des Dateiauswahl-Dialogs gestartet. Hat dieses eine OBJ-Datei selektiert, erfolgt das Laden des Modells durch den *ModelLoader*. Dieser ruft die *ImportOBJ*-Methode des *OBJModelImporter* auf, welche wiederum dessen *GetModelAsGameObject*-Methode aufruft und dem *ModelLoader* das 3D-Modell als *GameObject* übergibt.

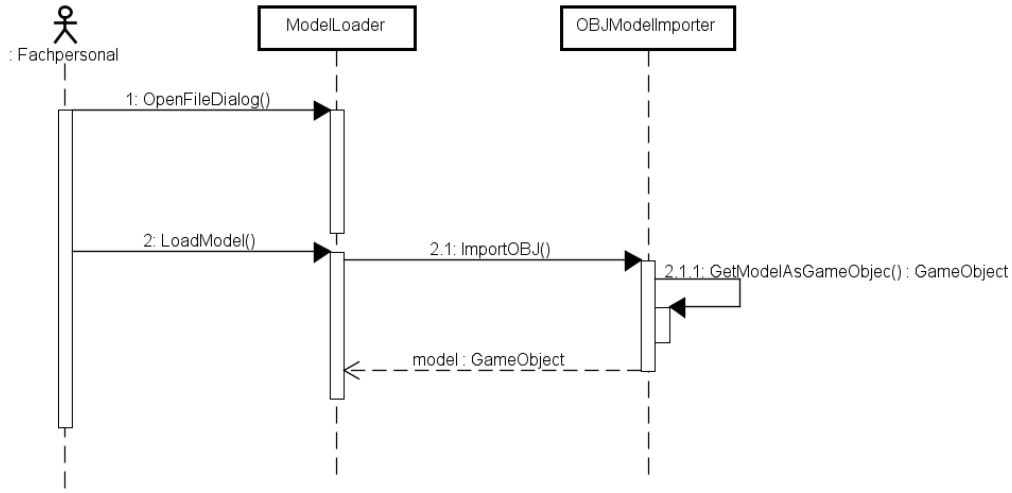



Abbildung 40: Sequenzdiagramm des Modellimports.

Das Auslesen der OBJ-Datei erfolgt unter Verwendung der in Tabelle 6 aufgeführten Schlüsselzeichen. Die Datei wird dabei zeilenweise auf das Vorhandensein dieser Zeichen geprüft. Wird eines gefunden, findet eine vom Schlüsselzeichen abhängige Fallunterscheidung statt. Diese dient dazu die Modelldaten korrekt auszulesen und in die entsprechenden Listen abzuspeichern. Beim Auslesen der Vertex Koordinaten, werden die x- und y-Koordinaten vertauscht, um die Modelldaten, welche im rechtshändigen Koordinatensystem vorliegen, in das linkshändige Koordinatensystem von Unity zu transformieren. Die Koordinaten werden außerdem mit dem Skalierungsfaktor *unit* multipliziert, welcher von dem Benutzer beim Import angegeben wird. So wird sichergestellt, dass sich die importierten Daten in der Maßeinheit Meter befinden, was zum Vereinheitlichen späterer Rechenvorgänge dient. Die Knotenpunkte werden fortlaufend in der *vertices*-Liste gespeichert, wohingegen die Polygonflächen den einzelnen Instanzen der Klasse *ModelPart* zugewiesen werden. Der Importvorgang ist in Algorithmus 1 durch Pseudocode beschrieben.

**Algorithmus 1** Auslesen der Modelldatei

1: **procedure** IMPORTOBJ(*path*, *unit*)
2: $lines \leftarrow$ zeilenweise eingelesene Modelldatei
3: $modelPart \leftarrow$ default $ModelPart$
4: $modelParts \leftarrow$ leere Liste
5: $vertices \leftarrow$ leere Liste

```
6:      for all l ∈ lines do
7:          columns ← Zeichen der aktuellen Zeile l
8:          if erstes Element in columns == o || g then
9:              modelPart ← neue Instanz der Klasse ModelPart
10:             modelParts ← modelPart der Liste hinzufügen
11:         end if
12:         if erstes Element in columns == v then
13:             vertex ← ausgelesener Koordinatenvektor
14:             vertex = vertex · unit
15:             vertices ← Koordinatenvektor hinzufügen
16:         end if
17:         if erstes Element in columns == f then
18:             modelPart.faces ← Polygonflächen der Liste hinzufügen
19:         end if
20:     end for
21:     if modelParts hat 0 Elemente then
22:         modelParts ← füge default-ModelPart hinzu
23:     end if
24:     return GetModelAsGameObject(modelParts, vertices)
25: end procedure
```

Die Methode *GetModelAsGameObject* legt für jede Instanz der Klasse *ModelPart* ein dazugehöriges *GameObject* an und fügt diesem eine *Mesh*-, *MeshFilter*- und *MeshRenderer*-Komponente hinzu. Der *Mesh*-Komponente werden die Knotenpunkte und Polygonflächen der Modelparts übergeben, womit die, in der OBJ-Datei beschriebene Geometrie, auf das *Mesh* übertragen wird. Jedem *Mesh* werden hierbei lediglich die von ihm benötigten Knotenpunkte, aus der *vertices*-Liste zugewiesen. Die Selektion der, für das *Mesh* benötigten, Punkte findet durch die Methode *RemapVertices* statt. Diese ermittelt mit Hilfe der in einem *ModelPart* gespeicherten Indizes der Polygonflächen (faces) und filtert die benötigten Koordinaten aus der Gesamtheit aller Knotenpunkte. Die Methode sorgt des Weiteren dafür, dass die Polygonflächen anhand der selektierten Knotenpunkte neu indiziert werden. Dieser Vorgang ergibt sich daraus, dass die selektierten Knotenpunkte in der gefilterten Liste nicht mehr die gleiche Position besitzen wie in der *vertices*-Liste. Die Methode *RemapVertices* ist in Algorithmus 2 dargestellt.

**Algorithmus 2** Mappen der selektierten Knotenpunkte

```
1: procedure RemapVertices(vertices)
2:     newOldIndexList ← neue Liste
3:     faces ←Indizes der Polygonflächen der ModelPart-Instanz
4:     for all f ∈ faces do
5:         if f ∉ newOldIndexList then
6:             newOldIndexList.Add(f)
```

```
7:          end if
8:      end for
9:      modelVertices ← leeres Array mit Länge der newOldIndexList
10:     for v ∈ modelVertices do
11:         modelVertices[i] = vertices[newOldIndexOf[i] − 1]
12:     end for
13:     for all f ∈ faces do
14:         faces[i] = newOldIndexList.IndexOf(faces[i])
15:     end for
16:     return modelVertices
17: end procedure
```

Listing 4 zeigt einen Auszug aus der OBJ-Datei des Modells der Hochschule Neubrandenburg (Haus 2). Dieses, im Rahmen des Projektes erstellte Modell, beinhaltet das Erdgeschoss der Hochschule sowie Teile der ersten, zweiten und dritten Etage. Das Modell wird im Rahmen der Arbeit zu Testzwecken verwendet.

Listing 4: Auszug aus der OBJ-Datei des 3D-Modells des Hauses 2.

```
1  # WaveFront *.obj file (generated by Autodesk ATF)
2
3  mtllib HS_Haus2_low.mtl
4
5  g Volumenkörper1
6
7  v 41614.068567 −12465.666567 1200.263127
8  ...
9  vt 0.000000 0.000000 0.000000
10 ...
11 vn −0.100000 −0.000000 −0.000000
12 ...
13 usemtl 191,191,191
14 f 1/1/1 2/2/2 4/3/3
15
16 g Volumenkörper1:1
```

Nach dem Import des Modells wird dieses in der Planungslösung dargestellt. Abbildung 41 zeigt das, in die Planungslösung importierte, 3D-Modell des Hauses 2.

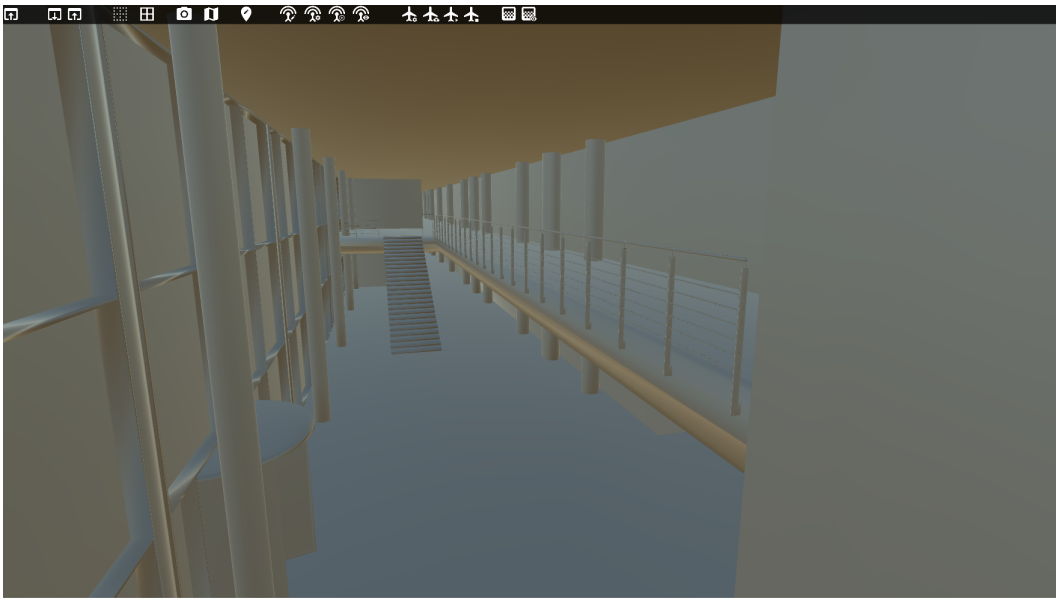


Abbildung 41: 3D-Modell des Foyers im Haus 2 der Hochschule Neubrandenburg, visualisiert in der Planungslösung.

## 5.2 Semi-physikalisches Funkausbreitungsmodell

Im Folgenden wird die praktische Umsetzung des in 4.4 Funkausbreitungsmodell theoretisch beschriebenen Funkausbreitungsmodells erläutert. Die Berechnung und Visualisierung der Funkausbreitung sowie der Import und die Verwaltung von Antennendiagrammen wird durch den *RadiationSimulationManager*, welcher als *GameObject* implementiert ist, umgesetzt (siehe Abbildung 42). Dem Manager sind zwei weitere Kind-GameObjects zugewiesen. Eines davon ist das *Raster*-GameObject, welches zur Umsetzung des dreidimensionalen Detektorzellen-Rasters dient. Des Weiteren hält der Manager das *Antennas*-GameObject. Dieses Spielobjekt beinhaltet Klassen zur Verwaltung der im 3D-Modell zu platzierenden Access Points. Zur Speicherung dieser verwendet der *AntennaManager* den *PersistanceManager*.

Die *RadiationSimulator*-Klasse dient zur Strahlenaussendung sowie zur Empfangsleistungsberechnung innerhalb des Detektorzellen-Rasters respektive des 3D-Modells. Ein Strahl wird hierbei durch die Klasse *ElectroMagneticWave* modelliert. Die Detektion einer Interaktion des Strahls mit einer Rasterzelle, wird durch die Klasse *ElectroMagneticHitReceiver* realisiert. Der Import von Antennendiagrammen findet durch die Klasse *AntennaCharacteristicManager* statt. Diese nutzt zur Abbildung eines Antennendiagramms die *AntennaCharacteristic*-Klasse. Für die Speicherung der importierten Antennendiagramme, wird der *PersistanceManager* verwendet.

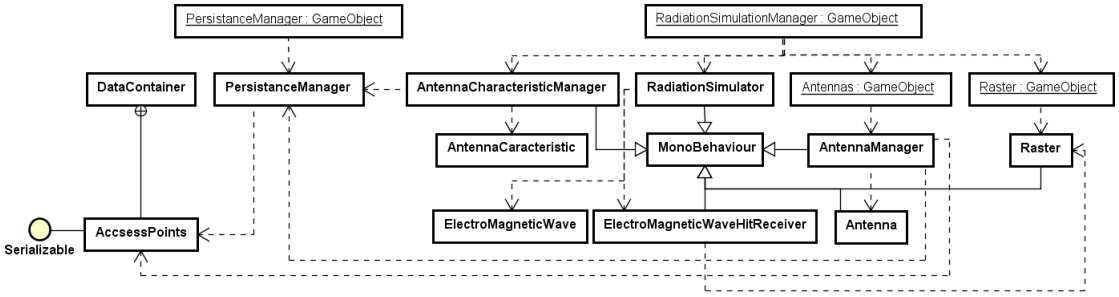



Abbildung 42: Klassenüberblick der *RadiationSimulationManager*-Komponente.

### 5.2.1 Verwaltung der Access Points

Die Verwaltung der Access Points wird durch den *AntennaManager* realisiert. Hierzu stellt der Manager Methoden zum hinzufügen (*AddAntenna*), selektieren (*SelectActiveAntenna*), deselektieren (*DeselectAntenna*) und löschen (*DeleteAntenna*) von AP's bereit. Innerhalb der Szene wird ein AP durch die Klasse *Antenna* repräsentiert, welche den Namen der ausgewählten Antennencharakteristik (*CharacteristicName*) sowie die Sendefrequenz ($F$ in GHz) als Attribut hält (siehe Abbildung 43).

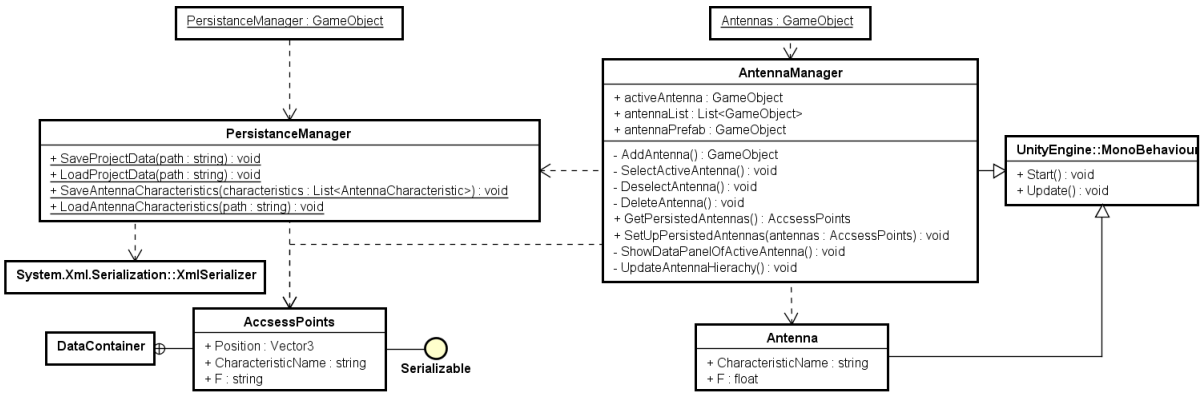



Abbildung 43: Vereinfachtes Klassendiagramm der Access Point-Verwaltung.

Will der Benutzer einen AP platzieren, erfolgt dies durch einen Doppelklick an der entsprechenden Position im 3D-Modell. Hierzu wird in der, von *MonoBehaviour* geerbten, *Update*-Methode der *AntennaManager*-Klasse auf die entsprechende Nutzereingabe geprüft. Erfolgt diese, wird die Methode *AddAntenna* aufgerufen und die erzeugte Antenne wird der Liste *antennaList* hinzugefügt. Die dreidimensionale Repräsentation des AP's innerhalb der Szene, wird durch das kugelförmige Prefab *antennaPrefab* der *AntennaManager*-Klasse realisiert (siehe Abbildung 44). Die Methode *UpdateAntennaHierachy* sorgt dafür, dass die AP-Liste (siehe Abbildung 36) mit den Einträgen des *antennaList*-Objektes synchronisiert wird.

Erfolgt die Selektion eines AP's durch den Benutzer über die linke Maustaste, ist dessen Referenz als *activeAntenna*-Attribut in der *AntennaManager*-Klasse hinterlegt und

die Methode *SelectActiveAntenna* wird aufgerufen. Diese bewirkt, dass der ausgewählte AP farblich hervorgehoben wird. Außerdem werden Pfeile zur Navigation aktiviert, welche dem Benutzer die Möglichkeit geben, den AP entlang der Koordinatenachsen zu verschieben (siehe Abbildung 44).

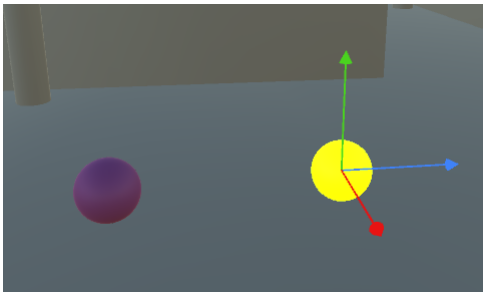


Abbildung 44: Access Point-Prefab deselektiert (links) und selektiert (rechts).

Durch die Selektion wird des Weiteren die Methode *ShowDataPanelOfActiveAntenna* aufgerufen, welche das Datenpanel des selektierten AP's anzeigt. Das Panel lässt den Benutzer die Antennencharakteristik und die Sendefrequenz einstellen. Weiter ist es dem Benutzer möglich, die exakten Koordinaten des AP's zu definieren (siehe Abbildung 45).

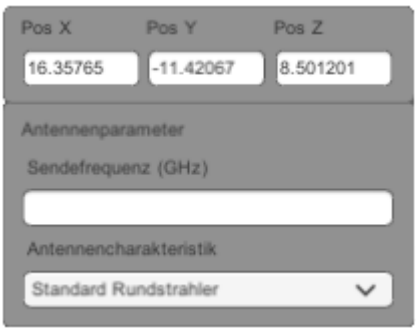



Abbildung 45: Datenpanel eines selektierten AP.

Zum Löschen eines AP's selektiert der Benutzer diesen zunächst und betätigt dann die Entfernen-Taste. Das Entfernen des AP-Spielobjekts aus der Szene erfolgt über die Methode *DeleteAntenna*. Diese löscht das entsprechende Element aus dem *antennaList*-Objekt und ruft wiederum die *UpdateAntennaHierachy*-Methode auf, welche die AP-Liste mit dem *antennaList*-Objekt synchronisiert.

Die persistente Speicherung der AP's erfolgt über die Datenklasse *AccessPoints*. Diese ist eine innere Klasse der Klasse *DataContainer* und beinhaltet die Position (*Position*), den Namen der Antennencharakteristik (*CharacteristicName*) sowie die Sendefrequenz (F in GHz) der zu speichernden AP's. Die Klasse erfüllt das *Serializable*-Interface und wird als Teil der Projektdatei durch die Methode *SaveProjectData* der *PersistanceManager*-Klasse im XML-Format serialisiert. Die für die Speicherung der AP's benötigten Informationen erhält der *PersistanceManager* von dem *AntennaManager*. Dieser stellt

die Informationen über die Methode *GetPersistedAntennas* bereit. Die Deserialisierung erfolgt über den Aufruf der Methode *LoadProjectData* im *PersistanceManager*. Dieser übergibt die, zur Initialisierung der AP-Prefabs in der Szene, notwendigen Informationen an den *AntennaManager*. Hierzu ruft der *PersistanceManager* die Methode *SetUpPersistedAntennas* auf und übergibt die Liste der deserialisierten AP's als Parameter (siehe Abbildung 46).

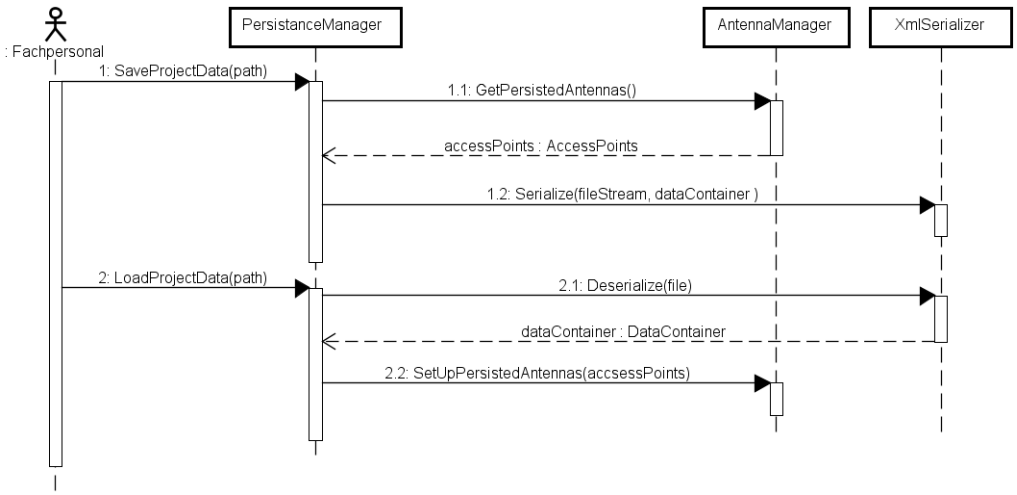



Abbildung 46: Sequenzdiagramm des Speicher- und Ladevorgangs für Access Points.

### 5.2.2 Import der Antennendiagramme

Der Import von Antennendiagrammen wird von der *AntennaCharacteristicManager*-Klasse umgesetzt. Diese ist Teil des *RadiationSimulationManager*-Spielobjekts und nutzt die Klasse *AntennaCharacteristic* zur Abbildung eines Antennendiagramms (siehe Abbildung 47).

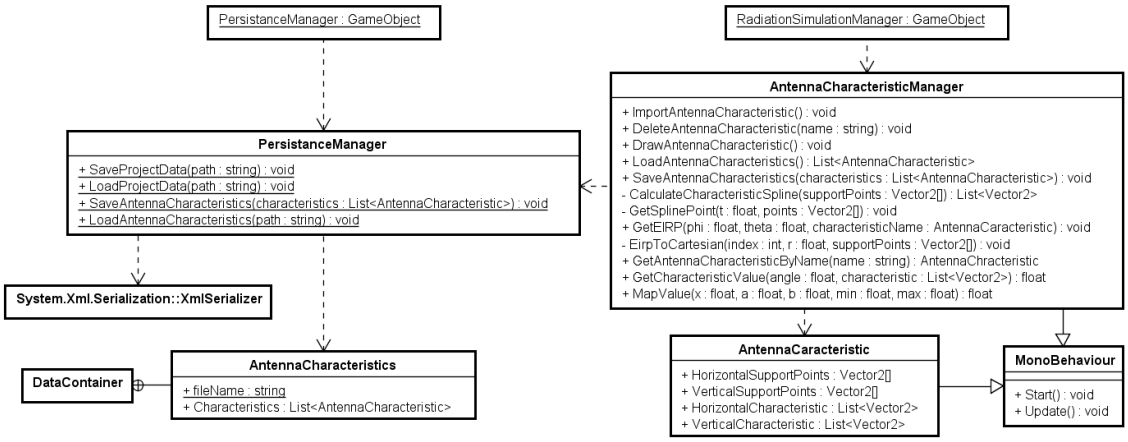



Abbildung 47: Vereinfachtes Klassendiagramm der Antennendiagrammverwaltung.

Der Import eines Antennendiagramms in der Horizontal- sowie in der Vertikalebene findet durch die Methode *ImportAntennaCharacteristic* statt. Diese liest eine CSV-formatierte Datei ein, welche in der ersten Spalte die Raumwinkel, in einer Winkelauflösung von 15° enthält. In der zweiten und dritten Spalte werden die, zu den Raumwinkeln gehörigen EIRP-Werte des horizontalen, beziehungsweise des vertikalen Antennendiagramms gespeichert. In Listing 5 wird ein Beispiel für eine Antennendiagramm-Datei gegeben.

Listing 5: Beispiel für ein importierbares Antennendiagramm im CSV-Format.

```
0;1.5;−20
15;1.5;−5.00
30;1.5;−6.00
45;1.5;−2.00
60;1.5;0.001
75;1.5;1.00
90;1.5;1.50
105;1.5;1.00
120;1.5;0.00
135;1.5;−2.00
150;1.5;−4.00
165;1.5;−9.00
180;1.5;−30
195;1.5;−9.00
210;1.5;−4.00
225;1.5;−2.00
240;1.5;5.00
255;1.5;1.00
270;1.5;3.50
285;1.5;1.00
300;1.5;0.00
315;1.5;−2.00
330;1.5;−4.00
345;1.5;−13.00
```

Um EIRP-Werte zu erhalten die zwischen den importierten Werten liegen, werden diese als Stützpunkte zur Interpolation genutzt. Diese wird durch die Methoden *CalculateCharacteristicSpline* und *GetSplinePoint* realisiert, welche eine Spline-Interpolation mit Hilfe der Stützpunkte durchführen. Zur Interpolation verwendet die *GetSplinePoint*-Methode sogenannte *Catmull-Rom-Splines* (siehe hierzu Catmull u. Rom 1974). Dabei werden die polaren Stützpunktkoordinaten (Winkel und EIRP), durch die Methode *EirpToCartesian* vom polaren in das kartesische Koordinatensystem überführt.

Die interpolierten Antennendiagramme werden im Diagrammverwaltungsfenster dargestellt (siehe Abbildung 48). Dieses dient dazu die Diagramme zu importieren, grafisch als Polardiagramm anzuzeigen und das Löschen von Antennendiagrammen zu

ermöglichen. Ein Diagramm ist projektübergreifend nutzbar und wird eindeutig durch dessen Namen identifiziert. Dieser wird beim Import festgelegt und kann nur einmalig vom Benutzer vergeben werden.

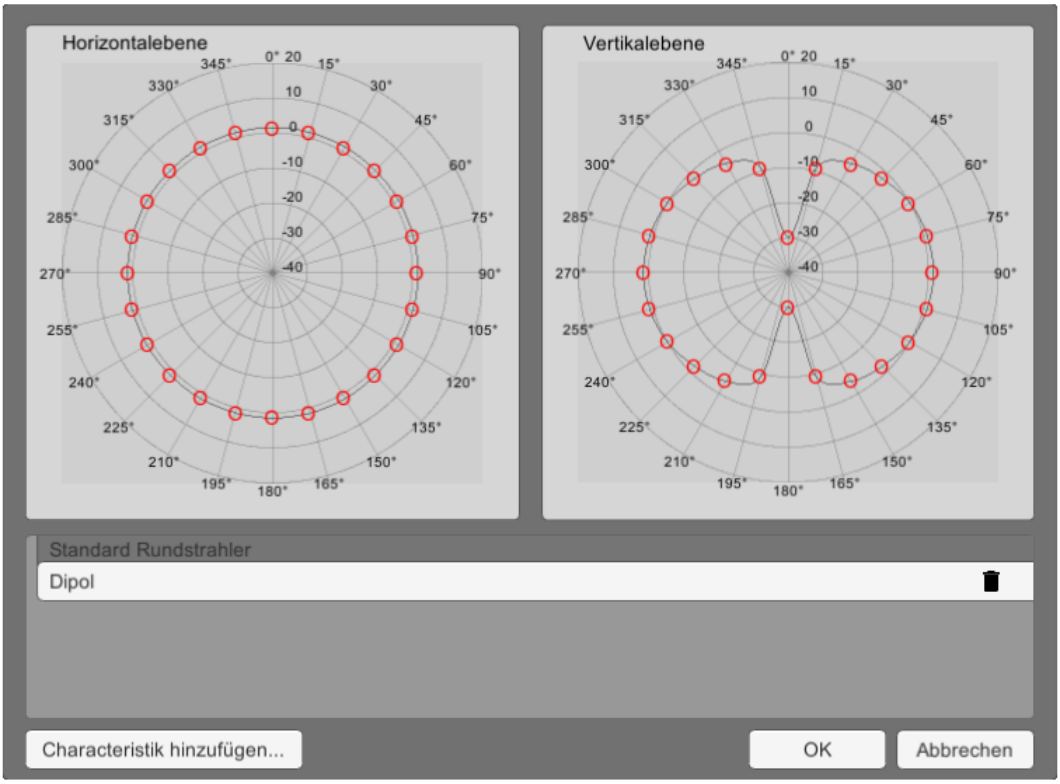



Abbildung 48: Fenster zur Verwaltung von Antennendiagrammen.

Zur Programmlaufzeit werden die Diagramme durch die Klasse *AntennaCharacteristic* repräsentiert. Diese speichert die horizontalen und vertikalen Stützpunkte (*HorizontalSupportPoints* und *VerticalSupportPoints*) sowie die, durch die Spline-Interpolation berechneten EIRP-Werte (*HorizontalCharacteristic* und *VerticalCharacteristic*).

Die Speicherung der Antennendiagramme erfolgt durch den *PersistanceManager*. Hierzu wird die Klasse *AntennaCharacteristics* genutzt, welche eine innere Klasse der *DataContainer*-Klasse ist. Das Speichern und Laden der Antennendiagramme wird durch die, im *PersistanceManager* befindlichen, Methoden *SaveAntennaCharacteristic* und *LoadAntennaCharacteristic* realisiert. Diese werden in den *Save*- und *LoadAntennaCharacteristics*-Methoden der *AntennaCharacteristicManager*-Klasse aufgerufen. Gespeichert werden die importierten Diagramme als Liste des Typs *AntennaCharacteristic* im XML-Format.

### 5.2.3 Berechnung und Visualisierung der Funkausbreitung

Die konkrete Implementierung des semi-physikalischen Funkausbreitungsmodells wird hauptsächlich in der Klasse *RadiationSimulator*, welche ein *Component* des *RadiationS-*

*imulationManagers* ist, umgesetzt (siehe Abbildung 49). Zur Modellierung einer elektromagnetischen Welle nutzt diese die Klasse *ElectroMagneticWave*, welche die effektive Sendeleistung (EIRP), den Signallaufweg ($D$), die Koordinaten des letzten Interaktionspunktes (IP) mit dem 3D-Modell, die Richtung des Strahls sowie die Anzahl von Interaktionen mit dem 3D-Modell, als Attribute vorhält.

Das in 4.4 Funkausbreitungsmodell beschriebene dreidimensionale Detektorzellenraster wird durch die Klasse *Raster* umgesetzt. Diese nutzt zur Rastererstellung das *Cell*-Prefab, welchem eine *BoxCollider*-Component zugewiesen ist, um so eine Strahlendetektion, im Rahmen des Ray Castings (siehe 3.2.3 Physiksimulation) zu ermöglichen. Zur Indikation des RSL besitzt das Prefab eine *PointLight*-Component (siehe 3.2.2 Lichtquellen). Dem *Cell*-Prefab ist weiter die *ElectroMagneticHitReceiver*-Component zugeordnet, welche das Reagieren auf einen Strahlendurchlauf durch eine Detektorzelle ermöglicht.

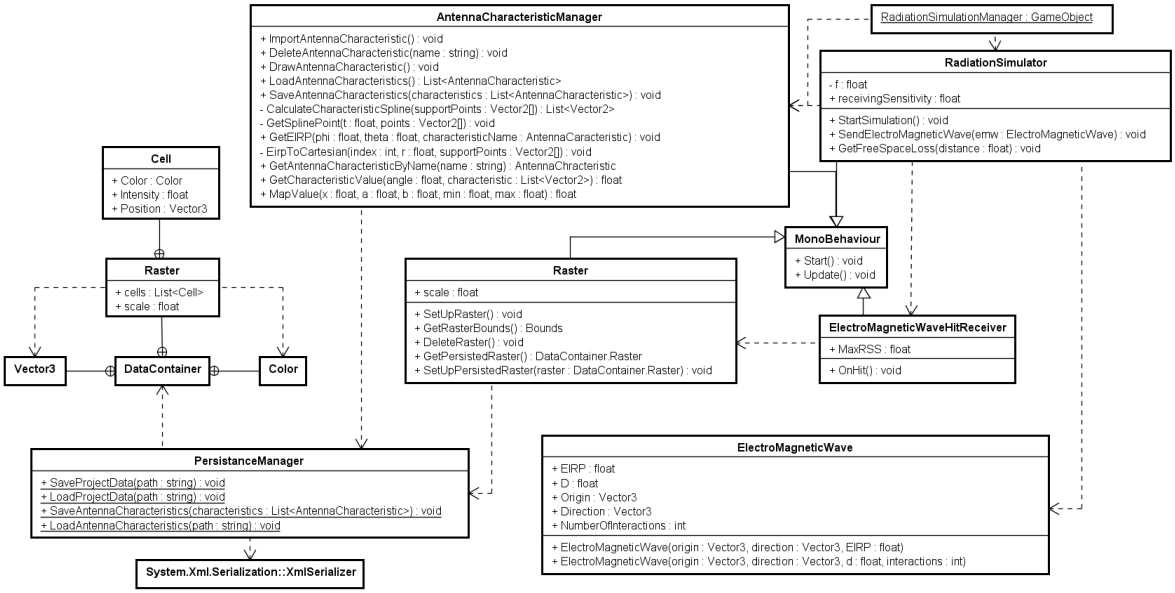



Abbildung 49: Vereinfachtes Klassendiagramm zur Berechnung der Funkausbreitung.

Gestartet wird die Berechnung der Funkausbreitung durch das Fachpersonal, indem die Methode *StartSimulation* über die grafische Benutzeroberfläche aufgerufen wird. Ist noch kein Raster erzeugt worden, wird dieses durch die Methode *SetUpRaster* erstellt. Besteht bereits ein Raster durch eine vorherige Berechnung, wird dieses zunächst gelöscht. Nach dem Erzeugen des Detektorzellenrasters, werden die Strahlen ausgesendet, welche die elektromagnetischen Wellen, im Rahmen der Modellierung approximieren. Dies geschieht für jeden platzierten AP im 3D-Modell, wobei das zugewiesene Antennendiagramm berücksichtigt wird (siehe Abbildung 50).

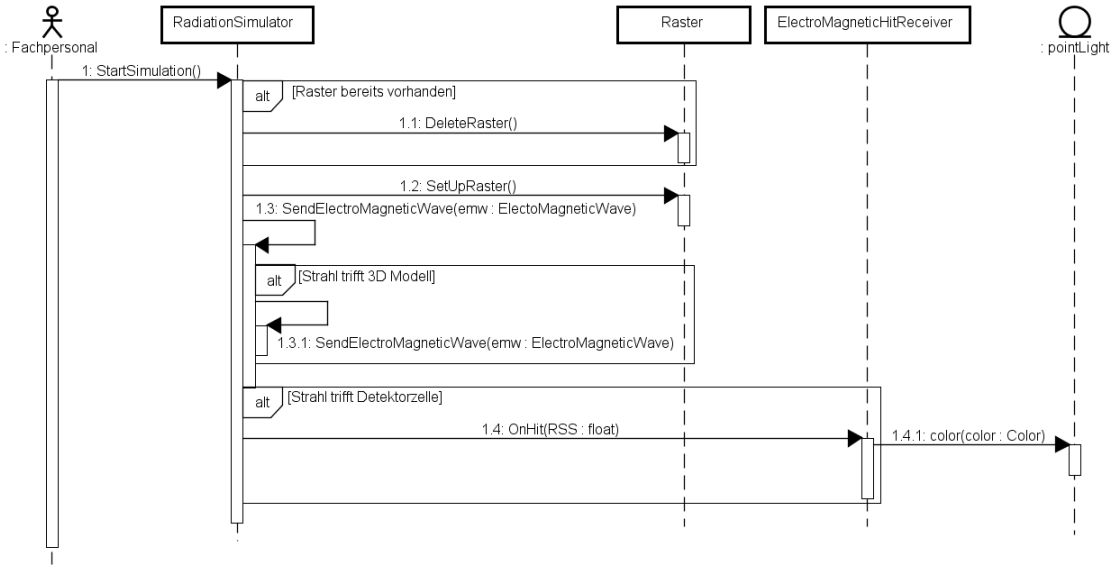



Abbildung 50: Sequenzdiagramm der Funkausbreitungsberechnung.

Die Modellierung des Antennendiagramms ist in Algorithmus 3 beschrieben. Zunächst werden nur die Access Points für die Funkausbreitungsberechnung betrachtet, denen eine valide Sendefrequenz und ein gültiges Antennendiagramm zugewiesen wurde. Die gültigen AP's sind in der Liste *validAntennas* zu finden. Sind Parameter einer oder mehrerer Antennen nicht gültig, so wird dem Benutzer eine entsprechende Fehlermeldung angezeigt.

Für alle gültigen APs werden die EIRP-Werte der, durch die Winkelauflösung *res* festgelegten, Raumrichtungen bestimmt. Hierzu wird zunächst der Vektor *normDir*, um *theta* um die x-Achse (siehe Algorithmus 3 Zeile 8) und danach um *phi* Grad um die y-Achse rotiert (siehe Algorithmus 3 Zeile 10). Anschließend wird der EIRP für die entsprechenden Werte für *phi* und *theta* bestimmt. Hierzu wird die Methode *GetEIRP*, des *AntennaCharacteristicManagers* verwendet. Diese berechnet den EIRP mit Hilfe der Formeln 22 bis 24. Ist der EIRP-Wert für die aktuelle Raumrichtung bestimmt, wird ein neues Objekt der Klasse *ElectroMagneticWave* erzeugt. Diesem wird die Position des Access Points (*aPos*), die initiale Raumrichtung für den Strahlenabschuss (*normDir*) sowie der EIRP des auszusendenden Strahls übergeben (siehe Algorithmus 3 Zeile 13). Abschließend erfolgt die Übergabe des *ElectroMagneticWave*-Objekts an die Methode *SendElectroMagneticWave* (siehe Algorithmus 3 Zeile 14).

**Algorithmus 3** Modellierung des Antennendiagramms in Unity

1: **procedure** StartSimulation
2: $validAntennas \leftarrow$ alle gültigen Antennen
3: $acm \leftarrow AntennaCharacteristicManager$
4: $res \leftarrow$ Winkelauflösung in Grad
5: $normDir \leftarrow$ normierter Vektor parallel zur y-Achse (0,1,0)

```
6:     for all a ∈ validAntennas do
7:         for theta = −180; theta < 180; theta += res do
8:             normDir wird um theta Grad um die x-Achse rotiert
9:             for phi = −180; phi < 180; phi += res do
10:                normDir wird um phi Grad um die y-Achse rotiert
11:                EIRP ← acm.GetEIRP(phi, theta, a.CharacteristicName)
12:                aPos ← a.transform.position
13:                emw ← new ElectroMagneticWave(aPos, normDir, EIRP)
14:                SendElectroMagneticWave(emw)
15:            end for
16:        end for
17:    end for
18: end procedure
```

Abbildung 51 zeigt das dreidimensionale Antennendiagramm eines Access Points, dargestellt in der Unity-Szene. Die Länge der Strahlen stellt hierbei den jeweiligen EIRP-Wert dar. Aufgrund der Übersichtlichkeit werden lediglich die Werte des horizontalen und vertikalen Diagramms in der Abbildung dargestellt.

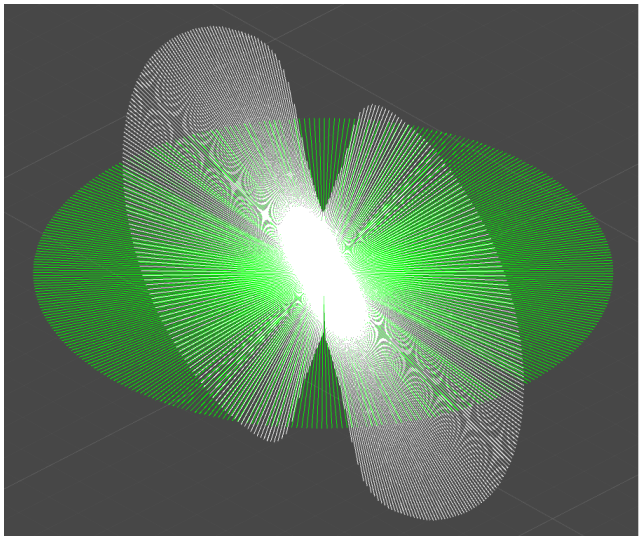


Abbildung 51: Antennendiagramm aus Abbildung 48, in der Unity-Szene.

Die in Algorithmus 4 beschriebene Methode *SendElectroMagneticWave* dient dazu, Strahlen (Rays) von einem koordinatenmäßig definierten Ursprung in eine bestimmte Raumrichtung auszusenden und Kollisionen mit dem 3D-Modell oder einer Detektorzelle festzustellen. Hierzu wird ein Ray erzeugt und von der Sendeposition (*emw.Origin*) in die Wellenausbreitungsrichtung (*emw.Direction*) gesendet. Findet eine Kollision des Strahls mit dem 3D-Modell statt, erfolgt dessen Reflexion anhand der Flächennormalen, an der Stelle des Auftreffens (Interaktionspunkt). Die hierbei ermittelte Senderichtung *reflectionDir* (siehe Algorithmus 4 Zeile 10) sowie die Koordinaten des IP werden an den

nächsten zu erzeugenden Strahl übergeben. Weiter wird die Teilstrecke zwischen dem letzten und aktuellen IP (*dPartial*) zugewiesen und die Anzahl der Interaktionen inkrementiert (siehe Algorithmus 4 Zeile 11). Nach dem Erzeugen des reflektierten Strahls erfolgt der rekursive Aufruf der *SendElectroMagneticWave*-Methode (siehe Algorithmus 4 Zeile 12). Die Rekursion beziehungsweise die Reflexion des Strahls wird unter den Bedingungen abgebrochen, dass die maximale Anzahl von Interaktionen *maxInteractions* überschritten, oder die minimale Empfängerempfindlichkeit *receivingSensitivity* unterschritten wird.

Trifft ein Strahl auf eine Detektorzelle, so wird die Distanz vom letzten Interaktionspunkt zum Durchlaufpunkt (DP) der getroffenen Detektorzelle ermittelt und auf den bereits zurückgelegten Signallaufweg *emw.D* addiert (siehe Algorithmus 4 Zeile 14). Die dabei resultierende Entfernung dient als Grundlage der Freiraumdämpfungsberechnung nach Formel 25. Die Berechnung der Freiraumdämpfung erfolgt durch die Methode *GetFreeSpaceLoss* der *RadiationSimulatior*-Klasse. Aufgrund der überwiegend aus Stahl bestehenden Umgebung, werden die in Formel 25 angebrachten Dämpfungsfaktoren bei der Berechnung der Freiraumdämpfung vernachlässigt. Die endgültige Empfangsleistung $RSL_{xyz}$ ergibt sich nach Formel 21 durch die Subtraktion der Freiraumdämpfung $FSL$ vom $EIRP$ des anfänglich ausgesendeten Strahls *emw.EIRP* (siehe Algorithmus 4 Zeile 16). Ist der berechnete EIRP-Wert größer als die Empfängerempfindlichkeit, wird dieser zur Visualisierung an das *WaveHitReceiver*-Objekt des getroffenen Detektorzellen-Spielobjekts, durch die Methode *OnHit*, übergeben (siehe Algorithmus 4 Zeile 21).

**Algorithmus 4** Rekursives Aussenden von elektromagnetischen Wellen

1: **procedure** SendElectroMagneticWave(emw)
2: $receivingSensitivity \leftarrow$ Empfaengerempfindlichkeit
3: $maxInteractions \leftarrow$ maximale Anzahl von Interaktionen
4: **if** $emw$.NumberOfInteractions $> maxInteractions$ **then**
5: **return**
6: **end if**
7: $ray \leftarrow$ erzeuge Strahl von $emw$.Origin in Richtung $emw$.Direction
8: $hits \leftarrow$ Liste aller Spielobjekte mit einem Collider die von $ray$ getroffen werden
9: **for all** Spielobjekte $\in hits$ **do**
10: **if** $ray$ trifft Modell **then**
11: $dPartial \leftarrow$ Distanz zwischen dem letzten $IP$ und aktuellem $IP$
12: $reflectionDir \leftarrow$ normierte Reflexionsrichtung
13: $reflectedWave \leftarrow$ new ElectroMagneticWave($hitPoint$, $reflectionDir$, $emw$.D + $dPartial$, $emw$.NumberOfInteractions + 1)
14: SendElectroMagneticWave($reflectedWave$)
15: **break**
16: **else if** $ray$ trifft Detektorzelle **then**
17: $distanceToCell \leftarrow$ Distanz des letzten $IP$'s zum aktuellen $DP$

```
18:             FSL ← Dämpfung des Signallaufwegs d (emw.D + distanceToCell)
19:             RSL ← emw.EIRP −FSL
20:             if RSL < receivingSensitivity then
21:                 return
22:             end if
23:             cell ← vom Strahl getroffene Detektorzelle
24:             cell.OnHit(RSL)
25:         end if
26:     end for
27: end procedure
```

Die *OnHit*-Methode setzt die Intensität der Lichtquelle auf den empirisch ermittelten Wert von 0.005 (siehe Algorithmus 5 Zeile 4) und prüft, ob der übergebene $RSL$ größer ist, als der bisher an dieser Detektorzelle ermittelte maximale RSL $MaxRSL$ (siehe Algorithmus 5 Zeile 5). Trifft dies zu, wird der übergebene RSL zum neuen maximalen RSL (siehe Algorithmus 5 Zeile 6). Anschließend wird der Lichtquelle in Abhängigkeit des RSL ein RGB-Farbwert zur Indikation der Empfangsstärke zugewiesen (siehe Algorithmus 5 Zeile 7 bis 12).

**Algorithmus 5** Funktion zur Visualisierung der Empfangsstärke

```
1: procedure OnHit(RSL)
2:     MaxRSL ← maximaler an der Detektorzelle gemessener RSL
3:     pointLight ← Lichtkomponente der Detektorzelle
4:     pointLight.intensity = 0.005
5:     if RSL > MaxRSL then
6:         MaxRSL = RSL
7:         if RSL > −50 then
8:             pointLight.color = Grün
9:         else if RSL < −50 & RSL ≥ −70 then
10:            pointLight.color = Blau
11:        else if RSL < −70 & RSL ≥ receivingSensitivity then
12:            pointLight.color = Rot
13:        end if
14:    end if
15: end procedure
```

Die Klassifizierung der RSL-Werte sowie die dazugehörigen Farbwerte der Lichtindikatoren können der Tabelle 7 entnommen werden.

Tabelle 7: Klassifizierte Empfangsstärken und deren Farbwerte im Rahmen der Visualisierung.

| **Empfangsstärke** | **RSL [dBm]** | **Farbe ([R,G,B])** |
|---|---|---|
| sehr gut | größer -50 | Grün (0,255,0) |
| mittel | -50 bis -70 | Blau (0,0,255) |
| schlecht | größer -70 bis -90 | Rot (255,0,0) |

Das Ergebnis einer Funkausleuchtungsberechnung im Foyer des Hauses 2 der Hochschule Neubrandenburg wird in Abbildung 52 dargestellt. Die Berechnung bezieht hierbei zwei AP's mit einer Sendefrequenz von 2,4 GHz ein, wobei das in Abbildung 51 dargestellte Antennendiagramm verwendet wird.

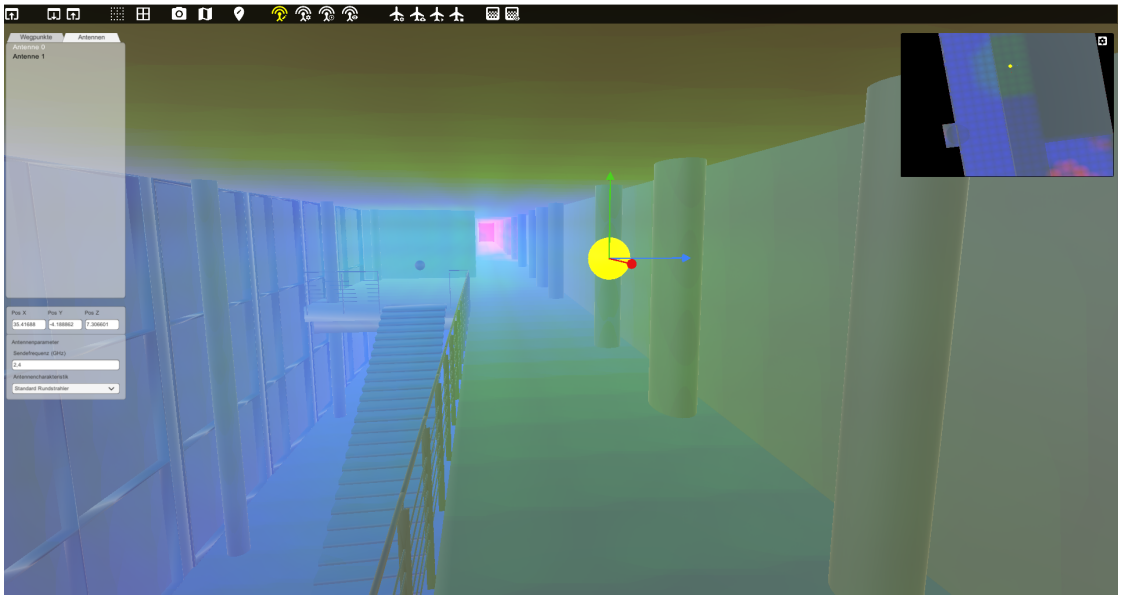



Abbildung 52: Visualisierung der Funkausbreitung in der Planungslösung.

## 5.3 Missionsmanagement

Das Missionsmanagement der Planungslösung umfasst die Flugcontrollerverwaltung, die Routen- und Aktionsplanung und den Export der geplanten Route in einer Missionsdatei. Weiter werden Funktionalitäten zum Starten und Stoppen der Mission sowie die Visualisierung des, von der Drohne zurückgelegten, Flugweges bereitgestellt. Abbildung 53 gibt einen generalisierten Überblick bezüglich der Klassenstruktur des Missionsmanagements.

Zur Umsetzung verschiedener Flugcontrollerschnittstellen wird der *ControllerProfile-Manager* verwendet, welcher die entwicklerseitig implementierten Controllerprofile verwaltet. Ein Profil wird durch die Klasse *ControllerProfile* abgebildet. Teil der *ControllerProfile*-Klasse ist die abstrakte *MissionData*-Klasse, welche die Elternklasse für die

Missionsdatenklasse darstellt. Diese dient zur Implementierung der missionsrelevanten Datenklassen, welche Missionsparameter, Wegpunkte, Aktionen und Aktionsparameter (*WaypointData* und *WayPointInitData*) vorhalten.

Die Verwaltung von Wegpunkten und Wegpunktaktionen wird durch die *RouteManager*-Klasse implementiert. Die Klasse ist außerdem für die Visualisierung der geplanten Route zuständig und stellt die Missionsdaten für die Klasse *WayPointData* zur Verfügung.

Der Missionsdatenexport, das Starten und Stoppen einer Wegpunktmission wird durch den *MissionControlManager* realisiert. Des Weiteren implementiert die Klasse die Flugwegvisualisierung. Hierzu dient die Klasse *MissionServer*, welche Positionsdaten der Drohne während der Missionsdurchführung erhält und zur Umsetzung der *ProcessMissionData*-Schnittstelle dient. Die Schnittstelle wird zum Empfang von Missionsdaten und Missionsstatus-Updates genutzt.

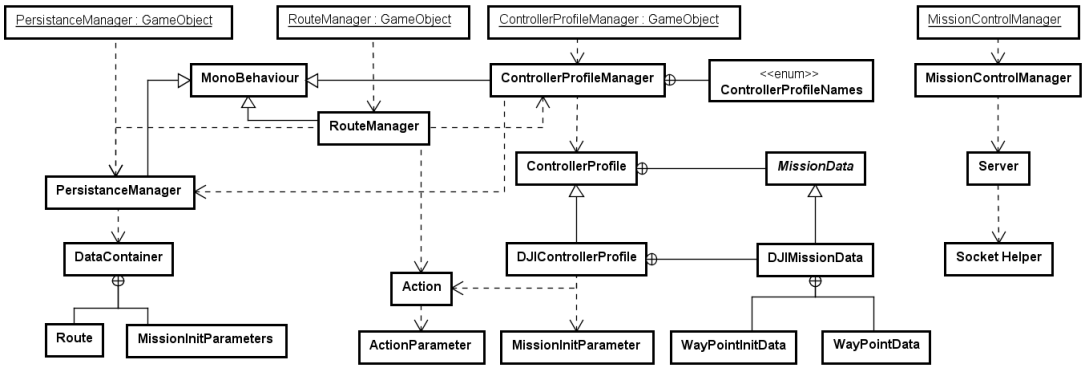



Abbildung 53: Generalisierte Übersicht der Klassen zum Missionsmanagement.

### 5.3.1 Controllerverwaltung

Die Planungslösung implementiert die Datenspezifikation des *DJI*-Flugcontrollers durch die Klasse *DJIControllerPofile*. Diese erbt Accessor-Methoden für den Zugriff auf die Missionsdaten (*Data* und *Actions*) sowie den Namen des Controllerprofils (*Name*), von der generalisierten Klasse *ControllerProfile*, welche die Elternklasse für alle zu implementierenden Controllerprofile darstellt. Die innere abstrakte Klasse *Missiondata* vererbt die abstrakte *ParseMissionData*-Methode, welche von der *DJIMissionData*-Klasse implementiert wird und der Bereitstellung der Datenklassen *WayPointData* und *WayPointInitData* dient (siehe Abbildung 54).

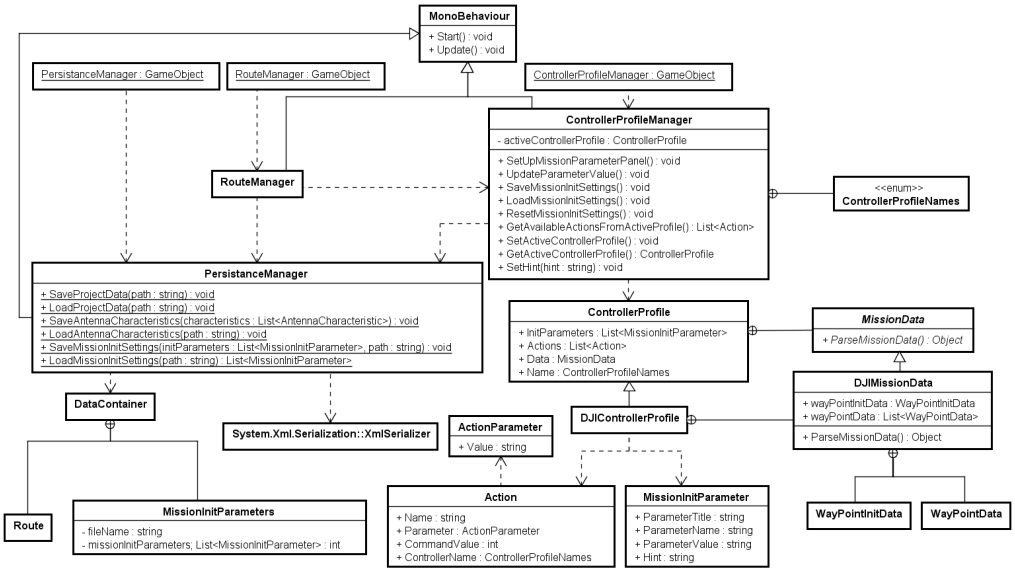



Abbildung 54: Vereinfachtes Klassendiagramm zur Verwaltung von Controllerprofilen.

Die in Abbildung 55 dargestellten Datenklassen stellen die von der OSDK-API verlangten Missionsdaten (siehe Abbildung 6 und 7) bereit. Wird eine Mission gespeichert, befüllt die *ParseMissionData*-Methode die Attribute *waypointData* und *waypointInitData* mit den vom Benutzer angegebenen Wegpunkten und Missionsparametern. Über welche Missionsparameter und Aktionen ein Controllerprofil verfügt, wird durch das entwicklerseitige Anlegen von *Action*- und *MissionInitParameter*-Objekten im Konstruktor des Controllerprofils festgelegt. Die maximale Anzahl von Wegpunkten respektive Wegpunktaktionen, wird durch die Attribute *MAX_ACTIONS* und *MAX_WAYPOINTS* festgelegt.

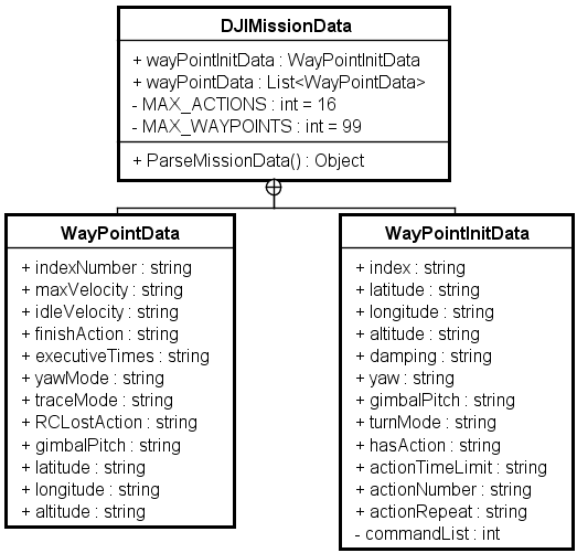



Abbildung 55: Schnittstelle für den Missionsdaten-Upload.

In der Planungslösung selbst findet die Eingabe der Missionsparameter über ein entsprechendes Panel statt (siehe Abbildung 56). Programmintern wird ein Missionsparameter durch die Klasse *MissionInitParameter* repräsentiert. Diese speichert den vom *DJI*-OSDK verwendeten Parameternamen (*ParameterName*), den im Panel angezeigten Parametertitel (*ParameterTitel*), den Wert des Parameters (*ParameterValue*) sowie eine Parameterbeschreibung (*Hint*). Das Panel selbst wird durch die *SetUpMissionParameterPanel*-Methode der *ControllerProfileManager*-Klasse aufgebaut und zeigt das aktive Controllerprofiel in einer Dropdown-Liste an, welches in der Klasse als Attribut hinterlegt wird (*activeControllerProfile*).

Unter der Dropdown-Liste befinden sich die dynamisch erzeugten Eingabefelder zur Bearbeitung der Missionsparameter. Wird eines der Eingabefelder fokussiert, aktualisiert sich die Parameterbeschreibung im unteren Teil des Panels, was durch die *SetHint*-Methode der *ControllerProfileManager*-Klasse realisiert wird.

Durch den Button *Standardwerte laden*, werden voreingestellte Werte für die Missionsparameter gesetzt. Das Laden dieser Werte erfolgt über die Methode *ResetMissionSettings*, welche in der *ControllerProfileManager*-Klasse zu finden ist. Die Standardwerte werden beim Anlegen der *MissionInitParameter*-Objekte festgelegt. Die Objekterzeugung findet im Konstruktor der *DJIControllerProfile*-Klasse statt, wobei dem *MissionInitParameter*-Konstruktor die Werte, als *ParameterValue*-Attribut übergeben werden.

Die Übernahme der Missionsparameter erfolgt durch das Betätigen der Schaltfläche *OK*, welche die Methode *SaveMissionInitSettings* aufruft. Diese wiederum ruft die gleichnamige Methode der *PersistanceManager*-Klasse auf, welche das Controllerprofil als XML-Datei unter dem, im Controllerprofil angegebenen Pfad (*FilePath*), abspeichert.


Abbildung 56: Fenster zur Auswahl des Controllerprofils und zur Angabe der dazugehörigen Missionsparameter.

### 5.3.2 Routenplanung

Die Flugroutenplanung der Drohne setzt sich aus dem Platzieren von Wegpunkten und der Vergabe von Wegpunktaktionen zusammen, wobei die verfügbaren Aktionen von dem verwendeten Controllerprofil bestimmt werden. Implementiert wird die Routenplanung durch die *RouteManager*-Klasse, welche ein Component des gleichnamigen Spielobjektes darstellt. Die Klasse enthält Funktionen zum Hinzufügen (*AddWayPoint*), Löschen (*DeleteWayPoint*) und Selektieren (*SelectActiveWayPoint*) von Wegpunkten (siehe Abbildung 57).

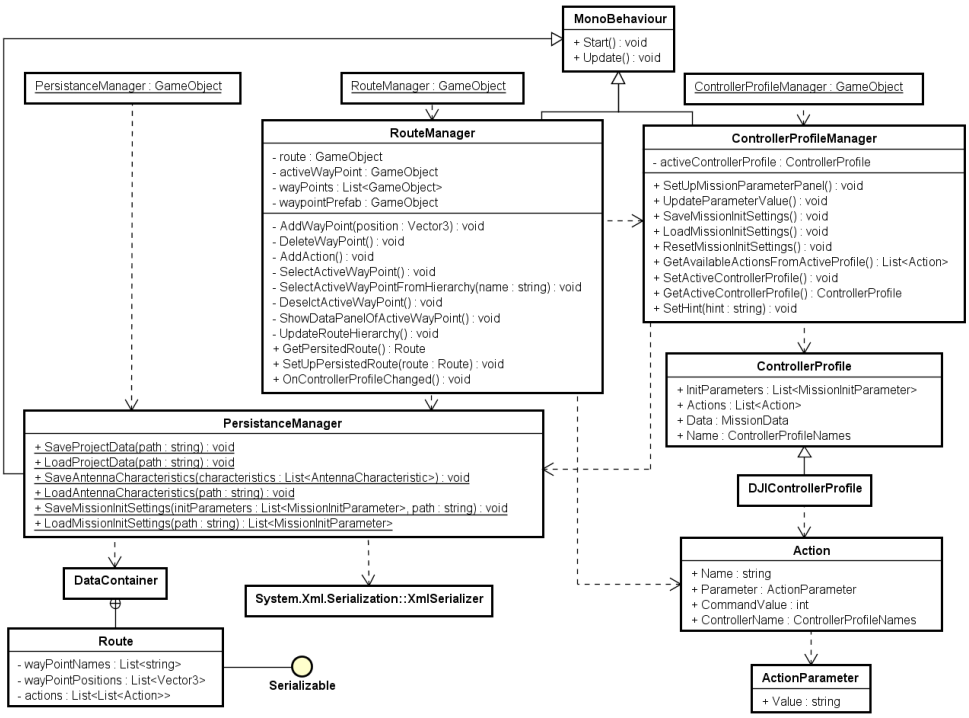



Abbildung 57: Vereinfachtes Klassendiagramm der Routenplanung.

Das Platzieren eines Wegpunktes, erfolgt durch einen Doppelklick an der entsprechenden Position innerhalb des 3D-Modells. Hierzu wird in der, von *MonoBehaviour* geerbten, *Update*-Methode der *RouteManager*-Klasse, auf die entsprechende Nutzereingabe geprüft. Wird diese festgestellt, erfolgt der Aufruf der *AddWayPoint*-Methode, welche den erzeugten Wegpunkt der Liste *wayPoints* hinzufügt. Ein Wegpunkt wird innerhalb der Szene, durch das kugelförmige Prefab *waypointPrefab* der *RouteManager*-Klasse repräsentiert. Die Flugroute der Drohne wird durch Linien zwischen den Wegpunkten dargestellt, wobei ein dreidimensionaler Pfeil als Indikator für die Flugrichtung dient. Die *UpdateRouteHierarchy*-Methode sorgt dafür, dass die Wegpunkt-Liste (siehe Abbildung 36) mit den Einträgen des *wayPoints*-Objektes synchronisiert wird.

Wird ein Wegpunkt vom Benutzer durch die linke Maustaste selektiert, ist dessen Referenz als *activeWayPoint*-Attribut in der *RouteManager*-Klasse hinterlegt und die

Methode *SelectActiveWayPoint* wird aufgerufen. Diese bewirkt, dass der ausgewählte Wegpunkt farblich hervorgehoben wird. Außerdem werden Pfeile zur Navigation aktiviert, welche dem Benutzer die Möglichkeit geben, den Wegpunkt entlang der Koordinatenachsen zu verschieben (siehe Abbildung 58).

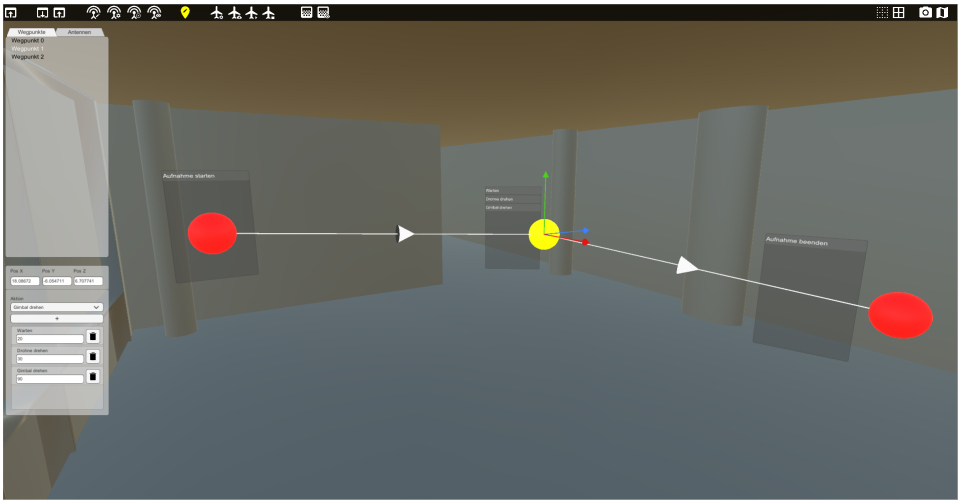



Abbildung 58: Visualisierung der Flugroute in der Planungslösung.

Durch die Selektion wird außerdem die Methode *ShowDataPanelOfActiveWayPoint* aufgerufen, welche das Datenpanel des selektierten Wegpunktes anzeigt. Unter Verwendung des Datenpanels kann eine exakte Positionierung des Wegpunktes vorgenommen werden. Außerdem wird die Vergabe von Wegpunktaktionen durch das Panel ermöglicht (siehe Abbildung 59).


Abbildung 59: Datenpanel eines selektierten Wegpunktes.

Der Benutzer wählt hierzu aus der Dropdown-Liste des Datenpanels eine Wegpunktaktion aus und fügt diese, unter Verwendung des Plus-Buttons, der Aktionsliste des selektierten Wegpunktes hinzu. Das Hinzufügen einer Aktion wird programmtechnisch durch die Methode *AddAction* implementiert. Erwartet die Aktion eine Parametereingabe, wird ein editierbares Eingabefeld unter dem Aktionsnamen erzeugt. Wird kein

Parameter erwartet ist das Eingabefeld nicht editierbar, was durch eine Graufärbung des Feldes ersichtlich wird. Ändert der Benutzer das Controllerprofil, wird die Methode *OnControllerProfileChanged* aufgerufen, was eine Aktualisierung der verfügbaren Wegpunktauktionen zur Folge hat, wobei der *ControllerProfileManager* die Controllerspezifischen Wegpunktaktionen, durch die Methode *GetAvailableActionsFromActiveControllerProfile*, an die *RouteManager*-Klasse übergibt.

Soll ein Wegpunkt gelöscht werden, wird dieser zunächst vom Benutzer selektiert und über das Betätigen der Entfernen-Taste gelöscht. Das Entfernen des Wegpunkt-Spielobjekts aus der Szene erfolgt über die Methode *DeleteWayPoint*. Die Methode löscht dabei das Spielobjekt aus der Wegpunktliste (*wayPoints*) und ruft die *UpdateRouteHierachy*-Methode auf, welche die grafische Wegpunkt-Liste mit der programminternen *wayPoints*-Liste synchronisiert.

Zur persistenten Speicherung der Wegpunkte wird die Datenklasse *Route* genutzt. Diese innere Klasse ist Bestandteil der Klasse *DataContainer* und beinhaltet die Namen der Wegpunkte (*wayPointNames*), die Positionen der Wegpunkte (*wayPointPositions*) sowie die Wegpunktaktionen (*actions*) der zu speichernden Flugroute. Die Klasse erfüllt das *Serializable*-Interface und wird als Teil der Projektdatei durch die Methode *SaveProjectData* der *PersistanceManager*-Klasse im XML-Format serialisiert.

Die für die Speicherung der Route benötigten Informationen erhält der *PersistanceManager* von der *RouteManager*-Klasse. Dieser stellt die Informationen über die Methode *GetPersistedRoute* bereit. Die Deserialisierung erfolgt über den Aufruf der Methode *LoadProjectData* im *PersistanceManager*. Dieser übergibt die, zur Initialisierung der Wegpunkt-Prefabs innerhalb der Szene, notwendigen Informationen an den *RouteManager*, indem der *PersistanceManager* die Methode *SetUpPersistedRoute* aufruft und eine Liste der deserialisierten Wegpunkte als Parameter übergibt (siehe Abbildung 60).

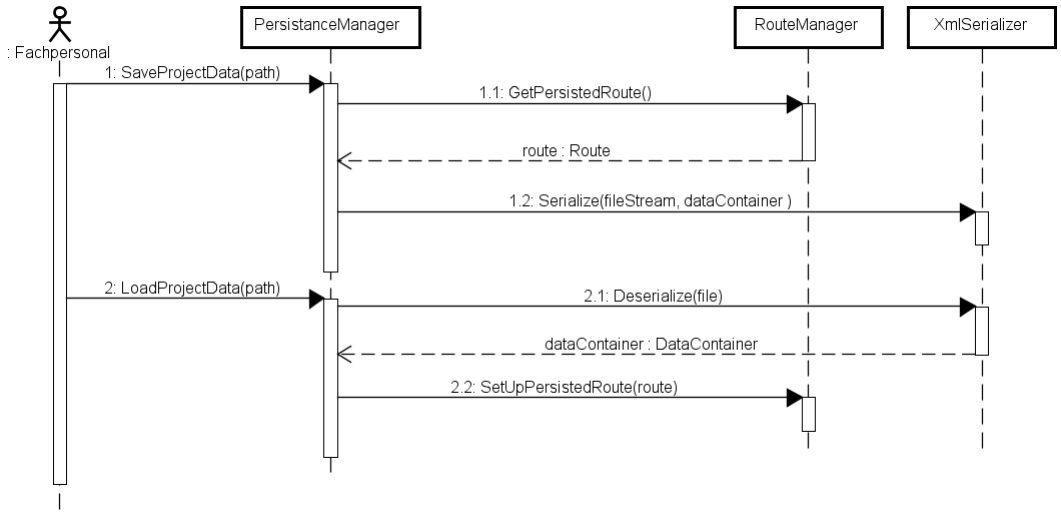



Abbildung 60: Sequenzdiagramm des Speicher- und Ladevorgangs der Route.

### 5.3.3 Missionsdurchführung und Visualisierung

Das Exportieren der Missionsdatei, das Starten und Stoppen der Wegpunktmission sowie die Missionsvisualisierung ist Aufgabe der *MissionControlManager*-Klasse. Diese erzeugt eine Instanz der *MissionServer*-Klasse, welche zusammen mit dem *SocketHelper* dazu dient, Daten zwischen der Planungslösung und der Indoor-Navigationslösung auszutauschen (siehe Abbildung 61).

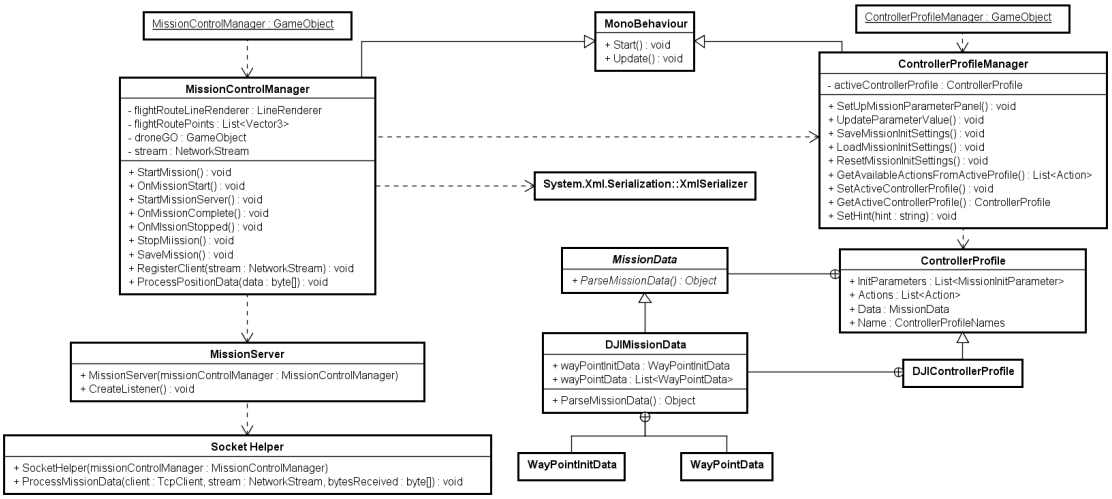



Abbildung 61: Vereinfachtes Klassendiagramm zur Durchführung des Missionsmanagements

Der Export der Missionsdatei wird vom Benutzer durch die Ausführung der *SaveMissionData*-Methode gestartet. Diese erhält von der *MissionControlManager*-Klasse die Referenz auf das aktive Controllerprofil (*activeControllerProfile*), worüber auch der Zugriff auf das Missionsdaten-Objekt (*missionData*) erfolgt. Unter Verwendung der Methode *ParseMissionData* liefert dieses Objekt, die zur Missionsausführung relevanten Daten als *Object* zurück. Die Methode *SaveMissionData* nimmt, anhand des ausgewählten Controllerprofils, eine Typumwandlung des allgemeinen Objektes vor und konvertiert es in ein Objekt der entsprechenden Missionsdatenklasse (in diesem Fall *DJIMissionData*). Zur Speicherung der Missionsdaten in Dateiform werden diese durch die Klasse *XMLSerializer* im XML-Format serialisiert. Ein Beispiel für eine Missionsdatei im XML-Format kann dem Anhang B entnommen werden.

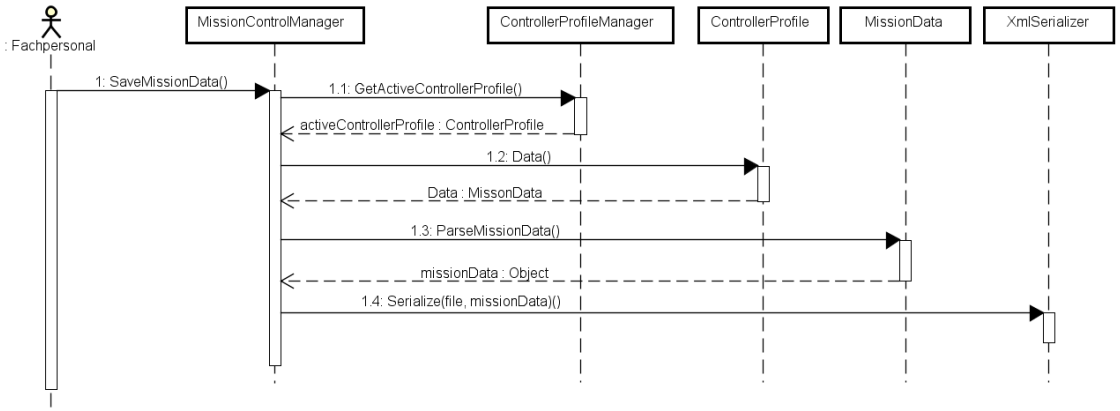



Abbildung 62: Sequenzdiagramm zum Speichern der Missionsdaten.

Die Steuerung einer Mission findet über das Versenden von Nachrichten zwischen Navigations- und Planungslösung statt. Hierzu wird eine Schnittstelle definiert, welche das Übertragen von Befehlscodes respektive Positionsdaten zwischen den Lösungen ermöglicht. Eine Nachricht setzt sich aus 2 Byte für den Befehlscode und 24 Byte für die Positionsdaten zusammen, welche als Double-Werte übertragen werden.

$$\underbrace{Befehlscode}_{2\ Byte}\ \underbrace{Positionsdaten}_{24\ Byte}$$

In Tabelle 9 sind die Befehlscodes zur Missionssteuerung und Positionsdatenübertragung aufgelistet.

Tabelle 8: Befehlscodes zum Austausch von Nachrichten zwischen der Planungs- und Navigationslösung.

| **Befehlscode** | **Beschreibung** | **Sender** |
|---|---|---|
| 0 | Drohne bei Planungslösung registrieren | Navigationslösung |
| 1 | Drohne wurde erfolgreich registriert | Planungslösung |
| 2 | Mission starten | Planungslösung |
| 3 | Mission erfolgreich gestartet | Navigationslösung |
| 4 | Mission stoppen | Planungslösung |
| 5 | Mission erfolgreich gestoppt | Navigationslösung |
| 6 | Mission erfolgreich beendet | Navigationslösung |
| 7 | Positionsdaten | Navigationslösung |

Zur Entgegennahme von Datenpaketen wird in der Planungslösung eine Instanz der *MissionServer*-Klasse erzeugt. Diese stellt einen TCP Listener bereit, welcher den Port 60000 belegt und einen TCP Client für die Navigationslösung instanziiert (*CreateListener*). Die durch den Client empfangenen Daten werden an die *ProcessMissionData*-Methode der *SocketHelper*-Klasse weitergereicht. Die Methode ruft in Abhängigkeit des Kommandocodes eine entsprechende Methode der *MissionControlManager*-Klasse auf.

Zur Registrierung der Drohne sendet die Navigationslösung den Befehlscode 0 an die Planungslösung. Die *ProcessMissionData*-Methode ruft daraufhin die *RegisterClient*-Methode der *MissionControlManager*-Klasse auf. Die Methode weist der *stream*-Variable, welche sich in der *MissionControlManager*-Klasse befindet, den Netzwerkstream des TCP Clients zu und sendet den Befehlscode 1 über den Stream zurück.

Nach dem Registriervorgang kann die Mission gestartet werden. Hierzu wird vom Benutzer die *StartMission*-Methode über das Betätigen eines Buttons ausgelöst. Die Methode sendet den Befehlscode 2 an die Navigationslösung. Wurde die Mission erfolgreich durch die Navigationslösung gestartet, sendet diese den Befehlscode 3 an die Planungslösung. Das Empfangen des Befehlscodes 3 bewirkt in der Planungslösung die Ausführung der Methode *OnMissionStart*. Diese legt eine neue Liste für die zu empfangenen Drohnenpositionen an (*flightRoutePoints*). Nach der Methodenausführung ist es der Drohne möglich Positionsdaten zu schicken. Dies erfolgt, wie oben beschrieben, durch das Versenden des Befehlscodes 7, gefolgt von den Positionsdaten der Drohne. Der Code löst das Aufrufen der Methode *ProcessPositionData*, der *MissionControlManager*-Klasse aus. Diese Methode liest die Positionsdaten aus und fügt diese der *flightRoutePoints*-Liste hinzu. Enthält diese Liste mehr als ein Element, wird diese an eine *LineRenderer*-Component[4] übergeben, welche die Flugroute als grüne Linie darstellt. Die aktuelle Drohnenposition wird durch ein 3D-Modell der *DJI Matrice 100* visualisiert (siehe Abbildung 63). Das Prüfen der Listenelementanzahl wird in der *Update*-Methode, der *MissionControlManager*-Klasse, vollzogen.

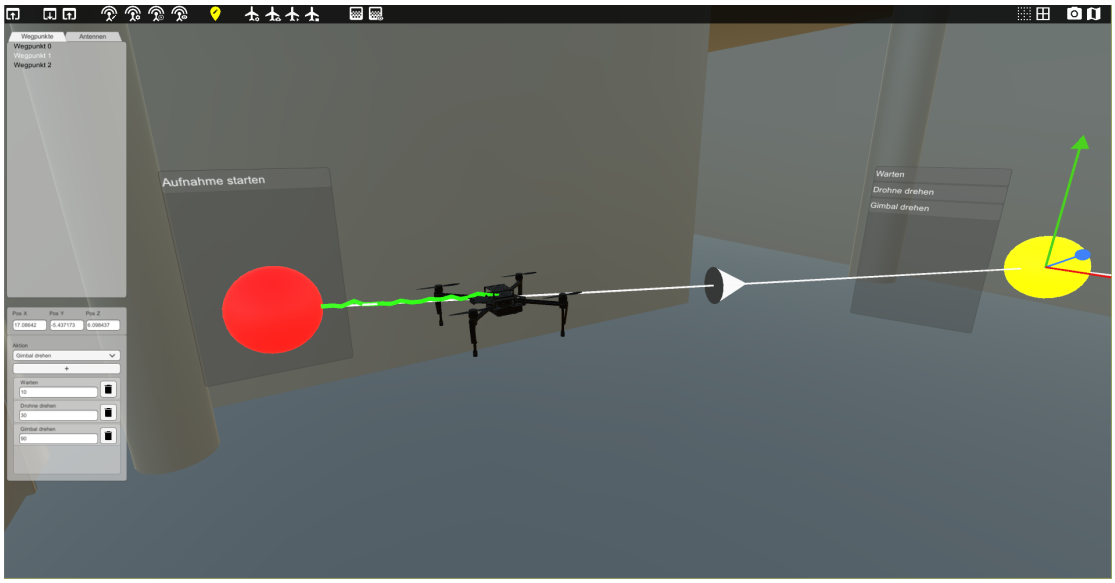



Abbildung 63: Visualisierung des Flugweges und der aktuellen Drohnenposition während der Missionsdurchführung.

[4]Die *LineRenderer*-Component ist Teil des Unity-Frameworks und dient dazu dreidimensionale Linien-Objekte zu erzeugen.

Soll eine Mission nach dem Starten gestoppt werden, wird die Methode *StopMission* aufgerufen, welche den Befehlscode 4 an die Navigationslösung sendet. Hat diese die Mission erfolgreich gestoppt, sendet sie den Befehlscode 5 an die Planungslösung zurück. Das Empfangen des Befehlscodes löst in der Planungslösung die Methode *OnMissionStopped* aus, welche eine leere *flightRoutePoints*-Liste erzeugt, was die Löschung der Flugroute zur Folge hat. Das erfolgreiche Beenden einer Mission wird von der Navigationslösung durch das Versenden des Befehlscodes 6 signalisiert. Hier wird in der Planungslösung die Methode *OnMissionComplete* aufgerufen, was ebenfalls das Löschen der bisherigen Route bewirkt. Die oben erläuterte Kommunikation zwischen der Navigations- und Planungslösung wird in Abbildung 64 dargestellt.

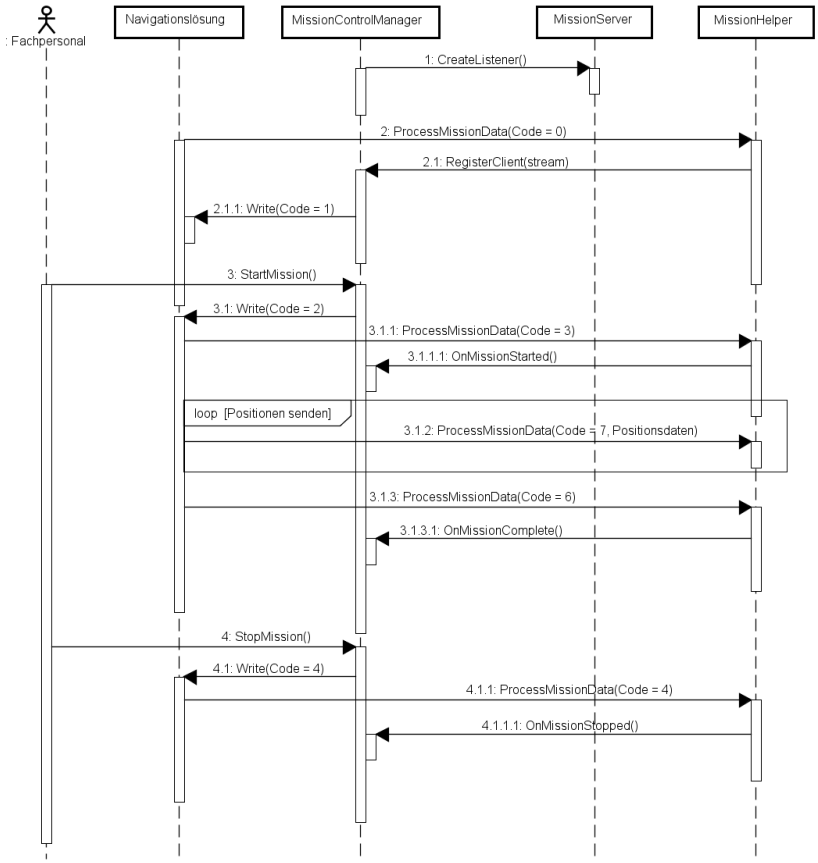



Abbildung 64: Sequenzdiagramm des Nachrichtenaustausches zwischen der Planungs- und Navigationslösung während der Missionsausführung.

## 5.4 Simulation der Gasverteilung

Nach erfolgreicher Missionsdurchführung werden die, durch die Drohne aufgenommenen, Gaskonzentrationen in die Planungslösung importiert. Auf Grundlage der Messwerte wird eine geostatistische Simulation durchgeführt, welche als Entscheidungsbasis zur Gasfreimessung des Ballastwassertanks dient.

Die Simulation der Gasverteilung wird durch die Verwendung der *krige*-Funktion

realisiert, welche durch die Softwareumgebung $R$ bereitgestellt wird (siehe 2.3 Simulationssoftware). Dies setzt voraus, dass die $R$-Umgebung auf dem Zielsystem der Planungslösung installiert ist. Für die Entwicklung der Planungslösung kommt $R$ in der Version 1.1.453 zum Einsatz.

Die Ausführung des, in $R$ erstellten, Skripts findet über die Klasse *GasManager* statt. Zur Prozessausführung nutzt diese die *Process*-Klasse (siehe Abbildung 65). Zur Visualisierung der simulierten Gaskonzentrationen wird die Klasse *Raster* verwendet. Diese erzeugt, wie bei der Visualisierung der Funkausbreitung, ein dreidimensionales Raster im Modellbereich, wobei eine Lichtquelle zur Indikation der Überschreitung des Grenzwertes fungiert. Zur Einschränkung des Simulationsbereiches wird das Raster auf eine lokale Untermenge im Bereich der Messwerte beschränkt.

Zur Selektion der relevanten Rasterzellen wird eine konvexe Hülle erzeugt, welche die im $\mathbb{R}^3$ lokalisierten Messwerte enthält. Zur Generierung der konvexen Hülle wird die Klasse *ConvexHull* verwendet, welche aus (Phillips 2016) entnommen ist und im Anhang C beigefügt wird. Eine Beschreibung des zugrundeliegenden Algorithmus ist in (Barber et al. 1996) zu finden.

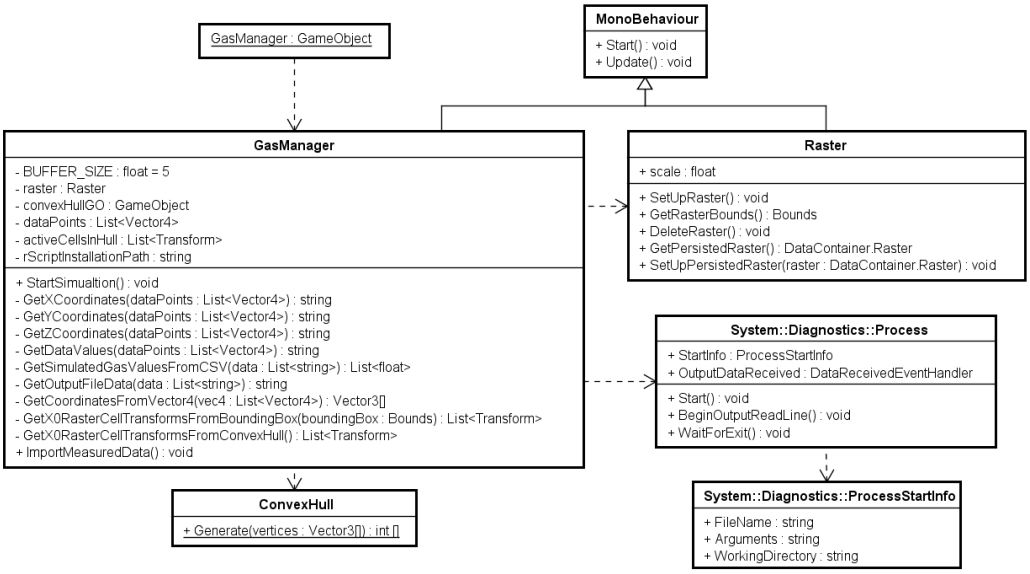



Abbildung 65: Vereinfachtes Klassendiagramm zur Ausführung der geostatistischen Simulation.

Die von der Navigationslösung bereitgestellten Gasmesswerte werden von der Planungslösung durch die Methode *ImportMeasuredData* importiert, welche durch den Nutzer über einen entsprechenden Button aufgerufen wird. Die Bereitstellung der Messwerte erfolgt durch eine Datei im CSV-Format. Diese enthält deren Koordinaten (x,y,z) gefolgt von der Gaskonzentration in ppm. Nach dem Importieren der Messdaten erfolgt das Starten der Simulation durch die *StartSimulation*-Methode, welche in Algorithmus 6 beschrieben ist. Die Methode erzeugt zunächst eine konvexe Hülle aus den Koordina-

ten der Messdaten, wobei sie die *Generate*-Methode der Klasse *ConvexHull* verwendet (siehe Algorithmus 6 Zeile 7). Aus den Knotenpunktindizes der konvexen Hülle (*indices*) und den Koordinaten der Messdaten (*coordinates*), wird ein *Mesh*-Spielobjekt (*convexHullMesh*) erzeugt (siehe Algorithmus 6 Zeile 8).

Zur Vergrößerung des Simulationsbereichs, wird das *Mesh* durch einen dreidimensionalen Buffer mit dem Abstand $BUFFER\_SIZE$ vergrößert. Hierbei werden die Knotenpunkte entlang der Knotenpunktnormalen um den Wert $BUFFER\_SIZE$ verschoben. Überschreitet ein Knotenpunkt dabei ein Polygon des 3D-Modells, wird dieser an den Schnittpunkt, des 3D-Modells mit der Knotenpunktnormalen, gesetzt (siehe Algorithmus 6 Zeile 9 bis 16). Anschließend folgt die Ermittlung aller, für die Simulation relevanten, Rasterzellen (*x0CellsInBox*), wobei alle Zellen selektiert werden die innerhalb der Bounding Box (*boundingBox*), der gepufferten konvexen Hülle liegen (siehe Algorithmus 6 Zeile 19 und 20).

Zur Durchführung der Simulation wird das Programm *RScript.exe* aufgerufen, welches zur Ausführung von, in *R* verfassten Skripten dient und Teil der *R*-Softwareumgebung ist. Das Starten des Programms erfolgt unter Verwendung der Klasse *Process* (siehe Algorithmus 6). Dem Programm wird hierbei der Dateipfad des Skripts *CSGS.R* übergeben. Dieses dient zur Durchführung der konditionierten sequentiellen Gauß'schen Simulation und ist im Anhang D beigefügt. Der Speicherpfad des Skripts wird unter Verwendung des *FileName*-Attributes angegeben, welches Bestandteil des *ProcessStartInfo*-Objekts *StartInfo*. Dieses stellt wiederum ein Attribut der Klasse *Process* dar. Dem Programm werden außerdem Simulationsparameter in Form von Startparametern übergeben. Dies erfolgt durch das Attribut *Arguments*. Die Simulationsparameter sind dabei durch ein Leerzeichen getrennt und können der folgenden Tabelle entnommen werden. Das Auslesen der Parameter erfolgt im Skript *CSGS.R* durch Auswertung der Argumentliste *args*. Die Angabe des, zur Simulation benötigten, Variogrammmodells und der dazugehörigen Parameter findet im Skript selbst statt.

Tabelle 9: Beschreibung der Simulationsparameter.

| **Parameter** | **Beschreibung** |
|---|---|
| xCoordinates | durch Kommata getrennte x-Koordinaten der Datenpunkte |
| yCoordinates | durch Kommata getrennte y-Koordinaten der Datenpunkte |
| zCoordinates | durch Kommata getrennte z-Koordinaten der Datenpunkte |
| dataValues | durch Kommata getrennte Gasmesswerte der Datenpunkte |
| xOrigin | x-Koordinate des 3D-Rasters |
| yOrigin | y-Koordinate des 3D-Rasters |
| zOrigin | z-Koordinate des 3D-Rasters |
| xDimension | Länge des Rasters entlang der x-Achse |
| yDimension | Länge des Rasters entlang der y-Achse |
| zDimension | Länge des Rasters entlang der z-Achse |
| cellWidth | Zellengröße einer Rasterzelle |
| nSim | Anzahl durchzuführender Simulationen |

Das Auslesen der vom Skript zurückgegebenen Simulationswerte erfolgt über den Ereignishandler *OutputDataReceived*, wobei die Werte in einer Liste gespeichert werden (Algorithmus 6 Zeile 22). Anschließend wird den Lichtquellen der Rasterzellen *x0CellsInBox* ein Farbwert und eine Intensität zugewiesen. Die Lichtfarbe und Intensität der Lichtquelle hängt davon ab, ob der angegebene Arbeitsplatzgrenzwert (*thresholdValue*) überschritten ist. Sollte dies der Fall sein, nimmt die Lichtquelle eine rote Färbung an und die Intensität wird auf 0.5 gesetzt. Wird der Grenzwert eingehalten, nimmt die Lichtquelle eine blaue Färbung an und leuchtet mit einer Intensität von 0.25 (siehe Algorithmus 6 Zeile 23 bis 33). Abschließend werden alle Rasterzellen ohne direkte Sichtverbindung zu einem Datenpunkt deaktiviert (siehe Algorithmus 6 Zeile 34 bis 46).

**Algorithmus 6** Starten der geostatistischen Simulation

1: **procedure** StartSimulation
2: $dataPoints \leftarrow$ Gasmesswerte als 4D Vector-Array
3: $thresholdValue \leftarrow$ Arbeitsplatzgrenzwert in ppm
4: $raster \leftarrow$ Raster zur Visualisierung der Gasmesswerte
5: $BUFFER\_SIZE \leftarrow$ Puffergröße der konvexen Hülle
6: $coordinates \leftarrow$ Koordinaten der Messpunkte
7: $indices \leftarrow$ Generieren der Konvexen-Hüllen-Indizes aus *coordinates*
8: $convexHullMesh \leftarrow$ Erzeugen eines Meshes aus den generierten Indizes
9: **for all** $v \in convexHullMesh.vertices$ **do**
10: $ray \leftarrow$ Strahl mit Länge $h$ von $v$ in Richtung der Normalen von $v$ aussenden
11: $hits \leftarrow$ alle Spielobjekte mit einem Collider die von $ray$ getroffen wurden
12: $modelHit \leftarrow$ Schnittpunkt mit dem 3D-Modell aus $hits$ ermitteln
13: **if** $modelHit \neq$ null **then**
14: $v$ wird die Position des Schnittpunktes mit dem 3D-Modell zugewiesen
15: **else**
16: $v$ wird um die $BUFFER\_SIZE$ entlang der Normalen von $v$ verschoben
17: **end if**
18: **end for**
19: $boundingBox \leftarrow$ Bounding Box der gepufferten konvexen Hülle
20: $x0CellsInBox \leftarrow$ Liste der Rasterzellen innerhalb von $boundingBox$
21: $process \leftarrow$ Starten des R-Skripts mit Übergabe der Simulationsparameter
22: $simulatedValues \leftarrow$ Liste der vom R-Skript zurückgegebenen Simulationswerte
23: **for all** $c \in x0CellsInBox$ **do**
24: $pointLight \leftarrow$ Lichtkomponente der Rasterzelle $c$
25: $simulatedValue \leftarrow$ Simulationswert an der Position der Rasterzelle $c$
26: **if** $simulatedValue > thresholdValue$ **then**
27: $pointLight.intensity = 0.5$
28: $pointLight$.color = Rot
29: **else**

```
30:             pointLight.intensity = 0.25
31:             pointLight.color = Blau
32:         end if
33:     end for
34:     for all c ∈ x0CellsInBox do
35:         for all Datenpunkte d ∈ dataPoints do
36:             ray ← Strahl zwischen c und d erzeugen
37:             hits ← alle Spielobjekte die von ray getroffen wurden
38:             modelHit ← Schnittpunkt mit dem 3D-Modell aus hits ermitteln
39:             if modelHit == null then
40:                 Rasterzelle c ist sichtbar
41:                 break
42:             else
43:                 c deaktivieren
44:             end if
45:         end for
46:     end for
47: end procedure
```

Abbildung 66 zeigt die Visualisierung der geostatistischen Simulation in der Planungslösung bei überschrittenem und eingehaltenem Grenzwert. Zusätzlich erstellt das *CSGS*-Skript einen 2D-Plot sowie eine CSV-Datei, welche die Simulationsergebnisse grafisch beziehungsweise tabellarisch angeben (siehe Anhang E und F).

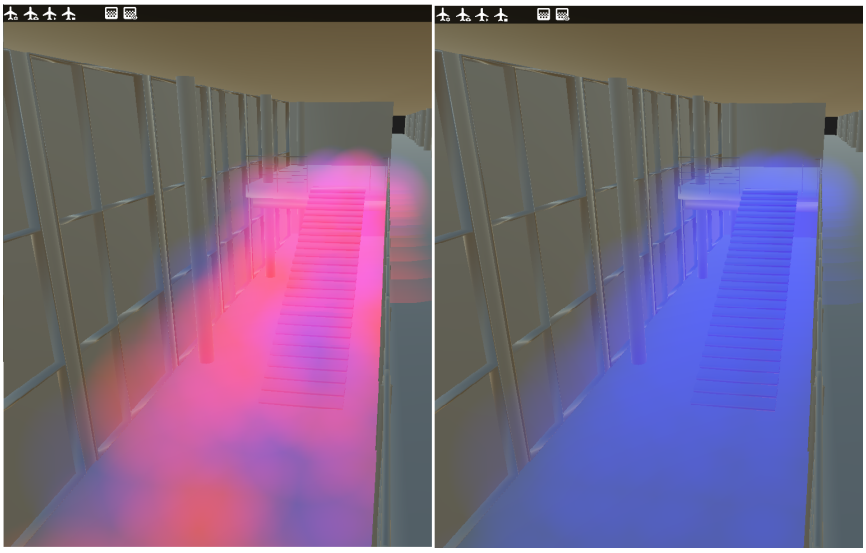


Abbildung 66: Visualisierung der simulierten Gaskonzentrationen in der Planungslösung, mit überschrittenem (links) und eingehaltenem Grenzwert (rechts).

# Teil IV
# Zusammenfassung

Das Ziel der vorliegenden Arbeit war es eine Planungslösung, für UAV-gestützte Gasmessungen in Ballastwassertanks von Schiffen, zu konzipieren und prototypisch zu implementieren. Die hierbei entwickelte Software sollte Funktionalitäten zur Missionsplanung und -durchführung bereitstellen sowie eine Berechnung der Funkausbreitung und die Simulation von Gaskonzentrationen innerhalb eines 3D-Modells ermöglichen.

Die im Rahmen der Arbeit entwickelte Planungslösung setzt die oben genannten Funktionalitäten wie spezifiziert um, wobei sich die Verwendung der Game Engine Unity 3D als hilfreich erweist. Die Engine liefert bereits Funktionalitäten für die Visualisierung von 3D-Modellen sowie Datentypen zur Verarbeitung dreidimensional verorteter Geodaten. Des Weiteren stellt sich das Ray Casting Feature der in Unity implementierten Physikengine als äußerst nützlich heraus, da sich hierdurch die elektromagnetische Welle im Hochfrequenzbereich als Strahl abbilden lässt. Darüber hinaus bietet die Engine punktuelle Lichtelemente, die eine einfache Visualisierung der Funkausbreitungs- und Simulationsergebnisse ermöglicht.

Für den Import des 3D-Modells eines Ballastwassertanks wurde eine Importfunktion umgesetzt, welche OBJ-Dateien einliest und diese in der Szene der Engine darstellt. Das OBJ-Format dient hierbei als offene Schnittstelle zum Modellimport und bietet eine leicht lesbare Struktur. Zudem ist es möglich, Materialinformationen als Metadaten zu speichern. Diese Metadaten können in fortführenden Versionen der Planungslösung dazu genutzt werden, Durchdringungsdämpfungen verschiedener Materialien in der Funkausbreitungsberechnung zu berücksichtigen.

Das im Rahmen dieser Arbeit entwickelte semi-empirische Funkausbreitungsmodell ermöglicht die Empfangsleistungsberechnung unter Einbezug der zur Sendeantenne gehörigen Abstrahlcharakteristik. Zur Angabe der Charakteristik wird eine Importfunktion von der Planungslösung bereitgestellt. Aufgrund der praktisch absoluten Reflexion hochfrequenter elektromagnetischer Wellen beim Auftreffen auf metallischen Oberflächen ist das entwickelte Modell, ohne die Verwendung von Dämpfungseigenschaften verschiedener Materialien implementiert worden. Soll das Modell außerhalb, vorwiegend metallischer Umgebungen eingesetzt werden, sind diese jedoch zu berücksichtigen. Des Weiteren wird der Beugungseffekt an Kanten nicht durch das Modell abgebildet. Während die Materialdämpfung unter geringem Aufwand in die schon bestehende Lösung integriert werden kann, muss zur Abbildung der Beugungseigenschaft eine automatisierte Kantendetektion für das importierte 3D-Modell umgesetzt werden. Da die Implementierung dieses Effektes einen, im Kontext dieser Arbeit, unverhältnismäßig hohen Programmieraufwand zur Folge hat, ist von einer Umsetzung dieser Eigenschaft abgesehen worden. Des Weiteren findet im Rahmen dieser Arbeit keine empirische Validierung des imple-

mentierten Modells statt. Daraus folgt, dass keine Genauigkeitsabschätzung des Modells möglich ist und die Validierung der Ergebnisdaten Bestandteil weiterer Untersuchungen bleibt.

Als weitere integrale Funktionen sind in der Planungslösung Werkzeuge zur Routenplanung und zum Missionsmanagement umgesetzt. Diese ermöglichen dem Benutzer zum einen die Angabe der Flugroute innerhalb des Ballastwassertanks zum anderen können den Wegpunkten verschiedene Aktionen zugewiesen werden. Um ein leicht erweiterbares Softwaredesign in Hinblick auf verschiedene Drohnenhersteller und deren Entwicklungsumgebungen zu schaffen, wird die Routenplanung unter Verwendung verschiedener Controllerprofile durchgeführt. Dieses Konzept vereinfacht die Implementierung herstellerabhängiger Schnittstellenspezifikationen von Wegpunkten sowie Wegpunktaktionen.

Die Bestimmung des Gefahrenpotenzials innerhalb des Ballastwassertanks wird unter Verwendung des sequentiellen Gauß'schen Simulationsverfahrens durchgeführt. Deren Integration findet mithilfe der Programmiersprache und Softwareumgebung *R* statt, wobei eine leicht anpassbare und erweiterbare Schnittstelle zwischen der Softwarelösung und *R* entwickelt worden ist. Die Ausgabe der Simulationswerte erfolgt sowohl grafisch als auch textlich, was eine weiterführende Darstellung sowie eine Weiterverarbeitung der Simulationsergebnisse zulässt. Eine Validierung der Simulationsergebnisse kann nicht vorgenommen werden, da zum Verfassungszeitpunkt dieser Arbeit, keine verorteten Gasmesswerte zur Verfügung standen. Aus diesem Grund ist die Berechnung der, für die geostatistische Simulation notwendigen, Variogrammparameter nicht möglich. Diese kann nachträglich, mit den von der Drohne aufgenommenen lokalisierten Gasmesswerten, vorgenommen werden. Während *R* robuste Algorithmen zur geostatistischen Simulation zur Verfügung stellt, erschwert dessen Einsatz eine Eins-zu-eins-Portierung der Softwarelösung auf mobilen Endgeräten. Der Aufwand, für die Portierung des bestehenden Programms auf Plattformen wie Android oder iOS, muss in weiteren Untersuchungen abgeschätzt werden. Hierzu kommt außerdem die Frage der Laufzeiteffizienz auf mobilen Plattformen.

# Literatur

[Akin u. Siemes 1988] Akin, Hikmet und Siemes, Heinrich: *Praktische Geostatistik - Eine Einführung für den Bergbau und ie Geowissenschaften.* 1. Berlin... : Springer, 1988

[Bachhuber 2011] Bachhuber, Martin: *Analyse und Modellierung der Funkausbreitung in Passagierkabinen von Großraumflugzeugen*, Universität Erlangen-Nürnberg, Dissertation, 2011

[Balzert 2009] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering.* 3. Heidelberg : Spektrum Akademischer Verlag, 2009. – ISBN 978–3–8274–1705–3

[Barber et al. 1996] Barber, C. B. und Dobkin, David P. und Huhdanpaa, Hannu: The Quickhull Algorithm for Convex Hulls. In: *ACM Transactions on Mathematical Software* 22 (1996), Nr. 4, S. 469–483. `http://dx.doi.org/10.1145/235815.235821`. – DOI 10.1145/235815.235821

[Blöschl 2006] Blöschl, Günter: Geostatistische Methoden bei der hydrologischen Regionalisierung. In: *Wiener Mitteilungen* 197 (2006), S. 21–40

[BSI TR-03209 - 1 2008] Bundesamt für Sicherheit in der Informationstechnik: Elektromagnetische Schirmung von Gebäuden - Theoretische Grundlagen. 2008. – Technische Richtlinie

[Buyuksaliha et al. 2017] Buyuksaliha, Ismail und Bayburta, Serdar und Buyuksaliha, Gurcan und Baskaracaa, A.P. und Karimb, Hairi und Rahmanb, Alias A.: 3D MODELLING AND VISUALIZATION BASED ON THE UNITY GAME ENGINE – ADVANTAGES AND CHALLENGES. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W4 (2017), S. 161–166. `http://dx.doi.org/doi.org/10.5194/isprs-annals-IV-4-W4-161-2017`. – DOI doi.org/10.5194/isprs–annals–IV–4–W4–161–2017

[Catmull u. Rom 1974] Catmull, Edwin und Rom, Raphael: A CLASS OF LOCAL INTERPOLATING SPLINES. Version: 1974. `http://dx.doi.org/https://doi.org/10.1016/B978-0-12-079050-0.50020-5`. In: BARNHILL, ROBERT E. (Hrsg.) und RIESENFELD, RICHARD F. (Hrsg.): *Computer Aided Geometric Design.* Academic Press, 1974. – DOI https://doi.org/10.1016/B978–0–12–079050–0.50020–5. – ISBN 978–0–12–079050–0, 317 - 326

[Chaabane 2005] Chaabane, Adnen: *Propagation modeling of wireless systems in shipboard compartments*, Naval Postgraduate School, Dissertation, 2005

[Chen 2012] Chen, Xiaomin: *Coding in 802.11 WLANs*, National University of Ireland Maynooth, Dissertation, 2012

[Christen et al. 2018] Christen, Markus und Guillaume, Michel und Jablonowski, Maximilian und Kurt Moll, Peter L.: *Zivile Drohnen – Herausforderungen und Perspektiven*. 1. Zürich : vdf Hochschulverlag AG, 2018. – ISBN 978–3–7281–3893–4

[Christensen et al. 2011] Christensen, Leif und Lemburg, Johannes und Vögele, Thomas und Kirchner, Frank und Fischer, Niklas und Ahlers, Reinhard und Psarros, George und Etzold, Lars-Eric: Tank inspection by cost effective rail based robots. In: *Proceedings of the 15th International Conference on Computer Applications in Shipbuilding. International Conference on Computer Applications in Shipbuilding (ICCAS-11), 15th, September 20-22, Trieste, Italy*, 2011

[Christopoulou u. Xinogalos 2017] Christopoulou, Eleftheria und Xinogalos, Stelios: Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. In: *International Journal of Serious Games* 4 (2017), 12, S. 21–36. `http://dx.doi.org/10.17083/ijsg.v4i4.194`. – DOI 10.17083/ijsg.v4i4.194

[Deutsch 2002] Deutsch, Clayton V.: *Geostatistical Reservoir Modelling*. 1. New York : Oxford University Press, 2002. – ISBN 0–19–513806–6

[DJI 2018a] DJI: *DJI Onboard SDK Documentation*. `https://developer.dji.com/onboard-api-reference/index.html`. Version: 2018

[DJI 2018b] DJI: *DJI::OSDK::WayPointInitSettings Struct Reference*. `https://developer.dji.com/onboard-api-reference/structDJI_1_1OSDK_1_1WayPointInitSettings.html`. Version: 2018

[DJI 2018c] DJI: *DJI::OSDK::WayPointSettings Struct Reference*. `https://developer.dji.com/onboard-api-reference/structDJI_1_1OSDK_1_1WayPointSettings.html`. Version: 2018

[DJI 2018d] DJI: *Ground Station Protocol*. `https://developer.dji.com/onboard-sdk/documentation/protocol-doc/ground-station-protocol.html`. Version: 2018

[DJI 2018e] DJI: *MISSIONS*. `https://developer.dji.com/onboard-sdk/documentation/guides/component-guide-missions.html`. Version: 2018

[DJI 2018f] DJI: *Onboard SDK Architectural Overview*. `https://developer.dji.com/onboard-sdk/documentation/introduction/sdk-architectural-overview.html`. Version: 2018

[Dolea et al. 2015] Dolea, Paula und Cristea, Octavian und Dascal, Paul V. und Moldovan, Iren-Adelina und Biagi, Pier F.: Aspects regarding the use of the INFREP network for identifying possible seismic precursors. In: *Physics and Chemistry of the Earth* 85-86 (2015), S. 34–43

[Dutter 1985] Dutter, Rudolf: *Geostatistik: Eine Einführung mit Anwendungen.* 2. Stuttgard : B. G. Teubner, 1985. – ISBN 3–519–02614–7

[Faruque 2014] Faruque, Saleh: *Radio Frequency Propagation Made Easy.* 1. New York : Springer, 2014. – ISBN 978–3–319–11394–4

[Fernández 2015] Fernández, Leyre A.: *Characterization of Wireless Propagation in Complex Indoor Environments*, Universidad Publica de Navarra, Dissertation, 2015

[Gau 2010] Gau, Christian: *Geostatistik in der Baugrundmodellierung: Die Bedeutung des Anwenders im Modellierungsprozess.* 1. Wiesbaden : Vieweg+Teubner, 2010. – ISBN 978–3–8348–1432–6

[Geng u. Wiesbeck 1998] Geng, Norbert und Wiesbeck, Werner: *Planungsmethoden für die Mobilkommunikation: Funknetzplanung unter realen physikalischen Ausbreitungsbedingungen.* 1. Berlin : Springer, 1998. – ISBN 978–3–642–58980–5

[Goovaerts 1999] Goovaerts, Pierre: Geostatistics in soil science: state-of-the-art and perspectives. In: *Geoderma* 89 (1999), S. 1–45. `http://dx.doi.org/10.1007/s10015-017-0372-3`. – DOI 10.1007/s10015–017–0372–3

[Hengl 2007] Hengl, Tomislav: *A Practical Guide to Geostatistical Mapping of Environmental Variables.* 2007

[Heuberger u. Gamm 2017] Heuberger, Albert und Gamm, Eberhard: *Software Defined Radio-Systeme für die Telemetrie Aufbau und Funktionsweise von der Antenne bis zum Bit-Ausgangl.* 1. Berlin : Springer Vieweg, 2017. – ISBN 978–3–662–53233–1

[Hoff et al. 2005] Hoff, Simon und Wetzlar, Joachim und Pütz, Wilhelm und Ternes, Berthold: *Sicheres WLAN (TR-S-WLAN) Teil 1: Darstellung und Bewertung der Sicherheitsmechanismen*, Bundesamt für Sicherheit in der Informationstechnik, Technische Richtlinie, 2005

[Hussain 2017] Hussain, Sajjad: *Efficient Ray-Tracing Algorithms for RadioWave Propagation in Urban Environments*, School of Electronic Engineering Dublin City Universityl, Dissertation, 2017

[Kark 2018] Kark, Klaus W.: *Antennen und Strahlungsfelder: Elektromagnetische Wellen auf Leitungen im Freiraum und Ihre Abstrahlungl.* 7. Wiesbaden : Springer Vieweg, 2018. – ISBN 978–3–658–22318–2

[Keller 1962] Keller, Joseph B.: Geometrical Theory of Diffraction. In: *JOURNAL OF TILE OPTICAL SOCIETY OF AMERICA* 52 (1962), Nr. 2, S. 116–130

[Kouyoumjian u. Pathak 1974] Kouyoumjian, Robert G. und Pathak, Prabhakar H.: A Unifm Geometrical Theory of Diffraction for an Edge in a Perfectly Conducting Surface. In: *Proceedings of the IEEE* 62 (1974), S. 1448–1460

[Lu et al. 2017] Lu, Zeming und Nagata, Fusaomi und Watanabe, Keigo und Habib, Maki K.: iOS application for quadrotor remote control. In: *Artif Life Robotics* 22 (2017), S. 374–379. http://dx.doi.org/10.1007/s10015-017-0372-3. – DOI 10.1007/s10015–017–0372–3

[Luo 2013] Luo, Meiling: *Indoor radio propagation modeling for system performance prediction*, INSA de Lyona, Dissertation, 2013

[Murray u. vanRyper 1996] Murray, James D. und vanRyper, William: *Enceclopedia of Graphics File Formats*. 2. Sebastopol : O'Reilly and Associates, Inc, 1996. – ISBN 1–56592–161–5

[Pebesma 2014] Pebesma, Edzer J.: *gstat usere's mauel*. 2014

[Phillips 2016] Phillips, Eric: *Project Title*. https://gist.github.com/YclepticStudios/c0d8cea56ee1e1d9714754ee9427085b, 2016

[Preiner et al. 2006] Preiner, Patrick und Schmid, Gernot und Lager, Daniel und Georg, Reinhard: *Bestimmung der realen Feldverteilung von hochfrequenten elektromagnetischen Feldern in der Umgebung von Wireless LAN-Einrichtungen (WLAN) in innerstädtischen Gebieten*, Bundesamt für Strahlenschutz, Forschungsbeleg, 2006

[Rech 2006] Rech, Jörg: *Wireless LANs 802.11-WLAN-Technologie und praktische Umsetzung im Detail*. 2. Hannover : Heise Zeitschriften Verlag, 2006. – ISBN 3–936931–29–1

[Retscher u. Tatschl 2017] Retscher, Günther und Tatschl, Thomas: Positionierung in Gebäuden mit differenziellem WLAN. In: *zvf* 2/2017 (2017), S. 111–125

[Sa et al. 2018] Sa, Inkyu und Kamel, Mina und Khanna, Raghav und Popovic, Marija und Nieto, Juan und Siegwart, Roland: Dynamic System Identification, and Control for a cost effective open-source VTOL MAV. In: *Field and Service Robotics: Results of the 11th International Conference* (2018), S. 605–620. http://dx.doi.org/10.1007/978-3-319-67361-5_39. – DOI 10.1007/978–3–319–67361–5_39

[Schafmeister 1999] Schafmeister, Maria-Theresia: *Geostatistik für die hydrogeologische Praxis*. 1. Berlin... : Springer, 1999. – ISBN 978–3–540–66180–1

[Unity 2018a] Unity, Technologies: *Cameras*. https://docs.unity3d.com/Manual/CamerasOverview.html. Version: 2018

[Unity 2018b] Unity, Technologies: *Colliders*. https://docs.unity3d.com/Manual/CollidersOverview.html. Version: 2018

[Unity 2018c] Unity, Technologies: *Creating And Using Scripts*. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. Version: 2018

[Unity 2018d] UNITY, Technologies: *GameObject*. https://docs.unity3d.com/Manual/class-GameObject.html. Version: 2018

[Unity 2018e] UNITY, Technologies: *Instantiating Prefabs*. https://docs.unity3d.com/Manual/InstantiatingPrefabs.html. Version: 2018

[Unity 2018f] UNITY, Technologies: *Lightning Overview*. https://docs.unity3d.com/Manual/LightingInUnity.html. Version: 2018

[Unity 2018g] UNITY, Technologies: *Materials, Schader and Textures*. https://docs.unity3d.com/Manual/Shaders.html. Version: 2018

[Unity 2018h] UNITY, Technologies: *Mesh Collider*. https://docs.unity3d.com/Manual/class-MeshCollider.html. Version: 2018

[Unity 2018i] UNITY, Technologies: *Mesh Renderer*. https://docs.unity3d.com/Manual/class-MeshRenderer.html. Version: 2018

[Unity 2018j] UNITY, Technologies: *Physics*. https://docs.unity3d.com/Manual/PhysicsSection.html. Version: 2018

[Unity 2018k] UNITY, Technologies: *Physics.Raycast*. https://docs.unity3d.com/ScriptReference/Physics.Raycast.html. Version: 2018

[Unity 2018l] UNITY, Technologies: *Scenes*. https://docs.unity3d.com/Manual/CreatingScenes.html. Version: 2018

[Unity 2018m] UNITY, Technologies: *Transforms*. https://docs.unity3d.com/Manual/Transforms.html. Version: 2018

[Unity 2018n] UNITY, Technologies: *Types of light*. https://docs.unity3d.com/Manual/Lighting.html. Version: 2018

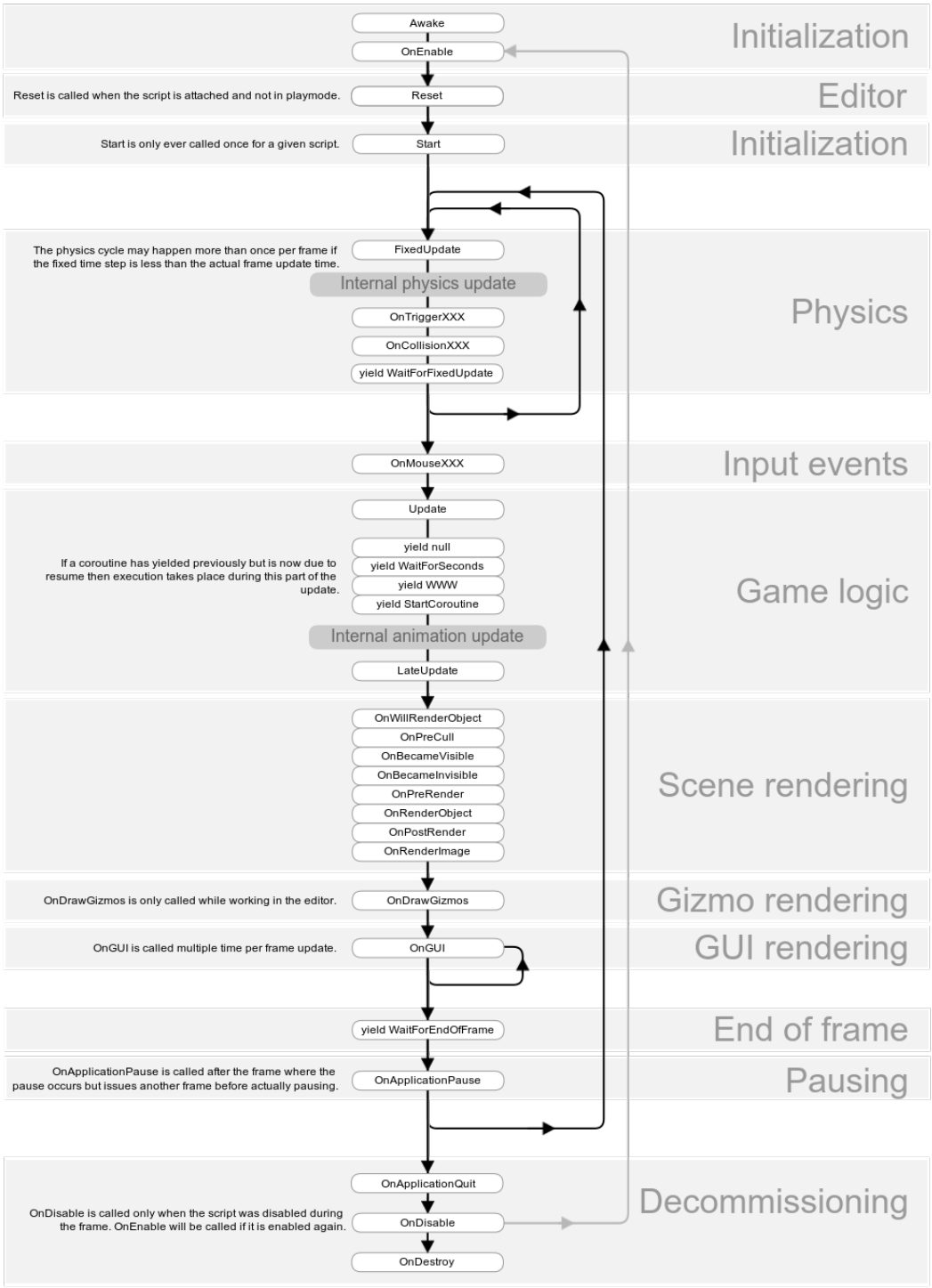
[Unity 2018o] UNITY, Technologies: *Using Mesh Class*. https://docs.unity3d.com/Manual/UsingtheMeshClass.html. Version: 2018

[Vazquez 2013] VAZQUEZ, Albert A.: *Experimelntal evaluation of the PHY layer of WSN focused on smart city applications*, Universitat Autonoma De Barcelona, Dissertation, 2013

[Viol et al. 2012] VIOL, Nicolai und LINK, Jo Agila B. und WIRTZ, Hanno und ROTHE, Dirk und WEHRLE, Klaus: Hidden Markov Model-based 3D Path-matching using Raytracing-generated Wi-Fi Models. In: *International Conference on Indoor Positioning and Indoor Navigation (IPIN)* (2012), S. 1–10

# Anhang

## Anhang A : Monobehaviour Flowchart



## Anhang B : Beispiel für eine Missionsdatei im XML-Format


```
<?xml version="1.0" encoding="Windows-1252"?>
<DJIMissionData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema">
  <wayPointInitData>
    <indexNumber>3</indexNumber>
    <maxVelocity>10</maxVelocity>
    <idleVelocity>5</idleVelocity>
    <finishAction>0</finishAction>
    <executiveTimes>1</executiveTimes>
    <yawMode>0</yawMode>
    <traceMode>0</traceMode>
    <RCLostAction>1</RCLostAction>
    <gimbalPitch>0</gimbalPitch>
    <latitude>0</latitude>
    <longitude>0</longitude>
    <altitude>0</altitude>
  </wayPointInitData>
  <wayPointData>
    <WayPointData>
      <index>0</index>
      <latitude>22.88744</latitude>
      <longitude>-4.804255</longitude>
      <altitude>5.640264</altitude>
      <damping>0</damping>
      <yaw>0</yaw>
      <gimbalPitch>0</gimbalPitch>
      <turnMode>0</turnMode>
      <hasAction>0</hasAction>
      <actionTimeLimit>100</actionTimeLimit>
      <actionNumber>0</actionNumber>
      <actionReapeat>0</actionReapeat>
      <commandList />
      <commandParameter />
    </WayPointData>
    <WayPointData>
      <index>1</index>
      <latitude>19.19181</latitude>
      <longitude>-7.691404</longitude>
      <altitude>5.640264</altitude>
      <damping>0</damping>
      <yaw>0</yaw>
      <gimbalPitch>0</gimbalPitch>
      <turnMode>0</turnMode>
```

```
      <hasAction>0</hasAction>
      <actionTimeLimit>100</actionTimeLimit>
      <actionNumber>0</actionNumber>
      <actionReapeat>0</actionReapeat>
      <commandList />
      <commandParameter />
    </WayPointData>
    <WayPointData>
      <index>2</index>
      <latitude>18.62908</latitude>
      <longitude>−5.819605</longitude>
      <altitude>5.640264</altitude>
      <damping>0</damping>
      <yaw>0</yaw>
      <gimbalPitch>0</gimbalPitch>
      <turnMode>0</turnMode>
      <hasAction>0</hasAction>
      <actionTimeLimit>100</actionTimeLimit>
      <actionNumber>0</actionNumber>
      <actionReapeat>0</actionReapeat>
      <commandList />
      <commandParameter />
    </WayPointData>
  </wayPointData>
</DJIMissionData>
```

## Anhang C : ConvexHull.cs

```
/**
 * ==========================================================
 * MIT License
 *
 * Copyright (c) 2016 Eric Phillips
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
     EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
     MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
     EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
     OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
     ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
     OTHER
 * DEALINGS IN THE SOFTWARE.
 * ==========================================================
 *
 * This file implements a 3D generalization of the convex hulling algorithm
 * known as the "QuickHull" algorithm. This algorithm generates a surface for a
 * point cloud which contains all of the points and all of the surface features
 * possible using only a convex mesh.
 *
 * The procedure followed here is based on the following example.
 * http://thomasdiewald.com/blog/?p=1888
 *
 * Created by Eric Phillips on October 23, 2016.
 */

using System.Collections.Generic;
using UnityEngine;
```

```
public static class ConvexHull
{
    /// <summary>
    /// Class for holding faces and their data.
    /// </summary>
    private class FaceData
    {
        public Vector4 Plane; // Equation of a face's plane
        public int[] FaceIndices; // Indices of the face's points
        public List<int> VisibleIndices; // Indices of points visible to face
        private Vector3[] _vertices; // Main vertex array


        /// <summary>
        /// Create a new data structure for a face.
        /// </summary>
        /// <param name="pt1Idx">Index of the first point on the face.</param>
        /// <param name="pt2Idx">Index of the second point on the face.</param>
        /// <param name="pt3Idx">Index of the third point on the face.</param>
        /// <param name="vertices">Array of vertices.</param>
        public FaceData(int pt1Idx, int pt2Idx, int pt3Idx, Vector3[] vertices)
        {
            Plane = Points2Plane(vertices[pt1Idx], vertices[pt2Idx],
                vertices[pt3Idx]);
            FaceIndices = new int[] { pt1Idx, pt2Idx, pt3Idx };
            VisibleIndices = new List<int>();
            _vertices = vertices;
        }

        /// <summary>
        /// Get the point farthest from the plane in the positive direction.
        /// </summary>
        /// <returns>The index of the point.</returns>
        public int GetFurthestPoint()
        {
            int maxIndex = 0;
            float maxDistance = float.NegativeInfinity;
            foreach (int index in VisibleIndices)
            {
                // Calculate something linearly equivalent to the plane distance
                Vector3 v = _vertices[index] - _vertices[FaceIndices[0]];
                float distance = Vector3.Dot(v, Plane);
                if (distance > maxDistance)
                {
                    maxIndex = index;
```

```
                    maxDistance = distance;
                }
            }
            return maxIndex;
        }
    }

    /// <summary>
    /// Compute indices of triangles which form a convex hull around the given
    /// array of vertices. This is formatted so that a set of mesh vertices can
    /// be passed in, and the output can be directly assigned to the triangles
    /// property of the mesh.
    /// </summary>
    /// <param name="vertices">Array of vertices passed in.</param>
    /// <returns>Array of triangles as indices into the input array.</returns>
    public static int[] Generate(Vector3[] vertices)
    {
        // Create initial simplex
        int[] extremePoints = FindExtremePoints(vertices);
        int[] initialSimplex = CreateInitialSimplex(extremePoints, vertices);
        // Assign each vertex to the first face to which it is visible
        List<FaceData> faceStack =
            AssignInitialPointsToFaces(initialSimplex, vertices);
        // Iterate until all faces have been processed
        FaceData face;
        while ((face = GetUnprocessedFace(faceStack)) != null)
        {
            // Get the point farthest from the plane in the positive direction
            int maxPtIndex = face.GetFurthestPoint();
            // Get all faces which are visible to this point and pop from stack
            List<FaceData> visibleFaces = ExtractVisibleFaces(
                vertices[maxPtIndex], faceStack);
            // Extract the horizon edges of the visible faces
            // All faces will be connected
            Dictionary<string, int[]> horizonEdges =
                ExtractHorizonEdges(visibleFaces);
            // Create new points from horizon edges and max point
            List<FaceData> newFaces = CreateNewFaces(maxPtIndex, visibleFaces,
                vertices, horizonEdges);
            // Add new faces to stack and repeat
            faceStack.AddRange(newFaces);
        }
        // Compile one array of all the triangle indices
        int[] indices = new int[faceStack.Count * 3];
        for (int ii = 0; ii < faceStack.Count; ii++)
            for (int jj = 0; jj < 3; jj++)
```

```
            indices[ii * 3 + jj] = faceStack[ii].FaceIndices[jj];
        return indices;
    }

    /// <summary>
    /// Find the indices of the points in the vertices array which are at a
    /// minimum or maximum on each of the three axis. So find the min and max
    /// X, the min and max Y and the min and max Z points, and return their
    /// indices.
    /// </summary>
    /// <param name="vertices">Array of vertices passed in.</param>
    /// <returns>Array of six indices into the vertices array.</returns>
    private static int[] FindExtremePoints(Vector3[] vertices)
    {
        // Setup variables
        int[] extremePoints = new int[6];
        float[] extremePointValues = new float[6];
        for (int ii = 0; ii < extremePoints.Length - 1; ii += 2)
        {
            extremePoints[ii] = 0;
            extremePoints[ii + 1] = 0;
            extremePointValues[ii] = float.PositiveInfinity;
            extremePointValues[ii + 1] = float.NegativeInfinity;
        }
        // Search point cloud
        for (int ii = 0; ii < vertices.Length; ii++)
            for (int jj = 0; jj < extremePoints.Length - 1; jj += 2)
            {
                float val = vertices[ii][jj / 2];
                if (val < extremePointValues[jj])
                {
                    extremePoints[jj] = ii;
                    extremePointValues[jj] = val;
                }
                else if (val > extremePointValues[jj + 1])
                {
                    extremePoints[jj + 1] = ii;
                    extremePointValues[jj + 1] = val;
                }
            }
        return extremePoints;
    }

    /// <summary>
    /// Create a tetrahedron of maximum volume out of the extreme points.
    /// This returns four indices in right-handed manner. The first three
```

```
/// define the base of the tetrahedron, and face away from the third point.
/// </summary>
/// <param name="extremePoints">The indices of the extreme points.</param>
/// <param name="vertices">Array of vertices passed in.</param>
/// <returns>The indices of the initial tetrahedron.</returns>
private static int[] CreateInitialSimplex(int[] extremePoints,
    Vector3[] vertices)
{
    int[] initialSimplex = new int[4];
    // Find two most distent extreme points (base line of tetrahedron)
    float maxDistance = float.NegativeInfinity;
    for (int ii = 0; ii < extremePoints.Length; ii++)
        for (int jj = ii + 1; jj < extremePoints.Length; jj++)
        {
            float distance = (vertices[extremePoints[ii]] -
                vertices[extremePoints[jj]]).sqrMagnitude;
            if (distance > maxDistance)
            {
                initialSimplex[0] = extremePoints[ii];
                initialSimplex[1] = extremePoints[jj];
                maxDistance = distance;
            }
        }
    // Find the extreme point most distent from the line
    maxDistance = float.NegativeInfinity;
    Vector3 normal = vertices[initialSimplex[0]] -
        vertices[initialSimplex[1]];
    for (int ii = 0; ii < extremePoints.Length; ii++)
    {
        Vector3 v = vertices[extremePoints[ii]] -
            vertices[initialSimplex[0]];
        Vector3 rejection = Vector3.ProjectOnPlane(v, normal);
        float distance = rejection.sqrMagnitude;
        if (distance > maxDistance)
        {
            initialSimplex[2] = extremePoints[ii];
            maxDistance = distance;
        }
    }
    // Find the most distant of all the points from the plane of the
    // triangle formed from the first three "initialSimplex" points
    maxDistance = float.NegativeInfinity;
    Vector3 v1 = vertices[initialSimplex[1]] - vertices[initialSimplex[0]];
    Vector3 v2 = vertices[initialSimplex[2]] - vertices[initialSimplex[0]];
    normal = Vector3.Cross(v1, v2);
    for (int ii = 0; ii < vertices.Length; ii++)
```

```
        {
            Vector3 v = vertices[ii] − vertices[initialSimplex[0]];
            float distance = Mathf.Abs(Vector3.Dot(v, normal));
            if (distance > maxDistance)
            {
                initialSimplex[3] = ii;
                maxDistance = distance;
            }
        }
        // Swap the two first vertices if the final point is in front of the
        // triangular base plane (this makes all faces of the tetrahedron point)
        // outward
        Vector4 baseFace = Points2Plane(vertices[initialSimplex[0]],
            vertices[initialSimplex[1]],
            vertices[initialSimplex[2]]);
        if (PointAbovePlane(vertices[initialSimplex[3]], baseFace))
        {
            int t = initialSimplex[0];
            initialSimplex[0] = initialSimplex[1];
            initialSimplex[1] = t;
        }
        return initialSimplex;
    }

    /// <summary>
    /// Create the first four faces and assign all the points to the first face
    /// to which they are visible.
    /// </summary>
    /// <param name="initialSimplex">The indices of the initial simplex.</param>
    /// <param name="vertices">Array of vertices.</param>
    /// <returns>The initial list of faces with their data.</returns>
    private static List<FaceData> AssignInitialPointsToFaces(
        int[] initialSimplex, Vector3[] vertices)
    {
        // Create the first four faces
        List<FaceData> faceData = new List<FaceData>();
        faceData.Add(new FaceData(initialSimplex[0], initialSimplex[1],
            initialSimplex[2], vertices));
        faceData.Add(new FaceData(initialSimplex[0], initialSimplex[2],
            initialSimplex[3], vertices));
        faceData.Add(new FaceData(initialSimplex[1], initialSimplex[3],
            initialSimplex[2], vertices));
        faceData.Add(new FaceData(initialSimplex[1], initialSimplex[0],
            initialSimplex[3], vertices));
        // Assign each point to the first face to which it is visible
        for (int ii = 0; ii < vertices.Length; ii++)
```

```
            foreach (FaceData face in faceData)
                if (PointAbovePlane(vertices[ii], face.Plane))
                {
                    face.VisibleIndices.Add(ii);
                    break;
                }
        return faceData;
    }

    /// <summary>
    /// Get a face from the stack which still needs to be processed.
    /// </summary>
    /// <param name="faceStack">The list of faces.</param>
    /// <returns>The face to process next.</returns>
    private static FaceData GetUnprocessedFace(List<FaceData> faceStack)
    {
        foreach (FaceData face in faceStack)
            if (face.VisibleIndices.Count > 0)
                return face;
        return null;
    }

    /// <summary>
    /// Find all the faces which can be seen from the given point. This is the
    /// same as if the point were a light and we were looking for all faces
    /// which the point illuminated. Faces pointing the opposite direction are
    /// ignored. These faces are removed from the stack.
    /// </summary>
    /// <param name="pt">The point in question.</param>
    /// <param name="faceStack">The list of faces to search.</param>
    /// <returns>A new list of faces.</returns>
    private static List<FaceData> ExtractVisibleFaces(Vector3 pt,
        List<FaceData> faceStack)
    {
        List<FaceData> visibleFaces = new List<FaceData>();
        foreach (FaceData face in faceStack)
            if (PointAbovePlane(pt, face.Plane))
                visibleFaces.Add(face);
        foreach (FaceData face in visibleFaces)
            faceStack.Remove(face);
        return visibleFaces;
    }

    /// <summary>
    /// Get pairings of all the edges around the set of visible faces.
    /// This basically finds one outline around all the given faces.
```

```
/// This is returned as a dictionary to reduce data copying between
/// functions.
/// </summary>
/// <param name="visibleFaces">A list of the visible faces.</param>
/// <returns>A dictionary with the edge pairs.</returns>
private static Dictionary<string, int[]> ExtractHorizonEdges(
    List<FaceData> visibleFaces)
{
    Dictionary<string, int[]> edges = new Dictionary<string, int[]>();
    foreach (FaceData face in visibleFaces)
        for (int ii = 0; ii < face.FaceIndices.Length; ii++)
        {
            // Get indices of triangle vertices
            int idx1 = face.FaceIndices[ii];
            int idx2 = face.FaceIndices[(ii + 1) % face.FaceIndices.Length];
            // Use keys to easily detect and remove duplicate edges
            // If an edge appears twice, it is shared by two triangles,
            // and not really an edge
            string key = idx1 + "," + idx2;
            string keyReversed = idx2 + "," + idx1;
            if (edges.ContainsKey(key) || edges.ContainsKey(keyReversed))
            {
                edges.Remove(key);
                edges.Remove(keyReversed);
            }
            else
                edges.Add(key, new int[] { idx1, idx2 });
        }
    return edges;
}

/// <summary>
/// Create new faces between the horizon edges and the maximum point.
/// This is somewhat like extruding the edges out to the maximum point to
/// form new faces.
/// </summary>
/// <param name="maxPtIndex">Point farthest from current face.</param>
/// <param name="visibleFaces">List of current visible faces.</param>
/// <param name="vertices">Array of vertices.</param>
/// <param name="horizonEdges">Dictionary of pairs of indices.</param>
/// <returns>A new list of faces.</returns>
private static List<FaceData> CreateNewFaces(int maxPtIndex,
    List<FaceData> visibleFaces, Vector3[] vertices,
    Dictionary<string, int[]> horizonEdges)
{
    List<FaceData> newFaces = new List<FaceData>();
```

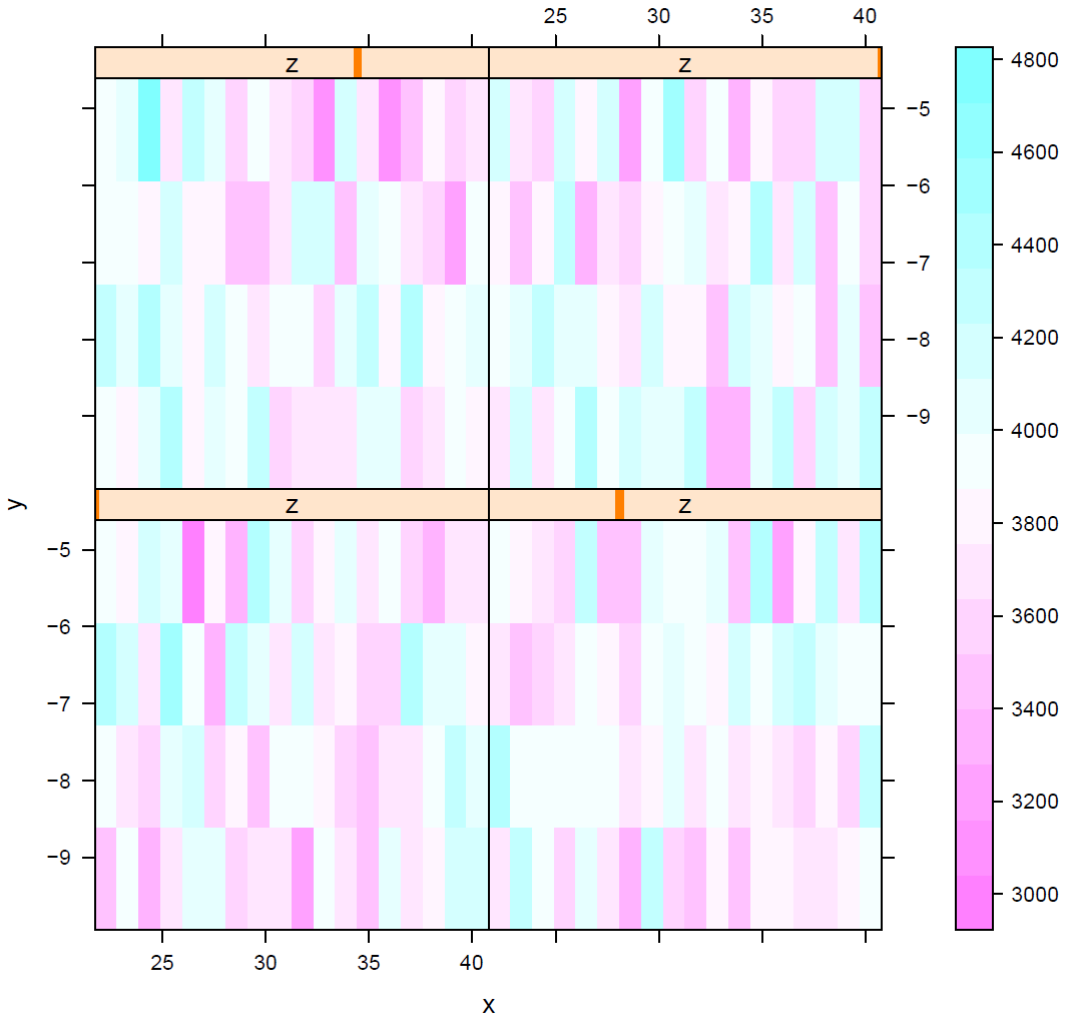
```
        // Create new faces from horizon edges and max point
        foreach (int[] edge in horizonEdges.Values)
            newFaces.Add(new FaceData(edge[0], edge[1], maxPtIndex, vertices));
        // Assign all points off all visible faces to the first of the new
        // faces to which the point is visible
        foreach (FaceData face in visibleFaces)
            foreach (int idx in face.VisibleIndices)
                foreach (FaceData newFace in newFaces)
                    if (PointAbovePlane(vertices[idx], newFace.Plane))
                    {
                        newFace.VisibleIndices.Add(idx);
                        break;
                    }
        return newFaces;
    }

    /// <summary>
    /// Convert three points to a plane equation of the form
    /// Ax + By + Cz + D = 0. The returned Vector4 contains A, B, C and D.
    /// </summary>
    /// <param name="pt1">The first point.</param>
    /// <param name="pt2">The second point.</param>
    /// <param name="pt3">The third point.</param>
    /// <returns>A new Vector4.</returns>
    private static Vector4 Points2Plane(Vector3 pt1, Vector3 pt2, Vector3 pt3)
    {
        Vector4 v = Vector3.Cross(pt2 - pt1, pt3 - pt1).normalized;
        v.w = -Vector3.Dot(v, pt1);
        return v;
    }

    /// <summary>
    /// Checks if the point is on the front face of the plane.
    /// This is the same as the point being able to "see" the front of the
    /// plane.
    /// </summary>
    /// <param name="point">The point in question.</param>
    /// <param name="plane">The plane in question.</param>
    /// <returns>A boolean value.</returns>
    private static bool PointAbovePlane(Vector3 point, Vector4 plane)
    {
        return Vector3.Dot(point, plane) + plane.w > 0.0000001f;
    }
}
```

## Anhang D : CSGS.R

```
library("gstat", lib.loc="~/R/win-library/3.5")
library("sp", lib.loc="~/R/win-library/3.5")

args <- commandArgs(trailingOnly = TRUE)

xPositions = unlist(strsplit(args[1], split=","));
yPositions = unlist(strsplit(args[2], split=","));
zPositions = unlist(strsplit(args[3], split=","));
values = unlist(strsplit(args[4], split=","));

class(xPositions) <- "numeric"
class(yPositions) <- "numeric"
class(zPositions) <- "numeric"
class(values) <- "numeric"

xOrigin = as.numeric(args[5])
yOrigin = as.numeric(args[6])
zOrigin = as.numeric(args[7])

xDimension = as.numeric(args[8])
yDimension = as.numeric(args[9])
zDimension = as.numeric(args[10])
cellWidth = as.numeric(args[11])

nSim = as.numeric(args[12])

data3D <- data.frame(x = xPositions, y = yPositions, z = zPositions, v = values)

coordinates(data3D) = ~x+y+z

xLength <- (xDimension / cellWidth)
yLength <- (yDimension / cellWidth)
zLength <- (zDimension / cellWidth)

# 3D-Raster erstellen
rangeX <- seq(from = xOrigin, to = xDimension + xOrigin, length = xLength)
rangeY <- seq(from = yOrigin, to = yDimension + yOrigin, length = yLength)
rangeZ <- seq(from = zOrigin, to = zDimension + zOrigin, length = zLength)

grid3D <- expand.grid(x = rangeX, y = rangeY, z = rangeZ)
gridded(grid3D) = ~x+y+z

#Konditionierte Sequentielle Gaußsche Simulaion durchführen
```

```
res3D <- krige(formula = v ~ 1, data3D, grid3D, model = vgm(80000, "Gau", .2),nsim=
    nSim,maxdist=10,nmax=10)

#Simulationsergebnisse plotten
library("lattice", lib.loc="~/R/win-library/3.5")
levelplot(sim1 ~ x + y | z, as.data.frame(res3D))

write.csv(as.data.frame(res3D))
```

## Anhang E : Plot der Simulationsergebnisse



## Anhang F : Simulationsergebnisse im CSV-Format

drawing 1 GLS realisation of beta...
[**using** conditional Gaussian simulation]
"","x","y","z","sim1"
"1",22.27406,−9.275667,2.220263,4307.24276799483
"2",23.3328835294118,−9.275667,2.220263,3547.87093970521
"3",24.3917070588235,−9.275667,2.220263,4199.79146856063
"4",25.4505305882353,−9.275667,2.220263,3912.97044149993
"5",26.5093541176471,−9.275667,2.220263,3230.74455878539
"6",27.5681776470588,−9.275667,2.220263,3844.90678386964
"7",28.6270011764706,−9.275667,2.220263,3589.24120395294
"8",29.6858247058824,−9.275667,2.220263,4397.60594409133
"9",30.7446482352941,−9.275667,2.220263,4091.87145335382
"10",31.8034717647059,−9.275667,2.220263,3629.38881831352
"11",32.8622952941176,−9.275667,2.220263,4004.21654794172
"12",33.9211188235294,−9.275667,2.220263,3677.89309510155
"13",34.9799423529412,−9.275667,2.220263,3403.64263172129
"14",36.0387658823529,−9.275667,2.220263,4037.90824409633
"15",37.0975894117647,−9.275667,2.220263,3586.3601255521
"16",38.1564129411765,−9.275667,2.220263,4143.50357336189
"17",39.2152364705882,−9.275667,2.220263,4438.40528369386
"18",40.27406,−9.275667,2.220263,4223.83899520847
"19",22.27406,−7.94233366666667,2.220263,4161.38687685717
"20",23.3328835294118,−7.94233366666667,2.220263,3859.15646354715
"21",24.3917070588235,−7.94233366666667,2.220263,3584.04512500772
"22",25.4505305882353,−7.94233366666667,2.220263,3972.42554650603
"23",26.5093541176471,−7.94233366666667,2.220263,4080.29959776367
"24",27.5681776470588,−7.94233366666667,2.220263,3777.65612053103
"25",28.6270011764706,−7.94233366666667,2.220263,3586.68559285266
"26",29.6858247058824,−7.94233366666667,2.220263,3722.59194218556
"27",30.7446482352941,−7.94233366666667,2.220263,3496.06390906312
"28",31.8034717647059,−7.94233366666667,2.220263,4346.44652575794
"29",32.8622952941176,−7.94233366666667,2.220263,3864.52739177204
"30",33.9211188235294,−7.94233366666667,2.220263,3773.82468210765
"31",34.9799423529412,−7.94233366666667,2.220263,4187.42519518627
"32",36.0387658823529,−7.94233366666667,2.220263,3930.039504705
"33",37.0975894117647,−7.94233366666667,2.220263,3704.12887582495
"34",38.1564129411765,−7.94233366666667,2.220263,3707.73556202214
"35",39.2152364705882,−7.94233366666667,2.220263,3897.93931776568
"36",40.27406,−7.94233366666667,2.220263,3269.45978792391
"37",22.27406,−6.60900033333333,2.220263,3533.96134475452
"38",23.3328835294118,−6.60900033333333,2.220263,3834.79370510304
"39",24.3917070588235,−6.60900033333333,2.220263,4060.85384724697
"40",25.4505305882353,−6.60900033333333,2.220263,3558.17701803142
"41",26.5093541176471,−6.60900033333333,2.220263,4041.12604569552

```
"42",27.5681776470588,−6.60900033333333,2.220263,3670.61855663598
"43",28.6270011764706,−6.60900033333333,2.220263,3659.34011905515
"44",29.6858247058824,−6.60900033333333,2.220263,3809.37275098688
"45",30.7446482352941,−6.60900033333333,2.220263,3817.26636709652
"46",31.8034717647059,−6.60900033333333,2.220263,3981.42162436518
"47",32.8622952941176,−6.60900033333333,2.220263,3848.04747803804
"48",33.9211188235294,−6.60900033333333,2.220263,3792.85224482378
"49",34.9799423529412,−6.60900033333333,2.220263,3775.69322267457
"50",36.0387658823529,−6.60900033333333,2.220263,3835.34005253286
"51",37.0975894117647,−6.60900033333333,2.220263,3777.55692716696
"52",38.1564129411765,−6.60900033333333,2.220263,3606.57928360672
"53",39.2152364705882,−6.60900033333333,2.220263,3767.54928980844
"54",40.27406,−6.60900033333333,2.220263,3865.65712031409
"55",22.27406,−5.275667,2.220263,3440.02962265698
"56",23.3328835294118,−5.275667,2.220263,3767.32115222855
"57",24.3917070588235,−5.275667,2.220263,3860.49349125588
"58",25.4505305882353,−5.275667,2.220263,3883.01444563373
"59",26.5093541176471,−5.275667,2.220263,3441.13236536825
"60",27.5681776470588,−5.275667,2.220263,3598.08343527195
"61",28.6270011764706,−5.275667,2.220263,3401.68049766056
"62",29.6858247058824,−5.275667,2.220263,3815.73910012576
"63",30.7446482352941,−5.275667,2.220263,3833.40011893992
"64",31.8034717647059,−5.275667,2.220263,4272.8145021548
"65",32.8622952941176,−5.275667,2.220263,3526.81943845304
"66",33.9211188235294,−5.275667,2.220263,3770.5969573454
"67",34.9799423529412,−5.275667,2.220263,3496.88460951649
"68",36.0387658823529,−5.275667,2.220263,3507.08339578911
"69",37.0975894117647,−5.275667,2.220263,3352.71933657886
"70",38.1564129411765,−5.275667,2.220263,3840.2066159345
"71",39.2152364705882,−5.275667,2.220263,4208.27068566475
"72",40.27406,−5.275667,2.220263,3840.6653821053
"73",22.27406,−9.275667,3.55359633333333,3819.67215886521
"74",23.3328835294118,−9.275667,3.55359633333333,3917.7738529566
"75",24.3917070588235,−9.275667,3.55359633333333,4072.61831829341
"76",25.4505305882353,−9.275667,3.55359633333333,3433.87854540389
"77",26.5093541176471,−9.275667,3.55359633333333,3869.01599790049
"78",27.5681776470588,−9.275667,3.55359633333333,3741.84071114098
"79",28.6270011764706,−9.275667,3.55359633333333,4198.07780195584
"80",29.6858247058824,−9.275667,3.55359633333333,4024.32101482734
"81",30.7446482352941,−9.275667,3.55359633333333,3996.70805453016
"82",31.8034717647059,−9.275667,3.55359633333333,3890.88517934793
"83",32.8622952941176,−9.275667,3.55359633333333,3993.03556560088
"84",33.9211188235294,−9.275667,3.55359633333333,3898.73953951586
"85",34.9799423529412,−9.275667,3.55359633333333,3970.4476442705
"86",36.0387658823529,−9.275667,3.55359633333333,4120.79780701667
"87",37.0975894117647,−9.275667,3.55359633333333,3803.27317047175
```

"88",38.1564129411765,−9.275667,3.55359633333333,3588.2206076064
"89",39.2152364705882,−9.275667,3.55359633333333,4088.08747710017
"90",40.27406,−9.275667,3.55359633333333,3603.37436356634
"91",22.27406,−7.94233366666667,3.55359633333333,3703.67098067265
"92",23.3328835294118,−7.94233366666667,3.55359633333333,3749.97406000064
"93",24.3917070588235,−7.94233366666667,3.55359633333333,3934.65143976241
"94",25.4505305882353,−7.94233366666667,3.55359633333333,3784.82913452143
"95",26.5093541176471,−7.94233366666667,3.55359633333333,3666.29494949238
"96",27.5681776470588,−7.94233366666667,3.55359633333333,3782.29401887196
"97",28.6270011764706,−7.94233366666667,3.55359633333333,3609.43735353351
"98",29.6858247058824,−7.94233366666667,3.55359633333333,4040.26418496231
"99",30.7446482352941,−7.94233366666667,3.55359633333333,4407.75387249778
"100",31.8034717647059,−7.94233366666667,3.55359633333333,4152.66804090617
"101",32.8622952941176,−7.94233366666667,3.55359633333333,3806.38012159714
"102",33.9211188235294,−7.94233366666667,3.55359633333333,3733.90874770266
"103",34.9799423529412,−7.94233366666667,3.55359633333333,3918.44868135261
"104",36.0387658823529,−7.94233366666667,3.55359633333333,3496.95004718826
"105",37.0975894117647,−7.94233366666667,3.55359633333333,3979.5089461481
"106",38.1564129411765,−7.94233366666667,3.55359633333333,3615.84908093638
"107",39.2152364705882,−7.94233366666667,3.55359633333333,4015.30255117403
"108",40.27406,−7.94233366666667,3.55359633333333,3588.6358478063
"109",22.27406,−6.60900033333333,3.55359633333333,3827.24588692564
"110",23.3328835294118,−6.60900033333333,3.55359633333333,4265.62650278451
"111",24.3917070588235,−6.60900033333333,3.55359633333333,3886.51359623825
"112",25.4505305882353,−6.60900033333333,3.55359633333333,3381.15110854587
"113",26.5093541176471,−6.60900033333333,3.55359633333333,3817.08271755499
"114",27.5681776470588,−6.60900033333333,3.55359633333333,3692.94506523835
"115",28.6270011764706,−6.60900033333333,3.55359633333333,3950.88359381815
"116",29.6858247058824,−6.60900033333333,3.55359633333333,3737.11204515948
"117",30.7446482352941,−6.60900033333333,3.55359633333333,3779.34517055178
"118",31.8034717647059,−6.60900033333333,3.55359633333333,3367.35216392879
"119",32.8622952941176,−6.60900033333333,3.55359633333333,4029.32852391687
"120",33.9211188235294,−6.60900033333333,3.55359633333333,4148.3767161411
"121",34.9799423529412,−6.60900033333333,3.55359633333333,4037.82748126335
"122",36.0387658823529,−6.60900033333333,3.55359633333333,4032.8102551987
"123",37.0975894117647,−6.60900033333333,3.55359633333333,3512.04841020251
"124",38.1564129411765,−6.60900033333333,3.55359633333333,3735.81864984416
"125",39.2152364705882,−6.60900033333333,3.55359633333333,3296.96877406377
"126",40.27406,−6.60900033333333,3.55359633333333,3466.03688200665
"127",22.27406,−5.275667,3.55359633333333,3984.07913722358
"128",23.3328835294118,−5.275667,3.55359633333333,4405.1260728524
"129",24.3917070588235,−5.275667,3.55359633333333,3462.21156234116
"130",25.4505305882353,−5.275667,3.55359633333333,3359.8955396538
"131",26.5093541176471,−5.275667,3.55359633333333,3958.23127604785
"132",27.5681776470588,−5.275667,3.55359633333333,3920.83125412845
"133",28.6270011764706,−5.275667,3.55359633333333,3987.71302615452

```
"134",29.6858247058824,−5.275667,3.55359633333333,3759.73388747755
"135",30.7446482352941,−5.275667,3.55359633333333,3463.09539676337
"136",31.8034717647059,−5.275667,3.55359633333333,3939.54725070549
"137",32.8622952941176,−5.275667,3.55359633333333,4009.8395705478
"138",33.9211188235294,−5.275667,3.55359633333333,3588.66093580435
"139",34.9799423529412,−5.275667,3.55359633333333,4425.68336576983
"140",36.0387658823529,−5.275667,3.55359633333333,3652.62014138172
"141",37.0975894117647,−5.275667,3.55359633333333,4151.22745228953
"142",38.1564129411765,−5.275667,3.55359633333333,3478.69086765797
"143",39.2152364705882,−5.275667,3.55359633333333,3670.62448671978
"144",40.27406,−5.275667,3.55359633333333,3937.88808282059
"145",22.27406,−9.275667,4.88692966666667,4268.94956845816
"146",23.3328835294118,−9.275667,4.88692966666667,4098.88770697161
"147",24.3917070588235,−9.275667,4.88692966666667,3678.40551892734
"148",25.4505305882353,−9.275667,4.88692966666667,3821.32013479856
"149",26.5093541176471,−9.275667,4.88692966666667,3875.63268755282
"150",27.5681776470588,−9.275667,4.88692966666667,3152.16382968983
"151",28.6270011764706,−9.275667,4.88692966666667,3304.55007658872
"152",29.6858247058824,−9.275667,4.88692966666667,4398.6246217844
"153",30.7446482352941,−9.275667,4.88692966666667,3325.31922755672
"154",31.8034717647059,−9.275667,4.88692966666667,3951.818378854
"155",32.8622952941176,−9.275667,4.88692966666667,4111.08964559568
"156",33.9211188235294,−9.275667,4.88692966666667,3779.96247646739
"157",34.9799423529412,−9.275667,4.88692966666667,3501.2333753125
"158",36.0387658823529,−9.275667,4.88692966666667,4025.50356815597
"159",37.0975894117647,−9.275667,4.88692966666667,3971.71823660515
"160",38.1564129411765,−9.275667,4.88692966666667,3532.15096387576
"161",39.2152364705882,−9.275667,4.88692966666667,3467.29989410426
"162",40.27406,−9.275667,4.88692966666667,3413.29848837523
"163",22.27406,−7.94233366666667,4.88692966666667,3860.83628309608
"164",23.3328835294118,−7.94233366666667,4.88692966666667,3919.69800757301
"165",24.3917070588235,−7.94233366666667,4.88692966666667,3303.92467838092
"166",25.4505305882353,−7.94233366666667,4.88692966666667,3570.35003465755
"167",26.5093541176471,−7.94233366666667,4.88692966666667,3908.3226738068
"168",27.5681776470588,−7.94233366666667,4.88692966666667,3854.24179810414
"169",28.6270011764706,−7.94233366666667,4.88692966666667,3655.86240133312
"170",29.6858247058824,−7.94233366666667,4.88692966666667,3560.62177550793
"171",30.7446482352941,−7.94233366666667,4.88692966666667,4079.00279171102
"172",31.8034717647059,−7.94233366666667,4.88692966666667,3757.71603314915
"173",32.8622952941176,−7.94233366666667,4.88692966666667,3816.29162645146
"174",33.9211188235294,−7.94233366666667,4.88692966666667,4342.91294082955
"175",34.9799423529412,−7.94233366666667,4.88692966666667,3963.63236857927
"176",36.0387658823529,−7.94233366666667,4.88692966666667,3502.93463240182
"177",37.0975894117647,−7.94233366666667,4.88692966666667,3772.17364537547
"178",38.1564129411765,−7.94233366666667,4.88692966666667,3821.82619109705
"179",39.2152364705882,−7.94233366666667,4.88692966666667,3627.82610535594
```

```
"180",40.27406,−7.94233366666667,4.88692966666667,3489.46483657013
"181",22.27406,−6.60900033333333,4.88692966666667,3781.33602019067
"182",23.3328835294118,−6.60900033333333,4.88692966666667,3466.02155109757
"183",24.3917070588235,−6.60900033333333,4.88692966666667,3933.46881218881
"184",25.4505305882353,−6.60900033333333,4.88692966666667,4005.431094966
"185",26.5093541176471,−6.60900033333333,4.88692966666667,4151.38920424356
"186",27.5681776470588,−6.60900033333333,4.88692966666667,3925.4426013306
"187",28.6270011764706,−6.60900033333333,4.88692966666667,3797.22076878493
"188",29.6858247058824,−6.60900033333333,4.88692966666667,3832.97549111459
"189",30.7446482352941,−6.60900033333333,4.88692966666667,4025.5689922924
"190",31.8034717647059,−6.60900033333333,4.88692966666667,3249.50638600278
"191",32.8622952941176,−6.60900033333333,4.88692966666667,3495.79203678686
"192",33.9211188235294,−6.60900033333333,4.88692966666667,3783.36486003405
"193",34.9799423529412,−6.60900033333333,4.88692966666667,3639.11800592684
"194",36.0387658823529,−6.60900033333333,4.88692966666667,3362.41721076315
"195",37.0975894117647,−6.60900033333333,4.88692966666667,3769.4111795612
"196",38.1564129411765,−6.60900033333333,4.88692966666667,3925.83097176394
"197",39.2152364705882,−6.60900033333333,4.88692966666667,3642.0080562244
"198",40.27406,−6.60900033333333,4.88692966666667,3579.13226576786
"199",22.27406,−5.275667,4.88692966666667,4033.78105588058
"200",23.3328835294118,−5.275667,4.88692966666667,3458.37468938799
"201",24.3917070588235,−5.275667,4.88692966666667,3897.99219982044
"202",25.4505305882353,−5.275667,4.88692966666667,3599.01183784925
"203",26.5093541176471,−5.275667,4.88692966666667,3772.79745086719
"204",27.5681776470588,−5.275667,4.88692966666667,4197.5978292073
"205",28.6270011764706,−5.275667,4.88692966666667,3693.28999896271
"206",29.6858247058824,−5.275667,4.88692966666667,3375.31485995065
"207",30.7446482352941,−5.275667,4.88692966666667,3467.63529457246
"208",31.8034717647059,−5.275667,4.88692966666667,3916.86389935765
"209",32.8622952941176,−5.275667,4.88692966666667,3329.00633293894
"210",33.9211188235294,−5.275667,4.88692966666667,4341.43424981962
"211",34.9799423529412,−5.275667,4.88692966666667,3433.39165985745
"212",36.0387658823529,−5.275667,4.88692966666667,3968.45659232138
"213",37.0975894117647,−5.275667,4.88692966666667,4545.96851006223
"214",38.1564129411765,−5.275667,4.88692966666667,3718.70920263392
"215",39.2152364705882,−5.275667,4.88692966666667,4440.80966670215
"216",40.27406,−5.275667,4.88692966666667,3859.2762314422
"217",22.27406,−9.275667,6.220263,3293.16007021132
"218",23.3328835294118,−9.275667,6.220263,3765.673334226
"219",24.3917070588235,−9.275667,6.220263,3720.6771466343
"220",25.4505305882353,−9.275667,6.220263,3660.49211304415
"221",26.5093541176471,−9.275667,6.220263,3687.6137838805
"222",27.5681776470588,−9.275667,6.220263,3453.59557942124
"223",28.6270011764706,−9.275667,6.220263,4144.71368530403
"224",29.6858247058824,−9.275667,6.220263,3397.26774153567
"225",30.7446482352941,−9.275667,6.220263,4309.79465197503
```

"226",31.8034717647059,−9.275667,6.220263,3695.91515593961
"227",32.8622952941176,−9.275667,6.220263,3684.69109694084
"228",33.9211188235294,−9.275667,6.220263,3925.55047300086
"229",34.9799423529412,−9.275667,6.220263,4021.54077276006
"230",36.0387658823529,−9.275667,6.220263,3831.87142490457
"231",37.0975894117647,−9.275667,6.220263,3548.13332899032
"232",38.1564129411765,−9.275667,6.220263,4502.16393147718
"233",39.2152364705882,−9.275667,6.220263,3936.18017937752
"234",40.27406,−9.275667,6.220263,3638.46583836331
"235",22.27406,−7.94233366666667,6.220263,4046.67200041596
"236",23.3328835294118,−7.94233366666667,6.220263,3851.53489640226
"237",24.3917070588235,−7.94233366666667,6.220263,4199.74435520469
"238",25.4505305882353,−7.94233366666667,6.220263,3769.56756496401
"239",26.5093541176471,−7.94233366666667,6.220263,3410.90716629477
"240",27.5681776470588,−7.94233366666667,6.220263,3947.76536547955
"241",28.6270011764706,−7.94233366666667,6.220263,3574.91173831706
"242",29.6858247058824,−7.94233366666667,6.220263,4033.56405422046
"243",30.7446482352941,−7.94233366666667,6.220263,3861.24400993772
"244",31.8034717647059,−7.94233366666667,6.220263,4034.64133862447
"245",32.8622952941176,−7.94233366666667,6.220263,3721.11840529665
"246",33.9211188235294,−7.94233366666667,6.220263,3957.10996746594
"247",34.9799423529412,−7.94233366666667,6.220263,3925.14527996633
"248",36.0387658823529,−7.94233366666667,6.220263,3762.65618738735
"249",37.0975894117647,−7.94233366666667,6.220263,3331.22571006309
"250",38.1564129411765,−7.94233366666667,6.220263,4506.04101858929
"251",39.2152364705882,−7.94233366666667,6.220263,3998.51455390227
"252",40.27406,−7.94233366666667,6.220263,3613.88184661507
"253",22.27406,−6.60900033333333,6.220263,3118.51566256873
"254",23.3328835294118,−6.60900033333333,6.220263,3909.09587001228
"255",24.3917070588235,−6.60900033333333,6.220263,3628.02227764186
"256",25.4505305882353,−6.60900033333333,6.220263,4349.68611321502
"257",26.5093541176471,−6.60900033333333,6.220263,4191.63644092349
"258",27.5681776470588,−6.60900033333333,6.220263,4380.0106812576
"259",28.6270011764706,−6.60900033333333,6.220263,3997.78185509683
"260",29.6858247058824,−6.60900033333333,6.220263,3958.4699767783
"261",30.7446482352941,−6.60900033333333,6.220263,3765.40366403316
"262",31.8034717647059,−6.60900033333333,6.220263,4135.12895169422
"263",32.8622952941176,−6.60900033333333,6.220263,3920.38954684746
"264",33.9211188235294,−6.60900033333333,6.220263,3320.0926305679
"265",34.9799423529412,−6.60900033333333,6.220263,3765.69436370242
"266",36.0387658823529,−6.60900033333333,6.220263,3743.14256154127
"267",37.0975894117647,−6.60900033333333,6.220263,4221.88570073581
"268",38.1564129411765,−6.60900033333333,6.220263,4009.4505436881
"269",39.2152364705882,−6.60900033333333,6.220263,4037.19935756619
"270",40.27406,−6.60900033333333,6.220263,3782.70792472153
"271",22.27406,−5.275667,6.220263,4228.18346449646

```
"272",23.3328835294118,−5.275667,6.220263,3368.93016188993
"273",24.3917070588235,−5.275667,6.220263,3635.85031470552
"274",25.4505305882353,−5.275667,6.220263,3512.58318696443
"275",26.5093541176471,−5.275667,6.220263,3563.08452118044
"276",27.5681776470588,−5.275667,6.220263,3711.52357976954
"277",28.6270011764706,−5.275667,6.220263,4000.47334075531
"278",29.6858247058824,−5.275667,6.220263,3594.36556261869
"279",30.7446482352941,−5.275667,6.220263,3786.34631797941
"280",31.8034717647059,−5.275667,6.220263,3689.38782727611
"281",32.8622952941176,−5.275667,6.220263,3676.68511564048
"282",33.9211188235294,−5.275667,6.220263,3406.99345664129
"283",34.9799423529412,−5.275667,6.220263,3773.53676882671
"284",36.0387658823529,−5.275667,6.220263,3801.76654613562
"285",37.0975894117647,−5.275667,6.220263,4008.42516839755
"286",38.1564129411765,−5.275667,6.220263,3561.34236823162
"287",39.2152364705882,−5.275667,6.220263,4210.83689102618
"288",40.27406,−5.275667,6.220263,3644.84983605997
```

## Anhang G : Inhaltsverzeichnis der beiliegenden CD

CD-ROM

- ↳ Verzeichnis: Masterarbeit
  - ↳ PDF der Masterarbeit
- ↳ Demo
  - ↳ ausführbares Programm inklusive Demodaten
- ↳ Verzeichnis: Drone Visualisation (Unity-Projekt)
  - ↳ Verzeichnis: Assets
    - ↳ Verzeichnis: Scripts
      - ↳ Manager-Klassen
    - ↳ MainWindow.unity (Projektdatei)

## Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit mit dem Titel „Prototypische Entwicklung einer Software zur Planung, Visualisierung und Dokumentation UAV-gestützter Gasfreimessungen in Ballastwassertanks von Schiffen unter Verwendung der Unity 3D Game Engine“ selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

| | |
|---|---|
| Ort, Datum | Unterschrift |