

Aus dem Institut für Technische Informatik der Universität zu Lübeck

Direktor:
Prof. Dr.-Ing. Erik Maehle

Echtzeitfähige Simulation von schwarmfähigen autonomen Unterwasserrobotern für Umweltmonitoring

Inauguraldissertation
zur
Erlangung der Doktorwürde (Dr.-Ing.)
der Universität zu Lübeck
- Aus der Sektion Informatik / Technik -

Vorgelegt von
Dipl.-Inf. Thomas Tosik
aus Gdynia, Polen

Lübeck 2016



Thomas Tosik
aus Gdynia, Polen,
geb. am 11. November 1983

1. Berichterstatter: Prof. Dr.-Ing. Erik Maehle
2. Berichterstatter: Prof. Dr. Stefan Fischer

Tag der mündlichen Prüfung: 02.03.2016

Zum Druck genehmigt: Lübeck, den 8. März 2016

Danksagung

Ein großer Dank gebührt meinem Doktorvater Prof. Dr.-Ing. Erik Maehle, der mir die Gelegenheit gegeben hat, am Institut meinem Forschungsthema nachzugehen und in Form einer Dissertation zu Papier zu bringen.

Ein herzlicher Dank gilt auch Herrn Prof. Dr. Stefan Fischer für die Übernahme des Koreferats.

Bedanken möchte ich mich ebenfalls bei den Studierenden Jasper Schwinghammer, Mandy Jane Feldvoß, Arne Brech, Fabian Uken, Fabian Busse, John Paul Jonte, die mich mit ihrer Arbeit unterstützt haben. Ebenfalls möchte ich allen weiteren Studenten danken, die ich während meiner Zeit in den verschiedensten Projekten betreut habe und die mich immer wieder überrascht haben.

Darüber hinaus danke ich allen Kollegen am Institut, mit denen immer ein guter Umgang möglich war und auch wirklich ehrlich nur vereinzelte Feierabendbiere. Besonders hervorheben möchte ich Jan Frost und Dariush Forouher. Die fachlichen Diskussionen, gemeinsamen Projekte und der zeitlich nah beieinander liegende Beginn unserer Dissertationen waren sehr wertvoll für mich, ebenso wie Brettspielabende und weitere Aktivitäten außerhalb des Instituts. Diesen beiden und Patrick Kurschies danke ich auch für Korrekturhilfe. Des Weiteren danke ich meinem Kollegen Ammar Amory für die Verwendung von MARS und wertvolles Feedback. Ebenfalls danke ich meinem ehemaligen Betreuer Marek Litza, der mir die Chance zur Diplomarbeit gegeben hat, die dieses ganze Projekt ins Rollen gebracht hat.

Meinen Eltern und Freunden gilt der abschließende Dank. Meinen Eltern dafür, dass sie mir das Studium ermöglicht haben, und meinen Freunden für viel weitere Unterstützung.

Zusammenfassung

Wasser stellt eine wichtige Ressource für den Menschen dar und bedeckt 71 Prozent unseres Planeten. Der größte Teil befindet sich außerhalb der direkten menschlichen Wahrnehmung unter der Wasseroberfläche. Die Erforschung und das Überwachen der Gewässer unseres Planeten ist eine schwierige und wichtige Aufgabe, die zunehmend durch autonome robotische Systeme ausgeführt wird. Speziell autonome Unterwasserfahrzeuge (AUV) erlauben es, unter der Wasseroberfläche Untersuchungen durchzuführen. Die Anforderungen an solche Systeme sind hoch und die Umwelt risikoreich. Da solche Systeme aus komplexen Softwarekomponenten bestehen, ist es erforderlich diese ausgiebig zu testen. Dies kann auf verschiedenen Ebenen geschehen. Eine Lösung des Problems ist dabei die Simulation solcher Systeme.

Simulationen sind seit den Anfängen moderner Computer ein Forschungsthema. In den letzten Jahrzehnten entstanden Simulatoren für autonome Robotersysteme und für AUVs. Während am Anfang die Simulation von Teilbereichen wichtig war, wie z.B. Thrustermotormodellierung, verschob sich der Fokus mehr zur Simulation ganzer Systeme in einer Umwelt. Allerdings beziehen sich die Simulationen und die erstellten Modelle häufig auf spezifische AUVs und die Verwendung anderer AUVs ist aufwändig. Die zugrunde liegenden Modelle unterscheiden sich stark untereinander und fokussieren sich auf unterschiedliche Teilbereiche. In der Industrie dominiert hingegen das Training von Personal bei Remotely Operated Vehicles (ROV).

In dieser Arbeit wird eine Simulationsumgebung für autonome Unterwasserroboter vorgestellt. Dabei haben die zwei Teilaspekte, Echtzeitfähigkeit und Schwärme, eine wichtige Bedeutung. Für den ersten Teilaspekt Echtzeitfähigkeit wurde ein Modell entwickelt, welches versucht Echtzeitfähigkeit und Realitätsnähe zu verbinden. Die Simulation bildet dabei die reale Welt zeitlich ab, ohne den Bezug zur Genauigkeit zu verlieren. Folglich müssen für die Berechnung der physikalischen Größen und Kräfte entsprechende Algorithmen verwendet und entwickelt werden. Ein weiterer Teilaspekt dieser Arbeit ist die Realisierung der Simulation mehrerer Teilnehmer in Form von Schwärmen. Dazu gehören Bereiche wie die Kommunikation über Unterwassermodems oder der Stromverbrauch. Aus der Simulationsumgebung ergibt sich eine Visualisierung, die für verschiedene Sensoren, etwa Kameras, verwendet wird, aber auch ein direktes visuelles Feedback der Umwelt und Gesamtsituation für den Benutzer ermöglicht.

Die Auswertung der Simulationsumgebung erfolgt über eine Performanz-Analyse und einem Vergleich von realen und simulierten Ergebnissen anhand einer Tiefenregelung. Für die Benutzungsschnittstelle wird eine benutzerorientierte Evaluation durchgeführt.

Abstract

Water is an important resource for humans and covers 71 percent of our planet. The water bodies elude the direct human perception because the vision under water is blocked. The exploration and monitoring of the water bodies of our planet are therefore, a complex and important task. This task is increasingly often conducted by autonomous robotic systems. Specifically, Autonomous Underwater Vehicles (AUVs) allow for the examination of the underwater environment. The requirements for such systems are high and the environment is risky. Since AUVs consist of complex software components, it is necessary to evaluate them extensively. One solution is the simulation of such systems.

Simulations have been an active research topic for many years. In the last decades, many simulators were developed for autonomous robotic systems and AUVs. While in the beginning the simulation of specific parts, such as thrusters, was important, the focus shifted to the simulation of whole systems in their environment. However, most models and simulations are developed for a specific AUV and the use of other AUVs requires an additional effort. The underlying models vary greatly among themselves and focus only on specific parts. The efficient simulation of several AUVs in the form of swarms is still an active topic. The industry, however is dominated by simulators for training of operators of remotely operated vehicles (ROVs).

This thesis presents a simulation environment for autonomous underwater vehicles. Real-time capabilities and swarms are most important. To cover real-time capabilities, a model was developed to combine real-time and realistic simulation. The simulation reproduces the real world without losing too much precision. Therefore, appropriate algorithms have to be developed and used to calculate the physical parameters and forces. Another important topic of this thesis is the swarm capability and the simulation of multiple members. This covers topics such as under water communication through under water modems and energy consumption. From the simulation environment results a visualization, which is used for sensors such as cameras. It further gives the user a direct visual feedback of the overall situation.

The evaluation is carried out through a performance analysis and a comparison between real and simulated results by means of a depth control experiment. For the user interface, a user-oriented evaluation is conducted.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung und Abgrenzung	2
1.3. Beitrag und Aufbau der Arbeit	3
2. Stand der Forschung und Technik	5
2.1. Definition	6
2.2. Historische Entwicklung	7
2.3. Klassifikation und Modellstruktur	8
2.4. Vorarbeiten	11
2.5. Stand der Forschung	13
2.5.1. Ältere Simulatoren	13
2.5.2. Aktueller Stand	14
2.6. Aktueller Stand der Technik	21
2.7. Zusammenfassung	22
2.7.1. Eigenschaften	22
2.7.2. Fortschritte zu älteren Simulatoren	27
2.7.3. Anforderungen und Unterschiede	28
3. Grundlagen	31
3.1. AUV	31
3.1.1. HANSE	33
3.1.2. MONSUN	35
3.2. Fluidmechanik	36
3.2.1. Hydrostatik	37
3.2.1.1. Auftrieb	37
3.2.1.2. Stabilität	39
3.2.2. Strömungslehre	39
3.3. Unterwasserakustik	42
4. Modell	45
4.1. AUV	46
4.2. Sensoren	50
4.2.1. CTD-Sensor	50
4.2.2. Sonar	53
4.2.3. Kamera	55

4.2.4.	Mapping Sensor	56
4.3.	Aktoren	56
4.3.1.	Thruster	56
4.3.2.	Ballasttank	58
4.4.	Rauschen und Ausfall	60
4.5.	Umwelt	60
4.5.1.	Vegetation	61
4.5.2.	Fischschwärme	63
4.5.3.	Strömung und Wellen	67
4.6.	Schwarm	68
4.6.1.	Kommunikation	69
4.6.2.	Energie	73
5.	Prototypische Simulationsumgebung MARS	77
5.1.	Architektur	77
5.2.	Implementierung	81
5.2.1.	Verwendete Technologien	81
5.2.1.1.	jMonkeyEngine 3	81
5.2.1.2.	Bullet Physics Library	82
5.2.1.3.	NetBeans Platform	82
5.2.1.4.	ROS und RosJava	83
5.2.2.	Szenegraph	83
5.2.3.	XML-Konfiguration	84
5.2.4.	Benutzungsschnittstelle	87
6.	Evaluation	93
6.1.	Echtzeitfähigkeit	94
6.1.1.	Testaufbau	94
6.1.2.	Grafischer Benchmark	95
6.1.2.1.	Auswertung	96
6.1.3.	AUV-Simulation	99
6.1.3.1.	Auswertung	99
6.1.4.	Kommunikation	103
6.1.4.1.	Auswertung	103
6.1.5.	Fazit	110
6.2.	Vergleich zwischen Simulation und Realität	111
6.2.1.	Tiefenregelung	111
6.2.2.	Sonar	113
6.3.	Schwärme	118
6.3.1.	Formationsfahrt	118
6.3.2.	Load-Balancing Verhalten	119
6.4.	Benutzungsschnittstelle	121
6.4.1.	Durchführung	121

6.4.2. Auswertung	122
7. Zusammenfassung und Ausblick	129
7.1. Zusammenfassung	130
7.2. Ausblick	132
A. Anhang	133
A.1. Fragebogen	135
A.2. Anleitung	140
A.3. AUV Parameterliste	142
A.4. Umweltparameter	143
A.5. Sensorliste	144
Abbildungsverzeichnis	i
Tabellenverzeichnis	vii
Literaturverzeichnis	ix
Eigene Publikationen	xxv

Kapitel 1.

Einleitung

Tu es oder tu es nicht! Es gibt kein Versuchen!

(Yoda - Star Wars)

Inhaltsangabe

1.1. Motivation	1
1.2. Zielsetzung und Abgrenzung	2
1.3. Beitrag und Aufbau der Arbeit	3

Dieses Kapitel beschäftigt sich mit der Motivation für diese Arbeit, dem Ziel und dem groben Aufbau. In Abschnitt 1.1 wird auf die Motivation der Arbeit eingegangen. Es wird dargestellt, warum die Simulation von AUVs wichtig ist. Abschnitt 1.2 definiert die Ziele, die erreicht werden sollen, und grenzt sich zu anderen Themengebieten ab. Im letzten Abschnitt 1.3 wird die Struktur der Arbeit erläutert und auf die Inhalte der einzelnen Kapitel eingegangen.

1.1. Motivation

Wasser ist für das Leben des Menschen unabdingbar. Der menschliche Körper benötigt Wasser und ist direkt und indirekt abhängig von dem Ökosystem innerhalb der Gewässer. Die Fische dienen als Nahrung und weitere komplexe Nahrungsketten wirken sich auf den Menschen aus. Die Bedeutung der Gewässerqualität äußert sich z.B. in der Wasserrahmenrichtlinie (Richtlinie 2000/60/EG) der EU [euw]. Obwohl 71 Prozent der Erde von Wasser bedeckt ist, entzieht sich ein Großteil der Überwachung. Autonome Unterwasserfahrzeuge (AUV) und USVs sind eine Möglichkeit, um die Erforschung, Überwachung und Kartografierung von Gewässern zu ermöglichen. Anwendungsbereiche sind Hydrografie, Schiffshülleninspektion, Exploration in der Öl- und Gasindustrie und Minenabwehr [NH08, Man08]. Die Entwicklung und Evaluation von AUVs erfordert einen hohen Zeitaufwand und ist mit Risiken verbunden. Bereits Brutzman stellt fest, dass [Bru95]:

“A critical bottleneck exists in Autonomous Underwater Vehicle (AUV) design and development. It is tremendously difficult to observe, communicate with

and test underwater robots, because they operate in a remote and hazardous environment where physical dynamics and sensing modalities are counterintuitive.”

Mehrere Personen sind in dem Entwicklungsprozess involviert und häufig steht aufgrund der Kosten von 50.000\$ - 5 Mio\$ nur ein System zur Entwicklung zur Verfügung [New]. Dies führt zu Zeitkonflikten, da bestimmte Tests, wie z.B. Verhalten, ein komplett funktionierendes reales System voraussetzen. Währenddessen müssen andere Entwicklungsprozesse in den Hintergrund treten. Das Testen selbst stellt ein großes Problem dar, da das AUV zu einem entsprechenden Testareal transportiert werden muss, ebenso alle Werkzeuge zur Unterstützung und Personal. Dies erhöht das Risiko von Schäden. Da diese Systeme sich in Entwicklung befinden, können Probleme wie z.B. Softwarefehler auftreten. Dies kann zu Beschädigungen oder Totalverlust des Systems führen. Wasser selbst ist dabei eine schwierige Umgebung, die sich ständig im Fluss befindet. Die Tiefe und der daraus resultierende Druck stellt eine weitere Bedrohung für bestimmte Szenarien dar.

Eine Lösung für solche Probleme ist die Simulation von AUVs [BKZ92]. Es existieren zahlreiche Simulationsumgebungen für die verschiedensten Robotersysteme, allerdings werden die Besonderheiten der Umgebung Wasser außer acht gelassen [CMBG07]. In den letzten Jahren entstanden daraufhin spezialisierte Simulatoren für AUVs [MKT08, RCREF04, CVL14]. Diese unterscheiden sich erheblich voneinander und setzen unterschiedliche Schwerpunkte. Des Weiteren werden bestimmte Aspekte in den Modellen vernachlässigt oder vereinfacht.

In den letzten Jahren wurden Schwarmkonzepte auf AUVs übertragen, da diese Vorteile mit sich bringen. Mehrere AUVs können ihre Aufgaben schneller erledigen und sind als Schwarm toleranter gegenüber Einzelverlusten. Allerdings gelten für Schwärme besondere Anforderungen und folglich für deren Simulation [DS04]. Das sind vor allem die effiziente Simulation von mehreren AUVs und Kommunikationsmöglichkeiten. Um mit Schwärmen Umweltmonitoring zu realisieren, muss die Umwelt simuliert und visuell dargestellt werden.

1.2. Zielsetzung und Abgrenzung

Ziel dieser Arbeit ist die Entwicklung eines Modells für AUVs, um Schwärme im Hinblick auf Umweltmonitoring zu simulieren, ohne die Echtzeitfähigkeit des Simulators außer acht zu lassen. Dies betrifft Bereiche wie Kommunikation und Stromverbrauch, die wichtig sind für Schwärme, sowie das zuverlässige Simulieren von mehreren AUVs. Die Simulation soll online (Echtzeitfähigkeit) und Hardware-in-the-Loop unterstützen, weil dadurch die Systeme genauer evaluiert werden können. Allerdings müssen das Modell und die Implementierung dies berücksichtigen. Die Schwärme sollen im Kontext des Umweltmonitoring eingesetzt werden, so dass ein entsprechendes Modell für die Umwelt entwickelt werden muss, um z.B. Fische und Vegetation zu simulieren. Das Gesamtmodell soll durch eine prototypische Simulationsumgebung implementiert und evaluiert werden.

Die Arbeit soll sich nicht mit der exakten Simulation des Fluids, z.B. über CFD (engl.

für computational fluid dynamics), in dem sich die AUVs bewegen, befassen. Dies würde die Echtzeitfähigkeit beeinträchtigen. Ebenso sollen durch die Hardware-in-the-Loop Simulation keine elektrischen Signale für die Sensoren oder Aktoren erzeugt werden.

1.3. Beitrag und Aufbau der Arbeit

In dieser Arbeit wird ein Simulator vorgestellt, der aus einem echtzeitfähigen Modell besteht, das explizit für AUV-Schwärme im Umweltmonitoring konzipiert ist. Dabei werden nicht nur einzelne direkte Parameter der Umwelt erfasst, wie z.B. Temperatur, sondern auch indirekte Bioindikatoren wie Fische und Pflanzen. Die Neuerung besteht aus dem Zusammenspiel verschiedener Komponenten, die dies zum ersten mal ermöglichen. Dabei wird in den einzelnen Komponenten auf bewährte Lösungen zurückgegriffen, die unter Umständen an die Anforderungen der Echtzeitfähigkeit angepasst werden. Die Beiträge sind im Einzelnen:

1. Ein modularer quelloffener Simulator basierend auf einer zentralen Architektur.
2. Eine gebrauchstaugliche und erweiterbare grafische Benutzungsschnittstelle.
3. Ein modulares Modell, das vom Nutzer eingestellt werden kann, und aus folgenden wichtigen Einzelmodellen besteht:
 - a) Physikalisches AUV Modell, um eine Vielzahl von AUVs performant in Schwärmen zu simulieren.
 - b) Strahlenbasiertes Kommunikationsmodell für die Unterwasserkommunikation in Schwärmen.
 - c) Stromverbrauch und Stromgewinnungsmodell, um längere Szenarien zu optimieren.
 - d) Umweltmodell, bestehend aus auf Craig-Reynolds basierenden Fischeschwärmen und Vegetation für Umweltmonitoring.

Die Arbeit gliedert sich in drei Teilbereiche. Im ersten Bereich wird auf die historische Entwicklung von Simulatoren im Allgemeinen und speziellere Simulatoren für AUVs eingegangen. Zusätzlich werden Grundlagen von AUVs und Unterwasserphysik behandelt. Der zweite Bereich befasst sich mit dem entwickelten Modell. Der letzte Teilbereich zeigt eine prototypische Implementierung und deren Evaluation. Abgeschlossen wird mit einer Zusammenfassung der gesamten Arbeit und einem Ausblick. Im Detail befassen sich die Kapitel mit folgendem:

Teil I

Kapitel 2 behandelt den Stand der Forschung und Technik. Zuerst wird der Begriff Simulation definiert. Daraufhin wird auf die historische Entwicklung von Simulationen eingegangen, speziell der AUV-Simulationen. Es folgt eine Klassifikation der Simulationssysteme und Anforderungen an eine echtzeitfähige Simulation von Schwarm-AUVs im Umweltmonitoring. Darauf aufbauend wird der aktuelle Stand der Forschung und Technik hinsichtlich der Anforderungen betrachtet. Es wird mit einer Zusammenfassung abgeschlossen.

Kapitel 3 geht auf Grundlagen ein, die für das Modell und die Implementierung benötigt werden. Am Anfang steht das AUV als Basis der Simulation mit zwei konkreten Ausprägungen in Form von HANSE und MONSUN. Diese werden in der Simulation verwendet und modelliert. Ein weiterer Punkt sind physikalische Grundlagen, besonders im Bereich der Hydrostatik, Strömungslehre und Unterwasserakustik.

Teil II

Kapitel 4 stellt das Modell vor. Es wird auf die einzelnen Komponenten eingegangen und wie sie simuliert werden. Dies umfasst das AUV, seine Sensoren und Aktoren, die Umwelt sowie für Schwärme, Kommunikation und Stromverbrauch.

Teil III

Kapitel 5 befasst sich mit der Implementierung der prototypischen Simulationsumgebung MARS. Dabei wird das Modell aus Kapitel 4 umgesetzt und gezeigt, dass eine effiziente Implementierung möglich ist. Es wird auf den Aufbau, die verwendeten Technologien und die Benutzungsschnittstelle eingegangen.

Kapitel 6 evaluiert die im Kapitel 5 implementierte Simulationsumgebung MARS und das ihr zugrunde liegende Modell aus Kapitel 4. Die Evaluationskriterien sind dabei die Echtzeitfähigkeit durch Performanzanalyse, Vergleiche zwischen simulierten und realen Ergebnissen über eine Tiefenregelung, Realitätsnähe von Schwarmverhalten und die Benutzungsschnittstelle mit einer benutzerorientierten Evaluation.

Kapitel 7 fasst abschließend die Arbeit zusammen. Es werden mögliche Ansätze besprochen, um die Simulation hinsichtlich Schwarm, Genauigkeit und Echtzeitfähigkeit zu verbessern.

Kapitel 2.

Stand der Forschung und Technik

Wer sich nicht an die Vergangenheit erinnern kann, ist dazu verdammt, sie zu wiederholen.

(George Santayana)

Inhaltsangabe

2.1. Definition	6
2.2. Historische Entwicklung	7
2.3. Klassifikation und Modellstruktur	8
2.4. Vorarbeiten	11
2.5. Stand der Forschung	13
2.5.1. Ältere Simulatoren	13
2.5.2. Aktueller Stand	14
2.6. Aktueller Stand der Technik	21
2.7. Zusammenfassung	22
2.7.1. Eigenschaften	22
2.7.2. Fortschritte zu älteren Simulatoren	27
2.7.3. Anforderungen und Unterschiede	28

In diesem Kapitel wird auf den Stand der Technik und Forschung eingegangen. Abschnitt 2.1 beschäftigt sich mit der Herkunft des Wortes Simulation. Es werden unterschiedliche Definitionen des Begriffs diskutiert und welche wesentlichen Eigenschaften eine Simulation erfüllen muss. Abschnitt 2.2 geht auf die historische Entwicklung von Simulationen ein. Begonnen wird mit Flugzeugsimulatoren und dem Fermi-Pasta-Ulam-Experiment. Daraufhin werden erste AUV-Simulationen betrachtet. Abschnitt 2.3 stellt ein Klassifikationssystem vor und aus welchen Komponenten AUV-Simulationen bestehen. Abschnitt 2.4 beschreibt Vorarbeiten in Form einer Diplomarbeit, die im Rahmen dieser Dissertation verwendet wird. Abschnitt 2.5 geht auf den Stand der Forschung ein und beschreibt AUV Simulationen der letzten 20 Jahre. Abschnitt 2.6 beschäftigt sich mit dem Stand der Technik. Dabei werden Simulatoren betrachtet, die von der Wirtschaft entwickelt und eingesetzt werden.

Hierbei handelt es sich zu meist um ROV- und Personaltrainingssimulationen. Der letzte Abschnitt 2.7 fasst die Erkenntnisse des Kapitels zusammen und stellt die Eigenschaften der Simulatoren in Tabellenform dar.

2.1. Definition

Da in dieser Arbeit eine Simulation erstellt wird, ist zunächst die Definition des Begriffs Simulation zu klären und die des verwandten Begriffs des Modells. Das Wort Simulation entstammt dem Lateinisch Wort *simulatio* und bedeutet so viel wie *Vorspiegelung* [Klu75]. In der deutschen Sprache wird unter Simulation laut [bro73] S.444 verstanden:

“Simulation: Sammelbegriff für die Darstellung oder Nachbildung physikalischer, techn., biolog., psycholog. oder ökonom. Prozesse oder Systeme durch mathemat. oder physikal. Modelle, wobei die Untersuchung des Modells einfacher, billiger oder ungefährlicher ist als die des Originals und die Erkenntnisse Rückschlüsse auf die Eigenschaften des Originals erlauben.”

Die Simulation versucht somit die Realität in ein Modell zu überführen, das einfacher zu untersuchen ist als diese selbst. Es lassen sich trotzdem Rückschlüsse auf die Realität ableiten.

Eine weitere Definition des Begriffs Simulation liefert uns [Shu60]. Demnach ist das Ziel einer Simulation:

“...ein Modell [zu konstruieren], das für Manipulationen zugänglich ist, die man an dem Original, das es darstellt, nicht durchführen kann, weil dies unmöglich, zu kostspielig oder nicht praktikabel wäre. Die Funktionsweise des Modells kann untersucht werden, und es lassen sich Schlussfolgerungen über die Verhaltenseigenschaften des wirklichen Systems oder eines Subsystems ziehen.”¹

Zusammengefasst bedeuten die Definitionen, dass einer Simulation ein Modell zu Grunde liegt. Dieses Modell erlaubt es, Untersuchungen durchzuführen, die kostengünstiger, einfacher und ungefährlicher sind als in der Realität. Die Ergebnisse lassen sich dann ganz oder teilweise auf die Realität übertragen.

Die Simulation selbst wird durch ein Simulator ausgeführt und ist laut [ZPK00] S.30:

“..., a model needs some agent capable of actually obeying the instructions and generating behaviour. We call such an agent a simulator. Thus, a simulator is any computation system (such as a single processor, a processor network, the human mind, or more abstractly an algorithm) capable of executing a model to generate its behaviour.”

¹zitiert nach [Shu60] S.909, übersetzt in [Neh02] S.170 von Claudia Nehmzow

Zusammengefasst ist ein Simulator ein Agent, z.B. ein Algorithmus oder Software, die die Instruktionen des Modells umsetzt und daraus ein Verhalten generiert.

Eng angelehnt an den Begriff der Simulation ist der des Modells. Ein Modell definiert sich nach [BBZP13] S.5 folgendermaßen:

“Unter einem Modell versteht man allgemein ein (vereinfachendes) Abbild einer (partiellen) Realität.”

Das Modell ist somit eine mathematische oder physikalische Beschreibung eines Teils der Realität. Das Modell bildet somit die Basis für die Simulation, die durch einen Simulator ausgeführt wird. Ein Kernpunkt von Simulationen ist, in wie weit ein Modell die Realität vereinfacht. Umso einfacher das Modell definiert wird, umso leichter lässt es sich durch einen Computer simulieren. Allerdings erhöht sich dadurch die Ungenauigkeit der Ergebnisse. Man muss folglich einen Kompromiss zwischen Genauigkeit und Geschwindigkeit finden.

Bezogen auf diese Dissertation stellt der in Kapitel 5 vorgestellte MARS den Simulator dar. Dieser nutzt das in Kapitel 4 vorgestellte Modell um AUVs in Schwärmen zu simulieren.

2.2. Historische Entwicklung

Der Vorteil von technischen Simulationen wurde schon früher erkannt. So wurden besonders im aeronautischen Bereich Flugzeugsimulatoren eingesetzt, um die Piloten auf das Fliegen vorzubereiten. Bereits im ersten Weltkrieg gab es erste Versuche in diesem Bereich. Als der erste erfolgreich funktionsfähige Flugsimulator wird der *Link Trainer* angesehen [BA13]. Die Entwicklung von Flugsimulatoren setzte sich im zweiten Weltkrieg fort. So wurde zum Beispiel der *Silloth Trainer* entwickelt und eingesetzt [RS86]. Mit Hilfe von pneumatischen Druckventilen konnte dadurch der Flug von verschiedenen Flugzeugmodellen erprobt werden und die Geräuschkulisse reproduziert werden. Allerdings waren die Simulationen mechanisch und dienten nur zum Training der Piloten.

Die US-Armee hatte während des zweiten Weltkriegs einen großen Bedarf an ballistischen Tabellen um Geschossbahnen von verschiedenen Geschützen unter verschiedenen Witterungsbedingungen zu berechnen. Diese Arbeit wurde anfangs von Hand durchgeführt, bis durch das Auftauchen von Rechenmaschinen wie dem ENIAC (Electronic Numerical Integrator and Computer) eine erhebliche Geschwindigkeitssteigerung ermöglicht wurde [Woo94]. So wurde die Berechnungszeit von 15 Minuten auf 60 Sekunden verringert.

Als erste Computersimulation wird im allgemeinen das Fermi-Pasta-Ulam-Experiment [Fer55] [DPR05] (auch bekannt als Fermi-Pasta-Ulam-Tsingou problem [Dau08]) angesehen, das 1955 in einem Bericht des Los Alamos Scientific Laboratory erwähnt wird. Das Experiment wurde auf dem Computer MANIAC (MATHematical Numerical Integrator And Computer) durchgeführt, der für Berechnungen bei der Entwicklung der Wasserstoffbombe verwendet wurde. Das Modell bestand aus einer Reihe von Massen, die über nicht-lineare Federn verbunden waren. Es wurde erwartet, dass diese Kette nach einer gewissen Zeit in einen Gleichgewichtszustand kommt, nachdem sie initial mit Energie angestoßen wurde.

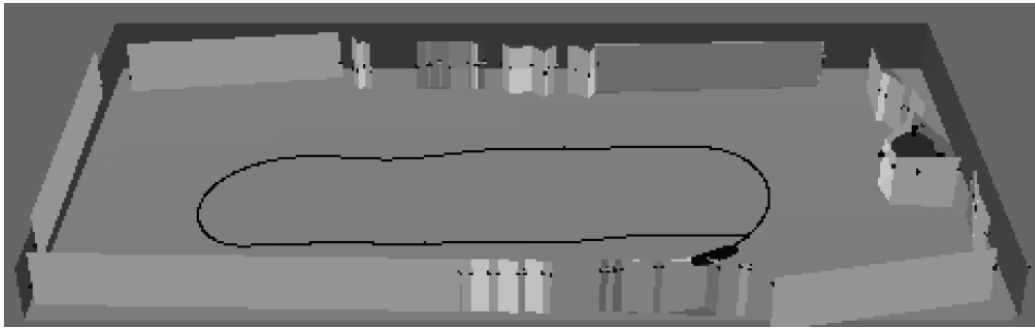


Abbildung 2.1.: Dreidimensionale Visualisierung der Simulation. Bild aus [BKZ92], mit freundlicher Genehmigung von Michael Zyda.

Die Simulation zeigte, dass nach einer gewissen Zeit das System zu 97 % seinen Initialzustand angenommen hatte. Der Initialzustand ist die Schwingungsmode des System am Anfang. Spätere Simulationen mit höherer Genauigkeit und Zeit unterstützten die Berechnung. Damit zeigte das Experiment den Wert von Computersimulationen in der Forschung und verhalf diesen zum Durchbruch.

Seit dem Ende der neunziger Jahre wird das Konzept der Simulation in vereinfachter Form auch auf AUVs angewendet [HJ89] [HTD⁺87] [LBM⁺95] [PSOW91]. Einer der ersten Simulatoren war der *NPS AUV Integrated Simulator* der Naval Postgraduate School in Monterey, Kalifornien [BKZ92]. Er verfügte bereits über wesentliche Eigenschaften, die später erweitert wurden [Bru94]. Der Simulator war in der Lage, ein spezifisches AUV samt der Hydrodynamik zu simulieren. Die Kommunikation mit Steuerungskomponenten fand über ein Netzwerk statt. Des Weiteren existierte eine dreidimensionale Visualisierung der Umgebung und des AUV, zu sehen in Abbildung 2.1. Bei der Sensorik wurde hauptsächlich das Sonar modelliert. Die Evaluierung wurde mit Hilfe des realen und modellierten NPS AUV durchgeführt. In Abbildung 2.2 sind die Struktur und der Einsatzzweck dargestellt. Es konnte mit dem Mikroprozessor des AUV gearbeitet werden oder mit einem anderen vom AUV unabhängigen im Labor. Der Mikroprozessor erfasste die Telemetriedaten des echten oder simulierten AUV. Diese konnten später über eine spezielle Arbeitsstation für Visualisierung angezeigt werden. Neuere Simulatoren werden in Abschnitt 2.5 beschrieben.

2.3. Klassifikation und Modellstruktur

Im Laufe der Zeit entstanden verschiedene AUV-Simulatoren mit spezifischen Eigenschaften, die sich Klassen zuordnen lassen. Des weiteren hat jedes Simulationsmodell eine Grundstruktur bestehend aus Untermodellen. Nicht jeder Simulator implementiert alle Untermodelle und bei einigen Simulatoren sind die Grenzen zwischen den Untermodellen verwischt. Die Klassifikation und Aufbau eines allgemeinen Simulators lehnt sich an [RBRC04] an. Es lassen sich vier Klassen ableiten.

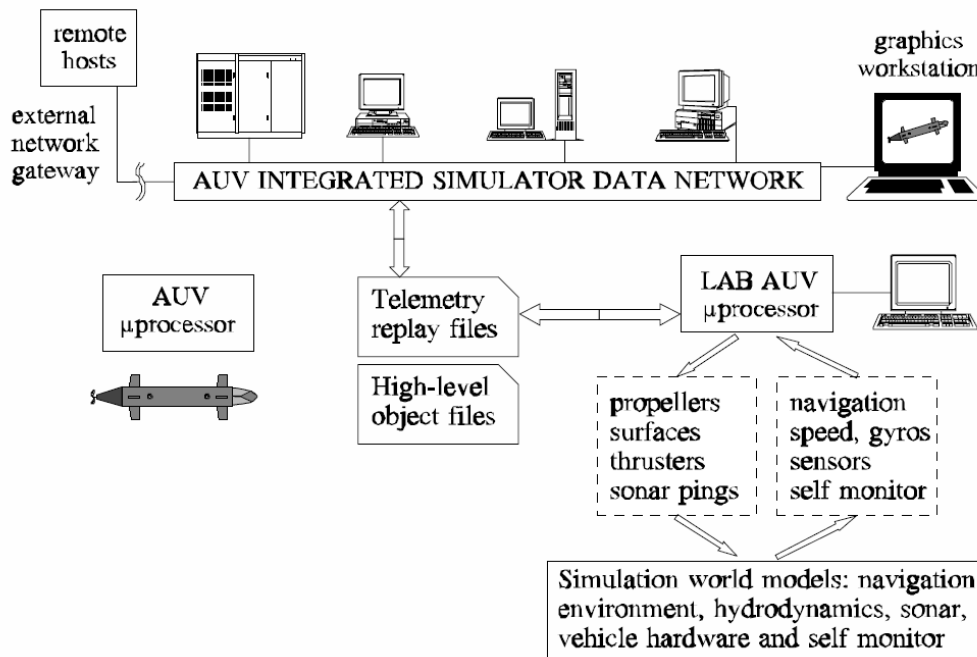


Abbildung 2.2.: Struktur und Einsatz des Simulators. Bild aus [BKZ92], mit freundlicher Genehmigung von Michael Zyda.

1. Offline (Off): Die Simulation rechnet langsamer als die Zeit in der Realität abläuft. Dies ist bei älteren Simulationen aufgrund der fehlenden Rechenleistung oder komplexer Modelle häufig der Fall.
2. Online (On): Die Simulation und die Realität sind zeitlich fast identisch.
3. Hardware-in-the-Loop Simulation (HiL): Bei einer HiL wird die reale Steuerungssoftware des AUV auf der Hardware des AUVs oder auf einem externen Gerät ausgeführt und mit der Simulation verbunden.
4. Hybrid Simulation (HS): Eine HS ist eine HiL mit der Besonderheit, dass das AUV sich in einer realen Umgebung, z.B. einem Wassertank, fortbewegt und einzelne oder alle Sensoren oder Aktoren durch virtuelle Modelle ersetzt werden. Dadurch kann die Realität das hydrodynamische Modell ersetzen. Dem realen AUV wird dabei vorgespielt, dass es echte Sensoren hat und Sensordaten bekommt.

Abbildung 2.3 veranschaulicht die Modellstruktur eines Simulators und welche Datenpfade bestimmte Modelle und Klassen verfolgen. Im Fall ohne Simulator existiert ein reales AUV, das mit der Realität in Wechselwirkung steht. Das AUV besteht aus einer Steuerungssoftware, Sensoren, die Daten aus der Realität erfassen und an die Steuerung weiterleiten und Aktoren, die Befehle der Steuerung entgegennehmen und in der Realität umsetzen. Dies führt dann zu einer veränderten Position des AUV in der Realität und der Kreislauf

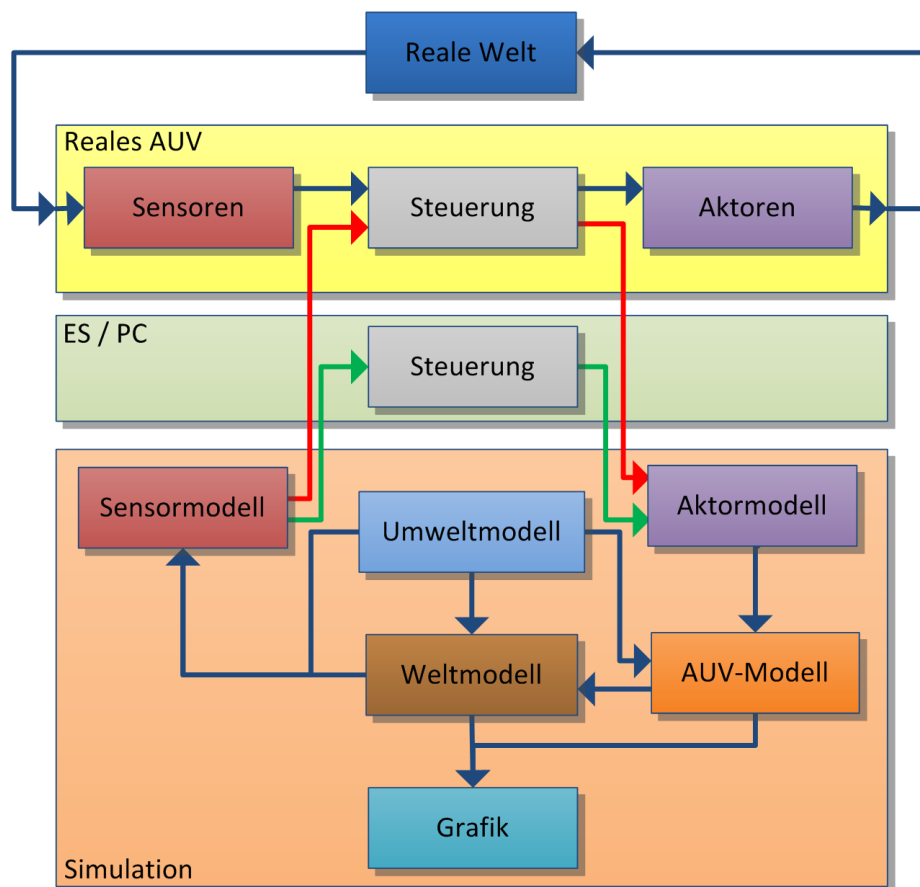


Abbildung 2.3.: Modellstruktur von AUV-Simulatoren. Das reale AUV (gelb) interagiert mit der realen Welt (blau). Die Steuerung (grau) kann ebenfalls auf einem externen PC oder eingebetteten System laufen. Das Simulationsmodell unterteilt sich in verschiedene Untermodelle.

beginnt von neuem.

Der Simulator ersetzt die Realität und besteht aus verschiedenen Modellen, die miteinander zusammenhängen. Eine genauere Erläuterung der Modelle ist in Abschnitt 2.7 aufgeführt. Sensor- und Aktormodelle ersetzen ihr jeweiliges Pendant in der Realität. Das Aktormodell wirkt auf das AUV-Modell ein, hauptsächlich das hydrodynamische Modell. Dieses interagiert mit dem Weltmodell (z.B. Terrain und Kollisionen). Das Sensormodell ist wiederum abhängig vom Welt- und vom Umweltmodell. Ein Sonar sieht Teile des Terrain und ein Temperatursensor erkennt das Temperaturprofil des Wassers. Zusätzlich beeinflusst das Umweltmodell, z.B. durch Wellen, die Welt und das AUV. Die Grafik visualisiert das Geschehen, zumeist das Weltmodell und die AUVs. Zur Umsetzung der einzelnen Modelle kommen zum Teil Physik-Engines zum Einsatz.

Bei einem HiL-Simulator wird die Steuerung an die Sensor- und Aktormodelle des Simulators angeschlossen. Die Steuerung kann dabei auf der AUV Hardware ausgeführt werden

(roter Pfad) oder extern (grüner Pfad). Im externen Modus läuft die Steuerung auf einem weiteren eingebetteten System, das dem im AUV ähnelt oder auf einem PC. Ein HS-Simulator erlaubt es, Teile der realen Sensorik und Aktorik durch virtuelle zu ersetzen. Das AUV selbst bewegt sich innerhalb einer realen Umgebung, womit das zumeist ungenaue hydrodynamische Modell durch die Realität ersetzt werden kann.

2.4. Vorarbeiten

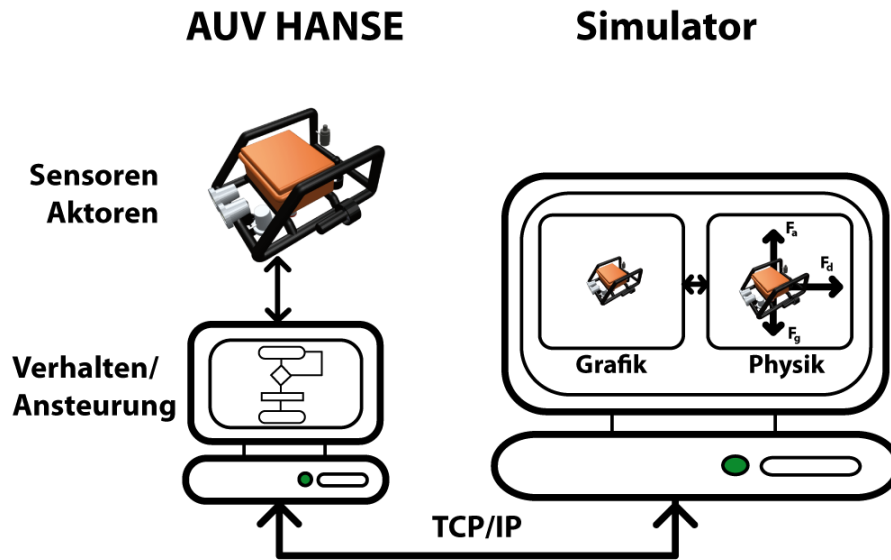
Im Rahmen einer Diplomarbeit mit dem Titel "Physikalisch realitätsnahe Simulationsumgebung für das AUV Hanse, mit Integration in die bestehende Steuerungs-Software", entstand eine Vorarbeit zu dieser Dissertation [Tos11]. Der dort entwickelte Simulator, mit dem Namen SimAUV (Simulator for AUVs), bildet die Grundlage für die späteren Arbeiten mit dem Simulator MARS. Motivation für das Projekt war das Fehlen einer brauchbaren Simulation für das HANSE-Projekt.

SimAUV ist ein online Hardware-in-the-Loop Simulator für mehrere AUVs, der in Java geschrieben ist. Für die Grafik wird die JMonkeyEngine verwendet. Für die Physik wird JBullet verwendet, eine native Portierung der Bullet Physics Library². Das Umweltmodell besteht aus einem Height-Map basierenden Terrain und statischen Objekten, wie z.B. Pipelines. Das AUV-Modell besteht aus einer Auftriebsberechnung basierend auf einem Ray-Casting Ansatz, dem Auftriebspunkt und dem Wasserwiderstand. Die Sensormodelle beinhalten Sonare, Drucksensoren, Kameras und Inertialsensoren. Als Aktoren wird der SeaBotix Thruster des HANSE-Projekts verwendet. Für diesen wurden Kraft- und Strommessungen durchgeführt und die resultierenden Werte über eine Regressionsfunktion in die Simulation eingebunden.

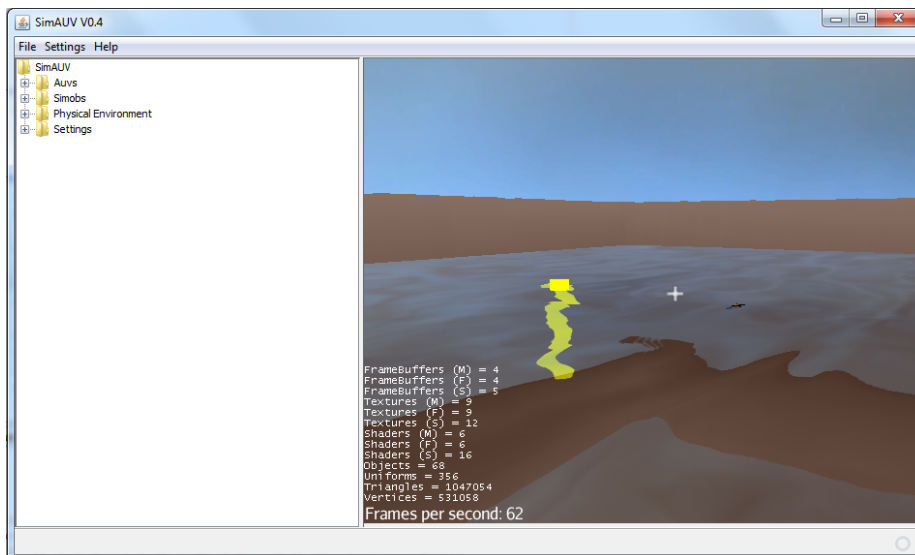
SimAUV besitzt eine rudimentäre, auf SWING aufbauende, Benutzungsschnittstelle, illustriert in Abbildung 2.4 b), die aus einer dreidimensionalen Ansicht und einer Baumstruktur besteht. Für die AUVs und Einstellungen gibt es XML-Konfigurationen. Für die Evaluation wurden Vergleichsexperimente durchgeführt. Diese bestanden aus einfachen Messungen, wie z.B. einer geradlinigen Fahrt, und komplexeren, wie einer Tiefenregelung. Zusätzlich wurden Tests mit Objekt- und Wandfolgeverhalten durchgeführt. Die auf der Framerate basierende Performanz-Evaluation zeigt einen performanten Einsatz des Simulators und dass genügend Ressourcen für Erweiterungen vorhanden sind. Ein wichtiger Bestandteil ist die Integration des Simulators in die bestehende Steuerungs-Software vom HANSE AUV. Dazu wurde eine auf TCP/IP basierende Schnittstelle konzipiert mit einem eigenen Request-Response-Protokoll, über das der Simulator mit der AUV-Software kommunizieren kann (siehe Abbildung 2.4 a)).

SimAUV bildet die Grundlage für das MARS-Projekt. So wurden hier bereits erste Erfahrungen mit JAVA, der JMonkeyEngine und Bullet, in Form von JBullet, gesammelt. Diese Komponenten bilden bei MARS die Basis. Allerdings ist die eigen-entwickelte TCP Schnittstelle für einen allgemeinen Einsatz zu unflexibel und an das HANSE AUV gebun-

²JBullet: <http://jbullet.advel.cz/>, Zugriff am 19.10.2015



(a) Grobe Struktur und Ablauf von SimAUV.



(b) Beispiel der dreidimensionalen Repräsentation und GUI von SimAUV.

Abbildung 2.4.: Struktur und Aussehen von SimAUV [Tos11].

den. Die Kamera beherrscht aufgrund der vereinfachten Unterwassersicht kaum Effekte und gibt die Unterwasserwelt schlecht wieder. Die GUI erlaubt eine minimale Interaktion und ist starr. Deshalb, und aufgrund von Struktur-, Performanz- und Modularitätsschwächen, wurde zusätzlich zu den Erweiterungen des MARS-Projektes selber ein Großteil des Programmcodes für das MARS-Projekt neu- oder umgeschrieben.

2.5. Stand der Forschung

Dieser Abschnitt stellt die wesentlichen AUV-Simulatoren der letzten 15 Jahre vor. Es wird die allgemeine Struktur erläutert und technische Details genannt, wie z.B. die genutzte Programmiersprache, Lizenzen und Kommunikationsschnittstellen. Die hydrodynamischen Modelle, Umweltmodelle und Unterstützung für Sensorik und Aktorik werden betrachtet. Ebenso wird dargestellt, ob der Simulator mit Hilfe eines echten AUV validiert wurde. Dabei wird in Unterabschnitt 2.5.1 auf ältere Simulatoren eingegangen und in Unterabschnitt 2.5.2 auf neuere.

2.5.1. Ältere Simulatoren

Der *Multi-Vehicle Simulator* (MVS) ist ein echtzeitfähiger Simulator für mehrere AUVs [KAU96]. MVS ist als verteilte Architektur aufgebaut, in der die einzelnen Komponenten über TCP/IP miteinander kommunizieren. Die Umgebung wird dreidimensional dargestellt, und es existiert eine grafische Benutzungsschnittstelle. Tests wurden mit dem AUV *The Twin-Burger I* durchgeführt.

Core Simulation Engine (CSE) ist ein verteilter Hardware-in-the-Loop Simulator auf Netzwerkbasis [LFR⁺98]. Es können Thruster, Gyroskope, Sonare und Kameras simuliert werden. Das hydrodynamische Modell basiert auf dem ANGUS ROV Modell mit sechs Freiheitsgraden.

Der Simulator von Gracanin [GVM98] ist ein verteilter netzwerkbasierter Simulator für mehrere AUVs. Die Kommunikation erfolgt über HTTP und JavaScript. Die dreidimensionale Darstellung enthält Terrain und Objekte. Zur Unterstützung können ein Head-Mounted Display und ein Datenhandschuh verwendet werden. Das hydrodynamische Modell ist ein sechs Freiheitsgrade Modell mit dynamischen Auftrieb, Dämpfung und Thrusterkräften. Als Testvehicle stand das *Phantom S2* AUV zur Verfügung.

Der Simulator von Bruzzone ist ein netzwerkbasierter verteilter Simulator [BBCV01]. Er besteht aus drei Komponenten, die über das User Datagram Protocol (UDP) miteinander kommunizieren. Die Visualisierung nutzt OpenGL. Als Sensoren werden Sonare, Ultraschall-Doppler-Profil-Strömungsmesser (eingesetzt als Doppler Velocity Log, DVL), Drucksensoren und Kompass unterstützt. Die Sensordaten können zusätzlich verrauscht werden. Die Hydrodynamik ist an Fossen [Fos02] angelehnt und unterstützt Widerstand und Auftrieb über sechs Freiheitsgrade. Die Thruster bestehen aus einem affinen Modell, das abhängig von der Drehzahl des Propellers ist. Als Testvehicle stand das *Romeo* ROV zur Verfügung.

Der Simulator von Song [SAF03] ist ein Hardware-in-the-Loop Simulator für mehrere AUVs. Die interne Kommunikation findet über shared memory statt, während eine Anbindung von außen über TCP/IP realisiert ist. Die Sensorik unterstützt akustische Long Baseline Unterwassernavigation (LBL), Drucksensoren, DVL und das Globale Positionsbestimmungssystem (GPS). Zusätzlich können die Sensordaten verrauscht werden. Das hydrodynamische Modell besteht aus sechs Freiheitsgraden, mit Wellen, Auftrieb und Wi-

derstand. Das Aktorenmodell enthält Tiefenruder und Thruster, die über ein Steady-State Modell modelliert sind. Zur Evaluation wurde das Ocean Explorer AUV verwendet.

Im Rahmen des CO³AUVs-Projekts (Cooperative Cognitive Control for Autonomous Underwater Vehicles) [BAC⁺11] entstand der verteilte netzwerkbasierte Simulator *JRSimulator* [RB11]. Als Physikengine kommt Bullet zum Einsatz, während OGRE für die dreidimensionale Visualisierung verwendet wird. Diese erlaubt Terrains und Unterwassernebel. Eine einfache Qt Widget-basierte grafische Benutzungsschnittstelle erlaubt über mehrere Fenster Daten zu visualisieren und AUVs zu manipulieren. Sonare werden über ein Ray-Tracing-Verfahren simuliert und Kameras werden ebenfalls unterstützt. Das hydrodynamische Modell verwendet Auftrieb und Widerstand.

Im Rahmen des DeepC Projekts, in dem ein neues tiefseetaugliches AUV entwickelt wurde, entstand ein neuer Simulator [Hor01]. Der Simulator besteht auf zwei Komponenten, die über CORBA miteinander kommunizieren. Die erste Komponente *AUV-Simulation* stellt das AUV dar. Die zweite Komponente *Virtual Reality* stellt die Welt dreidimensional dar. Die Umgebung wird über ein 3D-Terrain dargestellt, und es existieren Strömungen und Wasserschichten. Zusätzlich werden Fauna und Flora eingeblendet.

2.5.2. Aktueller Stand

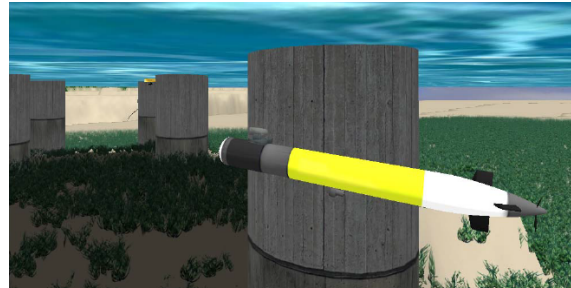
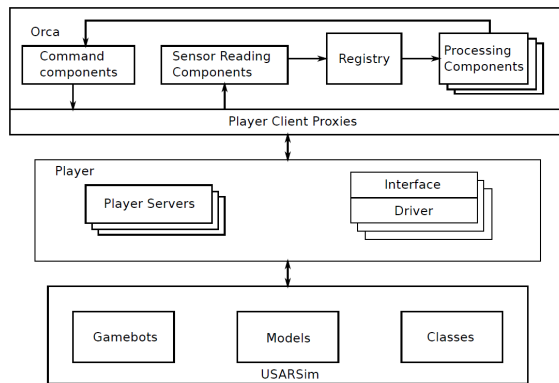
Dieser Abschnitt erläutert die aktuellen Simulatoren im Bereich der AUV-Simulation im Detail. Die Auflistung erfolgt in alphabetischer Reihenfolge.

CADCON

Cooperative AUV Development Concept (CADCON) ist ein verteilter Simulator für mehrere AUVs [CKPL99]. CADCON ist über das Internet erreichbar und wird auf einem externen System gehostet, auf das die Benutzer zugreifen können. CADCON besteht aus 4 einzelnen Servern, die per TCP/IP miteinander kommunizieren. Der *Environment Server* bildet dabei den Kern. Er kümmert sich um die eigentliche Simulation. Der *Visualizer Client* ist eine dreidimensionale Repräsentation auf Basis von OpenGL während der *Status Client* Informationen über den aktuellen Status der Simulation liefert. Der *AUV Simulator Client* ist der Vermittler zwischen dem *Environment Server* und den Nutzern im Internet.

Es kann ein Terrain und eine Eisbedeckung definiert werden. Im Modell werden außerdem in vereinfachter Form Thermokline, Salzgehalt und Strömungen betrachtet. Die Teilnehmer werden als simple geometrische Objekte modelliert, die untereinander und mit der Umgebung kollidieren können. Die Bewegungen werden durch ein einfaches Newtonsches Modell berechnet. Die Aktualisierungsrate ist auf 10 Hz gesetzt. Die Konfiguration erfolgt über ein eigenes Dateisystem. Bei der Kommunikation werden Nachrichten der Teilnehmer direkt zugestellt.

CADCON wurde in verschiedenen weiteren Projekten verwendet, so wie z.B. bei AOSN (autonomous oceanographic sampling networks) [ABT03] und um SAUVs (Solar-powered AUVs) zu simulieren [KC06].



(a) Dreischichtige Architektur von MarineSIM.

(b) Beispiel aus der dreidimensionalen Repräsentation.

Abbildung 2.5.: Struktur und Aussehen von MarineSIM. Bilder aus [NSWL⁺10], mit freundlicher Genehmigung von Bharath Kalyan.

DVECS

Distributed Virtual Environment Collaborative Simulator (DVECS) ist eine in C++ geschriebene hybride Simulation für mehrere AUVs [CMY00] [CY01]. Die einzelnen Komponenten kommunizieren über TCP/UDP miteinander. Zusätzlich zur OpenGL-basierten 3D-Visualisierung besteht die Möglichkeit, über Projektoren und Polarisationsbrillen einen virtuellen Realitätseindruck zu bekommen. Daneben gibt es eine GUI bestehend aus mehreren Fenstern und Control Panels. Die Unterwasserdynamik der AUVs besteht aus einem einfachen kinematischen Modell. Für Strömungen wird ein isotropischer Ansatz gewählt. Simulierte Tests wurden mit dem AUV ODIN [CYT95] durchgeführt.

MarineSIM

MarineSIM ist eine Simulationsumgebung für mehrere AUVs [NSWL⁺10]. Die Struktur von MarineSIM ist in Abbildung 2.5 a) dargestellt und besteht aus 3 Teilen. Die eigentliche Simulation besteht aus USARSim [CLW⁺07], einem Robotik Simulator Framework auf Basis der Unreal Engine. Dieses wird in einer eigenen Sprache UnrealScript programmiert. Innerhalb der Unreal Engine übernimmt die KarmaEngine die Physik, wie z.B. Kollisionen. Als Kommunikationsschnittstelle wird Player eingesetzt, ein TCP/IP basiertes Framework für Roboterkommunikation [GVH03]. Auf der obersten Ebene befinden sich das Robotik Framework ORCA [BKM⁺05], das sich über Player mit der Simulation verbindet und sich um die Verhalten kümmert.

Zusätzlich zu den Sensoren von USARSim wurden ein Sonar und ein Doppler Velocity Log implementiert. Getestet wurde das System mit dem Wasseroberflächenfahrzeug SCOUT und dem AUV REMUS. Eine visuelle Repräsentation ist in Abbildung 2.5 b) zu sehen und zeigt Schatten, Vegetation und eine einfache Wasseroberfläche als Unterwassereffekte.

Neptune

Neptune ist ein echtzeitfähiger HiL Simulator für mehrere AUVs [RCREF04, RBRC04]. Die Struktur und Beispielanwendung von Neptune ist in Abbildung 2.6 illustriert. Die Komponenten laufen in einem verteilten System. Die Kommunikation zwischen den Komponenten erfolgt über TCP/IP. Jeder Roboter hat sein eigenständig laufendes Bewegungsmodell.

Es besteht die Möglichkeit Neptune in einem hybriden Modus zu verwenden. In diesem können virtuelle und reale Sensoren kombiniert eingesetzt werden. Als Bewegungsmodell kommt ein einfaches kinematisches Modell ohne Hydrodynamik zum Einsatz. Die Visualisierung wird direkt über OpenGL realisiert. Das Terrain und Objekte werden über die Virtual Reality Modeling Language (VRML) geladen. Zur Kollisionserkennung kommt ein selbst entwickeltes Modell zum Einsatz, in dem Objekte als Kugeln modelliert werden. Das Terrain ist hierbei allerdings auf konvexe Formen beschränkt.

Als Sensoren stehen Kameras und interne Sensorik, wie Position, Geschwindigkeit und Tiefe zur Verfügung. Das Sonar erlaubt Distanzmessungen über eine einfache geometrische Schnittberechnung des Sonarkegels mit dem Kugelmodell der Objekte. Das Thrustermodell nutzt ein einfaches affines Modell durch eine lineare Funktion, die abhängig von der Drehgeschwindigkeit der Propeller ist. Zur Konfiguration der Roboter wird eine eigene Beschreibungssprache verwendet. Evaluiert wurde Neptune mit dem realen Roboter URIS [BRG⁺05].

SubSim

SubSim ist eine in C++ geschriebene Simulationsumgebung für einzelnen AUV, die später erweitert wurde [BB06, BBG⁺04]. Die grundlegende Struktur ist in Abbildung 2.7 illustriert. Sie ist plugin-basiert, somit sind Teile leicht austausch- oder erweiterbar. Die Kommunikation mit der AUV-Steuerungssoftware erfolgt über eine C++ API.

Für die Physik wird die Physical Abstraction Layer (PAL) verwendet in Ausprägung durch die *Newton Game Dynamics* Physik-Engine. Als Kräfte werden der statische und dynamische Auftrieb und Wasserwiderstand berechnet. Die Auftriebsberechnung erfolgt, indem das AUV als eine Menge von Kugeln modelliert wird, anhand derer dann das Volumen berechnet wird. SubSim simuliert mit einer Auflösung von 10ms und kann in 10 Sekunden, 6 Sekunden simulierte Zeit berechnen.

Als Sensoren werden IMU, Kamera, Kontakt und Distanzmesser wie Echolote unterstützt, wobei diese über einen einfache Ray-Casting Distanzmessung realisiert sind. Sämtliche Sensoren lassen sich mit einem einfachen Rauschmodell ausstatten, z.B. einer Gaussverteilung. Für das Aktorenmodell kommen einfach mathematische Thrustermodelle zum Einsatz oder die Kraft kann über interpolierte Look-Up Tabellen ausgelesen werden, deren einzelne Datenpunkte vorher gemessen wurden. Des weiteren werden Tiefenruder unterstützt.

Konfigurationsdaten der AUVs und der Umgebung werden über XML gespeichert. Eine einfache modulare GUI verwendet das wxWidgets Framework. Tests wurden mit dem Mako AUV durchgeführt.

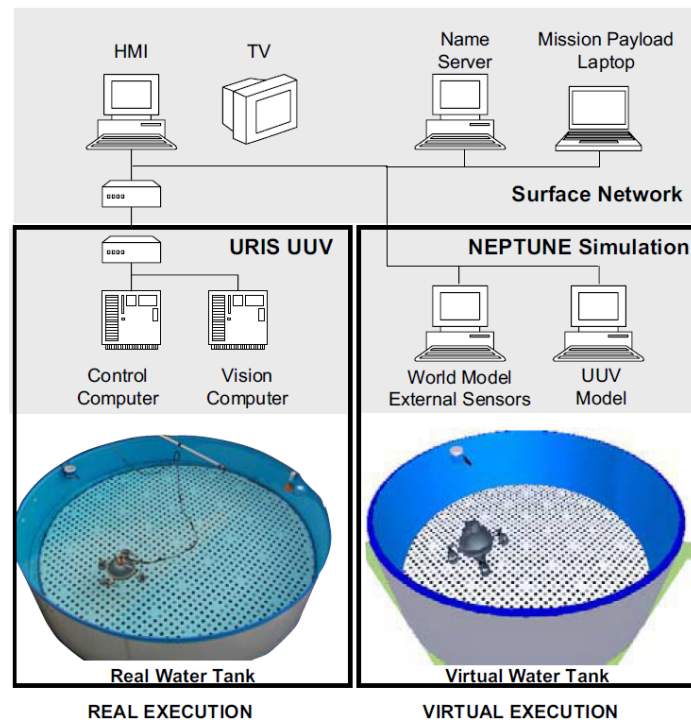


Abbildung 2.6.: Allgemeine Übersicht über die Struktur von Neptune mit Einsatzszenario im Hybridmodus. Das echte AUV ist im linken realen Wassertank (real execution) im Einsatz und verwendet die reale Hydrodynamik. Die Simulation rechts (virtual execution) verwendet virtuelle Sensoren, um dem echten AUV virtuelle Daten zu senden. Bild aus [RBRC04], mit freundlicher Genehmigung von Pere Ridao Rodríguez.

Thetis

Thetis ist ein echtzeitfähiger verteilter Hardware-in-the-Loop Simulator, der mehrere AUVs unterstützt [PLJ08]. Es wird eine verteilte Architektur verwendet, die aus vier einzelnen Simulatoren besteht, welche untereinander über UDP/IP kommunizieren (siehe Abbildung 2.8). Innerhalb der Komponenten wird ein Shared Memory Model verwendet. Der *Vehicles Simulator* kümmert sich um die Hydrodynamik der AUVs. Der *Sensors Simulator* verwaltet alle Sensoren. Der *Communications Simulator* ist zuständig für die Unterwasserkommunikation. Alle drei Simulatoren sind eng mit dem *Environment Simulator* verbunden der sich um Kollisionen und Signalausbreitung kümmert.

Das Hydrodynamische Modell ist an Fossen [Fos02] angelehnt und ist ein 6-DOF Modell mit Unterstützung für Auftrieb und Dämpfung. Zusätzlich werden Dämpfungseffekte durch die Kármánsche Wirbelstraße beachtet. Für die Thrustermodellierung wird ein bilineares Modell verwendet, bestehend aus nicht-linearen Funktionen in denen der Vortrieb abhängig ist von der Drehgeschwindigkeit des Propellers, entlehnt aus [RBRC04]. Zusätzlich gibt es ein Tiefenrudermodell, basierend auf dem vereinfachten dynamischen Auftriebsmodell.

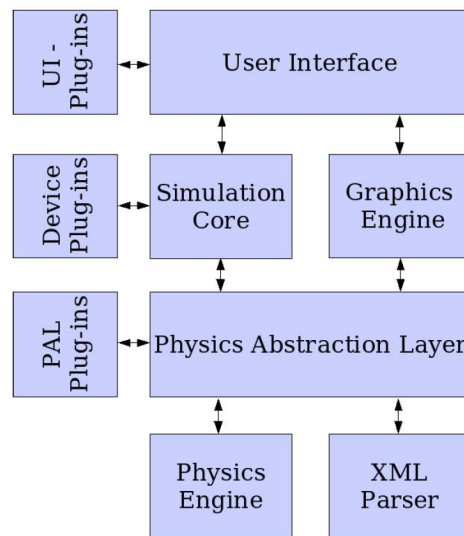


Abbildung 2.7.: Allgemeine Übersicht über die Struktur von SubSim [BBG⁺04].

Bild aus Bräunl, Thomas ; Boeing, Adrian ; Gonzales, Louis ; Koestler, Andreas ; Pettitt, Joshua: The Autonomous Underwater Vehicle Initiative-Project Mako. In: *Robotics, Automation and Mechatronics, 2004 IEEE Conference on* Bd. 1, IEEE, 2004, S. 446–451. <http://dx.doi.org/10.1109/RAMECH.2004.1438961>. DOI 10.1109/RAMECH.2004.1438961, mit freundlicher Genehmigung von Thomas Bräunl.

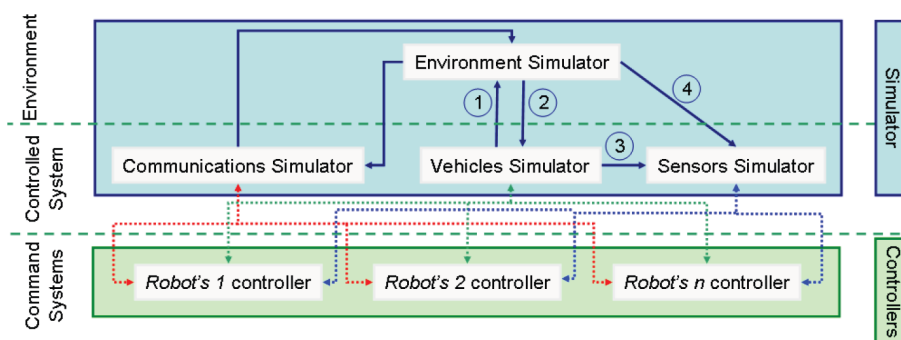


Abbildung 2.8.: Allgemeine Übersicht über die Struktur von Thetis. Bild aus [PLJ08], mit freundlicher Genehmigung von Lionel Lapierre.

Als Sensoren werden Temperatur, Leitfähigkeit, GPS, DVL und ein Attitude Heading Reference System (ARHS) unterstützt. Sämtliche Sensordaten können mit Rauschen versehen werden. Die Umgebung besteht aus einem Terrain, Temperatur und Salzverteilung und statischen Strömungen. Die Konfiguration der AUVs und der Umgebung basiert auf XML. Die Unterwasserkommunikation unterstützt Signalausbreitung, Umgebungsrauschen und Übertragungsverlust [PCJ08]. Test fanden mit dem Taipan 300 AUV statt [PCJX09].

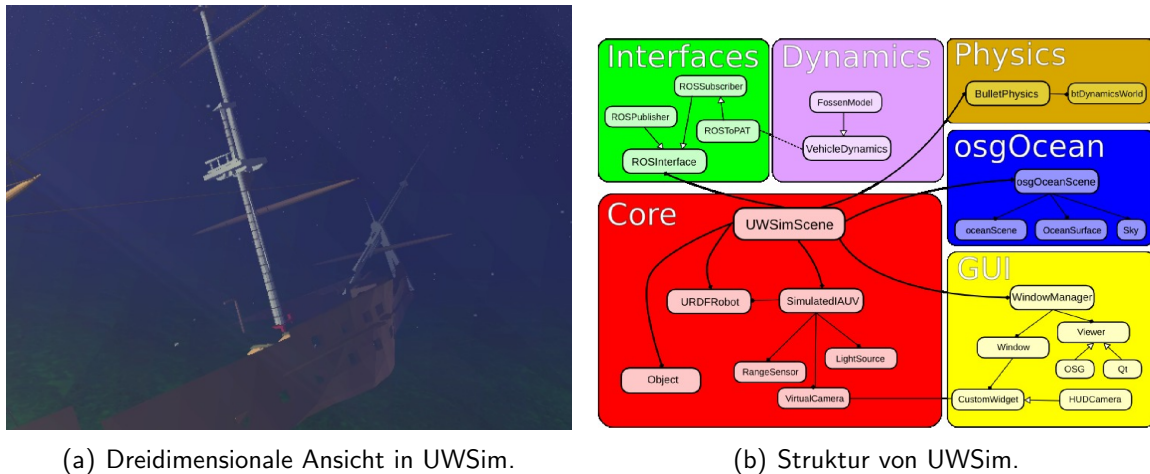


Abbildung 2.9.: UWSim Struktur und dreidimensionale Ansicht. Bilder aus [PPFS12], mit freundlicher Genehmigung von Javier Pérez Soler.

UWSim

Der UnderWater SIMulator (UWSim) ist eine open-source Simulationsumgebung für mehrere AUVs, die in C++ geschrieben wurde [PPFS12]. Die Struktur von UWSim ist in Abbildung 2.9 b) aufgezeigt.

Das *Core Modul* ist zuständig für das Laden der Szene und AUVs. Die Kommunikation mit der Simulation erfolgt über das ROS Framework (Robot Operating System) innerhalb des *Interfaces Modul*. Die Visualisierung wird durch OSG (OpenSceneGraph)³ und OSGOcean⁴ durchgeführt. Dies erlaubt realistische Unterwassereffekte wie Sichtbarkeit, Wasserfarbe, Partikel und Wellen. Ein Beispielansicht befindet sich in Abbildung 2.9 a). Das Terrain wird über vorgefertigte 3D-Modelle geladen.

Die Hydrodynamik wird in einem eigenen *Modul Dynamics* in Matlab berechnet. Es werden Wind und Wellenkräfte sowie Auftrieb, Widerstand und Antrieb der Motoren berechnet. Für Kollisionen ist im *Modul Physics* die Bullet Physics Library zuständig.

Die Sensoren bestehen aus einem einfachen 6-DOF Lokalisierungssensor, Kameras und einem einfachen Abstandssensor. Die Sensordaten lassen sich zusätzlich über eine Gaussverteilung verrauschen. Es besteht die Möglichkeit kinematische Ketten zu definieren, was den Einsatz von Unterwasser-Manipulatoren ermöglicht.

Beschreibungen der Umwelt und AUVs werden über XML im Unified Robot Description Format (URDF)⁵ vorgenommen. Des Weiteren besteht die Möglichkeit, Playbacks abzuspielen und den Simulator nur zur Visualisierung zu verwenden. Die rudimentäre GUI bietet ein einfaches Widget System für Erweiterungen. Simulierte Tests wurden mit dem *NPS AUV II (Phoenix)* ausgeführt.

³<http://www.openscenegraph.org/>, Zugriff am 20.10.2015

⁴<https://code.google.com/p/osgocean/>, Zugriff am 20.10.2015

⁵<http://www.ros.org/wiki/urdf>, Zugriff am 20.10.2015

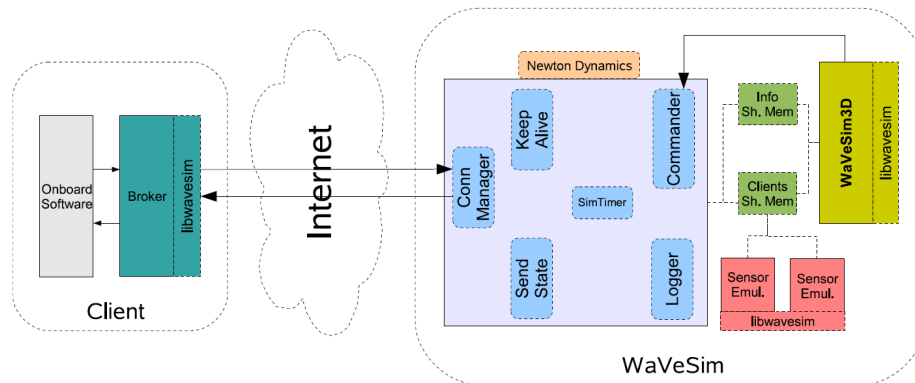


Abbildung 2.10.: Allgemeine Übersicht über die Struktur von WaVeSim. Bild aus [SDM06], mit freundlicher Genehmigung von Aníbal Matos.

WaVeSim

Der Water Vehicle Simulator (WaVeSim) ist ein in C++ geschriebener Simulator für mehrfache Unterwasser- und Wasseroberflächenfahrzeuge [SDM06]. Die Struktur von WaVeSim ist in Abbildung 2.10 zu sehen. Der Simulator teilt sich in zwei Bereiche: Einmal der Visualisierung und dann der Simulation. Die interne Kommunikation findet über ein Shared Memory Modell statt. Clients können sich per UDP am Server von WaVeSim anmelden und über ein eigens spezifiziertes Protokoll miteinander kommunizieren. Die Visualisierung basiert auf OGRE und ist stark vereinfacht und unterstützt keine Unterwassereffekte. Ähnliches gilt für die GUI. Die Konfiguration der Welt und der AUVs erfolgt über XML Dateien.

Für die physikalischen Berechnungen wird die *Newton Game Dynamics* Physikengine verwendet. Das hydrodynamische Modell besteht aus Motoren, dem Auftrieb und dem Widerstand. Zur Kollision werden einfache Kollisionsformen, wie z.B. kastenförmige, verwendet. Auf der Sensorseite werden Druck- und Temperatursensoren, Salzgehalt und Leitfähigkeit unterstützt. Für den Temperatursensoren und das Temperaturprofil des Wassers werden zweidimensionale Temperaturkarten mit einfachen linearen Funktionen kombiniert. Ein Echolot erlaubt es, die Entfernung zum Boden über eine einfache Distanzberechnung festzustellen. Eine Besonderheit ist die Simulation von akustischer Navigation über *beacons*, bei denen auch die Signalstärke- und ausbreitung berücksichtigt wird. Test oder Vergleiche mit einem echten AUV wurden nicht durchgeführt.

Wireless Simulation Server Plugin for USARSim

Der Wireless Simulation Server (WSS) [SC10, SCB10] ist ein Plugin für den allgemeinen Robotiksimulator USARSim [CLW⁺07] und erweitert diesen um Eigenschaften für AUVs. Die sind zum einem die Möglichkeit, mehr als ein AUV gleichzeitig zu simulieren und die Unterstützung von akustischer Unterwasserkommunikation. Die Unterwasserfähigkeiten von USARSim wurde bereits im Kapitel 2.5.2 (MarineSIM) besprochen. Außerdem

wurde das Antriebsmodell des vorhandenen U-Bootes um hintere Tragflächen, Ruder und Propeller erweitert.

Das Kommunikationsmodell betrachtet Signalverzögerung, Übertragungsverluste, Absorption und Umgebungsrauschen. Bei der Signalverzögerung kommt eine global gemittelte Thermokline-Funktion für die Temperatur zum Einsatz. Das Terrain wird verwendet, um Mehrwegempfang zu beachten, der durch Reflexionen entsteht. Dies wird mit Hilfe eines Ray-Tracing Verfahrens realisiert.

2.6. Aktueller Stand der Technik

Im aktuellen Stand der Technik dominieren ROV Simulatoren für Personaltraining. Diese haben gemein, dass sie geschlossene Systeme sind, die schwer mit bestehender AUV-Software zu verbinden sind. Es existieren mehrere frei verfügbare Simulatoren im Bereich der mobilen Robotik [CMBG07] und auch kommerzielle, wie z.B. V-Rep [RSF13], allerdings fehlen hier wesentliche Eigenschaften wie z.B. ein hydrodynamisches Modell.

Vortex Simulator ist ein ROV Trainingsimulator der Firma CM Labs Simulations [vor]. Er ist in C++ geschrieben und ist Teil eines größeren Simulationsframeworks der Firma. Die Simulation läuft in Echtzeit ab und verwendet eine eigene Physikengine mit Unterstützung für Kollisionen und Gelenke. Das hydrodynamische Modell besteht aus statischen und dynamischen Auftrieb, Widerstand und virtueller Masse. Zusätzlich können Strömungen aktiviert werden. Zwei Besonderheiten für die ROV-Unterstützung sind Unterwassermanipulatoren in Form von Roboterarmen und Versorgungskabel. Die Grafik erlaubt es, die Sichtweite durch Beleuchtung, Schwebestoffe und Nebel einzustellen. Es können verschiedene ROVs konfiguriert werden, die über ein eigenes Kontrollsystem verfügen. Es sind auch Operationen an der Wasseroberfläche im Zusammenspiel mit Schiffskränen möglich. *ROVsim²Pro* der Firma *Marine Simulation* ist ein Simulator für das Training von ROV-Personal [rov]. Das ROV Modell ist konfigurierbar und erlaubt die Simulation von ROVs verschiedener Hersteller. Das hydrodynamische Modell unterstützt Strömungen und das Wogen, das durch Wellen verursacht wird. Eine Kabelmodell als Versorgung des ROVs existiert ebenfalls. Als grafische Effekte werden Kaustiken, Nebel, und Oberflächenreflexionen unterstützt. Ein Terrain kann über 3D-Modelle geladen und mit Details, wie z.B. Tonnen, versehen werden. Auf der sensorischen Seite existiert ein Sonar, ein laserbasierter Entfernungsmesser und maximal 2 Videokameras. Als Aktoren gibt es einen 2-DOF Manipulator in Form eines Roboterarms und ein Saugsammler. Bei Teilen des Equipments können Fehler simuliert werden.

DeepWorks ist ein ROV Trainingssimulator [Ltd15]. Die proprietäre Physikengine unterstützt Kollisionen und Kabel. Das hydrodynamische Modell unterstützt Strömungen. Ebenfalls können elektrische und hydraulische Fehler simuliert werden. Grafische Effekte wie Nebel und Licht stehen zur Verfügung. Außerdem gibt es eine Sonarsimulation.

Die meisten kommerziellen Simulatoren setzen ihren Fokus auf die Simulationen einzelner ROVs und Training von Personal. In diesem Fokus liegen folglich die verschiedenen Modelle. Es wird für gewöhnlich nur ein einzelnes ROV simuliert. Strömungen sind wichtig

für das Training des Piloten, ebenso Sichtbarkeit in Form von grafischen Effekten wie Partikeln, Nebel und Licht. Weil ROVs ein Versorgungskabel und Manipulatoren, wie z.B. Roboterarme besitzen, werden diese immer modelliert. Es ist schwierig, Zugang zu diesen Systemen zu erhalten, um die eigene AUV Software anzuschließen. Obwohl eine Vielzahl an kommerziellen Simulatoren im robotischen Bereich existiert, ist Entwicklung bis jetzt nicht bei den AUV-Simulatoren angekommen. Aufgrund der oben erwähnten Punkte eignen sich die meisten kommerziellen Systeme nicht für Schwarmsimulation von AUVs.

2.7. Zusammenfassung

Simulationen werden seit Jahrzehnten erfolgreich eingesetzt, besonders im Bereich der Flugzeuge. Seit der Mitte des 19. Jhd. und dem Auftreten von Computern werden auf diesen verschiedenste Problemstellungen simuliert. Das Konzept der Simulation wurde aufgrund der verschiedenen Vorteile in den Achtziger Jahren auf AUVs übertragen. Über die letzten Jahre wurde viele AUV Simulatoren entwickelt, die verschiedene Fokusse aufweisen. Wichtige Eigenschaften von AUV-Simulatoren werden in Unterabschnitt 2.7.1 tabellarisch aufgelistet. Zusätzlich werden in Unterabschnitt 2.7.2 die Fortschritte von den älteren hin zu den neueren Simulatoren erläutert. Im letzten Unterabschnitt 2.7.3 werden die Anforderungen an einen echtzeitfähigen Umweltmonitoring-Simulator für Schwarm-AUVs diskutiert und Unterschiede aufgezeigt. Es gibt noch zahlreiche weitere Simulationen im Kontext von AUVs, allerdings werden in diesen häufig nur einzelne Aspekte betrachtet, viele konzentrieren sich z.B. auf die Simulation von Thrustern.

2.7.1. Eigenschaften

Jeder Simulator hat gewisse Eigenschaften, die ihn von anderen unterscheiden. Die nachfolgende Aufzählung behandelt die wichtigsten davon. Die Tabelle 2.1 listet die besprochenen Simulatoren aus Abschnitt 2.5 auf und fasst sie hinsichtlich allgemeiner Eigenschaften aus der Aufzählung zusammen.

1. Name und Quelle: Wie lautet der Name der Simulation und auf welche Quelle wird sich bezogen?
2. Architektur: Wie ist der grundlegende Aufbau der Simulation? Handelt es sich um einen verteilten Ansatz oder um ein zentrales Programm?
3. Simulationstyp: Welche Simulationsmodi unterstützt die Simulation. HiL, HS, online und offline. Siehe dazu Abschnitt 2.3.
4. Schnittstellen: Welche Schnittstellen führen nach außen, um den Simulator mit der Software der AUVs zu verbinden?

5. Physikengine: Welche Physikengine wird verwendet? Eine Eigenentwicklung, proprietäre oder freie Engine?
6. Grafik: Gibt es eine Form der visuellen Repräsentation der Simulation, z.B. 3D oder 2D?
7. Multiroboter: Besteht die Möglichkeit in der Simulation mehr als ein AUV gleichzeitig zu simulieren? Dies ist wichtig, um z.B. Schwärme zu untersuchen.
8. Testroboter: Welche AUVs werden verwendet, um die Simulation zu validieren?
9. Benutzungsschnittstelle (BSS): Besitzt der Simulator eine Benutzungsschnittstelle, um leichter mit der Simulation interagieren zu können?
10. Open-Source: Ist der Quellcode des Simulators frei verfügbar?
11. Programmiersprache: Welche Programmiersprache wurde verwendet, um den Simulator zu entwickeln?
12. Verfügbarkeit: Ist die Simulation verfügbar oder so veraltet, dass er nicht mehr bezogen werden kann?

Das Modell bildet den Kern einer AUV Simulation und lässt sich in verschiedene Untermodelle einteilen. In der folgenden Aufzählung werden die wichtigsten beschrieben. Die Tabelle 2.2 listet die Simulatoren hinsichtlich ihrer Modelle auf.

1. Weltmodell: Wie ist die Welt modelliert, in dem sich das AUV bewegt?
 - a) Terrain: Gibt es die Möglichkeit ein Terrain zu laden?
 - b) Objekte: Ist es möglich statische Objekte zu platzieren wie z.B. Pipelines oder Bojen?
2. Umweltmodell: Wird die Umwelt, z.B. das Wasser oder die Ökologie simuliert?
 - a) Wellen: Sind wichtig für den Einsatz von Überwasserfahrzeugen, aber auch bei flacheren Gewässern und Küstenbereichen.
 - b) Strömung: Beeinflussen das AUV auch in tieferen Gewässern.
 - c) Wind: Ist wichtig für Überwasserfahrzeuge, die Segel verwenden oder bei der Entstehung von Wellen.
 - d) Temperatur: Einfluss auf Flora, Fauna und Unterwasserkommunikation; benötigt für Drucksensordaten.
 - e) Salzgehalt: Einfluss auf Unterwasserkommunikation.
 - f) Pflanzen: Behindern die Sicht am nahen Bodenbereich und können fürs Umweltmonitoring verwendet werden.

- g) Fische: Wirken als Störfaktor für die Bildverarbeitung und können fürs Umweltmonitoring verwendet werden.
3. Sensormodell: Welche Sensoren stehen zur Verfügung, um die Welt und Umwelt zu erfassen?
 - a) Kamera: Visuelle Beobachtung von Umweltparametern wie Fischen oder Pflanzen. Die dreidimensionale Welt muss entsprechende Effekte unterstützen, um eine realistische Unterwassersicht zu ermöglichen, wie z.B. Kaustiken oder Nebel.
 - b) Sonar: Bei schlechter Sicht müssen andere Sensoren als Kameras eingesetzt werden. Deswegen werden Systeme auf akustischer Basis verwendet, wie z.B. Sonare.
 - c) Druck: Ermöglicht genaue Tiefenbestimmung unter Wasser.
 - d) GPS: Wichtiger Sensor zur Positionsbestimmung über Wasser.
 4. Aktormodell: Welche Aktoren stehen zur Verfügung, um sich innerhalb der Welt zu bewegen und bestehen Möglichkeiten mit ihr zu interagieren?
 - a) Thruster: Primäre Fortbewegung von AUVs.
 - b) Ruder: Um die Richtung oder Tiefe zu steuern.
 - c) Roboterarm: Möglichkeit der Interaktion mit der Umgebung, z.B. um Proben zu sammeln oder Ventile zu drehen.
 - d) Ballasttank: Veränderung des statischen Auftriebs.
 5. AUV-Modell: Wie genau wird die Hydrodynamik des AUV modelliert?
 - a) Auftrieb: Wird der statische Auftrieb behandelt?
 - b) Widerstand: Gibt es einen Widerstand bei der Fortbewegung im Wasser?
 - c) dynamischer Auftrieb: Gibt es einen dynamischen Auftrieb bei der Fortbewegung des AUVs?
 6. Rauschmodell: Können Sensordaten und Aktoren verrauscht werden?
 7. Kommunikationsmodell: Gibt es ein Modell für die Unterwasserkommunikation? Dies ist wichtig um Schwärme von AUVs zu simulieren.
 8. Stromverbrauch: Wird der Stromverbrauch gemessen und die Kapazität verwendet? Dies ist wichtig, um längere Missionen von Schwärmen zu simulieren.

Simulator	Architektur	Simulationstyp	Schnittstelle	Physikengine	Grafik	Multiroboter	Testroboter	BSS	Programmiersprache	Open-Source	Verfügbarkeit
Bruzzone	D	On, HiL	UDP	k.A.	3D	-	Romeo ROV	-	k.A.	-	-
CADCON	D	HiL	TCP	E	3D	+	SAUV	-	k.A.	-	+
CSE	D	k.A.	k.A.	k.A.	-	-	-	-	k.A.	-	-
DeepC	Z	On	k.A.	-	3D	-	-	-	k.A.	-	-
DVECS	D	HiL	TCP, UDP	k.A.	3D	+	ODIN	+	C++	-	-
Gracarin	D	HiL	HTTP	k.A.	3D	+	Phantom S2	-	k.A.	-	-
JR-Simulator	D	On, HiL	TCP, ROS	Bullet	3D	+	k.A.	+	k.A.	+	-
Marine-SIM	Z	On, HiL	Player	Karma	3D	+	Scout, Remus	-	U	k.A.	k.A.
MVS	D	On, HiL	TCP	k.A.	3D	+	Twin-Burger	+	k.A.	-	-
Neptune	D	HiL, HS	TCP	E	3D	+	Uris	-	k.A.	-	-
Song	Z	On, HiL	TCP	k.A.	-	+	Ocean Explorer	-	k.A.	-	-
SubSim	Z	On	API	Newton	3D	-	Mako	+	C++	-	+
Thetis	D	HiL	UDP	k.A.	-	+	Taipan 300	-	k.A.	-	-
UWSim	Z	On, HiL	ROS	Bullet	3D	+	NPS AUV II	+	C++	+	+
WaVeSim	Z	On	UDP	Newton	3D	-	-	+	C++	+	+
WSS	Z	On, HiL	k.A.	Karma	3D	+	-	-	U	-	-
SimAUV	Z	On, HiL	TCP	JBullet	3D	+	HANSE	+	Java	-	+
MARS	Z	On, HiL	TCP, ROS	Bullet	3D	+	HANSE, MON-SUN	+	Java	+	+

Tabelle 2.1.: Gegenüberstellung alle AUV-Simulatoren aus Abschnitt 2.5. Verwendete Abkürzungen: + für Ja, - für nein, D für Distributed, HiL für Hardware-in-the-Loop, E für Eigenentwicklung, HS für Hybrid Simulation, Z für Zentral, On für Online, Off für Offline, U für UScript. Ältere Simulatoren in grau unterlegt.

Simulator	Welt		Umwelt				Sensoren			Aktoren			AUV			Versch.								
	Terrain	Objekte	Wellen	Strömung	Wind	Temperatur	Salzgehalt	Pflanzen	Fische	Sonar	Kamera	Druck	GPS	Thruster	Ruder	Roboterarm	Ballasttank	Auftrieb	Widerstand	dynamisch	Rauschen	Kommunikation	Stromverbrauch	
Bruzzone	✓									✓				✓							✓			
CADCON	✓	✓		✓																		✓		
CSE									✓															
DeepC				✓																				
DVECS				✓																				
Gracahin		✓												✓										
JRSimulator	✓	✓							✓					✓										
MarineSIM	✓	✓							✓	✓				✓	✓									
MVS	✓													✓										
Neptune	✓													✓										
Song														✓							✓			
SubSim	✓	✓								✓				✓	✓						✓			
Thetis	✓													✓	✓						✓			
UWSim	✓	✓												✓							✓			
WaVeSim	✓	✓												✓							✓			
WSS	✓	✓												✓							✓			
SimAUV	✓	✓												✓							✓			
MARS	✓	✓												✓							✓			✓

Tabelle 2.2.: Gegenüberstellung alle AUV-Simulatoren aus Abschnitt 2.5 in Bezug auf die eingesetzten Modelle. In grün sind die wesentlichen Eigenschaften hervorgehoben. Ältere Simulatoren in grau unterlegt.

2.7.2. Fortschritte zu älteren Simulatoren

Im Bereich der Forschung gibt es eine Vielzahl an AUV-Simulatoren, die in den letzten Jahrzehnten entwickelt wurden. Durch die technische Entwicklung verschob sich der Fokus von technischen Problemen, wie z.B. Netzwerkprogrammierung, zu der Entwicklung von interessanten Eigenschaften.

Schon früh bewährte sich der Einsatz von dreidimensionaler Grafik als Unterstützung der Visualisierung. Durch die technische Entwicklung im Grafikbereich sind realistischere und performantere grafische Unterwassereffekte möglich geworden. Hiervon profitieren die simulierten Kameras, die die Unterwassersicht wiedergeben. Die Terrain- und Objektmodellierung ist mit polygonreichen Modellen dadurch ebenfalls komplexer geworden.

Die Entwicklung einer eigenen Physik-Engine ist bei neueren Simulatoren weniger notwendig. Hier dominieren etablierte Systeme wie Bullet oder Newton. Allerdings wurden besonders früher eigene Systeme entwickelt. Die hydrodynamischen Modelle unterscheiden sich im Detaillierungsgrad zwischen den Simulatoren. Je nach Fokus fallen bestimmte Aspekte weg, allerdings bilden statischer Auftrieb und Widerstand die Grundlage. Die restlichen Modelle haben je nach Einsatzgebiet des Simulators unterschiedliche Schwerpunkte. Bei der Motorik wurden bereits früh Thruster modelliert. Der Detaillierungsgrad reicht von einfachen Kraftmessungen zu komplexen Modellen, in denen verschiedenste Parameter des Propellers betrachtet werden. Der sensorische Bereich ist hingegen unterentwickelt und wenn bestimmte Sensoren, wie z.B. Sonare modelliert werden, dann geschieht dies nicht realitätsnah.

Aufgrund der limitierten Rechenleistung war eine echtzeitfähige Simulation nur bedingt möglich. So wurde entweder offline simuliert oder mit einer niedrigen Aktualisierungsrate von 6 – 10 Hz und dies bezog sich häufig auf nur ein AUV. Durch die Zunahme an Rechenleistung ist eine Erhöhung der Aktualisierungsrate erfolgt. Gleichzeitig konnten auch die Modelle weiter verfeinert werden.

Als Architektur wird häufig eine verteilte netzwerkbasierte Architektur gewählt. Dies hängt damit zusammen, dass ältere Simulatoren nicht die Hardware zur Verfügung stand, um sämtliche Eigenschaften von AUVs zu simulieren. Dies hat sich mit leistungsstärkerer Hardware, besonders durch den Einsatz von Grafikkarten für die Visualisierung und Mehrkernprozessoren, geändert. Neue Simulatoren sind für gewöhnlich auf einem zentralen PC lauffähig.

Verfügbarkeit stellt ein Problem dar, weil die meisten Simulatoren nicht mehr aktiv weiterentwickelt werden oder aus keiner Quelle mehr bezogen werden können. Dazu kommen Systeme, in denen der Quellcode nicht zugänglich und nicht modifizierbar ist. Wenige Ausnahmen verwenden ein Plugin-System, um Modifikationen zu ermöglichen.

Abschließend ist festzustellen, dass ältere Simulatoren einen Fokus auf das hydrodynamische Modell, Thruster und einzelne AUVs legen. Die limitierten Rechenkapazitäten führen zu der Bevorzugung von einer verteilten Architektur und technische Hintergründe der damaligen Zeit werden hervorgehoben.

2.7.3. Anforderungen und Unterschiede

Schwärme von AUVs echtzeitfähig und im Kontext des Umweltmonitoring zu simulieren, erfordert bestimmte Eigenschaften. Es zeigt sich, dass eine Lücke bei AUV-Simulatoren in diesem Bereich existiert. Allerdings sind einige Einzelkomponenten in verschiedenen Simulatoren zu erkennen. Die folgende Aufzählung listet die entsprechenden Anforderungen im Detail auf. In Tabelle 2.2 sind diese Anforderungen in grün hervorgehoben.

1. Echtzeitfähigkeit
 - a) Online Simulation, um AUVs *live* zu simulieren und eine Benutzungsschnittstelle, um diese zu manipulieren.
 - b) Die Simulation muss in der Lage sein, performant mehrere AUVs zu simulieren, die untereinander kommunizieren.
2. Umweltmonitoring
 - a) Direktes Umweltmonitoring mit Druck, Temperatur, Salzgehalt und weiteren Parametern.
 - b) Indirektes Umweltmonitoring über Bioindikatoren, wie Fische und Pflanzen.
 - c) Weitere Sensorik, um die Umwelt wahrzunehmen, wie Sonare und Kameras.

Einzelnen Anforderungen werden von einigen Simulatoren erfüllt, wobei der Detailgrad der Realisierung unterschiedlich ausfällt. Zwar besitzt z.B. MarineSim Vegetation, allerdings ist diese statisch und reagiert nicht auf das AUV.

Grafische Benutzungsschnittstellen stehen selten im Fokus der Entwicklung eines Simulators und die meisten sind stark vereinfacht. Der Einsatz von Schwärmen erfordert allerdings Werkzeuge, um diese gebrauchstauglich zu verwalten. Dazu kommt, dass Unterstützung für den Einsatz mehrerer AUVs per se wichtig für den Einsatz von Schwärmen ist. Die meisten Simulatoren ermöglichen dies, aber kleinere Unterschiede gibt es im Bereich der heterogenen Roboter. Das bedeutet, dass bei einigen Simulatoren nur AUVs des gleichen Typs eingesetzt werden können. Steuerruder sind besonders bei Simulatoren für größere AUVs im Tiefseewasser anzutreffen. Auf der sensorischen Seite werden meist interne Sensoren für Lage und Orientierung unterstützt. Sonare und Kameras werden aufgrund des Simulationsaufwandes grob modelliert. Fischschwärme als indirekter Bioindikator und ein Energiemodell sind in bisherigen Simulatoren nicht vertreten. Pflanzen als direkter Bioindikator tauchen nur in einem Simulator in vereinfachter Form auf.

Bis auf wenige Ausnahmen spielen Unterwasserkommunikation und somit Schwärme oder komplexes Missionsdesign keine Rolle. Das Welt- und Umweltmodell beschränkt sich auf Terrain und einfachere Temperatur oder Salzverteilungen. Aufgrund der Tatsache, dass bei Tiefseesimulationen das Terrain sich zu weit weg von den AUVs befindet, wird dieses häufig vernachlässigt. Einfache Rauschmethoden wie Gaussverteilungen werden häufig eingesetzt, um realistischere Ergebnisse zu liefern.

MARS und SimAUV unterscheiden sich hinsichtlich der Anforderungen zueinander und zu den restlichen Simulatoren. Während bei der Entwicklung von SimAUV der Fokus auf der

Simulation eines AUV und der engen Anbindung an das HANSE-Projekt lag, ist bei MARS die Simulationen mehrerer allgemeiner AUVs im Umweltmonitoring in den Vordergrund gehoben. Des weiteren ist der Programmcode von MARS quelloffen und verfügbar⁶. MARS erfüllt die an Echtzeitfähigkeit, Simulation von mehreren AUVs in Form von Schwärmen und Umweltmonitoring gestellten Anforderungen sehr gut. Zum ersten Mal sind eine Fischschwarmsimulation und Energieerzeugung und -verbrauch Teil eines Modells. In diesem Modell sind Teillösungen anderer Simulatoren enthalten und bilden zum ersten mal ein gesamtheitliches Modell, um die gestellten Anforderungen abzudecken. Die nachfolgenden Kapitel beschreiben die zugrunde liegenden Modelle im Detail (siehe Kapitel 4) und evaluieren Teile der Anforderungen in Kapitel 6.

⁶<https://github.com/iti-luebeck/MARS/>, Zugriff am 10.11.2015

Kapitel 3.

Grundlagen

Der General, der eine Schlacht gewinnt, stellt vor dem Kampf viele Berechnungen an. Der General, der verliert, stellt vorher kaum Berechnungen an.

(Sun-Tzu)

Inhaltsangabe

3.1. AUV	31
3.1.1. HANSE	33
3.1.2. MONSUN	35
3.2. Fluidmechanik	36
3.2.1. Hydrostatik	37
3.2.2. Strömungslehre	39
3.3. Unterwasserakustik	42

In diesem Kapitel werden Grundlagen betrachtet, die in den folgenden Kapiteln benötigt werden. Abschnitt 3.1 stellt die zwei AUVs HANSE und MONSUN vor, die modelliert wurden und im Evaluationskapitel 6.2 und 6.3 verwendet werden. Abschnitt 3.2 beschäftigt sich mit den physikalischen Grundlagen, die im Kapitel 4 für die Modellierung benötigt werden. Dabei handelt es sich hauptsächlich um Hydrostatik und Strömungslehre für das AUV-Modell. Im letzten Abschnitt 3.3 werden die Grundlagen zur Kommunikation unter Wasser erläutert.

3.1. AUV

Autonome Unterwasser Vehicle (AUV) sind Robotersysteme, die mobil und selbständig Aufgaben im Unterwasserumfeld erfüllen können. AUVs besitzen ein Antriebssystem, typischerweise Motoren in Form von Thrustern. Auf der sensorischen Seite dominieren Sonarsysteme aufgrund der optischen Probleme, was aber Kamerasystem auf kurze Distanz nicht ausschließt. Zu den internen Systemen gehören ARHS (Attitude Heading Reference

System) und GPS. Daneben CTD-Sensoren mit Druck- und Temperatursensoren. Für die Kommunikation an der Wasseroberfläche werden W-Lan oder Satellitenverbindungen eingesetzt, während unter Wasser hauptsächlich akustische Systeme zum Einsatz kommen. Zusätzliche kommen je nach Anwendung Messsensoren zum Einsatz, um z.B. physikalische Eigenschaften zu messen. Eingesetzt werden AUVs z.B. zur Überwachung, Suche, Minenabwehr, Kartografierung oder Exploration.

Ein beispielhaftes AUV ist das REMUS 6000 aus der REMUS Familie, zu sehen in Abbildung 3.1. Es wurde erfolgreich eingesetzt, um die Überreste eines Flugzeugabsturzes zu finden [PGP⁺11]. Daneben wird es für Minensuche [SAA⁺01] eingesetzt und um chemikalische Ausflüsse zu finden [FPLA03]. Neben den klassischen AUVs existieren auch noch Gleiter. Ein Gleiter ist dabei ein spezieller Typ eines AUV. Gleiter erzeugen ihren Vortrieb durch Auftriebsveränderung, z.B. über einen Ballasttank, und nicht über Thruster. Das Absinken im Wasser in Kombination mit der Form und den Flügeln ergibt einen ähnlichen dynamischen Auftriebseffekt wie bei einem Segelflugzeug. Diese Form der Fortbewegung hat den Vorteil, dass sie energiesparender ist, da nicht ständig aktiv ein Motor laufen muss. Zwar können Gleiter ähnliche Aufgaben erfüllen wie normale AUVs, aber sie sind weniger manövrierfähig und stärker äußeren Einflüssen ausgesetzt wie z.B. Strömungen. Das macht sie weniger geeignet für Gewässer wie z.B. Flüsse oder Häfen.

Ein Beispiel für einen Gleiter ist der *Slocum Glider* [WCA⁺10] (siehe Abbildung 3.2). Dieser verfügt über die Standard CTD Sensoren und über Sensoren für Sauerstoff- und Chlorophyll-Messungen. Eingesetzt werden Gleiter um z.B. Phytoplankton zu beobachten [FGC09]. Eine modifizierte Version überquerte sogar den Atlantik [Rut15].

Daneben existiert noch die Klasse der Remotly Operated Vehicles (ROVs), die allerdings kabelgebunden und nicht oder teilweise autonom arbeiten. Das Kabel dient zur Kommunikation und zur Stromversorgung. Auch ROVs können für Suche, Inspektionen und Exploration eingesetzt werden, erfordern aber Personal und sind durch das Kabel eingeschränkt [LMP⁺05]. Dies macht sie weniger geeignet für Schwärme.

Eine zukünftig wichtige Aufgabe für AUVs ist das Umweltmonitoring. Dabei werden AUVs eingesetzt, um Parameter in der Umwelt aufzuzeichnen und zu überwachen. Es kann sich dabei um einfache physikalische Eigenschaften handeln wie z.B Temperatur, um chemische wie Stickstoffgehalt, aber auch um biologische wie Fische und Pflanzen. Anhand dieser Daten können Forscher Aussagen über die Wasserqualität oder das Ökosystem treffen.

Weil ein einzelnes System bei einem Fehler zu einem Missionsabbruch führen kann, und weil mehr als ein AUV eine Aufgabe in der Regel schneller erfüllen kann, sind Schwärme eine Lösung. Es existieren Schwarmprojekte im AUV-Bereich, wie z.B das Serafina-Projekt [ser] oder CoCoRo [STT⁺11]. Diese befinden sich aber in einem frühen Stadium.

In den nachfolgenden Unterkapiteln werden die AUVs HANSE und MONSUN vorgestellt. Diese beiden kleinen AUVs dienen als Evaluationsgeräte für Experimente und sollen modelliert und simuliert werden. Aufgrund der Verfügbarkeit beider AUVs und da beim MONSUN Projekt Schwärme und Umweltmonitoring ein zentraler Bestandteil sind, eignen sich diese beiden AUVs besonders zur Evaluation. Die gemeinsame Softwarebasis aufbauend auf ROS vereinfacht die Kommunikation mit der Simulation.



Abbildung 3.1.: Das REMUS 6000 AUV. Bild aus [Mar], mit freundlicher Genehmigung von Hydroid Inc.



Abbildung 3.2.: Der Slocum Gleiter. Bild aus [Res], mit freundlicher Genehmigung von Ben Allsup (Teledyne Webb Research).

3.1.1. HANSE

HANSE (Hanseatic Autonomous Nautic-Bot for SAUC-E) [FHK12] ist ein AUV, das von Studenten der Universität zu Lübeck für den SAUC-E [sau] Wettbewerb konzipiert wurde. HANSE (siehe Abb. 3.3) wiegt ca 19,5 kg und hat die Abmessungen von 60cm×30cm×50cm (BHT). Aufgrund des low-cost Ansatzes betragen die Materialkosten des HANSE Projekts seit 2009 ungefähr 14.000 €.

Der interne Aufbau ist in Abbildung 3.4 dargestellt. Als Antrieb dienen vier SeaBotix Thruster [Rod07]: zwei für den Vortrieb und zwei für das Abtauchen. Kommuniziert wird über eine optionale Kabelverbindung oder über W-Lan an der Wasseroberfläche. Für die Tiefenbestimmung wird ein Drucksensor verwendet. Zwei Webcams ermöglichen es, Pipelines zu folgen oder Objekten auszuweichen. Als Hauptrecheneinheit wird ein kleiner Laptop verwendet. Auf diesem läuft ein selbst geschriebenes Qt Framework, bestehend aus drei Schichten (siehe Abbildung 3.5). Diese wurde für die neueste Version [HTH⁺12] auf ROS portiert [QCG⁺09]. Auf der untersten Schicht (Drivers) sind die Treiber angesiedelt. Auf der zweiten Schicht (Behaviours) folgen Regler und Lokalisierung. Die oberste Schicht (Tasks) besteht aus Verhalten.

Im Entwicklungsprozess wurde SimAUV eingesetzt, um die Verhalten zu testen (siehe Unterabschnitt 2.4). Der hauptsächliche Einsatzzweck von HANSE waren die Aufgaben beim



Abbildung 3.3.: Der autonome Unterwasserroboter HANSE [FHK12].

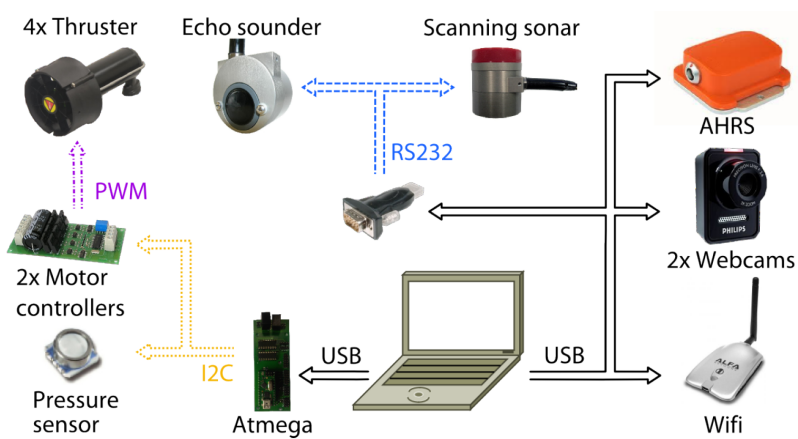


Abbildung 3.4.: Der schematische Aufbau von HANSE [FHK12].

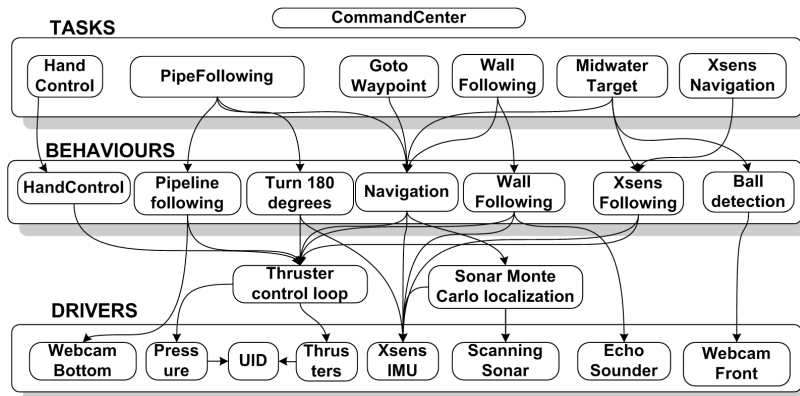


Abbildung 3.5.: Der schematische Aufbau der Software von HANSE.

Wettbewerb. Diese umfassten z.B. Pipeline-Following, Wall-Following und Geräuschquellendetektion.



Abbildung 3.6.: Der aktuelle autonome Unterwasserroboter MONSUN [MEIM14].

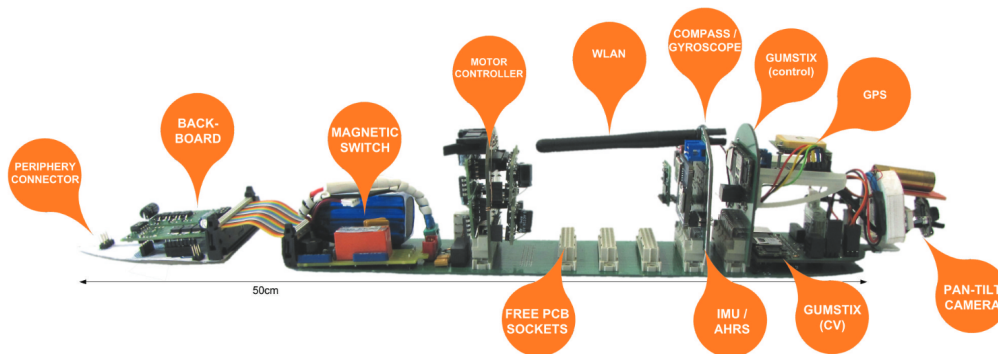
3.1.2. MONSUN

MONSUN (MONitoring System and Underwater Navigation Robot) ist ein autonomer Unterwasserroboter, der am Institut für Technische Informatik entwickelt wird [Mey13] [MEIM14]. Seit den Anfängen des Projekts ist die Schwarmfähigkeit und Umweltmonitoring ein wichtiger Entwicklungsaspekt [OPM12] [OMAM12] [OLM09].

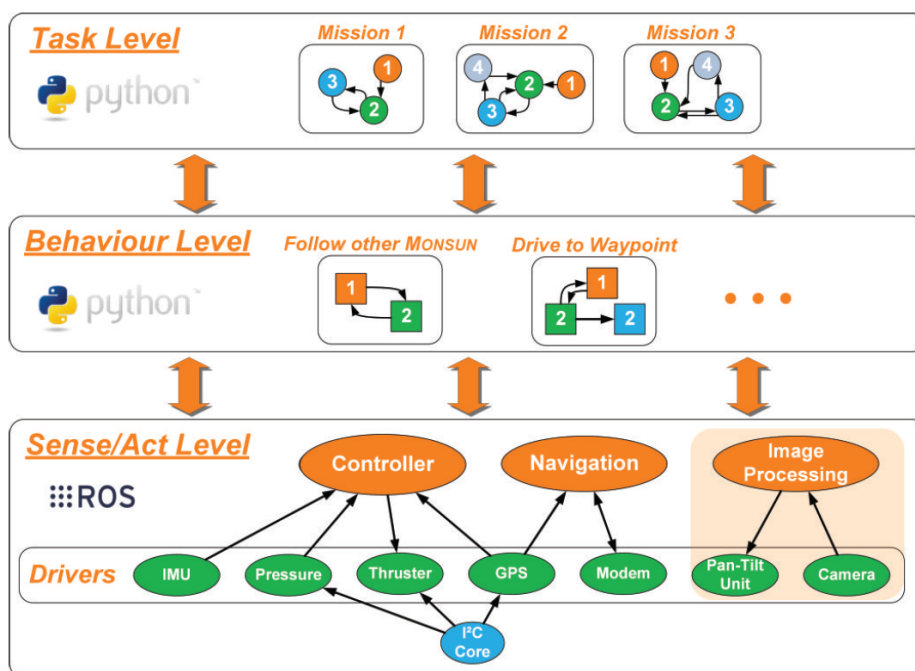
MONSUN hat eine Länge von 60 cm und einen Durchmesser von 10 cm bei einem Gewicht von 3 kg (siehe Abbildung 3.6). Aufgrund seines modularen Aufbaus lassen sich weitere interne Komponenten oder Zwischenmodule hinzufügen, z.B. um Proben zu nehmen oder Umweltsensoren anzuschließen. Zwei Thruster sorgen für den Vortrieb, während sich vier Thruster um die Tiefen- und Lageregelung kümmern. Der interne Aufbau ist in Abbildung 3.7 a) sichtbar. Als Hauptrechner dienen zwei Gumstix Miniaturcomputer. Einer davon ist für die Bildverarbeitung reserviert. Um die Umgebung zu erfassen, wird eine Pan-Tilt Kamera verwendet. Weitere interne Sensoren sind ein AHRS und GPS. Zur Kommunikation im Schwarm dient ein selbst entwickeltes Modem [OM10][RGM⁺15]. Für Überwasserkommunikation kommt W-Lan zum Einsatz.

In der Software dient ROS als Basissystem [QCG⁺09]. Darauf aufbauend besteht die Softwarearchitektur aus drei Schichten, ähnlich zu HANSE (siehe Abbildung 3.7 b)). Auf der untersten Schicht (Sens/Act Level) befinden sich die Treiber, Navigation, Regler und Bildverarbeitung. Auf der zweiten Schicht (Behaviour Level) gibt es einfache Verhalten, wie z.B. einen Wegpunkt anzufahren. Die oberste Schicht (Task Level) ist für die Missionsplanung zuständig.

Bisher fanden Tiefen- und Lageregelungstests statt [MEIM14]. Des Weiteren ein einfaches kamerabasiertes Folgeverhalten über *blob detection*, bei dem ein MONSUN einen zweiten MONSUN erkennt und ihm folgt [OMAM12]. Außerdem ein Ausweichverhalten, bei dem MONSUN beim Erkennen eines Hindernisses zurücksetzt und unter diesem hinweg taucht [OLM09]. Im Bereich Schwarm gibt es bisher nur simulierte Ergebnisse im Bereich Formationsfahrt [AMO13] und Load-Balancing Verhalten [ATM14]. Siehe Abschnitt 6.3 für eine genauere Erläuterung.



(a) Der schematische Aufbau von MONSUN [MEIM14].



(b) Der schematische Aufbau der Software von MONSUN [MEIM14].

Abbildung 3.7.: Das AUV MONSUN.

3.2. Fluidmechanik

Dieser Unterabschnitt beschreibt die physikalischen Grundlagen, die im Modell verwendet werden. Als Basis wird die allgemeine Hydrostatik erläutert (3.2.1). Darauf aufbauend folgt der statische Auftrieb (3.2.1.1). Abschnitt 3.2.1.2 geht auf die Stabilität eines schwimmenden Objektes ein, was in direktem Zusammenhang zum Auftrieb steht. Abschnitt 3.2.2 erläutert, wie sich Strömung und Widerstand auf ein Objekt unter Wasser auswirken.

3.2.1. Hydrostatik

Als Hydrostatik bezeichnet man die Lehre von unbewegten Flüssigkeiten. Entscheidend ist dabei die Dichte des Fluids und die Schwerkraft. Auf jeden Partikel des Fluids wirkt die Schwerkraft und zieht ihn nach unten. Der entstehende Druck wird als hydrostatischer Druck bezeichnet. Berechnet wird der Druck $P(h)$ für eine bestimmte Höhe der Wassersäule h über das Pascalsche Gesetz (siehe Gleichung 3.1). Die Dichte des Fluids ρ wird multipliziert mit der Schwerebeschleunigung g und der Höhe der Wassersäule h .

$$P(h) = \rho \cdot g \cdot h \quad (3.1)$$

Der hydrostatische Druck verteilt sich gleichmäßig über das ganze Volumen, unabhängig von der Form des Körpers (Pascalsches Prinzip). Daraus resultiert das Phänomen, dass als hydrostatisches Paradoxon bezeichnet wird. Dabei nehmen die Randbegrenzungen (z.B. Gefäßwände) einen Teil der Kraft des Fluids auf, so dass nur die Höhe ausschlaggebend ist. In Unterabschnitt 3.2.1.1 wird der Auftrieb und in Unterabschnitt 3.2.1.2 die Schwimmstabilität von Körpern in Fluiden erläutert.

3.2.1.1. Auftrieb

Auftrieb entsteht wenn sich ein Körper vollständig oder partiell innerhalb eines Fluids befindet und das Fluid verdrängt. Wiegt der Körper weniger als das verdrängte Wasser so entsteht Auftrieb. Entscheidend ist, dass die mittlere Dichte des Körpers geringer ist als die des verdrängten Fluids. Der statische Auftrieb ist auch bekannt als Archimedisches Prinzip, nach seinem Entdecker.

Der Auftrieb entsteht genauer durch die Schwerkraftwirkung des Fluides und Druckunterschied an der Unter- und Oberseite des Körpers. Abbildung 3.8 illustriert das Prinzip. Das Fluid übt eine Aufkraft und Abkraft auf den Körper aus. Daraus ergibt sich gemeinsam die Auftriebskraft F_a . Seitliche ansetzender Fluiddruck ist aufgrund des Pascal'schen Prinzips auf einer Höhe gleichwertig und hat somit keinen Einfluss auf den Auftrieb. Gegen die Auftriebskraft wirkt die Schwerkraft F_g und zieht den Körper nach unten. Es ergeben sich drei Fälle. Wenn $F_a > F_g$ so steigt der Körper auf. Bei $F_a < F_g$ sinkt er und bei $F_a = F_g$ befindet er sich in einem Gleichgewichtszustand. Die Formel für den statischen Auftrieb ist in Gleichung 3.2 zu sehen. Die Auftriebskraft F_a besteht demnach aus dem Volumen V des Körpers innerhalb des Fluids multipliziert mit der Dichte des Fluids ρ . Dadurch erhält man eine *virtuelle* Masse, die abschließend mit der Erdschwerebeschleunigung g multipliziert wird.

$$F_a = \rho \cdot V \cdot g \quad (3.2)$$

Die Auftriebskraft wird nicht auf den Schwerpunkt angewendet sondern wirkt im Auftriebspunkt. Dieser kann unter Umständen erheblich vom Schwerpunkt abweichen und beeinträchtigt die Schwimmstabilität (siehe 3.2.1.2). Gleichung 3.3 aus [TM15] S. 308, berechnet den Schwerpunkt eines in Bezug auf die Dichte gleich verteilten Objektes. Die Gleichung lässt sich für die Auftriebspunktberechnung verwenden, wenn die Fluidichte

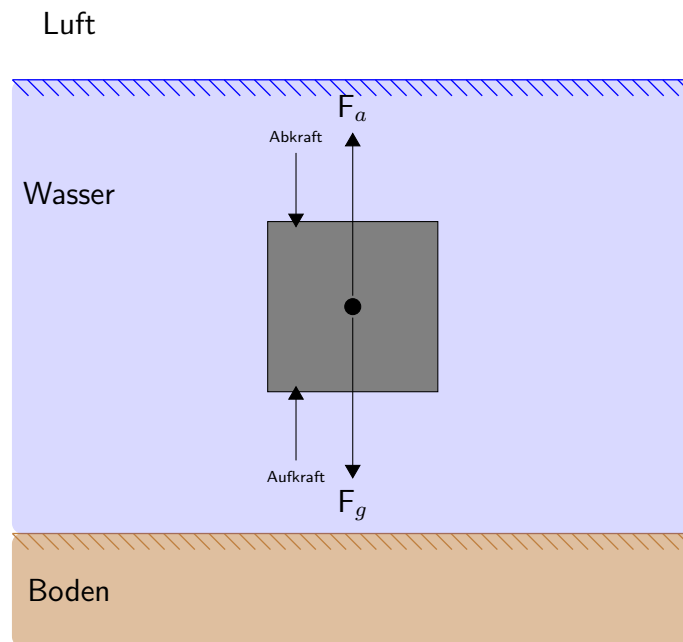


Abbildung 3.8.: Auftriebskraft F_a wirkt auf einen Körper im Fluid und führt zu seinem Auftauchen. Die Schwerkraft F_g wirkt dem entgegen und versucht den Körper nach unten zu ziehen.

verwendet wird. Der Auftriebspunkt im dreidimensionalen Raum (Komponenten P_x, P_y und P_z) berechnet sich aus der Summe von Einzelstücken ($m_i x_i$), von denen der Auftriebspunkt bekannt ist. Über eine geeignete Diskretisierung, wie z.B. Würfel oder Quader, von denen der Mittelpunkt geometrisch einfach berechnet werden kann, ist die Berechnung des Auftriebspunktes möglich.

$$P_x = \frac{\sum m_i x_i}{m}; P_y = \frac{\sum m_i y_i}{m}; P_z = \frac{\sum m_i z_i}{m} \quad (3.3)$$

Neben dem statischen Auftrieb existiert der dynamische Auftrieb. Diesen Effekt kann man bei Flugzeugen oder Unterwassergleitern beobachten. Beim dynamischen Auftrieb bewegt sich das Objekt innerhalb des Fluids vorwärts. Durch die Form des Körpers kommt es zu einer zeitlichen Verzögerung, wenn Fluidpartikel sich um das Objekt herum bewegen. Bei einer Tragfläche benötigen die Partikel aufgrund der Flügelform länger, um den Weg unterhalb zu durchschreiten, weil sie langsamer sind als oberhalb. Der Geschwindigkeitsunterschied entsteht durch Druckunterschiede (Bernoulli-Gleichung). Der Druckunterschied zwischen dem Tiefdruck oberhalb des Flügels und dem Hochdruck unterhalb des Flügels wiederum führt zu einer Auftriebskraft nach oben.

3.2.1.2. Stabilität

Wenn der Auftriebspunkt nicht mit dem Schwerpunkt zusammenfällt, so wird die Schwimmstabilität beeinträchtigt. Der Körper neigt sich und Schiffe können kentern. Änderungen des Auftriebspunktes entstehen bei Übergängen des Körpers zwischen Fluidschichten mit unterschiedlicher Dichte, wie z.B. von Wasser und Luft, oder bei einem Wassereinbruch. Der Schwerpunkt ändert sich z.B. durch Verlust von Komponenten, wie einem Thruster. Abbildung 3.9 illustriert die drei vorhandenen Fälle. Wenn der Auftriebspunkt (gelb) des Körpers über dem Schwerpunkt (rot) liegt, so wird die Lage als *stabil* bezeichnet. Es gibt ein korrigierendes Drehmoment nach links und der Körper wird versuchen, seine ursprüngliche Gleichgewichtslage einzunehmen. Diese ist erreicht wenn der Auftriebspunkt in einer Achse über dem Schwerpunkt liegt. Wenn der Schwerpunkt oberhalb des Auftriebspunktes liegt, so wird der Körper als *labil* bezeichnet. Ein Drehmoment tritt hier ebenfalls auf, allerdings führt es zum Kippen des Körpers. Fallen Schwerpunkt und Auftriebspunkt ineinander so wird der Körper als *indifferent* bezeichnet. Unabhängig von einem Stördrehmoment bleibt der Körper in seiner Gleichgewichtslage.

Abbildung 3.10 illustriert das entstehende Drehmoment wenn Auftriebs- und Schwerpunkt nicht zusammenfallen und das Stabilitätskriterium über das Metazentrum. Das Metazentrum ist der Schnittpunkt der Schwimmachse mit der Auftriebskraft beim neuen Auftriebspunkt in Schiefelage. Die Schwimmachse wird bestimmt, indem in Ruhelage eine Linie von Schwerpunkt zum Auftriebspunkt gezogen wird. Die Länge zwischen Schwerpunkt und Metazentrum bezeichnet man als Metazentrische Höhe h_m und ermöglicht die Feststellung der Schwimmstabilität eines Körpers. Die Schwimmlage ist (*labil*), wenn $h_m < 0$, *indifferent* bei $h_m = 0$ und *stabil*, wenn $h_m > 0$ gilt [Sig14] S.59. Das resultierende Drehmoment \vec{D} aus der Metazentrischen Höhe h_m und der Schwerkraft F_g (siehe Gleichung 3.4) bildet eine Art von Hebelarm.

$$\vec{D} = \vec{h}_m \times \vec{F}_g \quad (3.4)$$

3.2.2. Strömungslehre

Wenn ein Körper sich relativ zum umgebenden Fluid bewegt, so wird ihm ein Widerstand entgegengebracht, der ihn verlangsamt. Dieser Widerstand steigt mit der Fluidichte, somit ist Widerstand im Wasser ein wichtige Größe. Der Widerstand ergibt sich durch Volumen- und Druckkräfte und Viskosität (Reibung und Turbulenzen). Die Basis zur allgemeinen Beschreibung bilden häufig die Navier-Stokes Gleichungen. Diese sind aufgrund der Nichtlinearität und Differenzialquotienten zweiter Ordnung schwierig zu berechnen. Analytische Lösungen sind meistens nur über Vereinfachungen zu erbringen. Numerische Näherungslösungen mit Hilfe von Computern (CFD, engl. für Computational Fluid Dynamic) sind dagegen nicht schnell genug, um in Echtzeit verlässliche Ergebnisse zu liefern. Der Gesamtwiderstand eines Körpers besteht aus den folgenden Einzelkomponenten: Flächenwiderstand (Reibungswiderstand), Formwiderstand (Druckwiderstand), Interferenzwi-

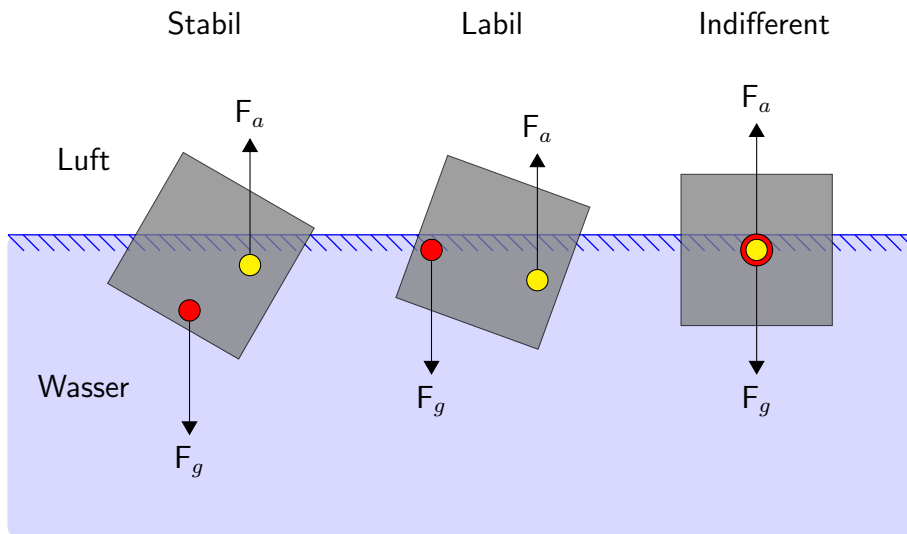


Abbildung 3.9.: Die drei Stabilitätszustände eines Körpers innerhalb des Fluids. Entscheidend ist die Position des Auftriebspunktes (gelb) und Schwerpunktes (rot).

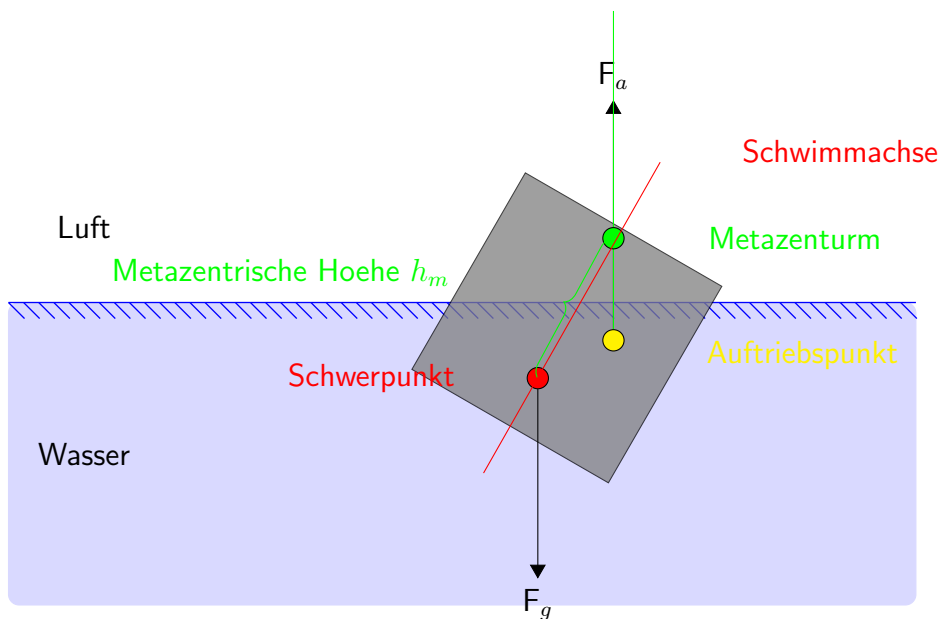


Abbildung 3.10.: Berechnung des Drehmoments bei einer Schiefelage des Körpers.

derstand und an der Wasseroberfläche der Wellenwiderstand. Die Körperoberfläche und Form sind die bestimmenden Größen. Das Zusammenspiel dieser beiden ist komplex und schwer auseinander zu halten. Interferenzwiderstand entsteht, wenn eigenständige Körper zusammenrücken und sich gegenseitig beeinflussen. Wellenwiderstand entsteht durch





Körperform	Anströmrichtung	$F_{W,R}$	$F_{W,D}$
längs angeströmte Platte		$\approx 100\%$	$\approx 0\%$
Stromlinienkörper		$\approx 90\%$	$\approx 10\%$
Kugelzylinder		$\approx 10\%$	$\approx 90\%$
quer angeströmte Platte		$\approx 0\%$	$\approx 100\%$

Abbildung 3.11.: Aufteilung des Gesamtwiderstandes in Formwiderstand $F_{W,D}$ und Reibungswiderstand $F_{W,R}$ für verschiedene Körperformen. Werte entnommen aus [Sig14] S.315.

die Umströmung und Bewegung des Wellensystems. Flächenwiderstand entsteht, wenn die Oberfläche eines Körpers in Kontakt mit dem Fluid tritt und damit Reibung entsteht. Der Flächenwiderstand ist experimentell schwierig nachzuweisen, so wird zumeist eine Differenzrechnung über dem Gesamtwiderstand und Formwiderstand durchgeführt. Formwiderstand ist von der Form des umströmten Körpers abhängig. Es bildet sich im vorderen Bereich ein Staupunkt mit hohem Druck. Durch Ablösung bilden sich Wirbel und ein Totraumgebiet im hinteren Bereich mit minimalen Druck. Durch diese Druckdifferenz entsteht der Formwiderstand. Abbildung 3.11 illustriert das Zusammenspiel der beiden Widerstände. Bei einer längs angeströmten Platte besteht der Gesamtwiderstand zu fast 100% aus dem Flächenwiderstand $F_{W,R}$. Ändert sich die Form hin zu einer quer angeströmten Platte so beträgt der Formwiderstand $F_{W,D}$ nahezu 100%. Die anderen Widerstände (Interferenz und Wellen) können in der Regel vernachlässigt werden.

Trotz der Komplexität der Navier-Stokes Gleichungen lässt sich eine vereinfachte Formel für den Gesamtwiderstand angeben (siehe Gleichung 3.5). Der Gesamtwiderstand F_d berechnet sich aus der Fluidichte ρ multipliziert mit der projizierten Anströmfläche (Stirnfläche) A und der Relativgeschwindigkeit v zum Quadrat. Alle nicht analytische fassbaren Einflüsse sind durch den Strömungswiderstandkoeffizienten C_w repräsentiert. Der C_w wird entweder experimentell bestimmt durch CFD oder indem man aus einer Vergleichstabelle eine dem Körper ähnlich Form auswählt, in dem der C_w eingetragen ist.

$$F_d = 1/2 \cdot C_w \cdot \rho \cdot v^2 \cdot A \quad (3.5)$$

3.3. Unterwasserakustik

Unterwasserakustik wird im Bereich der AUVs für Sensorik und Kommunikation eingesetzt. Mit Hilfe von Sonaren können unter Wasser Hindernisse auf weite Entfernung detektiert werden und mit Unterwassermodems weit kommuniziert werden.

AUVs bewegen sich unter Wasser fort und benötigen Sensorik, um sich in dieser Umgebung zu orientieren. Eine Möglichkeit sind optische Sensoren wie z.B. Kameras, die das Licht einfangen. Allerdings wird die Bildqualität durch Partikel und Schwebestoffe im Wasser eingeschränkt und ist von der Umwelt abhängig. In größeren Tiefen ist der Anteil des natürlichen Lichts gering, so dass eine aktive Lichtquelle wie z.B. Lampen verwendet werden müssen. An der Wasseroberfläche kommt zusätzlich der Tag- und Nachtzyklus zum tragen. Der Einsatz von künstlichen Lichtquellen kann die Sichtqualität weiter einschränken, wenn, wie bei Autoscheinwerfern im Nebel, das Licht durch Partikel im Wasser gestreut wird. Hinzu kommen aufwändige Bildverarbeitungsalgorithmen. Weitere Sensoren wie Laserscanner, Infrarot- oder Ultraschallsensoren, die problemlos an der Wasseroberfläche eingesetzt werden können, funktionieren nur eingeschränkt unter Wasser. Bei licht-basierten System kommt zusätzlich die Absorption im Wasser hinzu.

Um effizient Schwärme zu bilden, ist Kommunikation unter den Mitgliedern wichtig. Da AUVs eine mobile autonome Plattform darstellen, ist eine drahtgebundene Kommunikation mit Schwierigkeiten verbunden. Das Kabel kann sich verheddern und muss vom Gewicht mitgeschleppt werden. Deshalb bietet sich, wie in der Luft, drahtlose Kommunikation an. Dabei unterscheidet sich die Kommunikation unter Wasser von der über Wasser. Bedingt durch das Medium Wasser fallen elektromagnetische Wellen als Übertragungsmöglichkeit weg. Entscheidend ist hierbei die Leitfähigkeit des Wassers, die zu einer starken Dämpfung führt, besonders im hochfrequenten Bereichen. Es existieren zwar Lösungen, diese erfordern allerdings große Antennen. Es existieren ebenfalls optische Übertragungsmöglichkeiten, allerdings muss eine visuelle Verbindung bestehen, die von der Umwelt abhängig ist. So darf sich kein Objekt zwischen den Teilnehmern befinden und Schwebeteilchen, wie Sand und Pflanzen im Wasser, beeinflussen den optischen Kanal zusätzlich in Bezug auf seine Übertragungskapazitäten. Hinzu kommt die generelle Absorption von Licht durch Wasser. Bei Einsatz von Lasern müssen der Empfänger und Sender exakt ausgerichtet werden, was bei einem sich bewegendem AUV im flüssigen Medium Wasser schwierig ist. Wegen der vielen Probleme bei optischer Sensorik, elektromagnetischer und optischer Kommunikation hat sich als Lösung im Medium Wasser die Nutzung von akustischen System durchgesetzt.

Bei Unterwasserakustik wird eine Schallwelle ausgesendet, die bei einem Empfänger eintrifft oder einem Objekt auftrifft. Bei einem Auftreffen auf ein Objekt werden Teile der Schallwelle, je nach genauer Beschaffenheit des Objektes, wieder reflektiert. Durch die Zeit die die Welle vom Sender zum Objekt und wieder zurück benötigt, lässt sich die Distanz zu diesem berechnen. Die Schallwelle selbst kann dazu verwendet werden, Informationen zu versenden, indem verschiedene Frequenzen verwendet werden. Die Schallwelle benötigt im Gegensatz zu elektromagnetische Wellen ein Trägermedium wie z.B. Luft oder Wasser.

Die Geschwindigkeit von Schall unter Wasser liegt deutlich unterhalb der von elektromagnetischen Wellen. Je nach Beschaffenheit des Wassers liegt sie bei ungefähr 1500 m/s. Beeinflusst wird sie von Temperatur, Druck und Salzgehalt. Um die Schallwelle zu erzeugen werden Druckunterschiede erzeugt, z.B. durch Piezokeramiken, die sich dann in Form einer Welle fortbewegen. Die Energie des Signals verringert sich mit der Entfernung durch Absorption im Wasser. Dies wird durch die Bewegung der Moleküle verursacht. Hinzu kommt die Form der Ausbreitung der Schallwelle selbst, die dazu führt, dass diese sich über einen immer größeren Bereich verteilt und abschwächt. Während die Schallwelle sich ausbreitet, trifft sie auf andere Objekte, wie z.B. den Boden oder die Wasseroberfläche und wird von diesen reflektiert. Dies führt im Falle des Bodens zu einer weiteren Absorption und bei der Wasseroberfläche zu einer weiteren Streuung. Bei akustischer Kommunikation führen Reflexionen zur Mehrwegeausbreitung (engl. multipath propagation) des Signals. Dabei überlagert sich ein Signal zeitlich versetzt über mehrere Wege mit sich selbst, was zu einer zusätzlichen Störung führt. Eine planare Wasseroberfläche kann dabei als perfekter Spiegel angenommen werden, mit fast keinerlei Verlust der Signalstärke [Lur10] S.31-32. Reflexionen mit dem Boden sind aufgrund von dessen Zusammensetzung deutlich komplexer [LH12]. So kann ein hoher Sedimentanteil sich stark dämpfend auswirken. Das Medium Wasser ist zusätzlich nicht kollisionsfrei, da alle Teilnehmer senden können und sich die Signale überlagern können. Die Signalqualität wird weiter durch Rauschen gemindert. Dazu zählen künstliche Schallquellen wie Schiffsschrauben oder natürliche wie Luftblasen, Walgesänge, Wellen und die Brown'sche Molekularbewegung der Wassermoleküle.

Kapitel 4.

Modell

Geben Sie mir ein Ping, Vassili. Aber bitte nur ein einziges Ping.

(Jagd auf Roter Oktober)

Inhaltsangabe

4.1. AUV	46
4.2. Sensoren	50
4.2.1. CTD-Sensor	50
4.2.2. Sonar	53
4.2.3. Kamera	55
4.2.4. Mapping Sensor	56
4.3. Aktoren	56
4.3.1. Thruster	56
4.3.2. Ballasttank	58
4.4. Rauschen und Ausfall	60
4.5. Umwelt	60
4.5.1. Vegetation	61
4.5.2. Fischschwärme	63
4.5.3. Strömung und Wellen	67
4.6. Schwarm	68
4.6.1. Kommunikation	69
4.6.2. Energie	73

In diesem Kapitel wird auf das entwickelte Model eingegangen, wie es definiert wird und welche Unterschiede es zur Realität aufweist. Abschnitt 4.1 beschäftigt sich mit dem AUV Model. Abschnitt 4.2 behandelt die verschiedenen Sensormodelle, die von den Sensoren innerhalb des AUVs verwendet werden. Eine genauere Betrachtung findet statt in

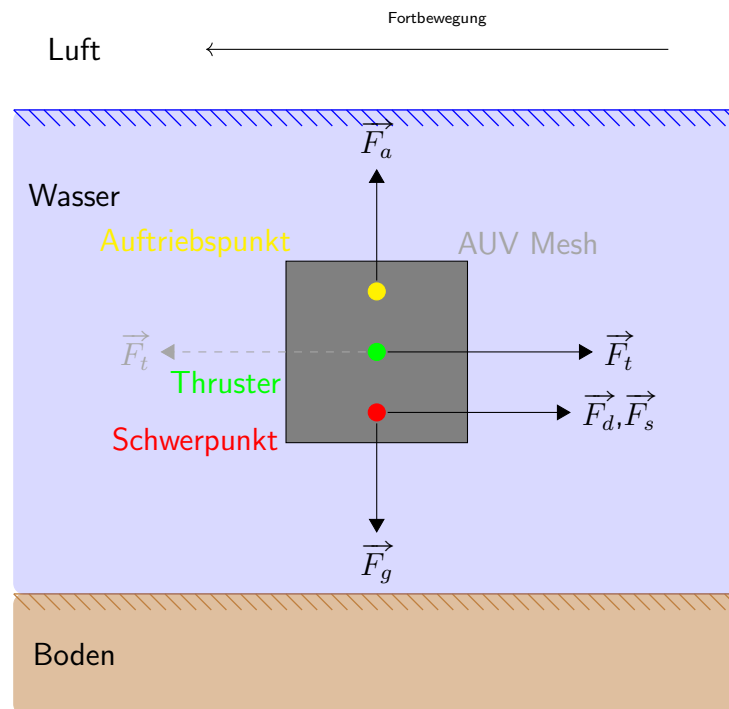


Abbildung 4.1.: Verschiedene Kräfte, die sich auf das AUV auswirken. Die Kraft der Thruster F_t kann auch in Gegenrichtung wirken.

Bezug auf Sonare, Kameras und allgemeine Umweltsensoren basierend auf statischen Kartendaten. Abschnitt 4.3 betrachtet die Aktorenmodelle für die Fortbewegung des AUVs. Dies sind Thruster und Ballasttanks. Abschnitt 4.4 behandelt, wie Ungenauigkeiten und Fehler in das Modell integriert wurden. Dies ist das Verrauschen von Sensordaten oder der Ausfall von ganzen Systemen, wie z.B. eines Motors. Abschnitt 4.5 geht auf das Umweltmodell ein. Dabei handelt es sich um Strömung, die sich auf das AUV auswirkt. Vegetation und Fischschwärme ermöglichen es indirektes Umweltmonitoring zu betreiben. Abschnitt 4.6 erläutert zwei Komponenten, Kommunikation und Stromverbrauch, die für die Schwarmfähigkeit des Modells wichtig sind.

4.1. AUV

Das wichtigste Teilmodell der Simulation ist das des AUV selbst. Es bestimmt, wie sich das AUV in Weltmodell fortbewegt und von ihm, anderen AUVs und vom Umweltmodell beeinflusst wird. Das AUV-Modell ist ein Starrkörpermodell und besteht aus sechs Freiheitsgraden (3 Translation, 3 Rotation). Es wird berechnet, welche physikalischen Kräfte entstehen und diese über die Physik-Engine (siehe 5.2.1.2) in eine Positions- und Rotationsänderung umgesetzt. Wenn Kräfte abseits des Schwerpunktes angesetzt werden, so resultiert dies in einem Drehmoment. Abbildung 4.1 zeigt alle berechneten und auf das

AUV wirkenden Kräfte. Dies sind die Gewichtskraft \vec{F}_g , Auftriebskraft \vec{F}_a , die Kraft, die Aktoren wie z.B. Thruster erzeugen \vec{F}_t , Wasserwiderstand \vec{F}_d und Strömungswiderstand \vec{F}_s . Die Gesamtkraft \vec{F}_{gesamt} besteht somit aus:

$$\vec{F}_{gesamt} = \vec{F}_a + \vec{F}_g + \vec{F}_t + \vec{F}_d + \vec{F}_s \quad (4.1)$$

Kräfte werden an dem berechneten Schwerpunkt angesetzt, mit Ausnahme der Auftriebskraft \vec{F}_a , die im Auftriebspunkt wirkt. Des Weiteren werden externe Kräfte \vec{F}_t , die z.B. über Thruster erzeugt werden, am entsprechenden Punkt angesetzt, an dem der Thruster im Modell positioniert wurde. Im Koordinatensystem des AUV gibt es den Nullpunkt, das Zentrum des Mesh. Der Mesh ist das dreidimensionale grafische Modell selbst und wird für die optische Darstellung, Kollisionen und Volumenberechnung verwendet. Von dem Nullpunkt ausgehend werden die weiteren Punkte definiert, an denen die Kräfte ansetzen, wie z.B. der Schwerpunkt. Sämtliche Aktoren und Sensoren werden starr im AUV-Modell positioniert und an dieses gebunden, in Relation zum Nullpunkt. Den Schwerpunkt zu berechnen, gestaltet sich aufgrund der verschiedenen und komplexen Masseverteilungen von AUVs als schwierig. Die Informationen über alle Komponenten und ihre Massen müssten ins Modell übertragen werden. Experimentelle Messungen durch Planimeter sind zwar möglich, aber aufwendig [San06]. Deshalb wird der Schwerpunkt vom Benutzer gesetzt. Die Physik-Engine läuft mit einer durchschnittlichen Rate von 60 Hz. Somit werden alle Kräfte 60 mal in der Sekunde neu berechnet und auf das AUV angewendet. Die Aktualisierungsrate der Kräfte kann einzeln eingestellt werden, wenn es die Performanz erfordert. So kann z.B. die Aktualisierungsrate der Auftriebsberechnung mit der gleichen Rate laufen wie die der Physik-Engine (60 Hz) oder mit einer geringeren (30 Hz). Bei einer Lücke zwischen den Aktualisierungen wird der zu letzt berechnete Wert verwendet. In der Parametertabelle A.1 stehen die drei wichtigsten Aktualisierungsraten:

- U_d Aktualisierungsrate für den Wasserwiderstand
- U_f Aktualisierungsrate für die Strömung
- U_b die Aktualisierungsrate für den Auftrieb

Die Rate für die Gewichtskraft und die Thruster beträgt fest 60 Hz, da sie einfache Berechnungen darstellen. Die Physik-Engine berechnet die Gravitationskraft und wendet sie an, somit verbleiben die restlichen Kräfte, die es zu berechnen gibt. Ebenso werden die Kollisionen von der Physik-Engine behandelt, allerdings muss ein entsprechendes Kollisionsmodell definiert werden.

Das AUV kann mit anderen AUVs oder dem Terrain aus dem Weltmodell kollidieren. Die Überprüfung, ob eine Kollision vorliegt und deren Auswirkungen übernimmt die Physik-Engine. Dafür notwendig ist ein Kollisionsmodell (siehe Abbildung 4.2). Für ein gegebenes dreidimensionales grafisches Mesh wird eine *Kollisionsbox* definiert, die die ungefähren Ausmaße des grafischen Mesh besitzt. Die Kollisionsbox kann verschiedene geometrische Formen annehmen, z.B. Quader, Zylinder oder Kugel, um der Form des dreidimensionalen

Meshs möglichst nahe zu kommen. Alternativ ist es möglich, mehrere der Kollisionsboxen einzeln zu erzeugen und zu einer starr verbundenen Gesamtkollisionsbox zusammenzufügen. Kräfte, die bei einer Kollision auf eine der einzelnen Kollisionsboxen wirken, bewegen die restlichen Kollisionsboxen mit. Das Kollisionsmodell ist dadurch genauer, allerdings steigt der Rechenaufwand mit jeder Kollisionsbox. Die letzte Möglichkeit ist, für die Kollisionsbox den Mesh selbst zu verwenden (*Kollisionsbox_m*). Das Kollisionsmodell ist dadurch akkurat, aber je nach Komplexität des Mesh steigt der Rechenaufwand erheblich an.

Um die Auftriebskraft \vec{F}_a zu berechnen, benötigt es das Volumen (siehe 3.2.1.1) für den statischen Auftrieb. Um das Volumen zu berechnen wird ein strahlenbasierter Ansatz gewählt wie in Abbildung 4.3 illustriert. Mehrere äquidistante Strahlen werden in Richtung AUV-Modell in einem Kreismuster losgeschickt. Der Radius des Kreismusters errechnet sich aus der maximalen Ausdehnung der zugehörigen Bounding Box. Die Strahlen treffen verschiedene Teile des 3D-Modells (rote Punkte) und setzen sich zu Quadern (V_b) zusammen. Dadurch ist die Volumenberechnung auf die Volumenformel eines Quaders reduziert. Die Wasseroberfläche wird als Ende angesehen, so dass das für die Strahlen die maximale Reichweite darstellt. Um den vollständigen Auftrieb zu berechnen, wird auch der Auftrieb an der Wasseroberfläche mitgezählt, der entsteht, wenn Teile des AUVs an der Luft exponiert sind. Um diesen Wert zu erhalten, wird einmalig das vollständig Volumen berechnet und der vom Wasser verdrängte Teil abgezogen. Dies ist notwendig aufgrund der unterschiedlichen Dichten der Medien. Siehe Gleichung 4.2 für den Gesamtauftrieb \vec{F}_a , der aus dem Auftrieb unter und über Wasser besteht (F_{wasser} und F_{luft}). Der Auftrieb berechnet sich aus dem Volumen des Körpers unter Wasser V_{wasser} , multipliziert mit der Dichte von Wasser ρ_{wasser} und der Erdschwerebeschleunigung g . Die Berechnung funktioniert analog für den Luftteil.

$$F_a = F_{wasser} + F_{luft} = \rho_{wasser} \cdot V_{wasser} \cdot g + \rho_{luft} \cdot V_{luft} \cdot g \quad (4.2)$$

Mit diesem Ansatz werden auch konvexe Objekte behandelt. Allerdings entsteht je nach eingestellter Dichte der Strahlenverteilung ein Verlust oder Zugewinn von Volumen (V_- und V_+) aufgrund der Anpassung des beliebig geformten 3D-Modells an die Quader. Durch das Vorhandensein der Quader ist es vergleichsweise einfach den Auftriebspunkt zu berechnen. Dieser besteht aus der gewichteten Summe aller Quader (siehe Formel 3.3). Ein weiterer Vorteil besteht darin, dass der Nutzer einstellen kann, wie viele Strahlen ausgesendet werden sollen, um eine Balance zu finden zwischen Genauigkeit und Geschwindigkeit. Als Alternative zur reinen Volumenberechnung besteht ein analytischer Ansatz [ZC01], der vom Nutzer ausgewählt werden kann. Allerdings erfordert die Auftriebspunktberechnung zusätzlichen Aufwand und wird nicht durch das Verfahren abgedeckt. Des Weiteren werden sämtliche Vertices des Modells betrachtet, obwohl ein kleinerer Teil bereits ausreichend ist. Korrupte Mesh-Daten führen zu einer deutlichen Vergrößerung des Volumens, weil bei dem Verfahren immer mit dem Nullpunkt im Koordinatensystem verglichen wird. Wellen an der Wasseroberfläche wirken sich aus, indem sich das verdrängte Wasservolumen durch die Welle ändert. Dadurch ändert sich die Gesamtauftriebskraft. Durch die resultierende Auftriebspunktänderung wird die Schwimmstabilität (siehe 3.2.1.2) beeinflusst und das

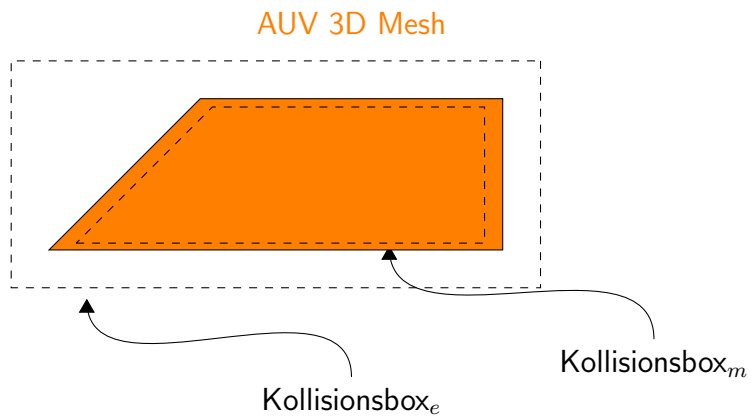


Abbildung 4.2.: Kollisionsmodell des AUV.

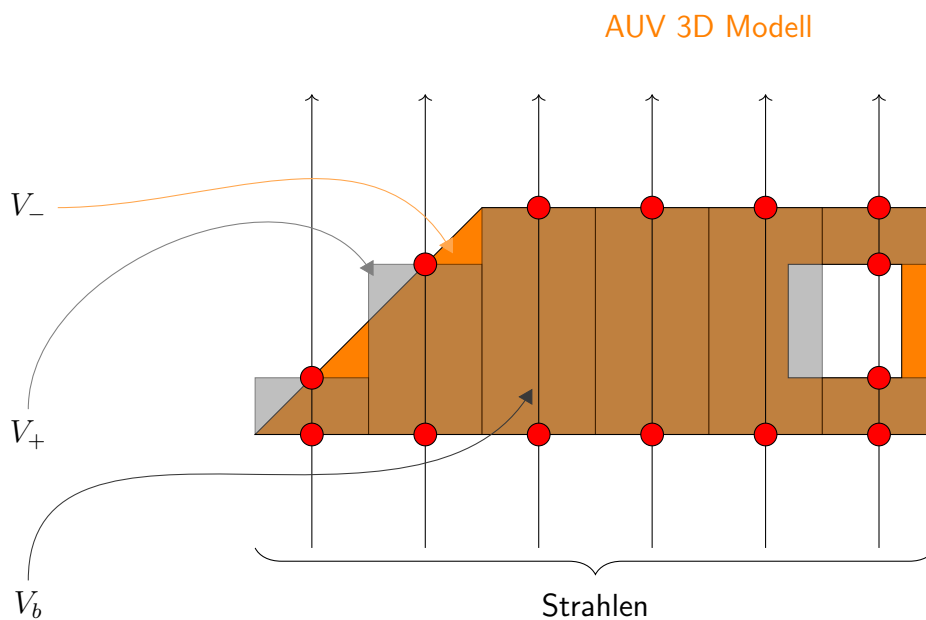


Abbildung 4.3.: Volumenmodell des AUV.

AUV kann stärkere Kippbewegungen ausführen.

Um den Wasserwiderstand F_d zu berechnen (siehe 3.2.2), benötigt es die angeströmte projizierte Fläche in Fahrtrichtung. Um diese zu erhalten, wird innerhalb der 3D-Engine jMonkeyEngine eine orthogonale Projektion verwendet. Dazu wird eine neue Rendersicht erstellt, die nur das AUV enthält. Die Auflösung dieser Sicht ist der Parametertabelle A.1 zu entnehmen, als Offview Camera Height und Width. Das AUV wird in entgegengesetzter Fahrtrichtung gerendert und die Pixel, die das AUV vereinnahmt, gezählt. Da die Höhe und Breite eines Pixel in Weltkoordinaten auf der Sichte ebene des Frustum (Kegelstumpf oder Sichtvolumen in der 3D-Grafik) bekannt ist, ist somit die Fläche bekannt, die ein Pixel vereinnahmt. Das Frustum ist das Sichtfeld der Kamera. Durch Multiplikation der

gezählten zum AUV gehörenden Pixel mit der Fläche eines Pixel erhält man die Gesamtfläche.

Die Thrusterkraft \vec{F}_t wird gesondert im Abschnitt 4.3.1 behandelt und der Strömungswiderstand \vec{F}_s im Abschnitt 4.5.3. In Tabelle A.1 im Anhang A.3 sind alle wichtigen Parameter des AUV Modells dargestellt mit beispielhaften Werten für das AUV HANSE.

4.2. Sensoren

Dieser Abschnitt beschäftigt sich mit den wichtigsten Sensormodellen, die eingesetzt werden, um AUVs zu simulieren. Dies sind die für die Tiefenbestimmung wichtigen Druck- und Temperatursensoren in Unterabschnitt 4.2.1. In Unterabschnitt 4.2.2 wird das strahlenbasierte Sonarmodell vorgestellt. Dadurch wird es möglich, bei schlechten Sichtbedingungen einen Eindruck der Umgebung zu bekommen. Unterabschnitt 4.2.3 stellt die Kamera vor. Unterabschnitt 4.2.4 erläutert die allgemeine Funktionsweise von Mapping Sensoren, z.B. für Salzgehalt oder Verschmutzung. Da einige Sensormodelle einfach gehalten sind, werden diese nicht weiter erläutert. Dies betrifft z.B. IMU-Sensoren, die lediglich die Beschleunigungswerte direkt aus der Physik-Engine auslesen, optional verrauschen und zurück geben. Eine vollständige Liste aller unterstützten Sensoren befindet sich Anhang A.5.

4.2.1. CTD-Sensor

Ein CTD-Sensor (auch CTD-Rosette genannt) ist eine Kombination von Sensoren um die Leitfähigkeit, Temperatur und Tiefe im Wasser zu bestimmen (von englisch Conductivity, Temperature, Depth). Über die Leitfähigkeit erhält man beispielsweise den Salzgehalt. Das Modell für den CTD-Sensor besteht aus drei einzelnen Modellen für den Druck-, Temperatur- und Leitfähigkeitssensor.

Im Gegensatz zur Lokalisierung in der Luft ist der Einsatz von GPS unter Wasser nicht möglich. Über Sonare oder LBL-System (engl. für long-baseline) lässt sich die Position unter Wasser feststellen, allerdings mit erheblichem Aufwand. Der Drucksensor erlaubt relativ einfach das Feststellen der Tiefenposition innerhalb des Fluids. Übliche Genauigkeiten von Drucksensoren zur Tiefenbestimmung liegen im Zentimeterbereich. Drucksensoren arbeiten z.B. über den piezoelektrischen Effekt. Das umgebende Fluid übt eine Kraft auf eine Fläche aus, die dann in einen elektrischen Strom umgewandelt und gemessen wird. Das Drucksensormodell ist einfach gehalten. Die Tiefenposition ist durch die verwendete 3D-Engine innerhalb des Modells bekannt, und es lässt sich die Tiefe direkt zurück geben. Alternativ erfolgt eine Umrechnung im mBar oder Pascal mit Hilfe des Pascalschen Gesetzes (siehe 3.2.1). Die Daten können optional verrauscht werden.

Temperatur ist ein wichtiger Umweltparameter und wird verwendet, um bei den Werten des Drucksensors die Temperatur zu kompensieren. Temperatursensoren verwenden zumeist den Effekt, dass sich der Widerstand ändert, wenn ein elektrischer Leiter Temperaturveränderungen ausgesetzt wird. Die Wassertemperatur bestimmt, welche Fische und Pflanzen

überleben können und beeinflusst das Strömungssystem. Ein Großteil der Sonnenenergie wird von den ersten Metern des Wassers absorbiert. Dies führt zu einem direktem Erhitzen der Oberfläche und einer Energiezufuhr für die Photosynthese von Pflanzen und Algen. Im Ozean und anderen Gewässern ist im Temperaturverlauf eine Thermokline zu beobachten. Als Thermokline wird der steile Temperaturübergang von einer Wasserschicht in eine andere bezeichnet. Abbildung 4.4 zeigt die typische Thermokline in einigen Ozeanregionen. Die Temperatur und Thermokline ist großen saisonalen Schwankungen unterworfen, besonders in mittleren Breitengraden [TPES11] S.79. Im Winter ist die Oberflächentemperatur niedrig und die Thermokline wird verkleinert. Zusätzlich kommt es durch Stürme und Wellen zu einer größeren Vermischung der Schichten. Ab einer Tiefe von 1500 Metern ist die Temperatur nahezu konstant. In Seen und langsam fließenden Gewässern bildet sich im Sommer ebenfalls eine Thermokline [Man10] S. 526. Diese reicht weniger tief und kann sich im Winter nicht halten, so dass ein See in diesem Fall eine gleichmäßig verteilte Temperatur aufweist.

Das Temperaturmodell verwendet eine eindimensionale Funktion, um die Temperatur in Abhängigkeit der Tiefe zu errechnen. In [Jos15] wurde eine Funktion erstellt auf Basis von Ozeandaten der Argo Datenbank. Die Funktion $T(D)$ in Formel 4.3 berechnet die Temperatur in Celsius in Abhängigkeit der Tiefe D in Metern. S ist die Temperatur an der Wasseroberfläche. Diese Funktion liefert ein typisches Temperaturprofil für den Ozean (siehe Abbildung 4.5). Sie gibt das Temperaturprofil für flachere Gewässer aber ungeeignet wieder. Deshalb wurde eine Erweiterung durchgeführt, um die Funktion über den Faktor a zu stauchen.

Das Modell für den Leitfähigkeitssensor besteht lediglich aus der Rückgabe eines vorher fest eingestellten Salzgehaltes.

$$T\left(\frac{D}{a}\right) = -0.338 + \frac{S * f(x)}{1.485 \times 10^{-4} * S * x + f(x)} \quad (4.3)$$

$$f(x) = 1 + e^{-0.016*x+1.244}$$

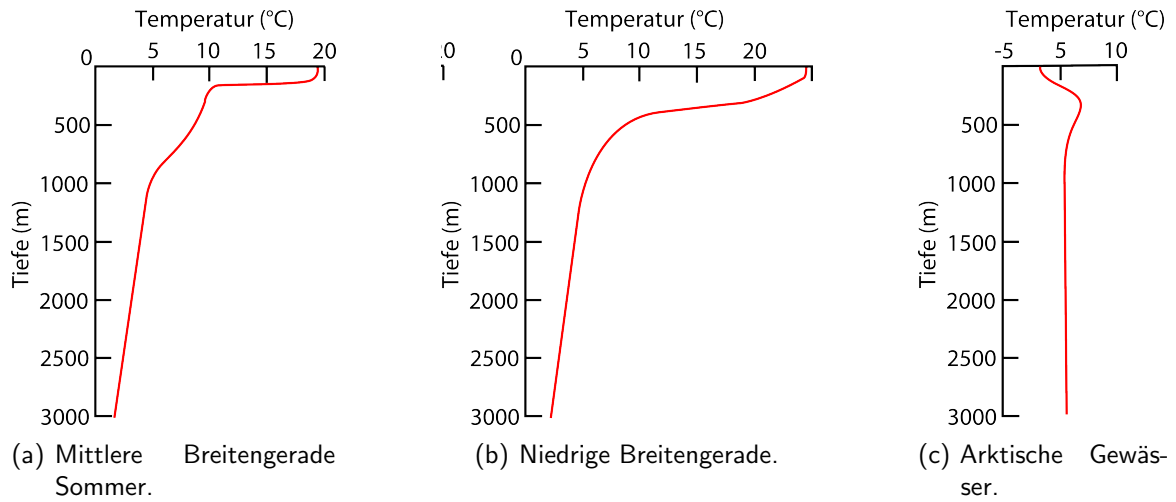


Abbildung 4.4.: Typische Thermokline vom Ozean in verschiedenen Regionen. Daten entnommen aus [Tea95], Kap.2, Seite 22.

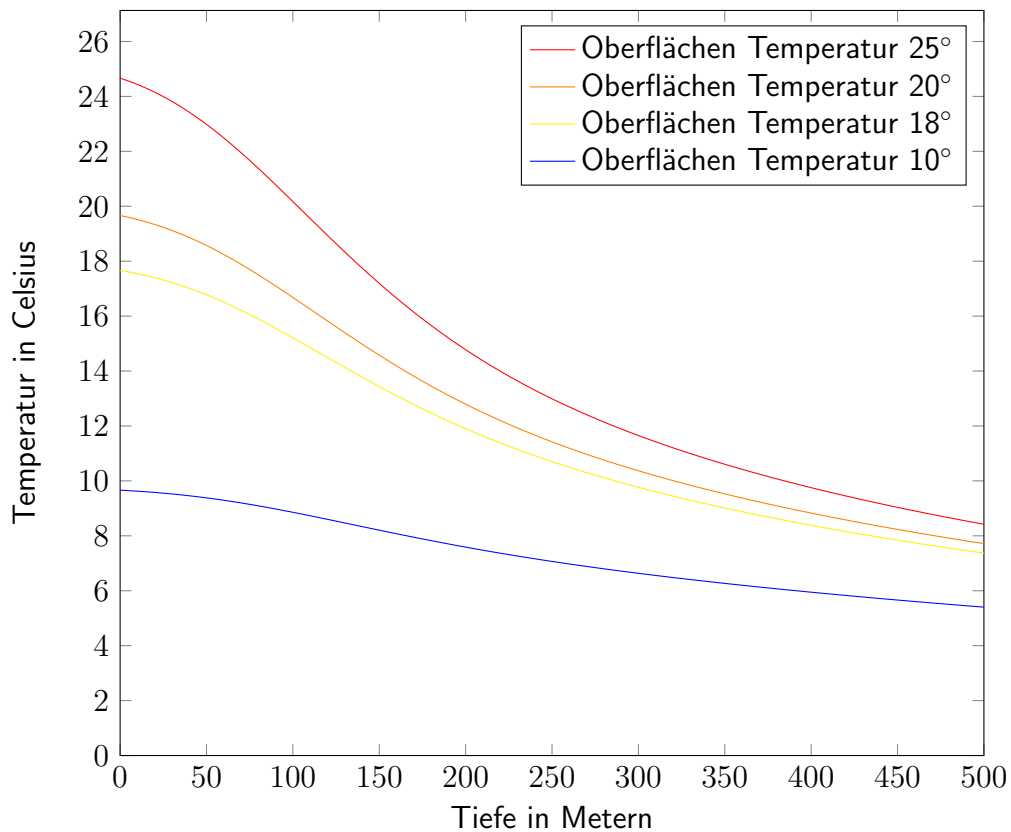


Abbildung 4.5.: Temperaturkurven für verschiedene Oberflächentemperaturen der Formel aus [Jos15]. a beträgt 1.

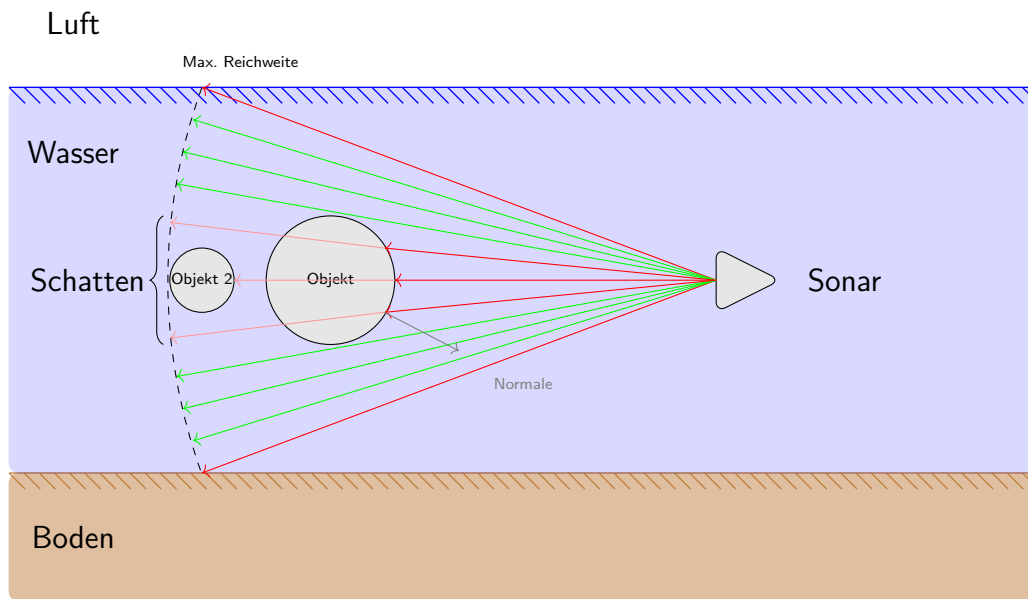


Abbildung 4.6.: Sonarmodell: Das Sonar sendet Strahlen in einem Muster aus, das Objekte trifft. Auf Basis der Distanz und des Auftreffwinkels werden die Daten gedämpft.

4.2.2. Sonar

Das Sonar ist einer der wichtigsten Sensoren für ein AUV. Aufgrund der schlechten Sichtverhältnisse und den Eigenschaften des Mediums Wasser sind optische Sensoren wie Kameras oder Infrarotsensoren weniger geeignet (siehe Abschnitt 3.3). Das Sonar ermöglicht, Distanzen der Umgebung zu messen und für die Lokalisierung zu verwenden. Es existieren verschiedene nicht echtzeitfähige Algorithmen und Modelle, um Sonare zu simulieren, die z.B. auf parabolischen Gleichungen basieren. Aufgrund der Anforderungen an das Gesamtmodell sind diese nicht geeignet. Eine Lösung ist die Nutzung von strahlenbasierten Ansätzen (Raycasting) wie in [Bel97].

Abbildung 4.6 zeigt das Sonarmodell. Es werden gerade Strahlen (rote und grüne) von einer Startposition (Sonar) ausgesendet. Diese treffen Objekte (rot) oder laufen ins Leere (grün). Innerhalb von Grafikengines existieren entsprechende Mechanismen, um das sogenannte Raycasting zu ermöglichen. Die in der Simulation verwendete jMonkeyEngine benutzt *bounding interval hierarchy trees* [WK06] für die performante Implementierung. Die Strahlen liefern eine Kontaktposition und die Normale der Oberfläche im Kontaktpunkt. Daraus lassen sich die Distanz und der Winkel berechnen. Strahlen ohne Kontaktpunkt werden ignoriert.

Um ein realistischeres Sonarbild zu erhalten muss mehr als ein Strahl gleichzeitig ausgesendet werden. Das Muster kann dabei variieren zwischen einem zentralen Strahl und davon ausgehend einer rechteckigen oder zirkulären äquidistanten Verteilung. Durch Änderung des Öffnungswinkels lässt sich die Verteilung weiter stauchen. Durch die fixe Einstellung

der Anzahl der ausgesendeten Strahlen lassen sich Werte für individuelle Sonarmodelle finden, die eine Balance zwischen realistischem Sonarbild und Berechnungsaufwand besitzen. Jeder Kontaktpunkt einer Oberfläche korrespondiert zu einem individuell festgesetzten Intensitätswert. Dieser Wert wird in ein Array gespeichert und die Position innerhalb des Array steht für die Distanz. Durch die Addition und Einordnung der verschiedenen Kontaktpunkte und derer Intensitätswerte entsteht ein realistisches Sonararray.

Um das Sonarbild realistischer zu machen, benötigt es eine Dämpfung und Rauschen. Sämtliche Strahlen besitzen eine maximale Reichweite, ab der diese keine weiteren Kontaktpunkte mehr erzeugen. Sollten sie keine Kontaktpunkte erzeugt haben, sind sie ins Leere gelaufen (grüne Strahlen). Der ursprüngliche Intensitätswert I_u wird zum einem durch die Distanz vom Kontaktpunkt zum Ursprung (Sonar) gedämpft, in dem dieser multipliziert wird durch die Distanz d normiert bezüglich der maximal möglichen Distanz d_{max} und mit einem zusätzlichen Faktor L (siehe Gleichung 4.4). Dies resultiert im gedämpften Intensitätswert I_d . Zusätzliche wird der Intensitätswert über den Winkel gedämpft. Die Normale im Kontaktpunkt wird dabei verwendet, um die Streuung zu modellieren. Der ursprüngliche Intensitätswert I_u wird multipliziert mit dem Winkel α zwischen Normale und Kontaktstrahl normiert auf $\pi/2$ und multipliziert mit einem zusätzlichen Faktor A (siehe Gleichung 4.5). Die beiden Faktoren können vom Benutzer eingestellt werden und sind standardmäßig auf 1 definiert.

$$I_d = I_u \cdot (d/d_{max}) \cdot L \quad (4.4)$$

$$I_d = I_u \cdot (\alpha/(\pi/2)) \cdot A \quad (4.5)$$

Auf die Intensitätswerte kann eine Gauss- oder Normalverteilung addiert werden, um das Sonarbild zu verrauschen. Bei Vorhandensein einer gemessenen Rauschfunktion eines realen Sonars kann diese aufaddiert werden. Um hinter Objekte zu blicken, die einen Sonarschatten werfen, werden die roten Strahlen, die ein garantiert kleines Objekt wie ein AUV treffen, weiter geführt. Nach einem Mindestabstand, der für den direkten Schattenwurf steht, wird ein weiterer Kontaktpunkt ermittelt. Der resultierende Intensitätswert wird abgeschwächt durch einen Faktor mitverwendet.

Die Nachteile dieses Modells sind CPU-lastige Berechnung und das Sampling der Strahlen. Das Raycasting wird in den Grafikengines auf der CPU ausgeführt, obwohl Ressourcen auf der GPU zur Verfügung stehen. Außerdem werden Objekte, die kleiner sind als der Abstand zwischen zwei Strahlen, nicht erfasst. Des weiteren werden keine Reflexionen betrachtet (siehe dazu auch Abschnitt 3.3). Der Boden absorbiert je nach Beschaffenheit einen Großteil der Schallwelle, über die Wasseroberfläche oder andere reflektierende Objekte wird ein Großteil der Schallwelle weiter ausgebreitet. Da die Ausbreitung der Schallwelle und Reflexionen nicht betrachtet werden, ist der "Schatten", den Objekte werfen, größer als bei einem echtem Sonar. Es kann somit schwieriger hinter ein Objekt geblickt werden und anhand des Schattenwurfs auf die Größe des Objekts geschlossen werden. Diese Probleme lassen sich über das Einsetzen weiterer Strahlen ausgehend vom Reflexionspunkt lösen. Wenn ein Strahl auf ein Objekt trifft, so wird ausgehend von dieser Position ein weiterer Strahl ausgesendet (rötliche Strahlen in Abbildung 4.6). Wenn dieser Strahl auf ein Ob-

jekt trifft das einen Mindestabstand, oder größer zum ersten Auftrittspunkt hat, so wird er abgeschwächt als Sonartreffer gewertet, ansonsten ist er zu nah am Objekt und folglich im Sonarschatten. Dies erhöht allerdings die Anzahl der Strahlen pro Reflexionspunkt und die CPU-Last. Zusatzinformationen zur Absorption bestimmter Objekte z.B. des Terrains könnten das Modell weiter verbessern. Als Alternativlösung, um die Performance und Genauigkeit weiter zu steigern, ist der Einsatz von Depth-Buffern der Grafikkarte möglich.

4.2.3. Kamera

Kameras werden seit jeher bei ROVs und AUVs eingesetzt. Die resultierenden Kamerabilder sind für den Menschen gut verständlich und über eine Fülle an Bildverarbeitungsalgorithmen können ebenfalls autonome Systeme Merkmale aus Bildern extrahieren. Allerdings ist die Bildqualität stark von der Umgebung abhängig. Dunkelheit, Pflanzen, Kaustiken und Schwebeteilchen verdecken den Hintergrund und schränken die Sichtweite ein. Eine Simulation der Kamera ist über moderne Grafik-Engines einfach möglich. Es wird eine zusätzliche *RenderView* erzeugt mit den nötigen Parametern, wie z.B. der Auflösung. Das Rendering kann mit entsprechenden Effekten versehen werden, um einer Untersicht näher zu kommen. Heutige Grafik-Engines, wie z.B. die in MARS verwendete *jMonkeyEngine*, ermöglichen die Verwendung von Effekten für Unterwassernebel, Kaustiken, Light-Scattering und Licht-Effekten. Um Schwebeteilchen darzustellen, können Partikel-Effekte verwendet werden. Diese erzeugen die Hauptrechenlast auf der CPU. Durch den Einsatz von Shadern lässt sich ein Schwebeteilchen-Effekt performant auf der GPU implementieren. Die Implementierung dieses Effekts entstand im Rahmen einer Bachelorarbeit [Jon14]. In der Implementierung wird zudem ein Verfahren von [FW07] verwendet, um durch mehrere Iterationen mehr Partikel zu erhalten.

Als Basis dient eine Rausch-Funktion, genannt *Simplex Noise* [Per01]. Diese ist eine Abwandlung des bekannten *Perlin-Noise* [Per02]. *Simplex Noise* ist eine mehrdimensionale Rauschfunktion. Sie liefert für den gleichen Eingabewert immer das gleiche Ergebnis und lässt sich gut auf der GPU implementieren. Beim *Simplex Noise* wird eine Gitterstruktur aufgespannt, bestehend aus gleichverteilten und gleichschenkligen Dreiecken im zweidimensionalen Fall. Jeder Eckpunkt erhält einen zufälligen Gradienten. Um einen Zufallswert für eine Koordinate zu erhalten wird die Gitterstruktur geschert und alle benachbarten Gradienten der Eckpunkte summiert. Somit erhält man einen Zufallswert für einen Punkt im Raum, der gleich bleibt. Durch den mehrdimensionalen Charakter lässt sich Zeit als zusätzliche Dimension verwenden.

Durch den Einsatz von *Simplex Noise* entsteht ein Partikel-Effekt, der räumlich gebunden ist und durch die Zeitdimension einer Variation unterliegt. Für ein Bild wird der *Simplex Noise* berechnet und überlagert. Ändert sich die Position der Kamera, so ändert sich das Ergebnis des *Simplex Noise*. Sollte die Kamera die selbe Position einnehmen wie zuvor, so wird der selbe *Simplex Noise* erzeugt. Durch die zusätzliche Zeitdimension wirken die Partikel *wabernd* oder *schwebend*. Der Partikel verschwindet nicht sofort, sondern wird zuerst kleiner. Das erzeugte Rauschen sieht punktförmiger aus und kommt einem Partikel

im Wasser nahe. Durch weitere Filter wird der Effekt verbessert [Jon14]. Die Partikel werden nicht vom AUV oder den Thrustern beeinflusst.

Abbildung 4.7 illustriert den Simplex Noise vor einem weißen Hintergrund. Der Ort der Aufnahme blieb fest und die zeitliche Dimension variabel. Man erkennt deutlich, wie über die Zeit von a) zu c) bestimmte Bereiche kleiner werden, bis sie ganz verschwinden (rot). Ebenso entstehen neue "Flecken", zu sehen im blauen Bereich.

4.2.4. Mapping Sensor

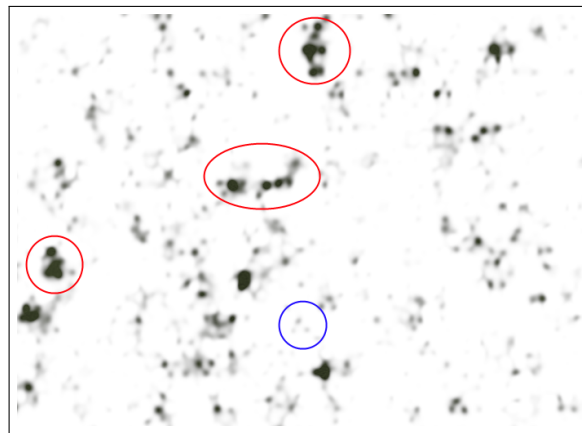
Unter Mapping Sensoren werden sämtliche Sensoren zusammengefasst, die einen Umweltwert aus der Simulation zurück geben. Die Basis bildet dabei eine zweidimensionale Karte (engl. Map) auf der die Werte gespeichert sind. Die X- und Y-Achsen zeigen die zweidimensionale Position im Raum. Am Punkt selbst ist der eigentliche Wert gespeichert. Die Realisierung erfolgt zu meist über Bilddateien. Dabei steht der Farbwert für den Umweltwert. In Kombination mit den eindimensionalen Funktionen, wie z.B. die des Temperatursensors (siehe 4.2.1), lässt sich ein dreidimensionaler Wert erhalten. Dazu wird eine zweidimensionale Temperaturkarte der Oberfläche erstellt. Der Oberflächenwert, an dem sich der Sensor befindet, wird als Startpunkt für die Temperaturfunktion verwendet. Ein Beispiel ist der Strömungssensor, der seine Werte aus dem Strömungsmodell bezieht, das ebenfalls auf Kartendaten basiert (siehe 4.5.3).

4.3. Aktoren

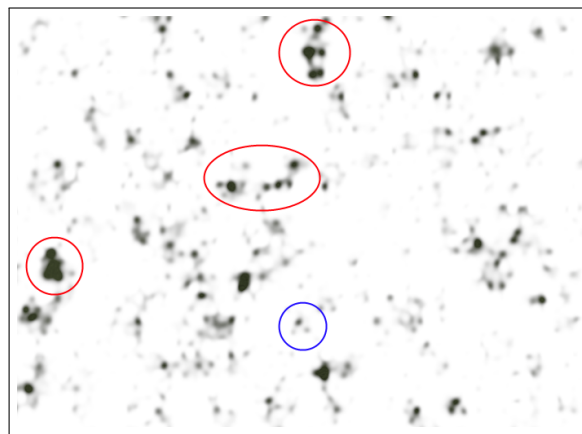
Dieser Abschnitt beschreibt die verwendeten Aktorenmodelle, um das AUV fortzubewegen. Das wichtigste Aktorenmodell ist das des Thrusters in Unterabschnitt 4.3.1. Auf Basis von Messungen wird eine Funktion erstellt, die fortan verwendet wird, um die Kraft zu berechnen. Des weiteren wird ein Steady State Modell als Alternative verwendet. Unterabschnitt 4.3.2 geht auf den Ballasttank ein, der für AUV mit energiesparender Tiefenregelung oder Gleiter gedacht ist.

4.3.1. Thruster

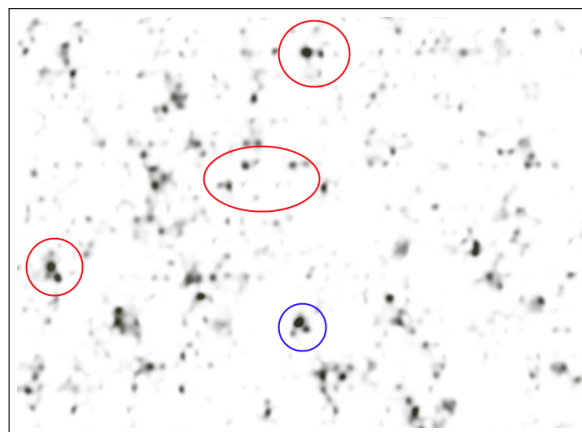
Der Thruster ist einer der wichtigsten Aktoren für die Fortbewegung eines AUV und präzises Manövrieren. Besonders beim Hovering, dem Schweben über einem Ziel, zeigt sich die Notwendigkeit. Die Funktionsweise eines Thruster ist wie folgt: Ein Motor treibt einen Propeller über eine Drehbewegung an. Die Form des Propellers erzeugt eine Sogwirkung im vorderen Bereich, der zum Ausstoß von Wasser nach hinten führt und ergibt einen Antrieb nach vorne. Bei Modellierung von Thrustern sind aufgrund der Komplexität von Strömungen in Fluiden und dem Einfluss des Thrusters auf das AUV selbst, verschiedene physikalische Eigenschaften zu beachten, so z.B. der Coandă-Effekt oder Küssner-Effekt [Fos02] S. 475. Dies macht die Modellerstellung komplex, aber es existieren verschiedene Thrustermodelle im wissenschaftlichen Bereich [WY99].



(a)



(b)



(c)

Abbildung 4.7.: Drei zeitliche Momentaufnahmen des Simplex Noise vor einem weißen Hintergrund.

Eine beliebtes Grundmodell für symmetrische Propeller ist in Gleichung 4.6 erläutert. In diesem *Steady State* Modell ist der Thrust T (Kraft) proportional abhängig zum Quadrat der Propellergeschwindigkeit ω . Der Koeffizient η gibt den Effizienzgrad des Propellers an, ρ ist die Fluidichte, a die Propellerfläche, p der Propellerwinkel und r der Radius des Propellers. Dieses Modell wird verwendet, um generische Thruster zu ermöglichen. Komplexe Modelle wie in [BLF00] sind präziser und arbeiten in verschiedensten Situationen. Allerdings erfordern diese nichtlinearen dynamischen Modelle n-ter Ordnung komplexe Experimente und Messungen um einzelne Parameter zu bestimmen.

$$T = \rho a r^2 \eta^2 \tan(p)^2 \omega |\omega| \quad (4.6)$$

Daneben wird ein auf experimentellen Messergebnissen aufbauendes Modell verwendet. Dazu wird der Thruster an einen Kraftsensor angebaut, der die Kraft in Newton zurückgibt. Der Thruster wird in ein Wasserbad gehängt und die Kraft und der Strom bei inkrementeller Steigerung der Drehgeschwindigkeit gemessen und aufgezeichnet. Die Messung wurde beispielhaft für den SeaBotix BTD150 Thruster vom HANSE AUV durchgeführt [Rod07] (siehe Abbildung 4.8 für die Messergebnisse). Auf die Daten wird ein Regressionsanalyse durchgeführt und die resultierenden Funktionen im Modell verwendet. Die Rückwärtsbewegung des Propellers wird gleich zur Vorwärtsbewegung gesehen. Gleichung 4.7 und 4.8 zeigen die errechneten Regressionskurven für Kraft und Strom. Für die Kraft wurde eine potenzielle Regression gewählt und für den Strom eine polynomiale Kurve 2-ten Grades. Die errechnet Kraft wird auf das AUV-Modell an der Position des Thrusters angesetzt, womit ein Drehmoment hinzukommt. Sollte sich die Position des Thrusters über Wasser befinden, so wird keine Kraft auf das AUV ausgeführt. Die Stromverbrauchsdaten werden für das Energiemodell in Unterabschnitt 4.6.2 benötigt.

$$f(x) = 0.00046655 \cdot x^{2.02039525} \quad (4.7)$$

$$g(x) = 0.00013053 \cdot x^2 - 0.00211047 \cdot x + 0.01576037 \quad (4.8)$$

4.3.2. Ballasttank

Ballasttanks sind neben Thrustern ein wichtiger Akteur für Gleiter und AUVs. Ein Ballasttank füllt sich über einen Motor oder Pumpe mit Wasser, was den Auftrieb des Körpers verändert und zu einer Bewegungsänderung führt. Ballasttanks sind wichtig für energieeffizient Systeme wie z.B. Gleiter. Die Auftriebsänderung in Kombination mit dem dynamischen Auftrieb resultiert in einer Vorwärtsbewegung des Gleiters. Wenn der Ballasttank voll gefüllt ist, ist ein weiterer Betrieb erst wieder notwendig wenn der Gleiter die definierte Auftauchtiefe erreicht hat. AUVs können Ballasttanks dazu verwenden zum Boden abzutauchen, um dort zu verweilen und um längere Messreihen durchzuführen. Da der Ballasttank keine Energie mehr benötigt, wenn er voll ist, kann Energie eingespart werden. Des Weiteren verbraucht die Tiefenregelung ebenfalls wenig Energie. Ein fehlerhafter Ballasttank kann zu einem Wassereinbruch führen oder zu einem dauerhaften Verweilen auf

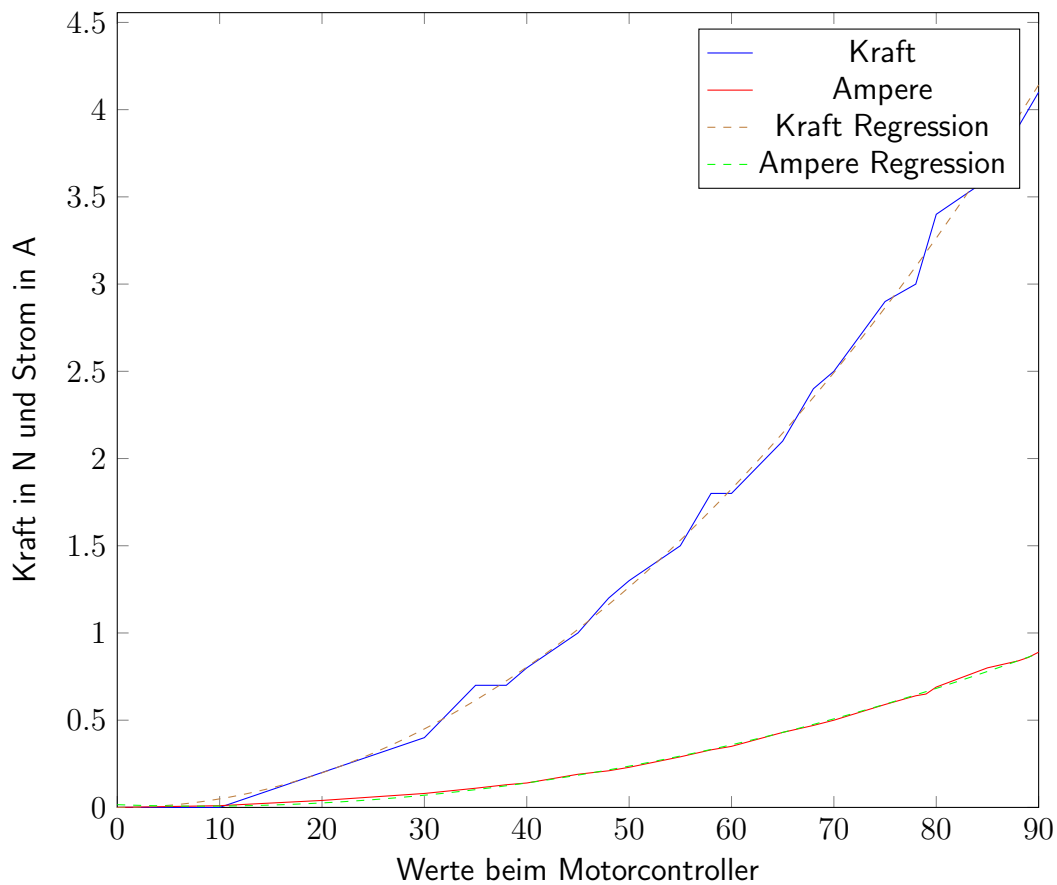


Abbildung 4.8.: Messergebnisse des SeaBotix BTD150 Thrusters [Rod07] für Kraft(blau) und Strom(rot) mit eingefügter Regression (gestrichelt).

dem Gewässergrund. Des weiteren verbraucht ein Ballasttank viel Platz im AUV Design, was bei kleineren AUVs wie dem MONSUN ein Problem darstellt.

Ballasttanks können auf verschiedene Weisen realisiert werden, so z.B. über Kolbentanks, Tauchsäcke oder Pressluft [Brü00]. Das Modell verwendet einen Kolbentank als Grundlage. Ein Kolbentank besteht aus einem zylindrischem Grundkörper. Über einen Kolben wird Wasser von der äußeren Umgebung aufgenommen oder wieder heraus gepresst. Der Einlaufstutzen stellt den Einsaugpunkt für das Fluid dar, während sich der Kolbentank an anderer Position befinden kann. Im Modell wird ein Grundvolumen als maximaler Wert angenommen, dass der Ballasttank besitzt. Das Volumen kann zwischen Null und dem Grundvolumen gesetzt werden. Das tatsächliche Annehmen des Volumens wird beim Ballasttank zeitlich verzögert durchgeführt. Dies ist notwendig, da normale Ballasttanks längere Ansaug- und Blaszeiten besitzen, die im Sekundenbereich liegen. Um sich auf das AUV-Modell auszuwirken, muss die Auftriebskraft berechnet werden. Dazu wird das zum Zeitpunkt der Kraftberechnung vorhandene Volumen in der Auftriebsformel (siehe 3.2.1.1) verwendet. Die erhaltenen Tauchtankauftriebskraft wird an der Position angewendet an

der sich der Ballasttank im Modell befindet. Somit kann es zu einem Drehmoment kommen und damit eine Lageänderung, wenn der Ballasttank ungünstig positioniert ist. Das Schwappen des Fluids, das entsteht wenn sich das AUV bewegt wird nicht betrachtet. Sollte sich der Einlaufstutzen über Wasser befinden, so wird kein Fluid angesogen.

4.4. Rauschen und Ausfall

Diese Abschnitt beschäftigt sich mit Rauschen und Ausfall von Daten, die Sensormodelle weiterleiten oder Aktorenmodelle aufnehmen. Sämtliche Daten, die von Sensoren erzeugt werden, können über eine Gleichverteilung oder Gaußverteilung mit Rauschen behaftet werden. Das gleich gilt für die Kraftberechnung der Aktoren. Zusätzlich kann jeder Sensor und Aktor de- oder aktiviert werden. Dadurch stellt er seinen Dienst ein, wodurch ein Ausfall eines System simuliert werden kann. In Bezug auf Schwärme ist es möglich, ein ganzes AUV zu deaktivieren. Dieses treibt dann im Wasser herum und ist nicht mehr kommunikationsbereit. Sensoren wie das Sonar haben ein zusätzliches erweitertes Rauschmodell (siehe 6.2.2). Die Unterwasserkommunikation verwendet ebenfalls ein erweitertes Rauschmodell (siehe 4.6.1).

4.5. Umwelt

Dieser Abschnitt beschäftigt sich mit dem Umweltmodell und den drei einzelnen Teilmodellen, aus denen es besteht. Als Umwelt wird hierbei die Beziehung vom AUV zu äußeren Einflüssen verstanden, wie z.B. Lebewesen oder physikalischen Kräften. Das Umweltmodell bildet die Basis, um komplexeres Umweltmonitoring über indirekte Bioindikatoren wie Fische und Vegetation durchführen zu können. Unterabschnitt 4.5.1 zeigt den Aufbau des Vegetationsmodells. Pflanzen sind ein wichtiger Bioindikator und es können durch ein entsprechendes Modell Bildverarbeitungsalgorithmen zur Zählung entwickelt werden. Des weiteren können AUV-Schwarmverhalten entwickelt werden, um große Bereiche effizient abzufahren. Unterabschnitt 4.5.2 erläutert das Fischeschwarmmodell. Fische sind ein wichtiger Bioindikator für Wasserqualität und da sie sich bewegende Lebewesen sind, sind sie schwer zu beobachten. Eine Simulationsmodell erleichtert die Entwicklung von AUV-Algorithmen für die Beobachtung. Unterabschnitt 4.5.3 geht auf das Strömungs- und Wellenmodell ein, das sich auf das AUV auswirkt und die Manövrierfähigkeit einschränkt. Die Tabelle im Anhang A.4 enthält die wichtigsten Umweltparameter des Umweltmodells.

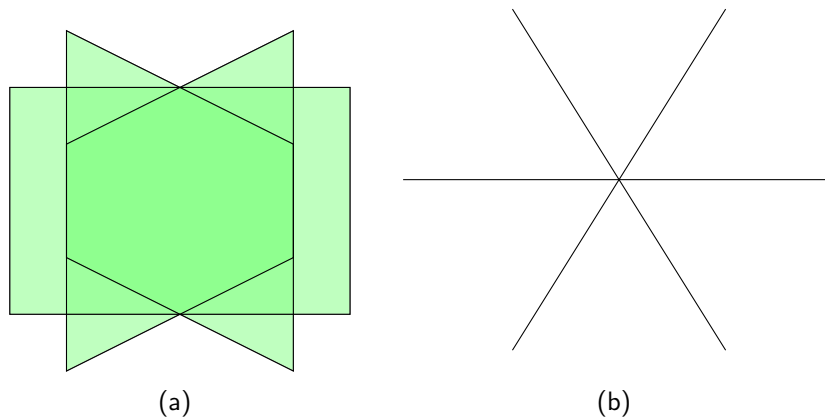


Abbildung 4.9.: Modell eines einzelnen Grasbüschels. (a) Schräge Draufsicht von vorne oben, in der man die drei einzelnen sternförmig angeordneten Flächen sieht. (b) Sicht von oben.

4.5.1. Vegetation

Vegetation unter Wasser ist ein wichtiger Bioindikator, um die Qualität eines Gewässers zu beurteilen und liefert Umweltdaten über räumliche und zeitliche Grenzen hinweg [Sch05]. Sie passt sich an ihre Umgebung an in Bezug auf Lichtintensität, Fließgeschwindigkeit, Sedimente, Kohlenstoffverfügbarkeit, pH-Wert und Nährstoffkonzentration. Unterwasserpflanzen verknüpfen außerdem den Boden mit dem Wasserkörper und erlauben die Untersuchung im Sinne eines Ökosystems. Vegetation bildet eine optische Verdeckung, wodurch andere Bioindikatoren schwerer zu erkennen sind oder Objekte von Interesse wie Pipelines. Somit ist es von Nöten, ein geeignetes Modell zu entwickeln, um Vegetation unter Wasser performant darzustellen. Die Implementierung des Vegetationsmodells entstand im Rahmen einer Bachelorarbeit [Bre14]. Die Idee der sternförmigen Anordnung und Animation einzelner Grasobjekte entstammt [Pel04].

Die Basis des Modells bildet ein einzelnes Grasbüschel. Durch DensityMaps wird in einem späteren Schritt die Verteilung dieser Grasbüschel festgelegt. Über ein Kachel- und Pagingssystem werden die Büschel gruppiert und verwaltet. Um die Bildqualität zu verbessern, werden abschließend LOD (Level-of-Detail), Fading und Animationen erläutert. Abbildung 4.9 zeigt das Modell eines einzelnen Grasbüschels. Es besteht aus 3 einzelnen planaren Flächen, auf die später die Grastextur projiziert wird. Die Flächen sind sternförmig angeordnet, so dass sichergestellt ist, dass sie von allen Blickrichtung gut sichtbar sind. Ein Nachteil ist die schlechtere Sichtbarkeit bei einer Sicht von oben. Die ist zu beachten, wenn Untersuchungen der Flora mit einem AUV in der Draufsicht durchgeführt werden. Die einzelnen Grasbüschel werden über eine Gauß-Verteilung in ihrer Skalierung variiert.

Im ersten Schritt muss die Verteilung der Grasbüschel durchgeführt werden. Aufgrund der großen Anzahl an Grasbüscheln, die notwendig sind, um eine natürlich Umgebung zu

erstellen, bietet sich eine automatisierte Verteilung auf Basis von Texturkarten an. Diese Texturen arbeiten ähnlich wie HeightMaps. In ihnen wird über Farbwerte die Information der Verteilung zweidimensional gespeichert. Diese Texturkarte wird *DensityMap* (engl. für Verteilungskarte) genannt. Ein Pixel der DensityMap korrespondiert mit einer vorher definierten Größe und Position im Weltmodell (Terrain). Da in einem Pixel potentiell mehr Grasbüschel vorhanden sein können, aufgrund der Ausmaße des Pixels im Weltmodell, wird der Platz über eine Gauß-Verteilung mit Grasbüscheln befüllt. Dabei wird ein vorher festgelegter Minimalabstand eingehalten. Die RGB-Werte innerhalb der DensityMap erlauben es, bis zu drei verschiedene Pflanzenarten zu kodieren. Dabei steht je einer der Farbwerte für einen Typ. Eine Mischfarbe resultiert in zwei bis drei Pflanzentypen, die sich in einem Pixel befinden.

Nach der Verteilung der Grasbüschel müssen diese sinnvoll organisiert und verwaltet werden. Dazu wird ein Paging-System verwendet. Hierbei wird die Menge an Grasbüscheln auf eine gleichmäßig verteilte Kachelanzahl aufgeteilt. Eine Kachel korrespondiert zumindest zu einem oder mehr Pixel in einem quadratischen 2^n -Muster und enthält sämtliche Grasbüschel dieses Bereichs. Anhand der Kameraposition werden einzelne Kacheln aus- und eingeblendet. Wenn sich eine Kachel innerhalb eines festgelegten Radius um die Kamera herum befindet, so wird sie eingeblendet. Ein zweiter größerer Radius ist für das Einblenden der detailarmen Kachel zuständig.

Um die Performanz zu steigern, wird ein Level-of-Detail Ansatz (LoD) verwendet. Aufgrund der Tatsache, dass in weiter Entfernung mehr Objekte zu sehen sind und diese folglich dargestellt werden müssen, muss die Darstellungslast verringert werden. Beim Vegetationssystem handelt es sich um eine Vielzahl von Objekten, so dass die Last besonders hoch ist. Bei einem LoD existieren von 3D-Modellen verschiedene Detailstufen mit unterschiedlicher Polygonanzahl. Das 3D-Modell wird dann in Abhängigkeit von verschiedenen Parametern gegen eines der reduzierten Modelle ausgetauscht. Ein üblicher Parameter ist dafür die Distanz von der Kamera zum Objekt. Für das Grasbüschel wurde ein zweites Modell erstellt, das aus einer Fläche besteht. Dieses wird verwendet, wenn die Entfernung zur Kamera größer wird. Durch den LoD und das Auswechseln der Kachel entsteht ein grafisch harter Übergang. Wenn eine Kachel ein- oder ausgewechselt wird, dann erscheint sie ohne zeitliche Verzögerung. Das gleiche betrifft den LoD. Dieser Effekt kann abgemildert werden, indem Fading verwendet wird. Beim Fading findet eine Überblendung zwischen der detailreichen und detailarmen Darstellung statt. Dies kann mit Hilfe des AlphaDiscard-Thresholds der jMonkeyEngine in Abhängigkeit der Entfernung über einen Shader realisiert werden.

Unterwasserpflanzen sind Strömungen ausgesetzt und dadurch in Bewegung. Die Animation der Grasbüscheln ist dabei über Vertex-Shader realisiert. Dem Vertex-Shader wird ein Vektor übergeben der die Richtung und Stärke der Strömung global kodiert beinhaltet. Durch die Übergabe eines zweiten Vektors ist ein zusätzlicher Thruster-Effekt möglich. Ein Thruster drückt in Bodennähe über das abgestrahlte Wasser die Pflanzen zur Seite oder wirbelt sie um. Dieser Effekt ist dadurch nachstellbar, dass die Position des Thrusters und die Entfernung zum Boden an den Shader übergeben werden. Dadurch verstärkt sich

der Effekt je näher der Thruster sich am Boden befindet.

4.5.2. Fischeschwärme

Fische sind ein wichtiger Bioindikator für Umweltmonitoring um die Gewässerqualität zu beurteilen [Boy82]. Des Weiteren ist es nötig, Fische in Aquakulturen zu beobachten, um sicher zu stellen, dass sie gesund sind und um Gefahren frühzeitig zu erkennen. Da Fische mobile Lebewesen sind, sind sie schwer zu beobachten. Um Fische zu beobachten, müssen entsprechende AUV-Algorithmen entwickelt und getestet werden. Die Implementierung des Fischeschwarmmodells entstand im Rahmen einer Bachelorarbeit [Fel14].

Das Grundmodell basiert auf der Arbeit von Craig Reynolds [Rey87]. Dieses Modell kann Herden, Schulen und Schwärme von verschiedenen Tierarten simulieren. Ein einzelner Teilnehmer wird als Boid bezeichnet. Aus dem Schwarmverhalten von Tieren leitete Reynolds drei einfache Grundregeln ab. Abbildung 4.10 zeigt die einzelnen Grundregeln: Separation a), Zusammenhalt b) und gleichartigen Ausrichtung c). Die Separation beschreibt die Neigung des aktiv betrachteten Boid (grün) einen bestimmten Abstand zum Rest der innerhalb des maximalen Radius befindlichen Boids (blau) zu halten. Boids, die sich außerhalb des Radius befinden (rot), werden nicht betrachtet. Die Positionen der blauen Boids werden von der Position des grünen Boid abgezogen, normalisiert und mit $1/r$ gewichtet. Wobei r den maximale Radius beschreibt. Somit berechnet sich die Abstoßung (roter Pfeil) des grünen Boid. Zusammenhalt beschreibt die Neigung der Boids beim Schwarm zu bleiben. Dafür wird die durchschnittliche Position (grüner Punkt) aller Boids innerhalb der Nachbarschaft (blaue Boids) bestimmt. Der aktiv betrachtete Boid (grün) muss die Differenz zwischen eigener und Schwarmmittelposition berechnen und erhält den Vektor, der die Neigung angibt (roter Pfeil). Die Ausrichtung beschreibt die Neigung der Boids sich ähnlich zu seinen Nachbarn auszurichten, hinsichtlich Winkel und Geschwindigkeit. Dazu werden die entsprechenden Durchschnittswerte in der Umgebung ermittelt. Der aktive Boid versucht seine Geschwindigkeit und Ausrichtung (grüner Strich) der durchschnittlichen anzupassen (blauer Strich). Reynolds erweiterte das Modell später um Verfahren zur Hindernisvermeidung und ein Such- und Fluchtverhalten [Rey99]. Dieses wird nicht verwendet, da Lösungen in der verwendeten jMonkeyEngine leichter und performanter zu realisieren sind im Hinblick auf echtzeitfähige Simulation von vielen Schwarmmitgliedern. Die wird durch die Kollisionseigenschaften der eingesetzten Physik-Engine ermöglicht. Das vorgestellte Modell bedient sich der drei Grundregeln von Reynolds.

Ein einzelner Boid wird in diesem, auf Reynolds basierenden Fischeschwarmmodell durch mehrere Vektoren beschrieben (siehe Abbildung 4.11 a). \vec{V}_p ist der Positionsvektor und gibt die Position des einzelnen Boid an. \vec{V}_b ist der Bewegungsvektor und gibt die aktuelle Richtung und über die Länge die Geschwindigkeit an. \vec{V}_l ist der Lenkungsvektor und beschreibt den Drang des Boid sich in eine bestimmte Richtung zu bewegen. \vec{V}_d ist der Distanzvektor und gibt die Distanz zwischen dem aktuellen Boid und seinem Nachbarn an. Zuerst muss bestimmt werden, ob sich innerhalb der maximalen Reichweite eines Boid

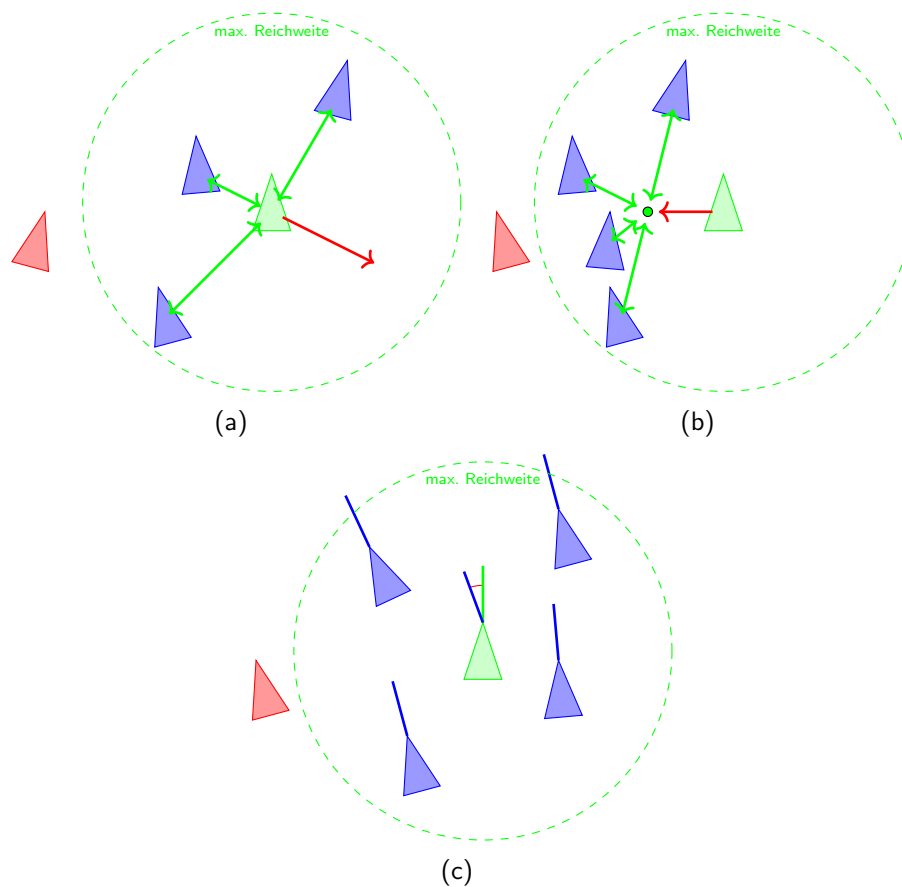


Abbildung 4.10.: Verhalten eines einzelnen Boid (Fisches) nach Craig-Reynolds. Separation a), Zusammenhalt b) und gleichartigen Ausrichtung c).

andere Boids befinden (siehe Abbildung 4.11 b). Wenn dies zutrifft, wird verglichen, ob der Winkel α kleiner ist als 150° . Sollte beides zutreffen, wird der Boid der Liste der Nachbarn hinzugefügt. Dadurch ergibt sich hinter dem Boid ein toter Bereich, denn er nicht sehen kann. Dies stellt das Sichtfeld eines realen Fisches dar, das 160° bis 180° horizontal und 150° vertikal beträgt [HB09] S.10. Die Größe der einzelnen Boids wird über eine Gausssverteilung variiert.

Für die Kollisionsbehandlung der Boids wurde ein anderer Ansatz als bei Reynolds gewählt. Bei Reynolds werden Kollisionen individuell betrachtet, indem innerhalb eines länglichen Zylinders, der um den Boid gestülpt ist, geprüft wird, ob Hindernisse im Weg sind. Dies beeinträchtigt die Echtzeitfähigkeit bei größeren Schwärmen sehr stark. Deshalb werden die Schwarmmitglieder nicht individuell betrachtet, sondern als ganzer Schwarm. Der Schwarm besitzt zwei Sphären (siehe Abbildung 4.12). Die innere ist für die direkte Kollisionsbehandlung zuständig, während die äußere das Sichtfeld darstellt. Die äußere Sphäre hat zusätzlich im hinteren Bereich eine Lücke von 60° . Bei einer Kollision, z.B. durch ein Terrain, wird der Kollisionspunkt über die Physik-Engine erfragt. Der Kollisionspunkt wird

von allen Positionen der Boids subtrahiert und normalisiert und auf den Lenkungsvektor \vec{V}_i addiert. Dadurch werden die Boids zusätzlich zu ihrer eigenen Ablenkung innerhalb des Schwarms von der Kollision abgelenkt. Sollte der Schwarm auf ein kleines Hindernis, wie z.B. ein neutrales AUV treffen, so teilt er sich in zwei Schwärme. Diese zwei Schwärme verschmelzen, nach dem eine fest eingestellte Zeit abgelaufen ist, wieder automatisch zusammen.

Zusätzlich wurden weitere Verhalten modelliert, um den Schwarm natürlicher erscheinen zu lassen. Zum einen Nahrungssuche und zum anderen ein Such- und Fluchtverhalten. Als Basis der Nahrungssuche dient eine Futterkarte, in der Futterquellen vom Benutzer eingetragen werden können. Die Futterquellen haben eine feste Kapazität, die durch die Schwärme verringert wird. Um den Geruchssinn der Fische zu simulieren, wird die am nächsten stehende Futterquelle als Zielquelle gesetzt. Die einzelnen Boids erhalten einen zusätzlichen Distanzvektor beim Zusammenhalt, der den Mittelpunkt des Schwarms in Richtung Futterquelle verschiebt. Dieser Distanzvektor wird zusätzlich normiert, damit sein Einfluss bei kleiner werdendem Abstand zur Futterquelle größer wird. Ein zusätzliches Verrauschen der Werte resultiert in einer weniger geraden Bewegung auf die Futterquelle zu. Sollte der Schwarm gesättigt sein, stoppt er die Nahrungsaufnahme.

Das Such- und Fluchtverhalten baut auf der Nahrungsaufnahme auf. Ein einzelner Fisch wird als Schwarm angesehen und als Angreifer definiert. Als Futterkarte werden ihm die anderen Fischeschwärme mitgeteilt. Sollte der Schwarm den Angreifer über seine äußere Sphäre sehen, so wird er ihm ausweichen und seine Geschwindigkeit erhöhen. Der Angreifer hat die Chance im toten Sichtbereich des Schwarms anzugreifen. Ist der Angriff erfolgreich, so werden Fische aus dem Schwarm entfernt. Sollte der Angreifer satt sein oder einen Schwarm zeitlich erfolglos verfolgt haben, sucht er sich einen neuen Schwarm. Interaktion zwischen AUVs und Fischeschwärmen basiert auf einem Bedrohungs-Grenzwert (siehe Anhang A.4). Der Schwarm weicht den AUVs aus, je größer der Bedrohungs Wert ist. Somit ist sichergestellt, dass die AUVs keine neutralen Beobachter sind, da in der Realität Licht, Aussehen des AUV und Lautstärke einen Einfluss auf den Schwarm haben. Der Schwarm wird allerdings nicht fliehen wie bei einem Angreifer oder sich teilen.

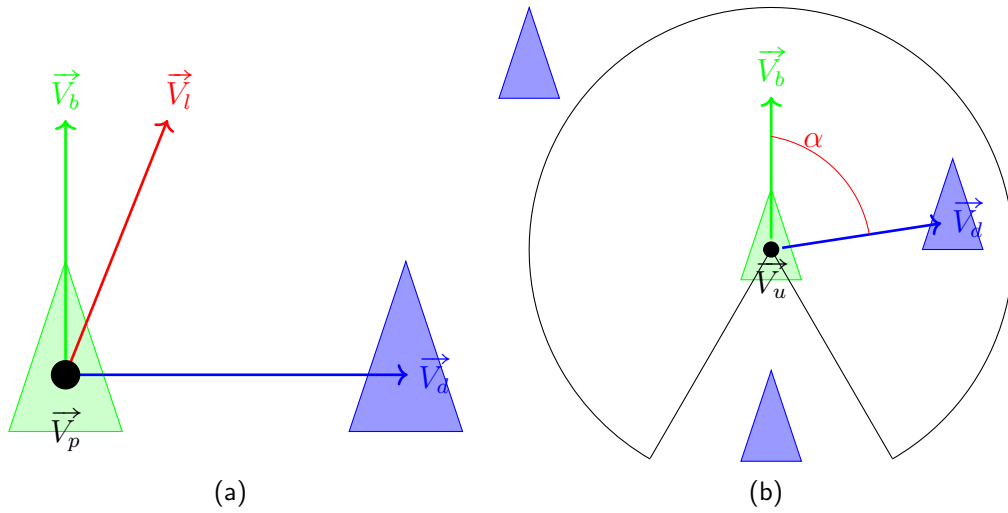


Abbildung 4.11.: (a) Lokales Koordinatensysteme eines Boids. (b) Sichtfeld eines Boid.

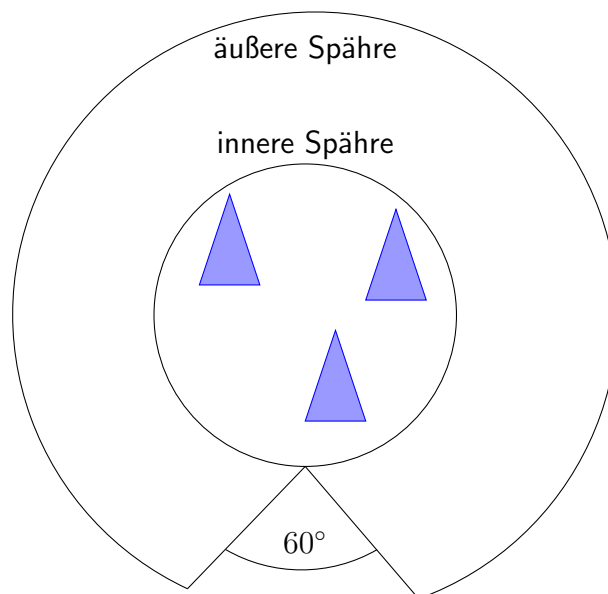


Abbildung 4.12.: Kollisions- und Sichtsphäre des Schwarms.

4.5.3. Strömung und Wellen

Strömungen und Wellen sind ein wichtiger physikalischer Einfluss für das AUV-Modell, und die Untersuchung von Strömungssystemen wird immer wichtiger. Die Fließgeschwindigkeit von Flüssen kann zwischen 0,1 und 6 m/s betragen. Dies kann den Pfad, den das AUV plant, verzerren, weil es gegen oder mit der Strömung fährt. Des Weiteren muss das AUV seinen Thruster stärker beanspruchen, um gegen eine Strömung anzukommen. Diese Gründe führen zu einer längeren Missionsdauer und einem erhöhtem Energieverbrauch. In geschlossenen Gewässern wie Seen und Teichen ist die Fließgeschwindigkeit geringer. Strömungen entstehen in diesem Fall durch Dichteunterschiede des Wassers und atmosphärische Kräfte [Man10] S.519. Des weiteren entstehen im Ozean und Seen Wasserwirbel. Diese Wasserwirbel sind für die Biologen von großem Interesse, da sie zu einem Nährstoffaustausch führen und in Zusammenhang mit Planktonblüten stehen [MAB⁺07]. Auch in kleinen Gewässern wie Seen entstehen Wirbel und sorgen für einen Nährstoffaustausch [KEG08]. Diese Wasserwirbel sind räumlich sehr klein und zeitlich von kurzer Dauer, im Vergleich zu den sonstigen Beobachtungen der Wissenschaftler. Hinzu kommt, dass in Seen diese Skalen nochmals kleiner sind. Die Untersuchung von Strömungs- und Wirbelsystemen durch autonome Systeme ist folglich wichtig für die Biologen und es ist daher notwendig, ein entsprechendes Strömungs- und Wellenmodell zu entwickeln, um AUV-Schwärme und deren Verhalten in diesem Kontext besser zu testen.

Wellen und Wasseroberflächen werden auf verschiedene Weisen modelliert und dem Nutzer ist überlassen, welches Modell er wählt. Im Allgemeinen wird die Wasseroberfläche als planare statische Fläche angesehen. Da das Wellental die Physik in tieferen Bereiche des Wassers wenig stark beeinflusst [DD91] und AUVs die meiste Zeit unter Wasser agieren, ist dieses Modell ausreichend und performant. USV (engl. für Unmanned Surface Vehicles) sind von Wellen stärker beeinflusst. AUVs sind in bestimmten Szenarien ebenfalls Wellen ausgesetzt wie im Load-Balancing Verhalten in Abschnitt 6.3.2 beschrieben. Wellen zu simulieren und sich physikalisch auf Objekte auswirken zu lassen, ist eine komplexe Aufgabe. Deshalb wird auf eine fertige Lösung innerhalb der jMonkeyEngine zurückgegriffen, die über ein projiziertes Wellengitter funktioniert. Darüber lässt sich die Höhe der generierten Wellen zu jeder Position erfragen. Kombiniert mit der Volumenberechnung aus dem AUV-Modell im Abschnitt 4.1 lässt sich dadurch die Änderung des Auftriebspunktes in Echtzeit errechnen und dadurch die Physik des AUV Modells an der Wasseroberfläche von USV und AUVs verbessern.

Um Strömungen zu simulieren, wird ein Vektorfeldansatz auf Basis von zweidimensionalen Karten gewählt. Dies erlaubt es, statische zweidimensionale Strömungen in der Ebene zu simulieren. Dieses vorberechnete Vektorfeld kann in die Simulation geladen, skaliert und neu-positioniert werden, damit es sich mit dem Terrain überdeckt. Das Vektorfeld besteht aus zwei einzelnen Bildern für die X- und Y-Richtung der Strömung. Der Farbwert des Pixels gibt den eigentlichen Wert der X- und Y-Komponente des Vektors an. Die Position des AUV wird periodisch mit dem Vektorfeld überprüft und der entsprechende Vektor, der zur Position passt, geladen und als Strömungskraft auf das AUV angewendet. Die Richtung des Vektors ist die Richtung, in der die Kraft wirkt und die Länge des Vektors die Stärke

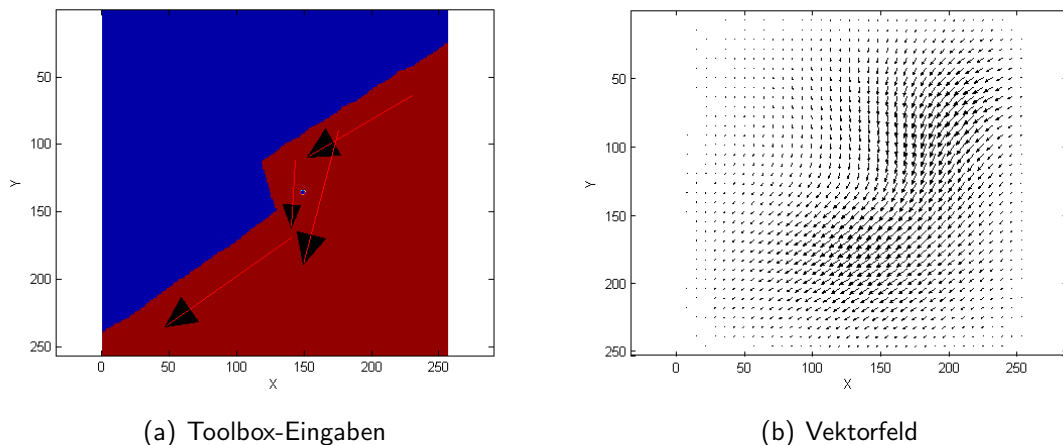


Abbildung 4.13.: Eingabe des Matlab-Scripts und Resultat. a) zeigt die Mauseingaben um das Vektorfeld zu berechnen. b) zeigt das resultierende Vektorfeld. Achsenangaben sind in Meter [TM14].

der Kraft. Um das Vektorfeld zu berechnen, wird ein externes Matlab-Script verwendet unter Zuhilfenahme des *griddata* und *seed* Befehls. Ein Beispiel ist zu sehen in Abbildung 4.13. In a) gibt der Nutzer im Matlab Script die generelle Richtung der Strömung durch vier Pfeile an. b) zeigt das errechnete Vektorfeld mit einer stärkeren Strömung, um das Hindernis herum. Abbildung 4.14 zeigt den Effekt der Strömungskarte auf ein AUV in der Simulationsumgebung. Der grüne Pfad ist der vom AUV (roter Kreis) beschrittene Pfad. Das AUV schwimmt hierbei an der Wasseroberfläche ohne aktiven Antrieb. Es ist deutlich zu sehen, wie die Strömung das AUV beeinflusst und um das Hindernis (blauer Kreis), einen Poller, herum führt.

4.6. Schwarm

Schwärme sind eine wichtige Anwendung für AUVs. Ein Schwarm von AUVs kann eine Aufgabe schneller lösen und ist tolerant gegenüber Ausfällen. Damit Schwärme sich organisieren können, müssen sie sich über Sensorik gegenseitig erkennen können. Dies kann z.B. optisch über Kameras geschehen. Um komplexere Verhalten zu ermöglichen, erfordert es eine Form von Kommunikation zwischen den Schwarmmitgliedern und nach außen. Eine Möglichkeit ist die akustische Unterwasserkommunikation, die realitätsnah modelliert werden muss (siehe Unterabschnitt 4.6.1). Ein weiterer Aspekt ist der des Energieverbrauchs. Damit der Schwarm seine Missionsdauer maximal ausnutzt, bedarf es eines Modells um Energie zu verbrauchen und zu erzeugen. Dadurch wird es möglich, die Missionen hinsichtlich des Energiehaushalts zu evaluieren. Unterabschnitt 4.6.2 behandelt das zu Grunde liegende Modell.

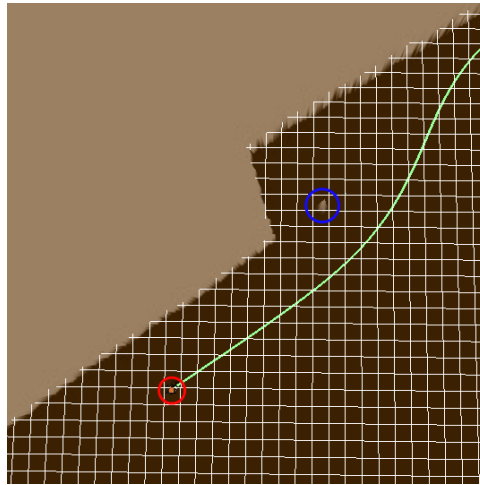


Abbildung 4.14.: Sicht auf die Simulationsumgebung und den Effekt der Strömung. Der kleine Punkt im roten Kreis ist das AUV. Der blaue Kreis ist ein Hindernis in Form eines Pollers, das Aufgrund der Strömung vom AUV umgangen wird. Die grüne Linie ist der genommene Pfad. Weiße Linien bilden ein Gitternetz zur besseren Einordnung [TM14].

4.6.1. Kommunikation

Damit AUV-Schwärme effizient zusammenarbeiten können, ist Kommunikation notwendig. Diese stellt einen wichtigen Aspekt dar und folglich muss das Modell Kommunikation hinreichend realistisch abbilden. Die Implementierung des Kommunikationsmodell entstand im Rahmen einer Bachelorarbeit [Sch15]. Abbildung 4.15 skizziert das Gesamtmodell der Kommunikation. Als Basis des Modells stehen einzelne Nachrichten, die zwischen den Teilnehmern gesendet und beim Eintreffen bestimmter Bedingungen gestört werden. Die Störung der Nachricht besteht aus den drei Komponenten Rauschen, Mehrwegausbreitung und Abschwächung wie im Grundlagenkapitel 3.3 beschrieben.

Der Nutzer oder die AUV-Steuerungssoftware versendet eine Nachricht von einem AUV zu einem anderen. Da die Bandbreite limitiert ist, wird die Nachricht aufgeteilt und die resultierenden Nachrichtenpakete einzeln nacheinander verschickt. Jedes Nachrichtenpaket enthält Informationen über die verwendete Frequenz, auf der es ausgesendet wurde und die aufgebrachte Sendeleistung des Signals. Diese sind abhängig von den Parametern, die der Nutzer im Modem eingestellt hat, ebenso wie die Bandbreite. Zwischen den AUVs existieren virtuelle Kommunikationskanäle, auf denen die Nachrichtenpakete versendet werden. Die Nachrichten werden beim erfolgreichen Eintreffen abgeschwächt. Zusätzlich werden die Nachrichten verrauscht und durch Mehrwegausbreitung entstehen Überlagerungen. In der Abbildung 4.15 wird das Nachrichtenpaket m von AUV_1 zu AUV_2 gesendet. Alle in Reichweite befindlichen AUVs bekommen das Paket ebenfalls, in diesem Fall das AUV_3 , aber nicht das AUV_4 , das sich außerhalb der maximalen Reichweite des Modems von AUV_1 befindet. Das AUV_3 ist dabei sämtlichen Effekten ausgesetzt wie das AUV_2 . Dies

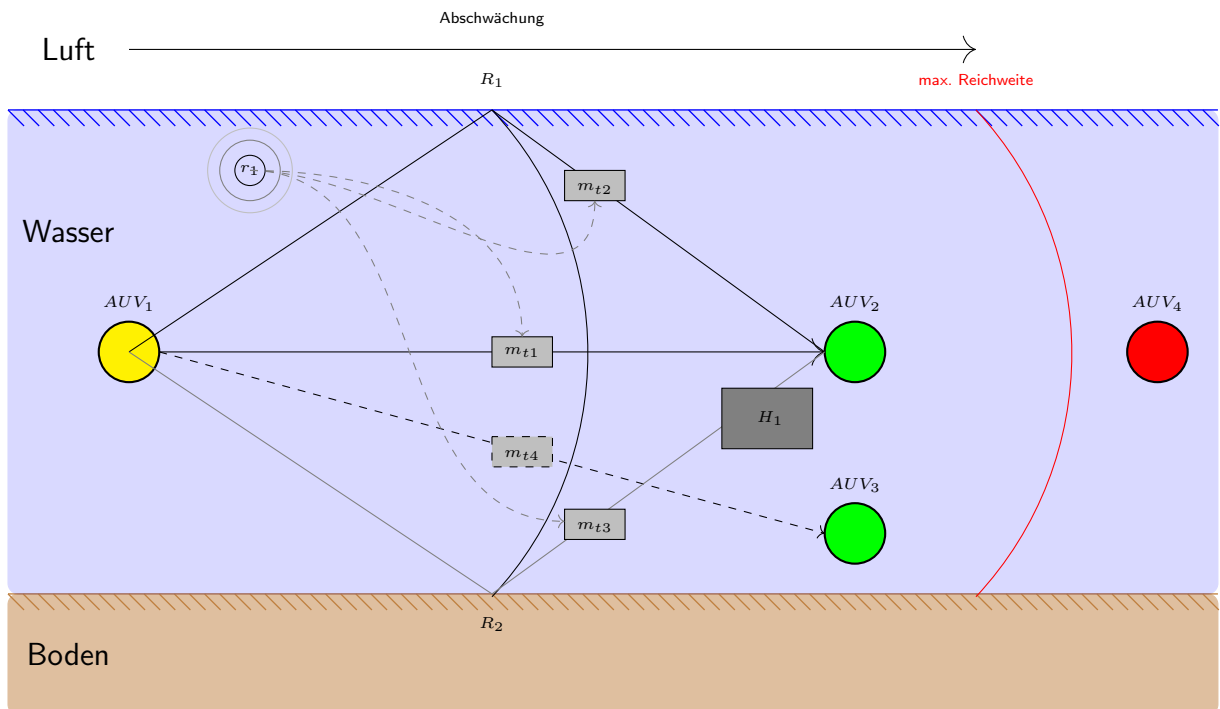


Abbildung 4.15.: Gesamtmodell der Unterwasserkommunikation.

sind hauptsächlich das Rauschen und die Mehrwegausbreitung. Sollte das Modem auf einer anderen Frequenz arbeiten, würde es das Paket verwerfen, es sei denn, die Frequenz würde sich durch andere Effekte ausreichend verändern. Das Kommunikationsmodell ist nicht auf die Kommunikation eines einzelnen AUVs mit Empfängern beschränkt. Es können "gleichzeitig" mehrere AUVs miteinander kommunizieren.

Ist das Nachrichtenpaket erfolgreich bei einem AUV eingetroffen, wird das Signal abgeschwächt. Um die tatsächliche Signalstärke S^* an einem Punkt zu berechnen, muss von der ursprünglichen Signalstärke S der *Transmission-Loss* abgezogen werden (siehe Formel 4.9). Sämtliche Angaben in dB.

$$S^* = 10 \log \left(\left(\frac{S}{10} \right)^{10} - \left(\frac{TL}{10} \right)^{10} \right) \quad (4.9)$$

Der *Transmission-Loss* TL in Formel 4.10 gibt den tatsächlichen Verlust in dB an. d steht für die Distanz, α ist der Absorptionskoeffizient und τ ist ein Koeffizient, der die Ausbreitung angibt. Diese einfache Formel ist ausreichend, um den Verlust zu errechnen, wobei Erweiterungen des Modells notwendig sind bei Brechungen und Mehrwegausbreitung [Lur10] S.27.

$$TL = \tau \log(d) + \frac{d}{1000} \alpha \quad (4.10)$$

α ist der Absorptions-Koeffizient. Es werden insgesamt zwei Absorptions-Koeffizienten eingesetzt und der Nutzer kann entscheiden, welchen er von beiden verwendet. Zum einem

Francois-Garrison (siehe 4.11) und zum anderen Thorp (siehe 4.13). Francois-Garrison [FRE82a][FRE82b] besteht aus drei einzelnen Komponenten. Die erste behandelt die Abschwächung des Signals durch Borsäure, die zweite die durch das Magnesiumsulfat und die letzte durch die Viskosität des Wassers. Die einzelnen Parameter sind den Gleichungen 4.12 zu entnehmen. Die Parameter A_1 , P_1 , f_1 und c sind für die Abschwächung durch die Borsäure verantwortlich. Für die Abschwächung durch das Magnesiumsulfat sind A_2 , P_2 und f_2 zuständig. Die Abschwächung durch die Viskosität des Wassers wird durch die Parameter A_3 und P_3 beschrieben. α ist der eigentliche Absorptionskoeffizient in dB/km . Hierbei steht T für die Temperatur in Grad Celsius, f für die Frequenz in kHz, S ist der Salzgehalt in Practical Salinity Units (p.s.u.) und z die Tiefe in Metern.

$$\alpha = A_1 P_1 \frac{f_1 f^2}{f_1^2 + f^2} + A_2 P_2 \frac{f_2 f^2}{f_2^2 + f^2} + A_3 P_3 f^2 \quad (4.11)$$

$$A_1 = \frac{8.86}{c} 10^{0.78pH-5}$$

$$P_1 = 1$$

$$f_1 = 2.8 \sqrt{\frac{S}{35}} 10^{(4-1245/(T+273))}$$

$$c = 1412 + 3.21T + 1.19S + 0.0167z$$

$$A_2 = 21.44 \frac{S}{c} (1 + 0.025T)$$

$$P_2 = 1 - 1.37 \times 10^{-4} z + 6.2 \times 10^{-9} z^2$$

$$f_2 = \frac{8.17 \times 10^{8-1990/(T+273)}}{1 + 0.0018(S - 35)}$$

$$T < 20^\circ C : A_3 = 4.937 \times 10^{-4} - 2.59 \times 10^{-5} T + 9.11 \times 10^{-7} T^2 - 1.5 \times 10^{-5} T^3$$

$$T > 20^\circ C : A_3 = 3.968 \times 10^{-4} - 1.146 \times 10^{-5} T + 1.45 \times 10^{-7} T^2 - 6.5 \times 10^{-10} T^3$$

$$P_3 = 1 - 3.83 \times 10^{-5} z + 4.9 \times 10^{-10} z^2$$

(4.12)

Das hauptsächlich verwendete Modell in der Literatur stellt Francois-Garrison dar. Es existieren viele experimentelle Nachweise und theoretische Studien darüber und es ist akkurat und recht vollständig (siehe [Lur10] S.24).

Das Modell von Francois-Garrison ist komplex und benötigt viele Parameter die nicht immer zur Verfügung stehen. Des Weiteren benötigt die Berechnung mehr Zeit als bei Thorp. Deshalb wird ein zweites Modell nach Thorp [Tho67] angeboten (siehe Gleichung 4.13). Es stellt eine Annäherung an Francois-Garrison dar, dass bis in den Bereich von 100 kHz ein hinreichend genaues Ergebnis liefert. Der einzige Parameter ist f in Kilohertz. Die Formel ist deutlich kompakter und lässt sich schneller berechnen.

$$\alpha = \left(\frac{0.11}{1 + f^2} + \frac{44}{4100 + f^2} \right) f^2 \quad (4.13)$$

	sphärisch	zylindrisch	vereinfacht
τ	2	1	1.5

Tabelle 4.1.: Ausbreitungskoeffizient τ nach [Tho67].

Name	Quelle	Wasser	Parameter
BilaniukWong	[BW93]	Süßwasser	Temperatur
Marczak	[Mar97]	Süßwasser	Temperatur
LubbersGraaff	[LG98]	Süßwasser	beschränkte Temperatur
BelogolskiiSekoyan	[BSSL99]	Süßwasser	Temperatur und Druck
Mackenzie	[Mac81]	Salzwasser	Temperatur, Tiefe und Salzgehalt
Coppens	[Cop81]	Salzwasser	Temperatur, Tiefe und Salzgehalt
ChenMillero	[CM77]	Salzwasser	Temperatur, Druck und Salzgehalt
Del Grosso	[DG74]	Salzwasser	Temperatur, Druck und Salzgehalt

Tabelle 4.2.: Verschiedene Ausbreitungsgeschwindigkeiten von Schall im Wasser.

τ ist der Ausbreitungskoeffizient. Entsprechende Werte können der Tabelle 4.1 entnommen werden. Abbildung 4.15 zeigt eine sphärische Ausbreitung von AUV_1 aus. Bei den Punkten R_1 und R_2 trifft die Schallwelle auf die Wasseroberfläche und den Boden. Die Schallwelle geht in eine zylindrische Ausbreitung über. Während im Ozean allgemein von einer sphärischen Ausbreitung ausgegangen werden kann, weil der Boden und die Wasseroberfläche weit entfernt sind, ist in flachen Gewässern wie Flüssen eine zylindrische Ausbreitung dominierend. Da die Ausbreitung in flachen Gewässern nicht sofort zylindrisch ist, wird ein Mittelwert verwendet, der zwischen zylindrisch und sphärisch liegt.

Die Nachrichtenpakete erreichen die Empfänger nicht sofort, sondern benötigen aufgrund der langsamen Ausbreitungsgeschwindigkeit des Schalls eine gewisse Zeit. Deshalb erlaubt das Modell die Auswahl verschiedener Geschwindigkeitsfunktionen, die in Tabelle 4.2 dargestellt sind. Dem Nutzer ist überlassen, eine entsprechende Funktion auszuwählen. Aufgrund der Länge der Formeln sei auf die einzelnen Referenzen verwiesen: [watb] [wata]. Die Formeln berücksichtigen Temperatur im Süßwasser und zusätzlich den Salzgehalt im Salzwasser.

Abschließend wird das Nachrichtenpaket auf Bit-Ebene verrauscht. In der Abbildung wirkt sich das Rauschen r_1 auf alle Nachrichtenpakete aus. Um über Schwellwerte zu bestimmen, ob ein Bit gedreht wird, muss das Signal-Rausch-Verhältnis (engl. signal-to-noise ratio SNR) 4.14 bestimmt werden. Das SNR ist das Verhältnis zwischen dem beim Empfänger eingetroffenen gedämpften Signal S^* und dem Hintergrundrauschen $N(f)$.

Um das Hintergrundrauschen zu bestimmen, werden nach Urick [Uri96] verschiedene Rauschkomponenten (siehe Gleichung 4.15) betrachtet. Dazu gehören Turbulenzen N_t , Schifffahrt N_s , wind-induzierte Wellen an der Oberfläche N_w und das thermische Rauschen N_{th} . Die Gleichungen 4.16 beschreiben die einzelnen Funktionen genauer. Alle sind von der Frequenz f abhängig. Zusätzlich kommt bei $N_s(f)$ der *shipping factor* s hinzu,

der zwischen 0 und 1 liegt und angibt wie stark präsent die Schifffahrt ist. Bei $N_w(f)$ kommt die Windgeschwindigkeit w in Metern pro Sekunde hinzu.

$$SNR = 10 \log \left(\frac{\left(\frac{S^*}{10}\right)^{10}}{\left(\frac{N(f)}{10}\right)^{10}} \right) \quad (4.14)$$

$$N(f) = N_t(f) + N_s(f) + N_w(f) + N_{th}(f) \quad (4.15)$$

$$\begin{aligned} N_t(f) &= 17 - 30 \log(f) \\ N_s(f) &= 40 + 20(s - 0.5) + 26 \log(f) - 60 \log(f + 0.03) \\ N_w(f) &= 50 + 7.5w^{1/2} + 20 \log(f) - 40 \log(f + 0.04) \\ N_{th}(f) &= -15 + 20 \log(f) \end{aligned} \quad (4.16)$$

Der letzte Bestandteil des gesamten Kommunikationsmodells ist die Mehrwegausbreitung. Diese basiert auf einem Strahlenmodell der Schallwelle. Nachrichtenpakete werden direkt zwischen Sender und Empfänger versendet (zwischen AUV_1 und AUV_2) und über Umwege, die aufgrund der Reflexion der Schallwelle an der Wasseroberfläche (Punkt R_1) und dem Boden (Punkt R_2) entstehen. Der Boden wird, was Reflexionseigenschaften angeht, als ähnlich zur Wasseroberfläche angenommen. Nachdem die Umwege berechnet wurden, wird überprüft, ob sich bei den Kanälen zwischen dem Sender und dem Empfänger ein hinreichend großes Hindernis H_1 (z.B. Terrain) befindet. Sollte dies der Fall sein, kann das Nachrichtenpakete den Kanal nicht nutzen und wird verworfen, in diesem Fall das Nachrichtenpaket m_{t3} . Die Nachrichtenpaket m_{t1} , m_{t2} und m_{t3} kommen zeitlich versetzt beim Empfänger-AUV AUV_2 an, wobei m_{t1} als Erstes eintrifft, aufgrund des kürzesten Weges. In Abhängigkeit der Ausbreitungsgeschwindigkeit und des gewählten Zeitfensters, in denen Nachrichtenpakete als gleichzeitig angesehen werden, treffen die restlichen Nachrichtenpakete ein und werden mit m_{t1} überlagert oder nicht. Da mehrere AUVs gleichzeitig senden können, können dadurch ebenfalls Überlagerungen zustande kommen, ähnlich zur Mehrwegausbreitung.

4.6.2. Energie

Energie ist ein wichtiger Aspekt für einzelne AUVs und innerhalb eines Schwarms. Sensoren und Aktoren verbrauchen Strom und reduzieren die Missionsdauer. Dies wirkt sich bei ineffizienten Verhalten deutlicher aus. Durch Energiegewinnungsmethoden lässt sich indes die Missionszeit erhöhen. Gleiter gelten aufgrund ihres auf Ballasttanks und dynamischen Auftriebs basierenden Antriebs als Energie-effizient (siehe Abschnitt 3.1). Der Einsatz von thermischer Energiegewinnung über Temperaturdifferenzen [WSJ01] und Solarzellen [AOY11] erhöht die Missionszeit von Gleitern. Im Bereich der AUVs gibt es ebenfalls Entwicklungen, die mit Solarzellen ausgestattet sind und sich während des Tages aufladen, um in der Nacht ihre Mission auszuführen [JBD⁺03] [BMC⁺05]. Bei einem Schwarm wird durch Kommunikation zusätzlich mehr Energie verbraucht und durch die Schwarmverhalten ergeben sich neue Einsparungspotentiale. Abschnitt 6.3.2 beschreibt

ein Load-Balancing Verhalten für einen Schwarm, in dem zwischen den Mitgliedern Kapazitätsunterschiede entstehen. Durch ein entsprechendes Energieverbrauchsmodell lässt sich das Verhalten validieren und evaluieren.

Abbildung 4.16 illustriert das Modell für den Energiehaushalt. Im Modell sind beliebig viele Akkumulatoren A_1 bis A_n erlaubt die einem AUV zugeordnet sind. Jeder Akkumulator hat seine eigene Kapazität, angegeben in mA/h. Sensoren und Aktoren, wie z.B. der Thruster, verbrauchen Energie eines frei zugeordneten Akkumulators. Sollte die Kapazität Null erreicht haben, so stellt der Aktor oder Sensor seinen Dienst ein. Sensoren verwenden einen einfachen aus den Datenblättern entnommenen statischen Verbrauchswert, um dem Akkumulator Strom zu entziehen. Aktoren, wie z.B. Thruster, verwenden Messkurven in denen der Stromverbrauch in Abhängigkeit zur Drehzahl steht (siehe Unterabschnitt 4.3.1).

Der Akkumulator kann aufgeladen werden durch Energieerzeuger, wie z.B. Solarpaneele. Das Solarpanel erzeugt in Abhängigkeit der Sonne Strom, der dem Akkumulator wieder zugeführt wird. Die jMonkeyEngine bietet ein Tag-Nacht-Zyklussystem, aus dem man den Vektor \vec{V}_s erhält, der die Richtung zur Sonne angibt. Mit Hilfe des Normalenvektors \vec{V}_n des Solarpanels erhält man den Winkel α . Das Solarpanel wird immer als planar ausgerichtet gesehen. Die Gleichung 4.17 liefert abschließend die Formel für die Energieausbeute des Solarpanels. Die erzeugte Energie E errechnet sich aus Winkel α vom Sinus multipliziert mit der erzeugten Energie E_0 , wenn das Solarpanel senkrecht von der Sonne angeschiessen wird und mit voller Leistung arbeitet. Sollte die Sonne unterhalb des Horizonts verschwinden oder sich das Solarpanel unter Wasser befinden, so wird kein Strom erzeugt.

$$E = E_0 * \sin(\alpha) \tag{4.17}$$

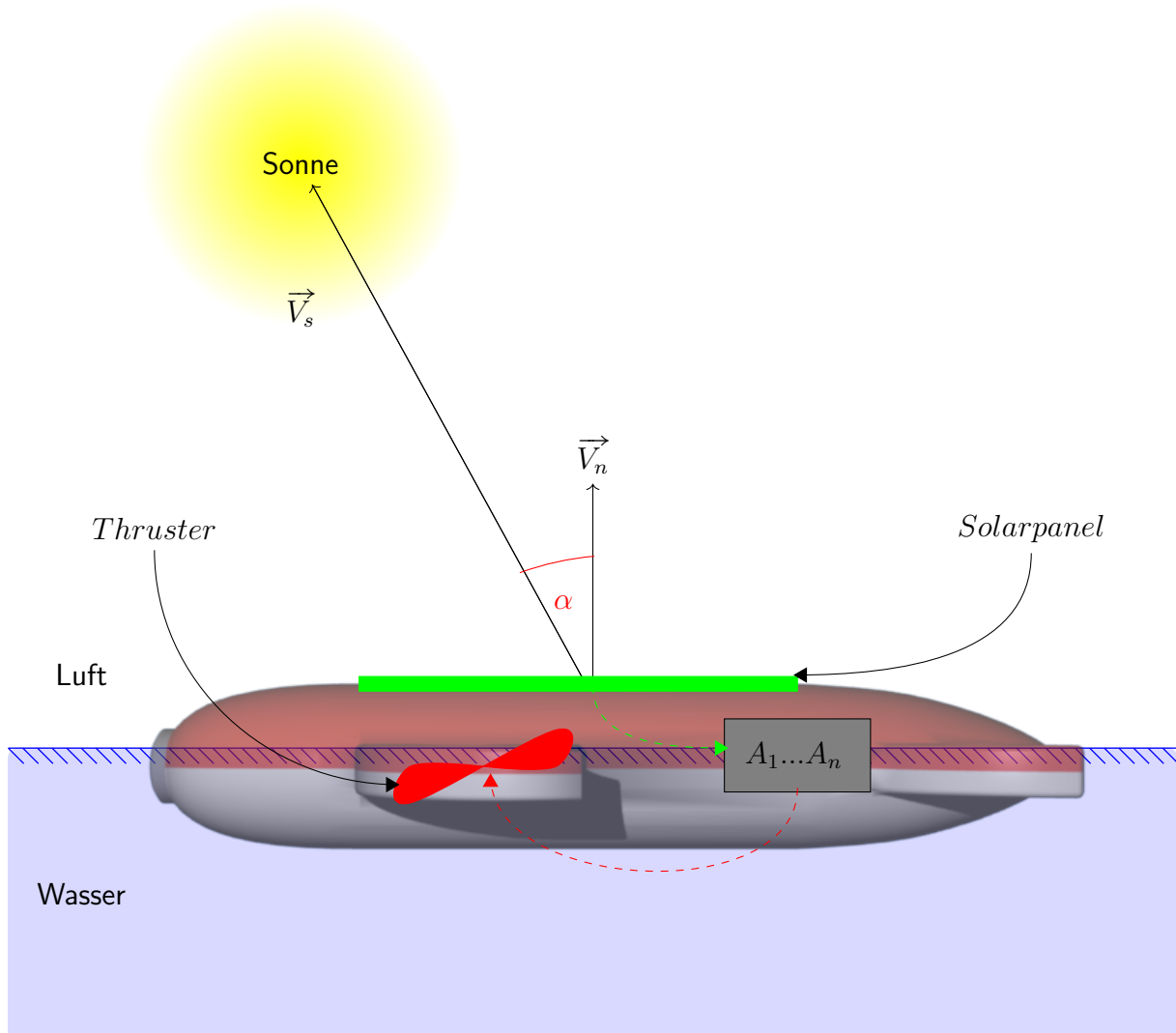


Abbildung 4.16.: Energiemodell für Stromverbrauch und -erzeugung. Das Solarpanel erzeugt in Abhängigkeit zum Einstrahlwinkel der Sonne Energie, die bis zu n Akkumulatoren zugeführt werden kann. Aktoren wie Thruster verbrauchen diese wieder.

Kapitel 5.

Prototypische Simulationsumgebung MARS

Das Ganze ist mehr als die Summe seiner Teile.

(Aristoteles)

Inhaltsangabe

5.1. Architektur	77
5.2. Implementierung	81
5.2.1. Verwendete Technologien	81
5.2.2. Szenegraph	83
5.2.3. XML-Konfiguration	84
5.2.4. Benutzungsschnittstelle	87

Dieses Kapitel beschäftigt sich mit der beispielhaften Umsetzung des Modells aus Kapitel 4 in Form der prototypischen Simulationsumgebung MARS. Abschnitt 5.1 erläutert die allgemeine Architektur von MARS. Abschnitt 5.2 beschäftigt sich mit konkreten Implementierungen, wie z.B. welche Technologien verwendet wurden und mit der Umsetzung der Benutzungsschnittstelle in Abschnitt 5.2.4.

5.1. Architektur

MARS steht für **MA**rine **R**obotics **S**imulator und stellt die beispielhafte Umsetzung des Modells aus Kapitel 4 dar. MARS ist open-source und unter BSD-Lizenz frei verfügbar¹. Abbildung 5.1 zeigt die vereinfachte Architektur von MARS. Als Grundlage dient die JVM (Java Virtual Machine), da in MARS Java als Hauptprogrammiersprache ausgewählt wurde. Diese hat den Vorteil der Plattformunabhängigkeit und dass weitere wichtige Bibliotheken in Java verfügbar sind. Auf der JVM setzt MARS auf, bestehend aus vier Komponenten. *JME* ist die verwendete Grafik-Engine (siehe 5.2.1.1). Sie kümmert sich um die dreidimensionale Repräsentation des Geschehens. *Bullet* ist die verwendete Physik-Engine (siehe 5.2.1.2), die in JME bereits integriert ist. Bullet kümmert sich um

¹<https://github.com/iti-luebeck/MARS>

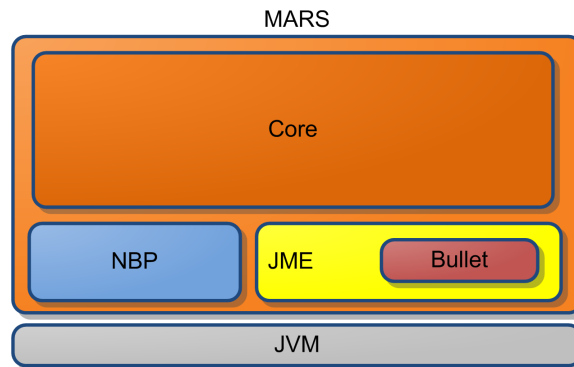


Abbildung 5.1.: Vereinfachte Architektur von MARS: MARS setzt auf der JVM auf. Die eigentliche Simulation (Core) verwendet NBP für die GUI, JME für die Grafik und Bullet für die Physik.

die Kollisionen und darum, dass die auf AUVs wirkenden Kräfte in eine entsprechende Bewegung umgesetzt werden. Die *NetBeans Platform* (NBP) bildet die Basis für die Benutzungsschnittstelle (siehe 5.2.1.3) und übernimmt das Fenstermanagement, Modularisierung und Datenmanipulation. Auf diesen drei Komponenten setzt der Core auf, die eigentliche Simulation. In dieser werden die Kräfte berechnet und die AUVs verwaltet. Aufgrund der zur Verfügung stehenden Rechenleistung von modernen CPUs und GPUs besteht MARS nicht aus einer verteilten Architektur, die über mehrere Server verteilt ist, sondern aus einer zentralen Architektur. Dies ist einfacher zu programmieren und in Betrieb zu halten als ein Multicomputer System. Allerdings verwendet das MARS Framework Threads, um die vorhandene Multiprozessorarchitektur (Multicore-CPU) auszunutzen. Die Anbindung von MARS an die Steuerungssoftware der AUVs erfolgt über das Ethernet-Netzwerk. Die heutige kostengünstige Netzwerktechnologie erlaubt Datenübertragungsraten im Gigabit-Bereich mit genug Kapazitäten für mehrere AUVs und ihre komplette Sensorik, bestehend aus Sonar- und Kameradaten. Die Hardware der meisten modernen AUV-Projekte ist zudem netzwerkfähig und netzwerkbasierte Robotik-Frameworks wie ROS[QCG⁺09], Player[GVH03], MOOS[New03] und YARP[MFN06] können sich netzwerkbasiert mit MARS verbinden.

Eine typische Nutzung von MARS ist in Abbildung 5.2 abgebildet. Auf einem einzelnen Computer ist MARS installiert. Über die TCP oder ROSJava Schnittstelle kann MARS über das Netzwerk mit anderen Teilnehmern kommunizieren. Auf der Gegenseite steht ein weiterer Computer, auf dem die Steuerungssoftware des AUVs läuft. Diese kann Aktordaten an den Simulator senden und Sensordaten empfangen. MARS berechnet anhand des hydrodynamischen Modells und der Aktordaten dann eine neue Position des AUVs innerhalb der Welt und sendet aktualisierte Sensordaten zurück. Alternativ kann sich direkt mit dem im AUV enthaltenen netzwerkfähigen eingebetteten System, wie z.B. einem Kleincomputer, verbunden werden. Dabei können rein virtuelle Sensoren und Aktoren (innerhalb von MARS) verwendet werden oder eine Mischung aus virtuellen oder realen. Wenn eine Kommunikationsverbindung unter Wasser besteht, ist auch ein hybrider Modus möglich, in

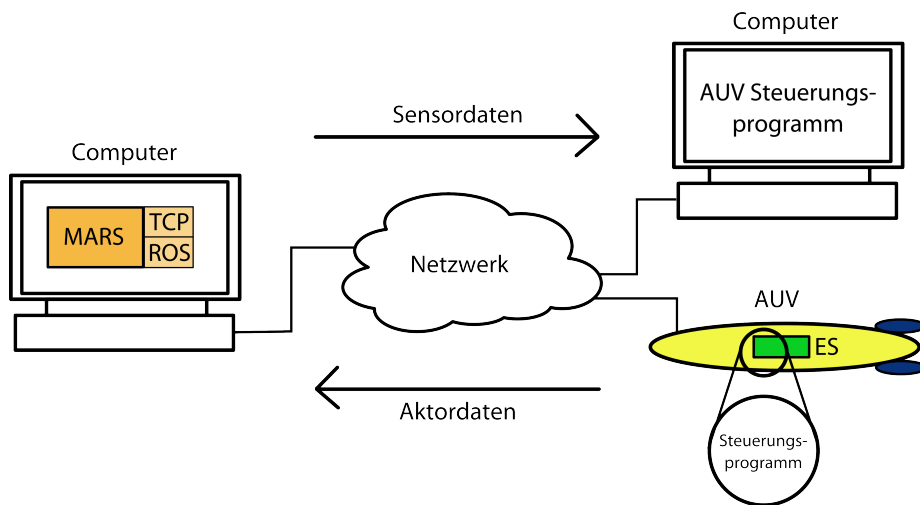


Abbildung 5.2.: Nutzungsszenarien in MARS: Auf einem Computer ist MARS installiert. Das AUV-Steuerungsprogramm, das auf einem anderem Computer oder eingebetteten System läuft, verbindet sich per Netzwerk mit der Simulation. Es erhält simulierte Sensordaten und sendet selbst Aktordaten aus. Mehrere Clients können sich mit der Simulation gleichzeitig verbinden.

dem einzelne Sensoren simuliert werden, aber die Hydrodynamik in der Realität stattfindet. Es können sich mehrere Steuerungsprogramme von verschiedenen AUVs oder Computern gleichzeitig mit der Simulation verbinden, solange sie netzwerkfähig sind und die entsprechenden Schnittstellen (TCP oder ROS) unterstützen.

Abbildung 5.3 beschreibt den feineren Aufbau von MARS. Den Einstiegspunkt bildet die Klasse *MARS_Main*, eine von JME geerbte und von MARS erweiterte *SimpleApplication*. Die *SimpleApplication* bietet eine Update-Schleife und ist ein Minimalbeispiel für eine verwendbare JME Anwendung. Innerhalb von *MARS_Main* werden die einzelnen AppStates durch den StateManager von JME verwaltet (siehe Abschnitt 5.2.1.1). In der folgenden Aufzählung werden die wichtigsten AppStates beschrieben:

1. SimState: Der SimState stellt den Kern von MARS dar und verwaltet und aktualisiert die AUVs und SimObjects. SimObjects sind statische Objekte innerhalb der Simulation, wie z.B. Pipelines.
2. MapState: Zeigt eine MiniMap an, eine kleine Karte mit der Position der AUVs.
3. Nifty: Verwendet die NiftyGUI² Komponente von JME, um Overlays in der Hauptansicht zu präsentieren. Dies wird z.B. verwendet um die aktuelle Akkukapazität darzustellen, wenn man mit der Maus ein AUV berührt.
4. GUI: Verarbeitet sämtliche Benutzereingaben im Hauptfenster. Ermöglicht das Verschieben und Rotieren von AUVs.

²<https://github.com/void256/nifty-gui>, Zugriff am 22.10.2015

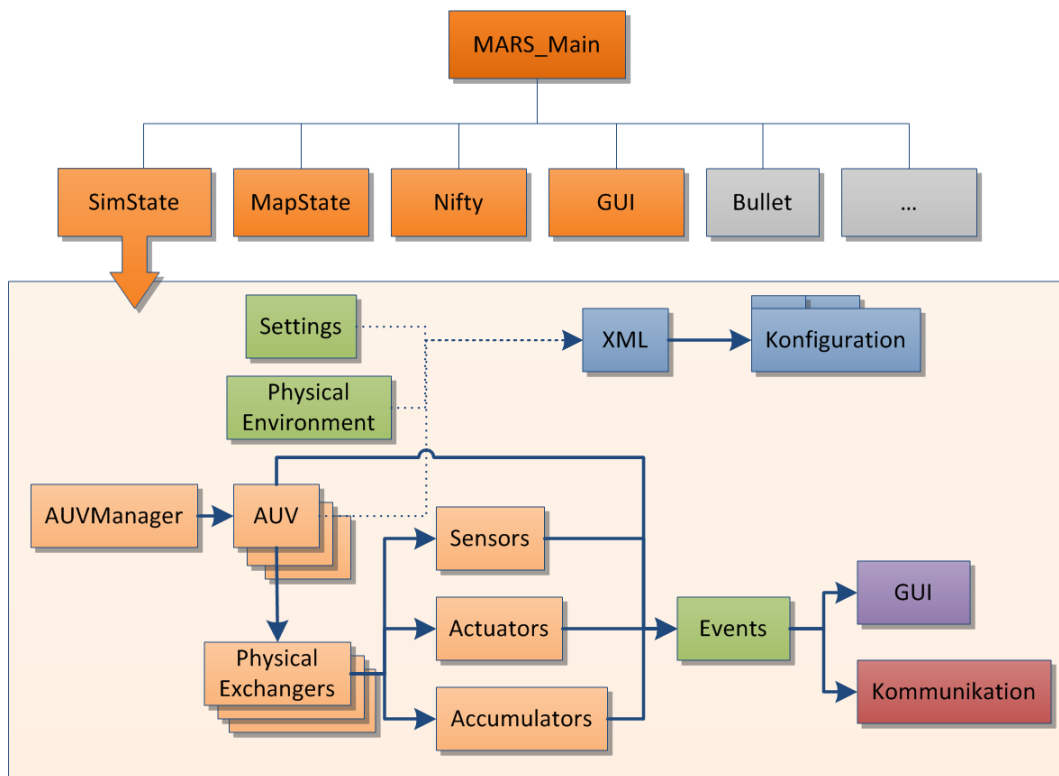


Abbildung 5.3.: Feinere Struktur von MARS und des SimState.

5. Bullet: Bereits von JME bereitgestellter State, der die Physikengine Bullet integriert.
6. weitere States: Jedes Plugin oder Module kann seinen eigenen States besitzen, dies ist z.B. der Fall für die Fischschwarmsimulation.

Im folgenden wird der *SimState* beschrieben, dem wichtigsten State innerhalb von MARS. Der *AUVManager* bildet das Hauptverwaltungselement für die AUVs (analog gibt es den *SimObjectManager* für *SimObjects*). In ihm werden AUVs registriert, initialisiert und aktualisiert. Die *AUV*-Klassen bilden AUVs ab. Innerhalb werden Parameter festgehalten, wie z.B. Masse oder Strömungskoeffizienten. Außerdem werden die Kräfte periodisch berechnet und in der Physikengine auf das AUV-Objekt angewendet.

AUVs beinhalten eine Reihe von *PhysicalExchangers*. Konkrete Ausprägungen der *PhysicalExchangers* bilden *Sensoren*, die Daten aus der Umwelt erfassen und weiterleiten. Dies können z.B. Sonare oder Kameras sein. *Aktoren* nehmen Daten entgegen, um auf die Welt einzuwirken. Dabei kann es sich um Thruster für den Vortrieb handeln oder um einen Unterwasser manipulator. *Akkumulatoren* stellen *Sensoren* und *Aktoren* Energie zu Verfügung, die sie verbrauchen. Wenn keine Energie mehr zur Verfügung steht, stellen diese ihren Dienst ein.

Sämtliche *AUVs* und *PhysicalExchangers* senden Daten eventbasiert. Dabei kann es sich um interne Debugdaten handeln, wie z.B. der aktuelle Auftrieb des AUVs, oder um Daten

der Sensoren, wie z.B. der Druck in mbar vom Drucksensor. Andere Komponenten können Listener registrieren, um entsprechende Events zu erhalten. Dies sind hauptsächlich GUI-Komponenten auf der NBP-Ebene (siehe Abschnitt 5.2.4).

Die AUVs und ihre *PhysicalExchangers* werden über den XML Parser serialisiert und in einer Konfiguration gesammelt (siehe Abschnitt 5.2.3). Dazu kommen die Sammelklassen *Settings* und *PhysicalEnvironment*. *Settings* sind Einstellungen innerhalb der Simulation, wie z.B. Grafik. In *PhysicalEnvironment* sind alle physikalischen Parameter der Welt enthalten, wie z.B. Wasserhöhe, Fluiddichte oder Oberflächentemperatur.

5.2. Implementierung

In diesem Abschnitt wird auf die konkrete Implementierung einzelner Komponenten von MARS eingegangen. Abschnitt 5.2.1 erläutert die eingesetzten Technologien. Dies beinhaltet JME für die Grafik, Bullet für die Physik, NBP für die GUI und ROS für die Kommunikation. Abschnitt 5.2.3 beschreibt den Aufbau von Konfigurationen, die verwendet werden, um AUVs zu beschreiben. Abschnitt 5.2.2 erläutert den Aufbau des Szenegraphen innerhalb von JME.

5.2.1. Verwendete Technologien

In diesem Unterkapitel werden die verwendeten Technologien beschrieben, die bei MARS zum Einsatz kommen. Für die dreidimensionale Darstellung wird die open-source Spieleengine jMonkey Engine 3 verwendet (5.2.1.1). Für die Physik wird die Bullet Physics Library eingesetzt, die in jMonkey bereits integriert ist (5.2.1.2). Für die grafische Benutzungsschnittstelle wird die NetBeans Plattform verwendet (5.2.1.3). Zuletzt wird für die Kommunikation mit den Beispiel-AUVs das Robot Operating System eingesetzt (5.2.1.4).

5.2.1.1. jMonkeyEngine 3

Die jMonkeyEngine 3 (jME3) ist eine shaderbasierte Szenegraph 3D-Engine, die in Java geschrieben ist [jme]. Sie ist open-source und steht unter der BSD Lizenz. Unterstützt werden die drei Betriebssysteme Windows, Linux und Mac. Außerdem existiert eine Version für Android und iOS. jME3 verwendet OpenGL als Renderer in Form von LWJGL (Lightweight Java Game Library). Als Physikengine kommt die Bullet Physics Library über einen Wrapper zum Einsatz (siehe Abschnitt 5.2.1.2). Sie ist voll in jME3 integriert, was ihren Einsatz vereinfacht.

jME3 bietet verschieden Features um die Entwicklung von dreidimensionalen Anwendungen, vornehmlich Videospielen, zu vereinfachen. Es gibt ein Terrainsystem, Materialsystem, um Texturen zu laden und darzustellen, Licht und Spezialeffekte, darunter Wassereffekte, Eingabeverarbeitung von Maus und Tastatur, Assetverwaltung von Modellen und Texturen und Level-of-Detail (LoD).

Der Szenegraph ist eine baumartige Datenstruktur bestehend aus *Nodes* und bildet die zentrale Verwaltungsstruktur von jME3. Eine Node ist ein abstrakter Container mit einem Vater- und mehreren Kinderknoten. Die oberste Node wird i.A. als *RootNode* bezeichnet. An die Node können weitere Nodes angebracht werden oder dreidimensionale Modelle. Durch den Einsatz des Szenegraph werden Transformationen vereinfacht. Außerdem können Optimierungen der Sichtbarkeit von Objekten einfacher durchgeführt werden (culling). Die Verarbeitung in jME3 folgt dem Update-Loop Prinzip. In einer Schleife werden nacheinander Nutzereingaben verarbeitet, der interne Zustand geändert (z.B. Rotation eines 3D-Modells) und die Szene gerendert.

Um den Code zu strukturieren, werden AppStates und Controls verwendet. Jeder AppState besitzt eine eigene RootNode und kann unabhängig von den anderen AppStates laufen. Controls können an Nodes angebunden werden und werden regelmäßig von der UpdateLoop aktualisiert. Sie beziehen sich immer auf die ihnen zugewiesene Node. Dabei kann es sich z.B. um die Translation einer Node in Abhängigkeit von Nutzereingaben handeln.

5.2.1.2. Bullet Physics Library

Die Bullet Physics Library (Bullet) [bul] ist eine quelloffene Physik-Engine, die unter ZLib-Lizenz steht [Cou13]. Bullet ist in C++ programmiert und läuft auf verschiedenen Plattformen darunter Linux, Windows, Android und Playstation 3. Bullet erlaubt es, Fest- und Weichkörper und dazugehörige Kollisionen zu simulieren.

Kollisionen können diskret oder kontinuierlich aufgelöst werden und Kollisionskörper können konvexe und konkave Formen annehmen. Es können entweder Primitivkörper, wie z.B. Kugeln, verwendet (auch in Kombination) werden, um den eigentlichen Körper anzunähern oder eine triangulierte Variante. Für Kollisionserkennung stehen AABB (axis-aligned bounding box) oder GPU zur Verfügung. Daneben gibt es noch Features für allgemeine 6-DOF Constraints, mit deren Hilfe man Joint Mechanik umsetzt. Es stehen verschiedene Solver zur Verfügung, wie z.B. der Projected Gauss Seidel oder der QuickStep Solver von ODE (Open Dynamics Engine)³.

Bullet läuft standardmäßig mit einer Aktualisierungsrate von 60 Hz. Bei Unterschieden zwischen Bullet und dem Renderer findet eine Interpolation statt. Eingesetzt wird Bullet in 3D-Grafiksoftware, Computer- und Videospiele, Simulationen und Filmproduktionen.

5.2.1.3. NetBeans Platform

Die NetBeans Platform (NBP) ist ein Framework für die Entwicklung von GUI und Rich-Client Anwendungen und wird in Java programmiert [nbpa]. NBP hat ein Module System mit deren Hilfe die Anwendung flexibel gekapselt werden kann. Es wird zusätzlich auch der OSGi (Open Services Gateway initiative) Standard unterstützt. Das Pluginsystem erlaubt es, Code während der Laufzeit einzupflegen. Das Lifecycle Management kümmert sich um Installation, Deployment und Updates. Es stehen Funktionen wie z.B. Fensterverwaltung,

³<http://www.ode.org/>, Zugriff am 10.11.2015

Menüeinträge und Werkzeugleistenelemente zur Verfügung. Daneben werden eine Vielzahl an fertigen GUI Kompetenzen angeboten, wie z.B. Baumstrukturen oder Eigenschaftsfenster. Die NBP ist in die Entwicklungsumgebung NetBeans integriert und profitiert vom GUI-Builder Matisse. NBP ist ein stabiles weitverbreitetes Framework, das in mehreren professionellen Softwareprodukten zum Einsatz kommt [nbp].

5.2.1.4. ROS und RosJava

Das Robot Operating System (ROS) ist ein quelloffenes Robotik-Framework für mobile Roboter und verwendet eine verteilte Peer-to-Peer netzwerkbasierende Architektur [QCG⁺09]. Es stellt verschiedene Softwarebibliotheken, Treiber und Debugmöglichkeiten bereit, um die Entwicklung von robotischen Systemen zu beschleunigen.

Anwendungen werden in *Packages* organisiert, die wiederum aus *Nodes* bestehen. Eine Node ist ein eigenständig laufendes Programm, das in C++ oder Python programmiert ist. Die Nodes registrieren sich beim *ROSCore*, einer Node, die für Kommunikation zuständig ist. Der *ROSCore* sorgt dafür, dass sich die Nodes im Netzwerk finden. Ab diesem Moment läuft die TCP oder UDP-basierte Kommunikation direkt zwischen den Nodes ab.

Nodes senden Nachrichten (Messages) über *Topics* an andere Nodes. Das Senden wird als *publish* bezeichnet und das Empfangen als *subscribe*. Topics bilden Schnittstellen an die jede Node publishen oder subscriben kann. Bei einem subscribe horcht die Node auf neue Nachrichten, um bei Erfolg zu reagieren.

Da sämtliche Testroboter in dieser Arbeit ROS verwenden, kommuniziert die Simulation über ROSJava mit ihnen. Dabei werden Sensordaten aus der Simulation zu den Robotern gesendet und Aktordaten entgegengenommen, um sie in der Simulation umzusetzen. RosJava [ros] ist eine quelloffene Portierung von ROS.

5.2.2. Szenegraph

Der Szenegraph ist eine baumartige Struktur und bildet die hauptsächliche Objektverwaltungsstruktur von 3D-Engines (siehe Abschnitt 5.2.1.1). MARS verwendet die in Abbildung 5.4 aufgezeigte Struktur. An oberster Stelle befindet sich die *RootNode*. Damit ein Objekt sichtbar und aktualisiert werden kann, muss es sich unterhalb der *RootNode* befinden. Im roten Bereich befindet sich die AppState-Ebene. Jeder AppState hat eine eigene *RootNode* und ist Kindknoten der obersten *RootNode*. Im grauen Bereich befindet sich der Inhalt des SimState AppStates, des Hauptsimulations-AppStates. Um das Terrain, AUVs und SimObjects besser zu trennen, befindet sich eine Zwischenebene in Form der *SceneReflectionNode* im State. Der *RayDetectable* Knoten ist eine Besonderheit der dazu dient, Knoten mehr als einem Vater anzuknüpfen. Dies ist im herkömmlichen Szenegraphen nicht möglich, wird aber für die performante Verwaltung der Knoten beim RayCasting (z.B. beim Sonar) verwendet. Beim blauen Bereich handelt es sich um den Aufbau des AUV-Knotenbaums. Die *AUVsNode* sammelt alle AUVs. Darunter befindet sich der eigentliche Knoten der einzelnen AUVs *auv_node*. Darunter befindet sich das eigentliche dreidimensionale Modell des AUV im Knoten *auv_spatial*. Der Knoten *Physi-*

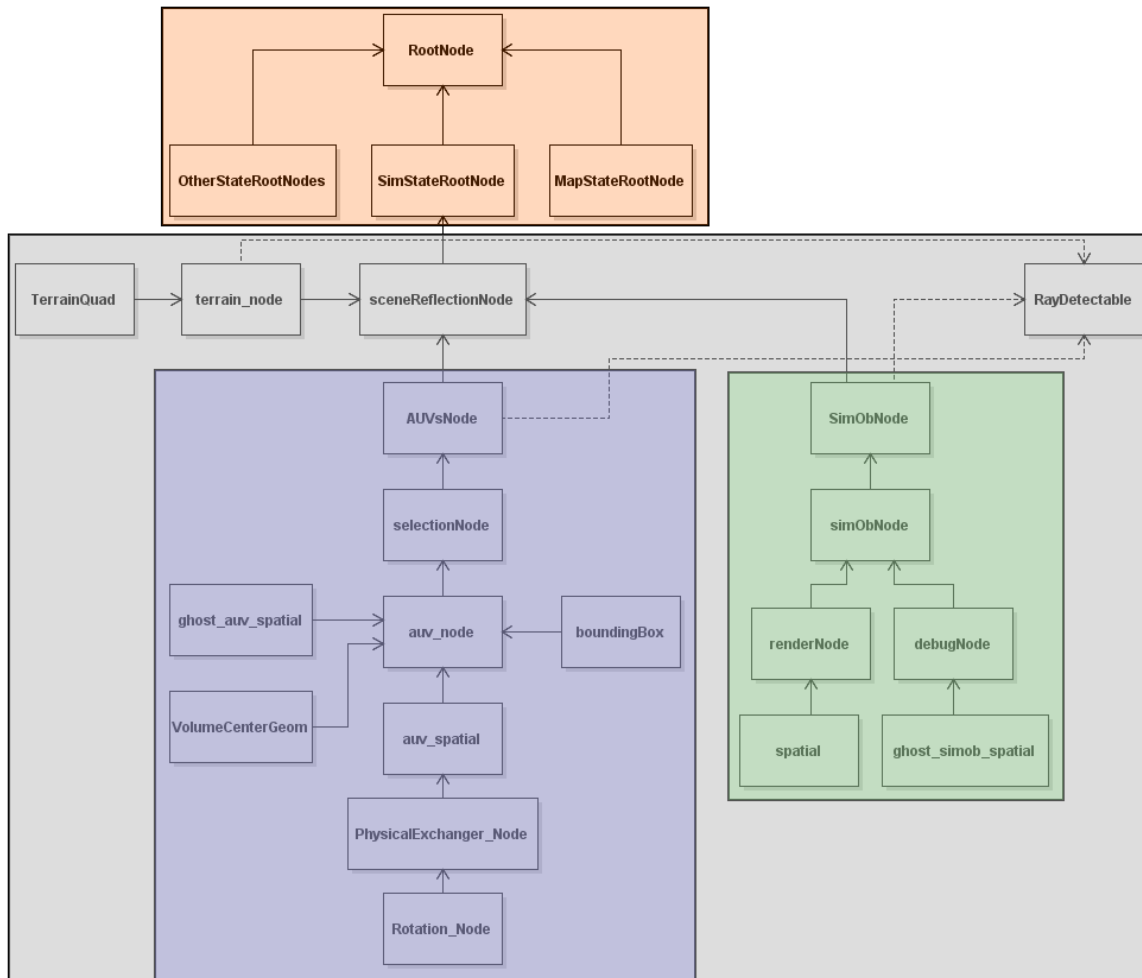


Abbildung 5.4.: Der von MARS verwaltete Szenegraph, bestehend aus AppState-Ebene (rot), SimState AppState (grau), AUV-Knotenbaum (blau) und SimObject-Knotenbaum (grün).

calExchanger_Node stellt die Knoten der Sensoren und Aktoren dar. Analog zum blauen AUV-Knotenbaum stellt der Grüne den Baum für SimObjects dar. Da SimObjects weniger mächtig sind als AUVs (statisch, keine Kräfte), ist die Struktur einfacher.

5.2.3. XML-Konfiguration

Um die Welt und AUVs zu beschreiben, verwendet MARS XML basierte Konfigurationen. Für die automatisierte Umsetzung wird *Java Architecture for XML Binding (JAXB)*⁴ verwendet. Über Annotationen werden die zu serialisierenden Klassen ausgewählt. Durch feingranulares Einstellen ist es möglich, einzelnen Variablen zu markieren. Der Marshaller

⁴<https://jaxb.java.net/>. Zugriff am 22.10.2015

kümmert sich um die Serialisierung, d.h. er wandelt die Objekte in XML-Dateien um. Es können verschiedene Gesamtkonfigurationen erstellt und einzeln geladen werden. Eine Gesamtkonfiguration besteht aus folgenden Einzelkonfigurationen:

- AUVs: Beinhaltet alle einzelnen AUV Konfigurationen als XML Dateien. Eine AUV Konfiguration beschreibt das AUV, seine physikalischen Eigenschaften wie Masse, Sensorik, Aktorik und Akkumulatoren.
- SimObjects: Beinhaltet alle einzelnen SimObject Konfigurationen als XML Dateien. Eine SimObject Konfiguration beschreibt das passive Objekt und seine Eigenschaften wie z.B. seine Position.
- Recording: XML Dateien, die die aufgezeichnete zeitliche Pose der AUVs innerhalb der Konfiguration darstellen. Diese lassen sich abspielen, um bestimmte Abläufe zu wiederholen.
- KeyConfig: Einzelne XML Datei, die Eingaben von der Tastatur zu bestimmten Funktionen der AUVs abbildet. Damit lässt sich eine Taste an einen Thruster binden und dieser zu Debugzwecken verwenden.
- PhysicalEnvironment: Einzelne XML Datei, die physikalische Parameter der Welt beschreibt, wie z.B. Wasserhöhe oder Temperatur.
- Settings: Einzelne XML Datei, die die Einstellungen der Simulation beinhaltet, wie z.B. grafische Effekte.

In Listing 5.1 ist eine beispielhafte XML Konfigurationsdatei des HANSE AUVs dargestellt. Die Konfiguration ist in 4 Bereiche eingeteilt. Der erste Bereich *Parameters* ist für die physikalischen Eigenschaften des AUVs zuständig. Dies beinhaltet z.B. die Initialposition, Masse und DebugEinstellungen. Der zweite Bereich *Sensors* ist eine Ansammlung aller Sensoren des AUVs, in diesem Beispiel ein Drucksensor (*pressureSensor*). Der dritte Bereich *Actuators* verwaltet analog die Aktoren, in diesem Beispiel einen Thruster. Im letzten Bereich *Accumulators* werden die Akkumulatoren definiert, von denen Energie bezogen werden kann.

```
<hanse>
  <Parameters>
    <params>
      <entry xsi:type="myHashMapEntryTypeVector3F" key="
        position">
        <value>
          <x>0.2</x>
          <y>-0.25</y>
          <z>0.0</z>
        </value>
      </entry>
      ...
    </params>
```

```
</Parameters>
<Sensors>
  <entry xsi:type="myHashMapEntryTypeSensors" key="pressure/
    depth">
    <value xsi:type="pressureSensor">
      <noise>
        ...
      </noise>
      <variables>
        <entry xsi:type="myHashMapEntryTypeVector3F" key=
          "Position">
          <value>
            <x>-0.12</x>
            <y>0.19</y>
            <z>-0.31</z>
          </value>
        </entry>
        ...
      </variables>
    </value>
  </entry>
</Sensors>
<Actuators>
  <entry xsi:type="myHashMapEntryTypeActuators" key="motors/
    downFront">
    <value xsi:type="seaBotixThruster">
      <noise>
        ...
      </noise>
      <variables>
        <entry xsi:type="myHashMapEntryTypeObject" key="
          enabled">
          <value xsi:type="xs:boolean">true</value>
        </entry>
        ...
      </variables>
      <Actions>
        ...
      </Actions>
    </value>
  </entry>
</Actuators>
<Accumulators>
  ...
</Accumulators>
</hanse>
```

Listing 5.1: Beispielhafte und gekürzte XML Konfiguration vom AUV HANSE. "..." deutet an das weitere XML-Knoten eingefügt werden können.

5.2.4. Benutzungsschnittstelle

Die Benutzungsschnittstelle wurde mit Hilfe von NBP programmiert (siehe Abschnitt 5.2.1.3). In dieser werden *Modes* deklariert, die Fensterbereichen entsprechen. Für MARS wurde eine eigenen Fensteranordnung deklariert. Diese ist aufgezeigt in Abbildung 5.5. Oben befindet sich das *File Menu*, in dem sich gewöhnliche Menueinträge befinden, wie z.B. das Speichern oder Laden einer Konfiguration oder das Aufrufen von Optionsdialogen. Direkt darunter ist die *Toolbar* positioniert. In dieser befinden sich Buttons, die bestimmte Aktionen schneller verfügbar machen, wie z.B. das Starten und Pausieren der Simulation oder eine ROS-Verbindung aufzubauen. Im rechten Bereich befindet sich die *Main View*. In dieser wird das Geschehen dreidimensional dargestellt. Weitere Ansichten können aufgerufen werden und werden über *Tabs* verwaltet. Die *Tree/MiniMap* befindet sich links von der *Main View*. In dieser befindet sich eine baumartige Struktur (Tree), die alle AUVs darstellt und deren interne sensorische und aktorische Struktur. Alternativ wird eine MiniMap angezeigt. Ebenfalls werden über *Tabs* die verschiedenen Sichten verwaltet. Direkt unterhalb der *Tree/MiniMap* befinden sich die *Properties*. In diesen werden die veränderbaren Eigenschaften der AUVs aus dem Tree angezeigt. Zuletzt ist unten ein Bereich reserviert für Output, typischerweise Textmeldungen, die ausgeblendet werden können. Dieser Bereich heißt *Log/Status*.

In Abbildung 5.6 ist die implementierte GUI abgebildet. In der Mitte befindet sich die *Main View* mit einer Beispielszenerie, in der sich die zwei AUVs HANSE und MONSUN befinden. Beim AUV HANSE (links) sind zusätzlich Debuganzeigen aktiviert, damit Sensoren und Aktoren besser erkannt verwenden können (verschiedenfarbige Pfeile). Das AUV MONSUN ist selektiert (gelbes Aufleuchten) und die MAUS befinden sich über ihm. Dieses führt zum darstellen einer Statusmeldung, in diesem Fall, wie viel Akkukapazität zur Verfügung steht. Im linken Bereich ist der *Tree* sichtbar. In diesem Fall wurde AUV HANSE selektiert und aufgeklappt bis zum Drucksensor. Die internen Variablen des Drucksensors werden direkt unterhalb im *Properties* Fenster angezeigt. Dort kann z.B. der Name des Sensors geändert werden.

Die MARS GUI bietet noch weitere Eigenschaften, die in Abbildung 5.7 und 5.8 aufgezeigt werden. Abbildung 5.7 a) zeigt das Optionpanel. In diesem stehen alle Einstellungen der aktuellen Konfiguration. Dies sind z.B. Grafikoptionen oder Einstellungen von Plugins die von fremden Quellen geschrieben wurden. Abbildung 5.7 b) zeigt einen Plot des Drucksensors über die Zeit. Jeder Sensor mit einfachen Sensordaten, wie z.B. skalaren oder vektoriellen Werten kann dargestellt werden. Die Plotfunktionalität wird durch die Java Bibliothek *JChart2D*⁵ ermöglicht. Abbildung 5.8 a) und Abbildung 5.8 b) zeigen Sonardaten in Form einer planaren und polaren Ansicht. Sämtliche Farben lassen sich modifizieren, um die bestmögliche Sicht auf die Daten zu ermöglichen.

Neben den oben genannten Sichten gibt es noch weitere Sichten und GUI-Eigenschaften. Es ist ebenfalls möglich, sich über ein separates Fenster in der *Main View*, die Kameraansicht des AUVs anzuzeigen. Jedes AUV, Sensor und Aktor lässt sich kopieren und

⁵<http://jchart2d.sourceforge.net/>, Zugriff am 22.10.2015

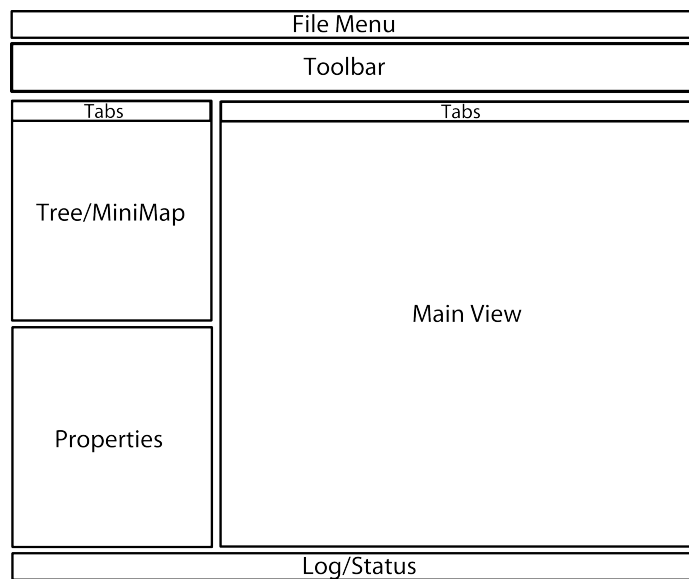


Abbildung 5.5.: Grundsätzlicher Aufbau der GUI von MARS. Im Bereich *MainView* befindet sich dreidimensionale Repräsentation. Im linken Bereich *Tree* und *Properties* werden AUVs in einer Baumstruktur dargestellt und ihre Werte können manipuliert werden.

sofort einsetzen. Dies ist über den *Tree* möglich oder indem AUVs per drag and drop in die *Main View* gezogen werden. Für Kopieren der Java Objekte wird die *Java DeepCloning library*⁶ verwendet. Dies ist nützlich für das schnelle Erstellen und Verändern einer Schwarmkonfiguration. Des weiteren kann durch einen Rechtsklick auf ein AUV im Baum oder in der Hauptansicht ein Kontextmenü aufgerufen werden, mit welchem man diverse Optionen einschalten kann, wie z.B. Debugfunktionen.

⁶<https://code.google.com/p/cloning/>

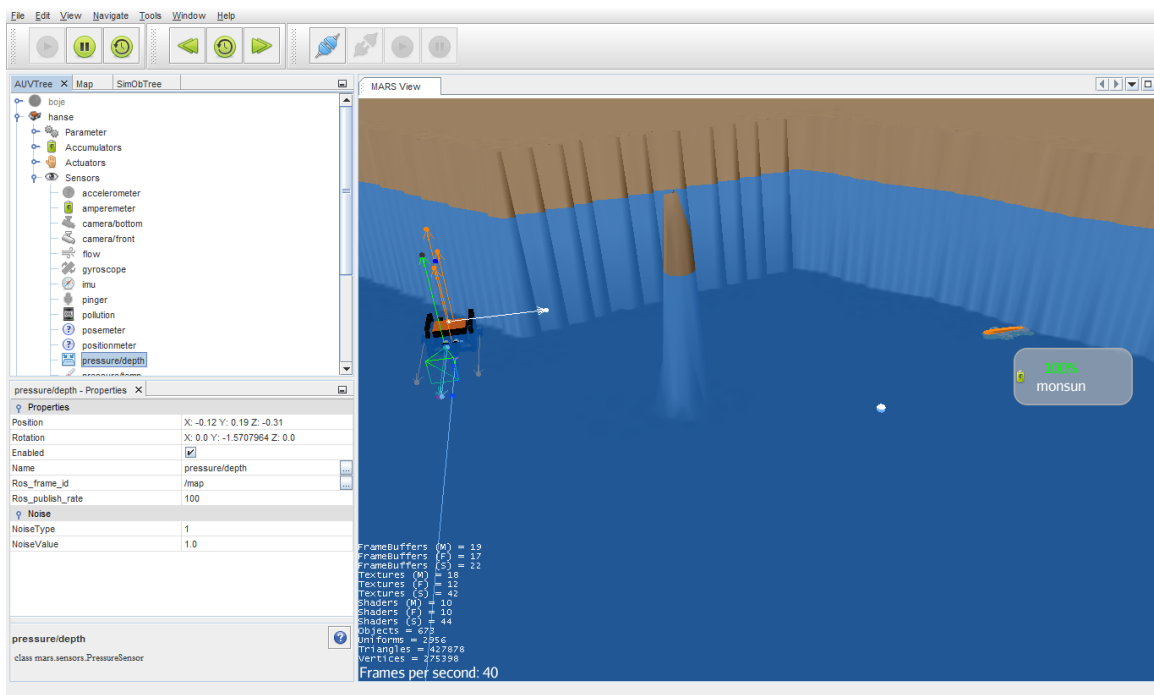
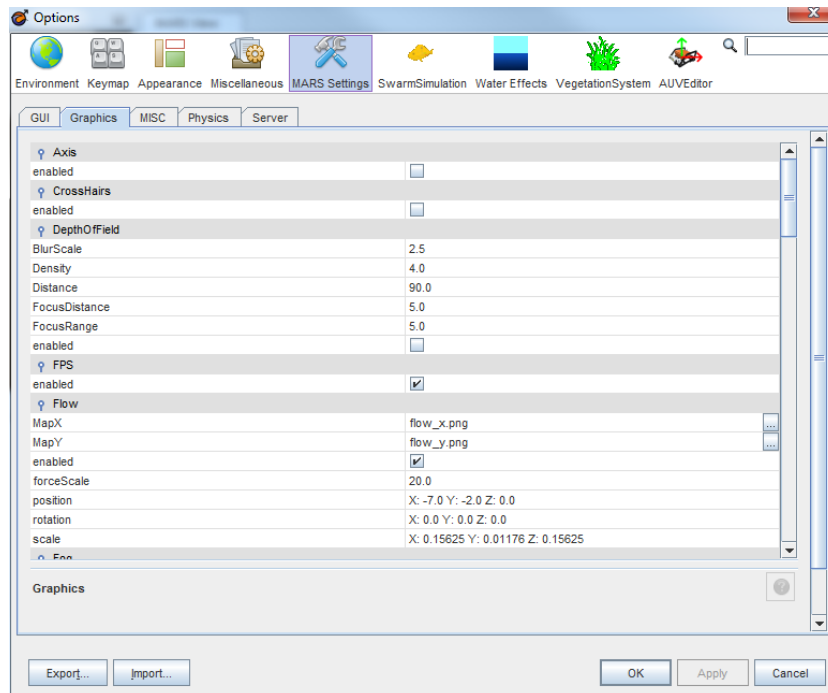


Abbildung 5.6.: Beispielhafte Ansicht von MARS. Mittig ist die 3D-Umgebung mit AUVs zu sehen. Links die aufgeklappte Baumstruktur.

Kapitel 5. Prototypische Simulationsumgebung MARS

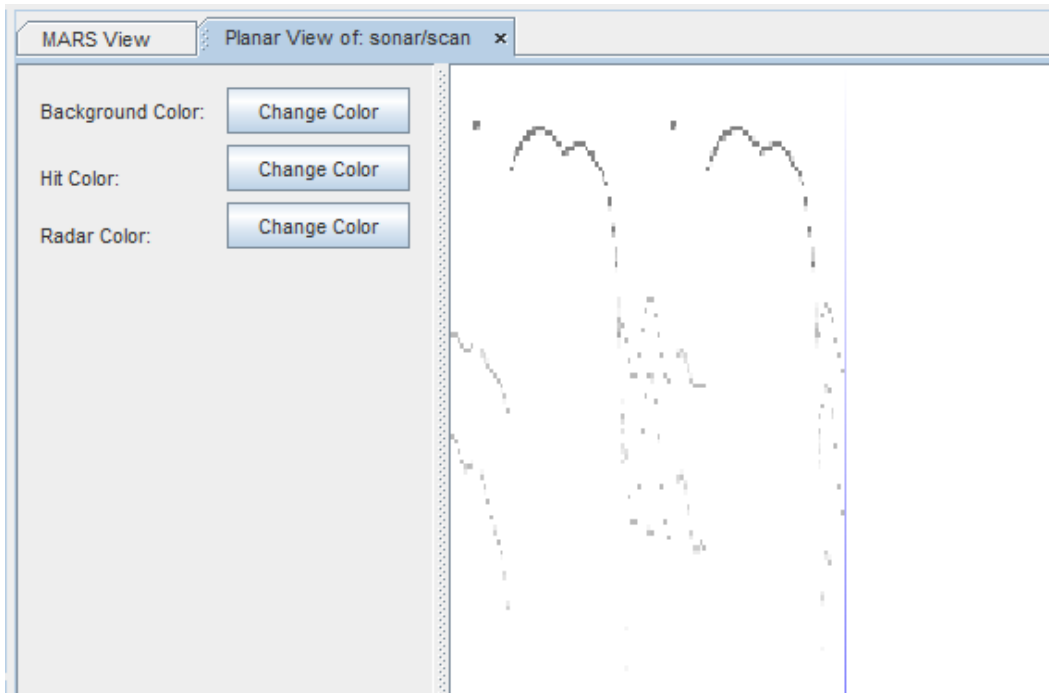


(a) Optionspanel.

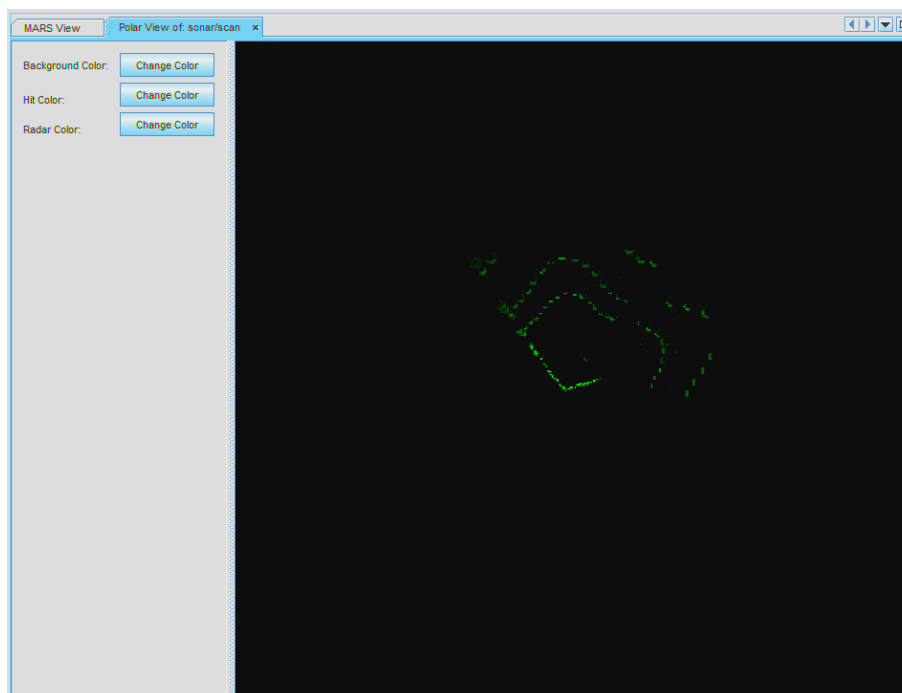


(b) Allgemeiner Plot von Sensordaten.

Abbildung 5.7.: Verschiedene zusätzliche Fenster in MARS.



(a) Planare Ansicht des Sonars.



(b) Polare Ansicht des Sonars.

Abbildung 5.8.: Visualisierung von Sonardaten in MARS.

Kapitel 6.

Evaluation

Irgend etwas wird von irgend jemandem nach irgendwelchen Kriterien in irgendeiner Weise bewertet.

(Helmut Kromrey)

Inhaltsangabe

6.1. Echtzeitfähigkeit	94
6.1.1. Testaufbau	94
6.1.2. Grafischer Benchmark	95
6.1.3. AUV-Simulation	99
6.1.4. Kommunikation	103
6.1.5. Fazit	110
6.2. Vergleich zwischen Simulation und Realität	111
6.2.1. Tiefenregelung	111
6.2.2. Sonar	113
6.3. Schwärme	118
6.3.1. Formationsfahrt	118
6.3.2. Load-Balancing Verhalten	119
6.4. Benutzungsschnittstelle	121
6.4.1. Durchführung	121
6.4.2. Auswertung	122

Dieses Kapitel beschäftigt sich mit der Evaluation des Modells und der prototypischen Simulationsumgebung MARS. Abschnitt 6.1 definiert die Gütekriterien für die Echtzeitfähigkeit und es werden Test durchgeführt, um diese nachzuweisen. Abschnitt 6.2 vergleicht simulierte und reale Ergebnisse von Experimenten mit den AUVs MONSUN und HANSE in Form von einer Tiefenregelung (6.2.1) und beim Sonarmodell (6.2.2) über einen qualitativen Vergleich. Abschnitt 6.3 weist die Schwarmfähigkeit nach, indem zwei Verhalten

mit MONSUN, V-Formationsfahrt und Load-Balancing, unter Zuhilfenahme von MARS simuliert und evaluiert werden. Der letzte Abschnitt 6.4 beurteilt die Benutzungsschnittelle von MARS in Form einer benutzerorientierten Evaluation.

6.1. Echtzeitfähigkeit

Um die Echtzeitfähigkeit zu testen, wurden verschiedene Szenarien definiert. Im ersten Szenario 6.1.2 wird die Bildwiederholrate gemessen, ähnlich zu grafischen Benchmarks. Im zweiten Szenario 6.1.3 wird auf die Simulation vieler AUVs eingegangen und wie sie sich auf die Bildwiederholrate auswirken. Das letzte Szenario 6.1.4 weist die Echtzeitfähigkeit der Kommunikation zwischen Simulator und der AUV-Software nach.

Die Bildwiederholrate bildet sich aus der Update-Loop der jMonkeyEngine heraus. Der Aufbau der Update-Loop ist in Abbildung 6.1 zu sehen. Am Anfang steht die Initialisierung der Update-Loop. Danach folgt die Eingabeverarbeitung, z.B. Maus- oder Tastatureingaben. Als zweites wird der interne Zustand aktualisiert, z.B. benutzerdefinierter Code. Als letztes folgt das eigentliche Video-Rendering. Die Update-Loop wiederholt sich dann, beginnend bei der Eingabeverarbeitung. Alternativ kann diese auch beendet werden. Ein kompletter Durchlauf der Update-Loop wird auch als Frame bezeichnet und die Anzahl der durchlaufenen Frames in der Sekunde als FPS (Frames per Second).

Für die grafische Qualität ist eine ausreichende FPS notwendig. Das menschliche Auge kann zwischen 10-12 Bilder in der Sekunde individuell erkennen [RM00]. Ab dieser Grenze beginnt es eine flüssige Bewegung wahrzunehmen, wobei es individuelle Unterschiede bei Personen gibt. Die Grenze der menschlichen Wahrnehmung für das Auge liegt bei 48 Lichtblitzen in der Sekunde, also Farbwechsel von schwarz zu weiß. In der Filmbranche haben sich historisch 24 Bilder in der Sekunde durchgesetzt, weil bei knapp über zwölf Bildern pro Sekunde immer noch ein Flimmern wahrzunehmen ist. Durch entsprechende Verschlussklappen und bei 24 Bilder in der Sekunde ist dies von den meisten Menschen nicht mehr wahrnehmbar. Da bedingt durch die Update Loop logische Veränderungen, die im *Update Game State* stattfinden, sich auf die FPS auswirken, kann die FPS-Anzahl für Performanzmessungen verwendet werden. Logische Veränderungen bestehen aus benutzerdefiniertem Code von MARS, wie z.B. das Aktualisieren der Sensoren.

6.1.1. Testaufbau

Die Szenarien wurden auf den Systemen in der Tabelle 6.1 getestet. Das System *Laptop* ist ein typisches Notebook mit einer schwachen NVidia Grafikkarte. Das System *PC* hat eine deutlich stärkere Grafikkarte und verfügt über 4 Kerne mit HT-Technologie und ist ein sehr leistungsfähiges System.

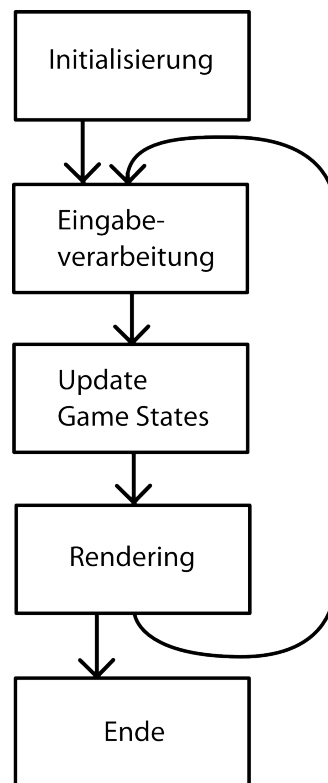


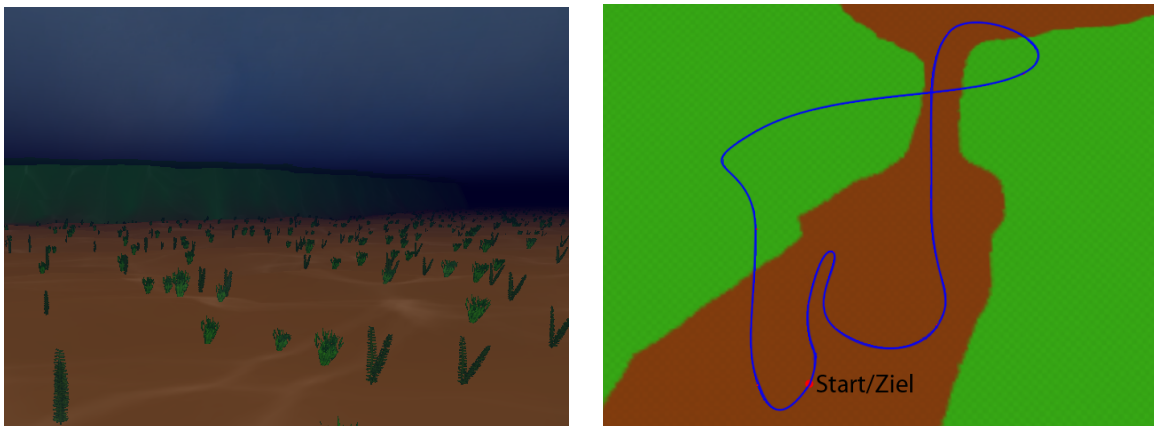
Abbildung 6.1.: Die Update-Loop der jMonkeyEngine.

Gerät	OS	Prozessor	Grafikkarte	RAM
Laptop	Win 7	Intel Core i7-2620M 4x2,7GHz	NVidia-GeForce NVS 4200M	8GB
PC	Win 7	Intel Core i7-3770 4x3.4GHz	NVidia-GeForce GTX 760	8GB

Tabelle 6.1.: Systeme auf denen der grafische Benchmark und die AUV-Simulation getestet wurden.

6.1.2. Grafischer Benchmark

Im BenchmarkszENARIO wird die generelle grafische Leistung bewertet. Dafür wurden sämtliche grafischen Effekte eingeschaltet. Dies beinhaltet Licht, Schatten, Kaustiken, Unterwassernebel, Light-Scattering, Grass, Sedimentaufwirbelung und Welleneffekte. Abbildung 6.2 a) zeigt die resultierende Unterwassersicht innerhalb des Benchmark, bestehend aus Terrain, Kaustiken, Unterwassernebel, Licht und Schatten, Pflanzen. Im Benchmark durchläuft die Kamera die Szenerie über und unter Wasser für 120 Sekunden (siehe Abbildung 6.2 b) für den Pfad der Kamera). Als Umgebung wird dieselbe HeightMap aus dem Simulationsszenario verwendet, siehe Abbildung 6.5 a). Die Auflösung betrug 640x460 Pixel.



(a) Unterwasseransicht wie sie im Benchmark auftaucht. Terrain, Kaustiken, Unterwassernebel und Pflanzen sind deutlich zu sehen.

(b) Verlauf der Kamera während des Benchmark. Grün ist einfaches Land ohne Effekte.

Abbildung 6.2.: Testszenario für den Benchmark.

6.1.2.1. Auswertung

Abbildung 6.3 zeigt den Verlauf der Framerate für die vier Fälle *keine Effekte aktiv*, *Grass aktiv*, *Wasser aktiv* und *alle Effekte aktiv*. In Tabelle 6.2 sind die durchschnittliche, minimale und maximale Framerate und die Standardabweichung angegeben. Der Benchmark lief in allen Fällen flüssig über 24 Frames mit Ausnahme des Falls, wenn alle Effekte aktiv waren. In diesem Fall gibt es einen Einbruch von ca. 3 Frames unterhalb der Grenze. Die Einbrüche der Framerate ab ca. 20 und 38 Sekunden resultieren aus einem Überflug über dem Szenario, bei dem viel des Terrains und Grasses sichtbar ist. Die durchschnittliche Framerate zeigt, dass die Grass- und Wassereffekte mit den gewählten Einstellungen ähnlich viel Leistung verbrauchen. Die limitierten Ressourcen des Laptop-Systems erlauben eine Simulation mit grafischen Effekten. Kleinere Unterschreitungen der 24 FpS Grenze wirken sich kaum auf die flüssige grafische Darstellung aus.

Abbildung 6.4 zeigt den Verlauf der Framerate für die vier Fälle *keine Effekte aktiv*, *Grass aktiv*, *Wasser aktiv* und *alle Effekte aktiv* auf dem PC-System. In Tabelle 6.3 ist die durchschnittliche, minimale, maximale Framerate und Standardabweichung angegeben. Mit sämtlichen Effekten aktiviert beträgt die durchschnittliche Framerate 144,6 FpS. Das liegt deutlich über der Grenze von 24 FpS, und es stehen genug Kapazitäten zur Verfügung für höhere Auflösungen oder mehr Effekte. Die Standardabweichung ist bei allen Tests im Bereich von 15,9 bis 17,1 und höher als beim Laptop-System, reicht aber nicht aus, um die 24 FpS Grenze zu unterschreiten. Die grafische Darstellung ist flüssig und stabil. Erwartete "Einbrüche" ergeben sich durch höhere Polygonzahl und Sichtweite während des Überflugs.

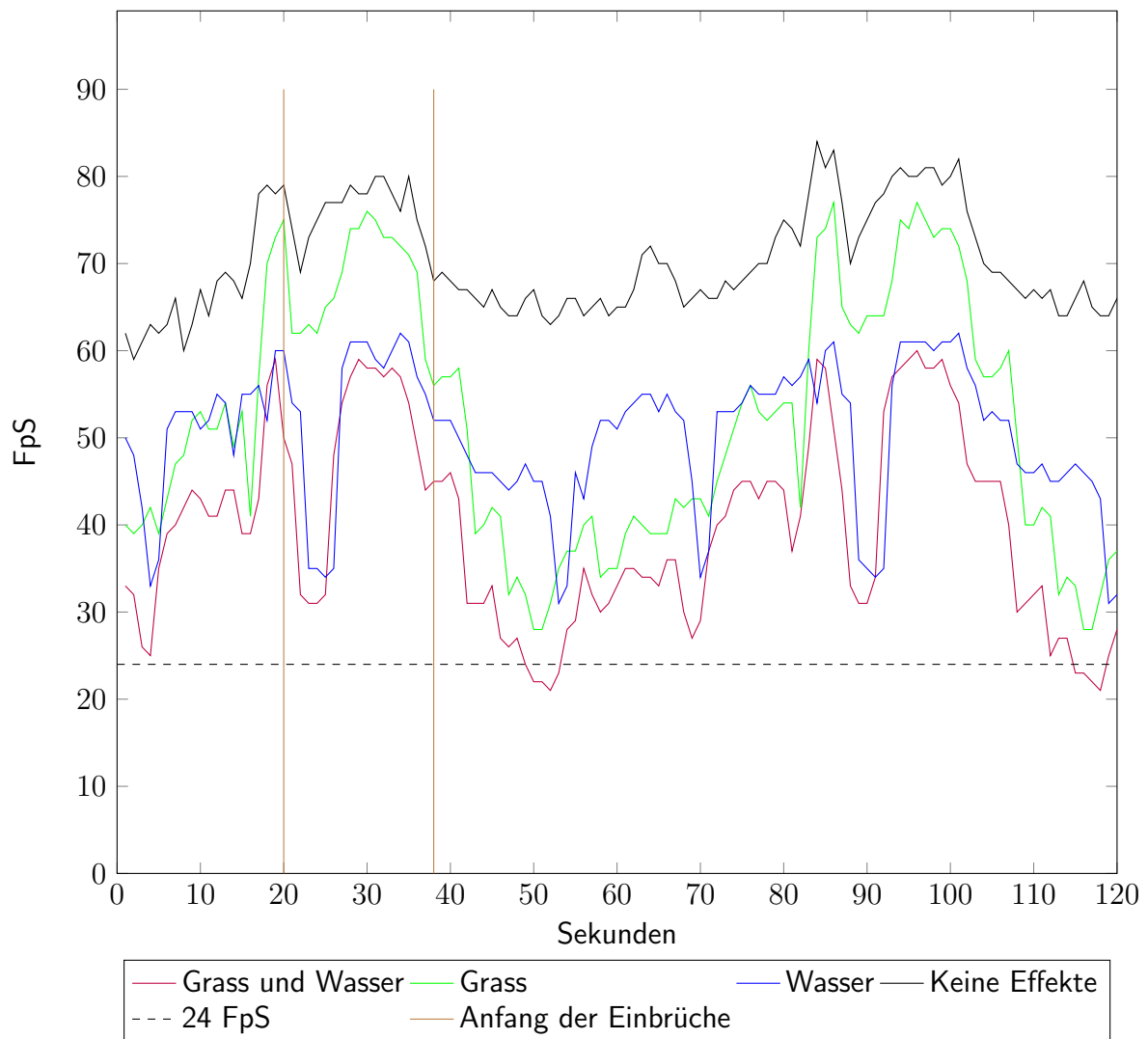


Abbildung 6.3.: Verlauf der Framerate des grafischen Benchmarks über 120 Sekunden auf dem System Laptop. Ab ungefähr Sekunde 67 wiederholt sich der Durchlauf.

Test	Grass und Wasser	Grass	Wasser	Keine Effekte
Min	21	28	31	59
Max	60	77	62	84
Durchschnitt	39,7	52,3	50,3	70,5
Standardabweichung	11,2	14,5	8,2	6,1

Tabelle 6.2.: Min, Max, Durchschnittswerte und Standardabweichung der Framerate des grafischen Benchmarks auf dem System Laptop.

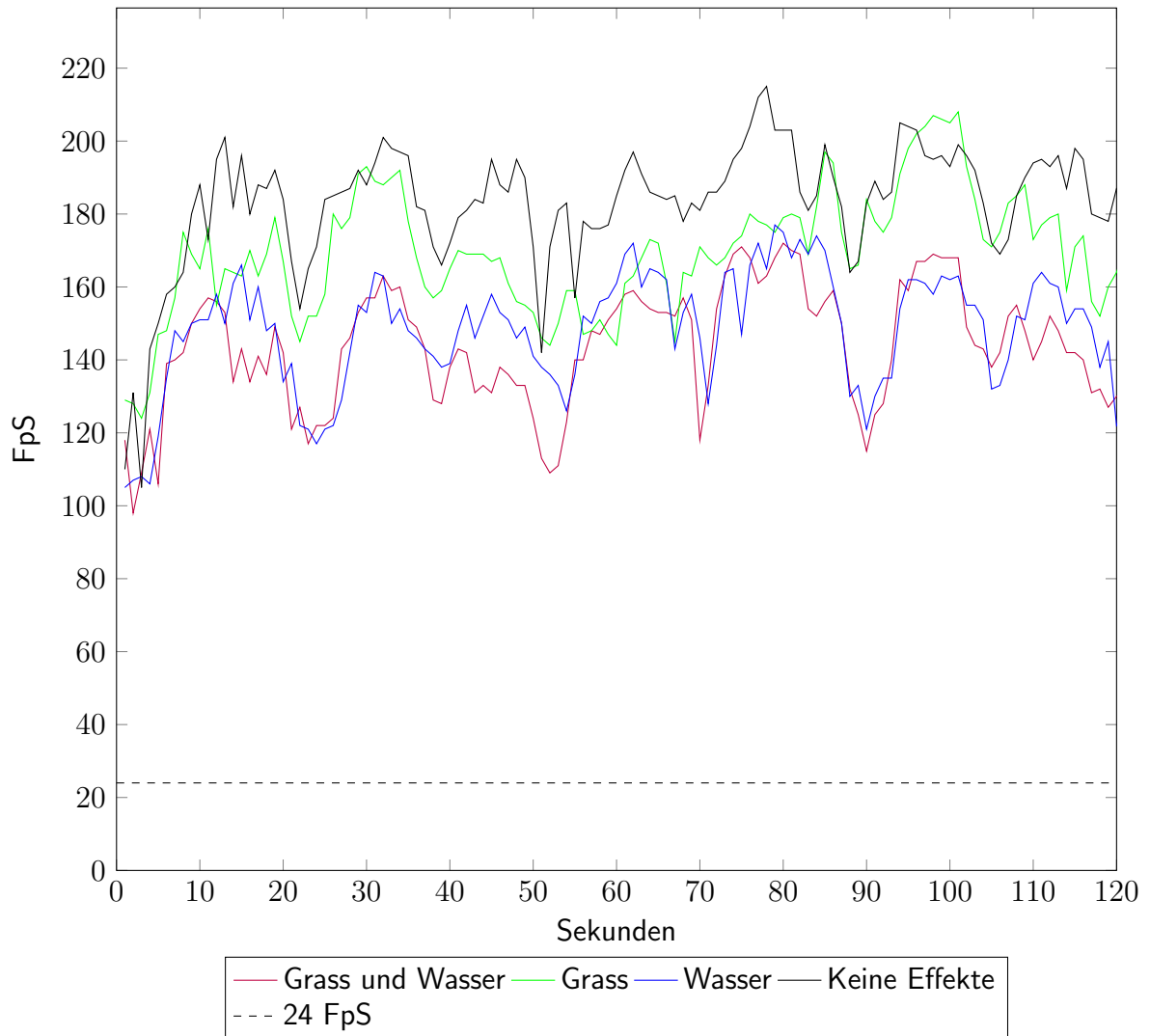


Abbildung 6.4.: Verlauf der Framerate des grafischen Benchmarks über 120 Sekunden auf dem System PC.

Test	Grass und Wasser	Grass	Wasser	Keine Effekte
Min	98	124	105	105
Max	172	208	177	215
Durchschnitt	144,6	169,3	148,2	183,1
Standardabweichung	16,4	16,7	15,9	17,1

Tabelle 6.3.: Min, Max, Durchschnittswerte und Standardabweichung der Framerate des grafischen Benchmarks auf dem System PC.

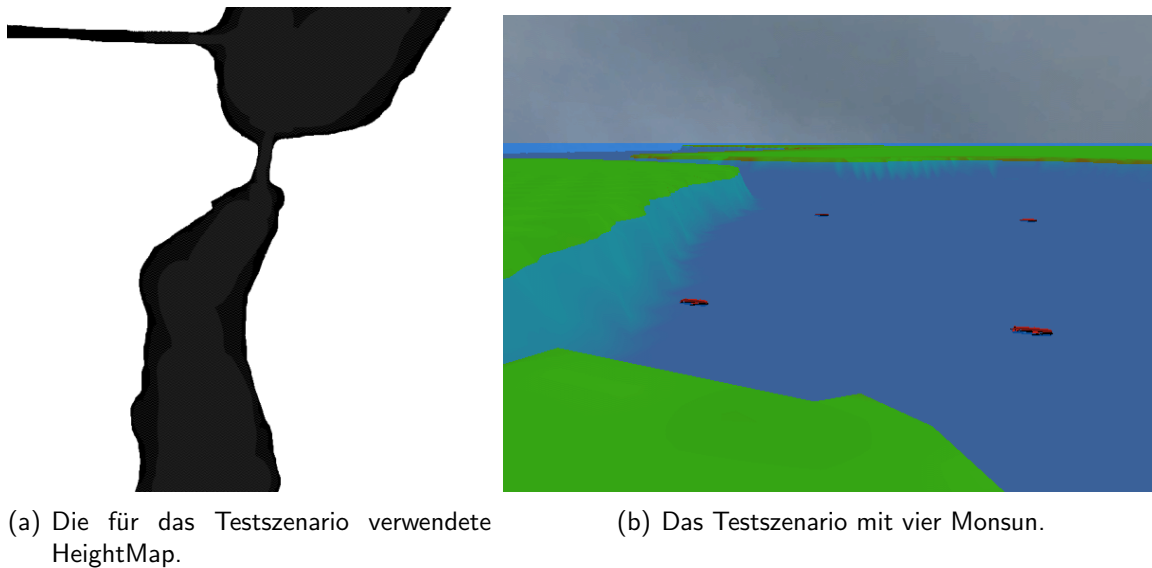


Abbildung 6.5.: Testscenario mit korrespondierender HeightMap.

6.1.3. AUV-Simulation

Das zweite Szenario bewertet die AUV-Simulation. In diesem Szenario werden mehrere AUVs des selben Typs mit ihrer Physik, Sensorik und Aktorik simuliert. Sämtliche von der Sensorik erstellten Daten werden an Visualisierungskomponenten der GUI weitergeleitet, damit eine realistische Lastsituation entsteht. Der grobe Aufbau des Szenarios ist der Abbildung 6.5 b) zu entnehmen. Die korrespondierende HeightMap ist in Abbildung 6.5 a) dargestellt. Das Terrain basiert auf einem ca. 500 Meter langem Flussabschnitt der Wakenitz, einem lokalen Gewässer in Lübeck (GPS Koordinaten: 53.862731, 10.703400). Für das AUV Simulationsszenario wird das MONSUN AUV verwendet, das wie folgt konfiguriert ist (siehe Tabelle 6.4). Das verwendete 3D-Modell besteht aus 606 Polygonen (siehe Abbildung 6.6). Level-of-Detail ist aktiviert. Beim AUV-Modell werden der Auftrieb und der Widerstand verwendet. 6 Thruster sind angebracht. Die Sensorik besteht aus einem Sonar, das mit der Ray-Casting-Technik 65 Strahlen mit einer Rate von 100 ms aussendet. Die Kamera sendet Bilder mit einer Rate von 1 Hz und einer Auflösung von 640 x 480 Pixel. Zusätzlich ist die Ansicht der Kamera in der GUI aktiv, die eine eigene *View* der jMonkeyEngine rendert. Die restliche Sensorik ist aufgrund der geringen Datenmenge oder geringer Prozessorlast zu vernachlässigen, stellt aber eine realistische Konfiguration des MONSUN AUV für eine Schwarmmission dar.

6.1.3.1. Auswertung

Abbildung 6.7 zeigt den zeitlichen Verlauf der Framerate über 120 Sekunden für bis zu 9 AUVs auf dem System *Laptop*. Tabelle 6.5 zeigt die entsprechende durchschnittliche,

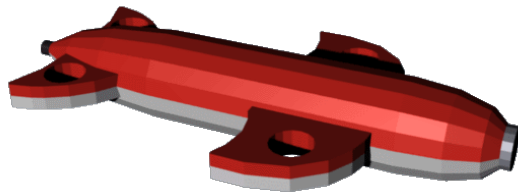


Abbildung 6.6.: 3D-Modell, das von MON-SUN innerhalb des Benchmarks verwendet wird.

Eigenschaft	Wert
Vertices	518
Polygone	606
LoD	aktiv
Auftrieb	aktiv
Widerstand	aktiv
6 x Thruster	-
1 x Sonar	65 Rays, Rate: 100 ms
1 x Kamera	640x480, Rate: 1 s
1 x Drucksensor	Rate: 100 ms
1 x Temperatur	Rate: 100 ms
1 x Modem	-
1 x IMU	Rate: 100 ms
1 x GPS	Rate: 100 ms
2 x Infrarot	je 1 Ray, Rate: 1 s

Tabelle 6.4.: Die Konfiguration des im Benchmark eingesetzten MON-SUN.

minimale, maximale Framerate und die Standardabweichung. Bis zu vier AUVs sind auf dem Laptop-System bis zu einer Framerate von 24 Fps flüssig verwendbar. Die Rate fällt kontinuierlich mit jedem hinzugefügtem AUV. Der Fall mit 9 AUVs ist bei einer durchschnittlichen Framerate von 14 FpS bedingt einsetzbar. Das eingesetzte Testsystem ist zu schwach, um ein Schwarmzenario mit 5 Teilnehmern zu simulieren, reicht aber aus um kleinere autonome Test mit bis zu vier Teilnehmern durchzuführen. Das fünfte AUV liegt knapp unterhalb der 24 Framegrenze und wäre vom Prinzip verwendbar. Die Standardabweichung von der durchschnittlichen Framerate beträgt über alle 9 Testfälle hinweg maximal 0,59. Dies zeigt, dass die AUV Simulation stabil berechnet und gerendert wird. Größere Sprünge oder Einbrüche der Framerate um mehr als 4 FpS sind nicht zu beobachten. Die gesamte CPU-Auslastung verweilte im Durchschnitt bei 35 % und die gesamte GPU-Auslastung bei 71 %. Somit sind genug Rechenkapazitäten verfügbar, um die Simulation von mehr AUVs zu ermöglichen. Dies könnte erreicht werden durch einen verbesserten Einsatz von paralleler Verarbeitung innerhalb von MARS und der jMonkeyEngine durch Threads.

Abbildung 6.8 zeigt den zeitlichen Verlauf der Framerate über 120 Sekunden für bis zu 9 AUVs auf dem System PC. 9 AUVs bilden die Grenze für die flüssige Darstellung von über 24 Fps. Die Tabelle 6.6 zeigt die entsprechende durchschnittliche, minimale, maximale Framerate und Standardabweichung. Die maximale Standardabweichung über alle 9 Testfälle hinweg beträgt 3,13. Diese ist leicht größer als beim schwächeren Laptop-System. Es sind keine größeren Einbrüche oder Sprünge der Framerate feststellbar mit Ausnahme des

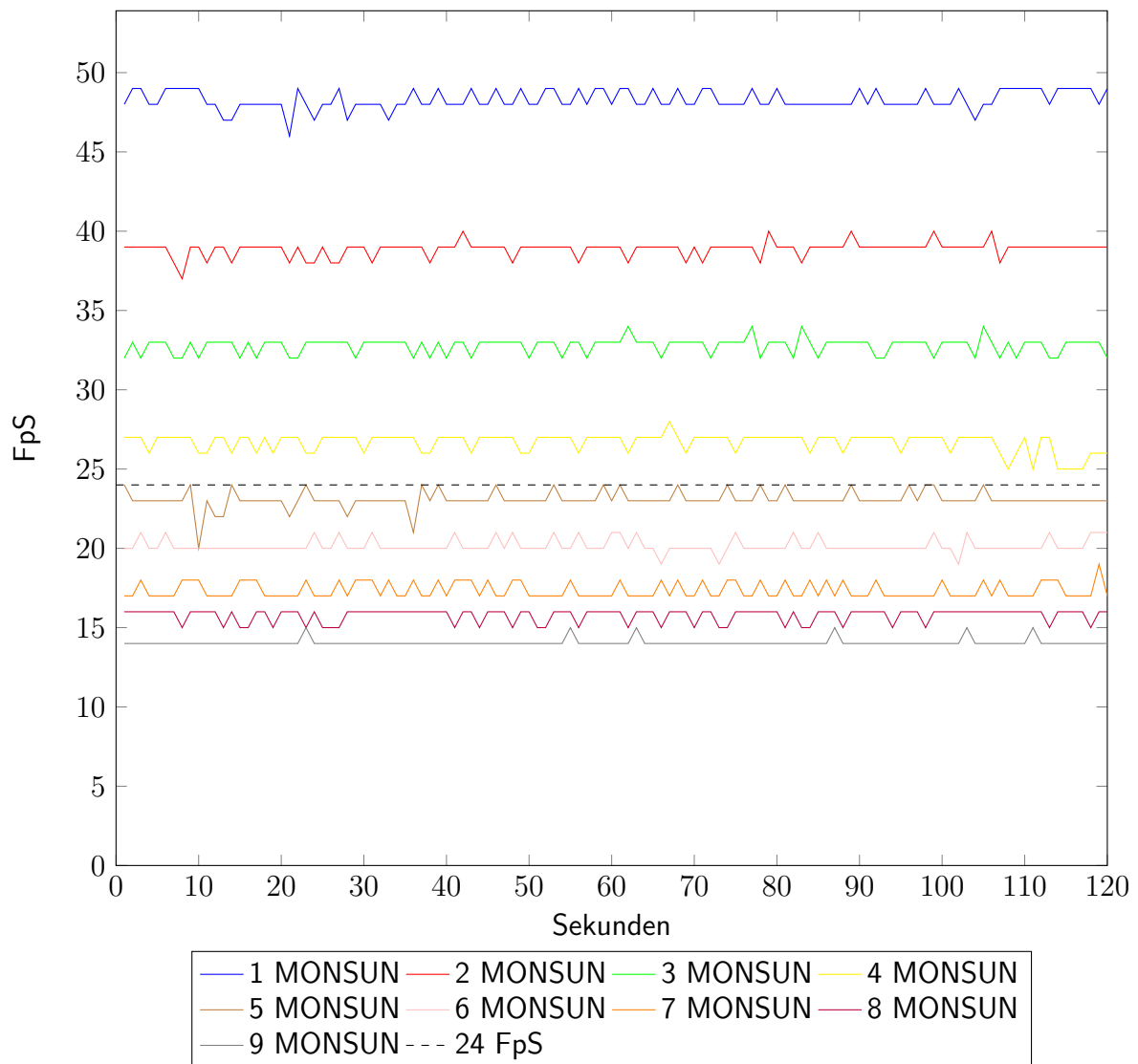


Abbildung 6.7.: Verlauf der Framerate des AUV-Simulation Benchmarks über 120 Sekunden auf dem System Laptop.

Monsunanzahl	1	2	3	4	5	6	7	8	9
Min	46	37	32	25	20	19	17	15	14
Max	49	40	34	28	24	21	19	16	15
Durchschnitt	48,3	38,9	32,8	26,9	23	20,2	17,4	15,8	14
Standardabweichung	0,59	0,46	0,49	0,58	0,54	0,43	0,5	0,42	0,22

Tabelle 6.5.: Min, Max, Durchschnittswerte und Standardabweichung der Framerate des AUV-Simulation Benchmarks auf dem System Laptop.

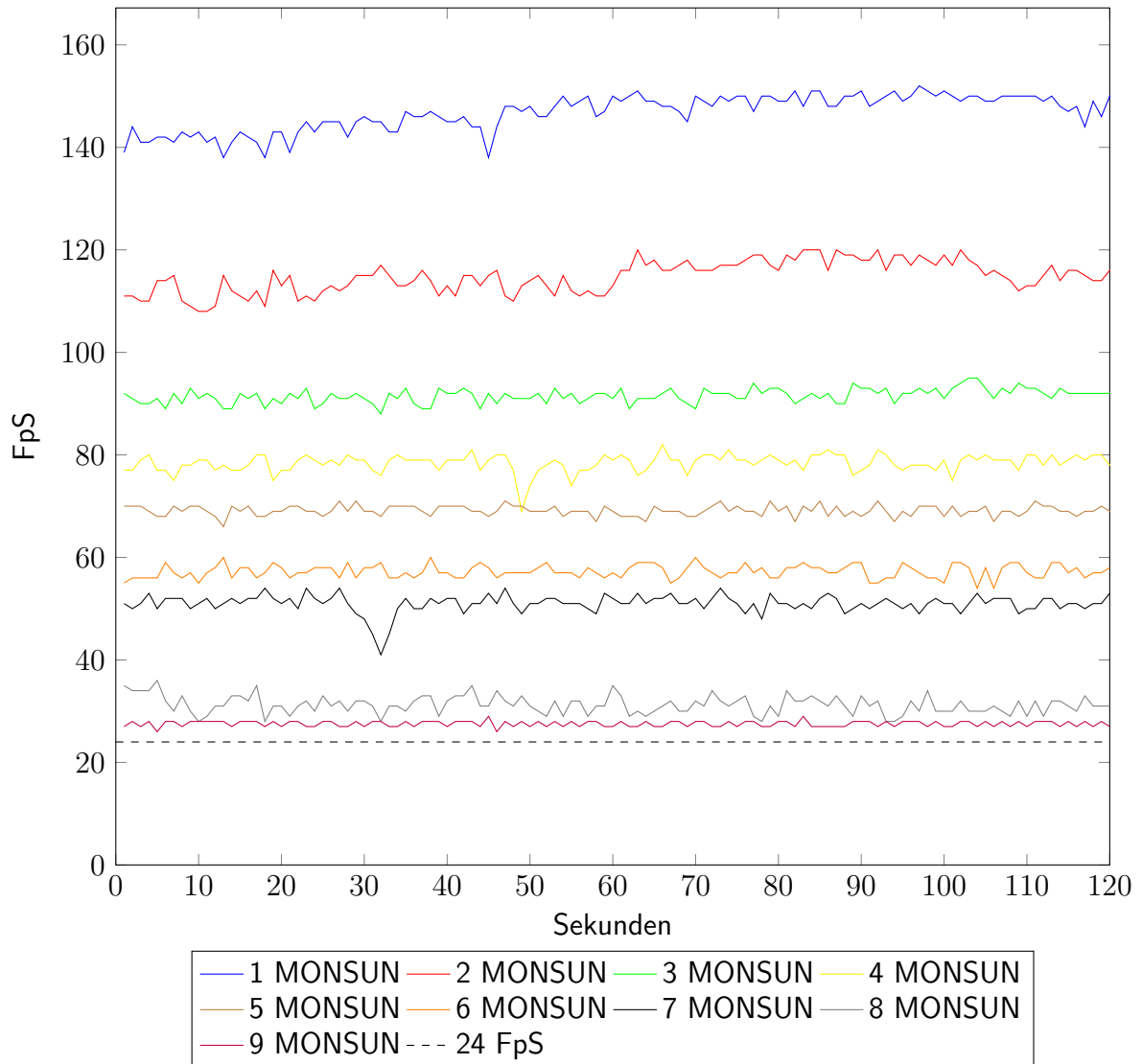


Abbildung 6.8.: Verlauf der Framerate des AUV-Simulation Benchmarks über 120 Sekunden auf dem System PC.

Monsunanzahl	1	2	3	4	5	6	7	8	9
Min	138	108	88	69	66	54	41	28	23
Max	152	120	95	82	71	60	54	36	25
Durchschnitt	148,8	114,8	92,5	78,4	69,1	57,2	50,9	31,3	23,5
Standardabweichung	3,38	3,07	1,38	1,71	1,0	1,3	1,73	1,75	0,56

Tabelle 6.6.: Min, Max, Durchschnittswerte und Standardabweichung der Framerate des AUV-Simulation Benchmarks auf dem System PC.

Falls mit 7 MONSUN AUVs. Wahrscheinliche Ursache ist ein Systemprozess, der zu einem kurzen Einbruch über 5 Sekunden von 10 Frames geführt hat. Aufgrund der zur Verfügung stehenden Kapazitäten lässt sich dies kompensieren. Somit lässt sich auf dem PC-System von einer stabilen Simulation sprechen. Die CPU-Auslastung betrug während des Test auf der gesamten CPU durchschnittlich 25 % und die gesamte GPU-Auslastung durchschnittlich 50 %. Somit sind noch ausreichend Kapazitäten verfügbar, um mehr AUVs zu simulieren. Um dies zu erreichen muss Threading stärker in JME und MARS verwendet werden, da einige Software-Komponenten davon profitieren könnten. So könnte die Struktur von MARS vermehrt auf die *ApplicationStates* der jMonkeyEngine angepasst werden und diese dann parallel betrieben werden können.

6.1.4. Kommunikation

In diesem Abschnitt wird die Echtzeitfähigkeit der Kommunikation untersucht. Die Neuimplementierung der Kommunikationsschnittstelle, die die Grundlage für diese Experimente bildet, wurde im Rahmen einer Bachelorarbeit vollzogen [Bus15]. Ebenfalls wurde das Testprogramm, das in dieser Arbeit zum Einsatz kam, für die Messungen verwendet.

Bei dieser Messung wurde ein MONSUN AUV in der Simulation erstellt. Es wurde eine TCP-Verbindung zu einem Testprogramm aufgebaut. Die Verbindung wurde ohne und mit Kamera erstellt und des Weiteren mit und ohne einer GZip Komprimierung der Daten über den Java GZIPOutputStream¹. Die Simulation und das Testprogramm liefen auf dem selben System über *localhost*. Dadurch sind Zeitdifferenzen durch unterschiedliche Systemzeiten ausgeschlossen. Ein Test auf getrennten, im Netzwerk auseinanderliegenden, System wurde nicht durchgeführt. Die Simulation sendet Datenpakete der einzelnen Sensoren an das Testprogramm, wobei ein Zeitstempel bei der Erstellung der Sensornachricht angehängt wird. Dadurch lässt sich die Verzögerung zwischen dem Erstellen der Nachricht in MARS bis zum Empfang im Testprogramm berechnen. Ein Datenpaket ist hierbei ein komplette Sensornachricht. Für den Versuch mit mehreren AUVs wurden die entsprechende Anzahl an Testprogrammen instantiiert und die Anzahl an MONSUN AUVs in MARS erstellt. Die Konfiguration des verwendeten MONSUN stammt aus Tabelle 6.4 in Abschnitt 6.1.3. Als Testsystem wurde das System *Laptop* aus Tabelle 6.1 in Abschnitt 6.1.1 verwendet.

6.1.4.1. Auswertung

Abbildung 6.9 zeigt die Verzögerung der einzelnen Datenpakete für 1000 Pakete. Bei der Übertragung mit Kameradaten sieht man eine deutliche Zunahme der Verzögerung der Datenpakete, in diesem Fall der Kamerapakete, auf bis zu 141 ms. Allerdings ist der Durchschnittswert mit 4,75 ms sehr gering, wie in Tabelle 6.7 zu sehen. Die Auswirkung auf die Verzögerung der anderen Pakete ist marginal. Die Übertragung ohne Kameradaten ist nahezu verzögerungsfrei. Der Maximalwert beträgt 5 ms und der Durchschnittswert 0,9 ms

¹<http://docs.oracle.com/javase/7/docs/api/java/util/zip/GZIPOutputStream.html>, Zugriff am 18.10.2015

bei einer Standardabweichung von 0,49 ms. Wenn man den realen Regler und Drucksensor von Monsun betrachtet, so stellt die Verzögerung kein Problem dar und man kann von Echtzeitfähigkeit sprechen. Der Tiefen- und Lageregler und Drucksensor arbeiten mit einer Rate von 10 Hz. Somit würde die Hinverzögerung 0,89 % der Gesamtverzögerung ausmachen. Lediglich die Kameradaten sind stark verzögert. Allerdings benötigen entsprechende komplexe Bildverarbeitungen ebenfalls eine gewisse Zeit und somit fällt die Verzögerung der Kameradaten weniger stark ins Gewicht.

Eine Kompression der Daten führt zu keiner Verbesserung der Verzögerung, obwohl die Netzwerkauslastung um ca. 88 % geringer ausfällt als ohne Kompression [Bus15]. Abbildung 6.10 zeigt die Verzögerung für den Fall der Kompression mit und ohne Kameradaten. Die durchschnittlichen Verzögerungen beider Messungen sind mit 253,8 ms und 316,99 ms sehr hoch, wie den Werten aus Tabelle 6.7 entnommen werden kann. Des Weiteren sind andere unkritische Datenpakete, wie z.B. einfache Drucksensordaten, entsprechend mitverzögert. Zwar läuft jede Verbindung in einem eigenem Thread, allerdings müssen nachfolgende Daten auf den Versand der vorherigen Daten warten. Dies verzögert die nachfolgenden Pakete. Eine Lösung wäre ein performantere Kompression als die GZip-basierte zu verwenden oder die Kameradaten in einen eigene Socket oder Thread auszulagern.

Die Abbildungen 6.11 bis 6.19 zeigen die Verzögerung der einzelnen Datenpakete von neun MONSUN AUV auf dem System PC, die sich gleichzeitig mit neun Testprogramme verbunden haben. Dabei wurde keine Komprimierung verwendet und die Frontkamera von MONSUN war eingeschaltet. Die Konfiguration der AUVs war die gleiche wie in den Messungen zuvor. Es zeigt sich keine Verschlechterung der Verzögerung bei irgend einem der neun AUVs im Vergleich zu einem einzigen AUVs aus dem Vorversuch. Der Maximalwert ist besser, im Vergleich zu den Vormessungen, aufgrund der stärkeren Leistungsfähigkeit des System PC. Somit lässt sich ein großes Szenario, im Hinblick auf die Kommunikation, echtzeitfähig simulieren. In der Implementierung läuft jede einzelne Verbindung in einem eigenen Thread, was der Performanz zu Gute kommt.

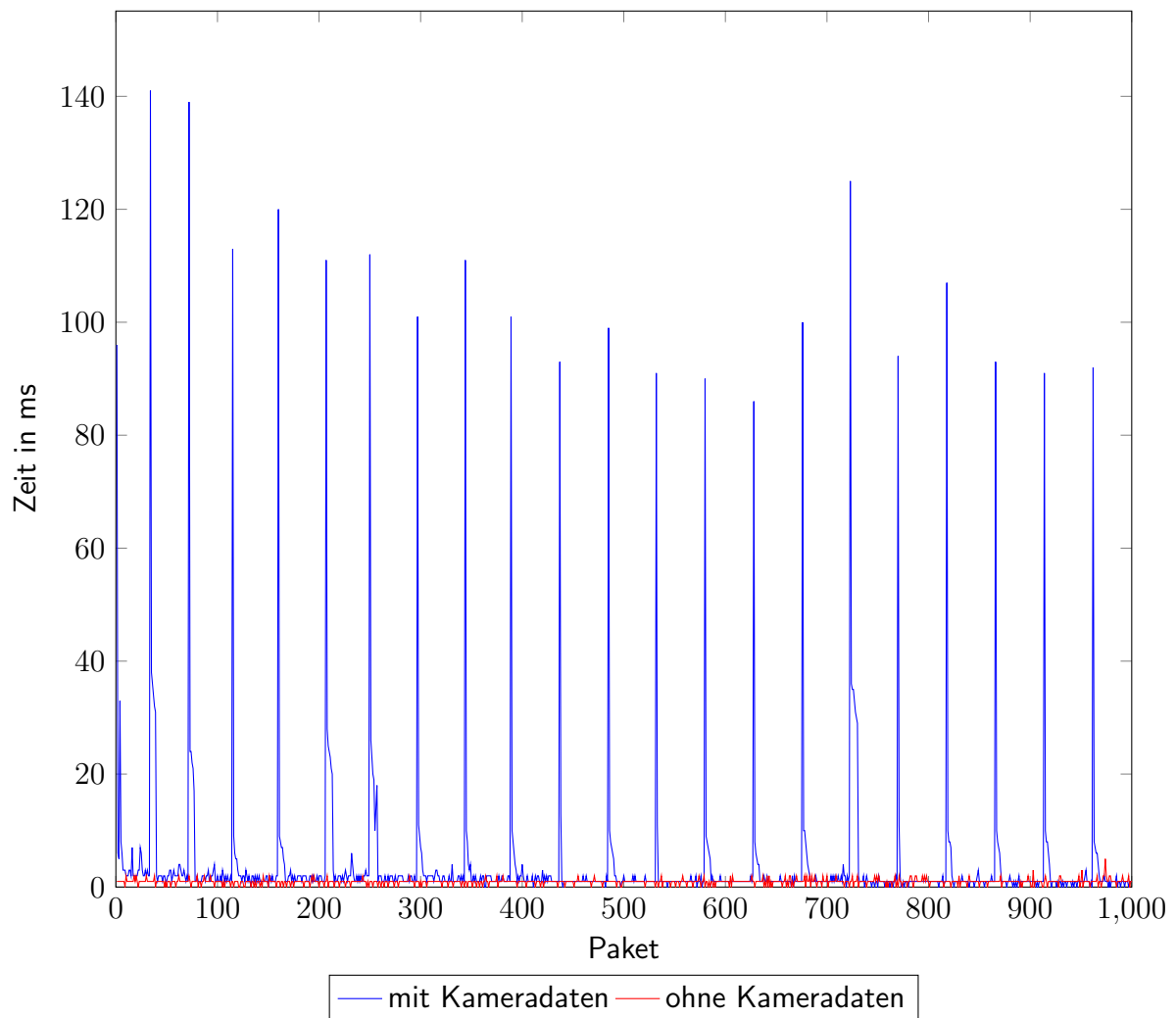


Abbildung 6.9.: Verzögerung für einzelne unkomprimierte Datenpakete eines MONSUN mit und ohne Kameradaten auf dem System Laptop.

Test	Kamera	keine Kamera	Kamera komprimiert	keine Kamera komprimiert
Min	0	0	2	2
Max	141	5	781	926
Durchschnitt	4,75	0,9	253,8	316,99
Standardabweichung	15,9	0,49	164,2	189,56

Tabelle 6.7.: Min, Max, Durchschnittswerte und Standardabweichung der Verzögerungen auf dem System Laptop in ms.

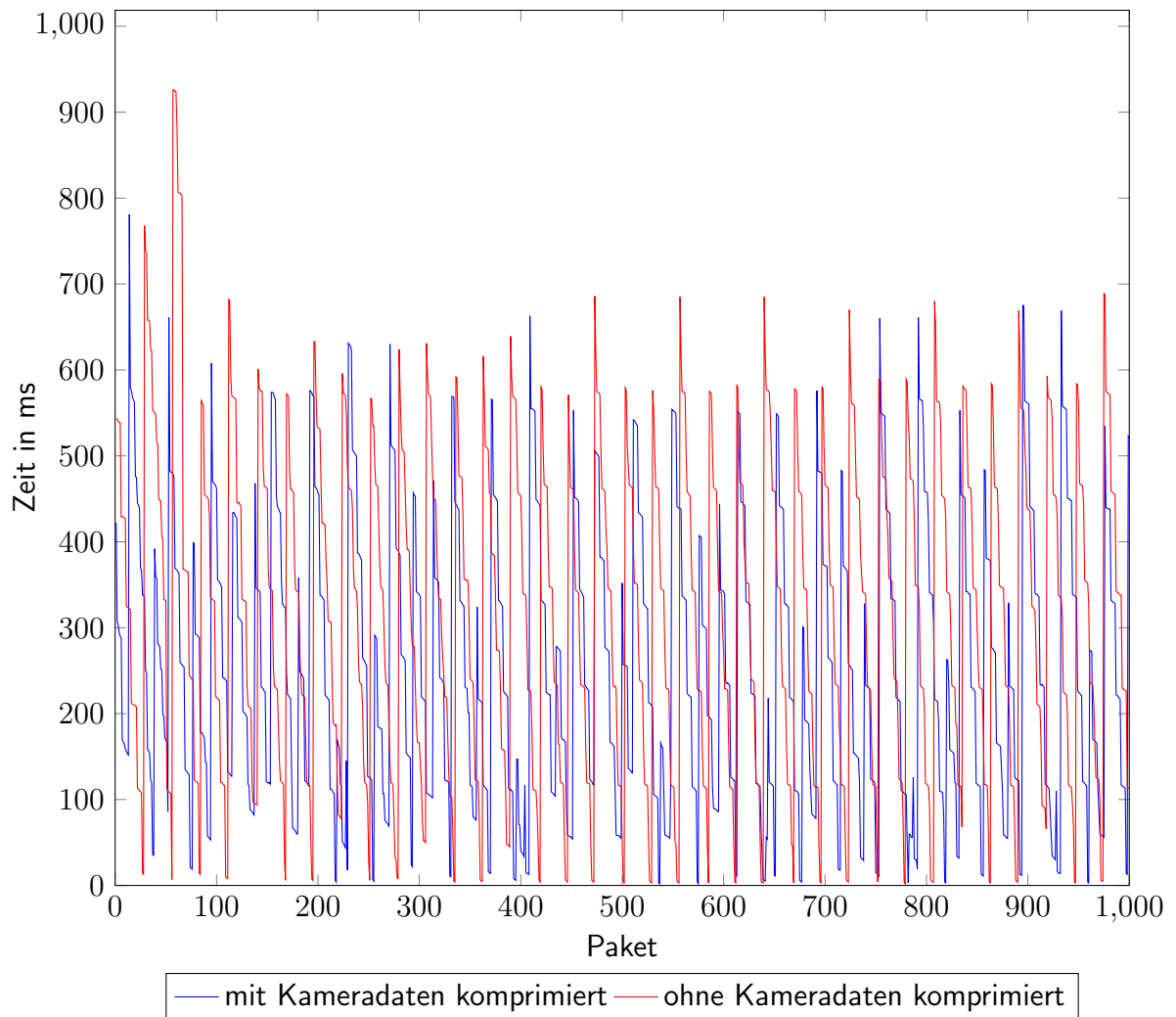


Abbildung 6.10.: Verzögerung für einzelne komprimierte Datenpakete eines MONSUN mit und ohne Kameradaten auf dem System Laptop.

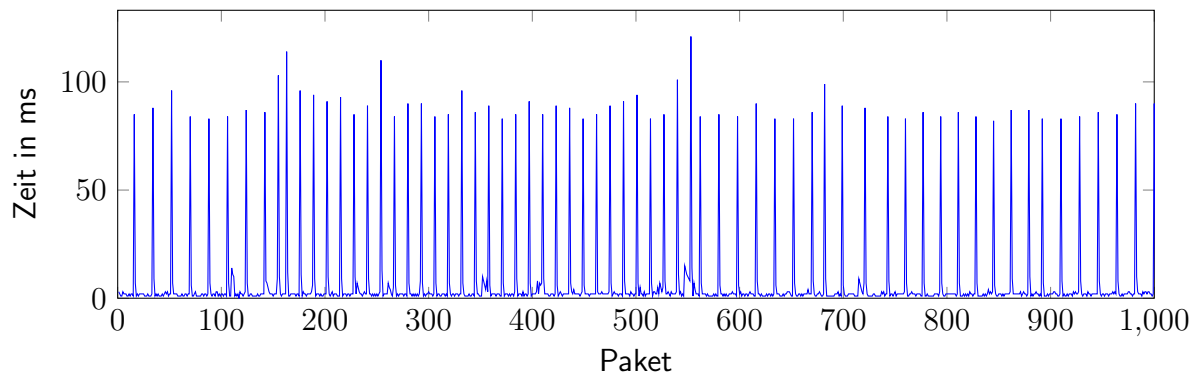


Abbildung 6.11.: Verzögerung einzelner Datenpakete für Monsun Nr.1 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

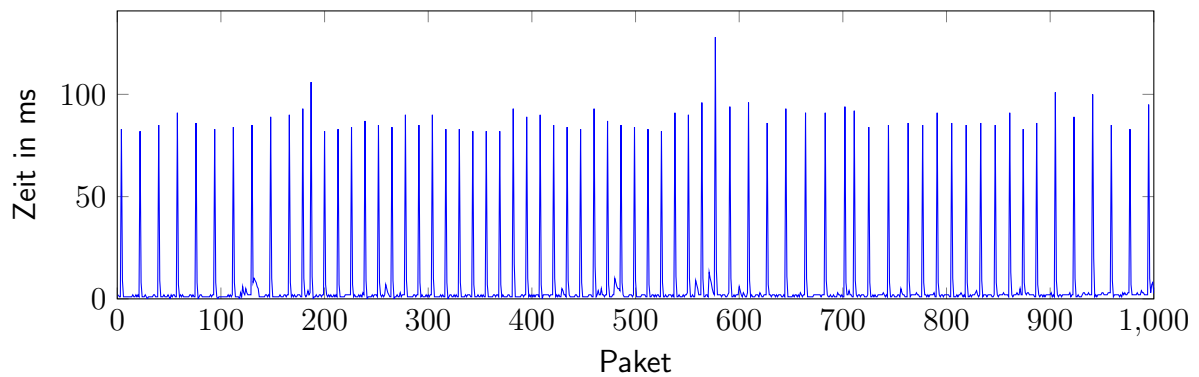


Abbildung 6.12.: Verzögerung einzelner Datenpakete für Monsun Nr.2 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

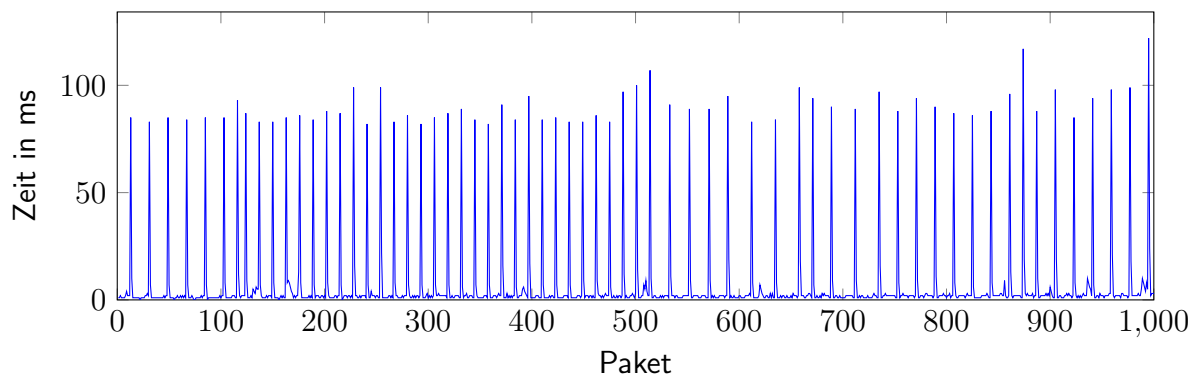


Abbildung 6.13.: Verzögerung einzelner Datenpakete für Monsun Nr.3 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

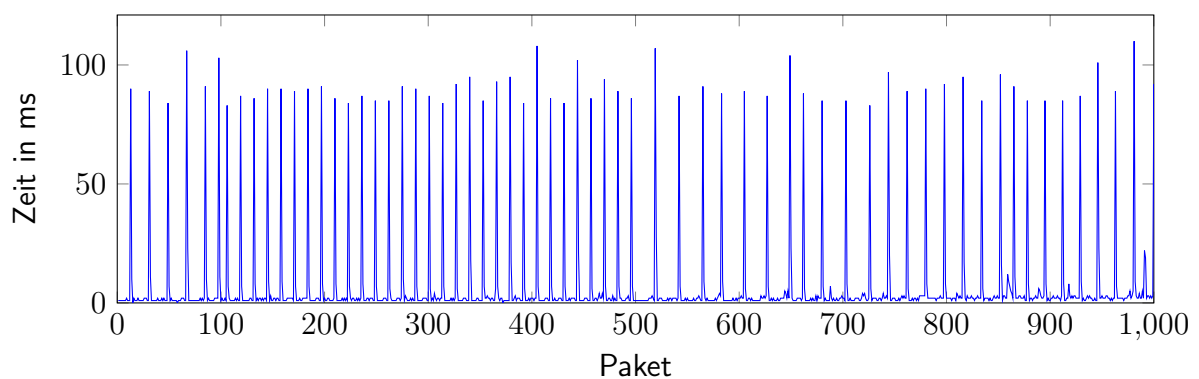


Abbildung 6.14.: Verzögerung einzelner Datenpakete für Monsun Nr.4 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

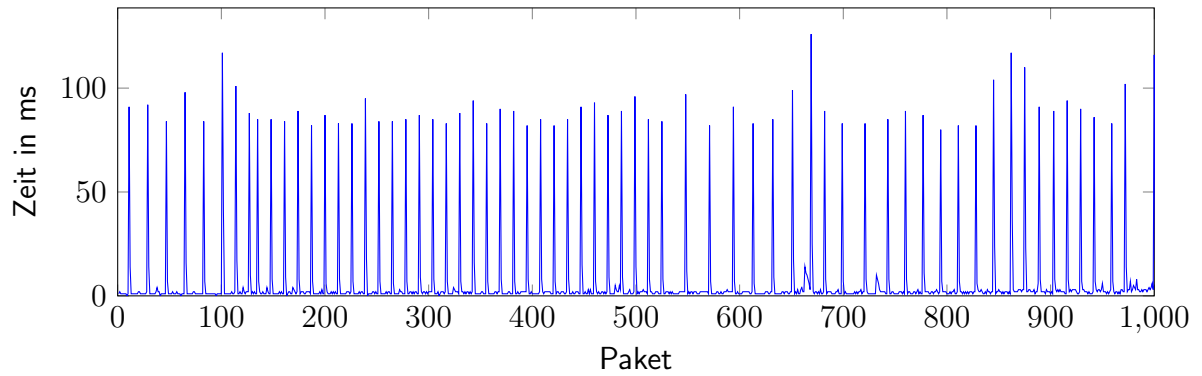


Abbildung 6.15.: Verzögerung einzelner Datenpakete für Monsun Nr.5 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

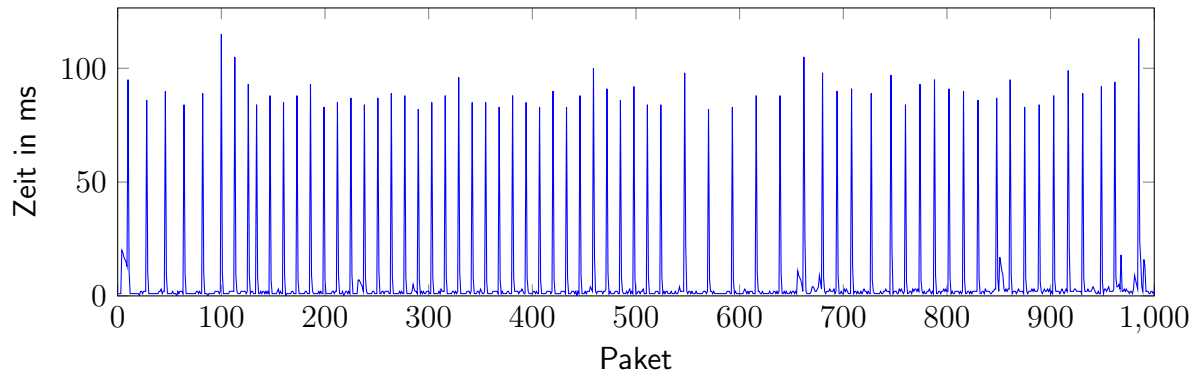


Abbildung 6.16.: Verzögerung einzelner Datenpakete für Monsun Nr.6 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

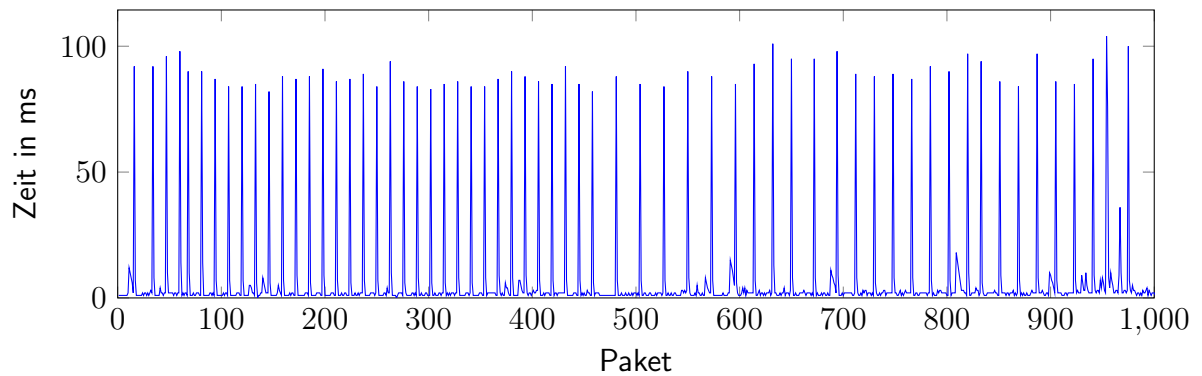


Abbildung 6.17.: Verzögerung einzelner Datenpakete für Monsun Nr.7 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

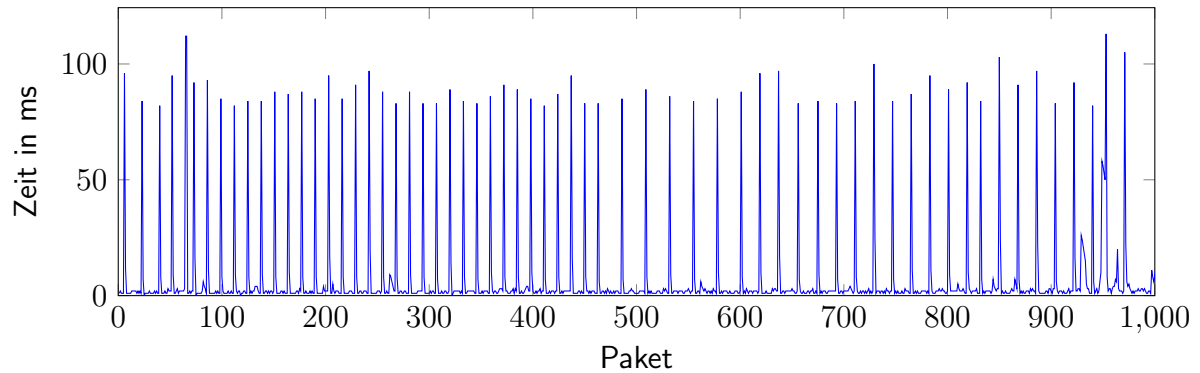


Abbildung 6.18.: Verzögerung einzelner Datenpakete für Monsun Nr.8 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

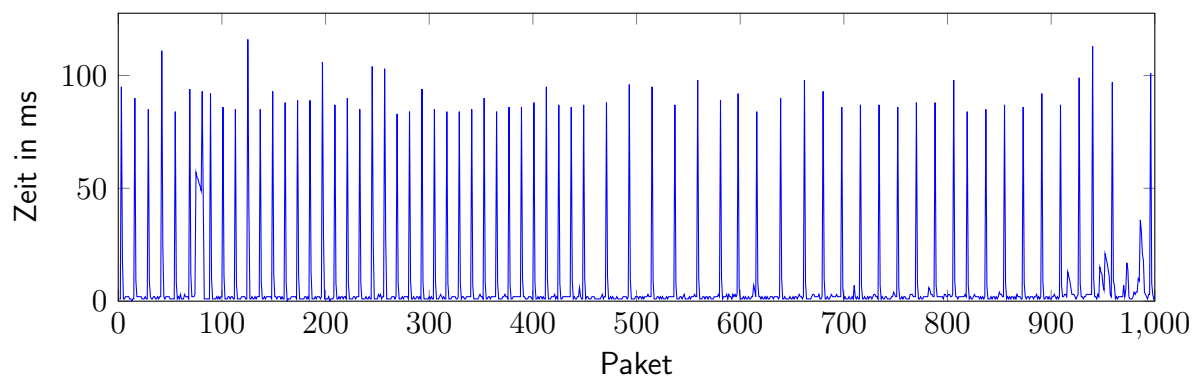


Abbildung 6.19.: Verzögerung einzelner Datenpakete für Monsun Nr.9 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.

6.1.5. Fazit

Die Evaluation des grafischen Benchmark zeigt, dass bereits ein schwaches System wie der Laptop die Grafik flüssig darstellen kann. Mit einem handelsüblichen Multimedia-PC und einer Mittelklasse-Grafikkarte sind, aufgrund der Leistungsreserven, mehr grafische Effekte und höhere Auflösungen möglich. Durch den Einsatz von Geometry- und Tessellationshadern beim Vegetationssystem könnte eine höhere grafische Güte und Performanz realisiert werden.

Die AUV-Simulation zeigt, dass auf dem PC-System ein komplettes realistisches Schwarmzenario mit 5 AUVs flüssig mit 69,1 FpS berechnet und dargestellt werden kann. Selbst mit 9 AUVs ist eine flüssige Darstellung von über 24 FpS gewährleistet. Es zeigt sich, dass genug CPU- und GPU-Ressourcen zur Verfügung stehen, um mehr AUVs zu simulieren. Bei der momentanen System-Konfiguration könnte die Anzahl wahrscheinlich verdoppelt werden, vorausgesetzt entsprechende Verbesserungen würden umgesetzt. Dafür müsste mehr Threading eingesetzt werden. Momentan sind einige Komponenten nicht ins Threading einbezogen, die durchaus ausgelagert werden könnten, und eine zusätzliche Aufteilung auf weitere *ApplicationStates* der jMonkeyEngine würde das potentielle Threading vereinfachen. Während der Tests blieb die Framerate stabil. Neun AUVs bilden die Grenze für eine flüssige Darstellung von über 24 Fps auf dem PC-System, wobei die momentan simulierten Szenarien mindestens fünf AUVs benötigen. Dies ist bedingt aufgrund des Konzepts, GPS Signale von der Wasseroberfläche über mindestens drei Stationen nach unten weiter zu leiten. Dadurch können sich die AUVs unter Wasser lokalisieren [OMAM12]. Benötigte Simulationsarbeiten in diesem Bereich wurden ebenfalls mit fünf AUVs durchgeführt (siehe Abschnitt 6.3).

Die Kommunikation zwischen dem Simulator MARS und der AUV-Steuerungssoftware ist, mit Einschränkungen, als echtzeitfähig anzusehen. Der Test ohne Frontkamera ergab eine durchschnittliche Verzögerung von 0,9 ms mit einer Standardabweichung von 0,49 ms. Lediglich wenn die Frontkamera mit ihrer großen Datenmenge zum Einsatz kommt, steigt die durchschnittliche Verzögerung auf 4,75 ms. Wobei die Datenpakete der Frontkamera selbst dafür hauptverantwortlich sind. Diese Daten sind bedingt echtzeitfähig, wobei lange Rechenzeiten in der Bildverarbeitung eine Echtzeitfähigkeit innerhalb der AUV-Steuerungssoftware an sich erschweren. Die Kompression der Daten verringert das Datenvolumen erheblich, allerdings steigt dadurch die Verzögerung einzelner Datenpakete in den Bereich von 600 ms. Somit ist der Einsatz von Kompression nicht echtzeitfähig. Dies könnte durch eine schnellere Komprimierung verbessert werden. Der Einsatz von neun zeitgleichen AUVs hatte keine Auswirkung auf die Verzögerung. Somit kann ein Szenario, wie in Abschnitt 6.3 beschrieben, ohne Beeinträchtigungen der Echtzeitfähigkeit simuliert werden.

Gesamt betrachtet, lassen sich große Szenarien mit neun Teilnehmern flüssig simulieren. Durch freie Rechenkapazitäten sind einige Verbesserungen möglich um weitere AUVs zu simulieren.

6.2. Vergleich zwischen Simulation und Realität

In diesem Abschnitt findet ein Vergleich zwischen Simulation und Realität statt. In Unterabschnitt 6.2.1 wird eine quantitative Evaluation der Tiefenregelung vom AUV HANSE durchgeführt. Unterabschnitt 6.2.2 beschäftigt sich mit einer qualitativen Analyse des Sonarmodels.

6.2.1. Tiefenregelung

Um die Simulation hinsichtlich ihrer Genauigkeit mit der Realität zu vergleichen, bietet sich die Tiefenregelung an. Bei der Tiefenregelung wird dem AUV eine Tiefe vorgegeben, die es unter Zuhilfenahme seiner Aktoren erreichen und halten muss. Tiefenregelung ist ein typisches und ständig aktives Verhalten, das alle AUVs benötigen, um ihre Position zu halten. In dem folgenden Unterkapitel wurde ein Vergleich der Tiefenregler von HANSE in der Realität und Simulation durchgeführt. Die Tiefenregelung findet mit den Reglern des respektiven AUV statt. Es handelt sich um einen PID-Regler. Die simulierten Tiefenfunktionen wurden über den *Root Mean Square Error (RMSE)* mit der realen Tiefenfunktion verglichen. RMSE bestraft Ausreißer stärker als andere Verfahren. Ergänzend wurde der *Mean Absolute Error (MAE)* berechnet. Die nachfolgenden Unterabschnitte erläutern die Ergebnisse der Tiefenregelungen vom HANSE AUV.

HANSE

HANSE verwendet einen PID-Regler für die Tiefenregelung. Der P-Wert wurde experimentell auf fünf bestimmt. Der I- und D-Anteil waren ausgeschaltet, und somit wird der Regler als P-Regler verwendet. Die Solltiefe betrug 0,45 cm. Der reale Versuch wurde im Rahmen der SAUC-E Vorbereitungen in einem kleinen Schwimmbecken durchgeführt. Diese Daten werden im Folgenden verwendet. Der Regler lief im realen und simulierten Versuch auf dem Laptop von HANSE. Derselbe Laptop wurde über Netzwerk mit dem Simulationsrechner verbunden. Die Masse von HANSE betrug zum Versuchszeitpunkt 19,5 Kg. Der Drucksensor hatte eine Aktualisierungsrate von 1 Hz. Diese Parameter wurden in der Simulation verwendet. Als Strömungswiderstandskoeffizient wurde aufgrund der kubischen Form von HANSE ein C_w von 1,10 verwendet. Im Versuchsablauf startet HANSE von der Oberfläche aus und strebt seine Solltiefe an. Diese wird für 60 Sekunden gehalten, bis HANSE wieder auftaucht.

Abbildung 6.20 stellt die simulierte und reale Tiefenregelung des HANSE AUV dar. Tabelle 6.8 und 6.9 zeigen den *Root Mean Square Error (RMSE)* und *Mean Absolute Error (MAE)* für die gesamte Tiefenregelung und für den Zeitabschnitt von 10 bis 60 Sekunden. Die Tiefendaten wurden mit einer Auflösung von 0,01 cm linear interpoliert, um die Differenz bilden zu können. Man sieht deutlich, wie der reale HANSE seine Solltiefe erreicht und einmal mit 10 cm überschwingt. Die restliche Zeit über bleibt er stabil auf seiner Solltiefe. Die Simulation mit dem gleichen P-Wert von 5 (rot) kommt nicht ganz

Fehler	Wert
$RMSE_{P5}$	0.037
$RMSE_{P6}$	0.042
$RMSE_{P5,t=10-60}$	0.027
$RMSE_{P6,t=10-60}$	0.019

Tabelle 6.8.: Der RMSE der simulierten HANSE-Tiefenregelung für verschiedene P-Werte und für die Zeit von 10 bis 60 Sekunden.

Fehler	Wert
MAE_{P5}	0.029
MAE_{P6}	0.026
$MAE_{P5,t=10-60}$	0.024
$MAE_{P6,t=10-60}$	0.014

Tabelle 6.9.: Der MAE der simulierten HANSE-Tiefenregelung für verschiedene P-Werte und für die Zeit von 10 bis 60 Sekunden.

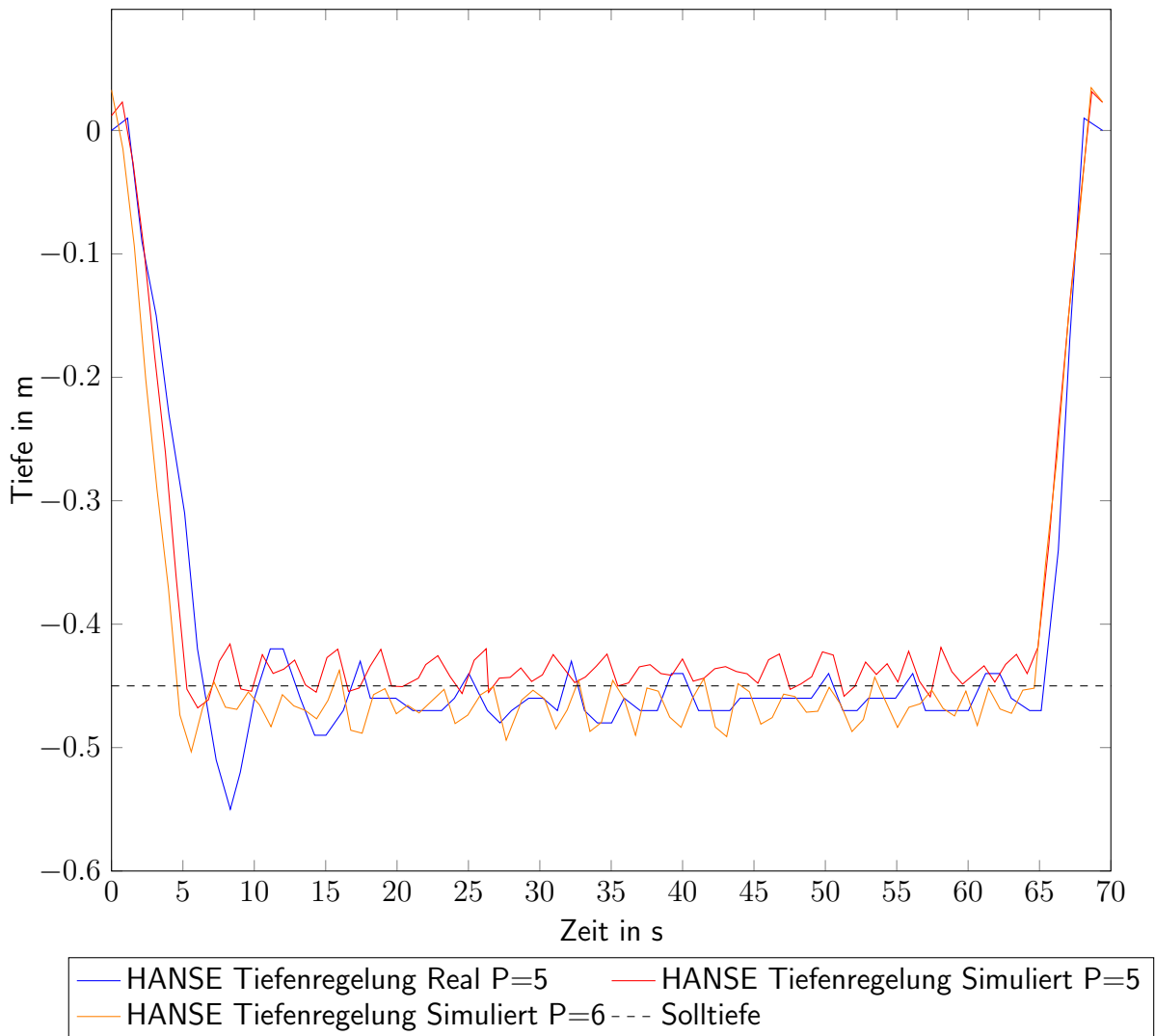


Abbildung 6.20.: Die simulierte (rot und orange) und reale (blau) Tiefenregelung des HANSE-AUV.

an die Realität heran. Mit einer Erhöhung des P-Werts auf 6 nähern sich die Funktionen gut an. Betrachtet man den RMSE über die Gesamtzeit, so ist die rote Funktion um ca. 0.0053 besser als die orange Funktion. Hauptverantwortlich ist dafür das steile Abtauchen zu Anfang. Wenn man sich den RMSE im Bereich von 10 bis 60 Sekunden anschaut, so ist die gelbe Funktion um ca. 0,0082 besser. Die insgesamt besseren Werte der gelben Kurve zeigen sich, wenn der MAE errechnet wird. Das einmalige Abtauchen fällt weniger stark ins Gewicht.

Bereits eine direkte Übernahme der realen Reglerparameter ermöglicht eine nachvollziehbare Simulation. Durch kleinere Anpassungen der Regler werden die Ergebnisse weiter verbessert. Es fehlt allerdings das starke Überschwingen, das in der Realität vorkommt. Dies hängt vermutlich mit dem einfachen Thrustermodell zusammen (siehe Abschnitt 4.3.1). Dieses ist nur ein Steady-State Modell, in dem unterschiedliche Drehrichtungen des Thrusters dieselben absoluten Ergebnisse liefern. Außerdem gibt es keine zeitliche Verzögerung wie bei Modellen mit höherer Ordnung.

6.2.2. Sonar

Das in Abschnitt 4.2.2 aufgestellte Modell für das Sonar wird in diesem Abschnitt evaluiert. Das reale und simulierte Sonar wurde im selben Szenario getestet wie in Abschnitt 6.1.3 beschrieben (siehe Abbildung 6.5). Beim verwendeten realen Sonar handelt es sich um das *Imagenex 852 ultra-miniature scanning sonar*, das in HANSE integriert ist[ima]. Abbildung 6.21 zeigt das resultierende Sonarbild der verschiedenen Strahlenmuster ohne Rauschen. In a) wird ein einzelner Strahl ausgesendet. Die Umgebungsstruktur ist klar zu erkennen, ebenfalls der Poller. Allerdings ist es möglich, dass kleinere Objekte wie Bojen oder Fische oder Objekte über und unterhalb des zentralen Strahls nicht detektiert werden. Dies ist im Verfahren selbst begründet, da ein einziger Strahl nicht ausreicht, um den gesamten Bereich abzudecken. Lösungen für die Probleme sind das Erhöhen der Samplingzahl, das Ignorieren oder Verbieten von kleinen Objekten oder einen erneuten Durchlauf des Sonar abzuwarten, bis das Objekt zufällig detektiert wird. In b) wurde die Strahlenzahl erhöht und in einem Kreismuster ausgesendet mit einem Öffnungswinkel von 22° . Die Addition von mehreren Intensitäten durch die Vielzahl an Kontaktpunkten resultiert in einem deutlicheren Bild im Vergleich zu a). Allerdings tauchen im Bild Kreisstrukturen auf, die vom Boden herrühren aufgrund des großen Öffnungswinkels. In c) wird ein schmaler Öffnungswinkel von 2.5° mit 64 Strahlen in einem rechteckigen Muster, ähnlich zum Imagenex Sonar, verwendet. Das Bild ähnelt stärker bekannten Sonarbildern. Ein einziger deutlicher Bodenkreis ist zu sehen.

Abbildung 6.22 illustriert den Einsatz von 4 Dämpfungen auf das Sonarbild in Abbildung 6.21 c). In a) wird keine Dämpfung verwendet. Die Umgebung wird klar und deutlich dargestellt, allerdings ist der Bodenkreis deutlich sichtbar. In b) wird eine Distanzdämpfung verwendet. Je weiter entfernt ein Kontaktpunkt ist, desto schwächer ist sein Gesamteinfluss auf das aufsummierte Sonararray. Der Bodenkreis ist abgeschwächt, aber deutlich zu sehen, ebenfalls die Umgebung. In c) wird die Winkeldämpfung eingesetzt. Der Winkel zwi-

schen Auftrittsstrahl und Normale bestimmt die Dämpfung. Dies simuliert die Streuung, die bei Schallwellen bei schiefen Oberflächen eintritt. Der Bodenkreis ist deutlich abgeschwächt, was sich aufgrund des flachen Auftrittswinkel ergibt. Die restliche Umgebung ist deutlich zu sehen. In d) wird eine Kombination von Distanz- und Winkeldämpfung eingesetzt. Der Bodenkreis ist nicht mehr zu sehen, und es ist ein klares leicht abgeschwächtes Sonarbild zu sehen.

Abbildung 6.23 zeigt einen Vergleich zwischen realem Sonar (links) und Simulation (rechts). Das Rauschen des echten Sonar wurde gemessen und zum simulierten Sonarbild hinzu addiert. Zusätzliches Rauschen für Distanz und Winkeldämpfung ist hinzugefügt. Im simulierten Bild sind die Umgebungsstrukturen gut erkennbar, z.B. die Ecke. Einige Unterschiede zum realen Sonarbild, wie z.B. die Verzerrung der Umgebung, erklären sich dadurch, dass es für das AUV HANSE, an dem das Sonar befestigt war, schwierig war, seine Position im Wasser zu halten.

Insgesamt liefert das Sonarmodell ein brauchbares, an die Realität angelehntes Sonarbild auf Strahlenbasis. Es ist flexibel genug, um eine Vielzahl an Sonaren und deren Parameter, oder auch statische Echos, zu simulieren. Des weiteren ist es performant genug, um mit der Anzahl an Strahlen umzugehen. Für Verbesserungen beim Rauschen sind unter Umständen individuelle Messungen von weiteren Sonaren notwendig. Der Realitätsgrad kann durch weitere Strahlen, die ähnlich zur Mehrpfadberechnung des Kommunikationsmodells in Abschnitt 4.6.1 funktionieren, gesteigert werden. Geschwindigkeitsverbesserungen sind durch den Einsatz von Depth-Buffern der Grafikkarte möglich.

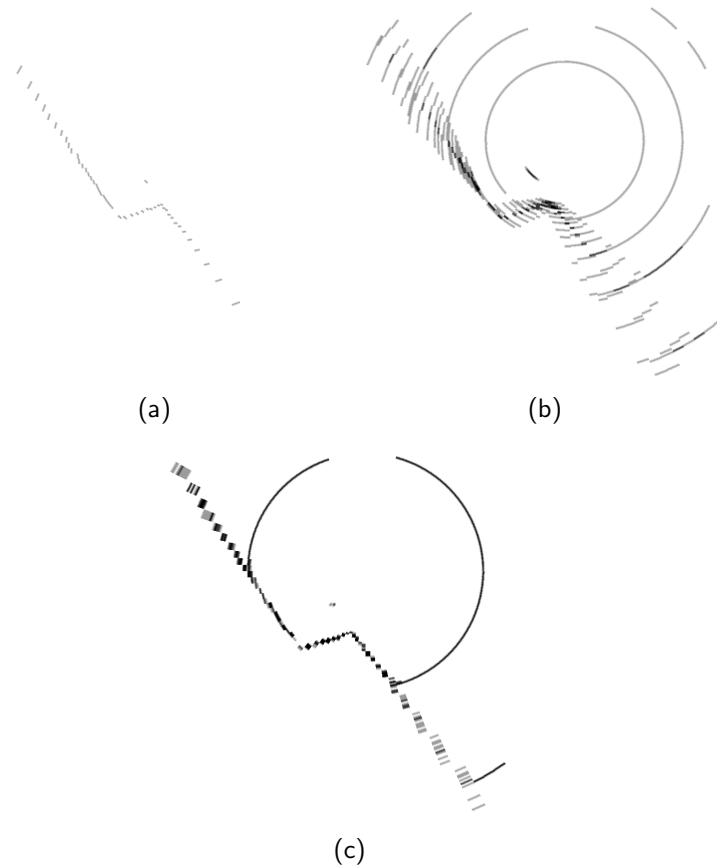


Abbildung 6.21.: Drei verschiedene Strahlenmuster für das simulierte Sonar. a) einzelner Strahl. b) mehrere Strahlen in einem zirkulären Muster und einem Öffnungswinkel von 22° . c) mehrere Strahlen in einem rechteckigen Muster und einem Öffnungswinkel von 2.5° [TM14].

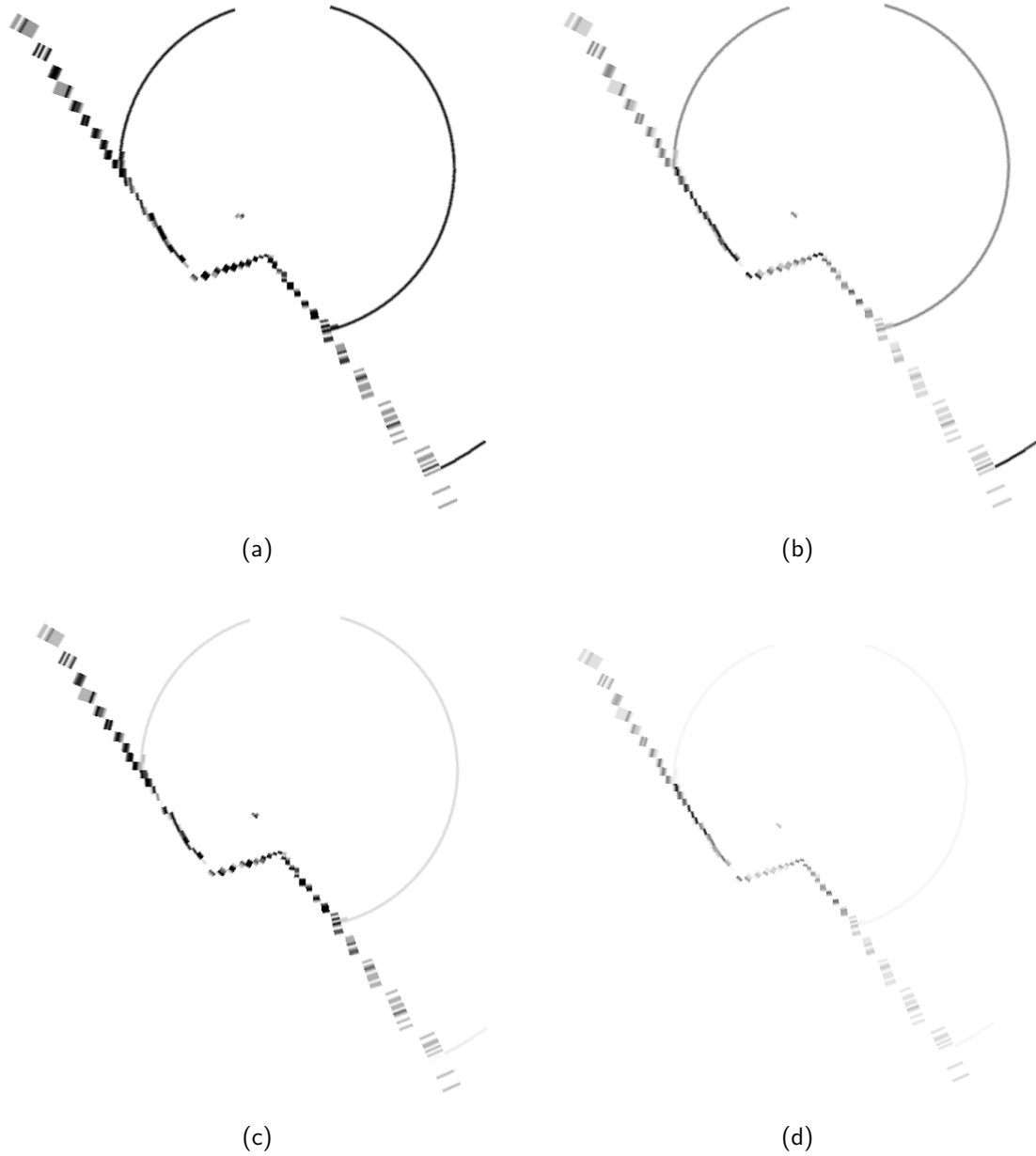


Abbildung 6.22.: Verschiedene Dämpfungen für das simulierte Sonar. a) Keine Dämpfung. b) Längendämpfung. c) Winkeldämpfung. d) Kombination aus Längen- und Winkeldämpfung [TM14].

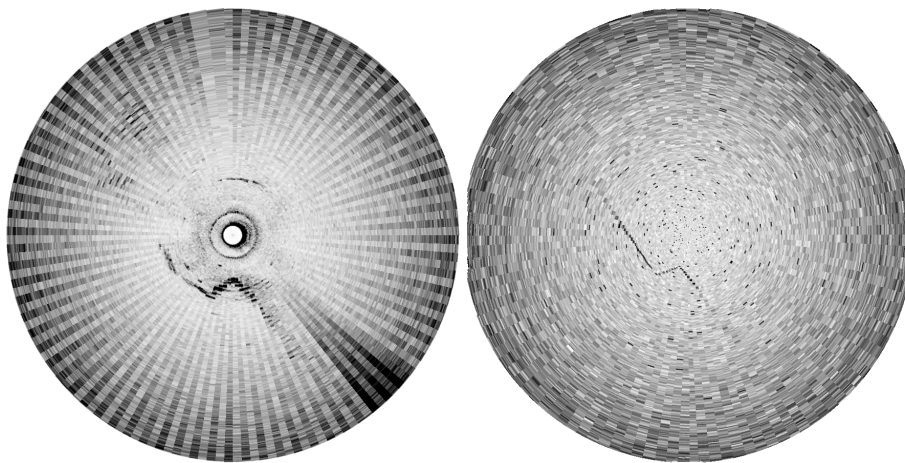


Abbildung 6.23.: Vergleich eines realen Sonarbildes (links) und des simulierten (rechts) [TM14].

6.3. Schwärme

Um die Schwarmfähigkeit des Modells und der Simulation zu zeigen, sind entsprechende Tests notwendig. Im Rahmen des MARS Projekts wurden zwei Schwarmverhalten basierend auf dem MONSUN AUV (siehe Kapitel 3.1.2) erstellt. Diese sollen im Detail vorgestellt und die Ergebnisse besprochen werden. Wichtige Bereiche sind die Simulation von mehreren AUVs, Kommunikation, Stromverbrauch und Auslösen von Fehlern. Die Schwarmverhalten wurden in Kooperation mit Ammar Amory entwickelt, wobei Herr Amory einen Großteil der Implementierung übernommen hat.

6.3.1. Formationsfahrt

In [AMO13] wurde gemeinsam mit Ammar Amory eine V-Formationsfahrt basierend auf fünf MONSUN AUV implementiert und simuliert. Das Verhalten läuft folgendermaßen ab. Fünf MONSUN starten zufällig positioniert an der Wasseroberfläche. Über W-Lan kommunizieren die AUVs untereinander und geben ihre Pose weiter. Alle MONSUN haben eine eindeutige ID und der mit der höchsten wird als Anführer bestimmt. Daraufhin führt der Anführer den Schwarm vom Startbereich zu einem Zielpunkt. Während der Fahrt befinden sich zwei der AUV unter Wasser, um die Umgebung zu untersuchen. Die restlichen drei schwimmen an der Wasseroberfläche als Bezugspunkt.

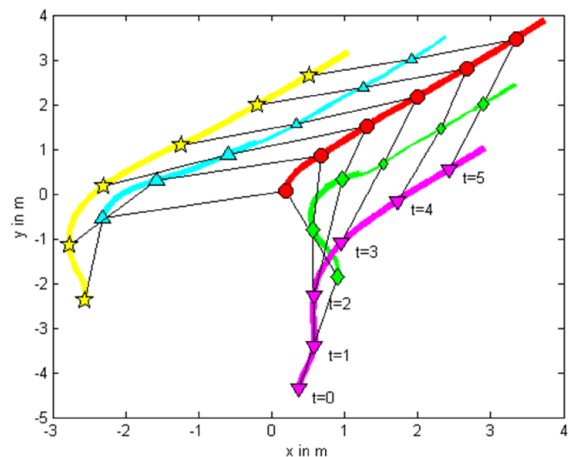
Kommuniziert wird an der Wasseroberfläche über W-Lan, während Unterwasser ein akustisches Modem zum Einsatz kommt. Der Anführer sendet während der Fahrt Zielpositionen an die anderen Schwarmmitglieder und diese versuchen die Orientierung und Geschwindigkeit, über PID-Regler, anzupassen, um ihre relative Position innerhalb des Schwarms beizubehalten. Die Mitglieder benötigen für die Formationsfahrt die relative Position und Orientierung ihres ihm zugeteilten Vorgängers. Dies führt sich fort bis zur Spitze der V-Formation, dem Anführer. Ein Mitglied kommuniziert nur mit seinem Nachfolger und dem Anführer. Durch Aufrechterhaltung (Keepalive) wird sicher gestellt, dass der Schwarm zusammenbleibt. Der Anführer sendet diese periodisch und bei Zeitüberschreitung wird das Mitglied als verloren angesehen und die Formation umgestellt.

Um eine relative Positionsbestimmung der untergetauchten MONSUN umzusetzen, müssen mindestens drei MONSUN an der Wasseroberfläche schwimmen [BLM12]. Diese können über GPS ihre absolute Position feststellen. Dies führt zu zwei Schichten, in die der Schwarm aufgeteilt wird. Die obere Schicht dient als Bezugspunkt für die untere Schicht, während die untere Schicht untergetaucht die eigentliche Aufgabe der Mission ausführt, wie z.B. Kartografierung oder Umweltdatenerfassung.

In Abbildung 6.24 wird das Verhalten gezeigt. Abbildung 6.24 a) zeigt den Schwarm innerhalb der Simulation bereits als V-Formation. Zwei der Mitglieder sind untergetaucht. Abbildung 6.24 b) zeigt die Position der einzelnen Schwarmmitglieder zu fest gesetzten Zeitpunkten. In der Anfangsphase bis zum Zeitpunkt $t = 3$ bilden sich die V-Formation. Daraufhin führt der Anführer (rot) den Schwarm zum Zielpunkt bei $t = 5$. Während der Fahrt tauchen die AUV grün und türkis ab. Dies wird gekennzeichnet durch die Dicke der



(a) V-Formation bestehend aus fünf MONSUN. Zwei davon sind bereits untergetaucht.



(b) Ergebnis der Formationsbildung und V-Fahrt. Die Dicke der Linie gibt die Tiefe an. Eine dünnere bedeutet dass das AUV untertaucht.

Abbildung 6.24.: V-Formationsfahrt aus [AMO13].

Linie. Eine dünnere bedeutet das eine größere Tiefe erreicht wird.

Die Simulation der Verhaltens erfolgte in MARS, und es konnte erfolgreich gezeigt werden, dass ein Schwarm aus fünf Mitgliedern kommunizieren, seine Formation aufstellen und sein Ziel erreichen kann.

6.3.2. Load-Balancing Verhalten

In [ATM14] wurde gemeinsam mit Ammar Amory ein Load-Balancing Verhalten implementiert und simuliert, basierend auf fünf MONSUN AUVs. Aufgrund der Aufteilung des Schwarms in Schichten [AMO13] ergibt sich ein Energiekapazitätsunterschied zwischen diesen. Die MONSUN AUVs haben einen positiven Auftrieb, damit sie bei Problemen leichter geborgen werden können, und müssen sich aktiv mit Thrustern nach unten drücken. Die untere Schicht benötigt außerdem mehr Sensorik, um ihre Aufgabe, wie z.B. Umweltmonitoring, unter Wasser auszuführen. Da die AUVs homogen sind, ist ein einfacher Austausch möglich, um die Missionszeit zu verlängern.

Die Mission verläuft in drei Phasen. In Phase 1 wird eine V-Formation nach dem Leaders-Follower Prinzip gebildet [FM02]. Der Anführer führt den Schwarm zu einem Zielpunkt. In Phase 2 tauchen zwei AUVs ab, um den Unterwasserbereich genauer zu untersuchen. In Phase 3 erfolgt dann ein Austausch eines AUVs aus der unteren Schicht mit einem AUV aus der oberen Schicht.

Die AUVs produzieren Gesundheitssignale (health signals), die eine Aussage über die allgemeine Fitness des Roboters darstellen [MM12]. Diese Nachrichten werden an den Schwarm gesendet. Wenn während der Fahrt die Fitness eines AUVs zu niedrig ausfällt, z.B. wenn zu wenig Energie vorhanden ist, so wird ein Austausch vom Anführer eingeleitet. Welches

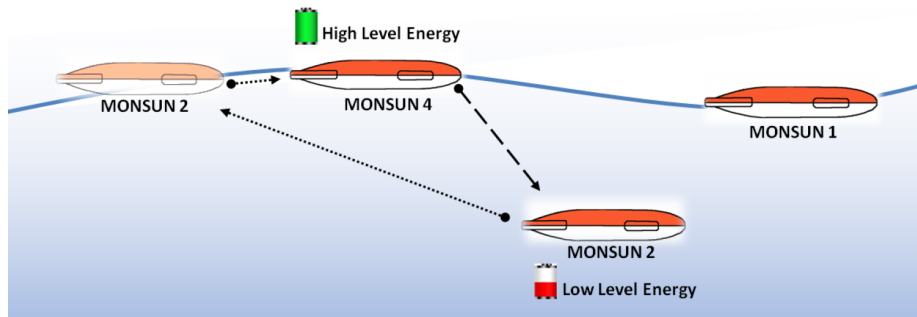


Abbildung 6.25.: Austausch eines MONSUN mit weniger Energie in der unteren Schicht mit einem aus der oberen Schicht [ATM14].

AUV zum Austausch eingesetzt wird, hängt von Kriterien wie Energiekapazität und Nachbarschaft ab. Zur Feststellung des Energieverbrauchs wurde eine Liste der Verbraucher und ihres typischen Verbrauchs aufgestellt, basierend auf Datenblättern. Für die Thruster erfolgte eine eigene Messung von Kraft und Stromverbrauch, die in die Simulation eingegeben wurde.

In Abbildung 6.25 sieht man einen beispielhaften Austausch. MONSUN 2 schwimmt in der unteren Schicht und hat eine geringe Energiekapazität. MONSUN 4 schwimmt in der oberen Schicht und hat eine hohe Energiekapazität. MONSUN 2 leitet den Austausch ein und lässt sich aufgrund des positiven Auftriebs an die Wasseroberfläche steigen. Wenn MONSUN 2 oben angekommen ist, baut er eine GPS Verbindung auf. Daraufhin kann MONSUN 4 die alte Position von MONSUN 2 einnehmen.

In Abbildung 6.26 ist die komplette Mission, bestehend aus fünf AUVs und einem Austausch zu sehen. Bis Phase 1 wird die V-Formation an der Wasseroberfläche gebildet. Bei Phase 2 folgt das Abtauchen von zwei AUVs. In Phase 3 sieht man den erfolgreichen Austausch vom blauen AUV mit wenig Energie zum pinken mit viel Energie. Durch diese Verhalten sind Missionszeitverlängerungen von bis zu 32 % möglich.

Durch Abschalten von Sensoren ließen sich falsche Daten erzeugen, mit denen das AUV und der Schwarm umgehen mussten. Der Drucksensor lieferte über einen Zeitraum von 20 Sekunden Daten, die außerhalb seines normalen Wertebereichs lagen. Dieses Verhalten führte zu einem zum Teil fehlerhaften Systemzustand des Roboters mit niedriger Funktionalität (fitness) und zu einem Austausch des Roboters mit einem funktionstüchtigeren von der Oberfläche.

In der Simulation konnte die Machbarkeit des Verhaltens gezeigt werden und dass eine Verlängerung der Missionszeit möglich ist. Die Kommunikation und der Energieverbrauch wurden dabei verwendet.

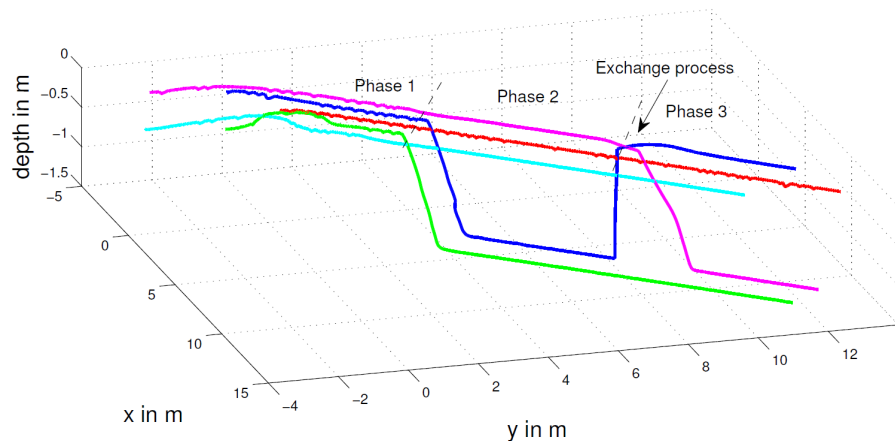


Abbildung 6.26.: Energy-Load Balancing Verhalten bestehend aus 3 Phasen. Der Austausch findet in Phase 3 statt [ATM14].

6.4. Benutzungsschnittstelle

Um die Benutzungsschnittstelle zu evaluieren, wurde eine benutzerorientierte Evaluation nach [Her05] durchgeführt. In diesem Verfahren wird der Benutzer in die Evaluation mit eingebunden und soll Aufschluss zur Gebrauchstauglichkeit der Anwendung geben. Für eine benutzerorientierte Evaluation benötigt man eine ausreichend große Anzahl an Probanden. Jakob Nielsen hat festgestellt, dass bei einer qualitativen Studie für Gebrauchstauglichkeit bereits 5 Probanden ausreichen [Nie00]. Die geringe Anzahl erklärt sich durch die Überdeckung der gefundenen Probleme. Allerdings sind für quantitative Aussagen 15 bis 20 Probanden notwendig [Nie06].

6.4.1. Durchführung

Der Proband soll sich mit der Simulationsumgebung MARS beschäftigen. Diese ist bereits auf einem PC installiert und gestartet. Er bekommt eine Anleitung, die er vorher durchliest. In dieser Anleitung stehen verschiedene Aufgaben, die er bewältigen muss. Nach den Aufgaben gibt es eine freie Phase, wo der Proband Verschiedenes ausprobieren kann. Die verwendete Anleitung befindet sich im Anhang A.2. Die zur Verfügung gestellte Zeit beträgt 20 Minuten. Bei Fragen konnte sich der Proband an den Versuchsleiter wenden. Nach dem Ausprobieren erfolgt eine Rückmeldung des Probanden über einen Fragebogen. Der Fragebogen orientiert sich an der ISO Norm 9241-110 [iso06] und ist in die 7 Bereiche der Gebrauchstauglichkeit aufgeteilt und einen allgemeinen Bereich für Gesamtbewertung und Erfahrungsstand. Während und nach dem Test werden Kommentare der Probanden notiert. Der verwendete Fragebogen ist im Anhang A.1 hinterlegt.

Bewertung	—	-	-	-/+	+	++	+++
Farbkodierung							
Skalar	-3	-2	-1	0	1	2	3

Tabelle 6.10.: Farbkodierung der Bewertungsskala des Fragebogens. -3 ist eine sehr schlechte Bewertung, 0 eine neutrale und +3 eine sehr gute.

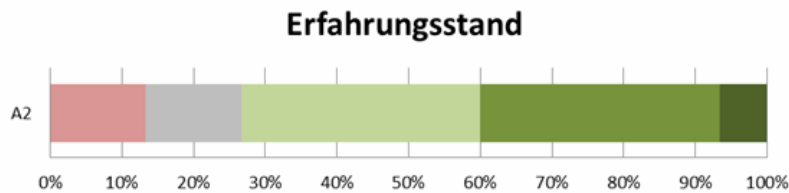


Abbildung 6.27.: Erfahrungsstand der Probanden zur Benutzung allgemeiner Simulationssoftware.

6.4.2. Auswertung

An der Evaluation haben insgesamt 15 Probanden teilgenommen. Davon waren 100 % männlich. Das Durchschnittsalter betrug 26,8 Jahre. Die Probanden waren Studenten die in verschiedenen Projekten am Institut für Technische Informatik tätig waren, wie z.B. dem MARS-Projekt. Weitere Probanden bestanden aus Mitarbeitern des Instituts. In Tabelle 6.10 steht die in den Evaluationsgrafiken verwendete Farbkodierung für die Bewertungsskala des Fragebogens. Abbildung 6.27 zeigt die Verteilung des Erfahrungsstandes der Probanden mit allgemeiner Simulationssoftware. Über 70 % der Probanden haben mit Simulationssoftware gearbeitet. Nachfolgend werden die Ergebnisse der 7 Hauptbereiche diskutiert, Nutzerkommentare erörtert und abschließend das Gesamtfazit diskutiert.

Aufgabenangemessenheit

Abbildung 6.28 zeigt die Aufteilung der Bewertungen für die Aufgabenangemessenheit, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die durchschnittliche Bewertung mit einem Wert von 1,653 zeigt eine gute Aufgabenangemessenheit. Lediglich der Punkte A3, bei dem es um Automatisierung von wiederholenden Bearbeitungsvorgänge geht, schneidet mit einer durchschnittlichen Bewertung von 0,53 schlechter ab als die anderen Punkte. Dies tritt z.B. auf, wenn man mehrere AUVs gleichzeitig verschieben möchte. Momentan muss man jedes einzeln verschieben. Ein Selektionsmechanismus mit Gruppierungsfunktionen würde hier Abhilfe schaffen. Außerdem wäre eine Makrofunktion nützlich, mit der die Benutzer selbst Automatisierungsfunktionen schreiben könnten.

Selbstbeschreibungsfähigkeit

Abbildung 6.29 zeigt die Aufteilung der Bewertungen für die Selbstbeschreibungsfähigkeit, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die Selbstbeschreibungsfähigkeit schneidet mit einer Gesamtbewertung von 0,693 am schlechtesten im Vergleich zu den anderen Fragegruppen ab. Insbesondere Frage A4 und A5 (Situationsspezifische Erklärungen, die weiter helfen auf Verlangen oder von sich aus) weisen ein schlechtes Durchschnittsergebnis auf von jeweils 0,2. Dies ergibt sich aufgrund der Komplexität des Modells, der daraus resultierenden Parameter und der Benutzungsschnittstelle, die versucht dies alles abzubilden. Die kontextsensitive Hilfe muss daher weiter ausgebaut werden.

Steuerbarkeit

Abbildung 6.30 zeigt die Aufteilung der Bewertungen für die Steuerbarkeit, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die Steuerbarkeit weist mit einer durchschnittlichen Bewertung von 2,413 ein sehr gutes Ergebnis auf. Lediglich der Punkt A2 (starre Einhaltung von Bearbeitungsschritten) fällt mit einer Wertung von 1,8 leicht zurück. Dies ist zum Teil zurückzuführen auf das nicht sofortige Übernehmen bestimmter Parameter, was den Benutzer zu einem gezwungen Neustarten der Konfiguration, seitens des Benutzers, führt. Abhilfe schafft das sofortige Übernehmen von Parametern.

Erwartungskonformität

Abbildung 6.31 zeigt die Aufteilung der Bewertungen für die Erwartungskonformität, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die Erwartungskonformität hat mit 1,96 eine sehr gute durchschnittliche Bewertung. Frage A2 (Unklarheit über erfolgreiche Eingaben) und A3 (unzureichende Informationen über das aktuelle Geschehen) weisen jeweils eine einzelne negative Bewertung auf. Diese können aufgrund der restlichen positiven Bewertungen vernachlässigt werden.

Fehlertoleranz

Abbildung 6.32 zeigt die Aufteilung der Bewertungen für die Fehlertoleranz, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die Fehlertoleranz ist mit einer Durchschnittsbewertung von 1,3 die zweitschlechteste Fragegruppe. Fragen A1(kleine Fehler führen zu schwerwiegenden Folgen) und A5 (konkrete Hinweise zur Fehlerbehebung) sind hierbei die Fragen mit der schlechtesten Bewertung von 1,13

und 0,73. Programmfehler, die unweigerlich bei einem komplexen Softwareprojekt entstehen, haben vermutlich Probanden beeinflusst. Um schwerwiegende Folgen zu verringern, wäre eine Undo-Funktion empfehlenswert oder die Einschränkung bestimmter Funktionen. Die Hilfefunktionen muss weiter ausgebaut werden. Das Problem besteht darin, zu Fehlern führendes Verhalten des Benutzers zu erkennen, um darauf mit Hilfe zu reagieren.

Individualisierbarkeit

Abbildung 6.33 zeigt die Aufteilung der Bewertungen für die Individualisierbarkeit, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Mit einer Durchschnittsbewertung von 1,92 ist die Individualisierbarkeit als sehr gut einzuschätzen. Frage A3 (ungeeignet für Experten und Anfänger) bildet eine negative Besonderheit mit einer Wertung von 0,73. Aufgrund der vielen Möglichkeiten und Parameter innerhalb der Benutzungsschnittstelle und dass es sich zusätzlich um ein dreidimensionales System handelt, sind Anfänger schneller überfordert. Abhilfe würde ein Anfänger-Modus schaffen, der Teile der Benutzungsschnittstelle verbirgt oder einschränkt.

Lernförderlichkeit

Abbildung 6.34 zeigt die Aufteilung der Bewertungen für die Lernförderlichkeit, gruppiert nach den Fragen (A1-A5) und der durchschnittlichen Gesamtbewertung des Bereichs (AG). Der blaue Strich gibt die durchschnittliche Bewertung innerhalb einer Frage an. Die Lernförderlichkeit ist mit einem Durchschnittswert von 1,46 als gut einzuschätzen. Frage A3 (viele Details zu merken) und A5 (schlecht ohne fremde Hilfe oder Handbuch bedienbar) fallen besonders auf mit einer Wertung von 1,26 und 0,6. Aufgrund der Komplexität des Modells sind viele Einstellungen möglich, was sich negativ auswirkt. Details könnten entweder verborgen werden oder über Automatisierung überflüssig gemacht werden. Eine kontextsensitive Hilfe und ein digitales Nachschlagewerk existieren nur rudimentär und müssen ausgebaut werden. Außerdem wäre ein Rundgang oder Tutorial hilfreich.

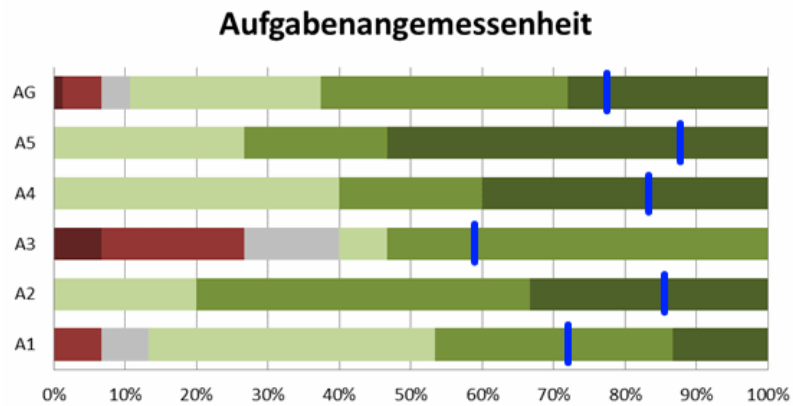


Abbildung 6.28.: Verteilung der Evaluationsergebnisse im Bereich Aufgabenangemessenheit.

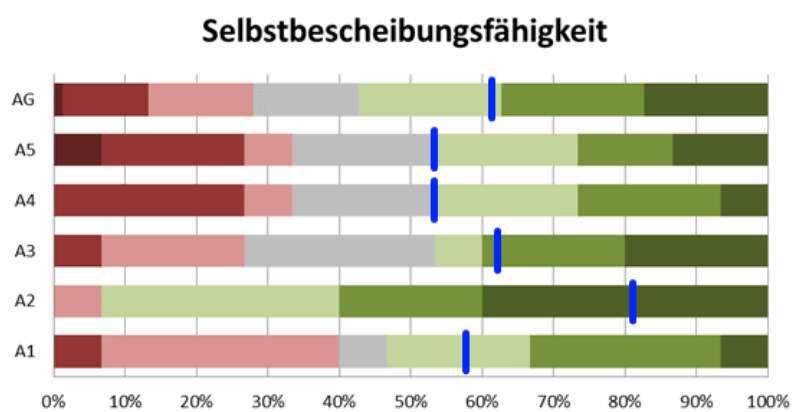


Abbildung 6.29.: Verteilung der Evaluationsergebnisse im Bereich Selbstbeschreibungsfähigkeit.

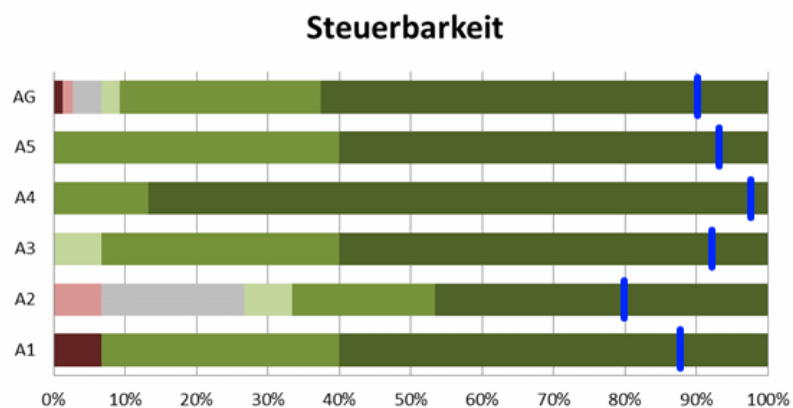


Abbildung 6.30.: Verteilung der Evaluationsergebnisse im Bereich Steuerbarkeit.

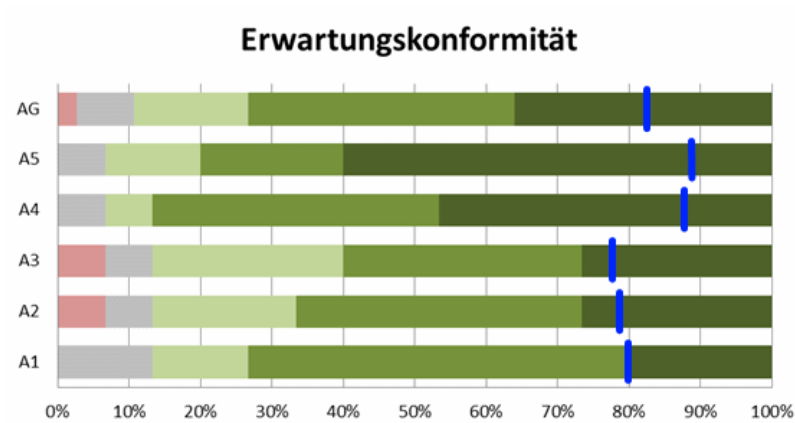


Abbildung 6.31.: Verteilung der Evaluationsergebnisse im Bereich Erwartungskonformität.

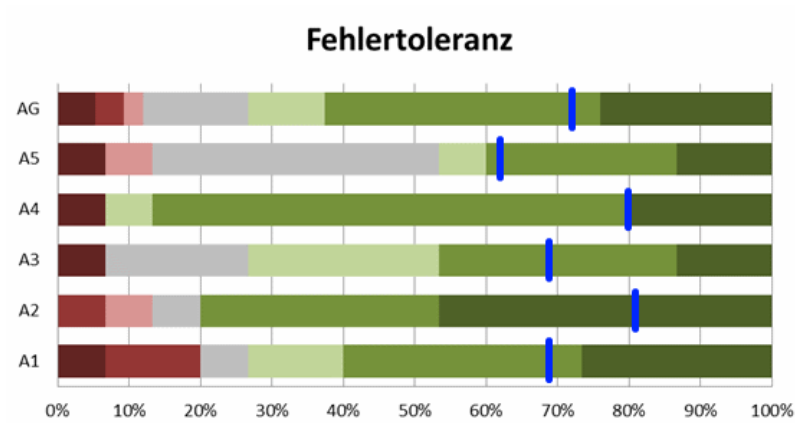


Abbildung 6.32.: Verteilung der Evaluationsergebnisse im Bereich Fehlertoleranz.

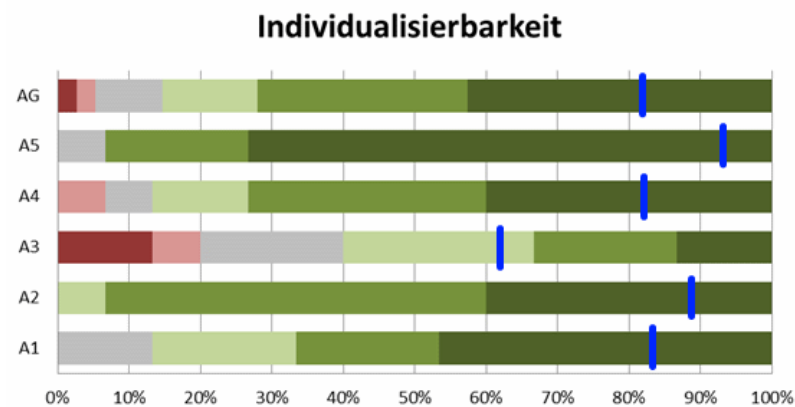


Abbildung 6.33.: Verteilung der Evaluationsergebnisse im Bereich Individualisierbarkeit.

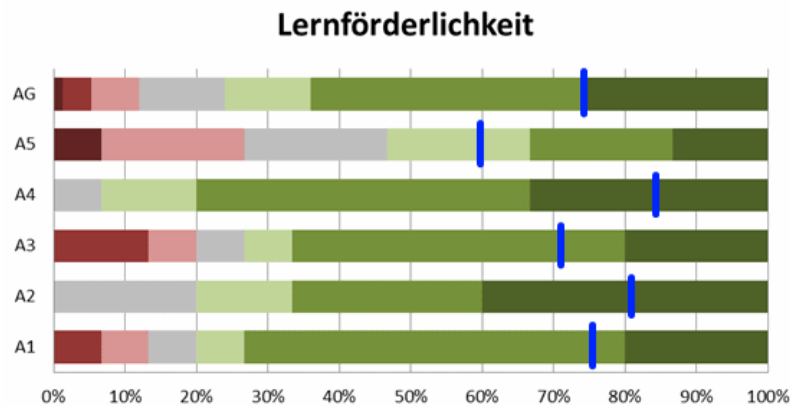


Abbildung 6.34.: Verteilung der Evaluationsergebnisse im Bereich Lernförderlichkeit.

Nutzerkommentare

Die Benutzer hatten die Möglichkeit, Kommentare während und am Ende der Evaluation abzugeben. In der nachfolgenden Aufzählung sind häufig vorkommende Verbesserungsvorschläge aufgelistet.

- Die Unterscheidung zwischen der Konfiguration und den Optionen ist nicht eindeutig.
- Es wurden mehr Buttons für einen schnelleren Zugriff gewünscht.
- Die Steuerung erschließt sich nicht sofort, ist aber verwendbar. Als Alternative wird ein achsenverschieben Werkzeug wie in der Blender 3D-Umgebung vorgeschlagen.
- Das Kopieren und Einfügen von Elementen wie AUVs oder Sensoren funktioniert nicht aus jeder Situation heraus.
- Nicht alle Änderungen in der Baumstruktur oder den Optionen werden sofort übernommen. Es ist zum Teil ein Neustart nötig.
- Im File Menü sind sich die Punkte *View* und *Window* zu ähnlich.
- Es sollte ein Kontextmenü angezeigt werden, wenn man in der Hauptansicht ins Leere klickt. Dort sollten sich Kamera-Optionen befinden.
- AUVs sollten sich auch über eine Drag-and-Drop Mechanik bewegen lassen können.
- Die Kamera sollte auch planar zur X-Y-Ebene bewegt werden können.
- Daten der Sensoren sollten in der Baumstruktur über einen Doppelklick aufgerufen werden können und nicht nur über das Kontextmenü.

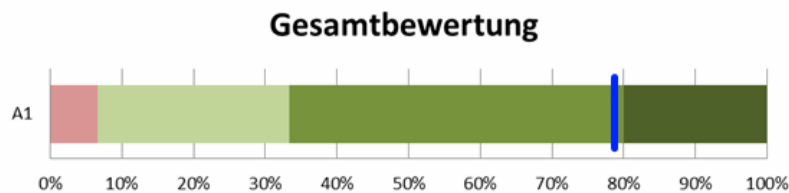


Abbildung 6.35.: Verteilung der Gesamtbewertung der Probanden zur Benutzungsschnittstelle von MARS.

Fazit

Abbildung 6.35 zeigt die Verteilung der Gesamtbewertung, mit der die Probanden die Benutzungsschnittstelle bewerten. Als blauer Strich dargestellt der Durchschnittswert. Aus den Ergebnissen lässt sich ableiten, dass die Benutzungsschnittstelle, trotz der Komplexität einer Simulationssoftware, mit einer Gesamtnote von 1,73 eine gute Gebrauchstauglichkeit aufweist. Es gab nur eine Bewertung im negativen Bereich und die restlichen sind positiv. Wenn man den Durchschnitt aller Bereiche bildet und diese gleich gewichtet, so erhält man einen Wert von 1,634. Dieser weicht nur um 5,88 % von der Gesamtbewertung ab, mit der die Probanden die Benutzungsschnittstelle bewerten. Die Benutzer stufen die Gesamtheit der Benutzungsschnittstelle als besser ein als die Summe ihrer Einzelkomponenten. Verbesserungsbedarf gibt es in den Punkten Selbstbeschreibungsfähigkeit und Fehlertoleranz. Aufgrund der Komplexität und Größe der Benutzungsschnittstelle sind Fehler in der Entwicklung nicht zu vermeiden. Allerdings erlaubt es die *Log-View* und der *Show Exception Dialog* Fehler dem Entwickler mitzuteilen und häufig führen Fehler nicht zu einem Absturz des Programms, so dass weiter gearbeitet werden kann. Die Selbstbeschreibungsfähigkeit schnitt am schlechtesten von allen Bereichen ab. Dies ist hauptsächlich auf die fehlende kontextsensitive Hilfe zurückzuführen.

Aufgrund der Evaluation konnten von den Probanden 23 Benutzungsschnittstellenfehler gefunden werden und zum Zeitpunkt des Abschlusses der Arbeit wurde ein Großteil davon behoben. Des Weiteren wurde einige der Vorschläge umgesetzt, wie z.B. die Möglichkeit die Kamera planar über die mittlere Maustaste zu bewegen.

Kapitel 7.

Zusammenfassung und Ausblick

Ich habe Dinge gesehen, die ihr Menschen
niemals glauben würdet...

(Roy Batty - Blade Runner)

Inhaltsangabe

7.1. Zusammenfassung	130
7.2. Ausblick	132

Diese Arbeit beschäftigt sich mit der echtzeitfähigen und realitätsnahen Simulation von Unterwasserumgebungen und Unterwasserrobotern (AUVs). Mehrere AUVs können parallel simuliert werden. Der Simulator ermöglicht somit die Evaluation von Schwarmverhalten, z.B. das Umweltmonitoring durch indirekte Bioindikatoren. Den Kern bildet das erstellte Gesamtmodell, bestehend aus wichtigen Einzelmodellen für die Kommunikation, Schwärme, Energieverbrauch und indirekte Bioindikatoren. Das Modell wird in der prototypischen Simulationsumgebung MARS implementiert und Teilaspekte werden anschließend evaluiert.

Wasser und Lebewesen, wie z.B. Fische, sind wichtig für die Nahrungskette und den Menschen. Deshalb ist eine Überwachung und Überprüfung wünschenswert. Taucher, Boote, Expeditionen und Messstationen sind teuer oder in ihrer Mobilität eingeschränkt. Deshalb werden zunehmend autonome robotische Systeme eingesetzt, wie z.B. AUVs. Diese Systeme bewegen sich in einer gefährlichen und fehlerintoleranten Umgebung. Um die vielfältigen Missionen zu bewältigen, sind komplexe Algorithmen, Steuerungen und Elektronik notwendig. Deshalb sind Tests im Vorfeld wichtig, allerdings aufwendig. Simulationen erlauben es, diese Tests kostengünstig und schnell durchzuführen und komplexere Szenarien ausgiebig zu testen. Ein besonderer Fokus liegt in dieser Arbeit auf dem Bereich der Schwärme, die aufgrund der großen Anzahl der Roboter schwierig sind, sowie Umweltmonitoringtests, um Gewässerqualität direkt oder indirekt durch Bioindikatoren wie Fische zu überprüfen. Letzteres steht häufig im engem Zusammenhang mit Schwärmen und der Entwicklung entsprechender Verhalten.

Dieses Kapitel fasst die vorhergehenden Kapitel in Abschnitt 7.1 zusammen, ebenso wie die Beiträge und Ergebnisse, und gibt in Abschnitt 7.2 einen Ausblick auf Themen, die nicht behandelt wurden und auf mögliche Verbesserungsvorschläge.

7.1. Zusammenfassung

Im Kapitel 2 *Stand der Forschung und Technik* wurden diverse AUV-Simulatoren beschrieben und gegenübergestellt, um zu zeigen, dass eine Lücke im Bereich der Schwarmfähigkeit und des Umweltmonitoring existiert. Kapitel 3 beschäftigt sich mit den nötigen *Grundlagen* im Bereich der AUVs und Physik im Unterwasserbereich. Kapitel 4 beschreibt das aufgestellte *Modell* der Simulation. Es wurde ein ganzheitliches Modell mit Fokus auf Echtzeitfähigkeit, Schwärmen und Umweltmonitoring entwickelt. Kapitel 5 erläutert die *prototypische Simulationsumgebung MARS*. Als Basis dient dabei das zuvor erstellte Modell. Ein wichtiger Teilaspekt ist die Benutzungsschnittstelle, die unter Zuhilfenahme der NetBeans Plattform entstanden ist und im Gegensatz zu den meisten anderen AUV-Simulatoren eine gebrauchstauglichere Bedienung ermöglicht. In Kapitel 6 *Evaluation* wurden einzelne Teile des Modells und der Simulationsumgebung evaluiert. Es wurde gezeigt, dass in der Simulation einfache und komplexe Verhalten im Schwarm zu realisieren sind. Die Beiträge dieser Arbeit sind im einzelnen:

1. Eine modulare Simulationsumgebung "MARS" für Schwärme von AUVs im Kontext von Umweltmonitoring.
 - a) Quelloffene auf Java basierende Umgebung mit 3D-Grafik und Physik-Engine.
 - b) TCP/IP- und ROS-Schnittstellen zur Kommunikation.
 - c) Moderne grafische Benutzungsschnittstelle mit zusätzlicher benutzerorientierter Evaluation.
2. Eine echtzeitfähiges, modulares Modell für Schwärme von AUVs.
 - a) Echtzeitfähiges AUV-Modell mit verschiedenen Sensoren und Aktoren, darunter einem strahlen-basierten Sonar.
 - b) Kommunikationsmodell über einen strahlen-basierten Ansatz mit Abschwächung, Rauschen und Mehrwegausbreitung.
 - c) Verbrauchs- und Energiegewinnungsmodell.
 - d) Umweltmodell für die indirekte Bioindikatoren Fische und Pflanzen.
3. Evaluation der wesentlichen Eigenschaften des Modells und der Umgebung, wie z.B. Echtzeitfähigkeit, Schwärme, Teilen des AUV- und Sensorenmodells und der Benutzungsschnittstelle.

Beim AUV-Modell wurde darauf geachtet, dass es sich physikalisch nachvollziehbar verhält, ohne aufwändig in der Berechnung zu sein. Dabei wurden die wesentlichen Kräfte wie Wasserwiderstand und Auftrieb berücksichtigt. In Bezug auf Umweltmonitoring wurden ein Vegetationsmodell, ein Fischschwarmmodell basierend auf Craig-Reynolds und ein Strömungsmodell aufgestellt. Durch die Vegetation und Fische sind indirekte Bioindikatoren vorhanden und die Strömung erschwert Positionierungen im Schwarmverhalten. Bei der Sensorik wurde ein Fokus auf das Sonar gelegt, das über einen strahlen-basierten Ansatz

modelliert wurde. Für Schwärme wurde ein Kommunikationsmodell für den Unterwassertinsatz entwickelt, das es in Echtzeit ermöglicht Daten zwischen AUVs auszutauschen. Beachtet werden dabei Entfernungen, Schallgeschwindigkeit, Rauschen und Mehrwegausbreitung über einen strahlen-basierten Einsatz. Des weiteren wurden Mechanismen bereitgestellt, um den für AUV-Schwärme interessanten Aspekt der Energieeffizienz in Form von Stromverbrauch und Aufladung handhabbar zu machen.

In der Evaluation wurden Benchmarks ausgeführt, die aussagen, dass auf herkömmlichen PC-Systemen eine echtzeitfähige Simulation von bis zu neun vollwertigen AUVs möglich ist. Teile des AUV-Modells konnten erfolgreich über eine simulierte Tiefenregelung im Vergleich zu der realen Tiefenregelung des HANSE AUV validiert werden. Zusätzlich wurden komplexe Szenarien in der Simulation getestet, wie z.B. eine Formationsfahrt und das Load-Balancing-Verhalten. Das Sonarmodell wurde mit realen Messergebnissen verglichen und gibt ein realistisches Sonarbild wieder. Für die Benutzungsschnittstelle wurde eine benutzerorientierte Evaluation durchgeführt.

Im Vergleich zu anderen Simulatoren fällt die Kombination aus Funktionen für Schwärme und Umweltmonitoring auf. Für Schwärme ist es durch die Anzahl der Roboter wichtig, performant genug zu sein, um mehr als zwei bis drei Roboter zu simulieren. Des weiteren ist ein an die Realität angelehntes Kommunikationsmodell wichtig, um mögliche Probleme in den komplexen Schwarmverhalten aufzudecken. Als direkte Bioindikatoren sind Fischschwärme gar nicht und Vegetation sehr selten vertreten. Ebenso wenig wird bei den Simulatoren, die komplexer sind und mehr Funktionalität bieten auf eine gebrauchstaugliche Benutzungsschnittstelle Wert gelegt. Dies ist aber wichtig, um in die Simulation eingreifen zu können, besonders wenn ein Schwarm viele Mitglieder aufweist. Im Hinblick auf die Vorarbeiten aus der Diplomarbeit (siehe Abschnitt 2.4) wurden verschiedene Aspekte vollständig umgearbeitet oder neu implementiert. Hervorzuheben sind im einzelnen:

1. Neue, auf der NetBeans Plattform aufsetzende Benutzungsschnittstelle.
2. Neue erweiterbare Kommunikationsschnittstelle.
3. Komplette Umstrukturierung und Modularisierung des Programmcodes.
4. Fischschwärme und Vegetation.
5. Verfeinerungen des Sonarmodells und der Unterwassersicht.
6. Hinzufügen eines Kommunikationsmodells und Energiemodells.

Abschließend lässt sich sagen, dass im Rahmen dieser Dissertation eine Simulationsumgebung entstanden ist, die gut geeignet ist, um die an sie gestellten Anforderungen zu erfüllen. Es ist möglich, ganze Schwärme von AUVs in komplexen Szenarien im Umweltmonitoring zu simulieren. Die Benutzungsschnittstelle erlaubt es, das Szenario in Echtzeit zu modifizieren und die zugrunde liegenden Modelle sind hinreichend genau und performant. Aufgrund der modularen Struktur von MARS und der Nutzung der NetBeans Plattform sind Erweiterung und Plugins vergleichsweise leicht möglich.

7.2. Ausblick

Ausgehend von dem aktuellen Stand der Arbeit bleiben eine Reihe von Verbesserungsmöglichkeiten und Forschungsthemen, die es zu untersuchen gilt. Die meisten möglichen Verbesserungen befinden sich im Bereich des Modells.

Eine Möglichkeit wäre es, die Fischeschwarmsimulation, Vegetation und Verschmutzung stärker zu verzahnen, um eine Art von Ökosystemsimulation zu erzeugen. Es würde eine Nahrungskette erzeugt werden, ausgehend von der Vegetation oder anderen Nahrungsquellen hin zu den Fischen. Verschmutzung würde sich auf die Fischpopulation und das Nahrungsangebot auswirken. Auf dieser Basis könnte man AUV-Schwarmalgorithmen zur Überwachung des Ökosystems entwickeln und spezielle Fälle nachstellen. Beim Vegetationsmodell könnte die Grafikkarte stärker eingesetzt werden. Neue Techniken im Bereich des *Geometry Based Grass* erlauben es, mehr Grass inklusive geometrischer Interaktion darzustellen [Mak]. Allerdings ist zu beachten, in wie weit die Verschiebung in die Shader das Zusammenspiel mit anderen Komponenten erschwert, wie z.B. bei der Ökosystemsimulation.

Beim AUV-Modell gibt es eine Reihe möglicher Verfeinerungen der Physik. Die Volumenberechnung für den Auftrieb ist ein wichtiger Teil. Um die Kapazitäten von Grafikkarten auszunutzen, könnte man in Zukunft auf grafische Verfahren zurückgreifen z.B. mit Hilfe von Depth-Peeling [KKKT06]. Allerdings spielt das Volumen nur für USVs und AUVs eine Rolle, die an der Wasseroberfläche schwimmen. Der verdrängte Wasseranteil am Volumen ändert sich dort ständig aufgrund der Wellen. Unter Wasser gibt es diesen Effekt nicht, so dass man abschätzen müsste, ob sich der Mehraufwand lohnt. Gleiches gilt für Wind, der sich auch auf das Wellenmodell auswirken könnte. Im Unterwasserbereich könnte man den Effekt der virtuellen Masse (engl. Added Mass) hinzufügen und das Widerstandsmodell verfeinern. Aufgrund der Komplexität des Themas ist im Bereich Strömungslehre wenig Potential für die echtzeitfähige Simulation vorhanden. Des weiteren wäre die Einführung von dynamischen Auftrieb nützlich, besonders für Gleiter.

Da das Sonar einer der wichtigsten Sensoren ist, würde es sich anbieten, das Modell zu verfeinern und auch die Performanz zu erhöhen. Um die Geschwindigkeit weiter zu erhöhen, würde sich eine Implementierung mit Hilfe des Depth-Buffer der Grafikkarte anbieten. Dadurch sind ähnliche Informationen zu erhalten wie beim strahlen-basierten Ansatz, mit dem Unterschied der deutlich höheren Auflösung und Geschwindigkeit. Allerdings sind der Blick hinter Objekte, um Sonarschatten zu implementieren, und Reflexionen schwieriger abzubilden.

Anhang A.

Anhang

Kommen wir jetzt zu etwas völlig anderem.

*(Monty Pythons wunderbare Welt der
Schwerkraft)*

Inhaltsangabe

A.1. Fragebogen	135
A.2. Anleitung	140
A.3. AUV Parameterliste	142
A.4. Umweltparameter	143
A.5. Sensorliste	144

A.1. Fragebogen

Aufgabenangemessenheit								
Unterstützt die Software die Erledigung Ihrer Arbeitsaufgaben, ohne Sie als Benutzer unnötig zu belasten?								
Die Software...	---	--	-	-/+	+	++	+++	
ist kompliziert zu bedienen.								ist unkompliziert zu bedienen.
bietet nicht alle Funktionen, um die anfallenden Aufgaben effizient zu bewältigen.								bietet alle Funktionen, die anfallenden Aufgaben effizient zu bewältigen.
bietet schlechte Möglichkeiten, sich häufig wiederholende Bearbeitungsvorgänge zu automatisieren.								bietet gute Möglichkeiten, sich häufig wiederholende Bearbeitungsvorgänge zu automatisieren.
erfordert überflüssige Eingaben.								erfordert keine überflüssigen Eingaben.
ist schlecht auf die Anforderungen der Arbeit zugeschnitten.								ist gut auf die Anforderungen der Arbeit zugeschnitten.
Selbstbeschreibungsfähigkeit								
Gibt Ihnen die Software genügend Erläuterungen und ist sie in ausreichendem Masse verständlich?								
Die Software...	---	--	-	-/+	+	++	+++	
bietet einen schlechten Überblick über ihr Funktionsangebot.								bietet einen guten Überblick über ihr Funktionsangebot.
verwendet schlecht verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs.								verwendet gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs.
liefert in unzureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind.								liefert in zureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind.
bietet auf Verlangen keine situationsspezifischen Erklärungen, die konkret weiterhelfen.								bietet auf Verlangen situationsspezifische Erklärungen, die konkret weiterhelfen.
bietet von sich aus keine situationsspezifischen Erklärungen, die konkret weiterhelfen.								bietet von sich aus situationsspezifische Erklärungen, die konkret weiterhelfen.

Steuerbarkeit								
Können Sie als Benutzer die Art und Weise, wie Sie mit der Software arbeiten, beeinflussen?								
Die Software...	---	--	-	-/+	+	++	+++	
bietet keine Möglichkeit, die Arbeit an jedem Punkt zu unterbrechen und dort später ohne Verluste wieder weiterzumachen.								bietet die Möglichkeit, die Arbeit an jedem Punkt zu unterbrechen und dort später ohne Verluste wieder weiterzumachen.
erzwingt eine unnötig starre Einhaltung von Bearbeitungsschritten.								erzwingt keine unnötig starre Einhaltung von Bearbeitungsschritten.
ermöglicht keinen leichten Wechsel zwischen einzelnen Menüs oder Masken.								ermöglicht einen leichten Wechsel zwischen einzelnen Menüs oder Masken.
ist so gestaltet, dass der Benutzer nicht beeinflussen kann, wie und welche Informationen am Bildschirm dargeboten werden.								ist so gestaltet, dass der Benutzer beeinflussen kann, wie und welche Informationen am Bildschirm dargeboten werden.
erzwingt unnötige Unterbrechungen der Arbeit.								erzwingt keine unnötigen Unterbrechungen der Arbeit.
Erwartungskonformität								
Kommt die Software durch eine einheitliche und verständliche Gestaltung Ihren Erwartungen und Gewohnheiten entgegen?								
Die Software...	---	--	-	-/+	+	++	+++	
erschwert die Orientierung, durch eine uneinheitliche Gestaltung.								erleichtert die Orientierung, durch eine einheitliche Gestaltung.
lässt einen im Unklaren darüber, ob eine Eingabe erfolgreich war oder nicht.								lässt einen nicht im Unklaren darüber, ob eine Eingabe erfolgreich war oder nicht.
informiert in unzureichendem Masse über das, was sie gerade macht.								informiert in ausreichendem Masse über das, was sie gerade macht.
reagiert mit schwer vorhersehbaren Bearbeitungszeiten.								reagiert mit gut vorhersehbaren Bearbeitungszeiten.
lässt sich nicht durchgehend nach einem einheitlichen Prinzip bedienen.								lässt sich durchgehend nach einem einheitlichen Prinzip bedienen.

Fehlertoleranz								
Bietet Ihnen die Software die Möglichkeit, trotz fehlerhafter Eingaben das beabsichtigte Arbeitsergebnis ohne oder mit geringem Korrekturaufwand zu erreichen?								
Die Software...	---	--	-	-/+	+	++	+++	
ist so gestaltet, dass kleine Fehler schwerwiegende Folgen haben können.								ist so gestaltet, dass kleine Fehler keine schwerwiegenden Folgen haben können.
informiert zu spät über fehlerhafte Eingaben.								informiert sofort über fehlerhafte Eingaben.
liefert schlecht verständliche Fehlermeldungen.								liefert gut verständliche Fehlermeldungen.
erfordert bei Fehlern im Großen und Ganzen einen hohen Korrekturaufwand.								erfordert bei Fehlern im Großen und Ganzen einen geringen Korrekturaufwand.
gibt keine konkreten Hinweise zur Fehlerbehebung.								gibt konkrete Hinweise zur Fehlerbehebung.
Individualisierbarkeit								
Können Sie als Benutzer die Software ohne großen Aufwand auf Ihre individuellen Bedürfnisse und Anforderungen anpassen?								
Die Software...	---	--	-	-/+	+	++	+++	
lässt sich von dem Benutzer schwer erweitern, wenn für ihn neue Aufgaben entstehen.								lässt sich von dem Benutzer leicht erweitern, wenn für ihn neue Aufgaben entstehen.
lässt sich von dem Benutzer schlecht an seine persönliche, individuelle Art der Arbeitserledigung anpassen.								lässt sich von dem Benutzer gut an seine persönliche, individuelle Art der Arbeitserledigung anpassen.
eignet sich für Anfänger und Experten nicht gleichermaßen, weil der Benutzer sie nur schwer an seinen Kenntnisstand anpassen kann.								eignet sich für Anfänger und Experten gleichermaßen, weil der Benutzer sie leicht an seinen Kenntnisstand anpassen kann.
lässt sich - im Rahmen ihres Leistungsumfangs - von dem Benutzer schlecht für unterschiedliche Aufgaben passend einrichten.								lässt sich - im Rahmen ihres Leistungsumfangs - von dem Benutzer gut für unterschiedliche Aufgaben passend einrichten.
ist so gestaltet, dass der Benutzer die Bildschirmdarstellung schlecht an seine individuellen Bedürfnisse anpassen kann.								lässt den Benutzer die Bildschirmdarstellung an seine individuellen Bedürfnisse anpassen .

Lernförderlichkeit								
Ist die Software so gestaltet, dass Sie sich ohne großen Aufwand in sie einarbeiten konnten und bietet sie auch dann Unterstützung, wenn Sie neue Funktionen lernen möchten?								
Die Software...	---	--	-	-/+	+	++	+++	
erfordert viel Zeit zum Erlernen.								erfordert wenig Zeit zum Erlernen.
ermutigt nicht dazu, auch neue Funktionen auszuprobieren.								ermutigt dazu, auch neue Funktionen auszuprobieren.
erfordert, dass man sich viele Details merken muss.								erfordert nicht, dass man sich viele Details merken muss.
ist so gestaltet, dass sich einmal Gelerntes schlecht einprägt.								ist so gestaltet, dass sich einmal Gelerntes gut einprägt.
ist schlecht ohne fremde Hilfe oder Handbuch erlernbar.								ist gut ohne fremde Hilfe oder Handbuch erlernbar.
Zum Schluss								
Wie gut bewerten Sie die Software insgesamt?	---	--	-	-/+	+	++	+++	
sehr schlecht								sehr gut
Wie gut kennen Sie Simulationssoftware allg.?	---	--	-	-/+	+	++	+++	
sehr schlecht								sehr gut
Ihr Geschlecht?	Männlich			Weiblich				
Wie alt sind Sie?								

A.2. Anleitung

Aufgabenstellung

Ablauf: Sie nehmen an einer benutzerorientierten Evaluation teil. Ihre Aufgabe ist es in der Simulationsumgebung MARS zwölf kleinere Aufgaben durchzuführen. Sie können während des Versuchs Fragen stellen oder Kommentare abgeben. Diese werden aufgeschrieben.

MARS ist eine Simulation für autonome Unterwasserroboter (AUV). In Abbildung 1 sehen Sie die GUI von MARS. Im oberen Bereich befinden sich Menüs und Buttons um die Simulation zu steuern. Im linken Bereich können Sie Parameter der AUVs ändern. Im rechten Bereich befindet sich die dreidimensionale Hauptansicht von MARS.

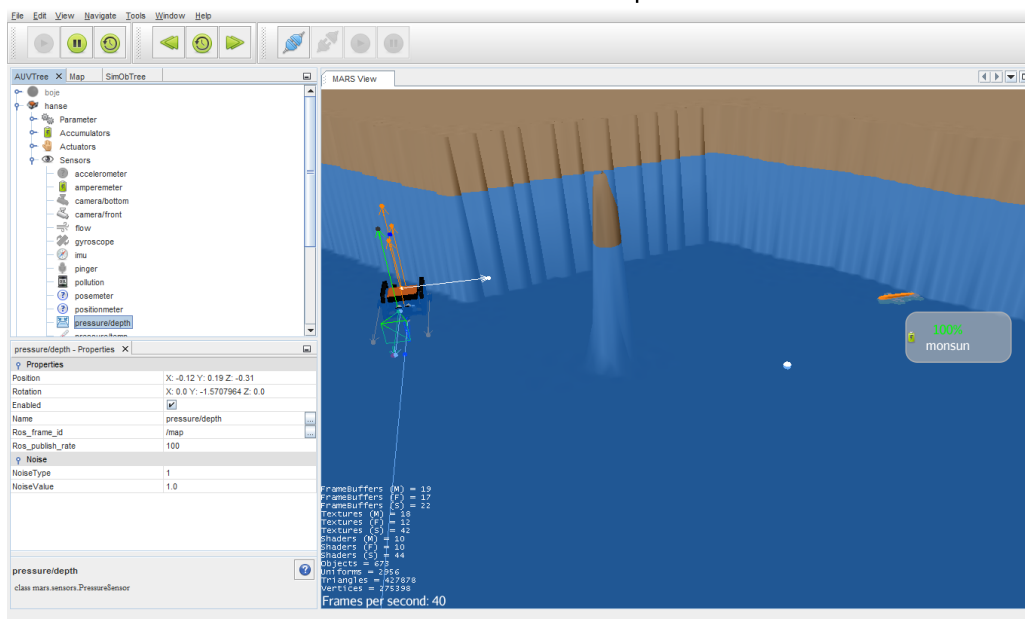


Abbildung 1: MARS GUI

Aufgabe 1: Fügen Sie ein neues AUV in die Simulation. Wählen Sie dazu aus dem Baum links das AUV HANSE und ziehen Sie es in das Hauptfenster.

Aufgabe 2: Verändern Sie Ihre Sicht auf das AUV. Bewegen Sie sich mit Hilfe der WASD-Tasten und der Maus innerhalb der Simulation (Hauptfenster).

Aufgabe 3: Verschieben und Rotieren Sie das AUV HANSE an eine andere Position. Selektieren Sie dazu das AUV in der Hauptansicht mit der Maus und benutzen Sie die STRG-Taste + Maus für Verschieben, STRG-Taste + Mausrad für die Tiefe und Shift + Maus für die Rotation.

Aufgabe 4: Starten Sie die Simulation über die Buttons in der Toolbar. Lassen Sie HANSE in eine Richtung fahren indem Sie die 8Num-Taste drücken. Über die Pfeiltasten können sie die Motoren links und rechts einzeln ansteuern. Abschließend beschleunigen Sie die Simulation.

Aufgabe 5: Schalten Sie eine Debugsicht der Sensoren/Aktoren ein. Selektieren Sie dazu das AUV in der Hauptansicht und rufen Sie mit Rechtsklick das Kontextmenü auf.

Aufgabe 6: Verändern Sie einige Parameter des AUVs. Gehen sie dazu zur Baumstruktur links und öffnen Sie die Knoten bis Sie beim Punkt *Parameters* ankommen. Verändern Sie die Parameter *position* und *waypoints->color*.

Aufgabe 7: Öffnen Sie die Sensoransicht im Baum und lassen Sie sich die Daten folgender Sensoren anzeigen: *camera/front, sonar/scan, pressure/depth*

Klicken Sie dazu mit der rechten Maus-Taste auf die Sensoren und wählen Sie im Kontextmenü den entsprechenden Punkt aus.

Aufgabe 8: Kopieren Sie das AUV HANSE um ein zweites in der Simulation zu erzeugen. Ziehen Sie dazu das vorhandene AUV HANSE unter zu Hilfenahme der STRG-Taste in die Hauptansicht.

Aufgabe 9: Schauen Sie sich die Karte unter MAP an. Diese ist zu finden im Reiterbereich des Parameterbaums der AUVs.

Aufgabe 10: Verändern Sie Optionen der Simulation. Gehen Sie dazu im Filemenü auf *Tools->Options*. Wählen Sie den Reiter *MARS_Settings->Graphics* aus. Deaktivieren Sie die den Punkt *PlaneWater* und aktivieren Sie *WavesWater*. Speichern Sie Ihre veränderte Konfiguration unter File und starten Sie sie neu.

Aufgabe 11: Deaktivieren oder löschen sie eines der AUVs.

Aufgabe 12: Sie haben jetzt noch Zeit frei mit dem Programm herum zu probieren und Fragen zu stellen.

Vielen Dank für Ihre Teilnahme!

A.3. AUV Parameterliste

Kürzel	Name	Wertebereich	Beispiel
P	Position	Vector	(0,0,0)
R	Rotation	Vector	(0,0,0)
K_p	Kollisionsbox Position	Vector	(0,0,0)
K_r	Kollisionsbox Rotation	Vector	(0,0,0)
K_t	Kollisionsbox Typ	int	1
S_p	Schwerpunkt	Vector	(0,0,0)
A_p	Auftriebspunkt	Vector	(0,0,0)
D_l	Dämpfung Linear	float	0.2
D_a	Dämpfung Angular	float	0.1
OC_h	Offview Camera Height	int	240
OC_w	Offview Camera Width	int	320
C_{dl}	Widerstandsbeiwert Linear	int	1.45
C_{da}	Widerstandsbeiwert Angular	int	0.3
m	Masse	float	19.5
U_d	Updaterate Drag	int	1
U_f	Updaterate Flow	int	1
U_b	Updaterate Buoy	int	1
b	Bedrohung	float	0.0

Tabelle A.1.: Parameter des AUV-Modell und Beispielwerte für das AUV HANSE.

A.4. Umweltparameter

Kürzel	Name	Größe	Beispiel
D_l	Dichte Luft	kg/m^3	1.2041
D_f	Dichte Fluid	kg/m^3	998.2071
T_l	Temperatur Luft	C°	20.0
T_f	Temperatur Fluid	C°	20.0
F_v	Fluid Viskosität	$mPa * s$	1.002
F_g	Schwerebeschleunigung	m/s^2	-9.80665
S_g	Globale Strömung	kgm/s^2	
W_h	Wasserhöhe	m	0.0
W_p	Druck auf Wasserhöhe	$mbar$	1013.25
S_f	Salzgehalt Fluid	$\%$	3.74

Tabelle A.2.: Die wichtigsten Umweltparameter des Umweltmodells.

A.5. Sensorliste

Name	Kommentar
Accelerometer	Einfacher Beschleunigungssensor, der die Beschleunigungswerte direkt aus der Physik-Engine zurückgibt.
Amperemeter	Liest den aktuellen Amperewert im Akkumulator aus.
Flowmeter	Einfacher Sensor, um die Fließgeschwindigkeit zu messen (siehe 4.5.3).
GPSReceiver	GPS-Sensor auf Basis von Vincentys Formel.
Gyroscope	Einfaches Kreiselinstrument, das direkt die Winkelbeschleunigung des AUV aus der Physik-Engine zurückgibt.
Hakuyo	Ein Laserscanner basierend auf Hakuyo.
IMU	Eine einfacher Inertialsensor der die Werte direkt aus der Physik-Engine zurückgibt.
InfraRedSensor	Ein Einstrahl-Infrarotsensor, der die Entfernung zu einem Hindernis angibt.
LaserScanner	Basissensor für laserscanartige Sensoren auf Strahlenbasis.
MappingSensor	Basissensor für alle Sensoren, die eine Karte verwenden, um Messdaten zurückzugeben.
Orientationmeter	Einfacher Sensor, der direkt die Orientierung des AUV aus der Physik-Engine zurückgibt.
PingDetector	Eine Art Hydrophone, das eine Geräuschquelle lokalisieren kann.
PollutionMeter	Misst die Verschmutzung in der Umgebung über eine zuvor definierte Verschmutzungskarte.
Posemeter	Einfacher Sensor, der direkt die Pose des AUV aus der Physik-Engine zurückgibt.
Positionmeter	Einfacher Sensor, der direkt die Position des AUV aus der Physik-Engine zurückgibt.
PressureSensor	Drucksensor (siehe 4.2.1)
RayBasedSensor	Basissensor für alle strahlenbasierten Sensoren wie z.B. Sonare.
SalinitySensor	Sensor für Salzgehalt
TemperatureSensor	Temperatursensor (siehe 4.2.1)
TerrainSender	Sendet die HeightMap des Terrains zurück, genutzt für OccupancyGrids in ROS.
Transformer	Sendet sämtliche Posen aller Sensoren und Aktoren zurück, genutzt in ROS für den TransformTree.

UnderwaterModem	Kommunikationsgerät, um unter Wasser zwischen AUVs Daten zu senden (siehe 4.6.1).
Velocimeter	Einfacher Sensor, der direkt die Geschwindigkeit des AUVs aus der Physik-Engine zurückgibt.
VideoCamera	Eine eigenen Rendersicht der Grafik-Engine, die ein Bild zurückgibt.
Voltagemeter	Liest den Spannungsstand im Akkumulator aus.
Sonar	Basisimplementierung für alle strahlenbasierten Sonare (siehe 4.2.2).
TriTech	Konkrete Implementierung des im SMART-E verwendeten TriTech Sonars [MEOM13].
ImagenexSonar_852_Echo	Konkrete Implementierung des im HANSE verwendeten Imagenex Echolot.
ImagenexSonar_852_Scanning	Konkrete Implementierung des im HANSE verwendeten Imagenex Scanning Sonar.

Tabelle A.3.: Alle im Sensormodell vorhandenen Sensoren.

Abbildungsverzeichnis

2.1. Dreidimensionale Visualisierung der Simulation. Bild aus [BKZ92], mit freundlicher Genehmigung von Michael Zyda.	8
2.2. Struktur und Einsatz des Simulators. Bild aus [BKZ92], mit freundlicher Genehmigung von Michael Zyda.	9
2.3. Modellstruktur von AUV-Simulatoren. Das reale AUV (gelb) interagiert mit der realen Welt (blau). Die Steuerung (grau) kann ebenfalls auf einem externen PC oder eingebetteten System laufen. Das Simulationsmodell unterteilt sich in verschiedene Untermodelle.	10
2.4. Struktur und Aussehen von SimAUV [Tos11].	12
2.5. Struktur und Aussehen von MarineSIM. Bilder aus [NSWL ⁺ 10], mit freundlicher Genehmigung von Bharath Kalyan.	15
2.6. Allgemeine Übersicht über die Struktur von Neptune mit Einsatzszenario im Hybridmodus. Das echte AUV ist im linken realen Wassertank (real execution) im Einsatz und verwendet die reale Hydrodynamik. Die Simulation rechts (virtual execution) verwendet virtuelle Sensoren, um dem echten AUV virtuelle Daten zu senden. Bild aus [RBRC04], mit freundlicher Genehmigung von Pere Ridao Rodríguez.	17
2.7. Allgemeine Übersicht über die Struktur von SubSim [BBG ⁺ 04].	18
2.8. Allgemeine Übersicht über die Struktur von Thetis. Bild aus [PLJ08], mit freundlicher Genehmigung von Lionel Lapierre.	18
2.9. UWSim Struktur und dreidimensionale Ansicht. Bilder aus [PPFS12], mit freundlicher Genehmigung von Javier Pérez Soler.	19
2.10. Allgemeine Übersicht über die Struktur von WaVeSim. Bild aus [SDM06], mit freundlicher Genehmigung von Aníbal Matos.	20
3.1. Das REMUS 6000 AUV. Bild aus [Mar], mit freundlicher Genehmigung von Hydroid Inc.	33
3.2. Der Slocum Gleiter. Bild aus [Res], mit freundlicher Genehmigung von Ben Allsup (Teledyne Webb Research).	33
3.3. Der autonome Unterwasserroboter HANSE [FHK12].	34
3.4. Der schematische Aufbau von HANSE [FHK12].	34
3.5. Der schematische Aufbau der Software von HANSE.	34
3.6. Der aktuelle autonome Unterwasserroboter MONSUN [MEIM14].	35
3.7. Das AUV MONSUN.	36

3.8. Auftriebskraft F_a wirkt auf einen Körper im Fluid und führt zu seinem Auftauchen. Die Schwerkraft F_g wirkt dem entgegen und versucht den Körper nach unten zu ziehen.	38
3.9. Die drei Stabilitätszustände eines Körpers innerhalb des Fluids. Entscheidend ist die Position des Auftriebspunktes (gelb) und Schwerpunktes (rot).	40
3.10. Berechnung des Drehmoments bei einer Schiefelage des Körpers.	40
3.11. Aufteilung des Gesamtwiderstandes in Formwiderstand $F_{W,D}$ und Reibungswiderstand $F_{W,R}$ für verschiedene Körperformen. Werte entnommen aus [Sig14] S.315.	41
4.1. Verschiedene Kräfte, die sich auf das AUV auswirken. Die Kraft der Thruster F_t kann auch in Gegenrichtung wirken.	46
4.2. Kollisionsmodell des AUV.	49
4.3. Volumenmodell des AUV.	49
4.4. Typische Thermokline vom Ozean in verschiedenen Regionen. Daten entnommen aus [Tea95], Kap.2, Seite 22.	52
4.5. Temperaturkurven für verschiedene Oberflächentemperaturen der Formel aus [Jos15]. a beträgt 1.	52
4.6. Sonarmodell: Das Sonar sendet Strahlen in einem Muster aus, das Objekte trifft. Auf Basis der Distanz und des Auftreffwinkels werden die Daten gedämpft.	53
4.7. Drei zeitliche Momentaufnahmen des Simplex Noise vor einem weißen Hintergrund.	57
4.8. Messergebnisse des SeaBotix BTD150 Thrusters [Rod07] für Kraft(blau) und Strom(rot) mit eingefügter Regression (gestrichelt).	59
4.9. Modell eines einzelnen Grasbüschels. (a) Schräge Draufsicht von vorne oben, in der man die drei einzelnen sternförmig angeordneten Flächen sieht. (b) Sicht von oben.	61
4.10. Verhalten eines einzelnen Boid (Fisches) nach Craig-Reynolds. Separation a), Zusammenhalt b) und gleichartigen Ausrichtung c).	64
4.11. (a) Lokales Koordinatensysteme eines Boids. (b) Sichtfeld eines Boid.	66
4.12. Kollisions- und Sichtsphäre des Schwarms.	66
4.13. Eingabe des Matlab-Scripts und Resultat. a) zeigt die Mauseingaben um das Vektorfeld zu berechnen. b) zeigt das resultierende Vektorfeld. Achsenangaben sind in Meter [TM14].	68
4.14. Sicht auf die Simulationsumgebung und den Effekt der Strömung. Der kleine Punkt im roten Kreis ist das AUV. Der blaue Kreis ist ein Hindernis in Form eines Pollers, das aufgrund der Strömung vom AUV umgangen wird. Die grüne Linie ist der genommene Pfad. Weiße Linien bilden ein Gitternetz zur besseren Einordnung [TM14].	69
4.15. Gesamtmodell der Unterwasserkommunikation.	70

4.16. Energiemodell für Stromverbrauch und -erzeugung. Das Solarpanel erzeugt in Abhängigkeit zum Einstrahlwinkel der Sonne Energie, die bis zu n Akkumulatoren zugeführt werden kann. Aktoren wie Thruster verbrauchen diese wieder.	75
5.1. Vereinfachte Architektur von MARS: MARS setzt auf der JVM auf. Die eigentliche Simulation (Core) verwendet NBP für die GUI, JME für die Grafik und Bullet für die Physik.	78
5.2. Nutzungsszenarien in MARS: Auf einem Computer ist MARS installiert. Das AUV-Steuerungsprogramm, das auf einem anderem Computer oder eingebetteten System läuft, verbindet sich per Netzwerk mit der Simulation. Es erhält simulierte Sensordaten und sendet selbst Aktordaten aus. Mehrere Clients können sich mit der Simulation gleichzeitig verbinden.	79
5.3. Feinere Struktur von MARS und des SimState.	80
5.4. Der von MARS verwaltete Szenegraph, bestehend aus AppState-Ebene (rot), SimState AppState (grau), AUV-Knotenbaum (blau) und SimObject-Knotenbaum (grün).	84
5.5. Grundsätzlicher Aufbau der GUI von MARS. Im Bereich <i>MainView</i> befindet sich dreidimensionale Repräsentation. Im linken Bereich <i>Tree</i> und <i>Properties</i> werden AUVs in einer Baumstruktur dargestellt und ihre Werte können manipuliert werden.	88
5.6. Beispielhafte Ansicht von MARS. Mittig ist die 3D-Umgebung mit AUVs zu sehen. Links die aufgeklappte Baumstruktur.	89
5.7. Verschiedene zusätzliche Fenster in MARS.	90
5.8. Visualisierung von Sonardaten in MARS.	91
6.1. Die Update-Loop der jMonkeyEngine.	95
6.2. Testszenario für den Benchmark.	96
6.3. Verlauf der Framerate des grafischen Benchmarks über 120 Sekunden auf dem System Laptop. Ab ungefähr Sekunde 67 wiederholt sich der Durchlauf.	97
6.4. Verlauf der Framerate des grafischen Benchmarks über 120 Sekunden auf dem System PC.	98
6.5. Testszenario mit korrespondierender HeightMap.	99
6.6. 3D-Modell, das von MONSUN innerhalb des Benchmarks verwendet wird.	100
6.7. Verlauf der Framerate des AUV-Simulation Benchmarks über 120 Sekunden auf dem System Laptop.	101
6.8. Verlauf der Framerate des AUV-Simulation Benchmarks über 120 Sekunden auf dem System PC.	102
6.9. Verzögerung für einzelne unkomprimierte Datenpakete eines MONSUN mit und ohne Kameradaten auf dem System Laptop.	105
6.10. Verzögerung für einzelne komprimierte Datenpakete eines MONSUN mit und ohne Kameradaten auf dem System Laptop.	106

6.11. Verzögerung einzelner Datenpakete für Monsun Nr.1 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	106
6.12. Verzögerung einzelner Datenpakete für Monsun Nr.2 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	107
6.13. Verzögerung einzelner Datenpakete für Monsun Nr.3 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	107
6.14. Verzögerung einzelner Datenpakete für Monsun Nr.4 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	107
6.15. Verzögerung einzelner Datenpakete für Monsun Nr.5 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	108
6.16. Verzögerung einzelner Datenpakete für Monsun Nr.6 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	108
6.17. Verzögerung einzelner Datenpakete für Monsun Nr.7 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	108
6.18. Verzögerung einzelner Datenpakete für Monsun Nr.8 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	109
6.19. Verzögerung einzelner Datenpakete für Monsun Nr.9 bei gleichzeitig acht weiteren sendenden MONSUN auf dem System PC.	109
6.20. Die simulierte (rot und orange) und reale (blau) Tiefenregelung des HANSE-AUV.	112
6.21. Drei verschiedene Strahlenmuster für das simulierte Sonar. a) einzelner Strahl. b) mehrere Strahlen in einem zirkulären Muster und einem Öffnungswinkel von 22°. c) mehrere Strahlen in einem rechteckigen Muster und einem Öffnungswinkel von 2.5° [TM14].	115
6.22. Verschiedene Dämpfungen für das simulierte Sonar. a) Keine Dämpfung. b) Längendämpfung. c) Winkeldämpfung. d) Kombination aus Längen- und Winkeldämpfung [TM14].	116
6.23. Vergleich eines realen Sonarbildes (links) und des simulierten (rechts) [TM14].	117
6.24. V-Formationsfahrt aus [AMO13].	119
6.25. Austausch eines MONSUN mit weniger Energie in der unteren Schicht mit einem aus der oberen Schicht [ATM14].	120
6.26. Energy-Load Balancing Verhalten bestehend aus 3 Phasen. Der Austausch findet in Phase 3 statt [ATM14].	121
6.27. Erfahrungsstand der Probanden zur Benutzung allgemeiner Simulationssoftware.	122
6.28. Verteilung der Evaluationsergebnisse im Bereich Aufgabenangemessenheit.	125
6.29. Verteilung der Evaluationsergebnisse im Bereich Selbstbeschreibungsfähigkeit.	125
6.30. Verteilung der Evaluationsergebnisse im Bereich Steuerbarkeit.	125
6.31. Verteilung der Evaluationsergebnisse im Bereich Erwartungskonformität.	126
6.32. Verteilung der Evaluationsergebnisse im Bereich Fehlertoleranz.	126
6.33. Verteilung der Evaluationsergebnisse im Bereich Individualisierbarkeit.	126

6.34. Verteilung der Evaluationsergebnisse im Bereich Lernförderlichkeit. . . .	127
6.35. Verteilung der Gesamtbewertung der Probanden zur Benutzungsschnittstelle von MARS.	128

Tabellenverzeichnis

2.1. Gegenüberstellung alle AUV-Simulatoren aus Abschnitt 2.5. Verwendete Abkürzungen: + für Ja, - für nein, D für Distributed, HiL für Hardware-in-the-Loop, E für Eigenentwicklung, HS für Hybrid Simulation, Z für Zentral, On für Online, Off für Offline, U für UScript. Ältere Simulatoren in grau unterlegt.	25
2.2. Gegenüberstellung alle AUV-Simulatoren aus Abschnitt 2.5 in Bezug auf die eingesetzten Modelle. In grün sind die wesentlichen Eigenschaften hervorgehoben. Ältere Simulatoren in grau unterlegt.	26
4.1. Ausbreitungskoeffizient τ nach [Tho67].	72
4.2. Verschiedene Ausbreitungsgeschwindigkeiten von Schall im Wasser.	72
6.1. Systeme auf denen der grafische Benchmark und die AUV-Simulation getestet wurden.	95
6.2. Min, Max, Durchschnittswerte und Standardabweichung der Framerate des grafischen Benchmarks auf dem System Laptop.	97
6.3. Min, Max, Durchschnittswerte und Standardabweichung der Framerate des grafischen Benchmarks auf dem System PC.	98
6.4. Die Konfiguration des im Benchmark eingesetzten MONSUN.	100
6.5. Min, Max, Durchschnittswerte und Standardabweichung der Framerate des AUV-Simulation Benchmarks auf dem System Laptop.	101
6.6. Min, Max, Durchschnittswerte und Standardabweichung der Framerate des AUV-Simulation Benchmarks auf dem System PC.	102
6.7. Min, Max, Durchschnittswerte und Standardabweichung der Verzögerungen auf dem System Laptop in ms.	105
6.8. Der RMSE der simulierten HANSE-Tiefenregelung für verschiedene P-Werte und für die Zeit von 10 bis 60 Sekunden.	112
6.9. Der MAE der simulierten HANSE-Tiefenregelung für verschiedene P-Werte und für die Zeit von 10 bis 60 Sekunden.	112
6.10. Farbkodierung der Bewertungsskala des Fragebogens. -3 ist eine sehr schlechte Bewertung, 0 eine neutrale und +3 eine sehr gute.	122
A.1. Parameter des AUV-Modell und Beispielwerte für das AUV HANSE.	142
A.2. Die wichtigsten Umweltparameter des Umweltmodells.	143
A.3. Alle im Sensormodell vorhandenen Sensoren.	145

Literaturverzeichnis

- [ABT03] Albert, Erik ; Bilodeau, Jonathan ; Turner, Roy M.: Interfacing the CoDA and CADCON Simulators: A Multi-Fidelity Simulation Testbed for Autonomous Oceanographic Sampling Networks. In: *Proceedings of the 13th International Symposium on Unmanned Untethered Submersible Technology (UUST), Durham, NH, August, 2003*, 2003
- [AMO13] Amory, Ammar ; Meyer, Benjamin ; Osterloh, Christoph ; Tosik, Thomas ; Maehle, Erik: Towards Fault-Tolerant and Energy-Efficient Swarms of Underwater Robots. In: *18th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (2013)*. <http://dx.doi.org/10.1109/IPDPSW.2013.215>. – DOI 10.1109/IPDPSW.2013.215
- [AOY11] Arima, Masakazu ; Okashima, T. ; Yamada, T.: Development of a solar-powered underwater glider. In: *Underwater Technology (UT), 2011 IEEE Symposium on and 2011 Workshop on Scientific Use of Submarine Cables and Related Technologies (SSC) IEEE, 2011*, S. 1–5. <http://dx.doi.org/10.1109/UT.2011.5774120>. DOI 10.1109/UT.2011.5774120
- [ATM14] Amory, Ammar ; Tosik, Thomas ; Maehle, Erik: A Load Balancing Behavior for Underwater Robot Swarms to Increase Mission Time and Fault Tolerance. In: *Proceedings of 28th International Parallel Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2014*. <http://dx.doi.org/10.1109/IPDPSW.2014.146>. DOI 10.1109/IPDPSW.2014.146
- [BA13] Birta, Louis G. ; Arbez, Gilbert: *Modelling and Simulation - Exploring Dynamic System Behaviour*. 2. Springer, 2013. – S.13
- [BAC⁺11] Birk, Andreas ; Antonelli, Gianluca ; Caiti, Andrea ; Casalino, Giuseppe ; Indiveri, Giovanni ; Pascoal, Antonio ; Caffaz, Andrea: The CO 3 AUVs (Cooperative Cognitive Control for Autonomous Underwater Vehicles) project: Overview and current progresses. In: *OCEANS, 2011 IEEE-Spain IEEE, 2011*, S. 1–10. <http://dx.doi.org/10.1109/Oceans-Spain.2011.6003552>. DOI 10.1109/Oceans-Spain.2011.6003552
- [BB06] Boeing, A. ; Bräunl, T.: SubSim: An autonomous underwater vehicle simulation package. In: *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005)* Springer, 2006, S. 33–38. http://dx.doi.org/10.1007/3-540-29344-2_5. DOI 10.1007/3-540-29344-2_5

- [BBCV01] Bruzzone, Ga. ; Bono, R. ; Caccia, M. ; Veruggio, G.: A simulation environment for unmanned underwater vehicles development. In: *OCEANS, 2001. MTS/IEEE Conference and Exhibition* Bd. 2 IEEE, 2001, S. 1066–1072. <http://dx.doi.org/10.1109/OCEANS.2001.968264>. DOI 10.1109/OCEANS.2001.968264
- [BBG⁺04] Bräunl, Thomas ; Boeing, Adrian ; Gonzales, Louis ; Koestler, Andreas ; Pettit, Joshua: The Autonomous Underwater Vehicle Initiative-Project Mako. In: *Robotics, Automation and Mechatronics, 2004 IEEE Conference on* Bd. 1, IEEE, 2004, S. 446–451. <http://dx.doi.org/10.1109/RAMECH.2004.1438961>. DOI 10.1109/RAMECH.2004.1438961
- [BBZP13] Bungartz, Hans-Joachim ; Buchholz, Martin ; Zimmer, Stefan ; Pflüger, Dirk: *Modellbildung und Simulation - Eine anwendungsorientierte Einführung*. 2. Springer Spektrum, 2013. <http://dx.doi.org/10.1007/978-3-642-37656-6>. <http://dx.doi.org/10.1007/978-3-642-37656-6>
- [Bel97] Bell, J.M.: Application of optical ray tracing techniques to the simulation of sonar images. In: *Optical Engineering* 36 (1997), S. 1806–1813. <http://dx.doi.org/10.1117/1.601325>. – DOI 10.1117/1.601325
- [BKM⁺05] Brooks, Alex ; Kaupp, Tobias ; Makarenko, Alexei ; Williams, Stefan ; Örebäck, Anders: Towards component-based robotics. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on* IEEE, 2005, S. 163–168. <http://dx.doi.org/10.1109/IROS.2005.1545523>. DOI 10.1109/IROS.2005.1545523
- [BKZ92] Brutzman, D.P. ; Kanayama, Y. ; Zyda, M.J.: Integrated simulation for rapid development of autonomous underwater vehicles. In: *Autonomous Underwater Vehicle Technology, 1992. AUV '92, Proceedings of the 1992 Symposium on*, 1992. <http://dx.doi.org/10.1109/AUV.1992.225199>. DOI 10.1109/AUV.1992.225199
- [BLF00] Blanke, Mogens ; Lindegaard, Karl-Petter ; Fossen, Thor I.: Dynamic model for thrust generation of marine propellers. In: *5th IFAC Conference on Manoeuvring and Control of Marine Craft*, 2000, S. 363–368
- [BLM12] Bahr, A. ; Leonard, J. ; Martinoli, A.: Dynamic positioning of beacon vehicles for cooperative underwater navigation. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, S. 3760–3767. <http://dx.doi.org/10.1109/IROS.2012.6386168>. DOI 10.1109/IROS.2012.6386168
- [BMC⁺05] Blidberg, Dick ; Mupparapu, Sai ; Chappell, Steve ; Komerska, Rick ; Jalbert, James C. ; Nitzel, Robert: The SAUV II (solar powered AUV) test results 2004. In: *Oceans 2005-Europe* Bd. 1 IEEE, 2005, S. 545–550. <http://dx.doi.org/10.1109/OCEANSE.2005.1511773>. DOI 10.1109/OCEANSE.2005.1511773
- [Boy82] Boyd, Claude E.: *Water quality management for pond fish culture*. Elsevier Scientific Publishing Co., 1982. – 318 S.

- [Bre14] Brech, Arne: *Pagingsystem für die Simulationsumgebung MARS zur Darstellung von Vegetation*, Universität zu Lübeck, Bachelorarbeit, 2014
- [BRG⁺05] Batlle, J ; Ridao, P ; Garcia, R ; Carreras, M ; Cufi, X ; El-Fakdi, A ; Ribas, D ; Nicosevici, T ; Batlle, E ; Oliver, G u. a.: URIS: Underwater Robotic Intelligent System. In: *Automation for the Maritime Industries* (2005), S. 177–203
- [bro73] *Brockhaus Enzyklopädie in 20 Bänden*. Bd. 17. F.A. Brockhaus, 1973
- [Bru94] Brutzman, Donald P.: *A Virtual World for an Autonomous Underwater Vehicle*, Naval Postgraduate School, Monterey California, Diss., December 1994. <http://faculty.nps.edu/brutzman/dissertation/>
- [Bru95] Brutzman, Don: Virtual world visualization for an autonomous underwater vehicle. In: *OCEANS'95. MTS/IEEE. Challenges of Our Changing Global Environment. Conference Proceedings*. Bd. 3 IEEE, 1995, S. 1592–1600. <http://dx.doi.org/10.1109/OCEANS.1995.528724>. DOI 10.1109/OCEANS.1995.528724
- [Brü00] Brüggem, Norbert: *Technik der U-Boot Modelle: Fernsteuerung, Tauchtechnik, Lagerregelung, Sonderfunktionen*. 4. Verlag für Technik und Handwerk Baden-Baden, 2000. – 144 S.
- [BSSL99] Belogol'skii, S.S. V.A .and S. V.A .and Sekoyan ; Samorukova, L.M. ; Stefanov, S.R. ; Levstov, V.I.: Pressure dependence of the sound velocity in distilled water. In: *Measurement techniques* 42 (1999), Nr. 4, S. 406–413. <http://dx.doi.org/10.1007/BF02504405>. – DOI 10.1007/BF02504405
- [bul] *Bullet Physics Library*. Online. <http://bulletphysics.org/wordpress/>. – Zugriff am: 24.03.2015
- [Bus15] Busse, Fabian: *Kommunikationsinterface für die Simulationsumgebung MARS*, Universität zu Lübeck, Bachelorarbeit, 2015
- [BW93] Bilaniuk, Nykolai ; Wong, George S.: Speed of sound in pure water as a function of temperature. In: *The Journal of the Acoustical Society of America* 93 (1993), Nr. 3, S. 1609–1612. <http://dx.doi.org/10.1121/1.406819>. – DOI 10.1121/1.406819
- [CKPL99] Chappell, Steven G. ; Komerska, Rick J. ; Peng, Liang ; Lu, Yingchun: Cooperative AUV Development Concept (CADCON) An Environment for High-Level AUV Simulation. In: *In Proceedings of the Eleventh International Symposium on Unmanned Untethered Submersible Technology, August 23-25, 1999, Durham NH. Published by Autonomous undersea Systems Institue, 1999*
- [CLW⁺07] Carpin, Stefano ; Lewis, Michael ; Wang, Jijun ; Balakirsky, Stephen ; Scrapper, Chris: USARSim: a robot simulator for research and education. In: *Robotics and Automation, 2007 IEEE International Conference on IEEE, 2007*, S. 1400–1405. <http://dx.doi.org/10.1109/ROBOT.2007.363180>. DOI 10.1109/ROBOT.2007.363180

- [CM77] Chen, Chen-Tung ; Millero, Frank J.: Speed of sound in seawater at high pressures. In: *The Journal of the Acoustical Society of America* 62 (1977), Nr. 5, S. 1129–1135. <http://dx.doi.org/10.1121/1.381646>. – DOI 10.1121/1.381646
- [CMBG07] Craighead, J. ; Murphy, R. ; Burke, J. ; Goldiez, B.: A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In: *Robotics and Automation, 2007 IEEE International Conference on*, 2007, S. 852 – 857. <http://dx.doi.org/10.1109/ROBOT.2007.363092>. DOI 10.1109/ROBOT.2007.363092
- [CMY00] Choi, S.K. ; Menor, S.A. ; Yuh, J.: Distributed virtual environment collaborative simulator for underwater robots. In: *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on (Volume:2)*, 2000, S. 861 – 866. <http://dx.doi.org/10.1109/IROS.2000.893127>. DOI 10.1109/IROS.2000.893127
- [Cop81] Coppens, Alan B.: Simple equations for the speed of sound in Neptunian waters. In: *The Journal of the Acoustical Society of America* 69 (1981), Nr. 3, S. 862–863. <http://dx.doi.org/10.1121/1.385486>. – DOI 10.1121/1.385486
- [Cou13] Coumans, Erwin: *Bullet 2.82 Physics SDK Manual*, 2013
- [CVL14] Cook, Daniel ; Vardy, Andrew ; Lewis, Ron: A survey of AUV and robot simulators for multi-vehicle operations. In: *Autonomous Underwater Vehicles (AUV), 2014 IEEE/OES IEEE*, 2014, S. 1–8. <http://dx.doi.org/10.1109/AUV.2014.7054411>. DOI 10.1109/AUV.2014.7054411
- [CY01] Choi, S.K. ; Yuh, J.: A virtual collaborative world simulator for underwater robots using multidimensional, synthetic environment. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on (Volume:1)*, 2001, S. 926 – 931. <http://dx.doi.org/10.1109/ROBOT.2001.932669>. DOI 10.1109/ROBOT.2001.932669
- [CYT95] Choi, Song K. ; Yuh, Junku ; Takashige, Gregg Y.: Development of the omni directional intelligent navigator. In: *Robotics & Automation Magazine, IEEE* 2 (1995), Nr. 1, S. 44–53. <http://dx.doi.org/10.1109/100.388292>. – DOI 10.1109/100.388292
- [Dau08] Dauxois, Thierry: Fermi, Pasta, Ulam and a mysterious lady. In: *arXiv preprint arXiv:0801.1590* (2008). <http://dx.doi.org/10.1063/1.2835154>. – DOI 10.1063/1.2835154
- [DD91] Dean, R.G. ; Dalrymple, R.A.: *Water Wave Mechanics for Engineers and Scientists*. World Scientific, 1991 (Advanced series on ocean engineering)
- [DG74] Del Grosso, Vincent A.: New equation for the speed of sound in natural waters (with comparisons to other equations). In: *The Journal of the Acoustical Society of America* 56 (1974), Nr. 4, S. 1084–1091. <http://dx.doi.org/10.1121/1.1903388>. – DOI 10.1121/1.1903388

- [DPR05] Dauxois, Thierry ; Peyrard, Michel ; Ruffo, Stefano: The Fermi–Pasta–Ulam 'numerical experiment': history and pedagogical perspectives. In: *European Journal of physics* 26 (2005), Nr. 5, S. S3. <http://dx.doi.org/10.1088/0143-0807/26/5/S01>. – DOI 10.1088/0143-0807/26/5/S01
- [DS04] Dorigo, Marco ; Sahin, Erol: Special issue on swarm robotics. In: *Autonomous Robots* 17 (2004), Nr. 2-3, S. 111–246. <http://dx.doi.org/10.1007/s11721-008-0020-6>. – DOI 10.1007/s11721-008-0020-6
- [euw] *The EU Water Framework Directive - integrated river basin management for Europe*. Online. http://ec.europa.eu/environment/water/water-framework/index_en.html. – Zugriff am: 19.03.2015
- [Fel14] Feldvoß, Mandy J.: *Erweiterung der Simulationsumgebung MARS um Fischschwärme nach Craig Reynolds*, Universität zu Lübeck, Bachelorarbeit, 2014
- [Fer55] Fermi, J. ; Ulam S. E. ; Pasta P. E. ; Pasta: Studies of Nonlinear Problems. In: *Los Alamos Report LA-1940* (1955)
- [FGC09] Fox, R.D. ; Gower, J.F.R. ; Curran, T.A.: Slocum glider observations during the spring bloom in the Strait of Georgia. In: *OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, IEEE, 2009, S. 1–9. <http://dx.doi.org/10.1109/OCEANSSYD.2010.5603909>. DOI 10.1109/OCEANSSYD.2010.5603909
- [FHK12] Forouher, Dariush ; Hartmann, Jan ; Klüssendorff, Jan H. ; Maehle, Erik ; Meyer, Benjamin ; Osterloh, Christoph ; Tosik, Thomas: HANSE - A Low-Cost Autonomous Underwater Vehicle. In: *AMS*, Springer, 2012 (Informatik Aktuell). – ISBN 978-3-642-32216-7, S. 147–155. http://dx.doi.org/10.1007/978-3-642-32217-4_16. DOI 10.1007/978-3-642-32217-4_16
- [FM02] Fredslund, J. ; Mataric, M.J.: A general algorithm for robot formations using local sensing and minimal communication. In: *Robotics and Automation, IEEE Transactions on* 18 (2002), Nr. 5, S. 837–846. <http://dx.doi.org/10.1109/TRA.2002.803458>. – DOI 10.1109/TRA.2002.803458. – ISSN 1042-296X
- [Fos02] Fossen, Thor I.: *Marine Control Systems Guidance, Navigation, and Control of Ships, Rigs and Underwater Vehicles*. Marine Cybernetics, 2002
- [FPLA03] Farrell, Jay A. ; Pang, Shuo ; Li, Wei ; Arrieta, Richard: Chemical plume tracing experimental results with a REMUS AUV. In: *OCEANS 2003. Proceedings* Bd. 2 IEEE, 2003, S. 962–968. <http://dx.doi.org/10.1109/OCEANS.2003.178458>. DOI 10.1109/OCEANS.2003.178458
- [FRE82a] Francois R. E., Garrison G. R.: Sound absorption based on ocean measurements. Part I: Pure water and magnesium sulfate contributions. In: *Journal of the Acoustical Society of America* 72 (1982), Nr. 3, S. 896–907. <http://dx.doi.org/10.1121/1.388170>. – DOI 10.1121/1.388170

- [FRE82b] Francois R. E., Garrison G. R.: Sound absorption based on ocean measurements. Part II: Boric acid contribution and equation for total absorption. In: *Journal of the Acoustical Society of America* 72 (1982), Nr. 6, S. 1879–1–890. <http://dx.doi.org/10.1121/1.388673>. – DOI 10.1121/1.388673
- [FW07] Frisvad, Jeppe R. ; Wyvill, Geoff: Fast high-quality noise. In: *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia ACM*, 2007, S. 243–248. <http://dx.doi.org/10.1145/1321261.1321305>. DOI 10.1145/1321261.1321305
- [GVH03] Gerkey, Brian P. ; Vaughan, Richard T. ; Howard, Andrew: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, 2003, S. 317–323
- [GVM98] Gračanin, Denis ; Valavanis, Kimon P. ; Matijašević, Maja: Virtual environment test-bed for autonomous underwater vehicles. In: *Control Engineering Practice* 6 (1998), Nr. 5, S. 653–660. [http://dx.doi.org/10.1016/S0967-0661\(98\)00059-8](http://dx.doi.org/10.1016/S0967-0661(98)00059-8). – DOI 10.1016/S0967-0661(98)00059-8
- [HB09] Haertel-Borer, Susanne: *Der wundersame Fisch und seine Welt*. Fischereiberatungsstelle FIBER, 2009 <http://www.fischereiberatung.ch/news/News2010/wunder.pdf>
- [Her05] Herczeg: *Software-Ergonomie: Grundlagen der Mensch-Computer-Kommunikation*. München: Oldenbourg Verlag, 2005
- [HJ89] Hillson, R. ; Jones, C.: Evolution of an Auv Mission Simulation Testbed. In: *Unmanned Untethered Submersible Technology, 1989. Proceedings of the 6th International Symposium on IEEE*, 1989, S. 525–535. <http://dx.doi.org/10.1109/UUST.1989.754744>. DOI 10.1109/UUST.1989.754744
- [Hor01] Hornfeld, Willi: *DeepC - the German AUV Development Project*. Online. http://www.imar-navigation.de/downloads/papers/deep_c_auv.pdf. Version: 2001. – Zugriff am: 20.04.2015
- [HTD⁺87] Hebert, M. ; Thorpe, C. ; Dunn, S. ; Cushieri, J. ; Rushfelt, P. ; Girodet, W. ; Schweizer, P.: A feasibility study for a long range autonomous underwater vehicle. In: *Unmanned Untethered Submersible Technology, Proceedings of the 1987 5th International Symposium on Bd. 5 IEEE*, 1987, S. 1–13. <http://dx.doi.org/10.1109/UUST.1987.1158591>. DOI 10.1109/UUST.1987.1158591
- [HTH⁺12] Hartmann, Jan ; Tosik, Thomas ; Harder, Jannis ; Fechner, Jan ; Reddecker, Hans-Joachim ; Kampsen, Andrea ; Zenker, Patrick ; Friedrichs, Sven ; Isokeit, Cedric ; Stahl, Patrik ; Hegen, Peter ; Maehle, Erik: SAUC-E 2012 - The Hanse Team. 7th Student Autonomous Underwater Challenge Europe. (2012)
- [ima] ; Imagenex Technology Corp. (Veranst.): *Imagenex model 852 ultra-miniature scanning sonar*. http://www.imagenex.com/852_Specs_rev4.pdf

- [iso06] *Ergonomie der Mensch-System-Interaktion. Teil 110: Grundsätze der Dialoggestaltung*. 2006. – Norm EN ISO 9241-110:2006
- [JBD⁺03] Jalbert, James ; Baker, John ; Duchesney, John ; Pietryka, Paul ; Dalton, William ; Blidberg, Steve D.R .and C. D.R .and Chappell ; Nitzel, Robert ; Holappa, Ken: A solar-powered autonomous underwater vehicle. In: *Oceans 2003. Proceedings* Bd. 2 IEEE, 2003, S. 1132–1140. <http://dx.doi.org/10.1109/OCEANS.2003.178503>. DOI 10.1109/OCEANS.2003.178503
- [jme] *jMonkey Engine 3*. Online. <http://jmonkeyengine.com/>. – Zugriff am: 24.03.2015
- [Jon14] Jonte, John P.: *Verbesserte Wassereffekte für die Simulationsumgebung MARS*, Universität zu Lübeck, Bachelorarbeit, 2014
- [Jos15] Joseph: *The Temperature of Ocean Water at a Given Depth*. Online. <http://residualanalysis.blogspot.de/2010/02/temperature-of-ocean-water-at-given.html>. Version: 2015. – Zugriff am: 28.07.2015
- [KAU96] Kuroda, Yoji ; Aramaki, Koji ; Ura, Tamaki: AUV test using real/virtual synthetic world. In: *Autonomous Underwater Vehicle Technology, 1996. AUV'96., Proceedings of the 1996 Symposium on IEEE*, 1996, S. 365–372. <http://dx.doi.org/10.1109/AUV.1996.532436>. DOI 10.1109/AUV.1996.532436
- [KC06] Komerska, R.J. ; Chappell, S.G.: A Simulation Environment for Testing and Evaluating Multiple Cooperating Solar-powered AUVs. In: *OCEANS 2006, Boston, MA*, 2006, S. 1–6. <http://dx.doi.org/10.1109/OCEANS.2006.306987>. DOI 10.1109/OCEANS.2006.306987
- [KEG08] Kirillin, Georgiy ; Engelhardt, Christof ; Golosov, Sergey: A mesoscale vortex in a small stratified lake. In: *Environmental fluid mechanics* 8 (2008), Nr. 4, S. 349–366. <http://dx.doi.org/10.1007/s10652-008-9101-8>. – DOI 10.1007/s10652-008-9101-8
- [KKKT06] Kim, Jinwook ; Kim, Soojae ; Ko, Heedong ; Terzopoulos, Demetri: Fast GPU computation of the mass properties of a general shape and its application to buoyancy simulation. In: *The Visual Computer* 22 (2006), Nr. 9-11, S. 856–864. <http://dx.doi.org/10.1007/s00371-006-0071-x>. – DOI 10.1007/s00371-006-0071-x
- [Klu75] Kluge, Friedrich: *Etymologisches Wörterbuch der deutschen Sprache*. 21. de Gruyter, 1975
- [LBM⁺95] Leonard, J.J. ; Bellingham, J.G. ; Moran, B.A. ; Kim, J.H. ; Chrysostomidis, C. ; Tuohy, S.T. ; Patrikalakis, N.M. ; Chrysostomidis, C. ; Schmidt, H.: Virtual Environments for AUV Development and Ocean Exploration. In: *Proc Int Symp on Unmanned Untethered Submersible Technology* University Of New Hampshire-Marine Systems, 1995, S. 436–443

- [LFR⁺98] Lane, D.M. ; Falconer, G.J. ; Randall, G.W. ; Duffy, N.D. ; Herd, J.T. ; Chernet, P. ; Hunter, J. ; Colley, M. ; Standeven, J. ; Callaghan, V. u. a.: Mixing simulations and real subsystems for subsea robot development. specification and development of the core simulation engine. In: *OCEANS'98 Conference Proceedings* Bd. 3 IEEE, 1998, S. 1382–1386. <http://dx.doi.org/10.1109/OCEANS.1998.726295>. DOI 10.1109/OCEANS.1998.726295
- [LG98] Lubbers, J. ; Graaff, R.: A simple and accurate formula for the sound velocity in water. In: *Ultrasound in medicine & biology* 24 (1998), Nr. 7, S. 1065–1068. [http://dx.doi.org/10.1016/S0301-5629\(98\)00091-X](http://dx.doi.org/10.1016/S0301-5629(98)00091-X). – DOI 10.1016/S0301-5629(98)00091-X
- [LH12] Li, Binghui ; Hall, Marshall V.: The loss mechanisms of plane-wave reflection from the seafloor with elastic characteristics. In: *Acoustics 2012 Fremantle: Acoustics, Development and the Environment* Australian Acoustical Society, 2012
- [LMP⁺05] Listak, Madis ; Martin, Georg ; Pugal, Deivid ; Aabloo, Alvo ; Kruusmaa, Maarja: Design of a semiautonomous biomimetic underwater vehicle for environmental monitoring. In: *6th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA2005)*, IEEE, 2005, S. 9–14. <http://dx.doi.org/10.1109/CIRA.2005.1554247>. DOI 10.1109/CIRA.2005.1554247
- [Ltd15] Ltd., Fugro General R.: *Deepworks*. Online. <http://www.fugrogrl.com/software/>. Version: 2015. – Zugriff am: 30.04.2015
- [Lur10] Lurton, Xavier: *An Introduction to Underwater Acoustics: Principles and Applications*. 2. Springer-Verlag Berlin Heidelberg, 2010 (724)
- [MAB⁺07] McGillicuddy, Dennis J. ; Anderson, Laurence A. ; Bates, Nicholas R. ; Bibby, Thomas ; Buesseler, Ken O. ; Carlson, Craig A. ; Davis, Cabell S. ; Ewart, Courtney ; Falkowski, Paul G. ; Goldthwait, Sarah A. u. a.: Eddy/wind interactions stimulate extraordinary mid-ocean plankton blooms. In: *Science* 316 (2007), Nr. 5827, S. 1021–1026. <http://dx.doi.org/10.1126/science.1136256>. – DOI 10.1126/science.1136256
- [Mac81] Mackenzie, Kenneth V.: Nine-term equation for sound speed in the oceans. In: *The Journal of the Acoustical Society of America* 70 (1981), Nr. 3, S. 807–812. <http://dx.doi.org/10.1121/1.386920>. – DOI 10.1121/1.386920
- [Mak] Makarov, Evgeny: *NVidia Turf Effects: Massive Grass Rendering With Dynamic Simulation*. Online. <http://on-demand.gputechconf.com/gtc/2015/presentation/S5748-Evgeny-Makarov.pdf>. – Zugriff am: 17.07.2015
- [Man08] Manley, J.E.: Unmanned surface vehicles, 15 years of development. In: *OCEANS 2008*, IEEE, 2008, S. 1–4. <http://dx.doi.org/10.1109/OCEANS.2008.5152052>. DOI 10.1109/OCEANS.2008.5152052

- [Man10] Maniak, Ulrich: *Hydrologie und Wasserwirtschaft - Eine Einführung für Ingenieure*. Springer Berlin Heidelberg, 2010. <http://dx.doi.org/10.1007/978-3-642-05396-2>. <http://dx.doi.org/10.1007/978-3-642-05396-2>
- [Mar] Maritime, Kongsberg: *Remus 6000*. Online. <http://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/481519DA1B0207CDC12574B0002A8451?OpenDocument>. – Zugriff am: 23.03.2015
- [Mar97] Marczak, Wojciech: Water as a standard in the measurements of speed of sound in liquids. In: *The Journal of the Acoustical Society of America* 102 (1997), Nr. 5, S. 2776–2779. <http://dx.doi.org/10.1121/1.420332>. – DOI 10.1121/1.420332
- [MEIM14] Meyer, B. ; Ehlers, K. ; Isokeit, C. ; Maehle, E.: The development of the modular Hard- and Software Architecture of the Autonomous Underwater Vehicle MONSUN. In: *ISR/Robotik 2014 - 41st International Symposium on Robotics 6th German Conference on Robotics*, VDE Verlag, 2014. – München 2014
- [MEOM13] Meyer, B. ; Ehlers, K. ; Osterloh, C. ; Maehle, E.: SMART-E: An Autonomous Omnidirectional Underwater Robot. In: *Journal of Behavioral Robotics* 4 (2013), S. 204–210. <http://dx.doi.org/10.2478/pjbr-2013-0015>. – DOI 10.2478/pjbr-2013-0015
- [Mey13] Meyer, C.; Maehle E. B.; Osterloh O. B.; Osterloh: MONSUN - A Modular Testbed for Swarms of Autonomous Underwater Vehicles. In: *ICRA2013 - Workshop on Many Robot Systems: Crossing the Reality Gap – From Single to Multi- to Many Robot Systems*, 2013. – Karlsruhe
- [MFN06] Metta, Giorgio ; Fitzpatrick, Paul ; Natale, Lorenzo: YARP: yet another robot platform. In: *International Journal on Advanced Robotics Systems* 3 (2006), Nr. 1, S. 43–48
- [MKT08] Matsebe, O. ; Kumile, C. M. ; Tlale, N. S.: A review of virtual simulators for autonomous underwater vehicles (AUVs). In: *Navigation, Guidance and Control of Underwater Vehicles* Bd. 2, 2008, S. 31–37. <http://dx.doi.org/10.3182/20080408-3-IE-4914.00007>. DOI 10.3182/20080408-3-IE-4914.00007
- [MM12] Maas, R. ; Maehle, E.: Fault tolerant and adaptive path planning in crowded environments for mobile robots based on hazard estimation via health signals. In: *ARCS Workshops (ARCS), 2012 IEEE*, 2012, S. 1–7
- [nbpa] *NetBeans Platform*. Online. <https://netbeans.org/features/platform/>. – Zugriff am: 24.03.2015
- [nbpb] *NetBeans Platform Showcase*. Online. <https://netbeans.org/features/platform/showcase.html>. – Zugriff am: 14.07.2015

- [Neh02] Nehmzow, Ulrich: *Mobile Robotik: Eine praktische Einführung*. Springer, 2002. <http://dx.doi.org/10.1007/978-3-642-55942-6>. <http://dx.doi.org/10.1007/978-3-642-55942-6>
- [New] Newman, Paul: *Unmanned vehicles for shallow and coastal waters*. Online. http://www.ths.org.uk/documents/ths.org.uk/downloads/shallowwater_auv_and_usv.pdf. – Zugriff am: 23.03.2015
- [New03] Newman, Paul M.: MOOS - Mission Orientated Operating Suite / Department of Ocean Engineering - Massachusetts Institute of Technology. Version: 2003. <http://www.robots.ox.ac.uk/~pnewman/papers/MOOS.pdf>. 2003. – Forschungsbericht
- [NH08] Nicholson, J. W. ; Healey, A. J.: The Present State of Autonomous Underwater Vehicle (AUV) Applications and Technologies. In: *Marine Technology Society Journal* 42 (2008), Nr. 1, S. 44–51. <http://dx.doi.org/10.4031/002533208786861272>. – DOI 10.4031/002533208786861272
- [Nie00] Nielsen, Jakob: *Why You Only Need to Test with 5 Users*. Online. <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>. Version: 2000. – Zugriff am: 25.03.2015
- [Nie06] Nielsen, Jakob: *Quantitative Studies: How Many Users to Test?* Online. <http://www.nngroup.com/articles/quantitative-studies-how-many-users/>. Version: 2006. – Zugriff am: 25.03.2015
- [NSWL+10] Namal Senarathne, P.G.C. ; Wijesoma, W.S. ; Lee, Kwang W. ; Kalyan, B. ; Moratuwage, M.D.P. ; Patrikalakis, N.M. ; Hover, F.S.: MarineSIM: Robot simulation for marine environments. In: *OCEANS 2010 IEEE - Sydney, 2010*, S. 1–5. <http://dx.doi.org/10.1109/OCEANSSYD.2010.5603839>. DOI 10.1109/OCEANSSYD.2010.5603839
- [OLM09] Osterloh, Christoph ; Litza, Marek ; Maehle, Erik: Hard- and Software Architecture of a Small Autonomous Underwater Vehicle for Environmental Monitoring Tasks. In: *German Workshop on Robotics*, Springer Berlin Heidelberg, 2009, S. 347–356. http://dx.doi.org/10.1007/978-3-642-01213-6_31. DOI 10.1007/978-3-642-01213-6_31
- [OM10] Osterloh, Christoph ; Maehle, Erik: Low-Power Microcontroller-based Acoustic Modem for Underwater Robot Communication. In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, 2010, S. 1–6
- [OMAM12] Osterloh, Christoph ; Meyer, Benjamin ; Amory, Pionteck T. Amory ; Maehle, Erik: MONSUN II - Towards Autonomous Underwater Swarms for Environmental Monitoring. In: *International Conference on Intelligent Robots and Systems (IROS)*,

- Workshop on Robotics for Environmental Monitoring, Vilamoura, Algarve, Portugal, 2012*, S. 7–12
- [OPM12] Osterloh, Christoph ; Pionteck, Thilo ; Maehle, Erik: MONSUN II: A small and inexpensive AUV for underwater swarms. In: *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, 2012, S. 1–6
- [PCJ08] Parodi, Olivier ; Creuze, Vincent ; Jouvencel, Bruno: Communications with Thetis, a Real Time Multi-vehicles Hybrid Simulator. In: *ISOPE'08: International Society of Offshore and Polar Engineers*, 2008, S. 326–331
- [PCJX09] Parodi, Olivier ; Creuze, Vincent ; Jouvencel, Bruno ; Xiang, Xianbo: Comparison between results obtained with Thetis, a real-time multi-vehicles hardware-in-the-loop simulator, and results obtained during sea trials. In: *OCEANS 2009-EUROPE IEEE*, 2009, S. 1–7. <http://dx.doi.org/10.1109/OCEANSE.2009.5278119>. DOI 10.1109/OCEANSE.2009.5278119
- [Pel04] Pelzer, Kurt: *Gpu Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Addison Wesley Pub Co Inc, 2004. – 816 S. http://http.developer.nvidia.com/GPUGems/gpugems_ch07.html. – Chapter 7. Rendering Countless Blades of Waving Grass
- [Per01] Perlin, Ken: Noise hardware. In: *Real-Time Shading SIGGRAPH Course Notes (2001)*
- [Per02] Perlin, Ken: Improving noise. In: *ACM Transactions on Graphics (TOG) Bd. 21* ACM, 2002, S. 681–682. <http://dx.doi.org/10.1145/566570.566636>. DOI 10.1145/566570.566636
- [PGP⁺11] Purcell, M. ; Gallo, D. ; Packard, G. ; Dennett, M. ; Rothenbeck, M. ; Sherrell, A. ; Pascaud, S.: Use of REMUS 6000 AUVs in the search for the Air France Flight 447. In: *OCEANS 2011*, 2011, S. 1–7
- [PLJ08] Parodi, Olivier ; Lapierre, Lionel ; Jouvencel, Bruno: Thetis: A real-time multi-vehicles hybrid simulator for heterogeneous vehicles. In: *IROS'08: International Conference on Intelligent Robots and Systems. Workshop on Robot Simulators: Available Software, Scientific Applications and Future Trends IEEE/RSJ*, 2008
- [PPFS12] Prats, M. ; Perez, J. ; Fernandez, J.J. ; Sanz, P.J.: An open source tool for simulation and supervision of underwater intervention missions. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, S. 2577 – 2582. <http://dx.doi.org/10.1109/IROS.2012.6385788>. DOI 10.1109/IROS.2012.6385788
- [PSOW91] Pappas, George ; Shotts, William ; O'Brien, Mack ; Wyman, William: The DARPA/Navy Unmanned Undersea Vehicle Program. In: *Unmanned Systems Bd. 9*, 1991, S. 24–30

- [QCG⁺09] Quigley, M. ; Conley, K. ; Gerkey, B. P. ; Faust, J. ; Foote, T. ; Leibs, J. ; Wheeler, R. ; Ng, A.: ROS: an open-source Robot Operating System. In: *ICRA Workshop on Open Source Software* Bd. 3, 2009
- [RB11] Rathnam, Ravi ; Birk, Andreas: *Release of the final simulator with a rich set of modeled sensors, vehicles and scenarios*. Online. <http://robotics.jacobs-university.de/projects/Co3-AUVs/publicdeliverables/D22-Simulator2.pdf>. Version: 10 2011. – Zugriff am: 12.05.2015
- [RBRC04] Ridao, P. ; Batlle, E. ; Ribas, D. ; Carreras, M.: Neptune: a hil simulator for multiple UUVs. In: *OCEANS '04. MTTs/IEEE TECHNO-OCEAN '04*, 2004, S. 524 – 531. <http://dx.doi.org/10.1109/OCEANS.2004.1402970>. DOI 10.1109/OCEANS.2004.1402970
- [RCREF04] Ridao, P. ; Carreras, M. ; Ribas, D. ; El-Fakdi, A.: Graphical simulators for AUV development. In: *Control, Communications and Signal Processing, 2004. First International Symposium on*, 2004. <http://dx.doi.org/10.1109/ISCCSP.2004.1296441>. DOI 10.1109/ISCCSP.2004.1296441
- [Res] Research, Teledyne W.: *Product Catalog - Slocum G2 Glider*. Online. http://www.webbresearch.com/pdf/G2_Product_Catalog.pdf. – Zugriff am: 23.03.2015
- [Rey87] Reynolds, Craig W.: Flocks, herds and schools: A distributed behavioral model. In: *ACM Siggraph Computer Graphics* Bd. 21 ACM, 1987, S. 25–34. <http://dx.doi.org/10.1145/37401.37406>. DOI 10.1145/37401.37406
- [Rey99] Reynolds, Craig W.: Steering behaviors for autonomous characters. In: *Game developers conference*, 1999, S. 763–782
- [RGM⁺15] Renner, C. ; Gabrecht, A. ; Meyer, B. ; Osterloh, C. ; Maehle, E.: *Low-Power Low-Cost Acoustic Underwater Modem*. In: *Ocean Engineering & Oceanography* (2015). – Springer
- [RM00] Read, P. ; Meyer, M.P.: *Restoration of motion picture film*. Butterworth Heinemann, 2000. – 22–24 S.
- [Rod07] Rodocker, J.: *Standard Thruster & 2 wire whip BTD150*. Online. http://www.seabotix.com/products/pdf_files/BTD150_Data_Sheet.pdf. Version: 2007. – Zugriff am: 24.07.2015
- [ros] *rosjava*. Online. <https://github.com/rosjava>. – Zugriff am: 24.03.2015
- [rov] ; Marine Simulation (Veranst.): *ROVsim2Pro*. Online. <http://marinesimulation.com/>. – Zugriff am: 06.08.2015
- [RS86] Rolfe, J.M. ; Staples, K.J.: *Flight Simulation*. Cambridge University Press, 1986 (Cambridge Aerospace Series). – 27 S.

- [RSF13] Rohmer, Eric ; Singh, Surya P. ; Freese, Marc: V-REP: A versatile and scalable robot simulation framework. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on IEEE*, 2013, S. 1321–1326. <http://dx.doi.org/10.1109/IROS.2013.6696520>. DOI 10.1109/IROS.2013.6696520
- [Rut15] Rutgers University: *The Scarlet Knight's Trans-Atlantic Challenge*. Online. <http://rucool.marine.rutgers.edu/atlantic/>. Version: 2015. – Zugriff am: 24.03.2015
- [SAA⁺01] Stokey, Roger ; Austin, Tom ; Allen, Ben ; Forrester, Ned ; Gifford, Eric ; Goldsborough, Rob ; Packard, Greg ; Purcell, Mike ; Alt, Chris von: Very shallow water mine countermeasures using the REMUS AUV: a practical approach yielding accurate results. In: *OCEANS, 2001. MTS/IEEE Conference and Exhibition Bd. 1 IEEE*, 2001, S. 149–156. <http://dx.doi.org/10.1109/OCEANS.2001.968696>. DOI 10.1109/OCEANS.2001.968696
- [SAF03] Song, Feijun ; An, P. E. ; Folleco, Andres: Modeling and simulation of autonomous underwater vehicles: design and implementation. In: *Oceanic Engineering, IEEE Journal of* 28 (2003), Nr. 2, S. 283–296. <http://dx.doi.org/10.1109/JOE.2003.811893>. – DOI 10.1109/JOE.2003.811893
- [San06] Sangwin, Christopher J.: Locating the centre of mass by mechanical means. In: *Journal of the Oughtred Society* 15 (2006), Nr. 2, S. 16–18
- [sau] *Student Autonomous Underwater Vehicle Challenge - Europe*. Online. <http://sauc-europe.org/>. – Zugriff am: 23.03.2015
- [SC10] Sehgal, Anuj ; Cernea, Daniel: A multi-AUV missions simulation framework for the USARSim Robotics Simulator. In: *Control & Automation (MED), 2010 18th Mediterranean Conference on IEEE*, 2010, S. 1188–1193. <http://dx.doi.org/10.1109/MED.2010.5547632>. DOI 10.1109/MED.2010.5547632
- [SCB10] Sehgal, Anuj ; Cernea, Daniel ; Birk, Andreas: Simulating Underwater Acoustic Communications in a High Fidelity Robotics Simulator. In: *7th IFAC Symposium on Intelligent Autonomous Vehicles, University of Salento, Lecce, Italy, 6-8 Sept. 2010* Elsevier, 2010, S. 587–592
- [Sch05] Schneider, Susanne: *Indikatoreigenschaften und Ökologie aquatischer Makrophyten in stehenden und fließenden Gewässern*. Version: 2005. <http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss20050711-1401492483> Technische Universität München
- [Sch15] Schwinghammer, Jasper: *Simulation und Visualisierung von akustischer Unterwasserkommunikation in MARS*, Universität zu Lübeck, Bachelorarbeit, 2015
- [SDM06] Santos, António N. ; De Matos, Aníbal Castilho C.: WaVeSim-Water Vehicle Simulator. In: *MCMC2006 7th IFAC Conference on Manoeuvring and Control of Marine Craft, Lisbon, 2006*

- [ser] *Serafina*. Online. http://users.rsise.anu.edu.au/serafina/public_html/. – Zugriff am: 18.03.2015
- [Shu60] Shubik, Martin: Simulation of the Industry and the Firm. In: *The American Economic Review* (1960), S. 908–919
- [Sig14] Sigloch, Herbert: *Technische Fluidmechanik*. 9. Springer, Berlin, 2014. <http://dx.doi.org/10.1007/978-3-642-54292-3>. <http://dx.doi.org/10.1007/978-3-642-54292-3>
- [STT⁺11] Schmickl, Thomas ; Thenius, Ronald ; Timmis, Jon ; Tyrrell, Andy ; Halloy, Jose ; Stefanini, Cesare ; Manfredi, Luigi ; Campo, Alexandre ; Sutantyo, Donny ; Kernbach, Serge: CoCoRo: The self-aware swarm of underwater robots IROS 2011–IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011
- [Tea95] Team, Open University C. ; Mark A. Suckow, Steven H. W. (Hrsg.) ; Franklin, Craig L. (Hrsg.): *Seawater: Its Composition, Properties and Behaviour*. 2 Edition. Elsevier Ltd., 1995
- [Tho67] Thorp, William H.: Analytic Description of the Low-Frequency Attenuation Coefficient. In: *The Journal of the Acoustical Society of America* 42 (1967), Nr. 1, S. 270–270. <http://dx.doi.org/10.1121/1.1910566>. – DOI 10.1121/1.1910566. – Acoustical Society of America
- [TM14] Tosik, Thomas ; Maehle, Erik.: MARS: A Simulation Environment for Marine Robotics. In: *OCEANS 14 MTS/IEEE St. Johns, St. Johns, Kanada* IEEE, 2014, S. 1–7. <http://dx.doi.org/10.1109/OCEANS.2014.7003008>. DOI 10.1109/OCEANS.2014.7003008
- [TM15] Tipler, Paul A. ; Mosca, Gene ; Wagner, Jenny (Hrsg.): *Physik: für Wissenschaftler und Ingenieure*. 7. Springer Spektrum, 2015
- [Tos11] Tosik, Thomas: *Physikalisch realitätsnahe Simulationsumgebung für das AUV Hanse, mit Integration in die bestehende Steuerungs-Software*, Universität zu Lübeck, Diplomarbeit, Mai 2011
- [TPES11] Talley, Lynne ; Pickard, George ; Emery, William ; Swift, James: *Descriptive Physical Oceanography, Sixth Edition: An Introduction*. Elsevier Ltd., 2011
- [Uri96] Urick, Robert J.: *Principles of Underwater Sound*. 3. Peninsula Publishing, 1996
- [vor] ; CMLabs (Veranst.): *Remotely Operated Vehicle (ROV) Simulator*. Online. <http://www.cm-labs.com/market/energy-offshore/products/remotely-operated-vehicle-rov-simulator>. – Zugriff am: 06.08.2015
- [wata] ; National Physical Laboratory (Veranst.): *Technical Guides - Speed of Sound in Pure Water*. Online. <http://resource.npl.co.uk/acoustics/techguides/soundpurewater/>. – Zugriff am: 07.07.2015

- [watb] ; National Physical Laboratory (Veranst.): *Technical Guides - Speed of Sound in Sea-Water*. Online. <http://resource.npl.co.uk/acoustics/techguides/soundseawater/content.html>. – Zugriff am: 07.07.2015
- [WCA⁺10] Woithe, H.C. ; Chigirev, I. ; Aragon, D. ; Iqbal, M. ; Shames, Y. ; Glenn, S. ; Schofield, O. ; Seskar, I. ; Kremer, U.: Slocum Glider energy measurement and simulation infrastructure. In: *OCEANS 2010 IEEE - Sydney*, 2010, S. 1–8. <http://dx.doi.org/10.1109/OCEANSSYD.2010.5603909>. DOI 10.1109/OCEANSSYD.2010.5603909
- [WK06] Wächter, C. ; Keller, A.: Instant Ray Tracing: The Bounding Interval Hierarchy. In: *Proceedings of the 17th Eurographics Symposium on Rendering*, 2006, S. 139–149
- [Woo94] Woolley, Benjamin: *Die Wirklichkeit der virtuellen Welten: Aus dem Englischen von Gabriele Herbst*. Birkhäuser, 1994. – 60 S.
- [WSJ01] Webb, Douglas C. ; Simonetti, Paul J. ; Jones, Clayton P.: SLOCUM: An underwater glider propelled by environmental energy. In: *IEEE Journal of Oceanic Engineering* 26 (2001), Nr. 4, S. 447–452. <http://dx.doi.org/10.1109/48.972077>. – DOI 10.1109/48.972077
- [WY99] Whitcomb, Louis L. ; Yoerger, Dana R.: Development, comparison, and preliminary experimental validation of nonlinear dynamic thruster models. In: *Oceanic Engineering, IEEE Journal of* 24 (1999), Nr. 4, S. 481–494. <http://dx.doi.org/10.1109/48.809270>. – DOI 10.1109/48.809270
- [ZC01] Zhang, C. ; Chen, T.: Efficient feature extraction for 2D/3D objects in mesh representation. In: *Image Processing, 2001. Proceedings. 2001 International Conference on* Bd. 3, 2001. <http://dx.doi.org/10.1109/ICIP.2001.958278>. DOI 10.1109/ICIP.2001.958278
- [ZPK00] Zeigler, Bernard P. ; Praehofer, Herbert ; Kim, Tag G.: *Theory of Modeling and Simulation*. 2. Elsevier, 2000

Eigene Publikationen

- [TM14] Tosik, Thomas ; Maehle, Erik.: MARS: A Simulation Environment for Marine Robotics. In: *OCEANS 14 MTS/IEEE St. Johns, St. Johns, Kanada* IEEE, 2014, S. 1–7. <http://dx.doi.org/10.1109/OCEANS.2014.7003008>. DOI 10.1109/OCEANS.2014.7003008
- [ATM14] Amory, Ammar. ; Tosik, Thomas. ; Maehle, Erik.: A Load Balancing Behavior for Underwater Robot Swarms to Increase Mission Time and Fault Tolerance. In: *Proceedings of 28th International Parallel Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, 2014. <http://dx.doi.org/10.1109/IPDPSW.2014.146>. DOI 10.1109/IPDPSW.2014.146
- [AMO13] Amory, Ammar ; Meyer, Benjamin ; Osterloh, Christoph ; Tosik, Thomas ; Maehle, Erik: Towards Fault-Tolerant and Energy-Efficient Swarms of Underwater Robots. In: *18th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (2013)*. <http://dx.doi.org/10.1109/IPDPSW.2013.215>. – DOI 10.1109/IPDPSW.2013.215
- [FHK12] Forouher, Dariush ; Hartmann, Jan ; Klüssendorff, Jan H. ; Maehle, Erik ; Meyer, Benjamin ; Osterloh, Christoph ; Tosik, Thomas: HANSE - A Low-Cost Autonomous Underwater Vehicle. In: *AMS*, Springer, 2012 (Informatik Aktuell). – ISBN 978-3-642-32216-7, S. 147–155. http://dx.doi.org/10.1007/978-3-642-32217-4_16. DOI 10.1007/978-3-642-32217-4_16
- [RGM15] Renner, Christian ; Meyer, Benjamin ; Bimschas, Daniel ; Gabrecht, Alexander ; Ebers, Sebastian ; Tosik, Thomas ; Amory, Ammar ; Maehle, Erik ; Fischer, Stefan: Poster Abstract: Hybrid Underwater Environmental Monitoring. In: *12th ACM Conference on Embedded Networked Sensor Systems*, 2014 <http://dx.doi.org/10.1145/2668332.2668354>. DOI 10.1145/2668332.2668354