Abschlussdokumentation der Projektgruppe MOPS 2012/2013

Themensteller: Prof. Dr.-Ing. Axel Hahn
Betreut durch: Dipl.-Inform. Sascha Hornauer und Rainer Droste, M.Sc.

Vorgelegt von:
Björn Borgmann, Justus-Sebastian Bücker, Daniel Fay, Jan Friedrichs, Konstantin Franzuski, Kamran Ghanaat, Julian Janke, Niels Klissing, Matthias Larisch, Jan Mathis Manemann und Philip Weber

Inhaltsverzeichnis

Abkürzungsverzeichnis III

Abbildungsverzeichnis V

Tabellenverzeichnis VI

1. Einleitung 1
1.1. Motivation . . . 1
1.2. Ziel des Projektes . . . 1
1.3. Aufbau der Projektdokumentation . . . 2

2. Anforderungen 3
2.1. Tabellarische Ansicht der Anforderungen . . . 3
2.2. Einsatzszenario . . . 5
2.2.1. Das Szenario . . . 5
2.2.2. Die Mission . . . 5

3. Entwurf 7
3.1. Bootsformen . . . 7
3.2. Morphologischer Kasten . . . 10
3.3. Auswahl und Begründung . . . 10

4. Hardware 13
4.1. Aufbau des MOPS . . . 13
4.1.1. Rahmen . . . 14
4.1.2. Motoren . . . 16
4.1.3. Schwimmer und Segmente . . . 17
4.1.4. Akku-Segment . . . 17
4.2. Elektronik . . . 18
4.2.1. BeagleBone . . . 19
4.2.2. RTC . . . 19
4.2.3. Sensorik . . . 19
4.2.4. Aktorik . . . 20
4.2.5. Motorregler . . . 20
4.2.6. Kommunikation . . . 21
4.2.7. Energieversorgung . . . 22
4.2.8. Segmentaufbau . . . 23
4.3. Sensorkonzept . . . 24
4.3.1. Busnode . . . 24
4.4. Fazit und Ausblick . . . 25

5. Software 26
5.1. Kommunikation . . . 26
5.1.1. Allgemeine Implementation . . . 26

5.1.2. WLAN . . . 28
5.1.3. XBee . . . 28
5.1.4. High-Level-Protokoll . . . 29
5.2. Missionsplanungs- und -überwachungssoftware . . . 32
5.2.1. Controller . . . 32
5.2.2. Benutzeroberfläche . . . 33
5.2.3. Kommunikation . . . 34
5.2.4. Modell . . . 35
5.3. Linux Kurzreferenz . . . 37
5.3.1. Systembeschreibung . . . 37
5.3.2. systemd . . . 37
5.3.3. Hardwarekonfiguration . . . 37
5.4. Onboard . . . 38
5.4.1. Konfiguration . . . 39
5.4.2. Launcher . . . 40
5.4.3. Aktorik . . . 40
5.4.4. Navigationssystem . . . 40
5.4.5. Storage . . . 44
5.4.6. Payload-Control . . . 45
5.4.7. Mission-Control . . . 45
5.4.8. Systemüberwachung . . . 46
5.4.9. Positionsbestimmung . . . 48
5.4.10. Kommunikation . . . 49
5.4.11. Fernsteuermodus . . . 51
5.4.12. Logging . . . 51
5.5. Simulation . . . 52

6. Benutzerhandbuch **53**
6.1. Missionsplanungs- und -überwachungssoftware . . . 53
6.1.1. Installation . . . 53
6.1.2. Benutzeroberfläche . . . 53
6.2. Inbetriebnahme . . . 70

7. Fazit **74**

8. Ausblick **75**

Linkliste **VIII**

A. Anhang **IX**
A.1. Hardwarezusammensetzung . . . IX
A.2. Elektrische Komponenten . . . XII
A.3. Schaltplan Batteriesegment . . . XIV
A.4. Schaltplan Steuersegment . . . XV
A.5. Hardwarebeschaffung . . . XVI
A.6. CAD-Modell des MOPS . . . XXII

Abkürzungsverzeichnis

ASV	Autonomous Surface Vehicle
CAD	Computer-aided design
DHCP	Dynamic Host Configuration Protocol
GPS	Global Positioning System
ICBM	Institut für Chemie und Biologie des Meeres
KG	Kanalgrundrohr
MOPS	Marine Observation Platform for Surfaces
NTP	Network Time Protocol
PG	Projektgruppe
PVC	Polyvinylchlorid
PWM	Pulsweitenmodulation
RTC	Real Time Clock
SSID	Service Set Identifier

Abbildungsverzeichnis

4.1. CAD-Modell des MOPS . . . 13
4.2. Alurahmen des MOPS . . . 14
4.3. elektronischer Außenbordmotor Rhino-VX-34 . . . 16
4.4. Akku-Kiste . . . 18
4.5. Verbindungsstruktur der elektronischen Komponenten im MOPS . . . 23
4.6. Beaglebone-Aufsteckplatine mit Markierungen für Sensoren . . . 24

5.1. Ein grobes Klassendiagramm der Kommunikation mit einigen wichtigen Membern und Methoden. . . . 27
5.2. Erweiterung des *TransferObject* zur Übertragung von Missionen . . . 32
5.3. Schematische Darstellung der Software . . . 33
5.4. Schematische Darstellung der Controller der Software . . . 34
5.5. Schematische Darstellung der Benutzeroberfläche . . . 35
5.6. Struktur der Onboard-Software. Hinweis: Interaktionpfade mit dem Launcher und dem System-Monitor werden nicht dargestellt. . . . 39
5.7. Regelstrecke den *CourseDifferentialController* . . . 42
5.8. Struktur der Datenhaltung auf dem MOPS. . . . 44

6.1. Benutzeroberfläche nach dem Start der Anwendung . . . 54
6.2. Reiter zum Wechseln des Anwendungsbereichs . . . 54
6.3. Dialog zum Erstellen/Bearbeiten von Wegpunkten . . . 55
6.4. Dialog bei Fehlern während der Übertragung . . . 56
6.5. Bereich der Anwendung zur Überwachung einer Mission . . . 57
6.6. Fernsteuerungsauswahl . . . 59
6.7. Graphical Remote Control . . . 59
6.8. Bereich der Anwendung zum Einsehen der MOPS-Logdaten . . . 61
6.9. Menüleiste der Anwendung . . . 62
6.10. Menü „Datei“ . . . 62
6.11. Menü „Kommunikation“ . . . 63
6.12. Verbindung herstellen Dialog . . . 63
6.13. Menüeintrag *[Mission]* . . . 64
6.14. Dialog „Missions-ID“ . . . 64
6.15. Fahre zu Koordinate Dialog . . . 65
6.16. (Sub-)Menü „Log anfordern“ . . . 66
6.17. Dialog zum Auswählen eines Log-Bereiches . . . 66
6.18. Dialog zum Auswählen eines Bereiches . . . 67
6.19. Menü „Extras“ . . . 67
6.20. Menü „Tile-Updater“ . . . 68
6.21. Menü „Ansicht“ . . . 68
6.22. Menü „Optionen“ . . . 69
6.23. Sensor Verwaltung . . . 69
6.24. Statusleiste . . . 70
6.25. Montage der vorderen Querverbindung an den Mittelrumpf . . . 71

6.26. Montage der Querstreben 72
6.27. Montage der Antenne 73

A.1. CAD-Modell: Räumliche Ansicht XXII
A.2. CAD-Modell: Ansicht von oben XXIII
A.3. CAD-Modell: Ansicht von unten XXIV
A.4. CAD-Modell: seitliche Ansicht XXV
A.5. CAD-Modell-Rahmen: Räumliche Ansicht XXVI
A.6. CAD-Modell-Rahmen: Ansicht von oben XXVII

Tabellenverzeichnis

2.1. Tabellarische Ansicht der Anforderungen 5

3.1. Morphologischer Kasten 10

4.1. Übersicht der Rahmenbestandteile 15
4.2. Bedeutung der Pulsbreite eines RC-PWM-Signals 21
4.3. Versorgungsspannungen im Steuersegment 22

5.1. Auswahl einiger XBee API Nachrichtentypen 29
5.2. ByteArray-Nachrichten 30
5.3. Positive Acknowledges 31
5.4. Auflistung verfügbarer Einstellungen für einen IO-Pin [3] 38
5.5. Werte zur Ermittlung der Stromaufnahme 46

A.1. Übersicht der Hardwarebestandteile XI
A.2. Übersicht der Ersatzteile XII
A.3. Übersicht Teilliste Antrieb XVII
A.4. Übersicht Teilliste Rumpf- und Bugsegment XVIII
A.5. Übersicht Teilliste Steuerungssegment XIX
A.6. Übersicht Teilliste Energiesegment XX
A.7. Übersicht Teilliste Rahmen XXI

1. Einleitung

Die *Marine Observation Platform for Surfaces* (MOPS) wurde im Rahmen einer einjährigen Projektgruppe von Masterstudierenden aus dem Fachbereich Informatik der Universität Oldenburg realisiert. Das Ziel dieses Projekts war die Entwicklung eines *Autonomous Surface Vehicle* (ASV) zur eigenständigen Durchführung von geplanten Messungen an der Oberfläche von Binnen- und Hochseegewässern.

Das Projekt ist in Kooperation mit dem Institut für Informatik, dem OFFIS und Prof. Dr. Oliver Zielinski, Professor für Marinesensorsysteme am ICBM, durchgeführt worden.

1.1. Motivation

Unser Planet besteht zu 71 % aus Wasser. Dadurch ergibt sich ein weites Forschungsfeld rund um die Meeresbiologie. Jedoch gibt es in diesem Bereich viele Probleme, denen sich Wissenschaftler auch in der heutigen technisch und elektronisch unterstützten Welt stellen müssen. Ein Problemfeld stellt die Beobachtung von Marineökosystemen dar. Zwar gibt es bereits existierende Lösungen, wie z. B. Forschungsschiffe, Bojen oder Systeme, die direkt auf dem Meeresgrund platziert werden, diese Lösungen sind allerdings teuer oder nicht flexibel nutzbar. Ein weiterer Aufgabenbereich ist z. B. die Analyse der Schelfmeere. Wissenschaftler möchten hierbei u. a. die Strömung an bestimmten Stellen bestimmen. Hierfür sind Prüfungen nötig, um zu sehen, ob ein Teilbereich eine geringe Wassertiefe oder Trübstoffe enthält.[1]

1.2. Ziel des Projektes

Das Ziel dieser Projektgruppe war die Entwicklung eines autonomen Bootes. Das entwickelte Boot soll in der Lage sein, Messungen von physikalischen und chemischen Parametern zwischen Wangerooge und Helgoland durchzuführen. Es soll kostengünstig sein und zudem für die Schiffe in seinem Einsatzgebiet keine Gefahr darstellen.[2]
Für die oben genannten Zielvorstellungen stellt das Projekt MOPS einen ersten Lösungsansatz dar, um den Forschern die Arbeit zu erleichtern, sodass sie sich vorrangig mit dem Auswerten der aufgenommenen Daten beschäftigen können. Der bisher nötige Zeit- und Kostenaufwand, die Daten an den verschiedenen Messstellen aufzunehmen, soll durch das

[1] Vgl. Projektvorstellung MOPS 2012
[2] Vgl. Projektvorstellung MOPS 2012

autonome Fahrzeug minimiert werden. Es soll neben dem autonomen Modus noch über einen manuellen Modus verfügen, um z. B. Messstellen, welche schwer autonom erreichbar sind, manuell ansteuern oder Anlegemanöver durchführen zu können. Die beiden aufgeführten Anforderungen sind im Rahmen einer Anforderungsliste mit dem Auftraggeber besprochen worden. Weitere Anforderungen finden sich in Kapitel 2 dieser Projektdokumentation.

1.3. Aufbau der Projektdokumentation

Nach der Einleitung, in der u. a. die Motivation und das Ziel des Projektes mit einigen Anforderungen vorgestellt wurden, beschäftigt sich Kapitel 2 mit der Anforderungsliste des Auftraggebers. Hier werden in einer tabellarischen Übersicht alle aufgenommenen Anforderungen noch einmal festgehalten, bevor im nächsten Kapitel 3 die verschiedenen Bootsformen, welche in der Entwurfsphase betrachtet wurden, ersichtlich sind. Des Weiteren sind durch einen morphologischen Kasten Kombinationen aufgezeigt, die der Bootsauswahl bzw. -entwicklung dienten. Zum Schluss wird die Bootsauswahl noch begründet.

Im Kapitel 4 sind die Aufbaukomponenten, aus denen der MOPS besteht, erläutert. Weiterhin wird die Elektronik, die Sensorik und die Energieversorgung beschrieben. Das Kapitel 5 beschreibt Aufbau und Umsetzung aller Softwarekomponenten, welche zur Funktion des MOPS notwendig sind. Es folgt Kapitel 6, welches die Missionsplanungs- und -überwachungssoftware und die Inbetriebnahme des Systems mit Schritt-für-Schritt Anleitung für den Anwender beschreibt.

Das im Anschluss nach dem Benutzerhandbuch aufgeführte Fazit und der Ausblick bilden den Abschluss dieser Projektdokumentation. Durch die im Anhang hinterlegten Übersichtstabellen zur Hardwarezusammensetzung und Hardwarebeschaffung ist u. a. ersichtlich, aus welchen Komponenten der MOPS besteht und welche Kosten hierbei angefallen sind. Des Weiteren werden durch entsprechende Schaltpläne und durch das CAD-Modell nähere Details des MOPS kenntlich.

2. Anforderungen

Im folgenden Abschnitt wird auf die Anforderungen an das System eingegangen. Diese wurden in Zusammenarbeit mit Prof. Zielinski vom ICBM in Oldenburg und in der Projektgruppe erarbeitet. Die Reihenfolge/ID ist dabei nicht gleichbedeutend mit der Wichtigkeit der jeweiligen Anforderung. Anschließend wird zur Verdeutlichung ein beispielhaftes Einsatz-Szenario beschrieben.

2.1. Tabellarische Ansicht der Anforderungen

ID	Name	Beschreibung
00	Schwimmfähigkeit	Das ASV soll auch bei dem im Einsatzraum zu erwartenden Seegang schwimmfähig sein, ohne das Komplikationen auftreten, welche zum Sinken des ASV oder dem Ausfall der Elektronik/Messsensorik führen können. Hierbei muss auch auf die Auswahl eines geeigneten Baumaterials und einer geeigneten Bootsform geachtet werden.
01	Größe	Bei der Größe des ASV ist, zum jetzigen Zeitpunkt, keine genaue Angabe möglich. Jedoch sollte es in einem Auto transportierbar sein. Dabei ist bei einem größeren ASV auch eine Bauweise aus mehreren kleinen Modulen zum Zusammensetzen denkbar.
02	Gewicht	Ähnlich wie bei der Größe ist beim Gewicht keine genaue Angabe möglich, da das Gewicht vor allem von der Bootsform und dem Material abhängt. Jedoch sollte das ASV von max. zwei Personen tragbar sein.
03	Kollisionsvermeidung	Das ASV soll in der Lage sein, Kollisionen auf See zu vermeiden. Hierzu gehören andere Boote, Bojen, Treibgut sowie weitere Objekte im Wasser.

04	Selbstaufrichtend	Im Falle des Kenterns soll das ASV in der Lage sein, sich selber aufzurichten. Dies wird durch eine angemessene Bootform sichergestellt. Im Falle von Bootsformen, bei denen Kentern keine Rolle spielt, kann dieser Punkt vernachlässigt werden.
05	Mögliche Einsatzzeit	Die mögliche Einsatzzeit des ASV sollte eine Tagfahrt (8-12 Stunden) betragen.
06	Messsensorik	Das ASV soll durch eine modulare Bauweise dazu in der Lage sein jegliche Sensorik tragen zu können, die durch ihre Schnittstelle und ihr Gewicht vom ASV unterstützt werden kann. Um auf die Anforderung Navigation (10) reagieren zu können, soll das ASV Strömungs- und Windrichtung bestimmen und nutzen können.
07	Antrieb	Der Antrieb des ASV soll auch bei der in möglichen Einsatzgebieten zu erwartenden Strömung dazu in der Lage sein, das ASV anzutreiben. Eine genaue maximale Geschwindigkeit ist (bisher) nicht abschätzbar. Außerdem sollen der Antrieb und die zugehörige Technik möglichst leichtgewichtig sein. Des Weiteren soll das ASV an gegebenen Wegpunkten verweilen können. (Siehe auch Anforderung 10 - Navigation)
08	Autonomes Fahren	Das ASV soll in der Lage sein, autonom Koordinaten an- bzw. Routen abzufahren. Dazu muss auch auf die Anforderungen Kollisionsvermeidung (03), Navigation (10) und Kommunikation (11) geachtet werden.
09	Navigation	Das ASV soll unter Berücksichtigung von Strömung und Wind in der Lage sein, übergebene Koordinaten anzufahren sowie an Koordinaten zu verweilen. (Siehe auch Anforderungen 03, 08, 11)
10	Kommunikation	Das ASV soll in der Lage sein, per Funk zu kommunizieren. Die mindestens benötigte Entfernung für die Kommunikation mit dem ASV ist bisher nicht abschätzbar, ergibt sich jedoch aus dem Einsatzszenario.
11	Energieversorgung	Die Energieversorgung soll dazu in der Lage sein die unter Anforderung 05 genannte Einsatzzeit überbrücken zu können.

12	Rechtliche Bedingungen	Das ASV darf keine der rechtlichen Rahmenbedingungen brechen. Diese sind allerdings auch vom jeweiligen Einsatzort abhängig.
13	Steuerung	Das ASV soll zur Navigation im Nahbereich über eine Möglichkeit verfügen, direkt steuerbar zu sein.
14	Bediensoftware	Es soll eine Bediensoftware existieren, mit der Missionen geplant, an das ASV übertragen und überwacht werden können.
15	Sensordaten	Die aufgenommenen Sensordaten sollen über eine Schnittstelle abrufbar sein.

Tabelle 2.1.: Tabellarische Ansicht der Anforderungen

2.2. Einsatzszenario

Zur Verdeutlichung der Ziele des ASV MOPS, erfolgt die Definition eines kurzen Einsatzszenarios. Hierbei wird eine Mission definiert, bei der der MOPS wichtige Messdaten sammelt. Weiterhin wird die Missionsplanung, sowie die Missionsdurchführung aufgezeigt.

2.2.1. Das Szenario

Eine Kläranlage in der Nähe einer Küste entsorgt bei einer zu hohen Menge an zu klärendem Abwasser einen Teil des Abwassers über eine Pipeline ins nahegelegene Meer. Dadurch wird ein Überlaufen der Tanks verhindert, welches zu großen Schäden innerhalb der Anlage führen würde. Dieses geschieht regelmäßig zwischen 9 und 10 Uhr am Morgen, wenn in den Haushalten und naheliegenden Betrieben viel Abwasser produziert wird. Um die Auswirkungen dieser Maßnahme auf die Küstengewässer zu untersuchen, hat das verantwortliche Umweltamt eine Messung der Trübstoffe innerhalb des Wassers an dem betroffenen Küstenabschnitt in Auftrag gegeben.

2.2.2. Die Mission

Der MOPS soll mit der notwendigen Sensorik ausgestattet werden, um den pH-Wert und Trübstoffe im Wasser messen zu können. Innerhalb eines Tages soll der MOPS fünf Koordinaten anfahren und an jedem der Punkte Messungen durchführen. Zwei der Koordinaten sollen vor dem Einleiten des Abwassers erreicht werden, damit die dort gesammelten Messdaten die Werte unter Normalbedingungen repräsentieren. Der dritte Punkt liegt vor der Pipeline. Hier soll der MOPS zwischen 8:30 und 10:30 Uhr verweilen und während dieser Zeit alle 2 Minuten Messungen durchführen, um die Belastung während des Ableitens des Abwassers zu ermitteln.

Um die erforderlichen Messdaten sammeln zu können, muss die passende Sensorik in das dafür vorgesehene Segment des MOPS eingebaut und mit der Bordelektronik verbunden werden. Dabei ist darauf zu achten, alle notwendigen Komponenten wasserfest zu verschließen, was bei den vorgesehenen Segmenten problemlos vornehmbar ist. Bevor der MOPS zu Wasser gelassen wird, sollte noch ein Systemcheck durchgeführt werden.

Die Missionsplanung wird mit der speziell entwickelten Missionsplanungs- und -überwachungssoftware durchgeführt, welche im Abschnitt 6.1 detailliert beschrieben wird. Als erstes muss die geplante Route erstellt werden, die der MOPS auf seiner Mission abfahren soll. Dazu steht in der Software das passende Kartenmaterial bereit, sodass auf der Karte die Route geplant und als GPS-Koordinaten später auf den MOPS übertragen werden kann. Je nach Bedingung auf dem Wasser, wie z. B. starker Wind oder Strömung, ist es sinnvoll und notwendig einen Toleranzbereich in Form eines Radius um den Routenpunkt anzugeben bei dem der Punkt als erreicht gilt. Auch wenn die autonome Navigation des MOPS mit den Einflüssen einer Strömung und des Windes umgehen kann, so ist es dennoch sinnvoll, bereits während der Routenplanung auf der Seekarte die herrschende Strömung zu berücksichtigen und vielleicht sogar energieeffizient auszunutzen.

Ist es notwendig, über einen längeren Zeitpunkt an einem Ort zu verweilen, wie z. B. in diesem Szenario über einen Zeitraum von 120 Minuten vor der Pipeline, so lässt sich auch dies in der Missionsplanung definieren. Dazu ist wieder ein Toleranzbereich anzugeben, in dessen Radius der MOPS trotz Strömung oder Wind verbleibt.

In dem Beispielszenario müssten fünf Punkte angelegt werden. Als erstes zwei Messpunkte in der Nähe der Pipeline, an dem die Trübung des Wassers und der pH-Wert gemessen werden soll. Anschließend wird ein weiterer Punkt mit einer Verweildauer von 120 Minuten vor der Pipeline angelegt. Zuletzt werden die Punkte 4 und 5 angelegt, an denen der MOPS wie bei den ersten beiden Punkten Messungen durchführen soll. Ist die Mission abgeschlossen, so fährt der MOPS zu seiner Zielkoordinate. Für das Manövrieren im Nahbereich ist es auch möglich, den MOPS direkt zu steuern und ihn auf diese Weise zielgenau anzulegen.

3. Entwurf

Im folgenden Abschnitt wird zunächst näher auf die verschiedenen Bootsformen eingegangen und anhand von Vor- und Nachteilen sollen die einzelnen Formen bzw. Architekturen von Booten aufgezeigt werden. Im Anschluss hierauf erfolgt eine Aufstellung in Form eines morphologischen Kastens, in der alle Formen mit den verschiedenen Antriebsmöglichkeiten, die in der Entwurfsphase angesprochen wurden, ersichtlich sind. Anschließend werden alle in Frage kommenden Kombinationen ausgewählt und wiederum durch Vor- und Nachteile evaluiert. Zum Schluss wird die Bootsauswahl begründet.

3.1. Bootsformen

Bei den Bootsformen gibt es verschiedene Rumpftypen, die jeweils Vor- und Nachteile aufweisen. Welche Rumpftypen die Projektgruppe näher betrachtete, wird hier aufgeführt.

Einrümpfer

- Verdräger
  - Vorteile (3):
    - gut bei Geradeauslauf und Seitenwind
    - energieeffizienter als Gleiter
    - Platz für Solar
  - Nachteile (4):
    - am besten schmal und lang (Platz für Solar)
    - schlecht manövrierfähig
    - geringe Rumpfgeschwindigkeit
    - abhängig vom mitgeführten Gewicht
- Gleiter
  - Vorteile (2):
    - schneller als Verdränger
    - Platz für Solar
  - Nachteile (2):
    - höherer Energieverbrauch, da stärkerer Motor nötig
    - abhängig vom mitgeführten Gewicht

Segelboot

- Jolle
  - Vorteile (2):
    - formstabil
    - hohe Breite (Solar)
  - Nachteile (1):
    - Selbstaufrichtung nicht möglich
- Yacht
  - Vorteile (3):
    - Selbstaufrichtung möglich
    - ist gewichtstabil (Gewicht unten bsp. durch Batterien)
    - Platz für Solar
  - Nachteile (1):
    - Breite max. viertel der Länge

Mehrrumpfer

- Zweirümpfer
  - Vorteile (2):
    - breitere Bauform - mehr Platz
    - schnell und wenig Tiefgang
  - Nachteile (3):
    - Selbstaufrichtung nicht möglich
    - zwei Motoren, vom Deck hängender Motor oder Segel
    - weniger manövrierfähig
- Dreirümpfer
  - Vorteile (3):
    - breite Bauform - mehr Platz
    - schnell und wenig Tiefgang
    - Motor im mittleren Rumpf
  - Nachteile (1):
    - Selbstaufrichtung nicht möglich

Sonderformen

- T-Form(SAUV2) [8]
  - Vorteile (5):
    - große Solarfläche
    - Selbstaufrichtung möglich
    - geringer Widerstand
    - gut manövrierbar

    - keine Windangriffsfläche
  - Nachteile (2):
    - neu und damit wenig Erfahrung
    - Platzproblem
- U-Boot
  - Vorteile (1):
    - hydrodynamische Bauform (stromlinienförmig)
  - Nachteile (1):
    - wenig Platz für Solar
- Mondfisch-Form
  - Vorteile (2):
    - steht gut im Wasser
    - Selbstaufrichtung möglich
  - Nachteile (2):
    - schlecht manövrierfähig
    - Platzproblem
- Wie Sonobot [7]
  - Vorteile (2):
    - ähnlich wie Katamaran, dadurch bei breiter Bauweise stabil im Wasser (nur bei einer sehr breiten Bauweise)
    - Platz für Solar
  - Nachteile (2):
    - bei schmaler Bauweise kann es bei stärkerem Wellengang zum Umkippen kommen
    - Schwerpunkt liegt oberhalb der Wasseroberfläche

3.2. Morphologischer Kasten

Im weiteren Verlauf der Bootsauswahl sollten Bootsform und Antriebsmöglichkeiten gegenübergestellt werden. Die Darstellung erfolgt als morphologischer Kasten, welcher in Tabelle 3.1 abgebildet ist.

Antrieb / Bootsform	Schraube	Jet-stream	Segel (Stoff)	Segel (Starr)	Kite	Flettner Rohr	Schlängeln
Motorboot	x	x	x	x	x	x	o
Gleiter	x	x	x	x	x	x	o
Jolle	x	x	x	x	x	x	o
Yacht	x	x	x	x	x	x	o
Zweirümpfer	x	x	x	x	x	x	o
Dreirümpfer	x	x	x	x	x	x	o
T-Form	x	x					o
U-Boot	xS						x
Mondfisch-form							o
Sonobot							x
Zwei Rohre	x						

- S: Solarfläche nicht nutzbar
- x und o: Aufbau möglich / nicht möglich

Tabelle 3.1.: Morphologischer Kasten

3.3. Auswahl und Begründung

Zur Eingrenzung der Bootsauswahl entschieden wir uns zunächst für diese vier Kombinationen, die wiederum durch Vor- und Nachteile evaluiert worden sind.

1. Motorbootform: Antrieb mit Schraube (und Ruder) bzw. Vektorantrieb
   - Vorteile (1):
     - Selbstaufrichtung möglich
   - Nachteile (2):
     - durch Beschränkungen der Transportfähigkeit (Breite) vergleichbar geringe Stabilität
     - bei modularem Aufbau des Rumpfes: hohe Anforderungen an die Dichtigkeit
2. U-Boot mit Schraubenantrieb
   - Vorteile (2):

  - sehr aquadynamisch
  - Selbstaufrichtung möglich (bei entsprechender Gewichtsverteilung)
- Nachteile (2):
  - wenig Platz für Solarzellen
  - wenig Platz im Innenraum

3. Dreirümpfer mit Schraube/ Segel
   - Vorteile (5):
     - sehr hohe Stabilität
     - Antrieb durch bis zu drei Motoren
     - bei zwei oder mehr Motoren: Einsparung des Ruders
     - modularer Aufbau für bessere Transportfähigkeit
     - Segel ist einsetzbar
   - Nachteile (1):
     - Selbstaufrichtung nicht möglich
4. T-Form(SAUV2) mit Schraube
   - Vorteile (4):
     - Selbstaufrichtung möglich
     - große Solarfläche
     - geringe Widerstand
     - gut manövrierfähig
   - Nachteile (2):
     - höhere Anforderungen an die Dichtigkeit
     - wenig Erfahrung

Begründung:

Bootsform: Nach Abwägung aller Kombinationen ist die Bootsform Dreirümpfer mit Schraube ausgewählt worden, weil diese Bootsform eine hohe Stabilität bietet, mehrere Motoren eingesetzt werden können, ein modularer Aufbau möglich ist und alternativ, neben einem Motorantrieb, ein Segel einsetzbar ist. Eine weitere Alternative stellte die T-Form (SAUV2) dar, jedoch ist diese Bootsform recht neu, sodass mögliche Hilfestellungen mangels Erfahrungen nicht möglich wären. Im Hinblick auf die Anforderungsliste stellte das Trimaran-Konzept die beste Lösung dar.

Bootsgröße: Eine weitere Entscheidung, welche im Rahmen dieser Phase beschlossen wurde, ist die Bootsgröße gewesen. Nachdem zunächst mit dem Auftraggeber keine festgelegte Größe des Bootes vereinbart wurde, entschieden wir uns während der Besprechung des Grundkonzeptes anfangs auf einen 2 m x 2 m breiten Grundrahmen. Diese Größe hatte aus unserer Sicht mehrere Vorteile. Beispielsweise ist der MOPS durch diese Größe Hochseetauglich in seinem Einsatzgebiet der Nordsee. Ferner ist die Stabilität damit gewährleistet, sodass ein Kentern oder Abtreiben in den Wellen minimiert wird. Der MOPS sollte außerdem nicht zu groß ausfallen, um einen unkomplizierten Transport in einem Anhänger zu garantieren.

4. Hardware

Das folgende Kapitel zeigt die Hardware- und Elektronikkomponenten des MOPS in seiner aktuellen Konfiguration auf. Neben der Beschreibung der Schwimmkörper, des Rahmens, der Segmente, des Akkus und der Motoren im ersten Teil dieses Kapitels, erfolgt im zweiten Teil die Betrachtung der Elektronikkomponenten, welche zur Steuerung, sowie der Informations- und Kommunikationsverarbeitung des MOPS dienen. Zusätzlich findet das Sensorkonzept, welches angedacht ist, eine Erläuterung.

4.1. Aufbau des MOPS

Dieser Abschnitt beschreibt den Aufbau und die aktuelle Konfiguration des Hardwareaufbaus des MOPS. Hierbei wird auf die Rahmenbestandteile mit Motoraufhängung, Schwimmer und Segmente eingegangen. Zur Verdeutlichung des Aufbaus dient das CAD-Modell in der folgenden Abbildung 4.1. Weitere Abbildungen befinden sich im Anhang A.6.

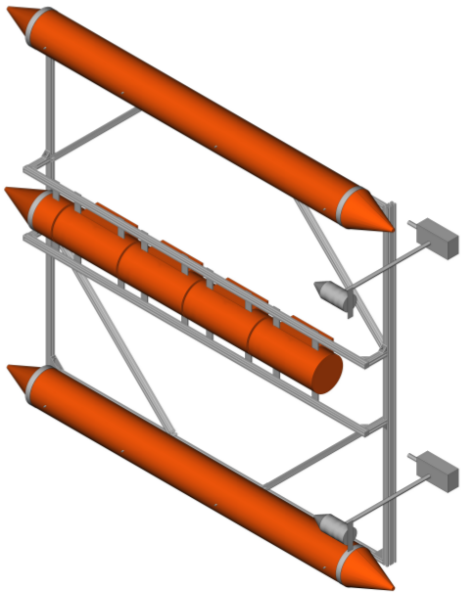


Abbildung 4.1.: CAD-Modell des MOPS

Der Aufbau zeigt einen ca. 2,00 m breiten und 2,50 m langen Trimaran bestehend aus KG-Elementen, wobei die Rümpfe rechts und links als Schwimmkörper dienen und der mittlere Rumpf die elektronischen Komponenten fasst. Der mittlere Rumpf besteht aus einzelnen modularen Komponenten, welche flexibel ausgetauscht werden können. Die Schwimmer bestehen aus jeweils einem DN 200 (200 mm Durchmesser) KG-Rohr in der Länge von 2,00 m, welche an beiden Enden einen Trichter als Bug angebracht bekommen haben. Angetrieben wird das autonome Wasserfahrzeug mit Hilfe von elektrischen Außenbordmotoren, welche sich am hinteren Ende befinden. Für die notwendige Stabilität sorgt ein Rahmen aus 30 mm x 30 mm Aluprofilleisten vom Hersteller „ITEM24“ [10]. Eine ausführliche Auflistung aller verbauten Einzelteile ist im Anhang A.1 zu finden. Die nachfolgenden Abschnitte erläutern die einzelnen Komponenten im Detail.

4.1.1. Rahmen

Der erste Prototyp wurde als „Proof of Concept“ mit einem günstigen Holzrahmen bestehend aus Fichtenholzprofilen realisiert. Nachdem der MOPS in diesem Aufbau ausgiebig getestet wurde, erfolgte der Austausch des Fichtenholzrahmens durch einen stabileren 30 mm x 30 mm Aluprofilrahmen, um die Hochseetauglichkeit gewährleisten zu können. Die Abbildung 4.2 zeigt hierbei den Aufbau des Rahmens.

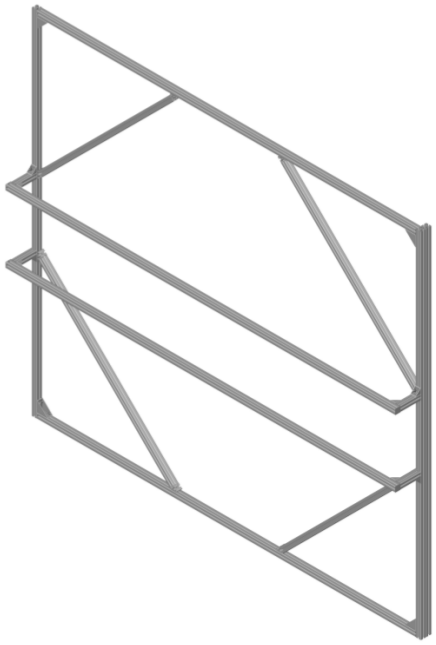


Abbildung 4.2.: Alurahmen des MOPS

Der Rahmen besteht aus einem äußeren Rahmen, welcher als tragendes Element für die Schwimmer und die Motoren dient, sowie einem inneren Rahmen für die einzelnen Seg-

mente. Mit Hilfe einzelner Winkelsätze werden die einzelnen Rahmenelemente miteinander verschraubt. Aus Gründen der Stabilität wurden zusätzlich zu den vorgesehenen Winkelelementen 1050 mm lange Querstreben in den vier Ecken angebracht. Diese Querstreben werden mit einem ITEM-Universalverbinder am Außenrahmen befestigt. In Tabelle 4.1 folgt eine Übersicht aller Rahmenteile.

Stückzahl	**Bezeichnung**	**Anmerkung**
3	Profil 6 30 mm leicht, natur 2000 mm	2 Profile für das Heck (Doppelung wegen Motorenaufhängung) und 1 Profil für den Bug
2	Profil 6 30 mm leicht, natur 2200 mm	Profile für den Segmentrahmen
2	Profil 6 30 mm leicht, natur 2140 mm	Profile für die Befestigung der Schwimmer
4	Profil 6 30 mm leicht, natur 115 mm	Verbindungsprofile zwischen Außenrahmen und dem Segmentrahmen
4	Winkelsatz 30 mm	größerer Winkelsatz zur Verbindung, bestehend aus Winkel, vier Nutensteinen und vier 16 mm Schrauben
8	Winkelsatz 30 mm	kleinerer Winkelsatz zur Verbindung, bestehend aus Winkel, zwei Nutensteinen und zwei 16 mm Schrauben
8	Abdeckklappen 30 mm	Abdeckkappen für Profilenden
30	Nutenstein 6 M6	Nutensteine zur Verbindung der Segmente an den Rahmen
8	Winkel T2 45 grad	45 Grad Winkel für stabilisierende Querstreben
4	Profil 6 30 mm leicht, natur 1050 mm	stabilisierende Querstreben
4	Halbrundschraube M6 x 16	Halbrundschrauben für T2 Elemente

Tabelle 4.1.: Übersicht der Rahmenbestandteile

Dieser Alurahmen ist trotz seiner Abmessungen recht stabil und weist dabei lediglich ein Gewicht von 0,93 kg pro Meter der Aluprofile, also ein Gesamtgewicht von ca. 20 kg, auf. Zudem bietet der geringe Querschnitt der Profile nur einen geringen Widerstand für Wind und Wellen. Ein weiterer Vorteil der Nutzung eines solchen Alurahmens ist die modulare Erweiterung, welche dadurch realisiert wird, dass die Befestigung neuer Elemente mit Hilfe von verschiebbaren Nutensteinen erfolgt. Diese Nutensteine können in die Nut der Aluprofile eingesetzt werden und ermöglichen das Verbauen weiterer Elemente an den Profilleisten. Somit kann eine Erweiterung oder ein Umbau des Alurahmens leicht vorgenommen werden.

4.1.2. Motoren

Der Antrieb des autonomen Wasserfahrzeugs wird mit Hilfe von zwei elektrischen Außenbordmotoren des Typs „Rhino-VX-34“ [20] realisiert. Dieser Motor ermöglicht den Antrieb von bis zu 1100 kg schweren Booten und besitzt 5 Schaltstufen vorwärts und 2 Schaltstufen rückwärts bei einer Versorgungsspannung von 12 V. Weiterhin ist die Eintauchtiefe des ca. 20 cm großen Propellers flexibel einstellbar. Die Motoren (siehe Abbildung 4.3)

Abbildung 4.3.: elektronischer Außenbordmotor Rhino-VX-34

werden an eine eigens konstruierte Motorenbefestigung am Heck des MOPS angebracht. Die Motorenbefestigung besteht hierbei aus zwei übereinander gesetzten 30 mm Aluprofilen, an denen Siebdruckplatten aus Holz befestigt wurden, um eine breite und rutschfeste Oberfläche herzustellen, auf welcher die Motoren mit der angebrachten Schraubzwinge befestigt werden können. Die Siebdruckplatten eignen sich für den Einsatz am MOPS besonders gut, da diese wasserfest beschichtet sind. Die Befestigung der Platten erfolgt mit rostfreien Schlossschrauben der Größe M6.

Die Gründe zur Entscheidung für dieses Antriebskonzept sind vielfältig. Als ersten Grund ist der hohe Schub der zwei Rhino Motoren zu nennen, welcher den MOPS auch bei schwierigen Verhältnissen auf eine ausreichende Geschwindigkeit von knapp 3 kn bringt. Dies wird durch eine Optimierung der mitgelieferten Schiffsschraube (Durchmesser, Anzahl und Steigung der Flügel) auf den Motor (Drehzahl und Leistung) durch den Hersteller erreicht.

Das ursprüngliche Antriebssegment sah ein mittels KG-Rohren selbst gebautes Segment vor, in dem ein kleinerer bereits vorhandener Elektromotor verbaut wurde, der den MOPS mittels einer Modellbau-Schiffsschraube antrieb. Zur Steuerung war ein Ruder vorhanden. Jedoch wurde schnell festgestellt, dass vorhandene und verfügbare Bauteile wie Motor, Welle und Propeller für ein Wasserfahrzeug der Größe des MOPS nicht ausreichend sind.

Die Entwicklung und Beschaffung eines passenden Antriebskonzepts wäre fehleranfällig und von fragwürdigem Erfolg gewesen. Aus diesem Grund wurde auf die Elektroaußenbordmotoren zurückgegriffen.

Aufgrund der Verwendung von zwei Motoren, welche am Heck nebeneinander angebracht sind, wird kein Ruder benötigt. Durch Veränderung des Schubverhältnisses der Motoren kann die Fahrtrichtung verändert und sogar auf der Stelle gedreht werden.

4.1.3. Schwimmer und Segmente

Als Schwimmkörper des Trimarans dienen zwei DN 200 KG-Rohre [24] in der Länge von 2,00 m. Diese Schwimmkörper sind mit Hilfe von jeweils 5 Gewindestangen der Größe *M8* an die Aluminiumprofile des Rahmens angebracht und mit Muttern fixiert. Das Volumen der Schwimmer wurde komplett mit Verpackungsmaterial aus Styropor ausgefüllt und die offenen Enden des KG-Rohrs mit den vorgesehen KG-Deckeln [25] verschlossen. Um eine Dichtigkeit der Schwimmer gewährleisten zu können, erfolgt der Verschluss aller möglichen undichten Stellen (wie Bohrlöcher, etc.) mit dem maritimen Dichtmittel *Sikaflex 291* [22]. Da es sich bei den KG-Elemente um PVC-U Kunststoff handelt, ist eine Vorbehandlung mit dem Sikaflex-Aktivator und dem Sikaflex-Multiprimer zwingend erforderlich, um eine Haftfähigkeit an den KG-Elementen gewährleisten zu können. Zur Verringerung der Reibung an Bug und Heck wird hier jeweils ein Trichter angebracht. Diese Trichter sind mit zwei-komponentigem PU-Schaum, welcher im Schiffsbau verwendet wird, ausgefüllt und mit Sikaflex an die Deckel der Schwimmer angeklebt. Eine zusätzliche Fixierung erfolgt durch Befestigung einer Metallschelle.

Für die einzelnen modularen Segmente des Mittelrumpfs werden *Wartungsrohre* [26] in DN 200 genutzt. Diese Wartungsrohre eignen sich besonders gut, da sie auf der Oberseite einen abnehmbaren Deckel aufweisen und somit einen Eingriff in das Innere des Segments nach Einbau ermöglichen. Die Länge eines Segments beträgt etwa 56 cm und hat hierbei einen Durchmesser von 200 mm. Pro Segment sind vier Edelstahlwinkel angebracht, welche zur Befestigung an den Segmentrahmen dienen. Wie bei den Schwimmkörpern erfolgt hier das Verkleben der Deckel sowie Abdichten der Bohrlöcher mit Hilfe von *Sikaflex 291*. Diese Segmente fassen hierbei alle elektronische Komponenten (im sogenannten Steuersegment), sowie die Sensorik. Bei der Umsetzung wurde darauf geachtet, mit nur wenigen Handgriffen einzelne Segmente austauschen zu können.

4.1.4. Akku-Segment

Als Akku dient aktuell ein 12 V Bleiakku, welcher über eine Kapazität von 108 Ah verfügt. Die Abmessungen sind: Länge 331 mm, Breite 174 mm, Höhe 218 mm bei einer Masse von etwa 26 kg. Dieser Akku befindet sich in einer wasserdichten Box, speziell für Akkus im Schiffsbereich, und verfügt über mehrere Kabeldurchführungen, die zu den beiden Antrieben und dem Steuersegment führen. Der Deckel der Box wurde erweitert, sodass auch

der Motorcontroller (siehe 4.2.5) Platz in der Box findet. Die gesamte Box steht auf einer Siebdruckplatte, welche auf dem inneren Rahmen mittels Nutensteinen verschraubt ist. Winkel und ein Spanngurt fixieren die Box auf der Platte.

Abbildung 4.4.: Akku-Kiste

Als alternative Energieversorgungen gibt es verschiedene Technologien, welche die PG untersucht hat. Am nächstliegendsten sind andere Akku-Technologien, wie z. B. Lithium-Polymer Akkus. Diese besitzen eine um Faktor 3-4 höhere volumetrische und gravimetrische Energiedichte. Ebenfalls gibt es sie in verschiedenen Bauformen, sodass eine kompakte Unterbringung in den Segmenten möglich ist. Die großen Nachteile sind allerdings der hohe Preis und die große Empfindlichkeit dieser Akkus. Bei Überladung droht ein Brand, welcher aufgrund der hohen nötigen Kapazitäten sehr gefährlich sein kann. Zum sicheren Umgang mit diesen Akkus sind entsprechende Batteriemanagementsysteme, bestehend aus einem Schutz gegen Überladung, Tiefentladung sowie zu hohe Ströme jeder Einzelzelle, notwendig.

Weitere Technologien wären z. B. die Brennstoffzelle. Jedoch wären diese Zellen ebenfalls wieder sehr teuer und für den Zweck des ASV zu überdimensioniert. Ähnlich wie z. B. bei einem Generator auf Diesel- oder Ethanol-Basis würde bei einer Brennstoffzelle das Problem der Sauerstoffversorgung und der eventuell entstehenden Wärme- und Abgas-Emissionen entstehen. Als Ergänzung wären regenerative Energien eine sinnvolle Idee, um den Stromverbrauch zu senken. Dazu gab es Überlegungen bezüglich Solartechnik, wie auch der Nutzung von Wellen- und Windenergie.

4.2. Elektronik

Der MOPS soll ein autonomes Wasserfahrzeug mit Potenzial für neue Erweiterungen werden. Damit dieses Konzept umgesetzt werden kann, werden verschiedene Sensoren, genügend Rechenleistung, Speicherplatz zur Auswertung und Messwertaufnahme sowie eventuell Kartenmaterial benötigt. Um bereits zu Beginn genügend Kapazität für Erweiterungen zu haben, wird als zentrale Steuereinheit das embedded Linux Board BeagleBone [4] eingesetzt.

4.2.1. BeagleBone

Das BeagleBone zeichnet sich durch eine hohe Rechenleistung, vergleichbar mit Smartphones der Generation 2010/2011, aus. Im Gegensatz zu anderen Boards wie dem Raspberry PI [18] bietet es zudem eine Reihe an leicht erreichbaren, für diesen Einsatzzweck brauchbaren Schnittstellen:

- 5x UART (3,3 V)
- 2x I2C
- CAN (externer Transceiver benötigt)
- ADC
- eine Vielzahl zusätzlicher GPIOs
- USB-Host
- Ethernet

Das Board ist mit 256 MB DDR2-Ram bestückt. Als Datenspeicher kommt eine handelsübliche MicroSD Karte zum Einsatz, welche aktuell 8 GB fasst. Das System wird mit 5 V versorgt und hat eine Leistungsaufnahme von etwa 2 W bis 3 W.

Als Betriebssystem kommt ein Ångström [1] in einer für das BeagleBone angepassten Version zum Einsatz. Als wichtigste Besonderheit gegenüber anderen, bekannten Linux Distributionen für den Anwender ist zu beachten, dass hier systemd [9] zum Einsatz kommt. Für den Einsatz auf dem MOPS sind zudem alle nicht benötigten Dienste deaktiviert worden.

4.2.2. RTC

Die Systemzeit kann auf dem BeagleBone nicht ohne Spannungsversorgung gehalten werden. Hierfür wird eine RTC (real time clock) von Dallas, DS1337, eingesetzt. Eine Lithium-Knopfzelle (CR2032) versorgt die RTC auch dann weiter, wenn der MOPS selbst über keine Versorgung verfügt. Bedingt durch die begrenzten Möglichkeiten des Aufbaus besitzt die RTC eine Drift von mehreren Sekunden pro Tag. Vor wichtigen Manövern sollte die Uhrzeit per NTP abgeglichen und in die RTC gespeichert werden. Hierzu sind im Abschnitt 5.3 benötigte Befehle dokumentiert.

4.2.3. Sensorik

In der derzeitigen Variante ist Sensorik zur Navigation verbaut. Hierbei handelt es sich um ein GPS-Modul sowie einen Kombinations-Inertialsensor mit neun Freiheitsgraden, welcher einen Beschleunigungssensor, ein Gyroskop sowie einen Magnetfeldsensor mit jeweils drei Achsen kombiniert.

4.2.3.1. GPS

Das GPS-Modul NL-652ETTL der Firma Navilock baut auf einem u-blox-6-Chipsatz auf. Die Kommunikation mit dem Modul erfolgt über eine 3,3 V UART. Es sendet im Sekundentakt NMEA Nachrichten. Zur Konfiguration kann das u-blox u-center [28] eingesetzt werden.

4.2.3.2. Inertialsensorik

Der Inertialsensor *9 DoF Razor IMU* [30] verfügt über einen AVR-Mikrocontroller, welcher die Auswertung der Einzelsensoren und Umrechnung in eine auf die Erde bezogene Ausrichtung vornimmt. Hierzu kommt die AHRS-Firmware [17] zum Einsatz. Das Modul ist über eine 3,3 V UART an das BeagleBone angeschlossen.

4.2.4. Aktorik

Bei den in Kapitel 4.1.2 beschriebenen Motoren handelt es sich um Gleichstrommotoren. Die Motoren sind für den Betrieb an einem 12 V Bleiakku ausgelegt. Nach Herstellerangaben beträgt der Strom bis zu 40 A. Da zwei solcher Motoren zum Einsatz kommen, ist für das Design des Versorgungs- und Reglerkonzepts ein sehr hoher Strom von bis zu 80 A zu berücksichtigen.

Um die Motoren mit einem Regler steuern zu können, wird der Geschwindigkeitsdrehschalter auf die maximale Einstellung (5) gestellt. Da dieser lediglich Wicklungen am Motor umschaltet, gibt es keine Beeinflussung durch einen externen Regler. Auch die eingebaute Batterieanzeige funktioniert weiterhin: Das Aktivierungssignal wird mittels Durchtrennen des Kabels dauerhaft gesetzt. Bereits bei geringen Geschwindigkeiten stellt die Anzeige einen Indikator für die Spannung der Batterie dar (0: leer, ab 6: voll).

4.2.5. Motorregler

Für die präzise Steuerung des MOPS ist eine elektronische Steuerung der Motorgeschwindigkeiten vorwärts wie rückwärts erforderlich. Zum Einsatz kommt der Regler Sabertooth 2x60 [6] des Herstellers DimensionEngineering. Dieser Regler zeichnet sich durch eine hohe Flexibilität in der Ansteuerung (Wahlweise kann die Geschwindigkeit über eine analoge Spannung, ein RC[1]-typisches PWM-Signal oder eine serielle Verbindung gestellt werden) aus. Die Betriebsspannung darf im Bereich von 6 V bis 30 V liegen und der maximale Laststrom 60 A, kurzzeitig bis zu 120 A, betragen.

Aufgrund der im Design verfügbaren Peripherie wird das RC-PWM-Signal zur Ansteuerung genutzt. Im Modellbau weit verbreitet ist ein PWM-Signal mit einer Periode von

[1] Remote-Controlled, Verweis auf den Modellbau

etwa 50 Hz, bei dem das Nutzsignal durch die Pulsbreite übertragen wird. Üblicherweise wird das Signal wie in Tabelle 4.2 dargestellt interpretiert.

Pulsbreite	**Prozentual**	**Bedeutung**
1000 µs	−100 %	linker Servoanschlag, Motorgeschwindigkeit maximal negativ
1500 µs	0 %	Mittelstellung des Servos, Motorgeschwindigkeit null
2000 µs	100 %	linker Servoanschlag, Motorgeschwindigkeit maximal positiv

Tabelle 4.2.: Bedeutung der Pulsbreite eines RC-PWM-Signals

Ist eine serielle Verbindung zu einem Windows-PC hergestellt, so können mittels des Tools DEScribe [5] verschiedene Parameter eingestellt werden. Der Regler besitzt zwei Modi, welche zur Interpretation des RC-PWM-Signals geeignet sind. Per Tool wird der RC-Modus so eingestellt, dass fest eingestellte Pulsbreiten (siehe Tabelle 4.2) verwendet werden. Zudem wird das Timeout des PWM-Signals auf etwa 100 ms eingestellt, um die Motoren bei verschiedenen Störungen oder Defekten abzuschalten. Als weiterer Schritt wird eine lineare Rampe mit einer Breite von etwa 1 s eingestellt, mit der Sprünge des Eingangssignals auf den Motor weitergegeben werden. So können Spitzenströme begrenzt und die Motoren geschont werden.

4.2.6. Kommunikation

Zur Kommunikation stehen zwei Schnittstellen bereit: Zum einen kann das Board einem 802.11b/g Wlan beitreten und per SSH bedient werden. Hierzu sind folgende Einstellungen notwendig:

- SSID: MOPS
- Verschlüsselung: WPA/WPA2 Personal (TKIP oder CCMP)
- Passphrase: uni-oldenburg-studenten
- IP-Adresse: (DHCP; bisher 192.168.7.116)

Zum anderen ist ein XBee Pro 868 Funkmodul verbaut. Dieses sendet im 868 MHz Band mit einer maximalen RF-Datenrate von 24 kbps. Aufgrund gesetzlicher Bestimmungen sind jedoch im Stundenmittel nur etwa 2 kbps nutzbar. Das Frequenzband erlaubt bei Sichtverbindung eine Reichweite von bis zu 80 km, mit einfachen Antennen sollten bis zu 20 km erreicht werden können.

Das Xbee-Modul ist über eine weitere 3,3 V UART an das BeagleBone angebunden. Zur Stromversorgung werden hier 3,3 V mit bis zu 0,8 A benötigt.

4.2.7. Energieversorgung

4.2.7.1. Akkumulator

Um den MOPS während der Einsatzzeit mit Energie zu versorgen, wird ein entsprechend großer Akku benötigt. Ideal aus Sichtpunkten der Energiedichte, sowohl auf Volumen als auch auf Masse bezogen, ist ein Lithium-Polymer Akkumulator. Aus Sicherheits- und Kostengründen wird jedoch auf einen vorhandenen Bleiakku aus dem Marinebereich zurückgegriffen. Ein Bleiakku zeichnet sich durch folgende technische Daten aus:

- Leerlaufspannung (6 Zellen): $\approx$ 11 V bis 12,7 V
- maximaler Entladestrom: abhängig vom Zellenaufbau etwa 1 C bis 10 C[2]
- Ladeverfahren: IU (CCCV, constant current constant voltage)

4.2.7.2. Ladevorgang

Um eine lange Lebensdauer zu erzielen, sollten Bleiakkus grundsätzlich vollständig geladen werden. Steht kein passendes Ladegerät bereit, so kann mit einem beliebigen Netzteil mit einstellbarer Spannung sowie Strombegrenzung geladen werden. Der Ladestrom sollte 2 C nicht überschreiten, die Ladeschlussspannung ist temperaturabhängig und beträgt bei Raumtemperatur 2,3 V pro Zelle (entspricht 13,8 V). Soll der Akku in kurzer Zeit aufgeladen werden, so darf die Ladeschlussspannung für einige Zeit überschritten werden. Ab etwa 2,5 V pro Zelle beginnt ein starker Gasungsvorgang in der Batterie, welcher nicht eintreten sollte. Eine Spannung von 2,4 V pro Zelle für einige Stunden ist jedoch unkritisch und führt zu einem vollständig geladenen Akku.

4.2.7.3. Spannungsregler

Die Batteriespannung wird nach dem größten Verbraucher, also dem Antrieb ausgelegt, um an dieser Stelle Wandlungsverluste einzusparen. Ein früherer Prototyp des MOPS, für den der Großteil der Elektronik ausgelegt wurde, wurde mit 24 V versorgt. Aus den vorherigen Kapiteln ergeben sich die in Tabelle 4.3 aufgeführten Anforderungen an zusätzliche Versorgungsspannungen.

Spannung	**Verwendung**	**Dauerstrom**	**Spitzenstrom**
5 V	BeagleBone, GPS	0,6 A	1,5 A
3,3 V	XBee, Inertialsensor	0,1 A	0,8 A
1,8 V bis 5,5 V	RTC	1,5 µA (standby)	150 µA (aktiv)

Tabelle 4.3.: Versorgungsspannungen im Steuersegment

Um diese Versorgungen möglichst effizient bereitzustellen, werden Schaltregler des Typs

[2]Diese Angabe beschreibt den Entladestrom als Vielfaches der Kapazität durch Zeit

LM2596 eingesetzt. Die Auslegung erfolgt nach Datenblatt mit bei Reichelt erhältlichen Komponenten. Der erreichte Wirkungsgrad liegt bei etwa 80 %.

4.2.8. Segmentaufbau

Der MOPS in der aktuellen Konfiguration besteht aus zwei bestückten Segmenten:

- Batteriesegment
- Steuersegment

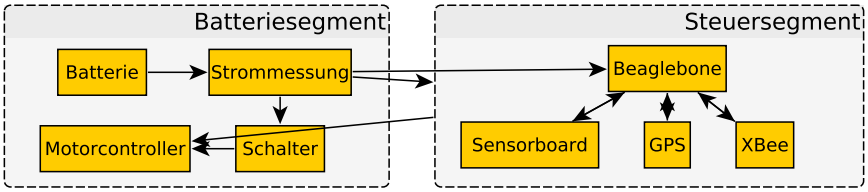


Abbildung 4.5.: Verbindungsstruktur der elektronischen Komponenten im MOPS

4.2.8.1. Steuersegment

Im Steuersegment sind alle o. g. Komponenten untergebracht und gemäß dem im Anhang A.4 befindlichen Schaltplan aufgebaut. Das BeagleBone ist auf eine Lochrasterplatine aufgesteckt, welche zusätzlich diverse Buchsenleistungen zur Verbindung des GPS- und Sensormoduls enthält. In Abbildung 4.6 ist eine Aufnahme der Platine dargestellt. Die Stecker des GPS- und Sensor-Moduls sind jeweils farblich markiert: Der Stecker zum GPS-Modul ist an einer Seite, dem Pin 1, rot markiert, während beim Sensorboard der Pin mit der braunen Ader auf der Platine markiert ist.

Die Spannungsregler und die RTC sind direkt aufgebaut. Das Xbee-Modul wird direkt aufgesteckt.

Die Verbindung mit dem Segmentsteckverbinder erfolgt zum Einen über den 2-poligen Stromversorgungsstecker sowie über die 6-adrige Leitung für Steuersignale, welche direkt aufgelötet ist.

4.2.8.2. Batteriesegment

Das Batteriesegment besteht hauptsächlich aus dem beschriebenen Marine-Akku. Zudem ist der Motorcontroller hier untergebracht, da für den prototypischen Aufbau der Aufwand eines weiteren Segments vermieden werden sollte. Zusätzlich kann die Versorgung des Motorcontrollers über ein Leistungsrelais geschaltet werden. Der durch die Batterie fließende Strom wird über einen Shuntmonitor verstärkt und kann über einen Analogeingang eingelesen werden. Der Schaltplan befindet sich ebenfalls in Anhang A.3.

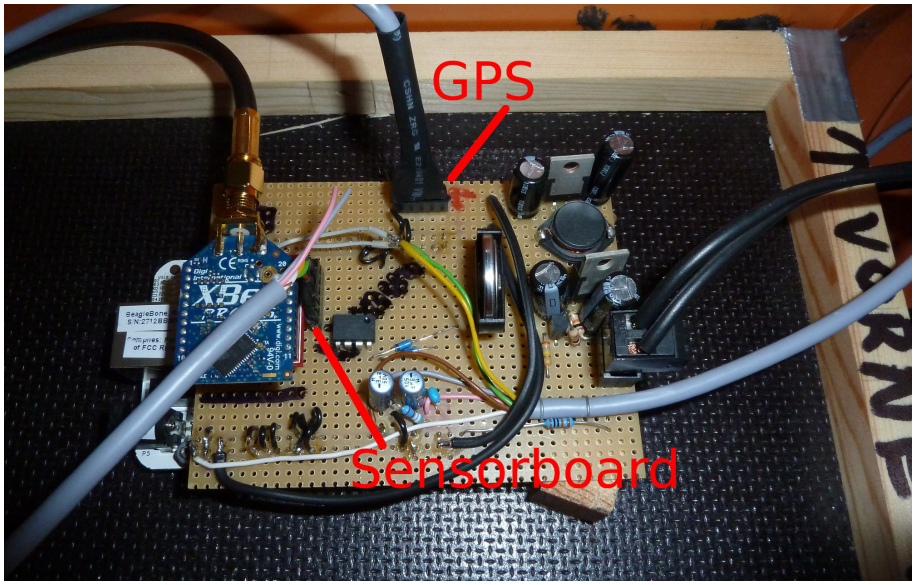


Abbildung 4.6.: Beaglebone-Aufsteckplatine mit Markierungen für Sensoren

4.3. Sensorkonzept

Ein zentraler Gedanke bei der Entwicklung des MOPS ist die Modularität. Es soll möglich sein, beliebige Segmente auszutauschen, um den MOPS für den jeweiligen Anwendungsfall vorzubereiten. Dies umfasst in erster Linie verschiedene Sensorsegmente, kann aber auch auf Antriebe, Stromversorgung oder das Steuersegment ausgeweitet werden.

Um eine solche Modularität zu erreichen, wird eine Verbindung zwischen den Segmenten benötigt, welche beliebig erweiterbar ist. Hierzu kann ein Bus zum Einsatz kommen, welcher durch alle Segmente durchgereicht wird. Da nur das Antriebssegment einen hohen Stromverbrauch hat, könnte hier eine weitere Stromversorgungsleitung eingesetzt werden, um Kosten und Platz an allen anderen Segmenten zu sparen.

4.3.1. Busnode

Aus obiger Überlegung kann ein Bussystem entwickelt werden. Es werden Busnodes benötigt, welche zum Einen die Schnittstelle des Busses implementieren und weiterhin benötigte Schnittstellen, z.B. für Sensorik (RS232, Analogeingänge, SPI, I2C), Motortreiber (RS232, PWM, I2C) oder diverse Hardware (GPIOs) zur Verfügung stellen. Zudem kann ein Spannungsregler vorgesehen werden, welcher die Versorgung der Busnode (3,3 V oder 5 V) und optional weitere Versorgungsspannungen für weitere Komponenten des Segments erzeugt.

Das Konzept dieser Busnodes ist beispielhaft am NXP LPC11C24 [16] geplant, jedoch nicht umgesetzt. Dieser Cortex M0 Mikrocontroller enthält einen CAN Transceiver. Dadurch wird die Entwicklung kostengünstiger, platzsparender Busnodes ermöglicht. CAN, bekannt aus der Automobilindustrie, ist eine robuste Schnittstelle mit geringen Anforderungen an

Kabel oder Energieverbrauch. Es eignet sich in der Standard Variante für mehr als 1024 Busteilnehmer, welche mit bis zu 1 MBps Nachrichten per Broadcast auf den Bus schicken können.

Auf dem BeagleBone wird lediglich ein CAN Transceiver benötigt, zudem muss das CAN Interface im Betriebssystem konfiguriert werden.

4.4. Fazit und Ausblick

Der MOPS in seiner aktuellen Hardwarekonfiguration ist ein stabiler und leistungsfähiger Trimaran. Dabei wird noch genügend Auftrieb geboten, sodass Modifikationen und Erweiterungen vorgenommen oder noch zusätzliche Lasten angebracht werden können. Der Aluminiumrahmen bietet eine ausreichende Stabilität, sodass sogar ein Einsatz auf mäßiger Hochsee möglich sein sollte und ermöglicht durch die Zerlegbarkeit und die einfache Montage einen schnellen Transport. Die Schwimmkörper sowie die einzelnen Segmente einschließlich der Steckverbindungen sind wasserdicht verschlossen, sodass das Risiko eines Wassereinbruchs minimal ist. Die beiden Antriebe bieten genügend Leistung, um gegen leichte Strömung und Wind anzukommen. Durch die verwendete Elektronik, insbesondere die Leistungsfähige Recheneinheit, ist eine Erweiterbarkeit und die notwendige Modularität gegeben.

Eine Gegenüberstellung der Anforderungen und der Umsetzung durch die Projektgruppe zeigt einige Diskrepanzen auf. Die Akkulaufzeit beträgt je nach Geschwindigkeit etwa zwei bis fünf Stunden, wodurch große Tagesfahrten nicht durchführbar sind. Zudem sind Rahmen und Schwimmer des MOPS zu groß, um einen leichten Transport, z. B. in einem PKW-Combi durchführen zu können. Des Weiteren ist ein entsprechendes Sensorkonzept bisher nicht umgesetzt. Zudem ist weitere Sensorik zur Navigation und Kollisionsvermeidung erforderlich. Dadurch ergeben sich im Ausblick noch einige offene Aufgaben auf der Hardwareseite, welche in Zukunft bearbeitet werden könnten.

5. Software

In diesem Kapitel wird die im Rahmen des MOPS Projektes entwickelte Software beschrieben und ihre Funktionalität erläutert. An einigen Stellen wird zusätzlich auf Möglichkeiten zur Weiterentwicklung eingegangen. Zunächst erfolgt eine Erklärung des für den MOPS entwickelten Kommunikationsmoduls. Anschließend wird die Missionsplanungs- und -überwachungssoftware erklärt. Danach wird das Linux System, welches auf dem MOPS läuft, kurz vorgestellt um daraufhin auf die Onboard-Software des MOPS einzugehen. Abschließend wird der Simulator, welcher im Rahmen des Tests der Onboard-Software verwendet wurde, erläutert.

5.1. Kommunikation

Unsere Entscheidung eine Software auf dem MOPS, so wie eine Missionsplanungs- und -überwachungssoftware zu entwickeln, hat uns vor das Problem der Kommunikation zwischen diesen beiden gestellt.

Hierzu haben wir für den direkten Austausch von Nachrichten ein zu verwendendes Protokoll festgelegt und ein Softwaremodul entwickelt, welches von On- und Offboard[1]-Seite verwendet werden soll. Diese Kommunikationssoftware soll dabei eine einfache Möglichkeit bieten Nachrichten zu verschicken, zu empfangen und zugleich eine sichere Kommunikation bieten, d.h. Nachrichten sollten nicht verloren gehen und Verbindungsabbrüche sollen mitgeteilt und automatisch behoben werden.

Als Kommunikationskanäle haben wir uns für WLAN (Sockets) und XBee entschieden.

5.1.1. Allgemeine Implementation

Der *CommunicationManager* stellt die Hauptklasse der Kommunikation dar und bietet die Methoden zur Erstellung einer Verbindung (z.B. *setSocketClient()*), zum Versenden (*send()*), zum Empfangen (*setMessageReceivedListener()*) und zur Statusüberwachung (*setConnectionListener()*). Die Funktionalitäten des CommunicationManagers sind zudem in dem Interface *ICommunication* gekapselt.

Die Verbindungen werden in Form des Interfaces *CommunicationHardware* verwaltet. Hierbei können jeweils maximal eine eingehende Socket- und eine eingehende XBee-Verbindung existieren, von denen wiederum nur eine beim Versenden verwendet wird (WLAN

[1] Missionsplanungs- und -überwachungssoftware

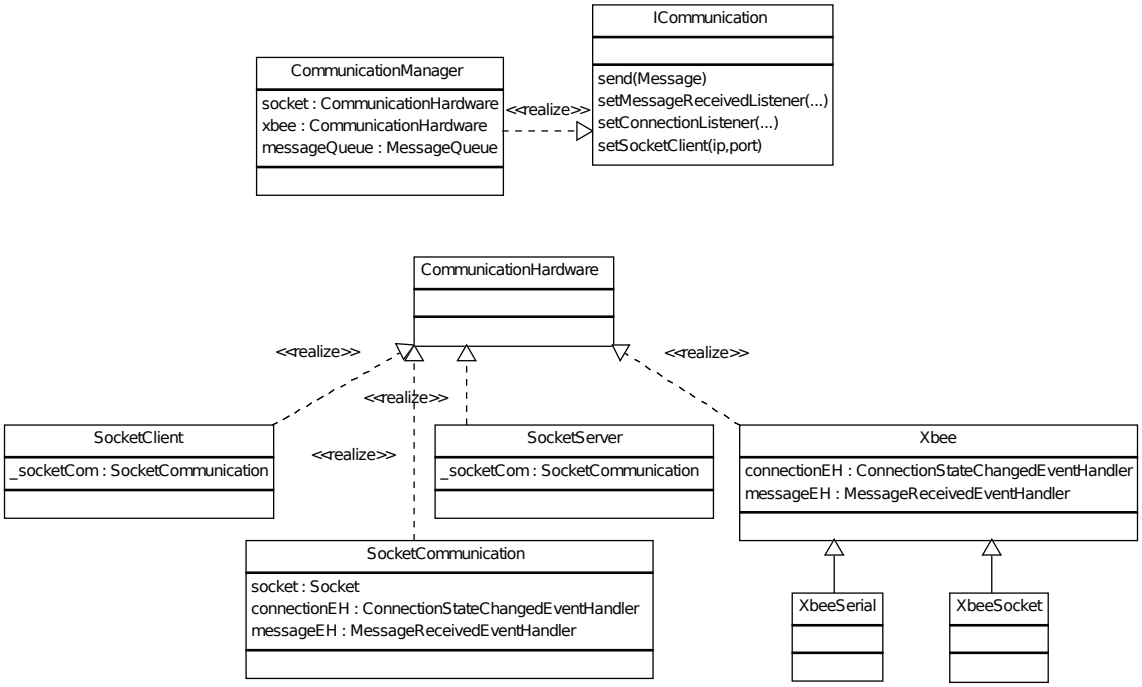


Abbildung 5.1.: Ein grobes Klassendiagramm der Kommunikation mit einigen wichtigen Membern und Methoden.

wird gegenüber XBee bevorzugt). Für die Limitierung auf eine Verbindung wurde sich aufgrund der steigenden Komplexität entschieden, da nur ein Kommunikationskanal benötigt wurde. Zudem hätte das Verwalten von Nutzernamen zu Nachrichten nicht nur für die Kommunikationssoftware, sondern auch für die On- und Offboardsoftware erhöhte Komplexität bedeutet.

Bevor eine andere Verbindung über den gleichen Kanal eingegangen werden kann, muss die alte Verbindung zunächst mit der entsprechenden *remove* Methode entfernt werden.

Hinweis: Bevor eine Verbindung genutzt werden kann, muss nach dem Aufruf der jeweiligen set Methode noch die entsprechende connect Methode aufgerufen werden.

Senden Zum Versenden einer Nachricht muss ein *Message* Objekt erstellt werden, welches ein Bytearray oder ein serialisierbares Objekt erwartet. Dieses Messageobjekt kann dann über *send* an den *CommunicationManager* versendet werden.

Das Versenden dieser Messageobjekte wird über eine Queue geleitet, die es ermöglicht auch bei einem Verbindungsabbruch dafür zu sorgen, dass die Nachrichten später in richtiger Reihenfolge versandt werden.

Zusätzlich zu dem normalen Versand wird die Möglichkeit geboten einer Nachricht einen Typen zu geben. Wenn eine Nachricht mit einem Typen der Queue hinzugefügt wird, wird eine bereits in der Queue vorhandene Nachricht mit diesem Typen durch die neue Nachricht ersetzt. Dies ist z.B. bei dem Versenden von manuellen Steuerungssignalen not-

wendig.

Empfangen Das Empfangen von Nachrichten wird über Events nach außen gegeben. Jede versandte Nachricht erzeugt bei dem Empfänger ein Event, welches die versandte Nachricht enthält. Zum Erhalten dieser Events muss ein Object, welches das Interface des Listeners implementiert, registriert sein.

Events Events werden in der Kommunikationssoftware genutzt um empfangene Nachrichten (*MessageReceived*) nach außen zu geben und um Verbindungsänderungen (*ConnectionStateChanged*) mitzuteilen.

Wenn auf der untersten Softwareebene der Kommunikation eine Nachricht eingeht oder eine Verbindungsänderung festgestellt wird, wird dies dem entsprechenden *EventHandler* mitgeteilt, welcher dann dieses Event über die registrierten *Listener* nach außen trägt.

5.1.2. WLAN

Die WLAN-Implementierung nutzt die von Java gestellten Klassen *Socket* und *ServerSocket*. Hierzu gibt es für den Client- und Serverteil jeweils eine Klasse, die das Erstellen des *Sockets* und höhere Aufgaben, wie das automatische Neuverbinden, übernimmt.

Nachdem ein *Socket* erstellt wurde, wird dieser an die Klasse *SocketCommunication* übergeben, welche eine allgemeine Verbindung über einen *Socket* verwaltet.

5.1.3. XBee

Die Implementierung der XBee-Kommunikation basiert auf der Rapplogic XBee-API Bibliothek. Diese Bibliothek ermöglicht die Verwendung von verschiedenen XBee-Modulen im API Modus. Im API Modus sind die XBee-Module in der Lage, ein Netzwerk bestehend aus mehreren Knoten aufzubauen. In der Anwendung im MOPS wird lediglich die Kommunikation zwischen zwei Knoten benötigt. Jegliche Kommunikation mit den XBee-Modulen erfolgt über API-Frames, in welchen verschiedene Nachrichten definiert sind. Eine Auswahl ist in Tabelle 5.1 aufgeführt.

Als Alternative zum API Modus kann auch ein transparenter Modus benutzt werden. Aufgrund der verfügbaren Statusinformationen im API Modus, die zur Diagnose des Verbindungsstatus genutzt werden können, wird dieser verwendet.

Die XBee-API Bibliothek ermöglicht erstmal nur eine Anbindung des XBee-Moduls über eine serielle Verbindung. Hierzu wird die RXTX [21] Bibliothek verwendet, mit welcher es häufig zu Problemen kommt. Aus diesem Grund wird die XBee-API Bibliothek um die Möglichkeit erweitert, über einen Socket mit dem XBee-Modul zu kommunizieren.

Die Länge eines zu übertragenden Pakets ist durch das XBee-Modul auf maximal 100 B begrenzt. Werden andere XBee-Module verwendet, so kann diese Beschränkung abweichen.

Frame Name	Beschreibung
AT Command (0x08)	Absetzen von AT Befehlen zur Konfiguration des XBees
Transmit Request (0x10)	Senden eines Datenpakets mit Angabe von Metadaten
AT Command Response (0x88)	Antwort auf einen AT Befehl zur Rückgabe von Parametern
Transmit Status (0x8B)	Antwort auf einen Transmit Request mit Statusinformationen über das gesendete Paket
Receive Packet (0x90)	Datenpaket empfangen

Tabelle 5.1.: Auswahl einiger XBee API Nachrichtentypen

Es wird also ein Protokoll benötigt, um auch längere Datenpakete übertragen zu können. Hierzu wird ein simples Protokoll eingeführt. Jedes Datenpaket sieht wie folgt aus:

1. 1 Byte: Flag
2. n Bytes: Nutzdaten

Da das XBee-Modul selbst sicherstellt, dass nur ein Paket als Ganzes verloren gehen kann, reicht diese Struktur aus. Die Flags beschreiben folgende Informationen über das Teilpaket:

- New Packet: Es handelt sich um den ersten Teil eines Pakets.
- Continuation: Dies ist (mindestens) der zweite Teil eines Pakets.
- Last Packet Part: Dieser Teil ist der letzte eines Pakets.
- Ping Request: Das Paket enthält keine Nutzdaten, es wird eine Pong Response zur Messung der Laufzeit erwartet.
- Pong Response: Dieses Paket enthält keine Nutzdaten und folgt als Antwort auf einen Ping Request.

Durch Auswertung des Zustands des Empfängers in Verbindung mit den Flags eines empfangenen Teilpakets kann zusammen mit der Sicherheit, dass bei Übertragungsfehlern keine weiteren Teilpakete gesendet werden garantiert werden, dass niemals korrupte oder nicht zusammengehörige Paketteile verarbeitet werden.

5.1.4. High-Level-Protokoll

Zusätzlich zum Kommunikationsprotokoll wurde ein High-Level-Protokoll [15] eingeführt. Dieses Protokoll dient der fachlichen Sicht auf die Kommunikation und muss sowohl von der Onboard-Software, als auch von der Missionsplanungs- und -überwachungssoftware implementiert werden.

Es wurden zum einen serialisierbare Transfer-Objekte, zum Austausch größerer zusammenhängender Daten und zum anderen ByteArray-Nachrichten definiert.

5.1.4.1. ByteArray-Nachrichten

Die ByteArray-Nachrichten bestehen, wie der Name schon sagt, aus einem ByteArray. Dabei enthält jede Nachricht mindestens ein Byte, dass den Typen der Nachricht angibt (siehe Tabelle 5.2).

Name	**Aufbau**
Heartbeat	byte (0), int (Latitude$*10^7$), int (Longitude$*10^7$), int (Ausrichtung in °), long (time in millis), byte, short (kn/h$*$100)
Mission starten	byte (1), String (UID)
Mission stoppen	byte (2)
Mission fortsetzen	byte (3)
Mission fortsetzen	byte (3), String (UID)
Fahre zu Koordinaten	byte (4), double (Latitude), double (Longitude), int (Radius in m)
Manuelle Steuerung	byte (5), byte (Geschwindigkeit), byte (Ruderstellung)
Umschalten in DriftMode	byte (6)
Position halten	byte (7)
Motor aus	byte (8)
Missionsstatus anfordern	byte (9)
Missionsstatus anfordern	byte (9), String (Missions UID)
Komplette Mission anfordern	byte (10)
Komplette Mission anfordern	byte (10), String (Missions UID)
Daten abrufen	byte (11), String (Missions UID)
Log anfordern (fortlaufend)	byte (12)
Missionsstatusantwort	byte (18), String (Missions UID), int (Versions ID), String (Wegpunkt UID), int (verbleibende Zeit am aktuellen Punkt in Sekunden)
Manuelle Steuerung starten	byte (19)
Manuelle Steuerung stoppen	byte (20)
Statusnachricht	byte (21), String (Message)
Log-Nachricht	byte (22), long (timeStamp), String (Name der Klasse), int (Log-Level), String (Thread-Name), String (Message)
Stoppen des Sendens von Log-Nachrichten	byte (23)

Tabelle 5.2.: ByteArray-Nachrichten

Positive Acknowledges Es wurden High-Level-Acknowledges eingeführt, damit nicht bloß der Empfang einer Nachricht festgestellt werden kann, sondern der Zeitpunkt wann die Nachricht auf Empfängerseite bearbeitet wurde.

Ein positives Acknowledge (siehe Tabelle 5.3) bedeutet, dass die Bearbeitung erfolgreich abgeschlossen wurde.

Name	Aufbau
Allgemeines Acknowledge	byte (100)
Missionsplanung/-änderungs Acknowledge	byte (101), String (Missions UID)
Mission started Acknowledge	byte (102), String (Missions UID)
Mission continued Acknowledge	byte (103), String (Missions UID)
Mission stopped Acknowledge	byte (104), String (Missions UID)
Fahre zu Koordinaten Acknowledge	byte (105), double (Latitude), double (Longitude)
Manuelle Steuerung start Acknowledge	byte (106)
Manuelle Steuerung stop Acknowledge	byte (107)
Umschalten in Drift Acknowledge	byte (108)
Position halten Acknowledge	byte (109)
Motor aus Acknowledge	byte (110)
Stop sending log entries Acknowledge	byte (111)

Tabelle 5.3.: Positive Acknowledges

Negative Acknowledges Negative Acknowledges bedeuten, dass es bei der Bearbeitung einer Aufgabe zu einem Fehler gekommen ist. Die negativen Acklowedges sind identisch mit den positiven Acknowledges, nur dass das erste Byte negiert ist. Bei einem normalen Acknowledge wäre der Wert des ersten Bytes also nicht 100 sondern -100.

5.1.4.2. Transfer-Objekte

Transfer-Objekte sind serialisierbare Objekte, die von der Klasse *TransferObject* erben. Sie beinhalten mindestens einen *TransferObjectType*, der beschreibt, um was für ein Transfer-Objekt es sich handelt. Mögliche Werte hierfür sind:

- LOG_ENTRIES
- SENSOR_DATA
- COMPLETE_MISSION
- MISSION_PLANNING

Von den geplanten Typen werden zum aktuellen Zeitpunkt nur *COMPLETE_MISSION* und *MISSION_PLANNING* verwendet. Zur Übertragung einer Mission wird die Klasse *TransferObject* von der Klasse *TOMission* erweitert (siehe Abbildung 5.2). In dieser sind alle eine Mission betreffenden Daten enthalten. Da an jedem Missionspunkt/Wegpunkt Aktionen durchgeführt werden können, reicht eine Übergabe von Wegpunkt-IDs alleine nicht aus. Deshalb wurde die Klasse *TOMissionPoint* erstellt. Sie beinhaltet alle Daten, die für die Erstellung bzw. Änderung eines Wegpunktes notwendig sind.

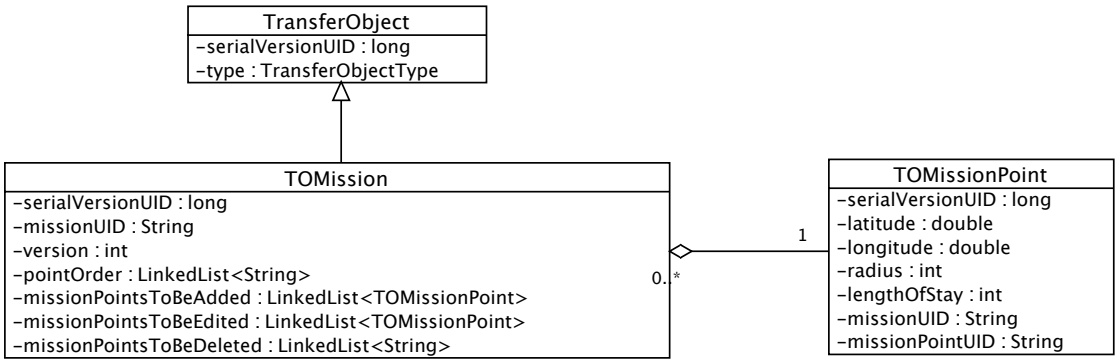


Abbildung 5.2.: Erweiterung des *TransferObject* zur Übertragung von Missionen

5.2. Missionsplanungs- und -überwachungssoftware

In diesem Abschnitt soll die Missionsplanungs- und -überwachungssoftware vorgestellt werden. Bei der Software handelt es sich um ein Java-Programm, dass prinzipiell auf jedem Computer mit jedem Betriebssystem, auf dem die JRE installiert ist, ausgeführt werden kann. Die Software ist objektorientiert programmiert und wurde prinzipiell nach dem von Trygve Reenskaug 1979 vorgestelltem Model-View-Controller(MVC)-Konzept aufgebaut. Es ist damit möglich, einzelne Komponenten der Software auszutauschen, ohne dabei große Änderungen in anderen Bereichen durchführen zu müssen. Ein schematischer Aufbau der entwickelten Software kann der Abbildung 5.3 entnommen werden. Die Software untergliedert sich demnach in vier Hauptkomponenten, die jeweils eigene Unterkomponenten besitzen. Die vier Hauptkomponenten sind: der Controller (dunkelblau), die Benutzeroberfläche (grün), das Modell (rot) und die Kommunikation (hellblau). Neben diesen Hauptkomponenten gibt es das Package *utils* in dem diverse Hilfsklassen untergebracht sind. Auf jede dieser Hauptkomponenten und ihr Zusammenspiel untereinander, soll in den folgenden Abschnitten jeweils genauer eingegangen werden.

5.2.1. Controller

Die Controller der Software stellen die Verbindung zwischen den Komponenten der Software her. Der Hauptcontroller ist in der Klasse *MainController* implementiert. Er erstellt und verwaltet alle Subcontroller. Jeder Subcontroller ist dabei modular gehalten, so dass einzelne Controller ausgetauscht werden können, ohne größere Änderungen in den anderen Teilen vornehmen zu müssen. Der MainController verwaltet vier Subcontroller; den *TilesUpdateController*, den *SensorManagementController*, den *GamepadController* und den *GamepadCalibrationController*. Der *MainController* erstellt die benötigten Modelle, auf die im Abschnitt 5.2.4 noch genauer eingegangen wird und implementiert die Funktionen auf der graphischen Oberfläche. Er stellt die internen Kommunikationsklassen, auf die wir in Abschnitt 5.2.3 genauer eingehen. Der Controller gibt damit die Benutzereingaben an die zuständigen Klassen weiter, indem die entsprechenden Funktionen aufgerufen werden.

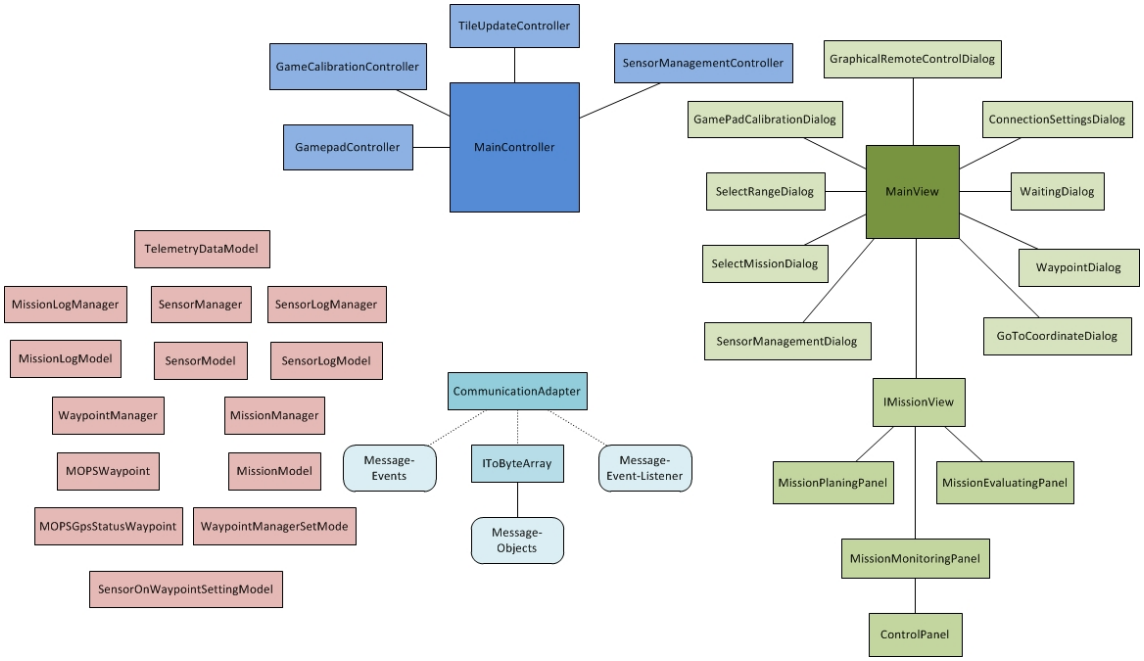


Abbildung 5.3.: Schematische Darstellung der Software

Dadurch ist es möglich, die Benutzeroberfläche auszutauschen und z.B. durch eine Weboberfläche zu ersetzen. Darüber hinaus werden auf dieser Ebene alle Exceptions, die in den anderen Klassen geworfen werden aufgefangen und verarbeitet. Mithilfe der Klasse *ApplicationLog* werden alle Fehler, die innerhalb des Programms entstehen, in einer Log Datei gespeichert. Darüber hinaus verwaltet der *MainController* wie bereits beschrieben, die vier Subcontroller. Der *TilesUpdateController* stellt alle Funktionalitäten zur Verfügung, die benötigt werden, um Kartenausschnitte, die auf der lokalen Festplatte gespeichert sind, auf den neusten Stand zu bringen, also Updates von den Kartenservern herunterzuladen. Der *GamepadController* kapselt alle Funktionalitäten, die im Zusammenhang mit einem angeschlossenem Gamepad hängen und gibt diese weiter. Dabei ist die Kalibrierung, die durch unterschiedliche Gamepads und Betriebssysteme notwendig ist, wiederum in einen eigenen Controller ausgelagert, den *GamepadCalibrationController*. Hier werden die Tasten des Gamepads den jeweiligen Aufgaben (z.B. voraus, zurück, u.a.) zugeordnet. Der an dieser Stelle zuletzt genannte Controller ist der *SensorManagementController*, welcher für das Sensormanagement verantwortlich ist. Er erstellt einen Sensormanager und fügt der GUI Funktionen im Zusammenhang mit den Sensoren des MOPS hinzu.

5.2.2. Benutzeroberfläche

Die Benutzeroberfläche ist eine zusammengefügte Komponente, die sich aus verschiedenen kleinen Teilen zusammensetzt (siehe Abbildung 5.5). Auf der obersten Ebene der Benutzeroberfläche befindet sich die Klasse *MainView*. Sie verbindet die drei Hauptkomponenten zu einer zusammengesetzten Oberfläche. Die drei Hauptkomponenten der Software sind: das Menü, die Missionsübersicht sowie das Statuspanel. Die Menüklasse beinhaltet sämtliche

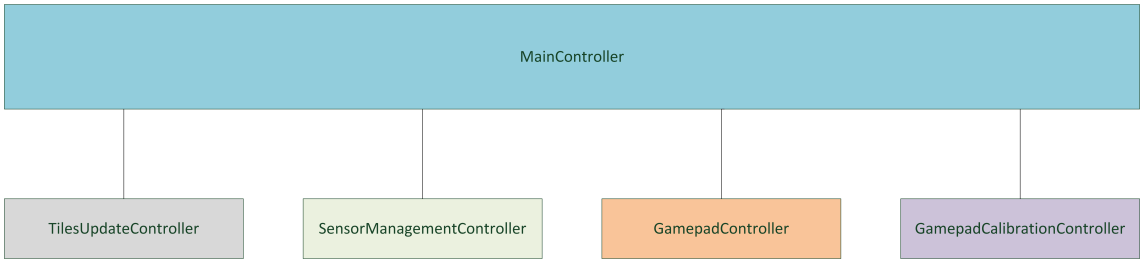


Abbildung 5.4.: Schematische Darstellung der Controller der Software

Menüeinträge, die in der Software verfügbar sind. In der Klasse *Statuspanel* befinden sich die Statusanzeigen für die Kommunikation sowie den aktuellen Onlinestatus der Software. Die Missionsoberfläche ist wiederum eine zusammengesetzte Komponente, welche die drei Ansichten für die Missionen verbindet. Diese sind das *MissionPlaningPanel*, das *MissionMonitoringPanel* und das *MissionEvaluatingPanel*. Jede dieser Komponenten setzt sich dabei aus anderen Komponenten zusammen. Das *MissionPlaningPanel* hat zwei Teile. Zum einen die Karte auf der die Wegpunkte einer Mission geplant werden können, zum anderen die Missionsplanungskomponente, in der Missionen und die enthaltenden Wegpunkte geplant, bearbeitet und an das Wasserfahrzeug übertragen werden können. Die Kartenkomponente findet sich dabei im *MissionMonitoringPanel* noch einmal wieder. Sie dient der Darstellung der aktuellen Mission, sowie der Darstellung der aktuellen Position des Fahrzeugs. Über die Telemetriekomponente können darüber hinaus aktuelle Daten ausgegeben werden. Neben diesen Überwachungskomponenten gibt es ein Steuerungselement, mit dessen Hilfe Missionen gestartet, gestoppt oder fortgesetzt werden können. Zusätzlich kann ein Notaus eingeleitet oder die manuelle Steuerung über ein Gamepad bzw. den *GraphicalRemoteControlDialog* aktiviert werden. Die dritte Seite ist die *Missionssevaluierung*, über die sowohl Missionslogdaten als auch Sensorlogdaten eingesehen werden können.

5.2.3. Kommunikation

Die Kommunikation stellt die Schnittstelle zwischen dem MOPS und der Missionsplanungs- und -überwachungssoftware dar. Innerhalb der Software wird die Bibliothek der Kommunikation durch die Klasse *CommunicationAdapter* gekapselt. Diese Kapselung trägt dazu bei eine Unabhängigkeit der (Kommunikations-)Schnittstelle von den übrigen Bereichen der Software zu erreichen. Die Architektur des Kommunikationsbereiches der Software stellt somit eine eigene Daten- und Ereignisstruktur zur Verfügung, die für die Interaktion über die eigentliche Kommunikationsschnittstelle genutzt wird. Die Basis der einzelnen Nachrichten stellen das Interface *IToByteArray* und die Klasse *Message*, die eine Klasse als generischen Parameter übergeben bekommt, die dieses Interface implementiert. Das Protokoll, das zur Kommunikation zwischen dem MOPS und der hier beschriebenen Software dient, und das durch die jeweiligen Parteien definiert wurde, beinhaltet eine Reihe unterschiedlicher Nachrichten bzw. Nachrichtentypen. Diese Nachrichtentypen wurden innerhalb der Software durch eigene *Message*-Klassen realisiert, die jeweils das *IToByte-*

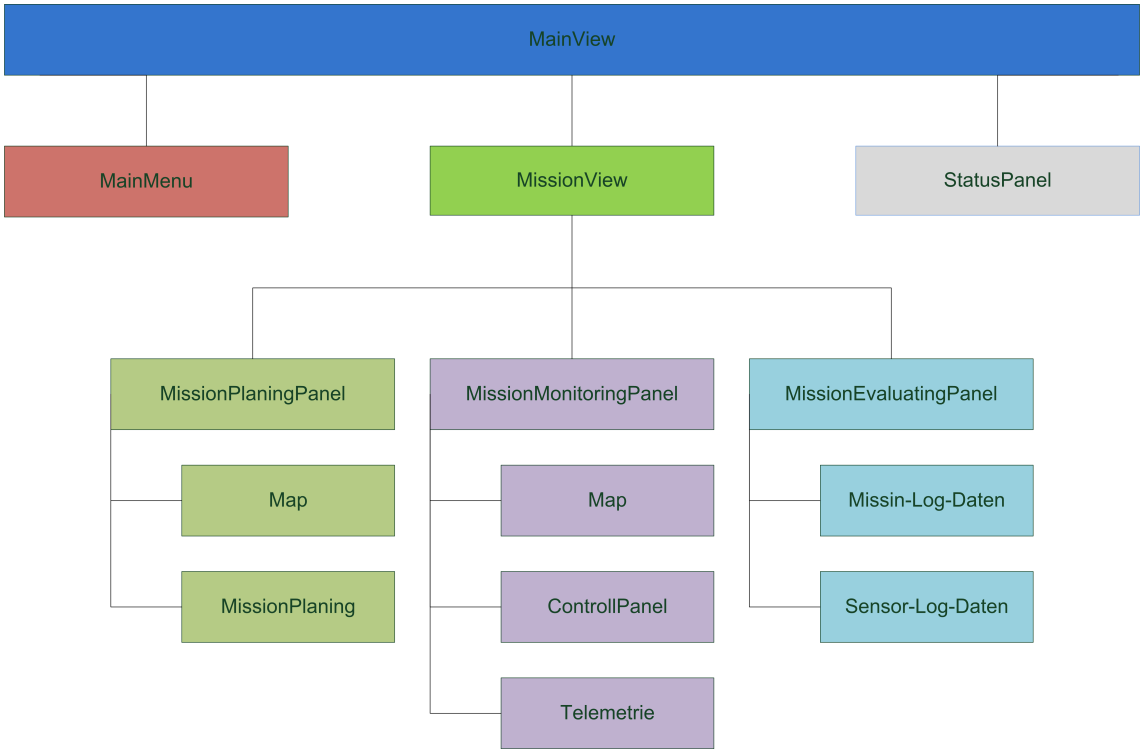


Abbildung 5.5.: Schematische Darstellung der Benutzeroberfläche

Array-Interface implementieren. Da diese Nachrichten binär übertragen werden, ist es somit nicht notwendig, dass die Klasse *CommunicationAdapter* Informationen über die Binärstruktur der einzelnen Nachrichtentypen bereithält, um diese an die Kommunikationsbibliothek weiterreichen zu können.
Neben Ereignissen die durch die Bilbiothek ausgelöst werden können, werden für sämtliche Nachrichtentypen eigene Events mit den entsprechenden Event-Listenern bereitgestellt. Erfolgt beispielsweise eine Zustandsänderung bei einer Verbindung oder wird eine Nachricht eines bestimmten Typs empfangen, so werden diese Infomationen über das „Event - Event-Listener“-Konstrukt an die registrierten Instanzen (hier an die Instanz der Klasse *MainController*) gemeldet. Der *MainController* übernimmt dann die Verarbeitung dieser Ereignisse und führt evtl. notwendige Maßnahmen durch, wie das Aktualisieren von internen Datenstrukturen oder das Benachrichtigen des Anwenders über die Benutzeroberfläche.
Neben der Übertragung von Nachrichten ist auch das Übertragen von serialisierbaren Objekten möglich. Zu diesem Zweck wurde eine Klassenstruktur definiert, auf die im Abschnitt 5.1.4 näher eingegangen wird.

5.2.4. Modell

Der strukturelle Bereich des Modells des MVC-Patterns gliedert sich innerhalb der Software in diverse Klassen, die die einzelnen Informationsbereiche voneinander abgrenzen. Eine zentrale Einheit der Softwarearchitektur sind die zu erstellenden bzw. zu verwal-

tenden Missionen. Eine Mission wird durch die Klasse *MissionModel* repräsentiert, und enthält neben der Version der jeweiligen Mission und dem Zeitstempel des Zeitpunktes, zu dem die Mission erstellt wurde, auch eine Reihe von Wegpunkten unterschiedlicher Typen. Auf der einen Seite haben wir Instanzen der Klasse *MOPSWaypoint*, die zur Planung der jeweiligen Mission dient und Daten zu einem bestimmten Wegpunkt enthalten, auf der anderen Seite haben wir Objekte der Klasse *MOPSGpsStatusWaypoint*, deren Basis die Klasse *MOPSWaypoint* darstellt und neben den üblichen Informationen eines Wegpunktes die Ausrichtung des Wasserfahrzeugs bereithält. Die Klasse *MOPSGpsStatusWaypoint* beinhaltet somit Teile der Informationen, die im Heartbeat (siehe Kapitel 5.4.8.3) des MOPS enthalten sind und wird beispielsweise für die Karte des Bereiches der *Überwachung* verwendet. Wie in allen Bereichen der Software, in denen eine Verwaltung von Objekten notwendig ist, stellt das Modell eigene Manager für die Verwaltung der Wegpunkte bereit (*WaypointManager*). Über das Setzen eines Parameters vom Typ *WaypointManagerSetMode* in der Klasse *WaypointManager* kann definiert werden, ob jeweils nur der letzte oder alle Wegpunkte innerhalb des internen Sets vorhanden sein sollen. Dies ist notwendig um die Sichtbarkeit der Wegpunkte in der jeweiligen Karte der Benutzeroberfläche festzulegen. Da die Notwendigkeit besteht mit vorhandenen Sensoren zu interagieren, wurde dem Modell die Klasse *SensorModel* hinzugefügt. Instanzen dieser Klasse kapseln Daten eines bestimmten Sensors wie dessen *ID*, den *Namen*, sowie den Zustand - also ob der jeweilige Sensor aktiv oder inanktiv ist. Auch Instanzen dieser Klasse können durch einen passenden Manager verwaltet werden (*SensorManager*). Innerhalb eines Wegpunktes werden Zustandsänderungen der Sensoren (also beim Erreichen bzw. Verlassen des Wegpunktes) in Instanzen der Klasse *SensorOnWaypointSettingModel* abgelegt. Instanzen dieser Klasse enthalten jeweils zwei Objektinstanzen der Klasse *SensorModel*, die diese Zustandsänderungen repräsentieren. Es liegt nahe, dass für die Verwaltung einzelner Missionen ebenfalls ein *MissionManager* zur Verfügung steht.
Ein weiterer wichtiger Bereich ist das Erstellen bzw. Vorhalten von Log-Daten. Zu diesem Zweck stehen zwei Datenstrukturen bereit: die Klasse *MissionLogModel* und *SensorLogModel*. Instanzen der Klasse *MissionLogModel* halten allgemeine Status- oder Ereignisinformationen vor, die während einer Mission auftreten. Im Falle von (beispielsweise empfangenen) Sensordaten, die vom MOPS an die Missionsplanungs- und -überwachungssoftware übertragen werden, werden diese Daten zusammen mit einem Zeitstempel und der ID des jeweiligen Sensors innerhalb einer Instanz der Klasse *SensorLogModel* gespeichert. Auch für das Verwalten dieser Daten stehen entsprechende Manager bereit (*MissionLogManager* und *SensorLogManager*).
Für das Kommunizieren der Telemetriedaten des MOPS und somit das Anzeigen innerhalb der Benutzeroberfläche dient die Klasse *TemeletryDataModel*. Enthalten sind neben der aktuellen Position und GPS-Geschwindigkeit (kn/h über Grund) beispielsweise der Ladezustand der Batterie (in %) oder die angestrebten Maschinen- und Ruderwerte. Durch die modulare Struktur des Modells innerhalb der Anwendung ist es sehr leicht möglich einzelne Bereiche zu ergänzen, zu korrigieren oder zu ersetzen.

5.3. Linux Kurzreferenz

5.3.1. Systembeschreibung

Auf dem Beaglebone wird ein Ångström Linux [1] eingesetzt. Es gibt in regelmäßigen Abständen Veröffentlichungen von stabilen Nightly Builds[2]. Auf dem MOPS ist eine Version aus dem März 2012 installiert. Es werden eine Reihe von Systemdiensten genutzt:

- systemd: Verwaltet das Starten und überwacht die Laufzeit von Prozessen
- hwclock: Setzt und liest die Systemzeit der externen RTC
- socat: Stellt die seriellen Schnittstellen als TCP-Socket zur Verfügung
- openjdk: Java 1.6 dient als Plattform für die MOPS-Software

5.3.2. systemd

Systemd [9] ersetzt den aus anderen Linux Distributionen bekannten *init* Dienst. Zu jedem Dienst gehört ein *service file*, in welchem Informationen zum Start- und Laufzeitverhalten beschrieben werden. Das Tool *systemctl* dient zur Steuerung von Diensten. Benutzerdefinierte *service files* sollten in */etc/systemd/system* abgelegt werden.

Für die MOPS-Software gibt es einen Systemdienst *mops.service*, welcher die Java-Anwendung startet und bei Bedarf neu startet. Zuvor wird ein Initialisierungsskript durch den Dienst *inithw.service* ausgeführt, um Hardwarekonfigurationen zu setzen.

5.3.3. Hardwarekonfiguration

Zur Verwendung der Hardware muss diese grundlegend konfiguriert werden. Dies betrifft einerseits die Auswahl der Funktion eines IO-Pins sowie andererseits die Konfiguration der Peripherie. Im Folgenden wird die verwendete Hardware grob beschrieben.

5.3.3.1. IO-Konfiguration

Die Konfiguration der Verwendung eines IO-Pins erfolgt im *sysfs*. Dazu liegt für jeden IO-PIN ein Datei im Pfad */sys/kernel/debug/omap_mux/* bereit, in welche der Pin-Modus geschrieben wird. Der Name der Datei entspricht der Pin Bezeichnung im Hardware-Handbuch des Prozessors. Die verfügbaren Modi sind in Tabelle 5.4 beschrieben. Wird von der Datei gelesen, so wird eine Beschreibung der aktuellen Konfiguration menschenlesbar ausgegeben.

[2] Automatisch erzeugte Veröffentlichungen aus dem Entwicklungsrepository

Modus-Bit	Beschreibung
5	Eingang (0) oder Ausgang (1)
4	Pullup (1) oder Pulldown (0)
3	Pullup/down aktiviert (0) oder deaktiviert (1)
2-0	Modus/Auswahl der Peripherie (0-7, siehe Hardware Handbuch)

Tabelle 5.4.: Auflistung verfügbarer Einstellungen für einen IO-Pin [3]

5.3.3.2. PWM

Zur Ansteuerung der Motoren wird eine PWM der Peripherie *ehrpwm* genutzt. Für diese Peripherie muss zuerst der Takt aktiviert werden. Hierzu wird ein Programm [3] verwendet, welches die entsprechenden Register setzt. Dann kann über den PWM-Treiber im Kernel die gewünschte Ausgangskonfiguration, in diesem Fall eine Periode von 20 ms bei einer An-Zeit von 1500 µs bis 2500 µs, über die Pfade */sys/class/pwm/ehrpwm.1:0/period_ns* bzw. */sys/class/pwm/ehrpwm.1:0/duty_ns* gesetzt werden.

5.3.3.3. RTC

Die batteriegepufferte RTC ist an den I2C 2 angeschlossen. Zur Verwendung muss das entsprechende Kernelmodul geladen und das I2C Gerät hinzugefügt werden. Danach kann mit *hwclock* die Zeit gelesen oder gesetzt werden. Der *systemd*-Service *rtc.service* initialisiert die RTC und setzt beim Systemstart die Systemzeit nach der Zeit aus der externen RTC: *hwclock -s -f /dev/rtc1; hwclock -w -f /dev/rtc0*

5.3.3.4. Serielle Schnittstellen

Damit der Zugriff auf die an den seriellen Schnittstellen angeschlossenen Module aus der Java-Anwendung einfach sowie gegebenenfalls zu Diagnosezwecken auch über Netzwerk erfolgen kann, wird die Schnittstelle per *socat* auf einem TCP-Socket bereitgestellt. Nachdem mit *stty* die Grundeinstellung von Baudrate, Datenflusskontrolle und Steuerzeichen erfolgt ist, kann der Dienst gestartet werden.

Ein Beispielaufruf: *socat /dev/ttyO2 TCP-LISTEN:10002,fork*

5.4. Onboard

In diesem Abschnitt wird die Onboard-Software des MOPS vorgestellt, also die Software welche auf dem MOPS selber ausgeführt wird und diesen steuert. Bei der Onboard-Software handelt es sich um ein Java Programm, welches von einem Linux-System auf dem BeagleBone (siehe 4.2.1) ausgeführt wird. Die Software ist objektorientiert und modular entwickelt. So lassen sich einzelne Funktionsbereiche austauschen, ohne dass große Anpassungen in anderen Bereichen notwendig sind.

In Abbildung 5.6 wird die grobe Struktur der Onboard-Software, ihre Hauptmodule sowie die wichtigsten Interaktionspfade zwischen den Hauptmodulen dargestellt. Auf der Darstellung der Interaktionen des Launchers und des System Monitors wird aus optischen Gründen verzichtet, da diese mit den meisten anderen Modulen interagieren. Es sollte auch beachtet werden, dass die Hauptmodule größtenteils in sich wiederrum modular sind. Die Hauptmodule finden sich auch im Wesentlichen in der Java-Packagestruktur wieder. Jedes dieser Packages hat dabei eine zentrale Klasse, durch die das Modul z. B. initialisiert wird. Diese Klassen sind dabei zumeist Singelton-Instanzen. Hinzu kommen noch die Packages *Common* (gemeinsam genutzte Klassen), *Util* (diverse Hilfsklassen) und *Events* (diverse Event Klassen). Im folgenden werden nun einzelne Bereiche der Onboard Software näher erläutert.

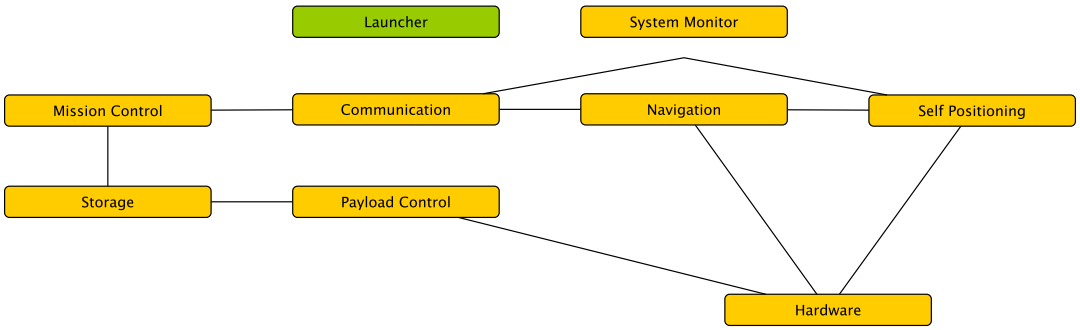


Abbildung 5.6.: Struktur der Onboard-Software. Hinweis: Interaktionpfade mit dem Launcher und dem System-Monitor werden nicht dargestellt.

5.4.1. Konfiguration

Die Onboard-Software des MOPS ist zu großen Teilen parametrisierbar. Diese Parametrisierung erfolgt in Properties-Dateien. Die einzelnen Properties haben dabei einen zweiteiligen Namen mit folgendem Aufbau: *Modulname.PropertyName*. So heisst z. B. die Property, die den Standardmodus des Navigationssystems definiert *navigation.defaultMode*. Es ist sowohl möglich die Properties in der Hauptkonfigurationdatei *mops.properties* abzulegen, als auch in modulspezifischen Properties-Dateien wie z. B. *navigation.properties*. Auch Mischformen sind denkbar. Hierbei wird bei widersprüchlichen Angaben die modulspezifische Properties-Datei bevorzugt.

In der Software wird der Zugriff auf die Properties-Dateien in der Klasse *ConfigurationManager* geregelt. Diese Klasse ermöglicht auch das Ändern von Properties zur Laufzeit und informiert andere Teile der MOPS-Software mithilfe von Events, über diese Änderungen. Damit eine Änderung von Properties zur Laufzeit wirksam wird, müssen die davon betroffenen Teile der Software ggf. auf solche Events reagieren und sich entsprechend umkonfigurieren.

5.4.2. Launcher

Der *Launcher* enthält die Main-Methode, die beim Starten der Software aufgerufen wird. Diese initialisiert die einzelnen Module der MOPS-Software indem sie Instanzen der entsprechenden Klassen anlegt. Der *Launcher* stellt nach dem Hochfahren der Software ein einfaches Konsoleninterface zur Verfügung, über das dem MOPS gewisse Kommandos gegeben werden können. Dieses ist primär zu Testzwecken (z. B. im Simulationsmodus) und nicht zum regulären Betrieb gedacht. Über eine shutdown-Methode kann der *Launcher* die Onboard-Software sauber beenden. Eine weitere wichtige Funktion des *Launchers* ist das Verwalten des aktuellen Betriebsmodus des MOPS. Es sind drei Betriebsmodi vorhanden:

- *Fully Stopped*: Die Software läuft, jedoch ist die Aktorik des MOPS abgeschaltet. Dieser Modus ist z. B. nach dem Hochfahren oder nach dem Auslösen des Software-Notaus in der Offboard-Software aktiv.
- *Remote Controlled*: Der MOPS wird über die Offboard-Software ferngesteuert.
- *Autonomous*: Der MOPS fährt autonom und wird vom Navigationssystem der Software gesteuert.

5.4.3. Aktorik

Die Ansteuerung der Aktorik des MOPS ist Teil des Hardware-Moduls. Sie erfolgt in der Klasse *ActuatingSystem*. Diese kapselt sowohl die Ruder- als auch die Motorsteuerung, wobei für beides Werte im Bereich von -100 bis +100 möglich sind. Dieses abstrakte Interface zur Steuerung der Aktorik hat gegenüber einem Interface, dem bspw. eine tatsächliche Fahrgeschwindigkeit oder Drehrate übergeben wird, den Vorteil, dass es auch bei Änderungen der Hardware weiter unverändert genutzt werden kann. Dies war im Rahmen des MOPS-Projektes bereits einmal nötig, als der prototypische Antriebsaufbau mit einer Schraube und einem Ruder, zu einem zweimotorigen Aufbau ohne Ruder geändert wurde. In der aktuellen Implementierung wird der Ruderwert in eine Drehzahldifferenz zwischen den beiden Motoren umgerechnet und so die Richtungsänderung erreicht. Wie dies geschieht kann (zum Teil) in der Konfiguration angepasst werden.

Zur eigentlichen Drehzahlsteuerung der Motoren wird ein PWM Signal erzeugt, dass über Device-Files an die Hardware weitergegeben wird. Auch für Anpassungen in diesem Bereich stehen diverse Parameter in den Properties-Dateien zur Verfügung.

5.4.4. Navigationssystem

Das Navigationsmodul ist für das autonome fahren des MOPS verantwortlich. Es navigiert den MOPS, z. B. im Rahmen einer Mission, zu den von der Missionskontrolle vorgegeben Punkten. Es ist zudem in der Lage den MOPS an seiner aktuellen Position zu halten. Um seine Aufgabe zu erfüllen interagiert das Modul über einen Datenpool mit dem Self-Positioning-System (siehe 5.4.9) und direkt mit dem Actuating-System, welches wiederum

die Motoren ansteuert. Die Navigation des MOPS hat zwei zentrale Funktionsbereiche: die Regelung und die Planung.

5.4.4.1. Regelung

Die Regelung ist dafür verantwortlich die Aktorik des MOPS so zu steuern, dass ausgehend von der aktuellen Position eine gegebene Zielposition angefahren wird. Sie erfolgt in der abstrakten Klasse *NavigationController*. Hierdurch ist es möglich unterschiedliche Algorithmen zur Regelung zu implementieren. Die *NavigationController*-Klasse implementiert dabei das Management eines Regelungsthreads, der über Methoden gestartet und gestoppt werden kann. Dieser Thread reagiert über Events auf das Eintreffen neuer GPS- oder Sensorboarddaten und führt dann einen neuen Regelungszyklus aus.

Im Rahmen des Projektes wurde nur ein Regelungsalgorithmus implementiert. Dieser befindet sich in der Klasse *CourseDifferentialController*. Im Folgenden wird die in Abbildung 5.7 dargestellte Regelstrecke des Reglers erläutert. Zu Beginn jedes Regelungszyklus wird ein Kurs von der aktuellen Position zu der Zielposition berechnet. Hierfür werden aus der Navigation bekannte Formeln verwendet [29]. In weiteren Schritten wird der errechnete Kurs mit dem tatsächlich gefahrenen verglichen. Dieser wird von dem Self-Positioning-System mithilfe des Sensorboards ermittelt. Die Abweichung zwischen dem Zielkurs und dem tatsächlichen Kurs wird nun als Regelabweichung an einen PID-Regler [19] übergeben. Die vom PID-Regler ermittelte Stellgröße wird abschliessend als neuer Ruderwert an die Aktorik des MOPS übergeben. Mit einem zusätzlichen PID-Regler wird, sobald ein zuvor konfigurierter Abstand zum Ziel unterschritten ist, auch die Fahrgeschwindigkeit des MOPS, ausgehend von dem Abstand zum Ziel geregelt, damit nicht über dieses hinweg gefahren wird. Solange der definierte Abstand überschritten ist, fährt der MOPS mit seiner konfigurierten Reisegeschwindigkeit.

Die beiden PID-Regler (Kurs und Geschwindigkeit) könnnen getrennt voneinander in den Properties-Dateien konfiguriert werden. Die hier bisher verwendeten Werte spiegeln die Erfahrungen aus den Testfahrten wieder. Allerdings hat sich bereits gezeigt, dass bei stärkerer Strömung bzw. Wind, wenn diese senkrecht zur Fahrtrichtung ausgerichtet sind, die Regelung Schwierigkeiten hat einen geraden Kurs zu fahren. Tatsächlich gefahren wird dann eine Kurve. Dies kann eventuell durch einen grösseren I-Faktor in dem Regler vermindert werden. Denkbar ist, dass sich solche Probleme erst durch das implementieren eines anderen Regelalgorithmus komplett lösen lassen.

Da ein Ziel aufgrund technischer Grenzen wie z. B. die GPS-Genauigkeit und die Grenzen der Aktorik, sowie durch äußere Einflüsse wie z. B. Strömung und Wind, nie zu 100 % erreicht werden kann, führt dies dazu das der MOPS nachdem er seinem Ziel sehr nahe ist, permanent in kurzen Abstand um dieses herumfährt. Dadurch das die Regelung ein Ziel nicht als erreicht erkennt, schaltet sie dementsprechend auch nicht ab und stoppt die Aktorik nicht vollständig. Eine Lösung für dieses Verhalten erfolgt durch die im folgenden Abschnitt beschriebene Planungskomponente des Navigations-Systems.

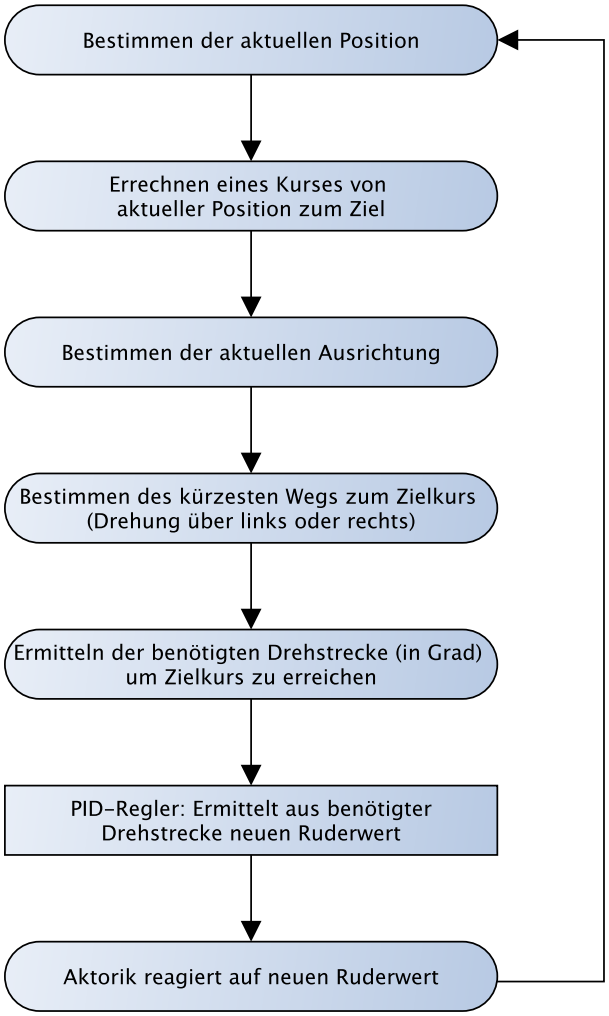


Abbildung 5.7.: Regelstrecke den *CourseDifferentialController*

5.4.4.2. Planung

Die Planungskomponente der Navigation findet sich im Wesentlichen in der Klasse *NavigationSystem* und hat sehr verschiedene, der Kurs- und Geschwindigkeitsregelung übergeordnete Aufgaben. Sie stellt zum einen das Interface zur Navigation des MOPS dar. Dies bedeutet, dass die anderen Komponenten des MOPS, bezogen auf das autonome Fahren im Wesentlichen mit der Klasse *NavigationSystem* bzw. mit von dieser Klasse ausgelösten Events interagieren. Sie nimmt so z. B. neue Fahrziele entgegen und informiert darüber, wenn diese erreicht sind. Auch über bestimmte erkannte Fehlerzustände, z. B. wenn die Navigation aufgrund von Ausfällen anderer Komponenten nicht mehr möglich ist, wird mithilfe von Events informiert.

Eine weitere Aufgabe der Planung ist überhaupt zu erkennen, wann ein Ziel erreicht ist. Jedes Navigationsziel, die durch Objekte der Klasse *NavigationPoint* repräsentiert werden, verfügt neben den eigentlichen Zielkoordinaten noch über einen Radius, der angibt wie ge-

nau das Ziel erreicht werden soll. Sobald der Abstand vom Ziel diesen Radius unterschreitet wird ein Event ausgelöst, welches über das erreichen des Ziels informiert. Desweiteren wird die Regelung und damit auch die Aktorik abgeschaltet, sobald der Abstand zum Ziel kleiner ist als der Radius abzüglich eines, in der Property *navigation.engineStopRange* definierten Puffers. So wird Energie gespart, falls länger an einem Punkt verblieben werden soll. Wird der Abstand wieder größer, wird die Regelung reaktiviert. Sollte der MOPS, z. B. aufgrund starker Strömung, nicht beim Ziel verweilen können wird ebenfalls wieder ein Event ausgelöst, sobald der Abstand wieder größer wird als der definierte Radius.

Es ist geplant in Zukunft auch eine erweiterte Routenplanung zu implementieren, die über das Abfahren vorher definierter Punkte hinausgeht. Eine solche Planung kann z. B. stationäre und mobile Hindernisse oder Strömungs- und Wetterbedingungen berücksichtigen. Sie sollte auch Teil der Planungskomponente des Navigationssystems sein. Die Klasse *NavigationSystem* hat hierfür bereits verschiedene Mechanismen vorgesehen. So wird intern zwischen Wegpunkten und dem eigentlichen Navigationsziel unterschieden. Es gibt darüber hinaus einen Threadpool, der bisher nur einen Thread geinhaltet. Dieser erkennt u.a. ob das Ziel erreicht ist. Es ist angedacht das weitere Planungsthreads innerhalb dieses Pools ausgeführt werden und bspw. Wegpunkte generieren, die zu dem eigentlichen Navigationsziel führen. Für derartige Planungsalgorithmen sind natürlich weitere Maßnahmen erforderlich. So sollte der MOPS um stationären Hindernissen ausweichen zu können über eine Seekarte vefügen. Um mobilen Hindernissen auszweichen ist eine Sensorik notwendig, um diese zu erkennen. Für die Berücksichtigung von Strömung und Wetter sind ebenfalls entsprechende Informationen (z. B. Strömungs- und Wetterkarten) erforderlich.

Die MOPS-Navigation kennt unterschiedliche Betriebsmodi: *Target*, *Hold Position*, *Drift* und *Error*. Diese werden nun erläutert.

Target Der gebräuchlichste Modus des Navigations-Systems. Es wird ein Ziel angefahren, bzw. bei einem Ziel verweilt, welches zuvor übegeben wurde. Dabei wird über erreichen und verlassen des Ziels, wie bereits beschrieben, informiert und die Aktorik gegebenfalls gestoppt und wieder gestartet.

Hold Position Dieser Modus kann verwendet werden, wenn der MOPS an seiner aktuellen Position verbleiben soll. Bei dem Wechsel in diesen Modus wird sich diese gemerkt und der MOPS versucht nun innerhalb eines, in der Property *navigation.holdPositionRadius* definierten, Radius um an dieser Position zu bleiben. Die Aktorik wird auch in diesem Fall gegebenfalls gestoppt um Energie zu sparen.

Drift Dieser Modus deaktiviert die Aktokrik des MOPS weitestgehend. Die Navigation soll hierbei nurnoch aktiv werden um eine bevorstehende Kollision zu vermeiden, sich aber ansonsten jedoch passiv verhalten. Da die Kollisionsvermeidung als solche bisher noch nicht implementiert ist, unterscheidet sich dieser Modus zurzeit in der Praxis vom

Verhalten her nicht von der kompletten Deaktivierung des Navigations-Systems. Er ist also für zukünftige Entwicklungen gedacht.

Error Dieser Modus repräsentiert einen Fehlerzustand in den gewechselt wird, falls bei einem Moduswechsel ein nicht behandelbarer Fehler auftritt. Solange das Navigations-System in diesem Modus ist verhält es sich passiv.

5.4.5. Storage

Im Storage-Modul erfolgt die persistente Datenhaltung des MOPS. Es wurde bei der Entwicklung auf zwei Dinge Wert gelegt. Zum einen sollte es ein gemeinsames Storage-System für alle Komponenten der Onboard-Software geben. Es sollte also nicht jede Komponente ihre Datenhaltung komplett selbst implementieren. Zum anderen sollte es mit möglichst wenig Änderungen an anderen Komponenten der Onboard-Software möglich sein, das Storage-System auszutauschen.

Es hat sich als nicht realisierbar herausgestellt eine Datenhaltung zu entwickeln, die keine Kenntnis von den zu speichernden Daten hat und von Komponenten verwendet wird, die wiederum keine Kenntnis von der Datenhaltung haben. Daher wurde stattdessen ein dreiteiliger Ansatz gewählt. Zu einer abstrakten *StorageHandler*-Klasse und einer Klasse die die Datenhaltung nutzt (z. B. die *MissionControl*-Klasse) kommt eine dritte Komponente, die zwischen beiden vermittelt. Im vorliegenden Beispiel ist dies die *MissionControlStorageHandler*-Klasse. In der abstrakten Klasse wird alles abgehandelt was übergreifend für die gesamte Datenhaltung relevant ist, z. B. das Transaktionsmanagement. Die *MissionControlStorageHandler*-Klasse regelt alles was spezifisch für den Bereich *MissionControl* ist, wie z. B. die Struktur der Daten. Die *MissionControl* selber benötigt keine Kenntnis davon wie die Datenhaltung funktioniert und verwendet nur Methoden ihres Storage-Handlers um auf diese zuzugreifen. Abbildung 5.8 stellt diesen Aufbau grafisch dar.

Implementiert wurde die Datenhaltung im Rahmen des Projektes als Sqlite-Datenbank [23]. Dies ermöglicht die Datenkonsistenz mit Schlüsselbeziehungen und Transaktionen sicherzustellen und hilft Datenverluste bei plötzlichen Softwareabstürzen zu minimieren. Bei der Implementierung wurden prepared-Statements mit Bind-Variablen verwendet, da diese ein Performance-Vorteil bringen. Der Grund hierfür ist, dass sie nur einmal bei der Initialisierung geparst werden müssen.

Abbildung 5.8.: Struktur der Datenhaltung auf dem MOPS.

5.4.6. Payload-Control

Die Paylad-Control (Nutzlastkontrolle) soll zukünftig die als Nutzlast mitgeführte wissenschaftliche Sensorik steuern und die von ihr gesammelten Daten ergänzt um Zeit, GPS-Position und evtl. weiteren Informationen mithilfe des Storage Systems speichern. Da im Rahmen des Projektes keine wissenschaftliche Sensorik integriert wurde, ist dieses Modul der Software bisher nicht implementiert. Vorhanden ist lediglich eine Klasse mit weitestgehend funktionslosen Methoden, die einen Teil des Interfaces zur Payload-Control darstellen, damit dieses bereits von anderen Modulen verwendet werden kann.

5.4.7. Mission-Control

Die Mission-Control regelt die Verwaltung und das Abarbeiten von Missionen durch den MOPS. Sie interagiert dabei mit dem Kommunikationssystem um neue Missionen zu empfangen bzw. ihren Status an die Missionsplanungs- und -überwachungssoftware zu übermitteln und mit dem Navigations-System, sowie der Payload-Control um diese Mission abzuarbeiten. Missionen bestehen auf den MOPS aus Missionspunkten. Diese haben einen Navigationspunkt mit Radius (siehe hierzu 5.4.4.2) und eine Verweildauer. Sie können zudem über mehrere Nutzlast-Aktionen, wie z. B. einschalten von Sensor 1, verfügen. Abgelegt werden die Missionen sowie ihr Fortschritt im Storage-System des MOPS. Im Verlauf einer Mission wird ihr Fortschritt regelmässig gespeichert, so dass Missionen im Falle eines technischen Problems oder einer anderen Unterbrechung dort wiederaufgenommen werden können, wo sie unterbrochen wurden.

Die Missionen werden in der Missionsplanungs- und -überwachungssoftware erstellt und an den MOPS übermittelt, der diese dann in seiner internen Datenhaltung ablegt. Sie werden dabei mit einer eindeutigen ID identifiziert. Missionspunkte und Sensoraktivitäten verfügen ebenfalls über eine eindeutige ID. Über die IDs ist es möglich eine spezifische Mission zu starten bzw. fortzusetzen und diese auch später noch zu ändern. Dabei wird über eine Versionsnummer sichergestellt, dass keine Inkonsistenzen entstehen. Bei Änderungen von bereits begonnenen Missionen gibt es noch weitere Konsistenzregeln, die z. B. sicherstellen das kein bereits erledigter Missionspunkt geändert wird.

Im Verlauf einer Mission gibt die Missionkontrolle den gerade aktiven Missionpunkt als Navigationsziel an das Navigations-System. Dieses wartet dann bis der Punkt erreicht ist und löst anschliessend die zum Missionspunkt gehörenden Nutzlast-Aktionen aus. Sollte eine Verweildauer für den Punkt vorgesehen sein, beginnt außerdem ein Timer zu laufen. Sobald dieser abgelaufen ist bzw. keine Verweildauer vorhanden ist, wird nun der nächste Punkt an das Navigations-System übegeben. Sind alle Punkte abgearbeitet, gilt die Mission als beendet. In diesem Fall wird auch ein entsprechendes Event ausgelöst.

5.4.8. Systemüberwachung

Der *SystemMonitor* überwacht kontinuierlich den Zustand des gesamten Systems. Dazu zählen die Batteriespannung, der aktuelle Stromverbrauch, die zur Positionsbestimmung benötigten Systeme (GPS und Sensorboard) (siehe 5.4.9) und der Zustand des Navigations-Systems.

Je nach Art des eingetretenen Zustands wird eine Nachricht in die Log-Datei geschrieben, eine Nachricht an die Missionsplanungs- und -überwachungssoftware gesendet oder sogar aktiv in die Funktionalität des MOPS eingegriffen.

5.4.8.1. Stromverbrauch

Der von den Motoren benötigte Strom ist sehr stark von der Belastung selbiger abhängig. Zur Ermittelung der aktuellen Stromaufnahme wird der BeagleBone-Pin `AIN2` 20 Mal ausgelesen und ein Mittelwert gebildet. Da das BeagleBone häufig beim ersten Auslesen der Pins falsche Werte liefert, wird der Pin einmal ausgelesen, der Wert verworfen und erst danach die 20 Werte aufgenommen. Die einzelnen Lesevorgänge werden von der Klasse *ActuatingSystemCurrentReader* durchgeführt. Diese liest die dem BeagleBoard-Pin `AIN2` zugehörige Datei[3] und parst den Wert in einen Integer.

Der Mittelwert alleine ist jedoch noch nicht aussagekräftig genug. Zur Berechnung wird die Formel 5.1 mit den ermittelten Werten aus Tabelle 5.5 verwendet.

Key	**Value**
$\mathrm{ADC}_{\mathrm{max}}$	4096
U_{ref}	1,8 V
R_{shunt}	0,000 85 Ω
$V_{\mathrm{shuntmonitor}}$	20

Tabelle 5.5.: Werte zur Ermittlung der Stromaufnahme

$$I_{\mathrm{Motorstrom}} = \frac{ADC_{\mathrm{Mittelwert}}}{\mathrm{ADC}_{\mathrm{max}}} * \frac{U_{\mathrm{ref}}}{R_{\mathrm{shunt}} * V_{\mathrm{shuntmonitor}}} \tag{5.1}$$

Übersteigt der so ermittelte Wert 50 A für mehr als 10 Sekunden, wird eine Warnung an die Missionsplanungs- und -überwachungssoftware gesendet.

5.4.8.2. Batteriespannung

Wie auch die Stromaufnahme wird die Spannung aus einem Mittelwert von 20 ausgelesenen Werten bestimmt. Auch hier wird der erste Wert verworfen. Ausgelesen wird hierzu der BeagleBone-Pin `AIN1` vom *VoltageReader* der die dem BeagleBone-Pin `AIN1` zugehörige

[3] /sys/devices/platform/omap/tsc/ain2

Datei[4] ausliest und den in ihr stehenden Wert in einen Integer parst. Dieser Mittelwert wird mit einem konstanten Faktor, welcher sich durch die Auslegung des Spannungsteilers zu 0.00924 ergibt, multipliziert.

Um die Batteriespannung auf den aktuell aufgenommenen Strom zu beziehen, wird eine korrigierte Spannung nach folgender Formel 5.2 mit einem ermittelten Innenwiderstand des Akkus von R_{internal} =0,025 Ω berechnet.

$$U_{\text{bat, corrected}} = U_{\text{Mittelwert}} * R_{\text{internal}} * I_{\text{Motorstrom}} \tag{5.2}$$

Für weitere Berechnungen wird der Spannungswert in einen prozentualen Ladezustand überführt (Formel 5.3). Dafür werden folgende Kenndaten des Akkumulators benötigt:

- Entladespannung ($U_{\text{Discharge}}$ =10,5 V)
- Ladeschlussspannung ($U_{\text{LoadDischarge}}$ =9,9 V)
- maximale Spannung der Batterie ($U_{\text{FullCharge}}$ =12,48 V)

$$\text{Ladezustand (in \%)} = \frac{U_{\text{bat, corrected}} - U_{\text{Discharge}}}{U_{\text{FullCharge}} - U_{\text{LoadDischarge}}} * 100 \tag{5.3}$$

Fällt der Ladezustand der Batterie unter 20 % werden eine eventuell laufende Mission und das Navigations-System gestoppt, der MOPS in den *Drift*-Modus versetzt und eine Nachricht an die Missionsplanungs- und -überwachungssoftware gesendet.

Wird ein Ladezustand von unter 5 % erreicht, werden eine eventuell laufende Mission und das Navigations-System gestoppt. Der MOPS wird *nicht* in den *Drift*-Modus versetzt, um die Batterie zu schonen. Somit können noch möglichst lange Heartbeats an die Missionsplanungs- und -überwachungssoftware gesendet werden, wodurch ein Einsammeln des MOPS erleichtert wird.

Sollte $U_{\text{bat, corrected}} < U_{\text{Discharge}}$ oder $U_{\text{Mittelwert}} < U_{\text{LoadDischarge}}$ sein, wurde ein kritischer Ladezustand erreicht. Um die Batterie vor Schäden zu bewahren, wird eine Nachricht an die Missionsplanungs- und -überwachungssoftware gesendet und daraufhin das System heruntergefahren.

5.4.8.3. Heartbeats

Der MOPS sendet alle 5 Sekunden einen Heartbeat an die Missionsplanungs- und -überwachungssoftware. Dieser dient der Überwachung und Positionsbestimmung des MOPS. Dazu wird ein *ScheduledHeartbeatSender* gestartet, bei dem es sich wiederum um einen *ScheduledExecutorService* handelt, der mithilfe eines *ThreadPools* kontinuierlich in vorgegebenem Intervall einen Heartbeat sendet.

Ein Heartbeat enthält folgende Daten:

[4] /sys/devices/platform/omap/tsc/ain1

- GPS-Position
- GPS-Datum und -Uhrzeit (UTC)
- Ausrichtung (in Grad)
- Ladezustand des Akkus (in %)
- Geschwindigkeit über Grund (in kn/h)

5.4.8.4. Überwachung der GPS- und Ausrichtungswerte

Um zu überprüfen, ob kontinuierlich neue GPS- und Ausrichtungswerte aus der Hardware ausgelesen werden, wird der *SystemMonitor* beim *SelfPositioningSystem* registriert. Bei jedem Eintreffen von GPS- oder Ausrichtungswerten wird der Zeitpunkt gespeichert und in der Hauptschleife des *SystemMonitors* wird überprüft, ob die Differenz zwischen der aktuellen Zeit und dem Zeitpunkt des letzten Eintreffens einen gewissen Grenzwert überschreitet. Sollte dies der Fall sein, wird ein entsprechendes Event geworfen (z. B. `GpsStatus.NOT_WORKING`). Sollten zu einem späteren Zeitpunkt erneut Daten eintreffen, wird das passende Event geworfen (z. B. `GpsStatus.WORKING_AGAIN`). Dadurch können von GPS- oder Ausrichtungswerten abhängige Systeme direkt beeinflusst werden.

5.4.9. Positionsbestimmung

Zur Positionsbestimmung wird das *SelfPositioningSystem* verwendet. Es dient als Schnittstelle zwischen nachfolgend beschriebenen Hardware-Klassen und dem Rest der Software. Hierzu aggregiert und bereitet es sämtliche Daten auf, die für eine genaue Positionsbestimmung und somit auch für die Navigation notwendig sind. Dazu gehören sowohl die GPS-Koordinaten, die Geschwindigkeit über Grund, als auch die Ausrichtung.

Hierzu greift das *SelfPositioningSystem* über eine *Factory* (die *HardwareFactory*) auf Klassen zu, die direkt mit der jeweiligen Hardware interagieren. Diese Factory liefert Singleton-Objekte der Klassen zurück, damit sichergestellt ist, dass z. B. die GPS-Position nur an einer Stelle ausgelesen wird. Das *SelfPositioningSystem* registriert sich als Listener bei den Hardware-Klassen damit diese diesem mittels Events mitteilen können, dass sie neue Daten ausgelesen haben, welche vom *SelfPositioningSystem* verarbeitet werden können.

Das *SelfPositioningSystem* wiederum benachrichtigt alle auf ihm registrierten Listener, wenn neue Positionsbestimmungsdaten eingegangen sind.

5.4.9.1. GPS

Die GPS-Hardware (siehe Kapitel 4.2.3.1) überträgt jede Sekunde eine Reihe von `NMEA` [2]-Datensätzen (unter anderem RMC, GGA und GLL) an die Onboard-Software. Aus diesen werden mithilfe der *Java Marine Api* [27], welche komplett in den Code integriert wurde, um einige Fehler (u.a. blockierende Threads, aktives Warten) beheben zu können, die benötigten Daten (Position, Geschwindigkeit über Grund, Uhrzeit und Datum) ausgelesen.

Bei jedem neuen Datensatz werden alle *Listener* (in diesem Fall nur das *SelfPositioningSystem*) notifiziert.

Da Koordinaten im NMEA-Protokoll durch positive Zahlen und einem Zusatz (Ost oder West bzw. Nord oder Süd) angegeben werden, die Onboard-Software aber mit Längengraden von $-180°$ bis $+180°$ und Breitengraden von $-90°$ bis $+90°$ rechnet, werden die Koordinaten vor dem Speichern in den *DatenPool* vom *SelfPositioningSystem* in das passende Format umgerechnet. Aus $+90°$W wird also $-90°$, wohingegen aus $+90°$E $+90°$ wird.

5.4.9.2. Kompass

Zusätzlich zu den erwähnten Daten liefert die GPS-Hardware auch die aktuelle Ausrichtung. Da diese aber nicht durch einen Magnetkompass erzeugt, sondern über Bewegung ermittelt wird, ist sie für eine Navigation zu ungenau – gerade während Standphasen oder sehr langsamer Fahrt ist auf die errechneten Werte kein Verlass. Deshalb wird vom *Sensorboard* (siehe Kapitel 4.2.3.2) auf Anfrage eine per Magnetkompass ermittelte Ausrichtung an die Onboard-Software geliefert. Hierzu wird alle 500 ms die Zeichenfolge `#ot#f` gefolgt von einem *Carriage Return* und einem *Line Feed* an das Sensorboard gesendet. Das Sensorboard gibt daraufhin Werte zu seiner aktuellen Lage (*Yaw*, *Pitch* und *Roll*) in folgendem Format zurück:

`#YPR=-142.28,-5.38,33.52` gefolgt von *Carriage Return* und *Line Feed*

Interessant ist hiervon (momentan) nur der *Yaw*-Wert, da es sich bei diesem um die Magnetkompass-Ausrichtung handelt. Auf diesen Wert wird ein Korrekturfaktor addiert, um eine beim Kalibrieren entstandene Missweisung zu kompensieren.

Ist die Verarbeitung des *Yaw*-Wertes abgeschlossen, werden alle *Listener* (in diesem Fall nur das *SelfPositioningSystem*) notifiziert. Das *SelfPositioningSystem* speichert diesen Wert daraufhin in den *DatenPool*.

5.4.10. Kommunikation

Die gesamte Kommunikation zwischen der Onboard-Software und der Missionsplanungs- und -überwachungssoftware findet über die *Communication*-Library statt. Auf der Onboard-Seite dient der *CommunicationController* als Schnittstelle zwischen der Communication-Library und dem Rest der Onboard-Software.

Der *CommunicationController* wird bei der Communication-Library registriert, so dass er sowohl bei eingehenden Nachrichten, als auch bei einer Änderung der Verbindungsstatus von XBee und/oder WLAN benachrichtigt wird.

5.4.10.1. Initialisierung der Kommunikation

Beim Initialisieren der Kommunikation wird anhand von Einträgen in Properties-Dateien (`communication.properties`) entschieden, wie eine Kommunikation stattfinden soll. Es sind drei Verbindungsmodus möglich:

- Nur WLAN
- Nur XBee
- WLAN und XBee

Beim Aufbau einer WLAN-Verbindung öffnet der *CommunicationController* eine Socket-Verbindung auf einem aus der Properties-Datei ausgelesenen Port.

Beim Aufbau einer XBee-Verbindung müssen mehr Schritte durchgeführt werden. Zuerst werden dem XBee des MOPS eine IP-Adresse und ein Port zugewiesen. Dann wird die IP-Adresse des Ziel-XBees gesetzt. Daraufhin wird überprüft, ob eine Verschlüsselung der XBee-Kommunikation stattfinden soll und gegebenenfalls ein *Encryption-Key* verwendet. Zum Schluss wird die Sendeleistung des XBee-Moduls konfiguriert. Alle Werte und Einstellungen liegen in der gleichen Properties-Datei wie auch die WLAN-Einstellungen.

5.4.10.2. Änderung des Verbindungsstatus

Bei jeder Veränderung des Verbindungsstatus wird überprüft, ob sich der MOPS im Fernsteuermodus befindet. Trifft dies zu und sollte weder eine Verbindung mittels XBee noch über WLAN bestehen, wird der Fernsteuermodus beendet und der MOPS in den *FullyStopped*-Mode versetzt. Dadurch werden alle der Navigation bzw. der Fortbewegung dienlichen Teilkomponenten des MOPS ausgeschaltet.

5.4.10.3. Senden von Nachrichten

Nachrichten werden grundsätzlich *untyped* (siehe Kapitel 5.1.1) versendet. Einzige Ausnahme bildet das Senden von Heartbeats, da nur der jeweils aktuellste Heartbeat von Interesse ist.

5.4.10.4. Empfangen neuer Nachrichten

Tritt ein *MessageReceived*-Event ein wird überprüft, ob es sich bei der Nachricht um eine *ByteArray*-Message oder eine *TransferObject*-Message (siehe Kapitel 5.1.4) handelt.

Handelt es sich um eine *ByteArray*-Message wird anhand des ersten Bytes entschieden welche Aktionen folgen sollen. Dies können u.a. das Starten einer Mission, das Starten des Fernsteuermodus oder das Anfordern von Missionsdaten sein.

Beinhaltet die Nachricht hingegen ein *TransferObject* wird anhand des *TransferObjectType* entschieden welche Aktionen durchgeführt werden müssen. Dies sind u.a. das Anlegen

einer neuen Mission oder das Ändern einer bereits auf dem MOPS befindlichen Mission.

5.4.11. Fernsteuermodus

Nach Erhalt der *Fernsteuermodus-Starten*-Nachricht wird der *RemoteControlWatcher* gestartet. Dieser versetzt den MOPS in den *RemoteControlMode*, indem sämtliche Mechanismen zum autonomen Fahren gestoppt werden.

Im Fernsteuermodus werden die Steuerkommandos direkt über das *ActuatingSystem* an die Motoren geleitet ohne vorher den Umweg über das Navigations-System zu nehmen. Die Werte des zuletzt empfangenen Fernsteuerkommandos bleiben solange erhalten, bis neue Werte empfangen werden.

Es wird zyklisch überprüft, ob kontinuierlich Fernsteuerkommandos eingehen. Sollten eine voreingestellte Zeit (aktuell 5 Sekunden) keine Fernsteuerkommandos eingehen, wird der Fernsteuermodus automatisch beendet. Dies hat den Grund, dass der MOPS nicht durch einen Kommunikationsabbruch oder durch das Verlieren von Datenpaketen außer Kontrolle gerät.

5.4.12. Logging

Zu Logging-Zwecken wird das *LOGBack* [14]-Framework verwendet. Dieses Framework bietet gegenüber z. B. Log4J unter anderem eine größere Auswahl an Loggern/Appendern (z. B. *AsyncAppender*, *DBAppender*). Vor allem aber soll es bedeutend performanter sein.

5.4.12.1. Asynchroner Datei-Logger

Um Log-Nachrichten in eine Datei zu loggen wird der *AsyncAppender* in Verbindung mit einem *FileAppender* verwendet. Bei diesem Vorgehen werden die Log-Nachrichten nicht direkt in die Log-Datei geschrieben, sondern in eine *Queue*. Asynchron arbeitende Worker-Threads mit niedriger Priorität wiederum lesen die Einträge aus der Queue und schreiben diese in die Log-Datei. Das bietet den Vorteil, dass das Gesamtsystem während eines Logging-Vorgangs nicht blockiert.

Bei den erstellten Log-Dateien handelt es sich um CSV-Dateien, in denen die einzelnen Einträge durch Semikolons (`;`) getrennt sind.

```
<Uhrzeit im Format: HH:mm:ss.SSS)>;<Threadname>;<LOG-Level>;<Loggername>;
<Log-Nachricht>
```

5.4.12.2. Konsolen-Logger

Zusätzlich zum Logging in Dateien werden die zu loggenden Nachrichten auch in der Konsole angezeigt. Damit diese aber nicht überladen wird, ist der *ConsoleAppender* so eingestellt, dass er nur Nachrichten ab dem *INFO*-Level anzeigt.

5.4.12.3. Versenden von Log-Nachrichten

Damit es möglich ist Live-Log-Nachrichten vom MOPS in der Missionsplanungs- und -überwachungssoftware anzeigen zu können, wurde der *ContinuousLogSenderAppender* erstellt.

Zum Starten dieses Appenders wird von der Missionsplanungs- und -überwachungssoftware ein Log-Level vorgegeben und alle Nachrichten die dem Level entsprechen ($\geq$ Level) werden an die Missionsplanungs- und -überwachungssoftware gesendet.

5.5. Simulation

Die Simulation wurde entwickelt, um die verschiedenen Teile der Onboard-Software, die auf Informationen von der Hardware angewiesen sind, testen zu können. Dies beinhaltet z. B. die GPS-Position und die Ausrichtung des MOPS. Hierbei ist die Simulation allerdings nur eine sehr grobe Darstellung der wirklichen Welt und dient lediglich als Funktionstest.

Die Software benötigt einige Parameter, um Simulieren zu können:

- Current Speed: Die Strömungsgeschwindigkeit - lässt sich während des Simulierens anpassen
- Current Direction: Strömungsrichtung - ist überall gleich und lässt sich während des Simulierens anpassen
- Start Latitude: Breitengrad, auf dem der MOPS starten soll
- Start Longitude: Längengrad, auf dem der MOPS starten soll
- Starting Direction: Kompassstellung, mit der der MOPS starten soll
- Turn Radius: Simulationsparameter, der die Größe des Wendekreises angibt
- Max Speed: Die maximale Geschwindigkeit, die der MOPS erreichen kann, da der MOPS prozentuale Angaben zum Motor überträgt

Mit Hilfe dieser Daten und den vom MOPS übermittelten Daten der Motorauslastung und Ruderstellung kann dann die Simulation die Änderungen in der Position und Ausrichtung berechnen.

Die neuen Daten werden dann dem MOPS über Adapterklassen zur Verfügung gestellt. Diese Adapterklassen verhalten sich für den MOPS identisch zu den normalen Hardwareklassen. Während der Simulation ersetzen sie diese also. Die restliche Onboard-Software bleibt dabei unverändert.

6. Benutzerhandbuch

6.1. Missionsplanungs- und -überwachungssoftware

Im Folgenden möchten wir auf die Verwendung der *Missionsplanungs- und -überwachungs-software* eingehen und dem Benutzer so eine Hilfestellung beim Einsatz der Anwendung bereitstellen. Die Software stellt Funktionalitäten bereit um bspw. Missionen des Wasserfahrzeugs zu planen, diese an die Hardware zu übertragen, den Ablauf sowie den Status einzelner Missionen zu überwachen, oder Befehle an MOPS zu übertragen.

6.1.1. Installation

Für die Installation der Software ist es lediglich notwendig den Dateiordner, in dem die Anwendung enthalten ist, von dem beigefügten Datenträger auf den Computer des Anwenders zu kopieren. Für die Ausführung der Software wird eine installierte Java-Laufzeitumgebung der Version 7 oder höher vorausgesetzt. Innerhalb der Software besteht die Möglichkeit das Wasserfahrzeug über einen Gamecontroller der Playstation 3 der Firma Sony (bspw. DualShock) fernzusteuern. Soll diese Funktionalität Anwendung finden, ist es notwendig, dass die Treiber dieses Gerätes vor dem Start der Anwendung installiert werden. Da die Firma Sony den Einsatz des Controllers an herkömmlichen Computern nicht unterstützt, muss auf Treiber von Drittanbietern zurückgegriffen werden. Eine gute Möglichkeit stellt das Projekt „MotioninJoy“[1] dar. Neben den Gamecontrollern der Playstation 3 werden auch die der Xbox 360 der Firma Microsoft unterstützt. Anleitungen zur Installation der jeweiligen Treiber sind bspw. im Wiki des Projektes zu finden.[2]
Um die Anwendung zu starten führen Sie die Datei *MOPSConfigurationTool.jar* durch einen Doppelklick aus, die im Dateiordner der Software zu finden ist.

6.1.2. Benutzeroberfläche

Nach dem Start der Anwendung öffnet sich die Benutzeroberfläche, über die der Benutzer mit der Anwendung, oder bei Bedarf mit der Hardware interagiert. Im oberen Bereich der angezeigten Karte befinden sich die Reiter, über die der Bereich der Anwendung gewechselt werden kann. So besteht die Möglichekeit zwischen der Planung einer Mission, der Überwachung einer (laufenden) Mission und dem Einsehen empfangener Log-Daten

[1] `http://www.motioninjoy.com/`
[2] siehe `http://www.motioninjoy.com/wiki/en/install`

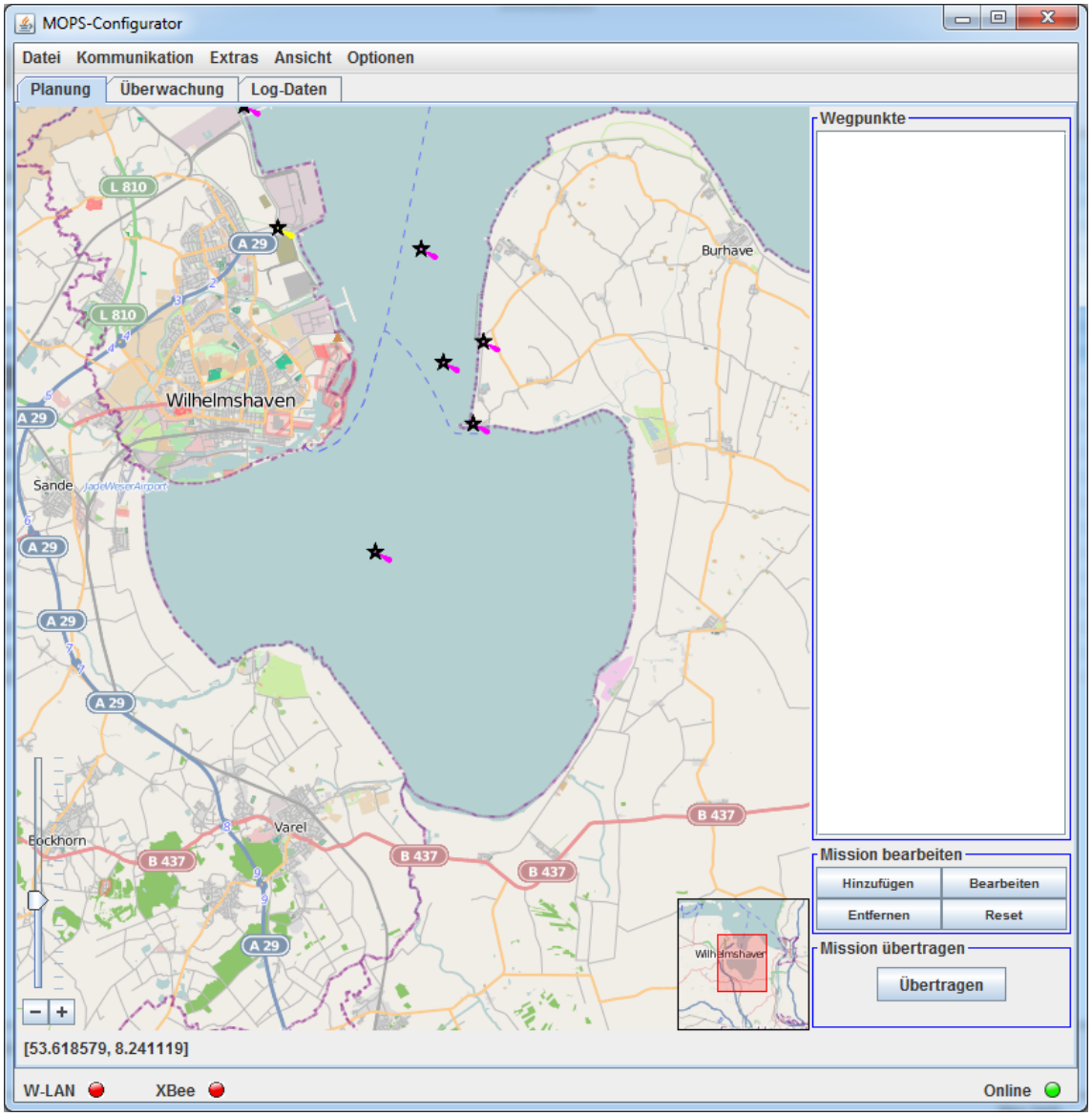


Abbildung 6.1.: Benutzeroberfläche nach dem Start der Anwendung

umzuschalten. Die über den Reitern positionierte Menüleiste stellt zusätzliche Funktiona-

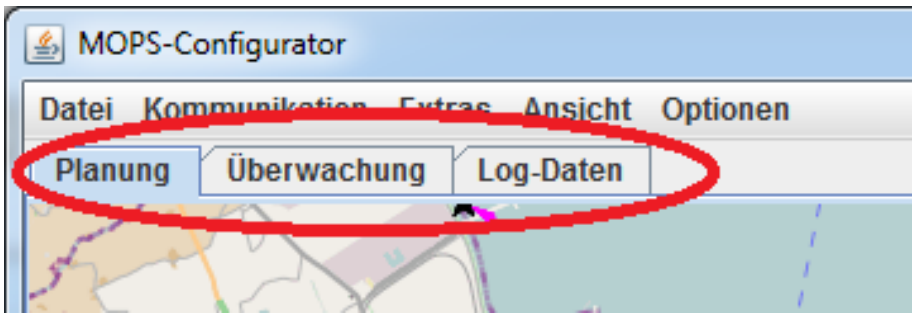


Abbildung 6.2.: Reiter zum Wechseln des Anwendungsbereichs

litäten, wie das Importieren und Exportieren von Missionen, das Senden von Befehlen an den MOPS, sowie das Vornehmen von Einstellungen bereit.

6.1.2.1. Missionsplanung

Über den Reiter *Planung* gelangen Sie in den Anwendungsbereich, der für die Planung von Missionen zur Verfügung steht. Mit Hilfe der Maus kann der Sichtbereich der Karte verändert werden. Wird die linke Maustaste über der Karte gedückt (und gehalten), ist es möglich durch das Bewegen der Maus diesen Bereich zu verschieben. Über das Mausrad kann in die Karte hinein- oder aus der Karte herausgezoomt werden. Durch einen Doppelklick mit der linken Maustaste auf der Karte ist es möglich Wegpunkte innerhalb der Mission zu platzieren. Die Koordinaten der Position, über der sich die Maus aktuell befindet, werden im linken Bereich unterhalb der Karte angezeigt. Nachdem der Doppelklick durchgeführt wurde öffnet sich ein Dialog zum Erstellen/Bearbeiten des jeweiligen Wegpunktes.

Wegpunkt-Dialog

Der Wegpunkt-Dialog gliedert sich in verschiedene Bereiche, über die dem jeweiligen Weg-

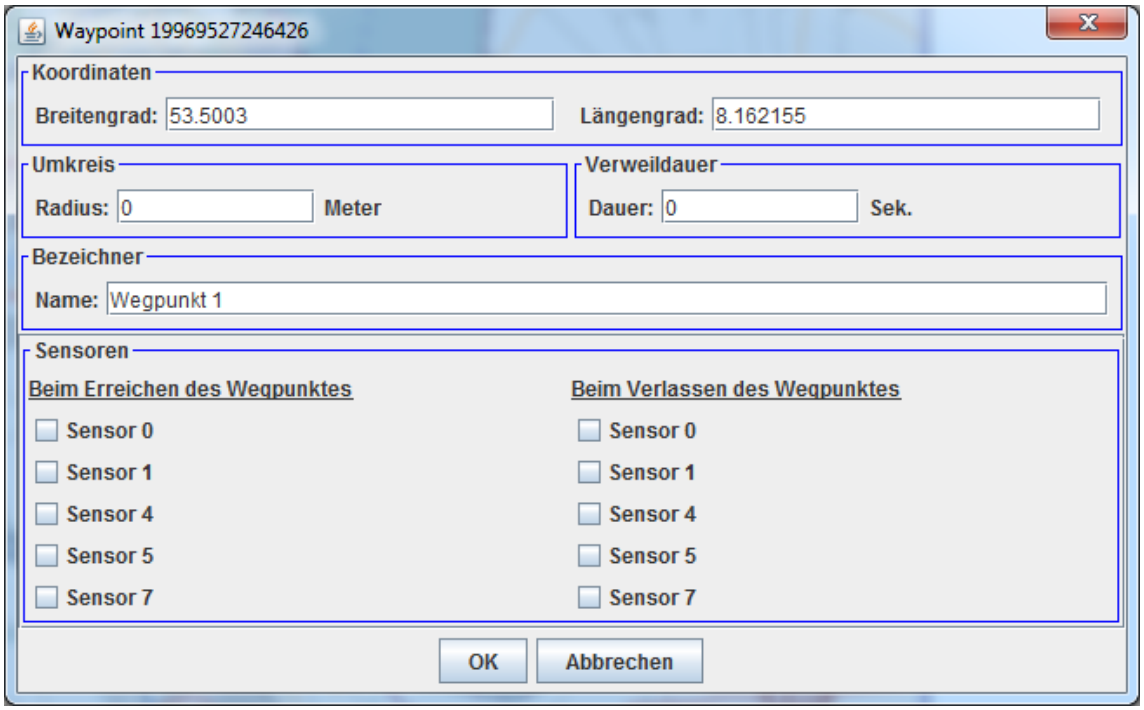


Abbildung 6.3.: Dialog zum Erstellen/Bearbeiten von Wegpunkten

punkt Eigenschaften zugewiesen werden können. Im Bereich *Koordinaten* kann die Position (also Längen- und Breitengrad) des Wegpunktes als Dezimalzahl mit maximal sechs Nachkommastellen definiert werden. Beim Wunsch einer höheren Genauigkeit, können auch mehr als sechs Stellen durch den Benutzer angegeben werden. Diese werden jedoch stan-

dardmäßig durch den MOPS ignoriert. Wurde der Dialog durch einen Doppelklick auf eine Position der Karte der Planungsumgebung geöffnet, werden die Koordinaten dieser Position in den Dialog übernommen. Unter *Umkreis* kann die Genauigkeit in Metern angegeben werden, mit der der Wegpunkt durch den MOPS angefahren werden muss, damit dieser Wegpunkt als „angefahren“ gezählt wird und hiervon abhängende Ereignisse angestoßen werden. Wird in dieses Feld beispielsweise 10 Meter eingegeben, so muss der MOPS diesen Wegpunkt in einem Umkreis von 10 Metern erreichen. Des Weiteren ist unter *Verweildauer* anzugeben, wie lange sich der MOPS, nach erreichen des Punktes, an diesem Punkt stationär aufhalten soll, bevor der nachfolgende Wegpunkt angefahren wird. Der *Name* im Bereich *Bezeichner* stellt die Zeichenkette dar, die zur Identifizierung des Wegpunktes (durch den Benutzer) in der Karte bzw. der Liste der Wegpunkte im Hauptfenster (im Bereich *Planung*) der Anwendung angezeigt wird. Im darunter liegenden Bereich *Sensoren* werden die im Programm registrierten Sensoren angezeigt und können die Zustände der jeweiligen Sensoren beim Eintreffen bzw. Verlassen eines Wegpunktes festgelegt werden. Die Checkboxen stellen somit die Aktivitätsänderungen der Sensoren in den jeweiligen Situationen dar.

Durch einen Klick auf die Schaltfläche *OK* kann der erstellte Wegpunkt in die Planung der aktiven Mission übernommen werden. Alle in der Mission enthaltenen Wegpunkte werden in Form eines blauen „Tropfens“ mit ihren entsprechenden Bezeichnern in der Karte, sowie in der Liste am rechten Fensterrand angezeigt. Durch das Markieren eines Wegpunktes und das Betätigen der entsprechenden Schaltfläche im Bereich *Mission bearbeiten* können in der Mission enthaltene Wegpunkte bearbeitet bzw. entfernt, neue Wegpunkte der Mission hinzugefügt oder die Mission vollständig zurückgesetzt werden. Das Zurücksetzen einer Mission entfernt alle enthaltenen Wegpunkte. Über die Schaltläche *Übertragen* im Bereich *Mission übertragen* kann die aktuell im Programm geladene Mission bei bestehender Verbindung an den MOPS übertragen werden. Tritt bei der Übertragung ein Fehler auf, bspw. da keine Verbindung besteht, wird dies durch eine Fehlermeldung dem Anwender mitgeteilt.

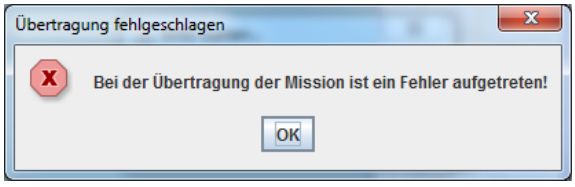


Abbildung 6.4.: Dialog bei Fehlern während der Übertragung

6.1.2.2. Missionsüberwachung

Durch einen Klick auf den Reiter *Überwachung* gelangt man in den Bereich der Anwendung, der zum Überwachen der aktuellen Mission sowie des MOPSes dient. Wie auch in

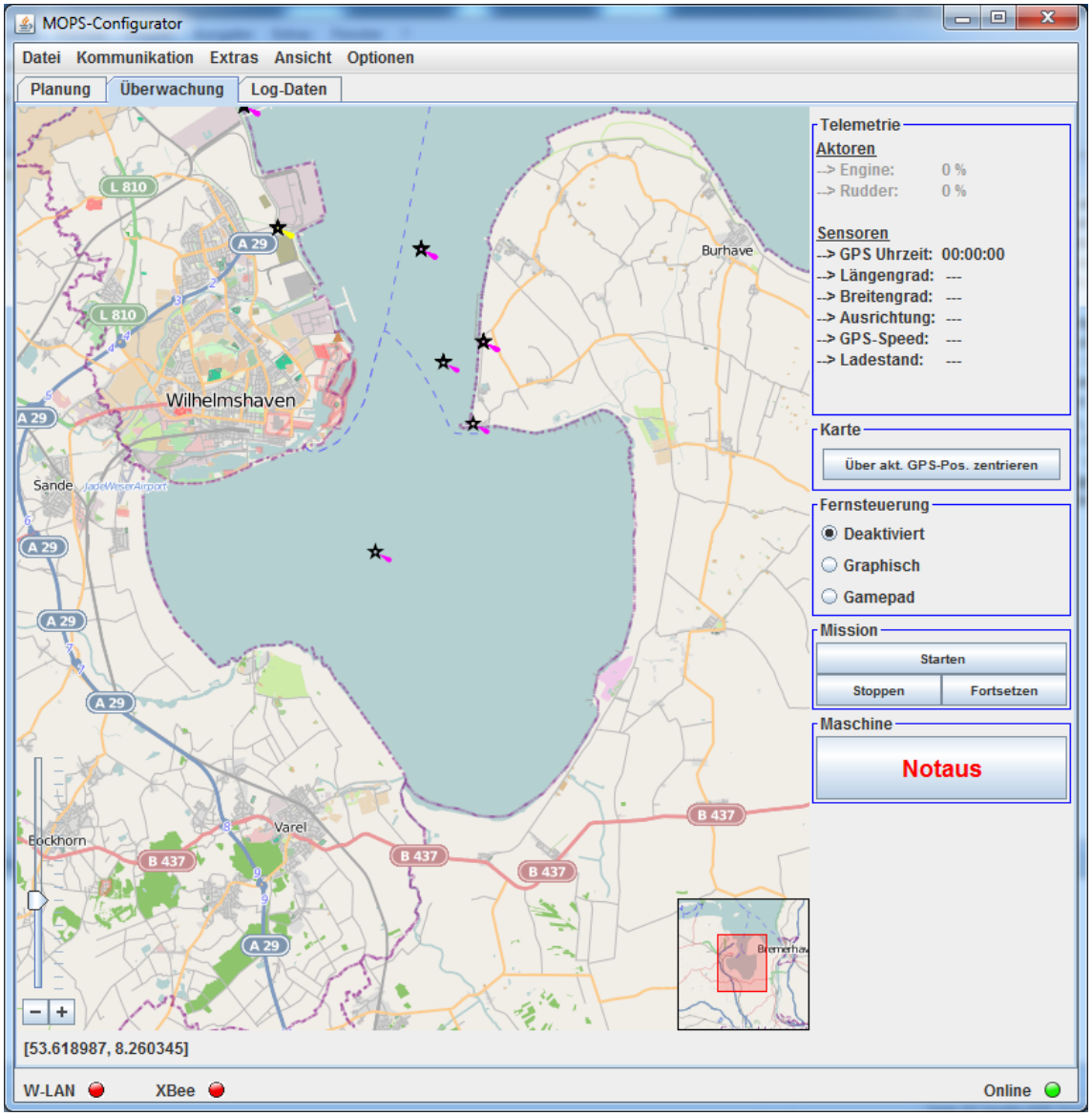


Abbildung 6.5.: Bereich der Anwendung zur Überwachung einer Mission

dem zuvor beschriebenen Bereich *Planung* ist im Bereich *Überwachung* eine Karte vorhanden, auf der die Wegpunkte der aktuell geladenen Mission zu sehen sind. Neben diesen blauen Wegpunkten werden alle GPS-Punkte der empfangenen Heardbeats als rote Wegpunkte dargestellt.

Der rechte Rand des Fensters dient zum Anzeigen der Telemetriedaten aus dem Heartbeat. Telemetriedaten sind u. a. bei aktiver manueller Steuerung die gewünschten Werte der Maschinenleistung und des „Rudereinschlags“, die tatsächliche Position, sowie die Ausrichtung, die GPS-Geschwindigket und den prozentualen Ladezustand der Batterien. Zu beachten ist, dass es sich bei den Werten für Maschinenleistung und „Rudereinschlag“ nicht um die tatsächlich auf dem MOPS anliegenden Werte handeln muss, sondern diese Daten als gewünschte Steuerdaten an den MOPS übertragen werden. Zusätzlich be-

steht die Möglichkeit dem MOPS bei einer bestehenden Verbindung bestimmte Befehle zu übertragen. Zu diesen Befehlen zählen beispielsweise das aktivieren der manuellen Steuerung, das Starten/Stoppen/Fortsetzen einer Mission, sowie das Durchführen eines 'Notaus'. Über die Schaltfläche *Über akt. GPS-Pos. zentrieren* kann der Kartenausschnitt über der Position des zuletzt empfangenen Heartbeats ausgerichtet werden. Durch einen Rechtsklick auf eine Position innerhalb der Karte, und des daraufhin erscheinenden Menüs *Fahre zu Koordinate...* kann veranlasst werden, dass der MOPS zu dieser Position fährt.

Fernsteuerung

Für die Fernsteuerung des MOPS gibt es zwei Möglichkeiten: Zum einen die Steuerung über einen an den Computer angeschlossenen Controller und zum anderen die Möglichkeit über eine graphische Oberfläche. Die Auswahl erfolgt über eine Checkbox auf der rechten Seite (siehe Abb. 6.6).

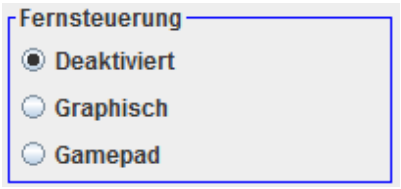


Abbildung 6.6.: Fernsteuerungsauswahl

Wählt man die Option *Graphisch*, so erscheint die graphische Fernsteuerung (siehe Abb. 6.7). Über die beiden Schieberegler können Motor- und Ruderwerte des MOPS geregelt werden. Die Steuerung über einen angeschlossenen Controller erfordert zuvor eine Kalibrierung mithilfe der eingebauten Kalibrierungskomponente.

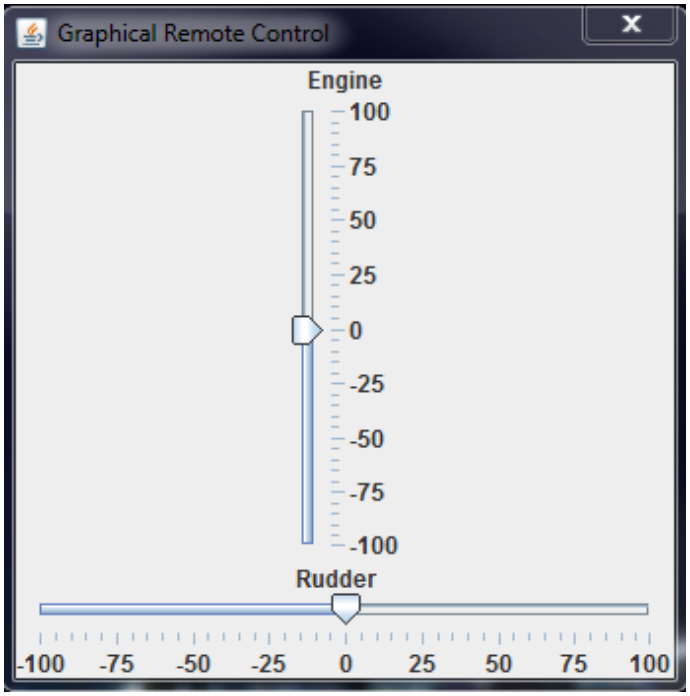


Abbildung 6.7.: Graphical Remote Control

6.1.2.3. MOPS Log-Daten

Wechselt man zum Reiter *Log-Daten*, so gelangt man in den Bereich der Anwendung, in dem man die vom MOPS empfangenen Log-Daten einsehen kann. Dieser untergliedert sich horizontal in zwei Komponenten: den *Mission-Log-Daten* und den *Sensor-Log-Daten*. Die obere Komponente enthält bspw. Daten wie die Heartbeats. Angezeit werden der Zeitpunkt zu dem der Log-Eintrag entstanden ist (*TimeStamp*), die Komponente, die den Entrag verursacht hat (*Caller*), sowie der Inhalt des Log-Eintrags (*Message*). Da der MOPS die eigenen Log-Daten intern speichert und diese, falls dies nicht explizit gewünscht wird, nicht über die bestehenden Kommunikationswege sendet, gibt es die Funktionalität innerhalb der Anwendung existierende Log-Daten anzufordern. Siehe hierzu *Log anfordern* im Abschnitt zum Menü *Kommunikation*.

Die untere Komponente (*Sensor-Log-Daten*) stellt die Daten bereit, die durch am MOPS angeschlossene Sensoren gemessen wurden. Ein Eintrag besteht immer aus dem Zeitpunkt zu dem der Sensorwert ermittelt wurde (*TimeStamp*), der ID des Sensors, der den Wert ermittelt hat (*Sensor_ID*) und dem jeweiligen Wert (*Value*). Wie auch die Daten des Missions-Logs werden die Einträge des Sensor-Logs durch den MOPS intern gespeichert und standardmäßig nicht automatisch über vorhandene Kommunikationswege übertragen. Zum Abrufen der Sensor-Daten siehe *Sensordaten anfordern* im Abschnitt zum Menü *Kommunikation*.
Beide Komponenten verfügen über die Mittel, enthaltene Einträge zu speichern, bereits gespeicherte Einträge wieder zu laden, oder alle Einträge der jeweiligen Komponente zu entfernen. Zum Speichern der Logeinträge wählt man in der jeweiligen Komponente die Schaltfläche *Speichern*, in dem sich öffnenden Dialog den entsprechenden Dateipfad, gibt einen Dateinamen an und bestätigt die Eingaben durch einen Klick auf *Speichern*. Entsprechend ist dies auch zum Laden bereits gespeicherter Einträge möglich, wobei in dem erscheinenden Dialog kein Dateiname angegeben werden muss, sondern die zu ladende Datei direkt ausgewählt werden kann. Um alle Einträge aus einer Komponente zu entfernen, die Liste also zu leeren, wird die Schaltfläche *Reset* bereitgestellt.
Beide Komponenten stellen ebenfalls eine Schaltfläche zur Verfügung die es ermöglicht, dass beim Hinzufügen von Logeinträgen automatisch ans Ende der jeweiligen Auflistung gescrollt wird. Möglich ist dies durch die Aktivierung der Checkbox *Autoscroll*.

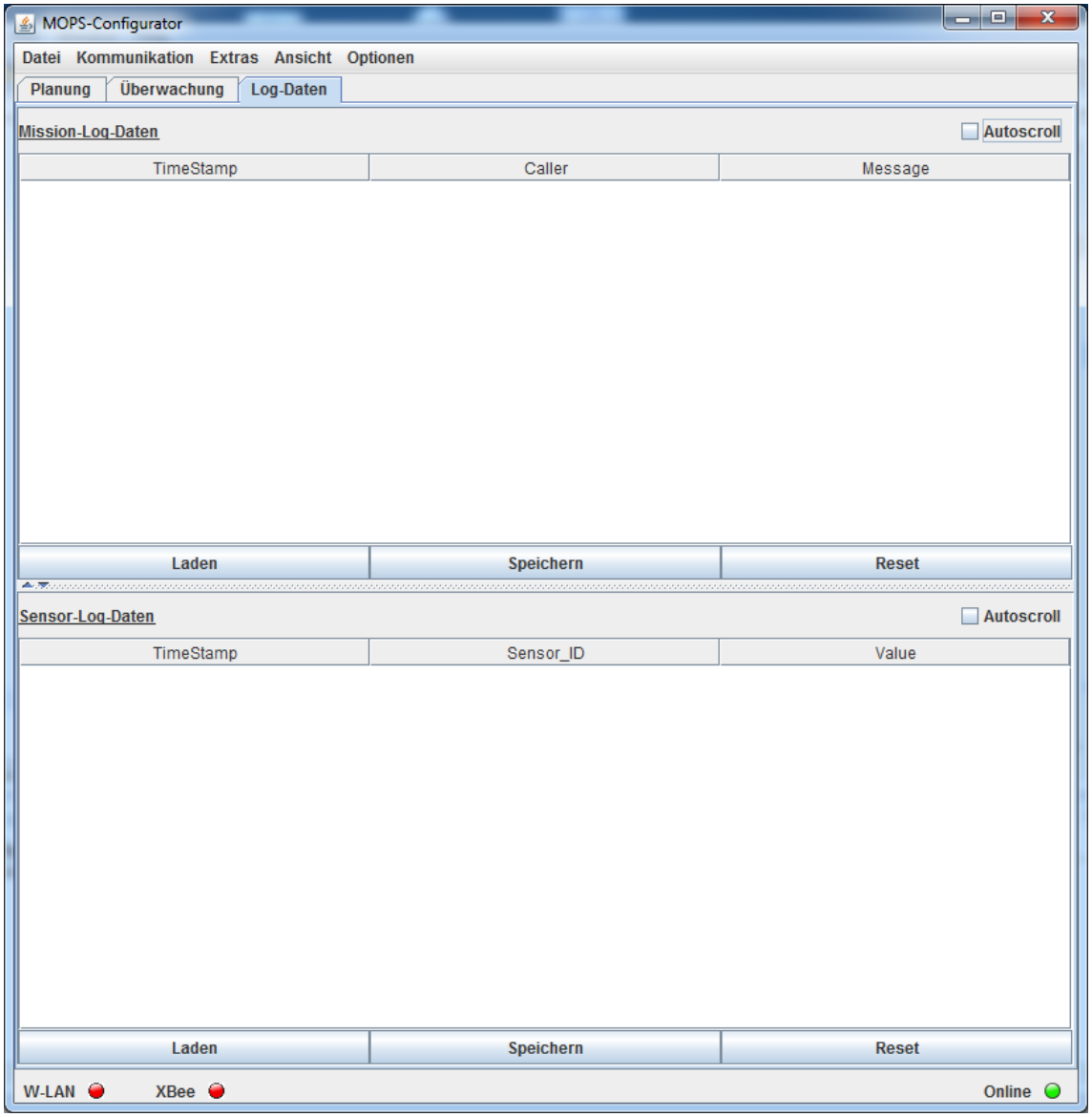


Abbildung 6.8.: Bereich der Anwendung zum Einsehen der MOPS-Logdaten

6.1.2.4. Menüleiste

Im oberen Bereich der Anwendung befindet sich die Menüleiste, über die viele Funktionalitäten und Einstellungen der Software genutzt werden können.

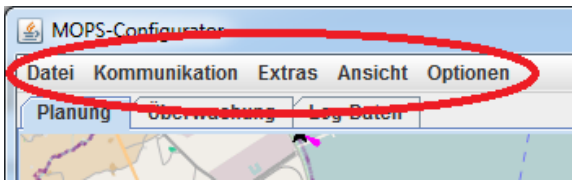


Abbildung 6.9.: Menüleiste der Anwendung

Datei-Menü

Über das Menü *[Datei]* kann eine neue Mission erstellt werden (*[Datei]* → *[Neue Mission]*), eine bereits existierende Mission geladen (*[Datei]* → *[Mission importieren]*) bzw. gespeichert (*[Datei]* → *[Mission exportieren]*) werden. Der MOPS-Verlauf, der sich aus allen Positionen der empfangenen Heartbeats zusammensetzt und in der Karte im Bereich *Überwachung* als rote Markierungen dargestellt wird, kann über die Menüs (*[Datei]* → *[MOPS-Verlauf]* → *[Laden]/[Speichern]*) geladen bzw. gespeichert werden. Der Menüeintrag *[Beenden]* beendet die Anwendung.

Abbildung 6.10.: Menü „Datei“

Kommunikations-Menü

Das Menü *Kommunikation* stellt die Mittel bereit die notwendig sind, um mit dem MOPS interagieren zu können. Über den Menüeintrag *Verbinden* öffnet sich ein Dialog zum herstellen einer bestimmten Verbindung.

Verbinden

Um eine Verbindung herzustellen wählt man im oberen Bereich des Dialogs die Art der Verbindung aus. Neben einer üblichen Verbindung über WLAN kann der Benutzer eine Verbindung über *XBee-Socket* oder *XBee-Serial* herstellen. Grundsätzlich unterscheiden

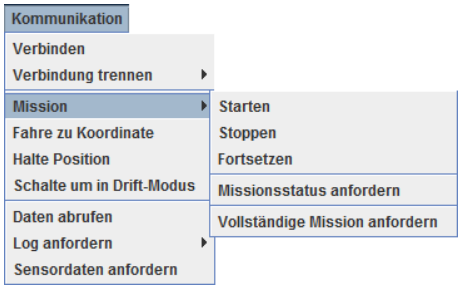


Abbildung 6.11.: Menü „Kommunikation“

sich diese beiden Arten der Verbindung darin, wie das Funkmodul an den Computer, auf dem die Anwendung ausgeführt wird, angeschlossen ist. Bei dem direkten Anschluss (bspw. übe ein USB-Kabel) kann über die Auswahl der Option *XBee-Serial* der Name des seriellen Anschlusses bzw. die gewünschte Übertragungsgeschwindigkeit (*Baud*-Rate) angegeben werden. Wenn im Gegensatz zu der eben beschriebenen Variante das Funkmodul an einen weiteren Computer angeschlossen ist, der sich bspw. in dem Netzwerk befindet wie der Computer, auf dem die Software ausgeführt wird, und der den Zugriff auf das Modul über einen Port ermöglicht, kann durch Auswähl der Option *XBee-Socket* und der Eingabe der *IP* bzw. des jeweiligen *Port*s eine Verbindung aufgebaut werden.

Der Status der Kommunikationsverbindungen wird unten links in der Statusleiste des

Abbildung 6.12.: Verbindung herstellen Dialog

Hauptfensters der Anwendung angezeigt. Insgesamt sind drei Zustände möglich: rot, gelb und grün. Der rote Zustand signalisiert, dass keine Verbindung über den jeweilgen Kommunikationsweg besteht. Bei dem Versuch eines Verbindungsaufbaus, bzw. im Falles des Verlustes einer Verbindung und dem Versuch diese wiederherzustellen, wird dies durch eine gelbe Statuslampe angezeigt. Diese Statuslampe nimmt eine grüne Farbe an, sobald eine Verbindung hergestellt werden konnte, und die Kommunikation über den jeweiligen Weg möglich ist.

Verbindung trennen

Der Menüeintrag *Verbindung trennen* und die enthaltenen Untermenüs bieten die Möglichkeit bestimmte Verbindungen zu trennen, bzw. über *[Verbindung trennen]* → *[Alle trennen]* alle offenen Verbindungen zu schließen.

Mission

Über den Menüeintrag *Mission* kann eine auf dem MOPS übertragene Mission gestartet,

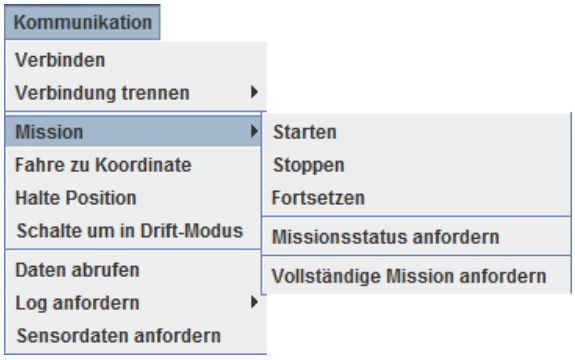


Abbildung 6.13.: Menüeintrag *[Mission]*

gestoppt oder eine gestoppte Mission fortgesetzt werden. Wählt man hierzu den entsprechenden (Unter-)Eintrag, so öffnet sich ein Dialog, über den die Informationen angegeben werden können, um die gewünschte Mission näher zu spezifizieren. Dieser Dialog bietet

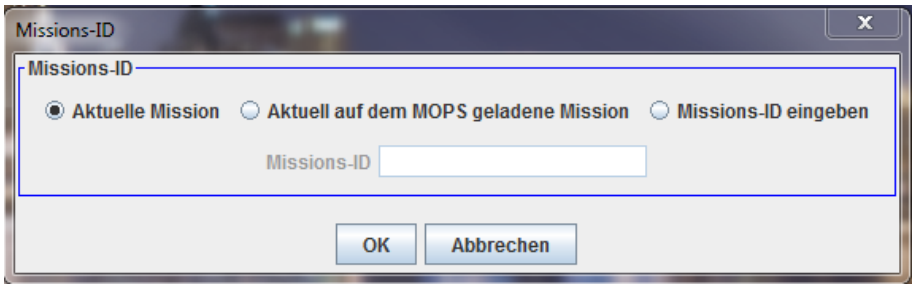


Abbildung 6.14.: Dialog „Missions-ID“

drei grundsätzliche Auswahlmöglichkeiten. Die Option *Aktuelle Mission* stellt die Mission dar, die aktuell in der Anwendung geladen ist. Die zweite Option *Aktuell auf dem MOPS geladene Mission* stellt die Mission dar, die aktuell auf dem MOPS aktiv ist. Falls dem Anwender die Missions-ID der gewünschten Mission bekannt ist, kann dieser diese ID über die dritte Option (*Missions-ID eingeben*) und dem sich aktivierenden Eingabefeld angeben. Durch einen Klick auf *[OK]* wird die Eingabe bestätig. Nachdem der übertragene Befehl durch den MOPS verarbeitet wurde, wird dies durch einen Dialog dem Benutzer mitgeteilt.
Neben den bereits beschriebenen Funktionalitäten kann über den Menüeintrag *[Mission]* der Status der bzw. einer aktuell auf dem MOPS vorhandenen Mission abgefragt und

angezeigt (*[Mission]* → *[Missionsstatus anfordern]*), oder eine auf dem MOPS befindliche Mission abgefragt und geladen werden (*[Mission]* → *[Vollständige Mission anfordern]*). In beiden Fällen wird (wie in den oben beschriebenen Fällen auch) der Dialog zur Eingabe der Missions-ID angezeigt.

Fahre zu Koordinate

Wünscht der Anwender, dass der MOPS zu einer bestimmten Position fährt, ohne eine

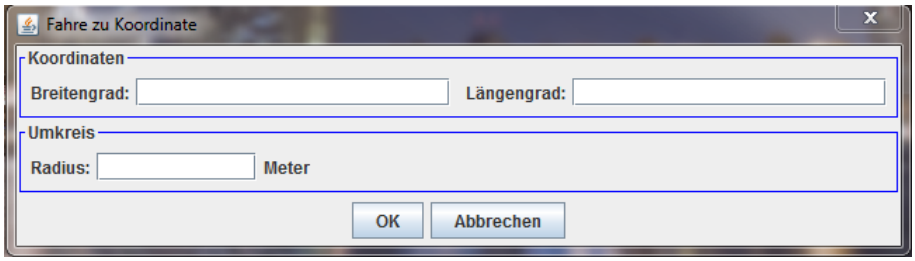


Abbildung 6.15.: Fahre zu Koordinate Dialog

Mission durchzuführen, so kann dies über das Menü *[Kommunikation]* → *[Fahre zu Koordinate]* oder einen Rechtsklick auf die Karte im Bereich *Überwachung* realisiert werden. In beiden Fällen öffnet sich ein Dialog, über den die Angabe der Koordinaten des jeweiligen Wegpunktes, sowie der Radius der Genauigkeit angegeben werden kann, mit der der Wegpunkt angefahren werden muss. Durch die Bestätigung mit *[OK]* wird der Befehl an den MOPS übertragen und nach Verarbeitung durch eine Mitteilung bestätigt.

Halte Position

Durch den Befehl *Halte Position* kann dem MOPS mitgeteilt werden, dass dieser seine aktuelle Position halten soll. Die dafür notwendigen Berechnungen übernimmt der MOPS eigenständig.

Schalte um in Drift-Modus

Über den Menüeintrag *[Schalte um in Drift-Modus]* kann der Antriebsmodus des Wasserfahrzeugs gewechselt werden.

Daten abrufen

Dieser Menüeintrag stellt die Möglichkeit bereit alle auf dem MOPS zu einer bestimmten Mission verfügbaren Daten abzurufen. Nach Auswahl des Eintrags öffnet sich der Dialog zur Spezifikation der gewünschten Mission.

Log anfordern

Das „Log“ bezeichnet sämtliche Logeinträge, die auf dem MOPS aufgezeichnet wurden. Das „Log“ kann durch die Software auf unterschiedliche Weise angefordert werden. Durch

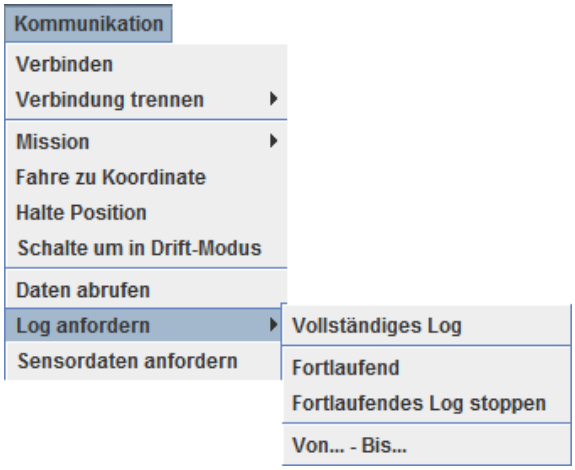


Abbildung 6.16.: (Sub-)Menü „Log anfordern“

Auswahl der ersten Möglichkeit (*[Vollständiges Log]*) wird der MOPS dazu veranlasst, alle Log-Einträge über den jeweiligen Kommunikationsweg zu übertragen. Durch die Menge der Daten sollte dies jedoch nur bei einer WLAN-Verbindung angestoßen werden. Um einen bestimmten Bereich im Hinblick auf das zeitliche Auftreten des Log-Eintrags bzw. die Anzahl der gewünschten Einträge die empfangen werden sollen auszuwählen, kann der Menüeintrag *[Von... - Bis...]* ausgewählt werden, wodurch sich ein Dialog zum Auswählen des jeweiligen Bereiches öffnet. Im oberen Bereich des Dialogs kann gewählt werden, ob

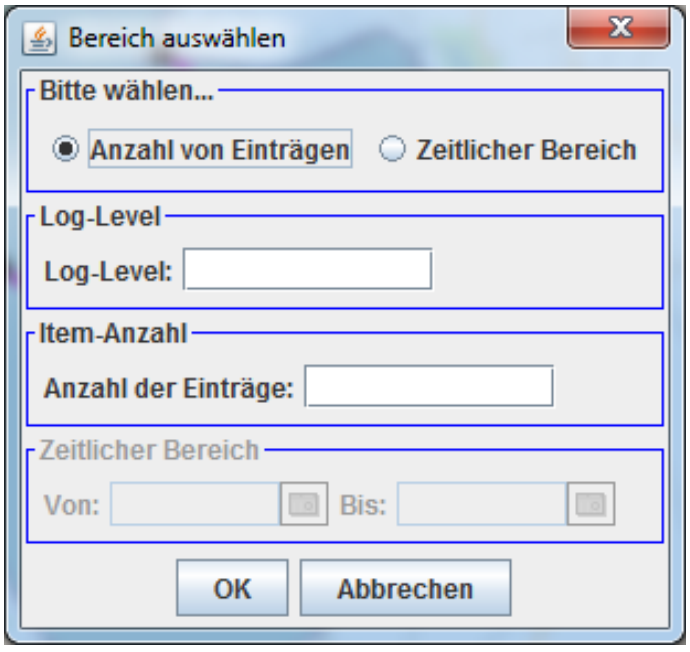


Abbildung 6.17.: Dialog zum Auswählen eines Log-Bereiches

eine bestimmte Anzahl von Einträgen oder ein zeitlicher Bereich empfangen werden soll. Wurde erstere Option gewählt, kann die gewünschte Anzahl im Bereich *Item-Anzahl* ein-

gegeben werden. Wurde die Option *Zeitlicher Bereich* ausgewählt, kann der Zeitraum in dem aktivierten Bereich *Zeitlicher Bereich* im unteren Teil des Dialogs definiert werden. Über den Bereich *Log-Level* kann definiert werden, ab welchem Log-Level die Log-Einträge übertragen werden sollen. Durch das Bestätigen des Dialogs mit *[OK]* wird der Befehl an den MOPS übertragen und der Dialog geschlossen.

Sensordaten anfordern

Nach einem Klick auf den Eintrag *[Sensordaten anfordern]* öffnet sich ein Dialog, der im

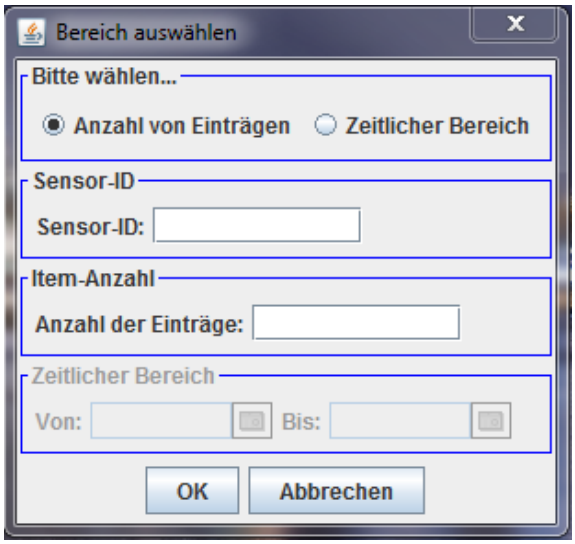


Abbildung 6.18.: Dialog zum Auswählen eines Bereiches

Aussehen und der Funktionalität sehr stark dem Dialog zum Anfordern von bestimmten Log-Einträgen ähnelt. Der Unterschied besteht in der Möglichkeit bei diesem Dialog die ID des jeweiligen Sensors, anstelle des Log-Levels anzugeben. Es werden lediglich die Sensordaten beachtet, bei denen die Sensor-ID mit der angegebenen ID übereinstimmt. Die empfangenen Sensordaten werden im Bereich *Log-Daten* der Anwendung aufgeführt.

Extras-Menü

Über das Menü *[Extras]* können die zusätzlichen Optionen in der Software ausgewählt werden. So können mit dem Tile-Updater, die lokal auf der Festplatte gespeicherten Kartenausschnitte auf die aktuelle Version gebracht werden (*[Ansicht]* → *[Tile-Update]*). Abbildung 6.20 zeigt den Tile-Updater. Der Update-Vorgang kann gestartet und abgebro-

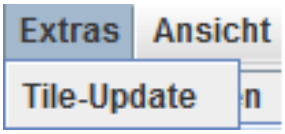


Abbildung 6.19.: Menü „Extras“

chen werden, und über die Fortschrittsanzeige wird der aktuelle Stand angezeigt.

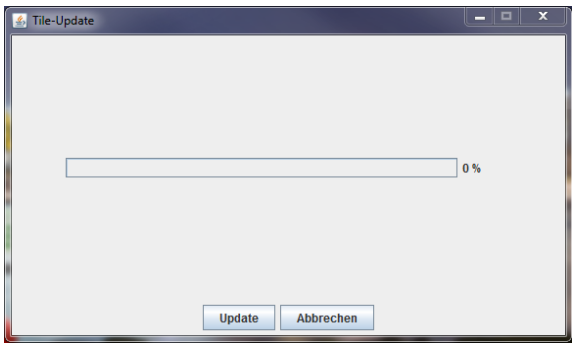


Abbildung 6.20.: Menü „Tile-Updater“

Ansicht-Menü

Über das Menü *[Ansicht]* können die Einstellungen für die Ansicht auf der Karte verändert werden. Über den Eintrag (*[Ansicht]* → *[Route zeichnen]*), können die einzelnen Wegpunkte mit einer Linie verbunden werden. Im Untermenü *[MOPS Verlauf]* können Einstellungen zum Verlauf der Mission verändert werden. So kann der gesamte Verlauf angezeigt werden (*[Ansicht]* → *[Gesamten Verlauf anzeigen]*), die GPS Punkte angezeigt(*[Ansicht]* → *[GPS-Punkt-Symbole anzeigen]*), sowie verbunden (*[Ansicht]* → *[GPS-Punkte verbinden]*) und ebenfalls gelöscht werden (*[Ansicht]* → *[Verlauf löschen]*).

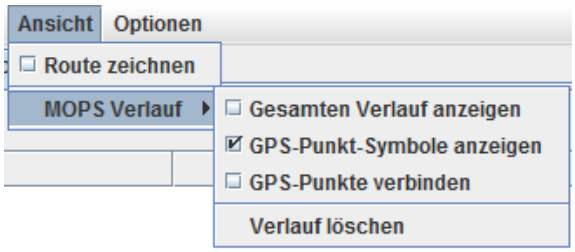


Abbildung 6.21.: Menü „Ansicht“

Optionen-Menü

Über das Menü *[Optionen]* kann die Sensor-Verwaltung oder (*[Optionen]* → *[Sensor-Verwaltung]*) die Gamepad-Kalibrierung gestartet werden (*[Optionen]* → *[Gamepad-Kalibrierung]*). Ebenfalls kann bei aktivierter XBee Verbindung die Sendeleistung des XBee eingestellt (*[Optionen]* → *[XBee-Sendeleistung]*) und der Online-Status der Software verändert werden (*[Optionen]* → *[Online-Modus]* bzw. *[Optionen]* → *[Offline-Modus]*)

Sensor-Verwaltung

Über die Sensor-Verwaltung(siehe Abb. 6.23 können die im System registrierten Sensoren

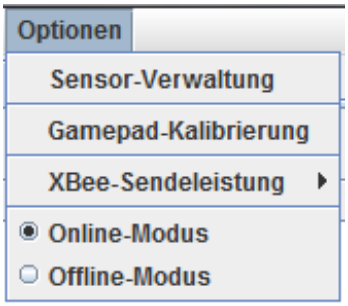


Abbildung 6.22.: Menü „Optionen“

bearbeitet, sowie Sensoren hinzugefügt oder gelöscht werden. Jeder Sensor verfügt dabei über eine ID und einen Namen.

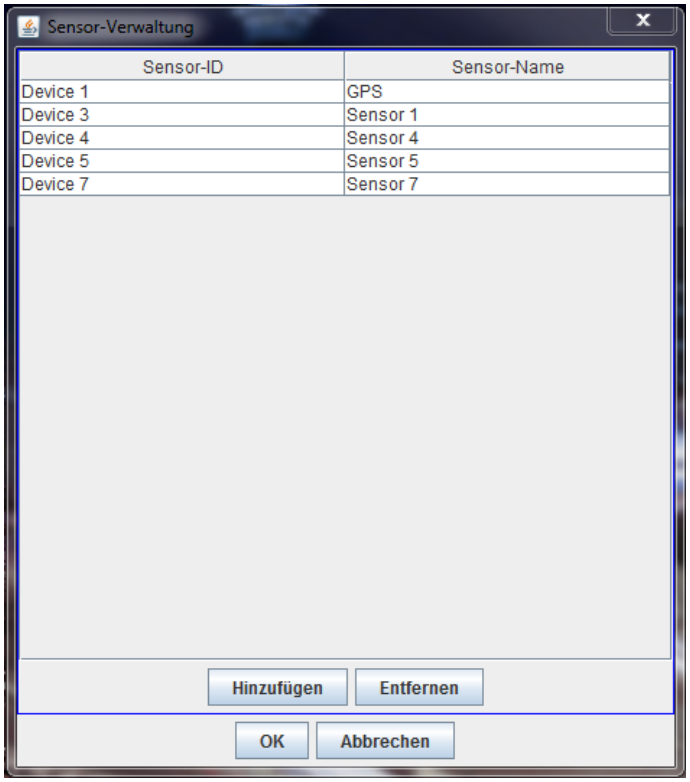


Abbildung 6.23.: Sensor Verwaltung

6.1.2.5. Statusleiste

Die Statusleiste gibt Informationen über den aktuellen Zustand der Kommunikationskanäle WLAN und Xbee aus. Eine rote Anzeige bedeutet, dass zur Zeit keine Kommunikationsverbindung besteht, eine gelbe Anzeige bedeutet, dass versucht wird eine Verbindung aufzubauen und eine grüne Anzeige bedeutet, dass eine Verbindung besteht. Auf der rechten Seite wird zudem der aktuelle Internet-Status der Anwendung angezeigt. Rot bedeutet die Software ist offline und es werden keine Karten aus dem Internet geladen (bei mobilen Verbindungen mit Datenbegrenzung empfehlenswert) und eine grüne Anzeige bedeutet, dass aktuelle Karten aus dem Web geladen werden.

Abbildung 6.24.: Statusleiste

6.2. Inbetriebnahme

Der nachfolgende Abschnitt widmet sich dem Zusammenbau und der Inbetriebnahme des MOPS. Hierbei wird eine Bauanleitung aufgezeigt, sowie auf die notwendigen Schritte eingegangen, die vor einer Mission des MOPS durchzuführen sind. Der MOPS besteht vor dem Zusammenbau aus den folgenden Komponenten:

- Linker Schwimmer
- Rechter Schwimmer
- Mittelrumpf
- Motorenbefestigung (hintere Querverbindung der Schwimmer)
- Vordere Querverbindung der Schwimmer
- 4 Querstreben für die Stabilisierung der Ecken
- Batteriekasten
- 2 Außenbordmotoren *Rhino-VX-34* inkl. Schrauben
- Spanngurt zur Befestigung des Batteriekastens
- Antenne

Als Werkzeug wird lediglich der Kugelkopf-Schraubendreher SW-4 [13] von ITEM benötigt (zweimal vorhanden).

1. Schritt: Richtiges Anordnen der Bestandteile Zuerst sollten die einzelnen Bestandteile des MOPS in richtiger Anordnung auf dem Boden ausgebreitet werden. Der Schwimmer mit dem Aufkleber L wird links und der Schwimmkörper mit der Aufschrift R rechts

vom Mittelrumpf angeordnet. Die Motorenbefestigung, die als hintere Querverbindung der beiden Schwimmer fungiert, wird am hinteren Ende des Mittelrumpfs und die vordere Querverbindung am vorderen Teil angeordnet.

2. Schritt: Zusammensetzen des äußeren Rahmens Die benötigten Nutensteine sind bereits an den richtigen Positionen in der Nut des Alurahmens angebracht. Die größeren Eckwinkel (60 mm x 60 mm) werden mit je vier Nutensteinen und vier 16 mm langen Schrauben inkl. Unterlegscheibe fixiert. Die kleineren Winkel (30 mm x 30 mm) an den Querverbindungen werden mit zwei Nutensteinen und zwei 16 mm langen Schrauben fixiert. Es ist empfehlenswert zuerst den Mittelrumpf mit der vorderen und der hinteren Querverbindung (Motorenbefestigung) über den kleineren 30 mm x 30 mm Winkelsatz zu verbinden. Im nächsten Schritt erfolgt die Montage der äußeren Schwimmer. Hierbei werden die 60 mm x 60 mm Winkelsätze benutzt. Diese sind ebenfalls so weit vorbereitet, sodass lediglich jeweils zwei Schrauben in die schon eingesetzten Nutensteine geschraubt werden müssen. Sind die Nutensteine leicht verrutscht (z. B. durch den Transport), so lassen sich diese leicht mit einem Schraubenzieher in die richtige Position verschieben.

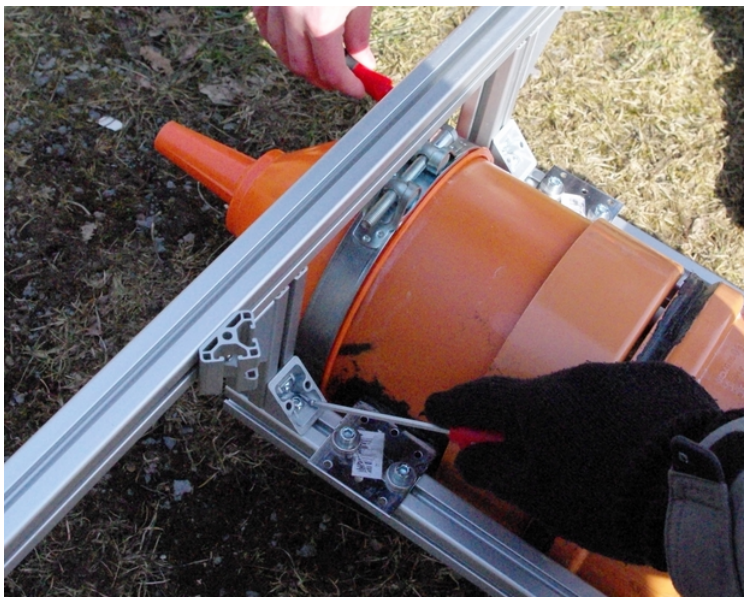


Abbildung 6.25.: Montage der vorderen Querverbindung an den Mittelrumpf

3. Schritt: Befestigung der Querstreben zur Stabilisierung in den Ecken Am Rahmen des MOPS sind bereits die T2-Elemente befestigt [11], welche zur Realisierung eines 45-Grad-Winkels benötigt werden. An diese T2-Elemente werden mithilfe von Universalverbindern [12] die vier 1050 mm langen Profilleisten fixiert. Die Best-Practice-Methode dazu ist, ein Ende der Querstreben an einem T2-Element zu fixieren, und am anderen Ende das T2-Element leicht zu lösen, um dieses in den richtigen Winkel zur Querverstrebung verschieben zu können. Sind alle vier Querstreben installiert, ist der Rahmen des MOPS

bereits vollständig montiert.

Abbildung 6.26.: Montage der Querstreben

4. Schritt: Anbringen der Motoren Die beiden Motoren des Typs *Rhino-VX-34* [20] werden mit der Schraubzwingenbefestigung an der Motorenaufhängung angebracht. Die Kabel der Motoren sollten um den Rahmen gewickelt zum Batteriekasten geführt werden. Zusätzlich können die Kabel mit Kabelbindern fixiert werden. Es ist darauf zu achten, dass die Propeller der Motoren fest angezogen sind, da diese sich sonst während einer längeren Rückwärts-Fahrt lösen können und verloren gehen.

5. Schritt: Befestigung des Batteriekastens Der Batteriekasten sollte zuerst auf die vorgesehene Plattform im Mittelrumpf gesetzt werden. Nun können die Kabel der beiden Motoren durch die spritzwassergeschützten Kabeldurchführungen im Batteriekasten geführt und am Motorcontroller, der sich ebenfalls im Inneren befindet, angeschlossen werden. Der korrekte Anschluss ist erfolgt, wenn an beiden äußeren Klemmen rote und an allen anderen Klemmen schwarze Kabel angebracht sind. Nach ordnungsgemäßem Anschluss kann der Batteriedeckel aufgesetzt und verschlossen werden. Anschließend wird der Batteriekasten mit einem Spanngurt auf der Batterieplattform fixiert.

6. Schritt: Befestigung der XBee-Antenne Die circa 2 m lange Antennenkonstruktion wird mit zwei Nutensteinen an den Rahmen beim Steuerungssegment fixiert. Durch die wasserdichte Durchführung wir die Antenne im Steuerungssegment an das innen liegende XBee-Modul angeschlossen. Bei stärkeren Windverhältnissen sollte ein zusätzliches Abspannen der Antenne zur Sicherung vorgenommen werden.

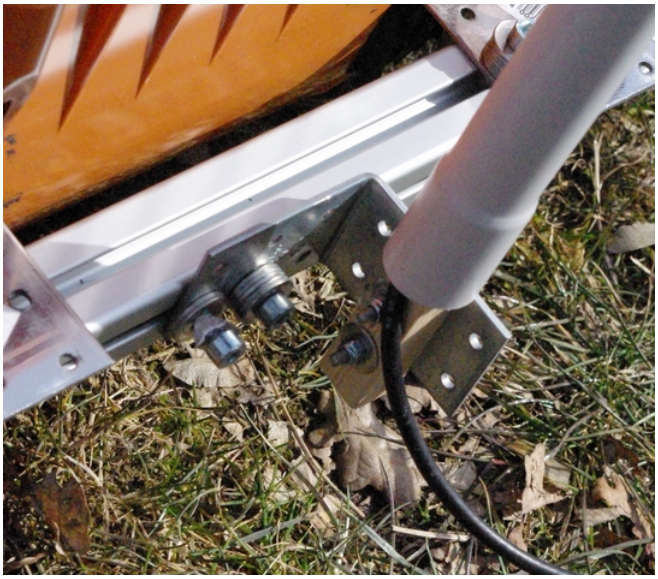


Abbildung 6.27.: Montage der Antenne

7. Schritt: Anschluss des Steuerungssegments zur Energieversorgung Über die wassergeschützte Steckverbindung erfolgt der Anschluss des Steuersegments an das Batteriesegment. Die Hardware wird eigenständig initialisiert, das System ist nach etwa 10 Sekunden einsatzbereit.

7. Fazit

Aus der einjährigen Projektgruppe „Marine Observation Platform for Surfaces“ ist der Prototyp eines autonom fahrenden Wasserfahrzeuges hervorgegangen, welches zur Beobachtung von Marineökosystemen genutzt werden kann. Der MOPS wurde als Low-Cost-Alternative zu Forschungsschiffen, Bojen und bereits im Handel erhältlichen ähnlichen Produkten entwickelt, dessen Materialkosten im Bereich 1000 € bis 2000 € liegen.
Die Projektgruppe hat das Wasserfahrzeug in den vergangenen Monaten von Grund auf entwickelt. Dabei wurde mit Hilfe des Auftraggebers, Herrn Prof. Ziellinski, ein Anforderungskatalog erstellt. Zudem wurden in einer Vorarbeitsphase für das Projekt relevante Informationen, wie z. B. über Bootsbau-, Antriebs-, Kommunikations- und Fertigungstechniken zusammengetragen. Aus der Evaluation verschiedener Konzeptalternativen ging der Trimaran hervor, welcher im Verlauf der Entwicklungszeit in mehreren Phasen entwickelt wurde. Markant beim Konzept des MOPS ist der Verbau von in üblichen Baumärkten verfügbaren Standardteilen. Dies sorgt für eine hohe Verfügbarkeit von Ersatz- und Nachbauteilen und hält die Materialkosten niedrig. Insbesondere die Nutzung von KG-Rohren zur Unterbringung von Computertechnik und Energiespeichern, sowie als Schwimmer, haben sich während der gesamten Entwicklungszeit bewährt. Zur praktischen Prüfung des Konzepts wurde ein Holzrahmen gefertigt, welcher später durch einen Aluminiumrahmen ersetzt wurde. Des Weiteren hat sich während der durchgeführten Testfahrten herausgestellt, dass das einst vorgesehene Antriebskonzept für unsere Anforderungen ungeeignet ist. Daher wurde auch der Antrieb in einem weiteren Schritt mit Hilfe eines neuen Antriebskonzepts realisiert.
Parallel zur Entwicklung der Hardware des Wasserfahrzeugs wurden auch Computerprogramme sowohl zur Steuerung des Wasserfahrzeuges als auch für die autonome Regelung der Hardware selbst auf dem Gerät entwickelt. MOPS kann wahlweise autonom Punkte abfahren oder direkt ferngesteuert werden. Die Kommunikation mit einem Steuerungsterminal (z. B. ein handelsüblicher Laptop-Computer) erfolgt via WLAN oder XBee.

8. Ausblick

Da einige der anfangs aufgestellten Anforderungen zum jetzigen Stand der Dinge nicht erfüllt werden, sind einige Weiterentwicklungen denkbar. Dies kann beispielsweise im Rahmen einer weiteren studentischen Projektgruppe erfolgen.

Die Einsatzzeit des Wasserfahrzeuges kann optimiert werden. Bei der derzeitigen Umsetzung der Energieversorgung ermöglicht der MOPS eine Einsatzzeit von ca. zwei bis fünf Stunden. Das Hantieren mit Hochleistungsakkumulatoren (z. B. Lithium Polymer) ist relativ gefährlich und teuer. Denkbar ist zudem eine Energiegewinnung während des Betriebs, z. B. mit Hilfe von Sonnenenergie. Außerdem könnte der Antrieb durch Windsegel unterstützt werden, um der gewünschten Einsatzzeit von acht bis zwölf Stunden näher zu kommen.

Beim MOPS ist zwar ein Sensorkonzept vorgesehen, realisiert wurde es allerdings bis dato nicht. Zur Installation der Sensorik müssen Löcher in den entsprechenden Bauteilen gebohrt und abgedichtet werden. Ggf. ist ein Steckerkonzept zu überlegen, um die Nutzung und den Umbau durch den Endbenutzer möglichst einfach und bequem zu halten. Ebenfalls muss die Software um die Ansteuerung und Protokollierung der Sensorik erweitert werden.

Das autonome Wasserfahrzeug kann zwar nach GPS-Punkten eigenständig fahren, erkennt jedoch Hindernisse wie Bojen oder andere Verkehrsteilnehmer im Wasser nicht. Auch hier sind Optimierungen im Bereich der Software und im Bereich der Hardware (z. B. durch Erweiterung mit Hilfe von Objekterkennungssensoren) möglich.

MOPS soll Hochseetauglich sein. Dies konnte bislang nicht getestet werden. Eine genaue Überprüfung und ggf. ein Test in einer Simulationsumgebung oder auf See können Klarheit für weitere Umbau- oder Dimensionierungsmaßnahmen am Fahrzeug schaffen.

Während der gesamten Entwicklung sollte stets weiterhin in regelmäßigen Abständen mit dem Auftraggeber Rücksprache gehalten werden, um den Fokus der Eigenschaften mit Priorität nicht zu verlieren.

Linkliste

[1] *Ångström Distribution website*. URL: http://www.angstrom-distribution.org.

[2] *Beschreibung von NMEA Datensätzen*. URL: http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm.

[3] Cameon. *BeagleBone: various useful snippets and information*. URL: http://beaglebone.cameon.net/home/pin-muxing.

[4] CircuitCo. *BeagleBone website*. URL: http://www.beagleboard.org/bone.

[5] DimensionEngineering. *DEScribe Software*. URL: http://www.dimensionengineering.com/software/setupST.exe.

[6] DimensionEngineering. *Sabertooth 2x60 Motorregler*. URL: http://www.dimensionengineering.com/products/sabertooth2x60.

[7] EvoLogics. *Sonobot Autonomous Hydrographic Survey Vehicle*. URL: http://www.evologics.de/en/products/sonobot/index.html.

[8] Inc. Falmouth Scientific. *SAUV II - Solar-powered Autonomous Underwater Vehicle*. URL: http://www.falmouth.com/systems/solarpowerdauv.html.

[9] freedesktop.org. *systemd website*. URL: http://www.freedesktop.org/wiki/Software/systemd.

[10] *ITEM: Industrietechnik GmbH*. URL: http://www.item24.de/.

[11] *ITEM: T2-Winkel-Elemente*. URL: http://www.item24.de/produkte/produktkatalog/produktdetails/products/winkelelemente/winkelelement-6-t2-30-natur-45972.html.

[12] *ITEM: Universalverbindungssatz*. URL: http://www.item24.de/produkte/produktkatalog/produktdetails/products/universal-verbindungssaetze/universal-verbindungssatz-6-rostfrei-44174.html.

[13] *ITEM: Werkzeug*. URL: http://www.item24.de/produkte/produktkatalog/produktdetails/products/schraubendreher/kugelkopf-schraubendreher-sw4-40660.html.

[14] *LOGBack*. URL: http://logback.qos.ch/.

[15] MOPS. *Planung der High Level Kommunikation*. URL: https://mops.informatik.uni-oldenburg.de/wiki/intern:kommunikation%5C_mops%5C_01.10.2012.

[16] NXP. *LPC11C24 Cortex-M0 Mikrocontroller*. URL: http://www.nxp.com/products/microcontrollers/cortex_m0_m0/LPC11C24FBD48.html.

[17] Sascha Spors Peter Bartz. *AHRS Firmware*. URL: https://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs.

[18] *Raspberry Pi website*. URL: http://www.raspberrypi.org/.

[19] *Regelungstechnik*. URL: http://www.rn-wissen.de/index.php/Regelungstechnik.

[20] *Rhino VX 34: Außenbordmotor*. URL: http://www.bootsmotoren4you.de/Rhino-VX-34.

[21] *RXTX Library*. URL: http://rxtx.qbang.org/.

[22] *Sikaflex: Maritime Dichtmasse*. URL: http://www.24trade.de/reparatur/dichtungsmasse/sika-sikaflex-291-kartusche-300ml-haftstarker-marine-dichtstoff.

[23] *Sqlite*. URL: http://www.sqlite.org.

[24] *Techbörse: KG-Rohr DN 200 mit Steckmuffe*. URL: http://www.techboerse.de/garten-landschaftsbau/kg-rohre-mit-steckmuffe-dn-200-kgem.html.

[25] *Techbörse: KG-Rohr Endverschlusskappe*. URL: http://www.techboerse.de/garten-landschaftsbau/endverschluss-muffenstopfen-kgm.html.

[26] *Techbörse: Wartungsrohr*. URL: http://www.techboerse.de/garten-landschaftsbau/reinigungsrohr-kgre.html.

[27] Kimmo Tuukkanen. *Java Marine Api Version 0.5*. URL: http://marineapi.sourceforge.net/.

[28] u-blox. *u-center Software*. URL: http://www.u-blox.com/en/evaluation-tools-a-software/u-center/u-center.html.

[29] Chris Veness. *Calculate distance, bearing and more between Latitude/Longitude point*. URL: http://www.movable-type.co.uk/scripts/latlong.html.

[30] Watterott. *9-DoF-Razor-IMU Bestellseite*. URL: http://www.watterott.com/de/9-DoF-Razor-IMU.

A. Anhang

A.1. Hardwarezusammensetzung

Stückzahl	Bezeichnung	Anmerkung
Bestandteile der Schwimmer, Segmente und Bug		
3	KG Reinigungsrohr mit Klappe	Wartungsrohre dienen als einzelne abgeschlossene Segmente
2	KG-Rohr 2,00 m	Schwimmer in der Länge2,00 m und einem Durchmesser von 200 mm
1	KG Rohre mit Steckmuffe DN 200 Länge 0,50 m	dient als Bugelementhalterung
6	KG Endverschluss / Muffenstopfen	Verschlussstopfen für Segmente und Schwimmer
6	KG Endkappen	Endkappen für Segmente und Schwimmer
1	Styropor-Füllmaterial	Styroporfüllmaterial zum Füllen des Inhalts der 2,00 m KG-Rohre (Schwimmer)
3	Sika Sikaflex 291 Kartusche 300 ml- haftstarker Marine-Dichtstoff	Sikaflex zum Abdichten der Endkappen und Muffenstopfen, sowie Bohrlöcher. Vorbehandelung mit Sikaflex Primer und Sikaflex Aktivator essentiell, um Dichtigkeit zu gewährleisten
1	2 Komponenten PU-Schaum (Schwimmfähig, Einsatz für Bojen)	zum Ausschäumen der Trichter für den Bug
5	Trichter „PRESSOL“-Polyethylen - 2,9 l Durchmesser 200 mm	Zur Realisierung eines Bugs
5	Gelenkbolzenschellen edelstahl	Befestigungsschelle für die Trichter an die KG Elemente

8	Gewindestangen m 8 in der Länge von 50 cm	Gewindestangen zur Befestigung der Schwimmer
14	Edelstahlwinkel	zur Befestigung der Segmente und des Bugsegments an den inneren Rahmen

Stückzahl	**Bezeichnung**	**Anmerkung**
Rahmenbestandteile		
3	Profil 6 30 mm leicht, natur 2000 mm	2 Profile für hinteres Ende (Doppelung wegen Motorenaufhängung) und 1 Profil für den vorderen Bereich
2	Profil 6 30 mm leicht, natur 2200 mm	Profile für den Segmentrahmen
2	Profil 6 30 mm leicht, natur 2140 mm	Profile für die Schwimmerbefestigung
4	Profil 6 30 mm leicht, natur 115 mm	Verbindungsprofile zwischen Außenrahmen und dem Segmentrahmen
4	Winkelsatz 30 mm	größerer Winkelsatz zur Verbindung, bestehend aus Winkel, vier Nutensteinen und vier 16 mm Schrauben
8	Winkelsatz 30 mm	kleinerer Winkelsatz zur Verbindung, bestehend aus Winkel, zwei Nutensteinen und zwei 16 mm Schrauben
8	Abdeckklappen 30 mm	Abdeckkappen für Profilenden
30	Nutenstein 6 m6	Nuttensteine zur Verbindung der Segmente an den Rahmen
8	Winkel T2 45 grad	45 Grad Winkel für stabilisierende Querstreben
4	Profil 6 30 mm leicht, natur 1050 mm	stabilisierende Querstreben
4	Halbrundschraube m6 x 16 mm	Halbrundschrauben für T2 Elemente
4	Schlossschrauben m 6 65 mm lang	dienen der Befestigung der beiden übereinanderliegenden Aluprofile
8	Edelstahlscheiben m 6	dienen der Befestigung der beiden übereinanderliegenden Aluprofile
4	Edelstahlmuttern m 6	dienen der Befestigung der beiden übereinanderliegenden Aluprofile

Stückzahl	Bezeichnung	Anmerkung
Motorenbefestigung		
4	Siebdruckplatten 9 mm stark 100 cm x 9 cm	wasserfest-verleimtes Holz, welches eine abrutsch-sichere Fläche zur Befestigung der Motoren bereitstellt
8	Schlossschrauben m 6 50 mm lang	dienen der Befestigung des Holz an die Aluprofile des Rahmens
16	Edelstahlscheiben m 6	dienen der Befestigung des Holz an die Aluprofile des Rahmens
8	Edelstahlmuttern m 6	dienen der Befestigung des Holz an die Aluprofile des Rahmens
Akkusegment		
1	Wasserdichter belüfteter Batteriekasten	da der genutzte Akku von den Abmessungen zu groß für das vorgesehene Segment ist, wird ein Batteriekasten genutzt, der auf einer Holzplattform an den Rahmen befestigt wird
1	Akku 108 aH	dieser Akku wurde freundlicherweise von Prof. Hahn zu Testzwecken bereitgestellt
1	Siebdruckplatte 12 mm 30 cm x 40 cm	wasserfest-verleimtes Holz als Plattform für den Batteriekasten
4	Inbusschrauben 16 mm	dienen zur Befestigung der Plattform an den Rahmen des Mittelrumpfs
4	Edelstahlscheiben	dienen zur Befestigung der Plattform an den Rahmen des Mittelrumpfs
Antrieb		
2	„Rhino-VX-34“-Außenbordmotoren (elektrisch) + Kabel	der Antrieb und das Lenkverhalten wird durch zwei Außenbordmotoren ermöglicht

Tabelle A.1.: Übersicht der Hardwarebestandteile

Stückzahl	Bezeichnung	Anmerkung
Ersatzteile		
1	Propeller für den „Rhino-VX-34“-Außenbordmotor	Original Ersatzpropeller
10	Halbrundschraube M6 x 16 mm	Ersatzschrauben für ITEM-Winkelsätze
2	Universal-Verbindungssatz	Ersatzverbindungssatz für Querstreben
20	Nutensteine rostfrei	Originale Ersatznutensteine

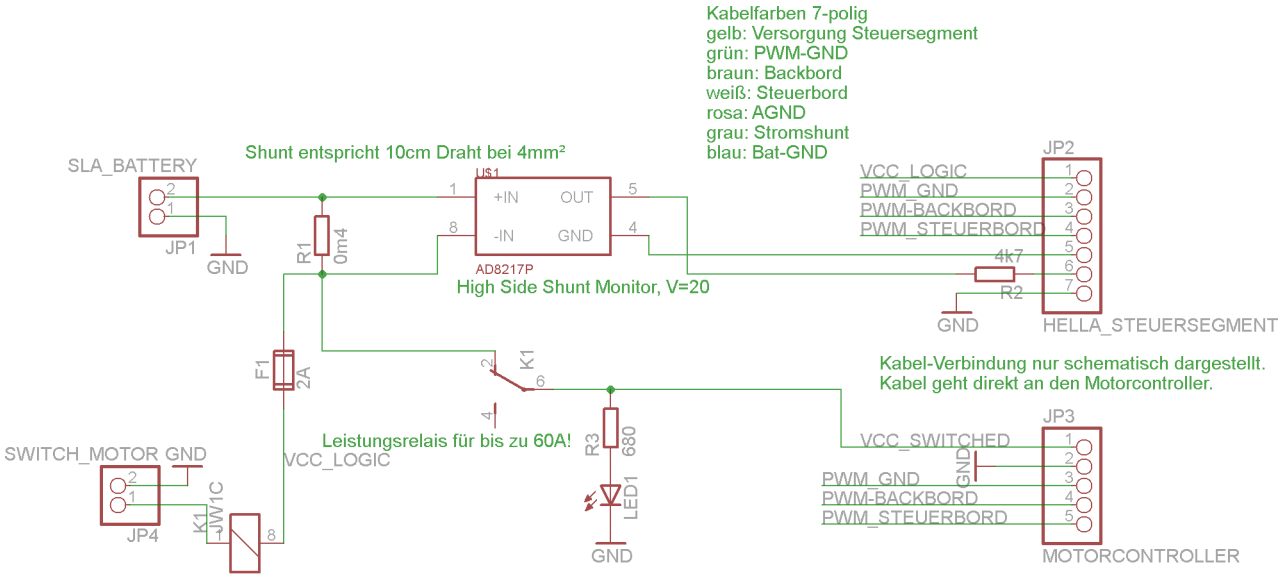
Tabelle A.2.: Übersicht der Ersatzteile

A.2. Elektrische Komponenten

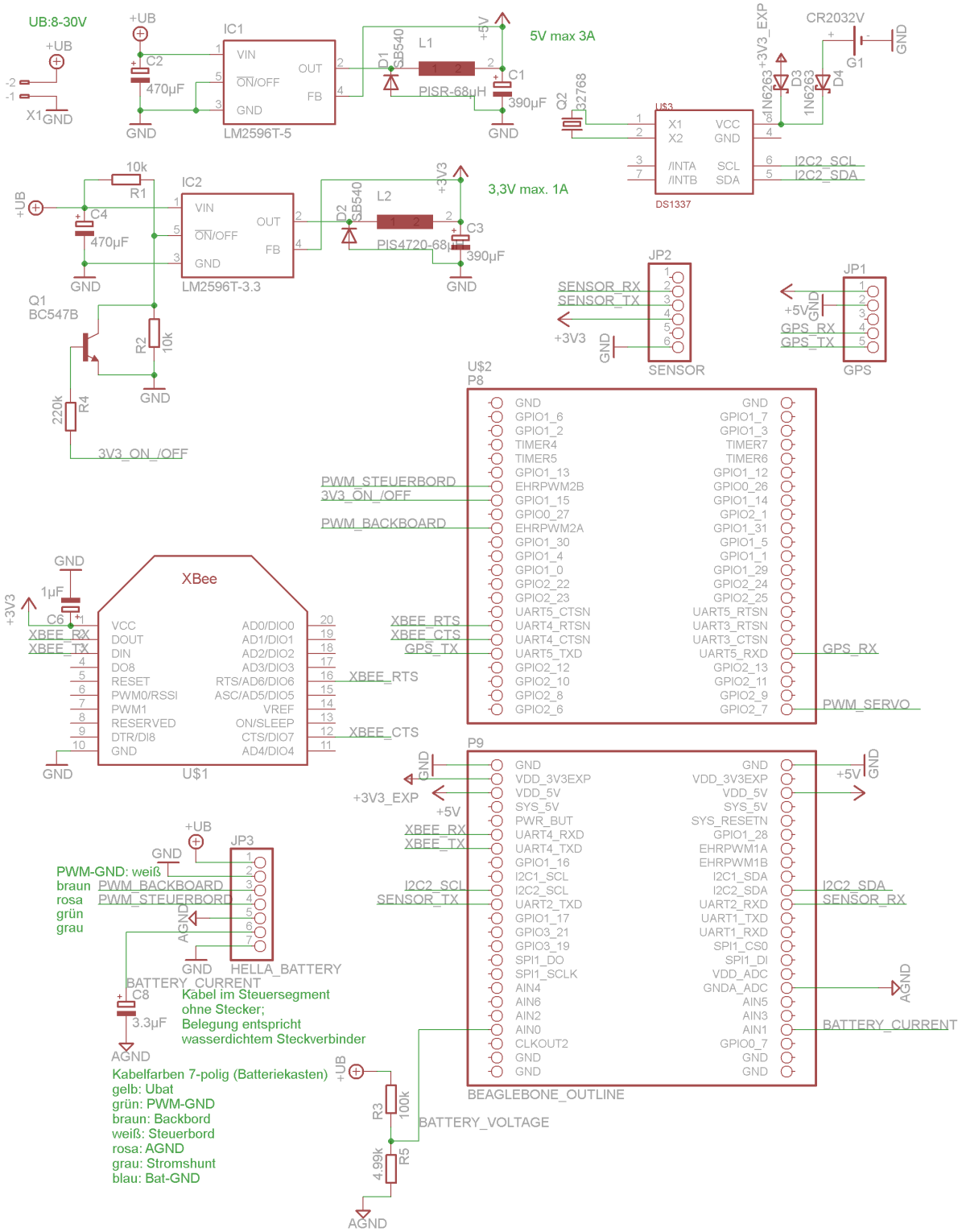
Stückzahl	Bezeichnung	Anmerkung
Baugruppen, Sensoren		
1	CircuitCo BeagleBone	Steuercomputer
1	Navilock NL-652ETTL	u-blox 6 GPS Modul
1	Sparkfun 9 Degrees of Freedom - Razor IMU (SEN-10736)	Inertialsensor mit Magnetfeld-, Beschleunigungs-, und Winkelbeschleunigungssensoren
2	Xbee Pro 868	868 MHz Funkmodule für hohe Reichweiten
1	DimensionEngineering Sabertooth dual 60A motor driver	Gleichstrom-Motor Regler für bis zu 60V 2x60A
Batteriekasten		
1	FRA2 C 12	12V Hochstromrelais
3	PGBF 13	Kabelverschraubung Motorkabel, Steuersegmentkabel
3	PGGM 13	Kabelverschraubung Motorkabel, Steuersegmentkabel
1	PGBF 7	Kabelverschraubung Schalter
1	PGGM 7	Kabelverschraubung Schalter
2	EL ESZ	Schalter Motorregler
1	LIYY 314-10	Kabel zu Motorschalter
8m	Cordial Speakerkabel 2 x 4,0qmm CLS 240	Motorkabel
2	AST 021-02	Anschlussklemmen
1	AD8217	Shunt Monitor

Stückzahl	Bezeichnung	Anmerkung
Steuersegment		
1	Lochrasterplatine, 160x100mm	
1	P 3596 L-5,0	Schaltregler 5V
1	P 3596 L-3,3	Schaltregler 3,3V
2	RAD FR 390/25	Spannungsversorgung
2	RAD FR 470/35	Spannungsversorgung
1	L-PISR 68µ	Spannungsversorgung
1	L-PIS4720 68µ	Spannungsversorgung
2	SB 540	Spannungsversorgung
1	TANTAL 1,0/35	Spannungsversorgung
5	MPE 087-2-040	Stiftleiste für Beaglebone
2	BL 1X20G 2,54	Xbee-Adapter, div. Steckverbindungen
2	SL 1X40G 2,54	Xbee-Adapter, div. Steckverbindungen
1	BL 1X10G 2,00	Xbee-Adapter
1	Watterott BOB-08276	Xbee-Adapter
1	ALLNET ALL0234M	USB-Wlan Stick
1	LIYY 614-10	div. Anschlusskabel
1	LITZE SW	div. Anschlusskabel
1	LA 205-5	div. Anschlusskabel
1	AKL 249-02	Stromversorgung
1	AKL 220-02	Stromversorgung
1	Hella/SVB 217+220+202	wasserdichte Steckverbindung, 7pol.
1	Hella/SVB 8XT006798001	Montagesockel für Steckverbindung
1	RAD FC 3,3/50	div. Bauteile
1	METALL 100K	div. Bauteile
1	METALL 4,99K	div. Bauteile
2	METALL 10K	div. Bauteile
1	METALL 220K	div. Bauteile
1	BC 547C	div. Bauteile
1	DS 1337	I2C-RTC
1	32,768 MS1V-6	RTC
2	1N 6263	RTC
1	CR 2032	RTC
1	KZH 20PCB-V	RTC

A.3. Schaltplan Batteriesegment



A.4. Schaltplan Steuersegment



A.5. Hardwarebeschaffung

- Teilliste Antrieb (siehe Tabelle A.3)
- Teilliste Rumpf- und Bugsegment (siehe Tabelle A.4)
- Teilliste Steuerungssegment (siehe Tabelle A.5)
- Teilliste Energiesegment (siehe Tabelle A.6)
- Rahmenkonstruktion (siehe Tabelle A.7)

Antriebssegment			
Gesamtkosten:			368,20
Menge	**Funktion / Bezeichnung**	**Hersteller / Bezugsquelle**	**Preis(€)**
2	Rhino VX 34 Motoren	bootsmotoren4you.de	333,00
8 m	Cordial Speakerkabel 2 x 4,0 mm CLS 240	amazon.de	19,20
4	Siebdruckplatten 9 mm stark 100 cm x 9 cm	Obi	16,00

Tabelle A.3.: Übersicht Teilliste Antrieb

Rumpfsegment			
Gesamtkosten:			464,26
Menge	**Funktion / Bezeichnung**	**Hersteller / Bezugsquelle**	**Preis(€)**
3	Reinigungsrohr mit Klappe / Reinigungsrohr (KGRE)	techboerse.de	174,57
2	KG Rohre 2,00 m mit Steckmuffe DN 200 (KGEM)	techboerse.de	48,47
9	Endverschluss / Muffenstopfen(KGM)	techboerse.de	40,14
9	Endkappen (KGK)	techboerse.de	43,29
3	Sikaflex 291 Kartusche 300 ml	Sika Deutschland GmbH / 24trade.de	26,70
4	KG Rohre mit Steckmuffe DN 200 (KGEM) 0,5 m	techboerse.de	30,04
1	PU-Schaum (zwei-komponentig)	schiffszubehoer.eu	50,60
1	Gleitmittel für Kg-Teile	techboerse.de	1,77
1	Styropor-Verpackungschips, 400 l	verpackungs- discount24.de	27,82
Satz	Bugverbinder (Schrauben, Scheiben,Muttern)	Hornbach	13,54
Bugsegment			
Gesamtkosten:			91,20
5	Trichter PRESSOL Polyethylen - 2,9 l 200 mm Durchmesser	ESSKA.de	35,55
5	Gelenkbolzenschellen	ESSKA.de	55,65

Tabelle A.4.: Übersicht Teilliste Rumpf- und Bugsegment

Steuerungssegment			
Gesamtkosten:			603,15
Menge	**Funktion / Bezeichnung**	**Hersteller / Bezugsquelle**	**Preis(€)**
1	CircuitCo BeagleBone	watterott.com	80,92
1	Navilock NL-652ETTL GPS Modul	amazon.de	32,55
1	Sparkfun 9 Degrees of Freedom - Razor IMU (SEN-10736)	sparkfun.com	96,00
2	Xbee Pro 868 MHz Funkmodule für hohe Reichweiten	watterott.de	169,90
1	Dimension-Engineering Sabertooth dual 60 A motor driver	lipoly.de	159,00
1	Lochrasterplatine, 160 mm x 100 mm	reichelt.de	1,90
1	P 3596 L-5,0 Schaltregler 5 V	reichelt.de	1,00
1	P 3596 L-3,3 Schaltregler 3,3 V	reichelt.de	1,15
2	RAD FR 390/25	reichelt.de	0,54
2	RAD FR 470/35	reichelt.de	0,62
1	L-PISR 68	reichelt.de	0,85
1	L-PIS4720 68	reichelt.de	0,95
2	SB 540	reichelt.de	0,28
1	TANTAL 1,0/35	reichelt.de	0,14
5	MPE 087-2-040	reichelt.de	1,40
2	BL 1X20G 2,54	reichelt.de	4,00
2	SL 1X40G 2,54	reichelt.de	0,52
1	BL 1X10G 2,00	reichelt.de	0,24
1	Watterott BOB-08276	watterott.de	2,29
1	ALLNET ALL0234M USB-Wlan Stick	reichelt.de	4,95
1	LIYY 614-10	reichelt.de	3,75
1	LITZE SW	reichelt.de	0,74
1	LA 205-5	reichelt.de	1,00
1	AKL 249-02	reichelt.de	0,26
1	AKL 220-02	reichelt.de	0,26
1	Hella/SVB 217+220+202 wasserdichte Steckverbindung, 7pol.	svb.de	27,90
1	Hella/SVB 8XT006798001 Montagesockel für Steckverbindung	svb.de	5,80
1	RAD FC 3,3/50	reichelt.de	0,12
1	METALL 100K	reichelt.de	0,08
1	METALL 4,99K	reichelt.de	0,08
2	METALL 10K	reichelt.de	0,16
1	METALL 220K	reichelt.de	0,08
1	BC 547C	reichelt.de	0,04
1	DS 1337	reichelt.de	2,15
1	32,768 MS1V-6	reichelt.de	0,36
2	1N 6263	reichelt.de	0,82
1	CR 2032	reichelt.de	0,44
1	KZH 20PCB-V	reichelt.de	0,45

Tabelle A.5.: Übersicht Teilliste Steuerungssegment

Energiesegment			
Gesamtkosten:			44,97
Menge	**Funktion / Bezeichnung**	**Hersteller / Bezugsquelle**	**Preis(€)**
1	FRA2 C 12	reichelt.de	1,30
3	PGBF 13	reichelt.de	1,50
3	PGGM 13	reichelt.de	1,50
1	PGBF 7	reichelt.de	0,36
1	PGGM 7	reichelt.de	0,16
2	EL ESZ	reichelt.de	1,90
1	LIYY 314-10	reichelt.de	2,40
2	AST 021-02	reichelt.de	0,88
1	AD8217 Shunt Monitor	reichelt.de	12,67
1	AG Akku 100 Ah	zur Verfügung gestellt durch Prof. Hahn	-
1	wasserdichter belüfteter Batteriekasten	Segelladen.de	22,30

Tabelle A.6.: Übersicht Teilliste Energiesegment

Rahmenkonstruktion			
Gesamtkosten:			411,82
Menge	**Funktion / Bezeichnung**	**Hersteller / Bezugsquelle**	**Preis(€)**
2	Profil 6 30 mm x 30 mm leicht, natur 2000 mm	item Industrietechnik GmbH	40,70
2	Profil 6 30 mm x 30 mm leicht, natur 2200 mm	item Industrietechnik GmbH	44,32
2	Profil 6 30 mm x 30 mm leicht, natur 2140 mm	item Industrietechnik GmbH	43,32
4	Profil 6 30 mm x 30 mm leicht, natur 115 mm	item Industrietechnik GmbH	13,20
4	Winkelsatz 6 60 mm x 60 mm	item Industrietechnik GmbH	23,51
8	Winkelsatz 6 30 mm x 30 mm	item Industrietechnik GmbH	32,11
8	Abdeckklappen 6 30 mm x 30 mm	item Industrietechnik GmbH	3,24
30	Nutenstein 6 m6	item Industrietechnik GmbH	46,71
1	Profil 6 30 mm x 30 mm leicht, natur 2000 mm	item Industrietechnik GmbH	20,35
8	Winkel 6 T2 45 grad	item Industrietechnik GmbH	43,20
8	Universal-Verbindungssatz 6	item Industrietechnik GmbH	28,15
4	Profil 6 30 mm x 30 mm leicht, natur 1050 mm	item Industrietechnik GmbH	64,44
4	Halbrundschraube m6 x 16	item Industrietechnik GmbH	0,68
10	Nutenstein 6 m6	item Industrietechnik GmbH	17,30

Tabelle A.7.: Übersicht Teilliste Rahmen

A.6. CAD-Modell des MOPS

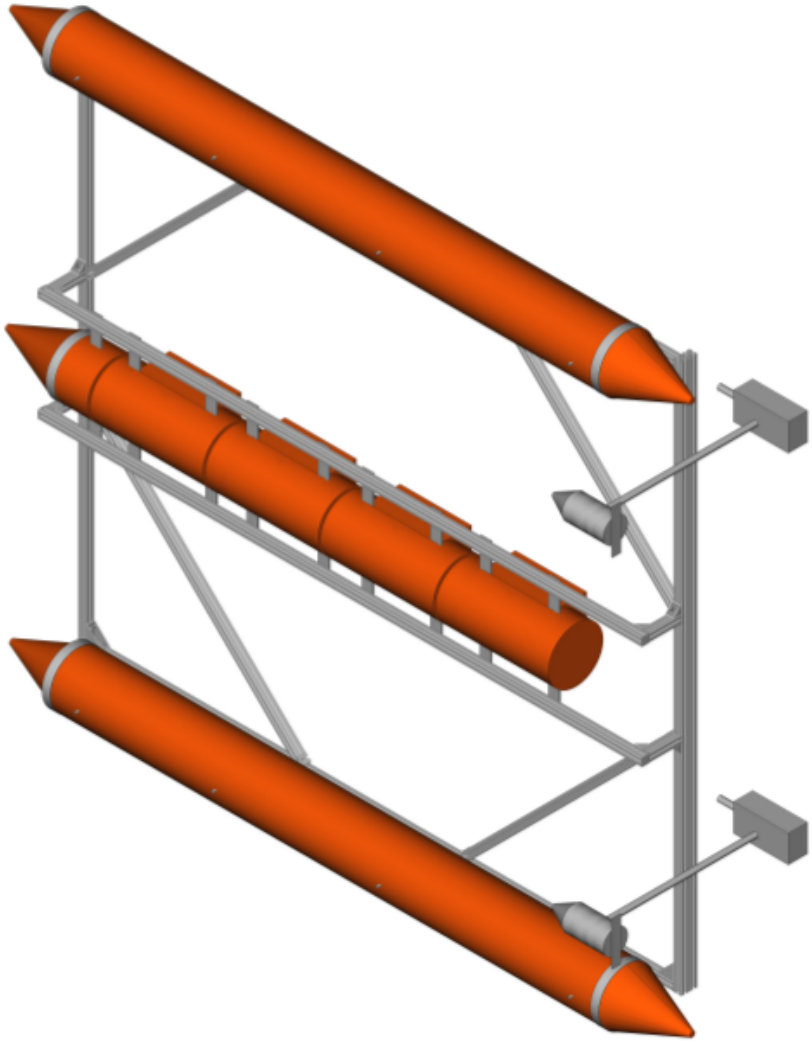


Abbildung A.1.: CAD-Modell: Räumliche Ansicht

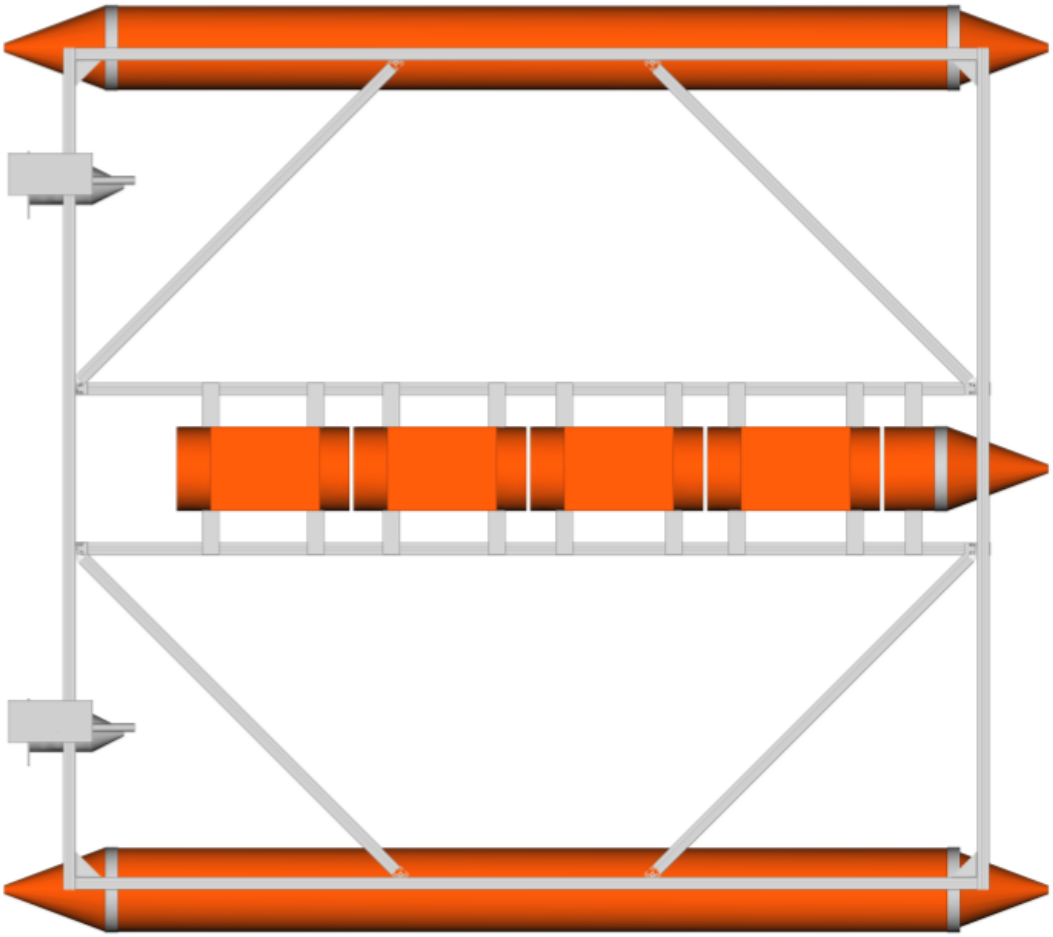


Abbildung A.2.: CAD-Modell: Ansicht von oben

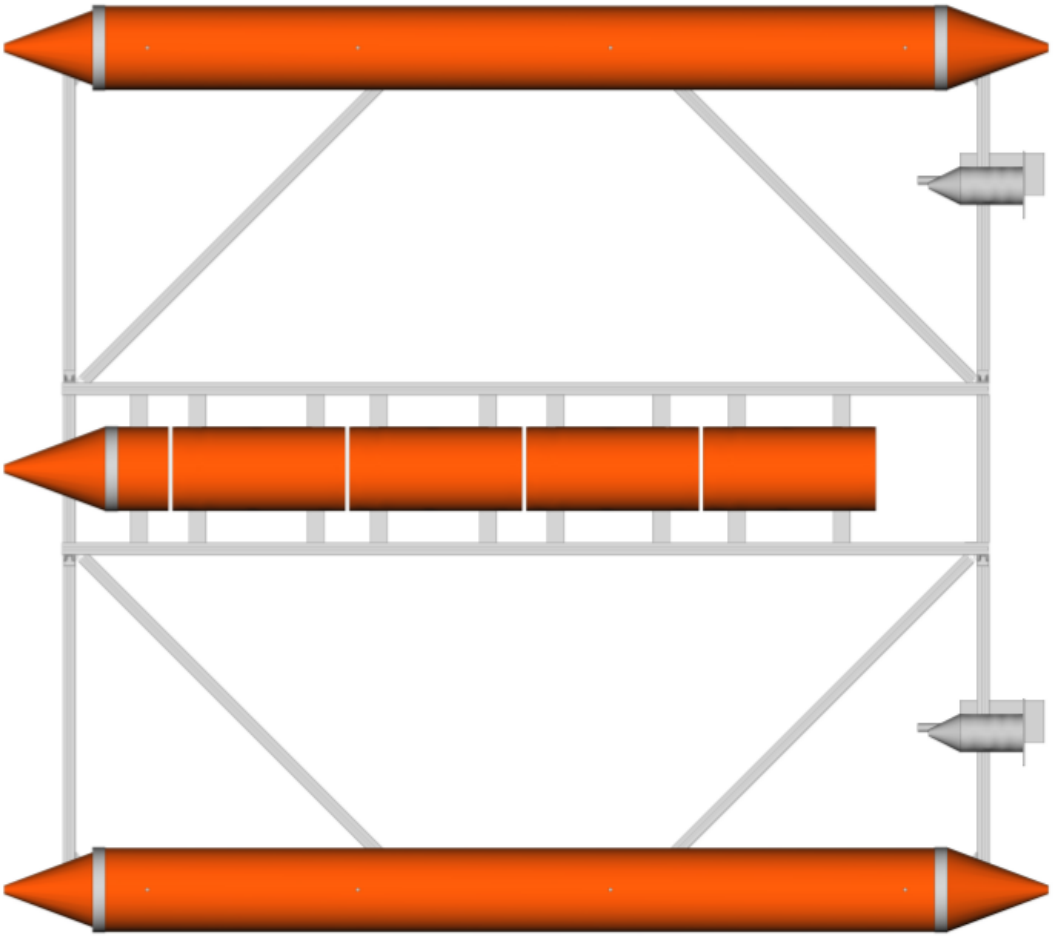


Abbildung A.3.: CAD-Modell: Ansicht von unten

Abbildung A.4.: CAD-Modell: seitliche Ansicht

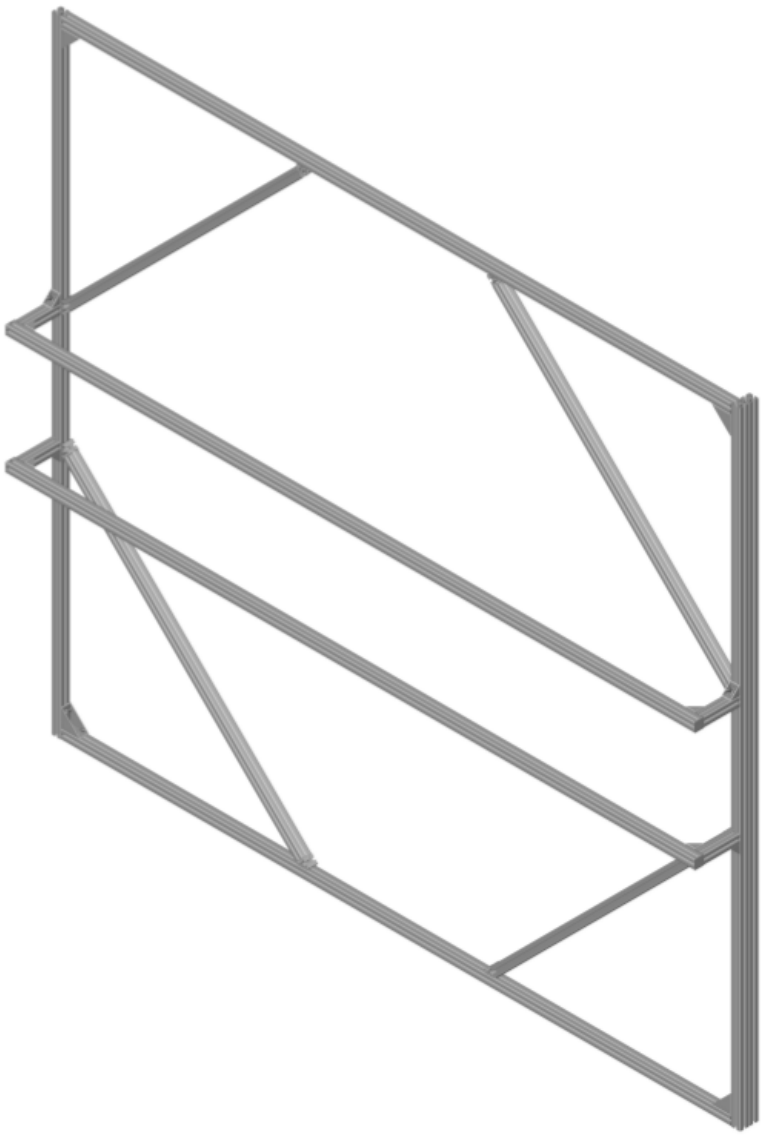


Abbildung A.5.: CAD-Modell-Rahmen: Räumliche Ansicht

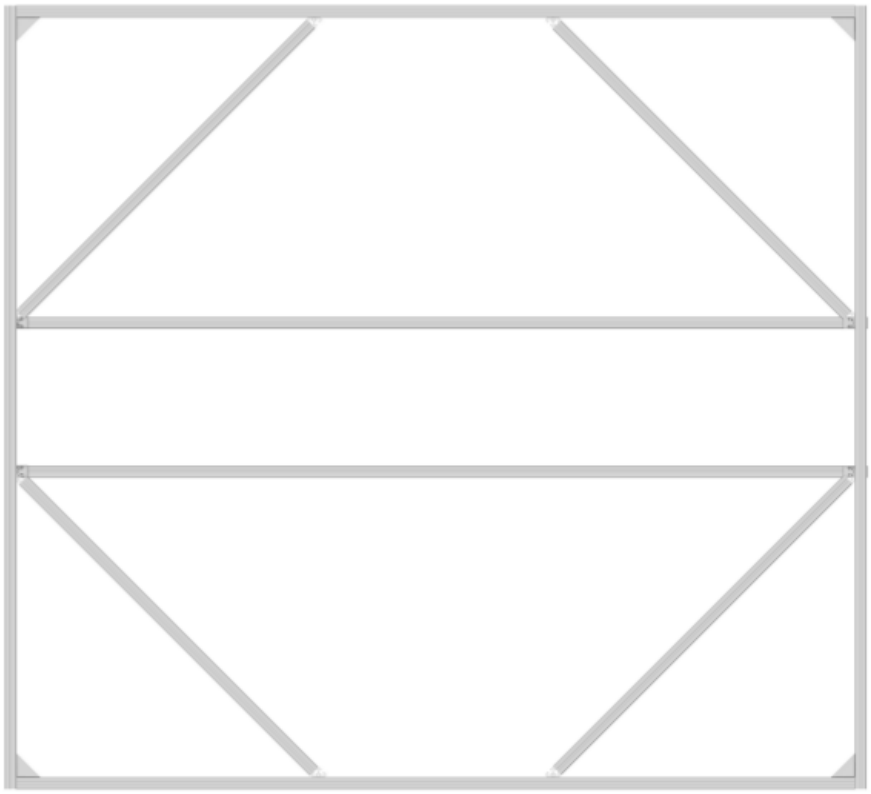


Abbildung A.6.: CAD-Modell-Rahmen: Ansicht von oben