ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG

INSTITUT FÜR INFORMATIK

Arbeitsgruppe Autonome Intelligente Systeme

Prof. Dr. Wolfram Burgard

Simulation eines Prallluftschiffs

Studienarbeit

Clemens Eppner

Oktober 2007

Betreuer: Axel Rottmann

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Außerdem erkläre ich, dass die Studienarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Clemens Eppner
Freiburg, den 22. Oktober 2007

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die aktuelle Renaissance von Luftschiffen hat mehrere Ursachen. Zum Einen bieten unbemannte Luftfahrzeuge ein großes Anwendungspotential, angefangen bei der Verkehrsüberwachung, über Stadtplanung, Überprüfung großräumiger Anlagen und archäologischer Stätten bis hin zu vielfältigen militärischen Aufgaben - überall kommen die sog. UAVs (*Unmanned Aerial Vehicle*) zum Einsatz. Zum Anderen spielt die Kontrolle der Umwelt eine immer größere Rolle. Dazu gehören das Taxieren von Artenvielfalten, die Klimaforschung, die Bestimmung von Wasser- und Luftverschmutzung, die Überwachung von Naturschutzparks oder landwirtschaftlich genutzter Flächen.

Gegenwärtig wird ein Großteil dieser Anwendungen noch auf der Basis von Sensordaten realisiert, die mittels Ballone, Satelliten oder bemannten Luftfahrzeugen gesammelt werden. Die Nachteile solcher Informationsquellen liegen auf der Hand: Ballone sind nicht manövrierfähig, das erkannte Gebiet kann so nicht direkt festgelegt werden. Für die zivile Nutzung freigegebene Satellitenbilder sind in Bezug auf räumliche Auflösung, verfügbare Frequenzbereiche sowie zeitliche und räumliche Abdeckung des Untersuchungsgegenstandes begrenzt. Während bemannte Lufterkundungen die Kontrolle über die o.g. Variablen dem Missionsplaner überlassen, ist diese Form mit hohen Kosten bezüglich des verwendeten Flugzeugs, Besatzung, Wartung, usw. verbunden.

Ein autonomes Luftschiff geht allen diesen Nachteilen aus dem Weg. Es lässt sich zielgerichtet navigieren und muss sich nicht bewegen um eine konstante Höhe zu halten. Zudem ist es nicht überempfindlich in Bezug auf Kontrollfehler, wie beispielsweise ein Helikopter, und verursacht nur geringe Betriebs- und Wartungskosten.
Man unterscheidet drei Typen von Luftschiffen: Prallluftschiffe (im folgenden als *Blimps* bezeichnet), halbstarre Luftschiffe und Starrluftschiffe (auch Zeppeline genannt). Während Blimps ihre Form durch den Überdruck in der Hülle erhalten, geschieht dies bei halbstarren und Starrluftschiffen durch eine Tragstruktur bzw. ein festes inneres Gerüst.

Die Diskussion um die Wirksamkeit von Simulationen im Bereich der mobilen Robotik zeigt eine große Bandbreite an Meinungen. Darunter finden sich auch radikale Vertreter wieder [1]:

> „Simulation in Robotics, control theory and AI has mostly been a complete waste of time“

Sobald eine funktionstüchtige Simulation erstellt ist, lassen sich ihre Vorteile jedoch kaum von der Hand weisen:

- Simulieren ist billiger. Denn echte Hardware verschleißt und geht kaputt. Zuvor nicht gewagte risikoreiche Experimente, bei denen den meist teuren Roboterplattformen Schäden drohen würden, können im Simulator beliebig oft durchgespielt werden.
- Simulieren ist schneller. Denn das Übertragen der Kontrollsoftware auf die Roboterplattform kostet Zeit. Hinzu kommt beim Blimp ein aufwändiges Vorbereitungsritual (Hülle mit Helium füllen etc.). Die Zeitersparnis schlägt sich auch auf zeitaufwändige Lernverfahren nieder, da die Simulationszeit beliebig beschleunigt werden kann.
- Simulieren ist aussagekräftig. Denn Experimente können im Simulator leichter überwacht und ausgewertet werden. Ein einmal konstruierter Versuchsaufbau kann problemlos wiederholt und mit variierenden Verfahren durchexerziert werden.

1.2 Ziele und Aufbau der Arbeit

Ziel dieser Arbeit ist es, eine möglichst realitätsnahe Simulation eines Blimps zu entwickeln, die es anschließend ermöglicht effektiv verschiedenste Anwendung für diese Plattform zu testen. Bei dem nachzuahmendem Luftschiff handelt es sich um eine ca. 180 cm lange, kommerziell erhältliche Hülle [11] kombiniert mit maßgefertigter Hardware (siehe Abbildung 1.1).

ABBILDUNG 1.1: Der zu simulierende Blimp

Die Fortbewegung erfolgt mittels dreier Triebwerke. Kamera, Kompass, Beschleunigungs- und Sonarsensor helfen dem Blimp, seine Umwelt wahrzunehmen. Diese gewonnenen Daten werden regelmäßig an die Bodenstation weitergesendet. Der Simulator hat demnach sowohl die Aufgabe, die von den verschiedenen Sensoren aufgenommenen Daten zu generieren, als auch die von

der Bodenstation eingehenden Steuerungskommandos mittels eines realistischen Blimpmodells zur Manipulation dessen Zustands zu verwenden.

Besonderer Augenmerk liegt auf der authentischen Modellierung der dynamischen und kinematischen Eigenschaften des Blimps, sowie der wirklichkeitsnahen Simulation des Sonarsensors.

Die Arbeit ist wie folgt gegliedert: Zunächst beschreibt Kapitel 2 den Aufbau und die Arbeitsweise des Blimps näher. In Kapitel 3 folgt eine Erläuterung der zentralen Simulationsumgebung einschließlich des Zusammenspiels der einzelnen, bei der Simulation beteiligten, Einheiten. Anschließend schildert Kapitel 4 das dem Luftschiff zu Grunde liegende physikalische Modell.
Die Modellierung des Sonarsensors, basierend auf der Bayes'schen Regression zuvor ermittelter Beispieldaten, wird in Kapitel 5 abgehandelt. Die Funktionstüchtigkeit des Simulators anhand eines Versuchs zum bestärkenden Lernen evaluiert Kapitel 6, bevor ein abschließendes Fazit gezogen wird.

2 Beschreibung des Blimps

2.1 Aufbau und Funktionsweise

Der Blimp besteht aus einer mit Helium gefüllten Kunststoffhülle, an deren Unterseite eine Gondel befestigt ist. Er wiegt insgesamt 430 g und misst entlang seiner größten Ausdehnungen 90 cm (Höhe), 180 cm (Länge) bzw. 80 cm (Breite).

Im Inneren der Gondel befinden sich zahlreiche Sensoren. Sie liefern visuelle Informationen der Umgebung (Kamera), bestimmen die Drehung des Blimps um die Gierachse (Kompass), messen auftretende Beschleunigungen in allen drei Dimensionen und somit die Lage im Raum (Beschleunigungssensor) und quantifizieren den Abstand zum Erdboden (Sonarsensor). Alle elektronischen Bauteile werden von einem 7,4V/910mAh Lithium-Ionen-Akku gespeist.

Des Weiteren dient eine Funkeinheit der drahtlosen Kommunikation mit der Bodenstation. Dieser sendet der Blimp regelmäßig alle Sensordaten. Im Gegenzug empfängt der Blimp Navigationskommandos zur Ansteuerung der einzelnen Motoren, sowie Einstellungen für die Sensoren.

ABBILDUNG 2.1: Die Gondel ist mit zwei um 180° schwenkbaren Propellern versehen.

Der Blimp verfügt insgesamt über drei Propeller: zwei davon befinden sich an der Gondel. Sie sind durch eine gemeinsame Achse verbunden, deren Drehung wiederum von einem Servo kontrolliert wird. So können die beiden Gondelmotoren um den Winkel $\mu \in [-90^\circ, +90^\circ]$ geneigt werden (siehe Abbildung 2.1). Entwickeln die Motoren einen positiven Schub, dann erfährt der Blimp eine Bewegung in Vorwärtsrichtung ($\mu = 90^\circ$), Rückwärtsrichtung ($\mu = -90^\circ$) oder nach oben ($\mu = 0^\circ$).

Es ist theoretisch möglich, die zwei Gondelpropeller unterschiedlich stark anzutreiben, um eine Drehung des Blimps um die Gierachse zu verursachen. Dies wird jedoch aus Stabilitätsgründen

Name	Wert	Bedeutung
l_{blimp}	$1,8m$	Länge
m_{blimp}	$430\ g$	Masse
r_{hull}	$0,4\ m$	Querradius der Hülle
μ	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	Winkelstellung der Gondelmotoren

TABELLE 2.1: Alle relevanten Kenngrößen im Überblick

vermieden. Stattdessen erfolgt das Gieren ausschließlich über den dritten, in der Heckflosse angebrachten Propeller.

Alle Maße, die für die Simulation des Blimps von Bedeutung sind und im Verlaufe der folgenden Kapitel immer wieder auftauchen, sind in Tabelle 2.1 definiert.

2.2 Datenaustausch mit der Bodenstation

Die bidirektionale Kommunikation zwischen dem Blimp und der Bodenstation beinhaltet zum einen die Sensordaten des Blimps, sowie die von der Bodenstation erzeugten Steuerungskommandos.

Der detaillierte Ablauf zum Austausch von Daten und Anweisungen sieht wie folgt aus:

1. Der Blimp sendet eine fest formatierte Zeichenkette, bestehend aus leerzeichengetrennten Sensordaten und einem Zeitstempel:

Zeitstempel	Beschleunigungssensor	Kompass	Sonarmessung
T%02d%02d%02d%02d	W%04X%04X%04X	K%.1f	D%.2f\n

2. Falls die Bodenstation ein Kommando (siehe Tabelle 2.2) geschickt hat, wird dieses vom Blimp analysiert, ausgeführt und eine Antwort zurückgesendet. Befehle, die keine Information verlangen (z.B. linken Gondelmotor stoppen), werden gemäß Protokollvereinbarung mit der identischen Zeichenkette beantwortet. So wird gewährleistet, dass die Bodenstation zu jedem Zeitpunkt mit hoher Wahrscheinlichkeit ein identisches Abbild vom Zustand des Blimps besitzt und Fehler bei der Datenübertragung ausgeschlossen werden können.
3. Gehe zu 1.

Kommando	**Bedeutung**
a{-127…128}	setzt Leistung des linken Gondelmotors -127: maximale Kraft zurück 0: Stillstand 128: maximale Kraft voraus
b{-127…128}	setzt Leistung des rechten Gondelmotors
c{-127…128}	setzt Leistung des Heckmotors -127: maximale Kraft zurück 0: Stillstand 128: maximale Kraft voraus
s{-2212…2212}	setzt Winkelausrichtung der Gondelmotoren -2212: $\mu = 90°$; bei positivem Gondelmotorenschub Bewegung nach vorn (positive x-Achse) 0: $\mu = 0°$; bei positivem Gondelmotorenschub Bewegung nach oben (positive z-Achse) 2212: $\mu = -90°$; bei positivem Gondelmotorenschub Bewegung nach hinten (negative x-Achse)
u{1…16}	setzt den Verstärkungsfaktor (*Amplifier*) des Sonarsensors 1: niedrigste Stufe 16: höchste Stufe
t	setzt den Zeitstempel der zu sendenen Sensordaten wieder auf null
m	Kalibrierung der Beschleunigungssensoren; während einer 200 ms Ruhephase in Nullstellung (Orientierung) werden die Mittelwerte für die Beschleunigungssensoren errechnet und zurückgegeben
v	fordert Batteriestatus an
o	setzt standby time; variiert die Sendefrequenz des Blimps (Standard: 5 Hz)

TABELLE 2.2: Alle Befehle in der Übersicht

3 Simulationsarchitektur

3.1 Allgemeine Struktur

Um den autonomen Blimp zu simulieren, ist es notwendig, dem zuständigen Steuerungsprogramm vorzugeben, es kommuniziere wie gehabt über eine serielle Schnittstelle mit dem echten Luftschiff. Dann ist eine nahtlose Integration der Steuerungssoftware von der Simulation in die Realität möglich; zusätzlicher Portierungsaufwand entfällt.

Die Simulationsumgebung *Gazebo*, auf die im folgenden Abschnitt detaillierter eingegangen wird, dient der Visualisierung und birgt zusätzlich ein allgemeines physikalisches Weltmodell, von deren Kollisionserkennung beispielsweise Gebrauch gemacht wird. Das eigentliche Simulationsprogramm (*./simRobot*) greift über Schnittstellen auf diese Daten zu und kann so die Funktionsweise der am Blimp angebrachten Sensoren nachahmen. Außerdem wird mittels einer Schnittstelle die Dynamik des in Gazebo modellierten Luftschiffs manipuliert.

Der Informationsaustausch zwischen Simulator (*./simRobot*) und der am „Boden“ befindlichen Steuerungssoftware (*./blimp*) erfolgt unter Einhaltung eines zuvor vereinbarten Protokolls (siehe 2.2). Das Steuerungsprogramm selbst ist durch IPC (*Inter-Process Communication*) in der Lage, Daten zwischen zwei oder mehreren Threads in einem oder mehreren Prozessen auszutauschen. Die Prozesse können auf mehreren Computern laufen, die durch ein Netzwerk verbunden sind (TCP/IP). Ein zentraler Server verwaltet alle registrierten Prozesse, d.h. leitet eingehende Nachrichten an alle interessierten Prozesse weiter. Diese Architektur ermöglicht es, beliebig zusätzliche Module hinzuzufügen, die mit Hilfe des Steuerungsprogramms Zugang zu allen Sensordaten und Steuerungskommandos des Blimps haben. Beispiele für solche Module sind ein Joystick, dessen Befehle die einzelnen Motoren des Luftschiffs kontrollieren, ein Logger, der alle auftretenden Ereignisse protokolliert, oder ein Lernverfahren (*Reinforcement Learning*, siehe Kapitel 6) zur Ermittlung des optimalen Steuerverhaltens für das Anfliegen einer bestimmten Höhe.

Eine zusammenfassende schematische Darstellung des Zusammenspiels aller bei der Simulation beteiligten Komponenten liefert Abbildung 3.1.

3.2 Player/Stage, Gazebo, ODE

Als Robotersimulationsumgebung wurde die weit verbreitete Plattform Player/Stage [12] verwendet. Ihr modularer Aufbau (siehe [5], [4] und [21]) ermöglicht eine einfache und direkte

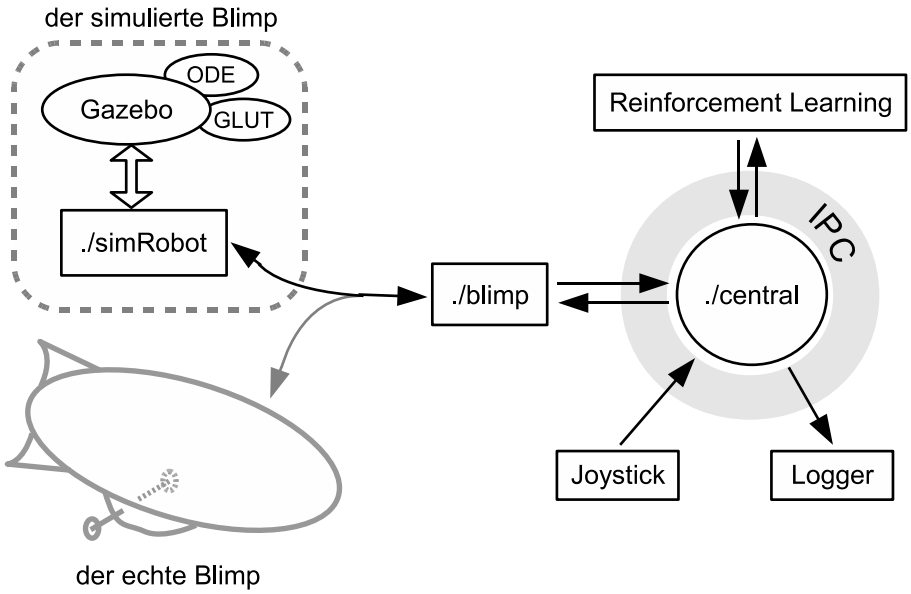

ABBILDUNG 3.1: Architektur des Simulators und den beteiligten Steuerungseinheiten

Portierung der Steuerungssoftware von der simulierten auf die echte Roboterplattform. Dabei stellt jeder Player-Server eine Netzwerkschnittstelle zu einem beliebigen Sensor oder Aktuator zur Verfügung, auf die das eigentliche Client-Programm zurückgreift, um Sensordaten zu verarbeiten und Kommandos zu generieren. Stage hingegen simuliert eine Mehrzahl mobiler Roboter, die in einer 2-dimensionalen, gerasterten, dynamischen Welt agieren.

Ein weiterer Bestandteil von Player/Stage ist die Simulationsumgebung Gazebo [3], die ebenfalls wie Stage eine Vielzahl von mobilen Robotern simuliert, jedoch nicht auf 2-dimensionale Welten beschränkt ist. Im Gegensatz zu Radrobotern spielt beim Einsatz eines Blimps die dritte Dimension eine wesentlich bedeutendere Rolle. Die zusätzliche Komplexität einer 3-dimensionalen Welt wird mit einer verminderten Anzahl an simulierten autonomen Robotern (maximal 10) erkauft. Da das Ziel dieser Arbeit in der Simulation eines einzelnen Luftschiffs besteht, kann der Nachteil einer reduzierten Multi-Roboter-Fähigkeit ignoriert werden.

Die Architektur von Gazebo (siehe [7]) soll eine einfache Erschaffung von Robotern, Aktuatoren, Sensoren und sonstigen Weltobjekten ermöglichen. Die dafür bereitgestellte API sorgt neben der Erstellung dieser Modelle auch für die Schnittstelle zu Client-Programmen. Eine Schicht unter der API befinden sich die mittels einer zusätzlichen Schicht abstrahierten Bibliotheken der Drittanbieter, zuständig für die physikalische Simulation (Open Dynamics Engine) und Visualisierung (OpenGL und GLUT). Diese Trennung verhindert, dass Modelle von spezifischen Werkzeugen abhängen können, die sich möglicherweise in der Zukunft ändern (siehe Abbildung

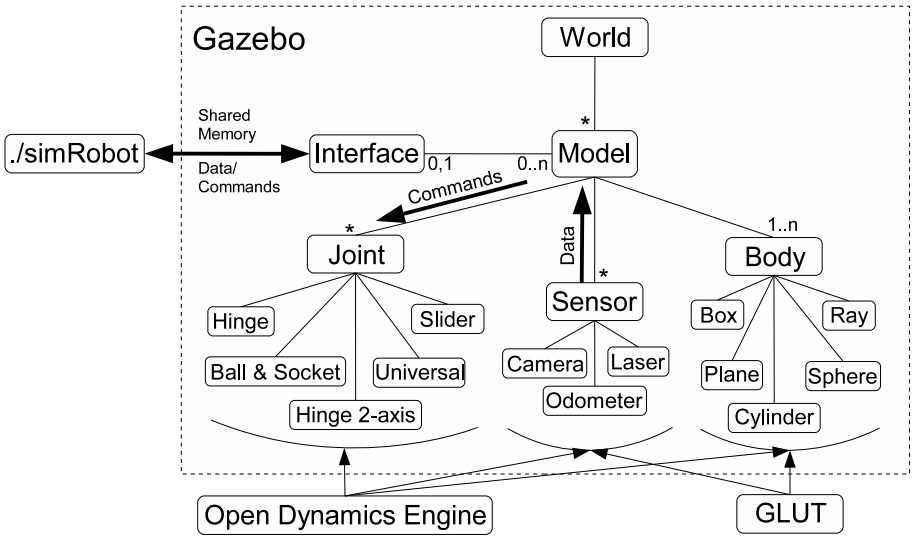

ABBILDUNG 3.2: Die Hierarchie in der Gazebo-Welt

3.2).

Die Welt repräsentiert die Menge aller Modelle und Umgebungsfaktoren wie Gravitation oder Beleuchtung. Jedes Modell besteht aus mindestens einem Körper und einer beliebigen Anzahl an Sensoren und Gelenken. Eine Gazebo-Welt wird mittels einer einfachen XML-Datei beschrieben. Über einen gemeinsamen Speicherbereich werden Client-Befehle empfangen und Daten zurückgegeben. Es existieren verschiedene solcher Schnittstellen, z.B. für die Übertragung von Kamerabildern oder Gelenksteuerungskommandos.

Die Open Dynamics Engine (ODE) [2] simuliert die Dynamik und Kinematik beweglicher fester Körper. Zu ihren Fähigkeiten zählen: Kollisionserkennung, Masse- und Rotationsfunktionen, sowie zahlreiche Gelenktypen. Um diese nutzen zu können, stellt Gazebo eine Abstraktionsschicht zwischen ODE und den Gazebo-Modellen bereit. So lassen sich normale und abstrakte Objekte, wie Laserstrahlen oder Bodenebenen erzeugen. Weiterhin wird gewährleistet, dass die physikalische Engine jederzeit problemlos ausgetauscht werden kann, sollte eine lohnendere Alternative verfügbar sein.

Eine vollständige Welt ist im Wesentlichen eine Sammlung von Modellen und Sensoren. Es wird zwischen stationären Modellen, wie Gebäuden oder Böden, und dynamischen, d.h. den autonomen Robotern, unterschieden. Sensoren bleiben getrennt von der dynamischen Simulation, da sie nur Daten sammeln oder, im Falle eines aktiven Sensors, Daten aussenden.

Es folgt eine kurze Beschreibung aller in der Simulation involvierten Komponenten:

- Modell (*model*): Jedes Objekt, das eine physikalische Repräsentation umfasst, ist ein Mo-

dell. Dies beginnt bei einfachen geometrischen Objekten und geht bis hin zu komplexen Robotern. Modelle bestehen aus mindestens einem Körper, beliebig vielen Gelenken und Sensoren, sowie Schnittstellen zur Steuerung des Datenflusses.

- Körper (*body*): Sind die grundlegenden Bausteine zur Erschaffung eines Modells. Ein Körper entspricht einem geometrischen Objekt (Würfel, Kugel, Zylinder, Ebene, Linie) und bekommt Eigenschaften zugewiesen, die sowohl die physikalische Simulation (Masse, Reibung, Sprungkraft), als auch die Visualisierung (Farbe, Textur, Transparenz) beeinflussen.
- Gelenk (*joint*): Verbindungsstellen, die Körper miteinander auf unterschiedlichste Art verknüpfen (Drehachse, Kugel-, Scharniergelenk) und so kinematische und dynamische Relationen zwischen ihnen definieren. Des Weiteren können sie als Motoren fungieren. Wenn eine Kraft auf ein Gelenk einwirkt, entsteht Reibung zwischen den verbundenen Körpern und den ihnen umgebenden; dies verursacht Bewegung. Die Bewegung des Blimps kann dagegen nicht mit Hilfe von Gelenken simuliert werden, da das Medium Luft im Unterschied zur Bodenebene nicht adäquat repräsentiert ist, ergo keine Reibungskräfte entstehen können.
- Sensor (*sensor*): Ein Sensor ist in Gabezo eine abstrakte Einheit ohne jegliche physikalische Repräsentation. Erst durch die Einbindung in ein Modell erhält der Sensor eine Gestalt. Dadurch kann ein einmal modellierter Sensor in beliebigen Modellen verwendet werden. Gazebo stellt drei fertige Sensorimplementierung zur Verfügung: Odometer, Kamera, sowie Strahlenlänge (*RayProximity*). Der Odometriesensor integriert die zurückgelegte Wegstrecke und liefert so eine absolute Position (bzgl. R_0, siehe Kapitel 4). Das Kamerabild wird, ausgehend von der Perspektive des Modells dem die Kamera anhaftet, mittels OpenGL gerendert. Zur Nutzung der Strahlenlängenschnittstelle wird ein Ausgangspunkt sowie eine Richtung angegeben. Dann erhält man die Länge des so definierten Strahls bis zum nächsten kollidierenden Objekt. Lediglich die Kamera des Blimps wird auf diese Weise, mit Hilfe eines schon vorhandenen Kameramodells, simuliert. Alle anderen Sensoren werden unter Zuhilfenahme der entsprechenden Schnittstellen berechnet.
- Schnittstelle (*external interface*): Jeder Sensor liefert seine Daten über die jeweilige Schnittstelle. So entspricht der Wert des Kompasssensors dem absoluten Gierwinkel des Blimps, der dem Positionsinterface entnommen wird. Der Sonarsensor hingegen benutzt die Daten des Strahlenlängeninterfaces (siehe Kapitel 5).

3.3 Repräsentation des Blimps

Der Zustand des Blimps muss nun im Simulator (*./simRobot*) repräsentiert sein. Ein Großteil seiner Eigenschaften wird dabei durch Gazebo determiniert, während einige (Batteriestatus, Sonarverstärkung) dort kein Äquivalent benötigen. Der Zugriff auf Gazebo erfolgt jeweils über die

entsprechenden Schnittstellen (*PositionInterface, SonarInterface*), die sich mit Gazebo einen gemeinsamen Speicherbereich teilen.

Zu den wichtigsten Eigenschaften zählen:

- `blimp.timestamp = PositionInterface->getTimestamp()`
  Der den Sensorinformationen mitgegebene Zeitstempel entspricht der Simulationszeit der Gazebo-Umgebung, die nicht mit der Laufzeit übereinstimmen muss. Eine beschleunigte Simulationszeit ermöglicht schnellere Experimentierabläufe, die gerade bei aufwändigen Lernverfahren wünschenswert sind.
- `blimp.accelerometer = (MEAN_ACC_X, MEAN_ACC_Y, MEAN_ACC_Z)`
  Die Beschleunigungssensoren geben die absolute Orientierung des Blimps bezüglich des Weltkoordinatensystems R_0 in Form von Fallbeschleunigungen in allen drei Dimensionen wieder. Ihre Funktionalität ist bisher noch nicht implementiert. Es werden die konstanten Werte der „Nullstellung“ verwendet.
- `blimp.compass = PositionInterface->yaw() * 180 / M_PI`
  Der Kompass berechnet sich aus der z-Orientierung des Blimps und wird in Grad angegeben.
- ```
  blimp.sonar = sonarModel->getMeasurement(
                SonarInterface->range(), blimp.sonarAmplifier)
  ```
  Dem Sonar unterliegt ein auf Gauß'schen Prozessen beruhendes Modell (siehe Kapitel 5). Seine Messwerte hängen vom *Amplifier*, der Sendestärke des Sonars, und dem tatsächlichen Abstand zwischen Sensor und Boden ab.
- ```
  PositionInterface->setSpeed( f(blimp.motorRear),
                               f(blimp.motorRight),
                               f(blimp.motorLeft),
                               0, 90 / 1438 * blimp.servo, 0)
  ```
  Die drei Antriebsmotoren werden mittels der stückweisen linearen Funktion f auf Kräfte abgebildet. Der für die Orientierung der Gondelmotorenachse zuständige Servomotor wird in einen Winkel in Grad umgerechnet. Alle Werte werden Gazebo zur Auswertung der resultierenden Luftschiffdynamik übermittelt (siehe Kapitel 4). Die maximale Motorleistung hängt bisher noch nicht vom aktuellen Batteriestatus ab.
- `blimp.battery = 7.8`
  Der Status der Batterie hat bisher keine Auswirkungen. Die Akkuspannung wird bei Anfragen immer mit dem konstanten Maximalwert angegeben. Möchte man in Zukunft Szenarien meistern, in denen der Batteriestatus des Blimps Konsequenzen auf dessen Verhalten haben soll, wäre eine lineare Approximation in Abhängigkeit der bisherigen Betriebsdauer denkbar. Da nicht nur das ununterbrochen aktive Funk-Modul Strom verbraucht, sondern auch die Elektromotoren, würde die Integration der beanspruchten Motorleistung

zu einer besseren Schätzung führen.

4 Physikalisches Modell

4.1 Theoretische Grundlagen

Das für den Blimp verwendete nicht-lineare deterministische physikalische Modell beruht zu großen Teilen auf den dynamischen Modellen aus [20] und [6].

Es wird zwischen zwei Koordinatensystemen unterschieden: ein körperfestes, dem Blimp anhaftendes, Bezugssystem R mit dem Massenschwerpunkt als Ursprung und ein starres Weltsystem R_0 (siehe Abbildung 4.1).

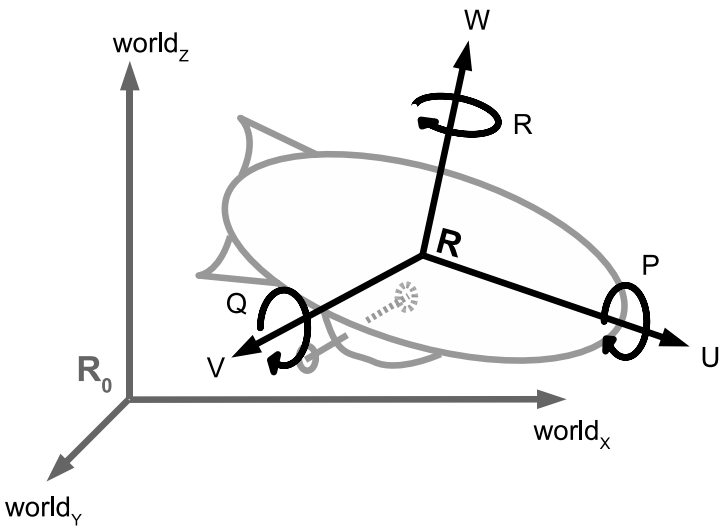

ABBILDUNG 4.1: Weltkoordinatensystem R_0 und Blimp-festes Koordinatensystem R mit den Translations- (U, V, W) und Winkelgeschwindigkeiten (P, Q, R) jeweils entlang der X-, Y-, sowie Z-Achse

Aus praktischen Gründen wird das physikalische Modell um folgende vereinfachende Annahmen limitiert:

- Der Blimp besteht aus einem festen Körper, so dass die auf der Aussenhülle stattfindenden aerolastischen Effekte ignoriert werden können.
- Der Blimp ist symmetrisch entlang der XZ-Ebene des körperfesten Koordinatensystems

R. Sowohl Massenschwerpunkt als auch das Auftriebszentrum liegen in der Symmetrieebene.

- Die Masse, das Volumen und somit der Auftrieb des Blimps sind konstant. Sie hängen weder von der Umgebungstemperatur noch dem -druck ab. Stattdessen werden ein konstanter Luftdruck auf Meereshöhe (1013,25 hPa) und eine konstante Umgebungstemperatur von 20°C angenommmen.
- Die Bewegung der Heliummoleküle im Inneren der Hülle und die damit verursachten Bewegungskräfte werden vernachlässigt.

Der kinematische und dynamische Zustand des Blimps

$$s = [p^T,\ \xi^T,\ v^T,\ \omega^T]$$

besteht aus der Position $p^T = [x,\ y,\ z]$, Orientierung $\xi^T = [\phi,\ \theta,\ \psi]$ (parametrisiert mittels *roll*, *pitch*, *yaw*), Translationsgeschwindigkeit $v^T = [U,\ V,\ W]$ und Winkelgeschwindigkeit $\omega^T = [P,\ Q,\ R]$. Dabei sind Position p und Orientierung ξ in Bezug auf das linkshändische, starre Weltsystem R_0 angegeben, die Geschwindigkeiten bezüglich des körpereigenen Koordinatensystems R.

Die zwei Gondelmotoren teilen sich eine gemeinsame Achse und sind somit jederzeit identisch gemäß Winkel μ ausgerichtet. Insgesamt verfügt der Blimp über vier Steuerungskommandos, jeweils eines für die drei Antriebsmotoren und den achsenlenkenden Servomotor:

$$u = [u_{heck},\ u_{gondel1},\ u_{gondel2},\ u_\mu].$$

Alle auf das Blimp einwirkenden Kräfte müssen, je nach Bedeutung, entweder bezüglich des starren Weltsystems R_0 oder des körperfesten Koordinatensystems R evaluiert werden. Dazu zählen (siehe Abbildung 4.2):

- **Gravitation**

  $$F_{gravitation} = [0,\ 0,\ -m_{blimp} \cdot g]^T,$$

  wobei m_{blimp} die Masse des Blimps und g die mittlere Erdbeschleunigung ist. Die Gravitationskraft wirkt im Massenschwerpunkt (*CoM*) bzgl. des Weltsystems R_0, d.h. unabhängig von der momentanen Positur des Blimps.

- **Auftrieb**

  $$F_{auftrieb} = [0,\ 0,\ g \cdot V \cdot \rho]^T,$$

  wobei g die Erdbeschleunigung, V das Volumen des Blimps und ρ die Luftdichte (20°C, 0m ü. NN). Gemäß des Archimedischen Prinzips erfährt ein Körper eine Kraft, die genauso groß ist, wie die Gewichtskraft des vom Körper verdrängten Mediums. Die Auftriebskraft wirkt im Volumenmittelpunkt (*CoB*; liegt etwas oberhalb vom *CoM*), relativ zum Weltsystem R_0.

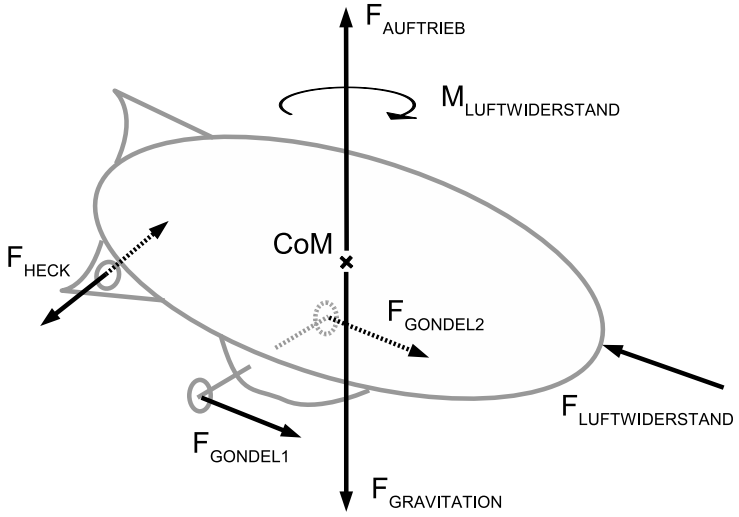

ABBILDUNG 4.2: Der Massenschwerpunkt CoM als Ursprung von R, sowie alle auf den Blimp einwirkenden Kräfte und Momente

Im Regelfall wird angenommen, daß gilt $F_{auftrieb} = -F_{gravition}$, d.h. der Blimp schwebt in konstanter Höhe.

- **Luftwiderstand (Translation)**

$$F_{Luftwiderstand} = \frac{c_w}{2} \cdot \rho \cdot v^2 \cdot A,$$

wobei ρ die Dichte der Luft (20°C, 0m ü. NN), v die Geschwindigkeit des Blimps, A die projizierte Frontfläche und c_w der Strömungswiderstandskoeffizient ist. Der Luftwiderstand bezeichnet die Kraft, die auftritt, wenn strömende Luft auf einen Gegenstand trifft bzw. sich ein Gegenstand durch ruhende Luft bewegt.
Die Projektionsfläche A des Blimps hängt von dessen Bewegungsrichtung ab; in x-Richtung ist die projizierte Frontfläche am kleinsten (kreisförmig), in y- und z-Richtung die resultierende Widerstandskraft somit größer. Die Gondel kann, wegen ihrer geringen Größe relativ zur Hülle, ignoriert werden. Damit ergibt sich folgende lineare Interpolation:

$$A = \frac{|v_x|}{\|v\|_1} \cdot A_x + \left(\frac{|v_y| + |v_z|}{\|v\|_1} \right) \cdot A_y,$$

wobei $A_x = r_{hull}^2 \cdot \pi$ und $A_y = A_z = r_{hull}^2 \cdot \pi + 2 \cdot r_{hull} \cdot (l_{blimp} - 2 \cdot r_{hull})$ ist.
Der Strömungswiderstandskoeffizient c_w ist ein dimensionsloses Maß, das den formbe-

dingten Widerstand eines von einem Fluid umströmten Körper beschreibt. Er wird üblicherweise experimentell im Windkanal ermittelt. Für unser Modell verwenden wir einen c_w-Wert von $0,48$ (ähnlich dem eines abgerundeten Zylinders).

- **Gondelmotoren**

$$F_{gondel1} = f_g(u_{gondel1}) \cdot [\cos(\mu),\ 0,\ -\sin(\mu)]^T,$$

$$F_{gondel2} = f_g(u_{gondel2}) \cdot [\cos(\mu),\ 0,\ -\sin(\mu)]^T,$$

wobei μ die vom Servomotor bewirkte Winkelstellung der beiden Motoren angibt und $f_g : \{-127 \dots 128\} \rightarrow \mathbb{R}$ die Motorbefehle auf die jeweils resultierenden Kräfte abbildet. Diese stückweise lineare Funktion (siehe Abbildung 4.3) interpoliert die in einem Versuch mittels eines Kraftmessers gewonnenen Messdaten. Die beiden Antriebskräfte wirken relativ zum Massenschwerpunkt in den Punkten $[0.0,\ -\frac{r_{hull}}{2},\ 0.0]^T$ bzw. $[0.0,\ +\frac{r_{hull}}{2},\ 0.0]^T$, also genau dort, wo sich im Modell auch die tatsächlich Motoren befinden.
Zwar ist es möglich, den Blimp mittels unterschiedlich starker Schubkommandos der beiden Gondelmotoren um die z-Achse drehen zu lassen, doch werden aus Gründen der Einfachheit bis jetzt beide Motoren zu jedem Zeitpunkt gleichstark betrieben, d.h. es gilt: $F_{gondel1} = F_{gondel2}$. Allein dem Heckmotor ist es somit vorbehalten, das Luftschiff um die z-Achse zu rotieren.

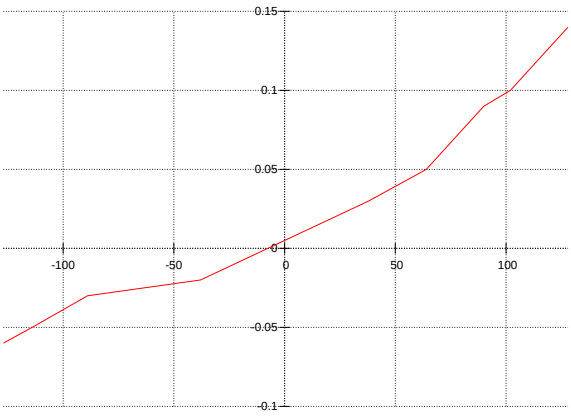

ABBILDUNG 4.3: Motorenkommandos werden durch die approximierte Funktion f_g auf Kräfte abgebildet.

- **Heckmotor**

$$F_{heck} = f_g(u_{heck}) \cdot [0,\ -1,\ 0]^T,$$

wobei f_g die Kraftfunktion ist (siehe Abbildung 4.3).
Der in der Heckflosse befindliche Motor entfaltet dieselbe Kraft wie die Gondelmotoren. Er wirkt jedoch nicht in x- sondern in y-Richtung, und zwar im Punkt $[-\frac{l_{blimp}}{2},\ 0.0,\ 0.0]^T$.

Jede der o.g., dem Modell hinzugefügten, Kräfte verursacht ein Drehmoment um den Massenschwerpunkt: $Drehmoment_i = r_i \times F_i$. Dabei ist r_i ein Abstandsvektor zwischen dem Massenschwerpunkt und dem Punkt, in dem die i-te Kraft Anwendung findet. Der einzig neue Term ist der Rotationswiderstand, der als reines Moment modelliert wird:

- **Luftwiderstand (Rotation)**

  $$M_{Luftwiderstand} = -C_r \cdot \omega,$$

  wobei C_r ein dimensionsloser Widerstandskoeffizient und ω die Winkelgeschwindigkeit des Blimps ist.

Das finale Bewegungsmodell ergibt sich so zu (siehe [6]):

$$\dot{s} = \frac{d}{dt}\begin{bmatrix} \mathrm{p} \\ \xi \\ \mathrm{v} \\ \omega \end{bmatrix} = \begin{bmatrix} R_b^e \\ H(\xi) \\ m^{-1}(\sum F - \omega \times mv) \\ J^{-1}(\sum D - \omega \times J\omega) \end{bmatrix},$$

wobei die Rotationsmatrix R_b^e Vektoren vom körperfesten Koordinatensystem R ins starre Weltkoordinatensystem R_0 überführt, H die Euler'sche kinematische Matrix, m die Massenmatrix und J die Matrix der Trägheitsmomente ist. Die Kräfte F sowie Drehmomente D setzen sich aus den oben beschriebenen zusammen.

4.2 Implementierung

Zur Implementierung des physikalischen Modells werden die Fähigkeiten der von Gazebo benutzten Physik-Engine ODE ausgeschöpft. Jedes in der Gazebo-Welt agierende Modell (siehe Kapitel 3) wird mittels einer geometrischen Repräsentation und den damit verbundenen Massen initialisiert. Die Grundkörper Kugel, Zylinder und Quader modellieren den Blimp (siehe Abbildung 4.4). Die benötigten Maße wurden vorher dem echten Luftschiff entnommen.

Die Gravitationskraft, die in der Gazebo-Welt auf den Blimp wirkt, wird deaktiviert, um sie wie auch alle anderen Kräfte explizit anzuwenden und so eine größtmögliche Modellierungsfreiheit zu sichern. ODE stellt Funktionen zur Anwendung von Kräften und Drehmomenten auf feste Körper zu Verfügung und unterscheidet dabei auch zwischen körperbezogenen Koordinaten und starren Weltkoordinaten. Die o.g. Kräfte und Momente können somit mit Hilfe der ODE-API problemlos dem Modell hinzugefügt werden.

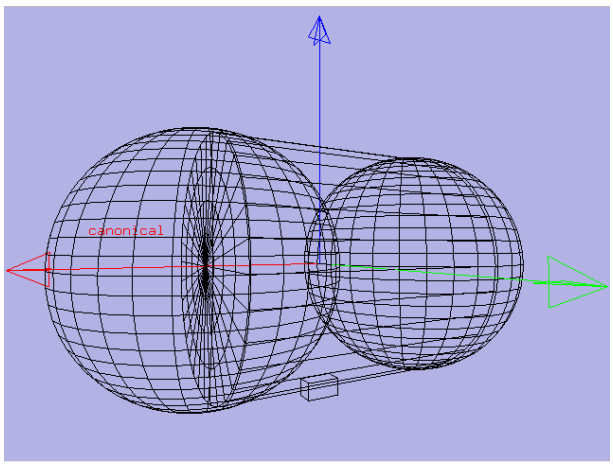

ABBILDUNG 4.4: Zwei Kugeln und ein Zylinder bilden die Hülle des Blimps. Die Gondel wird mittels einem angefügten Quader dargestellt.

Neben den schon genannten Kräften wäre es weiterhin denkbar, eine Windkraft zufälliger Orientierung und Länge zu generieren, die im Außenbereich auftretende Windböen simuliert. Da der Blimp bisher jedoch nur in geschlossenen Räumen eingesetzt wird, in denen solche Phänomene nur bedingt auftreten, ist dies noch nicht realisiert.

5 Sonarsensormodell

5.1 Funktionsweise des Sonars

Für die Höhenmessung wird ein gen Erdboden gerichteter, aktiver Sonarsensor verwendet, der im Gondelboden des Blimps eingelassen ist (Mini-Ultraschall-Entfernungssensormodul SRF10, siehe [17]).
Ein aktives Sonar (*SOund Navigation And Ranging*) misst die Distanz zu einem Objekt, indem es einen Schallimpuls aussendet und dessen Reflexion wahrnimmt. Die während dieses Vorgangs verstrichene Zeit verhält sich proportional zum Abstand des reflexionsverursachenden Objekts. Mit Hilfe der Schallgeschwindigkeit kann diese Entfernung berechnet werden. Das Problem einer Sonarmessung liegt in ihrer fehlenden Präzision. Im Gegensatz zum Laser ist mit dem Sonar keine punktgenaue Messung möglich; nur der Abstand zum nächsten größeren Objekt innerhalb des Schallkegels kann ermittelt werden.

Der *Amplifier* ist eine Einstellung des Sonarsensors, die erheblichen Einfluß auf die Meßgenauigkeit und maximal meßbare Reichweite hat. Ein höherer Amplifier läßt die maximal meßbare Distanz ansteigen, während die Genauigkeit der Messung sinkt. Unter Meßgenauigkeit soll hier die Auswirkung reflektierender Objekte in der näheren Umgebung des Sensors, respektive der Durchmesser des Abstrahlkegels, verstanden werden. Vergrößert sich der Kegeldurchmesser, so erhöht sich auch die Wahrscheinlichkeit, dass Objekte, die nicht unmittelbar lotrecht unter dem Blimp, jedoch näher als der Erdboden, platziert sind, den Schallimpuls reflektieren und somit Eingang in die Messung finden. Das im folgenden Abschnitt beschriebene Experiment lässt darauf schließen, dass der Kegeldurchmesser in Abhängigkeit des Amplifiers variiert.

5.2 Strahlenmodell

Um ein möglichst adäquates Sonarmodell zu erstellen, wurden zunächst in einer Versuchsreihe zahlreiche Messwerte aufgenommen. Dabei wurde der Blimp gemäß Abbildung 5.1 in fünf unterschiedlichen Höhen und drei Abständen zu einem Hindernis positioniert. Zusätzlich wurde der $amplifier \in \{1 \ldots 16\}$ des Sonars in jeder Position variiert, d.h. stichprobenartig Werte für $amplifier \in \{1, 3, 6, 9, 12, 15\}$ evaluiert.

Aus den so gewonnenen Daten der drei unabhängigen Variablen Höhe, Abstand zum Hindernis und Amplifier wird eine Verteilung ersichtlich (siehe Abbildung 5.2), die sich vom herkömmli-

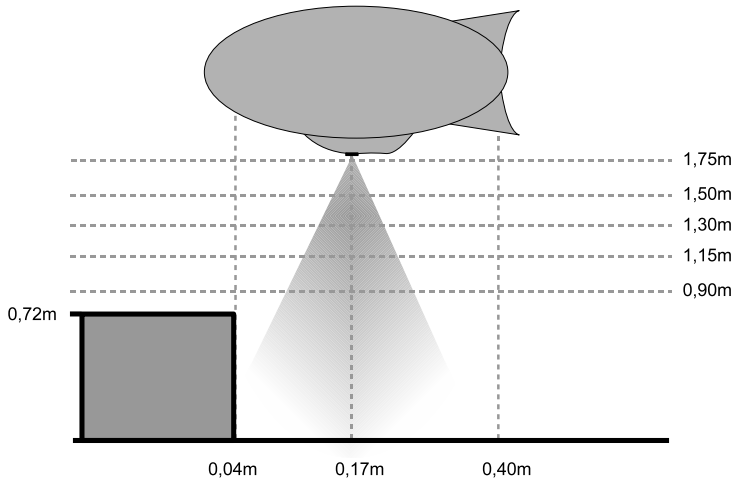

ABBILDUNG 5.1: Um die Auswirkung des Amplifiers auf den Abstrahlkegel des Sonarsensors sowie dessen sonstige Messeigenschaften in einem Modell zu repräsentieren, werden zunächst in einem Versuch Daten gesammelt.

chen Strahlenmodell eines aktiven Sensors nach [19] nur geringfügig unterscheidet. Zu diesen Punkten gehören:

- eine mit zunehmendem Amplifier erhöhte Wahrscheinlichkeit von Messungen $\approx 0,33m$
- keine Verteilung p_{short}, die sonst das wahrscheinliche Kreuzen eines Menschen mit dem Sensorstrahl repräsentiert
- ein vernachlässigbares, uniformes Hintergrundrauschen
- eine mit zunehmendem Amplifier steigende Wahrscheinlichkeit von Messungen umliegender, reflektierender Objekte

Unser Sonarmodell beinhaltet also vier essentielle Bestandteile: eine kleine Messungenauigkeit, Fehler aufgrund von Objekten innerhalb des Abstrahlkegels des Sonarsensors, Fehler aufgrund der Unfähigkeit Objekte zu erkennen und einem dem Sensor anhaftender, unerklärter Fehler. Alle vier Wahrscheinlichkeitsdichten ergeben zusammen das gewünschte Modell $P(z|x, amp)$, d.h. die Wahrscheinlichkeit für eine gemessene Entfernung z bei gegebener tatsächlicher Entfernung x und Amplifier amp (siehe Abbildung 5.3).
Die verschiedenen, jeweils korrespondierenden Fehlertypen im Detail:

- Korrekter Abstand mit lokalem Meßrauschen:
  Der „wahre“ Abstand z^* entspricht der Länge der Sensornormalen zum nächsten Objekt. Dieser Meßwert wird durch die begrenzte Auflösung, atmosphärische Einflüsse oder ähnlichem gestört. Das Messrauschen wird mittels der Gauß-Verteilung
  $p_{hit} \sim \mathcal{N}(z^*, \sigma^2_{hit})$
  mit Mittelwert z^* und geringer Standardabweichung σ^2_{hit} modelliert.

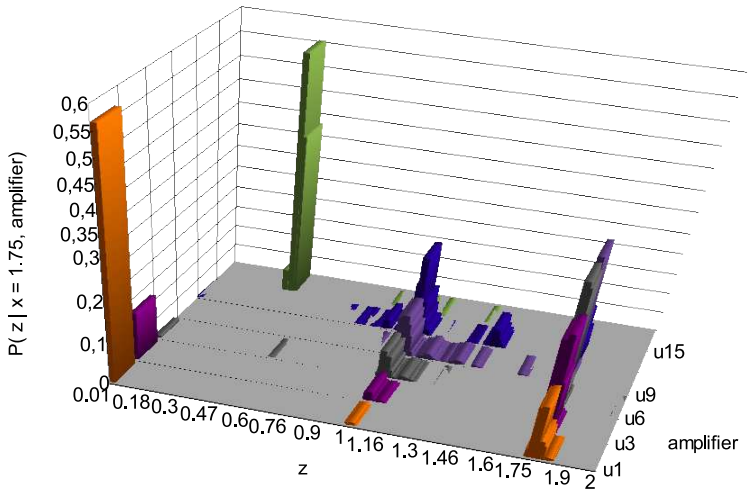

ABBILDUNG 5.2: Die Verteilung einiger aufgenommener Messungen aus einer Höhe von $x = 1,75m$, inklusive aller Abstände zum Hindernis. Aus diesen Daten lassen sich deutlich die vier Hauptkomponenten der Verteilung erkennen: Die max-range-Messungen $z^{max} = 0m$ bei niedrigem Amplifier, die unerklärbaren Messungen um $z^{strange} \approx 0,33m$ bei einem hohen Amplifier, sowie die korrekte Abstandsmessung $z^* = x = 1,75m$ und die Reflexion des Hindernisses bei $z^{min} = (x - 0,72m) = 1,03m$.

- Unerwartete Reflexionen umliegender Objekte:
  Die im Experiment gewonnenen Messdaten zeigen, dass mit ansteigendem Amplifier die Wahrscheinlichkeit für die Messung des Hindernisses anstatt des Bodens steigt. Im Allgemeinen wird also ein näher liegendes Objekt im äußeren Bereich, des vom Sonarsensor aufgespannten Abstrahlkegels, um so eher registriert, je höher der Amplifier ist.
  Wir bezeichnen mit z^{min} den im Abstrahlkegel kürzesten Abstand zu einem Objekt. Wir modellieren dieses Phänomen mit einer Gauß-Verteilung um z^{min}:
  $p_{scatter} \sim \mathcal{N}(z^{min}, \sigma^2_{scatter})$
- Objekte außer Reichweite:
  Es kann vorkommen, dass der Sonarsensor wegen seiner begrenzten Reichweite oder fehlgeleiteten Reflexionen, Objekte nicht erkennt. In solchen Fällen (max-range Messung) liefert er den Wert $z^{max} = 0$. Aufgrund des häufigen Auftretens dieses Fehlers muss er gesondert modelliert werden, und zwar mit Hilfe einer Massepunktverteilung um z^{max}. Da diese Funktion diskret ist, jedoch als Dichte dargestellt werden muss, wird eine Gleichverteilung auf dem Intervall $[-\epsilon, \epsilon]$ verwendet:
  $p_{max} \sim \mathcal{U}(z^{max} - \epsilon, z^{max} + \epsilon)$,
  wobei $\epsilon \in \mathbb{R}$ beliebig klein ist.

- Unerklärte Sensorverhaltensweise:
  Die empirischen Resultate haben gezeigt, dass der hier verwendete Sensor mit hoher Regelmäßigkeit in Abhängigkeit des Amplifier die unerklärbare Messung $z^{strange} \approx 0,33m$ liefert. Diese Messungen werden ebenfalls mit einer Gauß-Verteilung modelliert: $p_{strange} \sim \mathcal{N}(z^{strange}, \sigma^2_{strange})$.

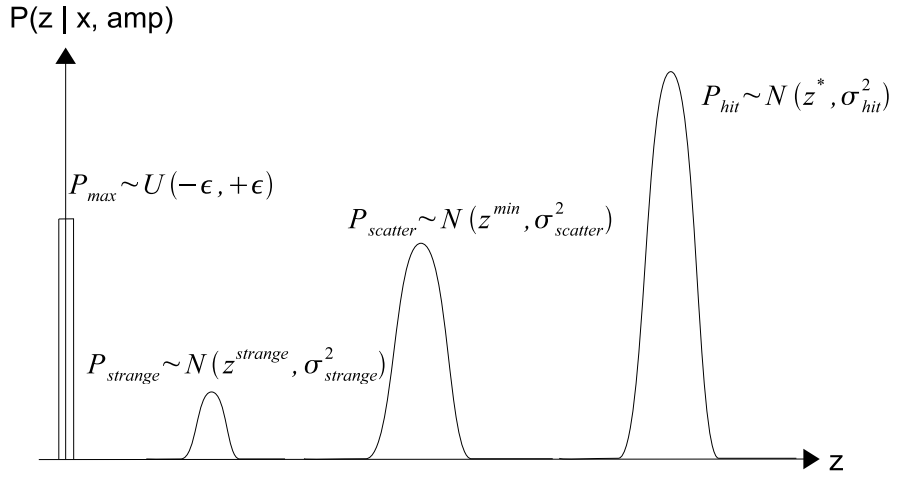

ABBILDUNG 5.3: Die zusammengesetzte Wahrscheinlichkeitsdichte einer Sonarsensormessung z in Höhe $x = z^*$

5.3 Gauß'sche Prozesse

Ausgehend von den aufgenommenen Trainingsdaten sollen nun die vier Wahrscheinlichkeitsverteilungen P_{hit}, $P_{scatter}$, P_{max} und $P_{strange}$ ermittelt werden.
Bei der Bayesschen Regression werden verrauschte Daten $\mathcal{D}$, bestehend aus N Paaren L-dimensionaler Eingangsvektoren $\mathbf{x}$ und skalaren Zielwerten t, $\{\mathbf{x}_n, t_n\}_{n=1}^N$ genutzt, um Vorhersagen für neue, bisher ungesehener Punkte zu machen. In unserem Fall entspricht der Zielwert $t = z$ der zu erwartenden Messung des Sonars und der Eingangsvektor $x = (z^*, amp)$ der aktuellen Höhe und dem Amplifier.
Es soll z.B. die Verteilung von t_{N+1} am Punkt $x_{N+1} \notin \mathcal{D}$, d.h. $p(t_{N+1}|x_{N+1}, \mathcal{D})$, vorausgesagt werden.

Dafür muss folgender Zusammenhang hergestellt werden:

$$t_i = f(x_i) + \epsilon_i,$$

wobei $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ ein normalverteiltes Rauschen ist. Um das Regressionsproblem zu lösen,

wird bei einem Gauß'schen Prozess, anders als bei parametrischen Modellen, jeder Trainingsdatenpunkt direkt im Modell repräsentiert.

Formal gesehen ist ein Gauß'scher Prozess (siehe [15], [22] und [8]) ein stochastischer Prozess, d.h. eine unendliche Sammlung von Zufallsvariablen $\mathbf{t} = (t(x_1), t(x_2), \ldots)$, bei dem jede endliche Teilmenge $(t(x_1), t(x_2), \ldots, t(x_k))$ eine gemeinsame multivariate Gauß-Verteilung mit Kovarianzmatrix C besitzt:

$$P(t|C, \mathbf{x}_n) = \frac{1}{Z} exp\left(-\frac{1}{2}(\mathbf{t}-\mu)^T C^{-1}(\mathbf{t}-\mu)\right),$$

wobei Z eine geeignete Normalisierungskonstante ist.
Der $n+1$-dimensionale Vektor $(f(x_1), \ldots, f(x_n), f(x_{n+1}))$, der den vorherzusagenden Zielwert $t_{n+1} = f(x_{n+1})$ enthält, entspringt somit einer $n+1$-dimensionalen Normalverteilung; damit ist $p(t_{N+1}|x_{N+1}, \mathcal{D})$ eindimensional normalverteilt.

Ein Gauß'scher Prozess kann als Verallgemeinerung einer Gauß-Verteilung über einem endlichen Vektorraum hin zu einem Funktionsraum unendlicher Dimensionalität angesehen werden. Genau wie eine Gauß-Verteilung vollständig durch ihren Mittelwertvektor und Kovarianzmatrix bestimmt ist, ist der Gauß'sche Prozess durch seine Mittelwert- und Kovarianzfunktion bestimmt.

Die Vorhersagen eines Gauß'schen Prozesses hängen also vollständig von der verwendeten Kovarianzfunktion C ab. Mehrere Möglichkeiten stehen bei der Wahl einer geeigneten Kovarianzfunktion zur Verfügung. Die einzige Bedingung ist, dass sie eine nicht-negativ definite Kovarianzmatrix C_{NN} für alle möglichen Inputs $\mathbf{x}_n$ erzeugt, um sicherzustellen, dass die Verteilung $P(t_n|C, \mathbf{x}_n)$ normalisierbar ist. Vom Standpunkt des Modellierens her, möchten wir a-priori-Kovarianzen spezifizieren, die unser a-priori Wissen über die Struktur der zu modellierende Funktion enthalten.

Die verwendete Kovarianzfunktion entspricht einer radialen Basisfunktion und eignet sich in Fällen, in denen domainspezifisches Wissen fehlt:

$$C(x^{(p)}, x^{(q)}; \Theta) = \sigma_f^2 \; exp\left(-\frac{1}{2}\sum_{i=1}^{L} w_i (x_i^{(p)} - x_i^{(q)})^2\right) + \delta_{pq}\sigma_n^2, \tag{5.1}$$

wobei $\Theta = (\sigma_f, w_1, \ldots, w_m, \sigma_n)^T$ der Vektor der Hyperparameter ist. Diese werden später optimiert, um das Modell den Trainingsdaten anzupassen.

Die Kovarianzfunktion ist aus zwei Komponenten zusammengesetzt. Der erste Teile, bestehend aus den Parametern σ_f und w_i, entspricht einer stationären, d.h. translationsinvarianten, radialen Basisfunktion bzw. einer Funktion quadrierter Exponenten. Sie drückt die Idee aus, dass Fälle mit nahen Inputs stark korrelierte, d.h. sehr wahrscheinlich ähnliche Outputs haben sollten. Nah bedeutet hier bezügliche des euklidischen Distanzmaßes und unter Rücksichtnahme auf die Längenmaße. Die Parameter w_i (Längenskalen) werden dafür mit den koordinatenweisen Abständen

im Eingangsraum multipliziert und können so eine unterschiedliche Gewichtung der einzelnen Eingangsdimensionen bewirken. Diese Technik ist im Kontext der Bayes'schen Modellierung als *Automatic Relevance Determination* (ARD) bekannt. Um irrelevante Inputs zu ignorieren wird das entsprechende w_i möglichst klein gewählt. Die charakteristischen Längen der Eingangsdimensionen ergeben sich zu $w_i^{-1/2}$. Wenn ein Parameter w_i groß wird, erhält die resultierende Funktion eine kurze charakteristische Länge und variiert sehr stark entlang der entsprechenden Achse. Dieser Input ist somit sehr wichtig. Die Variable σ_0 gibt die allgemeine vertikale Skalierung der lokalen Korrelationen an, d.h. in Relation zum Mittelwert des Gauß'schen Prozesses im Ausgangsraum.

Der zweite Teil der Kovarianzfunktion in Gleichung 5.1 stellt das Rauschmodell dar. Wir erwarten, dass das Rauschen zufällig ist und nicht mit einzelnen Outputs korreliert. Daher wirkt sich dieser Term nur auf die Diagonalelemente der Kovarianzmatrix aus. Die Varianz des Rauschens ist unabhängig von den Inputs und kann so mit einem einzigen Hyperparameter σ_n beschrieben werden.

Wir haben für unsere Daten ein Modell konstruiert. Nun möchten wir einen konsistenten Weg finden, die noch unbestimmten Hyperparameter Θ aus den Trainingsdaten zu lernen, d.h. so zu belegen, dass die Likelihood der Trainingsdaten maximiert wird. Idealerweise würde man über alle Hyperparameter integrieren, um Vorhersagen zu tätigen:

$$P(t_{N+1}|x_{N+1}, \mathcal{D}, C) = \int P(t_{N+1}|x_{N+1}, \mathcal{D}, C, \Theta) P(\Theta|\mathcal{D}, C)\, d\Theta \tag{5.2}$$

Dies ist für beliebige Kovarianzmatrizen C jedoch analytisch kaum zu bewältigen. Eine Approximation der Verteilung kann man entweder durch die numerische Integration mittels Monte-Carlo-Algorithmen erhalten, oder per Evidenzmaximierung. Wir wählen letzteres, obwohl nur geringe Performanzunterschiede bestehen: für kleinere Trainingsdatenmengen liefert Monte-Carlo bessere Resultate, größere werden vom Evidenzmaximierungsansatz schneller verarbeitet (siehe [14]).

Bei der Evidenzmaximierung wird das Integral 5.2 mit Hilfe der wahrscheinlichsten Menge von Hyperparametern Θ_{MP} genähert:

$$P(t_{N+1}|x_{N+1}, \mathcal{D}, C) \cong P(t_{N+1}|x_{N+1}, \mathcal{D}, C, \Theta_{MP})$$

Die Hyperparameter werden zunächst mit zufälligen Werten initialisiert und anschließend mit einer iterativen Methode, der konjugierten Gradientenoptimierungstechnik, nach einem (lokalen) Maximum der a-posteriori-Wahrscheinlichkeit gesucht. Der Vorteil liegt im geringen Rechenaufwand, denn nur wenige Funktions- und Gradientenevaluierungen sind bei einer geringen Anzahl an Hyperparametern (hier: 4) von Nöten, damit das Verfahren konvergiert. Das Risiko dieses gierigen Verfahrens liegt jedoch im Steckenbleiben in einem schlechten lokalen Maximum. Um dies zu umgehen, finden mehrere Durchläufe mit zufällig initialisierten Hyperparametern statt.

5.4 Implementierung

Für jede der vier Wahrscheinlichkeitsdichten wird ein Gauß'scher Prozess erzeugt und je vier Hyperparameter gelernt. Dieser Schritt erfolgte in Matlab [9] unter Zuhilfenahme des Codes von Rasmussen und Williams [13]. Die gefundenen optimierten Parameterwerte werden zusammen mit den Trainingsdaten verwendet, um die vier Gauß'schen Prozesse (siehe Abbildung 5.4) in C für den Simulator zu implementieren. Dabei ist zu beachten, dass die Anzahl der benutzten Trainingsdaten aufgrund der häufigen Anwendungen der Gauß'schen Prozesse für jede Sonarmessung während der Laufzeit möglichst gering bleibt, aber dennoch nicht an Aussagekraft verliert.

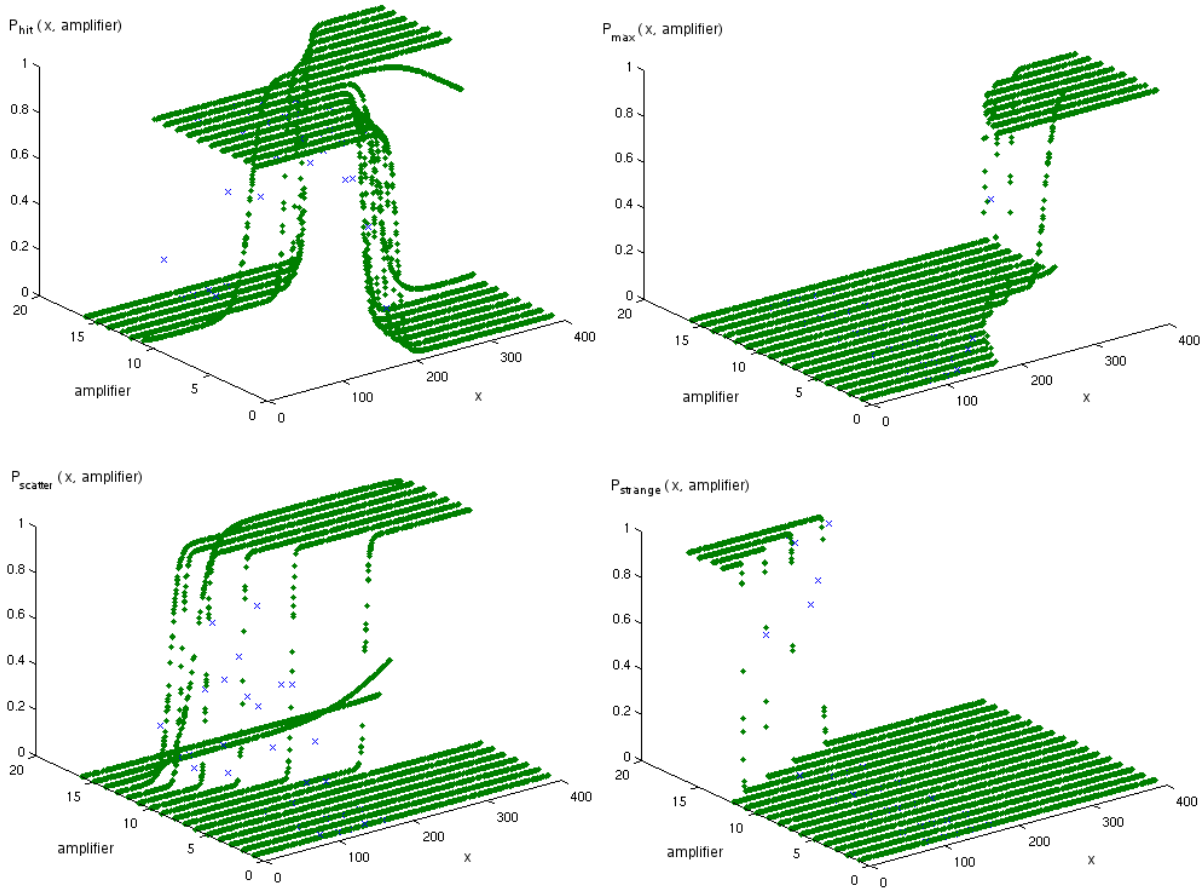

ABBILDUNG 5.4: Die vier einzelnen Gauß'schen Prozesse modellieren jeweils ein Teilphänomen des Sonarsensors.

Für aktive Sensoren wie Laser oder Sonar stellt Gazebo die Klasse *Rayproximity* zur Verfü-

gung. Sie berechnet den Kollisionspunkt mit dem nähesten Objekt entlang eines Strahles in der Gazebo-Welt. Der Abstrahlkegel des Sonars wird mit einer Menge $S = \{s_0, \ldots, s_8\}$ von 9 Strahlen modelliert, die konzentrisch angeordnet sind (siehe Abbildung 5.5). Der Kegeldurchmesser kann nicht während der Laufzeit der Gazebosimulation variiert werden, da die Erzeugung und Eliminierung von Weltobjekten zu einem Zeitpunkt außerhalb der Initialisierungsphase sehr kostenintensiv ist. Die Abhängigkeit des Kegeldurchmessers vom Amplifier wird daher durch die Auswahl der zur Messung herangezogenen Strahlen simuliert.
Während $z^* = s_0$ der Länge des sich im Zentrum befindlichen Strahls entspricht, ergibt sich der Abstand zu einem näheren, die wahre Abstandsmessung interferierenden Hindernis innerhalb des Kegels zu: $z^{min} = argmin_{s \in S} s$.

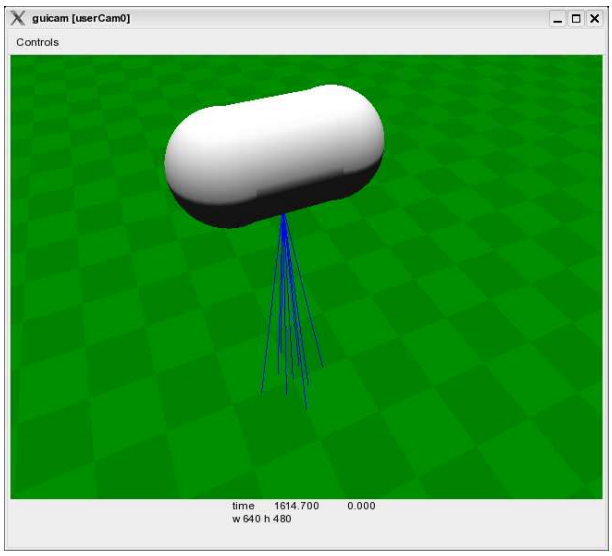

ABBILDUNG 5.5: Der modellierte Abstrahlkegel des Sonars in Gazebo

Das Sonarmodell ermittelt anschließend die Wahrscheinlichkeiten $P_i = GP_i(z^*, amplifier)$, $i \in \{hit, scatter, strange, max\}$ und normalisiert diese zu $\hat{P}_i$. Gemäß dieser Verteilung wird dann der vom Sonarsensor gemessene Wert z bestimmt:

$$z = \begin{cases} z^* & , \text{ falls } rand \in \left[0, \hat{P}_{star}\right) \\ z^{max} = 0 & , \text{ falls } rand \in \left[\hat{P}_{star}, \hat{P}_{star} + \hat{P}_{max}\right) \\ z^{min} & , \text{ falls } rand \in \left[\hat{P}_{star} + \hat{P}_{max}, \hat{P}_{star} + \hat{P}_{max} + \hat{P}_{obs}\right) \\ z^{strange} = 0.33 & , \text{ sonst} \end{cases}$$

wobei $rand \in [0, 1)$ eine uniform verteilte Zufallsvariable ist.

6 Experiment: Reinforcement Learning

6.1 Theoretische Grundlagen

Reinforcement Learning (Bestärkendes Lernen) ist ein Berechnungsansatz, der zielgerichtetes Lernen und Entscheidungsfindung auf der Grundlage von Interaktion betreibt (siehe [18]). Dabei soll optimales Verhalten gelernt werden, d.h. welche Aktion $a \in A$ in welchem Zustand $s \in S$ ausgeführt werden muss, um eine numerisches Belohnung $r(s, a)$ zu maximieren.

Ein Problem des Reinforcement Learning ist eindeutig als Tupel (S, A, δ, r) definiert. S ist die Menge der möglichen Zustände des Systems, A die ausführbaren Aktionen, $\delta : S \times A \rightarrow S$ eine Übergangsfunktion und $r : S \times A \rightarrow \mathcal{R}$ eine Belohnungsfunktion, die jedem Zustands-Aktion-Paar einen numerischen Wert zuordnet und so das zu erlernende Ziel vorgibt. Das Verhalten des Agenten wird mittels einer Funktion, der sog. *policy*, definiert: $\pi : S \rightarrow A$.

Dem lernenden Agenten wird nicht gesagt, welche Aktion er ausführen muss, so wie es beim überwachten Lernen der Fall ist, sondern er muss selbst durch Versuch und Irrtum herausfinden, welche Aktion die höchste Belohnung erzeugt. Ausgeführte Aktionen beeinflussen nicht nur die unmittelbare Belohnung, sondern auch den folgenden Zustand und damit alle folgenden Belohnungen. Diese zwei Charakeristika, Versuch und Irrtum sowie verspätete Belohnung, sind die beiden wichtigsten diskriminierenden Eigenschaften des Reinforcement Learning.

Eine weitere bedeutsame Eigenheit, die aus dem Prinzip des Reinforcement Learning erwächst, ist das Wechselspiel zwischen ausbeuten und auskundschaften (*exploit vs. explore*). Um eine möglichst hohe Belohnung zu erhalten, muss ein Agent diejenigen Aktionen favorisieren, die er in der Vergangenheit schon erprobt hat und die sich als effektiv für die Erzeugung von Belohnung erwiesen haben. Solche Aktionen jedoch zu entdecken, erfordert das Ausprobieren bisher noch ausgelassener Aktionen. Der Agent muss das ausbeuten, was er bereits weiß, um Belohnung zu erlangen, gleichzeitig aber muss er auskundschaften, will er in Zukunft bessere Aktionen selegieren.
Das Dilemma liegt darin, dass weder ausschließlich Ausbeutung noch Auskundschaftung eine Aufgabe erfolgreich lösen. Stattdessen muss der Agent ein Mehrzahl an Aktionen ausprobieren und sukzessive jene bevorzugen, die am besten erscheinen. Bei einer stochastischen Aufgabe muss jede Aktion mehrmals ausprobiert werden, bis eine vertrauenswürdige Schätzung seiner erwarteten Belohnung vorliegt.

Hinzu kommt die beim Reinforcement Learning vorherrschende Tatsache der ganzheitlichen

Betrachtung eines Problems, bei dem der zielgerichtete Agent mit einer ungewissen Umgebung interagiert. Andere Ansätze ziehen nur Teilprobleme in Betracht, ohne sich damit auseinanderzusetzen, wie deren Lösung in einen höheren Zusammenhang eingefügt werden könnte.

Ein verbreiteter Ansatz zur Lösung eines Problems des Reinforcement Learning liegt darin, die totale Belohnung über eine längere Laufzeit zu maximieren. Dabei gilt für die langfristige Belohnung:

$$R^{\pi}(s_t) = \sum_{i=0}^{\inf} \gamma^i t_{t+i}, \tag{6.1}$$

wobei s_t der aktuelle Zustand ist, π die *policy* gemäß derer die Aktionen ausgewählt werden und $\gamma \in [0, 1)$ ein Diskontierungsfaktor, der dafür sorgt, dass weiter in der Zukunft liegende Belohnungen immer schwächer in die Berechnung einfließen.

Die zu erwartende langfristige Belohnung in einem Zustand wird durch die *state-value function* $V : S \rightarrow \mathcal{R}$ ausgedrückt. Sie sagt also, wie gut bzw. ertragreich es für einen Agenten ist, sich im Zustand $s \in S$ zu befinden. Da die zukünftigen Zustände von den Aktionen abhängen, wird auch die *state-value function* in Bezug auf eine *policy* bestimmt:

$$V^{\pi}(s) = E\{R^{\pi} | s_t = s\}.$$

Ähnlich dazu wird die sog. *action-value function* $Q : S \times A \rightarrow \mathcal{R}$ definiert:

$$Q^{\pi}(s, a) = E\{R^{\pi} | s_t = s, a_t = a\},$$

wobei diese die erwartete Belohnung ausgehend vom Zustand s unter Ausführung der Aktion a und anschließendem Befolgen der *policy* π beschreibt.

Beinahe alle Algorithmen des Reinforcement Learning basieren auf der Abschätzung der *state-value* bzw. *action-value function*.

6.2 Anwendung in der Blimpnavigation

Für das Erlernen der *action-value function* bzw. einer optimalen *policy*, die den Blimp auf eine vorgegebenen Höhe h^* stabilisiert, kommt ein Monte-Carlo-Algorithmus zum Einsatz. Die Vorteile der Monte-Carlo-Technik liegen in der Möglichkeit direkt *online* aus der gewonnenen Erfahrung zu lernen, noch während der Agent in der ihm vollkommenen unbekannten Umwelt agiert. Ausserdem ist kein Bewegungsmodell nötig.

Der Zustandsraum S wird mit $s_t = (d_t, v_t)$ definiert, wobei $d_t = h_t - h^*$ den Abstand zur Zielhöhe h^* repräsentiert. h_t und v_t sind die geschätzte Höhe bzw. Geschwindigkeit, die mittels eines Kalmanfilters berechnet werden. Der Aktionsraum A wird durch die maximale Motorleistung begrenzt, so gilt: $a_t \in [a_{min}, a_{max}]$.

Die Belohnungsfunktion $r_t(a_t) = -|d_t|$ ist nur vom aktuellen Abstand zur Zielhöhe abhängig, nicht jedoch von der Geschwindigkeit oder ausgeführten Aktion. Sie ist umso größer, umso näher der Blimp sich zur Zielhöhe befindet.

Ohne ein vorhandenes Modell der Umgebung reicht die *state-value function* nicht aus, um eine optimale *policy* zu bestimmen. Es ist daher notwendig, eine *action-value function* $Q(s, a)$ zu erlernen. Da die vollständige Bestimmung dieser Funktion jedoch sehr zeitaufwändig ist, wird eine Monte-Carlo-Approximation auf Grundlage von Belohnungsstichproben benutzt. Wenn die Anzahl der Besuche eines Zustands-Aktions-Paar (s, a) gegen unendlich strebt, konvergiert die Monte-Carlo-Approximation $Q(s, a)$ gegen die wahre Q-Funktion. Denn jede berechnete langfristige Belohnung ist eine unabhängige, gleichmäßig verteilte Schätzung für $Q(s, a)$. Aufgrund des Gesetzes der großen Zahlen konvergiert die Reihe der Mittlungen dieser Schätzungen gegen ihre Erwartungswerte. Die Standardabweichung des Fehlers jeder gemittelten Schätzung sinkt mit $\frac{1}{\sqrt{n}}$, wobei n die Anzahl der gemittelten Werte ist.

Das Lernen erfolgt schrittweise auf Grundlage von Episoden $e_1, \ldots, e_N$. Eine Episode $e_n = ((s_1, a_1), \ldots, (s_T, a_T))$ ist eine Sequenz von Zustand-Aktions-Paaren, die unter Einhaltung einer gegebenen *policy* ausgewählt wurden. Hier wird ein $\epsilon - greedy$ Ansatz verwendet, der dafür sorgt, dass im Gegensatz zu einem deterministischen π sowohl Auskundschaftung als auch Ausbeutung stattfindet. Dabei wird im Zustand s zum Zeitpunkt t mit geringer Wahrscheinlichkeit $\epsilon \in [0, 1)$ eine zufällige Aktion a_t gewählt und mit Wahrscheinlichkeit $(1 - \epsilon)$ die derzeit ertragreichste Aktion: $a_t = \operatorname{argmax}_a Q(s, a)$.
Das Ende einer Episode tritt ein, wenn entweder ein vordefinierter Zielzustand erreicht oder eine maximale Episodenlänge überschritten wurde. Beim Höhenlernen müssen dynamische Episoden erstellt werden, während der Blimp in der Umgebung agiert. Deshalb werden nur die p Nachfolger des aktuellen Zustands s_t in Betracht gezogen, wobei die Episode beendet ist, sobald für den Diskontierungsfaktor aus Gleichung 6.1 gilt: $\gamma^p < \theta$, wobei θ ein gegebener Schwellwert ist.

Nach der Auswertung der *policy* anhand einer Episode wird die Q-Funktion für jedes auftretende Zustand-Aktions-Paar ausgewertet: $Q(s, a) = R(\bar{s}, a)$, wobei $R(\bar{s}, a)$ die gemittelte langfristige Belohnung ist (*every-visit Monte-Carlo*: jedes Auftreten eines Zustands s innerhalb einer Episode wird berücksichtigt). Anschließend generiert die so verbesserte *policy* π wieder eine neue Episode e_n und der Kreislauf beginnt von vorn.

Die endgültige *policy* π kann somit ausschließlich gierig in Bezug auf die *action-value function* sein:

$$\pi(s) = \operatorname*{argmax}_a Q(s, a)$$

6.3 Experiment im Simulator

Dass das Höhenlernen *online* auf dem echten Blimp funktioniert, wurde bereits in [16] gezeigt. Nun wollen wir diesen Ansatz benutzen, um die Fähigkeiten der Simulation zu evaluieren.

Dazu kommuniziert das Programm zum Reinforcement Learning per IPC mit dem simulierten Blimp (siehe Kapitel 3.1 und Abbildung 3.1) und kann so Aktionskommandos senden sowie die nötigen Sonarsensordaten zum Abschätzen der aktuellen Höhe empfangen.

Nach Abschluß des Lernens im Simulator steht eine policy zur Verfügung (siehe Abbildung 6.1), auf Grundlage derer der Blimp die richtigen Aktionskommandos auswählen kann, um eine vorher angegebene Zielhöhe anzusteuern und zu halten.

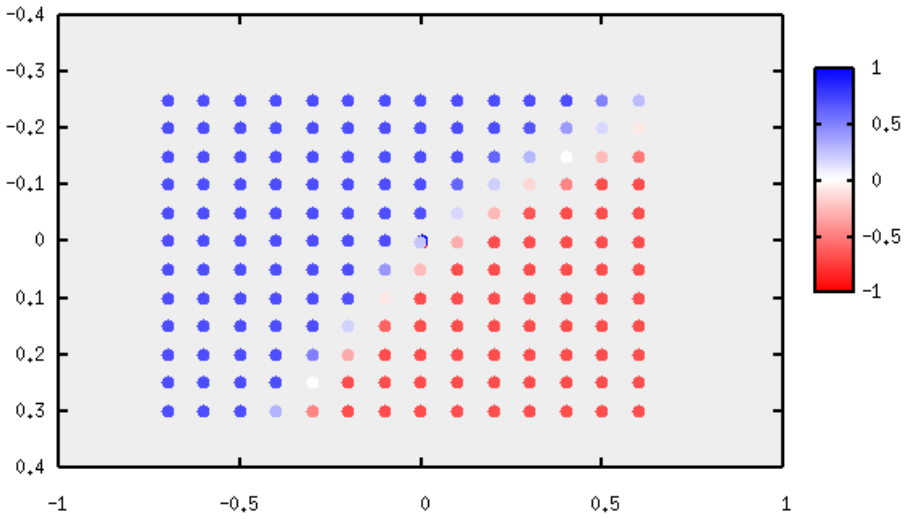

ABBILDUNG 6.1: Die im Simulator gelernte policy: Zu jedem Zustandspaar (x-Achse: Abstand zur Zielhöhe; y-Achse: Geschwindigkeit) gehört eine Aktion (blau: Schub nach oben; rot: Schub nach unten; weiß: kein Schub). Befindet sich der Blimp unter der anzusteuernden Zielhöhe und besitzt zudem eine negative Momentangeschwindigkeit, sorgen die Propeller für einen positiven, d.h. nach oben gerichteten, Schub. Gleiches passiert, wenn die Momentangeschwindigkeit zwar positiv jedoch nicht ausreichend hoch ist oder sich der Blimp überhalb der Zielhöhe befindet, aber mit zu hohem negativen, d.h. nach unten gerichteten Schub, also vorsorglich „abbremsen“ muss. Die blasse, teilweise weiße Diagonale gibt die Fälle wieder, in denen der Blimp die Zielhöhe mit korrekter Geschwindigkeit ansteuert und die Propeller nicht eingreifen müssen, da die Reibungskräfte ihn in der gewünschten Höhe zum Stillstand zwingen.

Die Korrektheit der gelernten policy wurde anschließend in einem schon bestehenden, speziell für die vertikale Navigation des Blimps angefertigten Modell evaluiert (siehe Abbildung 6.2). Dabei zeigt sich, dass der Blimp erfolgreich mit der in der Simulation gelernten policy eine beliebige Höhe anfliegen und halten kann.

Das Experiment zeigt die korrekte Funktionsweise der Blimpsimulation. Der Zeitvorteil im Vergleich zum Lernen auf dem echten Blimp lässt noch komplexeres Verhalten, wie z.B. im 6-dimensionalen Raum (Position und Orientierung), erlernbar erscheinen.

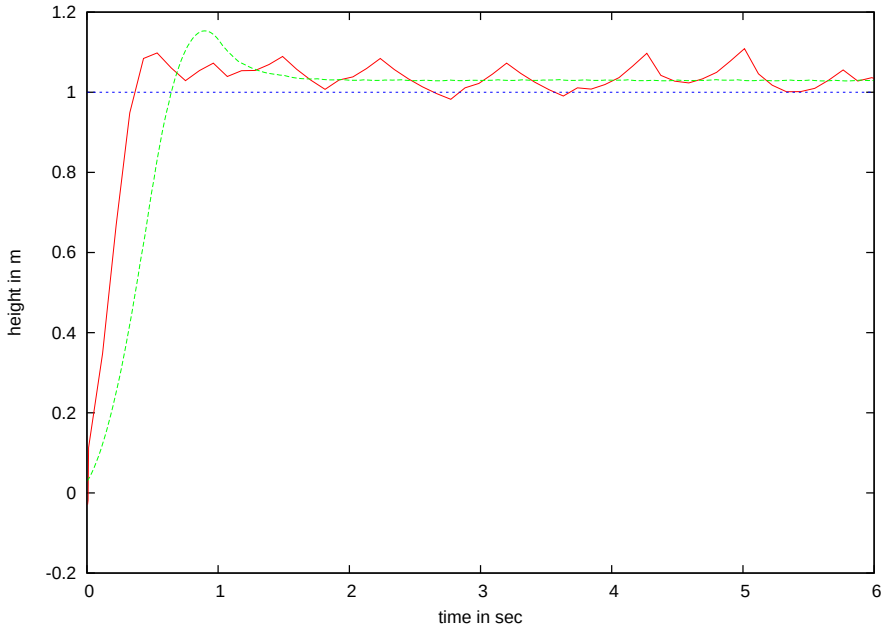

ABBILDUNG 6.2: Evaluation der gelernten policy: Die hier verwendete Zielhöhe von 1.0 m (blau) kann mit Hilfe des gelernten Verhaltens (rot) angesteuert und gehalten werden. Im Vergleich zur optimalen policy (grün), die auf Basis der Parameter des Höhenmodells gelernt wurde, steuert der Blimp die Zielhöhe frühzeitiger an, verhält sich im Anschluß jedoch weniger ruhig.

7 Abschlussbetrachtungen

7.1 Zusammenfassung

Bei der hier vorgestellten Simulation eines Blimps wurden verschiedenste Schwerpunkte gesetzt. Eine offene Architektur erlaubt die einfache Erweiterung des Simulators, sowie eine schnelle Integration schon bestehender Komponenten. Neben einem auf den physikalischen Prinzipien der Luftschiffoperation beruhendem Bewegungsmodell ist die Imitation der unterschiedlichen Sensoren ein Hauptbestandteil der Simulation. Dabei wurde besonders dem Sonarsensor verstärkte Aufmerksamkeit geschenkt und dieser anhand von echten Daten modelliert.

Schließlich wurde die Praxistauglichkeit des Simulators anhand eines Experiment zur Höhenkontrolle mit Hilfe von Reinforcement Learning gezeigt. Hinzu kommt dabei die Stärke der Simulation: Der Zeitaufwand des Lernprozesses ist weitaus geringer als in der echten Umgebung. Des Weiteren lässt sich die Funktionsfähigkeit erstellter Lernsoftware schnell und einfach testen. Verschiedene Lernverfahren können bei immer identisch simulierter Umgebung präzise miteinander verglichen werden.

Während das Lernen im vollständigen 6-dimensionalen Raum auf dem echten Blimp bisher fast unmöglich ist, kann dies in einem nächsten Schritt im Simulator verwirklicht werden.

Dennoch bleibt ein bitterer Beigeschmack: Denn das *online*-Lernen auf dem echten Blimp findet gerade deshalb statt, damit global schwer zu bestimmende äußere Einflussfaktoren wie Ladung, Temperatur oder Windverhältnisse nicht Eingang ins Lernmodell finden müssen. Doch eben solche Variablen werden bisher auch in der Simulation noch nicht repräsentiert, und lassen so die Ergebnisse nur bedingt brauchbar erscheinen.

7.2 Ausblick

Um diesem Problem zu entgehen ist es unausweichlich, das physikalische Blimpmodell weiter zu präzisieren und der Realität näher zu bringen. Beispielsweise könnte der Luftstrom durch zufällig generierte Kraftvektoren, die auf den Blimp einwirken, nachgeahmt werden. Weitere mögliche Erweiterungen beinhalten die Realisierung zusätzlicher Sensoren (z.B. Beschleunigungssensor).

Doch das Bestreben eine Simulation immer realitätsgetreuer zu machen, birgt die Gefahr, den

eigentlichen Sinn und Zweck dieser, nämlich das Einsparen von Ressourcen jeglicher Art (vornehmlich: der Zeit), zu verfehlen. Dieser Fakt bleibt auch Medeen [10] nicht verborgen:

> „Currently, simulations deal with relatively simple sensors, such as sonar, and simple affectors, such as wheels. As the sensory capabilities of robots become richer, it will become increasingly difficult to construct accurate enough simulations. In the near future it may become more time consuming and expensive to build a simulation than it is to just do online learning. Should we abandon simulations at this point?“

Während sich diese Frage für einfache Simulationen wie der in dieser Arbeit präsentierten noch nicht stellt, weil das Verhältnis von Kosten und Nutzen immer noch positiv ist, so werden in Zukunft wohl andere Ansätze, wie beispielsweise das Lernen der Simulation selbst, an Aufmerksamkeit gewinnen.

Literaturverzeichnis

[1] M. Brady and H. Hu. Software and hardware architectures for advanced mobile robots for manufacturing. *Journal of Experimental and Theoretical Artificial Intelligence*, 1997.

[2] Open Dynamics Engine. http://www.ode.org.

[3] Gazebo. http://playerstage.sourceforge.net/gazebo.

[4] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proc. of the Int. Conf. on Advanced Robotics (ICAR)*, Coimbra, Portugal, 2003.

[5] Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav S. Sukhatme, and Maja J. Matarić. Most valuable player: A robot device server for distributed control. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

[6] J. Ko, D. Klein, D. Fox, and D. Hähnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[7] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

[8] David J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, NATO ASI Series, pages 133–166. Kluwer Academic Press, 1998.

[9] Matlab. http://www.mathworks.de.

[10] Lisa Medeen and Douglas Blank. Use of robot simulations can enhance integration. In *Workshop on Integrating Robotics Research, AAAI Spring Symposium Series*, 1998.

[11] Radio Control Models. http://www.rcguys.com, rcguys.

[12] The Player Project. http://playerstage.sourceforge.net.

[13] Rasmussen and Williams. Gpml matlab code, http://www.gaussianprocess.org/gpml/code.

[14] Carl Edward Rasmussen. *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*. PhD thesis, University of Toronto, 1996.

[15] C.E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press,

2006.

[16] A. Rottmann, C. Plagemann, and W. Burgard. Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[17] Ultraschall-Entfernungssensormodul SRF10. http://www.roboter-teile.de/shop/themes/kategorie/detail.php?artikelid=7source=2.

[18] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[19] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[20] S. Varella Gomes and J. Ramos. Airship dynamic modeling for autonomous operation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

[21] Richard T. Vaughan and Brian P. Gerkey. Really reusable robot code and the player/stage project. In Davide Brugali, editor, *Software Engineering for Experimental Robotics*, Springer Tracts on Advanced Robotics. Springer-Verlag, Berlin, 2006.

[22] Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, volume 8. MIT Press, 1995.