Master-Thesis

Regelbasiertes autonomes Verhalten simulierter Schiffe auf Basis der internationalen Kollisionsverhütungsregeln (KVR)

Rule-based autonomous behavior of simulated vessels based on the Conventions on the International Regulations for Preventing Collisions at Sea (COLREGs)

Martin Kroker

Universität Bremen, Fachbereich 3, Postfach 330440, 28334 Bremen, Germany

Datum: 26. Oktober 2014

Erstprüfer:	Prof. Dr.-Ing. Udo Frese
Zweitprüfer:	Dr.-Ing. Frank Dylla

Abstract

This work presents a practical realization of an autonomous vessel control for a simulator. The path planning is implemented in two stages by combining a global and a local planner to join an optimum route calculation — in distance and time — with the strict requirements of the traffic rules. Global path planning determines a route as a sequence of convex polygons based on a *Navigation Mesh* using A* algorithm. Founding on such a route, local path planning is developed as a behavior-based, modular control architecture with rules for collision prevention being represented by different independent modules. For the fusion of module outputs a time-variant potential field is used. Each module provides a set of potentials. Using the potential field, a special module for the combination of rule module outputs plans a path for a particular time interval to detect traps in the potential field early. The resulting path is a sequence of steering commands each applying to a limited period of time. In case of invalid states within the path planned ahead, alternative paths are established using a *Backtracking* algorithm. The quality of the autonomous behavior's results is evaluated based on a set of scenarios. This presents a method for combining a set of rules for autonomous vessel control that can serve as a development platform for path planner and additional sets of rules.

Zusammenfassung

Diese Arbeit präsentiert die praktische Umsetzung einer autonomen Steuerung für Schiffe in einem Simulator. Die Pfadplanung wird zweistufig durch Kombination eines globalen und eines lokalen Planers umgesetzt, um die Berechnung eine optimale Route — in Distanz und Zeit — mit den strikten Anforderungen der Verkehrsregeln zu verbinden. Die globale Pfadplanung wird zur Ermittlung einer Route als Folge konvexer Polygone auf Basis eines *Navigation Mesh* mithilfe eines A*-Algorithmus verwendet. Mit einer solchen Route als Grundlage wird die lokale Planung durch eine verhaltensbasierte, modulare Kontrollarchitektur realisiert, bei der die Kollisionsverhütungsregeln durch unabhängige Module repräsentiert werden. Zur Fusion der Module wird ein zeitvariantes Potentialfeld verwendet. Jedes Modul liefert eine Menge von Potentialquellen. Unter Verwendung des Potentialfelds plant ein spezielles Kombinationsmodul einen Pfad für eine bestimmte Zeitspanne voraus, um Fallen im Potentialfeld frühzeitig zu erkennen. Der als Ergebnis berechnete Pfad ist eine Sequenz von Steuerungskommandos, die jeweils für ein begrenztes Zeitintervall verwendet werden. Treten ungültige Zustände innerhalb des vorausgeplanten Pfades auf, werden durch einen *Backtracking*-Algorithmus alternative Pfade ermittelt. Die Qualität der Ergebnisse des autonomen Verhaltens wird anhand einer Reihe von Szenarien evaluiert. Damit ist eine Methode zur Kombination einer Menge von Regeln, um ein Schiff autonom zu steuern, gegeben, die durch ihren modularen Aufbau als Entwicklungsplattform für Routenplaner und weitere Regelmengen dienen kann.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Master-Thesis selbständig angefertigt habe. Sämtliche verwendeten Unterlagen sind im Literaturverzeichnis aufgeführt.

Ort, Datum

Martin Kroker

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

COLREG Conventions on the International Regulations for Preventing Collisions at Sea

CPA Closest Point of Approach

GUI Graphical User Interface

IF Interface

IMO International Maritime Organization

KVR Kollisionsverhütungsregeln

RRT Rapidly-exploring Random Tree

TSA Traffic Separation Area

UML Unified Modelling Language

WSA Wandering Standpoint Algorithmus

XML Extensible Markup Language

Kapitel 1

Einleitung

Autonome Fahrzeuge gewinnen in der heutigen Gesellschaft immer mehr Relevanz. Ob es sich dabei nun um fliegende Drohnen als Paketzulieferer, autonome Unterwasserfahrzeuge für die Vermessung des Meeresboden oder selbständig fahrende Autos — um nur einige Beispiele zu nennen — handelt, finden sie Einzug in das alltägliche Leben. Von Unterstützungssystemen zur Verbesserung der Sicherheit im Verkehr, über die Erhöhung der Wirtschaftlichkeit von Unternehmen durch Automatisierung, hin zur Steigerung des Lebensstandards durch autonome Fahrzeuge sind die Einsatzmöglichkeiten vielfältig. In eben diesem Hinblick spielt eine automatische Steuerung für Schiffe eine Rolle. So können Unterstützungssysteme, die die geltenden Schifffahrtsregeln beachten, die Sicherheit auf See erhöhen. Selbständig fahrende Frachtschiffe können zusätzlich die Wirtschaftlichkeit entsprechender Unternehmen verbessern. Allerdings können sie nur eingesetzt werden, wenn sie sich korrekt verhalten. Als Beispiel dieser Relevanz sei das aktuelle Projekt *MUNIN* (*Maritime Unmanned Navigation through Intelligence in Networks*, [9]) genannt, das ein Konzept für ein autonomes Schiff entwickelt.

In dieser Arbeit geht es um die Entwicklung eines Verfahrens zur autonomen Steuerung von Schiffen auf Basis einer Menge von Regeln, für die hier als Grundlage die sogenannten Kollisionsverhütungsregeln verwendet werden. Um das gewählte Verfahren testen und evaluieren zu können, wird ein rudimentärer Simulator zur Darstellung und manuellen Steuerung simulierter Schiffe entwickelt, der die implementierte autonome Steuerung einbindet.

In diesem einleitenden Kapitel soll zunächst die Aufgabenstellung erläutert werden. Danach wird eine kurze Zusammenfassung des Inhalts gegeben, um einen Überblick über die Arbeit zu bieten. Abschließend wird auf die Umgebung der Software-Entwicklung sowie den strukturellen Aufbau des Dokuments eingegangen.

1.1 Aufgabenstellung

Eine mögliche Anwendung des vorgestellten Schiffsführungsalgorithmus liegt in der nautischen Ausbildung. Um eine möglichst gute Ausbildung junger Offiziere ohne Gefährdung ihrer Gesundheit bei relativ geringen Kosten zu gewährleisten, werden Simulatoren eingesetzt, bei denen Fehler nicht zu Schäden an Menschen und realen Schiffen führen. Neben der Steuerung des Schiffs selbst ist aber auch das Verhalten gegenüber anderen Schiffen ein wichtiger Bestandteil der Schulung. Diese anderen Schiffe müssen ebenfalls gesteuert werden, damit sie sich innerhalb der Übung bewegen können. Da der Aufwand einer solchen Steuerung durch einen Ausbilder offensichtlich mit der Anzahl der Schiffe steigt, stellt eine automatische Steuerung eine sinnvolle Ergänzung für einen Simulator dar. Dabei ist im Rahmen der Ausbildung zu beachten, dass sich

eigenständig bewegende Schiffe möglichst realistisch verhalten sollen.

Die Anwendung und Entwicklung von Simulatoren dient jedoch nicht allein Ausbildungszwecken. Sie bieten auch die Möglichkeit, verschiedene Verfahren ohne die Gefahr von Unfällen zu untersuchen und zu evaluieren. Damit unterstützen sie die Entwicklung auf dem Weg zu realen autonom fahrenden Schiffen.

Ähnlich den Regeln im Straßenverkehr gibt es auch für die Schifffahrt Regeln, die beim Verhalten beachtet werden müssen. Hier werden die von der *International Maritime Organization* spezifizierten *Internationalen Regeln von 1972 zur Verhütung von Zusammenstößen auf See* — kurz *Kollisionsverhütungsregeln (KVR)* — verwendet. Danach müssen sich alle Schiffe richten, die auf hoher See oder in darin angrenzenden Gewässern unterwegs sind. Entsprechend ist ein solcher Satz von Regeln als Grundlage für ein möglichst realistisches Verhalten autonom fahrender Schiffe sinnvoll. Da eine Menge von Regeln — nicht zwangsläufig die Kollisionsverhütungsregeln, da diese bereits seit vielen Jahren Bestand haben — aber Änderungen unterworfen sein kann, sollen sie möglichst unabhängig betrachtet werden können.

Der Fokus dieser Arbeit wird aber *nicht auf die Umsetzung der Kollisionsverhütungsregeln* gelegt, sondern auf die *Entwicklung eines Verfahrens zur modularen und voneinander unabhängigen Berücksichtigung einer Menge von Regeln und in der Kombination dieser Regeln zur Ermittlung von Steuerkommandos*. Die Kollisionsverhütungsregeln dienen als exemplarische Regelmenge, um das Verfahren zu realisieren und zu untersuchen. Zusätzlich soll dies in eine globale Planung eingebettet werden, um auch über weite Entfernungen Wege planen zu können. Daraus lassen sich die expliziten Aufgaben ableiten, die hier als Zielsetzung verwendet werden:

1. Es ist ein Simulator zu entwickeln, der als Umgebung zur Untersuchung und Evaluation des entwickelten autonomen Verhaltens dient.
2. Es ist ein geeignetes Verfahren zur modularen Verarbeitung der Kollisionsverhütungsregeln zu entwickeln.
3. Das gewählte Verfahren ist zu implementieren, im Simulator zu integrieren und abschließend zu evaluieren.

1.2 Überblick

Die Kollisionsverhütungsregeln definieren vorgegebene Verhaltensweisen in verschiedenen Situationen. Dazu gehören Beschränkungen in der erlaubten Geschwindigkeit, Vorfahrtsregeln sowie in bestimmten Situationen eine Vorgabe der Art des Ausweichens. Diese Regeln beziehen sich aber ausschließlich auf den Nahbereich eines Schiffs. Daher ist es notwendig, zusätzlich eine globale Planung zu entwickeln, um über weite Entfernungen unter Berücksichtigung von Landmassen bzw. Küstenlinien einen Weg zu einem vorgegebenen Ziel zu finden.

Unter Betrachtung dieser Vorgaben (Planung im Fern- und Nahbereich) wird für das Verfahren ein Kontrollzyklus vorgesehen, der sich in drei sequentielle Schritte aufteilt, die über wohldefinierte Schnittstellen miteinander kommunizieren: Aufgabenplaner, globaler Planer, lokales Verhalten. Der Aufgabenplaner gibt einen globalen Zielort vor. Der globale Planer ermittelt eine Route, auf der das lokale Verhalten unter Berücksichtigung der Kollisionsverhütungsregeln Steuerkommandos ermittelt. Um dem lokalen Verhalten eine flexible Möglichkeit zur Planung zu geben, wird nach Vergleich verschiedener Verfahren — insbesondere der Vergleich zwischen wegpunkt- und freiraumbasiertem Verfahren — das *Navigation Mesh* als Repräsentation des befahrbaren Bereichs der Umgebung gewählt. So kann die Route als eine Folge von konvexen Polygonen angegeben werden.

Das Verfahren zur Berücksichtigung einer gegebenen Regelmenge, i. e. das lokale Verhalten, nutzt ein Potentialfeld und plant in diesem über einen bestimmten Zeitraum voraus. Zur Vermeidung von Fallen im Potentialfeld wird außerdem ein *Backtracking*-Algorithmus verwendet, der alternative Wege ermittelt. Um dies zu realisieren, wird für den lokalen Planer als Ansatz eine verhaltensbasierte Kontrollarchitektur gewählt, die zu einem modularen Aufbau führt. Als Fusion der verschiedenen Regelmodule wird ein spezielles Kombinationsmodul entwickelt, welches ein Potentialfeld nutzt, um die Ausgaben der verschiedenen Regelmodule zu kombinieren. Entsprechend geben die Regelmodule abstoßende und anziehende Potentialquellen vor, die die Planung des zu wählenden Pfades beeinflussen.

1.3 Software-Entwicklung

Im Rahmen dieser Master-Thesis ist die Entwicklung diverser Software-Komponenten notwendig: der Simulator und das autonome Verhalten. Die durchgeführten Arbeiten werden in *C++* implementiert. Dazu werden die folgenden Programme und Bibliotheken verwendet.

- *Visual Studio 2013* (Version 12.0)
- *CMake* (Version 2.8.12.2)
- *Doxygen* (Version 1.8.7)
- *Boost* (Version 1.55.0)
- *Qt* (Version 5.2.1)

1.4 Dokumentaufbau

Es werden zunächst in Kapitel 2 und Kapitel 3 die Grundlagen zum Verständnis dieser Arbeit geschaffen. Kapitel 2 gibt eine Zusammenfassung des Aufbaus der Kollisionsverhütungsregeln (cf. Kapitel 2.1) sowie eine Beschreibung des entwickelten Simulators (cf. Kapitel 2.2). In Kapitel 3 werden Verfahren für verschiedene Bereiche der Navigation von autonomen Fahrzeugen vorgestellt. Dazu gehören die Repräsentation von Karten (cf. Kapitel 3.1), Pfadplanung (cf. Kapitel 3.3), Hindernisvermeidung (cf. Kapitel 3.4) und Kontrollarchitekturen (cf. Kapitel 3.5). Anschließend folgt der Kern dieser Master-Thesis: das autonome Verhalten (cf. Kapitel 4). Dieses Kapitel gliedert sich in zwei Abschnitte: die Entwicklung des Konzepts (cf. Kapitel 4.1) und dessen Umsetzung (cf. Kapitel 4.2). Danach folgt die Evaluation des gewählten Verfahrens (cf. Kapitel 5) und zuletzt eine Schlussbetrachtung der Arbeit (cf. Kapitel 6).

Kapitel 2

Grundlagen

In diesem Kapitel werden die für diese Arbeit notwendigen Grundlagen beschrieben. Zunächst werden die Kollisionsverhütungsregeln (cf. Kapitel 2.1) vorgestellt, die als Grundlage für die Richtlinien des autonomen Verhaltens verwendet werden. Anschließend wird der im Rahmen der Arbeit entwickelte Simulator (cf. Kapitel 2.2) erläutert, der als Testumgebung der Schiffssteuerung dient.

2.1 Kollisionsverhütungsregeln

Die *Internationalen Regeln von 1972 zur Verhütung von Zusammenstößen auf See* (engl. *Conventions on the International Regulations for Preventing Collisions at Sea (COLREGs)*) — kurz *Kollisionsverhütungsregeln (KVR)* — definieren die Regelung des Verkehrs auf hoher See und den damit verbundenen Gewässern. Diese Regeln wurden 1972 von der *International Maritime Organization* (IMO) spezifiziert (cf. [13]). Sie enthalten die folgenden fünf Teile (cf. [2]):

Teil A: Allgemeines. Dieser Teil definiert den Rahmen der Anwendung sowie allgemeine in den Regeln verwendete Begriffe.

Teil B: Ausweich- und Fahrregeln. Dieser Teil definiert, wie Fahrzeuge in verschiedenen Situationen bewegt werden müssen, i. e. die Dynamik.

Teil C: Lichter und Signalkörper. Dieser Teil definiert, wie Lichter geschaltet werden müssen, um den Zustand von Fahrzeugen anzugeben.

Teil D: Schall- und Lichtsignale. Dieser Teil definiert, wie Töne gesendet und Lichter geschaltet werden müssen, um mit anderen Fahrzeugen zu kommunizieren.

Teil E: Befreiungen. Dieser Teil definiert, unter welchen Bedingungen Fahrzeuge von den KVR ausgenommen sind.

In dieser Arbeit soll das autonome Verhalten lediglich die Dynamik von Schiffen berücksichtigen. Daher werden hier die Teile A (Allgemeines) und B (Ausweich- und Fahrregeln) genauer betrachtet.

2.1.1 Teil A: Anwendung

Teil A: Allgemeines der KVR umfasst die Regeln 1 bis 3 und gibt einleitende Richtlinien an. *Regel 1* betrifft die Anwendung der Regeln. Sie „gelten für alle Fahrzeuge auf hoher See und auf

den mit dieser zusammenhängenden, von Seeschiffen befahrbaren Gewässern“([2, S.1]). Weiter wird definiert, dass Sonderregelungen in bestimmten Gebieten, in speziellen Situationen und für bestimmte Fahrzeuge durch die KVR nicht berührt werden. *Regel 2* schränkt die Anwendung der KVR insofern ein, dass die Befolgung der Regeln nicht vor Konsequenzen schützt, sofern eine Gefahr durch Abweichen von den Regeln hätte verhindert werden können. Abschließend definiert *Regel 3* einige der in den folgenden Teilen der KVR verwendeten Begriffe. [2, S.1 ff.]

Zusammengefasst drückt dieser Teil aus, dass die KVR als Grundlage für alle Fahrzeuge auf hoher See und verbundenen Gewässern dienen, aber Sonderregelungen zu weiteren Regeln oder zur Ausnahme von Regeln führen können. Weiterhin stellen sie eher eine Richtlinie denn ein festes Regelwerk dar, von der abgewichen werden kann, wenn dadurch Gefahren und Zusammenstöße vermieden werden. Der Hauptfokus besteht darin, die Schifffahrt sicherer zu machen.

2.1.2 Teil B: Ausweich- und Fahrregeln

Teil B: Ausweich- und Fahrregeln der KVR umfasst die Regeln 4 bis 19 und gibt an, wie sich Fahrzeuge bei der Steuerung ihrer Dynamik in verschiedenen Situationen verhalten sollen. Dieser Teil enthält wiederum drei Abschnitte (cf. [2, S.3 ff.]):

- *Abschnitt 1: Verhalten von Fahrzeugen bei allen Sichtverhältnissen* (Regeln 4 bis 10)
- *Abschnitt 2: Verhalten von Fahrzeugen, die einander in Sicht haben* (Regeln 11 bis 18)
- *Abschnitt 3: Verhalten von Fahrzeugen bei verminderter Sicht* (Regel 19)

Aufgrund der großen Zahl an Regeln sollen diese im Folgenden jeweils mit einer kurzen Beschreibung aufgelistet werden. Es wird zuerst der Inhalt der Regel wiedergegeben und anschließend bei längeren oder im Rahmen der Arbeit umgesetzten Regeln — diese sind durch ein * markiert — eine grobe Zusammenfassung — kursiv formatiert — dargelegt.

Regel 4: Anwendung. Die Regeln von *Abschnitt 1* gelten bei allen Sichtverhältnissen. [2, S.3]

Regel 5: Ausguck. Jedes Fahrzeug muss sich mit allen zur Vefügung stehenden Mitteln einen Überblick über die Lage der Umgebung schaffen. [2, S.3]

Regel 6*: Sichere Geschwindigkeit. Jedes Fahrzeug muss jederzeit mit einer Geschwindigkeit fahren, bei der es geeignete Maßnahmen zur Verhinderung eines Zusammenstoßes treffen sowie innerhalb einer den äußeren Bedingungen entsprechenden Entfernung zum Stehen gebracht werden kann. Einfluss auf diese Geschwindigkeit haben die Sichtverhältnisse, Verkehrsdichte, Manövrierfähigkeit, Hintergrundhelligkeit (bei Nacht), Umweltbedingungen (Wind, Seegang, Strömung) und der Tiefgang im Verhältnis zur Wassertiefe. Speziell bei Fahrzeugen mit Radar hängt die sichere Geschwindigkeit noch von weiteren Einflüssen ab. [2, S.3]

Diese Regel beschränkt die Geschwindigkeit von Fahrzeugen nach oben hin abhängig von der jeweiligen Situation.

Regel 7: Möglichkeit der Gefahr eines Zusammenstoßes. Jedes Fahrzeug muss alle verfügbaren Mittel zur Feststellung einer Gefahr eines Zusammenstoßes heranziehen und im Zweifelsfall von einer solchen Gefahr ausgehen. Dabei sind Folgerungen aus unzureichenden Informationen zu unterlassen. Bei der Feststellung muss berücksichtigt werden, ob sich die Peilung eines sich nähernden Fahrzeugs ändert oder ob es trotz Peilungsänderung aufgrund seiner Dynamik möglicherweise nicht ausweichen kann. [2, S.4]

Nach dieser Regel muss jedes Fahrzeug alle Maßnahmen zur Feststellung der Möglichkeit einer Kollision treffen.

Regel 8: Manöver zur Vermeidung von Zusammenstößen. Manöver zur Vermeidung eines Zusammenstoßes müssen in Übereinstimmung mit den Regeln dieses Teils und, sofern die Umstände es zulassen, rechtzeitig erfolgen. Änderungen von Kurs und/oder Geschwindigkeit bei solchen Manövern sollen optisch oder durch Sensoren schnell erkennbar sein. Bei ausreichend Seeraum und entsprechendem Verkehr ist eine alleinige Kursänderung zur Vermeidung des Nahbereichs vorzuziehen. Ein Manöver muss zu einem sicheren Passierabstand führen und es muss bis zum Abschluss der Passage die Wirksamkeit des Manövers überprüft werden. Sofern notwendig muss das Fahrzeug seine Fahrt mindern oder durch Stoppen oder Rückwärtsgehen gänzlich wegnehmen. Fahrzeuge, die Vorfahrt gewähren, müssen frühzeitig Maßnahmen zur Gewährung der Vorfahrt ergreifen und sowohl Fahrzeuge, die Vorfahrt gewähren, als auch solche, die Vorfahrt haben, müssen bei Manövern zur Vermeidung von Zusammenstößen die Regeln dieses Teils berücksichtigen. [2, S.4]

Bei Manövern zur Vermeidung von Zusammenstößen oder zum Gewähren von Vorfahrt müssen Maßnahmen wie Kurs- und Geschwindigkeitsänderung frühzeitig und klar erkennbar durchgeführt werden. Kursänderungen sind Geschwindigkeitsänderungen vorzuziehen. Bei Manövern zur Vermeidung von Zusammenstößen muss die Einhaltung der Regeln dieses Teils berücksichtigt werden.

Regel 9*: Enge Fahrwasser. Fahrzeuge, die in Richtung eines engen Fahrwassers oder einer Fahrrinne fahren, müssen sich so weit rechts (Steuerbord) wie möglich halten. Dabei müssen Segelfahrzeuge sowie Fahrzeuge von weniger als 20 Meter Länge Schiffen, die nur innerhalb eines solchen engen Fahrwassers fahren können, jederzeit die Vorfahrt gewähren und fischende Fahrzeuge dürfen Fahrzeuge in einem engen Fahrwasser nicht behindern. Enge Fahrwasser dürfen nicht gequert werden, wenn dadurch ein anderes Fahrzeug, das nur in einer Fahrrinne sicher fahren kann, behindert wird. Das Ankern in einem engen Fahrwasser muss — sofern möglich — vermieden werden. [2, S.4 f.]

Fahrzeuge innerhalb eines engen Fahrwassers müssen sich so weit rechts wie möglich halten und abhängig von der Art und Größe von Fahrzeugen müssen diese anderen Fahrzeugen Vorfahrt gewähren.

Regel 10*: Verkehrstrennungsgebiete. Verkehrstrennungsgebiete sind von der Organisation festgelegte Bereiche, in denen der Verkehr nur in eine Richtung fahren darf. Entsprechend muss ein Fahrzeug in diesem Gebiet in die allgemeine Verkehrsrichtung fahren und sich dabei so weit wie möglich von der Trennzone entfernt halten. Ein- und Auslaufen muss in der Regel an den Enden des Einbahnwegs erfolgen. Geschieht dies dennoch von der Seite, muss dabei ein möglichst kleiner Winkel zur allgemeinen Verkehrsrichtung eingehalten werden. Queren des Weges sollte so weit wie möglich vermieden werden. Ist ein Fahrzeug jedoch dazu gezwungen, muss dies möglichst mit der Kielrichtung im rechten Winkel zur Verkehrsrichtung geschehen. Küstenverkehrszonen dürfen von Fahrzeugen nicht benutzt werden, wenn ein angrenzendes Verkehrstrennungsgebiet sicher befahren werden kann. Ausnahmen bestehen für Fahrzeuge mit einer Länge von weniger als 20 Meter, Segelfahrzeuge und fischende Fahrzeuge, wenn ein Fahrzeug von oder zu einem Ort in der Küstenverkehrszone fährt oder zur Abwendung einer unmittelbaren Gefahr. Eine Trennzone darf nicht befahren bzw. eine Trennlinie nicht überfahren werden außer beim Queren, beim Einlaufen oder zur Abwendung einer unmittelbaren Gefahr. Weiterhin dürfen Fahrzeuge im Bereich eines Verkehrstrennungsgebiets nicht Ankern und müssen, falls sie ein solches Gebiet nicht nutzen, von diesem einen möglichst großen Abstand halten. Fischende Fahrzeuge dürfen die Durchfahrt anderer Fahrzeuge und Fahrzeuge mit einer Länge von

weniger als 20 Meter sowie Segelfahrzeuge dürfen die Durchfahrt eines Maschinenfahrzeugs auf dem Einbahnweg nicht behindern. [2, S.5 f.]

Es gibt bestimmte definierte Bereiche, sogenannte Verkehrstrennungsgebiete, in denen Fahrzeuge auf durch eine Trennzone separierten Einbahnwegen fahren. Das Ein- und Auslaufen in einen solchen Einbahnweg und der Aufenthalt innerhalb einer Küstenverkehrszone statt innerhalb eines solchen Bereichs sind beschränkt. Queren eines Verkehrstrennungsgebietes ist möglich, wenn bestimmte Maßnahmen getroffen werden. Außerdem gelten innerhalb dieser Gebiete spezielle Vorfahrtsregeln.

Regel 11: Anwendung. Die Regeln von *Abschnitt 2* gelten für Fahrzeuge, die einander in Sicht haben. [2, S.6]

Regel 12*: Segelfahrzeuge. Bei der Annäherung zweier Segelfahrzeuge mit Gefahr eines Zusammenstoßes muss, falls sie den Wind nicht von derselben Seite haben, das Fahrzeug mit Wind von Backbord dem anderen ausweichen. Falls sie den Wind von derselben Seite haben, muss das luvwärtige Fahrzeug dem leewärtigen ausweichen. Die Luvseite ist hier „diejenige Seite, die dem gesetzten Großsegel gegenüber liegt, auf Rahseglern diejenige Seite, die dem größten gesetzten Schratsegel gegenüberliegt“([2, S.6]). Zusätzlich gilt, falls ein Fahrzeug mit Wind von Backbord bei einem anderen Fahrzeug, das in Luv gesichtet wird, nicht bestimmen kann, von welcher Richtung es den Wind hat, muss es dem anderen ausweichen. [2, S.6]

Bei Annäherung von Segelschiffen wird die Vorfahrt abhängig von der Windrichtung relativ zur Orientierung der beteiligten Schiffe bestimmt.

Regel 13*: Überholen. Beim Überholen muss das überholende Fahrzeug dem überholten ausweichen. Ein Fahrzeug gilt als überholend, „wenn es sich einem anderem aus einer Richtung von mehr als 22,5 Grad achterlicher als querab nähert“([2, S.6], cf. Abb. 2.1). Bei Unsicherheit muss ein Fahrzeug annehmen, dass es ein anderes überholt. Durch Fortschreiten des Überholens bzw. durch eine Peilungsänderung wird das überholende Fahrzeug nicht von seiner Ausweichpflicht entbunden (und wird auch nicht zu einem kreuzenden, cf. Regel 15), bis es das überholte Fahrzeug passiert hat. [2, S.6]

Ein überholendes Fahrzeug muss dem überholten immer ausweichen. Dabei gilt ein Fahrzeug als überholend, wenn es zum überholten eine Peilung von $112,5°$ *bis* $247,5°$ *hat.*

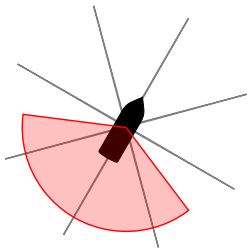


Abbildung 2.1: Der rote Kreisausschnitt zeigt den Bereich, aus dem sich ein Fahrzeug nähern muss, um als überholend zu gelten.

Regel 14*: Entgegengesetzte Kurse. Bei der Annäherung zweier Maschinenfahrzeuge auf annähernd entgegengesetzten Kursen mit Gefahr eines Zusammenstoßes, müssen beide Fahrzeuge ihren Kurs nach Steuerbord so ändern, dass sie beim Vorbeifahren das jeweils andere an Backbord haben. Im Zweifel muss ein Fahrzeug von einer solchen Situation ausgehen. [2, S.6]

Bei Annäherung zweier Maschinenfahrzeuge mit entgegengesetzten Kursen müssen beide Schiffe ihren Kurs jeweils nach Steuerbord ändern.

Regel 15*: Kreuzende Kurse. Im Falle eines möglichen Zusammenstoßes, wenn die Kurse zweier Maschinenfahrzeuge kreuzen, muss das Fahrzeug ausweichen, welches das andere an seiner Steuerbordseite hat. Sofern möglich soll das ausweichende Fahrzeug nicht den Bug des anderen kreuzen. [2, S.6]

Bei Annäherung zweier Maschinenfahrzeuge mit kreuzenden Kursen muss das Schiff ausweichen, welches das andere an seiner Steuerbordseite hat.

Regel 16: Maßnahmen des Ausweichpflichtigen. Ein Fahrzeug, das einem anderen ausweichen muss, i. e. der Ausweichpflichtige, muss möglichst frühzeitig und deutlich erkennbar handeln. [2, S.7]

Regel 17*: Maßnahmen des Kurshalters. Ein Fahrzeug, das Vorfahrt hat, i. e. der Kurshalter, muss Kurs und Geschwindigkeit beibehalten. Im Notfall darf er zur Vermeidung eines Zusammenstoßes selbst handeln, falls der Ausweichpflichtige nicht angemessen agiert. Dabei darf der Kurshalter in einer Situation kreuzender Kurse (cf. Regel 15), sofern es die Umstände zulassen, seinen Kurs gegenüber einem Fahrzeug an seiner Backbordseite nicht nach Backbord ändern. Er muss außerdem handeln, wenn eine Situation besteht, in der Maßnahmen des Ausweichpflichtigen allein zur Kollisionsvermeidung nicht ausreichen. [2, S.7]

Ein Schiff mit Vorfahrt muss dem anderen ausweichen, wenn dieses nicht angemessen agiert oder durch alleinige Manöver einen Unfall nicht verhindern kann. Das Schiff mit Vorfahrt führt dann ein Manöver des letzten Augenblicks durch.

Regel 18*: Verantwortlichkeiten der Fahrzeuge untereinander. Falls nach Regeln 9, 10 oder 13 nichts anderes gilt, müssen Fahrzeuge entsprechend den in Tabelle 2.1 dargestellten Ausweichverpflichtungen handeln. Für tiefgangbehinderte Fahrzeuge gilt die Sonderregel, dass diese nicht behindert werden dürfen. Weitere Sonderregelungen gelten für Wasserflugzeuge und Bodeneffektfahrzeuge. [2, S.7 f.]

Tabelle 2.1 definiert verschiedene Vorfahrtsprioritäten unabhängig von der jeweiligen Situation, ausgenommen solchen in engen Fahrwassern, Verkehrstrennungsgebieten oder beim Überholen.

	Maschine	Segel	fischend	manövrierbehindert	manövrierunfähig
Maschine		×	×	×	×
Segel			×	×	×
fischend				×	×

Tabelle 2.1: Vorfahrtsregeln zwischen verschiedenen Fahrzeugtypen. Bei Feldern mit einem × muss ein fahrendes Fahrzeug vom Typ der zugehörigen Zeile einem Fahrzeug vom Typ der zugehörigen Spalte ausweichen (e. g. ein Maschinenfahrzeug muss Segel-, fischenden, manövrierbehinderten und manövrierunfähigen Fahrzeugen ausweichen).

Abhängig vom Typ zweier beteiligter Schiffe, falls sich diese unterscheiden, sind bestimmte Ausweichpflichten definiert. Manövrierbehinderte oder -unfähige Schiffe haben immer Vorfahrt, maschinengetriebene Schiffe müssen immer ausweichen, Segelschiffe müssen fischenden Schiffen ausweichen.

Regel 19: Verhalten von Fahrzeugen bei verminderter Sicht. Alle Fahrzeuge müssen mit an die Umgebung und Sichtverhältnisse angepasster sicherer Geschwindigkeit fahren.

> Bei Gefahr eines Zusammenstoßes, wobei ein Fahrzeug ein anderes nur durch zusätzliche Sensoren (e. g. Radar) ortet, muss eine Kursänderung nach Backbord vermieden werden, falls das andere Fahrzeug vorlicher als querab — außer beim Überholen — geortet wird, und eine Kursänderung auf ein Fahrzeug zu vermieden werden, falls dieses querab oder achterlicher als querab geortet wird. Wird ein Nebelsignal eines anderen Fahrzeugs von anscheinend vorlicher als querab aufgefasst oder in einer Nahbereichslage mit einem anderen Fahrzeug vorlicher als querab, muss die Fahrt auf ein Mindestmaß zur Erhaltung der Steuerfähigkeit verringert oder gänzlich weggenommen werden. [2, S.8]
>
> *Befindet sich ein Fahrzeug in einer Situation mit Sichtbehinderung, i. e. eine Situation, wo durch die Umgebung die Sicht auf voraus- oder querabliegenede Bereiche versperrt oder durch Umwelt die Sichtweite vermindert ist (e. g. Regen, Nebel), muss es möglichst vorsichtig mit angepasster Geschwindigkeit fahren.*

2.2 Simulator

Um eine repräsentative Umgebung zum Testen und Integrieren der Schiffssteuerung zu erhalten, wird ein Simulator implementiert. Dieser soll eine Simulation der Dynamik von Schiffen und auf sie wirkender äußerer Einflüsse unterstützen. Als Anforderungen werden neben den Grenzen, die durch das Terrain gegeben sind (e. g. Landmassen), auch die im vorigen Kapitel beschriebenen Kollisionsverhütungsregeln (cf. Kapitel 2.1) herangezogen. Im Einzelnen sind dies:

- Küstenlinien und Landmassen
- Schiffe mit ihren geometrischen Abmessungen
- Schiffsdynamik
- Umwelteinfluss auf Dynamik (Strömung und Wind)
- Sichtbehinderung durch Umwelteinfluss
- Verkehrstrennungsgebiete

Um verschiedene Typen von Schiffen repräsentieren und den Einfluss des Schiffstypus auf die Ausführung von Kollisionsverhütungsregeln (cf. Kapitel 2.1) abbilden zu können, müssen einige Parameter variabel gehalten werden. Dies betrifft insbesondere:

- Schiffstyp: Fischerboot, Segelschiff, Maschinenschiff
- Abmessungen
  - Länge
  - Breite
  - Höhe
- Dynamik
  - maximale Geschwindigkeit
  - minimale Geschwindigkeit (beinhaltet auch Rückwärtsgeschwindigkeit)
  - maximale Beschleunigung
  - maximale negative Beschleunigung

– maximale Rotationsgeschwindigkeit

Der Simulator ist ein getaktetes System, das mit einer $10Hz$ Frequenz läuft. Da Schiffe eine — gegenüber zum Beispiel Flugzeugen — geringe Dynamik haben, ist diese Frequenz ausreichend. Dabei werden in jedem Zyklus folgende Schritte ausgeführt (cf. Abb. 2.2):

- Verarbeitung von Kommandos
- Aktualisierung der Position, Orientierung und Geschwindigkeit von Objekten
- Prüfung von Kollisionen der Objekte mit Land und mit anderen Objekten
- Aktualisieren der Simulationsdaten (und damit der am Graphical User Interface (GUI) angezeigten Situation)

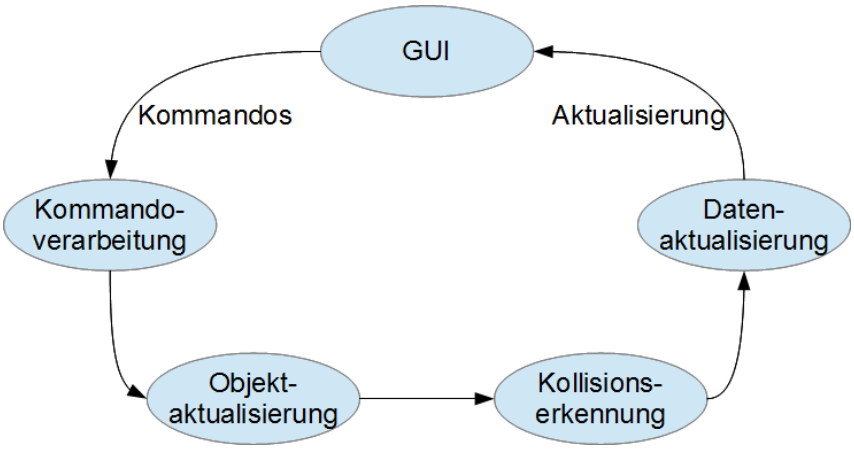


Abbildung 2.2: Übersicht des sequentiellen Ablaufs eines Simulationszyklus, während die Simulation im laufenden Zustand ist.

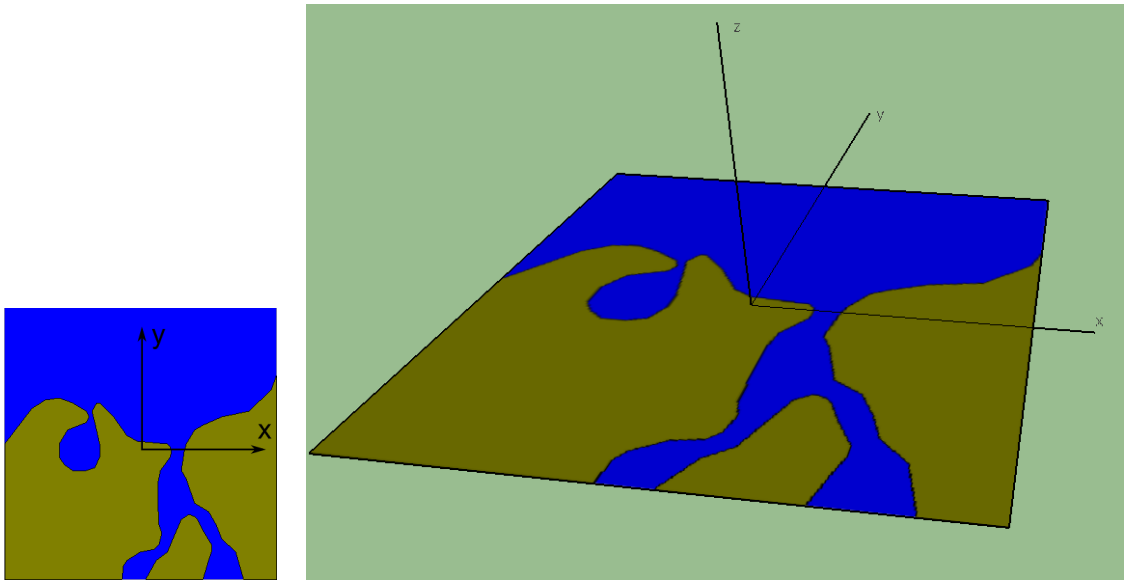
Im Folgenden werden die relevanten Anteile des Simulators genauer beschrieben. Zunächst wird das verwendete Koordinatensystem (cf. Kapitel 2.2.1) dargestellt. Anschließend werden die Repräsentation des Terrains und die Simulation der Umwelt (cf. Kapitel 2.2.2), die Simulation der Schiffsdynamik (cf. Kapitel 2.2.3) und die Kollisionserkennung von Schiffen sowohl mit dem Terrain als auch mit anderen Schiffen (cf. Kapitel 2.2.4) erklärt. Abschließend wird das zusätzliche Programm zur Definition von Szenarien (cf. Kapitel 2.2.5) erläutert.

2.2.1 Koordinatensysteme

Im Simulator werden zwei verschiedene Koordinatensysteme verwendet. Das eine ist ein globales (bzw. Welt-) Koordinatensystem, das andere ein objektrelatives (bzw. Body-) Koordinatensystem. Beide sind rechtshändige kartesische Koordinatensysteme. Das objektrelative Koordinatensystem wird zusätzlich zum globalen benötigt, da einige dynamische Parameter (Geschwindigkeit und Beschleunigung) einfacher relativ zur Orientierung angegeben werden können.

2.2.1.1 Welt

Das Weltkoordinatensystem liegt bei Sicht von oben auf die Karte so, dass die x-Achse nach rechts (bzw. Osten), die y-Achse nach oben (bzw. Norden) und die z-Achse aus der Bildebene



(a) (b)

Abbildung 2.3: Lage des globalen Koordinatensystems: (a) bei Sicht von oben auf die Karte und (b) bei dreidimensionaler Ansicht

heraus zeigt. Der Ursprung des Koordinatensystems liegt im Zentrum der Karte. Abb. 2.3 zeigt die Lage des Systems in der Karte in zwei- (cf. Abb. 2.3a) und in drei-dimensionaler (cf. Abb. 2.3b) Darstellung. Dabei ist $z = 0$ die Wasseroberfläche.

In diesem Koordinatensystem werden Positionen sowie Orientierungen in der Welt eindeutig angegeben (cf. Kapitel 2.2.3).

2.2.1.2 Objektrelativ

Das objektrelative Koordinatensystem liegt gegenüber der Orientierung des Objekts so, dass die x-Achse voraus, die y-Achse nach links und die z-Achse nach oben zeigt. Der Ursprung liegt im Zentrum des Objekts. Abbildung 2.4 stellt die Lage des objektrelativen Koordinatensystems dar.

Dieses Koordinatensystem wird hauptsächlich zur Angabe der Geschwindigkeit und Beschleunigung verwendet. Daneben spielt es aber auch für die Angabe relativer Entfernungen und Peilungen vom Objekt selbst eine Rolle.

2.2.1.3 Posen von Objekten

Als grundlegende Daten zur eindeutigen Bestimmung der Position und Orientierung eines Objekts in der Welt werden die Position in m (Meter) in Weltkoordinaten (cf. Kapitel 2.2.1.1)

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2.1}$$

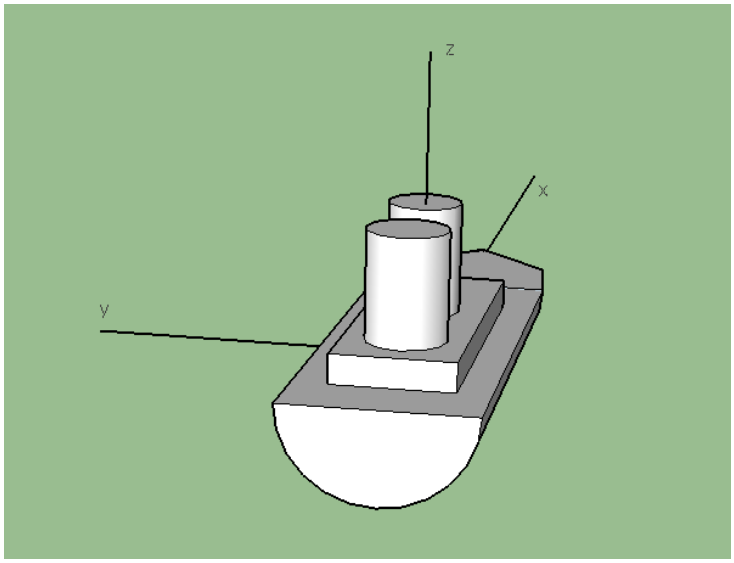


Abbildung 2.4: Lage des objektrelativen Koordinatensystems

und die Orientierung in *rad* (Radiant)

$$\vec{\phi} = \begin{pmatrix} \psi \\ \vartheta \\ \varphi \end{pmatrix} \tag{2.2}$$

angegeben. Dabei werden alle Winkel in mathematisch positiver Richtung angegeben, wobei ψ die Rotation um die x-Achse, ϑ die Rotation um die y-Achse und φ die Rotation um die z-Achse definieren. Dies orientiert sich an den nautischen Begriffen *roll* (ψ), *pitch* (ϑ) und *yaw* (φ).

2.2.1.4 Umrechnung Welt- und objektrelative Koordinaten

Sei $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ein freier Vektor und $\vec{\phi} = \begin{pmatrix} \psi \\ \vartheta \\ \varphi \end{pmatrix}$ die Orientierung eines Objekts in Weltkoordinaten nach Definition des Simulators (cf. Kapitel 2.2.1.3). Die Rotation eines Vektors erfolgt mithilfe einer Rotationsmatrix

$$R_{world \to body}(\vec{\phi}) = R_z(-\varphi)R_y(-\vartheta)R_x(-\psi) \text{ bzw.} \tag{2.3}$$

$$R_{body \to world}(\vec{\phi}) = R_x(\psi)R_y(\vartheta)R_z(\varphi) \tag{2.4}$$

mit

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{pmatrix}, \tag{2.5}$$

$$R_y(\beta) = \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix}, \tag{2.6}$$

$$R_z(\gamma) = \begin{pmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.7}$$

Die Rotation eines Vektors $\vec{x}$ von Weltkoordinaten in objektrelative Koordinaten $\vec{x'}$ erfolgt mit

$$\vec{x'} = R_{world \rightarrow body}(\vec{\phi})\vec{x} \tag{2.8}$$

bzw. eines Vektors $\vec{x'}$ von objektrelativen in Weltkoordinaten mit

$$\vec{x} = R_{world \leftarrow body}(\vec{\phi})\vec{x'}. \tag{2.9}$$

2.2.1.5 Beschränkung auf zwei Dimensionen

Die beschriebenen Koordinatensysteme sind jeweils für drei Dimensionen definiert. Allerdings werden im Rahmen dieser Arbeit lediglich zwei Dimensionen verwendet, i. e. es gilt hier für Positionen $z = 0$ (Wasseroberfläche) und für Orientierungen $\psi = 0$ und $\vartheta = 0$. Dennoch werden die Grundlagen des Simulators so entwickelt, dass eine Wiederverwendbarkeit für andere Objekttypen (e. g. Flugzeuge) und auch eine Erweiterbarkeit für Schiffe ins Dreidimensionale (e. g. Wellensimulation) gegeben sind.

2.2.2 Terrainrepräsentation und Umweltsimulation

Das Übungsgebiet der Simulation ist definiert durch eine Länge l_{area} und einer Breite w_{area} in Metern. Das Gebiet wird als Wasser angenommen. Unterschiedliche Wassertiefen werden ignoriert, da sonst der Tiefgang von Schiffen beachtet werden und die Generierung des *Navigation Mesh* (cf. Kapitel 4.1.1.4) abhängig vom Schiff erfolgen müsste.

Zusätzlich können innerhalb des Übungsgebiets Landmassen modelliert werden. Zur Vereinfachung und Konzentration dieser Arbeit auf die Schiffssteuerung sind Landmassen lediglich durch beliebige Polygone repräsentiert. Da Erhebungen auf dem Land nur bedingt Einfluss auf die Schifffahrt haben — nämlich im Fall von Sichtbehinderungen in Küstennähe bzw. bei Fahrt auf Flüssen — werden diese hier nicht betrachtet.

Abb. 2.5 zeigt die Darstellung einer wie oben beschrieben definierten Karte im Simulator.

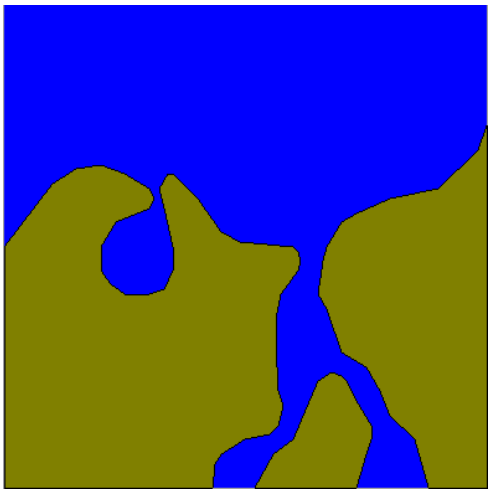


Abbildung 2.5: Karte eines Übungsgebiets

Als Umwelteinflüsse werden Wind, Strömung und Sichtweite umgesetzt (cf. Kapitel 2.2). Dabei werden Wind $\vec{v}_{Wind}$ und Strömung $\vec{v}_{Strom}$ als Geschwindigkeitsvektoren in Weltkoordinaten (cf. Kapitel 2.2.1.1) in $\frac{m}{s}$ angegeben. Die Sichtweite ist stark vereinfacht realisiert. So gibt es diverse Umweltzustände, die einen Einfluss auf die Sichtweite haben können, wie die Tageszeit (e. g. Tag,

Nacht, Dämmerung) oder das Wetter (e. g. Sonne, Nebel, Regen). Da für diese Arbeit nur die Sichtverhältnisse als solche benötigt werden, werden sie durch einen prozentualen Wert zwischen 0% und 100% angegeben, wobei 100% optimale Verhältnisse bedeuten.

Neben diesen natürlichen Bedingungen müssen außerdem aufgrund der zugrunde liegenden Kollisionsverhütungsregeln Verkehrstrennungsgebiete (cf. Kapitel 2.1.2, Regel 10) durch den Simulator bereitgestellt werden. Hier werden die Verkehrstrennungsgebiete auf gerade Wege beschränkt, i. e. es gibt innerhalb eines Verkehrstrennungsgebiets keine Kurven bzw. Knicke. Ein solches Gebiet ist definiert durch eine zentrale Position $\vec{x}_{tsa}$ (*tsa* steht für engl. *traffic separation area*, deutsch *Verkehrstrennungsgebiet*), Winkel φ_{tsa}, Länge l_{tsa}, Breite w_{tsa}, sowie Breite der Trennzone b_{tsa}. Diese Definition ist in Abb. 2.6 dargestellt.

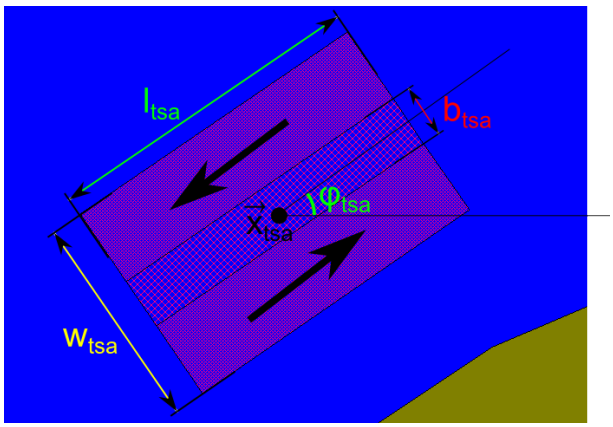


Abbildung 2.6: Definition eines Verkehrstrennungsgebiets im Simulator

2.2.3 Dynamik

Die Simulation der Dynamik basiert nicht auf der Wirkung von Kräften und den daraus resultierenden Beschleunigungen, sondern lediglich auf der Berechnung anhand notwendiger Größen zur Bestimmung der Bewegung von Objekten. Diese werden im Folgenden definiert.

2.2.3.1 Definition der Größen

Zusätzlich zu den in Kapitel 2.2.1.3 definierten Größen werden für die Simulation als dynamische Daten Geschwindigkeit in $\frac{m}{s}$ im objektrelativen Koordinatensystem (cf. Kapitel 2.2.1.2)

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \tag{2.10}$$

Beschleunigung in $\frac{m}{s^2}$ im objektrelativen Koordinatensystem

$$\vec{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \tag{2.11}$$

und Winkelgeschwindigkeit in $\frac{rad}{s}$ in Weltkoordinaten

$$\vec{\omega} = \begin{pmatrix} \omega_\psi \\ \omega_\vartheta \\ \omega_\varphi \end{pmatrix} \tag{2.12}$$

benötigt. Aufgrund der Beschränkung auf zwei Dimensionen (cf. Kapitel 2.2.1.5) kann im Rahmen dieser Arbeit $v_z = 0$, $a_z = 0$, $\omega_\psi = 0$ und $\omega_\vartheta = 0$ angenommen werden. Es wird weiterhin die seitliche Beschleunigung $a_y = 0$ und Geschwindigkeit $v_y = 0$ angenommen, da lediglich Schiffe betrachtet werden sollen, die vorwärts und rückwärts, aber nicht seitwärts beschleunigen können.

Eine Winkelbeschleunigung wird im Rahmen dieser Arbeit der Vereinfachung halber nicht berücksichtigt.

Das hier verwendete Dynamikmodell ist phänomenologisch. Die Schiffe werden als punktförmige Objekte angenommen, sodass Positionen und Rotationen sich immer auf den Mittelpunkt der Schiffe beziehen. Dies ist hier ausreichend, da nicht eine möglichst physikalisch genaue Dynamik, die rechenaufwendig wäre, benötigt wird, um das autonome Verhalten mit Daten zu versorgen und zu evaluieren.

2.2.3.2 Steuerkommando

Das Steuerkommando u wird aufgrund der im Rahmen dieser Arbeit verwendeten Beschränkungen, dass Schiffe lediglich vorwärts und rückwärts beschleunigen sowie durch das Ruder lediglich um die z-Achse drehen können (cf. Kapitel 2.2.3.1), als Tupel

$$u = (v_{target}, a, \varphi_{target}, \omega) \tag{2.13}$$

definiert, wobei v_{target} die zu fahrende Geschwindigkeit und a die zu verwendende Beschleunigung in Richtung x-Achse des objektrelativen Koordinatensystems, sowie φ_{target} die zu erreichende Orientierung und ω die zu verwendende Rotationsgeschwindigkeit um die z-Achse sind.

2.2.3.3 Aktualisierung der Dynamikdaten

Die Dynamik im Simulator wird dann wie folgt berechnet. Seien

- $\vec{a}$ die verwendete Beschleunigung,
- $\vec{\omega}$ die verwendete Winkelgeschwindigkeit,
- $\vec{x}_{t-1}$ die Position vor Aktualisierung der Dynamik,
- $\vec{v}_{t-1}$ die Geschwindigkeit vor Aktualisierung der Dynamik und
- $\vec{\phi}_{t-1}$ die Orientierung vor Aktualisierung der Dynamik.

Sei weiterhin Δt die Zeitdifferenz, für die die Dynamik berechnet werden soll. Dann gilt für die Geschwindigkeit und Orientierung nach der Zeit Δt:

$$\vec{v}_t = \vec{v}_{t-1} + \vec{a} \cdot \Delta t, \tag{2.14}$$
$$\vec{\phi}_t = \vec{\phi}_{t-1} + \vec{\omega} \cdot \Delta t. \tag{2.15}$$

Dabei muss die Geschwindigkeit auf die definierte minimale $\vec{v}_{min}$ und maximale $\vec{v}_{max}$ limitiert werden. Für $i = 1, ..., 3$ mit $v_1 = x$, $v_2 = y$ und $v_3 = z$ gilt

$$v_i = v_{max,i} \quad \text{, falls } v_i > v_{max,i} \text{ und} \tag{2.16}$$
$$v_i = v_{min,i} \quad \text{, falls } v_i < v_{min,i}. \tag{2.17}$$

Gleichung (2.15) gilt nur unter der in Kapitel 2.2.1.5 beschriebenen Beschränkung auf zwei Dimensionen, i. e. $\omega_\psi = 0$ und $\omega_\vartheta = 0$.

Zusätzlich werden für die Berechnung der neuen Position nach der Zeit Δt die Windgeschwindigkeit $\vec{v}_{Wind}$ sowie die Strömungsgeschwindigkeit $\vec{v}_{Strom}$ berücksichtigt. Diese Geschwindigkeiten sind — abweichend von der allgemeinen Definition für Geschwindigkeiten in Kapitel 2.2.3.1 — in Weltkoordinaten angegeben. Um einen individuellen Einfluss dieser Umweltparameter auf verschiedene Objekte zu simulieren, werden zwei Faktoren eingeführt, die bestimmen, wie stark der Wind (Faktor τ_{Wind}) bzw. die Strömung (Faktor τ_{Strom}) auf das Objekt wirken.

Weiterhin werden bei der Berechnung der Position die mittlere Geschwindigkeit $\vec{v}_{mean} = \frac{\vec{v}_{t-1}+\vec{v}_t}{2}$ und die mittlere Orientierung $\vec{\phi}_{mean} = \frac{\vec{\phi}_{t-1}+\vec{\phi}_t}{2}$ verwendet. Abb. 2.7 zeigt, wie anhand dieser Werte die neue Position $\vec{x}_t$ ermittelt werden kann. Dabei wird angenommen, dass sich Objekte innerhalb der Zeit Δt auf Kreisbahnen bewegen — trotz einer möglichen linearen Beschleunigung. Diese Annahme folgt aus der Vereinfachung, dass für kleine Δt gilt $\vec{\phi}_t \approx \vec{\phi}_{t-1}$, da Schiffe realistisch nicht beliebig große Werte für $\vec{\omega}$ besitzen. Daraus folgend gilt außerdem, dass die Länge der Kreisbahn $b = |\vec{v}_{mean} \cdot \Delta t|$ annähernd gleich dem Abstand d der Positionen $\vec{x}_t$ und $\vec{x}_{t-1}$ ist, i. e. $b \approx d$ (cf. Abb. 2.7).

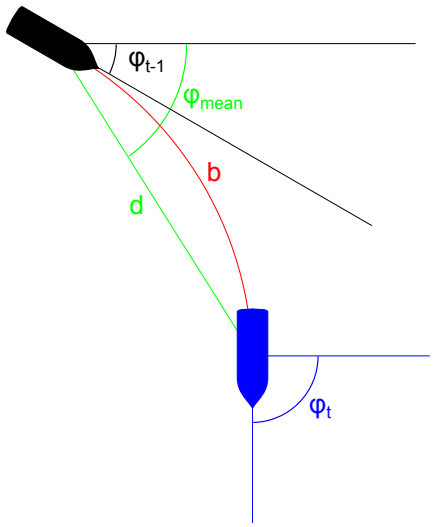


Abbildung 2.7: Darstellung der Positionsaktualisierung im Zweidimensionalen: schwarz ist Position und Orientierung zum Zeitpunkt $t-1$; blau ist die Position und Orientierung zum Zeitpunkt t; rot ist die realistisch zurückgelegte Strecke mit der Länge b; grün ist der Abstand der Positionen d.

Unter diesen Voraussetzungen lässt sich die neue Position $\vec{x}_t$ — ohne Berücksichtigung der Umwelteinflüsse — mithilfe der Rotation der Geschwindigkeit in Weltkoordinaten (cf. Kapitel 2.2.1.4) berechnen nach

$$\vec{x}_t = \vec{x}_{t-1} + R_{body \to world}(\vec{\phi}_{mean}) \cdot \vec{v}_{mean} \cdot \Delta t \tag{2.18}$$

und mit Berücksichtigung der Umwelteinflüsse durch Wind und Strömung nach

$$\vec{x}_t = \vec{x}_{t-1} + (R_{body \to world}(\vec{\phi}_{mean}) \cdot \vec{v}_{mean} + \vec{v}_{Wind} \cdot \tau_{Wind} + \vec{v}_{Strom} \cdot \tau_{Strom}) \cdot \Delta t. \tag{2.19}$$

2.2.4 Kollisionserkennung

Um Kollisionen erkennen und damit mögliche Probleme in der autonomen Schiffssteuerung feststellen zu können, wird eine entsprechende Prüfung implementiert, die im Simulationstakt läuft.

Dabei sind sowohl Kollisionen mit Küsten als auch zwischen Schiffen relevant. Im Falle einer Kollision wird die Dynamik der beteiligten Objekte deaktiviert, damit die problematische Situation unverändert bleibt.

Die Kollisionserkennung wird im Zweidimensionalen durchgeführt. Schiffe werden als durch Länge und Breite sowie ihre Orientierung definierte Rechtecke (*oriented bounding box*) repräsentiert. Eine Kollision eines Schiffs mit einer Küstenlinie wird durch Schnitt der Linie mit diesem Rechteck erkannt. Eine Kollision zwischen zwei Schiffen ist als Überlappung der Rechteckgrenzen definiert.

2.2.5 Szenariodefinition (Programm)

Aufgrund der Notwendigkeit, definierte Übungs- bzw. Testsituationen herzustellen, wird zusätzlich zum Simulator ein Programm entwickelt, welches der Erstellung der Übungsgebiete dient. Hier können das zu verwendende Terrain, die initiale Umwelt und eventuell existierende Verkehrstrennungsgebiete definiert werden (cf. Kapitel 2.2.2).

Außerdem können Objektdefinitionen, die die grundlegenden Eigenschaften wie Typ, geometrische Maße, und dynamische Beschränkungen (cf. Kapitel 2.2.3.1), i. e. minimale und maximale Geschwindigkeit, minimale und maximale Beschleunigung und maximale Winkelgeschwindigkeit, eines Objekts enthalten, erstellt werden. Jedem Objekt in der Simulation muss eine solche Objektdefinition zugrunde liegen.

Auf Basis dieser Objektdefinitionen können Objekte in der Übung angelegt werden. Die Objekte erhalten eine initiale Position, Geschwindigkeit und Orientierung.

Zuletzt kann — und bei Verwendung des autonomen Verhaltens von Schiffen muss — das *Navigation Mesh* (cf. Kapitel 4.1.1.4) definiert werden, indem eine Menge von Polygonen modelliert wird. Dies ist exemplarisch in Abb. 2.8 dargestellt.

Um die so definierte Übung zur Simulation zu übertragen, gibt es einen Speichermechanismus, der das Szenario in einem XML-Format abspeichert, und einen zugehörigen Lademechanismus im Simulator, der die Daten liest und die Übung entsprechend aufbaut.

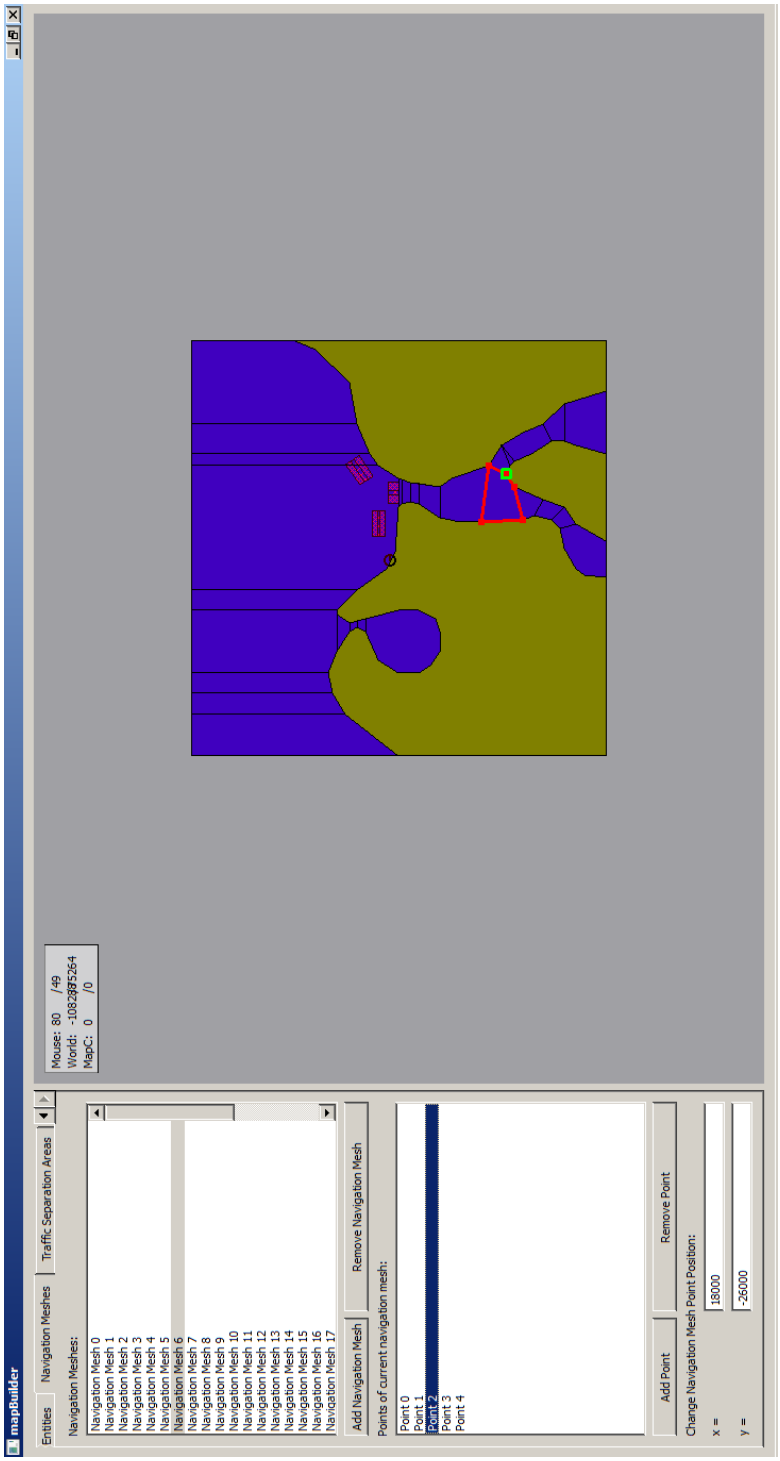


Abbildung 2.8: Modellierung des *Navigation Mesh* bei Szenariodefinition

Kapitel 3

Stand der Technik

In diesem Kapitel werden einige Grundlagen dargestellt, die für die in den folgenden Kapiteln untersuchte und implementierte autonome Steuerung von Schiffen verwendet werden. Da die Steuerung ähnlich derer für einen Roboter ist, entstammen viele der hier vorgestellten Verfahren und Algorithmen den Bereichen Künstliche Intelligenz und Robotik.

3.1 Kartenrepräsentation

Für eine autonome Steuerung eines Objekts ist die Kenntnis der Umgebung, i.e. des Weltmodells, unumgänglich. Sie wird in der Robotik insbesondere zur Lokalisierung und zur Pfadplanung benötigt. Zwar ist Lokalisierung in dieser Arbeit nicht relevant, die Repräsentation der Umgebung — insbesondere des Terrains — ist hingegen grundlegender Bestandteil der Pfadplanung und Kollisionsvermeidung. Daher werden im Folgenden verschiedene Repräsentationen der Umgebung beschrieben, die jeweils Vor- und Nachteile für verschiedene Anwendungszwecke haben.

Zunächst muss zwischen Systemen unterschieden werden, die auf die Welt zentriert sind, und solchen, die auf das Objekt selbst zentriert sind. So lassen sich bestimmte Probleme wie Pfadplanung leichter im weltzentrierten System lösen, andere wie Kollisionsvermeidung leichter im objektzentrierten. [12, S.106]

Zuletzt sei noch angemerkt, dass auch hybride Karten existieren, die verschiedene Umgebungsrepräsentationen miteinander verbinden.

3.1.1 Metrische Karte

Metrische Karten (engl. *metrical maps*) verwenden ein kontinuierliches Koordinatensystem, in dem die Daten entsprechend der Eingabe abgelegt werden. Handelt es sich bei der Eingabe zum Beispiel um Sensordaten, so ist die Genauigkeit der in der Karte abgelegten Daten lediglich abhängig von der Genauigkeit des Sensors. Daten können in dieser Repräsentation sowohl in simpler geometrischer Form (e. g. Punkte) als auch in komplexeren Varianten (e. g. Polygone, Boxen, Kreise) dargestellt werden. Diese Karte repräsentiert die Umgebung möglichst präzise. [12, S.106 f.]

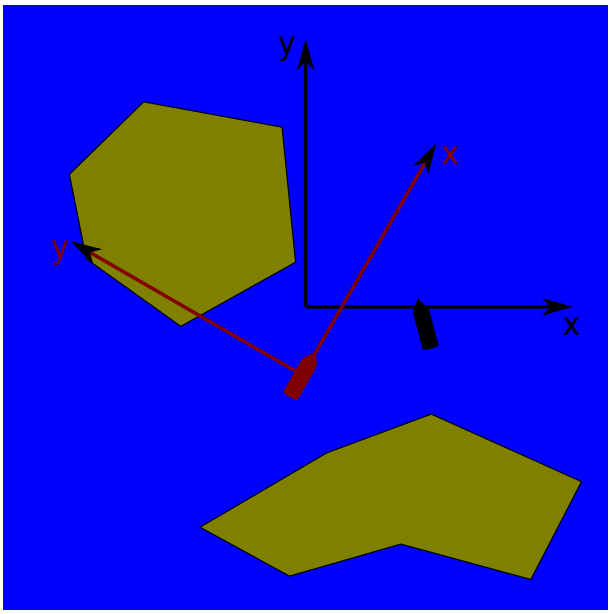


Abbildung 3.1: Umgebungsrepräsentation durch metrische Karte global (schwarzes Koordinatensystem) und objektrelativ (dunkelrotes Koordinatensystem)

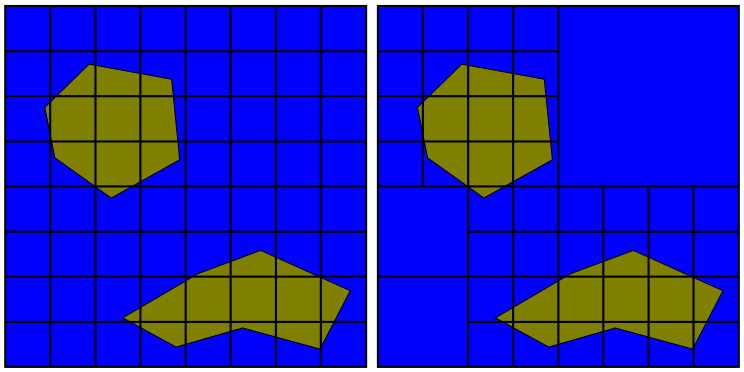
3.1.2 Raster-/Zellkarte

Bei Raster- bzw. Zellkarten (engl. *grid maps*) handelt es sich um eine auf der metrischen Karte (cf. Kapitel 3.1.1) basierenden Repräsentation. Hier wird lediglich das kontinuierliche System in Zellen diskretisiert. Das hat den Vorteil, dass die Repräsentation der Umgebung gegenüber verrauschten Daten robuster wird. Üblicherweise wird die Umgebung in gleichmäßige Zellen unterteilt. Im Zweidimensionalen sind dies für gewöhnlich Quadrate bzw. Rechtecke, im Dreidimensionalen Würfel bzw. Quader, es sind aber auch andere Formen möglich. Die Zellen werden dann mit entsprechenden Zuständen (e. g. frei, Hindernis) belegt, wenn etwas zum Teil in dieser Zelle wahrgenommen wurde. Nimmt also zum Beispiel ein Sensor ein Hindernis zum Teil in der Zelle wahr, wird die gesamte Zelle als Hindernis markiert. [12, S.107, S.126 ff.][8, S.164 ff.]

Ein Nachteil von Rasterkarten ist die Repräsentation eines großen Freiraums bzw. eines großen belegten Raums durch viele als frei bzw. belegt markierte Zellen. Dies lässt sich allerdings dadurch verhindern, indem anstatt vieler kleiner Zellen lediglich eine große Zelle verwendet wird. Dies kann mithilfe des *Quadtree*-Algorithmus (im Dreidimensionalen analog der *Octree*-Algorithmus) erreicht werden. Bei diesem wird mit einer einzelnen rechteckigen Zelle begonnen, die das gesamte Areal umfasst. Anschließend verläuft der Algorithmus rekursiv: Es wird geprüft, ob die Zelle sowohl belegte als auch freie Anteile enthält. Falls ja, wird sie in vier gleichgroße rechteckige Zellen unterteilt, andernfalls wird diese Zelle nicht weiter zerlegt. Hat eine Zelle die Größe der minimalen Rasterauflösung, wird sie nicht weiter zerlegt, sondern als belegt markiert, wenn sie noch einen belegten Anteil enthält. [8, S.165 f.]

3.1.3 Sektorkarte

Sektorkarten (engl. *sector maps*) sind ein Sonderfall der Rasterkarten. Dabei wird die Umgebung in mehrere zumeist polare Sektoren zerlegt. Üblicherweise enthält jeder Sektor Informationen über das nächste Hindernis in dem vom Sektor abgedeckten Bereich, i. e. Winkel und Abstand zum Hindernis. [12, S.129 ff.]



(a) (b)

Abbildung 3.2: Umgebungsrepräsentation durch Rasterkarte: (a) bei regelmäßigem Raster und (b) mithilfe des *Quadtree*-Algorithmus erzeugt.

Diese Umgebungsrepräsentation wird üblicherweise zur Kartierung der Umgebung des Objekts, also als objektzentrierte Karte, verwendet.

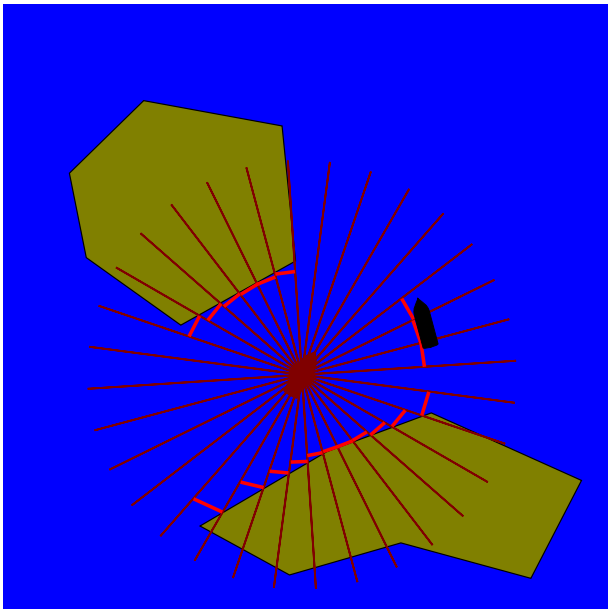


Abbildung 3.3: Umgebungsrepräsentation durch Sektoren relativ zum Objekt; dunkelrot stellt die Sektoreinteilung dar und hellrot eingetragene Hindernisse

3.1.4 Topologische Karte

Topologische Karten (engl. *topological maps*) sind eine abstraktere Darstellung der Umgebung als die zuvor dargestellten metrischen Varianten. Sie werden meistens als Graphen repräsentiert. Dabei entsprechen Knoten im Graph relevanten Orten und Kanten den Verbindungen zwischen Orten. Zusätzlich benötigte Daten wie die Position eines relevanten Ortes oder die Länge einer Verbindung können ebenfalls in der topologischen Karte gespeichert werden. Durch diese Abstraktion können geometrische Aspekte größtenteils unbeachtet bleiben und sie benötigen gegenüber metrischen Kartenvarianten weniger Rechenaufwand. [12, S.107 f., S.134 ff.][8, S.169]

3.2 Navigation

Navigation bezeichnet das Thema der zeitabhängigen Bewegung eines Objekts durch eine vorgegebene Umgebung. Dazu gehört einerseits das Finden eines Weges von einem Start- zu einem Zielpunkt (*„global path planning“*, cf. [12, S.173]), andererseits die Berücksichtigung der geometrischen Abmessungen des Objekts und die kinematischen Grenzen sowie das Ausweichen unvorhergesehener Hindernisse (*„local path planning“*, cf. [12, S.173]). Zusätzlich beschreibt Berns et al. in [12] die Generierung von Steuerkommandos auf Basis von Referenzpunkten (*„path control“*[12, S.173]). [12, S.173][8, S.261 ff.]

Abbildung 3.4 zeigt eine Kette von Modulen verschiedener Abstraktionsebenen zur Steuerung von Aktoren auf Basis der Wahrnehmung der Umgebung. Es wird zwischen vier Modulen mit unterschiedlichen Kontroll-Zyklen, i. e. Frequenzen, in denen geprüft wird, ob auszuführende Pläne bzw. Aktionen geändert werden müssen, differenziert (cf. [8, S.261 ff.]):

Reflexe. Hier werden die Aktuatoren allein auf Basis der Sensorrohdaten angesteuert. Das wohl bekannteste Beispiel für einen Roboter, der lediglich durch Reflexe gesteuert wird, ist das *Braitenberg-Vehikel*. Die Kontroll-Zyklen liegen bei etwa $10Hz$ bis $100Hz$.

Reaktionen. Bei der Steuerung auf Basis von Reaktionen werden fusionierte Sensordaten herangezogen. Die Aktion folgt dann aufgrund dieser Informationen. Die Kontroll-Zyklen liegen bei etwa $1Hz$ bis $10Hz$.

Aufträge. Dieses Modul soll auf Basis einer Situation, i. e. ein Modell der Umgebung, einzelne Aufträge verarbeiten. Dabei stellt jeder Auftrag einen Teil des vom unten beschriebenen Modul generierten Plans dar. Die Kontroll-Zyklen liegen bei etwa $0.1Hz$ bis $1Hz$.

Pläne. Hier werden die übergeordneten Pläne auf Basis des Weltmodells generiert. In diese Komponente fällt das Problem der Pfadplanung, die nicht allein auf Basis von Sensordaten arbeiten kann, sondern eine Karte der Umgebung benötigt, um den kürzesten oder schnellsten Weg zu finden. Die Kontroll-Zyklen liegen bei etwa $0.01Hz$ bis $0.1Hz$.

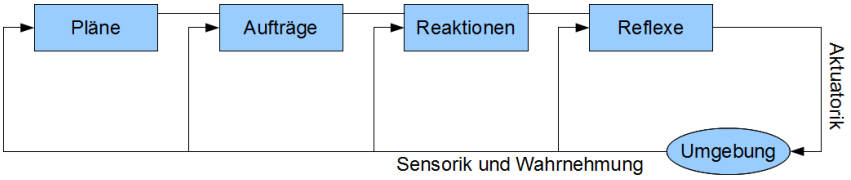


Abbildung 3.4: Kette von Modulen verschiedener Abstraktionsebenen zur Steuerung von Aktoren auf Basis der Wahrnehmung der Umgebung; ungerichtete Kanten stellen eine gegenseitige Beeinflussung dar (nach [8, S.261])

Im Folgenden werden verschiedene Teilprobleme des Bereichs Navigation genauer untersucht. Dies ist zum einen die Pfadplanung (cf. Kapitel 3.3), die oben als *global path planning* bezeichnet wird, und zum anderen die Vermeidung von Hindernissen (cf. Kapitel 3.4), die oben als *local path planning* bezeichnet wird. Die genannten Teilprobleme sind sich teilweise ähnlich und entsprechend können einige der aufgeführten Algorithmen ebenfalls auf das jeweils andere Problem angewandt werden. Hier wird dennoch eine separate Darstellung vorgenommen, da Pfadplanung eher planungsorientiert und Hindernisvermeidung eher reaktiv ist.

3.3 Pfadplanung

Das Hauptziel der Pfadplanung liegt darin, einen Weg von einem Startpunkt zu einem Zielpunkt zu finden. Dabei soll der Weg möglichst optimal sein, was abhängig von der jeweiligen Situation unterschiedliche Parameter berücksichtigen muss (e. g. Wegstrecke, Energieverbrauch). Für gewöhnlich wird die Umgebung zur Berechnung des Pfades als Graph repräsentiert, in dem Start- und Zielpunkt als Knoten dargestellt sind und auf dem ein Graphsuch-Algorithmus (e. g. der unten beschriebene A*-Algorithmus, cf. Kapitel 3.3.2) zum Finden der Lösung operiert. Die Knoten des Graphen entsprechen Zuständen bzw. Positionen, die Kanten Aktionen zum Übergang zwischen Zuständen. Der resultierende Pfad — sofern eine Lösung möglich ist — ist dann eine Folge von Knoten in diesem Graph. [12, S,173][8, S.272]

Ein möglicher Pfad ist abhängig von der Geometrie des Objekts, für das der Pfad geplant wird. Eine mögliche Berücksichtigung dieser Geometrie wird in Kapitel 3.3.1 vorgestellt. Anschließend werden der bereits erwähnte A*-Algorithmus und einige Varianten zum Aufbau des Suchgraphen beschrieben.

3.3.1 Berücksichtigung der Objektgeometrie

Man unterscheidet zwischen *Arbeitsraum* und *Konfigurationsraum*. Der Arbeitsraum beschreibt den physischen Raum, in dem sich das Objekt befinden kann, und ist abhängig von der Geometrie des Raums. Der Konfigurationsraum beschreibt einen Raum zur Definition der Menge gültiger Zustände, die das Objekt annehmen kann, in Abhängigkeit von den Freiheitsgraden dieses Objekts. [8, S.272]

Zur Berücksichtigung der Geometrie eines Objekts können die im Arbeitsraum definierten Hindernisse vergrößert werden. In einem einfachen Fall wird das Objekt als kreisförmig, um den Konfigurationsraum unabhängig von der Orientierung des Objekts zu halten, und als Punkt im Zentrum des Kreises angenommen. Werden alle Hindernisse um den Radius des als Kreis angenommenen Objekts vergrößert, entspricht das dem von der Orientierung unabhängigen Konfigurationsraum. [12, S.183][8, S.275]

3.3.2 A*

A* bezeichnet einen Suchalgorithmus zum Finden des optimalen Pfades in einem Graphen, der auf Basis der Bestensuche funktioniert. Bei der Bestensuche wird aus einem Graphen der Knoten zuerst expandiert, der nach einer Evaluierungsfunktion $f(x)$ als am besten gewertet wird. Da die Evaluierungsfunktion die Kostenabschätzung darstellt, betrifft dies also den Knoten mit der geringsten Bewertung. [6, S.128]

Die A*-Suche nutzt als Bewertungsfunktion eine Kombination aus Pfadkosten ähnlich der *uniform-cost-Suche* (cf. [6, S.119 ff.]) und einer Heuristikfunktion ähnlich der *greedy best-first-Suche* (cf. [6, S.128 ff.]). Das heißt, die Bewertungsfunktion setzt sich aus den tatsächlichen Pfadkosten ($g(x)$) vom Startknoten zum aktuellen Knoten x und den nach einer Heuristikfunktion ($h(x)$) geschätzten Kosten für den Weg vom aktuellen zum Zielknoten auf Basis einer Heuristik zusammen (cf. [6, S.130]):

$$f(x) = g(x) + h(x) \tag{3.1}$$

Um die Optimalität des A*-Algorithmus sicherzustellen, muss die Schätzfunktion $h(x)$ eine *zulässige Heuristik* sein, das heißt, sie darf die Kosten zum Ziel nicht überschätzen. Für die

Anwendung des Algorithmus auf eine Baumsuche ist diese Bedingung ausreichend für Optimalität. Soll er als Graphensuche verwendet werden, muss die Heuristik außerdem *konsistent* sein. Eine Heuristik ist konsistent, wenn gilt

$$h(x) \leq c(x, x') + h(x'), \tag{3.2}$$

wobei x' ein Nachfolgeknoten von x und $c(x, x')$ die tatsächlichen Kosten von x nach x' ist. [6, S.131 ff.]

3.3.3 Sichtbarkeitsgraph

Sichtbarkeitsgraphen sind eine Variante von Straßenkarten. Das heißt, sie definieren eine Menge von Punkten, die später die Knoten im Suchgraphen darstellen, sowie Verbindungen zwischen diesen Punkten, die entsprechend die Kanten repräsentieren. Sichtbarkeitsgraphen nehmen Hindernisse als Polygone an. Alle Ecken aller Polygone sind dann die Punkte der Straßenkarte. Anschließend werden Strecken von allen Polygonecken zu jeweils allen anderen sichtbaren Polygonecken gezogen. Dabei bedeutet sichtbar, dass die gezogene Strecke kein Hindernis, i. e. Polygon, schneidet. Dies sind dann die Verbindungen der Straßenkarte. Zusätzlich werden Start- und Zielpunkt hinzugefügt und Verbindungen zu allen sichtbaren Polygonecken gezogen. [8, S.275]

Ein Vorteil von Sichtbarkeitsgraphen liegt darin, dass sie den kürzesten Weg zwischen zwei Punkten enthalten. Ein Nachteil ist, dass sie das Objekt direkt an Hindernissen vorbeiführen. Sind diese entsprechend Kapitel 3.3.1 angepasst, ist der Abstand des Objekts zum Hindernis an dessen Ecken minimal. [8, S.276]

Abbildung 3.5 zeigt exemplarisch einen Sichtbarkeitsgraphen — ohne Start- und Zielpunkt. Gefundene Pfade würden auf den gestrichelt dargestellten Sichtbarkeitsstrecken verlaufen.

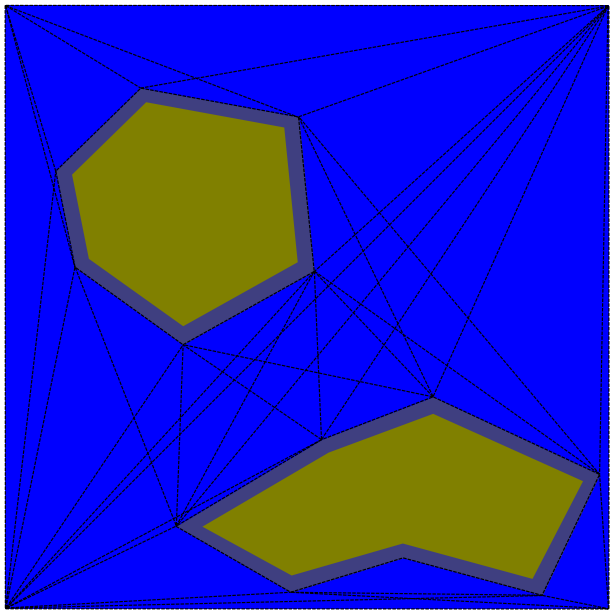


Abbildung 3.5: Sichtbarkeitsgraph auf einer Karte. Braune Flächen sind Hindernisse, der braunblaue Rand sind die nach Kapitel 3.3.1 angepassten Hindernisse und die schwarzen gestrichelten Linien sind Sichtbarkeitsstrecken.

3.3.4 Probabilistische Straßenkarte

Gegenüber dem zuvor in Kapitel 3.3.3 vorgestellten Sichtbarkeitsgraph bieten probabilistische Straßenkarten eine Variante, die den Freiraum zwischen Hindernissen besser füllen. Zusätzlich zu Start- und Zielpunkt werden N Punkte zufällig eingestreut — hier nicht innerhalb der Hindernisse. Anschließend werden die eingestreuten Punkte mit ihren n nächsten Nachbarn verbunden, wobei die Verbindungen keine (nach Kapitel 3.3.1 angepassten) Hindernisse schneiden dürfen. Diese Verbindungen stellen die Kanten im generierten Suchgraphen dar. Für gewöhnlich lohnt eine Beschränkung der Kantenlänge zwischen den Zufallspunkten. [8, S.276 f.]

Nachteile probabilistischer Straßenkarten sind neben den resultierenden gezackten Pfaden, die über nachgeschaltete Mechanismen geglättet werden können, eine starke Abhängigkeit von der Anzahl eingestreuter Punkte. So zeigt Abbildung 3.6, dass bei einer geringen Anzahl eingestreuter Punkte mögliche Pfade (rechts-unten in der Abbildung) vom Suchgraphen nicht gefunden werden können.

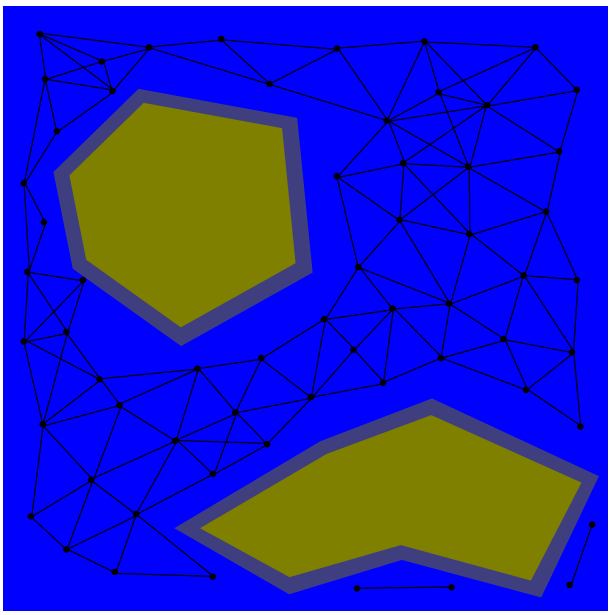


Abbildung 3.6: Probabilistische Straßenkarte auf einer Karte. Schwarze Punkte sind probabilistisch generierte Knoten des Graphen und schwarze Linien die Verbindungen.

3.3.5 Voronoi-Diagramm als Straßenkarte

Während die beiden vorangegangenen Varianten von Straßenkarten maximal dicht (Sichtbarkeitsgraphen) bzw. relativ nah (probabilistische Straßenkarten) entlang Hindernissen planen, sollen bei der Verwendung von Voronoi-Diagrammen als Straßenkarten Pfade entstehen, bei denen der Freiraum, i. e. der Abstand zu den Hindernissen, maximiert wird. [8, S.278]

Die Generierung der Knoten und Kanten des Voronoi-Diagramms soll hier nicht genauer beschrieben werden, stattdessen sei auf die zugrundeliegende Literatur verwiesen — cf. [8, S.278 ff.]. Abbildung 3.7 zeigt schematisch das Voronoi-Diagramm auf einer Karte.

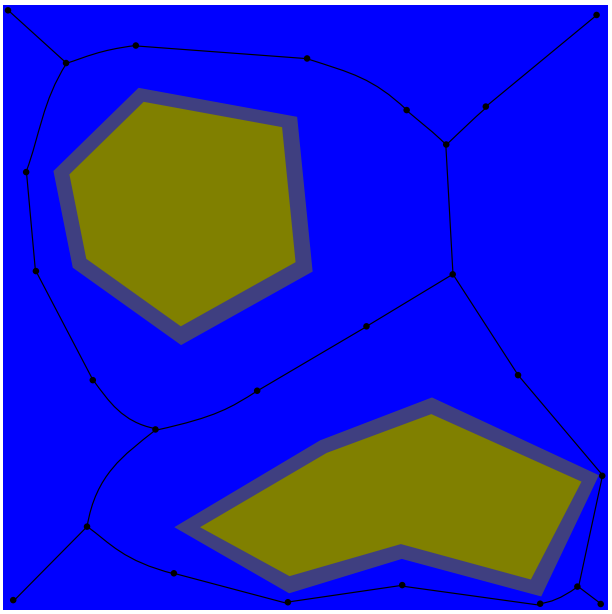


Abbildung 3.7: Voronoi-Diagramm auf einer Karte. Schwarze Punkte stellen Knoten des Suchgraphen und schwarze Linien dessen Kanten dar.

3.3.6 Rapidly-Exploring Random Tree

Ähnlich den probabilistischen Straßenkarten basieren Rapdily-exploring Random Trees (RRT) auf zufällig generierten Pfaden. Probabilistische Straßenkarten haben den Nachteil, dass sie eine Menge von Punkten generieren und diese miteinander verbinden. Zum einen stellt sich beim Generieren das Problem, dass die Punkte nicht überall liegen dürfen, da Hindernisse bestimmte Bereiche blockieren. Zum anderen berücksichtigen sie nicht die Kinematik von Objekten, wodurch das Folgen bestimmter Pfade, die der generierten Straßenkarte nach möglich sein sollten, nicht möglich ist. RRTs dagegen können bei ihrer Generierung kinematische Eigenschaften direkt berücksichtigen und so ausschließlich mögliche Pfade generieren. [7, S.1]

Ein RRT wird — ausgehend von einem Startpunkt x_{init} — so generiert, dass alle erzeugten Knoten sowie alle generierten Kanten ausschließlich im Konfigurationsraum liegen. Es wird zufällig ein Zustand x_{rand}, i. e. hier eine Position, aus dem Konfigurationsraum C gezogen (cf. $RANDOM\_STATE$ in Alg. 1). Anschließend wird aus der Menge aller bereits erzeugten Knoten X der diesem zufälligen Zustand am nächsten liegende x_{near} gesucht (cf. $NEAREST\_NEIGHBOR$ in Alg. 1). Anhand des ausgehenden Zustands x_{near} und des Zielzustands x_{rand} wird aus der Menge aller erlaubten Zustandsübergänge $u \in U$ derjenige gewählt, der den Abstand zu x_{rand} minimiert und dabei alle Seitenbedingungen erfüllt (cf. $SELECT\_INPUT$ in Alg. 1). Aus dem Ausgangszustand und dem Zustandsübergang wird dann ein neuer Zustand x_{new} erzeugt (cf. NEW_STATE in Alg. 1). Als Beispiel soll ein Dynamikmodell bei der Generierung des RRT berücksichtigt werden. Dann ist die Menge aller erlaubten Zustandsübergänge beschränkt durch die kinematischen Eigenschaften (e. g. Beschleunigung, Geschwindigkeit, Winkelgeschwindigkeit) sowie eine pro Zustandsübergang erlaubte Zeitspanne Δt. Ein entsprechender Algorithmus ist beispielhaft in Algorithmus 1 dargestellt. [7, S.2]

Für weitere Informationen zu RRTs, insbesondere zu deren Eigenschaften, sei auf die grundlegende Literatur [7] verwiesen. Abbildung 3.8 zeigt einen beispielhaft generierten RRT auf einer Karte. Dabei ist die Anzahl zu generierender Knoten K für die Kartengröße relativ klein gewählt.

Algorithmus 1 Generierung eines RRT mit K Knoten, ausgehend von einem Zustand x_{init} und Zeitschritten Δt (nach [7, S.2])

function $GENERATE\_RRT(x_{init}, K, \Delta t)$
  $X.init(x_{init})$;
  $k \leftarrow 1$
  for $k \leq K$ **do**
    $x_{rand} \leftarrow RANDOM\_STATE()$;
    $x_{near} \leftarrow NEAREST\_NEIGHBOR(x_{rand}, X)$;
    $u \leftarrow SELECT\_INPUT(x_{rand}, x_{near})$;
    $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta t)$;
    $X.add\_vertex(x_{new})$;
    $X.add\_edge(x_{near}, x_{new}, u)$;
  end for
  return X;
end function

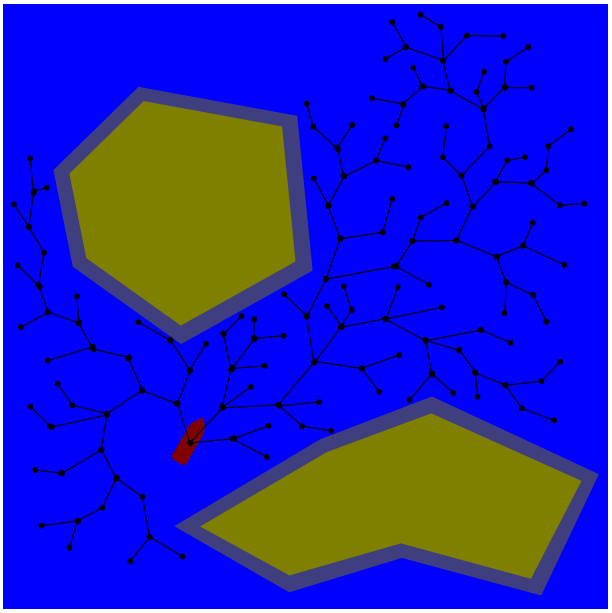


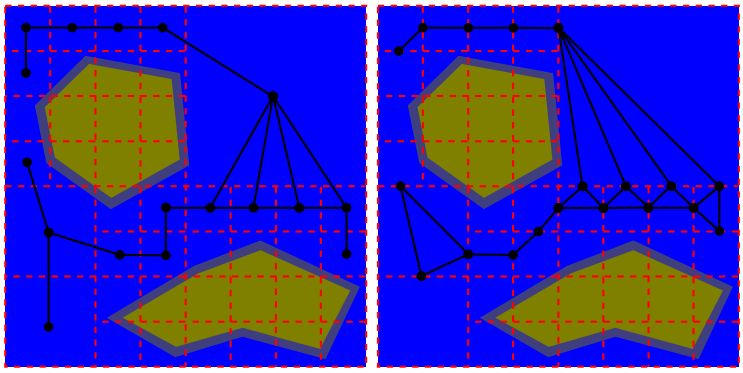
Abbildung 3.8: Generierter RRT auf einer Karte. Schwarze Punkte stellen Knoten des Suchgraphen und schwarze Linien dessen Kanten dar.

3.3.7 Pfadplanung in Raster- und Zellkarten

Wie in Kapitel 3.1.2 dargestellt, bieten Rasterkarten eine Möglichkeit, Freiraum zu repräsentieren. Zellen sind entweder frei oder belegt. Zwischen aneinander angrenzenden freien Zellen gibt es einen möglichen Weg — ohne Berücksichtigung kinematischer Beschränkungen. Um den für die Pfadplanung benötigten Suchgraphen zu erstellen, können freie Zellen also als Knoten und Grenzen zwischen freien Zellen als Kanten zwischen den entsprechenden Knoten angesehen werden. Es gibt nun verschiedene Varianten, wie die Zellen als Knoten dargestellt werden, um die entsprechenden Wegkosten für die Kanten zu ermitteln und die gefunden Pfade zu optimieren. Als einfachste Variante können die jeweiligen Mittelpunkte der Zellen als Knoten gewählt werden. Abhängig von der Art der Zellen — zum Beispiel wenn diese eine unregelmäßige Größe besitzen, wie es bei dem *Quadtree*-Algorithmus der Fall sein kann — können die Kanten zwischen diesen Mittelpunkten dann allerdings durch belegte Zellen und damit durch Hinder-

nisse laufen. Als Alternative können als Knoten für den Suchgraphen Punkte auf den Grenzen jeweils zweier freier Zellen gewählt werden. Sofern die Zellen auf konvexe Formen beschränkt sind, würden erzeugte Kanten nicht durch belegte Zellen bzw. durch Hindernisse laufen. Hier kann die Wahl der Form der Zellen allerdings dazu führen, dass unnötig lange Wege für die Durchquerung von Zellen geplant werden. [8, S.280 f.]

Abbildung 3.9 zeigt generierte Suchgraphen auf einer mithilfe des *Quadtree*-Algorithmus erzeugten Rasterkarte. Zum einen ist der Fall dargestellt, dass die Zellenmittelpunkte als Knoten angenommen werden, zum anderen der Fall, dass die Mittelpunkte auf den Grenzen zwischen zwei freien Zellen als Knoten verwendet werden. Hier sind lediglich Quadrate als Zellen gezeigt, wie in Kapitel 3.1.2 angemerkt sind aber auch andere Formen zur Einteilung der Umgebung in Zellen möglich.



(a) (b)

Abbildung 3.9: Pfadplanung auf einer mithilfe des *Quadtree*-Algorithmus erzeugten Rasterkarte: (a) bei Annahme der Zellmittelpunkte als Knoten und (b) bei Annahme der Mittelpunkte von Zellgrenzen als Knoten.

3.3.8 Navigation Mesh

Ein Spezialfall der Zellkarten und der darauf operierenden Pfadplanung ist das sogenannte *Navigation Mesh*. Die Welt wird dabei nicht in quadratische Zellen unterteilt, die als Freiraum oder blockiert markiert werden. Stattdessen wird der freie Bereich durch eine Menge konvexer Polygone (auch *Zellen* genannt) repräsentiert. Dieser Ansatz zur Beschreibung begehbarer Flächen, i. e. der Konfigurationsraum (cf. Kapitel 3.3.1) projiziert auf den Boden, wird häufig in Computerspielen für die automatische Pfadfindung verwendet. [4, S.231]

Die Polygone werden darauf beschränkt, die Eigenschaft *konvex* zu erfüllen, da so die Annahme getroffen werden kann, dass von jedem beliebigen Punkt innerhalb eines Polygons eine Linie zu jedem beliebigen anderen Punkt des Polygons gezogen werden kann, ohne das Polygon zu verlassen. Daraus folgt auch, dass von der Grenze zu einem beliebigen benachbarten Polygon eine solche Linie zur Grenze zu einem beliebigen anderen benachbarten Polygon gezogen werden kann. Eine Grenze ist dabei eine gemeinsame Kante zweier Polygone, i. e. die Polygone berühren sich im Bereich dieser Kante. [4, S.231]

Aus einem derart existierendem *Navigation Mesh* lässt sich ein Graph erzeugen, der für die Suche nach einer gültigen Route von einem Start- zu einem Zielpunkt verwendet werden kann. Ein exemplarisches *Navigation Mesh* ist in Abb. 3.10 dargestellt.

Der Vorteil des *Navigation Mesh* liegt darin, dass der Bereich angegeben wird, in dem sich ein Objekt bewegen kann. So lässt sich ein großer offener Bereich durch ein einzelnes Polygon —

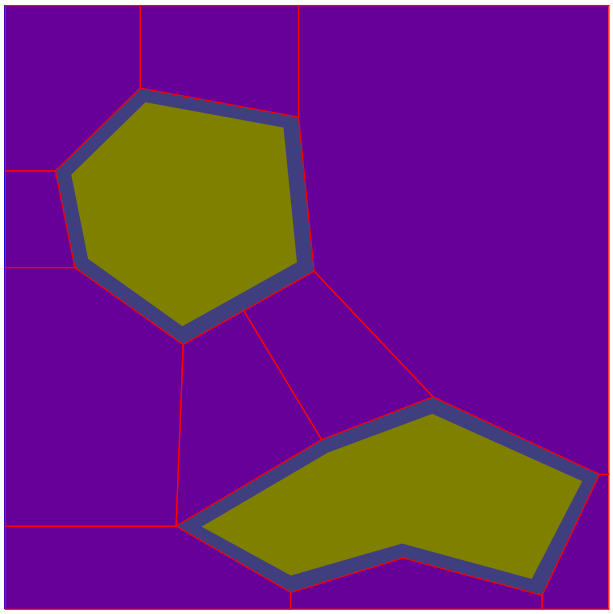


Abbildung 3.10: Exemplarisches *Navigation Mesh* auf einer Karte. Die konvexen Polygone sind rot umrandet dargestellt.

sofern dieses konvex ist — repräsentieren. Gleichzeitig enthalten die Polygone keine Informationen darüber, wie ein Objekt sich durch dieses Polygon hindurch bewegen muss. Hierfür gibt es verschiedene Ansätze, wie zum Beispiel eine simple Bewegung vom Zentrum einer Polygongrenze zur nächsten oder etwas anspruchsvoller durch die Verwendung von Bézier Splines. Ein weiterer Vorteil liegt darin, dass innerhalb der konvexen Polygone Ausweichbewegungen möglich sind, ohne den ermittelten Pfad zu verlassen. [4, S.231]

3.4 Hindernisvermeidung

Hindernisvermeidung ist neben der Pfadplanung ein zweites Hauptthema im Bereich der Navigation. Während jedoch die Pfadplanung eine übergeordnete Planung zum Finden eines Weges zu einem Zielzustand ist, betrifft die Hindernisvermeidung die nähere Umgebung eines Objekts. Dabei sollen reaktiv aufgrund von Veränderungen der Umgebung oder aufgrund von neuen Informationen zu dieser Entscheidungen getroffen werden, um eine Kollision mit Hindernissen zu vermeiden. Es geht also um reaktives Ausweichen. Im Folgenden sollen verschiedene Algorithmen zur Lösung dieses Problems kurz vorgestellt werden.

3.4.1 Wandering Standpoint Algorithmus

Der Wandering Standpoint Algorithmus (WSA) stellt eine simple Methode da, um auf einem Weg zum Ziel liegenden Hindernissen auszuweichen. Dabei folgt ein Objekt einem Pfad zum Zielpunkt. Nimmt das Objekt nun ein Hindernis in einem bestimmten Abstandsbereich $d_{min} < d < d_{max}$ wahr, weicht es diesem aus. Das Objekt dreht sich so, dass es parallel zum Rand des Hindernisses fahren kann und zwar in die Richtung, in die der Winkel der notwendigen Drehung kleiner ist. Sobald das Objekt das Hindernis so weit umfahren hat, dass der Weg zum Zielzustand wieder frei ist oder der Pfad, von dem zum Ausweichen abgewichen wurde, wieder erreicht wird, ist die Vermeidung des Hindernisses abgeschlossen. [12, S.203 ff.]

Wenn ein Objekt also auf ein Hindernis stößt, weicht es entweder nach links oder rechts aus und fährt entlang des Hindernis, bis der Weg zum Ziel durch dieses Hindernis nicht mehr blockiert ist.

Dieser Algorithmus kann aber in verschiedenen Fällen fehlschlagen. So kann ein dynamisches Hindernis ein Objekt vom Ziel wegdrängen (e. g. indem die beiden Objekte mit gleicher Geschwindigkeit nebeneinander herfahren) oder abhängig von der Anordnung mehrerer Hindernisse kann das Objekt niemals das Ziel erreichen. [12, S.205]

Abbildung 3.11 zeigt die Anwendung des WSA auf zwei beispielhafte Situationen. Die Richtung, in die das Objekt dreht, ist abhängig von dem Winkel zum Hindernis, wenn das Objekt auf dieses trifft.

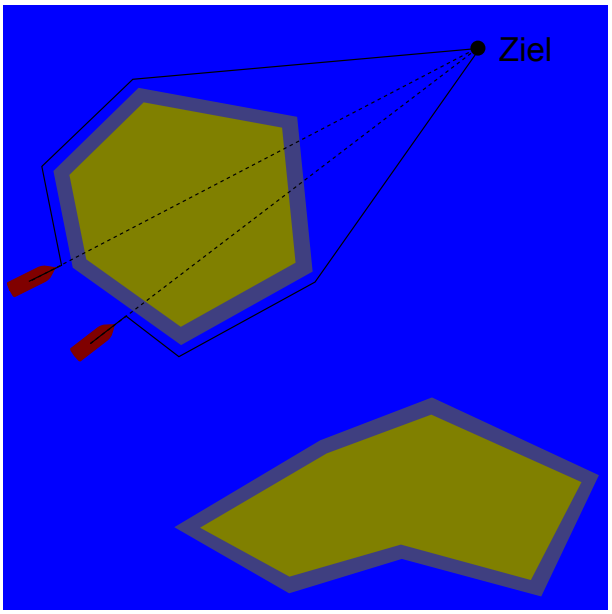


Abbildung 3.11: Anwendung des Wandering Standpoint Algorithmus auf zwei beispielhafte Objekte. Gestrichelte Linien zeigen jeweils den direkten Weg zum Ziel, durchgezogene Linien den tatsächlich abgefahrenen Weg.

3.4.2 Potential Field

Potentialfelder eignen sich sowohl für Pfadplanung als auch zur Hindernisvermeidung und basieren auf der Analogie zur Physik, dass das planende Objekt ein geladenes Teilchen ist, das sich in einem Raum bewegt, in dem anziehende und abstoßende Potentiale auf das Objekt mit Kräften wirken. [5, S.138]

Üblicherweise werden dabei für das Ziel ein anziehendes Potential $U_{goal}(\boldsymbol{q})$ und für Hindernisse abstoßende Potentiale $U_{obst,i}(\boldsymbol{q})$ verwendet, was in einem Potentialfeld

$$U(\boldsymbol{q}) = U_{goal}(\boldsymbol{q}) + \sum_i U_{obst,i}(\boldsymbol{q}) \tag{3.3}$$

resultiert. Daraus lässt sich die auf das Objekt wirkende Kraft und damit die Richtung, in die sich das Objekt bewegen soll, mithilfe des negativen Gradienten ermitteln (für $\boldsymbol{q} = \begin{pmatrix} x \\ y \end{pmatrix}$, i. e.

eine Position im Zweidimensionalen): [5, S.138]

$$F = -\nabla U(\boldsymbol{q}) = -\begin{pmatrix} \partial U/\partial x \\ \partial U/\partial y \end{pmatrix}. \tag{3.4}$$

Häufig wird für das anziehende Potential eine Funktion in Abhängigkeit vom Quadrat des Abstands

$$U_{goal}(\boldsymbol{q}) = \alpha \cdot dist(\boldsymbol{q}, \boldsymbol{q}_{goal})^2 \tag{3.5}$$

verwendet, wobei α ein konstanter Faktor ist. Diese Funktion führt dazu, dass das Ziel in großer Entfernung stärker anziehend wirkt als im Nahbereich, was ein Überfahren des Ziels vermeiden soll. [5, S.138 f.]

Für Hindernisse werden üblicherweise Funktionen verwendet, die gegen unendlich laufen, je näher das Objekt dem Hindernis kommt, und die mit steigender Entfernung einen geringeren Einfluss haben, also

$$U_{obst,i} = \beta_i \cdot dist(\boldsymbol{q}, \boldsymbol{q}_{obst,i})^{-1} \tag{3.6}$$

mit β_i als konstanter Faktor. Bei Hindernissen gibt es unterschiedliche Varianten, um die Recheneffizienz zu verbessern. So können lediglich Hindernisse innerhalb eines gewissen Abstands vom Objekt berücksichtigt werden, nur das nächste Hindernis oder alle Hindernisse — wobei letzteres abhängig von der Anzahl Hindernisse rechenintensiv sein kann. [5, S.139]

Die Potentialfeldmethode hat allerdings ein grundlegendes Problem: lokale Minima. Objekte bewegen sich in Richtung des Gradienten, der in erster Linie vom anziehenden Potential des Zielzustands beeinflusst wird, das über weite Entfernungen wirkt. Der Einfluss von Hindernissen auf das Objekt beschränkt sich eher auf den Nahbereich, was bei ungünstig liegenden Hindernissen dazu führen kann, dass sich das Objekt in eine *Falle* bewegt (cf. Abb. 3.12). [5, S.139]

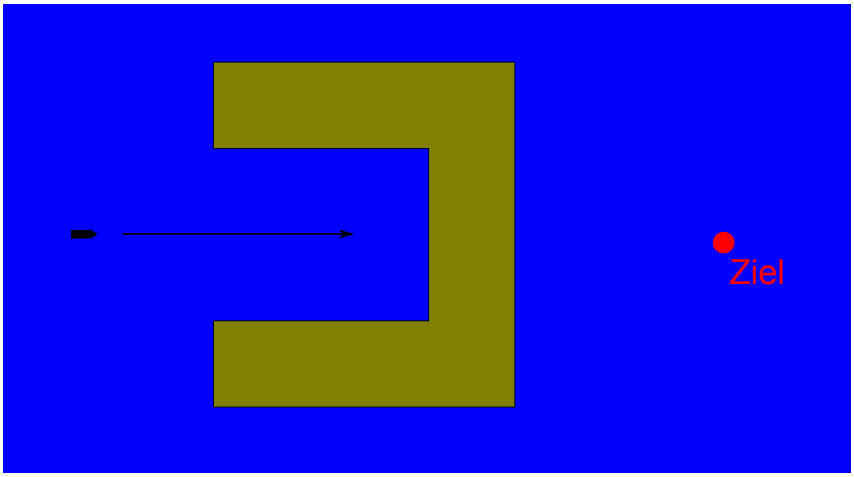


Abbildung 3.12: Darstellung des Problemfalls lokaler Minima bei der Potentialfeldmethode (nach [5, S.139]). Das Schiff fährt auf das Ziel zu und kommt aus dem Hindernis nicht heraus.

Für dieses Problem der Potentialfelder gibt es verschiedene Lösungsansätze, die jeweils eigene Vor- und Nachteile haben. So können Potentialfelder als Teil eines globalen Planers verwendet werden, der für Verbindungen zwischen verschiedenen Minima eine Graphstruktur verwendet, während der lokale Planer lediglich auf dem Potentialfeld operiert. Eine weitere Möglichkeit besteht in der Diskretisierung des Zustandsraums und der Suche nach einem Pfad, der das Ziel tatsächlich erreicht. Außerdem bieten Kombinationen mit anderen Algorithmen, die bei Auftreten eines lokalen Minimums greifen, Lösungsansätze. [5, S.139 f.]

Neben der Vermeidung lokaler Minima oder der Behebung des Falls, in ein lokales Minimum gelaufen zu sein, ergibt sich auch die Schwierigkeit, ein lokales Minimum zu erkennen. Dies stellt sich häufig als Problem dar, insbesondere weil das Objekt aufgrund kinematischer Eigenschaften, Trägheit oder Ungenauigkeiten nicht das Minimum genau erreicht, sondern stattdessen um dieses oszilliert. [5, S.140]

Da hier lediglich eine möglichst kompakte Einführung gegeben werden soll, sei für tiefgehendere Informationen — neben der oben als Quelle genannten — auf weitere Literatur verwiesen. So bieten [12, S.206 ff.] und [8, S.281 ff.] einen sehr knappen, jedoch die Idee der Potentialfelder vermittelnden Eindruck. Ein ausführlicher und allgemeiner Einstieg inklusive spezieller Algorithmen zur Vermeidung der Probleme von Potentialfeldern wird in [1, S.77 ff.] gegeben.

3.5 Kontrollarchitekturen

Ein Kernthema — insbesondere in der Robotik — ist der architektonische Aufbau von Modulen zur Kontrolle eines autonomen Objekts. Da in dieser Arbeit im Bereich der lokalen Planung (cf. Kapitel 4.1.1.5) eine ebensolche Kontrollarchitektur verwendet werden muss, um die verschiedenen Module zur Repräsentation der Kollisionsverhütungsregeln (cf. Kapitel 2.1) zu kombinieren, sollen hier einige Ansätze kurz dargestellt werden. So können die notwendigen Grundlagen für die spätere Diskussion zur Wahl der Architektur geschaffen werden.

In erster Linie wird bei Kontrollarchitekturen zwischen parallelen (*behavior-based*, cf. [12, S.223]) und sequentiellen (*task-oriented*, cf. [12, S.223]) Ansätzen unterschieden. Bei sequentiellen Ansätzen werden Daten von einem vorhergehenden zum folgenden Modul weitergegeben, wobei das erste Modul die Sensordaten erhält und das letzte Steuerungskommandos ausgibt. Parallele Architekturen hingegen arbeiten auf den gleichen Daten und geben unabhängige Kommandos aus. Letztere müssen dann kombiniert werden. [8, S.320 f.]

In den folgenden Abschnitten werden die beiden Ansätze für Kontrollarchitekturen genauer erläutert. Dabei wird die *SMPA*-Architektur (engl. *Sense, Model, Plan, Act*) zur Darstellung der sequentiellen Architektur verwendet. Neben diesen beiden Varianten gibt es außerdem hybride Architekturen, auf die hier allerdings nicht näher eingegangen werden soll.

3.5.1 Sequentielle Architektur (*SMPA*)

Die *SMPA*-Architektur ist eine extremisierte Darstellung und wird für gewöhnlich so nicht verwendet. Dennoch stellt es die Idee des sequentiellen Ansatzes heraus. SMPA geht davon aus, dass sich der Kontrollzyklus in vier nacheinander ablaufende Schritte unterteilt: *Sense, Model, Plan, Act*.

Sense. Aufnahme der Daten von Sensoren.

Model. Verarbeitung der aufgenommenen Daten in ein Modell der Umgebung.

Plan. Prüfung, in wie fern das Modell mit dem aktuell verfolgten Plan vereinbar ist, und eventuelle Anpassung des Plans oder Neuplanung.

Act. Ausführung der aktuell laufenden oder neu gewählten Aktion. [8, S.321 f.]

Diese vier Schritte werden in jedem Zyklus aufeinander folgend bearbeitet, was aufgrund der rechenaufwendigen Schritte *Model* und *Plan* zu einem erhöhten Optimierungsbedarf führt. Der

Kontroll- bzw. Datenfluss ist hierbei unidirektional in Richtung der Folge dieser vier Schritte. [8, S.321]

Der Zugriff auf die Interaktion mit der Umwelt ist dabei stark beschränkt. So hat das *Sense*-Modul exklusiven Zugriff auf Sensordaten, während das *Act*-Modul exklusiven Zugriff auf die Steuerung der Aktuatoren hat. [12, S.223]

3.5.2 Parallele Architektur

Parallele bzw. verhaltensbasierte Architekturen stützen sich auf die Idee voneinander unabhängiger Module bzw. Verhalten. Dabei hat jedes Modul Zugriff auf Sensordaten wie auch auf Aktuatorensteuerung und hält seine eigene — üblicherweise auf die Aufgabe des Moduls beschränkte — Repräsentation der Umwelt. Diese Unabhängigkeit erfordert allerdings einen Mechanismus zur Auflösung verschiedener Steuerungskommandos, die miteinander in Konflikt stehen. [12, S.223]

Für die Kombination verschiedener Verhaltensmodule lassen sich zunächst zwei Varianten unterscheiden:

Arbitrated Behaviors. Verhaltensmodule sind *arbitrated*, i. e. für eine begrenzte Zeit hat lediglich ein Modul oder eine bestimmte Menge von Modulen die Kontrolle.

Fusion. Fusion von Kommandos der Verhaltensmodule, i. e. alle Module können Kommandos geben, die dann fusioniert werden, sodass das resultierende Kommando eindeutig ist. [12, S.241]

Für den Fall der *arbitrated behaviors* — dieser Ansatz ist vorzuziehen, wenn die gegebenen Kommandos unverändert verwendet werden sollen — lassen sich folgende Ansätze unterscheiden (cf. [12, S.241 f.]):

Prioritätsbasiert. Verhaltensmodule werden entsprechend der ihnen zugewiesenen Prioritäten ausgewählt.

Zustandsbasiert. Verhaltensmodule werden entsprechend des aktuellen Zustands und ihrer jeweiligen Spezialisierung auf diesen Zustand ausgewählt.

Winner-takes-all. Ein einzelnes Verhaltensmodul wird anhand eines beliebigen Vergleichs ausgewählt.

Beispiele für Varianten der Fusion sind im Folgenden aufgelistet (cf. [12, S.242 f.]):

Voting. Jedes Verhaltensmodul stimmt für eine oder mehrere Aktionen, wobei die Aktion mit den meisten Stimmen gewählt wird.

Superposition. Aktionen von Verhaltensmodulen werden als Vektoren angegeben, die dann linear kombiniert werden.

Als bekannter Vertreter verhaltensbasierter Architekturen sei hier der Vollständigkeit halber noch die *Subsumption*-Architektur genannt. Diese soll allerdings nicht näher beschrieben werden. Stattdessen wird auf weiterführende Literatur verwiesen. Ein kurz gehaltener Überblick ist in [12, S.229 ff.] gegeben.

Kapitel 4

Regelbasiertes Autonomes Verhalten

In diesem Kapitel wird die Entwicklung des Kernthemas dieser Arbeit beschrieben. Dabei handelt es sich — wie in der Einleitung erläutert — um das autonome Verhalten eines Schiffs auf Basis der Kollisionsverhütungsregeln. Es beschränkt sich auf die Bewegung des Schiffs, während die Kontrolle und Verarbeitung von Kommunikationsmitteln wie Lichter, Signalhörner oder verbaler Kommunikation im Rahmen dieser Arbeit nicht beachtet werden.

Zunächst wird das grundlegende Konzept für das autonome Verhalten mit Fokus auf den Kontrollzyklus inklusive der Schnittstellen zwischen den Schritten beschrieben. Dieser wird in drei Schritte unterteilt: Auftragsplanung, globale Pfad- bzw. Routenplanung, lokale Planung (bzw. Verhalten). Da sich diese Arbeit mit der Verarbeitung der Kollisionsverhütungsregeln (cf. Kapitel 2.1) beschäftigt, liegt der Fokus auf den Schritten globale Pfadplanung und lokales Verhalten. Zum Konzept gehört außerdem die Schnittstelle zum Simulator. Anschließend werden die verschiedenen Anteile des Kontrollzyklus — unter Beachtung der Schwerpunkte — im Hinblick auf die Umsetzung ausgeführt. Zuletzt werden einige bekannte noch ausstehende Probleme, die bei der Entwicklung des Konzepts oder dessen Umsetzung aufgetreten sind, erläutert.

4.1 Konzept

In diesem Kapitel soll das Konzept für den Aufbau des autonomen Verhaltens beschrieben werden. Dazu gehört neben einer ausführlichen Beschreibung des Kontrollzyklus (cf. Kapitel 4.1.1), i. e. die Planung der Steuerung eines autonom fahrenden Schiffs, auch eine kurze Beschreibung der Anbindung des Simulators (cf. Kapitel 4.1.2).

4.1.1 Kontrollzyklus

Der Kontrollzyklus ist der zentrale Teil dieser Arbeit, da er anhand von Umgebungsdaten notwendige Steuerungskommandos ermittelt, mit denen das Schiff fahren soll. Zunächst wird der grundlegende Aufbau des Kontrollzyklus vorgestellt. Anschließend werden die Schnittstellen zwischen den verschiedenen Schritten diskutiert und festgelegt. Zuletzt werden die einzelnen Schritte jeweils genauer betrachtet.

4.1.1.1 Aufbau

Das autonome Verhalten fällt in den Bereich *Navigation*, der in Kapitel 3.2 mit verschiedenen darin auftretenden Teilproblemen näher beschrieben ist. Daher orientiert sich der hier verwende-

te Aufbau des Kontrollzyklus an der dort dargestellten Aufteilung in einen globalen Pfadplaner (cf. *global path planning*) und einen lokalen Planer (cf. *local path planning*)[1]. Allerdings kann eine Planung nur durchgeführt werden, wenn auch ein Ziel bekannt ist, i. e. es wird ein Auftrag benötigt, den das autonom fahrende Schiff abarbeiten kann. Da hier lediglich die Bewegungssteuerung der Schiffe relevant ist, beschränken sich die Aufträge auf die Vorgabe von Zielorten. Dieser grobe Aufbau ist in Abb. 4.1 dargestellt.

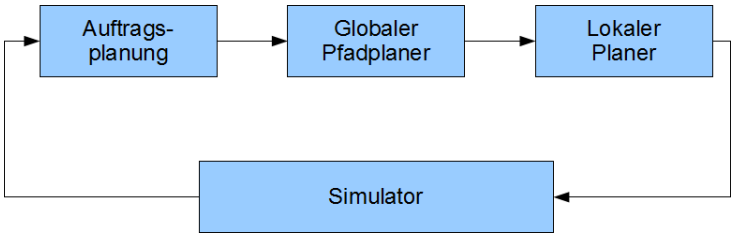


Abbildung 4.1: Grober Entwurf des Kontrollzyklus mit Aufteilung in drei sequentielle Schritte: Auftragsplanung, globaler Pfadplaner, lokaler Planer.

Der Ablauf dieser drei Schritte erfolgt sequentiell und wird in jedem Zyklus des Simulators aufgerufen. Die Entscheidung, diese Schritte sequentiell durchzuführen, ist offensichtlich, da der globale Routenplaner auf Basis des aktuellen Auftrags arbeiten muss und der lokale Planer nur im Rahmen einer gültigen Route arbeiten soll. Daraus folgt, dass der lokale Planer nur dann arbeitet, wenn eine gültige Route und damit auch ein gültiger Auftrag gegeben sind. Dies wird hier als ausreichend angenommen, ist allerdings nicht vollständig korrekt, da auch ohne Auftrag bzw. Route Ausweichbewegungen möglich sein müssen (cf. Kapitel 4.3). Der Aufruf aller drei Schritte in jedem Zyklus ist an sich unnötig, da sich Aufträge nur selten ändern und für gewöhnlich auch nur bei Änderung — was von außerhalb des Moduls und damit explizit ausgelöst wird — eine erneute Planung durchgeführt werden muss. Ebenfalls muss der globale Routenplaner nicht so häufig aufgerufen werden, sondern lediglich bei Änderung des Auftrags oder wenn die bestehende Route ungültig wird. Lediglich das lokale Verhalten muss häufig aufgerufen werden, um auf geänderte Situationen schnell reagieren zu können.

Eine Reduzierung der Aufrufhäufigkeit würde den Rechenaufwand verringern und lediglich zur Optimierung beitragen. Hier wird es der Umsetzung des Auftragsplaners (cf. Kapitel 4.2.2) und des globalen Pfadplaners (cf. Kapitel 4.2.3) überlassen, effizient auf unnötig häufige Aufrufe zu reagieren.

Neben der Sicht auf die Aufrufsequenz der Schritte innerhalb des Kontrollzyklus spielt auch die datenorientierte Sicht eine wichtige Rolle. So benötigt der Kontrollzyklus Daten von der Umgebung, auf Grundlage derer die verschiedenen Planer arbeiten können. Wäre der hier vorgestellte Kontrollzyklus in einer *SMPA*-Architektur (cf. Kapitel 3.5.1) eingebettet, würde er dem Planungsschritt entsprechen und damit auf dem Modell der Umwelt arbeiten. Hier wird das Modell der Umwelt (hier auch *Szenario* genannt) direkt vom Simulator zur Verfügung gestellt, was zu einer Vereinfachung führt, da die Umgebung als bekannt angenommen wird. Neben der Notwendigkeit, auf Daten der Umweltrepräsentation zugreifen zu können, müssen außerdem Steuerungskommandos vom Kontrollzyklus für das gesteuerte Schiff vorgegeben werden. Während der ausschließlich lesende Zugriff auf die Umgebungsdaten unproblematisch und von allen Schritten des Kontrollzyklus unabhängig erfolgen kann, müssen Steuerungskommandos eindeutig sein. Dieses generelle Probleme paralleler Kontrollarchitekturen (cf. Kapitel 3.5.2) wird hier reduziert bzw. auf eine feinere Entwurfsebene verschoben (cf. Kapitel 4.1.1.5), indem der Zugriff exklusiv dem Modul des lokalen Planers gewährt wird.

[1] cf. [12, S.173]

Zusätzlich zu diesen Schnittstellen zum Simulator werden zwischen den Schritten des Kontrollzyklus Schnittstellen benötigt. Der Auftragsplaner benötigt eine Vorgabe, um daraus einen Auftrag erzeugen zu können. Dies soll über eine Benutzerschnittstelle erfolgen, über die der Auftrag definiert werden kann. Für die Schnittstelle zwischen Auftragsplaner und globalem Pfadplaner wird hier die simple Vorgabe eines Zielortes definiert. Vom globalen Pfadplaner zum lokalen Planer soll eine Route angegeben werden, auf der der lokale Planer dann arbeiten kann. Diese Schnittstellen werden später genauer definiert (cf. Kapitel 4.1.1.2).

Ein Überblick über den oben beschriebenen Datenfluss und -zugriff sowie die verwendeten Schnittstellen ist in Abb. 4.2 dargestellt.

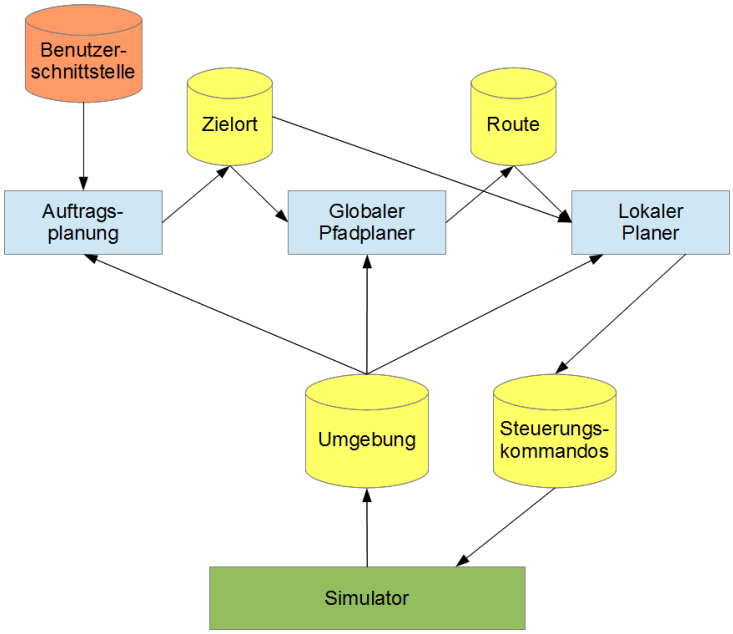


Abbildung 4.2: Übersicht über den Datenfluss innerhalb des Kontrollzyklus sowie zwischen Kontrollzyklus und Simulator: blau sind Module des Kontrollzyklus, gelb sind Datenschnittstellen, grün ist der Simulator, rot eine Datenschnittstelle, die vom Benutzer befüllt wird.

4.1.1.2 Schnittstellen

Für den Kontrollzyklus werden mehrere Schnittstellen zur Kommunikation mit der Umgebung und untereinander benötigt (cf. Abb. 4.2). Hier sollen lediglich die Schnittstellen genauer erläutert werden, die unabhängig vom Simulator sind.

Benutzerschnittstelle des Auftragsplaners

Die Eingangsschnittstelle des Auftragsplaners ist eine vom Benutzer zu befüllende und abhängig vom internen Aufbau bzw. von der Umsetzung des Auftragsplaners. Da sie somit sehr spezifisch ist, wird sie in Kapitel 4.1.1.3 näher erläutert.

Zielort (Auftragsplaner → Globaler Pfadplaner)

Die Schnittstelle zwischen Auftragsplaner und globalem Pfadplaner ist durch die Vorgabe eines Zielortes definiert. Die Idee dabei besteht darin, dass der Auftragsplaner — unabhängig von der

Art an Aufträgen — immer einen anzufahrenden Punkt vorgibt, zu dem sich das Schiff bewegen soll. So kann der Auftragsplaner intern lediglich eine Zielposition halten, die dann vom Schiff angefahren wird. Er kann allerdings auch eine Folge von Wegpunkten verarbeiten, wobei der jeweils nächste Wegpunkt in die Schnittstelle geschrieben wird.

Ein Zielort ist definiert als ein zweidimensionaler Punkt in globalen Koordinaten. Wie zu Beginn des Kapitels erwähnt, wird hier als globales Koordinatensystem das Weltkoordinatensystem des Simulators verwendet (cf. Kapitel 2.2.1.1). Zusätzlich zu der tatsächlichen Position wird eine Angabe benötigt, ob eine Zielposition vorgegeben ist, i. e. ob die aktuell gesetzte Zielposition gültig ist.

Route (Globaler Pfadplaner → Lokales Verhalten)

Die Schnittstelle zwischen globalem Pfadplaner und lokalem Verhalten wird hier als *Route* bezeichnet. Die Route definiert einen gültigen Weg von der aktuellen Position zum vorgegebenen Zielort. Gültig bedeutet hier, dass der Weg komplett innerhalb des Konfigurationsraums (cf. Kapitel 3.3.1) liegt. Ein Weg ist dabei eine Folge von Strukturen (e. g. Wegpunkte), die in dieser Folge abgefahren zum Zielort führen.

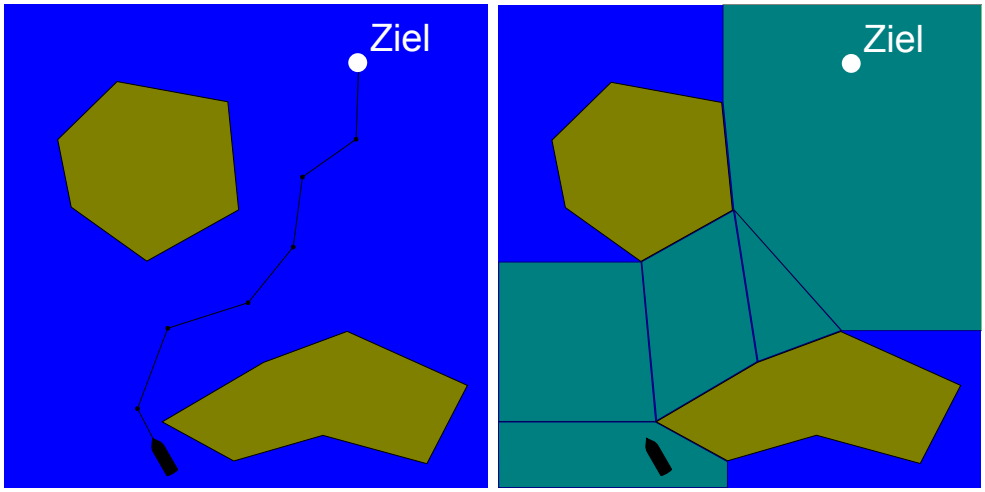
In Kapitel 3.3 sind diverse Algorithmen zum Ermitteln eines Graphen dargestellt, aus dem eine solche Route gefunden werden kann. Die meisten basieren auf der Berechnung von Wegpunkten und den Verbindungen zwischen diesen. Daneben existieren aber auch Ansätze auf Basis von freien geometrischen Flächen wie quadratischen Zellen (cf. Kapitel 3.3.7) oder Polygonen (cf. Kapitel 3.3.8). Die verschiedenen Ansätze bieten unterschiedliche Vor- und Nachteile. So werden bei Voronoi-Diagrammen als Straßenkarten (cf. Kapitel 3.3.5) Routen gewählt, die einen möglichst großen Abstand zu Hindernissen haben. Im Gegensatz dazu werden bei Sichtbarkeitsgraphen (cf. Kapitel 3.3.3) Hindernisse möglichst eng umfahren. Die Vorteile des einen sind also gleichzeitig die Nachteile des anderen.

Generell lassen sich aber die beiden Ansätze unterscheiden, die Route als eine Folge von Wegpunkten oder als eine Folge von Flächen zu repräsentieren. Der Vorteil der wegpunktbasierten Route liegt darin, dass die Steuerungskommandos anhand der Route leicht ermittelt werden können, da jeweils der nächste Wegpunkt direkt angesteuert wird. Außerdem können bei der Generierung der Wegpunkte je nach gewähltem Verfahren (e. g. *Rapidly-Exploring Random Tree*, cf. Kapitel 3.3.6) die kinematischen Beschränkungen des Objekts direkt berücksichtigt werden. Ein großer Nachteil der Nutzung von Wegpunkten ist jedoch der Verlust an Information über den zur Verfügung stehenden Freiraum. Ist die vom Planer ermittelte Route an einer Stelle durch ein Hindernis blockiert, muss das autonom fahrende Schiff diesem Hindernis ausweichen. Dadurch muss es die Wegpunktfolge der gegebenen Route verlassen, ohne dabei in ein anderes Hindernis zu fahren. Insbesondere bei dynamischen Hindernissen kann das Finden einer solchen Ausweichmöglichkeit sehr rechenintensiv werden. Zusätzlich muss, nachdem das Hindernis passiert ist, eine Rückkehr zur gegebenen Route gefunden werden. Ein weiterer Nachteil ist — abhängig vom verwendeten Verfahren zur Generierung der Wegpunkte (e. g. bei RRTs, cf. Kapitel 3.3.6, oder probabilistischen Straßenkarten, cf. Kapitel 3.3.4) — dass eine Route möglicherweise nicht optimal durch einen Freiraum läuft, sondern zu einer Zickzack-Bewegung führt.

Pfadplanung in Raster- und Zellkarten (cf. Kapitel 3.3.7) bieten hier einen Ansatz für eine Repräsentation der Route auf Basis von Freiraum. Verbessert wird dieser Ansatz durch Nutzung eines *Navigation Mesh* (cf. Kapitel 3.3.8), i. e. anstatt regelmäßiger Zellen werden konvexe Polygone verwendet, die den Freiraum spezifizieren. So lässt sich das *Navigation Mesh* als Graph repräsentieren, auf dem übliche Suchalgorithmen arbeiten können. Der resultierende Pfad ist

dann die Route, die als Folge von konvexen Polygonen sowie den *Toren*, i. e. Übergang zwischen zwei Polygonen, definiert ist. Dadurch ist die Information über den vorhandenen Freiraum in der Route enthalten und Ausweichbewegungen können innerhalb der Polygone durchgeführt werden, ohne dass die Route verlassen wird. Die Einschränkung auf konvexe Polygone ist sinnvoll aufgrund ihrer Eigenschaften, die in Algorithmen genutzt werden können. Dies betrifft insbesondere die Eigenschaft, dass von jedem beliebigen Punkt innerhalb eines konvexen Polygons zu jedem beliebigen anderen Punkt innerhalb desselben Polygons eine Strecke gezogen werden kann, die gänzlich innerhalb dieses Polygons liegt.

Eine Darstellung möglicher aus diesen Ansätzen resultierender Routen ist in Abb. 4.3 visualisiert. In diesem Bild sind die konvexen Polygone manuell erzeugt, wodurch der Nachteil des *Navigation Mesh*-Ansatzes hier nicht offensichtlich ist: Die resultierende Route und damit die Basis, auf dem das lokale Verhalten arbeitet, ist abhängig von der Definition der konvexen Polygone. Liegen die Polygone ungünstig, hat dies Einfluss auf das lokale Verhalten und schränkt es in den Möglichkeiten zur Durchführung von Ausweichbewegungen ein.



(a) (b)

Abbildung 4.3: Darstellung einer Route in der Schnittstelle vom globalen Pfadplaner zum lokalen Verhalten: (a) bei Verwendung eines wegpunktbasierten und (b) bei Verwendung eines freiraumbasierten Ansatzes.

Für diese Arbeit wird als Ansatz gewählt, die Route durch eine Folge von konvexen Polygonen zu repräsentieren. Grund dafür sind die teilweise komplexen Ausweichmanöver auf Basis der Kollisionsverhütungsregeln, die im dritten Schritt, i. e. im lokalen Verhalten, enthalten sein sollen. Durch diese Wahl sind die Informationen über den zur Verfügung stehenden Freiraum in der Route enthalten, sodass das lokale Verhalten das Terrain, i. e. die Küstenlinien, nicht mehr betrachten muss. Eine Wegpunktroute würde hier die Ausweichmöglichkeiten einschränken bzw. die Verarbeitung der Umgebung unnötig komplex und rechenintensiv machen. Für die Ermittlung der Route wird ein *Navigation Mesh* gewählt, das die konvexen Polygone enthält (cf. Kapitel 4.1.1.4).

Die Route wird zusätzlich erweitert, da bestimmte Informationen über die Route oder Elemente der Route, i. e. ein konvexes Polygon, zusätzlich im Modul des globalen Planers ermittelt und dem lokalen Verhalten zur Verfügung gestellt werden sollen. Dadurch enthält jedes Routenelement folgende Daten:

Konvexes Polygon. Das konvexe Polygon als solches enthält einen Teil des Freiraums, in dem sich das Schiff bewegen kann. Dies ist ein Bereich, in dem keine statischen Hindernisse

wie Landmassen existieren. Es ist definiert durch eine Menge von Eckpunkten in globalen Koordinaten.

Polygonzentrum. Die Zentrumsposition des gegebenen konvexen Polygons identifiziert eindeutig das Polygon — da alle Polygone konvex sind und Überschneidungen ausgeschlossen werden — und kann verwendet werden, um mögliche Änderungen der Route zu identifizieren.

Tor (bzw. *Gate*). Sofern das Routenelement nicht das letzte in der Folge ist, die die Route beschreibt, muss es einen Übergang zum folgenden Routenelement geben. Dieser Übergang ist die Schnittlinie, an der sich die Polygone berühren. Daraus folgt, dass es der Bereich ist, den das Schiff passieren muss, um der Route zu folgen — daher die Bezeichnung *Tor*.

Zielort. Hierüber wird identifiziert, ob das Routenelement den Zielort enthält. Damit wären die Daten für das Tor (siehe oben) ungültig und stattdessen muss innerhalb dieses Polygons der Zielort angefahren werden.

Aus diesen Informationen lässt sich eine Route wie folgt formal definieren.

Definition 1. *Eine **Route** R der Länge N ist eine Sequenz von Routenelementen E_i*

$$R = \{E_0, E_1, \ldots, E_{N-1}\}, \tag{4.1}$$

wobei ein Routenelement ein Tupel

$$E_i = (P_i, \vec{x}_{center,i}, G_i, T_i) \tag{4.2}$$

mit P_i als konvexes Polygon des Navigation Mesh, das von einer Menge von n_i Punkten definiert wird,

$$P_i = \{\vec{x}_{i,0}, \vec{x}_{i,1}, \ldots, \vec{x}_{i,n_i-1}\}, \tag{4.3}$$

$\vec{x}_{center,i}$ als Zentrum des Polygons P_i, G_i als Gate zum Polygon $i+1$ ein Tupel

$$G_i = (\vec{x}_{i,0}, \vec{x}_{i,1}) \tag{4.4}$$

für $i < N-1$ und T_i als Markierung des finalen Routenelements

$$T_i = \begin{cases} true & falls\ i = N-1 \\ false & falls\ i \neq N-1 \end{cases} \tag{4.5}$$

ist.

4.1.1.3 Auftragsplaner

Der Auftragsplaner ist der erste Schritt des Kontrollzyklus. Er kann auf die Umgebungsdaten zugreifen und soll über eine Benutzerschnittstelle gesteuert werden können (cf. Abb. 4.2). Als Ausgabe liefert er einen Zielort oder setzt den Zielort ungültig.

Der Begriff *Auftrag* ist hier sehr allgemein gefasst. So kann eine Vielzahl verschiedener Auftragsplaner diese Aufgabe übernehmen, die jeweils für eine spezielle Art von Auftrag genutzt werden können. Der vermutlich einfachste besteht in der simplen Vorgabe des Zielortes durch den Benutzer. Weitere könnten zum Beispiel eine Folge von Wegpunkten sein oder als Spezialfall von diesem die Vorgabe zweier Häfen, zwischen denen ein Schiff hin- und herfährt. Da — wie bereits zuvor erwähnt — der Fokus dieser Arbeit nicht auf dem Auftragsplaner liegt und sich

letztlich alle Aufträge auf den einfachen Fall der Vorgabe eines einzelnen Zielortes zurückführen lassen, wird hier lediglich diese simple Umsetzung verwendet.

Dennoch wird aus Gründen der Erweiterbarkeit für zukünftige Arbeiten vorgesehen, den Auftragsplaner modular aufzubauen. Das heißt, der Auftragsplaner besteht aus mehreren voneinander unabhängigen Modulen, die verschiedene Aufträge verarbeiten können. Dabei ist allerdings darauf zu achten, dass lediglich ein einzelner Zielort an die Schnittstelle zum globalen Planer gegeben wird. Ein schematischer Aufbau unter Berücksichtigung, dass jedes Auftragsplaner-Modul seine eigene — hier nicht näher beschriebene — Benutzerschnittstelle hat, ist in Abb. 4.4 dargestellt.

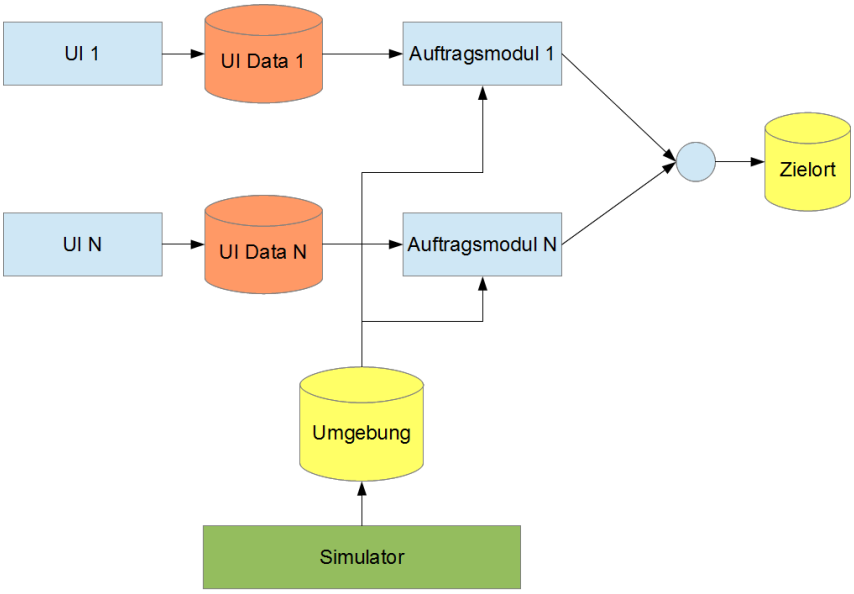


Abbildung 4.4: Darstellung des modularen Aufbaus des Auftragsplaners für Auftragsmodul 1 und Auftragsmodul N.

4.1.1.4 Globaler Pfadplaner

Der globale Pfadplaner ist der Schritt zwischen einer abstrakten Zielvorgabe durch die Auftragsplanung und den speziellen Steuerkommandos, die vom lokalen Verhalten bestimmt werden. Der Pfadplaner nutzt die aktuelle Position des autonom fahrenden Schiffs sowie den vorgegebenen Zielort (cf. Kapitel 4.1.1.2), sofern dieser gültig ist, und ermittelt daraus eine Route, die an das lokale Verhalten weitergegeben wird. Wie in Kapitel 4.1.1.2 diskutiert, soll die Route eine Folge von konvexen Polygonen sein. Für die Definition der konvexen Polygone, aus denen die Route ermittelt wird, wird ein *Navigation Mesh* verwendet.

Ein *Navigation Mesh* unterteilt den Konfigurationsraum in eine Menge konvexer Polygone. Dabei dürfen die konvexen Polygone einander nicht überschneiden, müssen aber an ihren Rändern aneinander grenzende Linien besitzen. Diese sind die Übergänge zwischen benachbarten Polygonen und werden hier als *Tore* oder *Gates* bezeichnet. Eine exemplarische Definition eines *Navigation Mesh* ist in Abb. 4.5 dargestellt.

Aus diesem *Navigation Mesh* lässt sich ein Graph generieren, wobei hier als Knoten die Zentren der *Gates* sowie die Schiffsposition zum Zeitpunkt der Pfadplanung und der Zielort verwendet werden. Kanten sind die Verbindungen der Knoten mit allen zum selben Polygon gehörenden anderen Knoten, wobei zu bedenken ist, dass ein *Gate* zu zwei Polygonen gehört. Dieser Graph ist für eine exemplarische Situation in Abb. 4.6 gezeigt.

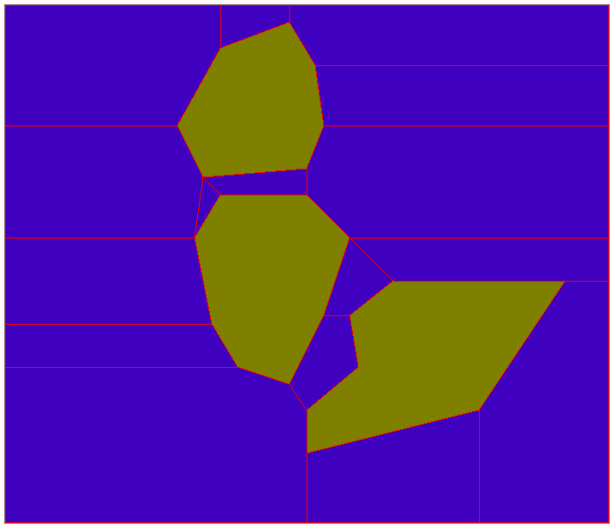


Abbildung 4.5: Beispiel einer *Navigation Mesh*-Definition auf einer exemplarischen Karte. Konvexe Polygone sind rot umrandet dargestellt.

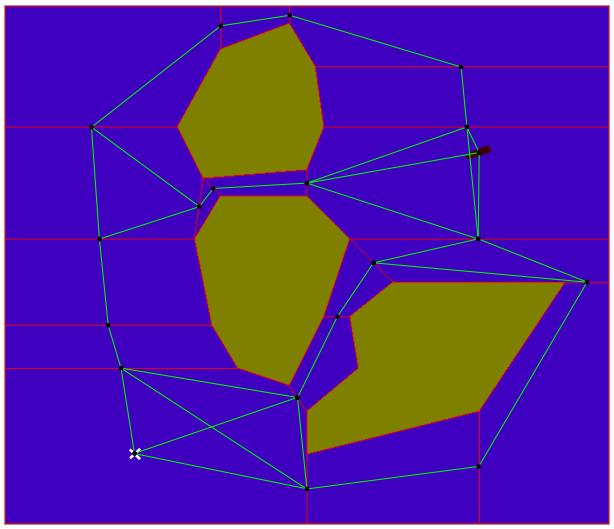


Abbildung 4.6: Beispiel für einen Graphen auf Basis einer *Navigation Mesh*-Definition für eine bestimmte Situation. Knoten (schwarze Punkte) sind die Schiffsposition, der Zielort (weißes Kreuz) sowie Mittelpunkte auf den *Gates*, Kanten sind grün dargestellt.

Auf einem so erzeugten Graph kann dann mithilfe eines Suchalgorithmus (e. g. A*, cf. Kapitel 3.3.2) nach Ermitteln des Polygons, in dem sich das Schiff befindet, und des Polygons, in dem der Zielort liegt, die benötigte Route ermittelt werden.

Dieser Ansatz, mithilfe eines *Navigation Mesh* eine Route zu erzeugen, ist — wie bereits in Kapitel 4.1.1.2 kurz angedeutet — abhängig von der Lage der konvexen Polygone. Sind diese schlecht modelliert (e. g. Abb. 4.7) kann das zu einer Route führen, auf der das lokale Verhalten nur schlecht arbeiten kann, da die Freiräume insbesondere in offenen Gewässern durch die Polygone stark eingeschränkt sind. Außerdem ist die Anzahl der Polygone in Abb. 4.7 höher als in Abb. 4.5, was zu einer höheren Komplexität des resultierenden Graphen und damit zu einem höheren Rechenaufwand führt.

Aufgrund dieser Relevanz der Definition des *Navigation Mesh* wird im Rahmen dieser Arbeit entschieden, dass die Polygone für das *Navigation Mesh* manuell erzeugt und vom Simulator

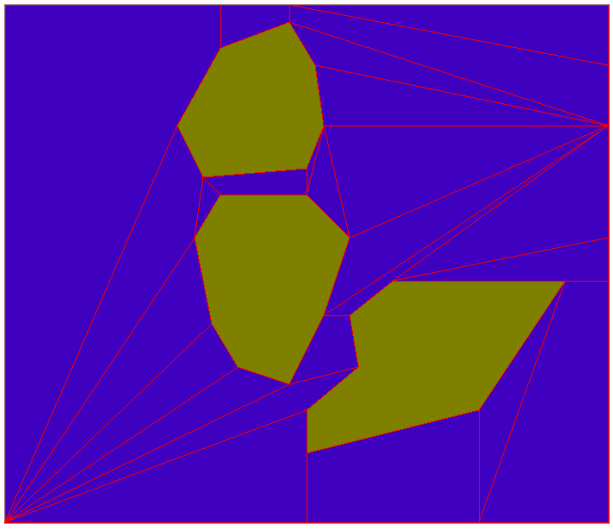


Abbildung 4.7: Beispiel einer schlechten *Navigation Mesh*-Definition auf einer exemplarischen Karte. Konvexe Polygone sind rot umrandet dargestellt.

dem autonomen Verhalten zur Verfügung gestellt werden müssen. Das automatische Generieren eines guten *Navigation Mesh* kann als Teil der Initialisierung des autonomen Verhaltens separat durchgeführt werden und dabei sogar die hier nicht beachteten Wassertiefen und Tiefgänge verschiedener Schiffe berücksichtigen. Aufgrund der klaren Definition, dass das *Navigation Mesh* von außen dem autonomen Verhalten zur Verfügung gestellt werden muss, lässt sich ein Generator für das *Navigation Mesh* entwickeln und ohne größere Schwierigkeiten in das Konzept aufnehmen, indem er während der Initialisierung vorgeschaltet die Umgebung verarbeitet und entsprechend Polygone erzeugt.

4.1.1.5 Lokales Verhalten

Das lokale Verhalten stellt den letzten Schritt des Kontrollzyklus dar und arbeitet auf Basis der vom globalen Planer gelieferten Route. Zusätzlich werden die Umgebungsdaten, i. e. das Szenario, wie Umweltdaten (e. g. Wind, Strömung, Sichtverhältnisse) oder andere Schiffe verwendet. Diese werden für ein Verhalten auf Basis der Kollisionsverhütungsregeln, die in diesem letzten Schritt herangezogen und verarbeitet werden, benötigt.

Die Kollisionsverhütungsregeln (cf. Kapitel 2.1) bestehen aus einer Menge mehr oder weniger unabhängiger Regeln, die beschreiben, wie sich Schiffe zu verhalten haben. Mehr oder weniger deshalb, weil einige Regeln für bestimmte Situationen gelten und damit unabhängig von anderen Regeln sind, die für andere Situationen gelten. Andere Regeln hingegen spezifizieren eine generelle Vorgabe für das Verhalten bei Greifen einer Regel (e. g. Verhalten des Kurshalters). Aufgrund dieser Tatsache soll innerhalb des lokalen Verhaltens eine modulare Architektur verwendet werden, wobei Regeln — sofern möglich — durch ein eigenes Modul abgebildet werden. Dieser Aufbau ist angelehnt an die parallele Kontrollarchitektur (cf. Kapitel 3.5.2) und ist schematisch in Abb. 4.8 dargestellt.

Wie bei parallelen Kontrollarchitekturen üblich besteht auch hier das Problem, die unabhängigen Module so zu kombinieren, dass keine widersprüchlichen Steuerungskommandos gegeben werden. Wie in Abb. 4.8 bereits skizziert, wird ein Ansatz zur Fusion der Ausgaben mithilfe eines einzigen Kombinationsmoduls gewählt, das die endgültigen Steuerungskommandos vorgibt und von den vorgeschalteten Verhaltensmodulen Daten über eine abstrakte Schnittstelle erhält.

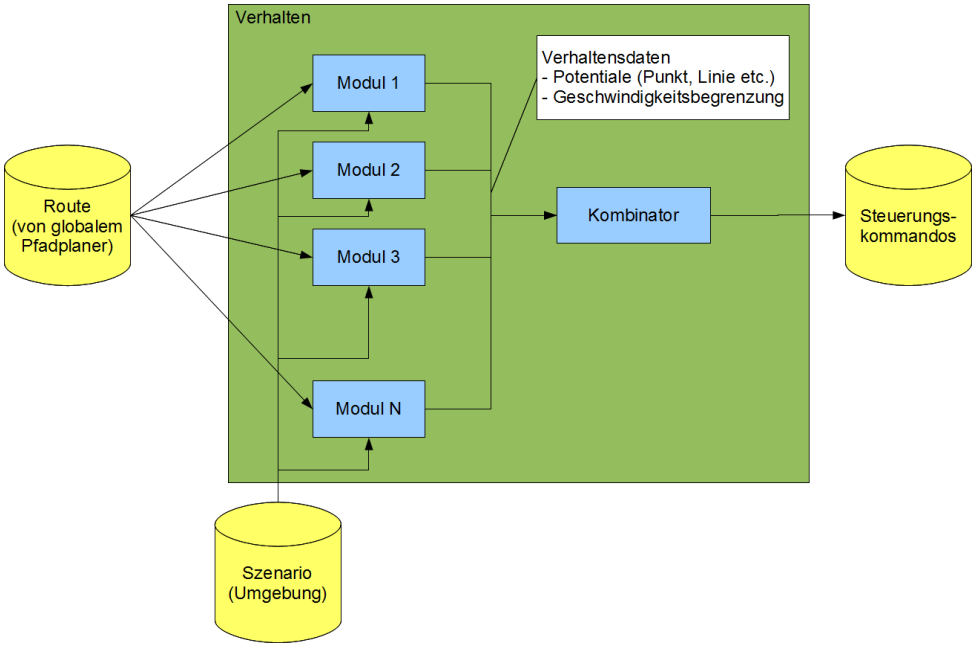


Abbildung 4.8: Schematischer Aufbau des lokalen Verhaltens.

In erster Linie geben die Kollisionsverhütungsregeln Richtungen vor, in denen sich das Objekt bewegen soll. Dies können Ausweichbewegungen sein mit einer vorgegebenen Richtung oder auch bevorzugte Fahrtrichtungen. Würden nun von jeder Regel unterschiedliche Vorschläge für Steuerungskommandos vorgegeben, ließen sich diese nur schwer kombinieren, da hier schlecht spezifiziert werden kann, ob zum Beispiel ein Steuerungskommando mit hoher Priorität übernommen oder mehrere Steuerungskommandos gemittelt werden sollen. Dies ist stark abhängig von der Situation. Stattdessen wird ein weniger diskreter Ansatz gewählt, der eine unpriorisierte Kombination leicht zulässt: Potentialfelder (cf. Kapitel 3.4.2).

Die Idee hinter diesem Ansatz liegt darin, dass die verschiedenen Regelmodule unabhängig voneinander Potentialquellen vorgeben können und das Kombinationsmodul anhand dieser und unter Berücksichtigung der kinematischen Beschränkungen den optimalen Weg ermittelt und ansteuert. Um ein autonomes Schiff zum Beispiel nach rechts ausweichen zu lassen, könnte eine abstoßende Potentialquelle, wie in Abb. 4.9 skizziert, vorgegeben werden. Die voraus liegende Linie soll ein Kreuzen vor dem Bug des anderen Schiffs verhindern und die an Backbord liegende Linie soll das autonome Schiff nach Steuerbord abdrängen. Es erlaubt zusätzlich bei zu starkem Einfluss abstoßender Potentiale eine Steuerung abzubrechen, also abzubremsen und stehen zu bleiben. Für diese allgemeine Schnittstelle zwischen Regelmodulen und Kombinationsmodul muss die aus der Literatur kommende Spezifikation eines Potentialfelds (cf. [5, S.138]) erweitert werden. So werden nicht nur das Ziel und Hindernisse als Potentialquellen verwendet, sondern beliebige anziehende und abstoßende Quellen müssen möglich sein. Außerdem wird das Ziel nicht wie üblich durch eine Funktion repräsentiert, die mit steigendem Abstand stärker wirkt, da die Entfernung zum Ziel sehr groß sein kann und somit Hindernisse zu stark überdecken kann. Neben diesen Anpassungen müssen außerdem verschiedene Arten von Potentialquellen möglich sein. Da hier durch andere Schiffe bewegliche Hindernisse berücksichtigt werden müssen, sind zeitvariante Potentialquellen notwendig, i. e. Potentialquellen, deren Position und Lage sich mit der Zeit verändern können. Die verschiedenen Arten benötigter Potentialquellen sollen hier nicht näher erläutert werden, stattdessen wird dies auf Kapitel 4.2.4 verschoben (cf. Abb. 4.19).

Aus dieser Berücksichtigung dynamischer Potentialquellen folgt, dass die hier verwendeten Po-

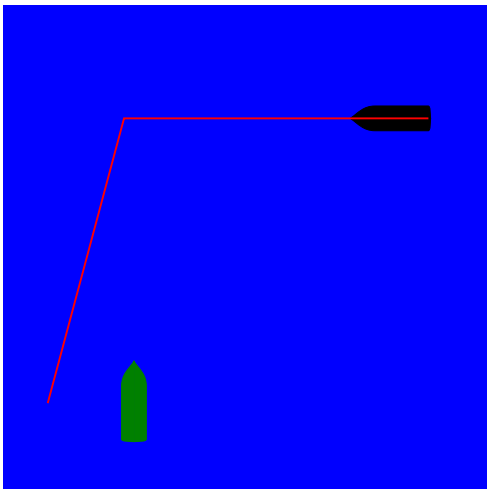


Abbildung 4.9: Mögliche Definition einer abstoßenden Potentialquelle (rote Linien), um das autonome Schiff (grün) nach rechts ausweichen zu lassen, e. g. bei kreuzenden Kursen nach KVR Regel 15.

tentialfelder als solche zeitvariant sind. Die auf das Objekt wirkende Kraft ist also nicht nur abhängig von der Position, sondern auch von dem Zeitpunkt, zu dem es sich dort befindet.

Neben den Potentialen in der Schnittstelle zum Kombinationsmodul muss allerdings noch eine Besonderheit in den Kollisionsverhütungsregeln berücksichtigt werden, die durch das Legen einer Potentialquelle nicht so einfach gelöst werden kann: Regel 6 (Sichere Geschwindigkeit; cf. Kapitel 2.1) beschränkt ein einzelnes Datum der Steuerungskommandos, nämlich die Geschwindigkeit. Dadurch wird das Potentialfeld als solches nicht beeinflusst, jedoch die Steuerungskommandos. Hierfür wird eine spezielle Implementierung der abstrakten Schnittstelle zum Kombinationsmodul vorgesehen, sodass beliebige Begrenzungen für die Geschwindigkeit vorgegeben werden können, die das Kombinationsmodul zu einer einzigen Begrenzung verbinden kann.

Nach dieser abstrakten Diskussion der Architektur innerhalb des lokalen Verhaltens werden im Folgenden zwei Konzepte für die Kombination und Berechnung der Steuerungskommandos im entsprechenden Modul vorgestellt. Der erste Ansatz war zunächst vorgesehen, wurde jedoch im Laufe der Umsetzung aufgrund von nicht haltbarem Rechenaufwand verworfen. Anschließend wurde der zweite Ansatz entwickelt und schließlich eingesetzt.

Kombinationsansatz: Suchbaum (verworfen)

Dieser Ansatz basiert auf der Idee, vom aktuellen Zustand des Schiffs (Position, Orientierung, Geschwindigkeit) aus einen Baum aufzuspannen, wobei die Knoten des Baums Zustände des Schiffs zu bestimmten Zeitpunkten sind. Die Zeitpunkte werden durch einen Zeitschritt Δt ermittelt, sodass Zustände zum Zeitpunkt $k \cdot \Delta t$ existieren für alle k mit $k \cdot \Delta t \leq t_{max}$. Die Zeit t_{max} ist also die Zeit, die das lokale Verhalten vorausplant. Zwischen zwei Knoten des Baums, i. e. für die Zeit Δt, wird hier angenommen, dass ein bestimmtes Steuerungskommando gilt. Es werden folgende mögliche Steuerungskommandos mit

- $a \in [a_{min}; a_{max}]$: Beschleunigung, minimale Beschleunigung (maximales Abbremsen), maximale Beschleunigung
- $\omega \in [\pm\omega_{max}]$: Winkelgeschwindigkeit (i. e. Drehrate), maximale Winkelgeschwindigkeit

beim Aufbau des Baums berücksichtigt:

- Mit gleicher Geschwindigkeit geradeaus fahren: $a = 0$, $\omega = 0$
- Mit gleicher Geschwindigkeit nach rechts drehen: $a = 0$, $\omega = -\omega_{max}$
- Mit gleicher Geschwindigkeit nach links drehen: $a = 0$, $\omega = \omega_{max}$
- Beschleunigen und geradeaus fahren: $a = a_{max}$, $\omega = 0$
- Beschleunigen und nach rechts drehen: $a = a_{max}$, $\omega = -\omega_{max}$
- Beschleunigen und nach links drehen: $a = a_{max}$, $\omega = \omega_{max}$
- Abbremsen und geradeaus fahren: $a = a_{min}$, $\omega = 0$
- Abbremsen und nach rechts drehen: $a = a_{min}$, $\omega = -\omega_{max}$
- Abbremsen und nach links drehen: $a = a_{min}$, $\omega = \omega_{max}$

Bestimmte Verzweigungen werden lediglich angewendet, wenn die kinematischen Beschränkungen dies noch zulassen. So wird das Beschleunigen und Abbremsen nur berücksichtig, sofern das Schiff nicht bereits an der entsprechenden kinematischen Beschränkung seiner Geschwindigkeit angelangt ist. Es ist offensichtlich, dass dies nur für kleine Zeitschritte Δt funktionieren kann, da sonst aufgrund der Verwendung der Extrema der kinematischen Beschränkungen zu grobe Pfade generiert werden (cf. Abb. 4.10). Eine sinnvolle Größe des Zeitschritts lässt sich anhand der maximalen Winkelgeschwindigkeit ω_{max} bestimmen, sodass sich das Schiff innerhalb eines Zeitschritts Δt zum Beispiel um 30° drehen kann.

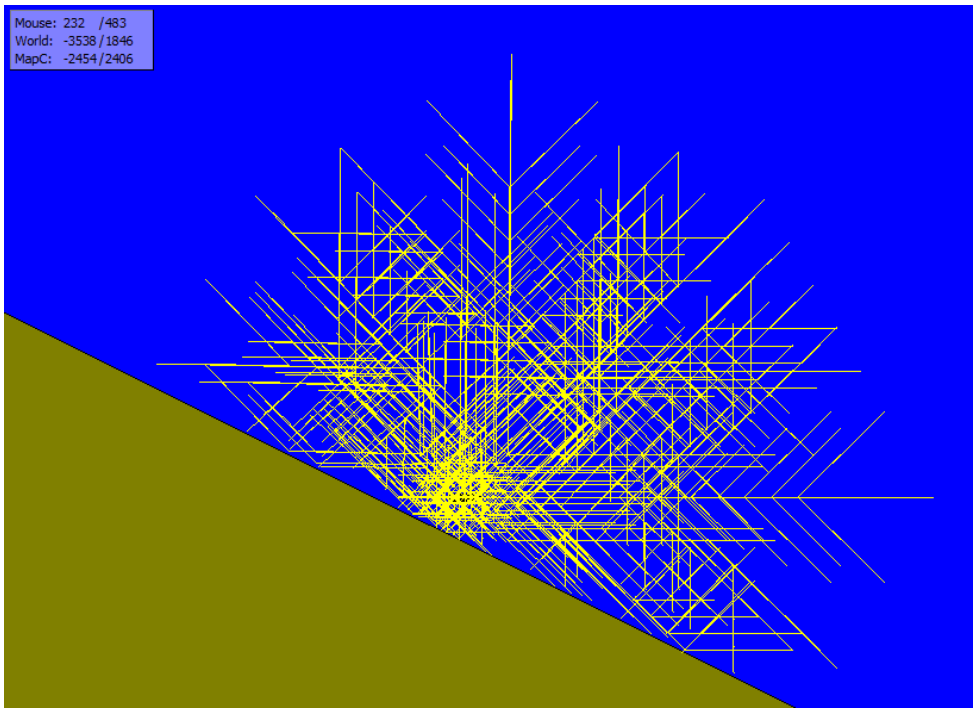


Abbildung 4.10: Problem der verwendeten Verzweigung für große Δt im Fall des Kombinationsansatzes: Suchbaum.

Nachdem der Baum aufgebaut ist, werden die Knoten bewertet. Hierfür wird zunächst eine Heuristik auf die unterste Ebene des Baumes, i. e. die Blätter, angewendet, um den Abstand des Punktes zum Zielort unter Berücksichtigung der Route näherungsweise zu bestimmen. Als

Heuristik — sofern die Route eine Länge von mehr als zwei hat — wird die Summe aus dem Abstand von der Position des untersuchten Knotens zum Mittelpunkt des übernächsten Gates und den Abständen zwischen den Mittelpunkten aller darauf folgender Gates benutzt. Für den Fall, dass die Länge der Route kleiner oder gleich zwei ist, wird einfach der Abstand zwischen Position des Knotens und dem Zielort verwendet. Seien also R die Route und R_i mit $i = \{0, \ldots, n\}$ die Elemente der Route, dann ist die Heuristik g für eine Position x für $len(R) > 2$:

$$g(x) = dist(x, pos(R_1)) + \sum_{i=1}^{n-2} dist(pos(R_i), pos(R_{i+1})) + dist(pos(R_{n-1}), pos(goal)). \quad (4.6)$$

Zusätzlich zu dieser Bewertung anhand des Abstands sollen die Knoten des Baums — nicht nur die Blätter — unter Berücksichtigung des zeitvarianten Potentialfelds bewertet werden. Beide Bewertungen sollen dann zur Ermittlung des optimalen Pfades durch ein Suchverfahren herangezogen werden.

Dieses Verfahren kann allerdings aufgrund nicht haltbaren Rechenaufwands nicht umgesetzt werden. Mit der oben stehenden Auflistung der Verzweigungen beim Aufbau des Baums, also einem Verzweigungsfaktor von $b = 9$, ist offensichtlich, dass der Baum nur über eine stark begrenzte Tiefe aufgebaut werden kann. Mit der zusätzlichen oben bereits genannten Problematik, dass die verwendete Verzweigung nur für kleine Δt annehmbare Ergebnisse liefert, kann der lokale Planer mit diesem Verfahren nur sehr kurz voraus planen, was bei den trägen kinematischen Eigenschaften von Schiffen zu Problemen führen kann. So dauert das Generieren des Suchbaums bei gewähltem $\Delta t = 5s$ und $t_{max} = 25s$, i. e. über eine Tiefe von $d = 5$ mit insgesamt $N = b^d = 9^5 = 59049$ Knoten, unoptimiert bereits knapp zehn Sekunden. Dabei ist ein Großteil der generierten Pfade im Allgemeinen nicht annähernd zielführend (cf. Abb. 4.11).

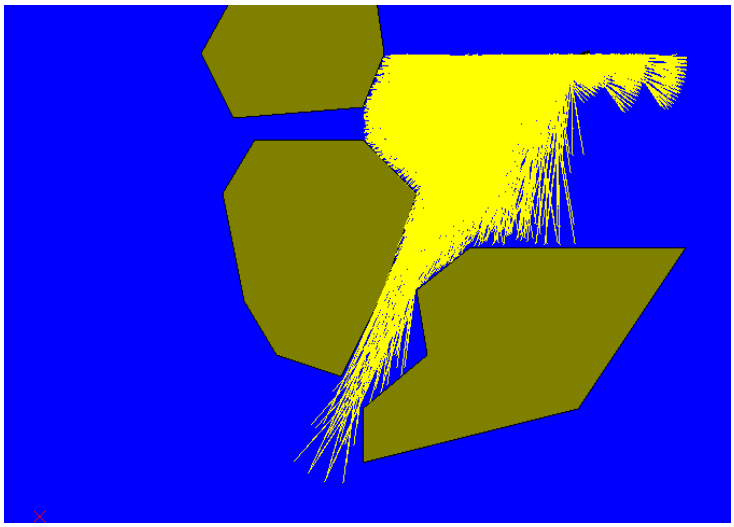


Abbildung 4.11: Problem vieler im Allgemeinen überflüssig generierter Pfade im Fall des Kombinationsansatzes: Suchbaum.

Kombinationsansatz: Potentialfeld

Nachdem der erste Ansatz verworfen ist, soll nun ein Verfahren vorgestellt werden, das die Vorteile eines Potentialfelds besser nutzt. Beim Suchbaum wird das Potentialfeld lediglich dafür vorgesehen, eine große Anzahl von möglichen Pfaden zu bewerten. Hier soll es hingegen verwendet werden, um einen initialen Pfad aufgrund der Potentiale und der damit wirkenden Kräfte anzupassen.

Die Idee, Zustände in bestimmten Abständen zu berechnen und so für eine gewisse Zeitspanne vorauszuplanen, bleibt aus dem ersten Ansatz bestehen. Die Zustände werden in einem Abstand von Δt ermittelt und es wird für eine Zeit t_{max} vorausgeplant. Es wird nun angenommen, dass in einem bestimmten Zustand, i. e. der aktuelle Zustand des Schiffs oder ein vorausgeplanter Zustand, ein von der Umgebung unabhängiges Steuerungskommando gegeben werden kann, mit dem eine optimale Annäherung an das nächste Ziel — der Zielort selbst oder das nächste *Gate*— erfolgt. Hier werden folgende drei diskrete Möglichkeiten unterschieden, wobei für das Fahren zum nächsten *Gate* andere Entscheidungsgrenzen gelten:

Ziel voraus. Das Ziel liegt gegenüber der Orientierung des aktuell weitergeplanten Zustands voraus, i. e. der relative Winkel ist klein ($\lesssim 30°$). In diesem Fall soll mit maximaler Geschwindigkeit — unter Berücksichtigung möglicher Nebenbedingungen für die Geschwindigkeit — auf das Ziel zugefahren werden.

Ziel liegt zur Seite. Das Ziel liegt gegenüber der Orientierung des aktuell weitergeplanten Zustands zur Seite, i. e. der relative Winkel ist etwas größer ($\gtrsim 30°$ und $\lesssim 60°$). In diesem Fall soll mit gleichbleibender Geschwindigkeit auf das Ziel eingedreht werden.

Ziel liegt weit zur Seite oder zurück. Das Ziel liegt gegenüber der Orientierung des aktuell weitergeplanten Zustands weit zur Seite oder hinter dem Schiff, i. e. der relative Winkel ist groß ($\gtrsim 60°$). In diesem Fall soll auf minimale Geschwindigkeit — unter Berücksichtigung möglicher Nebenbedingungen für die Geschwindigkeit — abgebremst und auf das Ziel eingedreht werden.

Aus diesen Fällen lassen sich Steuerungskommandos ableiten, die abhängig vom expandierten Zustand X_i zur Generierung des nächsten Zustands X_{i+1} verwendet werden. Anschließend werden die vom Potentialfeld auf den neuen Zustand X_{i+1} wirkenden Kräfte berechnet. Da das Potentialfeld — wie zu Beginn von Kapitel 4.1.1.5 beschrieben — zeitvariant ist, müssen die Kräfte abhängig von der Zeit $t = (i+1) \cdot \Delta t$ und der Position des Zustands $pos(X_{i+1})$ berechnet werden, also

$$F_{i+1} = F(t, \vec{x}) = F((i+1) \cdot \Delta t, pos(X_{i+1})). \tag{4.7}$$

Anhand der wirkenden Kraft F_{i+1} lässt sich der Zustand unter Berücksichtigung der kinematischen Beschränkungen des Schiffs entsprechend anpassen, sodass ein neuer Zustand $X_{i+1,1}$ ermittelt wird. Dieses Verfahren kann iterativ bis zu einer Abbruchbedingung angewendet werden. Die gewählte Abbruchbedingung bestimmt Laufzeit und Genauigkeit. So wird ein Zustand $X_{i,j}$ generiert, wobei i den vorausgeplanten Schritt vom aktuellen Zustand des Schiffs und j den Iterationsschritt der Verbesserung aufgrund des Potentialfeldes angibt.

Eine Visualisierung des beschriebenen Verbesserungsschritts durch die vom Potentialfeld wirkende Kraft ist in Abb. 4.12 gegeben. Der Unterschied einer nicht von einem Potentialfeld veränderten Folge von Zuständen und einer aufgrund eines wirkenden Potentialfeldes angepassten ist in Abb. 4.13 skizziert.

Es können nun bei Verwendung dieses Ansatzes Situationen auftreten, in denen das Schiff aufgrund seiner kinematischen Eigenschaften und aufgrund des Problems von Potentialfeldern, Fallen zu erzeugen (cf. Abb. 3.12), in ungültige Zustände gelangt. Dies kann durch die Vorausplanung über den Zeitraum t_{max} erkannt werden. Ungültige Zustände treten bei Verlassen der gegebenen Route, i. e. eine berechnete Position eines Zustands X_i liegt außerhalb der Polygone der Route, oder bei einer zu stark wirkenden Kraft, i. e. Summe der Beträge aller Kräfte, auf.

Um dieses Problem zu lösen, wird ein *Backtracking*-Algorithmus vorgesehen. Führt ein durch oben beschriebenes Verfahren ermittelter Pfad in einen ungültigen Zustand, werden schrittweise die Folge von Zuständen zurückgehend Alternativschritte berechnet. Ist also ein durch

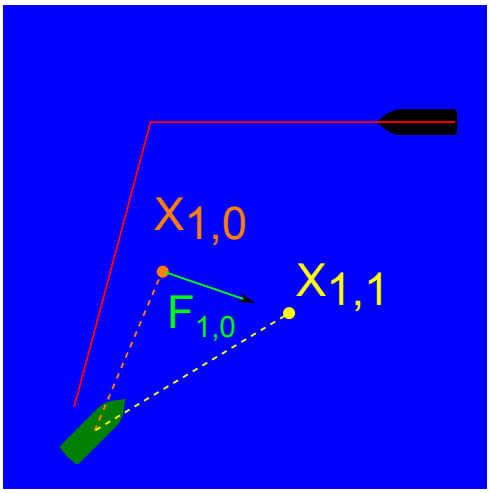
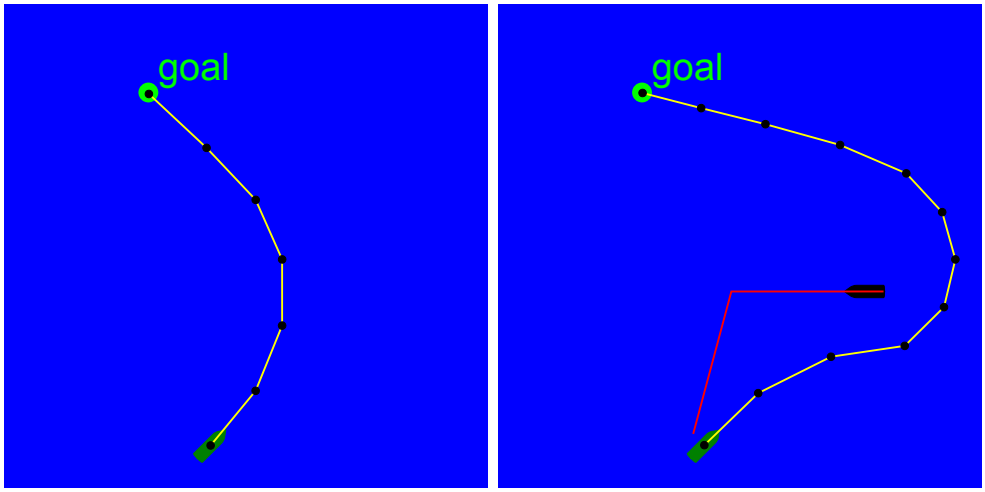


Abbildung 4.12: Visualisierung des Verbesserungsschritts eines vorausgeplanten Zustands aufgrund eines wirkenden Potentialfeldes beim *Kombinationsansatz Potentialfeld*. Die roten Linien stellen eine abstoßende Potentialquelle dar. Der orangene Punkt repräsentiert den unveränderten generierten Zustand $X_{1,0}$, auf den die grün dargestellte Kraft des Potentialfeldes wirkt. Der gelbe Punkt bildet den im ersten Iterationsschritt verbesserten Zustand $X_{1,1}$ ab.



(a) (b)

Abbildung 4.13: Darstellung einer vom *Kombinationsansatz Potentialfeld* generierten Folge von Zuständen (a) ohne wirkendes Potentialfeld und (b) bei Berücksichtigung eines wirkenden Potentialfeldes. Die roten Linien stellen eine abstoßende Potentialquelle dar.

Berücksichtigung des Potentialfeldes verbesserter Zustand $X_{i+1,n}$ ungültig, wird für den Zustand $X_{i,n}$ ein alternativer Folgezustand $X_{i+1,alt}$ berechnet. Ist dieser ebenfalls ungültig, wird der Zustand $X_{i,n}$ evenfalls ungültig, da er keinen gültigen Folgezustand ermöglicht. Entsprechend wird dann für den Zustand $X_{i-1,n}$ ein alternativer Folgezustand $X_{i,alt}$ ermittelt, der wiederum zunächst im Folgezustand das Potentialfeld berücksichtigt und bei dessen Ungültigkeit anschließend ein Alternativzustand untersucht wird. Dieser Algorithmus ist in Abb. 4.14 skizziert.

Der alternative Folgezustand $X_{i+1,alt}$ wird dabei anhand der Differenz zwischen dem Zustand in Richtung Ziel X_{i+1} und dem durch die aufgrund des Potentialfelds wirkenden Kräfte optimierten Zustand $X_{i+1,n}$ berechnet. Führt eine Optimierung also zu einer (stärkeren) Rechtsdrehung, i. e. in Richtung Steuerbord, gibt der alternative Folgezustand eine Linksdrehung vor. Führt eine Op-

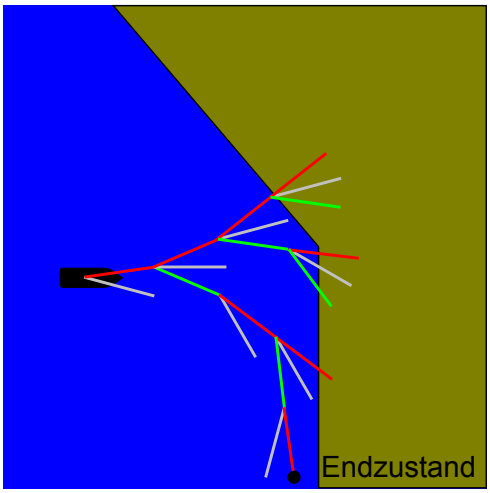


Abbildung 4.14: Skizze der generierten Pfade aufgrund des *Backtracking*-Algorithmus. Weiße Linien zeigen einen Schritt in Richtung Ziel, rote Linien diesen Schritt nach Optimierung durch ein hypothetisches Potentialfeld und grüne Linien den Schritt zum alternativen Folgezustand. Zustände in der Landmasse (ockerfarben) sind ungültig.

timierung zu einer (stärkeren) Beschleunigung, gibt der alternative Folgezustand eine negative Beschleunigung vor. Der Alternativzustand wird also entgegen der vom Potentialfeld vorgegebenen Kraft verschoben. Dabei wird allerdings nicht die Position der Zustände verglichen, sondern die dynamischen Daten für Kurs und Geschwindigkeit bzw. davon abgeleitet die Rotationsgeschwindigkeit und Beschleunigung. Zusätzlich müssen die kinematischen Beschränkungen des Schiffs berücksichtigt werden.

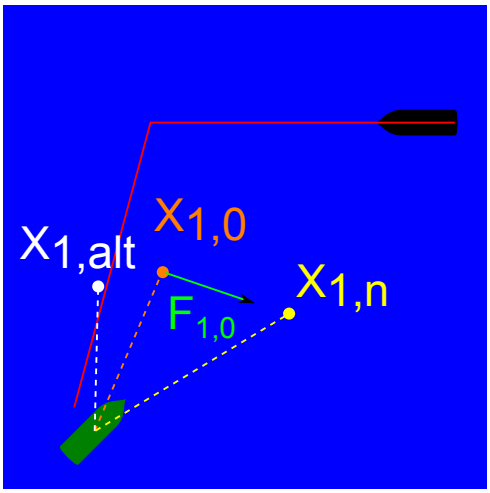


Abbildung 4.15: Visualisierung der Bildung des alternativen Folgezustands $X_{1,alt}$ als Teil des *Backtracking*-Algorithmus aufgrund eines wirkenden Potentialfeldes beim Kombinationsansatz: Potentialfeld. Die roten Linien stellen eine abstoßende Potentialquelle dar. Der orangene Punkt repräsentiert den unveränderten generierten Zustand $X_{1,0}$, auf den die grün dargestellte Kraft des Potentialfeldes wirkt. Der gelbe Punkt bildet den verbesserten Zustand $X_{1,n}$ ab. Der weiße Punkt repräsentiert den alternativen Folgezustand $X_{1,alt}$.

Die Bildung des alternativen Folgezustands $X_{i+1,alt}$ ist konzeptionell in Abb. 4.15 dargestellt. In dem gezeigten Beispiel wird davon ausgegangen, dass die Optimierung des Zustands $X_{1,0}$ zu einer Erhöhung der Geschwindigkeit Δv und einer Änderung des Kurses um $\Delta\varphi$ im Zu-

stand $X_{1,n}$ führt. Dann wird der alternative Folgezustand $X_{1,alt}$ gebildet durch Änderung der Geschwindigkeit um $-\Delta v$ und eine Änderung des Kurses um $-\Delta\varphi$ gegenüber des Zustands $X_{1,0}$.

4.1.2 Simulatorschnittstelle

Die Simulatorschnittstelle besteht aus zwei Teilen: eine Schnittstelle für den Simulator und eine für die Oberfläche desselben. Letztere soll hier nicht erläutert werden, da sie für diese Arbeit weniger interessant ist und lediglich eine softwaretechnische Herausforderung darstellt. Die Schnittstelle zum Simulator selbst teilt sich wiederum in zwei Aspekte: eine Datenschnittstelle (cf. Abb. 4.2), die das Szenario bzw. die Umgebung für ein bestimmtes Schiff befüllen sowie die Steuerungskommandos abrufen und in den Simulator überführen muss, und eine funktionale Schnittstelle, mit der das autonome Verhalten vom Simulator aus aufgerufen werden kann.

Für die funktionale Schnittstelle werden folgende Anforderungen gestellt:

Kontrolle über ein Schiff an das autonome Verhalten übergeben. Es muss möglich sein, dass der Simulator die Steuerung über ein bestimmtes Schiff an das autonome Verhalten übergibt. Dadurch muss eine entsprechende Instanz für dieses Schiff angelegt und der Kontrollzyklus vorbereitet werden.

Kontrolle über ein Schiff vom autonomen Verhalten nehmen. Es muss analog zum obigen Punkt möglich sein, das autonome Verhalten für ein bestimmtes Schiff zu deaktivieren und so wieder unter manuelle Kontrolle zu stellen.

Auslösen eines Zyklus. Da das autonome Verhalten eingebettet im Simulator läuft und von diesem gesteuert wird, muss es getaktet aufgerufen werden können. Dadurch muss die Datenschnittstelle bezüglich Umgebung (siehe unten) befüllt, der Kontrollzyklus ausgelöst und die Datenschnittstelle bezüglich Steuerungskommandos (siehe unten) gelesen werden.

Mitteilung über das Löschen eines Schiffs. Um unnötige Speicherhaltung zu vermeiden, muss das autonome Verhalten über das Löschen eines Schiffs informiert werden. Falls dieses noch Kontrolle über das gelöschte Schiff hat, muss die entsprechende Kontrolle beendet werden.

Die Datenschnittstelle bezüglich Umgebung benötigt alle relevanten Daten, um Entscheidungen auf Basis der Kollisionsverhütungsregeln treffen zu können. Hier sind dies die folgenden Daten:

- *Navigation Mesh*
- autonom gesteuertes Schiff
- andere Simulationsobjekte (hier nur Schiffe)
- Umwelt: Wind, Strömung, Sichtverhältnisse
- Verkehrstrennungsgebiete

Für die Datenschnittstelle der Steuerungskommandos werden die folgenden Daten definiert (cf. Kapitel 2.2.3.2):

- Zielgeschwindigkeit
- Beschleunigung
- Zielorientierung (im Zweidimensionalen)
- Winkelgeschwindigkeit (i. e. Drehrate im Zweidimensionalen)

4.2 Umsetzung

In diesem Kapitel wird die Umsetzung des oben beschriebenen Konzepts näher ausgeführt. Dabei wird die Realisierung der Software-Architektur hier nicht näher beschrieben. Zunächst werden allgemeine, für alle Module zur Verfügung stehende Mechanismen beschrieben, die von den Modulen des Kontrollzyklus genutzt werden. Anschließend wird auf die Umsetzung der einzelnen Schritte des Kontrollzyklus eingegangen.

4.2.1 Grundlegende Mechanismen

Es werden einige grundlegende Mechanismen entwickelt, die für alle Module zur Verfügung stehen und die Umsetzung des Kontrollzyklus vereinfachen.

4.2.1.1 Kommunikation zwischen Simulations- und Benutzerschnittstellenmodul

Ein bereits im Konzept (cf. Kapitel 4.1) angedeuteter notwendiger Mechanismus ist die Kommunikation zwischen einem Simulationsmodul und seiner zugehörigen Benutzerschnittstelle, die ebenfalls als eigenständiges Modul realisiert ist. Hierzu wird ein *Observer Pattern* implementiert, welches einen Datenstrom an den Beobachter übergibt. Module können sich für einen identifizierten Datenstrom anmelden, sodass durch eine Identifikation, die aus Modul- und Schiffsidentifikation besteht, eine eindeutige Zuordnung des Datenstroms zwischen Simulations- und Benutzerschnittstellenmodul getroffen werden kann.

4.2.1.2 Zeichnungen

Ein weiterer hilfreicher Mechanismus ist die Möglichkeit, aus Simulationsmodulen heraus Zeichnungen erstellen zu können, die in der Oberfläche des Simulators dargestellt werden. Die Module erstellen ein Objekt mit Informationen, die für die Darstellung relevant sind, und die Oberfläche hat die Möglichkeit, diese ein- und auszublenden. Es gibt eine vordefinierte Menge von Objekten, die gezeichnet werden können (e. g. Punkt, Linie, Menge von Polygonen).

4.2.2 Auftragsplaner

Entsprechend des in Kapitel 4.1.1.3 entwickelten Konzepts ist der Auftragsplaner modular aufgebaut, wobei die Module untereinander unabhängig sind. Da in dieser Arbeit der Fokus auf den folgenden beiden Schritten des Kontrollzyklus liegt, wird lediglich eine simple Realisierung des Auftragsplaners entwickelt. Diese besteht aus einem einzigen Auftragsplanermodul mit einem zugehörigen Modul für die Benutzerschnittstelle, die über den in Kapitel 4.2.1.1 vorgestellten Mechanismus miteinander kommunizieren.

Dieses Modul — es wird als Modul *MoveToTarget* bezeichnet — erwartet von der Benutzerschnittstelle eine Eingabe aller relevanten Daten, um die Schnittstelle zum globalen Routenplaner, i. e. den Zielort (cf. Kapitel 4.1.1.2), zu befüllen. Die Oberfläche zur Steuerung des Moduls hat also eine Eingabe für die Gültigkeit der Zielposition sowie eine Eingabe für die Zielposition $\vec{x}_{target} = \begin{pmatrix} x \\ y \end{pmatrix}$ selbst. Diese Daten werden vom Simulationsmodul in die Schnittstelle zum globalen Routenplaner geschrieben. Eine exemplarische Vorgabe eines Zielortes ist — unter Zuhilfenahme des in Kapitel 4.2.1.2 vorgestellten Mechanismus — in Abb. 4.16 dargestellt.

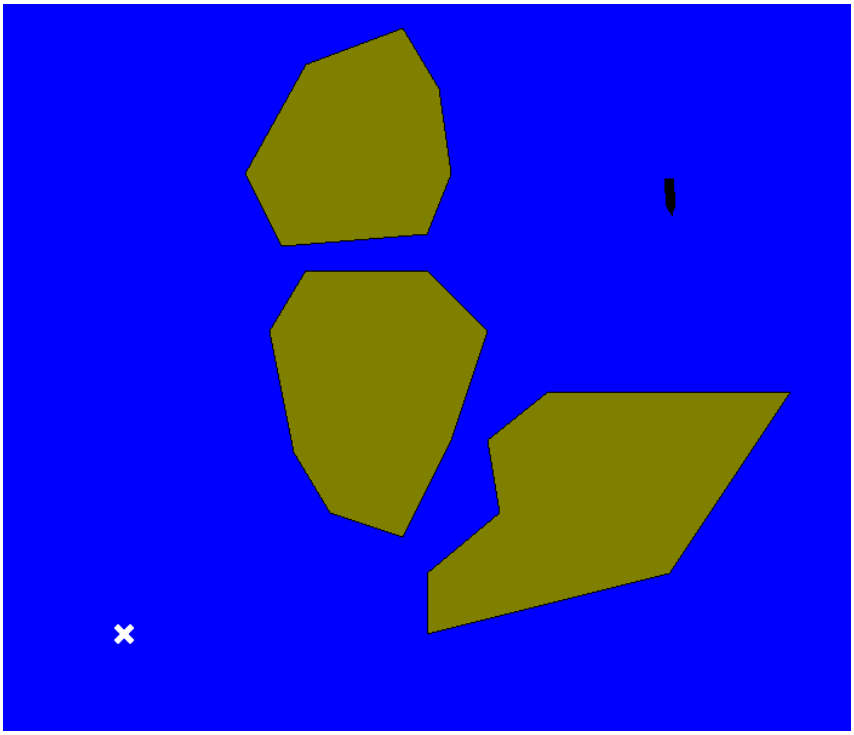


Abbildung 4.16: Beispiel einer Vorgabe eines Zielortes (weißes X) durch das Auftragsplanermodul *MoveToTarget*.

4.2.3 Globaler Routenplaner

Für den globalen Routenplaner wird eine Implementierung geschaffen, die den A*-Algorithmus (cf. Kapitel 3.3.2) zum Finden der optimalen Route auf Basis des *Navigation Mesh* nutzt.

Das Modul besitzt zwei relevante Aufrufe: Initialisierung und Aktualisierung. In der Initialisierung wird ein Graph $G_{NavMesh}$ aufgebaut, der das *Navigation Mesh* repräsentiert und später als Grundlage für den in Kapitel 4.1.1.4 beschriebenen und in Abb. 4.6 dargestellten Graphen G_{Route} verwendet wird. Der für die Suche benötigte Graph wird nicht direkt aufgebaut, da zu diesem Zeitpunkt weder das Schiff, noch der Zielort betrachtet werden. Stattdessen beschreibt dieser zunächst generierte Graph die Polygone des *Navigation Mesh* sowie ihre Nachbarschaften untereinander. Die Knoten repräsentieren die Polygone, die Kanten stellen Nachbarschaftsbeziehungen zwischen Polygonen dar und sind mit Informationen über das *Gate* zwischen je zwei benachbarten Polygonen markiert. Der Algorithmus zur Initialisierung ist in Alg. 2 aufgeführt.

In der Aktualisierung wird geprüft, ob eine Pfadplanung notwendig ist, und, falls dies zutrifft, wird eine neue Route geplant. Eine neue Planung muss durchgeführt werden, wenn sich der Zielort ändert oder wenn sich das autonome Schiff in einem anderen Polygon des *Navigation Mesh* befindet als zum Zeitpunkt des vorigen Aufrufs. Letzteres kann bedeuten, dass das Schiff manuell versetzt wurde oder es ist durch Folgen der Route in dieser vorangeschritten. Nicht berücksichtigt wird hier der Fall, dass das autonome Schiff manuell innerhalb desselben Polygons versetzt wird, wodurch möglicherweise ein anderer Pfad kürzer ist. Die Planung einer Route erfolgt durch Aufbauen eines Suchbaums aus dem Graphen G_{Route} auf Basis des Graphen $G_{NavMesh}$ unter Verwendung des A*-Algorithmus. Als Heuristik wird der euklidische Abstand eines Knoten v zum Zielort *goal*

$$h(\vec{x}_v, \vec{x}_{goal}) = dist(\vec{x}_v, \vec{x}_{goal}) = \sqrt{(x_v - x_{goal})^2 + (y_v - y_{goal})^2} \tag{4.8}$$

Algorithmus 2 Initialisierung des globalen Routenplaners. M ist das *Navigation Mesh* mit $M = \{P_0, P_1, \ldots, P_{n-1}\}$. V ist ein Knoten des Graphen $G_{NavMesh}$ mit $V.P$ als Polygon des *Navigation Mesh*, $V.x_{center}$ als Mittelpunkte des Polygons $V.P$ und einer Menge von Nachbarschaftsbeziehungen, i. e. Kanten dieses Knoten. G ist ein *Gate*.

```
function Initialize(M)
    i ← 0;
    for i < n do
        V.P ← P_i;
        V.x_center ← center(P_i);
        G_NavMesh.add(V);
    end for
    j ← 0;
    for j < size(G_NavMesh) do
        k ← j + 1;
        for k < size(G_NavMesh) do
            G ← gate(G_NavMesh[j].P, G_NavMesh[k].P);
            if G then
                G_NavMesh[j].addNeighbor(G_NavMesh[k], G);
                G_NavMesh[k].addNeighbor(G_NavMesh[j], G);
            end if
        end for
    end for
end function
```

verwendet. Es ist leicht zu sehen, dass diese Heuristik unterschätzend ist, also eine zulässige Heuristik ist. Als Pfadkosten wird die Länge einer Kante verwendet, i. e. der euklidische Abstand zwischen zwei Knoten v_i und v_j,

$$g(\vec{x}_{v_i}, \vec{x}_{v_j}) = dist(\vec{x}_{v_i}, \vec{x}_{v_j}). \tag{4.9}$$

Es wird beim Aufbauen des Suchbaums außerdem verhindert, dass Pfade mehrfach durchlaufen werden. Dazu werden bereits besuchte *Gates* gemerkt und ein erneutes Besuchen verhindert. Der Algorithmus zur Berechnung einer Route ist in Alg. 3 dargestellt.

Die gefundene Route wird dann an die Schnittstelle zum lokalen Verhalten übergeben. Ist kein gültiger Zielort gegeben oder kann keine gültige Route gefunden werden, so gibt der globale Routenplaner eine leere Route. Die visuelle Darstellung einer gefundenen Route für die in Abb. 4.16 definierte Situation ist in Abb. 4.17 zu sehen.

4.2.4 Lokaler Planer

Wie in Kapitel 4.1.1.5 erläutert, wird der lokale Planer modular aufgebaut, wobei ein spezielles Modul, das *Kombinationsmodul*, zur Berechnung der Steuerungskommandos anhand der von den anderen Modulen gegebenen Potentialquellen und Geschwindigkeitseinschränkungen verwendet wird. Das bedeutet, dass eine Vorgabe der Reihenfolge, in der die Module je Kontrollzyklus aufgerufen werden, möglich sein muss. Außerdem müssen die Ausgangsdaten der Module an das Kombinationsmodul weitergereicht werden können. Diese Daten werden hier als *Verhaltensdaten* (bzw. *BehaviorData*) bezeichnet und in Kapitel 4.2.4.1 näher beschrieben.

Anschließend werden die entwickelten Module näher beleuchtet. Hier wird unterschieden zwischen allgemeinen Modulen (cf. Kapitel 4.2.4.2), die unabhängig von den Kollisions-

Algorithmus 3 Berechnung einer Route im globalen Routenplaner mithilfe des A*-Algorithmus durch Generieren eines Suchbaums T. x_{own} ist die Position des autonomen Schiffs, x_{goal} die Zielposition, R die resultierende Route, V bezeichnet Knoten des Graphen $G_{NavMesh}$ und N repräsentiert den aktuellen Nachbarn eines Knoten.

function $calculatePath(x_{own}, x_{goal}, G_{NavMesh})$
  $V_{own} \leftarrow getPolygon(x_{own})$;
  $V_{goal} \leftarrow getPolygon(x_{goal})$;
  if $invalid(V_{own})$ **or** $invalid(V_{goal})$ **then**
    return R; // Leere Route
  else if $V_{own} = V_{goal}$ **then**
    $R.add(V_{own}.P)$;
  else
    $h_{root} \leftarrow distance(x_{own}, x_{goal})$;
    // createNode(Knoten des Graphen, Heuristik, Pfadkosten, Vaterknoten im Suchbaum)
    $T.root \leftarrow createNode(V_{own}, h_{root}, 0, 0)$;
    $T.addNode(T.root)$
    for all $neighbors(V_{own})$ **do**
      $h \leftarrow distance(center(N.G), x_{goal})$;
      $g \leftarrow distance(x_{own}, center(N.G))$;
      $V \leftarrow createNode(N.V, h, g, T.root)$;
      $T.addNode(V)$;
      $T.root.addChild(V)$;
    end for
    // Solange noch nicht expandierte Knoten im Baum vorhanden sind.
    while $T.leafs.size > 0$ **do**
      if $T.leafs.first.V = V_{goal}$ **then**
        $Break$;
      end if
      $expand(T, x_{goal})$; // Expandiere den nächsten Knoten im Baum.
    end while
    if $pathFound(T)$ **then**
      $R \leftarrow getPath(T)$; // Generiere Pfad vom gefundenen Zielknoten zur Wurzel.
    end if
  end if
  return R;
end function

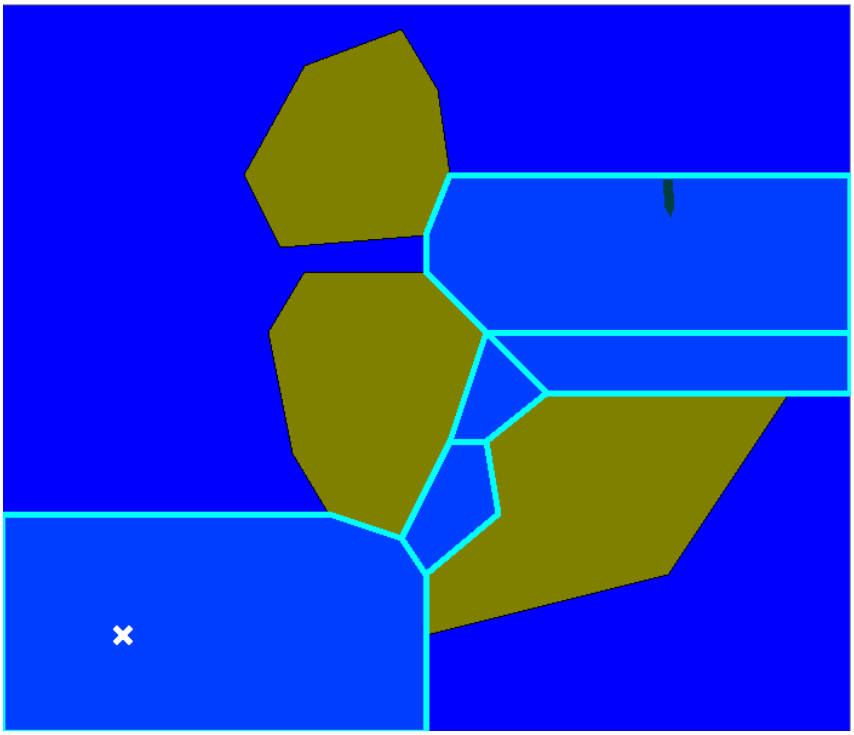


Abbildung 4.17: Visuelle Darstellung einer gefundenen Route (transparente Polygone) durch den A*-Routenplaner.

verhütungsregeln sind, Regelmodulen (cf. Kapitel 4.2.4.3) zur Abbildung der Kollisionsverhütungsregeln und dem Kombinationsmodul (cf. Kapitel 4.2.4.4).

4.2.4.1 Verhaltensdaten

Als Verhaltensdaten werden die Daten bezeichnet, die zwischen Modulen des lokalen Planers ausgetauscht werden. Insbesondere soll dies der Kommunikation zum Kombinationsmodul dienen, allerdings wird ein Datenaustausch zwischen den Modulen nicht verhindert. Diese Verhaltensdaten haben lediglich eine Gültigkeit für einen Aufruf des Kontrollzyklus, i. e. beim nächsten Auslösen des Kontrollzyklus sind alle zuvor abgelegten Daten gelöscht. Dadurch wird verhindert, dass veraltete Daten die Steuerung des Schiffs beeinflussen. Es folgt, dass Module — auf diesem Wege — ausschließlich Daten von solchen Modulen erhalten können, deren Verarbeitung zuvor innerhalb des aktuellen Kontrollzyklusaufrufs ausgelöst wurde. Es folgt weiterhin, dass das Kombinationsmodul zuletzt aufgerufen werden muss.

Bereits in Kapitel 4.1.1.5 sind die auszutauschenden Daten spezifiziert. Dabei handelt es sich um Potentialquellen verschiedener Arten und um die Vorgabe einer Geschwindigkeitsbegrenzung. Die gewählte Architektur zur Implementierung dieser Daten ist in Abb. 4.18 skizziert. Der Typ der Daten, die ein bestimmtes Element enthält, wird in der Basisklasse über die Variable *type* identifiziert. Die Geschwindigkeitsbegrenzung enthält lediglich eine untere und eine obere Schranke.

Wie in der Abbildung bereits zu sehen ist, gibt es einen Potentialtyp, der die Art der Funktion beschreibt, nach der die Stärke des Potentials in einem bestimmten Abstand berechnet wird. Die Stärke — unabhängig von der Richtung — in Abhängigkeit von der Distanz ist für den

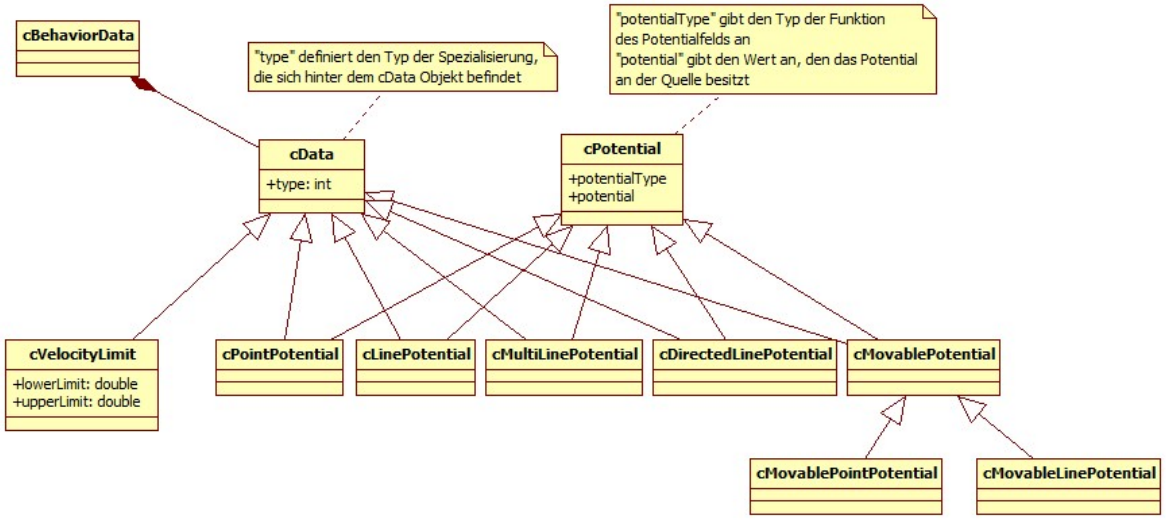


Abbildung 4.18: Modellierung der Verhaltensdaten im lokalen Planer.

verwendeten Potentialtyp

$$F(c,d) = \begin{cases} c \cdot (1 - d/|c|) & \text{für } d \leq c \\ 0 & \text{sonst} \end{cases} \tag{4.10}$$

mit d als Abstand zur Potentialquelle und c als Stärke direkt an der Potentialquelle. Die Funktion ist also eine lineare Interpolation zwischen Potentialquelle und Abstand d. Die Stärke der Quelle hat damit gleichzeitig einen Einfluss auf die Reichweite der Wirkung des Potentials.

Abhängig vom Typ der Potentialquelle werden die Richtung, in die das Potential wirkt, und der Abstand eines gegebenen Punkts zur Potentialquelle unterschiedlich bestimmt. Im Folgenden werden die Typen von Potentialquellen unterschieden, die zusätzlich in Abb. 4.19 schematisch visualisiert sind:

Punktpotential. Die Potentialquelle ist ein Punkt $\vec{x}_{src}$. Die Richtung der Kraft kann durch Berechnung des Vektors von diesem Punkt zum geprüften Punkt $\vec{x}$ ermittelt werden. Der Winkel dieses Vektors gibt die Richtung und die Länge den Abstand an.

Linienpotential. Die Potentialquelle ist eine Linie, die durch zwei Punkte begrenzt und definiert wird. Es wird der Punkt $\vec{x}_{src}$ auf dieser Linie ermittelt, der den geringsten Abstand zum geprüften Punkt $\vec{x}$ besitzt. Anschließend verhält es sich analog zum Punktpotential.

Mehrlinienpotential. Das Mehrlinienpotential wird durch eine Menge von Linien definiert. Für alle Linien werden analog zum Linienpotential die resultierenden Kraftvektoren ermittelt und aufsummiert. Daraus lässt sich die Richtung der wirkenden Kraft angeben. Bei der Prüfung der einzelnen Linien wird der minimale Abstand d_{min} gemerkt und dieser wird später verwendet, um den Abstand des geprüften Punkts $\vec{x}$ festzulegen.

Gerichtetes Linienpotential. Die Potentialquelle ist eine Linie, die ausschließlich für anziehende Potentiale verwendet werden soll. Die Idee dabei ist, mit einem statischen Potential eine Richtung vorzugeben, das aber nach Überfahren keine Wirkung mehr hat, da das Schiff sonst möglicherweise umdrehen könnte, wenn das anziehende Potential hinter ihm liegt. Daraus folgt, dass das Potential gleich 0 ist, wenn der geprüfte Punkt $\vec{x}$ hinter der

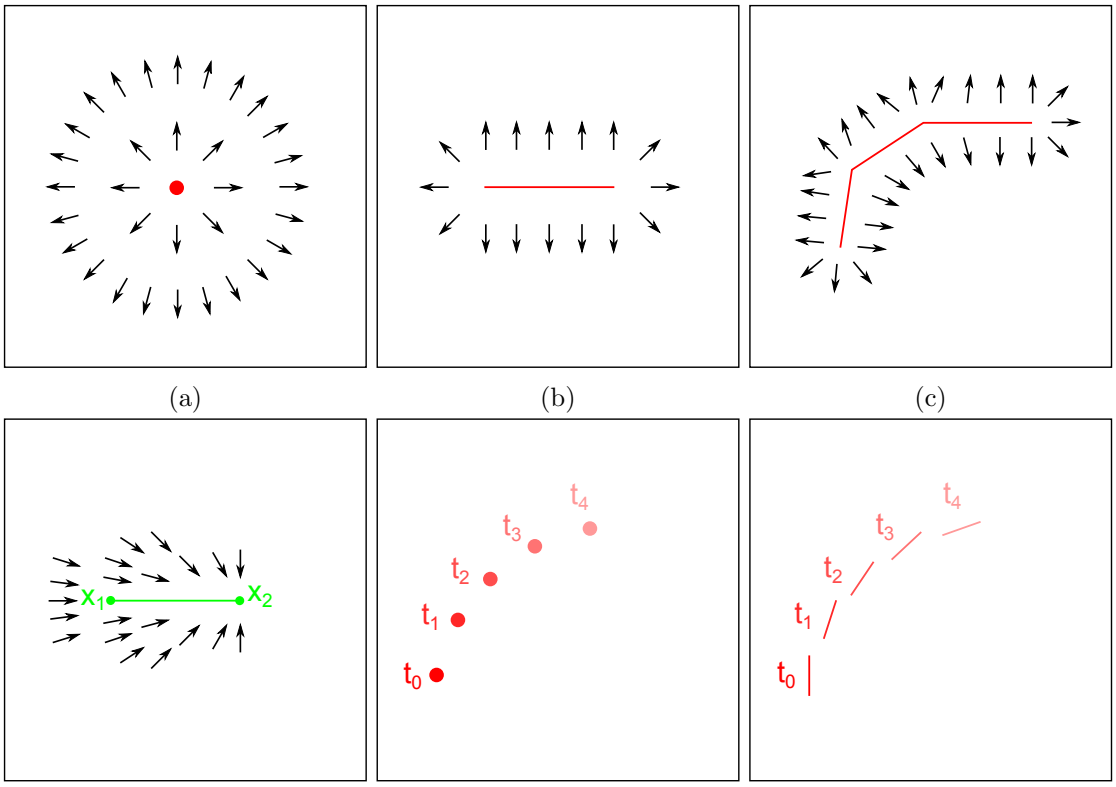


Abbildung 4.19: Skizzierung der wirkenden Potentialfelder für verschiedene Quellentypen: (a) Punktpotential, (b) Linienpotential, (c) Mehrlinienpotential, (d) Gerichtetes Linienpotential, (e) Bewegliches Punktpotential und (f) Bewegliches Linienpotential.

Linie liegt. Andernfalls wird der Abstand über den minimalen Abstand zwischen Linie und geprüftem Punkt analog zum Linienpotential ermittelt. Die Richtung des wirkenden Potentials ist immer zum Endpunkt der Linie hin.

Bewegliches Punktpotential. Die Potentialquelle verhält sich analog zum Punktpotential mit dem Unterschied, dass durch Angabe von kinematischen Parametern (Geschwindigkeit, Beschleunigung, Orientierung, Winkelgeschwindigkeit) die Position der Punktquelle abhängig von einer Zeitdifferenz Δt ist.

Bewegliches Linienpotential. Die Potentialquelle stellt eine Mischung aus dem Linienpotential — zur Berechnung des wirkenden Potentials — und dem beweglichen Punktpotential dar. Anhand der gegebenen kinematischen Daten sowie der Zeitdifferenz Δt wird zunächst die Positionsänderung des Zentrums der Linie analog zum beweglichen Punktpotential festgestellt. Anschließend wird eine Rotation der Linie durchgeführt.

4.2.4.2 Allgemeine Module

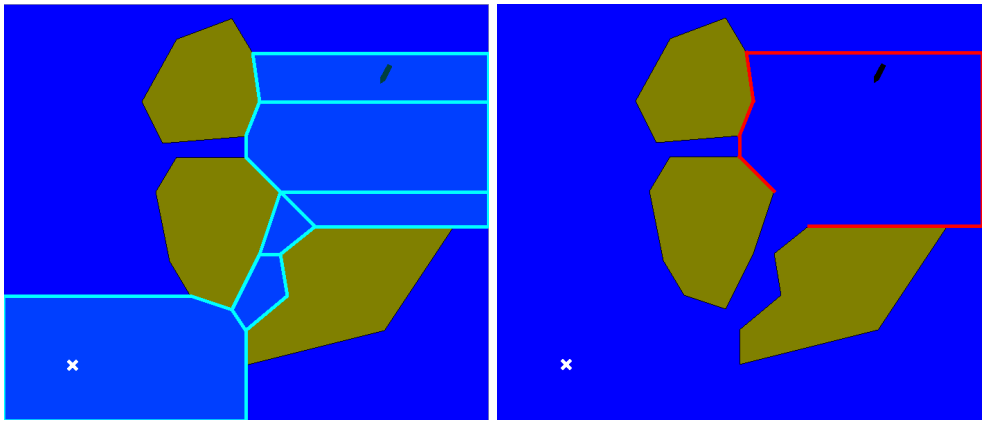
In diesem Abschnitt sollen die regelunabhängigen Module beschrieben werden. In diese Kategorie fallen zwei Module: Routengrenzen und Zielort.

Routengrenzen

Das autonome Schiff soll der Route folgen, ohne diese zu verlassen. Ein Verlassen wird hier untersagt, da davon ausgegangen wird, dass an den Grenzen der Polygone Küstenlinien sein können. Daher sollen die Grenzen der Route abstoßend wirken. Es müssen alle Kanten der Polygone von Routenelementen berücksichtigt werden, bei denen es sich nicht um ein *Gate* der Route handelt. Diese werden dann in ein einziges abstoßendes *Mehrlinienpotential* (cf. Kapitel 4.2.4.1) überführt.

Um auch bei beliebig langen Routen keine zu große Rechenintensität zu fordern, wird für eine parametrisierbare Zeit vorausgeplant, die gemeinsam mit der aktuellen Geschwindigkeit des Schiffs eine maximale Distanz für zu berücksichtigende Polygone angibt.

Eine exemplarische Lage der Potentialquelle des Mehrlinienpotentials ist in Abb. 4.20 dargestellt.



(a) (b)

Abbildung 4.20: Exemplarische Lage für ein Mehrlinienpotential durch das Modul *Routengrenzen*: (a) aktuelle Route; (b) Potentialquelle.

Zielort

Ein weiteres regelunabhängiges Hilfsmodul sorgt für die Markierung des Zielortes als anziehendes Potential. Entgegen dem üblichen Ansatz, eine Funktion zu wählen, die im Nahbereich kaum und mit steigender Entfernung stärker wirkt (cf. Gleichung (3.5)), wird hier analog zu den abstoßenden Potentialen Gleichung (4.10) verwendet. Da die Steuerung zum Ziel hin — wie in Kapitel 4.1.1.5 erläutert — auf anderem Wege geschieht, muss das anziehende Potential zum Ziel nicht in weiter Entfernung wirken. Es wird lediglich dazu benutzt, um eine Abstoßung durch in geringer Entfernung hinter dem Ziel liegende Potentiale zu vermeiden, sodass der Zielort auch nahe einer Küste angegeben werden kann. Das Potential wird als Punktpotential realisiert.

4.2.4.3 Regelmodule

In diesem Teil werden die implementierten Regelmodule erläutert. Die Regeln 5 (Ausguck), 7 (Möglichkeit der Gefahr eines Zusammenstoßes), 8 (Manöver zur Vermeidung von Zusammenstößen), 16 (Maßnahmen des Ausweichpflichtigen) und 19 (Verhalten von Fahrzeugen bei verminderter Sicht) werden nicht durch eigenständige Module realisiert, da sie entweder allgemeine Richtlinien zum Verhalten vorgeben, die von anderen Regeln berücksichtigt werden

müssen (7, 8, 16), für den Rahmen dieser Arbeit keine Relevanz besitzen (5) oder zusätzliche Anforderungen stellen, deren Umsetzung zu komplex ist (19).

Jede Regel wird durch ihre in Kapitel 2.1.2 geschriebene und hier wiederholte Zusammenfassung eingeleitet. Während von den unten näher erläuterten Regeln die Nummern 6, 9 und 10 individuelle Verfahren erfordern, können die übrigen (12, 13, 14, 15, 17, 18) ähnlich betrachtet werden. Diese zuletzt genannten Regeln lassen sich immer in folgende Schritte aufteilen und sind entsprechend ähnlich aufgebaut (cf. Alg. 4):

1. Feststellen, ob die Regel greift, i. e. Beginn einer Situation, und Entscheidung des Verhaltens
2. Reaktion auf die Situation, i. e. Aktualisieren von Potentialen
3. Prüfung auf Ende einer Situation

Algorithmus 4 Grundlegender Algorithmus für die Aktualisierung von Regelmodulen, die auf Zustandsüberwachung von Situationen basieren (i. e. 12, 13, 14, 15, 17, 18). e_{own} bezeichnet das autonome Schiff, E_{other} ist die Menge anderer Schiffe.

```
function update(e_own, E_other)
    i ← 0;
    for i < size(E_other) do
        mustAvoid ← FALSE;
        if isInSituation(E_other[i]) then
            if checkForSituationEnd(e_own, E_other[i]) then
                removeSituation(E_other[i]);
            else
                processSituation(e_own, E_other[i]);
            end if
        else if checkForSituationStart(e_own, E_other[i], mustAvoid) then
            addSituation(e_own, E_other[i], mustAvoid);
            processSituation(e_own, E_other[i]);
        end if
    end for
end function
```

Als *Situation* wird hier ein Zustand bezeichnet, in dem eine Regel beachtet werden muss. Eine bestimmte Situation gilt nur zwischen dem autonomen Schiff und einem anderen. Es können allerdings mehrere Situationen gleichzeitig bestehen, i. e. das autonome Schiff ist gleichzeitig in Situationen mit mehreren anderen Schiffen. Jede Regel hat dabei ihre eigene Zustandsüberwachung, es können also auch Situationen zwischen dem autonomen Schiff und einem anderen in verschiedenen Regelmodulen gleichzeitig existieren. Eine Situation ist erst vorbei, wenn die regelspezifische Prüfung auf Ende der Situation erfüllt ist.

Dies lässt sich also als eine Menge von Zustandsautomaten modellieren, wobei es je entsprechender Regel und je anderem Schiff einen Zustandsautomaten gibt mit den Zuständen *keine Situation* und *Situation* und mit den Zustandsübergängen *Beginn einer Situation* und *Ende einer Situation*. Alg. 4 realisiert diese Modellierung.

Regel 6: Sichere Geschwindigkeit

Regelzusammenfassung: *Diese Regel beschränkt die Geschwindigkeit von Fahrzeugen nach oben hin abhängig von der jeweiligen Situation.*

Das Modul zur *Regel 6: Sichere Geschwindigkeit* ist hier das einzige zur Beschränkung der Geschwindigkeit. Zur Prüfung, ob eine Beschränkung vorgegeben werden muss, müssen entsprechend der Regel (cf. Kapitel 2.1.2) verschiedene Umweltbedingungen berücksichtigt werden. Um die Komplexität gering und das Verfahren als solches im Fokus zu halten, werden hier nur die Sichtverhältnisse — abhängig von dem im Simulator verwendeten Wert (cf. Kapitel 2.2.2) — vereinfacht und die Verkehrsdichte berücksichtigt.

In Abhängigkeit von den Sichtverhältnissen $\tau \in [0.0; 1.0]$ wird die Begrenzung der Geschwindigkeit nach

$$v_{limit} = v_{max} - (1 - \tau)^2 \cdot v_{max} \tag{4.11}$$

mit v_{max} als Maximalgeschwindigkeit des Schiffs berechnet.

Für die Berücksichtigung der Verkehrsdichte werden andere Schiffe innerhalb einer maximalen Entfernung von d_{max} berücksichtigt. Der verwendete Algorithmus ist in Alg. 5 dargestellt. Danach werden alle nahen Schiffe einbezogen ($\frac{d_{max}}{2}$) und die weiter entfernten nur dann, wenn sie sich nicht vom autonomen Schiff wegbewegen. Die Stärke, mit der die Schiffe in die Verkehrsdichte eingehen, ist linear abhängig vom Abstand zum autonomen Schiff.

Regel 9: Enge Fahrwasser

Regelzusammenfassung: *Fahrzeuge innerhalb eines engen Fahrwassers müssen sich so weit rechts wie möglich halten und abhängig von der Art und Größe von Fahrzeugen müssen diese anderen Fahrzeugen Vorfahrt gewähren.*

Diese Regel soll dazu führen, dass sich das Schiff in engen Fahrwassern, e. g. Flüsse, relativ zu seiner Fahrtrichtung rechts hält. Dazu wird auf der vorgegebenen Route (cf. Kapitel 4.1.1.2) operiert und für die Polygone der Routenelemente geprüft, ob es sich um ein enges Fahrwasser handelt. Hier wird die Annahme getroffen, dass die Polygone des *Navigation Mesh* möglichst optimiert modelliert sind, i. e. große freie Flächen sind durch wenige große Polygone repräsentiert. Dann kann die Breite der Routenelemente und die Breite der *Gates* verwendet werden, um zu bestimmen, ob ein Polygon als enges Fahrwasser gilt.

Damit ein Polygon als enges Fahrwasser gilt, müssen zwei Bedingungen gelten: Die Breite des eingehenden *Gate* und des ausgehenden *Gate* müssen kleiner als ein Schwellwert $b_{gate,max}$ sein und die Breite des Polygons darf einen bestimmten Schwellwert $b_{poly,max}$ nicht überschreiten. Die Prüfung der Breite der *Gates* stellt kein Problem dar, die Ermittlung der maximalen Breite des Polygons jedoch ist schwieriger.

Die maximale Breite wird hier wie folgt berechnet. Sei $\vec{x}_{gate}$ der Vektor vom Zentrum des eingehenden *Gate* zum Zentrum des ausgehenden *Gate*. Dann werden zunächst die Vektoren vom Zentrum des eingehenden *Gate* zu allen Punkten des Polygons relativ zur Richtung von $\vec{x}_{gate}$ ermittelt und danach aufgeteilt, ob der Punkt links oder rechts vom Vektor $\vec{x}_{gate}$ liegt. Anschließend wird für den Punkt, der so am weitesten links liegt, eine Linie L_{left} berechnet, die den Vektor $\vec{x}_{gate}$ orthogonal schneidet und als Endpunkte den betrachteten Punkt sowie den Schnitt mit dem Rand des Polygons auf der anderen Seite von $\vec{x}_{gate}$ besitzt. Analog wird dies für den am weitesten rechts liegenden Punkt zur Berechnung der Linie L_{right} durchgeführt. Die Längen dieser beiden Linien L_{left} und L_{right} werden dann zur Prüfung der maximalen Breite des Polygons verwendet.

Algorithmus 5 Ermittlung der Geschwindigkeitsbegrenzung aufgrund von Verkehrsdichte. e_{own} bezeichnet die Daten des autonomen Schiffs, E_{other} ist die Menge anderer Schiffe, $e.pos$ ist die Position eines Objekts, $e.crs$ sein Kurs, $e.hdg$ die Orientierung. Parameter $b_{lower} = 2.0$ und $b_{upper} = 7.0$ bezeichnen Parameter für Entscheidungsschwellwerte und $v_{b,min}$ eine minimale obere Schranke für die Geschwindigkeit.

function $checkTraffic(e_{own}, E_{other}, d_{max})$
  $sum \leftarrow 0$;
  $n \leftarrow 0$;
  $i \leftarrow 0$;
  for $i < size(E_{other})$ **do**
    $d \leftarrow distance(e_{own}.pos, E_{other}[i].pos)$;
    if $d < d_{max}$ **then**
      if $d < 0.5 \cdot d_{max}$ **then**
        $n \leftarrow n + 1$;
        $sum \leftarrow sum + \left(1 - \frac{d}{d_{max}}\right)$;
      else
        if $isLeft(E_{other}[i])$ **and** $turnsRight(e_{own})$ **then**
          // Kurs des Fremdschiffs relativ zur Orientierung des autonomen Schiffs.
          $\alpha \leftarrow E_{other}[i].crs - e_{own}.hdg$;
          if not ($\alpha \geq 45°$ **and** $\alpha \leq 135°$) **then**
            // Schiff muss sich auf das autonome Schiff zubewegen
            $n \leftarrow n + 1$;
            $sum \leftarrow sum + \left(1 - \frac{d}{d_{max}}\right)$;
          end if
        else if $isRight(E_{other}[i])$ **and** $turnsLeft(e_{own})$ **then**
          // Kurs des Fremdschiffs relativ zur Orientierung des autonomen Schiffs.
          $\alpha \leftarrow E_{other}[i].crs - e_{own}.hdg$;
          if not ($\alpha \leq -45°$ **and** $\alpha \geq -135°$) **then**
            // Schiff muss sich auf das autonome Schiff zubewegen
            $n \leftarrow n + 1$;
            $sum \leftarrow sum + \left(1 - \frac{d}{d_{max}}\right)$;
          end if
        else
          $n \leftarrow n + 1$;
          $sum \leftarrow sum + \left(1 - \frac{d}{d_{max}}\right)$;
        end if
      end if
    end if
  end for
  if $sum \geq b_{lower}$ **then**
    if $sum \geq b_{upper}$ **then**
      $v_{limit} \leftarrow v_{b,min}$;
    else
      // Interpolation zwischen Begrenzungen in Abhängigkeit von sum.
      $v_{limit} \leftarrow v_{b,min} + \left(1 - \frac{sum - b_{lower}}{b_{upper} - b_{lower}}\right) \cdot (v_{max} - v_{b,min})$;
    end if
  end if
end function

Wird unter Berücksichtigung dieser beiden Grenzwerte festgestellt, dass ein Polygon als enges Fahrwasser gilt, wird für dieses Polygon ein abstoßendes Linienpotential erzeugt. Dazu wird auf den beiden *Gates* je ein Punkt ermittelt, der relativ zur Fahrtrichtung des autonomen Schiffs in der Mitte zwischen der linken Grenze des *Gate* und dessen Zentrum liegt. Diese beiden Punkte definieren dann die für das Potential verwendete Quelle. Ein Beispiel für diese Regel ist in Abb. 4.21 visualisiert.

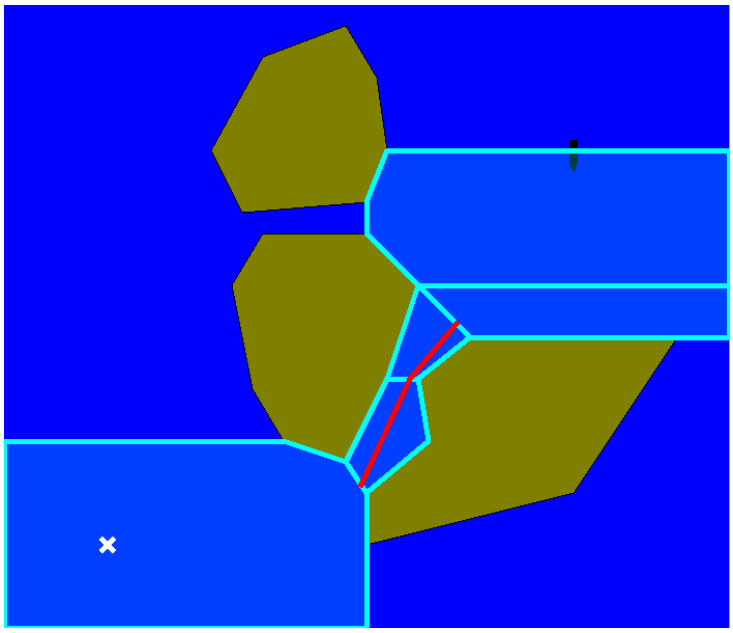


Abbildung 4.21: Beispiel für die Wirkung der Anwendung von *Regel 9: Enge Fahrwasser*. Die Route ist durch türkise, transparente Polygone dargestellt, die von der Regel erzeugten Linienpotentiale durch rote Linien.

Regel 10: Verkehrstrennungsgebiete

Regelzusammenfassung: *Es gibt bestimmte definierte Bereiche, sogenannte Verkehrstrennungsgebiete, in denen Fahrzeuge auf durch eine Trennzone separierten Einbahnwegen fahren. Das Ein- und Auslaufen in einen solchen Einbahnweg und der Aufenthalt innerhalb einer Küstenverkehrszone statt innerhalb eines solchen Bereichs sind beschränkt. Queren eines Verkehrstrennungsgebietes ist möglich, wenn bestimmte Maßnahmen getroffen werden. Außerdem gelten innerhalb dieser Gebiete spezielle Vorfahrtsregeln.*

Verkehrstrennungsgebiete (engl. *Traffic Separation Area*, *TSA*) sind Bereiche, in denen, durch eine Trennzone separiert, der Verkehr jeweils nur in einer Richtung fließt. Diese Bereiche sollen bevorzugt genutzt werden, anstatt an diesen vorbeizufahren. Zusätzlich können sie senkrecht zur Fahrtrichtung passiert werden. Die Bereiche liegen in der in Kapitel 2.2.2 beschriebenen Form vor.

Unter Berücksichtigung der Route als grundlegende Information über den Bereich, in dem sich das Schiff bewegen darf, müssen zunächst drei Fälle unterschieden werden.

Außerhalb. Liegt das Gebiet komplett außerhalb der Route, i. e. kein Teil des Gebiets überschneidet sich mit einem Polygon der Route, dann kann das Gebiet ignoriert werden.

Innerhalb. Liegt das Gebiet komplett innerhalb der Route, muss individuell geprüft werden, welche Rolle es in der aktuellen Situation spielt. Dieser Fall wird unten genauer beschrieben.

Teilweise innerhalb. Liegt das Gebiet nur zum Teil innerhalb der Route, wird vereinfacht angenommen, dass es nicht durchfahren werden kann. In diesem Fall werden die Ränder des rechteckigen Verkehrstrennungsgebiets durch vier abstoßende Linienpotentiale abgebildet.

Im Folgenden wird nun der Fall gänzlich innerhalb der Route liegender Verkehrstrennungsgebiete untersucht. Dazu wird zunächst ein spezieller Abstandswert des autonomen Schiffs zu allen Verkehrstrennungsgebieten berechnet. Der spezielle Abstandswert ist so definiert, dass er auch negativ sein kann. Dadurch wird angegeben, dass das Gebiet hinter der zu fahrenden Strecke des Schiffs liegt. Ist das Schiff innerhalb des Verkehrstrennungsgebiets, ist der Abstand gleich 0. Andernfalls gilt Folgendes:

Sei $\vec{x}_{own}$ die Position des autonomen Schiffs und $\vec{x}_{tgt}$ die nächste Zielposition, i. e. Zielort oder Zentrum des nächsten *Gate*. Seien weiterhin $\vec{x}_{tsa,1}$ und $\vec{x}_{tsa,2}$ die vordere und hintere Randposition des Verkehrstrennungsgebiets und A das Gebiet selbst. Dann wird der Abstand entsprechend Alg. 6 ermittelt. Abb. 4.22 stellt die so berechneten Abstände visuell dar.

Algorithmus 6 Berechnung des Abstands eines Verkehrstrennungsgebiets. Der x-Wert eines Vektors ist positiv voraus und negativ nach hinten relativ zur Richtung des Vektors.

function $tsaDistance(\vec{x}_{own}, \vec{x}_{tgt}, \vec{x}_{tsa,1}, \vec{x}_{tsa,2}, A)$
  if $isInside(\vec{x}_{own}, A)$ **then**
    return 0;
  end if
  $\vec{x} \leftarrow \vec{x}_{tgt} - \vec{x}_{own}$;
  $\vec{x}_1 \leftarrow \vec{x}_{tsa,1} - \vec{x}_{own}$;
  $\vec{x}_2 \leftarrow \vec{x}_{tsa,2} - \vec{x}_{own}$;
  $\vec{x}_1.rotate(\angle\vec{x})$;
  $\vec{x}_2.rotate(\angle\vec{x})$;
  if $\vec{x}_1.x < 0$ **and** $\vec{x}_2.x < 0$ **then**
    return $min(\vec{x}_1.x, \vec{x}_2.x)$;
  else if $\vec{x}_1.x < 0$ **or** $\vec{x}_2.x < 0$ **then**
    $mean \leftarrow (\vec{x}_1.x + \vec{x}_2.x)/2$;
    if $mean < 0$ **then**
      return $mean$;
    end if
  end if
  return $distance(\vec{x}_{own}, A)$;
end function

Wird nun ein negativer Abstand zurückgegeben, i. e. das Gebiet liegt hinter der zu fahrenden Strecke Richtung Ziel, wird das Gebiet analog zu einem teilweise innerhalb der Route liegenden Gebiet behandelt. Es wird also durch vier abstoßend wirkende Linienpotentiale umgeben, da es für die zu fahrende Route keinen Gewinn, sondern im Gegenteil einen Verlust bringen würde, da das Gebiet dann gänzlich durchfahren werden müsste und das Schiff so weiter vom Ziel wegbringen würde. Für alle Verkehrstrennungsgebiete mit positivem Abstand — oder Abstand gleich 0 — wird das Gebiet mit kleinstem Abstand gewählt und verarbeitet. Die anderen werden vorerst ignoriert, um zu verhindern, dass mehrere anziehend wirkende Gebiete gleichzeitig das autonome Schiff beeinflussen.

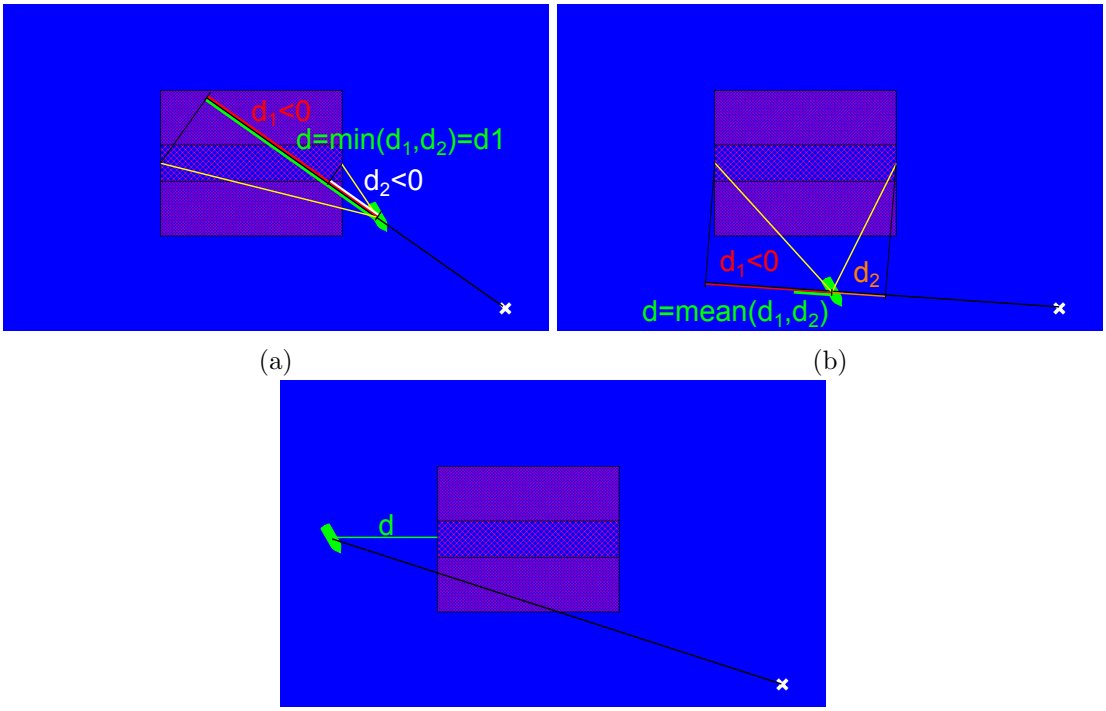


Abbildung 4.22: Skizzierung der verschiedenen berechneten Abstände vom Verkehrstrennungsgebiet bei Regel 10: (a) falls Ein- und Ausgangspunkt hinter dem Schiff relativ zum Weg zum Ziel liegen, (b) falls Ein- oder Ausgangspunkt hinter dem Schiff relativ zum Weg zum Ziel liegen und der Mittelwert kleiner null ist, (c) andernfalls.

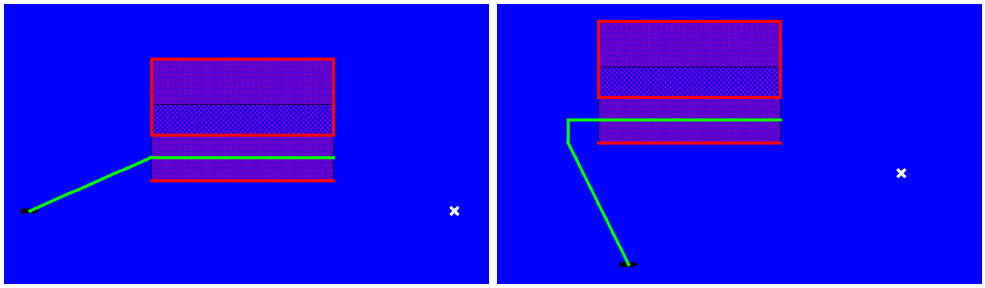
Bei der Verarbeitung des gewählten Gebiets wird die Idee verwendet, über das gerichtete Linienpotential ein anziehendes Potential von der Position des autonomen Schiffs zur Einfahrt und innerhalb des Gebiets von der Einfahrt zur Ausfahrt zu legen. Dadurch wird die Richtung vorgegeben, in die das Schiff fahren soll. Gleichzeitig werden verbotene Bereiche durch abstoßende Linienpotentiale umgeben.

Es wird zunächst unterschieden zwischen den beiden Fällen, dass das Schiff in korrekter Weise hindurchfährt oder dieses senkrecht kreuzt. Dies geschieht durch Vergleich der intendierten Fahrtrichtung zum Ziel mit der Lagerichtung des Verkehrstrennungsgebiets. Für die Durchfahrt wird dann das gerichtete Linienpotential von der Position des Schiffs zur Einfahrt gelegt. Befindet sich das Schiff bereits parallel zum Verkehrstrennungsgebiet, werden mehrere gerichtete Linienpotentiale durch Einfügen von Hilfspunkten erzeugt. Beim Kreuzen des Gebiets wird anhand eines Grenzwertparameters entschieden, ob das Gebiet tatsächlich gekreuzt werden muss oder besser umfahren werden kann. Die verschiedenen Fälle mit den entsprechend gelegten Potentialen sind in Abb. 4.23 visualisiert.

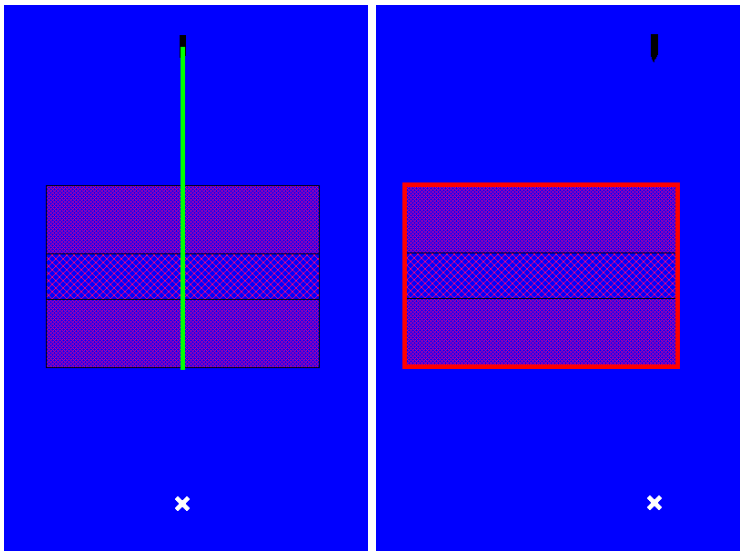
Regel 12: Segelfahrzeuge

Regelzusammenfassung: *Bei Annäherung von Segelschiffen wird die Vorfahrt abhängig von der Windrichtung relativ zur Orientierung der beteiligten Schiffe bestimmt.*

Diese Regel greift nur, falls das autonome Schiff ein Segelfahrzeug ist. Außerdem werden nur die anderen Schiffe betrachtet, die ebenfalls Segelfahrzeuge sind. Wie bereits zu Beginn dieses Kapitels erwähnt sind dieses und die folgenden Regelmodule ähnlich aufgebaut. Der zyklisch



(a) (b)



(c) (d)

Abbildung 4.23: Erzeugte Potentiale (anziehende gerichtete Linienpotentiale (grün), abstoßende Linienpotentiale (rot)) für das Verhalten bei einem Verkehrstrennungsgebiet: (a) Durchfahrt; (b) Durchfahrt mit Hilfspunkten für Einfahrt; (c) Kreuzen; (d) Umfahren.

aufgerufene Algorithmus ist in Alg. 4 abgebildet.

Die Regeln unterscheiden sich also nur in den zu Beginn dieses Kapitels aufgelisteten drei Schritten, die hier jeweils genauer beschrieben werden.

Situationsbeginn. Zunächst sollen nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. es muss gelten

$$d < d_{max} = s_{own} + s_{other} = v_{own} \cdot t_{max} + v_{other} \cdot t_{max} \tag{4.12}$$

wobei d der euklidische Abstand zwischen den Schiffen ist, v die jeweils aktuelle Geschwindigkeit bezeichnet und t_{max} ein Parameter ist. Anschließend wird geprüft, ob sich das andere Schiff auf das autonome zubewegt oder von diesem entfernt. Für den Beginn einer Situation muss es sich auf dieses zubewegen. Zuletzt wird der *Closest Point of Approach* (*CPA*) berechnet. Dieser Algorithmus berechnet den Punkt, an dem bei aktueller Geschwindigkeit und Fahrtrichtung der kleinste Abstand zum anderen Schiff vorliegt. Dieser Abstand d_{cpa} muss kleiner sein als ein Schwellwert $d_{cpa,max}$.

Der Abstand am *CPA* wird berechnet durch

$$d_{cpa} = |\vec{x}_{cpa1} - \vec{x}_{cpa2}| \tag{4.13}$$

wobei $\vec{x}_{cpa1}$ der *CPA* des ersten und $\vec{x}_{cpa2}$ der *CPA* des zweiten Schiffs ist mit

$$\vec{x}_{cpa1} = \vec{x}_1 + t_{cpa} \cdot \vec{v}_1 \tag{4.14}$$

$$\vec{x}_{cpa2} = \vec{x}_2 + t_{cpa} \cdot \vec{v}_2 \tag{4.15}$$

und

$$t_{cpa} = \frac{\vec{x}_1 \cdot \vec{v}_2 + \vec{x}_2 \cdot \vec{v}_1 - \vec{x}_1 \cdot \vec{v}_1 - \vec{x}_2 \cdot \vec{v}_2}{(\vec{v}_1 - \vec{v}_2)^2}. \tag{4.16}$$

Treffen all diese Bedingungen zu, wird dann abhängig von den Windrichtungen ermittelt, ob das autonome Schiff ausweichen muss oder nicht. Diese Entscheidung gilt für die gesamte Dauer der beginnenden Situation.

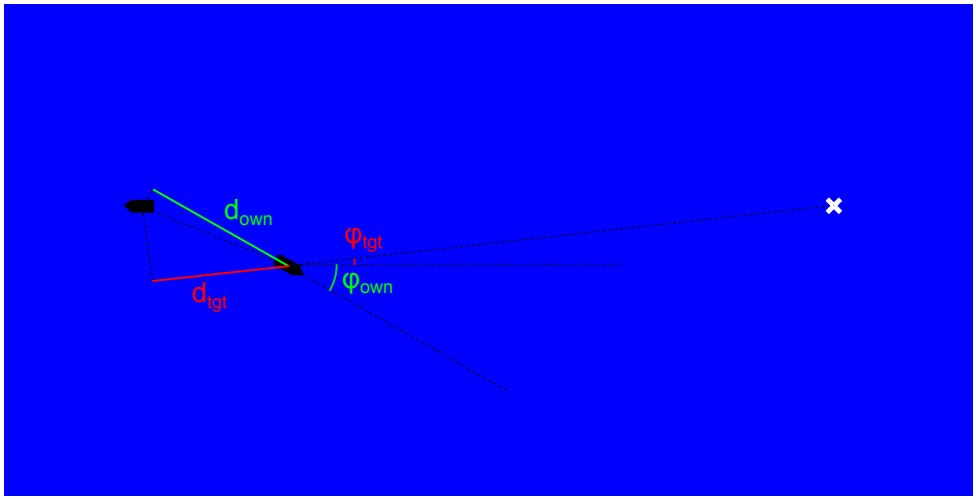


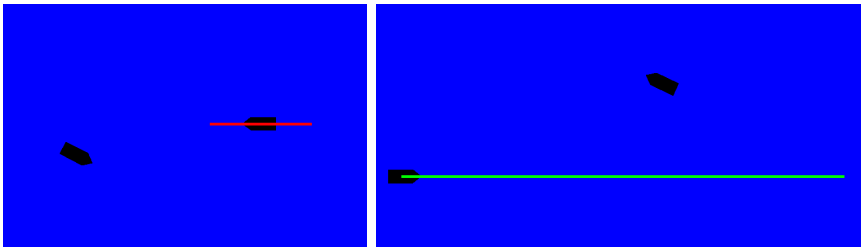
Abbildung 4.24: Berücksichtigte Abstände zur Feststellung des Endes einer Situation.

Situationsende. Das Situationsende ist erreicht, wenn das autonome Schiff das andere bezüglich zwei Richtungen um einen parametrisierten Sicherheitsabstand d_{off} hinter sich gelassen hat: aktuelle Orientierung des autonomen Schiffs (cf. d_{own} in Abb. 4.24) und aktuelle Richtung zum nächsten Ziel ((cf. d_{tgt} in Abb. 4.24)), i. e. Zielort oder nächstes *Gate*.

Situation. Bei Verarbeitung der Situation wird unterschieden, ob das autonome Schiff aufgrund der bei Situationsbeginn getroffenen Entscheidung ausweichen muss oder nicht. Muss es ausweichen, so wird das andere Schiff durch ein abstoßendes Potential abgebildet. Da die Regel keine bestimmte Ausweichrichtung vorgibt, wird lediglich ein bewegliches Linienpotential (cf. Kapitel 4.2.4.1) der Länge $l_{pot} = c \cdot l_{ship}$ über das andere Schiff gelegt, wobei c ein beliebiger Parameter ist, das sich mit den aktuellen kinematischen Daten des Schiffs bewegt.

Muss das autonome Schiff nicht ausweichen, so wird zur Erfüllung der Regel 17, i. e. Halten des Kurses, ein gerichtetes Linienpotential von der aktuellen Position des autonomen Schiffs in Richtung der aktuellen Orientierung gelegt. Als Länge der Linie wird dabei ein Parameter verwendet.

Die erzeugten Potentiale sind in Abb. 4.25 dargestellt.



(a) (b)

Abbildung 4.25: Erzeugte Potentiale für das Verhalten in Situationen der Regel 12: (a) Ausweichen aufgrund eines abstoßenden beweglichen Linienpotentials; (b) Kurshaltung durch gerichtetes Linienpotential.

Regel 13: Überholen

Regelzusammenfassung: *Ein überholendes Fahrzeug muss dem überholten immer ausweichen. Dabei gilt ein Fahrzeug als überholend, wenn es zum überholten eine Peilung von* $112,5°$ *bis* $247,5°$ *hat.*

Diese Regel greift unabhängig vom Typ der Schiffe. Die Struktur dieses Moduls ist analog zu *Regel 12: Segelfahrzeuge* und wird hier nicht erneut erläutert. Stattdessen werden die drei relevanten Schritte beschrieben. Es sei noch angemerkt, dass eine Erkennung, ob das autonome Schiff überholt wird, i. e. es soll den Kurs halten, nicht implementiert ist.

Situationsbeginn. Zunächst wird geprüft, ob das autonome Schiff schneller ist als das geprüfte, also $v_{own} \geq v_{other}$. Nur dann kann eine Situation, bei der das autonome Schiff das andere überholt, eintreten. Außerdem sollen nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. Gleichung (4.12) muss gelten. Im nächsten Schritt wird geprüft, aus welchem Winkel relativ zur Orientierung des anderen Schiffs sich das autonome nähert. Nur für einen bestimmten Winkelbereich gilt eine Situation als Überholen (cf. Abb. 2.1). Zuletzt wird analog zu Regel 12 der *CPA* berechnet und der Abstand gegen einen Schwellwert geprüft.

Situationsende. Für ein Situationsende gibt es hier mehrere Bedingungen, von denen eine zutreffen muss, um die Situation als beendet zu erklären. Eine ist die in *Regel 12: Segelfahrzeuge* vorgestellte. Eine weitere ist, wenn der Abstand zwischen den Schiffen sehr groß wird. Die dritte Bedingung ist, dass $v_{own} < v_{other}$ gilt und das autonome Schiff um einen Abstand d_{off} (cf. Regel 12) hinter dem zu überholenden Schiff liegt.

Situation. Bei Verarbeitung der Situation wird lediglich ein bewegliches Linienpotential der Länge $l_{pot} = c \cdot l_{ship}$ über das zu überholende Schiff gelegt. Dadurch wirkt das Schiff abstoßend und das autonome Schiff weicht diesem aus.

Regel 14: Entgegengesetzte Kurse

Regelzusammenfassung: *Bei Annäherung zweier Maschinenfahrzeuge mit entgegengesetzten Kursen müssen beide Schiffe ihren Kurs jeweils nach Steuerbord ändern.*

Diese Regel greift nur, wenn die beteiligten Schiffe maschinengetrieben sind. Die Struktur dieses Moduls ist analog zu *Regel 12: Segelfahrzeuge* und wird hier nicht erneut erläutert. Stattdessen werden die drei relevanten Schritte beschrieben.

Situationsbeginn. Es sollen nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. Gleichung (4.12) muss gelten. Anschließend wird geprüft, in welchem Winkel das autonome und das andere Schiff aufeinander zu fahren und die Differenz muss kleiner als ein Schwellwert $\varphi_{threshold,14}$ sein, also $norm(\varphi_{own} - norm(\varphi other + \pi)) \leq \varphi_{threshold,14}$. Zuletzt wird analog zu Regel 12 der *CPA* berechnet und der Abstand gegen einen Schwellwert geprüft.

Situationsende. Das Ende einer Situation wird analog zu *Regel 12: Segelfahrzeuge* festgestellt.

Situation. Bei Verarbeitung der Situation wird ein abstoßendes Linienpotential erzeugt, dessen einer Punkt die Position des auszuweichenden Schiffs und dessen anderer Punkt die Position des autonomen Schiffs plus eines Abstands d_{offset} zur Backbordseite — da die Regel ein Ausweichen zur Steuerbordseite erfordert (cf. Kapitel 2.1.2) — ist. Abb. 4.26 zeigt die Lage der Potentialquelle.

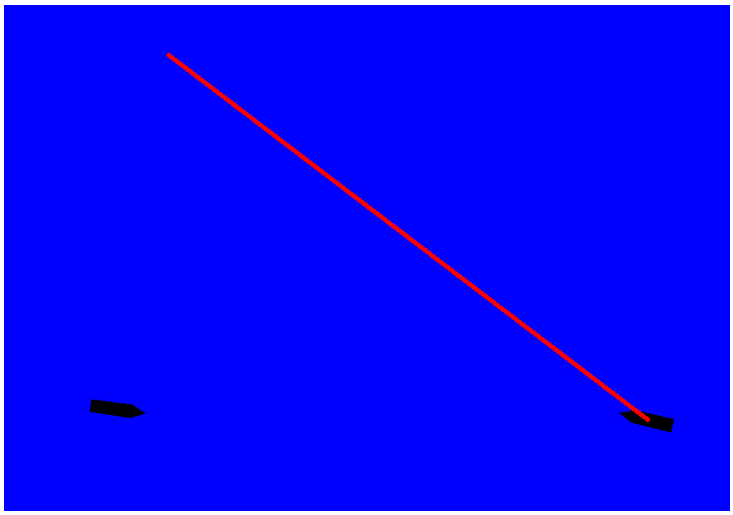


Abbildung 4.26: Exemplarische Lage des abstoßenden Linienpotentials in einer Situation der Regel 14.

Regel 15: Kreuzende Kurse

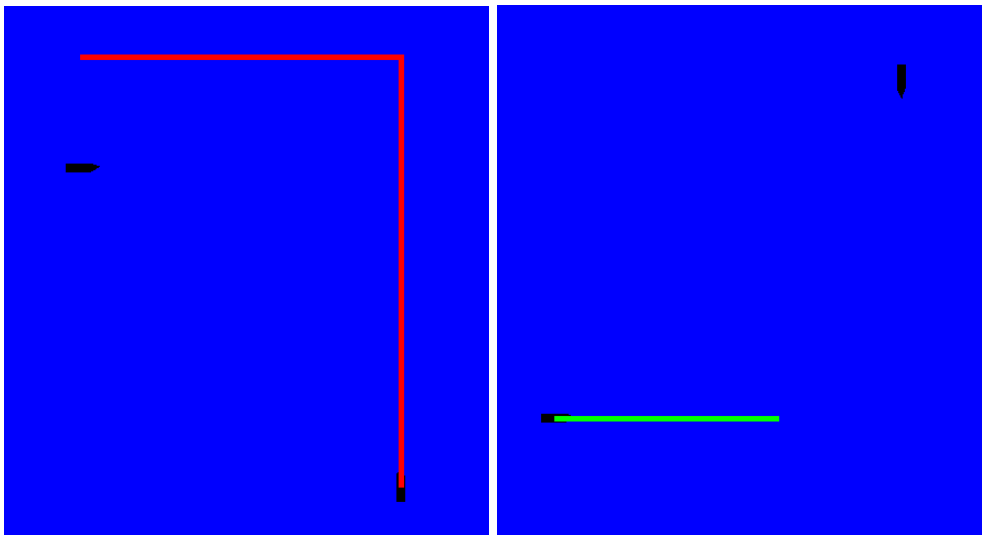
Regelzusammenfassung: *Bei Annäherung zweier Maschinenfahrzeuge mit kreuzenden Kursen muss das Schiff ausweichen, welches das andere an seiner Steuerbordseite hat.*

Diese Regel greift nur, wenn die beteiligten Schiffe maschinengetrieben sind. Die Struktur dieses Moduls ist analog zu *Regel 12: Segelfahrzeuge* und wird hier nicht erneut erläutert. Stattdessen werden die drei relevanten Schritte beschrieben.

Situationsbeginn. Es sollen nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. Gleichung (4.12) muss gelten. Eine Situation kreuzender Kurse liegt nur dann vor, wenn sich das andere Schiff relativ zur Orientierung des autonomen rechts von diesem befindet und nach links fährt oder links befindet und nach rechts fährt. Dabei wird ein Schwellwert $\varphi threshold, 15$ beachtet, um eine Situation der Regel 14 nicht versehentlich als eine Regel kreuzender Kurse zu betrachten. Wie in den vorigen Regeln wird hier auch der *CPA* berechnet und der Abstand gegen einen Schwellwert geprüft. Zuletzt wird abhängig davon, ob sich das andere Schiff rechts oder links vom autonomen befindet, die Entscheidung getroffen, ob ausgewichen werden muss oder nicht.

Situationsende. Das Ende einer Situation wird analog zu *Regel 12: Segelfahrzeuge* festgestellt.

Situation. Bei Verarbeitung der Situation wird ein anziehendes gerichtetes Linienpotential vom autonomen Schiff ausgehend in Richtung der aktuellen Orientierung gelegt, falls es nicht ausweichen muss, um der Regel 17, den Kurs zu halten, zu genügen. Muss es allerdings ausweichen, so werden zwei abstoßende Linienpotentiale erzeugt. Dazu wird zunächst ein Hilfspunkt $\vec{x}_{aux}$ berechnet, der den Schnittpunkt zwischen einer Linie in Richtung der Orientierung des anderen Schiffs durch dessen Position und einer Linie links des autonomen Schiffs in dessen Richtung darstellt. Das erste Linienpotential wird dann durch eine Linie von der Position links des eigenen Schiffs $\vec{x}_{own} + \vec{x}_{off}$ zum Hilfspunkt $\vec{x}_{aux}$ gebildet und das zweite Linienpotential durch eine Linie von der Position des anderen Schiffs $\vec{x}_{other}$ zum Hilfspunkt $\vec{x}_{aux}$. Abb. 4.27 zeigt die Lage der Potentialquellen für die jeweiligen Fälle.



(a) (b)

Abbildung 4.27: Erzeugte Potentiale für das Verhalten in Situationen der Regel 15: (a) Ausweichen aufgrund von abstoßenden Linienpotentialen; (b) Kurshaltung durch gerichtetes Linienpotential.

Regel 17: Manöver des Letzten Augenblicks

Regelzusammenfassung: *Ein Schiff mit Vorfahrt muss dem anderen ausweichen, wenn dieses nicht angemessen agiert oder durch alleinige Manöver einen Unfall nicht verhindern kann. Das Schiff mit Vorfahrt führt dann ein Manöver des letzten Augenblicks durch.*

Diese Regel setzt sich aus zwei Anteilen zusammen: Neben der in den vorigen Regeln bereits verwendeten Vorgabe, dass ein Schiff mit Vorfahrt seinen Kurs halten soll, muss auch ein Kurshalter im Notfall ausweichen, um einen Zusammenstoß zu vermeiden. Die hier erläuterte Umsetzung dieser Regel stellt eine Interpretation der Regel 17 der Kollisionsverhütungsregeln (cf. Kapitel 2.1) dar, die *Manöver des letzten Augenblicks* genannt wird. Unabhängig vom Typ der beteiligten Schiffe und unabhängig von einer durch eine andere Regel bestimmte Vorfahrtsregelung, soll das autonome Schiff einem anderen ausweichen, wenn die Gefahr einer Kollision besteht. Dazu werden die folgenden drei Schritte umgesetzt.

Situationsbeginn. Es sollen wiederum nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. Gleichung (4.12) muss gelten. Wie in den vorigen Regeln wird hier auch der *CPA*

Algorithmus 7 Verwendeter Algorithmus bei der Verarbeitung einer Situation von *Regel 17: Manöver des Letzten Augenblicks*. Alle Potentiale sind Linienpotentiale. Die Funktion $movesAwayFromOwnEntity$ bestimmt, ob sich das andere Schiff vom autonomen wegbewegt, i. e. relativ zum autonomen Schiff ist es links und fährt nach links oder ist rechts und fährt nach rechts. Die Funktion $intersection$ ermittelt anhand zweier Linien, die durch Position und Richtung gegeben sind, deren Schnittpunkt. d_{off} ist ein parametrisierter Abstand.

function $processSituation(e_{own}, e_{other})$
  $i \leftarrow 0$;
  if $isPowerDriven(e_{own})$ **and** $isPowerDriven(e_{other})$ **and** $isCrossingSituation(e_{own}, e_{other})$ **then**
    if $moveInSameDirection(e_{own}, e_{other})$ **then**
      // Potential in Richtung der Orientierung des anderen Schiffs (Fall 1)
    else
      // Potential orthogonal zur Orientierung des anderen Schiffs (Fall 2)
    end if
  else
    $\Delta\varphi = norm(e_{other}.\varphi - e_{own}.\varphi)$;
    if $movesAwayFromOwnEntity(e_{own}, e_{other})$ **then**
      // Bewegliches Potential auf anderem Schiff (Fall 3)
    else if $\Delta\varphi < \varphi_{threshold}$ **then**
      // Potential vom anderen Schiff in Richtung dessen Orientierung (Fall 4)
    else
      $\vec{x}_{off} = (0, d_{off})$;
      $\vec{x}_{off}.rotate(-e_{own}.\varphi)$;
      $\vec{x}_{aux} = intersection(e_{own}.pos + \vec{x}_{off}, e_{own}.\varphi, e_{own}.pos, e_{own}.\varphi)$;
      if $isInFrontOfOwn(\vec{x}_{aux}, e_{own})$ **then**
        // Potentiale zwischen autonomem und anderem Schiff über Hilfspunkt (Fall 5)
      else
        // Potential von Seite des autonomen Schiffs zur Position des anderen (Fall 6)
      end if
    end if
  end if
end function

berechnet und der Abstand gegen einen Schwellwert geprüft, wobei hier der Schwellwert geringer und abhängig von der Größe der beteiligten Schiffe berechnet wird:

$$d_{cpa} \leq max(l_{own}, w_{own}) + max(l_{other}, w_{other}) + d_{safety} \tag{4.17}$$

mit d_{safety} als spezieller Parameter, um einen Sicherheitsabstand vorzugeben. Zuletzt wird noch der minimale Drehkreis in Abhängigkeit von der aktuellen Geschwindigkeit des autonomen Schiffs

$$r = \frac{v}{\omega_{max}} \tag{4.18}$$

ermittelt und mit dem Abstand zur *CPA* Position verglichen, i. e. es muss gelten

$$|\vec{x}_{CPA,own} - \vec{x}_{own}| \leq r. \tag{4.19}$$

Sind diese Bedingungen erfüllt, beginnt eine Situation.

Situationsende. Das Ende einer Situation wird analog zu *Regel 12: Segelfahrzeuge* festgestellt.

Situation. Die Verarbeitung der Situation erfolgt nach dem in Alg. 7 präsentierten Algorithmus. Die Lage der Linienpotentiale ist in Abb. 4.28 visualisiert.

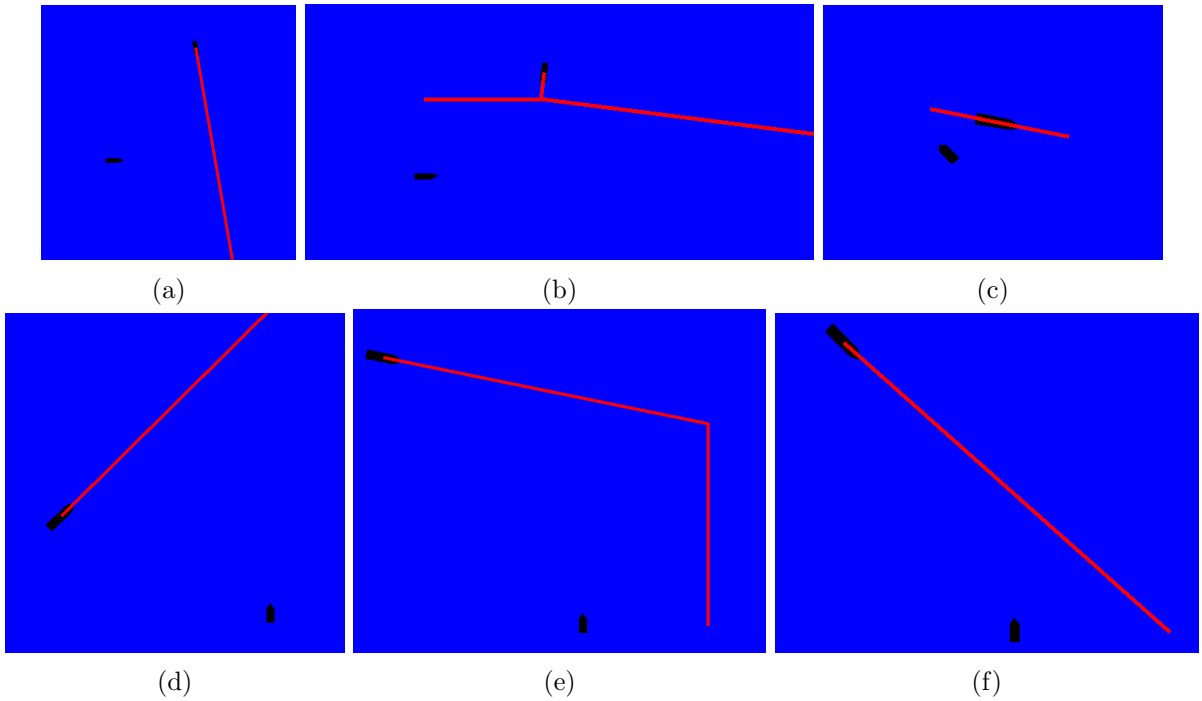


Abbildung 4.28: Erzeugte Potentiale für das Verhalten in Situationen der Regel 17: (a) Situation kreuzender Kurse zweier maschinengetriebener Schiffe bei Kursen in gleicher Richtung (Fall 1); (b) Situation kreuzender Kurse zweier maschinengetriebener Schiffe bei entgegengesetzten Kursen (Fall 2); (c) anderes Schiff bewegt sich vom autonomen weg (Fall 3); (d) Ausweichen in Fahrtrichtung des anderen Schiffs (Fall 4); (e) Ausweichen durch Linienpotential mit Hilfspunkt (Fall 5); (f) Ausweichen durch Linienpotential ohne Hilfspunkt (Fall 6).

Regel 18: Verantwortlichkeiten der Fahrzeuge untereinander

Regelzusammenfassung: *Abhängig vom Typ zweier beteiligter Schiffe, falls sich diese unterscheiden, sind bestimmte Ausweichpflichten definiert. Manövrierbehinderte oder -unfähige Schiffe haben immer Vorfahrt, maschinengetriebene Schiffe müssen immer ausweichen, Segelschiffe müssen fischenden Schiffen ausweichen.*

Diese Regel greift unabhängig vom Typ der Schiffe. Die Struktur dieses Moduls ist analog zu *Regel 12: Segelfahrzeuge* und wird hier nicht erneut erläutert. Stattdessen werden die drei relevanten Schritte beschrieben.

Situationsbeginn. Es sollen nur die Schiffe betrachtet werden, die nicht zu weit weg sind, i. e. Gleichung (4.12) muss gelten. Wie in den vorigen Regeln wird hier auch der *CPA* berechnet und der Abstand gegen einen Schwellwert geprüft. Zuletzt wird abhängig von den Typen der beteiligten Schiffe (cf. Tab. 2.1) die Entscheidung getroffen, ob ausgewichen werden muss oder nicht.

Situationsende. Das Ende einer Situation wird analog zu *Regel 12: Segelfahrzeuge* festgestellt.

Situation. Die Verarbeitung der Situation verhält sich analog zu *Regel 12: Segelfahrzeuge*, i. e. es wird unterschieden, ob das autonome Schiff aufgrund der bei Situationsbeginn getroffenen Entscheidung ausweichen muss oder nicht. Muss es ausweichen, so wird das andere Schiff durch ein abstoßendes Potential abgebildet. Da die Regel keine bestimmte Ausweichrichtung vorgibt, wird lediglich ein bewegliches Linienpotential (cf. Kapitel 4.2.4.1) der Länge $l_{pot} = c \cdot l_{ship}$ über das andere Schiff gelegt, wobei c ein beliebiger Parameter ist, das sich mit den aktuellen kinematischen Daten des Schiffs bewegt.

Muss das autonome Schiff nicht ausweichen, so wird zur Erfüllung der Regel 17, i. e. Halten des Kurses, ein gerichtetes Linienpotential von der aktuellen Position des autonomen Schiffs in Richtung der aktuellen Orientierung gelegt. Als Länge der Linie wird dabei ein Parameter verwendet.

Die erzeugten Potentiale liegen entsprechend Abb. 4.25.

4.2.4.4 Kombinationsmodul

Das Kombinationsmodul dient entsprechend dem Konzept (cf. Kapitel 4.1.1.5) dazu, unter Berücksichtigung der von den verschiedenen Modulen gelieferten Verhaltensdaten Steuerungskommandos zu ermitteln. Um dies zu erreichen, wird der im genannten Kapitel zum Konzept beschriebene Ansatz *Kombinationsansatz Potentialfeld* verwendet. Der implementierte Algorithmus zum Vorausplanen des abzufahrenden Weges ermittelt zunächst aus den Verhaltensdaten das Geschwindigkeitslimit L, indem als untere Schranke der höchste Wert für die untere Schranke in der Menge der Verhaltensdaten gewählt wird und entsprechend als obere Schranke der kleinste Wert aus der Menge oberer Schranken. Anschließend wird der Wurzelknoten N_{root} anhand der Daten des autonomen Schiffs ermittelt. Der nächste und relevante Schritt ruft den rekursiven Algorithmus *expand* zum Expandieren des Knotens auf (cf. Alg. 8). Zusätzlich ist der Block innerhalb des **If (not** $expandValid$**)** implementiert zur Realisierung des *Backtracking*-Algorithmus. Hier wird der alternative Folgezustand berechnet und auf Gültigkeit geprüft.

Dabei sind die im Algorithmus verwendeten Funktionen unten aufgeführt:

- Die Funktion $isNearTargetPosition$ ermittelt, ob der untersuchte Knoten nahe dem Zielort liegt. Das ist der Fall, wenn der Abstand zum Ziel kleiner als ein Schwellwert $d_{nearTgt}$ und die Geschwindigkeit annähernd 0 ist.

Algorithmus 8 Rekursives Expandieren von Knoten als Teil des Algorithmus zum Planen des zu fahrenden Weges durch das Kombinationsmodul. T bezeichnet die Struktur zur Verwaltung des expandierten Pfades, N den zu expandierenden Knoten, R die Route, $\vec{x}_{tgt}$ den Zielort, e_{own} die Daten des autonomen Schiffs, D die Verhaltensdaten, i. e. Potentiale und Geschwindigkeitsbegrenzung, und L die bereits ermittelte Begrenzung der Geschwindigkeit.

function $expand(T, N, R, \vec{x}_{tgt}, e_{own}, D, L)$
  $t \leftarrow N.t + \Delta t$;
  if $t > t_{max}$ **or** $isNearTargetPosition(N, \vec{x}_{tgt})$ **then**
    $evaluate(N)$;
    if $isValid(N)$ **then**
      $T.bestNode \leftarrow N$;
      return $TRUE$;
    end if
  end if
  $expandValid \leftarrow FALSE$;
  if $targetAhead(N, \vec{x}_{tgt})$ **then**
    $v_{new} \leftarrow L.upper$;
  else if $targetAtSide(N, \vec{x}_{tgt}, v_{new}, t)$ **then**
    $v_{new} \leftarrow N.v$;
  else
    $v_{new} \leftarrow L.lower$;
  end if
  $v_{new} \leftarrow adjustSpeedToApproachingTarget()$;
  $b \leftarrow isApproachingTarget(N, \vec{x}_{tgt}, e_{own}.a_{min})$;
  $\varphi_{new} \leftarrow angleToTarget(N, \vec{x}_{tgt})$;
  $N_{new} \leftarrow calculateNodeAt(N, \varphi_{new}, v_{new})$;
  if $isValid(N_{new})$ **then**
    $F \leftarrow calculatePotential(N_{new}.pos, t, D)$;
    if $improveByPotential(N_{new}, F, D, b, \Delta\varphi, \Delta v)$ **then**
      $expandValid \leftarrow expand(T, N_{new}, R, \vec{x}_{tgt}, D, L)$;
    end if
    if not $expandValid$ **then**
      // Berechne alternativen Folgezustand als Teil des Backtracking
      $N_{new} \leftarrow calculateNodeAt(N, \varphi_{new} - \Delta\varphi, v_{new} - \Delta v)$;
      if $isValid(N_{new})$ **then**
        $F \leftarrow calculatePotential(N_{new}.pos, t, D)$;
        if $F \leq F_{threshold}$ **then**
          $expandValid \leftarrow expand(T, N_{new}, R, \vec{x}_{tgt}, D, L)$;
        end if
      end if
    end if
  end if
  return $expandValid$;
end function

- Die Funktion *adjustSpeedToApproachingTarget* berechnet in Abhängigkeit vom Abstand zum Ziel d_{tgt} und der maximalen Bremsbeschleunigung a_{min} eine Geschwindigkeit, mit der das Ziel nicht überfahren wird.
- Die Funktion *isApproachingTarget* prüft, ob das Schiff in der Annäherungsphase zum Zielort ist. Dabei müssen der Abstand zum Zielort und der Bremsweg berücksichtigt werden.
- Die Funktion *angleToTarget* berechnet die Richtung zum Ziel. Dabei wird unterschieden, ob das Ziel der Zielort oder ein *Gate* ist. Im ersten Fall wird die Richtung zum Zielort selbst verwendet, im zweiten wird der Sektor zu den Randpunkten des *Gate* betrachtet und zusätzlich die Richtung zum Ziel nach diesem *Gate*.
- Die Funktion *calculateNodeAt* berechnet ausgehend vom aktuellen Knoten, einer Zielgeschwindigkeit und -orientierung unter Verwendung von maximaler Beschleunigung bzw. Bremsbeschleunigung und Drehrate einen neuen Knoten.
- Die Funktion *isValid* ermittelt, ob der geprüfte Knoten innerhalb der Route R liegt und damit gültig ist.
- Die Funktion *calculatePotential* berechnet unter Verwendung der Verhaltensdaten die auf den Punkt zum Zeitpunkt t wirkende Kraft.
- Die Funktion *improveByPotential* verändert den gegebenen Knoten aufgrund der gegebenen Kraft entsprechend Abb. 4.12. Dabei wird die neue Position unter Berücksichtigung der kinematischen Beschränkungen Beschleunigung und Drehrate ermittelt und zusätzlich die Geschwindigkeit und Orientierung angepasst, die aus der Positionsänderung folgen. Die Geschwindigkeit wird nur durch Erhöhung angepasst, wenn das Schiff nicht in der Annäherungsphase zum Zielort ist (cf. Funktion *isApproachingTarget*), um diesen nicht zu überfahren.

  Diese Funktion arbeitet iterativ, i. e. der gegebene Knoten wird verändert, anschließend wird das auf den veränderten Knoten wirkende Potential ermittelt und eventuell führt dies zu einer erneuten Anpassung des Knotens. Als Abbruchbedingungen werden hier eine maximale Anzahl Schritte n_{max} und ein Schwellwert für die Stärke der wirkenden Kraft $|F_{ignore}|$ verwendet. Ist also der Iterationsschritt $i \geq n_{max}$ oder die Stärke der wirkenden Kraft $|F| \leq |F_{ignore}|$, wird der Knoten nicht weiter verbessert. Zusätzlich wird der Verbesserungsschritt abgebrochen, wenn die wirkende Kraft in eine Richtung weist, in die der Knoten aufgrund bereits ausgereizter Drehrate nicht weiter verbessert werden kann.

  Ist die Summe der Stärken aller wirkenden Potentiale größer als ein Schwellwert, gibt die Funktion an den Aufrufer zurück, dass die Verbesserung fehlgeschlagen ist.

Trotz der Vorausplanung des zu fahrenden Weges wird der oben erläuterte Algorithmus aufgrund bestimmter Bedingungen angestoßen, eine Neuplanung durchzuführen. Dazu zählt eine Änderung der Parameter des Moduls, die die Planung betreffen, i. e. das verwendete Δt und t_{max}. Außerdem führen eine Änderung der Route, die nicht nur im Voranschreiten beim Folgen der Route besteht, sowie eine Änderung der Position des autonomen Schiffs um einen Abstand $d_{threshold}$ gegenüber dem letzten Aufruf des Kontrollzyklus, i. e. manuelles Verschieben des Schiffs, zu einer Neuberechnung. Zusätzlich müssen auch Änderungen in den Verhaltensdaten, also eine Änderung bei den gegebenenen Potentialen, zu einer Neuplanung führen. Aufgrund der hohen Komplexität dieses Problems und des begrenzten Rahmens dieser Arbeit ist ein entsprechender Algorithmus allerdings nicht implementiert. Stattdessen wird in bestimmten Zeitabständen eine Neuplanung ausgelöst.

Abb. 4.29 zeigt exemplarisch einen vom Kombinationsmodul vorausgeplanten Weg. Die gelben Linien stellen Verbindungskanten zwischen Knoten dar. Tatsächlich bewegt sich das Schiff zwischen diesen Knoten allerdings auf einem Kreisbogen.

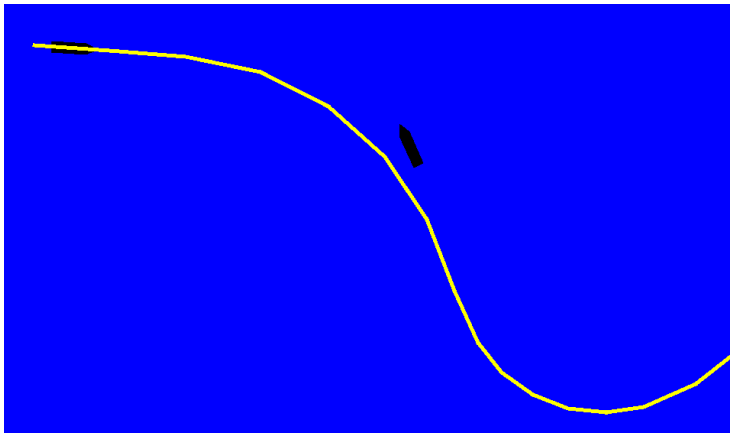


Abbildung 4.29: Exemplarische Lage des vorausgeplanten Weges durch das Kombinationsmodul.

4.3 Verwendete Vereinfachungen

Beschränkung auf zwei Dimensionen. Die Vereinfachung, innerhalb des autonomen Verhaltens lediglich im Zweidimensionalen zu arbeiten anstatt im Dreidimensionalen, liegt aufgrund der Beschränkung auf die Domäne Schiff nahe. Für eine allgemeinere autonome Steuerung, die auch dreidimensional bewegliche Objekte (e. g. Unterwasserboote oder Flugzeuge) verarbeiten kann, muss diese Beschränkung aufgehoben werden.

Kein autonomes Verhalten bei ungültiger Route. Ist keine gültige Route vorhanden, gibt das lokale Verhalten derzeit keine Steuerungskommandos bzw. lässt das Schiff mit aktueller Ausrichtung und ohne Geschwindigkeit stehen. Dadurch wäre es also auch möglich, dass ein Schiff innerhalb eines Verkehrstrennungsgebiets anhält, i. e. ankert. Die Kollisionsverhütungsregeln jedoch definieren bestimmte Bereiche, in denen nicht geankert werden darf oder in denen fahrenden Schiffen ausgewichen werden muss. Dies ist hier nicht abgedeckt.

Eine mögliche Lösung wäre, das aktuelle konvexe Polygon des Navigation Mesh als globale Route aktiv zu halten, sodass der lokale Planer innerhalb dieses Polygons ausweichen kann.

Umwelt wird als bekannt angenommen. In realen Anwendungen (e. g. Robotik) wird die Umwelt mithilfe von Sensoren, die mehr oder weniger ungenau sind, wahrgenommen und mithilfe von Modellen in eine stabilere Repräsentation übernommen. Häufig wird in den Modellen neben dem angenommenen Zustand auch die Ungenauigkeit dieses Zustands ermittelt. Hier wird die Umwelt allerdings als bekannt angenommen und auch in gewissem Maße — nämlich im Rahmen der Optimierung von Parametern — davon ausgegangen.

Keine Berücksichtigung des Tiefgangs. Im Rahmen dieser Arbeit wird der Tiefgang von Schiffen und die damit verbundene Modellierung verschiedener Tiefen im Terrain der Umgebung nicht berücksichtigt. Der Grund liegt in der statischen und schiffsunabhängigen Definition des *Navigation Mesh*. Um dies zu realisieren ist eine automatische Generierung des *Navigation Mesh* in Abhängigkeit von Tiefgang und Wassertiefen notwendig. Wie in Kapitel 4.1.1.4 angemerkt, kann dieser Schritt allerdings separat zum Zeitpunkt

der Initialisierung durchgeführt und hier vernachlässigt werden. Eine Erweiterbarkeit des autonomen Verhaltens zur Berücksichtigung der Tiefgänge kann also leicht durch diesen zusätzlichen Schritt erreicht werden.

Optimale Modellierung des *Navigation Mesh*. Im Rahmen der *Regel 9: Enge Fahrwasser* wird die Annahme getroffen, dass das *Navigation Mesh* möglichst optimal modelliert ist. Das bedeutet, dass große freie Flächen von wenigen großen Polygonen repräsentiert werden. Diese Annahme muss getroffen werden, da die Regel ausschließlich auf der Route operiert und abhängig von der Breite der Polygone ermittelt, ob ein Bereich als enges Fahrwasser gilt.

4.4 Ausstehende Probleme

Kein Kommunikationsrückweg vom lokalen Verhalten zum globalen. Es kann passieren, dass eine vom globalen Planer gegebene Route vom lokalen Verhalten abgelehnt werden soll, da sie möglicherweise durch dynamische Hindernisse, aufgrund zu enger Passagen oder gegebener Nebenbedingungen (e. g. spezifizierte Regeln) nicht verfolgt werden kann. In einem solchen Fall muss der lokale Planer eine Möglichkeit haben, dem globalen Planer rückmelden zu können, dass eine neue Route gesucht werden muss. Eine alternative Lösungsvariante wäre, dass der globale Planer dem lokalen Verhalten von vornherein nicht nur eine Route, sondern mehrere mögliche Routen zur Verfügung stellt.

Abdrängen des autonom fahrenden Schiffs. Es kann passieren, dass aufgrund einer geltenden Regel und der Annäherung eines Schiffs aus einem bestimmten Winkelbereich das autonom fahrende Schiff in Fahrtrichtung des anderen Schiff abdrehen muss. Es kann dann durch das verwendete Verfahren zu einer Situation kommen, bei der beide Schiffe mit annähernd gleicher Geschwindigkeit nebeneinander herfahren, ohne dass das autonome Schiff deutlich dem Ziel näher kommt. Allerdings läuft es damit auch nicht in eine Falle des Potentialfeldes, da es sich noch in einem Polygon der Route befindet.

Für diesen Fall müsste eine Erkennung entwickelt werden, ob durch Nutzung anderer Steuerungskommandos (e. g. Abbremsen und anschließend hinter dem anderen Schiff eindrehen) eine effektivere Reaktion auf die Situation möglich ist.

Strikte Grenzen durch Polygone des *Navigation Mesh*. Aufgrund der Nutzung eines *Navigation Mesh* haben die Polygone feste Grenzen, was dazu führt, dass auch die vom globalen Pfadplaner ermittelte Route feste Grenzen hat. Steht nun ein Schiff am Rande eines Polygons, sind seine Bewegungsmöglichkeiten stark eingeschränkt, obwohl möglicherweise auf der anderen Seite dieses Randes ein weiteres befahrbares Polygon liegt.

Eine Möglichkeit, mit diesem Problem umzugehen, besteht darin, die Definition der Route zu erweitern, sodass auch benachbarte Polygone in dieser enthalten sind. Eine weitere Möglichkeit könnte durch Definition von Übergangsbereichen zwischen Polygonen des *Navigation Mesh* auf einer weiteren Ebene gefunden werden.

Notwendigkeit zusätzlicher Testfälle. Die hohe Komplexität der autonomen Steuerung, insbesondere bei Berücksichtigung mehrerer verschiedener Regeln, erfordert eine sehr hohe Menge an Testfällen. Dies kann im Rahmen der Arbeit nicht gänzlich erfüllt werden und muss für eine größere Stabilität entsprechend erweitert durchgeführt werden. Eventuelle Anpassungen an Modulen oder verschiedenen Parametern zur Optimierung sind nicht ausgeschlossen.

Ungünstig liegende Potentiale. Liegen Potentialquellen ungünstig, kann es zu einem nicht beabsichtigten Verhalten des autonomen Schiffs führen. Sei zum Beispiel die *Regel 12: Segelfahrzeuge* angenommen. Fahren das autonome und ein anderes Schiff direkt aufeinander zu und das autonome Schiff soll ausweichen, dann wird eine abstoßende Potentialquelle auf das andere Schiff gelegt. Die davon ausgehende Kraft wirkt dann genau entgegen der Fahrtrichtung auf das gesteuerte Schiff, was dazu führt, dass dieses nur langsam den Kurs ändert und gleichzeitig stark abbremst.

Dieses Problem kann behoben werden, indem die Regelmodule die Situation näher betrachten und zum Beispiel ein zusätzliches, schwach abstoßend wirkendes Potential neben das autonome Schiff legen. Dadurch wirkt eine Kraft seitlich auf das gesteuerte Schiff, was letztlich zu einer Kursänderung führt.

Kapitel 5

Evaluation

In diesem Kapitel soll das in Kapitel 4 entwickelte Verfahren auf seine Einsatzfähigkeit evaluiert werden. Dazu werden verschiedene Situationen geschaffen. Zunächst sind diese auf spezielle Regeln bzw. spezielle Funktionen ausgerichtet, anschließend werden noch allgemeinere untersucht.

Die Evaluierung findet statt, indem der Aufbau der Situation beschrieben wird. Anschließend wird eine Person mit Kenntnis der Kollisionsverhütungsregeln befragt, auf Basis der gegebenen Situation vorherzusagen, wie das Schiff fahren soll. Diese Vorhersage wird als *erwarteter Pfad* bezeichnet und in der Karte skizziert. Dann wird das Szenario über den Simulator gestartet. Dieser erzeugt in Abständen von fünf Sekunden sogenannte *History Points*, i. e. Punkte an denen sich das Schiff zum entsprechenden Zeitpunkt befunden hat. Aus diesen *History Points* lässt sich der tatsächlich gefahrene Weg des Schiffs nachvollziehen und mit dem erwarteten Pfad vergleichen.

5.1 Testfälle

Im Folgenden werden die verschiedenen Testfälle durchgeführt. Zuerst wird der Aufbau der Situation beschrieben und dann der erwartete Pfad und der gefahrene Weg visuell dargestellt. Zuletzt wird eine kurze Auswertung gegeben. Die für den Aufbau definierten Szenarien und gegebenen Werte orientieren sich an den in der mitgelieferten Software enthaltenen vorbereiteten Szenarien sowie an den Eingabemöglichkeiten des Simulators zur Reproduzierbarkeit der Testfälle — dies führt zu einer weniger leserlichen Angabe der Winkel in Radiant.

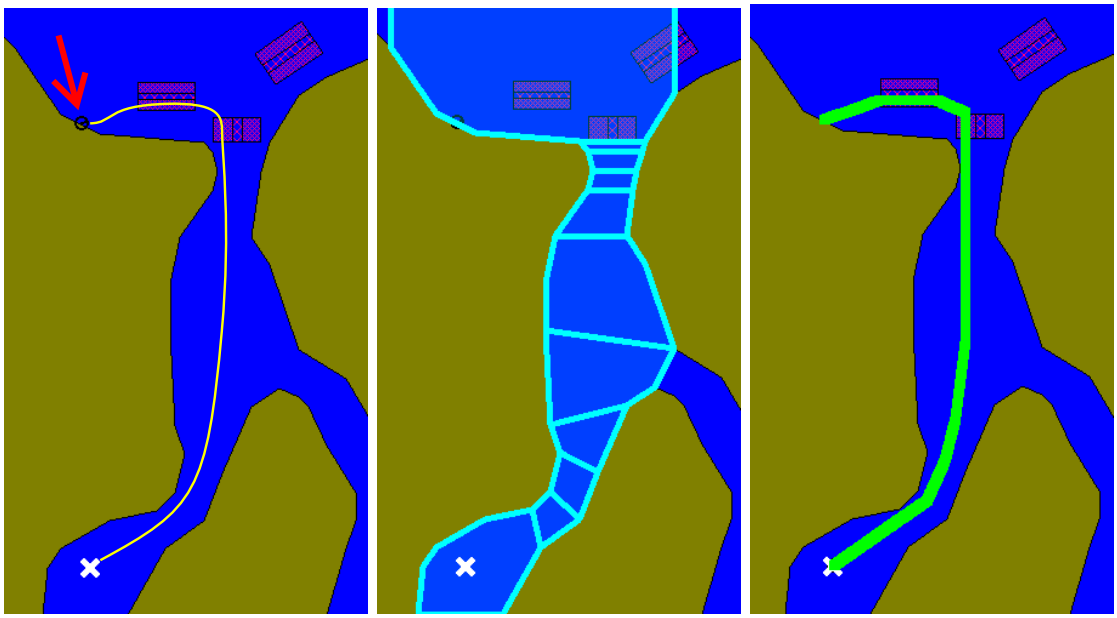
5.1.1 Globaler Planer

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation GlobalPlaner*

Das autonome Schiff ist bereits in dem geladenen Szenario vorhanden. Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = -2000$, $y = -45000$
- *Speed Factor* auf 8 einstellen



(a) (b) (c)

Abbildung 5.1: Evaluation globaler Planer: (a) erwarteter Pfad (roter Pfeil weist auf die Startposition des autonomen Schiffs); (b) ermittelte Route; (c) gefahrener Pfad.

Der erwartete und gefahrene Pfad, sowie die vom globalen Planer ermittelte Route sind in Abb. 5.1 dargestellt. Der globale Planer ermittelt eine gültige Route, die vom autonomen Schiff abgefahren wird. Zusätzlich werden die Verkehrstrennungsgebiete berücksichtigt und durchfahren.

5.1.2 Zielortannäherung

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = 0$, $y = -300$, $v = 0$, $\varphi = -1.57$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

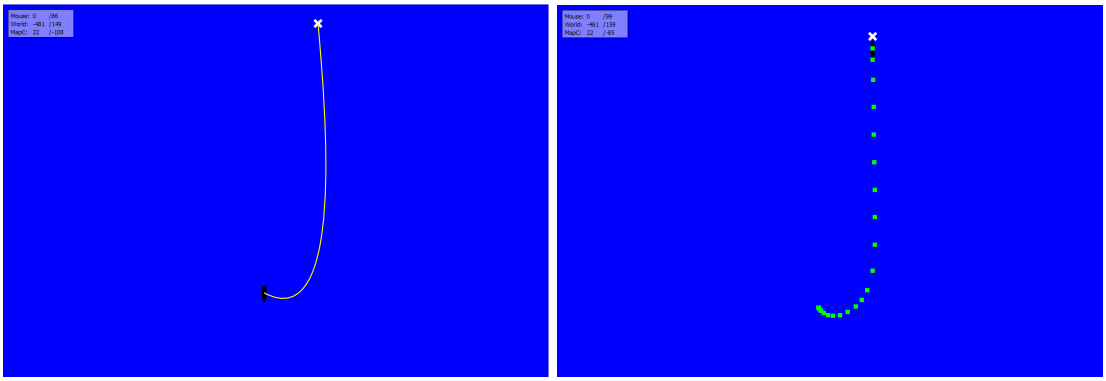
- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 100$, $y = 200$

Der erwartete und gefahrene Pfad sind in Abb. 5.2 dargestellt. Es ist zu erkennen, dass die Pfade sich stark ähneln, das Schiff verhält sich also wie erwartet.

5.1.3 Regel 6: Sichere Geschwindigkeit

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -300$, $y = 0$, $v = 0$, $\varphi = 0$
- *Sight Ratio* auf 50% einstellen



(a) (b)

Abbildung 5.2: Evaluation Zielortannäherung: (a) erwarteter Pfad; (b) gefahrener Pfad.

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 300$, $y = 0$

Der Simulator zeigt, dass das Schiff nicht mit maximaler Geschwindigkeit $v_{max} = 10\frac{m}{s}$ fährt, sondern lediglich mit einer angepassten Geschwindigkeit von $v \approx 7.5\frac{m}{s}$. Nach etwa der halben Strecke wird die *Sight Ratio* auf 25% reduziert. Nach kurzer Verzögerung bremst das Schiff auf $v \approx 4.375\frac{m}{s}$ ab. Abb. 5.3 zeigt dies anhand des größeren Abstands zwischen zwei *History Points* in der ersten Hälfte der Strecke als in der zweiten. Das Schiff berücksichtigt also diese Regel.

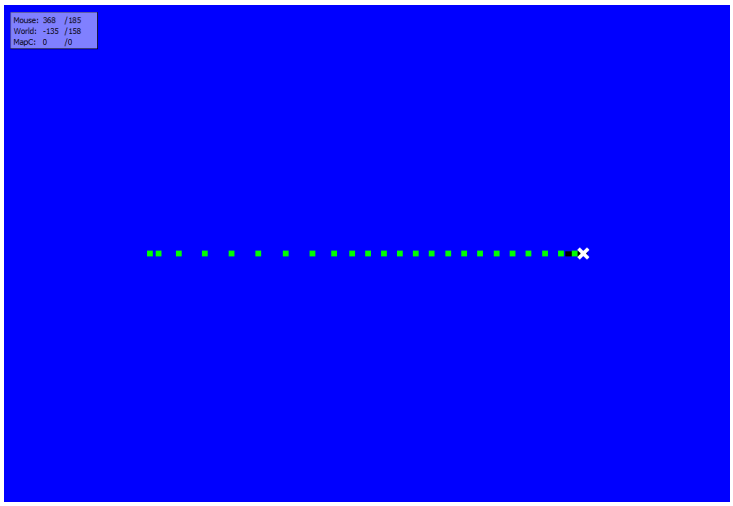


Abbildung 5.3: Evaluation Regel 6 bei reduzierten Sichtverhältnissen.

5.1.4 Regel 9: Enge Fahrwasser

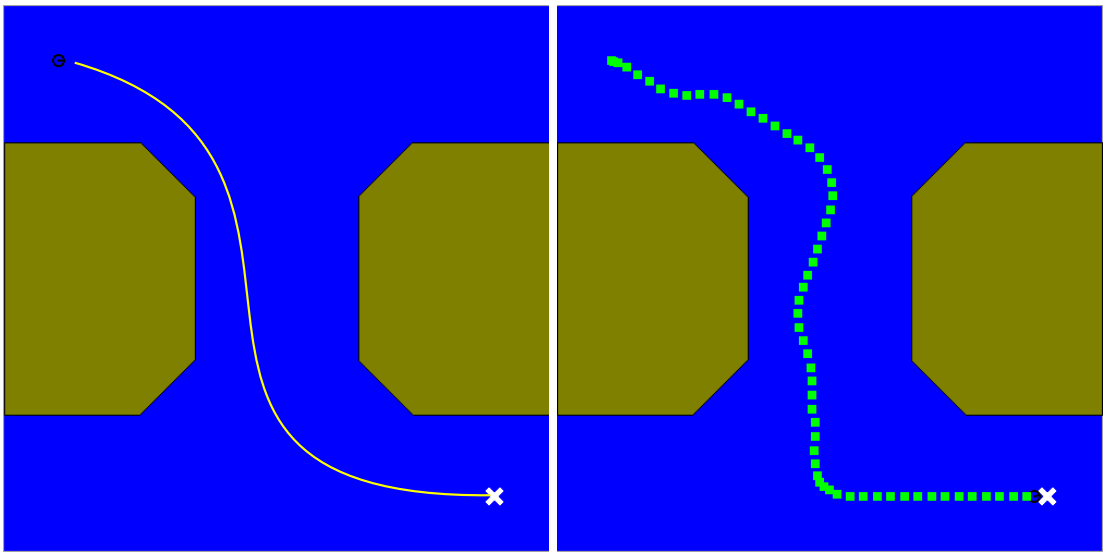
Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation Rule9*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -800$, $y = 800$, $v = 0$, $\varphi = 0$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 800$, $y = -800$

Der erwartete und gefahrene Pfad sind in Abb. 5.4 dargestellt. Obwohl das Schiff bei der Einfahrt gefährlich nahe in den Gegenverkehr steuern würde, lenkt es dennoch schnell auf die relativ zu seiner Orientierung rechte Seite des engen Fahrwassers. Auch bei der Ausfahrt ist durch die enge Kurve zu erkennen, dass es diese Regel korrekt berücksichtigt.



(a) (b)

Abbildung 5.4: Evaluation Regel 9: (a) erwarteter Pfad; (b) gefahrener Pfad.

5.1.5 Regel 10: Verkehrstrennungsgebiete (Durchfahrt)

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation Rule10*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -600$, $y = -500$, $v = 0$, $\varphi = 0$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

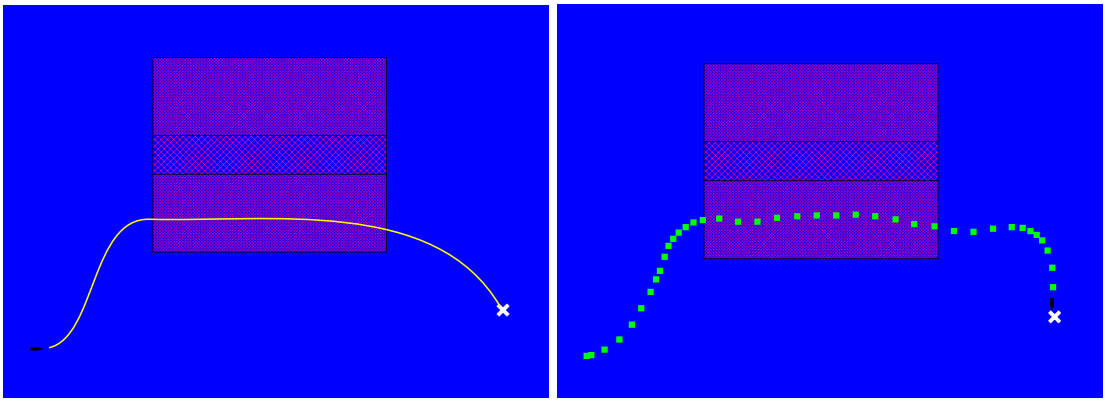
- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 600$, $y = -400$

Der erwartete und gefahrene Pfad sind in Abb. 5.5 dargestellt. Das Schiff berücksichtigt trotz des längeren Weges das Verkehrstrennungsgebiet und passiert dieses auf der korrekten Seite, ohne in die Trennzone hineinzufahren. Diese Regel wird also berücksichtigt.

5.1.6 Regel 10: Verkehrstrennungsgebiete (Kreuzen)

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation Rule10*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = 100$, $y = -500$, $v = 0$, $\varphi = 1.57$



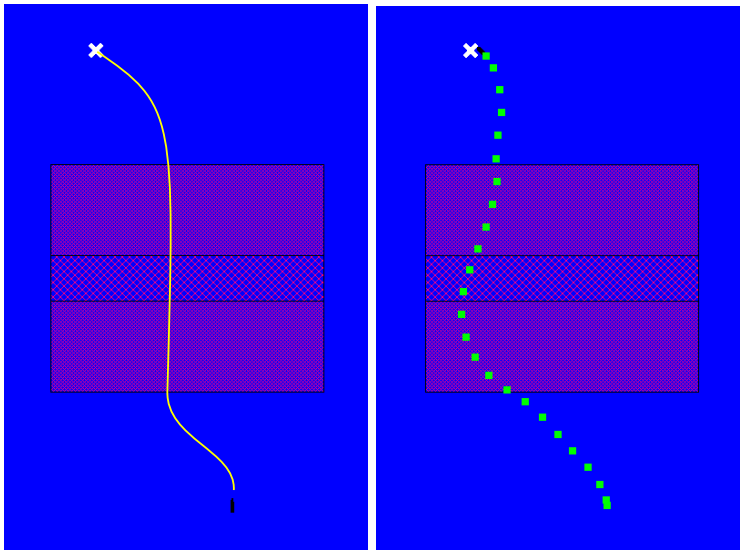
(a) (b)

Abbildung 5.5: Evaluation Regel 10 (Durchfahrt): (a) erwarteter Pfad; (b) gefahrener Pfad.

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = -200$, $y = 500$

Der erwartete und gefahrene Pfad sind in Abb. 5.6 dargestellt. Das Schiff kreuzt das Verkehrstrennungsgebiet, allerdings nicht genau rechtwinklig. Es lenkt zu spät ein und fährt damit innerhalb des Gebiets entgegen der vorgesehenen Fahrtrichtung. Der Eintrittspunkt und Austrittspunkt liegen tatsächlich so, dass sie eine Linie orthogonal zur Fahrtrichtung bilden. Das führt jedoch dazu, dass durch das Überschwingen nach dem Eintritt vor dem Austritt gegengelenkt wird und das Schiff wiederum entgegen der Fahrtrichtung fährt. Die Regel wird beim Kreuzen nicht korrekt erfüllt.



(a) (b)

Abbildung 5.6: Evaluation Regel 10 (Kreuzen): (a) erwarteter Pfad; (b) gefahrener Pfad.

5.1.7 Regel 12: Segelfahrzeuge

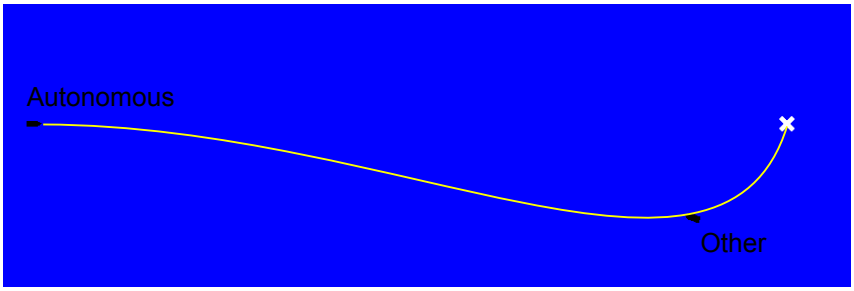
Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Sailing Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Sailing Vessel*, $x = 300$, $y = -100$, $v = 0$, $\varphi = 2.8$
- *Wind Direction* auf $-1.57rad$ einstellen
- *Wind Velocity* auf $2\frac{m}{s}$ einstellen

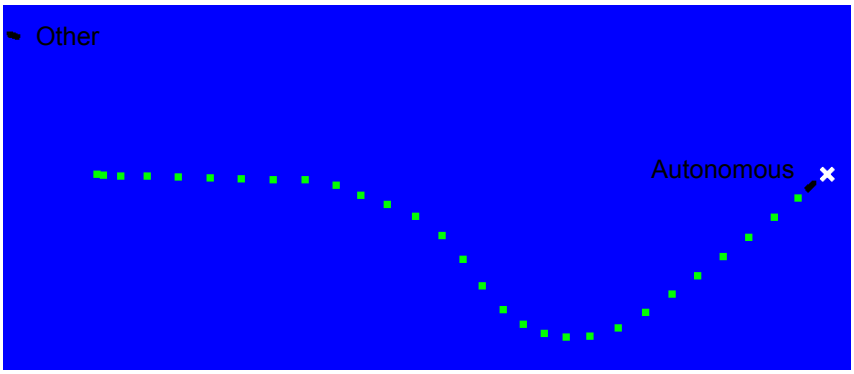
Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = 2.8$, $v = 6$

Der erwartete und gefahrene Pfad sind in Abb. 5.7 dargestellt. Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Allerdings beginnt es das Ausweichmanöver erst spät und lenkt auch erst spät zurück in Richtung des Ziels. Die Regel wird dennoch korrekt erfüllt, da der Wind von Backbord kommt und damit das autonome Schiff ausweichen muss.



(a)



(b)

Abbildung 5.7: Evaluation Regel 12: (a) erwarteter Pfad; (b) gefahrener Pfad.

5.1.8 Regel 13: Überholen

Das Szenario wird wie folgt aufgebaut:

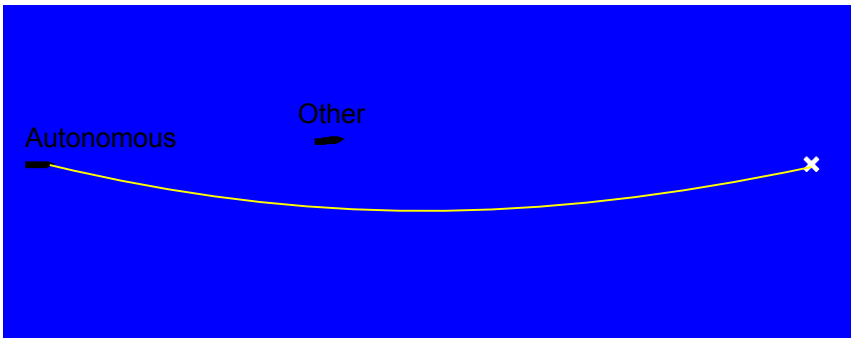
- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$

- Entity *Other* erzeugen: Typ: *Machine Vessel*, $x = -100$, $y = 25$, $v = 0$, $\varphi = 0.1$

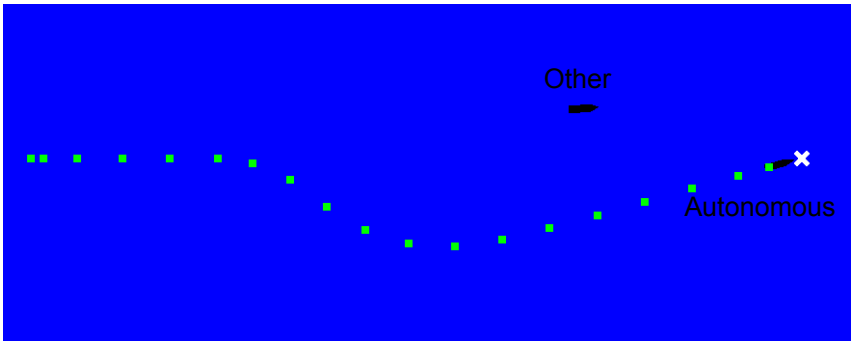
Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = 0.1$, $v = 3$

Der erwartete und gefahrene Pfad sind in Abb. 5.8 dargestellt. Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Ähnlich der Regel 12 beginnt es das Ausweichmanöver jedoch erst spät. Die Regel wird dennoch korrekt erfüllt, indem das überholende Schiff ausweicht.



(a)



(b)

Abbildung 5.8: Evaluation Regel 13: (a) erwarteter Pfad; (b) gefahrener Pfad.

5.1.9 Regel 14: Entgegengesetzte Kurse

Das Szenario wird wie folgt aufgebaut:

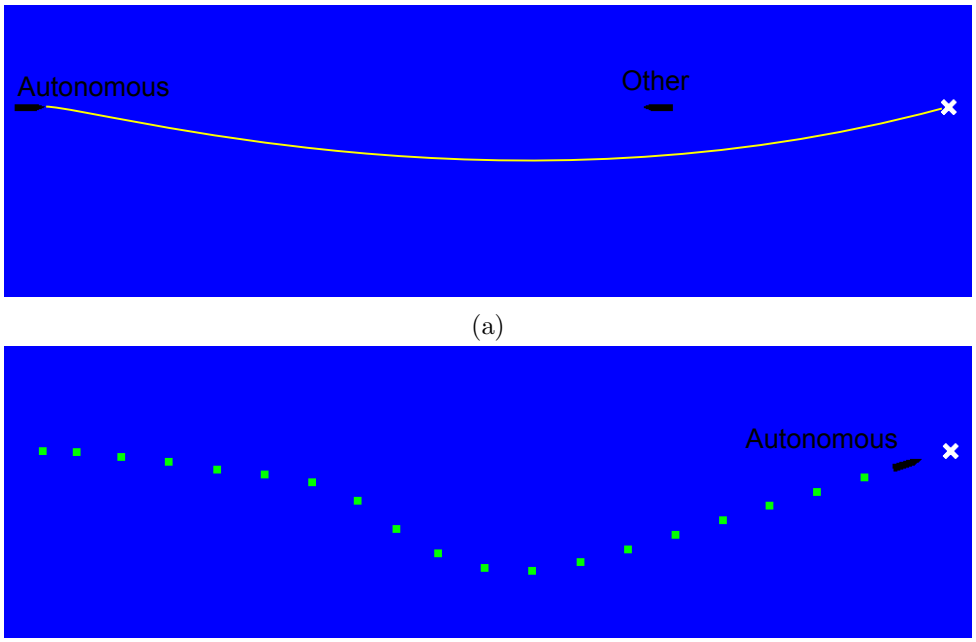
- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -450$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Machine Vessel*, $x = 200$, $y = 0$, $v = 0$, $\varphi = 3.14$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 500$, $y = 0$

- Entity *Other* Guidance: *Course*: $\varphi = 3$, $v = 10$

Der erwartete und gefahrene Pfad sind in Abb. 5.9 dargestellt. Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Ähnlich der Regel 12 beginnt es das Ausweichmanöver jedoch erst spät. Die Regel wird dennoch korrekt erfüllt, indem das autonome Schiff so ausweicht, dass es das andere Schiff an seiner Backbordseite hat.



(a)

(b)

Abbildung 5.9: Evaluation Regel 14: (a) erwarteter Pfad; (b) gefahrener Pfad.

5.1.10 Regel 15: Kreuzende Kurse

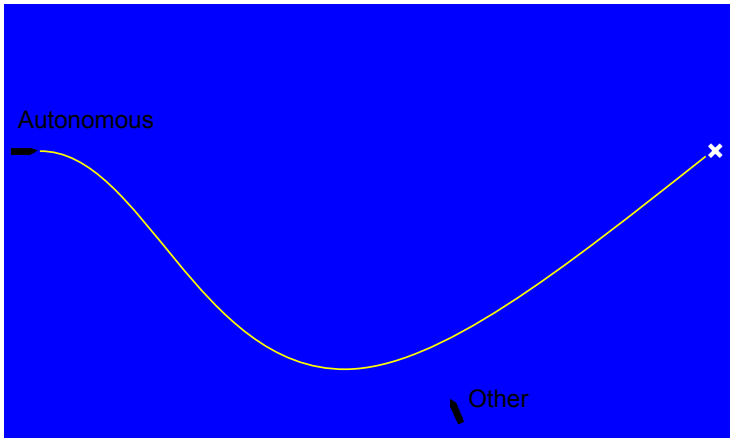
Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Machine Vessel*, $x = 100$, $y = -300$, $v = 0$, $\varphi = 2$

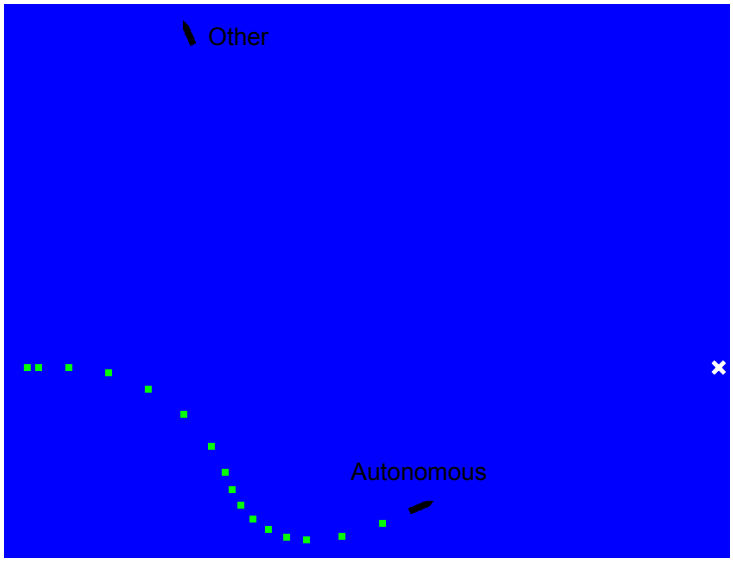
Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = 2$, $v = 10$

Der erwartete und gefahrene Pfad sind in Abb. 5.10 dargestellt. Bei dem gefahrenen Pfad ist die Simulation vor Annäherung des Ziels beendet, um die Situation gänzlich darstellen zu können. Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Nachdem das andere Schiff vorbei ist, wird jedoch erst spät auf das Ziel eingelenkt. Die Regel wird dennoch korrekt erfüllt, indem das autonome Schiff die Vorfahrt des anderen beachtet und dabei den Bug nicht kreuzt.



(a)



(b)

Abbildung 5.10: Evaluation Regel 15: (a) erwarteter Pfad; (b) gefahrener Pfad (ohne Zielannäherung).

5.1.11 Regel 17: Manöver des letzten Augenblicks

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Machine Vessel*, $x = 100$, $y = 300$, $v = 0$, $\varphi = -2$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = -2.$, $v = 10$

Der erwartete und gefahrene Pfad sind in Abb. 5.11 dargestellt. Bei dem gefahrenen Pfad ist die Simulation vor Annäherung des Ziels beendet, um die Situation gänzlich darstellen zu können.

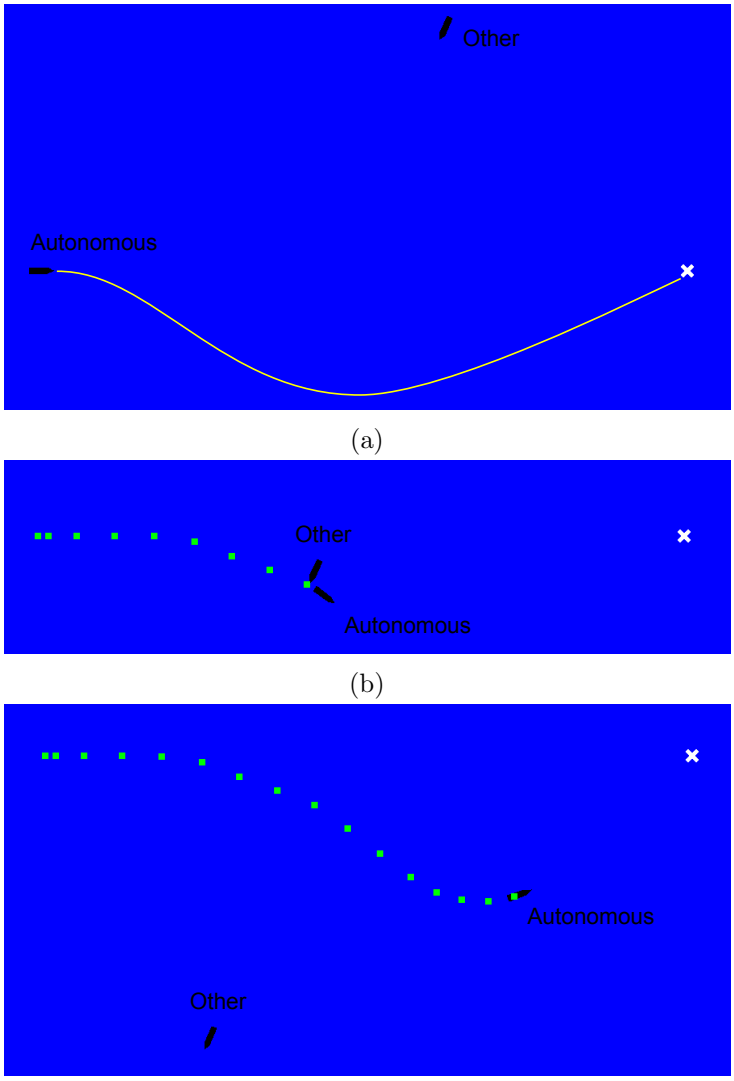


Abbildung 5.11: Evaluation Regel 17: (a) erwarteter Pfad; (b) nahe Annäherung;(c) gefahrener Pfad (ohne Zielannäherung).

Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Allerdings ist die Ausweichbewegung nicht ausreichend, da der Abstand zwischen den Schiffen sehr klein ist (cf. Abb. 5.11b). Die Regel wird dennoch korrekt erfüllt, weil das autonome Schiff eine Ausweichbewegung einleitet, obwohl es Vorfahrt hat (cf. *Regel 15: Kreuzende Kurse*) und nicht durch eine Kursänderung nach Backbord ausweicht (cf. Kapitel 2.1.2, *Regel 17: Maßnahmen des Kurshalters*).

5.1.12 Regel 18: Verantwortlichkeiten der Fahrzeuge untereinander

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Fishing Vessel*, $x = 100$, $y = 150$, $v = 0$, $\varphi = -2.3$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = -2.3$, $v = 5$

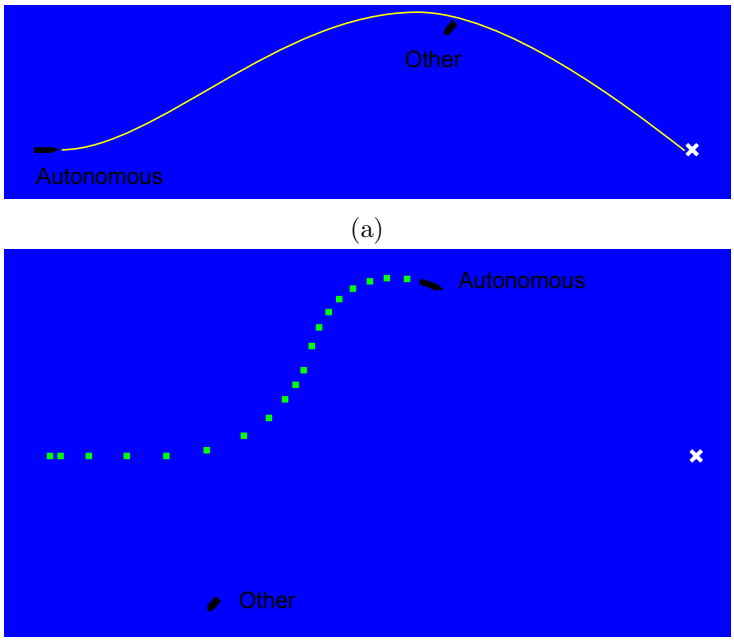


Abbildung 5.12: Evaluation Regel 18: (a) erwarteter Pfad; (b) gefahrener Pfad (ohne Zielannäherung).

Der erwartete und gefahrene Pfad sind in Abb. 5.12 dargestellt. Bei dem gefahrenen Pfad ist die Simulation vor Annäherung des Ziels beendet, um die Situation gänzlich darstellen zu können. Das Schiff weicht entsprechend dem erwarteten Pfad dem anderen Schiff aus. Nachdem das andere Schiff vorbei ist, wird jedoch — ähnlich zu Regel 15 — erst spät auf das Ziel eingelenkt. Die Regel wird dennoch korrekt erfüllt, indem das autonome Schiff die Vorfahrt des fischenden Schiffs beachtet und diesem ausweicht.

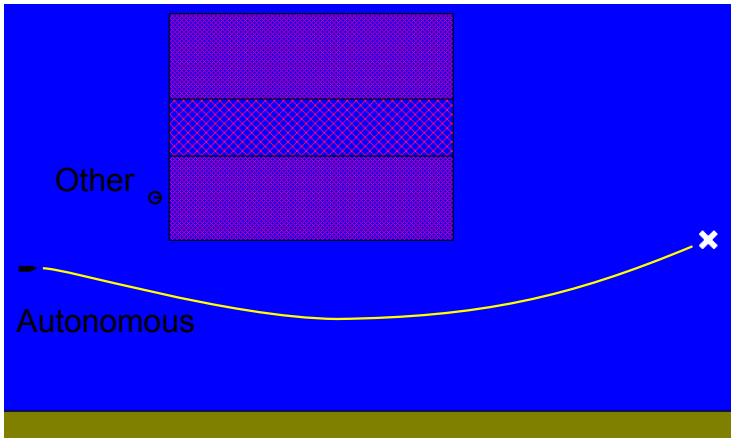
5.1.13 Backtracking

Das Szenario wird wie folgt aufgebaut:

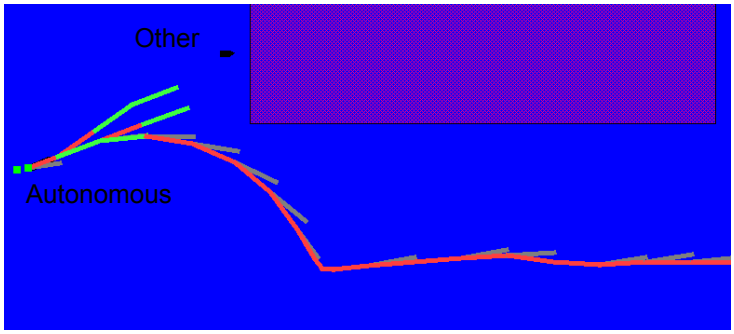
- Szenario: *Evaluation BT2*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -500$, $y = 250$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Sailing Vessel*, $x = -275$, $y = 375$, $v = 0$, $\varphi = 0$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

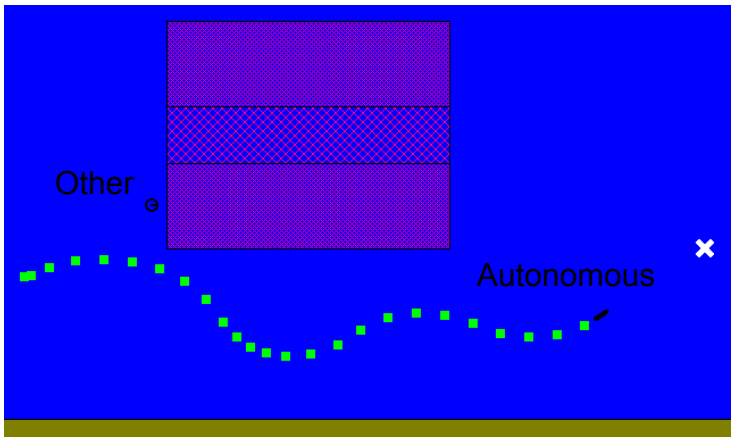
- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 700$, $y = 300$



(a)



(b)



(c)

Abbildung 5.13: Evaluation des *Backtracking*-Algorithmus: (a) erwarteter Pfad; (b) Aufbau des vom lokalen Planer ermittelten Weges (graue Linie zeigt Schritt zum Ziel, rote Linie nach Berücksichtigung der aufgrund des Potentialfeldes wirkenden Kräfte und grüne Linie den Schritt durch *Backtracking*-Algorithmus); (c) gefahrener Pfad.

Der erwartete und gefahrene Pfad, sowie eine Visualisierung des angewandten *Backtracking*-Algorithmus sind in Abb. 5.1 dargestellt. Nach Regel 10 (Verkehrstrennungsgebiete) müsste das autonome Schiff das Verkehrstrennungsgebiet nutzen. Allerdings ist das Gebiet hier schmal gewählt und an der Einfahrt befindet sich ein Segelschiff, dem das autonome Schiff nach Regel 18 (Verantwortlichkeiten der Fahrzeuge untereinander) ausweichen muss. In dieser Situation ist die Einfahrt in das Verkehrstrennungsgebiet also blockiert. Abb. 5.13b zeigt die Anwendung des *Backtracking*-Algorithmus, der das Schiff trotz der eigentlich zur Einfahrt hin wirkenden Kräfte

des Potentialfeldes an dem Gebiet vorbeifahren lässt. Dieser Testfall zeigt, dass der Algorithmus funktioniert.

5.1.14 Testfall A: Kreuzende Kurse und anderer Schiffstyp

Es folgen nun einige allgemeinere Situationen, um die Kombination von mehreren Regeln zu evaluieren.

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other1* erzeugen: Typ: *Machine Vessel*, $x = 100$, $y = -300$, $v = 0$, $\varphi = 2$
- Entity *Other2* erzeugen: Typ: *Sailing Vessel*, $x = 100$, $y = 150$, $v = 0$, $\varphi = -2.3$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other1* Guidance: *Course*: $\varphi = 2$, $v = 10$
- Entity *Other2* Guidance: *Course*: $\varphi = -2.3$, $v = 5$

Der erwartete und gefahrene Pfad sind in Abb. 5.14 dargestellt. Der gefahrene Pfad ist nur bis zum Auftreten einer problematischen Situation dargestellt. Der erwartete Pfad zeigt eine Ausweichbewegung nach unten, da das maschinengetriebene Schiff schneller fährt als das Segelschiff. So könnte durch Kursänderung nach Steuerbord eine Passage vor dem Segelschiff und hinter dem maschinengetriebenen gefunden werden. Der gefahrene Pfad zeigt allerdings, bevor das Segelschiff passiert wird, dass das autonome Schiff rückwärts fährt. Es hätte allerdings (durch das statische Bild leider nicht gut erkennbar) mit ausreichend Abstand vor diesem fahren können.

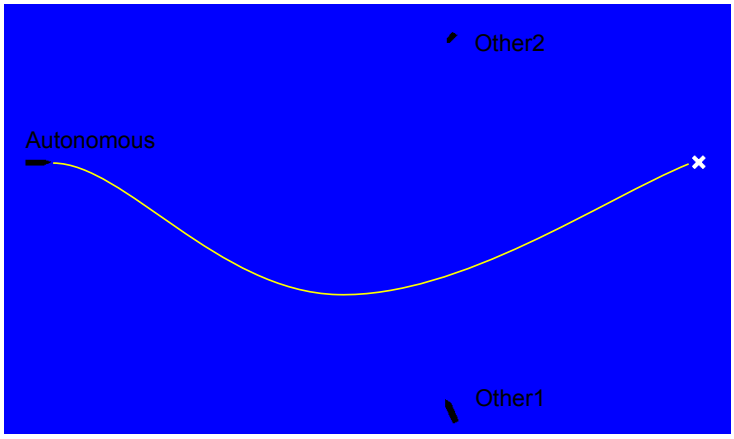
5.1.15 Testfall B: Überholen und entgegengesetzte Kurse

Das Szenario wird wie folgt aufgebaut:

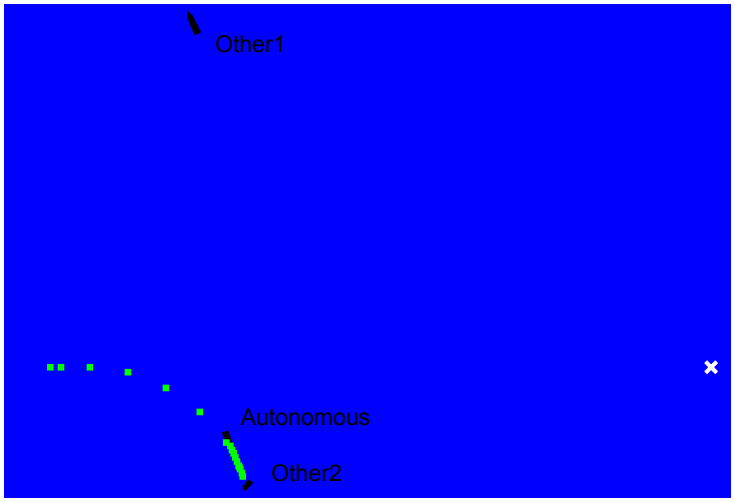
- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other1* erzeugen: Typ: *Machine Vessel*, $x = -100$, $y = -50$, $v = 0$, $\varphi = 0$
- Entity *Other2* erzeugen: Typ: *Machine Vessel*, $x = 350$, $y = 50$, $v = 0$, $\varphi = 3.14$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 500$, $y = 0$
- Entity *Other1* Guidance: *Course*: $\varphi = 0$, $v = 2$
- Entity *Other2* Guidance: *Course*: $\varphi = 3.14$, $v = 10$

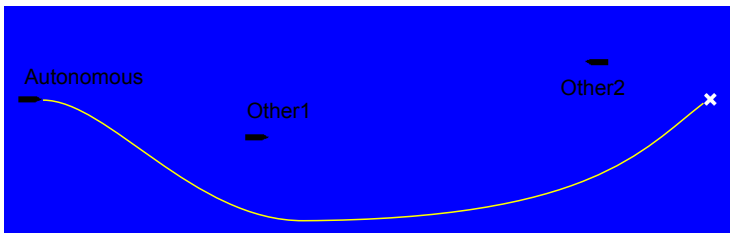


(a)

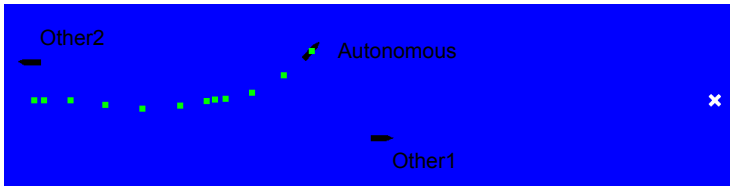


(b)

Abbildung 5.14: Evaluation Testfall A: (a) erwarteter Pfad; (b) gefahrener Pfad.



(a)



(b)

Abbildung 5.15: Evaluation Testfall B: (a) erwarteter Pfad; (b) gefahrener Pfad.

Der erwartete und gefahrene Pfad sind in Abb. 5.15 dargestellt. Der gefahrene Pfad ist nur bis zur Auflösung des Konflikts der Situation dargestellt. Der erwartete Pfad zeigt eine Ausweich-

bewegung nach unten, da das zu überholende Schiff aufgrund des entgegenkommenden nicht oben sicher passiert werden sollte. Der gefahrene Pfad zeigt allerdings, dass das autonome Schiff abbremst und zum Stillstand kommt, bis das entgegenkommende vorbei ist. Anschließend wird das zu überholende Schiff auf dessen Backbordseite passiert.

5.1.16 Testfall C: Stehendes Hindernis

Das Szenario wird wie folgt aufgebaut:

- Szenario: *Evaluation*
- Entity *Autonomous* erzeugen: Typ: *Machine Vessel*, $x = -400$, $y = 0$, $v = 0$, $\varphi = 0$
- Entity *Other* erzeugen: Typ: *Sailing Vessel*, $x = 0$, $y = 0$, $v = 0$, $\varphi = 1.57$

Folgende Einstellungen werden vorgenommen, bevor die Übung gestartet wird:

- Entity *Autonomous* Guidance: *Autonomous*, Zielort: $x = 400$, $y = 0$
- Entity *Other* Guidance: *Course*: $\varphi = 1.57$, $v = 0$

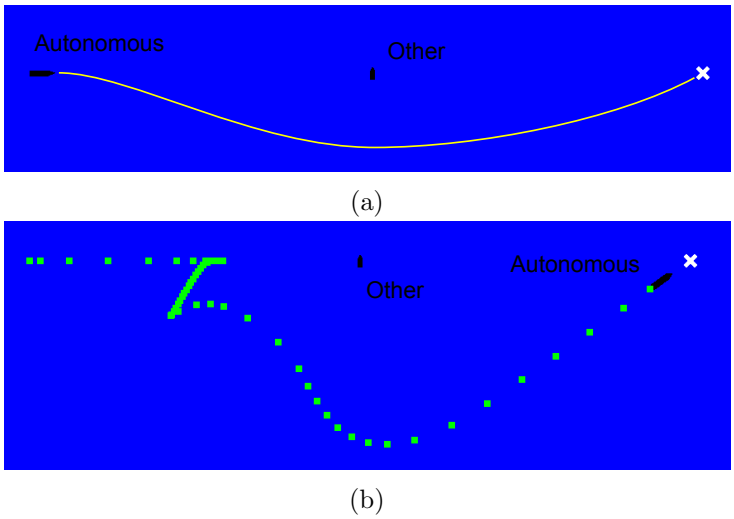


Abbildung 5.16: Evaluation Testfall C: (a) erwarteter Pfad; (b) gefahrener Pfad.

Der erwartete und gefahrene Pfad sind in Abb. 5.16 dargestellt. Der erwartete Pfad zeigt eine Ausweichbewegung nach unten. Der gefahrene Pfad zeigt allerdings, dass das autonome Schiff vor dem Hindernis (anderes Schiff) abbremst und anschließend rückwärts fahrend in eine Richtung lenkt, bis es so weit entfernt ist, dass es das andere Schiff umfahren kann. Anschließend wird das Segelschiff passiert.

5.2 Auswertung

Die Ergebnisse der verschiedenen Testfälle zeigen, dass die Regeln korrekt berücksichtigt werden. Ausweichmanöver werden bei Bedarf eingeleitet und bevorzugte Pfade werden genommen. Auch die Annäherung an ein Ziel erfolgt derart, dass das Schiff bei Erreichen des Ziels still steht. Lediglich das Kreuzen von Verkehrstrennungsgebieten zeigt einen zu starken Ausschlag entgegen

der vorgegebenen Fahrtrichtung innerhalb des Gebiets. Hinzu kommt, dass in vielen Fällen die gefahrenen Pfade spätere Reaktionen zeigen als für einen optimalen Pfad möglich. So werden Ausweichmanöver erst spät begonnen oder trotz bereits passiertem Hindernis (anderes Schiff) wird erst spät wieder auf das Ziel eingelenkt. Häufig wird infolge der Ausweichbewegung neben dem Kurs auch die Geschwindigkeit verändert, teilweise sogar fast bis zum Stillstand, was nach Regel 8 bei ausreichend Seeraum vermieden werden soll. Zusätzlich zeigen die gefahrenen Wege gewisse Instabilitäten im Sinne von Schlangenlinien anstatt gerader Strecken (cf. Abb. 5.5 und Abb. 5.4).

Der globale Planer zeigt im entsprechenden Testfall (cf. Kapitel 5.1.1) eine Route als Folge von konvexen Polygonen. Diese Route wird durch das lokale Verhalten problemlos abgefahren und hält dabei einen gewissen Abstand von Küstenlinien. Auch der *Backtracking*-Algorithmus erkennt im Testfall (cf. Kapitel 5.1.13) die versperrte Einfahrt des Verkehrstrennungsgebiets und findet den Weg an diesem vorbei. Die Anwendung des Algorithmus in der Situation ist also sinnvoll und korrekt. Diese Testfälle zeigen ein funktionierendes Zusammenspiel zwischen globalem Planer, lokalem Verhalten und *Backtracking*-Algorithmus.

Die Testfälle, die hier nicht zur Evaluation einer einzelnen Regel angemerkt sind (cf. Kapitel 5.1.14 bis Kapitel 5.1.16), zeigen allerdings, dass das Verfahren in bestimmten Situationen fehlschlägt. So können in Konflikt stehende Ausweichvorgaben zu einem unerwarteten Abbremsen führen, obwohl durch eine Kursänderung allein die Situation ebenfalls behoben werden könnte. Ebenfalls können bestimmte Grenzwertsituationen (cf. Kapitel 5.1.16) zu diesem Problem führen.

Kapitel 6

Schlussbetrachtung

Zum Abschluss dieser Master-Thesis sollen eine Analyse des gewählten Verfahrens und dessen Umsetzung durchgeführt sowie Weiterentwicklungs- und Verbesserungsmöglichkeiten diskutiert werden.

6.1 Analyse

Die Evaluation (cf. Kapitel 5) zeigt, dass das entwickelte autonome Verhalten die Regeln beachtet und sich entsprechend verhält. Auffällig für diese Methode ist ein verzögertes Manöververhalten im Vergleich zur korrespondierenden menschlichen Verhaltensweise in der Praxis. Zu spätes Einleiten von Ausweichbewegungen oder verzögertes Zurücklenken in Richtung des gegebenen Zielortes folgen aus der Wirkung der Potentialfelder. Abhängig von der Lage der Potentialquelle kann das autonome Schiff vom Zielort weggedrängt werden, obwohl ein Schiff mit Vorfahrt bereits passiert und der Weg zum Ziel wieder frei ist. Hier spielen die Umsetzungen der Regelmodule eine große Rolle, da die relevanten Schritte dieser Module (Beginn einer Situation, Ende einer Situation und Verarbeitung einer Situation; cf. Kapitel 4.2.4.3) die Lage und Existenz sowie Art der Potentialquellen festlegen.

Der gewählte Aufbau des Kontrollzyklus in eine Auftragsplanung, einen globalen und einen lokalen Planer stellt eine sinnvolle Trennung über definierte Schnittstellen dar. Insbesondere die Definition der Route als Schnittstelle zwischen globalem Planer und lokalem Verhalten (cf. Kapitel 4.1.1.2) bietet sehr gute Möglichkeiten, die Aufgaben der beiden Planer klar voneinander zu trennen. So kann sich das lokale Verhalten ohne eine rechenaufwendige Verarbeitung des Terrains auf das Folgen der Route und die Vermeidung von Hindernissen durch Beachtung einer Menge von Regeln konzentrieren. Die vorgegebene Route grenzt den befahrbaren Bereich allerdings auch ein (cf. Kapitel 4.4). Hier könnte eine Erweiterung der Definition einer Route um benachbarte Polygone des *Navigation Mesh* zu einer weniger strengen Begrenzung führen.

Bei dem lokalen Verhalten zeigt sich die gewählte Architektur durch den modularen Aufbau als hilfreich bei der Entwicklung einzelner Regelmodule. Die Verwendung eines einzelnen Kombinationsmoduls führt zur Ausgabe eindeutiger Steuerkommandos. Durch die definierte Schnittstelle über eine Klassenhierarchie von Verhaltensdaten ist diese Schnittstelle beliebig erweiterbar. Allerdings lassen sich die Regelmodule nicht so strikt wie hier angenommen voneinander getrennt betrachten. So kann es vorkommen, dass eine Situation entgegenkommender, maschinengetriebener Schiffe (i. e. Regel 14) dazu führt, dass das autonome Schiff ausweicht. Dadurch ändert sich allerdings die relative Orientierung der Schiffe zueinander, sodass es zusätzlich zu einer Situation kreuzender Schiffe (i. e. Regel 15) kommt. In einem solchen Fall muss beachtet werden,

dass das Schiff für das bestimmte andere an der Situation beteiligte Schiff bereits in der Befolgung einer Regel ist und nicht eine andere Regel befolgen soll. Um dieses Problem zu lösen, ließe sich ein zusätzliches Modul zur komplexeren Situationskontrolle zwischen Schiffen entwickeln, welches von den Regelmodulen gespeist wird und daraus entsprechende Potentialquellen an das Kombinationsmodul weitergibt.

Nun sei noch speziell das Kombinationsmodul untersucht. Dieses stellt den Kern des Verfahrens im lokalen Verhalten dar, da es die Informationen aller Regelmodule auswertet. Damit agiert es unabhängig von den Entscheidungen, ob Situationen vorliegen oder beendet sind. Es nutzt lediglich Potentialquellen, um einen möglichst günstigen Weg in Richtung Zielort bzw. nächstem *Gate* zu finden. In dieser Arbeit werden zwei Verfahren zur Nutzung der Informationen aus dem Potentialfeld vorgestellt. Das erste basiert auf dem Generieren einer Menge von Pfaden auf Basis der kinematischen Beschränkungen und Bewertung dieser Pfade, ohne die Kräfterichtungen zu beachten (cf. Kapitel 4.1.1.5, *Kombinationsansatz Suchbaum*). Für Echtzeit-Simulationen ist dieser Ansatz aufgrund seiner Laufzeit nicht geeignet. Das zweite und hier umgesetzte Verfahren generiert einen Pfad durch Erzeugen eines Punktes mit optimaler Steuerung in Richtung Ziel und anschließend durch Veränderung dieses Punktes durch die aufgrund des Potentialfeldes wirkenden Kräfte (cf. Kapitel 4.1.1.5, *Kombinationsansatz Potentialfeld*). Der in dieser Arbeit verwendete Kombinationsansatz führt allerdings, insbesondere bei komplexeren Situationen, aber auch teilweise bei simplen Situationen mit ungünstig gewählter Potentialquelle, zu nicht gewünschtem Verhalten des autonomen Schiffs (cf. Kapitel 5.1.14 bis Kapitel 5.1.16). Dies liegt an einigen Problemen, die aus der Verwendung von Potentialfeldern folgen:

- Die Potentialfelder wirken erst in einer gewissen Reichweite bzw. einem bestimmten Abstand von der Potentialquelle. Wird die Quelle also nur in die Nähe des Hindernisses gelegt, so wird ein Ausweichmanöver auch erst ab Erreichen dieser Reichweite begonnen, da das Kombinationsmodul lediglich Punkte verändert, auf die eine Kraft aufgrund des Potentialfeldes wirkt.

- Werden Potentialquellen ungünstig gelegt, kann es passieren, dass eine Kraft auf einen Punkt lediglich abstoßend entgegen seiner Fahrtrichtung wirkt, nicht jedoch zu einer Seite. In einem solchen Fall kann das autonome Schiff nur abbremsen, allerdings nicht zur Seite ausweichen, da es keine Information darüber hat, in welche Richtung es ausweichen soll — wie zum Beispiel in Kapitel 5.1.16 zu sehen. Entsprechend ist die Wahl der Potentialquellen wichtig.

- Auch bei günstiger Wahl der Potentialquellen können Situationen auftreten, in denen das Schiff in eine Falle fährt, da widersprüchliche Ausweichvorgaben von Modulen gegeben werden (e. g. eine Potentialquelle vor dem autonomen Schiff und links sowie eine weitere vor dem Schiff und rechts von diesem).

- Bei dem verwendeten Verfahren, folgende Punkte im lokalen Pfad durch das Potentialfeld zu verändern, können aus dem Pfad des lokalen Planers keine diskreten Entscheidungen (e. g. *ausweichen nach links* oder *abbremsen und ausweichen nach rechts*) abgeleitet werden, e. g. die Veränderung des ersten Punktes des Pfades zur Steuerbordseite könnte von einer Veränderung des zweiten Punktes zur Backbordseite gefolgt werden. Der lokale Planer trifft Entscheidungen über Steuerkommandos also nur aufgrund der Situation zu einem bestimmten Zeitpunkt, was zu einer *Instabilität* des Pfades führt, e. g. Schlangenlinien. Die Vorausplanung über einen gegebenen Zeitraum hat lediglich bedingt Einfluss durch die Prüfung, ob durch getroffene Entscheidungen im Pfad keine ungültige Situation, i. e. zu starke Kraft durch das Potentialfeld, entstehen kann.

Um einen Teil dieser Probleme von Potentialfeldern zu beheben, insbesondere die Wirkung über eine begrenzte Entfernung und die Möglichkeit von Fallen im Potentialfeld, wird der *Backtracking*-Algorithmus verwendet. In Kombination mit der Vorausplanung des lokalen Verhaltens kann so in gewissem Maße verhindert werden, dass das Schiff in Fallen gerät. Allerdings greift der Algorithmus nur dann, wenn eine vorausgeplante Position ungültig ist, also die auf diese Position aufgrund des Potentialfeldes wirkenden Kräfte zu groß sind oder die Position nicht mehr innerhalb der Route liegt. Häufig führen die wirkenden Kräfte aber dazu, dass das autonome Schiff abbremst und zum Stillstand kommt, was keine ungültige Position ist. In diesen Fällen findet der *Backtracking*-Algorithmus zur Zeit keine Anwendung. Durch geeignete Vorgaben zur Geschwindigkeitsregelung könnte dies mit eingebracht werden.

Dennoch überwiegen die oben aufgeführten Probleme und so kann man folgern, dass der gewählte Ansatz für das Kombinationsmodul nicht geeignet ist. Es ist zu stark abhängig davon, wie die vorgeschalteten Module Potentialquellen vorgeben, und selbst dann kann durch mehrere Vorgaben ein fehlerhaftes Verhalten durch das autonome Schiff auftreten.

Stattdessen bietet der *Kombinationsansatz Suchbaum* einen vielversprechenden Ansatz. Die generierten Pfade werden nicht vom Potentialfeld beeinflusst, sondern lediglich durch dieses bewertet. Bei entsprechend gut gewählter Bewertungsfunktion, die sowohl das Voranschreiten zum Ziel als auch die Stärke auf die Punkte des Pfades wirkender Kräfte und die Länge des Pfades berücksichtigt, kann so ein optimaler Pfad ermittelt werden. Dieser Ansatz ist zwar noch abhängig von den gegebenen Potentialquellen, allerdings kann er besser den gesamten geplanten Zeitraum berücksichtigen und ist nicht durch eine entgegen der Fahrtrichtung des autonomen Schiffs wirkenden Kraft eingeschränkt in der Ermittlung eines Pfades. Hierfür gibt es nämlich Alternativpfade, die auf Kursänderungen basieren.

Das Hauptproblem, aus dem der Ansatz in dieser Arbeit verworfen ist, bleibt allerdings bestehen: Der Rechenaufwand zum Generieren des Baums bei kleinem Zeitschritt und großem vorausgeplanten Zeitraum ist aufgrund der Breite zu groß, um sinnvoll eingesetzt zu werden. Um dies zu beheben, gibt es verschiedene Möglichkeiten:

- Optimierung der verwendeten Algorithmen.
- Technische Maßnahmen, e. g. Parallelisierung beim Aufbau des Baums.
- Priorisierter Aufbau von Pfaden. Dabei werden bevorzugt zu verwendende Pfade zuerst aufgebaut, e. g. Pfade mit Verwendung von negativer Beschleunigung werden zunächst nicht betrachtet. Wird bereits ein Pfad mit einer guten Bewertung gefunden, kann dieser verwendet werden, ohne andere zu betrachten.

Die Analyse ergibt also, dass der Aufbau des autonomen Verhaltens mit den definierten Schnittstellen sowie der globale Planer gut funktionieren. Die Architektur des lokalen Verhaltens ist flexibel gehalten und dadurch gut erweiterbar. Die Umsetzung der Regelmodule und das Konzept des gewählten Kombinationsansatzes weisen allerdings Probleme auf. Die Umsetzung der Regelmodule kann durch nähere Betrachtung und Überarbeitung der gewählten Entscheidungsgrenzen für Beginn und Ende einer Situation sowie der Lagen von Potentialquellen verbessert werden. Zusätzlich ist eine Kommunikation zwischen Regelmodulen unumgänglich, was durch den modularen Aufbau gut möglich ist. Der gewählte Kombinationsansatz im Kombinationsmodul über Potentialfelder und durch diese wirkende Kräfte kann jedoch nicht genutzt werden. Der *Kombinationsansatz Suchbaum* ist möglicherweise besser geeignet.

6.2 Weiterentwicklung

Die Arbeit bietet eine Grundlage für diverse Weiterentwicklungen. So stellt sie über den Simulator bereits eine Infrastuktur zum Testen und Untersuchen von implementierten Verfahren bereit. Auch die Architektur des autonomen Verhaltens bietet gute Möglichkeiten für die Entwicklung anderer Verfahren oder einzelner Aspekte des hier vorgestellten.

Eine bereits im Konzept des autonomen Verhaltens angesprochene Entwicklungsmöglichkeit stellt das automatische Generieren des *Navigation Mesh* dar. Dieses wird hier noch manuell definiert, kann aber bei automatischem Generieren zusätzliche schiffsspezifische Parameter, e. g. Tiefgang, berücksichtigen und dadurch zusätzliche Daten des Terrains für ein indiviuelles *Navigation Mesh* heranziehen. Da die Lage der konvexen Polygone des *Navigation Mesh* einen Einfluss auf die Route des globalen Planers hat, ist die Entwicklung eines Bewertungsalgorithmus für das generierte *Navigation Mesh* naheliegend.

Die Umsetzung des hier nicht verwendeten Kombinationsansatzes (*Kombinationsansatz Suchbaum*) bietet eine komplexe, aber vielversprechende Verbesserung des autonomen Verhaltens. Durch Lösung des Hauptproblems — Vorschläge für die Lösung sind oben erwähnt — könnte dieser Ansatz Anwendung finden und viele Probleme des in dieser Arbeit entwickelten Verhaltens lösen.

Zusätzlich lassen sich die einzelnen Regelmodule verbessern. Dazu müssen Entscheidungen, ob die Regel gilt oder nicht, sowie die Lage der Potentiale überarbeitet werden. Außerdem muss eine Kommunikation zwischen den Modulen bzw. eine modulübergreifende Situationsverwaltung entwickelt werden.

Davon abgesehen kann das Verfahren — wenn auch mit hohem Implementierungsaufwand verbunden, da das autonome Verhalten lediglich auf zwei Dimensionen ausgelegt ist — auf Fahrzeuge oder Flugzeuge erweitert und angewendet werden. Bei der Erweiterung auf Flugzeuge ist zusätzlich eine Erweiterung der Definition einer Route von konvexen Polygonen auf konvexe Polyeder notwendig.

6.3 Schlussbemerkung

Abschließend lässt sich sagen, dass das gewählte Verfahren, insbesondere der globale Planer sowie die Architektur des autonomen Verhaltens, einen Ansatz bilden, der zur regelbasierten autonomen Steuerung — hier anhand von Schiffen — verwendet werden kann. Die Nutzung von Potentialfeldern zur Kombination verschiedener Verhaltensmodule bietet einen vielversprechenden Ansatz, benötigt allerdings eine Erweiterung des *Backtracking*-Algorithmus, um Situationen zu verhindern, in denen ein Schiff aufgrund ungünstiger Potentialquellen abbremmst anstatt das Hindernis zu umfahren. Eine Umsetzung des *Kombinationsansatzes Suchbaum* kann dieses Problem eventuell beheben, da mehr Alternativpfade untersucht werden. Davon abgesehen müssen auch die einzelnen Regeln genauer betrachtet werden, um die Potentialquellen bei der Wahl dieses Verfahrens präziser legen zu können. Dazu gehört eine bessere Wahl der Bedingungen für einen Beginn und ein Ende der durch die jeweilige Regel hergestellten Situation. Außerdem ist eine Kommunikation zwischen den Verhaltensmodulen unerlässlich, da, wie in Kapitel 6.1 erläutert, die Ausführung einer Regel nicht zum Beginn einer Situation einer anderen Regel führen sollte.

Die Arbeit fand in einem begrenzten Zeitraum statt und der Inhalt war zu komplex, um alle Bereiche zu optimieren. Der zusätzliche Aufwand durch die Entwicklung der verwendeten Simulation hat den benötigten Zeitaufwand erhöht. Viel mehr bildet diese Master-Thesis also einen

Ansatz für ein Verfahren, das in seinen Bestandteilen näher betrachtet werden muss, wofür in Kapitel 6.2 einige Möglichkeiten genannt werden. Die erzeugte Software ist derart aufgebaut, um eben solche Weiterentwicklungen der verschiedenen Bereiche, i. e. Regelmodule, Kombinationsmodul, globaler Planer inklusive *Navigation Mesh*, durch weitere Forschungsarbeiten zu unterstützen.

Literaturverzeichnis

[1] Howie M. Choset. *Principles of robot motion: theory, algorithms, and implementation.* Intelligent robotics and autonomous agents. MIT Press, Cambridge, Mass. [u.a.], 2005. XIX, 603 S. ; 24 cm : Ill., graph. Darst.

[2] Bundesministerium der Justiz und für Verbraucherschutz. Internationale Regeln von 1972 zur Verhütung von Zusammenstößen auf See (Kollisionsverhütungsregeln - KVR), 2009. Available online: `http://www.gesetze-im-internet.de/bundesrecht/seestro_1972/gesamt.pdf`.

[3] Christer Ericson. *Real-time collision detection.* Morgan Kaufmann series in interactive 3D technology. Morgan Kaufmann, San Francisco, Calif. [u.a.], 2005.

[4] T. P. Hartley and Q. H. Mehdi. In-game adaptation of a navigation mesh cell path. In *Computer Games (CGAMES), 2012 17th International Conference on.*

[5] Gregory Dudek; Michael Jenkin. *Computational principles of mobile robotics.* Cambridge Univ. Press, Cambridge [u.a.], 2000. XII, 280 S. ; 26 cm : Ill., graph. Darst.

[6] Frank Kirchner. *Künstliche Intelligenz: ein moderner Ansatz.* it, Informatik. Pearson, Higher Education, München [u.a.], 3., aktualisierte Aufl. edition, 2012.

[7] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Computer Science Dept, Iowa State University, Tech. Rep. TR. 1998, S. 98–11. Available online: `http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf`.

[8] Joachim Hertzberg; Kai Lingemann. *Mobile Roboter: Eine Einführung aus Sicht der Informatik.* eXamen.press. Springer Vieweg, Berlin [u.a.], 2012.

[9] MUNIN. Maritime Unmanned Navigation through Intelligence in Networks. Available online: `http://www.unmanned-ship.org/munin/`.

[10] International Maritime Organization. Convention on the international regulations for preventing collisions at sea. Available online: `http://www.navcen.uscg.gov/?pageName=navRulesContent`.

[11] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler Band 1: ein Lehr- und Arbeitsbuch für das Grundstudium.* Vieweg Verlag, Wiesbaden, 10., erweiterte Auflage edition, 2001.

[12] Karsten Berns; Ewald von Puttkamer. *Autonomous Land Vehicles: Steps towards Service Robots.* IT. Vieweg + Teubner, Wiesbaden, 1. ed. edition, 2009.

[13] Wikipedia. Kollisionsverhütungsregeln (kvr), 2013. Available online: `http://de.wikipedia.org/wiki/Kollisionsverh%C3%BCtungsregeln`.