technische universität
dortmund

Abschlussbericht

PG565 – Kooperatives Data-Mining mit vernetzten Robotern

David Arnu, Benjamin Berghoff,
Jan Czogalla, David Gräff,
Manuel Groß, Sibylle Hess,
André Johnigk, Jens Möllmer,
Fabian Peternek, Matthias Petsch,
Patrick Vorlicek

25. April 2013

Betreuerin: Prof. Dr. Katharina Morik
Mitwirkende: Prof. Dr.-Ing. Christian Wietfeld
Dipl.-Inf. Daniel Behnke,
Dipl.-Inf. Hendrik Blom,
Dipl.-Inf. Niklas Goddemeier

Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz
Prof. Dr. Katharina Morik

Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Christian Wietfeld

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Eine stetig wachsende Bevölkerungsdichte und ein entsprechend steigender Bedarf an Energie erfordern die Suche nach innovativen Möglichkeiten der Energienutzung, insbesondere in Städten. Das Konzept der *Smart City*, als Beschreibung für einen intelligenten Aufbau von Logistik und Politik in den (Groß-)Städten der Zukunft, ist in der aktuellen Forschung dementsprechend von großem Interesse. Als intelligent gilt hierbei eine für Mensch und Natur gleichermaßen vorteilhafte Nutzung von Ressourcen, auch unter Gebrauch der gegebenen technischen Möglichkeiten.

Durch den Einsatz von verschiedenen Sensoren und Aktoren kann eine Umgebung intelligent gestaltet werden. Diese müssen nicht fest installiert sein, sondern können sich teilweise dynamisch oder frei bewegen. Um diese einzelnen Sensoren zu einem Netz zu verknüpfen, bedarf es einer geeigneten Kommunikation. Dabei ist zu beachten, dass die Anzahl der Knoten variieren kann und die Netze nicht auf eine bestimmte Aufgabe beschränkt sind.

Der Einsatz von unbemannten, autonomen Robotern bietet in diesem Zusammenhang einen interessanten Anwendungsfall. Insbesondere in den Gebieten der Informationstechnik, Logistik und Kommunikation lassen sich viele aussichtsreiche Anwendungen finden. Mögliche Einsatzgebiete sind Güter-/Personentransport und Erkundungsaufgaben in gefährlichen oder schwer zugänglichen Gebieten, um eventuelle Risiken für Menschen zu reduzieren. Wichtige Herausforderungen dabei sind die Energieversorgung der Roboter und ihre Autonomie. Gerade bei Erkundungseinsätzen in gefährlichem Terrain kann keine stabile Infrastruktur zur Energieversorgung gewährleistet werden. Um in einem solchen Fall die Funktionsfähigkeit der Roboter über einen längeren Zeitraum zu gewährleisten, kann Energie aus der Umgebung gewonnen werden, ein Vorgang der als *Energy-Harvesting* [18] bezeichnet wird.

Eine Idee ist es, Solarkollektoren zur Energiegewinnung zu nutzen. Bei ausreichender Größe der Roboter können die Kollektoren direkt auf diesen installiert werden. Alternativ kann ein Versorgungsfahrzeug mit einem großen Solarkollektor ausgestattet werden und als Basisstation für die zu versorgenden Roboter dienen. Fällt der Energiestand eines Roboters unter ein kritisches Niveau, sucht dieser eine geeignete Position oder das Versorgungsfahrzeug auf, um seine Energiequelle wieder aufzuladen. Das Versorgungsfahrzeug seinerseits muss ebenfalls an geeigneter Stelle seine Energiespeicher auffüllen. Das Finden solch geeigneter Positionen ist dabei sowohl von der Umgebung als auch der Tageszeit abhängig.

Der naive Ansatz wäre, dass jeder Roboter selbst nach guten Ladepunkten sucht. Dabei wird jedoch die Möglichkeit, die eine umfassende Kommunikation bietet, nicht berücksichtigt. Aus den gemeinsamen Erkenntnissen mehrerer Roboter, die sich zu einem Schwarm zusammen schließen, ergibt sich eine viel detailliertere Sicht auf die

Umgebung. Dass die Roboter sich in der Umgebung bewegen, während sie ihre primären Aufgaben verfolgen, kann genutzt werden, um gleichzeitig Umgebungsdaten in Form von Sensormessungen zu sammeln. Jedoch muss aus dieser, wahrscheinlich sehr großen, gewonnenen Datenmenge das entscheidende Wissen erst extrahiert werden.

Hierfür kommen Methoden des Data-Minings in Frage, mit deren Hilfe die Daten effizient ausgewertet werden können. Wenn die Roboter jedoch autonom agieren sollen, kann keine zentrale datenverarbeitende Instanz gewährleistet werden, an der alle Informationen gesammelt werden. Neben der nicht sichergestellten Verfügbarkeit ist es auch wichtig zu beachten, wie viele Roboter ihr Wissen verbreiten, denn die verfügbaren Kommunikationskanäle sind in der Regel ebenfalls begrenzt. Als logische Konsequenz ist es daher sinnvoll, auch die Data-Mining-Algorithmen so zu konzipieren, dass sie zwar vom Wissen anderer profitieren, aber die Informationen so redundant verarbeitet werden, dass der Ausfall eines einzelnen Roboters keine ernsten Konsequenzen für den restlichen Schwarm hat.

Die Energieversorgung von autonomen Fahrzeugen ist eine der großen Hürden in der Entwicklung alltagstauglicher Systeme. Inwieweit sich Data-Mining und verteilte Systeme nutzen lassen, um die Energieversorgung bei Erkundungsmissionen zu gewährleisten, soll im Rahmen der Projektgruppe *PG 565 : Kooperatives Data-Mining mit vernetzten Robotern* erprobt werden. Um eine ordentliche Projektplanung ermöglichen zu können, wird vorab ein klar umrissenes Zielszenario definiert, auf welches im folgenden Abschnitt näher eingegangen wird. Als Grundlage für das Projekt stehen den Teilnehmern bereits einige wichtige Werkzeuge zur Verfügung: ein Framework für die Verarbeitung von Stream-Daten, Hard- und Software für ein unbemanntes Bodenfahrzeug (UGV[1]), sowie eine Softwareplattform für die Fahrzeugsteuerstrategien und Autopiloten. Die spannende Herausforderung ist nun, aus all diesen Teilkomponenten ein funktionsfähiges System zu erstellen.

1.1 Aufgabengebiete der Projektgruppe

Die Projektgruppe hat sich zur Aufgabe gesetzt, selbstständig ein konkretes Szenario zu erstellen und dieses innerhalb von zwei Semestern zu realisieren. Das Referenzszenario ist das Ergebnis von intensiven Diskussionen und Überlegungen in der ersten Phase der Projektgruppe. Zunächst wird im folgenden Abschnitt das Referenzszenario dargelegt. Anschließend werden im nächsten Abschnitt die daraus abgeleiteten Teilziele beschrieben und die verwendeten Komponenten erläutert.

1.1.1 Definition des Referenzszenarios

Mit dem Hintergrund der autonomen Erkundung eines Gebietes hat die Projektgruppe das Zielszenario, aus dem sich Teilziele ableiten lassen, wie folgt definiert.

[1] **U**nmanned **G**round **V**ehicle

Abbildung 1.1: Beispielhafte Heat-Map eines Parkplatzes. Grüne Bereiche markieren geeignete Zellen zum Aufladen.

Zielszenario Es soll mit Hilfe einer Gruppe von UGVs erprobt werden, inwiefern Data-Mining für das Energy-Harvesting mittels Solarenergie von Nutzen sein kann, um die künftige Energieversorgung von autonomem Robotern in Smart Cities zu verbessern. Das Einsatzgebiet (im Folgenden auch „Spielfeld“ oder „Playground“ genannt) ist ein 150×130 Meter großer, hindernisfreier Parkplatz, der aufgrund der GPS-Auflösung in Zellen mit 2×2 Metern Kantenlänge aufgeteilt ist. Die UGVs sollen diesen Parkplatz autonom als Schwarm befahren und erkunden, ohne miteinander zu kollidieren, und dabei Umgebungsdaten wie Licht und Temperatur sammeln. Diese Daten sollen mittels Data-Mining verarbeitet werden, um Gebiete mit einer hohen Energiedichte zu liefern, sodass UGVs mit niedrigem Batteriestand diese Gebiete aufsuchen können, um sich dort aufzuladen. Dabei soll das Data-Mining durch den Austausch von Informationen zwischen den einzelnen Data-Mining-Agenten auf allen UGVs in etwa die gleiche Heat-Map [39] erzeugen, die Aufschluss über die Energiedichte in einem Gebiete gibt. Außerdem soll der Data-Mining-Agent dem Mobilitäts-Agenten Gebiete liefern können, die entweder noch gar nicht oder nur teilweise erforscht sind, um die Erkundungsrate zu erhöhen und die Absprache bei der Erkundung zu gewährleisten.

Motiviert wird dieses Szenario durch die Konsequenz, dass sowohl Datenrate als auch Speicherplatz begrenzte Ressourcen sind. Deswegen ist es nicht sinnvoll, die unverarbeiteten Daten unter allen Teilnehmern des Schwarms zu verteilen.

Zusammenfassend sollen die UGVs in der Lage sein, kooperativ ein Gebiet zu erkunden und dabei die Informationen über das Gelände effizient auszutauschen und aus diesem Wissen neue Erkenntnisse abzuleiten. Die Anforderung der Effizienz spiegelt sich in zwei Aspekten wieder. Zum einen sind die Ressourcen der Hardware beschränkt, insbesondere kann nicht gewährleistet sein, dass alle gesammelten Sensorwerte gespeichert werden. Die andere Einschränkung ist die Kommunikation zwischen

den UGVs. Die benötigte Datenrate sollte beschränkt sein, da die belegten Ressourcen auch von den anderen Agenten benötigt werden. Eine zu hohe Datenrate kann außerdem für eine erhöhte Verlustrate sorgen, wenn der Kommunikationskanal überlastet wird.

1.1.2 Referenzziele

Wichtige Wegpunkte für das Erreichen des Referenzszenarios sind die folgenden Aufgaben, wie sie bereits im Projektgruppenantrag[2] erstmals erwähnt werden:

Inter-Agent-Kommunikation Die Kommunikation sowohl der einzelnen Komponenten eines UGVs, als auch die der verschiedenen UGVs untereinander, ist essentiell für den Austausch von Daten. Hierfür müssen geeignete Protokolle entwickelt werden, um sowohl zwischen den einzelnen Software-Komponenten als auch zwischen den Robotern einen Datentransfer zu ermöglichen.

Hardware-in-the-Loop-Mobilitätssimulation mit Sensorsimulation Mit Hilfe dieses Verfahrens soll auf der Hardware mit Testdaten das komplette Szenario simuliert werden. So ist es möglich die Funktionsfähigkeit der Steuer- und Data-Mining-Algorithmen zu testen, ohne das UGV der echten Umgebung aussetzen zu müssen. Das Verfahren vereinfacht das Debuggen von Fehlverhalten und verhindert unnötige Schäden am UGV. Der Praxistest dient dann der Überprüfung aller Komponenten.

Datenaustausch zwischen Data-Mining-Agent und Mobilitäts-Agent Der Mobilitäts-Agent sammelt Umgebungsdaten, die an den Data-Mining-Agenten zur Verarbeitung weitergeleitet werden sollen. Die Gegenrichtung liefert extrahierte Informationen zurück.

Entwicklung kommunikationssensitiver Erkundungsstrategien Die Absprache zwischen den Agenten während der Erkundung muss gewährleistet sein, um effizient arbeiten zu können. Dabei müssen Energieverbrauch, Datenrate und Skalierbarkeit berücksichtigt werden.

Entwicklung von spezialisierten verteilten Data-Mining-Algorithmen Die dabei zu lösende Herausforderung an die Algorithmen ist die Verarbeitung von Sensor-generierten Daten auf verteilten Systemen.

Einige Begrifflichkeiten bedürfen einer kurzen Einführung, die im folgenden Abschnitt gegeben wird.

[2] `http://www.cs.tu-dortmund.de/nps/de/Studium/besondere_Lehrveranstaltungen/Projektgruppen/Termine_2012SS/PG-Infoheft-2012-SS.pdf`

1.1.3 Essenzielle Komponenten

Die verwendeten essentiellen Komponenten des Projektes sollen an dieser Stelle kurz umrissen werden. Eine erschöpfende Definition ist hier nicht möglich, sofern nähere Erläuterungen nötig sind, erfolgen diese im Verlauf des Berichts.

Data-Mining Im Allgemeinen wird mit Data-Mining die Auswertung von großen Datenmengen und das Erkennen von Zusammenhängen und Mustern in den Daten bezeichnet. Für die Projektgruppe ist dies von Interesse, da aus erhobenen Messwerten Informationen über die Umgebung gewonnen und sinnvoll aufbereitet werden sollen.

Stream Ein Stream stellt einen über die Zeit kontinuierlichen Strom von Daten dar. Es gibt besondere Anforderungen, wie die eingehenden Daten verarbeitet werden können, da theoretisch ein Stream unendlich lang sein kann und insbesondere nicht alle Daten gespeichert werden können. Als Konsequenz arbeiten spezielle Stream-Algorithmen meist entweder nur auf einem Ausschnitt der Daten oder liefern ein komprimiertes, approximatives Ergebnis. Die Verarbeitung von Sensorwerten ist ein klassisches Einsatzszenario für Stream-Algorithmen.

Agent Unter einem Agenten versteht man eine selbstständige Entität, welche mit ihrer Umwelt interagiert und versucht, ein gegebenes Ziel zu erreichen. In diesem Bericht wird der Begriff Agent synonym für zwei Dinge verwendet. Zum einen ist der Agent in der Sprache der Teamstrategien und der Kommunikation ein Repräsentant des Schwarms und wird gleichbedeutend mit Fahrzeug oder Roboter verwendet. Außerdem steht Agent für mehrere unabhängige Programme, welche jedoch auf einer gemeinsamen Hardware arbeiten und miteinander in Verbindung stehen. In diesem Sinne gibt es zum Beispiel einen Mobilitäts-Agenten und einen Data-Mining-Agenten. Welche Form von Agent gemeint ist, wird durch den jeweiligen Kontext ersichtlich.

All diese Komponenten zusammen bilden die Basis, die es der Projektgruppe ermöglichen soll, das oben definierte Szenario umzusetzen.

1.2 Aufbau des Dokuments

Der Bericht gliedert sich in folgende Kapitel: Im Kapitel 2 werden die notwendigen formalen Rahmenbedingungen, die verwendeten Tools und Programmiersprachen, welche zu Beginn festgelegt wurden, erläutert. Daraufhin wird zunächst in Kapitel 3 wird die allgemeine Struktur des in der Projektgruppe entwickelten Systems vorgestellt.

Daraufhin werden die einzelnen Komponenten im Detail beschrieben: Kapitel 4 widmet sich detailliert der Hardware des verwendeten Rovers, in Kapitel 5 werden die verwendeten Data-Mining-Algorithmen motiviert und vorgestellt, Kapitel 6 beschreibt Grundlagen zu Teamstrategien und den Algorithmus welcher zur Erkundung eines Gebietes entwickelt wurde. In Kapitel 7 schließlich wird die Kommunikationsarchitektur erläutert.

Nach diesen Implementierungskapiteln wird in Kapitel 8 zunächst die Methodik beschrieben, welche zur Evaluation des Systems verwendet wird. Daraufhin werden die Ergebnisse dieser Evaluation dargestellt und diskutiert. Abschließend wird in 9 noch einmal kurz zusammengefasst was innerhalb der Projektgruppe geleistet wurde und es werden mögliche Ansatzpunkte für fortführende Projektgruppen angegeben.

2 Projektorganisation

Ziel dieses Kapitels ist es, die Organisation des Projektes kurz vorzustellen. Zunächst wird die Aufteilung der Teilnehmer in einzelne Untergruppen beschrieben und der grobe Plan für den Ablauf des Projektes geschildert. Daraufhin wird das Konzept des Projektmanagements nach Scrum vorgestellt und erläutert wie die, durch dieses Konzept vorgeschriebenen, organisatorischen Abläufe innerhalb der PG konkret umgesetzt werden. Abgeschlossen wird das Kapitel mit einem Abschnitt zu den verwendeten Tools und Programmiersprachen.

2.1 Teammitglieder

Betreut wird die Projektgruppe durch

- Dipl-Inf. Hendrik Blom, Lehrstuhl 8, Fakultät Informatik
- Dipl-Inf. Daniel Behnke, Lehrstuhl für Kommunikationsnetze, Fakultät für Elektrotechnik und Informationstechnik
- Dipl-Inf. Niklas Goddemeier, Lehrstuhl für Kommunikationsnetze, Fakultät für Elektrotechnik und Informationstechnik

Teilnehmer der Projektgruppe sind

- David Arnu
- Benjamin Berghoff
- Jan Czogalla
- David Gräff
- Manuel Groß
- Sibylle Hess
- André Johnigk
- Jens Möllmer
- Fabian Peternek
- Matthias Petsch
- Patrick Vorlicek

2.2 Aufbau der Projektgruppe und Projektplan

Um die in Abschnitt 1.1.1 definierten Referenzszenarios möglichst effizient zu bearbeiten, wurden die Teilnehmer der Projektgruppe in drei Untergruppen eingeteilt. Diese Teilgruppen bearbeiten die Kernbereiche Hardware, Datamining und Kommunikation bzw. Datenaustausch zwischen einzelnen Fahrzeugen und auch Bodenstationen. Im Folgenden werden die Arbeitsgruppen genauer vorgestellt. Im nächsten Unterabschnitt wird der Projektplan dargestellt, welcher die zeitliche Aufteilung der Aufgaben beinhaltet.

2.2.1 Teilgruppen

Eine strikte Aufgabentrennung ist nicht wünschenswert, denn um die Kooperation zwischen einzelnen Gruppen und die Integration der verschiedenen Systeme zu unterstützen, sollten einige Teilnehmer auch einen gruppenübergreifenden Überblick erlangen. Insbesondere die Kooperation zwischen einzelnen Gruppen sowie die Integration und Evaluation der Ergebnisse bedürfen einer weichen Aufgabentrennung.

Hardwaregruppe Hier sollen die UGVs mit Licht- und Temperatursensoren ausgestattet und ein Autopilot integriert werden. Die Daten der Sensoren sollen in einem Stream zur Verfügung stehen. Dabei soll das bereitgestellte Framework *CNI UxV Framework* (vgl. Abschnitt 6.2) zum Management von UxVs vom Lehrstuhl für Kommunikationsnetze genutzt werden.

Data-Mining-Gruppe Innerhalb dieser Teilgruppe sollen verteilte Data-Mining-Algorithmen entwickelt werden, um Gebiete mit hoher Energiedichte zu finden. Dazu wird das in Abschnitt 5.2 vorgestellte Stream-Framework genutzt, das bereits die Implementierung von Algorithmen auf Datenströmen mit mehreren Prozessen/Threads ermöglicht. Ziel ist also auch die Erweiterung der Funktionalität des Frameworks, um auch die oben erwähnten verteilten Algorithmen realisieren zu können.

Kommunikationsgruppe Diese Gruppe soll mittels einer vorhandenen Matlab-Umgebung (vgl. Abschnitt 6.3) die Software-in-the-Loop-Mobilitätssimulation realisieren. Das Hauptaugenmerk liegt dabei auf der Entwicklung von Teamstrategien für das kooperative Erkunden der Umgebung. Sobald die Teamstrategien implementiert wurden, müssen die Erkundungsalgorithmen in das Framework *CNI UxV Framework* integriert werden, um die Hardware-in-the-Loop-Mobilitätssimulation durchführen zu können.

2.2.2 Projektplan

Inhalt dieses Abschnittes ist zunächst eine detailliertere Aufstellung der Aufgaben, welche die zuvor beschriebenen Untergruppen innerhalb des ersten Semesters erledigen sollen. Daraufhin wird eine Einteilung dieser Aufgaben in Analyse- und Imple-

mentierungsphase vorgenommen, ein Plan beschrieben, wann diese fertig sein sollen und schließlich ein konkretes Gesamtziel für das erste Semester definiert. Abschließend wird diskutiert welche Teile davon im ersten Semester nicht vollendet wurden und der Plan für das zweite Semester erläutert.

Aufgaben im ersten Semester Tabelle 2.1 ordnet jeder Teilgruppe eine Liste von Aufgaben zu. Diese Aufgaben lassen sich in Unteraufgaben teilen, derartige Details liegen allerdings in der Verantwortung der Untergruppen. Die einzelnen Listen sind grob geordnet: Die obersten Aufgaben sollten zuerst erledigt werden, da sie Voraussetzung zur Bearbeitung der weiteren Aufgaben sind. Entsprechend kann eine Einteilung in Analyse- und Implementierungsphase vorgenommen werden, was in der Tabelle durch *(A)* und *(I)* markiert wird. Der Übergang zwischen diesen beiden Phasen ist jedoch fließend, so dass eine genaue Abgrenzung nicht immer möglich ist. Die Fertigstellung dieser Aufgaben ist das Ziel des ersten Semesters, wobei tendenziell eher zu viele Aufgaben verteilt wurden als zu wenige. Im zweiten Semester werden die Implementierungsaufgaben fertiggestellt. Daraufhin kann das System getestet und weiterentwickelt werden, wozu im zweiten Semester eine Evaluationsphase dient.

Zielsetzung für das erste Semester Zuvor beschriebene Aufgaben sind allerdings kein kohärentes Zwischenergebnis, weshalb folgender Meilenstein als Zielpunkt für das erste Semester definiert wird: Am Ende des ersten Semesters soll es möglich sein einen prototypischen Ablauf der gesamten Verarbeitung durchzuführen. Das heißt, dass die Sensordaten korrekt gesammelt und an die Data-Mining-Algorithmen gegeben werden. Diese verarbeiten die Daten und erstellen ein Modell der Umgebung, welches abgefragt werden kann. Dabei wird der Aspekt des Schwarms noch nicht beachtet und auch die autonome Bewegung des Fahrzeugs ist noch nicht notwendig. Dies sind Ziele für das zweite Semester.

Nicht erreichte Ziele im ersten Semester Am Ende des ersten Semesters waren einige Ziele leider noch nicht erreicht worden, da insbesondere die Integration aller Komponenten zu einem Gesamtsystem bedeutend mehr Zeit zur Problemlösung benötigte als angesetzt. Von den zuvor genannten Aufgaben fehlten insbesondere noch die Aggregation der einzelnen Modelle des Data-Mining zu einem einzigen Modell und die Implementierung der ausgewählten Steerings für die Simulation. Die Steerings waren zwar bereits in Matlab implementiert, allerdings mussten sie auch in C++ noch umgesetzt werden.

Aufgaben im zweiten Semester Die Ziele für das zweite Semester beinhalten die Fertigstellung der benötigten Implementierungen um ein Gesamtsystem zu erstellen. Das *CNI UxV Framework* sollte in der Lage sein, einen Erkundungsdurchlauf mit mehreren kommunizierenden virtuellen Fahrzeugen durchzuführen. Dazu sollten nebenläufig zu dem für die Projektgruppe entwickeltem Erkundungsverhalten Data-Mining-Agenten

Gruppe	**Aufgaben**
Hardware	(A) Sensoren evaluieren und auswählen
	(A) Einarbeitung in I^2C-Schnittstelle
	(I) Integration des Autopilotmoduls
	(I) Elektrische Anbindung der Sensoren an das Fahrzeug
	(I) Integration der Sensoren in das *CNI UxV Framework*
Data-Mining	(A) Algorithmus zur Erzeugung zufälliger Testdaten erstellen
	(A) Auswahl zu implementierender Algorithmen
	(A) Methoden entwickeln, das *Stream-Framework* Meshnetzwerkfähig zu machen
	(I) Verteilte Kommunikation im *Stream-Framework* über das Netzwerk durch Implementierung einer der ausgewählten Methoden.
	(I) Implementierung der ausgewählten Algorithmen
	(I) Aggregation der Modelle der einzelnen Algorithmen
Kommunikation	(A) Entwicklung von Erkundungsstrategien (Steerings)
	(I) Simulationsumgebung von UAVs auf Einsatz mit UGVs anpassen
	(A) Nachrichten der Interprozesskommunikation definieren
	(A) Evaluierung der entwickelten Steerings innerhalb der Simulation
	(I) Implementierung der ausgewählten Steerings

Tabelle 2.1: Auflistung der Aufgaben, die jede Gruppe zu erfüllen hat. Genauere Beschreibungen der Aufgaben finden sich in den entsprechenden Abschnitten. (A) und (I) markieren ob die Aufgabe eher der Analyse- oder der Implementierungsphase des Projektes zugeordnet werden kann.

die gesammelten Informationen verarbeiten, effizient speichern und für einen geringen Datenaustausch sorgen. Neben der eigenen Strategie sollten weitere Verhalten getestet und ausgewertet werden. Die Simulation aller Erkundungsverhalten soll zur Evaluation des erstellten Systems verwendet werden. Im Detail mussten dazu folgende Aufgaben gelöst werden:

- Definition eines Referenzszenarios
- Finden geeigneter Vergleichskriterien inklusive einer Vergleichsbasis
- Implementierung einer Methode, viele Simulationsdurchläufe automatisiert durchzuführen
- Vergleich und Bewertung der gewählten Erkundungsstrategien anhand der Vergleichskriterien

Dies sind die Hauptaufgaben im zweiten Semester der PG. Außerdem wird weiterhin versucht, auch den Roboter in die Simulation einzubinden. Verzögerungen durch Herausforderungen im Hardwarebereich wurde entschieden sich bei der Evaluation zunächst auf die Simulationsumgebung zu konzentrieren.

2.3 Projektmanagement

Jedes Projekt benötigt eine gewisse Ordnung, beispielsweise in Form von festen Abläufen, um typischen Problemen, etwa bei der Absprache zwischen Untergruppen, vorausschauend entgegenzuwirken. In der Literatur ist immer häufiger das Projektmanagement nach Scrum (Gedränge) in der Softwaretechnik als etabliertes Verfahren zu finden. Die Konzepte dieser Methode des Projektmanagements und deren Umsetzung innerhalb der Projektgruppe sind Inhalt dieses Kapitels.

2.3.1 Konzept des Projektmanagement nach Scrum

Scrum verfolgt die Ansicht, das moderne Entwicklungsprojekte nicht durchgängig planbar sind und ein inkrementelles Vorgehen zielführender ist. Ein Ansatz ist die Minimierung der Bürokratie um schneller zu (Zwischen-) Ergebnissen zu kommen, respektive auf sie reagieren zu können. Damit verkörpert diese Art des Projektmanagements die Werte der Agilen Softwareentwicklung, wobei Scrum diese auf drei Grundprinzipien reduziert:

Transparenz Um auf neue Situationen und Hindernisse schnell und angemessen reagieren zu können, ist es wichtig den aktuellen Fortschritt des Projekts täglich festzuhalten und für alle beteiligten öffentlich zugänglich zu machen.

Überprüfung Nur durch regelmäßige Beurteilung der fertiggestellten Funktionalität und der Qualität der Abläufe können Probleme frühzeitig erkannt werden.

Anpassung Es existiert keine finale Spezifikation. Manche Hindernisse lassen sich am effektivsten umgehen, wenn Änderungen der Anforderungen während des Projekts möglich sind.

Die Anforderungen werden demnach nicht initial zu Projektbeginn durch ein detailliertes Pflichtenheft festgelegt, sondern inkrementell über einen längeren Zeitraum, in sich wiederholenden Intervallen, den sogenannten Sprints. Zu jedem Sprint gehört ein entsprechendes Meeting, wobei Scrum hier Teilnehmer erwartet, die spezielle Rollen einnehmen, wobei einige Rollen unter Umständen auch von der selben Person wahrgenommen werden können.

Product Owner Verantwortlich für die Festlegung des Konzepts und der Entwicklungsstrategie ist der Product Owner. Festgehalten wird dies im Product Backlog, unter Anderem in Form von sogenannten User-Stories, die am ehesten mit den aus der Unified Modeling Language (UML) bekannten Use-Cases (Anwendungsfällen) ähneln. Der Product Owner legt außerdem die Prioritäten einzelner Eigenschaften des Ergebnisses fest und beurteilt am Ende eines jeden Sprints, ob eine geplante Funktionalität auch tatsächlich erreicht wurde.

Entwicklungsteam Für die eigentliche Entwicklung anhand der vorgegebenen Prioritäten ist das Entwicklungsteam verantwortlich. Es entscheidet dabei in den Sprint-Meetings eigenständig und freiwillig in Form eines Commitment, wieviel Funktionalität im nächsten Sprint erreicht werden soll. Dabei tritt das Team in den Meetings grundsätzlich geschlossen auf, so dass zum Beispiel Fehler nicht einer einzelnen Person zugeordnet werden. Im Weiteren gehört zu den Aufgaben des Entwicklungsteams die Übersetzung der User Stories in technische Arbeitsabschnitte (Tasks) und die Abschätzung des Arbeitsaufwands der jeweiligen Task. Dabei sollte eine Task möglich nicht mehr als einen Tag einnehmen.

ScrumMaster Der ScrumMaster trägt dafür Sorge, dass die Scrum-Regeln eingehalten werden, er ist also für das Gelingen von Scrum verantwortlich. Damit diese Rolle objektiv und unabhängig ausgeführt werden kann, sollte die damit beauftragte Person keine weitere Rolle im Team einnehmen. Dabei stellt der ScrumMaster kein Vorgesetzter dar, sondern vielmehr eine Führungskraft die im regen Kontakt mit allen Beteiligten stehen sollte, um Probleme frühzeitig zu erkennen und Gegenmaßnahmen einzuleiten, etwa durch gezielte Einzelgespräche.

Die bisher oft angesprochenen Sprint-Meetings sollten idealerweise wöchentlich stattfinden. Sie bestehen im wesentlichen aus zwei Teilen in den einmal das "Was" und zum anderen das "Wie" für den nächsten Sprint geklärt werden soll. Die Frage nach dem "Was" wird durch eine Analyse der User Stories geklärt, wobei das Entwicklungsteam mit dem Product Owner die Anforderung bespricht und die Kriterien zur Bewertung der Funktionalität festgelegt werden. Kriterien können in diesem Fall zum einen Akzeptanz- oder Performanzaspekte oder auch direkt definierte Testfälle sein.

Das Meeting wird mit einer Auswahl an zu bearbeitenden User Stories für den nächsten Sprint beendet, wobei soviele User Stories vom Scrum Master vorgeschlagen werden, bis ein Teammitglied an der fristgerechten Umsetzung Zweifel hat. Im zweiten Teil des Meetings wird der Ablauf der Umsetzung festgehalten. Diese Besprechung wird im wesentlichen eigenverantwortlich vom Entwicklungsteam organisiert, Fragestellungen zum allgemeinen Design oder Architekturen werden im Kollektiv, Detailfragen auch in kleineren Gruppen geklärt. Am Ende der Sprint-Planung sollten die zu bearbeitenden User Stories in kleine Tasks heruntergebrochen und alle Verständnisfragen geklärt sein. Jeder sollte wissen, was von ihm am Ende des Sprints als Ergebnis erwartet wird, über das sogenannte Taskboard kann die Verknüpfung der einzelnen Tasks mit den jeweiligen User Stories verdeutlicht werden.

Daily Scrum Zur weiteren Steigerung der Transparenz gibt es zu der wöchentlichen Sprint-Planung einen täglichen Informationsaustausch (Daily Scrum). Im Vergleich zu den mit bis zu zwei-drei Stunden angesetzten Sprint-Meetings sollten für die tägliche Besprechung nicht mehr als 15 Minuten aufgewendet werden. Hier geht es auch nicht darum, Probleme zu identifizieren oder gar deren Lösung zu besprechen, sondern den Status Quo zu identifizieren, so dass jeder einen Überblick über den aktuellen Fortschritt hat. Jedes Teammitglied berichtet darüber, was seit dem letzten Daily Scrum passiert ist und was aktuell erreicht werden soll. Sind bereits Hindernisse bekannt, können diese auch angeführt werden, jedoch ohne eine Lösungsdiskussion anzustoßen. Ein frühzeitiger Hinweis auf zu erwartende Verzögerungen ermöglichen die Aufspaltung eines Tasks, so dass die Anforderungen an den jeweiligen Tag erfüllt werden können.

Sprint Review Am Ende eines jeden Sprints sollte ein Sprint Review angesetzt werden. In diesem Meeting werden die Ergebnisse durch das Entwicklungsteam präsentiert, während der Product Owner eine möglichst kompromisslose Überprüfung der Funktionalität (Soll - Ist?) durchführt. Wichtig dabei ist, dass auch "fast fertig" eine Verfehlung der Ziele darstellt, nicht erledigte User Stories kehren zurück ins Product Backlog und werden vom Product Owner neu priorisiert. Anregungen oder ganz neue Ideen, die sich während des vergangenen Sprints ergeben haben, werden vom Scrum-Master festgehalten und dem Product Owner als mögliche neue Einträge ins Product Backlog vorgelegt.

Retrospektive Als Bindeglied zwischen dem Review und der neuen Sprint Planung gibt es die Retrospektive. Hier sollen gemachte Erfahrungen reflektiert und Verbesserungsmöglichkeiten identifiziert werden, ohne dabei Kritik an einzelnen Personen zu üben oder gar Schuldzuweisungen zu betreiben. Ziel ist es, aus Fehlern zu lernen, weshalb dieses Meeting einer besonders ordentlichen Vorbereitung bedarf, um zu oberflächlichen Beobachtungen vorzubeugen.

In den Sprints selbst arbeitet das Entwicklungsteam anhand der Taskboards, dessen Tasks ihren Prioritäten entsprechend abgearbeitet werden sollen. Idealerweise arbeiten alle gemeinsam an einer User Story um einen größtmöglichen Austausch zu ge-

währleisten. Besonders wichtig ist, dass das Team eigenverantwortlich am Erreichen des Sprint-Ziels arbeitet. Zusätzliche, von außen angestoßene Aufgaben, werden nur vom ScrumMaster entgegengenommen und versucht im Sprint mit unterzubringen. Der SrcumMaster ist auch ansonsten dafür verantwortlich Hindernisse im Ablauf zu beseitigen. Sprints dürfen für das Gelingen dieser Form des Projektmanagements nie verlängert werden und sollten immer gleich lang sein, idealerweise eine Woche, aber nie länger als vier Wochen. Sollte es einmal dazu kommen, dass ein Sprintziel offensichtlich nicht mehr erreicht werden kann, besteht die Möglichkeit einen Sprint abzubrechen und ohne Review direkt eine Retrospektive durchzuführen, um die Ursache des Abbruchs zu identifizieren und vorbeugende Maßnahmen für den nächsten Sprint einzuleiten.

2.3.2 Konkrete Umsetzung

Scrum ist in erster Linie für wirtschaftlich orientierte Projekte entworfen worden, weshalb eine eindeutige Abbildung des Konzepts auf die Projektgruppe nur begrenzt sinnvoll möglich war. Als wesentliche Komponente von Scrum wurden die wöchentlichen Sprint-Meetings identifiziert, welche auch weitestgehend im Sinne von Scrum abgehalten wurden. Jede Woche traf sich die Gruppe um die Ergebnisse der einzelnen Teilgruppen vorzustellen und das weitere Vorgehen in der nächsten Woche abzustimmen. Dazu wurden jede Woche von allen Teilgruppen kurze Präsentationen erstellt, damit die restlichen Projektmitglieder über die Fortschritte in den Teilgruppen im Bilde waren. Die einzelnen Teilgruppen planten ihre Treffen in Eigenverantwortung und legten das Arbeitspensum für jede Woche selbst fest. Eine Protokollierung der wöchentlichen Treffen mit stetig wechselndem Protokollanten half bei der Nachvollziehbarkeit der Abläufe und war eine Grundlage für die Ausarbeitung dieses Abschlussberichts. Da nicht unbedingt immer alle Teilnehmer bei jedem Sprint-Meeting teilnehmen konnten, wurde vereinbart, dass Entscheidungen, welche das gesamte Projekt betrafen, nur getroffen werden konnten, wenn mindestens neun der elf Teilnehmer der Projektgruppe anwesend waren. Eine Abstimmung galt als erfolgreich, wenn die einfache Mehrheit einem gegebenen Vorschlag zustimmte. Innerhalb der Teilgruppen konnten eigene Regelungen für anstehende Abstimmungen getroffen werden.
Im zweiten Semester war es ein Ziel, noch stärker mit der Scrum-Methode zu arbeiten. Die Betreuer versuchten dabei die Rolle des Scrum-Masters einzunehmen, da von den Teilnehmern der Projektgruppe keiner diese Aufgabe übernehmen sollte bzw. wollte. Die wöchentlichen Spring-Meetings wurden weiterhin ausgeführt. Über das Subversion-Ticketsystem (SVN) wurden Zwischenziele feingranularer aufgeteilt, um Aufgaben über kleinere Zeiträume zu verteilen. Dabei bewachten die Betreuer während der wöchentlichen Treffen die Bearbeitung der Tickets und versuchten Probleme anzusprechen. Die zugeteilten Sprints konnten allerdings oft nicht eingehalten werden. Stellenweise waren die Beschreibungen der Tickets im SVN nicht hinreichend erklärend, außerdem waren sehr viele Tickets überflüssig oder veraltet. Nach einiger Zeit ist auch diese Vorgehensweise wieder in den Hintergrund geraten, sodass die Teilnehmer keine klar

definierten Aufgaben verfolgten und die Untergruppen und Teilnehmer nicht genau voneinander wussten, welche Tickets gerade bearbeitet wurden. Auch die Protokollerstellung ist nicht mehr regelmäßig durchgeführt werden. Zum Teil lag dies aber auch daran, dass keine klar erkennbaren Ergebnisse während eines Sprint-Meetings vorlagen.

2.3.3 Reflektion und Schwierigkeiten

Neben den aufgetretenen Organisationsproblemen bezüglich des Projektmanagements, die im vorherigen Abschnitt 2.3.2 beschrieben sind, traten im weiteren Verlauf der Projektgruppe noch die folgenden Herausforderungen auf.

Frameworks Zum einen ergab sich ein erhöhter Programmieraufwand, der aus dem häufigen Mergen des *CNI UxV Framework*s entstand. Da das verwendete Framework auch vom Lehrstuhl fortlaufend weiterentwickelt wird, war es notwendig, die von der PG genutzten Eigenschaften des Frameworks auf die neuen Funktionalitäten und Änderungen anzupassen. Ein weiterer Mehraufwand entstand dadurch, dass einige neue und für die PG wichtige Funktionalitäten des Frameworks nur für bestimmte Plattformen entwickelt bzw. getestet wurden. Die Funktionen mussten an die von uns verwendeten Plattformen angepasst werden, um das Framework nutzen zu können. Diese Herausforderungen hätten durch die Arbeit mit einem fertiggestellten Framework das alle benötigten Funktionen beinhaltet oder eine frühzeitige Festlegung auf eine bestimmte Version des Frameworks vermieden werden können. Eine weitere Lösung wäre auch die Entwicklung eines eigenen, komplett eigenständigen Frameworks gewesen, was natürlich einen erheblichen Mehraufwand bedeutet hätte.

Bei der Nutzung des *Stream-Framework* auf verschiedenen virtuellen Rechnern bzw. in unterschiedlichen Szenarien kam es mehrmals zu immer unterschiedlichen Fehlern. Diese resultierten aus der grundsätzlichen Programmierung der Verteilung durch den Naming-Service (vgl. Abschnitt 7.3.3). Es waren mehrere Anläufe und Tests nötig, um diese Probleme vollständig zu beseitigen. Zusätzlich ist das *Stream-Framework* ein in Arbeit befindliches externes Projekt, sodass einige Bugs und Fehler direkt aus dem Framework resultierten, die allerdings im Vergleich zu anderen Problemen recht schnell lokalisiert und beseitigt werden konnten.

Portierung Die zuvor entwickelten und getesteten Steerings aus der Matlab-Simulationsumgebung mussten zunächst aufwändig nochmal in der C++-basierten Entwicklungsumgebung neu programmiert werden, da eine automatische Portierung nicht möglich war. Die für die Kommunikation und das Auslesen der Messwerte benötigten Bibliotheken standen teilweise für die verwendete Rechenplattform nicht zur Verfügung und mussten durch Cross-Compilierung auf die Hardware angepasst werden. Um zusätzliche Funktionalitäten zu installieren und Probleme durch Versionsunterschiede, die eine Cross-Compilierung unmöglich machten, zu beheben, war es nötig, mehrmals die Version des Betriebssystems zu aktualisieren. Auch der Datenverlust durch häufig

auftretende Defekte des genutzten Flash-Speichermediums musste durch zeitaufwändige Sicherungskopien verhindert werden.

Positionsbestimmung In der Hardware traten neben der Störung des GPS-Moduls durch die Rechenkomponente auch Probleme mit der Kalibrierung des integrierten Kompasses des Autopiloten auf, die sich erst nach einer Aktualisierung der Autopilotsoftware auf eine neue Version beheben ließen. Auch diese Verzögerung hätte durch eine eigenständige und zeitaufwändige Anpassung der Autopilotsoftware vermieden werden können. Zudem hätte man bei der Auswahl der Autopilotsoftware darauf achten können, dass mehrere Entwickler an der Software beteiligt sind, so fällt der Ausfall eines einzelnen nicht so stark ins Gewicht. Zusätzlich benötigte die Bereitstellung von Parkplatz-Daten bzw. deren Umrechnung von lokalen zu GPS-Koordinaten mehrere Anläufe. Zudem führte die Umrechnungen der verschiedenen Wertebereiche von GPS zu kontinuierlichen lokalen Koordinaten und die anschließende Zuordnung zu GPS-Zellen seitens des Data-Mining-Agenten zu Unstimmigkeiten.

Kommunikation Die gemeinsam genutzten Daten wurden über Java Sockets kommuniziert und verursachten durch Falschinterpretation der verwendeten Datenstruktur zwischenzeitlich erhöhten Kommunikationsbedarf zwischen den einzelnen Teilgruppen. Die kommunizierten Datenstrukturen wurden zwar in XML-Schemata definiert, eine ausführliche kurze Erklärung zu den gewählten Begriffen hätte den Kommunikationsaufwand allerdings gesenkt. Außerdem zogen sich viele Fehler und Probleme durch die zuvor gebildeten Gruppen. Dadurch, dass die geschriebenen Programme auf gemeinsam genutzten Ressourcen arbeiten, wurde die Fehlerlokalisation erschwert, da die Zuständigkeitsfrage zunächst geklärt werden musste. Die Beseitigung eines Fehlers führte zudem oft zu anderen Fehlern, die in die Zuständigkeit einer anderen Teilgruppe fielen. Bessere bzw. häufigere Kommunikation hätte hier zu einer Zeitersparnis führen können. Eine weitere Möglichkeit zur Vermeidung der genannten Probleme wäre eine ausführlichere Dokumentation der erledigten Aufgaben gewesen.

Rückblickend hätten viele der genannten Probleme durch eine frühzeitigere Kommunikation der einzelnen Teilgruppen vermieden werden können. Schwerer zu behandeln waren die Herausforderungen, die durch Abhängigkeit mit externen Projekten entstanden. Hierbei traten Wartezeiten und zeitlichen Verzögerungen durch lange Kommunikationswege auf.

2.4 Tools und Programmiersprachen

Im folgenden Abschnitt werden zunächst die in der PG verwendeten Tools kurz erläutert, dann wird darauf eingegangen, welche Programmiersprachen verwendet werden und schließlich angegeben, welche Codekonventionen im Rahmen der Projektgruppe eingehalten werden sollen.

2.4.1 Tools

Innerhalb der PG finden eine Reihe Tools von Dritten Anwendung. Hier wird kurz beschrieben, welchen Zweck diese erfüllen und warum sie verwendet werden. Die konkret betrachteten Tools sind Trac, SVN, LaTeX, Eclipse und Maven.

Trac Trac[1] ist ein freies, webbasiertes Ticketing-System. Es wird für Softwareprojekte hauptsächlich eingesetzt, um Bugreports zu sammeln und deren Bearbeitung zu organisieren. Außer der Funktionalität als Ticketing-System bietet Trac allerdings auch ein Wiki und ist weiterhin durch Plugins erweiterbar.

Innerhalb der PG wird das Wiki zur Organisation verwendet: Die Protokolle der wöchentlichen Gruppentreffen werden im Wiki gesammelt und jede Teilgruppe hat einen Bereich im Wiki, der frei genutzt werden kann. Die Ticketfunktionalität wird genutzt, um die zu bearbeitenden Aufgaben zu sammeln und den Teilnehmern zuzuordnen. Weiterhin ist es möglich und erwünscht, zu den Tickets Kommentare zum aktuellen Stand zu verfassen.

SVN Da es in größeren Teams nicht sinnvoll ist, alle für das Projekt benötigten Dateien direkt (zum Beispiel auf externen Datenträgern) auszutauschen, muss dafür eine andere Lösung gefunden werden. SVN[2] ist ein Versionierungssystem, welches unter anderem genau dies ermöglicht: Alle Projektdaten liegen auf einem zentralen Server. Immer wenn eine Änderung durchgeführt wird, wird diese vom Teammitglied, welches diese Änderung durchgeführt hat, auf den Server geladen (ein Vorgang, welcher als *commit* bezeichnet wird). Alle anderen Teammitglieder können dann updaten und bekommen diese neue Version der Datei. SVN speichert außerdem alle Änderungen, so dass Änderungen, welche sich im Nachhinein als Fehler erweisen auch einfach wieder rückgängig gemacht werden können. Es ist relativ offensichtlich, dass ein solches System insbesondere in der Softwareentwicklung ausgesprochen nützlich ist, weshalb es in der Projektgruppe auch für alle regelmäßig geänderten anfallenden Daten verwendet wird.

LaTeX Bei LaTeX[3] handelt es sich um eine Sammlung von Makros, die den Umgang mit dem Textsatzsystem TeX vereinfachen. Die Verwendung von LaTeX ist insbesondere dann üblich, wenn die zu setzenden Texte mathematische Formeln enthalten, da der Textsatz für diese besonders gut ist. Im Gegensatz zu üblichen Office-Produkten arbeitet LaTeX nicht nach dem „What you see is what you get"-Prinzip. Stattdessen ist es notwendig, den Text in einer Markup-Sprache zu verfassen und schließlich zu kompilieren. Dieses Vorgehen hat einige Vorteile, bspw. ermöglicht es, besonders ausgefeilte Algorithmen zur Silbentrennung zu verwenden, welche bei TeX nicht nur auf der Ba-

[1] `http://trac.edgewall.org/`
[2] `http://subversion.tigris.org/`
[3] `http://www.latex-project.org/`

sis einer Zeile sondern eines ganzen Absatzes operieren, was meist zu einem deutlich ansprechenderen Blocksatz führt.

In der PG wird LaTeX für die Dokumentation verwendet, da diese unter anderem einige Algorithmen und Formeln enthält, welche sich mit LaTeX gut darstellen lassen. Außerdem eignet es sich gut zum verteilten Arbeiten, da sich das Dokument sehr einfach auf mehrere Dateien aufteilen lässt und die Dateien als Klartext-Dateien vorliegen, welche von SVN besser unterstützt werden als Binärdaten.

Eclipse Jedes Softwareprojekt nennenswerter Größe ist auf viele verschiedene Quellcodedateien und verschiedene notwendige andere Daten (Grafiken, Datenbanken,...) aufgeteilt. Es ist theoretisch zwar möglich, ein solches Projekt nur mit einem Dateimanager und einem einfachen Texteditor wie *Notepad* zu erstellen, aber nicht praktikabel. Stattdessen ist es sinnvoll, eine IDE zu verwenden, welche neben einem Quellcodeeditor üblicherweise auch Werkzeuge zur Projektverwaltung und zum Debugging enthält.

Zumindest für den Teil des Projektes welcher in Java geschrieben wird (siehe auch Abschnitt 2.4.2) wurde die Verwendung der IDE *Eclipse*[4] festgelegt. Eclipse ist eine von der Eclipse-Foundation entwickelte IDE und war ursprünglich auf Java-Entwicklung ausgelegt. Durch die Erweiterbarkeit mittels Plugins eignet sich Eclipse nun allerdings für eine Vielzahl von Sprachen. Eclipse selbst basiert dabei ebenfalls auf Java und ist daher weitgehend plattformunabhängig einsetzbar. Für die PG eignet sich Eclipse unter anderem deshalb sehr gut, weil es Plugins für SVN und Maven gibt, zwei weitere im Rahmen der Projektgruppe verwendete Tools.

Maven Apache Maven[5] ist ein *Build-Management-Tool* zur Verwendung mit Java. Als solches ist es das Ziel, den Entwickler bei allen Aufgaben, welche bei der Softwareentwicklung anfallen, zu unterstützen. Der gesamte Prozess – vom Anlegen eines Softwareprojekts, über das Kompilieren, das Testen, das Packen bis zum Verteilen – geht weitgehend automatisch, wenn die von Maven festgelegten Konventionen eingehalten werden. So ist es beispielsweise nicht notwendig, sich um die Installation und den Import von Bibliotheken selbst zu kümmern, stattdessen kann Maven dies automatisch durchführen, inklusive eventuell auftretender Abhängigkeiten.

Maven wird bereits für das *Stream-Framework* verwendet, welches als Basis des Data-Mining-Teils dieses Projektes dient (Für Details zum *Stream-Framework* siehe Abschnitt 5.2). Es ist daher nur sinnvoll, auch in der PG weiterhin Maven für den in Java implementierten Teil zu verwenden.

2.4.2 Programmiersprachen

Während es grundsätzlich sinnvoll ist, darüber nachzudenken, welche Programmiersprache für ein Projekt verwendet werden soll, waren die Auswahlmöglichkeiten für

[4] `http://www.eclipse.org/`
[5] `http://maven.apache.org/`

dieses Projekt eher beschränkt, da alle Teilbereiche auf bereits bestehenden Frameworks aufbauen. Es ist zwar sicherlich möglich, aber meist nicht sehr sinnvoll, eine neue Programmiersprache zur Erweiterung eines bestehenden Frameworks zu verwenden, weshalb die Programmiersprachen, die auch bisher zur Implementierung dieser Frameworks verwendet wurden, in der Projektgruppe weiter verwendet werden. Dabei handelt es sich um Java für das *Stream-Framework*, welches vom Data-Mining-Teil des Projekts verwendet wird und C++ für die hardwarenahen Funktionen der UGVs. Weiterhin existiert bereits eine in Matlab implementierte Simulation von Teamstrategien von UAVs, welche für die Verwendung mit UGVs angepasst wird.

Damit blieb nur noch eine Aufgabe für die eine Programmiersprache frei gewählt werden konnte: Die Generierung von Testdaten (siehe Abschnitt 5.3.1), welche nötig ist, um die Algorithmen zu testen, bevor das Fahrzeug soweit funktioniert, dass es reale Testdaten liefern kann. Da es sich hierbei um eine weitgehend statistische Aufgabe handelt (die Testdaten werden zufällig über eine Verteilung generiert), bietet es sich an, eine Programmiersprache zu verwenden, die solche statistischen Aufgaben gut beherrscht. Die Wahl fiel daher auf die Programmiersprache *R*[6], welche explizit für statistische Berechnungen entwickelt wurde und gute Visualisierungsmethoden beinhaltet. Außerdem besteht die Möglichkeit der Interaktion zwischen *R* und Java, sodass für spätere Zwecke die Programme zur Datengenerierung weiter verwendet werden können. Ein Szenario ist zum Beispiel die automatische Generierung neuer Datensätze für die Parameteroptimierung der Algorithmen.

[6] `http://www.R-project.org/`

3 Systemarchitektur

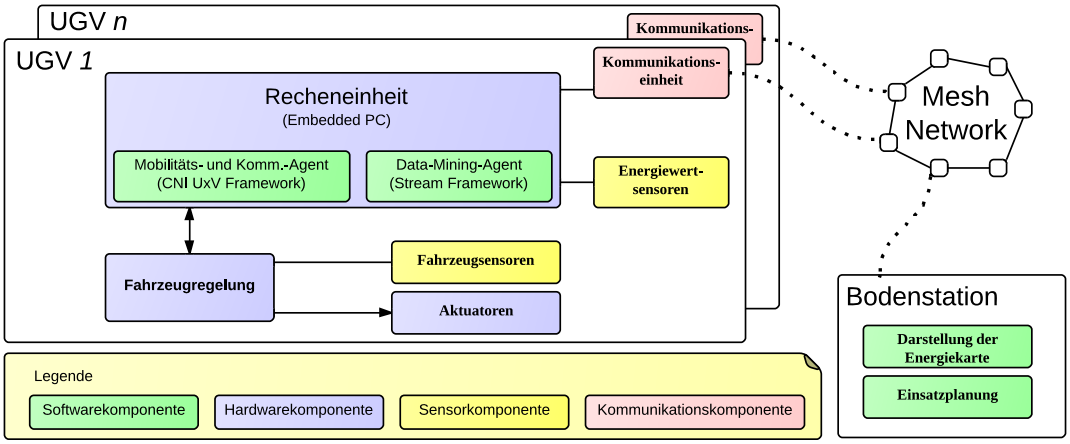


Abbildung 3.1: Systemarchitektur: Mehrere UGVs sind über ein vermaschtes Funknetzwerk miteinander und einer Bodenstation verbunden.

Folgendes Kapitel vermittelt ein Verständnis für die Zusammenhänge des entwickelten Systems. Es wird einleitend die Systemarchitektur erläutert. Anschließend wird detaillierter auf die Architektur der Experimentalplattform und nachfolgend auf die Softwarearchitektur eingegangen.

Wie in Abschnitt 1.1 zu den Aufgabengebieten der Projektgruppe festgehalten, ist das Zusammenspiel mehrerer UGVs für Energy-Harvesting mittels Solarenergie zu erforschen. Die Systemarchitektur ergibt sich aus den einzelnen UGVs, einer Bodenstation mit Auswertungsaufgaben und der Kommunikation zwischen den einzelnen UGVs sowie der Basisstation. Abbildung 3.1 veranschaulicht diesen Sachverhalt.

Kommunikation Die Fahrzeuge befinden sich über ein vermaschtes Funknetz miteinander und mit der Bodenstation in Kontakt. Ein vermaschtes Netz erlaubt nicht nur die direkte Kommunikation zwischen UGVs, sondern auch transitive Verbindungen untereinander um die Wahrscheinlichkeit für einen erfolgreichen Kommunikationsaustausch zwischen einzelnen Fahrzeugen zu erhöhen. Die genaue Funktionsweise vermaschter Netze (*Mesh-Network*) wird in Abschnitt 7.1.1 ausführlich beleuchtet.

Bodenstation Diese Software dient u. a. Auswertungszwecken, da auf einer Karte die Position, zurückgelegte Strecke und weitere detaillierte Daten der UGVs visualisiert werden. Mithilfe der Bodenstation kann manuell in den Ablauf der Algorithmen eingegriffen werden.

UGVs Jedes UGV setzt sich aus einer Recheneinheit zur Ausführung der Softwarekomponenten, einer Kommunikationseinheit und Energiewert-Sensoren zusammen. Die Recheneinheit kann über eine Fahrzeugregelung die physische Position des UGVs steuern. Die Fahrzeugregelung ist ein Echtzeitfähiges Microprozessorboard, welches die Aktuatoren ansteuert und Zugriff auf gesonderte Sensoren besitzt, die Informationen über die physikalische Lage des Fahrzeugs liefern. Die Firmware der Fahrzeugregelung beherrscht das eigenständige Anfahren von vorgegebenen Positionen durch die Übermittlung von GPS Koordinaten. Die Trennung in eine im Vergleich rechenstarke Prozessoreinheit und einen Microprozessor mit jeweils eigener Software bzw. Firmware ist aus folgendem Grund geschehen: Die Umsetzung der Erkundungsstrategien sowie der Data-Mining-Algorithmen geschieht in Programmierhochsprachen (Java/C++). Diese sind nicht auf dem Microprozessor der Fahrzeugregelung ausführbar. Andersherum sind die Ansteuerung der Aktoren und des GPS Sensors nicht ohne weiteres an der nicht-echtzeitfähigen Prozessoreinheit zu betreiben. Beide Komponenten finden daher ihren Weg in die Gesamtarchitektur. Das Kapitel zur Softwarearchitektur geht auf die Entscheidung für das *CNI UxV Framework* und das Stream Framework als Softwarebasis ein.

3.1 Hardwarearchitektur

Die Hardwarearchitektur basiert auf einer Fahrzeugregelung und einer Recheneinheit. Während die Recheneinheit die Grundlage für die Kommunikation, Auswertung der Energiewertsensoren und komplexe Berechnungen darstellt, ermöglicht die Fahrzeugregelung in Form eines Echtzeitsystems die Abstraktion der physikalischen Position und Steuerung des UGVs.

Die Recheneinheit kommuniziert über eine serielle Verbindung mit der Fahrzeugregelung um zum einen die aktuelle physische Position des UGVs zu ermitteln und zum anderen anzufahrende Wegpunkte zu übermitteln. Die Fahrzeugregelung implementiert einen Autopiloten, welcher eigenständig gesetzte Wegpunkte unter Berücksichtigung von physischen Hindernissen anfährt. Für die Positionierung nötige Aktoren (Motoren und Lenkservos), werden von der Fahrzeugregelung direkt über ein Pulsweitenmoduliertes Signal (PWM) angesteuert. Die GPS-, Kompass- und Lagesensoren werden zur Bestimmung der aktuellen Position eingesetzt und sind über eine weitere serielle Verbindung an die Fahrzeugregelung angebunden. Über den I^2C-Bus der Recheneinheit sind die Licht- und Temperatursensoren angebunden

Für die Datenübertragung zwischen den UGVs untereinander und der optionalen Bodenstation steht eine Einheit zur Funkkommunikation nach IEEE-802.11 bereit, welche über einen weiteren Bus an die Rechenkomponente angebunden ist. Die Fahrzeug-

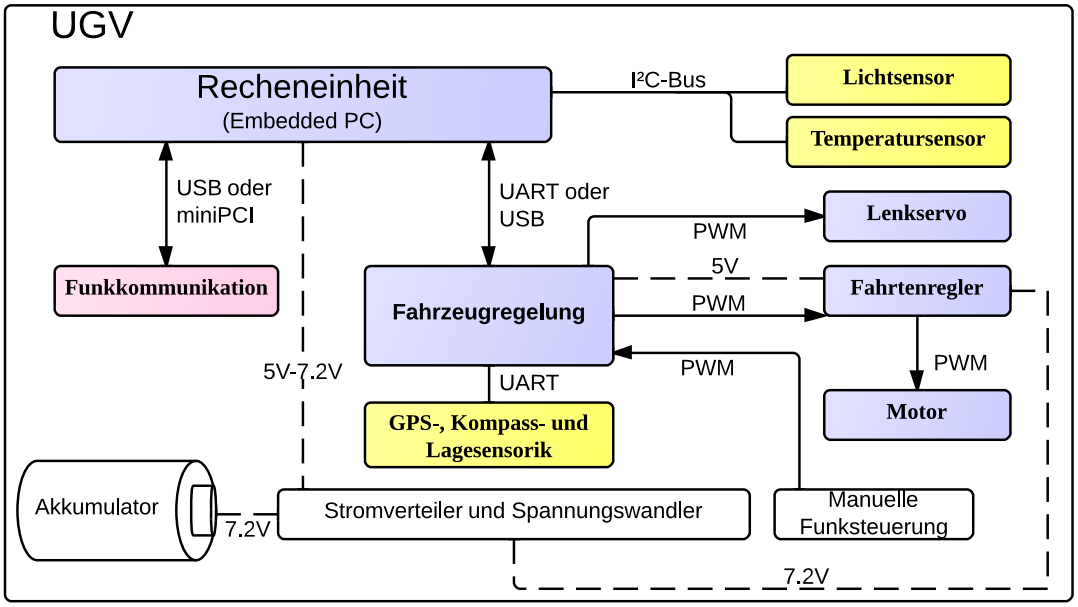


Abbildung 3.2: Hardwarearchitektur: Die Recheneinheit führt die Agentensoftware aus. Die Fahrzeugregelung kann selbstständig Koordinaten anfahren.

regelung verfügt über eine separate Funkkommunikation zur manuellen Steuerung des UGVs. Die Energieversorgung wird über eine Verteilereinheit gewährleistet, welche auch die unterschiedlichen Betriebsspannungen der jeweiligen Komponenten bereitstellt. Gespeist wird diese Verteilereinheit über einen Akkumulator.

3.2 Softwarearchitektur

Im folgenden Abschnitt wird erläutert, wie die verschiedenen verwendeten Softwarekomponenten zusammenarbeiten. Abbildung 3.3 visualisiert die einzelnen Komponenten. Grob besteht ein einzelnes UGV demnach aus zwei Agenten, welche regelmäßig Informationen austauschen: Der Data-Mining-Agent ist für die Auswertung und Verarbeitung der gesammelten Daten zuständig und gibt Empfehlungen, welche Gebiete erkundet werden sollen. Der Mobilitäts- und Kommunikationsagent kontrolliert die Fahrzeugsteuerung: Die Erkundungsstrategie wird in diesem Bereich ausgeführt und nutzt die Empfehlungen des Data-Mining-Agenten. Weiterhin werden die Sensordaten hier gesammelt und an den Data-Mining-Agenten weitergeleitet. Beide Agenten haben jeweils zwei Kommunikationsschnittstellen: Einmal untereinander (Interprozesskommunikation) zum Austausch von Informationen und einmal mit dem Mesh, welches aus den anderen UGVs besteht.

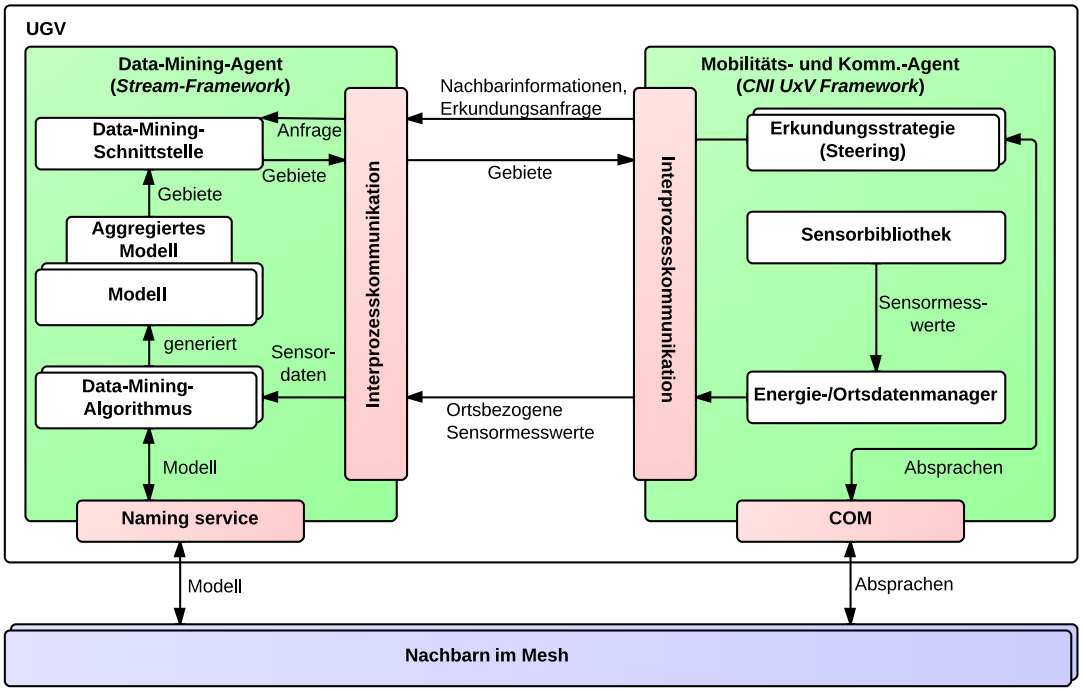


Abbildung 3.3: Die Softwarearchitektur: Data-Mining-Agent und Mobilitäts- und Kommunikationsagent, sowie deren Kommunikation mit dem Mesh

Diese Aufteilung auf Agenten ist aus dem Referenzszenario (siehe Abschnitt 1.1.1) und der vorhergehenden Vision hervorgegangen: Es wird in diesem klar definiert, dass die Data-Mining-Funktionalität des UGVs unabhängig von der derzeit ausgeführten Tätigkeit sein soll. So soll das UGV auch dann Informationen zu seiner Umgebung sammeln und mit anderen UGVs teilen, wenn dessen Aufgabe derzeit die explizite Erkundung gar nicht umfasst. Es ist daher sinnvoll, den Data-Mining-Agenten weitgehend vom Mobilitäts- und Kommunikationsagenten getrennt zu halten. Gleichzeitig ist es aber ebenfalls notwendig dem Mobilitäts- und Kommunikationsagenten zu gestatten, explizit um Gebiete zur Erkundung zu bitten, weshalb die Kommunikation zwischen den beiden Agenten benötigt wird.

Es folgen nun genauere Erläuterungen des Aufbaus der beiden Agenten (Abschnitte 3.2.1 und 3.2.2), sowie der Kommunikation, sowohl zwischen den Agenten als auch zwischen den UGVs (Abschnitt 3.2.3). Diese Beschreibungen sind zunächst recht generell und sollen in erster Linie einen Eindruck des Zusammenspiels der Komponenten vermitteln. Um Details zu den Komponenten zu erfahren, wird auf die entsprechenden Teile dieses Berichts verwiesen.

3.2.1 Architektur des Data-Mining-Agent

Zentrales Element des Data-Mining-Agenten ist das am Lehrstuhl 8 entwickelte Stream-Framework (vgl. Abschnitt 5.2). Dieses ermöglicht eine einfache und deskriptive Implementierung von Data-Mining-Algorithmen, die auf Datenströmen arbeiten. Die innerhalb des Stream-Framework implementierten Data-Mining-Algorithmen sind auch ein guter Einstiegspunkt in die Funktionsweise des Data-Mining-Agenten: Derzeit wird ein Hierarchical-Heavy-Hitters-Algorithmus verwendet, um nachvollziehen zu können welche Bereiche der Karte bereits erkundet wurden (siehe Abschnitt 5.3.4) und ein Clustering-Algorithmus, um zu speichern, wo energiereiche Gebiete gefunden wurden (siehe Abschnitt 5.3.3). Beide Algorithmen erzeugen jeweils ein eigenes Modell der Umgebung, welches die tatsächlichen Messwerte abstrahiert. Um nun Empfehlungen für die Erkundungsstrategie geben zu können, müssen diese Modelle aber zu einem einzigen Modell aggregiert werden. Auf welche Weise dies geschieht, wird detailliert in Abschnitt 5.3.5 beschrieben. Grundsätzlich gilt, dass immer dann, wenn eine Anfrage nach Gebieten vom Mobilitäts- und Kommunikationsagenten eintrifft, zunächst die Modelle aggregiert werden. Dieses aggregierte Modell wird dann nach Gebieten durchsucht, die die gestellte Anfrage erfüllen. Diese werden dann wieder an den Mobilitäts- und Kommunikationsagenten geschickt.

3.2.2 Architektur des Mobilitäts- und Kommunikationsagenten

Wie bereits der Data-Mining-Agent baut auch der Mobilitäts- und Kommunikationsagent auf einem zentralen Framework auf: dem am Lehrstuhl für Kommunikationstechnik entwickelten *CNI UxV Framework*. Dabei handelt es sich um ein umfangreiches Framework, welches Steuer- und Kontrollfunktionen auf UAVs und UGVs übernehmen kann. Details zum *CNI UxV Framework* befinden sich in Abschnitt 6.2. Für die Pro-

jektgruppe ergibt es Sinn, dieses Framework zu verwenden, da es bereits in ähnlichen Szenarien verwendet wird und die Kommunikation mit dem Autopiloten schon implementiert ist. Die Fahrzeugsteuerung wird daher mit Hilfe von Funktionen realisiert, die dieses Framework bereitstellt.

Erkundungsstrategie Die Hauptaufgabe des Mobilitäts- und Kommunikationsagenten ist nun, auf dem *CNI UxV Framework* aufbauend, die Erkundungsstrategie zu implementieren. Dazu muss entschieden werden, welches Gebiet das Fahrzeug erkundet. Die dazu entwickelte kooperative Strategie wird in Abschnitt 6.3.2 genau erläutert. Sie kommuniziert mit dem Data-Mining-Agenten, um Gebiete zu bekommen, die als nächstes erkundet werden sollen und wählt dann kooperativ mit den anderen Fahrzeugen entsprechende Gebiete aus.

Sensordaten Die zweite wichtige Aufgabe des Mobilitäts- und Kommunikationsagenten besteht darin, die Sensordaten abzufragen und an den Data-Mining-Agenten zur Verarbeitung weiterzuleiten. Die durch die Sensorbibliothek gelieferten Rohdaten werden dazu zunächst vom Energie-/Ortsdatenmanager verarbeitet, wobei beispielsweise die Rohdaten des Lichtsensors in Lux umgerechnet werden (vgl. Kapitel 4). Die so ermittelten Messwerte werden dann an den Data-Mining-Agenten weitergegeben.

3.2.3 Kommunikationsarchitektur

In den vorhergehenden Abschnitten wurde bereits mehrfach erwähnt, dass Daten zwischen den beiden Agenten ausgetauscht werden. Dies ist allerdings nicht der einzige Punkt an dem Kommunikation vorkommt: Neben dieser Interprozesskommunikation findet auch Kommunikation zwischen den einzelnen UGVs statt.

Kommunikation nach außen Um die einzelnen UGVs zu vernetzen, wird ein Mesh-Netz (Siehe Abschnitt 7.1.1) verwendet. Dies hat den Vorteil, dass keine zentrale Verwaltungsstelle für den Aufbau des Netzes benötigt wird, stattdessen übernimmt, grob gesagt, jedes UGV gleichzeitig auch die Rolle eines Routers für das Netz. Beide Agenten kommunizieren zu unterschiedlichen Zwecken mit den anderen UGVs im Mesh: Der Data-Mining-Agent tauscht die Modelle aus, um dafür zu sorgen, dass alle UGVs immer auf ungefähr dem gleichen Modell der Umgebung arbeiten. Der dazu entwickelte Naming-Service wird in Abschnitt 7.3 im Detail beschrieben.

Der Mobilitäts- und Kommunikationsagent benötigt die Kommunikation mit den anderen UGVs zur Absprache der Erkundungsstrategie. Diese ist auf kooperative Navigation ausgelegt und benötigt daher Absprachen von Erkundungsgebieten und Treffpunkten. Letztere sind notwendig, um die beschränkte Reichweite der Kommunikation auszugleichen. In diesem Sinne sorgt der Mobilitäts- und Kommunikationsagent also auch dafür, dass das Mesh für weitere Absprachen wiederhergestellt wird.

Interprozesskommunikation Abschließend gibt es noch die Kommunikation zwischen dem Data-Mining-Agenten und dem Mobilitäts- und Kommunikationsagenten. Um diese einfach zu halten, wird über Unix-Sockets kommuniziert, wobei im Wesentlichen XML-Dateien ausgetauscht werden, welche die gewünschten Informationen enthalten. Dies betrifft sowohl die Sensormesswerte, welche vom Mobilitäts- und Kommunikationsagenten an den Data-Mining-Agenten geschickt werden, als auch Erkundungsanfragen des Mobilitäts- und Kommunikationsagenten an den Data-Mining-Agenten und die Gebiete, welche als Antwort zurück gehen. Weiterhin schickt der Mobilitäts- und Kommunikationsagent Informationen über die benachbarten UGVs an den Data-Mining-Agenten. Dabei handelt es sich lediglich um eine Liste der gegenwärtig über das Mesh erreichbaren UGVs, welche vom Data-Mining-Agenten benötigt wird, um seine Modelle mit den anderen UGVs auszutauschen. Dies ist notwendig, da nur der Mobilitäts- und Kommunikationsagent die Struktur des Mesh-Netzwerks kennt. Details zum Aufbau der genannten XML-Nachrichten und dem Ablauf der Interprozesskommunikation werden in Abschnitt 7.2 erläutert.

4 Experimentalplattform

Dieses Kapitel beschreibt die Hardwareplattform, auf der die entwickelten Softwarekomponenten laufen, um die Ziele aus Abschnitt 1.1 experimentell umzusetzen. Für die Bestimmung der notwendigen Energiewerte und Positionsdaten werden Sensoren benötigt. Die Auswahl dieser Sensoren wird in den Abschnitten 4.1 und 4.2 thematisiert. Daraufhin wird in Abschnitt 4.3 das Fahrzeug beschrieben und im Anschluss auf den ersten fahrbereiten Prototyp eingegangen (Abschnitt 4.6). Dabei werden Herausforderungen dieser Konstruktion aufgezeigt und erläutert, welche letztendlich zur Konstruktion eines zweiten Prototyps führten, der in Abschnitt 4.7 vorgestellt wird.

4.1 Energiewertsensoren

Sensoren speziell für den mobilen Einsatz auf verteilten Systemen zeichnen sich durch eine geringe Leistungsaufnahme, kleine Bauformen und niedriges Gewicht aus. Sie sollen möglichst schnell genaue Messwerte erfassen und sich leicht in die bestehende Hardwareumgebung integrieren lassen. Um die Energiedichte durch Sonneneinstrahlung zu messen empfehlen sich Licht- und Temperatursensoren in Kombination, damit ein aussagekräftiger Energiewert ermittelt werden kann. Digitalsensoren zur Erfassung von Licht- und Temperaturmesswerten sind meist kostengünstiger, genauer, und effizienter als Analogsensoren. Auch die Messwerterfassung, Kalibrierung und der Anschluss von Digitalsensoren an eine Rechenkomponente ist einfacher zu realisieren als bei Analogsensoren. Eine auf dem Rover montierte Recheneinheit soll die Messwerterfassung durch Sensoren realisieren und durchführen. Im Folgenden werden Schnittstellen solcher Recheneinheiten näher beschrieben, um danach konkrete Sensoren für eine effiziente Anbindung auswählen zu können.

4.1.1 Schnittstellen

Recheneinheiten, welche in die Zieldomäne (geringer Energiebedarf, kleiner Formfaktor) passen, setzen hauptsächlich auf serielle Schnittstellen (RS232-Protokoll, USB) oder Bussysteme wie I^2C oder CAN zur Kommunikation mit anderen Komponenten. Viele verfügen auch direkt über A/D-Wandler (Analogwert-zu-Digitalwert-Wandler) zum Auslesen von Messwerten.

Über alle Schnittstellen wäre eine Realisierung der Messwerterfassung möglich, allerdings gibt es Unterschiede im Implementierungsaufwand, der Bauteilgröße, dem Energiebedarf und der Kosten:

A/D-Wandler Über den A/D-Wandler kann eine Fotodiode oder ein Thermowiderstand über eine Messbrücke ausgelesen werden. Neben der Kalibrierung der Bauteile müssen zusätzlich die analogen Messwerte in eine repräsentative Einheit umgewandelt werden und auch bauteilbedingte Messwertabweichungen müssen berücksichtigt werden. Die Messwertabweichung durch Eigenerwärmung ist Softwaretechnisch schwer ohne zusätzliche Hardware-Bauteile zu kompensieren.

USB Die Nutzung von USB-Sensoren senkt den Implementierungsaufwand und ermöglicht Plug & Play, ist aber mit hohen Anschaffungskosten und einem hohen Energieverbrauch durch die USB-Schnittstelle verbunden. Des Weiteren gibt es nur wenige komplette USB-Messmodule die eine genaue und schnell auf Veränderungen reagierende Messwerterfassung, wie im Anwendungsszenario erfordert, erlauben.

Serielle Schnittstelle Sensoren über die serielle Schnittstelle zu nutzen ist zwar kostengünstig und energiesparend, allerdings ist der Implementierungsaufwand bei Verwendung unterschiedlicher Sensoren sehr hoch, da es keine standardisierte Zugriffsmöglichkeit gibt. Meist bieten eingebettete Systeme auch nur eine geringe Anzahl von seriellen Schnittstellen an.

I²C-Bus Ein I²C-Bus [5] ist ein serieller, bidirektionaler Datenbus mit Transferraten von 100 kbit/s, 400 kbit/s, 1 Mbit/s oder 3.4 Mbit/s über zwei Adern. Daten werden dabei über die *Serial Data Line* (SDA) und der Takt über die *Serial Clock Line* (SCL) übertragen. Der Datentransfer ist bytebasiert und in Master/Slave-Topologie organisiert. Der Master kann Anfragen stellen und die Slaves antworten lediglich darauf. Vor jedem Datenbyte wird ein Adressbyte vorangesetzt. Das MSB[1] des Adressbyte gibt an, ob eine Lese- oder Schreiboperation (0 für Lesen, 1 für Schreiben) durchgeführt werden soll. Damit stehen 7 Bit zur Adressierung eines Teilnehmers zur Verfügung.

Die Nutzung des I²C-Bus ist einfach zu implementieren, da bei eingebetteten Systemen meist eine vorgefertigte API zur Verfügung steht. Es können mehrere Sensoren, auch des selben Typs, an den Bus angeschlossen werden, da diese durch Steuerbefehle und eingegrenzte Adressbereiche eindeutig identifiziert werden können. Durch regelmäßige Abfrage des Busses (Polling) ist Plug & Play möglich. Die Auswahl an I²C-Sensoren ist im Vergleich zu den bereits genannten Alternativen sehr groß und die Produkte sind günstig. Die Vielfalt an Sensoren und die schnelle und günstige Beschaffung sprechen für die Wahl der I²C-Schnittstelle. Die einfache Kommunikation, Adressierung und Zuverlässigkeit überzeugten soweit, sodass die Entscheidung zur Verwendung des I²C-Bus und den in den folgenden Abschnitten aufgeführten I²C-Licht- und Temperatursensoren fällt.

[1] most significant bit

4.1.2 Lichtsensor

Die Entscheidung zur Verwendung des Lichtsensors *TSL 2561* [1] fiel wegen der Anbindungsmöglichkeit über die I²C-Schnittstelle und des sehr geringen Energieverbrauchs von 0.75mA. Darüber hinaus kann der Sensor durch die größere Bauform im Vergleich zu ähnlichen Produkten wie dem *SFH 5712* einfacher montiert werden. Der Sensor *TSL 2561*, in Abbildung 4.1 zu sehen, erfasst Lichtwerte über je eine Fotodiode für sichtbares Licht und eine Fotodiode für Licht im Infrarotbereich. Damit besitzt er nicht nur die Möglichkeit sichtbares Licht, wie der vergleichbare Lichtsensor *SFH 5712*, zu erfassen, sondern auch Lichtwerte im Bereich der Wärmestrahlung, die sich besser zur Ermittlung eines repräsentativen Energiewertes eignen, können mit dem *TSL 2561* erfasst werden. Zusätzlich besteht hierdurch auch die Möglichkeit, zwischen künstlichem und Sonnenlicht zu unterscheiden. Die Werte der Fotodioden werden synchron erfasst, in je einem A/D-Wandler digitalisiert und in Registern gehalten. Eine Steckerleiste und die

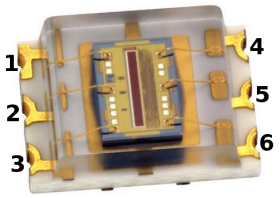


Abbildung 4.1: Der Lichtsensor TSL2561 mit folgenden Anschlüssen: Vdd (1), Addr Sel (2), Masse (3), SDA (4), INT (5) und SCL (6)

Montage auf Breakoutboards[2] vereinfacht die Anbindung an den I²C-Bus, beispielhaft in Abbildung 4.2 zu sehen. Über eine Pinleiste lässt sich mit einem Jumper manuell die Adresse, die standardmäßig 0x39 ist, verändern.

Der Sensor ist initial in einem Stromsparmodus und kann mit einem I²C-Kommando aufgeweckt werden. Die Empfindlichkeit des Lichtsensors lässt sich danach in zwei Stufen einstellen, der einfachen Stufe und der 16-fach empfindlicheren Stufe. Die Dauer der Belichtungszeit pro Abtastzeitschlitz ist wählbar zwischen 13.7 ms, 101 ms und 402 ms.

Aus einem Register des Sensors lassen sich die Teilenummer und die Revisionsnummer auslesen. Dadurch kann der Sensor über den Bus als Lichtsensor *TSL2561* eindeutig identifiziert werden, damit zwischen unterschiedlichen Sensortypen differenziert werden kann.

Die Sensorwerte werden aus schreibgeschützten Registern der beiden integrierten A/D-Wandler ausgelesen. Der gesamte Lichtwertdatenstrom ist kodiert in zwei Bytes

[2] Auf Breakoutboards werden kleine Integrierte Schaltungen angebracht, damit die Anschlüsse einfacher zu erreichen sind.

für das sichtbare und in weiteren zwei Bytes für das Infrarotspektrum. Es werden also pro Messung nacheinander vier Bytes abgefragt.

Um einen Messwert in LUX zu erhalten, werden die 32 Bit Rohdaten durch schrittweise Approximation einem LUX Wert zugeordnet. Dies geschieht in einem Programm in dem in einem Algorithmus die gemessenen Werte des Sensors im Bezug zu Referenzmessungen für LUX-Werte des Herstellers gesetzt werden [1].

4.1.3 Temperatursensor

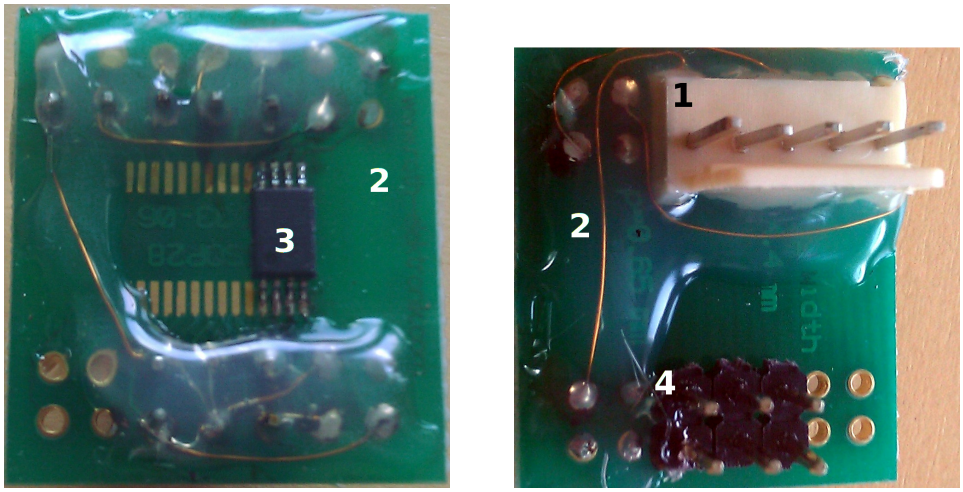
Auch der Temperatursensor *MCP9843*[2] wurde wegen der geringen Größe und Effizienz zur Ermittlung eines Energiewertes über die Umgebungstemperatur, die je nach Sonneneinstrahlung variiert, ausgewählt. Der Vorteil dieses Sensors gegenüber anderen ist die schnelle Reaktion auf Temperaturänderungen, weshalb er häufig zur Temperaturüberwachung von Bauteilen, wie Arbeitsspeicher, eingesetzt wird, deren Funktionsweise und Haltbarkeit extrem temperaturabhängig ist. Zudem verfügt er über eine I^2C-kompatible Schnittstelle, über die er an einen bestehenden I^2C-Bus angebunden wird.

Der Sensor ist mit einer Stromaufnahme von 0,2 mA sehr energiesparend und kann Temperaturen zwischen -20 °C und + 125 °C mit einer maximalen Auflösung in 0,0625 °C Schritten erfassen. Er wird vorkalibriert ausgeliefert, wobei die maximale Abweichung von der exakten Temperatur bei maximal 1 °C liegt. Diese Abweichung kann durch eine nachträgliche Interpolation auf Softwareebene behoben werden, da die Abweichung näherungsweise als konstant angenommen werden kann. Die Temperatur wird zunächst über einen analogen Bandlückensensorschaltkreis erfasst, mit einem integrierten A/D-Wandler digitalisiert und dann als Digitalwert in Registern vorgehalten.

Der Temperatursensor wurde auf ein für dessen Baugruppe TSSOP-x passendes Breakoutboard zur besseren Handhabung und Nutzung der Anschlussmöglichkeiten gelötet, siehe Abbildung 4.2. Es empfiehlt sich zusätzlich eine Stiftsteckleiste zur Plug & Play-Anbindung an den I^2C-Bus anzubringen, siehe Abbildung 4.2. Eine Pinleiste zur Änderung der Adresse des Temperatursensors mit Jumpern erhöht die Flexibilität der Anbindungsmöglichkeit. Damit können mit Jumpern Verbindungen der Adresspins *A0*, *A1* und *A2* mit *VDD*(+3,3V Versorgungsspannung) hergestellt werden.

Der Sensor lässt sich durch Auslesen des schreibgeschützten Device-ID-Registers genau identifizieren. Die Auflösung der Messdaten lässt sich wie folgt setzen, wobei sich die Umrechnungszeit näherungsweise linear zur Genauigkeit erhöht.

- 0,5 °C Genauigkeit mit typ. 30 ms Berechnungszeit
- 0,25 °C Genauigkeit mit typ. 65 ms Berechnungszeit, Standardkonfiguration
- 0,125 °C Genauigkeit mit typ. 130 ms Berechnungszeit
- 0,0625 °C Genauigkeit mit typ. 260 ms Berechnungszeit



(a) Vorderansicht (b) Rückansicht

Abbildung 4.2: Der MCP9843 Temperatursensor mit I^2C-Anschluss (1), Breakoutboard (2), Sensor (3) und Adressleiste (4)

Für eine Verwendung im definierten Zielszenario ist die voreingestellte Standardkonfiguration von 0,25 °C ein guter Kompromiss zwischen Genauigkeit und Berechnungsgeschwindigkeit.

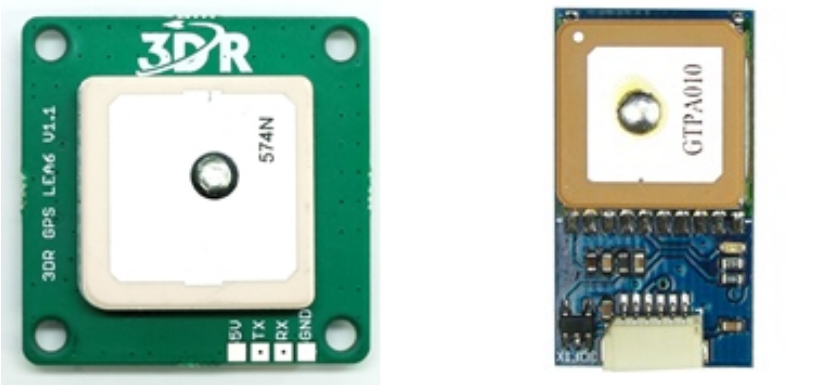
Die gemessene Temperaturwerte können aus einem schreibgeschützten Register mit einer Breite von 16 Bit ausgelesen werden. Bit 15-13 sind ungenutzt. Bit 12 gibt das Vorzeichen vor. Die Temperatur wird binär durch die Bits 11 bis 0 von 2^7 °C bis 2^{-4} °C repräsentiert.

4.2 GPS

Eine genaue Positionsbestimmung des Rovers findet über ein am Autopiloten angeschlossenes GPS Modul statt. Auf dem Markt sind verschiedene Module erhältlich, die mit unserem Autopiloten, siehe Abschnitt 4.4, betrieben werden können. Sie unterscheiden sich hauptsächlich in der Baugröße, Präzision der Positionierung und Geschwindigkeit der Positionserfassung. Tabelle 4.1 stellt zwei GPS Sensoren, das MediaTek MT3329 [4] und das 3DR GPS uBlox LEA-6 [3], im Vergleich vor. Aufgrund der genaueren Messung und schnelleren Startzeit wird der **3DR GPS uBlox LEA-6** Sensor verwendet.

	MediaTek MT3329	**3DR GPS uBlox LEA-6**
Abmessung	$16 \times 16 \times 6$ mm^3	$25 \times 25 \times 4$ mm^3
Strombedarf	53mA	40mA
Kaltstart	<38s	<30s
Warmstart	<34s	<10s durch 3V Batterie
Updaterate	10 Hz	5 Hz
Genauigkeit	<3m	<2m
Preis	ca. 30 Euro	ca. 60 Euro

Tabelle 4.1: GPS Sensor MediaTek MT3329 und 3DR uBlox LEA-6 im Vergleich.



(a) Modell 3DR GPS uBlox LEA-6 (b) Modell MediaTek MT3329 GPS

Abbildung 4.3: Die beiden betrachteten GPS-Sensoren.

4.3 Roverfahrwerk, Motorregler und Ansteuerplatine

Wichtige Merkmale des Roverfahrwerks sind die Gewinde-Alu-Öldruckstoßdämpfer, die Einzelradaufhängungen und der Kegelrad-Differenzialantrieb. Dieses erlaubt eine verlässliche Geradeausfahrt. Angetrieben wird der Rover durch einen 540er Elektromotor und ein MC 4519 Lenkservo, welche ihre Versorgung durch den 15T Carbon-Series Fahrtregler erhalten. Für den manuellen Betrieb ist auch eine 35Mhz Fernsteueranlage an den Fahrtregler angeschlossen. Die Motorsteuerung hat einen stabilen 5V Ausgang, der in dieser Konstruktion direkt mit der Autopilotplatine (Arduinoboard) verbunden ist. Bei angeschlossenem Akku wird das Arduinoboard ebenfalls mit Energie versorgt. Der Ausgang kann nicht genug Strom liefern, um auch die Rechenkomponente, Sensoren und WLAN zu treiben. Daher müssen die Einzelkomponenten auf eine andere Weise mit der Stromversorgung verbunden werden.

4.4 ArduPilot Hardware und Firmware

Das ArduPilot Kit[3] umfasst zwei Platinen, die aufeinander installiert sind. Die untere Platine basiert auf dem Atmega2560 SOC mit 8K Arbeitsspeicher, 256 kB Programmspeicher und 4k beschreibbarem EEPROM die Autopilotfirmware. Der Prozessor rechnet mit 16 MHz. Auf der Platine ist der Anschluss für einen GPS Sensor sowie 16 analoge Eingänge und 4 serielle Anschlüsse untergebracht. Auf dem Board ist ebenfalls dedizierte Hardware für Servoansteuerung und RC Fernsteueranlageneingänge vorhanden. Das zweite Board enthält Erweiterungen für den Prozessor, etwa Status LEDs, einen nativen USB Anschluss, Accelerometer, Gyrometer, I^2C-Anschlüsse, 16 MB beschreibbarer Speicher, 3,3 V Ausgänge, Temperatur- und Drucksensoren und 12 Bit genaue A/D-Wandler.

Die Autopilotsoftware *ArduPilot* ist für Fluggeräte konzipiert worden. Es werden Wegpunkte an *ArduPilot* übergeben und diese werden über GPS-Ortung angeflogen. Die Kommunikation erfolgt dabei über eine eigene Funktelemetrie. Für die Projektgruppe wird die Abwandlung *ArduRover*[4], eine relativ neue Entwicklung, für den Einsatz auf dem UGV eingesetzt. Die Arduinoplatine erhält die Wegpunkte nicht länger über eine eigene Funktelemetrie, sondern über die Recheneinheit auf dem UGV.

4.5 Die Stromversorgung

Ein 5000 mAh LiPo Akku mit einer Ausgangsspannung von 7,2 V versorgt das UGV und seine Komponenten mit Energie. Der Motorregler, die Servos und die Rechenkomponente (vgl. Abschnitte 4.6 und 4.7) werden über eine Stromverteilerplatine mit Spannung versorgt. Über einen 5V Ausgang des Motorreglers versorgt sich die Arduino-Autopilotschaltung.

4.6 UGV: Prototyp 1

Bisher sind die Einzelkomponenten für einen Fahrzeugprototyp näher betrachtet worden. Die folgenden Abschnitte beschreiben die Rechenkomponente und Laufzeitmessungen und die aufgeworfenen Herausforderungen dieses Prototypen.

4.6.1 Rechenkomponente: RoBoard

Das RoBoard RB-110 basiert auf einer Vortex86DX CPU, welche auf insgesamt 256 MB DDR2 fest verdrahteten Arbeitsspeicher zurückgreifen kann. Dem Vortex86DX Prozessor liegt eine x86-Architektur zu Grunde, die über eine sechsstufige Pipeline in Verbindung mit einem 32 KiB großen L1 Cache und einem vierfach assoziativen L2 Cache bei

[3] `https://store.diydrones.com/Full_ArduPilot_Mega_kit_from_Udrones_p/kt-apm-02.htm`, Aufgerufen am 20.09.2012

[4] `https://github.com/arktools/ardupilotone/wiki/ArduRover`, Aufgerufen am 22.09.2012

einer Größe von 256 KiB und bis zu 1 GHz Taktfrequenz für die nötige Rechenleistung sorgt. Über einen Mini-PCI-Slot können zudem eine Grafikkarte oder Erweiterungskarten für beispielsweise WLan angeschlossen werden. Das Board lässt Eingangsspannungen im Bereich von 5 bis 24 Volt zu.

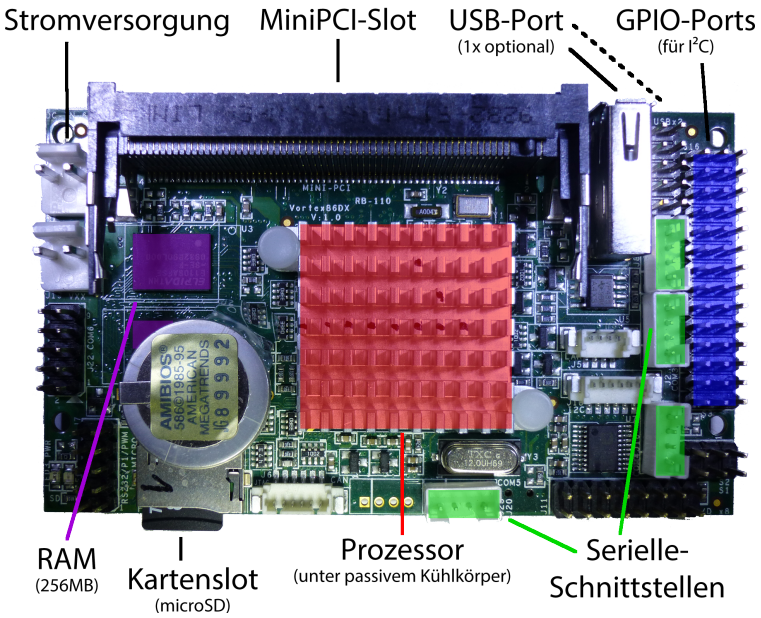


Abbildung 4.4: RoBoard RB-110: Vortex86DX CPU mit 256 MB Arbeitsspeicher, miniPCI-Slot, USB, seriellen Schnittstellen und General Purpose Input/Output PINs inklusive I^2C-Bus.

Mit in dem Gehäuse des Vortex86DX sind auch die North- und Southbridge sowie das Systembios integriert. Die Northbridge bietet über den bereitgestellten PCI-Bus neben einem Fast-Ethernet MAC [5] für verkabelte Netzwerkverbindung auch mehrere USB 2.0-Host- sowie eine USB 2.0-Client-Schnittstelle für weitere Peripherie. Als Massenspeicher kommt ein SD-Kartenleser zum Einsatz. Über die Southbridge sind mehrere GPIO- und serielle Schnittstellen genauso wie ein I^2C-Bus und Watchdog-Timer und eine parallele Schnittstelle verfügbar. Die Vielfalt an zur Verfügung stehenden Schnittstellen erfüllt somit den in Kapitel 3.1 beschriebenen Anforderungen.

Durch die x86-Architektur können auf die Speicherkarte alle vom Standardcomputer bekannten Betriebssysteme wie Windows XP oder diverse Linux-Distributionen ohne Probleme installiert werden und dafür bereitgestellte Programme ohne erneute Kompilierung ausgeführt werden.

[5] Media Access Control, abstrahiert die Sicherungsschicht des OSI-Modells

4.6.2 Datenübertragung zwischen den Komponenten

Die Datenübertragung zwischen dem Autopiloten und dem RoBoard erfolgt in einem ersten Versuch über ein selbst konfektioniertes serielles Kabel. Da jedoch das RoBoard einen ungenutzten USB Port und das Arduino über einen miniUSB Anschluss verfügt und der serielle Anschluss ggf. noch Verwendung finden soll, erfolgt die Datenübertragung im Weiteren über ein USB Kabel. Das Arduinoboard wird vom Betriebssystem des RoBoard dabei als virtueller COM-Anschluss erkannt und kann analog zu einer nativen, seriellen RS232 Schnittstelle angesprochen werden.

4.6.3 Strombedarf und geschätzte maximale Versorgungszeit

Ein wichtiger Aspekt des Einsatzszenarios ist die Laufzeit des UGV. Diese ist im wesentlichen durch den Energiebedarf der Motoren bestimmt. In Ruhephasen, etwa beim Aufladen des Akkus oder längeren Datentransfers spielt der Stromverbrauch der Rechenkomponente jedoch eine Rolle. Zur Evaluierung der benötigten Energie wurde das Roboard vom Fahrzeug getrennt und die Eingangsspannung mit zwei Multimetern, jeweils für Strom und Spannung, überwacht um die Leistungsaufnahme P nach der Formel $P = U \cdot I$ zu ermitteln.

Zunächst wurde im Leerlauf, d. h. nach abgeschlossenen Bootvorgang ein Verbrauch von $3,8$ W ermittelt. Dieser Wert ist unabhängig davon, ob das per miniPCI angebundene WLAN-Modul verbunden ist, oder nicht. Wird eine unverschlüsselte WLAN-Verbindung zu einem Accesspoint aufgebaut, steigt der Energiebedarf um $0,2$ W an. Netzwerktransfer wirkt sich je nach Richtung unterschiedlich auf den Stromverbrauch aus. Es wurden jeweils 2 GB randomisierte Daten[6] über das Network File System-Protokoll (NFS) transferiert, wobei sich beim Empfangen $1,7$ W zusätzlich zum Verbrauch bei ungenutzter Verbindung ergaben.

Beim Senden hingegen stieg der Gesamtenergiebedarf des Systems auf $6,7$ W, also um $2,7$ W im Vergleich zum Leerlauf der Verbindung an. Beim gegebenen Anwendungsszenario werden aber in unregelmäßigen Abständen kleinere Datenmengen transferiert. Gemittelt über mehrere Testläufe konnte ein Verbrauch von $5,6$ W ermittelt werden, wobei hier die CPU und RAM Auslastung berücksichtigt werden müssen. Der reine Stromverbrauch der Berechnungen und Speicherzugriffe lies sich ohne Kommunikation nicht ermitteln, so dass für diesen Test eine verkabelte Verbindung genutzt wurde, die sich nur sehr gering auf den Gesamtbedarf auswirkte. Abschließend kann festgehalten werden, dass die Kommunikation ungefähr 80% und die eigentliche Rechenlast 20% des benötigten Stroms der Rechenkomponente ausmacht.

Temperatur Bei einer Umgebungstemperatur von 20 °C wurden im Leerlauf knappe 40 °C am Kühlkörper des Prozessors gemessen. Zur Ermittelung der Temperaturen kommt ein Laserthermometer zum Einsatz. Bei den Leistungstests stieg diese auf 45 °C

[6] mittels des Konsolenbefehlt dd if=/dev/random of=/nfs/test bs=1M count 2048

an. Unter hoher CPU und Arbeitsspeicher Auslastung[7] konnten bis zu 50 °C gemessen werden.

Laufzeit Der im UGV eingesetzte Akku liefert bei 7,2 V Versorgungsspannung einen Strom von 5 $\frac{A}{h}$. Nach der Formel $P = U \cdot I$ stehen demnach theoretisch 36 W zur Verfügung. Da die Versorgungsspannung speziell bei niedriger Kapazität einbricht und Akkus dieses Typs nicht voll entladen werden dürfen, liegt der Laufzeitberechnung verfügbare Energie von 32 W zur Grunde. Bei dem errechneten durchschnittlichen Verbrauch von 5,6 W ist also eine Laufzeit von gut 6,5 Stunden zu erwarten.

Kommunikationskosten Das WLAN-Modul konnte im Schnitt einen Datendurchsatz von 3,6 MB/s in Senderichtung und 4,0 MB/s in Empfangsrichtung erreichen. Daraus ergibt sich ein Verbrauch von 1,1 W zum Senden von 2 GB Daten, respektive 0,85 W zum Empfangen der gleichen Datenmenge. Um 1 kB zu senden sind also 52 μW, für deren Empfang 40 μW nötig.

4.6.4 Integrationsherausforderungen

Der fertige erste Prototyp des UGV kann manuell über die 35 Mhz Fernbedienung bedient werden und bereits Sensordaten sammeln. Die Ortung über GPS schlägt jedoch fehl und eine Lösung musste gefunden werden. Dazu konnten drei Kandidaten als Verursacher des Problems ausgemacht werden: Der GPS Sensor, das WLan-Modul und das RoBoard.

GPS Sensor Der Austausch des GPS Sensors durch einen anderen, leistungsfähigeren Typen brachte keinen Erfolg.

WLAN-Modul In einem weiterem Schritt ist das WLAN-Modul temporär entfernt worden. GPS sendet auf 1,575 GHz und 1,227 GHz, WLAN im 2,4 GHz Band, dennoch sollte die Funkquelle WLAN als Störquelle ausgeschlossen werden. Jedoch auch ohne das Modul ist die Ortung nicht erfolgreich möglich.

RoBoard Sehr bald fiel auf, dass im ausgeschalteten Zustand des RoBoards eine Ortung schnell und genau möglich ist. Nach einer ersten, einfachen Kupferabschirmung wurde diese Vermutung bestätigt. Das RoBoard sendet elektromagnetische Strahlung im benötigten Frequenzband aus. Dieses Verhalten ist auch mit weiteren RoBoard Platinen nachvollzogen worden.

[7] generiert mittels der Anwendung *stress -c 1 -m 1 –vm-bytes 128M*

4.6.5 Integrationsherausforderungen lösen

Die erste Kupferabschirmung kann aufgrund ihrer Größe nicht verwendet werden. Eine in der Werkstatt angefertigte, an den Rover angepasste, Lösung aus Aluminum bietet leider nicht den gewünschten Schirmungseffekt. Eine dritte Abschirmung genau auf das RoBoard zugeschnitten ebenfalls nicht. Ein weiterer Versuch mit dem Ziel, einen größeren Abstand zwischen RoBoard und GPS-Empfänger herzustellen, schlägt ebenso fehl. Aus diesen Gründen wird die Recheneinheit durch eine andere ersetzt. Das darauf basierende UGV wird Prototyp 2 genannt.

4.7 UGV: Prototyp 2

Die folgenden Abschnitte beschreiben die Rechenkomponente, Laufzeitmessungen und Herausforderungen des zweiten Prototypen.

4.7.1 Rechenkomponente: Raspberry Pi

Durch die Integrationsprobleme mit dem RoBoard RB-110 fiel die Wahl nun auf den Raspberry Pi der Raspberry Pi Foundation. Dabei handelt es sich um einen scheckkartengroßen Einplatinen-Computer, basierend auf dem BCM 2835 System-on-a-Chip (SoC) von Broadcom, 512 MB Arbeitsspeicher, einem SD-Kartenleser als Massenspeicher-Medium, sowie diversen Ein- und Ausgabeschnittstellen. Das BCM 2835 SoC beinhaltet einen mit 700 MHz getakteten ARM1176JZ-F Prozessor. Der Prozessor versteht den ARMv6-Instruktionssatz sowie die Thumb-Erweiterung zur Steigerung der Code-Dichte und ist über einen 64 Bit breiten Bus mit dem Speicher verbunden.

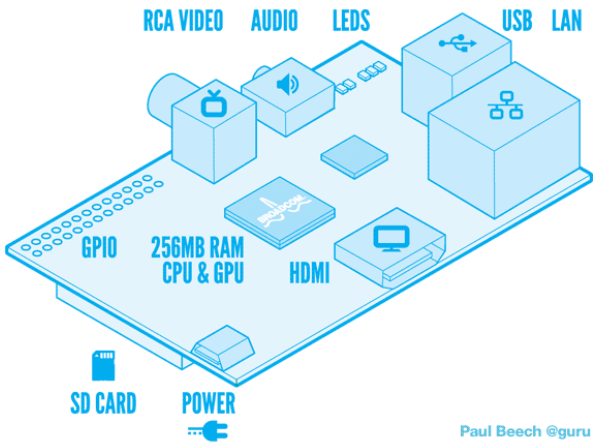


Abbildung 4.5: Raspberry Pi: Broadcom CPU/GPU System-on-a-Chip (SoC) mit 512 MB Arbeitsspeicher, USB, Ethernet, HDMI und Composite-Video, I^2C und General Purpose Input/Output (GPIO) PINs.

Mit Energie wird das Board über eine Micro-USB Buchse versorgt, welche laut Datenblatt mindestens 700 mA zur Verfügung stellen sollte. Die damit verbundene sehr geringe Thermal-Design-Power (TDP) erlaubt es, den SoC ohne Kühlung zu betreiben. Das Raspberry Pi verfügt über einen USB-Hub, der sowohl zwei physikalische USB-Ports, als auch einen 10/100 MBit Ethernet-Controller von SMSC (LAN9512) anbindet. Zudem besitzt es einen HDMI- und Composite-Video Ausgang. Für den konkreten Anwendungsfall interessant sind außerdem die 16 GPIO-Pins, welche Schnittstellen wie SPI, I^2C und UART implementieren können. In Abbildung 4.5 ist ein schematischer Aufbau des Boards exemplarisch zu sehen. Kabellose Kommunikation bietet das Board von Haus aus nicht, weshalb auf einen per USB angebundenen WLAN-Adapter zurückgegriffen wird.

Die Linux Distribution Raspbian (wheezy) hat sich inzwischen als gute Betriebssystemgrundlage etabliert. Diese enthält neben hardwarespezifischen Patches auch direkt integrierte Treiber. Außerdem stehen viele von Debian bekannte Pakete über ein separates Repository bereit.

4.7.2 Elektromagnetische Strahlung

Der Wechsel zum Raspberry Pi soll die zuvor festgestellte Problematik mit dem GPS-Empfang beheben. Es muss also zunächst geklärt werden, ob die elektromagnetische Strahlung des Raspberry Pi im Vergleich zum RB-110 für den Anwendungszweck passend ausfällt. Erste Test zeigen, dass selbst wenn das GPS-Modul direkt auf dem SoC des Raspberry Pi liegt, eine Positionsbestimmung ohne weiteres möglich ist. Es konnte zwar eine gewisse Störung beobachtet werden, diese manifestierte sich aber - im Vergleich zum kompletten Verbindungsabbruch beim RoBoard- nur in einer Ungenauigkeit der Position. Schon wenige Zentimeter Abstand sorgen für eine Verbesserung des GPS-Empfangs und der Genauigkeit der Position, bis hin zu einer stabilen Verbindung mit 7 Satelliten.

4.7.3 Strombedarf und geschätzte maximale Versorgungszeit

Für die Messungen wurde ein ähnlicher Versuchsaufbau wie beim ersten Prototyp in 4.6.3 verwendet. Im Leerlauf, ohne dass der WLAN-Stick verbunden ist, verbraucht das Raspberry Pi 2,3 W. Mit Stick liegt die Leistungsaufnahme bei 3,5 W. (0,78 W) Wird nun eine Verbindung zu einem Access-Point ohne Verschlüsselung aufgebaut, steigt der Verbrauch des gesamten Systems auf 4,1 W, der Anteil des WLAN-Moduls beträgt nun 1,22 W. Beim Übertragen von 2 GB werden 2,18 W für die WLAN-Kommunikation benötigt, wobei das System insgesamt 6,3 W aufnimmt. In Empfangsrichtung liegt der Verbrauch mit 1,33 W nur marginal über der Leerlaufleistung des Sticks. Gemittelt über mehrere Testläufe werden 4,2 W inklusive der Kommunikation benötigt.

Temperatur Der Prozessor des Raspberry Pi erreicht bei 20 °C Umgebungstemperatur unter den schon aus 4.6.3 bekannten Belastungstest maximal 43,5 °C. Hierbei ist aller-

dings anzumerken, dass der Prozessor nicht direkt vom Laserthermometer erfasst werden kann, da der RAM Bautstein im Package-on-Package Verfahren direkt auf die CPU verlötet ist. In jedem Fall ist kein zusätzlicher Kühlkörper wie beim Roboard vorhanden, weshalb die Temperaturen unter gleichen Voraussetzungen noch niedriger ausfallen würden. Der WLAN-Stick produziert natürlich ebenfalls Abwärme, seine Temperatur steigt von 28 °C bei leerlaufender Verbindung auf 34 °C beim längeren Senden, wobei die Temperatur nur am Plastikgehäuse gemessen wurde.

Laufzeit Im Vergleich zum Roboard verbraucht das Raspberry Pi basierte System mit 4,2 W fast 1,5 W weniger. Daraus ergibt sich eine gute Stunde mehr Laufzeit. Viel interssanter ist aber der sehr viel geringere Stromverbrauch im Leerlauf, da auch während des Ladevorgangs des Akkus die Rechenkomponente nicht komplett abgeschaltet werden darf und somit deren Leerlaufverbrauch den eigentlichen Ladestrom mindert. Durch diese Tatsache ist also ein potentiell schnellerer Ladevorgang des Akkus möglich.

Performanz Auch die Transferraten fallen mit dem per USB angebundenen WLAN-Modul sehr viel besser aus. Das Raspberry Pi kann die 2 GB Daten in knapp 4 Minuten senden, was einer Geschwindigkeit von 9,5 MB/s entspricht. Der Empfang ist mit fast 10 MB/s sogar noch einmal etwas schneller. Dieser starke Unterschied ist eigentlich nur auf ein Treiberproblem zurückzuführen, da beim RoBoard eigentlich sogar bessere Antennen zum Einsatz kamen und der Abstand zum Accesspoint in beiden Fällen der selbe war. In jedem Fall ergibt sich durch diese Werte eine sehr viel bessere Energieeffizienz: pro gesendetem Kilobyte Daten werden nur noch 35 μW benötigt, für jedes empfangene Kilobyte sogar nur noch 3 μW.

4.7.4 Integrationsherausforderungen

Die ARM-Architektur des Raspberry Pi macht ein Betriebssystem und eine Kompilierung der entwickelten Software für diese Architektur erforderlich. Die wichtige I^2C-Schnittstelle ist nach ein paar Anpassungen (vgl. Abschnitt 6.2.4) nutzbar, das Framework wurde portiert.

Das Raspberry Pi benötigt eine relativ stabile Spannung von 5V, während das RoBoard über einen weiten Spannungsbereich operieren kann. Da die Versorgungsspannung durch den Akku weit darüber liegt wird, ein Gleichspannungswandler nötig.

4.8 Auswertung Strombedarf und Performanz

Im direkten Vergleich schneidet das ARM basierte Raspberry Pi in allen untersuchten Punkten besser ab, als das zunächst verwendete Roboard. Abbildung 4.7 stellt die Leistungsaufnahme der beiden Prototypen gegenüber. Die sich aus dem mittlerem Energiebedarf ergebene Laufzeit der beiden Prototypen ist in Abbildung 4.8 aufgetragen.

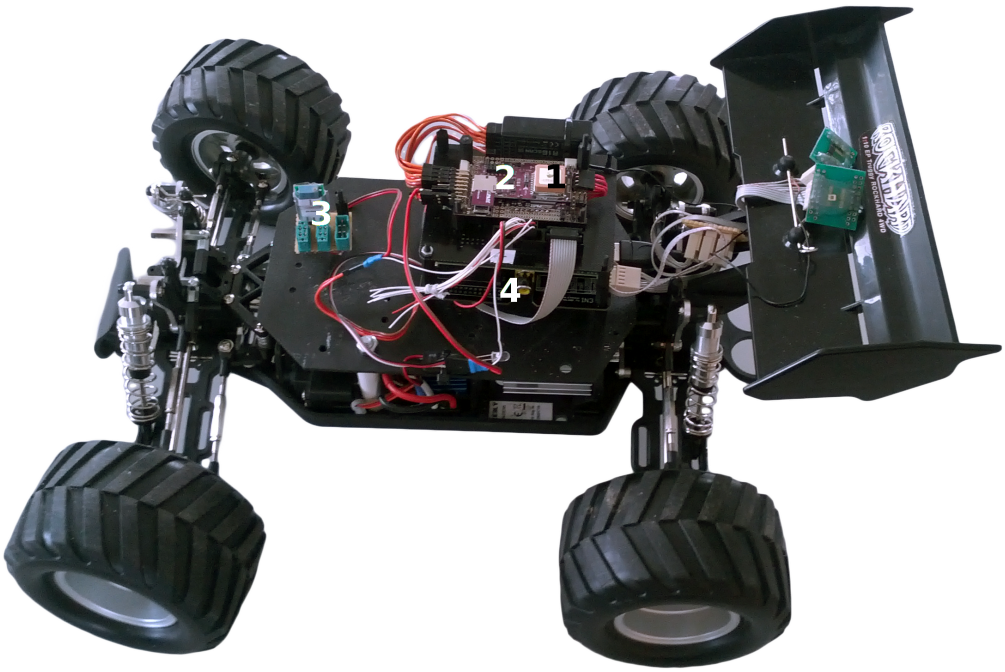


Abbildung 4.6: Raspberry Pi auf dem UGV (4) unter dem Autopilotenmodul (2) mit GPS-Sensor (1) und Gleichspannungswandler (3)

Der Energieverbrauch und der Durchsatz der WLAN-Module sind nur bedingt aussagekräftig, da das beim Raspberry Pi genutzte USB-Modul prinzipiell auch mit dem Roboard genutzt werden kann. Während der Tests konnte kein wesentlicher Unterschied beim Energiebedarf zwischen dem typischen *Infrastrukturmodus* und dem bei den Fahrzeugen zum Einsatz kommenden *Mesh-Network* aufgezeigt werden. Die Erläuterung dieser Funktechnologie erfolgt im weiteren Text.

Das Javabasierte Data-Mining benötigt verhältnismäßig mehr RAM als CPU-Leistung und in Verbindung mit dem Mobilitäts- und Kommunikationsagent wurde während mehrere Testläufe nur sehr selten eine volle Prozessorauslastung gemessen.

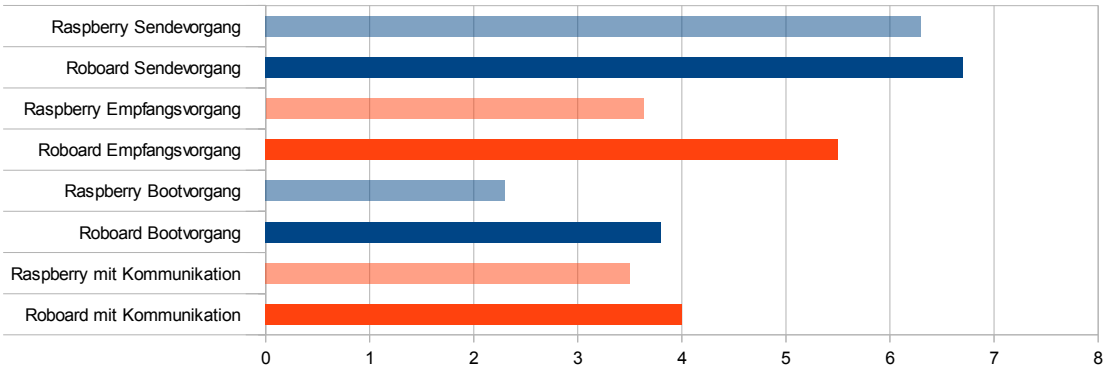


Abbildung 4.7: Gegenüberstellung der beiden Prototypen im Bezug auf die Leistungsaufnahme

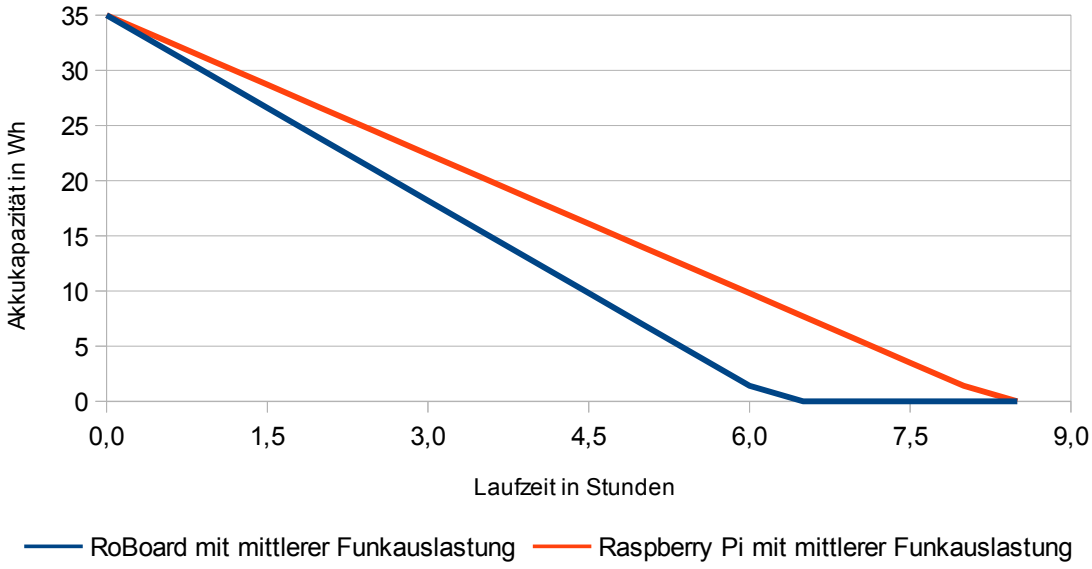


Abbildung 4.8: Die Laufzeit verbessert sich nach dem Wechsel auf eine effizientere Plattform

5 Data-Mining-Agent

Wie in Abschnitt 1.1.1 bereits erläutert bezeichnet Data-Mining generell den Prozess, aus einer gegebenen Menge von Daten mit Hilfe verschiedener Algorithmen neues Wissen abzuleiten. Diese Menge von Daten ist üblicherweise zentralisiert und in einem Stück verfügbar. Eine interessante Frage ist, was passiert, wenn diese Annahmen nicht zutreffen. Es ist beispielsweise vorstellbar, dass die verarbeiteten Daten nicht zentralisiert vorliegen, sondern auf mehrere Agenten innerhalb eines Netzes verteilt sind. In diesem Fall spricht man von „verteiltem Data-Mining“, einem vergleichsweise neuen Forschungsfeld, welches versucht, mit dieser Situation umzugehen (vgl. [16]). Dabei sollen die Daten – wie in Kapitel 1 beschrieben – nicht einfach zentralisiert werden, da die Kommunikation zu teuer ist, also zuviel Zeit und Energie benötigt. Dies macht spezialisierte Algorithmen bei denen nur die notwendigsten Informationen kommuniziert werden erforderlich.

Bei verteiltem Data-Mining sind die Daten zwar auf mehrere Agenten verteilt, allerdings kann es immer noch einen zentralen Server geben, welcher die Berechnung der verschiedenen Agenten koordiniert. Ein solcher Server existiert allerdings nicht in jedem Szenario: Ein Beispiel dafür sind sogenannte *Peer-to-Peer-Netze* (P2P-Netze). In diesen Netzen sind die Agenten, welche hier als *Peers* bezeichnet werden, dezentral vernetzt. Die Besonderheit, welche einen zentralen Server unmöglich macht, ist, dass Peers dem Netz nach Belieben beitreten und es auch jederzeit wieder verlassen können. Populäres Beispiel für P2P-Netze sind Filesharing-Netze wie *bittorrent*. Formal kann man sich ein solches P2P-Netz als ungerichteten Graphen (V, E)[1] vorstellen, welcher Operationen zum hinzufügen neuer Knoten/Kanten und zum Entfernen von Knoten hat. Die *Nachbarschaft* N_i eines Peers p_i kann dann als die Menge $\{p_j \mid \{p_i, p_j\} \in E\}$ definiert werden. P2P-Data-Mining ist ebenfalls ein vergleichsweise neues Forschungsfeld. Datta *et al.* haben in [16] folgende Paradigmen definiert, welche durch Algorithmen des P2P-Data-Mining erfüllen sollten:

Skalierbarkeit Moderne P2P-Systeme haben häufig eine sehr große Anzahl von Peers. Im hier betrachteten Fall ist das zwar noch nicht der Fall, aber die Grundidee ist dennoch, in Zukunft deutlich größere Netze zu verwenden. Daher müssen die verwendeten Algorithmen sowohl den Berechnungsaufwand als auch den Kommunikationsaufwand betreffend skalierbar sein. Idealerweise hängen beide Eigenschaften nicht oder nur in geringem Maße von der Größe des Netzes ab.

[1] In der üblichen Notation, dass V die Menge der Knoten und E die Menge der Kanten darstellt. Weiterhin werden die Kanten ungerichteter Graphen hier als zweielementige Mengen betrachtet, nicht als Tupel.

Kommunikationseffizienz Wegen des zuvor erwähnten hohen Energiebedarfs sollten Algorithmen für P2P-Data-Mining so wenig Kommunikation mit den anderen Agenten wie möglich erfordern.

Anytimeness Da Daten in P2P-Netzen durch das Hinzufügen/Entfernen von Peers veränderlich sind, müssen die Daten kontinuierlich verarbeitet werden. Es sollen keine „Schnappschüsse" der Daten zu einem Zeitpunkt erzeugt und verarbeitet werden, stattdessen sollen die Algorithmen zeitnah auf Veränderungen reagieren. Die derzeitigen aus den Daten gezogenen Schlüsse sollen entsprechend jederzeit abrufbar sein.

Asynchronität Aufgrund der Größe von P2P-Netzen kann nicht davon ausgegangen werden, dass diese in irgendeiner Form synchronisiert werden können. Algorithmen für P2P-Data-Mining sollten daher asynchron arbeiten können.

Dezentralität Eine wichtige Eigenschaft von P2P-Netzen ist, dass es keinen zentralen Server zur Koordination gibt. Die hier betrachteten Algorithmen sollten daher ebenfalls nicht von einer zentralen Kontrolle ausgehen, sondern dezentral arbeiten.

Fehlertoleranz P2P-Systeme sind sehr anfällig für Fehler, da Peers das System jederzeit verlassen/betreten können. Die verwendeten Algorithmen müssen daher ebenfalls mit ausfallenden Peers und Datenverlust umgehen können.

Insgesamt ist das P2P-Modell bereits eine recht gute Formalisierung der in Abschnitt 1.1.1 beschriebenen Situation. Wie ebenfalls dort beschrieben, liegen die Daten hier allerdings nur in Form von Streams vor und werden nicht dauerhaft gespeichert. Im Rahmen dieses Projektes wird also P2P-Data-Mining auf Streams betrieben, statt auf statischen Daten. Die im Folgenden vorgestellten Algorithmen beachten diese Besonderheit noch nicht unbedingt, sie müssen dann entsprechend angepasst werden. Weiterhin erfüllen die Algorithmen häufig nicht alle der oben genannten Paradigmen, allerdings sind auch nicht alle gleich wichtig. Für die PG am wichtigsten ist das Anytimeness-Paradigma, welches durch alle hier betrachteten Algorithmen umgesetzt wird.

Dieses Kapitel ist folgendermaßen aufgebaut: Zunächst werden im nächsten Abschnitt einige algorithmische Grundlagen eingeführt. Dies dient als Ausgangspunkt der Suche nach Algorithmen für P2P-Data-Mining auf Streams, die geeignet sind das gewünschte Szenario umzusetzen. Es werden dabei auch Algorithmen betrachtet, die in der Implementierung schließlich nicht berücksichtigt werden, da sie dennoch interessante Methoden aufzeigen. In Abschnitt 5.2 wird daraufhin das zugrundeliegende Data-Mining-Framework erläutert, welches speziell für den Umgang mit Streamdaten entwickelt wurde. Der Hauptabschnitt dieses Kapitels ist Abschnitt 5.3, in welchem erläutert wird, wie die bereits diskutierten Algorithmen konkret im Rahmen des *Stream-Framework* implementiert wurden, wie eventuelle Anpassungen auf das P2P-Data-Mining durchgeführt wurden und wie die entstehenden Modelle mehrerer verschiedener Algorithmen zu einem Gesamtmodell aggregiert werden. Abgeschlossen

wird das Kapitel durch einen Abschnitt zur Diskussion und Bewertung der Ergebnisse dieser Algorithmen.

5.1 Algorithmische Grundlagen

Im folgenden Abschnitt werden die algorithmischen Grundlagen ausgewählter Data-Mining-Algorithmen erläutert. Betrachtet werden dabei folgende Algorithmen und Probleme:

- Hierarchical-Heavy-Hitters
- Outlier Detection
- Top-l-inner Products
- Kompakte Darstellung von Entscheidungsbäumen durch Fourier-Transformation
- Verteiltes k-Means Clustering
- Distributed DBSCAN

Diese Algorithmen wurden aus der Menge von existierenden Algorithmen für verteiltes Data-Mining ausgewählt, da sie im Bezug auf das Referenzszenario erfolgversprechend sind. Welche Probleme diese Algorithmen innerhalb des in Abschnitt 1.1.1 beschriebenen Szenarios möglicherweise lösen können, wird für jeden Algorithmus in der entsprechenden Erläuterung betrachtet.

5.1.1 Hierarchical-Heavy-Hitter in Streams

In Datensammlungen ist es häufig von Interesse zu erfahren, welche Einträge oder Elemente besonders oft vorkommen, z. B. welche Artikel oft verkauft werden. Heavy-Hitter bieten hierzu eine Möglichkeit häufige Datenmengen anzuzeigen und in kompakter Form auszugeben. In hierarchischen Datenstrukturen lassen sich mit Hilfe der Heavy-Hitter-Methode häufige Mengen auf verschiedenen Abstraktionsebenen ausgeben. Weiterhin können die Merkmalsräume natürlich mehr als eine Dimension umfassen. Ein anschauliches Beispiel um diese Konzepte zu erörtern, ist die Verbindung von zwei Computersystemen mittels IP-Adressen.

Die Start- und Ziel-IP stellen jeweils ein Merkmal dar, welches eine hierarchische Abstraktion über die 8 Bit Adressbereiche ermöglicht. Das heißt zum Beispiel, dass zunächst alle lokalen Adressen von 192.168.0.0 bis 192.168.0.255 auf einer höheren Ebene zusammen gefasst werden können. Somit ergibt sich für eine einzelne IP die einfache Hierarchie $192.168.0.12 \prec 192.168.0.* \prec 192.168.*.* \prec 192.*.*.* \prec *$, wobei $*$ für ein beliebiges Element aus der Elementdomäne steht (in diesem Fall die Natürlichen Zahlen von 0 bis 255). Die höchste Abstraktionsebene wird allgemein auch mit einem einzelnen $*$ bezeichnet. Insgesamt ergibt sich nun für ein Paar von IP-Adressen eine Baum- oder Rautenstruktur, an deren Ende ein explizites Tupel von zwei IP- Adressen

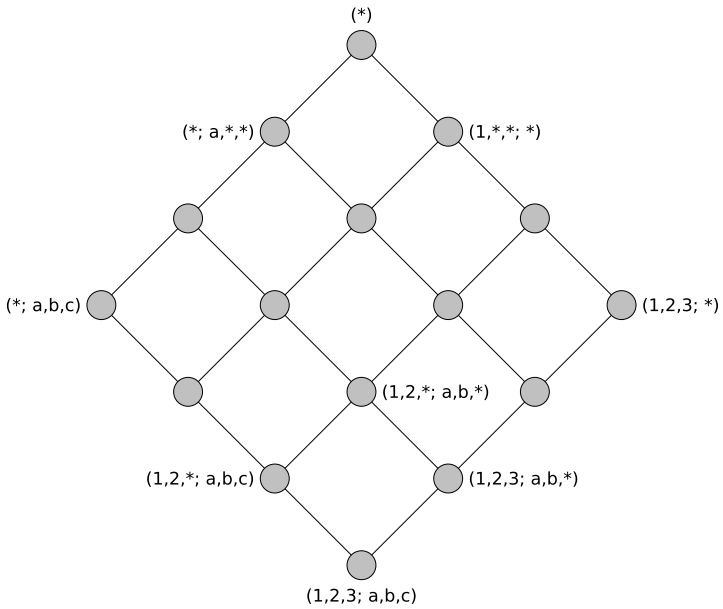


Abbildung 5.1: Beispiel einer zweidimensionalen hierarchischen Gitterstruktur.

steht und an der Spitze alle möglichen Kombinationen von Paaren vereint sind. Ein vereinfachtes Beispiel der hierarchischen Gliederung ist in Abb. 5.1 zu sehen.

Werden nun noch weitere Merkmale erhoben, beispielsweise Start- und Zielport einer Anwendung oder der Verbindungszeitpunkt, erhöht sich entsprechend die Komplexität der Datenstruktur. Für einfache Merkmale ohne hierarchische Strukturen lassen sich die Heavy-Hitter leicht bestimmen. Sie entsprechen all denjenigen Elementen e, deren absolute Häufigkeit über einem zuvor festgelegten Schwellwert liegt

$$\{e \mid \#e \geq \phi N\}$$

dabei steht N für die Anzahl aller betrachteten Elemente und der Faktor $\phi \in (0, 1]$ ist der relative Schwellwert.

Die Frage ist nun, wie diese Zählweise für hierarchische Strukturen angepasst werden kann. Eine Möglichkeit die *Hierarchical-Heavy-Hitter* zu definieren ist die folgende: Liegt die Häufigkeit eines Elementes nicht über dem Schwellwert ϕN, so wird diese Anzahl an alle direkten Nachfahren, also jene Elemente, die in der Hierarchie eine Ebene weiter oben liegen und somit allgemeiner sind, weiter gereicht. Kommt ein Element häufiger als ϕN vor, so wird dessen Anzahl nicht weiter gereicht und das Element wird als *Hierarchical-Heavy-Hitter* ausgegeben. Diese Zählweise wird in Abb. 5.2 an einem eindimensionalen Beispiel verdeutlicht.

Für mehrdimensionale Strukturen gibt es aus dieser Zählweise abgeleitet zwei Möglichkeiten die Häufigkeit eines Elementes weiter zu reichen. Bei der *split*-Regel wird die Anzahl gleichmäßig auf die Nachkommen aufgeteilt. Bei der *overlap*-Regel hinge-

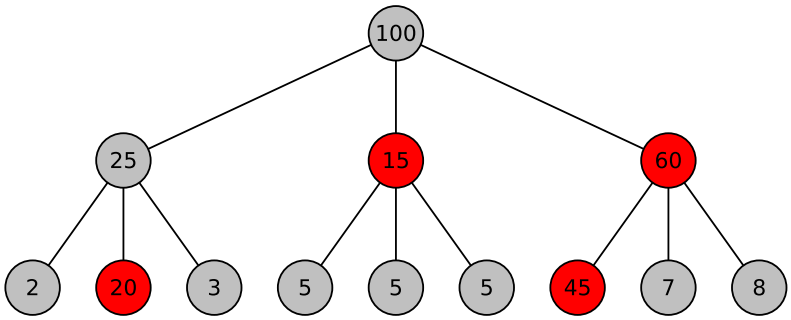


Abbildung 5.2: Hierarchical-Heavy-Hitter bei einem Schwellwert $\phi N = 15$, rot dargestellte Knoten sind die Heavy-Hitter, die Zahlen entsprechen dem Vorkommen der Elemente auf der jeweiligen Abstraktionsebene.

gen wird die Anzahl vollständig an alle Nachkommen weitergereicht, dies hat den offensichtlichen Nachteil, dass Elemente in höheren Abstraktionsebenen mehrfach gezählt werden. Im Gegenzug ist diese Zählweise parameterfrei, da nicht definiert werden muss, wie genau die Häufigkeiten aufgeteilt werden.

Für die hier betrachteten Algorithmen gilt die zusätzliche Anforderung, dass die Beobachtungen als Stream (siehe einleitende Definition 1.1.1) eingehen und verarbeitet werden. Dies führt dazu, dass das exakte Auffinden von Heavy-Hittern nicht ohne linearen Speicherplatzbedarf gelöst werden kann, da es dafür nötig wäre alle Daten explizit zu speichern. Dies ist bei sehr großen Datenmengen nicht akzeptabel. Als Konsequenz ergibt sich folgende formale Definition für die approximative Lösung des *HHH*-Problems:

HHH-Problem Gegeben sei ein Datenstrom S von N Elementen, ein Schwellwert $\phi \in (0, 1)$ und ein Fehlerparameter $\epsilon \in (0, \phi)$. Die Aufgabe des *HHH*-Problems ist es, eine Teilmenge von Präfixen $P \subset \mathbb{D}$, d. h. aus der Domäne aller möglichen Verallgemeinerungen zu finden, mit approximativen Schranken f_{min}, f_{max} für jedes Element $p \in P$, so dass folgende Eigenschaften gelten:

1. *Genauigkeit*: Es gilt $f_{min}(p) \leq f^*(p) \leq f_{max}(p)$, wobei $f^*(p)$ die wahre Häufigkeit von p ist und $f_{max}(p) - f_{min}(p) \leq \epsilon N$ ist.

2. *Abdeckung*: Für alle Präfixe $q \notin P$ gilt, dass q kein *HHH* ist, d. h. die Menge der wahren *HHH* ist eine Teilmenge von P.

Die Qualität einer Lösung lässt sich nach dieser Definition daran bewerten, wie groß die gefundene Ausgabemenge ist, denn die kleinstmögliche Menge ist die der exakten *HHH* bei vollständig bekannten Daten.

Beschreibung der Ancestry-Algorithmen Um große Datenmengen, bei gleichzeitig hoher Eingabegeschwindigkeit der einzelnen Datenelemente, wie sie in der Praxis häufig auftreten, verarbeiten zu können, bedarf es, hinsichtlich der Speichereffizienz und Verarbeitungsgeschwindigkeit, besonders effizienter Algorithmen. Als Grundlage für einen Stream-Algorithmus, der das *HHH*-Problem löst, dient zunächst folgender naiver Algorithmus:

Algorithmus 1 Naiver Hierarchical-Heavy-Hitter-Algorithmus

1. Berechne für ein eingehendes Datenelement e alle möglichen Abstraktionen p
2. Verwalte eine separate Datenstruktur für jede Abstraktionsebene und füge das passende p in jede dieser Datenstrukturen ein
3. Nach $w = \lceil \frac{1}{\epsilon} \rceil$ gelesenen Datenelementen führe einen Kompressionsschritt durch, der alle Elemente löscht, die kleiner sind als der Schwellwert ϕN

Die Verwaltung und Kompression der Elemente in den einzelnen Datenstrukturen wird durch den *Lossy-Counting*-Algorithmus von Manku und Motwani [33] realisiert, welcher als Blackbox auf den einzelnen Datenstrukturen verwendet werden kann. Offensichtlich ist obiger naiver Algorithmus nicht sehr speicherplatzeffizient, da für jede Abstraktionsebene eine eigene Datenstruktur verwaltet wird und für sehr große Hierarchien ist das mehrmalige Einfügen auch ungünstig. Der Ansatz von Cormode *et al.* [13] um diese Probleme zu beseitigen besteht darin, die Informationen über ein Element direkt an seine Nachfahren weiter zu reichen und diese in einer gemeinsamen Datenstruktur zu verwalten. Es wurden zwei Varianten entwickelt und zwar die sogenannten *Full-* und *Partial-Ancestry*-Algorithmen. Bei der *Full-Ancestry* Variante werden alle potentiellen Nachfahren eines Elementes benachrichtigt, bei *Partial-Ancestry* werden nicht mehr alle Nachfahren gespeichert. Stattdessen werden Nachfahren mit nur sehr wenigen Einträgen wieder aus der Datenstruktur entfernt. Die Verwaltung der Schranken f_{min} und f_{max} erfolgt über Zusatzinformationen, die als Tupel jedem Element hinzugefügt werden. Die tatsächliche Implementierung, insbesondere für mehrdimensionale Fälle, ist sehr technisch und würde den Rahmen dieser Kurzeinführung überschreiten. Die Experimente von Cormode *et al.* [13] zeigten jedoch, dass die Verfahren auch für sehr große Datenmengen gute und verlässliche Antworten liefern können, welche in ihrer Antwortgröße nicht stark von den exakten Algorithmen abweichen.

Für die Projektgruppe können die Heavy-Hitter von Interesse sein, da aufgrund der räumlichen Struktur der Messungen bereits eine Hierarchie der Messwerte vorgegeben ist. Außerdem ermöglicht die hierarchische Strukturierung der Sensorwerte eine unterschiedlich detaillierte Sicht auf das erkundete Gebiet.

5.1.2 Outlier Detection

Algorithmen zur Outlier Detection dienen dazu für den gegebenen Datensatz ungewöhnliche Daten – sogenannte Outlier – zu finden. Dies kann vielfältigen Zwecken dienen: Beispielsweise kann auf diese Weise versucht werden das Sensorrauschen zu mindern, Messfehler zu entdecken oder auch Kreditkartenbetrug aufzufinden. In all diesen Fällen liegen große Mengen weitgehend gleichartiger Daten vor, bei denen es darum geht diejenigen Datensätze zu finden, welche sich in einem signifikanten Maße von den anderen Daten unterscheiden. Für die Projektgruppe kann dies möglicherweise genutzt werden, indem helle oder dunkle Bereiche als ungewöhnliche Daten betrachtet werden. Dies setzt natürlich ein gewisses Vorwissen über die Umgebung voraus, da ein solches Vorgehen nur funktioniert, wenn wesentlich mehr Licht als Schatten – oder umgekehrt – existiert. Dieses Vorwissen könnte beispielsweise durch den in Abschnitt 5.1.1 beschriebenen Hierarchical-Heavy-Hitter-Algorithmus erlangt werden: Ist ein Gebiet ein Heavy-Hitter für helle Punkte, so wären die dunklen Punkte innerhalb dieses Gebietes Outlier. Ein Algorithmus zur Outlier Detection könnte dann verwendet werden die vorhandenen Informationen über das Gebiet zu konkretisieren.

Im Folgenden wird ein Algorithmus vorgestellt, welcher dieses Vorhaben in einem P2P-Netz umsetzt, also die top-n-Outlier in den Gesamtdaten findet, welche allerdings jedem Peer nur zum Teil vorliegen. Diese Beschreibung beschränkt sich allerdings auf das Wesentliche, da der Algorithmus bislang nicht in der Projektgruppe verwendet wird. Details finden sich im Paper von Branch *et al.* [9].

Voraussetzungen Vor der Beschreibung des Algorithmus gilt es allerdings zu klären, von welchem Szenario der Algorithmus genau ausgeht. Dabei handelt es sich um ein P2P-Netz mit Peers p_i. Jeder Peer hat eine Menge S_i von Datenpunkten, wobei die höchstens abzählbare Menge aller möglichen Datenpunkte als $\mathbb{D}$ bezeichnet wird. Es gilt $|S_i| = m_i \geq n$. Weiterhin kennt jeder Peer p_i seine direkten Nachbarn im Netzwerk, bezeichnet durch N_i.

Ein Algorithmus zur Outlier Detection $\mathcal{A}$ ist nun ein Algorithmus, welcher, gegeben eine endliche Menge von Punkten $P \subseteq \mathbb{D}$ und eine Rangfunktion $R : \mathbb{D} \times 2^{\mathbb{D}} \to \mathbb{R}^+$, die top-$n$-outlier ($\mathcal{A}[P]$) berechnet und zurückgibt. Die Rangfunktion dient dabei dazu die Punkte zu ordnen, und auf diese Weise die Outlier zu finden (Punkte mit höchstem Rang bezüglich der kompletten Menge P sind die Outlier). Damit dies korrekt funktioniert muss die Rangfunktion die folgenden zwei Axiome erfüllen:

Definition 5.1.1 (Axiome für Rangfunktionen)**.** *Seien $P_1 \subseteq P_2 \subseteq \mathbb{D}$ beliebig. Eine Rangfunktion R heißt* Anti-Monoton, *falls $R(x, P_1) \geq R(x, P_2)$ für alle $x \in \mathbb{D}$.*

Eine Rangfunktion R wird als Glatt *bezeichnet, falls gilt: Wenn* $R(x, P_1) > R(x, P_2)$ *für ein* $x \in \mathbb{D}$ *ist, dann gibt es auch einen Punkt* $z \in P_2 \setminus P_1$*, so dass* $R(x, P_1) > R(x, P_1 \cup \{z\})$*.*

Algorithmus Im Folgenden wird die grundsätzliche Funktionsweise des Algorithmus beschrieben: Jeder Peer p_i berechnet für sich die Outlier auf dem lokalen Datensatz S_i. Zusätzlich werden die Punkte berechnet, die – lokal – „gerade so" keine Outlier sind, also die Punkte, die, bzgl. R, am nächsten an den Outliern liegen. Diese Menge von Punkten (also alle lokalen Outlier und die zusätzlichen Punkte) wird dann an alle Nachbarn geschickt. Dies startet gewissermaßen eine neue Runde (wobei hier der Einfachheit halber missachtet wird, dass die Berechnung auf allen Peers asynchron abläuft). Die Peers verwenden diese neuen Informationen um ihr Wissen zu revidieren und möglicherweise neue Outlier zu finden. Dieser ganze Prozess wird so lange wiederholt, bis ein Fixpunkt erreicht ist: Alle Peers haben nun die gleichen Outlier gefunden und die Nachrichten, welche vielleicht noch gesendet werden, ändern diese nicht mehr. Es lässt sich nun zeigen, dass die so gefundenen Outlier der Menge $\mathcal{A}[\bigcup_i S_i]$, also der Menge der globalen Outlier auf dem gesamten Datensatz, entsprechen.

Zu Beachten ist wiederum, dass diese Beschreibung einige Details auslässt. Insbesondere wird hier vernachlässigt, dass es nicht unbedingt ausreicht nur die Punkte zu berechnen, die den Outliern am nächsten liegen, sondern manchmal auch noch einige weitere Punkte benötigt werden. Dies ist aber ebenfalls durch eine Fixpunktberechnung zu erreichen. Die genauen Details dazu, und der Beweis der Korrektheit des Algorithmus können in [9] gefunden werden.

Stream-Mining Das bislang betrachtete Szenario ging davon aus, dass alle Peers bereits einen Satz Daten haben und keine neuen hinzukommen. Im für die Projektgruppe betrachteten Szenario ist dies nicht der Fall, allerdings lässt sich der Algorithmus für den Umgang mit Streams anpassen, da die Punkte, welche von den Nachbarn gesendet wurden ohnehin bereits in gewissem Sinne neue Punkte sind. Kommt also ein neuer Punkt vom Stream, so wird dieser so behandelt, als wäre er von einem Nachbarn geschickt worden und es werden wieder die lokalen Outlier gesucht. Außerdem ist es ohne weiteres auf ähnliche Weise möglich Punkte zu „vergessen", die zu alt sind, wobei hierbei davon ausgegangen werden muss, dass die Uhren der einzelnen Peers ausreichend genau synchronisiert sind um zu verhindern, dass ein Peer einen Punkt entfernt, welchen ein anderer Peer noch als ausreichend neu ansieht.

5.1.3 Identifikation der Top-*l* Skalarprodukte

Skalarprodukte finden eine mannigfaltige Anwendung im Data-Mining. Sie werden im k-Means-Algorithmus (vgl. Abschnitt 5.1.5) zur Abstandsberechnung eingesetzt, die Entropie in den Knoten eines Entscheidungsbaums kann mit ihrer Hilfe bestimmt werden [20] und andere Methoden nutzen sie zur Ermittlung des Abstands von Punkten zu einer Hyperebene [11]. In diesem Sinne kann es für die verschiedensten Anwendungen von Interesse sein, einige der obersten Skalarprodukte bzgl. einer bestimmten Ord-

	Peer 1			
	Attribut 1	Attribut 2	...	Attribut k
Beobachtungen von Peer 1	$x_{11}^{(1)}$	$x_{12}^{(1)}$		$x_{1k}^{(1)}$
	$\vdots$		$\ddots$	
	$x_{b_1 1}^{(1)}$	$x_{b_1 2}^{(1)}$		$x_{b_1 k}^{(1)}$

	Peer N			
	Attribut 1	Attribut 2	...	Attribut k
Beobachtungen von Peer N	$x_{11}^{(N)}$	$x_{12}^{(N)}$		$x_{1k}^{(N)}$
	$\vdots$		$\ddots$	
	$x_{b_N 1}^{(N)}$	$x_{b_N 2}^{(N)}$		$x_{b_N k}^{(N)}$

Tabelle 5.1: Schematische Darstellung von horizontal verteilten Datenmengen unter N Peers.

nung, also z. B. die größten Abstände, unreinsten Knoten oder naheliegensten Punkte zur Hyperebene, bestimmen zu können. Insbesondere in P2P-Netzen, in denen die erhobenen Datenmengen i. A. sehr groß sind und Kommunikation ressourcenintensiv ist, ist eine Diskussion über die Durchführung dieses Ranking-Verfahrens notwendig, um zufriedenstellende Anwendungen implementieren zu können.

Das Skalarprodukt ist folgendermaßen definiert:

Definition 5.1.2 (Skalarprodukt)**.** *Seien $\vec{u}, \vec{v} \in \mathbb{R}^n, n \in \mathbb{N}$, und $\angle(\vec{u}, \vec{v})$ der von den Vektoren $\vec{u}$ und $\vec{v}$ eingeschlossene Winkel. Das Skalarprodukt von $\vec{u}$ und $\vec{v}$ ist definiert als*

$$\langle \vec{u}, \vec{v} \rangle = |\vec{u}| \cdot |\vec{v}| \cdot \cos \angle(\vec{u}, \vec{v}).$$

Wie man der Definition entnehmen kann, wird das Skalarprodukt umso größer, je eher die multiplizierten Vektoren in ähnliche Richtungen zeigen und je länger die Vektoren sind. Auf diese Art und Weise ist das Skalarprodukt auch als Ähnlichkeitsmaß interpretierbar, welches z. B. in Empfehlungsverfahren, die basierend auf den Vorlieben einzelner Nutzer einer Online-Community Vorschläge geben, genutzt wird [32].

In diesem Kontext wurde auch der Algorithmus von Das *et al.* [15] entwickelt, um innerhalb eines P2P-Netzes mit horizontal verteilten Daten einige der ähnlichsten Attribute, also größten Skalarprodukte bestimmen zu können. Daten heißen horizontal verteilt, wenn sich die Menge der global getätigten Beobachtungen aus den Beobachtungen der Peers zu einer global gegebenen Menge von Eigenschaften (Attributen) zusammensetzt (vgl. Tabelle 5.1). Ein Vektor über den Beobachtungen zu einem Attribut wird *Feature-Vektor* genannt. Die Spaltenvektoren aus Tabelle 5.1 $\left(x_{11}^{(p)} \dots x_{b_p 1}^{(p)}\right)^t, 1 \leq p \leq N$ entsprechen somit den Feature-Vektoren zu den jeweiligen Attributen. Die Ähnlichkeit

zwischen Attributen kann nun anhand der vereinten Beobachtungen aller Peers bzgl. dieser Attribute berechnet werden, d. h. mittels des Skalarprodukts

$$\underbrace{\left\langle \begin{pmatrix} x_{1i}^{(1)} \\ \vdots \\ x_{b_1 i}^{(1)} \\ \vdots \\ x_{1i}^{(N)} \\ \vdots \\ x_{b_N i}^{(N)} \end{pmatrix}, \begin{pmatrix} x_{1j}^{(1)} \\ \vdots \\ x_{b_1 j}^{(1)} \\ \vdots \\ x_{1j}^{(N)} \\ \vdots \\ x_{b_N j}^{(N)} \end{pmatrix} \right\rangle}_{\textit{globales Skalarprodukt}} = \sum_{p=1}^{N} \underbrace{\left\langle \begin{pmatrix} x_{1i}^{(p)} \\ \vdots \\ x_{b_p i}^{(p)} \end{pmatrix}, \begin{pmatrix} x_{1j}^{(p)} \\ \vdots \\ x_{b_p j}^{(p)} \end{pmatrix} \right\rangle}_{\textit{lokales Skalarprodukt}} \tag{5.1}$$

für Attribut i und Attribut j. Wie man Gleichung 5.1 entnehmen kann, hat das Skalarprodukt die vorteilhafte Eigenschaft, sich aus den lokalen Skalarprodukten der Peers zusammensetzen zu lassen. Dadurch wird zunächst ein naives Vorgehen ermöglicht, in dem ein Peer die globalen Skalarprodukte mithilfe der Information über alle lokalen Skalarprodukte der Peers bestimmt und die obersten auswählt. Dieses Verfahren ist natürlich nicht sehr effizient, denn um alle Peers nach ihren Skalarprodukten befragen zu können, wird ein Kettenbriefverfahren angewendet werden müssen, in welchem jeder Empfänger einer Anfrage diese an die ihm bekannten Peers weiterleitet (ausgenommen von dem sendenden Peer) und das lokale Skalarprodukt an den Fragesteller sendet. Auf diese Art und Weise werden viele unnötige Informationen verschickt, wie z. B. die lokalen Skalarprodukte, die zu den untersten Skalarprodukten gehören, die nicht von Interesse sind, oder der gleichen Anfrage über mehrere Male. Um diesen Algorithmus zu verbessern, werden des Weiteren zwei Ansätze vorgestellt: die ordinale und die kardinale Stichprobenentnahme.

Es wird nun die ordinale Stichprobenentnahme diskutiert. Um die Anzahl zu versendender Skalarprodukte zu verringern, kann der fragestellende Peer eine Wahrscheinlichkeit q wählen, aus der sich der Umfang (m) zu bestimmender Skalarprodukte errechnet, sodass mit Wahrscheinlichkeit $\geq q$ das größte der m Skalarprodukte unter den Top-p% Skalarprodukten liegt. Die Anzahl berechneter Skalarprodukte muss dazu lediglich die Ungleichung

$$m \geq \frac{\log(1-q)}{\log(1-\frac{p}{100})},$$

erfüllen. Demnach muss ein Peer, der eine Anfrage erhält, nun nicht mehr alle, sondern nur m angegebene lokale Skalarprodukte senden. Man beachte, dass die Schranke zu berechnender Skalarprodukte unabhängig von der Gesamtanzahl an Skalarprodukten ist. Bei kleiner Anzahl an Attributen wird dieses Resultat also kaum einen Vorteil bieten, bei steigender Anzahl von Attributen jedoch umso mehr.

Es wird nun die kardinale Stichprobenentnahme betrachtet, durch sie wird die Anzahl an befragten Peers verringert. Wie bereits beschrieben, lässt sich das globale Ska-

larprodukt aus den lokalen berechnen. Dementsprechend liegt die Überlegung nahe, eine Rangfolge nur mithilfe der lokalen Skalarprodukte von einer bestimmten Auswahl von Peers (n) zu erstellen, sodass das Ergebnis im Erwartungswert ähnlich ausfallen wird. Unter Benutzung der Hoeffding-Schranke [29] lässt sich zeigen, dass unter bestimmten Bedingungen zu einer gewählten Wahrscheinlichkeit $q' \in (0, 1)$ und einer Fehlerschranke $\epsilon > 0$, die Wahrscheinlichkeit, dass das approximative Skalarprodukt einen Fehler größer als ϵ aufweist, kleiner als q' ist. Zu den Bedingungen gehört, dass für die Anzahl summierter lokaler Skalarprodukte (n) folgende Ungleichung gilt:

$$n \geq \frac{(b-a)^2(\ln(2) - \ln(q'))}{2\epsilon^2},$$

wenn das errechnete Skalarprodukt Werte in $[a, b]$ annimmt. Die Abschätzung des Wertebereichs bleibt hier undiskutiert, erweist sich jedoch ggf. als schwierig. Weiterhin wird angenommen, dass die Auswahl zu summierender lokaler Skalarprodukte i.i.d.[2] gewählt wird. Dies bedeutet, dass Peers, die ihr Skalarprodukt senden sollen, von dem initiierenden Peer i.i.d. gewählt werden müssen. In einem P2P-Netz ist dies nicht ohne weiteres möglich, da nicht anzunehmen ist, dass es einen Peer gibt, der alle anderen kennt. Um dieses Problem zu lösen, wird ein spezifischer Random Walk [35] vorgeschlagen, der sich jedoch auf die Annahme stützt, dass die Nachbarschaften der Peers über die Zeit invariant sind. Da man diese Annahme für unsere mobilen Peers nicht treffen kann, bleibt dieses Problem zuerst offen. Wieder hängt die berechnete Schranke nicht von der Gesamtanzahl aller Peers ab, so dass dieses Resultat inbesondere für große P2P-Netzwerke von großem Nutzen ist.

Unter Verwendung dieser Methoden lässt sich schlußendlich zeigen, dass sich der versendete Datenumfang auf $64nm(20 \log n + 1)$ Bits insgesamt reduzieren lässt, um eins der Top-p% größten Skalarprodukte zu bestimmen. Man vergleiche dazu den versendeten Datenumfang des naiven Algorithmus, $64NM$ Bits, wobei N die Anzahl der Peers und M die Anzahl an Skalarprodukten beschreibt. Da die Werte n und m unabhängig von der Größe des Netzwerks und der Attribute sind, ist der Algorithmus insbesondere gut skalierbar.

In Bezug auf das gegebene Szenario sind nun mehrere Anwendungsfälle denkbar. Im Allgemeinen ist die Anwendung der beschriebenen Methoden sinnvoll um Gegebenheiten zu erfassen, die eine Ermittlung eines entsprechenden Modells zu umfangreich oder zu kompliziert gestalten würde. So könnte man zeitliche Abhängigkeiten, die i. A. schwer zu modellieren sind, mit diesem Verfahren erfragen, wie z. B. Licht- und Schattenverhältnisse. Das Sammeln von Informationen über die Lichtverhältnisse nach Tageszeiten ermöglicht Anfragen der Gebiete, die sich ähnlich gemäß des gegebenen Sonnenstands verhalten. Gebiete, die je nach Sonnenstand beispielsweise von dem Schatten einer Häuserwand erfasst werden, verhalten sich unabhängig von den gegebenen Wetterbedingungen ähnlich gemäß der Helligkeitswerten zu gegebenen Tageszeiten. Falls ein Peer also gegen Nachmittag in ein sehr dunkles Gebiet fährt, das aufgrund von Beobachtungen zur Mittagszeit als „hell“ deklariert wurde, ist anzunehmen,

[2] independent, identically distributed

dass dieses Gebiet von einem Schatten erfasst wurde. Andersherum kann die Messung von hoher Lichtintensität in einem als „schattig“ markiertem Gebiet die Existenz von mehreren erwartungsgemäß hellen Gebieten anzeigen, die nun nicht mehr von diesem Schatten erfasst werden. Eine Anfrage über diejenigen Gebiete, die sich ähnlich zu dem gegebenen verhalten, könnte eine Kategorisierung über viele länger nicht mehr besuchte Gebieten möglich machen. Die Anwendungen innerhalb von Online-Communities lassen sich ebenfalls miteinbeziehen. Empfehlungen von Wegen aufgrund von ähnlichen Start- und Zielvorgaben können so z. B. in einem Gelände mit Hindernissen von großem Nutzen sein.

5.1.4 Fourier-Spektrums basierte Analyse und Darstellung von Entscheidungsbäumen

In diesem Ansatz, von Hillol Kargupta und Byung-Hoon Park [31], geht es um die effiziente Nutzung von Entscheidungsbäume (DTs) in verteilten Systemen. Beim Data-Mining in Verteilten Systemen entstehen verschiedene Modelle, die zwischen den unterschiedlichen Peers kommuniziert werden müssen. Um diese Modelle zu verteilen wird, auf Grund der Ressourcenbeschränkung aus Abschnitt 1.1.1, eine platzsparende Darstellungsmethode benötigt. Um die verteilten Modelle verarbeiten zu können, ist es notwendig dass sie zu einem lokalen Modell aggregiert werden. Damit nicht jedes dieser verteilten Modelle bei der Aggregation „dekomprimiert“ werden muss ist es wünschenswert dass dieser Schritt direkt mit den „komprimierten“ Modellen funktioniert.

In diesem Ansatz wird beschrieben wie DTs durch Fourier Spektren dargestellt werden können. Im folgendem werden Attribute und Klassenlabel auf binäre Werte beschränkt, dies dient der Vereinfachung ist aber nicht notwendig für das Verfahren. Die Grundidee ist den DT als numerische Funktion (Fkt.) darzustellen. Es ist offensichtlich, dass man die textuellen Attributbelegungen und Klassifizierungslabel auf numerische Werte abbilden kann. Hierzu siehe Abbildung 5.3.

So beschreibt jeder Pfad von der Wurzel zu einem Blatt eine Fkt. die eine Attributbelegung auf ein Klassifizierungslabel abbildet. Auf die so entstandene diskrete Fkt. lässt sich natürlich auch jede Transformation anwenden. Also auch die Fourier-Transformation.

Fourier-Grundlagen Eine Funktion $f : X^l \to \mathbb{B}$ kann durch die Fourier-Basis-Funktionen $f(x) = \sum_j w_j \neg\psi_j(x)$ dargestellt werden.

Dabei beschreibt $\neg\psi_{j(x)}$ die komplexe Konjunktion von $\psi_{j(x)}$ mit $x, j \in \{0,1\}^l$. Der String j wird Partition genannt und die Ordnung einer Partition entspricht der Anzahl von Werten $\neq 0$.

$\psi_j(x) = (-1)^{(<x,j>)}$; $w_j = \frac{1}{2^l} \sum_x f(x)\psi_j(x)$

$f(x)$ ist die numerische Darstellung des Klassenlabels welches x zugeordnet wird.

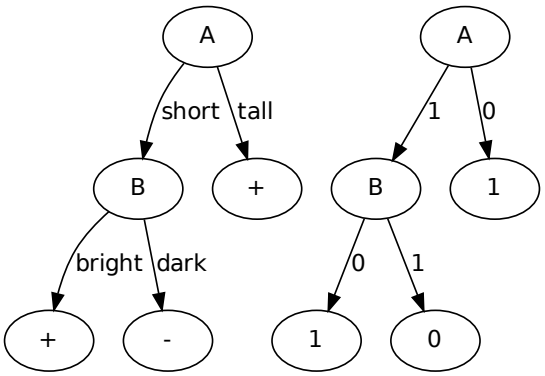


Abbildung 5.3: Vorbereitung eines DTs damit er als Fkt. dargestellt werden kann.

Vom DT zum Fourier-Spektrum Ein Pfad von der Wurzel zu einem Knoten beschreibt die Menge aller Instanzen die an diesem Knoten noch gültig sind. Sollte es sich um ein Blatt handeln so ist die Klassifizierung abgeschlossen, falls nicht muss weiter im Baum abgestiegen werden. So ein Pfad wird Schema genannt, mit $h \in \{0,1\}^l$. Sei m ein Pfad für $A = 0$ dann ist $h = \{0 * *\}$. Die „*“ sind Wildcard-Symbole für die Werte, die das entsprechende Attribut annehmen kann. Sei Λ die Menge aller $\{0,1\}^l$-Bit-Strings, also aller möglichen Instanzen, und n sei die Anzahl der Blätter.

Dann wird der FC w_j wie folgt berechnet:

$$\begin{aligned} w_j &= \frac{1}{|\Lambda|} \sum_{x \in \Lambda} f(x)\psi_j(x) \\ &= \frac{1}{|\Lambda|} \sum_{x \in h_{(1)}} f(x)\psi_j(x) + \cdots + \frac{1}{|\Lambda|} \sum_{x \in h_{(n)}} f(x)\psi_j(x) \end{aligned}$$

Hier muss man allerdings den vollständigen Bildbereich betrachten. Damit das nicht nötig ist wird das Ergebnis des Skalarprodukts wie folgt definiert:

$* \cdot 0 := 0$

und erweitern die Fkt. $\psi_j(x)$

$$\psi'_j(x) = \begin{cases} 0 & \text{wenn } \exists i : j_i = 1 \ \ und \ \ h_i = * \\ \psi_j(h) & \text{else} \end{cases}$$

Durch auflösen der Summen in w_j erhält man

$$w_j = \frac{|h_1|}{|\Lambda|} f(h_1)\psi'_j(h_1) + \cdots + \frac{|h_n|}{|\Lambda|} f(h_n)\psi'_j(h_n).$$

Die Berechnung eines FC w_j ist also effizient möglich. Also ist auch das Fourier-Spektrum effizient berechenbar wenn die Anzahl der FC nicht zu hoch ist.

Theorem 5.1.1. *Ein Fourier-Spektrum zu einem DT mit begrenzter Tiefe k, hat nur eine polynomielle Anzahl(in der Anzahl von Attributbelegungen) von FC die nicht Null sind.*

Beweis. Wenn $\exists i : j_i = 1$ und $h_i = *$ ist, ist $\psi'_j(h) = 0$. Und die maximale Tiefe des Baumes sei k. Also ist jeder FC w_j mit $j > k = 0$ da es in dem entsprechendem Baum keinen Pfad gibt auf dem alle durch j beschriebenen Attribute liegen. Angenommen es gäbe so einen Pfad, dann hätte der die Länge $j > k$ und das widerspricht der Annahme, dass der DT die Tiefe k hat. □

Vom Fourier-Spektrum zum DT Die Grundidee zur Überführung wird hier nur kurz skizziert.

Man berechnet den Baum anhand des Informationsgehaltes. Dafür wird die Durchschnittsfunktion

$$\phi(h) = \frac{1}{|h|} \sum_{x \in h} f(x)$$

betrachtet, die den durchschnittlichen Wert der Klassifizierung angibt.

- $|h|$ Anzahl aller möglichen Belegungen
- Problem: man müsste für alle $x \in h$ f(x) kennen

Wie in [23] und [30] beschrieben, kann $\phi(h)$ aber auch direkt durch die FCs bestimmt werden. Die $entropy(h)$ lässt sich dann wie folgt berechnen:

$$entropy(h) = -\phi(h)log(\phi(h)) - (1 - \phi(h))log(1 - \phi(h))$$

Damit ließe sich ein naiver Algorithmus, der dem C4.5[37] nachempfunden ist, sofort umsetzen. Allerdings steigt die Anzahl benötigter FC exponentiell zur Ordnung von h an. Hinzu kommt das viele FC Null sind, also unnötig berücksichtigt werden.

Sei h der Ordnung l, h' der Ordnung $l - 1$ und h' subsumiere h. Dann ist es möglich $\phi(h)$ effizient zu berechnen indem man gewonnenes Wissen, von der Berechnung von $\phi(h')$, nutzt.

Aggregation von Fourier-Spektren Es werden verschiedene Modelle erzeugt um jedoch eine möglichst genaue und korrekte vorhersage zu erhalten werden alle gesammelten Daten benötigt. Damit die erzeugten Modelle nicht zwischen den Übertragungen zurück in DTs umgewandelt werden müssen um sie zu aggregieren wäre es wünschenswert das sich die Fourier-Spektren direkt aggregieren lassen. Die Aggregation von DTs in der Darstellung als Fourier-Spektrum ist möglich. Ein häufiger und simpler Ansatz, eine Klassifikation von einem Ensemble von DTs zu bekommen, ist die gewichtete Linearkombination der einzelnen Klassifikationen. Eine genauere Betrachtung dieses Problems ist aber nicht Inhalt dieses Dokumentes.

5.1.5 Clustering Distributed-Data-Streams

Dieser Abschnitt befasst sich mit dem Peer-To-Peer *k*-Means-Algorithmus (P2PKM-Algorithmus) [6, 17], mit welchem verteilte Sensordaten in Gruppen von ähnlichen Daten, auch Cluster genannt, unterteilt werden können. Bevor der Algorithmus vorgestellt wird, wird zunächst der Begriff der Clusteranalyse erläutert. Anschließend wird der *k*-Means-Algorithmus vorgestellt, auf welchem der P2PKM-Algorithmus aufbaut.

Clusteranalyse Die Clusteranalyse ist eine Häufungsanalyse, um Cluster in multidimensionalen Datensätzen zu finden. Cluster sind Gruppen von "ähnlichen" Objekten auf Grundlage eines Distanz- bzw. Unähnlichkeitsmaßes [27]. Multidimensionale Datensätze können z. B. aus kartesischen Koordinaten bestehen. Ähnliche Objekte wären in diesem einfachen Beispiel Daten, die nah beieinander liegen. Die Merkmale eines Objekts können aber auch von ganz anderem Typ sein, wie z. B. Temperaturen, Druck, Lichtmessungen, Schall u. v. m. Bei der Clusteranalyse sind immer folgende Daten gegeben:

- Ein Datensatz $X = \{x_1, x_2, ..., x_n | x_i \in \mathbb{R}\}$
- Die Anzahl der Cluster: K
- Eine Distanzfunktion $D(x, y)$
- Eine Qualitätsfunktion

Gesucht wird bei der Clusteranalyse eine Menge an Clustern

$$C = \{C_1, C_2, ..., C_K | C_i \subset X, C_i \cap C_j = \emptyset\}.$$

Jeder Datenvektor ist dabei genau einem Cluster zugeordnet und die Qualitätsfunktion, auf die später noch eingegangen wird, wird dadurch optimiert. Der Abstand, bzw. die Unähnlichkeit zwischen zwei Objektmerkmalen $D(x_i, y_j)$ kann z. B. durch den Differenzbetrag, den Normalisierten Differenzbetrag oder die quadrierte Distanz bestimmt werden. Der Abstand zwischen zwei Objekten $D(x, y)$ kann z. B. durch den gewichteten Durchschnitt, den gewichteten Durchschnitt mit quadrierter Distanz, die euklidische Distanz oder durch eine Maximums-Metrik berechnet werden.
Die Qualitätsfunktion hängt von zwei Faktoren ab. Bei einem optimalen Clustering wird der innere Abstand in jedem Cluster *minimiert* und die Zwischenunähnlichkeit zwischen jedem Cluster *maximiert*:

- Innerer Abstand:
$$W(C) = \frac{1}{2} \sum_{i=1}^{K} \sum_{x \in C_i} \sum_{y \in C_i} D(x, y)$$
- Zwischenunähnlichkeit:
$$B(C) = \frac{1}{2} \sum_{i=1}^{K} \sum_{x \in C_i} \sum_{y \notin C_i} D(x, y)$$

Untergruppen in der Clusteranalyse bilden dichtebasierte Verfahren, hierarchische Clusteringverfahren, sowie partitionierende Verfahren. Der *k*-Means-Algorithmus im nächsten Abschnitt gehört zu den partitionierenden Verfahren.

***k*-Means-Algorithmus** Der am meisten verwendete *k*-Means-Algorithmus ist der Lloyd-Algorithmus, der die Methode der kleinsten Quadrate für ein optimales Clustering verwendet und die euklidischen Distanzen zwischen Objekten berechnet. Jeder Cluster besitzt einen Zentroiden, welchem eine Menge von Datenobjekten zugeordnet sind. Der Algorithmus besteht aus vier Schritten:

1. Initialisierung der Zentroide
2. Zuordnung jedes Objekts zu seinem nächsten Zentroiden
3. Neuberechnung des Zentroids in jedem Cluster
4. Wiederhole ab Schritt 2 bis Abbruchkriterium erfüllt ist

Im klassischen *k*-Means werden die initialen Zentroide zufällig gewählt. Es wird abgebrochen, wenn die neu berechneten Zentroide sich nicht von den Zentroiden der vorhergegangenen Iteration unterscheiden, oder wenn ein Schwellwert überschritten wird. In den Abbildungen 5.4 bis 5.7 wird der Algorithmus an einem Beispiel verdeutlicht.

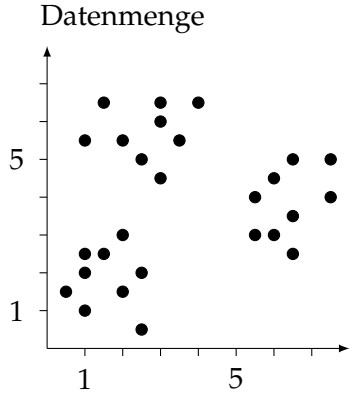


Abbildung 5.4: Illustration des *k*-Means: Initiale Datenmenge

Peer-To-Peer *k*-Means-Algorithmus Der Hintergrund zum P2PKM-Algorithmus beinhaltet die folgende zentrale Frage: Wie können Cluster gefunden werden, wenn die Daten nicht an einem zentralen Rechner verfügbar, sondern über mehrere Knoten verteilt sind?
Jeder Peer sammelt seine eigenen Sensordaten. Eine Energiekarte hilft dem Schwarm

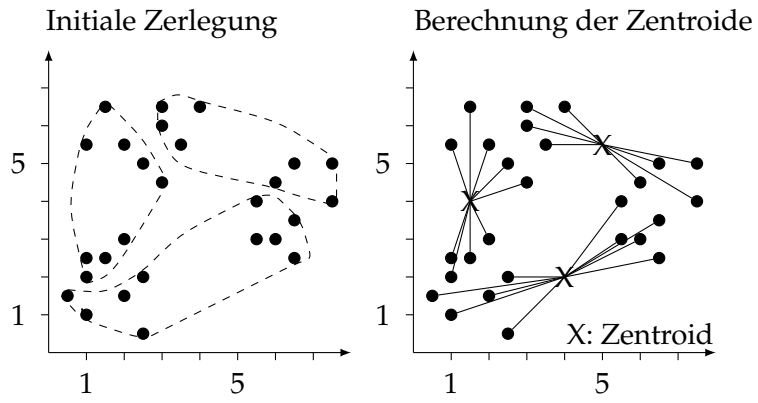


Abbildung 5.5: Illustration des *k*-Means: Initiale Zerlegung

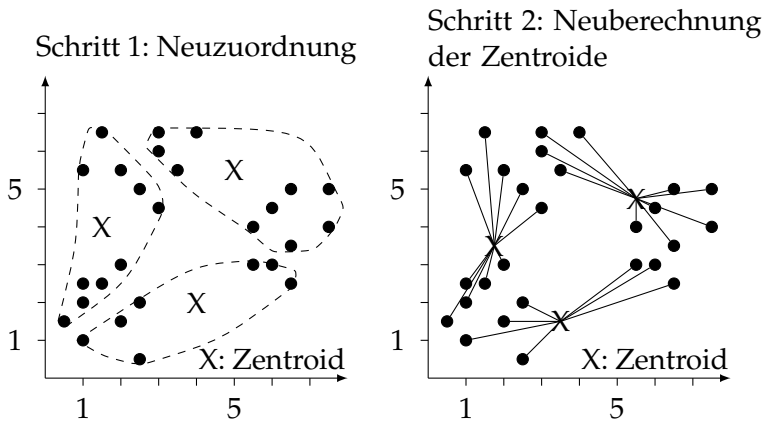


Abbildung 5.6: Illustration des *k*-Means: Zerlegung nach einer Iteration

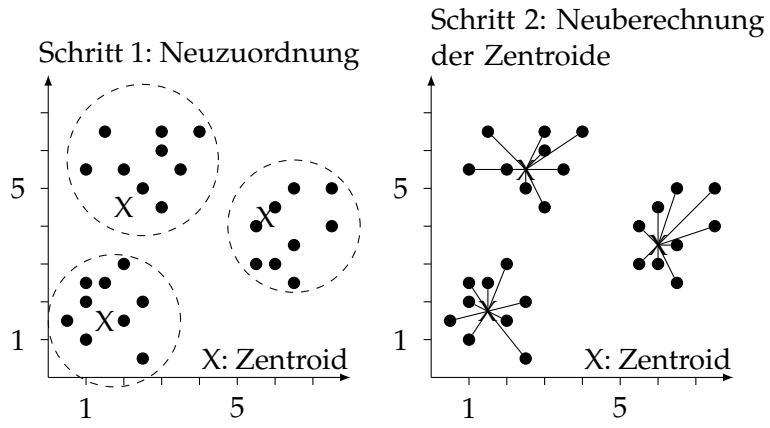


Abbildung 5.7: Illustration des *k*-Means: Zerlegung nach vier Iterationen

nur dann, wenn sie über alle Daten berechnet wurde. Mit dem klassischen *k*-Means-Algorithmus kann jeder Peer jedoch nur seine eigenen Daten verwenden, um eine lokale Energiekarte zu erstellen.
Die Anforderungen an das Szenario in der Projektgruppe sind damit folgende:

- Der Algorithmus muss verteilt, also auf jedem einzelnen Knoten stattfinden.
- Jeder Knoten sollte möglichst viele Informationen von anderen Knoten sammeln und diese in seinen Berechnungen nutzen.
- Der Datenverkehr zwischen den Knoten sollte möglichst gering gehalten werden.
- Der Fehler, den jeder Knoten aufgrund fehlender Informationen erhält, sollte möglichst klein gehalten werden.

Der P2PKM-Algorithmus könnte in sieben Schritte gegliedert werden:

1. Initialisierung der Zentroide:
   Wie beim *k*-Means-Algorithmus werden auch hier zur Einfachheit die initialen Zentroide zufällig ausgewählt. Dafür benötigt jeder Knoten den gleichen Zufallsgenerator.
2. Schritt 2 und 3 von *k*-Means mit lokalen Daten:
   Jeder Knoten muss zunächst ein Update seiner lokalen Zentroide durchführen. Dafür werden die letzten gesammelten Daten und die zuletzt berechneten Zentroide verwendet. Der Vorgang ist identisch mit den Schritten 2 und 3 des im vorherigen Abschnitt dargestellten *k*-Means-Algorithmus.
3. Lokale Zentroide und Clusteranzahl in *History Table* speichern:
   Alle Paare
   $$(w_{j,k}^{(i)}, n_{j,k}^{(i)}), 1 \leq j \leq K$$

werden in einer Tabelle gespeichert. $w_{j,k}^{(i)}$ steht für die Position des j-ten Zentroids des Peers i in der k-ten Iteration. $n_{j,k}^{(i)}$ steht für die Anzahl der Daten im j-ten Zentroid des Peers i in der k-ten Iteration. Die Datenpaare in der so genannten *History Table* werden benötigt, wenn beispielsweise Knoten Daten aus einem älteren Iterationsschritt anfragen möchten.
Weiter hat jeder Peer seine eigene *Poll Table*, in der sämtliche Datenanfragen anderer Knoten hinterlegt werden. Jede Anfrage ist eindeutig durch eine Knotenbezeichnung und einen Iterationsschritt gekennzeichnet.

4. Auf Anfragen anderer Knoten reagieren und aus *Poll Table* entfernen:
   Für alle gespeicherten Anfragen

   $$< i_1, k_a >, ..., < i_m, k_b >$$

   muss an jeden anfragenden Knoten eine Antwort

   $$< i, (w_{j,k}^{(i)}, n_{j,k}^{(i)}) >, 1 \leq j \leq K$$

   gesendet werden. Jede Anfrage, auf die geantwortet wurde, kann aus der *Poll Table* entfernt werden. Der Wert k bezeichnet hier immer den angefragten Iterationsschritt und jedes i gibt an, welcher Peer eine Anfrage gestellt hat. Auf die Anfragen anderer Knoten muss unterschiedlich reagiert werden:

   a) Der angefragte Iterationsschritt ist kleiner als der eigene Iterationsschritt:
      In diesem Fall kann ein Peer seine Zentroide versenden, da sie in der *History Table* gespeichert sind.

   b) Der angefragte Iterationsschritt ist größer oder gleich dem eigenen Iterationsschritt und der angefragte Knoten ist nicht terminiert:
      Wenn der angefragte Knoten die Daten für diesen Iterationsschritt bereits gesammelt hat, kann er sie versenden. Ansonsten muss er diese Anfrage in seiner *Poll Table* abspeichern.

   c) Der angefragte Iterationsschritt ist größer oder gleich dem eigenen Iterationsschritt und der angefragte Knoten ist terminiert:
      Der angefragte Knoten muss hier die Zentroide aus seinem letzten aktiven Iterationsschritt versenden.

5. Anfragen an andere Knoten senden:
   Es wird davon ausgegangen, dass jeder Peer weiß, mit welchen Nachbarknoten es gerade kommunizieren kann. Es muss darauf jeweils eine Anfrage $< i, k >$ an alle seine Nachbarknoten senden.

6. Neuberechnung der Zentroide mit globalen Daten:
   Der gerade betrachtete Peer hat nun Antworten auf seine Anfragen von ande-

ren Knoten erhalten. Jetzt findet eine Neuberechnung der Zentroide mit globalen Daten statt:

$$v_{j,k+1}^{(i)} = \frac{\sum_{l \in (comb^{(i)} \cup \{i\})} w_{j,k}^{(l)} n_{j,k}^{(l)}}{\sum_{l \in (comb^{(i)} \cup \{i\})} n_{j,k}^{(l)}}, j = 1, ..., K$$

$(Comb^{(i)} \cup \{i\})$ ist die Menge aller Indizes der Knoten, die auf die Anfrage von Knoten N_i geantwortet haben und der Index von Knoten N_i selbst. Es werden in der obigen Formel die Zentroide aller Knoten und die eigenen Zentroide gemäß ihrer Anzahl an Daten gewichtet und über deren gewichteten Mittelwert jeweils ein neuer Zentroid berechnet. Dadurch wird vermieden, dass ganze Datensätze transferiert werden. Damit nicht unendlich lange auf eine Antwort gewartet werden muss, sieht die Funktion *Comb* einen Timeout vor.

7. Wiederhole ab Schritt 2 bis Abbruchkriterium erfüllt ist:
   Im letzten Schritt des Algorithmus wird geprüft, ob das Abbruchkriterium bereits erfüllt ist. Wenn sich die im vorherigen Schritt berechneten Zentroide nicht mehr signifikant, z. B. anhand eines Schwellwertes, verändert haben, terminiert ein Knoten. Ansonsten beginnt er wieder bei Schritt zwei und erhöht seine Iteration um eins.

Ein Knoten wacht auch immer dann auf, wenn sich seine lokalen Daten geändert haben. Er muss danach auch allen anderen Knoten mitteilen, dass er neue Informationen der Umgebung sammeln konnte, indem er ihnen seine neuen Zentroide sendet.

5.1.6 Dichtebasierte Clusterverfahren

Der vorgestellte *k*-Means-Algorithmus (vgl. Abschnitt 5.1.5) ist ein einfaches und schnelles Verfahren zur weitgehend zuverlässigen Clusterbestimmung unter der Voraussetzung, dass die Cluster vorwiegend konvex sind und ihre Anzahl im Vorhinein abschätzbar ist. Falls diese Bedingungen nicht zutreffen, bietet das dichtebasierte Clustering eine Alternative. Der wahrscheinlich bekannteste dichtebasierte Algorithmus ist der Algorithmus *DBSCAN* von Ester *et al.* [19]. Cluster werden hier anhand einer Überdeckung von ϵ-Umgebungen bestimmt, welches Erfassungen beliebig geformter Cluster und eine Repräsentation gemäß der Form ermöglicht. Eingeführt werden diese ϵ-Umgebungen durch die im Folgenden definierten *core-Punkte*. Dazu werde von nun an angenommen, dass die Parameter $minPts \in \mathbb{N}$ und $\epsilon > 0$ gegeben seien und P eine Menge von Punkten bezeichne.

Definition 5.1.3 (core-Punkt, ϵ-Nachbarschaft)**.** *Sei $p \in P$ ein Punkt und $d : P \times P \to \mathbb{R}^+$ ein Abstandsmaß. Die* ϵ-Nachbarschaft *von p ist definiert als*

$$N_\epsilon(p) = \{q \in P | d(p,q) \leq \epsilon\}.$$

p heißt core-Punkt *wenn $|N_\epsilon(p)| \geq minPts$.*

Die Punkte in ϵ-Nachbarschaften haben verschiedene Relationen zueinander.

Definition 5.1.4 ((direkt) Dichte-erreichbar, -verbunden)**.** *Es seien $p, q \in P$. Wenn es einen* core-*Punkt c gibt, sodass $p \in N_\epsilon(c)$ ist, dann heisst p* direkt Dichte-erreichbar *von c. p heißt* Dichte-erreichbar *von q, wenn es eine Folge von Punkten*

$$p_1 = q, p_2, \ldots, p_n = p$$

gibt, sodass p_{i+1} direkt Dichte-erreichbar von p_i ist. p heißt Dichte-verbunden *zu q, wenn es einen Punkt c gibt, sodass p und q Dichte-erreichbar von c sind.*

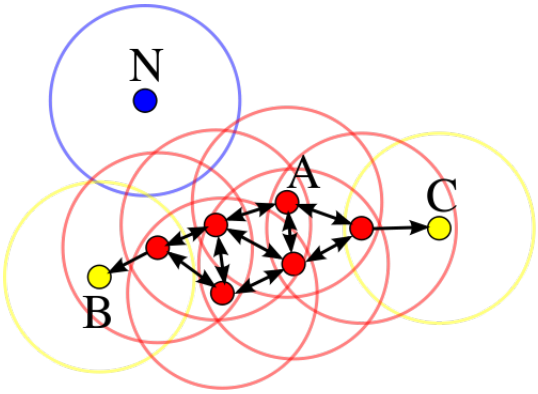


Abbildung 5.8: Darstellung der Dichte-Verbundenheit unter Punkten ($minPts = 3$). Direkte Dichte-Erreichbarkeit wird durch Pfeile symbolisiert, core-Punkte sind rot, Noise-Punkte blau und border-Punkte[3] gelb dargestellt[4].

Ein Punkt p ist also direkt Dichte-erreichbar von allen *core*-Punkten in seiner ϵ-Nachbarschaft, und er ist Dichte-erreichbar von allen *core*-Punkten c, für die eine Folge von *core*-Punkten $c_1 = c, \ldots, c_{n-1}$ existiert mit $c_i \in N_\epsilon(c_{i+1})$ und $c_{n-1} \in N_\epsilon(p)$. So ist der Punkt B in Abbildung 5.8 Dichte-erreichbar von allen roten Punkten, und Dichte-verbunden mit C. Durch diese Begriffe der *Dichte* wird nun ein Cluster definiert.

Definition 5.1.5 (Cluster)**.** *Eine Menge $C \subseteq D$, $C \neq \emptyset$ heißt* Cluster, *wenn*

1. *für alle core-Punkte $c \in C$ und alle Dichte-erreichbaren Punkte p von c gilt, dass $p \in C$ ist,*
2. *für alle Punkte $p, q \in C$ gilt, dass p und q Dichte-verbunden sind.*

[3] Border Punkte werden in [19] eingeführt als Punkte, die Dichte-Erreichbar sind von einem core-Punkt, aber selbst keine core-Punkte sind

[4] `http://de.wikipedia.org/w/index.php?title=Datei:DBSCAN-Illustration.svg&filetimestamp=20111020080029`

Alle roten und gelben Punkte aus Abbildung 5.8 bilden also einen Cluster. Es wird in [19] gezeigt, dass Cluster gemäß dieser Definition genau allen Dichte-erreichbaren Punkten von einem beliebigen *core*-Punkt (*seed*) aus dem Cluster entsprechen. Der Algorithmus DBSCAN nutzt diese Eigenschaft und sucht zunächst auf der Menge aller Punkte nach einem noch nicht bekannten core-Punkt. Dieser ist der seed eines noch nicht bekannten Clusters, der durch die Bestimmung aller Dichte-erreichbaren Punkte von diesem core-Punkt aus erschlossen werden kann. Die Punkte des Clusters werden aus der Menge an Punkten gelöscht und iterativ wird ein weiterer core-Punkt gesucht, bis alle Punkte betrachtet wurden.

Zu diesem Verfahren gibt es bereits eine Variante, die auf Streams arbeitet, den Algorithmus *DenStream* [12]. Ein wichtiger Unterschied liegt in der Interpretation der Cluster. Diese werden nicht mehr als eine Menge von Punkten, sondern als eine Menge von Gebieten, die den ϵ-Nachbarschaften entsprechen, angesehen. Die sogenannten Micro-Cluster $c(z, w, r)$, wobei z das Zentrum, w ein zugeordnetes Gewicht und $r \leq \epsilon$ den Radius angibt, fassen die Informationen über Datenpunkte innerhalb von r-Umgebungen zusammen, wie die *Cluster feature*-Vektoren in *BIRCH* [40]. Das Gewicht berechnet sich anhand der momentanen Zeit und den Zeitpunkten zu denen die überdeckten Punkte in dem Stream übermittelt wurden. Es werden *p-Micro-Cluster* (potentially Micro-Cluster) und *o-Micro-Cluster* (outlier Micro-Cluster) anhand ihrer Gewichte unterschieden. P-Micro-Cluster haben ein Gewicht $w \geq \mu$ für einen Schwellwert μ, ihre Zentren bilden ein Pendant zu den core-Punkten in DBSCAN. O-Micro-Cluster haben ein Gewicht von $w < \mu$, sie stehen im Verdacht nur Noise-Punkte zu überdecken.

DenStream arbeitet in zwei Phasen, einer online- und einer offline-Phase. In der online-Phase werden die Datenpunkte Micro-Clustern zugeordnet. Wenn ein Punkt von einem oder mehreren Micro-Clustern überdeckt wird, so geht die Information über den betrachteten Datenpunkt in die Parameter des nächstliegenden Micro-Clusters ein. Andernfalls wird durch den betrachteten Datenpunkt ein neuer Micro-Cluster erstellt. Auf diese Art und Weise wird in dieser Phase eine Punkteüberdeckung mithilfe von r-Umgebungen, $r \leq \epsilon$, erstellt. In der offline-Phase wird eine Variante von DBSCAN auf die p-Micro-Cluster angewandt, in der Micro-Cluster als virtuelle Punkte betrachtet und geclustert werden. Dementsprechend muss die Definition der Dichte-Erreichbarkeit angepasst werden.

Definition 5.1.6 ((direkte) Dichte-Erreichbarkeit, -Verbundenheit bei Micro-Clustern)**.** *Es seien $c_p(z_p, w_p, r_p)$ und $c_q(z_q, w_q, r_q)$ zwei p-Micro-Cluster. c_p heißt* direkt Dichte-erreichbar *von c_q, wenn $d(z_p, z_q) \leq \epsilon$. c_p heißt* Dichte-erreichbar *von/* Dichte-verbunden *zu c_q, wenn es eine Folge von p-Micro-Clustern*

$$c_{p_1} = c_q, c_{p_2}, \ldots, c_{p_n} = c_p$$

gibt, sodass $c_{p_{i+1}}$ direkt Dichte-erreichbar von c_{p_i} ist.

Dichte-Erreichbarkeit definiert sich hier also über die Schnittmengen von ϵ-Umgebungen der Zentren von p-Micro-Cluster und Cluster werden dementsprechend über

die Dichte-Verbundenheit definiert, d. h. zwei p-Micro-Cluster mit nichtleerem Schnitt gehören immer zum gleichen Cluster.

In Hinsicht auf die Vorgehensweise dieses Algorithmus gibt es einige Kritikpunkte. Die zufällige Wahl der Zentren der Micro-Cluster, die lediglich von der Reihenfolge eingehender Datenpunkte abhängt, kann leicht dazu führen, dass einige p-Micro-Cluster nicht als solche erkannt werden. Die Nichtreproduzierbarkeit der Punkteverteilung innerhalb eines Micro-Clusters und die einhergehende Absorption der Informationen über verarbeitete Punkte führt weiterhin dazu, dass für neu entstehende Micro-Cluster die Informationen über gesehene Datenpunkte in ihrer Umgebung nicht genutzt werden können, und p-Micro-Cluster, die als solche schon erkennbar sein könnten dennoch nicht identifiziert werden. Generell kann angenommen werden, dass durch die weitgehende Abstraktion in DenStream, auf einem endlichen Datenstrom weniger Cluster entdeckt werden als es bei gleichzeitiger Verfügbarkeit der Daten für DBSCAN der Fall wäre. In Bezug auf die gegebene Anwendung ist es dementsprechend wünschenswert nicht nur eine Möglichkeit zur verteilten Berechnung, sondern auch eine Möglichkeit zur Reproduzierbarkeit der gesammelten Dateninformationen zu finden.

5.2 Das *Stream-Framework*

Um die oben genannten Algorithmen umzusetzen und diese in einem gemeinsamen Kontext zu halten, erscheint ein Framework sinnvoll. Die *Stream-API*[5] ist ein Framework für Data-Mining auf Datenströmen, das am LS 8 entwickelt wird und dem Anytime-Paradigma (vgl. Einleitung zu Kapitel 5) folgt. Algorithmen für dieses Framework werden in einem XML-Format verfasst. Durch den modularen Aufbau der Algorithmen in XML besteht die Möglichkeit, schnell verschiedene Plotter oder Datenausgaben zu ergänzen, um die Ergebnisse zu veranschaulichen. Die *Stream-API* stellt außerdem schon einige Data-Mining-Lerner- und Statistik-Services zur Verfügung.

Da bei den Data-Mining-Algorithmen Sensorwerte evaluiert werden, die als kontinuierlicher Datenstrom vorliegen, und diese Algorithmen im Zweifel schnell umgestellt oder ergänzt werden müssen, bietet das *Stream-Framework* mit seiner Baukastenstruktur eine sehr gute Grundlage. Die Verteilung der Algorithmen in Form einer Erweiterung der *Stream-API* stellt dabei eine Herausforderung dar, die in Abschnitt 7.3.3 betrachtet wird.

Im Folgenden wird zum einen der Aufbau der *Stream-API* und der XML-Struktur näher erläutert und zum anderen ein Einblick in die Software-Architektur gegeben, da dort die Grundlagen für die Implementierung der Verteilung zu finden sind.

5.2.1 Aufbau

Abbildung 5.9 zeigt den generellen Aufbau eines *Stream-API*-Containers. In einem Container können mehrere so genannte Prozesse existieren. Sie verarbeiten Daten, die aus

[5] Repository: `https://bitbucket.org/cbockermann/streams`

verschiedenen Quellen stammen können, egal ob externe Daten, die per Stream eingelesen werden, oder interne Daten, die von einem anderen Prozess erzeugt und in eine Queue gelegt werden. Neben der Datenverarbeitung besteht auch noch die Möglichkeit des Monitorings, um z. B. den Speicherverbrauch zu überwachen und mitzuloggen. Im Folgenden werden die einzelnen Komponenten näher erläutert.

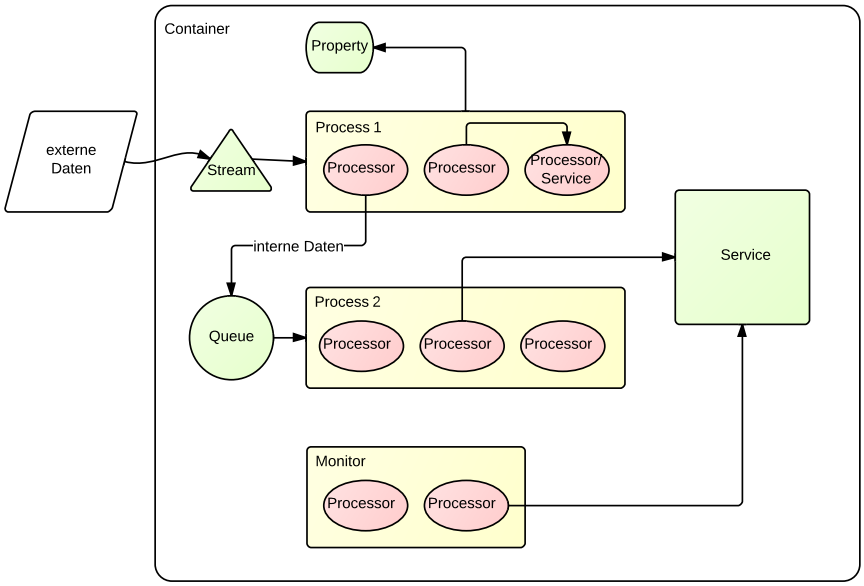


Abbildung 5.9: Aufbau eines Stream-API-Containers

Container Ein *Container* gibt eine Gesamtumgebung vor, in der alle anderen *Stream-API*-Objekte existieren können, er bildet die Hülle. Die interessantesten Attribute für einen Container sind:

- *id*: Name des Containers
- *import*: essentiell zum Finden der Prozessoren; hier werden alle Java-Packages mit Komma getrennt angegeben, in denen die verwendeten Prozessoren-Klassen liegen
- *namingService* (optional): Klasse des *NamingServices*

Alle Klassen, die als Attribute in der XML-Datei auftauchen, müssen mit dem kompletten Java-Package-Pfad angegeben werden.

Property Eine *Property* ist ein Name/Wert-Paar mit den entsprechenden Attributen *name* und *value*. Diese werden als containerweite Variablen als Mapping gespeichert und können mit *${name}* in Attributen referenziert werden.

Stream Ein *Stream* ist ein Strom von Daten in einem bestimmten Format und besteht aus drei essentiellen Dingen:

- *id*: die Referenz, über die er angesprochen wird
- *class*: der vollständige Name der Klasse des *Streams*
- *url*: die URL des eigentlichen Streams

Es können noch weitere Attribute gesetzt werden, sofern die *Stream*-Implementierung diese benötigt. Ein *Stream* kann als *input* für einen beliebigen *Process* genutzt werden, in dem der Datenstrom dann Datum für Datum verarbeitet wird.

Service Ein *Service* stellt eine Reihe von Funktionen bereit, um ein Objekt (z. B. ein Modell) zu manipulieren. Containerweite *Services* machen es möglich, sie in mehreren Prozessen referenzieren zu können. Auch sie haben wie *Streams* eine *id* und eine *class*, können aber noch mehr Attribute besitzen. Das ist rein von der Klasse abhängig, die den *Service* implementiert. Wie bei allen weiteren Klassen-basierten Tags wird auch hier vom Konzept der *Injection* Gebrauch gemacht, siehe weiter unten in Abschnitt 5.2.3. Neben den datenunabhängigen *Services* können sie auch Prozessoren sein. So kann z. B. ein Statistik-*Service* gleichzeitig ein *Processor* sein, um so die Daten direkt im *Service* auswerten zu können.

Process In einem *Process* werden nacheinander verschiedenste Prozessoren auf einen Datenstrom angewendet. Ein *Process* hat neben den Standard-Attributen noch einen *input*, der den Datenstrom angibt, auf dem gearbeitet werden soll. Zusätzlich gibt es die Möglichkeit, eine Zahl *copies* anzugeben, um Kopien dieses Prozesses anfertigen zu lassen. So können bequem mehrere Streams durch die gleichen Algorithmen laufen, ohne dass diese explizit in der XML-Datei aufgeführt werden müssen. Zur Unterscheidung der Kopien wird ein Index (von 0 bis *copies*-1) an den Namen angehängt.

Monitor Ein *Monitor* funktioniert ähnlich wie ein *Process*, mit dem Unterschied, dass er keinen *input* benötigt und dafür immer nach einem bestimmten *interval* ausgeführt wird. Ein *Monitor* kann z. B. verwendet werden, um Statistiken über *Services* oder System-Ressourcen wie Arbeitsspeicherverbrauch zu erstellen. Allgemein dient ein Monitor dazu, regelmäßige „Überwachungs"-Vorgänge durchzuführen.

Processor Die meisten Prozessoren erben von *AbstractProcessor*, sind aber die Komponenten, die der User/Entwickler selbst programmieren muss. Als XML-Tag wird hier nicht wie bisher *Processor* und darin ein Attribut *class* verwendet, sondern die Namen der Klassen selbst. Sofern der volle Java-Package-Name im *import* des Containers auftaucht (siehe oben), wird nicht der volle Name der Klasse benötigt. Wie gehabt werden die Attribute via *Injection* (s. u.) in den jeweiligen *Processor* eingefügt. Es ist darauf zu achten, dass ein *Processor* die Daten manipulieren kann, die eingespeist werden, da die Rückgabe eines *Processors* die Eingabe für den Nächsten ist.

5.2.2 Algorithmen in XML

Aus dem abstrakten Aufbau des *Stream-Framework*s lässt sich direkt die XML-Struktur der Algorithmen für das Framework ableiten. Ein Beispiel für einen solchen Algorithmus findet sich in Listing 5.1. Zuerst werden über eine *Property* eine globale Variable *count*, die angibt, was später gezählt werden soll, und ein CSV-Stream *mystream*, der dem einzigen *Process* in diesem Beispiel als *input* dient, angelegt. Der *Process* besteht hier aus mehreren Prozessoren. Wenn ein Datum über den Stream kommt, wartet der *Process* 10 ms, bevor er das Datum an den *Counter* weitergibt. Dieser prüft, ob in dem gegebenen Datum die Einträge *protocol_ type* oder *service* auftauchen und zählt diese. Der nächste Prozessor gibt die aktuelle Anzahl an gezählten Vorkommen aus. Abschließend werden zwei double-Werte über den *keys*-Parameter aus dem Datum geparst und dann geplottet. Die Data-Mining-Algorithmen aus Abschnitt 5.1 werden in diesem XML-basierten Format verfasst.

Listing 5.1: Beispiel für ein Stream-API XML-File

```
<container import=
  "stream.flow,pg.counter, stream.plotter,stream.parser">

  <property name="count" value="protocol_type,service"/>

  <stream id="mystream" class="stream.io.CsvStream"
      url="classpath:/kddcup-small.csv.gz"
      separator="," limit="10000"/>

  <process input="mystream">
    <Delay time="10ms" />

    <Counter id="counter" keys="${count}"/>
    <PrintCounter counter-ref="counter"/>

    <ParseDouble keys="src_bytes,dst_bytes"/>
    <Plotter keys="src_bytes,dst_bytes"/>
  </process>

</container>
```

Listing 5.1: Beispiel für ein Stream-API XML-File

5.2.3 Software-Architektur

Dieser Abschnitt betrachtet ein paar Mechanismen der Stream-API-Architektur genauer. Er bildet die Grundlage für die Implementierung der Verteilung in Abschnitt 7.3.3.

XML Handling Beim Parsen der XML-Dateien baut das Framework auf der Standard-Bibliothek org.w3c.dom[6] von Java auf. So existiert für die Standard-Tags, die oben beschrieben wurden, entweder ein *Element-* oder *DocumentHandler*, der für die Verarbeitung der entsprechenden Elemente zuständig ist. Anhand der im XML gegebenen Parameter und Referenzen werden neue Objekte erstellt, die das gewünschte Verhalten umsetzen sollen. Diese Objekte werden mittels Parameter-Injection (siehe nächster Absatz) in einer *ObjectFactory* erzeugt. Es gibt sowohl die Möglichkeit, eigene *ElementHandler* an das Framework weiterzugeben, als auch eigene *ObjectCreator* in die *ObjectFactory* einzubinden.

Parameter- und Service-Injection Das Konzept der *Injection* wird genutzt, um bei Zusammenhängen, die zur Compile-Zeit unklar sind, eine stabile und vor allem vollständige Gesamtkonstruktion zu erhalten. Da die Algorithmen, die mit der *Stream-API* ausgeführt werden, in XML geschrieben sind, gibt es keine Garantie, dass die einzelnen *Services*, *Streams*, *Processes* etc. in chronologischer Reihenfolge aufgelistet sind, beziehungsweise keine Querverweise bestehen.

Neue Objekte werden also mittels Default-Konstruktor erzeugt und die Attribute anschließend mit Setter-Methoden initialisiert. Bei der Parameter-Injection werden alle Setter außen vor gelassen, die eine Service-Referenz setzen würden. Diese Methoden werden zwischengespeichert, um in der abschließenden Service-Injection behandelt zu werden. So ist sichergestellt, dass alle Services und sonstigen Bestandteile erzeugt wurden, bevor sie zugewiesen werden.

Die oben erwähnten Property-Elemente in der XML-Datei dienen dazu, Variablen in Parametern zuzulassen. Während der Parameter-Injection werden diese Parameter expandiert, sprich durch die *Property* in das jeweilige echte Objekt umgesetzt. Etwas anders verhält es sich bei den *Services*: Sobald ein *Service* fertiggestellt ist, wird er im *NamingService* bekannt gemacht, d.h. der *Service* wird unter seiner ID registriert. Ein *Service* ist also eindeutig durch den Containernamen und seine ID bestimmt.

Bei der Service-Injection werden letztlich alle Referenzen gesetzt, die während der Parameter-Injection als offen markiert wurden. Dabei werden ähnlich wie bei den Parametern die Referenzen expandiert, in dem der Wert des Parameters im *NamingService* nachgeschlagen wird. Erst nach Abschluss dieser Injection werden die Prozesse gestartet. In Abbildung 5.10 ist der komplette Ablauf grafisch dargestellt.

***NamingService* und Kommunikation** Das Kernstück während der Ausführung ist der *NamingService*, denn er ist dafür zuständig, die *String*-Referenzen, die in der XML-Datei angegeben sind, als *Services* aufzulösen. Im *Stream-Framework* stehen zwei Standard-*NamingServices* zur Verfügung: Zum einen der *DefaultNamingService*, der ein einfaches Mapping von Strings zu Services verwaltet und nur im lokalen Container arbeiten kann. Zum anderen den *RMINamingService*, der, wie der Name schon sagt, auf der Standard-RMI[7]-Bibliothek von Java aufbaut (s. u.). Im Falle von RMI wird entweder ei-

[6] http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/package-summary.html
[7] **R**emote **M**ethod **I**nvocation

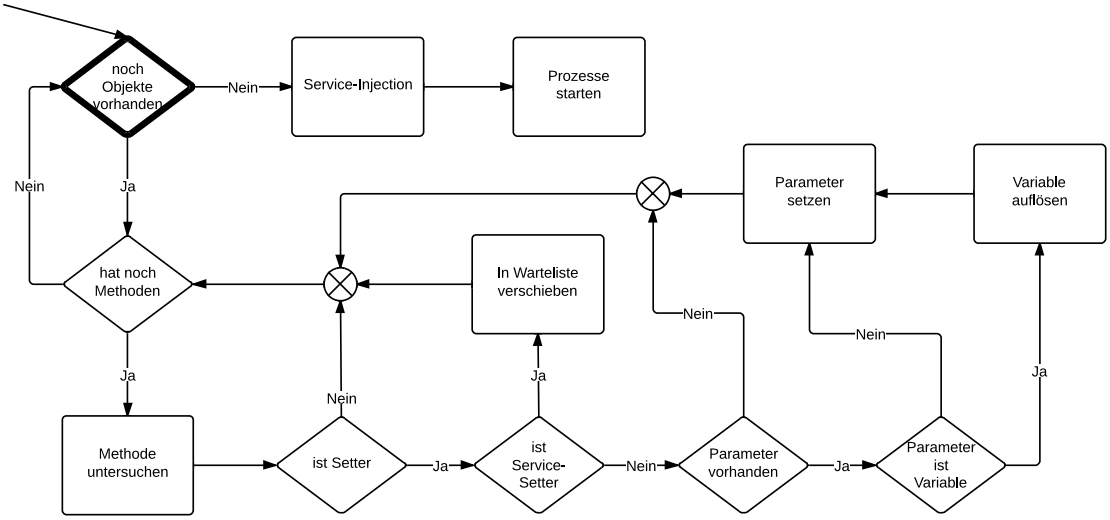


Abbildung 5.10: Ablauf der Parameter-Injection

ne neue RMI-Registry angelegt oder eine bereits existierende abgefragt, in der sowohl der *NamingService* selbst als auch alle anderen *Services* registriert werden und somit später abgerufen werden können. Soll entfernt auf einen *Service* zugegriffen werden, wird ein Lookup mit der Syntax *//containername/servicename* durchgeführt; dabei wird bei einem nicht-lokalen *Service* der entfernte *NamingService* herangezogen (in diesem Fall ein *RMIClient*), der vom lokalen Container aus nur Leserechte auf den entfernten Container hat.

5.3 Umsetzung

Im folgenden Abschnitt werden die Data-Mining-Algorithmen erläutert, welche für das *Stream-Framework* implementiert wurden. Zuerst wird jedoch die Generierung der Testdaten beschrieben, die unter anderem als Datensatz für verschiedene Simulationen dienen. Als erster Algorithmus wird der verteilte *k*-Means-Algorithmus erläutert, welcher ein Clustering auf mehreren verschiedenen Streams durchführt, so dass alle beteiligten Peers zu einem ähnlichen Ergebnis kommen. Daraufhin wird ein etwas ausgefeilterer Clusteringansatz beschrieben, bei dem es darum geht, mit Hilfe vieler kleiner Cluster ein akkurates Gesamtbild zu erhalten. Schließlich wird die Implementierung des Hierarchical Heavy Hitters-Algorithmus erläutert und darauf eingegangen, in welcher Form dieser für das Projekt von Nutzen ist.

Damit die verschiedenen Algorithmen nicht die selbe Arbeit doppelt machen, werden die Sensordaten auf den einzelnen Peers zentral verarbeitet, bevor sie an die entsprechenden Algorithmen übergeben werden.

5.3.1 Generierung von Testdaten

Sowohl für die Hardware-in-the-Loop-Mobilitätssimulation, als auch zur Erprobung der Data-Mining-Algorithmen werden Testdaten benötigt. Die Programme zur Erzeugung dieser Daten sollen dabei möglichst flexibel gehalten werden, um eine Vielzahl von Situationen nachstellen zu können. Die Funktionen hierfür sind in R geschrieben, da es eine rasche Implementierung erlaubt und gute Visualisierungsmöglichkeiten bietet.

Das definierte Spielfeld (vgl. 1.1.1), auf dem die Messwerte erhoben werden sollen, wird als Matrix dargestellt und jeder Zelleneintrag stellt einen Messwert dar. Dies spiegelt die reale Situation der Einteilung der Umgebung in GPS-Koordinaten dar, welche auch nur eine begrenzte Auflösung haben und keine kontinuierlichen Positionsangaben liefern. Die Messwerte sind ebenfalls diskrete Werte. Dies lässt sich damit begründen, dass die Algorithmen später ebenfalls eine Einteilung in beispielsweise hell, weniger hell und dunkel vornehmen. Somit ist es möglich, von den tatsächlichen Werten zu abstrahieren, zumal der tatsächliche Wertebereich der Sensormesswerte Anfangs noch nicht bekannt war.

Als Basis für die Messwerte wurde angenommen, dass sowohl durch Umwelteinflüsse als auch durch die Sensoren selbst mit einem gewissen Grundrauschen zu rechnen ist. Dies wird dadurch simuliert, dass jeder Zelle zufällig ein gleichverteilter Wert aus einem vorgegebenen Bereich zugewiesen wird. Um zusammenhängende Gebiete mit einem höheren Wertebereich zu erzeugen, welche später als Cluster erkannt werden sollen, werden Punkte mittels einer multivariaten Normalverteilung [26] zufällig erzeugt und ihre Position auf das Spielfeld übertragen. In den geschriebenen Skripten ist es möglich, sowohl die Anzahl als auch die Dichte und Streuung der Datenpunkte zu steuern. Dies geschieht, indem der Erwartungswert μ, die Kovarianzmatrix Σ und die Anzahl der gezogenen Punkte als Parameter der Funktion übergeben werden. Zu beachten ist dabei, dass die Anzahl der gezogenen Punkte höher ist als die Anzahl der Zellen, denen der entsprechende Wert zugewiesen wird. Dies liegt an der Diskretisierung der Werte, denn mehr als ein gezogener reelwertiger Zufallswert kann beim ganzzahligen Runden einer Zelle zugewiesen werden.

Als weitere Verfeinerung werden Randzonen für die Cluster definiert. Dies geschieht, indem die Kovarianzmatrix Σ mit einem festen Faktor multipliziert wird. Gemäß der neuen Verteilung werden weitere Punkte erzeugt und ihnen ein eigener Wert zugewiesen. Dies kann auch mehrfach wiederholt werden, um mehrere Übergangsbereiche zu erzeugen. Der Vorteil dieses Verfahrens besteht darin, dass die Grenzgebiete zwar erkennbare, aber nicht scharf getrennte Zonen haben und auch im inneren Bereich noch niedrigere Energiewerte liegen können, wie in Abbildung 5.11 zu sehen ist.

Sample-Strategien für Messwerte Der nächste Schritt besteht darin, aus diesen Messwerten Stichproben zu erzeugen, welche dem späteren tatsächlichen Verhalten der Roboter ungefähr entsprechen sollten. Als Strategie hierfür erweist sich ein einfacher Random Walk als geeignet. Das heißt von einer Startposition aus wird zufällig eine neue

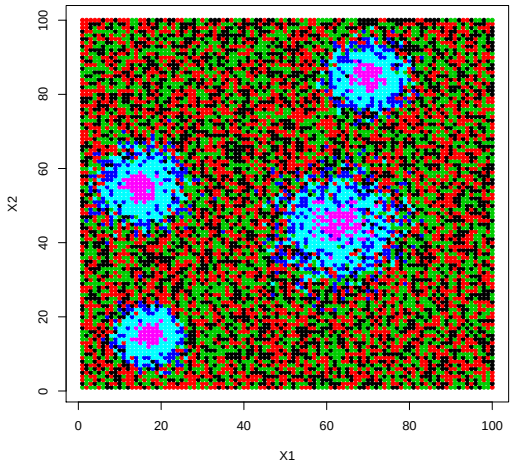


Abbildung 5.11: Beispiel Messwerte mit weichen Clusterrändern; Rot, Grün und Schwarz stellen niedrige, die übrigen Farben stellen hohe Energiewerte dar.

benachbarte Koordinate ausgewählt. Um diese Bewegung etwas realistischer zu gestalten, ist eine leichte Tendenz eingefügt, eine bestimmte Richtung zu bevorzugen. Auf dem begrenzten Spielfeld hat dies zunächst den Effekt, dass sich die Stichproben nach einer Weile an den Randbereichen häufen. Als Lösung wird eine weitere Regel hinzugefügt, welche dafür sorgt, dass die bevorzugte Richtung neu bestimmt wird, falls der Rand des Spielfeldes berührt wird. Ein nach diesem Verfahren erzeugter Random Walk zeigt die Abbildung 5.12.

Mit diesem Auswahlverfahren ist es nun möglich, beliebig große Stichproben aus einem zuvor generierten Spielfeld zu erzeugen. Die Stichproben werden als einfache CSV-Datei abgespeichert, in der jede Zeile die Position und den zugehörigen Messwert enthält. Für die ersten Testzwecke wird ein Spielfeld mit 100×100 Feldern gewählt, die niedrigen Energiewerte werden mit drei Werten dargestellt und ebenso die hohen Energiewerte. Jeder Cluster von hohen Messwerten ist somit von zwei Zonen mit etwas niedrigeren Werten umgeben. Das Spielfeld, welches während des ersten Semesters für die Testzwecke der Data-Mining-Gruppe verwendet wurde, weist vier separate Clustergebiete auf, wobei sich zwei Cluster an ihren Rändern leicht überschneiden. Als eigentliche Testdaten wurden drei Random Walks nach dem oben beschriebenen Verfahren und mit einer Länge von 5000 Schritten durchgeführt. Diese waren die Grundlage für die ersten Tests der Algorithmen.

In Hinblick auf die geplanten Feldversuche wurde eine weitere Klasse von Testdaten erdacht. Diese stellen im Groben die Situation auf einem realen Parkplatz dar, auf dem es lange freie Flächen gibt und schattige Gebiete durch gepflanzte Baumreihen. Diese sollen dem Beispiel aus der Einleitung entsprechen (Abb. 1.1). Allerdings sind für die

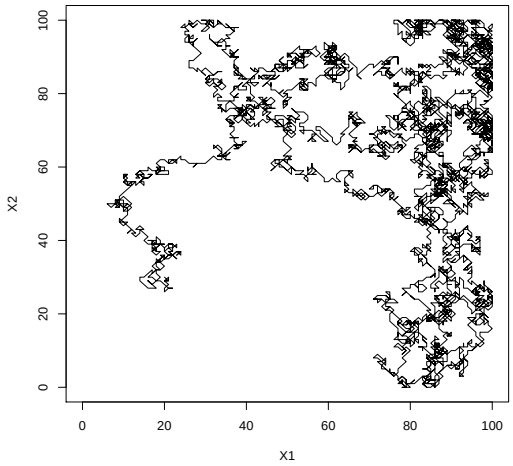
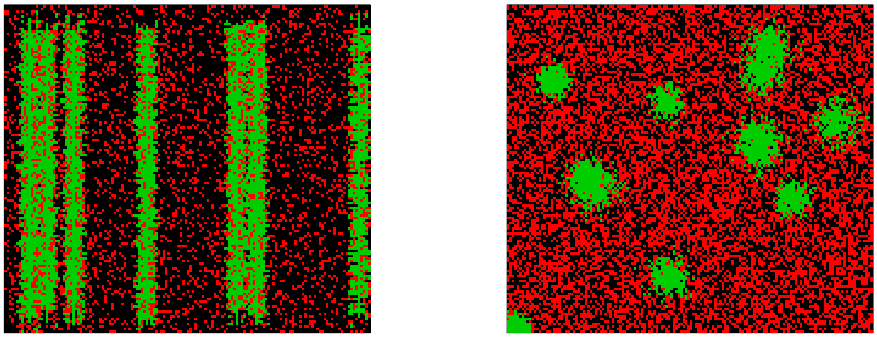


Abbildung 5.12: Random Walk mit Richtungsgewichtung

Auswertung der Algorithmen mehrere derartiger Daten notwendig um Unterscheide erkennen und die Leistungsfähigkeit der verwendeten Algorithmen beurteilen zu können. Deswegen wurden analog zu den obigen Clusterdaten eine neue Kategorie von Datensätzen erzeugt. Diese bestehen aus einem Grundrauschen von dunklen und mittel hellen Punkten, sowie unterschiedlich breiten länglichen Bereichen mit einer hohen Lichtintensität. Diese einfachere Einteilung der Kategorien wurde auch für die Clusterdaten benutzt, damit die Ergebnisse in der Simulation des Gesamtsystems leichter zu interpretieren sind. Die Abbildung 5.13 zeigt jeweils einen solchen Datensatztyp. Als Referenzdaten wurden je hundert dieser Datensätze erzeugt und die Klassenwerte durch dazu passende zufällige Lux-Werte ersetzt um alle Verarbeitungsschritte zu testen.

5.3.2 Distributed *k*-Means

Um die in Abschnitt 1.1.1 beschriebene Heat-Map erstellen und Orte mit besonderer Helligkeit oder Temperatur identifizieren zu können, wurde u. a. der Ansatz des Clusterings verfolgt. Der aus der Seminarphase bekannte verteilte *k*-Means-Algorithmus (siehe Abschnitt 5.1.5) wurde im *Stream-Framework* implementiert und als erster Algorithmus auf den generierten Testdaten evaluiert. Da der Algorithmus nicht für gestreamte Daten entwickelt wurde, mussten einige Anpassungen vorgenommen werden. Auf naive Art und Weise wurde zunächst ein one-pass Algorithmus (zur näheren Erläuterung siehe [25]) entwickelt, in dem jeder Datenpunkt nur einmal betrachtet und danach verworfen wird. Die Hoffnung lag hierbei in der Annahme, dass die Zuordnung zu den Zentroiden ausreichend viele Informationen über bereits gesammelte Daten transportieren würde, sodass keine Datenpunkte gespeichert werden müssen. Die



(a) Beispiel Parkplatzdatensatz (b) Beispiel Clusterdatensatz

Abbildung 5.13: Beispiele für Testdatensätze der Simulation: einem Parkplatz ähnlicher Datensatz und zufällige Clusterdaten

Testdaten wurden als dreidimensionale Datenpunkte interpretiert, sodass die ersten beiden Merkmale die Position bestimmen und das dritte den zugehörigen Energiewert anzeigt.

Der Aufbau des entstandenen Streaming-Algorithmus lässt sich Abbildung 5.14 entnehmen. In drei Prozessen werden drei verschiedene Datenströme behandelt. Die gemessenen Datenpunkte werden *Process 1* übermittelt, in welchem alle Berechnungen bzgl. der lokalen Daten durchgeführt werden, die Schritte 2, 3 und 5 des P2PKM-Algorithmus betreffen (siehe Abschnitt 5.1.5). Die Menge gesammelter Daten von Schritt 2 entspricht hier einer parametrisierten Anzahl von gestreamten Daten während eines Iterationsschritts. Nach dem Start einer Umfrage von Prozessor *Poll*, wird der Prozess zum Verarbeiten der Antworten anderer Peers durch das Einfügen einer Startnachricht in die *Answerqueue* gestartet. Die Startnachricht gibt dem Prozessor *Receiver* die jeweiligen Nachbarn an, auf deren Antworten maximal für eine gewisse Zeitspanne (timeout) gewartet werden soll, bis neue Zentroide berechnet und dem *Cluster*-Prozessor übergeben werden können.

Bei der Visualisierung der Ergebnisse wurden einige Probleme offenbar. Es stellte sich heraus, dass das Messen von Daten entlang eines Weges spezielle und unbeachtete Eigenheiten mit sich bringt. Wie bereits erwähnt, wurden bisher die neuen Zentroide nur aufgrund von Zuordnungen der Punkte zu den Zentroiden und den verteilten Berechnungen bestimmt. Die Parameter wurden in den Beispielen so gewählt, dass nach 200 bearbeiteten Datenpunkten ein neuer Iterationsschritt beginnt und die Clusteranzahl $K = 12$ beträgt. Zudem wurde das dritte Merkmal des Energieniveaus mit dem Faktor 15 multipliziert, sodass der Wertebereich des Energiebetrags dem Intervall $[15, 90]$ entspricht, und somit dem Energielevel eine ähnliche Bedeutung in der

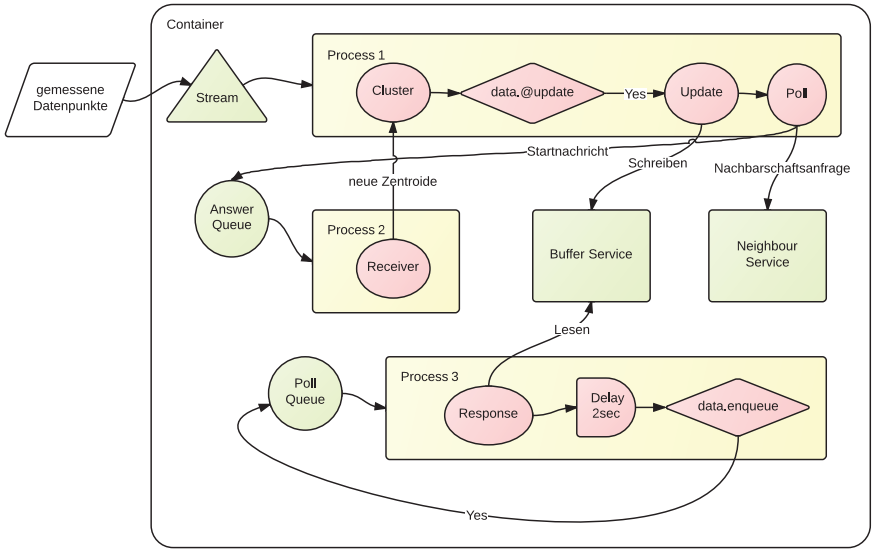


Abbildung 5.14: Der verteilte *k*-Means-Algorithmus im *Stream-Framework*.

Abstandsberechnung zukommt wie den Positionsmerkmalen, deren Werte in $[0, 100]$ liegen. Gemäß diesen Einstellungen waren anhand einiger Experimente die besten Resultate erzielt worden. Abb. 5.15 zeigt zwei Iterationsschritte des verteilten Streaming-*k*-Means aus der Sicht von einem Peer (*Peer0*). Das erkundete Gebiet wird durch kleine farbige Punkte symbolisiert. Man kann sehen, dass die Daten des ersten Iterationsschritts drei Clustern zugeordnet werden, ihre Updates werden in grün dargestellt. Zwei der Zentroide wurden nur von diesem Peer aktualisiert, sie befinden sich im zweiten Iterationsschritt an dem Platz ihrer Updates von *Peer0*. Der andere Zentroid (orange) befindet sich im zweiten Iterationsschritt links mittig. Die Abbildung zeigt ein Phänomen des entwickelten Algorithmus, die Zentroide wandern mit dem Peer. Die Zentroide, die von *Peer0* im ersten Iterationsschritt aktualisiert wurden, werden auch im zweiten aktualisiert. Die Updates bewegen die Zentroide jeweils in Richtung des erkundeten Gebiets, die Information über im vorherigen Iterationsschritt gesammelte Daten werden kaum widergespiegelt. Durch die stetige Bewegung des Peers liegen die Zentroide, die in einem Schritt nahe sind, häufig auch im darauffolgenden nah, wie es für die ersten Iterationsschritte von *Peer0* der Fall ist (siehe Abb. 5.15). Die verteilte Berechnung über die Peers relativiert zwar das Problem, da hierdurch auch Informationen über betrachtete Gebiete von anderen Peers in die Berechnung der Zentroide eingehen, jedoch ist der Algorithmus immer noch stark abhängig von den zuletzt gesehenen Datenpunkten.

Um dieses Problem zu behandeln, wurden mehrere Vorschläge gemacht. Ein besonders einfacher und naheliegender war zunächst die Datenpunkte zu speichern. Abbildung 5.16 zeigt die Resultate des implementierten *k*-Means-Algorithmus für verschie-

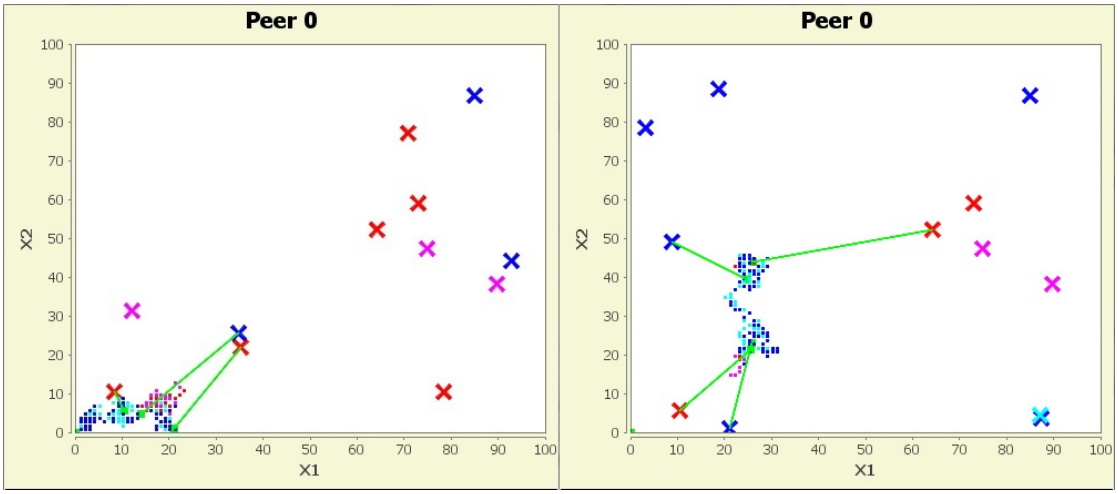


Abbildung 5.15: Visualisierung der ersten und zweiten Iteration des implementierten verteilten k-Means-Algorithmus für einen von drei Peers. Die verschiedenen Energiewerte werden durch die Farben hellblau, blau, rot und pink symbolisiert. Kreuze symbolisieren Zentroide und grüne Quadrate ihre Updates.

dene Speicherraten. Man kann sehen, dass selbst beim one-pass Algorithmus drei der vier Cluster erkannt werden. Allerdings ist dieses Ergebnis auch ein Glücksfall, meist wurden unter verschiedenen Parametereinstellungen während der Laufzeit höchstens zwei der Cluster richtig identifiziert. Der Algorithmus mit gespeicherten Datenpunkten identifiziert alle Cluster, wenn auch das Zentrum des Clusters links unten nicht ganz korrekt liegt. Das Ergebnis des Algorithmus mit jedem vierten gespeicherten Datenpunkt ist von dem mit allen gespeicherten Datenpunkten kaum zu unterscheiden. Allerdings ist zu beachten, dass der Effekt des one-pass Algorithmus zu Beginn des Random Walk beim Speichern jedes x-ten Datenpunkts ebenfalls auftritt, da bei der zweiten Iteration x-Mal soviele Datenpunkte in dem neu erkundeten Gebiet liegen wie in dem zuvor erkundeten. Dieser Effekt nimmt mit zunehmender Laufzeit linear ab, und je niedriger die Speicherrate ist, desto stärker ist der Effekt zu Beginn. Leider werden bei dieser Vorgehensweise trotzdem $\mathcal{O}(n)$ Punkte gespeichert, wenn n die Anzahl insgesamt betrachteter Datenpunkte beschreibt. Da anzunehmen ist, dass durch Schatten in der Umgebung des Peers durchaus auch abstraktere Formen von Clustern gesucht werden als in dem konvexen Fall der energiereichen Cluster in unseren Testdaten, erscheint der k-Means-Algorithmus zusätzlich ungeeignet. Aufgrund dieser Überlegungen wurde entschieden, anstatt den bestehenden Algorithmus noch weiter zu verbessern (andere Speicherungsstrategie/ andere Methode um zuvor betrachtete Datenpunkte in die Berechnungen miteinzubeziehen, Optimierung der Parameter, etc.) einen Clusteralgorithmus zu suchen, der in einer verteilten Umgebung und auf gestreamten Daten gut

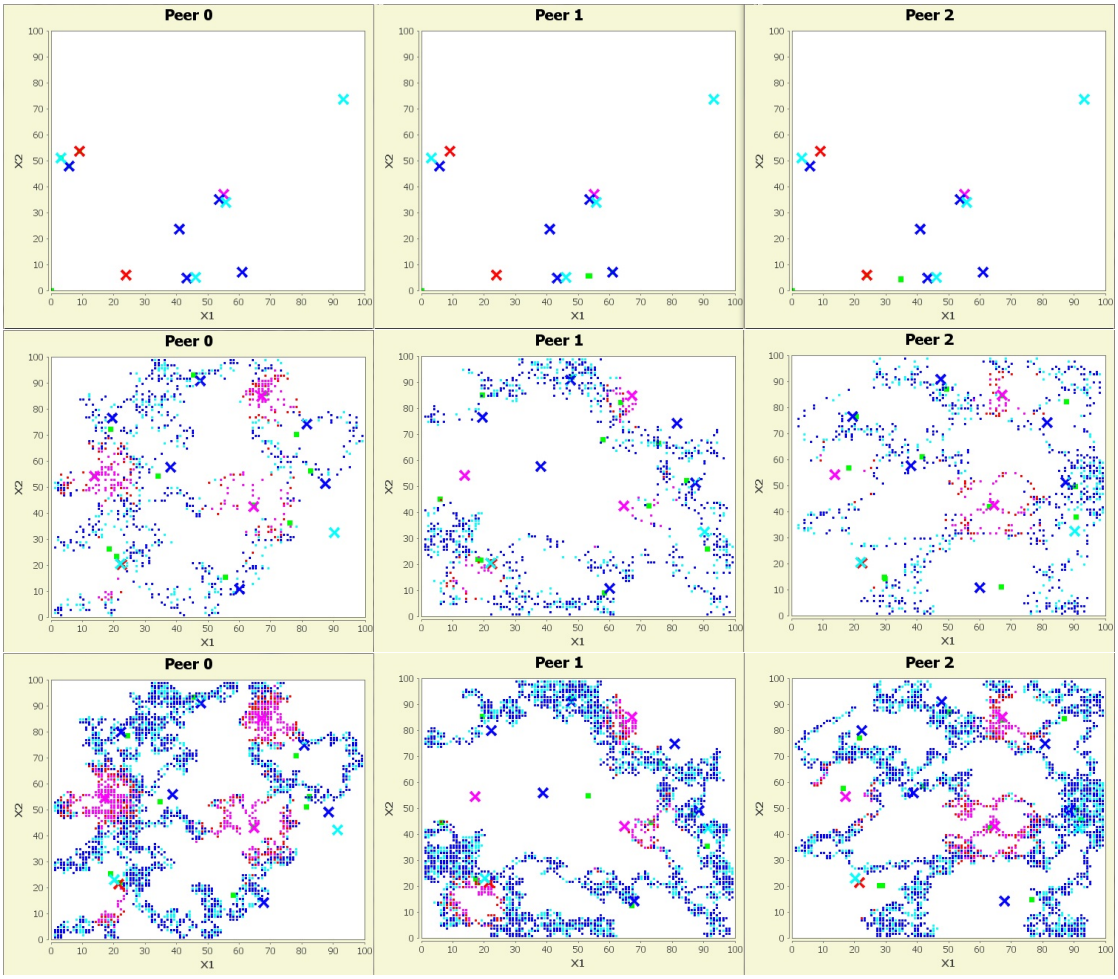


Abbildung 5.16: Die Ergebnisse des implementierten verteilten *k*-Means-Algorithmus auf den Testdaten (vgl. Abb. 5.11) für verschiedene Speicherraten (von oben nach unten: keine Speicherung, Speicherung jedes 4. Datenpunkts, Speicherung aller gesehener Datenpunkte)

funktioniert, sowie optimalerweise noch eine Möglichkeit zur Abschätzung der Clusterform/ -ausdehnung bietet.

5.3.3 Distributed DBSCAN

Unter dem Ziel einen dichtebasierten, verteilten streaming-Algorithmus zu entwerfen, wurde *DistrDBSCAN* entwickelt. Es wurde versucht von der Punkte-basierenden Darstellung von Clustern nicht so weit zu abstrahieren wie es in DenStream der Fall ist und relativ zuverlässig Cluster zu erstellen, die nach den Definitionen von DBSCAN ebenfalls als solche identifiziert werden würden.

Ein Micro-Cluster wird als Tupel $c(z_c, N_c)$ definiert, wobei z_c das Zentrum und $N_c = |N_\epsilon(z_c)|$ das Gewicht bezeichnet. Es ist $\mu = minPts$ und der Radius $r = \epsilon$ fest gewählt. Die Definition der Dichte-Erreichbarkeit bzgl. Micro-Cluster ist von DenStream übernommen. Es werden jedoch potentielle Noise-Punkte nicht zu o-Micro-Clustern zusammengefasst, d.h. es werden p-Noise-Punkte und p-Micro-Cluster unterschieden. In DenStream ist es möglich, dass das Gewicht eines p-Micro-Clusters im Laufe der Zeit unter einen Schwellwert sinkt und der p-Micro-Cluster zum o-Micro-Cluster wird. Da für die Heat-Map eine Darstellung der Entwicklung über die Zeit nicht vorgesehen ist (z.B. bzgl. des Sonnenstands), kann zunächst angenommen werden, dass Orte besonderer Helligkeit/Temperatur auch in Zukunft hell/warm sein werden, und so ein identifizierter p-Micro-Cluster auch in Zukunft ein p-Micro-Cluster sein wird.

Für p-Noise-Punkte gilt es festzustellen, ob sie einem core-Punkt entsprechen oder tatsächlich Noise-Punkte sind und gelöscht werden können. Um für einen eingehenden Punkt p die Anzahl $|N_\epsilon(p)|$ abschätzen zu können, wird die Anzahl gespeicherter p-Noise-Punkte in der ϵ-Nachbarschaft erfasst. Zudem werden in den p-Micro-Clustern $c(z_c, N_c)$ für die gilt, dass $d(z_c, p) < 2\epsilon$ ist, N_c Punkte standardnormalverteilt innerhalb der ϵ-Umgebung von c gezogen. Diese Punkte werden, falls sie in $N_\epsilon(p)$ liegen, ebenfalls dazugezählt. Auf diese Art und Weise werden relativ zuverlässig p-Micro-Cluster auch als solche identifiziert, jedoch erfordert das Speichern der p-Noise-Punkte einen höheren Speicherplatzbedarf als die Überdeckung durch o-Micro-Cluster in DenStream. Da die Bestimmung der Nachbarschaft eines Punktes eine zentrale Operation ist, werden alle Punkte in *R-Trees* [**?**] gespeichert, was eine effiziente Ausführung von Nachbarschaftsanfragen ermöglicht.

Die online- und offline-Phasen von DenStream entfallen in DistrDBSCAN. Stattdessen werden für jeden neu betrachteten Datenpunkt die Auswirkungen auf das Clustering berechnet. Es gibt verschiedene Fälle, die bei der Betrachtung eines neuen Datenpunkts p auftreten können.

1. p liegt in einem Micro-Cluster: In diesem Fall ändert sich das Clustering nur, wenn es Punkte $q \in N_\epsilon(p)$ gibt, für die $|N_\epsilon(q)| = minPts - 1$ ist, sodass durch das Einfügen von p neue Micro-Cluster eingeführt werden. Alle Cluster, deren Micro-Cluster p überdecken, werden nun zu einem Cluster zusammengefügt, da p in der Schnittmenge all dieser Cluster enthalten ist. p wird nicht gespeichert.

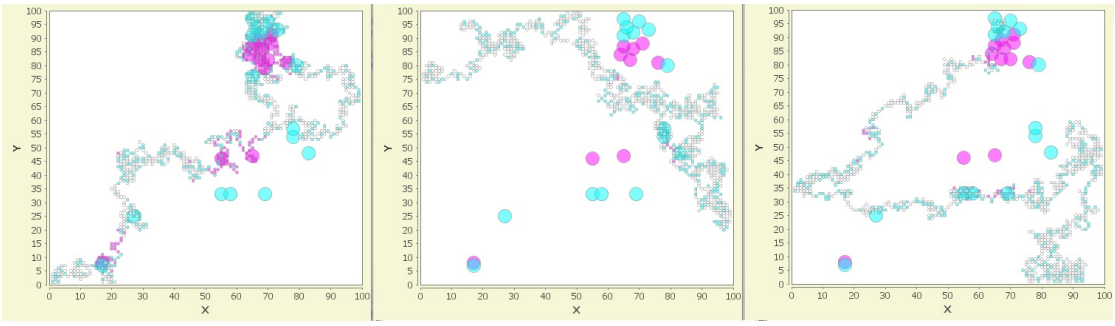


Abbildung 5.17: Zwischenergebnis bei der Ausführung von DistrDBSCAN auf den Testdaten. Die drei Farben symbolisieren die drei verschiedenen Energielevel (weiß- niedrig, blau - mittel, rot - hoch). Punkte stehen für die gesehenen Datenpunkte, Kreise für p-Micro-Cluster.

2. p ist ein core-Punkt: In diesem Fall bildet p einen Micro-Cluster der entweder einen neuen Cluster einführt, oder zu einem bestehenden gehört (wenn er eine nichtleere Schnittmenge zu einem bestehenden Cluster hat). Evtl. hat der Micro-Cluster zu p auch nichtleere Schnittmengen zu mehreren Clustern, dann werden diese Cluster wieder zu einem zusammengefügt. p wird als Micro-Cluster gespeichert und alle Punkte in der ϵ-Nachbarschaft von p werden gelöscht.

3. p ist ein p-Noise Punkt: Wenn p zu keinem Cluster gehört, wird der Punkt vorübergehend als Noise gespeichert. Alle Noise-Punkte werden nach einer parametrisierten Anzahl betrachteter Datenpunkte gelöscht.

Die Verteilung funktioniert nach dem *push*-Prinzip. Nach einer parametrisierten Anzahl neu entstandener p-Micro-Cluster werden diese an alle erreichbaren Peers gesandt. Ein Peer, der solch einen p-Micro-Cluster $c(z_c, N_c)$ empfängt, versucht diesen in sein bestehendes Clustering einzuarbeiten. Wenn z_c von einem bestehenden Micro-Cluster überdeckt wird, so hat dies keine Auswirkungen auf das bestehende Clustering. Wenn das Zentrum z_c nicht überdeckt wird, wird c als neuer Micro-Cluster hinzugefügt. Entweder schneidet die ϵ-Umgebung von c einen anderen p-Micro-Cluster c', dann ist c von c' direkt Dichte-erreichbar und c gehört zu dem gleichen Cluster wie c', oder c erzeugt einen neuen Cluster. Dieses Vorgehen hat im Vergleich zu k-Means den Vorteil, dass Kommunikation nicht mehr iterativ erfolgt und somit Zeiten, in denen besonders viele Peers miteinander kommunizieren, möglichst selten entstehen. Abbildung 5.17 zeigt einen Zwischenstand zu Beginn der Laufzeit für DistrDBSCAN auf den Testdaten. Hier wurden die Daten nicht im 3-Dimensionalen (wie bei k-Means) geclustert, sondern für 3 Energielevel separat als 2-dimensionale Punkte aufgefasst. Es wird jedoch nur für die Punkte der oberen beiden Energielevel das Clusterverfahren ausgeführt, da anzunehmen ist, dass Cluster für Punkte von besonders niedriger Energie nicht von Interesse sind. Die verschiedenfarbigen Punkte symbolisieren alle zuvor betrachteten Datenpunkte. Welche der Datenpunkte als p-Noise-Punkte gespeichert

sind, kann der Abbildung jedoch nicht entnommen werden. Nach drei neu entstandenen Micro-Clustern werden diese den anderen Peers mitgeteilt. Auf diese Art und Weise sind die Peers immer gut informiert, man kann erkennen, dass sich ihre Cluster kaum unterscheiden. Zudem werden alle erkennbaren Cluster auch als solche identifiziert. Die tatsächlichen Cluster werden zwar häufig nicht als Ganzes, sondern als mehrere erkannt, jedoch ist das auch dem Charakter des Datenstroms zuzusprechen, der durch den Random Walk nur zufällig ausgewählte Teile des Clusters beinhaltet.

5.3.4 Hierarchical-Heavy-Hitters

Der Hierarchical-Heavy-Hitters-Algorithmus, kurz *HHH*, liefert eine etwas andere Art der Wissensrepräsentation als ein Clustering-Algorithmus. Beim Clustering werden Ähnlichkeiten zwischen Datenpunkten gesucht und ähnliche Werte einem gemeinsamen Cluster zugewiesen. Beim *HHH*-Algorithmus erfolgt sowohl eine räumliche als auch eine inhaltliche Bündelung der Informationen. Zunächst muss aber die Form und Dimension der verwendeten Datenstruktur festgelegt werden. Wie im Anwendungsszenario (siehe Abschnitt 1.1.1) der Projektgruppe definiert, ist die kleinste räumliche Einheit die Kantenlänge einer Zelle der erstellten Karte, also zwei Meter. Davon ausgehend können mehrere Zellen (z.B. 2×2 Zellen) zu einer größeren Einheit zusammengefasst werden. Auch die Sensordaten werden dementsprechend zusammengefasst.

Folgende Informationen können nun aus der Ausgabe des Algorithmus abgeleitet werden:

- Auf der untersten Abstraktionsebene, also keinerlei Abstraktionen, kann festgestellt werden, ob für eine bestimmte Zelle ein bestimmtes Lichtniveau ein Heavy-Hitter ist (also häufig genug vorkommt); dies deckt sich mit der gestellten Aufgabe im Anforderungsszenario, eine Heat-Map des Geländes zu erstellen.
- Für die höheren Abstraktionsebenen können folgende Fälle betrachtet werden:
  - Tritt räumlich auf einer höheren Abstraktionsebene ein Heavy-Hitter auf, kann daraus abgeleitet werden, dass dieses Areal insgesamt eher hell ist. Für die Unterareale ist jedoch keine Aussage möglich, außer diese sind selbst Heavy-Hitter.
  - Ist eine einzelne Zelle oder ein Areal mit Lichtniveau * ein Heavy-Hitter, so ist das Ergebnis für die Heat-Map nicht eindeutig. Allerdings kann daraus die Information ableiten werden, dass in dieser Gegend schon (ausreichend) viele Messungen vorgenommen wurden, das Lichtniveau jedoch „durchwachsen“ ist. Eine solche Information ist für die Strategie des Autopiloten von Interesse, der, je nach Anforderung, ein solches Areal meidet um Gebiete mit weniger Messwerten zu untersuchen, oder es gezielt aufsucht, um mit neuen Messungen zu einer genaueren Klassifikation zu gelangen.

Im Kompressionsschritt des im Abschnitt 5.1.1 beschriebenen *HHH*-Algorithmus „altern“ die Messpunkte. Elemente e, deren Schranken $g_e + m_e \leq \lfloor \epsilon * N \rfloor$ liegen, können demnach entfernt werden, dabei ist N die Gesamtzahl der erfassten Elemente. Dies

kann eventuell zu Problemen führen, wenn davon ausgegangen wird, dass einzelne Positionen des zu erkundenden Gebiets nicht regelmäßig erneut betreten werden. Dann könnten die gewonnen Erkenntnisse über eine einzelne Zelle zu schnell verworfen werden. Hier ist die richtige Wahl des Schwellwerts ϕ möglicherweise entscheidend oder die Algorithmen müssen entsprechend angepasst werden. Dies müssen die geplanten Simulationen und reale Testszenarien zeigen.

Implementierungs-Details Um den Wertebereich der Daten des Lichtsensors flexibel zu halten, werden die Grenzen in der XML Beschreibung angeben. Die Helligkeitswerte werden entsprechend der n Grenzen auf 0,..,n abgebildet. Der Helligkeitswert -2 stellt den verallgemeinerten Helligkeitswert da.

Die Karte wird dynamisch aufgeteilt, so dass vier Felder der Abstraktionsebene $l-1$ ($2^{l-1}m^2$) die Abstraktionsebene l, mit der Größe $2^l m \times 2^l m$, bilden. Sofern das möglich ist, da es an den oberen und rechten Randbereichen vorkommen kann das dort kleinere Abstraktionsebenen vorkommen wenn die gesamte Karte nicht die Größe $2^x m^2$ hat.

Die Karte wird dynamisch aufgeteilt, so dass Felder einer Abstraktionsebene l, mit der Größe $2^l m^2$ durch vier Felder der Abstraktionsebene $l-1$ ($2^{l-1}m^2$) zusammensetzt. An den oberen und rechten Randbereichen entstehen kleinere Abstraktionsebenen wenn die gesamte Karte nicht die Größe $2^x m \times 2^x m$ hat.

Zählweise Da die Helligkeitswerte nur zwei Abstraktionsebenen haben, nämlich {*0,.., n*} und {*-2*}, ist es möglich ein eingefügtes Datum in jeder Abstraktionsebene nur einmal zu zählen. In Abbildung 5.18 wird graphisch dargestellt wie ein neues Datum in den verschiedenen Abstraktionsebenen gezählt wird.

Modell Nachdem die Vorverarbeitung abgeschlossen ist, wird das so erstellte Datenobjekt, vom Typ SimpleStructure, in die Queue des entsprechenden Peers gelegt. Sobald ein Datum in der Queue ist, wird der eigentliche *HHH*-Algorithmus (vgl. Abschnitt 5.3.4) genutzt.

In der aktuellen Implementierung wird eine Datenstruktur für alle Abstraktionsebenen verwendet; so enthält *ActualModel* (kurz *Modell*) ein Abbildungsobjekt der Form *Map<HHHDataStructure, HHHDataStructure>* in welchem Referenzen auf alle hinzugefügten Daten enthalten sind. Die eigentliche Struktur ergibt sich allerdings durch eine Elter-Beziehung, welche in den einzelnen HHHDataStructure-Objekten definiert ist, dabei hat jedes Element einen Zeiger auf die nächst höheren Abstraktionsebenen von denen es subsumiert wird.

Ein positiver Effekt ist, dass das Modell wirklich nur so groß ist wie es auch benötigt wird. In einem sehr frühen Prototypen wurden vollständiges Modells erstellt und die einzelnen Daten mit 0 initialisiert, was dazu führte, dass das Modell von Anfang an die volle Größe hatte. Auch die anschließende Suche der *HHH* war damit im Best-Case so schnell wie sie jetzt im Worst-Case ist.

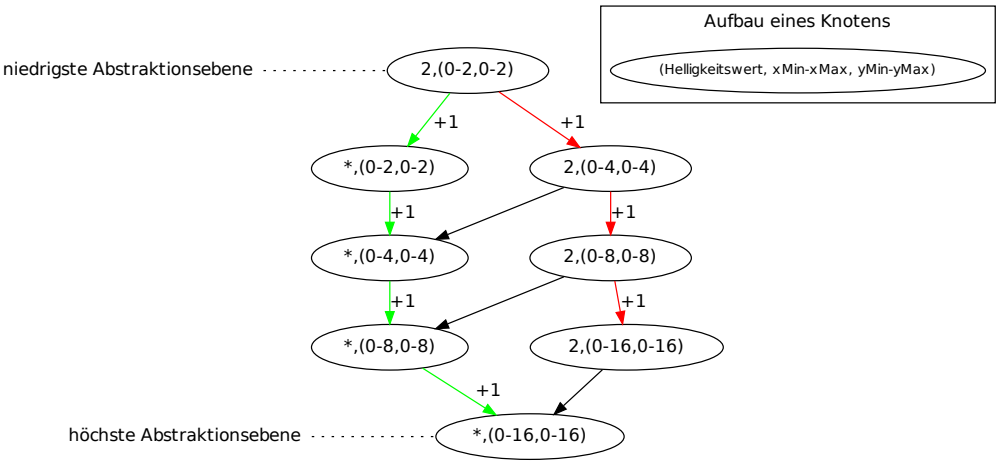


Abbildung 5.18: Ein Beispielhafter Ausschnitt eines Modells mit einem eingefügten Datum. Die Label an den Pfaden visualisieren wie ein neu hinzugefügtes Element auf den höherer Abstraktionsebenen gezählt wird.

Der zuvor erwähnte „Compress-Schritt“ ist bislang noch nicht implementiert. Ob dieser Schritt überhaupt angewendet wird, entscheidet sich erst nach den ersten Testläufe auf den Rovern.

Modell erweitern Wird ein Datum eingefügt, welches noch nicht in dem Modell vorhanden ist, werden schrittweise alle Eltern dieses Modells erzeugt, siehe Abbildung 5.19.

Sollte versucht werden, ein Datum oder ein Elter einzufügen, welches schon in dem Modell vorhanden ist, werden keine Elter-Elemente mehr erzeugt, sondern der Zähler der entsprechenden Eltern wird, wie oben beschrieben, erhöht.

Verteilung Bei der Verteilung der Daten werden nicht die vollständigen Modelle übertragen, da dies zu viel Traffic bedeuten würde; vielmehr werden für die Verteilung die lokalen *HHH* ausgewertet und per UDP an alle Fahrzeuge gesendet, die zu dem Zeitpunkt in Reichweite sind. Sollte es Verbindungsabbrüche oder sonstige Fehler in der Übertragung geben, so werden diese vollständig ignoriert. Die „externen Daten“ werden in die gleiche Queue geschrieben, in der auch die lokal gesammelten Daten gespeichert werden.

Erweitertes Anwendungsgebiet Mit dem *HHH*-Algorithmus ist es weiterhin möglich, noch nicht erkundete Gebiete zu bestimmen, um die Navigation zu unterstützen, wie im Zielszenario beschrieben (1.1.1). Hierfür bietet sich ein Top-Down-Ansatz an,

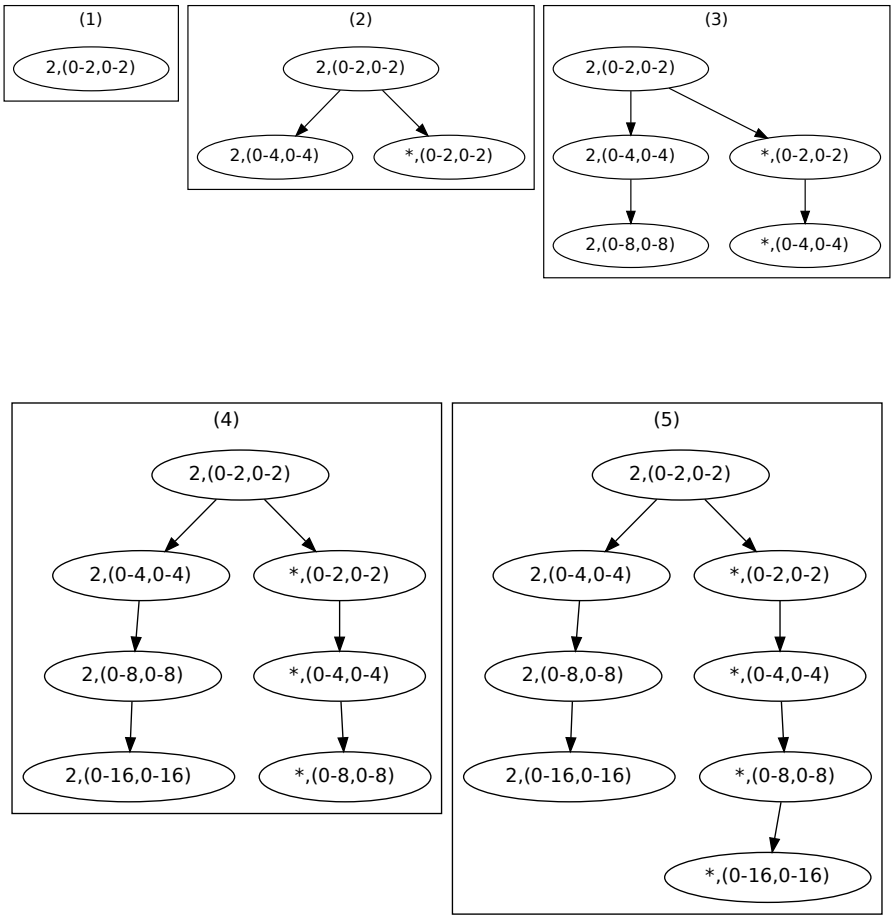


Abbildung 5.19: Beispielhafter Verlauf für das Einfügen eines Datums, bei dem noch keine höhere Abstraktionsebene existiert.

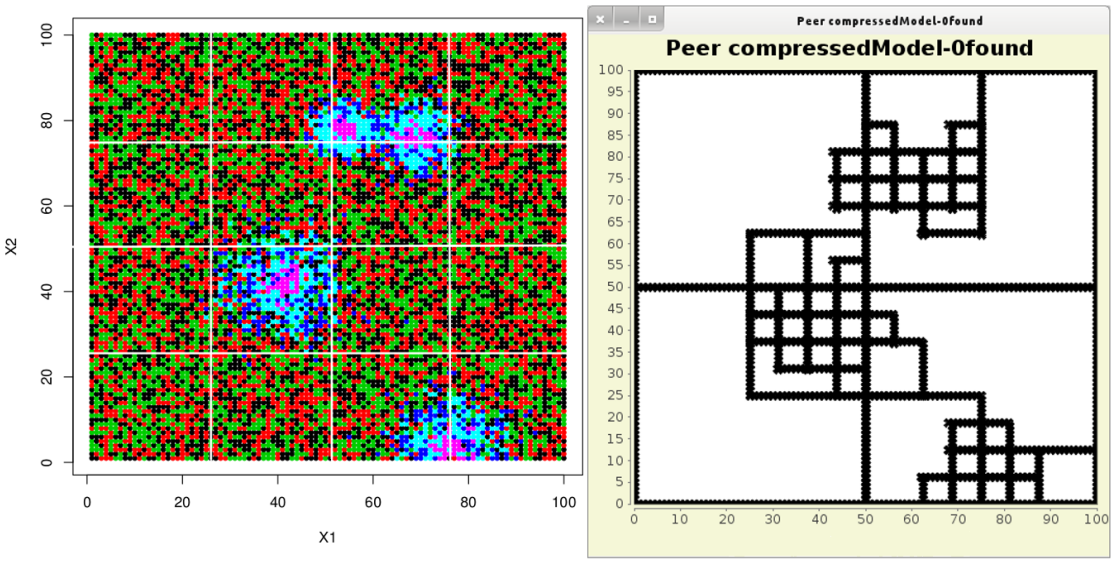


Abbildung 5.20: Optischer Vergleich zwischen der vollständigen Karte und der Ausgabe des *HHH*-Algorithmus bei gleicher Eingabe.

wenn auf einer Abstraktionsebene die Anzahl der gesammelten Daten unter einem Schwellwert liegt, gilt das Gebiet als „nicht erkundet“ und wird Teil der Ausgabe. Wenn der Schwellwert überschritten wird, wird für alle Spezialisierungen dieser Ebene der vorherige Schritt wiederholt. Die Rückgabe ist eine Liste der größtmöglichen Gebieten vom Typ „nicht erkundeten“Àls Anforderung der Kommunikationsgruppe ist die maximale Größe der Gebiete, die zurück gegeben werden, beschränkt auf $8m \times 8m$ „Felder“.

Auswertung Um die erzielten Ergebnisse besser vergleichen zu können, verfügt der *HHH*, für Test- und Evaluationszwecke, über eine Ausgabe. Hierfür wird JFreeChart[8] genutzt. Für den besseren Vergleich wurde einem Peer, der von der Verteilung ausgeschlossen ist, die gesamte Karte als Eingabe bereitgestellt. Abbildung 5.20 zeigt den direkten Vergleich der Ausgabe mit den Testdaten. Drei andere Peers arbeiten auf jeweils drei verschiedenen Beispiel-Routen über die Karte, mit jeweils 5000 Messpositionen. Diese drei Peers verteilen ihre *HHH* untereinander nach einer bestimmten Anzahl von gesammelten Daten. In Abbildung 5.20 wird das Ergebnis des *HHH*-Algorithmus, eines Peers mit der vollständigen Karte als Eingabe, mit den graphischen Darstellung der Karte verglichen. Dabei wurden nur Heavy-Hitter mit einem maximalen Lichtwert ausgeben. Was dabei ein guter Wert ist oder wo der Schwellwert am besten liegt, bedarf noch genauerer Untersuchung.

[8] http://www.jfree.org/jfreechart/

5.3.5 Aggregation der Ergebnisse der Data-Mining-Algorithmen

Eine der wichtigsten Aufgaben des Data-Mining-Agenten ist es, aus den gesammelten und verarbeiteten Daten Informationen abzuleiten und in kompakter Form weiter zu geben. Im Anwendungsfall der Projektgruppe geht es darum Wegpunktanfragen des Mobilitäts- und Kommunikationsagenten zu bearbeiten. Derzeit gibt es die drei mögliche Anfragetypen

- *Top-Energy*: Die Punkte mit der höchsten Energiedichte sollen zurückgegeben werden
- *Discover*: Noch nicht erforschte Gebiete sollen zurückgegeben werden
- *Improve*: Es sollen Gebiete ausgegeben werden, deren Erforschung das bestehende Modell verbessert

deren genauer Ablauf in Abschnitt 6.3.2 erläutert wird.

Die Aggregations-Algorithmen sind so konzipiert worden, dass sie möglichst unabhängig von den verwendeten Data-Mining-Algorithmen sind. Aus den Modellen der entsprechenden Data-Mining-Algorithmen werden die Messdaten in eine eigene Datenstruktur übertragen, die mittels direkter Verweisung auf die nächsten Einträge, einer dünnbesetzte Matrix entspricht. Der Vorteil ist dabei, dass so aus einer Sammlung von Gebieten ein einzelnes Objekt erzeugt wird, aus dem alle notwendigen Informationen extrahiert werden können.

***Top-Energy*-Anfragen** Bei diesem Anfragetyp sollen einzelne Punkte zurückgegeben werden, die eine möglichst hohe Energiedichte besitzen. Da die Sensormesswerte vorab diskretisiert werden, wäre eine einfache Ausgabe der Punkte der höchsten Klasse zu ungenau. Deswegen bedarf es weitere sekundäre Entscheidungskriterien um in dieser, möglicherweise sehr großen Menge, weiter sortieren zu können. Durch die Sortierung können bestimmte Punkte als besser angesehen werden, als andere Punkte mit gleicher Helligkeit. Als Kriterium bietet es sich an die Anzahl der direkten Nachbarpunkte zu betrachten. Die Überlegung hierbei ist, dass die Zahl der Nachbarn eine Konfidenz für die Gültigkeit dieses Punktes ausdrückt. Dies lässt sich am folgenden Beispiel verdeutlichen: Bei einem einzelnen Punkt mit hoher Intensität ohne weitere Nachbarn, könnte es sich leicht um einen Messfehler, oder um einen kurzzeitig auftretenden Effekt handeln. Im Gegenzug sprechen viele Nachbarn dafür, dass diese Region als gut erkundet angesehen werden kann und es anzunehmen ist, dass die Modelle in diesen Gebieten zuverlässige Aussagen treffen. Zu beachten ist, dass die Antwort auf diese Anfrage immer eine einzelne Zelle ist, während bei den anderen Anfragen bevorzugt größere Gebiete zurückgegeben werden. Bei diesem Anfragetyp geht es zum Beispiel um möglichst gute Punkte um sich wieder aufzuladen, weswegen hier eine hohe Präzision wichtiger ist, als ein großes Gebiet.

***Discover*-Anfragen** Die Motivation für *Discover*-Anfragen ist, möglichst große unerforschte Gebiete zu ermitteln, deren Erkundung sich möglichst gut auf die Erkundungsrate auswirkt. Dies lässt sich direkt aus dem *HHH*-Modell ableiten, in dem die Heavy-Hitter für den Typ "nicht erkundet" berechnet werden. Große, zusammenhängende Gebiete sind hierbei zu bevorzugen, da diese die Erkundungsrate am meisten verbessern. Der negative Extremfall wäre zum Beispiel, wenn eine einzelne, nicht erforschte Zelle in einem erforschten Areal liegen würde. Dann würde weder der Weg dorthin, noch die Erkundung der einzelnen Zelle das Modell nennenswert verbessern. In der Matrix wird nach zusammenhängenden und vollständigen Gebieten gesucht. Die anfängliche Größe beträgt 4×4 Felder. Gefundene Gebiete werde aus der Matrix entfernt und für spätere Durchläufe nicht mehr berücksichtigt. Dies ermöglicht, dass falls die maximale Anzahl von Antworten noch nicht erreicht wurde, effizienter nach kleineren Gebieten gesucht werden kann. Denn ist die Suche nach zusammenhängenden 4×4 Felder abgeschlossen, wird in diesem Fall nach 2×2 Feldern gesucht und im extremen Fall anschließend einzelnen unerforschten Felder.

***Improve*-Anfragen** Bei diesem Anfragetyp ist das Ziel solche Gebiete zu erkennen, für die zwar bereits Messungen vorliegen, sie also nicht mehr vollkommen unerforscht sind, aber noch nicht vollständig erkundet wurden. Als Beispiel könnten zum Beispiel einige verstreute energiereiche Punkte auf den Rand eines noch nicht entdeckten Cluster hindeuten. Diese Anfragen sind die wichtigste Schnittstelle zwischen dem Mobilitäts- und Kommunikationsagenten und dem Data-Mining-Agenten, da sie sich maßgeblich von einer klassischen Erkundungsstrategie unterscheiden. Es werden bevorzugt Gebiete mit wenigen Nachbarn und solche mit höherer Energie zurückgegeben. Die Fälle sind:

- weniger als 3 Nachbarn $\rightarrow$ 4×4 Feld
- Zwischen 6 und 3 Nachbarn $\rightarrow$ 2×2 Feld
- die Sortierung erfolgt mithilfe eines Vergleichsoperators, der primär absteigend nach der Anzahl der Nachbarn sortiert und sekundär nach absteigend nach dem Energie vergleicht

Werden 4×4 Felder bei der *Improve*-Anfrage zurückgegeben, so werden diese so konstruiert, dass der nach den obigen Kriterien gefunden Punkt, zufällig einem der vier zentralen Punkte des Feldes entspricht, also etwa in der Mitte des Feldes liegt. Bei 2×2 Feldern wird der ausgewählte Punkte zufällig als einer der Eckpunkte ausgewählt.

Bei allen Rückgaben wird darauf geachtet, dass die Gebiete innerhalb der zulässigen Grenzen liegen. So wird verhindert, dass zum Beispiel ein 4×4 Gebiet um einen zu erkundenden Punkt konstruiert wird, welches über den Rand des Spielfeldes hinausragt. Der Mobilitäts- und Kommunikationsagent kann derzeit zwar mit solchen Wegpunkten umgehen, aber in anderen Szenarien könnte ein überschreiten der Grenze problematischer sein. Ragen die zurückgegebenen Gebiete über die Grenzen hinaus, so werden sie so verschoben, dass sie im zulässigen Gebiet liegen.

5.4 Diskussion der verwendeten Algorithmen

Die Implementierung verschiedener Verfahren zur Identifikation von energiereichen Gebieten bietet bereits viele Einsichten in die Problematik der gegebenen Aufgabenstellung. Das Sammeln von Daten und die Extraktion der relevanten Informationen innerhalb eines P2P-Netzes mit mobilen Peers erfordert Algorithmen, die auf Streaming-Daten und verteilten Daten arbeiten. Die Problematik liegt tatsächlich in der Zusammenführung dieser beiden Eigenschaften. So liefert die Erweiterung des verteilt arbeitenden *k*-Means-Algorithmus auf Daten-Streams keine zufriedenstellende Resultate, zumindest solange keine Datenpunkte gespeichert werden.

Das Clustern von aggregierten Daten in einer offline-Phase, wie z. B. bei den Micro-Clustern in DistrDBSCAN, wäre zwar auch hier denkbar, jedoch gibt es noch weitere Problematiken bzgl. der Anwendung des *k*-Means-Algorithmus, die eine Erforschung von Alternativen wenig aussichtsreich erscheinen lassen. Derartig ist die Bestimmung einer festen Anzahl von Clustern insbesondere bei Aufgabenstellungen, die eine Erkundung des Gebiets vorhersehen, geradezu unmöglich. Dynamische Verfahren zur Bestimmung der Clusteranzahl wie z. B. in *x-Means* [36] sind im verteilten *k*-Means-Algorithmus nicht anwendbar, da hier jeder Peer die Zentroide anderer Peers eindeutig zu den seinen zuordnen können muss. Das bedeutet, dass zumindest die Clusteranzahl für alle Peers die gleiche sein muss, was sich bei nicht zu erwartender Erreichbarkeit aller Peers untereinander schwierig gestaltet.

Als alternativer Ansatz wurde der dichtebasierte Algorithmus DistrDBSCAN betrachtet. Eine Abschätzung der Clusteranzahl ist hier nicht notwendig, die einzigen Parametereinstellungen, die vorgenommen werden müssen betreffen den Radius der Micro-Cluster und den Schwellwert $minPts$. Beide lassen sich größtenteils aufgrund der Anforderungen an das Verfahren bestimmen. So lässt sich der Radius anhand des gewünschten Detaillierungsgrads bzgl. der Clusterformen und minimalen Ausdehnung der Clustern festlegen. Zur Bestimmung von $minPts$ ist eine Abschätzung der Dichte innerhalb eines Clusters erforderlich, was sich im hier betrachteten Anwendungsfall aus der erwarteten Anzahl gemessener heller Datenpunkte bei einer Fahrt über die ϵ-Umgebung eines core-Punktes ergibt. Der Parameter mag sorgfältig gewählt werden, da abzuwägen ist, ob eine schnell zu erlangende Übersicht der zuverlässigen Clusterbildung vorzuziehen ist. Ist $minPts$ niedrig gewählt, so werden wenige, nahe liegende Punkte des gleichen Energielevels zu Clustern zusammengefasst. Das kann zu Beginn der Erkundung vorteilhaft sein, wenn innerhalb der ϵ-Umgebungen wenig bekannt ist und die spärlich vorhandenen Messpunkte ein einheitliches Energieniveau aufweisen müssen, um Cluster zu bilden. Ist die Erkundungsrate jedoch sehr hoch, so kann das wiederholte Messen innerhalb einer Umgebung zu einer ausreichend hohen Anzahl von gemessenen Rauschpunkten des gleichen Energielevels führen, die dann ggf. als Cluster interpretiert werden. Die Problematik macht deutlich, dass die Modellierung auch von der Erkundungsstrategie abhängt. Wenn sich die Fahrzeuge langsam auf der Karte vortasten, ist die Wahl von $minPts$ eher groß zu wählen, wohingegen eine großflä-

chige Erkundungsstrategie, in der weite Strecken gefahren werden, das Heruntersetzen des Parameters notwendig machen mag.

Die Klassifikation durch Heavy-Hitter auf verschiedenen Abstraktionsebenen ermöglicht neben der Erfassung heller Gebiete auch die Registrierung wenig erforschter Gebiete bzw. einer Erkundungsrate. Diese Informationen sind für die Steering-Agenten interessant, um die Erforschung der gesamten Karte effizienter gestalten zu können. Die Erkundungsrate kann aber auch im Zusammenhang mit dem dichtebasierten Clustering-Verfahren von Nutzen sein, um Noise-Punkte schneller als solche bestimmen zu können. Ein vereinzelter heller Punkt kann evtl. zu einem Cluster gehören, wenn das betreffende Gebiet noch nicht ausreichend erforscht ist, andernfalls kann jedoch angenommen werden, dass der Punkt ein Noise-Punkt ist und gelöscht werden kann.

Eine Implementierung des Rankingverfahrens wie in Abschnitt 5.1.3 beschrieben, sollte vor allem zur Modellierung einer zeitlichen Komponente dienen. Da diese aufgrund der umfangreich diskutierten Modellierung über kurze Zeitabschnitte nicht mehr realisiert werden konnte, fand dieser Algorithmus in dem gegebenen Rahmen keine Anwendung.

Zusammenfassend gesehen, bietet der DistrDBSCAN-Algorithmus die Möglichkeit, ein gutes Modell zu erstellen, um energiereiche Gebiete zu kartographieren, und durch die Noise-Punkte ist eine Vorauswahl interessanter Gebiete gegeben. Auf der anderen Seite liefern die gefundenen Heavy-Hitter einen guten Überblick über bereits erkundete Gebiete und durch die Hierarchie eine komprimierte Darstellung der Gebiete mit hoher bzw. niedriger Energiedichte.

6 Mobilitäts-Agent

In diesem Kapitel wird die Entwicklung der in der PG verwendeten Erkundungsstrategie beleuchtet. Zuerst werden dafür in Kapitel 6.1 die Grundlagen von Steerings und Teamstrategien erläutert. Dies ist wichtig, um ein Verständnis dafür zu bekommen, nach welchen Gesichtspunkten eine Strategie entwickelt wird. Zusätzlich wird danach in Kapitel 6.2 das *CNI UxV Framework*, welches auch die Basis für die spätere Simulation darstellt, beschrieben. Als Abschluss wird in Abschnitt 6.3, zunächst eine Analyse von bestehenden Steerings durchgeführt und danach die in der PG entwickelte Strategie erklärt.

6.1 Algorithmische Grundlagen

Die folgenden zwei Abschnitte sollen die Grundlagen für die Mobilität von autonomen Agenten vermitteln. Dabei geht es insbesondere um die kooperative Erkundung von Gebieten. Wichtige Grundlagen zu diesem Thema betreffen kooperative Navigation und Teamstrategien. Diese Themen zeigen auf, welche Strategien und Vorgehensweisen es gibt, um einen Schwarm von Robotern zu einem Ziel zu bewegen. Bei der kooperativen Navigation liegt der Fokus auf der Einhaltung von Bedingungen die ein Schwarm erfüllen muss um z.B. kohärent zu bleiben. Die Teamstrategien werden eingesetzt um ein gegebenes Missionsziel zu erreichen. Das kann die Erkundung eines Gebietes sein, oder das finden eines Weges zu einer bestimmten Koordinate. Durch das Verständnis der vorgestellten Strategien wird später ein eigener Algorithmus entwickelt der die Vor- und Nachteile der unterschiedliche Ansätze gegeneinander aufwiegt und für das geforderte Ziel den besten Kompromiss findet.

6.1.1 Kooperative Navigation

Um einen Schwarm von mehreren Robotern über einen Playground zu navigieren, bedarf es einer Strategie, die die Vorteile eines Teams ausnutzt. Aufgabe der kooperativen Navigation ist es, die einzelnen Teammitglieder so zu koordinieren, dass sie gemeinsam ein Ziel verfolgen und die Informationen der anderen Mitglieder nutzen. Zudem übernimmt die kooperative Navigation die Aufgabe, dynamisch auf Hindernisse zu reagieren, Kollisionen zu erkennen, bis hin zum geeigneten Verhalten, wenn ein Fahrzeug ausfällt. Der Unterschied zu den Teamstrategien ist, dass diese das sogenannte Macro-Steering planen, also wie kommt der Schwarm vom aktuellem Standpunkt zum Missionsziel. Die kooperative Navigation übernimmt das Micro-Steering, also die Bewegung innerhalb des Schwarms.

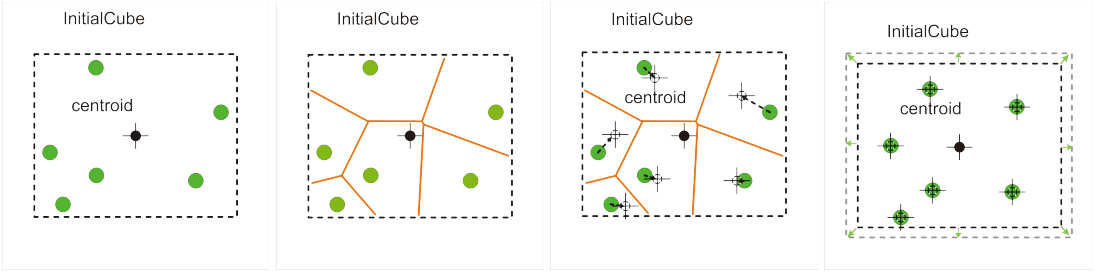
Ein Schwarm bewegt sich in einer Umgebung. Hier unterscheidet man zwischen einer statischen Umgebung und einer dynamischen Umgebung. Bei einer statischen Umgebung sind alle Komponenten der Umgebung zu jedem Zeitpunkt gleich und nur die Positionen der Roboter ändern sich. Da das System am Ende der PG demonstriert werden soll, ist von einer dynamischen Umgebung auszugehen. Eine dynamische Umgebung ist zu jedem Zeitpunkt anders, d.h. es können Fahrzeuge hinzukommen oder ausfallen. Zudem ändern sich im Lauf des Tages die Positionen der Energiequellen durch den Lauf der Sonne. Natürlich wirken auch andere Faktoren auf die Umgebung ein, wie z.B. das Auftreten von Wolken, die für eine Abschattung sorgen.

Es gibt grundlegende Anforderungen an einen Navigationsalgorithmus. Um den Vorteil des Schwarms zu nutzen, muss der Algorithmus dafür sorgen, dass der Schwarm kohärent bleibt. Würde der Schwarm getrennt werden, könnte jeder sogenannte Cluster das Gebiet zwar alleine erkunden, allerdings würde kein Austausch über bereits erkundete Gebiete erfolgen. Andererseits sollte die Erkundung möglichst effizient sein. Das heißt der Algorithmus muss einen Kompromiss zwischen hoher Erkundung und guter Kommunikation treffen. Zudem müssen die erforderlichen Kommunikationsanforderungen eingehalten werden. Durch diese Einhaltung von z.B. einer bestimmten Datenrate oder einem RSSI (Received Signal Strength Indicator) wird sichergestellt, dass eine gute Kommunikation innerhalb des Schwarms stattfindet.

Ein klassischer Ansatz, um einen Schwarm über den Playground zu bewegen, ist der „Leader-Follower“ Ansatz. Dabei übernimmt ein Roboter die Leader-Rolle und die anderen Teilnehmer (Follower) folgen ihm. Der Leader plant nun mit Hilfe der Teamstrategien den Pfad über den Playground und die Follower folgen ihm in einer bestimmten Formation. Die Formation ist dabei abhängig von der zu bewältigenden Aufgabe. Bei diesem Ansatz kristallisieren sich schnell Punkte heraus, die für unser Vorhaben von Nachteil sind. So hängt die Erkundungsrate stark von der gefahrenen Formation ab, zudem stellen sich Probleme ein, sollte einer der Follower ausfallen. Der Schwarm müsste dann neu formiert werden. Der schlimmere Fall wäre, wenn der Leader ausfallen würde, dann müssten die Teamstrategien komplett neu geplant werden. [38]

Um einen flexibleren Ansatz zu erhalten, wird ein agentenbasierter Schwarm verwendet. Bei einem agentenbasiertem Schwarm sind alle Teilnehmer gleich und es gibt keine direkte Rollenverteilung. Während der Erkundung, ist es sinnvoll, dass jeder Roboter möglichst gleichviel Fläche hat, die er erkunden soll. Dazu müssen die Roboter gleichmässig auf dem Playground verteilt werden. Ein möglicher Ansatz um die Roboter gleichmäßig auf dem Playground zu verteilen, ist die Voronoi-Zerlegung ergänzt um eine Lloyd-Bewegung.

Voronoi-Zerlegung und Lloyd-Bewegung Die Voronoi-Zerlegung teilt den Raum zuerst anhand von Robotern so, dass jeder Punkt innerhalb einer Voronoi-Region näher zum aktuellem Roboter als zu jedem anderen ist. Nun wird die aktuelle Position des Roboters mit der Lloyd-Bewegung angepasst. Dazu fährt der Roboter von seiner aktuellen Position zum Zentroid seiner Voronoi-Region. Wird das Verfahren der Zerlegung und Bewegung wiederholt, passt sich die Größe der Gebiete immer mehr an und der



(a) Ausgangssituation (b) Erfolgt Zerlegung (c) Lloyd-Iteration (d) Smart Cube

Abbildung 6.1: Voronoi-Zerlegung mit Lloyd-Bewegung [7]

Roboter befindet sich im Mittelpunkt der Region. Ein Beispiel für diese Iteration ist in Abbildung 6.1a-d zu sehen.

SmartCube Dieses Verfahren hat jedoch ein Problem. Die Zerlegung ist nicht kommunikationssensitiv. D.h. die Voronoi-Regionen können so groß werden, dass keine Kommunikation mehr zwischen den Robotern stattfinden kann. Um dieses Problem zu lösen, wird um den kompletten Schwarm ein sogenannter SmartCube gelegt. Bei dem SmartCube handelt es sich um einen fiktiven Bereich, dessen Größe abhängig von einer bestimmten Metrik ist. Diese Metrik könnte z.B. ein bestimmter RSSI-Wert sein, der innerhalb des SmartCubes immer gewährleistet ist. Das Verfahren der Voronoi-Zerlegung und der Lloyd-Bewegung wird jetzt innerhalb des SmartCubes angewandt und somit ist sichergestellt, dass die Roboter untereinander kommunizieren können.[14] [7]

Potentialfelder Ein weiteres Verfahren, mit dessen Hilfe sich Roboter durch ein Gebiet navigieren lassen, sind die Potentialfelder. Bei diesem Ansatz werden den Punkten, die von Interesse sind, eine anziehende Kraft zu geordnet und die Bereiche, die für den Roboter hinderlich sind, erhalten ein abstoßendes Potential. Zu beachten ist, dass innerhalb des Schwarms ein Roboter ebenfalls ein anziehendes, bzw. abstoßendes Potential auf andere Roboter ausübt. Damit lassen sich Kommunikationsbedingungen mit Potentialfeldern abbilden.

Um eine Navigation mittels Potentialfeldern durchzuführen, gibt es zwei Bereiche, die kombiniert werden müssen. Einerseits wird der einzelne Roboter von seinem Missionsziel angezogen, andererseits befindet er sich in einem Schwarm und muss dessen Mitglieder als Hindernisse bzw. Zwischenziel berücksichtigen. Die Navigation zum Missionsziel ist einfach, der Roboter muss sich lediglich vom Potentialfeld anziehen lassen. Komplizierter ist allerdings die Navigation innerhalb des Schwarms. Abbildung 6.2 zeigt ein mögliches Szenario, um einen Schwarm mittels Potentialfeldern kohärent zu halten.[22]

Die hier gewählte Kommunikationsbedingung ist, dass der RSSI zwischen den Robotern im Bereich -75dBm und -80dBm liegen soll. Wird dieser Wert wie zwischen UGV2

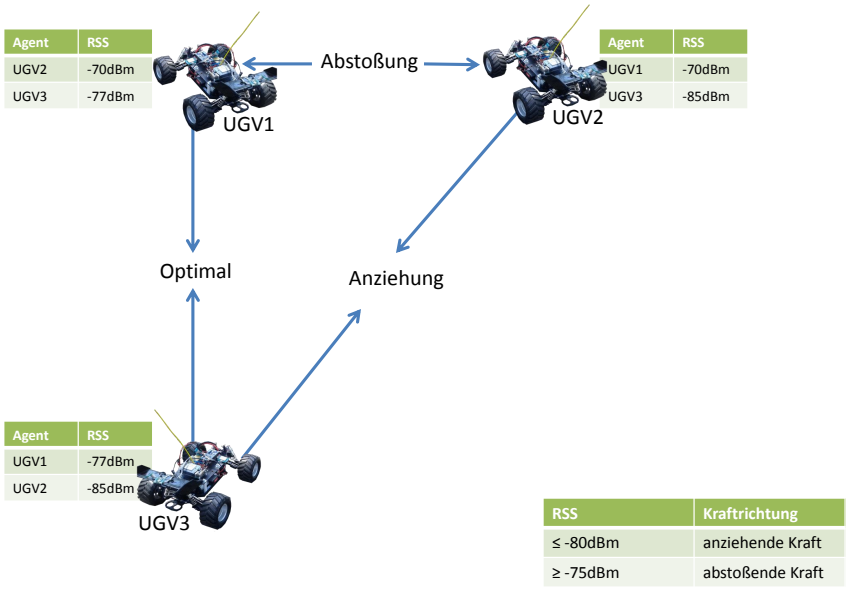


Abbildung 6.2: Potentialfelder zur Aufrechterhaltung der Kommunikationsbedingungen.

und UGV3 überschritten, hier beträgt die RSS -85 dBm, so herrscht zwischen den beiden Robotern ein anziehendes Potential. Wird der Wert wie zwischen UGV1 und UGV2 überschritten, hier beträgt die RSS -70dBm, so herrscht zwischen den beiden Robotern ein abstoßendes Potential. Diese Zustände müssen bei der Bewegung des Schwarms berücksichtigt werden. Um den Schwarm nun über den Playground zu navigieren, werden die Vektoren, die sich aus dem Missionsziel ergeben, mit den Vektoren, die die Kommunikation innerhalb des Schwarms sicherstellen, addiert. Aus diesem neuen Vektor ergibt sich der finale Richtungsvektor, welcher vom Roboter abgefahren wird. Abbildung 6.3 zeigt ein mögliches Szenario. Der blaue Vektor gibt die Richtung zum Missionsziel an; der grüne Vektor die Richtung um die Kommunikation aufrecht zu erhalten. Der rote Vektor zeigt die Addition der beiden Vektoren an und somit die Fahrtrichtung des Roboters. Diese Bedingungen werden ständig überwacht und auf ihre Gültigkeit überprüft.

Ein Hauptproblem der Potentialfelder sind die lokalen Minima oder auch U-Traps genannt. Dabei wird ein Roboter mit gleicher Kraft von einem Ziel angezogen wie von einem Hindernis abgestoßen. Fährt der Roboter in so eine Falle, steht er still. Aufgabe der Navigation ist es, solche Probleme zu erkennen und passende Maßnahmen zu treffen.

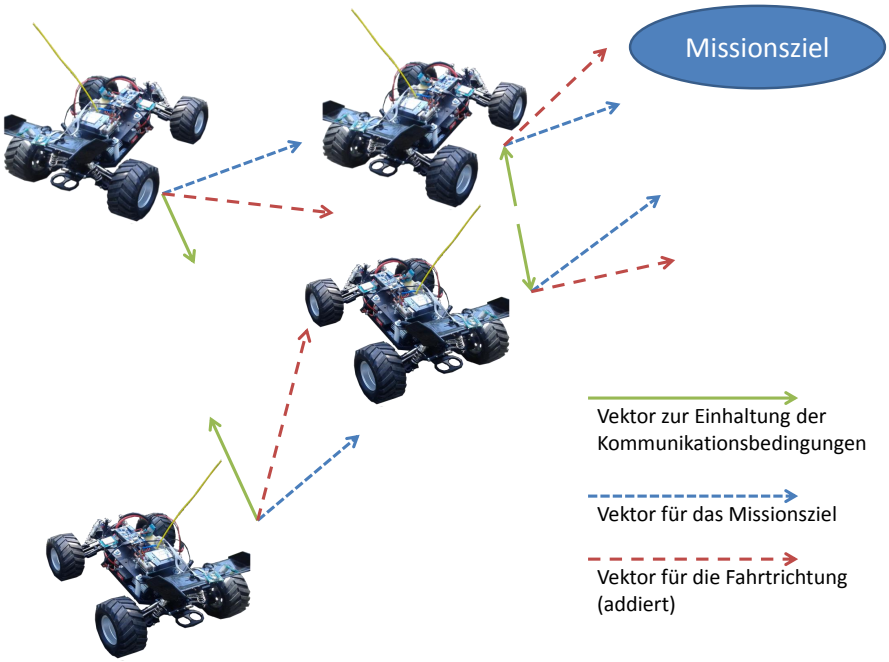


Abbildung 6.3: Addition von Richtungsvektoren für eine Navigation mit Potentialfeldern.

6.1.2 Teamstrategien

In diesem Abschnitt geht es um Teamstrategien und warum sie eingesetzt werden sollten. Allgemein ist eine Teamstrategie eine Vorgehensweise eines Teams um eine entsprechende Aufgabe zu lösen. In diesem Fall könnte dies entsprechend die Erkundung eines Gebietes oder das Sammeln von Sensordaten sein. Um nun eine passende Strategie zu entwickeln, muss auf die entsprechenden einzuhaltenden Parameter geachtet werden. Hier können viele Faktoren einfließen wie zum Beispiel die zur Verfügung stehende Zeit, Bedingungen wie Verfügbarkeit der Kommunikation oder die Datenrate, aber auch das generelle Ziel der Mission. Zusätzlich können sich viele Parameter während der Mission ändern, da sich entsprechend die Umgebung ändert oder ein Roboter ausfällt.

Um einen Überblick über verschiedene Strategien zu bekommen, werden im Folgenden mehrere Ansätze von Strategien vorgestellt. Ein einfaches Beispiel aus der Informatik ist der Divide-and-Conquer Ansatz. Hierbei wird das Problem (zum Beispiel Erkunden eines Gebiets) in kleinere Teile aufgespalten und diese entsprechend gelöst. In Abbildung 6.4 ist dies für das Erkunden eines Gebiets beispielhaft dargestellt. Die Kreise bzw. Dreiecke stellen Roboter dar, die sich in einem zu erkundenden Gebiet aufhalten. Das Gebiet wird nun in verschiedene Bereiche aufgeteilt, welche nun von jeweils einem Roboter erkundet werden sollen.

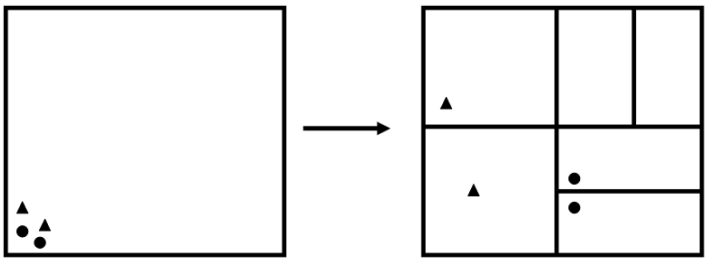


Abbildung 6.4: Divide-and-Conquer Ansatz

Multi Robot Exploration Als nächstes wird die Multi Robot Exploration [10] beschrieben . Dies ist ein Algorithmus bei dem ein Team von Robotern ein Gebiet erkunden soll. Jeder Roboter hat hierbei eine Grid Map der Umgebung, in welcher eingetragen ist welche Gebiete erkundet sind und welche entsprechend nicht. Wichtig für diesen Algorithmus sind die so genannten Grenzzellen. Dies sind unerkundete Zellen im Grid, welche neben erkundeten Zellen liegen.

Algorithmus 2 beschreibt die vier Phasen, welche der Algorithmus durchläuft.

Algorithmus 2 Multi Robot Exploration Algorithmus

1. Die Grenzzellen werden bestimmt und ihnen wird ein Wert von 1 zugewiesen.
2. Für jeden Roboter wird für jede Grenzzelle berechnet, wie lange er brauchen würde diese zu erreichen, wobei hier der Wert in der Zelle miteingeht.
3. Die beste Kombination von einem Roboter und einer Grenzzelle wird berechnet und diesem Roboter wird die entsprechende Zelle zugewiesen. Um nun zu verhindern, das ein weitere Roboter die gleiche Zelle besuchen würde, wird der Wert in der Zelle und den umgebenden Zellen gesenkt.
4. Phase 3 solange wiederholen bis jeder Roboter eine Zelle zugewiesen bekommen hat.

Ein wichtiger Bestandteil des Algorithmus ist die Absprache zwischen den Robotern. Diese wird in der Abbildung 6.5 verdeutlicht. In der Abbildung sind zwei Bilder in denen jeweils zwei Roboter zu sehen sind, welche gemeinsam das besuchte Gebiet erkunden sollen. Während dem oberen Bild keine Absprache zwischen den Robotern stattfindet, das heißt der Wert in den Zellen wird nicht verringert wenn ein Roboter diese besucht, findet dies dem unteren Bild statt. Es ist deutlich zu erkennen, dass im oberen Teil durch die fehlende Absprache beide Roboter als nächstes das selbe Gebiet erkunden wollen, während im unteren Teil beide Roboter unterschiedliche Gebiete anfahren. Wenn in dem Szenario eingeschränkte Kommunikation herrscht, kann der Ansatz auf Sub-Teams angewandt werden die miteinander kommunizieren können. Theoretisch

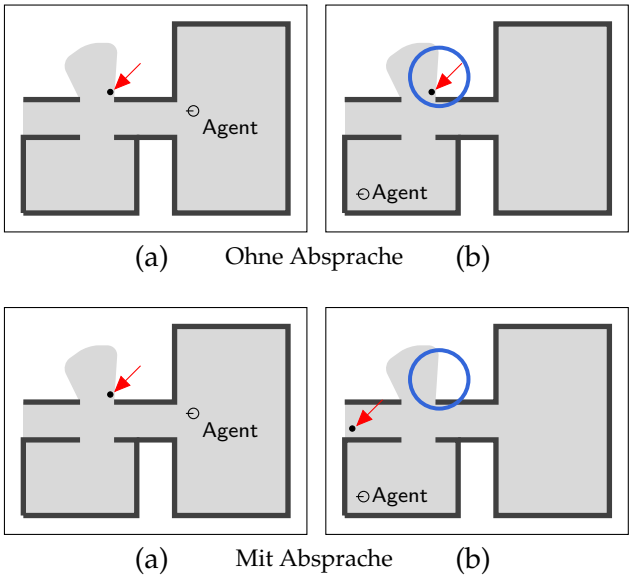


Abbildung 6.5: Auswahl des nächsten Wegpunkts ohne und mit Absprache der Roboter (a) und (b) [10]

kann es hierdurch passieren, das jeder Roboter die komplette Karte erkundet, jedoch ist dies praktisch meist nicht der Fall.

Eine gute Ergänzung zum Algorithmus ist, dass sich die Roboter merken welche Grenzzellen die anderen Roboter entsprechend anfahren. Hierdurch wird verhindert, dass viele Stellen der Karte mehrfach erkundet werden.

Multi Task Allocation for UAVs Die zweite Strategie, die vorgestellt wird, ist die sogenannte Multi Task Allocation for UAVs [8]. Hierbei werden verschiedene Ziele auf die Roboter verteilt, zum Beispiel Gasmessungen oder Fotoaufnahmen. Als Parameter werden die Start und Wegpunkte, sowie Geschwindigkeit, No-Fly Zonen und UAV Ausstattung gebraucht. Die No-Fly Zonen sind Gebiete, welche nicht betreten werden dürfen und die UAV Ausstattung bestimmt welche Wegpunkte angeflogen werden können, da z. B. für eine Gasmessung ein Gassensor vorhanden sein muss. Der Algorithmus nutzt eine Kostenfunktion für jedes UAV und eine für das gesamte Team. In Abbildung 6.6 ist der Ablauf des Algorithmus dargestellt. Zuerst werden für jedes UAV alle Wegpunktkombinationen gebildet, welche das UAV besuchen kann. Danach wird für jede dieser Kombinationen eine kürzeste Strecke mit Geraden gebildet. Im nächsten Schritt wird über die Kostenfunktion eine bestmögliche Kombination der Wegpunkte auf die UAVs gefunden. Nachdem das geschehen ist, wird eine detaillierte Wegplanung durchgeführt, da die Geraden oft nicht so abgeflogen werden können. Im letzten Schritt wird für die detaillierten Wege überprüft, ob hierbei eine Kollision entstehen kann, und falls ja wird umgeplant. Hierbei kann es passieren, das die ausgewählte Lösung durch

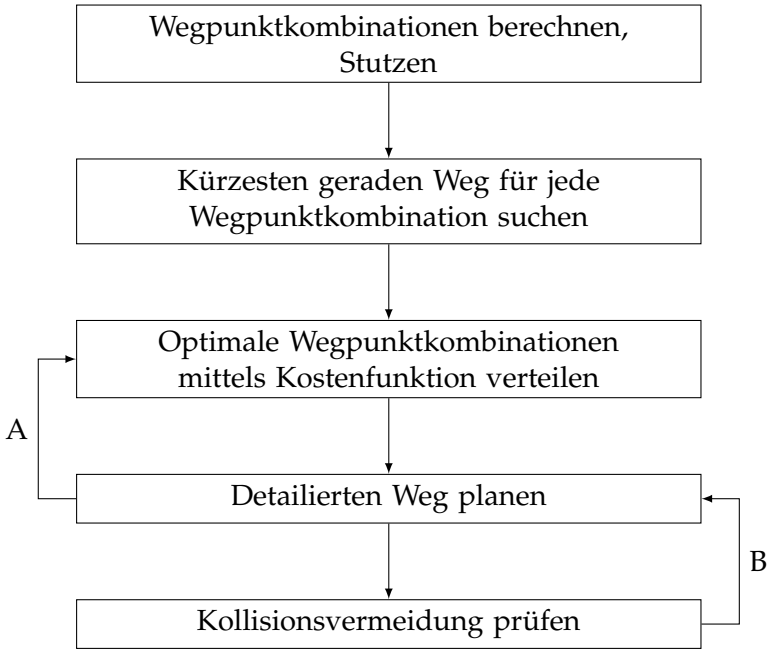


Abbildung 6.6: Ablauf der Multi Task Allocation for UAVs [8]

die detaillierte Wegplanung nicht die optimale Lösung darstellt und es kann eine andere Lösung genutzt werden. Es ist zu beachten, das die Bestimmung der detaillierten und der geraden Wege von jedem Agenten selbst berechnet werden kann, also eine Verteilung der Last möglich ist.

In Abbildung 6.7 ist ein Beispiel für eine Mission dargestellt. Auf der linken Seite der Karte sind sechs UAVs, welche unterschiedliche Formen besitzen. Die unterschiedlichen Formen, welche in der linken oberen Ecke erklärt werden, geben an, welche der Wegpunkte von diesem UAV angeflogen werden können. Auf der rechten Seite der Karte sind in Blau die No-Fly Zonen eingetragen und die entsprechenden Wegpunkte. Die Pfade der UAVs sind mit roten Linien gekennzeichnet. Hierbei ist gut zu erkennen, das es kein einfaches Problem ist die UAVs zu verteilen, besonders da unterschiedliche Typen von UAVs vorhanden sind.

Relaying Die letzte Strategie, die vorgestellt wird, bezieht sich auf das Relaying [21]. Hierbei wird durch eine Rollenverteilung der Roboter eine Verbindung zu einer Basisstation aufrechterhalten, indem bestimmte Rollen dazu da sind das Signal weiterzuleiten und somit die Reichweite der anderen Roboter zu erhöhen. Jeder Roboter kann eine von drei verschiedenen Rollen einnehmen: Scout Agent, Relay Node, Articulation Point. Der Scout Agent ist für die Erkundung des Gebietes verantwortlich, während der Relay Node eine direkte Verbindung zur Basisstation besitzt. Der Articulation Point stellt die Verbindung zwischen Relay Node und den restlichen Robotern her. Deswei-

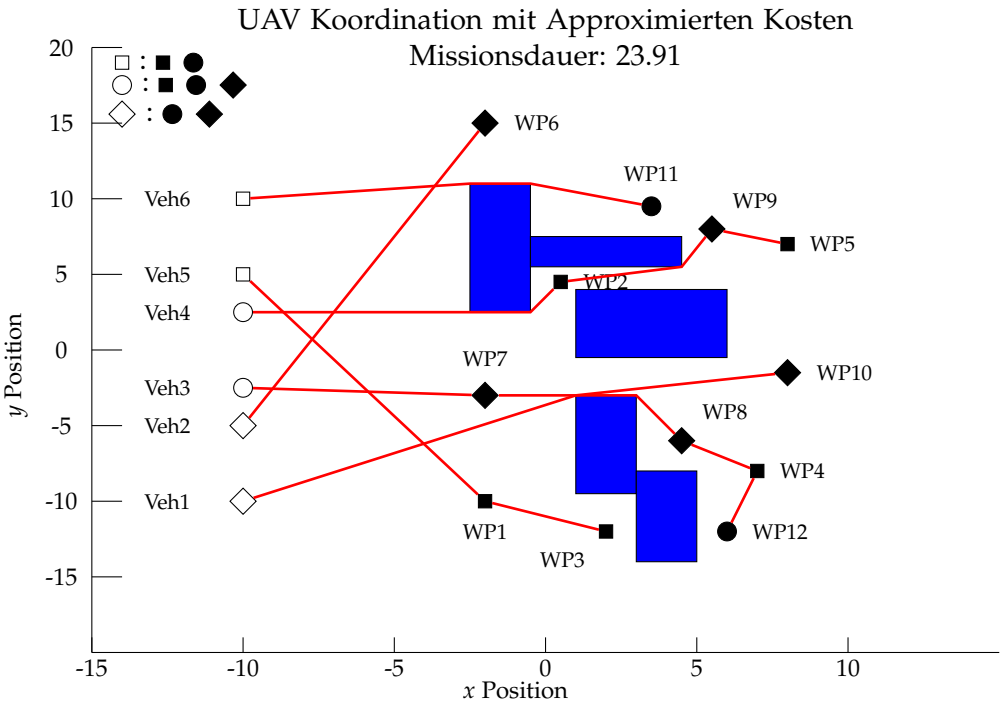


Abbildung 6.7: Multi Task Allocation Anwendung [8]

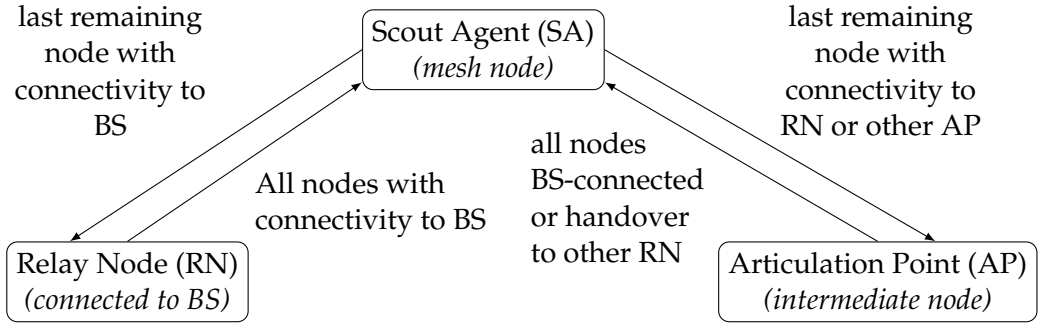


Abbildung 6.8: Rollenveränderung beim Relaying [21]

teren ist jedem Roboter das Netz bekannt und sie entscheiden selbst, welche Rolle sie einnehmen.

In Abbildung 6.8 wird dargestellt, wie die Roboter die unterschiedlichen Rollen wählen. Zuerst starten alle in der Scout Agent Rolle und falls die Reichweite der Basisstation verlassen wird, wird der letzte Roboter der noch eine Verbindung hat, zum Relay Node. Der Roboter wechselt erst wieder in die Rolle des Scout Agents, wenn alle anderen Roboter wieder eine Verbindung zur Basisstation haben. Die Rolle des Articulation Point nimmt ein Roboter ein, falls die Verbindung zu dem Relay Node oder einem anderen Articulation Point abzubrechen droht. Der Wechsel zur Rolle des Scout Agent wird vollzogen, wenn alle anderen Roboter wieder eine Verbindung zum Relay Node besitzen.

Abschließend lässt sich für das Projekt schlussfolgern, das eine Strategie ein wichtiger Bestandteil dessen ist, da nur so effizient ein Gebiet erkundet werden kann. Hierbei ist zu beachten, dass es schon viele Strategien gibt, welche für das Anwendungsgebiet des Projektes analysiert werden sollten, um damit eine möglichst effiziente Strategie für das Projekt zu finden.

6.2 Missions- und Kommunikationsframework *CNI UxV Framework*

Das *CNI UxV Framework* wird bereits als Steuer- und Kontrollsoftware für UAVs verwendet und besteht aus zwei wesentlichen Bestandteilen: Die Bodenkontrolleinheit und den Mobilitäts- und Kommunikations-Agenten für das UAV. Im Kontext der PG wird bisher ausschließlich letzteres verwendet und angepasst, daher wird auf die Bodenkontrolleinheit nicht weiter eingegangen.

Das *CNI UxV Framework* wird als Basis ausgewählt, da es bereits mit der aus Abschnitt 4.4 bekannten Autopilotsoftware kommunizieren kann, umfangreiche Loggingmöglichkeiten bietet und für Zielszenario-ähnliche Einsatzgebiete bereits verwendet wird.

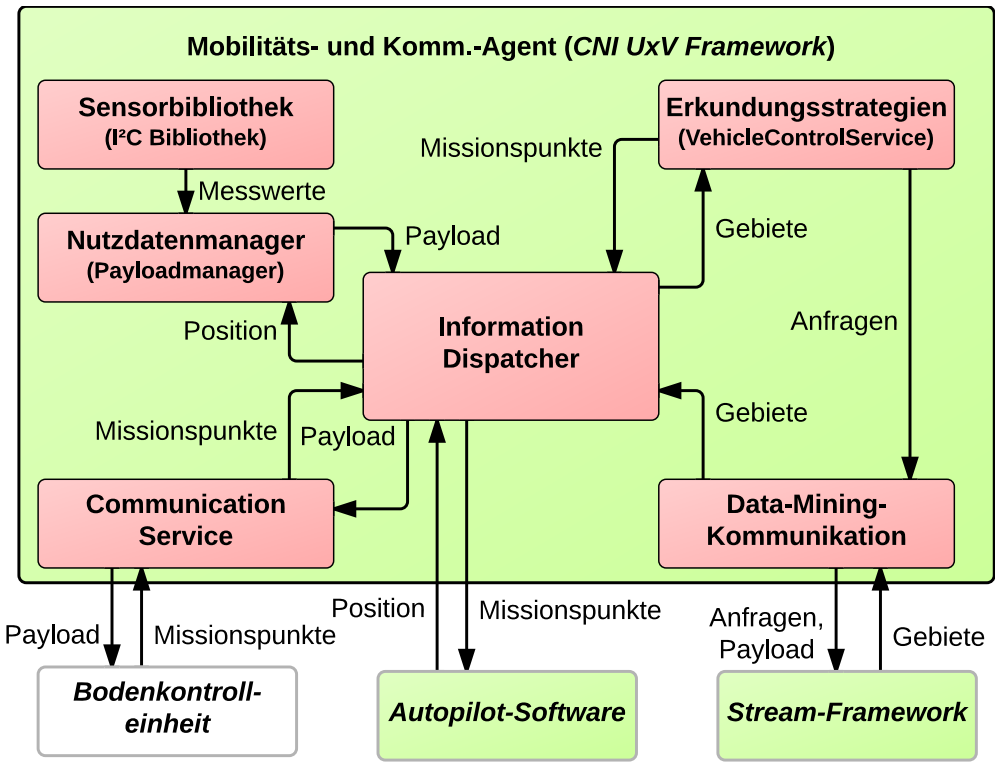


Abbildung 6.9: Grundlegende Architektur des *CNI UxV Framework*

Abbildung 6.9 zeigt einen Ausschnitt der Komponentenstruktur aller genutzten Teilkomponenten und die wesentlichen Kommunikationswege. Über den *Communication Service* ist eine Inter-Agent Kommunikation umgesetzt. Jedem Agenten sind alle seine Nachbarn, durch Auswertung der Funkstärke, bekannt. Durch eine, in der Übersicht nicht vertretenden, *CommLib* bietet das Framework Möglichkeiten wie UDP und TCP Verbindungen herzustellen um einen Kommunikationskanal zu anderen Agenten aufzubauen. Ein *Payloadmanager* sammelt Sensordaten und propagiert diese durch den *Information Dispatcher* an die Bodenkontrolleinheit. Umgekehrt werden von dort die GPS Daten an den *Payloadmanager* übertragen, da der *Information Dispatcher* mit dem Autopiloten verbunden ist. An diesen werden auch die Wegpunkte geschickt, welche die Steerings (vgl. Abschnitt 6.3.2) im *Flight Control Service* generieren.

Einige Anpassungen sind für den neuen Einsatzzweck durchgeführt worden, diese werden in den folgenden Abschnitten thematisiert.

6.2.1 MavLink

Für die Kommunikation zwischen dem *CNI UxV Framework* und der Autopilotsoftware wird das MavLink Protokoll verwendet. Das Protokoll wurde ursprünglich von QGroundControl[1] für die effiziente Übertragung von C-Structs über serielle Verbindungen vom *Vehicle* zu einer Bodenstation entwickelt. Das *CNI UxV Framework* implementierte bereits Version 0.9. Eine neuere Autopilotsoftware erforderte jedoch Version 1.0 und entsprechend die Umstellung auf diese neue Version.

6.2.2 Einbindung von zusätzlichen externen Bibiotheken

Einbindung von pugixml In Abschnitt 7.2 wird auf die Protokollauswahl eingegangen. Das grundlegende Protokolformat ist XML für die Kommunikation zwischen Data-Mining-Agent und *CNI UxV Framework*. Obwohl das Generieren von XML durch einfache Stringkonkatenation erfolgen kann, fiel für das Parsen von XML die Wahl auf eine externe Bibliothek. pugixml wird diese Funktionalität leisten und ist als neue Abhängigkeit im *CNI UxV Framework*.

Einbindung von Sensor I^2C Bibliothek Die im Rahmen dieser Arbeit entwickelte I^2C Bibliothek aus Abschnitt 6.2.4 zur Abfrage von Sensorwerten wurde ebenfalls als neue Abhängigkeit für das Framework definiert.

6.2.3 PG-Erweiterungsmodul

Um die in Abschnitt 4.1.2 und 4.1.3 beschriebenen Sensoren auszulesen und die gesammelten Daten bereitzustellen, ist ein eigenes Modul für das *CNI UxV Framework* erstellt worden. Das Modul wird außerdem für die Simulation verwendet.

[1] `http://qgroundcontrol.org`

Simulations- oder Sensordaten Ob die Positions- und Energiewertdaten von den tatsächlichen Sensoren oder für eine Software-in-the-loop Simulation aus eingelesenen Datensätzen stammen, bestimmt sich durch das Vorhandensein einer *eval.run*-Datei. Ist diese Datei vorhanden, wird von einer Simulation ausgegangen und das in der Datei referenzierte Simulationsdatum wird eingelesen.

Datenerfassung Das *CNI UxV Framework* verwendet keinen *Event*-basierten Ansatz für das Auslösen von Berechnungen oder den Aufruf von Funktionen, sondern einen Zeit-basierten. Jede Komponente innerhalb des Frameworks enthält eine *update*-Methode. Das PG-Modul wird hierbei im Intervall von 10ms aufgerufen. Um unnötigen Datenoverhead und Rechenressourcen zu sparen, finden alle Berechnungen, Messwertabfragen und Ausgaben nur dann statt, wenn sich die Position des Rovers geändert hat. In der Methode werden die aktuelle Position, sowohl im Universal Transverse Mercator (UTM)[2], als auch im World Geodetic System 1984 (WGS 84)[3] Koordinatensystem und die Satellitenanzahl über die *Information Dispatcher*-Komponente angefordert. Die Temperatur- und Lichtsensorwerte, Zeitstempel und Positionsdaten als Variablensammlung bilden zusammen ein Tupel γ.

Ausgabe Neben der Logausgabe wird die *CommLib*-Komponente des *CNI UxV Framework* verwendet, um das Tupel γ dem Data-Mining-Agent über einen Kommunikationskanal (vgl. Abschnitt 7.2) bereit zu stellen. Die Steeringalgorithmen können über eine API auf die aktuellen Positionsdaten, sowie auf die vom Data-Mining-Agent bereitgestellten Positionsvorschläge zugreifen.

Optionale Datenaufzeichnung Zusätzliche Auswertungsmöglichkeiten bieten die optionalen zwei Aufzeichnungsmöglichkeiten für die periodisch generierten Tupel γ. In die *Comma Separated Value*-Datei (CSV) wird ein γ-Tupel pro Zeile abgelegt, wobei die einzelnen Elemente mit Semikola getrennt sind. Diese Dateien können als Testeingaben für die Data-Mining-Algorithmen statt der in Abschnitt 5.3.1 beschriebenen Methode verwendet werden. In der *GPS Exchange Format*-Datei (GPX) werden die unterstützten Elemente jedes γ-Tupel in einem XML Format als Wegpunktliste („Trackpoint“) abgelegt. Dieses Format kann von diversen Internetseiten[4] als Eingabe verwendet werden, um eine Karte mit den Wegpunkten zu generieren. Wie in Abbildung 6.10 abgebildet, kann so eine gefahrene Route anschaulich nachvollzogen werden.

6.2.4 Sensoranbindung durch die I²C-Bibliothek

Um die ausgewählten I²C-Sensoren anzusteuern konnte das standardisierte Linuxkernelsubsystem für I²C genutzt werden. Dieses erlaubt das Initialisieren des I²C-An-

[2] Über transversalen Mercator-Projektion verebnete, im kartesischen Koordinatensystem angegebene Koordinaten

[3] Grundlage für Positionsangaben auf der Erde und im erdnahen Weltraum

[4] GPX Eingabe zum Beispiel auf `http://www.gpswandern.de/gpxviewer/gpxviewer.shtml`

Abbildung 6.10: Visualisierung der vom Rover gefahrenen Route durch GPX-Datei

schlusses, das Senden von Bytes an ein Busgerät und das Lesen von Bytes von einem Busgerät. Die Funktionen sind jeweils synchron. Der Kontrollfluss kehrt als erst nach der Kommunikation mit dem Busgerät zurück. Für das Lesen und Schreiben muss jeweils die 7 Bit Adresse des Zielgeräts angegeben werden. Die Kombination des Schreib- und Lesebefehls erlaubt auch festzustellen, ob an der angegeben Adresse ein Gerät angeschlossen ist. Die Ansteuerung der benutzten Energiewertsensoren wird zwecks leichter Erweiterbarkeit durch eine eigene Bibliothek abstrahiert.

Bibliotheksarchitektur In Abbildung 6.11 ist die Grundstruktur der Bibliothek ersichtlich. Die Hardwaresensoren werden über den I^2C-Bus regelmäßig in einem Intervall *intervalInMS* im Millisekundenbereich abgefragt, die erhaltenen Sensorwerte aufbereitet und in Variablen hinterlegt. Sobald eine externe Abfrage über das einfach gehaltene Interface beispielsweise durch das Framework erfolgt, werden die zwischengespeicherten Werte zurückgegeben. Damit ist klar, dass die Sensorwerte bis zu *intervalInMS* Millisekunden alt sein können. Das ist in der Praxis durch die Trägheit der Sensoren jedoch vernachlässigbar und ermöglicht das direkte zurückgeben von Daten statt bei einem Aufruf erst auf den Sensor zu warten. Das Interface erlaubt die Abfra-

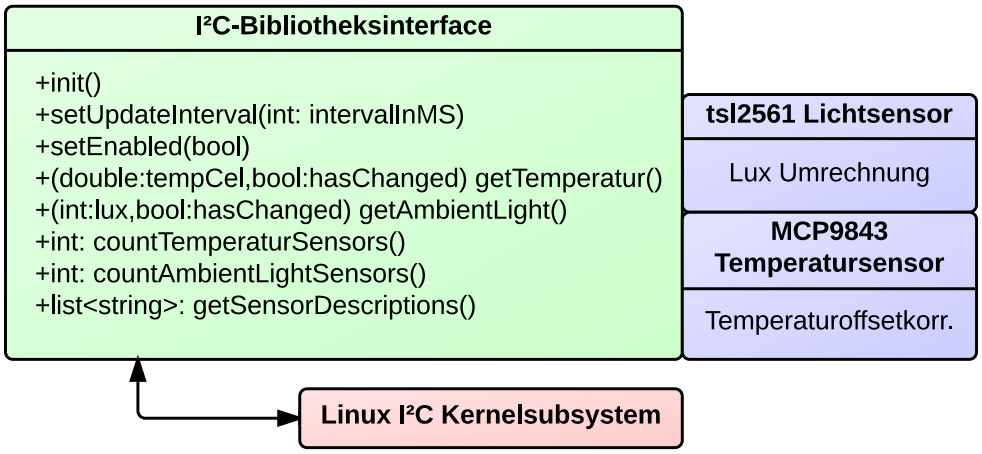


Abbildung 6.11: Grundlegende Architektur der I²C-Bibliothek

ge der Anzahl angeschlossener Temperatur- und Lichtsensoren, sowie die Abfrage der Typen und die Ausgabe eines Temperatur- und eines Lichtwertes mit der zusätzlichen Information, ob seit der letzten Abfrage neue Sensorwerte eingegangen sind.

Plug & Play Die regelmäßige Abfrage des Busses ermöglicht auch die Implementierung von Plug & Play, sprich im laufenden Betrieb angesteckte Sensoren werden erkannt und genutzt.

Mehrere Sensoren Falls mehrere Sensoren eines Typs (Licht, Temperatur) angeschlossen sind, wird der Mittelwert gebildet.

6.3 Strategieentwicklung

Folgender Abschnitt beschreibt die Entwicklung einer Strategie zur Erkundung eines Gebietes für die Roboter. Wie in der Einleitung beschrieben, sollen Roboter ein Gebiet nach hohen Energiewerten absuchen. Um die Energiewerte möglichst schnell zu detektieren, sind intelligente, kooperative Suchverfahren unerlässlich. Zur effektiven Entwicklung eines solchen Verfahrens werden zuerst schon vorhandene Konzepte evaluiert, um dann darauf aufbauend eine innovative Strategie zu entwickeln. Im Abschnitt 6.3.1 werden die von uns analysierten Steerings vorgestellt und evaluiert und in Abschnitt 6.3.2 wird die daraus entstandene Strategie *SUN Search* genau vorgestellt. Abschließend wird in Abschnitt 6.3.3 noch auf ein Problem eingegangen, welches bei der Portierung der Steerings von der Matlab-Entwicklungsumgebung auf die Zielhardware aufgetreten ist.

6.3.1 Erläuterung und Auswertung der Steerings

In diesem Abschnitt werden die getesteten Steerings beschrieben und evaluiert. Mithilfe der Steerings werden die Roboter bewegt, wobei auf unterschiedliche Parameter eingegangen werden kann. Diese können unter anderem der RSSI (Received Signal Strength Indicator) zu anderen Robotern, die Entfernung zum Ziel, die Anzahl der bestehenden Verbindungen oder die Entfernung zu anderen Robotern sein. Durch die Einbeziehung solcher Parameter können sich die Wege oder Ziele der Roboter ändern, da zum Beispiel nur durch einen Umweg die Kommunikation zu allen aufrecht gehalten werden kann, was durch einen Parameter ausgedrückt wird (RSS). Je nach Missionsanforderung muss also ein entsprechendes Steering gewählt werden.

Random-Walk-2D Der Random-Walk-2D erzeugt für jeden Roboter einen zufälligen Richtungsvektor, anhand dessen sich der Roboter bewegt. Hierbei werden keine Metriken beachtet, die zum Beispiel zwischen den Robotern existieren könnten. Es macht also keinen Unterschied, ob ein Roboter durch eine Bewegung eine Zelle erreicht, die dieser schon einmal besucht hat, oder ob der Roboter dadurch die Verbindung zu anderen Robotern verliert. Das neue Ziel wird gleichverteilt aus den umliegenden Zellen gezogen.

Self-Repelling-Walk Beim Self-Repelling-Walk wird für jeden Roboter ein neuer Zielpunkt in der direkten Umgebung in Abhängigkeit der schon von ihm besuchten Zellen berechnet. Hierzu wird für einen Roboter für die ihn umgebenden Zellen abgefragt, ob er diese schon besucht hat oder nicht. Dafür bekommt jede Nachbarzelle einen Zähler mit einer Zufallswahrscheinlichkeit zugeordnet. Falls der Roboter eine Zelle noch nicht besucht hat, steigt der Zähler der Zelle, sonst nicht. Am Ende wird die Zelle ausgewählt, welche den höchsten Zähler aufweist.

Coop-Repelling-Walk Der Coop-Repelling-Walk erzeugt für jeden Roboter einen neuen Zielpunkt in der direkten Umgebung, wobei die besuchten Zellen von allen Robotern beachtet werden, was bedeutet, dass hier die Kommunikation wichtig ist. Hierzu wird für einen Roboter für die ihn umgebenden Zellen abgefragt, ob diese schon besucht wurden oder nicht. Auch hier bekommt jede Nachbarzelle einen Zähler mit einer Zufallswahrscheinlichkeit zugeordnet. Falls eine Zelle von keinem Roboter besucht wurde, steigt der Zähler für diese Zelle, sonst nicht. Es wird diejenige Zelle ausgewählt, welche den höchsten Zähler aufweist.

Cluster-Repelling-Walk Beim Cluster-Repelling-Walk wird der Schwarm kohärent über das Gebiet bewegt. Der Zielpunkt muss hierbei vom Schwarmmittelpunkt erreicht werden, was bedeutet, dass die Roboter sich in einer Formation bewegen. Wenn nun ein Ziel erreicht wurde, wird ein neues Ziel für den Schwarmmittelpunkt wie beim Self-Repelling-Walk bestimmt.

Cooperative-Area-Exploration Bei der Cooperative-Area-Exploration wird der Schwarm kohärent über das Gebiet bewegt, jedoch wird der Zielpunkt hierbei nicht aus der

Nachbarschaft gesucht, sondern zufällig aus dem gesamten Missionsgebiet. Dadurch sind die Ziele mit hoher Wahrscheinlichkeit weiter auseinander und der Schwarm muss weitere Strecken zurücklegen, um zum Zielpunkt zu gelangen.

Potential-Fields Bei den Potential-Fields wirken anziehende und abstoßende Kräfte auf die Roboter. In der von der PG gewählten Variante versuchen die Roboter den RSSI der Signale zueinander zwischen zwei Grenzwerten zu halten. Hierzu wirken auf die Roboter anziehende Kräfte zueinander, falls das Signal zu schlecht wird, bzw. abstoßende Kräfte, falls das Signal zu stark wird. Die Roboter werden dabei auch von Hindernissen abgestoßen. Im Abschnitt 6.1.1 wird genauer auf die Potential-Fields eingegangen.

Cluster-Breathing Das Cluster-Breathing nutzt eine Hysterese, um die Anzahl von Verbindungen zu anderen Robotern stabil zu halten. Hierbei bewegt sich ein Roboter in der jeweiligen Formation entweder nach Innen oder nach Außen, je nachdem, ob er zu wenige oder zu viele Verbindungen hat.

Mission-Planning Das Mission-Planning ist ein deterministisches Steering bei dem das zu erkundende Gebiet in gleichgroße Teile aufgeteilt wird, wobei die Anzahl der Roboter die Anzahl der Gebiete bestimmt. Nun bekommt jeder Roboter ein Gebiet zugeteilt, welches er erkunden soll. Dazu fährt der Roboter das Gebiet linienförmig ab und erkundet damit jede Zelle des Gebiets genau einmal.

Um statistische Effekte zu minimieren, werden für die Auswertung der Steerings jeweils immer fünf Simulationsläufe mit drei Robotern und den selben Steerings genutzt. Es wird dabei auf die Erkundungsrate und den Zusammenhalt des Schwarms geachtet, was im Folgenden beschrieben wird. Zusätzlich ist zu beachten, dass es um eine qualitative Bewertung der Steerings ging, um daraus generelle Schlüsse für unsere Strategie zu ziehen.

Die Abbildung 6.12 zeigt die Erkundung des Gebiets, wobei Steerings genutzt werden, welche jeden Roboter unabhängig vom Schwarm bewegen. Auf der X-Achse ist die Zeit der Simulation und auf der Y-Achse die Erkundung des Gebiets in Prozent angegeben. Hierbei werden drei Steerings gegenübergestellt: Random-Walk-2D (blau), Self-Repelling-Walk (rot) und Coop-Repelling-Walk (grün). Gut zu sehen ist der große Unterschied (ca. 17% nach 15 Minuten) zwischen dem Random-Walk und den anderen beiden Steerings. Dieser folgt daraus, dass der Random-Walk nicht auf die schon erkundeten Zellen achtet und nur zufällig agiert. Der Unterschied zwischen dem Coop-Repelling-Walk und dem Self-Repelling-Walk beträgt ca. 10% bei 15 Minuten, da der Coop-Repelling-Walk durch die Beachtung der besuchten Zellen der anderen Roboter weniger Zellen anfährt, welche schon einmal besucht wurden. Generell zeigt sich hier, dass die Aufklärungsrate durch das Beachten der anderen Roboter ansteigt.

In Abbildung 6.13 wird ein Graph zur Erkundung dargestellt, wobei Steerings genutzt werden, welche den Schwarm kohärent über das Gebiet fahren lassen. Auf der X-Achse ist die Zeit der Simulation und auf der Y-Achse die Erkundung des Gebiets in Prozent angegeben. Als Steerings wurden hierbei die Cooperative-Area-Exploration

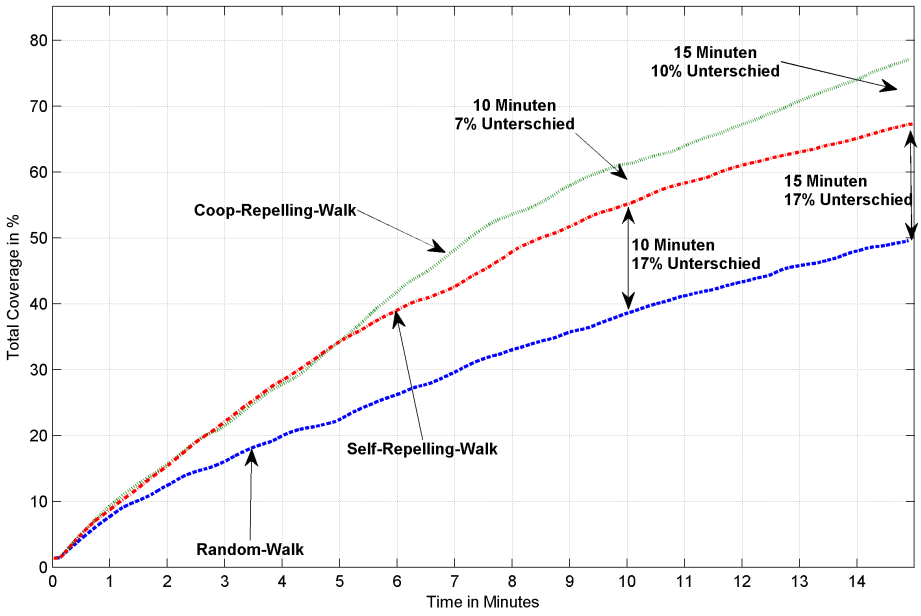


Abbildung 6.12: Vergleich von Steerings anhand der Erkundungsrate, mit einem sich untereinander unabhängig bewegendem Schwarm.

(blau), Coop-Repelling-Walk (rot) und der Cluster-Repelling-Walk (grün) genutzt. Insgesamt hat nach 15 Minuten der Coop-Repelling-Walk die höchste Erkundung erreicht und ist ca. 6% vor der Cooperative-Area-Exploration. Dies liegt daran, dass sich die Roboter aufteilen und somit bei fortschreitender Zeit mehr unerkundete Gebiete erreichen können. Die Cooperative-Area-Exploration kann jedoch eine viel höhere Erkundung erreichen als der Cluster-Repelling-Walk. Der große Unterschied zwischen Cooperative-Area-Exploration und Cluster-Repelling-Walk von ca. 25% bei 15 Minuten ist damit zu begründen, dass beim Cluster-Repelling-Walk durch die kurzen Wege des Schwarms viele Zellen häufiger besucht werden, da sich die Roboter im selben Bereich aufhalten, was bei der Cooperative-Area-Exploration durch die längeren Wege vermieden wird. Dies wird auch in den Abbildungen 6.14 und 6.15, welche 3D-Plots der besuchten Zellen darstellen, ersichtlich. Die Höhe der einzelnen Balken gibt die Anzahl der Besuche der entsprechenden Zelle an, wobei am rechten Rand eine Legende für die unterschiedliche Farbgebung dargestellt ist. Hier muss darauf geachtet werden, dass bei der Cooperative-Area-Exploration in 6.14 die Legende nur bis maximal neun Besuche, die Legende des Cluster-Repelling-Walk in 6.15 jedoch bis 20 Besuche reicht. Damit wird deutlich, dass bei der Cooperative-Area-Exploration viel weniger Zellen mehrfach besucht werden als beim Cluster-Repelling-Walk und dadurch die Erkundung insgesamt steigt.

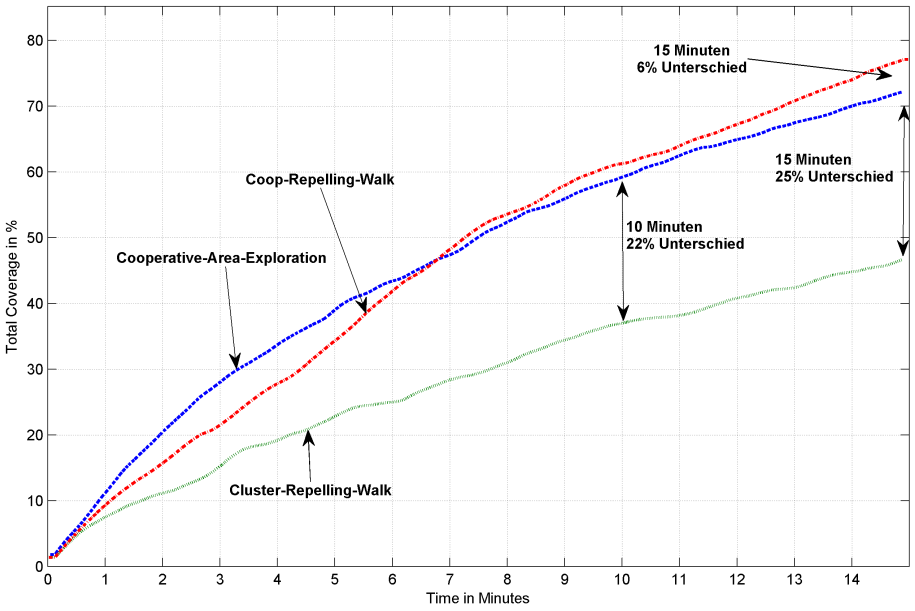


Abbildung 6.13: Vergleich von Steerings anhand der Erkundungsrate, mit einem sich kohärent bewegendem Schwarm.

Um die Erkundung bei der Cooperative-Area-Exploration zu erhöhen, können weitere Steerings eingesetzt werden. Diese sorgen dafür, dass der Schwarm kohärent bleibt und damit die Erkundungsrate steigt. Die Abbildung 6.16 zeigt einen Graphen zur Erkundung, wobei die Cooperative-Area-Exploration mit Steerings kombiniert wird, welche versuchen, die Schwarmkohärenz aufrecht zuhalten. Auf der X-Achse ist die Zeit der Simulation und auf der Y-Achse die Erkundung des Gebiets in Prozent angegeben. Hierbei werden drei Kombinationen der Steerings überprüft: Cooperative-Area-Exploration (blau), Cooperative-Area-Exploration + Cluster-Breathing (rot) und Cooperative-Area-Exploration + Potential-Fields (grün). In Kombination mit einem weiteren Steering (Cluster-Breathing oder Potential-Fields) erzielt die Cooperative-Area-Exploration höhere Werte bei der Erkundung (bei 15 Minuten, 79% bzw. 73% vs 72%). Auffallend ist vor allem die Kombination mit den Potential-Fields, da hier der Unterschied zur Cooperative-Area-Exploration alleine über einen weiten Zeitraum deutlich zu erkennen ist. Allgemein kann durch die Verwendung eines Steerings zur Aufrechterhaltung einer Formation die Erkundungsrate positiv beeinflusst werden, nicht nur bei der Cooperative-Area-Exploration, auch beim Cluster-Repelling-Walk lassen sich solche Ergebnisse beobachten.

Insgesamt gesehen sind die Cooperative-Area-Exploration + Potential-Fields und der Coop-Repelling-Walk gleich auf. Allgemein sind also zwei Strategien zu unterscheiden:

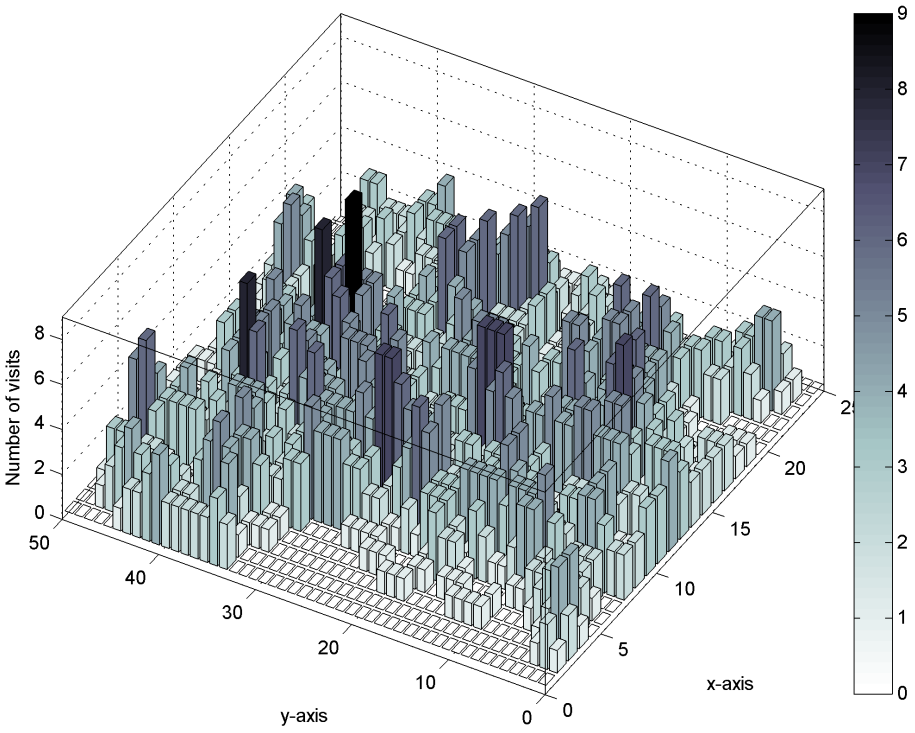


Abbildung 6.14: Anzahl an Wiederbesuchen bei Cooperative-Area-Exploration. Die Höhe der Balken gibt an wie oft eine Zelle besucht wurde.

- Den Schwarm konsistent zu halten
- Die Roboter aufzuteilen, wobei die Absprache zu beachten ist

Im Projekt wird die zweite Strategie ausgewählt und die Roboter zur Erkundung aufgeteilt. Es ist jedoch wichtig, dass die gesammelten Daten ausgetauscht werden, damit Gebiete nicht unabsichtlich öfters erkundet werden und damit die Erkundungsrate sinkt. In der Strategie treffen sich die Roboter nach jeder Erkundungsrunde kurz zum Austausch der gesammelten Daten und sprechen das weitere Vorgehen ab. Da die gesammelten Daten vom Data-Mining-Agenten ausgewertet und gespeichert werden und dann nur diese Daten ausgetauscht werden, hält sich der Datenaustausch in Grenzen. Um zu überprüfen, wie gut die Strategie funktioniert, wird sie im späteren Verlauf mit der Cooperative-Area-Exploration + Potential-Fields verglichen. In diesem Projekt werden zwei Varianten der Cooperative-Area-Exploration + Potential-Fields festgelegt: Einmal die hier vorgestellte und dann eine, in der die Auswahl des anzufahrenden

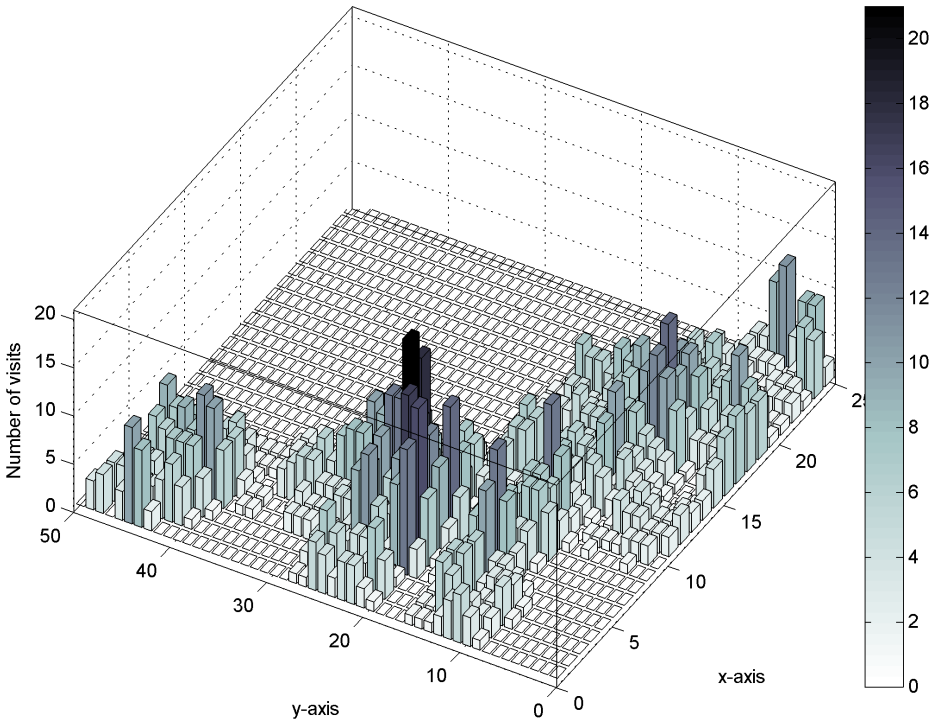


Abbildung 6.15: Anzahl an Wiederbesuchen bei Cluster-Repelling-Walk. Die Höhe der Balken gibt an, wie oft eine Zelle besucht wurde.

Gebiets vom Data-Mining-Agenten getroffen wird. Der genaue Ablauf der im Projekt entwickelten Strategie *SUN Search* wird im nächsten Abschnitt behandelt.

6.3.2 Strategie

Algorithmus Übersicht Nachdem die vorhandenen Algorithmen analysiert wurden und die Vor- und Nachteile herausgearbeitet waren, konnten die gesammelten Informationen in den neuen Algorithmus einfließen. Zu Beginn stimmen die einzelnen Roboter ab, welche Gebiete erkundet werden sollen. Ist diese Entscheidung getroffen, wird entschieden, welchen Punkt die Roboter nach Erkundung ihrer Gebiete als gemeinsamen Treffpunkt ansteuern. Damit ist eine Iteration des Algorithmus abgeschlossen und das Verfahren startet wieder von vorne. Abbildung 6.17 zeigt eine Übersicht über den Algorithmus.

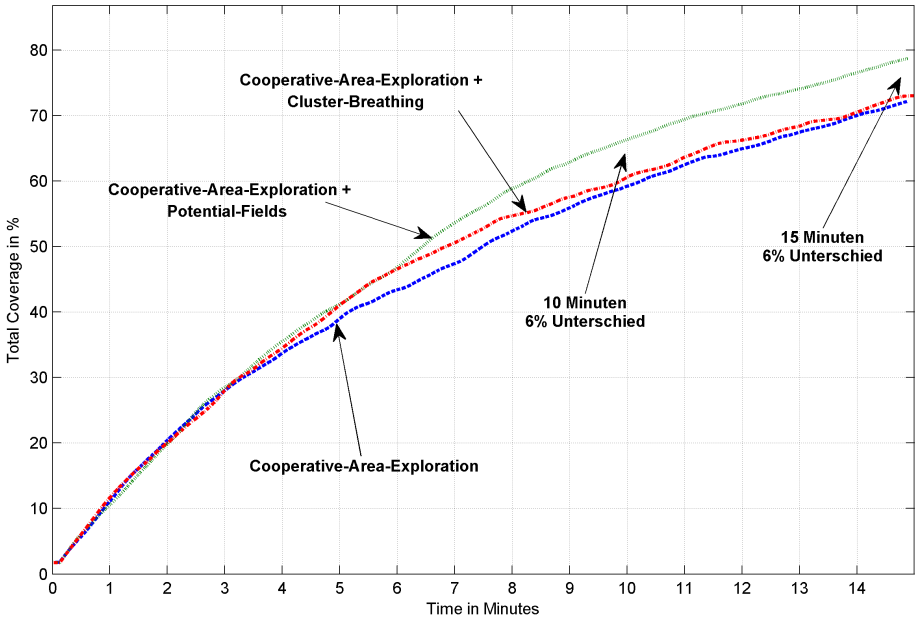


Abbildung 6.16: Vergleich von Steerings anhand der Erkundung

Algorithmus im Detail Der Algorithmus gliedert sich in zwei Phasen. Die erste Phase ist die sogenannte Entscheidungsphase. In dieser Phase wird entschieden, welche Gebiete von welchen Robotern erkundet werden sollen und in welchem Gebiet sich die Roboter treffen werden. In der zweiten Phase, der Erkundungsphase, werden die einzelnen Gebiete mit einer bestimmten Strategie erkundet, wobei die Wahl der Strategie abhängig von der Größe des Gebiets ist, welches erkundet werden soll. Nach Abschluss der Erkundung steuern die Roboter das in der ersten Phase ausgewählte Ziel an.

Entscheidungsphase Diese Phase setzt voraus, dass die Roboter miteinander kommunizieren können. Im ersten Schritt dieser Phase stellt jeder Roboter eine Anfrage an seinen eigenen Data-Mining-Agenten welches die nächsten potentiellen Ziele sind, die entweder erkundet werden sollen oder nochmals abgefahren werden müssen, um das Umgebungsmodell zu verbessern. Unabhängig von der Art, verbessern oder erkunden, ist die Antwort vom Data-Mining-Agenten eine Menge von Gebieten der Größe 4×4, 2×2 oder 1×1. Da nicht klar ist, welcher Data-Mining-Agent das genaueste Modell der tatsächlichen Umgebung hat, muss jeder Roboter diese Anfrage an seinen eigenen Data-Mining-Agenten stellen. Um sicherzustellen, dass nicht alle Roboter dasselbe Gebiet als Ziel auswählen, fangen die Roboter abwechselnd damit an, die eigenen Gebiete

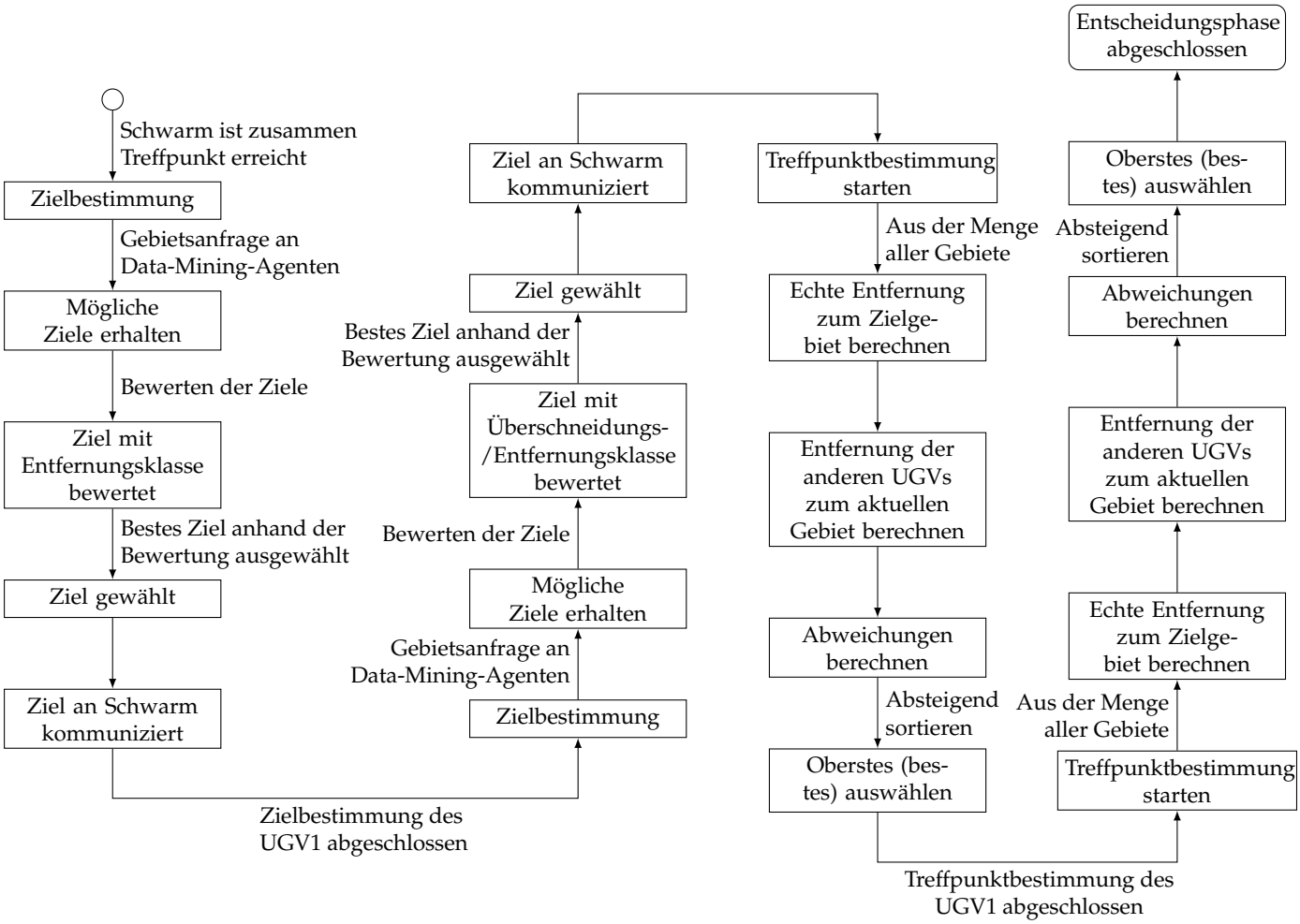


Abbildung 6.17: SUN Search: Zielbestimmung für zwei UGVs

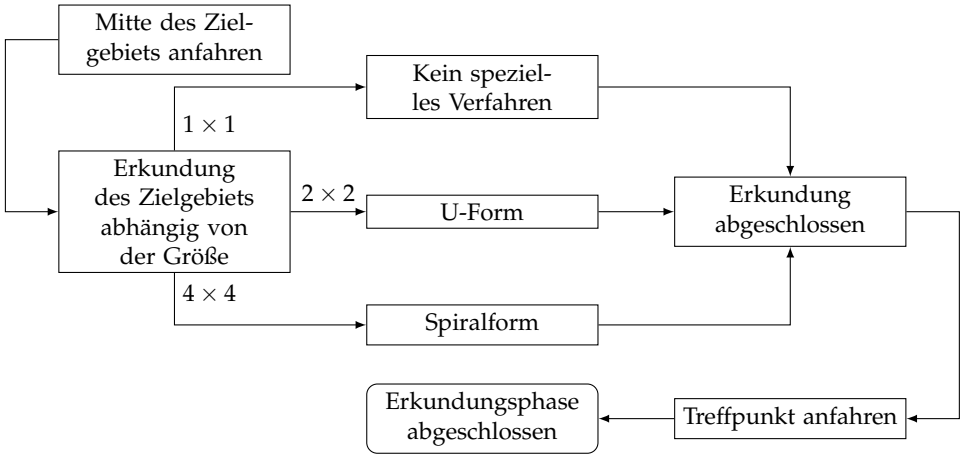


Abbildung 6.18: SUN Search: Ablauf der Erkundung des Zielgebiets

zu bewerten. Grundsätzlich unterteilt sich die Bewertung eines Zielgebietes in zwei Faktoren. Der erste Faktor ist die Überschneidungsklasse.

Überschneidungsklasse Diese gibt an, zu wie viel Prozent sich das aktuelle Gebiet mit einem anderen (ausgewählten) Gebiet überschneidet.

Die Überschneidung eines Gebietes mit anderen Gebieten zu kennen ist ein großer Vorteil für den Schwarm. Würden zwei Roboter in ein Gebiet fahren, welches eine starke Überschneidung hat, wird die Gefahr, dass die Roboter kollidieren, stark erhöht und die Erkundungsrate würde sinken. Der zweite Faktor ist die Entfernungsklasse.

Entfernungsklasse Diese gibt an, wie weit ein Zielgebiet von der aktuellen Roboterposition entfernt ist. Sind die jeweiligen Entfernungen zu den unterschiedlichen Zielgebieten etwa gleich, rücken ebenfalls die Fahrzeiten, bzw. Erkundungszeiten der einzelnen Roboter zusammen.

Der Roboter, der in der aktuellen Iteration als erstes seine Gebiete bewertet, nimmt eine Sonderrolle ein. Da noch keine Gebiete von anderen Robotern als Zielgebiet festgelegt worden sind, braucht er keine Überschneidungsklassen zu berechnen. Es muss lediglich die Entfernungsklasse berechnet werden. Nachdem der Roboter die Entfernungsklasse für jedes Gebiet berechnet hat, wählt er das Gebiet mit der besten Klasse aus. Sollte es mehrere Gebiete mit der besten Klasse geben, wird per Zufall ein Gebiet ausgewählt. Der erste Roboter hat jetzt sein Zielgebiet festgelegt. Diese Wahl wird nun den anderen Fahrzeugen im Schwarm mitgeteilt. Da es ab diesem Zeitpunkt ein festes Zielgebiet gibt, können die nächsten Roboter dieses Gebiet in ihre Bewertungen mit einbeziehen. Dazu benutzen sie die Überschneidungsklasse, die jetzt aus dem gewählten Gebiet des Vorgängers und dem aktuell zu betrachtendem Gebiet berechnet werden kann. Nachdem die erste Bewertung nun Gebiete mit einer hohen Überschneidung als schlecht eingestuft hat, wird die Entfernung der Gebiete mit in die Bewertung einbezo-

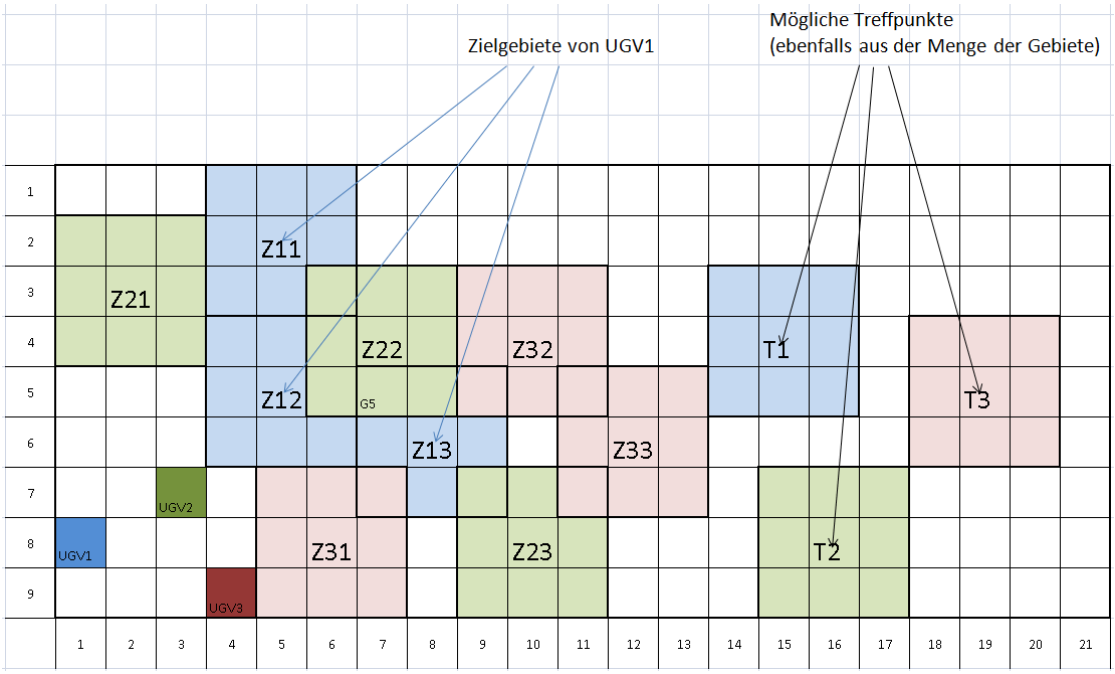


Abbildung 6.19: SUN Search: Treffpunkte und Zielgebiete aller Schwarm-Teilnehmer

gen. Dies geschieht genauso wie bei dem ersten Roboter mit Hilfe der Entfernungsklasse. Diese beiden Faktoren werden jetzt addiert und jedes Gebiet hat so eine Bewertung. Der Roboter wählt als nächstes das Gebiet mit der besten Bewertung aus und kommuniziert es an die anderen Schwarm-Mitglieder. Dieses Verfahren wird fortgesetzt, bis alle Roboter ein Zielgebiet festgelegt haben. Nachdem dieser Vorgang abgeschlossen ist, hat jedes Fahrzeug ein Zielgebiet und Information darüber, in welches Gebiet die anderen Schwarm-Mitglieder fahren.

Abbildung 6.19 zeigt eine mögliche Ausgangssituation für drei Roboter, die jeweils vier 4×4 große Gebiete von ihrem Data-Mining-Agenten zurück bekommen haben.

Angenommen UGV1 wäre das erste Fahrzeug, welches sein Zielgebiet wählen soll, dann wird für Z11, Z12 und Z13 jeweils die Entfernungsklasse berechnet. Abbildung 6.20 zeigt das Szenario, nachdem sich UGV1 für das Zielgebiet Z11 durch die Auswertung der Bewertungen entschieden hat.

Die Gebiete Z12 und Z13 sind nicht mehr relevant. Nun beginnt UGV2 damit, seine Gebiete zu bewerten. Bei der Betrachtung von Gebiet Z22 wird eine Überschneidung festgestellt und das Gebiet wird schlechter bewertet. Unter Einbeziehung der Entfernungen zu den möglichen Zielgebieten trifft UGV2 seine Entscheidung und wählt Gebiet Z23. Das gleiche Verfahren findet ebenfalls bei UGV3 Anwendung, sodass sich das Abbildung 6.21 dargestellte Endszenario ergibt.

Der nächste Schritt in der Entscheidungsphase ist nun, einen Treffpunkt zu definieren, den die Roboter ansteuern, sobald sie ihr Zielgebiet erkundet haben. Ein solcher

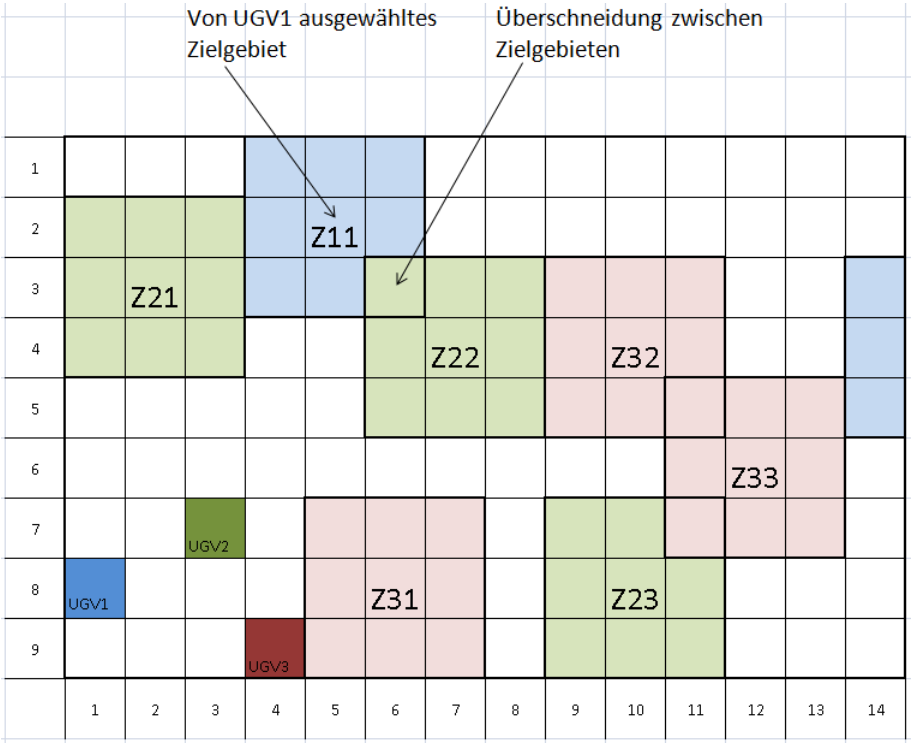


Abbildung 6.20: SUN Search: Nach Gebietsauswahl durch UGV1

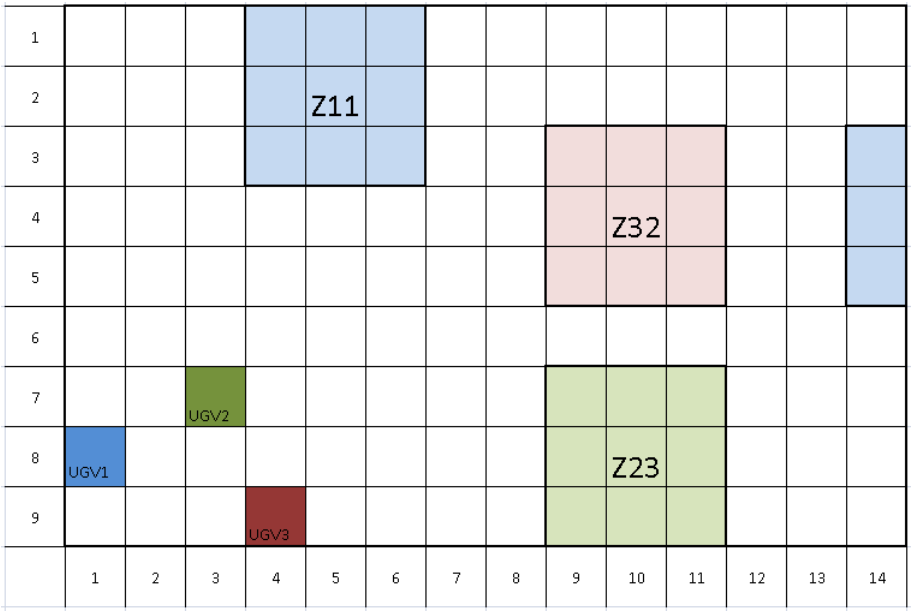


Abbildung 6.21: SUN Search: Die Zielgebiete sind ausgewählt

Treffpunkt ist wichtig, weil bei der Auswahl der Zielgebiete nicht berücksichtigt wird, ob die Roboter während der Erkundung der Zielgebiete kommunizieren können. In dem Beispiel könnten die Abstände zwischen den Gebieten Z11, Z32 und Z23 so weit sein, dass keine Kommunikation mehr möglich ist. Der Treffpunkt, den die Roboter nach der Erkundung anfahren, wird ebenfalls aus der ursprünglichen Menge der Zielgebiete gewählt. Da es wünschenswert ist, dass alle Roboter möglichst zum gleichen Zeitpunkt am Treffpunkt sind, wird dieser nicht zufällig gewählt, sondern abhängig von den zu erkundeten Zielgebieten. Wie auch bei der Auswahl der Zielgebiete beginnt wieder ein Roboter mit der Bewertung eines potentiellen Treffpunktes. Im ersten Schritt der Bewertung wird die Entfernung vom Zielgebiet zum Treffpunkt berechnet. Hier wird die tatsächliche Distanz berechnet, nicht wie bei den Erkundungsgebieten die Entfernungsklasse. Anschließend wird für jeden weiteren Roboter im Schwarm die eigene Entfernung zum aktuellen Treffpunkt berechnet. Aus diesen beiden Entfernungen kann jetzt die Abweichung der anderen Fahrzeuge zum aktuellem Treffpunkt berechnet werden. Diese Abweichungen werden auf die eigentliche Distanz addiert um so den Treffpunkt zu bewerten.

$$Abweichung = d + \frac{\sum_{k=2}^{n} (d - Distance(p_k, m))}{n}$$

dabei gilt,

- d = *Aktuelle Entfernung*
- p = *Position des Roboters*
- m = *Betrachteter Treffpunkt*
- n = *Anzahl Roboter*

Die Abbildung 6.22 zeigt eine Situation in der ein möglicher Treffpunkt T1 bewertet werden soll. Es wird also zuerst die direkte Entfernung zwischen Z11 und T1 berechnet. Dann die Abweichung von Z32 zu T1 d.h. Entfernung Z11 und T1 minus Entfernung Z32 und T1. Das gleiche wird mit Z23 gemacht.

Nachdem alle potentiellen Treffpunkte vom aktuellem Roboter bewertet worden sind, werden diese aufsteigend sortiert und der erste Eintrag wird an die anderen Schwarmteilnehmer weiter gegeben. Wenn jeder Roboter seine Treffpunkte bewertet hat, wird per Zufall ein Treffpunkt ausgewählt und an alle Roboter kommuniziert. Dieses Verfahren der Treffpunktbewertung hat für die Strategie diverse Vorteile. Der Treffpunkt wird so gewählt, dass er für alle Roboter in etwa gleich gut ist. Würde ein Treffpunkt zufällig gewählt werden, könnte es passieren, dass ein Roboter diesen Treffpunkt sehr schnell erreichen würde, die anderen Teilnehmer aber sehr lange fahren müssten. So würden die Roboter, die bereits am Treffpunkt sind, lange warten und könnten den Playground nicht weiter erkunden.

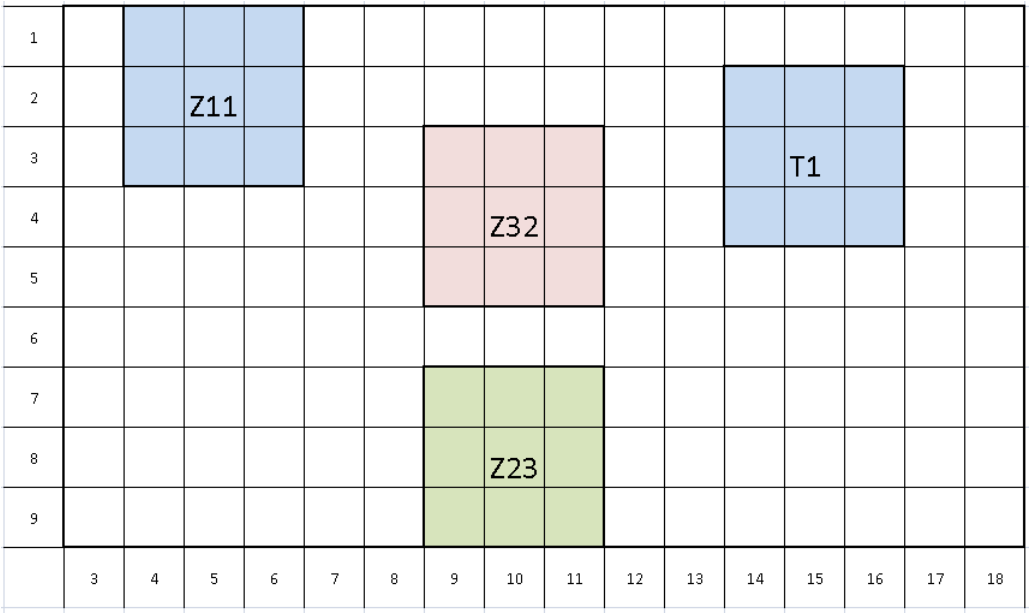


Abbildung 6.22: SUN Search: Auswahl eines Treffpunktes

Der letzte Punkt der Entscheidungsphase ist das Berechnen der Wartezeiten. Diese Zeit wird eingesetzt, damit die Fahrzeuge wissen, wie lange sie maximal am Treffpunkt warten müssen. Die eigentliche Berechnung gestaltet sich relativ simpel. Jedes Fahrzeug berechnet dazu den eigenen Weg und den Weg aller anderen Schwarmteilnehmer. Für den längsten berechneten Weg wird nun die Zeit ermittelt und die eigene Fahrzeit abgezogen, so ergibt sich die Zeit, die das Fahrzeug am Treffpunkt warten muss. Mit dem Berechnen der Wartezeit ist die erste Phase, die Entscheidungsphase, abgeschlossen.

Erkundungsphase Jeder Roboter beginnt jetzt damit, seine gewählten Gebiete zu erkunden. Dazu fährt jedes Fahrzeug direkt zum Mittelpunkt des jeweiligen Zielgebietes. Nun wird dieses Gebiet abhängig von der Größe erkundet. Ein 1×1 Gebiet wird von dem Roboter einfach angefahren und mit keinem speziellem Verfahren erkundet. Bei dem Gebiet mittlerer Größe, 2×2, fährt das Fahrzeug in einer U-Form durch das komplette Zielgebiet. Das größte Gebiet, 4×4, wird von der Mitte aus in einer Spiralform erkundet. Nachdem ein Roboter sein Zielgebiet erkundet hat, bewegt er sich in Richtung des abgestimmten Treffpunktes. Da der Treffpunkt nicht zu den Gebieten gehört, die erkundet werden, bleibt ein Roboter auf seinem Weg stehen, sobald er eine Verbindung mit allen anderen Schwarmteilnehmern hat. Besteht die Verbindung wieder zu allen Teilnehmern, ist eine Iteration abgeschlossen und der Algorithmus startet wieder mit der Entscheidungsphase.

Da Roboter nicht fehlerfrei sind, können selbige ausfallen oder es können andere Situationen eintreten, auf die reagiert werden muß. Im Worst Case fällt ein Roboter kom-

plett aus. Das Verfahren kann einen Ausfall feststellen, indem es am Treffpunkt zwei Faktoren überprüft. Zum einen muss die Wartezeit abgelaufen sein und zum anderen existieren weniger Verbindungen als am vorherigen Treffpunkt. Treffen diese beiden Faktoren für die am Treffpunkt eingetroffenen Roboter zu, wissen diese, dass mindestens ein Fahrzeug ausgefallen ist. Da die Roboter einen Schwarm bilden und es somit keine direkte Rollenverteilung gibt, kann flexibel reagiert und das Verfahren mit den noch vorhandenen Robotern durchlaufen werden. Sollte der Roboter keinen Komplettausfall haben, sondern sich erheblich verspäten, durch z. B. Hindernisse, trifft der Roboter am Treffpunkt ein und findet keine anderen Roboter mehr vor. Da der Roboter durch seine eigene Wartezeit erkennt, dass er zu spät ist und die anderen Fahrzeuge schon weg sind, fährt er zum Startpunkt bzw. zu vorher definierten Koordinaten. Diese Koordinaten sollten sich allerdings in der Nähe der Basisstation befinden. (Abhängig von der gewählten Verbindungsmetrik.) Dieses Fahrzeug bleibt jetzt auf den Koordinaten stehen und wartet. Sobald der aktive Teil vom Schwarm genug Informationen gesammelt hat, oder zwecks Missionsplanung Verbindung mit der Basisstation aufnehmen muss, wird das vorher verlorene Fahrzeug wieder in den Schwarm mit aufgenommen.

6.3.3 Synchronisationsprobleme bei der Portierung nach C++

Nach der Umsetzung in C++ kann die Strategie im Framework genutzt werden und damit auch in der Hardware-in-the-Loop-Simulation. Da die Agenten auf unterschiedlichen Rechnern laufen, können nun auch Synchronisations-Probleme auftreten, da die Rechner nicht vollkommen synchron gestartet werden und die auszutauschenden Daten über das Netz geschickt werden müssen.

Hierbei ist aufgefallen, dass die verschiedenen Status der Strategie nicht robust arbeiten, da es passieren kann, dass durch die fehlende Synchronisation ein Roboter seinen Status nicht mehr verlassen kann und die anderen Roboter im Folgestatus auf diesen warten. Dies geschieht dadurch, dass im Folgestatus Informationen gelöscht wurden, welche für den Statusübergang gebraucht werden. Wechselt ein Roboter den Status nicht schnell genug, werden durch die eingehende Nachricht die Informationen gelöscht und dann als fehlend interpretiert. Das führt dazu, dass der Roboter den Status nicht mehr wechseln kann, weil er weiter auf die korrekte Nachricht wartet. Die Roboter warten hierbei zyklisch aufeinander.

Dieser Fehler konnte erst durch die Hardware-in-the-Loop-Simulation erkannt und behoben werden, da solche Fehler in der Matlab Simulation nicht auftreten können. Als Lösung werden die Daten beim Statusübergang erst gelöscht, wenn diese definitiv nicht mehr benötigt werden, sodass diese Situation nicht mehr auftreten kann.

7 Kommunikation zwischen Agenten

Es liegt in der Natur des entwickelten Systems, dass die Komponenten miteinander kommunizieren müssen. Ziel ist es schließlich, dass eine Gruppe von Robotern kooperativ nach Gebieten hoher Energiedichte sucht. Kommunikation kommt dabei an einigen Stellen vor: zum einen müssen innerhalb des Roboters die Daten der Sensorik an den Data-Mining-Agent weitergeleitet werden, zum anderen müssen die Data-Mining-Agenten verschiedener Roboter miteinander kommunizieren um die P2P-Data-Mining-Algorithmen umzusetzen. Weiterhin muss auch die Erkundungsstrategie SUN Search (siehe Abschnitt 6.3.2) intern mit dem eigenen Data-Mining-Agent kommunizieren um die Empfehlungen für neue Gebiete zu bekommen und sich extern mit den Mobilitätsagenten der anderen Roboter absprechen damit es zu einer Entscheidung kommen kann, welche Gebiete als nächstes angefahren werden sollen. Bislang wurde diese Kommunikation in der Beschreibung der jeweiligen Algorithmen weitgehend als Blackbox betrachtet, Inhalt dieses Kapitels ist daher nun die notwendigen Details der Lösungen für all diese Kommunikationsprobleme zu beschreiben.

7.1 Grundlagen

Bevor die Details der entwickelten Kommunikationsmethoden betrachtet werden können, folgen in diesem Abschnitt zunächst einige Grundlagen zu Kommunikation im Allgemeinen. Es wird in Abschnitt 7.1.1 zunächst eine technische Einführung in Mesh-Netze gegeben, da die Agenten ohne eine Zentrale Einheit zur Organisation der Kommunikation (Router, Access Point, o. ä.) auskommen sollen. Daraufhin wird in Abschnitt 7.1.2 auf theoretische Grundlagen des Designs von Kommunikationsprotokollen eingegangen.

7.1.1 Mesh-Networks

Der nachfolgende Abschnitt führt in das Thema der vermaschten Netze (sogenannte Mesh-Networks) ein. Es werden Begriffe aus dem Gebiet vermaschter Netze eingeführt und die Routingproblematiken solcher Netze aufgezeigt. Darauf aufbauend werden grundsätzliche Methoden für paketorientierte Kommunikation und ein häufig verwendetes Protokoll vorgestellt. Die Hardwarespezifikationen (siehe Kapitel 4) waren zu Beginn bereits bekannt, daher konzentriert sich dieses Kapitel ausschließlich auf die Funktechnologie IEEE 802.11, weitere Technologien wie IEEE 802.15 (WPAN) oder IEEE 802.16 (WiMax) werden außenvorgelassen. Der Abschnitt schließt mit der Evaluierung von relevanten Standards aus dem IEEE 802.11 Normenkatalog.

Der Titel der Projektgruppe „Kooperatives Datamining mit vernetzten Robotern“ hebt bereits die Relevanz der Kommunikation zwischen mehreren Teilnehmern hervor. Die Anforderungen an solch ein Kommunikationsnetz sind wie folgt formuliert:

Direkt Direkte Kommunikation zwischen Teilnehmern in Reichweite.

Transitiv Indirekte Kommunikation über weiterleitende Teilnehmer.

Adressierbarkeit Teilnehmer sollen eindeutig adressierbar sein.

Robustheit Alternative Route bei Wegfall eines weiterleitenden Teilnehmers.

Ein Netz mit diesen Eigenschaften wird vermaschtes Netz genannt. Misra, *et al.* [34] zählen die grundlegenden fünf Arten auf, ein Funknetz aufzuspannen. Die beiden Modi **Infrastrukturmodus**, in welchem ein zentraler Router und mehrere Teilnehmer agieren sowie der **Ad-Hoc Modus**, wo alle Teilnehmer untereinander verbunden sind und Datenaustausch nur zu Teilnehmern in Funkreichweite möglich ist, werden in diesem Kapitel nicht weiter erläutert. Die drei anderen Arten werden jetzt näher untersucht.

Wireless Mesh Network (WMN) In einem WMN gibt es 2 Arten von Teilnehmern. **Endgeräte** sind die Daten generierenden Knoten, **Mesh-Access-Points** (MAP) spannen ein Infrastruktur-Mesh auf. Es sind also nur die MAPs vermascht, nicht jedoch die Endgeräte, daher erfüllt diese Art von Funknetzen das Zielszenario nicht.

Mobile Ad-hoc Networks (MANET) Teilnehmer werden in solch einem Netz als Mesh Station (mesh STA) bezeichnet und können direkt oder per multi-hop Übertragung miteinander kommunizieren. Jede STA fungiert als Router und ist anders als im WMN frei beweglich. Die dadurch entstehende variierende Netztopologie inkl. dem hinzukommen von weiteren STAs sowie dem Verlassen von STAs aus dem Netz kann von den Routingprotokollen abgefangen werden, das Netz ist also „selbstheilend“. Die Anforderungen werden von MANETs erfüllt.

Vehicular Ad-Hoc Networks (VANET) Als letztes seien noch die VANETs erwähnt. Diese ähneln Ad-Hoc Netzen, nur das Verbindungen zu Teilnehmern sehr schnell aufgebaut und wieder abgebaut werden können müssen. Netze dieser Art sind für geringen Datenverkehr und anonyme schnelle Kommunikation zwischen Teilnehmern ausgerichtet und erfüllen damit die oben geforderten Anforderungen nicht.

Routingarten in einem MANET Zentraler Bestandteil eines Netzwerks, dessen Topologie sich jederzeit ändern kann, ist das Routing. Zur Abgrenzung vom IP Routing wird in der Literatur der Begriff „Path selection“(Wegfindung) eingeführt, welcher im Folgenden auch verwendet wird. Sofern die gesamte Topologie oder auch nur Teile bzw. beliebige Teilstrecken zwischen zwei kommunikationswilligen Teilnehmern bekannt sind, werden Wegfindungsalgorithmen wie Bellman-Ford oder Dijkstra angewendet.

Erkundungsstrategien, um die relevanten Teile des Netzwerkes zu bestimmen, werden in drei Kategorien einsortiert, die nachfolgend erläutert sind.

Proaktives Routing Beim proaktiven Routing werden periodisch Routentabellen über die gesamte Netztopologie verteilt. Dadurch ist eine Übertragung durch sehr geringe Latenzen gekennzeichnet, da die optimale Route im Sender direkt berechnet werden kann. Das periodische Verteilen der gesamten Topologie, neben der eigentlichen Erkennung derselben, verursacht jedoch auch hohen Protokolloverhead. Prinzipbedingt kann nur langsam auf Teilnehmerausfall oder sich bewegende Teilnehmer reagiert werden.

Reaktives Routing Das reaktive Routing zeichnet sich durch die Routensuche zu Beginn einer Übertragung aus. Dadurch entsteht eine hohe initale Latenzzeit beim Versenden von ersten Paketen zwischen Teilnehmern. Auf Teilnehmerausfall oder sich bewegende Teilnehmer kann dadurch jedoch flexibel reagiert werden. Um eine Wegfindung durchzuführen, muss das Netz mit *Route Request*-Paketen geflutet werden. Beinhaltet der Anwendungsfall, dass viele kurze Nachrichten ausgetauscht werden, so ist der Protokollaufwand größer als die Nutzdaten und die Latenzzeit des gesamten Netzwerkes hoch.

Hybrides Routing Dieser Ansatz vereint beide zuvor vorgestellten Ansätze. Das initale Wegfindungsproblem wird mit dem proaktiven Ansatz angegangen, es werden also periodisch Routingtabellen verteilt. Bei Verbindungsproblemen auf einer Route wird jedoch nicht auf die nächste periodische Verteilung von Routen gewartet, sondern der reaktive Ansatz für die *Path selection* genutzt. Weiter unten wird das *Hybrid Wireless Mesh Protocol* (HWMP) vorgestellt, welches diesen Ansatz verwendet.

Routingmetrik

Nach einem Blick auf Abbildung 7.1 wird klar, dass es eine Metrik geben muss, welche bei mehreren möglichen Pfaden für die Auswahl eines Übertragungsweges sorgt. Auf der Abbildung sind fünf Knoten $R_{Start}, R_2 \dots R_4, R_{Ziel}$ zu sehen, wobei R_{Start} Daten an R_{Ziel} senden möchte. Zur Wahl steht der Pfad $R_{Start} \dashrightarrow R_2 \dashrightarrow R_{Ziel}$ oder $R_{Start} \dashrightarrow R_3 \dashrightarrow R_4 \dashrightarrow R_{Ziel}$. Nachfolgend sind drei Metriken erläutert.

Hop Count Durch die *Hop Count*-Metrik wird der Pfad gewählt, welcher am wenigsten Knoten enthält. Damit tendiert diese Metrik zu weiter entfernten Knoten. Ein erhöhter Sendestrom ist die Folge, zumal weite Funkstrecken störanfälliger sind. Dies induziert eine erhöhte Chance, dass Pakete neu gesendet werden müssen und die Paketlaufzeit ansteigt.

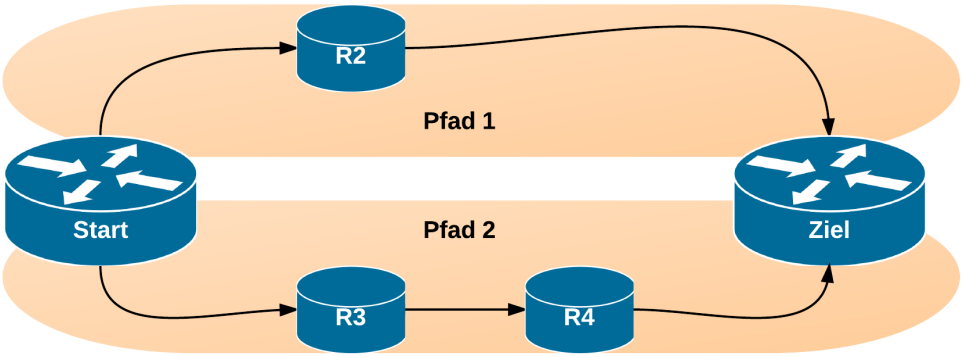


Abbildung 7.1: Mehrere Pfade von Quell- zu Zielknoten in einem Funkmeshnetzwerk erfordern eine Routingmetrik

Signal Strength Durch die *Signal Strength*-Metrik wird ein Pfad so gewählt, dass jeder an der Übertragung beteiligte Knoten nur minimal Energie für die Weiterleitung des Pakets aufwenden muss. Dafür ist jedoch ein regelmäßiger Kontakt zu Nachbarknoten notwendig, um die Signalstärke zu kennen, was einen gewissen Funkoverhead induziert. Eine Eigenschaft der Funkübertragung ist zudem eine asynchrone Sende/-Empfangsqualität und das Mess-Beacons eine andere Charakteristik als Nutzdaten aufweisen.

Airtime Die *Airtime*-Metrik gibt für das Senden eines Datenpakets über einen speziellen Pfad die Menge an verbrauchten Kanalressourcen an. Die Kosten eines Pfades werden über folgende Formel berechnet, die auf Konstanten aus Tabelle 7.1 zurückgreift:

$$ca = (OCA + OP + BT/r) * 1/(1 - efr).$$

	5 GHz	**2,4GHz**
Kanalzugriffsoverhead OCA	72μS	335μS
MAC Protokolloverhead OP	110μS	364μS
Testframe in Bits BT	8224	8224

Tabelle 7.1: IEEE 802.11 *PHY Layer*-Parameter für die *Airtime*-Metrik in vermaschten Netzen

Abgesehen von den definierten Konstanten ist die Formel abhängig von der *Übertragungsbitrate r* in MBit/s und der *Fehlerframerate efr* [1]. Es fließen also ähnliche Überlegungen wie bei der *Signal Strength*-Metrik ein, wobei jedoch mehr Wert auf die Auswertung einer Datenübertragung gelegt wird, als nur die Signalstärke als Alleinstellungsmerkmal zu nutzen.

Hybrid Wireless Mesh Protokol Wie zuvor erwähnt nutzt das *Hybrid Wireless Mesh Protocol* (HWMP) eine Kombination aus proaktivem und reaktivem Routing (Hybrides Routing) [28]. Es basiert auf dem reaktiven *Ad hoc On-Demand Distance Vector* (AODV) Protokoll. Die Funktionsweise lässt sich wie folgt zusammenfassen: Solange keine Kommunikation stattfindet, werden auch keine Daten ausgetauscht. Bei einem Verbindungsaufbau zwischen einem Startknoten K_S und einem Zielknoten K_D werden an alle erreichbaren Knoten Verbindungsanfragen verschickt, welche diese dann weiterleiten, dabei aber den Sender der Nachricht und die Anzahl der zurückgelegten Knoten abspeichern. Der Zielknoten K_D erhält so möglicherweise viele Verbindungsanfragen von unterschiedlichen weiterleitenden Knoten und antwortet auf dem Weg mit den wenigsten Knoten. Erreichbare Nachbarknoten und Routen werden in jedem Knoten zwischengespeichert. Wenn eine Verbindung zu einem Knoten abbricht, wird vom vorhergehendem Knoten eine Fehlermeldung an den Startknoten K_S zurückgeliefert und die Wegfindung beginnt von neuem. Zusätzlich zu diesem reaktivem Verfahren nutzt HWMP auch pro-aktive Mechanismen. So werden die Routinginformation von Knoten periodisch verteilt und Zugangsknoten („Gate Nodes“) in das Mesh haben durch proaktive Techniken die gesamte Topologie des Netzwerks vorliegen. HWMP nutzt für die Routenbestimmung die *Airtime Metrik*.

802.11p - Wireless Access in Vehicular Environments Diese Spezifikation erweitert die IEEE 802.11 Normenfamilie um Fahrzeug-zu-Fahrzeug Kommunikation[1]. Dieses Szenario erinnert an unser Projektziel und war daher Teil der Evaluation. IEEE 802.11p zeichnet sich durch schnell variierende Netzteilnehmer mit möglicherweise nur kurzen Verbindungsdauern aus. Routing zwischen Teilnehmern findet nicht statt. Der schnelle Aufbau der Kommunikation ist durch den Verzicht eines Basic Service Set (BSS) Kontextes, der sonst in allen anderen Netzen errichtet wird, sowie dem Fehlen sämtlicher Authentifizierungs- und Assozierungsmethoden, möglich. Die Addressierbarkeit, eine der oben genannten Anforderungen an das Kommunikationsnetz, ist damit nicht gegeben und müsste selbst implementiert werden. IEEE 802.11p ist daher nicht ohne weiteres geeignet.

802.11s - Mesh Networking Diese Spezifikation erweitert die IEEE 802.11 Normenfamilie um Mesh Netzwerkkommunikation[2]. Teilnehmer verbinden sich explizit mit einem Mesh, identifiziert über einen Mesh Basic Service Set (MBSS) Kontext. Jedes Endgerät ist zugleich Router. Endgeräte können den Status eines Mesh Gates erhalten. Über solch einen Zugang können Knoten außerhalb mit Teilnehmern des Netzes kommunizieren. Das Routing läuft über das vorgestellte *Hybrid Wireless Mesh Protocol* (HWMP) mit *Airtime metric*. Es existiert eine vollständige Implementierung des Standards im Linux Kernel ab 2.6.26.

[1] IEEE 802.11p `http://standards.ieee.org/getieee802/download/802.11p-2010.pdf`
[2] IEEE 802.11s `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6018236`

Fazit

Aufgrund der vorhandenen Unterstützung in aktuellen Linux Kerneln und der Möglichkeit diese Art von Mesh über handelsübliche WLan-Karten zu nutzen wird für die Projektgruppe das 802.11s Mesh verwendet.

7.1.2 Protokolldesign

In jedem Projekt, das einem verteilten Ansatz folgt, muss ein Protokoll gefunden werden, mit dem die verteilten Komponenten miteinander kommunizieren können. Im Folgenden soll ein grober Überblick über einige Protokolle verschiedener Kommunikationsebenen gegeben werden, sowie ein paar Hinweise, was beim Entwurf eines eigenen Protokolls beachtet werden sollte.

Schichten-Modell Das ISO/OSI Schichten-Modell beziehungsweise der TCP/IP Protokollstack zeigen, wie die Protokollhierarchie von der Anwendungsebene bis zur physikalischen Bitübertragungsschicht aufgebaut ist. In der Projektgruppe werden verschiedene Protokollebenen bedient, hier soll der Schwerpunkt aber eher im High-Level-Bereich liegen. Daher werden als Beispiele nur Protokolle der Anwendungs- und Transportschicht betrachtet.

Designfragen Beim Entwurf von eigenen Protokollen müssen neben den anwendungsspezifischen Anforderungen drei wichtige Punkte beachtet werden: Verzögerung, Fehlertoleranz und Effizienz.

Jede Anfrage in einem verteilten System an eine nicht-lokale Komponente braucht mehr Zeit als eine lokale Anfrage. Entwickler sollten also immer einplanen, dass bei zeitkritischen Anfragen das Netz einen erheblichen Störfaktor darstellen kann. Im Zweifelsfall heißt das, nicht-lokale Anfragen so selten wie möglich zu verwenden. Auf der anderen Seite ist auch immer die Frage, wann eine Anwendung keine Antwort mehr erwarten kann und wie sie mit diesem Timeout umgeht.

Eine unbekannte Größe in einem verteilten Szenario ist immer, ob eine Komponente nur sporadisch oder permanent nicht mehr erreichbar ist. Beim Designen eines Protokolls muss also auch darauf geachtet werden, dass der Zusammenhalt erhalten bleibt, auch wenn einmal eine einzelne Komponente ausfällt (vgl. Abschnitt 7.1.1).

Die Effizienz eines Protokolls ist der letzte wichtige Punkt, der noch beachtet werden muss. Besonders im Fall der Projektgruppe muss darauf geachtet werden, da nur begrenzt Ressourcen zur Verfügung stehen (vgl. Abschnitt 1.1.1). Redundanzen in der Protokollnachricht, Duplikate und der Overhead müssen in solchen Szenarien minimiert werden. Tiefer liegende Protokolle müssen so gewählt werden, dass auch hier die Anforderungen eingehalten werden können.

Protokoll-Beispiele Im Folgenden werden ein paar mehr oder minder bekannte Protokolle vorgestellt. Bis auf ein Beispiel aus dem Bereich der Transportprotokolle sind sie auf der Anwendungsebene angesiedelt.

XML und XML-basiert XML[3] steht kurz für e**X**tensible **M**arkup **L**anguage, was auch schon beschreibt, was dieses Format ist: Eine Sprache, die sich leicht erweitern lässt und, ähnlich wie bei HTML, mit Markups arbeitet, in diesem Falle mit Tags. XML-Files können recht einfach mit XML Schema[4] (auch kurz XSD) validiert werden. Dabei handelt es sich sozusagen um einen Bauplan für XML-Dateien, der selbst in XML verfasst wird. Für XML und XSD gibt es für fast jede Programmiersprache Parser beziehungsweise Generatoren, die einem Entwickler die Arbeit sehr vereinfachen können. Das Problem bei XML ist allerdings, dass durch die Tags viel Overhead entstehen kann, zusätzlich sind die Parser kein zu unterschätzender Zeitfaktor. Allerdings ist die Menschenlesbarkeit ein klarer Vorteil von XML.

Um zumindest bei der Übertragung keine zu hohe Datenrate zu erzeugen, wurde EXI[5] entwickelt, dass XML nicht text- sondern bitbasiert abspeichert/verschickt und damit einen effizienteren Austausch ermöglicht. Dabei wird die XML-Datei als eine Folge von Codes verschlüsselt, sodass häufige Bitfolgen kürzer kodiert werden. Zusätzlich kann noch ein Kompressionsschritt eingebaut werden. XML ist also insgesamt ein sehr generisches Transportprotokoll auf Anwendungsebene.

JSON RPC JSON RPC ist ein Protokoll zum entfernten Aufruf von Prozeduren, welches das JSON[6]-Format verwendet um Nachrichten auszutauschen. Um eine Prozedur am Server aufzurufen wird eine Nachricht verschickt, die spezifiziert, welche Prozedur mit welchen Parametern aufgerufen werden soll. Der Server führt diese entsprechend aus und schickt die Antwort in einer weiteren Nachricht an den Client zurück. Die JSON-Nachrichten könnten dabei auch durch ein beliebiges anderes Format (bspw. XML) ersetzt werden, aber JSON-Nachrichten sind üblicherweise kürzer, wodurch weniger Aufwand bei der Kommunikation entsteht.

Robot OS Messaging Das Robot OS[7] ist ein komplettes Framework für Roboter, um die Programmierarbeit von der Hardware zu abstrahieren. Es liefert sehr viele Treiber mit, interessant für die Projektgruppe ist hierbei allerdings das Messaging-Format. Das Aussehen von Nachrichten wird wie bei XML und XSD über ein Meta-File definiert, das aber ohne die Tags auskommt. Die einzelnen Zeilen bestehen nur aus Objektname und Objekttyp, sodass die Nachricht selbst einfach

[3] `http://www.w3.org/TR/REC-xml/`
[4] `http://www.w3.org/XML/Schema`
[5] `http://www.w3.org/TR/exi/`
[6] **J**ava**S**cript **O**bject **N**otation, `http://www.json.org`
[7] `http://www.ros.org`

zeilenweise bearbeitet werden kann und nur einen Wert beinhaltet, der über das Meta-File interpretiert wird. Das hält die Kommunikation so minimal wie möglich. Um diese Nachrichten zu parsen und zu generieren, muss allerdings das Robot OS komplett verwendet werden. Die Idee kann aber in Eigenarbeit in entsprechenden Tools implementiert werden.

ZeroMQ Zum Abschluss soll noch ein Protokoll der Transportebene vorgestellt werden. ZeroMQ[8] ist eine Alternative zum üblichen UDP/TCP. Es ist laut Aussage der Homepage genauso mächtig, kann also genauso mit Broadcasts und sicherer Übertragung umgehen wie das Original. ZeroMQ gibt es in vielen Sprachen und es ist für P2P- und Cluster-Netzwerke besser konfigurierbar als UDP/TCP. Hauptmerkmal sind asynchrones I/O und Kompatibilität zu fast allen Plattformen.

7.2 Interprozesskommunikation

Interne Kommunikation zwischen den Agenten (vgl. Abb. 3.3) kommt an zwei Stellen vor: Eine unidirektionale Verbindung erlaubt dem Data-Mining-Agenten die Sensordaten vom Payloadmanager abzurufen und eine bidirektionale Kommunikationsverbindung zwischen *CNI UxV Framework* und *Stream-Framework* erlaubt, die Steerings an die Ergebnisse des Data-Mining-Agenten anzupassen.

Um diese Kommunikation zu verwirklichen, gibt es verschiedene Möglichkeiten: Von Dateien auf dem Festspeicher, über gemeinsam genutzte Speicherbereiche bis zu Unix-Sockets. Da es problematisch sein kann, Dateien zu nutzen, wenn Schreibzugriffe von mehreren Prozessen gleichzeitig kommen und auch bei gemeinsamem Speicher auf die Synchronisation geachtet werden muss (neben dem eventuell zusätzlichen Problem, dass mehrere verschiedene Programmiersprachen verwendet werden), werden Unix-Sockets zur Kommunikation verwendet. Als Protokoll dient UDP, da bei Interprozesskommunikation nicht von Datenverlusten auszugehen ist, und diese, falls sie denn vorkommen, unproblematisch sind. Auf den Overhead, der bei Verwendung von TCP entstünde, kann hier also verzichtet werden.

Schließlich muss noch ein Datenformat festgelegt werden. Auch dafür gibt es wieder verschiedene Möglichkeiten, wie beispielsweise JSON, XML oder gar einfach Textdateien in einem selbst spezifizierten Format. Bei dieser Auswahl ist XML zu bevorzugen, da es einfach zu Parsen ist (fertige Parser existieren für jede relevante Programmiersprache) und die Möglichkeit bietet, das Format korrekter Nachrichten zu definieren. Dazu wird die Sprache *XML Schema* verwendet. Diese erlaubt es, die Struktur für XML-Dokumente vorzugeben. Es ist dann möglich, ein gegebenes XML-Dokument mit dem Schema zu vergleichen und auf diese Weise zu entscheiden, ob das Dokument wohlgeformt ist. In den folgenden drei Abschnitten wird kurz beschrieben für welche Nachrichten solche XML Schema-Spezifikationen benötigt werden und wie diese umgesetzt sind.

[8] `http://www.zeromq.org`

7.2.1 Spezifikation der Sensordatenübertragung

Zunächst ist es notwendig, die Nachrichten vom Payloadmanager des *CNI UxV Framework* zu spezifizieren. Mit Hilfe dieser Nachrichten sollen die Sensordaten an den Data-Mining-Agenten gegeben werden, um diese dann mittels der Data-Mining-Algorithmen zu verarbeiten. Dies sind 4 Datensätze: Eine Position, welche als UTM-Koordinaten gegeben wird, einen Timestamp, die Temperatur und die Lichtintensität in Lux. Diese Daten können in der Nachricht einfach aufeinander folgend gegeben werden. Die genaue Spezifikation ist durch Listing 7.1 gegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="measurement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="position">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="x" type="
                                xs:decimal"/>
                            <xs:element name="y" type="
                                xs:decimal"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="timestamp" type="xs:decimal"/
                    >
                <xs:element name="temp" type="xs:decimal"/>
                <xs:element name="lux" type="xs:decimal"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 7.1: XSD-Spezifikation der Nachrichten zur Übermittlung der Sensordaten.

7.2.2 Spezifikation der Nachbarschaftserkennung

Weiterhin muss der Data-Mining-Prozess häufig wissen, welche anderen Fahrzeuge innerhalb des Netzes als benachbart gelten. Dazu muss es eine Nachricht geben, die lediglich die IP-Adressen der Nachbarn enthält. Dies kann erreicht werden, indem für eine XML-Datei erlaubt wird, ausschließlich beliebig viele IP-Tags zu enthalten. Das entsprechende Schema ist in Listing 7.2 gegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="neighbors">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ip" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 7.2: XSD-Spezifikation der Nachrichten zur Übermittlung einer Liste von (in der Netzwerkarchitektur) benachbarten Robotern.

7.2.3 Spezifikation der Wegpunkteabfrage

Abschließend ist es notwendig den Teil der Kommunikation zwischen dem *CNI UxV Framework* und dem *Stream-Framework*, welcher die Anpassung der Steerings möglich macht, zu spezifizieren. In Abschnitt 6.3.2 wird erläutert, wie genau die Fahrzeuge sich neue Ziele suchen, ein Teil dieses Vorganges ist, beim Data-Mining-Agent anzufragen welche Gebiete erkundet werden sollen. Dazu gibt es, wie bereits in Abschnitt 5.3.5 erwähnt, drei mögliche Szenarien:

1. Es sollen Gebiete erkundet werden, über die noch keine Informationen existieren (Anfragetyp *discover*).
2. Es sollen Gebiete erkundet werden, für die bislang zu wenig Informationen existieren (Anfragetyp *modelimprovement*).
3. Das Fahrzeug benötigt Energie und es sollen daher Gebiete angegeben werden, die eine hohe Energiedichte aufweisen (Anfragetyp *topenergy*).

Benötigt werden also zwei verschiedene Nachrichten: Die Anfrage, welche die Steerings im *CNI UxV Framework* an den Data-Mining-Agent stellen, und die Antwort des Data-Mining-Agent.

Anfrage Bei der Anfrage an den Data-Mining-Agent gibt es nur eine wichtige Information: den Anfragetyp. Anhand dessen wählt der Data-Mining-Agent passende Gebiete aus dem Modell aus. Für diesen sind, entsprechend der vorherigen Beschreibung, die Werte *discover*, *modelimprovement* und *topenergy* erlaubt. Zusätzlich wird noch eine ID zur Anfrage hinzugefügt, damit die Antwort des Data-Mining-Agent eindeutig dieser Anfrage zugeordnet werden kann, da sie die gleiche ID enthält. Das entsprechende XML Schema ist in Listing 7.3 gegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="query">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="requestid" type="xs:integer"/
                    >
                <xs:element name="type">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="discover"/>
                            <xs:enumeration value="
                                modelimprovement"/>
                            <xs:enumeration value="topenergy"/>
                            <xs:enumeration value="fullmodel"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 7.3: XSD zur Spezifikation von Anfragen des *CNI UxV Framework* an das *Stream-Framework*, um Gebiete für das Steering zu erhalten.

Antwort Der Data-Mining-Agent muss in seiner Antwort genug Informationen liefern um die Antwort einer Anfrage zuzuordnen. Dazu wird die ID verwendet, welche dieselbe ist, wie in der Anfrage. Auf den Anfragetyp kann in der Antwort verzichtet werden, da alle Anfragen das gleiche Antwortformat erwarten: eine Liste von Gebieten, in welche sich das UGV bewegen soll. Diese werden als Rechtecke angegeben, wobei die zwei Eckpunkte als x- und y-Koordinaten innerhalb des Gitters angegeben werden, auf welchem die UGVs sich bewegen. Von diesen Gebieten dürfen beliebig viele in der Antwort enthalten sein, die Steering-Algorithmen wählen aus dieser Liste dann das passende Gebiet aus. Listing 7.4 enthält das vollständige XML Schema, welches diese Antworten definiert.

7.3 Kommunikation zwischen den Data-Mining-Agenten

Wie in Abschnitt 5.2 beschrieben, wird das bereits vorhandene *Stream-Framework* als Grundlage für die Implementierung der Data-Mining-Algorithmen verwendet. Dieses kann allerdings bislang nur für Stream-Data-Mining in statischen Netzwerken verwen-

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="answer">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="requestid" type="xs:integer"/
                    >
                <xs:element name="area" minOccurs="0" maxOccurs
                    ="unbounded">
                 <xs:complexType>
                     <xs:sequence>
                         <xs:element name="x" type="
                             xs:integer"/>
                         <xs:element name="y" type="
                             xs:integer"/>
                         <xs:element name="length" type="
                             xs:integer"/>
                     </xs:sequence>
                 </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 7.4: XSD zur Spezifikation von Antworten des *Stream-Framework* an das *CNI UxV Framework*, mit neuen Gebieten für das Steering.

det werden. Das Szenario in Abschnitt 1.1.1 beschreibt allerdings ein verteiltes Problem mit dynamisch wechselnden Nachbarn.

Das *Stream-Framework* nutzt für das Auflösen von Services von anderen Containern die native Java-RMI-Bibliothek. So kann z. B. eine einfache Queue benutzt werden, um Daten, die in Container A erzeugt werden, als Eingabe für einen Prozess in Container B einzusetzen. Ein Ziel der Data-Mining Untergruppe ist es, die Mechanismen des *Stream-Framework* so anzupassen, dass auch bei sich ändernden Nachbarschaftsbeziehungen die Kommunikation zwischen den Containern gewährleistet ist.

Im Folgenden werden zunächst einige RMI-Bibliotheken kurz vorgestellt und daraufhin begründet, welche am besten für die Zwecke dieser Projektgruppe geeignet ist. Anforderungen sind dabei unter anderem, dass das Verfahren wenig Overhead verursacht und die Übertragungsrate hoch ist. Abschließend werden einige Details zur Implementierung der Verteilung auf mehrere Container mit dieser Methode erläutert.

7.3.1 Betrachtete Methoden

Es gibt einige RMI Implementierungen für Java, welche unterschiedliche Anforderungen erfüllen. Interessant sind Methoden, die einen geringen Ressourcenverbrauch und wenig Kommunikations-Overhead haben. Vier solche Methoden werden in diesem Abschnitt vorgestellt und anschließend evaluiert: Java-RMI, SIMON, JSON RPC und Hessian.

Java RMI Java RMI[9] ist Oracles Standardimplementierung einer RMI-Methode. Entsprechend ist es auch die Methode, welche durch das *Stream-Framework* standardmäßig verwendet wird. Problematisch ist allerdings, dass Java RMI recht viel Overhead produziert und vergleichsweise kompliziert zu verwenden ist. Alle mit Hilfe von Java RMI initiierten entfernten Methodenaufrufe müssen durch einen try-catch-block geschachtelt werden, da eine Exception auftreten könnte. Dies ist zwar ein prinzipiell sinnvolles Vorgehen (schließlich müssen Verbindungsabbrüche in irgendeiner Form behandelt werden), macht den Code aber unübersichtlich.

SIMON Die zweite getestete Bibliothek ist SIMON[10]. Dabei handelt es sich um eine Modernisierung der Standard Java RMI-Bibliothek, wobei insbesondere auf einfache Nutzung wert gelegt wird. Mit SIMON ist es nicht notwendig, bei jeder Verwendung eines entfernten Objektes mögliche Exceptions zu behandeln. Weiterhin kann eine Klasse einfach im Netzwerk freigegeben werden, dazu muss lediglich eine Annotation hinzugefügt werden.

JSON RPC Wie bereits in Abschnitt 7.1.2 beschrieben ist auch JSON-RPC eine RMI-Methode, die natürlich auch innerhalb von Java verwendet werden kann. Gedacht ist JSON RPC allerdings als Format für Webanwendungen, weshalb es als Servlet[11] angewendet wird und demnach der Server auch einen Servlet-Container benötigt. Es wurden zwei verschiedene Implementierungen der JSON RPC Spezifikationen getestet (jsonrpc4j[12] und jpoxy[13]), allerdings ist bei beiden die Dokumentation unzureichend, was eine zufriedenstellende Nutzung kaum möglich macht.

Hessian Hessian[14] ist, ähnlich wie JSON RPC, in erster Linie als Web-Protokoll gedacht: Es wird also ein Servlet-Container benötigt und die Implementierung muss entsprechend auch einer Client-Server-Architektur folgen. Anders als JSON RPC wird nicht in Klartext kommuniziert, sondern ein eigenes Binärformat verwendet. Weiterhin

[9] `http://www.oracle.com/technetwork/java/javase/overview/index.html`
[10] **S**imple **I**nvocation of **M**ethods **O**ver **N**etwork, `http://dev.root1.de/projects/simon/`
[11] Servlets sind Java-Klassen, deren Instanzen innerhalb eines Webservers laufen und Anfragen von Clients entgegennehmen und bearbeiten.
[12] `http://code.google.com/p/jsonrpc4j/`
[13] `http://code.google.com/p/jpoxy/`
[14] `http://hessian.caucho.com`

ist Hessian ausgesprochen einfach zu verwenden: Eine Klasse muss lediglich die Klasse *HessianServlet* erben, und ist dann auch bereits im Netzwerk verwendbar. Es werden also weder eine Annotation noch aufwendige Behandlung von Exceptions benötigt.

7.3.2 Evaluation und Details der gewählten Methode

Aus den genannten vier Methoden soll jetzt diejenige ausgewählt werden, die am ehesten den Anforderungen der Projektgruppe entspricht: Wenig Overhead, schnelle Übertragung und vor allem einfach zu handhaben. Java RMI und JSON RPC erfüllen diese Anforderungen nicht. Java RMI fordert neben der Trennung von Interface und Implementierung (die durch die Service-Architektur des *Stream-Framework* bereits gegeben ist) die Deklaration einer RemoteException für alle Methoden. Das führt zu viel Programmier-Overhead in Bezug auf das Abfangen der Exceptions. Die Dokumentation der Java JSON RPC Implementierungen sind so unvollständig, dass die Suche nach einer dritten Umsetzung für die Projektgruppe nicht als sinnvoll erachtet wird. Nach Tests mit SIMON und Hessian stellt sich heraus, dass Hessian geringfügig schneller ist als SIMON.

Die Wahl von Hessian (in der Version 2) wird durch eine kleine Studie von Daniel Gredler untermauert, der einige Java Remote-Protokolle miteinander verglichen hat. In [24] hat er neben XML-basierten Protokollen auch Java-RMI und Hessian 2 auf die Antwortzeit mit einer Liste von Objekten untersucht. Die XML-basierten Umsetzungen sind aufgrund der Struktur der Nachrichten erwartungsgemäß signifikant langsamer, während bei den binären Protokollen Hessian 2 mit am besten abschneidet. Bei einer Liste mit maximal 250 Elementen bleibt die Antwortzeit unter 10 ms. Neben der Reduzierung des Verarbeitungsaufwandes gegenüber XML-basierten Protokollen ist die Struktur von Hessian, die hauptsächlich aus Proxys besteht, sehr flach. Hessian 2 wird somit als RMI-Protokoll eingesetzt.

Funktionsweise Hessian funktioniert nach dem Client/Server-Schema. Die Serverseite muss zum einen das *HessianServlet* erweitern, zum anderen ein Interface implementieren. Die Client-Seite kann dann über die *HessianProxyFactory* eine HTTP-Anfrage stellen, ein Objekt, dass dieses Interface implementiert, von einer bestimmten URL zu bekommen. Die Daten werden dann über Streams ausgetauscht, die in der üblichen HTTP-Anfrage mitgesendet werden. Die Idee mit den Interfaces passt auch deshalb zum Data-Mining dieses Projekts, da das *Stream-Framework* auf *Services* aufbaut, die ebenfalls durch Interfaces spezifiziert und damit von der eigentlichen Implementierung getrennt sind. Der Austausch zwischen den *Containern* findet ausschließlich über *Services* statt[15], auf die entfernt zugegriffen werden soll.

[15] bis Version 0.9.7 des *Stream-Framework*

7.3.3 Implementierung

Die folgenden Abschnitte befassen sich mit der Umsetzung des RMI-Verfahrens für die Projektgruppe. Kernstück ist dabei ein *RemoteNamingService*, der Hessian benutzt und aus drei Teilen besteht: Dem (lokalen) Client, dem Server bzw. Servlet und einem Proxy. Da in einem dynamischen P2P Netzwerk für den Anwender nicht klar ist, welche Nachbarn gerade verfügbar sind, benötigt man zusätzlich einen *Service*, der dafür ausgelegt ist, Daten an alle bekannten Nachbarn zu verschicken. Dieser *MultiService* ist dabei auf seinen *NamingService* angewiesen.

***NamingService* allgemein** Ein *NamingService* hat grundsätzlich zwei wichtige Funktionen: Die Registrierung von *Services* und das Nachschlagen eben dieser, auch Lookup genannt. Zusätzlich gibt es die Möglichkeit, sich alle verfügbaren Services als Liste ausgeben zu lassen, um so zu prüfen, ob ein bestimmter Service ansprechbar ist.

Beim *RemoteNamingService* kommt eine *call*-Methode hinzu, die ähnlich wie bei einem *InvocationHandler*[16] funktioniert: Sie erlaubt es, die Methode eines Services nur über die Servicereferenz, den Methodennamen und die Argumente aufzurufen, ohne selbst die Referenz auflösen zu müssen und dann die Methode aufzurufen, und ist daher sehr generisch verwendbar. Die vererbungstechnischen Zusammenhänge der Hessian-Umsetzung sind in Abb. 7.2 dargestellt. Der *HessianNamingService* hat zusätzlich eine Methode, um die Nachbarn abfragen zu können.

Client Der *HessianNamingClient* nutzt den *DefaultNamingService* des *Stream-Framework* für isolierte Container und erweitert ihn um die Möglichkeit, die momentan vorhandenen Nachbar-Container abzufragen. Dazu gibt es zwei separate Threads, die je nach Situation entweder eine Meldung mit Name, IP, Port und *NamingService*-Protokoll verschicken und solche Meldungen empfangen, oder eine Liste von IP-Adressen im XML-Format (vgl. Listing 7.2) bekommen und so die Nachbar-Container bestimmen. Wenn ein nicht-lokaler *Service* aufgelöst werden soll, wird der entsprechende Proxy des zugehörigen *RemoteNamingService* herangezogen, wie in Abb. 7.3 dargestellt.

Server und *Service*-Proxys Um den *NamingService* auch von außen zugänglich zu machen, wird ein *HessianServlet* benötigt. Der *HessianNamingServer* erweitert diese Klasse also und gibt Anfragen an den zugehörigen *HessianNamingClient* weiter. Dazu läuft das Servlet in einem jetty[17]-Servlet-Container, der sehr viel leichtgewichtiger als z. B. ein Apache Tomcat Webserver ist. Damit aber nicht jeder Service einzeln als Servlet verfügbar sein muss, wird die *call*-Methode des *RemoteNamingService* und das Proxy-Konzept von Java genutzt: Wenn ein entfernter Service angefragt wird, wird ein *Proxy* erstellt, der mittels eines *InvocationHandler* die Aufrufe an den Service in einen *call*-Aufruf für den jeweiligen *RemoteNamingService* umwandelt, siehe Abb. 7.4.

[16] siehe *java.lang.InvocationHandler* in der Java-API
[17] `http://jetty.codehaus.org/jetty/`

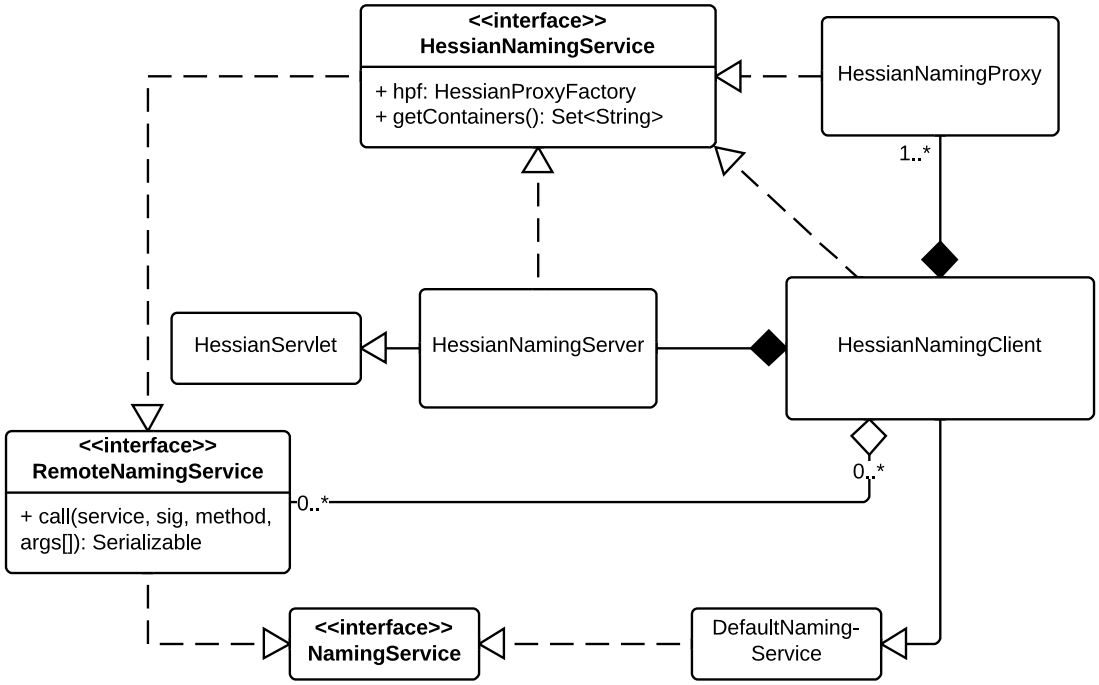


Abbildung 7.2: Vererbung und Zusammenhang des HessianNamingServices

Proxy Als Mittelstück zwischen dem lokalen *NamingService* und dem entfernten *HessianNamingServer* wird der *HessianNamingProxy* genutzt. Zum einen verwaltet er eine Hessian-Referenz auf den *HessianNamingServer*, an die Anfragen außer der *lookup*-Methode weitergeleitet werden; zum anderen wird beim lookup der oben erwähnte Service-Proxy erstellt. Der Service-Proxy wird direkt auf der Client-Seite erstellt, damit es nicht nötig ist, die Anfrage an den entfernten *HessianNamingServer* zu stellen. Es wird lediglich anhand der Service-Liste überprüft, ob der Service tatsächlich existiert.

Verteilung der Algorithmen Eine Herausforderung bei den verwendeten verteilten Algorithmen ist, dass diese sich zwischen den unterschiedlichen Agenten absprechen müssen. In jedem Container läuft entsprechend ein *Service* mit dem gleichen Namen. Um aber Daten an jeden dieser anderen Container zu versenden, und das an eine möglicherweise wechselnde Anzahl oder Menge von Agenten, stellt ein gewisses Hindernis dar. Die Lösung ist der von der PG entwickelte *MultiService*.

MultiService Durch die Funktionalität des *HessianNamingService*, die gerade verfügbaren Nachbarn abzufragen, kann ein *MultiService* stets die entsprechenden Container gezielt ansprechen. Ein *MultiService* besteht dabei aus dem (lokalen) Namen des Services und der Klasse, die in den einzelnen Containern für den Service verwendet wird, der angesprochen werden soll. Der *MultiService* spricht dann auf allen verfüg-

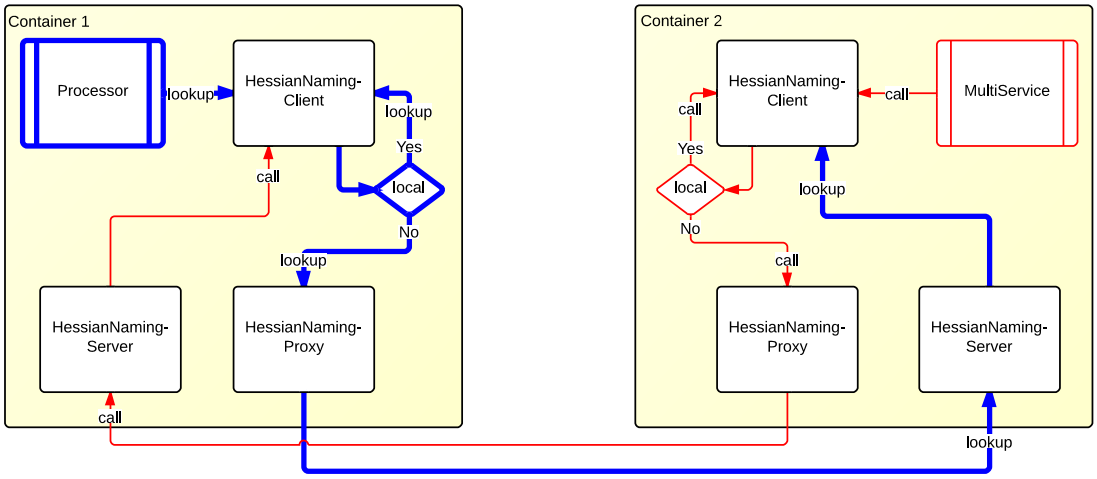


Abbildung 7.3: Kontrollfluss des NamingService mit Hessian. Links (blau, dick) Service Lookup, rechts (rot, dünn) MultiService call.

baren Containern den benannten Service an und führt jeweils die Methode aus, die auf dem *MultiService* aufgerufen wurde. Dabei wird wieder die *call*-Methode des *RemoteNamingService* genutzt. Der Kontrollfluss eines Aufrufs auf einem *MultiService* ist in Abb. 7.3 dargestellt.

Eine lokale Nutzung des *MultiService* ist auch möglich, sodass also mehrere Services in einem Container angesprochen werden können. Dabei wird die laufende Nummer (beginnend bei 0) an den angegebenen Namen des Services angehängt, in Übereinstimmung mit dem Erstellen von Kopien eines Prozesses (vgl. Abschnitt 5.2.1). Um Deadlocks im im lokalen Fall zu vermeiden, wird die *call*-Methode synchronisiert.

Um einen Überblick über den Absender und die Empfänger zu behalten, werden die *Service*s so konzipiert, dass ein *Data*-Objekt als Parameter vorkommt. So können *@sender* und *@receiver* eingetragen werden, um zu verhindern, dass ein *Service* sich selbst noch einmal Daten zuschickt.

Eine Einschränkung beim *MultiService* ist, dass nur schreibende Methoden (also mit Rückgabetyp *void* oder *boolean*) verwendet werden können, da das Aggregieren der Ergebnisse für allgemeine Datentypen nicht trivial ist und eine erhöhte Deadlock-Gefahr durch das Warten auf einen der beteiligten Container gegeben ist.

Erweiterung des *Stream-Frameworks* Anders als ein normaler Service, der in Interface und Implementierung unterteilt ist, gibt es für einen *MultiService* im Regelfall keine implementierende Klasse. Mit der Standard-Initialisierung des *Stream-Framework* kann also kein Java-Objekt zu einem *MultiService* erzeugt werden. Es bedarf also einer Erweiterung des *Stream-Framework* um ein „MultiService“-XML-Tag. Wie in Abschnitt 5.2.3 beschrieben, ist das möglich, indem ein entsprechender *ElementHandler* an das *Stream-Framework* übergeben wird.

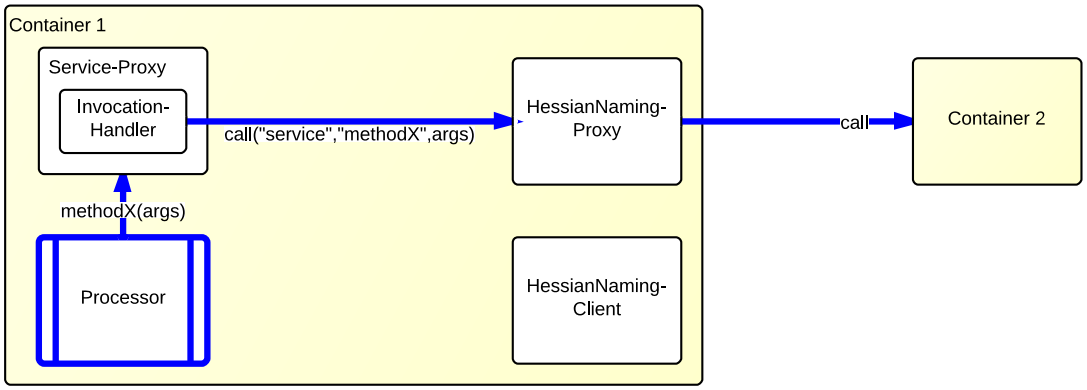


Abbildung 7.4: Kontrollfluss beim Aufruf über einen Service-Proxy.

Die Attribute spiegeln die oben beschriebenen Bestandteile eines *MultiService* wieder. Zwingend erforderlich sind die Attribute *class* und *ref*, die die Klasse und den Namen festlegen. Optional kann durch verwenden des Boolean-Attributes *local* bestimmt werden, dass lokale Kopien von Services angesprochen werden sollen. Ist ein lokaler *MultiService* gewünscht, muss außerdem in *copies* angegeben werden, wie viele solcher Kopien vorhanden sind.

Ein *MultiService*-Objekt ist dann ein Java-*Proxy*, der alle *Service*-Interfaces implementiert, die auch die angegebene Klasse erweitert. Mittels des *MultiInvocationHandler* wird dann die Funktionalität der remote-Aufrufe wie oben beschrieben umgesetzt.

7.4 Kommunikation zwischen den Mobilitäts-Agenten

In diesem Abschnitt wird die Kommunikation der Roboter untereinander näher betrachtet. Die Roboter haben verschiedene Nachrichtentypen mit denen sie die unterschiedlichen Informationen austauschen, die zum reibungslosen Ablauf der Strategie notwendig sind. Im Folgenden werden diese aufgezählt und genauer beschrieben:

GeoData Diese Nachricht wird dazu verwendet, allen anderen Robotern mitzuteilen, an welcher Position sich der sendende Roboter aktuell befindet.

TopologyData Mit dieser Nachricht teilen die Roboter allen anderen Robotern mit zu welchen Robotern sie eine Verbindung haben und ggf. auch die entsprechende Verbindungsqualität. Dies ist hilfreich, da die Roboter auch im Mesh kommunizieren können und dazu wissen müssen, ob sie mit einer Verbindung zu einem Roboter auch andere Roboter erreichen können zu denen sie keine direkte Verbindung haben.

UAVStatus In dieser Nachricht werden Informationen ausgetauscht, die vor allem für die fliegenden Roboter wichtig sind. Dies ist z.B. der Flugstatus oder genauere

Positionsdaten inklusive z.B. der Neigung. Zusätzlich werden auch Agenten ID und Schwarm ID übermittelt.

PlatformStatus Mit dieser Nachricht tauschen die Roboter Informationen darüber aus, um was für eine Plattform es sich handelt, dass heißt auf welchem System sie gerade basieren. Dies könnte z.B. die Software-in-the-Loop Simulation sein oder auch ein Roboter der auf dem RoBoard läuft. Zusätzlich werden auch die IP-Adresse und Betriebsystem Informationen wie z.B. Speicherauslastung oder Kernel Version ausgetauscht.

VehicleInfromation Dies ist eine Nachricht die für die in der Projektgruppe entwickelte Strategie genutzt wird. In dieser Nachricht kommunizieren die Roboter die Informationen über die von ihnen zu besuchenden Gebiete. Dies ist notwendig, um abzustimmen, wer welches Gebiet erkundet um damit Überschneidungen zu vermeiden.

NextTravelPoint In dieser Nachricht wird ein Zielpunkt für die nächste Erkundung vorgegeben. Sie wird beim Steering Cooperative-Area-Exploration genutzt um den anderen Robotern mitzuteilen, dass der Roboter den Punkt erreicht hat und nun ein neues Ziel angesteuert wird.

8 Analyse und Bewertung des Gesamtsystems

Bis zu diesem Punkt wurde in erster Linie erläutert, wie die einzelnen Komponenten jedes Agenten funktionieren und zusammenarbeiten. Dabei wurden diese einzelnen Komponenten auch schon teilweise für sich genommen bewertet, allerdings lässt dies alleine noch keine ausreichende Aussage über die Leistungsfähigkeit des entwickelten Gesamtsystems zu. Inhalt dieses Kapitels ist daher die Bewertung des entstandenen Gesamtsystems. Zu diesem Zweck wird zunächst in Abschnitt 8.1 erläutert, welche Methoden für die Analyse und Bewertung angewandt werden. Dazu zählt eine Erklärung der verwendeten Simulationsumgebungen und warum diese benötigt werden, sowie die Definition des Analyseszenarios und der Vergleichsmetriken. Daraufhin werden die Ergebnisse der Bewertung in Abschnitt 8.2 aufbereitet, ausgewertet und interpretiert.

8.1 Methodik

Um die Analyse nachvollziehen zu können, muss zunächst erläutert werden, wie sie durchgeführt wurde. Dazu wird in diesem Abschnitt zunächst beschrieben, welche Möglichkeiten der Simulation vorhanden sind, da die Analyse hauptsächlich innerhalb einer Simulationsumgebung stattfindet. Ein Grund dafür war, dass es zeitlich nicht machbar ist, eine ausreichende Datenbasis für die Bewertung innerhalb realer Umgebungen zu erstellen. Es werden daher zunächst die Begriffe Hardware-in-the-Loop und Software-in-the-Loop erläutert und darauf folgend die verwendete Testlaufumgebung und die Parametrisierung der Algorithmen vorgestellt. Schließlich werden die eingesetzten Metriken und Vergleichsmaße erklärt. Dabei wird deutlich, welche Algorithmen und Erkundungsstrategien gegeneinander verglichen werden, und welche Eigenschaften dabei im Fokus liegen.

8.1.1 In-the-Loop-Simulationen

Folgender Abschnitt beschreibt eine Methode, welche für die Bewertung des Systems besonders wichtig ist: Software- bzw. Hardware-in-the-Loop-Simulationen ermöglichen es, das System vorab zu testen, ohne dabei das Gesamtsystem in der Zielumgebung einsetzen zu müssen. Der Vorteil ist, dass durch diese Simulationen bestimmte Szenarien beliebig oft reproduziert werden können. Durch diese Wiederholbarkeit können Auffälligkeiten an Soft- und Hardware, wie Programmierfehler oder technische Mängel,

besser nachgewiesen werden. Müsste das Gesamtsystem direkt in seiner Realumgebung ausgewertet werden, wäre diese Reproduzierbarkeit aufgrund von Sensordaten bzw. Umwelteinflüssen nicht mehr gewährleistet. Nachfolgend werden die beiden Methoden weiter erläutert.

Software-in-the-Loop Bei einer Software-in-the-Loop-Simulation werden die entwickelten Algorithmen nicht auf ihrer Zielhardware ausgeführt, sondern auf einem Entwicklungsrechner ohne erwähnenswerte Ressourcenbeschränkungen. Es soll ausschließlich die Funktionalität der Software verifiziert werden, die später auf dem Zielsystem eingesetzt wird. Dazu gehört auch, dass bereits eine reale Kommunikation stattfindet. Durch diese Simulationsmethode können hardwarespezifische Probleme vorläufig vernachlässigt werden und Herausforderungen, wie z.B. eine Portierung auf die Zielhardware, werden erst in der Hardware-in-the-Loop-Simulation adressiert. Software-in-the-Loop-Simulationen eignen sich gut für Machbarkeitsstudien, da schon früh der Aufwand der geforderten Algorithmen bewertet werden kann, ohne von den üblichen Problemen der technischen Seite eines Projekts aufgehalten zu werden.
Zu Beginn der Software-in-the-Loop-Simulationen lag unser Schwerpunkt bei der Überprüfung des Nachrichtenaustausches. Mit Hilfe der Simulation ließ sich gut feststellen, ob alle für die Strategie (vgl. Abschnitt 6.3.2) benötigten Nachrichten versendet und auf der Gegenseite empfangen wurden.
Nachdem die volle Funktionsfähigkeit der Kommunikation sichergestellt war, konnten die einzelnen Phasen der Strategie überprüft werden. Durch das Setzen von passenden Breakpoints und geeigneten Debug-Ausgaben ließen sich die Zustände der Strategie gut überwachen und ein eventuelles Fehlverhalten konnte schnell erkannt werden.

Hardware-in-the-Loop Die Hardware-in-the-Loop-Simulation stellt ein Konzept bereit, mit welchem das Gesamtsystem auf der Zielhardware getestet werden kann. Dieser Prozess soll Feldtests vorausgehen und das Zusammenspiel zwischen der entwickelten Software und der eingesetzten Hardware nachahmen. Während in der Software-in-the-Loop-Phase beispielsweise die Kommunikation über eine Ethernetverbindung stattfindet, wird bei der Hardware-in-the-Loop-Methode die reale Funkhardware verwendet.
In der Projektgruppe konnte durch eine Hardware-in-the-Loop-Simulation das erwartete Verhalten der Fahrzeughardware nachgewiesen werden. Dazu gehört die Funktionalität der Licht- und Wärmesensoren und die des GPS-Empfängers, sowie das Zusammenspiel mit den dafür entwickelten Treibern.

8.1.2 Testlaufumgebung

Dieser Abschnitt befasst sich mit der Konfiguration der Hard- und Software für die Testläufe und wie diese durchgeführt werden. Die Ergebnisse der Testläufe werde dann in Abschnitt 8.2 diskutiert und ausgewertet.

Hard- und Software-Konfiguration Der Projektgruppe stehen mehrere virtuelle Maschinen zur Verfügung. Auf diesen werden nach der Software-in-the-Loop Methodik Testläufe durchgeführt. Anders als im vorhergehenden Abschnitt beschrieben, werden Ressourcenbeschränkungen angewandt, so ist der Arbeitsspeicher auf 512 MB beschränkt. Auf diese Weise ist es möglich eine virtuelle Maschine jederzeit durch reale Hardware, welche die genannte Einschränkung aufweist, zu ersetzen (vgl. Hardware-in-the-Loop aus Abschnitt 8.1.1). Die Prozessorleistung der virtuellen Maschinen wird nicht begrenzt, jedoch zeigte sich in Abschnitt 4.8, dass die Recheneinheit der Experimentalplattform nicht übermäßig ausgelastet ist.

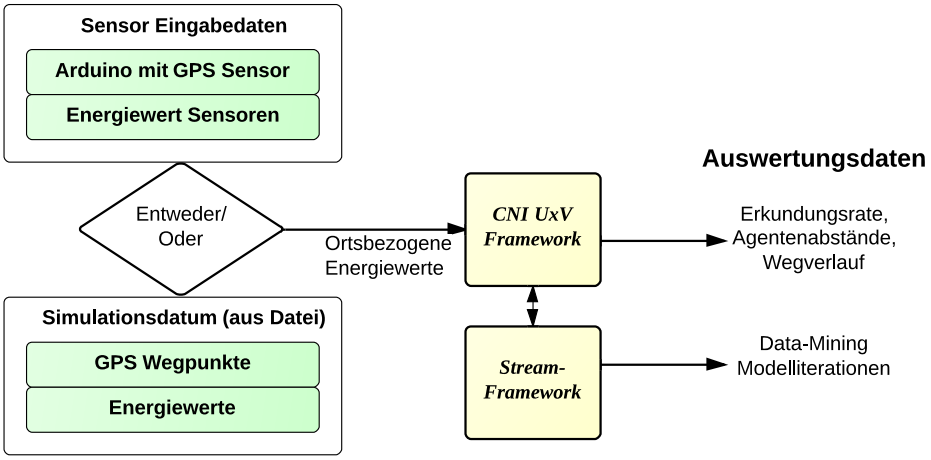


Abbildung 8.1: Das *CNI UxV Framework* kann in einen Simulationsmodus umgeschaltet werden. Ortsbezogene Energiewerte stammen dann aus einer Sensorsimulation.

Die Plattformen sind über ein LAN miteinander verbunden, das *Mesh-Network* wird durch die Empfangsstärke emuliert. Die dafür notwendigen RSS-Werte werden vom Mobilitäts- und Kommunikationsagenten im Simulationsmodus direkt berechnet bzw. interpretiert.

Der Mobilitäts- und Kommunikationsagent wechselt automatisch vom Sensor- in den Simulationsmodus, sofern eine Datei mit Simulationsdaten vorgefunden wird (vgl. Abschnitt 6.2.3). Abbildung 8.1 veranschaulicht den entsprechenden Datenfluss. Da der Data-Mining-Agent von der Laufzeitumgebung vollkommen unabhängig ist, weil alle Daten und Nachbarinformationen direkt vom Mobilitäts- und Kommunikationsagenten gesendet werden, wird dieser mit normalen Einstellungen gestartet.

Um die Testläufe auswerten zu können, schreiben der Mobilitäts- und Kommunikationsagent und der Data-Mining-Agent Daten mit. Beim Mobilitäts- und Kommunikationsagent sind das die Erkundungsrate, Abstände zu den anderen Agenten (sowohl als euklidische Distanz als auch über die RSSI-Metrik), GPS-Trace und die TX Rate

in Mb/s. Der Data-Mining-Agent speichert in regelmäßigen Abständen das aktuelle Modell und merkt sich, wie oft jede Zelle des Missionsgebietes besucht wurde. In Abschnitt 8.1.3 wird beschrieben, wie diese Daten zur Auswertung genutzt werden.

Testlaufszenario Entsprechend des Referenzszenarios ist ein 150×130 m^2 großes Missionsgebiet gegeben. Um verschiedene Lichtverhältnisse zu testen, gibt es jeweils zehn Karten mit Parkplatz- bzw. Clusterdaten, die wie in Abschnitt 5.3.1 beschrieben erzeugt werden. Jede dieser Karten wird fünfmal verwendet, um etwaige Ausreißer auszugleichen und eine allgemeinere Diskussion zuzulassen.

Die Steerings, die bei den Testläufen jeweils zum Einsatz kommen und in Abschnitt 6.3 näher erläutert werden, sind:

Mission-Planning Da dieses Steering immer die gleiche Route abfährt, wird jede Karte nur einmal genutzt.

Cooperative-Area-Exploration Der Schwarm fährt in Formation zufällige Punkte an.

SUN Search In zwei Versionen bzgl. der Treffpunktwahl; Version 1 sucht ein Gebiet, sodass die addierten Distanzen der einzelnen Roboter möglichst gering sind (L_1 Norm). Version 2 bevorzugt das Gebiet, bei dem die längste Distanz über alle Roboter die kleinste ist (L_∞ Norm).

Um möglichst zeitnah Ergebnisse zu erzeugen, wurde die Geschwindigkeit der Fahrzeuge für die Testläufe auf $10,8$ km/h erhöht; somit wurde die effektive Laufzeit eines Durchgangs auf 10 min reduziert. Anhaltspunkt für diese Zeit war das MP, um eine gewisse Vergleichbarkeit herstellen zu können. Die Sendeleistung der anderen Fahrzeuge wird im Simulationsmodus berechnet und dann um 30 dBm verringert, um Verbindungsabbrüche im Mesh zu simulieren. Das Fahrzeug hat einen Wahrnehmungsbereich von 1 m^2, alle $0,5$ s wird eine Messung durchgeführt.

Die Data-Mining-Algorithmen wurden wie folgt parametrisiert:

Distributed DBSCAN Es wird eine ϵ-Umgebung von $2,45$ vorgegeben, in der mindestens sechs Punkte des gleichen Energielevels liegen müssen; nach drei gefundenen Micro-Clustern werden diese an die Nachbarn kommuniziert.

Hierarchical Heavy Hitter Updates des Modells werden nach 150 vom Data-Mining-Agent erfassten Daten geteilt. Die Heavy Hitter Rate liegt bei 0.015.

Es kommen vier Agenten (sprich virtuelle Maschinen) zum Einsatz, die mittels eines Skriptes synchronisiert werden. Das ist nötig, da die Testläufe automatisiert gestartet werden und Interferenzen durch noch laufende Prozesse vom Vorlauf vermieden werden sollen. Zusätzlich ist es bei CAE und MP nötig, dass die Bodenstation die Wegpunkte liefert. Dazu muss nach dem Start der Agenten ein Kommando an die Bodenstation gegeben werden, was ebenso über das Skript synchronisiert wird.

8.1.3 Metriken und Vergleichswerte

Folgender Abschnitt enthält die Erläuterung wichtiger Analysedetails: Es wird dargestellt, mit welchen Metriken der Vergleich der im vorigen Abschnitt genannten Erkundungsstrategien und Data-Mining-Algorithmen stattfindet.

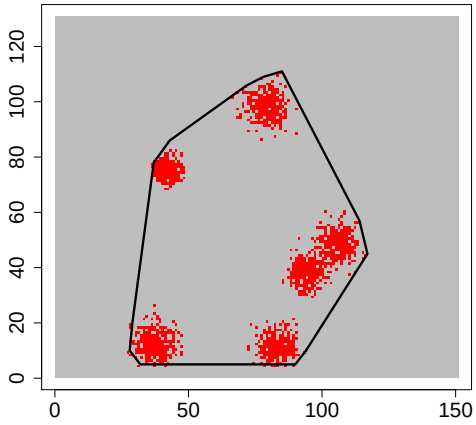
Varianzmetrik für Data-Mining Ziel ist es, herauszufinden, ob der Einsatz des Data-Minings einen erkennbaren Nutzen hat. Dazu ist es nicht sinnvoll, nur die reine Erkundungsrate zu betrachten, da diese bei einem deterministischen Mission-Planning-Algorithmus bei gleicher Dauer des Laufs immer besser sein wird, da dieser darauf ausgelegt ist, so wenige Zellen mehrfach zu besuchen wie möglich. Stattdessen soll herausgefunden werden, ob SUN Search mit Unterstützung des Data-Mining-Agenten ein besseres Gesamtbild der Umgebung liefert. Beispielsweise ist es zu bevorzugen, wenn in jedem Bereich der Karte zumindest einige helle Cluster gefunden wurden, während Mission-Planning in der gleichen Zeit vielleicht nur die linke Hälfte der Karte umfassend erkundet hat, in der rechten Hälfte aber keinerlei Daten vorliegen. Es wird also eine Metrik benötigt, die diesen Unterschied verdeutlichen kann.

Sei dazu angenommen, es gebe 20 Messwerte, die zwei gleich große, klar getrennte Cluster bilden. Werden alle 10 Punkte eines Clusters gefunden, so unterscheiden sich diese aufgrund der Lokalität nur wenig voneinander, ihre Varianz ist also gering. Vergleicht man diese Punkte nun allerdings mit den Punkten des zweiten Clusters, so ist der Unterschied deutlich größer, was in höherer Varianz resultiert, auch wenn nur 5 Punkte jedes Clusters gefunden werden. Zur Bewertung wird daher die Varianz der 1-Norm aller gefundenen hellen Punkte verwendet:

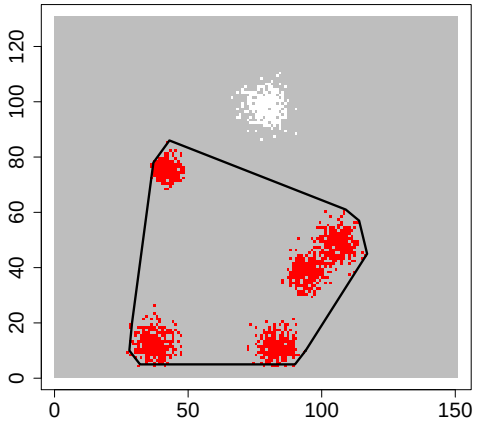
$$\mathrm{Var}(z)\,,\ \text{mit}\, z \in \mathbb{R}^n \,\text{und}\, z_i = x_i + y_i$$

Ist $\mathrm{Var}(z)$ groß, so sind die gefundenen hellen Punkte weit verteilt und die Karte entsprechend umfassender erkundet. Abbildung 8.2 verdeutlicht diesen Zusammenhang. Da das Modell des Data-Mining-Agenten alle zwei Minuten für jeden Lauf gespeichert wird, ist dies auch für die einzelnen Zeitabschnitte nachvollziehbar. Es wird erwartet, dass sich auf diese Weise zeigt, dass SUN Search wesentlich schneller ein nutzbares Ergebnis hat als Mission-Planning.

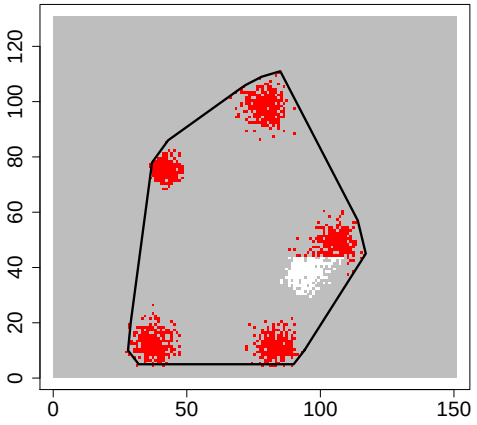
Erkundungsrate Trotz der Aussage, dass für den Nutzen des Data-Mining die reine Erkundungsrate nicht allein interessant ist, muss ein guter Erkundungsalgorithmus dennoch auch dabei gute Ergebnisse vorweisen. Ist das nicht der Fall, besteht zu hohe Gefahr, große, aber interessante Bereiche der Umgebung nicht zu betrachten. Daher wird die Erkundungsrate und die Häufigkeit, mit der einzelne Zellen besucht werden, herangezogen. Ziel ist es, die Erkundungsrate hoch und die Häufigkeit der Zellbesuche niedrig zu halten. Dabei ist Mission-Planning optimal für beide Eigenschaften, da die Strategie ausschließlich auf möglichst effiziente Erkundung ausgelegt ist. SUN Search mit Data-Mining-Unterstützung sollte aber zumindest nicht wesentlich schlechter als Cooperative-Area-Exploration abschneiden.



(a) Varianzmetrik = 1918, Fläche =6705



(b) Varianzmetrik = 1450, Fläche =5215



(c) Varianzmetrik = 2266, Fläche =6705

Abbildung 8.2: Einfluss der Verteilung der gefundenen Cluster: (b) Das Entfernen eines Clusters am Rand der konvexen Hülle reduziert sowohl die Varianzmetrik als auch den Flächeninhalt der konvexen Hülle; (c) wird ein Cluster innerhalb der konvexen Hülle entfernt, bleibt der Flächeninhalt gleich, aber die Varianzmetrik wird größer im Vergleich zu den Ursprungsdaten (a)

Genau wie die Varianzmetrik, ist auch bei der Erkundungsrate der Zeitverlauf interessant: Ist es bei einer Strategie häufig der Fall, dass die Erkundungsrate ab einem gewissen Punkt einbricht, weil bereits erkundete Zellen noch einmal besucht werden? Um zumindest einen Hinweis darauf zu bekommen, wird überprüft ob die Erkundungsrate in allen Strategien in etwa linear steigt. Ändert sich diese Steigung an einem gewissen Punkt, so ist das ein Indiz dafür, dass ab diesem Zeitpunkt nur noch selten neue Gebiete erkundet wurden. Der Grund dafür kann natürlich sein, dass große Teile der Karte bereits erkundet wurden. Ist dies nicht der Fall, so wird es nötig den Algorithmus genauer zu untersuchen, da dieses Verhalten unerwünscht ist.

Kommunikationsaufwand Ein weiteres Ziel ist es, die Kommunikation zwischen den Agenten möglichst gering zu halten. Dabei sollten insbesondere die verwendeten Data-Mining-Algorithmen weniger Daten verursachen als notwendig wäre, würden alle unverarbeiteten Sensordaten versendet. Um dies zu prüfen wird die Kommunikation eines Laufs aufgezeichnet und daraufhin ausgewertet, wieviele Bytes durch den Data-Mining-Agent zwischen den UGVs im Vergleich zum Versenden der unverarbeiteten Sensordaten entsteht.

Weitere Messgrößen Abschließend sollen noch drei weitere Eigenschaften erwähnt werden, die ebenfalls Aussagen über die Qualität der Erkundung ermöglichen:

Überdeckung der Modelle Die Frage, die sich hier stellt ist, ob die einzelnen Data-Mining-Agenten auch tatsächlich ähnliche Modelle haben. Dies ist wichtig, da die Modelle in einer kompakten Form übermittelt werden. Um dies zu prüfen, wird die Überdeckung der lokalen Modelle berechnet: Jedes lokale Modell ordnet jedem Punkt eine Intensität zwischen 0 (keine Energie) und 2 (hohe Energie) zu. Um die Übersichtlichkeit zu erhöhen werden diese zunächst in je zwei Modelle aufgetrennt, eines für hohe und eines für mittlere Energie. Da es in diesen aufgeteilten Modellen keine Unterscheidung verschiedener Energieniveaus mehr gibt, werden sie außerdem auf die Werte 0 (keine Energie) und 1 (Energie vorhanden) normalisiert. Um nun die Überdeckung zu berechnen werden die Intensitäten der lokalen Modelle aller vier Agenten aufsummiert, was zu zwei Modellen mit Werten zwischen 0 und 4 führt. Eines dieser Modelle stellt die Divergenz der Cluster mit mittlerer Energie dar, das andere die der Cluster mit hoher Energie. Im besten Fall kommen ausschließlich die Werte 0 und 4 vor, denn dann wären alle lokalen Modelle identisch. Je mehr Werte zwischen 0 und 4 vorkommen, desto stärker unterscheiden sich die lokalen Modelle. In Abbildung 8.3 sind beispielhaft die vier Modelle der Agenten nach 8 Minuten Erkundung mit der Cooperative-Area-Exploration-Strategie dargestellt, gefiltert auf die Punkte mit hoher Energie. Leichte Unterschiede sind zu erkennen, welche in der aufsummierten Darstellung (Abb. 8.4) deutlicher zum Tragen kommen.

Überdeckung mit den Originaldaten Die Analyse in einer Simulation mit vollständig bekannten Testdaten hat den Vorteil, dass sich die Möglichkeit ergibt, die von

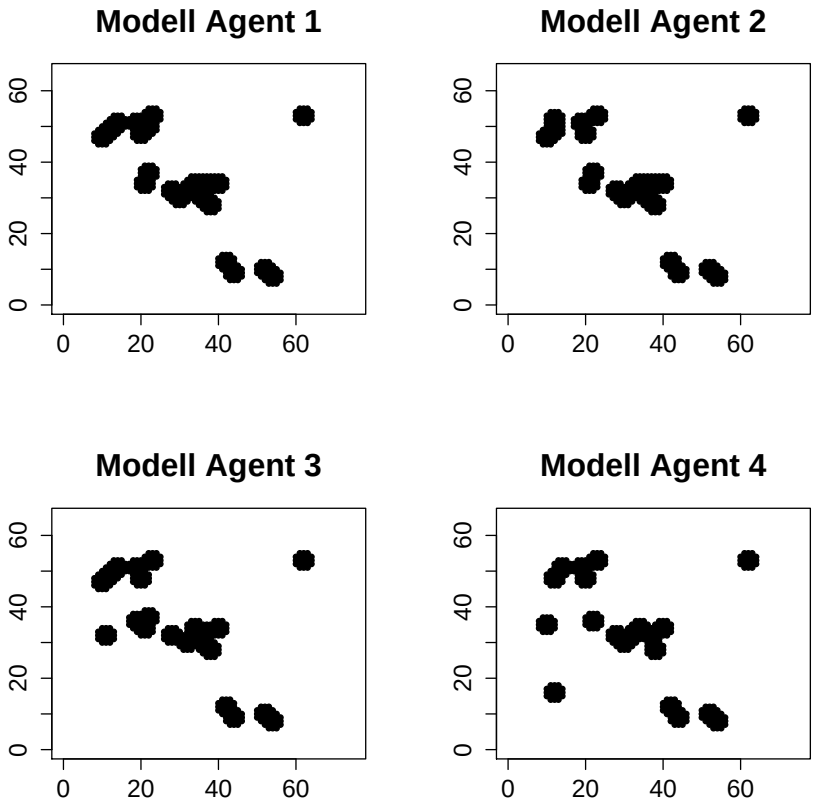


Abbildung 8.3: Punkte mit hoher Energie in den lokalen Modellen nach 8 Minuten Erkundung mit Cooperative-Area-Exploration

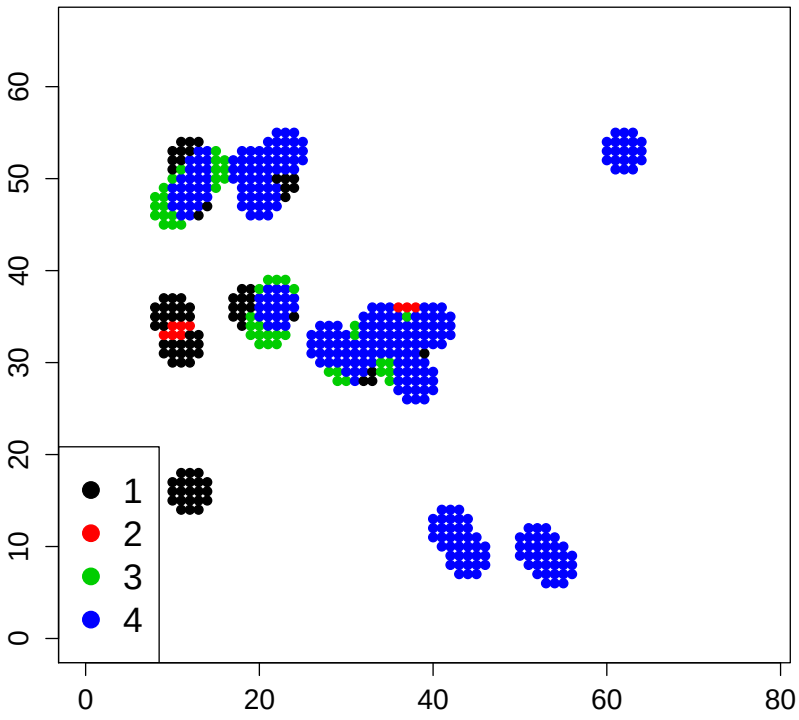


Abbildung 8.4: Überdeckung der Modelle aus Abb. 8.3; insgesamt stimmen alle Modelle in 306 (blau) von 448 Punkten überein

den Agenten entwickelten Modelle mit den Originaldaten zu vergleichen. Dazu werden die Modelle und die Originaldaten übereinander gelegt und auf diese Weise geprüft, wie stark diese sich unterscheiden. Es ist weiterhin auch möglich in Prozent anzugeben, wie viele helle Zellen vom Modell korrekt erkannt werden.

Konvexe Hülle der Cluster Schließlich wird noch die konvexe Hülle um die gefundenen Cluster berechnet, um herauszufinden, wie groß die erkundete Fläche ungefähr ist. In Kombination mit der Varianz der Cluster kann diese aufschlussreiche Informationen über die Struktur der Daten bieten. So spricht eine große Fläche bei gleichzeitig großer Varianz für weit entfernt voneinander liegende Cluster. Andererseits spräche eine kleine Varianz bei gleichzeitig großer Fläche eher für ein großes, aber auch sehr gut erkundetes Gebiet. Der schlechteste Fall liegt bei kleiner Fläche und kleiner Varianz vor, dann wären nur wenige, dicht beieinander liegende Punkte gefunden worden.

8.2 Analyse der erhobenen Kenngrößen

Der folgende Abschnitt betrachtet die durch die Testläufe gewonnenen Ergebnisse im Detail. Durch die Auswertung der gewonnenen Daten werden die entwickelten Strategien und Algorithmen bewertet und es wird beantwortet ob die gesetzten Ziele, nämlich verbesserte Modellbildung, eine verbesserte Erkundungsrate und ein geringerer Kommunikationsaufwand durch den Einsatz von Data-Mining-Algorithmen erreicht werden. Zunächst werden die verwendeten Kenngrößen und Metriken ausgewertet, anschließend wird in Abschnitt 8.2.3 die Erkundungsrate näher untersucht und abschließend beschreibt Abschnitt 8.2.4 den Effekt des Data-Mining auf den Kommunikationsaufwand.

8.2.1 Auswertung der Kenngrößen

Im Folgenden werden die Vergleichswerte aus Abschnitt 8.1.3 im Detail analysiert. Dabei wird hier nur Version 2 des SUN Search-Algorithmus betrachtet, da die Unterschiede zwischen den beiden Versionen sich als sehr gering herausstellten (siehe dazu auch Abschnitt 8.2.3).

Überdeckung Zunächst wird überprüft, wie gut die Modelle der einzelnen Agenten tatsächlich übereinstimmen oder ob sie stark voneinander abweichen. Dies ist ein wichtiger erster Schritt um sicherzustellen, dass Aussagen, die über alle Ergebnisse getroffen werden, nicht durch große Schwankungen verzerrt werden. Um die Vergleichbarkeit der einzelnen Ergebnisse zu ermöglichen, wird folgende Skalierung verwendet.

Zunächst wird ermittelt, in wie vielen Punkten die Modelle der vier Agenten für je einen Simulationslauf komplett übereinstimmen. Anschließend wird der Quotient aus

$$\frac{\#\text{ vollständige Übereinstimmungen}}{\#\text{ Cluster Agent i}}$$

berechnet. Dieser Quotient nimmt den Wert 1 an, wenn die Punkte des lokalen Modells eines Agenten komplett mit den Punkten übereinstimmen, die alle anderen Agenten auch in ihrem Modell haben und wird kleiner, je mehr das Modell von dieser gemeinsamen Basis abweicht. Ein Wert größer 1 kann nicht auftreten, denn es würde bedeuten, dass das lokale Modell weniger Punkte besitzt, als die Übereinstimmung aller vier Modelle. In Abbildung 8.5 sind die Ergebnisse für die drei untersuchten Erkundungsstrategien dargestellt.

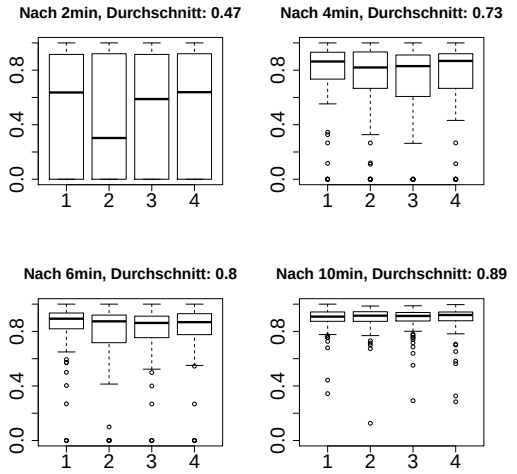
Anhand der beiden Strategien Cooperative-Area-Exploration und SUN Search ist gut zu sehen, wie die Modelle sich über die einzelnen Zeitabschnitte hinweg immer stärker ähneln. Die etwas schlechteren Überdeckungsraten bei SUN Search, gerade nach vier und sechs Minuten, lassen sich damit erklären, dass die Agenten bei dieser Strategie nicht unbedingt immer miteinander in Kontakt stehen müssen und dann ihre Erkenntnisse nicht austauschen können. Es zeigt sich auch, dass die Kombination aus Mission-Planning und DBSCAN nicht gut funktioniert: Insgesamt werden beim Mission-Planning meist nur sehr wenige Cluster gefunden, wie später noch zu sehen ist, und wenn dann einzelne Micro-Cluster nicht verteilt werden, fällt dies sehr stark ins Gewicht. In Tabelle 8.1 ist zu sehen, dass die Überdeckungsrate niedriger ist als bei den anderen beiden Algorithmen und fast keine Verbesserung mit größerer Erkundung aufzeigt.

Strategie	**2 min**	**4 min**	**6 min**	**10 min**
CAE	0,47	0,73	0,80	0,89
SUN Search	0,25	0,64	0,72	0,89
MP	0,62	0,59	0,6459	0,69

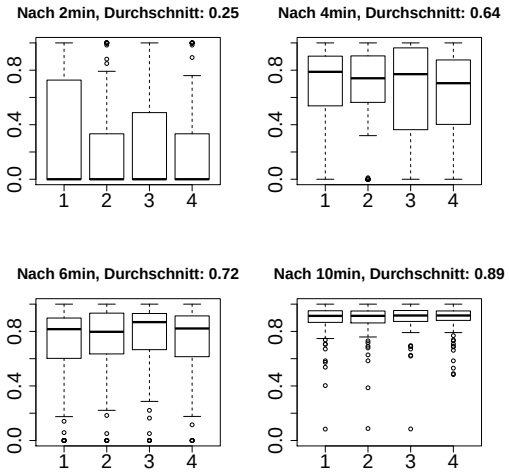
Tabelle 8.1: Übersicht der durchschnittlichen Überdeckungen der lokalen Modelle, zu den jeweiligen Zeitpunkten

Flächenabdeckung Der Vergleich der Fläche der umspannenden konvexen Hülle um die gefundenen Cluster zeigt sich kein deutlichen Unterschiede zwischen den Algorithmen. Für diesen Vergleich wird die Fläche der lokalen Modelle mit der Fläche der ursprünglichen Daten verglichen. Ein Verhältnis von 1 zeigt dabei nicht an, dass alle Cluster gefunden wurden, sondern lediglich dass alle Randpunkte gefunden wurden. Die SUN Search- und Cooperative-Area-Exploration-Strategie sind hier sehr ähnlich, wie in Abbildung 8.6 zu sehen ist. Die relativ niedrigen Ergebnisse lassen sich dadurch erklären, dass es auf beiden Kartentypen häufig Punkte in den Randbereichen gibt, welche eine sehr große Referenzfläche aufspannen. Die Agenten hingegen erkunden den Rand weniger intensiv. Die Mission-Planning-Strategie kommt hier aufgrund des engen Verbunds der Agenten nur auf einen sehr geringen Wert, zumal auch hier wieder zu beachten ist, dass insgesamt kaum Cluster gefunden werden.

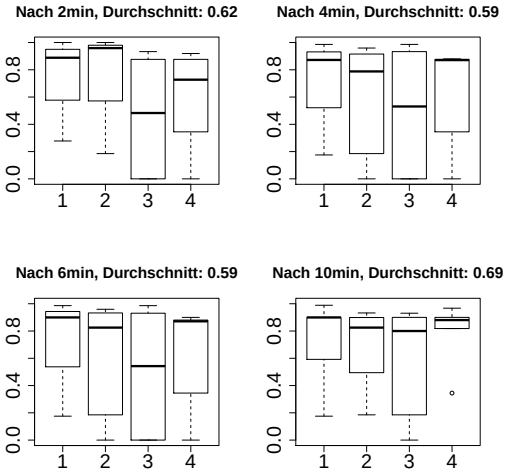
Clustervarianz Für die in Abschnitt 8.1.3 erläuterte Varianz-Metrik ist eine Normierung nicht sinnvoll, da die Werte nicht im direkten Verhältnis zu den Referenzwerten



(a) Überdeckung CAE, Niveau 2

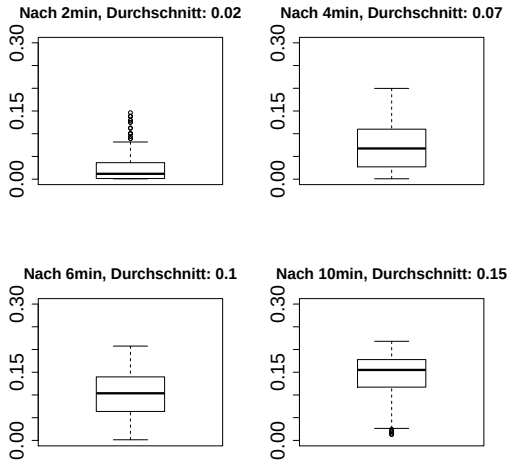


(b) Überdeckung SUN Search, Niveau 2

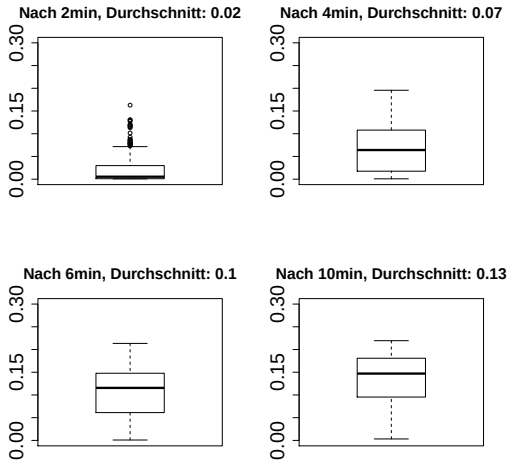


(c) Überdeckung MP, Niveau 2

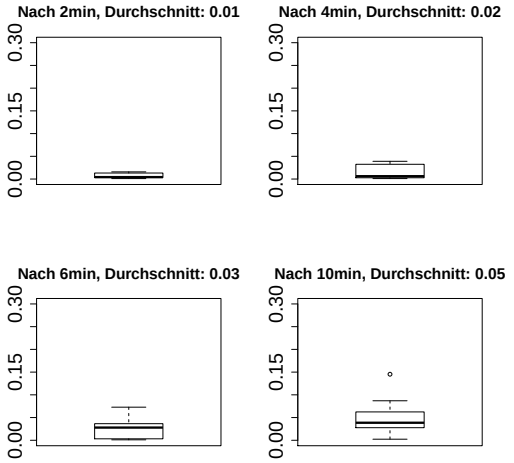
Abbildung 8.5: Entwicklung der Übereinstimmung der Modelle für helle Cluster-Punkte: der CAE-Algorithmus (a) weist eine etwas bessere Überdeckungsrate auf, als der SUN Search Algorithmus (b), MP (c) ist am schlechtesten



(a) Flächenverhältnis CAE



(b) Flächenverhältnis SUN Search



(c) Flächenverhältnis MP

Abbildung 8.6: Entwicklung der eingenommen Fläche der gefunden hellen Cluster: die CAE-Strategie (a) und SUN Search (b) weisen eine ähnliche Rate auf, die Mission-Planning-Strategie (c) ist deutlich schlechter

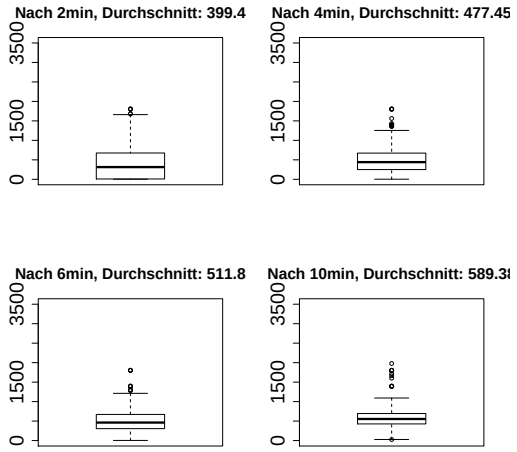
der ursprünglichen Daten stehen, weil die Anzahl der Cluster nicht mit einfließt. In Abbildung 8.7 sind deswegen die Ergebnisse ohne Normierung gegenübergestellt. Hier zeigt sich kein klarer Unterschied zwischen den Verfahren. Allerdings ist zu erkennen, dass die SUN Search-Strategie sehr viel mehr Ausreißer aufweist, bei denen die Varianz sehr hoch ist. Dies kann ein Indiz dafür sein, dass die verteilte Erkundung mit zwischenzeitlichen Verbindungsabbrüchen leichte Vorteile bringt.

Wie beschrieben sind die Werte der Varianz-Metrik zwar nicht über alle Datensätze hinweg vergleichbar, jedoch lässt sich durchaus ein Vergleich zwischen verschiedenen Ergebnissen auf der gleichen Karte führen. Die Abbildungen 8.8 und 8.9 stellen genau dies für einige Parkplatz-, respektive Clusterdatensätze dar. Die Daten geben, unter Berücksichtigung der Probleme die insbesondere Mission-Planning mit dem verwendeten Clusteringalgorithmus hat, Aufschluss über die Verteilung der UGVs auf die Umgebung: Gerade auf den parkplatzähnlichen Datensätzen zeigt sich, dass SUN Search sehr schnell auf eine hohe Varianz kommt, häufig schneller als CAE und Mission-Planning. Dies ist ein Indiz dafür, dass SUN Search sehr gut darin ist, schnell einen groben Überblick über die Umgebung zu schaffen. Dieser muss danach nur noch verfeinert werden.

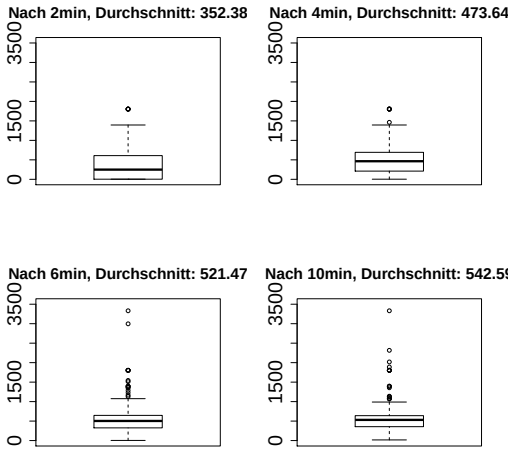
Gefundene Datenpunkte Letztendlich ist es auch von Interesse, wie viele Datenpunkte mit mittlerer oder hoher Energie die einzelnen Strategien gefunden haben. In diesem Fall lässt sich das Ergebnis wieder mit ursprünglichen Daten in Relation setzen, wie Abbildung 8.10 zeigt. Die auf den ersten Blick sehr guten Ergebnisse müssen jedoch kritisch betrachtet werden. Offenkundig gibt es einige Ausreißer, welche ein Verhältnis von gefundenen zu tatsächlichen hellen Punkten größer als 1 aufweisen. Dies erklärt sich dadurch, dass der DBSCAN-Algorithmus um ein Cluster-Zentrum herum alle Punkte diesem Cluster zuweist. Dadurch kann es an Randbereichen von größeren Konzentrationen heller Punkte zu Fehlklassifikationen kommen. In Datensätzen mit sehr wenigen Datenpunkten fällt dieser Fehler natürlich stärker ins Gewicht. Jedoch sind die Raten von 55% für Cooperative-Area-Exploration bzw. 44% für SUN Search relativ gut, insbesondere wenn sie mit der Erkundungsrate in Relation gesetzt wird, welche im Anschluss in Abschnitt 8.2.3 betrachtet wird.

Ausführliche Auswertung einer Simulation Die folgende Untersuchung soll die einzelnen Schritte der vorherigen Auswertungen veranschaulichen und das Verhalten sowie die Ergebnisse der einzelnen Erkundungsstrategien im Detail analysieren. Es ist zu beachten, dass der ausgewählte Durchlauf nicht repräsentativ ist. Stattdessen wird ein möglichst guter und aussagekräftiger Fall beschrieben. Betrachtet werden die beiden bereits in Kapitel 5.3.1 vorgestellten Testdatensätze (Abb. 5.13).

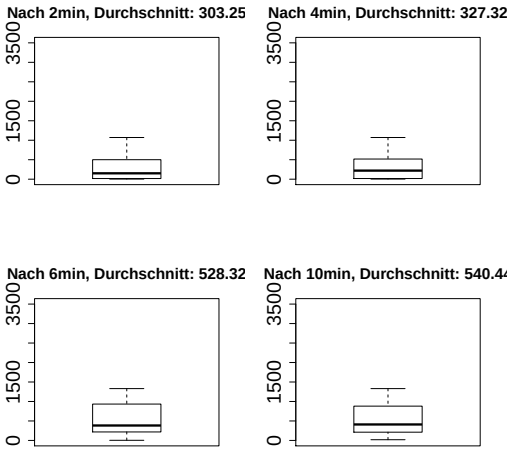
Man erkennt deutliche Unterschiede zwischen den einzelnen Strategien. Durch die kompakte Fahrweise beim Cooperative-Area-Exploration sehen alle Agenten in etwa die gleichen Werte, was zwar zu vielen aber auch sich oft überlappenden Clustern führt. In Abbildung 8.11(a) sind daher auf einem relativ kleinen Gebiet viele Cluster konzentriert. Die Einzelgänger-Erkundung beim SUN Search dagegen sorgt für mehrere verteilte Cluster, die nicht sehr konzentriert sind, sondern höchstens im weiteren



(a) Cluster-Varianz CAE

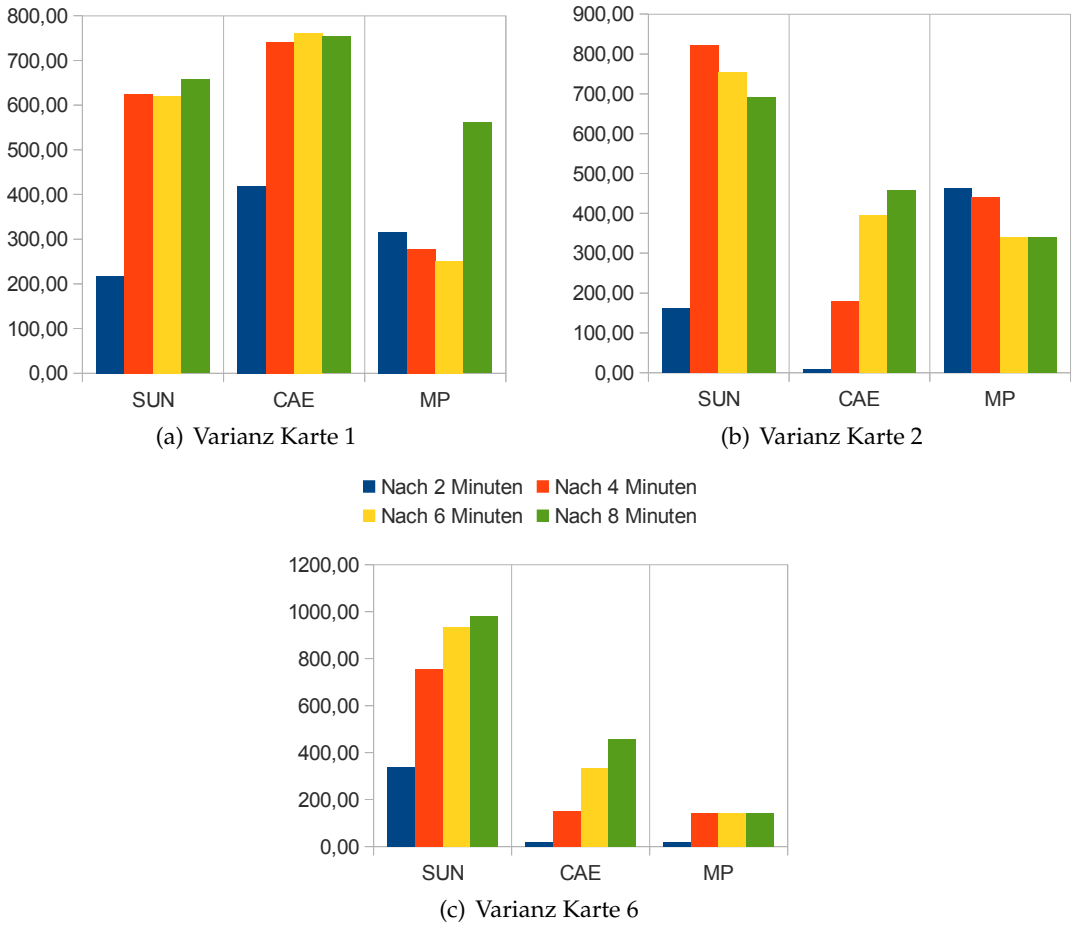


(b) Cluster-Varianz SUN Search



(c) Cluster-Varianz MP

Abbildung 8.7: Entwicklung der Cluster-Varianz für gefundene helle Cluster

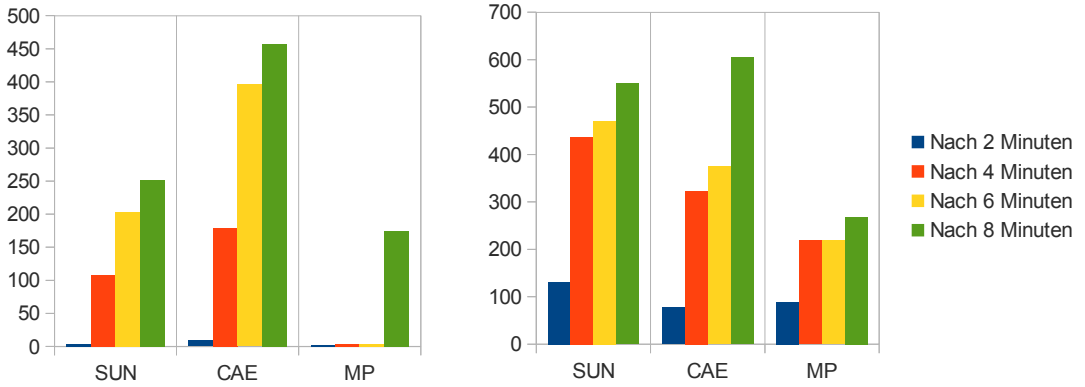


(a) Varianz Karte 1

(b) Varianz Karte 2

(c) Varianz Karte 6

Abbildung 8.8: Vergleich der Varianzen für drei Parkplatzähnliche Datensätze



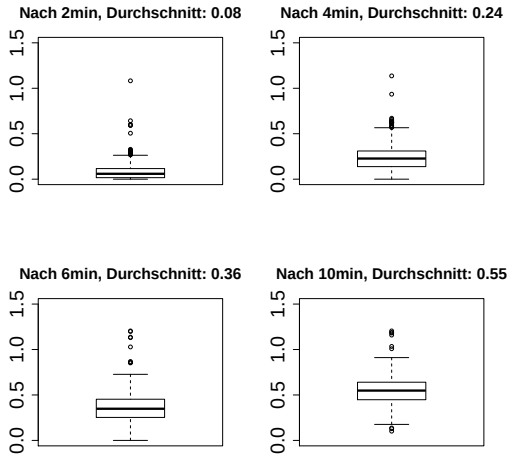
(a) Varianz Karte 2 (b) Varianz Karte 10

Abbildung 8.9: Vergleich der Varianzen für zwei Cluster-Datensätze

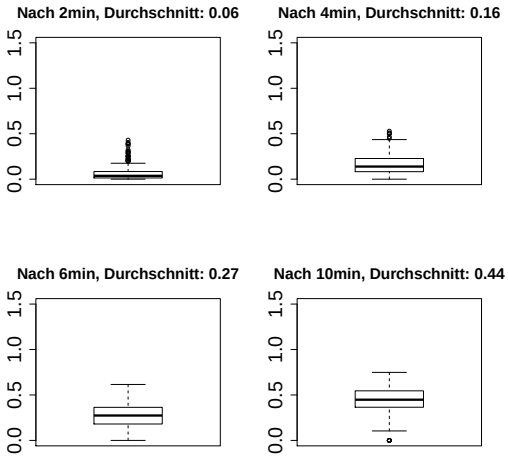
Verlauf ergänzt und erweitert werden. Ein Beispiel für eine solche Erweiterung zeigt sich in Abbildung 8.11(b) am linken Rand im Schritt von 6 Minuten zu 10 Minuten. Das Mission-Planning schließlich findet am linken Rand, also direkt nach dem Start, den stark ausgeprägten hellen Bereich. Durch die Fahrweise liegen gerade am Anfang die erste Auf- und Abwärtsbewegung eines Agenten nahe beieinander und bieten so dem DBSCAN die Möglichkeit, Micro-Cluster zu bilden. (vgl. dazu auch Abb. 8.12(c)). Im weiteren Verlauf werden nur noch sehr wenige Cluster gefunden, bevorzugt aber dort, wo Wendepunte der Agenten liegen oder eine Auf- und Abwärtsbewegung sehr nahe beieinander gefahren wurde (vgl. Abb. 8.11(c)).

Für den finalen Zustand des Modells ist auch interessant zu betrachten, welche Gebiete der Agent selbst erkundet hat. In Abbildung 8.12 ist zu sehen, dass der Agent auch Cluster aus Gebieten in sein Modell aufgenommen hat, die er selbst nie gesehen hat. Außerdem werden gleich mehrere Schwächen des Systems sichtbar. Es zeigt sich, dass der gewählte Clustering-Algorithmus Schwächen hat, wenn die Stichproben in einer geraden Linie gesammelt werden, denn dann werden mit der gewählten Parametrisierung keine Cluster gebildet. Dies ist der Hauptgrund, warum die Agenten bei der Mission-Planning-Strategie trotz ihrer organisierten und gleichmäßigen Fahrweise kaum Cluster in das Modell aufnehmen, obwohl entsprechende Gebiete befahren werden.

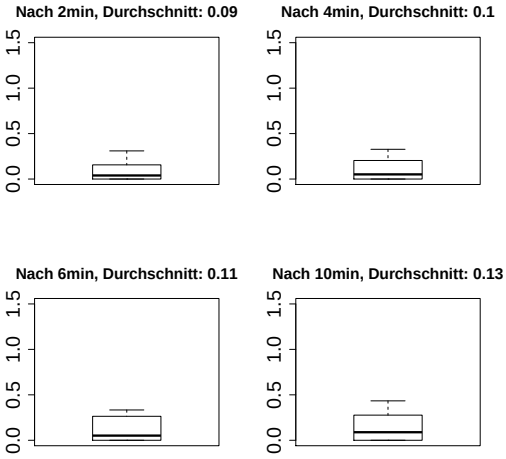
Die Pfade der Agenten in Verbindung mit den gefundenen Clustern lässt auch direkte Rückschlüsse auf das Fahrverhalten zu. So kann im Fall von Cooperative-Area-Exploration in Abbildung 8.12(a) beobachtet werden, dass alle Cluster in Reichweite der abgefahrenen Strecke liegen, woraus man schließen kann, dass die Agenten immer sehr nah beieinander waren und eine ähnliche Strecke abgefahren sind. Beim SUN Search dagegen gibt es teilweise deutliche Diskrepanzen zwischen befahrenem Gebiet und gefundenen Clustern, was auf eine bessere Aufteilung der Erkundungsgebiete hindeutet (vgl. Abb. 8.12(b)).



(a) Gefundene helle Cluster CAE

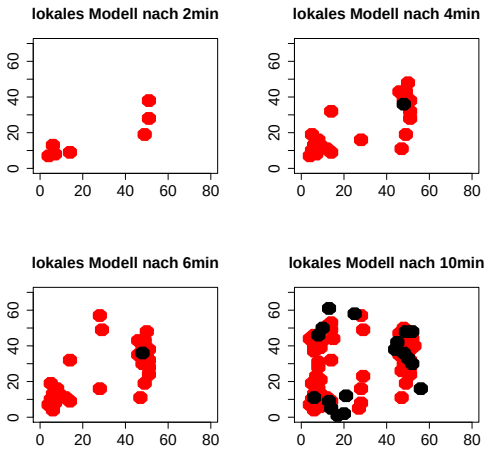


(b) Gefundene helle Cluster SUN

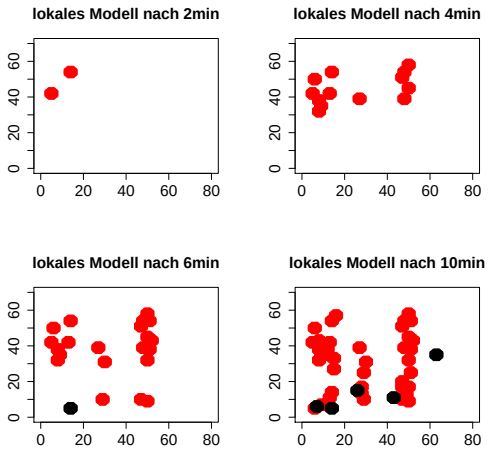


(c) Gefundene helle Cluster MP

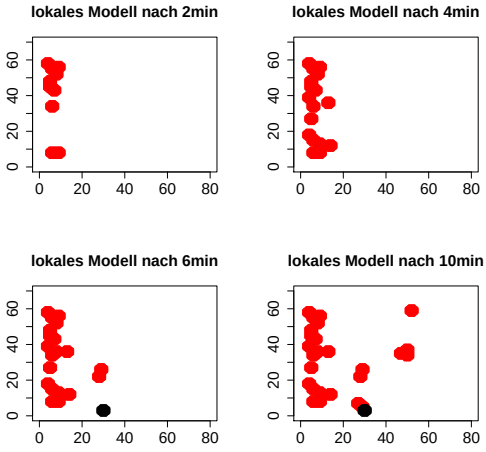
Abbildung 8.10: Verhältnisse der insgesamt gefundenen hellen Cluster: Werte größer 1 entstehen durch die Unschärfe des DBScan an Randbereichen von größeren Clustern



(a) Entwicklung des Modells bei CAE

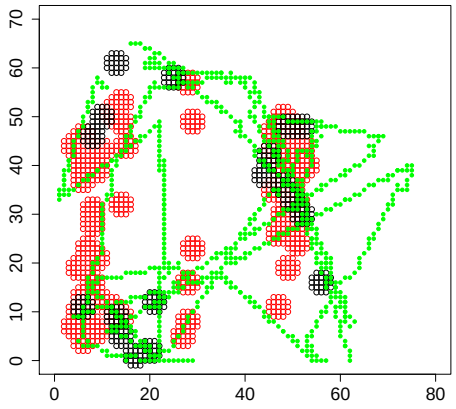


(b) Entwicklung des Modells bei SUN Search

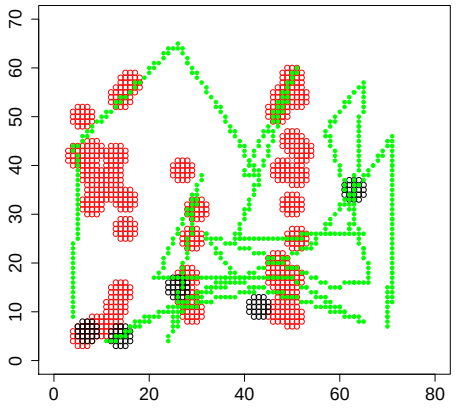


(c) Entwicklung des Modells bei MP

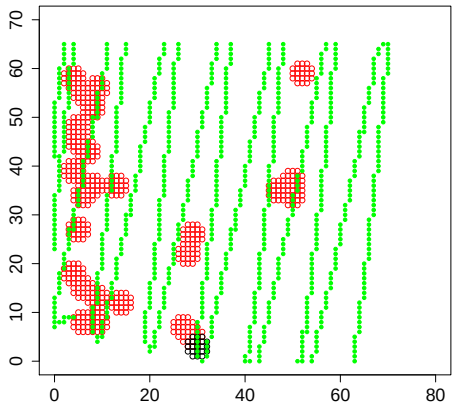
Abbildung 8.11: Entwicklung des lokalen Modells: rote Punkte sind markierte sehr helle Punkte, schwarz sind Punkte mittlerer Intensität



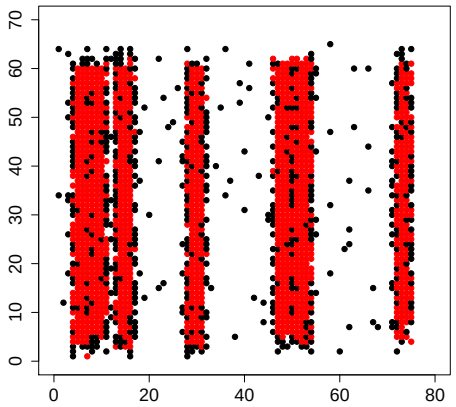
(a) Finales Modell bei CAE



(b) Finales Modell bei SUN Search



(c) Finales Modell bei MP



(d) Alle Cluster der Ausgangsdaten

Abbildung 8.12: Bekannte Cluster des finalen Modells (rot = hohe Energie, schwarz = mittlere Energie) mit eingezeichnetem Pfad der vom Agenten erkundeten Punkte (grün) und im Vergleich alle relevanten Punkte der Ausgangsdaten 8.12(d)

8.2.2 Nutzen des Data-Mining

Folgender Abschnitt befasst sich mit der Frage, ob die Verwendung von Data-Mining bei der Erkundung einen positiven Effekt bietet. Wie bereits in Abschnitt 8.1.3 beschrieben wird dazu nicht die reine Erkundungsrate betrachtet, sondern, wie die Varianz der gefundenen hellen Punkte sich im Laufe der Zeit verhält. Insbesondere ist es wichtig, dabei nicht nur das Endergebnis der drei Strategien zu betrachten, denn zu dem Zeitpunkt hat bspw. Mission-Planning bereits die komplette Karte abgefahren. Insbesondere im Anbetracht größerer Karten und des angedachten Szenarios (bei dem ja keine explizite Erkundung stattfindet, sondern die Daten stattdessen eher nebenbei gesammelt werden) ist es daher wichtig, auch den Verlauf zu betrachten: Ab wann bieten die einzelnen Erkundungsstrategien bereits eine recht umfassende Karte, welche nicht nur einen Teilbereich darstellt? Wie in Abschnitt 8.2 dargestellt sehen die Ergebnisse objektiv folgendermaßen aus: Die Sun Search Strategie zeigt durchaus die Tendenz schnell über die Karte verteilt Cluster zu finden. Im Schnitt weisen die damit erstellen Modelle eine etwas höhere Varianz auf und insbesondere in einigen Fällen ist sie deutlich höher, als bei den anderen beiden Strategien.

Insgesamt ergibt sich, dass der Einsatz von Data-Mining-Methoden durchaus einen positiven Nutzen haben kann, auch wenn dieser Effekt nicht sonderlich stark ausfällt. SUN Search findet Cluster etwas schneller als Mission-Planning und ist besser darin, die Agenten über die Karte zu verteilen. Nachteilig wirkt sich leider die gewählte Data-Mining-Methode aus: Es hat sich herausgestellt, dass das Density-Stream-Clustering im betrachteten Szenario und mit den gewählten Parametern nicht so gut funktioniert wie erwartet. Dennoch hat SUN Search bereits mit den gefundenen Clustern gute Varianzwerte, weshalb davon ausgegangen wird, dass die Nutzung eines Clustering-Algorithmus, der besser auf die vorliegende Situation abgestimmt ist, auch das Ergebnis verbessert.

8.2.3 Erkundungserfolg

Nachdem im vorherigen Abschnitt diskutiert wurde, welche Probleme sich mit dem Clustering gezeigt haben, ist als nächstes zu klären, ob SUN Search bei der Erkundungsrate kompetitiv mit anderen Strategien ist. Dabei ist festzustellen, dass die prozentuale Erkundungsrate tatsächlich mit der Cooperative-Area-Exploration vergleichbar ist. Dies ist zunächst einmal ein Erfolg, denn es zeigt, dass das Modell, welches die Data-Mining-Agenten erstellen, gut funktioniert, um Empfehlungen zur Erkundung von Gebieten zu geben. Abbildung 8.13 zeigt die durchschnittliche Erkundungsrate von vier Agenten, während sie ein Gebiet erkunden. Es ist deutlich zu erkennen, dass hier die Mission-Planning-Strategie einen klaren Vorteil hat, der einzige Grund warum keine Erkundungsrate von 100% erreicht wird, ist die Paramatrisierung der Strategie und die Tatsache, dass die Agenten nicht exakt gerade über alle Zellen fahren. Die Cooperative-Area-Exploration-Strategie und SUN Search trennen in diesem Fall etwa 10% Erkundungsrate. Im Durchschnitt ist der Unterschied aber nicht mehr so groß, wie Abbildung 8.14 zeigt. Im Mittel über alle 200 Durchgänge ist die SUN Search-Strategie

nur um etwa 7% schlechter als Cooperative-Area-Exploration. Die sprunghaften Änderungen gegen Ende sind eine Folge der Tatsache, dass nicht immer alle Agenten exakt zehn Minuten protokollieren, sondern eventuell einige Sekunden früher beendet werden. Weiterhin ist an dieser Abbildung deutlich zu sehen, dass SUN Search Version 2 nur unwesentlich besser ist als SUN Search Version 1, die Änderung der gewählten Norm für die Auswahl des Treffpunktes (vgl. auch Abschnitt 8.1.2) also keinen wesentlichen Einfluss auf die Erkundungsrate hat. Daher wurde in den vorigen Abschnitten auch nur SUN Search Version 2 betrachtet. Es ist allerdings möglich, dass ausgefeiltere Metriken zur Auswahl des Treffpunkts bessere Ergebnisse liefern.

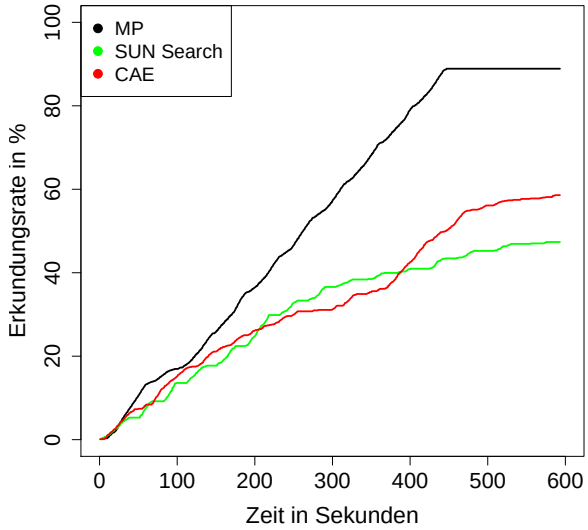


Abbildung 8.13: Vergleich der durchschnittlichen Erkundungsrate der Agenten für einen Durchgang für die jeweiligen Strategien

Bei genauerer Betrachtung der Daten stellt sich gar heraus, dass dieses Ergebnis jedoch noch verbesserungsfähig ist: Abbildung 8.13 zeigt, dass die Erkundungsrate bei SUN Search immer wieder für einige Zeit stagniert, diese Abschnitte sind teilweise sogar recht lang. Es handelt sich dabei um die Zeit, welche die Agenten zum Absprechen ihrer nächsten Ziele und des nächsten Treffpunkts benötigen. Weiterhin ist es dabei nötig, beim Data-Mining-Agent nach Empfehlungen zu fragen, was ebenfalls Zeit verbraucht. Vier Agenten benötigen für diese Absprache durchschnittlich 6 Sekunden, wobei 4 Sekunden für die Anfrage beim Data-Mining-Agent benötigt werden, und 2 Sekunden für die Absprache der UGVs untereinander. Es ist durchaus vorstellbar, dass sich der dafür benötigte Aufwand verringern lässt, wodurch ein besseres Ergebnis zu erwarten ist. Außerdem zeigen weder Cooperative-Area-Exploration noch SUN Search

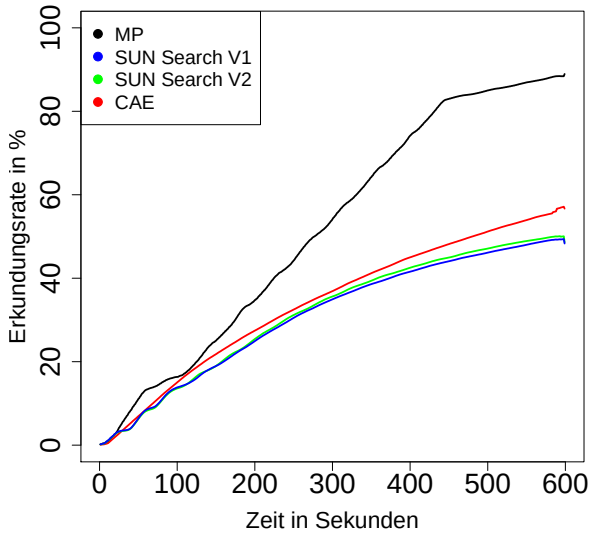
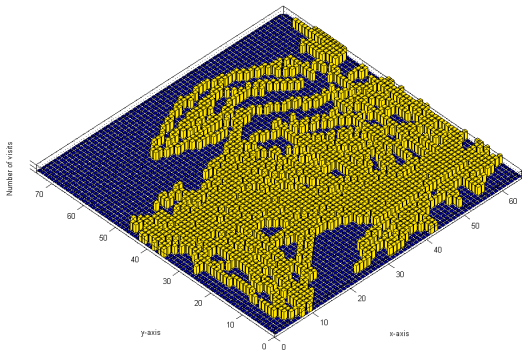


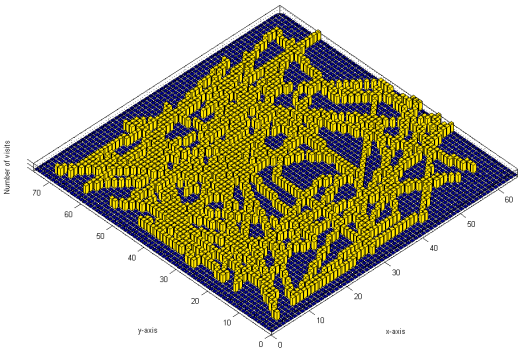
Abbildung 8.14: Vergleich der durchschnittlichen Erkundungsrate der Agenten für alle Durchgänge für die jeweiligen Strategien

einen linearen Verlauf der Erkundungsrate. Stattdessen nimmt die Steigung der Erkundungsrate ab und die Kurve wird deutlich flacher, vor allem bei CAE.

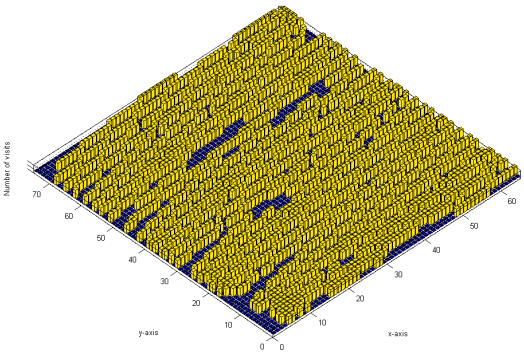
Zusätzlich zur absoluten Erkundungsrate ist auch von Interesse, wie effektiv die Fläche erkundet wurde. Eine gute Strategie sollte dabei möglichst wenig Zellen mehrfach besuchen, zumindest nicht in einem statischen System, bei dem sich die Messwerte nicht verändern, wie in Abschnitt 6.3.1 näher erläutert wird. ein Problem bei dieser Auswertung ist, dass der Data-Mining-Agent nur solche Zellen protokolliert hat, an denen auch Daten erfasst wurden. Dies hat zur Folge, dass die Aufzeichnung der Daten oftmals kleinere Lücken aufweist, insbesondere wenn das Fahrzeuge sich längere Strecken geradeaus bewegt. Aus diesem Grund wäre eine Auswertung der absoluten Häufigkeit, wie oft die Agenten einzelne Zellen besucht haben, weder repräsentativ und insbesondere nicht vergleichbar mit den vorherigen Ergebnissen aus der Simulationsumgebung, die dieses Problem nicht aufweist. Aus diesem Grund wurde eine ähnliche Betrachtungsweise herangezogen, die nur berücksichtigt, ob während der Erkundung für eine Zelle Daten vorliegen oder nicht, ohne die Häufigkeit zu berücksichtigen. Wie in Abbildung 8.15 zu sehen ist, erkundet die Cooperative-Area-Exploration-Strategie die Fläche sehr kompakt, jedoch nicht vollständig, sondern (vgl. Abb. 8.14) etwa 50%. Die SUN Search-Strategie erkundet die Fläche weniger dicht, erreicht dafür mit einzelnen Wegen aber auch Randbereiche. Dies ist der Zustand nach 10 Minuten Erkundung, zu früheren Zeitpunkten wäre der Unterschied wahrscheinlich noch deutlicher.



(a) Erkundete Zellen CAE



(b) Erkundete Zellen SUN Search



(c) Erkundete Zellen MP

Abbildung 8.15: Erkundete Gebiete der 4 Agenten für einen Durchgang der jeweiligen Strategie

8.2.4 Kommunikationsaufwand

Folgender Abschnitt befasst sich mit der Frage, ob die Verwendung von Data-Mining beim Kommunikationsaufwand, also der benötigten Bandbreite und Größe der übertragenen Daten, einen positiven Effekt bietet. Intuitiv ist von weniger Aufwand auszugehen, da nicht alle gefundenen Energiewerte einzeln kommuniziert, sondern Modelle verschickt werden. Zuerst soll auf theoretischer Basis untersucht werden, welche Ersparnisse bei der Übertragung möglich sind, wobei von der vollen Erkundung der Karte ausgegangen wird. Da beide Data-Mining-Algorithmen leicht unterschiedliche Ziele verfolgen, werden diese einzeln betrachtet. Anschließend wird das theoretische Ergebnis der Messung eines einzelnen Laufs gegenübergestellt.

Referenzüberlegung Als Referenz wird von den Messwerten der Sensoren eines jeden Fahrzeugs ausgegangen, welche über einen Broadcast-Mechanismus an alle anderen Fahrzeuge übertragen werden. D.h. bei einer Karte mit $k = i * j$ Karteneinheiten sind k Übertragungen mit jeweils der Größe eines Messwertes notwendig.

Theoretische Ersparnisse durch DBSCAN Beim DBSCAN werden die Daten komprimiert, indem eine gewisse Anzahl von relevanten Punkten in einem Micro-Cluster zusammengefasst werden. Wie in Abschnitt 8.1.2 beschrieben, werden sechs Punkte in einem Radius von 2,45 Karteneinheiten als Micro-Cluster angesehen. Da nur mittelhelle und helle Punkte beachtet werden, führt das zu einer Reduktion von bis zu 44% auf einer Cluster-Karte bzw. bis zu 55% auf einer Parkplatzkarte.

Zusätzlich führen nicht alle gefundenen Punkte zu einem neuen Micro-Cluster. Einige Punkte werden als Rauschen gewertet (vgl. Abschnitt 5.3.3), andere werden in einen bestehenden Micro-Cluster aufgenommen. Die Komprimierung hat im schlechtesten Fall den Faktor 6 aufgrund der Zuordnung zu den Micro-Clustern, was die relevanten Daten nochmal um ca. 80% reduziert.

Werden drei Micro-Cluster gefunden, wird diese Änderung allen verfügbaren Nachbarn mitgeteilt. Dadurch wird im Vergleich zum direkten Versenden eines einzelnen neuen Micro-Cluster das Verhältnis von Nutzdaten zum Overhead verringert. Außerdem ist es nicht nötig, dass Modell als Ganzes zu verschicken, sondern nur die Änderungen zu verbreiten.

Insgesamt bietet DBSCAN zwei wesentliche Möglichkeiten, den Kommunikationsaufwand zu reduzieren: Durch die Kompression der Daten in Micro-Cluster und das konzentrierte Verschicken von Änderungen.

Theoretische Ersparnisse durch den *HHH*-Algorithmus Der *HHH*-Algorithmus verwertet alle gesehenen Daten, um einen Gesamtüberblick über bereits besuchte Gebiete zu behalten. Eine Kompression des eigentlichen Modells ist nicht vorgesehen. Bei einem Update, dass nach jeweils 150 Messwerten verteilt, werden sämtliche *HHH* gesendet. Durch viele 1×1 und 2×2 Hierarchiefelder kann das Datenaufkommen allerdings sehr groß werden. Hierbei ist die Wahl des Schwellwertes ein entscheidender Faktor,

der sich auf die Anzahl der *HHH* auswirkt. Ein höherer Schwellwert führt zu einer gewissen Ungenauigkeit, deren Relevanz in weiterführenden Arbeiten geprüft werden muss. Genaue Abschätzungen sind an dieser Stelle nicht vorgesehen. Des Weiteren ist davon auszugehen, dass sich auch überlappende Hierarchieebenen negativ auswirken. Im Regelfall sollten jedoch zwei Elemente niedrigerer Ebene nötig sein, damit ein Element der nächst höheren Ebene als *HHH* gewertet wird. So kann es zwar eine Überlappung geben, die maximale Anzahl potentieller *HHH* wäre allerdings durch die Anzahl möglicher Elemente auf der untersten Ebene beschränkt.

Aber man betrachte folgendes Szenario: Zu einem Zeitpunkt t_1, sei ein Element x_1 der Hierarchieebene $l+1$ kein *HHH* und o.B.d.A. alle Kinderelemente von x seien *HHH* im Modell des Agenten $Agent_i$. Zu einem Zeitpunkt t_2 verteilt $Agent_j$ seine Updates, darunter auch ein Element x_2, welches identisch zu x_1 auf $Agent_i$ ist. Als Folge wird Element x_1 um den Zählerstand von x_2 erhöht. Unter der Bedingung, dass der Zählerstand von x_2 zum Zeitpunkt t_2 hoch genug ist, um auf $Agent_i$ noch als *HHH* zu gelten, wären sowohl alle Kind Elemente von x_1 als auch x_1 selbst *HHH*.

Auswertung eines Laufes Ein gesonderter Lauf mit vier Agenten und einer Laufzeit von 10 Minuten (600 Sekunden) wird durchgeführt, um den Kommunikationsaufwand in der Praxis zu bewerten. Auf der Simulationsmaschine eines Agenten läuft das Programm *Wireshark*[1] im Aufzeichnungsmodus mit. Das Programm zeichnet in der gewählten Einstellung den Netzwerkverkehr der lokalen Maschine und den an der Netzwerkkarte ein- und ausgehenden Verkehr auf.

Die Kommunikation zwischen den Mobilitäts- und Kommunikationsagenten untereinander und der Bodenstation wird nicht berücksichtigt und ist hier nicht weiter relevant. Durch die gewählte Art der Interprozesskommunikation (IPC) können die vom Mobilitäts- und Kommunikationsagent zum Data-Mining-Agent ausgetauschten Sensordatenpakete s eines Agenten mitgeschnitten werden. Da von einer im Schnitt gleichmäßigen Erkundung jedes Agenten ausgegangen werden kann, gilt bei n Agenten:

$$\text{Alle Sensordatenpakete} = s * n$$

Abbildung 8.16 visualisiert den Datenaustausch über den 10-minütigen Lauf, mit den Sensordaten als Referenzwert, den beiden Data-Mining-Algorithmen und dem sonstigen Datenverkehr zwischen den Data-Mining-Agenten.

Durch die Verwendung des XML Formats für die IPC fällt mehr Kommunikationsaufwand für die Sensordaten an. Die Verwendung von Hessian für die Data-Mining-Agenten-Kommunikation erzeugt ebenfalls einen erheblichen Zusatzaufwand. Abbildung 8.17 zeigt daher noch einmal den Verlauf über 10 Minuten, wobei diesmal weitestgehend nur die Nutzdaten berücksichtigt werden.

Bei einem Vergleich der Abbildungen 8.16 und 8.17 fällt auf, dass die gewählten Protokolle selbst bei niedriger Nutzlast hohe Kosten verursachen. Hier besteht Optimierungspotential. Auch wenn nur die Nutzlast berücksichtigt wird, geht die eingangs getroffene Überlegung jedoch nicht auf. Das Data-Mining, insbesondere der *HHH*-Algorithmus,

[1] `http://www.wireshark.org/`

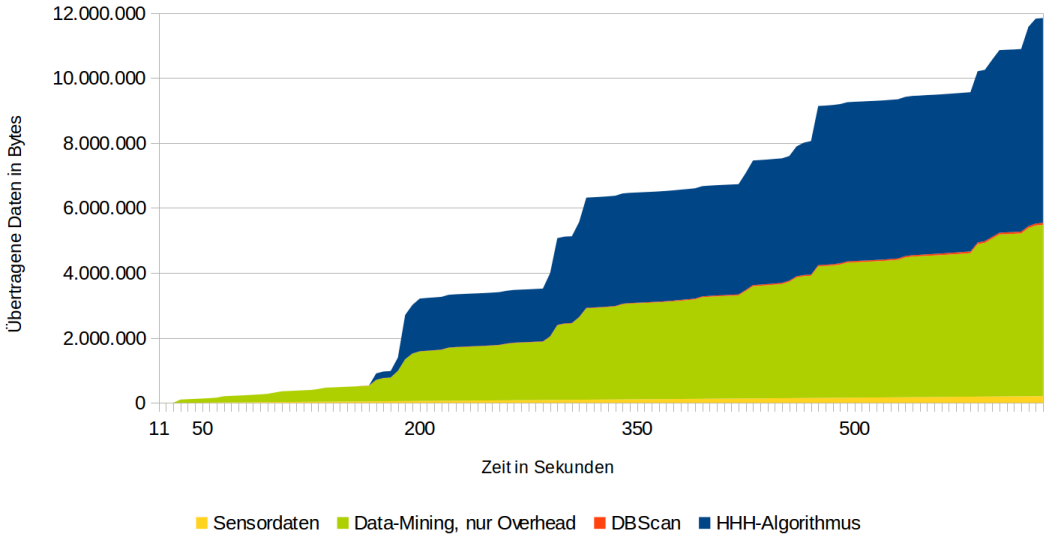


Abbildung 8.16: Über 10 Minuten Laufzeit aufsummierte Datenbytes. Die Sensordaten (ca. 208 KByte) sind exemplarisch nur von einem Agenten. Das Data-Mining produziert auffällig viel Datenverkehr (ca. 11,6 MByte.) und wird vom *HHH*-Algorithmus mit ca. 6,3 MByte dominiert.

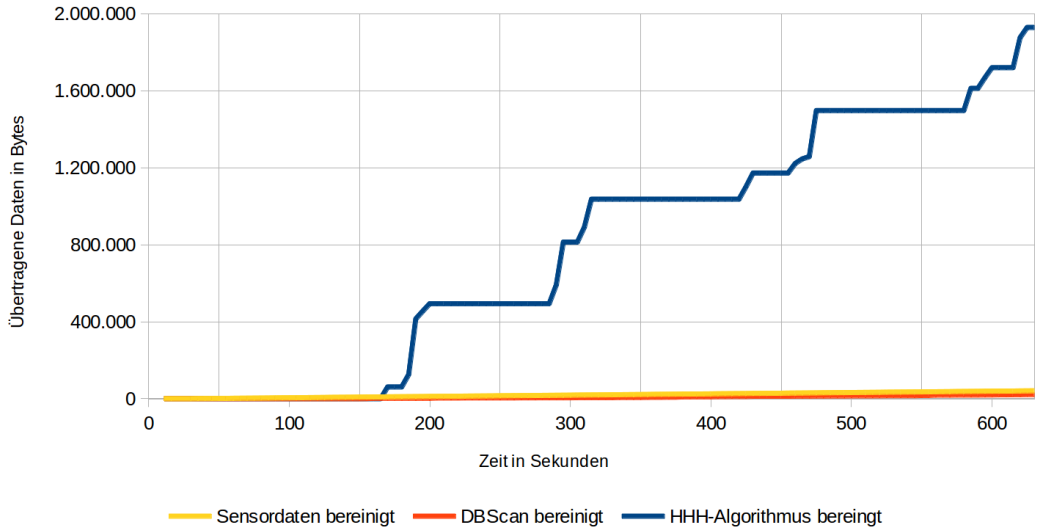


Abbildung 8.17: Über 10 Minuten Laufzeit aufsummierte Nutzdaten in Bytes. Die Sensordaten (ca. 40 KByte) sind exemplarisch nur von einem Agenten. Der *HHH*-Algorithmus produziert weiterhin den größten Datenverkehr von ca. 1,9 MByte, der DBSCAN überträgt ca. 20 KByte.

verursacht einen hohen Kommunikationsaufwand von fast 2 MB. Die unverarbeiteten Referenzdaten der Sensoren kommen zusammen auf geringe 41 KByte $*4 = 164$KByte.

Es fällt der stufenartige Anstieg des Datenaufkommens der *HHH* auf. Dieser erklärt sich mit einem Blick auf Abbildung 8.18. In dieser sind die versendeten Pakete logarithmisch über die Laufzeit aufgetragen. Der *HHH*-Algorithmus versendet demnach in kurzer Zeit viele Pakete und hält sonst Kommunikationsstille. Dieses Verhalten ist auch beim DBSCAN feststellbar, jedoch nicht so stark ausgeprägt. Anders ist dies bei den Sensordatenpaketen, welche kontinuierlich kommuniziert werden und daher nur der Mittelwert im Diagramm aufgetragen ist.

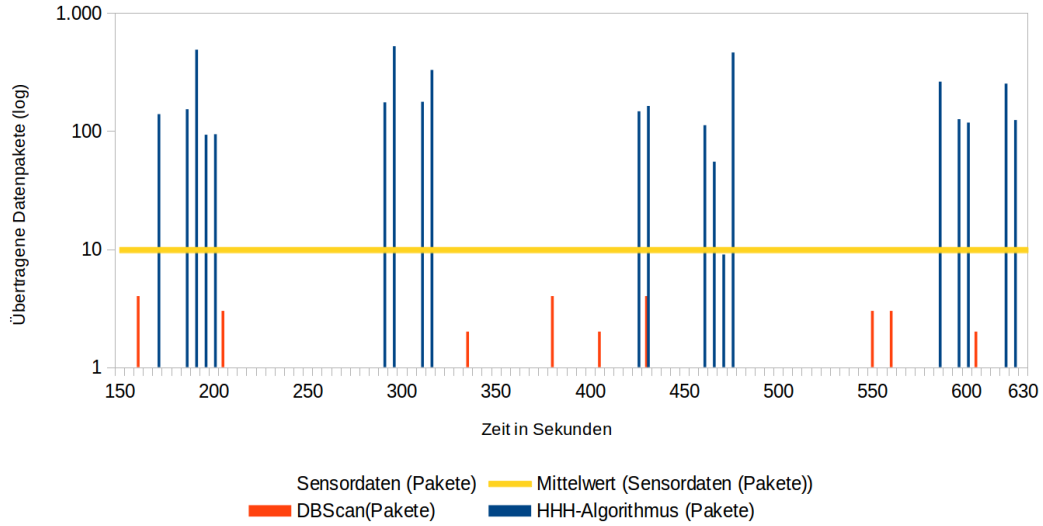


Abbildung 8.18: Über 10 Minuten Laufzeit ausgetauschte Pakete. Die Kommunikationspeeks des *HHH*-Algorithmus sind deutlich zu erkennen.

Bandbreite Mit zunehmendem Abstand der Fahrzeuge verringert sich die zur Verfügung stehende Bandbreite der Verbindung. Über die erhobenen Messwerte zeigt Abbildung 8.19 logarithmisch aufgetragen die notwendige Bandbreite in der Praxis. Demnach reicht eine Übertragungsrate von 10 KByte/s nur für den DBSCAN und auch die unverarbeiteten Sensordaten aus. Mit den *HHH* wird eine Bandbreite von 1 MByte und mehr notwendig.

Energieaspekte Im Bezug auf die Messungen des Energieverbrauchs der Kommunikation aus Kapitel 4.7.3 bringt das Data-Mining dann einen Vorteil, wenn Daten über einen längeren Zeitraum gesammelt werden können und damit keine persistente Verbindung notwendig ist. Somit kann die Kommunikationseinheit über entsprechende Zeiträume komplett abgeschaltet werden. Die benötigte Energie zum Transferieren der

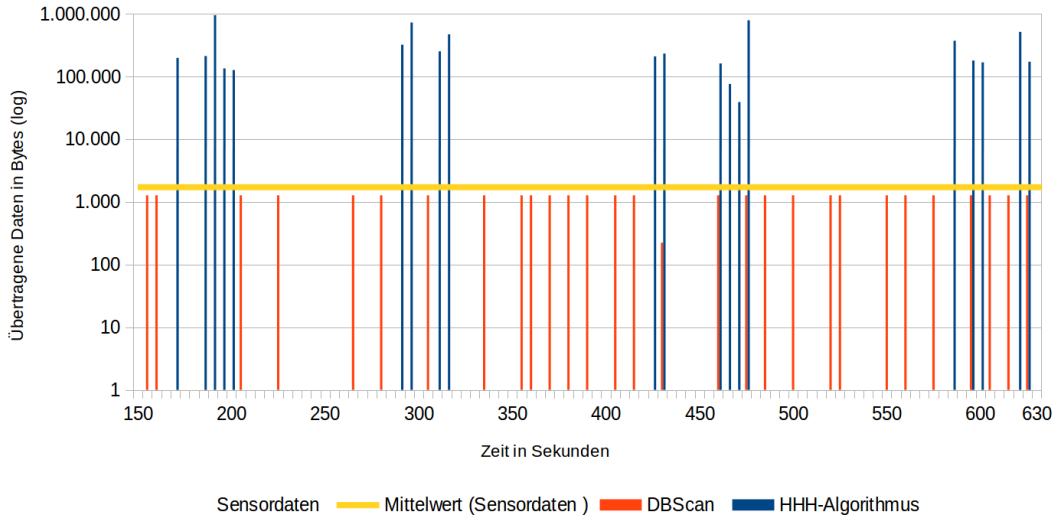


Abbildung 8.19: Über 10 Minuten Laufzeit ausgetauschte Bytes. Der *HHH*-Algorithmus benötigt die höchste Bandbreite (1 MByte) aller untersuchten Messgrößen.

eigentlichen Nutzdaten ist im Verhältnis zum Energiebedarf für die generelle Funkbereitschaft gering und die Transferraten für die anfallenden Daten auch auf längere Distanz mehr als ausreichend. Dadurch ergibt sich eine erhebliche Relevanz der eigentlichen Sendezeit im Verhältnis zu der zu sendenden Datenmenge.

Bewertung und Verbesserungen Zusammenfassend lässt sich feststellen, dass die durch das Data-Mining verursachten Nutzdaten deutlich höher sind als die Sensormesswerte. Betrachtet man allerdings die beiden Data-Mining-Algorithmen jeweils für sich, lassen sich durchaus positive Auswirkungen und neue Ansätze zur Verbesserung finden. Ein allgemeines Problem der Data-Mining-Kommunikation ist die Verwendung von TCP-Verbindungen, die beim Broadcast der Modelle zu einem n-fachen Datenaufwand führt. Zugunsten der Skalierbarkeit müsste hier die Kommunikation grundlegend überdacht werden.

Der DBSCAN verursacht wie erwartet insgesamt weniger Datenaufkommen als die Sensormesswerte, allerdings verringert die Nutzung des Hessian-Protokolls den Faktor signifikant. Durch die vorhandene Parametrisierung, dass nach drei gefundenen Micro-Clustern ein Update versendet wird, ist je nach Energiekarte nicht viel Zeit zwischen den einzelnen Sende- und Empfangsphasen (vgl. Abb. 8.19), sodass eine komplette Abschaltung der Kommunikationseinheit hier nicht sinnvoll erscheint.

Die *HHH* in ihrer jetzigen Form können als erster Ansatz verstanden werden, um ein lauffähiges Gesamtsystem bereitstellen zu können. Das verhältnismäßig große Datenaufkommen hat verschiedene Ursachen. Die Nutzung von Hessian erzeugt aufgrund

der darunterliegenden Verwendung von HTTP und TCP viel Overhead. Dadurch kann zwar bequem auf Objekten gearbeitet werden, aber das Einsparpotenzial ist zu groß, als es ungenutzt zu lassen. Benutzerdefinierte Serialisierer für Hessian sind ein Ansatz, um das Datenaufkommen zu verringern. Jedoch gibt es, wie aus der praktischen Auswertung zu erkennen, auch dringenden Bedarf das Modell oder mindestens die verteilten Daten der *HHH* zu optimieren. Hier sollte als erstes versucht werden, durch Parameteroptimierung, insbesondere beim Schwellwert, bessere Ergebnisse zu erzielen. Ein Schwellwert von 1,5% der eigenständig gefundenen Daten ist hier zu gering.

Ein weiterer Ansatz nach dem in Abschnitt 5.1.4 vorgestellten Verfahren ist denkbar, in dem eine effiziente Struktur des Modells genutzt wird. Auch wenn exakt dieses Verfahren hier nicht anwendbar ist, so ist das Anpassen an die *HHH*-Struktur sicher ein interessanter Ansatzpunkt. Bezüglich der Häufigkeit der gesendeten Daten gilt allerdings, dass hier die Sendeperioden in deutlicheren Abständen stattfinden (vgl. Abb. 8.18) und so eine temporäre Sendepause möglich ist.

9 Diskussion und Ausblick

Der vorliegender Bericht beschrieb, wie die in Abschnitt 1.1.2 beschriebene Aufgabe der Projektgruppe erreicht wurde. Es wurde erläutert, wie die einzelnen Subkomponenten (Data-Mining, Hardware, Kommunikation und Erkundungsstrategie) für sich genommen funktionieren und, wie sie schließlich zusammen arbeiten. Das so entstandene System wurde mittels einer Software-in-the-loop Simulation bewertet. Die dazu nötige Methodik ist ebenfalls in diesem Bericht beschrieben. Bei der Analyse stellte sich dann heraus, dass das entwickelte System noch Schwächen aufweist, aber insgesamt bereits gut funktioniert. Dabei ist insbesondere das eingesetzte Clustering für die Art der Erkundung weniger gut geeignet als erwartet. Demgegenüber steht allerdings, dass das Modell bereits gut für die Erkundung eines Gebiets funktioniert. Insbesondere hat sich die Erwartung bestätigt, dass die entwickelte Erkundungsstrategie vergleichsweise schnell einen guten Überblick über die Umgebung verschafft. Abschließend wird in diesem Kapitel noch einmal das Gesamtergebnis zusammengefasst und bewertet. Daraufhin wird auf einige der offen gebliebenen Fragen näher eingegangen und es werden erste Ansätze für eine zukünftige Lösung dieser Probleme gegeben.

9.1 Diskussion der Ergebnisse

Folgender Abschnitt bewertet noch einmal abschließend die Ergebnisse des Projekts und diskutiert aufgetretene Probleme. Es werden dazu zunächst die Analyseergebnisse bezüglich der verwendeten Data-Mining-Methoden zusammengefasst und daraufhin auf die Erkundungsrate, und den Nutzen des Data-Minings für diese, eingegangen.

9.1.1 Bewertung der Data-Mining-Methoden

Wie bereits erwähnt, hat die Analyse des Systems Schwächen offenbart, welche zumindest zum Teil in den verwendeten Data-Mining-Methoden begründet liegen. Es hat sich herausgestellt, dass das verwendete Clustering nicht so gut wie erwartet mit den gegebenen Datenströmen funktioniert. Die verwendete Methode benötigt eine Mindestanzahl an Punkten innerhalb eines gewissen Radius um einen Cluster zu bilden. Diese Bedingungen werden jedoch nur vergleichsweise selten erfüllt, weshalb weniger Cluster gefunden werden als erwartet. Sowohl SUN Search als auch Cooperative-Area-Exploration sind zu einem gewissen Grad in der Lage, damit umzugehen, da beide Algorithmen häufiger Kurven fahren, wo dies nicht in diesem Maße auffällt. Diese Einschränkung ist allerdings ausgesprochen nachteilig für die Mission-Planning-Strategie, welche fast ausschließlich gerade Linien fährt.

Modellbildung Nichtsdestotrotz fällt auf, dass die Erkundung sehr gut funktioniert, wenn dieses Problem nicht auftritt. Weiterhin haben die Agenten am Ende der Erkundungsphase immer ein sehr ähnliches Modell, das Ziel durch Kommunikation verteilt ein gutes Bild der Umgebung zu bekommen wurde also ebenfalls erreicht. Mit Hilfe anderer Clustering-Algorithmen sollte es daher möglich sein, die Ergebnisse signifikant zu verbessern. Weiterhin hat sich gezeigt, dass die Varianz der gefundenen hellen Cluster mit SUN Search sehr schnell ansteigt. SUN Search erfüllt also die Aufgabe, möglichst schnell ein grobes Gesamtbild der Karte zu bilden, welches es dann zu verfeinern gilt, sehr gut.

Aufgabenteilung Die Tatsache, dass sich schnell eine grobe Übersicht eines Gebietes finden lässt, passt außerdem sehr gut in das zu Beginn in Kapitel 1 vorgestellte Szenario von autonom agierenden Robotern, welche verschiedene Aufgaben in unbekanntem oder nur teilweise bekanntem Terrain durchführen. Es ist mit den gegebenen Data-Mining-Methoden gut möglich, die Erkundung des Gebiets nur in kurzen Schüben neben der eigentlichen Aufgabe des Roboters durchzuführen. Auf diese Weise können Wartezeiten effizient genutzt werden.

9.1.2 Bewertung der Erkundungsrate

Während der Analyse hat sich gezeigt, dass SUN Search verglichen mit Cooperative-Area-Exploration im Bezug auf die Erkundungsrate gute Ergebnisse liefert. Beide Algorithmen erkunden ungefähr den gleichen Prozentsatz der Karte. Daraus lässt sich schließen, dass die von SUN Search durchgeführte Auswahl von Gebieten zur Erkundung und die darauf folgende Trennung des Schwarms eine ähnlich sinnvolle Strategie ist, wie die Erkundung des Gebiets als kohärenter Schwarm. Damit dies gut funktioniert, werden die Gebiete mit Hilfe der Ergebnisse der durchgeführten Data-Mining-Methoden ausgewählt. Wegen der guten Erkundungsrate kann hier definitiv von einem Erfolg gesprochen werden: Weitere Verfeinerungen der verwendeten Methoden werden sicherlich sogar zu einer noch besseren Erkundungsrate führen.

9.2 Bewertung des Projekts

Zur Bewertung des Projektes lässt sich zunächst feststellen, dass die in Abschnitt 1.1.2 definierten Ziele erreicht wurden. Damit ist das Projekt definitionsgemäß ein Erfolg. Darüber hinausgehend ist sind die Analyseergebnisse ein klares Signal, dass sich weitere Forschung an der Vereinigung von Data-Mining und Erkundungsstrategien lohnt. Nichtsdestotrotz gibt es immer Grund sich mit dem Ablauf und den Ergebnissen kritisch auseinanderzusetzen, was im Folgenden geschieht.

Aufgetretene Probleme Bei Betrachtung des Berichts und einem Vergleich mit der Einleitung fällt allerdings auf, dass die Ziele erreicht, viele interessante Fragen allerdings nicht betrachtet wurden. Die Gründe dafür sind vielfältig, der gewichtigste Grund

ist allerdings die mangelnde Zeit: Insbesondere bei der Integration der einzelnen Komponenten zu einem zusammenhängenden Gesamtsystem sind ausgesprochen viele Probleme aufgetreten, welche nicht immer optimal behandelt wurden. Auf diese Weise hat diese Integration wesentlich länger gedauert als geplant, was schließlich dazu führte, dass am Ende Zeit fehlte, um das System noch zu optimieren. Die vorliegende Analyse beschreibt im Wesentlichen den Stand der ersten funktionierenden Fassung des Agenten. Das dabei noch einige Probleme auftraten ist dementsprechend nicht überraschend.

Die konkreten Probleme, die während der Integration auftraten, sind lassen sich meist durch Implementierungsfehler beschreiben. So passierte es häufig, dass die Lösung eines Fehlers in einer Komponente zeigte, dass auch andere Komponenten noch Fehler hatten. Außerdem führten Änderungen an den zugrundeliegenden Frameworks oftmals zu Inkompatibilitäten, welche nur durch aufwendige Merges zu beheben waren. Schließlich gab es, insbesondere zu Beginn, größere Schwierigkeiten mit der genutzten Hardware: Der notwendige Wechsel der Architektur (vgl. Abschnitt 4.6.5) hat zu signifikantem Portierungsaufwand geführt.

Lösungsansätze Für die aufgetretenen Probleme lassen sich sicherlich eine Reihe Lösungen finden. Hier sollen nun einige Möglichkeiten genannt werden.

Projektmanagement Im Sinne von Scrum wurde innerhalb der Gruppe kein Projektleiter gewählt. Dabei wurde aber außer Acht gelassen, dass es aus gutem Grund mit dem Scrummaster eine ähnliche Rolle gibt. Diese (möglicherweise mit Anpassungen) umzusetzen hätte vermutlich bereits einige Probleme – insbesondere bei der Integration – verhindert, da diese Rolle auch ein gewisses forcieren von Kommunikation vorsieht.

Kommunikation Die einzelnen Teilgruppen haben stellenweise zu wenig miteinander kommuniziert wo Probleme auftreten. Insbesondere das zuvor angesprochene Problem der „Fehlerkaskaden“ hätte durch häufigere Kommunikation zwischen den Teilgruppen entschärft werden können.

Auflösung der Teilgruppen Eine weitere Methode zuvor genanntes Problem zu lösen wäre es gewesen die Teilgruppen weitgehend aufzulösen bzw. dafür zu sorgen, dass immer, wenn an der Integration gearbeitet wird, ein Teilnehmer jeder Teilgruppe anwesend ist. Auf diese Weise hätten Probleme, die sich durch die Behebung eines Fehlers in einem anderen Teil des Systems ergeben, sofort bearbeitet werden können und nicht erst einige Zeit später.

Frameworks Bezüglich der verwendeten Frameworks wäre es sinnvoll gewesen sich frühzeitig auf eine Version festzulegen und diese nicht mehr zu ändern. Da dies nicht immer möglich war (es kam vor, dass auch für die Projektgruppe spezielle Erweiterungen nötig waren) wäre es zumindest ratsam gewesen für die Projektgruppe einen eigenen Entwicklungszweig zu pflegen um dafür zu Sorgen, dass größere API-Änderungen das laufende Projekt nicht beeinflussen.

Hardware Die Probleme mit der Hardware wurden generell bereits gut gelöst. Statt darauf zu warten, dass diese funktioniert wurde auf simulierten Testsystemen entwickelt und darauf geachtet, dass die Kompatibilität zur Zielhardware bestehen bleibt. Dieses Vorgehen hat sich als angemessen herausgestellt.

9.3 Offene Fragen und Lösungsansätze

Wie bereits erwähnt, konnten nicht alle Aspekte des behandelten Problems in der zur Verfügung stehenden Zeit beleuchtet werden. Dieser Abschnitt befasst sich daher schließlich mit einigen offenen Fragen und nennt erste Ansätze zur Lösung. Dabei wird zunächst auf mögliche Methoden eingegangen, das Clustering der Daten zu verbessern. Darauf hin wird die weiter führende Frage gestellt, wie mit veränderlichen Daten umgegangen werden könnte. Abschließend werden Vorschläge genannt, welche die Kommunikation zwischen den UGVs reduzieren könnten und die Probleme mit dem Autopiloten kurz thematisiert.

9.3.1 Verbesserung des Clusterings

Es stellte sich heraus, dass eines der größten Probleme des Systems darin liegt, zu viele helle Punkte als Rauschen zu identifizieren, statt Cluster an diesen Stellen zu bilden. Die verwendete Clustering-Methode funktioniert also in der vorgestellten Form noch nicht optimal. Es ist allerdings zu erwarten, dass dieses Ergebnis verbessert werden kann. Dazu gibt es im Prinzip zwei Möglichkeiten: Zum einen kann versucht werden, die Parameter des bestehenden Verfahrens so anzupassen, dass sich ein besseres Ergebnis ergibt. Dies ist allerdings nur begrenzt erfolgversprechend, da die Parameter dazu so gewählt werden müssten, dass sich beinahe überall ein Cluster ergibt. Das so entstehende Modell wäre zwar vermutlich sehr akkurat, allerdings kann von Datenreduktion kaum noch die Rede sein. Zum anderen ist es natürlich möglich, andere Verfahren zu verwenden. Eine erste Idee wäre ein entropiebasiertes Clustering: Dabei werden gefundene Punkte sehr schnell zu Clustern zusammengefasst, die zunächst aber eine niedrige Konfidenz haben. Im Folgenden wird dann die Entropie von hellen Punkten zu dunklen Punkten innerhalb des Clusters betrachtet und wenn sich zeigt, dass deutlich mehr dunkle Punkte im Cluster sind als helle, so wird der Cluster wieder aus dem Modell entfernt. Eine mögliche Formel für diese Berechnung der Entropie wäre

$$\sum_{i=1}^{K} \frac{k_i}{n}(-\log k_i - \log n),$$

wobei K die Anzahl verschiedener Energiewerte ist, k_i die Anzahl innerhalb des Clusters gefundener Punkte mit Energiewert i und n die Gesamtzahl der gesehenen Punkte im Cluster. Auf diese Weise würden bereits wenige Daten zur Erzeugung eines Clusters ausreichen und es könnten für die improve-Anfragen gezielt Cluster mit geringer Konfidenz ausgewählt werden, um diese zu verbessern. Dazu ist es natürlich ebenfalls not-

wendig, ein vernünftiges Modell für die Konfidenz in einem Cluster zu entwickeln. Neben dazu offensichtlich geeigneten Wahrscheinlichkeitstheoretischen Ansätzen könnte hier auch über den Einsatz einer nichtmonotonen Logik diskutiert werden. Diese sind häufig intuitiv verständlicher als Wahrscheinlichkeiten.

9.3.2 Dynamische Daten

Eine weitere offene Frage ist das Verhalten bei sich verändernden Daten. Das angedachte Szenario ist bereits so ein Fall: Aufgrund des über den Tag wechselnden Sonnenstands ändern sich auch die Schattenwürfe mit der Zeit und eine Position, die vor einer Stunde noch gut zum Laden des Akkus geeignet war, mag nun nicht mehr sinnvoll sein. Derzeit ignoriert das System dieses Problem vollständig. Mögliche Lösungsansätze sind alte Daten mit neuen einfach zu überschreiben, wenn sie sich ändern, Daten automatisch mit der Zeit als obsolet zu erklären und/oder über mehrere Tage hinweg ein gewissermaßen vorausschauendes Modell zu lernen. Messfehler werden bei diesem Problem ebenfalls wesentlich interessanter: Was macht man, wenn ein Agent überzeugt ist, dass eine bestimmte Stelle dunkel ist, die anderen sie jedoch als hell deklarieren? Es ist vermutlich nicht immer sinnvoll, einfach den neuesten Wert als gültig zu erklären. Stattdessen muss in diesem Szenario ganz besonders sichergestellt werden, dass ein gewisses Vertrauen in die Messwerte vorliegt. Das in Abschnitt 5.1.2 vorgestellte Verfahren zur Outlier Detection könnte in diesem Zusammenhang durchaus interessant sein.

Forschungsansätze zu Data-Mining auf dynamischen Daten gibt es ebenfalls bereits. Roddick und Spiliopoulou haben dazu in [**?**] einige Einführende Artikel kurz vorgestellt. Für den hier vorgestellten Zweck insbesondere interessant wäre die Nutzung eines Spatio-Temporalen Modells, welches sowohl Zeit als auch Raum modelliert. Die Forschung in diesem Bereich ist allerdings noch Jung, derzeit ist daher schwer abzusehen, welche Einflüsse ein solches Modell auf das bestehende System hätte. Es ist aber davon auszugehen, dass die Modellkomplexität dadurch ansteigt, da weitere Dimensionen betrachtet werden müssen. Wenn die Modelle außerdem weiterhin zwischen den Agenten ausgetauscht werden sollen, wird damit auch der Kommunikationsaufwand weiter ansteigen, es wird also besonders wichtig, hier weitere Kompressionsverfahren zu verwenden.

9.3.3 Verbesserung der Erkundungsrate

Es wurde angesprochen, dass die durch SUN Search erreichte Erkundungsrate bereits kompetitiv ist. Über alle für die Analyse durchgeführten Läufe gemittelt ist sie nur wenig schlechter als Cooperative-Area-Exploration. Es wurde weiterhin festgestellt, dass diese an zwei Stellen womöglich noch verbessert werden kann: Die Absprachephasen dauern noch sehr lange und die Wahl der Treffpunkte ist suboptimal.

Absprachephase Ein gewichtiger Faktor in der Absprachephase ist die Anfrage nach dem aggregierten Modell des Data-Minings. Es dauert etwa 4 Sekunden auf der ver-

gleichsweise kleinen Umgebung die beiden Modelle zu aggregieren, in größeren Gebieten wird sich diese Zeit noch erhöhen, von der Anzahl der UGVs ist sie allerdings unabhängig. Als Hauptproblem lässt sich hier die Tatsache identifizieren, dass die Aggregation bei jeder Anfrage von neuem durchgeführt wird. Es ist sicherlich möglich hier ein dynamisches Modell zu verwenden, also das aggregierte Modell persistent zu halten und nur lokale Änderungen durchzuführen, wenn sich Änderungen an den lokalen Modellen der Data-Mining-Algorithmen ergeben.

Ein zweiter Punkt ist die Absprache der UGVs untereinander. Diese macht im Analyseszenario den kleineren Anteil an der Wartezeit aus, allerdings besteht Grund zu der Annahme, dass diese Zeit mit steigender Anzahl beteiligter UGVs steigt, da diese Absprache sequentiell durchgeführt wird. Die Größe der Umgebung hat an dieser Stelle allerdings keinen Einfluss auf die Dauer der Absprache. Zu überlegen wäre also wie diese Absprache parallelisiert werden kann. Eine erste Idee dazu wäre, dass alle UGVs parallel ihr Zielgebiet wählen und nur bei Konflikten weitere Absprachen tätigen. Dies müsste allerdings voraussichtlich weiter verfeinert werden: Da die lokalen Modelle der Data-Mining-Agenten sich stark ähneln, sind auch die vorgeschlagenen Gebiete häufig gleich. Insbesondere am Anfang der Erkundung, wenn alle Agenten nah beieinander stehen, wird es dann häufig zu Konflikten kommen. Diese Konflikte könnte man beispielsweise lösen indem das UGV mit der niedrigsten Schwarm-ID das gewählte Zielgebiet bekommt, während die anderen nun, mit dem Wissen welche Gebiete bereits belegt sind, die neuen Gebiet wählen können. Dennoch wird dieser Ansatz zu mehr Überschneidungen bei der Erkundung von Gebieten führen, es wäre also zu analysieren, wie sehr sich die Zeitersparnis hier lohnt.

Treffpunktwahl Die Wahl des Treffpunktes nach der Erkundungsphase kann ebenfalls noch verbessert werden. Dieser wurde zunächst mit der L_1-Norm über dem Vektor der einzelnen Distanzen gewählt, jedoch stellte sich heraus, dass dies bei schlechter Verteilung der UGVs keine gute Idee ist (sind 3 UGVs in einer Ecke der Umgebung und das vierte in der gegenüberliegenden, so wird ein Treffpunkt in der nähe der Dreiergruppe gewählt, statt auf halbem Weg zwischen den beiden). Jedoch hat sich bei der Analyse gezeigt, dass die Umstellung auf die L_∞-Norm keine wesentliche Verbesserung gebracht hat. Entweder ist der Problemfall in Version 1 nicht signifikant oft aufgetreten, oder auch Version 2 wählt den Treffpunkt noch nicht immer sinnvoll. Letzteres kann insbesondere dann passieren, wenn die UGVs entscheiden sich sternförmig auszubreiten und den Treffpunkt in etwa in die Mitte des Sterns setzen. In diesem Fall fahren sie die gleichen Wege wieder in die Mitte zurück. Es ist also sicherlich von Vorteil hier noch nach besseren Metriken zu suchen, die möglicherweise auch die Wege oder andere Zusatzinformationen berücksichtigen. Weiterhin könnten beispielsweise Höhenunterschiede zwischen den Fahrzeugen beachtet werden, da diese sich sowohl auf die Geschwindigkeit, als auch den Energieverbrauch auswirken können.

9.3.4 Kommunikationseffizienz

Von großem Interesse ist die Frage, inwieweit die Kommunikation zwischen den UGVs noch eingeschränkt werden kann. Das Cluster-Modell komprimiert die Daten bereits sehr gut, aber das verwendete Heavy-Hitter-Modell muss noch weiter betrachtet werden, denn derzeit wird das gute Ergebnis bei der Erkundungsrate mit dem Verschicken großer Datenmengen erkauft. Der erste Schritt, dies zu verbessern, läge darin, die im Hierarchical-Heavy-Hitters-Algorithmus beschriebene Kompressionsphase des Algorithmus auch umzusetzen (vgl. Abschnitt 5.1.1). Dies wird in der derzeit verwendeten Implementierung nicht getan. Eine weitere Möglichkeit ergibt sich durch die Verwendung von Kompressionsverfahren, wie beispielsweise dem in Abschnitt 5.1.4 vorgestellten. Dieses ist zwar zur Komprimierung von Entscheidungsbäumen gedacht, aber das *HHH*-Modell hat ebenfalls eine baumartige Struktur, also ist es sicherlich möglich das Verfahren in ähnlicher Weise auch auf dieses anzuwenden. Als weiteren Ansatzpunkt kann auch das Netzwerkprotokoll um ein Kompressionsverfahren erweitert werden.

9.3.5 Hardware

Resourcen Für die derzeit im Einsatz befindliche Software ist die Prozessorleistung der verwendeten Rechenkomponente ausreichend dimensioniert. Die bereits angesprochenen Änderungen und Erweiterungen bzw. die dafür notwendigen Algorithmen können aber schon zusätzliche Prozessorlast verursachen, welche von der aktuellen Plattform nicht mehr bedient werden kann. Auch der Arbeitsspeicher wird im Moment nicht annähernd komplett gefüllt, bei einem größeren zu erkundenden Gebiet und damit wachsenden Modellen für das Data-Mining kann sich das allerdings ebenfalls schnell ändern. Hier könnte also eine Komprimierung der Daten notwendig werden, welche den Prozessor zusätzlich belastet. Denkbar ist natürlich eine ganz andere, performantere Rechenkomponente, welche dann aber eventuell auch einen höheren Energieverbrauch aufweist.

Energieaspekte In Kapitel 4.7.3 wurde experimentell eine Laufzeit von gut sieben Stunden ermittelt, wenn nur die Rechenkomponente inklusive Kommunikation genutzt wird. Vor allem die Kommunikation über WLAN verbraucht viel Energie, weshalb hier andere Funkstandards wie beispielsweise ZigBee eventuell besser geeignet sind. Wieviel Strom die Motoren bei unterschiedlichen Geschwindigkeiten benötigen wurde genauso wie das Verhältnis vom Gewicht des Rovers zum Energiebedarf noch gar nicht betrachtet. Desweiteren konnte dieses Projekt zwar aufzeigen, dass in einem Schwarm organisierte Fahrzeuge mittels Data-Mining zuverlässig geeignete Orte zum Aufladen der Akkus zu finden, aber der eigentliche Ladevorgang wurde nicht thematisiert. Da das aktuell genutzte Fahrzeug einen Akku mit recht hoher Kapazität besitzt und nur verhältnismäßig wenig Platz für große Solar-Panels bietet, könnte die Ladezeit ein entscheidendes Problem werden. Denkbar wäre hier ein separates Ladefahrzeug,

das auf Kosten von Agilität mit größeren Akkus und Solar-Panels bestückt wird und eine Art Tankstelle für die anderen Fahrzeuge bildet.

Autopilot Aufgrund von Problemen mit der Autopilotsoft- und hardware fährt das UGV noch nicht selbstständig. Für eine endgültige Bewertung ist es notwendig, die Ergebnisse aus der Simulation in realen Testläufen zu verifizieren. Die hier vorgestellten Simulationsergebnisse können also derzeit lediglich einen grundlegenden Hinweis darauf geben, ob die Verbindung von Data-Mining und klassischen Erkundungsstrategien wirklich sinnvoll ist. Außerdem wurden bis jetzt weder temporäre noch dauerhafte Hindernisse thematisiert. Auch in dieser Richtung könnten beispielsweise zukünftige Projektgruppen weitere Arbeit leisten.

Kommunikation Bis jetzt wurde die Kommunikation zwischen den Agenten unter optimalen Bedingungen getestet. In der Simulation wurde anhand einer fiktiven Verbindungsqualität in Abhängigkeit zur Entfernung der jeweiligen Stationen zueinander entschieden, ob eine Kommunikation stattfindet. In der Realität können aber durch Interferenzen und Hindernisse Störungen auftreten, die noch nicht betrachtet wurden. Es ist gar möglich das über längere Zeit, etwa durch eine Störung des Frequenzbands, auch auf kurzen Distanzen gar keine Kommunikation möglich ist. Hierfür müssen entsprechende Ausweichstrategien entwickelt werden, um für akkuschwache Fahrzeuge einen Notfallplan zu haben.

Stabilität Fehlfunktionen der Hardware werden aktuell noch nicht behandelt. Ein ausgefallener Sensor oder auch nur ein abgestürzter Dienst müssen im Idealfall erkannt und angemessene Fehlerbehebungsroutinen gestartet werden. Ein Fahrzeug muss dem Schwarm mitteilen können, dass es keine zuverlässigen Messwerte liefern kann oder Neustarten können, wenn das System hängen geblieben ist. Hier könnte dann auch das Thema Redundanz einzelner Komponenten zur Erhöhung der Fehlertoleranz Anwendung finden.

Literaturverzeichnis

[1] Tsl2560, tsl2561 light-to-digital converter, datasheet. http://www.adafruit.com/datasheets/TSL2561.pdf, december 2005. Dok.Nr. TAOS059D.

[2] Mcp9843/mcp98243, memory module temperature sensor /w eeprom for spd, datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/22153c.pdf, november 2009. Dok.Nr. DS22153C.

[3] Lea-6 series, u-blox 6 gps, qzss, glonass and galileo modules, product summary. http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_ProductSummary_%28GPS.G6-HW-09002%29.pdf, 2012. Dok.Nr. GPS.G6-HW-09002-E1.

[4] Mediatek-3329, 66-channel gps engine board antenna modul with mtk chipset, datasheet. http://inmotion.pt/documentation/diydrones/MediaTek_MT3329/mediatek_3329.pdf, april 2012. Rev.A03.

[5] Um10204, i^2c-bus specification and user manual, rev.4. http://www.nxp.com/documents/user_manual/UM10204.pdf, february 2012.

[6] S. Bandyopadhyay, C. Gianella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering distributed data streams in peer-to-peer environments. In *Information Sciences*, pages 1952—-1985. Department of Computer Science and Electrical Engineering, University of Maryland, 2006.

[7] D. Behnke, K. Daniel, and C. Wietfeld. Comparison of Distributed Ad-hoc Network Planning Algorithms for Autonomous Flying Robots. In *Proc. of IEEE Globecom 2011*, Houston, Texas/USA, december 2011.

[8] J. Bellingham, M. Tillerson, A. Richards, and J. P. How. Multi-task allocation and path planning for cooperating UAVs. In *Cooperative Control: Models, Applications and Algorithms at the Conference on Coordination, Control and Optimization*, pages 1–19, november 2001.

[9] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, page 51, 2006.

[10] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21:376–386, 2005.

[11] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[12] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM Conference on Data Mining*, pages 328–339, 2006.

[13] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in streaming data. *TKDD*, 1(4), 2008.

[14] K. Daniel, S. Rohde, N. Goddemeier, and C. Wietfeld. Cognitive agent mobility for aerial sensor networks. *Sensors Journal, IEEE*, 11(11):2671–2682, november 2011.

[15] K. Das, K. Bhaduri, K. Liu, and H. Kargupta. Distributed identification of top-l inner product elements and its application in a peer-to-peer network. *IEEE Transactions on Knowledge and Data Engineering*, 20:475–488, 2008.

[16] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *Internet Computing, IEEE*, 10(4):18–26, 2006.

[17] S. Datta, C. Giannella, and H. Kargupta. K-Means Clustering over a Large, Dynamic Network. In *Proceedings of 2006 SIAM Conference on Data Mining*, 2006.

[18] K. Dembowski. *Energy Harvesting für die Mikroelektronik: Energieeffiziente und-autarke Lösungen für drahtlose Sensorsysteme*. Vde Verlag GmbH, 2011.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[20] C. Giannella, K. Liu, T. Olsen, and H. Kargupta. Communication efficient construction of decision trees over heterogeneously distributed data. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 67–74. IEEE, 2004.

[21] N. Goddemeier, S. Rohde, K. Daniel, and C. Wietfeld. Communication Constrained Mobility and Topology Management for Relay Sensor Networks . In *IEEE WCNC*, Mexico, 2011.

[22] N. Goddemeier, S. Rohde, J. Pojda, and C. Wietfeld. Evaluation of Potential Fields Mobility Strategies for Aerial Network Provisioning. In *Proc. of IEEE Globecom Workshop on Wireless Networking for Unmanned Autonomous Vehicles*, Houston, TX, USA, december 2011. IEEE.

[23] D. Goldberg. Genetic algorithms and walsh functions: Parti, a gentle introduction. *Complex Systems*, 3(2):129–152, 1989.

[24] D. Gredler. Java remoting: Protocol benchmarks. `http://daniel.gredler.net/2008/01/07/java-remoting-protocol-benchmarks/`. Aufgerufen: 09.09.2012.

[25] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, march 2003.

[26] J. Hartung, B. Elpelt, and K.-H. Klösener. *Statistik: Lehr- und Handbuch der angewandten Statistik*. Oldenbourg, München [u.a.], 13 edition, 2002.

[27] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, july 2003.

[28] G.R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke. Ieee 802.11s: The wlan mesh standard, february 2010.

[29] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[30] H. Kargupta, B. Park, D. Hershberger, and E.Johnson. Collective data mining: A new perspective toward distributed data mining. *AAAI/MIT Press*, pages 133–184, 2000.

[31] H. Kargupta and B.-H. Park. A fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments, 2004.

[32] Kun Liu, Kanishka Bhaduri, Kamalika Das, Phuong Nguyen, and Hillol Kargupta. Client-side web mining for community formation in peer-to-peer environments. *SIGKDD Explor. Newsl.*, 8(2):11–20, December 2006.

[33] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[34] Sudip Misra, Subhas Chandra Misra, and Isaac Woungang, editors. *Guide to Wireless Mesh Networks (Computer Communications and Networks)*. Springer, december 2010.

[35] K. Pearson. The problem of the random walk. *Nature*, 72:294–294, may 1905.

[36] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[37] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[38] P. Urcola and L. Montano. Cooperative robot team navigation strategies based on an environment model. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4577–4583, october 2009.

[39] L. Wilkinson and M. Friendly. The history of the cluster heat map. *The American Statistician*, 63(2):179–184, 2009.

[40] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.