Masterarbeit

zur Erlangung des akademischen Grades

Diplomingenieur

Autonome Roboter in der Intralogistik: Möglichkeiten zur Optimierung der Auftragsverteilung

vorgelegt von:	Erik Himmelsbach
Studienrichtung:	Industrielogistik
Matrikelnummer:	0735296
Universitärer Betreuer:	Univ.-Prof. Dr. Peter Auer

Verfasst am Lehrstuhl für Informationstechnologie an der Montanuniversität Leoben.

1. Oktober 2014

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using ony literature cited in this volume.

Datum	Unterschrift

Danksagung

Für die Unterstützung beim Erstellen meiner Masterarbeit möchte ich mich bei folgenden Personen bedanken:

Meinem universitären Betreuer Univ.-Prof. Dr. Peter Auer danke ich dafür, dass er stets verfügbar war und mit vielen Ideen und wertvollen Anregungen den Erfolg dieses Projekts ermöglicht hat.

Ein besonderer Dank gilt auch meinem Betreuer in der Firma Knapp Systemintegration GmbH, Dipl.-Ing. Stephan Maurer, durch dessen Einsatz und Hilfsbereitschaft die Arbeit an diesem Thema überhaupt erst möglich wurde.

Ebenfalls bedanken möchte ich mich bei allen Mitarbeitern der incubed IT GmbH, die mich beinahe ein ganzes Jahr in ihr Team aufgenommen und mich während dieser Zeit immer mit gutem Rat unterstützt haben.

Schließlich möchte ich mich auch bei meinem Vater für sein genaues Vorgehen beim Korrekturlesen bedanken.

Abstract

OpenShuttle ist ein neues Produkt der KNAPP Systemintegration GmbH, welches auf einer Flotte von freifahrenden Robotern beruht, die für Transporttätigkeiten im Bereich der Intralogistik eingesetzt werden. Mithilfe von Lasernavigation können sie sich ohne zusätzliche Infrastruktur durch das Lager bewegen und bieten somit eine höchst flexible und leicht zu installierende Alternative zu herkömmlichen Transportsystemen. Ein aktuelles Projekt der Firma sieht den Einsatz von *OpenShuttle* in einem Lebensmitteldistributionszentrum vor, in dem Paletten zwischen unterschiedlichen Stationen transportiert werden müssen. Der Durchsatz des aktuellen *OpenShuttle*-Systems reicht jedoch nicht aus, um die Anforderungen zu erfüllen, die dieses Projekt an sein Transportsystem stellt.

Zielvorgabe dieser Masterarbeit war es daher, durch eine Optimierung der Auftragsverteilung den Durchsatz des Systems zu erhöhen. Dazu erwies es sich als notwendig, zunächst die bestehende Simulation um die Batterieladestände der Roboter zu erweitern und so in der Simulation ihr Ladeverhalten möglichst jenem von realen Robotern anzunähern. Zudem wurde die derzeitige Auftragsverteilung insofern erweitert, als in der neuen zweistufigen Auftragsverteilung ein Roboter zunächst nur eine Region zugewiesen erhält, und erst wenn er diese erreicht hat, den genauen Auftrag definitiv erhält.

Aufbauend auf diesen Vorarbeiten wurde die Auftragsverteilung mit dem Ziel optimiert, den Durchsatz des gesamten Transportsystems zu steigern. Bei Verteilung der Aufträge muss darauf geachtet werden, dass jeder einzelne Transportauftrag innerhalb eines vorgegebenen Zeitfensters erfolgt, und dass dabei auch der Batterieladestand der Roboter berücksichtigt wird.

Bei dieser neuen Auftragsverteilung wird die Zeit, die ein Roboter für die Durchführung eines bestimmten Auftrag benötigt, im voraus geschätzt. Diese Schätzung basiert auf Erfahrungswerten, die durch eine zeitliche Messung zuvor getätigter Auftragsdurchführungen gesammelt wurden. Durch eine Evaluierung möglicher Auftragsverteilungen wird jene Verteilung ermittelt, bei der die Anfahrtszeiten und die Wartezeiten an den Stationen auf ein Minimum reduziert werden. Ein Vergleich der herkömmlichen Auftragsverteilung mit der neuen Auftragsverteilung ergab, dass durch diese die durchschnittliche Durchführungsdauer eines Auftrags um bis zu 13,6% verringert werden kann.

Abstract

OpenShuttle is a new product by KNAPP Systemintegration GmbH, based on a fleet of autonomous robots performing intralogistic transportation tasks. Due to their laser-based navigation they do not need additional infrastructure for moving through the warehouse and are therefore a highly flexible and easy-to-install alternative to conventional transportation systems. One of the company's current projects involves the use of *OpenShuttle* at the distribution centre of a food retailer where the transport of pallets between different stations is necessary. However, the current *OpenShuttle*-system's throughput is not sufficient to meet this project's requirements.

The goal of this master thesis was to increase the throughput by optimizing the task distribution to the robots. For this purpose it was necessary to first expand the current simulation by the battery status of the robots to approximate their charging behaviour in the simulation. Moreover, the previous task distribution algorithm has been expanded into a new, two-staged distribution algorithm: first a region is assigned to a robot, and only when the robot reaches its assigned region, then it is given the exact task.

Based on this preparatory work the task assignment has been optimized to increase the throughput of the entire transportation system. This new task assignment also takes into account that every order has to be executed within a certain time frame. Furthermore, the assignment considers the current state of battery charge of the robots.

This new task assignment algorithm estimates in advance the time it takes a robot to execute a certain order. This estimation is based on empirical values which were collected by measuring the time needed to execute previous orders. The algorithm chooses the assignment for which the time of travel to the first station and the waiting time are as low as possible. A comparison of the previous and the new task assignment showed that the mean execution time of an order with the new task assignment is by 13,6% lower than with the previous task assignment.

Inhaltsverzeichnis

1 Einleitung

Diese Masterarbeit wurde im Rahmen einer Anstellung bei der KNAPP Systemintegration GmbH (KSI) verfasst, die ihren Sitz in Leoben hat. Diese unterhält eine Partnerschaft mit der Firma incubed IT GmbH, die auf dem Gebiet der Robotik und individueller Softwarelösungen tätig ist.

1.1 Motivation

Die Wahl, in dieser Masterarbeit ein Thema im Bereich der Algorithmenentwicklung für Roboter zu behandeln, wurde vor allem aufgrund der im technischen Master des Industrielogistikstudiums absolvierten Fächern getroffen. Dieses Thema kombiniert die Bereiche der Entwicklung effizienter Algorithmen, Operations Research, Software Engineering und Automatisierungstechnik - alles Gebiete, denen ich während des Studiums großes Interesse entgegen gebracht habe. Zugleich ist der Einsatz von autonomen frei fahrenden Transportrobotern ein noch junges Gebiet mit großem Potential in der Logistik, und dieses Thema bietet Gelegenheit, seine Möglichkeiten genauer zu erforschen und meine Kenntnisse um diesen Bereich zu erweitern. Durch die bereits bestehende Anstellung bei der KSI bot sich die Möglichkeit dieses Thema im Rahmen einer Masterarbeit zu behandeln, und ich möchte mich bei den Verantwortlichen der KSI für diese bedanken.

1.2 Problemstellung

Der Einsatz von mobilen Transportrobotern in der Lagerlogistik ist ein neues Konzept der KSI und wird unter dem Namen *Openshuttle* schon bald in neuen Projekten zum Einsatz kommen. Dabei sollen Transportaufträge von einer zentralen Stelle an eine Flotte von autonomen Robotern verteilt und von diesen durchgeführt werden. Derzeit erfolgt die Auftragsverteilung iterativ, ohne Berücksichtigung zukünftiger Aufträge oder der Ladestände der Roboter. Außerdem ist es nicht möglich, eine Auftragszuweisung, nachdem sie einmal vergeben wurde, zu einem späteren Zeitpunkt zu ändern.

In einem aktuellen Projekt der KSI ist geplant, den internen Transport von Paletten in einem Lebensmittel-Distributionszentrum mit *Openshuttle* zu realisieren. Wegen der großen Distanzen und des hohen Durchsatzes für dieses Zentrum wird eine große Zahl an Robotern erforderlich sein. Da der geforderte Durchsatz nahe am maximalen Durchsatz liegt, der mit *Openshuttle* erreicht werden kann, ist der Einsatz eines neu-

en Auftragszuweisungsalgorithmus geplant. Dieser soll auf die Anforderungen, die das Projekt stellt, zugeschnitten sein und so ermöglichen, mit hoher Zuverlässigkeit die vom *Openshuttle*-System geforderte Leistung zu erbringen.

Aufgabenstellung dieser Masterarbeit ist es, einen neuen Algorithmus zur Auftragsverteilung zu entwickeln und zu implementieren. Dieser soll im Gegensatz zur herkömmlichen Auftragsverteilung sowohl die Ladestände der Roboter berücksichtigen als auch durch eine zweistufige Auftragsvergabe eine nachträgliche Änderung einer Auftragszuweisung ermöglichen. Um für die Entwicklung des neuen Algorithmus die bestehende Simulation verwenden zu können, müssen zunächst die Roboter der Simulation mit einem Ladestand versehen werden. Dafür soll das Entladeverhalten der Roboter in der Realität untersucht und in die Simulation integriert werden.

Um die zweistufige Vergabe von Transportaufträgen zu realisieren muss auch das System angepasst werden. Im ersten Schritt wird dem Roboter nur ein Bereich zugewiesen, in den er sich begeben soll. Bis zum Zeitpunkt, an dem er dort ankommt, kann der Auftragsverteilungsalgorithmus noch entscheiden, dass ein anderer Auftrag in diesem Moment besser für diesen Roboter geeignet ist. Durch die Zuweisung dieses Auftrags unterbricht der Roboter die Fahrt und begibt sich zum neu zugewiesenen Bereich.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, durch die Entwicklung eines neuen Algorithmus für die Verteilung von Transportaufträgen an Roboter den Durchsatz des Transportsystems *Openshuttle* zu steigern. Gelingt dadurch eine ausreichende Verbesserung, kann auch die Anzahl der für ein Projekt erforderlichen Roboter reduziert werden.

Durch die Erweiterung der Simulation mit Ladeständen soll ermöglicht werden, bei der Entwicklung der neuen Auftragsverteilung den Ladestand der Roboter zu verwenden. Dazu wird das Lade- und Entladeverhalten der Roboter untersucht, um es möglichst realitätsnahe in die Simulation einbauen zu können. Die Erweiterung der Simulation um den Ladestand der Roboter bietet in Zukunft auch die Möglichkeit, einen Wechsel von Batterien zu *Supercaps* als Energiequelle zu untersuchen.

Ziel dieser Arbeit ist es ebenfalls, die Schritte, die zur Anpassung des Systems an eine zweistufige Auftragsvergabe notwendig waren, darzustellen.

2 Umfeld

Roboter werden zwar schon seit langer Zeit in vielen Bereichen der Industrie erfolgreich eingesetzt werden, doch ist der Einsatz von autonomen und intelligenten Transportroboter erst am Entstehen. Diese Entwicklung wird beschleunigt durch das wachsende Bedürfnis nach Flexibilität, ausgelöst durch steigende Kundenorientierung und Verringerung der Produktlebenszyklen.

Im folgenden Kapitel wird die Nutzung von mobilen Transportrobotern in der Intralogistik als Möglichkeit der Flexibilitätserhöhung und Kostenreduktion beschrieben.

2.1 Roboter als Alternative zu herkömmlichen Transport-Systemen

In vielen Bereichen der Intralogistik wird heute Automation eingesetzt. Trotzdem werden noch viele Transporte manuell durchgeführt. Daraus folgen hohe Lohnkosten und ein hohes Verletzungsrisiko.

Eine Alternative zum manuellen Transport bietet eine fest verbaute Fördertechnik. Diese ist jedoch sehr aufwendig in der Installation, kostenintensiv und unflexibel bezüglich Layoutänderungen. Außerdem ist durch die serielle Ausführung die Ausfallsicherheit sehr nieder, da bei einem Ausfall eines Teils der Fördertechnik die gesamte Anlage stillsteht.

Eine andere Möglichkeit des Transports besteht im Einsatz von fahrerlosen Transportsystemen. Aus [13] ist zu entnehmen, dass diese, auch als Automated Guided Vehicle (AGV) bezeichneten Fahrzeuge, schon seit mehreren Jahrzehnten in der Logistik zum Einsatz kommen: Neben ihrem Haupteinsatzgebiet in der Automobilindustrie, sind sie in vielen anderen Bereichen wie der Papier-, Textil- oder Stahlerzeugung in Verwendung. Sogar Fahrzeuge für den Transport von 65 Tonnen schweren Ladungseinheiten wurden in den Häfen von Rotterdam und Brisbane erfolgreich automatisiert. Um AGVs einsetzen zu können, muss die Einsatzumgebung jedoch bestimmten Voraussetzungen genügen. Dazu gehören vor allem ein stufenfreier flacher Boden und eine entsprechende Infrastruktur zur Lenkung der Fahrzeuge. Dies sind meist Pfade, die durch Bodenmarkierungen oder Induktionsleitungen gekennzeichnet sind. Alternativ werden auch rotierende Laser verwendet, welche mit Hilfe von reflektierenden Orientierungspunkten und Triangulation ihre Position bestimmen können. In beiden Fällen muss diese notwendige Infrastruktur durchgehend von Hindernissen frei gehalten werden: Bei Lasernavigation darf

der Sichtkontakt zum Orientierungspunkt nicht unterbrochen werden, und bei Induktionsleitungen müssen die Fahrzeuge sehr nahe an der Leitung bleiben. Die Einrichtung, Instandhaltung und Änderung dieser Infrastruktur ist mit erheblichen Aufwand verbunden. Die Steuerung der Fahrzeuge erfolgt über eine zentrale Stelle, was bei einer größeren Anzahl an Fahrzeugen einen großen Rechenaufwand bedeutet. Außerdem dürfen zur Erfüllung von Sicherheitsstandards die Fahrzeuge nur mit sehr geringer Geschwindigkeit fahren und nicht mit manuell gelenkten Fahrzeugen in Kontakt kommen.

Der Einsatz einer Flotte von autonomen kooperierenden Robotern verspricht viele Vorteile gegenüber diesen herkömmlichen Systemen. Nicht nur ist die Installation des Systems sehr einfach und in kurzer Zeit möglich, wie in Abschnitt 2.3 beschrieben wird, auch ist keine Infrastruktur für die Navigation der Roboter notwendig [1]. Außerdem können durch die ständige Überwachung ihrer Umgebung die Roboter in einem Umfeld mit Arbeitern oder manuell gesteuerten Fahrzeugen eingesetzt werden, wodurch die erlaubte Geschwindigkeit zur Erfüllung der Sicherheitsstandards deutlich höher sein kann als bei AGVs. Die Steuerung wird von [9] als verteilte Architektur bezeichnet, dabei handelt es sich um eine Kombination aus zentraler und dezentraler Architektur. Dies bedeutet, dass die Lokalisierung, die Bestimmung der Fahrwege, das Ausweichverhalten und das Programm, welches die Roboter durchzuführen haben, dezentral auf jedem Roboter ausgeführt werden. So wird sichergestellt, dass die Roboter möglichst schnell und gut auf das von ihren Sensoren wahrgenommene Umfeld reagieren. Die Koordination der Roboter wird von einer zentralen Stelle aus gesteuert, da so die Effizienz des gesamten Systems maximiert werden kann. Sie beinhaltet neben der Auftragsverteilung auch die Verwaltung der Stationen, die von den Robotern während der Durchführung ihres Programms besucht werden. Diese Kombination aus Systemsteuerung auf höherer Ebene und Bewegungssteuerung auf tieferer Ebene erlaubt es, den komplexen Anforderungen eines intelligenten Logistiksystems zu genügen.

Durch diese Architektur ist eine unvergleichlich größere Flexibilität mit diesem System möglich. Einerseits kann eine Veränderung des Layouts im Robotersystem schnell angepasst werden. Des weiteren kann bei Kapazitätsschwankungen die der jeweiligen Auslastung entsprechende Zahl an Robotern in Betrieb genommen werden, wodurch Energiekosten gespart werden. Schließlich ist auch eine Kapazitätssteigerung durch den Einsatz weiterer Roboter einfach zu realisieren. Im Vergleich dazu muss beispielsweise bei Fördertechnik für eine Kapazitätserweiterung meist das gesamte Layout verändert werden, was mit hohen Kosten und einem Produktionsstopp verbunden ist.

2.2 Das *Robot Operating System*

Das Produkt *Openshuttle* arbeitet mit der Software der incubed IT. Diese basiert auf dem Robot Operating System (ROS), einem Software-Framework für Roboter. Es wurde 2007 am *Stanford Artificial Intelligence Laboratory* entwickelt und wird seit 2008 vom Robotik Forschungsinstitut *Willow Garage* weiterentwickelt [16]. Ziel der Software ist, sie mit nur geringem Änderungsaufwand auf unterschiedlichen Robotern zum Laufen zu bringen.

ROS verfügt über die Standard-Funktionalitäten eines Betriebssystems wie zum Beispiel Hardware-Abstraktion, Implementierung von allgemeinen Funktionalitäten, Nachrichtendienste zwischen Prozessen, Paketverwaltung etc. Es basiert auf einer Graph-Architektur, in der die Arbeitsabläufe in Knoten ausgeführt werden, welche Nachrichten senden oder empfangen können. ROS ist für Unix-basierte Systeme entwickelt worden, wobei Ubuntu als *supported* und andere Systeme wie *Fedora* oder *Mac OS X* als experimentell angegeben werden.

Viele Forschungsinstitutionen entwickeln mittlerweile Projekte mit ROS, wodurch die Anzahl an bereits verfügbarer Software stetig steigt. Außerdem haben durch die weite Verbreitung des Systems viele Hersteller von Sensoren und Aktoren ihre Produkte für die Benutzung mit diesem angepasst. Die Software läuft unter der BSD-(Berkley Software Distribution) Lizenz und ist Open Source Software. Die Wiederverwendung und Weiterentwicklung von Code wird angestrebt, viele Pakete wie zum Beispiel für Visualisierung, Lokalisierung oder Navigation sind bereits vorhanden und ersparen den Entwicklern viel Aufwand in neuen Projekten.

Die Kommunikation zwischen ROS-Knoten erfolgt auf zwei Arten: über ROS-Messages und ROS-Services. Bei einer Kommunikation über ROS-Messages werden in einem Knoten Themen (topics) definiert, zu denen sie Nachrichten veröffentlichen. Andere Knoten können diese topics abonnieren und empfangen dann alle Daten, die von den veröffentlichenden Knoten gesandt werden. Jede Nachricht, die zu einem Topic veröffentlicht wird, muss dem Typ des topics entsprechen. Dieser wird in einer deklarativen Datei definiert und kann beliebig viele Datentypen enthalten.

Möchte man jedoch eine Anfrage von einem Knoten an einen anderen senden, so muss man ein ROS-Service erstellen. Der Knoten, an den die Anfrage gestellt wird, deklariert ein Service mit einem eigenen Typ für die Anfrage und die Antwort, und einer *ResponseBuilder*-Methode. Diese bekommt die Anfrage übergeben und erstellt daraus die Antwort. Der Aufruf eines Services in ROS ist blockierend, das bedeutet, der aufrufende Knoten wartet mit der Ausführung des restlichen Codes darauf, eine Antwort

vom Service des anderen Knotens zu erhalten.

Die nachrichtenbasierte Kommunikation gestattet es auch, ein System über mehrere Rechner zu verteilen. So macht es beispielsweise keinen Unterschied, ob ein Roboter eine Anfrage an einen seiner eigenen Knoten oder an den zentralen Server stellt.

Die Entwicklung von ROS-Knoten war anfangs an die Programmiersprachen *C++* und *Python* gebunden, mittlerweile ist es mit der *Rosjava*-Bibliothek auch möglich, Knoten in Java zu entwickeln.

2.3 Funktionsweise des Produkts *Openshuttle*

Das Produkt *Openshuttle* wird von der vor zwei Jahren gegründeten Firma incubed IT entwickelt. Diese besteht aus einer Gruppe von Absolventen der TU Graz, die während ihres Studiums über den RoboCup Erfahrungen im Bereich der Robotik erlangt haben und nun in Kooperation mit der Firma Knapp Systemintegration an dieser neuen Form des intralogistischen Transports arbeiten. Noch ist kein Lager mit diesem System in Betrieb, es sind jedoch bereits Verträge für den Bau von Lager mit diesen System abgeschlossen worden.

Bisher wurden zwei Arten von Robotern entwickelt: eine für den Transport von Behältern mit einer Grundfläche von $600mm \times 400mm$ und eine für den Transport von Euro-Paletten. Die Aufnahme und Abgabe von Transporteinheiten ist über Fördertechnik oder - bei Behältern - auch durch Lagerarbeiter möglich.

Soll das System in einem neuen Lager in Betrieb genommen werden, muss erst die Karte des Standorts aufgezeichnet werden. Dies geschieht, indem zuerst alle Hindernisse, die nicht permanent an diesem Ort sind, entfernt werden und dann ein Roboter mit einem Joystick durch das Lager gelenkt wird. Ist diese Karte gespeichert, so können im Editor die Standorte der Aufnahme- und Abgabestellen sowie Ladestationen eingezeichnet werden. Außerdem ist es möglich, Wartepositionen und Verkehrsregeln, zum Beispiel Einbahnen, zu definieren. Ist dieser „Lageplan“ erstellt, ist die Inbetriebnahme-Phase abgeschlossen und das System ist funktionsbereit.

Die Karte befindet sich auf dem zentralen Server und wird vom Flotten Management System (FMS) an die einzelnen Roboter verteilt. Auf dieser Grundlage basiert die Pfadplanung des Roboters, die aus einem globalen Pfad und einem lokalen Pfad besteht. Der globale Pfad wird mit dem Dijkstra-Algorithmus berechnet und führt von der Position des Roboters zum Ziel, unter Berücksichtigung der in der Karte eingezeichneten Hindernisse. Der lokale Pfad wird auf Grund der vom Laser gemessenen Umgebung eine gewisse Entfernung vom Roboter aus nach vorne berechnet. Dadurch kann der Roboter

einem Hindernis auf dem globalen Pfad ausweichen, das beim Aufzeichnen der Karte noch nicht vorhanden war.

Der *Trajectory Planner* berechnet unterschiedliche Möglichkeiten eines Pfads und einer Geschwindigkeit. Ist die beste Lösung bestimmt, so wird der lokale Pfad in Fahrkommandos umgewandelt und an die Fahrwerkssteuerung des Roboters geschickt. Um Kollisionen unter Robotern zu vermeiden und ein gutes Ausweichverhalten zu ermöglichen, wird dieser lokale Pfad auch mit den anderen Robotern im Einsatz geteilt. Kreuzen sich zwei Pfade, so berechnet der Roboter mit der niedrigeren Priorität seinen Pfad neu. So können auch durch dieses vorausschauende Planen sogenannte *Deadlocks* vermieden werden, Situationen also, in denen kein Roboter mehr ausweichen kann und das System zum Stillstand kommt.

2.4 Software-Architektur

Die KSI sieht vor, das Produkt Openshuttle in unterschiedlichen Projekten einzusetzen. Dabei soll es einem KSI-System oder einem Fremdsystem untergeordnet sein, welches Transportaufträge an das *incubed IT*-System zur Durchführung übergibt. Das *incubed IT*-System selbst besteht aus dem FMS und den Systemen, welche auf den eingesetzten Robotern laufen. All diese System funktionieren mit ROS und kommunizieren über dessen Nachrichtensystem. Am Roboter selbst laufen einerseits Knoten, die für die grundlegende Funktionsfähigkeit und Steuerung des Roboters notwendig sind, wie zum Beispiel die Treiber der Sensoren, oder die Knoten, die dem Roboter die Lokalisierung oder Navigation auf einer Karte ermöglichen. Diese Knoten sind in C++ geschrieben und stammen aus der Basisimplementierung von ROS, wurden jedoch für die Verwendung im *incubed IT*-System angepasst. Eine zweite Form von Knoten ist für die spezielle Funktionalität des *incubed IT*-Systems zuständig. Sie wurden zur Gänze neu und in Java entwickelt.

Um ein Programm festlegen zu können, welches das Verhalten des Roboters auf oberster Ebene bestimmt, wurde von der incubed IT eine Finite State Machine (FSM) entwickelt. Durch sie ist es möglich, für den Roboter ein beliebiges Programm aus bestehenden Bausteinen zusammenzusetzen oder ein neues Verhalten zu entwickeln. Das Default-Programm beispielsweise versucht, einen Transportauftrag für den Roboter, auf dem die FSM läuft, zu reservieren und diesen, falls die Reservierung erfolgreich war, auszuführen. Dieser Ablauf kann durch Hinzufügen von Zuständen und Übergängen zur Finite State Machine beliebig verändert oder erweitert werden. Eine genauere Beschreibung folgt in Abschnitt 4.1.2.

2.5 Projekt Lebensmittel-Distributionszentrum

Die Lebensmitteldistribution stellt an den Händler besondere Herausforderungen, besonders wenn er ein Vollsortiment anbietet. Ein solches umfasst ein breites Artikelspektrum, an jede der vielen Filialen wird jedoch nur eine kleine Menge jedes Artikels geliefert. Viele Artikel müssen bei einer bestimmten Temperatur gelagert werden, sind wegen ihrer Form nicht stapelbar oder sind empfindlich. Wegen ihrer begrenzten Haltbarkeit sind viele Artikel nur relativ kurz im Distributionszentrum und werden schon bald nach ihrer Anlieferung wieder ausgeliefert. Für die Kommissionierung in diesem Zentrum ist vorgesehen, eine Packbild-Software einzusetzen. Ziel ist es, die Berechnung der auszuliefernden Ladungsträger so spät wie möglich zu starten, um zu diesem Zeitpunkt schon über möglichst genaue Informationen über die Verfügbarkeit der gewünschten Artikel im Lager zu verfügen. Aus diesen Gründen sind die internen Transportaufträge, die den Robotern zugewiesen werden sollen, erst spät abrufbar.

Das Projekt, in dem der neue Auftragszuweisungs-Algorithmus zum Einsatz kommen wird, wurde ursprünglich mit einer Paletten-Fördertechnik geplant. Durch die Fortschritte, die bei der Entwicklung des Produkts *Openshuttle* gemacht wurden und angesichts der Vorteile, die sich daraus ergeben, wurde stattdessen der Einsatz von Paletten-Shuttles beschlossen. Dieses Projekt wird voraussichtlich eines der ersten sein, bei welchem *Openshuttle* in Betrieb genommen wird. Aufgrund der fehlenden Erfahrungswerte ist besonderer Wert auf eine umfangreiche Simulation zu legen.

Das Distributionszentrum ist folgendermaßen aufgebaut: Es besteht aus zwei Teilen, die sehr ähnlich aufgebaut sind: dem *Frischdienst* und dem *Trockensortiment*. Größe und Raumaufteilung der beiden Teilen sind identisch, es gibt ausschließlich Unterschiede bei den geplanten Durchsätzen und somit bei den Kapazitäten der einzelnen Komponenten.

Am Wareneingang werden die Güter auf Paletten angeliefert und von dort entweder zum Hochregallager (HRL) oder zu einer der Depalettierstationen gebracht. An einer Depalettierstation werden Artikel von den Paletten in kleinere Trays geladen und im Open Storage and Retrieval (OSR)-System gelagert. Im Lager befinden sich zwei Arten von Kommissionierstationen: Palettenkommissionierstationen und Traykommissionierstationen. Erstere werden aus dem HRL und Letztere aus dem OSR bedient. Kommissioniert wird auf Rollbehälter, die im Lebensmittelhandel übliche Transporteinheit. Diese werden nach Abschluss des Kommissioniervorgangs von AGVs zum Warenausgang gebracht.

Für den Transport mit Shuttles sind alle Strecken relevant, auf denen Paletten innerhalb des Distributionszentrum transportiert werden. Dies betrifft die Strecke von Wareneingang zu Hochregallager, die Strecke von Hochregallager zu Palettenkommis-

sionierstation oder Depalettierstation und die Transportstrecke der Leer-Paletten zur Leergutstation. Das Hochregal besteht aus fünf Regalgassen, in denen jeweils ein Regalbediengerät die Paletten von einer Übergabestelle übernimmt und einlagert bzw. auslagert und übergibt. Es gibt zwei Arten von Depalettierstationen, eine manuelle und eine automatische. Dort werden die Paletten entweder ganz oder nur teilweise entladen, im ersten Fall müssen die leeren Paletten zum Leergutstation gebracht werden, im zweiten Fall werden die teilentladenen Paletten zurück ins HRL gebracht.

Die Grundfläche des Lagerbereichs, in dem sich Shuttles bewegen werden, beträgt $87m \times 72m$. Die längste Distanz, die ein Shuttle für einen Transport zwischen zwei Stellen zurücklegen muss, beträgt 85 Meter. Bei einer maximal erlaubten Geschwindigkeit von $1,5\frac{m}{s}$ bedeutet diese Strecke eine Transportdauer von mindestens 50 Sekunden.

3 Batteriemanagement in der Simulation

Die Optimierung der Auftragsverteilung unter Berücksichtigung der einzelnen Batterieladestände der Roboter setzt eine realitätsnahe Abbildung des Ladeverhaltens der Roboter in der Simulation voraus. Gegenstand dieses Kapitels ist die Beschreibung der zur Realisierung einer solchen Abbildung erforderlichen Schritte.

3.1 Messung von Entladevorgängen an realen Robotern

Beobachtet man den Stromverbrauch eines Roboters im Testbetrieb, kann man feststellen, dass es mehrere Zustände gibt, in denen der Stromverbrauch konstant bleibt:

- Ruhelage mit abgestellten Servomotoren (Verbrauch ca. $4,5A$)
- Ruhelage mit eingeschalteten Servomotoren (Verbrauch ca. $7A$)
- Fahren mit konstanter Geschwindigkeit (Verbrauch je nach Geschwindigkeit)

Für den Stromverbrauchs während eines Beschleunigungsvorgangs sind zwei Faktoren maßgeblich: die Stärke der Beschleunigung und die jeweilige Geschwindigkeit.

Um nun den Zusammenhang zwischen dem Fahrverhalten des Roboters und der Entladung der Batterien bestimmen zu können, wurde über die Dauer einer gesamten Entladung der Batterien die aktuelle Geschwindigkeit und der aktuelle Stromverbrauch mit einer Frequenz von einem Hertz aufgezeichnet. Diese Messung wurde bei zwei unterschiedlichen Robotern insgesamt fünf mal durchgeführt, wobei sich diese, nach einer vollständigen Aufladung in der Ladestation, zwischen zwei Punkten bewegten. Jeder der Roboter besuchte abwechselnd die 8m voneinander entfernten Punkte so lange, bis sein Ladezustand so niedrig war, dass er vom Steuerungsprogramm zur Ladestation geschickt wurde. Durch dieses Vorgehen wurde eine Datenmenge von 75 800 Log-Einträgen mit der jeweiligen Zeit, Geschwindigkeit und dem Stromverbrauch aufgezeichnet. Die Aufzeichnung der Geschwindigkeit erfolgte in den im Robot Operating System üblichen Einheiten *velocity linear* und *velocity angular*, also die Translations- und Rotationsgeschwindigkeiten. Diese wurden bei der weiteren Verarbeitung der Daten über Gleichung 1 und Gleichung 2 in die einzelnen Radgeschwindigkeiten umgerechnet und aufsummiert.

$$v_{linear} = \frac{R + L}{2} \tag{1}$$

$$v_{angular} = \frac{L - R}{D} \tag{2}$$

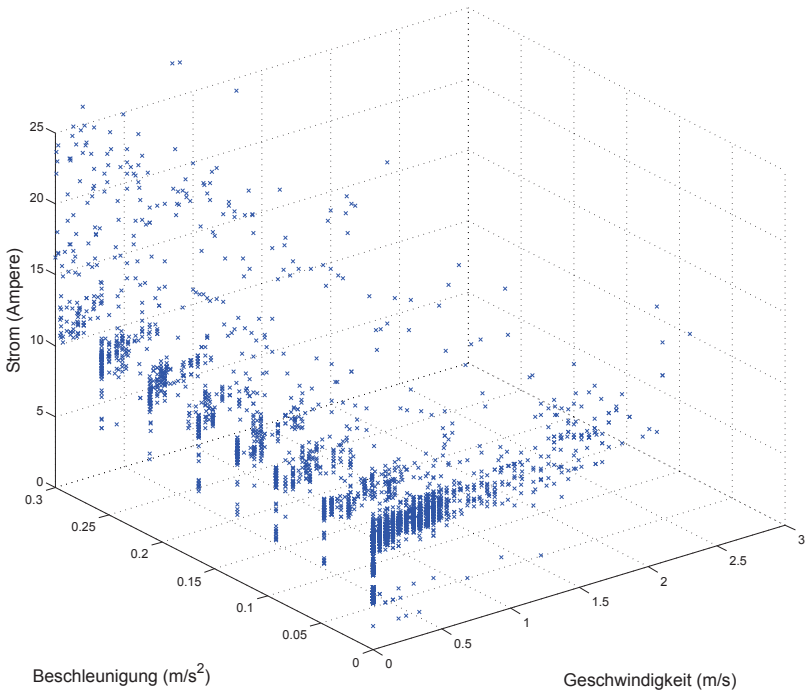

Abbildung 1: Stromverbrauch in Abhängigkeit von Geschwindigkeit und Beschleunigung

3.2 Bestimmung von Regressionskoeffizienten

Um den momentanen Stromverbrauch eines Roboters in der Simulation bestimmen zu können, müssen die am realen Roboter erfassten Daten untersucht werden. Stellt man den Stromverbrauch in Abhängigkeit von Geschwindigkeit und Beschleunigung in einem Dreidimensionalen Diagramm dar (siehe Abbildung 1), so ist ein direkter Zusammenhang zwischen diesen 3 Größen zu erkennen. Mithilfe von multipler linearer Regression ist es möglich, aufgrund von mehreren Variablen(*Prädikatoren*) den Wert einer anderen Variable (*Kriterium*) vorherzusagen. Dabei werden über die Methode der kleinsten Quadrate die Koeffizienten β_i bestimmt. Mit diesen ist es dann durch Einsetzen in Gleichung 3 möglich, anhand der momentanen Geschwindigkeit v und Beschleunigung a den aktuellen Stromverbrauch A zu bestimmen.

$$A = \beta_0 + \beta_1 * v + \beta_2 * a \tag{3}$$

Der Koeffizient β_0 beschreibt dabei den Stromverbrauch des Roboters im Stillstand, also bei $v = a = 0$. Da die Servomotoren erst fünf Sekunden nach der letzten Bewe-

gung abgeschaltet werden, sinkt erst danach der Stromverbrauch. Im Testfall jedoch, der für die Aufzeichnung der Daten durchgeführt wurde, kam es zu keinen fünf oder mehr Sekunden dauernden Stillständen, weshalb dies bei der Berechnung der Koeffizienten nicht zu berücksichtigen ist, wohl aber in der Umsetzung des Stromverbrauchs in der Simulation. Abbildung 2 stellt für den dreidimensionalen Wertebereich die berechneten Koeffizienten als Ebene dar.

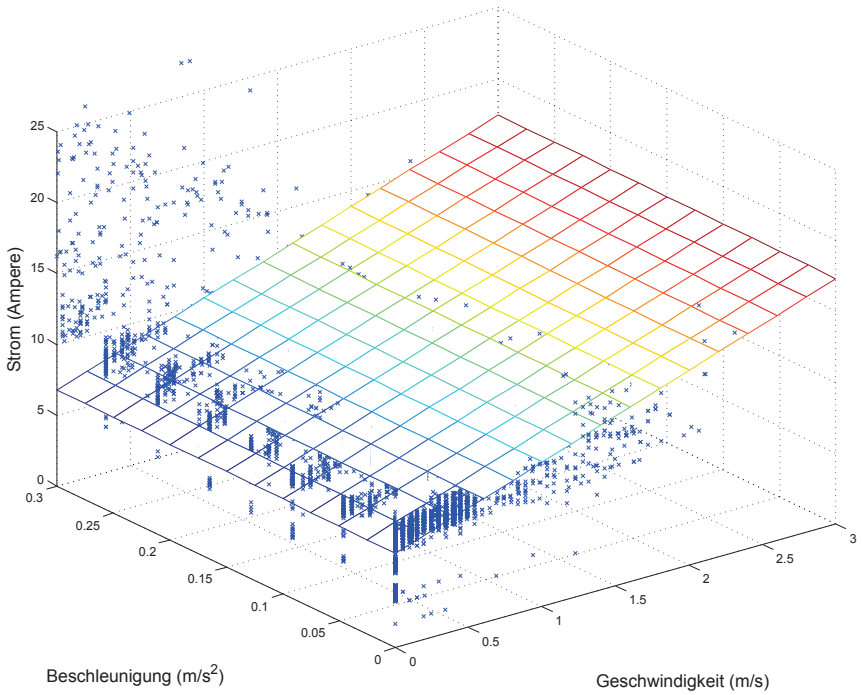

Abbildung 2: Darstellung der durch lineare Regression bestimmten Koeffizienten

3.3 Testergebnisse

Um die Plausibilität dieser Schätzung zu überprüfen, wurde für jede Messung der Stromverbrauch anhand der berechneten Koeffizienten berechnet. Es wurde die Abweichung zwischen dem geschätzten und dem gemessenen Stromverbrauch berechnet und ein Durchschnittswert über alle Abweichungen gebildet. Die durchschnittliche Abweichung des geschätzten Stromverbrauchs liegt bei $5,14\%$ des gemessenen Werts.

Die Messung des Stromverbrauchs erfolgt am Roboter mithilfe des Batteriemanage-

mentsystems. Dieses muss auf jedem Roboter manuell kalibriert werden, deshalb ist ein gewisser Unterschied bei der Messung des Stromverbrauchs an den beiden Robotern nicht zu vermeiden. Die Koeffizienten und den Vergleich nur anhand der Daten eines einzigen Roboters zu berechnen, würde zwar erlauben, eine geringere Abweichung zu erhalten. Für eine projektbezogene Simulation eines Warenlagers ist jedoch zu bevorzugen, die Daten verschiedener Roboter heranzuziehen, weil dadurch ein realistischeres Ergebnis erzielt werden kann.

Abbildung 3 stellt einen Ausschnitt aus den verglichenen Daten dar. Es ist ersichtlich, dass das Verhalten des geschätzten Verbrauchs mit dem des gemessenen Verbrauchs übereinstimmt, bei der Höhe des Stromverbrauchs sind jedoch teilweise Abweichungen zu erkennen. Diese Abweichungen sind auf Schwankungen im Stromverbrauch zurückzuführen, die nicht die Motoren verursachen, sondern andere Verbraucher wie der Industrierechner oder die Sensoren, deren Bedarf von der Geschwindigkeit des Roboters unabhängig ist. Diese Abweichungen sind jedoch so gering, dass sie für die Simulation des Entladeverhaltens eines Roboters vernachlässigt werden können.

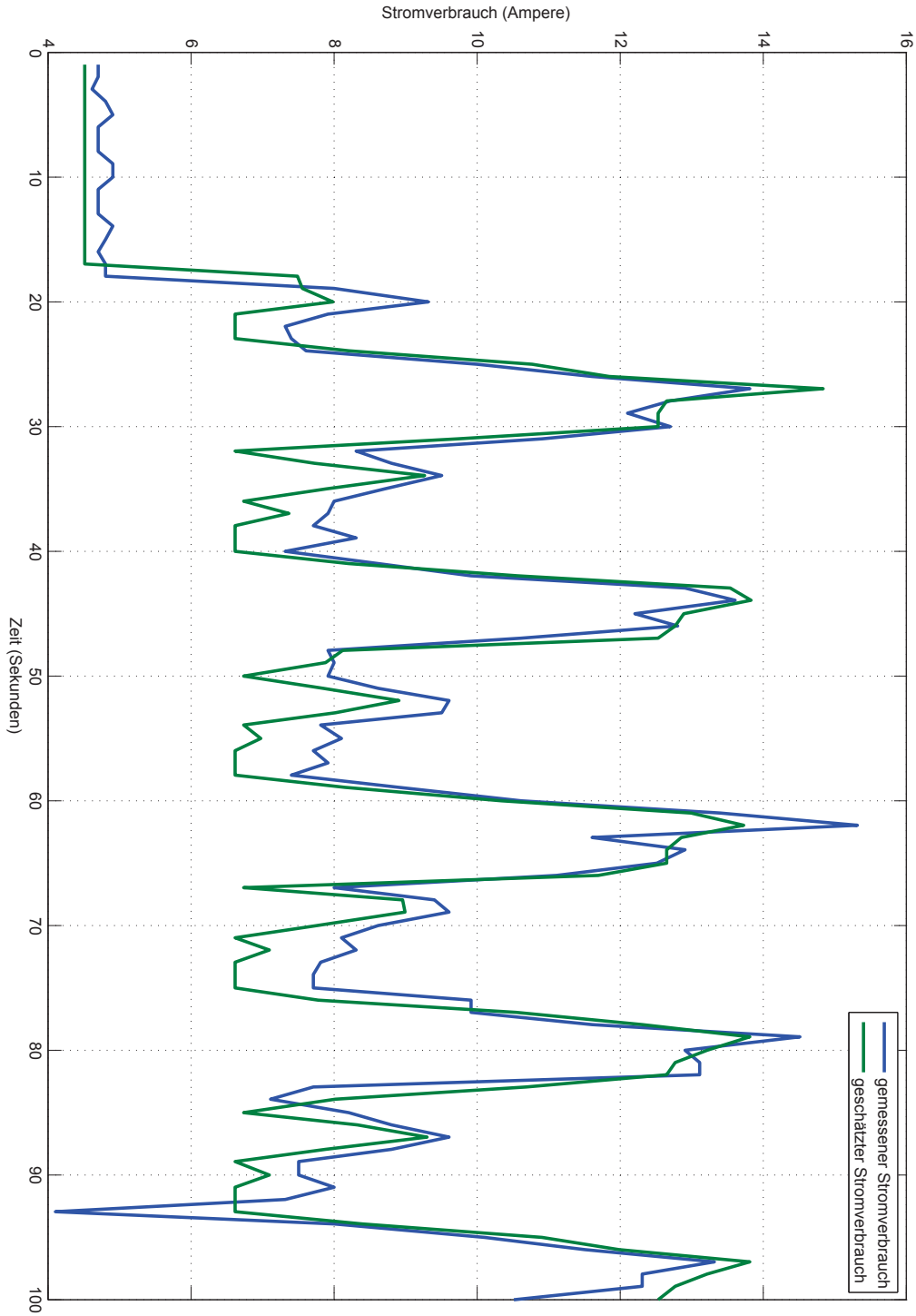

Abbildung 3: Vergleich zwischen gemessenem und geschätztem Stromverbrauch

3.4 Darstellung des Stromverbrauchs

Die Information, in welchem Ladezustand sich die Batterie des Roboters befindet, wird am realen Roboter vom Batteriemanagementsystem (BMS) verwaltet; dieses misst die aktuelle Spannung an den Batteriezellen. Dadurch kann das BMS die Batterie zwar gut vor Unter- oder Überladung schützen, aufgrund der schwachen Spannungsabnahme während der Entladung und des abrupten Spannungsabfalls gegen deren Ende ist es jedoch schwierig, die noch verbleibende Nutzzeit vorherzusagen. Das Programm des Roboters berücksichtigt diesen Ladezustand, und fällt er unter einen konfigurierbaren Wert, so nimmt der Roboter keine Aufträge mehr an und fährt zu einer Ladestation. Dieser Wert ist im aktuellen System auf 35% eingestellt.

In der Simulation ist es jedoch nicht sinnvoll, den nicht-linearen Verlauf der Batterie-Spannung darzustellen. Stattdessen wird die Ladung der Batterie in Amperesekunden dargestellt und von dieser der in jeder Sekunde verbrauchte Strom abgezogen. Um das Entladeverhalten des Roboters in der Simulation abzubilden, muss die maximal verfügbare Ladung bestimmt werden. Dazu wurden die Daten der aufgezeichneten Entladevorgänge betrachtet. Ein Entladevorgang entspricht einer Abnahme des BMS-Ladestands von 100 auf 35 Prozent. Summiert man in den aufgezeichneten Daten den Stromverbrauch zu jeder Sekunde auf, so lässt sich feststellen, dass dieser Wert bei allen gemessenen Entladungen ähnlich ist und ungefähr 120.000 Amperesekunden beträgt. Wird dieser Wert als 65 -Prozent der Gesamtladung angenommen, so entsprechen 185000 Amperesekunden 100 Prozent. Dieser Wert wurde in der Simulation als volle Beladung übernommen. Ausgehend von diesem Wert der höchstmöglichen Ladung wird dann der prozentuale Ladestand der Simulation berechnet, wobei aus dem Anteil des momentanen Ladestands laufend die verbrauchten Amperesekunden abgezogen werden.

3.5 Integration des Energiemanagements in die Simulation

Durch den Nachrichten-basierten Aufbau von ROS ist es mit geringem Aufwand möglich, einen Roboter zu simulieren. Die Simulation erfolgt mithilfe des Pakets *Stage*. Dieses simuliert Sensordaten und erlaubt damit dem System, auf seine Umwelt so zu reagieren, als würde es in der Realität laufen. Somit ist es möglich, die exakt gleiche Steuerung wie beim tatsächlichen Einsatz des Roboters zu verwenden, was zu einem sehr realitätsnahen Verhalten des Roboters in der Simulation führt.

Die Steuerung der Simulation erfolgt über den Node *stageros*, welche in C++ geschrieben ist. An dieser Stelle wurde die Batterieverwaltung für alle eingesetzten Robo-

ter angesiedelt. In einer Klassenvariable wird der momentane Ladestand der Roboter in Amperesekunden gespeichert. Ebenfalls in diesem Node wird der aktuelle Ladestand als prozentualer Anteil am gesamt möglichen Ladestand in der Nachricht *robotHardwareStatus* veröffentlicht. An dieser Information erkennt die Finite State Machine, zu welchen Moment der Robter zu einer Ladestation fahren muss und wann der Ladestand hoch genug ist, um die Ladestation wieder zu verlassen.

Wird dieser Vorgang von einem realen Roboter durchgeführt, so reicht es für das Aufladen des Roboters aus, an eine Ladestation zu fahren und den Ladestand zu beobachten. In der Simulation ist es jedoch notwendig zu erkennen, wann der Roboter an einer Ladestation steht, ohne in seine Steuerung (welche ja dieselbe wie in Realität ist) einzugreifen. Ist dies der Fall, so kann der momentane Ladestand um eine bestimmte Rate erhöht werden. Ist der gewünschte Ladestand erreicht, verlässt der Roboter die Station, und die Simulation muss statt zu laden wieder entladen.

Mit einer Frequenz von einem Hertz wird die Methode *calculateChargeState* aufgerufen. Sie bestimmt zu Beginn, ob sich der Roboter an einer Ladestation befindet oder nicht. Dazu ist die Node *stageors* am topic */map_goals* abonniert. Dieses topic veröffentlicht alle Ziele der Karte mit ihrer Position und ihrem Typ. So ist es möglich, die Position des Roboters mit allen Ladestationen zu vergleichen und festzustellen, ob der Ladestand im Moment zu- oder abnimmt. Wie die Ab und Zunahme berechnet wird, wird in Abschnitt 3.5.1 und Abschnitt 3.5.2 beschrieben.

Ist die Veränderung des Ladestands für diese Sekunde bestimmt, so wird für jeden Roboter der momentane prozentuale Ladestand als Anteil vom aktuellen Ladestand in Amperesekunden an der maximalen Anzahl an Amperesekunden berechnet und auf dem topic *simulation_charge_state* veröffentlicht. Die *SimulationRobotDriverNode* ist auf dieses topic abboniert, sie ist für die Verwaltung aller hardwarespezifischen Informationen zuständig, die in der Simulation künstlich erzeugt werden müssen. Sie veröffentlicht diese Informationen auf dem topic *robot_hardware_status*, über welches bei einem realen Roboter tatsächlich gemessene Sensordaten veröffentlicht werden. So kann sichergestellt werden, dass diese künstlich erzeugten Messwerte keinen Unterschied für das ansonsten gleich funktionierende System darstellen.

3.5.1 Simulation des Entladevorgangs

Ist der Roboter gerade in Bewegung oder er befindet sich nicht an einer Ladestation, so wird die Summe der Geschwindigkeiten der beiden Räder berechnet. Danach wird die Beschleunigung aus der Differenz der letzten Messung, welche in einer eigenen Va-

riable gespeichert ist, und der aktuellen Geschwindigkeit gebildet. Ist diese negativ, so wird sie auf Null gesetzt. Anhand dieser Informationen und der in der Klasse gespeicherten Koeffizientenwerte aus der Regression wird dann für diese Sekunde eine Stromverbrauchsschätzung in Ampere nach Gleichung 3 berechnet und von den gesamten Amperesekunden des Ladestands abgezogen.

3.5.2 Simulation des Aufladevorgangs

Der gemessene Ladestrom an den Robotern beträgt 30 Ampere. Wird festgestellt, dass sich der simulierte Roboter in der Ladestation befindet, so werden bei jedem Aufruf der Methode *calculateChargeState* 30 Amperesekunden dem momentanen Ladestand hinzugefügt. Wird der maximal mögliche Ladestand erreicht und der Roboter verlässt trotzdem nicht die Ladestation, so bleibt der Ladestand auf diesem maximalen Wert und steigt nicht mehr weiter an.

4 Implementierung der zweistufigen Auftragsvergabe

Für die Vergabe der Aufträge an die Roboter ist die Implementierung eines zweistufigen Systems geplant, um die Zeit, die ein Roboter für das Erreichen eines Ziels benötigt, noch für eine Optimierung der Auftragsvergabe nutzen zu können.

Dafür muss in zwei bestehende Systeme eingegriffen werden: Einerseits muss die Auftragsverwaltung, die am zentralen Server läuft, angepasst werden. Andererseits ist auch das Programm, welches das Verhalten jedes einzelnen Roboters bestimmt, zu verändern.

4.1 Ursprünglicher einstufiger Ablauf

Im folgenden Teil wird zuerst näher auf die Funktionsweise dieser zu ändernden Systeme eingegangen. Danach werden in Abschnitt 4.2 die Änderungen beschrieben, die an diesen Systemen notwendig waren, um eine zweistufige Auftragsvergabe zu ermöglichen.

4.1.1 Der Order-Manager

Die Auftragsverwaltung und -Verteilung wird von dem Rosjava-Knoten Order Manager durchgeführt, welcher auf dem zentralen Server läuft. Dieser besteht aus zwei Klassen: der Klasse *OrderManagerNode*, die für die Kommunikation mit dem restlichen System zuständig ist, und der Klasse *BaseOrderManager*, die die Aufträge sammelt und verwaltet.

Wird ein neuer Auftrag über die Methode *addOrder(Order o)* dem *BaseOrderManager* übergeben, so wird dieser in die Liste *startTimeSortedOrders* vom Typ *ArrayList<Order>* eingetragen. Diese wird sofort nach dem Hinzufügen nach der Startzeit der Transportaufträge sortiert. Die Methode *getAvailableOrders()* gibt all jene Aufträge aus *startTimeSortedOrders* zurück, dessen Startzeit bereits erreicht wurde.

Die Methode *reserveOrder()* ruft den in der Konfiguration bestimmten Auftragszuweisungsalgorithmus (*OrderAssignmentAlgorithm*) auf, dieser gibt einen geeigneten Auftrag für den anfragenden Roboter zurück. Der entsprechende Auftrag wird aus der Liste *startTimeSortedOrders* entfernt und mit dem anfragenden Roboter als Key in die *HashMap<String, Order> activeOrders* gegeben.

Alle Methoden des *BaseOrderManager* sind *synchronized*, das bedeutet, sie werden beim Aufruf durch einen Thread für diesen reserviert. Somit kann verhindert werden,

dass durch gleichzeitigen Zugriff von zwei unterschiedlichen Threads inkonsistente Daten entstehen.

Die Klasse *OrderManagerNode* bietet Services an, über die Transportaufträge hinzugefügt oder reserviert werden können. Die Kommunikation mit dem restlichen System erfolgt einerseits durch das Senden und Empfangen von ROS-Nachrichten und andererseits durch das zur Verfügung stellen von ROS-Services (siehe Abschnitt 2.2). Das Service *reserveOrder* wird von dem Agenten des Roboters aufgerufen und gibt das Ergebnis der zuvor beschriebene Methode *reserveOrder* des *BaseOrderManagers* an den Roboter zurück.

Der *UnassignedOrderPublisher* beispielsweise veröffentlicht die Aufträge, die momentan noch nicht reserviert sind. Mit einer Frequenz von $1Hz$ wird eine Nachricht vom Typ *UnassignedOrder* in dem topic */unassigned_orders* versandt. Ein Roboter, der ohne Auftrag an einer Idle-Position steht, erfährt durch das Abbonieren dieses topics, ab wann wieder ein neuer Auftrag zur Verfügung steht.

4.1.2 Die Finite-State-Machine

Wie bereits in Abschnitt 2.3 kurz beschrieben, ist es möglich, den Roboter unterschiedliche Programme ausführen zu lassen. Dafür wird ein Programm als Finite-State-Machine definiert und dem Roboter übergeben. Sobald die automatische Ablaufsteuerung für diesen Roboter aktiviert ist, wird der *Agent* des Roboters gestartet. Dieser Agent ist ein Programm, welches die momentan eingestellte FSM durchläuft und den Code, der in den einzelnen Knoten steht, ausführt. Jeder Knoten einer FSM verfügt über *Transitions* und *Conditions*. Eine *Transition* ist ein Übergang zu einem anderen Knoten, der durchgeführt wird, wenn eine bestimmte Bedingung, eine *Condition*, erfüllt ist. Eine *Default-Transition* ist ein Übergang, der ausgeführt wird, wenn keine der in diesem Knoten bestehenden Bedingungen erfüllt ist.

Die Implementierung der FSM ist generisch: Es ist möglich, eine eigenständige FSM als Knoten einer übergeordneten FSM zu definieren. Dies hat zum Einen den Vorteil, dass die große Anzahl an Knoten in Gruppen zusammengefasst werden kann und dadurch die Übersicht erhalten bleibt. Zum Anderen können bestimmte Verhaltensweisen wie zum Beispiel die Behälteraufnahme oder das Fahren zu einem bestimmten Ziel als eigene FSM definiert werden. So ist es möglich, bei der Erstellung eines neuen Programms diese bereits bestehenden FSM nach dem Baukastenprinzip in dem Programm einzubauen.

Zu Beginn befindet sich der Agent des Roboters in der *dock_global_fsm*. In dieser prüft er seinen Ladestand, unterschreitet dieser eine definierte Grenze, so fährt er zu einer La-

destation. Andernfalls wechselt der Agent des Roboters in den Zustand *reserve_order*. Bei der Ausführung dieses Knotens wird über das nachrichtenbasierte Kommunikationssystem von ROS eine Anfrage an das *reserveOrder*-Service der *OrderManagerNode* gesandt. Ist diese Anfrage erfolgreich, so wird der reservierte Auftrag zurückgegeben. In diesem Fall ist die Bedingung für einen Übergang in den Zustand *Retrieve* erfüllt, und der Agent wechselt in diesen. Konnte kein Auftrag reserviert werden, so wird in die FSM *go_to_idle_fsm* gewechselt. Diese besitzt die Form eines *Conditional-Go-To*-Knotens, welcher die Eigenschaft besitzt, während seiner Ausführung den Roboter so lange zu einem bestimmten Ziel fahren zu lassen, bis eine oder mehrere Bedingungen erfüllt sind.

In diesem Fall ist das Ziel des Roboters die nächst gelegene der Idle-Stationen. Dies sind vom Benutzer definierte Orte, an die Roboter fahren, um nicht andere Roboter oder Arbeiter zu behindern. Die Bedingung, unter der die Fahrt zur Idle-Station unterbrochen wird, ist eine *Order-Available-Condition*: Sobald ein Auftrag zur Verfügung steht, unterbricht der Agent die Fahrt und wechselt zurück zum Startzustand, von welchem er dann erneut versucht, einen Auftrag zu reservieren.

Abbildung 4 stellt die eben beschriebenen Abläufe und Interaktionen dar.

4.2 Zweistufiger Ablauf

Die folgenden Ausführungen beschreiben die Änderungen, die an diesem System vorzunehmen waren, um die angestrebte zweistufige Auftragsvergabe zu ermöglichen.

4.2.1 Cluster

Wie bereits am Anfang dieses Abschnitts erwähnt, ist eine zweistufige Auftragsvergabe geplant, um eine Änderung der Auftragszuweisung eines Roboters noch so lange zu ermöglichen, bis sich der Roboter im Bereich der Warenaufnahme oder -abgabe befindet. Dazu werden Aufnahme- und Abgabestationen, die zur selben logistischen Einheit gehören, zu *Cluster* zusammengefasst. Im Karteneditor kann jeder Station ein Cluster zugewiesen werden und für jeden Cluster muss eine Zone sowie eine repräsentative Position in der Karte eingezeichnet werden. Die repräsentative Position stellt das Ziel des Roboters so lange dar, bis der Roboter die Zone des Clusters erreicht und eine genaue Station als Ziel bekommt.

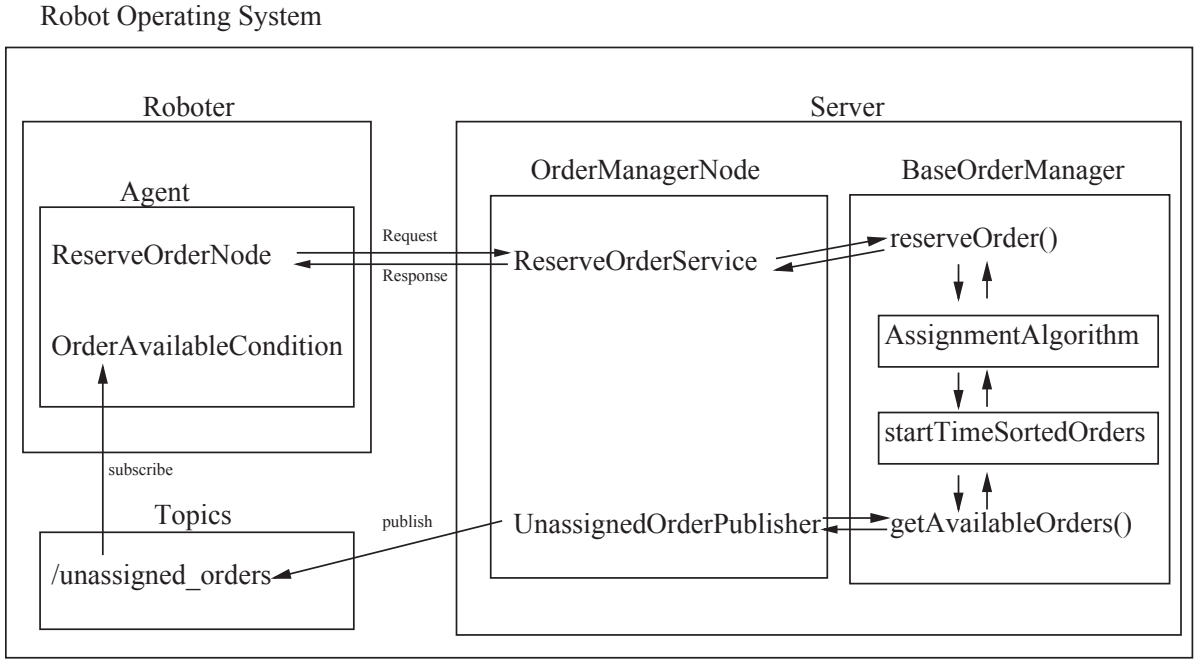

Abbildung 4: Darstellung der Interaktionen zwischen dem Agenten des Roboters und den Klassen des Order Managers

4.2.2 Order Manager

Um die Auftragsverteilung trotz noch fehlender Information über die genaue Quelle und das genaue Ziel durchführen zu können, wird im *BaseOrderManager* zwischen einem *clusterOrder*, also einem Bereich als Ziel, und einem genauen Ziel, einem *preciseOrder*, unterschieden. Bei beiden Objekten handelt es sich um Instanzen der Klasse *Order*, bei einem *clusterOrder* werden jedoch, durch die geringere Informationsmenge, nicht alle Felder verwendet.

Der OrderManager generiert für jeden *preciseOrder*, den er vom übergeordneten System erhält, einen dazugehörigen *clusterOrder*. Die Verwaltung dieser Objekte erfolgt einerseits in der *ArrayList<Order> clusterOrders* und andererseits über die *HashMap<Order, Order> clusterToPreciseOrderMap*, welche für jeden *clusterOrder* auf den entsprechenden *preciseOrder* verweist.

Die Startzeit bzw. die Endzeit in der herkömmlichen, einstufigen Auftragsvergabe geben den frühest möglichen Zeitpunkt der Abholung bzw. den spätest möglichen Zeitpunkt der Abgabe an. Diese zeitlichen Vorgaben werden in der zweistufigen Auftrags-

vergabe als Startzeit und Endzeit eines *preciseOrder* übernommen.

Der neue Auftragszuweisungsalgorithmus weist einem Roboter einen *preciseOrder* zu. Der Roboter bekommt jedoch zuerst vom OrderManager über das neue Service *reserveCluster* nur den dazugehörigen *clusterOrder* zurückgegeben. Solange sich der Roboter nicht in der entsprechenden Cluster-Region befindet, hat der Algorithmus noch die Möglichkeit, den zugewiesenen Auftrag zu ändern. Trifft der Roboter in der Cluster-Region ein und versucht, von dort einen Auftrag über das Service *reserveOrder* zu reservieren, so bekommt er vom OrderManager einen *preciseOrder* zurück, welcher danach vom Algorithmus nicht mehr verändert werden kann.

Bei einstufiger Auftragsvergabe wird vom OrderManager über das topic */assignedOrders* für jeden Roboter, der über einen fest zugewiesenen Auftrag verfügt, dieser veröffentlicht. In der zweistufigen Auftragsvergabe wird über dieses topic zuerst der dem Roboter übergebene *clusterOrder* veröffentlicht. Kommt es in dieser Zeit zu einer Änderung der Auftragszuweisung, so wird dies durch das ständige Veröffentlichen sofort nach außen bekannt gegeben. Sobald der Roboter einen *preciseOrder* reserviert hat, wird dieser bis zu seiner Fertigstellung veröffentlicht.

4.2.3 Finite State Machine

Um das gewünschte Verhalten des Roboters zu erreichen, wurde auch die Finite State Machine verändert. Der Agent besucht zuerst den Knoten *reserve_cluster* und ruft damit den gleichnamigen Service des *OrderManagerNode* auf. Wurde vom *BaseOrderManager* dem aufrufenden Roboter ein Auftrag erfolgreich zugewiesen, so bekommt der Agent des Roboters den entsprechenden *clusterOrder* zurück. Konnte kein Auftrag zugewiesen werden, so wird der leere Wert *null* zurückgegeben und der Agent wechselt in die bereits beschriebene FSM *go_to_idle_fsm*. Bei einer erfolgreichen Anfrage wechselt der Agent in die neu erstellte *go_to_cluster_fsm*. Der erste Knoten dieser FSM ist *go_to_cluster*, welcher vom Typ *Conditional-Go-To* ist. Durch ihn beginnt der Roboter, zu der repräsentativen Position des reservierten Cluster zu fahren.

Es wurden zwei Bedingungen definiert, die diese Fahrt unterbrechen können: Eine *ChangeInAssignmentCondition*, die überprüft, ob es eine Veränderung in der Auftragszuweisung dieses Roboters gegeben hat. Dafür wird das topic */assignedOrders* abonniert, und es wird die momentane Auftragsverteilung festgehalten. Stellt sich bei einem Aufruf heraus, dass sich der dem Roboter zugewiesene Auftrag seit dem letzten Aufruf verändert hat, so gibt die *ChangeInAssignmentCondition true* zurück. In diesem Fall wird die *go_to_cluster_fsm* über den Knoten *final_failure* verlassen, und es wird in ihrer erneu-

ten Ausführung der Cluster des neuen Auftrags angefahren. Die zweite Bedingung ist vom Typ *RobotInRegionCondition* und prüft ob sich der Roboter in der Zone des Cluster befindet. Ist dies der Fall, wird zum Knoten *reserve_order* gewechselt. Bei dessen Ausführung wird eine Anfrage an den *reserveOrder*-Service der *OrderManagerNode* gesandt, ist diese erfolgreich, wird die *go_to_cluster_fsm* über den *final_success*-Knoten verlassen, falls nicht, über *final_failure*.

Im Falle einer erfolgreichen Auftragsreservierung wird über die FSMs *Retrieve* und danach *Deliver* der Transportauftrag ausgeführt. War die Reservierung nicht erfolgreich, wird zurück zur *dock_global_fsm* gewechselt und der beschriebene Ablauf von vorne begonnen.

Abbildung 5 stellt ein vereinfachtes Modell der veränderten *RobotExecutionFiniteStateMachine* und ein genaues Modell der neu erstellen *go_to_cluster_fsm* dar.

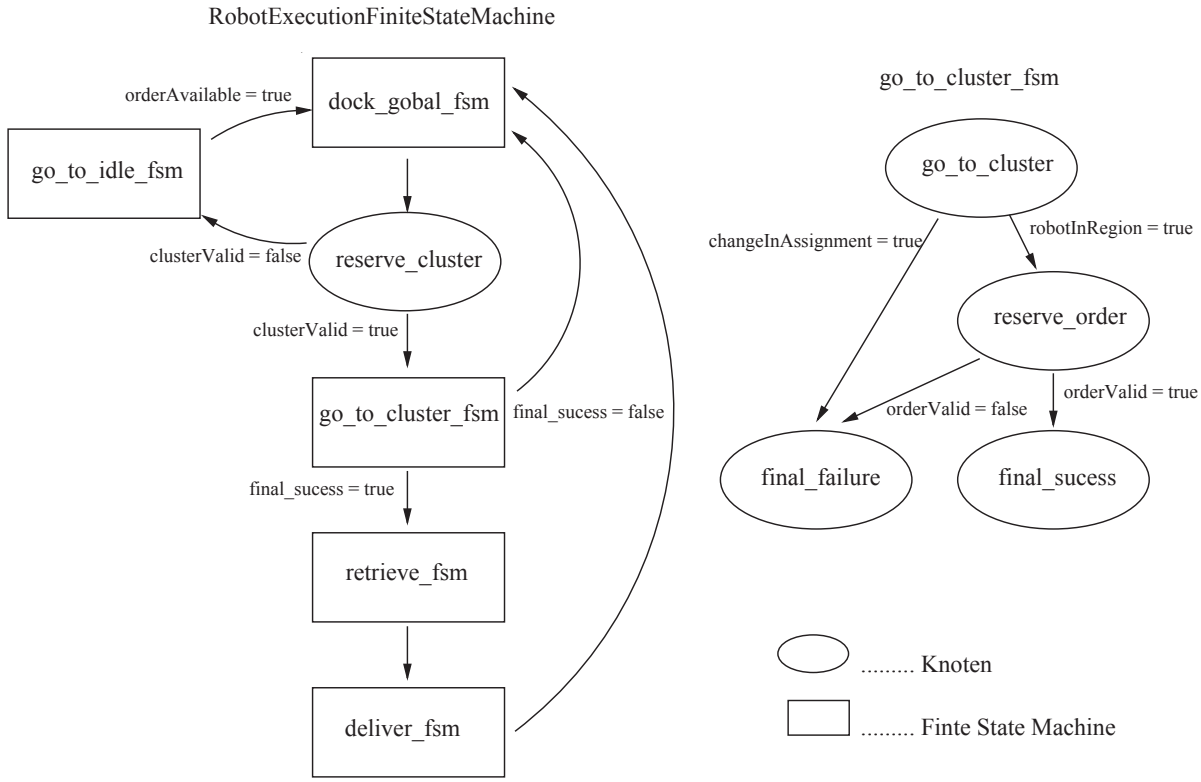

Abbildung 5: Darstellung der Robot-Execution-FSM und der Go-to-cluster-FSM

5 Auftragsverteilungsalgorithmen

Der Entwicklung eines Algorithmus für eine optimierte Auftragsverteilung an eine Flotte von Robotern gingen zunächst die Betrachtung der Literatur und die Suche nach bereits vorhandenen Lösungen ähnlicher Problemstellungen voraus. Daran schloss sich die Analyse des bestehenden Algorithmus.

5.1 Auftragsverteilung in Multi-Roboter-Systemen

Das Problem, wie in einem Multi-Robotersystem Aufträge optimal zu verteilen sind, ist seit den frühen 1990er Jahren Gegenstand von Untersuchungen. Es ist als Multi-Robot Task Allocation (MRTA) bekannt. Im Folgenden werden die Ansätze, die in der Literatur zu diesem Thema vorhanden sind, kurz vorgestellt.

5.1.1 Ansätze zur Roboterauftragsverteilung

In [11] wurden folgende Klassifikationsmöglichkeiten für die Auftragsverteilung an mehrere Roboter definiert:

- **Single Task Robots (ST)** versus **Multi-Task-Robots (MT)**: ST bedeutet, dass jeder Roboter in der Lage ist, höchstens einen Auftrag auf einmal auszuführen; MT hingegen, dass ein Roboter mehrere Aufträge simultan ausführen kann.
- **Single-Robot Tasks (SR)** versus **Multi-Robot Tasks (MR)**: SR bedeutet, dass jeder Auftrag genau einen Roboter für seinen Erfüllung benötigt; MR besagt, dass manche Aufträge auch mehrere Roboter für ihre Erfüllung benötigen können.
- **Instantaneous assignment (IA)** versus **time-extended assignment (TA)**: Mit IA ist gemeint, dass die über die Roboter, die Aufträge und die Umwelt verfügbaren Informationen nur eine augenblickliche Auftragszuweisung ohne Berücksichtigung zukünftiger Aufträge erlauben. TA bedeutet, dass mehr Informationen verfügbar sind, zum Beispiel eine Liste von allen Aufträgen, die zugewiesen werden müssen, oder ein Modell, das angibt, welche Aufträge über die Zeit zu erwarten sind.

Die Variante *ST-SR-IA* entspricht dem aus dem Operations Research bekannten Zuordnungsproblem. Bei diesem suchen n Arbeiter nach einem Auftrag, und m Aufträge

benötigen einen Arbeiter. Die Aufträge können unterschiedliche Priorität haben, und jeder Arbeiter hat einen nicht-negativen Qualifikations-Wert, einen Auftrag auszuführen. Je höher dieser Qualifikations-Wert ist, desto höher ist der Leistungsgrad, mit dem ein Auftrag erfüllt werden kann. Das Problem besteht darin, den Arbeitern die Aufträge so zuzuweisen, dass der erwartete gesamte Leistungsgrad maximal ist.

Zur Lösung dieses Problems kann ein Verfahren der linearen Programmierung verwendet werden, wie zum Beispiel die Ungarische Methode, die 1955 von Harold W. Kuhn entwickelt wurde. Diese findet die optimale Lösung zu diesem Problem in $\mathcal{O}(mn^2)$ Schritten. Damit gehört dieses Problem der Komplexitätsklasse P, der in polynomialer Zeit lösbaren Entscheidungsprobleme, an.

Nur wenige MRTA-Probleme weisen diese einmalige Zuweisungsstruktur auf, in vielen Fällen kann man Probleme jedoch als iterative Instanzen der Klasse *ST-SR-IA* ansehen. Eine Möglichkeit, dieses iterative Zuordnungsproblem zu lösen, ist die Methode Broadcast of local eligibility (BLE), bei welcher jeder Roboter seine Eignung für die momentan verfügbaren Aufträge berechnet und den anderen Robotern mitteilt [11]. Jeder Roboter kann sich so den Auftrag suchen, für den er am besten geeignet ist und diesen ausführen. Ähnlich vorgegangen wird in der in [18] beschriebenen Methode *ALLIANCE*. Hier wird die Motivation eines Roboters, einen bestimmten Auftrag auszuführen, aus zwei Kriterien bestimmt - Duldung und Ungeduld. Merkt ein Roboter, dass er einen Auftrag nicht gut ausführt, so sinkt seine Duldung für diesen Auftrag. Merkt dies ein anderer Roboter, so steigt dessen Ungeduld.

Stehen alle Aufträge nicht von Anfang an zur Verfügung und können Auftragszuweisungen nicht verändert werden, so spricht man von *Online Assignment*. Eine Möglichkeit, diese Art von Problemen zu lösen, ist der in [11] beschriebene *MURDOCH*-Zuweisungsalgorithmus. Bei diesem wird nach dem Prinzip einer Auktion dem meistbietenden Roboter ein Auftrag zugewiesen. Den "Preis", den ein Roboter für einen Auftrag bietet, besteht aus seiner von ihm berechneten Eignung für diesen Auftrag. Durch ein möglichst wirtschaftliches Handeln der einzelnen Roboter wird so versucht, ein Gesamtoptimum zu erreichen.

In den meisten wissenschaftlichen Untersuchungen wird angenommen, dass das System dezentral ist und die Roboter in einer Gruppe selbst entscheiden müssen, welche Aufträge sie durchführen. In diesen Untersuchungen wird auch berücksichtigt, dass die Kommunikations-Reichweite der Roboter beschränkt ist, und es wird versucht, die Menge der zu übertragenden Daten zu minimieren. Unter diesen Aspekt fällt auch die Forschung auf dem Gebiet der *swarm robotics*, welches von [17] ausführlich untersucht wurde. Hier

wird versucht, mit einer großen Anzahl von relativ simplen und kostengünstigen Robotern komplexe Aufgaben zu erfüllen. Inspiriert ist diese Forschung durch das Verhalten von Insekten und anderen Tieren, die in Gemeinschaften leben. Diese sind in der Gruppe, trotz stark limitierter Kommunikation, zu beeindruckenden Leistungen fähig.

Der Auftragsverteilungsalgorithmus, der derzeit im Einsatz ist, wurde ursprünglich auch für eine dezentrale Steuerung entwickelt. Er wird im Folgenden beschrieben.

5.1.2 Distributed Hash-Tables

Der zur Zeit verwendete Algorithmus stammt von Alexander Kleiner und beruht auf dem Konzept der *Distributed Heterogeneous Hash Tables* (DHHT). Dieser Ansatz stammt aus dem Bereich der Paketverteilung an Server, bei der eine gleichmäßige Auslastung der Server entsprechend ihrer Leistungsfähigkeit angestrebt wird [19]. Die Auftragsverteilung an Roboter nach dem DHHT-Konzept wird in [20] folgendermaßen beschrieben: Es wird versucht, den unbeschäftigten Robotern Aufträge so zuzuweisen, dass die Stationen, an denen die Aufträge abgeholt werden, möglichst gleichmäßig bedient werden. Hierbei wird neben der Anzahl an erledigten und noch zu erledigenden Aufträgen auch die Besuchshäufigkeit eines Roboters an dieser Station und die Distanz zwischen Roboter und Station berücksichtigt. Es wird für jede Station die gewichtete Distanz D_i nach Gleichung 4 berechnet und der Auftrag mit dem geringsten Wert ausgewählt. Dabei bezeichnet d_i die tatsächliche Distanz zwischen Roboter und Station und w_i den Gewichtungsfaktor, der nach Gleichung 5 berechnet wird. In dieser Gleichung bezeichnet $N_Q - N_C$ die Differenz zwischen den Aufträgen dieser Station, die noch zu erledigen sind, und jenen, welche bereits einem Roboter zugewiesen sind. Des weiteren beschreibt $N_{delivered}$ die Anzahl an Behälter, die insgesamt von dieser Station abgeholt wurden. Die Effizienz dieser Station wird mit e_i bezeichnet. Diese wird nach Gleichung 6 aus dem Durchsatz Tr, mit welcher Aufträge momentan von dieser Station abgeholt werden, und Tr_{max} berechnet. Der maximale Durchsatz Tr_{max} ist wiederum durch das Minimum zwischen der Rate, mit der Behälter zur Abholung an dieser Station bereitgestellt werden, und einer Konstante k gegeben, wobei $\frac{1}{k}$ die Zeit beschreibt, die ein Roboter für das Andocken an die Station, die Behälterübergabe und das Verlassen der Station benötigt.

$$D_i = \frac{log(d_i)}{w_i} \tag{4}$$

$$w_i = \frac{1}{e_i} \frac{N_Q - N_C}{N_{delivered}} \tag{5}$$

$$e_i = \frac{Tr}{Tr_{max}} \tag{6}$$

Durch die Gewichtung bei dieser Zuweisung wird sichergestellt, dass Stationen, die selten von Robotern besucht wurden, eher einen Roboter zugewiesen bekommen. So kann verhindert werden, dass Stationen, in deren Nähe sich keine Roboter befinden, durch eine Auswahl, die auf der kürzesten Distanz beruht, nie besucht werden.

5.2 Untersuchung des zu behandelnden Problems

Wie in Abschnitt 5.1 bereits erwähnt, entspricht ein Problem der Form *ST-SR-IA* dem Zuordnungsproblem. Bei diesem wird jedoch jedem Arbeiter nur ein Auftrag zugewiesen, und für eine möglichst gute Auftragsverteilung muss jede zu diesem Zeitpunkt bekannte Information berücksichtigt werden. Da jedoch im vorliegenden Problem die Anzahl der bekannten Aufträge meist größer als die Anzahl an Robotern ist, sollte deshalb versucht werden, dem einzelnen Roboter mehr als nur einen Auftrag zuzuweisen. Das Erstellen eines Ablaufplanes, welcher Prozessoren zeitlich begrenzte Ressourcen zuteilt, ist unter dem Namen *Scheduling* bekannt und wird im folgenden Teil beschrieben.

5.2.1 Das *Scheduling Problem*

Das Scheduling Problem stammt aus der produzierenden Industrie und beschäftigt sich mit der Frage, in welcher Reihenfolge Aufträge von Maschinen bearbeitet werden sollen, um die Bearbeitung in möglichst kurzer Zeit durchzuführen. Dabei wird in [8] zwischen Flowshop-Scheduling und Jobshop-Scheduling unterschieden. Während bei Jobshop-Scheduling die Reihenfolge, in der ein Auftrag die Maschinen durchlaufen muss, bei jedem Auftrag unterschiedlich sein kann, ist diese Reihenfolge bei Flowshop-Scheduling bei allen Aufträgen gleich. Die Zeit, die ein Bearbeitungsschritt dauert, hängt jedoch bei beiden Varianten vom Auftrag ab. Das Flowshop-Scheduling-Problem gehört für eine Anzahl $b \geq 3$ an Bearbeitern der Komplexitätsklasse *NP-complete* an [10]. Dies bedeutet, dass es zu der am schwersten lösbaren Sorte von Entscheidungsproblemen zählt, welche von einem nichtdeterministischen Verfahren in polynomialer Zeit (NP) gelöst werden können. Gäbe es einen Algorithmus für die effiziente Lösung dieses Problems, so könnte dieser auch für das Lösen aller Probleme der Komplexitätsklasse NP eingesetzt werden.

Das Jobshop-Scheduling-Problem gehört bereits ab zwei Bearbeitern der Komplexitätsklasse *NP-complete* an [3]. In [12] wird dieses Problem folgendermaßen formuliert:

Gegeben sei eine Menge an $n \geq 1$ Aufträgen (Jobs), $J = \{T_1, ..., T_n\}$ und eine Menge $m \geq 1$ an Bearbeitern (Processor) $P_1, ..., P_m$. Ist $m = 1$, so spricht man vom *single-machine scheduling problem*, ist $m > 1$, vom *parallel machine scheduling problem*. Die Durchführungszeit des Auftrags T_j durch den Bearbeiter P_i wird mit τ_{ij} bezeichnet. In einem Ablaufplan (Schedule) σ hat ein Auftrag J_i eine Startzeit $S_j(\sigma)$ und eine Fertigstellungszeit $C_j(\sigma)$. Der mögliche Zeitpunkt für die Ausführung eines Auftrags T_j kann durch einen Veröffentlichungstermin als eine untere Grenze für die Startzeit und durch eine Abgabefrist als obere Grenze für die Fertigstellungszeit eingeschränkt werden. Die mittlere Fertigstellungszeit $\bar{t}$ eines Ablaufplans σ ist gegeben durch $\bar{t}(\sigma) = \frac{1}{n} \sum_{j=1}^{n} C_j(\sigma)$ [3].

In [3] werden ebenfalls zwei Scheduling-Methoden beschrieben: Die Methode des *shortest-processing-time (SPT) scheduling* bestimmt einen Ablaufplan nach minimaler mittlerer Fertigstellungszeit, indem Aufträge mit kürzerer Durchführungszeit zuerst verteilt werden. Sie minimiert jedoch nicht die maximale Fertigstellungszeit. Ist dies gewünscht, ist die Methode des *largest-processing-time (LPT) scheduling* geeignet, bei welcher Aufträge mit größerer Durchführungszeit zuerst verteilt werden.

Um ein Problem als klassisches Jobshop-Scheduling-Problem behandeln zu können, muss jedoch die Annahme erfüllt sein, dass die Bearbeitungszeiten vom Ablaufplan unabhängig sein müssen [8]. Das bedeutet, dass die Vorbereitungszeit einer Maschine für den nächsten Auftrag unabhängig vom letzten Auftrag der Maschine ist.

Betrachtet man das vorliegende Problem als Scheduling-Problem, so kann man die Roboter als Maschinen, die Aufträge abhängig von ihrer Position in unterschiedlicher Zeit bewältigen können. Da beim vorliegenden Problem die Durchführungszeiten der Aufträge jedoch von der Position des Roboters zu Beginn der Auftragsdurchführung abhängen und diese sehrwohl vom letzten durchgeführten Auftrag bestimmt wird, kann dieses Problem nicht als klassisches Jobshop-Scheduling-Problem behandelt werden.

Die Bestimmung der Reihenfolge, in der eine gewisse Anzahl an Fahrzeugen räumlich verteilte Ziele besucht, ist als *Vehicle Routing Problem* bekannt und wird im Folgenden beschrieben.

5.2.2 Das *Vehicle Routing Problem*(VRP)

In [7] wird das VRP als folgendes Graphen-theoretische Problem formuliert: Sei $G = (V, A)$ ein vollständiger Graph mit einer Menge $V = \{0, 1, ..., n\}$ an Knoten und einer Menge A an Kanten. Die Knoten $j = 1, ..., n$ entsprechen Kunden, von welchen jeder einen nicht-negativen Bedarf d_j hat. Der Knoten mit dem Index 0 entspricht dem Zen-

trallager. Jede Kante verfügt über nicht-negative Kosten c_{ij}, welche den Aufwand für eine Reise von Knoten i zu Knoten j beschreiben. Die maximale Menge, die auf einer Rundreise ausgeliefert werden kann, wird als Kapazität C bezeichnet. Ziel des VRP ist es, eine Menge k an Rundreisen zu finden, sodass die Summe der Bedarfe der besuchten Knoten jeder Rundreise nicht die Kapazität C übertrifft und dass die Summe der Kosten aller Rundreisen minimal ist.

Endet jede Route nicht am Zentrallager sondern beim letzten besuchten Knoten, so spricht man vom *Open Vehicle Routing Problem*. Jede Route in einer Lösung dieses Problems entspricht einem Hamilton-Pfad, eine optimale Lösung des *OVRP*s besteht also aus einer Reihe von minimalen Hamilton-Pfaden. Da das *Minimum Hamilton Path Problem* der Komplexitätsklasse der *NP-schweren* Probleme angehört, ist das *OVRP* ebenfalls dieser Komplexitätsklasse zuzuordnen [6]. Diese beinhaltet jene Probleme, die mindestens so schwer zu lösen sind, wie die schwierigsten Probleme der Klasse *NP*.

In [2] wird außerdem zwischen statischen und dynamischen Routing-Problemen unterschieden. Bei einem statischen Problem sind alle Input-Daten der Problemstellung bekannt, bevor Routen berechnet werden. Bei einem dynamischen Routing-Problem werden manche Input-Daten erst während der Ausführungszeit bekannt.

Das Vehicle Routing Problem ist aufgrund seiner praktischen Relevanz seit den 1960er Jahren intensiv untersucht wordern. Es wurden viele exakte Methoden, Heuristiken und Metaheuristiken in der Literatur vorgestellt, es ist jedoch nur bei relativ kleinen Instanzen des Problems möglich, dieses optimal zu lösen. Eine häufig zum Einsatz kommende Variante, das VRP zu lösen, ist in zwei Stufen unterteilt: In einem ersten Schritt wird mit einer *konstruktiven Heuristik* eine Ausgangslösung erstellt, in einem zweiten Schritt wird diese durch eine *lokale Suche* schrittweise verbessert [22].

In [21] wird eine für das VRP geeignete Konstruktionsheuristik beschrieben: der *Savings*-Algorithmus von Clarke and Wright. Dabei handelt es sich um einen Greedy-Algorithmus, welche in Abschnitt 5.2.4 genauer beschrieben werden. Der Savings-Algorithmus versucht, zwei Routen $(0, ..., i, 0)$ und $(0, j, ..., 0)$ zu einer Route $(0, ..., i, j, ..., 0)$ zu verbinden. Dadurch wird die Distanz $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, welche als *saving* bezeichnet wird, eingespart. Es wird zuerst für jeden zu besuchenden Knoten eine eigene Route erstellt und danach für jedes mögliche Routen-Paar der *saving*-Wert berechnet. Anschließend werden die Routen, sofern durch die vorgegebenen Bedingungen möglich, nach abfallendem *saving*-Wert zu größeren Routen kombiniert.

Eine verbreitete Methode zur lokalen Suche, welche aus den Untersuchungen am *Traveling Salesman Problem* stammt, ist *λ-opt*. Bei diesem in [22] beschriebenen Verfahren

wird versucht, durch das Tauschen von λ Kanten eine bessere Rundreise zu erhalten. Eine Rundreise wird beispielsweise als 2-*opt* bezeichnet, wenn durch Tauschen von zwei Kanten keine bessere Lösung erzielt werden kann.

5.2.3 Das *Pickup and Delivery Problem (PDP)*

Eine Unterart des Vehicle Routing Problems ist das *Pickup and Delivery Problem*, bei welchem Objekte oder Personen von einem oder mehreren Fahrzeugen zwischen einer Quelle und einem Ziel transportiert werden sollen. Innerhalb des *PDP* werden nach [2] wiederum drei Gruppen unterschieden:

- ***Many-to-Many*-Probleme:** Bei dieser Form kann jeder Knoten als Quelle oder auch Ziel für jedes Gut verwendet werden.
- ***One-to-Many-to-One*-Probleme:** Solche Probleme treten auf, wenn Güter anfangs an einem Depot verfügbar sind und zu Kunden-Knoten transportiert werden. Außerdem werden auch Güter, die an den Kunden-Knoten verfügbar sind, zum Depot transportiert.
- ***One-to-One*-Probleme:** Diese Art des PDPs tritt auf, wenn jedes Transportgut eine bestimmte Quelle und ein bestimmtes Ziel besitzt.

Jene Variante des PDPs, bei der die Fahrzeuge immer nur einen Auftrag auf einmal durchführen, bezeichnet [5] als *Stacker Crane Problem (SCP)*: Dieser Name stammt aus dem Gebiet der Verwaltung von Kränen, die Aufgabenstellung findet inzwischen jedoch vor allem bei der Planung von Lastkraftwagen für den Containertransport Verwendung. Deshalb ist es auch unter dem Name *Full Truckload Pickup and Delivery Problem* bekannt. Sind für das Abholen und Abliefern der Güter Zeitfenster gegeben, so spricht man vom *Full Truckload Pickup and Delivery Problem with Time Windows (FTPDPTW)*. Bei einer Aufgabenstellung mit harten Zeitfenstern wird eine verspätete Ankunft der Fahrzeuge nicht erlaubt.

Um dieses Problem zu lösen wird in [5] eine zwei-Phasen Einsetzungsheuristik beschrieben, bei der versucht wird, durch Zusammenfügen von einzelnen Abholungs- und Ablieferungszielen eine Ausgangslösung zu konstruieren. In der ersten Phase werden die Abholungen und Ablieferungen zu Paaren kombiniert und in Routen eingefügt. Dabei wird versucht, unter Einhaltung der Zeitfenster, die von den Fahrzeugen zurückzulegende Distanz möglichst gering zu halten. In der zweiten Phase werden die erstellten

Ausgangslösungen durch eine Suche in den folgenden drei lokalen Nachbarschaften verbessert:

- **CROSS:** Hierbei werden zwei Paare von Abhol- und Abliefer-Knoten aus unterschiedlichen Rundreisen vertauscht. Dies ist nur zulässig, wenn nach dem Tausch noch alle Zeitfenster erfüllt sind. Es werden mehrere CROSS-Vertauschungen untersucht, jene mit der größten Ersparnis wird durchgeführt.
- **COMBINE:** Es wird versucht, zwei Rundreisen von unterschiedlichen Fahrzeugen zu einer einzigen zusammenzufügen. Während CROSS versucht, die Fahrkosten der Fahrzeuge in der Zielfunktion zu reduzieren, so versucht COMBINE, die Anzahl an nötigen Fahrzeugen zu verringern.
- **INSERT:** Nach einer Verbesserung wird gesucht, indem ein Paar von Abhol- und Abliefer-Knoten aus einer Rundreise entfernt und einer anderen Rundreise hinzugefügt wird. Dies kann sowohl zu einer Reduktion der gesamten Fahrkosten führen als auch die Eliminierung einer Route bewirken und somit in einer Reduktion der Anzahl an benötigten Fahrzeugen resultieren.

5.2.4 Greedy-Algorithmen

Greedy-Algorithmen sind dadurch gekennzeichnet, dass sie dem Gradientenverfahren entsprechend immer die aktuell oder lokal größte Verbesserung auswählen. Dafür müssen sie zuerst alle zur Verfügung stehenden nächsten Schritte bewerten. Ein Greedy-Algorithmus findet meist in geringer Zeit eine gute Lösung, eine optimale Lösung findet er jedoch nur, wenn die zulässigen Lösungen des Problems die unabhängigen Mengen eines Matroids sind [15].

Ein Matroid wird in [4] folgendermaßen definiert: Sei $\mathcal{I}$ eine nicht leere Familie von Mengen aus einer Grundmenge X. Das Paar $M = (X; \mathcal{I})$ ist ein Matroid, wenn folgende Bedingungen erfüllt sind:

- Die Leere Menge befindet sich in M.
- Die Elemente von $\mathcal{I}$ sind unabhängige Mengen von M. Jede Teilmenge einer zu $\mathcal{I}$ gehörenden Menge gehört ebenfalls zu $\mathcal{I}$.
- Wenn $Y, Z \in \mathcal{I}$ und $|Z| > |Y|$, dann existiert ein $z \in Z$ sodass $Y \cup \{z\} \in \mathcal{I}$

Die ersten beiden Bedingungen stellen sicher, dass es sich um ein Unabhängigkeitssystem handelt, somit ist ein Matroid eine Sonderform eines Unabhängigkeitssystems. Laut [14] können viele kombinatorische Optimierungsprobleme als Unabhängigkeitssystem oder Matroid formuliert werden, da diese Ideen auf mehrere konkretere Strukturen, wie zum Beispielen einen Graph, umgelegt werden können. Ein Graph $G = (K, E)$ besteht aus einer Menge von Knoten K und von Kanten E. Die Menge E besteht aus einer Menge an Paaren von Elementen aus K. Es ist klar ersichtlich, dass ein Graph ein Unabhängigkeitssystem darstellt, da jede Teilmenge einer zum Graphen gehörenden Menge von Kanten ebenfalls zum Graphen gehört. Wenn wir für einen ungerichteten Graphen G die Menge $\mathcal{I}$ als Menge der Teilmengen von E definieren, die keine Kreise enthalten, dann ist $M(G) = (E, \mathcal{I})$ ein Matroid [14].

Inklusionsmaximale unabhängige Mengen eines Unabhängigkeitssystem heißen Basen, in einem Graphen entspricht eine Basis einem Spannbaum. Versucht man nun, ein Unabhängigkeitssystem zu optimieren, so wird nach einem maximalen oder minimalen Spannbaum gesucht.

Abschließend wird eine einfache Form des Greedy-Algorithmus aus [14] angegeben: Es sei ein Unabhängigkeitssystem $(E, \mathcal{F})$ und Kosten $c : E \to \mathbb{R}_+$ gegeben. Außerdem wird davon ausgegangen, dass es ein Unabhängigkeits-Orakel gibt, dass für eine Menge $F \subseteq E$ entscheidet, ob $F \in \mathcal{F}$ oder nicht. Folgender Greedy-Algorithmus löst das Maximierungsproblem für $(E, \mathcal{F}, c)$:

Algorithmus 1 Greedy-Algorithmus

Sortiere $E = \{e_1, e_2, ..., e_n\}$, so dass $c(e_1) \geq c(e_2) \geq \cdots \geq c(e_n)$
Setze $F := \emptyset$
for $i := 1$ **to** n **do**
    if $F \cup \{e_i\} \in \mathcal{F}$ **then**
        $F := F \cup \{e_i\}$
    end if
end for

6 Ermittlung eines Algorithmus

Für die weitere Behandlung der gegebenen Aufgabenstellung ist es nun erforderlich, anhand der in der Fachliteratur gesammelten Informationen einen adäquaten Algorithmus zu entwickeln. Dazu müssen zunächst die für diese Aufgabenstellung erforderlichen Voraussetzungen betrachtet werden.

6.1 Projektspezifische Bedingungen

In Abschnitt 2.5 wurden der Aufbau des geplanten Lebensmittel-Distributionszentrums und die darin vorgesehenen Abläufe beschrieben, an denen der neue Auftragszuweisung-Algorithmus gemessen werden soll. Folgende Eigenschaften des Zentrums sind für den Entwurf eines Algorithmus relevant:

- die maximale Anzahl an Robotern im Einsatz
- die Anzahl an Aufträgen, die im Moment der Verteilung bekannt sind
- die voraussichtliche Dauer für die zurückzulegenden Strecken
- die voraussichtliche Dauer für Auf- und Abgabe von Ladungsträgern

Die Zahl der im beschriebenen Projekt Lebensmitteldistributionszentrum einzusetzenden Roboter beträgt fünfzehn bis siebzehn Roboter. Wie viele davon zu einem bestimmten Zeitpunkt in Betrieb sind, wird jedoch der momentanen Auslastung des Zentrums angepasst werden. Die Transportaufträge, die das Warehouse-Management-System generiert, stehen dem Algorithmus ca. 10 min im Voraus für die Verteilung zur Verfügung. Bei einem durchschnittlichen Durchsatz von 420 Paletten pro Stunde sind also zu jedem Moment ungefähr 70 Aufträge bekannt.

Neben den Charakteristiken des Lagers und des Warehouse-Management-Systems, ist für eine erfolgreiche Auftragsverteilung der jeweils aktuelle Zustand des Transportsystems von Bedeutung. Dieser wird durch folgende Punkte bestimmt:

- die momentane Position aller Roboter
- den momentanen Ladestand aller Roboter
- die Zeitfenster aller Aufträge

Der momentane Ladestand wird berücksichtigt, weil zu vermeiden ist, dass ein Roboter aufgrund zu geringen Ladestands liegenbleibt. Ein Auftrag, bei dem eine große Distanz zurückzulegen ist, soll nicht einem Roboter zugewiesen werden, der einen niedrigen Ladestand hat und sich bald zur Ladestation begeben muss; auch muss der Roboter nach Abschluss seines letzten Auftrags ja noch zur Ladestation fahren. Wenn also eingeplant wird, dass ein solcher Auftrag den Roboter in die Nähe einer freien Ladestation führt, können die Fahrwege der Shuttles reduziert werden. Paletten-Shuttles können über ein Gesamtgewicht von bis zu zwei Tonnen verfügen, hier ist das Verhindern des Liegenbleiben eines solchen Shuttles umso wichtiger, da es nur von einem Gabelstapler wieder fortzubewegen ist.

Die Aufträge, die ein Roboter im Laufe eines Tages auszuführen hat, sind meist ähnlich und wiederholen sich häufig. Das Verhalten des Roboters ist zwar nicht deterministisch, in ähnlichen Situationen verhält er sich jedoch meist auf die selbe Weise und braucht daher auch ungefähr die selbe Zeit, um ähnliche Aufträge durchzuführen. Dies ermöglicht es, durch eine Aufzeichnung der für eine Strecke benötigten Zeit schnell Durchschnittswerte zu erhalten und so zu einer ziemlich genauen Abschätzung der Durchführungsdauer eines Auftrags zu gelangen.

Die Zeit, die ein Roboter tatsächlich benötigt kann jedoch durch unvorhergesehene Hindernisse oder das Kreuzen eines anderen Roboters verändert werden. Da das Ausweichverhalten je nach Situation unterschiedlich und nicht deterministisch ist, kann es in ungünstigen Fällen sogar dazu führen, dass sich zwei Roboter gegenseitig blockieren und so viel länger brauchen, um den ihnen zugewiesenen Auftrag auszuführen.

6.2 Ableiten einer adäquaten Auftragsverteilung

In den folgenden Ausführungen soll dargelegt werden, wie – ausgehend von den in der Fachliteratur dargestellten Erkenntnissen und unter Einbeziehung der im vorhandenen Umfeld herrschenden Bedingungen – ein Algorithmus zur effizienten Verteilung von Transportaufträgen an eine Flotte von Robotern entwickelt werden kann. Dieser soll die Aufträge so effizient vergeben, dass die für ihre Durchführung benötigte Zeit minimiert wird. Gelingt dies, so kann bei einem geforderten hohen Durchsatz besser auf Spitzenauslastungen reagiert werden; ist jedoch kein so hoher Durchsatz erforderlich, kann die Anzahl an Robotern, die für ein Projekt eingesetzt werden müssen, reduziert werden.

Um diese Beschleunigung der Durchführung zu erzielen, gilt es, eine Auftragsverteilung zu entwickeln, die den Robotern die Aufträge auf Basis der Zeit, die sie voraussichtlich

benötigen werden, zuordnet. Eine Schätzung der Zeiten, die für die einzelnen Schritte der Auftragserfüllung benötigt werden, ermöglicht es, für jeden einzelnen Roboter einen Zeitplan zu erstellen. Anhand der Zeitpläne aller Roboter können dann die Aufträge so an die Roboter verteilt werden, dass die Zeitfenster aller Aufträge berücksichtigt werden und dadurch die Zeit, die für die Erfüllung aller Aufträge benötigt wird, möglichst gering ist.

Diese Auftragsverteilung muss jedoch aus zwei Gründen regelmäßig neu ermittelt werden: Einerseits werden in kurzen Abständen neue Aufträge erteilt, und diese müssen sofort in der aktuellen Verteilung berücksichtigt werden. Andererseits kann sich die Situation, für die sich eine bestimmte Auftragsverteilung gut geeignet hat, schnell ändern. Dies liegt vor allem am nicht deterministischen Verhalten der Roboter, bei einer Verzögerung eines Roboters muss schnell auf diese Änderungen der Gesamtsituation reagiert werden. Die Tatsache, dass jeder Roboter für jeden Auftrag verwendet werden kann und dass die Quellen und Ziele vieler Aufträge nahe beieinander liegen, bewirken, dass durch eine kleine Änderung der Gesamtsituation bereits eine andere Auftragsverteilung besser wäre. Würde zu einem bestimmten Zeitpunkt durch eine langwierige Berechnung eine nahezu optimale Verteilung berechnet werden, so könnte diese zu einem späteren Zeitpunkt – aufgrund der sich ändernden Situation – keine geeignete Verteilung mehr darstellen. Aus diesen Gründen ist ein Algorithmus zu bevorzugen, der nur eine geringe Zeit zur Berechnung der Auftragsverteilung benötigt und so in kurzen Zeitabständen ausgeführt werden kann.

6.2.1 Erster Algorithmus

Auf Basis der Erkenntnisse, die aus den zuvor beschriebenen Untersuchungen gewonnen wurden, wurde ein erster Auftragsverteilungsalgorithmus entwickelt. Er wurde mit dem Ziel erstellt, dass er alle zum Zeitpunkt der Berechnung bekannten Aufträge berücksichtigt und diese auf die zur Verfügung stehenden Roboter verteilt. Dabei werden mithilfe von Erfahrungswerten Zeitpläne erstellt, mit dem Bemühen, die Einhaltung der Zeitfenster, in denen die Aufträge durchgeführt werden müssen, zu gewährleisten. Zur Berechnung der passenden Verteilung wurde ein zweistufiges Verfahren implementiert, welches im Folgenden kurz beschrieben wird.

Im ersten Schritt dieses Verfahrens wird eine Ausgangslösung erstellt, und diese wird in einem zweiten Schritt verbessert. Die Ausgangslösung wird erstellt, indem der Reihe nach für jeden Auftrag der am besten geeignete Roboter gesucht wird. Dazu wird der Auftrag probeweise allen Robotern erteilt, und danach wird für jeden Roboter für jede

mögliche Reihenfolge die Zeit ermittelt, in der er die ihm zugewiesenen Aufträge ausführen kann. Der Auftrag wird dann so zugewiesen, dass der Roboter, der am längsten für die Durchführung seiner Aufträge benötigt, am frühesten fertig ist. Dieses Vorgehen ist jedoch nur möglich, solange die Anzahl an bekannten Aufträgen pro Roboter wie angenommen maximal fünf beträgt. Wäre eine größere Anzahl an Aufträgen bekannt, so würde die Evaluierung jeder mögliche Reihenfolge zu viel Rechenzeit in Anspruch nehmen.

Sind erst einmal alle Aufträge verteilt, so wird noch versucht, diese Ausgangslösung weiter zu verbessern. Dabei werden die in Abschnitt 5.2.3 beschriebenen Schritte *insert* und *cross* eingesetzt. Bei *insert* wird einem Roboter, der überdurchschnittlich lange für die Durchführung seiner Aufträge benötigt, ein Auftrag entfernt. Danach wird versucht, diesen Auftrag einem anderen Roboter zuzuweisen. Wird eine Paar gefunden, bei dem nach dem Tausch der spätest abgeschlossene Auftrag früher beendet wird, so wird die getauschte Verteilung als neue Auftragsverteilung übernommen. Bei *cross* wird ähnlich vorgegangen, nur dass versucht wird, durch das Vertauschen von zwei Aufträgen unterschiedlicher Roboter eine Verbesserung zu erzielen.

In Simulationsläufen mit dieser neuen Auftragsverteilung zeigt sich jedoch, dass die Auftragsverteilung im Laufe der Durchführung häufig wechselt. Bedingt ist dies durch eine Abweichung von den berechneten Zeitplänen, und ab einem gewissen Ausmaß der Abweichung erweist sich dann meist eine andere Auftragsverteilung als günstiger. Diese Abweichung entsteht durch das nicht vorhersagbare Aufeinandertreffen und Ausweichverhalten der Roboter bei der Fahrt zwischen zwei Stationen. Da die Schätzungen der benötigten Zeiten auf Durchschnittswerten beruhen, können diese Abweichungen auch nicht eliminiert werden.

Werden für einen Roboter bei der Berechnung der Verteilung mehrere Aufträge zeitlich geplant, so wächst mit jedem weiteren durchgeführten Auftrag die Abweichung vom berechneten Plan. Der letzte Auftrag einer Verteilung wird dann mit hoher Wahrscheinlichkeit nicht zu jener Zeit durchgeführt, die bei der Berechnung für ihn vorgesehen wurde.

Das ständige Hinzukommen von neuen Aufträgen und das Abweichen von der geplanten Zeit führen dazu, dass sich bei einer Neuberechnung die Auftragsverteilung für einen Roboter häufig ändert. Da jedoch der Algorithmus ein Gesamtoptimum für alle Aufträge, die dem Roboter zugewiesen wurden, sucht, kann es sein, dass der erste Auftrag, den er durchzuführen hat, nicht der günstigste ist, der zur Verfügung steht.

Bei dieser Form der Auftragsverteilung kommt es also vor, dass der erste zugeteilte

Auftrag nicht der günstigste ist, und dass die restlichen zugewiesenen Aufträge nach der Durchführung des ersten Auftrags geändert wurden. Dies hat zur Folge, dass von der Auftragsverteilung längere Strecken und Wartezeiten in Kauf genommen wurden, ohne danach von einer besseren Situation profitieren zu können. Aus diesem Grund ist es sinnvoll, die Zahl der Aufträge, die einem Roboter auf einmal zugewiesen werden, soweit zu reduzieren, dass die Zeit, die er für ihre Durchführung benötigt, mit ausreichender Genauigkeit abgeschätzt werden kann. Somit kann besser sichergestellt werden, dass jeder Roboter den für ihn momentan günstigsten Auftrag zugewiesen bekommt und somit unnötige Fahr- und Wartezeiten vermieden werden können.

6.2.2 Neuer Algorithmus

Angesichts der eben aufgezeigten Gründe ist es sinnvoll, den Robotern nicht alle verfügbaren Aufträge auf einmal zuzuteilen. In einem neuen Versuch wird daher ein Algorithmus entwickelt, welcher jedem Roboter zunächst nur einen einzigen Auftrag zuweist. Erst wenn der Roboter einen Auftrag, der nicht mehr verändert werden kann, bereits ausführt, weist ihm der Algorithmus einen weiteren Auftrag zu. Dieser kann jedoch, im zweistufigen Verfahren, bei einer Neuberechnung noch so lange verändert werden, bis der Roboter die Get-Station des neuen Auftrags zugewiesen bekommen hat.

Die Anzahl an möglichen Kombinationen ist allerdings bei Vergabe eines einzigen Auftrags pro Roboter weitaus geringer als bei Verteilung aller Aufträge. Deshalb kann bei diesem neuen Algorithmus jede mögliche Auftrag-Roboter-Kombination untersucht werden. Danach soll mithilfe einer *Greedy*-Strategie (siehe Abschnitt 5.2.4) immer die beste Kombination aus Roboter und Auftrag ausgewählt und zugewiesen werden.

Da das Ziel des Algorithmus – den Durchsatz des Transportsystems zu maximieren –vornehmlich durch Minimierung der Zeit erreicht werden kann, die für die Durchführung des Auftrags nicht unbedingt nötig ist, gilt es also, bei der Auftragsdurchführung eines Roboters folgende “Leerzeiten“ zu vermeiden:

- **Anfahrtszeit zur Get-Station**: Die Zeit, die für das Zurücklegen der Strecke zwischen Get- und Put-Station benötigt wird, ist konstant und kann durch die Wahl des Roboters nicht beeinflusst werden. Es ist jedoch möglich, durch die Wahl des Roboters, welcher sich am nächsten zur Get-Station eines Auftrags befindet, der Durchsatz des Systems zu erhöhen.

- **Wartezeit an einer Station:** Ist die Get- oder Put-Station bereits durch einen anderen Roboter reserviert bzw. besetzt, so fährt der Roboter an eine Wartepo-

sition und verweilt dort, bis die Station wieder freigegeben wurde. Durch eine gleichmäßige Verteilung der Aufträge auf alle Stationen kann die durchschnittliche Wartezeit der Roboter reduziert und der Durchsatz somit erhöht werden.

Der Algorithmus berechnet für jede Auftrag-Roboter-Kombination die Leerzeit und übernimmt dann jene mit der geringsten Leerzeit in die neue Verteilung. Danach wird der Roboter und der Auftrag dieser Kombination aus den Listen gestrichen, und alle übrigen Kombinationen werden erneut berechnet. Dies wird so lange fortgeführt, bis entweder keine Aufträge oder keine Roboter mehr zur Verfügung stehen.

Um die Leerzeit einer Auftragsverteilung bestimmen und minimieren zu können, ist es zuvor erforderlich, die Zeit, die von einem Roboter für einen bestimmten Auftrag benötigt wird, abzuschätzen. Dafür müssen erst Erfahrungswerte für ein gewisses Layout gesammelt werden, mit deren Hilfe dann eine Aussage über die Fahrzeit zwischen und die Übergabezeit an den Stationen getroffen werden kann.

Kapitel 7 beschreibt, wie einerseits die Zeitmessung und die Verwaltung der Erfahrungswerte und andererseits die zeitliche Evaluierung einer Auftragsverteilung umgesetzt wurde; danach geht Kapitel 8 auf die Implementierung des eben beschriebenen Algorithmus und der dafür notwendigen unterstützenden Klassen ein.

7 Implementierung der zeitlichen Erfassung und Schätzung der Auftragsdurchführung

Je mehr Roboter bei einer bestimmten Aufgabenstellung für die Auftragserfüllung eingesetzt werden, umso stärker können sie sich behindern. Dabei kommt es zu zwei Formen der Verzögerung: Zum Einen ergeben sich längere Fahrzeiten zwischen den Stationen, da die Roboter – infolge ihres Ausweichsverhaltens – mehr oder weniger große Umwege fahren müssen, je nach Winkel und Geschwindigkeit bei ihrem Aufeinandertreffen. Zum Anderen kommt es zu Wartezeiten an Stationen, wenn diese bereits durch einen anderen Roboter belegt sind.

Eine gute Auftragsverteilung kann zwar nicht die erste, sehr wohl aber die zweite Form der Verzögerung minimieren. Dazu müssen die Aufträge auf eine Weise verteilt werden, dass Situationen, bei denen eine Station von mehr als einem Roboter beansprucht wird, weitgehend vermieden und so die Wartezeiten reduziert werden. Der Weg zur Erreichung dieses Ziels führt über eine möglichst große Annäherung der geschätzten Zeiten, die ein Roboter für die Durchführung eines Auftrags benötigt, an die tatsächlich benötigten.

Da die benötigten Zeiten jedoch auch von der Qualität der Auftragsverteilung abhängig sind, setzt eine genaue Abschätzung eine große Menge an Zeitmessungen voraus. Eine gute Auftragsverteilung ist also erst erreichbar, wenn die Qualität der Verteilung durch ständige Zeitmessungen gesteigert wird und deren Ergebnisse zur laufenden Verbesserung herangezogen werden.

Für die neue Form der Auftragsverteilung, die den Gegenstand dieser Untersuchung bildet, ist also zunächst die Zeit, die ein Roboter für die Durchführung der Aufträge benötigt, zu schätzen. Dafür wurde eine Zeitmessung eingerichtet, welche für jeden Roboter die für die Ausführung eines Auftrags benötigte Zeit aufzeichnet. Diese ermöglicht, dank der großen Menge an gesammelten Werten, eine genaue Schätzung des für die Durchführung eines Auftrags benötigten Zeitbedarfs. Darauf aufbauend wurde eine Funktion entwickelt, welche für eine bestimmte Abfolge an Aufträgen, die ein Roboter durchzuführen hat, die benötigte Zeit berechnet. Wie diese beiden Funktionalitäten umgesetzt wurden, wird in den folgenden Abschnitten beschrieben.

7.1 Implementierung der Zeitmessung

Die Zeitmessung soll für jede Strecke, die im Laufe einer Auftragsdurchführung zurückgelegt werden kann, und für jede Übergabe an einer Station, die dafür von den Robotern

benötigte Zeit messen und aufzeichnen. Anhand dieser Daten sollen dann Durchschnittswerte ermittelt werden, auf deren Basis die zukünftige Auftragsverteilung geplant werden kann. Außerdem soll diese Zeitmessung im laufenden Betrieb fortgesetzt werden, sodass die tatsächlich benötigten Zeiten immer genauer bewertet werden können.

In der aktuellen Version des Systems ist bereits eine Zeitmessung vorhanden. Diese misst jedoch nur die Zeit, die zwischen der Reservierung eines Auftrags und dessen Abschluss vergeht. Sie verwendet die Knoten *setStartTime* und *orderFinished* in der Finite State Machine. *setStartTime* wird zu Beginn eines Auftrags durchlaufen und der aktuelle Zeitpunkt wird in einer Variablen gespeichert. Ist der Auftrag beendet, wird der Knoten *orderFinished* erreicht und die Differenz zwischen der Startzeit und der aktuellen Zeit berechnet. Diese wird dann, zusammen mit allgemeiner Information zum abgeschlossenen Auftrag, auf dem topic */finished_orders* veröffentlicht. Der *OrderFinishedListener* im *OrderManager* ist auf dieses topic abonniert und erstellt bei einer neuen Nachricht ein Objekt vom Typ *OrderHistoryEntry*, welches alle Informationen über den abgeschlossenen Auftrag enthält. Dieses Objekt wird an den *OrderHistoryManager* übergeben, welcher für die Speicherung und statistische Auswertung der Informationen über durchgeführte Aufträge verantwortlich ist. Die Sammlung von *OrderHistoryEntry*s wird regelmäßig in eine CSV-Datei geschrieben, anhand welcher auf einfache Weise statistische Auswertungen über die Leistung der einzelnen Roboter erstellt werden können.

In unserem Projekt soll nun eine Zeitmessung implementiert werden, die mehr als nur die Startzeit und die Endzeit der Durchführung eines Auftrags aufzeichnet. Eine detailliertere Erfassung soll alle Fahrzeiten und auch die Zeiten, die der Roboter an den Übergabestationen verbringt, aufzeichnen. Dazu müssen zusätzliche Knoten in der *Finite State Machine* erstellt werden, welche den Zeitpunkt, zu dem ein Teilabschnitt der Ausführung eines Auftrags abgeschlossen ist und mit dem nächsten begonnen wird, speichern.

Die gesamte Auftragsdurchführung ist dabei in Abschnitte zu unterteilen, für welche Start- und Endzeit aufgezeichnet werden, um die Dauer der Durchführung berechnen zu können. Zu Beginn dieser Abschnitte wird auch die Statusnachricht des ausführenden Roboters gesetzt, welche in der graphischen Oberfläche angezeigt wird, die zur Steuerung und Überwachung der Roboter dient und eine schnelle Identifizierung der momentanen Tätigkeit des Roboters erlaubt. In folgende Abschnitte wurde ein Transportauftrag unterteilt:

Tabelle 1: Abschnitte der Auftragsdurchführung

Status	**Beschreibung**
Go_To_Cluster	Abschnitt zwischen der Reservierung eines *clusterOrder* und dem Erreichen der Cluster-Region
Go_To_Get	Abschnitt zwischen dem Erreichen der Cluster-Region und dem Erreichen der Get-Station
Get_Queue	Abschnitt zwischen dem Erreichen der Get-Station und dem Freiwerde der Get-Station
Get	Abschnitt zwischen dem Freiwerden und dem Verlassen der Get-Station
Go_To_Put	Abschnitt zwischen dem Verlassen der Get-Station und dem Erreichen der Put-Station
Put_Queue	Abschnitt zwischen dem Erreichen der Put-Station und dem Freiwerden der Put-Station
Put	Abschnitt zwischen dem Freiwerden und dem Verlassen der Put-Station

Bei dieser Unterteilung wird die Zeit, die ein Roboter zum Aufnehmen und Abgeben einer Palette an einer Übergabestation (Put oder Get) benötigt, vom Beginn der Fein-Positionierung bis zum Verlassen der Station gemessen. Sie wird für jede Station separat gemessen, da die Dauer der Positionierung durch verschiedene räumliche Gegebenheiten unterschiedlich sein kann.

Die Zeit, die sich ein Roboter in den Status *Get_Queue* und *Put_Queue* befindet, ist ausschließlich davon abhängig, wie vielen Robotern von der momentanen Auftragsverteilung dieselbe Station zugeteilt wurde. Aus diesem Grund wird für das vorausschauende Planen einer Auftragsverteilung kein Erfahrungswert einer Wartezeit verwendet, sondern sie wird in Abhängigkeit von der Auftragsverteilung bestimmt (siehe Abschnitt 7.3).

Um den Zeitpunkt zu speichern, an dem ein Abschnitt abgeschlossen ist und der nächste beginnt, muss die Finite State Machine um die Knoten *arrived_at_cluster*, *arrived_at_get*, *get_finished* und *arrived_at_put* erweitert werden. Außerdem ist das Objekt *OrderHistoryEntry* ebenfalls um diese neuen Zeitpunkte zu ergänzen. Zusätzlich wird ihm das Feld *orderSequence* hinzugefügt, welches die Reihenfolge, mit der ein Roboter seine Aufträge fertiggestellt hat, angibt. Wird zum Abschluss eines Auftrags der be-

stehende Knoten *order_finished* erreicht, wird ein Eintrag mit der genauen Zeitnehmung erstellt. Anhand aller Einträge des *OrderHistoryManagers* können die durschnittlichen Fahr- und Übergabezeiten berechnet werden.

7.2 Implementierung der Durchschnittszeitberechnung

Für die Berechnung der Zeiten, die aufgrund der aufgezeichneten Daten für die Durchführung der einzelnen Schritte eines Auftrags voraussichtlich benötigt werden, werden folgende vier Methoden in der Klasse *OrderHistory* implementiert:

Tabelle 2: Methoden der Durchschnittszeitberechnung

Methode	**Funktion**
calculateMeanGetTimes()	Berechnet die durchschnittliche Zeit für die Feinpositionierung und die Behälterübergabe für alle Get-Stationen.
calculateMeanGetToPutTimes()	Berechnet die durchschnittliche Zeit für die Fahrt von der Get- zur Put-Station für jede in den Daten vorhandene Get-Put-Kombination.
calculateMeanPutTimes()	Berechnet die durchschnittliche Zeit für die Feinpositionierung und die Behälteraufnahme für alle Put-Stationen.
calculateMeanPutToGetTimes()	Berechnet die durchschnittliche Zeit für die Fahrt von der Put-Station des letzten Auftrags zur Get-Station des nächsten Auftrags für jede in den Daten vorhandene Put-Get-Kombination. Da es für diese Auswertung notwendig ist, die Zeiten von zwei aufeinanderfolgenden Aufträgen zu betrachten, muss für jeden Roboter die Reihenfolge bekannt sein, mit welcher er die in der OrderHistory gespeicherten Aufträge durchgeführt hat. Dafür kann das neu eingeführte Feld *orderSequence* der Klasse *OrderHistoryEntry* verwendet werden.

Für die Fahrt von der Put-Station des letzten Auftrags zur Get-Station des nächsten Auftrags wird also nur ein einziger Wert berechnet, obwohl sie mit *Go_To_Cluster* und

Go_To_Get aus zwei Teilabschnitten besteht. Eine getrennte Berechnung erweist sich jedoch aus zwei Gründen als nicht sinnvoll:

- Die Zeit, die ein Roboter ab dem Eintritt in die Cluster-Region für die Fahrt zur reservierten Get-Station noch benötigt, hängt von dem Ort ab, an dem er die Cluster-Region erreicht. Dieser wiederum hängt davon ab, von welcher Put-Station der Roboter zu dieser Cluster-Region aufgebrochen ist. Deshalb kann keine einheitliche Zeit für jede Get-Station ermittelt werden, die die Dauer der Fahrt zwischen dem Erreichen der Region und dem Erreichen der Station beschreibt.

- Die Zeit, die der Roboter von der Put-Station des letzten Auftrags zur Cluster-Region des nächsten Auftrags benötigt, auf Basis von Erfahrungswerten zu schätzen, ist nur dann sinnvoll, wenn der Roboter von Abschluss des letzten Auftrags bis zur Ankunft in der Cluster-Region denselben *ClusterOrder* zugewiesen hat. Ist dies nicht der Fall, so ändert sich das Ziel des Roboters während der Fahrt und der Fahrweg des Roboters entspricht nicht mehr dem, den der Roboter gewählt hätte, wäre ihm gleich der neue *ClusterOrder* zugewiesen worden.

Aufgrund des zweiten Punkts ist es also nach einer Veränderung der Auftragsverteilung nicht mehr möglich, Erfahrungswerte für die Zeit, die noch für die Fahrt zur Get-Station benötigt wird, zu verwenden. Stattdessen kann jedoch die benötigte Zeit anhand der Distanz zwischen dem Roboter und der Station und der Durchschnittsgeschwindigkeit des Roboters bestimmt werden. Als Durchschnittsgeschwindigkeit wird ein einziger Wert verwendet, der anhand der euklidischen Distanzen und der entsprechenden Erfahrungswerte berechnet wird.

Je mehr Roboter in einem Lager unterwegs sind, desto öfter treffen zwei Roboter aufeinander und müssen gegenseitig ausweichen. Das bedeutet, dass die durchschnittliche Zeit, die ein Roboter für die Fahrt zwischen zwei Stationen benötigt, von der Anzahl an Robotern im Einsatz abhängt. Deshalb muss die Berechnung dieser Zeit auf Basis eines Datensatzes erfolgen, der mit der gleichen Anzahl eingesetzter Roboter aufgezeichnet wurde. Dies wiederum bedeutet, dass im laufenden Betrieb für jede mögliche Anzahl eingesetzter Roboter ein Datensatz vorhanden sein muss, sodass je nach Anzahl benötigter Roboter eine möglichst genaue Durchschnittszeit berechnet werden kann.

Es kann jedoch auch vorkommen, dass zwei Roboter so ungünstig aufeinandertreffen, dass sie sich für eine gewisse Zeit gegenseitig vollkommen blockieren. Dies verlängert für beide Roboter deutlich die Dauer des Abschnitts der Auftragserfüllung, in dem sie sich gerade befinden. So eine starke Verzögerung kann den für den jeweiligen Abschnitt be-

rechneten Durchschnittswert merklich beeinflussen. Aus diesem Grund berücksichtigen die eben beschriebenen Methoden zur Berechnung der Durchschnittswerte keine Werte, die um das $1,5$-fache größer sind als der bereits bestehende Durchschnittswert. Auch muss bei der Berechnung sichergestellt sein, dass es nicht bereits beim ersten hinzugefügten Wert zu einer großen Verzögerung gekommen ist, da sonst der Durchschnittswert von Beginn an zu hoch angesetzt wäre und starke Verzögerungen so nicht ausgeschlossen würden. Das arithmetischen Mittel, welches bei diesem einfachen Vorgehen verwendet wird, ist jedoch nicht robust gegenüber Ausreißern. Bei einer Verbesserung des Vorgehens wäre es aus diesem Grund angebracht, anstelle des Durchschnittswerts den Median der Messwerte zu verwenden.

Um diese Voraussetzung zu erfüllen, kann für die Initialisierung aller Durchschnittszeiten für jede in der gegebenen Aufgabenstellung mögliche Kombinationen aus eine Get- und einer Put-Station ein Auftrag generiert werden. Um diese Fahrstrecken mit einer minimalen Fahrzeit zu initialisieren, werden diese Aufträge dann in der Simulation von einer so geringen Anzahl an Robotern durchgeführt, dass es zu keiner gegenseitigen Behinderung unterwegs und an den Stationen kommt. Da die Ausführung dieser Aufträge durch einen einzigen Roboter jedoch sehr viel Zeit in Anspruch nehmen würde, kann ein Kompromiss aus der Anzahl an Robotern und der Wahrscheinlichkeit eines Zusammentreffens gewählt werden. Durch Überwachung der Simulation kann sichergestellt werden, dass, falls es dennoch zu einer Verzögerung kommen sollte, die betroffenen Zeiten durch erneutes Simulieren korrekt erfasst werden.

7.3 Belegungsverwaltung der Stationen

Für eine möglichst genaue Evaluierung der für die Auftragsdurchführung benötigten Zeit sind auch die Wartezeiten an jenen Stationen einzuberechnen, die bereits durch andere Roboter besetzt sind. Dafür muss für jede Zuweisung eines Auftrags an einen Roboter die Zeit, die sich dieser an der GET- und an der PUT-Station befindet, in einen Zeitplan für diese Station eingetragen werden. Wird nun die Zeit für einen neuen Auftrag evaluiert, muss geprüft werden, ob sich dieser mit einem existierenden Eintrag überschneidet. Ist dies der Fall, so muss der geplante Aufenthalt nach dem bereits eingetragenen Aufenthalt erfolgen, und die Ausführungszeit des neuen Auftrags verlängert sich um die Wartezeit an dieser Station. Werden unter Verwendung der Belegungsverwaltung die Wartezeiten an den Stationen minimiert, so ergibt dies eine möglichst gleichmäßige Auslastung der Stationen.

Für die Umsetzung dieser Belegungsverwaltung wurde die Klasse *StationTimeHandler*

implementiert. Diese merkt sich für jede Station eine chronologisch sortierte Liste von Objekten des Typs *TimeTableEntry*, welche eine Belegung dieser Station durch einen Roboter darstellen. Das Objekt enthält Informationen über den Roboter und den Auftrag sowie die Start- und die Endzeit dieser Belegung. Die Methode *TimeTableEntry reserveStation(String station, Order order, double startTime, double endTime, String robot)* erlaubt es, während einer zeitlichen Evaluierung einen neuen Eintrag im *StationTimeHandler* zu erstellen. Bei ihrem Aufruf wird der Zeitpunkt übergeben, zu dem der in dieser Zuweisung betrachtete Roboter an der Station eintrifft. Die Methode prüft, ob die Station zu diesem Zeitpunkt bereits durch einen anderen Eintrag belegt ist. Ist dies nicht der Fall, so wird ein neuer Eintrag an dieser Stelle erstellt, und die Endzeit ergibt sich aus der Summe aus Startzeit und der Dauer, die mithilfe der berechneten Durchschnittswerte als Aufenthaltsdauer an der Station geschätzt wurde.

Die Station ist jedoch etwas länger als die reine Dauer der Feinpositionierung und Behälterübergabe besetzt, da ein Roboter bereits beim An- und Abfahren die Station für einen anderen Roboter blockiert. Deshalb werden Start- und Endzeit eines neuen Eintrags eine gewisse Zeit vor und nach der Aufenthaltsdauer des Roboters an der Station gewählt und die Station wird somit länger als die tatsächliche genutzte Zeit für den Roboter reserviert. Da außerdem die Fahrzeit zur Station je nach Aufeinandertreffen der Roboter während der Fahrt unterschiedlich ausfällt und so Abweichungen von der geschätzten Fahrzeit entstehen, kann durch eine etwas längere Reservierung besser vermieden werden, dass sich die Roboter an den Stationen gegenseitig behindern.

Der optimale Wert, um wie viel eine Stationsreservierung verlängert werden soll, ist von Station zu Station unterschiedlich und stellt uns bei der Parametrierung der Belegungsverwaltung vor eine schwierige Aufgabe. Da das getrennte Einstellen dieses Puffers jeder Station viel Erfahrung und Zeit benötigt, wurde in einer ersten Implementierung dieselbe Einstellung für jede Station verwendet und die Parametrierung des Puffers wurde auf einen Wert reduziert. Dieser gibt die absolute Größe des Puffers an und wird bei der Reservierung zur Hälfte vor und zur Hälfte nach der tatsächlichen Zeit an der Station hinzugefügt.

Durch diese längere Reservierung muss jedoch zwischen zwei Zeiten unterschieden werden: Die Reservierungs-Endzeit beschreibt den Zeitpunkt, ab dem ein anderer Roboter in Zukunft die Station reservieren kann. Die Übergabe-Endzeit beschreibt den Zeitpunkt, zu dem der Roboter nach der Übergabe an der Station diese voraussichtlich verlassen wird. Diese Endzeit wird auch zurückgegeben, da sie bei der zeitlichen Evaluierung eines Auftrags eine präzisere Auskunft über die Aufenthaltsdauer des Roboter an der Station

gibt. Abbildung 6 stellt einen Eintrag mit Puffer und Reservierungszeiten dar.

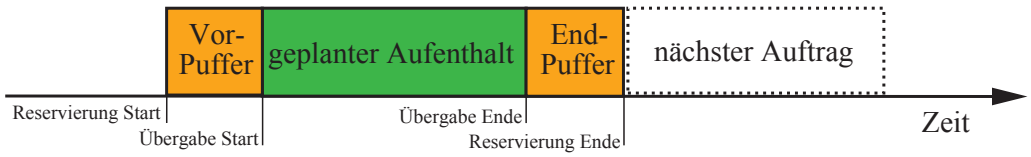

Abbildung 6: Darstellung der verlängerten Stationsreservierung

Ist die Station zum gewünschten Zeitpunkt bereits belegt, wird ein Eintrag an der nächsten freien Stelle erstellt. Die Methode *reserveStation* gibt den erstellen Eintrag an die Zeitevaluierung zurück, woraus diese die relevanten Informationen für die Evaluierung dieser Zuweisung – wie zum Beispiel den Zeitpunkt, zu dem die Station verlassen wird – entnehmen kann.

Abbildung 7 stellt am Beispiel einer Get-Station den eben beschriebenen Ablauf dar.

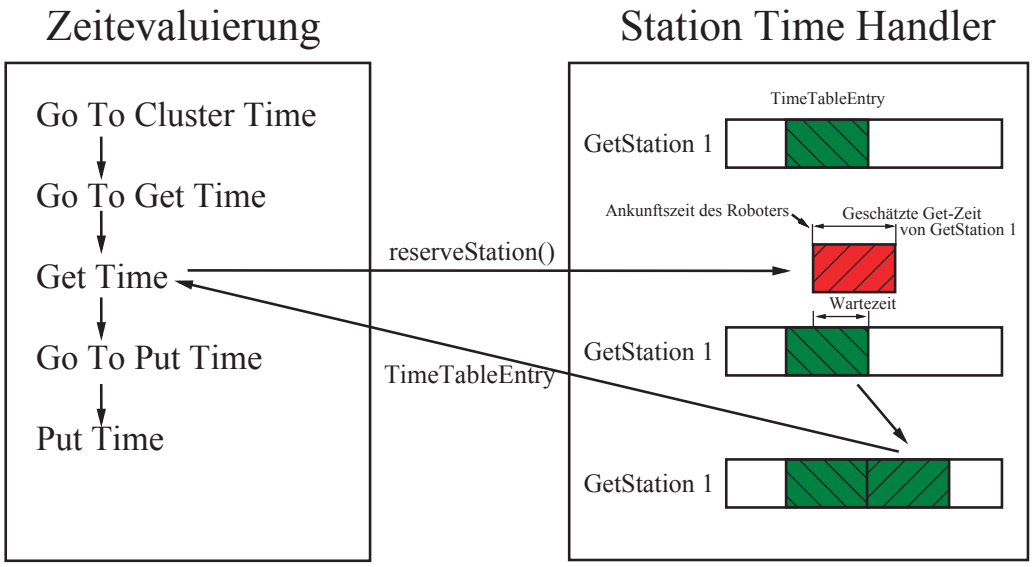

Abbildung 7: Aufbau und Ablauf der Stationsbelegungsverwaltung am Beispiel einer Get-Station

7.4 Implementierung der zeitlichen Evaluierung einer Auftragsverteilung

Im Folgenden wird beschrieben, wie für eine gegebene Auftragsverteilung die Zeit bestimmt wird, die jeder Roboter für die Durchführung der ihm zugewiesenen Aufträge benötigt. Dabei wird neben den geschätzten Zeiten der Einzelschritte und der Position des Roboters zum Zeitpunkt der Berechnung auch die Wartezeit an einer Station, die bereits durch einen anderen Roboter besetzt ist, berücksichtigt.

Für die Implementierung der Methoden, die für die zeitliche Evaluierung einer Auftragsverteilung notwendig sind, wurde die Klasse *TimeEvaluation* entwickelt. Sie verfügt über eine Instanz der Klasse *OrderHistory*, über deren Methoden – wie zuvor beschrieben – Erfahrungswerte für jede zurückzulegende Strecke und für die Aufnahme- und Abgabezeiten an einer Station ermittelt werden können.

Zum Zeitpunkt einer Evaluierung kann sich der Roboter in einem der folgenden Zustände befinden:

- Der Roboter ist im Status *Idle*: Er fährt momentan zu einer Warte-Position oder befindet sich an einer solchen.
- Der Roboter befindet sich im Status *Go_To_Cluster* und fährt momentan zu einer bestimmten Cluster-Region, hat aber noch keinen *preciseOrder* reserviert. Durch Zuweisen eines neuen Auftrags für eine andere Cluster-Region unterbricht der Roboter seine Fahrt und fährt zur Cluster-Region des neu zugewiesenen Auftrags.
- Der Roboter hat bereits einen *preciseOrder* reserviert und führt diesen gerade aus. Das Zuweisen eines neuen Auftrags kann die Durchführung des aktiven Auftrags nicht mehr beeinflussen.

Zu Beginn der Evaluierung muss die Zeit ermittelt werden, die der Roboter für das Erreichen der Get-Station des nächsten ihm zugewiesenen Auftrags benötigt. In den ersten beiden Fällen kann für die Fahrzeit zur Get-Station des ersten Auftrags kein Erfahrungswert verwendet werden, da diese Erfahrungswerte nur für die Fahrt zwischen zwei bekannten Positionen existieren und die Möglichkeit besteht, dass der Roboter gerade unterwegs ist. Deshalb wird stattdessen die Distanz zwischen dem Roboter und der Get-Station ermittelt und die benötigte Zeit für diese Strecke anhand der Durchschnittsgeschwindigkeit des Roboters geschätzt. Da im Layout des zu untersuchenden Projekts die Wege zwischen zwei Stationen meist aus geraden Linien bestehen, weichen diese Schätzungen nur gering von den tatsächlichen Fahrzeiten ab.

Im dritten Fall - bei dem der Roboter gerade einen *preciseOrder* ausführt - setzt sich die Zeit, die er für das Erreichen der Get-Station benötigt, aus zwei Komponenten zusammen: der Zeit, die er für die Vollendung des aktiven Auftrags braucht und der Zeit, die er von der Put-Station des aktiven Auftrags zur Get-Station des nächsten Auftrags benötigt. In Abschnitt 7.4.1 wird dargelegt, wie die Zeit ermittelt wird, die ein Roboter noch für seinen aktuellen Auftrag benötigt. Anschließend wird in Abschnitt 7.4.2 beschrieben, wie – ausgehend von dem Ergebnis dieser Berechnung – die Zeit evaluiert wird, die ein Roboter für einen möglichen nächsten Auftrag benötigt.

Das Ergebnis der zeitlichen Evaluierung eines Roboters wird in einem Objekt des Typs *TimeEvaluationResult* gespeichert. Dieses verfügt über die Felder *activeOrder* und *nextOrder*, welche auf den aktuellen und nächsten zugewiesenen Auftrag verweisen, und die Felder *activeOrderTimePlan* und *nextOrderTimePlan*. Letztere sind vom Typ *OrderTimePlan* und enthalten für beide Aufträge die Zeitpunkte, zu denen der Agent des Roboters die einzelnen Abschnitte der Auftragsdurchführung voraussichtlich erreichen wird. Anhand der aktuellen Zeit und der Statusnachricht des Roboters kann mithilfe dieser Objekte für einen bestimmten Auftrag jederzeit festgestellt werden, ob der berechnete Zeitplan vom durchführenden Roboter eingehalten wird.

7.4.1 Zeitliche Evaluierung des aktuellen Auftrags

Um zu jedem Zeitpunkt in der Lage zu sein, die für einen Teilschritt noch benötigte Zeit zu schätzen, wird die Status-Nachricht, die der Agent des Roboters auf dem topic *robot_status* veröffentlicht, um das Feld *duration* erweitert. Dieses gibt die Dauer an, die sich der Roboter bereits in diesem Status befindet. Anhand dieser Dauer und der Dauer, die für den gesamten Teilschritt ermittelt wurde, kann die noch verbleibende Zeit in diesem Teilschritt geschätzt werden.

Um die Zeit zu bestimmen, die jeder Roboter für die Vollendung des aktiven Auftrags benötigt, wurde in der Klasse *TimeEvaluation* die Methode *Map<String, OrderTimePlan> getAcitveOrderFinishTimes(Map<String, Order> activeOrders, StationTimeHandler stationTimeHandler, double currentTime)* implementiert. Zuerst wird für jeden Roboter sein momentaner Statuszustand bestimmt. Während der Durchführung eines *preciseOrder* kann sich der Roboter in folgenden Status befinden:

a) *Go_To_Get*

b) *Get*

c) *Go_To_Put*

d) *Put*

In den Fällen *b) - d)* wird die Dauer, die sich der Roboter in diesem Status befindet, mithilfe des Felds *duration* der Statusnachricht bestimmt. Befindet sich der Roboter jedoch im Status *Go_To_Get*, wird die Fahrzeit, die der Roboter noch bis zur Ankunft an der Station benötigt, anhand der Distanz zwischen ihm und der Station geschätzt. Dies ist notwendig, da es für die Zeit, die der Roboter in diesem Status verbringt, keine Schätzungen gibt.

Danach wird die Zeit bestimmt, die der Roboter noch im aktuellen Status verbringen wird. Sie wird aus der Differenz zwischen der gesamt benötigten Zeit aus der *OrderHistory* und der Zeit in diesem Status berechnet. Die Differenz wird dann zur aktuellen Zeit addiert und als Endzeit für den aktuellen Auftragsabschnitt in eine neue Instanz der Klasse *OrderTimePlan* eingetragen. Ist die berechnete Differenz negativ, so bedeutet dies, das sich der Roboter schon länger als die dafür geschätzte Zeit in diesem Abschnitt befindet. In diesem Fall wird angenommen, dass der aktuelle Abschnitt jeden Moment abgeschlossen wird, und es wird die aktuelle Zeit in den Zeitplan eingetragen.

Danach wird für jeden Abschnitt, der dem aktuellen in der Auftragsdurchführung folgt, die geschätzte benötigte Dauer dieses Abschnitts zur Endzeit des letzten Abschnitts addiert und in denselben Zeitplan eingetragen. In den Abschnitten, in denen der Roboter an einer Station steht, wird über die Methode *reserveStation* der übergebenen Instanz des *StationTimeHandler* eine Reservierung in dieser Instanz erstellt. Es wird die Zeit übergeben, die für die Ankunft an der Station geschätzt wurde. Als Rückgabe kommt der erstellte Eintrag vom Typ *TimeTableEntry*, aus dem die Zeit, zu der der Roboter die Station wieder verlässt, entnommen werden kann. Da anhand dieser Zeit die Evaluierung der für den aktuellen Auftrag benötigten Zeit fortgesetzt wird, wird bei dieser Evaluierung die Wartezeit an den Stationen mitberücksichtigt.

Beim ersten Aufruf der Methode *getActiveOrderFinishTimes* wird eine neu erstellte Instanz der Klasse *StationTimeHandler* übergeben, in der noch keine Stationsreservierungen vorhanden sind. Nachdem alle Roboter evaluiert wurden, die über einen aktiven Auftrag verfügen, sind die aktuellen Stationsreservierungen in diese neue Instanz eingetragen. Im folgenden Abschnitt wird beschrieben, wie anhand der bestimmten Endzeiten und der erstellten *StationTimeHandler*-Instanz begonnen werden kann, mögliche nächste Auftragszuweisungen zu untersuchen.

7.4.2 Zeitliche Evaluierung eines potentiellen nächsten Auftrags

Anhand der eben erklärten *activeOrderFinishTimes* kann nun die Zeit evaluiert werden, die ein beliebiger Roboter benötigt, um einen neuen, noch nicht fix zugewiesenen Auftrag, durchzuführen. Dabei soll nicht nur der Zeitpunkt ermittelt werden, zu dem der Auftrag abgeschlossen wird, sondern auch die Leerzeit, die bei der Durchführung dieses Auftrags durch diesen Roboter anfällt. Dafür wurde die Methode *executeOrder(String robot, Map<String, OrderTimePlan> getAcitveOrderFinishTimes, StationTimeHandler stationTimeHandler)* zur Klasse *TimeEvaluation* hinzugefügt. Sie überprüft zuerst, ob für diesen Roboter ein Eintrag in der Liste *activeOrderFinishTimes* existiert. Ist dies der Fall, so führt dieser Roboter gerade einen *preciseOrder* aus, und die zeitliche Evaluierung des nächsten Auftrags beginnt zum Endzeitpunkt des aktuellen Auftrags, welcher aus dem in *activeOrderFinishTimes* gespeicherten *activeOrderTimePlan* entnommen wird. Als Position des Roboters zu Beginn der Evaluierung wird die Put-Station des aktuellen Auftrags verwendet. Führt der Roboter momentan keinen *preciseOrder* aus, so kann er sofort mit der Durchführung des nächsten Auftrags beginnen, und es wird die aktuelle Zeit und die aktuelle Position des Roboters als Ausgangspunkt für die zeitliche Evaluierung verwendet.

Zuerst wird die Zeit bestimmt, die der Roboter zur Get-Station des zu evaluierenden Auftrags benötigt. Führt der Roboter gerade einen Auftrag aus, so wird diese Zeit anhand der Schätzung aus der *OrderHistory* bestimmt. Führt er hingegen keinen Auftrag aus, so erfolgt dies anhand der Distanz zwischen dem Roboter und der Get-Station. Die für das Erreichen der Get-Station erforderliche Zeit wird an drei unterschiedlichen Stellen verwendet:

- Sie wird zur aktuellen Zeit addiert und als *getArrivalTime* in einen neuen Zeitplan vom Typ *OrderTimePlan* eingetragen. Dieser Wert wird in der Zeitüberwachung verwendet, um die Dauer, die der Roboter bis zu diesem Zeitpunkt in diesem Abschnitt sein sollte, zu bestimmen.

- Sie wird als getrenntes Feld *goToGetDuration* in das *TimeEvaluationResult* eingetragen. Dieser Wert wird von der Zeitüberwachung verwendet, um die erlaubte Zeit in diesem Abschnitt zu bestimmen.

- Sie wird zur Variable *delay*, welche die Leerzeit dieser Auftragsevaluierung bestimmt und zuvor mit Null initialisiert wurde, aufsummiert. Die Leerzeit wird vom Auftragsverteilungsalgorithmus verwendet, um den besten Auftrag für einen Roboter zu finden.

Anschließend wird für jeden weiteren Abschnitt der Auftragsdurchführung die benötigte Zeit anhand der *OrderHistory* bestimmt. Die Zeitspanne, die der Roboter an den Stationen verbringt, wird anhand der übergebenen Instanz des *StationTimeHandler*s ermittelt. Aus der Instanz vom Typ *TimeTableEntry*, welche bei der Reservierung der Station vom *StationTimeHanlder* zurückgegeben wird, kann die Wartezeit für diesen Roboter an der Station entnommen werden. Da diese Zeit auch zur Leerzeit einer Auftragsdurchführung zählt, wird sie der Variable *delay* aufsummiert.

Die benötigte Zeit jedes Abschnitts wird zur Endzeit des letzten Abschnitts addiert und in den Zeitplan eingetragen. Dies wird für jeden Abschnitt durchgeführt, bis alle Informationen in die Ergebnis-Instanz vom Typ *TimeEvaluationResult* eingetragen wurden. Diese enthält dann den vollständigen Zeitplan, wann der Roboter die einzelnen Teilabschnitte des Auftrags absolvieren kann, und die Leerzeit, die bei einer Durchführung des Auftrags von diesem Roboter anfällt.

8 Implementierung des Auftragsverteilungsalgorithmus

Im folgenden Kapitel wird ausgeführt, welche Erweiterungen am System vorgenommen und welche neuen Komponenten dem System hinzugefügt wurden, um die in Abschnitt 6.2 beschriebene Auftragsverteilung zu realisieren.

8.1 Auftragsverwaltung

Zur Verwaltung der Auftragsverteilung wurde dem *BaseOrderManager* eine Instanz der neu entwickelten Klasse *AssignmentHandler* zur Verfügung gestellt. Sie verfügt über die Liste *availableRobots*, in der alle Roboter stehen, die momentan aktiv sind und Aufträge zugewiesen bekommen können. Denn nicht jeder Roboter, der eingeschaltet und im System aufscheint, kann auch Aufträge durchführen. Ist beispielsweise der Ladestand eines Roboters zu gering, sodass er zur Ladestation fahren muss, soll er aus der Auftragsverteilung genommen und erst dann wieder hinzugefügt werden, wenn der Ladevorgang in absehbarer Zeit abgeschlossen sein wird.

Aus diesem Grund wurde ein An- und Abmeldesystem entwickelt. Dazu bietet der *AssignmentHandler* zwei neue Services an, *addRobotToAssignment* und *removeRobotFromAssignment*. Über sie kann ein Roboter veranlassen, in die Liste der *availableRobots* aufgenommen oder aus ihr entfernt zu werden. Außerdem wurden der *FiniteStateMachine* des Roboters zwei neue Knoten hinzugefügt. Der Knoten *add_to_assignment* wird jedes Mal durchlaufen, wenn bei einem Roboter die Auftragsdurchführung aktiv ist, sein Ladestand noch über einem gewissen Wert ist und er keinen Auftrag zugewiesen hat. Dabei wird über das Service *addRobotToAssignment* geprüft, ob der Roboter bereits in der Liste der *availableRobots* steht – wenn nicht, wird er hinzugefügt. Der Knoten *remove_from_assignment* wird an jeder Stelle in der Finite State Machine durchlaufen, an der der Roboter nicht mehr an der Auftragsbearbeitung teilnehmen soll, zum Beispiel wenn er zur Ladestation fahren muss.

Die Klasse *AssignmentHandler* speichert die aktuelle Auftragsverteilung in der *Map<String, TimeEvaluationResult> currentAssignments*, in der für jeden Roboter, welcher bei der letzten Neuberechnung bereits zur Auftragsverteilung angemeldet war, ein Element vom Typ *TimeEvaluationResult* hinterlegt ist. Dieses ist das Ergebnis der zeitlichen Evaluierung und enthält für jeden Roboter – neben dem aktiven und dem nächsten Auftrag (*activeOrder* und *nextOrder*) – auch die Zeitpläne, die für diese beiden Aufträge berech-

net wurden (*activeOrderTimePlan* und *nextOrderTimePlan*).

Versucht der Agent eines Roboters über die *OrderManagerNode* einen Cluster zu reservieren, wird ihm der *clusterOrder* des zu diesem Zeitpunkt an erster Stelle zugewiesenen Auftrags zurückgegeben. Ist er in der Cluster-Region angekommen und versucht, einen *preciseOrder* zu reservieren, wird ihm der momentan erstgereihte Auftrag übergeben, und dieser wird aus den *currentAssignments* dieses Roboters entfernt.

Hat ein Roboter einen *preciseOrder* erfolgreich zugewiesen bekommen, so merkt sich der *BaseOrderManager* diese Roboter-Auftrags-Kombination in der *Map<String, Order> activeOrders*. Wurde der Auftrag erfolgreich abgeschlossen und der *OrderFinished* Knoten in der *Finite State Machine* erreicht, wird diese Roboter-Auftrags-Kombination aus dieser Map wieder entfernt. Über sie ist es also möglich, zu jedem Zeitpunkt genau zu bestimmen, welcher Roboter gerade einen Auftrag ausführt, der durch eine Neuzuweisung nicht mehr verändert werden kann. Dies trifft auf alle Roboter zu, die momentan zur Auftragsverteilung angemeldet sind und nicht im Zustand *Go_To_Cluster* oder *Idle* sind.

Der *AssignmentHandler* verfügt über die Methode *updateAssignments(List<Order> orders, Set<String> robots), Map<String, Order> activeOrders, double currentTime)*, welche vom *BaseOrderManager* regelmäßig aufgerufen wird. Sie prüft, ob eine Neuberechnung der Auftragsverteilung notwendig ist. Ist dies der Fall, so berechnet sie eine neue Verteilung und weist diese dem Feld *currentAssignments* als neuen Wert zu.

Im folgenden Abschnitt wird nun dargestellt, wie die Berechnung der Auftragsverteilung implementiert wurde. Danach zeigt Abschnitt 8.3, wie geprüft wird, ob eine Neuberechnung der Auftragsverteilung notwendig ist.

8.2 Berechnung der Auftragsverteilung

Der Auftragsverteilungsalgorithmus soll aus allen Aufträgen, die zum Zeitpunkt der Berechnung verfügbar sind, jedem Roboter einen dieser Aufträge zuweisen. Um für einen Roboter seinen nächsten Auftrag bestimmen zu können, muss zuerst ermittelt werden, wann er seinen aktiven Auftrag – also den Auftrag, den er im Augenblick der Berechnung gerade ausführt – abschließen wird. Ausgehend von diesem Zeitpunkt und dem Ort, an dem sich der Roboter nach Erledigung dieses Auftrags befinden wird, kann dann der beste nächste Auftrag für diesen Roboter gewählt werden.

Bei der Auftragsverteilung ist darauf zu achten, dass für jeden Auftrag dessen Zeitfenster respektiert wird. Dieses besteht zum Ersten aus einer Startzeit, welche den frühesten Zeitpunkt angibt, zu dem der Roboter den Behälter von der Get-Station des Auftrags

abholen kann. Zum Zweiten besteht sie aus einer Endzeit, welche den Zeitpunkt angibt, an dem der Roboter spätestens den Behälter an der Put-Station abgeben muss.

Die Einhaltung der Startzeit eines Auftrags wird sichergestellt, indem bei der zeitlichen Evaluierung eines Auftrags geprüft wird, ob zum Zeitpunkt der Ankunft des Roboters an der Get-Station die Startzeit bereits erreicht ist. Ist dies nicht der Fall, so wird dieser Auftrag noch nicht zugewiesen. Die Einhaltung der Endzeit eines Auftrags wird sichergestellt, indem zuerst die Aufträge verteilt werden, deren Endzeit bald erreicht ist. Dabei erfolgt die Verteilung nach frühester Fertigstellungszeit, das bedeutet, dass jener Roboter den Auftrag zugewiesen bekommt, der ihn zum frühest möglichen Zeitpunkt abschließen kann.

Zuerst wird mithilfe der in Abschnitt 7.4.1 beschriebenen Methode *getActiveOrderFinishTimes* für jeden Roboter, der momentan einen Auftrag ausführt, der Zeitplan dieses Auftrags berechnet und die Zeiten, an denen er eine Station besetzt, werden in den *StationTimeHandler* eingetragen. Danach werden aus allen verfügbaren Aufträgen jene Aufträge ermittelt, deren Endzeit in einer bestimmten Zeit erreicht werden, und in die Liste *lateOrders* verschoben. Diese wird dann nach aufsteigender Endzeit sortiert.

Algorithmus 2 beschreibt, wie bei der Zuweisung der *lateOrders* vorgegangen wird. Eine Zuweisung erfolgt, indem in der neu erstellten *HashMap<String, TimeEvaluationResult> newAssignments* ein neuer Eintrag hinzugefügt wird.

Bei der Zuweisung der *lateOrders* sowie bei der restlichen Auftragsverteilung wird die Instanz des *StationTimeHandlers*, in der bereits die Stationsreservierungen für die Fertigstellung der aktiven Aufträge eingetragen sind, für das Ausprobieren von möglichen Verteilungen verwendet. Dazu wird beim Ausführen der Evaluierung eines Auftrags eine Kopie der Instanz übergeben, in der die untersuchte Auftragsverteilung eingetragen wird. Diese wird dann mit der *TimeEvaluationResult*-Instanz zurückgegeben. Ist nach dem Evaluieren mehrerer möglicher Zuweisungen die beste Zuweisung ermittelt, so wird dem Roboter dieser Auftrag fix zugewiesen, und die davor verwendete Instanz des *StationTimeHandler*s wird durch jene aus dem *TimeEvaluationtResult*s dieser Zuweisung ersetzt. Auf diese Weise werden bei jeder weiteren Untersuchung einer möglichen Auftragszuweisung die bisher durchgeführten Zuweisungen berücksichtigt.

Danach wird geprüft, ob die Menge der noch verfügbaren Aufträge die Zahl der Roboter ohne zugewiesenen Auftrag übersteigt bzw. gleich groß ist. Trifft dies zu, so stehen für jeden Roboter mehrere Aufträge zur Auswahl, und es wird – wie in Abschnitt 6.2.2 beschrieben – die Roboter-Auftrags-Kombination mit minimaler Leerzeit gewählt.

Ist die Anzahl an verfügbaren Aufträgen jedoch geringer als die Anzahl an Robotern

Algorithmus 2 Verteilung der *lateOrders*

```
Input: robots, lateOrders, activeOrderFinishTimes
stationTimeHandler ← new StationTimeHandler()
for all lateOrder ∈ lateOrders do
    bestRobot ← null
    minFinishTime ← ∞
    bestTimeEvalResult ← null
    for all robot ∈ availableRobots do
        timeEvalResult ← TimeEvaluation.executeOrder(lateOrder,
            activeOrderFinishTimes, stationTimeHandler.clone())
        finishTime ← timeEvalResult.getNextOrderTimePlan.getPutFinishTime()
        if finishTime < minFinishTime then
            minFinishTime ← finishTime
            bestRobot ← robot
            bestTimeEvalResult ← timeEvalResult
        end if
    end for
    if bestTimeEvalResult ≠ null then
        newAssignments.put(bestRobot, bestTimeEvalResult)
        availableRobots.remove(bestRobot)
        stationTimeHandler ← bestTimeEvalResult.getStationTimeHandler()
    end if
end for
```

ohne Auftrag, so ist für eine Maximierung des Gesamtdurchsatzes eine Verteilung nach minimaler Leerzeit nicht mehr so gut geeignet. Bei einer solchen Zuweisung werden nämlich Aufträge Robotern zugeteilt, die unter anderem die kürzeste Anfahrtszeit zu einer Get-Station haben. Dabei wird aber nicht berücksichtigt, dass ein Roboter mit längerer Anfahrtszeit seinen aktuellen Auftrag rascher erfüllt hat als der andere Roboter den seinen. Dies kann bei einer Verteilung nach minimaler Leerzeit dazu führen, dass ein Roboter ohne zugewiesenen Auftrag bereits an seiner Warteposition steht, während ein anderer Roboter zuerst seinen aktuellen Auftrag abschließt und dann noch einen weiteren Auftrag zugewiesen bekommen hat. Deshalb werden, wenn es weniger verfügbare Aufträge als Roboter gibt, die Aufträge jenen Roboter zugeteilt, die sie am frühesten fertigstellen können. Algorithmus 3 beschreibt die Implementierung des Auftragsverteilungsalgorithmus.

Algorithmus 3 Auftragsverteilungsalgorithmus

Input: $robots, orders, activeOrderFinishTimes, stationTimeHandler$
while $robots.size() > 0 \wedge orders.size() > 0$ **do**
  $minDelay \leftarrow \infty$
  $minFinishTime \leftarrow \infty$
  $bestRobot \leftarrow null$
  $bestOrder \leftarrow null$
  $bestTimeEvalResult \leftarrow null$
  for all $order \in orders$ **do**
    for all $robot \in robots$ **do**
      $timeEvalResult \leftarrow TimeEvaluation.executeOrder(order,$
        $activeOrderFinishTimes, stationTimeHandler.clone())$
      $finishTime \leftarrow timeEvalResult.getNextOrderTimePlan.getPutFinishTime()$
      $delay \leftarrow timeEvalResult.getTotalDelay()$
      if $robots.size() \leq orders.size() \wedge delay < minDelay$ **then**
        $minDelay \leftarrow delay$
        $bestRobot \leftarrow robot$
        $bestOrder \leftarrow order$
        $bestTimeEvalResult \leftarrow timeEvalResult$
      else
        if $robots.size() > orders.size() \wedge finishTime < minFinishTime$ **then**
          $minFinishTime \leftarrow finishTime$
          $bestRobot \leftarrow robot$
          $bestOrder \leftarrow order$
          $bestTimeEvalResult \leftarrow timeEvalResult$
        end if
      end if
    end for
  end for
  if $bestTimeEvalResult \neq null$ **then**
    $newAssignments.put(bestRobot, bestTimeEvalResult)$
    $orders.remove(bestOrder)$
    $availableRobots.remove(bestRobot)$
    $stationTimeHandler \leftarrow bestTimeEvalResult.getStationTimeHandler()$
  end if
end while

8.3 Auslösen einer Neuberechnung

Nachdem nun klar ist, wie die Berechnung einer neuen Auftragsverteilung erfolgen soll, wird in diesem Abschnitt beschrieben, wie der geeignete Moment bestimmt wird, an dem sie durchgeführt wird.

8.3.1 Zeitpunkt der Neuberechnung

Eine Neuberechnung kann aus verschiedenen Gründen notwendig sein. Zum Einen muss ja für einen Roboter, der für die Auftragsverteilung angemeldet ist und dem im Augenblick kein Auftrag zugewiesen ist, ein neuer Auftrag bestimmt werden.

Zum Anderen muss auch überprüft werden, ob sich die Bedingungen, die zum Zeitpunkt der Berechnung gegeben waren, inzwischen nicht geändert haben. Eine Verteilung, die bei der letzten Berechnung noch alle Aufträge in den gegebenen Zeitfenstern und mit geringer Leerzeit erfüllt hat, kann nach kurzer Zeit schon nicht mehr für die aktuelle Situation geeignet sein. Zu einer solchen Veränderung der Bedingungen kommt es, wenn ein oder mehrere Roboter durch ihr nicht-deterministisches Verhalten für einen Abschnitt der Auftragsdurchführung mehr oder weniger Zeit benötigen als gewöhnlich.

Eine Möglichkeit, auf solche Verzögerungen zu reagieren, besteht darin, die Auftragsverteilung in kurzen Abständen ständig neu zu berechnen. Ist jedoch der Unterschied zwischen zwei möglichen Aufträgen eines Roboters sehr gering, kann es bei ständiger Neuberechnung dazu kommen, dass der Auftrag, welcher dem Roboter zugewiesenen ist, häufig wechselt. Da dieser Roboter bei jedem Wechsel ein neues Ziel zugewiesen bekommt, kann er in keine Richtung richtig Fahrt aufnehmen, sodass er zwischen den beiden Aufträgen "hängenbleibt". Dadurch geht mehr Zeit verloren, als hätte der Roboter von Beginn an den schlechteren Auftrag zugewiesen bekommen.

Aus diesem Grund soll die Auftragsverteilung nur dann neu berechnet werden, wenn ein Roboter für seinen aktuellen Abschnitt der Auftragsdurchführung um einen gewissen Schwellenwert länger braucht als vorgesehen. So kann sichergestellt werden, dass bei einer Verzögerung alle Aufträge noch innerhalb ihres Zeitfensters erfüllt werden, ohne einen häufigen Wechsel der zugewiesenen Aufträge zu veranlassen. Der Schwellenwert wurde so gewählt, dass ab einer Abweichung von über 30% der für einen Abschnitt geplanten Dauer die Neuberechnung der Auftragsverteilung ausgelöst wird.

Wird jedoch immer nur die Dauer eines Abschnitts mit der für diesen erlaubten Zeit verglichen, so kann es vorkommen, dass bei mehreren aufeinanderfolgenden Abschnitten eine Abweichung in die selbe Richtung auftritt, die jedes Mal knapp unter dem

Schwellenwert liegt. In diesem Fall würde durch die kumulierten Abweichungen eine deutliche Abweichung vom ursprünglich berechneten Zeitplan entstehen, ohne dass eine Neuberechnung ausgelöst wird. Um dies zu verhindern, soll beim Übergang von einem Abschnitt in den nächsten die Abweichung vom letzten in den neuen Abschnitt übernommen werden. Dies ist zu erreichen, indem die Abweichung nicht als Unterschied zwischen der geplanten und der tatsächlichen Dauer in einem Abschnitt berechnet wird, sondern als Unterschied zwischen dem geplanten Zeitpunkt, der bei der Berechnung des Zeitplans bestimmt wurde, und der aktuellen Zeit. So ist sichergestellt, dass sogleich eine Neuberechnung ausgelöst wird, wenn ein Roboter zu sehr von seinem vorausberechneten Zeitplan abweicht.

Es ist jedoch auch möglich, dass ein Roboter die einzelnen Abschnitte der Auftragsdurchführung schneller als geplant erfüllt, da ja die Erfahrungswerte aus der *OrderHistory* auf Messungen aus einer Simulation basieren, die mit einer gewissen Anzahl an Robotern durchgeführt wurde, um die Verzögerungen, die beim Aufeinandertreffen von zwei Robotern entstehen, in den Erfahrungswerten zu berücksichtigen. Trifft ein Roboter dann zufällig bei der Durchführung seines Auftrags in mehreren aufeinanderfolgenden Abschnitten auf keinen anderen Roboter, so entsteht eine Vorsprung gegenüber dem Zeitplan. Da dies zudem eine Änderung der Bedingungen einer Auftragsverteilung bedeutet, wird auch in diesem Fall eine Neuberechnung ausgelöst.

8.3.2 Implementierung des Auslösemechanismus

Um diese Anforderungen umzusetzen, wurde die Methode *updateAssignments* im *AssignmentHandler* erstellt. Sie wird vom *OrderManager* mit einer einstellbaren Frequenz aufgerufen und überprüft zuerst, ob ein Roboter, der zur Auftragsverteilung angemeldet ist, noch keinen Eintrag in *currentAssignments* hat. Dies ist der Fall, wenn der Roboter neu zur Auftragsverteilung hinzugefügt wurde und noch überhaupt keinen Auftrag zugewiesen bekommen hat. Danach wird überprüft, ob das Feld *nextOrder* des zum Roboter gehörenden *TimeEvaluationResult* leer ist. Dies ist der Fall, wenn für diesen Roboter der Wechsel vom nächsten zum aktiven Auftrag stattgefunden hat. Trifft eine dieser beiden Bedingungen zu, wird sofort eine Neuberechnung veranlasst.

Das *null*-setzen des nächsten Auftrags erfolgt jedoch nicht sofort beim Wechsel vom aktiven zum nächsten Auftrag. Schließt der Roboter seinen Auftrag ab, so wird in *currentAssignments* das Feld *activeOrder* auf den Wert des Felds *nextOrder* und das Feld *activeOrderTimePlan* auf den Wert des Felds *nextOrderTimePlan* gesetzt. Dadurch hat der Roboter einen neuen Auftrag zugewiesen bekommen und er beginnt diesen auszu-

führen, es wird jedoch noch keine Neuberechnung ausgelöst. Dies geschieht erst, wenn der Roboter bei der Ausführung des neuen Auftrags die Get-Station erreicht.

Dieses verspätete Auslösen der Neuberechnung hat zwei Vorteile: Zum Ersten ist die Präzision der Schätzung der Zeit, zu der der Roboter die Abschnitte des nächsten Auftrags erreicht, umso höher, je später in der Durchführung des aktiven Auftrags die Berechnung durchgeführt wird. Der zweite Grund hängt mit der Bestimmung der Zeit zusammen, die der Roboter für die Fahrt von der Put- zur Get-Station benötigt. Diese kann entweder durch das Zurückgreifen auf einen Erfahrungswert bestimmt werden, oder indem die Distanz zwischen dem Roboter und der Station gemessen und mithilfe der Durchschnittsgeschwindigkeit des Roboter die dafür benötigte Zeit berechnet wird. Da die erste Variante präziser ist, wird, solange sich die Auftragszuweisung eines Roboters nicht ändert, die benötigte Zeit anhand des Erfahrungswerts bestimmt. Bekommt der Roboter jedoch – während er sich auf dem Weg zur Cluster-Region befindet – einen neuen Auftrag zugewiesen, so kann für diesen neuen Auftrag auf keinen Erfahrungswert zurückgegriffen werden, da sich der Roboter zum Zeitpunkt der Berechnung nicht an einer Station befindet und der Weg, den er gefahren ist, nicht jenem entspricht, den er bei der Messung der benötigten Zeit zurückgelegt hat. Ist dies der Fall, so muss die zweite Variante verwendet und die Zeit anhand der verbleibenden Distanz zur Station bestimmt werden.

Da nur dann mit Sicherheit gesagt werden kann, dass der Roboter direkt von der Put- zur Get-Station fährt, wenn inzwischen keine Neuberechnung stattgefunden hat, wird eine solche erst nach dem Erreichen der Get-Station ausgelöst. So basiert die Schätzung des Zeitpunkts, an dem der Roboter an der Get-Station ankommt, solange noch auf den Erfahrungswerten. Wird jedoch eine Neuberechnung von einem anderen Roboter ausgelöst, so kann für jeden Roboter im Status *GoToCluster* die verbeleibende Zeit nur mehr anhand der Distanz bestimmt werden.

Dies wird sichergestellt, indem – wie in Abschnitt 7.4.1 beschrieben – bei der Neuberechnung der Status jedes Roboters geprüft wird; befindet er sich im Status *Go_To_Get*, wird die benötigte Zeit anhand der noch zurückzulegenden Distanz berechnet. Dieser Wert wird dann im *activeOrderTimePlan* der momentanen Auftragsverteilung gespeichert, welche bei dieser Zeitüberwachung verwendet wird. Solange jedoch keine Neuberechnung stattgefunden hat, ist der *activeOrderTimePlan* der momentanen Auftragsverteilung schon zuvor berechnet worden, als der Roboter noch einen anderen Auftrag durchführte und dieser Auftrag noch als *nextOrder* behandelt wurde. In diesem Fall basiert die geschätzte Zeit, die der Roboter von der Put- zur Get-Station benötigt, ebenfalls

auf Erfahrungswerten.

Wurde allen Robotern ein nächster Auftrag zugewiesen, wird von der Methode *updateAssignments* noch überprüft, ob jeder Roboter in der für den aktiven Auftrag berechneten Zeit liegt. Dies wird mit der Methode *boolean checkOrderTimes(Map<String, TimeEvaluationResult> currentAssignments, OrderHistory orderHistory, double now)* der Klasse *TimeControl* durchgeführt.

Dazu werden die in *currentAssignments* hinterlegten Zeitpläne verwendet. Es wird für jeden Roboter zuerst anhand des Status geprüft, in welchem Abschnitt der Auftragsdurchführung er sich gerade befindet. Für diesen Abschnitt wird dann der Zeitpunkt, an dem der Roboter in diesen Abschnitt hätte wechseln sollen, aus dem Zeitplan der aktuellen Auftragsverteilung ausgelesen. Durch Subtraktion dieses Zeitpunkts von der aktuellen Zeit wird die Dauer berechnet, die sich der Roboter laut Zeitplan bereits in diesem Abschnitt befinden sollte. Diese Dauer wird dann mit der für diesen Abschnitt aus den Erfahrungswerten geschätzten Dauer verglichen. Ist Gleichung 7 für irgendeinen der Roboter erfüllt, so gibt die Methode *checkOrderTimes false* zurück und eine Neuberechnung wird durchgeführt.

$$\left| \frac{Dauer\ aktuell}{Dauer\ geschätzt} \right| > 1,3 \tag{7}$$

Abbildung 8 stellt ein Beispiel dar, bei dem sich zwei Roboter gegenseitig lange behindern, weil ein Hindernis ihre Möglichkeiten zum Ausweichen einschränkt. *Roboter 1* hat den Abschnitt *Get* schon verspätet abgeschlossen, diese Abweichung wird in die aktuelle Dauer des folgenden Abschnitts *Go_To_Put* übernommen. In diesem Abschnitt wird er durch *Roboter 2* so lange behindert, bis die Zeit, die zwischen dem geplanten Beginn dieses Abschnitts und diesem Moment vergangen ist, die geschätzte Dauer für diesen Abschnitt um 30% überschreitet.

Eine Möglichkeit zur Verbesserung dieses Auslösemechanismus besteht darin, zu versuchen, eine Abweichung vom Zeitplan schon vor Ablauf der für diesen Abschnitt geplanten Zeit zu erkennen. Dies erlaubt es, eine Neuberechnung der Auftragsverteilung so rasch als möglich nach dem Eintreten der Verzögerung auszulösen. Da Verzögerungen hauptsächlich in den Abschnitten vorkommen, in denen sich ein Roboter zwischen zwei Stationen bewegt, kann die Distanz, die ihn noch von seinem Ziel trennt, als Fortschritt der momentanen Fahrt überwacht werden.

Abbildung 8: Zeitüberwachung der Auftragsdurchführung

8.3.3 Blockieren von Robotern

Wurde eine Neuberechnung ausgelöst, so wird die Zeitüberwachung beim nächsten Aufruf von *updateAssignments()* auf Basis von den neu berechneten Zeitplänen durchgeführt. Befindet sich ein Roboter während einer Neuberechnung jedoch schon länger in einem Abschnitt, als die für diesen Abschnitt geschätzte Gesamtzeit, so steht als Abschlusszeit für diesen Abschnitt die aktuelle Zeit im Zeitplan, und der Roboter löst nach kurzer Zeit wieder eine Neuberechnung aus. Das bedeutet, dass die Auftragsverteilung in kurzen Abständen immer wieder neu berechnet wird, falls ein Roboter an einer Stelle im Layout blockiert ist und es längere Zeit nicht schafft, seinen aktuellen Abschnitt der Auftragsdurchführung zu beenden. Wurde diesem Roboter ein nächster Auftrag zugewiesen, dessen Endzeit bald erreicht ist, wird dieser bei einer Neuberechnung einem anderen Roboter übertragen, der diesen früher erledigen kann. So kann durch die ständige Neuberechnung sichergestellt werden, dass bei keinem Auftrag, welcher noch nicht fix einem Roboter zugewiesen ist, die Endzeit des Zeitfensters stark überschritten wird.

Trotzdem bringt das Blockieren eines Roboters andere Probleme mit sich. Es kann sein, dass der aktuelle Auftrag des Roboters über eine hohe Priorität verfügt und der Schaden, den seine Verspätung bewirkt, von hohem Ausmaß ist. Außerdem können durch blockierte Roboter, die beispielsweise an einer engen Stelle stehen, immer mehr Roboter behindert werden, die an dieser Stelle vorbeifahren wollen. Dadurch wird in kurzer Zeit der Durchsatz des gesamten Transportsystems stark reduziert.

Aus diesen Gründen ist es wichtig, beim Blockieren eines Roboters so schnell wie möglich von außen einzugreifen. Die im Rahmen dieser Masterarbeit entwickelte Zeitüberwachung bietet eine gute Möglichkeit, ein Frühwarnsystem für das Lagerpersonal zu entwickeln, sodass durch sofortiges Eingreifen der Schaden, der durch das Blockieren eines Roboters entsteht, minimiert werden kann.

8.4 Berücksichtigung des Ladestands

Durch die in Kapitel 3 getätigten Maßnahmen ist es möglich, bei der Entwicklung und beim Testen der neuen Auftragsverteilung den Ladestand der Roboter zu berücksichtigen. Es gibt zwei Möglichkeiten, den Ladestand in der Auftragsverteilung zu berücksichtigen:

a) Versucht man, den Durchsatz des Transportsystems durch eine Minimierung der Fahrwege zu maximieren, so kann der letzte Auftrag, den der Roboter vor der Fahrt

zur Ladestation durchführt, so gewählt werden, dass der Weg von der Put-Station des letzten Auftrags zur Ladestation möglichst kurz ist.

b) Es kann anhand des Ladestands schon im Voraus festgestellt werden, nach welchem Auftrag der Roboter einen so niedrigen Ladestand haben wird, dass er zur Ladestation wird fahren müssen. Nach diesem Auftrag soll ihm kein weiterer Auftrag mehr zugewiesen werden.

Bei der derzeit gewählten Form der Energieversorgung mit Batterien kann von einer Entladedauer von ungefähr $3,5$ Stunden ausgegangen werden. Auf dem aktuellen Layout des zu untersuchenden Projekts sind die Positionen der Ladestationen zwar noch nicht bestimmt, geht man jedoch vom schlimmsten Fall aus und nimmt an, die Ladestation befindet sich ganz am anderen Ende des Lagers, so dauert die Fahrt eines Roboters zu dieser Ladestation trotzdem maximal 55 Sekunden.

Versucht man jetzt, wie in *a)* erwähnt, den Durchsatz des Transportsystems zu maximieren, so besteht die einzige Möglichkeit darin, die Fahrzeit zur Ladestation zu minimieren. Bei einer Ladezeit von 50 Minuten und einer daraus folgenden gesamten Zykluszeit von über 4 Stunden, beträgt die Ersparnis der Zeit, die durch eine Minimierung der Fahrzeit zur Ladestation erreicht werden kann, maximal $0,38\%$ pro Ladezyklus. Da außerdem davon ausgegangen werden kann, dass die Ladestationen so verteilt sein werden, dass in der Nähe jeder Put-Station eine Ladestation zur Verfügung stehen wird, ist das Optimierungspotential durch Berücksichtigung der Ladestände bei diesem Projekt so gering, dass es vernachlässigt werden kann.

Durch die Maßnahme in *b)* soll verhindert werden, dass Aufträge zuerst Robotern zugewiesen werden, die diese dann jedoch aufgrund zu geringen Ladestands doch nicht ausführen können. Werden diese dann erneut einem unterschiedlichem Roboter zugewiesen, kann es sein, dass dieser den Auftrag nicht mehr ohne Verletzung der Zeitfenster durchführen kann. Um zu verhindern, dass ab einem gewissen Ladestand einem Roboter noch ein Auftrag zugewiesen wird, muss der durchschnittliche Stromverbrauch für einen Auftrag auf dem zu untersuchenden Layout ermittelt werden. Erreicht der Roboter einen Ladestand, bei dem er während des nächsten zugewiesenen Auftrags die Ladestands-Grenze erreichen würde, meldet er sich von der Auftragsverteilung ab (siehe Abschnitt 8.1). So wird rechtzeitig verhindert, dass ein Auftrag an einen Roboter vergeben wird, der diesen dann doch nicht ausführen kann.

9 Simulation der Auftragsverteilung

Das folgende Kapitel ist der Simulation der neuen Auftragsverteilung gewidmet, wobei in Abschnitt 9.1 dargelegt wird, welche Maßnahmen zur Durchführung der Simulation notwendig sind und unter welchen Bedingungen sie durchgeführt wird. In Abschnitt 9.2 werden danach Ergebnisse der Simulation der neuen Auftragsverteilung präsentiert, und es wird ein Vergleich mit dem Algorithmus, der bisher im Einsatz war, durchgeführt.

9.1 Vorbereitung

Ziel der Simulation ist es vor allem, durch den direkten Vergleich der herkömmlichen mit der neuen Auftragsverteilung die Verbesserung des Durchsatzes, die durch die neue Auftragsverteilung erzielt werden konnte, zu messen und zu belegen. Dieser Vergleich wird unter den Bedingungen des in Abschnitt 2.5 beschriebenen Projekts Lebensmitteldistributionszentrum durchgeführt, welche durch die Anzahl an eingesetzten Robotern, dem Layout und der Auftragsstruktur charakterisiert werden. Wie diese Bedingungen in der Simulation umgesetzt wurden, wird in den folgenden Abschnitten beschrieben.

9.1.1 Simulation der Roboter

Wie bereits in Abschnitt 3.5 erfolgt die Simulation eines Roboters mithilfe des Pakets *Stage*, welches ausschließlich Sensordaten generiert. Das System, das den simulierten Roboter steuert, ist identisch mit dem System eines realen Roboters und reagiert exakt gleich auf die von *Stage* erzeugten Sensordaten wie ein realer Roboter auf reale Daten reagieren würde. Dies hat zum Vorteil, dass die tatsächliche Funktionsweise und Leistung eines solchen Transportsystems schon vor dem Bau eines Lagers untersucht werden kann. Sollten bei einer solchen Untersuchung Situationen auftreten, bei denen das Verhalten der Roboter durch bestimmte bauliche Gegebenheiten beeinträchtigt wird, können schon im Voraus Maßnahmen zur Änderung dieser baulichen Gegebenheiten ergriffen werden.

Eine solche Simulation hat aber auch einen großen Nachteil: Bei der Ausführung des gesamten Systems für jeden simulierten Roboter entsteht ein großer Bedarf an Rechenleistung. Auf einem realen Roboter befindet sich ein Industrierechner, der ausschließlich für das System dieses Roboters benutzt wird und deshalb über ausreichend Rechenkapazität verfügt, um alle notwendigen Prozesse flüssig auszuführen. Bei voller Kapazitätsauslastung eines Rechners für die Simulation ist es möglich, maximal drei solche Systeme auf einem Rechner parallel auszuführen. Außerdem kann aus demselben Grund in der

Simulation auch die Zeit nicht beschleunigt werden, da schon bei einer Durchführung in realer Zeit die Kapazität des Rechners voll benützt wird.

Im beschriebenen Projekt ist der Einsatz von 15 bis 17 Robotern geplant, was also einem Minimum von sechs Rechnern entspricht. Ebenso muss beachtet werden, dass auch das zentrale Flottenmanagementsystem, über welches unter anderem die Auftragsverteilung läuft, Rechenkapazitäten in Anspruch nimmt. Für die Simulation wurden sechs Server zur Verfügung gestellt, wobei es sich um Lenovo ThinkServer mit einem Prozessor aus der Intel E3-1200 Serie handelt. Bei Versuchen hat sich herausgestellt, dass es auf diesen sechs Servern möglich ist, maximal 17 Roboter zu simulieren, indem auf fünf Servern jeweils drei Roboter-Systeme und auf einem Server zwei Roboter-Systeme und das Flottenmanagementsystem ausgeführt werden.

Das parallele Ausführen von drei Roboter-Systemen auf einem Server führt jedoch auch zu zwei Problemen unterschiedlicher Art: Einerseits kann die hohe Auslastung der Server-Rechenkapazität bewirken, dass die Roboter erst mit Verzögerung auf ihre Sensor-Daten reagieren. Dies kann dazu führen, dass zwei Roboter zusammenstoßen und dann aufgrund von Schutzmechanismen nicht mehr in der Lage sind, sich voneinander zu entfernen. Das andere Problem hat seine Ursache in jenem Bereich des Robotersystems, welches ein neues Ziel auf der Karte in Fahrbefehle umwandelt. Ist das System überlastet, tritt hier ein Fehler auf, durch den ein zufälliges Ziel angefahren wird, welches nicht auf der Karte existiert. Passiert dies, so bleibt das Programm des Roboters an der aktuellen Stelle hängen und die Auftragsdurchführung wird nicht fortgesetzt. Dieses Problem wird jedoch weder durch die Auftragsverteilung noch die Simulation verursacht, sondern ist tief in einem grundlegenden Element des *Robot Operating System* verankert. Aus diesem Grund konnte es im Rahmen dieser Masterarbeit nicht behoben werden.

Um trotz dieser Probleme Ergebnisse anhand von Simulationsläufen zu erhalten, muss das Verhalten der Roboter während der Simulation durchgehend beobachtet werden. Da gegenseitige Behinderung und ein zögerliches Ausweichverhalten auch an realen Roboter beobachtet werden kann, wird beim Auftreten solcher Verzögerungen nicht in die Simulation eingegriffen. Tritt jedoch einer der eben beschriebenen Fehler aufgrund mangelnder Rechenkapazität auf, so bedeutet dies eine deutliche Veränderung in der Leistung des Transportsystems, welche jedoch nicht mit der Auftragsverteilung zusammenhängt. Um bei der Simulation jedoch die Leistung der neuen Auftragsverteilung messen und vergleichen zu können, wird bei einem Problem dieser Art sofort über die Steuerungsoberfläche eingegriffen, und das Problem wird so schnell behoben, dass es keinen Einfluss auf den Durchsatz des Transportsystems hat.

Da diese Probleme umso häufiger auftreten, je länger ein Simulationslauf bereits andauert, wird immer nur so lange simuliert, wie dies ohne gröbere Behinderungen möglich ist. Die Simulationsläufe, deren Ergebnisse in Abschnitt 9.2 präsentiert werden, basieren jeweils auf 100 bis 200 durchgeführten Aufträgen, was je nach der Zahl der eingesetzten Roboter zwischen 15 und 45 Minuten in Anspruch nimmt.

9.1.2 Layout

Das Layout des Distributionszentrums wurde für die Simulation als Karte im *incubed-Map-Editor* erstellt. Wie in Abschnitt 4.2 beschrieben, ist jede logistische Einheit als Cluster definiert und mit einer Zone und einer repräsentativen Position versehen. Außerdem wurde um jede Übergabestation eine *Single-Robot-Area* eingezeichnet. Diese erlaubt es einem Roboter nur dann in die Zone zu fahren, wenn sich dort kein anderer Roboter befindet. So kann verhindert werden, dass sich zwei Roboter beim Wechsel an einer Station durch den dort oft herrschenden Platzmangel gegenseitig blockieren.

Abbildung 9 gibt die erstellte Karte des Lebensmitteldistributionszentrum wieder. Blaue Stationen stellen eine Get-Station dar, grüne Stationen eine Put-Station und gelbe Stationen eine Ladestation. Es gibt auch zwei Arten von Warte-Positionen: Positionen mit rotem Rand sind *Idle*-Positionen, zu denen ein Roboter fährt, wenn er keinen Auftrag mehr zugewiesen bekommt. Positionen mit orangem Rand sind *Queue*-Positionen, zu denen ein Roboter fährt, wenn die Station, die er aufsuchen will, bereits durch einen andern Roboter belegt ist.

9.1.3 Aufträge

Für die Simulation des Lebensmitteldistributionszentrum wurden Transportaufträge so generiert, dass die Anforderungen, die in der Simulation an das Transportsystem gestellt werden, mit den Anforderungen an das zukünftige System vergleichbar sind. Die in Tabelle 3 angegeben Daten beschreiben den zukünftigen durchschnittlichen Paletten-Durchsatz zwischen den logistischen Einheiten des Zentrums.

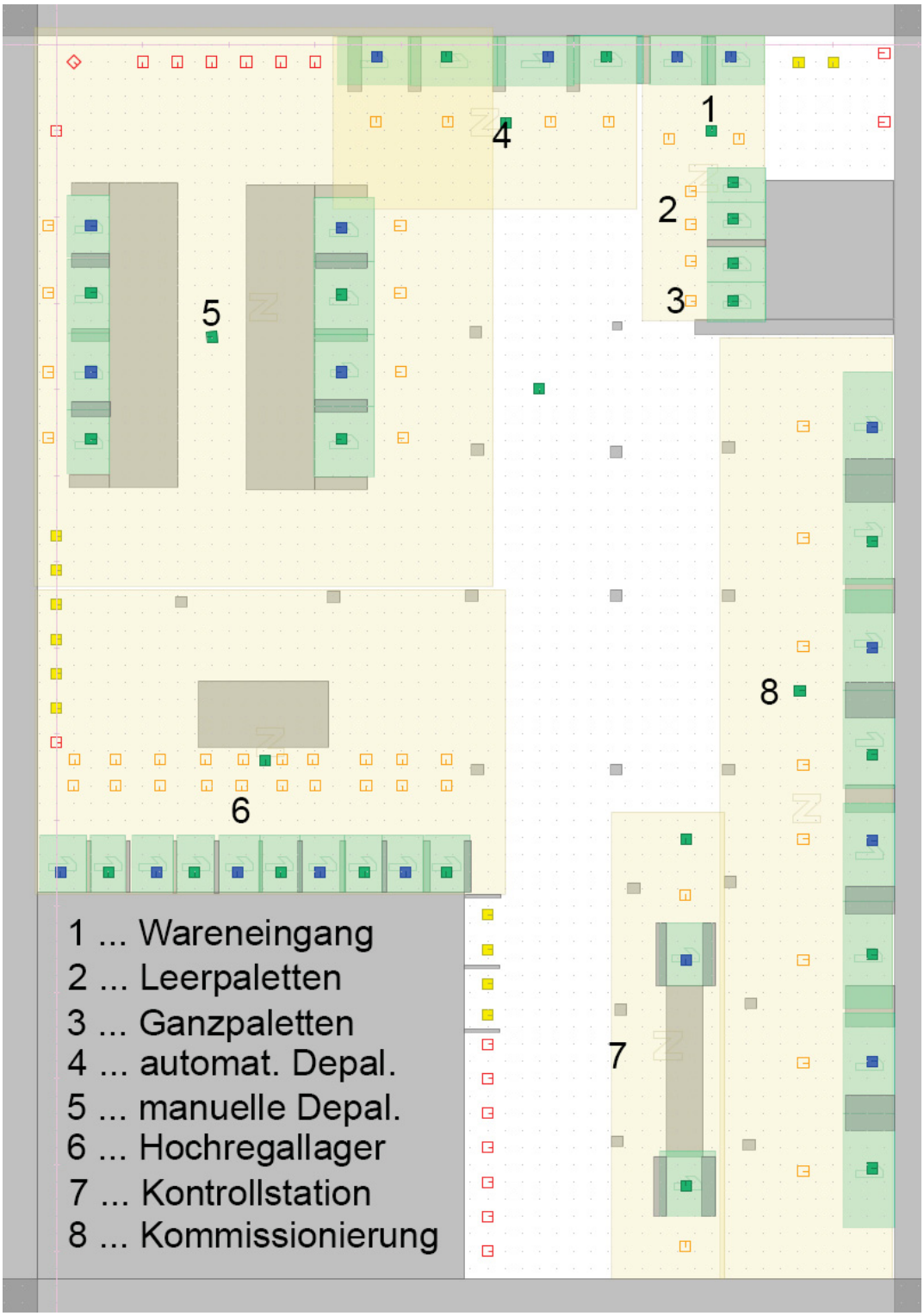

Abbildung 9: Layout

Tabelle 3: Durchschnittlicher Durchsatz zwischen den logistischen Einheiten

Von	**Nach**	**⌀ Paletten / Stunde**
Wareneingang	Hochregallager	72, 3
Hochregallager	Depalettierung	103, 0
Depalettierung	Hochregallager	67, 9
Depalettierung	Leerpaletten	35, 1
Hochregallager	Kommissionierung	67, 0
Kommissionierung	Hochregallager	42, 4
Kommissionierung	Leerpaletten	24, 7

Diese Durchsätze beziehen sich auf die Transporte zwischen logistischen Einheiten, welche bei der Planung des Distributionszentrums je nach gefordertem Durchsatz mit einer unterschiedlichen Anzahl an Übergabe-Stationen versehen wurden. Im realen Betrieb übergibt das übergeordnete System Aufträge, welche bereits eine Übergabe-Station als Quelle und als Ziel beinhalten, an das Flottenmanagementsystem.

Für das Generieren von Transportaufträgen für die Simulation wurde ein eigenständiges Programm entwickelt. Die Anzahl an Aufträgen, die generiert werden, hängt von einem Eingangsparameter ab. Dieser beschreibt die Dauer, die die Simulation für die Durchführung der Aufträge benötigt, falls der Durchsatz des simulierten Systems dem geforderten durchschnittlichen Durchsatz entspricht. Für jede Relation zwischen zwei logistischen Einheiten wird die Anzahl an Transporten pro Stunde dann auf die angegebene Dauer umgerechnet. Dies ergibt die Anzahl an Aufträgen, die für diese Relation generiert werden müssen. Dabei wird darauf geachtet, dass die Verteilung auf die Stationen einer logistischen Einheit (z.B. die Gassen des Hochregallagers) möglichst regelmäßig erfolgt. Berücksichtigt man alle möglichen Relationen zwischen logistischen Einheiten, so gibt es insgesamt 150 unterschiedliche Strecken, die zwischen den Get- und Put-Stationen dieser logistischen Einheiten zurückgelegt werden können.

Die Transporte zur Kontrollstation erfolgen nicht nach einer festen Relation, sondern sind immer dann notwendig, wenn eine Palette beschädigt oder nicht eindeutig identifizierbar ist. Dies wird beim Generieren berücksichtigt, indem für jede Relation eine Zuverlässigkeit von 98% angenommen wird und deshalb pro 100 Aufträge einer Relation zwei Aufträge hinzugefügt werden, wovon der erste Auftrag von der Quelle der Relation zur Kontrollstation und der zweite Auftrag von der Kontrollstation zum Ziel der Relation führt.

9.1.4 Zeitmessung

Um mithilfe der Simulation die neue Auftragsverteilung testen zu können, musste erst eine Datenbasis geschaffen werden, anhand welcher die Ermittlung der durchschnittlichen Zeiten für die Auftragsverteilung erfolgen konnte. Dazu wurde, mithilfe des zuvor beschriebenen Programms zur Auftragsgenerierung, für jede der 150 möglichen Get-Put-Relationen ein Auftrag erstellt und mit der herkömmlichen Auftragsverteilung in der Simulation von fünf Robotern durchgeführt. Auf Basis dieser Daten wurden dann Simulationsläufe mit 14 und 17 Robotern durchgeführt, bei denen jeweils 1500 Aufträge abgeschlossen und zeitlich gemessen wurden. Auf diesen Datensätzen basiert die Zeitschätzung, die in der Auftragsverteilung der anschließenden Simulationsläufe eingesetzt wurde.

9.2 Ergebnisse

Schließlich sollen natürlich die Ergebnisse präsentiert werden, die durch die neue Auftragsverteilung erzielt werden konnten, und ein Vergleich mit der herkömmlichen Auftragsverteilung angestellt werden. Zur Auswertung der Simulationsläufe wird auf die selbe Datenstruktur zurückgegriffen, die auch in der in Abschnitt 7.2 beschriebenen Durchschnittszeitberechnung verwendet wird. Es wird für jeden Auftrag, der in einem Simulationslauf von einem beliebigen Roboter durchgeführt wurde, die benötigte Zeit für diesen Auftrag bestimmt. Diese Dauer beginnt mit dem Zeitpunkt, zu dem die Cluster-Region reserviert wurde, und endet mit dem Zeitpunkt, an dem die Übergabe an der Put-Station beendet wurde. Danach wird die durchschnittliche Durchführungszeit eines Auftrags über alle Aufträge ermittelt. Dieser Wert wird als Maß für den Durchsatz verwendet, über den das Transportsystem bei einer bestimmten Auftragsverteilung mit einer bestimmten Anzahl an eingesetzten Robotern verfügt.

9.2.1 Wahl der Puffergröße

Wie in Abschnitt 7.3 beschrieben, spielt die richtige Parametrisierung der Puffer-Größe in der Stationsverwaltung eine wichtige Rolle. Sie bestimmt, um wie viel eine Stationsreservierung länger ist als die tatsächlich geschätzte Zeit eines Roboter an einer Station. Bevor also die Leistung der neuen Auftragsverteilung mit der des herkömmlichen Algorithmus verglichen werden kann, muss eine passende Einstellung für diesen Parameter gefunden werden.

Dazu wurden sechs Simulationsläufe mit der neuen Auftragsverteilung und einer Puffergröße von $8, 18, 28, 38, 48$ und 58 Sekunden durchgeführt. Dabei waren 17 Roboter an der Auftragsdurchführung beteiligt, und es wurden 150 Aufträge durchgeführt. Bei dieser Simulation wurden noch mehr Aufträge zur Verteilung übergeben, die Auftragsdurchführung wurde jedoch nach 150 Aufträgen abgebrochen.

Abbildung 10 stellt das Ergebnis dieser Simulationsläufe dar. Dabei wird in Blau die durchschnittliche Durchführungszeit jedes Simulationslaufs dargestellt, und in Orange die durchschnittliche Zeit, die ein Roboter warten musste, weil eine Station durch einen anderen Roboter besetzt war. Es ist zu erkennen, dass die durchschnittliche Wartezeit mit steigender Puffergröße sinkt. Dies hängt damit zusammen, dass es aufgrund der Abweichungen von der geschätzten Fahrzeit eines Roboters trotz Belegungsverwaltung zu Situationen kommt, in denen ein Roboter an eine Station fahren möchte, die bereits durch einen anderen Roboter belegt ist. Je größer der Puffer bei der Stationsreservierung gewählt wird, desto besser können die Abweichungen der geschätzten Fahrzeit ausgeglichen werden. Es gibt jedoch ein Minimum, unter welches die Wartezeit in diesem Simulationsszenario nicht gesenkt werden kann, da bei dieser großen Anzahl an Robotern und einer beschränkten Auswahl an möglichen Aufträgen ein durch den Verteilungsalgorithmus bewusst gewähltes Warten an einer Station nicht verhindert werden kann. Aus diesem Grund sinkt die durchschnittliche Wartezeit bei steigender Pufferzeit zu Beginn sehr stark und gegen Ende nur mehr gering.

Betrachtet man in diesem Vergleich den Verlauf der durchschnittlichen Durchführungszeit, so erkennt man, dass sie mit steigender Pufferzeit zuerst abnimmt und ab einer Pufferzeit von 38 Sekunden wieder zunimmt. Zu Beginn nimmt die durchschnittliche Durchführungszeit um den selben Wert ab wie die Wartezeit, was darauf hinweist, dass die Reduktion der Wartezeit an den Stationen zu 100% zur Reduktion der Durchführungszeit beiträgt. Mit steigender Pufferlänge nimmt jedoch der Einfluss der sinkenden Wartezeit auf die Durchführungszeit ab, und ab einem gewissen Punkt nimmt diese sogar wieder zu. Dies kann dadurch erklärt werden, dass bei steigender Pufferzeit der Auftragsverteilungsalgorithmus durch die steigende Wartezeit an den Stationen eher Aufträge verteilt, deren Stationen noch gar nicht durch andere Roboter reserviert wurden. Dies ist jedoch meist nur durch das in Kauf Nehmen von längeren Fahrwegen möglich. Der Vergleich zeigt, dass je stärker versucht wird, das Aufeinandertreffen von zwei Robotern an einer Station zu vermeiden, umso länger werden die gewählten Fahrwege und damit ab einem gewissen Punkt auch die durchschnittlichen Durchführungszeiten.

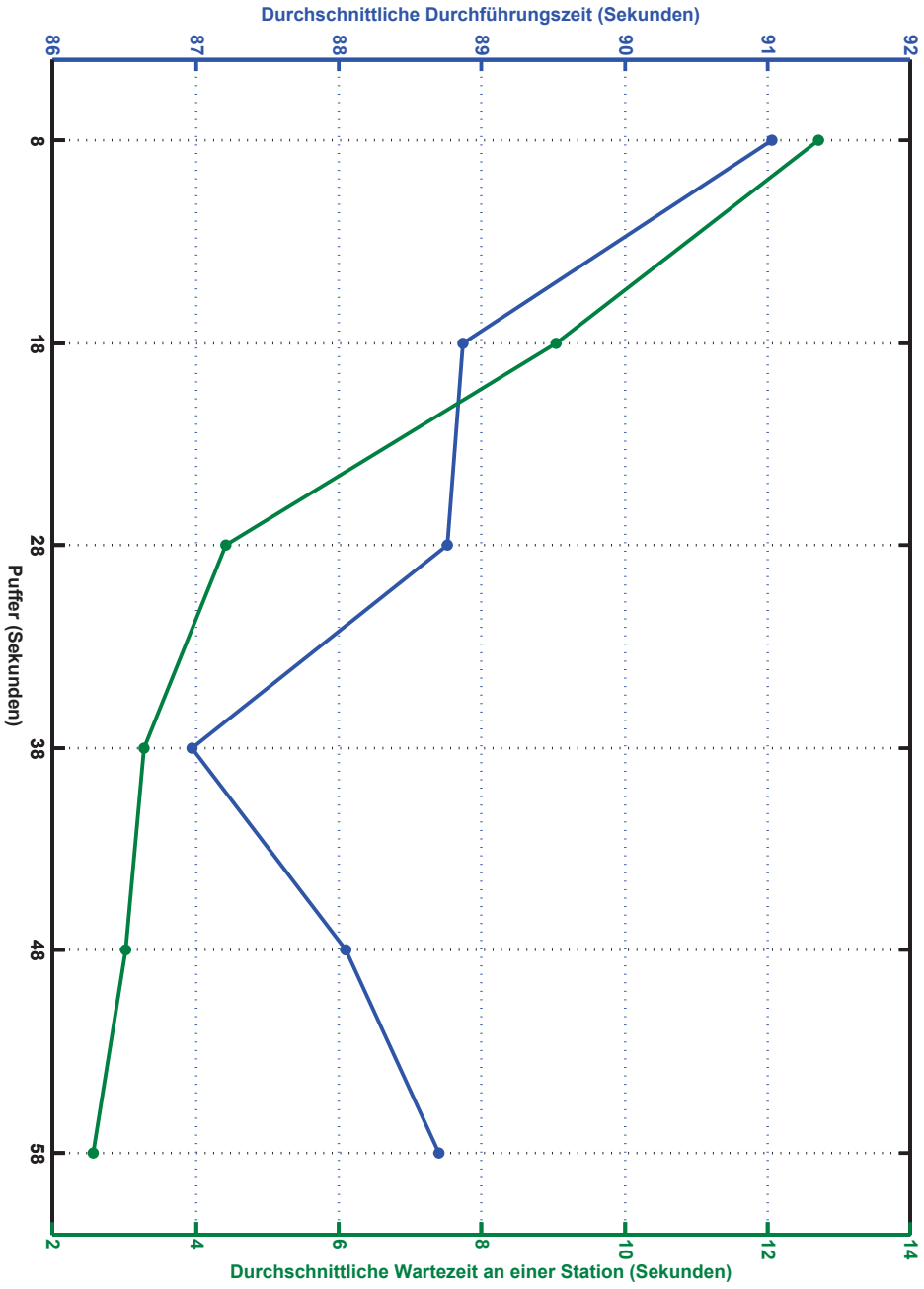

Abbildung 10: Vergleich unterschiedlicher Puffergrößen

9.2.2 Vergleich des Durchsatzes beider Auftragsverteilungen

Nachdem eine für die vorliegende Simulation passende Parametrisierung der Puffergröße gefunden wurde, konnte ein Vergleich der neuen Auftragsverteilung mit der herkömmlichen durchgeführt werden. Dazu wurde mit den selben zur Verfügung stehenden Aufträgen acht Simulationsläufe durchgeführt, wobei bei einer Anzahl an eingesetzten Robotern von $11, 13, 15$ und 17 jeweils beide Auftragsverteilungen simuliert wurden.

Beim Beobachten der Simulation mit der herkömmlichen Auftragsverteilung fällt vor allem auf, dass es häufig dazu kommt, dass ein Roboter und manchmal sogar zwei auf das Freiwerden einer Put-Stationen warten. Dies hängt damit zusammen, dass der in Abschnitt 5.1.2 beschriebene Algorithmus bei den Aufträgen nur auf eine regelmäßige Auslastung der Get-Stationen achtet, die Put-Station des Auftrags wird dabei aber nicht berücksichtigt. Der Durchsatz des Transportsystems wird dadurch nicht nur aufgrund der unproduktiven Zeit der wartenden Roboter gesenkt, sondern diese behindern oft auch andere Roboter, die an der besetzten Station nur vorbei fahren möchten.

Abbildung 11 stellt die durchschnittliche Durchführungszeit aller Simulationsläufe dar und die Differenz zwischen der durchschnittlichen Durchführungszeiten der beiden Auftragsverteilungen, bei gleicher Anzahl an eingesetzten Robotern. Zunächst ist zu erkennen, dass bei jeder der untersuchten Mengen an eingesetzten Robotern die neue Auftragsverteilung eine kürzere durchschnittliche Durchführungszeit erzielt – also einen höheren Durchsatz aufweist. Außerdem ist festzustellen, dass bei beiden Auftragsverteilungen mit steigender Anzahl an Robotern auch die durchschnittliche Durchführungszeit zunimmt. Dies ist darauf zurückzuführen, dass beim Einsatz von mehr Robotern diese auf der Fahrt zwischen zwei Stationen öfter auf andere Roboter treffen, wodurch ihre Fahrt stärker verzögert wird. An der Differenz zwischen den durchschnittlichen Durchführungszeiten ist jedoch erkennbar, dass bei steigender Anzahl an eingesetzten Robotern die durchschnittliche Durchführungszeit der herkömmlichen Auftragsverteilung stärker steigt als die der neuen. Die Erklärung dafür liefert die Tatsache, dass bei der herkömmlichen Auftragsverteilung nicht auf die gleichmäßige Verteilung der Roboter auf alle Stationen geachtet wird. Je mehr Roboter auf dem selben Layout eingesetzt werden, umso wahrscheinlicher ist es, dass zwei Roboter zur selben Zeit zur selben Station möchten und so unnötige Wartezeit entsteht. Bei 17 eingesetzten Robotern beträgt die durchschnittliche Zeit pro Auftrag, die ein Roboter mit der herkömmlichen Auftragsverteilung in der beschriebenen Simulation auf das Freiwerden einer Station warten muss, $19,5$ Sekunden, also fast 20% der durchschnittlichen Durchführungszeit.

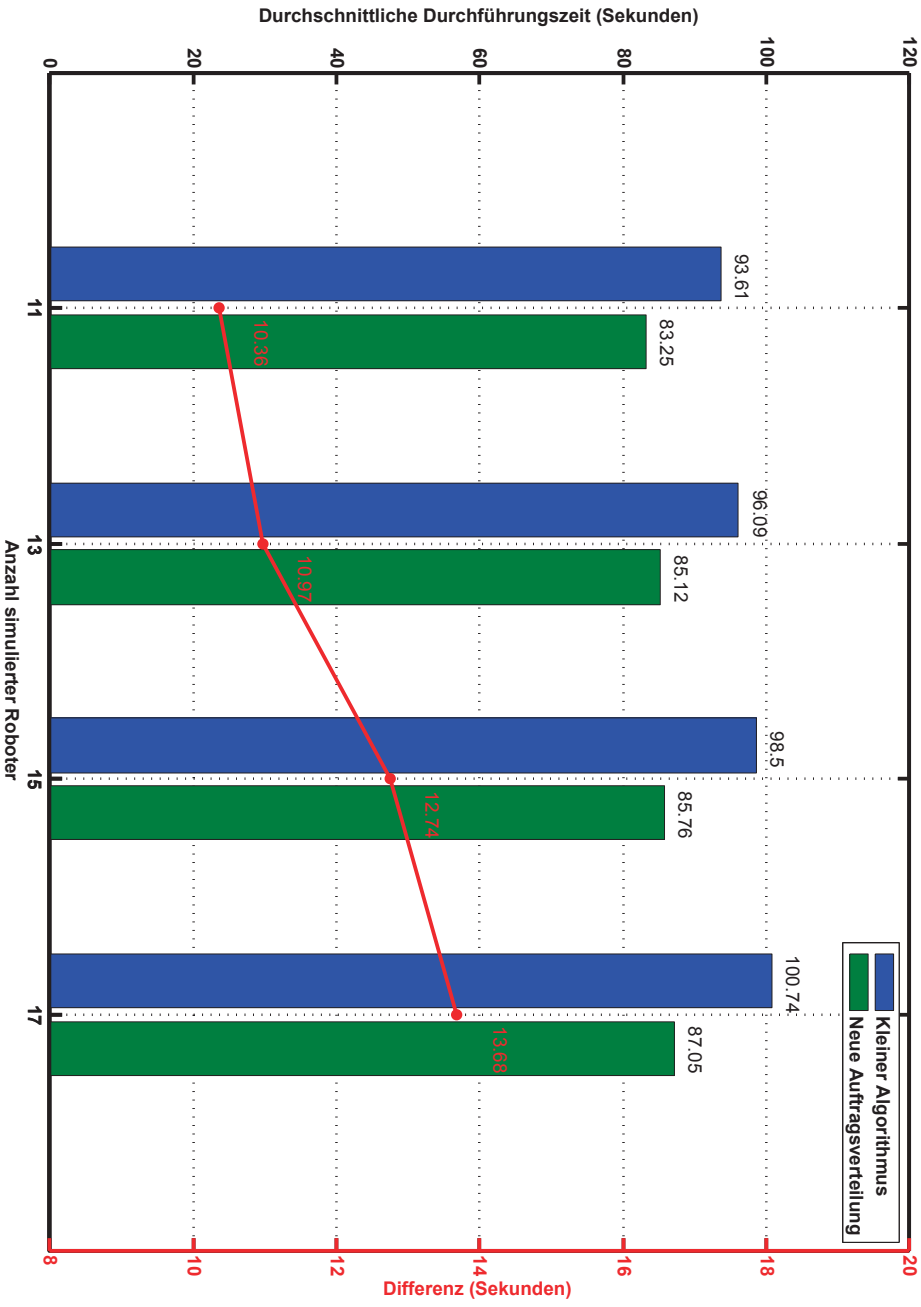

Abbildung 11: Vergleich des Durchsatzes beider Auftragsverteilungen

9.2.3 Vergleich der benötigte Zeit beider Auftragsverteilungen

Bei den zuvor beschriebenen Vergleichen ist den Auftragsverteilungsalgorithmen jedoch eine große Anzahl an Aufträgen zur Verfügung gestanden, aus denen sie dann immer die für die aktuelle Situation passenden Aufträge wählen konnten. Dies hat ermöglicht, eine Aussage über den maximalen Durchsatz der beiden Auftragsverteilungen zu treffen. Da diese Bedingungen im realen Umfeld eines Distributionszentrums jedoch nicht gegeben sind, da die Auftragsverteilungsalgorithmen nur aus einer begrenzten Anzahl an zur Verfügung stehenden Aufträgen auswählen können, wurde noch ein weiterer Vergleich zwischen den beiden Auftragsverteilungen durchgeführt.

Bei diesem Vergleich wurden in zwei Simulationsläufen die selben 100 Aufträge mit beiden Auftragsverteilungen durchgeführt. Dies entspricht dem Auftragsvolumen, welches im Distributionszentrum bei durchschnittlicher Auslastung in $14,4$ Minuten anfällt und annähernd auch der Zeit, die Aufträge im Distributionszentrum voraussichtlich im Voraus bekannt gegeben werden.

Vergleicht man nun die Zeit, die bei 17 eingesetzten Robotern die beiden Auftragsverteilungen für die Durchführung aller 100 Aufträge benötigen, so ist die neue mit 11 Minuten und 7 Sekunden um 47 Sekunden schneller als die herkömmliche Auftragsverteilung, die 11 Minuten und 54 Sekunden für die Bewältigung der 100 Aufträge benötigt hat. Das bedeutet eine Reduktion der benötigten Zeit um $6,57\%$. Da bei 17 Robotern ein Roboter $5,88\%$ des Durchsatzes darstellt, sollte durch die Reduktion der Durchführungszeit um $6,57\%$ ein Roboter eingespart werden können. Dies wurde in einem weiteren Simulationslauf untersucht, bei dem nur 16 Roboter mit der neuen Auftragsverteilung die selben 100 Aufträge durchzuführen hatten. Dazu benötigten sie 11 Minuten und 37 Sekunden, wodurch die Möglichkeit der Einsparung eines Roboters in diesem Fall bestätigt wurde.

Abbildung 12 stellt die Durchführungszeiten der 100 Aufträge in der Reihenfolge dar, in der sie von den 17 Robotern abgeschlossen wurden. Außerdem wird der Durchschnittswert über alle Durchführungszeiten einer Auftragsverteilung dargestellt. Der Durchschnittswert der Durchführungszeiten liegt bei herkömmlichen Verteilung bei $99,98$ Sekunden und unterscheidet sich damit kaum vom Durchschnittswert, der bei der gleichen Anzahl an Robotern, jedoch mit einer großen Auswahl an zu verteilenden Aufträgen erzielt wurde (siehe Abschnitt 9.2.2). Bei der herkömmlichen Auftragsverteilung macht die Größe der Auswahl an zu verteilenden Aufträgen also keinen wesentlichen Unterschied aus. Der Durchschnittswert der Durchführungszeit der von der neuen Auftragsverteilung durchgeführten Aufträge beträgt $92,53$ Sekunden, also um $5,48$ Sekunden mehr als bei

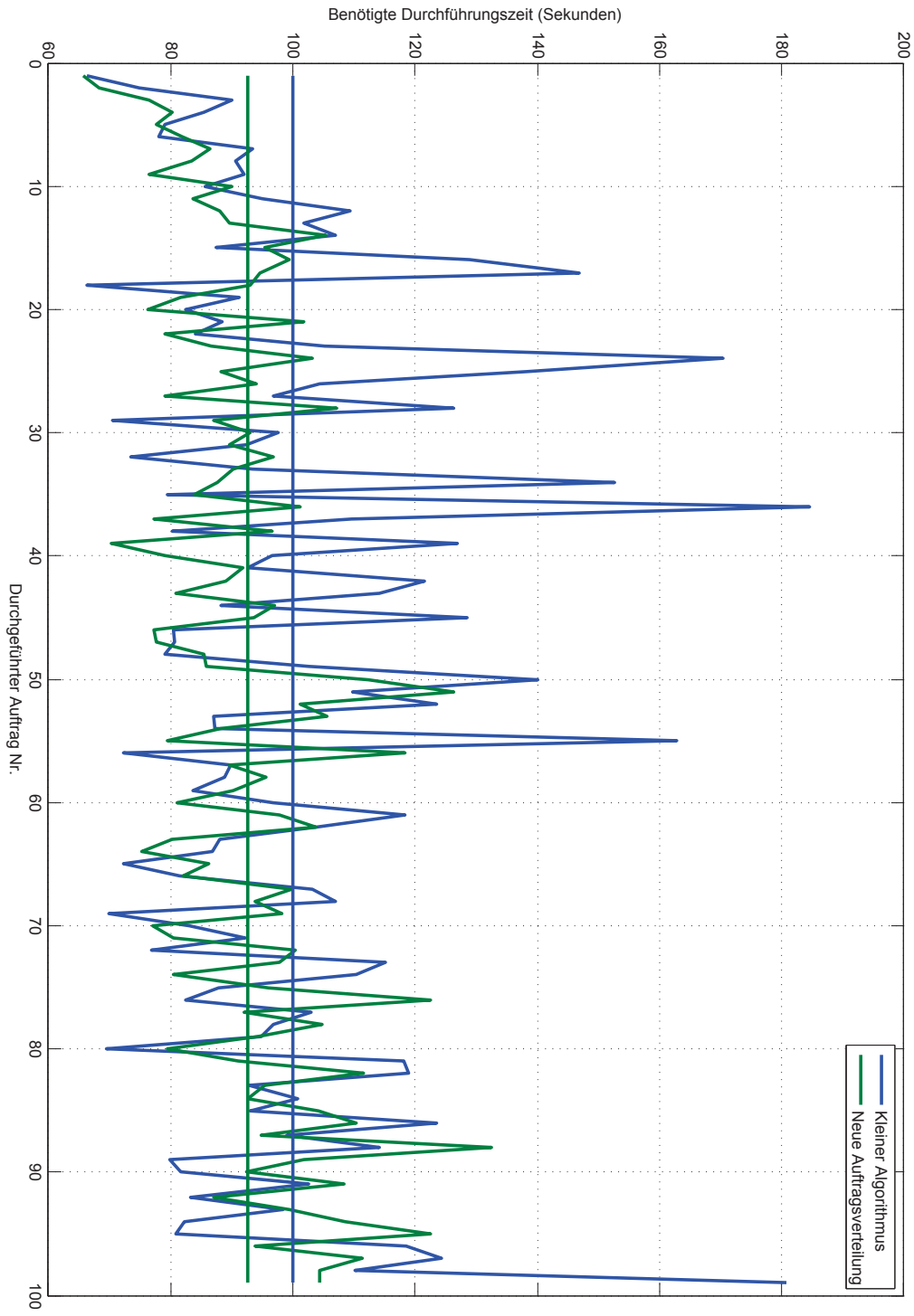

Abbildung 12: Vergleich der Durchführungszeiten von 100 Aufträgen

dem Simulationslauf, in dem mit der gleichen Anzahl an Robotern die neue Auftragsverteilung eine große Auswahl an zu verteilenden Aufträgen zur Verfügung hatte. Damit ist die durchschnittliche Durchführungszeit der neuen Auftragsverteilung zwar immer noch um $7,45\%$ geringer als bei der herkömmlichen Auftragsverteilung, diese Reduktion ist in diesem Fall jedoch um $6,14\%$ geringer als bei einer Auftragsverteilung mit einer unbegrenzten Menge an Aufträgen.

Die Erklärung dafür ist, dass bei diesem Vergleich alle zur Verfügung stehenden Aufträge durchgeführt werden müssen. So stehen gegen Ende der Durchführung immer weniger Aufträge für die Verteilung zur Verfügung, und es kann nicht verhindert werden, dass die neue Auftragsverteilung Aufträge vergibt, die die selben Stationen beanspruchen und so Wartezeiten entstehen. Im realen Betrieb des Distributionszentrums wird jedoch selten der Fall eintreten, dass eine geringe Anzahl an Aufträgen gänzlich durchgeführt werden muss, ohne dass bereits andere Aufträge zur Verfügung gestellt worden sind. Die meiste Zeit werden regelmäßig neue Aufträge dem System übergeben, und die neue Auftragsverteilung hat so die Möglichkeit, einem Roboter einen Auftrag zuzuweisen, bei dem keine Wartezeiten an Stationen entstehen.

Der Durchsatz, den das Transportsystem mit der neuen Auftragsverteilung im zu untersuchenden Umfeld haben wird, kann jedoch noch nicht genau bestimmt werden. Dies liegt daran, dass die genaue Auftragsstruktur noch nicht festgelegt wurde. Abhängig von den Bedingungen, die vom übergeordneten Warehouse-Management-System an das Transportsystem bezüglich der Auftragsdurchführung gestellt werden, steht der neuen Auftragsverteilung mehr oder weniger Freiheit zu, Aufträge so zu verteilen, dass der Durchsatz des gesamten Systems maximiert wird. Auf Basis der Vergleiche, die mit einem geringen und einem hohen Maß an Einschränkung durchgeführt wurden, kann jedoch angenommen werden, dass im realen Betrieb des Distributionszentrums die durchschnittliche Durchführungszeit bei 17 Robotern mit der neuen Auftragsverteilung bei einem Wert zwischen $87,05$ und $92,53$ Sekunden liegen wird.

10 Fazit

Die wesentlichen Erkenntnisse, zu denen die Arbeit an diesem Projekt geführt hat, werden nun in einer abschließenden Betrachtung zusammenfassend dargestellt. Vorbereitend für die Untersuchungen zu den zentralen Punkten unseres Themas wurden zwei dafür wesentliche Voraussetzungen geschaffen: die Einführung von Batterieladeständen in der Simulation (Kapitel 3) und eine Auftragsvergabe, die in zwei Stufen erfolgt (Kapitel 4).

Für die Einführung von Batterieladeständen wurde zunächst bei realen Robotern der Stromverbrauch und die Geschwindigkeit in mehreren kompletten Entladungen aufgezeichnet. Durch den Einsatz des Verfahrens der multiplen linearen Regression wurden die Koeffizienten ermittelt, die den Zusammenhang zwischen Geschwindigkeit, Beschleunigung und Stromverbrauch am besten beschreiben. Anhand dieser Koeffizienten kann der aktuelle Stromverbrauch eines Roboters zu jedem Zeitpunkt geschätzt werden. Die Simulation wurde dann um ein Element erweitert, welches den Ladestand jedes Roboters verwaltet und seine Höhe jede Sekunde neu berechnet, je nachdem ob der Roboter Aufträge ausführt oder an einer Ladestation steht.

Als zweite Voraussetzung musste eine zweistufige Auftragsverteilung implementiert werden, da durch sie eine Erhöhung des Optimierungspotentials möglich ist. Es wurden die Finite-State-Machine, die den Roboter steuert, und die Auftragsverteilung angepasst. Dadurch wird einem Roboter ein Auftrags nicht sofort definitiv erteilt, sondern er fährt zunächst in die Region der ersten ihm zugewiesenen Station. Während der Fahrt kann der Roboter einen neuen Auftrag erhalten – das Ziel dieser Fahrt kann sich zu diesem Zeitpunkt noch ändern. Erst wenn er in der Region des tatsächlich erteilten Auftrags angekommen ist, wird der Auftrag fest zugewiesen.

Aufbauend darauf wurde mit der Entwicklung einer neuen Vorgangsweise bei der Auftragsverteilung begonnen (Kapitel 5 - Kapitel 6). Da die Zeit, die ein Roboter für die Durchführung eines bestimmten Auftrags benötigt, sich bei wiederholter Durchführung nur geringfügig ändert, wurde für die neue Auftragsverteilung die Dauer der Durchführung im voraus geschätzt und so eine zukunftsorientierte Verplanung der Aufträge ermöglicht. Für diese Schätzung wurde ein Auftrag in Teilabschnitte unterteilt und eine Zeitmessung implementiert, die die für jeden dieser Abschnitte benötigte Zeit aufzeichnet. Danach wurde eine Funktion implementiert, die aus der Menge aller aufgezeichneten Daten Durchschnittswerte berechnet, mit deren Hilfe dann für jeden möglichen Auftrag die Zeit, die ein Roboter für dessen Durchführung benötigt, berechnet werden kann.

Um darüber hinaus zu berücksichtigen, dass ein Roboter, falls eine Station bereits

durch einen anderen Roboter besetzt ist, auf das Freiwerden dieser Station warten muss, wurde eine Stationsverwaltung implementiert. Diese bewirkt, dass bereits bei der Auftragsplanung die entsprechende Station für den vorgesehenen Roboter zum geplanten Zeitpunkt reserviert und eine eventuelle Wartezeit berücksichtigt wird. Auf diese Weise gelingt es in der neuen Auftragsverteilung, eine gleichmäßige Auslastung der Stationen sicherzustellen. Durch einen Parameter kann die Dauer einer Reservierung größer ausgelegt werden als die Zeit, die für den Aufenthalt eines Roboters an einer Station eigentlich vorgesehen war – so können zeitliche Abweichungen der Roboter ausgeglichen werden.

Damit eine zukunftsorientierte Planung der Aufträge möglich ist, wurde auch eine Zeitevaluierung implementiert. Diese bestimmt einerseits, zu welchem Zeitpunkt jeder Roboter seinen aktuellen Auftrag fertigstellt, und andererseits, wie lange er für die Fertigstellung eines möglichen nächsten Auftrags benötigen würde. Anschließend wurde die tatsächliche Auftragsverteilung entwickelt, die anhand dieser Informationen jedem Roboter, zusätzlich zu seinem aktuellen Auftrag, einen weiteren Auftrag zuweist. Dieser wird mithilfe eines Greedy-Algorithmus jenem Roboter zugewiesen, welcher ihn mit geringster Leerzeit – also der Summe aus der Fahrzeit zur ersten Station und der Wartezeit an beiden Stationen – ausführen kann. Durch die Minimierung der Leerzeit kann der Durchsatz des gesamten Transportsystems maximiert werden.

Sinkt der Ladestand eines Roboters unter einen gewissen Wert, so wird dem Roboter kein weiterer Auftrag mehr zugewiesen und er fährt nach Abschluss seines aktuellen Auftrags zur Ladestation. Aufgrund des geringen Optimierungspotential wurde die Tatsache, dass ein Roboter nach einem Auftrag zur Ladestation fahren muss, in der Auftragsverteilung nicht berücksichtigt.

Schließlich wurde noch eine Auftragsverwaltung implementiert (Kapitel 8), welche die den Robotern zugewiesenen Aufträge speichert und auch überprüft, zu welchem Zeitpunkt eine Neuberechnung der Verteilung notwendig ist. Diese wird ausgelöst, wenn ein Roboter keinen Auftrag mehr zugewiesen hat oder wenn es bei der Ausführung eines Auftrags zu einer Abweichung von der geplanten Zeit kommt. Falls also ein Roboter – aufgrund seines nicht-deterministischen Fahrverhaltens – für eine Fahrt länger als gewöhnlich braucht, kann die Auftragsverwaltung diese Verzögerung bei einer Neuberechnung berücksichtigen und so sicherstellen, dass wichtige Aufträge an andere Roboter, die früher wieder verfügbar sind, umverteilt werden können.

Um eine Aussage über die Qualität der neuen Auftragsverteilung treffen zu können, wurden Simulationsläufe unter den Bedingungen des geplanten Distributionszentrums durchgeführt (Kapitel 9). Dabei wurde die neue Auftragsverteilung der herkömmlichen

gegenübergestellt. Die Anwendung der neuen Auftragsverteilung hat ergeben, dass der Parameter, mit dem die Reservierungen der Stationsverwaltung vergrößert werden können, einen starken Einfluss auf die Qualität der Ergebnisse hat. Erstellt man größere Einträge als die tatsächliche Zeit, die nach der Zeitschätzung der Roboter an der Station verbringen wird, so können zwar Abweichungen von diesen Schätzungen ausgeglichen werden, es werden jedoch auch längere Fahrwege zu nicht reservierten Stationen in Kauf genommen.

Nicht zuletzt wurde auch die Zahl der eingesetzten Roboter als Basis für eine Vergleichssimulation zwischen herkömmlicher und neuer Auftragsverteilung herangezogen. Auch dieses Ergebnis fiel zugunsten letzterer aus: Mit ihr kann die durchschnittliche Durchführungszeit um bis zu 13,6% gesenkt werden, und dieser Unterschied zum Vorteil der neuen Auftragsverteilung wird umso größer, je mehr Roboter zum Einsatz kommen. Schließlich hat die Simulation mit beiden Auftragsverteilungen bei einer begrenzten Anzahl an Aufträgen gezeigt, dass – mit 17 eingesetzten Robotern – bei gleicher Menge an Aufträgen die neue Auftragsverteilung die Einsparung eines Roboters ermöglicht. Diese Erhöhung des Durchsatzes kann vor allem auf die Reduktion der Wartezeit an den Stationen zurückgeführt werden, da die herkömmliche Auftragsverteilung bereits eine Minimierung der Anfahrtszeiten zur ersten Station eines Auftrags berücksichtigt.

Zusammenfassend kann also gesagt werden, dass – unter den derzeit bekannten Bedingungen des geplanten Distributionszentrums und im Vergleich zur herkömmlichen Auftragsverteilung – dank des Einsatzes dieser neuen Auftragsverteilung eine deutliche Erhöhung des Durchsatzes des Transportsystems erzielt werden kann. Zum Zeitpunkt der Simulation waren jedoch noch nicht genügend Details dieses Projekts festgelegt, um eine genaue Aussage über den Durchsatz treffen zu können, der bei einer bestimmten Anzahl an eingesetzten Robotern möglich ist. Zukünftige Arbeit an diesem Projekt wird zum Ziel haben, die aktuelle Simulation noch stärker an die tatsächlichen Bedingungen anzunähern und so schließlich eine präzise Aussage über die Anzahl an Robotern treffen zu können, die für die Bewältigung der Auftragslast in diesem Distributionszentrum notwendig sind.

Literatur

[1] F. Abrate, B. Bona, M. Indri, and L. Carlone. Cooperative robotic teams for supervision and management of large logistic spaces: Methodology and applications. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2010.

[2] Gerardo Berbegliaa, Jean-François Cordeaub, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202:8–15, 2010.

[3] J. Bruno, E.G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17, 1974.

[4] Peter J. Cameron. *Combinatorics*. Safari-Verlag, Berlin, 1994.

[5] A. Caris and G.K. Janssens. A local search heuristic for the pre- and end-haulage of intermodal container terminals. *Computers & Operations Research*, 36:2763–2772, 2008.

[6] Ping Chen, Youli Qu, Houkuan Huang, and Xingye Dong. A new hybrid iterated local search for the open vehicle routing problem. *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, 1:891 – 895, 2008.

[7] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57:1472–1483, 2009.

[8] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*. Wiley, New York, 1982.

[9] Jianhua Gao, Xudong Hu, and Chuanyu Wu. Design and simulation of multi-robot logistic system. *Proceedings of the 2nd IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pages 1–6, 2006.

[10] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[11] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23, 2004.

[12] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623, 2005.

[13] A. Kelly, B. Nagy, D. Stager, and R. Unnikrishnan. Field and service applications - an infrastructure-free automated guided vehicle based on computer vision - an effort to make an industrial robot vehicle that can operate without supporting infrastructure. *Robotics and Automation Magazine, IEEE*, 14(3):24–34, 2007.

[14] Bernhard Korte and Jens Vygen. *Kombinatorische Optimierung: Theorie und Algorithmen*. Springer-Verlag, Berlin, Heidelberg, New-York, 2008.

[15] Doina Logofătu. *Grundlegende Algorithmen mit Java*. Springer-Verlag, Berlin, Heidelberg, New-York, 2014.

[16] Aaron Martinez and Enrique Fernández. *Learning ROS for Robotics Programming*. Packt Publishing, Birmingham, United Kingdom, 2013.

[17] Y. Mohan and S.G. Ponnambalam. An extensive review of research in swarm robotics. *Nature & Biologically Inspired Computing*, pages 140–145, 2009.

[18] L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, pages 220–240, 1998.

[19] C. Schindelhauer and G. Schomaker. Weighted distributed hash tables. *In 17th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 218–227, 2005.

[20] D. Sun, A. Kleiner, and C. Schindelhauer. Decentralized hash tables for mobile robot teams solving intra-logistics tasks. *9th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 923–930, 2010.

[21] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM monographs on discrete mathematics and applications, Philadelphia, Pa., 2002.

[22] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231:1–21, 2013.

Abbildungsverzeichnis

Abkürzungsverzeichnis

AGV	Automated Guided Vehicle
BLE	Broadcast of local eligibility
BMS	Batteriemanagementsystem
FMS	Flotten Management System
FSM	Finite State Machine
HRL	Hochregallager
KSI	KNAPP Systemintegration GmbH
MRTA	Multi-Robot Task Allocation
OSR	Open Storage and Retrieval
ROS	Robot Operating System