Fähigkeitsbasierte Kooperation von Heterogenen Robotersystemen

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Georg Heppner
aus Hamburg

Tag der mündlichen Prüfung: 06. Mai 2024

1. Referent: Prof. Dr.-Ing. Rüdiger Dillmann
2. Referent: Prof. Dr.-Ing. habil. Jürgen Beyerer

Danksagung

Als ich 2009 das erste Mal das FZI betrat, wollte ich eigentlich nur schauen, was es mit diesem Laufroboter LAURON auf sich hat. Roboter fand ich schon immer faszinierend, aber dass ich selbst einmal ganz neue Roboter von Grund auf entwickle, mit Flugrobotern Einsatzkräfte unterstütze oder aber Laufroboter dazu befähigen würde, fremde Planeten zu erforschen, hatte ich so nicht erwartet. Ich bin froh, dass es so gekommen ist, denn ich kann mir kaum eine bessere Betätigung vorstellen als so faszinierende Technik für die Zukunft zu gestalten.

Die vorliegende Arbeit ist in meiner Zeit als wissenschaftlicher Mitarbeiter und stellv. Abteilungsleiter der Abteilung Interaktive Diagnose- und Servicesysteme (IDS) am Forschungszentrum Informatik in Karlsruhe entstanden. Während der gesamten Zeit habe ich zahlreiche Themen bearbeitet und mich mit großartigen Personen, sowohl aus dem FZI als auch aus Forschungseinrichtungen auf der ganzen Welt, ausgetauscht. Die Erfahrungen und die spannenden Diskussionen haben alle dazu beigetragen, dass ich meine Arbeit erfolgreich abschließen konnte. Stellvertretend möchte ich einigen Personen besonders danken:

Herrn Prof. Dr.-Ing. Rüdiger Dillmann danke ich insbesondere für sein entgegengebrachtes Vertrauen und sein großes Interesse an meiner Arbeit, aber natürlich auch für die Übernahme des Hauptreferats und das ausgesprochen detaillierte Feedback zu meiner Ausarbeitung. Seine unerschütterlich positive Grundhaltung hat mich immer wieder motiviert, meine Ergebnisse in einem positiven Licht zu sehen. Für die freundliche Übernahme des Korreferates und die konstruktiven Gespräche danke ich Herrn Prof. Dr.-Ing. habil. Jürgen Beyerer ganz herzlich.

Die vorliegende Arbeit wäre ohne die tatkräftige Unterstützung von zahlreichen Studenten und Abschlussarbeiten nicht möglich gewesen. Mein besonderer Dank gilt Nils Berg, der mit mir nicht nur den Space Bot Cup gemeistert hat, sondern mit seiner Abschlussarbeit auch wichtige Grundlagen der Entwicklung der Behavior Trees umgesetzt hat und David Oberacker, der mit mir die Fähigkeiten in die Behavior Trees gebracht hat und weiterhin dafür sorgt, dass die Ergebnisse dieser Arbeit als Open-Source Bibliothek aktiv eingesetzt werden können. Darüber hinaus geht mein Dank an Kai Uwe Hermann, Carsten Plasberg, Christopher Wecht, Viktor Van Wetten, Anton Schirg, Friedolin Gröger und Michael Schäuble, die alle auf verschiedene Arten zu meiner Forschung beigetragen haben.

Meinen aktuellen und ehemaligen Kollegen bei IDS und TKS danke ich ganz herzlich für die großartige Zusammenarbeit, die angeregten Gespräche und vor

allem für die großartige Einstellung, dass Roboter das Coolste sind. Ich danke Tristan Schnell, Felix Exner, Lennart Puck, Timothee Büttner und Marvin Große Besselmann für die intensiven Diskussionen und die tolle Unterstützung bei zahlreichen Vorhaben, Lea Steffen für das Beantworten unzähliger Fragen zum Promotionsablauf, Andreas Hermann, Steffen Rühl und Marc Essinger für ihre Freundschaft und gute Stimmung und David Oberacker für die kontinuierliche Unterstützung mit den Behavior Trees sowie Genessis Perez, David Timmermann und Vincenzo Di Pentima für den regelmäßigen Austausch in Forschungstreffen.

Einen ganz besonderen Dank möchte ich meinem Freund Arne Rönnau aussprechen. Er hat mich ans FZI geholt, mir in unzähligen Projekten gezeigt, wie Forschung funktioniert und als Abteilungsleiter wirklich alles möglich gemacht, um meine Forschung zu unterstützen. Die zahlreichen gemeinsame Veröffentlichungen, tagelangen Diskussionen und sein hoher Anspruch haben mir wichtige Techniken für das wissenschaftliche Arbeiten vermittelt und die Qualität meiner Arbeiten kontinuierlich verbessert. Als gebürtige Nordlichter sind wir beide Freunde des direkten und ungefilterten Feedbacks, etwas, was ich von ihm über die Jahre immer wieder erhalten habe und wofür ich sehr dankbar bin, auch weil es ein Stück Heimat für mich ist.

Ganz besonders möchte ich auch meiner großartigen Frau Bianca danken. Sie hat mich während des kompletten Schreibens stets unermüdlich unterstützt und mir dabei nicht nur einmal das Imposter-Syndrom ausgeredet. Ohne ihre emotionale Unterstützung und die Betreuung unseres Kindes hätte ich diese Arbeit nicht fertigstellen können. Sie ist darüber hinaus der einzige Grund, dass in dieser Arbeit Kommas verwendet werden, da sie die Mammutaufgabe der Fehlerkorrektur für die komplette Arbeit übernommen hat, wofür ich extrem dankbar bin. John Freytag und Peter Thorben Pfeiffer danke ich, dass sie als meine besten Freunde stets ein offenes Ohr für mich hatten und ich mir jede Sorge von der Seele reden konnte, egal, wie selten wir wirklich dazu Zeit haben.

Meinen Eltern Lydia und Herwigh möchte ich dafür danken, dass sie mir stets vermittelt haben, dass ich alles werden kann, was ich möchte und mich bei allen Vorhaben stets bedingungslos unterstützt haben. Ihnen und meinen Geschwistern Felix, Dagmar und Conny danke ich für ihren tollen Rückhalt, ihre Begeisterung für mein Thema und ihre guten Vorbilder, die mich stets motiviert haben, selbst etwas erreichen zu wollen.

Und zuletzt geht mein Dank an meinen Sohn Jan, dessen begeisterte Rufe, eine Robotergeschichte zu hören, mir immer wieder die Begeisterung zeigen, mit der auch ich an diesem Thema arbeite und dessen Lächeln mir jeden Tag so viel Kraft schenkt. Danke!

Esslingen, im Februar 2024 Georg Heppner

Kurzfassung

Durch stetige Weiterentwicklung gibt es nicht nur immer mehr Roboter in unserem Umfeld, sondern vor allem mehr Spezial-Roboter die für einen eng umrissenen Anwendungsfall konzipiert wurden. In dieser Arbeit wird ein Konzept für die Kooperation dieser heterogenen Robotersysteme entwickelt um das ganze Potential aktueller Robotik für Aufgaben nutzen zu können. Es wird ein Behavior Tree basierter Ansatz für die Modellierung von Fähigkeiten und Missionen Entwickelt. Durch in den Behavior Tree integrierte Konzepte wie Parametrisierung oder entfernte Ausführungsumgebungen können Missionen schnell erstellt und über das ganze Team verteilt ausgeführt werden. Zusammen mit einem Framework für die multiroboter Koordination welches einen automatischen Austausch, Marktbasierte Verteilung und Ausführungsüberwachung der Fähigkeiten umsetzt können Roboter ihre Fähigkeiten oder Teile davon dadurch dynamisch zum Erreichen einer Mission beisteuern.

Während aktuelle Lösungen die Kommandierung sowie Kommunikation aller Systeme auf den kleinsten gemeinsamen Nenner abstrahieren wird in dieser Arbeit untersucht wie insbesondere sehr heterogene Systeme in einem Team eingebunden werden können ohne deren Komplexität zu beschränken. Durch individuelle Kostenberechnungen kann jedes System selbst bestimmen wie geeignet es ist, durch einen universellen Health-Werte jedoch im Team teilen. Eine dynamische Abbildung der Fähigkeiten auf eine roboterspezifischen Implementierungen zur Laufzeit ermöglichen Aufgaben homogen zu verteilen, aber individuell auszuführen.

Die Entwickelten Konzepte und Umsetzungen werden mit realen und simulierten Experimenten mit verschiedenen Roboterteams und Domänen evaluiert.

Inhaltsverzeichnis

1. Motivation und Forschungsfrage

Roboter haben eine rasante Entwicklung hinter sich und sind weiterhin dabei, in jedem Bereich unserer Gesellschaft Anwendung zu finden. Während in den 80er und 90er Jahren der Begriff Roboter noch mit Industrierobotern gleichgesetzt wurde, haben spätestens seit Beginn des 20 Jahrhunderts Systeme wie der Humanoide Roboter Asimo [1] oder der Staubsaugerroboter Roomba[1] die Zeit der Serviceroboter eingeläutet. Serviceroboter müssen, im Gegensatz zu ihren industriellen Vorfahren, sehr vielfältige Aufgaben erfüllen und sich ständig an neue Situationen anpassen. Eine Möglichkeit, dieses Ziel zu erreichen, ist es, Robotersysteme immer komplexer zu gestalten, um jede Eventualität abzudecken. Eine Vielzahl von leistungsfähigen Sensoren, intelligenten Algorithmen und technologische Fortschritte in allen für die Robotik relevanten Bereichen wie etwa Prozessorleistung, Regelung oder Mechanik haben dazu geführt, dass komplexe Systeme, wie etwa der humanoide Roboter Atlas[2], entstanden sind, die zwar über beeindruckende Fähigkeiten verfügen, jedoch aufgrund ihrer Komplexität (und damit Kosten) in naher Zukunft nicht massentauglich und damit wirtschaftlich einsetzbar werden. Gleichzeitig gibt es durch diese Entwicklungen viele sehr spezialisierte Systeme. UAVs (Unmanned Aerial Vehicles) etwa, wie die inzwischen weit verbreiteten Kameradrohnen, bieten eine Fülle von Funktionalitäten bei sehr geringem Preis, sind in ihrem Anwendungsfeld jedoch stark spezialisiert. Ähnlich kann in nahezu jedem Anwendungsbereich ein robotisches Spezialsystem gefunden werden. Komplexe Missionen für Serviceroboter benötigen mehr Fähigkeiten als ein einzelnes solcher Spezialsystem bietet. Diverse oder sogar komplementäre Fähigkeiten und deren Kombination über das erforderliche Maß hinaus können die Erfolgschancen für Missionen hingegen signifikant erhöhen. Der Einsatz spezialisierter Systeme ist daher nur dann zielführend, wenn diese in der Lage sind, ihre Fähigkeiten mit denen anderer Systeme zu kombinieren.

Um das volle Potential aktueller Roboter zu nutzen und so die Anwendungsgebiete und Effektivität zu erhöhen, ist es daher unabdingbar, geeignete Systeme zur *Kooperation* dieser *heterogenen Robotersysteme* zu entwickeln. Um alle beteiligten Systeme berücksichtigen zu können, ist es wichtig, insbesondere auch ihre *Fähigkeiten* geeignet zu modellieren und so eine aufgabenspezifische Kooperation zu erreichen, bei der jeder Roboter seine Fähigkeiten zum Erreichen der Ziele einer Mission beisteuern kann. Diese Arbeit befasst sich daher mit der Fragestellung, wie ein solches System für die ***fähigkeitsbasierte Kooperation von heterogenen Robotersystemen*** konzipiert und umgesetzt werden kann.

[1] https://www.irobot.com
[2] https://www.bostondynamics.com/atlas

1.1. Zielsetzung und Problemanalyse

Ziel dieser Arbeit ist es, Roboter in die Lage zu versetzen, ihre eigenen Fähigkeiten zu verstehen und diese flexibel mit anderen Robotern zu kombinieren oder diesen zur Verfügung zu stellen, um so das gesamte Potential eines Roboterteams für eine gemeinsame Mission zu nutzen und dadurch:

- Die zur Lösung einer Mission zur Verfügung stehenden Fähigkeiten durch weitere Teilnehmer zu erweitern
- Neue Fähigkeiten dynamisch in ein Missionsteam einzubringen
- Die Effizienz bei der Erfüllung einer Mission über das Team hinweg zu steigern
- Roboter zu befähigen, sich gegenseitig zu unterstützen und dadurch neue Lösungswege ermöglichen
- Spezialisierte Roboter für komplexe Missionen nutzbar machen
- Die Robustheit eines Teams durch Redundanz und komplementäre Eigenschaften zu erhöhen

Es soll insbesondere möglich sein alle Roboter, auch solche, die für spezielle Anwendungsfälle entwickelt wurden, effektiv und flexibel in eine Mission einzubinden, auch wenn sie allein nicht in der Lage wären, die Mission zu bearbeiten. Im Mittelpunkt stehen dabei der individuelle Roboter und dessen Fähigkeiten, nicht etwa eine spezifische Aufgabe oder Team-Komposition. Eine weitere Zielsetzung ist die Möglichkeit, einen Roboter flexibel und je nach seiner Fähigkeit für eine Mission einzusetzen, ohne eine Neuentwicklung für geänderte Missionen zu fordern. Wenn der Roboter seine Fähigkeiten weiter entwickelt oder ein weiterer Roboter für die Mission verfügbar wird, soll die Mission flexibel angepasst werden, um die gleiche Aufgabe mit den neuen Möglichkeiten zu erfüllen und so alle vorhanden Potentiale auszunutzen. Die Heterogenität der Systeme soll nicht, wie es bei nahezu allen aktuellen Ansätzen der Fall ist, homogenisiert werden, sondern im Gegenteil als Kern-Konzept von vornherein berücksichtigt werden und dadurch das zielgerichtete Ausnutzen der Komplementarität erlauben. Durch eine vereinheitlichte Modellierung von Fähigkeiten soll zudem nicht nur die Kooperation im Team, sondern auch die Übertragbarkeit von Fähigkeiten zwischen unterschiedlichen Robotern erhöht werden.

Die Arbeit behandelt damit das Forschungsgebiet der MRSs, deren Fragestellungen in zahlreichen Übersichten [2, 3, 4] und Taxonomien [5, 6] klassifiziert wurden (siehe 3.1). Zunächst gibt es einige grundlegende Fragen, welche das Forschungsgebiet prägen, etwa die Frage nach dem Ursprung der Kooperation, der geeigneten Größe von Teams oder der Nachbildung biologischer Vorbilder.

Es gibt viele Ansätze zu jeder Fragestellung aber keine eindeutig beste Lösung, lediglich die Entscheidung, in welchem Bereich sich eine Lösung bewegen soll.

Für diese Arbeit wurden auf Basis der Zielstellungen folgende Festlegungen getroffen:

- Es kooperieren ***Roboterteams***, also wenige[3], meist sehr komplexe Systeme miteinander, ***ohne biologisches Kommunikationsvorbilder zu kopieren***.
- Das Team ist eine ***dynamische Gruppe***, es können also jederzeit neue Teilnehmer zum Team hinzustoßen oder es wieder verlassen.
- Die Teammitglieder sind ***heterogene Roboter***, also eigenständige Roboter mit unterschiedlichen **Fähigkeiten**, **Ausprägungen** und **Autonomiegraden**
- Die Teammitglieder verfügen über ***variable Autonomiegrade***, sie können also die Bandbreite von teleoperiertem System bis hin zum vollautonomen Roboter abdecken.
- Es handelt sich um ***bewusste Kooperation*** (intentional cooperation). Jeder Roboter des Teams entscheidet sich bewusst dafür, einen anderen Roboter zu unterstützen.
- Die Roboter handeln ***kooperativ***, getroffene Aussagen können als wahr angenommen werden.

Eine ebenso große Fülle an Forschungsarbeiten beschäftigt sich mit anwendungsbezogenen Fragestellungen, etwa eine kooperative Lokalisierung, das gemeinsame Manipulieren eines Objektes oder das Lernen von bestimmten Werten. Während es eine Vielzahl sehr interessanter Probleme gibt, die für Roboterteams eine hohe Relevanz haben, hängt deren Lösung sehr stark davon ab, wie die Kooperation selbst bewerkstelligt wird. Kooperatives Kartographieren etwa erfordert zunächst Annahmen darüber, wie die Aufgabe, einen Bereich zu explorieren, beschrieben und ausgeführt wird, bevor eine optimale Zerlegung und Verteilung der Aufgabe an die Teammitglieder erfolgen kann. Während eine Lösung etwa ein gemeinsames Optimierungskriterium für Positionsregler ermittelt, muss in einer anderen zunächst eine Flächenzerlegung stattfinden.

Die wichtigsten Fragestellung sind daher solche, die notwendigerweise gelöst werden müssen, um eine Kooperation zu ermöglichen. Darunter fallen vor allem die Fragen nach der Kommunikation, dem Auflösen von Ressourcenkonflikten und insbesondere die Architektur bzw. Organisation der Teilnehmer und Aufgaben. Ziel dieser Arbeit es die Aufgabenteilung unterschiedlicher Roboter unabhängig von der eigentlich gewählten Aufgabe zu ermöglichen. Daher ist vor allem diese Kategorie von Problemen für diese Arbeit relevant. Einige der Fragestellungen, wie die Kommunikation oder das Verhindern von Ressourcenkonflikten lassen sich meist mit genügend technischen Maßnahmen lösen und sind daher nicht der Fokus dieser Arbeit. Folgende Annahmen können aufgrund der Zielstellung und den gewählten Randbedingungen festgeschrieben werden:

[3] Üblicherweise 2-20

- Das Team nutzt zur Verständigung ***explizite Kommunikation***, das heißt, die Individuen tauschen Nachrichten aus, um sich über die Verteilung von Aufgaben zu verständigen.
- Es ist davon auszugehen, dass das Team eine ***gemeinsame Sprache*** spricht. Dies bezieht sich sowohl auf den eigentlichen Nachrichtenaustausch als auch auf die semantische Bedeutung bestimmter Begriffe.
- Die Kooperation erfolgt auf Ebene der ***Fähigkeiten***, nicht etwa auf Systemebene oder auf die Mission fokussiert.
- Das Team erfüllt ***wechselnde Aufgaben***, welche ***spontan*** auftreten können und nicht im voraus bekannt sind.
- Der Fokus der Kooperation liegt auf dem Verteilen von Aufgaben an unterschiedlich fähige Roboter, es werden also primär ***Single-Robot Tasks (SR) und Multi-Task Robots (MT)*** betrachtet, nicht etwa Aufgaben, die mehrere Roboter erfordern (MR).
- Missionen werden ***dynamisch*** und auf unterschiedlichen ***Granularitätsebenen*** zerlegt. Es werden sowohl abgeschlossene Aufgaben (suche das Objekt) wie auch Teile einer Aufgabe (öffne die Tür) verteilt.

Als bisher nicht ausreichend adressiert erscheint in diesem Kontext die Frage der Aufgabenzuordnung, also die Frage, welcher Roboter welche Aufgabe der Mission ausführt oder überhaupt ausführen kann. Die Kernfrage dieser Arbeit lautet daher:

Wie können heterogene Robotersysteme mit unterschiedlichem Autonomiegrad ihre Fähigkeiten effektiv koordinieren und austauschen, um sich bei der Bearbeitung komplexer Aufgaben zu unterstützen?

Unterstützung bei Aufgaben bedeutet im Kern die Frage zu beantworten, wer welche Aufgabe übernimmt, also das Problem der MRTA (Multi-Robot Task Allocation) zu lösen. Eine Aufgabe und die Zuordnung zu dieser kann auf sehr unterschiedlichen Ebenen erfolgen. In dieser Arbeit sollen vor allem die Fähigkeiten von sehr unterschiedlichen Robotern genutzt werden, daraus ergibt sie die erste Forschungsfrage:

> **Forschungsfrage 1.** Wie können verschiedene Fähigkeiten über mehrere Roboter koordiniert, kombiniert oder fehlende Fähigkeiten von Robotern durch andere kompensiert werden, ohne die Fähigkeiten oder die Darstellung der einzelnen zu beschränken?

In den betrachteten Arbeiten zeigen sich zwei grundsätzliche Vorgehensweisen für diese Fragestellung. Die eine ist die Abstraktion der Mission und der Fähigkeiten für eine gemeinsame Stellgröße. Das Projekt MARS2020 (3.3.3) etwa nutzt die Minimierung der Unsicherheit durch jeden einzelnen Agenten als Eingabe für das individuelle Verhalten. Diese implizite Koordination erfordert gar

keinen Austausch der Fähigkeit selbst, sondern lediglich die Festlegung auf ein gemeinsames Optimierungskriterium (in diesem Fall den Informationsgewinn). Dadurch können zwar unterschiedlich umgesetzte Fähigkeiten einheitlich eingesetzt werden, jedoch müssen alle Fähigkeiten in etwa das gleiche Ziel verfolgen, was die Art der Fähigkeiten letztendlich wieder stark homogenisiert. Die Verhaltensbasierten Frameworks ALLIANCE und BLE unterdrücken oder stimulieren bestimmte Verhalten der anderen Roboter, wofür zwar kaum Informationen über den Aufbau der Fähigkeiten benötigt werden, jedoch alle Fähigkeiten bereits bei der Implementierung bekannt sein müssen.

Die Frage der Koordination wird bei diesen Ansätzen implizit gelöst, indem diese bereits in die formulierte Problemlösung integriert wird. Die individuellen Fähigkeiten der Teilnehmer sind irrelevant, es geht allein um das Erfüllen der Mission. Während dadurch zwar unterschiedliche Fähigkeiten möglich sind, verletzt ein solcher Ansatz die Prämisse eines dynamischen Teams, bei dem Teilnehmer jederzeit kommen oder gehen können. Ebenso verschiebt sich der Fokus von einzelnen Robotern und der Möglichkeit, Fähigkeiten zu erweitern oder zu nutzen, hin zur Missionsdefinition.

Die andere Vorgehensweise abstrahiert den Begriff der Fähigkeit, um diese als diskrete Einheiten zu kommunizieren und auf die Mission abzubilden. Frameworks mit Auktionen wie M+ oder MURDOCH einigen sich dafür auf fest definierte Fähigkeiten, welche dann von den Robotern flexibel angeboten werden können. Es wird jedoch primär die Frage beantwortet, welcher Roboter derzeit die besten Voraussetzungen für eine Fähigkeit erfüllt, neue Fähigkeiten, die während der Missionserstellung noch nicht bekannt waren, können nicht berücksichtigt werden. ASyMTRE (3.2) hingegen behandelt das andere Extrem. Jede Aufgabe wird zunächst durch spezielle Schema-Blöcke synthetisiert, welche zusammen eine Fähigkeit ergeben und dadurch die Mission erfüllen können. Während dieses Vorgehen flexibel ist, ist es schwierig, komplexere Aufgaben auf diese Art und Weise geeignet zu modellieren.

Diese Ansätze lösen das Problem der MRTA vor allem als Optimierungsproblem. Es wird zunächst sichergestellt, dass ein Ziel oder eine Aufgabe durch mindestens einen Teilnehmer des Teams bearbeitet werden kann. Sind mehrere Teilnehmer in der Lage, eine Aufgabe zu erfüllen, wird mittels der Berechnung von Optimalitätskriterien eine Zuordnung von Robotern bzw. deren Fähigkeiten zu einer Aufgabe erreicht.

Es wird klar, dass bei Systemen, die auf einer fähigkeitsbasierten Koordination beruhen, die Frage der Modellierung der Fähigkeiten eine zentrale Rolle spielt. Dynamische, heterogene Systeme, bei denen nicht klar ist, welche Fähigkeiten überhaupt vorhanden sind, erschweren die Entscheidung zusätzlich. Die zweite Forschungsfrage lautet daher:

Während für die Koordination der Fähigkeiten vor allem die Frage der optimalen Verteilung der Aufgaben gelöst werden muss, erfordert die Kombination oder Kompensation von Fähigkeiten eine komplexere Modellierung und insbesondere

> **Forschungsfrage 2.** Wie können unterschiedliche Fähigkeiten für heterogene Systeme modelliert und diese Informationen dynamisch ausgetauscht werden, um eine einheitliche Verwendung der Fähigkeiten zu ermöglichen ?

eine feinere Granularität der Darstellung. Soll eine Aufgabe durch mehrere Fähigkeiten, teilweise von unterschiedlichen Robotern, erfüllbar sein, müssen diese ihre Fähigkeiten kombinieren können. Gleichzeitig ist für das Kompensieren einer nicht vorhandenen Fähigkeit entweder ein zweiter Roboter mit der gleichen Fähigkeit oder aber das Aufbrechen der Aufgabe in kleinere Aufgaben erforderlich.

Ansätze für die hierarchische Zerlegung, etwa HTNs, eignen sich für diese Art der Darstellung als auch der Aufgabenverteilung, allerdings erfordern diese ebenfalls die Definition von eindeutigen Zerlegungsregeln. Die Mission wird dadurch zwar flexibel abgebildet, bei heterogenen Systemen und damit unterschiedlichen Abbildungsregeln ist die Definition jedoch schwierig.

Es ist bisher kein (anerkannter) gemeinsamer Nenner für eine Aufgabensprache bekannt. Bei Ansätzen, die doch eine zu nutzen, wie etwa der Flight Intention Description Language [7], wird die Komplexität der Nachrichten massiv reduziert. Häufig sind „gehe zu x“ oder „nutze einen Sensor“ die komplexesten Ausdrücke. Es muss ein besserer Weg gefunden werden, wie Intentionen oder Missionen in nennenswert komplexer Weise übermittelt werden können, ohne dabei die Möglichkeiten zur unterschiedlichen Ausführung durch die Roboter zu beschränken. Die Frage nach der Darstellung der Mission ist untrennbar mit der Definition der Fähigkeiten selbst verbunden, die dritte Forschungsfrage lautet daher:

> **Forschungsfrage 3.** Wie können die Fähigkeiten eines Teams auf die Mission abgebildet werden, ohne dabei die Flexibilität und dynamische Zusammensetzung des Teams und der Individuen einzuschränken ?

Zuletzt ist die Granularität der Aufgaben, Kommunikation und Funktionsblöcke ein wichtiges Merkmal, welches viel Aufmerksamkeit erfordert. Einige Projekte, etwa COMETS oder ARCAS, definieren unterschiedliche Autonomielevel, um zu unterscheiden, was übermittelt werden kann und zu welcher Art der Kommunikation ein Zielsystem überhaupt in der Lage ist. Beim TraderBots Framework werden Aufgaben als Compound Tasks oder Elementary Task definiert, was eine Unterscheidung bezüglich ihrer Zerlegbarkeit ermöglicht. Bei dem Einsatz sehr unterschiedlicher Systeme ist es jedoch nicht ohne Weiteres klar, welche Teile einer Mission überhaupt an andere Systeme ausgelagert werden können. Während einige Ansätze komplette Missionen austauschen können, übermitteln andere nicht viel mehr als ein Utility Wert und koordinieren sich dadurch implizit. Die Nutzung von stark spezialisierten Systemen verstärkt diese Problematik noch. Einige sind in der Lage vollständige Missionspläne auf symbolischer Ebene zu erstellen und auszuführen, andere können nur Befehle einer API entgegen nehmen. Es muss also weiterhin nicht nur eine Möglichkeit zur Darstellung komplexer Missionen gegeben sein, sondern dabei auch unterschiedliche Ebenen an

Granularität und Autonomie der Systeme berücksichtigt werden. Die vierte Forschungsfrage dieser Arbeit lautet daher:

> **Forschungsfrage 4.** Wie können variable Autonomiegrade, insbesondere von lediglich teilautonomen Robotern, im Team berücksichtigt werden ?

1.2. Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut. Nach dieser Einleitung und eingehenden Problemanalyse folgt in Kapitel 2 die Einführung einiger Grundlagen, die für das spätere Verständnis erforderlich sind, etwa die Unterscheidung zwischen Team und Schwarm oder die verwendeten Frameworks. Das Kapitel 3 untersucht dann den State of The Art genauer, der in der Problemanalyse bereits angeschnitten wurde. Zunächst um die allgemeinen Forschungsrichtungen der MRSs herauszustellen, dann mit speziellem Blick auf die verwendeten Frameworks in Literatur und realen Projekten. Danach folgt ein tieferer Einblick in die verwendeten Behavior Trees deren Entwicklung. Im Kapitel 4 wird dann der Zentrale Ansatz dieser Arbeit präsentiert dabei geht dieser insbesondere auf die Teilkomponenten Roberfähigkeiten (4.2), das umgebende Framework (4.3) Behavior Tree Implementierung (4.4) und Fähigkeitsverteilung und Koordination (4.5) und ihre entsprechende Umsetzung eingegangen wird. Kapitel 5 stellt das Team von Robotern, durchgeführte Analogmissionen und Simulationen vor mit denen das Konzept experimentelle evaluiert wird und zeigt die Ergebnisse der verschiedenen Aspekte. Im Kapitel 6 werden die erzielten Ergebnisse und mögliche Erweiterungen diskutiert bevor 7 eine Abschließende Zusammenfassung der Ergebnisse und des Beitrags der Arbeit gibt.

2. Grundlagen

In diesem Kapitel werden zunächst einige Abgrenzungen und Definitionen getroffen und Technologien erläutert, die wichtige Grundlage für das Verständnis der Arbeit darstellen.

2.1. Roboterteams und verwandte Themen

Wie bei vielen Themen in der Robotik und Informatik ist eine klare Abgrenzung des Themengebietes nicht strikt möglich, da viele Methoden und Verfahren bei unterschiedlichen Problemstellungen Anwendung finden. Roboterteams können je nach Sichtweise auch als Sensornetzwerk oder gar Schwarm verstanden werden, die dabei verfolgte Zielstellung ist jedoch meist eine andere. Um zu verdeutlichen, wo der Schwerpunkt der verwandten, jedoch inhaltlich durchaus unterschiedlichen, Themengebiete liegt, sollen diese hier kurz vorgestellt werden.

2.1.1. Multiagentensysteme

Agentensysteme ist ein Begriff aus der Softwaretechnik, mit welchem ein eigenständiger Prozess, je nach Anwendung beliebig komplex, bestimmte Aufgaben selbstständig, also ohne ständigen externen Trigger, erledigt. Ein Agent besitzt innerhalb eines definierten Handlungsspielraums die Möglichkeit, Entscheidungen zu treffen um seine Aufgabe zu erfüllen und wird oft für Aufgaben wie die Informationssuche oder wiederkehrende Wartungsaufgaben eingesetzt. Vor allem auch im Feld der KI wird oft von Agenten als Akteuren gesprochen. Multiagentensystem sind ein Verbund von vielen Agenten die meist kooperativ eine gemeinsame Aufgabe erfüllen. Typische Themen für Multiagentensysteme sind vor allem die Datenfusion gesammelter Ergebnisse sowie die Koordination der Ausführung und der Aufgaben untereinander. Gerade im Bereich der KI sind vor allem selbstlernenende Agenten ein wichtiges Forschungsthema.

Unterschiede zu Roboterteams

Da der Begriff „Agent“ nicht zweifelsfrei definiert ist, können auch Roboter mit dieser Bezeichnung belegt werden. Üblicherweise wird er jedoch für reine Softwareagenten verwendet. Im Gegensatz zu Robotern werden diese in einer grund-

sätzlich anderen Umgebung eingesetzt, wodurch die Problemstellungen von denen in der Robotik abweichen. Softwareagenten sind in der Regel (nahezu) allwissend, was ihre direkte Umgebung, ihre Möglichkeiten und ihre Fähigkeiten betrifft. Sie operieren in einer überwiegend deterministischen Umgebung, bei der eine gewisse Aktion stets mit der gleichen oder zumindest vorhersagbaren Reaktion einhergeht. Die wichtigsten Themen für Agenten sind daher Ressourcenallokation und Scheduling. Auch wenn diese Themen für Roboterteams relevant sind unterscheiden sich die Lösungen durch die unterschiedlichen Prämissen meist enorm. Bereits ein simple Aufgabe, wie das Abfragen gewisser Daten, ist bei mobilen Systemen ohne permanente Funkverbindung eine nicht triviale Aufgabe, bei Softwareagenten ist damit in der Regel nicht zu rechnen.

2.1.2. Schwarmroboter

Der Begriff Schwarm stammt aus dem Tierreich und bezeichnet eine große Menge gleichartiger Tiere. Regelmäßig können große Schwärme, etwa die in den Süden fliegenden Vögel, beobachtet werden, wobei auffällt, dass das Verhalten des gesamten Schwarms koordiniert abläuft. Nicht nur kommen sich die individuellen Tiere nicht in die Quere, sondern es ist sogar möglich, komplizierte Manöver zu beobachten, ohne dass eine aufwendige Kommunikation oder gar Absprachen zu erkennen wären.

Genau diese Eigenschaften, also das Kontrollieren einer sehr großen Menge von Entitäten durch implizite Kommunikation, ist wiederum eine wünschenswerte Eigenschaft für sehr große Mengen von Robotern, vor allem wenn kostengünstige, wenig komplexe und meist sehr kleine Roboter mit schwachen Sensoren eingesetzt werden. In der Robotik werden Schwarmansätze beispielsweise zur Koordination von vielen UAVs verwendet, wobei gewöhnlich ein Roboter die Führung übernimmt und die anderen Teilnehmer nach vorher festgelegten Regeln folgen. Auch explizite Kooperation ist bei Schwärmen möglich, etwa durch Pheromonspuren. Oft erfolgt die Kooperation zudem über die Umwelt oder die Aufgabe selbst, etwa wenn viele Ameisen einen Gegenstand tragen und sich lediglich indirekt über die ausgeübten Kräfte auf das zu bewegende Objekt koordinieren.

Unterschiede zu Roboterteams

Roboterschwärme gehen bei ihren Teilnehmern von einer Gleichartigkeit aus und konzentrieren sich auf das Können des gesamten Schwarmes. Durch das Ausnutzen von einfachen Regeln und einer impliziten Koordination konzentrieren sich die Algorithmen auf den kleinsten gemeinsamen Nenner der Teilnehmer und erfordern daher wenig Voraussetzungen. Sie verlieren, im Gegensatz zu einem Team, dabei jedoch den Blick auf die Fähigkeiten der einzelnen Individuen. Spezielle Fähigkeiten wie etwa die Möglichkeit, Gegenstände zu greifen, während

andere Teilnehmer des Schwarmes sich nur bewegen können, werden nicht weiter berücksichtigt, wodurch der komplette Schwarm als Gesamtheit nicht in der Lage ist, Objekte zu greifen.

Allgemeingültige Ansätze zur Vermeidung von Ressourcenkonflikten oder dem Verteilen von Ressourcen, etwa beim Bestimmen des Aufenthaltsortes in einer Formation oder dem Aufspannen eines Kommunikationsnetzwerkes, sind ohne weiteres auch auf Teams übertragbar. Vor allem jedoch die Planung und Koordination von verschiedenen Fähigkeiten und komplexeren Aktivitäten die die bewusste Kooperation erfordern fallen nicht mehr in das Gebiet der Schwarmrobotik.

2.1.3. Sensornetzwerke

Der Begriff „Sensornetzwerk“ kann eine Vielzahl von Dingen bezeichnen, denen gemeinsam ist, dass jeder Teilnehmer dieses Netzwerkes über mindestens einen Sensor verfügt und diesen mit einer gewissen Rechenleistung verarbeitet. Ein einzelner Sensorknoten kann dabei sowohl durch einen wenige mm großer integrierter Sensor repräsentiert sein, welcher lediglich einzelne Messungen aufnimmt und zum Beispiel in Materialien eingelassen werden kann, oder durch ein komplexeres Sensorsystem, wie etwa Messbojen, welche mittels GPS und weiteren Sensoren die Meeresströmungen messen. Auch Handys werden mehr und mehr als mobiles Sensornetzwerk eingesetzt, etwa um die Stauvorhersage in der Google Maps Applikation mit Daten aller Android Handys zu verbessern [8].

Relevante Kernthemen für Sensornetzwerke sind vor allem Routing und Kommunikation in ad-hoc Netzwerken, Auswertung von verteilten Daten und bei mobilen Systemen Energieaspekte.

Unterschiede zu Roboterteams

Roboter verfügen grundsätzliche über Sensorik, Rechenkapazität und meistens über Kommunikationsmöglichkeiten, weshalb sie gelegentlich als Knoten innerhalb eines Sensornetzwerkes betrachtet werden. Das AWARE Projekt nutzt etwa das TinyCubus [9] Framework für Sensornetzwerke um damit Roboter und reine Sensornetzwerke als Einheit zu steuern bzw. zu überwachen um Brände schnell und zielgerichtet zu detektieren. In den meisten Fällen bestehen Sensornetzwerke vor allem aus kleinen, oft batteriebetriebenen Knoten, welche nur über geringe Leistung und keine eigene Aktorik verfügen. Diese werden beispielsweise für die Waldbrand-Detektion aus einem Flugzeug abgeworfen. Bei solch einem Netzwerk stehen die Fragen nach einem effizienten Routing der Daten oder das energieeffiziente Kommunizieren im Vordergrund. Ein Roboter hingegen kann durch seine Aktorik die Umgebung verändern und sich in ihr bewegen, wodurch Themen wie Pfadplanung und Hindernisvermeidung in den Vordergrund rücken,

während Themen wie Kommunikation und Routing weiterhin relevant, jedoch nicht die vorrangigen Probleme sind.

Wie auch bei der Schwarmrobotik ist die Unterscheidung zwischen Roboterteams und Sensornetzwerken vor allem auch eine des betrachteten Individuums. In Sensornetzwerken liegt der Fokus auf dem Netzwerk an sich, also dem Auswerten und Kommunizieren von Daten, während das Individuum lediglich als Beschränkung der Reichweite oder Batterielaufzeit einbezogen wird. Ein Roboterteam hingegen besteht aus individuellen Systemen, welche durch ihr Handeln zu einem gemeinsamen Ziel beitragen. Die entwickelten Algorithmen sind zweifelsfrei für beide Gebiete im Wechsel relevant, der betrachtete Fokus ist daher jedoch ein anderer.

2.2. ROS

Das ROS (Robot Operating System) [10] ist eine Middleware für die einfache Wiederverwendung, Verbreitung und Kontrolle von Software für Roboter. Trotz seines Namens handelt es sich nicht um ein echtes Betriebssystem, es setzt auf Linux (eingeschränkt auch auf Windows oder ausgewählte andere Plattformen) auf und stellt neben einer Reihe wichtiger Werkzeuge vor allem eine einheitliche Ausführungs- und Kommunikationsumgebung zur Verfügung. Seit 2017 werden auch aktiv ROS 2 Distributionen veröffentlicht. Die zweite Version des Frameworks wurde vollständig überarbeitet und adressiert viele offene Fragen von ROS, wie z.B. eine Unterstützung für Windows, Echtzeitunterstützung oder automatische Discovery per DDS Kommunikation. Seit seiner Einführung wurde erst zögerlich auf ROS 2 gewechselt, da ROS bereits auf vielen Systemen etabliert war und stabilere Features bot. Bis 2023 ist die Bedeutung von ROS 2 stetig gewachsen so das es inzwischen der Standard geworden ist und auch für integrierte Lösung wie Robotersteuerung untersucht wird [Puck et al., 2020a]. ROS spielt jedoch weiterhin eine sehr aktive Rolle. Die grundlegenden Konzepte von ROS und ROS 2 unterscheiden sich kaum. Erklärtes Ziel von ROS ist es, den permanenten Zyklus von Re-Implementierung von Basis Funktionalität („re-inventing the wheel“) zu durchbrechen, um so die Möglichkeit für Innovation zu schaffen. ROS hat sich zu einem der am weitesten verbreiteten Frameworks, vor allem in der Forschung, zunehmend aber auch in der Industrie, entwickelt und wächst ständig weiter. Der jährliche Report über die Metriken [11] berichtet in 2018 von 11.770 eigenständigen Paketen und einer Erwähnung in 4806[1] Veröffentlichungen. In 2022 sind es bereits 24.069 unterschiedliche Pakete und insgesamt 10.467 Nennungen der ROS-Referenz [12]. ROS bietet 3 wichtige Säulen für die Entwicklung, Verbreitung und Nutzung von Applikationen für Robotern: Ein Kern-System für die Ausführung von Algorithmen, eine flexible Kommunikation und Werkzeuge zur Entwicklung und Nutzung des Frameworks.

[1] Die Zahl bezieht sich auf alle Nennungen, nicht nur in 2018

2.2.1. ROS Packages, Nodes & Core

Software wird bei ROS in unabhängigen Paketen veröffentlicht, welche dezentral in eigenen Repositories verwaltet oder aber als vorübersetztes Debian-Paket verteilt werden. Durch eine einheitliche Paketstruktur mit Metainformationen in der *package.xml* können Pakete schnell in ein bestehendes System integriert und mit anderen Entwicklern ausgetauscht werden. Die Software in einem Paket wird in (beliebig vielen) *ROS-Nodes* organisiert und kann primär in C++ und Python implementiert werden. Eine *ROS-Node* ist eine modulare Ausführungseinheit, welche sich zur Laufzeit beim zentralen *ROS-Core* anmeldet und durch diesen verwaltet wird. Der *ROS-Core* ermöglicht neben der dynamischen Handhabung des Lebenszyklus (starten, stoppen) von *Nodes* auch Parameterhandling und die netzwerktransparente Kommunikation über *ROS-Messages*. In ROS2 ist ein dedizierter *ROS-Core* nicht mehr erforderlich, da die Kommunikation und Discovery über DDS-Kommunikation erfolgt, die diese Themen direkt unterstützt.

2.2.2. ROS Messages

Um die Wiederverwendbarkeit zu stärken und die Softwareentwicklung voneinander unabhängig zu machen, verwendet ROS drei auf *ROS-Messages* basierende Kommunikationsmuster: *Topics*, *Services* und *Actions*. *ROS-Messages* definieren den Datentype von Nachrichten. Sie werden in einer Beschreibungssprache definiert und für die jeweilige Zielsprache (und einfache Serialisierung) kompiliert. Eine *ROS-Node* muss zum Übersetzungszeitpunkt lediglich die Nachrichtendefinition kennen um sie zu verwenden und ist damit vor allem von den Komponenten die den Inhalt der Nachricht produzieren unabhängig. *Topics* sind unquittierte UDP-Nachrichten, welche periodisch geschickt werden, z.B. der aktuelle Batteriestand. *Services* sind blockierende, synchrone RPCs von einem *service client* an einen *service server*. Sie bestehen aus einer Anfrage (*Request*) und einer Antwort (*Response*) mit beliebigen *ROS-Message* Typen. Da ein Service die aufrufende *Node* blockiert, wird gefordert, dass *Services* augenblicklich terminieren, sie werden daher für kurze Steueranfragen wie das Aktivieren eines Motors eingesetzt. Als Alternative bieten *Actions* die Möglichkeit asynchrone, langfristige Aktionen anzufragen. Eine *Action* definiert ein Ziel (*Goal*), welches die Zielvorgabe an den *Action Server* übermittelt, *Feedback*, welches periodisch vom Server an den Klienten zurückgeschickt wird, um den Fortschritt anzuzeigen, und ein Ergebnis(*Result*), welches bei Fertigstellung oder Abbruch der Aktion den aktuellen Zustand und die Ergebnisse an den Auftraggeber zurückliefert. Aktionen werden für langfristige Aktivierungen, z.B. die Vorgabe, zu einem Zielpunkt zu fahren, genutzt und können während ihrer Ausführung abgebrochen werden. *Topics*, *Services* und *Actions* werden von *ROS-Nodes* mit einem eindeutige Namen nach dem Muster */namespace/name* veröffentlicht, wodurch sie von anderen *ROS-Nodes* angesprochen werden können (*publish/subscribe*). Wird eine entsprechende Nachricht über diese Namen geschickt, wird in zugehörigen *ROS-Node* eine

Callback-Funktion aufgerufen, welche die Nachricht verarbeiten kann.

2.2.3. ROS Tools

Einen großen Teil zur Popularität des Frameworks haben die vielen verfügbaren Werkzeuge und Kern-Bibliotheken beigetragen. Für die Koordinatentransformation etwa gibt es das *TF-System* (TF für Transform), welches das fehleranfällige Rechnen mit Rotations- und Translationsmatrizen abstrahiert. Grafische Werkzeuge wie RVIZ[2] bietet die intuitive und erweiterbare Anzeige komplexer 3-dimensionaler Modelle und Kommandozeilen-Werkzeuge wie *rostopic list* ermöglichen eine Introspektion in die Ausführung des Roboters zur Laufzeit auch ohne grafische Oberfläche.

2.3. Behavior Trees

Behavior Trees sstellen eine zunehmend populär werdende Alternative zu hierarchischen Zustandsautomaten oder vergleichbaren Methoden dar, um Verhalten von Robotern in Software abzubilden. Ein Behavior Tree ist ein gerichteter Graph aus einem Wurzel-Knoten, Flusskontroll-Knoten und Blatt-Knoten (Abbildung 2.1). Knoten sind von links nach rechts geordnet und durch Kanten verbunden, der Ursprungsknoten heißt Elternknoten, die Ziele Kindknoten.

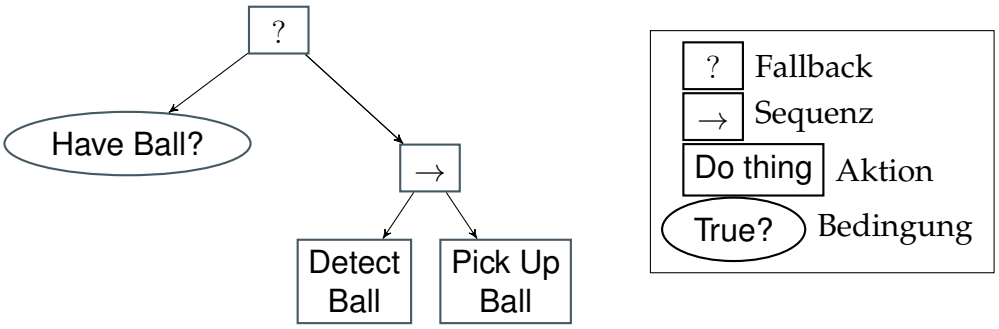


Abbildung 2.1.: Beispiel für einen einfachen Behavior Tree. Flusskontroll-Knoten (Fallback: ? oder Sequenz: →) definieren, wie der *Tick* entlang des Baumes weitergeleitet wird. Bedingungen werden geprüft und enden entweder im Zustand *Succeeded* oder *Failed*, was wiederum nach oben propagiert wird. Aktionen führen konkrete Handlungen aus, sobald sie aktiviert werden und liefern ebenfalls *Succeeded* oder *Failed* als Ergebniszustand zurück. Quelle: [Heppner et al., 2023], ©2023 IEEE

Ein Behavior Tree kann ausgeführt werden, indem ein sogenannter *Tick*, also ein Aktivierungssignal für die einzelnen Knoten, in einer festen Frequenz vom

[2] `http://wiki.ros.org/rviz` Zugriff am 09.11.2023

Wurzelknoten aus an alle Kinder gesendet wird. Der Vorgang, dass ein Knoten durch das *Tick* Signal aktiviert wird wird als *ticken* des Knotens bezeichnet. Ein **Flusskontroll-Knoten** hat einen Elternknoten und mindestens einen Kindknoten, er bestimmt wie der *Tick*, also die Ausführungsberechtigung für die Knoten, an die nachfolgenden Kindknoten übergeben und dessen Rückgabewerte ausgewertet werden. Ein **Blatt-Knoten** hat keine Kinder und führt beim Erhalt eines *Tick* eine definierte Aktion aus oder prüft eine Bedingung, bevor eine Status zurückgegeben wird. Wenn die Ausführung eines Knotens nach dem *Tick* noch nicht beendet ist, wird der Status *Running* zurückgegeben, andernfalls *Succeeded*, wenn die Ausführung oder Bedingung erfolgreich war, und *Failed*, falls nicht.

Es gibt folgende Basis-Flusskontroll-Knoten:

- Sequenz - Aktiviert seine Kinder, von links nach rechts, in einer Sequenz, solange diese *Succeeded* als Status zurückgeben. Es setzt den Status *Succeeded* wenn alle Kinder erfolgreich sind und schlägt fehl, sobald ein Kind ein *Failed* meldet. Wenn ein Kind *Running* zurückgibt, ist die Sequenz auch *Running*.

- Fallback - Aktiviert seine Kinder, von links nach rechts. Sobald ein Kind *Succeeded* oder *Running* zurückgibt nimmt auch der Fallback den Status *Succeeded* oder *Running* an. Nur solange der aktiviere Kind-Knoten fehlschlägt, wird das nächste Kind-Knoten aktiviert.

- Parallel - Aktiviert seine Kinder gleichzeitig. Hat den Zustand *Running*, solange seine Kinder diesen haben und ist je nach Einstellung erfolgreich, wenn n Kinder im *Succeeded* Zustand sind, wobei $1 \leq n \leq$ *numberOfChildren*. Der Knoten wechselt in den *Failed* Status wenn mehr als *numberOfChildren*$-n$ Kinder den Status *Failed* zurückgeben.

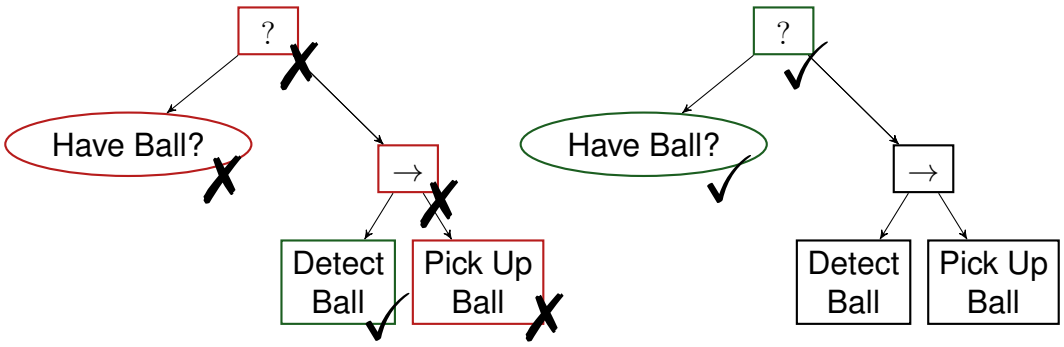


Abbildung 2.2.: Beispiel für einen Einfachen Behavior Tree nach der Ausführung. Links: *Have Ball* wird zu *Failed* ausgewertet, durch den *Fallback* wird dann die *Seuenz* ausgeführt. Beim *Pick Up Ball* schlägt diese Fehl, der Gesamtzustand des Baumes ist damit *Failed*. Rechts: Die Bedingung ergibt bereits ein *Succeeded*, daraufhin wird der komplette Baum als *Succeeded* markiert.

Neben diesen Basis-Knoten gibt es noch weitere Varianten, etwa eine *Memory Sequenz*, bei der die bereits erfolgreichen Kinder nicht noch einmal aktiviert werden, oder auch spezielle Knoten, wie z.b. eine Selektion der Kinder auf Basis einer Kostenfunktion. Eine letzte Art Flusskontrolle sind die Dekoratoren. Ein Dekorator hat genau ein Elternteil und ein Kind und modifiziert den Rückgabewert oder Tick etwa durch Invertierung des Ergebnisses.

Abbildung 2.2 zeigt zwei mögliche Ergebnisse der Ausführung eines Behavior Trees. Im Beispiel wird zunächst *Have Ball?* aktiviert, welches den Status *Failed* zurückgibt. Der *Fallback* wählt daraufhin sein zweites Kind, die *Sequenz* aus die wiederum ihr erstes Kind *Detect Ball* aktiviert. Der Knoten endet im *Succeeded* Status, es wurde also ein Ball gefunden. Die *Sequenz* aktiviert daraufhin das zweite Kind *Pick Up Ball*, welches fehlschlägt. Durch den Fehlschlag ist auch die *Sequenz* sofort ein Fehlschlag und da der *Fallback* keine weiteren Kinder als Alternativen hat wird auch sein Status zu *Failed*. Im rechten Beispiel wird erneut *Have Ball?* aufgerufen. Der Knoten stellt fest, dass ein Ball bereits vorhanden ist und beendet sich im *Succeded* Status. Der Fallback *Succeeded*, sobald eines seiner Kinder dies tut, daher wird die Sequenz gar nicht weiter ausgeführt.

Ein Behavior Tree kann alle Verhalten abbilden, die ein hierarchischer Zustandsautomat auch abbilden kann, verfügt darüber hinaus jedoch über einige vorteilhafte Eigenschaften. Die wichtigsten sind Reaktivität und die Modularität. Durch das periodische aktivieren (ticken) des ganzen Baumes reagiert ein BT dynamisch auf Änderungen und kann daher schnell sein Verhalten an die Gegebenheiten anpassen. Die Art der Modellierung erlaubt es außerdem, Teile des Behavior Trees oder sogar ganze Bäume ohne Abhängigkeiten ineinander einzu setzen oder Teile zu verschieben, ohne dass konkretes Wissen über den Inhalt vorhanden sein muss. Das macht sie zu vielversprechenden Kandidaten für die Verhaltensdefinition von Robotern.

2.4. Marktbasierte Roboterkoordination

Dias, eine Expertin auf dem Gebiet der Martkbasierten Roboterkoordination gibt in [13, S. 1257] eine treffende Zusammenfassung für marktbasierte Ansätze:

> *"coordinating a multirobot team requires overcoming many formidable research challenges. Humans have met these coordination challenges forthousands of years with increasingly sophisticated market economies. In these economies, self-interested individuals and groups trade goods and services to maximize their own profit; simultaneously, this redistribution results in an efficient production of output for the system as a whole."*

Auch die weiteren Ausführungen basieren zu großen Teilen auf der übersichtlichen Zusammenfassung von Dias in [13].

Marktbasierte Ansätze zur Koordination von Roboterteams sind ein häufig eingesetztes Mittel, von dem erste Ansätze bereits 1980 [14] veröffentlicht wurden. Sie

stellen einen hybriden Ansatz aus vollkommen zentraler und dezentraler Koordination dar [13] und können daher viele Vorteile von beiden Ansätzen nutzen.

In einem zentralen Ansatz werden die Aufgaben von einer einzigen zentralen Instanz verwaltet und an die Roboter im Team vergeben. Dadurch können theoretisch optimale Lösungen erzeugt werden, gleichzeitig gibt es einen hohen Kommunikationsoverhead, langsame Reaktionszeiten [13] und das Problem, dass die notwendigen Umweltinformationen meist nicht global verfügbar sind. Dezentrale Ansätze hingegen basieren einzig auf dem lokal verfügbaren Wissen einzelner Individuen, wodurch die Ansätze sehr schnell, flexibel und robust sind produzieren aber gleichzeitig suboptimale Lösungen [13].

Die Verteilung von Aufgaben in einem marktbasierten Ansatz funktioniert üblicherweise per Auktionen, die alle nach einem ähnlichen Schema ablaufen [14, 13]:

- Announcement - Ein Auktionär gibt bekannt, welche Aufgaben auktioniert werden sollen.
- Biding - Die anderen Team-Teilnehmer berechnen ihre Kosten und geben ein Gebot für die Aufgaben ab.
- Clearing - Nach einer festgelegten Deadline werden keine Gebote mehr angenommen, der Auktionator bestimmt, welches Gebot der Gewinner der Auktion ist und gibt dies an die Teilnehmer bekannt

Die einfachste Form der Auktion ist die *single-item* Auktion, bei der immer nur eine einzelne Aufgabe gleichzeitig auktioniert wird. Alternativ werden bei *combinatoriual* Auktionen eine ganze Menge an Aufgaben angeboten und Teilnehmer können auf Teilmengen daraus bieten. Die Komplexität steigt jedoch exponentiell mit der Menge der angebotenen Aufgaben [15]. Eine *multi-item* Auktion ist ein Kompromiss aus beiden Extremen, bei dem mehrere Aufgaben angeboten werden, Gebote jedoch immer nur auf einen ganzen Satz an Aufgaben abgegeben werden können. [16] Zeigt darüber hinaus einen Ansatz, *komplexe* Aufgaben zu auktionieren, bei denen auch eine Struktur der Aufgaben durch Aufgabenbäume (*Task-Trees*) berücksichtigt wird.

Weitere Modifikationen sind parallele Auktionen, um die Verarbeitungsgeschwindigkeit zu erhöhen oder mehrere Auktionsrunden, bei denen nicht alle Aufgaben sofort, sondern in mehreren aufeinander folgenden Auktionen vergeben werden und dabei vorherige Ergebnisse berücksichtigen können [13]. Re-Auktionen können genutzt werden, um bereits gefundene Lösungen nachträglich erneut zu verhandeln, was insbesondere wünschenswert ist, wenn sich die Kosten oder andere Gegebenheiten signifikant geändert haben.

Marktbasierte Ansätze sind für die in dieser Arbeit betrachteten dynamischen Roboterteams aus heterogenen Robotern besonders geeignet. Zum einen können Sie mit Änderungen im Team oder den Fähigkeiten umgehen da Ressourcen und

Aufgaben stets neu verhandelt werden, zum anderen können Sie ihre Kosten individuell berechnen und dadurch trotz Heterogenität ein vereinendes Gebot abgeben ohne das andere über die Fähigkeiten wissen müssen. Viele Projekte setzen daher auf marktbasierte Verfahren, da die Mehrkosten durch Auktionssysteme, verglichen mit anderen Kosten, vertretbar und ihre Vorteile sehr gut sind.

3. Stand der Forschung

Die fähigkeitsbasierte Kooperation heterogener Robotersysteme beinhaltet verschiedene Forschungsgebiete. In diesem Kapitel wird daher zunächst der Stand der Forschung der für diese Arbeit relevanten Themen analysiert, um Ansätze und Probleme aktueller Techniken zu identifizieren.

Anwendungsspezifische Themen, die auch einzelne Roboter betreffen, wie Navigation, Regelung oder Missionsplanung sind für sich genommen jeweils eigene Forschungsgebiete, die teilweise für mehrere Roboter adaptiert werden können, teilweise aber auch spezielle Lösungen erfordern, die nur im Team funktionieren. MRSs (Multi-Roboter-Systeme) haben darüber hinaus eine Reihe an speziellen Herausforderungen wie Ressourcenkonflikte, Rollen- und Aufgabenverteilung. Zunächst wird daher ein Überblick über die Forschungsfragen von MRSs und existierende Klassifikationen im Feld der Multi-Roboter-Kooperation gegeben. Ziel dieser Arbeit ist es, ein System zu entwickelt, das die Koordination von Fähigkeiten für komplexe Aufgaben erlaubt. Daher sind insbesondere die Architekturen, die verwendet werden, um Aufgaben im Team geeignet zu verteilen, ein besonderer Fokus nachfolgender Betrachtungen. Dafür werden zunächst die einflussreichsten Frameworks für die Multi-Roboter-Kooperation vorgestellt, bevor die entstandenen Architekturen in verschiedenen Projekten genauer betrachtet werden. Nach diesen allgemeinen Betrachtungen wird der Stand der Technik von Behavior Trees und deren Einsatz für MRS analysiert, da diese von besonderer Bedeutung für die Arbeit sind.

3.1. Multi-Roboter-Systeme

Der Bereich Multi-Roboter-Systeme ist spätestens seit Ende der 1980iger Jahre ein aktiver Forschungsbereich, der über die Jahre immer wieder andere Schwerpunkte gesetzt hat. Aufgrund des großen Umfangs der Forschungsarbeiten beschränken sich die folgenden Ausführungen vor allem auf einflussreiche Arbeiten, oder solche die selbst einen Überblick über das Themenfeld geben.

Dudek et al. [17] haben 1996 eine Taxonomie zur Klassifizierung von "Robot Collectives"vorgestellt. Sie unterscheiden Lösungen für diese Kollektive basierend auf *Größe*, *Kommunikations-Reichweite*, *-Topologie*, *-Bandbreite*, *Rekonfigurierbarkeit*, *Rechenfähigkeit* und *Komposition*. Die Kommunikationsreichweite und Rekonfigurierbarkeit, inbesondere aber die Komposition finden eine Entsprechung

in den von Cao et al.[2] beschriebenen Forschungsachsen. Während die Klassifikationen eine sehr gute Einordnung ermöglichen, sind durch die Verfügbarkeit leistungsstarker Elektronik viele der Probleme, und damit auch der Klassen and Forschungsfragen, heute weniger relevant. Während etwa die Kommunikationsreichweite durchaus ein limitierender Faktor ist, kann dieser durch moderne Funkübertragung wie 5G weitestgehend ausgeglichen werden. Die Fragen nach Kommunikations-Bandbreite,-Topologie und -Reichweite sind daher heute nicht mehr grundlegende Forschungsfragen, sondern vielmehr anwendungsspezifische Fragestellungen, etwa beim Einsatz in schwierigem Gelände mit zerstörter Infrastruktur.

Cao et al. [2] geben 1997 eine umfassende Übersicht über das Themengebiet der MRS bis 1995 und identifizieren 5 Forschungsachsen, in welche verschiedene Architekturen eingeordnet werden.

- *Group Architecture*
  Da Fragen zur Architektur oft unterschiedliche Themen behandeln, geben sie zunächst eine kompakte Definition: "The group architecture of a cooperative robotic system provides the infrastructure upon which collective behaviors are implemented, and determines the capabilities and limitations of the system."[2]. Diese kann *Zentral* oder *Dezentral* organisiert sein, wobei es häufig vorkommt, dass eine Architektur beide Komponenten aufweist indem einzelne Teilnehmer eine Führungsrolle einnehmen. Sie nennen eine Gruppe *homogen*, wenn alle Roboter über die gleichen Fähigkeiten verfügen und *heterogen* in allen anderen Fällen. Die Unterscheidung der *Kommunikationsstruktur* mit den drei Klassen *Interaktion über die Umgebung*, *Interaktion über Sensoren* und *Interaktion über Kommunikation* ist weit weniger komplex als die von Dudek [17]. Zuletzt wird die *Modellierung anderer Teilnehmer* als Kriterium angegeben, ohne dass explizite Klassen dafür definiert werden.

- *Resource Conflicts*
  Wenn viele Roboter gleichzeitig eingesetzt werden, müssen die verfügbaren Ressourcen koordiniert werden. Vor allem das gemeinsame Nutzen von Funkkanälen, das Vermeiden von Kollisionen und der koordinierte Zugriff auf Objekte in der Umgebung sind relevante Themen.

- *Origins of Cooperation*
  Wird ein System nicht von vornherein zur Kooperation entwickelt, dann ist es eine wichtige Frage, warum Systeme kooperieren. Die Autoren unterscheiden etwa *Eusozialität*, eine angeborene Form der Kooperation von Insekten, bei denen die Interaktion für das Überleben notwendig ist und etwa Koordination bezüglich der Nahrungsquelle beinhaltet oder aber die bei Wirbeltieren vorherrschende *bewusste Kooperation*, welche bewusst durch die individuelle Bestrebung angetrieben wird, den eigenen Vorteil zu optimieren.

- *Learning*
  Reinforcment Learning auf Ebene einer Gruppe bietet die Möglichkeit, die

inhärente Komplexität zu bewältigen. Wie das Lernen innerhalb eines Teams gestaltet werden kann oder Gelerntes ausgetauscht werden kann, sind dabei wichtige Fragestellungen.

- *Geometric Problems*
  Bei MRSs spielt im Gegensatz zu den MASs die Position in der physikalischen Welt eine große Rolle. Insbesondere die Pfadplanung oder aber das Einhalten von Formationen ist daher für MRSs ein relevanter Forschungsbereich.

Bemerkenswerterweise wird von den Autoren der Bereich der *task decomposition and allocation*, also der Zerlegung und Zuweisung von Aufgaben im Team, als Forschungsgebiet explizit ausgeschlossen. Sie begründen dies damit, dass es 1. kaum Forschung auf dem Bereich gibt, 2. die relevanten Aufgaben in Bezug darauf trivial sind (z.B. für das Schieben einer Box) und 3. die Zerlegung ohnehin nur von der gewählten Architektur abhängt.

Arai et. Al [3] geben 2002 ein Update der relevanten Forschungsergebnisse und identifizieren 7 primäre Forschungsfelder für MRSs: *Biological Inspirations*, *Communication*, *Architectures, Task Allokation and Control*, *Localization, Mapping and Exploration*, *Object Transport and Manipulation*, *Motion Coordination*, *Reconfigurable Robotics*. Die Forschungsfelder decken sich größtenteils mit denen von Dudek und Cao identifizierten, bringen jedoch einige Neuerungen. Die biologischen Systeme sind überwiegend verhaltensbasierter Natur und viel Forschung wurde im Bereich der *eusozialen* Themen veröffentlicht, weniger im Bereich der *geplanten Kooperation*. Bei der Kommunikation sind gemeinsame Sprachen, das Verankern der Sprachen in der Welt und Fehlertoleranz beim Kommunizieren stärker in den Vordergrund gerückt. Im Bereich der Architekturen hat sich die Vergabe von Aufgaben durch die marktbasierten Verfahren weiterentwickelt. Lokalisierung und gemeinsame Exploration haben sich zudem zu einem dominierenden Anwendungsfall für kooperative Roboterteams entwickelt. Der Bereich der Rekonfiguration von Teams hat ebenfalls in seiner Popularität stark zugenommen, die damit verbundenen Themen sind jedoch oft eher im Bereich Schwarmrobotik anzusiedeln.

Während bisherige Übersichten versuchen, den Themenbereich nach Forschungsfragen oder Architekturen einzuordnen, erstellen Gerkey und Matarić [5] 2004 eine Taxonomie für die Problemstellungen der MRTA und formalisieren damit die Forschung unabhängig von den gewählten Architekturen. Dabei schränken sie ihre Untersuchungen auf den Bereich der *geplanten Kooperation* ein und setzten die MRTA als Kerngebiet der MRSs in den Mittelpunkt: „the fundamental question: which robot should execute which task in order to cooperatively achieve the global goal? By “task”, we mean a subgoal that is necessary for achieving the overall goal of the system, and that can be achieved independently of other subgoals (i.e., tasks)“ [5].
Um diese Frage zu entscheiden, definieren sie *Utility* als Optimierungsgröße für die Verteilung von Aufgaben. In ihrer Definition werden ein Quality-Wert, der angibt, wie gut eine Aufgaben erfüllt werden kann, und einen Cost-wert, der

angibt, wie aufwändig das Ausführen der Aufgabe ist, berechnet und voneinander abgezogen, um die gesamt Utility zu bestimmen. Durch die Berechnung und den Vergleich eines Utility-Wertes kann entschieden werden, welcher Roboter eine Aufgabe ausführen soll. Sie argumentieren, dass das Verteilen von Aufgaben analog zum optimalen Scheduling-Problem sei und damit letztendlich ein Problem der optimalen Verteilung sei. Für die Klassifizierung der unterschiedlichen Ausprägung des Problems schlagen sie 3 Hauptachsen vor:

- *single-task robots (ST)* vs. *multi-task robots (MT)*
  Während im *ST* jeder Roboter exakt eine Aufgabe ausführen kann sind *MT* Roboter in der Lage, mehrere Aufgaben simultan auszuführen.
- *single-robot Tasks (SR)* vs. *multi-robot tasks (MR)*
  SR bedeutet, dass eine Aufgabe durch einen einzelnen Roboter erfüllt werden kann. *MR* Aufgaben können auch mehrere Roboter erfodern.
- *instantaneous assignment (IA)* vs. *time-extended assignment (TA)*
  Bei der Aufgabenverteilung mit *IA* werden die Aufgaben immer basierend auf den aktuell vorliegenden Informationen vergeben, ohne weitere Schritte zu berücksichtigen. Die Verteilung mit *TA* hingegen berücksichtigt auch längerfristige Pläne, zeitliche Abfolgen und Modelle.

MRTA Problemstellungen werden nach dieser Taxonomie mit 3 Buchstabenkombinationen bezeichnet:

ST-SR-IA: single-task robots, single-robot tasks, instantaneous assignment ist das einfachste, und zugleich am häufigsten anzutreffende Problem, welches genau dem OAP [18] entspricht. Eine Menge von Aufgaben müssen auf eine Menge von Robotern verteilt werden, wobei es genügend Roboter für die Aufgaben gibt und alle Aufgaben bekannt sind, so dass jede Aufgabe zugeordnet werden kann. Mit einer zentralen Architektur kann dieses Problem durch ein lineares Programm optimal gelöst werden, oft wird jedoch ein einfacher greedy Ansatz verwendet, welcher insbesondere bei dezentralen Systemen den theoretischen Vorteil geringerer Kommunikation aufweist. Die Problemklasse tritt noch in anderen Varianten auf, etwa durch iteratives Zuweisen, bei der die instantane Zuweisung wiederholt wird und dadurch etwa viele Aufgaben zu kleinen Teams zugewiesen werden können. Insbesondere bei Teams mit wechselnden Rollen, etwa Fußballrobotern, ist diese Form häufig anzutreffen. Eine weitere Variante ist die des online-assignements, bei dem die Tasks nicht alle bekannt sind, sondern nach und nach eintreffen und die Roboter nicht neu zugeordnet werden können. In diesem Fall wird jeder Task beim Eintreffen an den zu dem Zeitpunkt fähigsten Roboter (also dem mit der höchsten Utility) vergeben.

ST-SR-TA: single-task robots, single-robot tasks, time-extended assignment ist der Fall, in dem mehr Tasks als Roboter vorhanden sind oder ein Modell der später auftretenden Aufgaben existiert, die Roboter also eine Sequenz von Tasks ausführen müssen. Die Utility muss als gewichtetes Mittel über alle Zuordnungsreihenfolgen bei Berücksichtigung parelleler Ausführung minimiert werden. Diese Fragestellung ist ein Scheduling-Problem und NP-Schwer, also nicht ohne Weiteres

berechenbar. Die von Gerkey et al. vorgeschlagene Lösung ist eine Kombination aus einer Durchführung des ST-SR-IA assignments mit nachfolgender online Zuweisung der Tasks.

ST-MR-IA: single-task robots, multi-robot tasks, instantaneous assignment beschreibt Probleme, bei denen mehrere Roboter für eine Aufgabe, etwa das Schieben einer Box, benötigt werden. Kern-Fragestellung für die Optimierung der utility Werte ist die Aufteilung der Roboter in Teams, auch als „coalition formation" bekannt. Die Fragestellung fällt in die Gruppe der NP-Schweren Mengenpartitionierungsprobleme (set-partitioning problem). Für die Lösung wurden in anderen Bereichen, etwa für das Erstellen von Einsatzplänen für Flugzeugbesatzungen, heuristische Algorithmen entwickelt, welche verwendet werden können.

ST-MR-TA: single-task robots, multi-robot tasks, time-extended assignment erweitert das Problem zusätzlich um das eines Scheduling-Problemes. Während für das ST-MR-IA Problem viele Ansätze mit akzeptablen Rechenzeiten existieren, wird für das TA Problem entweder ein iterativer IA Ansatz verwendet oder ganz andere Aufteilungen, etwa mit einem führenden Roboter, dem andere folgen, genutzt.

MT-SR-IA und MT-SR-TA : multi-task robots, single-robot tasks, instantaneous/time-extended assignment bezeichnen Aufgaben, bei denen Roboter mehrere Tasks gleichzeitig ausführen und damit vor allem Aufgaben, bei denen verschiedene Sensorik verwendet werden muss, da die Sensorik von Robotern oftmals für eine Aufgabe spezialisiert ist. Der Lösungsansatz von MT-SR-IA Problemen ist äquivalent zu ST-MR-IA Ansätzen, da statt einer Menge an Robotern eine Menge an Tasks gefunden werden muss. Die MT-SR-TA Problemlösungsansätze entsprechen damit denen der ST-MR-TA.

MT-MR-IA und MT-MR-TA : multi-task robots, multi-robot tasks, instantaneous/time-extended assignment beschreibt Probleme, bei denen sowohl die Roboter mehrere Tasks gleichzeitig ausführen können, aber auch mehrere Roboter für eine Aufgabe benötigt werden. Dieses Problem tritt insbesondere bei Überwachungsaufgaben auf, bei denen eine Observation mit mehreren Robotern durchgeführt wird, aber jeder Roboter nur über einen begrenzten Sensorhorizont verfügt (z.B. weil heterogene Systeme eingesetzt werden). Die TA Variante erfordert zusätzlich zeitliche Bedingungen, etwa dass ein bestimmter Bereich zuerst untersucht wird. Diese Problematik wird von den Autoren als Mengenüberdeckungsproblem (set-covering problem) eingeordnet welches bei der TA zusätzlich um ein Scheduling-Problem erweitert wird. Während für den IA Fall Heuristiken existieren, ist das TA Problem weitestgehend ungelöst.

In 2013 schlagen Korsah, Stentz und Diaz eine Erweiterung der Taxonomie vor [6], welche auch die Abhängigkeiten zwischen den Tasks genauer definiert. Sie folgen zunächst der Definition von Zlot [19] und teilen die Task Definition auf in *Elemental Task*, für nicht weiter zerlegbare Aufgaben, *Simple Tasks*, welche in Elemental Tasks zerlegt werden können oder abermals Simple Tasks enthalten, *Compound Tasks*, die aus mehreren Simple Tasks oder wiederum Compound Tasks bestehen, wobei die unterschiedlichen Simple Tasks unterschiedlichen Robotern

zugeordnet werden können und *complex tasks*, welche aus verschiedenen Compound Tasks bestehen können, die auf verschiedene Roboter verteilt werden können. In der vorgeschlagenen Kategorisierung werden die bestehenden Klassen von Gerkey et al. weiter verwendet, jedoch um die Dimension der Abhängigkeit erweitert, wofür folgende Klassen definiert werden:

- *No Dependencies (ND)*
  Beschreibt Aufgaben (*Simple oder Compound Tasks*) die voneinander unabhängig sind und deren Utility-Wert nicht durch die Zuordnung zu einem Roboter oder die Reihenfolge der Ausführung verändert wird.

- *In-schedule Dependencies (ID)*
  Wenn der Utility Wert einer Aufgabe (*Simple oder Compound Tasks*) von der Reihenfolge der Ausführung abhängt. Für eine optimale Zuordnung müssen scheduling Konflikte gelöst werden, da die Aufgaben nicht in beliebiger Ordnung ausgeführt werden können.

- *Cross-schedule Dependencies (XD)*
  Beschreibt Abhängigkeiten, bei denen der Utility-Wert von Aufgaben (*Simple oder Compound Tasks*) von den Plänen anderer Roboter abhängt. Die erlaubten Abhängigkeiten sind jedoch *simpel*, so dass die Zuordnung gelöst werden kann, indem zunächst eine Zerlegung der Aufgaben (task decomposition) und dann eine davon unabhängige Zuordnung (task allocation) erfolgt.

- *Complex Dependencies (CD)*
  Beschreibt komplexe Abhängigkeiten zwischen den UtilityWerten der Pläne (*Complex Tasks*) verschiedener Roboter. Es gibt sowohl interne PlanAbhängigkeiten als auch Abhängigkeiten zu den Plänen anderer Roboter. Da der Utility Wert eines Plans durch die Auswahl eines Plans auf einem anderen Roboter beeinflusst wird, erfordern Probleme dieser Klasse die gleichzeitige Zerlegung und Zuordnung der Aufgaben sowie das Auflösen von Scheduling-Problemen.

Basierend auf dieser erweiterten Taxonomie geben die Autoren eine Einordnung der verschiedenen Frameworks und Forschungsergebnisse, deren Wiedergabe hier zu weit führen würde. Für diese Arbeit relevant ist vor allem die eingeführte Zerlegung von Tasks in elementare, einfache oder zusammengesetzte, da dies eine natürliche Folge der Aufteilung der Aufgaben auf verschiedene Roboter ist.

Ebenfalls 2013 geben Yan et. al [4] eine sehr ausführliche Übersicht über Multi-Roboter-Kooperation, indem sie die jeweiligen Problemstellungen analysieren. Sie untersuchen die Fragestellungen: *single-robot versus multi-robot*, *multi-robot-environment: cooperative versus competitive*, *inherent problem: resource conflict*, *coordination: static versus dynamic*, *communication: explicit versus implicit*, *planning: task planning and motion planning*, *decision-making: centralized Versus decentralized*.

Vergleichsweise aktuell ist die Übersicht des Kapitels *Multiple Mobile Robot Systems* im Springer Handbook of Robotics. Parker et al. geben hier in 2016 erneut

eine Übersicht relevanter Forschungsfragen[20], wobei sich die Themengebiete nicht wesentlich verschoben haben. Sie behandeln die Themen *architectures for multirobot cooperation*, *comminication issues*, *swarm robotics*, *modular robot systems*, *heterogenity in cooperative teams*, *task allocations* und *multirobot learning*. Speziell für heterogene Systeme stellen sie wie Gerkey et al. die Verteilung der Aufgaben (Task Allocation) und Rollenvergabe als eine der zentralen Forschungsfragen in den Mittelpunkt.

In 2019 erstellen Rizk, Awad und Tunstel [21] ein Review kooperativer MRSs. Sie geben dabei nicht nur auf die existierende Forschung ein, sondern geben auch eine Übersicht über Ansätze existierender MRSs und ihrer Schwerpunkt wobei sie diese nach Komplexität der ausgeführten Aufgaben bzw. Autonomie gruppieren. Sie identifizieren eine Vielzahl an offenen Forschungsfragen wie *big-data*, *internet of things*, m*achine und transfer learning*, *human in the loop* und *unified framework*. Für diese Arbeit relevant sind vor allem die identifizierten Bereiche *task complexity*, s*calbility and heterogeneity trade off*, *coalition formation and task allocation*. Die Autoren argumentieren, dass für komplexe Aufgaben eine automatische Zerlegung auf Basis der Roboterfähigkeiten und eine dynamische Aufgaben- und Team-Zusammenstellung erforderlich ist. Insbesondere die dynamischen Teams wurden laut den Autoren unter realen Bedingungen bisher zu wenig beachtet. Bei den existierenden wird bisher die Heterogenität der Systeme gegen die Skalierbarkeit des Systems abgewogen.

In 2023 erstellen Antonyshyn et al. [22] erneut eine Übersicht, vor allem für die kombinierte Lösung der Aufgaben- und Bewegungsplanung. Sie führen eine weitere Taxonomie ein, um Probleme in die Kategorien *task scheduling (TS)*, *task allocation (TA)* und *task decomposition (TD)* einzuordnen. Sie unterscheiden dabei außerdem *path planning (PP)* und *trajectory planning (TP)* als konkrete Kategorien der Bewegungsplanungslösung und *fixed plan (FP)* sowie *adaptive plan (AP)* Lösungen basierend auf ihrer Adaptivität. Sie analysieren existierende Arbeiten und kommen zu dem Schluss, dass insbesondere TS-Arbeiten in der Literatur unterrepräsentiert sind und weiterer Forschung bedürfen. Des Weiteren stellen sie als offene Herausforderungen vor allem die Umsetzung von realen Szenarien und praktischen Applikationen, aber auch die Robustheit gegenüber Roboterausfällen oder die Skalierbarkeit herraus. Zudem bemerken sie, dass die Heterogenität der Fähigkeiten (vor allem für die Bewegung) bisher nicht ausreichend adressiert und das mit spezialisierten Systemen verbundene Potential nicht adäquat genutzt wird.

3.1.1. Einordnung und Bezug zur vorliegenden Arbeit

Die unterschiedlichen Einordnungen zeigen, wie umfangreich das Themengebiet der MRSs ist und dass es sehr verschiedene Arten gibt, sich den Themen zu nähern. Diese Arbeit behandelt vor allem die Frage, wie heterogene Roboter miteinander kooperieren können, wofür vor allem die Frage nach der Verteilung

der Aufgaben (MRTA) die zentrale Rolle spielt. Bemerkenswert ist hierbei, wie sich gerade dieser Themenkomplex seit 1980 entwickelt hat. Während in frühen Arbeiten die Verteilung als trivial angesehen wurde und lediglich als Teilgebiet der Architektur vorkommt, wurde spätestens durch die häufig genutzte Taxonomie von Gerkey et al. [5] deutlich, dass die MRTA eines der wichtigsten Themen ist, zumindest, wenn es um die bewusste Kooperation geht. Kern-Fokus der genannten Lösungsvorschläge ist es stets, eine optimale Verteilung der Aufgaben zu erzielen, was jedoch bei allen komplexeren Fällen NP-Schwer ist. Die Erweiterung der Taxonomie [6] zeigt, dass jedoch selbst die heuristischen Lösungen nicht mehr ausreichen und der Fokus verschiebt sich weg vom reinen Ermitteln einer optimalen Verteilung hin zu der Frage, wie eine Verteilung von Aufgaben überhaupt stattfindet da, auch durch vielen modernen Methoden wie marktbasierte Ansätze, nicht mehr eine streng getrennte Aufteilung und Verteilung stattfinden kann, sondern ein iterativer Prozess nötig ist, bei dem Aufgaben mit wechselnder Granularität aufgeteilt, verteilt und zugeordnet werden. Die häufige Verwendung von greedy Algorithmen trotz der oftmaligen Erwähnung des Fakts, dass sich die Verteilung optimal berechnen lassen, zeigt, dass ein gewisser Pragmatismus für eine reale Umsetzung angebracht ist.

Insbesondere bei heterogenen Systemen, wie sie in dieser Arbeit betrachtet werden, ist es also eine der wichtigsten Fragestellungen, wie Fähigkeiten dargestellt und ausgetauscht werden können. Dies beinhaltet vor allem auch die Frage, wie unterschiedliche Granularitätslevel miteinander kombiniert werden können und wie eine Balance aus vollständiger Modellierung und berechenbarer Verteilung erreicht werden kann. Die Arbeit adressiert damit das in [21] aber auch [22] genannte Problem, dass die Heterogenität bisher nicht ausreichend berücksichtigt wird. Die Frage nach der Berücksichtigung verschiedener heterogener Systeme und unterschiedlicher Autonomielevel ist dabei eng verbunden mit der Frage der Aufgabengruppierung und Zuteilung. Die Trennung von Modellierung und Koordinationsarchitektur wird nicht nur von [21] als Problem identifiziert, sie wird insbesondere auch bei den nachfolgenden Projekten sichtbar.

Der Fokus liegt in dieser Arbeit auf intelligenten, autonomen Systemen, welche miteinander kooperieren, um Fähigkeiten zu tauschen, nicht um in synchronisierter Weise Aufgaben gemeinsam zu erledigen. Es werden daher also primär single-robot Tasks (SR) betrachtet, auch wenn bei der Verteilung einer Aufgabe auf mehrere Robotersysteme teilweise von multi-robot Tasks (MR) gesprochen werden kann. Mit einer Vielzahl an Fähigkeiten, wovon allerdings immer nur eine für das Verteilungssystem relevante aktiv ist behandelt die Arbeit single-task robots (ST). Aufgrund der komplexen Aufgaben und Missionen und einer dynamischen Teamkomposition müssen die Zuordnungen wiederholt überprüft und angepasst werden, es wird also ein time-extended assignment (TA) benötigt. Die Capabilities sind unabhängig, können jedoch Vor- und Nachbedingungen haben, die gegeneinander aufgelöst werden müssen. Insbesondere durch die Kombination von Fähigkeiten durch unterschiedliche Roboter kann von complex dependencies (CD) gesprochen werden, da jedoch meist disjunkte Teile betrachtet und das Scheduling in dieser Arbeit kein Kern-Fokus ist, scheint die Einordnung in

cross-schedule dependencies (XD) realistischer. Damit behandelt die vorliegende Arbeit also ein **XD (ST-SR-TA)** System. Wie auch in anderen Ansätzen wird der TA-Teil vor allem durch eine iterative Zuordnung gelöst, wodurch die Lösung selbst in die Kategorie XD (ST-SR-IA) fällt, obwohl sie komplexere Probleme betrachtet.

3.2. Frameworks für die Multi-Roboter-Kooperation

Nahezu alle Abhandlungen über MRS orientieren sich an wichtigen Frameworks für die Multi-Roboter-Kooperation, welche durch ihr jeweiliges Design einen großen Einfluss auf die Entwicklung des Forschungsbereichs als Ganzes hatten. Im Folgenden sind die Kern-Konzepte einiger der wichtigsten Architekturen dargestellt[1].

3.2.1. ALLIANCE

ALLIANCE [23] ist eines der ersten Frameworks für die Multi-Roboter-Kooperation, welches 1994 von Lynn Parker in ihrer Dissertation [24] „ Heterogeneous Multi-Robot Cooperation“ vorgestellt wurde. Kernaufgabe des Frameworks ist die robuste, fehlertolerante Verteilung von Aufgaben an heterogene Roboter.

Das verhaltensbasierte [25] Framework nutzt zur Verteilung der Aufgaben sogenannte *Behavior sets* (vgl. Abbildung 3.1). Jedes *Behavior Set* kann einer bestimmten Aufgabe zugeordnet werden und gruppiert die dazugehörigen Verhalten, wodurch alle zu einer Aufgabe gehörigen Verhalten gemeinsam unterdrückt oder aktiviert werden können, effektiv also eine Aufgabe bearbeitet wird oder nicht. Um zu bestimmen, ob eine Aufgabe bearbeitet werden soll, werden die *Motivational Behavior* genutzt, welche neben Sensorinput auch die eigenen Verhalten, Kommunikation mit anderen Robotern und vor allem eine interne Motivation (*robot impatience* und *robot acquiescence*) berücksichtigen, um bestimmte *Behavior Sets* zu aktivieren oder deaktivieren. Die *Motivational Behavior* funktioniert folgendermaßen: Ein Roboter hat eine gewisse Motivation, eine Aufgabe zu erledigen, solange per Sensorik erfasst wird, dass die Aufgabe noch erledigt werden muss. Wird die Motivation groß genug, aktiviert der Roboter sein *Behavior Set* und beginnt mit der Aufgabe, was auch an die anderen Roboter kommuniziert wird. Wenn ein anderer Roboter eine Aufgabe bereits ausführt, reduziert das die Rate, mit der die Motivation für die Aufgabe steigt. Dabei bestimmt der Wert der Ungeduld, wie lange die Reduktion anhält. Dadurch wird verhindert, dass Roboter ständig bereits angefangene Aufgaben übernehmen. Gleichzeitig wird das Versagen eines Roboters berücksichtigt, wodurch Aufgaben neu vergeben werden können. Umgekehrt gibt der *Acquiescence-Wert* an, wie lang ein Roboter selbst an seiner

[1] Die Liste ist natürlich nicht vollständig, sondern lediglich ein Auszug.

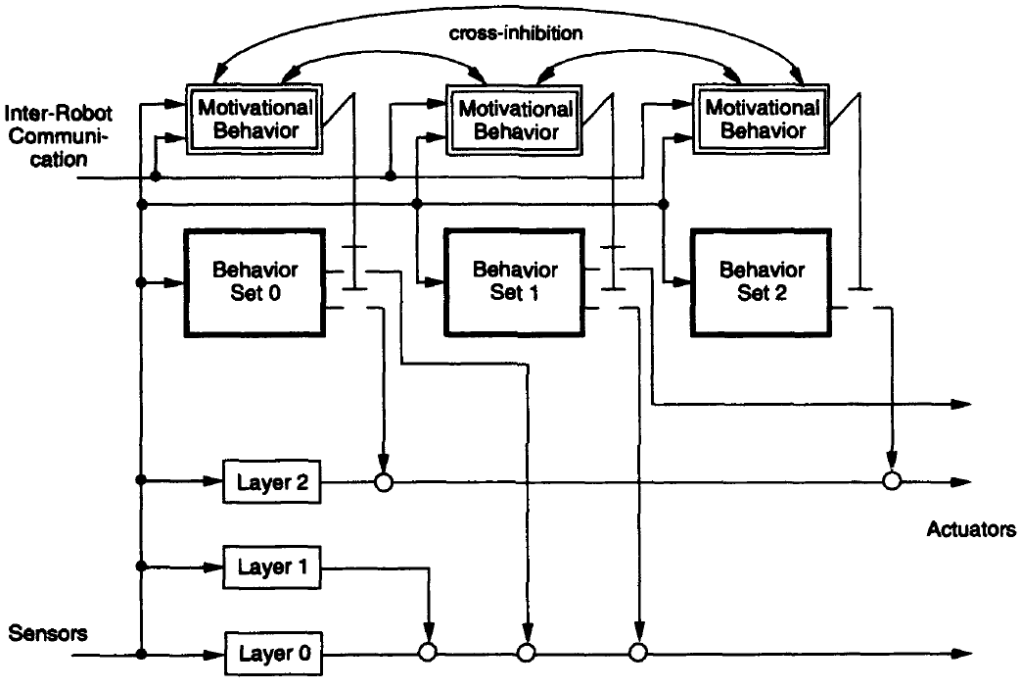


Abbildung 3.1.: Struktur des ALLIANCE Frameworks. Die verhaltensbasierte Architektur zur Vergabe von Aufgaben nutzt *Behavior Sets*, welche durch die *Motivational Behavior* aktiviert oder deaktiviert werden. Quelle: [24]

Aufgabe festhält (wenn er sie nicht erfolgreich ausführt), bevor er sich einer anderen Aufgabe zuwendet. Mit der L-ALLIANCE Erweiterung wurde das System um die Fähigkeit erweitert, die Schwellen für die Aktivierungen selbst zu lernen.

Die Architektur erreicht durch ihre dezentrale, verhaltensbasierte Implementierung und innovativen Ansatz zur Vergabe der Aufgaben ein hohes Maß an Robustheit gegenüber Störungen und hält durch die lediglich unidirektionale Kommunikation den Aufwand für die Koordination äußerst gering. Durch die Möglichkeit, Schwellwerte für die Aktivierungen sowie die Zuordnung der Fähigkeiten zu den *Behavior Sets* kann auch die Heterogenität der Roboter effizient berücksichtigt werden, da außer einem gemeinsamen Verständnis über die zu lösende Aufgabe kein weiteres Modell notwendig ist. Gleichzeitig wird jedoch für jeden Roboter eine Vorab-Modellierung der *Motivational Behaviors* benötigt, welche festlegen, für welche Aufgaben der Roboter eine Motivation zeigen kann. Zudem sind die Verhalten selbst eng mit dem jeweiligen Roboter verknüpft und schwer austauschbar. Die Roboter, ihre Motivation und die entsprechenden Schwellwerte müssen damit jeweils speziell für eine Mission entworfen werden, was die Generalität des Ansatzes einschränkt.

3.2.2. BLE

Broadcast of Local Eligibility - BLE [26] wurde 2000 von Barry Werger und Maja Matarić vorgestellt. Wie auch ALLIANCE ist es eine verhaltensbasierte Architektur, bei der jeder Roboter über eine Menge von Verhalten verfügt, um Aufgaben zu bearbeiten. Um zu entscheiden welcher Roboter eine Aufgabe durchführen darf, berechnet jeder Roboter für das entsprechende Verhalten ein *eligibility* Wert, also ein Utility-Wert, der angibt, wie geeignet der Roboter für die Aufgabe ist.

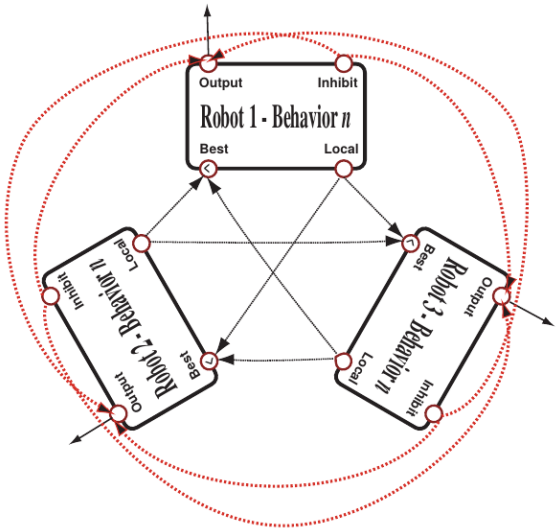


Abbildung 3.2.: Aufbau des BLE Frameworks. Durch die Bekanntgabe der *eligibility* an die anderen Verhalten kann entscheiden werden, welches Verhalten die Aufgabe am besten bearbeiten kann. Die anderen Verhalten werden unterdrückt. Quelle: [26]

Um das BLE System zu nutzen, werden die regulären Verhalten um 3 Ports erweitert: *Local*,*Best* und *Inhibit* (siehe Abbildung 3.2). Jedes Verhalten berechnet seine *eligibility* und kommuniziert diese über den *Local*-Port an alle anderen Verhalten. Diese akzeptieren an ihrem *Best*-Port nur den höchsten ankommenden Wert. Ist der *Local*-Wert größer als der *Best*-wert, wird der *Inhibit*-Port aktiviert und die entsprechenden Verhalten der anderen Roboter unterdrückt.

Die BLE Architektur bietet ähnlich wie ALLIANCE eine verhaltensbasierte Architektur, bei der stets nur ein Roboter mit seinem Verhalten aktiv ist, was den Aufwand der Kommunikation und Koordination der Aufgaben gering hält. Der Vorteil von BLE besteht darin, dass lediglich die Verschaltung der Ports darüber entscheidet, welche anderen Verhalten und damit Aufgaben berücksichtigt werden müssen, was es flexibler und skalierbarer macht als ALLIANCE. Gleichzeitig ist das System extrem simpel, sodass die möglichen Abhängigkeiten und Koordinationsmöglichkeiten zwischen den Robotern stark beschränkt sind. Die Reihenfolge von Tasks kann ebenso wenig berücksichtigt werden wie veränderte

Utility-Werte durch Abhängigkeiten oder aber Fähigkeiten oder Aufgaben, die zur Design-Zeit noch unbekannt waren.

3.2.3. M+

M+ wurde 1999 von Botelho und Alami vorgestellt [27]. Das Framework ist das erste, welches einen marktbasierten Ansatz verfolgt und wurde als zusätzliche Ebene zwischen Missions- und Ausführungsebene in die Architektur auf jedem Roboter eingefügt. Die Missionsebene erzeugt durch einen Planer oder menschliche Vorgabe einen Plan, also eine Abfolge von Aufgaben, die der Roboter zeitlich geordnet durchzuführen hat. Die Ausführungsebene überwacht die korrekte Ausführung und ist in der Lage, kooperative Aufgaben wie Navigation oder Manipulation zu überwachen und Ressourcenkonflikte zu lösen. Die eingefügte M+ Ebene erfüllt 3 Funktionen:*M+ task allocation*, *M+ cooperative reaction* und *M+ execution*. Für die *M+ task allocation* evaluiert jeder Roboter den gegebenen Plan (alle erhalten den gleichen) und wählt aus der Menge der ausführbaren Aufgaben eine beliebige aus. Mit einer modifizierten Version des Contract Net Protokolls [14] handeln die Roboter aus, wer eine Aufgabe ausführt, dabei ist der Roboter, der das initiale Gebot abgegeben hat, verantwortlich für die Kommunikation. Wird währen der Ausführung ein Problem festgestellt, kann der Roboter mittels der *M+ cooperative reaction* andere Roboter um Hilfe fragen. Dafür wird erneut der aktuelle Task auf den Markt gesetzt, allerdings werden auch aktuelle Statusinformationen des anbietetenden Roboters hinzugefügt. Die anderen Roboter prüfen, ob sie den Task unter den gegebenen Bedingungen mit ihrem aktuellen Task vereinbaren können und geben wenn möglich ein Gebot ab, um die Aufgabe zu übernehmen. Die *M+ execution* synchronisiert die Ausführung mit anderen Robotern auf Task-Ebene, indem Start und Ende einer Aufgabe mitgeteilt werden. Zudem wird in Falle einer Übernahme bei einem Fehler die Aktion zwischen beiden involvierten Robotern koordiniert.

Als erste Architektur verwendet M+ einen marktbasierten Ansatz für die Aufgabenverteilung. Durch die Möglichkeit, Aufgaben bei einem Problem weiter zu verteilen, ist eine große Flexibilität und Fehlertoleranz gegeben. Im Vergleich zu den verhaltensbasierten Ansätzen erhöht sich der Kommunikationsaufwand dadurch erheblich. Zudem ist das System weniger stark auf heterogene Systeme ausgelegt. Jedes System benötigt die gleiche Architektur und das gleiche Verständnis der Aufgaben um eine Verteilung auf dieser Art möglich zu machen, was gerade bei heterogenen Systemen einen größeren Aufwand während der Design-Zeit bedeutet.

3.2.4. MURDOCH

MURDOCH [28] wurde 2002 von Brian Gerkey und Maja Matarić vorgestellt. Während die vorher genannten Architekturen eine iterative Zuweisung nutzen,

betreibt die marktbasierte Architektur MURDOCH eine online Aufgabenverteilung mittels Publish-Subscribe-Architektur. Die Autoren argumentieren, dass durch die Natur eines realen Systems die Welt und die Aufgaben nicht genau genug modelliert werden könnten, als dass ein Planungsansatz einen signifikanten Mehrwert erzielen kann, und daher eine greedy online Zuweisung optimal ist. Ebenso führen sie die Publish-Subscribe basierte Kommunikation als Methode zur Aufgabenverteilung ein, bei der der Fokus auf einer Ressource bzw. Aufgabe und nicht dem jeweiligen Teilnehmer liegt. Durch das Abonnieren einer Ressource kann ein Roboter anzeigen, dass er über eine bestimmte Fähigkeit verfügt (etwa "Kamera") oder einen bestimmten Zustand (etwa "halte Objekt") besitzt. Aufgaben werden verteilt, indem Nachrichten per Broadcast an die dafür nötigen Ressourcen geschickt werden. Da die Aufgaben bekannt gegeben werden, indem die Ressourcen angesprochen werden, bekommen nur Roboter die derzeit verfügbar sind und über die geeigneten Ressourcen verfügen, die Anfrage. Die Anfrage beinhaltet eine Metrik, die jeder Teilnehmer berechnet und als Kosten für die Aufgabe veröffentlicht. Nach einer festgelegten Zeit erhält der Roboter mit den besten Kosten den Zuschlag durch eine spezifische Nachricht, alle übrigen Roboter melden sich wieder für die Annahme von Aufgaben verfügbar. Während der Ausführung überwacht der Auktionsanbieter die Ausführung durch periodische Erneuerungs-Nachrichten. Antwortet der ausführende Roboter auf diese nicht, wird von einem Fehler ausgegangen und die Aufgabe neu vergeben. Das gesamte System ist dezentral implementiert, so dass jeder Roboter in der Lage ist, Aufgaben anzubieten oder entgegen zu nehmen.

MURDOCH trifft viele Annahmen, die in anderen Frameworks nicht vorhanden sind. Die Autoren gehen explizit davon aus, dass Roboter jederzeit dem Team hinzugefügt werden können oder verschwinden, was sie durch das Nutzen von Broadcast-Nachrichten und die ressourcenorientierte Publish-Subscribe-Kommunikation umgehen können. Da Aufgaben anonym vergeben werden, ist nicht relevant, welcher der Roboter die Aufgabe annimmt, was die Flexibilität und Skalierbarkeit des Ansatzes gegenüber anderen deutlich verbessert. Das verwendete Auktionsschema ist simpel aber effektiv und durch die periodische Erneuerung des Vertrages zwischen Auktionator und Bieter können fehlerhafte Roboter schnell kompensiert werden. Die Darstellung der Tasks über die Ressourcen ist jedoch zugleich nicht trivial. Werden die Aufgaben komplexer, ist eine höher angelegte Zerlegung vermutlich erforderlich. Gleichzeitig können komplexere Anforderungen nur schwierig abgebildet werden ohne ein darauf spezialisiertes System anzusprechen was komplexere Missionen wiederum komplizierter macht.

3.2.5. ASyMTRe

ASyMTRe - Automated Synthesis of Multirobot Task solutions through software Reconfiguration [29] ist ein Framework von Fang Tang und Lynn Parker von 2005, welches einen komplett neuen Weg der Koordination verfolgt. Anstatt eine

Aufgabe auf einem sehr hohen Level zu beschreiben, in kleine Blöcke zu zerlegen und dann auf die Roboter zu verteilen, wird der umgekehrte Ansatz verfolgt und ausgehend von einer Aufgabenbeschreibung die nötigen Schritte für die Ausführung synthetisiert. Durch die Synthese einer Fähigkeit zur Bearbeitung der Aufgabe werden jedes Mal alle verfügbaren Resourcen genutzt, was z.B. erlaubt, dass ein Roboter ohne die Fähigkeit sich zu lokalisieren zu einem Ziel navigieren kann, da die Lokalisierungskomponente eines anderen Roboters eingesetzt werden kann.

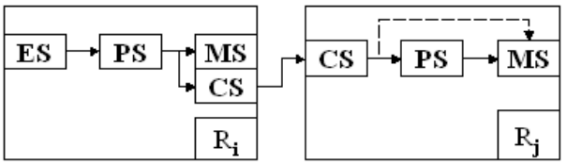


Abbildung 3.3.: Verknüpfung der Schemas im ASyMTRe Framework. Die *environmental sensors ES* nehmen Daten auf und verarbeiten diese mittels der *perceptual schemas PS*, um die Daten für das Ausführen eines *motor schemas MS* zu nutzen oder per *communication schema CS* an andere Roboter zu übertragen. Quelle: [29]

Dazu werden in ASyMTRe eine Reihe von Funktionsblöcken verwendet: *environmental sensors*, *perceptual schemas*, *communication schemas* und *motor schemas*. Aufgaben werden als parametrisierte *motor schemas* definiert, die Fähigkeiten der Roboter als *environmental sensors*. Um eine Aufgabe zu erfüllen, werden die Roboter zunächst nach ihren Fähigkeiten (Sensoren) sortiert. Dann werden die Funktionsblöcke genutzt um eine Fähigkeit für die Aufgabe auf dem am wenigsten Fähigen Roboter zu synthetisieren. Wenn bei der Erstellung festgestellt wird, dass eine Vorbedingung, etwa aufgrund eines fehlenden Sensors, nicht durch den Roboter selbst ausgeführt werden kann, wird der nächste, fähigere Roboter geprüft und wenn möglich dessen *environmental sensor* verwendet. Durch die *perceptual schemas* werden Berechnungen durchgeführt, etwa Sensorergebnisse in nutzbare Daten Transformiert. Mittels der *communication schemas* können Daten zwischen den unterschiedlichen Robotern ausgetauscht werden. Die Synthese einer Aufgabe wird ähnlich wie das Erstellen eines Planes mittels STRIPS-planer durchgeführt, indem, ausgehend von dem gewünschten Ergebnis, alle Vorbedingungen sukzessive hinzugefügt werden.

3.2.6. TraderBots und zusammengesetzte Aufgaben

TraderBots [30] wurde von Dias in 2004 in ihrer Dissertation als Framework für die marktbasierte Multi-Roboter-Koordination in dynamischen Umgebungen vorgestellt und ist seither eine der wichtigsten Referenzen für marktbasierte Ansätze. Jeder Roboter ist mit einem expliziten *trader*-Modul ausgestattet, welches mit

den anderen Team-Teilnehmern über das Contract-Net-Protokoll [14] kommuniziert. Über das Modul werden Aufgaben zwischen den Händlern zur Versteigerungen angeboten und Gebote auf Aufgaben abgegeben. Jedes System tritt als eigennütziger Händler auf, der versucht, seinen eigenen Profit zu maximieren. Da aller *Gewinn* in der Mission nur durch das Erreichen von Gesamtzielen möglich ist, versucht daher jedes System seine Kosten zu minimieren, indem Tasks an andere Teilnehmer gegeben werden, die besser geeignet sind. Dadurch wird die Ausführung der Gesamtmission optimiert. Anders als MURDOCH, welches eine Auktion mit sofortiger Zuweisung nutzt (i*nstantaneous assignment-IA*), werden Zuweisungen andauernd optimiert (*time-extended assignment-TA*). Jeder Händler hält ein Portfolio seiner erworbenen Aufgaben und ihrer (zeitlichen) Abhängigkeiten sowie seiner aktuell laufenden Aufgaben. Aufgaben, die noch nicht begonnen wurden, können per Unterauftrag (*subcontract*) vergeben oder verkauft (*transfer*) werden. Bei Unteraufträgen wird der Status der Aufgabe an den übergeordneten Händler gemeldet und erst bei Erledigung der Aufgaben gezahlt, bei einem Verkauf wird der Gewinn sofort ausgeschüttet und kein Status an den verkaufenden Roboter gemeldet. Gebote und Verträge können so lange verändert werden, bis ein Gebot nach einer Deadline final akzeptiert wurde. Fehler während der Ausführung können teilweise auch von den ausführenden Systemen selbst erkannt werden, woraufhin eine erneute, proaktive Vergabe der Aufgaben erfolgt. Die Haupteinschränkung des Systems ist die Vergabe von *einfachen* Aufgaben, die von einem System allein ausgeführt werden können.

Zlot erweitert das Framework daher 2006 in seiner Dissertation um komplexe Aufgaben [19]. Er schlägt für die Modellierung Aufgabenbäume (task trees) (Abbildung 3.4) vor. Eine Aufgabe wird durch Teil- oder Alternativaufgaben in einem hierarchischen Baum dargestellt. Die Eltern-Kind-Beziehung sind *UND*/*ODER*-Verknüpfungen zwischen den Teilaufgaben. Vor- und Nachbedingungen können ebenfalls als Bedingungen dargestellt werden, während parallele Ausführungen von gemeinsamen Aufgaben zwar möglich sind, aber eine weitere Erweiterung benötigen. Jede Ebene des Baums ist eine weitere Abstraktionsebene. Während der Wurzelknoten eine abstrakte bzw. semantische Beschreibung der Aufgabe ist beinhalten die Blatt-Knoten spezifische, einfache Aufgaben, die ein Roboter allein ausführen kann.

Durch die Erweiterung können in TraderBots nicht nur einfache Aufgaben sondern auch komplette Bäume oder aber Teile davon vergeben werden. Dadurch wird die Vergabe und insbesondere die Entscheidung, welcher Händler die Auktion gewinnt, deutlich komplexer, aber auch effektiver. Die Knoten der Bäume werden einzeln durch die jeweiligen Händler bewertet, welche entweder die vorgeschlagene oder aber eine eigene Zerlegung nutzen können. Jeder Händler kann dann auf eine Menge an Knoten bieten während der Auktionator versucht, die niedrigsten Kosten für das gesamte Team zu erzeugen. Wie zuvor sind die Gewinner für die Überwachung der Ausführung zuständig. Wie in [6] noch einmal aufgegriffen, wird durch diese Art der Zerlegung und Vergabe für komplexe Aufgaben berücksichtigt, dass eine inititale Zerlegung und nachfolgende Zuweisung nicht immer möglich ist, da Abhängigkeiten auch erst während der Ausführung

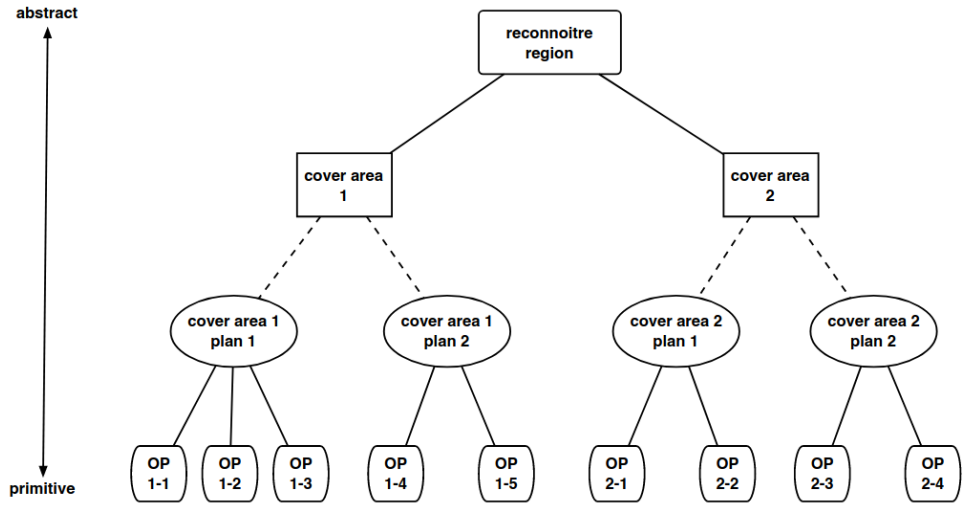


Abbildung 3.4.: Taskbaum nach Zlot für eine Aufklärungsmission. Die durchgehenden Linien sind *UND*-Verknüpfungen, die gestrichelten *ODER*-Verknüpfungen. Durch die Hierarchie werden die Aufgaben in immer feinere Teilschritte zerlegt, die *ODER*-Verknüpfungen ermöglichen alternative Pläne. Quelle: [31]

auftreten können. Dadurch werden selbst initial nicht so modellierte Aufgaben teilweise zu komplexen Aufgaben, die eine solche integrierte Zerlegung und Zuweisung benötigen.

3.2.7. Einordnung und Bezug zur vorliegenden Arbeit

Obwohl die vorgestellten Frameworks bereits älter sind, sind sie weiterhin relevant, da sie die Kernprinzipien verschiedener Ansätze aufzeigen. Neuere Lösungen sind nach gleichen Prinzipien, jedoch anwendungsspezifischer aufgebaut.

Die verhaltensbasierten Ansätze sind reaktiv, robust und erfordern wenig Kommunikation. Dadurch sind sie insbesondere auch sehr unabhängig von der Art der Systeme und Aufgaben, weshalb sie eine gute Wahl für heterogene Systeme sind. Gleichzeitig sind Verhalten kaum zur Laufzeit erweiterbar und müssen meist auf spezielle Missionen abgestimmt werden, weshalb sie die aufgestellten Anforderungen an ein dynamisches Team und dynamische Aufgaben nicht erfüllen können. Marktbasierte Ansätze hingegen zeigen gerade bei dynamischen Aufgaben und dynamischen Teams ihre Stärke, da sie die Zuordnung von Fähigkeiten zu Aufgaben ständig neu ermitteln und durch die Kostenermittlung auch die individuelle Robotereigenschaften und Umstände berücksichtigen können. Die Komplexität verschiebt sich hier hin zur Frage, wie Aufgaben und Fähigkeiten modelliert werden können, um eine effektive Zuordnung zu erreichen. Gerade für heterogene Systeme ist die Notwendigkeit einer einheitlichen Modellierung über das Team hinweg eine Herausforderung. Die zweite und dritte

Forschungsfrage dieser Arbeit (vgl. Abschnitt 1.1) zielen daher genau auf diese Fragestellung als Schlüssel ab. Ein marktbasiertes Verfahren in Verbindung mit einer geeigneten Modellierung verspricht die größte Flexibilität und Dynamik für ein Team. Zlot zeigt mit den komplexen Aufgaben eindrücklich, dass eine Zerlegung und Zuweisung der Aufgaben auf unterschiedlichen Ebenen einen signifikanten Vorteil bietet, was die Nutzung von Behavior Trees als Darstellungsform motiviert hat. Diese bieten die Vorteile einer sehr feingranularen Zerlegung der Aufgabe wie schon mit den Aufgabenbäumen. Gleichzeitig modellieren sie jedoch auch alle relevante Abhängigkeiten und Zusammenhänge der Aufgaben so das diese direkt ausführbar sind. Synthese-Ansätze wie *ASyMTRe* verfolgen eine Bottom-Up-Philosophie, wodurch insbesondere sehr unterschiedliche Fähigkeitsausprägungen berücksichtigt werden können. Wie bei der Zuordnung der Fähigkeiten selbst ist die Granularität insbesondere bei heterogenen Teams hierbei eine Herausforderung, um nennenswerte Missionen erfüllen zu können. Die vierte Forschungsfrage dieser Arbeit zielt daher auf diese Fragestellung.

3.3. Relevante Forschungsprojekte

Eine Vielzahl von Projekten, vor allem auf europäischer Ebene, befasst sich mit der Koordination und Kooperation von Robotern in Teams. Insbesondere bei Projekten mit UAVs spielt die Koordination von heterogenen Roboter-Teams oft eine große Rolle, da meist sehr unterschiedliche Systeme entkoppelt voneinander eingesetzt werden und die Kooperation mit bodengebundenen Systemen sehr wünschenswert ist. Projekte, die sich nur mit Bodenrobotern beschäftigen, legen den Fokus oft auf Exploration und gemeinsame Kartenbildung oder die Entwicklung von Koordinationsmechanismen im Schwarm und damit weniger stark auf die Koordination der heterogenen Systeme. Im Folgenden werden einige relevante Projekte und insbesondere deren entwickelte Architekturen und Modellierungen vorgestellt, um einen Einblick in die in der Praxis verwendeten Konzepte zu erhalten[2]. Es wurden vor allem Projekte gewählt, deren Architekturen öffentlich sind und die einzelne Aspekte gut darstellen.

3.3.1. COMETS und AWARE

Das EU-Projekt COMETS (2002-2005) [32] hat das Ziel eine verteilte Kontrollarchitektur zu entwickeln um mittels kooperativer UAVs Waldbrände zu detektieren und zu überwachen. Kernthemen sind vor allem die Sensorfusion, Echtzeit-Bildauswertung und die verteilte Kontrolle von verschiedenen Flugrobotern wobei ein spezieller Fokus auf die Robustheit des Gesamtsystems gelegt wurde. Das

[2]Die Liste der Projekten hat keinen Anspruch auf Vollständigkeit, die gewählten Projekte geben jedoch einen guten Überblick über relevante Konzepte. Andere Projekte verfahren oft ähnlich.

Nachfolger-EU-Projekt AWARE[3] (2006-2009) [33], führte die Arbeiten für fliegende Teams fort, fokussierte dabei aber vor allem in Gebiete ohne Kommunikationsinfrastruktur in denen Sensornetzwerke zur Lageerkundung aufgebaut werden sollten.

Ein wichtiger Aspekt beider Projekte ist die Frage wie heterogene UAV (etwa ein Helikopter und ein Luftschiff), bezogen sowohl auf die physikalischen Eigenschaften, wie auch auf ihre Fähigkeiten, in einem MRS genutzt werden können und wie Aufgaben unter den Teilnehmern koordiniert werden können. Da die Projekte damit im Kern die gleiche Fragestellung wie diese Vorliegende Arbeit behandelt werden Aspekte der entwickelte Architektur im Folgenden ausführlicher dargestellt.

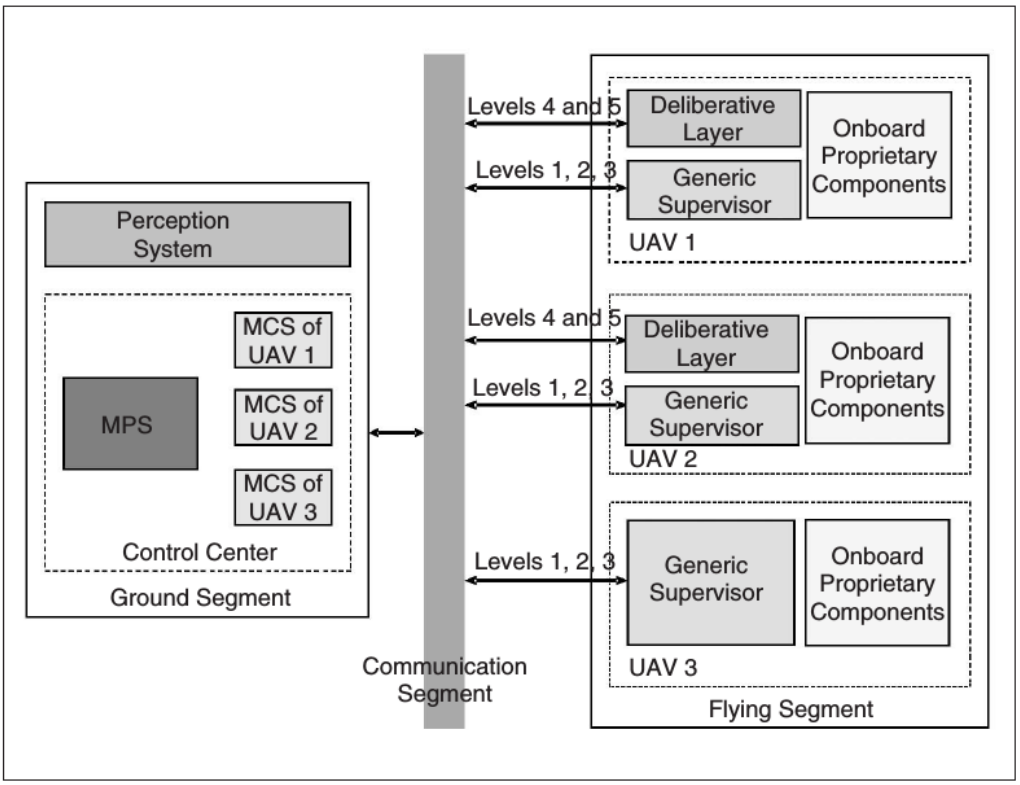


Abbildung 3.5.: Generelle Architektur des COMETS Projektes. Das Bodensegment übernimmt die Planung (MPS) Überwachung (MCS) der Aufgabenausführung und fusioniert die Daten (Perception System). Jedes Flugsegment besteht aus den proprietären Komponenten des Fluggerätes, einem Modul zur Ausführung generalisierter Aufgaben (Generic Supervisor) sowie der Ausführung komplexerer Missionen (Deliberative Layer). Quelle: [32]

[3]Platform for **A**utonomous self-deploying and operation of **W**ireless sensor-actuator networs cooperating with **A**e**R**ial obj**E**cts

Architektur und Bodenstation

Das COMETS System besteht aus drei unterschiedlichen UAV-Plattformen, welche verschiedene Fähigkeiten sowie Autonomiegerade von Teleoperation bis hin zu autonomer Aufgabenausführung besitzen und per Funk mit einer Bodenstation kommunizieren. Die Bodenstation (Abbildung 3.5) besteht aus dem mission planning system (MPS), welches vom Nutzer vorgegebene Missionen in primitive Aktionen zerlegt, die direkt durch die UAV ausführbar sind, je einem monitoring and control system (MCS) pro UAV, welches die Ausführung der aktuellen Mission und Aufgaben überwacht und wenn nötig mit Korrekturen eingreift und dem perception system (PS) welches die Daten der verschiedenen UAVs fusioniert. Jedes UAV ist neben seinen proprietären Komponenten zur Steuerung mit einem generic supervisor ausgestattet, welcher die möglichen Handlungen des Systems abstrahiert und die Schnittstelle zu anderen UAVs und der Bodenstation vereinheitlicht. Falls das System über Entscheidungsautonomie verfügt, wird diese über die deliberative layer angesprochen.

Zur Zerlegung der Mission wird die reactive tabu search metaheuristic [34] eingesetzt und mittels Konfliktanalyse optimiert, wodurch für jeden Roboter konfliktfreie, ausführbare Aufgaben generiert und kommuniziert werden. Um die Heterogenität der Systeme abzubilden, wird eine Datenbank mit Modellen verwendet, welche geeignete Parametersätze für jede Kombination aus System und Aufgabe enthalten. Diese Modelle werden auch bei der zentralen Aufgabenplanung, etwa der Pfadplanung, verwendet, um die Charakteristiken der jeweiligen Systeme zu berücksichtigen.

Autonomie und Robotersystem

Zur Einordnung der Autonomie jedes System werden in COMETS ([32, 35]) die folgenden Level an Autonomie definiert (deutsche Bezeichnungen ergänzt):

- Level 1: Keine Autonomie
  Der Roboter ist lediglich in der Lage, *elementare Aufgaben* auszuführen
- Level 2: Ausführungs-Autonomie
  Der Roboter ist in der Lage, teilweise geordnete Sequenzen an Aufgaben selbstständig abzuarbeiten und den Status zu melden.
- Level 3: koordinierte Ausführungs-Autonomie
  Wie Level 2, der Roboter kann jedoch ebenfalls mit anderen Systemen des gleichen Levels kommunizieren und Aktionen synchronisieren.
- Level 4: Planungs-Autonomie
  Der Roboter ist in der Lage, deliberativ zu handeln und kann Aufgabenplanung oder Wegplanung durchführen. Aufgaben werden mit anderen Robotern des gleichen Levels koordiniert.

- Level 5: Volle Autonomie
  Wie Level 4. Zusätzlich kann der Roboter auch Aufgaben umplanen und Aufgaben von anderen Robotern übernehmen.

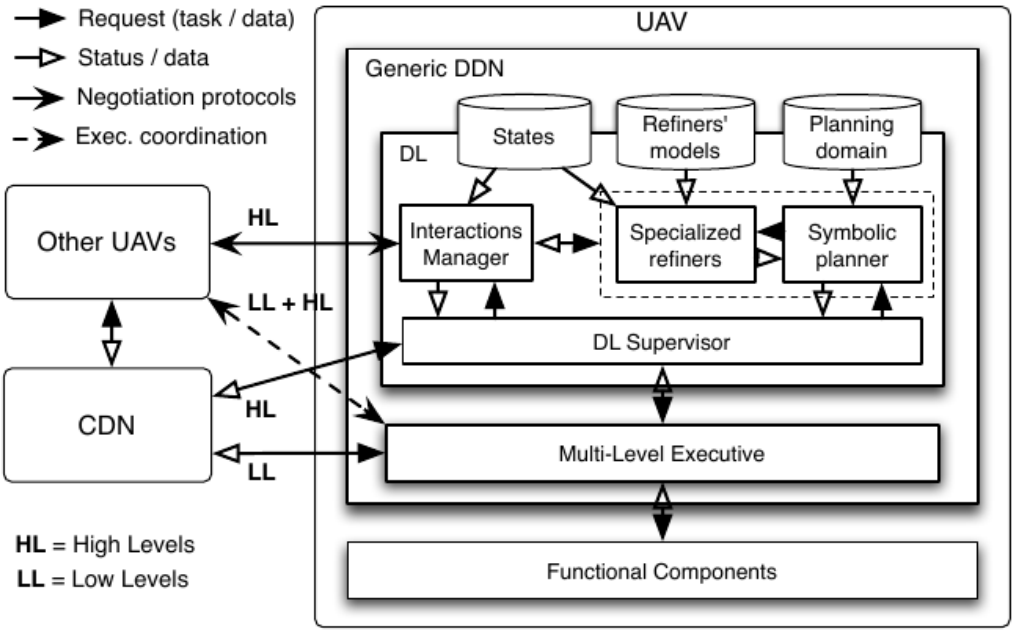


Abbildung 3.6.: Die Distributed Descisional Node (DDN) in der COMETS Architektur verbindet die Agenten auf ihrem jeweiligen Autonomielevel. Quelle: [35]

Jedes UAV des COMETS-Systems ist mit einer distributed decisional node (DDN) (Abbildung 3.6) ausgestattet, welche die unterschiedlichen Level der Autonomie in das Gesamtsystem einbindet. Die Multi-Level-Executive (MLE) ist die Abstraktionsschicht für den Roboter. Sie erlaubt die Ausführung von Aufgaben auf jedem Level, wobei sie bei Level 1 Systemen nicht viel mehr als eine Weiterleitung der Aufgabe ist, während bei Level 3 Systemen Aufgaben bereits mit anderen Systemen koordiniert werden. Für Aufgaben der Level 4 und 5 wird die Deliberative Layer (DL) genutzt, welche mehrere Aufgaben erfüllt. Der symbolische Planer zerlegt Missionen in teilweise geordnete Pläne von Aufgaben. Die specilaized refiners bilden den symbolischen Plan auf die Realität ab, indem geeignete Modelle verwendet werden. Die eingesetzten Planer ähneln dabei HTNs. Der Interactions Manager koordiniert high level tasks mit anderen UAVs und der Supervisor überwacht die komplette Ausführung und leitet erstellte Taskslisten an den MLE weiter. Zur Koordination der Aufgaben kommen zum einen einfache räumliche Ausschlusskriterien zum Einsatz, zum anderen eine erweiterte Form des Contract Net Protocolls [36]. Die Kommunikation zwischen den Systemen wird über eigens entwickelte Protokolle basierend auf Blackboard Kommunikation realisiert.

Im AWARE Projekt wird weitgehend die gleiche Architektur verwendet, wenngleich etwas weniger stringent strukturiert [37]. Kern ist weiterhin eine On-Board

Deliberative Layer (ODL), welche die Koordination zwischen den Systemen regelt und einer Executive Layer (EL), welche die proprietäre Hardware kapselt.

Aufgabenmodell

In [35] werden das Aufgabenmodell des DL sowie die Primitiven des MLE genauer erläutert. Aufgaben werden innerhalb der DL Eventbasiert dargestellt und lösen bei ihrer Ausführung ein Start-Event und bei Beendigung ein End-Event aus. Zudem verfügen Aufgaben über Vor- und Abbruchbedingungen, welche sowohl zeitbasiert als auch auf andere Tasks und externe Trigger bezogen sein können. Die von der MLE ausführbaren Tasks sind eng umrissen: Take-Off, Lang, Go-To, Take-shot, Wait sowie ein Spezialtask Synchronize. Für die Planung der Aufgaben kommt der Shop2 HTN Planer [38] zum Einsatz, welcher durch Ausnutzen von Multi-Timeline-Preprocessing (MTL) auch zeitliche Abfolgen und vor allem Parallelitäten unterstützt.

AWARE nutzt die gleichen Befehle wie COMETS, berücksichtigt jedoch stärker Aspekte unterschiedlicher Payloads und führt spezielle Befehle wie *MERGING* ein, um Konflikte mit anderen Agenten aufzulösen. Die Allokation der Aufgaben erfolgt ebenfalls über einen marktbasierten Ansatz, jedoch auf Basis des SET-Algorithmus [39]. Zudem werden Missionspläne mittels EUROPA Planer [40] erstellt und optimiert.

Einordnung und Bezug zur vorliegenden Arbeit

Die Projekte COMETS und AWARE zeigen die Anwendbarkeit eines Frameworks für heterogene Robotersysteme außerhalb von reinen Laborbedingungen und binden dabei effektiv nicht nur unterschiedliche Systeme, sondern auch Autonomiegrade ein.

Die definierten Autonomielevel erscheinen zunächst sinnvoll, führen jedoch zu einer sehr unterschiedlichen Behandlung in der DDN getauften Ausführungsschicht auf dem Robotersystem. In Anlehnung an HTNs werden Aufgaben aus Aufgabenprimitiven (Go-To, Land, Take-Shot,etc.) aufgebaut, wodurch eine einheitliche Sprache zwischen Bodenstation und Roboter sowie den Robotern selbst geschaffen wird. Die Primitiven abstrahieren das proprietäre Kontrollsystem des Roboters (in der MLE) und vereinheitlichen so die heterogene Struktur der Systeme. Ein Go-To kann also in gleicher Form an jedes Robotersystem kommuniziert werden, die spezielle Ausprägung des Kommandos wird lediglich über eine geeignete Parametrisierung aus einer zur Laufzeit anpassbaren Datenbank geladen. Dadurch werden die Autonomielevel 1-3 direkt miteinander verbunden.

Die Abstraktion der Mission und Fähigkeiten ist ein zentraler Aspekt der Multiroboter-Koordination und daher für die vorliegende Arbeit sehr relevant. Die

Systeme werden bei diesem Ansatz jedoch stark vereinfacht modelliert. Die Fähigkeitsprimitiven stellen den kleinsten gemeinsamen Nenner an Fähigkeiten dar, die die Systeme miteinander teilen. Sie bieten selbst keine Möglichkeiten zu einer unterschiedlichen Ausführung, wodurch die Heterogenität der Systeme effektiv auf einen Satz homogener Fähigkeiten reduziert wird. Wenngleich effektiv, beschränkt diese Form der Modellierung der Fähigkeiten die Möglichkeiten des Roboterverbundes und führt außerdem zu Schwierigkeiten, wenn stärker heterogene Missionen, etwa das Greifen von Gegenständen, berücksichtigt werden sollen. Die vorliegende Arbeit will diesen Effekt explizit vermieden. Die vorgeschlagene Strukturierung der High Level Schicht (DL) sieht eine deutlich aufwendigere Koordinierung der Aufgaben vor, bei der mittels eines marktbasierten Ansatzes Aufgaben verteilt werden können. Auch wenn an dieser Stelle durch Vor- und Nachbedingungen sowie die speziellen refinement-planer die heterogenen Eigenschaften erneut eingebaut werden können unterschieden sich die genierten Pläne lediglich durch Parametrisierung. Die Autonomieschicht 4 und 5 werden signifikant anders behandelt. An dieser Stelle wäre eine engere Verzahnung der Planungsschicht (in diesem Fall DL) mit der Ausführungsschicht (in diesem Fall MLE) flexibler. Insbesondere muss es eine Möglichkeit geben, auch heterogene Fähigkeiten in einem einheitlichen System mit abzubilden. In dieser Arbeit wird daher ein Ansatz verfolgt der die Simplizität der gemeinsamen Sprache ausnutzen kann, dabei aber nicht die Komplexität der individuellen Systeme reduziert.

Die Möglichkeit, sowohl zentral zu planen, als auch die Aufgaben ohne weitere Zerlegung an die Teilnehmer zu geben, erlaubt eine große Flexibilität, die ebenfalls eine wichtige Voraussetzung für den Einsatz solch heterogener Systeme ist. Das zentrale Planen durch jedes einzelne System ist dabei allerdings nicht ohne Risiko, insbesondere, wenn Ressourcenkonflikte oder nichtdeterministische Ergebnisse ein Problem darstellen. Wie auch für die Komplexität ist vor allem eine möglichst durchgehend gewählte Modellierung ein Ansatz, der diese Umstände verbessern könnte.

Die Netzwerkaspekte des AWARE Projekts sind klassische Themen des Routings und der Datenaggregation in Sensornetzwerken mit begrenzter Bandbreite und Leistung. Die entwickelten Techniken sind daher zwar für MRS anwendbar, aber nur bedingt relevant, wenn von einer hohen Bandbreite ausgegangen werden kann.

3.3.2. ARCAS

Das EU Projekt ARCAS (2007 - 2013) ist aus dem COMETS Projekt hervorgegangen, weshalb etliche Konzepte übernommen bzw. weitergeführt wurden. In dem Projekt werden Flugroboter, ausgestattet mit Armen, dazu befähigt, verschiedene Manipulationsaufgaben zu erfüllen, etwa das Zusammenstecken von Plastikröhren. Das Projekt behandelt dabei eine Vielzahl spezieller Themen für die

Konstruktion, wie etwa eine Kompensation der Greifkräfte während des Fluges oder den kooperativen Transport von Bauteilen. Für die Koordination des Multi-Roboter-Systems werden zwei sehr unterschiedliche Systeme eingesetzt: Die lose gekoppelte Koordination baut auf den vorherigen Ergebnissen auf und nutzt Primitive für die Koordination, erweitert diese jedoch um ein Missionsparsing und ontologie-unterstützte Planung. Die eng gekoppelte Koordination hingegen präsentiert einen Ansatz zur Kombination verschiedener Verhalten.

Lose gekoppelte Koordination

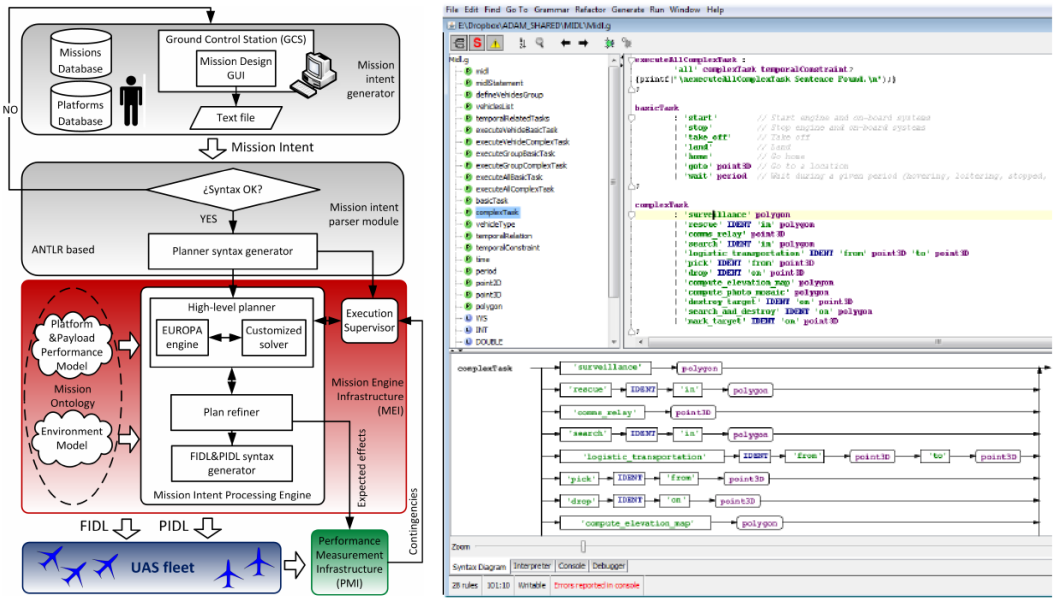


Abbildung 3.7.: High Level Architektur für ein heterogenes Team aus Flugrobotern im Rahmen des ARCAS Projektes. Links: Erstellen individueller Aufgabenpläne aus einer Missionsvorgabe. Eine Mission wird definiert, auf korrekte Syntax überprüft und dann mittels Planer und Ontologiewissen in Einzelpläne überführt. Rechts: ANTLER Werkzeug zum Definieren einer Missionssyntax. Durch Terme wie *search IDENT in polygon* werden Missionen für das gesamte Team definiert. Die erstellte Syntax wird für die Definition der Mission genutzt und kann auf Korrektheit geprüft werden. Quelle: [7]

Im ARCAS wurde eine Koordinationsarchitektur (siehe Abbildung 3.7 links) für eine Vielzahl an Flugrobotern außerhalb einer Formation [7] entwickelt. Das System setzt auf Missions- bzw. Aufgabenbeschreibungssprachen, in diesem Fall die von Boeing Research & Technology Europe entwickelte *FIDL (Flight Intention Description Language)*, um einen Plan von der Missionseinheit an ein UAV weiterzugeben, welches den zu diesem Zeitpunkt noch generell gehaltenen Flugplan durch vorgegebene Missions- und Systemeigenschaften zu einem konkreten Flugplan und nachfolgend in Steuerkommandos umwandelt.

Die Mission Engine Infrastructure (MEI) erweitert das Konzept um die Möglichkeit, eine Missionsdefinition (siehe Abbildung 3.7 rechts) zu verwenden oder per GUI zu erstellen, die dann von einem ANTLR[4] Parser [41] auf Korrektheit geprüft und ggf. iterative verbessert wird. Eine gegebene Mission wird dann erneut mit dem *EUROPA Planer* [40] in eine symbolische Planung umgesetzt, in der Konflikte und zeitliche Randbedingungen zentral aufgelöst werden. Für die Planung und Verifikation werden Missions-Ontologien verwendet und Konflikte mittels Backtracking aufgelöst. Der High-Level Plan wird im Folgenden vom *Plan Refiner* noch um Ausführungsdetails verfeinert, die der Planer nicht automatisch erstellt und abschließend vom *FIDL&PIDL syntax generator* in die FIDL-Syntax übersetzt. Zuletzt wird die Ausführung überwacht und ggf. werden Notfallpläne ausgelöst.

Eng gekoppelte Koordination

Für die Koordination von mehreren UAVs inklusive Manipulator wurde die CAVIS (Control software Architecture for cooperative unmanned aerial VehiclemanIpulator Systems) Architektur [42, 43] entwickelt. Grundgedanke des Systems ist es, die Komplexität des Gesamtsystems durch Aufteilen in elementare Verhalten (*elementary behaviors*) beherrschbar zu machen und diese wiederum mittels Null-Space based Behavioral (NSB) Ansatz [44] zu kombinieren.

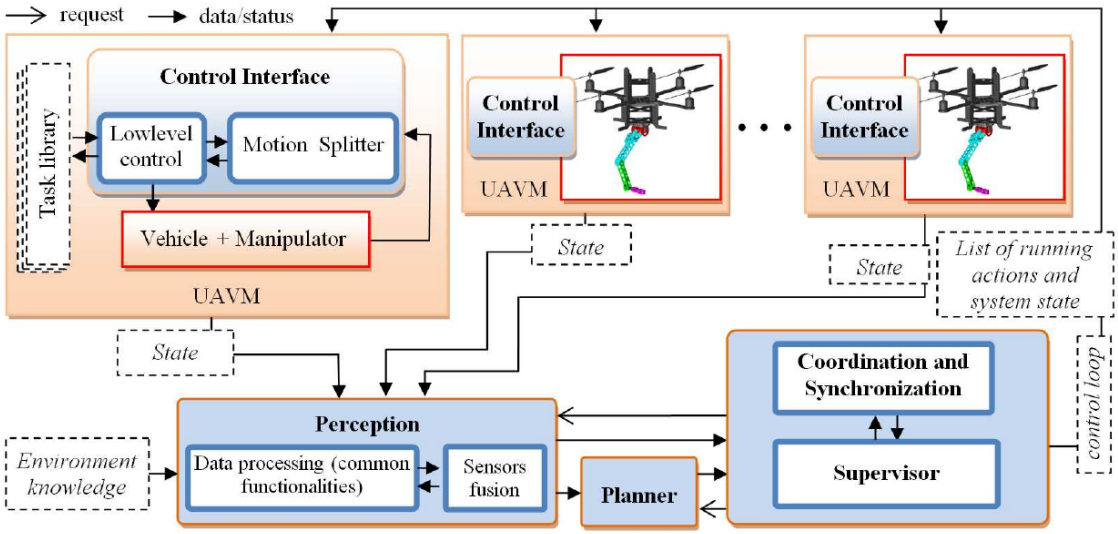


Abbildung 3.8.: Softwarekomponenten der CAVIS Architektur für eine enge Kopplung kooperativer Aufgaben. Verschiedene Verhalten (task libary) werden auf Controller-Ebene miteinander kombiniert. Die Gesamtmission wird von einem Bodensegment gesteuert welches die Perzeption, Planung und Aufgabenüberwachung bündelt. Quelle: [42]

[4] ANother Tool for Language Recognition

Wie die lose gekoppelte Architektur gibt es ein Bodensegment (blaue Kästen in Abb. 3.8) für die Auswertung der Wahrnehmungen, einen Planer für das Erstellen eines Missionsplans und einem Task Manager, der die Aufgaben synchronisiert und Ausführungsschranken überwacht. Eine Besonderheit der Architektur ist, dass nicht wie sonst üblich Aufgaben-Primitive kommandiert werden, sondern dass direkt verschiedenen Low-Level-Controller in den UAVs aktiviert werden. Die Architektur wird vor allem dadurch sehr flexibel, dass komplexe Verhalten aus elementaren Verhalten kombiniert werden können (vgl. Abb. 3.9).

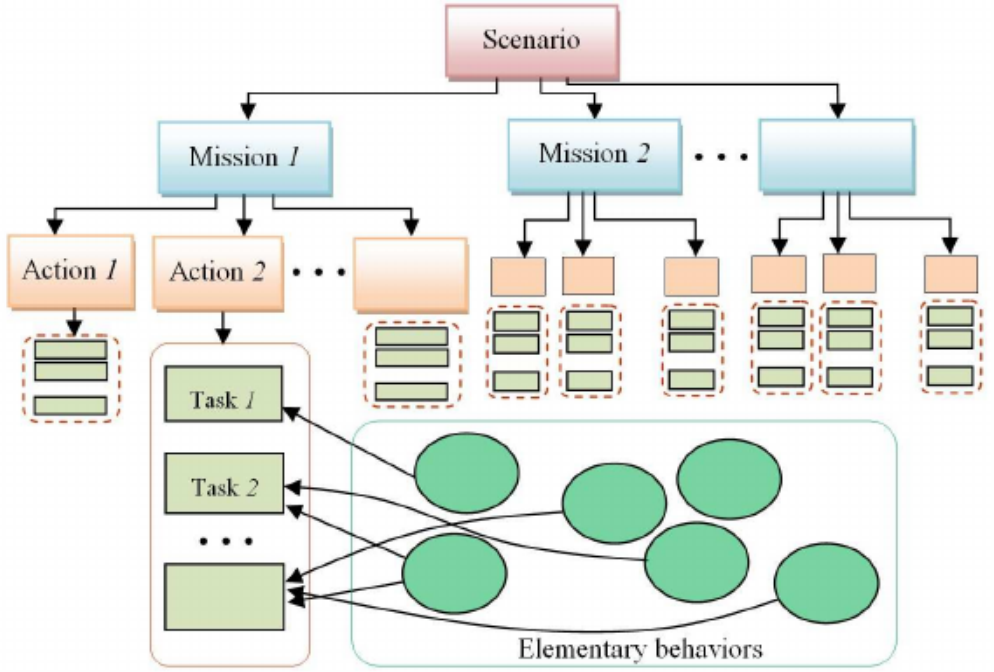


Abbildung 3.9.: Bottom-Up Aufgabenzerlegung in der CAVIS Architektur. Die *Elementary behaviors* werden zu *Tasks* kombiniert welche in *Actions* verwendet werden können. *Actions* werden dann zu *Missionen* verknüpft. Quelle: [42]

Elementary behaviors sind die kleinste definierte Kontrolleinheit, die einem UAVM zugewiesen werden kann. Sie sind durch eine Task-Variable definiert, welche als Funktion der UAVM-Konfiguration definiert ist. Beispiele für diese Variable sind die *Roboterposition* für die Bewegung der Plattform, *Endeffektor-Position*, um den Arm exakt zu positionieren (unter Berücksichtigung der Flugeigenschaften) oder *Objektkonfiguration*, welche eine Bewegungstrajektorie für ein von mehreren UAVMs gegriffenes Objekt vorgibt.
Compound behaviors oder ***Tasks*** ist die Kombination der elementaren Verhalten auf Basis des NSB-Ansatzes [44]. Der NSB-Ansatz nutzt aus, dass die DoFs, die für eine Aufgabe benötigt werden, geringer sind als die DoFs des Gesamtsystems. Die Ausgaben der Verhalten werden basierend auf den Prioritäten kombiniert, wobei die Ausgaben eines weniger prioren Verhaltens auf den Nullraum des höher prioren Verhalten projiziert wird. Dadurch wird die Ausgabe des hö-

herprioren Verhaltens nicht beeinflusst. So wird etwa während einer Ausweichbewegung nur dann eine Bewegung in Richtung des Ziels ausgeführt, wenn diese der Ausweichbewegung nicht entgegensteht. Im Vergleich zu klassischen verhaltensbasierten Ansätzen kann dadurch eine Kombination mehrerer Verhalten überlagert werden, ohne zu fordern, dass nur ein Verhalten gleichzeitig aktiv ist.
Actions gruppieren die *Tasks* auf einer höheren Ebene, wobei ein *Task* zu mehreren *Actions* gehören kann. Beispiele für *Actions* sind: *moveV* für Aufgaben die das UAVM als ganzes bewegen sollen, etwa, um einen Punkt zu erreichen oder *moveC*, bei dem integrierte Sensoren bewegt werden, entweder, indem auf die Bewegung des Fahrzeugs (*moveV*) oder des Endeffektors zurückgegriffen wird.
Missions sortieren *tasks* in einer zeitlichen Abfolge und aufgrund von Vorbedingungen wie dem Greifen eines bestimmten Objektes. Die Ausführung einer Mission, bzw. der darin enthaltenen *tasks*, wird vom *task manager* überwacht. Dieser prüft, ob die Bedingungen eines Tasks erfüllt sind und synchronisiert die Ausführung.

Einordnung und Bezug zur vorliegenden Arbeit

Das Projekt ARCAS befasst sich wie COMETS und AWARE mit UAV und baut auf den vorherigen Ergebnissen auf, beinhaltet jedoch einige Erweiterungen, die so auch in anderen Projekten und Frameworks zu sehen sind.

Die lose gekoppelte Kommunikation nutzt weiterhin die bereits aus den vorherigen Projekten bekannten Aufgabenprimitiven, neu ist allerdings der Versuch, diese Kommandierung in ein standardisiertes Zwischenformat zu bringen, um möglichst viele Systeme zu unterstützen. Im Sinne der Übertragbarkeit ist dies in jedem Fall wünschenswert, stellt letztendlich jedoch auch nur ein Übersetzen der Primitiven in eine andere Sprache dar. Das Problem, dass diese Sprache durch neue Team-Teilnehmer nicht mächtiger wird oder sich auf neue Aufgaben einstellen kann, bleibt bestehen. Um dieses Problem zu adressieren muss ein Ansatz auf Ebene der Fähigkeitsmodellierung erfolgen. Der Fokus einer solchen Koordination liegt allerdings auch auf einer langfristigen kohärenten Missionsplanung, nicht auf einer aktiven Task-Zuordnung für spontane Handlungen der Systeme und ist daher eher als Erweiterung einer unterliegenden Koordination relevant.

Ebenfalls neu ist die Modellierung von Missionssyntax mittels ANTLER und das Abfragen von Ontologien bei der Erstellung und Planung der Mission. Auch dieses Muster ist bei vielen weiteren Ansätzen zu erkennen. Durch Einschränkung der Missionsbeschreibung wird sichergestellt, dass erst gar keine Missionsanforderungen gestellt werden können, die nicht durch das Team erfüllbar sind. Während diese Einschränkung aus Usability-Sicht natürlich sinnvoll ist, da alle Missionen immer ausführbar sind, wird dadurch jedoch abermals die Mächtigkeit des Systems auf die fest definierten Grenzen der erlaubten Syntax beschränkt. Ein ähnlicher modellbasierter Ansatz wird in dieser Arbeit ebenfalls untersucht.

Eine Besonderheit ist der Ansatz zur eng gekoppelten Kommunikation. Durch die *elementary behaviors* wird ebenfalls eine kleinste Befehlseinheit definiert, welche allerdings maximal systemspezifisch ausgelegt ist. Anstatt die Verhalten zu vereinheitlichen, wird durch den NSB-Ansatz eine innovative Methode zur Kombination mehrerer Verhalten eingeführt, die für die Kombination der Regler sehr effektiv ist. Das System ist dadurch zu synchronisierten Handlungen und schnellen Reaktionen fähig. Die Regler sind jedoch extrem systemspezifisch und auch die Kombination ist nicht einfach auf beliebige Systeme übertragbar insbesondere, wenn diese sehr unterschiedlich sind. Während die Konzepte daher nicht direkt für den in dieser Arbeit verfolgten Ansatz verwendet werden können ist insbesondere der NSB-Ansatz z. B. für gekoppelte Bewegungen eine vielversprechende Ergänzung, wenn synchronisierterte Roboterfähigkeiten genutzt werden sollen (also bei ST-MR-TA Problemen).

3.3.3. MARS2020

Das DARPA Projekt MARS 2020 (2002-2004) [45] zielt darauf ab, eine kooperative Exploration mit einer großen Gruppe aus sieben sehr unterschiedlichen UGVs und zwei Tragflächen UAVs zu ermöglichen, welche lediglich einen einzelnen menschlichen Operator benötigt. Für die Steuerung der Roboter kommen drei verschiedene Frameworks zum Einsatz: Player[5] erlaubt vor allem die einfache Sensorabstraktion und bietet netzwerktransparente Möglichkeiten, Sensoren und Aktoren zu kombinieren. ROCI wurde vor allem für die Parallelisierung von Verhalten, Abstraktion und das Logging von Daten und die Ausführung der Verhalten auf einem Roboter genutzt. MissionLab ist eine Sammlung von Tools für die Entwicklung und Verkettung von Verhalten, welche dann als ausführbarer Code auf die Roboter übertragen werden können. Um die unterschiedlichen Systeme und Architekturen in ein System zu kombinieren, werden die Systeme nicht integriert, sondern stattdessen mit einem Adapter verbunden, mit welchem eine unabhängige Kommandosprache gelesen und als Mission für den jeweiligen Roboter übernommen wird. Die verfügbaren Befehle richten sich nach den vorher entwickelten Verhalten und werden auf diese abgebildet.

Während die explizite Kontrolle über diese Kommandos und Verhalten im Vergleich zu anderen Projekten sehr rudimentär ist, sind die impliziten Koordinationsmechanismen komplexer. Ein graphbasierter Routing-Algorithmus stellt die Konnektivität über das gesamte Team jederzeit sicher und überwacht die Signalstärke, um ggf. die Positionen der Teilnehmer anzupassen. Die Exploration der Umgebung erfolgt durch eine geteilte Unsicherheitskarte [46]. Durch die Modellierung der Sensorkegel von UAV und UGVs und der im Team geteilten Karte erfolgt eine implizite Regelung der einzelnen Team-Teilnehmer in die noch unbekannten Gebiete und damit eine effiziente Exploration, bei der die unterschiedlichen Sensoren der Systeme ausgenutzt werden. Die Observationen werden von

[5] Üblicherweise zusammen mit der Simulationsumgebung Stage als Player/Stage bezeichnet.

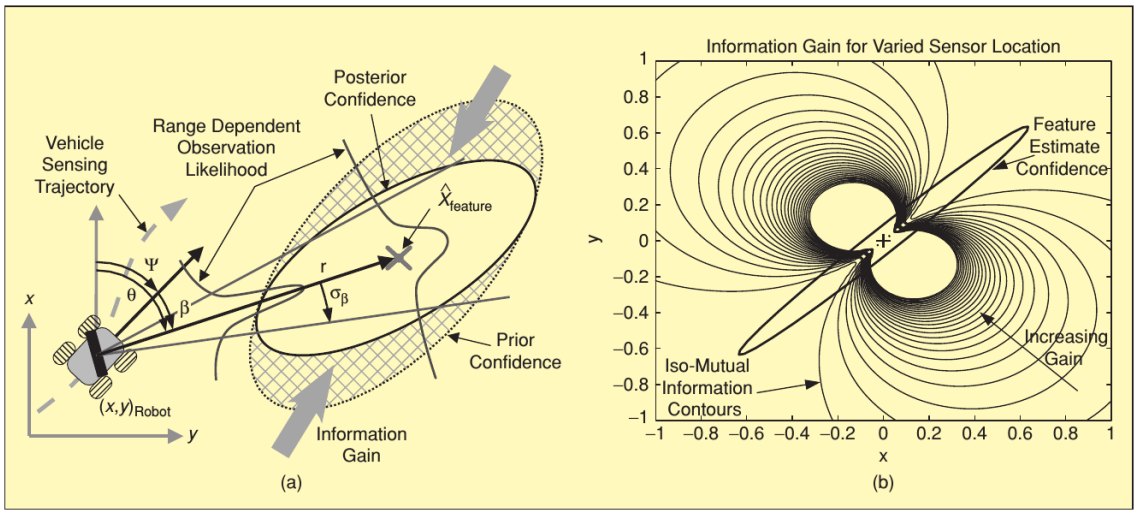


Abbildung 3.10.: Informationsbasierte Regler aus dem MARS Projekt. Die Systeme nutzen die geteilten Observationen um zu berechnen durch welche Kommandos der größte Informationsgewinn erzielt werden kann und optimieren ihr verhalten dadurch in Bezug auf die Gesamtmission.Quelle: [46]

jedem Roboter lokal gespeichert. Eine Auswertungskomponente wird zu jedem Roboter übertragen, welche nach einem gewissen Featuresatz sucht und dann nur das Ergebnis zurückmeldet.

Bezug zur vorliegenden Arbeit

Das MARS 2020 Projekt hat einen starken Fokus auf die Gesamtmission und auf eine impliziten Koordinierung der Exploration. Die verwendeten Frameworks sind inzwischen größtenteils überholt, gerade Player/Stage verfolgt jedoch sehr ähnliche Ansätze wie das heute populäre und in dieser Arbeit genutzte ROS. Die Auswertung der gemeinsamen Daten durch Verständigung auf ein gemeinsam relevantes Format (die Observation der Zielobjekte) ist eine geeignete Möglichkeit, die Kommunikation im Vergleich zur Synchronisation der Rohdaten signifikant zu reduzieren. Ein ähnlicher Ansatz wird auch in dieser Arbeit verfolgt.

Ein wichtiges Konzept des Projektes ist die implizite Koordination des Routings durch die Nutzung eines Ultitity-Wertes (der Unsicherheitskarte). Hier wird durch die unterschiedlichen Sensoren und Blickwinkel ein echter Mehrwert aus der Heterogenität der Teilnehmer gezogen. Zudem ist diese Art der Koordination gut skalierbar, was mit der großen Teamgröße im Projekt eindrucksvoll demonstriert wird. Durch die Verlagerung der Koordination in die Regelung der einzelnen Roboter wurde der Aufwand für die Koordination quasi eliminiert, jedoch auch die Flexibilität, andere Missionen durchzuführen, negiert. Die Art der Synchronisation entspricht damit dem der verhaltensbasierten Frameworks. Es ist zwar

beliebig skalierbar, neuen Bedingungen gegenüber jedoch inflexibel. Die systemunabhängige Kommando-Sprache ist, wie bei COMETS oder ARCAS, lediglich der kleinste gemeinsame Nenner der Systeme und nicht in der Lage, komplexere Missionen geeignet abzubilden. In dieser Arbeit wird daher ein expliziter Kommunikationsansatz verfolgt. Die implizite Koordination ist jedoch vor allem für das Konzept der PoI relevant.

3.3.4. RECONFIG

Das EU-Projekt RECONFIG (2013-2016) hat das Ziel, eine robuste Koordination zwischen heterogenen Robotern zu ermöglichen und so die komplementären Eigenschaften der Roboter zu nutzen. Es werden dabei drei wichtige Kernbereiche erforscht: Erstens wird Bildverarbeitung eingesetzt, um ein akkurates Verständnis der Szene um die Roboter aufzubauen. Beispielsweise wird die Zeigegeste eines humanoiden NAO Roboters genutzt, um anderen Team-Teilnehmern ein bestimmtes Objekt als Zielobjekt zu kommunizieren [47]. Diese Informationen wird mit symbolischen Informationen über die Umwelt kombiniert, um Aussagen über bestimmte Objekte treffen zu können und so Wissen von einem Roboter zu einem anderen übertragen zu können, beispielsweise um die Information „dieses Objekt ist eine Tasse“ zu vermitteln. Durch die Einbettung der Information in die Umgebung können symbolische Zusammenhänge dadurch auch ohne gemeinsame Wissensbasis vermittelt werden und die Kooperation zwischen Robotern verbessern. Zweitens werden gemeinsame Aufgaben zwischen mehreren Robotern durch implizite und explizite Kommunikation geplant und koordiniert. Diese beinhalten etwa die gemeinsame Exploration mit direkter Kommunikation, Griffplanung, bei der ein Roboter den Griff bestimmt, den ein zweiter ausführt oder aber das gemeinsame Tragen eines Objektes mittels Kraftsensorik, die eine implizite Kommunikation durch das Objekt selbst erlaubt [48]. Neben diesen spezifischen Problemen der Interaktion beschäftigt sich das Projekt mit dem für diese Arbeit relevantesten Bereich: Die Fragestellung, wie die heterogenen Roboter und deren Aufgaben geplant und untereinander koordiniert werden können.

Für die Bewegungs- und Aufgaben-Planung der einzelnen Teilnehmer, aber insbesondere auch einer Planung über das gesamte Team hinweg, wird eine Formalisierung mittels LTL (Linear Temporal Logic) verwendet [50]. Für jeder Teilnehmer werden zunächst individuelle lokale Aufgaben mittels LTL-Formeln spezifiziert. Dabei können Aufgaben sowohl als zwingend (hard) als auch als optional (soft) definiert werden. Basierend auf der initialen Informationen der Welt wird darauf basierend ein koordinierter Plan für jeden Teilnehmer synthetisiert, welcher alle zwingenden Anforderungen berücksichtigt und versucht, die optionalen so gut wie möglich zu erfüllen. Das Projekt liefert einen wichtigen Beitrag, indem zum einen eine Methode entwickelt wird, um die Planung auf einem variablen Horizont durchzuführen und zum anderen das Wissen der individuellen Roboter zu teilen und dadurch die Planung iterativ zu verbessern. Durch den beschränkten Planungshorizont wird keine initial global optimale Lösung erstellt,

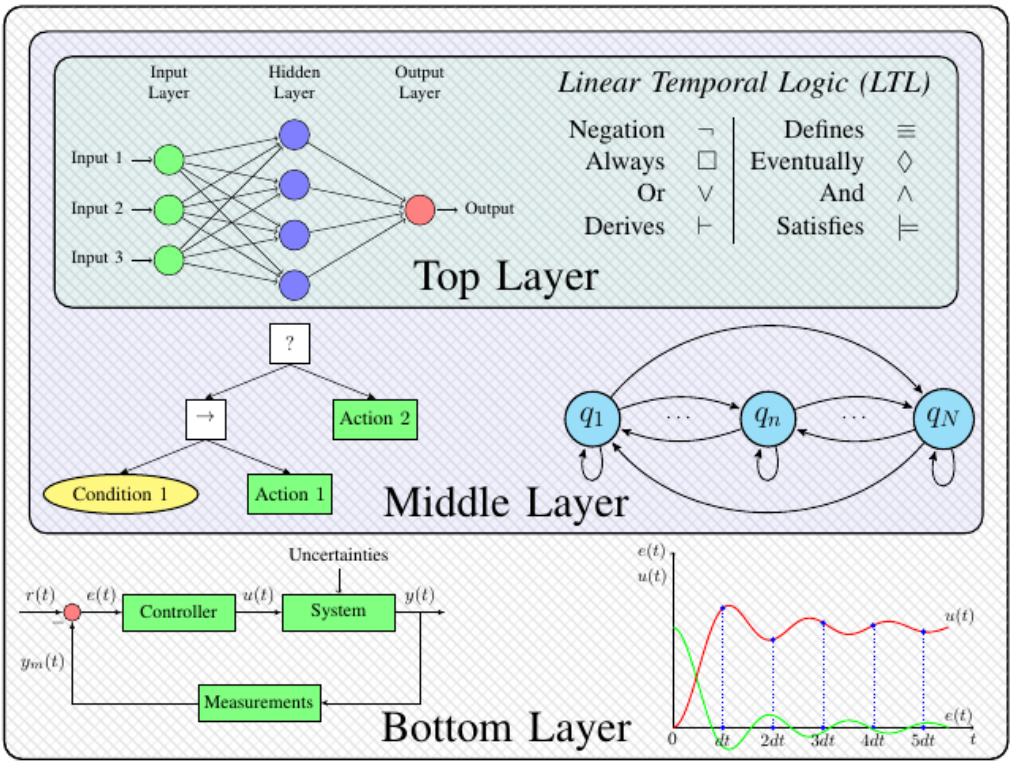


Abbildung 3.11.: Ebenen der Planung und Ausführung im RECONFIG Project. Der High Level Plan wird mittels LTL (Linear Temporal Logic) oder vergleichbaren Ansätzen erstellt. Controlled Hybrid Dynamical Systems (CHDSs) bzw. BTs (Behavior Trees) steuern die Ausführung des symbolischen Plans auf Tasks-Ebene. Auf unterster Ebene übernehmen spezifische Controller die kontinuierliche Ansteuerung. Quelle: [49]

gleichzeitig wird die Berechnung mit vielen Teilnehmern in einer unbekannten Umgebung überhaupt erst ermöglicht. Kontinuierlicher Austausch der von den Individuen gesammelten Informationen über die Welt führen durch die iterativ ausgeführten Planungsschritte dazu, dass sich der synthetisierte Plan immer weiter optimiert und dadurch auch die weichen Anforderungen erfüllt. Durch die Koordination der Controller der einzelnen Teilnehmer können so auch weitere Randbedingungen, wie etwa Formationen, berücksichtigt werden.

Während die LTL für die (verteilte) Planung auf Missions-Ebene (top layer) genutzt wird, findet die Steuerung von Robotern auf der Controller-Ebene (bottom layer) statt. Zwischen den beiden Ebenen (siehe Abbildung 3.11) ist ein koordinierendes System nötig, welches flexibel auf geänderte Bedingungen wie etwa einen Sensorfehler reagieren kann und optimalerweise modular erweiterbar ist. Bei RECONFIG wurde dafür auf BTs (siehe 3.4) gesetzt. Insbesondere ist durch eine Formalisierung der BTs [49] eine wichtige Grundlage für die Verwendung der Technik in der Robotik gelegt wurden. Auch wurde mit [51] bereits über die Vorzüge von BTs für Multi-Roboter-Anwendungen berichtet. Da BTs und insbe-

sondere die genannten Veröffentlichungen die grundlegende Basis für die Entwicklungen in dieser Arbeit sind, wird auf sie in 3.4 im Detail eingegangen. Die in diesem Projekt entwickelten Formalismen sind die Grundlage für nahezu alle BT-Ansätze.

Einordnung und Bezug zur vorliegenden Arbeit

Das RECONFIG Projekt behandelt sehr unterschiedliche Fragestellungen im Bereich der MRSs. Das Ausnutzen der Welt als Informationsspeicher ist insbesondere bei heterogenen Systemen mit einer unterschiedlichen Darstellung von Informationen ein mächtiges Konzept, welches auch bei keiner vorherigen Einigung auf Datenstrukturen, Semantik oder ähnliches einen Austausch ermöglicht. Damit einher geht jedoch auch die Notwendigkeit einer sehr aufwendigen Auswertung der Sensordaten, wie es etwa die vielfältigen Veröffentlichungen zum Thema der Erkennung einer Zeigegeste verdeutlichen. Insbesondere erfordert die Verständigung über die Kommunikation aber vorher ausgetauschte Konzepte über gewünschte Handlungen oder Ziele. Wie bei den verhaltensbasierten Frameworks erfordert die Reaktion auf einen Trigger in der Umgebung das Wissen darüber, welche Reaktion dies sein soll. Das Konzept wird auch in dieser Arbeit etwa durch die PoI eingesetzt.

Die Koordination über Kräfte ist vor allem bei kooperativen Aufgaben die von mehreren Robotern relevant und spielt daher für diese Arbeit keine wichtige Rolle. Zu bemerken ist jedoch, dass die kooperativen Regelungsaufgaben sehr viel weniger allgemein definiert werden und stets ein gemeinsames grundlegendes Verständnis der Aufgabe benötigen. Es scheint daher zielführend, zunächst nur lose gekoppelte Koordination zu behandeln und das System für engere Kopplung weiter zu entwickeln.

Die Koordination der einzelnen Teilnehmer ist eine wichtige Grundlage für diese Ausarbeitung. Während das formale Planen mit LTL zwar einige Vorteile wie etwa Aussagen über die Ausführbarkeit von Missionen bietet, ist der Modellierungs- und Rechenaufwand hoch. Insbesondere bei wenig planbaren Missionen bzw. unbekannten Weltzuständen ist eine zielgerichtete Planung dadurch aufwendig. Die vorgeschlagenen Behavior Trees bieten hingegen eine sehr gute Balance aus Flexibilität und einfacher Anwendung, insbesondere im Kontext der Koordination von fähigkeitsbasierten Aufgaben. Ihre erweiterbare Struktur, einfache Art der Definition und Modularität machen sie zu einer optimalen Kontrollstruktur für komplexe Robotersysteme bei denen während der Entwicklung noch nicht alle Fähigkeiten fest stehen. Das Konzept der BTs wird daher in dieser Arbeit aufgegriffen und um Konzepte der Multi-Roboter-Kooperateration erweitert.

3.3.5. SHERPA

Das EU Projekt SHERPA (2013-2017) [52] realisiert die Kooperation Menschen („busy Genuius“) und verschiedenen Arten von Robotern („SHERPA anymals“), um in alpinen Gegenden zielgerichtet nach Verschütteten zu suchen. Das heterogene Team aus Mensch, Bodenroboter und verschiedenen Flugrobotern mit unterschiedlichen Ausstattungen (Tragflächen-UAV und Multikopter) und Fähigkeiten sammelt Daten mit unterschiedlichen Sensoren und unterstützt den Menschen bei der Rettung. Um den Menschen bei seiner eigentlichen Arbeit möglichst wenig zu belasten, wurde auch eine Kommandierung über multimodale Schnittstellen, etwa Gesten, erforscht und ein großer Fokus auf die Robustheit und Autonomie der Systeme gelegt.

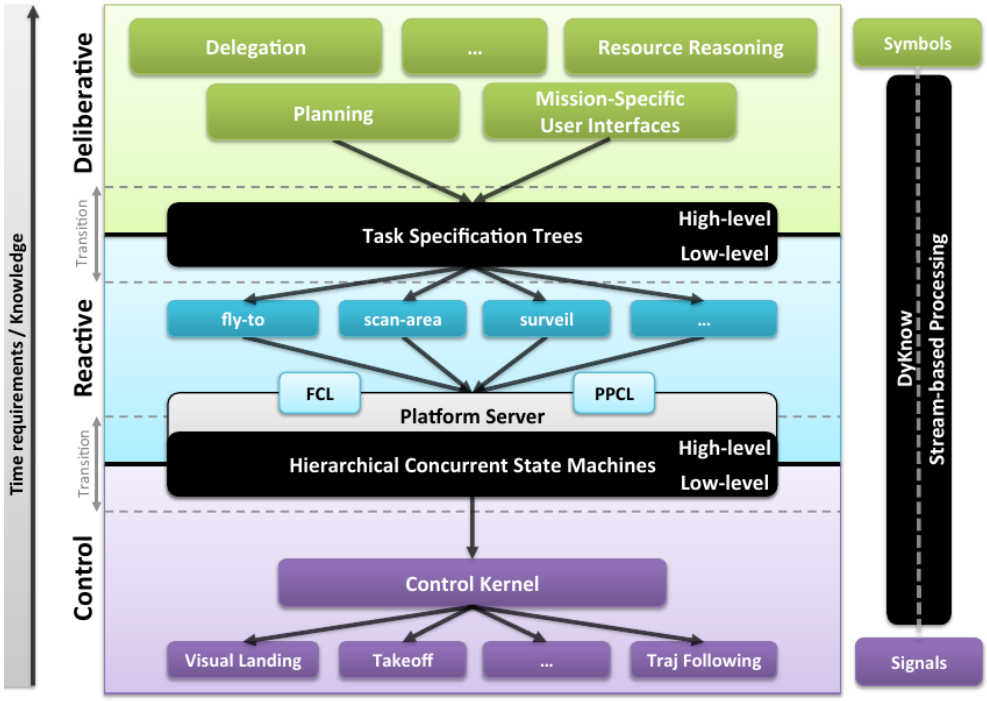


Abbildung 3.12.: Die HDRC3 Architektur im SHERPA Projekt. Die drei Ebenen realisieren jeweils unterschiedlich reaktive (ms bis Minuten) Steuerungen. Die *Control*-Ebene arbeitet direkt mit Steuerbefehlen, die *Reactive*-Ebene steuert Aufgabenabläufe wo reaktionen im Sekunden-Bereich liegen dürfen. Die *Deliberative*-Ebene beinhaltete symbolische Planer die auch mehrere Minuten rechnen können. Quelle: [53].

Die Koordination des Roboterteams erfolgt mit der in dem Projekt entwickelten HDRC3 Architektur [53] (Abbildung 3.12). Die Architektur ist in diesem Fall für UAV entwickelt worden, kann aber auch für beliebige andere Systeme verwendet werden. Die Architektur folgt der von Gat [54] beschriebenen 3-Schichten-Architektur, welche nicht nach dem SPA-Paradigma operiert, sondern stattdes-

sen die Arten der Algorithmen unterscheidet:

- Reaktive Steueralgorithmen
  Für die sensorabhängige Regelung von Vorgängen die eine Reaktionszeit im ms Bereich erfordern.

- Algorithmen für Aktionssequenzen ohne Suche
  Aktionen im Bereich einiger Sekunden die zwar stark auf interne Zustände zurückgreifen, aber keine aufwendigen Suchen erfordern.

- Zeitintensive, suchbasierte Algorithmen
  Berechnungen im Minutenbereich, welche oft einen großen Zustandsraum nach einer optimalen Lösung durchsuchen müssen.

Entsprechend dieser Einteilung definierten sie die Bereiche

- Control Layer
  Diese Ebene beinhaltet eine Bibliothek von Reglern, welche jeweils spezifische Aufgaben erfüllen, etwa das bildbasierte Landen, die Folge einer Trajektorie oder einfach das Halten der aktuellen Position. Der *Control Kernel* ist über mehrere Rechnersysteme verteilt und koordiniert die Ausführung der Regler und insbesondere das Umschalten zwischen verschiedenen Reglern durch die *Hierarchical Concurrent State Machines*, also hierachische Zustandsautomaten mit Nebenläufigkeit. Diese Ebene operiert in Echtzeit über die Real-Time-Kernel-Erweiterung (RTAI) von Linux.

- Reactive Layer
  Auf dieser Ebene werden High-Level-Pläne aus der *Deliberative Layer* mittels Task Specification Trees [55] in Handlungsanweisungen überführt. TSTs modellieren Aktionen wie *fly-to* mit ihren Parametern und Abfolgen.

- Deliberate Layer
  Hier finden alle Auswertungen auf symbolischer Ebene statt. Es werden High-Level-Pläne erstellt und Ressourcenkonflikte gelöst.

Neben den Ebenen selbst sind die Übergänge zwischen diesen in der Architektur relevant. Zwischen *Deliberative und Reactive Layer* kommen die TSTs zum Einsatz. Die Ergebnisse vom Planer können direkt in eine TST-Darstellung überführt werden, da diese neben der reinen Beschreibung auch die zeitliche Abfolge kodiert. Tasks auf der *Reactive Layer* können dann die FCL oder Payload and Perception Control Language (PPCL) nutzen, um hardwareunabhängig mit dem *Platform Server Interface* zu kommunizieren. Ein FCL-Kommando wird auf der *Control Layer* dann einem hierarchischen Zustandsautomaten zugeordnet, der die Ausführung kontrolliert. Die Umsetzung der Architektur erfolgt im ROS Framework, so das sdie Komponenten auf verschiedenen Systemen laufen können.

Einordnung und Bezug zur vorliegenden Arbeit

In SHERPA wird wie in vielen anderen Projekten eine dreistufige Architektur verwendet. Planungs- und Ressourcenkonflikte zentral abgehandelt, während für die Umsetzung individueller Fähigkeiten auf dem Roboter die Regelungs-Ebene verwendet wird. Interessant ist die Nutzung der TST, welch der Struktur von BTs sehr ähneln. Die einfache Zerlegung der Aufgaben durch eine Baumstruktur aus Tasks ist für heterogene Systeme eine gute Wahl. Die Modellierung erfolgt jedoch nicht durchgängig. Während die TSTs wie die BTs gut für die Koordination der Tätigkeiten geeignet sind wird die eigentliche Kommandierung auf Systemebene durch hierarchische Zustandsautomaten umgesetzt. Die Umsetzung zwischen den beiden Darstellungsformen, inklusive nutzen der FCL oder PPCL als Zwischenformat, erfordert wieder eine Festlegung auf ein festes Vokabular und Fähigkeiten und verliert damit den Vorteil aus der Modellierung. Diese Arbeit setzt mit BTs auf ein ähnliches Konzept, reizt durch die tiefere Integration aber vor allem die Modellierungsoptionen von BTs besser aus.

3.3.6. Weitere Projekte

Das französische Projekt ACTION (20072015) untersucht die gemeinsame Koordination von Flug, Boden und Wasserrobotern vor allem im Bereich kooperativer Einsatzplanung und Fusion der heterogenen Observationen. Das EU-Projekt TRADR (2013-2017) beschäftigt sich mit dem Einsatz von Roboterteams für Ersthelfer wie die Feuerwehr und konzentriert sich stark auf Nutzerinteraktion, aber auch die Frage, wie unterschiedliche Deskriptoren aufeinander abgebildet werden können sowie der generellen 3D-Lokalisierung. Co4Robots (2017-2020) untersucht komplexe dezentrale Kollaboration von mobilen Plattformen, wofür zentrale LTL-Pläne erstellt und dann mittels lokalen Controllern fusioniert werden. Das EU-Projekt PRO-ACT (2019-2021)[56] koordiniert Manipulationsfähigkeiten heterogener Roboter für das Erstellen von Strukturen auf dem Mond und nutzt lineare Optimierung zur Berechnung der optimale Aufgabenverteilung, Auktionsmechanismen zur Zuordnung vergleichsweise disjunkter Tasks und zeitliche Synchronisierung von Trajektorien für kooperative Manipulation. Das EU-Projekt FlexiGroBots (2021-2023) zielt auf eine gemeinsame Plattform für heterogene Roboter im Agrarbereich ab wobei vor allem eine vernetzte Auswertung und koordinierte Beauftragung, weniger die Koordination im Fokus steht. Das EU-Projekt CANOPIES (2021-2024) behandelt Multi-Roboter-Teams unter Berücksichtigung des Menschen für *precission agriculture* für den Weinbau. Für die Koordination wird ein optimaler Plan mittels linearer Programmierung unter Berücksichtigung von Auslastung, Qualität oder menschlicher Supervision erstellt [57] und zeitliche und räumliche Randbedingungen aufgelöst. Eine Besonderheit ist, dass die Kosten für den Taskwechsel explizit berücksichtigt werden. Jedes Projekt zeigt dabei interessante Aspekte für MRS, bringt jedoch keinen signifikanten Mehrwert zu den intensiver diskutierten.

3.4. Behavior Trees in der Robotik

BTs (Behavior Trees) sind eine Weiterentwicklung von endlichen Automaten für die Ablaufsteuerung. Ihre erste Erwähnung fanden BTs in der Entwicklung von NPCs in Computerspielen, insbesondere HALO [58, 59]. Seit dem wurden viele der verwendeten Begriffe und Methoden vereinheitlicht und formalisiert, so dass BTs heute eine populäre Technik für die Spielentwicklung sind wobei auch stetig Verbesserungen wie zum Beispiel generierte Verhalten [60] oder das koordinierte Verhalten von mehreren Agenten [61] berücksichtigt werden. Die oft genutzte Unreal- [62] oder Unity-Engine [63] enthalten etwa einen grafischen Editor für BTs als „Standard-Methode“ der Modellierung eines NPCs.

BTs erlauben insbesondere modular und reaktiv komplexe Verhalten zu modellieren und stehen den FSMs dabei in nichts nach, weshalb sie sich schnell als effiziente und dadurch auch kostengünstige Methode in der Videospielindustrie durchgesetzt haben und sich seitdem auch in der Robotik immer mehr zu einem Standard entwickelt haben. Michele Colledanchise und Peter Ögren [64] geben mit ihrer ständig aktualisierten Übersicht „Behavior Trees in Robotics and AI - An Introduction“ nicht nur eine sehr weitreichende Einführung über das Gebiet und relevanter Themen der BTs, sondern stellen auch den entscheidenden Vorteil gegenüber FSM dar: Während eine FSM ähnlich wie ein GoTo-Statement in Programmiersprachen ein „one-way control transfer“ darstellt, bieten BTs wie die Funktionsaufrufe in Programmiersprachen ein „two-way control transfer“ [64, S.5]. Diese Eigenschaft zeichnet sich vor allem dann aus wenn ein Teil der FSM geändert oder entfernt werden soll. In diesem Fall müssen alle existierenden Übergänge überprüft werden, während BTs dies über ihre internen Knoten automatisch abhandeln können. Ivono et al. zeigen in [65] darüber hinaus, dass auch der Entwicklungsaufwand durch das Nutzen von BTs im vergleich zu FSMs reduziert wird.

3.4.1. Erweiterungen der Kozepte

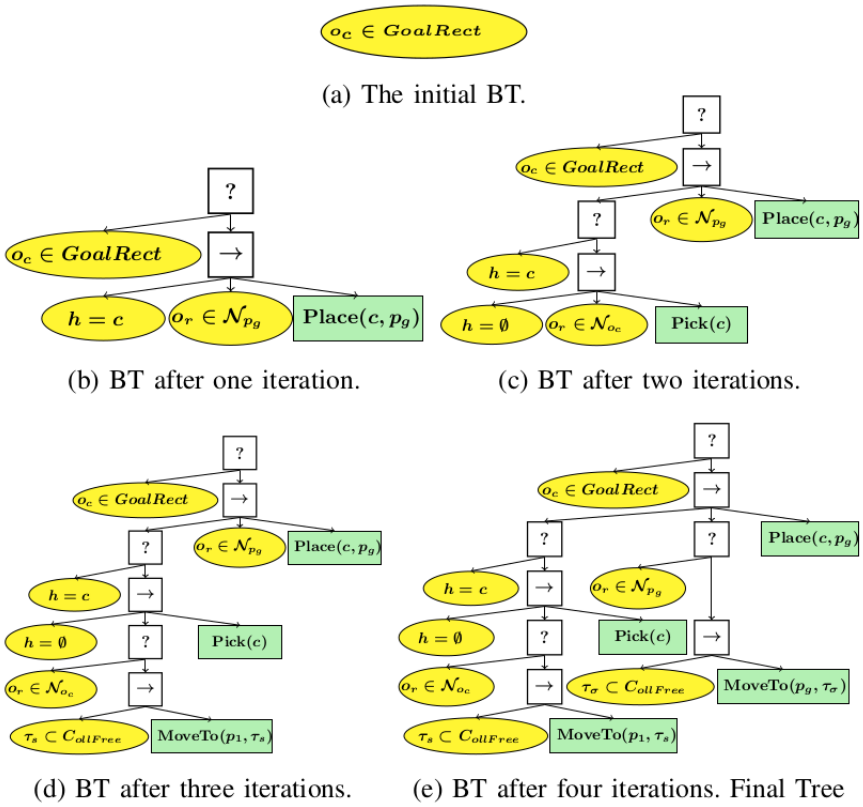
Ögren war einer der ersten, der BTs für die Robotik eingesetzt hat, indem er die Controller von UAVs durch BTs als ein Hybrid Dynamic System modellierte, bei dem Controller durch die BTs gewechselt werden [66] . Auch Klöckner hat BTs für die Koordination von UAVs verwendet [67] und dabei den neuen Zustand *activating* für einen Knoten eingeführt und definiert, was bei der Aktivierung und Deaktivierung zu geschehen hat. Durch diese Erweiterung wurden lang andauernde Aktionen berücksichtigt, welche explizit beendet werden mussten, bevor der Knoten deaktiviert werden konnte. Dadurch wird die Reaktivität der BTs zwar beschränkt, gleichzeitig ist dies für viele Fälle in der Robotik erforderlich, da etwa eine Armbewegung nicht unvermittelt abgebrochen werden kann. Andere Erweiterungen des BT-Konzeptes sind die Erweiterung um Parameter [68], womit eine bessere Wiederverwendbarkeit der BTs erreicht wird, oder neue Kontrollfluss-Knoten für explizite Parallelität von Colledanchise [69]. Das

in dieser Arbeit entwickelte BT-Modell berücksichtigt Erweiterungen wie lang dauernde Asynchrone Tasks und Parameter, geht dabei jedoch noch weiter, indem etwa Ultity-Berechnungen und insbesondere *Shovables* und *Capabilities* für die Verteilte und dynamische Ausführung von BTs eingeführt werden.

3.4.2. Formalisierung, Planung und Synthese

Marzinotto und Colledanchise haben im Rahmen des RECONFIG Projektes (3.3.4) die bestehenden Ansätze für BTs in einen formellen Rahmen gebracht [49] auf dem auch die Definitionen in dieser Arbeit basieren. Die formelle Definition in [49] wurde mit [70] von Colledanchise um die Synthese eines Plans mittels LTL erweitert. Durch die LTL Beschreibung wird die Planung der Aufgaben an die BT basierte Ausführung angenähert. Martens, Butler und Osborn gehen mit [71] noch einen Schritt weiter und definierten die BTL, wodurch ein Synthese-Schritt eingespart und BT direkt als Teil der temporalen Logik verwendet werden. Ziel ist es, die automatische Synthese und Verifikation von BTs zu ermöglichen. Die Verwendung von temporalen Logiken entkoppelt das Planungsproblem jedoch von der Ausführung der BTs. Mit [72] schlägt Colledanchise eine Möglichkeit vor, durch *Back-Chaining* die Planung direkt in ansonsten reaktive BT zu integrieren. Dabei werden Aktionen mit einer Menge von Vor- und Nachbedingungen als *atomare* BT definiert und eine Mission zunächst als eine Menge an Zielbedingungen definiert. Die Zielbedingung wird evaluiert und bei Nichterfüllung durch einen atomaren BT nach einer festen Vorschrift aus Fallback und Sequenz erweitert, der in seinen Nachbedingungen die ursprüngliche Bedingung erfüllt. Durch das Ersetzen werden neue Vorbedingungen eingeführt die nach dem gleichen Schema aufgelöst werden, bis der Baum ohne Fehler ausgeführt werden kann (Abbildung 3.13). Ögren hat diese Planungskonzept kürzlich noch einmal um ein explizites labeling auf Basis der enthaltenen Bedingungen erweitert [73]. Durch das explizite Labeln eines subtrees als „make sure to condition“ werden die erzeugten Lösungen für den Menschen les- und nachvollziehbarer.

Dieser Ansatz verfolgt ein ähnliches Ziel wie die in dieser Arbeit entwickelten *Capabilities*, die ihre Vorbedingungen ebenfalls explizit modellieren und vor allem durch *Meta-Capabilities* eine Substitution ermöglichen, fokussiert aber vor allem den Planungsaspekt. Durch den STRIPS-Artigen Planer können Handlungssequenzen aufgebaut und nachträglich auch optimiert werden, etwa um Widersprüche aufzulösen oder Prioritäten zu optimieren, es wird jedoch immer ein konkreter Behavior Tree aufgebaut und modifiziert, was insbesondere auch erfordert, dass es sich stets um reaktive Bäume handeln muss, die zunächst auch nur für einen Roboter optimiert sind. Die hier vorgestellten *Capabilites* legen den Fokus stärker auf eine dynamische Auswahl von Implementierungen und Verteilung im Team und nutzen daher eher einen Template-Ansatz als die Synthese durch Vorbedingungen.



(a) The initial BT.

(b) BT after one iteration.

(c) BT after two iterations.

(d) BT after three iterations.

(e) BT after four iterations. Final Tree

Abbildung 3.13.: Beispiel für den iterativen Aufbau eines Behavior Trees durch Backchaining. Durch das Auswerten des *Condition*-Knotens wird evaluiert, ob alle Ziele erfüllt sind. Wenn nicht, werden *atomare BT* nach einer Vorschrift eingefügt, mit denen die Bedingung erfüllt werden kann. Durch iteratives Erweitern kann so eine komplette Mission aufgebaut werden. Quelle: [72]

Einen anderen Ansatz der Planung verfolgt Rovida et al. welche in [75] ein Modell namens *extended Behavior Trees* (*eBT*s) einführen und damit die BT mehr an die HTN annähern. Durch die Modellierung von Verhalten als abstrakte Skills, konkrete Prozesse oder Primitive inklusive Vor- und Nachbedingungen und Parametern kann eine Planung mittels PDDL-Planer durchgeführt würden. Dabei wird durch die Planung vor allem eine Sequenz aus Skills als semantisch relevante Aktion definiert, die dann wiederum durch Prozesse konkretisiert und durch die Primitiven komplett aufgelöst werden können. Der Ansatz nutzt stärker als die vorherigen eine enge Formalisierung auf HTN Basis und Symbolisierung der Welt. Durch Modellierung eines konkreten Weltmodells und der aktuellen Szenen können sich die Vor- und Nachbedingungen auf konkrete Weltzustände beziehen und diese modifizieren. Aus der Vorgabe eines gewünschten Zielzustandes kann dadurch eine Skill-Sequenz zur Erfüllung einer Aufgabe erstellt werden. Durch das Auflösen der abstrakteren Skills in die konkreten Prozesse und Primitive wird dann ein konkreter Behavior Tree erstellt und Konflikte aufgelöst. Die Arbeiten an eBTs wurden mit dem 2023 veröffentlichten Framework SkiROS2

[74] insbesondere auch auf mehrere Roboter erweitert, die sich über ein gemeinsames Weltmodell synchronisieren. Die *eBT* Konzepte sind ebenfalls sehr ähnlich zu den in dieser Arbeit entwickelten Konzepten. Die hier entwickelten *Fähigkeiten* sind konzeptionell mit den *Procedures* aus SkiROS2 vergleichbar, die *Skills* und *Primitiven* mit unseren Konkreten *Implementierungen*. Durch den starken Fokus auf die Planungsebene spielen die BTs in diesem Ansatz jedoch vor allem auf der Optimierungsebene eine Rolle, bei der dann, wie bei dem von Colledanchise gezeigten Backtracking, die Sortierung geändert werden kann. Wichtig ist auch eine starke Modellierung durch Ontologien und Abbildung auf einen Weltzustand, wodurch der Ansatz eine größere Modellierungskomponente aufweist die in dieser Arbeit ebenfalls untersucht, jedoch nicht favorisiert wird.

Um die verschiedenen Tasks, die durch BT modelliert werden vergleichen zu können schlagen Cao und Lee eine Methode zur Abbildung von BTs auf einen Vektorraum vor [76], bei dem Semantik und Struktur erhalten bleiben. Dadurch werden verschiedenen Tasks nicht nur vergleichbar, sondern sogar ein Transfer bzw. eine Synthese ermöglicht. Während bisher nur einfach Bäume mit Sequenzen unterstützt werden, ist die Darstellung vor allem relevant, weil dadurch die bisherige symbolische Task-Planung hin zu maschinellem Lernen, insbesondere mit large language models (LLM) entwickelt werden könnten.

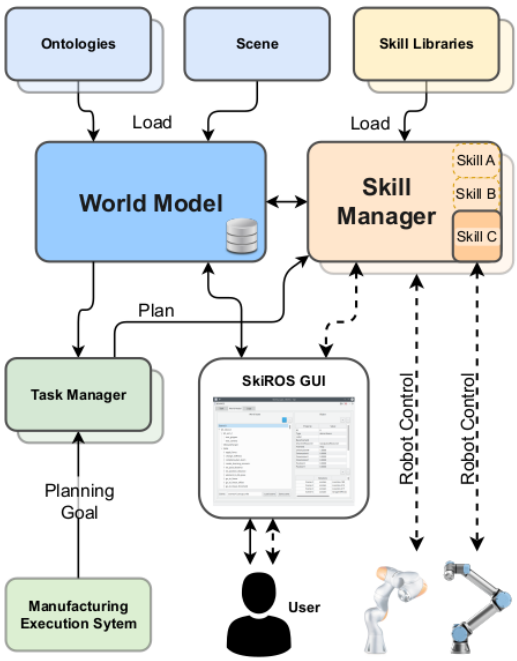


Abbildung 3.14.: Architektur von SkiROS2. Eine Synchronisation und Planung erfolgt vor allem auf Basis der modellierten Welt bzw. Szene. Durch abstrakte *Prozesse* kann auf einer semantischen Ebene geplant werden, durch konkrete *Skills* können diese dann nach und nach aufgelöst werden. Durch Synchronisierung der Szene können sich auch mehrere Systeme koordinieren. Quelle: [74]

3.4.3. Anwendungen

Behavior Trees finden in der Robotik bereits in einer Vielzahl von Applikationen Anwendung. Im CoStar Projekt [77, 78] werden BTs etwa für die nutzerfreundliche Programmierung von Industrierobotern für die Montage über ein grafisches Interface eingesetzt und das Unternehmen Rethink Robotics bietet einen BT-Editor als Standard-Werkzeug [79] für die Programmierung des Roboterverhaltens des Industrieroboters Sawyer. Auch das erste von der Firma Intrinsic [6] angekündigte Produkt *Flowstate* nutzt BTs [80]. In [81] wurden BTs für die Steuerung eines medizinischen Roboters für Operationen genutzt und mit micROS.BT [82] wurden BTs speziell für Sachwarmroboter adaptiert und in [83] für das Lernen von Schwarmverhalten eingesetzt. Iviono et al [84] und Ghzouli et al. [85] fassen in ihren Arbeiten die breiten Anwendungsmöglichkeiten und unterschiedlichen Ausprägungen der verschiedenen Ansätze zusammen. Es wird deutlich, dass BTs schon längst zu einem Standard für die Definition komplexer Verhalten geworden sind. In [86] fasst Colledanchise vor allem aktuelle BT Implementierungen zusammen und gibt Hinweise für die Entwicklung von BT-Bibliotheken, von denen viele Aspekte bei der Entwicklung dieser Arbeit ebenfalls unabhängig berücksichtigt wurden. Insbesondere die dabei vorgeschlagene Architektur und das Nutzen der BTs als *Skill* bestärkt die in in dieser Arbeit getroffenen Entscheidungen.

3.4.4. Behavior Trees für Multi-Roboter-Systeme

Colledanchise et al. haben Behavior Trees in der Robotik, insbesondere auch für Multi Roboter, als ein vielseitiges Werkzeug vorgeschlagen [51] und mit einer Veröffentlichung als ROS-Paket [87] auch eine Referenzimplementierung geschaffen. Für die Verteilung auf ein Team schlagen sie vor, einen BT in einen globalen Missions-BT und einen lokalen Ausführungs-BT und einen Zuweisungs-BT umzuwandeln. Der globale BT beinhaltet die Mission, optimiert jedoch deren Ausführung durch Umwandeln in eine möglichst parallele Struktur. Der lokale Ausführungs-BT überwacht die Ausführung einer Aufgabe auf dem jeweiligen System und der Zuweisungs-BT löst das optimal assignment Problem um die Zuordnung der Aufgaben zu den jeweiligen Systemen zu ermitteln. Alle drei Bäume werden parallel ausgeführt und sind nur erfolgreich, wenn alle Teilbäume erfolgreich sind. Jeong et al. [88] schlagen eine auf BTs und ROS2 basierende Koordination der Navigation eines Teams vor. Sie sehen ebenfalls einen zentralen Zuweisungs-BT vor, der per ROS-Message die Navigations-Ziele an das jeweilige System schickt, welches dieses mit einem lokalen Ausführungs-BT bearbeitet und Sonderfälle behandeln kann. Die Ansätze ähneln damit den in dieser Arbeit vorgeschlagenen parallelisierten Bäumen für eine maximal flexible Zuweisung im Team, bei dem jeweils eine Fähigkeit in einem parallelen Blatt ausgeführt wird,

[6] Intrinsic ist ein Moonshot-Projekt von Alphabet, ehemals Google, an das hohe Erwartungen von der Robotik Community gestellt werden.

bieten jedoch wenig positive Eigenschaften wie eine dynamische Zuweisung der Aufgaben, roboterspezifsche Implementierungen oder Dezentralität.

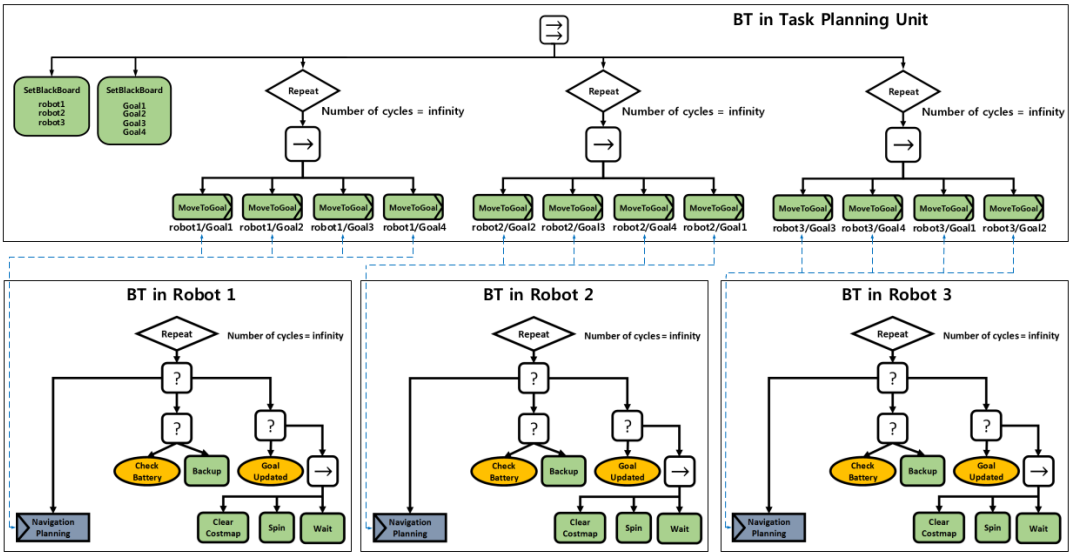


Abbildung 3.15.: Architektur für die Verteilte Navigation nach Jeong et al. . Ein zentraler Missionsbaum wertet die Ziele aus und verteilt diese in parallel ausgeführte Teile des Baums. Über ROS-Messages werden die Roboterspezifischen Ziele an die einzelnen Systeme gegeben die neben der Navigation auch einen Notfall-Fallback, etwa für die Batterieüberwachung ausführen. Quelle: [88].

Einen interessanten Ansatz verfolgen Venkata et al. mit ihrem Framework *KT-BT* [89]. Sie argumentieren, dass Wissen vor allem beeinflusst, welche Handlungen ausgeführt werden können und tauschen daher explizite Verhalten in Form von Behavior Trees zwischen den Systemen. Durch die entsprechende Anordnung der Verhalten (siehe Abbildung 3.16) wird, wie bei anderen Ansätzen auch, zunächst das Überleben des Systems durch BT, die die kritischen Systeme absichern, sichergestellt. Danach wird dann allgemeines Wissen, Vorwissen und dann gelerntes Wissen, bzw. die jeweiligen BT, ausgeführt. Parallel zu den Verhalten gibt es Lehrer- und Schüler-BT, welche den Wissenstransfer anfordern oder beantworten können. Zur Übertragung wird das entwickelte Format *stringBT* verwendet, in dem BT zum einen durch xml ähnelnden Steuersymbole und zum anderen durch String-Repräsentationen für konkrete Aktionen dargestellt werden, die wiederum durch Subtrees aufgelöst werden können. Gerade die Repräsentation der Aktionen und die Möglichkeit, Wissen durch deren Austausch zu lernen, ähnelt dem in dieser Arbeit beschriebenen Konzept der *Meta-Capabilities*. Das System ist jedoch für Schwärme ausgelegt und zielt daher auf eine Homogenisierung der Fähigkeiten im Team ab und behandelt die einzelnen System wiederum als gleich, was den Ansatz in der Anwendbarkeit beschränkt.

Taewos et al. [90] kombinieren eine marktbasierte Verteilung von Aufgaben im Team mit einer Synthese eines lokalen BT nach Art des Backchaining, wie von

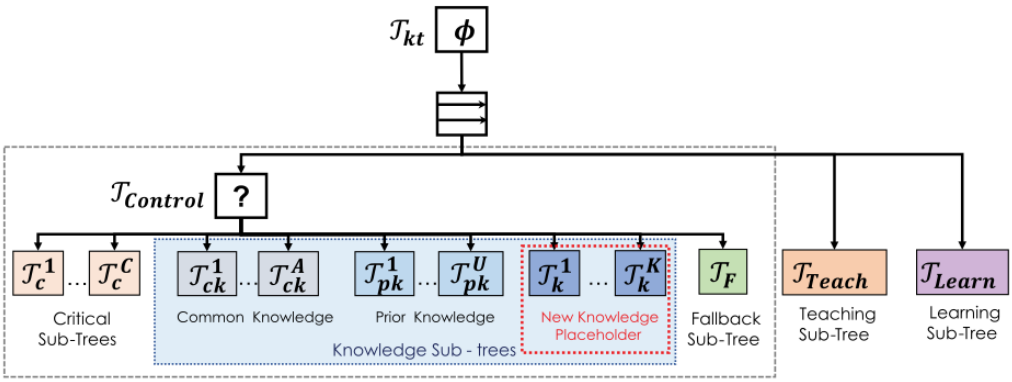


Abbildung 3.16.: Architektur des KT-BT Frameworks. Die hochprioren BT stellen die Sicherheit der Systeme sicher, danach folgen in absteigender Priorität verschiedene Wissensgrundlagen, die jeweils als BT umgesetzt werden. Durch den Fallback wird das Wissen so nach und nach traversiert. Parallel werden ein Lehrer- und ein Schüler BT ausgeführt, über die der Austausch des Wissens gestartet wird. Quelle: [89].

Colledanchise vorgeschlagen [72]. Teilnehmer des Teams könne auf Basis eines lokalen *Action Stores* und berechneten Kosten, basierend auf Metriken wie Energie oder Distanz, Gebote für Tasks berechnen und bei Zuschlag einen entsprechenden BT für den Task synthetisieren. Durch einen mehrstufigen Auktionsansatz können Teilnehmer Teile der Aktionen wieder auf andere Roboter auslagern, solange dabei keine zyklischen Vergaben konstruiert werden. Das mathematisch vorgestellte Konzept wurde bisher noch nicht für reale Systeme evaluiert, ist aber ebenfalls sehr ähnlich zu dem in dieser Arbeit entwickelten Konzept. Durch die Marktvergabe können Aufgaben dynamisch verteilt werden, die eingesetzten *Action Stores* werden jedoch nicht zur Laufzeit erweitert, und auch die Kosten sowie insbesondere die verfügbaren Aktionen und deren Granularität müssen bereits zu Beginn der Vergabe fest stehen.

Einen besondere Aspekt der verteilten Ausführung betrachten Sidorenko et. al. in [91]. Sie verwenden ebenfalls BTs als *Skills* für verschiedene Tätigkeiten in der Produktion. Für die Synchonsierung eines verteilten BT schlagen sie ein *BT Snchonisierungs Protokoll* vor, welches insbesondere die Knotenzustände wie *Running, Succes, Failure* kontrollieren und übertragen kann. Das Konzept wird ebenfalls für die Parametrisierung vorgeschlagen, um ein *dynamic wiring* zu erreichen. Das vorgeschlagene Konzept ermöglicht eine verteilte Ausführung von BTs, braucht dafür allerdings zusätzliche Komponenten, die nicht nativ in der BT-Definition enthalten sind. In dieser Arbeit werden solche Konzepte durch die *Wirings* umgesetzt, welche die nativen Parameter und Interfaces nutzen.

Ein BTs-System für die Multi-Roboter-Koordination, welches sowohl ein dynamisches Team, als auch dynamische Fähigkeiten und Aufgaben berücksichtigen kann, ist derzeit noch nicht bekannt und daher Fokus dieser Arbeit.

4. Fähigkeitsbasierte Kooperation

Wie Kapitel 3 aufgezeigt hat, wird Kooperation von Roboterteams auf sehr unterschiedliche Art und Weisen realisiert. Basierend auf den in Kapitel 1 aufgestellten Einschränkungen bezüglich des Teams und der damit verbundenen Themen wurde ein System für die *fähigkeitsbasierte Kooperation* von heterogenen Roboterteams entwickelt, welches in der Lage ist, die gestellten Forschungsfragen zu adressieren und damit die effektive Kooperation von Robotern im Team ermöglicht. Die Abstraktion zur möglichst einheitlichen Verwendung von Fähigkeiten auf der einen und die Komplexität einzelner Fähigkeiten und Roboter auf der anderen Seite ist dabei das zentrale Spannungsfeld, welches die Entwicklung und Entscheidungen beeinflusst hat.

In diesem Kapitel wird zunächst das Gesamtkonzept der Arbeit vorgestellt, bevor die Roboterfähigkeiten und das umgebende Framework im Detail erläutert werden. Mit der formellen Herleitung des Behavior-Tree-Ansatzes wird dann die Kernkomponente dieser Arbeit erklärt, um zuletzt die durch diese Komponenten mögliche fähigkeitsbasierte Koordination im Team aufzuzeigen.

4.1. Konzept

Kernkonzept für die fähigkeitsbasierte Kooperation sind die *Roboterfähigkeiten (Capabiltites)*, welche als funktionale Einheit für das Erfüllen einer Mission verkettet und darüber hinaus mit anderen Robotern koordiniert werden können. Durch das Verwenden von *Behavior Trees als Modell für die Missions- und Fähigkeitsdarstellung* wird eine implizite Verwendung der Fähigkeiten in einer Mission ermöglicht, was wiederum für die Koordination zwischen den Systemen ausgenutzt werden kann. Ein *Framework* implementiert die dafür nötigen Funktionen und Zugriffsmöglichkeiten. Durch die Kombination dieser Ansätze wird eine *fähigkeitsbasierte Kooperation* ermöglicht, welche *variable Autonomiegrade* und *heterogene Systeme* für eine Mission berücksichtigen kann.

Roboterfähigkeiten (Capabilities) sind in sich geschlossene Verhaltensbausteine, die die konkrete Fähigkeiten des Roboters als semantische Einheit abstrahieren. Die Granularität einer Fähigkeit ist nicht begrenzt und kann sowohl sehr konkret, etwa „Aktiviere ein Licht “, aber auch sehr abstrakt, etwa „Untersuche ein Objekt“, formuliert werden. Jede Fähigkeit besteht aus:

- Modell-Informationen, um die Fähigkeit zu identifizieren & beschreiben

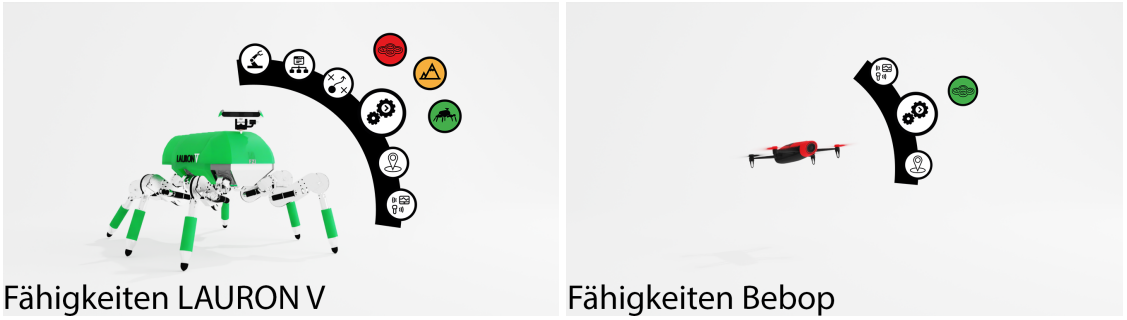


Abbildung 4.1.: Roboterfähigkeiten und ihre unterschiedliche Ausgestaltung. Die Fähigkeiten sind zunächst abstrakte Fähigkeiten (schwarz/-weiße Darstellung) und damit roboterunabhängig. Jede abstrakte Fähigkeit kann dann durch eine oder mehrere konkrete Implementierungen (farbige Darstellung) umgesetzt werden. Während komplexe Roboter wie LAURON V eine Reihe von Fähigkeiten und teilweise mehrere Implementierungen für eine Fähigkeit besitzen, haben weniger autonome Roboter oft nur einzelne Fähigkeiten oder sehr spezifische Implementierungen.

- Einer oder mehreren Ablaufsteuerungen (Koordinator), welche die Komplexität verschiedener interner Aufrufe kapselt
- Einer oder mehreren konkreten Implementierungen

Fähigkeiten sind zunächst abstrakt und roboterunabhängig, solange nur das Modell verwendet wird. Erst bei der Instanziierung einer konkreten Implementierung (Abb. 4.1), etwa während der Ausführung einer Mission, wird die Ablaufsteuerung ausgeführt, welche dann konkrete Funktionen eines Roboters über die *Robot-Abstraction-Layer* anspricht.

Abstrakte Fähigkeiten sind eindeutig, können jedoch durch unterschiedliche Implementierungen umgesetzt werden. Haben zwei Roboter die gleiche abstrakte Fähigkeit, etwa einen Zielort zu erreichen, kann die Umsetzung durch die jeweiligen Eigenheiten des Roboters erfolgen, z.B. durch Laufen oder Fliegen. Gleichzeitig wird durch die äquivalente abstrakte Fähigkeit eine Austauschbarkeit innerhalb einer Mission erzielt, beide Roboter können einen Zielort erreichen und damit gleichberechtigt für diesen Teil der Mission verwendet werden.

Durch die Ablaufsteuerung wird das für die Fähigkeit benötige Vorwissen für die Ausführung einer Fähigkeit auf das Nötigste reduziert. Spezifische Implementierungsdetails, etwa, dass eine Bremse gelöst werden muss, bevor der Roboter sich bewegen kann, werden so in der Ablaufsteuerung gekapselt. Auch die externe Parametrisierung von Fähigkeiten kann dadurch reduziert werden, wenn sie für die Mission nicht weiter relevant ist. Während die Fähigkeit, einen Ort zu erreichen, beispielsweise eine Genauigkeitsanforderung für den Zielort oder eine gewünschte Geschwindigkeit enthalten kann, ist die Angabe von spezifischen Parametern wie der dabei zu nutzenden Flughöhe nicht erforderlich, aber auch

nicht strikt ausgeschlossen. Diese Ausgestaltung der Roboterfähigkeit erzielt damit folgende Eigenschaften:

- Spezifische Fähigkeiten werden als semantische Bausteine gekapselt, welche einfacher für das Erfüllen einer Mission genutzt werden können
- Komplexe Handlungen und die Ansteuerung einer Vielzahl an Komponenten werden mit möglichst einfacher Schnittstelle abstrahiert und vereinfachen damit den Einsatz
- Die roboterabhängige Implementierung wird auf eine roboterunabhängige Darstellung abgebildet, dadurch können auch unterschiedliche (heterogene) Systeme die semantisch gleiche Fähigkeit individuell implementieren
- Durch die einheitliche Adressierung der Fähigkeiten können Fähigkeiten zwischen den Robotern einfacher ausgetauscht werden

Ein Framework (Abb. 4.2) bildet die Grundlage für die Definition und Ausführung der *Fähigkeiten* sowie der Koordination der Fähigkeiten zwischen unterschiedlichen Systemen. Es stellt die Verbindung der Fähigkeiten zur *Roboter Hard-*

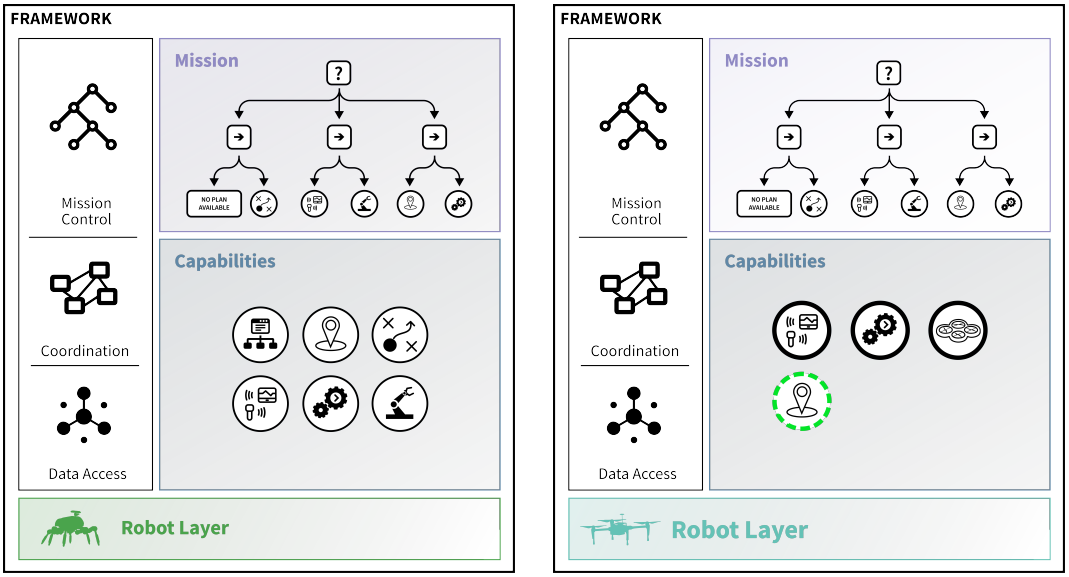


Abbildung 4.2.: Das Framework für die fähigkeitsbasierte Kooperation bietet den Zugriff auf die jeweiligen Fähigkeiten durch eine einheitliche Definition. Die Menge und Ausgestaltung der Fähigkeiten ist jedoch nicht beschränkt. Es stellt zudem die notwendigen Funktionalitäten für die Koordination der Fähigkeiten aber auch der Datenfusion bereit. Über die Robot Layer werden die roboterspezifischen Funktionen aufgerufen. Links: Framework und Fähigkeiten eines Laufrobters. Rechts: Framework und Fähigkeiten eines Flugroboters.

ware (Robot Layer) her und führt eine *Mission*, aufgebaut aus Fähigkeiten, aus. Insbesondere für das dynamische Erstellen einer Mission und Abbilden der Fähig-

keiten auf konkrete Implementierung gibt es eine *Mission Control*, welche wiederum die *Coordination* Komponente zum Austausch mit anderen Systemen nutzt. Darüber hinaus werden Fragestellungen die nicht im Fokus dieser Arbeit stehen, jedoch notwendigerweise gelöst werden müssen, etwa der Zugriff (*Data Access*) und Synchronisation von Daten, vom Framework adressiert. Wichtigste Design-Entscheidung dieses Konzeptes ist es, die Komplexität einzelner Roboter nicht von vornherein durch das Framework einzuschränken, sondern bewusst nur einen Rahmen zu geben, der den Zugang zu komplexeren Implementierungen (den Fähigkeiten) ermöglicht.

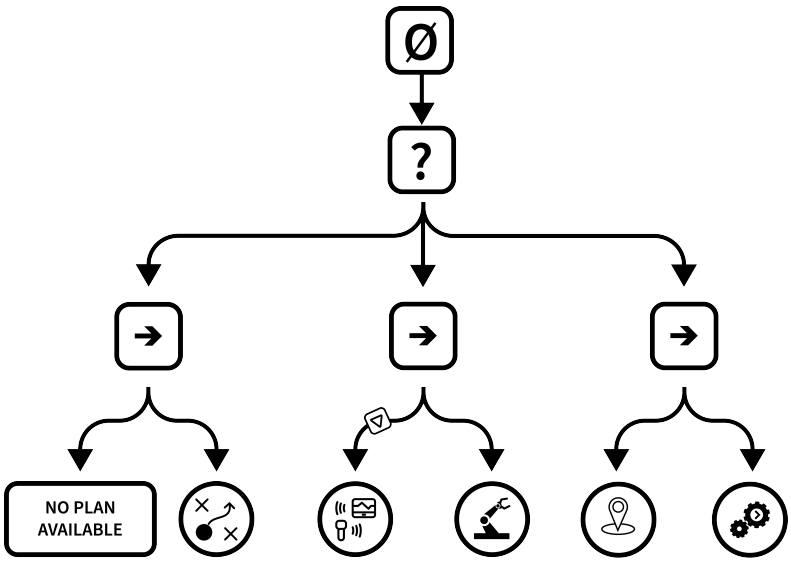


Abbildung 4.3.: Beispiel einer Missionsdarstellung mittels Behavior Tree und abstrakter Fähigkeiten. Durch einen Fallback-Knoten werden nacheinander die 3 Sequenz-Knoten ausgeführt, bis einer ein *succeeded* als Ergebnis meldet. Die erste Sequenz prüft, ob es bereits einen Plan gibt, wenn nicht, wird dieser erstellt. Die zweite Sequenz prüft, ob Zielobjekte detektiert werden können und beginnt das Einsammeln, wenn dem so ist. Durch einen „shovable"-Dekorator wird die Detektion der Zielobjekte als auslagerbar deklariert. Die letzte Sequenz führt eine Selbstlokalisierung durch, bevor zum nächsten Ziel navigiert wird. Um die Mission wirklich auszuführen, müssen die abstrakten Fähigkeiten weiter aufgelöst werden. Da die Fähigkeiten selbst ebenfalls als Behavior Tree modelliert werden, kann dieser Baum direkt anstatt der abstrakten Fähigkeit eingesetzt werden.

Behavior Trees als Modelle für die Missions- und Fähigkeitsdarstellung sind ein zentraler Baustein für die Funktion des Gesamtkonzepts. Die *Fähigkeiten* können als Bausteine für die Robotermission genutzt werden, welche wiederum vorgibt, unter welchen Bedingungen die Fähigkeiten eingesetzt werden sollen. Oft werden HFSMs für diesen Zweck genutzt: Durch eine Abbildung der Zustände bzw. der Übergänge auf die Fähigkeiten wird modelliert, wann welche Fähigkeit

eingesetzt werden soll. Jedoch ist die Modellierung starr, das Verschieben einzelner Zustände erfordert das Anpassen von Zustandsübergängen. Durch den Einsatz von Behavior Trees, sowohl für die Modellierung der Mission als auch in der Ausführung der Fähigkeit selbst, kann diese Abbildung signifikant vereinfacht werden, da Behavior Trees ineinander eingesetzt werden können, ohne einen zusätzlichen Modelierungsoverhead zu erzeugen.

Eine Mission wird zunächst durch den Aufbau eines Behavior Trees mit abstrakten Fähigkeiten definiert (Abb. 4.3). Eine Fähigkeit wiederum definiert ihren Koordinator, also die intrinsische Ablaufsteuerung der Fähigkeit, ebenfalls als Behavior Tree, welcher die RAL Funktionen aufruft. Ähnlich wie bei HTNs kann ein Fähigkeitskoordinator aber auch weitere Fähigkeiten nutzen und damit Tasks immer weiter hierarchisch zerlegen. Sind alle Fähigkeiten und deren Modellierungen bekannt, kann eine abstrakt modellierte Mission daher so lang durch konkretere Fähigkeiten aufgelöst werden, bis ein Behavior Tree mit konkreten Kommandos und Bedingungen erzeugt wird, welcher sofort ausgeführt werden kann. Während Behavior Trees eine gute theoretische Grundlage für diesen Mechanismus bilden, erfordert die tatsächliche Nutzung eine Erweiterung bisheriger Behavior-Tree-Konzepte. Für die Missions- und Fähigkeitsdarstellung im Team wurde daher eine eigene Behavior-Tree-Umsetzung entwickelt, welche Merkmale wie Datenkanten, verteilte Ausführung von Subtrees und asynchrone Funktionsaufrufe umsetzt und damit den Grundstein legt, eine Mission nicht nur auf einem Roboter auszuführen, sondern über mehrere Roboter zu verteilen.

Fähigkeitsbasiere Kooperation nutzt die Modellierung der Fähigkeiten und der Mission im gezeigten Framework, um variable Autonomiegrade bei Robotern zu berücksichtigen und dadurch alle Vorteile und Fähigkeiten der unterschiedlichen Systeme zu nutzen (Abb. 4.4). Durch das *Framework* werden die jeweiligen *Roboterfähigkeiten* im Team bekanntgegeben, andere Roboter können diese Fähigkeiten für Ihre Mission nutzen, sie *erweitern* ihre Fähigkeiten und dadurch ihren Autonomiegrad. Für die Verteilung wird ein Auktionsansatz verwendet, dadurch können individuelle Kosten für die Fähigkeiten, aber auch der Zustand des jeweiligen Roboters berücksichtigt werden. Die gewählte Modellierung ermöglicht es darüber hinaus, dass Fähigkeiten aus anderen Fähigkeiten aufgebaut werden. Mehrere Fähigkeiten eines, aber auch mehrerer, Roboter können daher *kombiniert* werden, um eine Aufgabe zu erfüllen. Wird die Mission auf mehrere Roboter verteilt, wird die Mission z.B. durch Vor- oder Nachbedingungen entsprechend erweitert, die Ausführung der einzelnen Teile damit im gesamten Team koordiniert. Sollte selbst das Ausführen eines Frameworks nicht möglich sein, etwa weil ein Roboter über keine eigene, zugängliche Rechenleistung verfügt, kann letztendlich auch ein anderer Roboter im Team die eine Fähigkeit für diesen Roboter *synthetisieren*. Durch das Ausführen des jeweiligen Kommunikationstreibers wird eine RAL *instanziiert*, welche wiederum die Fähigkeiten des Zielsystems bietet, die dann für die Mission genutzt werden können.

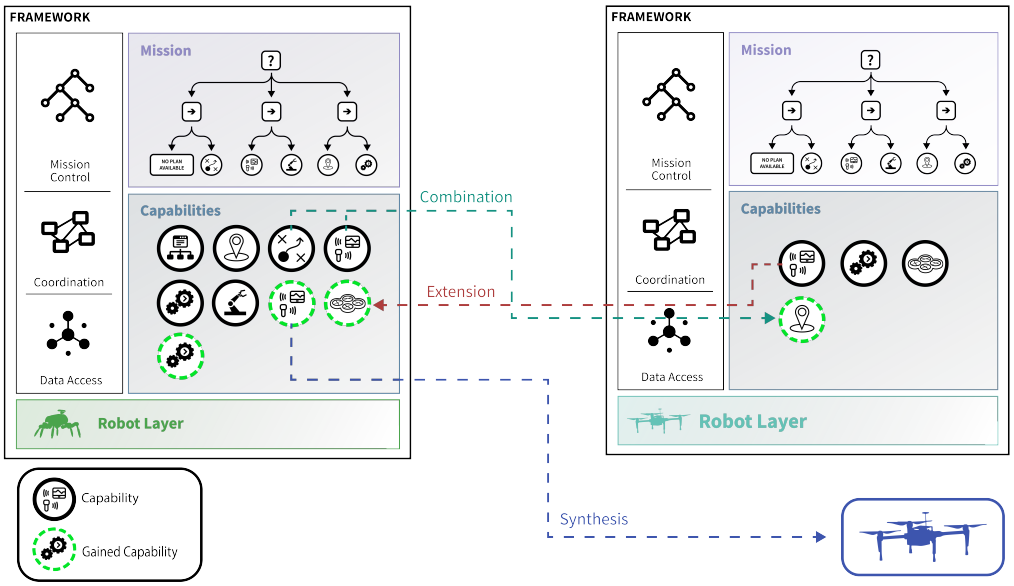


Abbildung 4.4.: Mögliche Kooperationen im Framework auf Fähigkeitsbasis. Durch die verteilte Ausführung können Fähigkeiten durch die anderer Roboter *erweitert* oder durch die *Kombination* mit anderen Fähigkeiten innerhalb des Teams zu neuen kombiniert werden. Auch Roboter ohne eigene Rechenleistung können durch die *Synthese* neuer Fähigkeiten, in diesem Fall das *Instanziieren* eines Kommunikationstreibers, für die Ausführung von Missionsteilen eingebunden werden.

4.2. Roboterfähigkeiten

Als *Fähigkeit* wird etwas bezeichnet, mit dem ein Roboter in der Lage ist, eine Aktivität auszuführen, um eine Funktion zu erfüllen. Die beabsichtigte Funktion ist für Roboter jedoch nicht ohne Weiteres mit einer Aktivierung oder einem Funktionsaufruf identisch. Die Funktion *vorwärts fahren* eines Modellautos wird z.B. durch das Aktivieren des Antriebsmotors umgesetzt. Aber selbst bei diesem simplen Beispiel gibt es keine 1:1 Abbildung einer Fähigkeit auf eine Funktion des Roboters. Wird ein langsam betriebener Motor zum *vorwärts Fahren* genutzt, könnte der gleiche Motor durch ruckartiges Beschleunigen für ein *Springen* des Gesamtsystems genutzt werden. Die anders parametrisierte Aktivierung des Motors führt also zu einer anderen Fähigkeit des Systems. Sind zwei Systeme darüber hinaus unterschiedlich aufgebaut, könnten sie durchaus die gleiche Fähigkeit aufweisen, die jedoch vollständig unterschiedlich durch die Hard- und Software des Roboters realisiert wird, etwa bei ketten- oder radgetriebenen Systemen. Diese Diversität legt nahe, dass eine Modellierung der Fähigkeiten erfolgen muss, mit dem die semantische Bedeutung einer Fähigkeit auf die technische Implementierung des Roboters abgebildet werden kann, um eine Vergleich- und Austauschbarkeit herzustellen. Gleichzeitig unterstreichen bereits diese Beispie-

le, dass eine solche Modellierung in keinem Fall die Implementierung beschränken darf.

Diese Arbeit definiert Fähigkeiten als elementare Funktionsbausteine, die für die Ausführung einer Mission zur Verfügung stehen und das zentrale Element für die Koordination der Kooperation im Team darstellen. Sie nehmen eine höhere semantische Ebene ein als direkte Funktionsaufrufe, da sie in ihrer Definition zunächst nicht auf eine spezifische Implementierung festgelegt sind und komplexere Abläufe beinhalten können. Wie bei allen Komponenten dieser Arbeit ist gerade die Möglichkeit, unterschiedlicher Fähigkeiten einheitlich zu beschreiben und damit einhergehend sehr unterschiedliche Komplexität und Granularität zu modellieren, eine Kern-Anforderung. Um dies zu realisieren, wird eine Fähigkeit daher durch 3 Teile definiert: Ein **Modell**, einen **Koordinator** und die eigentliche **Implementierung** (Abb. 4.5).

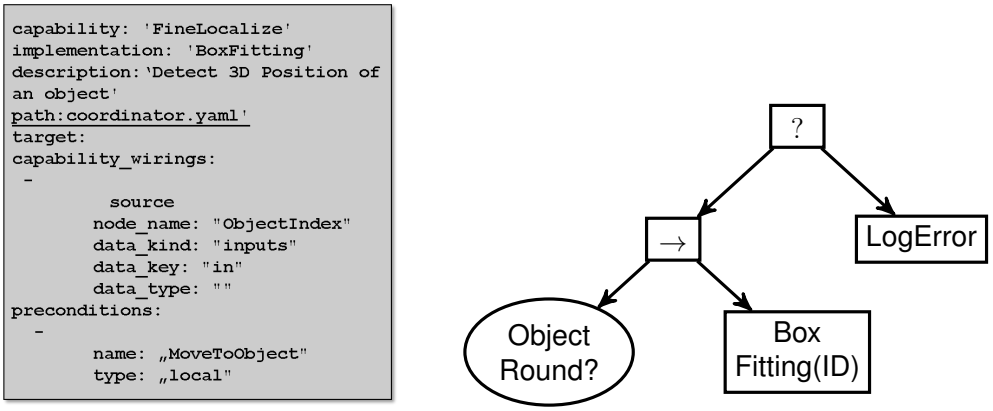


Abbildung 4.5.: Jede *Fähigkeit(Capability)* besteht aus einem Modell (links), einem Koordinator (rechts) und einer Implementierung. Das Modell beschreibt die Fähigkeit und ermöglicht eine automatische Nutzung. Der Koordinator abstrahiert die nötigen Schritte der Ausführung und bietet damit eine einfach zu nutzende Schnittstelle. Die eigentliche Implementierung erfolgt separat, z.B. als ROS-Node.

4.2.1. Implementierung der Fähigkeit

Die Implementierung einer Fähigkeit ist im einfachsten Fall der Aufruf einer vom Roboter zur Verfügung gestellten Schnittstelle (z.B. das Aktivieren eines Motors) über die *Robot Abstraction Layer*, kann jedoch auch durch die komplexe Verbindung verschiedener Algorithmen (z.b. das Erkennen von Landmarken auf Bilddaten) realisiert werden. Da in der Robotik oftmals bereits eine signifikante Codebasis in verschiedenen Frameworks existiert, ist es wichtig, diese Funktionen mit möglichst geringem Aufwand in eine Fähigkeit einzubinden und Code-Dopplungen dadurch zu vermeiden. Das in dieser Arbeit entwickelte Framework

baut auf ROS als Softwareframework auf. ROS-Pakete enthalten eine oder mehrere ROS-Nodes, welche unabhängig ausführbar sind und über ROS-Messages mit anderen Bausteinen kommunizieren oder Funktionen zur Verfügung stellen. Üblicherweise ist also die Implementierung einer Fähigkeit identisch mit einer ROS-Node oder einer ihrer Schnittstellen, die wiederum eine spezifische Funktion der *Robot Abstaction Layer* aufruft. Während es keine strikten Vorgaben bezüglich der Implementierung gibt, helfen einige Designentscheidungen dabei, eine Funktionalität als modulare Fähigkeit bereit zu stellen z.B.:

- Funktionen sollten möglichst direkt durch den Aufruf von externen Schnittstellen ausführbar sein, etwa ROS-Actions oder Services. Dadurch wird eine zu enge Kopplung zwischen Ablaufsteuerung und Implementierung vermieten und eine Kapselung von Funktionen erleichtert.
- Daten sollten durch den Zugriff auf zentrale Datenquellen beschafft werden (anstatt sie im Funktionsaufruf zu übergeben). Dadurch wird die Komplexität von Interfaces reduziert und damit die Austauschbarkeit erhöht.
- Aktive Funktionen, etwa das Durchsuchen einer Datenquelle nach einem Wert, lassen sich einfacher nutzen als passive, etwa das Scannen der Daten mittels Hintergrundprozess, da sie meist keine asynchrone Kommunikation erfordern.

Welche Funktionalität ein Paket bereitstellt, ist nicht oder meist nur für den Entwickler oder aufgrund der Dokumentation erkennbar. Flexible Schnittstellen bedeuten zudem auch, dass diese nicht einheitlich zu verwenden sind. Ein Paket oder eine Schnittstelle richtig zu nutzen, erfordert daher Einblick in dessen Funktionsweise, weshalb eine Implementierung immer in Kombination mit einem *Koordinator* eingesetzt wird.

4.2.2. Koordinator der Fähigkeit

Der Koordinator adressiert das Schnittstellenproblem durch eine Modellierung des Ablaufs der benötigten Aktivitäten einer Fähigkeit. Die Kapselung der Prozessinformationen im Koordinator ermöglichen es, eine Fähigkeit als semantischen Baustein zu nutzen, ohne dessen Implementierungsdetails kennen zu müs sen. Das Bewegen eines Armes etwa könnte erfordern, dass zunächst eine Bremse deaktiviert und ein bestimmter Modus aktiviert wird, bevor Steuerbefehle gesendet werden. Solche Details sind lediglich für den Entwickler einer Komponente bzw. Fähigkeit wirklich relevant und werden durch den Koordinator in einen einfachen Aufruf abstrahiert, die komplexen Belange der Algorithmen also nach außen hin vereinheitlicht. Als Koordinator kommen verschiedene Arten der Umsetzung in Frage, die im Rahmen dieser Arbeit untersucht wurden.

In einer ersten Implementierung von Koordinatoren für den SpaceBot Cup [Heppner et al., 2015] wurde das ROS-Framework SMACH genutzt, um Fähigkeiten mittels HFSM zu modellieren (Abb 4.6). Durch die Modellierung können kom-

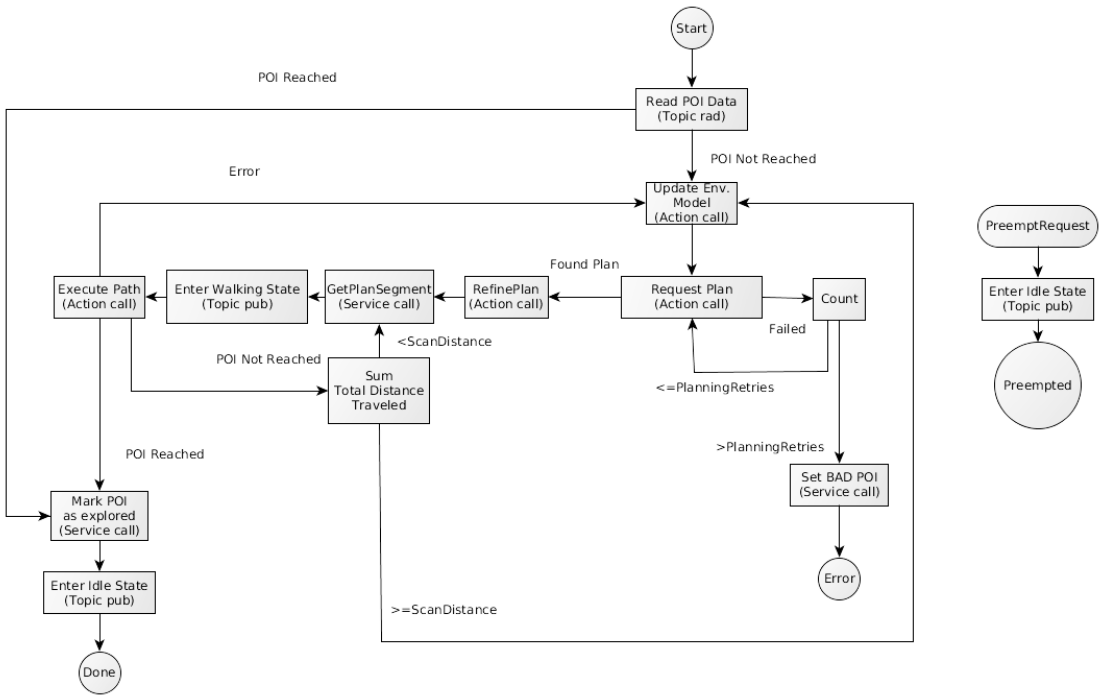


Abbildung 4.6.: Beispiel einer *MoveToPOI* Fähigkeit, modelliert mit Hilfe des *SMACH* Frameworks für den SpaceBot Cup. Die Fähigkeit kommt mit wenigen Vorgaben (PoI ID, Parameter für die Art der Lokomotion) aus und beschafft sich weitere notwendige Daten selbst von zentralen Informationsspeichern. Quelle: [H+15]

plexe Abläufe wie das Erzeugen und Modifizieren eines Roboterpfades sowie roboterspezifische Eigenheiten, etwa das Aktivieren eines Laufmusters, abgebildet werden. Die Fähigkeit selbst beschafft sich weitere Informationen, im Beispiel etwa die Zielposition eines PoI, anhand der zentral verfügbaren Daten und prüft den Fortschritt der Ausführung, um diesen an eine Missionssteuerung zurück zu melden. Konkrete Implementierungen werden über ROS-Messages, z.B. *Request Plan*, für den Aufruf eines globalen Planers, eingebunden. Durch die Verwendung von ROS-Actions als Einstiegspunkt für den Koordinator können auch direkt weitere Fähigkeiten (bzw. deren Koordinatoren), etwa *Update Environment Model*, genutzt werden. Um den Koordinator vergleichbar zu dem anderer Fähigkeiten zu nutzen, muss ein möglichst einheitliches Interface für den Aufruf der Fähigkeiten genutzt werden. In dieser Implementierung wurden für den ROS-Action-Aufruf daher zwei generische IDs als Parameter übergeben, die, je nach Fähigkeit, etwa für die Identifikation der Zielpose verwendet werden können.

Mit einem IEC61499 Modul wurde das Koordinator-Konzept im Rahmen des ReApp Projektes [Awad et al., 2016] auch mittels industrieller Standards umgesetzt. Auch hier wird der Koordinator durch einen Zustandsautomaten realisiert, dieser kommuniziert mit den weiteren Funktionsblöcken jedoch mittels simpler Event-Kommunikation, was die Mächtigkeit des Konzeptes deutliche reduziert,

aber andererseits für den Einsatz im Industriekontext qualifiziert.

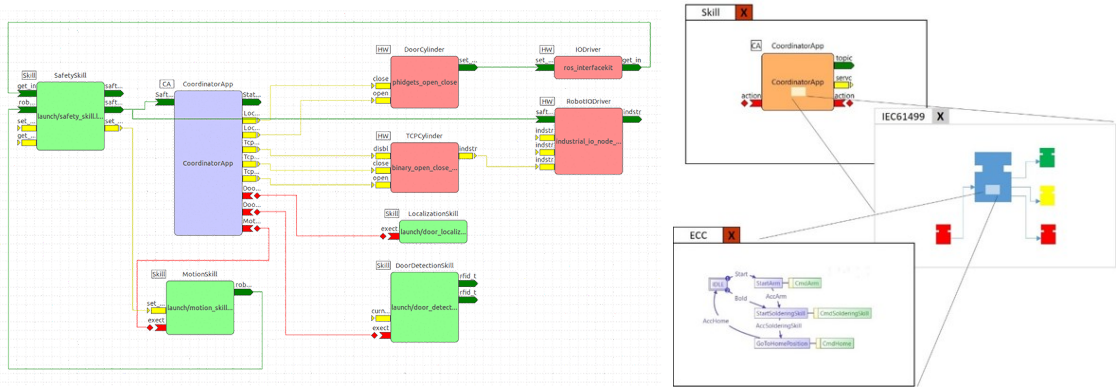


Abbildung 4.7.: Beispiel eines ReApp *Skills* bestehend aus Hardware-Bausteinen (rot), Skill-Bausteinen (grün) und einem Koordinator (blau), der die Ausführung kontrolliert. Der Koordinator ist als IEC61499 Baustein umgesetzt, welcher ein Execution Control Chart (ECC) nutzt, um die Ablaufsteuerung zu modellieren. Quelle: [H+16]

Das Koordinator-Konzept wird in der finalen Ausprägung mittels Behavior-Trees (BTs) realisiert (Abbildung 4.8).BTs können die gleiche Funktionalität wie eine HFSM abbilden, haben jedoch inhärente Vorteile, insbesondere die Möglichkeit, BTs ohne Overhead ineinander einsetzen und verschieben zu können sowie eine hohe Reaktivität auf geänderte Bedingungen.

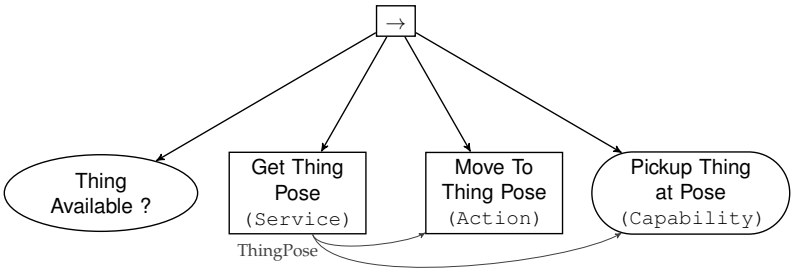


Abbildung 4.8.: Beispiel eines Koordinators für die Fähigkeit *Pickup Thing*. Der Koordinator prüft zunächst, ob das Zielobjekt überhaupt vorhanden ist. Wenn ja, wird die Objektposition durch einen ROS-Service aus der globalen Datenbank angefragt und dann per ROS-Action angelaufen. Ist der Roboter vor dem Objekt angekommen, wird die eigentliche Manipulation zum Einsammeln des Objektes durch eine weitere Fähigkeit koordiniert.

Durch diese positive Eigenschaft eignet sich der BT sowohl für den Koordinator, aber auch für die Modellierung einer kompletten Mission. Diese einheitliche Modellierung ermöglichst es, den Abbildungs- bzw. Instanziierungsschritt zwischen Mission und Fähigkeit deutlich zu vereinfachen, da der BT direkt als Teil der Modellinformation verwendet wird. Dabei ergeben sich ähnliche Fragen wie bei der

Verwendung von HFSM, die jedoch mit dem eigens entwickelten Behavior Tree Framework (Abschnitt 4.4) adressiert werden können. Datenkanten erlauben etwa, die Parametrisierung der Fähigkeit im BT zur berücksichtigen. Darüber hinaus bietet es einige besondere Eigenschaften, wie z.b. die Utility-Berechnung (Abschnitt 4.5.3), welche für die Vergabe von Fähigkeiten im Team genutzt wird. *Capabilities* wurden in der Umsetzung als Knotentyp für die BTs definiert, wodurch direkt der jeweilige *Koordinator* ohne weitere Indirektstufe aufgerufen wird.

4.2.3. Modell der Fähigkeit

Das Modell einer Fähigkeit erweitert diese um ein maschinenlesbares Datenblatt, wodurch diese automatisiert verwendet werden können. Die primäre Aufgabe des Modells ist es, die Fähigkeit selbst zu identifizieren, um sie an andere Systeme kommunizieren und mit dessen Fähigkeiten vergleichen zu können. Die Bekanntgabe der Fähigkeiten im Team, aber auch die Auktion von Fähigkeiten nutzen das Modell der Fähigkeiten als Austauschformat. Darüber enthält das Modell etwaige Randbedingungen oder Eigenschaften einer Fähigkeit, etwa die zu erfüllenden Vorbedingungen. Zwei unterschiedliche Ansätze für die Modellierung wurden für diese Arbeit untersucht:

Der erste Ansatz ist ein **MDE-Ansatz**, bei dem das Modell die Eigenschaften der Fähigkeit möglichst vollständig abbildet. Dieser Ansatz [Awad et al., 2016], [Zander et al., 2015], [Bastinos et al., 2014] definiert Software(SW)-, und Hardware Acces(HA)-Komponenten als grundlegende, wiederverwendbare Softwarebausteine, welche auf Basis einer Ontologie klassifiziert und beschrieben werden, in der konkrete Eigenschaften für bestimmte Komponenten gefordert werden (siehe Abbildung 4.9). Mehrere Komponenten können zusammen mit einem Koordinator (siehe 4.2.2) zu einem *Skill*[1], also einer kompletten Fähigkeit Kombiniert werden.

Jeder *SW-Type* oder *HW-Type* definiert ein Bündel aus Attributen (etwa ein Gewicht für ein konkretes Gerät oder eine Komplexität für einen Algorithmus), ROS-Interfaces (etwa ein image-Topic) und Capabilities, die von einer Komponenten dieses Typs erwartet werden. Dadurch können Eigenschaften für die automatische Code-Erstellung abgeleitet und die Kompatibilität zwischen den unterschiedlichen Komponenten geprüft werden. Durch das Verwenden von semantisch relevanten Modellen, in diesem Fall durch das Modellieren von Domänenontologien, ist es durch diese Beschreibung möglich, lediglich mit dem Modell zu arbeiten und z.B. durch Reasoning zu entscheiden, ob die Kombination von Fähigkeiten ggf. eine weitere Fähigkeit nach sich zieht (Abbildung 4.10). Die Informationen des Koordinators oder der Implementierung selbst sind bei diesem Ansatz erst bei der eigentlichen Ausführung relevant.

[1] Während in dieser Arbeit der Begriff Capability für eine Fähigkeit verwendet wird, wurde in dem beschriebenen Modell der Begriff *Skill* verwendet. Da ein *Skill* z.b. nicht als BT modelliert wird, wird der Begriff *Skill* immer dann verwendet, wenn über die Fähigkeit des MDE-

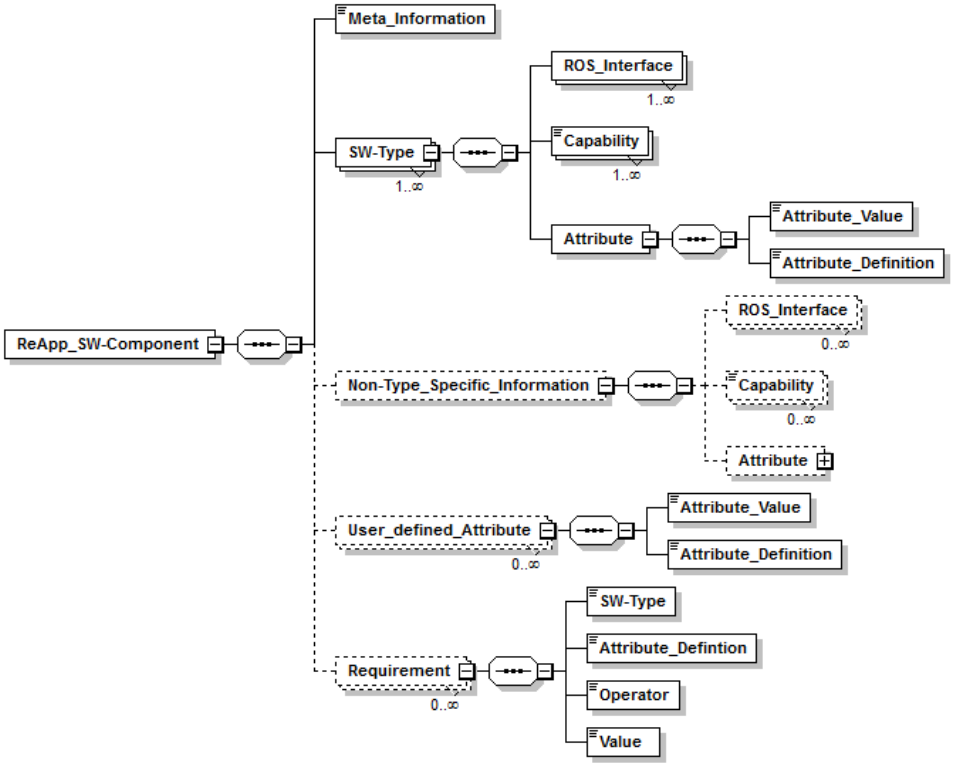


Abbildung 4.9.: Modellierung von Software(SW)-Komponenten durch einen MDE-Ansatz. Jede Komponente wird mittels Meta-Informationen beschrieben und über den SW-Type oder HW-Type klassifiziert. Die Typen definieren eine Menge von Interfaces, Fähigkeiten und Attributen. Für HW-Komponenten (nicht gezeigt) werden darüber hinaus konkrete Hardwareinformationen hinterlegt. Alle Komponenten können Benutzer-Attribute(user-defined), zusätzliche Qualifizierungen und Anforderungen an andere Komponenten mit aufnehmen. Quelle (modifizert) [Zander et al., 2015] ©2015 IEEE

Das Wissen darüber, welche *SW-Type*, *HW-Type* und Fähigkeiten es gibt und welche Eigenschaften diese aufweisen, wird durch Ontologien realisiert. Diese definieren ein Schema für Fähigkeiten und Regelsätze, welche Bedingungen und Folgen beinhalten. Konkrete Komponenten oder Fähigkeiten werden als Instanz dieses Schemas erstellt und können in Inferenzschritten automatisiert um abgeleitete Eigenschaften erweitert werden. Abbildung 4.11 zeigt einen Ausschnitt dieser Ontologie für den industriellen Kontext [Zander et al., 2015].

Der Zweite Ansatz ist der im Rahmen dieser Arbeit erstellte **fähigkeitsbasierte Ansatz**. Er ist primär schnittstellenbasiert, nutzt jedoch gleichzeitig die *BT*-Modellierung als Kernkomponenten (Abb. 4.12). Das Modell einer Fähigkeit beinhaltet nur die Informationen, die für eine automatisierte Nutzung bzw. Refe-

Ansatzes gesprochen wird

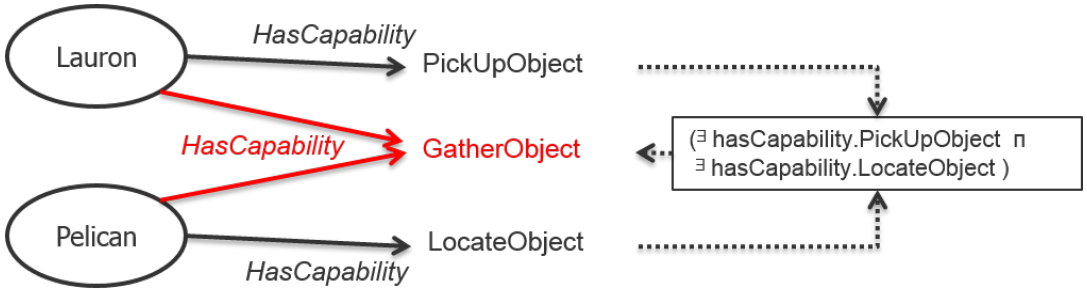


Abbildung 4.10.: Durch das detaillierte Modellieren von Fähigkeitseigenschaften sowie Regeln für die Kombination und Äquivalenz können mittels Reasoning und Inferenz neue Eigenschaften abgeleitet werden. In diesem Beispiel wird aus dem Vorhandensein der Fähigkeiten *PickUpObject* und *LocateObject* geschlossen, dass auch die Fähigkeit *GatherObject* vorhanden ist. Die Einzelnen Fähigkeiten müssen dabei nicht notwendigerweise durch den gleichen Roboter zur Verfügung gestellt werden. Hier kann der Laufroboter LAURON das Objekt aufnehmen und die Flugdrohnen Pelican das Objekt lokalisieren.

renzierung innerhalb der Mission erforderlich sind, verzichtet aber auf jede darüber hinausgehende Information. Im Gegensatz zur MDE-Variante werden zusätzliche zu den Modell-Informationen auch die Koordinatordetails, genutzt um zusätzliche Eigenschaften, etwa die Kosten einer Fähigkeit, zu ermitteln.

Eine abstrakte Fähigkeit wird durch ihren Namen in Verbindung mit der Schnittstelleninformation eindeutig beschrieben. Die einfach zu lesende yaml-Datei (inteface.yaml) beinhaltet daher genau diese Informationen, also den Namen und eine für den menschlichen Nutzer gedachte Beschreibung. Darauf folgen Schnittstelleninformationen, also *Inputs*, *Outputs* und *Optionen*, die von der Fähigkeit genutzt werden. Innerhalb eines BT wird eine solche abstrakte Fähigkeit als Knoten mit entsprechendem Interface dargestellt, welche erst durch Zuordnen eines Koordinators ausführbar wird und bis dahin einen automatisierten Austausch der Fähigkeit ermöglicht.

Der Koordinator wird, zusammen mit einigen weiteren Meta-Informationen, in der implementation.yaml[2] gespeichert. Neben einer Versionsinformation enthält diese für die Missionssteuerung relevante Vorbedingungen und wo diese erfüllt werden müssen (z.B. lokal) und den Koordinator der (konkreten) Fähigkeit als BT. Im entwickelten Framework werden Modellinformation und Koordinator in einem gemeinsamen Paket abgelegt (siehe 4.12 links), da sie für gewöhnlich gleichzeitig definiert und genutzt werden. Es ist jedoch auch möglich, zu jedem

[2]Um Verwirrungen zu vermeiden: Die in den yaml Dateien gespeicherten *implementations* sind *Fähigkeitsimplementierungen*, also ein konkreter *Koordinator* für eine Fähigkeit. Die in Abbildung 4.5 genannte Implementierung bezieht sich auf den Code, der die tatsächlichen Algorithmen implementiert.

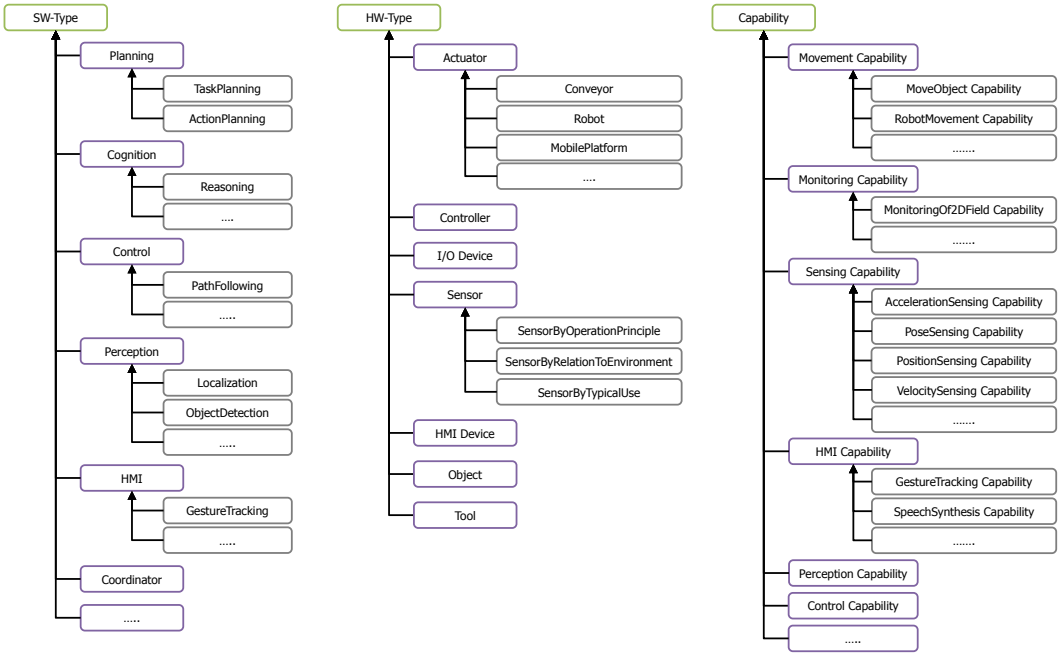


Abbildung 4.11.: Drei Ontologien für die Modellierung der HW-,SW-Module und deren Fähigkeiten im industriellen Kontext. Die Ontologien sind hierarchisch aufgebaut und zielen darauf ab, die Domäne möglichst vollständig abzudecken. Sie werden als Beschreibung für eine einzelne Komponente oder eine komplette Lösung aus verschiedenen Bausteinen genutzt. Es wird deutlich, dass die Erfassung solcher Fähigkeiten problematisch ist, da insbesondere unterschiedliche Granularitäten der Beschreibung existieren. Während einige Fähigkeiten auf sehr konkrete Use-Cases abzielen (z.B. SpeechSynthesis Capability), sind andere allgemein gehalten (z.B. RobotMovement Capability). Quelle [Zander et al., 2015] ©2015 IEEE

Interface eine beliebige Anzahl an Koordinatoren, also auch gar keine, zu hinterlegen. Ist kein Koordinator hinterlegt, kann lediglich die abstrakte Fähigkeit genutzt werden, das heißt, dass Missionen oder andere Fähigkeiten darauf aufbauen, aber nicht ausgeführt werden können, solange kein Koordinator ausgewählt wurde. Es können aber auch beliebig viele Koordinatoren hinterlegt werden, um unterschiedliche Umsetzungen der gleichen Fähigkeit abzubilden. Die Zuordnung erfolgt entweder durch eine händische Auswahl/Modellierung, durch den Nutzer oder eine Auswertung der jeweiligen Ausführungskosten (siehe 4.5.3).

Dieser gegenüber einer kompletten Modellierung deutlich vereinfachte Ansatz fordert vom Nutzer nur die zwingende Einhaltung bzw. Definition der Schnittstellen einer Fähigkeit. Da diese in der Regel ohnehin aufgrund der Implementierung bekannt sind und eine Funktionalität maßgeblich prägen, ist die Definition sehr viel einfacher. Das erfolgreiche ROS Framework zeigt, dass eine funktionale Entkopplung von Komponenten durch ein definiertes Interface (die ROS-

Abbildung 4.12.: Artefakte einer Fähigkeit. Links: Ordnerstruktur, jede Fähigkeit wird als Ordner im Dateisystem gespeichert und enthält die Modellinformationen in der interface.yaml sowie beliebig viele Implementierungen. Rechts: Modellinformationen bestehen aus dem Fähigkeitsmodell mit Metainformation (Name und Beschreibung) sowie den Interfaces (Options, Inputs, Outputs) und dem Koordinator, bestehend aus ergänzenden Modellinformationen (Vorbedingungen, Version) und dem zugehörigen *BT*. Quelle [OH22]

Messages) sehr gut funktioniert und den Austausch von Komponenten bzw. deren Wiedernutzung fördert.

Im Team gibt jeder Roboter seine Fähigkeiten durch Übermittlung der ihm bekannten Fähigkeits-Interfaces bekannt. Wenn mehrere Roboter die gleiche abstrakte Fähigkeit besitzen, kann durch das einheitliche Interface frei gewählt werden, welcher Roboter die eigentliche Ausführung übernehmen soll (siehe 4.5.4). Da die Koordinatoren für gewöhnlich eng mit dem Implementierungsdetails verzahnt und damit roboterspezifisch sind, ist ein Austausch über Roboter hinweg zunächst nicht vorgesehen, aber auch nicht ausgeschlossen. Ist ein Koordinator vollständig aus anderen (abstrakten) Fähigkeiten konfiguriert, so könnte dieser auch an andere Systeme weitergegeben werden, wodurch diese eine neue Fähigkeit erlernen könnten.

4.3. Multi Roboter Framework

Ein Framework bildet den Rahmen für die Ausführung der anwendungsspezifischen Funktionen eines Systems und stellt die dafür notwendigen Werkzeuge zur Verfügung. Für die fähigkeitsbasierte Kooperation muss ein Framework daher diese Kernaufgaben erfüllen:

- Definition und Ausführung von Fähigkeiten auf unterschiedlichen Robotern
- Austausch und Aggregation von Modellinformationen der Fähigkeiten
- Ausführen einer Mission und Abbildung auf die Fähigkeiten des Roboters
- Koordination der Missionsausführung mit anderen Robotern
- Austausch und Aggregation von Daten der unterschiedlichen Roboter

Beim Blick auf den Stand der Technik (Kapitel 3.2) wird deutlich, dass die meisten eingesetzten Frameworks 3 Ebenen aufweisen: Eine Roboterschicht für die roboterspezifische Ansteuerung, eine reaktive Aufgabenschicht für die koordinierte Handlung und eine deliberative Schicht, welche high-level Pläne erstellt und koordiniert. Auch das in dieser Arbeit entwickelte und im Rahmen von SBC erstmals vorgestellte Framework [Heppner et al., 2015, Heppner et al., 2017] folgt einer solchen dreistufigen Aufteilung (Abbildung 4.13).

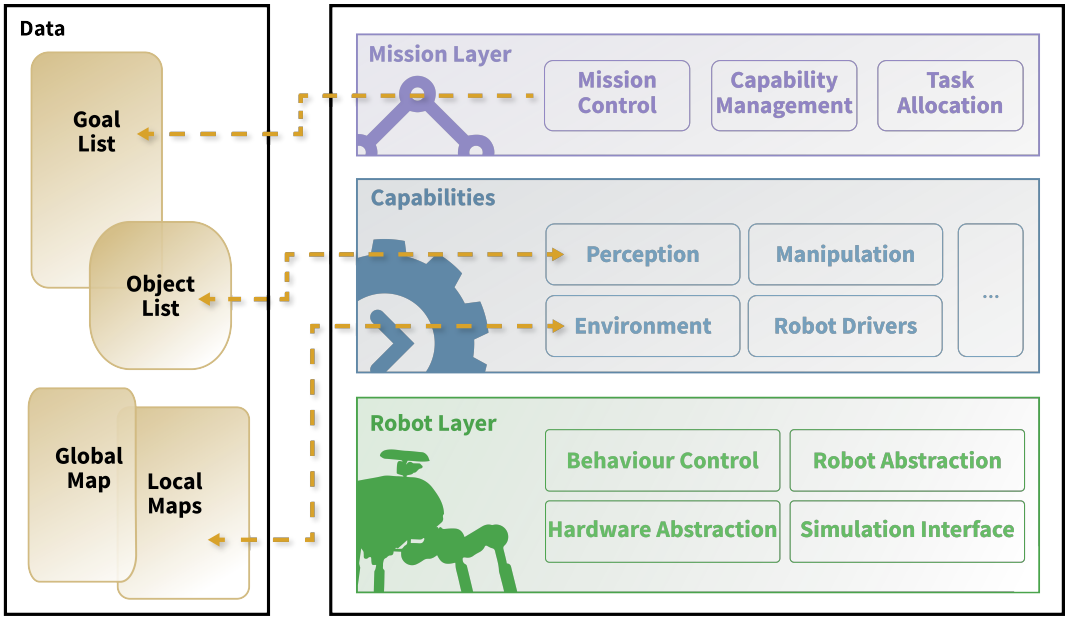


Abbildung 4.13.: Architektur des Framework für die Ausführung komplexer Missionen und fähigkeitsbasierte Kooperation. Die Robot Layer (grün) beinhaltet die roboterspezifischen Funktionen, welche über eine RAL aufgerufen werden, der Skill Layer (blau) stellt eine Menge von erweiterbaren Fähigkeiten (Capabilities) zur Verfügung, welche durch die Mission Layer (lila) sowohl lokal als auch mit anderen Robotern koordiniert und für die Ausführung einer Mission genutzt werden. Die Daten (gelb) werden global geteilt und stehen jeder Eben zur Verfügung.

Die Roboterschicht (*Robot Layer*) zur Ansteuerung roboterspezifischer Funktionen, die Fähigkeitsschicht (*Skill Layer*), welche primär die verfügbaren Fähigkeiten bereitstellt, und die Missionsschicht (*Mission Layer*), welche die Fähigkeiten

zur Mission zuordnet, die Mission selbst ausführt und die Koordination mit anderen Systemen übernimmt. Die Daten werden über alle Schichten und auch das gesamte Team hinweg als zentraler Speicher verwendet und synchronisiert.

Die Architektur ist darauf ausgerichtet, als Softwarekomponenten auf jedem am Team beteiligten Roboter ausgeführt zu werden und ermöglicht dadurch die dezentrale Koordination, kann jedoch auch als zentrale Instanz eingesetzt werden, welche mit dem jeweiligen Roboter lediglich über die Hardware-Abstraktionsschicht (etwa einer Funkverbindung) verbunden ist.

4.3.1. Robot Layer

Roboter sind überwiegend komplexe Einzelsysteme, meist ohne einheitliche Komponenten, Sprachen oder Schnittstellen. Insbesondere um die spezialisierte Hardware mit ihrem jeweiligen Protokoll anzusteuern, ist eine individuelle Implementierung daher unumgänglich. Neben der reinen Ansteuerung von Komponenten gibt es zudem viele roboterspezifische Eigenschaften, etwa die Umrechnung von Encoder-Ticks zu Radianten unter Berücksichtigung der Getriebeübersetzung, aber auch spezielle Algorithmen, die auf den Roboter ausgelegt und nicht ohne Weiteres übertragbar sind.

Die **Robot Layer** bezeichnet daher eine beliebige Implementierung der Funktionalität auf dem Roboter selbst. Bei dem Laufroboter LAURON V [Rönnau et al., 2014] ist dies zum Beispiel eine verhaltensbasierte Steuerung (*Behavior Control*) mit zahlreichen funktionalen Ebenen [Rönnau et al., 2010, Goeller et al., 2011, Rönnau et al., 2011, Roennau et al., 2011, Roennau et al., 2014b, Roennau et al., 2013, Roennau et al., 2014a, Rönnau et al., 2012]. Diese übernimmt die Ansteuerung der Beine, das Ausregeln des Körpers sowie alle Sicherheitsfunktionen, die speziell auf LAURON angepasst sind. Bei einem kommerziellen Roboter wie Spot [3] könnte dies hingegen eine SDK-Bibliothek sein, welche verschiedene Kommandos zur Verfügung stellt.

Um trotz der speziellen Implementierung die Anbindung an allgemein verfügbare Softwarekomponenten, wie etwa SLAM oder Planungskomponenten, zu ermöglichen werden die Funktionen mittels RAL auf ROS Schnittstellen abgebildet und dadurch so weit wie möglich verallgemeinert. Beispielsweise kann die Soll-Geschwindigkeit für LAURON mittels in ROS üblicher Twist-Nachricht kommandiert werden, welche auf 0-1 normierte Geschwindigkeiten (lateral und rotation) für jede Raumachse vorgibt. Die RAL bildet diese Vorgabe auf intern genutzte Werte ab und aktiviert das geeignete Laufmuster sowie alle dafür nötigen Parameter.

Die reale Hardware wird über die *Hardware Abstraction* angesprochen, welche z.B. Treiber für spezielle Bussysteme bereitstellt oder Spezialkonstruktionen wie

[3] https://www.bostondynamics.com/products/spot Zugriff am 18.02.2024

eigene Gripper [Heppner et al., 2014, Buettner et al., 2018] ansteuerbar macht. Alternativ kann an dieser Stelle auch ein *Simulation Interface* genutzt werden, welches mit einer geeigneten Simulationsumgebung verwendet wird. Für LAURON wurden beispielsweise Anbindungen an die eigens entwickelte Simulationsumgebung RoAdS [Rönnau et al., 2013, Rönnau et al., 2015] sowie den ROS Simulator Gazebo [92] entwickelt.

Die Roboterschicht ist streng genommen keine eigenständige Schicht des Frameworks, sondern eben jene Funktionalität, die bei der Entwicklung des Roboters entstanden ist, unabhängig davon, wie diese umgesetzt ist. Erst die RAL kann als Teil des Frameworks gewertet werden, wobei auch diese für jeden Roboter individuell implementiert werden muss. Da die Implementierung des Frameworks in ROS erfolgt, ist die RAL vor allem dafür zuständig, die Schnittstellen des Roboter auf ROS Kommunikation abzubilden. Eine feste Struktur oder erzwungene Schnittstellen gibt es hierbei nicht. Aufgrund der weiten Verbreitung von ROS haben sich allerdings de-facto Standards gebildet, die von den meisten Entwicklern ohnehin umgesetzt werden.

Die Roboterschicht und deren Komplexität in keiner Weise zu beschränken, ist eine wichtige Designentscheidung, um die variablen Autonomiegrade und heterogenen Systeme einbinden zu können. Einige andere Ansätze bilden die roboterspezifischen Funktionen auf einen festen Satz von Fähigkeiten ab, diese sind jedoch oft der kleinste gemeinsame Nenner aller Systeme, was gerade für heterogene Roboter nicht akzeptabel ist.

4.3.2. Skill Layer

Die **Skill Layer** kapselt die Fähigkeiten (*Capabilities*) eines Roboters und bietet damit die Bausteine für komplexere Fähigkeiten sowie das Erfüllen von Missionen. Das Modell und die Koordinatoren einer Fähigkeit werden in einem an die ROS-Paketstruktur angelehnten Ordner gespeichert (vergleiche 4.2) und können dadurch flexibel verteilt werden. Ohne die notwendigen Implementierungen handelt es sich jedoch zunächst um eine rein abstrakte Fähigkeit. Die Implementierung, auf die die Fähigkeit zurückgreift, wird in normalen ROS-Paketen umgesetzt, wodurch es insbesondere möglich ist, existierende Funktionalität zu einer Roboterfähigkeit zu erweitern. Die Fähigkeiten, die für den SBC entwickelt wurden, sind beispielsweise in thematischen Paketen wie *Manipulation* oder *Perception* gebündelt.

Die Granularität bzw. semantische Eben von Fähigkeiten ist bewusst nicht vorgegeben oder eingeschränkt. Der Fokus einer Fähigkeit liegt, im Vergleich zu den RAL-Schnittstellen, allerdings auf der algorithmischen Umsetzung einer Aktivität, nicht einer konkreten Ansteuerung von Roboterhardware. Für sehr einfache Fähigkeiten oder sehr mächtige *Robot Layers* kann der Koordinator einer Fähigkeit auf dem *Skill Layer* einen einzelnen Aufruf, etwa einer RAL-Schnittstelle, beinhalten. Da jedoch Fähigkeiten auch andere Fähigkeiten aufrufen können, ent-

stehen wiederverwendbare Bausteine mit teils großer Komplexität. Im Folgenden sollen zwei Beispiele für konkrete Fähigkeiten gegeben werden:

Fine-Localize-Object kann genutzt werden, um die Position eines gefunden Objektes durch das Einpassen von geometrischen Primitiven genauer zu bestimmen (Abb 4.14).

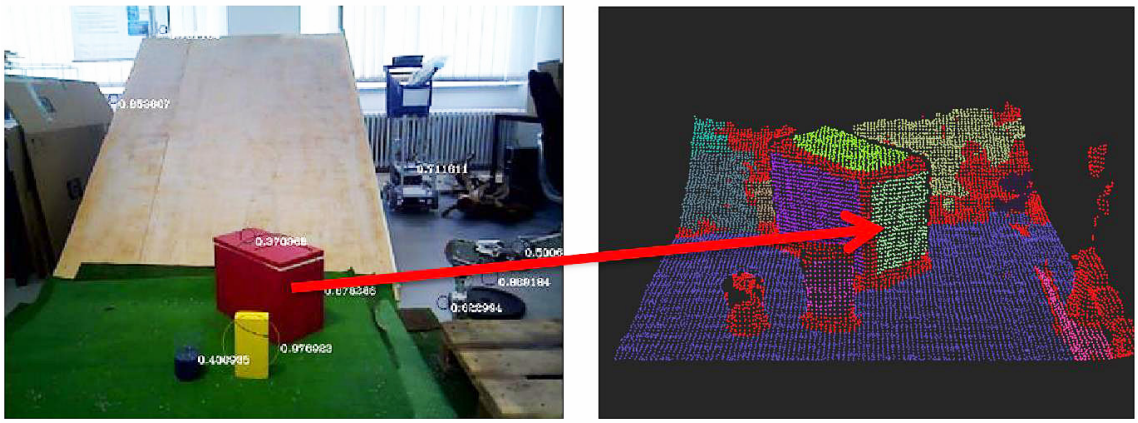


Abbildung 4.14.: Beispiel für die *FineLocalize* Fähigkeit. Links: RGB-Kamerabild mit Konfidenzwerten von Objektdetektionen, basierend auf einfacher Farberkennung. Rechts: 3D Punktwolke mit eingepassten gemoetrischen Körpern für die Lagebestimmung der Objekte.

Diese Fähigkeit ist einfach strukturiert und beinhaltet insbesondere den konkreten Aufruf eines ROS-Services, welcher das eigentliche Fitting durchführt. Allerdings wird durch den Koordinator der Fähigkeit die Nutzung vereinfacht. Der einzige Eingabeparameter ist die Objekt-ID des relevanten Objektes, alle weiteren Daten, etwa die verfügbaren geometrischen Primitiven, die grobe Position des Objektes oder die Sensordaten werden durch die Fähigkeit selbst beschafft und müssen nicht vom Nutzer übergeben oder definiert werden. In diesem Fall ist die Fähigkeit vor allem ein Wrapper der Funktionalität, die Details der Handhabung von Daten oder der genauen Ablauf der Lokalisierung werden vor dem Aufrufenden verborgen.

Die Fähigkeit **Update-Environment-Modell** wird genutzt, um aktiv die Umgebung zu analysieren und so viel Umgebungs- und Objektinformation zu sammeln wie möglich. Der Ablauf der Fähigkeit ist etwas komplexer und zudem stark roboterspezifisch. Für den Laufroboter LAURON wird die Fähigkeit durch die folgende Abfolge realisiert:

- Der Roboter wird angehalten und in eine Ruheposition gebracht. Das Anhalten selbst ist eine weitere Fähigkeit, die wiederum die RAL aufruft, welche etliche roboterspezifischen Schritte durchführt
- Die Umgebungskarte wird durch einen 360-Grad-Laserscan aktualisiert

- Der Sensorkopf wird genutzt, um die Kameras durch den beobachtbaren Bereich (etwa 180 Grad vor dem Roboter) zu bewegen

Dabei gefundene Objekte und Kartendaten werden global im ROS-System bekanntgegeben und stehen damit den weiteren Komponenten und Fähigkeiten zur Verfügung.

In diesem Fall regelt die Fähigkeit vor allem spezifische Abfolgen und nutzt andere Fähigkeiten, wie etwas das Ändern eines Zustandes oder eine Bewegung mit dem Sensorkopf, um eine komplexe Fähigkeit einzusetzen. Andere Roboter, etwa ein Flugroboter oder ein stationärer Sensorpunkt, könnten die Fähigkeit ebenfalls implementieren und auch Teile, etwa einen 360 Grad Laserscann, wiederverwenden, während andere Schritte roboterspezifisch umgesetzt werden.

Duch das Nutzen von Fähigkeiten zum Aufbau von Fähigkeiten werden roboterspezifische Aufrufe weiter abstrahiert, lediglich die letzte Ebene von Fähigkeiten muss konkrete Funktionsaufrufe umsetzen, welche im besten Fall durch die RAL der Roboter auf identische Interfaces abgebildet werden.

4.3.3. Mission Layer

Die **Mission Layer** überwacht die Ausführung der Fähigkeiten und koordiniert die Zuordnung der Fähigkeiten mit anderen Robotern. Sie stellt die Kern-Funktionalitäten für die MRTA zur Verfügung, nutzt dabei jedoch maßgeblich die von den Fähigkeiten zur Verfügung gestellten Modelle und Koordinatoren. Die Funktionen können drei Bereichen zugeordnet werden: *Task Allocation*, *Capability Management* und *Mission Control* (Abbildung 4.15). Daten werden global verwaltet. Die *Mission Control*, *Task Allocation* und die *Capabilities* kommunizieren Daten durch Veröffentlichung auf ROS-Topics, welche interessierte Teilnehmer wiederum abonnieren können. Welche Daten verfügbar sind, hängt vor allem von den verschiedenen *Capabilties* ab, für die Exploration werden etwa verschiedene Umgebungskarten, eine Liste aktueller Explorationsziele sowie erkannte Objekte ausgetauscht.

Das Capability Management verwaltet die für den jeweiligen Roboter verfügbaren Fähigkeiten und steht mit den übrigen Teilnehmern des Teams, also anderen Instanzen des Frameworks, in Kontakt, um diese Information zu teilen. Dazu werden verfügbarer Fähigkeiten von der *Skill Layer* gesammelt und die Fähigkeiten anderer Roboter durch Austausch der Interface-Informationen gesammelt. Aus dem so erstellten *Capability Repository* kann eine Planungskomponente oder der menschliche Nutzer Fähigkeiten wählen und für das Erstellen einer Mission nutzen. Wird das Ausführen einer Fähigkeit über die *Task Allocation* angefragt, ermittelt das *Capability Management* die Kosten basierend auf der Parametrisierung der Aufgaben und kombiniert diese (Details in 4.5.3) um ggf. ein Gebot auf dem Markt abzugeben. Sind benötigte Teile der Fähigkeit nicht vorhanden, wird diese

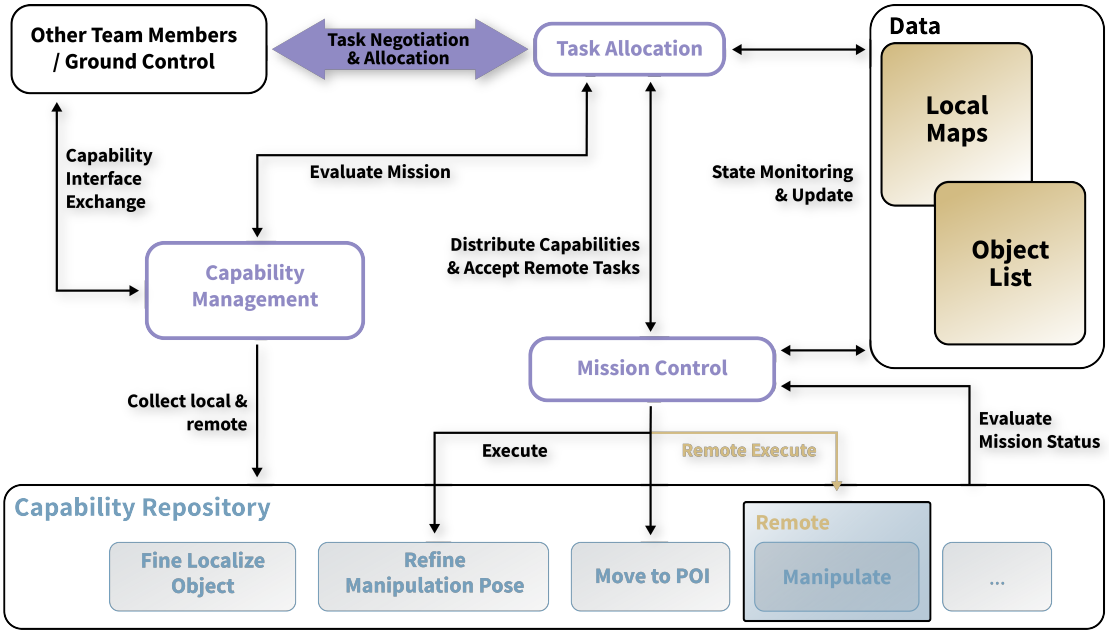


Abbildung 4.15.: Mission Layer des Frameworks. Sie beinhaltet die deliberativen Komponenten, die eine Koordination der Aufgaben mit anderen Team-Teilnehmern ermöglichen und die Fähigkeiten eines Roboters verwalten.

nicht angenommen oder aber durch das *Capability Management* und die *Task Allocation* ermittelt, welche Teile der Mission an einen anderen Roboter ausgelagert werden können.

Die Task Allocation übernimmt die eigentliche Verteilung von Fähigkeiten im Team. Sie gibt bekannt, ob der Roboter generell bereit ist, Fähigkeiten für andere Team-Teilnehmer zu übernehmen und verhandelt mit ihnen darüber, wer den Zuschlag für eine Ausführung erhält. Basierend auf den von *Capability Management* ermittelten Kosten der Ausführung gibt die *Task Allocation* ein Gebot ab, überwacht die anderen Gebote und erteilt Zuschläge. Ebenso kann der Roboter Fähigkeiten zur Vergabe auf dem Markt bereitstellen und nach Geboten der anderen Teilnehmer fragen. Wird ein Task auf diese Weise vergeben, überwacht die *Task Allocation* die Ausführung, um etwa bei Nichterfüllung der Aufgabe eine Neuvergabe zu starten. Der Vergabeprozess wird im Abschnitt 4.5.4 näher beschrieben.

Die Mission Control kontrolliert die Ausführung einer Mission und nutzt die anderen Komponenten zur korrekten Verteilung der Fähigkeiten. Eine Mission wird, ebenso wie die Koordinatoren von Fähigkeiten, als Behavior Tree modelliert, wobei die Blätter des Baumes jeweils Fähigkeiten aus dem *Capability Repository* sind. Um ein Mission auszuführen wird zunächst geprüft, ob alle relevanten Fähigkeiten vorhanden sind und die Kosten für die lokale Ausführung ermittelt. Wenn eine Fähigkeit für eine mögliche Ausführung durch andere Team-Teilnehmer gekennzeichnet ist, werden diese durch die *Task Allocation* zur Ab-

gabe eines Gebots aufgefordert. Sobald alle Fähigkeiten zu Robotern zugeordnet sind, werden die Fähigkeiten durch das *Capability Management* instanziiert, also die abstrakten Fähigkeitsmodelle endgültig auf konkrete Implementierungen abgebildet, wodurch ein verteilter BT entsteht, der nun über alle Teilnehmer hinweg ausgeführt werden kann. Wenn für die entfernte Ausführung von Anfang an *Capabilities* (vgl. 4.4.6) genutzt werden erfolgt die Instanziierung nicht beim Starten des BT, sondern kontinuierlich während der Ausführung. Zudem überwacht die *Mission Control* mittels einer roboterspezifischen Mission, dass notwendige Funktionen, etwa die Überwachung der Batterien, ausgeführt werden und implementiert Fallback-Funktionen, wie etwa ein Idle-Verhalten.

Die Komponenten der *Mission Control* sind vergleichbar mit denen anderer Ansätze. Es gibt zumeist eine aushandelnde Komponente, ein Modul zur Verwaltung von Fähigkeiten und eine Missionssteuerung, die die eigentliche Mission überwacht. Kernproblematik der meisten Ansätze ist die notwendige Abbildung zwischen den jeweiligen Darstellungen, insbesondere das Abbilden der Fähigkeiten auf die Mission selbst. Hier unterscheidet sich diese Arbeit signifikant von anderen bekannten Ansätzen, indem BTs (Abschnitt 4.4) als Modellierung sowohl für die Fähigkeiten als auch für die Mission selbst genutzt werden und damit eine durchgängige Modellierung bieten, ohne die Komplexität und Granularität an einer künstlichen Schnittstelle zu beschränken.

4.4. Behavior Trees als Modell für die Missions- und Fähigkeitsdarstellung

Behavior Trees bieten zahlreiche positive Eigenschaften für die Modellierung von Verhalten: Insbesondere die Möglichkeit, Teile eines Baumes jederzeit zu verschieben und ineinander einzusetzen sowie ihre Reaktivität werden von dieser Arbeit sowohl für die Modellierung der Mission als auch der Fähigkeiten eines Roboters genutzt.

Bis dato bekannte Umsetzungen von Behavior Trees erfüllen jedoch nicht alle Anforderungen, weshalb eine eigene Behavior-Tree-Definition entwickelt und als Bibliothek (`ros_bt_py`) implementiert wurde. Die grundlegende Definition folgt bekannten Ansätzen, wie denen von Marzinotto [49] oder Colledanchise [64], erweitert dieses jedoch um:

- Lang andauernde asynchrone Prozesse
  Um vor allem für Robotik-Anwendungen relevante Funktionen wie Plattformbewegung abbilden zu können.

- Parametrisierung und Datenkanten
  Um eine einfache und modulare Wiederverwendbarkeit von Bäumen zu ermöglichen.

- Utility-Berechnung
  Um die Kosten für die Ausführung einer Fähigkeit zu bestimmen und zu aggregieren.

- Capabilities
  Um die heterogenen Fähigkeiten dynamisch durch BTs abbilden zu können und dabei die anderen Vorteile des Modells auszunutzen.

- Verteilte Ausführung
  Um Missionen durch explizite oder implizite Modellierung transparent auf ein Team zu verteilen.

Im Folgenden wird die formale Definition des vorgeschlagenen Modells gegeben. Das Behavior Tree Modell wurde in [Heppner et al., 2023] veröffentlicht, speziell die *Capabilities* wurden in einer darauf aufbauenden Arbeit [Heppner et al., 2024] zur Veröffentlichung akzeptiert. Das Modell wurde in der Bibliothek `ros_bt_py` als Open-Source-Software[4] sowohl für ROS 1 als auch ROS 2 veröffentlicht.

4.4.1. Allgemeine Behavior Tree Definition

Definition 1 (BT-Knoten)**.** Ein *Knoten*(Node) n im BT hat einen Zustand

$$state(n) \in \mathcal{S} = \{\text{uninitialized}, \text{error}, \text{idle}, \text{succeeded}, \text{failed}, \text{running}, \text{shutdown}\}$$

und einer maximalen Anzahl an Kindern $maxChildren(n) \in [0, \infty]$. Der Zustand des Knotens kann durch eine $update$ Funktion (siehe Definition 10) basierend auf der *Aktion* a gewechselt werden

$$a \in \mathcal{A} = \{\text{setup}, \text{tick}, \text{untick}, \text{reset}, \text{shutdown}\}$$

Abbildung 4.16 zeigt die möglichen Transitionen für jede Aktion. Welche Transition gewählt wird, ist abhängig vom aktuellen Zustand des BT-Knotens und der gewählten Aktion.

Definition 2 (BT-Knoten Klassen)**.** BT-Knoten können in 3 grundlegende *Knoten Klassen* eingeteilt werden: *Flusskontrolle*, *Dekoratoren* und *Blätter*. Für jeden BT-Knoten n gilt:

$$\forall n \in \mathcal{N}_T : nodeClass(n) = \begin{cases} \text{Blatt} & \text{wenn } maxChildren(n) = 0, \\ \text{Dekorator} & \text{wenn } maxChildren(n) = 1, \\ \text{Flusskontrolle} & \text{wenn } maxChildren(n) > 1 \end{cases}$$

[4] `https://github.com/fzi-forschungszentrum-informatik/ros_bt_py`
`https://github.com/fzi-forschungszentrum-informatik/ros2_ros_bt_py`

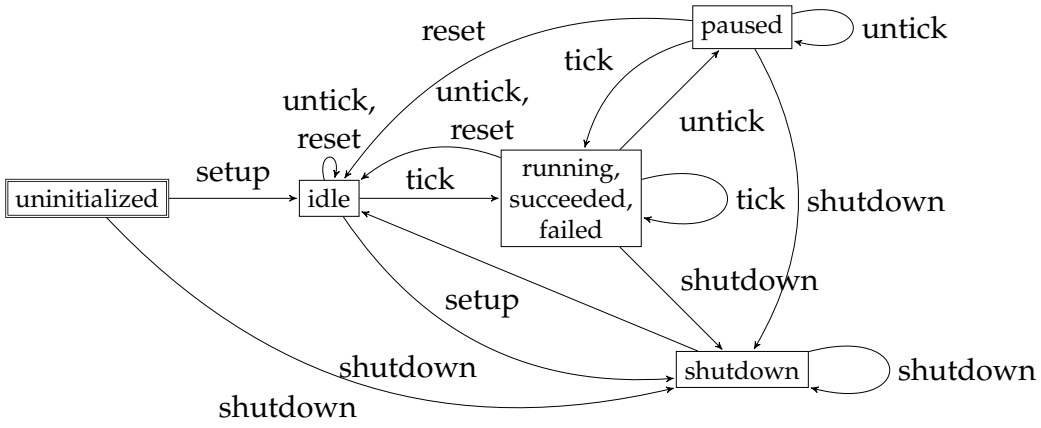


Abbildung 4.16.: Die möglichen Zustandsübergänge eines BT-Knotens. Die Zustände *succeeded*, *failed* und *running* wurden zusammengefasst. Quelle [BH22]

Blätter führen Aktionen aus oder evaluieren Ausdrücke. Dekoratoren kapseln Blatt-Knoten und modifizieren ihre Ergebnisse und Verhalten. Flusskontroll-Knoten beinflussen, wie der *tick* im Baum weitergeleitet wird. Üblicherweise verwendet werden die *sequence, fallback* und *parrallel* Knoten. Marzinotto [49] hat eine detaillierte Definition aller Verhalten für die Flusskontrolle gegeben.

Definition 3 (Behavior Tree)**.** Ein BT (Behavior Tree) ist ein *geordneter Baum* [93, p. 573], welcher als Graph $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ beschrieben wird, dabei ist $\mathcal{N}_T$ die Menge der Knoten im BT, $\mathcal{E}_T$ die Menge der gerichteten Kanten (p, c) mit $p, c \in \mathcal{N}_T$ und $p \neq c$, und $\wr \colon \mathcal{E}_T \to \mathbb{N}$ eine Funktion, die jede *Kante* (p, c) auf seine Position unter den Kindern von p abbildet.

Dafür muss $\wr$ die folgende Bedingung erfüllen:

$$\wr((p, c)) \neq \wr((p, x)) \quad , \forall x \in \mathcal{N}_T \setminus \{p, c\}$$

Zwei Kinder desselben Elternteils dürfen also nicht die gleiche Position einnehmen.

Definition 4 (Wurzel eines BT)**.** Die *Wurzel* (root) eines BT ist ein Knoten, von dem alle anderen Knoten erreichbar sind. Da BT geordnete Bäume sind, gibt es also immer maximal eine *Wurzel*, welche keine weiteren Eltern hat. Das heißt, in jedem BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ gibt es eine *Wurzl* $r \in \mathcal{N}_T$, so dass $(n, r) \notin \mathcal{E}_T, \forall n \in \mathcal{N}_T \setminus \{r\}$. Der Zustand der *Wurzel* $state(r)$ ist gleichbedeutend mit dem Zustand des BT $\mathcal{G}_T$.

Definition 5 (BT Beziehungen)**.** In einem BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$, wird der Knoten $p \in \mathcal{N}_T$ *Elternteil* und der Knoten $c \in \mathcal{N}_T$ *Kind* eines Knotens von p genannt, wenn

diese über eine Kannte miteinander verbunden sind: $(p, c) \in \mathcal{E}_T$. Die Menge aller Kinder von p wird durch die Funktion $children(p, \mathcal{N}_T, \mathcal{E}_T)$ ermittelt.

Ebenso werden zwei Knoten $c_1, c_2 \in \mathcal{N}_T$ mit $c_1 \neq c_2$ *Geschwister* genannt, wenn diese das gleichen *Elternteil* p haben, d.h. wenn $(p, c_1) \in \mathcal{E}_T \wedge (p, c_2) \in \mathcal{E}_T$.

Definition 6 (BT Randbedingungen)**.** Damit ein als $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ gegebener BT als *gültig* gilt, muss dieser einige Randbedingungen zu jeder Zeit gewährleisten. Die erste Randbedingung ist die der $maxChildren$:

$$|children(n)| \leq maxChildren(n) \quad \forall n \in \mathcal{N}_T$$

In den weiteren Definitionen werden zusätzliche Randbedingungen aufgeführt.

Definition 7 (Datengraph)**.** Der *Datengraph* $\mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D)$ enthält die Informationen über die von BT genutzten Daten. Seine Knoten definieren die *Parameter* für BT-Knoten, seine Kanten die *Wirings* genannten Verbindungen zwischen diesen. Abschnitt 4.4.3 definierten den *Data Graph* detaillierter.

Definition 8 (Weltzustand)**.** Um die Interaktion eines gegeben BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ und dem dazugehörigen Datengraph $\mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D)$ mit der realen Welt zu modellieren, wird der *Weltzustand* definiert. Dieser bildet die Sensorwerte der aktuellen Welt auf BT-Knoten-Zustände und Parameter-Werte ab. Dabei ist der *Weltzustand* definiert als $\mathcal{W} \in \Sigma^x, x \in \mathbb{N}$, wobei Σ ein beliebiges *Alphabet* ist welches die Zustände und Parameter abbildet.

Wie genau das Alphabet aufgebaut ist spielt keine Rolle, solange mit den Funktionen $state(n, \mathcal{W})$, $value(p, \mathcal{W})$ der aktuelle Zustand eines Knotens $n \in \mathcal{N}_T$ und der Wert einer Parameters $p \in \mathcal{N}_D$ von einem gegebenen *Weltzustand* $\mathcal{W}$ extrahiert werden kann. In den Definitionen wird der Weltzustand oft weggelassen, um die Lesbarkeit zu erhöhen, dabei sollte jedoch stets bedacht werden, dass dieser, und damit auch die Zustände der Knoten und deren Parameterwerte, sich nur durch die im Folgenden definierte $update$ Funktion ändern können.

Jeder Roboter hat seinen eigenen *Weltzustand*, da dieser die individuellen Sensorwerte, nicht den tatsächlichen Zustand der Welt, abbildet. Wenn zwei Roboter miteinander kommunizieren können, können sie auch einen Teil ihres *Weltzustands* miteinander teilen.

Definition 9 (Nahe Roboter)**.** Gegeben der lokale Weltzustand $\mathcal{W}$ eines BT, kann der Weltzustand von Robotern in der Nähe aus dem lokalen Weltzustand mit der Funktion $nearbyWorlds(\mathcal{W})$ gebildet werden. Dies setzt natürlich voraus, dass die anderen Roboter auch BTs benutzen.

Definition 10 (BT Environment)**.** Das 4-Tupel aus BT, Datengraph, Alphabet und Weltzustand wird als *BT Environment* $Env_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$ bezeichnet. Durch Aufruf der $update$ Funktion kann der Zustand eines Knotens entsprechend der

$\mathcal{N}_S \coloneqq \{n\}$;
while *true* **do**
  `// Füge alle Kinder-Knoten hinzu die bereits in` $\mathcal{N}_S$ `sind`
  $\mathcal{N}_S' \coloneqq \mathcal{N}_S \cup \left(\bigcup_{x \in \mathcal{N}_S} children\,(x, \mathcal{N}_T, \mathcal{E}_T) \right)$;
  if $|\mathcal{N}_S'| = |\mathcal{N}_S|$ **then**
    break;
  end
  $\mathcal{N}_S \coloneqq \mathcal{N}_S'$;
end
$\mathcal{E}_S \coloneqq \{(n_1, n_2) \mid (n_1, n_2) \in \mathcal{E}_T \wedge n_1, n_2 \in \mathcal{N}_S\}$;
`// Da gilt` $\mathcal{N}_S \subseteq \mathcal{N}_T$`, kann die Ordnung` $\wr$ `wiederverwendet werden`
$\wr_S = \wr$;
return $(\mathcal{N}_S, \mathcal{E}_S, \wr_S)$;

Abbildung 4.17.: Algorithmus zu Erstellung eines Subtrees

gewählten Aktion und dem aktuellen Environment bzw. dem darin enthaltenen Weltzustand aktualisiert werden:

$$Env_T{}' = update(n, a, Env_T)$$

wobei $n \in \mathcal{N}_T$ und $a \in \mathcal{A}$ gilt (siehe Definition 1).

Definition 11 (Subtree)**.** Für einen gegebenen BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ kann durch den Algorithmus 4.17 ein *Subtree* (Teilbaum) $\mathcal{G}_S = (\mathcal{N}_S, \mathcal{E}_S, \wr_S)$ mit der Wurzel $n \in \mathcal{N}_T$ erstellt werden. Um das Environment eines Subtrees zu erstellen, muss der Datengraph $\mathcal{G}_D$ des ursprüngliche Environments $Env_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$ ebenfalls gefiltert werden, damit nur die Wirings erhalten bleiben, die zu den im Subtree $\mathcal{G}_S$ verbleibenden Knoten $\mathcal{N}_S$ gehören (siehe Definition 18). Die Ordnungsfunktion $\wr$, das Alphabet Σ und der Weltzustand $\mathcal{W}$ können direkt übernommen werden.

4.4.2. Andauernde asynchrone Prozesse

Standard Behavior Trees kennen vor allem die drei Zustände *succes, failure* und *running*. Wird eine Aktion ausgeführt, die nicht innerhalb eines *ticks* bearbeitet

werden kann, hat der Knoten den Zustand *running*, bis ein Ergebnis ermittelt werden kann oder ein Flusskontroll-Knoten entscheidet, dass der Knoten nicht mehr getickt wird. Gerade in der Robotik ist das unvermittelte Beenden von länger andauernden bzw. asynchronen Prozessen jedoch kritisch, da z.B. ein Roboterarm nicht ohne Weiteres aus jedem Zustand in einen anderen übergehen kann, weshalb dies explizit berücksichtigt wurde.

Der Zustand von Knoten nach der Aktion *untick* hängt davon ab, wie ein Hintergrundprozess gestoppt wurde. Der Prozess kann pausiert werden, der neue Zustand ist dann *paused*, welcher jederzeit mit einem *tick* fortgesetzt werden kann. Wenn der Prozess komplett gestoppt wird, ist der neue Zustand *idle* und ein erneuter *tick* führt zu einem *restart*. Damit ist das System ähnlich zu dem von Klöckner [67], umgeht jedoch einige extra Schritte bei der Aktivierung. Im Gegensatz zu [67] geht diese Definition zudem davon aus, dass *idle*, *succeeded* und *failure* Ruhezustände sind, d.h. der Knoten muss alle Hintergrundprozesse beenden, bevor einer dieser Zustände aktiviert werden darf.

Definition 12 (Hintergrundprozesse)**.** Ein Knoten $n \in \mathcal{N}_T$ darf Hintergrundprozesse nur genau dann ausführen, wenn $state(n, \mathcal{W}) = \text{running}$. Beim Übergang zu irgendeinem anderen Zustand **muss** der Hintergrundprozess gestoppt werden. Das pausieren im *Paused*-Zustand zählt dabei natürlich ebenfalls als gestoppt.

4.4.3. Parametrisierung und Datengraph

Eine Schwachstelle von klassischen Behavior Trees ist die Dopplung von großen Teilen der Verhaltensbausteine, die sich nur in kleinen Details unterscheiden. Das verwenden einer Parametrisierung von Knoten erleichtert die Wiederbenutzung von Knoten. Aufbauend auf den Ideen in [68] werden daher für jeden Knoten 3 stark getypte und gemeinsam als *Parameter* bezeichnete Schnittstellen definiert: Options, Inputs und Ouptuts.

Definition 13 (Parameter)**.** Sei $\mathbf{T}$ die Menge aller nicht leeren Mengen. Bei einem BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$, ist ein *Parameter* ein Tripel

$$
\begin{aligned}
p = (n, k, t) \text{ mit Knoten } & n \in \mathcal{N}_T \\
& \textit{Art } k \in \{\text{option}, \text{input}, \text{output}\} \\
& \textit{Typ } t \in \mathbf{T} \cup \{\mathit{optionRef}\,(p_{\text{ref}}) \mid p_{\text{ref}} = (n, \text{option}, \mathbf{T})\}
\end{aligned}
$$

Ein Parameter $p = (n, k, t)$ wird je nach seiner *Art* k als *Option*, *Input* oder *Output* bezeichnet. Die Menge aller Parameter sei $\mathbb{P}$.

Der Typ t eines Parameters kann entweder eine Menge aus $\mathbf{T}$ sein, die die zulässigen Werte enthält, die der Parameter annehmen kann (wie $\mathbb{N}$ oder die Menge aller gültigen Python `str`-Objekte), oder eine *Referenz* auf eine Option, geschrieben $optionRef\,(p_{\text{ref}})$.

Für die Referenz muss p_{ref} die Form $(n, \text{option}, \mathbf{T})$ haben, wobei $\mathbf{T}$ wiederum die Menge aller Mengen ist, die als Werte für den Typ t verwendet werden können. Da $\mathbf{T}$ nur Mengen enthält, schließt es insbesondere Referenzen nicht mit ein, so dass zyklische Referenzen verhindert werden.

Definition 14 (Parameter-Werte)**.** Gegeben ein BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ und Welt $\mathcal{W}$, hat jeder Parameter einen *Wert*:

$$\forall p = (n, k, t) \in \mathbb{P}\colon\ value(p, \mathcal{W}) \in resolve(t, \mathcal{W}) \cup \{\texttt{None}\}$$

$$resolve\,(t, \mathcal{W}) = \begin{cases} t & \text{if } t \in \mathbf{T} \\ value\,(p_{\text{ref}}, \mathcal{W}) & \text{if } t = optionRef\,(p_{\text{ref}}) \wedge\ p_{\text{ref}} = (n, \text{option}, \mathbf{T}) \\ & \qquad \wedge\, value\,(p_{\text{ref}}, \mathcal{W}) \neq \texttt{None} \\ \emptyset & \text{else} \end{cases}$$

Initial gilt für Inputs und Outputs: $value(p, \mathcal{W}) = \texttt{None}$.

Definition 15 (Initiale Optionswerte)**.** Die Werte von Optionen sind statisch, d.h. sie erhalten lediglich einmal während der Initialisierung ihre Werte, welche separat definiert werden und während der Laufzeit nicht mehr geändert werden. Um sicherzustellen, dass die Typen aller Parameter bekannt und festgelegt sind, gibt es in einem BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ mit Welt $\mathcal{W}$ eine statische Menge von *option values*, so dass

$$\forall o = (n, \text{option}, t) \in \mathbb{P}\colon\ value(o, \mathcal{W}) = optionValue(o, \mathcal{W})$$

In der Implementierung wird dies durch ein Wörterbuch realisiert, welches im Konstruktor an den Knoten übergeben wird.

Der Vorteil der Optionswerte lässt sich leicht an einem Beispiel demonstrieren. In Abbildung 4.18 ist ein sehr einfacher BT dargestellt, mit welchem nach farbigen Bällen gesucht und ein roter oder grüner Ball aufgesammelt werden sollen. Es ist ersichtlich, dass die Blattknoten sehr wahrscheinlich einen großen Teil ihrer Funktionalität teilen könnten, da sich die Detektion und das Aufsammeln lediglich in Bezug auf die Farbe des Balles unterscheiden.

Abbildung 4.19 zeigt den gleichen Baum unter Verwendung von Optionen. Durch die Parametrisierung können die Knoten zu *Detect Ball* und *Pick Up Ball* zusammengefasst und wieder verwendet werden.

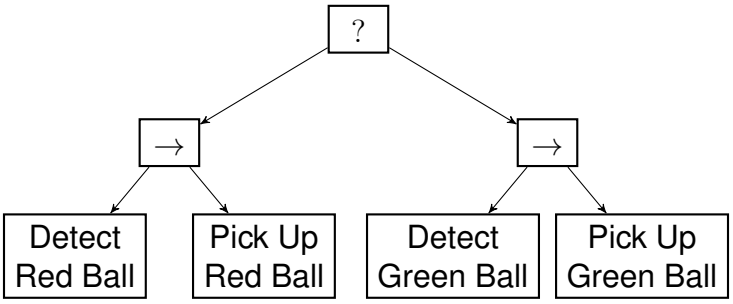


Abbildung 4.18.: Beispiel eines Behavior Trees, der entweder einen roten oder grünen Ball detektiert und diesen in der Folge aufhebt. In traditionellen BTs muss jede Bedingung und Aktion als individueller Blattknoten umgesetzt werden, was zu unnötiger Code-Dopplung führt. Die Reihenfolge der Ausführung ist durch die Anordnung fixiert, die Auswertung von Zweigen des Baumes hängt nur von dem Rückgabewert der einzelnen Knoten ab. Quelle [BH22]

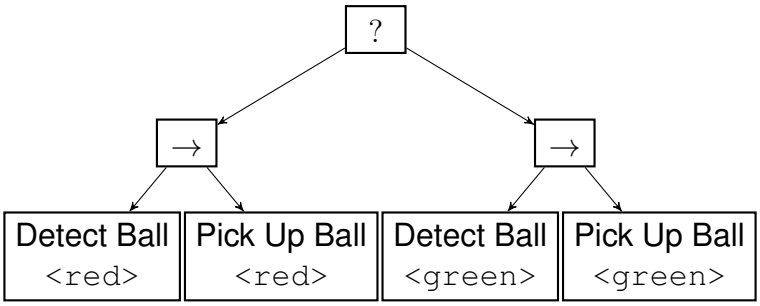


Abbildung 4.19.: Der BT aus Abbildung 4.18, erweitert um das Konzept der parametrisierten Knoten. Durch das Parametrisieren können Dopplungen der Knoten reduziert werden. Quelle [BH22]

Definition 16 (Datengraph)**.** In einem BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$ können die Ein- und Ausgänge der Knoten durch Kanten in einem gerichteten *Datengraph* $\mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D)$ verbunden werden, wobei

$$\mathcal{N}_D \subseteq \mathbb{P}$$
$$\mathcal{E}_D = \{(p_s = (n_s, k_s, t_s), p_t = (n_t, k_t, k_t)) \mid (\, p_s, p_t \in \mathcal{N}_D \wedge k_s = \text{output} \wedge k_t = \text{input} \wedge \mathit{resolve}(n_s, t_s, \mathcal{W}) = \mathit{resolve}(n_t, t_t, \mathcal{W})\,)\}$$

Definition 17 (Datenquelle & Ziel)**.** Eine Kante $(p_s, p_t) \in \mathcal{E}_D$ wird als *Data Wiring* bezeichnet. Das erste Element p_s wird seine *Source (Quelle)* und das zweite, p_t, sein *Target (Ziel)* genannt.

Wann immer der Wert der *Source* ($\mathit{value}(p_s, \mathcal{W})$) geändert wird, wird der Wert des *Target* ($\mathit{value}(p_t, \mathcal{W})$) aktualisiert. Mehrere *Data Wirings* können dabei dieselbe *Source* oder dasselbe *Target* verwenden.

Das Beispiel aus Abbildung 4.19 kann durch diese Definitionen weiter vereinfacht werden (Abbildung 4.20). Durch die Datenkanten kann die doppelte Instanz des *Pick Up Ball* Knotens eingespart werden, der *Detect Ball* Knoten wird trotzdem zwei Mal im Baum genutzt, um den Rückgabewert der Erkennung für die Flusskontrolle weiter zu nutzen.

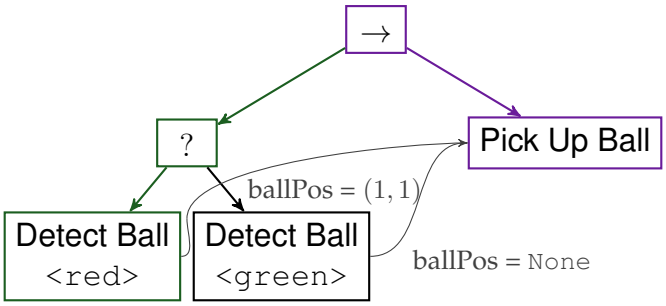


Abbildung 4.20.: Behavior Tree mit Datenkanten. Durch Übergabe der Daten zwischen den Knoten wird *Pick Up Ball* nur einmal benötigt. Im Beispiel wurde ein *roter* Ball entdeckt (Datenkannte *ballPos* mit dem Wert (1,1)) welcher durch *Pick Up Ball* aufgehoben wird. Quelle [BH22]

Mit diesen zusätzlichen Definitionen kann die Definition des Daten Graphs eines Subtrees (Siehe Definition 11) erweitert werden:

Definition 18 (Subtree Data Graph)**.** Gegeben ein BT $\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr)$, sein Datendiagramm $\mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D)$ und einer seiner Teilbäume $\mathcal{G}_S = (\mathcal{N}_S, \mathcal{E}_S, \wr_S)$ ist das Datendiagramm des Teilbaums $\mathcal{G}_D{}'$ wie folgt definiert:

$$\mathcal{N}_D{}' = \{(n,k,t) \mid (n,k,t) \in \mathcal{N}_D \wedge n \in \mathcal{N}_S\}$$
$$\mathcal{E}_D{}' = \{(p_s,p_t) \mid (p_s,p_t) \in \mathcal{E}_D \wedge p_s, p_t \in \mathcal{N}_D{}'\}$$

Anders als RAFCON erlaubt der BT Data Graph nicht nur Verbindungen zwischen Parametern, deren Knoten eine Geschwister- oder Eltern-Kind-Beziehung haben, sondern zwischen beliebigen Eingängen und Ausgängen im Baum. Dadurch wird vermieden, dass ein Teil der Daten mehrfach durch nicht miteinander verbundene Knoten weitergeleitet werden muss, um sie dort verfügbar zu machen, wo sie benötigt werden.

4.4.4. Utility Berechnung

Wenn BT für das Treffen von Entscheidungen genutzt werden, wie es in in dieser Arbeit der Fall ist, dann ist es sinnvoll oder sogar notwendig, den Nutzen bzw. die Kosten einer Aktivität ermitteln zu können. Ein solcher Wert wird als *Utility*-Wert bezeichnet und kann als Basis für eine Entscheidung zwischen zwei

gleichberechtigten Aktionen oder aber zur Bestimmung ihrer Reihenfolge genutzt werden. In dem bisherigen Beispiel des Ball-Aufhebens ist die Reihenfolge des Aufhebens durch die Anordnung der Knoten vorgegeben (vgl. Abbildung 4.18). Wären in diesem Beispiel sowohl ein grüner als auch ein roter Ball vorhanden, würde lediglich der rote Ball eingesammelt werden, selbst wenn es eventuell sinnvoller wäre, den grünen Ball zu sammeln. Würde statt des Fallbacks ein Knoten verwendet, der zuerst den Teil des Baumes evaluiert, der eine bessere Utility aufweist, könnte diese Entscheidung dynamisch angepasst werden, etwa wenn bereits eine der Farben gesammelt wurde und daher die jeweils andere einen höheren Stellenwert hat.

Das Konzept eines Utility-Wertes wird wichtiger, aber auch komplexer, wenn die Aktivitäten komplexer werden und zudem stärker von dem Weltzustand abhängen, wie es bei Robotik-Anwendungen nahezu immer der Fall ist. Abschnitt 4.5.3 geht genauer auf die Frage ein, wie wir einen solchen Utility Wert bestimmen können.

Ein wichtiges Konzept der BTs, welches diese Arbeit für Fähigkeiten ausnutzen will, ist der modulare Aufbau. Daher ist es erforderlich, neben der Möglichkeit der Utility Berechnung für einzelne Knoten auch Kombinationsvorschriften zu definieren, damit eine zusammengesetzte Fähigkeit, also ein kompletter BT, wieder ihre/seine Utility berechnen und dadurch für die Missionsentscheidung nutzen kann. Dieses Konzept wird zudem noch wichtiger, wenn unterschiedliche Roboter entscheiden sollen, wer eine, ggf. unterschiedlich implementierte, Fähigkeit einsetzen soll.

Definition 19 (Utility-Berechnung)**.** In einer BT Umgebung Env_T kann für jeden Knoten abhängig vom Weltzustand $\mathcal{W}$ ein *Utility Value* $utility(n, \mathcal{W})$ berechnet werden. Dieser Wert stellt die geschätzten *Kosten* (d.h. kleiner ist besser) der Ausführung des Knotens unter Verwendung des 4-Tupels $(c^s_{\text{min}}, c^s_{\text{max}}, c^f_{\text{min}}, c^f_{\text{max}}) \in \mathcal{U}$ dar, wobei $\mathcal{U} = \mathbb{R} \cup \{✗, ?\}$. In diesem Tupel bezeichnen c^s_{min} und c^s_{max} die minimalen und maximalen Kosten für den Erfolg des Knotens und c^f_{min} und c^f_{max} die minimalen und maximalen Kosten für seinen Fehlschlag. Ein Kostenwert von ✗ in einem der vier Werte bedeutet, dass der Knoten mit dem gegebenen Weltzustand $\mathcal{W}$ überhaupt nicht ausgeführt werden kann und impliziert somit, dass **alle** Werte ✗ sein müssen. Ein Kostenwert von $?$ bezeichnet den Fall, in dem ein Knoten ausgeführt werden kann, aber die Schätzung der Kosten für diesen Fall nicht möglich oder definiert sind.

Definition 20 (Addition von Utility-Werten)**.** Additionen der Ultility Werte $\mathcal{U} = \mathbb{R} \cup \{✗, ?\}$ werden wie folgt definiert:

$$\forall a, b \in \mathbb{R} \cup [✗, ?] : a + b = \begin{cases} a + b & \text{if} \quad a \in \mathbb{R} \wedge b \in \mathbb{R} \\ ✗ & \text{if} \quad a = ✗ \vee b = ✗ \\ ? & \text{else} \end{cases}$$

Durch die Definition von Utility Werten (19) und der zugehörigen Rechenvorschrift (20) ist es möglich, die BT Struktur zu nutzen, um einen Utility-Wert für einen ganzen Baum zu ermitteln.

Definition 21 (Utility-Wert eines BT)**.** In einer BT-Umgebung $Env_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$ wird der Utlity-Wert des Wurzelknotens $utility(r, \mathcal{W})$ als Utility-Wert des BT bezeichnet.

Abbildung 4.21 und die dazugehörigen Tabellen 4.1 für das Fallback-Beispiel sowie 4.2 für das Parallel-Beispiel zeigen die Berechnung des Ultility-Werts für einen Gesamtbaums an einfachen Beispielen.

Abbildung 4.21.: BTs für die Berechnung der Werte in den Tabellen 4.1 und 4.2. Der Parallel-Knoten gibt *success* zurück, sobald eines seiner Kinder *success* zurückmeldet. Quelle [BH22]

Für den Fallback ist die Berechnung der Kosten für den Fehlerfall einfach, da der Baum nur dann fehlschlägt, wenn alle Kinder ebenfalls fehlschlagen. Daher sind die minimalen Kosten (c^f_{min}) und maximalen Kosten (c^f_{max}) die Summe aller c^f_{min} und c^f_{max} der Kind-Knoten (Tabelle 4.1, Fall 5). Für den Erfolg hingegen können bis zu 3 Kinder fehlschlagen, bevor der Erfolg eines Kindes zum Erfolg des ganzen Baumes führt (Tabelle 4.1, Fall 1-4). Die Kosten für die jeweiligen Kindknoten müssen für jeden Fall aufaddiert werden ($utility(n_i, \mathcal{W})$), aus den Gesamtergebnissen können dann die minimalen (Fall 1) und maximalen (Fall 4) Kosten ermittelt werden. Für einen *Sequenz*-Knoten ist das Verhalten exakt invers zum Fallback. Nur wenn alle Kinder Erfolg haben, hat der Gesamtbaum Erfolg, für den Misserfolg reicht ein Fehlschlag nach bis zu 3 erfolgreichen Knoten.

Im zweiten Beispiel werden 2 Kindknoten parallel ausgeführt. Der Baum ist erfolgreich, wenn eines der Kinder erfolgreich ist. Wie im vorherigem Beispiel sind die Kosten für einen Fehlschlag die Summe der Kosten der fehlgeschlagenen Kinder, da nur dann ein Fehlschlag erreicht wird, wenn alle Kinder ebenfalls fehlschlagen (Tabelle 4.2, Fall 6). Für den erfolgreichen Fall gibt es hingegen eine zusätzliche Besonderheit: Nicht nur das Erzielen eine Erfolgs für beide (Fall 1) oder einen der Kindknoten bei gleichzeitigem Fehlschlag des anderen (Fall 2 und 5) müssen berücksichtigt werden, sondern auch die Möglichkeit, dass ein Kindknoten noch aktiv (*running*) ist (Fall 3 und 4). Aus allem möglichen Ergebnissen werden nun abermals ein Minimum und Maximum bestimmt.

Case	1	2	3	4	5
$state(n_1, \mathcal{W})$	succeeded	failed	failed	failed	failed
$(c^1_{\text{min}}, c^1_{\text{max}})$	$(1, 10)$	$(2, 5)$	$(2, 5)$	$(2, 5)$	$(2, 5)$
$state(n_s, \mathcal{W})$	idle	succeeded	failed	failed	failed
$(c^2_{\text{min}}, c^2_{\text{max}})$	—	$(1, 10)$	$(2, 5)$	$(2, 5)$	$(2, 5)$
$state(n_3, \mathcal{W})$	idle	idle	succeeded	failed	failed
$(c^3_{\text{min}}, c^3_{\text{max}})$	—	—	$(1, 10)$	$(2, 5)$	$(2, 5)$
$state(n_4, \mathcal{W})$	idle	idle	idle	succeeded	failed
$(c^4_{\text{min}}, c^4_{\text{max}})$	—	—	—	$(1, 10)$	$(2, 5)$
$state(n_f, \mathcal{W})$	succeeded	succeeded	succeeded	succeeded	failed
$(c^F_{\text{min}}, c^F_{\text{max}})$	$(\mathbf{1}, 10)$	$(3, 15)$	$(5, 20)$	$(7, \mathbf{25})$	$(\mathbf{8}, \mathbf{20})$

$$utility(n_f, \mathcal{W}) = (c^s_{\text{min}} = \mathbf{1}, c^s_{\text{max}} = \mathbf{25}, c^f_{\text{min}} = \mathbf{8}, c^f_{\text{max}} = \mathbf{20})$$

Tabelle 4.1.: Beispiel für die Brechnung der Utility für einen Fallback-Knoten n_F mit vier Kindknoten $n_1, \ldots, n_4$. $c^i_{\{\text{min,max}\}}$ ist die Kurzform für den Utility-Wert $utility(n_i, \mathcal{W})$ entsprechend des aktuellen Zustandes $state(n_i, \mathcal{W})$. Die Spalten geben die Entwicklung der Stati an. Es wird angenommen, dass $\forall i \in \{1, 2, 3, 4\}: \; utility(n_i, \mathcal{W}) = (c^s_{\text{min}} = 1, c^s_{\text{max}} = 10, c^f_{\text{min}} = 2, c^f_{\text{max}} = 5)$. Quelle [BH22]

4.4.5. Shoving und Slots - Verteilte Ausführung von BTs

Mit den bisher eingeführten Definitionen können komplexe Missionen für einzelne Roboter definiert und ausgeführt werden, wodurch asynchrone Ausführung, Parametrisierung und Utility-Werte eine große Flexibilität in der Ausführung erlauben, während die Formalismen des BT zu einem les- und wartbarem Verhaltensbaum führen. Durch Subtrees können Funktionen in einem wiederverwendbaren Format definiert und schnell wieder verwendet und kombiniert werden. Diese Flexibilität soll weiterhin auch nutzbar gemacht werden, um die Ausführung auch über mehrere Roboter hinweg zu ermöglichen. Eine frühe Design-Entscheidung war es hierbei, möglichst einen Ansatz zu verfolgen, bei dem auch die Verteilung über die Roboter hinweg mit dem Formalismus der BTs erreicht werden können, ohne dass ein Modellwechseln erfolgen muss. Die erste Entwicklung dafür sind die sogenannten *Shovables*, welche genutzt werden können, um einen subtree transparent auf andere Roboter auslagern zu können.

Definition 22 (Shovable)**.** Der *Shovable*-Knoten ist ein Dekorator, der bei einem Tick, das Subtree-Environment Env_S, von seinem einzigen Kind c ausgehend, aus dem Environment Env_T extrahiert und an die bestmögliche Ausführungsumgebung auslagert. Er verwendet die Weltzustände aller in der Nähe befindlicher Roboter, $nearbyWorlds(\mathcal{W})$, um mittels der Utility-Berechnung festzustellen, ob

Case	1	2	3	4	5	6
$state(n_1, \mathcal{W})$	succeeded	succeeded	succeeded	running	failed	failed
$(c^1_{\text{min}}, c^1_{\text{max}})$	$(1, 10)$	$(1, 10)$	$(1, 10)$	—	$(2, 5)$	$(2, 5)$
$state(n_s, \mathcal{W})$	succeeded	failed	running	succeeded	succeeded	failed
$(c^2_{\text{min}}, c^2_{\text{max}})$	$(1, 10)$	$(2, 5)$	—	$(1, 10)$	$(1, 10)$	$(2, 5)$
$state(n_f, \mathcal{W})$	succeeded	succeeded	succeeded	succeeded	succeeded	failed
$(c^P_{\text{min}}, c^P_{\text{max}})$	$(2, \mathbf{20})$	$(3, 15)$	$(\mathbf{1}, 10)$	$(\mathbf{1}, 10)$	$(3, 15)$	$(\mathbf{4}, \mathbf{10})$

$$utility(n_f, \mathcal{W}) = (c^s_{\text{min}} = \mathbf{1}, c^s_{\text{max}} = \mathbf{20}, c^f_{\text{min}} = \mathbf{4}, c^f_{\text{max}} = \mathbf{10})$$

Tabelle 4.2.: Besipiel für die Berechnung der Utility-Werte for einen parallelen Knoten n_P mit zwei Kindknoten n_1, n_2. $c^i_{\{\text{min,max}\}}$ ist die Kurzform für Utility-Wert $utility(n_i, \mathcal{W})$ entsprechend des aktuellen Zustandes $state(n_i, \mathcal{W})$. Es wird angenommen, dass $\forall i \in \{1, 2\}$: $utility(n_i, \mathcal{W}) = (c^s_{\text{min}} = 1, c^s_{\text{max}} = 10, c^f_{\text{min}} = 2, c^f_{\text{max}} = 5)$. Quelle [BH22]

der Teilbaum lokal oder ausgelagert ausgeführt werden soll, und schickt ihn im letzteren Fall an einen anderen Roboter zur Ausführung.

Definition 23 (Öffentliche Inputs und Outputs)**.** Gegeben ein BT Environment $Env_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$ und ein Subtree $\mathcal{G}_S = (\mathcal{N}_S, \mathcal{E}_S, \wr_S)$, dann sind die *Public Inputs* und *Outputs* des Subtrees $\mathbb{P}_{iop}$ diejenigen Knoten des Datengraphs, die Teil eines Data Wirings von einem Knoten, der Teil des Unterbaums ist, zu einem Knoten, der nicht Teil des Unterbaums ist:

$$\{(n, k, t) \in \mathbb{P}_{iop} \mid n \in \mathcal{N}_S \wedge ((((n, k, t), (n', k', t')) \in \mathcal{E}_D \wedge n' \notin \mathcal{N}_S) \\ \vee (((n', k', t'), (n, k, t)) \in \mathcal{E}_D \wedge n' \notin \mathcal{N}_S))\}$$

Definition 24 (Slot)**.** Ein *Slot* s ist ein Knoten, der ein entferntes Subtree-Environment Env_S von einem *Shovable* Dekorator empfängt. Beim Empfang eines Subtree Environments werden die Werte ihrer Eingaben aus dem Subtree World State $\mathcal{W}'$ extrahiert und in den World State $\mathcal{W}$ aus dem Environment $Env_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$ des *Slots* eingetragen. Die Funktion $update$ für einen *Slot* geht dann so vor, als ob der Subtree Teil des übergeordneten BT des *Slot* wäre.

Sobald der Subtree in einen anderen Zustand als *running* eintritt, wird das Environment des Subtrees zurück an den *Shovable* Dekorator geschickt, von dem er empfangen wurde. Das zurückgeschickte Environment enthält den aktualisierten Weltzustand, der die Knotenzustände und Parameterwerte des Unterbaums beinhaltet. Alle Informationen über das Environment des Teilbaums werden aus dem Weltzustand $\mathcal{W}$ des *Slot* entfernt, wenn $update(s, a, Env_T)$ das nächste Mal nach Beendigung der Ausführung des Subtrees aufgerufen wird.

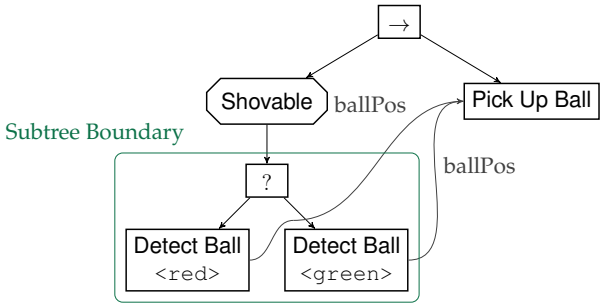


Abbildung 4.22.: Behavior Tree mit *Shovable*-Dekorator. Der BT aus Abb. 4.20, erweitert durch den *Shovable*-Dekorator. Der komplette Subtree, bestehend aus dem *Fallback* und den zwei *Detect Ball* Knoten, wird durch den *Shovable* Dekorator auf ein Entferntes System übertragen, dort ausgeführt und das Ergebniss wieder in den lokalen Baum integriert. Datenkannten können über die Grenzen des Subtrees hinweg gnutzt werden. Quelle [BH22]

Durch die Definition von *Shovables* und *Slots* (Definitionen 22, 24) ist es möglich, einzelne Teile des Baumes auf einem entfernten System auszuführen und dadurch effektiv in einem Team zu verteilen. Durch die Definition von *Utility*-Werten (Definition 19) kann die Zuordnung von *Subtrees* zu einem ausführenden System erfolgen, dabei wird durch das lokale Auswerten der *calculate-utility* auch berücksichtigt, dass die Systeme unterschiedliche Kosten für die Ausführung aufweisen können. Problematisch an dieser Darstellungen ist jedoch, dass die Zuordnung während der Instanziierung eines Baumes, also einmalig, erfolgt und durch die Notwendigkeit eines *Shovable*-Dekorators außerdem eine händische Festlegung ausführbarer Bestandteile des Baumes während der Erstellung erfordert. Beides behindert die dynamische Verteilung von Aufgaben, insbesondere, wenn neue Systeme zum Team hinzustoßen oder es verlassen. Um auch die dynamische Verteilung zu ermöglichen, wurde das Konzept daher noch einmal um *Capabilities* erweitert.

4.4.6. Capabilities

Capabilities sind spezielle Knoten in dem BT, welche das initial dargestellte Konzept von gekapselten Funktionen als semantische Einheit (vgl. Abschnitt 4.1) umsetzen und dabei die Vorteile der modularen BT-Definition ausnutzen. In der bisher vorgestellten Definition wird ein BT offline erstellt und kann während der Ausführung selbst nicht mehr verändert werden. Durch die *Shovables* kann zwar der Ort der Ausführung flexible bestimmt werden, der gesamte BT bleibt dabei jedoch unverändert. Damit aber heterogene Roboter eine Fähigkeit auf unterschiedliche Art und Weise realisieren können, um ihre individuellen Voraussetzungen zu erfüllen, muss der BT auch eine dynamische Komponente aufweisen. Ein *Shovable* erfordert darüber hinaus die explizite Modellierung einer Verteilung

im Team, wünschenswert wäre hingegen, die Fähigkeiten komplett nach aktuellem Bedarf zu verteilen und diese Entscheidung möglichst transparent treffen zu können. Um das komplette Konzept umzusetzen, wurden daher die *Capabilites* als explizite Knoten eingeführt, die einige Erweiterungen der bisherigen Definitionen erfordern.

Definition 25 (Capabilities)**.** Eine Capability (Fähigkeit) c ist ein Knoten im BT, der eine abstrakte Darstellung von zusammengesetzten Handlungen mit einer semantischen Bedeutung ist. Die Menge aller Capabilities in einem BT sei $\mathcal{C}$.

$$\forall c \in \mathcal{C} : c \in \mathcal{N}_T$$

Jede Capability ist ein *Blatt-Knoten*, es gilt also:

$$\forall c \in \mathcal{C} : maxChildren(c) = 0$$
$$\forall c \in \mathcal{C} : nodeClass(c) = \text{Leaf}$$

Sie verfügen zudem über einen Namen und eine Beschreibung:

$$\forall c \in \mathcal{C} : name(c) \in \text{String}$$
$$\forall c \in \mathcal{C} : description(c) \in \text{String}$$

Im Gegensatz zu anderen Blatt-Knoten implementiert die Capability keine eigene Funktionalität, sondern koordiniert die Zuordnung einer *Capability-Implementierung* zur Capability und stellt deren Einbindung in den BT sicher. Dadurch nimmt die *Capability* die Rolle des Fähigkeit-Modells ein, welches beschreibt, wie die Funktionalität zu nutzen ist, bzw. welche Schnittstellen existieren. Ohne eine zugeordnete Implementierung handelt es sich bei einer Capability daher um eine abstrakte Fähigkeit.

Definition 26 (Parameter für Capabilities)**.** Capabilities haben, wie alle anderen Knoten auch, Parameter, welche allerdings nicht durch die Capability selbst verarbeitet, sondern an ihre jeweilige Implementierung weitergereicht bzw. von dieser empfangen werden. Der Definition eines Parameters (siehe Def.13) folgend können die inputs und outputs der Capability über das Parameter Tripel $p = (n, k, t)$ bestehend aus dem Knoten n, Art k und Typ t bestimmt werden:

$$inputs : \mathcal{C} \rightarrow \mathcal{P}(\mathbb{P})$$
$$inputs(c) = \{p = (n, k, t) \mid p \in \mathbb{P} \wedge n = c \wedge k = \text{input}\}$$

$$outputs : \mathcal{C} \rightarrow \mathcal{P}(\mathbb{P})$$
$$outputs(c) = \{p = (n, k, t) \mid p \in \mathbb{P} \wedge n = c \wedge k = \text{output}\}$$

$\mathbb{P}$ bezeichnet die Menge aller Parameter, $\mathcal{P}(\mathbb{P})$ entsprechend alle möglichen Kombinationen. Capabilities können im Gegensatz zu anderen Knoten jedoch keine

Optionen verwenden, da dies mit der dynamischen Bindung der Implementierung nicht ohne Weiteres kompatibel ist:

$$\forall c \in \mathcal{C} : \neg\exists p \in \mathbb{P} : p = (n, k, t) \mid n = c \land k = \text{option}$$

Da die Capability selbst keine Logik implementiert, bietet sie folglich auch keine Logik zur Nutzung der Optionswerte. Da Capability-Implementierungen dynamisch eingebunden werden und aus einem ganzen BT bestehen, wäre zum einen eine implementierungsspezifische Abbildung der statischen Optionswerte auf einzelne Knoten innerhalb der Implementierung BT erforderlich, zum anderen eine Abbildung der statischen Optionswerte auf eine variable Menge an implementierungsspezifischen Optionen. Während dies grundsätzlich natürlich möglich ist, wurde zugunsten der einfacheren Abbildung zunächst darauf verzichtet.

Definition 27 (Vergleich von Capabilities)**.** Zwei Capabilities c_1 und c_2 werden als gleichwertig betrachtet, wenn ihre Namen und ihr Interface, also die Inputs und Outputs, identisch sind:

$$\begin{aligned}\forall c_1, c_2 \in \mathcal{C} : c_1 = c_2 \iff & \mathit{name}(c_1) = \mathit{name}(c_2) \\ & \land \mathit{inputs}(c_1) = \mathit{inputs}(c_2) \\ & \land \mathit{outputs}(c_1) = \mathit{outputs}(c_2)\end{aligned}$$

Definition 28 (Capability Implementierung)**.** Die *Capability* ist zunächst nur eine abstrakte Fähigkeit, also eine Kapselung der semantischen Bedeutung mit definiertem Interface. Erst durch die Zuordnung einer Implementierung zu einer Capability wird diese zu einer konkreten Fähigkeit, also ausführbar. Die Implementierung einer Capability i ist ein kompletter BT, besteht also aus Aktionen, Bedingungen, Flusskontroll-Knoten und ggf. auch weiteren Capabilities sowie den dazugehörigen Datenkanten. Die Implementierung ist roboterspezifisch und berücksichtigt daher auch den Weltzustand $\mathcal{W} \in \Sigma^x; x \in \mathbb{N}$:

$$i = (\mathcal{W}, \mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr), \mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D))$$

Die Implementierung i erfüllt dadurch die Rolle des *Koordinators* einer Fähigkeit. Die Definition ist quasi identisch zu der des BT Environments (Def 10), da es sich um einen kompletten Behavior Tree inklusive Datenkanten handelt.

Sobald eine Capability innerhalb eines BT *getickt* wird, leitet diese den tick an ihrer Implementierung weiter. Der Umstand, dass die Implementierung einer Fähigkeit erst zur Laufzeit gewählt wird, wird als *dynamische Bindung* bezeichnet. Da für diese Bindung auch eine Implementierung auf einem anderen Roboter genutzt werden kann, funktionieren Capability-Implementierungen wie implizite *shovables*. Konkret wird die Ausführung durch *remote capability slot* (section 4.4.8) und *IO Bridges* (section 4.4.7) für die Datenübermittlung ermöglicht. Die Möglichkeit, Fähigkeiten sowohl durch lokale als auch durch entfernte Implementie-

rungen zu erfüllen und diese zur Laufzeit dynamisch zu wählen, ist ein wichtiges Konzept, was für die Verteilung der Aufgaben im kompletten Team genutzt wird.

Definition 29 (Capability-Implementierung Beziehung)**.** Eine Capability kann sowohl durch die lokalen Implementierungen als auch durch die der anderen Team-Teilnehmer erfüllt werden. $\mathcal{I}$ bezeichnet die Menge aller Capability-Implementierungen über alle Roboter des aktuellen Teams hinweg:

$$\forall i = (\mathcal{W}, \_, \_) \in \mathcal{I} : \forall i' = (\mathcal{W}', \_, \_) \in \mathcal{I} : \mathcal{W} \in nearbyWorlds(\mathcal{W}')$$

Durch die Definition der Funktion $required_{local}$ kann allerdings der Ausführungsort explizit eingeschränkt werden. Gibt dieses *wahr* zurück, werden nur lokale Implementierung berücksichtigt:

$$required_{local} : C \rightarrow \{\text{False}, \text{True}\}$$

Jede Implementierung i setzt eine einzelne Fähigkeit c um. Eine Fähigkeit c hingegen kann beliebig viele Implementierungen i referenzieren. Die Beziehung zwischen Implementierung und Fähigkeit wird über die c_{impl} Funktion definiert:

$$c_{impl} : \mathcal{I} \rightarrow \mathcal{C}$$

Die $impl$ Funktion hingegen gibt alle ausführbaren Implementierungen für eine gegebene Capability c:

$$\begin{aligned} impl : \mathcal{C} &\rightarrow \mathcal{P}(\mathcal{I}) \\ impl(c) &= \{i \mid i \in I \wedge c_{impl}(i) = c \wedge executable(c, i) \wedge validate_{precondition}(i)\} \end{aligned}$$

Eine Implementierung ist ausführbar, wenn sie eine Capability umsetzt, deren Vorbedingungen (Def. 31) erfüllt und es eine gültige Implementierung und Ausführungsort gibt (Def. 30) ($executable(c, i) =$ True) , was sowohl lokal, als auch remote erfüllt werden kann.

Definition 30 (Capability Ausführbarkeit)**.** Mit der $executable(c, i)$ Bedingung kann geprüft werden, ob für eine Fähigkeit c eine ausführbare Implementierung i und ein notwendiges Environment zur Ausführung verfügbar ist. Dies ist zum einen der Fall, wenn eine Implementierung auf dem gleichen Roboter verfügbar ist wie die Fähigkeit selbst. Dann wird die Fähigkeit als Subtree der Capability angelegt (lokale Ausführung). Zum anderen ist dies der Fall, wenn keine lokale Ausführung gefordert wurde, aber es einen anderen Roboter gibt, auf dem die Implementierung verfügbar ist und ein *Remote Capability Slot* im BT enthalten ist

(remote Ausführung).

$$\begin{aligned} \mathit{executable}(c,i) = &(\mathcal{W}_i = \mathcal{W}_c) \vee \\ &(\neg\ \mathit{required}_{\mathit{local}}(c) \wedge \\ &\exists Env_T = (\mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \wr), \_, \_, \mathcal{W}_i) : \\ &\exists n \in \mathcal{N}_T : n = \text{RemoteCapabilitySlot}) \end{aligned}$$

Dabei bezeichnet $\mathcal{W}_i$ den Weltzustand, in dem die Implementierung i vorhanden ist und $\mathcal{W}_c$ den Weltzustand, aus dem die Fähigkeit c stammt. Die beiden sind dann identisch, wenn Fähigkeit und Implementierung vom selben Roboter stammen. Env_T Bezeichnet in diesem Fall ein Environment, das die Implementierung enthält.

Definition 31 (Vorbedingungen)**.** Insbesondere, wenn Fähigkeiten auf einem entfernten Roboter ausgeführt werden, muss sichergestellt werden, dass die notwendigen Voraussetzungen für das Ausführen der Fähigkeit vorhanden sind. Vorbedingungen sind eine Menge an Fähigkeiten, die notwendigerweise vor der aktuellen Fähigkeit ausgeführt werden müssen, wobei für jede einzelne Fähigkeit definiert werden kann, ob diese lokal ausgeführt werden muss oder eine Ausführung durch irgendeinen der Roboter im Team ausreicht. $\mathcal{R}$ beschreibt die Menge aller möglichen Vorbedingungen:

$$\mathcal{R} = \{(c,k) \mid c \in \mathcal{C}; k \in \{\text{local}, \text{remote}\}\}$$

Die $\mathit{precondition}$ Funktion gibt alle Vorbedingungen für eine Implementierung:

$$\text{precondition} : \mathcal{I} \rightarrow\ \mathcal{P}(\mathcal{R})$$

Bevor eine Implementierung ausgeführt wird, wird geprüft, ob alle Vorbedingungen im relevanten Kontext erfüllt sind:

$$\begin{aligned} \mathit{validate}_{\text{precondition}}(i) = & \\ &\forall r = (c,k) \in \mathit{precondition}(i) : \\ &\quad (k = \text{local} \wedge c \in \mathcal{N}_T) \vee \\ &\quad (k = \text{remote} \wedge \exists \mathcal{G}_T(\mathcal{N}'_T, \mathcal{E}'_T, \wr') \in \mathit{nearbyTrees}(\mathcal{W}) : c \in \mathcal{N}'_T) \end{aligned}$$

Wobei $\mathit{nerabyTrees}$ die BTs der *nearby worlds* liefert. Können die Vorbedingungen nicht validiert werden, wird die Implementierung fehlschlagen, bevor irgendein Knoten getickt wird. Die Vorbedingungen können auch genutzt werden, um fehlende Fähigkeiten zu synthetisieren. In der Praxis gestaltet sich dies jedoch insbesondere aufgrund der nötigen Verbindung der Optionen als schwierig.

Definition 32 (Dynamische Bindung)**.** Eine Capability $c \in \mathcal{C}$ ist nicht statisch mit einer Implementierung $i \in \mathit{impl}(c)$ verknüpft, sondern kann die Zuordnung

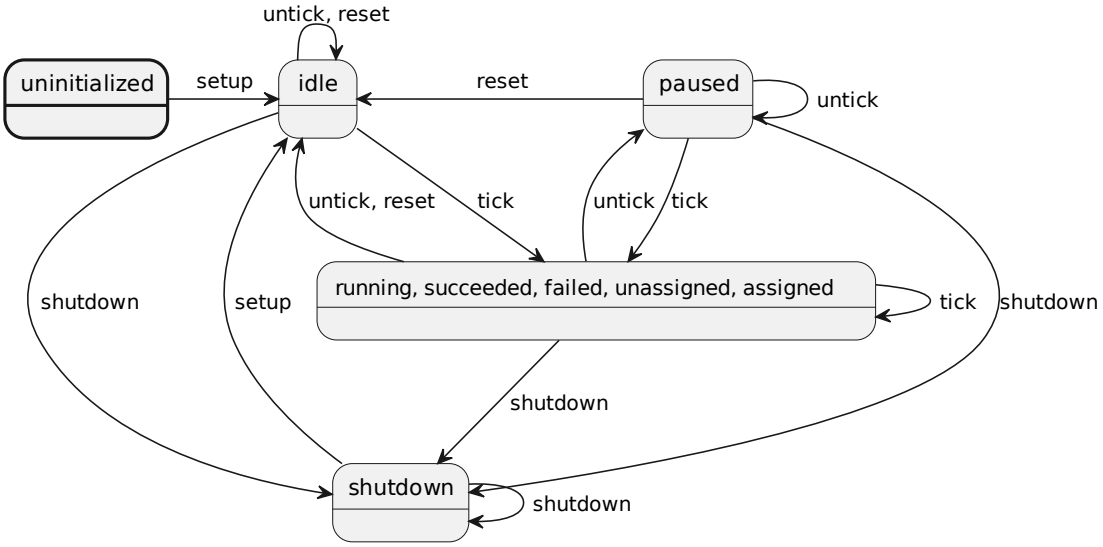


Abbildung 4.23.: Zustandsübergänge für **non-capability** BT Knoten während eines Update-Schrittes. Das Diagramm erweitert die Zustände aus Abbildung 4.16 um die neuen Zustände *unassigned* und *assigned*. Die *error*-Zustände und Übergänge wurden für bessere Lesbarkeit weggelassen und *running*, *succeeded* und *failed* wurden zusammengefasst. Quelle [OH22]

während der Ausführung des BT mehrfach ändern. Durch diese neue Fähigkeit eines Knotens sind zwei neue Knoten-Zustände erforderlich:

$$\begin{aligned} state(n) \in \mathcal{N}_T = \{&uninitialized, error, idle, \\ &succeeded, failed, running, \\ &shutdown, unassigned, assigned\} \end{aligned}$$

Da auch nicht-Capability-Knoten in diese Zustände eintreten können, etwa wenn eines ihrer Kinder ein Capability-Knoten ist, müssen die Zustände und Übergänge für alle Knoten definiert werden. Abbildung 4.23 zeigt die Zustandsübergänge für solche Knoten, wenn die Update-Funktion $update(n, a, Env_T)$ aufgerufen wird. Die neuen Zustände sind identisch mit dem *running* Status und haben damit für nicht-Capability-Knoten keinen Effekt. Capabilities hingegen verhalten sich wie in Abbildung 4.24 dargestellt. Mit der Funktion $assign_c(c)$ kann die aktuell zugeordnete Implementierung eines Knoten ermittelt werden.

$$assign_c(c) : \mathcal{C} \rightarrow \mathcal{I} \cup \{\varnothing\}$$

$$c \rightarrow \begin{cases} \varnothing & state(c) \in \{idle, shutdown, error, unassigned\} \\ i \in impl(c) & state(c) \in \{assigned, paused, running, \\ & \qquad succeeded, failed\} \end{cases}$$

Die gewählte Implementierung muss während des *unassigned* Zustandes festgelegt werden, da sich die Zuordnung in den Zuständen *assigned, paused, running,*

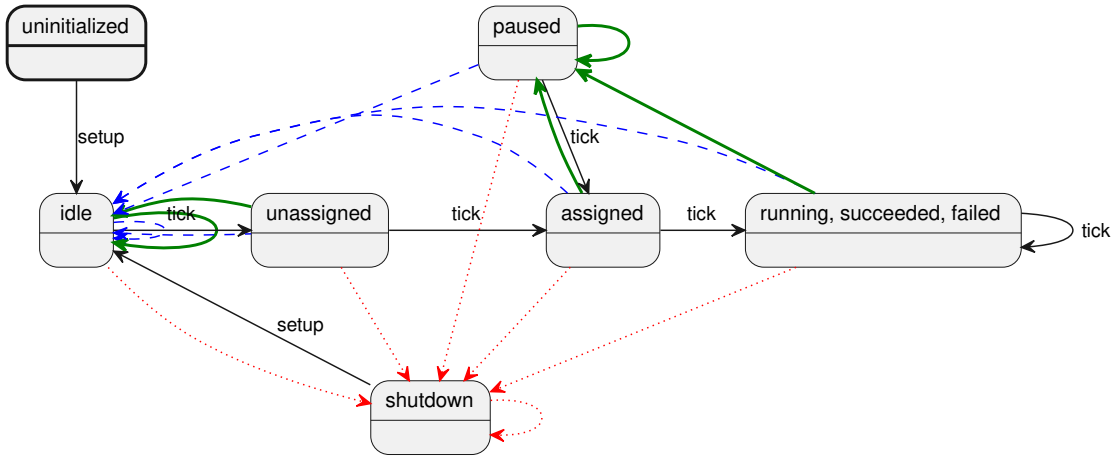


Abbildung 4.24.: Zustandsübergänge für **capability** BT Knoten während eines Update-Schrittes. Die *error*-Zustände und Übergänge wurden weggelassen und *running*, *succeeded* und *failed* wurden zusammengefasst. Rot gestrichelte Linien sind *shutdown* Aktionen, blau gestrichelte Linien *reset* Aktionen und grüne Linien die *untick* Aktionen. Quelle [OH22]

succeeded und failed nicht ändern darf. Nur beim Ausführen eines *reset* oder *shutdown* wird der Wert auf $\varnothing$ zurückgesetzt.

Das Verhalten der Capabilites in den neuen Zuständen wird am Abschnitt 4.4.9 noch einmal im Detail erklärt.

4.4.7. Capability IO Brücke

Die Capability c ist ein Knoten im BT $\mathcal{G}_T$ und damit Teil des Environemts Env_T. Die Capability-Implementierung $i \in impl(c)$ wird hingegen im potentiell komplett separaten Environment Env_T^i ausgeführt. Für eine sinnvolle Ausführung der Implementierung müssen daher die *Capability Parameter* auf die Implementierung übertragen werden.

Auch die *Shovables* (Def. 22) extrahieren die *öffentlichen Inputs und Ouputs* eines Subtrees aus dem ausführenden Environment Env_T und senden diese an den *Slot* , wo ein Update-Schritt in seinem lokalen Environment berechnet wird. Die aktualisierten Inputs und Outputs werden dann an den übergeordneten Baum zurückgegeben und in das aktualisierte Environment Env_T' eingesetzt. Diese Art der Ausführung erlaubt es zwar, den Teil des Baumes transparent an einer anderen Stelle auszuführen und die Ergebnisse wieder einzusammeln, überträgt den Ausschnitt des Environments aber lediglich einmalig. Dadurch kann der - möglicherweise entfernt ausgeführte - Teilbaum nicht auf zur Laufzeit geänderte Inputs reagieren, was vor allem für lang andauernde Prozesse unbefriedigend ist.

Da durch das Capability-Konzept nicht nur ausgewählte Teile des Baumes entfernt ausgeführt werden, sondern ein beliebig granularer Austausch angestrebt wird, wurden daher die *CapabilityIOBridges* eingeführt:

Definition 33 (Capability IO Brücke)**.** IO Brücken sind spezielle Knoten als Teil des BT der Implementierung i. Da eine Capability durch verschiedene Implementierungen ersetzt werden kann, sind die folgenden Bedingungen nur gültig, wenn der Capability eine Implementierung zugeordnet wurde.

$$assign_c(c) \neq \varnothing$$

Für jede Capability c existiert eine c_{in} (Input Brücke) und c_{out} (Output Brücke) , für die gilt:

$$\forall c \in \mathcal{C} : \exists c_{\text{in}} \in \mathcal{C}_{in} : c_{\text{in}} = \ br_{in}(c)$$
$$\forall c \in \mathcal{C} : \exists c_{\text{out}} \in \mathcal{C}_{out} : c_{\text{out}} = \ br_{out}(c)$$

$br_{in}(c)$ und $br_{out}(c)$ sind bijektive Funktionen, die den Capability-Knoten auf die Bridge-Knoten abbilden. $\tilde{br_{in}}(c_{br})$ und $\tilde{br_{out}}(c_{br})$ sind die entsprechenden Inversen. Wie auch die Fähigkeiten selbst, sind beide Brücken Blatt-Knoten im BT:

$$\forall c_{br} \in \mathcal{C}_{in} \cup \mathcal{C}_{out} : c_{br} \in \mathcal{N}_T$$
$$\forall c_{br} \in \mathcal{C}_{in} \cup \mathcal{C}_{out} : maxChildren(c_{br}) = 0$$
$$\forall c_{br} \in \mathcal{C}_{in} \cup \mathcal{C}_{out} : nodeClass(c_{br}) = \text{Leaf}$$

Definition 34 (Capability Input Brücke)**.** Die Capability-Input-Brücke dient dazu, den Input der Capability für die Implementierung verfügbar zu machen. Dafür werden die an der Capability anliegenden Inputs als Output der Bridge abgebildet:

$$\begin{aligned} outputs(c_{in}) : \mathcal{C}_{in} &\rightarrow \mathcal{P}(\mathbb{P}) \\ c &\rightarrow \{p = (n,k,t) \mid p \in inputs(\tilde{br_{in}}(c_{in}))\} \end{aligned}$$

Input-Brücken haben keine eigenen Inputs:

$$\begin{aligned} inputs(c_{in}) : \mathcal{C}_{in} &\rightarrow \mathcal{P}(\mathbb{P}) \\ c &\rightarrow \emptyset \end{aligned}$$

Während der Zustand der Capability c running ist $state(c) = \text{running}$, werden bei jedem Aufruf von $Env'_T = update(c, \text{tick}, Env_T)$ die durch die Brücke abgebildeten Werte aktualisiert:

$$\forall c \in \mathcal{C} : c_{\text{in}} = \ br_{in}(c) :$$
$$\forall p = (n,k,t) \in inputs(c) : \forall p' = (n',k',t') \in outputs(c_{in}) :$$
$$n = n' \implies value(p, \mathcal{W}) = value(p', \mathcal{W}')$$

$value(p, W)$ ist die Parameterwert-Funktion aus Def. 14. W and W' sind die jeweiligen Weltzustände der Capability und ihrer Implementierung. Durch diese Definition wird zum einen die Menge der übertragenen Umgebungsinformation im Vergleich zu den *Shovables* reduziert und zum anderen die Reaktivität des Verhaltens über den Austausch bei jedem *tick* sichergestellt.

Definition 35 (Capability-Output-Brücke)**.** Die Capability-Output-Brücke ermöglicht analog zur Input-Brücke die Output-Werte der Implementierung an die Capability selbst weiterzugeben. Dementsprechend werden die Inputs des Brücken-Knotens auf die Outputs der Capability abgebildet:

$$\begin{aligned} inputs(c_{out}) : \mathcal{C}_{out} &\rightarrow \mathcal{P}(\mathbb{P}) \\ c &\rightarrow \{p = (n, k, t) \mid p \in outputs(\tilde{br_{out}}(c_{in}))\} \end{aligned}$$

Output-Brücken haben keine eigenen Outputs:

$$\begin{aligned} outputs(c_{out}) : \mathcal{C}_{out} &\rightarrow \mathcal{P}(\mathbb{P}) \\ c &\rightarrow \emptyset \end{aligned}$$

Während des *running* Zustandes $state(c) =$ running der Capability c werden, wie bei der Input-Brücke bei jedem Aufruf von $Env'_T = update(c, \text{tick}, Env_T)$, die entsprechenden Ausgangswerte aktualisiert:

$$\begin{aligned} \forall c \in \mathcal{C} : c_{\text{out}} = \ & \text{br}_{\text{out}}(c) : \\ \forall p = (n, k, t) \in outputs(c) : \forall p' = (n', k', t') \in inputs(c_{out}) : & \\ n = n' \implies & value(p, \mathcal{W}) = value(p', \mathcal{W}') \end{aligned}$$

4.4.8. Remote Capability Slot

Der *Remote Capability Slot* ist ein BT-Knoten, der es, ähnliche wie der zuvor eingeführte *Slot*, erlaubt, einen Teil eines BT auf einem anderen Roboter auszuführen. Anders als bei dem *Slot* wird jedoch nicht der komplette Subtree übertragen, sondern durch den *Remote Capability Slot* eine lokal verfügbare Implementierung für diese Capability ausgeführt.

Definition 36 (Remote Capability Slot)**.** Ein Remote Capability Slot c_r ist ein Knoten, der auf Anfrage einer Capability c_{other} eines entfernten Roboters eine lokale Implementierung $i \in \mathcal{I}$ für diese Capability ausführen kann. Ein Slot kann nur eine Fähigkeit gleichzeitig ausführen. Die Anfrage bzw. Auswahl einer geeigneten Fähigkeit wird durch das Vergabesystem der Mission Control durchgeführt und ist nicht Teil des Slots. Sobald der Knoten eine Ausführungsanfrage, bestehend aus der Fähigkeit c und der zu verwendenden Implementierung $i = (\mathcal{W}, \mathcal{G}_T(\mathcal{N}_T, \mathcal{E}_T, \iota), \mathcal{G}_D(\mathcal{N}_D, \mathcal{E}_D))$ erhält, erstellt dieser eine neue Ausführungsumgebung Env^c_T mit

$$Env^c_T = (\mathcal{G}_T, \mathcal{G}_D, \Sigma, \mathcal{W})$$

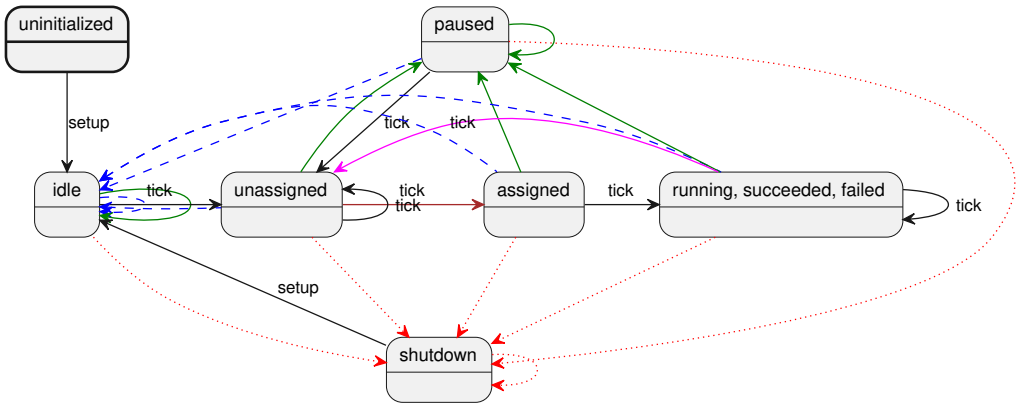


Abbildung 4.25.: Zustandsübergänge für den *Remote Capability Slot* Knoten während eines Update-Schrittes. Der error Zustand und dessen Übergänge wurden weggelassen und die Zustände *running*, *succeeded* und *failed* kombiniert. Rot gestrichelte Linien sind *shutdown* Aktionen, blau gestrichelte Linien *reset* Aktionen und grüne Linien die *untick* Aktionen. Der braune Pfeil ist der durch die Zuweisung einer *Capability* ausgelöste Übergang zu *assinged* und die pinke Linie der Übergang bei einem Tick nachdem *succeeded* oder *failed* Zustände erreicht wurden. Quelle [OH22]

Diese Ausführungsumgebung existiert im gleichen Weltzustand wie der BT der den *Remote Capability Slot* c_r enthält, nutzt aber einen unabhängigen Knoten und Datengraphen $\mathcal{G}_T, \mathcal{G}_D$. Aktionen für das Environment Env_T^c werden wie bei einem Dekorator durch den Knoten c_r geleitet, wodurch das Environment Env_T^c mit der gleichen Tick-Rate wie der Knoten c_r aktualisiert wird. Durch die *CapabilityIOBrücken* werden die Parameter transparent mit denen des entfernten Roboters ausgetauscht, zudem wird der Wurzelzustand vom $\mathcal{G}_T$ als Zustand für die entfernte Capability c_{other} zurückgegeben. Sobald die Implementierung einen Endzustand erreicht, werden alle zugehörigen Artefakte aus dem lokalen Environment, in dem der *RemopteCapabilitySlot* ausgeführt wird, entfernt. Die Umgebung wird damit auf den Zustand zurückgesetzt, wie er vor der Annahme der Anfrage war. Der *Remote Capability Slot* nutzt ähnliche Zustände wie eine normale Capability-Node, jedoch mit kleinen Änderungen (4.25): Nach dem *untick* gehen die Zustände *unnasigned*, emphassigned und emphrunning über emphpaused wieder in den emphunassigned Zustand über, zudem erfordert der Übergang vom emphunassigned Zustand in in den emphassigned Zustand ein externes Signal von der Capability, welche die entfernte Ausführung anfordert.

4.4.9. Verhalten der Capabilities

Durch die gezeigten Konzepte kann die Capability dazu genutzt werden, um dynamisch Fähigkeiten zu instanziieren und auszuführen, unabhängig davon, ob

diese lokal oder entfernt ausgeführt werden.

Im Folgenden wird das Verhalten des Knotens in den neue Zuständen noch einmal im Detail erklärt.

Definition 37 (Verhalten im *unassigned* Zustand)**.** Wenn ein Updateschritt $Env'_T = update(c, \text{tick}, Env_T)$ aufgerufen wird, während der Knoten noch nicht zugeordnet ist ($state(c) = \textit{unassigned}$), wird eine geeignete Implementierung für die Fähigkeit gesucht. Dafür kommt ein Auktionssystem zum Einsatz (siehe Abschnitt 4.5.4). Wenn eine gültige Implementierung gefunden wird, geht die Capability in den *assigned* Zustand über und es gilt in Env'_T : $state(c) = \textit{assigned}$ sowie $assign_c(c) \neq \varnothing$.

Definition 38 (Verhalten im *assigned* Zustand)**.** Wenn ein Updateschritt $Env'_T = update(c, \text{tick}, Env_T)$ aufgerufen wird, während der Knoten bereits zugeordnet ist ($state(c) = \textit{assigned}$), wird die Fähigkeit den in der assign Funktion definierten Baum $assign_c(c)$ initiieren, indem eine *setup* Aktion durchgeführt wird. Das Setup funktioniert für lokale Capabilities identisch wie das der Subtrees in Slots (siehe Def. 24), wobei die Fähigkeit selbst den Slot darstellt. Für entfernt ausgeführte Capabilities werden *Remote Capability Slots* verwendet, welche als Knoten in den entfernten BT enthalten sein müssen (siehe 4.4.8). Das dadurch erzeugte Environment (egal ob lokal oder remote) wird als Env^c_T bezeichnet, wobei $r \in \mathcal{N}_T$ den Wurzelknoten der Implementierung bezeichnet. Nach der Initialisierung geht der Knoten in den *running* Zustand über.

Definition 39 (Verhalten im *running* Zustand)**.** Wenn ein Updateschritt $Env'_T = update(c, \text{tick}, Env_T)$ aufgerufen wird, während der Knoten bereits läuft ($state(c) = \textit{running}$) leitet die Fähigkeit den *tick* direkt an die Implementierung im jeweiligen Environment weiter. Vor dem tick werden die aktuellen Inputs $inputs(c)$ und die Umgebung Env^c_T durch die *CapabilityIOBridge* weitergegeben. Dann wird ein Update-Schritt mittels $Env^{c\,'}_T = update(r, \text{tick}, Env^c_T)$ aufgerufen. Nach dem tick werden dann die Werte der $outputs(c)$ und der $state(r) \in S$ aus dem Environment $Env^{c\,'}_T$ durch die *CapabilityIOBridge* extrahiert. $state(c) = state(r)$ wird gesetzt und die Werte für die $outputs(c)$ des Environments Env'_T aktualisiert.

4.4.10. Implementierung als `ros_bt_py`

Um die vorangegangenen Konzepte umzusetzen, wurde die `ros_bt_py`-Bibliothek entwickelt [Heppner et al., 2023],[Heppner et al., 2024], [BH22], [HH18], [OH22] und als Open-Source-Projekt [5] veröffentlicht. Die Bibliothek erlaubt es, BTs zu erstellen, zu verwalten und auszuführen, die die vorangegangenen Definitionen erfüllen. Durch eine native Integration und Nutzung von ROS können verschiedene Roboter und ihre bestehende Funktionalität schnell angesprochen werden.

[5] `https://github.com/fzi-forschungszentrum-informatik/ros_bt_py`

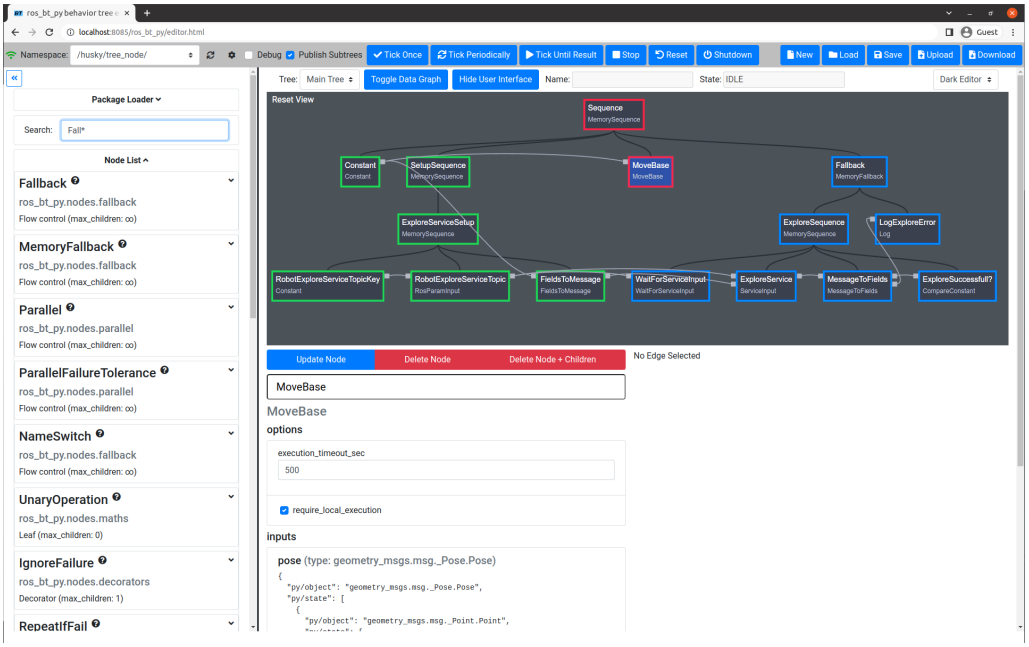


Abbildung 4.26.: Web-Basierter BT-Editor in der `ros_bt_py`-Bibliothek. Links werden die verfügbaren Knoten angezeigt. Hier können jederzeit neue Knoten aus externen Paketen nachgeladen werden, etwa um anwendungsspezifische Knoten oder Fähigkeiten zusammen mit ihrer Implementierung auszuliefern. In der Mitte wird der aktuelle Baum angezeigt. Solange der Baum nicht gestartet ist, können Knoten aus der Liste per Drag & Drop hinzugefügt werden. Zur Laufzeit wird der aktuelle Status der einzelnen Knoten durch unterschiedliche Farben angezeigt (Grün = succes, Rot = failed, Blau = idle). Mit der oberen Leiste kann die Ausführung des Baumes (z.b. tick once) gesteuert werden, in dem unteren Detailfeld werden Optionen oder aktuelle IO-Werte angezeigt. Im Hauptfenster werden die Datenkanten grau angezeigt.

Ein ebenfalls entwickelter Web-Editor (Fig. 4.26) erlaubt es, existierende Knoten per Drag&Drop zu einem BT zusammenzustellen und dadurch in kürzester Zeit neue Verhalten zu entwickeln oder diese als subtree wiederzuverwenden. Komplette Bäume können zudem als Capability abgespeichert und mittels der speziellen Capability-Knoten im BT genutzt werden.

Die Kern-Bibliothek besteht auf dem Behavior-Tree-Manager als Ausführungsumgebung und der Web-Oberfläche des Editors. Dazu kommen weitere Module, welche das Capability-Repository für das Laden und Speichern von Fähigkeiten bereitstellen und die mission control, welche die Ausführung der Fähigkeiten lokal und zwischen verschiedenen Robotern koordiniert sowie das Assignment-System, welches die Auktion und Zuordnung von Fähigkeiten koordiniert. Alle Aktionen, etwa das Ticken des Baumes oder das Laden eines Trees, können auch

Name	Reactivity	Arguments	Black Board	Data Wirings	Async	GUI	ROS	Distributed Exec.
BehaviorTree.cpp[a]	✓	✓	✓	✗	✓	✓	✓	✗
Py_trees[b]	✗	✗	✓	✗	✗	✓	✓	✗
CoSTAR[c]	✗	✓	✓	✗	✗	✗	✗	✗
BrainTree[d]	✓	✓	✓	✗	✗	✗	✗	✗
go-behave[e]	✗	✓	✓	✗	✗	✗	✗	✗
ActionFW[f]	✓	✗	✗	✗	✗	✗	✗	✗
ros_bt_py[g]	✓	✓	✗	✓	✓	✓	✓	✓

Tabelle 4.3.: Vergleich verschiedener aktiver BT-Implementierungen und ihrer Merkmale. Quelle: [Heppner et al., 2023] ©2023 IEEE

[a] `https://github.com/BehaviorTree/BehaviorTree.CPP`
[b] `https://github.com/splintered-reality/py_trees`
[c] [94]
[d] `https://github.com/arvidsson/BrainTree`
[e] `https://github.com/askft/go-behave`
[f] `https://bitbucket.org/brainific/action-fw`
[g] `https://github.com/fzi-forschungszentrum-informatik/ros_bt_py`

per ROS-Interface gesteuert werden. Dadurch kann entweder ein web-basiertes GUI transparent verwendet werden oder ein Roboter direkt auf die Funktionalität zugreifen. Darüber hinaus bietet der Editor einige Funktionen, wie etwa das Debugging eines BT, die Anzeige der Knoten-Zustände während der Ausführung sowie das Speichern und Laden eines Trees.

Im Vergleich mit anderen aktiven BT-Bibliotheken (siehe Tab 4.3) ist die Implementierung nicht nur mit ihren Features auf dem aktuellen Stand der Technik, sondern erweitert diesen, insbesondere durch die native Integration der verteilten Ausführung.

4.5. Fähigkeitsbasierte Kooperation

In Abschnitt 4.2 wurden das Konzept und Modell von Fähigkeiten erläutert, in Abschnitt 4.3 das Framework zur Nutzung der Fähigkeiten vorgestellt und in 4.4 eine formelle Definition der BTs für die Fähigkeits- und Missionsdarstellung gegeben. Mit diesen Definitionen als Grundlage kann nun gezeigt werden wie durch diese Bausteine eine Kooperation auf Fähigkeits-Basis zwischen den Robotern erzielt werden kann. Dieser Abschnitt behandelt konkretere Implementierungsentscheidungen und die Nutzung der Modelle in der Mission-Control.

Die Kooperation von den Robotern im Team bedeutet nun vor allem die Verteilung einer Mission bzw. der Fähigkeiten, aus der diese aufgebaut ist, auf die unterschiedlichen Systeme und die Koordination der Ergebnisse und Ausführung. Dafür werden vor allem die verschiedenen Systeme der *Mission Layer* (vgl. 4.3.3 und Abb. 4.15) genutzt. Für die Kooperation im Team ist es notwendig, folgende Fragen zu beantworten:

- Wie kann eine fähigkeitsbasierte Mission für das Team erstellt werden?
- Wie werden die Fähigkeiten ermittelt und zwischen den Robotern kommuniziert?
- Wie kann entschieden werden, wer eine Fähigkeit ausführen soll?
- Wie können die Fähigkeiten oder Teile davon in einer Mission durch andere Roboter übernommen werden?
- Wie können wechselnde Gegebenheiten und Team-Kompositionen berücksichtigt werden?
- Wie können Fähigkeiten kombiniert und erweitert werden?

4.5.1. Missionen

Eine Mission wird als BT aus *Capabilities* modelliert und mit Hilfe der vorgestellten Werkzeuge sowohl erstellt als auch ausgeführt, wobei der Vergabe-, Instanzzierungsprozess und die Ausführung permanent durchlaufen werden. Es dürfen auch explizit abstrakte Fähigkeiten für die Modellierung genutzt werden, da die genaue Implementierung erst zur Laufzeit bestimmt wird und dadurch die Flexibilität der Modellierung optimal ausgenutzt werden kann. Abb 4.27 zeigt, wie eine Mission durch die verschiedenen Subsysteme der Mission Control (Abb. 4.15) aufgebaut wird.

Durch das *Capability Management* werden die für einen Roboter verfügbaren Fähigkeiten ermittelt und im Team bekannt gegeben (1). Ein einzelner Roboter hat zunächst seine ihm selbst bekannten Fähigkeiten zur Verfügung, dabei ist es unerheblich, ob diese als rein abstrakte Beschreibungen oder konkrete Implementierung vorhanden sind. Sobald weitere Roboter mit dem entwickelten Framework das Netzwerk betreten, geben diese ihre Fähigkeiten bekannt, es kann dann also die Menge aller bekannten Fähigkeiten über alle Roboter verwendet werden. Eine Mission kann dann aus den bekannten Flusskontrollknoten eines Behavior Trees, den lokal vorhandenen Knoten des BT-Frameworks, den lokal bekannten Fähigkeitsdefinitionen (Capabilities) sowie den im Team vorhandenen Fähigkeitsdefinitionen (Capabilities) erstellt werden (2). Die entstehende Mission kann nicht direkt ausgeführt werden, da sie zunächst nur aus der Modellinformation der Fähigkeiten aufgebaut ist, sie wird daher an die *Task Allocation* übergeben. Die *Task Allocation* ermittelt im Wechsel mit dem *Capability Management*, welche

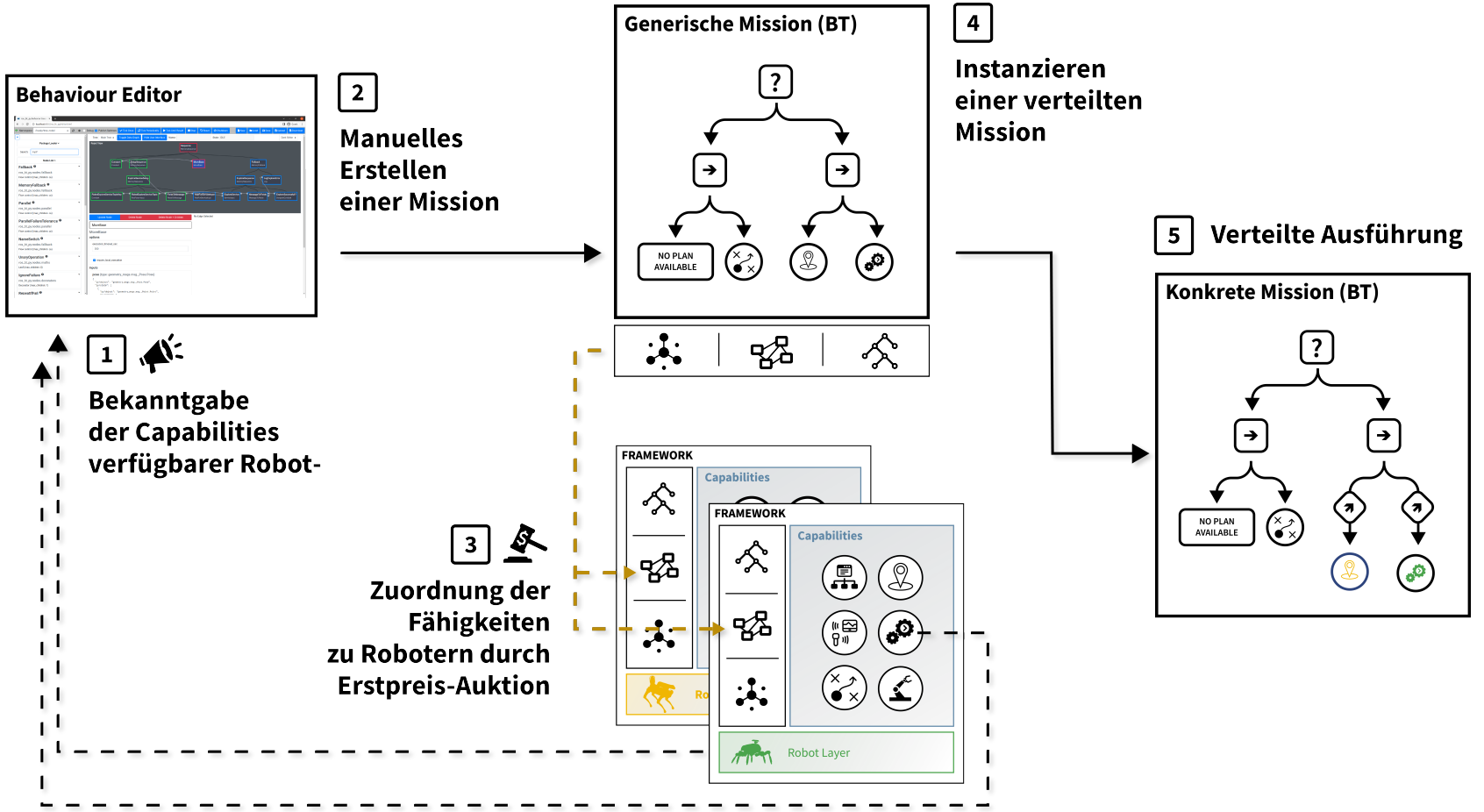


Abbildung 4.27.: Aufgaben und Ablauf der Missionssteuerung. Fähigkeiten werden durch die Mission Control global bekannt gegeben (1), der Nutzer oder andere Roboter können diese und eigene Fähigkeitsdefinitionen nutzen, um eine Mission zu beschreiben (2) welche dann wiederum im Team aufgeteilt (3) und vor der Ausführung final instanziiert wird (4). Die Missionssteuerung überwacht auch die Ausführung der verteilten Mission und greift ggf. ein, falls eine Neuvergabe notwendig wird (5).

der Fähigkeiten lokal ausgeführt werden sollten oder welche an andere Roboter vergeben werden können bzw. müssen, weil keine lokale Implementierung vorhanden ist oder ein Subtree explizit als Shovable deklariert wurde. Um die Fähigkeit im Team zu verteilen, wird die Fähigkeit durch die *Task Allocation* zur Vergabe auktioniert (3). Sobald ein Roboter den Zuschlag erhalten hat, wird die Fähigkeit per *Remote Capability Slot* an diesen ausgelagert (oder per *Slot* wenn das *Shovable* System verwendet wird). Durch diesen Schritt werden alle Fähigkeiten so weit aufgelöst, dass, verteilt über mehrere Roboter, nur noch ausführbare Knoten im gesamten Behavior Tree vorhanden sind (4). Vorbedingungen oder andere Syntheseschritte werden zu diesem Zeitpunkt durch die *Mission Control* in den Baum eingefügt. Die vollständig instanziierte Mission wird dann durch die Mission Control ausgeführt bzw. während der Ausführung überwacht (5). Wenn z.B. eine vergebene Fähigkeit nicht mehr ausgeführt wird, etwa, weil ein Roboter einen Defekt hat, wird die entsprechende Fähigkeit erneut vergeben.

Während ein BT mit Shovables explizit vor der Ausführung zugeordnet werden muss, erfolgt dies durch Fähigkeiten erst während der Laufzeit, Schritt 3 wird also regelmäßig wiederholt, sobald die Fähigkeiten selbst getickt werden, nicht statisch im Voraus.

Das Aufbauen einer Mission aus Fähigkeiten erlaubt es, diese durch die integrierten Funktionen der Bibliothek automatisch zu verteilen. Die eigentlich designte Mission wird für die Verteilung nicht verändert, was auf der einen Seite sicher stellt, dass die Mission exakt so durchgeführt wird, die der Designer es beabsichtigt hat, aber auf der anderen Seite nicht automatisch nebenläufig ist oder Aktivitäten auf eine gemeinsame Zeit oder Aktivierung optimiert.

4.5.2. Capability Repository and Discovery

Das *Capability Management* verwaltet alle verfügbaren Capabilities und koordiniert diese mit anderen Robotern im selben Netzwerk über das *Capability Interface Exchange Protocoll*.

Lokale Capabilities werden beim Starten des Systems oder auch nachträglich von der Festplatte geladen und im *Capability Repository* im Speicher gehalten. Fähigkeiten werden, wie in 4.2.3 beschrieben,aus einer Ordnerstruktur geladen (oder gespeichert), bei der die interface.yaml die Fähigkeit selbst definiert und dazu beliebig viele Implementierungen in einem Unterordner bereitstellt. Durch die Serialisierung und Speicherung als menschenlesbares Format und die Speicherung in der angegebenen Ordnerstruktur soll eine einfache Wiederverwendbarkeit und Transferierbarkeit der Fähigkeiten nach dem Vorbild der ROS-Pakete erzielt werden.

Remote Capabilites werden vom *Capability Management* über das *Capability Interface Exchange Protocoll* ausgetauscht, welches die ROS-publisher und -subscriber

Architektur ausnutzt. Jedes *Capability Management* verbindet sich mit einem globalen ROS-Capability-Interface-Topic, auf dem die serialisierten Fähigkeitsdefinitionen übermittelt werden. Die übertragenen Fähigkeiten werden ebenfalls im lokalen *Capability Repository* hinterlegt, können jedoch nicht abgespeichert[6] werden und enthalten auch keine Implementierungsinformationen. Zu jeder Fähigkeit wird außerdem ein Zeitstempel hinterlegt, um veraltete Capabilties zu erkennen und aus dem *Capability Repository* zu entfernen.

Das Protokoll sieht folgende Trigger für das Übertragen der Fähigkeiten auf dem Topic vor:

- Ein neues Capability Management verbindet sich mit dem Topic
- Ein Capability Management meldet sich von dem Topic ab
- Eine neue Capability wurde von der Festplatte geladen (nur die Liste des ladenden Moduls wird neu übertragen)
- Ein expliziter Trigger, um einen Austausch zu beginnen
- Ein periodischer Trigger

4.5.3. Fähigkeitskosten

Sobald eine Fähigkeit in der Mission genutzt werden soll, entscheidet die *Task Allocation* darüber, welche Implementierung, und dadurch auch welcher Roboter, am besten geeignet ist. Um zu entscheiden, welche Implementierung zu einem gegebenen Moment die geeignetste ist, wird ein eine Metrik erforderlich, mit der die verschiedene Implementierungen bewertet und miteinander verglichen werden können. Ein oft genutztes Maß dafür ist der *Utility*-Wert (Nutzwert), welcher als Konzept nativ in die `ros_bt_py` Bibliothek integriert wurde. Die Utility-Theorie wurde ursprünglich in den Wirtschaftswissenschaften eingeführt, um das Verhalten von Individuen unter Berücksichtigung ihrer Präferenzen und Bedürfnisse zu erklären. Dabei kann durch den Utility-Wert eine Priorität für eine konkrete Handlung unter Berücksichtigung der eigenen Präferenzen als auch des aktuellen Weltzustandes quantifiziert werden [95].

Das Konzept selbst ist vergleichsweise simpel, eine Berechnungsvorschrift ermittelt entweder einen Nutzwert (höher = besser) oder aber Kosten (niedriger = besser) für eine Aktion durch eine individuelle Abbildungsvorschrift. Dadurch kann eine Reihenfolge für eine Menge von Aktionen gebildet werden, also festgelegt werden, welche der möglichen Aktionen ausgeführt werden sollte. Die Abbildungsvorschrift selbst ist jedoch nicht trivial, da hierbei teilweise sehr unterschiedliche Kontexte zu berücksichtigen sind. Das Ziel, möglichst heterogene Systeme zu berücksichtigen, führt zu dem Problem, dass nicht alle Systeme mit

[6] Dies ist eine Implementierungsentscheidung, es wäre natürlich möglich die *Capability* zu speichern.

den gleichen Bedingungen und Variablen arbeiten können. Während der Utility-Wert einer Fahrfunktion eines Autos z.B. von seinem Benzinverbrauch abhängen könnte, wäre dieselbe Fahrfunktion bei einem Elektrofahrzeug seine Batterieladung und bei einer Gruppe von Fahrzeugen sicherlich eher von der Performance der gesamten Gruppe als von ihrem individuellen Verbrauch abhängig. Es ist also auf der einen Seite eine individuelle Berechnung und auf der anderen Seite eine vergleichbare, möglichst unabhängige Metrik nötig, wenn ein individuelles Maß für mehrere Systeme genutzt werden soll.

Die Utility kann auch direkt für die vereinheitlichte Koordination genutzt werden. Im Projekt Mars2020 (Abschnitt 3.3.3) wird der Utility Wert für die Koordination der Regelung genutzt, indem alle Agenten danach streben, die Entropie im Gesamtsystem zu erhöhen. Als Ergebnis steuern alle Systeme automatisch auf den nächsten unbekannten Punkt zu und koordinieren ihre Arbeiten dabei automatisch auf Basis der aktuellen Sensordaten. Das Problem an dieser Art der Koordination ist zum einen die mangelnde Synchronisierung zwischen den Systemen, etwa wenn mehrere Systeme das gleiche Ziel explorieren wollen, zum anderen aber die Notwendigkeit, das Missionsziel statisch vorzugeben und nicht mehr anpassen zu können.

Das Ziel dieser Arbeit ist es, ein dynamisches Team in einer flexiblen oder sogar spontanen Mission zu koordinieren. Dadurch sind gewisse Annahmen oder Absprachen vorab zwar möglich, konkrete und statische Metriken pro Anwendungsfall jedoch nicht zielführend. Deshalb wurde die Utility-Berechnung in der `ros_bt_py` für jeden Knoten Integriert (siehe Definition 19). Durch Aufruf der $caclulate\_utility()$ Funktion kann für jeden Knoten ein Utility wert bestimmt werden, handelt es sich bei dem Knoten um eine Kontrollfluss-Knoten, Subtree oder eine Capability dann werden die Utilities aller darin enthaltenen Knoten berechnet und nach den in den Definitionen 20 und 21 vorgestellten Regeln aggregiert. Viele Knoten liefern keine Kosten zurück und beeinflussen das Ergebnis daher nicht, andere Knoten hingegen können in ihrer Berechnung beliebig komplex werden.

Durch dieses System ist die Berechnung direkt an die konkrete Implementierung gebunden, wodurch automatisch systemspezifische Berechnungen genutzt werden. Die gleiche Fähigkeit könnte also auf zwei unterschiedlichen Systemen unterschiedliche Ergebnisse liefern, je nach gewählter Implementierung. Dadurch können dann die Ausführungsschranken bestimmt werden, welche dann wiederum von der *Task Allocation* genutzt werden, um die optimale Implementierung zu bestimmen oder ein Gebot für die Auktionierung abzugeben. Drei Ansätze der Kostenbestimmung wurden für diese Arbeit untersucht.

Statische Kosten

Die einfachste Festlegung von Kosten ist eine statische Definition durch den Ersteller der Fähigkeiten. Durch das Hinterlegen von Kosten in der Implementie-

rungsdefinition können durch den Ersteller Prioritäten codiert werden, um etwa immer die bevorzugte Implementierung auf einem bestimmten Roboter zu wählen. Statische Kosten sind vor allem dann sinnvoll, wenn sie mit einfachen Interface- oder Weltzustand-Abfragen kombiniert werden. Beispielsweise könnte eine Implementierung einer Erkennungsfunktion per RGB-Kamera immer dann bevorzugen, wenn ein Scheinwerfer aktiviert ist, andernfalls die der Tiefenbildkamera. Der wohl häufigste reale Anwendungsfall sind die Kosten ✗ (siehe Def. 19), also das Ergebnis, dass eine Fähigkeit bei gegebenem Weltzustand überhaupt nicht ausgeführt werden kann.

Der Vorteil des Ansatzes liegt in seiner einfachen Umsetzung. Fähigkeiten können dadurch vor allem zur dynamischeren Ausführung von Missionen beitragen, da immer nur die Fähigkeiten verwendet werden können, die die aktuellen Bedingungen des Weltzustandes erfüllen. Gerade für das Ermitteln der Kosten über mehrere Systeme hinweg erfordert ein statischer Ansatz allerdings auch eine größere manuelle Normierung außerhalb des eigentlichen Koordinationssystems, damit die ermittelten Werte vergleichbar bleiben und gemeinsam genutzt werden können.

Individuelle Kosten

Durch die Berechnungen als Teil des BT können der Aufruf und die Aggregation einheitlich erfolgen, die Berechnungsvorschrift selbst kann hingegen Systemspezifisch durchgeführt werden. Dadurch ist es möglich, komplexe Kosten, abhängig vom aktuellen Roboterzustand und seiner spezifischen Implementierungsdetails zu ermitteln.

Als Beispiel einer systemspezifischen Berechnung wird hier die Bestimmung der Pfadkosten des Laufroboters LAURON V (Abb. 4.28) für den SpaceBot Cup vorgestellt.

LAURON V ist ein sechsbeiniger Laufroboter, der am FZI entwickelt wurde. Er verfügt über einen 3D-Laserscanner und verschiedene Kameras für die Umgebungswahrnehmung und eine komplexe Kinematik mit 4 DoF pro Bein, wodurch er insbesondere auch schwieriges Terrain sicher durchqueren kann. Für den SpaceBot Cup [Heppner et al., 2015, Heppner et al., 2017] wurde die PlexMap [Oberländer et al., 2014] als Kartendarstellung der Umgebung und Planungsgrundlage für LAURON entwickelt. Die 2,5D Höhenkarte verwendet einen Quadtree, um die große Menge an 3D Punkten des Laserscanners in ein vor-interpretiertes Format zu bringen und dieses für die effiziente Planung zu nutzen.

Jeder Knoten des Quadtrees ist eine geschätzte Ebene und enthält unter anderen Informationen über die

- Höheninformationen für die Niveau-Zuordnung und weitere Verarbeitung
- Meta-Informationen wie die Punktdichte des Kartenabschnitts

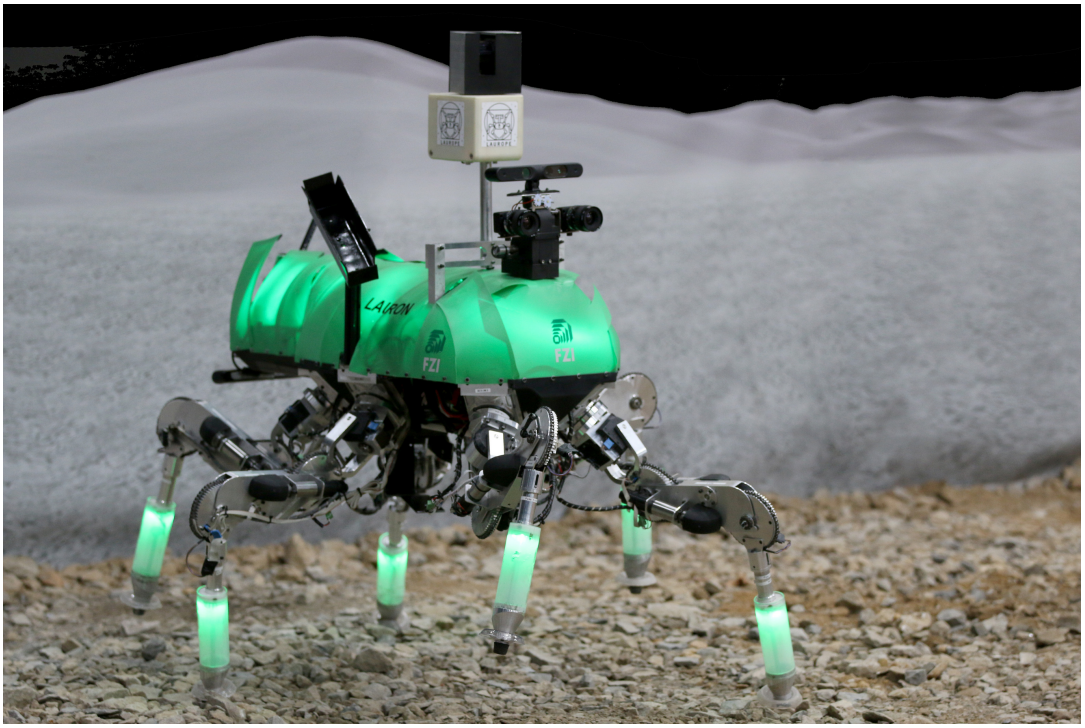


Abbildung 4.28.: LAURON V beim SpaceBot Camp 2015. Der sechsbeinige Laufroboter kann mit jeweils 4 DoF auch schwieriges Gelände überqueren und die vorderen Beine auch für Manipulationen nutzen. Neben missionsspezifischen Ausstattungen verfügt der autonome Roboter über eine Vielzahl von Sensoren, unter anderem einen rotierenden Laserscanner für hochaufgelöste 3D Punktwolken mehrere RGB und RGBD Kameras für die Objekterkennung und lokale Wahrnehmung.

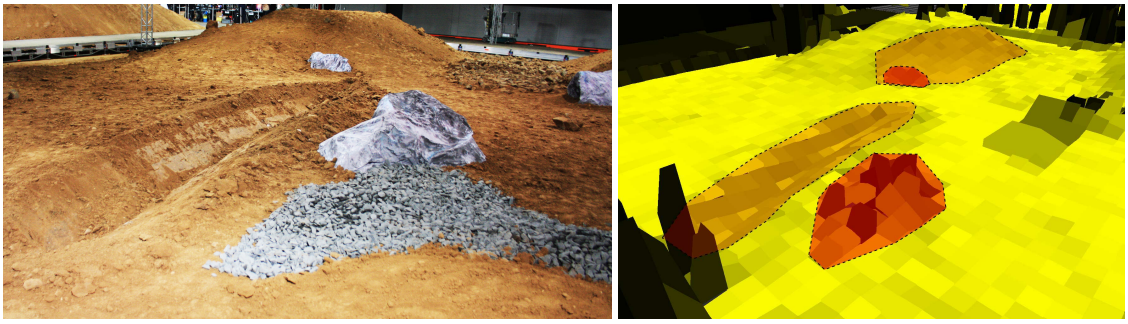


Abbildung 4.29.: SpaceBot Cup 2013 Umgebung als Foto (links) und als PlexMap Darstellung (rechts). Unterschiedliche Schattierungen repräsentieren die Steigung einer Ebene, die durch die Messpunkte aufgespannt wird. Schwarze Bereiche sind Höhenunterschiede, die normalerweise Hindernisse darstellen. Die roten Flächen wurden manuell eingezeichnet, um die Hindernisse zu illustrieren. Quelle [WH15]

- Die Oberflächennormale der geschätzten Ebene
- Varianz der Höhendaten in Richtung der Normale als Indikator für die Rauigkeit

Für eine möglichst akkurate Planung (aus Basis eines RRT*) werden diese Informationen zusammen mit der Auswertung der Kinematik bei einer Kostenberechnung für die Navigation genutzt. Für die Bestimmung der Kosten werden die

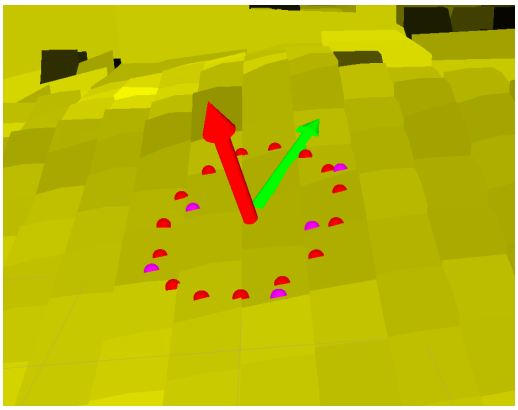


Abbildung 4.30.: Aufnahmepunkte für die Berechnung der Kosten eines Wegpunktes. Um den Anfragepunkt (Ursprung der Pfeile) werden kreisförmig Punkte abgefragt (rot) um die Orientierung der Normale über die Aufstandsfläche des Roboters (roter Pfeil) sowie den Inklinationswinkel (grüner Pfeil) zu ermitteln. Entlang der Orientierung werden die Fußpunkte (lila) speziell abgefragt. Quelle [WH15]

Punkte in einem Kreis um den den Anfragepunkt gesamplet und dadurch die Oberflächennormale und der Inklinations- bzw. Orientierungsvektor (grün) bestimmt [7]. Entlang der Orientierung können dann die Fußunkte (lila) gesampled werden. Die dadurch ermittelten Daten werden dann genutzt, um verschiedene Kosten-Kriterien auszuwerten:

- Zeitbedarf
  Der Zeitbedarf richtet sich maßgeblich nach der Strecke und der erzielten Geschwindigkeit. Strecke und Winkeldifferenz können aus der Distanz des angefragten Punktes zum vorherigen ermittelt werden, die Geschwindigkeit wird abgeschätzt indem die mögliche Schrittlänge unter Berücksichtigung der Steigung und Rauheit für jedes Bein ermittelt und kombiniert wird.

- Energieverbrauch
  Der Energieverbrauch wird anhand der statisch benötigten Energie für Sensoren, PCs usw. berechnet und mit der dynamischen Belastungsschätzung

[7] Die Orientierung des Roboter muss nicht der Inklination entsprechen, aus Stabilitätsgründen wurde dies in der Planung jedoch bevorzugt.

auf Basis von Höhendifferenzen kombiniert. Eine weitere Modifikation auf Basis des detektierten Untergrundtyps wäre zudem denkbar.

- Stabilität
  Die *Normalized Energy Stability margin* wird genutzt, um die Stabilität auf Basis der Aufstandspunkte zu bestimmen. Zudem wird die Schritthöhe verwendet, um die Rauheit des Untergrundes zu berücksichtigen.

- Unsicherheit
  Auf Basis der bekannten Punkte kann ermittelt werden, wie erforscht ein Teil der Karte bereits ist. Je mehr Punkte in dem betrachteten Kartenstück liegen, desto „bekannter“ ist die Region.

Die individuellen Kostenfaktoren werden durch eine Kostenfunktion, welche die jeweiligen Einflüsse gewichtet zu Gesamtkosten kombiniert. Durch die Gewichte können Zeit oder Energieoptimierungen für einen Pfad in den Vordergrund gestellt werden (siehe Abbildung 4.31).

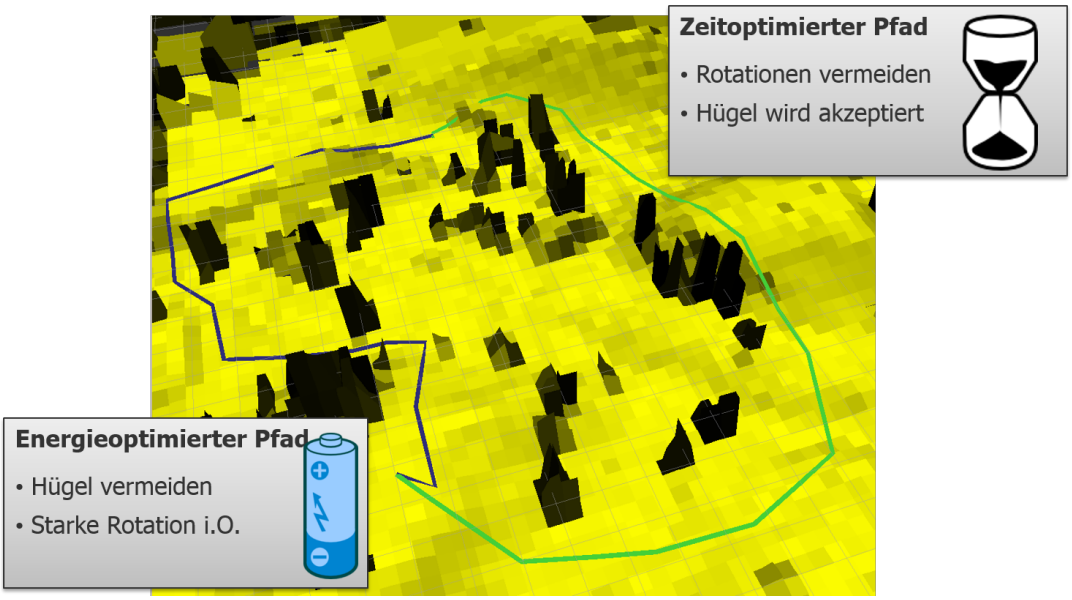


Abbildung 4.31.: Verschiedene Pfade basierend auf unterschiedlichen Präferenzen bei der Parametrisierung. Durch Gewichtung der Einflussfaktoren werden Pfade für verschiedene Kriterien, z.B. Zeit (grün) oder Energie (schwarz), optimiert. Quelle [WH15]

Alle Werte beruhen auf Schätzungen die etliche Einschränkungen in Kauf nehmen. Sie berücksichtigten aber alle individuellen Eigenschaften des Zielsystems und erlauben es daher dem jeweiligen Entwickler, die Kostenfunktion für die Planung derart einzustellen, wie es für das System optimal ist. Die dadurch berechneten Kosten können durch die BTs dann verwendet werden, um die Kosten eines Planes bzw. einer Bewegung abzufragen ohne selbst tiefgehende Wissen über das System haben zu müssen. Gleichzeitig ist es möglich, die Implementierung

mit unterschiedlicher Parametrisierung als verschiedene Implementierungen einer Fähigkeit zu hinterlegen. Eine Capability würde dadurch die verschiedenen Kosten berechnen und die geeignete Implementierung auswählen. Diese Option verdeutlicht die Symbiose aus übergeordneter Missionssteuerung und roboterspezifischer Kostenermittlung. Der Missionsdesigner braucht keine tiefgehenden Kenntnisse der Robotereigenschaften, gleichzeitig wird das Verhalten des Systems nicht starr auf eine Option festgelegt, da eine energieoptimierte Strecke zu Beginn der Mission vielleicht die beste Utility aufweist, gegen Ende der Mission aber die Geschwindigkeit wieder im Vordergrund steht.

Vereinheitlichte Kosten: Robot Health

Hochspezialisierte Metriken sind zwar genau für den individuellen Roboter, allerdings haben sie verschiedene Probleme:

- Sie sind aufgrund ihrer Komplexität schwierig zu parametrisieren bzw. zu verwenden.
- Einstellungen, die die Berechnung modifizieren, oder die Kenntnis, was eine gewisser Wert der Metrik bedeutet, erfordern einen genauen Einblick in das System
- Die Werte sind schwierig zu vergleichen, da sie keine gemeinsame Basis aufweisen.

Zwar wird für gewöhnlich versucht, Metriken auf physikalische Grundeinheiten zu abstrahieren, allerdings beinhalten diese immer noch das Wissen über die Domäne der Fähigkeit, also etwa eine Geschwindigkeit für Bewegungen, Energieverbrauch für Motoraktivierungen oder Präzision in Metern oder Millimetern bei Handhabungssystemen. Es gibt keinen Zwang, Metriken nach einem gewissen Schema zu definieren und durch verschiedene Anwendungen verschiedene Ansichten, welche Metriken relevant sind. Eine Fusion ist daher weiterhin schwierig. Als Ansatz für eine vereinheitlichte Metrik wird daher das *Health* (H) vorgestellt, welches in [Heppner et al., 2019] veröffentlicht wurde.

Health ist ein in Prozent angegebenes Maß für die Einsatzbereitschaft einer Komponenten ($comp$) zum aktuellen Zeitpunkt ($readiness_{comp}(t)$), verglichen mit der Einsatzbereitschaft zum Design-Zeitpunk ($readiness_{comp}(t_{design})$) [Heppner et al., 2019]:

$$H_{comp}(t) = \frac{readiness_{comp}(t)}{readiness_{comp}(t_{design})} * 100$$

Das Ausführen einer Fähigkeit wird mit einem Risiko (R) des potentiellen Health-Verlustes verbunden:

$$R_c = H(t_0) - H(t_1)$$

t_0 ist der Zeitpunkt *vor* dem Ausführen der Fähigkeit, und t_1 die Zeit danach.

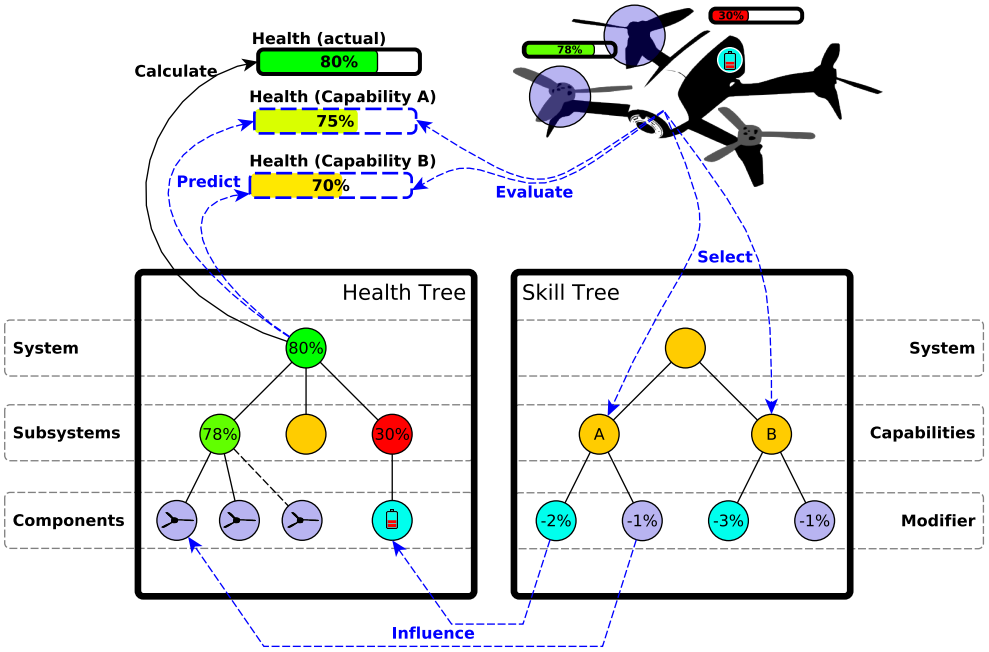


Abbildung 4.32.: Übersicht der risikobasierten Fähigkeitsauswahl. Der *Skill Tree* enthält die dem System zu Verfügung stehenden Fähigkeiten in allen Ausprägungen (Implementierungen). Jede Fähigkeitsimplementierung definiert Einflussfaktoren für unterschiedliche Komponenten. Der *Health Tree* modelliert alle Komponenten des Systems und gruppiert sie in Subsysteme. Auf Basis der aktuellen Sensorwerte und Abbildungsvorschriften können Komponenten ihren eigenen Health-Wert ermitteln, der durch die Aggregationsvorschriften der Subsysteme zu Subsystem- oder System-Health-Werten aggregiert werden können. Für einen Planungsschritt werden die Modifikatoren des Skill Tree auf Health-Werte der Komponenten angewendet und so ein gesamt Health-Wert des Systems nach Ausführung der Fähigkeit prädiziert. Quelle [Heppner et al., 2019] ©2019 IEEE

Diese Metrik ist universell, da sie weder von der Mission, noch von dem eingesetzten System selbst abhängt, für jeden Roboter relevant, da ein defektes System auch immer den Fehlschlag der Mission zur Folge hat und individuell, da das Maß des Risikos von zahlreichen individuellen Systemfaktoren abhängt. Ein Health-Wert von 100% bedeutet, die Komponente ist in einwandfreiem Zustand, etwa weil sie frisch eingebaut wurde, oder weil eine Software innerhalb ihrer Spezifikationen operiert. Ein Wert von 0% bedeutet, dass die Komponente defekt ist und ihre Funktion nicht mehr erfüllen kann. Das Bein eines Laufroboters etwa wäre direkt nach dem Einbau auf 100%. Ein Wert von 20% würde bedeuten, dass es starke Verschleißerscheinungen gab, eventuell defekte Verbindungselement oder ein Bein verformt ist, die Funktion jedoch noch gegeben ist. Erst bei 0% kann die Komponente ihre Funktion nicht mehr erfüllen, etwa weil das Bein

gebrochen ist oder der Motor keine Antriebsleistung mehr aufbringen kann.

Die Ermittlung der Metrik erfordert eine komplexe Modellierung durch die Entwickler des Roboters. Da Fehlererkennung und Mitigation relevante Aufgaben in der Robotik sind, fallen diese Überlegungen jedoch meist ohnehin während der Entwicklung an. Für diesen Zweck wurden zwei Baumstrukturen definiert: Der *Health Tree* (Abb. 4.32,links) ermittelt den aktuellen Health-Wert des Roboters und seiner Komponenten. Er besteht aus Komponenten-Knoten, welche eine Hardware- oder Software-Komponente modellieren und reale Sensorwerte durch eine individuelle Abbildungsvorschrift auf einen Health-Wert umrechnen. Die Komponenten werden in hierarchisch verschachtelbare Subsysteme gruppiert, welche wiederum eine Aggregationsvorschrift beinhalten und dadurch Subsystem oder Gesamtsystem-Werte ermitteln können. Die Aggregation kann dadurch Konzepte wie z.B. Redundanzen berücksichtigen, indem sich der Gesamtwert eines Subsystems z.b. erst nach dem Ausfall der redundanten Komponenten signifikant ändert. Der *Skill Tree* (Abb. 4.32, rechts) ist eine zweite Baumstruktur, welche dafür genutzt wird, um den Einfluss einer Fähigkeit auf die Health-Werte zu formulieren. Die Capability-Knoten stellen im Skill-Tree konkrete Capability-Implementierungen dar. Jede Capability-Implementierung hat eine Menge von Komponenten und dazugehörigen Modifikatoren, z.b. eine Reduktion um einen fixen Wert oder eine prozentuale Größe. Dabei müssen nur die Komponenten modelliert werden, die explizit durch die Fähigkeit beeinflusst werden sollen.

Wenn das System eine Fähigkeit ausführen soll, kann das Risiko durch das Anwenden eines virtuellen Zeitschritts berechnet werden. Je nach Fähigkeit werden die aktuellen Werte der Komponenten des Health-Tree entsprechend modifiziert, wodurch ein neuer Health-Wert für das System ermitteln werden kann. Die Differenz des potentiellen Health-Wertes vom aktuell ermittelten stellt das Risiko einer Fähigkeit dar und kann dementsprechend als Utility-Value oder aber für die direkten Planung bzw. Auswahl von Fähigkeiten genutzt werden. Das System wurde seit der initialen Entwicklung wiederholt erweitert, um weitere Risiken zu berücksichtigen bzw. die Berechnung und Modellierung zu vereinfachen. In [Puck et al., 2020b] wird das System um die Klassifikation und Fusion von Umgebungsrisiken erweitert. In [Schnell et al., 2020] wird insbesondere die Health-Berechnung durch den Einsatz von Gaussian Mixture Models erleichtert, um diese ohne menschliches Training aus Sensordaten zu lernen.

4.5.4. Aufgabenverteilung

Die *Task Allocation* evaluiert anhand der berechneten *Utlitiy Werte*, welche Implementierung gewählt wird. Neben den lokalen Implementierungen müssen aber auch die entfernt verfügbaren Implementierungen berücksichtigt werden. Dafür kommt ein Auktionssystem zu Einsatz, welches für diesen Anwendungsfall einige Vorteile gegenüber anderen Ansätzen bietet:

- Auktionen weisen einen guten Kompromiss aus Optimalität der Verteilung und Berechnungsaufwand auf
- Auktionen funktionieren sehr gut für dezentrale Systeme
- Bieter können ihre *Utility Werte* individuell berechnen
- Auktionen ermöglichen eine einfache, aber jederzeit erweiterbare Konfliktlösung bei gleichen *Utilities* oder anderen Ressourcenkonflikten
- Auktionssysteme können sehr gut mit einer variablen Teamkomposition arbeiten oder auf unterschiedliche Teamgrößen skaliert werden

Auktionssysteme für die Aufgabenverteilung sind ein intensiv bearbeitetes Gebiet und es gibt zahlreiche Systeme, die die Optimalität oder Performance der Zuweisungen durch mehrstufige Verfahren oder komplexere Gebotsberechnungen optimieren. Gleichzeitig wird argumentiert, dass die komplexeren Ansätze nicht signifikant bessere Ergebnisse liefern als eine einfache parallele Auktion [96], [97]. Da der Schwerpunkt dieser Arbeit auf der Kooperation weniger fähiger Systeme und damit vorrangig der Modellierung der Fähigkeiten und Mission liegt, wurde eine einfache parallele Auktion gewählt, welche sich vor allem im Protokoll stark an MURDOCH [28] (siehe auch 3.2.4) und [97] orientiert, die wiederum vom ursprünglichen Contract Net Protokoll inspiriert sind [14].

Auktionen können jederzeit von der *Task Allocation* gestartet werden, wenn die *Mission Control* eine abstrakte Fähigkeit instanziieren oder eine Auktion wiederholen möchte.

Gebote werden von allen Teilnehmern berechnet, sobald ein eine Versteigerung von einem System gestartet wurde. Jeder Bieter berechnet seine *Utility*, ermittelt unbelegte *Remote Capability Slots* und startet damit sein Gebot.

Kommunikation zwischen allen Teilnehmern erfolgt über ein zentrales Auktions-Topic, auf dem alle Teilnehmer ihre Nachrichten an alle Teilnehmer senden die dieses abonniert haben. Dadurch ist sichergestellt, dass alle Teilnehmer immer über die aktuellsten Informationen verfügen. Jeder Teilnehmer kann sowohl Auktionator als auch Bieter sein. Es dürfen beliebig viele parallele Auktionen durchgeführt werden. Antworten erfolgen asynchron, da längere Berechnungen wie das Ermitteln der Utility die Kommunikation nicht verzögern dürfen.

Das **Auktionsprotokoll** beinhaltet alle Informationen, sowohl für Auktionen wie auch Gebote, so dass ein einziger Nachrichtentyp erforderlich ist:

- **Eindeutiger Identifier**
  Identifiziert die Auktion für alle Interaktionen. Wird bei der ersten Auktionsankündigung festgelegt.
- **Zeitstempel**
  Ermöglicht die Einordnung der Gebote und Einhaltung von Zeitgrenzen

- **Eindeutige Absende ID**
  Robotername als eindeutige Bezeichnung des Absenders
- **Nachrichten Typ**
  Mögliche Typen:(Auktion (Announcement), Gebot(Bid), Aktualisierung (Update), Abbruch (Abort), Abschluss (Close), Ergebnis (Result)
- **Capability**
  Fähigkeit, die auktioniert werden soll. Nur für Auktionen relevant
- **Deadline**
  Gibt die Gültigkeit der Auktion als Zeitstempel an. Nur für Auktionen oder Updates
- **Gebot**
  Utility Wert eines Bieters. Nur für Gebote
- **Anzahl der Ausführungsumgebungen**
  Gibt die Anzahl an *Remote Capability Slots* an, die nicht bereits eine Fähigkeit ausführen. Nur für Gebote oder Updates
- **Begründung**
  Begründung bei einem Gebotsabbruch. Nur für Abbrüche
- **Eindeutige Ergebnis ID**
  Eindeutiger Robotername als Identifier für das Ergebnis. Nur für Ergebnisse

Der **Auktionsablauf** entspricht den im MURDOCH Framework [28] aufgezeigten Schritten:

- **Ankündigung (Task Announcement)**
  Start der Auktion durch Senden einer Auktions-Nachricht durch die *Task Allocation*. Diese muss eine eindeutige ID, eine Deadline und die zu auktionierende Fähigkeit enthalten.
- **Bewertung(Metric Evaluation)**
  Alle Roboter prüfen beim *Capability Management*, ob sie die angefragte Capability besitzen. Wenn ja, wird durch die *Mission Control* die aktuelle *Utlity* sowie die Anzahl an *Remote Capability Slots* bestimmt. Die Werte werden von jedem Roboter individuell und ohne Berücksichtigung der anderen Team-Teilnehmer oder anderer Gebote durch die *Utlity Berechnung* (siehe 4.4.4) bestimmt. Der Weltzustand wird bei der Berechnung der *Utlity* berücksichtigt, wodurch stets der aktuelle Kontext des Roboters in die *Utility* einfließt. Um aus den berechneten *Utility*-Schranken ein Gebot $\mathcal{B}$ zu berechnen, werden diese nach folgender Formel kombiniert:

$$\mathcal{B} = success_{min} + \frac{failure_{max} - failure_{min}}{success_{min}}$$

- **Gebot(Bid Submission)**
  Jeder Roboter gibt sein ermitteltes Gebot und Anzahl der Ausführungsumgebungen durch Absenden einer Gebots-Nachricht ab.

- **Auktionsende(Close of Auction)**
  Nach dem Ablauf der Deadline wird eine Abschluss-Nachricht gesendet und die Auswertung der Gebote gestartet. Wenn ein Gewinner ermittelt werden kann, wird eine Ergebnis-Nachricht mit dem Namen des Gewinners versendet. Der Gewinner wiederum lädt die an ihn auktionierte Fähigkeit in den *Remote Capability Slot*. Kann kein Gewinner ermittelt werden, wird eine Update-Nachricht geschickt, mit der eine neue Deadline angekündigt und der Prozess wie nach Schritt 1 weiter geführt wird oder die Auktion wird ohne Ergebnis beendet.

- **Fortschrittsüberwachung(Progress Monitoring/contract renewal)**
  Die *Mission Control* prüft, ob die Ausführungsumgebungen während der Ausführung regelmäßig einen Ping zurücksenden. Bleibt der Ping für eine definierte Schwelle an Nachrichten aus, wird davon ausgegangen, dass der Roboter die Mission nicht weiter ausführen kann und die Auktion wird erneut gestartet.

Eine **erneute Auktion**, also das Beenden der Ausführung durch einen Roboter und die Weitergabe an einen anderen, sollte immer dann durchgeführt werden, wenn sich die Bedingungen im Team derart geändert haben, dass die bisherige Lösung als nicht mehr optimal anzunehmen ist. Dadurch kann das Team dynamisch auf Änderungen der Umwelt oder der Teams selbst reagieren, was insbesondere bei längeren Tasks wichtig sein kann. Das erneute Auktionieren muss jedoch mit Zurückhaltung erfolgen, um keine ungewollten Schleifen zu erzeugen oder aber unnötiges Hin- und Herschieben von Capabilities zwischen einigen Robotern zu vermeiden. Es wurden daher folgende Auslöser für eine erneute Auktion definiert:

- Wenn der ausführende Roboter das Netzwerk verlässt oder nicht mehr reagiert
- Wenn der ausführende Roboter keinen signifikanten Fortschritt erzielt
- Wenn ein neuer Roboter das Team betritt
- Wenn die Kosten der Ausführung höher geworden sind als die des zweitbesten Bieters in der ursprünglichen Auktion

Sowohl der ursprüngliche Auktionator als auch der Gewinner der Auktion dürfen im Fall, dass eine dieser Bedingungen eintritt, eine erneute Auktion anstoßen. Nachdem eine Abbruch-Nachricht geschickt wurde, wird die aktuelle Ausführung abgebrochen und eine neue Auktion durch den ursprünglichen Auktionator veranlasst. Dadurch, dass die Zuordnung aufgelöst und die Fähigkeit wieder an den original Auktionator *zurückgegeben* wird, werden Ringschlüsse vermieden, die bei einer Auktion durch den vorherigen Gewinner auftreten könnten.

Shovables und Slots wurden vor den Capabilities entwickelt und folgen daher einem etwas anderen Ablauf, der noch keine Auktionen verwendet. Für die Ausführung eines Shovables wird der komplette Subtree an alle möglichen Systeme in Reichweite, also alle Systeme mit einem aktiven Slot, gesendet (geschoben). Die Systeme evaluieren ihre Utility und antworten mit ihrem aus dem Utility-Wert berechneten Gebot (Vergleiche Definition 19 mit Utiltiy-Wert: $\mathcal{U} = \mathbb{R} \cup \{✗, ?\}$). Der Shovable-Dekorator wählt dann anhand folgender Heuristik:

- Entferne alle Ergebnisse, die ein ✗ enthalten
- Sortiere die übrigen Ergebnisse aufsteigend nach dem Durchschnitt aller Werte, die nicht ? sind
- Sortiere die Ergebnisse mit einem stabilen Sortieralgorithmus in aufsteigender Reihenfolge der Anzahl der ? in den Ergebnissen
- Wähle das erste Element der sortierten Liste

Die Heuristik wählt damit den kleinsten Utility-Wert mit den geringsten Unbekannten. Die „Vergabe“ erfolgt dann nach dem *Winner-Takes-All*-Prinzip, es gibt also keine Auktion oder erneute Auktionen bei Änderungen. Da die Verwendung von Shovables durch Capabilties abgelöst werden soll, wurde der Auktionsmechanismus hier nicht nachgezogen.

Lokale Capabilities werden von der *Task Allocation* bei der Entscheidung für die beste Ausführungsumgebung ebenfalls berücksichtigt, so dass die *Mission Control* keine Unterscheidung durchführen muss, wo eine angeforderte Fähigkeit ausgeführt werden soll. Wenn eine lokale Implementierung vorhanden ist, wird für diese genau wie für die entfernt ausgeführten Varianten ein Gebot berechnet und bei der Entscheidung über den Ausführungsort berücksichtigt. Wird die Fähigkeit lokal ausgeführt, verhält sie sich nahezu simultan zu einer entfernt ausgeführten, da sie in einer dedizierten Ausführungsumgebung gestartet wird die nicht ohne Weiteres mit dem ursprünglichen Ausführungskontext verbunden ist. Allerdings wird der *Tick* direkt wie bei einem *Subtree* weitergegeben.

4.5.5. Kompatibilität und Vorbedingungen

Durch die Möglichkeit, Fähigkeiten innerhalb einer Mission über unterschiedliche Roboter zu verteilen und dabei eine jeweils individuelle Implementierung zu verwenden, tauscht jedes System effektiv einen Teil der Mission gegen seinen eigenen aus. Damit die Mission dadurch nicht verändert oder gefährdet wird, ist es notwendig sicherzustellen, dass die gewählten lokalen Fähigkeiten mit der ursprünglich gewählten Fähigkeit der Mission auch wirklich kompatibel sind, alle Vorbedingungen für die Ausführung erfüllen und ein kompatibles Interface aufweisen, um mit den übrigen Komponenten zu interagieren. Vorbedingungen sind dabei nicht wie bei High-Level Planern zu verstehen, die eine Aktionssequenz durch Vor- und Nachbedingungen vom Axiomen beschreiben, sondern

dienen dazu, die dynamische Verteilung von Fähigkeiten auf multiple Systeme zu verifizieren und durch Synthese von (lokalen) Fähigkeiten zu unterstützen. Für diese Arbeit wurden zwei konkrete Ansätze untersucht die Kompatibilität sicherzustellen.

Der **MDE Ansatz** [Awad et al., 2016], [Zander et al., 2015], [Bastinos et al., 2014] wurde bei der Vorstellung des Capability-Modells (Siehe Abschnitt 4.2.3 und Abbildung 4.9) bereits erläutert. Jeder Komponenten wird nach diesem Modell ein Typ zugeordnet, der eine Menge an Attributen, ROS-Interfaces und Capabilities definiert. Die Definition dieser Typen erfolgt dabei in einer hierarchisch angeordneten Ontologie (Abbildung 4.11 auf Seite 74), die Subklassen eines Typs sind Spezialisierungen mit konkreteren Interfaces oder feingranulareren Fähigkeiten.

Komponenten, die vom gleichen Typ, z.B. *ObjectDetection* sind, werden als gleichwertig und dadurch austauschbar angenommen. Je spezifischer ein Typ ist, desto genauere Anforderungen hat dieser an die entsprechenden Interfaces oder Fähigkeiten. Eine *Perception* Komponente könnte etwa ein *sensor_msgs/Image* ROS-Topic erwarten und die Fähigkeit lediglich als *SensingCapability* beschreiben, während die *ObjectDetection* auch noch ein spezielles Topic für die Beschreibung des Objektes erfordern und die Fähigkeit mit *PositionSensing* deutlicher spezifizieren könnte. Das heißt für die Kompatibilität, dass eine Komponente vom typ *ObjectDetection* durch eine spezialisierte Komponente *Detect_object_in_image* ersetzt werden könnte, nicht aber andersherum. Dieses Konzept ist mit dem der Vererbung in Programmiersprachen vergleichbar. Ontologien bieten einen darüber hinausgehenden Vorteil der Inferenz. Mittels hinterlegten Axiomen (siehe Abschnitt 4.2.3) können etwa aus der Kombination von einzelnen Komponenten neue Erkenntnisse gewonnen und Typen aufeinander abgebildet werden, wenn es Abbildungsregeln für diese geben kann.

Neben der Kompatibilität über die Typen können Vorbedingungen über eine konkrete Modellierung in der Komponente abgebildet werden. Die Anforderungen werden dabei ebenfalls als Komponente beschrieben, dadurch können später die vorhandenen Komponenten leicht gegen die gewünschten Anforderungen geprüft werden. Die Komponenten *MyVision* vom Typ *Detect_Object_in_Image* könnte etwa die Anforderung eines *RGBD_Camera_Wrappers* haben. Durch die Festlegung eines Wertes von 30 für dessen FPS Eigenschaft werden nur Komponenten mit 30 oder mehr FPS zur Erfüllung dieser Eigenschaft zugelassen.

Das für den MDE Ansatz entwickelte Tooling nutzt diese Eigenschaften bei der Erstellung eines *Skills*, also der Verbindung von Komponenten zu einem kompletten Programmablauf oder einer größeren Fähigkeit sowohl zur Prüfung als auch zur Bestimmung geeigneter Komponenten. Werden zwei Komponenten in dem grafischen Editor verbunden, kann geprüft werden, ob die Interfaces die verbunden werden sollen, übereinstimmen und ob die verbundenen Komponenten die definierten Vorbedingungen erfüllen. Ist dies nicht der Fall, können durch die Definition auch Vorschläge generiert werden, welche Komponenten die fehlenden Anforderungen noch erfüllen könnten, indem ein Komponenten-Repository nach

den entsprechenden Anforderung durchsucht wird. Ein Problem bei dem Ersetzen von Komponenten durch andere ist vor allem die Konnektivität mit anderen Komponenten. Gerade wenn eine spezialisierte Komponenten genutzt wird, die mehr Interfaces beinhaltet als die zu Ersetzende, gibt es zunächst keine klaren Regeln, wie diese zu verbinden sind. Bisher wird diese Entscheidung dem Nutzer einer grafischen Oberfläche überlassen.

Der für diese Arbeit gewählte **Fähigkeitsbasierte Ansatz** definiert ein deutlich vereinfachtes Modell (siehe 4.2.3), welches sich vor allem auf die Schnittstellen der Fähigkeit konzentriert und Fähigkeiten ansonsten primär über ihren Namen identifiziert. Der primäre *Austausch* einer Fähigkeit erfolgt daher über das System der Implementierungen (siehe Def. 28). Die in der Mission definierte Fähigkeit wird von jedem System individuell durch eine entsprechende Implementierung ersetzt, die nur auf dem jeweiligen System gültig ist. Da die Interfaces bereits durch das Modell der Fähigkeit selbst definiert wurden, muss dieses bei der Verwendung einer anderen Implementierung nicht geändert werden.

Eine zweite Option auch gänzlich andere Fähigkeiten zum Austausch zu nutzen, ist durch die Koordinatoren und *Meta-Capabilities* gegeben. Eine Fähigkeit wird in ihrer Implementierung durch einen Koordinator, also einen konkreten BT beschrieben, der meist Aufrufe an die *RAL* beinhaltet. Ein Koordinator der nur aus anderen *Capabilites* besteht, wird auch *Meta-Capability* genannt. Wenn im Team nun alle *Capabilities* dieser *Meta-Capability* vorhanden sind, können diese als zusammengesetzt Fähigkeit die ursprüngliche Fähigkeit ersetzen (Details in Abschnitt 4.5.7).

Vorbedingungen werden ebenfalls nicht als komplexe Einschränkungen, sondern auf der Basis von geforderten Fähigkeiten angegeben. Dafür können Vorbedingungen in der `ros_bt_py` (siehe Def. 31) auf zwei Arten angegeben werden. Eine normale Vorbedingung bedeutet, dass die angegebene Fähigkeit als bereits erfolgreich ausgeführter Knoten im Missions-Tree entweder lokal oder auf entfernten Systemen vorhanden sein muss. Speziell als *lokal* definierte Vorbindungen hingeben müssen immer direkt auf dem System ausgeführt worden sein, auf dem auch die relevante Capability ausgeführt werden soll. Dies ist vor allem dann relevant, wenn die Vorbedingungen physikalische Gegebenheiten erfordern. Zum Beispiel muss ein Roboter notwendigerweise vor dem Objekt stehen, wenn er dieses aufnehmen soll. Sind diese Lokal nicht erfüllt, so kann die Mission Control versuchen, diese in eine Sequenz vor die aktuelle Fähigkeit zu synthetisieren (Abbildung 4.33). Dafür wird eine Sequenz eingefügt und die Vorbedingung sowie die eigentliche Fähigkeit darunter gehängt.

In der aktuellen Implementierung wird lediglich eine Prüfung der Bedingung durchgeführt, da die Instanziierung Annahmen bezüglich ihrer Eingabewerte benötigt. Um sinnvolle lokalen Parameter zu erzeugen, ist jedoch deutlich mehr Vorwissen über die Capability erforderlich als üblicherweise vorhanden. Mögliche Ansätze wären eigene Inferenzsysteme, Ableitung von möglichen Parametern aus den Parametern der ausgelagerten Fähigkeit oder zusätzliche Modellinformationen in den Vorbedingungen selbst oder in der Mission zum Erstellungs-

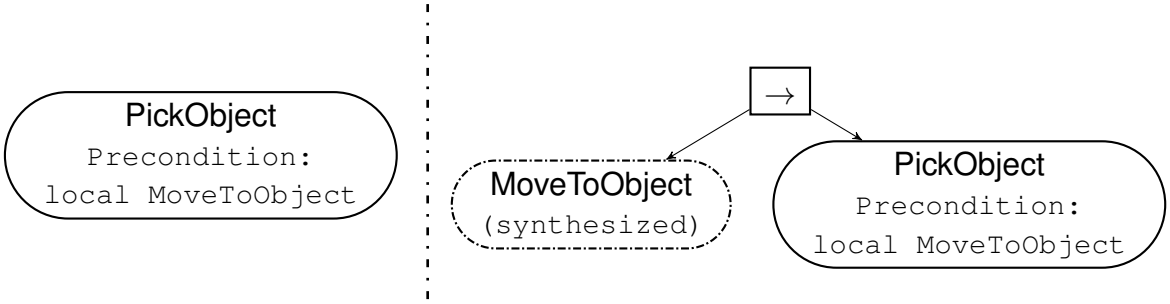


Abbildung 4.33.: Das PickObject erfordert als lokale Vorbedingung die Ausführung der Fähigkeit MoveToObject. Beim Instanziierten der Fähigkeit wird daher versucht, diese Abhängigkeit per synthetisierter Fähigkeit aufzulösen.

zeitpunkt. Da all diese Ansätze ein signifikantes Fehlerpotential bieten, wurde stattdessen zunächst eine Design-Philosophie eingesetzt, bei der vor allem möglichst in vollständige (sprich, ohne unaufgelöste Vorbedingungen) Fähigkeiten für den Austausch vorgesehen wurden und die Modellierung damit beim Bediener verbleibt.

4.5.6. Online Adaption

Eine dynamische Teamkomposition, spontan wechselnde Aufgaben und Flexibilität im Team sind zentrale Anforderungen, die für diese Arbeiten gestellt wurden.

```
@define_bt_node(NodeConfig(
    options={'passthrough_type': type},
    inputs={'in': OptionRef('passthrough_type')},
    outputs={'out': OptionRef('passthrough_type')},
    max_children=0))
class PassthroughNode(Leaf):
    """Pass through a piece of data

    Useful for testing, and to mark the inputs of
    a BT that is meant to be loaded as a subtree.
    """
    # implementation...
    def _do_shutdown(self):
         # State change implementation...
```

.

Abbildung 4.34.: Beispiel einer BT-Node Definition mit dem `@define_bt_node` Dekorator, in dem die Paramter definiert werden. Methoden für die jeweiligen Aktionen (Shutdown,Tick, etc.) können überschrieben weden. Quelle [BH22]

Die `ros_bt_py` Bibliothek implementiert die eingeführten BT-Begriffe und Funktionen und ermöglicht es insbesondere sehr einfach, eigene Knoten zu implementieren, indem die *Node*-Basisklasse oder deren Spezialisierung, die *Leaf*, abgeleitet wird (Abbildung 4.34). Die Basisklasse implementiert alle für den BT nötigen Funktionen wie das Verwalten der Zustandsübergänge (siehe 4.16), welche per abstrakter Funktionen für ein spezielles Verhalten überschrieben werden können. Über einen Dekorator können die Parameter (siehe Definition 13) für den Knoten gesetzt werden, was das manuelle Erstellen von eigenen Knoten sehr vereinfacht. Durch die bewusste Verwendung von Python als Programmiersprache können neue Klassen auch zur Laufzeit der Software nachgeladen und biem *Capability-Repository* angemeldet werden, sobald der BT jedoch gestartet wird, müssen konkrete Subklassen der *Node*-Klassen für die Erstellung des kompletten Baumes genutzt werden, was die dynamischen Aufgabenverteilung erschwert.

Das Konzept der *Shovables* und *Slots* ermöglicht die Verteilung der zusammengestellten Knoten auf ein entferntes System, jedoch zunächst auch nur während der Instanziierung des BT. Die Verwendung von *Subtrees* hat jedoch auch hier bereits erlaubt, die konkrete Implementierung zur Laufzeit durch das ausführende System zu laden, wodurch eine systemspezifische Implementierung genutzt werden konnte. In einer ersten Umsetzung vom Capability-Konzept wurden die *Subtrees* erweitert, so dass die Vergabe eine Auktion einsetzt und zudem *Data-Bridges* definiert, mit denen Parameter auch zur Laufzeit ausgetauscht werden können [HH18]. Das Ausführen des BT erforderte jedoch weiterhin das komplette Instanziieren jeder Capability, wodurch diese insbesondere zur Laufzeit nicht flexibel genug auf Änderungen der Teamkompositionen reagiert.

Die letztendlich umgesetzte *Capability* hat das initiale Design daher noch einmal um das Konzept der *Dynamischen Bindung* (siehe Def. 32) erweitert, wodurch Fähigkeiten und Implementierungen stärker voneinander getrennt und die Nutzung einer abstrakten *Capability-Node* auch zur Laufzeit des BT möglich ist, ohne dass die Implementierung bereits feststehen muss. Konkret wurde dafür eine *Capability*-Basisklasse umgesetzt, welche die *Node*-Klasse erweitert und die *Capability* spezifischen Übergänge und Optionen (siehe Abb. 4.24) ermöglicht. Würde nur die Basisklasse verwendet werden, könnten allerdings keine Inputs oder Outputs definiert werden, da die *Capability*-Klasse selbst keine Inputs besitzt. Würde andererseits für jede *Capability* eine eigene Klasse angelegt, schränkte dies die Flexiblität der Nutzung massiv ein. Stattdessen wird in `ros_bt_py` für jede Fähigkeit zur Laufzeit durch Ausnutzen der Metaprogrammierungsmöglichkeiten von Python eine dynamische Subklasse im Speicher erzeugt, die dann bei der *Capability-Repository* angemeldet wird und wie jeder andere Knoten sofort zur Laufzeit verwendet werden kann. Das dynamische Erstellen einer Subklasse im Speicher des Systems ist vor allem deshalb wichtig, da die Fähigkeiten anderer Systeme zur Laufzeit über das *Capability Interface Exchange Protocoll* ausgetauscht werden können und weil die Verwendung der *Capabilites* im Editor definierte Parameter erfordert, was mit reinen Subtrees nicht möglich ist. Bei jedem Update des *Capability-Repository* wird die Erstellung der *Capability*-Klasse und ihrer *CapabilityIOBridges* ausgelöst, so dass alle im Team bekannten Fähigkeiten sofort für

die anderen Systeme oder einen Menschlichen Nutzer verfügbar sind. Das lokale Erstellen des Interfaces im Speicher spart im Vergleich zum kompletten Übertragen der serialisierten Klasse Kommunikationsaufwand und legt außerdem den Fokus auf das Modell, so dass es einfacher ist, Fähigkeiten bei Veränderungen aktuell zu halten.

Jede Fähigkeit kann dadurch lokal derart verwendet werden, als wäre die komplette Implementierung auf dem System und lediglich durch einen Knoten zusammengefasst. Wird die Fähigkeit dann zu Laufzeit getickt, ohne dass der Ausführende Baum etwas davon merkt, wird die *Task Allocation* mit der Zuweisung einer konkreten Implementierung beauftragt. Durch die Ermittlung der Fähigkeitskosten (siehe 4.5.3) und ggf. einer darauf basierender Auktion (siehe 4.5.4) wird dann abhängig vom aktuellen Weltzustand die beste Implementierung gewählt. Die Zuordnung ist dabei nur so lang aktiv, wie auch die Fähigkeit ausgeführt wird. Wird diese beendet und zu einem späteren Zeitpunkt erneut getickt, erfolgt eine neue Vergabe, die wiederum die dann aktuellen Gegebenheiten berücksichtigt und damit ggf. anders entscheidet. Durch die erneute Auktion bei veränderten Bedingungen kann diese erneute Zuordnung auch während der Ausführung erfolgen und passt sich damit den Fähigkeiten und Umständen des aktuellen Teams an

Die Trennung der Implementierung vom ausführenden BT führt dazu, dass sich die Behandlung einer lokal ausgeführten Fähigkeit von einer im *Remote Capability Slot* kaum unterscheidet. In beiden Fällen wir ein eigener *Tree Manager*, also eine eigene BT-Instanz, angelegt, in der die Fähigkeit ausgeführt wird. Um diese Instanz mit dem ursprünglichen BT zu verknüpfen, muss die Implementierung die *CapabilityIOBridges* zusammen mit den restlichen Knoten in einer Sequenz nutzen (Abbildung 4.35). Zwischen Input und Fähigkeit kann an dieser Stelle zudem die Vorbedingung synthetisiert werden. Sobald die Fähigkeit einen *succeeded* oder *failed* Zustand erreicht hat, wird die Ausführungsumgebung gestoppt und wieder beseitigt. Der Hauptunterschied zwischen der lokalen oder der entfernten Ausführung ist, dass im lokalen Fall der Tick des Haupt-BT weiter geleitet wird, während die entfernte Ausführung einen eigenständigen Tick verwendet.

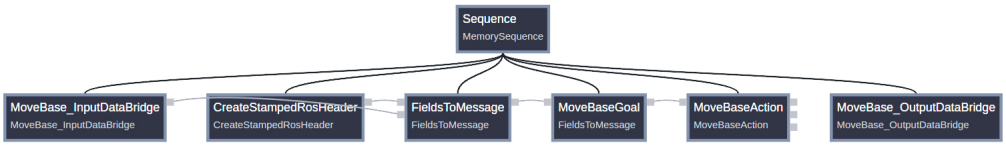


Abbildung 4.35.: Beispielimplementierung der MoveBase Capability. Die *InputDataBridge* und die *OutputDataBridge* werden automatisch für eine Capability erstellt und können für die Implementierung genutzt werden. Quelle [OH22]

Durch den Austausch über das *Capability Interface Exchange Protocoll*, die dynamische Erzeugung von *Capability*-Klassen und *IOBridges*, können Fähigkeiten schnell

im Team bekannt gemacht und für konkrete Implementierungen oder als abstrakte Klasse genutzt werden. Durch den Black-Box-Ansatz, bei dem die Fähigkeiten erst zur Laufzeit ausgewählt werden, muss die Mission nicht verändert werden, um mit einer geänderten Teamkomposition oder in geänderte Bedingungen ausgeführt zu werden. Die Mission passt sich durch die Online-Adaption dynamisch an die aktuelle Situation an. Das system geht von kooperativen Teilnehmern ohne schlechte Absichten aus. Sicherheitsaspekte im sinne von Security, werden in der aktuellen Umsetzung komplett außen vorgelassen. Durch das nutzen von Public-Key-Verfahren zum signieren der Nachrichten oder spezieller Prüfsummen der Fähigkeitsdefinition könnte bereits ein relevantes maß an Sicherheit erzielt werden.

4.5.7. Kombinierte Fähigkeiten

Eine Mission besteht aus einer Menge von Fähigkeiten, die durch einen BT geordnet ausgeführt werden. Jede Fähigkeit wiederum besteht aus einem *Koordinator*, ebenfalls in Form eines BT, der die eigentliche *Implementierung* über seine Knoten anspricht. Da Capabilities als normale Knoten in einem BT verwendet werden, kann eine Fähigkeit auch aus beliebigen weiteren Fähigkeiten aufgebaut sein, ein Koordinator, der komplett aus Fähigkeits-Knoten aufgebaut ist wird dabei als *Meta-Capability* bezeichnet. Es gibt keinen Unterschied in der Behandlung einer *Meta-Capability*, allerdings kann diese auch als Anweisung oder Rezept verstanden werden. Dadurch kann eine Mission also in immer kleiner werdende Fähigkeiten zerlegt werden, bis nur noch konkrete Fähigkeiten mit Knoten, die mit der *RAL* kommunizieren, verbleiben. Durch die vorgestellten Möglichkeiten, die Fähigkeiten zu jedem Zeitpunkt auch von anderen Systemen übernehmen zu lassen, können durch diese Art der Modellierung beliebig kleine Teile einer Aufgabe übertragen und dadurch auch Systeme mit sehr spezialisierten Fähigkeiten berücksichtigt werden. Ebenso muss eine Aufgabe nicht vollständig von einem System erfüllt werden, es könnte auch lediglich einen Teil der Fähigkeit besitzen, aber wissen, wie man die vollständige geforderte Fähigkeit zerlegt und dadurch andere Systeme zur Hilfe holen, um die Aufgabe doch noch zu erfüllen. Zuletzt können auch Systeme berücksichtigt werden, die überhaupt nicht für die Kooperation vorgesehen waren, aber offene Schnittstellen aufweisen. Die verschiedenen Optionen zur Auf- und Verteilung der Fähigkeiten sollen im Folgenden anhand eines konkreten Beispiels erklärt werden. Gegeben sind 3 Robotersysteme:

Roboter 1 ist ein simpler mobiler Roboter mit guten Mobilitätseigenschaften, einem Lokalisierungsystem und einer Kamera, der für die Suche nach Golfbällen genutzt wird. Er verfügt insbesondere über die Fähigkeiten zum Explorieren (*Explore)* und Erkennen der Golfbälle mit seiner Kamera (*LocateBalls*) (Abbildung 4.36). Er besitzt außerdem zwei Meta-Capabilites: *FindBalls* beschreibt die Strategie für das Absuchen und Auffinden der Golfbälle, *CollectBalls* gibt auf einem hohen Niveau an, wie Bälle eingesammelt werden können.

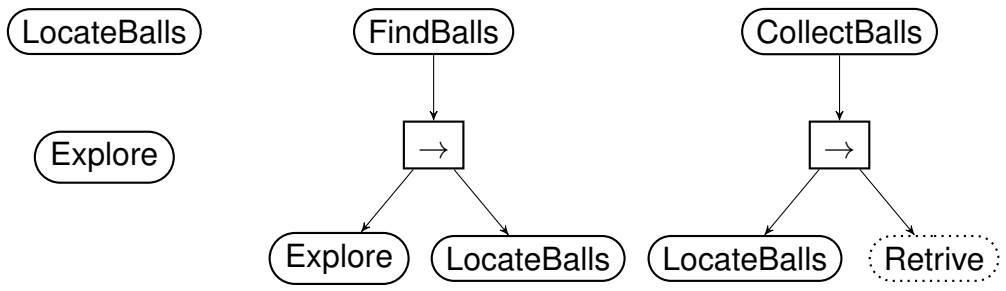


Abbildung 4.36.: Fähigkeiten des Roboter 1 als Beispiel für die Kombination von Fähigkeiten. Ellipsen = Fähigkeiten, gepunktete Ellipse = Remote-Fähigkeit, die nicht auf dem System vorhanden ist. Der einfache Roboter kann die Position von Golfbällen finden (*LocateBalls*), die Umgebung absuchen (*Explore*) und verfügt außerdem über 2 Meta-Fähigkeiten, die beschreiben, wie Bälle gefunden (*FindBalls*) oder aufgesammelt (*CollectBalls*) werden können. Die Fähigkeiten sind stark vereinfacht und nicht abschließend dargestellt.

Roboter 2 ist ein deutlich langsameres System, welches mit einem Arm ausgestattet ist und Müll entlang eines vordefinierten Pfades einsammelt (*CollectTrash*). Wie Roboter 1 kann das System seine Kamera nutzen, um die Position von Müll zu erkennen (*LocateTrash*) und zusätzlich seinen Arm nutzen, um Gegenstände aufzunehmen und an eine Ablageposition zu bringen (*Retrive*) (Abbildung 4.37).

Roboter 3 ist ein schneller Flugroboter, der lediglich über die Fähigkeit zur Exploration verfügt.

Der erste Fall betrachtet eine **Mission mit nur einem Roboter** (Abbildung 4.38 links). Die Mission besteht aus zwei High-Level-Anweisungen: Sammel Bälle ein (*CollectBalls*) und finde Bälle in der Umgebung (*FindBalls*), falls das Einsammeln nicht möglich ist. Sobald der Missions-BT getickt wird, fordert die *Mission Control* von der *Task Allocation* eine geeignete Implementierung für *CollectBalls*. Diese lädt die Implementierung von *CollectBalls* und überprüft wiederum die Kosten für die darin enthaltenen Knoten, wofür deren Implementierungen geladen werden. Während für *LocateBalls* lokale Kosten ermittelt werden können gibt es keine Implementierung für *Retrieve*. Alle Fähigkeiten werden zudem auch zur Auktion ausgeschrieben. Da es keinen anderen Roboter gibt, schlägt die Auktion fehlt, die Kosten für die Ausführung werden als ✗, also nicht ausführbar, ermittelt, woraufhin die komplette Fähigkeit fehlschlägt und der Knoten in den *failed* Zustand wechselt. Der Vorgang für die zweite Fähigkeit *FindBalls* ist analog dazu. Die lokale Implementierung von *FindBalls* wird ausgewertet und die Fähigkeit zur

Robot 2 Capabilites

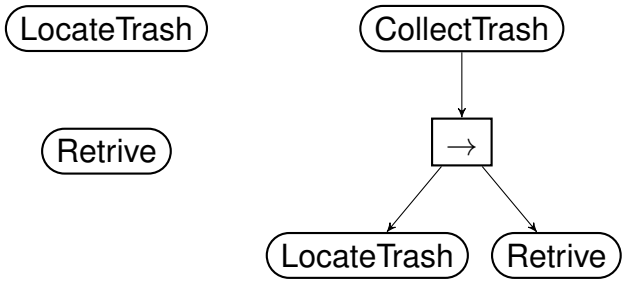


Abbildung 4.37.: Fähigkeiten des Roboter 2 als Beispiel für die Kombination von Fähigkeiten. Ellipsen = Fähigkeiten. Das System kann die Position von Müll bestimmen (*LocateTrash*) und diesen mit seinem Arm einsammeln (*Retrieve*). Beide Fähigkeiten werden für das Einsammeln von Müll (*CollectTrash*) genutzt. Roboter 2 folgt vorgegebenen Wegpunkten und kann daher nicht wie die anderen Systeme die Umgebung explorieren.Die Fähigkeiten sind stark vereinfacht und nicht abschließend dargestellt.

Auktion angeboten. Da es nur die lokale Implementierung gibt, wird diese geladen und der Vorgang für die darin enthaltenen Fähigkeiten wiederholt. Da der Roboter über alle nötigen Fähigkeiten für *FindBalls* verfügt, kann er diese Mission ausführen, wobei niemals Bälle aktiv aufgesammelt werden (Abbildung 4.38 rechts).

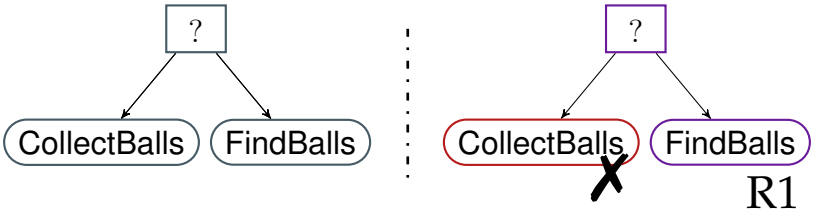


Abbildung 4.38.: Die Mission (links) sieht vor, dass der Roboter vorhandene Bälle einsammeln (*CollectBalls*) soll und, falls das nicht möglich ist, zumindest alle Bälle finden (*FindBalls*) soll. Wird die Mission dem Roboter 1 zugewiesen (rechts), kann dieser nur den *FindBalls* Teil der Mission ausführen.

Für den zweiten Fall (Abbildung 4.39), eine **Mission mit zwei Robotern**, wird das Team um den Roboter 3 erweitert. Wie im ersten Fall wird geprüft, welche Implementierungen für die Teile der Mission gefunden werden können. Da weiterhin kein Roboter über die *Retrieve* Funktion verfügt oder diese aus anderen Fähigkeiten bilden kann, wird die *CollectBalls* Fähigkeit weiterhin als nicht ausführbar ausgewertet und nicht weiter betrachtet. Die *Explore*-Fähigkeit hingegen ist sowohl lokal als auch auf Roboter 3 verfügbar. Da der wendige Flugroboter speziell für diese Aufgabe entwickelt wurde, ergibt sich eine deutlich kleinere

Utility als für Roboter 1, die Fähigkeit wird also an diesen vergeben. Sollte Roboter 3 jedoch ausfallen, etwa weil die kurze Akku-Laufzeit von UAV dazu führt, dass dieses nur kurzfristig verfügbar ist, wird die *Mission Control* dies durch den regelmäßigen Ping registrieren und eine erneute Vergabe prüfen. In diesem Fall würde die Aufgabe zurück an Roboter 1 geben, der die Aufgabe dann allein weiterführt. Da eine solche Neubewertung bei jeder Änderung des Teams durchgeführt wird können dadurch auch nur kurzfristig verfügbare Roboter opportunistisch in die aktuelle Mission eingebunden werden (Abbildung 4.39 rechts).

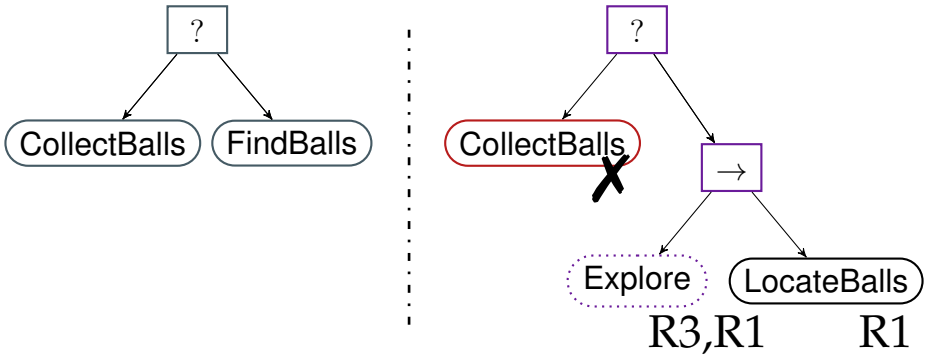


Abbildung 4.39.: Beispielmission mit dem Team aus Roboter 1 und Roboter 3. Die *CollectBalls* Fähigkeit kann weiterhin nicht ausgeführt werden, da kein Roboter über *Retrieve* verfügt. Die *Explore* Fähigkeit kann jedoch an Roboter 3 oder Roboter 1 vergeben werden. Da Roboter 3 eine signifikant neue Implementierung (abfliegen der Umgebung) mit deutlich reduzierten Kosten einbringt, werden die Fähigkeiten des Teams damit erweitert bzw. verbessert und die Redundanz erhöht.

Im dritten Fall (Abbildung 4.40) wird die gleiche Aufgabe als **Mission mit drei Robotern** realisiert. Wie bereits im Fall 1 und 2 wird die *FindBalls* Fähigkeit aufgelöst und durch eine Kombination aus Roboter 3 und Roboter 1 bearbeitet. Bisher konnte Roboter 1 die Fähigkeit *CollectBalls* nicht ausführen und auch an kein anderes System auslagern. Durch das Einbringen der *Retrieve* Fähigkeit durch den Roboter 2 kann der Roboter 1 diese Fähigkeit mit seiner eigenen *LocateBalls* Fähigkeit kombinieren. Das Team erlangt damit also eine *Kombinierte Fähigkeit* die sich implizit auf die zwei Systeme verteilt. Relevant ist zudem, das durch die Datenkanten die von Roboter 1 erkannte Pose für die *Retrieve*-Fähigkeit auf Roboter 2 genutzt wird. Da Roboter 2 über keinen Algorithmus zum Erkennen von Bällen verfügt, könnte Roboter 2 diese Tätigkeit also niemals komplett allein ausführen.

Das Kombinieren bzw. Verteilen der Fähigkeiten lagert diese zunächst jedoch nur an die unterschiedlichen Systeme aus. Die Flusskontrollknoten werden nicht geändert, so dass sicher gestellt ist, dass die Mission genau so ausgeführt wird, wie sie ursprünglich gegeben wurde. Während dadurch Ressourcenkonflikte vermieden und Abhängigkeiten wie das Greifen an einer Pose, die zuvor mittels *Locate*

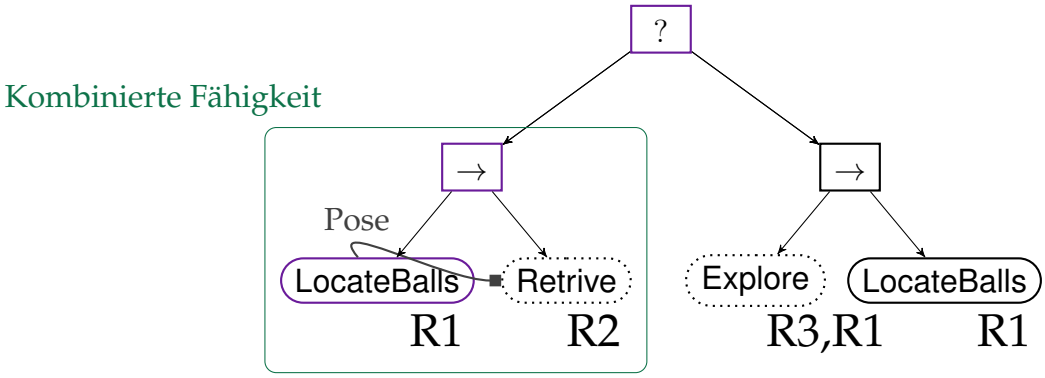


Abbildung 4.40.: Beispielmission mit dem Team aus Roboter 1, Roboter 2 und Roboter 3. Durch Roboter 2 kann die Fähigkeit *CollectBalls* als *Kombinierte Fähigkeit* ausgeführt werden. Während Roboter 1 die *LocateBalls* Fähigkeit übernimmt und damit die Pose zum Aufnehmen der Bälle ermittelt, wird das Aufnehmen (*Retrieve*) von Roboter 2 ausgeführt, der allein nicht dazu in der Lage wäre.

detektiert wurde, beibehalten werden, wird dadurch keinerlei Potential der Parallelität ausgenutzt. Roboter 1 könnte beispielsweise bereits weiter die Positionen der Bälle ermitteln, während Roboter 3 das Gebiet exploriert. Um diese Optionen auszunutzen, kann daher die ursprüngliche Mission überarbeitet werden. Abbildung 4.41, zeigt wie eine **parallelisierte Mission** erstellt werden kann. In diesem

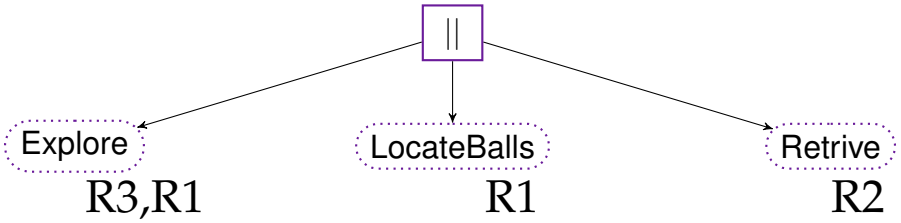


Abbildung 4.41.: Parallele Umsetzung der Beispielmission. Alle Roboter nehmen eine reine Folgerrrolle ein und erhalten durch die parallele Ausführung stets die aktuell für sie am besten geeignete Fähigkeit. Dies setzt allerdings eine stärkere Entkopplung und Prüfung innerhalb der Fähigkeiten voraus.

Fall würde jeder der Roboter nichts anderes als sein *Remote Capability Slot* ausführen. Eine zentrale Instanz, oder einer der Roboter, übernimmt außerdem die Rolle der Missionssteuerung und vergibt die parallelen Aufgaben. Ressourcenkonflikte und Abhängigkeiten müssen bei diesem Ansatz stärker in die individuelle Fähigkeit modelliert werden, was jedoch ohnehin ein Design-Grundsatz (siehe 4.2.1) ist. Wenn ein *Retrieve* beispielsweise immer nur dann aktiv ist, wenn ein greifbares Objekt vorhanden ist, würde eine Zuweisung an Roboter 2 immer nur dann erfolgen, wenn diese Voraussetzungen erfüllt sind. Es wäre grundsätzlich möglich, eine Mission auch automatisch auf mehr Parallelität zu optimieren,

dies erfordert aber zusätzliche Abwägungen oder Modellierungen.

Roboter 1-3 wurden in den bisherigen Fällen explizit modelliert und haben ihre Fähigkeiten über das *Capability Exchange Protocol* ausgetauscht. Mit dem vorgestellten System ist es aber auch möglich, **Nicht-Autonome Systeme in der Mission** zur verwenden. Nicht-Autonom bedeutet in diesem Fall primär, dass die Systeme nicht in der Lage sind, Nutzer-Code auszuführen und dadurch nicht ohne Weiteres in ein Roboterteam integriert werden können. Die kann zum einen der Fall sein, wenn ein System nur über wenig Autonomie verfügt, etwa wenn es komplett ferngesteuert wird und der Roboter selbst lediglich über Aktorik und Sensorik ohne Rechenleistung verfügt oder zum anderen, wenn das System proprietär ausgelegt ist und die Ausführung von eigenem Code nicht gestattet. Es können zwei Ansätze genutzt werden, um diese Systeme für das Team nutzbar zu machen: Der Host-System kann ein komplette zweite Framework-Instanz in einem anderen Namespace starten, welche im Namen des zu kontrollierenden Systems agiert, Fähigkeiten bereitstellt und Gebote abgibt. Die Fähigkeiten selbst interagieren mit dem Zielsystem, indem für die RAL die Proprietären API-Funktionen aufgerufen werden. Die Fähigkeiten selbst können jedoch insbesondere durch das Modellieren von Vorbedingungen und statischen Utilities auch direkt vom Host-System genutzt werden. Eine *ExploreWithUAV* Implementierung für die *Explore* Fähigkeit könnte etwa die proprietären Schnittstellen der jeweiligen API ansteuern, um das UAV zu kontrollieren. Die *CalculateUtility*-Funktion würde wiederum prüfen, ob die erforderliche API verfügbar ist oder nicht. Die spezifische Implementierung wird also nur genau dann als Möglichkeit für die *Task Allocation* angeboten, wenn ein Treiber auf dem Host-System geladen wurde und das zu steuernde System wirklich verfügbar ist. Es wäre sogar möglich, noch einen Schritt weiter zu gehen und als Vorbedingung eine Fähigkeit zu definieren, welche den eigentlichen Treiber lokal startet. Diese Fähigkeit wiederum kann in ihrer Utility-Funktion die Existenz der Hardware (z.b. durch einen Ping oder das Prüfen einer Log-Datei) sicherstellen. In frühen Experimenten wurde diese Art der Kooperation zwischen LAURON und einem UAV umgesetzt [Heppner et al., 2013]. Durch LAURON wurde der Treiber für das UAV gestartet aber auch die Lokalisierungsfähigkeit übernommen, da das UAV nicht über akkurate Onboard-Sensorik verfügte. Im Gegenzug konnte LAURON dadurch seine Sensorreichweite und Positionierung deutlich verbessern.

4.6. Zusammenfassung und Fazit Fähigkeitbasierte Kooperation

In diesem Kapitel wurden die Kernkonzepte für die fähigkeitsbasierte Kooperation von Roboterteams vorgestellt und erläutert. Nach dem Überblick über das gesamte Konzept in Abschnitt 4.1 wurde in Abschnitt 4.2 dargestellt wie eine Fähigkeit modelliert und aufgebaut werden sollte. Durch die Verwendung eines

Koordinators (4.2.2) können Implementierungsdetails (4.2.1) gekapselt und dadurch leichter wiederverwendet werden, in Kombination mit einer leichtgewichtigen Modellierung (4.2.3) ergeben sich damit leicht austausch- und kombinierbare Funktionseinheiten als Grundlage für die Mission einzelner Roboter, aber auch von Roboterteams.

Im Abschnitt 4.3 wurde das Multi Roboter Framework zur Verwendung und Koordination dieser Fähigkeitsbausteine erklärt. Echte Hardware wird auf den Robotern durch die *Robot Layer* (4.3.1) angesprochen, welche insbesondere sicherstellt, dass alle weiteren Funktionen mit ROS arbeiten können. Die *Skill Layer* (4.3.2) stellt die Fähigkeiten und insbesondere ihre Koordinatoren und Modelle bereit und erlaubt das sammeln und Austauschen der Fähigkeiten. Die *Mission Layer* (4.3.3) implementiert alle nötigen Kernfunktionen zur Nutzung im Team wie etwa das *Capability-Repository* oder die *Task Allocation*. Das Framework orientiert sich an den im State of the Art untersuchten Frameworks (3.2) und Projekten (3.3) legt den Fokus jedoch vor allem auf eine leichtgewichtige Umsetzung ohne zu starke Beschränkungen, welche die Kernkompetenz vor allem in Kombination mit den Fähigkeit realisieren kann.

Das wohl wichtigste unterliegende Konzept dieser Arbeit sind die in Abschnitt 4.4 erläuterten Behavior Trees. Konzepte wie asynchrone Prozesse (4.4.2), Parametrisierung (4.4.3) und native ROS Integration richten die Modellierung explizit auf die Robotik aus, zusätzliche Funktionen wie die Utility Berechnung (4.4.4), Shoving und Slots (4.4.5) erweitern den Ansatz auf Multiroboter Systeme. In der Modellierung des BT wurde das Fähigkeits-Konzept durch die Capabilities (4.4.6) und die Remote Capability Slots (4.4.8) explizit berücksichtigt, wodurch eine einheitliche Verwendung von BTs ermöglicht wird. Dieses BT-Konzept erweitert den aktuellen State of the Art um Features für das einfache Nutzen von Fähigkeiten und deren transparente Verteilung im Team. Es wurde in [Heppner et al., 2023] veröffentlicht, eine Veröffentlichung über die Fähigkeiten ist aktuell im Review [Heppner et al., 2024].

Alle Konzepte können dann, wie in Abschnitt 4.5 gezeigt, integriert werden, um Robotermissionen zu modellieren und auszuführen, wobei eine implizite Aufteilung der Fähigkeiten realisiert werden kann. Eine Mission kann mit dem gegebenen Tooling einfach erstellt werden (4.5.1) und nutzt dafür die *Capability Repositories*, (4.5.2) um Fähigkeiten aus dem kompletten Team zu berücksichtigen. Um zu bestimmen, wer eine Fähigkeit ausführt, wurden verschiedene Ansätze für die Bestimmung der Fähigkeitskosten (4.5.3) aufgezeigt, unterschiedliche Modelle für die Prüfung der Kompatibilität zwischen den Fähigkeiten getestet (4.5.5) und ein Auktionsystem (4.5.4) realisiert. Es wurden verschiedene Ansätze für die einfache Verteilung über die gezielte Kombination der Fähigkeiten bis hin zur Einbindung neuer Roboter in das Team aufgezeigt (4.5.7) und damit eine stabile und flexible Basis für die fähigkeitsbasierten Kooperation im Roboterteam geschaffen.

Im nächsten Kapitel wird evaluiert, ob die gezeigten Werkzeuge und Konzepte auch unter realen Bedingungen für den Einsatz im Roboterteam geeignet sind.

5. Experimente und Evaluation

Nachdem im Kapitel 4 die Kernkonzepte dieser Arbeit erläutert wurden, sollen hier die Erkenntnisse aus der Umsetzung der Konzepte aufgezeigt und die Anwendbarkeit der entwickelten Software belegt werden. Dafür werden mehrere zentrale Experimente bzw. Anwendungen als Evaluation der Konzepte betrachtet: Abschnitt 5.1 zeigt die Entwicklungen für den Einsatz autonomer Systeme für die Exploration fremder Planeten. Diese Experimente haben nicht nur wichtige Entscheidungen geprägt, sondern evaluieren nahezu alle Teile der Entwicklungen unter realen, schwierigen Bedingungen. Im Abschnitt 5.2 wird die Anwendbarkeit in einer weiteren Domäne evaluiert und die aufgezeigten Konzepte zur Modellierung und Kapselung von Fähigkeiten mit einem konkreten Use-Case aus der Industrie validiert. Der Abschnitt 5.3 belegt mit einer Simulation die Anwendbarkeit von *Behavior Trees* und insbesondere *Shovables* für die verteilte Ausführung von Missionen mit einem Multi-Roboter-Team und gibt Beispiele für die Berechnung von *Utility*-Werten und deren Verwendung für die Fähigkeitsauswahl. Zuletzt zeigt eine umfangreiche Simulation im Abschnitt 5.4 die Anwendung des Fähigkeitskonzeptes mit BTs inklusive den *Capability*-Knoten und der der Auktionierung im Team. In jedem Abschnitt werden zuerst die experimentelle Durchführung und dabei erzielte Ergebnisse vorgestellt, bevor dann die Ergebnisse diskutiert und schließlich deren Bedeutung für diese Arbeit herausgestellt werden.

5.1. Fähigkeitsbasierte Exploration fremder Planeten und Mission Control

Die initialen Versionen des Frameworks und Konzepte wie Fähigkeiten wurden im Rahmen des SpaceBot Cup [Heppner et al., 2015] [98], einem Robotik-Wettbewerb des DLR, bei dem die Teilnehmer möglichst autonome Lösungen für die planetare Exploration einbringen sollten, umgesetzt und erprobt. Mit dem SpaceBot Camp [Heppner et al., 2017] wurde der Wettbewerb zwei Jahre später noch einmal wiederholt, wofür vor allem das *Framework* und die *Mission Control* und verbundene Themen verbessert wurden. Mit der ESA Space Ressource Challenge [99] wurden insbesondere die BTs erneut in einem Weltraumkontext evaluiert. Im Folgenden sollen die drei Experimente und die danach erfolgten Weiterentwicklungen der Konzepte vorgestellt und evaluiert werden.

Der **SpaceBot Cup** [98] [Heppner et al., 2015] war der erste Robotik-Wettbewerb des DLR mit dem Ziel, insbesondere die Autonomie von mobilen Robotern zu steigern und das Potential von bereits existierenden Robotik-Lösungen für die planetare Exploration zu identifizieren. Aufgabe des Wettbewerbs war es, eine für die Teilnehmer unbekannte Umgebung (Abbildung 5.1 links) zu erforschen, die einem fremden Planeten nachempfundene war. Dabei müssen verschiedene Gegenstände (Wasserglas, Batterie, Waage) gefunden, aufgenommen und zusammengesetzt werden (Abbildung 5.2 rechts). Die Ergebnisse der Exploration sollen in einem Report gesammelt werden. Insbesondere muss der Roboter zur Bewältigung dieser Herausforderungen autonom in der Lage sein, da ein Eingriff der Bodenstation nur in kurzen Zeitfenstern und mit massiver Verzögerung in der Kommunikation möglich ist.

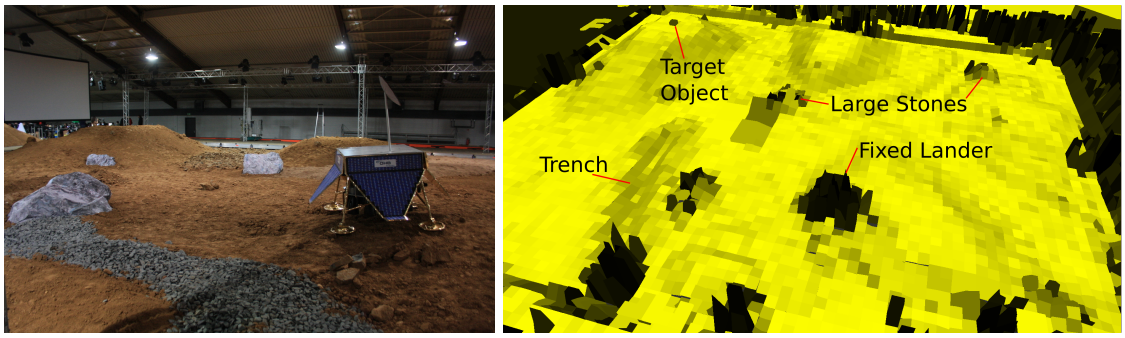


Abbildung 5.1.: Links: Wettbewerbs-Arena mit verschiedenen Hindernissen wie einem Graben, großen Steinen als Hindernisse, verschiedenen Untergründen, Steigungen und einem feststehenden Lander-Modul. Quelle [Heppner et al., 2015]. Rechts: Die durch LAURON V erstellte 2.5D-Karte der Umgebung. Quelle: [Heppner et al., 2015]

Der Roboter LAURON V (Abbildung 5.2 links) des FZI wurde für den Wettbewerb mit auf die Aufgaben abgestimmten Sensoren und Zusätzen ausgestattet. Ein speziell konstruierte Greifer am Vorderbein [Heppner et al., 2014] erlaubt es zusammen mit mobilen Manipulationsfähigkeiten die gesuchten Gegenstände aufzunehmen und in den ebenfalls speziell konstruierten Halterungen zu platzieren. Mittels des KaRoLa [100] Sensors können hochaufgelöste 3D-Punktwolken erstellt und daraus eine Umgebungskarte (Abbildung 5.1 rechts) im dafür entwickelten 2.5D-Kartenformat PlexMap [Oberländer et al., 2014] berechnet werden. Das Kartenformat kann neben der Auswertung für die Bodenstation insbesondere auch zur Pfadplanung unter Berücksichtigung von Präferenzen und der individuellen Kostenberechnung (vergleiche 4.5.3) verwendet werden. Der schwenkbare Kamerakopf ist mit RGB und RGB-D Kameras ausgestattet, um Hindernisse und die gesuchten Objekte in der Umgebung zu erkennen.

Um die Ziele des Wettbewerbs zu erreichen, wurden eine Vielzahl von Fähigkeiten implementiert, wobei die Fähigkeiten zwar bereits nach Fähigkeiten gruppiert wurden, sie allerdings noch keinen Koordinator oder separate Modellie-

Abbildung 5.2.: Links: LAURON V Roboter beim SpaceBot Cup. Die Beine sind für das Laufen auf weichem Sand optimiert und mit einem speziellen Greifer für die Mission ausgestattet. Am Körper wurden spezielle Halterungen für die Zielobjekte angebracht und als Sensorik ein KaRoLa für hochauflösende 360° Laserscans verwendet. Rechts: Die zu findenden und aufzunehmenden Zielobjekte Batterie (gelb), Waage (rot) und Probenbehälter (blau) und die gewünschte Konfiguration zum Missionsende. Quellen: [Heppner et al., 2015]

rungsinformation hatten. Zu den entwickelten Fähigkeiten gehören unter anderem die im Kapitel 4.3.2 beschriebenen *Fine Localize Object*, um die exakte Orientierung der gesuchten Objekte zu bestimmen oder auch das *Update-Environment-Modell*. Zudem wurden Fähigkeiten für die Navigation, insbesondere das Kartographieren und Planen über das schwierige Gelände, Einschätzen und Auswerten der Umgebung und natürlich der Manipulation der Objekte entwickelt. Eine Video einiger Fähigkeiten kann unter [HI+15] eingesehen werden. Um die Fähigkeiten zu verwalten und in einer Missionsstrategie zu verwenden, wurde ein Framwork entwickelt (Abbildungen 5.3), welches das Konzept der *RAL Layer* für die Entkopplung der Fähigkeiten von der Hardwareumsetzung vorsah und die High-Level Entscheidungen in der *Mission Layer* angesiedelt hat. Die modularen Fähigkeiten wurden in der Planning Layer (Abbildung 5.3 links) durch verschiedene ROS-Pakete bereitgestellt und dann direkt aus der Missionssteuerung (Abbildung 5.4) zentral aktiviert. Ebenfalls wurde hier das Konzept der globalen Datenhaltung eingeführt, welches bis heute beibehalten wurde. Alle erkannten Objekte oder Missionsziele werden in öffentlichen Topics veröffentlicht wo sich jede Komponente wiederum bedienen kann. Diese offene Struktur erleichtert den modularen Aufbau und bietet darüber hinaus einfache Visualisierungs- und Modifikationsmöglichkeiten der existierenden Daten.

Während die jeweiligen Entwicklungen, der Datenaustausch und der Roboter an sich korrekt funktionierten, war der Wettbewerbslauf ein Fehlschlag. Der eigentliche Lauf wurde durch kombinierte Fehler beim Deployment der Lösung, der Hardware und den verwendeten Failsafes verursacht wodurch der Roboter sich gar nicht erst vom Startpunkt wegbewegen konnte. Trotz Versuche der Bodenstation, das Problem zu beheben, konnte insbesondere durch die verzögerte Kommunikation und die nur kurzen Zeitfenster für den Zugriff das komple-

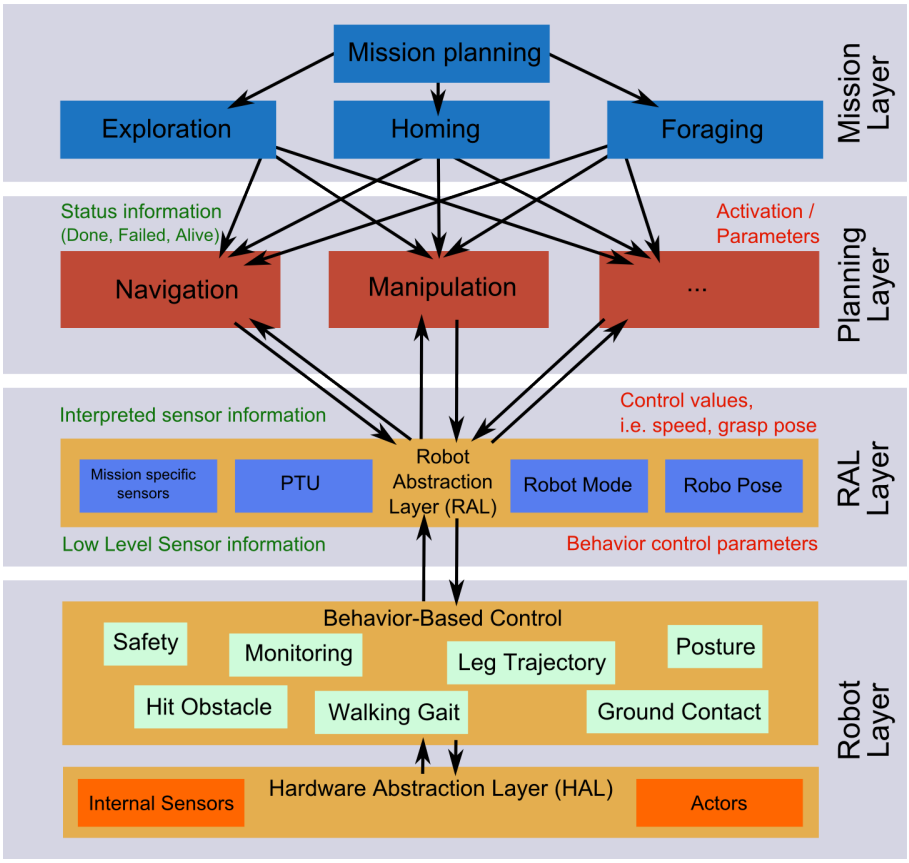


Abbildung 5.3.: Software Architektur auf LAURON V. Die Robot Layer implementiert die roboternahen Funktionen wie etwa das Laufen und abstrahiert den Zugriff auf diese über die Robot Abstraction Layer (RAL). Auf der Planning Layer sind die verschiedenen Fähigkeiten, die für die Mission notwendig sind, implementiert. Die Mission Layer gibt die zentrale Strategie vor und spricht die Fähigkeiten auf der Planning Layer an. Quelle: [H+13]

xe Problem nicht behoben werden. Gerade in den letzten Phasen der Entwicklung hat sich dabei die starke Bindung zwischen Missionskontrolle und den Fähigkeiten als problematisch erwiesen, da die Missionskontrolle zu viele Details der Umsetzungen berücksichtigen musste. Dadurch konnten die Modular entwickelten Fähigkeiten bei weitem nicht so modular verwendet werden, wie es geplant war und die Missionskontrolle wurde fehleranfällig. Durch die zu starke Bindung führten bereits einfache Fehler zum Abbruch der Gesamtmission. Das Beispiel wird auch deshalb hier genannt, da der aufgetretene Fehlerfall ein gängiges Problem in Robotikanwendungen illustriert, dessen subsequente Lösung in den weiteren Wettbewerben um so signifikanter ist. Trotz des Fehlschlags im Wettbewerb wurden durch die Entwicklungen wichtige Erkenntnisse gewonnen und grundlegende Framework-Konzepte evaluiert. Insbesondere die Aufteilung in Fähigkeitsgruppen, das Abstrahieren der jeweiligen Roboterschichten und die Koordination der Daten über globale Topics haben konzeptuell gut funktioniert.

Während der Vorbereitungen für den Wettbewerb konnten Systeme wie die Objekterkennung und die Manipulation darüber erfolgreich Daten austauschen und aufgrund der Kapselung in Funktionsgruppen effektiv unabhängig entwickelt werden. Vor der Einführung der *RAL* konnten nur geübte Anwender LAURON wirklich verwenden, da die komplexe verhaltensbasierte Steuerung viele Parameter und Erfahrungswerte für verschiedene Aufgaben erfordert. Durch die *RAL* wurde dieses Anwenderwissen abstrahiert, so dass auch fachfremde Entwickler, etwa für die Bilderkennung, in der Lage waren, den Roboter zu steuern und zu überwachen. Die gesammelten Erkenntnisse wurden für weitere Entwicklungen für weitere Wettbewerbe genutzt.

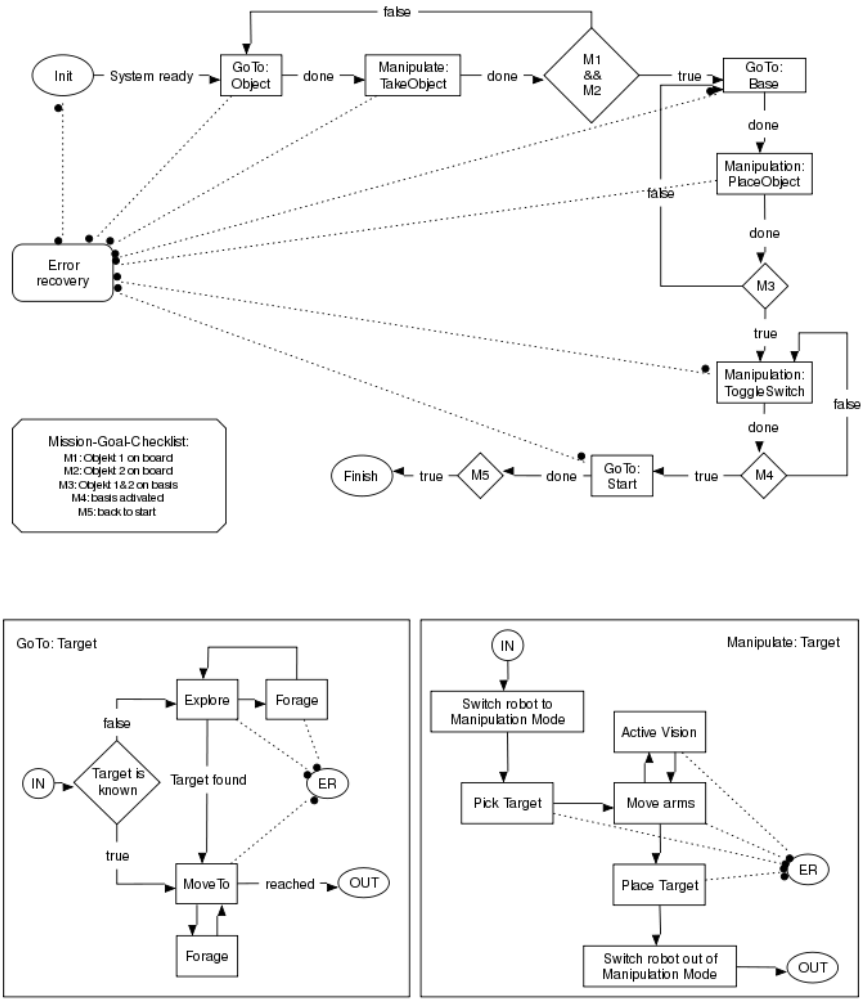


Abbildung 5.4.: Die High Level Missionssteuerung des SBC13 als Statemachine.Quelle: [H+13]

Beim **SpaceBot Camp** Wettbewerb [Heppner et al., 2015, Heppner et al., 2017] war die Aufgabe erneut, ein vorher unbekanntes Gelände (Abbildung 5.5) zu explorieren und dabei die schon aus dem vorherigen Wettbewerb bekannten Objekte zu finden und zusammenzusetzen. In dem einem erd-fernen Planeten nach-

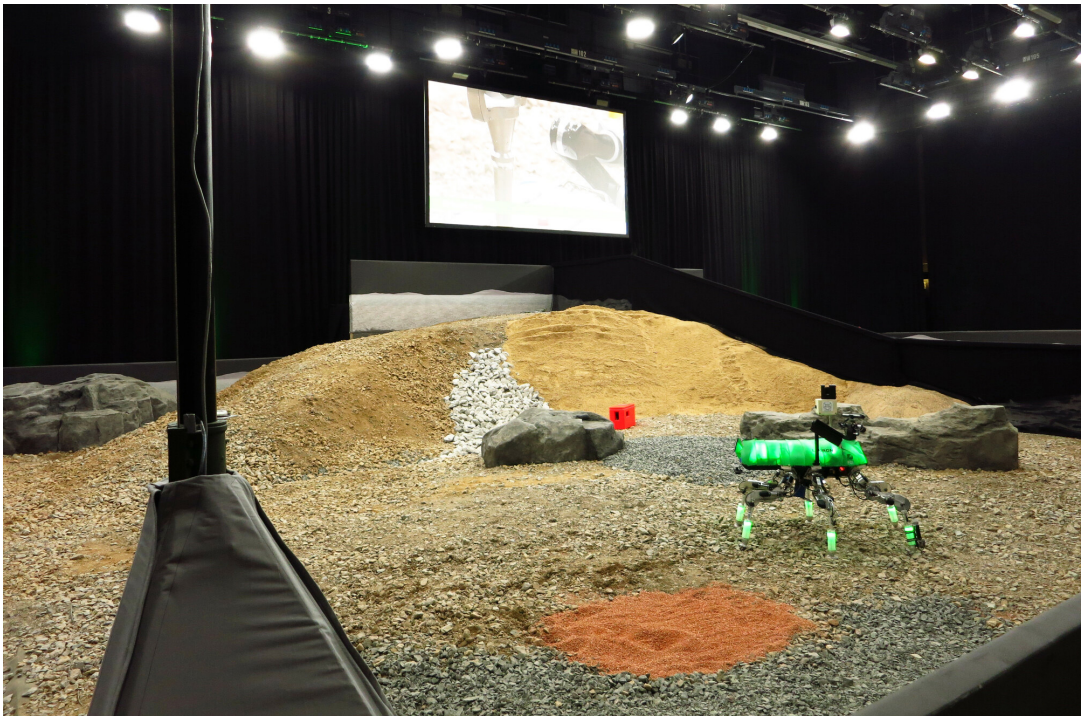


Abbildung 5.5.: Terrain des SpaceBot Camps. Das Gelände ist einem Krater auf einem fremden Planeten nachempfunden. Der Start befindet sich auf einem Hügel, von dem nur eine lange Rampe mit mäßiger Steigung hinabführt. Andere Routen beinhalten losen Sand und Geröllfelder. Der rote Sand ist feiner Treibsand. Die großen Steine stellen Hindernisse dar. Links im Bild ist das zweite Robotersystem der *Lander* zu sehen, rechts bewegt sich LAURON V in Richtung des Zielobjektes (roter Kasten).

empfundenen Gelände mussten vor allem über eine längere Steigung, verschiedene Untergründe und größere Hindernisse navigiert werden. Wie zuvor war Autonomie eine Kernkompetenz und ein direkter Eingriff war nur für kurze Zeit vorgesehen und nur über eine mit einer Verzögerung versehene Kommunikationsstrecke möglich. Nachdem in einer Qualifikation nur wenige Teams alle Aufgaben überhaupt lösen konnten, wurden die Regeln gelockert und von einem Wettbewerbscharakter zu einem Demonstrator verschoben.

Die Anzahl der Robotersysteme wurde auf eine maximale Gesamtmasse von $100kg$ beschränkt, weshalb die 2 Robotersysteme LAURON V und ein Bodenstation mit ausfahrbaren Kameramast, der sogenannte *Lander* (beide in Abbildung 5.5), für die Mission ausgewählt wurden. Der *Lander* diente zum einen als Relais für das Steuersignal von der Bodenstation und agierte zum anderen als zweiter Roboter, indem mit einem mobilen Mast eine hochaufgelöste pan-tilt Kamera auf bis zu 6 Meter Höhe ausgefahren wurde.

Das Setup von LAURON V hat sich in der Hardware kaum zum vorherigen Wettbewerb verändert. Nahezu alle vorher entwickelte Fähigkeiten, etwa die Objekt-

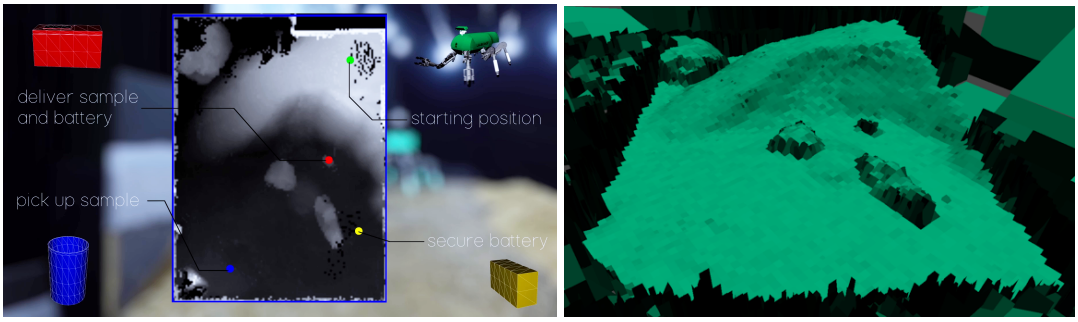


Abbildung 5.6.: Links: Die während des Wettbewerbs aus den Sensordaten erstellte 2D-Höhenkarte mit den eingezeichneten Fundorten der Missionsziele die denen des SpaceBot Cup ähneln. Quelle: [HI+18] Rechts: Die aus den Sensordaten erstellte 2.5D-PlexMap Karte, auf deren Grundlage die Planungen durchgeführt werden. Quelle: [H+15]

erkennung, das Mapping oder die Pfadplanung wurden weiter benutzt oder weiter entwickelt. Durch die hochaufgelösten Laserscanns konnte wieder eine 2.5D-Höhenkarte, die sogenannte PlexMap (Abbildung 5.6 rechts), erstellt und darauf basierend Auswertungen für die vom System geforderten Berichte (Abbildung 5.6 links) erstellt werden, in denen etwa die Höhenkarte des Geländes dargestellt und Fundorte von Objekten verzeichnet werden sollen.

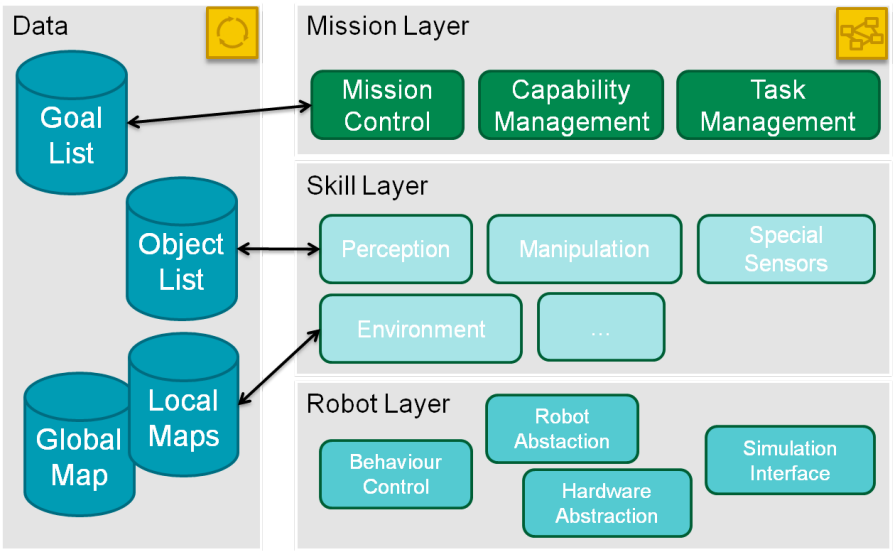


Abbildung 5.7.: Die überarbeitete Software-Architektur während des SpaceBot Camps, die mit der aktuell vorgeschlagenen (4.3) nahezu identisch ist. Quelle: [Heppner et al., 2015].

Vollständig überarbeitet wurden hingegen die Art und Weise, wie die Fähigkeiten verwaltet und eingesetzt wurden. Die Architektur des Frameworks (Abbildung 5.7) wurde verschlankt und entspricht damit bereits der in dieser Arbeit vorgestellten Architektur, wenngleich die Komponenten noch teilweise andere

Bedeutungen bzw. Funktionen haben. Die *Robot Layer* ist die roboterspezifische Abstraktionsebene, über die insbesondere die verhaltensbasierte Steuerung von LAURON V angesprochen wird. Die *Skill Layer* beinhaltet weiterhin die in Paketen gebündelten *Fähigkeiten*, die jedoch um das Konzept der *Koordinatoren* (Abschnitt 4.2.2) erweitert wurden. Durch das Koordinator-Konzept sollte die starke Bindung zwischen Mission Control und den eigentlichen Fähigkeiten, die im vorherigen Wettbewerb zu massiven Problemen geführt hat, reduziert werden, was die Fähigkeiten wart-, test- und wiederverwendbarer gemacht hat. In Abschnitt 4.2.2 ist in Abbildung 4.6 der Koordinator der *MoveToPOI* Fähigkeit zu sehen, die genutzt wird, um einzelne PoI anzulaufen. Abbildung 5.10 (rechts) zeigt den Koordinator der Fähigkeit *RefineManipulationPose* mit der LAURON seine Pose vor einem Objekt optimiert, bevor ein Manipulationsversuch durchgeführt wird. Die Koordinatoren für die jeweiligen Funktionen sind in SMACH[1], einem ROS Framwork für Zustandsautomaten, umgesetzt und erlauben es, die Fähigkeiten lokal und fast unabhängig von anderen Komponenten zu testen und bei Bedarf schnell zu erweitern.

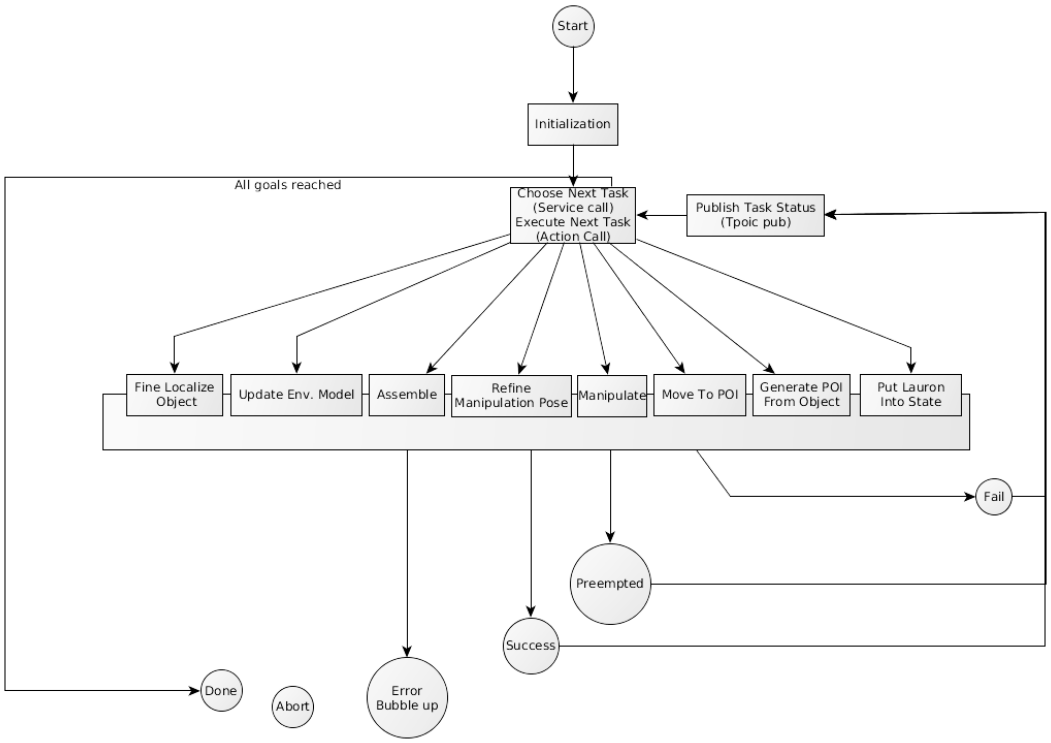


Abbildung 5.8.: Der Zustandsautomat für die Mission Control besteht primär aus einer Ausführungs- und Feedbackschleife, die ein externes Planungsmodul anfragt. Success, Fail und Preempted beziehen sich auf die Aktionen und beeinflusst damit die Mission, Error sind kritische Fehler die übergeordnet behandelt werden müssen. Quelle: [H+15]

[1] `http://wiki.ros.org/smach` abgerufen am 22.11.2023

Die *Mission Layer* wurde vollständig überarbeitet, wobei die vorgesehenen Komponenten zunächst bewusst einfach umgesetzt wurden, um diese dann bei Erfolg zu erweitern. Das *Capability Management* ist eine einfache Liste mit den verfügbaren Fähigkeiten und Endpunkten (ROS Actions). Die *Mission Control* ist ebenfalls als SMACH Zustandsautomat umgesetzt (Abbildung 5.8) der, im Gegensatz zum Ansatz im vorherigen Wettbewerb, nahezu keine eigene Aufgabe erfüllt, sondern primär die Koordinatoren der jeweiligen Fähigkeiten aufruft, die Ausführung überwacht und dann eine externe Entscheidungs-Komponente anfragt. Die externe Komponenten kann beliebig komplex umgesetzt werden, wurde in diesem Fall jedoch vor allem als einfacher FiFo-Speicher umgesetzt. Durch das *Task Management* werden Topics bereitgestellt, über das die Fähigkeiten in diesen Speicher eingefügt oder gelöscht werden können und der aktuelle Ausführungsstatus an die Bodenstation übermittelt wird.

Ein wichtiger Aspekt für die Entkopplung der Koordinatoren von der Missionssteuerung ist ein einheitliches Interface. Jede Fähigkeit wird daher gleich behandelt und mit lediglich ihrem Namen und zwei numerischen IDs aufgerufen, wodurch entsprechende Parameter übergeben werden. Im Falle von *MoveToPoi* ist das zum Beispiel die *PoI-ID* und die Art der Bewegung, also z.b. omnidirektional oder mit einer Differential-Kinematik. Die *PoI-ID* wird wiederum von der Fähigkeit selbst verwendet, um sich die Zielpose und ggf. weitere Eigenschaften des gewählten PoI abzufragen.

Ein derart eingeschränktes Interface bringt naturgemäß Nachteile mit sich, wie etwa eine fehlende Typsicherheit, eingeschränkte Ausdrucksstärke und die Notwendigkeit der Fähigkeiten, selbst ihre Daten zu beschaffen. Auf der anderen Seite wurde das Aufrufen der Fähigkeiten dadurch signifikant vereinfacht und es wurde insgesamt eine starke Entkopplung ermöglicht. Abbildung 5.9 zeigt das Interface mit dem auf der Bodenstation mit dem *Task Management* der *Mission Control* kommuniziert werden kann. Der Nutzer kann Fähigkeiten mit den Buttons auf der linken Seite zur Task-Liste hinzufügen und dort parametrisieren. Durch *Submit Task* wird dann eine ganze Liste an Tasks auf einmal übermittelt, was notwendig ist, da ursprünglich nur kurze Kommunikationsfenster vorgesehen waren[2]. Wird kein Task durch die Bodenstation vorgegeben, generiert das *Entscheidungsmodul* auf dem Roboter den nächsten Task, was im Zweifelsfall immer ein *idle* ist. Der Nutzer oder der Entscheider auf dem Roboter kann auf diese Weise eine Mission aus Fähigkeiten aneinanderreihen, etwa ein *Generate PoI From Object*, um zu einem gefundenen Objekt PoIs zu generieren, die nicht in einem Hindernis liegen und zwischen Roboter und Zielobjekt liegen. Diese können dann mittels *Move To PoI* angelaufen werden, welches selbstständig die nötigen Planungsschritte ausführt, bevor *Fine Localize Object* die volle 3D-Pose des Objektes berechnet, um den Roboter dann mittels *Refine Manipulation Pose* vor dem Objekt zu platzieren und mittels *Manipulate* die Aufnahme zu starten.

[2]Die beschränkte Kommunikation ist eine der wenigen Beschränkungen, die das FZI-Team aufgehoben hat, um vor allem im Fehlerfall die Entwicklungen trotzdem testen zu können.

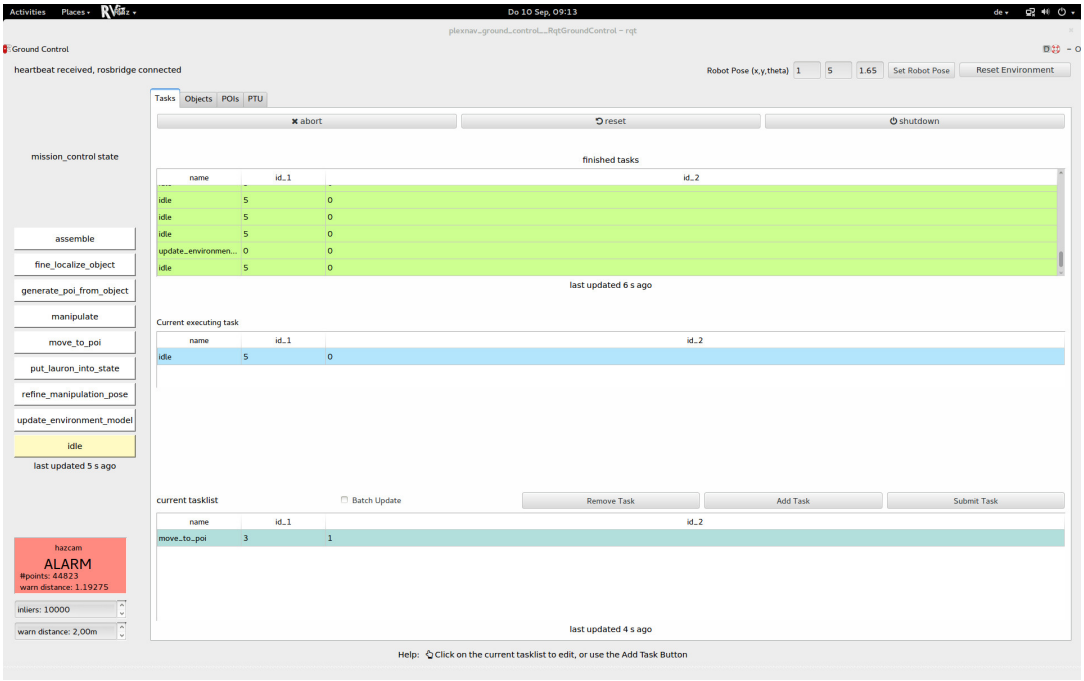


Abbildung 5.9.: Mission Control Plugin zur Kommunikation der Bodenstation mit dem *Task Management* der *Mission Control*. Der Nutzer kann Fähigkeiten auswählen und dadurch zur *Current Task List* (unteres Fenster) hinzufügen und dort parametrisieren. Die komplette Liste kann durch *Submit Task* auf einmal an den Roboter geschickt werden. In den weiteren Reitern (Objects, PoIs, PTU) können vor allem aktuelle Missionsdaten eingesehen und Spezialfunktionen (z.B. set Robot Pose) ausgeführt werden. Das *hazcam* Fenster zeigt die Auswertung der Hindernisdetektion des Roboters an. Quelle: [Heppner et al., 2017]

Die Fähigkeiten selbst können beliebig komplex in Form und Größe sein. *Refine Manipulation Pose* etwa (siehe Abbildung 5.11) nutzt den gesamten Körper und die Navigation, um optimale Griffpositionen zu erreichen. Für jedes Objekt wurden zunächst optimale Griffpunkte identifiziert und dann unter Berücksichtigung der Kinematik mögliche Manipulationsposen berechnet (5.10). Der Koordinator der Fähigkeit (5.11) prüft zunächst, ob das Zielobjekt direkt gegriffen werden kann. Ist dies nicht der Fall, wird durch Objekt-Fitting (Siehe 4.3.2 und Abbildung 4.14) die 3D-Pose des Objektes bestimmt und dann eine mögliche Manipulationspose um das Objekt herum berechnet und gegenüber den Kartendaten validiert. Dann wird ein neuer *PoI* dieser Posen geniert und angelaufen. Zuletzt wird erneut auf Erfolg geprüft oder der Prozess noch einige Male durchlaufen. Der Koordinator nutzt dabei andere Fähigkeiten wie etwa *Fine Localize Object*, erstellt Daten, etwa Manipulations-PoIs durch *getGoodManipulationPose* und liest Daten von globalen Topics, etwa die PoI-Pose, welche dann für *isValidManipulationPose* genutzt wird.

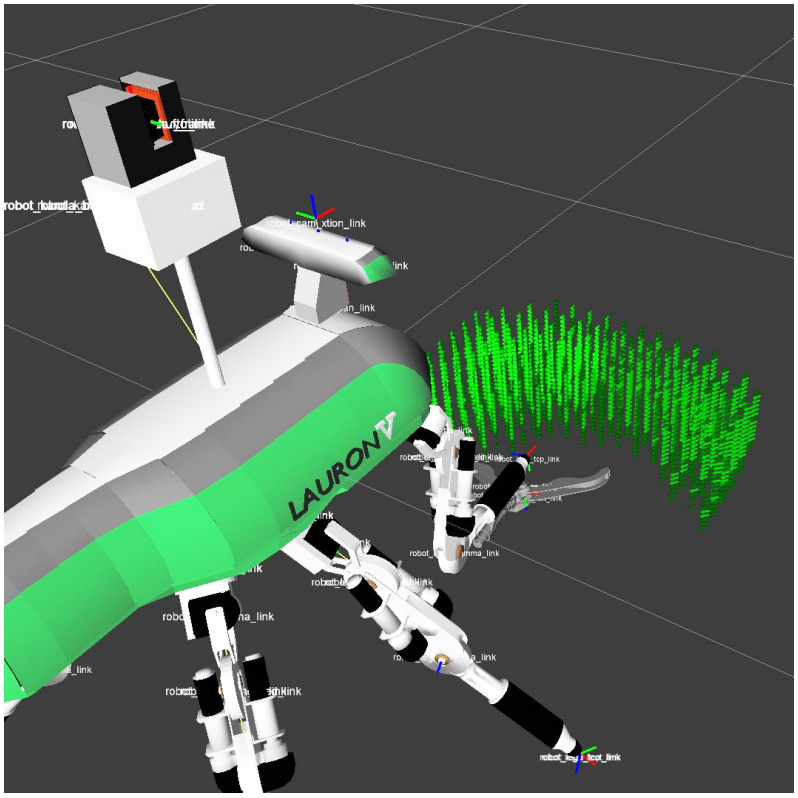


Abbildung 5.10.: Visualisierung der berechneten Griffpunkte für LAURONs Kinematik. Auf Basis der Griffpunkte und der Lokalisierung der Zielobjekte kann eine Manipulationspose eingenommen und von dort aus gegriffen werden. Es werden also sowohl die Freiheitsgrade des Körpers als auch des Arms genutzt. Quelle: [H+15].

Obwohl zwei Roboter beteiligt sind, findet in diesem Beispiel keine Verteilung statt, da die Fähigkeiten disjunkt sind. Beide Roboter tragen allerdings ihre Daten in die globale Datenbank ein, Objekte, die von der Pan-Tilt Einheit auf dem *Lander* gefunden werden, sind also auch für LAURON ein valides Ziel. Insbesondere für die menschlichen Bediener in der Bodenstation hat sich während des Laufs diese zweite Ansicht als wertvoll erwiesen. Das Kommandieren beider Roboter erfolgt dabei natürlich über das vorgestellte Mission-Control-Plugin, in dem die Fähigkeiten des zusätzlichen Systems und einige Entwicklungen durch Erweiterung der Konfigurationsdatei hinzugefügt werden konnten. Abbildung 5.12 zeigt die typische Ansicht der Bodenstation, in der die aktuelle Mission (mit den erweiterten Fähigkeiten), die Sensordaten und eine 3D-Ansicht der Umgebung angezeigt werden.

Die für den SpaceBot Camp entwickelten Komponenten weisen deutlich robustere Funktionalität auf als die vorherige Iteration. Insbesondere die Entscheidung, stark auf die verteilten Koordinatoren in einer Fähigkeit zu setzen und diese einheitlich anzusteuern, hat während der Demonstration erlaubt, nahezu die ganze Mission autonom ablaufen zu lassen, aber auch die Option gegeben, jederzeit re-

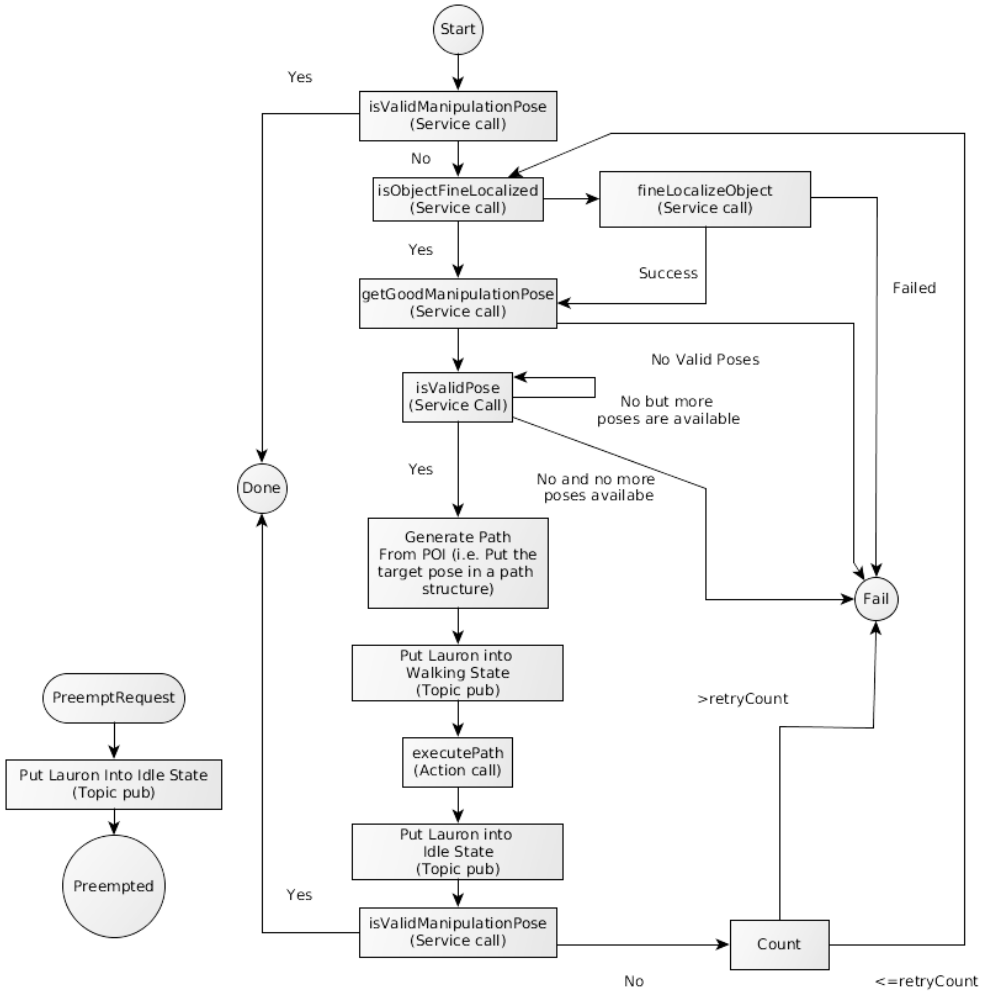


Abbildung 5.11.: Der Koordinator für die Fähigkeit *RefineManipulationPose* mit der die Positionierung von LAURON vor dem Objekt auf Basis der Griffberechnung optimiert wird bevor ein Greifversuch durchgeführt wird. Quelle: [H$^+$15]

aktiv einzugreifen. Nachdem das erste Objekt etwa im Treibsand aufgenommen wurde, wurde dieser nicht als schwieriges Gelände klassifiziert. Eine schnelle Anweisung von der Bodenstation *auszuweichen* konnte sofort übernommen werden und hat die nachfolgende Missionsentscheidung, das nächste Objekt anzulaufen, nicht beeinflusst. Auch die Kapselung der Entwicklungen in der gezeigten Architektur haben gut funktioniert und wurden mit dem *Lander* auf ein zweites (und technisch mit der Bodenstation sogar auf ein drittes) System übertragen. Zwar fand hierbei noch kein Austausch der Fähigkeiten zwischen den Systemen statt, allerdings konnten auch die Fähigkeiten für die weiteren Systeme schnell mit in die Gesamtmission integriert werden, was für den hier gezeigten Ansatz übernommen wurde. Bei der Demonstration selbst konnten die System alle ihre Fähigkeiten einsetzen und dadurch eine äußerst komplexe Weltraummission inklusive Fähigkeiten wie Kartieren, Pfadplanung, mobile Manipulation und Be-

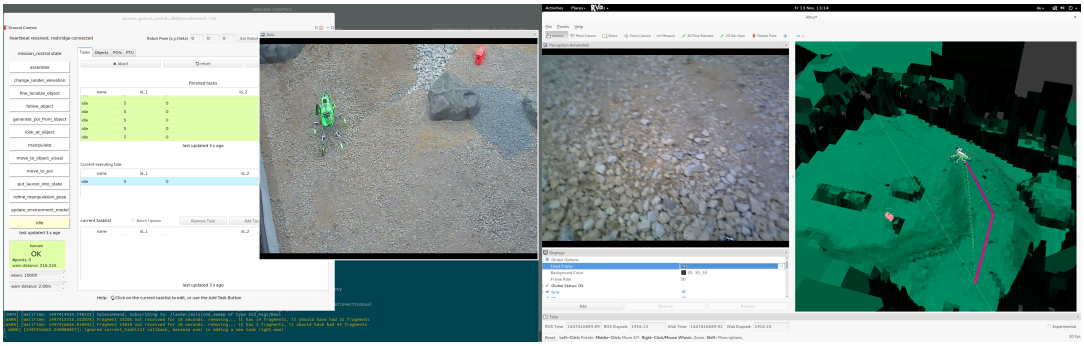


Abbildung 5.12.: Ansicht der Bodenstation während des SpaceBot Camps. Links sieht man die Mission Control mit den wählbaren Fähigkeiten und der Anzeige, welche Aufgaben gerade erledigt wurden, aktuell ausgeführt werden oder sich in der Queue befinden. Rechts daneben die Ansicht vom ausgefahrenen *Lander* und noch einmal rechts daneben das RGB-Bild aus Sicht von LAURON V. Ganz rechts die Kartenansicht der aktuellen Plexmap mit eingezeichnetem Faktorgraph (rote Linien). Quelle [H+15]

wegung in schwierigem Gelände absolviert werden. Auch der gewählte Ansatz der Datenfusion im Team hat sich erneut bewährt.

Bei der Demonstration hat der *Lander* zunächst eigenständig eine Fähigkeit gestartet um auf die vorher definierte Höhe auszufahren, während LAURON, ebenfalls ohne weitere Interaktion, seine Umgebung per *Update Environment* gescannt und kartiert hat. Es wurde sich bewusst dafür entschieden, den Abstieg in den Krater per Teleoperation von der Bodenstation aus durchzuführen, da aufgrund der Größe nahezu keine Fehler oder Abweichungen möglich waren. Die *Robot Abstaction Layer* hat hierbei sehr gut funktioniert und den Roboter in einen Sicherheitszustand versetzt, als er zu dicht an den Abgrund gesteuert wurde. Nach dem Abstieg hat die Bodenstation einige Tasks für das Aufsammeln des Probenbehälters vorgegeben. Dies hat zunächst sehr gut funktioniert, allerdings wurde das Objekt während des Anlaufens verloren und bei der erneuten Sichtung mit einer neuen ID als neues Objekt erkannt. Die bereits erteilten Befehle bezogen sich noch auf das alte Objekt und schlugen daher teilweise fehl. Zu diesem Zeitpunkt wurden jedoch durch die Entscheidungs-Komponenten neue Tasks mit korrektem Ziel generiert und die Mission erfolgreich fortgeführt. Nach einem weiteren Eingriff aufgrund schwierigen Terrains konnte die Mission dann autonom bis zum Zielobjekt fortgesetzt werden, wo ein mechanischer Schaden am Roboter die Demonstration vorzeitig beendet hat.

Die gezeigte Mission hat dadurch bewiesen, dass insbesondere das Fähigkeitskonzept mit Koordinator, aber auch das umgebende Framework unter schwierigen Bedingungen in der Lage sind, nicht nur einen, sondern mehrere Roboter zu koordinieren und autonomes Handeln zu ermöglichen. Neben der erneuten Validierung der *Robot Layer* und *Skill Layer* konnte auch die neu entwickelte *Mission Layer*, trotz einfacher Umsetzungen vieler Details, eine komplexe Mission

verwalten und neue Fähigkeiten autonom, aber auch durch den Nutzer initiiert, für die Mission verwenden. Insbesondere die Möglichkeit, einzelne Fähigkeiten reaktiv für die Mission zu nutzen, hat zu einer hohen Fehlertoleranz geführt. Die „Modellierung“ durch ein einheitliches Interface hat es erlaubt, auch heterogene Systeme (Lander und LAURON) im Team identisch zu behandeln und dadurch das komplette Team effizient in die Mission einzubinden.

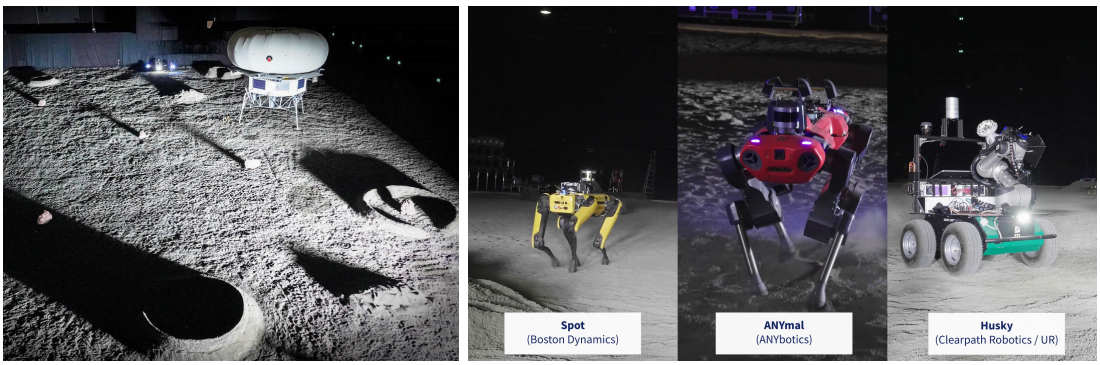


Abbildung 5.13.: Links: Wettbewerbsgelände der ESA Space Ressource Challenge. Das Gelände ist den Bedingungen des Mondes nachempfunden (Beschaffenheit, Beleuchtung, etc.) und beinhaltet neben freiliegenden Steinen auch spezielle Konstellationen wie Krater mit nur einem Eingang oder einen Lander-Modul von früheren Missionen. Oben ist das FZI-Team beim Start zu sehen. Rechts: Das heterogene Team, mit dem das FZI bei dem Wettbewerb angetreten ist. Die beiden Laufroboter Spot und Anymal wurden als agile Explorationsplattformen eingesetzt, während der radgetriebene Husky Equipment für die Vermessung von Bodenpunkten und Steinen an einem Arm mitführt. Quelle [FZI22]

Bei der **ESA SRC (Space Ressources Challenge)** wurden Teile des Systems erneut für den Weltraumkontext und ein heterogenes Roboterteam eingesetzt. Ziel des Wettbewerbs waren die Erkundung einer mondähnlichen, unbekannten Umgebung (Abbildung 5.13 links), das Kartieren und Lokalisieren von besonderen Vorkommnissen und die Suche nach Mineralien im Boden und in Steinen. Zwar wurde auf das Einsammeln von bestimmten Gegenständen verzichtet, durch die Anforderung, Bodenproben und Gesteinsfunde auszuwerten, gab es jedoch auch hier wieder eine mobile Manipulationskomponente. Wie bei den anderen Wettbewerben war keine Sicht auf das eigentliche Roboterteam aus der Bodenstation erlaubt und Kommunikation nur über eine verzögerte Kommunikationsstrecke möglich. Neben Kommunikations-Blackouts gab es während des Wettbewerbs außerdem noch einen simulierten Systemausfall, was die Notwendigkeit eines dezentralen Systems vergrößerte.

Das FZI hat für den Wettbewerb auf ein Team aus drei heterogenen Robotern gesetzt: Die Laufroboter Spot und ANYmal als schnelle Explorationsplattformen

und der Radroboter Husky als mobiles Labor mit mobiler Manipulation (Abbildung 5.13). Die beiden Laufroboter waren jeweils mit einem 360°Laserscanner und einer Kamera mit Beleuchtungseinheit ausgestattet, die Husky Plattform verfügt ebenfalls über Laserscanner und Kamera, zusätzlich jedoch einen Universal Robots Leichtbauarm. An dessen Endeffektor wurden ein Röntgenspektrometer, ein batteriebetriebener Trennschleifer und eine Kameraeinheit, die das Außenlicht bei einer Aufnahme abschirmen kann, montiert. Das Team wurde zudem von einer mobilen Bodenstation (Abbildung 5.14 links) überwacht.

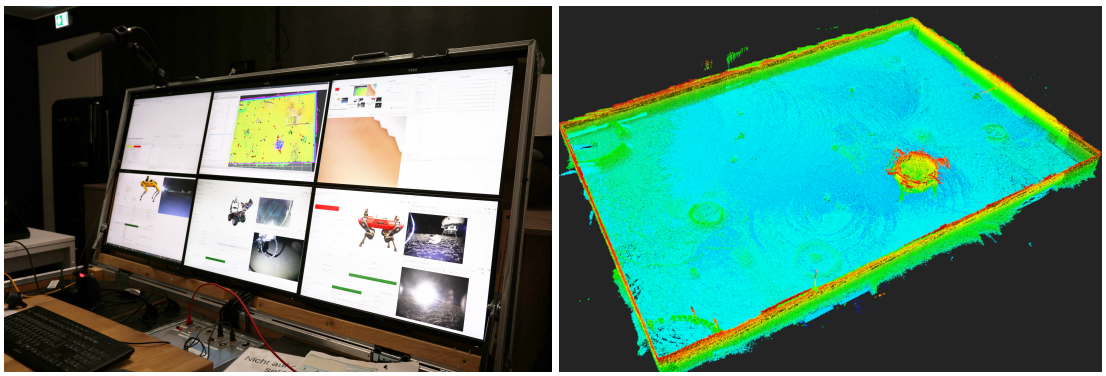


Abbildung 5.14.: Links: Mobile Bodenstation, welche für Weltraumeinsätze der mobilen Roboter entwickelt wurde. Auf den 6 Monitoren werden verschiedene UI zum aktuellen Missionszustand, der Status der einzelnen Roboter im Team und die erstellte 3D-Umgebungskarte angezeigt. Rechts: Die während der Exploration erstellte 3D-Karte der unbekannten Umgebung in einem Voxelformat. Die Krater und der vorher platzierte Lander sind deutlich zu erkennen.

Das Konzept für die Mission war ein starker Fokus auf die Autonomie der Systeme mit der expliziten Möglichkeit, auf unterschiedlichen Ebenen (Mission, Planung, Steuerungsebene) eingreifen zu können. Der dafür eingesetzte *RET* wurde neu entwickelt und verfolgt insbesondere für die Koordination der Roboter einen anderen Ansatz als in dieser Arbeit vorgestellt, nutzt jedoch ebenfalls Fähigkeiten und deren Modellierung mittels BTs als Kern für die flexible Autonomie. Etliche der genutzten Module und Fähigkeiten wurden schon in vorherigen Projekten eingesetzt und bilden zusammen mit missionsspezifischen Modulen die Bausteine für die komplexen Missionen der Roboter: Die *FZI Navigation Pipeline* stellt etwa die Fähigkeit *MoveToPoi* für die Planung und Bewegung der einzelnen Roboter, die *FZI Motion Pipeline* ermöglicht für das Ausführen von Arm-Bewegungen und das *VDB-Mapping* [101] erstellt eine 3D-Karte des Geländes (Abbildung 5.14 rechts).

Ein wichtiger Unterschied der SRC zum in dieser Arbeit vorgestellten Ansatz der expliziten Koordination per Auktion ist die Verwendung von PoIs und den erkannten Steinen (als eine andere Art PoI) als maximal dezentralisiert und parallelisiertes Koordinationswerkzeug. PoI wurden auch beim SBC bereits einge-

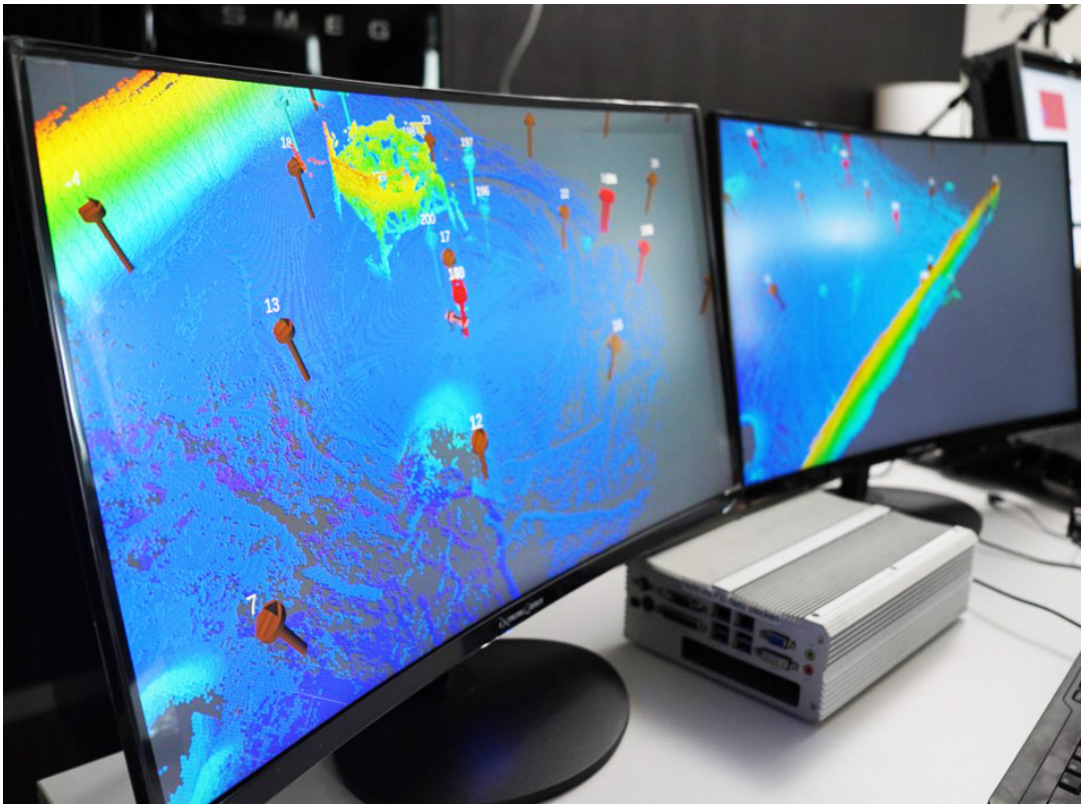


Abbildung 5.15.: PoI-Marker in der erstellten 3D-Karte beim ESA-SRC. Die PoI können automatisiert oder durch die Bediener der Bodenstation gesetzt werden und werden von den Robotern individuell als Ziel verwendet.

setzt um die Datenabhängigkeit der Fähigkeiten zu reduzieren. Hier wurden etwa durch die Fähigkeit *Generate Manpilation Poses* PoI vor dem Objekt generiert auf die der Roboter dann mit einer anderen Fähigkeit navigieren konnte. Bei der SRC wurde dieses Konzept dahingehend erweitert, das die PoIs die Mission definieren. Basierend auf dem aktuellen Modus des Roboters (Autonom, Manuell, Blackout), Typ des PoI (Explorationspunkt, Analysepunkt, Gesteinsprobe, etc.), der aktuellen Roboterposition und weiterem Weltzustand (gefundene Ziele, Batteriestand, etc.) wird durch jeden Roboter selbst entschieden, welche PoI als nächstes ausgewählt wird. Basierend auf dem Typ des PoI wird dann die entsprechend damit verknüpfte Fähigkeit ausgeführt. Die PoIs können dabei auf mehrere Arten erstellt werden (Abbildung 5.15):

- Manuell
  Der Bediener kann auf der Bodenstation jederzeit manuell Punkte jeden Typs einfügen und damit auf Missionsebene Ziele vorgeben, etwa wenn ein Teil der Karte besonders interessant ist oder die automatische Generierung fehlschlägt.

- Vorberechnet
  Die gezielte Analyse des Bodens erfordert eine systematische Vermessung.

Dafür werden auf Basis der vorher bekannten Kartenparameter (Größe des Suchbereichs, grobe Höhenkarte) Analysepunkte in einem Raster platziert. Verschiedene Kriterien wie Messgenauigkeit oder Hindernisse im Analysebereich können die Generierung dabei beeinflussen

- Sensorbasiert
  Insbesondere die Explorationspunkte werden auf Basis eines Frontier-Algorithmus und der erstellen 3D-Karte dort generiert, wo bisher am wenigsten exploriert wurde. Zusätzlich werden durch die Detektion von relevanten Steinen Analysepunkte für diese generiert

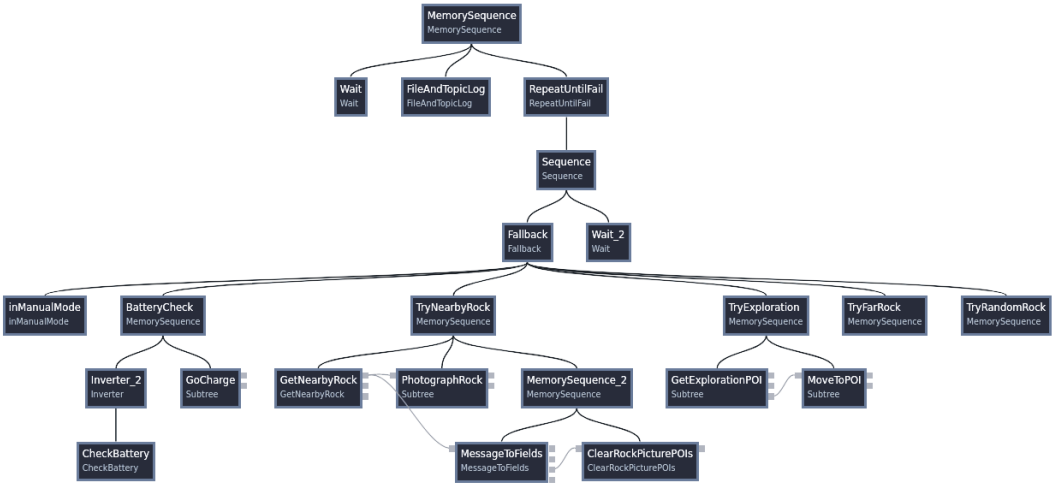


Abbildung 5.16.: Reduzierter Behavior Tree für das Explorationsverhalten bei der SRC. Unter dem Fallback sind die verschiedenen Handlungsoptionen angeordnet (*TryFarRock* und *TryRandomRock* wurden für die kürzere Darstellung entfernt). Auf der Top-Level-Ebene werden vor allem die passenden PoI ermittelt und dann an die spezialisierten Fähigkeiten in Subtrees, wie z.b. *PhotographRock*, übergeben.

Husky führt durchgehend die *Analyse*-Mission durch, bei der an den generierten Analyse-PoI Bodenmessungen durchgeführt werden, die vollautonom ablaufen. Gefunden Steine werden, wenn der Modus dies zulässt, teilautonom abgeschliffen und genauer vermessen. Die beiden Laufroboter übernehmen die Exploration, wobei der in Abbildung 5.16 gezeigte reduzierte [3] Top-Level Behavior Tree verwendet wird. Unterhalb des *Fallback* sind die verschieden priorisierten Handlungsoptionen klar zu erkennen. Nur wenn der manuelle Modus nicht aktiviert und keine niedrige Batterie erkannt wurde, wird zunächst geprüft, ob ein Stein-PoI in der Nähe des Roboter ist (*TryNearbyRock*). Ist dies nicht der Fall, wird entweder die Karte exploriert (*TryExploration*) oder weiter entfernt liegende Ziele (*TryFarRock* und *TryRandomRock*) angelaufen, die hier zur besseren Darstellung

[3] Voller BT im Appendix A

entfernt wurden. Klar erkennbar ist das Muster, dass für jede Handlungsoption zunächst ein PoI oder ein erkannter Stein angefordert wird und auf Basis der Verfügbarkeit eine entsprechende Fähigkeit ausgeführt wird. Die Fähigkeiten sind in diesem Fall als *Subtrees* modelliert, denen der ausgewählte PoI von diesem Tree übergeben wird.

Das Beispiel des *PhotographRock*-Subtree ist in Abbildung 5.17 zu sehen. Durch den *IOInputOption* Knoten können für Subtrees öffentliche Parameter modelliert werden, die grauen Linien zwischen den rechteckigen Parametern (Input/Ouputs) der Knoten zeigen die Datenkanten an. In der Fähigkeit werden zunächst, ähnlich zu der beim SBC gezeigten *Refine Manipulation Pose*, PoIs für das anlaufen des Steines generiert, diese dann für die Ausführung markiert, angelaufen, und wieder gelöscht, bevor dann mit dem Knoten *CandidateImage* die eigentliche Aufnahme des Bildes von einem interessanten Objekt, hier ein Stein, durchgeführt wird. Ein noch nicht bearbeitetes Problem der BT-Implementierung sind die zahlreichen Logging-Knoten, die in nahezu jeder Sequenz eingebaut sind. Ein integriertes Logging würde die Übersichtlichkeit der Bäume noch einmal verbessern.

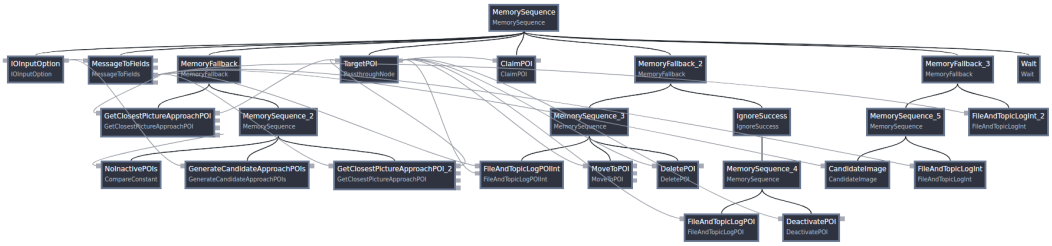


Abbildung 5.17.: Behavior Tree für die *PhotographRock* Fähigkeit bei der SRC. Die *IOInputOption* wird genutzt, um die Input-Paramter des Subtrees explizit zu modellieren. Im Verhalten selbst wird dann ein Approach-PoI erzeugt, dieser angelaufen und dann mit *CandidateImage* das eigentlich Bild aufgenommen. Die hellgrauen Linien zwischen den Knoten sind die Datenkanten mit denen die Parameter an die jeweiligen Knoten übergeben werden.

Sowohl der Top Level BT als auch der *PhotographRock*-BT werden nahezu identisch auf Spot und ANYmal ausgeführt. Unterschiede finden sich erst auf der untersten Ebene, etwa bei der Parametrisierung des *CandidateImage*-Knotes (Abbildung 5.18). Während bei Spot eine externe Basler Kamera aktiviert wird, nutzt ANYmal einen Framegrabber, der ein Bild des internen Video-Streams von ANYmal aufzeichnet. Wären für die Implementierung *Capabilites* anstatt *Subtrees* eingesetzt worden, hätte an dieser Stelle der identische *ProtographRock*-BT verwendet werden können, die roboterspezifische Parameterisierung könnte dann als eigene Implementierung abgelegt und auf jedem System automatisch verwendet werden.

Das Beispiel illustriert, damit wie Missionen natürlich aus einzelnen Task-Blöcken zusammengesetzt werden, die jeweils eine Fähigkeit darstellen. *GoCharge*, *Pho-*

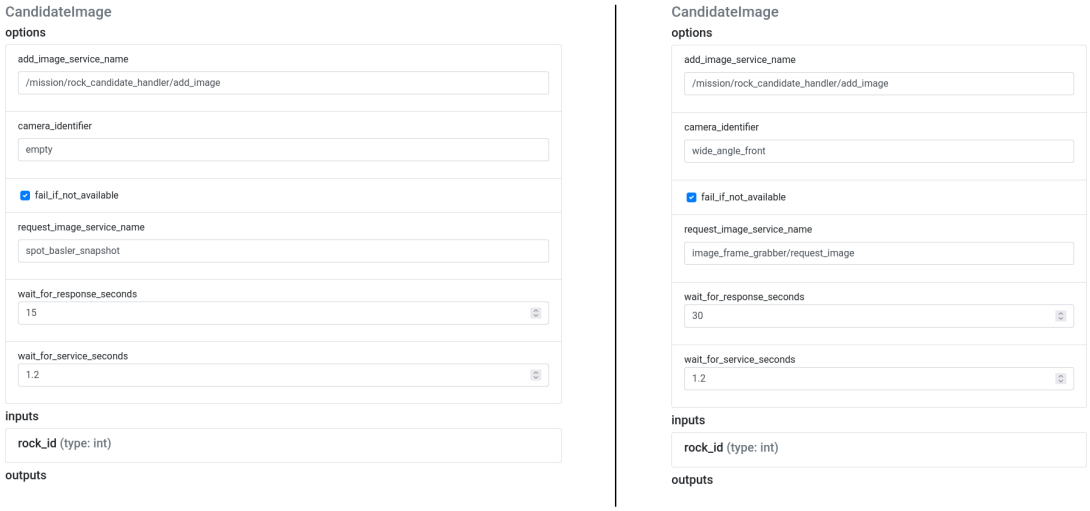


Abbildung 5.18.: Unterschiedliche Parametrisierung des *CandidateImage* Knotens. Links: Paramter auf Spot, Rechts: Parameter auf ANYmal.

tograpRock, *MoveToPoi* sind Anweisungen, die vor allem auf semantischer Ebene eine Bedeutung haben, die auch über heterogene Systeme hinweg einheitlich genutzt werden kann. Erst bei der Implementierung treten die eigentlichen Besonderheiten zu Tage. Das fähigkeitsbasierte Konzept dieser Arbeit nutzt genau diesen Umstand.

Das FZI-Team hat den Wettbewerb gewonnen und dabei vor allem durch die Autonomie des Systems überzeugt. Nach etwa 35 Minuten hat das autonome Team die Exploration größtenteils abgeschlossen. Spot hat zu diesem Zeitpunkt 37 PoI bei einer Laufdistanz von 330m exploriert und dabei 12 potentielle Steine für die weitere Untersuchung fotografiert. Husky hat währenddessen insgesamt 400m² Bodenfläche mit autonomen Messungen untersucht. Die gezeigten BTs belegen deutlich, wie einfach verschiedene Verhaltensblöcke durch einen Fallback-Knoten zu einer Mission kombiniert werden können. Auch Spezialverhalten, wie etwa das aufsuchen der Ladestation bei geringem Akku-Stand, können durch die Reaktivität der BTs einfach berücksichtigt werden.

5.1.1. Diskussion und Einordnung der Ergebnisse

Die präsentierten Anwendungsfälle im exoplanetaren Bereich stellen sehr hohe Anforderungen an die Autonomie und Fähigkeiten der Systeme. Die Umgebungen sind unstrukturiert und vorher nicht bekannt, so dass zum einen erst aktiv die Umwelt wahrgenommen werden muss und zum anderen stets eine aktive Entscheidungen auf Basis des aktuellen Weltzustandes getroffen werden muss. Durch eine beschränkte Kommunikation mit hoher Verzögerung oder sogar vollständigen Verbindungsabbrüchen können menschliche Bediener zudem kaum eingreifen. Auch die in den Anwendungen geforderten Fähigkeiten wie mobile Manipulation oder eigenständige Untersuchung von Bodenproben zählen mit zu

den anspruchsvollsten Aufgaben für mobile Roboter. Es wurde gezeigt, dass die vorgeschlagenen **Fähigkeis- und Missionsmodellierungungen für komplexen Missionen bzw. Aufgaben geeignet sind und reale Anwendungsfälle umsetzen können**. Während beim SpaceBot Cup die Missionssteuerung noch ein massives Problem darstellte, hat der Fokus auf den entkoppelnden Koordinator beim nachfolgenden Camp einen signifikanten Fortschritt erbracht. Auch beim SRC wurden Fähigkeiten für die einzelnen Teile der Aufgabe verwendet, hier wurde jedoch keine explizite Modellierung mehr benötigt, da **sowohl Mission als auch Fähigkeiten als BT realisiert** wurden. Gleichzeitig muss allerdings auch angemerkt werden, dass bei der SRC ein Austausch der einzelnen Fähigkeiten nicht explizit vorgesehen wurde und die Fähigkeiten daher als *Subtree* modelliert wurden. Da Fähigkeiten als Fähigkeitsimplementierung effektiv einen *Subtree* laden sind die Ergebnisse jedoch vergleichbar.

Der in der SRC gezeigte Ansatz verwendet die bereits in den vorherigen Wettbewerben gezeigte PoI-Konzepte nicht nur zum Datenaustausch, sondern nutzt dieses auch zur Synchonisation der Aufgaben über die Roboter hinweg. Dabei wird die durchzuführende Aufgabe auch hier als Modellierung verwendet, wobei die Implementierung auf mehreren, heterogenen Robotern unterschiedlich umgesetzt werden kann (insbesondere die Bewegung und Datenaufzeichnung). Dies zeigt die **Möglichkeit, die Fähigkeiten zur Koordination einzusetzen und dass sie für die Koordination eines heterogenen Teams geeignet sind**. Der Ansatz der Koordination über PoIs hat den Vorteil einer massiven Parallelisierung im Team ohne signifikanten Overhead, kommt jedoch mit dem Nachteil der starken Spezialisierung des Teams. Wenn etwa die Mission derart geändert würde, dass die Laufroboter Proben transportieren sollen, die ihnen Husky auflädt, dann gäbe es keine Möglichkeit, diese geänderte Mission umzusetzen, obwohl alle Robter über die nötigen Fähigkeiten dazu verfügen. Dieser Aspekt wird in Abschnitt 6.3.2 weiter diskutiert. Die tatsächlich entwickelten Fähigkeiten wurden in der gleichen oder weiterentwickelter Form in inzwischen zahlreichen Projekten verwendet und konnten insbesondere die verteilte Entwicklung verbessern.

Der SpaceBot Cup und Camp haben gezeigt, dass das **modulare Framework gut für den Einsatz und die Koordination von Fähigkeiten funktioniert hat**. Auch wenn die *Mission Control* simpel umgesetzt wurde, konnte gezeigt werden, dass die Aufteilung in die entsprechenden Komponenten funktioniert und neben autonomen Entscheidungen auch der Eingriff durch den Bediener möglich ist. Neben der Ausführung konnte hier auch deutlich der leicht verstehbare Charakter der Fähigkeiten genutzt werden, um **schnell und einfach komplexe Missionen für das Roboterteam zu erstellen**.

Zuletzt wurden insbesondere für das SpaceBot Camp und die Space Ressource Challenge **sehr heterogene Robotersysteme eingesetzt und erfolgreich mit den gezeigten Fähigkeiten angesteuert und koordiniert**.

Bisher nicht gezeigt wurde vor allem der Austausch der Fähigkeiten zwischen den Systemen, da die Space Ressource Challenge den Ansatz einer Synchonisie-

rung über die Missionsziele gewählt hat und im SpaceBot Cup und Camp vor allem das Einzelsystem LAURON im Vordergrund stand.

5.1.2. Beitrag zur Arbeit

Die Entwicklungen im Rahmen der Wettbewerbe stellen eine iterative Implementierung und Evaluation der vorgeschlagenen Konzepte unter strikten Realbedingungen dar. Durch die fortgeführte Nutzung der Konzepte konnten dabei sowohl die erfolgreichen Umsetzungen evaluiert, sowie aus den nicht erfolgreichen Ansätzen gelernt werden.

Das **Fähigkeitskonzept** hat sich in allen drei vorgestellten und in zahlreichen, nicht erwähnten weiteren Arbeiten, als eine sinnvolle Modellierungsgröße herausgestellt. In jedem Fall werden komplexe Fähigkeiten aus den einfacheren hierarchisch aufgebaut und sind dadurch in der Lage, auch komplexeste Aufgaben relativ einfach zu verbinden. Das Kommandieren auf Fähigkeitsebene erlaubt es, sowohl komplexe Abläufe wie das Aufsammeln von Proben, aber auch einfache Befehle, wie das Ausfahren eines Mastes ohne Wechsel der Modellierung, zu kommandieren. Durch die einheitliche Verwendung können auch **heterogene Systeme mit einem einheitlichen System koordiniert** werden und dadurch Aufgaben wie das Absuchen der Umgebung auf ein Team aufgeteilt werden.

Das Konzept des **Koordinators** wurde als Notwendigkeit für die Fähigkeiten eingeführt und hat sich seitdem als gute Abstraktion erwiesen. Der Ansatz des einheitlichen Interfaces war, wie vorhersehbar, zu einschränkend, hat jedoch ein frühes Testen des Konzeptes erlaubt und gezeigt, dass ein einheitliches Ansprechen des Koordinators ein wichtiges Kriterium für eine Entkopplung und dadurch bessere Modularität ist. Aufgrund der Erfahrungen wurden die **Behavior Trees als Modellierung auf Missions- und Fähigkeitsebene** entwickelt und der *Capability*-Knoten für die Behavior Trees entwickelt.

Das **Framework** für die Ausführung der Fähigkeiten wurde wiederholt verschlankt und auf das Notwendige reduziert. Es konnte durch die verschiedenen Experimente gezeigt werden, dass das Framework geeignet ist, die Anwendung der Fähigkeiten zu koordinieren und in geeigneter Weise zu abstrahieren. **Die Möglichkeit, Fähigkeiten von verschiedenen Systemen zu Sammeln und einheitlich zur Verfügung zu stellen, erlaubt die schnelle Koordination eines Teams.** Die reaktiven Elemente in der Missionssteuerung und das schnelle Ausführen von extern zugewiesenen Fähigkeiten machen den Roboter robust und flexibel.

Die **Modellierung von Fähigkeiten und Missionen mittels Behavior Tree** ermöglicht es, mittels BT komplexe Missionen für heterogene Teams aufzubauen, was durch die erfolgreiche SRC bewiesen wurde. Durch die Modellierung der Fähigkeiten direkt im BT-Framework können diese nicht nur einfacher in die Mission integriert werden, sondern sind auch unabhängiger von der *Mission Control*

Komponente und können damit in weiteren Entwicklungen, wie etwa dem RET, einfach verwendet werden.

Die **Implementierung** auf Basis von ROS wurde in zahlreichen Implementierungen bestätigt, da nahezu alle Roboter mit einer ROS-Schnittstelle arbeiten können und heterogene Systeme damit angesprochen und koordiniert werden können.

5.2. Modellierung und Nutzung wiederverwendbarer Fähigkeiten im Industriellen Kontext

Der im Kapitel 4.2 vorgestellte MDE-Ansatz für die Modellierung einer Fähigkeit und das Nutzen eines Koordinators wurde in einem Demonstrator umgesetzt, bei dem vor allem die Austauschbarkeit einzelner Komponenten gezeigt wurde. Der Demonstrator realisiert den Anwendungsfall eines Automobilherstellers (BMW), bei dem eine eine Dämmmatte in die Türen einer BMW-Limousine und eines BMW-Coupés eingebaut werden.

Im Montageprozess wird eine Dämmmatte inklusive der Klebelinie von einem Zulieferer bereitgestellt und von menschlichen Mitarbeitern an einer Montagelinie in den Türrahmen eingelegt. Im folgenden Zyklus wird ein Roboter eingesetzt, der entlang der nicht trivialen Klebelinie einen konstanten Druck ausübt, um die Verbindung mit dem Türrahmen zu fixieren.

Der Anwendungsfall wurde bereits mit herkömmlichen Methoden automatisiert, das Hauptziel des Demonstrators bestand darin, die Anwendbarkeit und Flexibilität des gewählten Modellierungs-Ansatzes zu demonstrieren. Durch Austauschen von Fähigkeiten konnte gezeigt werden, wie das Modell die Anwendung dabei unterstützen kann, unterschiedliche Implementierungen schnell zu wechseln.

Im Demonstrationsaufbau (siehe Abbildung 5.19) wird eine FANUC M-710iC/50 (A) verwendet, um die Pressbewegung auszuführen, wobei eine Rolle (B) auf einem ausfahrbaren Pneumatikzylinder (C) entlang der Klebelinie geführt wird. Die Autotür wird an der speziellen Vorrichtung (D), die in der Produktionslinie verwendet wird, befestigt und von einem weiteren Pneumatikzylinder an ihrer Rückseite gestützt. Mit einem bodennahen RFID-Lesegerät (E) wird der Typ der Tür erkannt. Alle Softwarebausteine und die Workbench (I), das speziell entwickelte Tooling zum Erstellen und Modifizieren der MDE-Lösung, werden innerhalb der Integrationsplattform ausgeführt, die auf einem Industrie-PC läuft, der sich im Schaltschrank (H) befindet. Während die Workbench zur Modifikation der aktuellen Lösung verwendet wird, zeigt eine webbasierte Benutzeroberfläche den aktuellen Prozessschritt an (J). Der Schlüsselaspekt des Demonstrators sind die verschiedenen Sensoren, die zur Lokalisierung der Tür verwendet werden.

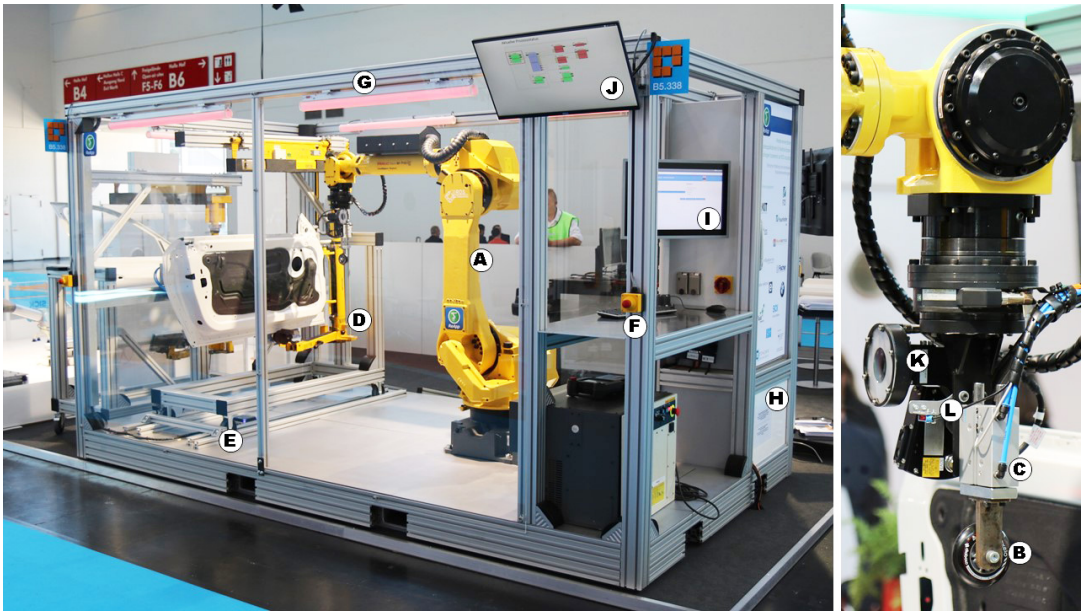


Abbildung 5.19.: BMW Demonstrator zur evaluation und Präsentation der Fähigkeitsmodellierung auf der Automatica 2016. A: FANUC M-710iC/50, B: Walze, C: TCP Pneumatik C: pneumatischer TCP-Zylinder, D: Türbefestigung, E: RFID-Leser, F: Sicherheit (ausgewertet mit traditioneller SPS), G: Statusleuchten (gesteuert durch Sicherheits-Skill), H: Schaltschrank mit Integrationsplattform, I: ReApp Workbench, J: aktueller Prozessstatus, K: FANUC Vision System, L: SICK Lichtreflexschalter.

Ihre Position wird entweder durch das integrierte FANUC Bildverarbeitungssystem (K) oder einen am TCP montierten Lichtreflexschalter (L) erkannt.

Die verwendete Modellierung unterscheidet 4 Typen von Komponenten die zusammen zu einer *Lösung* kombiniert werden:

- Sofware (SW)-Komponenten
  Module, die reine Softwarefunktionalität zur Verfügung stellen.
- Hardware Access (HA)-Komponenten
  Treiberbausteine, die direkt oder indirekt mit einer Hardware interagieren .
- Koordinatoren
  Module, zur Steuerung des Ablaufs einer *Lösung*.
- Skills
  Module die aus SW-Kompontenten, HA-Komponenten, ggf. weiteren Skills und einem Koordinator aufgebaut sind.

Die Skills sind eine Umsetzung der *Fähigkeiten* im Sinne dieser Arbeit, allerdings ohne die BT-Modellierung für die Koordinatoren oder den Aufbau der Fähigkeit.

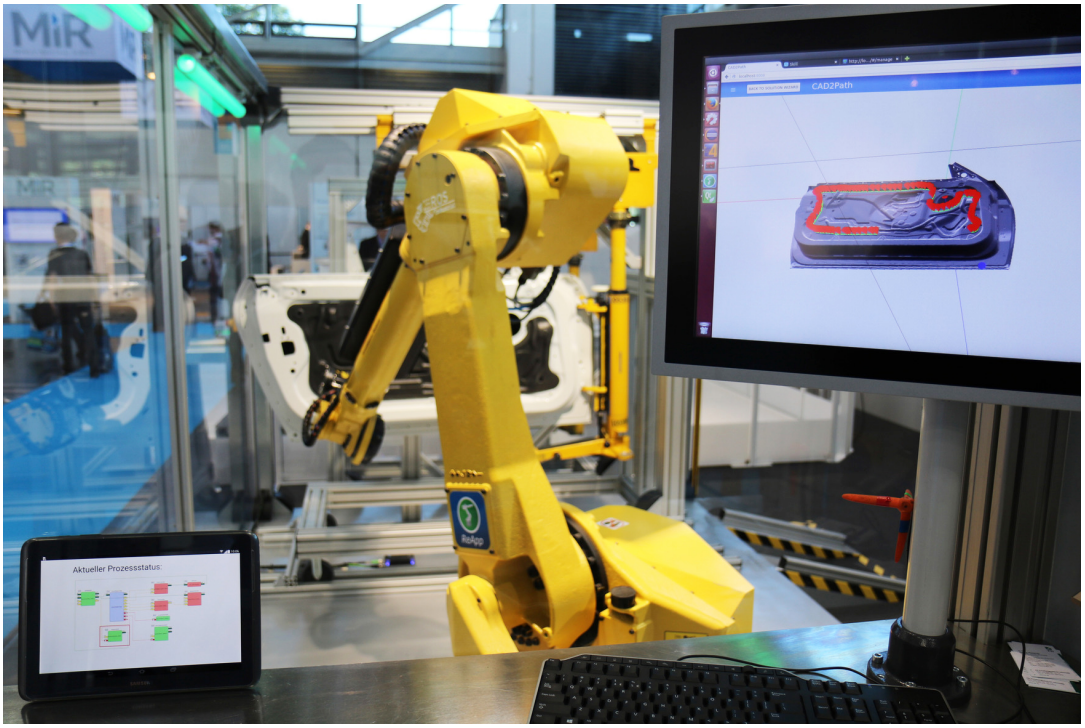


Abbildung 5.20.: CAD2Path Tooling, mit dem die Trajektorie für den Roboter definiert wird und der Solution Wizard, mit dem der Nutzer die laufende Lösung modifizieren kann.

Die vollständige Top-Level-Lösung (siehe Abb. 5.21) enthält 4 Skills (grüne Kästen), vier HA-Komponenten (rote Kästen) und einen Koordinator (blauer Kasten). Der Koordinator nimmt seine Arbeit auf, sobald er instanziiert ist, und durchläuft einen vollständigen Zyklus des Prozesses. Im ersten Schritt wird der Türerkennungs-Skill aktiviert, der auf das Vorhandensein einer Tür wartet, um den erkannten Typ zu melden. Im zweiten Schritt wird der Türlokalisierungs-Skill aktiviert, der auf zwei verschiedene Arten implementiert werden kann: Die erste Option ist das FANUC Vision System (Abb. 5.19(K)), das den Vorteil der one-shot Lokalisierung (mit Laser und Musterabgleich) bietet, aber wesentlich teurer ist als die zweite Option.

Es wurde eine alternative Lokalisierungsmethode implementiert, bei der ein SICK-Lichtreflexschalter (Abb. 5.19(L)) verwendet wird, der entlang der Kanten der Tür bewegt wird, um die Kanten zu erkennen. Diese Methode erfordert mehrere Messungen bei niedrigen Geschwindigkeiten, um die Tür genau zu lokalisieren, kann aber mit Standardhardware durchgeführt werden.Beide Methoden aktualisieren das Referenz-Koordinatensystem (den TF) der Tür in Bezug auf den Roboter und erreichen daher das gleiche Ziel mit unterschiedlichen Methoden. Schließlich wird der Bewegungs-Skill verwendet, um die Trajektorie entlang des Klebstoffs auszuführen, der im Türrahmen definiert ist und mit dem CAD2PATH-Tool [Heppner et al., 2020] (Abbildung 5.20) erstellt wurde, bei dem der Benutzer auf ein 3D-Cad-Modell des Werkstücks zeichnet. Der Safety Skill

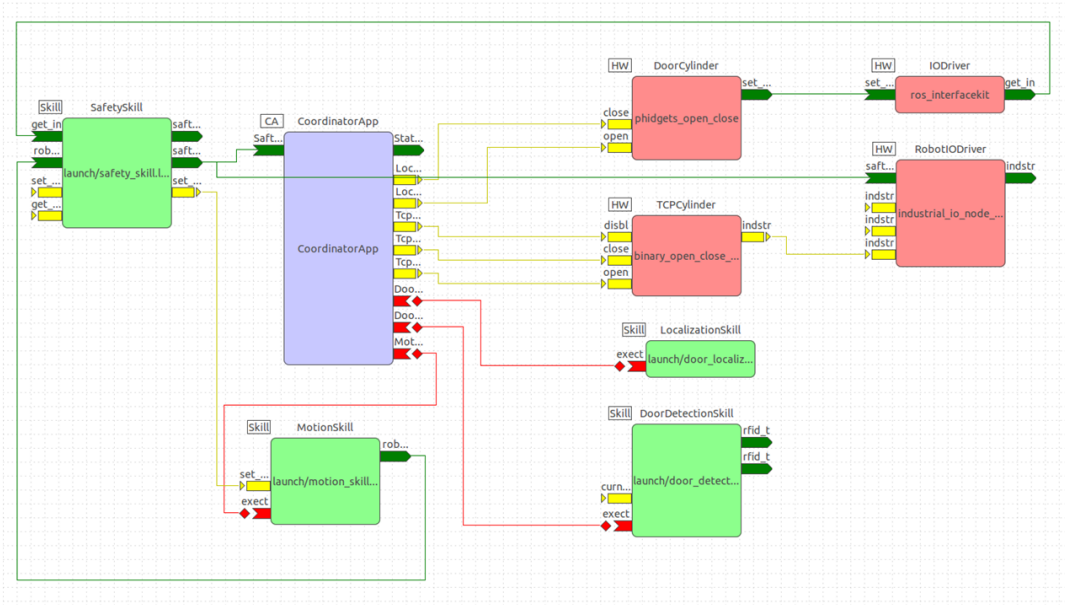


Abbildung 5.21.: Software-Struktur für den Dämmmattenmontage-Skill (Fähigkeit). Er besteht aus vier Skills/Fähigkeiten (grüne Kästchen) zur Handhabung der Türlokalisierung, Türerkennung, Roboterbewegung und Sicherheitsüberwachung, vier Hardware Access Components (rot) zur Steuerung der Pneumatikzylinder über IO-Control und einem Koordinator (blau), der die Ablauflogik handhabt. Die einzelnen Anwendungen sind über ROS-Topics (grün), -Services (gelb) und -Actions (rot) miteinander verbunden. Quelle [H+16]

wertet jederzeit die Daten der Sicherheits-SPS aus und verlangsamt entweder die Robotergeschwindigkeit oder zeigt an, dass ein kompletter Stillstand eingetreten ist, woraufhin der Koordinator Wiederherstellungsmaßnahmen einleitet. Die übrigen HA-Komponenten dienen als Schnittstelle zu den Pneumatikzylindern.

Der Koordinator (blauer Block in Abb. 5.19) wurde mit dem Industriestandard IEC61499 [102] in der 4DIAC Entwicklungsumgebung [103] umgesetzt. IEC61499 definiert Funktionsblöcke, die wie Klassische FSMs auf Events reagieren ihre Aktion ausführen und wiederum eigene Events auslösen können. Durch Verschachtlung der Blöcke können dadurch auch komplexe Funktionen realisiert werden. Sie werden insbesondere genutzt um klassische Controller-Funktionalitäten durch eine herstellerübergreifenden Definition zu modellieren. Für die Verwendung mit ROS wurden spezielle Funktionsblöcke definiert, die ein Interfacing mit den ROS-Topics, -Services und -Actions ermöglichen [104]. Diese werden durch die entwickelte Workbench automatisch generiert und mit einem Basis-Funktionsblock verbunden, dessen Execution Control Chart (ECC) wiederum für die Definition der Ablaufsteuerung (Abb. 5.22) genutzt werden kann. Das ECC definiert ähnlich wie eine UML-Zustandsdiagramm Zustände und Events, wobei in jedem Zustand Algorithmen ausgeführt werden können. Der Koordinator nutzt

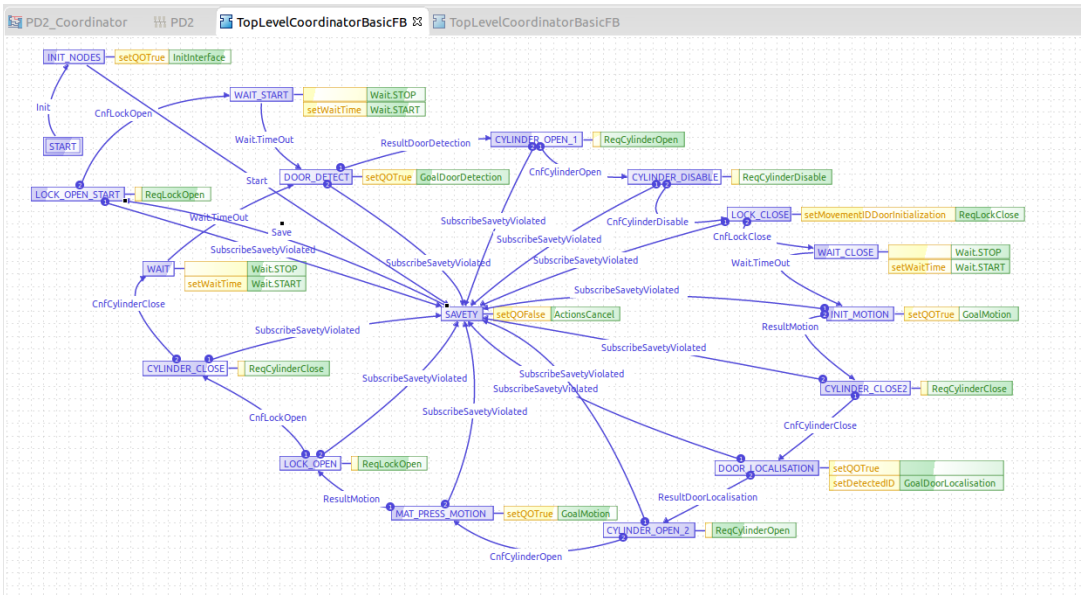


Abbildung 5.22.: Auf dem Industriestandard IEC61499 basierender Koordinator für den Demonstrator. Spezielle Funktionsblöcke für das Interfacing mit ROS werden durch das entwickelte Tooling automatisch mit einem Basis-Funktionsblock verbunden. Durch das Execution Control Chart (ECC) dieses Funktionsblocks kann der Ablauf der Applikation definiert werden. Der Koordinator startet mit einer Intitalisierung im Safety-Zustand und wartet dann auf eine erfolgreiche Erkennung einer Tür. Ist eine Tür vorhanden, wird diese zunächst mit den Pneumatikzylindern fixiert. Dann erfolgt das Triggern der Posendetektion und das Ausführen der eigentlichen Roboterbewegegung, bevor die Fixierung gelöst und erneut auf eine detektierte Tür gewartet wird. Von jedem Zustand kann durch das Auslösen der Sicherheitsanlage in den Safety-Zustand gewechselt werden. Die Bausteine werden in der 4DIAC-Laufzeitumgebung (FORTE) ausgeführt. Quelle [H+16]

seine Zustände primär dafür, die ROS-Schnittstellen der jeweiligen Komponenten oder Skills aufzurufen. Die meisten gespeicherten Daten beziehen sich auf Wartezeiten oder einfache Daten, wie z.b. den erkannte Tür-Typ.

In dieser Umsetzung ist nur vom Top-Level Koordinator gefordert, dass dieser als IEC61499 Baustein realisiert wird, während die jeweiligen Skill-Koordinatoren etwa auch mit Python Programmen umgesetzt werden dürfen. Diese Dualität ist dadurch entstanden, dass die Skills roboternah und damit in einer Linux Umgebung ohne Echtzeitanforderungen aber mit hoher Komplexität ausgeführt wurden, während der Top-Level-Koordinator stets mit der 4DIAC Ausführungsumgebung (FORTE) ausgeführt werden sollte, um auch den Echtzeitbetrieb zu ermöglichen. Durch die eingeschränkten Möglichkeiten innerhalb der IEC61499 Umgebung und der Notwendigkeit, Datentypen fest in die Entwicklungsumge-

bung zu kompilieren, ist der Ansatz zudem vergleichsweise inflexibel.

Wichtiges Merkmal des Demonstrators ist die Möglichkeit, die zwei Optionen der Türkerkennung (Fanuc Vision System oder Lichtschranken) schnell gegeneinander austauschen zu können. Dafür wurde die ebenfalls auf 4DIAC basierende Workbench oder ein darauf aufsetzendes Web Frontend mit starker Benutzerführung (Abbildung 5.20 links) verwendet. In der Workbench werden verfügbare Software-Komponenten in einer Liste angezeigt, wobei dafür sowohl lokale Komponenten, als auch in der Cloud gespeicherte Komponenten abgefragt werden. Bis eine Komponente wirklich gewählt wurde, basiert diese Anzeige allein auf den modellierten Metadaten, also vor allem Name, Beschreibung und Typ. Der Nutzer kann eine Komponenten per Drag&Drop auf eine andere ziehen oder per Kontextmenü auf einer Komponente deren Austausch anfragen. Im zweiten Fall wir für die Abfrage der Komponenten bereits der Typ der auszutauschenden Komponente Verwendet und nur Subtypen dieser Klasse zugelassen. Sobald der Nutzer eine Komponente ausgewählt hat, wird geprüft, ob diese mit der auszutauschenden kompatibel ist, in diesem Fall also ob die die gewählte Komponente vom SW-Type *ObjectLocalization* ist und dessen Capabilities aufweist. Zudem wird geprüft, ob die gleichen Interfaces vorhanden und vom richtigen Typ für die Verbindung mit dem Rest der Lösung sind. Ist dies der Fall, kann die Komponente ausgetauscht werden. Ein Video des Vorgangs findet sich unter [HI+17].

5.2.1. Diskussion und Einordnung der Ergebnisse

Die Modellierung der Komponenten, der Austausch von Bausteinen und der eigentliche Prozess konnten in dem Experiment erfolgreich demonstriert werden. Der Demonstrator führte den Anpressvorgang über vier Tage ununterbrochen durch und wurde nur angehalten, um die laufende Lösung zu ändern, die Lokalisierungsfähigkeit auszutauschen, wenn die Sicherheit ausgelöst wurde oder um ein zweites Türmodell einzusetzen. Der Tauschvorgang verlief problemlos und konnte durch die semantische Unterstützung auch von untrainierten Bedienern durchgeführt werden. Bei BMW ist der Vorgang in 57 Sekunden abgeschlossen, der Pilotdemonstrator benötigte 76 Sekunden bei Verwendung des Vision-Systems und etwa 120 Sekunden bei Verwendung des Lichtreflexschalters. Das sind gute Zeiten, wenn man berücksichtigt, dass die Höchstgeschwindigkeit des Roboters aus Sicherheitsgründen auf 500 mm/s begrenzt war, die Software teilweise aus nicht-optimierten Standard-Komponenten besteht und dass keine Zeit- oder Bahnoptimierung durchgeführt wurde.

Mehrere Schlüsselfakten dieses Demonstrators zeigen die Vorteile des gewählten Ansatzes:

- **Austauschbarkeit und Wiederverwendbarkeit der Komponenten**
  Der Benutzer muss für eine Änderung der Lokalisierungsmethode nur eine Komponente (die Lokalisierungsfertigkeit) der Lösung austauschen. Alle anderen Komponenten bleiben unverändert und benötigen keine Einga-

ben des Benutzers. Dies ist eine wesentliche Verbesserung gegenüber anderen Entwicklungsansätzen, da nur der relevante Teil der Anwendung, also der, der sich tatsächlich ändert, ausgetauscht wird. Die gewählte Modellierungstiefe und Kapselung erscheint daher zweckmäßig.

- **Semantisches Modell ermöglicht automatisiertes Matching**
  Der gesamte Prozess den Lokalisierungs-Skill zu wechseln, erfordert nur wenige Sekunden. Auf Basis der semantischen Modellierung (Vergleiche 4.2.3) werden nur andere *LocalizationSkills* als mögliche Austauschkomponenten zugelassen, welche zudem über ein korrektes Interface verfügen. Dadurch ist ein in-place Austauschen möglich und die Wahl von unpassenden Komponenten wird verhindert. Durch die semantischen Informationen in den Modellen wird der Benutzer aktiv darauf hingewiesen werden, welche App für den gewünschten Zweck geeignet ist, was den Auswahlprozess erheblich erleichtert.

- **Fokus auf Interface und ROS hat sich bewährt**
  Wie in Abschnitt 4.2.3 genauer beschrieben, verwenden die Komponenten einen *Typ* als zentrale Modellierungseigenschaft. Der Typ legt zwar insgesamt Interfaces, Attribute und Fähigkeiten fest, hat aber insbesondere für das Interface eine Bedeutung gehabt. Die Signifikanz wurde durch das Zusammenspiel mit der ROS nur verstärkt, da dieses mit seinem Fokus auf Nachrichten zur Abstraktion zwischen Modulen automatisch einen stark interfaceorientierten Entwurfsstil präferiert. Es hat sich aber auch gezeigt, dass dadurch oft eine ausreichende Typisierung möglich ist. Einige der in dieser Lösung verwendeten Komponenten sind sehr spezialisiert, etwa die HA-Komponente für die Verbindung mit dem FANUC Vision System. Der Großteil, wie etwa das Ansteuern von IOs oder das Berechnen einer 3D-Tür-Pose, sind relativ universell und auch auf andere Ansätze übertragbar. Dadurch wird durch die Art der Modellierung auch die Erweiterung vereinfacht, etwa indem Trajektorien als Roboter- und Werkstück-unabhängige 3D-TCP-Posen definiert und auf Basis des erkannten Türmittelpunkts neu berechnet werden, anstatt die Trajektorien werkstückspezifisch zu erstellen.

- **Koordinator-Konzept verbessert die Kapselung signifikant**
  Die große Heterogenität (sowohl im Interface als auch der Verarbeitung und Abstraktion von Zuständen und Daten) von einzelnen Komponenten macht es schwer, eine einheitliche Ablaufsteuerung zu definieren. Das unter 4.2.2 beschriebene und im Experiment umgesetzte Koordinator-Konzept ist einer der Schlüssel zur erfolgreichen Kapselung solch variabler Module und komplexerer Prozesse zu einem *Skill*.

Insgesamt zeigt der Demonstrator deutlich die Fähigkeit, komplexe Systeme aus einzelnen Bausteinen zu konfigurieren und einige von ihnen schnell auszutauschen. Dies belegt insbesondere, dass der gewählte Modellierungsansatz nicht nur theoretisch valide ist, sondern darüber hinaus auch für konkrete industrielle Anwendungsfälle bei einer signifikanten Performance verwendet werden kann.

Die Modularisierung und semantische Unterstützung hat die Wiederverwendbarkeit der Komponenten deutlich erhöht und ermöglicht den einfachen Austausch über Systemgrenzen hinweg.

Gleichzeitig gibt es einige Aspekte, die sich bei der Umsetzung als nicht optimal herausgestellt haben:

- **Zu komplexe Modellierung erzeugt geringen Mehrwert**
  Die verwendete Modellierung (siehe auch 4.2.3) war sehr komplex, was einen hohen Overhead bei der Nutzung bedeutet, gleichzeitig konnte die komplexe Modellierung kaum zum Vorteil genutzt werden. Dies hat mehrere Gründe. Zum einen sind Taxonomien oder Klassifikationen wie sie für das Model benötigt werden, sehr schwierig zu erstellen, da Robotik sehr unterschiedlich eingesetzt werden kann. Vor allem, wenn unterschiedliche Domänen, Nutzer und Anwendungen beteiligt sind, wird bereits die Festlegung von Anwendungs-Klassen uneindeutig. Zum anderen ist die Festlegung auf nennenswerter Eigenschaften für Softwarekomponenten in der Robotik nicht trivial. Trotz größter Anstrengungen von Expertenteams, Informationsanalysen bestehender Softwarebeschreibungen im Internet und spezieller Software zur Erhebung solcher Eigenschaften konnten, abgesehen von Hardware-Spezifikationen wie einer Baud-Rate, nur wenig oder teilweise gar keine Parameter gefunden werden, die sich für eine vereinheitlichte Modellierung eignen würden. In der tatsächlichen Nutzung waren daher vor allen die Interface-Informationen und ein genereller Typ die am meisten verwendeten Informationen (abgesehen von Meta-Informationen wie der textuellen Beschreibung).
- **Komplexes Tooling und Infrastruktur**
  Für einen MDE Ansatz ist eine dedizierte Software für die Bearbeitung der Komponenten nahezu unumgänglich, dadurch wird die Nutzung, gerade für bestehenden Code, allerdings erschwert, da Entwickler sich nicht nur dem Tooling, sondern auch dessen Design-Paradigmen anpassen müssen. Ontologien und deren Modelle erfordern nicht nur spezielles Tooling, sondern auch eine spezielle Denkweisen etwa bezüglich A-Box und T-Box Modellen und eine konstante Synchronisierung oder die Verwendung einer zentralen Datenbasis. All diese Teile führen zu einem wenig flexiblem System mit hohem Overhead.
- **Inhomogene Abstraktion erschwert Nutzung und Modellierung**
  Das Verwenden verschiedener Abstraktionen und Modelle auf unterschiedlichen Ebenen erschwert die Erstellung und Nutzung von Komponenten. Während einige Koordinatoren als Zustandsautomat in einer Hochsprache umgesetzt wurden, wurden andere mit IEC 61499 modelliert. Während einige Komponenten eine Vielzahl an Parametern und anderen Komponenten brauchen, sind einige als stand-alone Black-Box definiert. Diese Heterogenität kann mit einem modellbasierten Ansatz, der versucht, Komponenten in eine homogene Modellierung zu bringen, nur schwer realisiert werden.

5.2.2. Beitrag zur Arbeit

Der Anwendungsfall wurde im Rahmen des ReApp-Projektes erstellt, welches es zum Ziel hatte, wiederverwendbare Softwarebausteine für die Robotik in einem App-Store bereitzustellen. Roboterteams werden dadurch zwar nicht direkt adressiert, viele der Entscheidungen bezüglich der Modellierung und Strukturierung der einzelnen Komponenten sind aber trotzdem direkt übertragbar und sind ein wichtiger Baustein für das letztendliche Konzept. Das entwickelte ***Koordinator*** **Konzept und die Kapselung von Komponenten auf Fähigkeitsbasis wurde durch das Experiment in einem konkreten industriellen Anwendungsfall verifiziert** und konnte vor allem die durch die sinnvolle Kapselung komplexer Abläufe überzeugen. Das in dem Demonstrator bestehende Problem der inhomogenen Modellierung wurde in dieser Arbeit durch den Einsatz von BTs adressiert.

Wichtigster Beitrag der Arbeiten ist die Evaluation der **komplexeren Modellierung mittels Ontologien** in einem realen Setting. Während diese Art der Modellierung akademisch interessant ist und viele Arbeiten eine solche vorschlagen, konnte der tatsächliche Mehrwert durch einen Mangel an verfügbaren Modellinformationen nicht gezeigt werden. Der aufgezeigte *Modellierungsansatz* wurden aufgrund der beschriebenen Komplexität, Ineffektivität und Modellierungsschwierigkeiten nahezu vollständig verworfen und neu konzipiert. In der daraus entstandenen **Fähigkeitsmodellierung** wurde der Fokus vor allem auf die funktionierenden Konzepte der Interfaces und generellen Typisierung gelegt, welche sich bewährt haben. Zusammen mit der Wahl von BTs als Modellierung für den Koordinator wurde dadurch die **Grundlage für die letztendlich vorgeschlagene Modellierung** gelegt, welche die Fehler einer zu breit aufgestellten Modellierung vermeidet.

Die komplexe Modellierung hat darüber hinaus auch ein umfangreiches Tooling für die Erstellung, Anzeige und Nutzung der Komponenten benötigt. Für die Definition des Koordinators musste z.b. der ECC Editor verwendet werden, der Vorwissen über den IEC61499 erfordert und in dem spezielle ROS-Datentypen explizit einkompiliert werden müssen. Die Modellinformationenen müssen zunächst durch die Ontologie unterstützt und dann in der Workbench definiert werden, was teilweise mehrere Iterationen jedes Tools erfordert hat. **Die Hürden der Komplexität des Toolings haben zu einem „pragmatischen" Online-Ansatz in der vorliegenden Arbeit geführt**. Mit dem entwickelten Web-Editor können Koordinatoren und Missionen ohne jede Installation oder Vorwissen erstellt werden. Eigenschaften von Knoten und Komponenten können schnell und einfach definiert (vgl. Abschnitt 4.5.6), live ausgetauscht und ebenfalls über den Editor modifiziert werden. Dadurch hat der in dieser Arbeit entwickelte Ansatz insgesamt eine deutlich bessere Nutzbarkeit.

5.3. Behavior Tree Shovables und Fähigkeitskosten

Das Konzept von *Shovables* und *Slots* für die verteilte Ausführung(Siehe 4.4.5) ist der in dieser Arbeit genutzte Mechanismus, um Fähigkeiten auf mehrere Roboter zu verteilen. Aufgrund der Laufzeiteinschränkungen wurden sie zu *Capabilities* und *Remote Capability Slots* weiterentwickelt, ihre Funktionsweise ist jedoch sehr ähnlich. Um die Funktionsweise der entwickelten BT Frameworks und insbesondere der *Shovables* und *Slots* zu validieren ,wird eine einfache Simulation durchgeführt. Entscheidungsgrundlage für jeden Austausch (egal ob durch *Shovables* oder *Capabilities*) zwischen den Robotern ist die Berechnung der Utility, für die im Folgenden konkrete Implementierungsbeispiele präsentiert werden.

5.3.1. Multiroboter Team mit Shovables

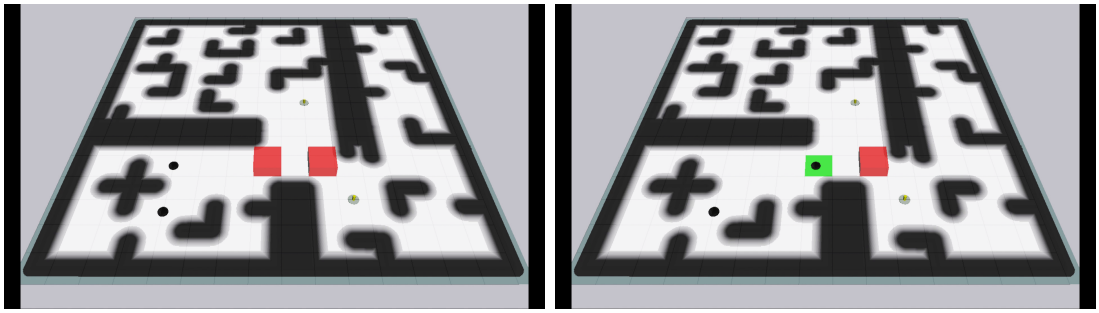


Abbildung 5.23.: Simulationsumgebung für die Tür-Simulation. Eine einfache Welt im Simple Two Dimensional Robot Simulator in der 2 TurtleBots eine Tür öffnen müssen, um zum Objekt (vgl. Abbildung 5.24) zu gelangen. Links: Beide Türen geschlossen, Rechts: Eine Tür geöffnet.Quelle [BH22]

Das entwickelte BT-Framework `ros_bt_py` implementiert die gezeigten Konzepte für die Multiroboter-Koordination. Um die Basisfunktionalität und die **Koordination mittels Shovables** zu testen, wurde eine einfache Simulation auf Basis des ROS-STDR [4] (Simple Two Dimensional Robot Simulator) erstellt (Abbildung 5.23). Der leichtgewichtige Simulator kann verschiedene Roboter und Kartierungsfunktionen auf einer 2D-Ebene simulieren. Für die Roboter wird ein TurtleBot[5] mit einem vereinfachten Kobuki-Roboter als Modell genutzt. Für die Navigation wird die Ground-Truth der Simulation verwendet, die Bewegungen werden hingegen vom *move_base* Paket durchgeführt, welches einen kompletter Planungsstack beinhaltet. Die Roboter können über 2 ROS-Services mit den simulierten Objekten interagieren, zum einen um Türen zu öffnen oder schließen

[4] `http://wiki.ros.org/stdr_simulator` Zugriff am 22.11.2023

[5] Der TurtleBot ist ein in ROS sehr gängiges Simulations bzw. Robotermodell eines extrem simplen Roboters (der Schildkröte - Turtle) mit Differentialantrieb

(Abbildung 5.24 links und Mitte) und zum anderen um Zielobjekte aufzusammeln (Abbildung 5.24 rechts). Beide Funktionen prüfen, ob der Roboter dicht genug am zu manipulierenden Objekt steht und schlagen fehl, falls nicht. Effektiv hat der Roboter dadurch 3 Fähigkeiten, die Bewegung zu einem Ziel, das Öffnen einer Tür und das Aufnehmen eines Objektes. Die `ros_bt_py` bietet für die ROS-Knoten einen binäre Utility-Berechnung die *can_execute=True* zurückgibt, wenn die konfigurierte ROS Ressource (Topic, Service oder Action) im aktuellen Namespace gefunden wird. Durch das Mappen der Services in den jeweiligen Roboter-Namespace können die Fähigkeiten dadurch für einen Roboter verfügbar gemacht werden.

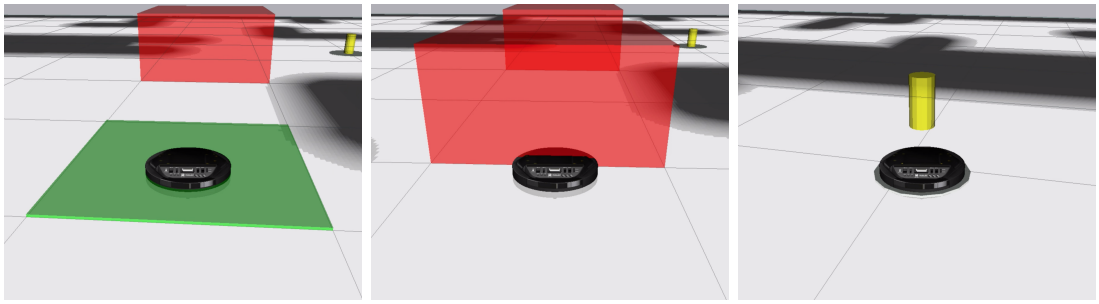


Abbildung 5.24.: Simulierte Tür im offenen (links) und geschlossenem (Mitte) Zustand sowie das simulierte Objekt im aufgenommenen Zustand. Quelle: [Heppner et al., 2023] ©2023 IEEE

Das Verhalten wurde in dem in Abbildung 5.25 gezeigten BT modelliert. Das Verhalten legt zunächst einige Parameter wie das Referenzkoordinatensystem für die Serviceaufrufe, die zu öffnende Tür und das Zielobjekt fest und ruft dann den *OpenDoorSubtree* auf, um die virtuelle Tür zunächst zu öffnen und dann den *PickupObjectSubtree*, um das Zielobjekt aufzusammeln. Der BT ist zunächst für einen einzelnen Roboter konzipiert und dann lediglich um einen *Shovable*-Dekorator für den *OpenDoorSubtree* erweitert worden, wodurch dieser *Subtree* auch entfernt ausgeführt werden kann.

Verhalten für einen Roboter:
Für einen Roboter werden alle Services in den Namespace des Roboters abgebildet, so dass dieser über alle benötigten Fähigkeiten verfügt. Nach dem Starten des in Abbildung 5.25 gezeigten BT werden die Knoten in der Sequenz schnell erfolgreich ausgeführt, da sie lediglich Parameter für die weitere Ausführung setzen. Der *Shovable*-Dekorator prüft, ob es andere Ausführungsumgebungen gibt, findet jedoch keine und hat daher keinen Effekt. Der *OpenDoorSubtree* (voller Tree im Anhang, Abbildung A.2) ermittelt die Position der Tür, prüft dann, ob diese geöffnet ist und, falls nicht, fährt den Roboter mit dem *move_base*-Planer zur Tür, bevor zuletzt der *OpenDoor*-Service aufgerufen und der *Subtree* mit einem *succeeded* beendet wird. In nahezu gleicher Sequenz wird dann der *PickupObjectSubtree* durchgeführt (voller Tree im Anhang, Abbildung A.3), welcher zum Objekt fährt und dieses mit dem *InteractObject*-Service aufhebt.

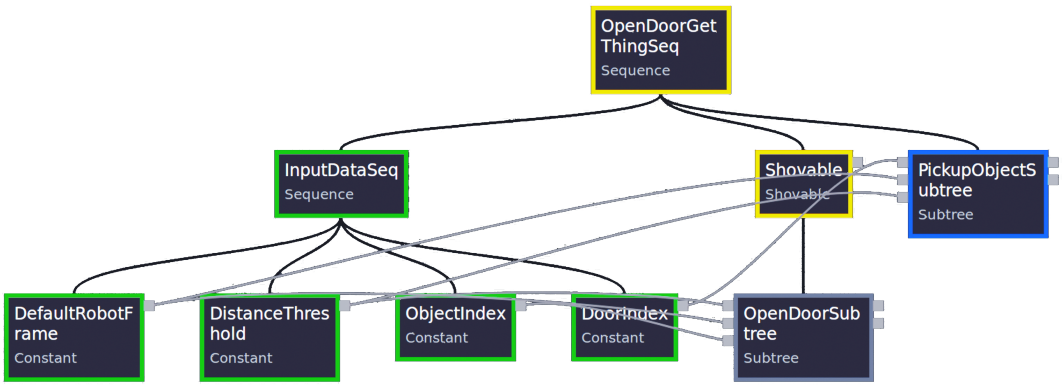


Abbildung 5.25.: Behavior Tree der Tür-Simulation. Das Öffnen der Tür (*OpenDoorSubtree*) wird durch den *Shovable*-Dekorator für die Verteilung auf andere Systeme markiert. Die Farben markieren den Ausführungszustand der Knoten: Das Shovable ist aktiv dabei eine Ausführungsumgebung zu finden, der *OpenDoorSubtree* wurde nicht initialisiert, da dies durch den shovable abgefangen wird und der *PickupObjectsSubtree* ist noch im *idle* Zustand. Quelle (modifiziert) [Heppner et al., 2023] ©2023 IEEE

Die Fähigkeiten und Mission wurden bewusst reaktiv aufgebaut, das heißt, dass bei jedem Tick der Sequenz alle Kinder ebenfalls erneut getickt werden. Durch die Prüfung ob die Tür bereits offen ist gibt der *OpenDoorSubtree* bei jedem Tick sofort ein *succeeded* zurück, sobald die Tür einmal geöffnet wurde. Wird die Tür durch den Nutzer jedoch manuell geschlossen, nachdem der Roboter bereits auf dem Weg zur Aufnahme des Objektes ist, trifft die Tür-Geöffnet-Bedingung im *OpenDoorSubtree* nicht mehr zu und der Roboter wird erneut zur Tür geschickt bzw. der Service aufgerufen. Für mindestens einen Tick gibt der *OpenDoorSubtree* daher *running* als Status zurück, woraufhin die Sequenz für alle rechts gelegenen Knoten die *untick()* Methode aufruft, welches die Ausführung des *PickupObjectSubtree* in jedem Fall stoppt. Im Verhalten bedeutet das, dass der Roboter die Tür immer offen halten will, auch wenn der Nutzer wiederholt die Tür schließt. Der Roboter wird jedes Mal zurückkehren und die Tür wieder öffnen, sobald sie nicht mehr geöffnet ist, es sei denn, der Tree wurde komplett beendet. Nachdem die Tür offen ist, wird erneut das Aufnehmen des Objektes gestartet.

Verhalten für mehrere Roboter:
Für das Roboterteam werden zwei TurtleBots in unterschiedlichen Namespaces gestartet und der *InteractObject*-Service in den ersten und der *OpenDoor*-Service in den zweiten gemappt und dadurch ein heterogenes Team erzeugt. Der erste Roboter nutzt den gleichen BT wie zuvor, der zweite startet lediglich einen *RemoteTreeSlot*.

Beide Trees werden gestartet, woraufhin der *RemoteTreeSlot* zunächst lediglich wartet, während der Haupt-BT wie zuvor ausgeführt wird. Der Shovables detektiert dieses Mal jedoch den *Slot* als mögliche Ausführungsumgebung und fragt die lokale und entfernte Utility für den *OpenDoorSubtree* an. Da der nötige Service im lokalen Namespace nicht vorhanden ist, ergibt die Auswertung der Utility ein *can_execute = False* für den lokalen und *can_execute = True* für den remote Fall, womit nur die entfernte Ausführung gewählt werden kann. Der *Subtree* wird daraufhin in den *RemoteTreeSlot geshoved*, woraufhin dieser sofort mit der Ausführung beginnt. Sobald die Tür geöffnet wurde, wird der Subtree beendet und der Haupt-Tree weiter ausgeführt, woraufhin der Roboter das Zielobjekt aufheben kann. Effektiv wurde die Mission damit auf die zwei Roboter aufgeteilt, die allerdings weiterhin strickt nacheinander arbeiten. Tatsächlich bewegt sich der erste Roboter erst, sobald der zweite die Tür erfolgreich geöffnet hat. Sollte die Tür manuell geschlossen werden, wird sie jedoch sofort wieder vom zweiten Roboter geöffnet, der erste muss also nicht umdrehen.

5.3.2. Utility Berechnungen

Das Health-System wurde auf verschiedenen Roboter umgesetzt und evaluiert. Auf LAURON V wurde ein komplexer Health-Tree implementiert (Abb. 5.26) und im Betrieb ausgewertet. Im Baum ist deutlich zu erkennen, wie die Modellierung hierarchisch strukturiert ist, aber auch verschiedene Subsysteme beeinflussen kann. *leg0* ist stellvertretend für alle Beine vollständig dargestellt und zeigt die Trennung in mechanische Einflüsse, die wiederum an die unterschiedlichen Gelenke (alpha, beta, gamma, delta) gebunden sind und den Controller, der im Fall von LAURON V aus zwei *UCOM*s [105] konfiguriert ist. An dieser Stelle wären noch viele weitere Komponenten denkbar, etwa die Kabel zu den Gelenkencodern, eine häufige Fehlerquelle. Es muss jedoch auch berücksichtigt werden, dass es eine Messvorschrift bzw. Messwerte oder Heuristiken geben muss, um den Zustand der Komponente auch tatsächlich realitätsgetreu zu ermitteln. Die Beine beeinflussen nicht nur die Lokomotion, sondern auch die Manipulation. Auch hier wären noch weitere Querverbindungen denkbar, solange eine geeignete Kombinationsvorschrift gefunden werden kann, um die Einflüsse zu berücksichtigen.

Durch die Auswertung des Health-Tree konnten gefährlich belastende Gelenkstellungen, welche für Nicht-Experten schwierig zu erkennen sind, durch das System einfach und intuitiv sowohl auf dem echten Roboter als auch auf einer AR-Visualisierung angezeigt werden (Abb. 5.27). Eine Schwierigkeit, die sich bei der Ermittlung geeigneter Risiko-Werte gezeigt hat, ist die Einschätzung dessen, was einen normalen oder guten Zustand ausmacht. Konkret z.B. die Frage, welcher Motorstrom als nominal zu werten ist und ab wann mit Beschädigungen zu rechnen ist. Dieser Umstand wurde seitdem vor allem in [Schnell et al., 2020] angesprochen, bei dem die Health-Berechnung durch den Einsatz von Gaussian

Abbildung 5.26.: Reduzierter Ausschnitt des *Health-Tree* auf LAURON V. Zur besseren Lesbarkeit wurden lediglich ausgewählte Komponenten angezeigt, jede Box stellt eine Komponente (Blätter) oder eine Subsystem (andere Knoten) dar, die Pfeile zeigen, welche Subsysteme durch die Komponenten beeinflusst werden. Es ist z.b. zu erkennen, dass die Beine nicht nur für die Lokomotion sondern auch die Manipulation eine wichtige Rolle spielen. Aber auch Software-Komponenten wie z.b. ROS können in die Berechnung einbezogen werden. Quelle: [PH18]

Mixture Models aus Sensordaten gelernt wird und ohne menschliches Training auskommt.

Abbildung 5.27.: Visualisierung der durch den Health-Tree ermittelten Werte für LAURON V. Die ermittelten Metriken werden auf dem echten Roboter durch LEDs in den Füßen angezeigt. Ein AR-Overlay zeigt die Daten mit mehr Präzision und möglichen Zusatzinformationen an. In VR können diese Informationen noch besser aufbereitet und durch virtuelle Objekte wie einen Sichtkegel erweitert werden, um das echte System möglichst auch ohne direkte Sicht einschätzen zu können. Quelle: [Heppner et al., 2019] ©2019 IEEE

Die Verwendung des Health-Systems für die Fähigkeitsauswahl wurde mit dem Flugroboter Bebop experimentell evaluiert [Heppner et al., 2019]. Das System beherrscht vom Hersteller aus eine GPS-Navigation zu vorgegebenen Wegstrecken, verfügt aber über keine weitergehende eigene Autonomie. Es wurden drei unterschiedlich lange Flugtrajektorien (kurze, mittlere und lange Strecke) definiert und als Capabilities definiert. Der Health-Tree des Systems repräsentiert neben den Rotoren des Systems vor allem die Batterie-Komponenten als kritischste Komponente mit dem größten Einfluss auf das Gesamtsystem. Während in diesem Fall eine nahezu lineare Entladekurve für die Modellierung der Batterie ausreichend genau ist, sollte angemerkt werden, dass beliebig komplexe Mess- und Abbildungsfunktionen genutzt werden können, um den Health-Wert der Batterie Komponente zu ermitteln. Das Gesamtsystem für dieses Experiment wird in Abbildung 4.32 dargestellt. Die Capabilities wurden im Skill-Tree jeweils mit Einflussfaktoren auf die Komponenten modelliert. Durch die Aggregation der Komponenten und Subsystem-Health-Werte kann ein potentieller Health-Wert nach Ausführen einer Fähigkeit und damit das Risiko R_c, also der Health-Verlust, für jede Fähigkeit ermittelt werden. Für jede der Strecken wurde ein statische Beloh-

nung festgelegt (5.1) und ein Greedy Planer zur Auswahl der nächsten Fähigkeit verwendet.

Distanz [m]	180	90	36
Belohnung	179	89	35

Tabelle 5.1.: Belohnungen für die unterschiedlichen Strecken, die das UAV zurücklegen kann. Quelle: [Heppner et al., 2019]

Wird nur der Greedy Planer genutzt (Abb. 5.28 links), versucht das UAV zwei Mal die lange Distanz zurückzulegen. Während der zweiten Durchführung reicht die Batterie jedoch nicht mehr aus und das System muss während der Ausführung landen. Wird hingegen der erwartete Health-Wert in einem, virtuellen Planungs-

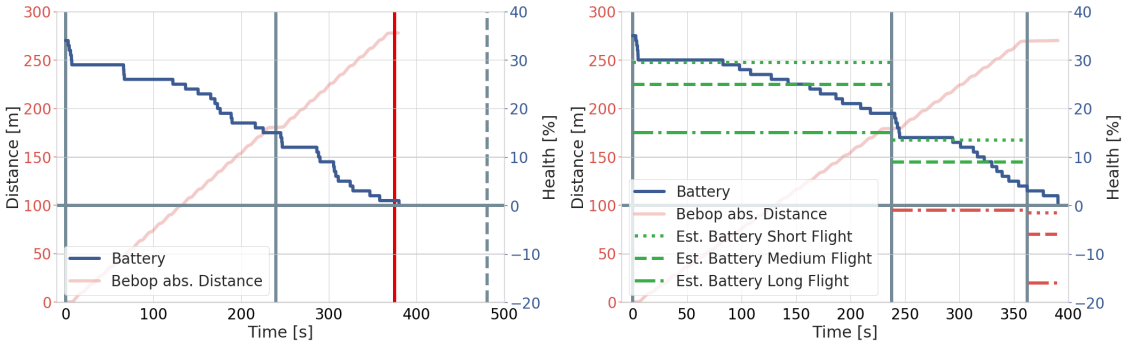


Abbildung 5.28.: Health der Bebop-Batterie mit Greedy Planner **ohne** (links) und **mit** (rechts) Rücksicht auf die Risiko-Prognose. Blaue Linie = Batterie-Health, graue Linie = Entscheidungszeitpunkte, rote Linie = System nicht mehr flugfähig/Notlandung, gestrichelte/gepunktete Linien = geschätzter Health-Wert nach Fähigkeitsausführung. Quelle [Heppner et al., 2019] ©2019 IEEE

schritt genutzt (Abb. 5.28 rechts) wird nach Ausführung der ersten Fähigkeit (erste graue Linie) erkannt, dass eine weitere Durchführung des langen Fluges nicht mehr möglich ist (rot gestrichelte Linie), allerdings die anderen Fähigkeiten noch beide möglich wären (grüne gepunktete/gestrichelte Linie). Das UAV kann den Flug mit der mittleren Distanz erfolgreich durchführen und danach am Startpunkt landen, da kein weiterer Flug mehr möglich ist. Theoretisch können beliebig viele Fähigkeiten hintereinander verkettet werden, um die Auswirkungen einer ganzen Abfolge bzw. Mission zu ermitteln. Da die Komponenten jedoch auch die aktuellen Sensorwerte berücksichtigen, ist es sinnvoller, in regelmäßigen Abständen neu zu planen, um geänderte Bedingungen während der Ausführung schneller zu berücksichtigen. So ist etwa ersichtlich, dass der Health-Wert der Batterie nicht stets mit der gleichen Geschwindigkeit sinkt, sondern von den externen Bedingungen, wie z. B. Windstärke und Windrichtung, abhängig ist.

5.3.3. Diskussion und Einordnung der Ergebnisse

Die gezeigte Simulation belegt klar, dass die **die entwickelte `ros_bt_py` Bibliothek für die Erstellung und Ausführung einer Mission genutzt werden kann** und die für die Multi-Roboter-Koordination vorgeschlagenen ***Shovables* in der Lage sind, Fähigkeiten auf andere Systeme auszulagern**. Durch die *Shovables* konnten Teile des für die Mission verwendeten BT mit minimalem Overhead auf einen anderen Roboter ausgelagert und dadurch sofort ein Multi-Roboter-Team konfiguriert werden. Die Verteilung erfolgt basierend auf den aktuell verfügbaren Interfaces und dadurch ausführbaren Fähigkeiten und kann daher nicht nur das aktuell beste System auswählen, sondern sogar heterogene Systeme mit unterschiedlichen Fähigkeiten sinnvoll berücksichtigen.

Es fallen bei dem gezeigten Beispielen einige Aspekte auf:

- **Eine Mission kann mit 1-n Robotern ohne Änderung ausgeführt werden**
  Der Wechsel von einem zu mehreren Robotern benötigt keine weitere Anpassungen außer das Vorhandensein eines *Shovables*. Durch das Integrieren von Verteilungsfunktionen wie dem *Shovable* direkt in das für die Mission genutzte BT-Framework ist keine separate Auswertung oder Anpassung der Mission oder eine separate Abbildung der Mission auf die Zielsysteme erforderlich. Teilnehmende Roboter müssen allerdings einen *Slot* ausführen.

- **Die Mission bleibt Zeitlich konsistent**
  Ein sowohl positiv als auch negativ zu bewertender Aspekt ist die strikte Einhaltung des ursprünglichen BT. Durch das Auslagern der *OpenDoor* Fähigkeit wurde, im Vergleich zum Beispiel mit nur einem Roboter, zunächst keine Zeit oder Effizienz gewonnen. Gerade für diesen Fall sollte man jedoch anmerken, dass die einzelnen Roboter gar nicht in der Lage waren, beide Teile der Mission durchzuführen und darüber hinaus identisch waren. Wäre der zweite Roboter etwa sehr viel schneller als der erste, würde ein Verteilen auch dann einen Vorteil bringen, wenn beide Systeme über alle Fähigkeiten verfügen. Dadurch, dass die Teile der Mission weiterhin in gleicher Reihenfolge durchgeführt werden, wie sie definiert wurden, wird sichergestellt, dass die ursprüngliche Intention der Mission nicht verändert wird, egal welche Systeme einen Teil der Mission übernehmen. Die Alternative wäre eine künstliche Parallelisierung, die jedoch nicht ohne Risiko ist (vergleiche 6.3.1).

- **Das System ist reaktiv**
  Trotz der Verteilung über mehrere Systeme kann die Mission weiterhin reaktiv ausgeführt werden und reagiert sofort auf geänderte Bedingungen wie z. B. das Schließen einer bereits geöffneten Tür.

- **Binäre Utilities ermöglichen bereits Koordination**
  Obwohl keine spezielle Utility-Funktion festgelegt wurde, konnte das System für die effiziente Verteilung der Aufgaben im Team genutzt werden.

Die Basis-Funktionalität, dass Interface zu prüfen ist bereits ausreichend, um Fähigkeiten basierend auf tatsächlichen Ressourcen zu verteilen.

Die gezeigten Implementierungen für die Health-Berechnung zeigen die Anwendbarkeit des Systems auf einem komplexen Laufroboter, wobei vor allem die Modellierung und die Abhängigkeiten der Subsysteme illustriert werden können. Zudem wird die Nutzbarkeit für die Fähigkeits-Selektion mit einem UAV evaluiert. Obwohl das Beispiel des UAV simpel realisiert ist, **kann der Mehrwert des Health-Systems gezeigt werden**. Während in diesem Fall grundsätzlich auch die direkte Nutzung der Batteriespannung möglich wäre, hätte das mehrere Folgen:

- Es kann eine deutlich engere Bindung zur Hardware entstehen, indem die Schwellwerte und Entladekurve für die Batterie mit in die Planungsbedingung integriert werden. Wird eine andere Batterie verwendet oder das UAV gewechselt, muss die Planungsfunktion selbst angepasst werden. Durch Verwendung des Health-Wertes wird ein unabhängigeres Maß verwendet. Die verwendeten Entladekurven oder Grenzwerte sind Teil der Komponente und nicht Teil der eigentlichen Entscheidungsfunktion.
- Die Batteriespannung ist nicht gleichbedeutend mit dem Health-Wert. Der Health-Wert kann zum einen verschiedene Messungen und andere Komponenten verwenden, z. B. die Umgebungstemperatur, zum anderen eine Beliebig komplexe Abbildungsvorschrift und fähigkeitsspezifische Abschätzungen beinhalten, etwa um die Aggressivität der Manöver mit zu berücksichtigen.

5.3.4. Beitrag zur Arbeit

Wichtigster Beitrag der gezeigten Simulation ist die **Validierung des entwickelten Behavior Tree Konzeptes generell sowie der *Shovables* und *Slots* im Besonderen**. Das Experiment zeigt, dass die umgesetzte Bibliothek die entwickelten theoretischen Konzepte (siehe 4.4) anwenden kann, um eine Robotermission zu koordinieren. Dabei wird nicht nur nativ mit ROS-Interfaces interagiert, sondern auch die integrierte Utility-Berechnung verwendet, die das Vorhandensein der ROS-Interfaces für die Berechnung nutzt. Konzepte wie *Subtrees*, *Lang laufende Prozesse*, *Parametrisierung* und die inhärente Reaktivität der BTs werden alle in der Simulation genutzt und dadurch ihre Anwendbarkeit gezeigt. Insbesondere wird jedoch die Verteilung über die Methodik der *Shovables* und *Slots* evaluiert und gezeigt, dass diese funktioniert, um die Mission aufzuteilen. Auf Basis dieser Eigenschaften wurden die *Capabilities* als Weiterentwicklung des Shovable-Konzeptes entwickelt.

Die präsentierte Evaluation der Fähigkeitsauswahl auf Basis des Health-Wertes mittels UAV belegt die Nutzbarkeit des Health-Werts als geeignete Metrik. Der komplexere Use-Case auf LAURON illustriert sowohl die Stärken, aber auch die

Schwächen des Health-Ansatzes und hat vor allem weitere Forschungsrichtungen aufgezeigt, die seitdem auch bereits durch verschiedene Ansätze bearbeitet wurden [Schnell et al., 2020, Puck et al., 2020b].

5.4. Fähigkeitsbasierte Koorperation heterogener Robotersysteme

Um die Verwendung von *Capabilites* in BTs und deren Zuweisung innerhalb eines heterogenen Teams zu evaluieren, wurde eine Simulationsumgebung erstellt, in der ein heterogenes Team eine umfangreichere Such-und-Berge-Mission durchführt. Durch das Verwenden einer Simulationsumgebung können vor allem die Zuweisungen und verschiedenen Dynamiken im Team besser analysiert werden als in einem realen Experiment. Um trotzdem möglichst dicht an einer realen Ausführung zu bleiben, wurde eine realistische Mission mittels Gazebo[6] umgesetzt. Gazebo ist ein weit verbreitetes Werkzeug für physikbasierte Simulationen in einer 3D-Umwelt mit effizienter ROS-Unterstützung. In der modellierten Welt wurde dann ein Team aus heterogenen Robotern eingesetzt, um Objekte zu finden, identifizieren und für die Dekontamination an einen externen Sammelpunkt zu bringen.

5.4.1. Die Simulationsumgebung

Als Umgebung wurde die „Cave World“ [106] verwendet, eine Open Source verfügbare Karte, welche ein Tunnelsystem aus der DARPA SUB-T Challenge nachbildet. Der Wettbewerb hatte es zum Ziel, vor allem autonome Robotersysteme und Teams aus autonomen Robotersystemen zu testen und bietet damit einen sinnvollen Grad an Komplexität für ein Roboterteam. Da die Karte einige schwierige Stellen mit Geröll oder großen Steigungen beinhaltet, die für die Koordination des Teams zunächst nicht relevant sind, wurden diese Teile für das aktuelle Experiment gesperrt.

Für die Navigation wurde initial eine statische Karte (Abbildung 5.30) mit Husky, der slam_toolbox [7] als SLAM-Stack, move_base [8] für die Navigation und dem explorer_lite paket [9] für eine autonome Exploration erstellt. Abbildung 5.31 zeigt die simulierten laserscanner und die detektierten Wandbereiche mit Expansion.

[6] `https://gazebosim.org/home` (Zugriff am 02.11.2023)
[7] `http://wiki.ros.org/slam_toolbox` (Zugriff am 02.11.2023)
[8] `http://wiki.ros.org/move_base` (Zugriff am 02.11.2023)
[9] `http://wiki.ros.org/explore_lite` (Zugriff am 02.11.2023)

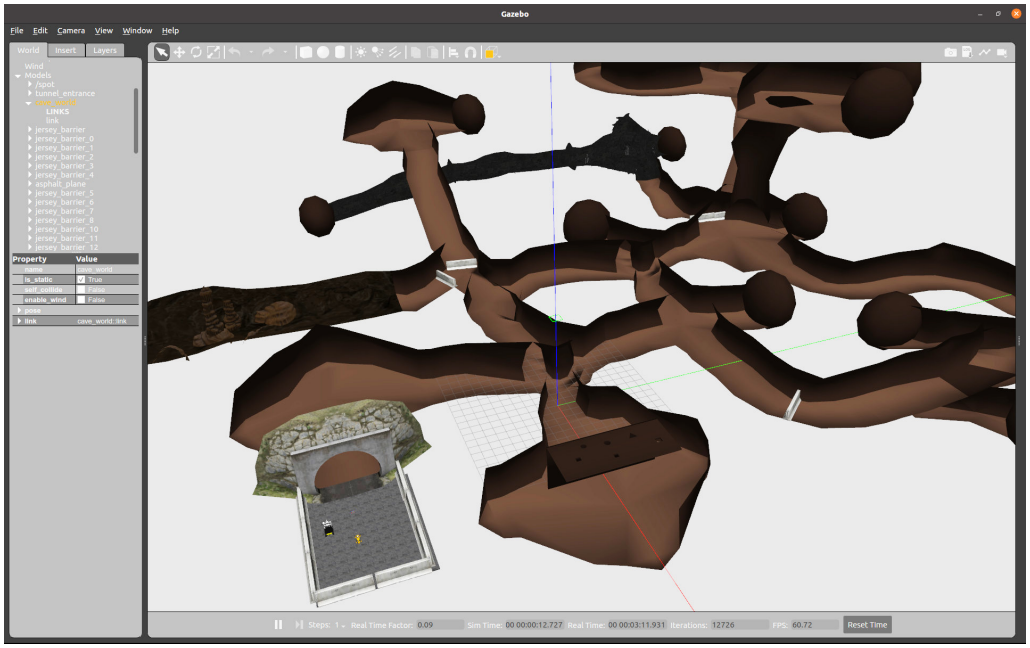


Abbildung 5.29.: Simulationsumgebung für das Roboterteam basierend auf der Open Source Cave World [106]. Die Karte wurde modifiziert, um Bereiche mit schwierigem Gelände und starken Steigungen zunächst außen vor zu lassen.

Die Karte (Abbildung 5.30) zeigt auch die „Start Area “, welche sowohl als Startpunkt, als auch als Dekontaminationszone verwendet wird sowie Explorationspunkte (rot) und Zielobjekte (grün) sowie dazugehörige Nummern, mit denen die verschiedenen Punkte identifiziert werden können.

Aufgabe in der Simulation ist die Nachbildung eines Szenario, in dem Gefahrstoffe aus einer Miene geborgen werden müssen. Das Roboterteam soll die Höhlenumgebung explorieren, kontaminierte Objekte finden und diese dann in die Dekontaminationszone außerhalb der Höhle bringen. Damit ein Objekt dekontaminiert werden kann, muss es zunächst gefunden und dann identifiziert werden. Die Mission (Abbildung 5.32) wird auf der Top-Level-Ebene durch drei Fähigkeiten modelliert: *Exploration*, *Identification* und *Decontamination*. Diese nutzen wiederum die Fähigkeiten *Explore*, *Identify* und *MoveObjectToDecontamination*. Um die Parallelität im Team ausnutzen zu können wurde die Mission von vornherein parallel ausgelegt.

Die Mission wird für die Simulation durch eine Basis-Station ausgeführt, die selbst nur über die Top-Level-Fähigkeiten verfügt und für die Ausführung aller weiteren Fähigkeiten auf die *Remote Capability Slots* der anderen Roboter angewiesen ist. Zur besseren Testbarkeit wurden die 3 Top-Level-Fähigkeiten auf der Basis-Station jeweils eigenständig ausgeführt. Jede Fähigkeit auf der Basisstation besteht aus einer Reihe an Abfragen der Vorbedingungen für das Ausführen einer konkreten Fähigkeit, bevor dann die jeweilige Fähigkeit aufgerufen wird, wodurch eine Auktionierung der Fähigkeit im Team angestoßen wird.

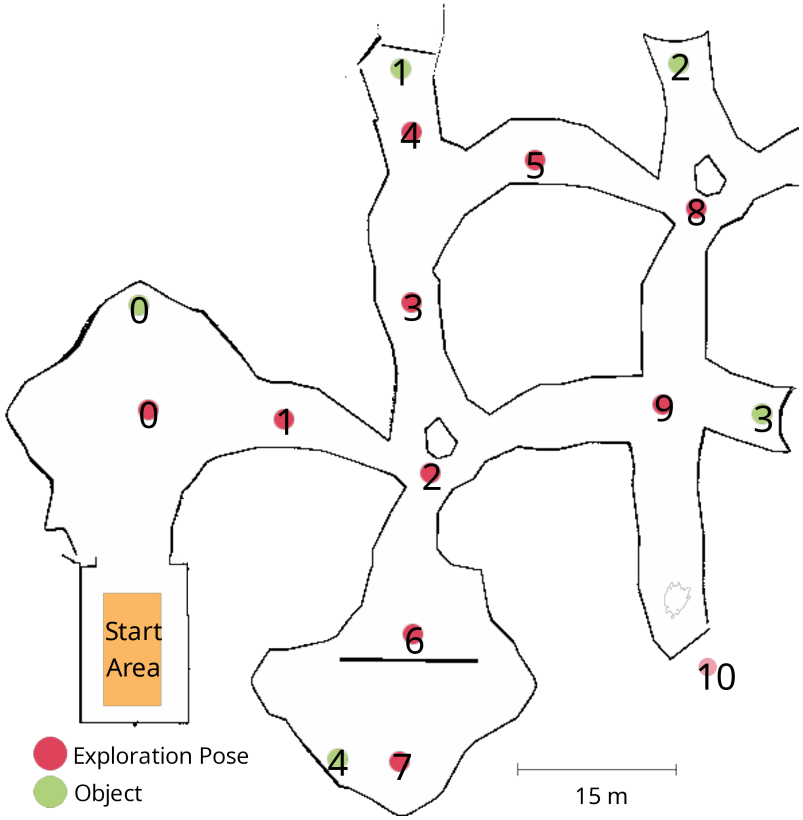


Abbildung 5.30.: Karte der Cave World mit simulierten Zielen und Explorationsposen. Alle Roboter verwenden diese Karte für ihre Navigation mit dem *move_base* Stack. Die Explorationsposen (rot) und die Zielobjekte (grün) sind nicht Teil der Karte, sondern werden während der Simulation bekanntgegeben. Die Explorationspose 10 (leicht Roter Punkt) kann nur von der Bebop erreicht werden. Quelle: (modifiziert) [OH22]

Ziel der Mission ist es, stets die gesamte Höhle zu explorieren, alle Objekte zu identifizieren und dann zu dekontaminieren. Untersucht wird dabei neben der benötigten Gesamtzeit des Teams vor allem die Zuordnung der Fähigkeiten zu den jeweiligen Systemen die Abfolge von Auktionen und Aktivitäten.

Die drei Fähigkeiten, die an die Roboter vergeben werden, sind:

- **Explore**
  Die Fähigkeit realisiert die *exploration* der Umgebung. Das heißt, der Roboter bewegt sich zu einer Zielposition und sucht die Umgebung dort nach Zielobjekten oder weiteren Explorationspunkten ab. In der Realität läuft dafür ein kompletter Objekterkennnungs- und Explorationsstack. Die Fähigkeit erhält eine Explorationspose als Input, navigiert dorthin und speichert die Ergebnisse der Exploration in die global geteilten Daten (vergleiche etwa die PoI beim SRC).
- **Indentifiy**

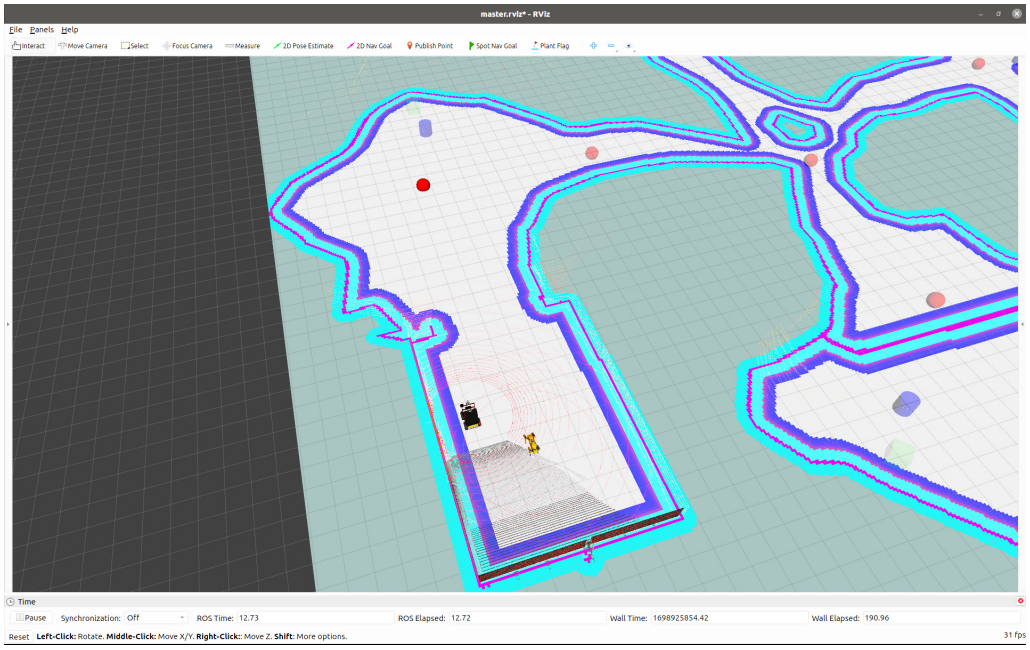


Abbildung 5.31.: RVIZ Ansicht der aus den Laserscannern generierten Karte mit Markern für die Objekte und aktuellen Sensordaten

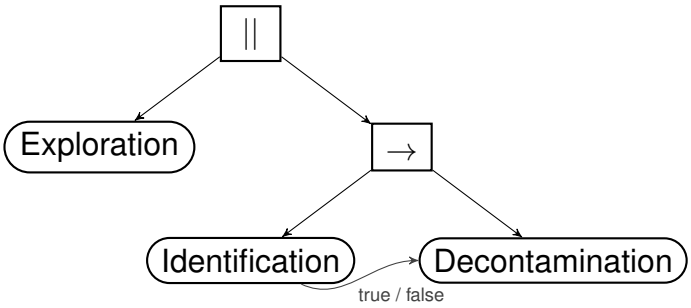


Abbildung 5.32.: Missions-BT, der für die Parallelisierung der Mission ausgelegt wurde. Jede Aktion ist eine *Fähigkeit*, die einem Roboter zugeordnet werden kann. Die Wurzel des Baums ist ein Parallel-Knoten. Identify und Move To Decontamination erfüllen zusammen die Aufgabe der Dekontamination.

Die Fähigkeit beurteilt, ob das Zielobjekt dekontaminiert werden muss. Auch diese Fähigkeit erhält eine Pose als Input, zu der der Roboter navigiert, bevor dann die Indentifikation gestartet wird. Die Fähigkeit liefert einen Boolan als Ausgabe, der angibt, ob das Objekt zur Dekontamination gebracht werden muss. In einer realen Mission würde diese Fähigkeit eine Messprozedur mit einem Spezialsensor durchführen, etwa um eine Strahlenbelastung auf dem Objekt zu identifizieren.

- **Move To Decontamination**
  Diese Fähigkeit wird genutzt, um kontaminierte Objekte in die Dekontaminationszone zu bringen. Als Input werden die Pose des Objektes und der

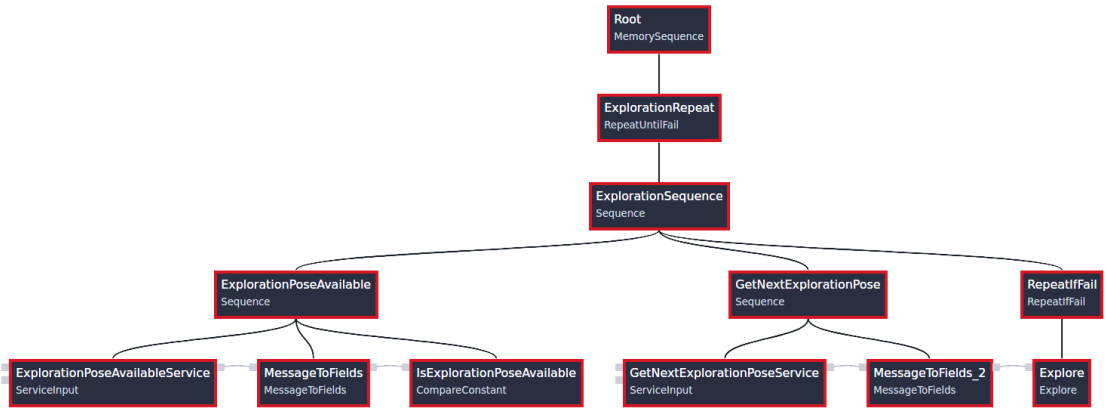


Abbildung 5.33.: Reduzierte *Exploration* Fähigkeit, die auf der obersten Ebene das Ausführen der *Explore* Fähigkeit durch Aufrufe an den *SimulationDirector* koordiniert. Das Initialisieren aller Parameter wurde für die bessere Lesbarkeit entfernt.

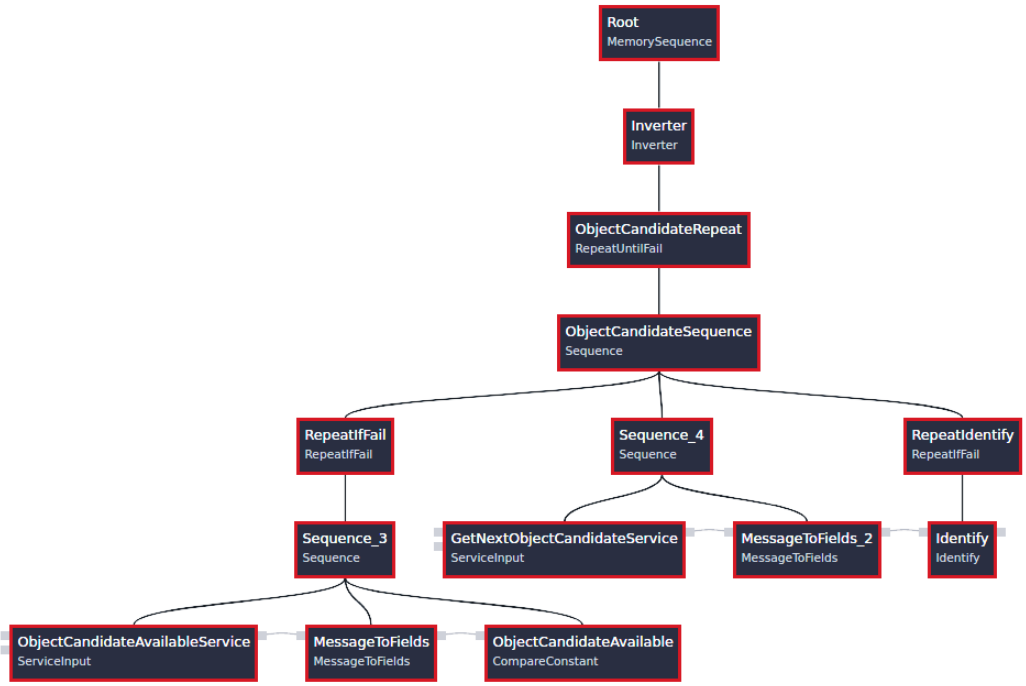


Abbildung 5.34.: Reduzierte *Identification* Fähigkeit, die auf der obersten Ebene das Ausführen der *Identify* Fähigkeit durch Aufrufe an den *SimulationDirector* koordiniert. Das Initialisieren aller Parameter wurde für die bessere Lesbarkeit entfernt.

Dekontaminationszone gegeben. Die Fähigkeit navigiert dann zum Objekt, nimmt es auf, navigiert zur Dekontaminationszone und legt es dort wieder ab. Die Fähigkeit hat keine Outputs.

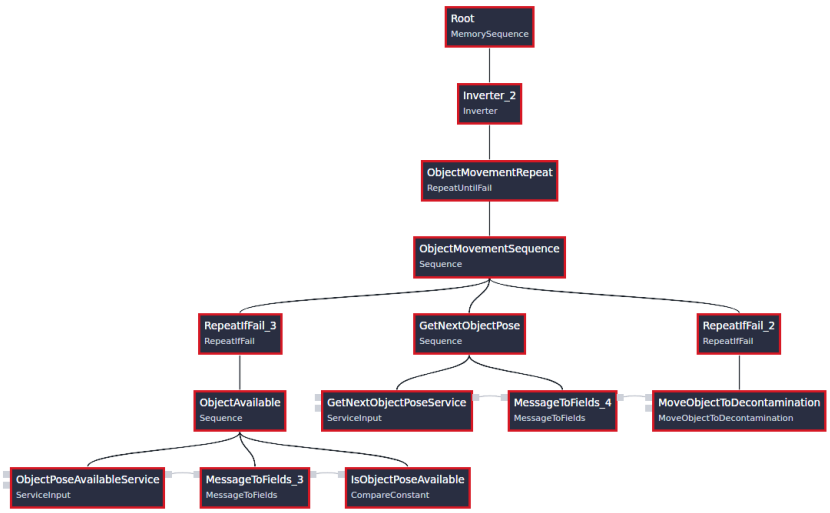


Abbildung 5.35.: Reduzierte *Decontamination* Fähigkeit, die auf der obersten Ebene das Ausführen der *Move To Decontamination* Fähigkeit durch Aufrufe an den *SimulationDirector* koordiniert.Das Initialisieren aller Parameter wurde für die bessere Lesbarkeit entfernt.

Alle Aufgaben brauchen in der realen Welt einen signifikanten Softwarestack und spezialisierte Sensoren, Greifer und Roboterarme, die noch allesamt fehleranfällig sein können. Dass es möglich ist, all dies in die BT-Fähigkeiten zu integrieren, haben die Beispiele im Weltraumkontext (siehe 5.1) bereits gezeigt. Für die Simulation wurde daher ein *SimulationDirector* erstellt, welcher die Aufgaben durch Service-Aufrufe abstrahiert und dadurch einen signifikanten Overhead in der Simulation einspart. Der *SimulationDirector* bietet folgende Aufrufe:

- **Explore** Wenn ein Roboter diesen Service aufruft, wird geprüft ob sich der Roboter an der angefragten Explorationspose befindet. Ist dies der Fall, wird für eine roboterspezifische Zeit gewartet, bevor in einem ebenfalls roboterspezifischen Radius die Explorationsposen und Objekte als erkannt markiert werden. Die erkannten Posen und Objekte können vom *SimulationDirector* angefragt werden.

- **Identify** Wie bei Explore wird geprüft, ob der Roboter sich in der Nähe des angefragten Objektes befindet. Ist dies der Fall, wird für das Objekt nach einer roboterabhängigen Wartezeit ein Rückgabewert übermittelt, in dem angegeben ist, ob das Objekt dekontaminiert werden muss oder nicht. Nur Husky und Spot dürfen den Service nutzen.

- **Pick Up & Put Down** Wie bei den anderen Services wird zunächst die Position des Roboters geprüft. Ist dieser dicht genug am Zielobjekt und wurde das Zielobjekt vorher durch ein *Explore* gefunden, wird dessen ID gespeichert und das Objekt von der Karte entfernt. Sobald das Objekt durch den Service abgelegt werden soll, wird es an der aktuellen Gazebo-Pose wieder

in die Karte eingefügt und als erkannt, aber noch nicht erledigt markiert. Ist die aktuelle Position die Dekontaminationszone, wird das Objekt als erledigt markiert und der Marker gelöscht. Nur Husky darf den Service nutzen.

5.4.2. Die Roboter

Es wurde ein heterogenes Team aus drei unterschiedlichen Robotern für die Mission genutzt. Ein Boston Dynamics Spot Laufroboter, eine Clearpath Robotics Husky Plattform und das Parrot Bebop UAV. Jeder Roboter hat seinen eigenen ROS Stack für autonomes Verhalten und wird in seinem individuellen Namespace gestartet.

Husky

Abbildung 5.36.: Husky Roboter in der Simulationsumgebung. Der radgetriebene Roboter ist sehr kräftig und bringt einen UR5 Roboterarm als Payload mit, so dass er auch Objekte transportieren kann. In der Simulation wurde der Arm nicht explizit simuliert, da die Funktionalität nicht im Fokus der Arbeit steht. Quelle: [OH22]

Husky (Abbildung 5.36) ist eine flexible und starke radgetriebene Plattform von Clearpath Robotics. Für das nativ unter ROS betriebene System gibt es eine Vielzahl an Simulationspaketen und Plugins, so dass die Default-Implementierung bereits eine Intel Realsense D415 Stereokamera für Tiefenbilder und einen Sick LMS 100 2D Laserscanner für die Navigation besitzt und GPS sowie IMU Werte liefert. Husky nutzt das *robot_localization*[10] und *amcl*[11] package für die Lokalisierung und Navigation. Als lokaler Planer wurde der *teb_local_planner* [12], als globaler der *global_planner* [13] des *move_base* Pakets [14] genutzt. Insgesamt entspricht

[10] `http://wiki.ros.org/robot_localization` Zugriff am 03.11.2023
[11] `http://wiki.ros.org/amcl` Zugriff am 03.11.2023
[12] `http://wiki.ros.org/teb_local_planner` Zugriff am 03.11.2023
[13] `http://wiki.ros.org/global_planner` Zugriff am 03.11.2023
[14] `http://wiki.ros.org/move_base` Zugriff am 03.11.2023

diese Kombination in etwa dem ROS 1 Standard für den Navigation Stack.

Das Modell verfügt auch über einen Universal Robots UR5 Leichtbauarm mit Greifer und kann zudem um einen Kontaminationssensor erweitert werden. Da Physik-Simulationen extrem schlecht im Simulieren von Kontaktkräften sind [Rönnau et al., 2013] und der Fokus der Simulation nicht darauf liegt, wurde das komplette Manipulationssystem nicht aktiv simuliert.

Fähigkeiten: Husky hat in der Simulation (wie auch bei realen Experimenten) die Rolle des flexiblen Labors, welches in der Lage ist, alle Aufgaben durchzuführen, dies aufgrund der vielen Subsysteme jedoch mit höheren Kosten tut als die anderen Systeme. Es ist zudem als einziges System in der Lage, die Manipulation durchzuführen. Es bietet die folgenden Fähigkeitsimplementierungen:

- **MoveBase** - Plant einen Pfad zum gegebenen Zielpunkt auf Basis der aktuellen Karte und führt die Bewegung sofort aus
- **Explore** - Nutzt intern die *MoveBase* Fähigkeit, um zum Explorationspunkt zu navigieren und ruft dann den *SimulationDirector Explore* call auf. Die Exploration braucht für Husky exakt 60 Sekunden und deckt Explorationspunkte und Objekte im Radius von 15 Metern auf.
- **Identify** - Nutzt intern die *MoveBase* Fähigkeit, um zum Zielobjekt zu navigieren und ruft dann den *SimulationDirector Identify* call auf. Die Identifikation dauert 60 Sekunden.
- **PickUp** - Ruft die *PickUp* Funktion des *SimulationDirector* auf, um ein Objekt neben der aktuellen Roboterpose aufzunehmen. Das Aufnehmen dauert 4 Sekunden.
- **PutDown** - Ruft die *PutDown* Funktion des *SimulationDirector* auf, um ein Objekt an der aktuellen Roboterpose abzulegen. Vorraussetzung ist das aktuelle etwas getragen wird. Das Ablegen dauert 4 Sekunden.
- **MoveToDecontamination** - Bringt ein Objekt zur Dekontamination, indem zunächst *MoveBase* genutzt wird, um das Zielobjekt zu erreichen, bevor es dann mit der *PickUp* aufgenommen wird. *MoveBase* wird erneut genutzt um zum Dekontaminationsbereich zu navigieren, bevor das Objekt mit *Put Down* abgelegt wird

Der Utility-Wert von *MoveBase* basiert auf der euklidischen Distanz zwischen aktueller und Zielpose. Für *Explore* und *Identify* wird der Utility-Wert durch die Distanz zum Objekt und der Ausführungszeit bestimmt.

Spot

Spot von Boston Dynamics (Abbildung 5.37) ist ein agiler vierbeiniger Laufroboter, der vom Hersteller aus bereits mit einer signifikanten Robustheit für schwieriges Gelände und externe Störungen konfiguriert ist. Das proprietäre System

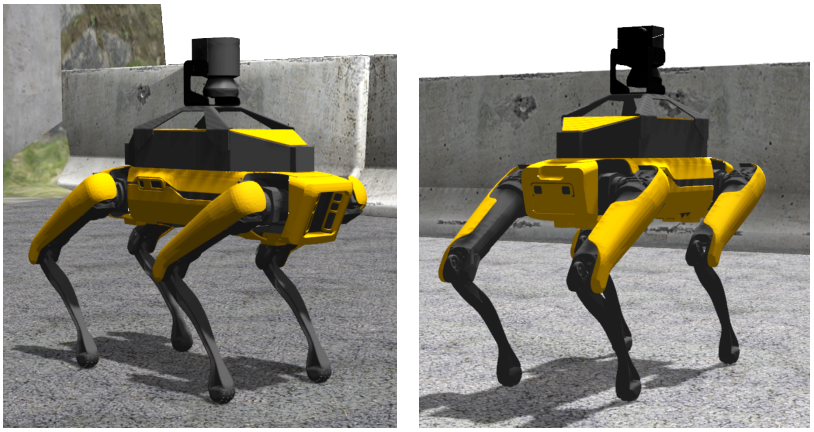


Abbildung 5.37.: Spot Laufroboter in der Simulationsumgebung. Der Laufroboter ist sehr agil und kann darüber hinaus auch in schwierigem Gelände operieren. Quelle: [OH22]

kann durch zusätzliche Payload-Komponenten modifiziert werden und bietet eine umfangreiche API um auch die internen Kameras zu nutzen. In der Simulation wird das System mit dem SICK LMS100 Laser betrieben, den auch die Husky Plattform mit sich führt. Die Software des Roboters wird vor allem durch eine modifizierte Version des Champ Pakets[15] simuliert, welches intern ebenfalls *amcl* für die Navigation und *move_base* für die Planung nutzt und damit das gleiche Interface bietet wie die Husky.

Fähigkeiten: Spot wird in der Simulation als schnelles Spezialsystem für das Freimessen der Objekte verwendet und beinhaltet daher:

- **MoveBase** - Plant einen Pfad zum gegebenen Zielpunkt auf Basis der aktuellen Karte und führt die Bewegung sofort aus.
- **Identify** - Nutzt intern die *MoveBase* Fähigkeit, um zum Zielobjekt zu navigieren und ruft dann die *Identify* Funktion des *SimulationDirector* auf. Die Identifikation dauert 2 Sekunden.

Die Utility-Werte berechnen sich identisch zu denen von Husky.

Bebop

Das Flugsystem Bebop (Abbildung 5.38) ist eine agiler Quadrocopter von Parrot, der vor allem auf den Endanwendermarkt abzielt und damit kaum autonome Fähigkeiten bietet, jedoch über eine API angesteuert werden kann. Sie kommt mit einer Weiterwinkelkamera mit Fischaugenobjektiv und ist daher ideal für schnelle Aufklärungsflüge. Als Softwarestack wird *BebopS* [16] verwendet, welches die Gazebo-Simulation des UAV erlaubt. Es bietet das Modell der Drohne und einen

[15] `https://github.com/chvmp/champ` Zugriff am 03.11.2023
[16] `https://github.com/gsilano/BebopS` Zugriff am 03.11.2023

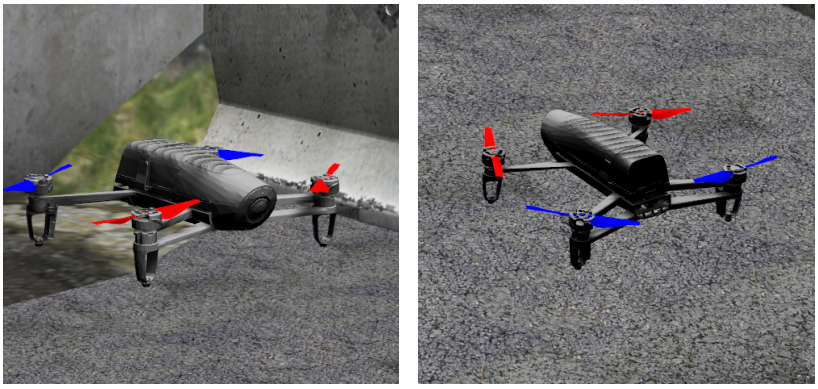


Abbildung 5.38.: Bebop Flugroboter in der Simulationsumgebung. Das UAV ist schnell und kann über die Hindernisse hinweg fliegen. Es hat eine zeitlich eng begrenzte Akkulaufzeit. Quelle: [OH22]

einfachen Navigations-Stack. Da dieser allerdings nicht mit Kollisionen umgehen kann, wurde ein eigenes *bebop_navigate* Paket entwickelt, welches die 2D-Karte für einen kollisionsfreie Navigation auf einem Höhenniveau nutzt.

Fähigkeiten:
Die Bebop wird als Spezialplattform für die Exploration eingesetzt. Zudem kann sie durch ihre Agilität sehr viel schneller zu neuen Punkten gelangen als die anderen Roboter. Sie bietet die Fähigkeiten:

- **MoveBase** - Plant einen Pfad zum gegebenen Zielpunkt auf Basis der aktuellen Karte und führt die Bewegung sofort aus.
- **Explore** - Nutzt intern die *MoveBase* Fähigkeit, um zum Explorationspunkt zu navigieren und ruft dann den *SimulationDirector Explore* call auf. Die Exploration braucht für die Bebop exakt 10 Sekunden und deckt Explorationspunkte und Objekte im Radius von 30 Metern auf.

5.4.3. Experimente mit statischem Team

In verschiedenen Experimenten mit einer festen Team-Zusammensetzung wurden die Einflüsse unterschiedlicher Roboter und dadurch Fähigkeitsverteilungen auf das Roboter-Team evaluiert. Da Husky der einzige Roboter mit der *Move To Decontamination* Fähigkeit ist, ist er Teil jeder Team-Komposition. Es wurden folgende Fälle untersucht:

- Husky - Einzelner Roboter mit allen Fähigkeiten als Referenz
- Husky1, Husky2 - Zwei fähige Roboter mit den gleichen Fähigkeiten
- Husky,Bebop, Spot - Das komplette Team mit maximaler Heterogenität
- Husky1, Husky2, Bebop - Dreierteam mit weniger Spezialisierung

Team	Missionszeit (sec)	Explore Gesamt (sec)	Explore Auktion (sec)	Identify Gesamt (sec)	Identify Auktion (sec)	Decontaminate Gesamt (sec)	Decontaminate Auktion (sec)
Husky	6492	2027	257 *12.67 %*	5647	3696 *65.45 %*	2341	922 *39.38 %*
Husky, Husky2	4056	2177	333 *15.29 %*	2371	965 *40.7 %*	2618	376 *14.36 %*
Husky, Spot, Bebop	3112	1485	348 *23.43 %*	739	50 *6.76 %*	2798	310 *11.07 %*
Husky, Husky2 Bebop	3873	1953	557 *28.52 %*	1626	126 *7.74 %*	3239	496 *15.31 %*
Husky, Husky2 Bebop (DYN)	3916	3487	586 *16.80 %*	1526	571 *37.42 %*	3008	424 *14.09 %*

Tabelle 5.2.: Zeiten die auf den unterschiedlichen Tasks verbracht wurde abhängig von den unterschiedlichen Team-Kompositionen. Die Gesamtzeit beinhaltet alle Aktivitäten, also aktive Bearbeitung und Auktionen. Die Auktionszeit ist dementsprechend nur der Teil der Gesamtzeit in der der Task vergeben wurde. Diese Zeit kann insbesondere auch die Wartezeit bis ein Roboter verfügbar wird beinhalten. Der prozentuale Wert gibt die Beziehung von Auktions- zu Gesamtzeit an, dieser Wert ist jedoch mit Vorsicht zu interpretieren. Quelle: [OH22]

In Tabelle 5.2 sind die Zeiten der unterschiedlichen Teams für einen vollständigen Missionsdurchlauf gelistet und nach den jeweilige Tasks aufgeteilt. Insbesondere durch die Physik-Simulation haben volle Missionen bis zu 6 Stunden gedauert, weshalb die Zahlen lediglich in Relation zueinander ausgewertet werden sollten.

Es zeigt sich, dass das Hinzufügen von weiteren Robotern die Gesamtmissionszeit verbessert. Nennenswert ist vor allem der deutliche Einfluss der heterogenen bzw. spezialisierten Systeme. Während 2 Huskies und ein Bebop bereits schneller sind als 2 Huskies, kann durch das volle Team aus Husky, Spot und Bebop die Zeit noch einmal reduziert werden.

Die Gesamt- und Auktionszeiten bedürfen einer kritischen Betrachtung, da in den Werten mehr als nur der Auktionsoverhead enthalten ist. In Abbildung 5.39 ist die Ausführung der Fähigkeiten durch Husky abgebildet. Die *Identify* Fähigkeit beginnt ab der ersten durchgeführten *Explore* Fähigkeit zu arbeiten, kann jedoch erst mit der eigentlichen Arbeit beginnen nachdem keine *Explore* Fähigkeit mehr vergeben wird, da sich dieser durch ihre geringeren Kosten stets gegen die anderen Fähigkeiten durchsetzt. Als einzelnes System kann Husky stets nur eine Fähigkeit gleichzeitig ausführen. Obwohl die Fähigkeit *Identify* also bereits ausführbar ist, muss sie aufgrund einer fehlenden Ausführungsumgebung noch zurückgestellt werden. Es ist zu sehen, dass insbesondere bei heterogenen Teams der prozentuale Anteil der Auktionszeit für *Identify* deutlich niedriger ist, da hier schneller eine Ausführungsumgebung zur Verfügung steht.

Ebenso wird ersichtlich, dass durch die Vergabe per Auktion immer Overheads

Team	**Missionszeit** (sec)	**Husky** (sec)	**Bebop** (sec)	**Spot** (sec)	**Husky** 2 (sec)
Husky	6492	5140 *79.17%*	-	-	-
Husky, Husky2	4056	2516 *62.03 %*	-	-	2976 *73.37 %*
Husky, Spot, Bebop	3112	2488 *79.94 %*	1137 *36.53 %*	689 *22.14 %*	-
Husky, Husky2, Bebop	3873	2167 *55.95 %*	1396 *36.04 %*	-	2076 *53.60 %*
Husky, Husky2, Bebop (DYN)	3916	2610 *66.65 %*	1058 *27.02 %*	-	2772 *70.79 %*

Tabelle 5.3.: Gesamtmissionszeit und Anteil der jeweiligen aktiven Arbeitszeit des Roboter. Aktive Arbeitszeit ist nur die Zeit, in der der Roboter wirklich etwas getan hat, also keine Auktions- oder Wartezeit. Der Prozentwert gibt die Beziehung von aktiver Arbeitszeit zu Gesamtzeit an. Quelle: [OH22]

entstehen. Ein einzelner Husky (siehe Tabelle 5.3) ist etwa nur 79% der Missionszeit dabei, eine Fähigkeit aktiv auszuführen. Auch der Blick auf den Anteil der Auktion an *Explore* für den ein Husky bzw. zwei Husky Fall zeigt, dass obwohl *Explore* nahezu durchgehend ausgeführt wird (vergleiche Abbildung 5.39 und 5.40), ein Overhead von etwa 12% besteht, der für das Zweierteam sogar noch größer wird. Gründe, dass Aufgaben trotz statischem Team nicht durchgehend bei einem Roboter verbleiben, sind zum einen die Re-Auktionsbedingung beim Überschreiten von 110% der berechneten Kosten oder das Fehlschlagen einer Fähigkeit, etwa weil die Ausführung einer Bewegung zu lang dauert, obwohl die Aufgabe noch nicht abgeschlossen ist.

Die Zuordnung der Fähigkeiten bei zwei Huskys (Abbildung 5.40) und die reduzierte Missionszeit zeigen deutlich die Fähigkeit redundante Fähigkeiten zur Effizienzsteigerung einzusetzen, wenn die Mission dies explizit berücksichtigt. *Identify* wird in diesem Fall unmittelbar nach der Auktionierung durch Husky2 bearbeitet, während Husky weiter den *Explore* Task durchführt. Fähigkeiten werden nur einmal vergeben, dadurch fährt Husky etwa nach der Identifizierung von Objekt 3 mit der Identifizierung von Objekt 2 fort, anstatt zunächst Objekt 3 zu dekontaminieren. Es zeigt sich auch bereits bei diesem homogenen Team, dass die Berechnung der Kosten einen signifikanten Einfluss auf die Vergabe der Fähigkeiten hat. Die Kosten der Fähigkeiten basieren unter anderem auf der Distanz der Roboter zu dem jeweiligen Ziel und, für den Fall der Dekontamination, zu der Dekontaminationszone. Da der Roboter für *Identify* bereits direkt vor dem Objekt steht, wird *Decontaminate* daher an den gleichen Roboter zugewiesen, da die Kosten die geringsten sind. Während Husky1 Objekt 2 Identifiziert (Sek. 2432 - 2726) wird daher das *Decontaminate* für Objekt 3 (bei Sekunde 2628) zur Auktion gestellt, aber an Husky2 vergeben, da die Kosten für Husky 1 durch den bereits

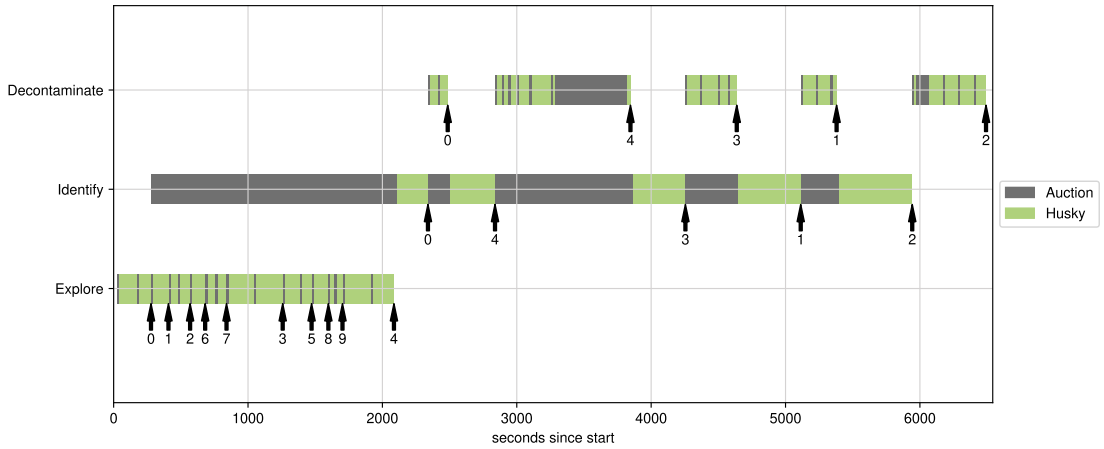


Abbildung 5.39.: Ausführung der Fähigkeiten durch einen Roboter (Husky). Es ist klar erkennbar, dass Fähigkeiten ausgeführt werden könnten (*Identify*), aber zugunsten einer günstigeren Fähigkeit erst später zugeordnet werden. Auch die Bedingung, dass *Decontaminate* erst dann durchgeführt werden, wenn ein *Identify* durchgeführt wurde, ist klar zu sehen. Farbig = Fähigkeit wird ausgeführt, grau = Auktion der Fähigkeit läuft. Die Pfeile markieren das Ende eines Tasks, die Nummer dessen ID wie auf Bild 5.30 vermerkt. Quelle: [OH22]

laufenden Task als höher berechnet werden. Wäre an dieser Stelle z.B. der Einfluss der Strecke vom Roboter zum Zielobjekt stärker gewichtet, hätte es auch passieren können, dass Husky die Dekontamination von Objekt 3 übernimmt und dafür den *Identify* Task wieder abbricht. Die Dekontamination von Objekt 2 wird zunächst an Husky2 vergeben da, dieser direkt auf der Dekontaminationszone steht. Erst nach einiger Zeit wird eine Neuberechnung angestoßen bei der Festgestellt wird das Husky der eigentlich bessere Kandidat für die Aufgabe war und der Task wird neu vergeben.

Ein Team aus Robotern mit heterogenen und spezialisierten Fähigkeiten (Abbildung 5.41) erzielt in dem gewählten Simulationsbeispiel die besten Ergebnisse. Die Missionszeit ist mit fast 52 Minuten die kürzeste Gesamtzeit von allem Experimenten, aber auch die individuelle Belastung der Systeme ist sehr viel geringer, da die Roboter aufgrund ihrer geringeren Kosten exklusiv zu ihren optimalen Aufgaben zugeordnet werden können. Husky muss aufgrund dessen, dass er der einzige Roboter mit Manipulationsfähigkeit ist, zwar wie bei seinem Einzeleinsatz immer noch nahezu 80% der Missionszeit arbeiten, verbringt diese Zeit aber ausschließlich mit dem *Decontaminate* Task und kann seine absolute Arbeitszeit im Vergleich zum Einzeleinsatz mehr als halbieren[17]. Zwar müssen

[17] In der Mission wurde Objekt 3 nicht geborgen. Ein Vergleich mit Abbildung 5.42 zeigt jedoch, dass dies lediglich eine lineare Zusatzbelastung von etwa 500s für Husky darstellen würde.

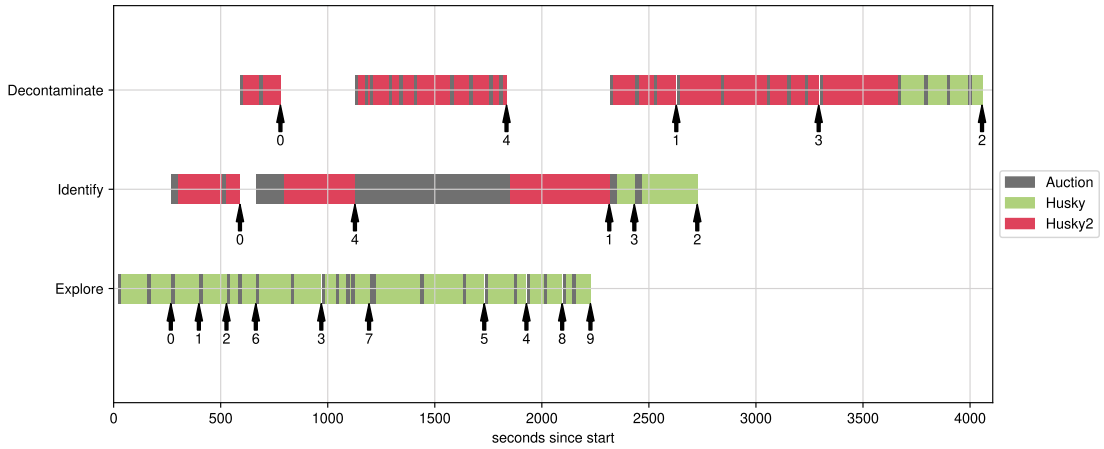


Abbildung 5.40.: Ausführung der Fähigkeiten durch zwei identische Roboter (Husky1, Husky2). Da die Mission alle Fähigkeiten parallel vergibt, können die Redundanzen in den Fähigkeiten ausgenutzt und dadurch die Missionszeit verkürzt werden. Farbig = Fähigkeit wird ausgeführt, grau = Auktion der Fähigkeit läuft. Die Pfeile markieren das Ende eines Tasks, die Nummer dessen ID wie auf Bild 5.30 vermerkt.Quelle: [OH22]

zwei weitere Roboter eingesetzt werden, aber selbst die gesamte Arbeitszeit aller Systeme ist geringer als die des einzelnen Huskys. Durch die bessere Spezialisierung werden Tasks auch nicht zwischen den verschiedenen Systemen hin- und hergegeben sondern, primär einem Roboter zugeordnet. Eine weitere Besonderheit des Teams ist das Nutzen eines Flugroboters und seiner hohen Explorationsreichweite. Es ist der einzige Roboter, der in der Lage ist, den Explorationspunkt 10 aufzudecken, welcher sich am unteren rechten Ende der Karte hinter einer Betonabsperrung befindet. Nicht nur könnten ihn die anderen Systeme nicht erreichen, sie sind aufgrund ihrer geringeren Sensorreichweite auch gar nicht in der Lage, den möglichen Explorationspunkt überhaupt aufzudecken. Der Einsatz der spezialisierten Systeme sorgt damit also nicht nur für eine schnellere Bearbeitung sondern auch für bessere Ergebnisse.

Ein Team aus zwei Huskies und der Bebop (Abbildung 5.42) zeigt ähnliche Ergebnisse wie das volle Team, indem die *Explore* Tasks ausschließlich an das spezialisierte System vergeben werden und die beiden anderen Huskys durch ihre Redundanz die übrigen Aufgaben aufteilen. Da die Husky jedoch deutlich länger für das *Identify* braucht als der vorher genutzt Spot Roboter wird die für *Identify* benötigte Zeit mehr als verdoppelt. Da Husky allerdings ohnehin länger für die Dekontamination braucht, hat dies kaum Auswirkung auf die Gesamtmission. Es zeigt sich jedoch wieder der Einfluss der Kostenberechnung auf das Zuweisungssystem: Obwohl nach dem *Identify* Aufruf für Objekt 4 der Husky direkt vor dem

An den grundsätzlichen Ergebnissen ändert dies daher nichts.

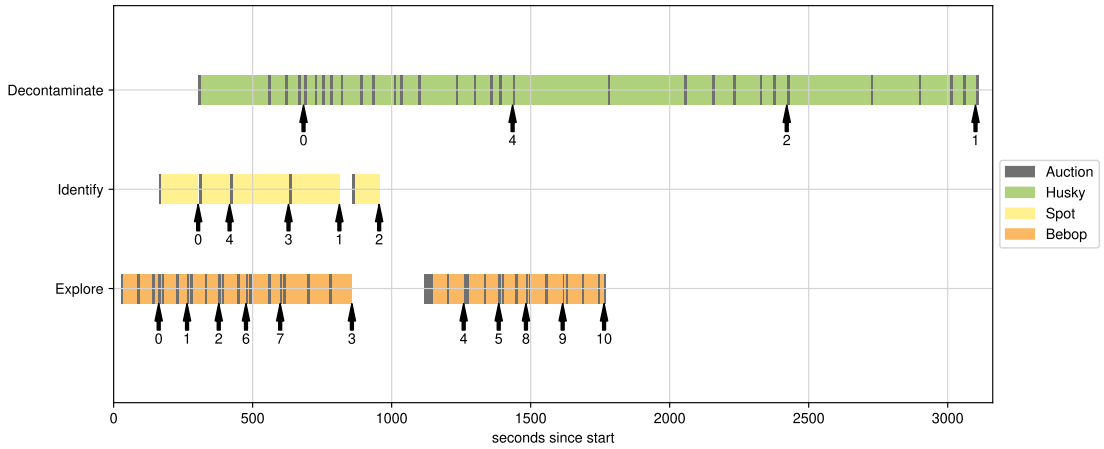


Abbildung 5.41.: Ausführung der Fähigkeiten durch ein komplettes Team aus Husky, Spot und Bebop. Durch die komplementären Fähigkeiten und Spezialiserung der Systeme ergibt sich eine optimale Verteilung der Fähigkeiten auf die Systeme und damit eine effizientere Mission. Farbig = Fähigkeit wird ausgeführt, grau = Auktion der Fähigkeit läuft. Die Pfeile markieren das Ende von Tasks, die Nummer deren ID wie auf Bild 5.30 vermerkt. Quelle: [OH22]

Objekt steht, wird die Dekontamination an Husky 2 vergeben, was in diesem Fall zu einer sichtbar längeren Bearbeitungszeit für Objekt 4 führt.

Experimente mit dynamischem Team

Der Aspekt eines dynamischen Teams, bei dem jederzeit Teilnehmer das Team verlassen oder betreten können, wird von den meisten bekannten Ansätzen nicht oder nur spärlich behandelt. Um das Verhalten in einem dynamischen Fall zu untersuchen, wurde die Simulation mit verschiedenen Teams durchgeführt, bei denen einzelne Teilnehmer durch Starten oder Stoppen ihres kompletten Stacks als Teilnehmer zum Team hinzukommen oder es wieder verlassen.

In Abbildung 5.43 ist ein Beispiel mit zwei Huskys und einer Bebop Drohne zu sehen. Zu Beginn sind lediglich zwei Husky Roboter im Team und es ist zu sehen, dass die *Explore* und *Identify* Tasks den beiden Robotern wechselnd zugeordnet werden. Ab Sekunde 1100 kommt die Bebop zum Team hinzu. Es erfolgt eine sofortige Re-Auktion und die Bebop übernimmt als Spezialsystem mit den niedrigsten Kosten sofort die *Explore*-Tätigkeit, während sich die Huskies die anderen Aufgaben teilen. Sobald die Bebop bei Sekunde 2400 das Team unerwartet verlässt, wird erneut eine Re-Auktion durchgeführt und Husky 1 übernimmt erneut den *Explore*-Task. Bei Sekunde 2900 verlässt Husky 2 das Team. Husky 1 übernimmt das *Decontaminate*, während das *Explore* unbesetzt bleibt, da es keinen an-

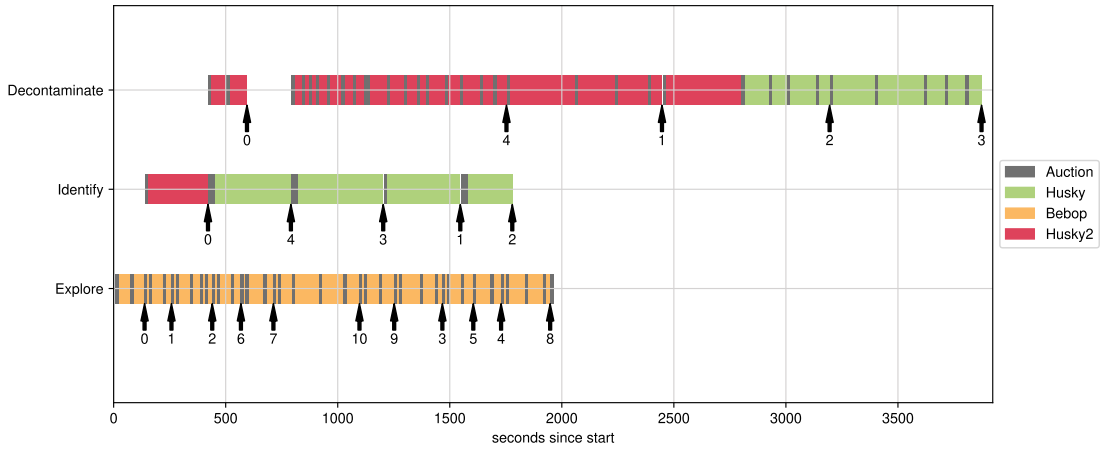


Abbildung 5.42.: Ausführung der Fähigkeiten durch ein Team aus zwei Husky und einer Bebop. Wie auch das voll heterogene Team können Aufgaben an spezialisierte Systeme vergeben werden und damit eine sehr gute Gesamtperformance erzielt werden. Aufgrund der Kostenberechnung kommt es jedoch zu unnötigen Wechseln und die Systeme sind nicht so schnell wie die Spezialsysteme. Farbig = Fähigkeit wird ausgeführt, grau = Auktion der Fähigkeit läuft. Die Pfeile markieren das Ende von Tasks, die Nummer deren ID wie auf Bild 5.30 vermerkt. Quelle: [OH22]

deren Roboter zur Ausführung mehr gibt. Wenn Husky 2 das Team bei Sekunde 3300 erneut betritt, übernimmt er den nicht vergebenen *Explore*-Task.

In dem Beispiel wird gezeigt, wie die Aufgaben dynamisch neu vergeben werden, wenn sich das Team oder die aktuelle Situation ändert. Dabei werden Fähigkeiten auch an andere Systeme vergeben, als sie es eventuell initial wurden, da die Zuordnung immer eine spontane Entscheidung basierend auf den aktuellen Kosten der Systeme ist. Diese Re-Evaluierung bei der Änderung des Teams führt zu einer großen Flexiblitität, kann jedoch auch dazu führen, dass die Qualität der Lösung insgesamt schlechter wird, wie in [107] gezeigt. Das Team kann jedoch, ohne vorheriges Wissen über die Teilnehmer oder deren Verfügbarkeit, auf jede Änderung reagieren und neue Teammitglieder sofort in die Mission integrieren, wenn diese über passende Fähigkeiten verfügen. Interessant ist darüber hinaus, dass die Missionszeit des dynamischen Teams nur minimal schlechter als die des statischen ist. Die Huskies übernehmen, wie zu erwarten, mehr Arbeit als die Bebop, da diese nicht dauerhaft im Team ist und die Fähigkeit *Explore* hat eine deutlich längere Laufzeit. Aufgrund der lang andauernden *Decontaminate*-Fähigkeit beeinflusst dies die Gesamtzeit der Mission jedoch nur wenig. Vielmehr wird der Vorteil, den die Bebop kurzfristig bringt, ausgenutzt und dadurch die Mission signifikant beschleunigt. Das Ausnutzen dieser Fähigkeitspotentiale, also die spontane Verfügbarkeit der Bebop, ist damit deutlich belegt.

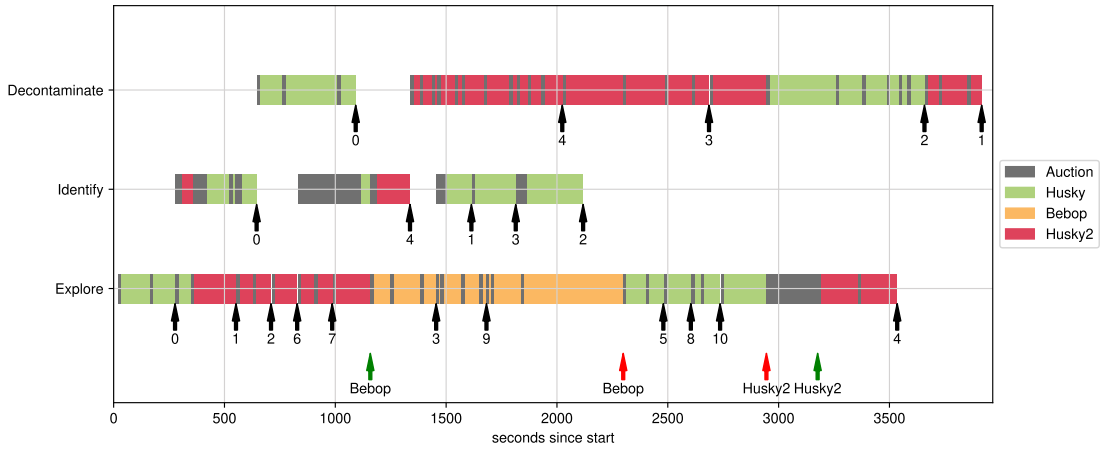


Abbildung 5.43.: Ausführung der Fähigkeiten durch ein Team von zwei Huskies und einer Bebop mit dynamischer Team-Zusammensetzung. Farbig = Fähigkeit wird ausgeführt, grau = Auktion der Fähigkeit läuft. Die Pfeile markieren das Ende von Tasks, die Nummer deren ID wie auf Bild 5.30. Ein roter Pfeil markiert, wenn ein System das Team verlässt, ein grüner wenn es das Team betritt. Quelle: [OH22]

5.4.4. Diskussion und Einordnung der Ergebnisse

Die durchgeführte Simulation und die erzielten Ergebnisse der Fähigkeitszuordnung mit verschiedenen Teams belegt, dass das vorgeschlagenen **Fähigkeitsbasierte System mit *Capabilties* und einer *marktbasierten Verteilung* dazu geeignet ist, eine komplexe Mission mit verschiedenen heterogenen Roboterteams zu koordinieren**. Es wurde gezeigt, dass ohne Änderung an der Mission oder spezifischer Modellierung der Teilnehmer eine **automatische Zuordnung der Mission auf Basis der Fähigkeiten** erfolgt ist und dass der fähigste Teilnehmer auf Basis einer lokalen Kostenberechnung ausgewählt wurde. **Es konnten erfolgreich sehr unterschiedliche Teams aus Robotern eingesetzt werden**, um die Aufgabe zu erfüllen, wobei sowohl **Redundanzen als auch Spezialisierungen ausgenutzt werden.** Die vorgeschlagene Auktion von Fähigkeiten konnte erfolgreich für die Verteilung der Fähigkeiten eingesetzt werden und hat die gewünschte Flexibilität gebracht, so dass selbst **dynamische Teams ohne explizite Modellierung berücksichtigt** werden konnten. Auch konnte durch die Experimente abermals die Modellierung der Fähigkeiten, aber insbesondere auch die ***Mission Layer* des entwickelten Frameworks und dessen Zusammenspiel mit den *Capabilites* evaluiert werden**. Die Mission wurde von der *Mission Control* ausgeführt und die *Capabilites* durch die *Task Allocation* zur Auktion ausgeschrieben. Das *Capability Management* konnte lokale Kosten berechnen und diese für das Gebot verwenden, so dass die Mission automatisch verteilt, ausgeführt und wenn nötig angepasst wurde.

Es ist klar, dass die Berechnung der Kosten für die Fähigkeiten einen signifikanten Einfluss auf das Verhalten des Teams hat. Das liegt auch daran, dass die Mission in diesem Fall für Parallelität ausgelegt wurde. Aber auch für eine lineare Mission ist der Einfluss der Berechnungsfunktion sehr relevant. Wird z.B. der aktuelle Status nicht genug berücksichtigt, kann es durchaus passieren, dass ein zweiter Roboter zu einer Aufgabe gerufen wird, die der erste eigentlich gut hätte erfüllen können. Dieser Umstand ist aber ein allgemeines Problem der Multiroboter Task Allokation, der aufgrund der nicht vollständig bekannten Welt und Modellierung nicht vollständig lösbar ist. Ebenfalls eine Frage der Performance ist die Möglichkeit der Re-Auktion bei stark gestiegenen Kosten. Arbeiten wie die von [108] setzen sich mit Re-Auktionen auseinander und zeigen, dassd dadurch insbesondere die Robustheit des Ansatzes steigt. In dieser Arbeit wurde bewusst ein flexibler Ansatz gewählt, bei dem global durchaus mit suboptimalen Lösungen zu rechnen ist, da das Ziel vor allem die Integration verschiedener Teilnehmer, nicht aber das optimale Zuweisen der Aufgaben ist.

Ein weiterer offener Aspekt der aktuellen Lösung sind die von Zlot [31] im größeren Detail betrachteten compound Tasks, also die Frage, welche Tasks bei der Vergabe gruppiert werden sollten, um eine bessere Verteilung zu ermöglichen. Prinzipiell ist die Vergabe von mehr als einem Task gleichzeitig durch das Auktionsprotokoll möglich, es wurde jedoch auch gezeigt, dass der Auktionsoverhead dadurch abermals steigt. Aktuell wird außerdem nur die Fähigkeit, nicht ihre Implementierung, auktioniert. Falls mehrere Fähigkeiten also in einem komplexeren BT kombiniert würden, müsste dieser mit übertragen werden. Wie die Frage nach Kosten und Re-Auktionsbedingungen ist auch die Frage nach Gruppierungen der Aufgaben vor allem eine Frage von Optimalität vs. Overhead und Aufwand. Ob es zielführend ist, eine solche Optimierung durchzuführen oder auch komplette Bäume zu übermitteln, bedarf weiterer Untersuchungen und stellt damit eine zukünftige Forschungsrichtung für den gezeigten Ansatz dar.

5.4.5. Beitrag zur Arbeit

Die gezeigten Experimente sind ein zentraler Beitrag, um den vorgeschlagenen Ansatz der fähigkeitsbasierten Kooperation zu validieren. Durch den Einsatz der *Capability*-Knoten konnte das Fähigkeitskonzept in einer BT-Umgebung gezeigt und dessen Vorteile validiert werden. Die Modellierung der Fähigkeiten und Mission als BT und die Verwendung von roboterspezifischen *Fähigkeitsimplementierungen* erlaubt eine **einheitliche Modellierung und Verwendung der Fähigkeiten, ohne dabei die Heterogenität der Systeme einzuschränken oder dies in der Mission explizit berücksichtigen zu müssen**. Dies stellt eine signifikante Verbesserung gegenüber existierenden Systemen dar. Auch wenn Einschränkungen bzw. Optimierungspunkte des Ansatzes, insbesondere bei der Auktionierung und Kostenberechnung, sichtbar sind, wurde gezeigt, dass die Kooperation und vor allem **Koordination im Team nicht nur grundsätzlich möglich ist, sondern der Ansatz einen deutlichen Mehrwert bietet, indem spezialisierte Systeme,**

auch dynamisch, mit in das Team integriert werden können. Die gezeigten Simulationsergebnisse sind in die reale Anwendung übertragbar, wie es die anderen gezeigten Beispiele bereits belegt haben. Insbesondere wurde durch die Experimente belegt, dass **variable Teamkompositionen und sogar dynamische Team-Zusammensetzungen jederzeit nativ berücksichtigt werden können**. Dies stellt einen signifikanter Beitrag dar, der über andere Ansätze hinausgeht.

5.5. Zusammenfassung und Fazit Experimente und Evaluation

In diesem Kapitel wurden verschiedene Experimente und Anwendungen präsentiert, in deren Rahmen Teile oder das komplette Konzept dieser Arbeit validiert und teilweise auch weiterentwickelt wurden. Die Anwendbarkeit und Funktion des Fähigkeitskonzeptes, der Behavior Trees und des Frameworks wurden in verschiedenen Weltraum-Wettbewerben unter realistischen Bedingungen unter Beweis gestellt (5.1). Die Modellierung und generell die Anwendung des Koordinator und Fähigkeitskonzeptes in einem industriellen Umfeld konnten mit dem erfolgreichen Demonstrator in 5.2 belegt werden. In 5.3 wurden die Konzepte der *Shovables* und der *Utility* mit simulierten und realen Experimenten belegt, bevor dann mit der umfangreichen Simulation in 5.4 die wichtigsten Aspekte der fähigkeitsbasierten Kooperation detaillierter evaluiert wurden.

Während durch die gezeigten Experimente die wichtigsten Eigenschaften des Ansatzes belegt wurden, konnten auch verschiedene Probleme identifiziert werden, die noch nicht endgültig gelöst sind. Im nächsten Kapitel sollen einige der kontroversen Fragestellungen mit offenen Themen weiter diskutiert werden.

6. Diskussion

Die Evaluation hat gezeigt, dass einige Entscheidungen bzw. Gegebenheiten das Verhalten des Teams stark beeinflussen. Abwägungen über diese Einflüsse sollen noch einmal zusammengefasst und diskutiert werden.

6.1. Optimalität und Kostenberechnungen

Ein in der Arbeit verfolgtes Ziel im Bereich der MRTA ist die optimale Verteilung der Aufgaben im Team zu erzielen. Dafür sind zum einen Algorithmen erforderlich, die zu einer optimalen Lösung führen, und zum anderen müssen die Kosten korrekt berechnet werden.

6.1.1. Optimale Zuordnung

Viele Verbesserungen wie komplexe Auktionsysteme, etwa mit mehrstufigen Gebotsrunden oder Preisanpassungen, Modifikationen der Marktbedingungen oder eine Bündelung von Aufgaben für die Auktion wurden entwickelt, um das Gesamtergebnis eines Teams zu verbessern. Teile solcher Entwicklungen könnten auch für die vorliegende Arbeit verwendet werden. Die Optimierungen erfordern jedoch eine geschlossene Welt oder zumindest eine vollständige Aufgabendefinition. Wenn ein dynamisches Team berücksichtigt werden soll, können stets neue Fähigkeiten hinzukommen oder verschwinden, was diese Forderung schnell zunichte macht. Viele Ansätze, und so auch diese Arbeit, nutzen daher eine iterativer Zuweisung für die dynamische Aufgabenverteilung. Dadurch sind Optimierungen zwar möglich, aber erschwert. Eine iterative Aushandlung von optimalen Lösungen erfordert einen nicht unerheblichen Aufwand.

Ziel dieser Arbeit ist das Ermöglichen der Kooperation, insbesondere unter dem Aspekt, die individuellen Lösungen nicht einzuschränken und neue Fähigkeiten und Systeme zu berücksichtigen. Diese Ausrichtung und der Anspruch, das entwickelte System jederzeit in echten Szenarien einsetzen zu können, haben dazu geführt, dass viele Entscheidungen zugunsten der Flexiblität und Nutzbarkeit des Systems und nicht seiner Optimalität getroffen wurden. Themen wie Re-Auktionen, Ringschlüsse und Kostenberechnungen wurden im Zweifel durch Heuristiken gelöst, die zunächst für die reale Anwendung handhabbar sind. Auch

komplexe Modellierungen oder Kostenberechnungen wurden zugunsten handhabbarer Schnittstellen und genereller Aussagen zur Einsetz- oder Kombinierbarkeit von Fähigkeiten vermieden.

Eine suboptimale Lösung erscheint im Vergleich zu anderen Kosten etwa der Dauer von Tasks an sich und der stattdessen aufrechterhaltenen Flexibilität vertretbar. Lediglich falsche Zuordnungen von Aufgaben an Systeme aufgrund von fehlerhaften Kostenberechnungen haben während der Evaluation ein nennenswertes Problem dargestellt.

6.1.2. Kostenberechnung

Die Berechnung der Kosten für eine Fähigkeit basiert auf der *calulcate_utility()* Funktionaltiät im BT. Ziel der Funktion ist es, unter Berücksichtigung des aktuellen Roboter und Weltzusantendes die Kosten für die Ausführung des jeweiligen Knotens zu bestimmen. Gerade der Weltzustand wird jedoch von den aktuellen Inputs der anderen Knoten definiert, welcher teilweise erst während der Ausführung der vorherigen Knoten generiert wird. Die *Capability*-Knoten wurden daher mit einem *Simulate*-Flag ausgestattet, mit dem ein virtueller *Tick* durchgeführt werden kann ohne Seiteneffekte zu erzeugen, um möglichst viele Parameter zu berechnen. Komplexere Knoten, etwa für die Berechnung eines Roboterpfades hängen jedoch stark von der zeitlichen Entwicklung ab und nutzen darüber hinaus häufig sampling basierte Verfahren, die nicht deterministisch das gleiche Ergebnis erzeugen. Auch zeitlich ausgedehnte Aktionen, wie die Bewegung entlang des Roboterpfades oder das Greifen eines Objektes, erfordern eine Planung in die Zukunft um Umstände wie z.b. eine erst später geschlossene Tür zu berücksichtigen.

Da die Kosten von Fähigkeiten einen großen Einfluss auf die Koordination im Team haben, ist ihre Berechnung wichtig. Die Komplexität beginnt jedoch, wie auch bei der Frage nach der Optimalität, schnell zu wachsen oder regelrecht zu explodieren. Das vorgeschlagene System bietet daher eine große Bandbreite an Möglichkeiten für die Berechnung der Kosten von statischer Festlegung bis hin zu komplexen Berechnungen mit simulierten Inputs, erzwingt jedoch keine explizit. Die einfachste Art der Kostenberechnung ist die Bestimmung der Ausführbarkeit, also insbesondere, ob die relevanten ROS-Schnittstellen vorhanden sind. Diese Prüfung sollte daher die Basis für Entwicklung sein. Durch die Kombinationsvorschriften müssen auch nicht alle Knoten komplexe Berechnungen für die Kosten durchführen, stattdessen sollte sich auf Kernkomponenten, wie etwa *move_base*, konzentriert werden.

6.2. Modellierung und Abbildungen

Es wurde ein stark auf Schnittstellen konzentriertes Modell der Fähigkeiten entwickelt, bei dem das Haupt-Identifizierungsmerkmal der Name der Fähigkeit ist (vgl. 4.2.3). Es stellt sich die Frage, ob eine solche Modellierung für die komplexen Anwendungen wirklich ausreichend ist. Andere Ansätze sind deutlich komplexer, oft werden Ontologien verwendet, um eine Zuordnung von Eigenschaften zu Fähigkeiten zu modellieren und darüber Beziehungen ableiten zu können.

Ein modellbasierter Ansatz wurde in dieser Arbeit (Abschnitt 4.2.3 und 4.5.5) untersucht und mit dem industriellen Demonstrator (5.2) in einer Anwendung evaluiert. Trotz der sehr viel größeren Ausdrucksstärke eines solchen Ansatzes konnten nur wenig Vorteile gegenüber einem pragmatischen Ansatz gefunden werden, da die Ontologien ihren Ursprung in der Modellierung durch Experten haben, wodurch sie oft nicht mehr abbilden als eine Typenhierarchie. Ein Vorteil findet sich in der Verwendung von verschiedenen sekundären Kriterien in Kombination mit einem Reasoner, aber auch hier ist die Mächtigkeit der semantischen Ansätze durch die verfügbare Modellierung beschränkt.

Das Matching zwischen Fähigkeiten erfordert bei der Identifikation auf Namensbasis, dass zwei Roboter ihre Fähigkeit gleich benennen. Während diese Forderung für ein eigenes Team noch erfüllbar ist, wird spätestens bei zwei disjunkten Teams eine Ambiguität unvermeidbar. Dieses Problem kann jedoch durch vergleichsweise einfache Look-Up-Tables gelöst werden. Neben einer direkten Übersetzung von Begriffen, etwa „NavigateToPoint“ anstatt „MoveToPoint“, könnten auch die in den BTs gespeicherten Informationen und Fähigkeiten genutzt werden, um eine semantische Assoziation zwischen Begriffen zu finden. Durch das Modellieren von Assoziationen mit *Meta Fähigkeiten* können solche „Übersetzungen“ oder aber darüber hinausgehende Anweisungen selbst zur Laufzeit dynamisch in das Team eingebracht werden. Vor allem in der Anwendung ist dieser Ansatz deutlich zielführender als eine erschöpfende Festlegung des Vokabulars, da Entwickler diese Assoziationen nicht vorab erstellen müssen, sondern während der Entwicklung genau dann einführen können, wenn sie wirklich gebraucht werden.

Eine wichtige Eigenschaft der natürlichen Sprache als Bezeichner von Fähigkeiten in Kombination mit den flexibel ineinander einsetzbaren BTs als Koordinator für die Fähigkeiten ist die flexible Granularität der Modellierung. Auch dieser Aspekt könnte als negativ gewertet werden, da durch die wechselnde Granularität weniger Vergleichbarkeit möglich ist und dies eine Kombination der Fähigkeiten erschwert. Dies ist jedoch nur dann der Fall, wenn die unterschiedlichen Modellierungen disjunkt sind. Gerade bei heterogenen Systemen ist die wechselnde Granularität ein Vorteil, da damit die Anforderung, verschiedenste Systeme miteinander kombinieren zu können, ohne sie in eine fixe Modellierung zu zwingen, erfüllt werden kann. Die Definition einiger *Meta Fähigkeiten*, die eine Verbindung

zwischen den unterschiedlichen Granularitäten herstellen, ist bedeutend einfacher und signifikant flexibler, als eine Fähigkeit in ein vorher strikt festgelegtes Beschreibungsschema zu zwingen.

6.3. Missionen, Parallelität und Aufgabenverteilung

Der vorgestellte Ansatz verwendet Behavior Trees, um eine Abfolge von Fähigkeiten zu definieren, die selbst wiederum als BT dargestellt sind. Durch die Nutzung der BT als hauptsächliche Modellierung ist das maximal instanziierte und ggf. über mehrere Roboter verteile Endergebnis immer *ein* großer Behavior Tree. Das gezeigte Konzept macht sich diesen Umstand explizit zu Nutze, erzeugt dadurch aber auch einige Besonderheiten, die einer weiteren Abwägung für den Einsatz bedürfen.

6.3.1. Parallelität in Missionen

Der Kontrollfluss des Missions-BT wird durch das Auslagern der Fähigkeiten an andere Roboter nicht verändert. Dadurch kommt es zu dem Effekt, dass mehrere Roboter vorhanden, aber nur einer aktiv ist. Gut sichtbar ist dieses Verhalten im Beispiel der Tür-Simulation (vergleiche Abschnitt 5.3). Roboter 1 erhält die vollständige Mission, lagert jedoch das Tür-Öffnen an den Roboter 2 aus. Bis der Roboter 2 die Tür geöffnet hat. bewegt sich Roboter 1 nicht, da die ursprüngliche Mission in einer Sequenz definiert wurde, welche die weiteren Kindknoten erst ausführt, wenn die davor liegenden im Status *Succeded* enden.

Die Erwartungshaltung bei einer *Multi-Roboter-Aufgabe* ist hingegen, dass beide Roboter simultan mit ihrer Handlung beginnen und Roboter 1 ggf. auf das Öffnen der Tür wartet. Eine Möglichkeit, das Verhalten mit den gezeigten Mitteln zu realisieren, ist es, die Mission als parallele Mission auszulegen, wie etwa im Beispiel des Abschnitt 4.5.7 oder aber bei der simulierten Multi-Roboter-Mission in Abschnitt 5.4 zu sehen. Durch das Erstellen einer parallelisierten Mission können alle Aufgaben gleichzeitig auktioniert werden, was zu einer besseren Ausnutzung der Redundanzen und damit einer effizienteren Mission führt.

Es stellt sich daher die Frage, ob Missionen automatisiert in nebenläufige Versionen umgewandelt oder erweitert werden sollten. Es wäre z. B. möglich, die Sequenz im Tür-Beispiel durch die Mission control durch einen *Parallel*-Knoten zu ersetzen. Es wäre darüber hinaus auch denkbar, einzelne Knoten in eine künstliche Parallelisierung einzusetzen. Im Simulationsbeispiel (Abschnitt 5.4) könnte etwa automatisch bestimmt werden, dass die *Explore*-Fähigkeit nicht nur einmal, sondern gleich mehrere Male auktioniert wird.

Während eine solche Umwandlung technisch möglich wäre, erfordert sie jedoch einige Voraussetzungen. Zum einen müsste sichergestellt sein, dass eine Fähigkeit auch ohne Weiteres mehrfach ausgeführt werden kann. Während *Explore* recht eindeutig parallelisierbar ist, müsste beim *Decontaminate* erst geklärt werden, ob z.B. mehrere Objekte gleichzeitig zur Dekontaminationsstelle gebracht werden dürften. Zum anderen müssen die Fähigkeiten selbst ihre nötigen Vorbedingungen prüfen. Im Beispiel der simulierten Robotermission wird durch die Mission (siehe Abbildung 5.33) explizit geprüft, ob etwa Explorationsposen vorhanden sind. Während es guter Stil ist, solche Prüfungen durch die Fähigkeit selbst zu realisieren, ist die Prüfung in der Fähigkeit keine Selbstverständlichkeit. Zuletzt müssen in parallelen Missionen mehr Bedingungen berücksichtigt werden, als dies im linearen Ablauf der Fall ist. Würde keine Tür geöffnet, sondern eine Brücke hochgehoben werden, um über einen Abgrund zu fahren, wäre dies im linearen Ablauf kein Problem. In der parallelisieren Variante müsste der Roboter, der das Objekt aufhebt, jedoch selbstständig erkennen, dass die Brücke nicht vorhanden ist, um nicht in den Abgrund zu stürzen.

Da das Ziel dieser Arbeit nicht die maximale Effektivitätssteigerung eines Teams ist, sondern sie untersucht, wie unterschiedliche Fähigkeiten in ein Team eingebunden werden können, ist die Integrität der Mission das wichtigere Kriterium, weshalb auf eine automatische Parallelisierung verzichtet wurde. Neben schwierigeren Anforderungen an die Fähigkeiten müsste auch zusätzlicher Aufwand in der Mission-Control betrieben werden, um etwa parallelisierte Abschnitte nach der Ausführung wieder in eine lineare Abfolge umzuwandeln. Insbesondere wären zeitliche Synchronisierungen, also das Fordern konkreter Reihenfolgen von Ereignissen, signifikant schwieriger, wenn die ursprüngliche Flusskontrolle nicht eingehalten wird. Der Nutzer soll daher selbst entscheiden, ob er Parallelisierung nutzen möchte oder nicht.

Eng gekoppelte Aufgaben, wie z. B. der gemeinsame Transport eines Objektes, sind mit dem vorgeschlagenen System teilweise realisierbar, es ist jedoch nicht optimal dafür geeignet. Durch die Weiterleitung des *Tick* können Aufgaben prinzipiell synchronisiert ablaufen, wenn zwei *Capabilities* parallel ausgeführt werden. Über die *Capability IO Brücke* können Daten, etwa Korrekturen der Steuerung, ausgetauscht werden. Würden zwei Systeme z. B. bei jedem *Tick* genau $1mm$ vorwärts fahren, könnten sie so eine gemeinsame Last tragen. Die Umsetzung der *Parallel*-Knoten ist dabei jedoch problematisch, da die derzeitigen Implementierungen keine deterministische Parallelität garantieren. Auch die Verteilung und Auswertung der Signale ist nicht deterministisch, hinzu kommen Ungenauigkeiten in der Ausführung der Einzelsysteme. Alle Ungenauigkeiten reichen aus, um die Anforderungen einer engen Kopplung zu verletzen. Für diese Aufgaben ist es daher sinnvoller, spezielle reglerbasierte Ansätze zu verwenden, diese könnten jedoch als Fähigkeit des Systems gestartet werden.

6.3.2. Koordination über die Umgebung

Die Koordination erfolgt im vorgestellten Ansatz explizit durch die Kommunikation der Teilnehmer, indem einzelne Fähigkeiten auktioniert werden. Zudem sind, wie im vorigen Abschnitt dargestellt, verteilte Fähigkeiten weiterhin Teil des ursprünglichen BT. Üblicherweise wird eine Mission an einen Roboter gegeben, der dann selektive Teile dieser Mission weiter geben kann.

Ein alternativer Ansatz ist die Koordination über die Missionsziele bzw. Umgebung. Beim SRC wurde ein solcher Ansatz (siehe Abschnitt 5.1) verfolgt, indem die PoI nicht nur als Information bzw. Zieldefinition für die Fähigkeiten genutzt wurden, sondern als Aufgabenqueue für alle Robotern dienen. Die Roboter verfügen über einen eigenen BT, der ihre Mission definiert und können die Bearbeitung eines PoI starten, indem sie ihn für sich beanspruchen. Andere Team-Teilnehmer sehen, dass der PoI bereits bearbeitet wird und wählen ein anderes Ziel. Der Ansatz hat offensichtliche Vorteile:

- Maximale Parallelisierung für variables Team: Aufgaben werden auf Missionsebene definiert und für alle Interessierten zur Verfügung gestellt. Es spielt keine Rolle, ob ein oder 100 Roboter die Aufgaben bearbeiten und durch ein großes Team entsteht nur ein geringfügig erhöhter Kommunikationsoverhead, da die Systeme nicht miteinander reden oder Gebote aushandeln müssen.
- Einfache Missionsdefiniton: Anstatt Aufgaben zu parallelisieren, werden die Ziele festgelegt und je nach verfügbaren Resourcen bearbeitet. Da das Abfragen und Beanspruchen der PoI eine generelle Arbeitsweise darstellt, müssen keine zusätzlichen Abfragen zur Synchonisierung der Missionsziele entwickelt werden.
- Gute Planbarkeit: Das Erstellen, Planen und Aktualisieren von Zielen ist einfacher als das Festlegen von Aktionen. Im SRC Beispiel können etwa Analyse-PoI als geometrische Struktur in der Missionsebene berechnet werden. Gerade auch die Definition von Zielen durch mehrere Teilnehmer ist einfacher.
- Natürliche Visualisierung: Die defnierten PoI können intuitiv dargestellt werden. Das Format „Führe an dieser Stelle folgende Aktion durch“ ist für Menschen intuitiv verständlich und damit gut geeignet, Missionen darzustellen.

Gleichzeitig hat ein solcher Ansatz mehrere Nachteile, weshalb er für diese Arbeit nicht gewählt wurde. Die zwei wichtigsten sind fehlender Determinismus und der Fakt, dass die Mission und Team-Fähigkeiten nicht erweitert werden können. Wie bereits in 6.3.1 beschrieben, können Abhängigkeiten wie Roboter 1 muss an Position x sein, bevor Roboter 2 eine Aktion ausführt oder zeitliche Synchonisierungen mit einer vollständigen Parallelisierung schwerer realisiert werden. Die notwendige Synchonisation muss explizit in die Zieldefinition mit

aufgenommen werden, wodurch diese wieder komplexer wird. Auch Planung, etwa zur Bestimmung von Kosten für die Ausführung einer bestimmen Aktion sind schwieriger da die Abfolgen der Aktionen durch explizites Scheduling zunächst bestimmt werden müssen. Schwerwiegender für diese Arbeit ist jedoch der Umstand, dass neue Fähigkeiten und Missionsziele nicht, oder nur schwer, im Team berücksichtigt werden. Die Roboter wählen PoI basierend auf ihren Typen und damit den gewünschten Fähigkeite. Soll eine neue Fähigkeit verwendet werden müssen auch die Auswahlkriterien der Roboter aktualisiert werden. Eine neue Fähigkeit, etwa durch einen neuen Team-Teilnehmer, kann nicht ohne Weiteres berücksichtigt werden, da diese nicht auf die definierten Missionsziele angewendet werden kann.

6.3.3. Aufgabenverteilung durch Remote Capability Slots

Die Vergabe einer Mission erfordert, dass die Systeme aktiv einen *Remote Capability Slot* zur Verfügung stellen und ihre eigene Mission während der Bearbeitung pausieren. Das vorgesehene System, um die Verfügbarkeit zu bestimmen, ist die *Mission Control* bzw. die modellierte Basis-Mission der Roboter (Abbildung 6.1). Ein kleiner, aber hoch-priorer Teil des BT sollte immer die Selbstschutzfunktionen des Roboters beinhalten, also etwa das Landen eines UAV, sobald dessen Batterieleistung nachlässt. Ist kein Notfall vorhanden, kann die eigene Mission des Systems bearbeitet werden. Erst wenn diese keine aktuellen Ziele mehr aufweist, sollte der dritte Strang, das Bereitstellen des Systems als *Remote* für andere Roboter aktiviert werden.

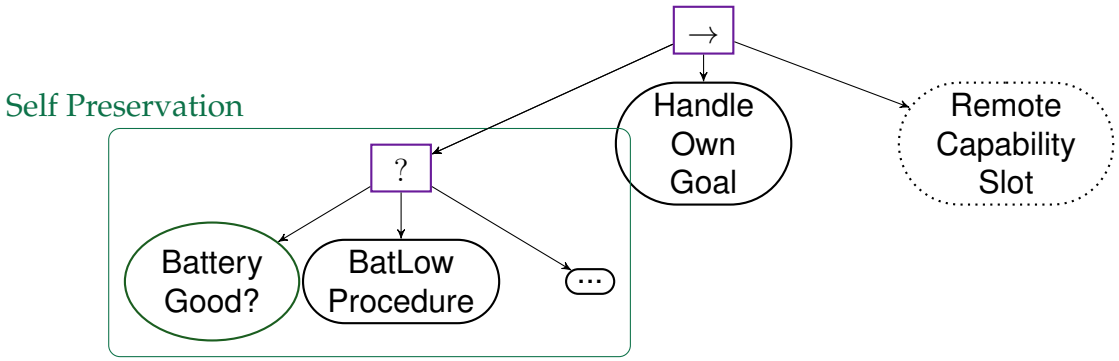


Abbildung 6.1.: Vorschlag für eine Basis-Mission für jeden Roboter, bestehend aus den Komponenten *Selbsterhalt*, *Core-Mission* und *Remote Capability Slot*.

Die eigene Mission muss dabei nicht besonders ausgeprägt sein oder könnte auch komplett fehlen, dann würde nur der *Remote Capability Slot* ausgeführt, wodurch die Missionsplanung effektiv an eine andere Instanz abgetreten wird und der Roboter lediglich *folgt*. Die Umsetzung auf diese Art hat mehrere Vorteile:

- Jedes System hat immer eine lokal ausgeführte Selbsterhaltung und weitere eigene Missionsziele, die die übernommene Mission überschreiben können.

Dadurch kann sehr flexibel auf unvorhergesehene Situationen reagiert und die Sicherheit der einzelnen Roboter sichergestellt werden.

- Jeder Roboter entscheidet selbst, wann er in der Lage ist, eine Mission anzunehmen. Zunächst durch Abwägung, ob überhaupt ein *Remote Capability Slot* angeboten wird und zusätzlich durch die eigene Gebotsberechnung. Dadurch können verschiedene Partizipationsstrategien realisiert werden, etwa, dass der Roboter periodisch prüft, ob es Angebote gibt und ansonsten seine eigene Mission ausführt.
- Es muss nicht vor der Ausführung feststehen, wer Teil des Teams ist. Durch die allgemeine Bereitstellung eines *Remote Capability Slot* kann jeder Roboter zu einem Teilnehmer des aktuellen Teams werden.
- Wird die lokale Mission komplett übergangen, kann eine effektivere Koordination im Team erfolgen (vergleiche 5.4).

Gleichzeitig erfordert diese Art der Bereitstellung auch eine starkes Bekenntnis zur Kooperation durch die Teilnehmer. Wird etwa eine umfangreiche eigene Mission durchgeführt und niemals ein *Remote Capability Slot* bereitgestellt, werden die Anfragen anderer Roboter ignoriert. Wird hingegen stets jede Anfrage beantwortet, können kaum eigene Ziele bearbeitet werden. In diese Arbeit wurden Teilnehmer in einem Team zunächst immer als reine Folger eingesetzt, d.h. jeder Roboter führt nur einen *Remote Capability Slot* aus, dadurch wird die komplette Einsatzplanung allerdings zentralisiert.

7. Zusammenfassung und Ausblick

Im **Kapitel 1** werden zunächst die Ziele dieser Arbeit motiviert und die Forschungsfrage herausgearbeitet:

Wie können heterogene Robotersysteme mit unterschiedlichem Autonomiegrad ihre Fähigkeiten effektiv koordinieren und austauschen, um sich bei der Bearbeitung komplexer Aufgaben zu unterstützen?

Die Frage fällt in die Kategorie der MRTA-Probleme. Durch bewusste Festlegungen wurden verschiedene Fragestellungen bereits adressiert, etwa dass die Kommunikation explizit und kooperativ stattfindet. Weiter eingegrenzt wurde die Fragestellung durch das Aufstellen von 4 Forschungsfragen, welche nachfolgend genauer beantwortet werden.

Im **Kapitel 2** werden wichtige Begriffe für die weitere Arbeit definiert und erläutert. Zunächst wird der Begriff Roboterteam gegenüber verwandten Begriffen wie einem Schwarm oder Multiagentensystem abgegrenzt (**2.1**), da diese zwar thematisch verwand sind, jedoch andere Fragestellungen behandeln. ROS ist ein zentraler Bestandteil aller Implementierungen und hat auch das Design des Frameworks und der Fähigkeiten maßgeblich beeinflusst, weshalb die Kernkonzepte kurz dargestellt werden (**2.2**). Eine Einführung in Behavior Trees (**2.3**) und marktbasierte Roboterkoordinaton (**2.4**) erklären Grundlagen zweier im Kern des Framework genutzter Technologien.

Im **Kapitel 3** wird der Stand der Technik analysiert. Zunächst wird die allgemeine Forschung im Bereich der MRTA und die offenen Probleme bzw. Klassifikationen vorgestellt (**3.1**). Durchgesetzt für die Einordnung hat sich die Klassifikation nach Gerkey und Matarić [5], in der sie eine Taxonomie der MRTA Problemklassen aufstellen. Diese Arbeit fällt in die Kategorie *XD (ST-SR-TA)*, da die Roboter einen Task (ST) zur Zeit ausführen und dabei explizit disjunkt (SR) vorgehen, die vergebenen Aufgaben jedoch komplexe Missionen mit zeitlichen Abfolgen (TA) und voneinander abhängigen Utilites (XD) sein können. Dadurch sind vor allem die Aufgaben *task decomposition*, *task allocation* und *optimal assignment* die relevanten Kernthemen dieser Arbeit. Bei der Betrachtung existierender Frameworks (**3.2**) zeigen vor allem ältere Arbeiten gewisse Archetypen für die Entwicklung von Multirobotersystemen, die auch für jüngere Arbeiten noch bestand haben. Ein Blick auf verschiedene Projekte und deren Ansätze für die Koordination von Robotern (**3.3**) zeigt vor allem die unterschiedlichen Ansätze zur Entkopplung der High-Level-Steuerung und der Ausführung der Fähigkeiten. Es zeigt sich dabei,

dass eine dreischichtige Architektur ein zielführendes Entwufsmuster für Architekturen dieser Art ist. Es zeigt sich weiterhin der Missstand, dass nahezu alle Systeme versuchen, alle Roboter auf einen Befehlssatz zu reduzieren, aber keine Erweiterung oder Aufbau dieses Befehlssatzes vorsehen, wodurch die Komplexität der Systeme immer beschränkt wird, ein Umstand, den diese Arbeit explizit adressiert. Behavior Trees (**3.4**) zeigen sich auch in anderen Projekten als ein probates Mittel für die Koordination komplexer Abläufe, während sie insbesondere die Möglichkeit bieten schnell ineinander eingesetzt zu werden. Vor allem Colledanchise und Ögren [64] haben mit ihren Definitionen einen wichtigen Grundstein für die Verwendung von BTs in der Robotik gelegt und sind eine wichtige Grundlage, auf denen die Entwicklungen dieser Arbeit aufbauen.

Kapitel 4 legt dann das Konzept dieser Arbeit dar. **4.1** gibt zunächst eine Gesamtübersicht der Konzepte mit denen aufgezeigten Herausforderungen adressiert werden. Es beinhaltet die Komponenten: **Roboterfähigkeiten (4.2)**, ein **Multi Roboter Framework (4.3)**, **Behavior Trees für die Modellierung (4.4)** und die Anwendung der Konzepte für die **fähigkeitsbasiere Kooperation (4.5)**. Durch die Entwicklungen können die aufgestellten Forschungsfragen beantwortet werden:

> **Forschungsfrage 1.** Wie können verschiedene Fähigkeiten über mehrere Roboter koordiniert, kombiniert oder fehlende Fähigkeiten von Robotern durch andere kompensiert werden, ohne die Fähigkeiten oder die Darstellung der einzelnen zu beschränken?

Ein Multi Roboter Framework (**4.3**) bietet eine flexible Basis, um semantisch relevante Aktionen eines Roboters als *Fähigkeiten* zu verwenden. Durch das *Skill Layer* (**4.3.2**) des Frameworks werden die Fähigkeiten zur Verfügung gestellt und gesammelt, die Granularität wird dabei frei gewählt und kann damit sowohl sehr einfache, als auch komplexe Fähigkeiten berücksichtigen. Die *Robot Layer* (**4.3.1**) abstrahiert die unterschiedlichen roboterspezifische Schnittstellen auf allgemeinere ROS-Schnittstellen, wodurch der Fokus von Fähigkeiten auf Algorithmen und nicht auf der Hardware liegt. Durch die *Mission Layer* (**4.3.3**) können diese Fähigkeiten dann zwischen den Robotern ausgetauscht bzw. koordiniert werden. Ein *Capability Management* (**4.5.2**) ermöglicht es die eigenen, aber auch entfernte Fähigkeiten zu sammeln und für das Definieren einer Mission durch die *Mission Control* (**4.5.1**) zu verwenden. Sollen die Fähigkeiten anderer Roboter für die Mission verwendet werden, so wird zunächst geprüft, ob sie geeignet sind (**4.5.5**) und falls ja mittels Auktion (**4.5.4**) von der *Task Allocation* verteilt. Jeder Roboter berücksichtigt seine lokal berechneten Fähigkeitskosten (**4.5.3**), die über die Definition als *Health Wert* auch zwischen sehr unterschiedlichen Robotern verglichen werden können. Letztendlich wird durch diesen feingranularen Austausch einzelner Missionsteile zwischen den unterschiedlichen Robotern eine Bearbeitung der Aufgaben als Team ermöglicht, womit Fähigkeiten koordiniert, kombiniert und kompensiert werden können (**4.5.7**). Ein Schlüssel für den Austausch und die Kombination von Fähigkeiten liegt in der verwendeten Fähigkeitsmodellierung.

> **Forschungsfrage 2.** Wie können sehr unterschiedliche Fähigkeiten für heterogene Systeme modelliert und diese Informationen dynamisch ausgetauscht werden, um eine einheitliche Verwendung der Fähigkeiten zu ermöglichen ?

Jede Fähigkeit wird durch 3 Bausteine definiert. Ein schnittstellenorientiertes *Modell* (**4.2.3**) beschreibt abstrakte Fähigkeiten und erlaubt es, diese zum einen als Platzhalter für die Mission (**4.5.1**) zu verwenden und zum anderen mit anderen Robotern auszutauschen (**4.5.2**). Erst durch die Zuordnung eines *Koordinators* (**4.2.2**) wird die Fähigkeit eine konkrete Fähigkeit, welche dann Funktionen in der eigentlichen *Implementierung* (**4.2.1**) aufruft. Koordinatoren werden durch einen Behavior Tree (**4.4**) modelliert. Eine neu aufgestellte Behavior Tree Definition (**4.4.1**) wurde für den Einsatz in Robotern um neue Knoten-Zustände für asynchrone Prozesse (**4.4.2**) und einen Datengraph für die Parametrisierung (**4.4.3**) erweitert und bietet eine integrierte lokale Kostenberechnung und Aggregation (**4.4.4**). Durch den Einsatz von BTs werden die komplexen Interfaces einer Fähigkeit reduziert und dadurch eine sehr viel einheitlichere Verwendung von Fähigkeiten ermöglicht. Das externe Interface wird auf das Nötigste und vor allem semantisch Relevanteste reduziert und als *Capability*-Knoten (**4.4.6**) in der Mission verwendet.

Die Zuordnung eines konkreten Koordinators nutzt die online Adaption (**4.5.6**), das heißt, dass eine Fähigkeit erst zur Laufzeit eine *CapabilityImplementation* auswählt, die nicht notwendiger weise auf dem eigenen Roboter ausgeführt werden muss. Die Implementierung ist roboterspezifisch, es kann also die gleiche Fähigkeit in einer Mission von sehr unterschiedlichen Robotern sehr individuell implementiert werden. Durch die in den BT integrierte Kostenberechnungen können Kosten für alle Fähigkeiten identisch angefragt, aber lokal berechnet (**4.5.3**) werden. Fähigkeiten können dadurch ohne Beschränkung ihrer Komplexität mit einer lokalen Implementierung umgesetzt, aber trotzdem einheitlich verwendet werden. Die flexible Ausführung ist die Grundlage für die Verteilung der Fähigkeiten.

> **Forschungsfrage 3.** Wie können die Fähigkeiten eines Teams auf die Mission abgebildet werden, ohne dabei die Flexibilität und dynamische Zusammensetzung des Teams und der Individuen einzuschränken ?

Jede Fähigkeit wird gestartet, indem ihr Koordinator (**4.2.2**) gestartet, also der entsprechende Behavior Tree *getickt* wird. Durch Entwicklung des *Capability*-Knotens (**4.4.6**) kann ein Koordinator wiederum in einem anderen Behavior Trees als Knoten verwendet werden und Fähigkeiten dadurch hierarchisch schachteln. Da eine *Capability* zunächst eine abstrakte Fähigkeit darstellt, können so auch „Rezepte“ als *Meta-Capability* definiert werden, die komplett aus anderen *Capabilities* bestehen und wie HTNs genutzt werden können, um eine komplexe Mission aufzubauen. Eine *Capability* entscheidet bei ihrem ersten *Tick*, welche Implementierung aktuell die geeignetste ist (**4.4.9**). Je nach verfügbaren Fähigkeiten des Roboters und aktuellem Weltzustand wird daher die Mission zunächst dy-

namisch „aufgelöst“ und dann roboterspezifisch ausgeführt. Durch einen *Shovable*-Dekorator können Teile des Behavior Trees außerdem transparent an andere Roboter im Netzwerk ausgelagert werden, wenn diese einen *Slot* (**4.4.5**) bereitstellen. *Capabilties* führen dieses Konzept mit den *Remote Capability Slots* (**4.4.8**) weiter und halten auch die Daten durch die *Capability IO Brücken* (**4.4.7**) konsistent. Eine Mission wird durch die *Task Allocation* unbd *Mission Control* im kompletten Team verteilt (**4.5.4**), transparent ausgeführt und während der Ausführung überwacht. Eine Änderung der Team-Komposition, neue oder geänderte Fähigkeiten können dadurch jederzeit berücksichtigt werden. Jeder Roboter bringt dadurch beliebige Fähigkeiten ein und erweitert die Möglichkeiten des Teams, ohne selbst beschränkt zu werden.

> **Forschungsfrage 4.** Wie können variable Autonomiegrade, insbesondere von lediglich teilautonomen Robotern, im Team berücksichtigt werden ?

Durch die gewählte Modellierung mit getrenntem Koordinator (**4.2.2**) und davon unabhängigen Implementierung (**4.2.1**) kann ein Roboter alle seine Funktionen oder auch nur Teile davon sehr flexibel für das Team zur Verfügung stellen. Durch die Auflösung von abstrakten Fähigkeiten zu konkreten können dabei sehr unterschiedliche Implementierungen und damit Systeme für die gleiche Aufgabe berücksichtigt werden und durch *kombinierte Fähigkeiten* (**4.5.7**) auch nur Teile der Mission beigesteuert werden. Vor- und Nachbedingungen (**4.5.5**) ermöglichen andere Fähigkeiten in einer Mission zu synthetisieren bzw. ihr Vorhandensein zu erzwingen. Zusammen mit *Meta Capabilities* kann dadurch z.B. das Starten eines Treibers für ein entferntes System mit in einer Fähigkeit berücksichtigt werden. Durch die Utility-Berechnung für die Überprüfung von Interfaces und ROS-Namespaces für das Starten einer kompletten *Robot Layer* können selbst nichtautonome Roboter flexibel in eine Mission integriert werden.

Die Verifikation dieser Konzepte wird in **Kapitel 5** durch verschiedene Experimente und Missionen erbracht. Durch die Verwendung in 3 unterschiedlichen Wettbewerben (**5.1**) konnte nicht nur die Anwendbarkeit des Fähigkeitskonzepts, des Frameworks und der Behavior Tree Bibliothek unter realen und komplexen Bedingungen validiert, sondern auch eine zeitliche Entwicklung einzelner Aspekte und damit verbundener Verbesserungen gezeigt werden. Der Modellierungsansatz und Koordinator wurden in einem industriellen Kontext auf einem realen Demonstrator von BMW evaluiert (**5.2**). Dabei konnte die Verwendung von Ontologien als Modellierungswerkzeug als nicht zielführend herausgestellt werden, was zu einem pragmatischeren Interface-Ansatz geführt hat. Die Vorteile der Behavior Tree Bibliothek wurden in Verbindung mit dem *Shovables* in einem Multi-Roboter-Experiment gezeigt (**5.3**), welches die reaktiven Aspekte und dynamische Verteilung von Teilen des BT ohne Änderung der Mission demonstriert. In Abschnitt **5.3** werden auch reale Beispiele für die Health-Wert-Berechnung auf dem Laufroboter LAURON und auf dem Health-Wert basierte Fähigkeitsauswahl gezeigt. In einer umfangreichen Gazebo Simulation eines heterogenen Multi-Roboter-Teams (**5.4**) wurde die fähigkeitsbasierte Kooperation und vor allem auch die Vergabe der Aufgaben im Team analysiert und gezeigt, dass das

System für die Koordination eines Teams geeignet ist. Insbesondere wird auch der Aspekt eines dynamischen Teams und die Fähigkeit der entworfenen Methode damit umzugehen präsentiert.

Die erzielten Ergebnisse führen zu Diskussionspunkten, von denen die wichtigsten in **Kapitel 6** aufgegriffen werden. Hier wird auf die Fragen der nicht vorhandenen Optimalität in der Zuweisung (**6.1**), dem Pro und Contra der gewählten Modellierung (**6.2**) und der Frage nach automatischer Parallelität in den Missionen (**6.3**) eingegangen.

7.1. Beitrag der Arbeit

Roboter erlauben uns bereits heute sehr präzise oder effizient zu sein, sie nehmen dem Menschen gefährliche und unliebsame Aufgaben ab und erlauben uns sogar, fremde Planeten zu erforschen. Durch stetige technische Neuerungen und fortschrittlichere KI-Systeme, die gleichzeitig immer günstiger eingesetzt werden können, gibt es eine enorme Proliferation hochspezialisierter Systeme. Um das volle Potential aller Roboter nutzen zu können, wird es bald nicht mehr reichen, nur ein System zu betrachten. Es ist daher unabdingbar, geeignete Systeme für die Kooperation dieser heterogenen Robotersysteme zu entwickeln, was zur **Zielstellung dieser Arbeit** geführt hat:

> *"Ziel dieser Arbeit ist es, Roboter dazu zu befähigen, ihre eigenen Fähigkeiten zu verstehen und diese flexibel mit anderen Robotern zu kombinieren oder diesen zur Verfügung zu stellen, um so das gesamte Potential eines Roboterteams für eine gemeinsame Mission zu nutzen (...)"*

Dieses Ziel wurde durch den in dieser Arbeit entwickelten Ansatz für die **fähigkeitsbasierte Kooperation von Heterogenen Robotersystemen** vollständig adressiert. Dabei wurden alle Forschungsfragen sowohl konzeptuell beantwortet, als auch durch konkrete Experimente in ihrer Wirksamkeit belegt. Das entwickelte Behavior Tree Framework (4.4.10) wurde als Open Source Paket veröffentlicht und wird aktiv in verschiedenen Projekten verwendet. Weitere Beiträge dieser Arbeit sind insbesondere:

- **Eine Modellierung von Fähigkeiten als Kombination aus leichtgewichtigem Meta-Modell und einem Behavior Tree Koordinator**
  Die Modellierung erlaubt es, Fähigkeiten effizient und ohne großen Overhead zu kapseln und flexibel für Missionen einzusetzen. Die Wirksamkeit dieses Ansatzes wurde in zahlreichen Experimenten über verschiedene Domänen belegt.
- **Eine neue Behavior Tree Definition als Grundlage für die Missions- und Fähigkeitsdefinition von Robotern**
  Die Behavior Tree Definition ermöglicht nicht nur die Modellierung von Fähigkeiten und deren transparente Nutzung als Baustein für eine Mission,

sondern bietet darüber hinaus wichtige Eigenschaften, die über andere BT Ansätze hinaus gehen:

  - **Parametrisierung, Asynchronität und ROS-Support** - die BTs wurden insbesondere für den Einsatz auf Robotern entworfen und integrieren sich leicht in das existierende Ökosystem
  - **Individuelle Implementierungen einer Fähigkeit** - erst zur Laufzeit steht fest, welcher Roboter eine Fähigkeit ausführt. Dieser lädt seine individuelle Implementierung, wodurch auch heterogene Teams ohne Beschränkung ihrer Komplexität berücksichtigt werden können.
  - **Utility-Berechnungen** - durch das Bestimmen individueller Kosten in einer Fähigkeit entscheidet jeder Roboter selbst, ob und für welchen Preis er in der Lage ist, eine Fähigkeit auszuführen.
  - **Shovables,Slots, Capabilites und Remote Capability Slots** - die Möglichkeit zur Verteilung von Teilen einer Mission oder Fähigkeit auf andere Systeme ist eine Grundfunktionalität des Ansatzes, kein Add-On.

- **Ein Framework für die dynamische Koordination von Fähigkeiten im Team**
  Fähigkeiten werden gesammelt, zu anderen Systemen kommuniziert und per Auktion verteilt und kombiniert. Dabei wird insbesondere die Komplexität der Systeme nicht beschränkt und neue Fähigkeiten oder geänderte Voraussetzungen in einem Team dynamisch berücksichtigt.

Durch die entwickelten Lösungen ist es möglich, alle im Team verfügbaren Fähigkeiten dynamisch für eine Mission zu verwenden und damit auch neue oder geänderte Team-Kompositionen jederzeit zu berücksichtigen. Der Fokus auf Fähigkeiten als Koordinationselement, welches jedoch individuell ausgeführt werden kann, erlaubt es, auch hochspezialisierte Systeme und neue oder geänderte Fähigkeiten auch nachträglich zu berücksichtigen und so das gesamte Potential aller verfügbaren Roboter auszunutzen.

7.2. Ausblick

Die in der Arbeit entwickelten Konzepte und die entsprechende Umsetzung sind für das Ermöglichen der Kooperation heterogener Systeme auf Fähigkeitsbasis zu verstehen und stellen eine solide Grundlage für weitere Entwicklungen dar. Durch die Entwicklungen wurden die Lücken im aktuellen Stand der Technik adressiert.

Da Optimalität für den realen Einsatz eine wichtige Rolle spielt, sollten die umfangreich erforschten Fragen der Optimalität bei der Aufgabenverteilung, etwa durch komplexere Auktionssysteme, in den weiteren Entwicklungen für die gezeigten Konzepte adaptiert werden.

Da die Kostenberechnung immer wieder zu Schwierigkeiten geführt hat, sollten auch Arbeiten zur automatisierten oder adaptiven Bestimmung von Kosten, wie sie bereits für die Health-Wert-Bestimmung entwickelt wurden, durchgeführt werden. Da der Health-Wert aktuell in verschiedenen Arbeiten zu einer umfassenderen *Risk Awareness* inklusive Umgebungseinschätzung und automatischer Bestimmung erweitert wird, könnte es zielführend sein, diesen Wert als Standard-Utility für alle Funktionen zu verwenden.

Aktuell werden die Fähigkeiten vor allem durch die Kombination mit anderen Systemen erweitert, wobei davon ausgegangen wird, dass eine mehr oder weniger ähnliche Sprache gesprochen wird. Gerade jedoch durch die Ausrichtung auf dynamische Teams wäre die Entwicklung eines „Sensei[1]“ äußerst vielversprechend. Dieser könnte durch Übertragen der Koordinatoren Meta-Capabilites als Fähigkeiten speichern und an andere Roboter verteilen, wodurch diese effektiv neue Fähigkeiten erlernen können. Ebenso könnte der Sensei die Übersetzung von unterschiedlichen Fähigkeitsbezeichnungen übernehmen. Da diese auf natürlicher Sprache basieren, wäre es interessant zu sehen, wie gut aktuelle Large Language Models für eine solche Assoziation semantischer Begriffe geeignet wären. Durch das Lernen und Übersetzen könnte eine dauerhafte Fähigkeitsverbesserung des Teams und nicht nur eine temporäre erzielt werden.

Der Ansatz behält die ursprünglichen Abfolgen der Mission bei und parallelisiert nicht selbständig. In der Zukunft sollte das parallele Mehrfach-Auktionieren basierend auf einer zusätzlichen Modellierung im *Capability*-Knoten oder aber konkrete Werkzeuge zum bewussten Einbauen der Parallelität umgesetzt werden. Durch die bewusste Parallelisierung könnte der Ansatz besser mit anderen Ansätzen verglichen und leichter für die Beschreibung von Team-Missionen eingesetzt werden. Eine solche Entwicklung ist eng mit der Entwicklung oder Anbindung eines Planers verbunden. Die Wahl der Planungsebene muss jedoch sorgfältig erfolgen, um die Flexibilität der Modellierung nicht erneut einzuschränken. Ein opportunistischer Planer mit kurzem Zeithorizont und iterativen Planungsschritten erscheint für das dynamische System dabei am vielversprechendsten.

Der Ansatz der massiven Parallelisierung mittels PoI (siehe Abschnitt 6.3.2) nutzt bereits jetzt Fähigkeiten, die basierend auf dem PoI-Typ ausgeführt werden. Es könnte daher lohnenswert sein, den PoI Ansatz mit dem hier präsentierten fähigkeitsbasierten Ansatz zu kombinieren und mit einem Hybrid die Vorteile beider Ansätze (massive Parallelisierung bei Einhalten der ursprünglichen Mission durch explizite Synchronisierung und flexible Modellierung der Aufgaben) zu verwenden. Die Berechnung von Utilites oder die Entscheidung, wann eine Fähigkeit per PoI oder Capability-Knoten vergeben werden sollte oder wie die Daten zwischen diesen Modellen synchronisiert werden, bedürfen jedoch weiterer Forschung.

Die *Mission Control* sollte durch eine „Standard-Mission“ erweitert werden. Ein wichtiges Forschungsthema dabei ist ein System für die Abwägung der Partizipa-

[1] Lehrmeister

tionsstrategie, also wann *Remote Capability Slots* angeboten werden sollten. Auch für die *Mission Control* wäre die Integration des PoI Ansatzes denkbar, etwa indem PoIs für die eigene Mission genutzt werden und gleichzeitige periodisch oder aufgrund von Kriterien die Bereitstellung des *Remote Capability Slots* für die gezielte Kooperation ausgelöst wird. Mensch-Maschine-Kooperationen sind durch das vorgestellte System ebenfalls realisierbar, da alle Modellierungsschritte menschlich interpretiert werden können und diese z.B. eine aus der Modellierung generierte Arbeitsanweisung erhalten könnten. Gleichzeitig ist die Definition von Verhalten und Missionen eine Aufgabe, in der Menschen besonders gut sind. Durch die bereitgestellten Werkzeuge können die Fähigkeiten von Robotern schneller von Menschen erfasst und sehr einfach in Missionen eingesetzt werden. Das System wird somit insbesondere das Führen kompletter Teams weiter vereinfachen.

Die Popularität von Behavior Trees ist in den letzten Jahren enorm gestiegen und vor allem in der Robotik inzwischen ein Standard-Werkzeug geworden. Der vorgeschlagene Ansatz ist daher für viele aktuelle und zukünftige Systeme geeignet und kann mit wenig Adaptionen umgesetzt werden. Die fähigkeitsbasierte Kooperation ist damit eine reale Option für den flächendeckenden Einsatz auf sehr unterschiedlichen Systemen.

Anhang

A. Appendix 1

A.1. Behavior-Trees

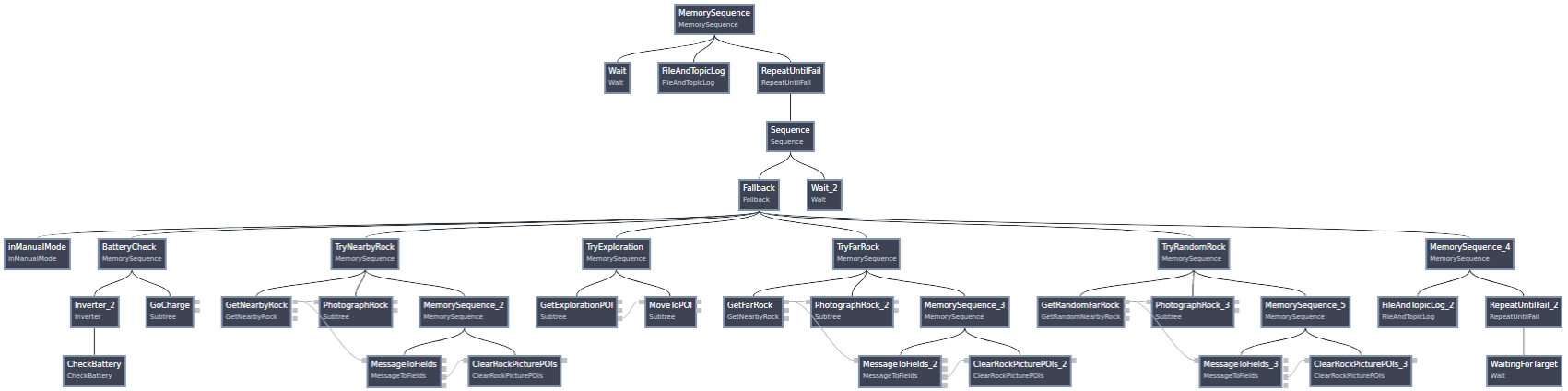


Abbildung A.1.: Kompletter Top Level Behavior Tree für das Explorationsverhalten während des SRC für den Spot Roboter. Die Sturktur der einelnen Aufgaben ist klar erkennbar. Die Umsetzung komplexerer Aufgaben wird mittels Subtrees gekapselt.

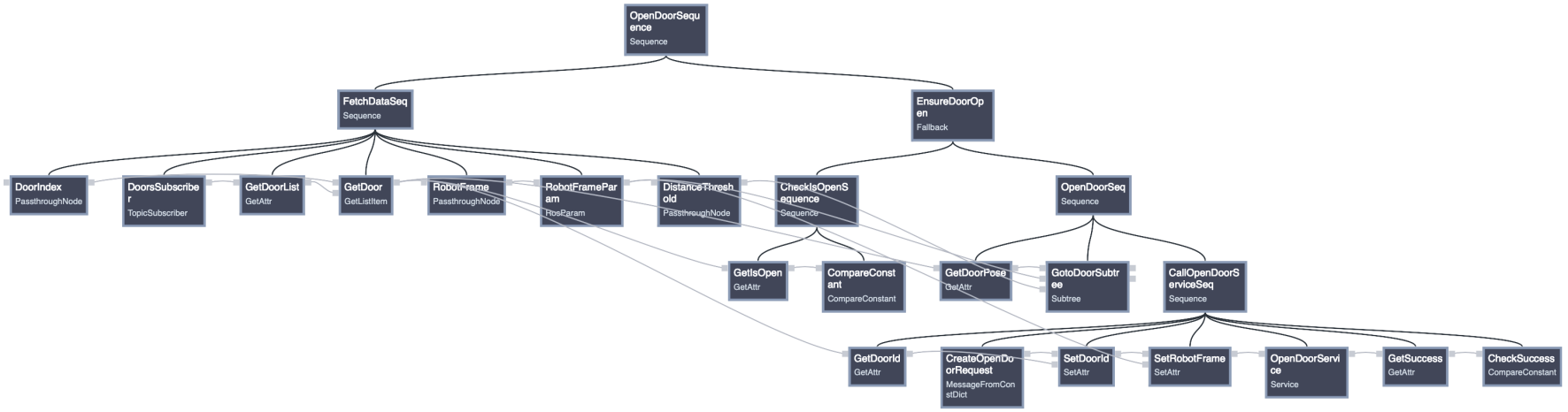


Abbildung A.2.: Kompletter *OpenDoorSubtree* der Tür-Simulation. Da der Tree prüft, ob die Tür bereits offen ist, kann der gesamte Tree reaktiv gestaltet werden. Wenn die Tür wieder geschlossen wird, fängt der Subtree sofort damit an, den Roboter wieder zur Tür zu schicken und dieser zu öffnen. Quelle: [BH22]

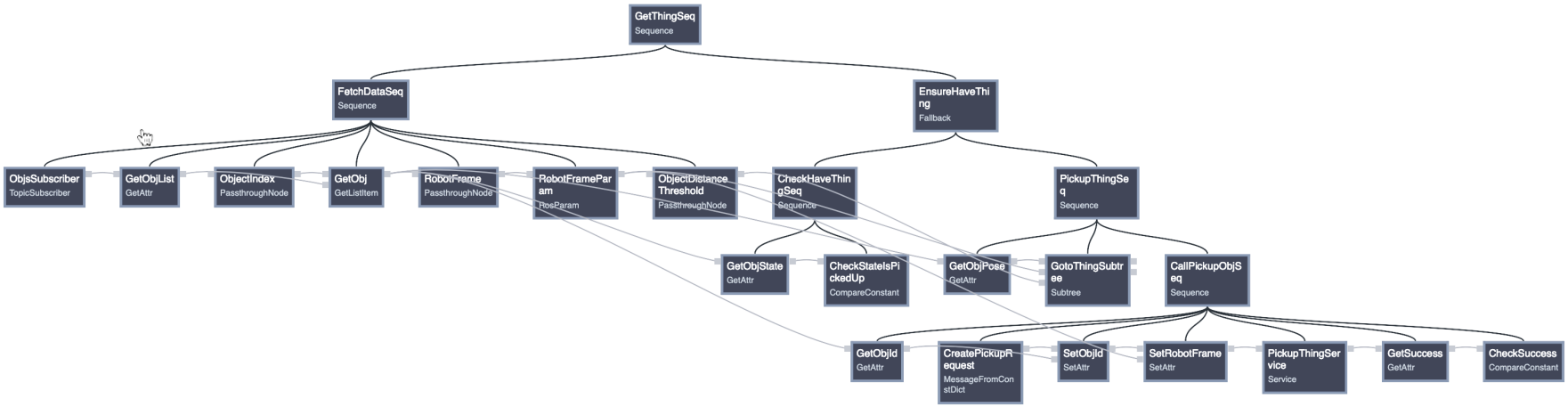


Abbildung A.3.: Kompletter *GetThingSubtree* der Tür-Simulation. Wie auch beim Öffnen der Tür wird geprüft ob das Objekt bereits gegriffen wurde. Es wird erneut gegriffen falls das nicht der Fall sein sollte. Quelle: [BH22]

Akronyme

AMR Autonomous Mobile Robot 220, *Glossary:* Autonomous Mobile Robot

API Application Programming Interfaces 181, 182, *Glossary:* Application Programming Interfaces

AR Augmented Reality 170, *Glossary:* Augmented Reality

BT Behavior Tree 16, 19, 35, 51, 53–59, 69, 70, 72, 74, 81–85, 87–101, 103–109, 112, 116, 125–129, 134, 135, 137, 148, 152, 153, 155, 157, 164–168, 172, 174, 176, 179, 191, 194, 196–198, 202–204, 206, 222, 224, 225

BTL Behavior Tree Logic 54, *Glossary:* Behavior Tree Logic

DARPA Defense Advanced Research Projects Agency 174

DDS Data Distribution Service 12, 13, *Glossary:* Data Distribution Service

DLR Deutsches Zentrum für Luft- und Raumfahrt 137, 138

DoF Degree of Freedom 43, 112, 113, *Glossary:* Degree of Freedom

ESA European Space Agency 137, 147, 148, 150, 219, 223

FCL Flight Control Language 51, *Glossary:* Flight Control Language

FIDL Flight Intention Description Language 41

FiFo First in - First Out 143

FPS Frames Per Second 124, *Glossary:* Frames Per Second

FSM Finitie State Machine 53, 159, 218, *Glossary:* Finitie State Machine

HFSM Hierarchical Finite State Machine 64, 68–70, *Glossary:* Hierarchical Finite State Machine

HTN Hirachical Task Network 38, 39, 55, 65, 203, *Glossary:* Hirachical Task Network

KaRoLa Karlsruhe Rotierender Laser Scanner 138, 139, *Glossary:* Karlsruhe Rotierender Laser Scanner

KI Künstliche Intelligenz 9, *Glossary:* Künstliche Intelligenz

LTL Linear Temporal Logic 52, 54, *Glossary:* Linear Temporal Logic

MAS Multi-Agenten-System 21, *Glossary:* Multi-Agenten-System

MDE Model Driven Engineering 71–73, 123, 124, 156, 157, 164, 222, *Glossary:* Model Driven Engineering

MRS Multi-Roboter-System 2, 7, 19–21, 25, 27, 36, 40, 48, *Glossary:* Multi-Roboter-System

MRTA Multi-Robot Task Allocation 4, 5, 21, 22, 26, 80, 193, 201, *Glossary:* Multi-Robot Task Allocation

NPC Non Player Character 53

OAP Optimal Assignment Problem 22, *Glossary:* Optimal Assignment Problem

PoI Point of Interest 46, 49, 68, 142, 143, 149, 151, 154, 178, 197, 198, 207, 208, *Glossary:* Point of Interest

RAFCON RMC advanced flow control 89

RAL Robot-Abstraction-Layer 65, 66, 76–79, *Glossary:* Robot-Abstraction-Layer

RET Robot Exploration Team - Stack 148, 155, *Glossary:* Robot Exploration Team - Stack

ROS Robot Operating System 12, 13, 57, 58, 67, 75, 77–80, 106, 134, 155, 163, 201, 202, *Glossary:* Robot Operating System

RPC Remote Procedure Call 13

SBC SpaceBot Cup/Camp 75, 78, 149, 151, *Glossary:* SpaceBot Cup/Camp

SPA Sense Plan Act 50, *Glossary:* Sense Plan Act

SRC Space Ressources Challenge 147, 149–155, 178, 197, 198, 212, 219, 223, 224, *Glossary:* Space Ressources Challenge

TCP Tool Center Point 157, *Glossary:* Tool Center Point

TST Task Specification Tree 51, *Glossary:* Task Specification Tree

UAV Unmanned Aerial Vehicle 1, 35–38, 41, 42, 44–46, 49, 50, 53, 131, 134, 171, 179, 182, 198, 220, *Glossary:* Unmanned Aerial Vehicle

UAVM Unmanned Aerial Vehicle Manipulator System 43, *Glossary:* Unmanned Aerial Vehicle Manipulator System

UGV Unmanned Ground Vehicle 45, *Glossary:* Unmanned Ground Vehicle

VR Virtual Reality 170, *Glossary:* Virtual Reality

Glossar

Application Programming Interfaces Eine API ist eine vom Hersteller eines Systems bereitgestellte Schnittstelle auf Quelltext-Ebene zur Verwendung des Systems aus anderen Programmen heraus. Die Schnittstelle besteht aus Definitionen und Dokumentationen von Aufrufen, die durch das System zur Verfügung gestellt und von anderen Programmen verwendet werden können.

Augmented Reality Darstellung eines virtuellen Modells, welches auf die reale Welt überlagert dargestellt wird und dadurch zusätzliche Informationen anzeigen kann. Normalerweise werden virtuelle Marker, Spielfiguren oder Messinstrumente durch ein System aus Kamera und Bildschirm angezeigt und mittels Keypoint-Tracking stabil an einer in der Realität verankerten Position angezeigt.

Autonomous Mobile Robot Überbegriff für autonom agierende Robotersysteme. Meistens werden mobile Plattformen, etwa in der Logistik, damit gemeint, der Begriff kann aber jeden Roboter, also z.B. auch Laufroboter oder selbstfahrende Autos, beinhalten.

Behavior Tree Logic Temporale Logik, welche als Prädikatenlogik zusätzlich temporale Beziehungen beinhaltet und als Behavior Tree formuliert wird.

Blackboard Blackboard bezeichnet eine Datenstruktur, in der mehrere Prozesse oder Teilnehmer Daten gemeinsam bearbeiten oder austauschen. Oft einfach ein unstrukturierter Speicherbereich, der durch die jeweiligen Teilnehmer interpretiert wird.

Data Distribution Service Der Data Distribution Service ist ein Standard bzw. Middleware für die datenzentrierte Kommunikation in dynamischen verteilten Systemen. Es nutzt das Publish-Subscribe-Konzept für die Kommunikation, Quality of Service Definition für die Übertragung und Discovery-Mechanismen für dass Serverlose Auffinden der Kommunikationspartner. Es gibt verschiedene Implementierungen für DDS, sowohl als Open Source als auch kommerziell, die, in der Theorie, interoperabel sind.

Degree of Freedom Deutsch : Freiheitsgrade eines Systems. Die Menge der unabhängig voneinander beeinflussbaren Größen in einem System. Eine einfache Rotationsachse besitzt einen Freiheitsgrad. Die Position (ohne Orientierung) eines Objektes im 3-dimensionalen Raum wird durch 3 Freiheitsgrade (x,y,z) angegeben.

Finitie State Machine Ein endlicher Automat (Englisch: finite state machine, FSM) ist ein Modell eines Verhaltens, definiert durch Zustände, Übergänge und Aktionen.

Flight Control Language Sprache zum Beschreiben verfügbarer Flugmanöver und deren Parametrisierung. Während verschiedene Architekturen eine FCL nutzen und der geplante Nutzen gerade die Abstraktion von Kommandos ist, ist die Umsetzung leider oft von Projekt zu Projekt verschieden.

Frames Per Second Angabe für die Geschwindigkeit, mit der bildbasierte (Frames) Videosignale angezeigt oder aufgenommen werden können. Üblicherweise wird der Wert für Kameras und Anzeigegeräte wie Monitore angegeben.

Hierarchical Finite State Machine Hierachische Verkettung von endlichen Automaten FSM

Hirachical Task Network Planungsansatz zum Zerlegen einer abstrakten Aufgabe (Task) in ausführbare Aufgabenprimitive. Durch das iterative Ersetzen einer Aufgabe durch eine Beschreibung der Teilaufgaben, aus denen die Aufgabe besteht, kann eine Abfolge von Aufgabenprimitiven erzeugt werden.

Karlsruhe Rotierender Laser Scanner Eigenentwickelter Sensor des FZI. Ein Hokuyo Laserscanner seitlich montiert und während der Messung mittels Motor 360° gedreht. Die integrierte Recheneinheit setzt die Scans zu einer hochaufgelösten 3D-Punktwolke der Umgebung zusammen.

Künstliche Intelligenz Eine Software, welche *intelligentes*, Verhalten aufweist. Der Begriff Intelligenz wird dabei vor allem als der eines rationalen Agenten ausgelegt, oft wird aber auch auch von Menschenähnlichem oder imitierenden Verhalten gesprochen [Heppner and Dillmann, 2018]. Ein Kern-Merkmal ist für gewöhnlich die Fähigkeit zu lernen, weshalb nahezu alle Machine-Learning-Ansätze, neuronale Netze oder Large Language Models als KI Bezeichnet werden.

Linear Temporal Logic Temporale Logik, welche als Prädikatenlogik zusätzlich temporale Beziehungen beinhaltet.

Model Driven Engineering Ansatz für die Entwicklung, bei der das Erstellen eines Modells im Vordergrund steht, auf dessen Basis dann alle weiteren Entwicklungsschritte durchgeführt werden können. Insbesondere wird eine abstrakte Entwicklung ermöglicht, bei der Implementierungsgendetails, wie etwa die Programmiersprache, sehr leicht gewechselt werden können.

Multi-Agenten-System Gruppe verteilter (Software-) Agenten, die unabhängig oder als Gruppe agieren. Oft wird dieser Begriff verwendet, wenn reine Softwareagenten gemeint sind, auch wenn gelegentlich von Robotern als Agent gesprochen wird.

Multi-Robot Task Allocation Forschungsgebiet, das der Frage der Allokation, also der Zuordnung und Koordination von Aufgaben an Roboter in einem Team aus Robotern, nachgeht.

Multi-Roboter-System Eine Gruppe von (heterogenen) Robotern, die ein gemeinsames Ziel verfolgen.

Optimal Assignment Problem Die Frage, wie n Aufgaben an m Arbeiter verteilt werden können, um die Performance des gesamten Teams zu maximieren.

Point of Interest Ein PoI beschreib eine Koordinate bzw. eine Pose, die für die Robotermission relevant ist und an der bestimmte Aktionen durchgeführt werden sollen. Ein PoI könnte z.b. ein Explorationspunkt sein oder aber ein Ort, an dem eine Manipulation durchgeführt werden soll. In der vorgestellten Architektur sind PoIs global verfügbare Daten, die zwischen den Robotern geteilt und auch zur Synchonisation bzw. Koordination genutzt werden können, wenn an einem PoI z.b. eine Fähigkeit hängt.

Robot Exploration Team - Stack Bezeichnung des Autonomie-Stacks des FZI, der für die ESA-SRC eingesetzt wurde.

Robot Operating System Eine populäre Middleware (trotz des Namens handelt es sich nicht um ein Betriebssystem) für die Entwicklung von Robotersoftware. ROS bietet Methoden und Werkzeuge, um Software in Pakete zu kapseln und handhabt Ausführung, Parametrisierung und Kommunikation zwischen verschiedenen Bausteinen.

Robot-Abstraction-Layer Abstraktionsschicht zur Ansteuerung der roboterspzifischen Hardware, etwa das Umwandeln von metrischen Geschwindigkeitskommandos in roboterspezifischen Einheiten wie mm/s und Ansteuerung der relevanten Robotermodule.

Sense Plan Act Traditionelles Paradigma der Architekturen für mobile Roboter. In Diskreten Schritten wird die Umwelt erfasst (sense) und in ein Modell überführt, eine Aktion basierend auf dem Modell geplant (plan) und dann ausgeführt (act). Das Paradigma begünstigt sehr die auf Planung fokussierten Ansätze und wird immer mehr durch andere Ansätze abgelöst.

Space Ressources Challenge Wettbewerb der ESA, bei dem möglichst autonome Systeme eine mondähnliche Umgebung explorieren und Ressourcen im Boden und Gesteinsproben finden müssen.

SpaceBot Cup/Camp Ein vom DLR ins Leben gerufener Wettbewerb und Leistungsschau, um die aktuellen Fähigkeiten der Robotik für die autonome Exploration im Weltraum einzuschätzen. Bei dem Wettbewerb müssen Roboter möglichst autonom Proben in einer unbekannten Umgebung finden und zur Basis transportieren.

Task Specification Tree Verteilte Datenstruktur für die Beschreibung komplexer Multiagentensystem. Aufgabenbäume zur strukturierten Beschreibung von Tasks. Zu jedem Tasks können Parameter, Vorbedingungen, Randbedingungen und Ausführungsumgebungen angegeben werden [55].

Tool Center Point Der Tool Center Point ist der Werkzeugmittelpunkt, also der Punkt, mit dem der Roboter ein Werkstück mit seinem jeweiligen Werkzeug berührt. Roboterbewegungen werden oft mit Bezug auf den TCP definiert, um eine korrekte Bewegung sicherzustellen oder im TCP-Koordinatensystem definiert, um Relativbewegungen auszuführen. Oft wird TPC gleichbedeutend für Endeffektor, also das Werkzeug an sich, verwendet.

Unmanned Aerial Vehicle Ein unbemanntes Fluggerät, auch UAS (Unmanned Aerial System) genannt, ist ein fliegendes Robotersystem mit variablem Grad an Autonomie. Die Bezeichnung wird sowohl für Helikopter, Multikopter, als auch Tragflächenflugzeuge verwendet. Ein Unterschied zu ferngesteuerten Fluggeräten ist, dass die Systeme immer über einen minimalen Grad an Autonomie verfügen

Unmanned Aerial Vehicle Manipulator System Ein UAV, ausgestattet mit einem zusätzlichem Roboterarm für die Manipulation von Objekten.

Unmanned Ground Vehicle Ein autonom auf dem Boden agierendes Robotersystem. Üblicherweise sind dies radgestützte Systeme oder auch Logistikroboter. Siehe auch AMR.

Virtual Reality Komplett virtuelle Darstellung eines Modells in einer virtuellen Umgebung, bei der der Nutzer durch technische Maßnahmen wie VR-Headsets in diese Umgebung eintauchen kann. VR findet bisher primär Anwendung in Spielen, kann jedoch auch für die virtuelle Vorbereitung, etwa auf bestimmte Arbeitsschritte, genutzt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, und K. Fujimura, "The intelligent asimo: System overview and integration," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, pp. 2478–2483, IEEE, 2002.

[2] Y. U. Cao, A. S. Fukunaga, und A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous robots*, vol. 4, no. 1, pp. 7–27, 1997.

[3] T. Arai, E. Pagello, und L. E. Parker, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[4] A. A. C. Z. Yan, N. Jouandeau, "A survey and analysis of multi-robot coordination," *Journal of Advanced Robotic Systems*, vol. 10, pp. 1–18, 2013.

[5] B. P. Gerkey und M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[6] G. A. Korsah, A. Stentz, und M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[7] I. Maza, J. Muñoz-Morera, F. Caballero, E. Casado, V. Perez-Villar, und A. Ollero, "Architecture and tools for the generation of flight intent from mission intent for a fleet of unmanned aerial systems," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 9–19, May 2014.

[8] Google LLC, "Google blogpost:"the bright side of sitting in traffic: Crowdsourcing road congestion data"." `https://googleblog.blogspot.com/2009/08/bright-side-of-sitting-in-traffic.html`. Abgerufen am: 15.02.2024.

[9] P. J. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, und K. Rothermel, "Tinycubus: a flexible and adaptive framework sensor networks," in *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pp. 278–289, Jan 2005.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, und A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[11] T. Foote, "Ros community metrics report 2018." http://download.ros.org/downloads/metrics/metrics-report-2018-07.pdf, 2018. Abgerufen am: 15.02.2024.

[12] T. F. Katherine Scott, "Ros community metrics report 2022." http://download.ros.org/downloads/metrics/metrics-report-2022-07.pdf, 2022. Abgerufen am: 15.02.2024.

[13] M. B. Dias, R. Zlot, N. Kalra, und A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[14] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. 29, pp. 1104–1113, Dec. 1980.

[15] T. Sandholm, "Algorithm for optimal winner determination in combinatorial auctions," *Artificial intelligence*, vol. 135, no. 1-2, pp. 1–54, 2002.

[16] R. Zlot und A. Stentz, "Complex task allocation for multiple robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1515–1522, April 2005.

[17] G. Dudek, M. R. M. Jenkin, E. Milios, und D. Wilkes, "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, pp. 375–397, Dec 1996.

[18] D. Gale, *The theory of linear economic models*. University of Chicago press, 1960.

[19] R. M. Zlot, *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, December 2006.

[20] L. E. Parker, D. Rus, und G. Sukhatme, "Multiple mobile robot systems," in *Springer Handbook of Robotics* (B. Siciliano und O. Khatib, eds.), ch. 53, Springer, 2016.

[21] Y. Rizk, M. Awad, und E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[22] L. Antonyshyn, J. Silveira, S. Givigi, und J. Marshall, "Multiple mobile robot task and motion planning: A survey," *ACM Computing Surveys*, vol. 55, no. 10, pp. 1–35, 2023.

[23] L. E. Parker, "Alliance: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots," in *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, IROS*, vol. 2, pp. 776–783 vol.2, Sep 1994.

[24] L. E. Parker, *Heterogeneous Multi-Robot Cooperation*. PhD thesis, Massachusetts Institute of Technology, 1994.

[25] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[26] B. B. Werger und M. J. Matarić, *Broadcast of Local Eligibility for Multi-Target Observation*, pp. 347–356. Tokyo: Springer Japan, 2000.

[27] S. C. Botelho und R. Alami, "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement," in *Proceedings 1999 IEEE International Conference on Robotics and Automation, ICRA*, vol. 2, pp. 1234–1239 vol.2, 1999.

[28] B. P. Gerkey und M. J. Mataric, "Sold!: auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 758–768, Oct 2002.

[29] F. Tang und L. E. Parker, "Asymtre: Automated synthesis of multi-robot task solutions through software reconfiguration," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA*, pp. 1501–1508, April 2005.

[30] M. B. Dias, *Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments*. Carnegie Mellon University, 2004.

[31] R. Zlot und A. Stentz, "Market-based multirobot coordination for complex tasks," *The International Journal of Robotics Research*, vol. 25, no. 1, pp. 73–101, 2006.

[32] A. Ollero, S. Lacroix, L. Merino, J. Gancet, J. Wiklund, V. Remuss, I. V. Perez, L. G. Gutierrez, D. X. Viegas, M. A. G. Benitez, A. Mallet, R. Alami, R. Chatila, G. Hommel, F. J. C. Lechuga, B. C. Arrue, J. Ferruz, J. R. M.-D. Dios, und F. Caballero, "Multiple eyes in the skies: architecture and perception issues in the comets unmanned air vehicles project," *IEEE Robotics Automation Magazine*, vol. 12, pp. 46–57, June 2005.

[33] A. Ollero, P. J. Marron, M. Bernard, J. Lepley, M. la Civita, E. de Andres, und L. van Hoesel, "Aware: Platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with unmanned aerial vehicles," in *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, pp. 1–6, Sept 2007.

[34] K. P. O'Rourke, T. G. Bailey, R. Hill, und W. B. Carlton, "Dynamic routing of unmanned aerial vehicles using reactive tabu search," in *67th MORS Symposium*, November 1999.

[35] J. Gancet, G. Hattenberger, R. Alami, und S. Lacroix, "Task planning and control for a multi-uav system: architecture and algorithms," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1017–1022, Aug 2005.

[36] T. Lemaire, R. Alami, und S. Lacroix, "A distributed tasks allocation scheme in multi-uav context," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 4, pp. 3622–3627 Vol.4, April 2004.

[37] I. Maza, F. Caballero, J. Capitan, J. R. Martinez-de Dios, und A. Ollero, "A distributed architecture for a robotic platform with aerial sensor transportation and self-deployment capabilities," *Journal of Field Robotics*, vol. 28, no. 3, pp. 303–328, 2011.

[38] D. Nau, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, und F. Yaman, "Shop2: An htn planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[39] A. Viguria, I. Maza, und A. Ollero, "Set: An algorithm for distributed multirobot task allocation with dynamic negotiation based on task subsets," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3339–3344, April 2007.

[40] T. Bedrax-Weiss, C. McGann, und M. Iatauro, "Europa 2: plan database services for planning and scheduling applications," in *International Conference on Automated Planning and Scheduling*, pp. 18–19, 2005.

[41] T. J. Parr und R. W. Quong, "Antlr: A predicated-ll(k) parser generator," *Software: Practice and Experience*, vol. 25, no. 7, pp. 789–810, 1995.

[42] G. Antonelli, K. Baizid, F. Caccavale, G. Giglio, und F. Pierri, "Cavis: a control software architecture for cooperative multi-unmanned aerial vehicle-manipulator systems," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1108–1113, 2014.

[43] A. Gianluca, B. Khelifa, C. Fabrizio, G. Gerardo, M. Giuseppe, und P. Francesco, "Control software architectures for cooperative multi unmanned aerial vehicles manipulator systems," *JOURNAL OF SOFTWARE ENGINEERING IN ROBOTICS*, vol. 5, no. 2, pp. 1–12, 2014.

[44] G. Antonelli, F. Arrichiello, und S. Chiaverini, "The nsb control: a behavior-based approach for multi-robot systems," *Paladyn*, vol. 1, pp. 48–56, Mar 2010.

[45] M. A. Hsieh, A. Cowley, J. F. Keller, L. Chaimowicz, B. Grocholsky, V. Kumar, C. J. Taylor, Y. Endo, R. C. Arkin, B. Jung, D. F. Wolf, G. S. Sukhatme, und D. C. MacKenzie, "Adaptive teams of autonomous aerial and ground robots for situational awareness," *Journal of Field Robotics*, vol. 24, no. 11-12, pp. 991–1014, 2007.

[46] B. Grocholsky, J. Keller, V. Kumar, und G. Pappas, "Cooperative air and ground surveillance," *IEEE Robotics Automation Magazine*, vol. 13, pp. 16–25, Sept 2006.

[47] P. Kondaxakis, K. Gulzar, und V. Kyrki, "Temporal arm tracking and probabilistic pointed object selection for robot to robot interaction using deictic gestures," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 186–193, Nov 2016.

[48] A. Tsiamis, C. K. Verginis, C. P. Bechlioulis, und K. J. Kyriakopoulos, "Cooperative manipulation exploiting only implicit communication," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 864–869, Sept 2015.

[49] A. Marzinotto, M. Colledanchise, C. Smith, und P. Ögren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5420–5427, May 2014.

[50] M. Guo und D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[51] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, und P. Oegren, "The advantages of using behavior trees in mult-robot systems," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pp. 1–8, June 2016.

[52] L. Marconi, C. Melchiorri, M. Beetz, D. Pangercic, R. Siegwart, S. Leutenegger, R. Carloni, S. Stramigioli, H. Bruyninckx, P. Doherty, A. Kleiner, V. Lippiello, A. Finzi, B. Siciliano, A. Sala, und N. Tomatis, "The sherpa project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments," in *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–4, Nov 2012.

[53] P. Doherty, J. Kvarnström, M. Wzorek, P. Rudol, F. Heintz, und G. Conte, "Hdrc3 - a distributed hybrid deliberative/reactive architecture for unmanned aircraft systems," in *Handbook of Unmanned Aerial Vehicles :*, pp. 849–952, 2014.

[54] E. Gat, R. P. Bonnasso, R. Murphy, *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[55] P. Doherty, F. Heintz, und D. Landén, "A distributed task specification language for mixed-initiative delegation," in *International conference on principles and practice of multi-agent systems*, pp. 42–57, Springer, 2010.

[56] S. Govindaraj, J. Gancet, D. Urbina, W. Brinkmann, N. Aouf, S. Lacroix, M. Wolski, F. Colmenero, M. Walshe, C. Ortega, *et al.*, "Pro-act: Planetary robots deployed for assembly and construction of future lunar isru and supporting infrastructures," in *proceedings of 15th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2019). ESA/Estec Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA-2019)*, 2019.

[57] M. Lippi, P. Di Lillo, und A. Marino, "A task allocation framework for human multi-robot collaborative settings," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7614–7620, IEEE, 2023.

[58] D. Isla, "Handling complexity in the halo 2 ai," in *Game Developers Conference (GDC)*, 2005.

[59] D. Isla, "Halo 3-building a better battle," in *Game Developers Conference (GDC)*, 2008.

[60] C.-U. Lim, R. Baumgarten, und S. Colton, "Evolving behaviour trees for the commercial game defcon," in *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, pp. 100–110, Springer, 2010.

[61] R. A. Agis, S. Gottifredi, und A. J. García, "An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games," *Expert Systems with Applications*, vol. 155, p. 113457, 2020.

[62] Epic Games, Inc, "Unreal engine behavior tree documentation." `https://docs.unrealengine.com/5.3/en-US/behavior-trees-in-unreal-engine/`. Abgerufen am: 15.02.2024.

[63] PADAONE GAMES S.L., "Behavior bricks documentation." `http://bb.padaonegames.com/doku.php`. Abgerufen am: 15.02.2024.

[64] M. Colledanchise und P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Chapman and Hall/CRC Artificial Intelligence and Robotics Series, Taylor & Francis Group, 2018.

[65] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, und C. Smith, "On the programming effort required to generate behavior trees and finite state machines for robotic applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5807–5813, IEEE, 2023.

[66] P. Ogren, "Increasing modularity of uav control systems using computer game behavior trees," in *Aiaa guidance, navigation, and control conference*, p. 4458, 2012.

[67] A. Klöckner, "Behavior trees for uav mission management," *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*, vol. P-220, pp. 57–68, 09 2013.

[68] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, und N. I. Badler, "Parameterizing behavior trees," in *Motion in Games: 4th International Conference, MIG 2011, Edinburgh, UK, November 13-15, 2011. Proceedings 4*, pp. 144–155, Springer, 2011.

[69] M. Colledanchise und L. Natale, "Handling concurrency in behavior trees," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2557–2576, 2021.

[70] M. Colledanchise, R. M. Murray, und P. Ögren, "Synthesis of correct-by-construction behavior trees," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 6039–6046, IEEE, 2017.

[71] C. Martens, E. Butler, und J. C. Osborn, "A resourceful reframing of behavior trees," *arXiv preprint arXiv:1803.09099*, 2018.

[72] M. Colledanchise, D. Almeida, und P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8839–8845, IEEE, 2019.

[73] P. Ögren und J. Alfredson, "Creating trustworthy ai for uas using labeled backchained behavior trees," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1029–1036, IEEE, 2023.

[74] M. Mayr, F. Rovida, und V. Krueger, "Skiros2: A skill-based robot control platform for ros," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6273–6280, IEEE, 2023.

[75] F. Rovida, B. Grossmann, und V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6793–6800, IEEE, 2017.

[76] Y. Cao und C. G. Lee, "Behavior-tree embeddings for robot task-level knowledge," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12074–12080, IEEE, 2022.

[77] K. R. Guerin, C. Lea, C. Paxton, und G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 6167–6174, IEEE, 2015.

[78] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, und G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 564–571, IEEE, 2017.

[79] Rethink Robotics GmbH, "Rethink robotics intera plattform." `https://www.rethinkrobotics.com/intera`. Abgerufen am: 15.02.2024.

[80] Intrinsic Innovation GmbH, "Introducing intrinsic flowstate." `https://intrinsic.ai/blog/posts/introducing-intrinsic-flowstate/`. Abgerufen am: 15.02.2024.

[81] D. Hu, Y. Gong, B. Hannaford, und E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3868–3875, IEEE, 2015.

[82] Y. Wu, J. Li, H. Dai, X. Yi, Y. Wang, und X. Yang, "micros. bt: An event-driven behavior tree framework for swarm robots," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9146–9153, IEEE, 2021.

[83] A. Neupane und M. A. Goodrich, "Learning swarm behaviors using grammatical evolution and behavior trees.," in *IJCAI*, pp. 513–520, 2019.

[84] M. Iovino, E. Scukins, J. Styrud, P. Ögren, und C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[85] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, und A. Wąsowski, "Behavior trees in action: a study of robotics applications," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, pp. 196–209, 2020.

[86] M. Colledanchise und L. Natale, "On the implementation of behavior trees in robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.

[87] M. Colledanchise und P. Ögren, "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees," *IEEE Transactions on Robotics*, vol. 33, pp. 372–389, April 2017.

[88] S. Jeong, T. Ga, I. Jeong, und J. Choi, "Behavior tree-based task planning for multiple mobile robots using a data distribution service," in *2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1791–1798, IEEE, 2022.

[89] S. S. O. Venkata, R. Parasuraman, und R. Pidaparti, "Kt-bt: A framework for knowledge transfer through behavior trees in multirobot systems," *IEEE Transactions on Robotics*, 2023.

[90] T. G. Tadewos, L. Shamgah, und A. Karimoddini, "On-the-fly decentralized tasking of autonomous vehicles," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 2770–2775, IEEE, 2019.

[91] A. Sidorenko, J. Hermann, und M. Ruskowski, "Using behavior trees for coordination of skills in modular reconfigurable cppms," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2022.

[92] N. P. Koenig und A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[93] R. P. Stanley, *Enumerative Combinatorics (Cambridge Studies in Advanced Mathematics, 49)*. Cambridge University Press, 2011.

[94] K. R. Guerin, C. Lea, C. Paxton, und G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *International Conference on Robotics and Automation (ICRA)*, pp. 6167–6174, May 2015.

[95] Y. Shoham und K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, UK: Cambridge University Press, 2009.

[96] M. Otte, M. J. Kuhlman, und D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," vol. 44, no. 3, pp. 547–584. 26 citations (Crossref) [2022-05-19].

[97] Y. Zhang und M. Q.-H. Meng, "Comparison of auction-based methods for task allocation problem in multi-robot systems," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2609–2613. 2 citations (Crossref) [2022-05-20].

[98] T. Kaupisch und D. Noelke, "Dlr spacebot cup 2013: A space robotics competition," *KI - Künstliche Intelligenz*, vol. 28, pp. 111–116, March 2014.

[99] T. Schnell, D. Oberacker, F. Exner, L. Puck, M. G. Besselmann, N. Spielbauer, C. Plasberg, A. Roennau, und R. Dillmann, "An efficient scalable autonomy approach for teams of heterogeneous mobile robots," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pp. 1–7, 2023.

[100] L. Pfotzer, J. Oberländer, A. Roennau, und R. Dillmann, "Development and calibration of KaRoLa, a compact, high-resolution 3d laser scanner," in *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pp. 1–6, Oct 2014.

[101] M. G. Besselmann, L. Puck, L. Steffen, A. Roennau, und R. Dillmann, "Vdb-mapping: A high resolution and real-time capable 3d mapping framework for versatile mobile robots," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 448–454, 2021.

[102] A. Zoitl und R. Lewis, *Modelling control systems using IEC 61499. 2nd Edition*. 05 2014.

[103] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, und A. Martel, "Framework for distributed industrial automation and control (4diac)," in *2008 6th IEEE international conference on industrial informatics*, pp. 283–288, IEEE, 2008.

[104] M. Wenger, W. Eisenmenger, G. Neugschwandtner, B. Schneider, und A. Zoitl, "A model based engineering tool for ros component compositioning, configuration and generation of deployment information," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2016.

[105] K. Regenstein, T. Kerscher, C. Birkenhofer, T. Asfour, M. Zollner, und R. Dillmann, "Universal controller module (ucom) - component of a modular concept in robotic systems," in *2007 IEEE International Symposium on Industrial Electronics*, pp. 2089–2094, 2007.

[106] A. Koval, C. Kanellakis, E. Vidmark, J. Haluska, und G. Nikolakopoulos, "A subterranean virtual cave world for gazebo based on the DARPA subt challenge," *CoRR*, vol. abs/2004.08452, 2020.

[107] J. Wang, G. Jia, H. Xin, und Z. Hon, "Research on dynamic task allocation method of heterogeneous multi-uav based on consensus based bundle algorithm," in *2020 Chinese Automation Congress (CAC)*, pp. 2214–2219, 2020.

[108] M. Nanjanath und M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 900–909, 2010.

Eigene Veröffentlichungen mit Bezug zur Dissertation

Dieses Verzeichnis listet alle Publikationen mit Bezug zur Dissertation, bei denen der Autor dieser Dissertation entweder der Erstautor ist, oder als Co-Autor maßgeblich zu der Veröffentlichung beigetragen hat (in Form von Problemstellung, -lösung, Diskussion oder experimenteller Evaluation).

[Awad et al., 2016] Awad, R., Heppner, G., Roennau, A., und Bordignon, M. (2016). Ros engineering workbench based on semantically enriched app models for improved reusability. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–9. IEEE.

[Bastinos et al., 2014] Bastinos, A. S., Haase, P., Heppner, G., Zander, S., und Ahmed, N. (2014). Reapp store-a semantic appstore for applications in the robotics domain. In *International Semantic Web Conference (Industry Track)*.

[Buettner et al., 2018] Buettner, T., Roennau, A., Heppner, G., und Dillmann, R. (2018). Design of an exchangeable, compact and modular bio-inspired leg for six-legged walking robots. In *Human-Centric Robotics: Proceedings of CLAWAR 2017: 20th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, pages 89–96. World Scientific.

[Goeller et al., 2011] Goeller, M., Roennau, A., Gorbunov, A., Heppner, G., und Dillmann, R. (2011). Pushing around a robot: Force-based manual control of the six-legged walking robot lauron. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2647–2652. IEEE.

[Heppner et al., 2023] Heppner, G., Berg, N., Oberacker, D., Spielbauer, N., Roennau, A., und Dillmann, R. (2023). Distributed behavior trees for heterogeneous robot teams. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–8. IEEE.

[Heppner et al., 2014] Heppner, G., Buettner, T., Roennau, A., und Dillmann, R. (2014). Versatile-high power gripper for a six legged walking robot. In *Mobile Service Robotics Procedings of the 17th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*, pages 461–468.

[Heppner and Dillmann, 2018] Heppner, G. und Dillmann, R. (2018). Autonomy of mobile robots. In *Dehumanization of Warfare: Legal Implications of New Weapon Technologies*, pages 77–98. Springer.

[Heppner et al., 2020] Heppner, G., Mauch, F., Scherzinger, S., Timmermann, D., Becker, P., Ulbrich, S., Rönnau, A., Heiligensetzer, P., und Fürst, F. (2020). FLA²IR — FLexible Automotive Assembly with Industrial Co-workers. *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-users: The Experience of the European Robotics Challenges*, pages 97–126.

[Heppner et al., 2024] Heppner, G., Oberacker, D., Roennau, A., und Dillmann, R. (2024). Behavior tree capabilities for dynamic multi-robot task allocation with heterogeneous robot teams. In *Accepted for Publication at the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, page forthcoming. IEEE.

[Heppner et al., 2019] Heppner, G., Plasberg, C., Puck, L., Schnell, T., Büttner, T., Roennau, A., und Dillmann, R. (2019). Risk aware robots - health estimation and capability selection. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1193–1199. IEEE.

[Heppner et al., 2013] Heppner, G., Roennau, A., et al. (2013). Enhancing sensor capabilities of walking robots through cooperative exploration with aerial robots. *Journal of Automation Mobile Robotics and Intelligent Systems*, 7.

[Heppner et al., 2017] Heppner, G., Roennau, A., Mauch, F., und Dillmann, R. (2017). Exploration and sample-return missions with a walking robot. In *14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*.

[Heppner et al., 2015] Heppner, G., Roennau, A., Oberländer, J., Klemm, S., und Dillmann, R. (2015). Laurope - six legged walking robot for panetary exploration participating in the spacebot cup. In *13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*.

[Oberländer et al., 2014] Oberländer, J., Klemm, S., Heppner, G., Roennau, A., und Dillmann, R. (2014). A multi-resolution 3-d environment model for autonomous planetary exploration. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 229–235. IEEE.

[Puck et al., 2020a] Puck, L., Keller, P., Schnell, T., Plasberg, C., Tanev, A., Heppner, G., Roennau, A., und Dillmann, R. (2020a). Distributed and synchronized setup towards real-time robotic control using ros2 on linux. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1287–1293. IEEE.

[Puck et al., 2020b] Puck, L., Schnell, T., Plasberg, C., Büttner, T., Heppner, G., Roennau, A., und Dillmann, R. (2020b). Modular, risk-aware mapping and fusion of environmental hazards. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–6.

[Roennau et al., 2011] Roennau, A., Heppner, G., und Dillmann, R. (2011). Bio-inspired heterogeneous step sizes for a six-legged walking robot. In *International Symposium on Adaptive Motion in Animals and Machines (AMAM), Hyogo, Japan*.

[Roennau et al., 2014a] Roennau, A., Heppner, G., und Dillmann, R. (2014a). Online adaptive leg trajectories for multi-legged walking robots. In *Mobile Service Robotics*, pages 369–376.

[Roennau et al., 2014b] Roennau, A., Heppner, G., Nowicki, M., Zöllner, J. M., und Dillmann, R. (2014b). Reactive posture behaviors for stable legged locomotion over steep inclines and large obstacles. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4888–4894. IEEE.

[Roennau et al., 2013] Roennau, A., Heppner, G., Pfotzer, L., und Dillmann, R. (2013). Lauron v: Optimized leg configuration for the design of a bio-inspired walking robot. In *Nature-Inspired Mobile Robotics - Procedings of the 16th International Conference on Climbing and Walking Robots (CLAWAR 2013)*, pages 563–570.

[Rönnau et al., 2010] Rönnau, A., Heppner, G., Kerscher, T., und Dillmann, R. (2010). A behaviour-based free gait inspired by a walking stick insect. In *Emerging Trends In Mobile Robotics - Proceedings of the 13th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*, pages 181–188.

[Rönnau et al., 2011] Rönnau, A., Heppner, G., Kerscher, T., und Dillmann, R. (2011). Fault diagnosis and system status monitoring for a six-legged walking robot. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 874–879. IEEE.

[Rönnau et al., 2014] Rönnau, A., Heppner, G., Nowicki, M., und Dillmann, R. (2014). Lauron v: a versatile six-legged walking robot with advanced maneuverability. In *Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on*, pages 82–87. IEEE.

[Rönnau et al., 2012] Rönnau, A., Heppner, G., Pfotzer, L., und Dillmann, R. (2012). Foot design evaluation for a six-legged walking robot. In *Adaptive Mobile Robotics - Proceedings of the 15th International Conference on Climbing and Walking Robots and the Support Technologies (CLAWAR)*, pages 511–518.

[Rönnau et al., 2013] Rönnau, A., Sutter, F., Heppner, G., Oberländer, J., und Dillmann, R. (2013). Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots. In *Advanced Robotics (ICAR), 2013 16th International Conference on*, pages 1–7. IEEE.

[Rönnau et al., 2015] Rönnau, A., Heppner, G., Klemm, S., und Dillmann, R. (2015). Roads—robot and dynamics simulation for biologically-inspired multilegged walking robots. In *Robotics and Biomimetics (ROBIO), 2015 IEEE International Conference on*, pages 1870–1876. IEEE.

[Schnell et al., 2020] Schnell, T., Plasberg, C., Puck, L., Buettner, T., Eichmann, C., Heppner, G., Roennau, A., und Dillmann, R. (2020). Robot health estimation through unsupervised anomaly detection using gaussian mixture models. In

2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), pages 1037–1042.

[Zander et al., 2015] Zander, S., Heppner, G., Neugschwandtner, G., Awad, R., Essinger, M., und Ahmed, N. (2015). A model-driven engineering approach for ros using ontological semantics. In *Proceedings of the 6th International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob-15) co-located with the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015*.

Studentische und Andere Arbeiten mit Bezug zur Dissertation

Dieses Verzeichnis listet studentische Arbeiten mit Bezug zur Dissertation, die durch den Autor dieser Dissertation im Rahmen seiner Forschung ausgeschrieben und betreut wurden. Dies beinhaltet die maßgebliche Vorgabe der Problemstellung, Diskussion der Arbeit sowie Randvorgaben zur Lösung, Visualisierung und experimentelle Evaluation. Dieses Verzeichnis listet weiterhin andere Arbeiten mit Bezug zur Dissertation, die durch den Autor dieser Dissertation im Rahmen seiner Forschung erstellt und durchgeführt wurden. Dies beinhaltet die maßgebliche Konzeption und Erarbeitung der relevanten Ergebnisse.

[BH22] Nils Berg und Georg Heppner. Distributed Execution of Behavior Trees using Heterogeneous Robot Teams. Masterarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2022.

[FZI22] FZI Forschungszentrum Informatik. Youtube Video: Autonomous Team of Robots Explores the Moon at ESA-ESRIC Space Resources Challenge. `https://youtu.be/bcAapWdIFMY?si=dZ_mlActjP7iGh8I`, November 2022. Abgerufen am 12.02.2024.

[H+13] Georg Heppner et al. Forschungsprojekt DLR SpaceBot-Cup - LAUfROboter für die Platare Exploration - LAUROPE. Gefördert durch das Bundesministerium für Wirtschaft und Technologie (BMWi), 2013.

[H+15] Georg Heppner et al. Forschungsprojekt SpaceBot-Cup 2015 - LAUfROboter für die Platare Exploration - LAUROPE 2015. Gefördert durch das Bundesministerium für Wirtschaft und Energie (BMWi), 2014-2015.

[H+16] Georg Heppner et al. Forschungsprojekt ReApp. Gefördert durch das Bundesministerium für Wirtschaft und Energie (BMWi) im Rahmen des Technologieprogramms AUTONOMIK für Industrie 4.0., 2014-2016.

[HH18] Kai-Uwe Hermann und Georg Heppner. On-Line task allocation in an ad-hoc network of complex robots. Masterarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2018.

[HI+15] Georg Heppner, FZI Forschungszentrum Informatik, et al. Youtube Video: Walking Robot LAURON V: Grasping and other skills developed for DLR SpaceBot Cup. `https://youtu.be/m8MXGFzjJxs?si=cwQEZR25uvUVtuQi`, December 2015. Abgerufen am: 15.02.2024.

[HI+17] Georg Heppner, FZI Forschungszentrum Informatik, et al. Youtube Video: ROS-Industrial Applications | Reusable Robotic Apps – ReApp@AUTOMATICA 2016. https://youtu.be/w898_9vWz3E?si=xDSM5vNabz6z7lxN, August 2017. Abgerufen am: 15.02.2024.

[HI+18] Georg Heppner, FZI Forschungszentrum Informatik, et al. Youtube Video: Walking Robot LAURON V - Planetary Sample Return Mission at DLR SpaceBot Camp 2015. https://youtu.be/M90_2M9ghZs?si=4q6zFy0e4GFG1Mj5, 2018. Abgerufen am: 15.02.2024.

[OH22] David Oberacker und Georg Heppner. Dynamic Auction-Based Allocation of Tasks for Heterogenous Robot Teams. Masterarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2022.

[PH18] Carsten Plasberg und Georg Heppner. Erstellen einer Risiko-Taxonomie und -Metrik sowie Methoden zur Risikofusion für mobile Roboter im Kontext planetarer Exploration. Masterarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2018.

[WH15] Christopher Wecht und Georg Heppner. Kostenbasierte Pfadplanung für Laufroboter in komplexen Umgebungen. Bachelorarbeit, KIT Karlsruher Institut für Technologie, Karlsruhe, Germany, 2015.