TECHNISCHE UNIVERSITÄT
ILMENAU

Department of Computer Science and Automation
Systems and Software Engineering Group

Bachelorarbeit

Konzeptionierung und Implementierung einer intelligenten Ladestation in einer diskreten eventbasierten Simulationsumgebung

Registration Date: XX. January 20XX
Submission Date: XX. June 20XX

Supervisor: Prof. Responsible

Submitted by: Ludwig Breitsprecher
Matriculation Number 54131
ludwig.breitsprecher@tu-ilmenau.de

Acknowledgments

I would like to thank ...

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Energie ist eine allgegenwärtige physikalische Größe. Elektrische Energie steht in den Industriestaaten durch einfachen Steckdosenzugriff in großer Menge zur Verfügung. Der sparsame Umgang mit diesem Gut schont die finanziellen Ressourcen und reduziert die negativen Auswirkungen der Stromerzeugung auf unsere Umwelt.

Die unbemannte Luftfahrt ist unter anderem für ihren moralisch bedenklichen Einsatz für militärische Zwecke bekannt. Davon abgesehen werden Drohnen auch in zivilen Bereichen immer verbreiteter. Große Logistikunternehmen wie DHL oder Amazon nutzen Prototypen für die unbemannte Auslieferung von Paketen. Die Suche nach Opfern eines Lawinenunglücks ist eine äußerst gefährliche Aufgabe. Weitere Lawinen können ausgelöst werden oder andere Gefahren werden von der Schneedecke verborgen.

Während die Nutzungsmöglichkeiten von unbemannten Luftfahrzeugen grundverschieden sind, wird Energie benötigt. Die Speicherung größerer Mengen von Energie ist ein anhaltendes Problem, neue Akkumulatoren verbessern die Situation, doch das Grundproblem bleibt bestehen. Energie ist knapp und sie zu transportieren zieht Aufwand, oft in Form von Gewicht, nach sich. Gerade für mobile Geräte ist das Gewicht ein ausschlaggebender Punkt. Für jede, nicht in kürzester Zeit lösbare, Aufgabe muss die Stromversorgung der flugfähigen Helfer gesichert werden.

1.2. Problemstellung

Ladestationen sind ein wichtiger Bestandteil von Elektromobilität jeglicher Art.

1.3. Zielsetzung

2. Grundlagen

Im folgenden Kapitel wird auf die nötigen Grundlagen und Hintergründe eingegangen. Die Ladestation wird in ein bestehendes Projekt eingepflegt. Dieses bestehende Projekt beschäftigt sich mit der Simulation von Missionen für unbemannte Luftfahrzeuge. Einleitend wird das angewendete Verständnis eben jener Luftfahrzeuge erläutert. Im Anschluss wird die Projektumgebung grob skizziert.

2.1. Unbemannte Luftfahrzeuge

Ein Unbemanntes Luftfahrzeug (engl. Unmanned aerial vehicle, UAV) ist ein Luftfahrzeug, welches ohne Besatzung an Bord betrieben wird. Für die Steuerung können Fernsteuersysteme oder an Bord befindliche Computer zum Einsatz kommen. [ICA11] Der von der ICAO definierte Begriff schränkt wenig ein. Abgesehen vom Ausschluss von Flugmodellen für Freizeit- und Luftsportaktivitäten werden keine Flugobjekte ausgeschlossen. Sowohl moralisch umstrittene militärische Drohnen, als auch die Prototypen der automatisierten Paketzustellung fallen unter die Definition. Im Rahmen des bestehenden Projektes wird ein UAV als Multicopter, der zu zivilen Zwecken eingesetzt wird, verstanden.

Quellen

2.1.1. Multicopter

Multicopter sind Flugobjekte mit mehreren Rotoren, die für Auftrieb sorgen. Einer der bekannten Vertreter ist der Quadrocopter (mit vier Rotoren), darüber hinaus werden unter anderem auch Tricopter, Hexacopter und Octocopter eingesetzt. Die Anzahl der Rotoren beeinflussen in erster Linie die Stabilität in der Luft. Darüber hinaus können

Quellen

Ausfälle einzelner Rotoren bei größerer Anzahl ausgeglichen werden. Eine höhere Rotorenanzahl birgt nicht nur Vorteile. Das Fluggewicht und die nötige Leistung steigen. Daraus folgt, dass entweder größere Energiespeicher nötig sind, welche das Gewicht weiter erhöhen würden oder die Flugdauer mit steigender Rotorenanzahl abnimmt.

Seit einigen Jahren finden UAVs auch in privaten Bereichen Einsatzmöglichkeiten. Sie werden als Spiel- und Spaßwerkzeug zum Beispiel in Verbindung mit Augmented Reality (AR) verwendet. Dafür wird unter anderem eine einfache Anwendung benötigt. Die Geräte müssen nach dem Auspacken, Zusammenbauen und Aufladen ohne Fachkenntnisse benutzbar sein. [BCVP11, S. 1477] Diese Vereinfachung für Endanwender wird durch Bordelektronik, die Steuerung und Stabilisierung unterstützt ermöglicht.

Nachfolgend wird beispielhaft näher auf den Quadrocopter eingegangen. Viele der Informationen sind auf die anderen Multicopterarten übertragbar.

Antrieb

Die Rotoren des Multicopters sind üblicherweise an einen Elektromotor angeschlossen. Die benötigte Energie wird von einem integrierten Energiespeicher geliefert. Der Antrieb eines Multicopters ist üblicherweise ein Elektromotor. Die benötigte Energie wird von einem Energiespeicher geliefert. Nahezu ausschließlich kommen hier Lithium-Polymer-Akkumulatoren zum Einsatz, weitere Details dazu können im Kapitel nachgelesen werden.

Kapitel einfügen

Steuerung und Stabilisierung

2.2. Projektumgebung

Dieses Kapitel stellt die Projektumgebung dar. Zu Beginn wird auf die verwendete Programmiersprache und das benutzte Framework eingegangen. Anschließend wird das bestehende Projekt abseits der zu implementierenden Ladestation vorgestellt.

2.2.1. C++ und Framework OMNeT++

C++ ist eine Erweiterung der Programmiersprache C. Hinzugefügt werden Eigenschaften, die das objektorientierte Paradigma ermöglichen. Durch den Sprachursprung bleibt C++ effizient und mächtig. Beispielsweise eine automatische Speicherbereinigung findet nicht statt. Der Anwender hat alles in der Hand. Dessen ungeachtet kann die Abstraktion der Objektorientierung eingesetzt werden, um übersichtliche und wartbare Software zu schreiben.

Frameworks werden eingesetzt, um den Entwicklungsaufwand zu reduzieren. Dies geschieht unter anderem durch die Bereitstellung von häufig genutzten Funktionen für den jeweiligen Anwendungsbereich. Das im Projekt eingesetzte Framework nennt sich OMNeT++, es basiert auf der Programmiersprache C++. Es beschreibt sich selbst als ëxtensible, modular, component-based C++ simulation library and framework primarily for building network simulators.". Die Nutzung ist für wissenschaftliche Arbeiten gebührenfrei. Die Installation ist durch eine mitgelieferte Entwicklungsumgebung und Anleitung einfach. Eine Tutorialreihe und die online Dokumentation vereinfachen den Einstieg. Das Framework ermöglicht die Erstellung von Objekten, welche miteinander und der Umwelt interagieren können. Darüber hinaus bietet es eine optionale grafische Darstellung, welche den Ablauf insbesondere für Außenstehende veranschaulicht.

Quelle (Webseite)

Quellen

2.2.2. Projekt

Die Arbeit baut auf ein bestehendes Projekt auf. In diesem Projekt "multiUAV-simulation"werden Missionen für UAVs simuliert. Die Missionen setzen sich aus einzelnen Kommandos zusammen, welche in der selben Form auch in der realen Welt genutzt werden können. Eine einfache Mission könnte aus einer Abfolge von mehreren Bewegungskommandos, die unendlich wiederholt werden sollen, bestehen. Im Laufe der Mission wird entleert sich der Akkumulator des eingesetzten UAV. Vor jedem Kommando führt ein UAV eine Ressourcenkalkulation durch und prüft, ob die Energie noch für das nächste Kommando und einen anschließenden Flug zur nächstgelegenen Ladestation reicht. Falls dieser Test ein negatives Resultat liefert, wird das Flugobjekt ausgetauscht und das vorher im Einsatz befindliche fliegt zur nächsten Ladestation.

??

Ein komplexeres Szenario könnte mehrere simultan genutzte UAVs enthalten und zusätzlich eine Begrenzung der vorhandenen Ersatz-UAVs.

2.2.3. Implementierung

Das Projekt nutzt die objektorientierten Möglichkeiten der Sprache C++. Im folgenden Klassendiagramm wird ein Überblick über die vorhandene Struktur zu Beginn der Arbeit gezeigt. Wie in vielen Softwareprojekten wird nicht ein Teil nach dem anderen entwickelt, sondern unterschiedliche Personen arbeiten an unterschiedlichen Teilen des Projektes. Während der Arbeit hat sich auch die Struktur des restlichen Projektes stetig weiterentwickelt.

Klassendiagramm

Ausführung Klassendiagramm

3. Ladeverhalten

Im folgenden Kapitel wird das Ladeverhalten der Ladestation entworfen. Begonnen wird mit dem Vergleich mehrerer Energiespeicher. Dieser Vergleich soll herausarbeiten, aus welchen Gründen in mobilen Anwendungen vorzugsweise Lithium-Ionen basierte Akkumulatoren eingesetzt werden. Darauf aufbauend werden übliche Ladeverfahren und deren Spezifika untersucht. Nach dieser von Literatur angetriebenen Untersuchung, werden Beispielwerte eines Ladegerätes und Akkumulators gemessen. Abschließend wird ein Fazit gezogen, welches die Grundlage für den zu implementierenden Ladealgorithmus liefert.

3.1. Energiespeicher

Im einleitenden Kapitel über Multicopter , wurde festgestellt, dass diese üblicherweise verweis von Elektromotoren angetrieben werden. Elektromotoren benötigen eine elektrische Energiezufuhr, welche von Akkumulatoren bereit gestellt wird.

Akkumulatorentypen lassen sich in Gruppen basierend auf ihren chemischen Bestandteilen einteilen. Drei häufig verwendete Typen basieren auf Lithium-Ionen, Nickel und Bleisäure. Durch die gleiche Basis haben Akkumulatoren innerhalb einer Gruppe ähnliche Merkmale. Für die Auswahl eines Energielieferanten sind neben den funktionalen Eigenschaften Energiedichte (Wh/kg), Kapazität (Ah), Spannung pro Zelle (V) auch weitere nicht funktionale Eigenschaften wie Langlebigkeit, Sicherheit und Ladegeschwindigkeit relevant. In der Anwendung sind die wirtschaftlichen Aspekte ebenfalls ein relevanter Faktor. Für die rein wissenschaftliche Betrachtung ergeben sich hier viele Probleme. Unterschiedliche Marktpreise basieren nicht zwingend auf der Wertigkeit der verwendeten Güter, sondern können auch durch Skaleneffekte der Massenproduktion, Wettbewerb oder Gesetze bezüglich der Technologie entstehen. Diese Effekte

wiederum basieren unter anderem auf der Nachfrage nach dem Produkt und damit auf der Bekanntheit. Dessen ungeachtet wird die Ladestation für aktuelle Multicopter konzeptioniert.

weiterschreiben

3.1.1. Bleisäure Akkumulatoren

In einem Bleisäure Akkumulator besteht die Elektrode aus Blei und Bleioxid, der dazugehörige Elektrolyt besteht aus verdünnter Schwefelsäure. Dieser Akkumulatortype gilt als der älteste Entwurf und reicht bis in das 19. Jahrhundert zurück. Die Nennspannung einer Zelle beträgt 2 V und ist damit ebenso wie die Energiedichte mit 20-50 Wh/kg vergleichsweise gering. Je nach Qualität kann eine Lebensdauer von 200-2000 Ladezyklen erwartet werden. Durch die lange Bekanntheit der Technologie, sind diese Akkumulatoren relativ günstig.

wirklich?

Für den Einsatz in Multicoptern sind diese Akkumulatoren aufgrund der geringen Energiedichte und der geringen Spannung ungeeignet. Einsatzgebiete sind in erster Linie stationäre Anwendungen, für die Energiedichte und Spannung weniger relevant sind oder Anwendungen in denen Robustheit und Sicherheit einen hohen Stellenwert haben, beispielsweise Autobatterien.

3.1.2. Nickel Akkumulatoren

In dieser Gruppe gibt es vor allem zwei relevante Ausprägungen, Nickel-Cadmium und Nickel-Metalhydrid. Nickel-Cadmium Akkumulatoren sind die älteren und in der Energiedichte und Lebensdauer unterlegen. Darüber hinaus ist Cadmium ein giftiger Stoff und muss entsprechend sorgfältig behandelt werden. Der einzige Vorteil gegenüber Nickel-Metalhydrid ist die geringere Selbstentladung, welche für die Anwendung in Multicoptern von untergeordneter Relevanz ist. Nickel-Metalhydrid Akkumulatoren haben eine Energiedichte von 60-80 Wh/kg und Nennspannung von 1,2 V. Die Lebensdauer beträgt weniger als 3000 Ladezyklen. Nachteile sind vor allem die hohe Selbstentladung, die im Vergleich zu Bleiakkumulatoren höheren Kosten und der Memory-Effekt, welcher bei Teilentladungen zu einer kleineren nutzbaren Kapazität führt.

Um das Jahr 2000 waren diese Akkumulatoren die Wahl für mobile Anwendungen. Ältere Multicopter könnten daher auf Nickel Akkumulatoren basiert haben, inzwischen wurden diese jedoch durch Lithium-Ionen basierte Akkumulatoren ersetzt. Eingesetzt werden diese Energiespeicher, wenn die im Vergleich zur Bleisäure größere Energiedichte einen Vorteil darstellt und dennoch die Kosten ausschlaggebend sind, beispielsweise in Fernbedienungen, Taschenlampen oder Elektrowerkzeugen.

3.1.3. Lithium-Ionen Akkumulatoren

In der Regel kommen in mobilen Anwendungen aktuell Lithium-Ionen Akkumulatoren zum Einsatz. Elektroautos, Handys und auch Multicopter werden von ihnen mit Energie versorgt. Innerhalb dieser Akkumulatorgruppe gibt es eine Vielzahl von eingesetzten Verbindungen. Die für Elektromobilität genutzten Akkumulatoren sind meistens Lithium-Polymer Akkumulatoren (LiPo). Diese haben eine Energiedichte von 200 Wh/kg, eine Spannung von 3,8 V und sie können mehr als 1000 Ladezyklen durchleben. Darüber hinaus gibt es bei diesem Batterietyp keinen Memory-Effekt. Doch neben den offensichtlichen Vorteilen entstehen auch einige Nachteile beim Einsatz von LiPo's. Der Akkumulator reagiert sensibler während des Ladeprozesses, eine zu starke Auf- oder Entladung sind schädlich und verringern die Lebensdauer dramatisch. Für einen reibungslosen Ablauf muss der ßtate of charge"bekannt sein und die einzelnen Zellen (wenn mehrere Zellen zum Einsatz kommen) müssen balanciert aufgeladen werden. Darüber hinaus sind diese Akkumulatoren die teuerste Variante, in erster Linie wegen der vergleichsweise neuen Technologie und der begrenzten zur Verfügung stehenden Menge von Lithium.

Trotz Betrachtung der Nachteile sind die hohe Energiedichte und die hohe Spannung ausschlaggebend für den Einsatz in mobilen Anwendungen. Im Rahmen von Elektromobilität stehen diese Akkumulatoren im Zentrum vieler Forschungen. Neben der Verbesserung der bestehenden Verbindungen wird auch mit neuen Batterietypen experimentiert. Ein Beispiel für diese Forschung sind Lithium-Sauerstoff Akkumulatoren, welche eine zehnfach höhere Energiedichte versprechen, aber wenigstens bisher durch ihre kurze Lebensdauer von 50 Zyklen nicht eingesetzt werden.

3.2. Ladeverfahren

Das Ladeverfahren mit dem ein Akkumulator aufgeladen wird beeinflusst diesen maßgebend. Im Bereich der Elektromobilität ist die Ladezeit essentiell. Elektroautos werden unnütz, wenn die tägliche Aufladung den Bedarf für beispielsweise den Arbeitsweg nicht abdecken kann. Im Fall von Multicoptern die eine Mission erfüllen sollen, ist die Audladegeschwindigkeit ausschlaggebend für die Menge der benötigten Fluggeräte. Unter Annahme begrenzter zur Verfügung stehender Ressourcen (Multicopter), wird durch die Ladegeschwindigkeit die Maximalleistung des Schwarms beeinflusst.

Lithium-Ionen Akkumulatoren reagieren sensibel auf Überladung oder zu hohe Spannung. Die chemischen Bestandteile werden beschädigt und die Leistung des Energiespeichers fällt ab. Ein einfacher Ladevorgang mit einer konstanten Stromstärke kommt deshalb nicht in Frage. Je näher der Füllstand der Kapazität des Akkumulators kommt, desto größer wird die Spannung. Lithium-Polymer Akkumulatoren haben üblicherweise eine Maximalspannung von 4,2 V, eine Überschreitung verursacht Schäden und reduziert die Lebensdauer dramatisch.

Name

Prozent

Das üblicherweise genutzte Ladeverfahren, um sowohl eine möglichst hohe Ladegeschwindigkeit, als auch die Lebensdauer der Akkumulatoren zu sichern nennt sich constant current constant voltage CCCV. Dieses Verfahren besteht aus zwei Phasen. In der ersten Phase (CC) wird der leere Akkumulator mit einer konstanten Stromstärke beladen. Im Verlauf des Ladevorgangs steigt die Spannung an. Die Füllmenge des Akkumulators steigt linear über die Zeit an. Der Übergang zur zweiten Phase (CV) findet statt, sobald die Maximalspannung (für LiPo's 4,2V) erreicht ist. Zu diesem Zeitpunkt ist der Akkumulator mit 70-80 Prozent , seiner Kapazität geladen. In der zweiten Phase bleibt die Spannung konstant beim Maximalwert und die Stromstärke nimmt ab. Daraus folgt, dass die Ladegeschwindigkeit nun nicht mehr linear verläuft, sondern immer weiter abfällt, je stärker der Akkumulator geladen ist. Phase zwei endet üblicherweise sobald eine festgelegte minimale Stromstärke (z.B. 100 mA) erreicht wurde. Der Akkumulator ist dann abhängig vom festgelegten Minimalwert gefüllt.

3.3. Beispielmessungen

Die Ausführungen aus der Literatur geben einen Überblick über die Funktionsweise der Akkumulatoren und ihre Aufladung. Mit Hilfe der Beispielmessungen sollen Werte gefunden werden, die den Ladeverlauf realistisch abbilden.

Für die Messungen wurde ein Turnigy Accucell 6 Ladegerät verwendet, welches das CCCV Verfahren benutzt, um unter anderem LiPo Akkumulatoren aufzuladen. Als Testakkumulator wird ein bereits für Drohnenflüge gebrauchter Akkumulator mit einer Maximalkapazität von 1800mAh verwendet. Der Akkumulator wurde vor jedem Ladevorgang mit dem Ladegerät im Modus "Discharge"mit 1A entladen. Für den Ladevorgang wurde der Akkumulator mit einer Stromstärke von 1 A, 2 A und 4 A geladen. Dieser Ablauf wurde dreimal wiederholt um die Wahrscheinlichkeit für zufällige Erscheinungen zu reduzieren.

Akku

Ergebnisstabelle, ein Diagramm

3.4. Fazit

Die Literatur und Testmessungen bringen ähnliche Ergebnisse zum Vorschein. Es wird deutlich, dass der Ladevorgang in der ersten Phase linear verläuft und abhängig von der verwendeten Stromstärke ist. Die Dauer dieser Phase kann durch eine höhere Stromstärke verringert werden. Die Kapazität des Akkumulators zum Start der zweiten Phase ist in der Literatur und den Testmessungen verschieden. Während in der verwendeten Literatur 70-80 angegeben sind, startet Phase zwei in den Testmessungen erst bei 95 . Möglicherweise ist die Nennkapazität des Akkumulators (1800mAh) geringer, als der wirkliche Maximalwert. Der erzeugte Puffer sorgt dafür, dass erstens der Ladevorgang stark beschleunigt wird und zweitens, dass der Akkumulator nicht vollständig geladen werden kann. Da die Lebensdauer des Akkumulators durch vollständiges Entladen und Laden stark abnimmt, kann der Hersteller damit für eine größere Lebensdauer garantieren.

Prozent

Wert und Prozent

Für die Implementierung der Ladestation werden Näherungswerte der Beispielmessungen verwendet.

Nährwerte

4. Anforderungsanalyse

Alle Anforderungen werden natürlich sprachlich ausgedrückt und erhalten eine hervorgehobene Bezeichnung. Diese Bezeichnung wird in der tabellarischen Darstellung am Ende des Kapitels aufgegriffen, welche eine Zusammenfassung der Anforderungen darstellt. Jede Anforderung wird gemäß der MoSCoW-Priorisierung gewichtet. Quelle

Tabelle 4.1. – MoSCoW Akkronym

M	MUST	Diese Anforderung muss erfüllt werden.
S	SHOULD	Diese Anforderung sollte erfüllt werden.
C	COULD	Diese Anforderung könnte erfüllt werden, falls erstens die höher priorisierten Anforderungen abgearbeitet sind und zweitens sie diese nicht negativ beeinflusst.
W	WON'T	Diese Anforderung wird verschoben und nicht in dieser Iteration der Entwicklung betrachtet. Diese Priorisierung wertet nicht die Wichtigkeit der Anforderung für das System, sondern lediglich die Wichtigkeit für die aktuelle Entwicklung. Eine solche Priorisierung erkennt die Relevanz der Anforderung an, verschiebt diese jedoch auf eine spätere Entwicklunsiteration.

4.1. funktionale Eigenschafen

Ladeplätze: Jede Ladestation hat eine begrenzte Anzahl von Ladeplätzen, diese bestimmen wie viele Ladevorgänge gleichzeitig parallel ablaufen können. *Priorität MUST*

Warteplätze: Jede Ladestation hat unbegrenzt viele Warteplätze. Hier können die Multicopter während sie auf ihre Aufladung warten verbleiben und bereits aufgeladene Multicopter können hier bis zu ihrem nächsten Einsatz bleiben. Die Warteplätze sollen

begrenzt werden können. Eine Überschreitung der Kapazität soll eine Fehlerinformation erzeugen. *Priorität MUST*

Verbindung mit Multicopter: Ein Multicopter soll sich mit einer Ladestation verbinden können. Nach der Verbindung kann der Multicopter ruhen und sein Akkumulator wird ohne weitere erforderliche Aktivität aufgeladen. *Priorität MUST*

Realistisches Ladeverhalten: Die Ladestation soll einen realistischen Ladealgorithmus implementieren. *Priorität MUST*

Reservierungen: Ein Multicopter kann einen Ladeplatz einer Ladestation vorzeitig reservieren. Die Abarbeitung der Ladeaufträge soll basierend auf dem Reservierungszeitpunkt erfolgen. *Priorität MUST*

Vorhersagen: Die Ladestation kann eine Aussage über den Ladevorgang treffen und diese Aussage mit anderen Netzwerkteilnehmern kommunizieren. Für Vorhersageanfragen wird unterschieden in zwei Fälle. Erstens soll der Zeitpunkt für einen abgeschlossenen Aufladevorgang abgefragt werden können. Es müssen ein prozentualer Start- und Zielwert berücksichtigt werden. Zweitens soll die Kapazität des Akkumulators, die bis zu einem Zielzeitpunkt aufgeladen werden kann, ausgegeben werden können. Die Vorhersage soll die, zum Zeitpunkt der Anfrage bestehenden, Reservierungen mit einbeziehen. Dadurch ist das Ergebnis einer Vorhersage immer als pessimistische Angabe zu verstehen. Falls andere Reservierungen nicht wahrgenommen werden, ist ein besseres Ergebnis möglich. *Priorität MUST*

Anfrage für Multicopter: Die Ladestation soll auf Ersatzanfragen für Multicopter reagieren können und einen zur Anfrage passenden Multicopter auswählen. *Priorität MUST*

Nutzungsstatistiken: Im Laufe der Simulation sollen anfallende Daten aggregiert und zur Verfügung gestellt werden. Insbesondere soll die Anzahl der erfolgreichen Ladevorgänge und die aufgeladene Energie erfasst werden. *Priorität MUST*

Ressourcenverwaltung: Die Ladestationen sollen über einen Energiespeicher verfügen, welcher als begrenzt oder unbegrenzt konfigurierbar sein soll. Eine versuchte Aufladung an einer Ladestation mit leerem, begrenztem Energiespeicher soll fehlschlagen und eine Fehlerinformation ausgeben. *Priorität SHOULD*

Umverteilung von Multicoptern: Alle Ladestationen sollen untereinander kommunizieren und die zur Verfügung stehenden Multicopter aufteilen. Die Zahlenwerte für die Abgabe und Aufnahme von Multicoptern anderer Ladestationen soll konfiguriert werden können. *Priorität COULD*

Kooperation zwischen Ladestationen: Alle Ladestationen sollen miteinander kooperieren, um Kapazitätsengpässe auszugleichen. *Priorität COULD*

Mobile Ladestation: Jede Ladestation soll über die Fähigkeit zur eigenen Fortbewegung verfügen. *Priorität WON'T*

4.2. nicht funktionale Eigenschafen

Codesprache: Im Quelltext darf ausschließlich die englische Sprache verwendet werden. *Priorität MUST*

Nachrichten Zeitverzögerung: Der Versand und Empfang von Nachrichten soll in der Simulation gleichzeitig erfolgen. Die Dauer der Übertragung soll vernachlässigt werden. *Priorität MUST*

Ladeplatzorganisation Zeitverzögerung: Die Organisation der Ladeplätze soll ohne Zeitverzögerung erfolgen. Die benötigte Zeit, um einen Multicopter an den Lademechanismus an- und abzuschließen soll vernachlässigt werden. *Priorität MUST*

Coderichtlinien: Die bestehenden Coderichtlinien, welche in der automatischen Formatierung der Entwicklungsumgebung hinterlegt sind, sollen eingehalten werden. *Priorität MUST*

Graphische Darstellung: Die Ladestationen sollen in das bestehende Projekt eingepflegt werden und in der graphischen Simulationsansicht angemessen dargestellt werden. *Priorität SHOULD*

4.3. Übersicht

Tabelle 4.2. – Übersicht Anforderungen

Bezeichnung	**Funktional**	**Priorität**
Ladeplätze	Ja	MUST
Warteplätze	Ja	MUST
Verbindung mit Multicopter	Ja	MUST
Realistisches Ladeverhalten	Ja	MUST
Reservierungen	Ja	MUST
Vorhersagen	Ja	MUST
Anfrage für Multicopter	Ja	MUST
Nutzungsstatistiken	Ja	MUST
Ressourcenverwaltung	Ja	SHOULD
Umverteilung von Multicoptern	Ja	COULD
Kooperation zwischen Ladestationen	Ja	COULD
Mobile Ladestation	Ja	WON'T
Codesprache	Nein	MUST
Nachrichten Zeitverzögerung	Nein	MUST
Ladeplatzorganisation Zeitverzögerung	Nein	MUST
Coderichtlinien	Nein	MUST
Graphische Darstellung	Nein	SHOULD

5. Konzeptionierung

Im folgenden Kapitel wird die Konzeptionierung der intelligenten Ladestation dargestellt. Grundlage dafür sind die funktionalen Eigenschaften, sowie das bestehende Projekt. Die Darstellung des Entwicklungsprozesses wird durch UML-Klassendiagramme unterstützt. Dort werden die neu hinzugefügten Eigenschaften und Methoden aufgeführt, deren Entstehung im Text erläutert wird. Die Diagramme geben einen Überblick über die für das behandelte Thema relevanten Aspekte der Klassen. Innerhalb des Kapitels wird es einen Zwischenüberblick geben. Dort wird ein Klassendiagramm, welches alle MUST-Anforderungen abdeckt, präsentiert. Im abschließenden Überblick werden die erledigten SHOULD- und COULD- Anforderungen ergänzt.

5.1. Grundlegende Designentscheidungen

Einige grundlegende Designentscheidungen werden durch das bestehende Projekt vorgegeben. Die Entwicklung ist auf ein Objektorientiertes Modell in der Programmiersprache c++ und das Simulationsframework OMNeT++ festgelegt. Konfigurationen werden aus einer zentralen Datei eingelesen und können während der Initialisierung der Objekte in Eigenschaften überführt werden.

Das Netzwerk der Simulationsobjekte wird in (.ned) Dateien definiert. Neben den Informationen für die grafische Darstellung werden hier auch die Verbindungen der einzelnen Elemente festgelegt. Bevor Netzwerkteilnehmer miteinander kommunizieren können muss eine Verbindung definiert werden. Die Ladestationen müssen mit den Multicoptern und den anderen Ladestationen kommunizieren können.

Objekteigenschaften sind im Sinne der Datenkapselung protected und je nach Bedarf werden sogenannte Getter- und Settermethoden für den äußeren Zugriff hinzugefügt.

Quelle

5.2. Nachrichten

Nachrichten die nur Namen haben

Für die Kommunikation zwischen verschiedenen Netzwerkteilnehmern stellt OMNeT++ Nachrichten zur Verfügung. Nachrichten sind Objekte vom Typ cMessage. Darüber hinaus können eigene Nachrichtentypen in speziellen (.msg) Dateien definiert werden. Die Nachrichtentypen unterscheiden sich durch ihre Bezeichnung und enthalten individuelle Parameter. Diese vereinfachten Klassendefinitionen werden während des Kompilierprozesses in vollwertige c++ Klassen übersetzt, welche alle nötigen Eigenschaften und Methoden für den Versand enthalten. Parameter können über Setter-Methoden angehangen und über Getter-Methoden ausgelesen werden.

Damit die Nachricht erfolgreich versendet werden kann muss außerdem der richtige Empfänger bekannt sein. Die definierten Verbindungen (5.1) werden in der Simulation mit In- und Outputgates organisiert. Ein Gate verbindet dabei immer genau zwei Netzwerkteilnehmer. Durch die Unterteilung in In- und Output können so auch einseitige Kommunikationswege abgebildet werden. Da die Auswahl des richtigen Gates für den Nachrichtenversand ein für alle Objekte relevantes Problem ist, wird eine Helfermethode in der GenericNode geplant. Diese Methode soll das passende Gate zu einem übergebenen Objekt im Netzwerk zurückgeben.

Die Nachrichten werden in den Objekten in der handleMessage() Methode verarbeitet. Basierend auf der Bezeichnung kann der Nachrichtentyp identifiziert werden. Nach dem das Objekt der Basisklasse cMessage in ein Objekt der abgeleiteten Klasse überführt wurde, kann auf die individuellen Parameter zugegriffen werden.

Im Verlauf der Konzeptionierung werden verschiedene Nachrichtentypen hinzugefügt, um die Kommunikation mit der Ladestation zu ermöglichen. Diese werden nur in

ihrer vereinfachten Form modelliert, da die Entwicklungsumgebung die Übersetzung automatisch übernimmt.

5.3. GenericNode

Die GenericNode stellt ein Grundgerüst für Objekte innerhalb der Simulation zur Verfügung. Die Klasse für die Ladestation ChargingNode wird von der GenericNode abgeleitet. Durch die Bereitstellung der im Diagramm gezeigten Eigenschaften und Methoden, kann sich die Weiterentwicklung auf die relevanten Eigenschaften für die intelligente Ladestation fokussieren.

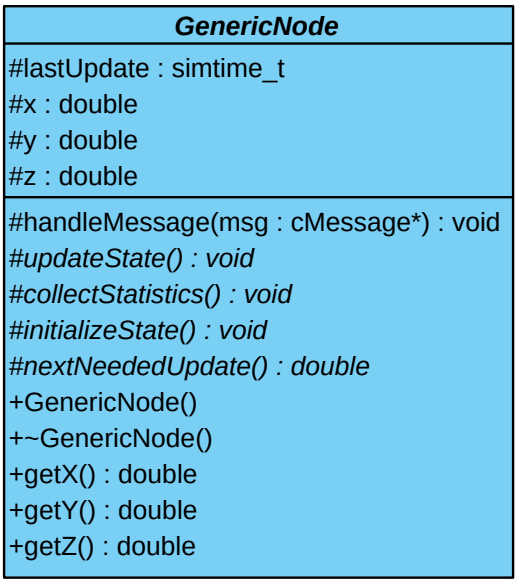
verweis

Nach der Instanziierung der Klasse wird die initializeState() Methode aufgerufen. Diese ließt die Konfigurationsdatei ein und übergibt die Werte für die Koordinaten x, y, z und die timeStep Eigenschaft. Anschließend wird das Objekt auf der Karte gemäß der angegebenen Koordinaten platziert. Darüber hinaus wird ein regelmäßiger Aufruf vorbereitet. Nach einer initialen Aktualisierungsnachricht wird die updateState() Methode aufgerufen. Hier werden die regelmäßigen Aufgaben definiert. Abschließend wird die Eigenschaft lastUpdate mit der aktuellen Zeit belegt und mit Hilfe der Methode nextNeededUpdate() eine neue Aktualisierungsnachricht generiert.

Abschließend wird die im Nachrichtenkapitel (5.2) aufgegriffene Helfermethode für den Nachrichtenversand hinzugefügt.

Diagramm ändern

Abbildung 5.1. – GenericNode UML-Klassendiagramm



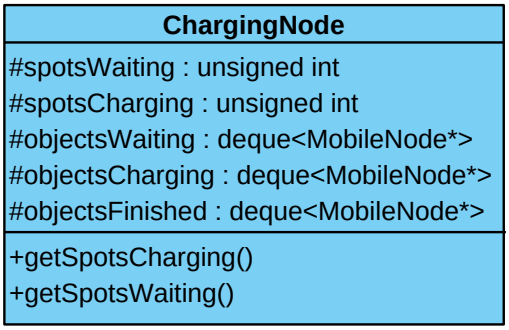
5.4. Lade- und Warteplätze

Für die Lade- und Warteplätze muss die Anzahl der jeweiligen Plätze gespeichert werden. Darüber hinaus müssen Containerobjekte für die Multicopter vorhanden sein. Abgelegt werden Verweise auf die Objekte. Die Multicopter werden in der Simulation als UAVNode dargestellt. Deren Basisklasse ist die MobileNode, welche bereits den Energiespeicher beinhaltet. Der Akkumulator ist der für die Ladestation relevante Aspekt ist. Daher liegt es Nahe jede MobileNode als Konsumenten zuzulassen.

An der Ladestation können die Objekte in drei unterschiedliche Phasen sein. Das Objekt wartet auf seine Aufladung, es wird gerade aufgeladen und es wurde bereits fertig aufgeladen. Um diese drei Zustände klar voneinander zu trennen werden drei ähnliche Containerobjekte konzeptioniert. Die Container müssen dabei eine variable Größe haben können und Objekte müssen an jeder Stelle des Container hinzugefügt und entfernt werden können. Aus der c++ Bibliothek bietet sich dafür die deque (double-ended queue) an.

Während die Anzahl der Plätze von externen Objekten einsehbar sein soll, sollten die Container ausschließlich vom Objekt selbst und möglichen Erben eingesehen und geändert werden können.

Abbildung 5.2. – ChargingNode UML-Klassendiagramm



5.5. Verbindung mit Multicopter

Der Anschluss eines Multicopters an die Ladestation benötigt eine Kommunikationsschnittelle. Als trivialer Anfang wird dafür eine Nachricht konzeptioniert, die vom Multicopter an die Ladestation versendet wird, sobald diser bereit für den Ladeprozess ist. Die Ladestation muss nach dem Empfang einer solchen Nachricht prüfen, ob der Multicopter an der Ladestation ist und kann ihn dann zu den wartenden Objekten hinzufügen.

Diagramm

5.6. realistisches Ladeverhalten

Zentrales Element einer Ladestation ist der zugehörige Ladealgorithmus. Im Kapitel über das Ladeverhalten (Kapitel 3) wurde bereits festgestellt, dass Akkumulatoren und die zugehörigen Ladealgorithmen zu aktuellen Forschungsthemen gehören. Während

der Konzeptionierung von Software ist es essentiell die Erweiterbarkeit und Austauschbarkeit der einzelnen Komponenten zu beachten. Ein Interface stellt eine Beschreibung für eine zukünftige Implementierung bereit. Es werden Methoden definiert, welche von den erbenden Klassen zwingend implementiert werden müssen. Dadurch wird sichergestellt, dass die unterschiedlichen Implementierungen eines Interfaces untereinander kompatibel sind. Für den Ladealgorithmus werden ein Interface IChargeAlgorithm und eine Implementierung davon ChargeAlgorithmCCCV konzeptioniert. Der verwendete Ladealgorithmus kann während der Initialisierung spezifiziert werden und in eine Eigenschaft der Ladestation gespeichert werden. Für den Algorithmus relevante Eigenschaften müssen an dieser Stelle übergeben werden, so auch die verwendete Stromstärke, welche maßgebend Einfluss auf die Ladegeschwindigkeit hat. Anschließend kann über die Methode calculateChargeAmount() die zu ladende Menge für einen Zeitraum abgefragt werden. Die Methode calculateChargeTime() hingegen errechnet für einen Zielladezustand die benötigte Zeit.

Damit die Multicopter an der Ladestation geladen werden können müssen diese von den Warteplätzen auf die Ladeplätze verschoben werden. Dafür muss bei jeder Aktualisierungsnachricht die Methode scheduleChargeSpots() ausgelöst werden, dort werden fertig geladene Multicoper in den entsprechenden Container verschoben und anschließend die freien Warteplätze gefüllt. Die Auswahl des nächsten zu ladenden Multicopters ist in eine weitere Methode getNextWaitingObjectIterator() ausgelagert. Diese Auslagerung soll bei einem komplexen Auswahlalgorithmus die Gesamtkomplexität der Anwendung reduzieren. Im trivialsten Fall wird hier nur das erste Objekt aus dem Warteplätzecontainer ausgewählt. Nachdem die Ladeplätze gefüllt wurden, wird die Methode charge() aufgerufen. Diese nutzt die lastUpdate Eigenschaft aus der GenericNode, um die vergangene Zeit seit dem letzten Ladevorgang zu errechnen. Basierend auf der vergangenen Zeit seit dem letzten Ladevorgang wird für jeden Multicopter auf einem Ladeplatz die die aufzuladene Kapazität errechnet. Im Anschluss wird die Batterie des Multicopters um den entsprechenden Wert aufgeladen. Abschließend muss der Wert der letzten Aktualisierung auf den momentanen Zeitpunkt gesetzt werden und ein Aufladezyklus ist abgeschlossen.

diagramm

5.7. Reservierungen und Vorhersagen

Für andere Netzwerkteilnehmer ist die Information wie ein potenzieller Ladeprozess verlaufen würde wohlmöglich interessant, um entscheiden zu können ob dieser wahrgenommen wird. Die Ladestation im aktuellen Zustand könnte Vorhersagen über die vermutliche Ladedauer abgeben und dabei alle in den Warte- und Ladecontainern befindlichen Multicopter einbeziehen. Neue Multicopter können allerdings ohne vorherige Information jederzeit eintreffen und der Warteliste hinzugefügt werden. Vorhersagen für Multicopter die noch nicht eingetroffen sind, werden dadurch entwertet und nicht verlässlich. Damit die Ladestation ihre Vorhersagen einhalten kann, muss den Multicoptern eine Möglichkeit gegeben werden, einen Platz auf der Warteliste zu reservieren, ohne das diese bereits an der Ladestation angekommen sind. Reservierungen werden per Nachricht eingeleitet. Die Nachricht enthält dabei Informationen über die voraussichtliche Ankunftszeit, den aktuellen Ladezustand und den voraussichtlichen Verbrauch bis zur Ankunft an der Ladestation. Darüber hinaus ist ein prozentualer Zielzustand enthalten. Dadurch wird den Multicoptern die Möglichkeit geboten nur eine Teilaufladung zu erhalten. Diese Informationen, der Absender und der Zeitpunkt der Reservierung müssen innerhalb der Ladestation gespeichert werden. Dadurch kann bereits zum Zeitpunkt der Reservierung eine Vorhersage über die Ladedauer getroffen werden. Da für jede Vorhersage die Fertigstellungszeitpunkte der vorherigen Multicopter benötigt werden, wird dieser Zeitpunkt mit im Container abgespeichert. Die individuelle Containerklasse benötigt eine Eigenschaft für jede zu speichernde Information und je eine Getter- und Settermethode für den Zugriff. Die Warte- und Ladeliste wird nun nicht mehr von den Multicoptern selbst gefüllt, sondern von eigenen Containerobjekten (ChargingNodeSpotElement).

Die Reservierungen ermöglichen der Ladestation die Warteliste basierend auf der Reservierungszeit zu organisieren. Dafür muss der Auswahlalgorithmus angepasst werden und die Reservierungszeit einbeziehen. Durch diese Änderung kann die Ladestation ihre Vorhersagen garantieren. Falls frühere Reservierungen nicht wahrgenommen werden, kann die Aufladezeit sich verringern, aber eine Verschlechterung ausgehend von der Vorhersage ist nicht möglich.

Die Vorhersagen sind in zwei unterschiedliche Formen vorgesehen. Für die interne Verwendung ist der Fertigstellungszeitpunkt relevant. Dieser wird ebenfalls externen Interessenten zur Verfügung gestellt. Darüber hinaus kann eine Vorhersage über den Ladezustand zu einem spezifizierten Zeitpunkt angefragt werden. Deshalb gibt es zwei verschiedene Nachrichten für eine Vorhersagenanfrage. Die Antwort hingegen hat immer den selben Nachrichtentyp, in dieser ist sowohl der erreichte Ladezustand, als auch der Zeitpunkt der Fertigstellung enthalten.

Abbildung 5.3. – Nachrichten für Vorhersagen UML-Klassendiagramm

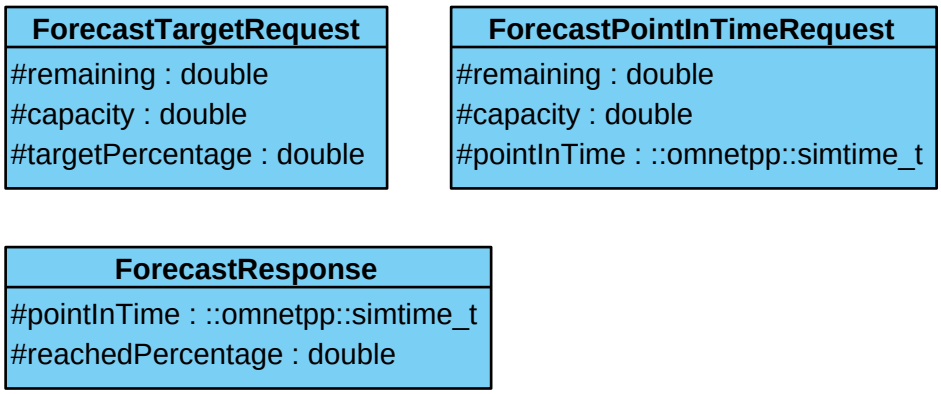
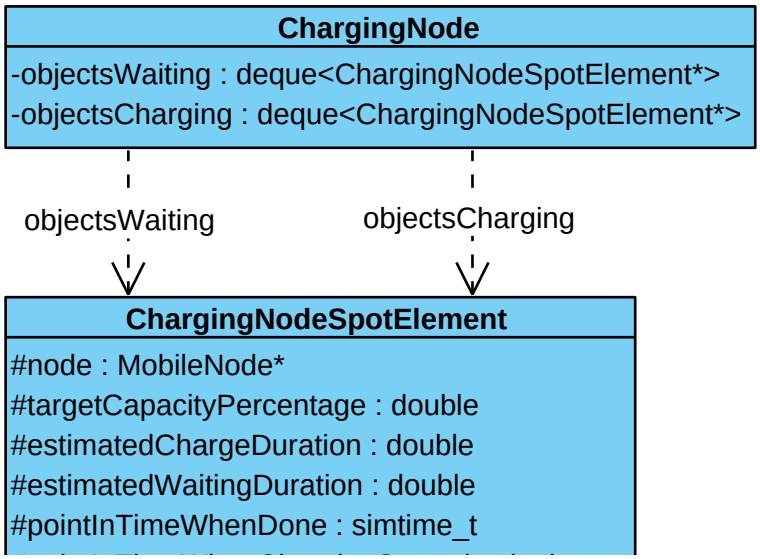


Abbildung 5.4. – Containerklasse UML-Klassendiagramm



6. Results and Comparative Analysis

In this chapter an analysis and evaluation of the TimeNET implementation of the multi-trajectory algorithm is presented. First, an analysis of its parameters is conducted and for this purpose a model is designed. Execution of this model with all possible values of different multi-trajectory parameters are shown, and the best combination of parameters is found. Afterwards, a comparative analysis of the implementation with the other three TimeNET implemented simulation algorithms is presented. This analysis is based on a previous research conducted by the author of the current work, *Rare-Event Algorithms Analysis and Simulation* [**?**]. Its goals were to evaluate RESTART TimeNET implementation and the implemented heuristic for its importance function. The multi-trajectory simulation results are compared together with the results obtained from the earlier work, followed by an analysis of how variations of the model properties exert different impact on the execution time of the algorithms. Finally, a non-Markovian model is analyzed and different simulations executed on it are presented.

6.1. Multi-Trajectory Results

In this section the results from the multi-trajectory simulations are compared with the results from the previous work. The multi-trajectory algorithm has been executed with the default parameters over all variations of the models, but only the most significant results are presented here.

Model Variations					Standard Simulation Results			
Mod	Pla	Tok	Del	Math Res	CPU Time	Overall T.	Value	Error
1	5	1	10	9.00E-05	0:00:13	0:00:03	9.20E-05	2.22%
1	5	1	100	9.90E-09	2:41:08	0:28:59	9.19E-09	7.17%
1	5	1	1000	9.99E-13		» 24 hs		
1	5	3	10	2.70E-04	0:00:11	0:00:03	2.71E-04	0.37%
1	5	3	100	2.97E-08	2:45:31	0:29:44	2.83E-08	4.71%
1	5	3	1000	3.00E-12		» 24 hs		
1	5	10	10	9.00E-04	0:00:29	0:00:06	9.23E-04	2.56%
1	5	10	100	9.90E-08	2:48:56	0:32:47	9.67E-08	2.32%
1	5	10	1000	9.99E-12		» 24 hs		
2	5	1	10	6.83E-05	0:00:30	0:00:06	6.92E-05	1.32%
2	5	1	100	9.61E-09	2:41:24	0:28:34	8.95E-09	6.87%
2	5	1	1000	9.96E-13		» 24 hs		
2	5	3	10	2.05E-04	0:00:11	0:00:03	2.04E-04	0.49%
2	5	3	100	2.88E-08	2:49:14	0:29:59	2.63E-08	8.68%
2	5	3	1000	2.99E-12		» 24 hs		
2	5	10	10	6.83E-04	0:00:17	0:00:04	6.79E-04	0.59%
2	5	10	100	9.61E-08	3:09:25	0:34:05	9.32E-08	3.02%
2	5	10	1000	9.96E-12		» 24 hs		

Tabelle 6.1. – Standard Simulation Results [**?**]

Model Variations					RESTART Simulation Results			
Mod	Pla	Tok	Del	Math Res	CPU Time	Overall T.	Value	Error
1	5	1	10	9.00E-05	0:00:15	0:00:04	8.87E-05	1.44%
1	5	1	100	9.90E-09	0:02:20	0:00:25	9.74E-09	1.62%
1	5	1	1000	9.99E-13	0:47:54	0:08:01	1.10E-12	10.11%
1	5	3	10	2.70E-04	0:00:17	0:00:05	3.45E-04	27.78%
1	5	3	100	2.97E-08	0:03:27	0:00:35	3.06E-08	3.03%
1	5	3	1000	3.00E-12	0:59:34	0:11:24	3.00E-12	0.00%
1	5	10	10	9.00E-04	0:00:22	0:00:04	1.06E-03	17.78%
1	5	10	100	9.90E-08	0:13:53	0:02:21	1.17E-07	18.18%
1	5	10	1000	9.99E-12	2:20:50	0:23:54	9.84E-12	1.50%
2	5	1	10	6.83E-05	0:00:17	0:00:05	6.74E-05	1.32%
2	5	1	100	9.61E-09	0:00:35	0:00:07	9.75E-09	1.46%
2	5	1	1000	9.96E-13	0:21:25	0:03:38	9.52E-13	4.42%
2	5	3	10	2.05E-04	0:00:17	0:00:04	2.14E-04	4.39%
2	5	3	100	2.88E-08	0:00:47	0:00:10	3.05E-08	5.90%
2	5	3	1000	2.99E-12	0:36:54	0:06:19	3.49E-12	16.72%
2	5	10	10	6.83E-04	0:00:17	0:00:04	7.18E-04	5.12%
2	5	10	100	9.61E-08	0:01:15	0:00:13	9.51E-08	1.04%

A. Appendix

This appendix presents the details of the stationary analysis and multi-trajectory simulation executions of the model introduced in Figure ?? (Section ??, page ??).

Interconnected Model Analysis Output

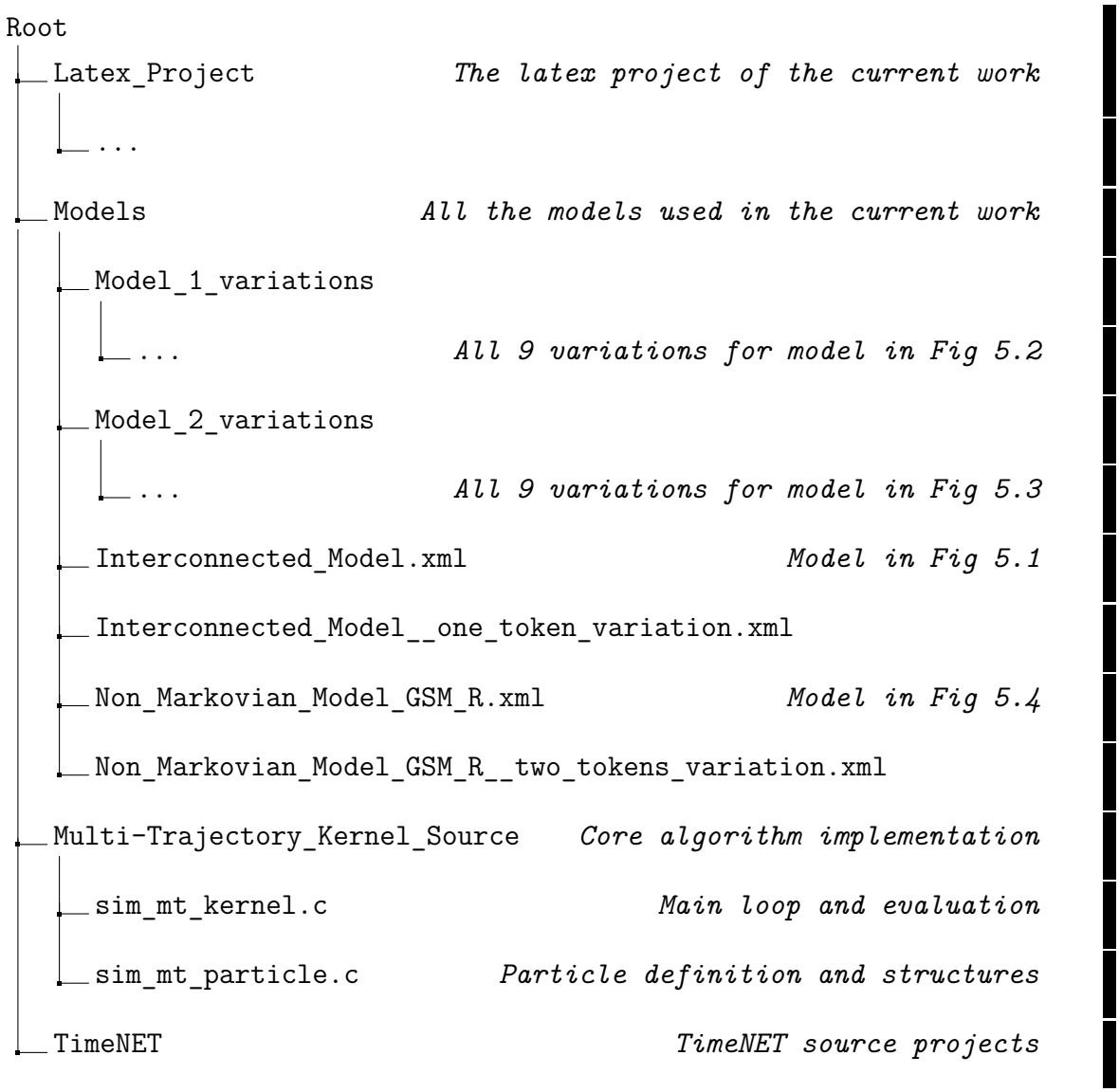
Interconnected Model Analysis Output

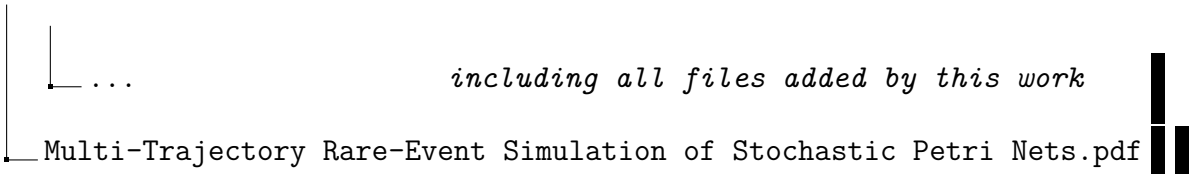
```
Analysis Output:

STEADY STATE SOLUTION OF NET Interconnected_Model


STRUCTURAL ANALYSIS ...


GENERATING THE REDUCED REACHABILITY GRAPH USING TIME EFFICIENT ALGORITHM ...
The reduced reachability graph of this eDSPN
cannot be generated because of lacking main memory.

ERROR occurred while derive_SMC execution.
SOLUTION OF MODEL Interconnected_Model FAILED.

Removing temporary files.


--------------------------------------------------------------------------------
                              Analysis error, see above.
```

```
-------------------------------------------------------------------------------
c@FancyVerbLinee-------------------------------------------------------------------------
```

CD Structure

A soft version of the current work is submitted. Following, the CD structure is presented.




...
including all files added by this work
Multi-Trajectory Rare-Event Simulation of Stochastic Petri Nets.pdf

Tabellenverzeichnis

Abbildungsverzeichnis

List of Source Code

Literaturverzeichnis

[BCVP11] Bristeau, Pierre-Jean ; Callou, François ; Vissière, David ; Petit, Nicolas: The Navigation and Control technology inside the AR.Drone micro UAV. In: *IFAC Proceedings Volumes* 44 (2011), Nr. 1, S. 1477–1484. http://dx.doi.org/10.3182/20110828-6-IT-1002.02327. – DOI 10.3182/20110828–6–IT–1002.02327. – ISSN 14746670 2.1.1

[ICA11] ICAO: *ICAO circular*. Bd. 328: *Unmanned aircraft systems: (UAS)*. Montréal : International Civil Aviation Organization, 2011. – ISBN 978–92–9231–751–5 2.1

Liste der noch zu erledigenden Punkte

Declaration Of Authorship

I, Ludwig Breitsprecher, Matriculation Number 54131, hereby declare that my Bachelorarbeit with the Title

> *Konzeptionierung und Implementierung einer intelligenten Ladestation in einer diskreten eventbasierten Simulationsumgebung*

was produced by me, independently and without using any other references than those mentioned. This work is submitted in completion of the requirements for the Master Degree Research in Computer and Systems Engineering in Ilmenau University of Technology, and I did not publish nor submit this thesis to another university examination board.

Ilmenau, XX.XX.20XX

Ludwig Breitsprecher