Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwicklung eines neuronalen Netzwerks zur Optimierung der Datenübertragungsqualität von Kleinsatelitenplattformen

Cedric Holeczek

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Steffen Staab, Prof. Dr.-Ing. Sabine Klinkner
Betreuer/in:	Dr. Nico Potyka, Susann Pätschke, M.Sc.
Beginn am:	1. Februar 2021
Beendet am:	2. November 2021

Kurzfassung

Kleinsatelliten sind heute eine zunehmend wichtige Möglichkeit für die Forschung, Experimente und Nutzlasten in Erdumlaufbahnen zu bringen. Bei diesen Satellitensystemen besteht einerseits ein wachsender Bedarf an Datenübertragung zu einer Bodenstation, andererseits gelten in der Raumfahrt verschiedene Einschränkungen bei Aufbau und Betrieb von Funkkommunikationssystemen. Heutige Funksysteme haben meist statische Transceiver-Konfigurationen und die genutzten Übertragungsraten entsprechen der Worst-case-Betrachtung für den jeweiligen Fall. Neuerdings werden adaptive Ansätze für den Betrieb von Funkverbindungen zwischen Satellit und Bodenstation beschrieben, bei denen die Transceiver-Konfiguration, gesteuert durch einen lernenden Algorithmus, im Betrieb verändert und den jeweiligen äusseren Einflüssen angepasst wird. Die Grundlage für die verwendeten Algorithmen ist das Reinforcement Learning Model. Ziel dieser Arbeit war die Weiterentwicklung eines für die Raumfahrtfunkkommunikation entwickelten Algorithmus und der Test dieses Algorithmus' innerhalb einer Simulationsumgebung, um die Auswirkungen der Veränderungen zu bewerten. Der modifizierte Algorithmus wurde in einer Software-Simulationsumgebung betrieben, welche die digitale Signalverarbeitung von Sender und Empfänger sowie die sich verändernden Übertragungsbedingungen während eines Satellitenüberfluges abbildete. Die Modifikation wurde mit dem Originalalgorithmus hinsichtlich der erreichbaren Datenübertragung verglichen. Unter anderem wurde die Anzahl der für das Lernen verwendeten Neuronalen Netze variiert. Weiterhin wurden verschiedene Hyperparameter variiert und die Auswirkungen auf die Datenübertragung untersucht. Eine Anpassung der Hyperparameter führte dabei zu einer Übertragung von 57,8% mehr Daten als bei der Baseline-Implementierung. Sowohl die Veränderung in der Architektur als auch die Reduzierung der parallel ausgeführten Neuronalen Netze führte zu leichten Performanzeinbußen von 4,0% und 3,3%. Die Ergebnisse der hier durchgeführten Software-Simulationen lassen sich jedoch nicht auf Hardware-Simulationsumgebungen oder tatsächliche Funkverbindungen übertragen. Die Implementierung der verschiedenen Algorithmusvarianten erlaubt es, diese Varianten zukünftig auch mit Hardware-Übertragungsstrecken zu testen und später ein solches System auf einem Kleinsatelliten zu erproben.

Abstract

Today, small satellites are an increasingly important way for researchers to launch experiments and payloads into Earth orbit. On the one hand, these satellite systems have a growing need for data transmission to a ground station; on the other hand, various constraints apply to the design and operation of radio communication systems in space. Today's radio systems usually have static transceiver configurations and the transmission rates used correspond to the worst-case consideration for the particular case. Recently, adaptive approaches for the operation of radio links between satellite and ground station have been described, in which the transceiver configuration, controlled by a learning algorithm, is changed during operation and adapted to the respective external influences. The basis for the algorithms used is the reinforcement learning model. The goal of this work was the further development of an algorithm developed for satellite communication and the testing of this algorithm within a simulation environment in order to evaluate the effects of the changes. The modified algorithm was run in a software simulation environment that included the digital signal processing of the transmitter and receiver and a model for the changing transmission conditions during a satellite overflight. The modification was compared with the original algorithm with respect to the achievable data transmission. Among other things, the number of neural networks used for learning was varied. Furthermore, different hyperparameters were varied and the effects on the data transmission were examined. An adjustment of the hyperparameters led to the transmission of 57.8% more data than in the baseline implementation of the algorithm. Both, the change in architecture and the reduction in the number of neural networks executed in parallel, led to slight performance losses of 4.0% and 3.3%, respectively. The results of the software simulations performed here, however, cannot be directly transferred to hardware simulation environments or actual communication links. The implementation of the different algorithm variants allows to test these variants with hardware transmission links in the future and to test such a system on a small satellite later on.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Kleinsatellitenplattformen

Durch die Nutzung von Commercial off-the-shelf Komponenten und kommerziell erhältlichen standardisierten Bus Designs wurden die Entwicklungskosten für Kleinsatelliten soweit reduziert, dass die Kosten für den Start immer ausschlaggebender werden. Da sich diese Kosten hauptsächlich aus der Masse des Satelliten ergeben [CSH14], besteht ein großer Vorteil von Kleinsatelliten, die maximal 600 Kilogramm wiegen, in den geringen Startkosten. Aufgrund dessen wurde der Bau von Kleinsatelliten immer verbreiteter [ST20]. Die niedrigeren Startkosten dieser masseärmeren Satelliten ermöglichen es auch Institutionen, wie zum Beispiel Universitäten, die kleinere Budgets haben als Weltraumagenturen, Militär oder Großfirmen, Satelliten zu starten [ST20].

1.2. Constraints in der Raumfahrt

In der Raumfahrt existieren mehrere Einschränkungen und Bedingungen an die eingesetzte Hardware. Zum einen werden möglichst leichte und hochintegrierte Komponenten verwendet, um die Startkosten zu minimieren. Da Satelliten einem sehr großen Temperaturgradient ausgesetzt sind, muss die Hardware sowohl extremer Wärme als auch extremer Kälte standhalten. Außerdem ist es erforderlich, die Komponenten teilweise in Strahlungsumgebungen funktionieren, da in verschiedenen Orbits unterschiedlich hohe Strahlung vorhanden ist, die nicht darauf ausgelegte Hardware beschädigen oder zerstören kann.

Auf Grund der hohen Kosten für den Bau und Start eines Satelliten, werden für die wichtigen Systeme in der Regel nur solche Komponenten benutzt, die bereits in funktionsfähigen Satelliten zum Einsatz kamen und sehr intensiv getestet wurden, um eine möglichst hohe Wahrscheinlichkeit für die Funktionsfähigkeit des Satelliten nach dem Start zu erreichen.

Da im Orbit sowohl die Stromerzeugung als auch die Möglichkeit der Abstrahlung von Verlustleitungen der Komponenten beschränkt sind, besteht die Notwendigkeit, Komponenten mit einer geringen elektrischen Leistung einzusetzen. In Folge dieser zahlreichen Anforderungen und Einschränkungen wird Hardware eingebaut, die typischerweise älter und weniger leistungsstark ist. Bemerkbar macht sich dies beispielsweise an der verfügbaren Rechenleistung der On-Board-Computer. Deshalb sind komplexe und rechenleistungsintensive Berechnungen auf einem Satelliten meist nicht möglich.

1.3. Kommunikation

Ein Satellit besteht üblicherweise aus einem Bussystem, das alle notwendigen Systeme für seinen Betrieb enthält, wie zum Beispiel den Hauptcomputer oder die Stromversorgung, und aus Nutzlasten, welche die Systeme zur Ausführung der Aufgaben des Satelliten enthalten, wie beispielsweise Kameras. Ein wichtiger Bestandteil des Bussystems ist das Kommunikationssystem eines Satelliten, das die Datenübertragung zwischen Satellit und Bodenstation durchführt. Ohne eine Datenübertragung kann die Satellitenmission nicht erfolgen, da ihm keine Kommandos übertragen werden können und es auch nicht möglich ist, Daten des Satelliten zu empfangen.

Die Datenübertragungen zwischen Bodenstation und Satellit können in Uplink und Downlink eingeteilt werden. Mit dem Uplink werden Daten von der Bodenstation zum Satelliten übertragen. Dies sind hauptsächlich Kommandos, die der Satellit ausführen soll, und Softwareupdates. Die Daten, welche mit dem Downlink übertragen werden, sind zum einen Telemetriedaten, die die Zustandsinformationen eines Satelliten beschreiben, und zum anderen Nutzlastdaten, die von der Mission des Satelliten abhängen, zum Beispiel Bilder einer Kamera. Für den Downlink der Nutzlastdaten wird meist eine hohe Datenrate benötigt.

Um Daten zu übertragen, muss ein Link zwischen Bodenstation und Satellit aufgebaut werden. Dies ist nur möglich, wenn Bodenstation und Satellit durch eine Sichtlinie verbunden sind. Die Qualität der Übertragung ist von vielen Parametern abhängig. Ein essentieller Parameter ist die Signalstärke. Die gesendete Signalstärke nimmt durch verschiedene Faktoren ab, wie beispielsweise durch Atmosphärendämpfung oder Ausrichtungsungenauigkeiten. Damit ein Link zustande kommt, muss die Signalstärke, die empfangen wird, einen gewissen Threshold aufweisen. Diese wird bei der Entwicklung eines Satelliten mit Hilfe von Linkbudgets berechnet.

Die Linkbudgets werden so kalkuliert, dass selbst im schlechtesten Fall ein Link aufgebaut werden kann. Ein solcher Worst-Case ist eine Übertragung bei der unter anderem die Distanz der Sichtlinie zwischen Bodenstation und Satelliten maximal ist und weitere Verluste wie zum Beispiel Regendämpfung auftreten. Die meist nicht veränderliche Konfiguration dieser Werte erfolgt in der Kommunikationssoft und -hardware. Da nicht immer der schlechteste Fall der Datenübertragung vorliegt, besteht noch ein nicht ausgeschöpftes potenzielles Datenvolumen. Somit wird in den meisten Fällen nicht die höchstmögliche Datenmenge übertragen, was zu einem verringerten Betrieb der Nutzlast führen kann.

1.4. Adaptive Ansätze in der Kommunikation

Um die im Unterkapitel 1.3 dargestellten Probleme zu lösen, wird aktuell an adaptiven Ansätzen für die Kommunikation geforscht [FPW+18][DKK+15][JJD19]. Da die Rechenleistung auf einem Satelliten eingeschränkt ist, gibt es den Ansatz, die nötigen Berechnungen am Boden durchzuführen. Hierbei analysiert die Bodenstation das empfangene Signal, berechnet die für eine bessere Übertragung erforderlichen Funkparameter und sendet diese Werte zum Satelliten. Dort erfolgt die Konfiguration des Transmitters mit den neuen Parametern und somit die Verbesserung der Übertragung, siehe [DKK+15].

Ein Problem dieses Ansatzes ist die Signallaufzeit, die bei Satelliten im geostationären Orbit bei ungefähr 120 Millisekunden liegt. Daher entsteht eine Verzögerung von mindestens 240 Millisekunden, da das empfangene Signal 120 Millisekunden alt ist, und das Kommando an den Satelliten mit den neuen Parametern weitere 120 Millisekunden Signallaufzeit hat. In dieser Abschätzung sind weder die Rechenzeit am Boden für die Auswertung des Signals und die Festlegung der neuen Parameter noch die Zeit für das Verarbeiten der Daten und die Konfiguration des Transmitters im Satelliten berücksichtigt. Die Verzögerung von 240 Millisekunden bildet also eine untere Grenze. Zur Lösung dieses Problems bestehen Konzepte, bei denen der Algorithmus versucht, den Zustand des Übertragungsweges vorherzuberechnen, siehe [FMW14].

Neben der Durchführung der erforderlichen Berechnung am Boden ist ein weiterer Ansatz, die Berechnung der Parameter mit Hilfe von Reinforcement Learning auf dem Satelliten durchzuführen. Auf diese Weise wird das Problem der Übertragungsdauer deutlich reduziert, da der Satellit sowohl das empfangene Signal auswertet als auch die Konfigurationswerte für den Transceiver, welcher aus einem Transmitter und einem Receiver besteht, berechnet. Ein Vorteil des Reinforcement Learning Algorithmus ist, sein gutes Anpassungspotenzial an dynamische Umgebungsbedingungen.

In dieser Arbeit soll ein Reinforcement Learning Algorithmus weiterentwickelt und implementiert werden, der Daten mit Hilfe eines Software Defined Radios, das ein Transceiver ist, welcher durch Software frei konfiguriert werden kann, empfängt, um dann aus diesen Daten, die beim Empfang gemessen wurden, wie beispielsweise das Signal zu Rausch Verhältnis, die Parameter für das Senden der Daten zu optimieren, indem der Algorithmus angepasste Transmitterkonfigurationen bestimmt. Die Datenübertragung soll im S-Band erfolgen, das den Frequenzbereich von 2,0 bis 4,0 Gigahertz umfasst [ITU15] und entsprechend den Consultative Committee for Space Data Systems (CCSDS) Standards [AA20] und [AA17a] erfolgen, die in der Datenübertragung mit Satelliten weit verbreitet sind und Vorgaben für die einzelnen Open Systems Interconnection Reference Model (OSI)-Layer machen. Das Ziel ist die Entwicklung eines Programms, welches mit einem Software Defined Radio im S-Band den CCSDS Standards folgend die Übertragungsqualität adaptiv verbessert. Eine Einschränkung ist, dass diese Software auf einer leistungsbeschränkten Plattform, wie zum Beispiel einem Laptop, funktionieren muss.

Im folgenden Kapitel 2 werden Grundlagen der Satellitenkommunikation vorgestellt und der in der Arbeit verwendete Kommunikationsstandard, welcher zum Teil die möglichen Ausgaben des Algorithmus und damit auch Teile der potentiellen Transmitterkonfigurationen festlegt. Danach schließt eine Einführung in Linkbudgets und in das Signal-Rausch-Verhältnis an, einem der wichtigsten Input-Parameter für den Algorithmus. Anschließend folgt eine Einführung in Reinforcement Learning und eine Vorstellung des Neural Network (NN)-based Reinforcement Learning (RLNN2) Algorithmus, welcher die Grundlage für die Weiterentwicklungen in der Arbeit ist. In Kapitel 3 erfolgt der Vergleich des RLNN2 Algorithmus mit aktuellen Alternativen und in Kapitel 4 die Vorstellung der verwendeten Software und Simulationsumgebung. Im Anschluß werden die Veränderungen am RLNN2 Algorithmus dargestellt und in Kapitel 6 die Ergebnisse der Veränderungen präsentiert.

2. Grundlagen

2.1. Einleitung

In diesem Kapitel werden die relevanten Grundlagen zur Satelliten Kommunikation vorgestellt. Zunächst erfolgt die Beschreibung eines weit verbreiteten Standards der Funkübertragungen mit Satelliten sowie eine Einführung zu Software Defined Radios und der technischen Grundlagen einer Funkübertragung. Es folgt eine Einführung in Reinforcement Learning und Function Approximation mit Neural Networks. Als Abschluss des Kapitels wird der RLNN2 Algorithmus aus [FPW+18] vorgestellt. Dieser Algorithmus bildet die Grundlage für die Reinforcement Learning Software dieser Arbeit.

2.2. Space Kommunikation

2.2.1. CCSDS

In den letzten Jahrzehnten wurden mehrere Datenübertragungsstandards für Satelliten entwickelt. Zwei Beispiele sind “The Consultative Committee for Space Data Systems” (CCSDS), welcher häufig für die Übertragung von Zustandsinformationen und Kommandos bei Satelliten genutzt wird und “Digital Video Broadcasting - Satellite” (DVB-S), der in Fernsehsatelliten zum Senden der Inhalte zum Einsatz kommt. Der Vorteil der Nutzung eines solchen Standards ist die große Zuverlässigkeit und Vorhersehbarkeit dieser Standards, sofern sie korrekt implementiert wurden.

Die Entscheidung, CCSDS als Standard in dieser Arbeit zu nutzen, wurde aus verschiedenen Gründen getroffen. Zum einen ist CCSDS ein weit verbreiteter Standard. Ein adaptiver Algorithmus, der einen derart gängigen Standard enthält, hat das Potential, gewinnbringend für viele Missionen zu sein. Des weiteren kommuniziert der vom Institut für Raumfahrtsysteme (IRS) entwickelte und betriebene Referenzsatellit Flying Laptop (FLP), der für die Simulation genutzt wird, auch nach dem CCSDS Standard.

Die in dieser Arbeit genutzten CCSDS Standards sind zum einen Radio Frequency and Modulation Systems [AA20], der die Parameter des physikalischen Layers des OSI-Modells definiert, sowie Telemetry (TM) Synchronization and Channel Coding [AA17b], der einen Teil des Data Link Layers im OSI-Modell festlegt. Der Standard Telecommand (TC) Synchronization and Channel Coding [AA17a] ist der hier anzuwendende Standard für den Uplink.

Die für den Anwendungsfall in dieser Arbeit wichtigen Parameter aus dem Standard Radio Frequency and Modulation Systems sind die vorgeschlagenen Modulationen. Der Standard empfiehlt nur so genannte Phase-shift keying (PSK) Modulationsarten, bei denen mit Hilfe von Phasenshifts

die Daten auf das Signal moduliert werden. In Abhängigkeit der Anzahl übertragenen Bits pro Phasenshifts spricht man von BPSK bei einem Bit und QPSK bei zwei Bit und 8PSK bei drei Bits [AA20].

Aus dem Standard TM Synchronization and Channel Coding sind zwei Bereiche für die Arbeit relevant, zum einen Forward-Error-Correction (FEC) Codes und zum anderen Synchronization. Der Standard bietet vier verschiedene Forward-Error-Correction Codes an: Convolutional Coding, Reed-Solomon Coding, Turbo Coding und Low-Density Parity-Check (LDPC) Coding. Sowohl Turbo Coding als auch LDPC Coding sind sehr komplex, wodurch sie ein höheres Korrekturpotential besitzen, aber gleichzeitig auch mehr Rechenleistung und eine deutlich komplexere Implementierung erfordern [AA17b]. Zusätzlich sind die beiden Coding-Verfahren für hohe Datenraten ausgelegt, wodurch sie nicht in jeder Situation der Simulation sinnvoll genutzt werden können. Aus den genannten Gründen wäre es nur sinnvoll, diese Coding-Verfahren in wenigen Situationen der Simulation zu nutzen. Um die Komplexität zu senken, werden diese Verfahren daher in dieser Arbeit nicht angewandt. Das angewandte Coding Verfahren ist das Convolutional Coding, das mit dem Reed-Solomon Coding konkateniert werden kann. Die in der Simulation versendeten Daten sind mit Reed-Solomon als äußerem Code und Convolutional Code als innerem Code codiert [AA17b].

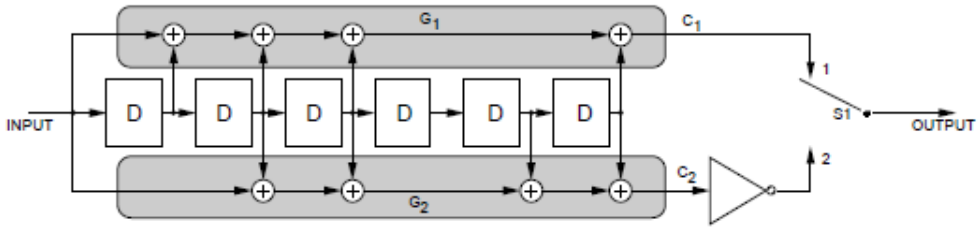

Abbildung 2.1.: Implementierung Convolutional Encoder mit der Coderate $\frac{1}{2}$ [AA17b]

Abbildung 2.1 stellt die Funktionsweise des Convolutional Encoders dar. Der Encoder besteht aus zwei Polynomen G1 171_8 und G2 133_8, die Daten kommen in den Input und laufen durch eine Flip-Flop-Kette, im Bild mit D bezeichnet. Die Ausgabe des G2 Polynom wird invertiert. Bei einer Coderate von $\frac{1}{2}$ wird für jedes Bit am Input zwei Bit ausgegeben, das erste Bit aus dem oberen Pfad und das zweite aus dem unteren Pfad. Es sind noch weitere Coderates laut CCSDS Standard möglich: $\frac{2}{3}$, $\frac{3}{4}$, $\frac{5}{6}$ und $\frac{7}{8}$ [AA17b]. Der Einsatz der höheren Coderaten ist bei einer entsprechend guten Signalqualität sinnvoll, da weniger Korrekturen bei der Decodierung erforderlich sind.

Der Reed-Solomon Code ist ein FEC mit einer guten Korrektur bei so genannten Burst-Fehlern, welche eine kurzfristige starke Häufung von Fehlern in der Übertragung darstellt. Dafür werden für jedes zu sendende Codeword Check-Symbole berechnet, welche am Ende eines übertragenen Codeblocks angehängt werden. Zusätzlich wird ein sogenanntes Interleaving benutzt, das in Abbildung 2.2 zu sehen ist. Dabei werden immer acht Bits des zu codierenden Inputs an einen Encoder weitergeleitet, dann erfolgt das synchrone Umschalten der Schalter S1 und S2 auf den nächsten Encoder. Dadurch erfolgt die Datenübertragung einer Bytesequenz nicht sequenziell, sondern wird aufgeteilt, sodass bei Burst-Fehlern genügend Daten erhalten bleiben, um die ursprünglichen Daten zu rekonstruieren.

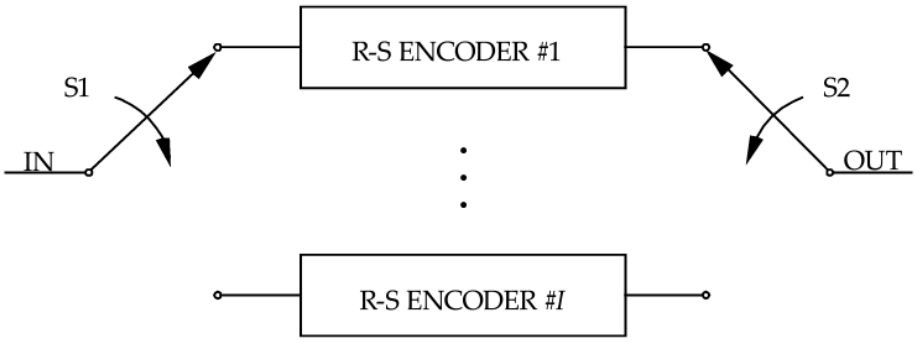

Abbildung 2.2.: Implementierung Interleaving Reed-Solomon Encoder [AA17b]

Die Synchronisation der Frames wird gemäß dem CCSDS Standard mit Attached Sync Markern (ASM) umgesetzt. Für die Wahl der Kodierung mit einem inneren Convolutional Code und einem äußeren Reed-Solomon Code wird im Standard bestimmt, dass die ASM nur in der Code Symbol Domain des Reed-Solomon Codes angefügt werden können. Somit muss die Frame Synchronisation nach dem Decodieren des Convolutional Codes erfolgen [AA17b].

2.2.2. Software Defined Radios

Ein Software Defined Radio (SDR) ist ein Transceiver, der Teile der Signalverarbeitung mit Hilfe von Software an Stelle von Analogkomponenten umsetzt. Wie in Abbildung 2.3 zu sehen, gibt es fünf verschiedene Bereiche beim Empfangen und Senden mit einem Transceiver. Die verschiedenen Bereiche können zum Teil in Software oder in Hardware realisiert werden. Bei einem klassischem Transceiver wird nur das Verarbeiten der Daten und das Framing in Software umgesetzt. Danach erfolgt die Umwandlung des Signals durch einen Digital-Analog-Wandler (DAC) beim Senden und durch einen Analog-Digital-Wandler (ADC) beim Empfangen. Sowohl Kodierung als auch Modulation werden anschließend in Hardware umgesetzt, wodurch nur eine oder mehrere vorher festgelegte Konfigurationen möglich sind. Vor der Antenne erfolgt das Filtern und Verstärken des kodierten und modulierten Signals, dargestellt als “RF” Block in Abbildung 2.3.

Bei einem Software-Defined-Radio wird die ganze Signalverarbeitung von der Datenverarbeitung bis zur Modulation mit Software realisiert und erst nachdem das Signal digital moduliert wurde, wird es mit einem DAC konvertiert und dann ähnlich wie bei einem klassischen Transceiver gefiltert und verstärkt. Sobald eine Machine Learning Software oder ein adaptiver Algorithmus an die verschiedenen Blöcke angeschlossen ist und mit Hilfe von Messungen das Signal verändert oder Blöcke rekonfiguriert, wird dies als Cognitive Radio bezeichnet. Die Vorteile von Software Defined Radios sind zm einen die Nähe des DAC und ADC zur Antenne, wodurch es zu weniger Verlusten und Störungen zwischen Empfang und Umwandlung des Signals kommt, sowie zum anderen ihre Einsatzmöglichkeit bei vielfältigen Anwendungen auf Grund ihrer hohen Flexibilität und der Möglichkeit, Funkparameter unkompliziert zu ändern, zu messen und darzustellen. So besteht die Option, zahlreiche Werte beim Empfang zu messen und die Konfiguration des Transmitters und Receivers im Betrieb anzupassen.

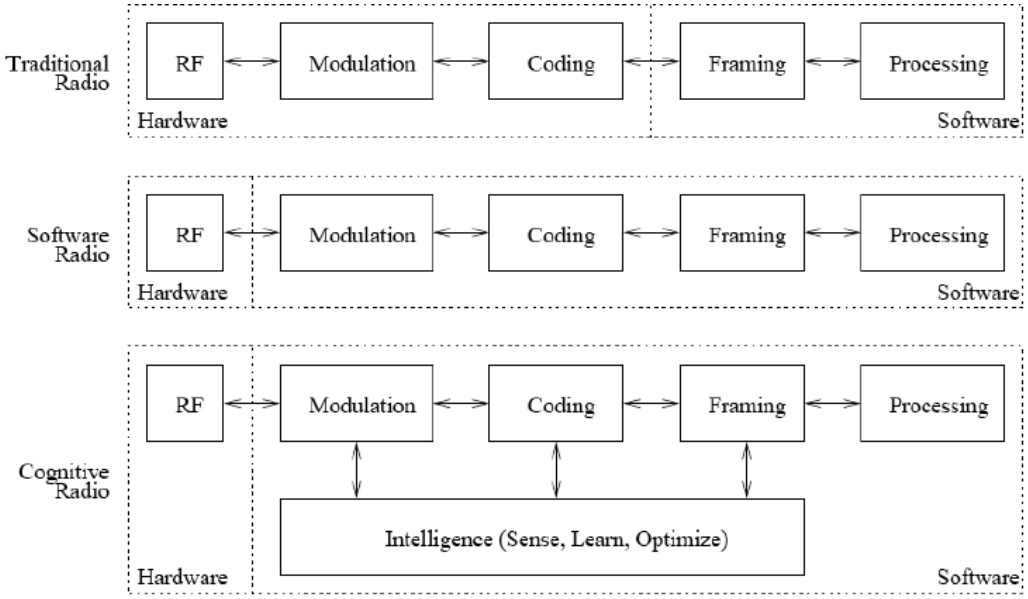

Abbildung 2.3.: Vergleich klassischer Transceiver und Software Defined Radio [KRS11]

2.2.3. Linkbudget

Die Übertragung zwischen einem Transmitter und einem Receiver ist von zahlreichen Faktoren abhängig. Die Kommunikation in dieser Arbeit ist immer so ausgelegt, dass eine Sichtverbindung zwischen Transmitter und Receiver nötig ist, um einen Link zu schließen. Das Schließen eines Links beschreibt einen erfolgreichen Datenaustausch. Um vorherzubestimmen, ob ein Link zwischen einer ausgewählten Konfiguration von Transmitter und Receiver für vorher festgelegte Übertragungsbedingungen geschlossen werden kann, wird ein Linkbudget berechnet. Die Berechnung dieser Linkbudgets erfolgt in der Raumfahrt für einen Best-Case und einen Worst-Case. Mit den Ergebnissen des Best-Cases bestimmt man, ob das Signal innerhalb der zulässigen Leistungsgrenzen ist, um sowohl eine Beschädigung des Receivers durch ein zu starkes Signal zu vermeiden als auch die rechtlichen Grenzen zur Signalstärke [ITU95] einzuhalten. Mit Hilfe des Worst-Case Linkbudgets wird sichergestellt, dass selbst unter den ungünstigsten Bedingungen ein Link zustande kommen kann.

$$C = P_t + G_t - L + G \tag{2.1}$$

Zur Bestimmung der an einem Receiver empfangenen Energie C wird die Formel aus Formel 2.1 genutzt. Die Formel beschreibt Gewinne, also Komponenten, die das Signal verstärken und Verluste, die das Signal abschwächen. Die verstärkenden Komponenten sind die Sendeenergie des Transmitters P_t, der Antennengewinn des Transmitters G_t sowie der Antennengewinn des Receivers G. Die Verluste, die sich aus mehreren Faktoren zusammensetzen, werden mit L beschrieben [E L00].

$$L_0 = 20 \log\left(\frac{4\pi d}{\lambda}\right) \tag{2.2}$$

Der größte Verlust einer Funkübertragung ist die Freiraumdämpfung. Diese beschreibt die Abnahme der Leistung über die steigende Distanz der Übertragung. Formel 2.2 zeigt die Formel zur Berechnung der Freiraumdämpfung L_0. Die durch den Freiraumdämpfung eintretende Dämpfung des Signals hängt von den Werten der Entfernung d und der Wellenlänge λ ab [E L00].

Bei einer Funkübertragung kommt es außerdem zu weiteren Verlusten, insbesondere bei der Kommunikation zwischen einer Bodenstation und einem Satelliten. Sowohl in der Tropo- als auch in der Ionosphäre treten Verluste auf Grund atmosphärischer Gase und der Dämpfung durch Regen, Wolken und Nebel auf. Je höher die Übertragungsfrequenz des Signals ist, desto größer ist die Auswirkung der Regendämpfung, die ab einer Signalfrequenz von 10 GHz neben der Freiraumdämpfung die stärkste Abschwächung des Signals verursacht [E L00].

Zur Verarbeitung eines empfangenen Signals sind verschiedene Faktoren wichtig, wie die Signalstärke sowie das Verhältnis von Signalstärke und Rauschen. Dieses Verhältnis wird Signal-to-Noise Ratio (SNR) genannt. Das Rauschen im Receiver ist unter anderem das thermische Rauschen, welches beim Empfang das eingehende Signal überlagert. Die beiden Hauptfaktoren für das Rauschen im Empfängersystem sind das Rauschen, welches von der Antenne aufgefangen wird und das Rauschen im ersten Verstärker des Receivers, dem Low-Noise Amplifier (LNA). Zur Bestimmung der effektiven Rauschtemperatur erfolgt die Addition der Rauschtemperatur von Antenne und Receiver. Thermisches Rauschen wird durch die Thermal Noisepower Spectral Density $N_0 = kT$ charakterisiert und ist konstant über die Frequenz. k ist die Boltzmann Konstante und T beschreibt die effektive Gesamtrauschtemperatur [E L00].

$$\frac{C}{N_0} = P_t + G_t - L + G - k - T \tag{2.3}$$

Formel 2.3 veranschaulicht die Formel für das im Buch [E L00] so genannte Linkbudget. Diese Formel wurde durch die Anpassung der Formel für die empfangene Energie C aus Formel 2.1 um die Boltzmann-Konstante k und die effektive Gesamtrauschtemperatur T erstellt [E L00]. Die SNR lässt sich mit $\frac{C}{N_0}$ und B_r, der Noise Equivalent Bandwidth, mit Formel 2.4 berechnen [E L00].

$$\frac{C}{N} = \frac{C}{N_0} * \frac{1}{B_r} \tag{2.4}$$

Mithilfe von $\frac{C}{N_0}$ lässt sich ein weiteres Verhältnis berechnen, nämlich die Symbol Energy-to-Noise Power Density $\frac{E_s}{N_0}$ [E L00].

$$\frac{E_s}{N_0} = \frac{C}{N_0} - R_s \tag{2.5}$$

Formel 2.5 zeigt die Formel für die Berechnung von $\frac{E_s}{N_0}$ aus $\frac{C}{N_0}$ mit Hilfe der Symbolrate R_s [E L00].

Der Wert $\frac{E_s}{N_0}$ lässt sich zur Bit Energy-to-Noise Power Density $\frac{E_b}{N_0}$ mit der Formel 2.6 berechnen, wobei k die Anzahl an Bit pro Symbol für die gewählte Modulation angibt [EA16].

$$\frac{E_b}{N_0} = \frac{E_s}{N_0} - 10 * \log_{10} k \tag{2.6}$$

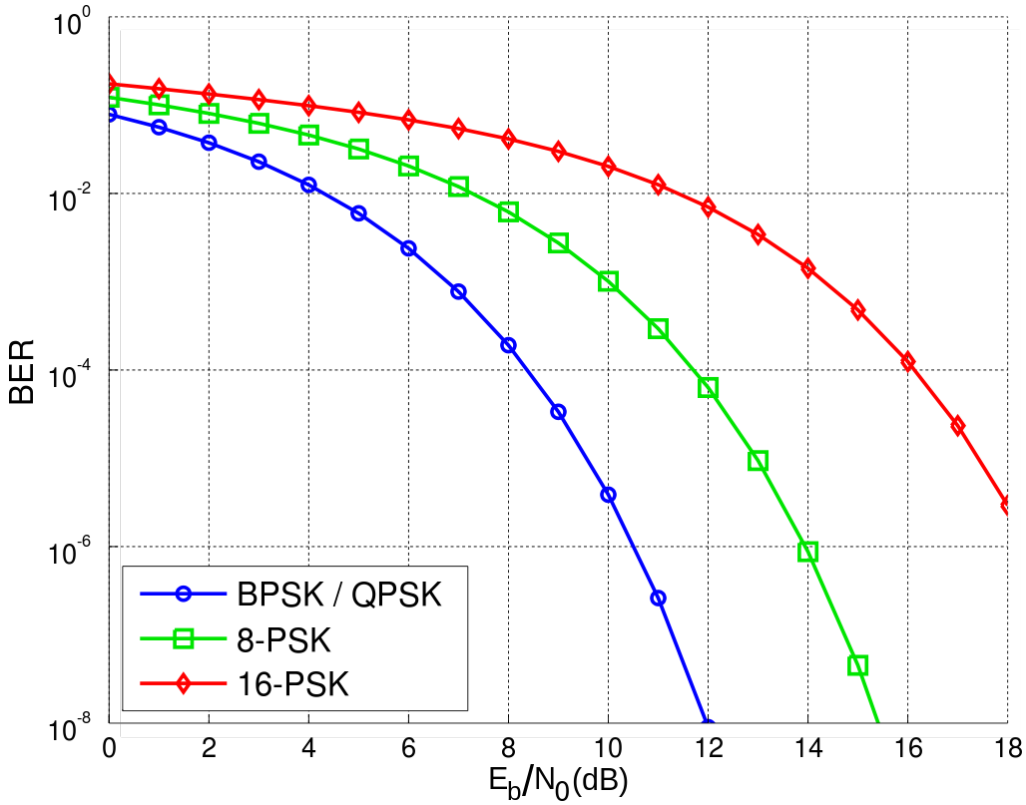

Abbildung 2.4.: BER zu $\frac{E_b}{N_0}$ für verschiedene PSK Modulationen [Spl07]

Dieser Wert lässt sich zur Bestimmung einer oberen Grenze der Bit-Error-Rate (BER) nutzen für einen festgelegten Wert $\frac{E_b}{N_0}$ und einer definierten Modulation. Aus dem Graphen in Abbildung 2.4 lässt sich beispielsweise erkennen, dass bei einer maximalen BER von 10^{-6} und einer 8-PSK Modulation eine Bit Energy-to-Noise Power Density von 14 dB nötig ist.

Da der Wert der SNR alle Gewinne einer Übertragung sowie sämtliche Verluste auf der Übertragungsstrecke und beim Empfang des Signals beinhaltet, spiegelt der beim Empfang gemessene Wert alle Veränderungen wider. Somit ist der Wert der SNR nicht nur entscheidend bei der Auslegung eines Kommunikationssystems, sondern er lässt sich auch im Betrieb des Kommunikationssystems nutzen, um Störungen oder Veränderungen während der Übertragung zu entdecken.

2.3. Maschine Learning

2.3.1. Reinforcement Learning

In [KLM96] wird das Reinforcment Learning Model wie folgt beschrieben, siehe Abbildung 2.5: Bei jedem Schritt der Interaktion erhält der Agent einen Input i und einen Indikator für den aktuellen Zustand der Umgebung s. Anhand dieser Inputs wählt Agent dann eine Aktion a. Diese Aktion

Reinforcement Learning	Problem
Agent	Adaptiver Algorithmus
Umgebung	Übertragungssimulation
Aktion	Wahl neuer Transmitterkonfiguration
Reward	Rewardfunktion
Input	SNR

Tabelle 2.1.: Vergleich von Reinforcement Learning und dem zu untersuchenden Problem in dieser Arbeit

ändert den Zustand der Umgebung und der Wert dieses Zustandsübergangs wird dem Agenten durch ein skalares Reinforcement Signal r kommuniziert. Das Verhalten des Agents B soll Aktionen auswählen, um langfristig die Summe der Werte des Reinforcement Signals zu vergrößern.

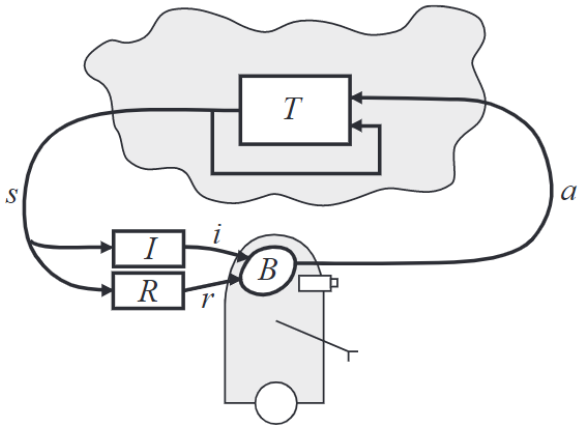

Abbildung 2.5.: Übersicht Standard Reinforcement Learning [KLM96]

Das in der Arbeit untersuchte Problem setzt sich aus mehreren Teilen zusammen. Zum einen gibt es eine Umgebung, welche das Senden, den Übertragungsweg und das Empfangen enthält. Diese Umgebung bietet den Wert der SNR, der alle Veränderungen widerspiegelt und welcher in Kombination mit der aktuellen Transmitterkonfiguration als Zustand der Umgebung angesehen werden kann. Zur Lösung des Problems wird eine Funktion gesucht, die anhand des Zustands eine Aktion bestimmt. Diese Aktion ist eine neue Transmitterkonfiguration, welche im Idealfall besser zum aktuellen Zustand der Umgebung passt. Die Funktion oder das Verhalten des Agenten soll regelmäßig sowohl neue Aktionen berechnen, als auch auf Veränderungen in der Umgebung reagieren. Zur Bestimmung der Performanz der Aktionen oder zur Abschätzung der Performanz zukünftiger Aktionen wird eine Rewardfunktion ausgewertet. Die Aktion im folgenden Rechenschritt der Funktion hängt von dem Ergebnis des aktuellen Rechenschrittes ab, da in Abhängigkeit von der aktuellen Transmitterkonfiguration, der aktuellen Aktion und dem Zustand der Umgebung sowohl die Belohnung als auch die folgende Aktion berechnet wird.

Das in [KLM96] vorgestellte Modell enthält Komponenten, die als Teile des Problems im zweiten Absatz definiert wurden. Ein Vergleich dazu ist in Tabelle 2.1 ersichtlich. Der Input i aus [KLM96] entspricht dem Input des Algorithmus aus SNR und Transmitterkonfiguration, der aktuelle Zustand der Umgebung entspricht der SNR. Das skalare Reinforcement Signal r wird durch das Ergebnis der Rewardfunktion abgebildet und das Verhalten des Agenten B soll die Funktion zur Bestimmung der nächsten Transmitterkonfiguration darstellen. Da das Problem exakt auf die Definition von Reinforcement Learning passt, wurde in dieser Arbeit Reinforcement Learning als Grundtechnik zur Lösung des Problems gewählt. Ein weiterer Grund für die Wahl eines Reinforcement Learning Algorithmus ist, dass das Sammeln eines Trainingsdatensets für eine Mehrzahl an möglichen Zuständen und idealen Aktionen kaum möglich ist. Eine ideale Aktion ist oftmals nicht nur durch eine einzige mögliche Transmitterkonfiguration definiert. Der Zustand des Kommunikationskanals kann sehr viele unterschiedlich Zustände annehmen, welche sich auch sehr dynamisch ändern können. Daher lassen sich kaum sinnvolle Trainingsdaten mit einer guten Abdeckung sammeln, zudem wären diese immer unvollständig. Aus diesen Gründen ist es nicht möglich, Supervised Learning Algorithmen zu benutzen. Auch Unsupervised Learning Algorithmen können hier nicht angewendet werden, da die Interaktion des Agenten mit der dynamischen Umgebung für das Problem elementar sind.

Q-Learning ist ein bekannter Algorithmus aus dem Bereich des Reinforcement Learnings. Die erlernte action-value Funktion approximiert dabei direkt die optimale action-value Funktion. Eine Übersicht über Q-Learning bietet Abbildung 2.6, wobei S die Zustände, A die Aktionen und R den Reward bestimmt.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Abbildung 2.6.: Übersicht Q-Learning [Sut18]

Da der Q-Learning Algorithmus sowohl Exploration als auch Exploitation betreiben muss, ist vor jedem Schritt die Entscheidung zu treffen, welche der Optionen gewählt werden soll. Die Exploration beschreibt das Austesten beliebiger Aktionen, um neue Beobachtungen machen zu können und die Exploitation beschreibt das bestimmen der besten zu erwartenden Aktion im aktuellen Zustand. Zur Entscheidung wird kann unter anderem der ϵ-greedy Algorithmus benutzt werden. Vor jeder Aktion entscheidet dieser Algorithmus mit der Wahrscheinlichkeit von $1 - \epsilon$ für die Exploitation. Bei der Exploitation versucht der Algorithmus, im aktuellen Zeitschritt der Berechnung den Reward für jede mögliche Aktion im aktuellen Zustand vorherzuberechnen und wählt die Aktion mit dem höchsten zu erwartenden Reward. Mit einer Wahrscheinlichkeit von ϵ wird Exploration gewählt, dies entspricht dem Auswählen einer zufälligen Aktion [MKS+13].

Q-Learning in seiner ursprünglichen Form basiert auf der Repräsentation der Action-Value Funktion in einer Tabelle. Diese Tabelle besteht aus allen möglichen Paaren an Zuständen und möglichen Aktionen in diesem Zustand. Das Speichern der Q-Funktion in einer Tabelle kann nur erfolgen, wenn die Anzahl der Zustände und der Aktionen endlich und nicht zu groß sind. Da bei dem Anwendungsfall in der Arbeit ein unendlich großer Raum an Zuständen vorhanden ist, kann die Action-Value Funktion nicht in einer Tabelle abgebildet werden. Der unendliche Raum an Zuständen erklärt sich durch die Möglichkeit jedes theoretisch möglichen SNR Wertes bei jeder möglichen Konfiguration des Transmitters. Selbst eine Begrenzung der Werte der SNR und die Definition einer festen minimalen Schrittgröße zur Erlangung eines endlichen Zustandsraumes führt zu einem extrem großen Zustandsraum. In der Konsequenz würde dies zum einen zu enormen Speicherplatzanforderungen führen und zum anderen ist nicht zu erwarten, dass eine optimale Policy gefunden wird [Sut18]. Zur Lösung dieses Problems gibt es Ansätze, die Action-Value Funktion zu approximieren.

Eine Möglichkeit der Funktionsapproximation ist eine Weiterentwicklung des Q-Learning Algorithmus, welche Deep Q-learning mit Experience Replay genannt wird [MKS+13]. Diese Weiterentwicklung umfasst zwei wichtige Änderungen am ursprünglichen Q-Learning Algorithmus. Die erste Modifikation ist das Nutzen einer Funktionsapproximation durch ein Neural Network. Durch die Verwendung eines NN zur Abbildung der Funktionen entfällt das Problem des unendlichen Zustandsraumes. Anstatt einer Tabelle mit Zustands- und Aktionswerten wird ein Deep Neuronal Network genutzt, diese Modifikation wird auch Deep Q-learning genannt. Dieses Netzwerk wird mit Sampels von beobachteten Erfahrungen trainiert. Ein solches Sampel enthält die Werte s_t, a_t, r_t, s_{t+1}, welche beim Interagieren des Agenten mit der Umwelt gespeichert werden und nachfolgend zufällig beim Training der NN genutzt werden, was Experience Replay genannt wird [MKS+13].

Der Ansatz des Deep Q-Networks wie in [MKS+13] vorgestellt nutzt Experience Replay. Dieser Algorithmus verwendet im ersten Schritt eine ϵ-greedy Policy, um zu entscheiden, ob eine zufällige Aktion ausgeführt wird. Sofern keine zufällige Aktion ausgeführt wird, wird mit Hilfe des NNs, das die Q-Funktion abbildet, die beste vorhergesagte Aktion bestimmt und ausgeführt. Der aktuelle Reward und der nächste Zustand werden beobachtet und das Tupel (s_t, a_t, r_t, s_{t+1}) wird im so genannten Replay Memory gespeichert. Danach wird ein Tupel oder eine Minibatch an Tupeln zufällig aus dem Replay Memory bestimmt, um mit diesen Daten das NN und damit die Q Funktion zu trainieren [MKS+13].

2.3.2. RLNN2

Der in [FPW+18] vorgestellte Algorithmus RLNN2 benutzt eine weiterentwickelte Form des Deep Q-Learnings mit Experience Replay. Der Hauptunterschied im Konzept des Algorithmus im Vergleich zum Deep Q-Learning mit Experience Replay aus [MKS+13] ist die virtuelle Exploration im Gegensatz zur zufälligen Exploration. Bei der zufälligen Exploration wählt der Algorithmus aus allen in dem Zustand möglichen Aktionen zufällig eine Aktion aus. Bei der virtuellen Exploration teilt der Algorithmus jede mögliche Aktion in gut oder schlecht ein und entscheidet dann, ob eine gute oder schlechte Aktion ausgewählt wird. Entsprechend der Entscheidung wählt der Algorithmus zufällig aus der Menge der entsprechenden Aktionen eine aus. Der grundlegende Aufbau des Algorithmus wird in Abbildung 2.7 veranschaulicht. Der Agent besteht aus drei Komponenten, zum einen den RLNN2 Algorithmus, der entscheidet ob eine Exploration oder eine Exploitation stattfindet und welcher Input für das NN gewählt wird. Zusätzlich gibt es zwei Blöcke aus verschiedenen NN,

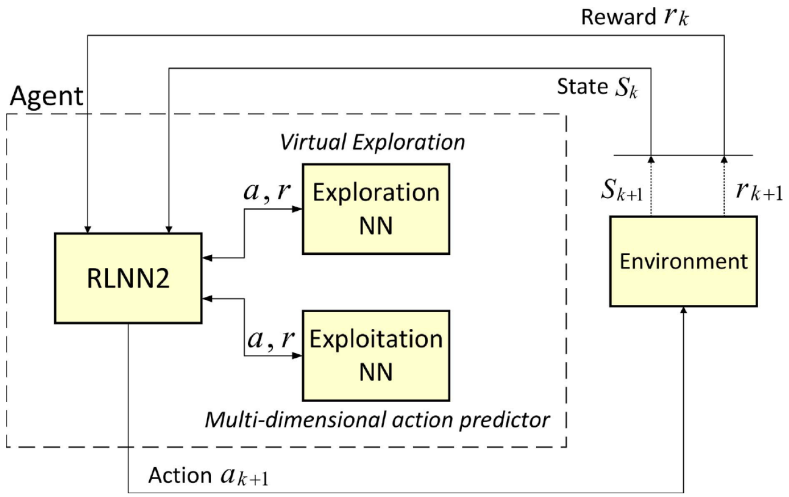

Abbildung 2.7.: Übersicht RLNN2 Algorithmus[FPW+18]

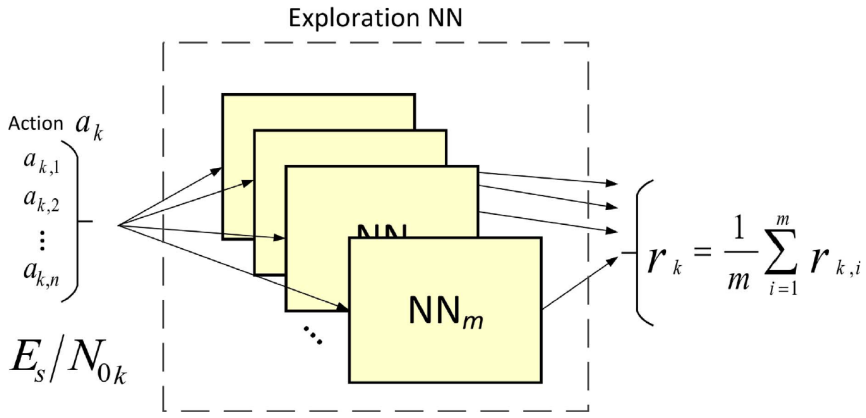

Abbildung 2.8.: Übersicht Exploration NN [FPW+18]

das Explorations NN und das Exploitation NN. Die möglichen Aktionen, die der Agent ausgeben kann, sind eine Kombination an Werten, um den Transmitter neu zu konfigurieren. Die benutzen Parameter sind Modulation, Code-Rate, Roll-off Faktor, Symbolrate und zusätzliche Sendeleistung. Der zu maximierende Reward des Algorithmus lässt sich aus einer gewichteten Formel berechnen: $f_{obs}(x) = w_1 f_{Thrp}(x) + w_2 f_{BER}(x) + w_3 f_{BW}(x) + w_4 f_{Spc\_eff}(x) + w_5 f_{Pwr\_eff}(x) + w_6 f_{Pwr\_com}(x)$. Die Werte für Bit-Error-Rate (BER), Durchsatz (Thrp), Bandbreite (BW), spektrale Effizienz (Spc_eff), zusätzliche Sendeleistung (Pwr_con) und Energieeffizienz (Pwr_eff) können aus der aktuellen Transmitterkonfiguration und dem Verhältnis empfangener Symbol Energy-to-Noise Power Density $\frac{E_s}{N_0}$ berechnet werden. Für die Gewichte $w_1, ..., w_6$ besteht die Möglichkeit der Definition unterschiedlicher Werte, wobei die Summe aller Werte stets 1 ergeben muss. Je nach dem, wie stark welche Parameter in der aktuellen Situation optimiert werden sollen, können die Gewichte entsprechend angepasst werden. Mit verschiedenen Gewichtungsprofilen lässt sich der Algorithmus auf unterschiedliche Situationen eines Satelliten anwenden. Der Zustand der Umgebung ist die Kombination aus der gemessenen Symbol Energy-to-Noise Power Density und den akuellen Konfigurationswerten des Transmitters [FPW+18].

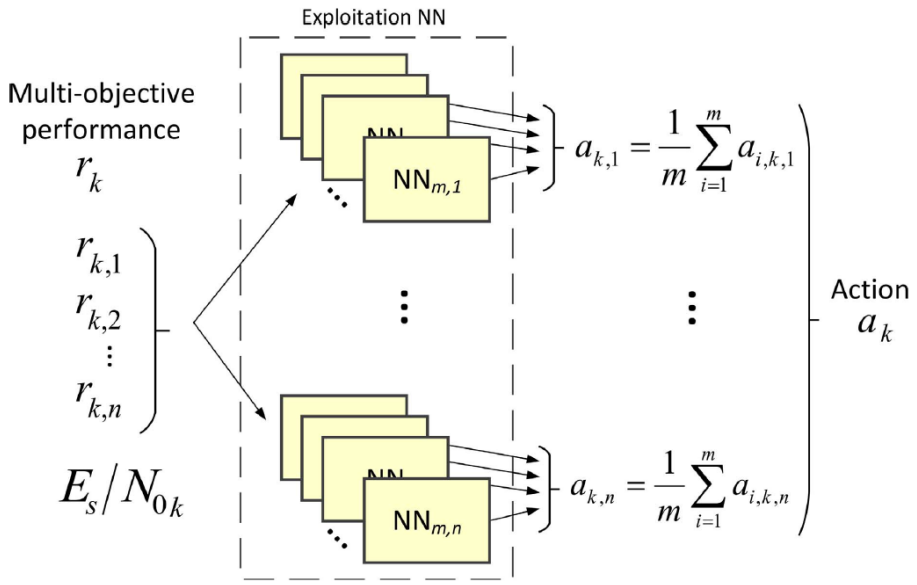

Abbildung 2.9.: Übersicht Exploitation NN [FPW+18]

Der Hauptunterschied zum Deep Q-Learning mit Experience Replay ist die Exploration. Da ein Überflug eines Satelliten nur relativ kurz ist, versucht man die vorhandene Zeit möglichst effektiv zu nutzen. Eine zufällige Exploration kann zu einer sehr schlechten Wahl der Transmitterkonfiguration führen. Als Konsequenz können solange wenige oder im schlechtesten Fall gar keine Daten übertragen werden, bis der Algorithmus eine andere und bessere Konfiguration gewählt hat. Bei einer zufälligen Exploration besteht außerdem die Möglichkeit der Wahl einer Aktion, die bereits vorher in einem sehr ähnlichen Zustand ausprobiert wurde. Um diesen Problemen entgegenzuwirken, wird im RLNN2 Algorithmus ein NN benutzt, welches eine so genannte virtuelle Exploration ausführt. In Abbildung 2.8 wird die Übersicht über das Exploration NN dargestellt. Die Eingabe für die Exploration NNs ist zum einen die aktuelle Symbol Energy-to-Noise Power Density und zum anderen wird jede potentielle Aktion als Eingabe verwendet. Die Ausgabe der NN ist der geschätzte Reward jeder Aktion im Bezug zum aktuellen Zustand der Umgebung. Da in [FPW+18] 20 parallele identisch trainierte NNs benutzt werden, wird das Ergebnis der NNs gemittelt [FPW+18].

Die in Abbildung 2.9 dargestellte Übersicht ist die Struktur des Exploitation NN. Als Input für diese NN wird auch die Symbol Energy-to-Noise Power Density benutzt. Zusätzlich erhalten die NNs den Reward. Für jeden möglichen Paramter der Aktion gibt es 20 NN, welche die beiden Eingaben erhalten und daraus den besten Wert für diesen Parameter bestimmen. Somit ist die kombinierte und gemittelte Ausgabe der NN eine vollständige Aktion [FPW+18].

Nach der Einführung in die beiden NNs folgt jetzt die Entscheidungslogik, welche in Abbildung 2.7 als RLNN2 Block veranschaulicht wird. Abbildung 2.10 zeigt die Logik des Algorithmus. Nach der Initialisierung des Algorithmus erfolgt im ersten Schritt die Füllung des Trainingsdatenbuffers, indem zufällige Aktionen ausprobiert werden. Sobald der Trainingsdatenbuffer gefüllt ist, werden die NNs das erste Mal trainiert. Nach dem Training ist die Initialisierung des Algorithmus abgeschlossen und er wechselt in den regulären Betrieb. Wenn der Agent eine neue Aktion auswählen soll, entscheidet er sich anhand einer Zufallszahl und der ϵ-Funktion, ob eine Exploration oder eine Exploitation ausgeführt werden soll, siehe Zeile 10 in Abbildung 2.10. Die ϵ-Funktion ist als $\epsilon = \frac{1}{k}$ definiert,

Require: NN1 and NN2 setup, and R_s, E_s, k, β, M, $\bar{c}$, NN_{bs}, NN_{dump}, $f(\varepsilon)$, tr_a, $\text{min}_{\text{good\%}}$

1: **U** $\leftarrow$ all combinations of $(\bar{R}_s, \bar{E}_s, \bar{k}, \bar{\beta}, \bar{M}, \bar{c})$
2: **loop**
3: **if** $(\bar{R}_s, \bar{E}_s, \bar{k}, \bar{\beta}, \bar{M}, \bar{c})$ has changed **then**
4: **U** $\leftarrow$ all combinations of $(\bar{R}_s, \bar{E}_s, \bar{k}, \bar{\beta}, \bar{M}, \bar{c})$
5: **end if**
6: **while** NN training data buffer not full **do**
7: 'Forced exploration': only explore
8: **end while**
9: z $\leftarrow$ uniform random number $[0, 1]$
10: **if** z $< f(\varepsilon) = \varepsilon_k$ **then** ▷ with prob. ε_k (Explore)
11: Predict actions using NN1
12: Classify actions into good or bad using $\text{min}_{\text{good\%}}$
13: u $\leftarrow$ uniform random number $[0, 1]$
14: **if** u$<$ tr_a **then** ▷ Action rejection
15: Randomly select one good action a
16: **else**
17: Randomly select one bad action a
18: **end if**
19: **else** ▷ with probability $1 - \varepsilon_k$ (Exploit)
20: Predict action to be exploited using NN2
21: Execute selected a
22: Measure and/or read RL states $\bar{s}$
23: Compute multi-objective fitness function $f_{\text{obs}}(x)$
24: Select next $\text{NN2}_{\text{input}}$ ▷ see Algorithm 2
25: Update NN training data buffer
26: Build NN1 and NN2 training datasets
27: **if** NN training data buffer is full **then**
28: Remove NN_{dump} entries from NN training dataset
29: **end if**
30: **end if**
31: **end loop**

Abbildung 2.10.: Übersicht Entscheidungslogik RLNN2[FPW+18]

wobei k bei jedem Durchlauf um den Wert 1 inkrementiert wird bis $\epsilon = 10^{-4}$ erreicht und k auf 1 zurückgesetzt wird. Nach der Rücksetzung dieses Wertes steigt die Wahrscheinlichkeit der Durchführung einer Exploration. Der Wert für den Reset wurde aus [FPW+18] übernommen. Im folgenden wird die virtuelle Exploration erklärt, welche im Gegensatz zu einer zufälligen Exploration nur sehr selten schlechte Aktionen auswählt und dadurch verhindert, dass der Algorithmus in der beschränkten Zeit eines Überflugs unpassende Transmitterkonfigurationen auswählt. Wenn der Algorithmus Exploration auswählt, wird das Explorations NN benutzt, um für den aktuellen Zustand und jede mögliche Kombination an Parametern einen Reward vorherzubestimmen. Sämtliche errechneten Aktionen werden im nächsten Schritt anhand des erwarteten Rewards sortiert, wobei die besten 10 % der Aktionen als gute Aktionen gelten und der Rest als schlechte. Folgend wird mit einer Zufallszahl bestimmt, ob eine gute oder eine schlechte Aktion ausgewählt wird, siehe Zeile 14 in Abbildung 2.10. Im Falle der Wahl einer guten Aktion wird aus dem Pool der guten Aktionen zufällig eine ausgewählt und ausgeführt. Das gleiche passiert für eine schlechte Aktion.

Abschließend wird der neue Zustand und der Reward für die Aktion gemessen, und zusammen mit dem vorherigen Zustand im Trainingsdatenbuffer gespeichert und dann eine neue Aktion ausgewählt.

Sollte der Algorithmus statt Exploration Exploitation wählen, beginnt dieser Schritt in Zeile 20 in Abbildung 2.10. Als erstes wird das Exploitation NN genutzt, um die nächste Aktion zu bestimmen, welche dann direkt ausgeführt wird. Nach der Ausführung wird der neue Zustand und der neue Reward berechnet und zusammen mit dem vorherigen Zustand im Trainingsdatenbuffer gespeichert. Danach wird der nächste Input für das Exploitation NN bestimmt, der detaillierte Algorithmus wird weiter unten beschrieben.

Jedes Mal, wenn der Trainingsdatenbuffer voll ist, der in [FPW+18] 200 Einträge ohne nähere Begründung umfasst, werden die beiden NN mit den Daten trainiert. Während des Trainings der neuen NN wird die nächste Aktion mit dem Exploitation Netzwerk bestimmt. Nach dem Training werden die ältesten 50 Einträge im Trainingsdatenbuffer gelöscht und die nächste Aktion wird bestimmt, siehe Zeile 10 in Abbildung 2.10.

Algorithmus 2.1 Input Algorithmus Exploitation NN

```
1:  if RewardObserved >= RewardObservedMax then
2:      Input = Zustand
3:  else
4:      if RewardObserved < ExploitScore then
5:          if (ExploitScore - RewardObserved) > forceExploreThreshold then
6:              Reset Trainingsdatenbuffer
7:              Goto ForcedExploration
8:          else if RewardObserved < 0.9*ExploitScore then
9:              Get Input from Trainingsdatenbuffer
10:         else if RewardObserved > 0.9*ExploitScore then
11:             ExploitScore = RewardObserved
12:         else
13:             Input = LastInput
14:         end if
15:     else
16:         ExploitScore = RewardObserved
17:         LastInput = Zustand
18:     end if
19: end if
```

Das Exploitation NN bekommt je nach Performanz andere Inputs. Wenn der aktuelle Reward-Wert größer oder gleich dem bisher maximal beobachteten Reward ist, erhält das Exploitation NN als Input die aktuell gemessenen Werte, siehe Zeile 1 in Algorithmus 1. Sofern der aktuelle Reward im Verhältnis zum maximalen Reward der Exploitation unter einen gewissen Wert fällt, wird der Trainingsdatenbuffer zurückgesetzt und der Algorithmus wird zurückgesetzt. Im nächsten Schritt findet dann eine erzwungene Exploration statt, siehe Zeile 5. Sobald der Wert des Rewards unter 90% des maximalen Reward der Exploitation fällt, werden für den nächsten Exploitation NN Input die Daten des Eintrags mit dem höchsten Reward aus dem Trainingdatenbuffer genommen, siehe Zeile 8. Liegt der beobachtete aktuelle Reward über 90% des maximalen Rewards der Exploitation, wird dieser Wert als neuer maximaler Reward der Exploitation eingesetzt, siehe Zeile 9. Sofern der

aktuelle Reward kleiner ist als der maximale Reward der Exploitation und keiner der bisherigen Fälle zutrifft, wird als nächster Input der vorherige Input genutzt. In den Fällen, in denen der aktuelle Reward größer ist als der maximale Reward der Exploitation, wird der neue Maximalwert auf den Wert des Rewards gesetzt und der Wert des letzten Inputs auf den momentanen Zustand aktualisiert.

3. Related Work

Der im vorherigen Kapitel vorgestellte RLNN2 Algorithmus wurde in [FPW+18] nur in einer Simulation getestet. Daher wurde in [HBF+18] der Algorithmus für die Verwendung in einem Testbed am NASA Glenn Research Center implementiert. Das Ziel war eine Erprobung des Algorithmus mit einer Funkverbindung zwischen dem Testbed am Boden und dem Space Communications and Navigation (SCaN) Testbed auf der Internationalen Raumstation (ISS). Das SCaN Testbed auf der ISS ist ein experimentelles Testbed, mit welchem Versuche im Vereich Cognitiv Communication durchgeführt werden können. Die Tests wurden mit einem S-Band SDR des SCaN Testbeds ausgeführt. In Abbildung 3.1 ist der komplette Aufbau des Testsetups zu sehen. Auf der Cognitive Engine Workstation läuft der RLNN2 Algorithmus. Ein Input für die Workstation und für den Algorithmus ist die Symbol Energie zu Rauschleistungsdichte $\frac{E_s}{N_0}$, welche von dem ViaSat Modem gemessen wird. Zusätzlich werden die empfangenen Daten, die vom Newtec Modem demoduliert und decodiert wurden, auf der Workstation gespeichert, um nach dem Test Analysen durchführen zu können. Beide Modems erhalten das gleiche Funksignal, das zuvor mit der Funkhardware

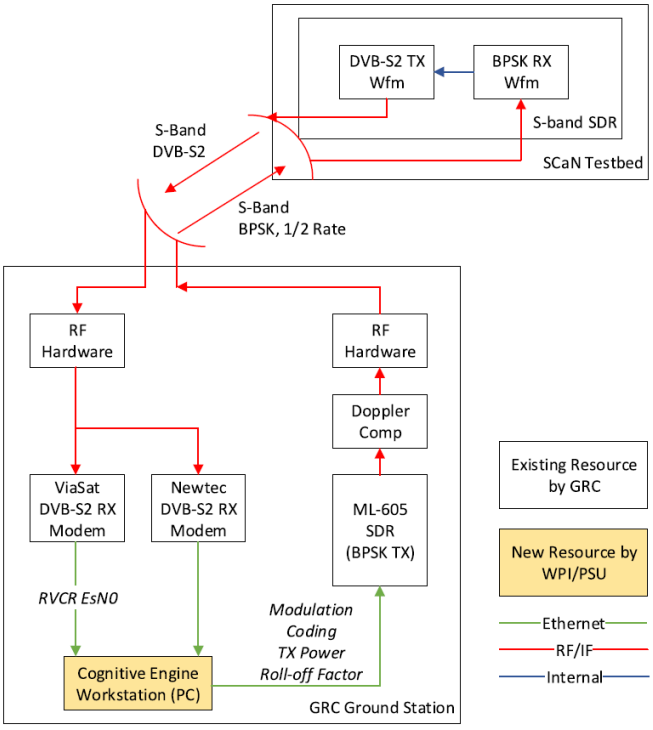

Abbildung 3.1.: Übersicht Testumgebung RLNN2 ISS Test [HBF+18]

empfangen wird. Die vom RLNN2 Algorithmus errechneten Transmitterkonfigurationen werden mit dem ML-605 SDR an das S-Band SDR auf dem SCaN Testbed geschickt. Mit diesen Daten erfolgt die Neukonfiguration des Transmitterteils des SDRs [HBF+18].

Die Ergebnisse des Tests zeigen, dass der Algorithmus bereits am Ende des ersten Überflugs eine sehr gute Performanz aufweist und sich an die gegebenen Funktionsparameter, welche den Gewichten in der Reward-Formel entsprechen, gut anpasst. So hat der Algorithmus am Ende einer Notfall-Mission, bei der ein Raumfahrzeug meist ein technisches Problem hat und eine zuverlässige Kommunikation nötig ist, die höchste Transmitterleistung und die niedrigste Kombination aus Modulation und Codierung gewählt, was zu den meisten Reserven während einer Übertragung führt. Die Performanz des Algorithmus steigt, wenn bereits ein Überflug durchgeführt wurde. Allerdings verbessert sich nur die Performanz am Anfang des Überflugs, bei dem ein nicht vortrainierter Algorithmus bis zu einer Minute benötigt, in denen er zufällige Aktionen ausführt und die NNs das erste Mal trainiert. Die Dauer eines Überfluges wird im Paper mit 450 Sekunden angegeben. Ein Problem der Implementierung ist, dass der Algorithmus bei besonders schlechten Übertragungsbedingungen, die in sehr kleinen Werten der Symbol Energy-to-Noise Power Density resultieren, keine gute Konfiguration findet, sondern den Buffer immer wieder nach einer passenden Konfiguration durchsucht [HBF+18].

Obwohl die National Aeronautics and Space Administration (NASA) den Bedarf an adaptiven Algorithmen postuliert [KM17], um zukünftige Missionen effizient durchführen zu können, finden sich wenig Veröffentlichungen zu Projekten oder Forschung in diesem Bereich. Es gibt einige Ansätze und Algorithmen, welche in den letzten Jahren entwickelt und veröffentlicht wurden. Die meisten dieser Algorithmen wurden nur unter Laborbedingungen entwickelt und getestet, wobei oftmals auf die Verwendung von Hardware verzichtet wurde und die Algorithmen nur mit simulierten oder aufgezeichneten Daten trainiert und ausgewertet wurden. Siehe [FMW14][DKK+15][JJD19].

In [KM17] erfolgt die Aufteilung des Bedarfes an adaptiven Optimierungs-Algorithmen in drei Bereiche. Zum einen werden Algorithmen benötigt, welche Interferenzen ausgleichen, indem das mit anderen Nutzern geteilte Spektrum effizienter genutzt wird. Hierfür ist beispielsweise die Verwendung ungenutzter Lücken im Frequenzspektrum möglich mit dem Ziel, eine größere Menge an Informationen zu übertragen [KM17]. Ein weiterer Bereich ist der Einsatz von Algorithmen, um die Kodierung und die Modulation dynamisch an die Übertragungsbedingungen anzupassen. Ein hierfür bereits angewandter Ansatz ist der Adaptive Kodierungs- und Modulationsalgorithmus (ACM), welcher mit dem Standard DVB-S2 genutzt werden kann. Dieser Algorithmus basiert auf einer Lookup Tabelle und wählt anhand der Übertragungsbedingungen eine neue Konfiguration aus [KM17]. Zur Bestimmung der nächsten Kombination aus Modulation und Kodierung wird ein Rückkanal vom Empfänger zum Sender benötigt, über den die Übermittlung der Empfangsbedingungen an den Sender vom Empfänger erfolgt [GZK09]. Der dritte Bereich ist intelligente Netzwerke und Routing, wozu ein kognitives Kommunikationssystem auch höhere Layer des OSI-Modells nutzt. Dazu wurde ein Konzept getestet, das auf Zwischenspeicherung und Weiterleitung von Daten basiert. Dies ist nötig, da Kommunikationslinks in der Raumfahrt immer wieder unterbrochen werden oder eine sehr hohe Latenz aufweisen können, insbesondere bei der Kommunikation zwischen Planeten. Zusätzlich gab es in den letzten Jahren Netzwerktests auf der ISS wie zum Beispiel Internet Protokoll (IP) über CCSDS. Zur Lösung der Probleme in den drei Bereichen wird die Verwendung von kognitiven Systemen auf Basis verschiedener Machine Learning Konzepte vorgeschlagen [KM17].

Der Vorgängeralgorithmus RLNN wurde auch mit dem SCaN Testbed untersucht. Dabei erfolgte die Testung der Performanz des Algorithmus gegen Adaptive Coding and Modulation (ACM) und einen so genannten klassischen Reinforcement Learning Algorithmus, der tabellenbasiert ist. Die beiden Reinforcement Learning Algorithmen wurden auf die Art abgeändert, dass sie nur die Kodierung und Modulation optimieren. Alle drei Ansätze zeigten eine sehr ähnliche Performanz [FPW+19]. Dieses Ergebnis deckt sich mit den Ergebnissen des RLNN2 SCaN Testbed Tests in [HBF+18], bei welchen die Autoren zu dem Schluss kamen, dass der RLNN2 Algorithmus beim Training auf Modulation und Kodierung eine ähnliche Performanz wie ACM erreichen kann. Der Vorteil des Algorithmus liegt in der Möglichkeit, zusätzlich auf mehrere weitere Ziele zu optimieren, wie zum Beispiel Durchsatz oder Energieverbrauch [HBF+18].

Die Implementierung von ACM im DVB-S2 Standard beruht auf Lookup-Tabellen [FPW+19]. Um die Möglichkeit zu untersuchen, die Leistung von ACM zu erhöhen, wurden in [DME+16] die Lookup-Tabelle mit zwei verschiedenen Ansätzen zur Änderung der Schwellwerte für eine Änderung von Modulation und Kodierung sowie einem Neuronalen Netz ergänzt. Die vier Varianten wurden mit dem SCaN Testbed auf der ISS getestet, wobei der Datendurchsatz aller Ansätze vergleichbar war. Sowohl der Ansatz mit dem NN als auch einer der Ansätze mit geänderten Schwellwerten konnten eine kürzere Zeit bis zum Ziel-Linkmargin erzielen. Die Autoren kommen zum Schluss, dass weitere Performanzverbesserungen mit ACM möglich sind, wenn Vorhersagealgorithmen für den Link eingesetzt werden. Ein potentiell kritischer Parameter für die ACM System Performanz ist die Roundtrip Verzögerung [DME+16]. Die Roundtrip Verzögerung besteht aus der Gesamtzeit für die Übertragung der vier Pfade in Abbildung 3.2 und ist die benötigte Zeit, bis die rechte Bodenstation Feedback von der linken Bodenstation zur Signalqualität zwischen Satellit und linker Bodenstation erhält.

Im gerade vorgestellten Paper wird die Roundtrip Verzögerung als kritischer Parameter angesehen, während die Orbithöhe des Raumfahrzeugs zwischen 370 und 460 Kilometer beträgt [ESA]. Geostationäre Satelliten befinden sich in einer Orbithöhe von ungefähr 36.000 Kilometern, wodurch die Roundtrip Verzögerung bei knapp 500 Millisekunden liegt. Um höhere Datenraten bei Satelliten zu erzielen, werden auch höhere Frequenzbänder benutzt, unter anderem das Ka-Band welches zwischen 27 Gigaherz und 40 Gigaherz liegt [ITU15]. Da in diesem Frequenzbereich der größte Verlust abgesehen von der Freiraumdämpfung die Regendämpfung ist, muss bei Regen eine Anpassung der Transmitterkonfiguration in Abhängigkeit vom Wetter und den Eigenschaften des Links erfolgen. Für stationäre Bodenstationen ändert sich die Regendämpfung langsam, da sich Regen meist mit einer fast konstanten Geschwindigkeit fortbewegt. Im Gegensatz dazu kann die Regendämpfung bei sich bewegenden Bodenstationen, zum Beispiel ein Schiff oder ein Fahrzeug an Land, je nach Geschwindigkeit der Bodenstation und den regionalen Regenbedingungen sehr dynamisch werden. Die Bewegungsdynamik ist in diesem Fall umgekehrt zur Bewegung in der Anwendung in dieser Arbeit, im Paper ist der statische Teil der Satellit und der sich Bewegende Teil die Bodenstation, im Anwendungsfall dieser Arbeit ist der statische Teil die Bodenstation und der sich bewegende Teil der Satellit. Da die Roundtrip Verzögerung ohne Verarbeitung der Daten bei 500 Millisekunden liegt, ist die Messung der Übertragungsbedingungen schon veraltet, wenn diese in der Bodenstation mit einem ACM Algorithmus ankommt. Dies sieht man im Abbildung 3.2: während die linke Bodenstation die sich bewegende in einem Gebiet mit Regen ist, bildet die rechte Bodenstation, die den ACM Algorithmus enthält, das Gegenstück bei der Kommunikation [FMW14]. Zur Lösung dieses Problems wird in [FMW14] die Verwendung eines linearen Kalman Filters vorgeschlagen. Dieser Filter berechnet aus den vorherigen Daten eine Vorhersage für den nächsten k-Schritte entfernten Wert und wählt anhand dieses Wertes die

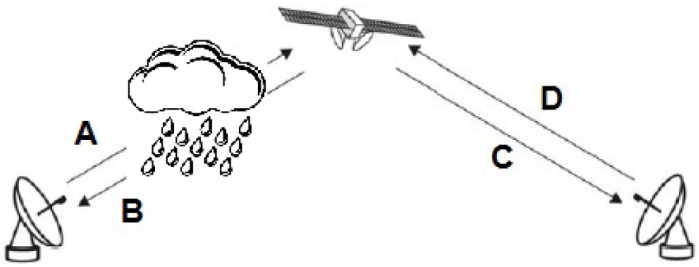

Abbildung 3.2.: Übersicht Übertragung zwischen zwei Bodentationen und einem Satelliten [FMW14]

nächste Transmitterkonfiguration. Die Ergebnisse einer Simulation dieses Algorithmus zeigen eine deutliche Verbesserung der BER, während ohne Vorhersage mehr Daten übertragen wurden bei einer deutlich höheren BER $6,7013 * 10^{-5}$ zu $2,2 * 10^{-3}$. Die signifikant höhere Fehlerrate entsteht durch das Umschalten der Konfigurationen zum falschen Zeitpunkt, wodurch gezeigt wurde, dass eine Steigerung der Performanz eines solchen Systems mit Hilfe von vorhersagenden Algorithmen möglich ist [FMW14].

Alle hier vorgestellten Algorithmen werden auf dem DVB-S2 Standard angewendet. Viele Ansätze beziehen sich auf ACM und versuchen, diese Technik zu verbessern oder zu erweitern. Es existieren einige wenige Ansätze zur Verwendung von Reinforcement Learning zur Performanzverbesserung von Kommunikationslinks in der Raumfahrt, dabei ist der RLNN2 Algorithmus nach dem Wissensstand des Autors der am längsten entwickelte Algorithmus. Zusätzlich ist er einer der wenigen Algorithmen, der mit einem Raumfahrzeug und nicht nur in einer Simulation oder unter Laborbedingungen getestet wurde. Im nächsten Kapitel wird der Aufbau der Simulationsumgebung für die Tests mit dem nachimplementierten und teilweise modifizierten RLNN2 Algorithmus vorgestellt.

4. Aufbau der Simulationsumgebung

In diesem Kapitel wird der Aufbau des Softwaresystems zur Simulation und zum Betrieb des RLNN2 Algorithmus vorgestellt. Ursprünglich sollte in dieser Arbeit ein System mit Hard- und Software entwickelt werden, mit dessen Hilfe der RLNN2 Algortihmus getestet und weiterentwickelt werden kann. Aufgrund von Lieferschwierigkeiten konnten essentielle Teile der Hardware nicht geliefert werden und somit war der Betrieb einer Übertragungssimulation mit Hardware nicht möglich. Infolgedessen wurde komplett auf die Umsetzung der Simulation mit SDRs und weiterer Hardware verzichtet. Eine komplette Reimplementierung in anderen Systemen ließ sich zeitlich nicht mehr realisieren. Aus diesen Gründen sind einige Entscheidungen zur Wahl der Softwarewerkzeuge nicht ideal für einen reinen Simulationsbetrieb, sondern wurden auf eine Hardwareumgebung ausgelegt.

Das erste Unterkapitel gibt eine Übersicht des ganzen Systems, gefolgt von einer detaillierteren Vorstellung der zur Funkübertragung eingesetzten Software sowie der Simulationssoftware für die Funkübertragung und schließlich des Datenmanagements und der Kommandierung der einzelnen Komponenten.

4.1. Infrastruktur

Das gesamte System besteht aus den drei Hauptkomponenten GNURadio für die Implementierung der Funktechnik sowie die Simulation der Funkübertragung, dem Adaptiven RLNN2 Algorithmus und einer Managementsoftware für Kommunikation und Datenverarbeitung. Abbildung 4.1 zeigt

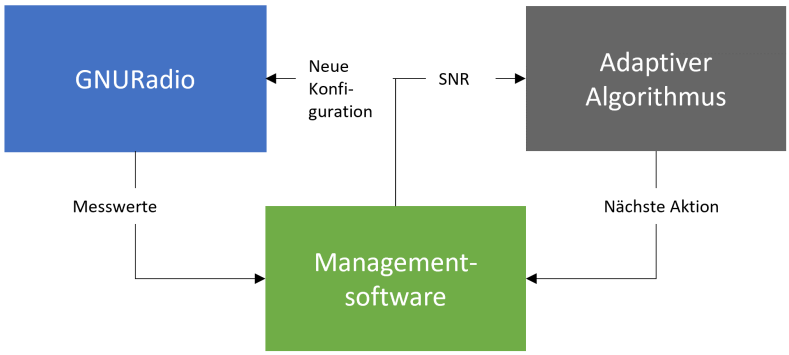

Abbildung 4.1.: Übersicht über das Softwaresystem

eine Übersicht des Gesamtsystems. Der Block GNURadio enthält einerseits die Software, um die Funksignale digital zu kodieren, modulieren, demodulieren und decodieren. Zusätzlich enthält GNURadio einen Softwareblock zur Simulation der Funkübertragung. Detailliertere Informationen zu GNURadio folgen im nächsten Unterkapitel.

Der Block des Adaptiven Algroithmus in Abbildung 4.1 enthält den veränderten RLNN2 Algorithmus. Dieser wurde in Python auf Basis des Papers [FPW+18] und der Masterarbeit [Li18] nachimplementiert unter Verwendung der Bibliothek TensorFlow. Die Wahl der Programmiersprache Python wurde getroffen, weil GNURadio Python-Dateien für die Ausführung erzeugt und somit nur eine Programmiersprache für alle Softwarekomponenten verwendet wird, was die Kommunikation zwischen den Komponenten erleichtert. Ein weiterer Vorteil von Python ist die komplikationslose Verwendung von Netzwerkprotokollen wie zum Beispiel User Datagram Protocol (UDP) und die gute Unterstützung verschiedener Bibliotheken für Machine Learning. Es existieren mehrere große Bibliotheken für Machine Learning in Python, unter anderem TensorFlow, PyTorch oder Theano. Tensorflow wird sehr aktiv entwickelt [Ins21] und bietet einfache Möglichkeiten durch die von der Bibliothek angebotenen Funktionen zahlreiche unterschiedliche Machine Learning Konzepte zu implementieren [Goo21]. Eine große Anzahl an Dokumentation zu den einzelnen Funktionen und an Beispielen zur Verwendung, siehe [Goo21], erleichtern die Arbeit mit der Bibliothek. Aus den eben genannten Gründen wird Tensorflow als Machine Learning Bibliothek in dieser Arbeit genutzt.

Die dritte Komponente des Systems ist die Managementsoftware. Diese ist hauptsächlich für die korrekte Verwaltung, Verteilung und Speicherung sämtlicher Daten verantwortlich. Während eines Überfluges schickt GNURadio alle relevanten gemessenen Parameter an die Managementsoftware. Die Software wertet die Daten aus und sendet dem Adaptiven Algorithmus den SNR Wert. Die neue Transmitterkonfiguration, welche der Adaptive Algorithmus berechnet, wird an die Managementsoftware geschickt, die dann diese Konfiguration nach der entsprechenden Verarbeitung an GNURadio schickt. Dort werden die Werte in die laufende Simulation übernommen. Weitere Details zur Managementsoftware folgen am Ende dieses Kapitels.

Die Kommunikation zwischen Managementsoftware und dem Adaptiven Algorithmus wird über UDP realisiert, da dies eine einfache Art ist Daten zwischen verschiedenen Prozessen auszutauschen. Da die Daten nur lokal auf einem Computer ausgetauscht werden, ist das Problem mit nicht ankommenden UDP Paketen zu vernachlässigen. Die Kommunikation mit GNURadio erfolgt über ZeroMQ Sockets, da dies einerseits eine der wenigen von GNURadio implementierten Möglichkeiten des Datenaustausches ist und andererseits eine gute Bibliothek für Python existiert [min21].

4.2. GNURadio

Im Block GNURadio aus Abbildung 4.1 sind zwei wesentliche Komponenten enthalten. Zum einen die Software zur Signalverarbeitung, die in diesem Kapitel vorgestellt wird und zum anderen die Software für die Übertragungssimulation, welche im nächsten Unterkapitel präsentiert wird.

Da die zu implementierende Software für die Signalverarbeitung sehr komplex ist und der zeitliche Rahmen der Masterarbeit nicht für eine komplette Implementierung des Signalpfades ausreicht, wurde die Entscheidung getroffen, eine Software zu benutzen, welche bereits einzelne Bausteine der Signalverarbeitung enthält. Die beiden Favoriten für eine solche Software sind MATLAB

und GNURadio. MATLAB enthält sehr viele Toolboxen für die Signalverarbeitung und für die Simulation von Übertragungen siehe [mat21b][mat21c][mat21a]. Eine Hardwareunterstützung für die ursprünglich geplanten SDRs von Ettus Research LLC wird nur von der Communications Toolbox angeboten [mat21d]. GNURadio bietet sehr viele Funktionen für die Signalverarbeitung an und erlaubt es so genannte OutOfTreeModules (OOT) zu installieren. Es gibt einige OOT Module, die auf Github veröffentlicht wurden und die weitere Funktionalitäten wie zum Beispiel andere Kodierungsverfahren anbieten. Zudem ist es möglich, sein eigenes OOT Modul zu implementieren und zu benutzen [wik21]. Zudem sind für viele gängige SDRs integrierte Inferfaces oder entsprechende OOT Module vorhanden, die die nötigen Interfaces anbieten. Für die ursprünglich geplanten SDRs von Ettus Research LLC existiert ein OOT Modul, das eine unkomplizierte Anbindung der Hardware erlaubt [Pan16].

Durch die unproblematische Kombination vieler Funktionen, auch aus verschiedenen OOT Modulen besteht die Möglichkeit, komplexe und stark angepasste Software zur Signalverarbeitung mit GNURadio zu erstellen. Außerdem ist es vorteilhaft, dass die graphisch implementierten Signalverarbeitungspfade vor der Ausführung in Python übersetzt und als Pythonprogramm ausgeführt werden. Dadurch ist es realisierbar, die Signalverarbeitung in GNURadio zu implementieren und das erstellte Pythonprogramm so anzupassen, dass es mit der restlichen Software kommunizieren kann und Rekonfigurationen der Parameter zur Laufzeit erlaubt. Durch die problemlose Anbindung von SDRs in GNURadio und die unkomplizierte Möglichkeit, weitere Funktionsbausteine zum Aufbau der Signalverarbeitungskette zu importieren, fiel die Wahl auf GNURadio. Positiv auf die Entscheidungsfindung wirkte sich auch die in [Sch21] entwickelte Software aus, die in GNURadio Signalverarbeitungspfade anhand eingestellter Parameter erzeugt. Diese generierten GNURadio Programme werden für Tests mit Satellitentransceivern am Institut für Raumfahrsysteme eingesetzt, die ähnliche Anforderungen stellen an Funktionalität und CCSDS Standards wie der Referenzsatellit in dieser Arbeit. Die erzeugten Programme wurden bereits mit einem Satellitentransceiver erfolgreich getestet und erfüllen somit die nötigen Teile der hier benötigten CCSDS Standards [Sch21].

Mit Hilfe dieser Software erfolgte für diese Arbeit die Erzeugung von Signalverarbeitungspfaden, die dann auf die spezifischen Parameter angepasst und verändert wurden. Zusätzlich wurde durch den Einsatz der Kombination mehrerer solcher veränderten Pfade die gesamte erforderliche Kommunikation realisiert. Anschließend wurde in das erzeugte Pythonprogramm mit den verschiedenen angepassten Signalpfaden noch die nötige Kommunikation mit der Managementsoftware eingefügt.

Abbildung 4.2 veranschaulicht den vereinfachten Aufbau der Signalverarbeitungssoftware und die Anbindung an die Managementsoftware. Es gibt drei relevante Bereich in der von GNURadio erstellten und angepassten Software: das Raumfahrzeug (S/C), die Bodenstation (G/S) und die Übertragungssimulation. Der rot markierte Uplink beginnt mit der Datasource in der Bodenstation und wird dann kodiert und moduliert. Anschließend wird das digital modulierte Signal durch die Übertragungssimulation geschickt, um im Raumfahrzeug demoduliert und dekodiert zu werden. Anhand der Messwerte, die im Raumfahrzeug beim Empfang des Signals ermittelt werden, entscheidet der Algorithmus im Anschluss an die Verarbeitung der Werte der Managementsoftware, welche Transmitterkonfiguration als nächstes benutzt wird. Der grün markierte Downlink funktioniert umgekehrt wie der Uplink und ist der Link, der vom Algorithmus angepasst wird. Sowohl die Parameter bei der Modulation und Kodierung beim Senden auf dem Raumfahrzeug und der

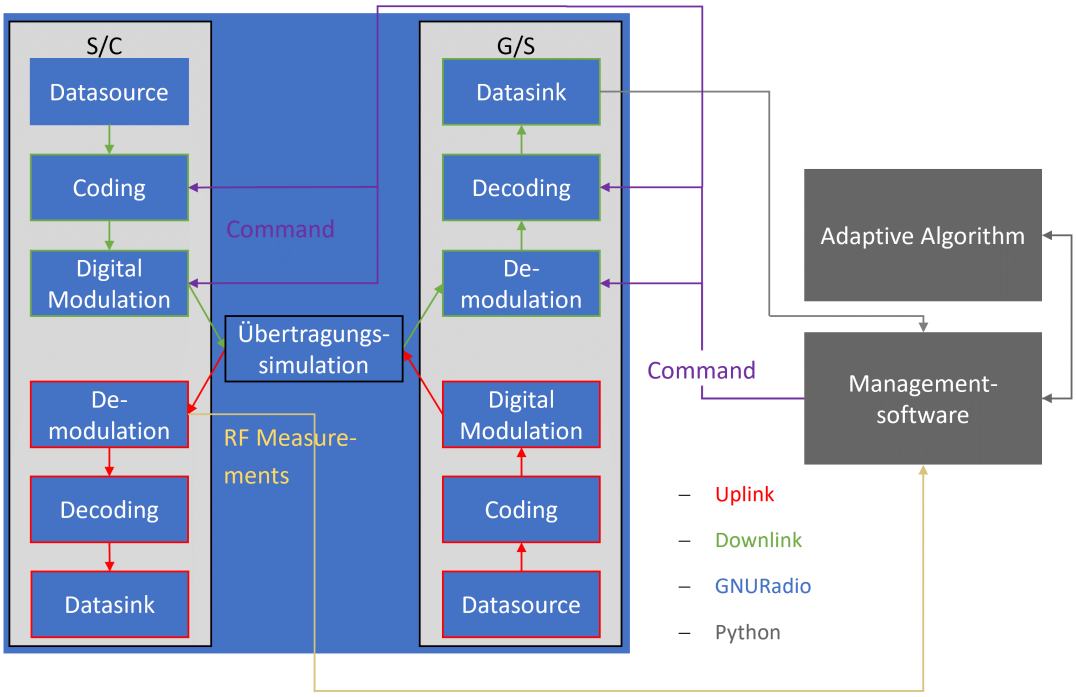

Abbildung 4.2.: Übersicht über die Signalverarbeitungssoftware

Demodulation und Dekodierung beim Empfang in der Bodenstation müssen angepasst werden. Diese Werte dürfen nicht voneinander abweichen, da ansonsten die Bodenstation das Signal nicht mehr korrekt verarbeiten kann.

Die in GNURadio definierte Kette an Funktionen zum Empfangen und Senden von Signalen ist deutlich komplexer als hier dargestellt, jeder Block in Abbildung 4.2 enthält eine ganze Kette an verschiedenen Funktionen. Der komplette GNURadiograph für alle Funkstrecken findet sich im Anhang. Im folgenden Unterkapitel wird der dritte wichtige Bereich Übertragungssimulation vorgestellt.

4.3. Übertragungssimulation

Die Wahl der Übertragungssimulation ist für die Auswertung und Weiterentwicklung des Algorithmus wichtig, da zum einen eine ähnliche Simulation wie in [FPW+18] sinnvoll ist, um die Ergebnisse vergleichen zu können, sowie eine möglichst realitätsnahe Simulation der Übertragung zu einem Satelliten nötig ist, um Aussagen zur Leistung des Algorithmus in seiner Anwendungsumgebung treffen zu können. Zum Aufbau einer möglichst anwendungsnahen Simulations- und Testumgebung, fiel die Wahl auf eine Mischung aus Hardware und Software, um die Übertragung und die Übertragungssimulation zu realisieren.

Wie in Abbildung 4.3 dargestellt, sollten in der ursprünglich geplanten Simulations- und Testumgebung zwei Computer benutzt werden, um die Bodenstation mit dem Block PC und den Satelliten mit dem Block Yoga, welcher ein ThinkPad Yoga 3. Generation mit einem i7-8550U Prozessor darstellt,

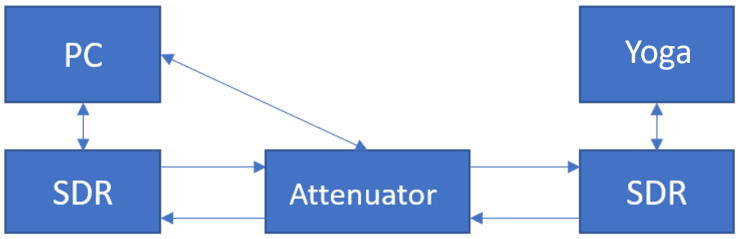

Abbildung 4.3.: Übersicht über die geplante Simulations- und Testumgebung

zu repräsentieren. An jeden dieser Computer wäre ein SDR angeschlossen worden, welches jeweils mit einem GNURadio Signalverarbeitungsgraph angesteuert werden sollte. Durch die Verwendung der SDRs wäre die Übertragung zwischen simulierter Bodenstation und Satellit unter Einsatz realer Funkhardware durchgeführt worden. Zur Simulation der Übertragungsbedingungen sollte zwischen den beiden SDRs ein regelbares Dämpfungsglied eingebaut werden, welches die verschiedenen Dämpfungen, die bei einer Übertragung auftreten, auf das Signal anwendet. Dieses Dämpfungsglied wäre durch einen Computer gesteuert worden und hätte unter anderem die sich ändernde Freiraumdämpfung bei einem Satellitenüberflug auf das Signal angewendet. In Kombination mit zusätzlich eingefügtem Rauschen auf der Senderseite zwischen Modulator und Interface des SDRs wäre so die Simulation verschiedener Szenarien und Einflüsse auf das Signal während eines Überflugs möglich gewesen.

Große Teile der Software waren bereits angepasst auf diesen Testaufbau, aber da das Kabel für den Anschluss des Dämpfungsglieds eine lange unvorhersehbare Lieferverzögerung hatte, wurde es nötig, kurz vor Beginn des letzten Drittels der Arbeit eine neue Simulationsumgebung zu finden, da ansonsten keine Untersuchung und Weiterentwicklung des Algorithmus möglich gewesen wäre. Weil der Großteil der Software, welche bis zu diesem Zeitpunkt implementiert wurde, auf die Benutzung von GNURadio und SDRs ausgelegt war und die Zeit für eine komplette Reimplementierung mit einer anderen Software für die digitale Signalverarbeitung nicht möglich war, wurde untersucht welche Möglichkeiten es in GNURadio zur Simulation eines Überflugs gibt. GNUradio bietet einige einfache Channel Models an, welche hauptsächlich für das Einfügen von Rauschen gedacht sind. Diese Channel Models bieten keine Möglichkeit, die sich verändernden Werte für die Dämpfung des Freiraums, Regen und atmosphärischen Gase während eines Überflugs zu simulieren. Bei weiteren Recherchen wurde das OOT-GNURadio Modul GNU Radio space telecommunication simulator (LEO) [Lib21] gefunden, das die erforderlichen Funktionen zur Simulation eines Überflugs bietet. Da die Recherche zu keinen weiteren Alternativen führte und die Zeit keine weitere Suche oder eine Implementierung einer vereinfachten Überflugssimulation erlaubte, fiel die Wahl auf das LEO Modul.

In diesem Modul ist es möglich, eine Bodenstation und ein Satellit mit den dazugehörigen Antennen zu definieren. In der Simulation wurden die Werte für die S/X-Neo Bodenstation und den Flying Laptop Satellit des Instituts für Raumfahrsysteme eingetragen. Zusätzlich zu den Funkparametern der einzelnen Komponenten muss eine Flugbahn des Satelliten angegeben werden, welche in diesem Fall die zu dem Zeitpunkt aktuellen Daten eines Überflugs des Flying Laptops über der

IRS Bodenstation sind und eine maximale Elevation von fast 90° aufweist und somit einen fast maximalen Überflug bezüglich Dauer und maximaler Elevation darstellt. Weitere technische Details zum Überflug finden sich im Anhang.

Das LEO Modul bietet unter anderem die Simulation der Freiraumdämpfung nach [ITU17b], die Simulation der atmosphärischen Gase Dämpfung nach [ITU16] und der Regendämpfung nach [ITU17a] mit einer Wahrscheinlichkeit der Überschreitung der simulierten Regenrate von 0,01% in einem durchschnittlichen Jahr. Somit bietet dieses Modul alle wichtigen Parameter, um einen Überflug so zu simulieren, wie er in diesem Fall benötigt wird. Um weiterhin mit Hardware den Testaufbau realisieren zu können, wurde die Möglichkeit untersucht, das LEO Modul zwischen der digitalen Modulation und dem Interface zu den SDRs in GNURadio zu benutzen. Dabei wurde festgestellt, dass die Signalwerte die aus dem Simulatorblock ausgegeben werden, extrem niedrig sind und die SDRs nicht in der Lage sind, mit diese niedrigen Werte ein Signal zu erzeugen. Dadurch ist kein sinnvoller Einsatz der SDRs mit dem Modul möglich, weshalb die Entscheidung getroffen wurde, die gesamte Signalverarbeitung und Simulation auf einem Computer auszuführen.

Bei der Untersuchung und Testung des LEO Moduls in der reinen Softwareumgebung ohne Hardware wurden mit der Zeit immer mehr Probleme entdeckt, welche zum Großteil nicht gelöst oder ausgeglichen werden konnten. In Abhängigkeit der simulierten Ausgangsleistung der Transmitter wurden die Ausgabewerte des Simulatorblocks so klein, dass diese nicht mehr von den folgenden Blöcken als Signal erkannt und verarbeitet werden konnte. Diese Problem wurde behoben, indem der Flying Laptop mit einer höheren Transmitterausgangsleistung anstatt des ursprünglich geplanten Satelliten benutzt wird. Nach Lösung dieses Problems ist in weiteren Experimenten aufgefallen, dass die Übertragung deutlich gestörter ist, als sie es von den angewendeten Signalverabeitungsblöcken sein sollte. Diese Störungen führen dazu, dass einige Messblöcke, wie zum Beispiel die SNR, sehr stark schwankende und teilweise unrealistische Werte ausgeben, welche vor der Verwendung im Algorithmus gefiltert und zum Teil verworfen werden müssen. Diese Störungen äußern sich zusätzlich in Degradierungen im Signal, welche so stark sind, dass die Decoder sehr viele Daten korrigieren müssen und nicht kodierte Daten fast immer fehlerhaft demoduliert und decodiert werden. Dies lässt sich im nur differenziell kodierten Uplink beobachten; obwohl kein Rauschen in der Übertragung hinzugefügt wird und die Signalstärke ausreichend ist, sind viele decodierte Daten fehlerhaft. Der Grund für diese Störungen liegt vermutlich in der Berechnung der Signaldämpfung des LEO Moduls, ließ sich aber auch bei einer Codeanalyse der entscheidenden Funktionen nicht herausfinden oder beheben. In weiteren Testreihen wurden alle Parameter so bestimmt, dass die Übertragung des Signals funktioniert, wenn auch stark gestört, sodass Experimente mit dem Algorithmus möglich sind. Der simulierte Übertragungskanal bietet durch die starken Störungen keine realitätsnahe Simulation der Übertragung für diesen Anwendungsfall, ließ sich aber durch die verbleibende Zeit für diese Arbeit nicht besser anpassen oder durch eine andere Simulationsumgebung ersetzten. Ein weiterer Nachteil der rein digitalen Signalverarbeitung und Simulation ist, dass diese in GNURadio nicht in Echtzeit berechnet wird, sondern so schnell wie möglich Rechenresourcen auf dem verwendeten Computer zur Verfügung stehen. Dies führt zu Schwankungen in der Geschwindigkeit der Berechnung, welche schwankend langsamer oder schneller als die Realzeit sind.

Die Analyse vieler dieser Probleme wurde dadurch erschwert, dass keine Messmöglichkeiten mit externen Werkzeugen bei der digitalen Signalverarbeitung zur Verfügung stehen, wie zum Beispiel ein Signal Analyser, welcher bei den ersten Testreihen mit den SDRs am IRS genutzt werden konnte und einige Probleme schnell zu beheben.

4.4. Managementsoftware

Um die Daten aller Softwarekomponenten an einer Stelle zu aggregieren, weiterzuverarbeiten, zu speichern sowie die erforderlichen Daten an die Softwarekomponenten zu verteilen, wird die Managementsoftware eingesetzt. Zu Beginn einer Simulation fragt die Managementsoftware nach dem Dateinamen zur Speicherung aller Daten in einer Comma-separated values (CSV) Datei für die im Anschluss ausgeführte Simulation. Alle verfügbaren Daten werden in der CSV Datei gespeichert, sodass nach der Simulation eine Auswertung der Daten durchgeführt werden kann. Um die vielen Datenströme von GNURadio empfangen und verarbeiten zu können, werden mehrere Threads in der Managementsoftware gestartet, welche alle auf den Empfang von Daten warten, um diese anschließend zu verarbeiten und dem Hauptprozess durch Queues zur Verfügung zu stellen. Der Hauptprozess aggregiert im nächsten Schritt die Daten aus den verschiedenen Queues und schreibt bei Änderungen in den Daten in eine neue Zeile der CSV Datei. Gleichzeitig schickt die Software die aktuellen SNR Werte an den RLNN2 Algorithmus und leitet neue Transmitterkonfigurationen an die GNURadiosoftware weiter.

5. Veränderungen am RLNN2 Algorithmus

Die Einschränkungen der Übertragungssimulation erforderten diverse Änderungen am Algorithmus während der Reimplementierung. Die Implementierung richtet sich hauptsächlich nach dem C++ Code in [Li18]. In dieser Arbeit besteht eine deutliche Abweichung der Implementierung auf Grund des Fehlen des Messwertes der BER. Weil außerdem in der Simulation der Roll-Off Faktor nicht variiert werden kann, wird dieser in der Implementierung als fester Wert von 0,35 definiert. Da als Referenz für den Satelliten der FLP dient, der über einen Transmitter mit fester Leistung verfügt, erfolgte die Simulation eines Transmitters mit fester Ausgangsleistung. Auf Grund dieser Änderungen und der großen Dynamiken in der Übertragungssimulation sind die Ergebnisse nicht mit den Resultaten aus dem Paper [FPW+18] vergleichbar.

5.1. Änderungen an der Architektur

Die Vereinfachung der Inputlogik für das Exploitation NN stellt eine größere Veränderung des Algorithmus dar. Bei Vereinfachung erhält das Netz ausschließlich aktuelle Messdaten als Input. Dies steht im Gegensatz zu dem in [FPW+18] definierten Algorithmus zur Inputauswahl, welcher je nach Situation auch Daten aus dem Trainingsbuffer als Input für das NN benutzt. In [HBF+18] wurde beobachtet, dass der Algorithmus bei schlechten Übertragungsbedingungen den Trainingsbuffer sehr häufig erfolglos nach einer passenden Konfiguration durchsucht. Da die Übertragungssimulation sehr schlechte Übertragungsbedingungen bietet, soll untersucht werden ob der Algorithmus auf Grund der Änderung eine höhere Datenmenge übertragen kann. Wenn die ungünstigen Übertragungsbedingungen wie in [HBF+18] eine schlechte Performanz verursachen, besteht die Möglichkeit, dass sich der Algorithmus durch die Änderung dynamischer an die Bedingungen anpasst. Zur Untersuchung dieser Anpassung erfolgt ein Vergleich der Gesamtperformanz mit dem Ziel festzustellen, ob die Variationen in einer bessere Gesamtperformanz resultieren.

5.2. Änderungen der Hyperparameter

In einem ersten Schritt der Tests werden, wie im Paper vorgestellt, 20 ExploreNNs und 10 ExploitNNs parallel benutzt und die Outputs der NN gemittelt. In dieser Konfiguration und der veränderten Architektur, wie im vorheringen Unterkapitel beschrieben, wurden mehrere Hyperparameter des Algorithmus variiert, eine Übersicht ist in Tabelle 5.1 zusammengefasst. Der Wert, welcher die Prozentzahl zur Löschung von Einträgen aus dem Trainingsdatenbuffer nach dem Training festgelget, wurde im Paper mit dem Wert 25% definiert. Um die Auswirkungen unterschiedlich umfangreicher Löschungen des Buffers zu untersuchen, erfolgte die Testung dieses Parameters mit den Werten 25%, 50% und 75%. Dadurch ändert sich zum einen die Zeit zwischen den Trainings des Netzes als auch die Menge an neuen Daten für das nächste Training. Im Paper wurde die Größe des

Hyperparemter	Wert aus Paper	veränderte Werte
Löschfaktor	25 %	25 %, 50 %, 75 %
Größe Trainingsdatenbuffer	200	200, 400, 800
Gewichte	siehe[FPW+18]	[1, 0, 0, 0, 0] [0.6, 0.1, 0.1, 0.1, 0.1] [0.2, 0.2, 0.2, 0.2, 0.2]
Ablehnungsrate für schlechte Aktionen	0.95	0.95, 0.9, 0.75

Tabelle 5.1.: Übersicht variierter Hyperparameter

Trainingsdatenbuffers auf 200 Einträge festgesetzt. Da diese Menge an Trainingsdaten eher gering ist, werden auch die Größen 400 und 800 getestet. Dadurch wird das erste Training der NN erst deutlich später ausgeführt und bei einem konstanten Löschfaktor finden die Trainings seltener statt. Die Gewichtungen für die Fitnessscore-Funktion lassen Optimierungen auf unterschiedliche Werte zu. Da die Maximierung der übertragenen Datenmenge ein Ziel ist, wird eine Gewichtung nur auf Durchsatz, mit 60% nur auf Durchsatz und alle Gewichte gleich getestet. Dies ist in Tabelle 5.1 in der Zeile Gewichte ersichtlich, wobei das erste Gewicht in der Matrix auf den Durchsatz angewendet wird. Die Intention der Veränderung der Gewichtung war die Erhöhung der Datenübertragungsrate des Algorithmus. Als weiterer Hyperparameter ist die Ablehnungsrate für schlechte Aktionen zu nennen, die im Paper mit 0,95 festgelegt wurde. Eine Verringerung des Wertes führt zum vermehrten Ausführen von Aktionen, die als nicht gut vorherberechnet wurden. Daher besteht die Erwartung der Abnahme der Performanz des Algorithmus beim Test der Werte 0,9 und 0,75. Alternativ besteht die Möglichkeit einer gleichbleibenden oder sogar verbesserten Performanz, da mehr Aktionen ausgeführt und damit auch getestet werden, die bei höheren Werten direkt verworfen werden. Aus diesen Gründen hat der Algorithmus mehr Optionen, Aktionen zu beobachten, was indirekt zu einer Erhöhung der Exploration führt.

Da der Algorithmus theoretisch auf einem Raumfahrzeug ausgeführt werden soll, ist es sinnvoll, die benötigte Rechenleistung zu reduzieren, weil diese meist begrenzt ist. Eine Möglichkeit hierfür ist die Reduktion der parallel ausgeführten NN. Dazu wird im ersten Schritt die Anzahl der Netze von 20 auf 1 reduziert und mit der ursprünglichen Implementierung in dieser Arbeit verglichen. Um die Möglichkeiten zu untersuchen, mit einem einzelnen Netz ähnliche oder bessere Ergebnisse zu erzielen, ohne dass ein einzelnes Netz Over- oder Underfitted soll dieses Netz um Dropouts für die Hidden Layer erweitert werden.

6. Ergebnisse

6.1. Baseline

Zum Vergleich der Performanz des Algorithmus mit den Anpassungen wurden zwei verschiedene Baselines gewählt. Die erste dieser Baselines ist die Nachimplementierung des RLNN2 Algorithmus und soll die Möglichkeit bieten, die Performanzveränderung durch die Veränderungen am Algorithmus vergleichen zu können. Die Messergebnisse für diese Baseline finden sich am Anfang des kommenden Unterkapitels. Die andere dieser Baselines ist aus Sicht der Kommunikation und besteht aus dem Worst-Case Szenario mit einer fest definierten Transmitterkonfiguration sowie einer Transmitterkonfiguration, die für jede 10° Änderung in der Elevation angepasste Werte aus einem statisch vorberechnetem Linkbudget erhält. Die feste Konfiguration, welche hier als Worst-Case Szenario gilt, ist das Worst-Case Linkbudget des FLPs, das mit 128 kBit/s mit einer BPSK Modulation und einer Coderate von $\frac{1}{2}$ definiert ist. Bei der Simulation dieser Konfiguration wurden 5,847 MByte übertragen. Eine Übertragungsrate von 9,234 MByte konnte bei der Simulation des Überflugs, bei dem alle 10° Evaluationsunterschied die Übertragungsgeschwindigkeit geändert wurde, erreicht werden. Bereits ab 50° Elevation bietet das Linkbudget so viel Margin, dass mit der maximalen Symbolrate, die in der Simulation möglich ist, 1.500.000 Bit/s übertragen wird, die Modulation bleibt konstant bei BPSK, genauso wie die Coderate.

Beim Training erfolgte eine Aufteilung der Daten in 70% Trainingsdaten, 15% Testdaten und 15% Validationsdaten. Bei dieser Aufteilung wurde sich an den Werten in [FPW+18] orientiert. Außerdem wurde die Loss-Funktion übernommen, die der Mean Squared Error ist. Bei der Evaluierung der Modelle nach dem Training konnte eine Abnahme des Wertes beobachtet werden, woraus sich ableiten lässt, dass bei den Trainings eine Verbesserung der Vorherberechnung der Daten durch die NN erreicht werden kann.

Eine Anleitung zur Bedingung der Software und die benötigten Bibliotheken finden sich im Anhang.

6.2. Messergebnisse

Durch die technischen Rahmenbedingungen wird die komplette Software auf einem virtuellen Server mit acht vCPUs ausgeführt, wobei die zugrunde liegenden Hardware CPUs Intel Xenon Gold 6130 sind. In einem ersten Schritt erfolgt die Testung der Performanz der Software bei unterschiedlich vielen NN mit dem Ziel, zu eruieren, ob ein direkter Vergleich der Ergebnisse einer beliebigen Anzahl paralleler NN möglich ist. Dafür wurden drei Konfigurationen der RLNN2 Software getestet: erstens wie in [FPW+18] 20 parallele Explorations NN und 10 parallele Exploitations NN, zweitens jeweils 5 parallele Explorations- und Exploitations NN sowie drittens jeweils ein Explorations- und

Anzahl Exploration NN	Anzahl Exploitation NN	Anzahl Entschei-dungen	Anzahl Trainings	Durchschnitt Entschei-dungen	Varianz Entschei-dungen
20	10	361,351,346,349,349	3,3,3,3,3	351,2	26,56
5	5	337,314,328,322,331	3,3,3,3,3	326,4	61,84
1	1	776,775,907,900,734	12,12,14,14,11	818,4	5.062,64

Tabelle 6.1.: Anzahl Entscheidungen für unterschiedlich viele parallele NN

Exploitations NN. Zur Ermittlung, ob und inwiefern die Anzahl der Entscheidungen und die Anzahl der Trainings der NN differieren, erfolgte die Messung dieser beiden Parameter in fünf Durchläufen.

Auffallend bei den Ergebnissen in Tabelle 6.1 ist die deutlich geringere Anzahl sowohl der Entscheidungen als auch der Trainings bei mehreren parallelen NN im Vergleich zu jeweils nur einem NN. Die geringere Anzahl der Trainingsdurchläufe resultiert daraus, dass die parallelen NN sequenziert trainiert werden und somit ein komplettes Training deutlich länger dauert. Die deutlich geringere Anzahl an Entscheidungen bei mehreren parallelen NN lässt sich dadurch erklären, dass zur Auswahl der jeweils nächsten Aktion nicht nur ein sondern mehrere NN ausgewertet werden müssen. Die starken Schwankungen bei einem NN für Exploration und Exploitation sowohl in der Anzahl an Entscheidungen wie an Trainings lässt sich nicht durch den Algorithmus erklären. Mögliche Erklärungen sind stark schwankende Rechenleistungsanforderungen der digitalen Signalverarbeitung oder externe Einflüsse auf die Rechenleistung durch andere Benutzer des Servers. Da der simulierte Überflug eine Dauer von 731 Sekunden aufweist, entscheidet sich der Algorithmus in der Konfiguration der ersten Zeile in Tabelle 6.1 im Durchschnitt alle 2,08 Sekunden wohingegen bei nur einem NN sowohl für Exploration als auch für Exploitation eine Entscheidung des Algorithmus im Durchschnitt alle 0,893 Sekunden erfolgt. Unabhängig von der Anzahl der NN benötigt der Algorithmus bis zur ersten Füllung des Trainingsdatenbuffers 85 Sekunden. Deshalb entscheidet er sich im ersten Szenario ab Beginn des ersten Trainings im Schnitt alle 4,27 Sekunden für eine neue Aktion, während diese Entscheidung bei der Nutzung von jeweils einem NN im Durchschnitt alle 1,04 Sekunden getroffen wird. Um das Ziel einer möglichst dynamischen Anpassung der Transmitterkonfiguration durch den Algorithmus zu erreichen, ist eine kürzere Zeit zwischen den Entscheidungen zu bevorzugen.

Das Hauptziel des Algorithmus ist es möglichst viele Daten zu übertragen. Infolgedessen wurde als Metrik der Ergebnisbewertung die Menge an übertragenen Daten gewählt. Da es kaum Paper gibt welche sowohl einen ähnlichen Anwendungsfall simulieren als auch einen ähnlichen Algorithmus benutzen, wurde versucht die Metriken aus dem RLNN2 Paper [FPW+18] zu benutzen. Die Hauptmetrik dieses Paper ist die Abweichung des beobachteten Fitnessscores vom theortisch möglichen Fitnessscore, der durch eine exhaustive search für jedes Szenario ermittelt wurde. Die Ermittlung des theoretisch besten Scores für einen Überflug ist in diesem Aufbau nicht möglich, da durch die verwendete Simulationssoftware die Übertragung nicht jedes mal einen gleichen Verlauf aufweist. Durch diese Schwankungen und die Übertragungsstörungen variiert der theoretisch erreichbare Score bei jedem Durchlauf. Daher lässt sich diese Metrik nicht in dieser Arbeit nutzen und da die weiteren Metriken im Paper auf der Abweichung beruhen, lässt sich keine Metrik aus dem Paper übernehmen. Der Test des RLNN2 Algorithmus mit der ISS in Paper [HBF+18] kommt

Anzahl Exploration NN	Anzahl Exploitation NN	Übertragene Daten (MByte)	Durchschnitt Daten (MByte)	Varianz Daten
20	10	107,872		
20	10	60,734	85,0517	370,747
20	10	86,599		
5	5	95,592		
5	5	72,971	91,083	177,806
5	5	104,686		
1	1	76,5		
1	1	87,451	81,68	20,162
1	1	81,088		

Tabelle 6.2.: Performanz für unterschiedliche viele paralleler NN

den Übertragungsbedingungen aus der hier verwendeten Simulation näher, aber auch in diesem Paper wurde mit Brute-Force der ideale Fitnessscore für jedes Sampel für jeden Überflug ermittelt und als Performanzmetrik benutzt.

Sowohl in [FPW+18] als auch in [HBF+18] wird die Geschwindigkeit des Algorithmus als kritischer Wert für die Performanz gesehen, welcher in dieser Arbeit als Anzahl Entscheidungen gemessen wird. Da dieser Wert entscheidend ist, werden nach einem Performanz Vergleich unterschiedlich vieler paralleler Netze die Experimente mit 1 Exploration und 1 Exploitation NN durchgeführt, da der Algorithmus in dieser Form deutlich schneller Entscheidungen berechnet. In [FPW+18] wurde der Vergleich zwischen 20 Exploration NN und 10 Exploitation NN gezogen und das Ergebnis sind ähnliche Werte für den Fitnessscore und ein ungefähr doppelt so großer Fehler bei nur 1 NN jeweils für Exploration und Exploitation.

In den Arbeiten, die ähnliche Bedingungen wie die dieser Arbeit zugrunde liegenden Kriterien aufweisen, kommen Metriken zum Einsatz, die hier nicht genutzt werden können. Daher wird in dieser Arbeit die Metrik der übertragenen Datenmenge benutzt, da es die einzige messbare stabile Metrik ist, welche einen Rückschluss auf die Performanz des Algorithmus zu lässt. Eine zweite aussagekräftige Metrik wäre die SNR, da diese aber in der Simulation extrem stark schwankend ist und teilweise unrealistische Werte aufweist, ist es nicht möglich diesen Wert zur Bewertung des Algorithmus heranzuziehen.

In Tabelle 6.2 sind die Ergebnisse für die drei Konfigurationen des Algorithmus vom Anfang des Kapitels. Die übertragene Datenmenge ist bei der ersten Konfiguration im Durchschnitt ähnlich hoch wie bei der dritten Konfiguration mit nur jeweils einem NN. Allerdings ist die Varianz bei der dritten Konfiguration deutlich niedriger und die Ergebnisse liegen näher zusammen. Im zweiten Szenario sind die Ergebnisse etwas besser, aber auch in diesem Szenario ist die Varianz der Werte sehr hoch. Für die Veränderungen am Algorithmus wird daher nur die Konfiguration mit 1 NN für Exploration und 1 NN für Exploitation genutzt, da dort die Ergebnisse weniger schwanken und daher besser vergleichbar sind. Bei jeder Veränderung des Algorithmus werden drei Durchläufe untersucht und aus diesen Daten der Durchschnitt und die Varianz berechnet.

Anzahl Exploration NN	Anzahl Exploitation NN	Durchschnitt Daten (MByte)	Varianz Daten
20	10	68,229	171,968
1	1	62,361	41,704

Tabelle 6.3.: Exploit Input immer auf Messdaten

Input	Dropout in Prozent	Durchschnitt Daten (MByte)	Varianz Daten
einfach	10	69,118	172,848
einfach	25	53,731	5,379
einfach	40	55,052	163,097
orginal	10	74,621	126,131
orginal	25	61,704	143,582
orginal	40	60,999	127,652

Tabelle 6.4.: Performanz für unterschiedliche Dropout Werte

In Tabelle 6.3 wurde die Änderung des Exploitinputs im Algorithmus untersucht. Dafür wurde der komplexe Entscheidungsalogrithmus für den Input, welcher je nach Situation aus den aktuellen Messwerten, Daten aus dem Trainingsdatenbuffer oder der vorherige Input besteht, abgeändert. Der Input für das Exploitation NN ist in diesem Fall immer der aktuelle Messwert. Diese Änderung wurde untersucht, weil der Algorithmus bei den in der Simulation vorherrschenden schlechten Übertragungsbedingungen, versucht im Trainingsdatenbuffer eine passende Konfiguration für den Transmitter zu finden. Dabei findet der Algorithmus normalerweiße keine gut passende Konfiguration [FPW+18]. Die Ergebnisse des Experiments sind für die beiden Fälle von 10 Exploitation und 20 Exploration NNs ähnlich der Ergebnisse mit jeweils einem NN, aber die Varianz ist im ersten Fall deutlich größer. Im Vergleich mit der ursprünglichen Implementierung des Algorithmus ist das Ergebnis um ca. 20 MByte niedriger. Daraus lässt sich schließen, dass der Algorithmus trotz der Suche im Trainingsdatenbuffer eine bessere Performanz aufweist, als in der vereinfachten Version.

In Tabelle 6.4 werden verschiedene Werte für den Dropout der Hidden Layer in den NN untersucht. In der ursprünglichen Implementierung sind keine Dropouts definiert. Sowohl für den einfachen wie auch den originalen Input bringen die Dropoutwerte von 10% das beste Ergebnis. Dabei ist die Performanz bei der modifizierten Inputbestimmung für das Exploit NN um ca. 7 MByte besser als eine Konfiguration ohne Dropout. Die Performanz des originalen Inputbestimmungsalgorithmus liegt ca. 7 MByte über der besten Version mit Dropout. Daher ist bei der ursprünglichen Implementierung kein Dropout zu bevorzugen und bei der vereinfachten Inputfindung für das Exploit NN ein Dropout von 10%.

Input	Lösch-faktor in Prozent	Durchschnitt Daten (MByte)	Varianz Daten
einfach	25	62,361	41,704
einfach	50	86,624	19,772
einfach	75	53,549	5,006
orginal	25	81,68	20,162
orginal	50	65,564	177,43
orginal	75	59,757	10,91

Tabelle 6.5.: Performanz für unterschiedliche Löschfaktor Werte

Input	Buffer-größe	Durchschnitt Daten (MByte)	Varianz Daten
einfach	200	62,361	41,704
einfach	400	100,557	110,161
einfach	800	74,543	18,428
orginal	200	81,68	20,162
orginal	400	97,252	5,415
orginal	800	83,759	12,261

Tabelle 6.6.: Performanz für unterschiedliche Buffergröße

Die Wahl des Löschfaktors nach jedem Training beeinflusst wie schnell nacheinander die NN neu trainiert werden und wie stark sich die Trainingsdaten unterscheiden. Die Wahl 25% des Buffers nach dem Training zu löschen ist für den originalen Algorithmus die Beste von den hier untersuchten Möglichkeiten. Bei Nutzung des vereinfachten Input-Algorithmus zeigt der Faktor von 50% mit Abstand die besten Ergebnisse bei einer niedrigen Varianz, wie in Tabelle 6.5 zu sehen ist.

Durch die Wahl der Größe des Trainingsdatenbuffer wird zum einen beeinflusst wann der Algorithmus das erste Mal die NN trainiert und zum anderen auch wie viele Trainingsdaten den NN zum Training zur Verfügung stehen. In den Ergebnissen dieses Experimentes in Tabelle 6.6 ist unabhängig vom Inputalgorithmus für das Exploitation NN ein Performanzpeak bei einer Größe von 400 zu sehen.

Die Gewichtungsmatrix beeinflusst die Auswirkung einzelner gemessener Parameter auf den Fitnessscore. Der erste Wert in der Matrix ist die Gewichtung für den Durchsatz, welcher in den Experimenten als Metrik gilt und maximiert werden soll. Der orginale Algorithmus zeigt das zu erwartende Verhalten, zu sehen in Tabelle 6.7. Je höher die Gewichtung auf dem Durchsatz ist, desto mehr Durchsatz konnte bei den Experimenten gemessen werden. Daher hat der Algorithmus sich in Abhängigkeit der Gewichte optimiert, was dem erwarteten Verhalten entspricht. Beim veränderten Inputalgorithmus für den Exploit NN Input ist keine Abhängigkeit zwischen Gewichten und übertragener Daten zu erkennen, was nicht den Erwartungen entspricht und auch nicht durch den Aufbau des Algorithmus erklärt werden kann.

Input	Gewichte	Durchschnitt Daten (MByte)	Varianz Daten
einfach	[1,0,0,0,0]	73,471	78,751
einfach	[0.6,0.1,0.1,0.1,0.1]	62,361	41,704
einfach	[0.2,0.2,0.2,0.2,0.2]	87,766	24,026
orginal	[1,0,0,0,0]	103,507	32,654
orginal	[0.6,0.1,0.1,0.1,0.1]	81,68	20,162
orginal	[0.2,0.2,0.2,0.2,0.2]	65,376	72,448

Tabelle 6.7.: Performanz für unterschiedliche Gewichtung

Input	Ablehnungs-rate Prozent	Durchschnitt Daten (MByte)	Varianz Daten
einfach	0,95	62,361	41,704
einfach	0.9	65,000	47,142
einfach	0,75	93,232	39,772
orginal	0,95	81,68	20,162
orginal	0.9	98,766	34,465
orginal	0,75	91,356	68,512

Tabelle 6.8.: Performanz für unterschiedliche Ablehnungsraten für schlechte Aktionen

Der originale Algorithmus zeigt eine bessere Leistung bei einer Ablehnungsrate von 90% statt 95% für Aktionen, welche bei der Exploration als schlecht eingeordnet wurden. Da der Algorithmus somit öfter Aktionen testet, welcher er sonst nicht ausprobieren würde, können bei schlechten Übertragungsbedingungen mehr verschiedene Aktionen beobachtet werden. Dadurch können auch mehr gute Aktionen, welche fälschlicherweise als schlecht vorherberechnet werden bei der Exploration beobachtet werden. Bei dem vereinfachten Inputalgorithmus für das Exploitation NN kann eine deutlich erhöhte Performanz bei einer Abhlehnungsrate von 75% beobachtet werden.

Anzahl Exploration NN	Anzahl Exploitation NN	Input	Durchschnitt Daten (MByte)	Varianz Daten
20	10	orginal	134,191	118,218
20	10	einfach	82,207	83,59
1	1	orginal	72,109	16,947
1	1	einfach	59,398	1,927

Tabelle 6.9.: Kombination aller besten Hyperparameter

Um festzustellen ob die besten ermittelten Hyperparameter kombiniert eine verbesserte Leistung des Algortihmus zeigen, wird in finalen Experiment die Parameter in Kombination mit dem orginalen und dem geänderten Inputalgorithmus mit jeweils 20 Exploration NN und 10 Exploitation NN und mit jeweils 1 NN getestet. Der orginal Algorithmus wird dabei mit keinem Dropout, einem Löschfaktor von 25%, einem Trainingsbuffer mit 400 Einträgen, einer Gewichtung von [1.0,0,0,0,0] und einer Ablehnungsrate für schlechte Aktionen von 90% konfiguriert. Der veränderte Algorithmus mit der vereinfachten Inputbestimmmung für das Exploitation NN wird mit einem Dropout von 10% für alle Hidden Layer, einem Löschfaktor von 50%, einer Trainigsbuffergröße von 400, den Gewichten [0.2,0.2,0.2,0.2,0.2] und einer Abhlehnrate von 75% konfiguriert. Der orginale Algorithmus mit den optimierten Hyperparmetern zeigt eine deutliche Performanzverbesserung bei der Verwendung mit 20 Exploration und 10 Exploitation NN. Bei der Benutzung von jeweils 1 NN liegt die Performanz unter der Konfiguration mit den Werten aus [FPW+18]. Der vereinfachte Algorithmus zeigt bei der Verwendung von jeweils einem NN keine Verbesserung durch die Wahl der Hyperparameter, sondern eine leicht reduzierte Leistung im Vergleich zu den Hyperparametern aus [FPW+18]. Bei der Verwendung von 20 Exploration NN und 10 Exploitation NN für den vereinfachten Algorithmus und den in den Experimenten herausgefunden optimierten Hyperparametern ist ein Steigerung der Performanz um ca. 11 MByte möglich. Obwohl die Optimierung der Hyperparameter mit jeweils 1 NN für Exploration und Exploitation durchgeführt wurde, zeigt die Kombination der Parameter bei der Verwendung von 20 Exploration und 10 Exploitation NN eine deutliche Verbesserung, sowohl für die ursprüngliche Implementierung als auch für die Implementierung mit vereinfachtem Inputalgorithmus für das Exploitation NN.

6.3. Einfluss der Simulationsumgebung auf die Ergebnisse

Die hohe Dynamik in der Übertragungssimulation und die stark schwankende Geschwindigkeit der digitalen Signalverarbeitung resultieren in zwei großen Herausforderungen für den Algorithmus. Zum einen versucht der Algorithmus, die starken Schwankungen und Störungen der Übertragungssimulation auszugleichen, welche deutlich größer sind als die kleine Schwankungen durch den Überflug. Ersichtlich ist dies in der Schwankung der SNR Werte, die für die BPSK Modulation extrem groß ist. Dies erschwert dem Algorithmus die korrekte Beobachtung der Änderungen durch eine neue Transmitterkonfiguration. Als zweites Problem ist die schwankende Geschwindigkeit in der Ausführung der Digitalen Signalverarbeitung und der Übertragungssimulation. Zusätzlich konnten bei den Experimenten immer wieder eine deutliche Verlangsamung aller ausgeführter Software beobachtet werden, ohne, dass ein Grund dafür ersichtlich ist. Diese Verlangsamungen treten auf obwohl die vorhandenen Kerne nicht komplett ausgelastet sind und sie treten sehr ungleichmäßig auf. Oftmals hatten diese Verlangsamungen einen starken Ausreißer im Ergebnis im Vergleich zu weiteren Experimentdurchläufen mit der gleichen Konfiguration zur Folge, weshalb diese Ergebnisse verworfen wurden. Da für diese Verlangsamungen kein Grund in der Software oder der zur Verfügung gestellten gefunden werden konnte, ist zu vermuten, dass diese Störungen durch externe Faktoren auf dem Server ausgelöst wurden. Durch diese Nichtlinearität der Zeit hat der Algorithmus unterschiedlich viel Zeit zwischen der Aktion und der dazugehörigen Änderung der SNR, wodurch die beobachtete SNR teilweise nicht der zuletzt ausgeführten Aktion zugeordnet werden kann. Dies erschwert es dem Algorithmus massiv eine gute Approximation zu erlernen, mit der die Performanz verbessert werden kann.

Anzahl Exploration NN	Anzahl Exploitation NN	Input	Hyper-parameter	Durchschnitt Daten (MByte)	Varianz Daten	Performanz-verbesserung in %
20	10	orginal	orginal	85,051	370,747	0
20	10	orginal	angepasst	134,191	118,218	57,777
20	10	einfach	angepasst	82,207	83,59	-3,344
1	1	orginal	orginal	81,68	20,162	-3,964
1	1	orginal	angepasst	72,109	16,947	-15,217
1	1	einfach	angepasst	59,398	1,927	-30,162

Tabelle 6.10.: Vergleich aller Änderungen mit der Baseline-Implementierung

Trotz der großen Herausforderungen für den Algorithmus wurde in jedem Simulationsdurchlauf eine deutlich höhere Performanz als bei den klassischen Linkbudget berechneten Transmitterkonfigurationen erreicht. Um diese Ergebnisse in einer möglichst realitätsnahen Simulationsumgebung zu verifizieren, wird die Verwendung eines besseren und stabileren Simulationsumgebung empfohlen. Diese Simulationsumgebung kann aus einem Hardwaresimulator wie dem ursprünglich geplanten Simulator bestehen oder durch Überflugsimulationen in MATLAB erstellt werden. Da die Ausführung der Software teilweise unvorhersehbar verzögert wurde, ist eine Nachprüfung der Experimente auf einem Server ohne weitere Nutzer mit einer mindestens so großen Rechenleistung wie hier zu Verfügung stand zu empfehlen. Es bestand während der Arbeit kein Zugriff auf ein solches System, daher konnten die Ergebnisse nicht in einer kontrollierten Umgebung überprüft werden. Zudem ist es durch die Konfiguration des vorhandenen Servers nicht möglich Teile der Software auf einem anderen PC auszuführen, daher ist eine Überprüfung der Ergebnisse mit einer besseren Simulationsumgebung und der ursprünglich geplanten Nutzung eines Laptops zur Ausführung des Algorithmus empfohlen.

6.4. Bewertung der Ergebnisse

In Tabelle 6.10 sind die Ergebnisse der größten Änderungen am Algorithmus dargestellt. In der ersten Zeile ist die Baseline Implementierung des Algorithmus zu sehen, hier werden die ursprüngliche Anzahl an NN, der ursprüngliche Input-Algorithmus für das Exploitation NN und die originalen Hyperparameter benutzt. Die Spalte Input beschreibt, ob der originale oder der vereinfachte Input-Algorithmus für das Exploitation NN genutzt wird und in der Spalte Hyperparameter ist dargestellt ob die originalen Hyperparameter oder eine Kombination der Besten Hyperparameter aus dem Unterkapitel Messergebnisse genutzt wird. Die Ergebnisse zeigen, dass der originale Algorithmus mit angepassten Hyperparametern ein um 57,8% besseres Ergebniss erziehlt, als die ursprüngliche Implementierung. Sowohl das Ergebnis mit angepassten Hyperparametern als auch die Baseline-Implementierungsergebnisse schwanken stark, dies lässt sich in den höheren Varianz-Werten sehen. Daher ist eine Verifizierung mit einer stabileren Übertragungssimulation zu empfehlen um die Werte zu bestätigen oder zu entkräften. Der Algorithmus mit der angepassten Architektur und angepassten Hyperparametern liefert ein ähnliches Ergebnis wie die Baseline-Implementierung mit einer Performanzverschlechterung von -3,3%, ähnlicher der Nutzung von jeweils 1 NN für Exploitation und Exploration. Die Verwendung von jeweils 1 NN und angepassten

Hyperparametern zeigt eine deutlich schlechtere Performanz als die Baseline-Implementierung bei einer veränderten Architektur um -30,12% und bei der originalen Architektur um -15,2%. Das Ergebnis der angepassten Architektur mit den angepassten Hyperparametern und jeweils 1 NN für Exploration und Exploitation ist deutlich schlechter als alle anderen Ergebnisse und weist eine niedrige Varianz auf. Daher ist die weitere Untersuchung dieser Kombination, bei der aktuellen Übertragungssimulation, nicht sinnvoll und sollte nur dann stattfinden, wenn in einer besseren Simulationsumgebung deutlich bessere Ergebnisse beobachtet werden können.

Die Performanz sämtlicher Ansätze ist deutlich höher als bei den beiden Kommunikationsbaselines. Dies ist ein zu erwartendes Ergebnis, da die Baselines, wie in der Raumfahrt typisch, sehr konservativ auf den Worst-Case ausgelegt sind. Daher ist es in jedem Fall sinnvoll die Ergebnisse der Experimente mit einer realistischen Überflugssimulation zu verifizieren und gegebenenfalls zu optimieren.

7. Probleme und Verbesserungen

In diesem Kapitel geht es um die Probleme, welche in der Arbeit durch Lieferverzögerungen aufgetreten sind. Im ersten Teil werden die ursprünglich geplanten Tests und Aufbauten beschrieben, anschließend wird auf die Probleme eingegangen, welche aus der Umstellung resultieren.

Die ursprünglich geplante Test- und Entwicklungsumgebung für den Algorithmus besteht aus einem Grundaufbau mit zwei Computern, welche zum einen die Bodenstation und zum anderen den Satelliten simulieren sollen. An diese Computer wird jeweils ein SDR angeschlossen, über welche dann die Funkübertragung realisiert wird. In einem ersten Schritt wird die Verbindung der SDRs mit Kabeln realisiert. In diesem Aufbau wird die digitale Signalverarbeitung getestet werden, damit sichergestellt ist, dass die SDRs und die digitale Signalverarbeitung in jeder Konfiguration korrekt arbeitet und Daten senden und empfangen kann. Diese Tests wurden für eine erste Auswahl an Konfigurationen erfolgreich abgeschlossen.

Der nächste Schritt wäre der Einbau des digital regelbaren Dämpfungsgliedes um eine dynmische Veränderung der Dämpfung während eines Überflugs zu simulieren. In diesem Aufbau waren Tests mit verschiedene Konfigurationen und unterschiedlichen Dämpfungen geplant um die in Linkbudgetes berechneten Werte für verschiedene Situationen zu verifizieren. Durch Lieferverzögerungen des Anschlusskabels für das digitale Dämpfungsglied musste eine andere Lösung gefunden werden, welche in Unterkapitel 4.3 vorgestellt wurde. Nach dieser Verifizierung des Testaufbaus wären erste Tests mit dem Algorithmus durchgeführt und eine Baseline zum Vergleichen der Ergebnisse erstellt worden. Nach der Weiterentwicklung des Algorithmus waren weitere Tests mit einem erweiterten Testaufbau geplant. Der nächste Entwicklungsschritt beim Testaufbau wäre der Einbau von Antennen zur Übertragung der Signale durch die Luft gewesen. Dieser Aufbau ist in Abbildung 7.1 zu sehen, wobei der PC die Bodenstation und die Channel Simulation aus dem Dämpfungglied und in der digitalen Signalverarbeitung eingefügtem Rauschen besteht. Das Signal wird dann an eine Antenne übertragen um dort gesendet und mit einer weiteren Antenne mit SDR empfangen zu werden. Dieser Aufbau wäre der Testaufbau mit dem Algorithmus gewesen, der der Realität am nächsten gekommen wäre. Zur Verifizierung der genutzten CCSDS Standards waren auch Tests der digitalen Signalverarbeitung gegen die IRS Bodenstation geplant.

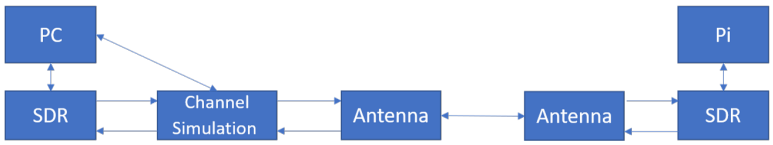

Abbildung 7.1.: Übersicht über die geplante Simulations- und Testumgebung mit Antennen

Da die Regendämpfung mit steigender Frequenz einen immer größeren Teil der Gesamtdämpfung während der Übertragung ausmacht, waren auch Tests in höheren Frequenzen geplant. Die zur Verfügung gestellten SDRs können Signale bis zu einer Frequenz von 6 GHz senden und empfangen [Res21]. Für die Tests mit höheren Frequenzen war der Input einer Wetterstation oder eines Regensensors für den Algorithmus geplant, um zu untersuchen ob der Algorithmus bei der Verfügbarkeit von Wetterdaten sich gut an die Verhältnisse anpassen kann.

Durch die Umstellung der hardwarebasierten Simulationsumgebung auf eine reine Softwaresimulationsumgebung sind einige Probleme und Schwierigkeiten aufgetaucht. Da durch die Umstellung ein deutlich höherer Rechenaufwand nötig ist, musste die digitale Signalverarbeitung auf einen Server ausgelagert werden, da dem Autor keine ausreichend leistungsstarken Computer zur Verfügung stehen. Der vom Institut für Parallele und Verteilte Systeme bereit gestellte virtuellen Server hat die zwei Einschränkungen. Zum einen wird der virtuelle Server auf einem von mehreren Benutzern geteilten Server ausgeführt, dies führte bei den Experimenten zu starken Schwankungen bei der beobachteten Geschindigkeit der Berechnung und somit ab und zu zu starken Ausreißern bei den Ergebnissen. Dies wurde durch eine extrem hohe Varianz in Kombination mit einem verlangsamten Aufzeichnung der Systemzeit in den Datenaufzeichnungen beobachtet. Diese Experimente wurden wiederholt, wenn bei den Tests eine verlangsamte Ausführung und eine große Varianzen beobachtet wurden. Diese komplett wiederholten Testreihen weisen deutlich konsistentere Ergebnisse auf, daher sind diese Schwankungen auf externe Faktoren zurückzuführen und nicht auf die Performanz des Algorithmus. Der zweite Nachteil der Nutzung des virtuellen Servers ist die fehlende Möglichkeit des SSH-Port Forwarding, weshalb die komplette Software auf dem Server ausgeführt werden musste. Damit war es nicht möglich den Algorithmus wie ursprünglich geplant auf einem Laptop auszuführen oder nur die Nutzung des Servers für die Übertragungssimulation und die Nutzung zweier privater Computer für die Simulation von Bodenstation und Satelliten. Somit sind Untersuchungen der Geschwindigkeit des Algorithmus in verschiedenen Konfigurationen auf einer Plattform mit konstanter und garantierten Rechenleistung nicht möglich gewesen. Durch die Nutzung der reinen digitalen Signalverarbeitung in GNURadio wird die Übertragung nicht mehr in Realzeit durchgeführt, sonder so schnell wie die Signalverarbeitung mit der zur Verfügung gestellten Rechenleistung ausgeführt werden kann. Dadurch erhält der Algorithmus die Ergebnisse der Änderung der berechneten Aktion in unterschiedlichen Geschwindigkeiten. Dieses Problem konnte nicht in der vorhandenen Zeit gelöst werden, ein Ansatz für die Lösung dieses Problems ist die Verwendung eines extrem rechenleistungsstarken Servers für den Algorithmus, damit dieser schneller als benötigte Realzeit ausgeführt werden kann, wenn die Signalverarbeitung schneller abläuft als Realzeit. Zusätzlich ist eine Software, welche konstant die Abweichung der Ausführungsgeschwindigkeit misst und Nachrichten und Ausführungszeiten des ganzen Systems an die Geschindigkeit der Signalverabeitung anpasst nötig. Durch solch ein Zeitverwaltungsprogramm wäre es möglich die geänderten Messwerte durch eine neue Transmitterkonfiguration zeitlich konstant dem Algorithmus als Input zu liefern.

Die großen Probleme in dieser Arbeit sind alles Folgen von der erzwungenen Umstellung der Simulationsumgebung nachdem schon die Hälfte der zur Verfügung stehenden Zeit verstrichen war. Mit dem ursprünglich geplanten Simulations- und Testaufbau wären vermutlich deutlich bessere und konsistentere Ergebnisse möglich gewesen. Einige der Hauptprobleme, wie das fehlen realistischer und stabiler Messwerte, zum Beispiel die SNR, oder die Ausführung in nicht Realzeit, waren vor der Umstellung der Simulationsumgebung nicht vorhanden. Erste Testreihen mit der vorhandenen Hardware zeigten verlässliche und realistische Messwerte und keine Probleme wie die massiven

Störungen durch die Übertragungssimulation. Daher ist eine Wiederholung der Experimente mit dem ursprünglichen Testaufbau sinnvoll und würde belastbarere Daten für eine Anwendung des Algorithmus schaffen.

8. Zusammenfassung und Ausblick

Diese Arbeit befasst sich mit der Optimierung eines Algorithmus zur Verbesserung der Kommunikation zwischen Bodenstation und Satelliten. Durch die technischen Einschränkungen und Rahmenbedingungen werden in der Raumfahrt Systeme oftmals nach dem Worst-Case ausgelegt und implementiert. Dies trifft auch auf das Kommunikationssystem von Satelliten zu, wobei oftmals eine Transmittkonfiguration definiert wird, nach den Ergebnissen und Parametern des Worst-Case-Linkbudgets. Das Worst-Case-Linkbudget enthält alle maximalen Verluste und Dämpfungen, wie zum Beispiel die maximale Freiraumdämpfung, wenn der Satellit am Horizont ist, und soll garantieren, dass immer mit dem Satelliten kommuniziert werden kann. Da in den meisten Fälle deutlich bessere Übertragungsbedingungen herrschen, als im Worst-Case-Linkbudget angenommen, werden somit deutlich weniger Daten übertragen als technisch möglich wäre. Um dieses Problem anzugehen wird an adaptiven Algorithmen geforscht, welche die Transmitterkonfiguration dynamisch an die Übertragungsbedingungen anpasst. Ein System im Übertragungsstandard für Fernsehsatelliten ist das Adaptive Coding und Modulation (ACM) System, welches sich mit Hilfe einer Look-Up Tabelle an die Übertragungsbedingungen anpassen kann. Ein weiterer verbreiteter Standard für die Satellitenkommunikation ist CCSDS, welcher unter anderem bei der Kommandierung von Satelliten und der Übertragung von Nutzlastdaten zum Einsatz kommt. In diesem Standard wurde bisher kein adaptives Systems veröffentlicht. Da bei vielen Missionen die übertragbare Menge an Nutzlastdaten ein wichtiger Faktor ist, werden auch unabhängig vom ACM System adaptive Algortihmen benötigt [KM17]. Ein solcher Algorithmus wird in der Arbeit untersucht und angepasst, die Grundlage für diesen Algorithmus bietet die Implementierung des in [FPW+18]. Dieser Algorithmus beruht auf einer weiterentwickelten Art des Deep-Q-Learnings mit Expirience Replay und besteht aus drei Hauptkomponenten. Zum einen gibt es einen Agenten, welcher mit der Umwelt interagiert, indem er eine neue Transmitterkonfiguration bestimmt anhand der gemessenen Werte des empfangenen Signals. Zur Bestimmung der Transmitterkonfiguration wählt der Agent aus ob eine Exploration oder eine Exploitation erfolgen soll. Bei der Exploitation benutzt der Algortihmus ein Deep NN und berechnet die am besten passende Konfiguration. Auch bei der Exploration benutzt der Algorithmus ein Deep NN, dabei klassifiziert der Algorithmus alle möglichen Konfigurationen im aktuellen Zustand in gute und schlechte Aktionen. Danach entscheidet sich der Agent ob eine gute oder eine schlechte Konfiguration benutzt wird und wählt zufällig aus der jeweiligen Menge eine Konfiguration.

Dieser Algorithmus wird in einer Simulationsumgebung getestet und verändert. Diese Simulationsumgebung wurde auf Grund von Lieferverzögerungen benötigter Teile in GNURadio realisiert, da bereits die digitale Signalverarbeitung in GNURadio implementiert wurde. Durch die Verwendung dieser Übertragungssimulation gibt es Einschränkungen bei den Tests und auch bei der Aussage der Ergebnisse. Die Baseline Implementierung weißt, auf Grund fehlender oder unzuverlässiger Messwerte in der digitalen Signalverarbeitung, einige Unterschiede zu der ursprünglichen Implementierung in [Li18]. Nach ersten Tests der Simulationsumgebung mit dem Algorithmus wurden starke Schwankungen bei der Anzahl an Entscheidungen des Algorithmus während eines simulierten

Überflugs sichtbar. Diese lassen sich teilweise aus dem sequentiellen Training mehrere parallel ausgewerteter NN für Exploration und Exploitation erklären, als ein weiterer Grund wird eine schwankende zur Verfügung stehende Rechenleistung durch die geteilte Benutzung des Servers vermutet. Trotz verschiedener Schwierigkeiten und Einschränkungen sowohl der Übertragungssimulation als auch des genutzten Servers wurden Veränderungen in der Architektur und der Hyperparameter des Algorithmus untersucht. Eine Anpassung der Hyperparameter führte dabei zu einer Übertragung von 57,8% mehr Daten als bei der Baseline-Implementierung. Sowohl die Veränderung in der Architektur als auch die Reduzierung der parallel ausgeführten NN führte zu leichten Performanzeinbusen von 4,0% und 3,3%. Die Kombination der veränderten Architektur und angepasster Hyperparameter führt zu deutlich schlechteren Ergebnissen im Verhältnis zur Baseline-Implementierung. Auf Grund der Probleme der softwarebasierten Übertragungssimulation lassen sich die Ergebnisse nicht mit dem ursprünglichen Paper vergleichen. Zudem ist die Aussage für den Anwendungsfall in der Realität stark eingeschränkt.

Auf Grund der Schwierigkeiten und Probleme mit der Übertragungssimulation und den nicht konstanten Bedingungen auf dem Server, ist eine Wiederholung der Experimente mit dem Hardwarebasierten Übertragungssimulation und Software Defined Radios sinnvoll um die Ergebnisse für den Anwendungsfall zu verifizieren oder zu widerlegen. Da die Regendämpfung eines Funksignals mit der Frequenz an Bedeutung gewinnt und Regen je nach Region auf der Erde sehr dynamisch und sehr stark auftreten kann, ist eine Untersuchung des Algorithmus bei höheren Frequenzen sinnvoll, da dort mehr Dynamiken bei der Übertragung auftreten können. Um diese Algorithmus zukünftig weiter zu entwickeln sind weitere Variationen in der Architektur des Algorithmus und der NN für Exploration und Exploitation sinnvoll. Damit könnte es möglich sein die benötigte Anzahl an NN zu senken, was zu einer schnelleren Ausführung des Algorithmus führt oder zu einem Sinken der benötigten Rechenleistung, wodurch eine Anwendung auf einem Satelliten realistischer wird. Eine weitere mögliche Verbesserungen am Algorithmus wäre der Input weitere Messdaten für den Algorithmus wie zum Beispiel die Daten einer Wetterstation mit Daten zur aktuellen Regenrate oder Feuchtigkeit.

Eine weitere Möglichkeit wäre das vortrainieren der NN bei mehreren simulierten Überflügen und ein Test der Performanz bei ähnlichen Bedingungen, womit es möglich wäre angepasste NNs für bestimmte Übertragungsbedingungen zu wählen. In Kombination mit diesen vortrainierten NN kann auch eine Verwendung von Wettervorhersagen als Input für das Netz oder für die Wahl des richtigen NNs für den nächsten Überflug getroffen werden.

Weitere mögliche Schritte zur Verwendung dieses Algorithmus mit einem Satelliten ist der Aufbau einer Bodenstation, welche entweder diesen Algorithmus beinhaltet und den Transmitter eines Satelliten bei einem Überflug entsprechend adaptiv konfiguriert. Bei diesem Szenario besteht des Problem, dass der Algorithmus die beste Aktion für die Zukunft, bis das Signal beim Satelliten ist und die Rekonfiguration abgeschlossen ist, vorherberechnen muss, da sonst die Konfiguration bei der Ausführung sonst veraltet und nicht mehr ideal ist. Ein zweiter Ansatz wäre es diesen Algorithmus nach deutlichen Verbesserungen bezüglich der benötigten Rechenleistung mit Hilfe eines Software Defined Radios auf einem Satelliten zu implementieren und die Bodenstation auf diesen Betrieb anzupassen.

Literaturverzeichnis

[AA17a] C. S. N. Aeronautics, S. Administration. *TC Synchronization and Channel Coding*. Techn. Ber. 231.0-B-3. CCSDS Secretariat National Aeronautics und Space Administration, Washington, DC, USA, 2017 (zitiert auf S. 13, 15).

[AA17b] C. S. N. Aeronautics, S. Administration. *TM Synchronization and Channel Coding*. Techn. Ber. 131.0-B-3. CCSDS Secretariat National Aeronautics und Space Administration, Washington, DC, USA, 2017 (zitiert auf S. 15–17).

[AA20] C. S. N. Aeronautics, S. Administration. *Radio Frequency and Modulation Systems*. Techn. Ber. 401.0-B-30. CCSDS Secretariat National Aeronautics und Space Administration, Washington, DC, USA, 2020 (zitiert auf S. 13, 15, 16).

[CSH14] N. Crisp, K. Smith, P. Hollingsworth. „Small satellite launch to LEO: a review of current and future launch systems“. In: *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan* 12.ists29 (2014), Tf_39–Tf_47 (zitiert auf S. 11).

[DKK+15] M. Darmetko, S. Kozłowski, K. Kurek, J. Skarzyński, J. Modelski, K. Szczygielska, M. Stolarski. „Adaptive communication system using software defined radio“. In: *2015 IEEE MTT-S International Microwave and RF Conference (IMaRC)*. 2015, S. 185–187 (zitiert auf S. 12, 30).

[DME+16] J. A. Downey, D. Mortensen, M. Evans, J. C. Briones, N. Tollis. „Adaptive coding and modulation experiment with NASA's Space Communication and Navigation Testbed“. In: *34th AIAA International Communications Satellite Systems Conference*. 2016, S. 11 (zitiert auf S. 31).

[E L00] A. J. E. Lutz M. Werner. *Satellite Systems for Personal and Broadband Communications*. Springer, Berlin, Heidelberg, 2000 (zitiert auf S. 18, 19).

[EA16] M. C. Erturk, Y. Aksan. „A tool for beamforming and real-time link budget analysis in aeronautical communications using kinematics“. In: *International Journal of Aerospace Engineering* 2016 (2016) (zitiert auf S. 19).

[ESA] ESA. *ISS: International Space Station*. Besucht 23.10.21. URL: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_Space_Station/ISS_International_Space_Station (zitiert auf S. 31).

[FMW14] P. V. R. Ferreira, R. Metha, A. M. Wyglinski. „Cognitive radio-based geostationary satellite communications for Ka-band transmissions“. In: *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2014, S. 1093–1097 (zitiert auf S. 13, 30–32).

[FPW+18] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, D. J. Mortensen. „Multiobjective Reinforcement Learning for Cognitive Satellite Communications Using Deep Neural Network Ensembles“. In: *IEEE Journal on Selected Areas in Communications* 36.5 (2018), S. 1030–1041 (zitiert auf S. 12, 15, 23–27, 29, 34, 36, 41–46, 49, 57).

[FPW+19] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilen, R. C. Reinhart, D. J. Mortensen. „Reinforcement learning for satellite communications: From LEO to deep space operations“. In: *IEEE Communications Magazine* 57.5 (2019), S. 70–75 (zitiert auf S. 31).

[Goo21] Google. *Bibliotheken & Erweiterungen*. 26. Okt. 2021. URL: https://www.tensorflow.org/resources/libraries-extensions (zitiert auf S. 34).

[GZK09] G. Gardikis, N. Zotos, A. Kourtis. „Satellite media broadcasting with adaptive coding and modulation“. In: *International journal of digital multimedia broadcasting* 2009 (2009) (zitiert auf S. 30).

[HBF+18] T. M. Hackett, S. G. Bilén, P. V. R. Ferreira, A. M. Wyglinski, R. C. Reinhart, D. J. Mortensen. „Implementation and On-Orbit Testing Results of a Space Communications Cognitive Engine“. In: *IEEE Transactions on Cognitive Communications and Networking* 4.4 (2018), S. 825–842 (zitiert auf S. 29–31, 41, 44, 45).

[Ins21] G. Insights. *Commit Activity*. 26. Okt. 2021. URL: https://github.com/tensorflow/tensorflow/graphs/commit-activity (zitiert auf S. 34).

[ITU15] ITU-R. *Nomenclature of the frequency and wavelengh bands used in telecommunications*. Techn. Ber. ITU-R, 2015 (zitiert auf S. 13, 31).

[ITU16] ITU-R. *Attenuation by atmospheric gases*. Techn. Ber. ITU-R, 2016 (zitiert auf S. 38).

[ITU17a] ITU-R. *Characteristics of precipitation for propagation modelling*. Techn. Ber. ITU-R, 2017 (zitiert auf S. 38).

[ITU17b] ITU-R. *Propagation data required for the evaluation of interference between stations in space and those on the surface of the Earth*. Techn. Ber. ITU-R, 2017 (zitiert auf S. 38).

[ITU95] ITU-R. *MAXIMUM PERMISSIBLE ALUES OF POWER FLUX-DENSITY AT THE SURFACE OF THE EARTH PRODUCED BY SATELLITES IN THE FIXED-SATELLITE SERVICE USING THE SAME FREQUENCY BANDS ABOVE 1 GHz AS LINE-OF-SIGHT RADIO-RELAY SYSTEMS*. Techn. Ber. ITU-R, 1995 (zitiert auf S. 18).

[JJD19] A. Jagannath, J. Jagannath, A. Drozd. „Artificial Intelligence-based Cognitive Cross-layer Decision Engine for Next-Generation Space Mission“. In: *2019 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*. 2019, S. 1–6 (zitiert auf S. 12, 30).

[KLM96] L. P. Kaelbling, M. L. Littman, A. W. Moore. „Reinforcement learning: A survey“. In: *Journal of artificial intelligence research* 4 (1996), S. 237–285 (zitiert auf S. 20–22).

[KM17] E. J. Knoblock, A. Madanayake. „Assessment of Cognitive Communications Interest Areas for NASA Needs and Benefits“. In: *Proceedings of the IEEE Cognitive Communications for Aerospace Applications Workshop, Cleveland, OH, USA*. 2017, S. 27–28 (zitiert auf S. 30, 57).

[KRS11] S. Kapoor, s. Rao, G. Singh. „Opportunistic Spectrum Sensing by Employing Matched Filter in Cognitive Radio Network“. In: Juli 2011, S. 580–583. DOI: 10.1109/CSNT.2011.124 (zitiert auf S. 18).

[Li18] M. Li. „Extension on Adaptive MAC Protocol for Space Communications“. Magisterarb. Worcester Polytechnic Institute, 2018 (zitiert auf S. 34, 41, 57).

[Lib21] Librespacefoundation. *A GNU Radio space telecommunication simulator*. 2021. URL: https://gitlab.com/librespacefoundation/gr-leo (zitiert auf S. 37).

[mat21a] mathworks.com. *MATLAB für die drahtlose Kommunikation*. 26. Okt. 2021. URL: https://de.mathworks.com/solutions/wireless-communications.html (zitiert auf S. 35).

[mat21b] mathworks.com. *RF-Toolbox*. 26. Okt. 2021. URL: https://de.mathworks.com/products/rftoolbox.html (zitiert auf S. 35).

[mat21c] mathworks.com. *Satellite Communications Toolbox*. 26. Okt. 2021. URL: https://de.mathworks.com/products/satellite-communications.html (zitiert auf S. 35).

[mat21d] mathworks.com. *USRP Support from Communications Toolbox*. 26. Okt. 2021. URL: https://de.mathworks.com/hardware-support/usrp.html (zitiert auf S. 35).

[min21] minrk. *zeromq / pyzmq*. 12. Okt. 2021. URL: https://github.com/zeromq/pyzmq (zitiert auf S. 34).

[MKS+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. „Playing atari with deep reinforcement learning“. In: *arXiv preprint arXiv:1312.5602* (2013) (zitiert auf S. 22, 23).

[Pan16] N. Pandeya. *Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux*. 1. Mai 2016. URL: https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux (zitiert auf S. 35).

[Res21] E. Research. *USRP B200mini-i*. 2021. URL: https://www.ettus.com/all-products/usrp-b200mini-i-2/ (zitiert auf S. 54).

[Sch21] R. Schweigert. „Development of a Digital Radio Frequency Special Checkout Equipment (RF SCOE) Solution for Small Satellites“. Magisterarb. Institute of Space Systems University Stuttgart, 2021 (zitiert auf S. 35).

[Spl07] Splash. *Bitfehlerwahrscheinlichkeit (BER) als Funktion des Eb/N0 für verschiedene PSK-Modulationsarten*. 2007. URL: https://de.wikipedia.org/wiki/Eb/N0#/media/Datei:PSK_BER_curves.svg (zitiert auf S. 20).

[ST20] B. Space, Technology. *Smallsats by the Numbers 2020*. 2020. URL: https://brycetech.com/reports/report-documents/Bryce_Smallsats_2020.pdf (zitiert auf S. 11).

[Sut18] A. G. Sutton Richard S. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press, 2018 (zitiert auf S. 22, 23).

[wik21] wiki.gnuradio.org. *OutOfTreeModules*. 19. Aug. 2021. URL: https://wiki.gnuradio.org/index.php/OutOfTreeModules (zitiert auf S. 35).

Alle URLs wurden zuletzt am 01. 11. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Esslingen 2.11.21 C. Hoeger

Ort, Datum, Unterschrift

A. Anhang

A.1. Software

Benötigte Software:
Python Version 3.8.10
GNURadio 3.8.1.0
Benötigte GNURadio OOT-Module:
gr-satellites 3.6.0 (https://github.com/daniestevez/gr-satellites)
Hinweis: Je nach Version und Betriebsystem gibt es Probleme mit Volk, siehe https://gr-satellites.readthedocs.io/en/latest/installation.html
gr-leo Version vom 1.11.2021 (https://gitlab.com/librespacefoundation/gr-leo)
gr-rf_scoe kompatibel zu GNURadio 3.8.1 (erhältlich auf Anfrage bei: rschweigert@irs.uni-stuttgart.de)
Benötigte Python Bibliotheken: math, queue, struct, zmq, pmt, csv, multiprocessing, socket, json, datetime, tensorflow, numpy, random, time

Anleitung zum Starten der Softwarekomponenten:
0. (Vor dem ersten Start in DataCapture/main.py in Zeile 387 den Pfad zum Ablegen der Logs auf das aktuelle System anpassen)
1. DataCapture/main.py auf der Konsole starten
2. Dateinamen zu Speicherung eingeben
3. Tensorflow/main.py starten
4. allWorkModify.py starten

Anleitung zum Ändern der Parameter im Algorithmus in Tensorflow/main.py:
Zeile 394: Ablehnungsrate für schlechte Aktionen
Zeile 396: Löschfaktor
Zeile 402: Größe Trainingsbuffer
Zeile 405: Gewichtungsmatrix
Zeile 347-353: Definition Exploration NN, Dropout in Zeile 350 und 352
Zeile 364-368: Definition Exploitation NN, Dropout in Zeile 367
Zeile 346: Anzahl paralleler Exploration NN
Zeile 362: Anzahl paralleler Exploitation NN
Zeile 442-443: Wahl zwischen originalen und modifziertem Input-Algorithmus für das Exploitation NN

A.2. GNURadio Graphen

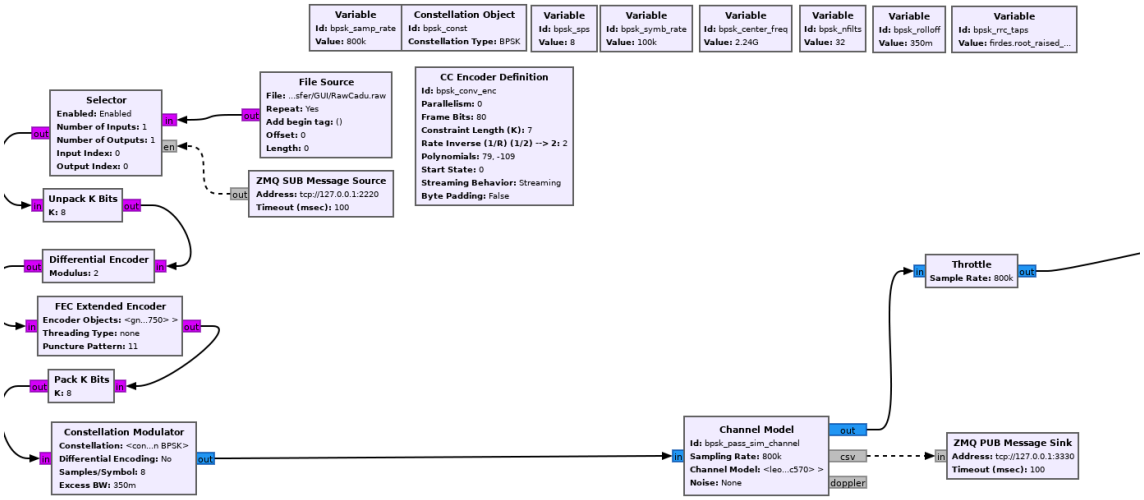

Abbildung A.1.: GNURadio Graph Variablen Downlink und Sendepfad BPSK Downlink

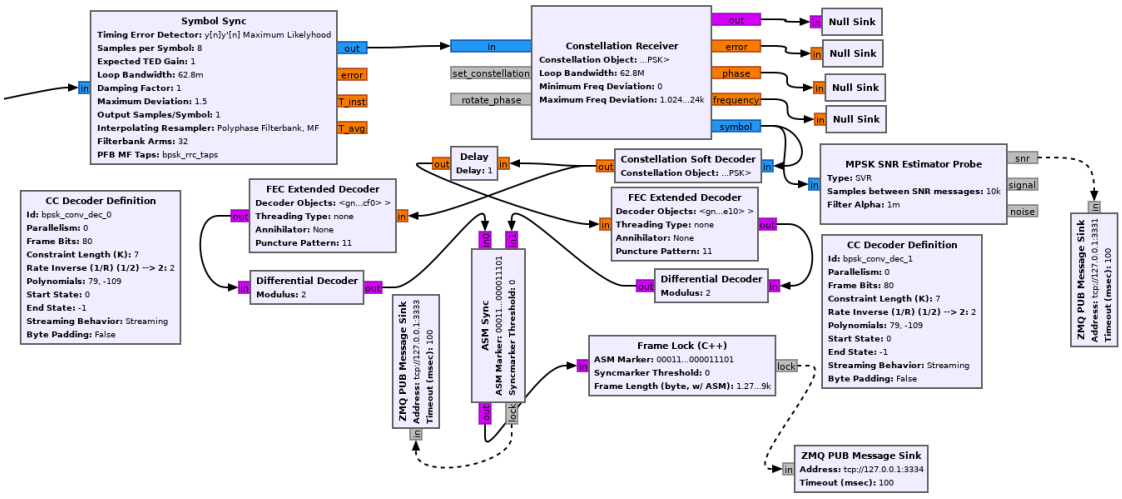

Abbildung A.2.: GNURadio Graph Empfangspfad BPSK Downlink

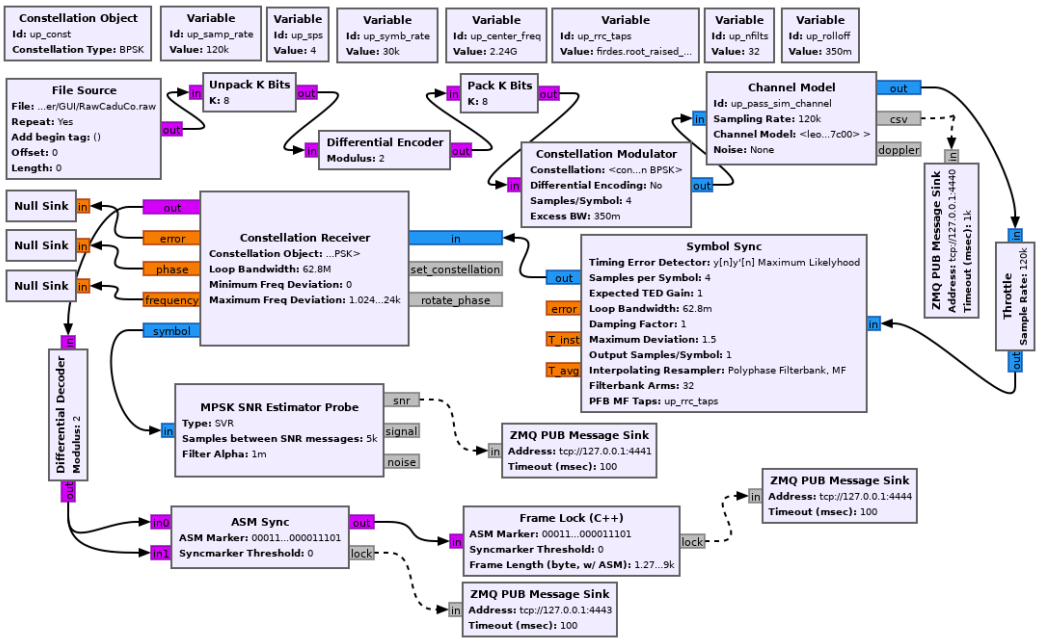

Abbildung A.3.: GNURadio Graph Sende- und Empfangspfad Uplink

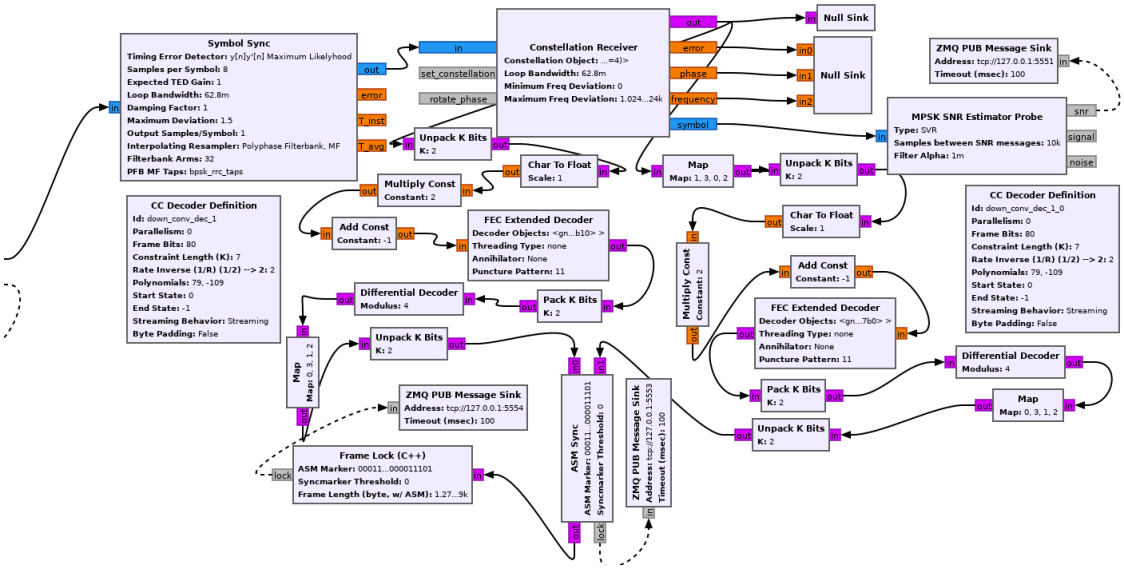

Abbildung A.4.: GNURadio Graph Empfangspfad QPSK Downlink

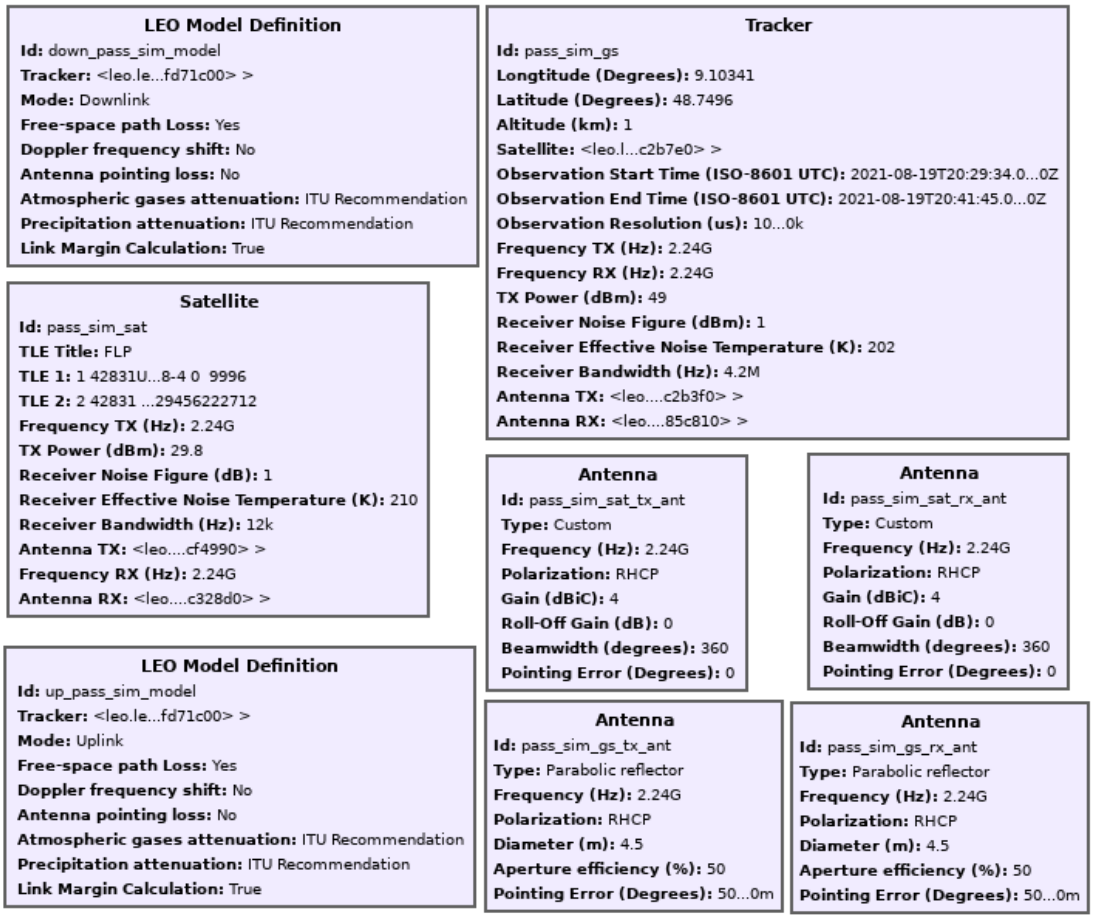

Abbildung A.5.: GNURadio Parameter Überflugssimulation

1 42831U 17042G 21229.01570423 -.00000289 00000-0 -23718-4 0 9996
2 42831 97.4630 95.5502 0012795 310.4637 49.5463 14.91329456222712

Abbildung A.6.: Benutztes TLE Flying Laptop