Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Hagen Hasberg

Entwicklung einer dynamischen adaptiven Flugsteuerung auf Basis künstlicher neuronaler Netze und lernender Agenten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hagen Hasberg

Entwicklung einer dynamischen adaptiven Flugsteuerung auf Basis künstlicher neuronaler Netze und lernender Agenten

Masterthesis eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Lehmann
Zweitgutachter: Prof. Dr. Andreas Meisel

Eingereicht am: 30. Mai 2018

Hagen Hasberg

Thema der Arbeit

Entwicklung einer dynamischen adaptiven Flugsteuerung auf Basis künstlicher neuronaler Netze und lernender Agenten

Stichworte

adaptive Flugsteuerung, Flugregelung, Autopilot, künstliche neuronale Netze, rekurrente Netze, Systemidentifikation, Flugsimulation, Ausfallerkennung, Control Allocation

Kurzzusammenfassung

Ein notwendiger Aspekt sicherheitskritischer Systeme ist eine Toleranz gegenüber Teilausfällen. In dieser Thesis wird eine fehlertolerante Flugstuerung entwickelt, welche den Ausfall von Steuerflächen durch eine dynamische Reallokation der benötigten Steuerleistung auf die intakte Aktorik kompensiert. Zur Ausfallerkennung und Abschätzung der verbliebenen Integritäten werden rekurrente neuronale Netze eingesetzt. Für die adaptive Steuerflächenzuordnung wird ein Allokationsalgorithmus entwickelt, welcher die Verteilungsstrategie für die vom Flugregler geforderte Steuerleistung im Fehlerfall dynamisch anlernt. Der Ansatz wird in einem simulierten Regelkreis evaluiert. Es wird eine Arbeitsumgebung geschaffen, mit welcher sich der präsentierte Ansatz auf andere Trägersysteme übertragen lässt.

Hagen Hasberg

Title of the paper

Development of a dynamic adaptive flight control system based on artificial neural networks and learning agents

Keywords

adaptive flight control, automatic flight control, autopilot, artificial neural networks, recurrent nets, system identification, flight simulation, fault detection, control allocation

Abstract

A necessary aspect of safety-critical systems is a tolerance to partial failures. In this thesis, a fault-tolerant flight control system is developed, which compensates for failures of control surfaces by a dynamic redistribution of the required control effort among the intact actuators. Recurrent neural networks are used for failure detection and estimation of the remaining actuator integrity. An algorithm for adaptive control allocation is developed, which learns a dynamic reallocation control scheme online in case of a failure. The approach is evaluated in a simulated control loop. A developement environment is created, with which the presented approach can be transferred to other carrier systems.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Eine Eigenschaft realer Systeme ist die Tatsache, dass diese anfällig für Fehler, Fehlfunktionen und unerwartete Verhaltensweisen sind. Für sicherheitskritische Systeme besteht ein Bedarf an zuverlässigen und fehlertoleranten Steuersystemen. Für Luftfahrzeuge ist eine Fehlertoleranz von großer Relevanz. Ein Verlust der Steuerbarkeit kann katastrophale Folgen für Mensch, Maschine und Umwelt zur Folge haben. Unter Ausnutzung intrinsischer Redundanzen der vorhandenen Steuerflächen kann eine adaptive Flugsteuerung die Steuerbarkeit trotz Teilausfällen der Aktorik in allen Freiheitsgraden gewährleisten, so dass eine sichere Notlandung durchgeführt werden kann. Eine adaptive Flugsteuerung erhöht somit die Sicherheit des Flugzeugs und kann einen möglichen Schaden reduzieren. Besonders für unbemannte Luftfahrzeuge (*UAV*) ist eine solche Fehlertoleranz des Steuersystems von Vorteil, da ein Ausfall nicht von einem menschlichen Piloten und seiner Intuition und Erfahrung kompensiert werden kann. Durch den erweiterten Funktionsumfang erhöht sich die Systemsicherheit, da die Wahrscheinlichkeit eines Absturzes minimiert wird.

Viele Flugzeuge sind hinsichtlich ihrer Aktorik inhärent überaktuiert. Bei diesen sind mehr Steuerflächen vorhanden, als Freiheitsgrade gesteuert werden müssen. Zusätzlich sind die von den Steuerflächen erzeugten Bewegungsformen typischerweise gekoppelt. Der Ausschlag eines Ruders bewirkt Bewegungen in allen Freiheitsgraden. Die Überaktuierung kann so ausgenutzt werden, dass trotz des Ausfalls einer Steuerfläche die Steuerbarkeit in allen rotatorischen Freiheitsgraden in gewissem Maße erhalten bleibt und das Flugzeug mit reduzierter Performance weiterfliegen kann. Eine notwendige Bedingung hierfür ist, dass nicht nur einzelne, sondern alle Freiheitsgrade überaktuiert sind.

Um die Überaktuierung ausnutzen zu können, muss die Flugsteuerung adaptiv ausgelegt sein. Sie muss den Ausfall einer Steuerfläche erkennen und die verlorene Steuerleistung dynamisch auf die intakte Aktorik verteilen. Durch diese Reallokation der benötigten Steuerleistung ist es möglich, einen Ausfall zu kompensieren. So kann etwa ein nötiges Giermoment bei ausgefallenem Seitenruder von differentiell angesteuerten Triebwerken erzeugt werden. Steigrate und Höhe können wahlweise durch die Erzeugung eines Nickmoments der Höhenruder oder durch eine Variation des Schubs verändert werden, da aus mehr Fahrt mehr Auftrieb resultiert. Dieser

Kompensation sind jedoch Grenzen gesetzt. Wenn etwa die Triebwerke und damit die Erzeugung des Schubs ausfallen, ist lediglich die zum Zeitpunkt des Ausfalls restliche potenzielle und kinetische Energie für den Weiterflug vorhanden. Weiterhin variiert die Effektivität der einzelnen Steuerflächen für die verschiedenen Freiheitsgrade, da diese für ihre speziellen Aufgaben ausgelegt und an ihre eigene Rotationsachse gebunden sind. Das *Seitenruder* ist primär für die Erzeugung des *Giermoments*, die *Querruder* für das *Rollmoment* und die *Höhenruder* für das *Nickmoment* zuständig.

Das in der vorliegenden Masterthesis entwickelte System soll die vollständige Manövrierfähigkeit bei Teilausfällen der Aktorik gewährleisten und so einen sicheren Weiterflug ermöglichen. Inspiriert wurde die Idee zur Durchführung von verschiedenen Flugzeugunglücken. So etwa von Flug 232 der United Airlines am 19. Juli 1989 [National Transportation Safety Board (1990)], bei welchem die gesamte Hydraulik infolge einer Explosion in einem der drei Triebwerke ausgefallen war. Den Piloten standen nach der Explosion lediglich die zwei intakten Triebwerke unterhalb der beiden Tragflächen der *McDonnell Douglas DC-10-10* zur sicheren Landung zur Verfügung. Nach wenigen Minuten Eingewöhnungszeit hatten diese sich eingeprägt, wie sie das Flugzeug allein mit den Triebwerke steuern und so den Rest an Manövrierfähigkeit ausnutzen konnten. Mit der Änderung der Fluggeschwindigkeit konnte der Auftrieb variiert werden, wodurch sich die Flughöhe steuern ließ. Durch eine differentielle Ansteuerung der Triebwerke konnte eine Kurve eingeleitet werden. Mit diesen Steuermöglichkeiten konnte das Flugzeug noch zu einer Landebahn navigiert und der Versuch einer kontrollierten Notlandung durchgeführt werden. Zwar kam es dennoch zu einer Bruchlandung, in der das Flugzeug in mehrere Teile zerbrach, jedoch konnte durch die beispiellose Leistung der Piloten 185 der 296 sich an Board befindlichen Menschenleben gerettet werden.

Dieses Unglück lieferte die Inspiration für die Entwicklung eines Systems, welches die Piloten in dieser Situation unterstützt hätte. [Smaili u. a. (2009)] beschreibt einige Flugzeugunglücke, bei denen Experten davon ausgehen, dass eine fehlertolerante Flugsteuerung das Unglück hätte Verhindern oder deren Konsequenzen minimieren können.

1.1. Projektumfeld

Im Jahr 2001 wurde die Projektgruppe *Blended Wing Body* (*BWB*) an der HAW Hamburg gegründet. Seitdem befassen sich Studierende aus dem Department Fahrzeugtechnik und Flugzeugbau mit der Erforschung von unkonventionellen BWB-Flugzeugkonfigurationen [Reimann (2004), Danke (2005)], welche sich langfristig als neuer Standard für Großraumflugzeuge etablieren könnten.

Für die von der BWB-Projektgruppe genutzten Versuchsträger sollte ein neues Messsystem inklusive autonomer Flugsteuerung (*Flight Control Unit*, *FCU*) vom Department Informatik entwickelt werden. Hierfür wurde 2013 das Projekt *Airborne Embedded Systems* (*AES*) gestartet. Die AES-Projektgruppe beschäftigt sich mit der Entwicklung und der Fertigung der FCU, sowie der dafür erforderlichen Systeme, um mit diesen ein UAV autonom zu steuern.

Neben den Zielen der BWB-Projektgruppe, welche das Messen und die Analyse von aerodynamischen Kenndaten der Versuchsträger beinhalten, soll die FCU auch als vom Träger unabhängiges Autopilotensystem agieren können. Es wird unter Anderem ein Einsatz für den Katastrophenschutz angestrebt. Weiterhin wird eine mögliche Zulassung für Flüge in Deutschland beabsichtigt. Daraus ergeben sich für das Autopilotensystem und jeweiligen Träger die Kernanforderungen Sicherheit und Zuverlässigkeit.

Unabhängig von den beiden Projektgruppen wurde 2014 die *Interdisziplinäre Forschungsgruppe Optronik und Avionik* (*OPAV*) an der HAW gegründet. Mit dieser wird in enger Zusammenarbeit weiter an der FCU und dem Simulations- und Testsystem [Hasberg (2014)] entwickelt. Die (Teil-) Systeme, welche im Rahmen von AES entstanden sind, kommen in vielen Projekten von OPAV zum Einsatz. Hierbei wird die FCU etwa mit einem Antikollisionssystem ausgestattet, der Flugregler auf einem FPGA instanziiert oder etwa die FCU redundant betrieben.

Die Masterthesis entsteht im Kontext von AES und behandelt die Entwicklung einer adaptiven Flugsteuerung für das Autopilotensystem, welches die Sicherheit und Zuverlässigkeit der FCU erhöht.

1.2. Motivation

Die FCU soll in verschiedenen Szenarien und Umgebungen eingesetzt werden und stellt damit ein universell einsetzbares Autopilotensystem dar. Für die verschiedenen Einsatzgebiete existieren jeweils eigene Anforderungen an das System. Jedoch sind folgende Aspekte für alle Einsatzgebiete gemein:

- Ausfallsicherheit und Zuverlässigkeit
- Flexibilität und Energieeffizienz
- Miniaturisierung und Gewichtsreduktion
- Hohe Verfügbarkeit durch geringe Wartungsintervalle
- Kostenreduktion

Die adaptive Flugsteuerung soll die Absturzwahrscheinlichkeit verringern. Wenn ein Absturz nicht verhindert werden kann, soll das UAV die negativen Auswirkungen auf die Drohne selbst und die Umgebung minimieren. Dies soll dadurch erreicht werden, dass bei einem (Teil-) Ausfall der Aktorik der Träger dennoch in gewissem Maße manövrierfähig bleibt. So kann bei einem Ausfall eine geeignete Lande- oder Absturzzone angeflogen werden, was weniger negative Auswirkungen auf die Umwelt oder den Träger selbst hat. Sobald die Drohne aus der kritischen Umgebung, wie etwa bewohntes Gebiet oder Wasserflächen, manövriert wurde, kann dort eine Landung oder, soweit nötig, ein kontrollierter Absturz eingeleitet werden.

Die stetige Manövrierbarkeit ist besonders dann wichtig, wenn eine Gefahr für Personen besteht, etwa wenn über urbanes Gebiet geflogen wird. Diese Gefahr zu minimieren, ist die grundlegende Motivation der vorliegenden Arbeit. Weiterhin führt eine geringere Absturzrate zu einer Reduktion von Kosten im Allgemeinen. So muss die Drohne zum Beispiel seltener kompliziert geborgen werden. Dies führt wiederum zu einem Zeitersparnis für die Betreiber. Neben dem Schutz der Umwelt kann eine adaptive Flugsteuerung auch den Verlust des Trägers selbst verhindern. Mit einer fehlertoleranten Flugsteuerung gewinnen UAVs weiter an Autonomie, wodurch in der Zukunft womöglich auf Sicherheitspiloten verzichtet werden kann.

Für das Messen von aerodynamischen Kenndaten eines Flugzeugs sind unter Anderem Struktur-belastende Manöver, wie beispielsweise *Trudeln* oder ein *Männchen*, zu fliegen [Danke (2005)]. Diese beanspruchen neben der Struktur des Flugzeugs auch die Servos, welche die Steuerflächen bewegen, zu einem hohen Grad. Um Kosten zu sparen, werden hierfür meist handelsübliche Modellbauservos anstelle von für industrielle Nutzung ausgelegte Modelle verwendet. Hieraus resultiert eine meist geringere Qualität und Haltbarkeit. Daher besteht für das Szenario des Messens der aerodynamischen Kenndaten ein erhöhtes Risiko, dass die Steuerflächen aufgrund von defekten Servos zu Teilen ausfallen.

Maskieren des Ausfalls

Ein Pilot verwendet auf seiner Funkfernsteuerung typischerweise je einen der vier Steuerknüppel für je einen Freiheitsgrad. Er benötigt kein oder wenig Wissen darüber, welche Steuerflächen für die Erzeugung eines gewünschten Drehmoments im angesteuerten Freiheitsgrad verwendet werden. Außerdem hat er auch nicht die Möglichkeit, einzelne Steuerflächen anzusteuern, sondern stets nur als Gruppe, welche den Freiheitsgrad beeinflussen sollen.

Beim Ausfall einer Steuerfläche oder eines Freiheitsgrades könnte er versuchen, das Flugzeug durch Bewegungen in den anderen Freiheitsgraden zu manövrieren. Dies gestaltet sich als schwierig bis unmöglich, da er typischerweise keine Erfahrung mit einer alternativen Strategie zur Steuerung hat. Aus diesem Grund ist es wichtig, dass die Steuerknüppel seiner Funkfern-

steuerung zu jeder Zeit die gleichen Freiheitsgrade steuern. Der Ausfall muss vor dem Piloten maskiert werden, um ein sicheres Weiterfliegen ermöglichen zu können.

Zwar wird die Performance der Manövrierbarkeit durch eine Reallokation gemindert, jedoch muss der Pilot in seiner Steuerung nicht *umdenken* und ad-hoc eine neue Steuerknüppel-Konfiguration lernen, bzw. hierfür nicht geschult sein. Gleiches gilt für die Steuerung durch den Autopiloten. Dafür ist es womöglich erforderlich, die Reglerparameter anzupassen, nicht jedoch die Reglerstrukturen.

Durch den Einsatz einer adaptiven Flugsteuerung kann ein Ausfall vor dem Piloten und dem Autopiloten maskiert werden, da das Systeminterface von seinen Wirkprinzipien her gleich bleibt.

1.3. Zielsetzung

Als Ergebnis der Masterthesis soll eine prototypische Umsetzung einer adaptiven Flugsteuerung vorliegen, welche den Ausfall einer Steuerfläche durch die dynamische Umverteilung der Steuerleistung auf die intakte Aktorik kompensiert. Diese soll die FCU erweitern, um die Sicherheit des autonomen Flugs zu erhöhen. Das Ergebnis soll mit geringem Aufwand auf verschiedene Trägersysteme übertragen werden können. Somit ist ein generischer Ansatz zu entwerfen, mit welchem auf eine komplexe Modellbildung des Trägersystems verzichtet werden kann. Neben der prototypischen Umsetzung ist eine Arbeitsumgebung zu schaffen, um Dritten die Übertragung ohne Modellbildung zu ermöglichen.

Das System soll zunächst für einen Träger entwickelt und getestet werden. Die Ergebnisse verschiedener Experimente sollen Aufschluss darüber geben, ob die Umsetzung einen generischen Ansatz darstellt und die entwickelte adaptive Flugsteuerung mit geringem Aufwand auf andere Trägersysteme übertragen werden kann. Zunächst soll das System einen Träger aus einer PC-Flugsimulator Software in einem simulierten Regelkreis steuern.

Das System muss die folgenden drei Hauptfunktionalitäten unterstützen:

- Steuern im fehlerfreien Zustand
- Erkennen eines Ausfalls
- Kompensieren des Ausfalls durch Reallokation der benötigten Steuerleistung auf die intakten Steuerflächen

Beim beschriebenen Flugzeugunglück kannten die Piloten das neu entstandene Systemverhalten zunächst nicht. Durch reines Ausprobieren und Intuition konnten sie ad-hoc eine neue Steuerstrategie zur weiteren Manövrierung des Flugzeugs ableiten. Inspiriert durch die

erzielte Leistung dieses Vorgehens, soll in dieser Arbeit zunächst evaluiert werden, wie so ein Vorgehen als generischer Algorithmus implementiert werden kann. Die adaptive Flugsteuerung soll aufgrund dessen nach Möglichkeit nicht auf die Vorgabe des Systemverhaltens oder Expertenwissen angewiesen sein. Dies bedingt, dass keine feste Zuordnungstabelle, ein statisches Fehlermodell oder anderes vorgegebenes Regelwerk implementiert ist, wie die benötigte Steuerleistung im Fehlerfall zu verteilen ist. Die nötige Verteilung im Fehlerfall ist demnach erst nach Auftreten des Ausfalls dynamisch zu entscheiden.

Die nachfolgende Auflistung fasst die selbst definierten Anforderungen zusammen:

- Das System soll einen Ausfall maskieren, so dass ein Pilot nicht umdenken oder der Autopilot nicht umprogrammiert werden muss, wenn eine Reallokation der Steuerflächen stattgefunden hat.
- Das System soll leicht auf einen anderen Träger übertragen werden können. Das Übertragen darf keine tiefgreifenden Kenntnisse aus der Flugmechanik oder Regelungstheorie erfordern. Demnach ist auf eine umfassende analytische Beschreibung des Flugverhaltens zu verzichten.
- Das System soll auf einen Träger mit beliebiger und teils exotischer Steuerflächenkonfiguration übertragen werden können.
- Es sollen beim Übertragen keine Programmanpassungen, welche das System an den Träger bindet, nötig sein.
- Die Festlegung der Reallokation soll erst nach dem Ausfall stattfinden.
- Die Algorithmen sollen auf ein eingebettetes System übertragbar sein.

Die Abbildung 1.1 zeigt das grobe Konzept des zu entwickelnden Systems als Regelkreis. Die drei Hauptfunktionalitäten werden auf zwei logische Komponenten aufgeteilt. Die Komponente *Ausfallerkennung* (engl. *Fault Detection*) soll den Ausfall einer Steuerfläche erkennen und deren verbliebene Steuereffektivität abschätzen. Die Steuerflächenzuordnung (engl. *Control Allocation*) soll die Steuerung des Flugzeugs im fehlerfreien und im Ausfallzustand durchführen. Die adaptive Flugsteuerung ist dem Autopiloten nachgelagert und setzt die von ihm vorgegebenen Bewegungen im dreidimensionalen Raum um. Im Sinne eines Regelkreises ist die Control Allocation eine Abbildung von geforderten Drehmomenten auf Steuereingaben für die am Flugzeug vorhandenen Steuerflächen, mit welchen die Drehmomente optimal erzeugt werden. Das System soll dabei im Ausfallzustand selbstständig lernen, welche Abbildung von Freiheitsgrad auf Steuerflächen geeignet ist, um eine weitere Manövrierung des Flugzeugs zu ermöglichen.

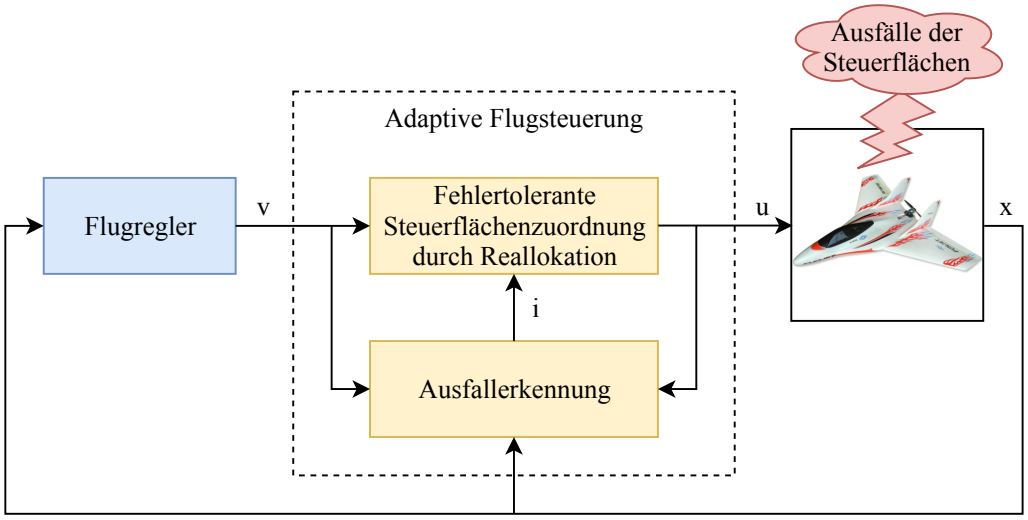

Abbildung 1.1.: Konzept der adaptiven Flugsteuerung mit Komponenten Steuerflächenzuordnung und Ausfallerkennung

Zur Erfüllung der Ziele ist in dieser Arbeit zunächst eine umfassende Analyse über mögliche Techniken und Konzepte durchzuführen. Daraufhin wird ein Ansatz abgeleitet und präsentiert, wie die adaptive Flugsteuerung umgesetzt werden kann. Es sind die Teilsysteme Ausfallerkennung und Steuerflächenzuordnung zu entwickeln. Die entwickelten Komponenten sind zu evaluieren. In vorangegangenen Projektarbeiten wurden bereits Teilkomponenten des Systems und des simulierten Regelkreises entworfen und implementiert. Das Flugverhalten soll von künstlichen neuronalen Netzen nachgebildet werden, um aus einer Differenz zwischen erwartetem und tatsächlichem Verhalten einen Ausfall zu folgern. In [Hasberg (2017a)] wurde eine erste Exploration dieser durchgeführt und Teile der zu entwickelnden Arbeitsumgebung zur Übertragung geschaffen. In [Hasberg (2017b)] wurde die zur Umsetzung und Evaluierung des Systems zu verwendende Simulationsinfrastruktur dahingehend erweitert, dass für die Steuerflächen des simulierten Trägers ein Ausfall ausgelöst und behandelt werden kann.

1.4. Gliederung

Die vorliegende Thesis umfasst insgesamt neun Kapitel. Der Leser wird systematisch entlang des roten Fadens durch die vom Autor durchgeführten Arbeitsschritte begleitet.

Kapitel 1 gibt eine Einführung in das Thema und beschreibt das Projektumfeld, die Motivation sowie die Zielsetzung dieser Arbeit.

Kapitel 2 liefert dem Leser ein Grundwissen über verschiedene Aspekte der Aufgabenstellung, so dass die einzelnen Arbeitsschritte nachvollzogen werden können.

Kapitel 3 gibt eine Übersicht verschiedener Verfahren zur Ausfallerkennung und adaptiven Steuerflächenzuordnung. Weiterhin werden vergleichbare Gesamtsysteme vorgestellt.

Kapitel 4 erläutert den eigenen Ansatz zur Realisierung des Gesamtsystems adaptive Flugsteuerung.

Kapitel 5 stellt die Systemarchitektur des simulierten Regelkreises vor. Es wird die zur Umsetzung notwendige Erweiterung der vorhandenen Simulationsinfrastruktur vorgestellt.

Kapitel 6 beschreibt die Realisierung des Teilsystems Ausfallerkennung. Es wird die umgesetzte Arbeitsumgebung zur Entwicklung der neuronalen Netze sowie deren Instanziierung im simulierten Regelkreis vorgestellt.

Kapitel 7 beschreibt die Realisierung des Teilsystems adaptive Steuerflächenzuordnung. Es wird das dynamische Anlernen des Systemverhaltens sowie der entwickelte Algorithmus zur Allokation der Steuerflächen vorgestellt.

Kapitel 8 zeigt eine Evaluierung des umgesetzten Systems. Zunächst werden die einzelnen Teilkomponenten in Isolation evaluiert und die Ergebnisse diskutiert. Daraufhin folgt ein Test des Gesamtsystems.

Kapitel 9 fasst die durchgeführten Arbeitsschritte zusammen. Zum Abschluss wird ein Fazit über die Ergebnisse gezogen und im Zuge dessen ein Ausblick über mögliche Weiterentwicklungen gegeben.

2. Grundlagen

Im Folgenden werden einige Grundlagen der verwendeten Begriffe und Konzepte, die in der vorliegenden Thesis Anwendung fanden, beschrieben. Diese Grundlagen sollen dem Leser eine solide Basis für das Verständnis der noch folgenden Kapitel geben.

2.1. Flugregelung und -steuerung

Dieser Abschnitt soll eine kurze Einführung in die Begriffe *Flugregelung* und *Flugsteuerung* geben. Bei der Flugregelung handelt es sich um die Disziplin, die Regelungstechnik im Flugwesen einzusetzen. Typische Aufgaben von Flugreglern gehen von einfachen, wie etwa den Kurs, die Fluglage oder die Fluggeschwindigkeit auf einem konstanten Sollwert zu halten, bis hin zu komplexeren Aufgaben, wie etwa die autonome Landung oder Navigation.

Begriffe

Als Nomenklatur im Zusammenhang mit der Flugregelung sollen in dieser Arbeit die folgenden Begrifflichkeiten gelten:

- **Flugregelung**: Disziplin der Regelungstechnik, die sich mit der Regelung von Fluggeräten beschäftigt.
- **Flugregler**: Das mathematische Modell des Reglers. Typischerweise aus mehreren kaskadierten Einzelreglern zusammengesetzt.
- **Autopilot**: Die konkrete Umsetzung des Flugreglers in einem oder mehreren vernetzen Computersystemen.
- **Avionik, Avioniksystem**: Bezeichnet die Gesamtheit aller elektrischen und elektronischen Systeme zur Steuerung eines Fluggeräts.
- **Flugzustand**: Bezeichnet die Gesamtheit aller zur Verfügung stehenden Werte, die den Zustand Flugzeugs in seinem Flug beschreiben.
- **Steuerfläche / Aktuator**: Aerodynamisches Stellglied des Flugzeugs. Diese Umfassen alle Ruder und die Triebwerke

- **Flugsteuerung**: Dem Flugregler nachgelagertes System zur Ansteuerung der Steuerflächen.
- **Steuerflächenzuordnung / Control Allocation**: Abbildung von Freiheitsgrad auf Steuerflächen.
- **P/Q/R/A**: Formelzeichen für Rollrate, Nickrate, Gierrate und Beschleunigung in Richtung der Längsachse

Struktur eines einfachen Flugreglers

Die Abbildung 2.1 zeigt exemplarisch einen Regelkreis, der ein Flugzeug auf einem gewünschten Soll-Kurs halten kann. Analoge Reglerkaskaden werden auch für die Geschwindigkeits- oder Höhenregelung eingesetzt.

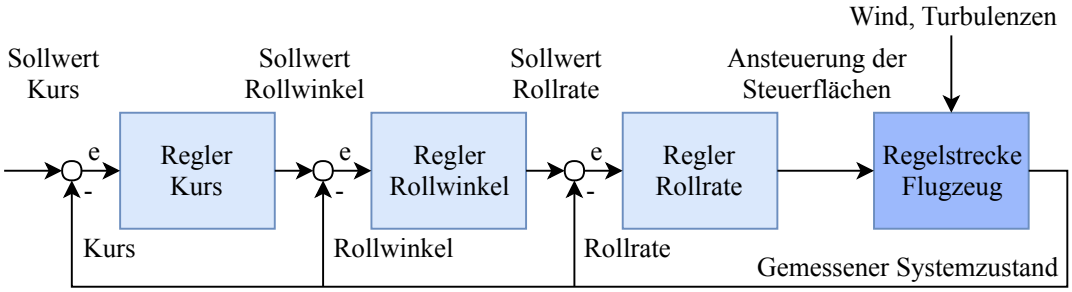

Abbildung 2.1.: Kaskadierter Flugregler zum Regeln des Kurses

In dieser Abbildung stellt das Flugzeug die Regelstrecke dar. Als (Gesamt-) Regler werden drei kaskadierte einzelne Regler verwendet. Der erste passt den tatsächlichen Kurs an den vom Piloten eingestellten Soll-Kurs an. Die Ausgabe ist die Vorgabe des Soll-Rollwinkels für den nächst-inneren Regler. Dessen Ausgabe ist wiederum die Vorgabe der Soll-Rollrate für den hintersten Regler. Dessen Ausgabe ist je nach Aufbau des Avioniksystems ein Ausschlag des Stellglieds (Steuerfläche) oder ein normierter Steuerwert, welcher an eine nachgelagerte Flugsteuerung weitergereicht wird.

Die Rückführung des *Flugzustands* kann auf verschiedene Arten erfolgen. Viele Werte können von im Flugzeug verbauten Sensoren gemessen werden. Nicht sensorisch erfassbare Werte können in einigen Fällen durch *Beobachter*-Systeme *geschätzt* oder *beobachtet* werden. Auf solche Zustandsbeobachter wird in Abschnitt 3.1.2 eingegangen.

Guidance, Navigation and Control (GNC)

Komplexe Autopiloten bestehen aus mehreren Komponenten und Systemen, die ihren Aufgaben nach, in die Kategorien *Guidance*, *Navigation* und *Control (GNC)* eingeteilt werden. Diese bilden eine kaskadierte Kette an Systemen. Der oben beschriebene Regler ist ein Teil der Kategorie Guidance, da er das Flugzeug entlang einer gewünschten Trajektorie *führt*. Die Sollwerte der Trajektorie werden vom vorgelagert System Navigation oder dem Piloten selbst geliefert. Ein Soll-Kurs ergibt sich etwa aus dem Wunsch, von der aktuellen Position zu einem Wegpunkt zu fliegen. Die dritte Komponente Control erhält die Sollwerte für die tatsächliche Steuerung des Flugzeugs. Diese beinhalten eine Herbeiführung der Änderung der Fluglage oder Fluggeschwindigkeit mit Hilfe der Steuerflächen. Grundsätzlich soll die Komponente Control das reale Flugzeug und dessen Steuerflächen von der Komponente Guidance abstrahieren. Die Aufgaben der Komponente Control werden in dieser Arbeit unter dem Begriff *Flugsteuerung* oder *Control Allocation* zusammengefasst. In Abschnitt 3.2 wird diese Aufteilung der Systeme motiviert und detailliert beleuchtet.

Für ausführlichere Informationen sei auf den *Brockhaus* [Brockhaus u. a. (2011)] verwiesen, welcher als das deutsche Standardwerk zum Thema Flugregelung angesehen werden kann. Aus der englischen Literatur wird McLean [McLean (1990)] empfohlen.

2.2. Modellbildung Flugzeug

Die geschlossene Steuerung von Maschinen wird typischerweise in Regelkreisen modelliert und umgesetzt. Hierbei stellt das zu steuernde technische System die Regelstrecke und die elektronische Steuereinheit den Regler dar. Im Kontext der Flugregelung ist der Autopilot (FCU) der *Regler* und ein UAV-Trägersystem die *Regelstrecke*. Die Abbildung 2.2 zeigt einen einfachen Regelkreis im Kontext der Flugregelung.

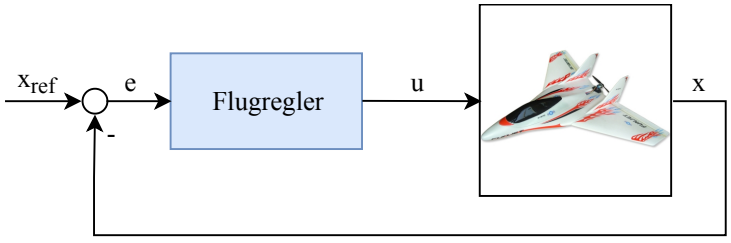

Abbildung 2.2.: Einfacher Regelkreis im Kontext der Flugregelung

Die Variable x stellt den Systemzustand dar. Im Kontext der Flugregelung ist dies der Flugzustand. Er enthält Daten wie Geschwindigkeit, Lage oder Höhe. Der zu regelnde Wert wird durch die Führungsgröße x_{ref} angegeben. Die Regelabweichung e wird aus der Differenz zwischen Systemzustand und Führungsgröße berechnet. Der Regler versucht diese Abweichung mit der Zeit zu minimieren. Hierfür berechnet er eine mehrdimensionale Stellgröße u, welche bei Flugzeugen den Ansteuerungen der Steuerflächen entsprechen. Diese werden entsprechend der Werte von u ausgelenkt, bzw. angesteuert.

Der Systemzustand wird von Sensoren oder Zustandsbeobachtern erfasst. Die Stellgrößen werden von Aktoren umgesetzt. Der Regler wird in aktuellen Systemen typischerweise durch eine elektronische Steuereinheit auf Basis eines eingebetteten Mikroprozessors umgesetzt. Zu den Aufgaben des eingebetteten Systems gehören das Einlesen der Sensoren, das Berechnen der Regelabweichung und einer geeigneten Stellgröße sowie das Ansteuern der Aktorik durch diese Stellgröße. Diese Schleife (*Regelschleife*) wird mehrere Male pro Sekunde durchlaufen. Die Frequenz der Schleife wird von der Dynamik des zu regelnden Systems vorgegeben. So benötigen etwa akrobatische Flugzeuge höhere Frequenzen als schwerfällige Linienflugzeuge.

Zur optimalen Auslegung der Regelung, bzw. der Steuerung muss das Systemverhalten des Flugzeugs bekannt sein. Hierfür wird die *Übertragungsfunktion* als mathematisches Modell gebildet. Das Bilden der Übertragungsfunktion eines hochkomplexen Systems wie das eines Flugzeugs gilt als nicht trivial. Die Bewegung im dreidimensionalen Raum wird durch viele Einflussfaktoren der *Flugmechanik* bestimmt. Dabei spielen Flugzeugmerkmale wie Masse, Geometrie, Aeroelastizität, Massenträgheit und Trägheitsmoment, Steuerflächengröße, -anordnung und -dynamik, Triebwerksleistung, Flügelprofile und einige weitere eine Rolle (vgl. [Hasberg (2014)]).

Bilden der linearisierten Bewegungsgleichungen

Im dreidimensionalen Raum besitzt ein Flugzeug drei rotatorische und drei translatorische Freiheitsgrade. Auf das Flugzeug wirken die vier Kräfte Auftrieb, Widerstand, Gewicht und Schub [Grundmann (2007)]. Dabei ist die Bewegung "*durch das Wechselspiel zwischen Kräften und Momenten, die infolge der Massenträgheit sowie infolge der Aerodynamik, des Antriebs des Gewichts und der Bodenreaktion auftreten*"[Nguewo (2014)] bestimmt. Die Flugzeugbewegung vollständig in einer Übertragungsfunktion zu beschreiben ist sehr komplex und eine analytische Lösung ist nicht möglich. Je nach von der Aufgabenstellung benötigtem Realitätsgrad kann sich hierfür auf Teilmodelle beschränkt werden (vgl. [Nguewo (2014)]).

Zur Vereinfachung des mathematischen Modells werden typischerweise zwei voneinander unabhängige *Bewegungsgleichungen* gebildet [McLean (1990)]. Die *Seitenbewegung* beschreibt

das Flugverhalten entlang der Querachse, die *Längsbewegung* entlang der Längsachse und der Hochachse. Beide Bewegungsgleichungen sind nichtlinear und mehrdimensional. Zur weiteren Vereinfachung werden sie linearisiert und in der *Zustandsraumdarstellung* formuliert. Eine *Zustandsgleichung* wird durch die folgende allgemeine Gleichung beschrieben:

$$G(u) = \dot{\underline{x}}(t) = \underline{\underline{A}} * \underline{x}(t) + \underline{\underline{B}} * \underline{u}(t) + \underline{\underline{E}} * \underline{z}(t) \tag{2.2.1}$$

Die Elemente $\dot{x}$, x, u und z sind zeitabhängige Vektoren. Dabei enthält x die Zustände (*Zustandsvektor*), u die Eingänge (*Eingangsvektor* oder *Steuervektor*) und z die Störgrößen (*Störgrößenvektor*) des Systems. Die Elemente A (*Systemmatrix* oder *Zustandsmatrix*), B (*Steuermatrix* oder *Effektivitätsmatrix*) und E (*Störgrößenmatrix*) sind Matrizen, welche das Flugverhalten durch verschiedene Koeffizienten beschreiben. Der Folgezustand $\dot{x}$ ergibt sich aus dem aktuellen Zustand, der aktuellen Eingabe und der aktuell anliegenden Störung.

Die Form der Zustandsgleichung hängt von der für die Aufgabenstellung erforderlichen Güte des mathematischen Modells ab. Abbildung 2.3 zeigt eine beispielhaft Bewegungsgleichung der Längsbewegung. Der Zustandsvektor enthält die Variablen Fluggeschwindigkeit ΔV, Anstellwinkel $\Delta\alpha$, Nickrate Δq und Nickwinkel $\Delta\Theta$. Der Steuervektor die normierten Steuerwerte für die Höhenruder η, die Landeklappen η_K und die Triebwerke η_F. Der Störvektor die Windeinflüsse $\Delta\dot{u}$ und $\Delta\dot{w}$.

$$\begin{bmatrix} \Delta\dot{V} \\ \Delta\dot{\alpha} \\ \Delta\dot{q} \\ \Delta\dot{\Theta} \end{bmatrix} = \begin{bmatrix} X_V & X_\alpha & X_q & X_\Theta \\ Z_V & Z_\alpha & Z_q + 1 & Z_\Theta \\ M'_V & M'_\alpha & M'_q + M_{\dot{\alpha}} & M_{\dot{\alpha}} Z_\Theta \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta V \\ \Delta\alpha \\ \Delta q \\ \Delta\Theta \end{bmatrix} + \begin{bmatrix} X_\eta & X_{\eta_K} & X_{\eta_F} \\ Z_\eta & Z_{\eta_K} & Z_{\eta_F} \\ M'_\eta & M'_{\eta_K} & M'_{\eta_F} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \eta_K \\ \eta_F \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & -\frac{1}{V_r} \\ 0 & -M_{\dot{\alpha}} \frac{1}{V_r} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\dot{u}_{Ws} \\ \Delta\dot{w}_{Ws} \end{bmatrix}$$

Abbildung 2.3.: Aus [Nguewo (2014)]: Linearisierte Bewegungsgleichung für die Längsbewegung in der Zustandsraumdarstellung

Die Bewegungsgleichung aus Abbildung 2.3 beschreibt beispielhaft das Flugverhalten der Längsbewegung. Der erste Summand berechnet das Systemverhalten aufgrund des aktuellen Zustands ohne weitere Einflüsse. Der zweite Summand berechnet das Verhalten mit den momentan anliegenden Steuergrößen der Stellglieder. Der dritte Summand gibt das Systemverhalten aufgrund von atmosphärischen Werten wie Wind und Turbulenzen wider. Die Bewegungsgleichung beschreibt somit die Reaktion des Flugzeugs abhängig vom Zustand, den Klappenstellungen und dem Wind. Die Zustandsgleichung kann als Antwort folgender Frage aufgefasst werden: "*Wie verhält sich das Flugzeug zum Zeitpunkt t, bei den Ruderstellun-*

gen und Triebwerksschub u, dem Zustand x und den Windeinflüssen z?"[Hasberg (2014)]. Solch eine Bewegungsgleichung wird etwa von Flugsimulatoren verwendet, um das Verhalten von Flugzeugen nachzubilden.

Die Koeffizienten der Matrizen A, B und E werden aus den oben genannten flugmechanischen Eigenschaften des Flugzeugs abgeleitet. Aus den Eigenschaften, wie etwa Masse, den Geometrien, den aerodynamischen Beiwerten oder den Eigenschaften der Steuerflächen ergeben sich *Ersatzgrößen* (auch *Derivativa* genannt), welche die Koeffizienten der Matrizen widerspiegeln. Die Derivativa können durch verschiedene Ansätze ermittelt werden. Zunächst kann eine Abschätzung anhand von bekannten Geometrien und Eigenschaften von ähnlichen Flugzeugen erfolgen. Daraufhin können die ermittelten Werte durch Windkanaltests verfeinert werden. Hierbei wird ein verkleinertes Modell in einem Windkanal verwendet. Die ermittelten Werte werden dann auf die Originalgröße hochskaliert (siehe [Nguewo (2007)]) und zur Verfeinerung der reinen Abschätzungen verwendet. Der finale Schritt besteht aus realen Flugversuchen (siehe [Danke (2005)]). Diese liefern die genauesten Werte, jedoch muss das Flugzeug bereits gebaut sein und unter realen Bedingungen fliegen. Die Derivativa des Flugzeugs werden in jedem Schritt sukzessive verfeinert. Hierdurch verbessert sich das mathematische Modell des Flugzeugs, da die Übertragungsfunktion das Systemverhalten besser widerspiegelt. Bei ausreichender Genauigkeit kann das Modell neben der Auslegung von Systemen zur Flugregelung und Flugsteuerung auch in Flugsimulatoren zur Ausbildung von Piloten eingesetzt werden.

Für die mathematische Modellbildung des Systems Flugzeug eignen sich als Literatur die im vorangegangen Abschnitt empfohlenen Werke von Brockhaus [Brockhaus u. a. (2011)] und McLean [McLean (1990)].

2.3. Neuronale Netze

Künstliche neuronale Netze sind dem menschlichen neuronalen System nachempfunden. Hierbei wird versucht, ein mathematisches Modell aus biologischen neuronalen Netzen abzuleiten. Solch ein Modell kann angelernt werden, um Muster in Daten zu erkennen und vorhersagen zu treffen. Dieser Ansatz des *maschinellen Lernens* hat sich über die vergangenen Jahre bewährt, um ein lernendes System oder Programm auf einem Computersystem zu realisieren und konnte in der Vergangenheit in verschiedensten Disziplinen beachtliche Erfolge gegenüber klassischen Verfahren aus der künstlichen Intelligenz erzielen (vgl. [Karpathy (2015)] und [Heaton (2015)]). Nachfolgend bezeichnet der Begriff *neuronales Netz* stets ein *künstliches*. Dieser Abschnitt gibt

lediglich einen kurzen Einblick in dieses Themengebiet. Eine ausführliche Abhandlung ist in [Heaton (2015)] gegeben.

Neuronale Netze lassen sich generell in Netze zur *Klassifikation* und Netze zur *Regression* einteilen. Erstere werden verwendet, um eine Klassifikation *diskreter* Werte zu erreichen. Die Regression wird verwendet, um *kontinuierliche* Werte zu approximieren. Beim Anlernen des Flugverhaltens fallen ausschließlich kontinuierliche Werte an. Daher sind alle in dieser Arbeit verwendeten Netze in der Kategorie der Regression einzustufen. Weiterhin müssen Flugmanöver als Funktionsverläufe abgebildet werden, welche eine temporale Komponente besitzen. Deshalb müssen die Netze sequentielle Daten verarbeiten und ausgeben können. Bestehen die Anforderungen der Problemstellung aus der Kombination von regressiv und temporal, werden typischerweise *rekurrente* neuronale Netze eingesetzt.

2.3.1. Feed-Forward Netze

Ein neuronales Netz besteht aus einer Vielzahl untereinander vernetzter *Neuronen*. Die sich aus der Vernetzung der Neuronen ergebene Struktur bestimmt den Netztyp. Typischerweise werden die Neuronen in mehreren Schichten (*Layern*) angeordnet, wobei jede Schicht die verarbeiteten Informationen an die ihr nachfolgende weiter gibt. Diese Arten von Vernetzung ergeben ein *Feed-Forward Netz* (*FF*) oder *Multi-Layer Perzeptron* (*MLP*). Daten und Informationen fließen hier ausschließlich vom vordersten *Input-*, über einen oder mehrere *Hidden-* zum *Output*-Layer. Es entsteht ein gerichteter Graph, bei welchem die Layer untereinander vom Input- zum Output-Layer voll vermascht sind und keine Rückkopplungen vorhanden sind.

Neuronen

Ein Neuron wird durch eine mathematische Funktion beschrieben und bildet einen Vektor von Eingabedaten auf einen Ausgangswert ab. Die gewünschte Ausgabe kann durch ein *Training* der Parameter *angelernt* werden.

In klassischen Feed-Forward Netzen ist ein Neuron mit jedem Neuron aus dem vorherigen Layer verbunden und erhält als Eingabedaten die Ausgangswerte aller Neuronen des ihm vorgelagerten Layers. Abbildung 2.4 zeigt schematisch ein Neuron und ein Netz aus Neuronen.

Die Ausgabe eines Neurons wird durch mehrere Parameter bestimmt. Jedes Datum des Inputvektors i (In der Abbildung x) wird mit einer Gewichtung w_j multipliziert. Die resultierenden Produkte werden aufsummiert. Die Vektoren i und w müssen stets die selbe Länge aufweisen. Neben den Gewichtungen wird ein Bias-Wert b verwendet, welcher auf die aus Inputvektor und Gewichtungen resultierenden Summe addiert, bzw. von dieser subtrahiert wird (siehe

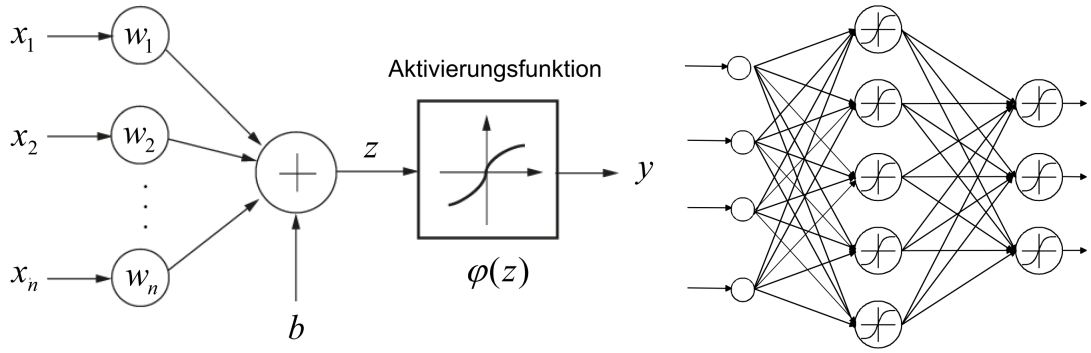

Abbildung 2.4.: Aus [Meisel (2017)]: Links: Schematische Darstellung eines künstlichen Neurons. Rechts: Zusammenschluss mehrerer Neuronen in Schichten zu einem künstlichen neuronalen Netz

Abbildung 2.4). Durch das Bias kann die Ausgabe des Neurons besser an die gewünschte Funktionsabbildung angepasst werden, wodurch die Performance des gesamten Netzes gesteigert wird.

Das Resultat aus der Summe der einzelnen Multiplikationen der Gewichtungen mit dem Inputvektor und der Addition des Bias bildet den Eingabewert einer Aktivierungsfunktion f. Das Ergebnis dieser Funktion bestimmt den Input-Wert für die Neuronen im nachfolgenden Layer, bzw. die Ausgabe des Netzes. Die allgemeine Formel zur Berechnung des Outputs o eines Neurons lautet in Anlehnung an Abbildung 2.4 wie folgt:

$$o = f(i * w + b) \tag{2.3.1}$$

Man betrachtet ein Neuron für gewöhnlich als Teil eines Layers und berechnet die Ausgaben aller Neuronen des Layers in einem Arbeitsschritt. Hierdurch können die Ausgabe-Werte durch eine Formel dargestellt werden. Die Gewichtungen aller Neuronen eines Layers werden durch die Gewichtungsmatrix W_{nj} beschrieben. Hierbei entspricht j der Anzahl an Neuronen im zu berechnenden Layer und n der Anzahl im vorherigen Layer. n hat somit ebenfalls die Länge des Inputvektors i. Die Gewichtungsmatrix lautet wie folgt:

$$W_{nj} = \begin{pmatrix} W_{11} & W_{21} & \dots & W_{1j} \\ W_{21} & W_{22} & \dots & W_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \dots & W_{nj} \end{pmatrix} \tag{2.3.2}$$

Der Output-Vektor o eines Layers enthält die Ausgabewerte o_j aller Neuronen des Layers. Dieser kann durch folgende Formel berechnet werden:

$$o_j = f((\sum_{j=1}^{n}(i_n * w_{nj})) + b_j) \tag{2.3.3}$$

Die Wahl der Aktivierungsfunktion f bestimmt zu einem großen Teil die Ausgabe eines Neurons und hat damit einen entscheidenden Einfluss auf die Performance des Netzes. Häufig finden der *Tangens Hyperbolicus* (*tanh*), die *Sigmoid*-Funktion (*sgm*), die *Linear*-Funktion (*lin*) oder eine Linear-Funktion mit einem minimalen Schwellwert (engl. *Rectified Linear Unit, ReLU*) Anwendung als Aktivierungsfunktionen. Der Tangens Hyperbolicus ist eine spezielle Form der Sigmoid-Funktion mit erweitertem Wertebereich. Beide haben einen S-förmigen Verlauf. Die Sigmoid-Funktion (Gleichung 2.3.4) bildet im Wertebereich $[0; 1]$ ab, der Tangens Hyperbolicus (Gleichung 2.3.5) im Bereich $[-1; 1]$.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3.4}$$

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.3.5}$$

Wird eine Sigmoid-Funktion als Aktivierungsfunktion verwendet, sollten die Input-Werte normalisiert und zu 0 hin zentriert werden. So wird das Training verkürzt und man erhält schneller ein erfolgreiches Ergebnis (vgl. [Lecun u. a. (1998)]). Weiterhin findet bei großen Datenwerten eine Sättigung statt, da diese von einer Sigmoid-Funktion nicht mehr erfasst, bzw. unterschieden werden können. Diese Sättigung wird als *Vanishing Gradient* bezeichnet (vgl. [Bengio u. a. (1994)] und [Pascanu u. a. (2012)]). So kann es zu Rundungsfehlern und damit schlechten Ergebnissen kommen. Abhilfe kann die Verwendung der linearen (Gleichung 2.3.6), bzw. der ReLU (Gleichung 2.3.7) Aktivierungsfunktion schaffen.

$$f(x) = x \tag{2.3.6}$$

$$f(x) = max(0, x) \tag{2.3.7}$$

Die Linear-Funktion kann Datenwerte über 1 bzw. -1 hinaus abbilden, wodurch eine Sättigung vermieden wird. Weiterhin wird eine Normalisierung der Eingangsdaten überflüssig, da auf den gesamten Zahlenbereich abgebildet werden kann. Kommen nur positive Zahlenwerte

vor, kann eine ReLU verwendet werden. Abbildung 2.5 stellt die Verläufe der beschriebenen Funktionen um den Bereich von 0 grafisch dar.

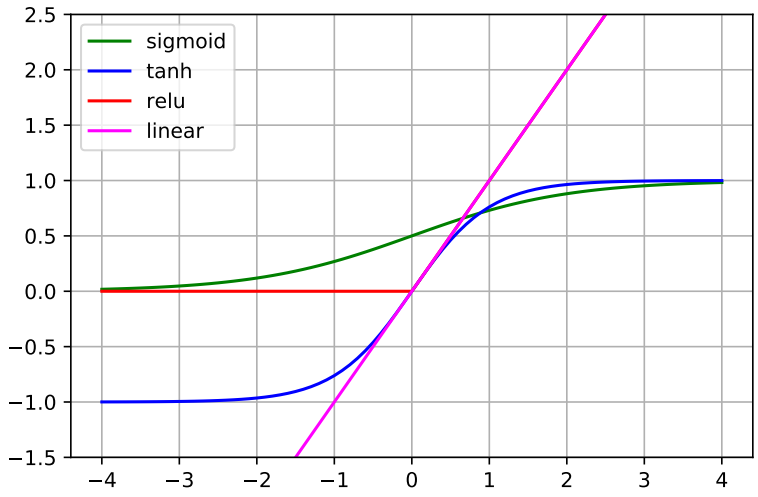

Abbildung 2.5.: Darstellung unterschiedlicher Aktivierungsfunktionen um den Nullpunkt

Hyperparameter

Verschiedene *Hyperparameter* definieren die *Architektur* des Netzes. Diese sind u. A. die Anzahl der Neuronen und Layer sowie deren Einteilung in die Layer und die Wahl der Aktivierungsfunktionen. Typischerweise verwenden alle Neuronen in einem Layer die gleiche Aktivierungsfunktion.

Die Wahl der Netz-Architektur ist maßgeblich dafür entscheidend, wie geeignet das Netz für das Lösen einer gegebenen Problemstellung ist. Um eine gute Performance zu erreichen, muss eine für die Problemstellung geeignete Anzahl an Neuronen gewählt werden. Sind zu wenige Neuronen vorhanden, resultiert zumeist ein *Underfitting*. Bei diesem kann das Netz den Funktionsverlauf (im Falle der Regression) nicht ausreichend abbilden. Dies kann damit verglichen werden, dass bei einem Fitting oder einer Spline Interpolation zu wenige Stützstellen für eine gute Approximation vorhanden sind. Gleiches gilt für den Fall der Klassifikation. Hier können die Klassen nicht ausreichend genau separiert werden. Grundsätzlich gilt, dass eine größere Anzahl von Neuronen komplexere Problemstellungen lösen kann. Jedoch kann eine zu große Anzahl an Neuronen zu einem *Overfitting* führen. Bei diesem kann das Netz die Trainingsdaten nicht ausreichend *generalisieren*, so dass nur die Klassen aus dem Trainings-

datensatz korrekt erkannt werden. Bei einer Regression wird im Falle von Overfitting nicht nur der generelle Funktionsverlauf gelernt, sondern im speziellen auch das Rauschen. Soll nun eine Funktion approximiert werden, würde auch ein oft unerwünschtes Rauschen generiert werden. Abbildung 2.6 zeigt die Regression von Daten, bei denen die Approximation jeweils overfitted und underfitted ist sowie eine gute Approximation, welche dem Funktionsverlauf folgt und Rauschen filtert.

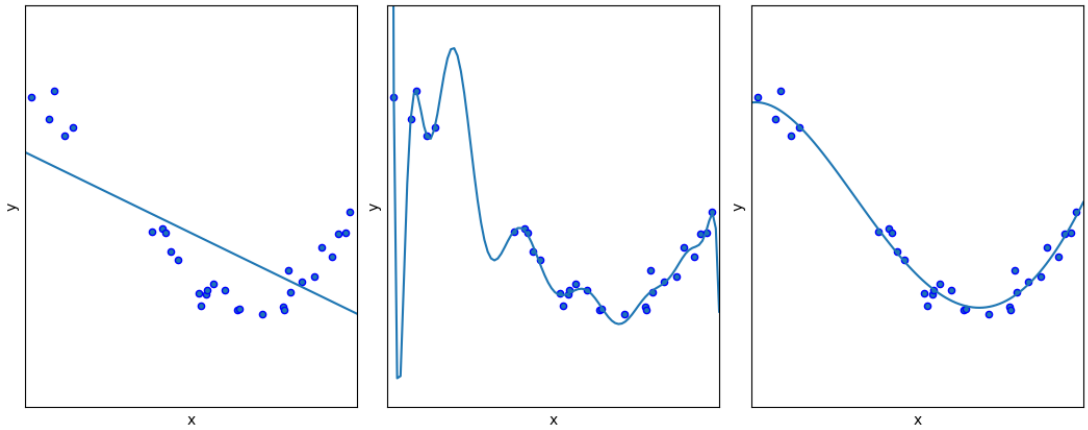

Abbildung 2.6.: Links: Schlechte Approximation durch Underfitting. Mitte: Zu aggressive Approximation durch Overfitting. Rechts: Gute Approximation

Neben der Anzahl der Neuronen spielt auch die Einteilung dieser in den einen oder die mehreren Hidden-Layer eine Rolle. In [Cybenko (1989)] ist das *Universal-Approximation-Theorem* (*UAT*) definiert. Das UAT sagt aus, dass ein Multi-Layer-Perzeptron, welches Sigmoid-Aktivierungsfunktionen verwendet und nur einen Hidden Layer enthält, grundsätzlich jede Funktion approximieren kann. Jedoch gilt hierfür, wie oben beschrieben, dass eine ausreichende Anzahl an Neuronen vorhanden sein muss, um einem Underfitting vorzubeugen. Jedoch sagt das UAT nichts darüber aus, ob ein Multi-Layer-Perzeptron mit nur einem Hidden-Layer besser performt oder lernt, als eines mit mehreren Layern.

Es gilt im Allgemeinen, dass Netze mit mehr Neuronen komplexere Problemstellungen lösen können. Eine hohe Anzahl an Neuronen führt jedoch zu praktischen Problemen. So wird etwa die Lernphase verlängert, da mehr Neuronen antrainiert werden müssen. Es wird mehr Rechenleistung benötigt, um der verlängerten Trainingszeit entgegenzuwirken. Hinton et al. haben in 2006 eine Methode vorgestellt, um Multi-Layer-Perzeptrons mit mehr als einem Hidden-Layer effizient zu trainieren (vgl. [Hinton u. a. (2006)]). Zusätzlich wurde es in den letzten Jahren möglich, die Netze nicht mehr ausschließlich mit der CPU, sondern auch mit Grafikprozessoren (engl. *GPUs*) trainieren zu lassen. Die von Hinton et al. vorgestellten

Konzepte und die Möglichkeit, die Netze von der GPU trainieren zu lassen, ermöglichen die effiziente Entwicklung und Training von vielschichtigen Netzen mit mehreren Hidden-Layern. Solche Netze, bzw. die Verwendung dieser, wird als *Deep Learning* bezeichnet.

Die Suche nach geeigneten Hyperparametern ist für jedes Problem individuell und nach wie vor schwierig. Es existieren wenige Faustregeln, mit welchen Parametern und welcher Architektur eine Problemstellung im Ersten Schritt angegangen werden sollte (vgl. [Bengio (2012a)]). Meisel schreibt hierzu in [Meisel (2008)]:

> "*Der Entwurf eines Neuronalen Netzes ist mehr Kunst als Wissenschaft.*"

Trainingsprozess

Sowohl biologische, als auch künstliche neuronale Netze müssen *antrainiert* (auch *angelernt*) werden, um eine gewünschte Funktion möglichst genau abbilden zu können. Für das *Training* gibt es verschiedene allgemeine Verfahren, wie etwa *überwachtes*, *unüberwachtes* oder *verstärkendes* Lernen. In diesem Abschnitt wird auf überwachtes Lernen eingegangen.

Das Ziel des Trainings ist es, bei einer bestimmten Eingabe die Ausgabe des Netzes an eine gewünschte Ausgabe anzunähern. Hierzu wird ein Trainingsdatensatz verwendet, welcher aus Paaren von Eingangswerten (*Features*) und gewünschten Ausgangswerten (*Labeln*) besteht. Durch den Prozess des Trainings werden die Gewichtungen w der Verbindungen zwischen den Neuronen und deren Bias-Werte b schrittweise angeglichen, wodurch der Fehler zwischen tatsächlicher und gewünschter Ausgabe verringert wird. Das Training von neuronalen Netzen wird typischerweise mit dem Backpropagation Algorithmus unter Verwendeung des Stochastic Gradient Decent Verfahrens durchgeführt.

Backpropagation

Die Anpassung der Gewichtungen und der Biase, durch welche der Fehler minimiert wird, erfolgt zumeist mit dem *Backpropagation* Algorithmus. Dieser besteht aus einer Initialisierungsphase und drei sich wiederholenden Phasen. Zunächst werden alle Gewichtungen und Biase mit Zufallswerten initialisiert. Die folgenden drei Schritte werden solange wiederholt, bis ein zuvor festgelegter Fehlerwert unterschritten ist oder eine maximale Anzahl an Wiederholungen erreicht wurde. Der maximale Fehlerwert muss für jedes Netz individuell so festgelegt werden, dass ein für die Problemstellung passendes Verhältnis aus hoher Generalisierbarkeit und wenig Overfitting entsteht.

Die drei sich wiederholenden Phasen können grob wie folgt beschrieben werden:

1. Es wird ein Elemente-Paar aus dem Trainingsdatensatz gewählt. Die Eingabe (Feature) wird in das Netz gegeben, welches daraus eine Ausgabe erzeugt.
2. Diese Ausgabe wird mit der gewünschten Ausgabe (Label) verglichen. Hieraus wird ein Fehlerwert berechnet.
3. Mit dem Fehlerwert wird eine Anpassung der Gewichtungen und Biase vorgenommen, wobei dieser durch das Netz *zurück-propagiert* wird.

Dieser Algorithmus wird für jedes Element-Paar aus dem Trainingsdatensatz angewendet. Daneben existiert auch ein Batch-Verfahren, bei dem zuerst die Fehlerwerte aller Element-Paare bestimmt werden. Erst wenn alle Fehlerwerte berechnet wurden, werden die Gewichtungen angepasst. Ein kompletter Durchlauf des Algorithmus über den Trainingsdatensatz wird in beiden Fällen als *Epoche* bezeichnet.

Zur Errechnung des Fehlerwertes zwischen gewünschter und berechneter Ausgabe muss eine Metrik verwendet werden. Diese kann zum Beispiel die Fehlerquadratsumme E sein:

$$E = \sum (label - output)^2 \tag{2.3.8}$$

Für die Anpassung der Gewichtungen und Biase wird ein Gradient ∇E verwendet, der sich aus der partiellen Ableitung der Fehlerfunktion ergibt. Mit Hilfe der Kettenregel der Differenzialrechnung wird dieser Gradient für jedes Neuron gebildet (vgl. [Meisel (2017)]). Der Gradient wird daraufhin von den Gewichtungen aller Verbindungen zu den jeweils nachfolgenden Neuronen subtrahiert. Weiterhin fließt eine Multiplikation mit einer *Lernrate* μ ein, mit welcher die Anpassungsstärke der einzelnen Schritte angepasst werden kann. Diese ist ein für das Training zu bestimmender Faktor, mit dem das Training parametriert werden kann. Die Anpassung der Gewichtungen für den nachfolgenden Iterationsschritt $t + 1$ wird durch folgende Gleichung dargestellt:

$$w_{t+1} = w_t - \mu * \nabla E \tag{2.3.9}$$

Die durch diese Methode angepassten Gewichtungen und Biase führen in der Regel zu einer Annäherung der Ausgabe hin zu der gewünschten Ausgabe, welche durch das Label bestimmt wird.

(Stochastic) Gradient Decent

Die Suche nach guten Datenwerten für die Gewichtungen ist ein Optimierungsproblem. Es wird versucht, das globale Minimum in einem mehrdimensionalen Fehlergebirge zu finden,

bei welchem der gewünschte maximale Fehlerwert unterschritten wird. Die Abbildung 2.7 soll das Verfahren verdeutlichen. Gezeigt wird das Fehlergebirge von zwei Gewichtungen, wobei die X- und Y-Achsen den Datenwerten der Gewichtungen und die Z-Achse den Fehler zwischen gewünschter und tatsächlicher Ausgabe darstellt. Die partielle Ableitung des Fehlerwertes, und damit des Gradienten, beschreibt die benötigte Veränderung der Datenwerte der Gewichtungen, um sich dem Fehlerminimum zu nähern. Dieser Abstieg in ein Fehlertal wird als *Gradientenabstieg* (engl. *Gradient Decent*) bezeichnet. Der Gradient kann aus einer beliebigen Anzahl von Element-Paaren aus dem Trainingsdatensatz gebildet werden. Wird der komplette Trainingsdatensatz verwendet, ist das Ergebnis der *wahre* Gradient, da er die Veränderung der Gewichtungen aller Element-Paare widerspiegelt. Man spricht hierbei schlicht vom *Gradient Decent* (*GD*) Verfahren.

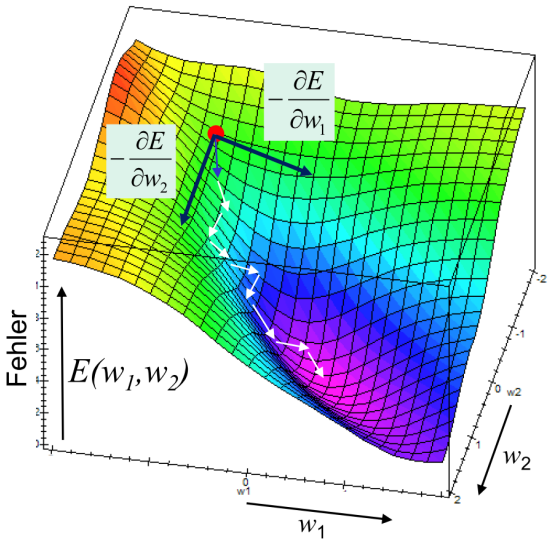

Abbildung 2.7.: Aus [Meisel (2017)]: Gradientenabstieg in ein Fehlertal

Die Gewichtungen werden so nur einmal pro Epoche modifiziert. Dies führt besonders bei großen Datensätzen zu einem enormen Rechenaufwand, da unter Umständen eine sehr große Anzahl an Epochen benötigt wird, um den gewünschten minimalen Fehlerwert über alle Datensätze zu erreichen. Es existiert ein auf Rechenzeit optimiertes *Stochastic* Gradient Decent (*SGD*) Verfahren. Bei diesem wird nur eine Teilmenge der Element-Paare (*Mini-Batch*) aus dem Trainingsdatensatz zur Bildung des Gradienten verwendet. Hierbei leitet sich der Begriff Stochastic davon ab, dass der Gradient nicht dem *wahren* entspricht, welcher sich aus allen Element-Paaren ergibt, sondern er wird aus zufällig gewählten Teilmenge gebildet und ist daher stochastisch approximiert.

Beim Gradientenverfahren kann es zu verschiedenen Problemen kommen. Die Abbildung 2.8 zeigt, wie der Gradient beispielsweise in steilen Abstiegen anfangen kann, stark zu oszillieren oder in flachen Plateaus stagnieren kann. Beide Probleme verlängern den Lernprozess, bzw. minimieren die Chance, ein optimales Fehlertal zu finden, bei welchem das Netz eine gute Performance abliefert.

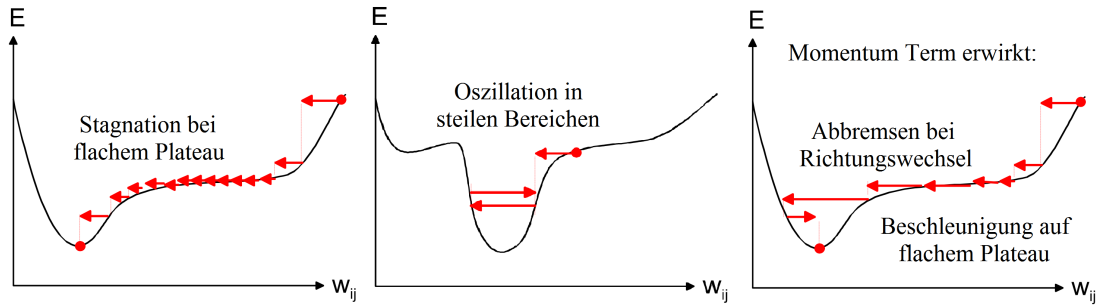

Abbildung 2.8.: Aus [Meisel (2017)]: Probleme beim Trainingsverlauf verursacht durch Stagnation des Gradienten auf Plateaus (links) und ungewollte Oszillation in Fehlertälern (mitte). Rechts: Einführung eines Momentum-Terms zur Anpassung der Sprungweite des Gradienten

Diese Probleme können durch das Hinzufügen eines zusätzlichen *Momentum-Terms* m minimiert werden (siehe Gleichung 2.3.10). Dieser berücksichtigt die Schrittweite der vorherigen Iteration, so dass auf flachen Plateus *beschleunigt* und bei Oszillation *abgebremst* werden kann.

$$w_{t+1} = w_t - \mu * \nabla E * m \tag{2.3.10}$$

2.3.2. Time-Recurrent Netze

Einige Problemstellungen erfordern die Klassifizierung von Merkmalen über einen Zeitraum von Eingangsdaten. Zum Beispiel, wenn etwa basierend auf aktuellen Wetter- und Klimadaten das Wetter des nächsten Tages oder der zu erwartende Verlauf eines Aktienkurses bestimmt werden soll. Essentiell handelt es sich hierbei um eine Extrapolation, bei welcher der Verlauf einer Funktion *vorhergesagt* wird. Für die eigene Problemstellung muss das resultierende Flugverhalten aus aktuellem Zustand und aktueller Steuereingabe vorhergesagt werden.

Ein *rekurrentes neuronales Netz* (engl. *Time-recurrent neural network, RNN*) kann Zusammenhänge in Sequenzen von Eingangsdaten erkennen sowie verarbeiten und daraus den weiteren Verlauf der Funktion vorhersagen. Die Eingangsdaten sind nicht länger unabhängig, sondern sie bilden eine Sequenz von zueinander abhängigen Inputvektoren, welche nacheinander in

das Netz eingespeist werden. Die Erkennung der Zusammenhänge in den Sequenzen kann auf grundlegend unterschiedliche Arten erfolgen.

Sliding-Window Verfahren

Ein gewöhnliches Feed-Forward Netz kann *temporale* Abhängigkeiten innerhalb der Eingangsdaten unter Anwendung des *Siding Window* Verfahrens erkennen. Hierfür werden neben dem *"aktuellen"* Inputvektor X_t zum Zeitpunkt t auch die n letzten Inputvektoren X_{t-i} als Eingangsdaten verwendet. Der Inputvektor erweitert sich in diesem Falle um die n letzten Inputvektoren.

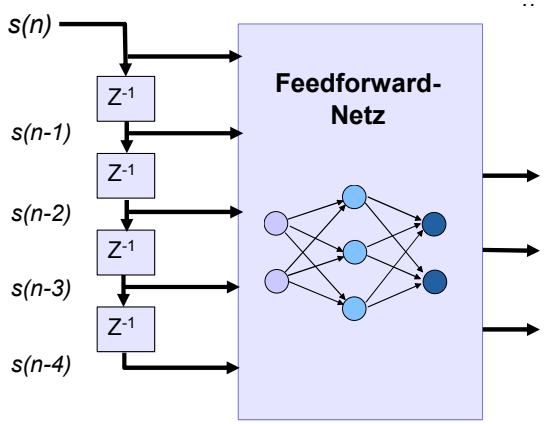

Abbildung 2.9.: Aus [Meisel (2017)]: Ein Feed-Forward Netz mit einem Sliding-Window aus Daten des aktuellen und n vorheriger Zeitschritte der Sequenz

Die Abbildung 2.9 zeigt exemplarisch die Erweiterung eines Feed-Forward Netzes um ein Sliding-Window von vier zusätzlichen Inputvektoren. Die Länge des Windows bestimmt, wie viele zeitlich zurückliegende Datenpunkte in der Sequenz verwendet werden sollen, um die Ausgabe Y_t zum Zeitpunkt t vom Netz erzeugen zu lassen und damit die Ausgabe zu extrapolieren. Das Sliding-Window aus vergangenen Inputvektoren verhält sich ähnlich zu einem Kurzzeitgedächtnis für das Netz, welches sich die letzten Inputvektoren merkt.

Recurrent Neural Network

In *inhärent* rekurrenten neuronalen Netzen (*RNNs*) bildet eine Untermenge der Neuronen Rückkopplungen, wobei deren Outputwerte als Inputwerte für sich selbst oder für Neuronen in weiter vorne liegenden Schichten dienen. Abbildung 2.10 zeigt schematisch ein RNN mit einem Hidden-Layer, in welchem alle Neuronen auf sich selbst rückgekoppelt sind, wodurch

die Ausgaben im nächsten Zeitschritt als Eingaben vorliegen. Die Zwischenspeicherung erfolgt in einer *Kontextschicht*. Dieser spezielle Netztyp wird als *Elman*-Netz bezeichnet [Elman (1990)], welcher für die Verarbeitung sequentieller Daten geeignet ist und daher häufig für Problemstellungen solcher Art verwendet wird.

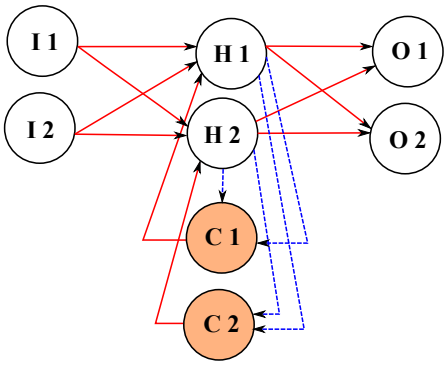

Abbildung 2.10.: Elman-Netz mit *Gedächtnis* durch Zwischenspeicherung in Kontextschicht

Die *Ausführung* neuronaler Netze erfolgt in diskreten Ausführungsschritten, wobei nacheinander die Ausgabe jedes Neurons einzeln berechnet wird. Sind die Ausgangswerte aller Neuronen berechnet, ist der Ausführungsschritt abgeschlossen und es liegt ein neues Ergebnis in Form eines Outputvektors vor. Im nächsten Ausführungsschritt stehen die Outputwerte von rückgekoppelten Neuronen als Inputwerte für andere Neuronen zur Verfügung. Die Berechnung des Outputwertes eines rückgekoppelten Neurons erfolgt durch die in Gleichung 2.3.11 dargestellte Formel.

$$o_t = f(i * w_i + o_{t-1} * w_o + b) \tag{2.3.11}$$

Durch diese Rückkopplungen kann ein rekurrentes Netz Daten über mehrere Ausführungsschritte hinweg, und damit über eine Sequenz, speichern und Daten mit sequentiellen Abhängigkeiten verarbeiten. So entsteht ein *Gedächtnis* in Form von rückgekoppelten Daten, die über mehrere Ausführungsschritte hinweg bestehen bleiben. Auf diese Weise kann ein RNN eine Extrapolation von sequentiell vorliegenden Daten vornehmen.

Das Lernen bei rekurrenten Netzen findet nicht länger lediglich über die einzelnen Datensätze, sondern auch über die Sequenzen dieser statt. Das Vorgehen wird als *Backpropagation through time* (*BPTT*) bezeichnet. Diese *entfaltet* die Rückkopplungen, so dass im Prinzip ein gewöhnliches Feed-Forward-Netz entsteht. Dieses kann daraufhin mit der einfachen Backpropagation und dem Stochastic Gradient Decent trainiert werden. Allerdings muss bei RNNs

sowohl beim Training, als auch beim Ausführen ein initialer Wert für den internen Zustand der Rückkopplungen bestimmt werden. Dieser ist typischerweise 0.

Durch das Entfalten ergibt sich je nach Grad der Rückkopplung ein Netz mit vielen Hidden-Layern. Hierdurch erhöht sich die Wahrscheinlichkeit für das Vanishing Gradient Problem, da der Gradient über viele Schichten hinweg berechnet werden muss [Hochreiter und Schmidhuber (1997)]. Dieses Problem kann mit der Verwendung der Linearen, bzw. der ReLU bei rein positiven Datenwerten, als Aktivierungsfunktion umgangen werden.

Analog zu anderen dynamischen Systemen, welche mit Rückkopplungen arbeiten, hat auch die BPTT eine besondere Anfälligkeit für ein chaotisches Verhalten. Hierdurch entsteht ein hohes Risiko dafür, dass der Gradient im Fehlergebirge unkontrolliert herumspringt oder in lokalen Minima feststeckt [Cuéllar u. a. (2006)], wodurch das Training schlechte Ergebnisse liefert. Dies macht die Wahl der Trainingsparameter, wie etwa der Lernrate, für RNNs und BPTT besonders schwierig.

3. Techniken und vergleichbare Systeme

Die Steuerung eines Flugzeugs mit Teilausfällen der Aktorik ist ein komplexes Vorhaben. Das vorliegende Kapitel gibt eine Übersicht verschiedener Verfahren zur Ausfallerkennung und adaptiven Steuerflächenzuordnung. Im letzten Abschnitt werden vergleichbare Systeme und Konzepte vorgestellt, die eine andere Aufgabenteilung verwenden.

Die Grundlegende Idee, das Gesamtsystem in zwei Komponenten aufzuteilen, wurde dem Ansatz von Ducard et al. entnommen [Ducard (2007), Schneider u. a. (2012)]. Bei seiner Umsetzung einer fehlertoleranten Flugsteuerung wurden die Aufgaben des Systems in *Fault Detection* und *Control Allocation* aufgeteilt. Die Abbildung 3.1 zeigt seinen Systementwurf zur fehlertoleranten Steuerung von Multirotorsystemen. Der Regelkreis besteht aus vier Komponenten. Der Träger bildet die Regelstrecke. Die restlichen drei Komponenten ergeben die Avionik. Das *Stabilization Control System* stellt den Flugregler dar, welcher die gewünschten Änderungen der Freiheitsgrade ausgibt. Die reine Flugsteuerung besteht aus der *Fault Detection* und der *Fault-tolerant Control Allocation*. Diese ist dafür zuständig, die Kommandos vom Flugregler durch die am Träger vorhanden Rotoren in Bewegungen umzusetzen. In Abschnitt 3.3 werden Systeme präsentiert, bei welchen die einzelnen Teilprobleme nicht durch isolierte Komponenten gelöst werden.

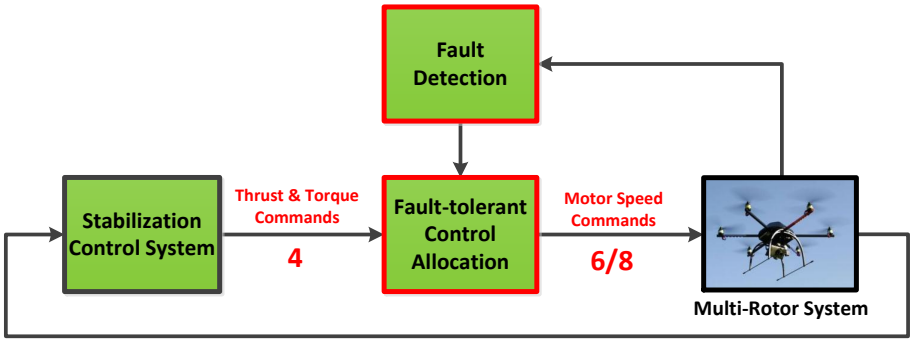

Abbildung 3.1.: Aus [Schneider u. a. (2012)]: Aufteilung der Flugsteuerung in einzelne Komponenten Fault Detection und Control Allocation

3.1. Ausfallerkennung

Die Diagnose von Systemen zur Steuerung von physikalischen oder industriellen Prozessen hat zum Ziel, Fehlverhalten zu detektieren, zu isolieren und schließlich zu identifizieren. Dabei ist es unerheblich, ob das Fehlverhalten durch einen Fehler im System oder durch den Prozess selbst erzeugt wird. Es wird eine möglichst zeitnahe Erkennung des Fehlverhalten angestrebt, um dieses kompensieren zu können. Wird dieses nicht oder zu spät kompensiert, sind Schäden am System selbst oder der Umwelt die Folge (vgl. [Patan (2008)]). Die grundsätzliche Idee der Ausfallerkennung besteht darin, Inkonsistenzen zwischen erwartetem und gemessenen Prozess- oder Systemverhalten zu erkennen. Aus erkannten Differenzen und Anomalien kann auf ein Fehlverhalten und (Teil-) Ausfall des Systems geschlossen werden. Zur Erkennung von Fehlverhalten kann auf zwei grundlegende Ansätze zurückgegriffen werden. Zum einen können die vom Prozess ausgegebenen Signale auf erlaubte Grenzwerte, Plausibilität oder unerlaubte Korrelationen zwischen einander überprüft werden, welche ein fehlerfrei arbeitendes System verbieten würde. Zum anderen kann ein Modell des Prozesses, bzw. der Regelstrecke gebildet werden, welches die erwarteten Ausgangssignale des Prozesses generiert. Aus der Differenz zwischen erwartetem und gemessenen Signal wird ein Restwert (engl. *Residual*) gebildet. Der Restwert wird über die Zeit evaluiert, wobei nach Mustern oder Grenzwerten gesucht wird, welche auf einen bestimmten Fehler oder Ausfall schließen lassen. Abbildung 3.2 zeigt das allgemeine Konzept der Erzeugung von Restwerten und der Evaluierung dieser, um daraus Fehler zu erkennen.

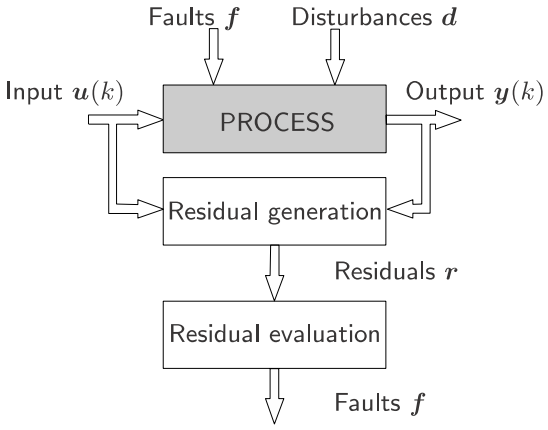

Abbildung 3.2.: Aus [Patan (2008)]: Erkennung und Ableiten von Fehlern durch die Erzeugung und anschließende Evaluierung eines Restwerts

Für die Ausfallerkennung werden zumeist die englischen Begriffe *Fault Detection and Identification* (*FDI*) oder *Fault Detection and Diagnosis* (*FDD*) verwendet. [Patan (2008)] stellt eine umfassende Übersicht und Einführung in das Thema dar. Nachfolgend sind einige typische Ansätze beschrieben, wie eine modellbasierte Ausfallerkennung basierend auf der Methode der Evaluation von Restwerten umgesetzt werden kann.

3.1.1. Sensorik

Die tatsächliche Stellung der Steuerflächen kann durch verschiedene Sensoren erfasst werden. Die Klappen und Ruder werden typischerweise durch Servomotoren bewegt. Bei großen Flugzeugen werden diese durch eine Hydraulik oder elektrisch angesteuert. Bei älteren Sportflugzeugen kommen meist Seilzüge zum Einsatz, bei neueren ebenfalls eine elektrische Steuerung. Bei UAVs und Modellflugzeugen werden fast ausschließlich elektrische Modellbauservos eingesetzt. Unabhängig vom Übertragungsweg der vom Piloten oder Autopiloten erzeugten Signale können Dreh- oder Linearencoder eingesetzt werden. Diese Messen die tatsächliche Stellung des Servomotors.

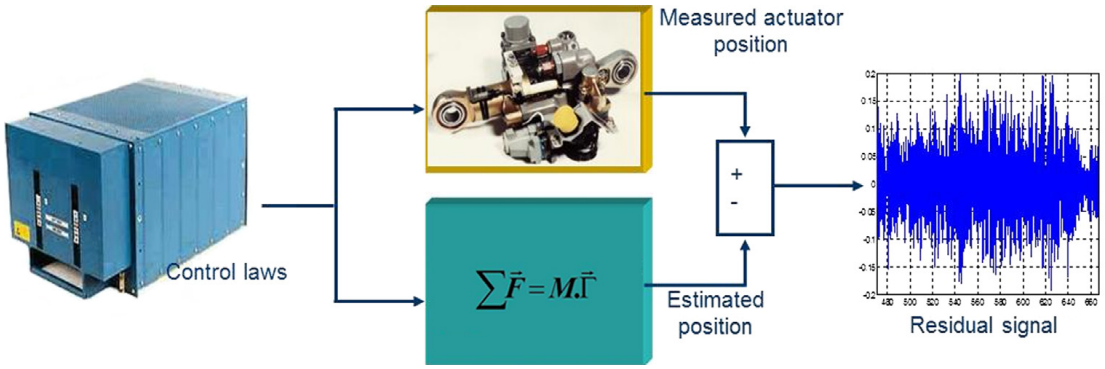

Abbildung 3.3.: Aus [Efimov u. a. (2013)]: Erzeugung des Restwerts durch Positions-Sensor am Aktuator

Über ein Zeitfenster wird die Fehlerquadratsumme zwischen angesteuerter und tatsächlicher Stellung akkumuliert. Die Abbildung 3.3 zeigt den grundsätzlichen Ansatz dieser Messung. Durch die Anwendung eines geeigneten Fehlermodells kann auf verschiedene Ausfalltypen geschlossen werden. Wie etwa ein *Einfrieren*, *Nachlaufen* oder *Schwingen* der Steuerfläche. Efimov et al. beschreiben den Einsatz von Differentialtransformatoren, um ein Schwingen der Quer- und Höhenruder beim Airbus A380 zu erkennen [Efimov u. a. (2013)]. Die Abbildung 3.4 stellt das verwendete Systemmodell dar. Die linke Seite zeigt die Erzeugung des

Restwerts (*Residual Generation*), die rechte Seite das Erkennen von Schwingungen anhand einer Untersuchung des Restwerts auf bestimmte Muster (*Residual Evaluation*).

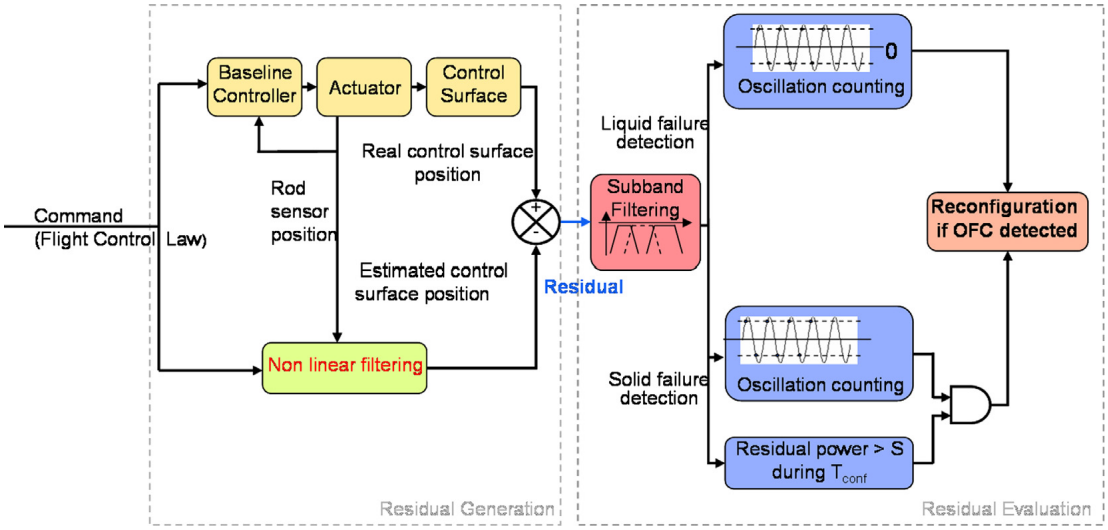

Abbildung 3.4.: Aus [Efimov u. a. (2013)]: Systemmodell zur Erkennung von Oszillation eines Aktuators

Diese Technik benötigt keine umfassende mathematische Beschreibung des kompletten Flugverhaltens. Lediglich ein Modell der Dynamik der Servomotoren muss vorhanden sein. Die Komplexität dieses Modells hängt von den zu erkennenden Ausfalltypen ab. Soll lediglich ein Einfrieren erkannt werden, kann sich auf das Erfassen von Änderungen in der gemessenen Stellung beschränkt werden. Alternativ oder komplementär können zwischen Tragfläche, bzw. Leitwerk, und Steuerfläche Dehnungsmessstreifen, Wägezellen oder Abstandssensoren eingesetzt werden. Nachteilig wirkt an dieser Methode, dass die Servomotoren, bzw. Steuerflächen mit der Sensorik ausgestattet sein müssen. Diese erhöhen das Gewicht und die Kosten des Flugzeugs.

Weiterhin kann die Stellung der Steuerflächen durch ein Kamerabild erfasst werden. Der Vorteil liegt darin, dass alle Steuerflächen durch eine einzige Kamera auf einem schwenkbaren Gimbal überwacht werden können. Jedoch kann ein extern angebrachter Gimbal die Aerodynamik maßgeblich beeinträchtigen. Weiterhin wird das Modell zur Ausfallerkennung ungleich komplexer, da das Themenfeld der Bildverarbeitung hinzukommt.

Durch die Beschränkung der mathematischen Beschreibung auf die Aktuatordynamik, kann die beschriebene Technik der Verwendung von Sensoren aus Sicht der Regelungstechnik und Zustandsüberwachung relativ einfach umgesetzt werden. Jedoch bedingen sie das Vorhandensein der Sensoren bei allen zu überwachenden Steuerflächen und ist daher unpraktikabel

bei einem Wechsel des realen Trägersystems. Sie stellen damit keine allgemeine Lösung dar, welche durch ein reines Computersystem und Algorithmen erreicht werden kann.

3.1.2. Analytischer Zustandsbeobachter

Eine Alternative zum Einsatz von Sensorik ist die Implementierung von auf analytischen Modellen basierenden *Zustandsbeobachtern*. Hierbei wird das Flugverhalten durch ein mathematisches Modell der Bewegungsgleichung beschrieben (siehe Abschnitt 2.2), welches in jedem Regelschritt berechnet wird. So können Teile des Flugzustands, wie etwa Geschwindigkeit, Höhe oder Lage, geschätzt werden. Die Schätzung entspricht dem erwarteten Systemzustand. Ein Beobachter berechnet diesen erwarteten Zustand rein aus der mathematischen Beschreibung des Systems.

Aus gemessenem und erwartetem Zustand ergibt sich eine Divergenz. Diese Divergenz kann auf einen Ausfall der Steuerflächen hindeuten. Zum Beispiel leitet der (Auto-) Pilot eine Kurve ein und es wird ein Rollmoment erzeugt. Innerhalb eines Zeitfensters wird die Ausgabe des Drehraten- und Lagesensors mit dem erwarteten (geschätzten) Zustand verglichen. Analog zur Berechnung der Differenz aus angesteuerter und gemessener Steuerflächenposition kann aus der hier berechneten Divergenz ebenfalls ein Ausfall und dessen Ausfalltyp abgeleitet werden, sofern dem mathematischen Modell vertraut wird und ein Fehlverhalten der Sensorik ausgeschlossen wird. Wird in einem Zeitfenster nach der Ansteuerung etwa keine Kurve geflogen, liegt ein Einfrieren der Querruder vor. Bei einer nicht adaptiven Steuerflächenzuordnung folgt daraus ein Komplettausfall der Steuerbarkeit im Freiheitsgrad Rollen.

Das Bilden der Bewegungsgleichung ist eine komplexe Aufgabe der Flugmechanik. Die in Gleichung 2.2.1 und Abbildung 2.3 dargestellte Bewegungsgleichung stellt lediglich ein allgemeines Modell dar. Für dieses gelten verschiedene Annahmen (vgl. [Nguewo (2014)] und [McLean (1990)]). Etwa, dass das Flugzeug symmetrisch und starr ist. Weiterhin wird angenommen, dass es sich um eine konventionelle *Drachenkonfiguration* handelt. Zur Herleitung der Derivativa, welche die Koeffizienten der Matrizen A, B und E ergeben, wird ein allgemeines Tragflächenprofil angenommen. Treffen diese Annahmen bei einem Träger nicht zu, muss die Struktur der Bewegungsgleichung angepasst werden. So etwa bei einer Blended-Wing-Body Flugzeugkonfiguration. Diese kann zwar auch als symmetrisch und starr angenommen werden, jedoch ergeben sich durch das unterschiedliche Tragflächenprofil und die besondere Steuerflächenkonfiguration eine andere Zusammensetzung der Derivativa. Dies führt dazu, dass die Struktur der in Abbildung 2.3 dargestellten Bewegungsgleichung das mathematische Modell des Flugverhaltens nur unzureichend abbildet.

Wird die Struktur der Bewegungsgleichung als gegeben angesehen und verwendet die allgemeine Form, benötigt dennoch jeder Träger andere Koeffizienten der Matrizen. Zwar können die Derivativa, wie in Abschnitt 2.2 beschrieben, von flugmechanisch ähnlichen Flugzeugen abgeleitet werden, jedoch ist hierdurch ebenfalls nur eine unzureichende Abbildung des Flugverhaltens gegeben. Dies führt zu einer Fehlschätzung der Flugzustände und somit zu False-Positives bei der Erkennung von Ausfällen. Daher kann diese Methode der Ausfallerkennung nur angewendet werden, wenn das mathematische Modell das Flugverhalten ausreichend realitätsnah nachbildet.

Typischerweise werden Zustandsbeobachters mithilfe von erweiterten Kalman-Filtern (*EKF*) implementiert [Barton (2012)]. Bei diesen ist das Flugverhalten im Filter modelliert, so dass der erwartete Folgezustand geschätzt werden kann. Abbildung 3.5 zeigt schematisch einen erweiterten Kalman-Filter, welcher zum Beispiel in UAVs eingesetzt werden kann. Verschiedene Sensorwerte dienen als Eingangssignale. Aus diesem und dem vorgehaltenen internen Zustand des Filters kann der Folgezustand, bzw. der aktuell vorliegende Zustand, geschätzt werden.

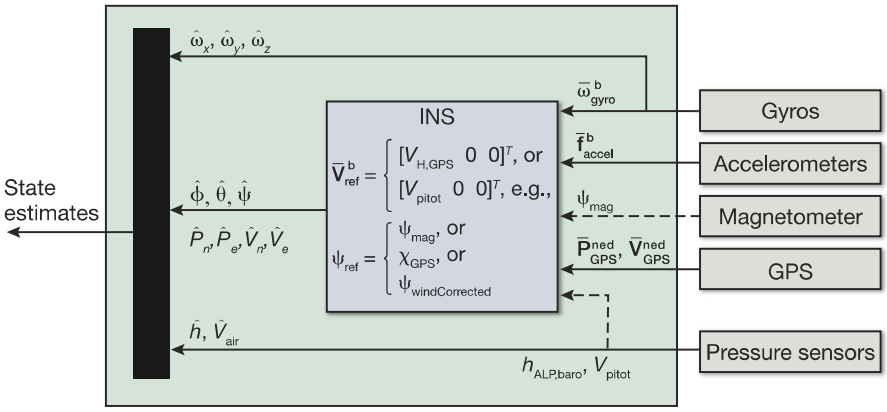

Abbildung 3.5.: Aus [Barton (2012)]: INS-State Estimator zur Verbesserung der Schätzung des Flugzustands basierend auf erweiterten Kalman-Filtern

In UAVs sind die Sensorwerte zumeist zu ungenau, um direkt den Systemzustand zu bilden [Barton (2012)]. In Abbildung 3.5 ist eine *Sensorfusion* dargestellt, welche aus verschiedenen Sensorwerten die Fluglagen, Positionen und Geschwindigkeiten in mehreren Achsen schätzt. Solch ein Zustandsschätzer (engl. *State Estimator*) wird häufig dafür eingesetzt, um Sensorungenauigkeiten herauszufiltern und so ein genaueres Abbild der Realität zu erhalten [Barton (2012)]. Je nach Ausprägung des EKF wird solch ein System häufig als *Inertial Navigation System* (*INS*) oder *Attitude and Heading Reference System* (*AHRS*) bezeichnet.

Manche Größen des Systemzustands können sensorisch nicht oder nur schwer erfasst werden. Zum Beispiel den Einflussvektor des Windes. Dieser kann, sofern er Teil des Systemzustands sein soll, aus den anderen Sensorwerten abgeleitet werden (vgl. [Barton (2012)]). Der Einflussvektor wird hierbei *beobachtet* oder *geschätzt*. Ein *Predictor* leitet eine Größe allein aus vergangenen Zuständen ab. Ein *Estimator* verwendet neben diesen auch sensorisch erfasste Messwerte aus dem aktuellen Systemzustand.

Das Verwenden eines Zustandsbeobachters hat den Vorteil, dass am Träger keine spezielle Sensorik verbaut werden muss. Ein Ausfall kann rein durch den Einsatz von Algorithmen und der ohnehin gemessenen Systemzustände, wie etwa Höhe, Lage und Geschwindigkeit, erkannt werden. Dieser Ansatz ist daher auf jeden Träger übertragbar, jedoch bedingt er das Bilden der Bewegungsgleichungen und das Identifizieren der Koeffizienten des individuellen Trägersystems. Daher ist mit der Verwendung eines Zustandsbeobachters eine allgemeine und übertragbare Lösung der Ausfallerkennung gegeben, jedoch erfordert eine Übertragung einen hohen Arbeitsaufwand. Weiterhin sind für das Bilden der Bewegungsgleichungen, die analytische Modellbildung eines Kalman-Filters und die Identifikation der Systemkoeffizienten tiefgreifende Kenntnisse in den Gebieten Signal- und Systemtheorie, Flugmechanik und Regelungstechnik nötig. Aus diesen Gründen ist das Konzept eines auf analytischen Modellen basierenden Zustandsbeobachters für die Verwendung im angestrebten System der adaptiven Flugsteuerung unattraktiv.

3.1.3. Neuronale Netze

Eine Vielzahl der heutigen Systeme basieren auf Kalman-Filtern und analytischer Modellbildung. Diese birgt jedoch verschiedene Probleme. Bei sehr komplexen Systemen, wie etwa das eines Flugzeugs, ist die Modellbildung zunehmend schwieriger. Entweder das Modell bietet nicht den nötigen Realitätsgrad, die Parameteridentifikation ist zu zeit- und kostenaufwendig oder die Modellbildung ist aufgrund von zu geringem Wissen über das zu modellierende System nicht möglich. Selbst wenn ein Modell existiert, bzw. gebildet wurde, kommt es bei diesem unausweichlich zu Ungenauigkeiten aufgrund von unbekannten Einflüssen, Vereinfachung und Idealisierung, Linearisierung oder ungenauen Parametern (vgl. [Patan (2008)]), woraus ungenügende Ergebnisse resultieren können.

Eine alternative Methode zur analytischen Modellbildung ist die Verwendung von neuronalen Netzen. Diese können als universelle Funktionsapproximatoren komplexe dynamische Prozesse abbilden (vgl. [Horvath (2002)], [Trischler und D'Eleuterio (2015)], [Behera und Rana (2014)] und [Funahashi und Nakamura (1993)]), welche für deterministische Algorithmen nicht ausreichend definiert sind [Patan (2008)]. Neuronale Netze bieten den Vorteil, dass sie kein a

priori aufgestelltes analytisches Modell benötigen. Stattdessen können sie das Verhalten des Systems anhand von realen Eingangs- und Ausgangsdaten anlernen und nachbilden. Dabei werden auch die funktionalen Zusammenhänge und Korrelationen zwischen den Daten erkannt und nachgebildet. Hierdurch eignen sie sich als flexibles Werkzeug zur Modellierung von dynamischen Prozessen, welche zu komplex für ein analytische Lösungen sind oder deren analytische Modellierung sehr aufwendig ist. Abbildung 3.6 zeigt schematisch die Generierung eines Restwertes mit Hilfe von neuronalen Netzen.

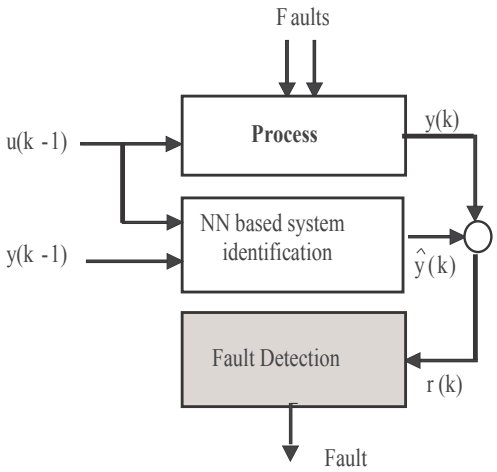

Abbildung 3.6.: Aus [Fekih u. a. (2007)]: Restwerterzeugung durch neuronales Netz, welches das Systemverhalten nachbildet

Für das Problem der Identifizierung von Fehlern und Fehlverhalten bieten sie einen weiteren Vorteil. Sie können als Klassifikator verwendet werden, welcher Ausfälle und deren Ausfalltypen aus dem generierten Restwertsignal identifiziert. So kann eine Ausfallerkennung entwickelt werden, welche auf allen Ebenen ab der Sensordatenerfassung bis zur Entscheidungsebene, ob ein Fehler vorliegt oder nicht, ohne einen auf das Problem hin entwickelten deterministischen Algorithmus oder analytisches Modell auskommen. Neuronale Netze können in der Ausfallerkennung somit angewandt werden, um das Problem der Zustandsschätzung und das der Fehlerklassifikation zu lösen [Patan (2008)].

Die Idee zur Verwendung von neuronalen Netzen zur Ausfallerkennung in der Flugsteuerung trat etwa Mitte der 1990er Jahre in Erscheinung. Einen der ersten Ansätze lieferten Silvestri et al. [Silvestri u. a. (1994)]. Sie verwendeten ein Feed-Forward Netz mit einem Sliding-Window, um den Ausfall von Sensoren zu erkennen und zu kompensieren. Das Netz wurde während des Fluges mit dem Input des Piloten und den Sensorwerten antrainiert (*online*-Lernen). Der Ausfall

wurde detektiert, wenn der Restwert aus der Differenz zwischen Sensorwert und Ausgabe des Netzes einen Grenzwert überschritten hatte. In diesem Moment wurde das Training beendet und die Gewichtungen und Biase des Netzes eingefroren. Fortan wurde anstelle des ausgefallenen Sensors die Ausgabe des neuronalen Netzes, als Zustandswert verwendet. Der Ansatz lieferte je nach Zeitpunkt des Auftritts und damit verbundener Lernzeit gute Ergebnisse (vgl. [Silvestri u. a. (1994)]). Cork et al. haben den Ansatz aufgegriffen, und auf UAVs angewandt. Sie verwendeten ebenfalls Feed-Forward Netze mit Sliding-Windows und stellten fest, dass diese vielversprechende Ergebnisse liefern [Cork u. a. (2005)].

Dieser Ansatz ist für das in der vorliegenden Arbeit gegebene Problem nicht direkt anwendbar, da Silvestri et al. eine zu große Divergenz zwischen Sensor und Netz als Ausfall des Sensors interpretiert haben. Jedoch kann er übertragen werden, so dass die Divergenz als Ausfall einer Steuerfläche interpretiert werden kann.

Jüngere Arbeiten beschäftigen sich zumeist mit der Anwendbarkeit von neuronalen Netzen zur Ausfallerkennung bei verschiedenen Flugzeugkonfigurationen. Zhengdao und Weihua haben ein auf Radialen Basisfunktionen basiertes Netz (*RBFNN*) präsentiert, welches für eine effektive Ausfallerkennung in Kampfflugzeugen eingesetzt werden kann [Zhang und Zhang (2009)]. Das Netz wurde als Zustandsschätzer für eine *one-step-ahead prediction* verwendet. Für die Identifizierung des Ausfalltyps wurde eine regelbasierte Tabelle verwendet. Sie gehen in ihrem Fazit explizit darauf ein, dass für das verwendete Flugzeug kein a priori aufgestelltes analytisches Modell benötigt wurde. Fekih et al. haben verschiedene NARMAX (auch NARX, engl. *Nonlinear Autoregressive Moving Average with eXogenous input*, [Connor u. a. (1992)]) Netztypen zur Ausfallerkennung für eine Boeing 747 verwendet und erhielten gute Ergebnisse: *"The NN was able to approximate the complicated and highly nonlinear model of the B747 aircraft."* [Fekih u. a. (2007)]. Shan und Hou verwendeten ebenfalls NARMAX Netze [Shan (2016)]. Sie haben den Ausfall der Querruder eines in X-Plane simulierten UAVs erkannt, und daraufhin eine Rekonfiguration der Steuerflächenzuordnung vorgenommen, um eine weitere Steuerbarkeit aller Freiheitsgrade zu ermöglichen.

Die präsentierten Resultate zeigen, dass neuronale Netze einen probaten Ansatz zur Nachbildung von hochgradig nichtlinearem und komplexem dynamischen Flugverhalten darstellen. Für die vorliegende Arbeit macht die nicht-benötigte Modellierung des Flugverhaltens dieses Verfahren besonders attraktiv. Dieser Ansatz setzt einen Zustandsschätzer ohne die Verwendung von aufwendig zu modellierenden Kalman-Filtern um. Statt eines a priori aufgestellten analytischen Modells erlernt das Netz die mathematische Abbildung der Bewegungsgleichungen. Nachteilig ist hieran jedoch, dass die Verifizierbarkeit der korrekten Funktionalität gegenüber analytischen Modellen ungleich schwieriger ist [Schumann u. a. (2003)], [Soares u. a. (2006)].

3.2. Adaptive Steuerflächenzuordnung

Das Problem der Steuerflächenzuordnung (engl. *Control Allocation*) ist so alt wie das Problem der Steuerung eines Flugzeugs selbst. Jedoch wurde bereits von den Pionieren der Luftfahrtgeschichte frühzeitig erkannt, dass für jeden rotatorischen Freiheitsgrad eine separate Steuermöglichkeit vorhanden sein muss. Daraufhin haben sich die Konzepte der Querruder (engl. ailerons), Höhenruder (engl. elevators) und Seitenruder (engl. rudder) etabliert. Diese Konzepte dominieren die kommerzielle Luftfahrtechnik bis heute. Auch in den modernsten Verkehrsflugzeugen wie dem Airbus A350XWB oder dem Airbus A380 gilt diese feste Zuteilung der Aufgaben.

3.2.1. Virtuelle Steuersignale

Zur Steuerung eines Flugzeug hat ein Pilot typischerweise vier primäre Steuerorgane. Im einfachsten Fall steuert er direkt die Querruder, das oder die Höhenruder, das Seitenruder sowie dem Triebwerksschub, wodurch sich die gewünschten Bewegungen im dreidimensionalen Raum ergeben. Mit den Querrudern wird um die Längsachse gedreht (*Rollen*), mit den Höhenrudern um die Querachse (*Nicken*) und mit dem Seitenruder um die Hochachse (*Gieren*). Mit dem Triebwerksschub wird die translatorische Bewegung entlang der Längsachse gesteuert (Beschleunigung und Fluggeschwindigkeit). Neben diesen vier primären haben sich weitere Steuerflächen etabliert. Bei diesen als sekundäre Steuerflächen bezeichnete Aktorik handelt es sich meist um Hochauftriebshilfen, wie etwa *Slats* oder *Flaps*. Flugzeuge der Transportklasse wie die Boeing 747 besitzen insgesamt bis zu 20 unabhängige Aktuatoren [Gundy-Burlet und Bryant (2003)].

Für unkonventionelle Flugzeugkonfigurationen, wie etwa einem Blended-Wing-Body Flugzeug, existieren nicht immer dedizierte Steuerflächen für jeden der Freiheitsgrade. Zum Beispiel besitzt das AC2030 Modell der BWB-Projektgruppe der HAW Hamburg auf jeder Tragflächenseite so genannte *Elevons*, welche eine Kombination aus Ailerons und Elevators darstellen und deren Aufgaben äquivalent übernehmen. Die direkte Ansteuerung eines solchen Elevons stellt einen Piloten vor eine schwierige bis unmögliche Aufgabe, da eine unabhängige Steuerung der beiden Freiheitsgrade nicht möglich ist. Er muss eine unabhängige zweidimensionale Rotation durch eine jeweils abhängige eindimensionale Ansteuerung durchführen. Statt der direkten Ansteuerung müssen die Eingabesignale für das Rollen und das Nicken *gemischt* werden. So kann der Pilot weiterhin die zwei gewohnten und vertrauten Eingabekanäle bedienen. Dieser Ansatz ermöglicht es einem Piloten jegliche Art von Flugzeug steuern zu können, ohne dessen explizite Steuerflächen und deren Konfiguration kennen zu müssen. Unabhängig von der Art

und Anzahl der Steuerflächen verwendet der Pilot zur Bewegung des Flugzeugs auch hier lediglich seine vier primären Eingabemöglichkeiten. Durch diese Abstraktion stellen die Eingaben nicht länger den Ausschlag der Steuerflächen, sondern die Steuerung der Bewegungen in den Freiheitsgraden des Flugzeugs dar. Ein analoges Konzept wird für Fahrzeuge verwendet. Für gewöhnlich wird der laterale Freiheitsgrad über ein Lenkrad gesteuert, unabhängig davon, ob für die Umsetzung eine vordere Rad-Lenkung (Auto, Traktor), eine hintere Rad-Lenkung (Gabelstapler), eine Knick-Lenkung (Radlader) oder ein differentieller Kettenantrieb (Planierraupe) verwendet wird.

Die Abbildung von Freiheitsgraden auf vorhandene Steuerflächen wird als *Steuerflächenzuordnung* (engl. *Control Allocation*) bezeichnet. Sie ist von der Flugzeugkonfiguration, den am Flugzeug vorhandenen Steuerflächen und dem sogenannten *Flight Control Law* (*FCL*) abhängig. Die Steuerflächenzuordnung und das FCL werden durch das Design des Flugsteuerungssystems bestimmt.

Wie in Abschnitt 1.2 bereits motiviert, ist es für einen Piloten sehr hilfreich, wenn sich sein Steuerkonzept in einer Ausfallsituation nicht ändert. Für einen Autopiloten hat es den Vorteil, dass sich dessen Flight Control Law nicht ändern muss. Zum Beispiel, dass eine Kurve auch weiterhin durch ein Rollmoment eingeleitet wird. Wie dieses erzeugt wird, ist für den Autopiloten unerheblich. Abbildung 3.7 zeigt schematisch einen Regelkreis, bei welchem der Regler eine virtuelle Steuereingabe v ausgibt und eine Control Allocation aus diesem reale Ansteuerungen u für die Aktuatoren des technischen Systems ableitet. Die Dimension des Vektors u entspricht dabei der Anzahl der Aktuatoren. Eine geeignete Zuordnung zu finden, wird in der Steuer- und Regelungstechnik allgemein als *Control Allocation Problem* bezeichnet. Dieses wird in Abschnitt 3.2.5 detailliert beschrieben.

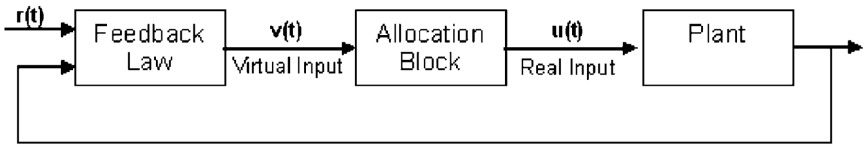

Abbildung 3.7.: Aus [Casavola und Garone (2010)]: Einführung eines virtuellen Steuersignals zur Abstraktion des Control Allocation Problems vom Regelsystem

In den nachfolgenden Abschnitten werden typische Konzepte und Techniken beschrieben, wie eine Steuerflächenkonfiguration adaptiv ausgelegt werden kann, so dass die ausgefallene Steuerleistung auf die intakten Steuerflächen umverteilt werden kann.

3.2.2. Redundante Steuerflächen

Eine aus algorithmischer Sicht einfache Methode, den Ausfall von primären Steuerflächen zu kompensieren, ist die Umverteilung auf redundant vorhandene Aktuatoren (vgl. [Casavola und Garone (2010)]). Jedoch kommt dies in Flugzeugen der Transportklasse für gewöhnlich nicht vor[1], da die Systeme hierdurch zu teuer, schwer und ineffizient werden. Ausnahmen bieten mehrstrahlige Flugzeuge, bei denen der Ausfall eines Triebwerks durch die verbleibenden ausreichend kompensiert werden kann. Park und Rokhsaz beschreiben die aerodynamische Wirkung von in Winglets integrierten vertikalen Klappen [Park und Rokhsaz (2003)], welche ein direktes Giermoment erzeugen können, um ein ausgefallenes Seitenruder zu kompensieren.

Dieser Ansatz ist innerhalb der Flugsteuerung algorithmisch einfach umzusetzen. Jedoch hat er den Nachteil, dass der Träger über explizit redundante Steuerflächen verfügen muss. Somit bietet der Ansatz keine Übertragbarkeit.

3.2.3. Prioritäten-basierte Zuordnungstabelle

Es werden jedoch nicht zwingend explizit redundant ausgelegte Steuerflächen benötigt. Stattdessen kann auf eine intrinsische Redundanz und eine unabhängige Ansteuerung der Steuerflächen zurückgegriffen werden. Diese erzeugen unausweichlich gekoppelte Bewegungsformen, so dass der Ausschlag eines einzelnen Ruders immer eine Bewegung in allen Freiheitsgraden herbeiführt. Man betrachtet hierbei sowohl die gewünschten Bewegungsformen in den Freiheitsgraden, als auch die einzelnen Steuerflächen. Fällt eine der Steuerflächen aus, können die intakten so in einer Weise angesteuert werden, dass weiterhin Bewegungen in allen Freiheitsgraden induziert werden können.

Sei beispielhaft ein konventionelles Flugzeug mit zwei Querrudern, zwei Höhenrudern, einem Seitenruder und zwei Triebwerken gegeben. Der Pilot steuert mit seinen Eingabemöglichkeiten die vier Freiheitsgrade Rollen, Nicken, Gieren und den Schub, bzw. Beschleunigung. Jedoch bietet das Flugzeug sieben Aktuatoren, welche dem Flugsteuerungssystem zur Verfügung stehen. Bei dem Prinzip der Prioritäten-basierten Zuordnungstabelle werden jedem Freiheitsgrad primäre, sekundäre, tertiäre, etc. Steuerflächen zugeteilt, welche die jeweiligen Zustandsveränderungen in diesen erzeugen sollen.

Die Tabelle 3.1 zeigt eine mögliche Konfigurationstabelle (engl. *Allocation Table*) für das gegebene Flugzeug. Im fehlerfreien Zustand werden für alle Freiheitsgrade die für sie primären Steuerflächen verwendet. Somit führen differentiell angesteuerte Querruder die Rollbewegung,

[1] Landeklappen, die ähnlich zu den Querrudern verbaut sind, lassen sich typischerweise nur nach unten ausfahren. Somit zeigen sie keine direkte Redundanz zu den Querrudern

symmetrisch angesteuerte Höhenruder die Nickbewegung, das Seitenruder die Gierbewegung aus. Symmetrisch angesteuerte Triebwerke erzeugen den Schub.

	Diff. Aileron	Sym. Elevator	Rudder	Sym. Thrust	Sym. Aileron	Diff. Elevator	Diff. Thrust
Rollen	Primär		Tertiär			Sekundär	
Nicken		Primär		Tertiär	Sekundär		
Gieren	Tertiär		Primär				Sekundär
Schub		Sekundär		Primär			

Tabelle 3.1.: Allocation Table zur Umsetzung der virtuellen Steuersignale je nach Prioritäten der Steuerflächen zu den jeweiligen Freiheitsgraden

Kommt es zu einem Ausfall eines oder beider Querruder, wird für das Rollen auf eine differentielle Ansteuerung der Höhenruder umgeschaltet. Diese übernehmen fortan die Steuerung beider Freiheitsgrade und werden somit zu Elevons. Fällt das Seitenruder aus, kann dies durch differentiell angesteuerte Triebwerke kompensiert werden. Falls die Höhenruder ausfallen, kann mit symmetrisch angesteuerten Querrudern weiterhin ein Nickmoment erzeugt werden. Für die Erzeugung der translatorischen Bewegung entlang der Längsachse gibt es keine effektive Alternative zu den Triebwerken.

Jedoch kann bei ausreichender Höhe potentielle in kinetische Energie umgewandelt werden. Die Änderung der Höhe erfolgt standardmäßig durch die Höhenruder. Es kann zwar nicht länger an Energie hinzugewonnen werden, jedoch kann durch die Variation der Höhe auch bei ausgefallenen Triebwerken noch die Fluggeschwindigkeit beeinflusst werden. Der umgekehrte Fall ist ebenfalls möglich. Durch Erhöhen des Triebwerkschubs wird neben kinetischer auch potentielle Energie aufgebaut, da durch eine höhere Geschwindigkeit mehr Auftrieb erzeugt wird. Durch die Veränderung des Triebwerkschubs wird inhärent die Höhe beeinflusst. Die Triebwerke könnten somit als tertiäre Steuermöglichkeit des Nickens, bzw. der Höhe, dienen.

Aus der allgemeinen Bewegungsgleichung für die Seitenbewegung (vgl. [Nguewo (2014), McLean (1990)]) geht hervor, dass die Bewegungen des Rollens und des Gierens inhärent miteinander gekoppelt sind. Ein gewünschtes Rollen induziert somit immer auch ein unerwünschtes Gieren und umgekehrt. In besonders kritischen Situationen, wenn zum Beispiel mehrere Steuerflächen ausfallen, können daher differentiell angesteuerte Querruder sowie das Seitenruder die aerodynamische Funktion der jeweils anderen Steuerfläche(n) übernehmen.

Das Prinzip der Priorisierung beschreibt hierbei lediglich, dass eine a priori definierte Tabelle von Steuerflächenzuordnungen existiert. Diese gibt vor, welche Zuordnung in welchem Fehlerfall, bzw. welchem Ausfall, verwendet wird. Diese Tabelle wird durch Erfahrung und Expertenwissen vorgegeben und fest in die Flugsteuerung einprogrammiert. Das Vorgeben einer definierten Zuordnungstabelle ist ein typischer Ansatz für die Control Allocation einer

fehlertoleranten Flugsteuerung (vgl. [Gundy-Burlet und Bryant (2003)], [Gundy-Burlet (2004)], [Zhang und Zhang (2009)]).

3.2.4. Daisy Chaining

Der Ansatz des *Daisy Chaining* ist eine Erweiterung zur Prioritäten-basierten Zuordnungstabelle. Bei diesem werden die statischen und dynamischen Eigenschaften der Aktuatoren mit einbezogen. Wenn vom Flugregler eine Dynamik gefordert wird, die eine Steuerfläche alleine nicht liefern kann, wird eine weitere hinzugezogen, um die Steuerwirkung zu vergrößern (vgl. [Johansen und Fossen (2013)], [Gundy-Burlet und Bryant (2003)]).

Die Methode wurde Mitte der 1990er Jahre von Berg et al. [Berg u. a. (1996)] sowie Buffington und Enns vorgestellt [Buffington und Enns (1996)]. Die Aktuatoren wurden in Gruppen eingeteilt. Wenn die erste Gruppe nicht ausreicht, um die vom Flugregler gewünschte Drehrate zu erzeugen, wird die nächste Gruppe aktiviert. Kim et al. haben den Ansatz um eine Ausfallkompensation erweitert [Kim u. a. (2013)]. Im originalen Ansatz von Berg et al. wurden die letzten Gruppen nicht durch weitere unterstützt, wodurch ein Ausfall dieser nicht kompensiert werden kann. Beim von Kim et al. präsentierten Ansatz gewinnen Gruppen, in denen Aktuatoren ausgefallenen sind, an Priorität, wodurch die dynamisch nachgelagerten Gruppen die ausgefallene Steuerleistung wiederherstellen können. Durch die dynamische Priorisierung können Ausfälle kompensiert werden.

3.2.5. Optimale Steuerflächenzuordnung

Der Ansatz einer optimalen Steuerflächenzuordnung sieht nicht länger vor, den Steuerflächen dedizierte Aufgaben und starre Zuständigkeiten zuzuteilen. Stattdessen werden die virtuellen Steuersignale der Freiheitsgrade anhand der Effektivitäten der einzelnen Steuerflächen auf diese verteilt.

Für die nachfolgenden Erläuterungen wird die in Abschnitt 2.2 dargestellte Bewegungsgleichung herangezogen. Aus der Systemmatrix A und dem Zustandsvektor x lässt sich ableiten, wie sich die Eigenbewegung des Flugzeugs ohne äußere Einflüsse oder Eingaben verhält. Aus der Steuermatrix B und dem Steuervektor u ergibt sich die resultierende Bewegung anhand der Ruderstellungen. Der dritte Summand bestimmt die Reaktion des Flugzeugs anhand von Wind und Turbulenzen. Für die nachfolgenden Betrachtungen ist sind der erste und dritte Summand nicht relevant und werden vernachlässigt. Der Steuervektor u enthält die *Sollwerte* der Aktuatoren, sie werden nachfolgend jedoch wie deren *Istwerte* behandelt.

Der Steuervektor u enthält die zum Zeitpunkt t am Flugzeug anliegenden Klappenstellungen. Die Koeffzienten der Steuermatrix B geben an, wie sich der Flugzustand in Abhängigkeit der Steuereingaben verändert. B bestimmt demnach, wie das Flugzeug auf bestimmte Stellungen der Steuerflächen reagiert. Aus dieser Abhängigkeit können die Effektivitäten der einzelnen Steuerflächen für die gewünschten Bewegungen abgeleitet werden. Die Steuermatrix B wird daher auch als *Effektivitätsmatrix* bezeichnet.

Die referenzierte Zustandsgleichung berechnet einen Folgezustand $\dot{x}$ aus dem aktuellen Zustand und einer gegebenen Steuereingabe. Das Ziel bei der Control Allocation ist hingegen, einen Steuervektor u für einen gewünschten Folgezustand $\dot{x}$ zu finden. Gleichung 3.2.1 stellt diesen Zusammenhang dar:

$$G^{-1}(\dot{x}) = \underline{u}(t) = \underline{\underline{B^{-1}}} * (\underline{\dot{x}}(t) - \underline{\underline{A}} * \underline{x}(t)) \tag{3.2.1}$$

Die Gleichung soll lediglich die Herangehensweise verdeutlichen und stellt das Problem vereinfacht dar. In realen Anwendungen wird in der Zustandsraumdarstellung neben der Zustandsgleichung 2.2.1 auch eine *Ausgangsgleichung* mit dem Ausgang y modelliert. Für die Control Allocation muss diese abgeleitet werden und die beiden Gleichungen müssen über ein lineares Gleichungssystem kombiniert und gelöst werden.

Das Lösen dieses Gleichungssystems ist aus mathematischer Sicht ein simples Problem. Die Schwierigkeit in der Control Allocation besteht darin, dass die Aktuatoren, welche die Steuerflächen bewegen, physikalischen Limitierungen in den Stellgeschwindigkeiten (engl. *rate limit*), den Ausschlägen (engl. *deflection limit*) oder den aerodynamischen Auswirkungen (engl. *saturation of effect*) unterliegen (vgl. [Frost u. a. (2010)]).

Diese Limitierungen geben Bedingungen für das Lösen des Gleichungssystems vor. Hierdurch ergeben sich die Möglichkeiten, dass:

- viele Lösungen existieren
- genau eine Lösung existiert
- keine Lösung existiert

Das Finden der "*besten*", bzw. unter verschiedenen externen Vorgaben "*optimalen*", Lösung wird als *Control Allocation Problem* bezeichnet [Frost u. a. (2010)]. Die externen Vorgaben können etwa sein, dass die Bewegung zum Folgezustand $\dot{x}$ möglichst schnell, Bauteil schonend, energiesparend oder sanft vollzogen werden soll. Diese Hängen von den Einsatzgebieten der Flugzeuge ab. Kampfflugzeuge erfordern eine hohe Dynamik und schnelles Reaktionsvermögen, Passagierflugzeuge hingegen sollen gemäßigte Bewegungen umsetzen. Dies wird durch die sogenannten *Flying Qualities* und *Handling Qualities* bestimmt, welche beim gesamten

Entwicklungsprozess des Flugzeugs und der Avionik eingezogen werden (vgl. [de Castro (2001)]).

In seiner allgemeinen Form kann das Control Allocation Problem so verstanden werden, dass es versucht, den insgesamt benötigten kollektiven Steueraufwand möglichst optimal auf alle zur Verfügung stehenden Aktuatoren verteilt (vgl. [Frost u. a. (2010)]). Durch die Bedingungen der Aktuatoren selbst und den externen Vorgaben wird das Finden eines optimalen Steuervektors u zu einer komplexen Aufgabe, welches mathematisch ein Optimierungsproblem darstellt. Besonders bei hochgradig überaktuierten Systemen ist das Finden einer optimalen Lösung schwierig. In der Flugmechanik ist dies häufig bei Kampfflugzeugen der Fall. Diese sollen extrem dynamisch und wendig sein, was typischerweise durch eine Vielzahl von unabhängig ansteuerbaren Steuerflächen erreicht wird. Für das Finden einer Lösung existieren verschiedene Methoden, wie zum Beispiel *linear programming*, *direct control allocation* oder *pseudo-inverse-redistribution* (vgl. [Casavola und Garone (2010)]). Durham et al. geben eine ausführliche Übersicht über die Control Allocation im Allgemeinen und im Besonderen für die Flugsteuerung [Durham u. a. (2017)].

Dieser Ansatz der optimalen Steuerflächenzuordnung kann in jeder Flugsituation angewandt werden. Wenn eine oder mehrere Steuerflächen ausfallen, kann der benötigte Steueraufwand auf die noch intakten Steuerflächen verteilt und trotz der Einschränkungen die noch maximal mögliche Performance des in Teilen ausgefallenen Flugzeugs ausgeschöpft werden. Bei einer strukturellen Beschädigung müssen die Koeffizienten der Steuermatrix B so angepasst werden, dass sie die verbliebenen Effektivitäten der Steuerflächen widerspiegeln, bzw. die Struktur der Steuermatrix muss angepasst werden. Bei einem Komplettausfall (Einfrieren) einer Steuerfläche muss deren Komponente des Steuervektors u ebenfalls eingefroren werden. Bei einem eigenständigen Schwingen, Nachlaufen oder Zittern muss dieses ebenfalls im Steuervektor nachgebildet werden.

Der Vorteil gegenüber Daisy Chaining ist bei diesem Verfahren, dass auch Fehlerfälle abgedeckt werden, die in den a priori durch Expertenwissen definierten Fehlermodellen und der festgelegten Allocation Table nicht abgedeckt werden. Weiterhin werden in der Allocation Table keine Effektivitäten berücksichtigt. Zum Beispiel müsste, wenn differentiell angesteuerte Höhenruder ein Rollmoment induzieren sollen, diese mit einem deutlich höheren virtuellen Steuersignal gespeist werden, da sie eine geringere Effektivität bei der Erzeugung eines Rollmoments aufweisen, als die Querruder. Nachteilig wirken hier hingegen die deutlich komplexeren Algorithmen, welche gegenüber einfachen Lookup-Tabellen erheblich größeren Rechenaufwand, Speicherplatz und Implementationsaufwand erfordern.

Ein weiterer Nachteil ist, dass analog zur Ausfallerkennung das mathematische Modell des Flugverhaltens vorhanden sein muss. Hier jedoch in umgekehrter Form, da nicht die Reaktion auf bestimmte Steuereingaben, sondern die benötigten Steuereingaben für eine gewünschte Reaktion gesucht wird. Es ist daher eine Form des *model inverse control* [Frost u. a. (2010)], bei welchem im Allgemeinen ein System anhand seiner inversen Übertragungsfunktion gesteuert wird. Auch hier kann das Modell durch analytische Methoden oder neuronale Netze gebildet werden, wobei die gleichen Vor- und Nachteile, wie bei der Modellbildung zur Ausfallerkennung gelten. Beispiele für Umsetzungen mit analytischen Modellen sind in [Casavola und Garone (2010)], [Johansen (2004)] und [Härkegård und Glad (2005)] gegeben, Beispiele für Umsetzungen mit neuronalen Netzen in [Hovakimyan u. a. (2001)] und [Ippolito u. a. (2007)].

3.3. Vergleichbare Systeme

Das zu entwickelnde System der vorliegenden Arbeit soll eine klare Trennung der Verantwortlichkeiten zwischen Flugregler, Flugsteuerung und Ausfallerkennung umsetzen. Dies soll die Austauschbarkeit und eine mögliche Verteilung der Komponenten auf verschiedene Computersysteme vereinfachen. In der Literatur ist die Unterteilung von Flugregelung, Ausfallerkennung und Steuerflächenzuordnung bei weitem nicht immer in unabhängige Systeme mit klaren Grenzen und Zuständigkeiten unterteilt. Shin et al. schreiben hierzu in [Shin u. a. (2006)]:

> "It is fairly common for integration of failure detection and accommodation systems to be problematic if they are designed separately."

Dies gilt besonders für reale Systeme, die nicht nur wissenschaftliche Relevanz haben. Wenn ein System hochsicher sein muss, wird es für genau sein Einsatzgebiet hin entwickelt und optimiert. Hierdurch entstehen zumeist hochintegrierte Systemarchitekturen und Insellösungen, bei welchen die einzelnen Komponenten starken Kopplungen unterliegen. Zwar lassen sich hier auch einzelne Techniken in den Systemkomponenten identifizieren, sie sind jedoch ungleich stärker miteinander verzahnt. Damit ist gemeint, dass nicht länger eine klare Grenze zwischen Flugregler (Führung des Systems) und Flugsteuerung (Umsetzen der Führung) gezogen werden kann. Der Flugregler benötigt ein größeres Wissen über die Arbeitsweise der Flugsteuerung, den Fehlerzustand und den am Flugzeug vorhandenen Aktuatoren. Hierdurch sind die Systeme zwar schwerer auszutauschen, jedoch kann der Ausfall einer Steuerfläche so auch besser kompensiert werden. Für eine optimale Kompensation kann es hilfreich sein, dass der Flugregler sein Flight Control Law dahingehend umstellt, dass eine bestimme Bewegung anders vollzogen wird. Wenn ihm zum Beispiel bekannt ist, dass das Flugzeug Querruder

besitzt und diese ausgefallen sind, soll er nicht länger versuchen, enge Kurven zu fliegen, bzw. diese früher beginnen, da das Flugzeug in der Erzeugung von Rollmoment mit reduzierter Performance arbeitet [Ducard (2009)].

Ducard schreibt zum Vorteil der klaren Trennung zwischen Flugregler und Control Allocator in [Ducard (2009)]:

> *"The biggest advantage of using a control allocation technique is that actuator failures can be compensated without the need for modifying the flight control laws"*

Zum Nachteil schreibt er:

> *"(For the flight controller,) ... the dynamics and limitations of the actuators after a failure are not taken into account in the control laws. This means that the controller will still attempt to achieve the original system performance even though the actuators are not capable of achieving it"*

Die beiden Aussagen stehen im klaren Widerspruch zueinander. Es ergibt sich ein Trade-off, welcher für jedes System einzeln behandelt werden muss. Ein wiederverwendbarer Flugregler erfordert eine Trennung und einseitige Abhängigkeit in Richtung Control Allocator in Form von virtuellen Steuersignalen. Eine *"bessere"* fehlertolerante Flugsteuerung erfordert, dass der Flugregler Kenntnis vom Ausfall und der Steuerflächenzuordnung hat. Für die vorliegende Arbeit wurde die Anforderung definiert, dass eine klare Trennung zwischen Flugregler und Steuerflächenzuordnung besteht und der Flugregler keine Kenntnis von einem Ausfall erhält und der Fehler für diesen maskiert wird.

Nachfolgend sind zur Referenz Systeme aus der Literatur aufgeführt, bei denen eine klare Einteilung in Flugregler und Steuerflächenzuordnung schwer möglich ist, bzw. die Zuständigkeiten verschmelzen und der Flugregler bei einem Ausfall in ein anderes Flight Control Law umschaltet.

Calise und Kim haben 1997 eine umfassende Systemarchitektur zur adaptiven Flugsteuerung präsentiert [Kim und Calise (1997)], welche Modellierungsunsicherheiten und strukturelle Veränderungen des Flugzeugs durch ein online lernendes neuronales Netz kompensiert.

Abbildung 3.8 zeigt den Regelkreis eines Freiheitsgrades des Systems. Für den Regler (rot) erscheint das System linear. Dies wird durch eine Kombination aus *Model Inversion* und *Feedback Linearization* erreicht. Die Feedback Linearization ist ein Verfahren, bei dem ein nichtlineares in ein lineares System überführt wird (vgl. [Müller (2008)]). Dies wird durch eine Art der Model Inversion möglich, bei welchem die Übertragungsfunktion des Systems mit Hilfe des Feedbacks invertiert wird. Das zu regelnde System erscheint für den Regler somit als linear und kann wie

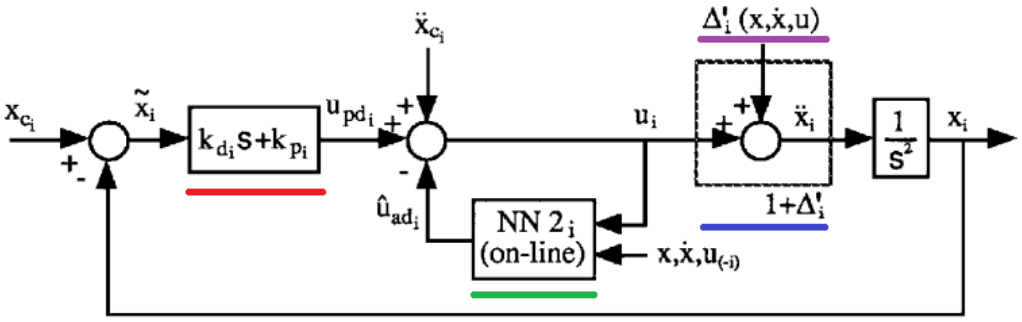

Abbildung 3.8.: Aus [Kim und Calise (1997)]: Nichtlineare Regelung durch Feedback Linearization und Model Inversion mittels neuronalen Netzen

ein solches geregelt werden. Dies hat den Vorteil, dass auf ein aufwendiges *Gain-Scheduling* verzichtet werden kann. Mit diesem müsste das System in mehreren Arbeitspunkten linearisiert werden, wobei für jeden Arbeitspunkt Reglerparameter bestimmt werden müssten.

Der Index i gibt die Nummer des Freiheitsgrades (Rollen, Nicken, Gieren) an und kann nachfolgend vernachlässigt werden. Die Model Inversion (blau) lässt das System durch eine Feedback Linearization (lila) für den Regler als lineares erscheinen. Das virtuelle Steuersignal u, mit welchem das System gespeist wird, setzt sich aus drei Teilen zusammen, wobei der Anteil $\ddot{x}_c$ hier vernachlässigt wird. Der Anteil u_{pd} wird von einem linearen PD-Regler (rot) erzeugt. Durch eine sogenannte *Inverse Model Augmentation* (grün) werden Ungenauigkeiten und Fehler bei der Modellierung der inversen Übertragungsfunktion dynamisch ausgeglichen. Sie liefert einen zusätzlichen Anteil $\hat{u}_{ad}$ auf das virtuelle Steuersignal u.

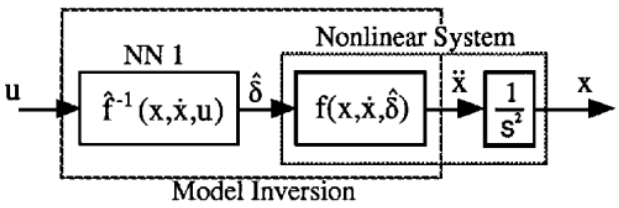

Abbildung 3.9.: Aus [Kim und Calise (1997)]: Model Inversion durch offline-trainiertes neuronales Netz

Die Model Inversion ist in Abbildung 3.9 dargestellt. Die Abbildungsfunktion von u auf $\hat{\delta}$ stellt die Control Allocation dar. An der Systemarchitektur ist hervorzuheben, dass es zwei neuronale Netze verwendet. Das eine wird zur Model Inversion genutzt und wird offline anhand von aufgezeichneten Simulatordaten trainiert. Das zweite bildet die Augmentation. Es hat den

Zweck, Unsicherheiten in der Modellierung des inversen Modells (NN 1) auszugleichen. Daher wird es online im Flug trainiert.

Durch die Model Inversion und die Feedback Linearization ergibt sich für den Regler ein lineares System, dessen Realitätsnähe durch die adaptive Augmentation im laufenden Betrieb verbessert wird. Dieser Ansatz hat in der wissenschaftlichen Szene großen Anklang gefunden und bildet die Basis für viele veröffentlichte Abhandlungen.

Das Intelligent Flight Control System (*IFCS*) des NASA Dryden Flight Research Centers war ein über mehrere Jahre angelegtes Forschungsprojekt, um *"einen revolutionären technologischen Durchbruch bei der Flugsteuerung zu erzielen, welcher die Leistung des Flugzeugs sowohl unter normalen als auch unter Störungsbedingungen effizient optimieren kann"* [NASA (2014)]. Der Ansatz von Calise und Kim diente als grundsätzliche Basis. Adaptive Flugsteuerungssysteme auf Basis dieser Systemarchitektur wurden etwa für ein F-15B Kampfflugzeug [Williams-Hayes (2005), Bosworth und Williams-Hayes (2007)] oder auch ein UAV [Ippolito u. a. (2007)] entwickelt. Weiterhin haben Calise und Rysdik den Ansatz verwendet, um Helikopter und Tiltrotoren zu steuern [Calise und Rysdyk (1998)], [Calise u. a. (2004)]. Drozeski hat mit dem Ansatz einen unbemannten Modellhelikopter gesteuert [Drozeski (2005)].

4. Ansatz

Das vorliegende Kapitel beschreibt die Herangehensweise und den entworfenen Ansatz zur Umsetzung der adaptiven Flugsteuerung. Weiterhin werden die benötigten Softwarekomponenten beschrieben.

Aus den gegebenen Anforderungen und den analysierten Konzepten wurde ein Ansatz gebildet, mit welchem eine fehlertolerante Flugsteuerung durch eine dynamische und adaptive Reallokation der Steuerflächen realisiert werden soll. Durch die Anforderungen ist vorgegeben, dass die funktionalen Einheiten Flugregelung, Flugsteuerung und Ausfallerkennung in klar separierbare Komponenten aufgeteilt werden sollen. Der Flugregler hat die Aufgabe der Führung und Navigation des Flugzeugs, die Control Allocation die des Steuerns. Durch die Trennung soll das Single-Responsibility-Prinzip sowie das Prinzip des Separation of Concerns gestärkt werden, wodurch die Komponenten austauschbar und wiederverwendbar sind.

4.1. Gesamtsystem adaptive Flugsteuerung

Die adaptive Flugsteuerung wird durch die Komponenten Fault Detection und Control Allocation realisiert. Die Fault Detection hat die Aufgabe, den Integritätszustand der Steuerflächen zu überwachen. Die Control Allocation hat die Aufgabe, unabhängig vom Integritätszustand, die vom Flugregler gewünschten Bewegungen und Drehmomente durch die intakten Steuerflächen zu erzeugen.

Abbildung 4.1 zeigt die Ansätze zur Realisierung der beiden Komponenten im erweiterten Regelkreis. Das mathematische Modell des Flugzeugs soll ein vortrainiertes rekurrentes neuronales Netz bilden. Dieses erzeugt einen Restwert, welcher durch einen Zustandsautomaten evaluiert wird. Die dynamische Reallokation soll durch einen Allokationsalgorithmus umgesetzt werden, welcher die tatsächlich verbliebenen Integritätszustände durch einen Agenten anlernt. Die Komponenten Fault Detection und Control Allocation ergeben im Verbund eine adaptive Flugsteuerung, wodurch das Flugzeug bei Ausfall einer Steuerfläche mit unverändertem Flight Control Law in allen Freiheitsgraden manövrierbar bleibt. In den Abschnitten 4.2 und 4.3 werden die einzelnen Ansätze jeweils ausführlich beschrieben. Es wird dabei insbesondere auf den Prozess der Entscheidungsfindung eingegangen und die Auswahl der Techniken begründet.

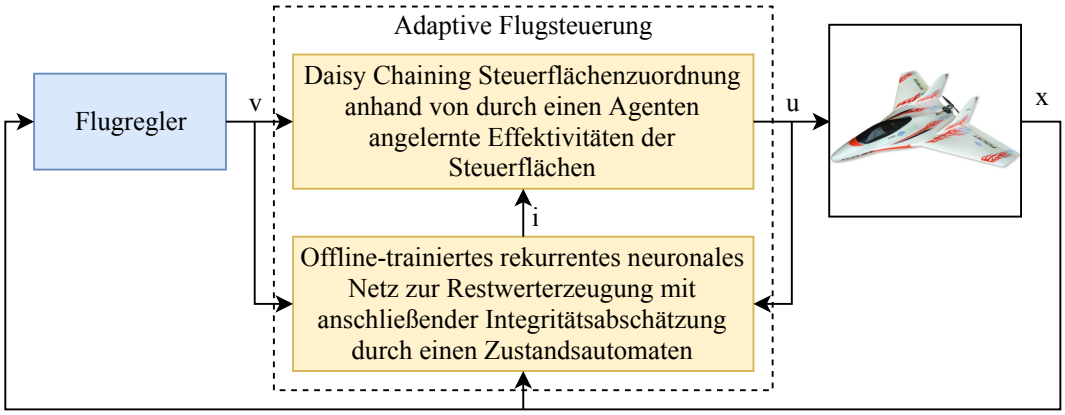

Abbildung 4.1.: Ansätze der zu realisierenden Komponenten Fault Detection und Control Allocation

Bei Ausfall einer Steuerfläche verbleibt diese in einer festen Position und die Steuerbarkeit dieser geht verloren. Hierdurch wird deren Komponente im Steuervektor u unveränderlich. Um den Ausfall zu kompensieren, muss ein geeignetes u gefunden werden, welches dennoch zum gewünschten x_{ref} führt, ohne dass der Flugregler seine Flight Control Laws ändern muss und ein anderes v erzeugt. Dies soll durch eine Reallokation geschehen, bei dem die Aufgabe der Momenterzeugung der ausgefallenen Steuerfläche auf die verbliebenen intakten verteilt wird.

Der eigene Ansatz macht, anders als in Abschnitt 2.2 beschrieben, keine Unterscheidung zwischen Längs- und Seitenbewegung. Für die folgenden Erläuterungen gilt somit, dass lediglich eine Bewegungsgleichung für alle Freiheitsgrade vorhanden ist. Die Zustandsgleichung des erweiterten Regelkreises mit adaptiver Flugsteuerung ergibt sich wie folgt:

$$\underline{\dot{x}}(t) = \underline{\underline{A}} * \underline{x}(t) + \underline{\underline{B}} * (\underline{u}(t) \circ \underline{i}(t)) \tag{4.1.1}$$

Der Vektor i enthält den Integritätszustand der Steuerflächen und beschreibt damit deren verbliebene Effektivität. v und u haben die gleiche Dimension und werden komponentenweise multipliziert (*Hadamard-Produkt*). Der Wertebereich von i_i definiert als $0 \leq i_i \leq 1$. Bei voller Integrität hat i für jeden Aktuator den Wert 1. Bei Totalausfall einer Steuerfläche enthält i für deren Komponente den Wert 0. Durch die Information über den Integritätszustand kann die Control Allocation ableiten, welche Steuerflächen sie für die Umsetzung der vom Flugregler gewünschten Bewegung verwenden kann. Die Abbildung von v nach u ist damit von i abhängig. Hierdurch kann das Flugzeug trotz des Ausfalls einer Steuerfläche weiterhin

in allen Freiheitsgraden gesteuert werden. Der Zustand x lässt sich somit weiterhin an den gewünschten x_{ref} annähern.

Den Vektor i liefert zunächst die Komponente Fault Detection. Diese berechnet den Folgezustand $\dot{x}$ anhand eines mathematischen Modells des Flugzeugs. Der Ausfall einer Steuerfläche liegt vor, wenn gemessener (tatsächlicher) und berechneter (erwarteter) Folgezustand über einen Zeitraum stark divergieren. Aus dieser Divergenz wird geschlossen, welchen Integritätszustand eine Steuerfläche besitzt und welches i sich fortan für das Flugzeug ergibt.

Unabhängig von dem von der Fault Detection gelieferten Integritätszustand i, wird innerhalb der Control Allocation ein Lernalgorithmus verwendet, mit welchem die tatsächlichen verbliebenen Effektivitäten identifiziert werden sollen. Hierdurch soll das in Abschnitt 1.3 beschriebene Vorgehen zum *ad-hoc* Anlernen der benötigten Steuerstrategie umgesetzt und damit gleichzeitig evaluiert werden, ob solch ein Vorgehen generell angewandt werden kann.

4.2. Ausfallerkennung

Das in der Komponente Fault Detection verwendete Modell des Flugzeugs soll durch ein *time-recurrent neural network* (RNN) abgebildet werden. Dieses setzt einen Zustandsbeobachter ohne die Verwendung von analytisch aufwendig zu modellierenden Kalman-Filtern um. Statt eines a priori aufgestellten analytischen Modells erlernt das Netz die Übertragungsfunktion, bzw. die Bewegungsgleichungen des Flugzeugs.

Das Flugverhalten, wie auch andere dynamische Systeme, wird durch ein mathematisches Modell in Form von diskretisierten Differentialgleichungen abgebildet (siehe Abschnitt 2.2). Zur Lösung der Differentialgleichung werden in einem Flugsimulator numerische Methoden verwendet. Einer numerischen Lösung liegen zeitliche Abfolgen und Zeitschritt-gebundene Rechenoperationen zugrunde. Daher muss die Lösung, welche das Netz abbildet, ebenfalls an eine zeitliche, bzw. sequentielle, Abarbeitung der Daten gebunden sein. Die Umsetzung zur Ausfallerkennung soll rekurrente neuronale Netze verwenden. Dieser Netztyp eignet sich aufgrund der Fähigkeit, sequentielle Abhängigkeiten zu erkennen, für die Systemidentifikation und Nachbildung von dynamischen Systemen (vgl. [Patan (2008)], [Horvath (2002)], [Trischler und D'Eleuterio (2015)], [Behera und Rana (2014)] und [Funahashi und Nakamura (1993)]). Der umzusetzende Prototyp soll dabei sowohl mit auf Sliding-Window basierenden Feed-Forward Netzen, als auch mit intrinsischen rekurrenten Netzen evaluiert werden. Die nachfolgende Darstellung basiert auf inhärent rekurrenten Netzen. Die zu entwickelnde Arbeitsumgebung zur Übertragung soll es ermöglichen, die Komponenten des zu überwachenden Systemzustands

frei zu wählen. Es wird zunächst angenommen, dass dieser ausschließlich aus den Drehraten besteht.

Die Abbildung 4.2 zeigt schematisch den Aufbau der Komponente Fault Detection. Die Basis hierfür bildet der Ansatz von Uhlig, Selig und Neogi [Uhlig u. a. (2010)].

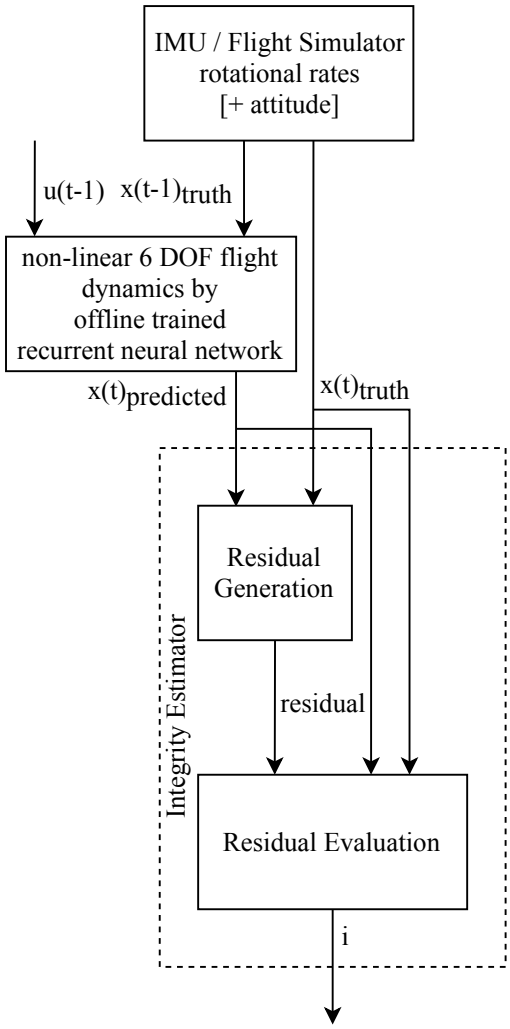

Abbildung 4.2.: Ansatz zur Abschätzung der verbliebenen Integritätszustände

Der Integritätszustand i soll in jedem Schritt der Regelschleife berechnet werden. Hierfür schätzt das Netz zunächst den aktuellen Zustand $x(t)_{predicted}$. Als Eingabedaten werden der Systemzustand $x(t-1)$ und die Aktuatoransteuerungen $u(t-1)$ des letzten Regelschrittes verwendet. Der tatsächliche Zustand $x(t)_{truth}$ wird vom Drehratensensor (IMU) geliefert. Der Restwert $residual$ wird über eine Fehlerquadratsumme (engl. *mean-squared-error*, *MSE*) aus

$x(t)_{predicted}$ und $x(t)_{truth}$ gebildet. Die Integritätsabschätzung i wird durch einen Zustandsautomaten berechnet.

4.2.1. Erzeugung des Restwerts

Für eine genaue Integritätsabschätzung muss ein Restwert geeignet erzeugt und evaluiert werden. Zur Erzeugung haben Uhlig et al. die Differenz aus gemessenem und erwartetem Zustand herangezogen. Sie haben dabei angemerkt, dass es in bestimmten Situationen zu Impulsspitzen (engl. *spikes*) kommen kann. Diese treten besonders zu Beginn von raschen Zustandsänderungen auf, zum Beispiel wenn ein Rollmoment aus einer Ruhephase heraus erzeugt wird. Ihrer Ansicht nach lag der Grund darin, dass die Systemverzögerung (engl. *lag*) vom Netz unzureichend modelliert wurde (vgl. [Uhlig u. a. (2010)]). Die Spikes könnten von der nachgelagerten Restwertevaluierung je nach Umsetzung zu False-Positives führen, da sie fälschlicherweise als Ausfall der Steuerbarkeit aufgefasst werden. Daher sollte eine vom Netz nicht modellierte Systemverzögerung in der Generierung des Restwerts beachtet und kompensiert werden. Uhlig et al. haben deshalb neben dem *direct residual*, welches lediglich die aktuellen Werte zur Generierung des Restwerts heranzieht, ein *time-lagged residual* gebildet. Für dieses wird die aktuell geschätzte Drehrate $x_{predicted}$ mit den n-letzten gemessenen Drehraten x_{truth} verglichen. Es wird der Restwert ausgewählt, welcher die geringste Differenz ergibt. Die Gleichungen 4.2.1 und 4.2.2 stellen die Berechnungen des direkten und verzögerten Restwerts dar. Im eigenen Ansatz soll anstelle der einfachen Differenz, das Fehlerquadrat verwendet werden. Hiervon wird sich eine bessere Generierung des Restwerts versprochen.

$$R_{direct}(t) = x_{predicted}(t) - x_{truth}(t) \tag{4.2.1}$$

$$R_{lagged}(t) = \min_{i=0:n}^{n} \{x_{predicted}(t) - x_{truth}(t-i)\} \tag{4.2.2}$$

Neben den von Uhlig et al. vorgeschlagenen Varianten soll eine Berechnung des Restwerts anhand der Fehlerquadratsumme (MSE) über die n-letzten Datenwerte möglich sein. Diese weitere Wahlmöglichkeit soll die Anpassung der Ausfallerkennung flexibler gestalten. Die Formel zur Berechnung ist in Gleichung 4.2.3 dargestellt.

$$R_{windowed}(t) = \frac{1}{n} \sum_{i=0:n}^{n} (x_{predicted}(t-i) - x_{truth}(t-i))^2 \tag{4.2.3}$$

Geeignete Längen für MSE- und Zeitverzögerungs-Fenster hängen von verschiedenen Faktoren ab. Diese können die Güte der Abbildung des Modells des Flugverhaltens, die Dynamik

des Flugzeugs, die gewünschte Zeitverzögerung bis zum Erkennen eines Ausfalls, usw. sein. Die Länge n des Fensters soll daher variabel eingestellt werden können.

4.2.2. Evaluierung des Restwerts

Aus dem Funktionsverlauf des Restwerts lassen sich durch die Auswertung von bestimmten Mustern Ausfälle und deren Typen ableiten. In Abbildung 3.4 zur Erkennung von Oszillationsmustern der Steuerflächen ist in der Komponente Residual Evaluation eine parallele Architektur dargestellt. In dieser wird der Restwert in unabhängige Analysatoren gegeben, von denen jeder auf die Erkennung eines bestimmten Musters programmiert ist. Dieser modulare Ansatz ist einfach erweiterbar, indem zusätzliche Analysatoren parallel dazugeschaltet werden. Das Bestimmen des Ausfalltyps ist dann nützlich, wenn der Flugregler und die Steuerflächenzuordnung gekoppelt sind und gemeinsam in ein anderes Flight Control Law umschalten. In der vorliegenden Problemstellung sollen die Komponenten jedoch vollständig getrennt sein. Für den Control Allocator ist lediglich der Integritätszustand als Indikator für die verbliebene Effektivität der Steuerflächen von Bedeutung. Daher wird hier vorerst nur ein Analysator realisiert, welcher die Effektivitäten in Form des Integritätszustands i schätzt.

Der Analysator soll ebenfalls so ausgelegt sein, dass nach Möglichkeit keine False-Positives erzeugt werden. Es erfolgt daher eine Analyse des Restwerts, bei der kurze Spikes erlaubt sind. Weiterhin gelten einige Annahmen. Zum einen, dass bei Start des Flugzeugs, bzw. der Avionik, keine Ausfälle vorliegen und das Flugzeug mit voller Performance fliegen kann. Zum anderen, dass ein Ausfall permanent ist und dieser während das System eingeschaltet ist, nicht behoben werden kann. Der Wert der Integritätsabschätzung i ist somit monoton fallend von 1 nach 0.

Der Analysator wird als Zustandsautomat realisiert. Dieser ist an die Umsetzung von Uhlig et al. angelehnt. In einem Zustandsautomaten können die zu erkennenden Muster in den Zuständen und Transitionen abgebildet werden. Aus den Funktionsverläufen des Restwerts kann abgeleitet werden, dass erwarteter und gemessener Systemzustand divergieren.

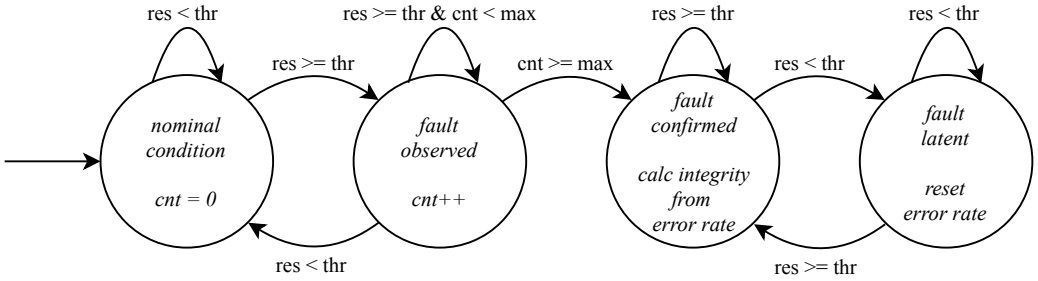

Abbildung 4.3.: Zustandsautomat zur Integritätsabschätzung durch Evaluierung des Restwerts

Abbildung 4.3 zeigt das Modell des zu realisierenden Zustandsautomaten. Der Zustand *nominal condition* stellt den Startzustand dar. Solange der Restwert res unter einem bestimmten Grenzwert thr bleibt, wird in diesem Zustand verweilt. Bei Überschreitung des Grenzwertes wird in den Zustand *fault observed* gewechselt. Dieser stellt einen temporären Zwischenschritt zwischen *"kein Ausfall erkannt"* und *"die Steuerfläche arbeitet eingeschränkt"* dar. Er wird eingeführt, um Spikes herauszufiltern. Hierfür wird ein Zähler cnt verwendet, welcher im Zustand *fault observed* inkrementiert und im Zustand *nominal condition* zurückgesetzt wird. Wenn der Restwert im Zustand *fault observed* unter den Grenzwert fällt, wird in den Zustand *nominal condition* zurückgesprungen und der Zähler zurückgesetzt. So wird das Überschreiten des Grenzwertes lediglich als kurzer Spike gewertet. Wenn der Grenzwert jedoch weiterhin überschritten wird und der Zähler ein Maximum max erreicht hat, wird in den Zustand *fault confirmed* gewechselt. Über das Maximum kann die maximale erlaubte zeitliche Länge eines Spikes variiert werden. Im Zustand *fault confirmed* kann sicher davon ausgegangen werden, dass es sich nicht um einen kurzen Spike, sondern um einen echten Effektivitätsverlust der Steuerfläche handelt.

Ab diesem Zeitpunkt kann nicht mehr zu den beiden vorherigen Zuständen zurückgesprungen werden. Das System befindet sich im Zustand *fault confirmed*, wenn der Restwert den Grenzwert überschreitet. Wird der Grenzwert unterschritten, wechselt das System in den Zustand *fault latent*. Dieser bildet eine Situation ab, bei der der Ausfall zwar vorhanden ist, zu dem gegebenen Zeitpunkt jedoch verborgen bleibt. Der Fall tritt zumeist auf, wenn keine oder wenig Ansteuerung der Steuerfläche vorliegt, wie etwa bei einem Horizontalflug. In dieser Situation sind die Ausschläge der Steuerflächen zu gering, so dass sich der Fehler nicht zeigt. Daraus ergibt sich ein niedriger Restwert, da das neuronale Netz ohne eine Ansteuerung auch eine geringe Zustandsänderung des Systems schätzt.

Dies führt dazu, dass sich das System bei einer Ansteuerung typischerweise im Zustand *fault confirmed* und ohne eine vom Autopiloten gewünschte Erzeugung eines Drehmoments im Zustand *fault latent* befindet. Die Schätzung der verbliebenen Effektivität kann somit lediglich bei einem angesteuerten Ausschlag erfolgen. Nur in solchen Situationen lässt sich der Integritätszustand ableiten. Hierfür wird eine $error\ rate$ eingeführt. Diese ergibt sich aus dem Quotienten von gemessenem und erwartetem Systemzustand. Zum Beispiel, wenn die gemessene Drehrate lediglich 80% der erwarteten ist, kann von einer verbliebenen Effektivität von 80% der Querruder, bzw. der für die Erzeugung des Rollmoments verantwortlichen Steuerflächen, ausgegangen werden. Uhlig et al, erzielten mit dieser Methode gute Effektivitätsabschätzungen [Uhlig u. a. (2010)].

Zur Berechnung der $error\ rate$ werden bei jedem Eintritt in den Zustand *fault confirmed* die gemessenen und erwarteten Systemzustände in jeweils eigene Listen eingefügt. Die $error\ rate$ wird in jedem Zeitschritt aus allen aufgezeichneten Systemzuständen nach der in Gleichung 4.2.4 dargestellten Formel berechnet. Es wird dabei in den Listen nach dem kleinsten Quotienten gesucht, welcher ein einzelnes Paar bildet. Hiermit soll sichergestellt werden, dass bei der Abschätzung der verbliebenen Effektivität der worst-case Fall angenommen wird. Die berechnete $error\ rate$ wird als neue Integritätsabschätzung gesetzt, sofern diese niedriger als die bisherige ist.

$$er(t) = \min_{i=0:n}^{n} \{x_{truth}(t-i) \ / \ x_{predicted}(t-i)\} \tag{4.2.4}$$

Die Kombination aus neuronalen Netzen, welche den erwarteten Systemzustand schätzen, einer Restwertgenerierung sowie einer Evaluierung des Restwerts ergibt ein mächtiges Werkzeug zur Erkennung von Steuerflächenausfällen. Die zu realisierende Arbeitsumgebung soll das Entwickeln einer Fault Detection mit einem möglichst hohen Grad an Wahl- und Entscheidungsmöglichkeiten bieten. So etwa, welche rekurrenten Netztypen, Komponenten des Zustandsvektors, ob ein Netz pro Steuerfläche oder ein einzelnes Netz, welche Parameter für die Restwerterzeugung und -Evaluierung, usw. verwendet werden sollen. Mit der Arbeitsumgebung soll eine beispielhafte Umsetzung für einen simulierten Träger erstellt werden. In diesem Zusammenhang wird in Kapitel 8 eine Reihe von Experimenten durchgeführt und präsentiert.

4.3. Ausfallkompensation

Die Komponente Control Allocation hat die Aufgabe der Flugsteuerung. Sie muss die vom Flugregler gewünschten Zustandsänderungen der Drehmomente durch eine geeignete Ansteuerung der am Flugzeug zur Verfügung stehenden Steuerflächen umsetzen. Im Kontext des Regelkreises aus Abbildung 4.1 bedeutet dies, dass eine geeignete mathematische Abbildung der virtuellen Steuersignale v auf die real verfügbaren Aktuatoren u gefunden werden muss. Bei konventionellen Drachenflugzeugen mit je einem Satz Querrudern, Höhenrudern, Seitenrudern und Triebwerken ist die Abbildung simpel, da diese speziell für die Steuerung der Freiheitsgrade ausgelegt sind. Die Zuordnung zwischen Freiheitsgrad und Steuerfläche wird hier direkt von diesen vorgegeben. Für eine nicht adaptive Flugsteuerung kann die Zuordnung fest einprogrammiert werden.

Bei hochgradig überaktuierten Systemen, wie etwa modernen Kampflugzeugen, stehen weitaus mehr Steuerflächen als Freiheitsgrade zur Verfügung. Hierfür muss das Control Allocation Problem (CAP) gelöst werden, um eine optimale Ansteuerung zu erhalten [Durham u. a. (2017), Frost u. a. (2010)]. Soll eine Flugsteuerung adaptiv arbeiten, werden die Steuerflächen aus dem oberen Beispiel nicht länger als untrennbare Sätze von immer gleichlaufenden (engl. *ganging*) Aktuatoren, sondern als einzeln ansteuerbare Aktuatoren verwendet. Für eine auf Reallokation basierende adaptive Flugsteuerung ist das Control Allocation Problem somit ebenfalls zu lösen.

Die degradierte Integrität einer Steuerfläche soll durch eine dynamische Umverteilung des Steueraufwandes auf die verbliebenen intakten Steuerflächen kompensiert werden. Durch eine solche Neuverteilung der Aufgaben werden die Steuerflächen *reallokiert*. Die Aufgaben der Komponente erweitern sich damit von der einfachen Ansteuerung im fehlerfreien Zustand, zu einem komplexeren System, welches einen gewissen Grad an Fehlertoleranz sicherstellen muss.

4.3.1. Lernen statt a priori Systemwissen

Durch die eigenen Anforderung an die Lösung soll das System wenig vorgegebene Systemwissen vom zu steuernden Flugzeug besitzen. Es soll somit so wenig a priori Wissen wie möglich einprogrammiert werden. Das System muss demnach lernfähig sein, um eine geeignete Steuerung ohne a priori Wissen erst zu ermöglichen. Die Umsetzung der Komponente Control Allocation soll daher einen Algorithmus beinhalten, welcher die Systemzusammenhänge im laufenden Betrieb lernt. Im Kontext des Regelkreises bedeutet dies, dass die Systemzusammenhänge zwischen v und u angelernt werden müssen. Hierfür muss das System erkennen, welcher Aktuator (Komponente aus u) welchen Effekt auf welchen Freiheitsgrad (Komponente in x bzw. v) hat. Konkret wird dies durch die Steuermatrix B ausgedrückt. Die Dimensionen der Matrix sind durch die Dimensionen der Vektoren x und u vorgegeben. Der Lernalgorithmus hat die Aufgabe, die Koeffizienten der Steuermatrix durch ein *"anlernen"* zu approximieren.

Zum Lernen der Koeffizienten muss eine Systemidentifikation durchgeführt werden. Dies ist eine Standardtechnik zur Analyse des Systemverhaltens von technischen Systemen und Prozessen. Durch verschiedene definierte Testsignale wird die Abhängigkeit zwischen Eingangs- und Ausgangssignalen ermittelt. Aus diesen kann das Systemverhalten abgeleitet und ein mathematisches Modell gebildet werden. Abschnitt 6.1.1 führt das Thema Systemidentifikation und mögliche Testsignale weiter aus. Nachfolgend wird das Thema im Sinne der Anwendung im Entwicklungsprozess eines Regelsystems beleuchtet.

Die Systemidentifikation kann in einer definierten Testumgebung (*offline*) oder im laufenden Betrieb (*online*) durchgeführt werden. Bei einigen Systemen existiert als Testumgebung ausschließlich ein Modell des realen Systems. Wie zum Beispiel in einem Flugsimulator. Für ein Flugzeug macht das Konzept einer realen Testumgebung wenig Sinn, da es in diesem Fall ebenfalls real fliegen würde. Hierdurch befindet es sich jedoch nicht länger in einer simulierten Testumgebung, sondern in der realen Umgebung. Eine Offline-Identifikation heißt bei einem Flugzeug demnach, dass dieses durch ein simuliertes Modell, wie etwa in einem Flugsimulator, abgebildet wird. Eine Online-Identifikation, dass diese bei einem realen Flug durchgeführt wird.

Beide Varianten haben Vor- und Nachteile. Bei einer Offline-Identifikation kann eine umfangreiche Datenbasis verwendet werden, um das System zu analysieren. Bei der Online-Identifikation steigen die Kosten ungleich stärker, da jedes Testsignal real geflogen werden muss. Weiterhin ist die Online-Variante an die reale Zeit gebunden, wohingegen ein Flugsimulator eine Vielzahl an Manövern pro Sekunde simulieren kann. Die Anzahl der Testsignale pro Zeiteinheit ist damit direkt von der Rechenkapazität der Testumgebung abhängig. Dagegen steht jedoch, dass die Testumgebung stets eine nicht optimale Abbildung der Realität darstellt, und somit das System nicht perfekt simuliert. Die Ergebnisse der Systemidentifikation haben damit eine geringere Realitätsnähe, womit auch das mathematische Modell des Systems an Realitätsnähe verliert. Eine Offline-Identifikation hat weiterhin den Vorteil, dass auch kritisches Systemverhalten analysiert werden kann, ohne das System in Gefahr zu bringen. So etwa, dass extrem enge Kurven oder steile Nickwinkel simuliert werden, bei denen das reale Flugzeug womöglich einen Strömungsabriss (engl. *stall*) erleiden könnte.

Die Unterschiede bei den beiden Varianten ergeben sich demnach dazu, dass durch die Offline-Identifikation eine große und vielfältige Datenbasis aufgebaut werden kann, durch die Online-Identifikation eine kleinere, dafür jedoch genauere Datenbasis. Komplexe Ansätze, wie etwa das *Neural Network augmented Feedback-Linearization*-System von Calise und Rysdik [Calise und Rysdyk (1998)] (siehe Abschnitt 3.3) kombinieren beide Varianten. Es wird zuerst eine umfangreiche Offline-Identifikation durchgeführt, welche für eine grundlegende Festlegung der Reglerparameter ausreicht. Mit einer Online-Identifikation im laufenden Betrieb werden die Ungenauigkeiten des Modells durch eine Adaption der Parameter ausgeglichen.

Für die vorliegende Problemstellung ist die Anforderung gegeben, dass der Control Allocation so wenig a priori Wissen über das System wie möglich vorliegt. Deshalb soll der Ansatz eine reine Online-Identifikation beinhalten, durch welche die Steuermatrix B und deren Koeffizienten rein im laufenden Betrieb angelernt wird.

Inspiriert wird diese Herangehensweise neben dem beschriebenen Flugzeugunglück von dem Umstand, dass besonders in den letzten Jahren eine Vielzahl an Arbeiten vorgestellt wurden, welche verschiedene Problemstellungen rein durch maschinelles Lernen und künstliche Intelligenz lösen konnten, ohne dass a priori festgelegte Modelle oder Expertenwissen einprogrammiert war. Ein prominentes Beispiel hierfür stellt die für das Brettspiel *Go* entwickelte KI *AlphaGo* dar [Silver u. a. (2016)]. Das System konnte, bzw. kann, die besten menschlichen Go Spieler der Welt bezwingen, ohne dass ihm die Spielregeln und Mechaniken von Go einprogrammiert wurden. Das Wissen darüber, wie Go gespielt wird um zu gewinnen, hat es rein durch das Spielen gegen sich selbst erlernt [Burger (2016)]. Im August 2017 wurde auf der Weltmeisterschaft des Videospiels *Dota 2* ein computergesteuerter Spieler (engl. *bot*) präsentiert, welcher die weltbesten menschlichen Spieler mit nahezu 100%er Gewinnrate schlagen konnte [OpenAI (2017a)]. Dem Bot selbst wurden nur wenige Spielmechaniken fest einprogrammiert, die meisten wurden durch das Spielen gegen sich selbst erlernt [OpenAI (2017b)]. In [Peng u. a. (2017)] werden Gangarten und Fortbewegung von mehrbeinigen Robotern angelernt, ohne dass den Steuerungssystemen zuvor bekannt war, welche Aktuatoren welche Auswirkungen auf das System haben.

Eine Lösung, die komplett ohne a priori Wissen auskommt ist wünschenswert, beim technischen System Flugzeug jedoch nur schwer realisierbar. Die Systeme der vorgestellten Ansätze unterscheiden sich zum Flugzeug dahingehend, dass sich die Systeme zu jeder Zeit in einem *Safe-State* befinden. Das heißt, sie können zu keiner Zeit weder das zu steuernde System selbst, noch die Umwelt beschädigen.

Um die Steuermatrix B anzulernen, müssen Testsignale auf die Aktuatoren gegeben werden. Aus den Reaktionen des Flugzeugs lassen sich die Koeffizienten ableiten. Durch zufällige Testsignale, bei denen die Systemreaktion unbekannt ist, können schnell kritische Fluglagen oder Geschwindigkeiten entstehen. Diese müssen entsprechend korrigiert werden, um die Fluglage wieder zu stabilisieren. Der Regler würde virtuelle Steuersignale generieren, die das Flugzeug wieder *"fangen"* würden. Mit einer zu Beginn leeren Steuermatrix können diese jedoch nicht umgesetzt werden, da keine Kenntnis darüber besteht, welche Steuerfläche welchen Einfluss auf welchen Freiheitsgrad. Daraus ergibt sich demnach eine zirkuläre Abhängigkeit.

Um dieses Problem zu lösen, könnte das Flugzeug aus einer sehr großen Höhe starten, bei dem der Lernalgorithmus genügend Zeit hat die Koeffizienten zu lernen, bevor das Flugzeug abstürzt. Hieraus ergeben sich jedoch andere Probleme. Zum einen lässt sich zuvor schwer abschätzen, welche Höhe für einen ausreichend lange andauernden Lernprozess benötigt wird. Zum anderen würde sich aller Wahrscheinlichkeit nach bereits nach kurzer Zeit ein unkontrollierter Fall einstellen, da es zu einem Strömungsabriss kommt. In dieser Situation

erzeugt das Flugzeug keinen Auftrieb mehr und ist lediglich der Gravitation ausgesetzt. Wenn in diesem Moment Testimpulse auf die Steuerflächen gegeben werden, kommt es womöglich zwar zu Änderungen der Drehraten, jedoch gelten die Abhängigkeiten zwischen Ein- und Ausgängen des Systems für den unkontrollierten freien Fall. Diese stehen jedoch in keinem Zusammenhang mit den Abhängigkeiten einer normalen Flugsituation, bei der Auftrieb erzeugt wird und sich der Flug kontrollieren lässt. Die erlernten Zusammenhänge sind somit unbrauchbar. Den Lernprozess in großer Höhe starten zu lassen löst das Problem demnach nicht.

4.3.2. Konzept der Umsetzung

Aufgrund des fehlenden Safe-States kann nicht auf eine initiale Zuordnung von Freiheitsgrad auf Steuerfläche verzichtet werden. Daher muss die Komponente Control Allocation mit einer fest einprogrammierten Standard-Zuordnungstabelle (engl. *Allocation Table*) ausgestattet sein.

Weiterhin gilt die Anforderung, dass die Komponente bei Ausfall eine dynamische Reallokation vornimmt. Im ausfallfreien Zustand wird mit der Standardzuordnung der vorgegebenen Allocation Table gesteuert. Fällt der Integritätszustand einer Steuerfläche unter einen bestimmten Grenzwert, muss die Reallokation vorgenommen werden.

Die grundlegende Idee ist, den folgenden Ablauf zu umzusetzen, um ein sicheres Weiterfliegen trotz degradierter Performance zu ermöglichen:

1. Im ausfallfreien Zustand wird die Standard Allocation Table zur Steuerung verwendet
2. Der Integritätszustand einer Steuerfläche fällt unter einen Grenzwert
3. Das System führt die Lernphase aus, um die Koeffizienten der Steuermatrix B zu approximieren
4. Das System wählt geeignete Zuordnung von v auf u anhand der Steuermatrix und steuert das Flugzeug fortan mit dieser

Die Lernphase im fehlerfreien Zustand durchzuführen würde verschiedene Problemfälle nicht abdecken. So etwa, wenn die Steuerfläche zwar noch mit voller Geschwindigkeit bewegt wird, die Ausschläge jedoch geringer sind. Der umgekehrte Fall wäre ebenfalls nicht abgedeckt. Weiterhin können so auch strukturelle Ausfälle kompensiert werden, die im fehlerfreien Zustand noch nicht gegeben waren.

Der Integritätszustand, welcher von der Fault Detection geliefert wird, gibt keinen Aufschluss über den Fehlertyp. Es wird lediglich *abgeschätzt*, wie effektiv die Steuerfläche noch arbeiten kann. Die Lernphase soll Aufschluss darüber geben, wie effektiv die Steuerflächen *tatsächlich* noch arbeiten. Wie gut der Ansatz des Lernens nach Ausfall in dieser Form arbeitet, muss durch die Experimente evaluiert werden. Es ist notwendig zu evaluieren, ob das dynamische Anlernen

im Ausfallzustand zu viel Zeit benötigt, bevor eine dynamische Reallokation, bzw. alternative Zuordnung der Steuerflächen vor einem unmittelbaren Absturz angelernt werden kann. Da das Online-Lernen Zeit benötigt, ergeben sich prinzipiell ebenfalls die oben angesprochenen Probleme. Mit dem Vorhandensein einer Standardzuordnung, welche das Flugzeug womöglich noch stabilisieren kann, werden diese jedoch minimiert.

Durch das Anlernen des benötigten Verhaltens der Control Allocation ergibt sich eine Form eines Software-Agenten. Das benötigte Verhalten wird dynamisch angelernt und bei Bedarf angepasst, wodurch sich eine adaptive Flugsteuerung ergibt, welche fehlertolerant gegen Ausfälle der Steuerflächen arbeitet. Bis auf die Standard Allocation Table ist dem Agenten kein a priori Wissen über das zu steuernde Flugzeug einprogrammiert. Nachfolgend werden die zu realisierenden Lern- und Zuordnungsphase beschrieben, welche die Control Allocation beinhalten soll.

4.3.3. Lernphase

Die Lernphase hat zum Ziel, die Koeffizienten der Steuermatrix B anzulernen, aus welchen sich eine Zuordnungstabelle ableiten lässt. Da sich das Flugzeug durch den Ausfall einer Steuerfläche in einem womöglich kritischen Zustand befindet, ist es eine unabdingbare Anforderung, dass die Lernphase in möglichst kurzer Zeit möglichst viel Systemverhalten identifizieren kann. Dies limitiert die Systemidentifikation dahingehend, dass lediglich ein kleiner Satz an Testsignalen verwendet werden kann. Aus diesem Grund kann kein klassisches *Reinforcement Learning* erfolgen, da dieses typischerweise ein sehr langsames Lernverhalten zeigt (vgl. [Wagner (2018)], [Silver u. a. (2016)]).

Es muss daher eine an den Prozess vereinfachte Lernphase verwendet werden. Der konzeptionelle Ablauf soll wie folgt lauten:

1. Stabilisiere das Flugzeug mit der Standard Allocation Table für t_1 Sekunden
2. Gebe ein Sprung-Signal auf Steuerfläche u_i für t_2 Sekunden
3. Messe die Systemreaktion in allen vier Freiheitsgraden Rollen, Nicken, Gieren und Längsbeschleunigung
4. Trage die Systemreaktion in die Steuermatrix B ein
5. Wiederhole die Schleife mit der nächsten Steuerfläche u_{i+1}, bis alle Steuerflächen angeregt wurden

Die für eine ausreichend genaue Abschätzung mindestens benötigten Sekunden der Parameter t_1 und t_2 hängen stark von der Systemdynamik ab und müssen daher einstellbar sein. Weiterhin kann hierdurch eine Zeitschätzung erfolgen, wie lange die Lernphase dauert. In

einem real implementierten System können die Parameter etwa in Abhängigkeit der verbliebenen Flughöhe und Fluggeschwindigkeit erfolgen. Sind diese ausreichend gegeben, kann eine entsprechend längere Dauer gelernt werden, wodurch sich genauere Abschätzungen der verbliebenen Effektivitäten ergeben.

4.3.4. Zuordnungsphase

Anhand der identifizierten Koeffizienten der Matrix B muss eine Zuordnung erfolgen, durch welche das Flugzeug manövrierfähig bleibt.

Der vorgeschlagene Lernprozess verwendet lediglich ein Testsignal pro Aktuator. Dieses wird in Isolation zu den restlichen Aktuatoren angelegt, ohne dass eine Kombination stattfindet, durch welche sich Kopplungen erkennen lassen würden. Beide Faktoren lassen darauf schließen, dass die angelernten Koeffizienten nicht optimal approximiert werden. Es muss evaluiert werden, wie effektiv diese Methode die Effektivitäten abschätzen kann.

Diese Annahmen schränken den Prozess der Auswahl, welche Steuerfläche für welchen Freiheitsgrad verwendet werden soll, stark ein. Es wird angenommen, dass die Koeffizienten unzureichend approximiert werden und lediglich eine geringe Aussagekraft über die tatsächlich mögliche verbliebene Performance des Flugzeugs bieten. Eine optimale Steuerflächenzuordnung (siehe Abschnitt 3.2.5) lässt sich hierdurch schwer realisieren. Jedoch soll diese dennoch umgesetzt werden, um das Verfahren evaluieren zu können. Weiterhin soll eine Prioritätenbasierte Zuordnungstabelle mit variablem Daisy Chaining implementiert werden, welche statisch vorgegeben werden kann.

$$\begin{bmatrix} Rate_{roll} \\ Rate_{pitch} \\ Rate_{yaw} \\ Accel_x \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0.3 & 0.3 & 0.1 & 0.05 & 0.05 \\ 0.1 & 0.1 & 1 & 1 & 0 & 0.1 & 0.1 \\ 0.25 & 0.25 & 0.6 & 0.6 & 1 & 0.3 & 0.3 \\ -0.3 & -0.3 & 0.05 & 0.05 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} Aileron_l \\ Aileron_r \\ Elevator_l \\ Elevator_r \\ Rudder \\ Thrust_l \\ Thrust_r \end{bmatrix} \tag{4.3.1}$$

Die Allocation Table wird indirekt aus der Steuermatrix gebildet. Direkt nach der Lernphase enthält diese vorerst noch die gemessenen Systemreaktionen als Drehraten und der Beschleunigung. Zunächst werden die Werte normiert. Daraufhin werden mit einem *greedy-pick* Verfahren die Steuerflächen ausgewählt, welche die höchsten Auswirkungen auf den jeweiligen Freiheits-

grad haben. Die Gleichung 4.3.1 stellt beispielhaft eine angelernte Steuermatrix (idealisiert) für ein Flugzeug mit sieben Steuerflächen dar.

Die angelernten Werte geben den Faktor des erzeugten Systemverhaltens vermutlich unzureichend wieder. Daher kann keine Umrechnung erfolgen, wie viel Steuerleistung zu welcher Drehrate oder Beschleunigung führt. Somit ist nicht ersichtlich, wie viele der Steuerflächen benötigt werden und mit welchen Werten diese angesteuert werden müssen. Diese Problemstellung muss während der Realisierung evaluiert und gegebenenfalls eine Lösung implementiert werden.

4.4. Einschätzung

Durch die nicht benötigte analytische Modellierung des Flugverhaltens und der Systemdynamik ist der eigene Ansatz besonders attraktiv. Die Implementierung der Algorithmen erfolgen unabhängig vom Flugzeug. Hierdurch ist eine hohe Generalisierung gegeben, wodurch sich eine Übertragung auf einen bestimmten Träger relativ einfach durchführen lässt. Eine statische Steuerflächenzuordnung mit einem eventuell erweiterten Daisy Chaining muss starr in der Avionik implementiert werden und unterscheidet sich von Flugzeug zu Flugzeug. Durch das Ausnutzen der intrinsischen Kopplung der aerodynamischen Wirkung müssen keine redundant vorhandenen Steuerflächen am Flugzeug vorhanden sein. Stattdessen wird die erforderliche Steuerwirkung durch eine zur Laufzeit angelernte dynamische Reallokation erreicht.

Weiterhin hervorzuheben ist, dass der dynamisch lernende Teil der Control Allocation, angelehnt an die in Abschnitt 4.3 präsentierten Systeme, prinzipiell über wenig einprogrammiertes Wissen verfügt, welches System gesteuert wird. Der Ansatz kann somit auch auf andere zu steuernde Systeme übertragen werden.

Bei der Verwendung von Methoden aus der künstlichen Intelligenz kann auf eine starre und auf die Flugzeugkonfiguration angepasste Implementierung verzichtet werden. Das Flugverhalten und die benötigte Steuerflächenkonfiguration werden angelernt. Somit müssen die Algorithmen beim Wechsel des Trägers nicht umprogrammiert werden, lediglich die Parameter und Koeffizienten unterscheiden sich bei unterschiedlichen Flugzeugen. Diese können bei dem präsentierten Ansatz antrainiert werden.

Die nicht benötigte analytische Modellierung hat weiterhin den Vorteil, dass bei einer Übertragung auf einen neuen Träger deutlich weniger Domänenwissen aus den Gebieten der Signale und Systeme, Flugmechanik und Regelungstechnik benötigt wird, da keine händische umfangreiche Systemidentifikation durchgeführt werden muss. Durch den präsentierten Ansatz

wird die Systemidentifikation, welche das mathematische Modell des Flugzeugs als Ergebnis hat, durch die Algorithmen und das Anlernen selbst durchgeführt.

Jedoch bietet der auf Lernverfahren basierende Ansatz auch gravierende Nachteile gegenüber klassischen analytischen Methoden. So gehen durch die neuronalen Netze Teile der Nachvollziehbarkeit verloren, da die mathematischen Zusammenhänge nicht sofort und ohne weiteres erkennbar sind. Bei sicherheitskritischen Systemen ist dies ein deutlicher Schwachpunkt gegenüber der analytischen Modellbildung. Eine Zertifizierung für den allgemeinen Luftfahrtgebrauch ist mit der Verwendung von neuronalen Netzen nicht ohne weiteres möglich [Schumann u. a. (2003)].

Die neuronalen Netze werden zunächst offline angelernt. Dies ist technologiebedingt, da zum aktuellen Zeitpunkt kein Learning Framework für die Hardwareplattform der FCU zur Verfügung steht, welches ein Online-Lernen möglich machen würde. Ein Online-Lernen wäre überaus attraktiv, da eine Übertragung auf einen neuen Träger weniger Entwicklungsaufwand bedeuten würde. Weiterhin können durch ein Online-Lernen unvermeidliche Modellierungsfehler und -Ungenauigkeiten kompensiert werden. Die Ansätze von Calise et al. [Calise und Rysdyk (1998), Kim und Calise (1997)] sowie die darauf basierenden verwenden jeweils eine Online-Systemidentifikation, um sowohl Fehler bei der Modellierung, als auch durch plötzliche Beschädigung nicht modellierte Systemzustände und -Dynamiken im laufenden Betrieb zu kompensieren.

Die für die Ausfallkompensation verwendete Online-Systemidentifikation ist minimalistisch ausgelegt. Durch diese lässt sich die verbliebene Performance des Flugzeugs nur in geringem Maße beurteilen. Die Ausfallkompensation ist für ein reales System in dieser Form daher vermutlich lediglich für eine Schadens- und Gefahrenminimierung einsetzbar, wobei die Mission abgebrochen wird. Für eine leistungsstärkere fehlertolerante Flugsteuerung, mit welcher die Mission fortgesetzt werden kann, wird eine umfangreichere Systemidentifikation im fehlerfreien Zustand benötigt. Eine Systemidentifikation nach dem Ausfall kann erfolgen, um den Fehlergrad besser einschätzen zu können und die nötige Reallokation zu verbessern.

5. Simulationsumgebung und Systemarchitektur

Zur Evaluierung des Ansatzes soll die erste Instanziierung in einer geeigneten Simulationsumgebung erfolgen. Die einzelnen Komponenten des Regelkreises sind eigene Betriebssystemprozesse. Diese laufen in Realzeit[1] ab und regeln ein von einem Flugsimulator simuliertes Flugzeug. Ausfallerkennung, Ausfallkompensation, das Ändern von Parametern sowie das Triggern eines Ausfalls sollen live vom Benutzer eingesehen und durchgeführt werden können. Die nachfolgenden Abschnitte erläutern die entworfene Systemarchitektur, die Simulationsinfrastruktur *AESLink* sowie die Fehlerinjektion.

5.1. AESLink

AESLink ist ein umfangreiches Simulationsframework, welches im Rahmen des AES-Projekts entstanden ist [Hasberg (2014)]. Es bietet diverse Anwendungen zur Durchführung, Aufzeichnung, Analyse und Reproduktion von Simulations- und Testläufen zur Unterstützung der Entwicklung von Algorithmen für Flugregler. Der Kern von AESLink wird durch einen UDP-Multicast Softwarebus gebildet. Über diesen kommunizieren die Komponenten wie etwa Datenaufzeichnung, Autopilot und Flugsimulator in Realzeit miteinander.

Zum aktuellen Stand von AESLink sind als Flugsimulatoren AeroSimRC[2], X-Plane[3] und eine Anbindung an MATLAB/Simulink Modelle integriert. Ein instanziierter Flugregler kann als Autopilot aufgrund des Bussystems an beliebiger Stelle ausgeführt werden. Von einem Simulinkmodell (*MIL*-) über eine eigenständige Anwendung (*SIL*-) bis zu einem eingebetteten System (*HIL*-Simulation) kann der Flugregler schrittweise in immer realer werdenden Plattformen instanziiert werden. Zur Kommunikation mit eingebetteten Systemen stehen verschiedene Proxy Tools zur Verfügung. Diese stellen die Simulationsdaten für ein eingebettetes System beispielsweise über *UART* / *RS232* oder *CAN* zur Verfügung. So kann die Instanz des Flugreglers

[1] die Simulation läuft in *weicher* Echtzeit ab
[2] http://www.aerosimrc.com [Stand 10.05.2018]
[3] http://www.x-plane.com [Stand 10.05.2018]

etwa auf einem FPGA (vgl. [Schulz u. a. (2017a)]) oder einem Mikrocontroller (vgl. [Hasberg (2014), Schulz u. a. (2017b)]) ausgeführt werden.

Für die erste Evaluierung und Realisierung des Ansatzes wird zunächst AeroSimRC verwendet. Zwar bietet X-Plane eine realitätsnähere Flugphysik, jedoch ist AeroSimRC ungleich simpler in der Anpassung, Verwendung, Steuerung und Interoperabilität mit dem Softwarebus (vgl. [Hasberg (2014)]). Da es sich um eine erste Evaluierung handelt, überwiegen die Vorteile von AeroSimRC gegenüber denen von X-Plane. Daher wird für die Realisierung zunächst AeroSimRC als Flugsimulator gewählt.

5.2. Systemarchitektur

Die Abbildung 5.1 zeigt die entworfene Systemarchitektur für den laufenden Betrieb einer Simulation. Die einzelnen Komponenten sind jeweils eigene Anwendungen und formen einen Regelkreis. Die Anwendungen kommunizieren im Softwarebus über Nachrichten miteinander. Die Simulationsgeschwindigkeit in *Regelschritte-pro-Sekunde* wird von der Bildwiederholfrequenz (engl. *Frames-per-Second*, *FPS* oder *Framerate*) des Flugsimulators vorgegeben.

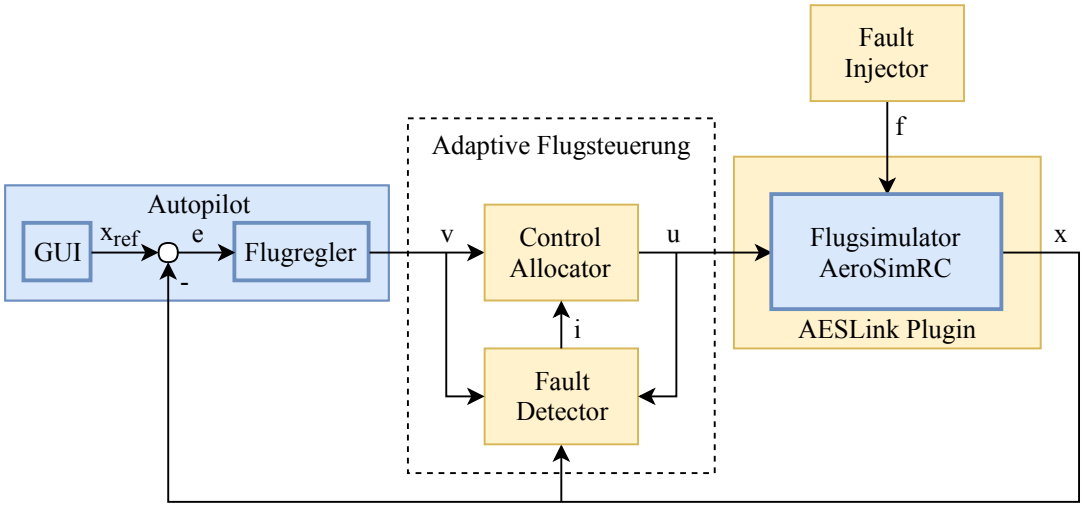

Abbildung 5.1.: Systemarchitektur zur Simulation und Evaluierung des Gesamtsystems adaptive Flugsteuerung

Die in Abbildung 5.1 in blau dargestellten Komponenten Flugsimulator und Autopilot sind bereits vorhanden. Der Autopilot und eine Basisversion des Flugsimulatorplugins werden aus AESLink übernommen (siehe [Hasberg (2014)]). Für die vorliegende Arbeit sind die in gelb dargestellten Komponenten zu realisieren, welche den Ansatz instanziieren und zugleich

evaluieren. Die Realisierung der Fault Detection wird in Kapitel 6, die der Control Allocation in Kapitel 7 erläutert. Die Erweiterungen von AESLink, des Flugsimulatorplugins sowie die Realisierung der Fault Injection ist in den nachfolgenden Abschnitten beschrieben.

5.3. Erweiterung von AESLink

Für die Realisierung der Systemarchitektur wurden neue Nachrichtentypen in AESLink definiert. Vorabversionen dieser wurden bereits in Vorarbeiten beschrieben (siehe [Hasberg (2017b)]).

Folgende drei Nachrichtentypen wurden zu AESLink hinzugefügt:

- ***Message_DataFromAllocator***: Definiert das Signal u. Die Nachricht beinhaltet die Steuersignale für die Steuerflächen und wird vom Control Allocator an den Flugsimulator gesendet.
- ***Message_DataFromDetector***: Definiert das Signal i. Die Nachricht beinhaltet die Integritätsabschätzungen der Freiheitsgrade (i_{vcs}) sowie der Steuerflächen (i_{cs}) und wird vom Fault Detector an den Control Allocator gesendet.
- ***Message_ControlSurfaceFaults***: Definiert das Signal f. Die Nachricht beinhaltet die Ausfalltypen und -Parameter der Steuerflächen und wird vom Fault Injector an den Flugsimulator gesendet. Anders als die anderen Nachrichten, welche im Regelkreis während einer Simulation pro Regelschritt verteilt werden, wird eine Nachricht f lediglich sporadisch verteilt.

5.4. Fault Injection

Um die realisierte adaptive Flugsteuerung als Gesamtsystem zu evaluieren, muss der Ausfall einer Steuerfläche durch die Simulationsumgebung unterstützt werden. Die Realisierung einer solchen *Fault Injection* teilt sich in zwei Arbeitspakete. Zum einen muss der Flugsimulator, bzw. dessen mit dem Softwarebus kommunizierendes Plugin, erweitert und ein passendes zu simulierendes Flugzeug ausgewählt werden. Zum anderen braucht es ein Programm, mit welchem die Ausfälle getriggert werden können, wodurch sich eine Ausfallsituation einstellt. Beide Arbeitspakete wurden bereits in Vorarbeiten realisiert (siehe [Hasberg (2017b)]). Nachfolgend werden die wichtigsten Entwicklungen und Ergebnisse dieser Vorarbeiten zusammengefasst.

5.4.1. Konfiguration Flugsimulator und Flugmodell

Um den Ausfall einer Steuerfläche mit der intakten Aktorik durch eine dynamische Reallokation kompensieren zu können, muss die Flugzeugkonfiguration des simulierten Modells entsprechend dafür ausgelegt sein. Hierfür sind verschiedene Voraussetzungen erforderlich. Einerseits müssen genügend Steuerflächen vorhanden sind, die unabhängig angesteuert werden können. Andererseits müssen diese so am Flugzeug angeordnet sein, dass sie durch ihre Auslenkung gekoppelte Bewegungen in mehreren Freiheitsgraden induzieren. So kann etwa die Rollbewegung durch die Höhenruder umgesetzt werden, sofern diese symmetrisch zur Längsachse angeordnet sind und differentiell angesteuert werden können.

Als erster Träger, auf welchen die adaptive Flugsteuerung übertragen und wodurch diese evaluiert werden soll, wird ein Modellflugzeug vom Typ *Trainer40* verwendet. Dieses wurde angepasst und verfügt über folgende Steuerflächen:

- 2 Querruder, im äußeren Bereich der Tragflächen
- 2 Höhenruder, im hinteren Bereich des Höhenleitwerks
- 2 Seitenruder an 2 Seitenleitwerken. Die Seitenleitwerke sind jeweils oberhalb der beiden Seiten des Höhenleitwerks mit Abstand zum Rumpf angebracht
- 2 Landeklappen, innen liegend an den Tragflächen
- 2 Motoren, unterhalb der Tragflächen im äußeren Bereich

Mit dieser Konfiguration sind in der Theorie genügend unabhängige Steuerflächen vorhanden, um den Ausfall jeglicher Steuerfläche, bzw. den Ausfall von zwei Steuerflächen für den gleichen Freiheitsgrad, mit den intakten zu kompensieren.

Ein simuliertes Modell wird im Flugsimulator über die Tastatur, eine Fernbedienung oder einen Joystick gesteuert. Dabei werden die virtuellen Steuersignale v durch den Benutzer vorgegeben, da eine direkte Ansteuerung von u für einen Menschen typischerweise zu komplex ist (vgl. Abschnitt 3.2.1). Daher findet die Abbildung von v auf u im Flugsimulator statt.

Die im Flugsimulator inhärent eingebettete Steuerflächenzuordnung muss umgangen werden, so dass die Komponente Control Allocation die Steuerflächen direkt ansteuern kann. Dafür wurde das Plugin, welches den Flugsimulator an AESLink anbindet und die Interprozesskommunikation übernimmt, erweitert. Bei Empfang einer Nachricht v, welche die virtuellen Steuersignale enthält, wird wie zuvor die im Flugsimulator vorhandene Steuerflächenzuordnung verwendet. Bei Empfang einer Nachricht u wird die eingebettete Steuerflächenzuordnung umgangen und es werden stattdessen direkt die Steuerflächen angesteuert. So können der Flugsimulator und das Plugin weiterhin für eine Simulation ohne eine externe Steuerflächenzuordnung verwendet werden.

5.4.2. Fault Injector und Injection im Flugsimulator

Für das externe Triggern eines Ausfalls wurde die Anwendung *Fault Injector* entwickelt. Mit diesem können die (Ausfall-) Zustände der Steuerflächen einzeln eingestellt werden. Die Abbildung 5.2 zeigt einen Ausschnitt der Benutzeroberfläche. Dargestellt ist eine Groupbox, welche alle Konfigurationsmöglichkeiten einer Steuerfläche beinhaltet. Abgebildet ist die Konfiguration für das linke Querruder (*AIL_L*). Für die restlichen Steuerflächen sind analog jeweils identische Groupboxen inklusive der Schaltflächen vorhanden.

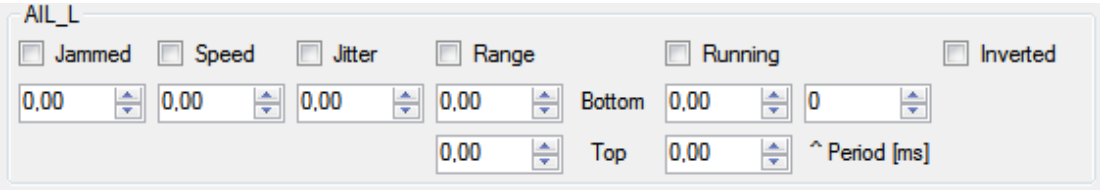

Abbildung 5.2.: Konfigurierbare Ausfalltypen je Steuerfläche

Ausfalltypen

Es können sechs unabhängige *Ausfalltypen* konfiguriert werden. Die Option *Jammed* blockiert die Steuerfläche und friert die Position im eingestellten Wert ein. Mit der Option *Speed* wird die Drehgeschwindigkeit der Steuerfläche variiert (*rate limit*). Ein *Jitter* addiert ein Rauschen auf das Steuersignal und lässt die Steuerfläche zittern. Mit der Option *Range* kann der Maximalausschlag eingeschränkt werden (*deflection limit*). Die Option *Running* lässt die Steuerfläche oszillieren. Durch die Option *Inverted* wird die Ansteuerung invertiert. Die Optionen Jammed, Range und Speed haben einen direkt ableitbaren Einfluss auf die Integritätsabschätzungen der Fault Detection.

Die einstellbaren Zahlenwerte stellen normierte Prozentwerte dar, wobei -1 und $+1$ jeweils die Maximalausschläge und Geschwindigkeiten darstellen. Ein Jammed von 0.5 bedeutet ein Einfrieren auf halber, positiver Auslenkung. Dies gilt analog für Range und Running. Der Wert des Jitters gibt das Maß an in jedem Regelschritt zufällig zu addierendem normierten Ausschlag an. Die Interpretation der Werte geschieht im Flugsimulator. Die Umrechnung von z. B. $+1$ in eine Auslenkung der Einheit Bogengrad ist von den Parametern des Flugmodells abhängig.

Aufgrund fehlender Informationen über die Dynamik der Steuerflächen des Flugmodells innerhalb des Plugins konnte der Wert der Drehgeschwindigkeit nicht linear abgebildet werden. Selbst minimale Abweichungen vom Standardwert 1 führen bereits zu großen Veränderungen der Drehgeschwindigkeit.

Die AESLink-Nachricht f enthält die Konfigurationsmöglichkeiten der einzelnen Steuerflächen. Das Flugsimulatorplugin wurde dahingehend erweitert, dass es diesen Nachrichtentyp in jedem Regelschritt empfangen und anwenden kann. Ein gesetzter (Ausfall-) Zustand bleibt solange gleich, bis eine neue Nachricht empfangen wurde.

Triggern des Ausfalls

Das Setzen eines neuen Konfigurationszustands der Steuerflächen ist nicht als kontinuierlicher Datenstrom im Simulationstakt entworfen. Stattdessen werden neue Konfigurationszustände *sporadisch* an den Flugsimulator gesendet. Hierfür stehen die in den Abbildungen 5.3 und 5.4 dargestellten Schaltflächen zur Verfügung.

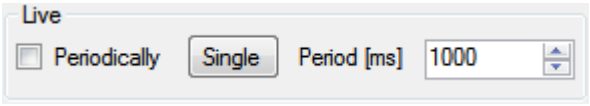

Abbildung 5.3.: Händisches Triggern zur Laufzeit

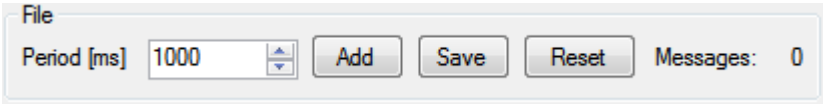

Abbildung 5.4.: Aufzeichnen der Ausfälle für autarkes Triggern bei der Erzeugung von Trainingsdaten

Das manuelle Setzen eines neuen Konfigurationszustands wird über den Button *Single* getriggert. Bei aktivierter Checkbox *Periodically* wird die Nachricht f periodisch verteilt. Durch ein periodisches Versenden können die neuen Konfigurationszustände live erzeugt und nachvollzogen werden, ohne dass nach jeder Änderung erneut auf den Button *Single* geklickt werden muss.

Für eine autark ablaufende Simulation kann über die Schaltflächen in der Groupbox *File* eine Datei erzeugt werden. Diese Datei wird von den in AESLink enthaltenen Tools wie eine aufgezeichnete Simulation behandelt und kann daher wiedergegeben werden. So können verschiedene Ausfallsituationen vorkonfiguriert und autark durch das Wiedergabe-Tool aus AESLink getriggert werden.

Die Simulationsinfrastruktur AESLink wurde um ein Flugmodell mit insgesamt zehn unabhängigen Steuerflächen, den Fault Injector, der Implementierung der Ausfallsituationen im Flugsimulator und die neuen Nachrichtentypen erweitert. Durch die Kombination dieser

Neuentwicklungen in AESLink wurde die Simulationsinfrastruktur um die Möglichkeiten der Verwendung und Entwicklung einer adaptiven Flugsteuerung erweitert. Die neu definierten Nachrichtentypen werden verwendet, um die Steuerflächen des simulierten Modells unabhängig ansteuern zu können, für diese eine Ausfallsituation zu simulieren und den Integritätszustand der Steuerflächen verteilen zu können. Diese ermöglichen es, die einzelnen Komponenten als eigenständige und unabhängige Anwendungen zu realisieren. Die Komponenten sind damit austauschbar, wiederverwendbar und es können Mockups eingesetzt werden.

6. Fault Detection

Das vorliegende Kapitel beschreibt die Realisierung der Komponente Fault Detection. Der zentrale Bestandteil sind die neuronalen Netze, welche das mathematische Modell des Flugverhaltens ohne eine analytische Modellbildung nachbilden. Zunächst wird auf die Erzeugung der Trainingsdaten eingegangen, welche für das offline-Lernen verwendet werden. Daraufhin wird die entwickelte Arbeitsumgebung zur einfachen Übertragung der Netze auf neue Trägersysteme erläutert. Zuletzt wird die Implementierung des Fault Detectors beschrieben, welcher die ausführbare Anwendung in der Systemarchitektur des simulierten Regelkreises aus Abbildung 5.1 darstellt.

6.1. Trainingsdatenerzeugung

Für eine qualitativ hochwertige Funktionsapproximation durch neuronale Netze sind neben der Wahl der Netz-Architektur und Hyperparameter auch die Trainingsdaten von entscheidender Rolle. Diese müssen entsprechend der Komplexität der zu approximierenden Funktion in ausreichender Menge vorhanden sein. Bei dynamischen Systemen, welche über mehrere Eingangsvariablen, interne Zustände und Nichtlinearitäten verfügen, ergeben sich eine Vielzahl von möglichen Funktionsverläufen. Für das gegebene Problem der Nachbildung eines hochgradig nichtlinearen Flugverhaltens werden für eine gute Approximation des Systemverhaltens Daten über das Flugverhalten in möglichst vielen Flugzuständen benötigt. Aufgrund der unzähligen unterschiedlichen Flugmanöver und -zustände, die angelernt werden müssen, wird die benötigte Datenmenge als sehr groß eingeschätzt.

Es wird daher ein Datengenerator benötigt, welcher autark über mehrere Stunden oder Tage hinweg den Flugsimulator anregt und Flugmanöver erzeugt. Die vom Datengenerator erzeugten Signale sind die Features des Netzes, die Ausgangssignale des Flugsimulators bilden die Labels. Ein Netz kann daraufhin nach dem Verfahren *Überwachtes Lernen* (engl. *supervised learning*) trainiert werden, wobei ein aufgezeichneter simulierter Flug einen Trainingsdatensatz darstellt.

6.1.1. Systemidentifikation und Datenreduktion

Für eine präzise arbeitende Ausfallerkennung mit hoher Erkennungsrate und wenigen False-Positives muss das Netz das Flugverhalten funktional möglichst exakt nachbilden. Hierfür ist es unerlässlich, dass das Netz bei den gleichen Eingangssignalen mit dem Flugsimulator übereinstimmende Flugzustände erzeugt. Es ist daher zwingend erforderlich, dass das Netz und der Flugsimulator bei identischen Inputs auch identische Outputs liefern. Aus den sechs Freiheitsgraden des dreidimensionalen Raums, den Nichtlinearitäten sowie den unzähligen möglichen Flugmanövern und Flugzuständen resultiert ein praktisch unendlich großer Zustandsraum an möglichem Systemverhalten. Für ein effizientes Training muss der Zustandsraum daher zunächst eingegrenzt werden.

Der Flugregler hat eine *Flight Envelope Protection* implementiert. Diese verhindert zu jeder Zeit kritische Fluglagewinkel des Trägers. Durch diese wird der Flugbereich (engl. *Flight Envelope*) des Trägers eingeschränkt. Der Datengenerator muss lediglich solche Flugmanöver und Flugzustände erzeugen, welche durch die Flight Envelope Protection des Flugreglers zugelassen werden. So wird das Flugmodell bei der Trainingsdatenerzeugung in den gleichen Flugbereichsgrenzen simuliert, wie sie im produktiven Einsatz vorkommen.

Um die Datenmenge weiter zu reduzieren, muss erreicht werden, dass der Flugsimulator durch wenige Inputs des Datengenerators möglichst viele unterschiedliche Flugmanöver erzeugt. Hierfür werden Methoden aus der Systemidentifikation verwendet.

Zur Modellierung eines Systems gehören dessen Struktur und Koeffizienten. Die Systemidentifikation wird verwendet, um Abhängigkeiten zwischen In- und Outputs der mathematischen Abbildung des Systems zu erkennen und quantitative Aussagen über diese zu treffen. Hierdurch können die Struktur modelliert und die Koeffizienten bestimmt werden. Zu typischen Eingangssignalen oder *Testsignalen* zählen unter anderem ein Sprung, eine Rampe oder eine Sinus-Funktion. Diese Testsignale eignen sich für eine Systemidentifikation, da durch sie möglichst viele Rückschlüsse auf das dynamische Systemverhalten geschlossen werden können. Daher sollen diese vom Datengenerator als Steuer-Inputs für den Flugsimulator generiert werden.

6.1.2. Zeitverhalten und Reproduzierbarkeit

Das vom Flugsimulator berechnete Flugverhalten unterliegt einer Sequenz aus physikalisch plausiblen Zustandswechseln. Die Berechnung eines rekurrenten neuronalen Netzes muss aufgrund der temporalen Bedingung des dynamischen Systems an diese sequentielle Abfolge gebunden sein (siehe Abschnitt 2.3.2). Bewegungsabläufe, wie etwa Änderungen der Fluglagen,

müssen im Netz dahingehend abgebildet sein, dass sie das dynamische Verhalten des Flugzeugs widerspiegeln. Hieraus ergibt sich die Anforderung, dass jeder Flugzustand, in welchem ein Ausfall erkannt werden soll, durch ein in sequentiellen Abfolgen plausibel und korrekt berechnetes Flugverhalten antrainiert werden muss. Daher sind Sprünge des Flugzustands nicht erlaubt. Hieraus ergibt sich, dass ein durchgeführter Flug zunächst einem kompletten Trainingsdatensatz entspricht.

Der Flugsimulator AeroSimRC ist in seiner Simulation an die real ablaufende Zeit gebunden (Realzeit). Somit kann ein Simulationsdurchlauf nicht beliebig schneller oder langsamer als die Realzeit ablaufen. Hieraus leitet sich eine zwingende Anforderung für das Zeitverhalten des Datengenerators ab. Durch die Vorgabe der Simulationsgeschwindigkeit des Flugsimulators muss die Datenerzeugung an die Zeitbasis des Simulators gebunden sein. Der Simulationstakt wird vom Flugsimulator vorgegeben, der Datengenerator muss im gleichen Takt Eingangssignale liefern und demnach auch an diesen gekoppelt sein.

Die Gewichtungen und Biase von neuronalen Netzen werden vor dem ersten Anlernen typischerweise mit Zufallswerten initialisiert [Bengio (2012b)]. Dies macht es schwieriger, verschiedene Trainingsdaten, Parameter oder Architekturen miteinander zu vergleichen. Um dennoch möglichst reproduzierbare Ergebnisse zu erhalten, ist eine eigene Anforderung an den Datengenerator, dass dieser möglichst reproduzierbare Signale erzeugt.

Diese Anforderungen geben vor, dass die Signale in den Datenwerten und in der zeitlichen Abfolge reproduzierbar erzeugt werden. Dies soll sicherstellen, eine konsistente Arbeitsumgebung für die Entwicklung der neuronalen Netze zu schaffen. So können verschiedene Simulationsdurchläufe mit unterschiedlichen Netzen oder Parametern in ihrer Performance miteinander verglichen werden.

6.1.3. Prozess der Datenerzeugung

Als Datengenerator wurde die Anwendung *Pilot Simulator* entwickelt. Die Realisierung wird im nachfolgenden Abschnitt beschrieben. Zunächst soll jedoch der Prozess der Datenerzeugung erläutert werden.

Abbildung 6.1 zeigt die Systemarchitektur der Trainingsdatenerzeugung. Der Pilot Simulator ersetzt den Autopiloten und tritt an dessen Stelle. Über den Zeitraum der Datenerzeugung generiert dieser den Input der virtuellen Steuersignale v für den Flugsimulator. Der Control Allocator wird mit der Standardzuordnung der Steuerflächen verwendet (siehe Abschnitt 4.3.2). Durch die Anordnung der Komponenten als Regelkreis ist der Pilot Simulator an die Zeitbasis des Flugsimulators gekoppelt.

Die Signale v, u und x werden vom *Logger* aufgezeichnet und in ein Logfile geschrieben. Das Logfile enthält somit die virtuellen Steuersignale des Piloten, die tatsächlichen Ansteuerungen der Steuerflächen und den Systemzustand des simulierten Flugzeugs. Damit bildet das aufgezeichnete Logfile einen Trainingsdatensatz.

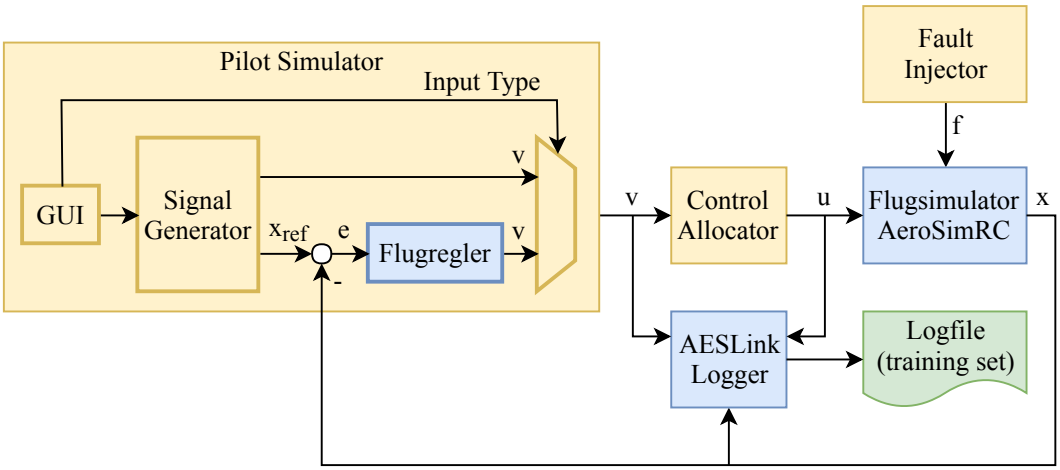

Abbildung 6.1.: Simulationsarchitektur für die Erzeugung von Trainingsdaten

6.1.4. Pilot Simulator

Die primäre funktionale Anforderung an den Datengenerator ist die autarke Erzeugung von Eingangssignalen, welche im Flugsimulator eine Vielzahl an verschiedenen komplexen Flugmanövern produzieren. Die nicht-funktionalen Anforderungen sind die Datenreduktion und die Reproduzierbarkeit. Aus diesen gegebenen Anforderungen wurde die Anwendung *Pilot Simulator* entwickelt. Für den Prozess der Trainingsdatenerzeugung tritt dieser an die Stelle des Autopiloten und emuliert diesen, wobei er die Eingangssignale in der Form erzeugt, wie der Autopilot sie erzeugen würde. Durch die Variationen der Testsignale kann bei einem zeitlich lange genug andauernden Simulationsdurchlauf zur Datenerzeugung der komplette Flight Envelope an Flugmanövern- und Zuständen, die der Flugregler im normalen Betrieb erzeugt, generiert werden.

Die grundlegende Basisversion und der Signalgenerator des Pilot Simulators wurden bereits in Vorarbeiten realisiert (siehe [Hasberg (2017b)]). Der Signalgenerator bildet den algorithmischen Kern der Anwendung. Für dessen Umsetzung und Evaluierung sei hier auf [Hasberg (2017b)] verwiesen. Die Evaluierung hatte gezeigt, dass die realisierte Version nicht für einen komplett autarken Betrieb geeignet ist. Daher wurde der Pilot Simulator dahingehend erwei-

tert, dass ein ungewollter Absturz des Flugzeugs vermieden wird. Weiterhin konnten in der bisherigen Version ausschließlich Signale für die Rollachse erzeugt werden. Der Pilot Simulator wurde daher um die Funktionalität der Steuerung aller Achsen erweitert.

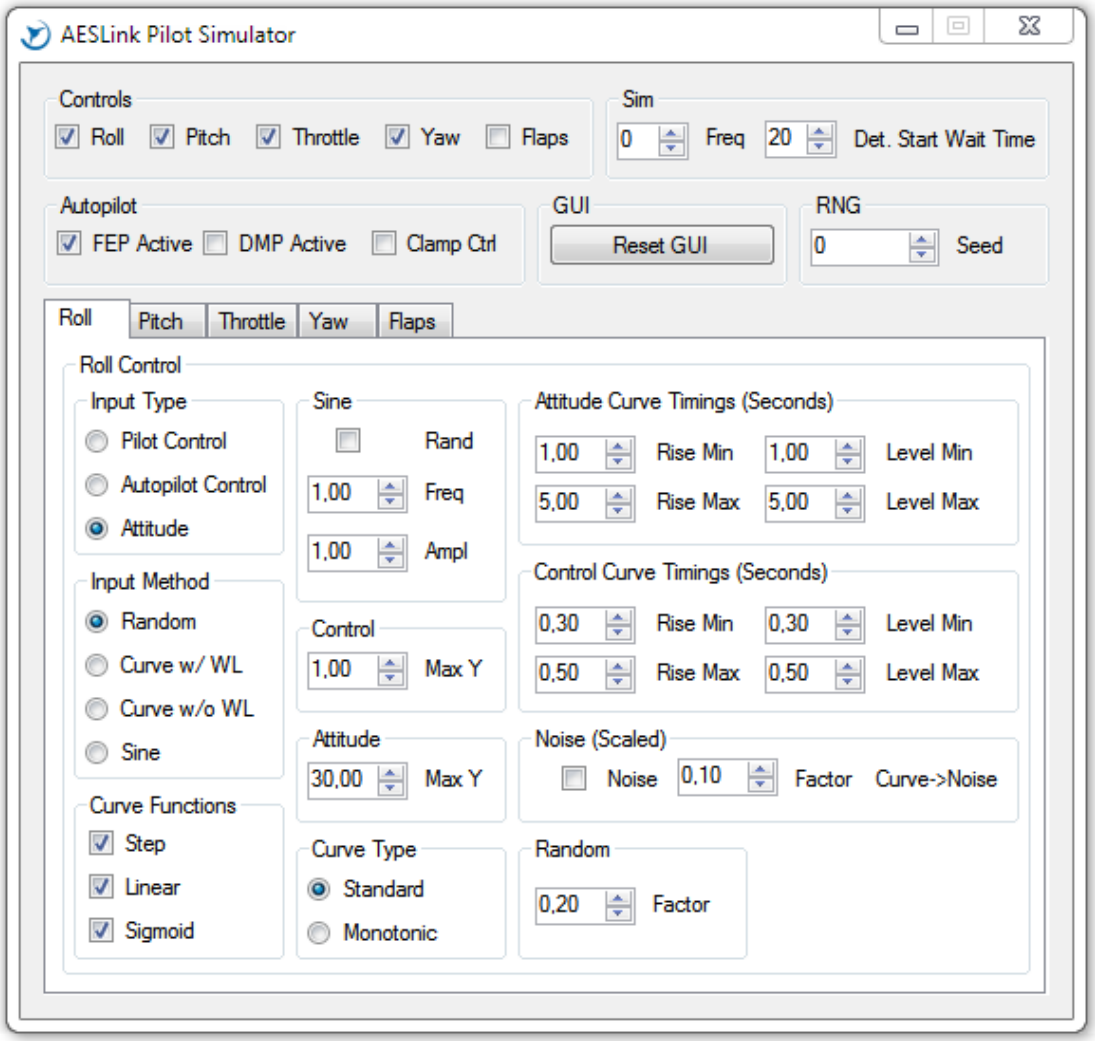

Abbildung 6.2.: Benutzeroberfläche des erweiterten Pilot Simulators

Abbildung 6.2 zeigt die Benutzeroberfläche der Weiterentwicklung. Über die Schaltflächen lässt sich flexibel eine Vielzahl an unterschiedlichen Eingangssignalen für den Flugsimulator generieren. Der Pilot Simulator hat selbst Teile des Flugreglers, wie etwa eine Regelung der Fluglagen, der Geschwindigkeiten oder des Kurses sowie eine Flight Envelope Protection implementiert. Die Verläufe der Sollwerte der Regler können mit Sprung-, Linear-, S-, Sinus- oder zufällig monoton steigenden Funktionen belegt werden. Die Funktionen können ebenfalls

den Verlauf der virtuellen Steuersignale vorgeben. Für die Vorgabe der Sollwerte kann ebenfalls festgelegt werden, ob nach einer Bewegung kurze Zeit später wieder auf den Ausgangszustand des eingeschwungenen Horizontalflugs geregelt werden soll. Allen Signalen kann ein additives Rauschen hinzugefügt werden. Die zusätzliche Möglichkeit der Erzeugung von komplett zufälligen Sollwerten oder Steuersignalen erweitert das Spektrum an möglichen Flugmanövern und -zuständen über den Flight Envelope des Flugreglers hinaus. Der Signalgenerator kann die beschriebenen Kombinationen an Signalen für jede Achse separat erzeugen.

Durch die flexible Konfiguration von Sollwertvorgaben, direkte Signalerzeugung, Kurvenfunktionen, additives Rauschen, usw. kann der Pilot Simulator ein großes Spektrum an verschiedenen Eingangssignalen generieren. Durch diese Vielfalt der Kombinationsmöglichkeiten des Signalgenerators kann ein umfangreiches Spektrum an unterschiedlichen Flugmanövern, die der Autopilot durchführen kann, in den Datensätzen abgebildet werden. Durch die implementierten Regler in allen Achsen und der Flight Envelope Protection kann der Flugbereich in Lage, Höhe und Kurs eingeschränkt werden, wodurch eine unbeaufsichtigte und autarke Simulation möglich ist. Hierdurch kann mit geringem Aufwand eine umfangreiche Datenbasis für das Anlernen der neuronalen Netze erzeugt werden.

Das im Simulator verwendete Flugmodell kann ohne Aufwand gewechselt werden, da das funktionale Modell des Signalgenerators im Pilot Simulator hierfür nicht angepasst werden muss. Über eine Konfigurationsdatei können eventuell nötige Anpassungen der Reglerparameter und -grenzen vorgenommen werden. Daraus ergibt sich eine hohe Flexibilität und die Möglichkeit der einfachen Wiederverwendung mit anderen Flugmodellen, bzw. in anderen Projekten.

Mit dem Pilot Simulator wurde ein flexibles Werkzeug zur Generierung von Eingangsdaten für die Erzeugung einer Datenbasis realisiert. Er ist darauf ausgelegt, um autark über lange Zeiträume hinweg verschiedene Flugmanöver durchzuführen und damit ein umfangreiches Spektrum an Test- und Trainingsdaten für das Training der neuronalen Netze zu erzeugen.

6.2. Arbeitsumgebung zur Entwicklung der neuronalen Netze

Eine primäre Anforderung an das Ergebnis dieser Masterthesis ist es, dass die adaptive Flugsteuerung ohne großen Aufwand auf verschiedene Träger übertragen werden kann. Der präsentierte Ansatz unterstützt diese Anforderung dahingehend, dass keine eigene analytische Modellbildung für das Flugverhalten durchgeführt werden muss. Stattdessen werden trainierbare neuronale Netze verwendet, welche das Systemverhalten des Flugzeugs nachbilden.

Zur einfachen Entwicklung und Übertragung der Netze ist eine Arbeitsumgebung nötig. Mit dieser soll ein Nachbilden des Flugverhaltens durchgeführt werden können, ohne dass tiefgreifende Kenntnisse aus der Flugmechanik oder Regelungstheorie nötig sind. Der vorliegende Abschnitt beschreibt zuerst den Entwicklungsprozess und darauffolgend die finale Realisierung dieser Arbeitsumgebung.

Zunächst wurden verschiedene verbreitete und anerkannte Frameworks zur Einbettung und Entwicklung von neuronalen Netzen evaluiert. In Betracht gezogen wurden unter anderem *TensorFlow* [Abadi u. a. (2015)], *scikit-learn* [Pedregosa (2011)], *CNTK* [Yu u. a. (2014)] und Matlabs *Neural Network Toolbox* [The MathWorks, Inc. (2017)]. Aufgrund der auf AESLink basierenden Simulationsumgebung und deren bereits bestehenden Softwaretools zur Interaktion und Interprozesskommunikation mit Matlab und Simulink, wurde dieses als Framework zur Entwicklung der neuronalen Netze ausgewählt (siehe [Hasberg (2017a)]).

Daraufhin wurden die zur Verfügung stehenden Netztypen, -Architekturen und -Parameter, der Trainingsprozess, die Dateneingabe und -ausgabe sowie der generelle Umgang und die typische Arbeitsweise mit der Neural Network Toolbox evaluiert und erarbeitet. Als Ergebnis wurden verschiedene Matlab-Skripte entwickelt (siehe [Hasberg (2017a)]). Unter anderem übernehmen diese die Präparation der Daten, wodurch in AESLink aufgezeichnete Flüge unkompliziert und ohne Rechenzeitaufwand als Trainingsdaten für die Netze dienen. Weiterhin wurden für jeden als geeignet angesehenen Netztyp eigene Trainings- und Klassifizierungsskripte geschrieben. In diesen werden die Trainingsdaten eingelesen und die Netze im Matlab-Workspace instanziiert und trainiert. Im Anschluss erfolgt eine Klassifizierung, bei welcher die Trainingsergebnisse geplottet werden und eine Vorabanalyse der Performance der Funktionsapproximation stattfinden kann. Die Performance der Funktionsapproximation korreliert mit der Güte der Nachbildung des Flugverhaltens.

Im nächsten Arbeitsschritt (siehe [Hasberg (2017b)]) wurde eine mögliche Einbettung der trainierten Netze in eigene Anwendungen evaluiert und diskutiert. Aus den Erkenntnissen wurden die bereits vorhanden Matlab-Skripte dahingehend erweitert, dass nach jedem Trainingsprozess eine in .NET Anwendungen einbindbare DLL generiert wird, welche die trainierten Netze enthält. Durch diese DLL können die trainierten Netze außerhalb der Matlab Umgebung verwendet werden. Zuletzt wurde ein Mockup entwickelt, welcher in der Simulationsumgebung den Fault Detector darstellt und wie dieser agiert. Dadurch wurde gezeigt, dass das Konzept der Arbeitsumgebung zur Entwicklung und anschließenden Verwendung der neuronalen Netze in eigenen Anwendungen möglich ist.

Im Arbeitspaket der Masterthesis wurden die bereits entwickelten Skripte und die gewonnenen Erkenntnisse dazu verwendet, eine automatisierte Arbeitsumgebung zu realisieren. Diese

ermöglicht ein einfaches Erzeugen und Trainieren der Netze. Weiterhin werden automatisiert Schritte zur Einbettung in eigene Anwendungen durchgeführt. Im folgenden werden zunächst die verfügbaren Netztypen erläutert. Daraufhin wird die Arbeitsumgebung anhand des Workflows beschrieben.

6.2.1. Netztypen

Neuronale Netze werden in Matlabs Neural Network Toolbox über spezielle Befehle instanziiert und sind daraufhin als Objektinstanzen im Workspace vorhanden. Ein gewöhnliches Feed-Forward Netz z. B. durch den Aufruf von *feedforwardnet*. Das instanziierte Objekt bietet viele Optionen zur Parametrierung.

Über verschiedene Parameter können die Anzahl der Hidden-Layer und deren Neuronen bestimmt werden. Die Anzahl der Neuronen im Input- und Output-Layer wird anhand der Vektorlängen der Features und Labels beim Starten des Trainings automatisch von Matlab bestimmt.

Weiterhin können die zu verwendenden Aktivierungsfunktionen bestimmt werden, wobei alle Neuronen in einem Layer stets die gleiche Aktivierungsfunktion nutzen. Neben den in Abschnitt 2.3.1 genannten stehen weitere, hier nicht beschriebene Aktivierungsfunktionen, zur Verfügung.

Als Trainingsparameter können etwa die maximalen *Epochen*, der Trainingsalgorithmus oder das Target eingestellt werden. Die Targets können der Hauptprozessor (*CPU*) oder der Grafikprozessor (*GPU*) sein. Als Trainingsalgorithmen stehen mehrere Verfahren, wie etwa *Levenberg-Marquardt backpropagation*, *Bayesian Regularization backpropagation*, *Gradient Descent backpropagation* oder *Scaled conjugate gradient backpropagation* zur Auswahl.

Matlab unterscheidet für Regressionsnetze grundsätzlich zwischen *Feed-Forward*-Typen (*FF*) und *Time-Series-Prediction*-Typen (*TS*). TS-Typen sind speziell für die Verwendung von sequentiell vorliegenden Daten ausgewiesen und eignen sich daher für dynamische Systeme. Den FF-Typen kann ein Sliding-Window an Daten vorgegeben werden, um mit diesen sequentielle Daten verarbeiten zu können. Mit der Arbeitsumgebung kann aus insgesamt 8 verschiedenen Netztypen ausgewählt werden.

Abbildung 6.3 zeigt beispielhaft ein *Feed Forward Net* (FF) mit folgender Parametrierung:

- 20 Neuronen im Input-Layer
- 1 Neuron im Output-Layer mit linearer Aktivierungsfunktion
- 1 Hidden-Layer mit 30 Neuronen tanh Aktivierungsfunktion

Die 20 Neuronen im Input-Layer stellen zwei Features mit einer Sliding-Window Länge von 10 Elementen dar. Die Features können so etwa die letzten 10 Rollmoment-Eingaben des Autopiloten und die letzten 10 Rollraten des Lagesensors sein. Der Output ist die geschätzte Rollrate für den nächsten, bzw. diesen Regelschritt.

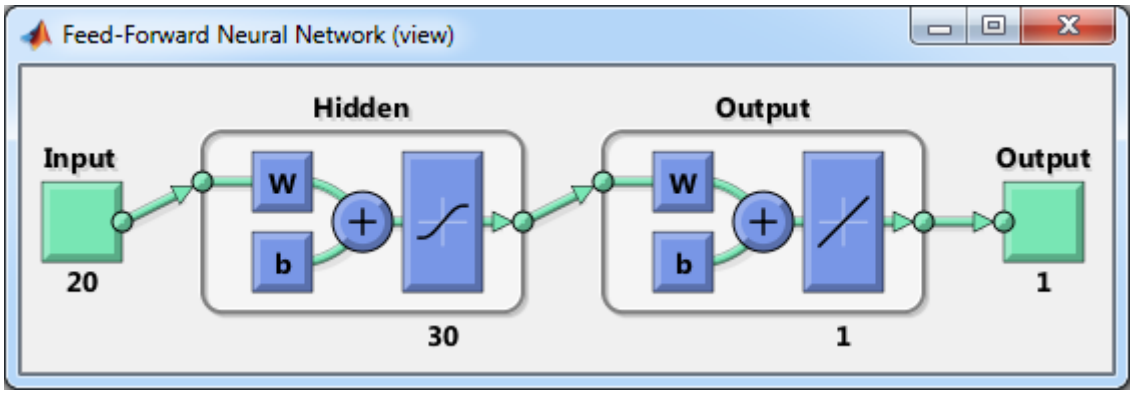

Abbildung 6.3.: Schematische Darstellung eines Feed-Forward Netzes in Matlab

Ein *Function Fitting Net* (FF) ist speziell als Netz zur Regression von kontinuierlichen Datenwerten ausgewiesen und vorgesehen[1]. Es handelt sich dabei um ein für diese Zwecke vorparametriertes Feed Forward Net.

Ein *Radial Basis Net* (FF) zeichnet sich dadurch aus, dass im Hidden-Layer Radiale Basisfunktionen als Aktivierungsfunktionen verwendet werden. Für dieses gilt eine Besonderheit bei der Instanziierung, da die Anzahl der Neuronen im Hidden-Layer vorher nicht bestimmt werden können. Stattdessen wird ein einstellbarer Fehlerwert vorgegeben. Durch das Training werden solange Neuronen selbstständig hinzugefügt, bis der Fehlerwert erreicht ist.

Bei einem *Time Delay Net* (TS) werden die Eingangsdaten jeweils um *n*-viele *tapped-delay lines* verzögert, was letztlich einem Feed-Forward Netz mit Sliding-Window Verfahren entspricht. Somit ist eine der beiden Optionen überflüssig. Jedoch wird das Sliding-Window hier von Matlab intern verwaltet. Es wird davon ausgegangen, dass der Trainingsprozess u. U. effizienter vonstatten gehen kann, da das Sliding-Window von Matlab intern effizienter gespeichert werden kann.

Ein *Distributed Delay Net* (TS) erweitert ein Time Delay Net um tapped-delay lines für die Neuronen im Output-Layer.

Bei einem *Layer Recurrent Net* (TS) handelt es sich um ein rekurrentes *Elman*-Netz [Elman (1990)]. Bei diesem wird der Ausgang des Hidden-Layers auf sich selbst zurückgeführt. Die

[1] http://www.mathworks.com/help/releases/R2016a/nnet/ref/feedforwardnet.html [Stand: 13.05.2018]

Anzahl der zurückgeführten Zeitschritte kann angegeben werden. Weiterhin können für den Input tapped-delay lines eingestellt werden.

Ein *Nonlinear autoregressive neural network with external (exogenous) input* (*NARX-Net*) ist eine abgewandelte Form eines Layer Recurrent Nets. Bei diesem werden nicht die Hidden-Layer untereinander zurückgeführt, sondern der Output mit einer Verzögerung auf den Input. Ebenfalls können für den Input tapped-delay lines eingestellt werden.

Weiterhin sollten *Long short-term memory* (*LSTM*) Netze [Hochreiter und Schmidhuber (1997)] in der Arbeitsumgebung vorhanden sein. Diese haben das Potential einige Probleme von rekurrenten neuronalen Netzen, wie etwa ineffizientes Training, schwierige Parametrierung oder ein zu kleines Gedächtnis zu lösen [Olah (2015)]. Besonders langfristige Abhängigkeiten innerhalb der sequentiell vorliegenden Daten können mit LSTMs effektiv verarbeitet werden [Karpathy (2015)]. Für weitere Informationen sei auf [Olah (2015)] verwiesen.

In der verwendeten Matlab Version (*R2016a*) gibt es keine Implementierung von LSTMs. Daher wurde die frei verfügbare Bibliothek *Cortexsys* [Cox (2016)] verwendet und in die Arbeitsumgebung integriert.

6.2.2. Workflow der Arbeitsumgebung

Die Arbeitsumgebung besteht aus insgesamt über 30 Matlab-Skripten. Um ein neues Netz zu erzeugen und zu trainieren, bzw. einen anderen Datensatz zu trainieren, existieren mehrere einfach zu verwendende *Fassaden*-Skipte. Das nachfolgende Listing zeigt das *Do_Train.m* Skript:

```
%% FF Types
nnFF = Net_StandardParams(10,'yes','FFNet',  'RollSimple_CS');
nnFT = Net_StandardParams(10,'yes','FitNet', 'All_VCS', 'FTN_DSA1');
nnRB = Net_StandardParams(10,'no', 'RBFNet', 'All_CS');
%% TS Types
nnTD = Net_StandardParams( 1,'no', 'TimeDelayNet', 'All_VCS', );
nnDD = Net_StandardParams( 1,'no', 'DistDelayNet', 'P_to_VCS_roll');
nnLR = Net_StandardParams( 1,'no', 'LayRecNet',    'RollSimple_VCS');
nnNX = Net_StandardParams( 1,'no', 'NarxNet',      'All_CS', 'TDN_DSB1');
%% LSTM Types
nnLS = Net_StandardParams( 1,'no', 'LSTMNet', 'RollSimple_VCS');

%% Change standard parameters for a specific instance
nnFF.normParams.scaleDo = 1;
```

```
nnFF.normParams.shiftDo = 1;
% This datafile is global for all nets
nnFF.dataParams.dataSource = 'data/flight_8.mat';

%% FF Types
nnFF = Train(nnFF);
nnFT = Train(nnFT);
nnRB = Train(nnRB);
%% TS Types
nnTD = Train(nnTD);
nnDD = Train(nnDD);
nnLR = Train(nnLR);
nnNX = Train(nnNX);
%% LSTM Types
nnLS = Train(nnLS);
```

Listing 6.1: Fassaden-Skript *Do_Train.m*

Mit dem Skript *Net_StandardParams.m* wird zunächst für jeden Netztyp eine Parameterstruktur (*NetStruct*) erzeugt. Der erste Parameter gibt die Sliding-Window Länge der Features an. Der zweite Parameter gibt an, ob das Training auf dem Grafikprozessor durchgeführt werden soll. Über einen optionalen fünften Parameter kann ein Name vergeben werden.

Als vierter Parameter wird ein String *NetTask* angegeben. Dieser gibt an, welche Daten aus dem Logfile als Features und Labels verwendet werden sollen. Bei dem String *RollSimple_VCS* wird etwa die Rollrate und das Steuersignale für den Freiheitsgrad Rollen als Features verwendet und die Rollrate des nächsten Regelschrittes als Label. Beim String *All_CS* werden zum Beispiel alle Drehraten, die lineare Beschleunigung in Längsrichtung sowie alle Ansteuerungen der Steuerflächen als Features verwendet. Die Labels bestehen aus den Drehraten und der Beschleunigung des nächsten Regelschrittes. Aus Sicht der Systemmodellierung gibt der NetTask die Struktur der Übertragungsfunktion des Systems, bzw. die Struktur der in Abschnitt 2.2 beschriebenen Bewegungsgleichung, vor. Die Features bilden dabei die Eingangsvektoren x und u, die Labels den Ausgangsvektor $\dot{x}$. Das nachfolgende Listing zeigt die beiden beschriebenen NetTasks *RollSimple_VCS* und *All_CS*. Die *indices* geben jeweils die Zugehörigkeiten zwischen den Vektoren v, u, x und den Ein- und Ausgangsneuronen an.

```
if    (strcmp(netTask, 'RollSimple_VCS')  == 1)
indices_in  = ... % 2 input neurons
```

```
[indices.angularRate_P; indices.ctrlsAP_roll];
indices_out =  ... % 1 output neuron
[indices.angularRate_P];

elseif(strcmp(netTask, 'All_CS') == 1)
indices_in  =  ... % 12 input neurons
[indices.angularRate_P; indices.csAil_l; indices.csAil_r; ...
 indices.angularRate_Q; indices.csElv_l; indices.csElv_r; ...
 indices.accLinear_X;   indices.csMot_l; indices.csMot_r; ...
 indices.angularRate_R; indices.csRud_l; indices.csRud_r];
indices_out = ... % 4 output neurons
[indices.angularRate_P; ...
 indices.angularRate_Q; ...
 indices.accLinear_X;   ...
 indices.angularRate_R];
```

Listing 6.2: Beispiele für vordefinierte *NetTasks* aus dem Skript *Net_StandardTaskParams.m*

Die beiden Listings zeigen jeweils lediglich vom Autor vorkonfigurierte Optionen. Die Skripte können ohne großen Aufwand erweitert werden, so dass zum Beispiel mehrere Netze eines Typs mit unterschiedlichen Sliding-Window Längen oder NetTasks mit einem Aufruf erzeugt und trainiert werden können. Neue NetTasks können im Skript *Net_StandardTaskParams.m* durch das Ergänzen eines weiteren *elseif* Falls hinzugefügt werden.

Im Skript *Net_StandardParams.m* werden die NetStructs mit Standardparametern belegt. So etwa die Parameter für das Präparieren der Daten, den Netztyp, die Layer- und Neuronen-Anzahlen, die zu nutzenden Aktivierungsfunktionen, die Vektorlängen der Rückführungen für TS-Typen, den Trainingsalgorithmus und die Trainingsparameter, usw. Auch hier kann eine Anpassung, bzw. Erweiterung ohne großen Aufwand erfolgen. Im mittleren Abschnitt der *Do_Train.m* Fassade können die Parameter für jede NetStruct individuell festgelegt werden, falls benötigt.

Nach der individuellen Festlegung der Parameter wird das *Train.m* Skript für jedes Netz ausgeführt, wobei nicht benötigte oder gewollte Instanzen auskommentiert werden können. Dieses Skript instanziiert die Netze zunächst und führt daraufhin das Training durch. Zur Vorabanalyse kann das Skript *Classify.m* mit dem NetStruct als Skript-Parameter aufgerufen werden. Hier wird der Fit zwischen tatsächlichem und approximiertem Funktionsverlauf sowie der berechnete MSE-Wert für den kompletten Trainingsdatensatz ausgegeben und geplottet.

Die Fassaden-Skripte sind lediglich als Beispiele und Einstiegspunkte zu sehen. Für eine produktiv genutzte Arbeitsumgebung können eine Vielzahl an Fassaden-Skripten verwendet werden, die zum Beispiel automatisch mehrere Netze des gleichen Typs mit unterschiedlichen Parametern versehen, sie instanziieren und trainieren, und daraufhin das Netz mit der besten Performance speichern, wobei die anderen verworfen werden. Nachfolgend werden die einzelnen Arbeitsschritte und Parameter erläutert.

Einlesen und Präparieren der Daten

Sechs der Skripte sind für das Einlesen und die Präparation der im Datenformat von AESLink aufgezeichneten Flüge zuständig. Die Daten müssen für die Verwendung als Eingangsdaten für die Netze präpariert und umgewandelt werden. Das nachfolgende Listing ist ein Auszug aus dem Skript *Net_StandardParams.m* und zeigt die möglichen Konfigurationsparameter des Datensatzes.

```
1  %% Data Source
2  dataSource = 'data/flight_8.bin';
3  %dataSource = 'data/flight_8.mat';
4
5  %% Sliding-Window
6  sw_size_in  = 1; % e.g. 10 for FF-Types
7  sw_size_out = 1;
8
9  %% Norm Parameters
10 cutFrontIdx = 0;
11 cutBackIdx  = 0;
12 scaleDo     = 1;
13 shiftDo     = 1;
```

Listing 6.3: Parameter zur Präparation der Trainingsdaten

Zunächst werden die binär vorliegenden Daten eingelesen (2) und daraus für Matlab direkt verwendbare *.mat* Dateien erzeugt. Dieser Schritt muss allerdings lediglich ein einziges Mal pro Logfile durchgeführt werden. Befinden die Daten sich im Workspace, kann Zeile 2 leer gelassen werden. Ist die Umwandlung bereits erfolgt, aber es befinden sich keine Daten im Workspace, kann die zu verwendende .mat Datei in Zeile 3 angegeben werden.

Die ersten Datensätze können entfernt werden (10), um das Training mit einem eingeschwungenen Flugzustand oder einem beliebigen Regelschritt zu beginnen. Weiterhin können die letzten Datensätze entfernt werden (11).

Zur Normalisierung der Daten (siehe Abschnitt 2.3.1) können diese gesondert skaliert (12) und zentriert (13) werden. Bei Deaktivierung wird dies von Matlab selbstständig durchgeführt.

Zur Vorhersage des nächsten Systemzustands müssen die Eingangsdaten um einen Regelschritt verschoben werden, um sie als Labels verwenden zu können.

Daraufhin werden die Features (6) und Labels (7) in parametrierbare Sliding-Windows formatiert.

Im nächsten Schritt werden die Daten umgewandelt, damit diese für die vordefinierten Netztypen verwendet werden können. Die FF-Typen verwenden Vektoren und Matrizen. Für diese ist keine Konvertierung nötig, da Daten bereits in diesem Format vorliegen. Die TS-Typen verwenden hingegen *Cell*-Datentypen. Daher müssen die Vektoren zunächst in Cells konvertiert werden. Nach diesem Arbeitsschritt liegen die Daten im richtigen Format vor.

Mit den enthaltenen Skripten können aufgezeichnete Flüge unkompliziert und ohne Rechenzeitaufwand als Trainingsdaten für die Netze genutzt werden.

Erzeugung und Trainieren der Netze

Das Skript *Train.m* führt zunächst die oben beschriebenen Schritte zur Präparation der Daten durch. Dabei wird aus dem Parameter NetTask abgeleitet, welche der Daten als Features und Labels verwendet werden sollen. Daraufhin werden die Netze mit dem Skript *Train_CreateNet.m* instanziiert. Für jeden Netztyp existieren eigene Befehle zur Instanziierung im Matlab-Workspace, wie etwa *fitnet* oder *narxnet*. Als Parameter werden bei den FF-Typen die Hidden-Layer Konfigurationen und die Trainingsfunktion angegeben. Für die TS-Typen kommen die Längen der Rückführungen hinzu. Die Instanziierung eines LSTM Netzes aus Cortexsys ist ungleich komplexer und wird hier vernachlässigt.

Nach der Instanziierung werden für jedes NetStruct zunächst die *Call Strings* erzeugt. Im Zuge dessen werden für die TS-Typen die zurückgeführten Zustände erzeugt und vorbelegt. Beide werden im nächsten Abschnitt erläutert.

Im nächsten Schritt werden die Netze trainiert. Hierfür wird lediglich der Befehl *train* mit den Features und Labels der Trainingsdaten als Übergabeparameter aufgerufen. Falls das Training mit dem Grafikprozessor durchgeführt werden soll, muss dies als Argument übergeben werden. Weiterhin sind dafür je nach Netztyp verschiedene Datenkonvertierungen nötig, für welche die Neural Network Toolbox jedoch fertige Funktionen anbietet.

Die Dauer des Trainings ist von der Komplexität der zu approximierenden Funktion und der Größe des Trainingsdatensatzes abhängig. Als Abbruchkriterien können die maximale Anzahl an Epochen oder ein minimaler Fehlerwert festgelegt werden. Beide Parameter sind über das Skript *Net_StandardParams.m* konfigurierbar. Bei Beendigung des Trainings wird die benötigte Zeit ausgegeben.

Persistieren des Netzes

Nach dem Training ist ein trainiertes Netz als Objektstruktur im Matlab-Workspace vorhanden. Somit ist es nach dem Beenden der Session nicht länger verfügbar. Um ein Netz zu persistieren, wird aus diesem mit dem Befehl *genFunction* ein eigenes Matlab-Skript generiert. Im Skript sind die antrainierten Gewichtungen und Biase, die Aktivierungsfunktionen sowie etwaige Normierungsfunktionen eingebettet, sodass keine Abhängigkeiten zu anderen Matlab Dateien oder Skripten bestehen. Neben der Persistenz ergibt sich der Vorteil, dass das Netz ausgeführt werden kann, ohne dass die Neural Network Toolbox installiert oder vorhanden sein muss.

Das Skript selbst ist eine aufrufbare Funktion. Der Funktionsrumpf entspricht dem oben beschriebenen Call String. Der Datei- und Funktionsname wird bei der Instanziierung des NetStructs als optionaler fünfter Parameter übergeben. Die Parameterliste der erzeugten Funktion besteht mindestens aus einem Vektor der Features. Bei einem Sliding-Window mit der Länge > 1 werden die Daten an den Vektor angehangen. Sollen mehrere Schritte mit einem Aufruf berechnet werden, ist eine Matrix zu übergeben.

Für TS-Typen müssen neben den Features auch die internen Zustände übergeben werden. Diese werden vom Skript selbst verarbeitet und als Rückgabewerte zurückgegeben. Diese Parameter sind für einen Aufrufer daher ausschließlich als Speicherobjekte zu bewerten. Sie müssen lediglich vor der ersten Ausführung als Cell-Strukturen erzeugt werden. Je nach TS-Typ ist die Erzeugung unterschiedlich komplex und es sind verschiedene Datenkonvertierungen nötig.

Neben dem Netz selbst wird mit gleichem Dateinamen eine *.mat*-Datei[2] erzeugt, welche alle Werte der Datenstruktur NetStruct enthält. Nach dem Trainingsvorgang werden somit das Netz sowie dessen Parameter automatisch persistiert.

[2] *.mat*-Dateien haben ein binäres Datenformat und werden von Matlab für das Persistieren von Variablen und Daten genutzt.

Klassifizierung und Test

Zur Vorabanalyse mit verschiedenen Datensätzen stehen das Skript *Classify.m* sowie eine eigene Fassade *Do_Classify.m* zur Verfügung. Das nachfolgende Listing zeigt die Fassade, wobei beispielhaft lediglich ein NetStruct in Verwendung ist:

```
%% FF Types
nnFF = Classify(nnFF);
%nnFT = Classify(nnFT);
%nnRB = Classify(nnRB);

%% TS Types
%nnTD = Classify(nnTD);
%nnDD = Classify(nnDD);
%nnLR = Classify(nnLR);
%nnNX = Classify(nnNX);

%% LSTM Types
%nnLS = Classify(nnLS);
```

Listing 6.4: Fassaden-Skript *Do_Classify.m*

Da die Netze nicht länger auf den Matlab-Workspace der Session angewiesen sind, kann der Klassifizierungsprozess vom Training entkoppelt ausgeführt werden. Daher müssen zunächst die gleichen Schritte zum Einlesen und Präparieren der Daten sowie zum Erzeugen der für die TS-Typen benötigten internen Zustände durchgeführt werden. Zur Ausführung der persisiterten Netze wird mit dem Matlab-Befehl *eval(callString)* das generierte Funktions-Skript aufgerufen. Als Parameter des Befehls wird der Call String verwendet. Durch diesen sind die Variablennamen der Features und Zustände sowie der Rückgabewerte vorgegeben.

Nach dem Aufruf liegen die berechneten Werte vor. Diese werden zusammen mit den Labels geplottet und können so verglichen werden. Weiterhin werden die drei in Abschnitt 4.2.1 beschriebenen Fehlerquadratsummen berechnet und ausgegeben. So kann eine Vorabanalyse durchgeführt werden, wodurch der Trainingserfolg überprüft und die Performance der Nachbildung des Flugverhaltens abgeschätzt werden kann.

Neben der statischen Klassifizierung wurde ein Skript zur Live-Klassifizierung implementiert, welches zum aktuellen Entwicklungsstand jedoch lediglich Netze FF-Typen unterstützt. Das Skript kann während eines Simulationsdurchlaufs verwendet werden und gibt den mit der Gleichung 4.2.3 berechneten MSE-Wert aus und plottet diesen zur Laufzeit. Die vollständige

Umsetzung der Live-Klassifizierung per Matlab-Skript wurde verworfen, da dieses aufgrund unzureichender Echtzeitfähigkeit seitens Matlab Geschwindigkeitsprobleme aufweist (vgl. [Hasberg (2017a)]). Die Klassifizierung im live simulierten Regelkreis muss daher durch eine eigenständige Anwendung, welche außerhalb der Matlab Instanz läuft, ausgeführt werden.

Einbettung in eigene Anwendung

Zur Einbettung in eigene Anwendungen wurden verschiedene Verfahren evaluiert (vgl. [Hasberg (2017b)]). Umgesetzt wurde die Erzeugung einer *.NET*-DLL, welche in mit dem .NET-Framework kompatible Anwendungen eingebunden werden kann. Die DLL enthält die Netze als aufrufbare Methoden.

Zur Realisierung wurde das *Matlab Compiler SDK*[3] (*MCS*) verwendet. Mit diesem lassen sich gewöhnliche Matlab-Skripte in verschiedene Laufzeitbibliotheken einbetten. Unterstützt werden z.B. *.NET DLLs*, *Java Packages*, *Python Packages* und *C/C++ DLLs*. Zur Verwendung der Laufzeitbibliothek muss Matlab nicht installiert sein. Somit kann diese in Umgebungen verwendet werden, auf der das Matlab Softwarepaket nicht installiert ist oder werden kann. Jedoch wird die *Matlab Compiler Runtime*[4] (*MCR*) benötigt, welche kostenfrei heruntergeladen und installiert werden kann. Neben den erzeugten Laufzeitbibliotheken muss auch die MCR-Laufzeitbibliothek in die eigenen Anwendungen eingebunden werden.

Für das Aufrufen der einkompilierten Skripte, bzw. Funktionen, müssen zwischen .NET Datentypen und denen, welche die MCR benötigt und erwartet, konvertiert werden. Hierfür stellt das MCR verschiedene Funktionen zum Marshalling bereit. Dennoch ist diese Konvertierung je nach Komplexität der Übergabeparameter und Rückgabewerte mit hohem Programmieraufwand verbunden. So werden etwa für Cell-Datentypen oder tief-geschachtelte Structs deutlich mehr Lines of Code benötigt, als für einfache Vektoren oder Matrizen.

Zur Erzeugung der .NET-DLL wurde das Fassaden-Skript *Do_GenerateDLL.m* implementiert. Dieses ruft das MCS auf, welches die DLL generiert. Zunächst wurde über die Konfigurationsoberfläche des MCS eine Basis Projektdatei angelegt, welche einem XML ähnlichen Aufbau folgt. In dieser wurden verschiedene Optionen vorkonfiguriert, wie etwa der Name der zu erzeugenden DLL und andere Metadaten einer .NET-Assembly. Jedoch enthält sie noch keine Informationen über die einzubettenden Skripte. Das Fassaden-Skript ruft ein weiteres Skript auf, welches die Basis Projektdatei temporär dupliziert und die Dateipfade der einzubettenden Skripte und eventuell weiterer benötigter *.mat*-Dateien in die dafür vorgesehenen XML-Knoten der duplizierten Projektdatei schreibt. Hierfür mussten Einschränkungen in Kauf

[3] https://de.mathworks.com/products/matlab-compiler-sdk.html [Stand: 15.05.2018]
[4] https://de.mathworks.com/products/compiler/mcr.html [Stand: 15.05.2018]

genommen werden. Um die einzubettenden Dateien keiner Inhaltsanalyse, ob diese eingebettet werden soll oder nicht, unterziehen zu müssen, werden lediglich Dateien aus zwei bestimmten Verzeichnissen eingebettet. Diese sind die Unterverzeichnisse *Nets* und *Resources*. Die oben beschriebene automatische Persistierung der Netze und NetStructs speichert diese standardmäßig im Verzeichnis Nets. Im Verzeichnis Ressources liegen die für die Ausführung der LSTM-Netze benötigten Skripte und Dateien der Bibliothek Cortexsys. Jedoch konnten die LSTM-Netze trotz aufwendiger Fehlersuche in einer eigenen Anwendung nicht zur Ausführung gebracht werden. Beim Versuch folgt stets eine Exception, welche von der Funktionsbibliothek der MCR erzeugt wird. Daher können zum aktuellen Stand der Arbeitsumgebung keine LSTM-Netze in eigene Anwendungen eingebettet werden.

Nach dem Schreiben der Dateipfade in die temporäre Projektdatei ruft das Fassaden-Skript den Matlab Compiler des MCS mit dieser als Übergabeparameter auf. Dieser generiert daraufhin die .NET-DLL. Die DLL enthält alle Netze, welche sich zum Zeitpunkt des Aufrufs im Unterverzeichnis Nets befinden. Somit kann die Einbettung zusätzlich per Hand gesteuert werden. Neben den Netzen enthält die DLL die NetStructs. Diese enthalten Metadaten der zugehörigen Netz-Skripte, welche von der eigenen Anwendung benötigt werden. So zum Beispiel die Anzahl der Ein- und Ausgangsneuronen, die Länge der Sliding-Windows, die Zugehörigkeit zwischen den Vektoren v, u, x und den Ein- und Ausgangsneuronen (NetStruct), usw. Ohne diese Metadaten kann die eigene Anwendung nicht entscheiden, wie das Netz aufgerufen werden muss und welche Bedeutung welcher Parameter oder Rückgabewert hat.

In der erzeugten DLL sind somit alle Informationen enthalten, um die trainierten neuronalen Netze in eigenständigen .NET Anwendungen ausführen zu können und mit diesen eine Ausfallerkennung zu realisieren. Wie oben beschrieben, können lediglich die LSTM-Netze zum aktuellen Entwicklungsstand nicht genutzt werden.

6.3. Fault Detector

Zur Realisierung der Ausfallerkennung in live durchgeführten Simulationen wurde der *Fault Detector* implementiert. Dieser empfängt die Vektoren v, u sowie x und schätzt den Integritätszustand der Funktionsfähigkeit der Steuerflächen, bzw. der Steuerbarkeit der Freiheitsgrade, ab. Die Abschätzung wird als Vektor i in jedem Simulationsschritt über den Softwarebus verteilt. Die interne Umsetzung folgt dem in Abschnitt 4.2 beschriebenen Ansatz.

6.3.1. Benutzeroberfläche

In Abbildung 6.4 ist die Benutzeroberfläche des Fault Detectors dargestellt. In der Groupbox *Observation Source* kann angegeben werden, ob der Integritätszustand i die Abschätzung der virtuellen Steuersignale (*Virtual CS*) oder der realen Steuerflächen (*Real CS*), bzw. beide, enthält. Die Auswahl ist in der AESLink Nachricht, welche den Vektor i enthält, kodiert. Zu der Auswahl korreliert der untere Bereich der Benutzeroberfläche. Hier existiert für jede Auswahl jeweils eine Registerkarte (*Tab*). Exemplarisch wird nachfolgend der Tab Virtual CS erläutert, wobei die Erläuterungen analog für den Tab Real CS gelten.

Im oberen Bereich des Tabs werden die einzelnen Abschätzungen dargestellt. Diese korrelieren zu den acht virtuellen Steuersignalen, die im AESLink Framework verwendet und standardmäßig benannt sind. Die Werte stellen die Integritätsabschätzungen in normierten Prozenten dar. In der Groupbox *Net to Integrity Calculation* kann ausgewählt werden, ob die Abschätzungen durch einzelne Netze durchgeführt werden soll oder ein Netz für alle Werte verwendet wird. Je nach Auswahl ist entweder die Combobox in der Groupbox *Net Single* aktiv oder alle anderen Comboxen und die genannte nicht. Die Auswahl wurde eingeführt, um eine höhere Flexibilität bei der Übertragung zu erreichen.

Über die Comboboxen kann das zu verwendende Netz vorgegeben werden, welche das Flugverhalten und die Systemzustände abschätzt. Beim Start der Anwendung werden die möglichen Netze aus der oben beschriebenen DLL geladen. Die Abbildung 6.5 zeigt beispielhaft eine Auswahl von in der DLL eingebetteten Netzen. Wird als Netz *Tool* ausgewählt, können die Integritätsabschätzungen händisch in die Textboxen eingetragen werden.

6.3.2. Auswahl des Netzes

Bei Auswahl eines Netzes wird zunächst das zugehörige NetStruct aus der DLL geladen. Daraufhin werden die internen Mechanismen zur Formatierung der empfangenen Daten auf das ausgewählte Netz eingestellt. Weiterhin werden die zum Netz passenden Funktionen, welche die DLL anbietet, über .NET *Reflection*-Techniken enumeriert und abgespeichert. Danach befinden sich die Interna des Fault Detectors in einem Zustand, in dem die eingebetteten Matlab-Skripte als Funktionen aufgerufen werden können.

6.3.3. Berechnung des Systemzustands

Die Abschätzung der Integritätszustände erfolgt in diskreten Simulationsschritten. Hierbei wird in jedem Schritt auf den Empfang der Vektoren v, u und x gewartet. Sind alle eingetroffen, werden die im vorherigen Regelschritt empfangenen Vektoren für den Aufruf in die vom

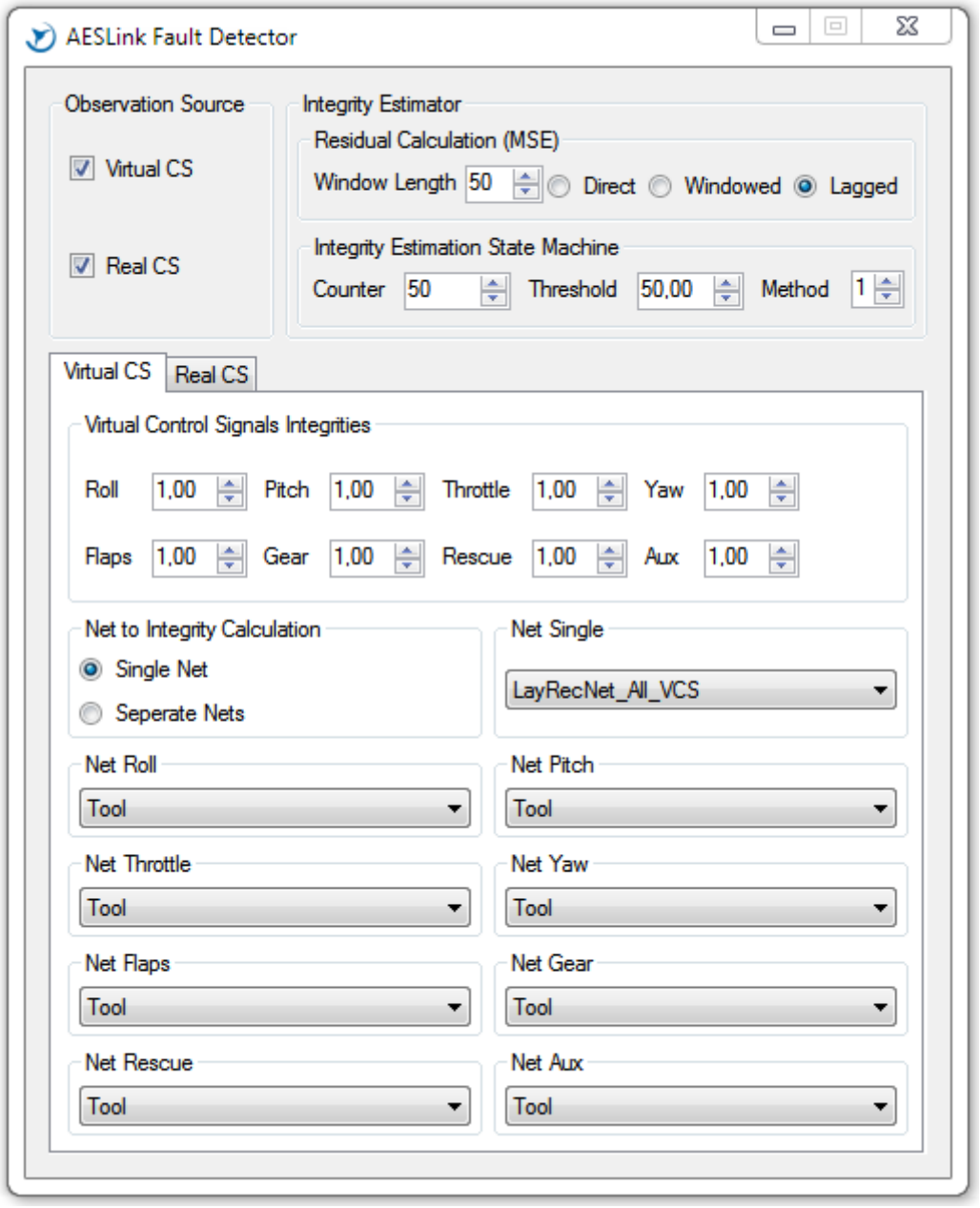

Abbildung 6.4.: Benutzeroberfläche des Fault Detectors

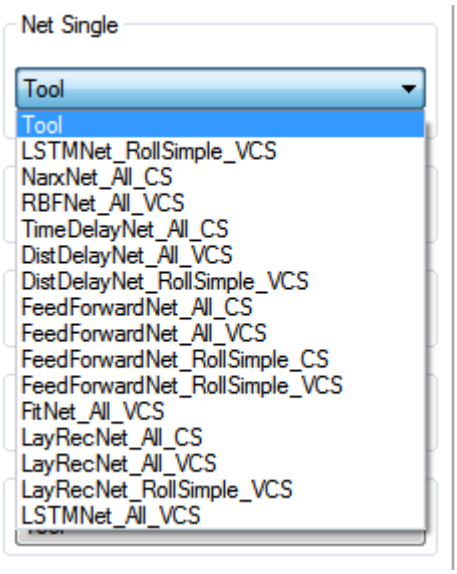

Abbildung 6.5.: Auswahl der in der DLL eingebetteten neuronalen Netze

MCR benötigten Datentypen formatiert. Daraufhin erfolgt die Ausführung des Netzes und es wird der geschätzte Systemzustand für den aktuellen Regelschritt vom Netz berechnet. Durch die Indizes in der NetStruct erfolgt die Zuordnung, welches Ausgangsneuron zu welcher Komponente des Vektors x gehört.

6.3.4. Berechnung des Restwerts

Aus dem geschätzten und dem empfangenen Systemzustand muss zunächst der Restwert gebildet werden. Über die Schaltflächen der Groupbox *Residual Calculation* kann die Berechnung parametriert werden. Es stehen die drei in Abschnitt 4.2.1 beschriebenen Methoden zur Auswahl. Die Länge des Fensters kann über die Textbox eingestellt werden. Je nach Auswahl und Fensterlänge werden die geschätzten und empfangenen Systemzustände der letzten Regelschritte intern zwischengespeichert.

6.3.5. Abschätzung des Integritätszustands

Zur Abschätzung des Integritätszustands aus dem Restwert wurde der in Abschnitt 4.2.2 beschriebene Zustandsautomat implementiert. Dieser kann über die Schaltflächen der Groupbox *Integrity Estimation State Machine* parametriert werden. Zum einen kann der Grenzwert angegeben werden, bei dessen Überschreitung durch den Restwert ein Zustandswechsel in den nächst höheren Zustand erfolgt. Zum anderen der maximale Zählerwert, für wie viele

Regelschritte im Zustand *fault observed* verweilt wird, bevor in den Zustand *fault confirmed* gewechselt wird. Durch die Variation des Zählers kann die maximal erlaubte Länge eines Spikes eingestellt werden.

Im Zustand *fault confirmed* erfolgt die Integritätsabschätzung. Diese ergibt sich aus der Divergenz zwischen geschätztem und empfangenen Systemzustand. Die berechnete Integritätsabschätzung wird als Wert in die Textboxen eingetragen. Die Zuordnung zwischen Freiheitsgrad und berechnetem Wert erfolgt hierfür erneut über das eingelesene NetStruct und wird mit dem NetTask vorgegeben.

Nachdem die Erzeugung aller durch Netze zu berechnenden Integritätsabschätzungen erfolgt ist, wird die zugehörige AESLink Nachricht erzeugt und über den Softwarebus verteilt. Hiermit ist der Regelschritt für den Fault Detector vollendet.

Mit dem Fault Detector wurde der in Abschnitt 4.2 präsentierte Ansatz zur Ausfallerkennung realisiert. Durch die Verwendung von angelernten neuronalen Netzen kann dieser die Integritätszustände der Freiheitsgrade und Steuerflächen in jedem Regelschritt einer live durchgeführten Simulation abschätzen.

7. Control Allocation

Das vorliegende Kapitel beschreibt die Realisierung der Komponente Control Allocation. Diese soll die Steuerbarkeit des Flugzeugs bei Teilausfällen der Steuerflächen sicherstellen. Anders als bei der Komponente Fault Detection sind hier keine vorherigen Arbeitsschritte nötig, um den in Abschnitt 4.3 präsentierten Ansatz zu realisieren. Zur Umsetzung wurde ausschließlich die Anwendung *Control Allocator* implementiert. Dieser stellt die ausführbare Anwendung der Komponente Control Allocation in der Systemarchitektur des simulierten Regelkreises aus Abbildung 5.1 dar.

7.1. Aufbau

In Abschnitt 2.1 wurde beschrieben, dass in einem Avioniksystem die Komponente Flugsteuerung den Träger vom Flugregler abstrahiert, so dass dieser für den Wechsel auf einen anderen Träger die gleichen Regler und Reglerstrukturen verwenden kann und dafür lediglich die Reglerparameter auf den neuen Träger hin angepasst werden müssen. Bei der Umsetzung der eigenen Flugsteuerung wurde versucht, auch diese möglichst abstrakt vom verwendeten Träger zu halten. Daher ist die primäre nicht-funktionale Anforderung an die Software, dass der Aufbau und die Algorithmen zur Steuerflächenzuordnung wenig statisch einprogrammierte Abhängigkeiten und Kenntnisse über den zu steuernden Träger aufweisen. Dies soll es vereinfachen, die Algorithmen auf andere Träger zu übertragen und diese verwenden zu können. Es sollen verschiedene Zuordnungsstrategien geboten werden, um den präsentierten Ansatz gegen etablierte Techniken evaluieren zu können.

In Abbildung 7.1 ist die Softwarearchitektur der Anwendung dargestellt. Der zentrale *Core Mediator* verbindet die Softwaremodule miteinander und vermittelt zwischen diesen. Zunächst empfängt er die Nachrichten v, i sowie x aus dem AESLink Netzwerk und antwortet mit der Nachricht der Ansteuerungen u. Den algorithmischen Kern der Steuerflächenzuordnung bilden die verschiedenen *Allocators*, welche jeweils unterschiedliche Zuordnungsstrategien implementieren. Um diese möglichst abstrakt gegenüber dem Träger realisieren zu können, wurde eine Datenstruktur *Aircraft Control Surfaces Configuration* (*ACCSC*) definiert. Durch

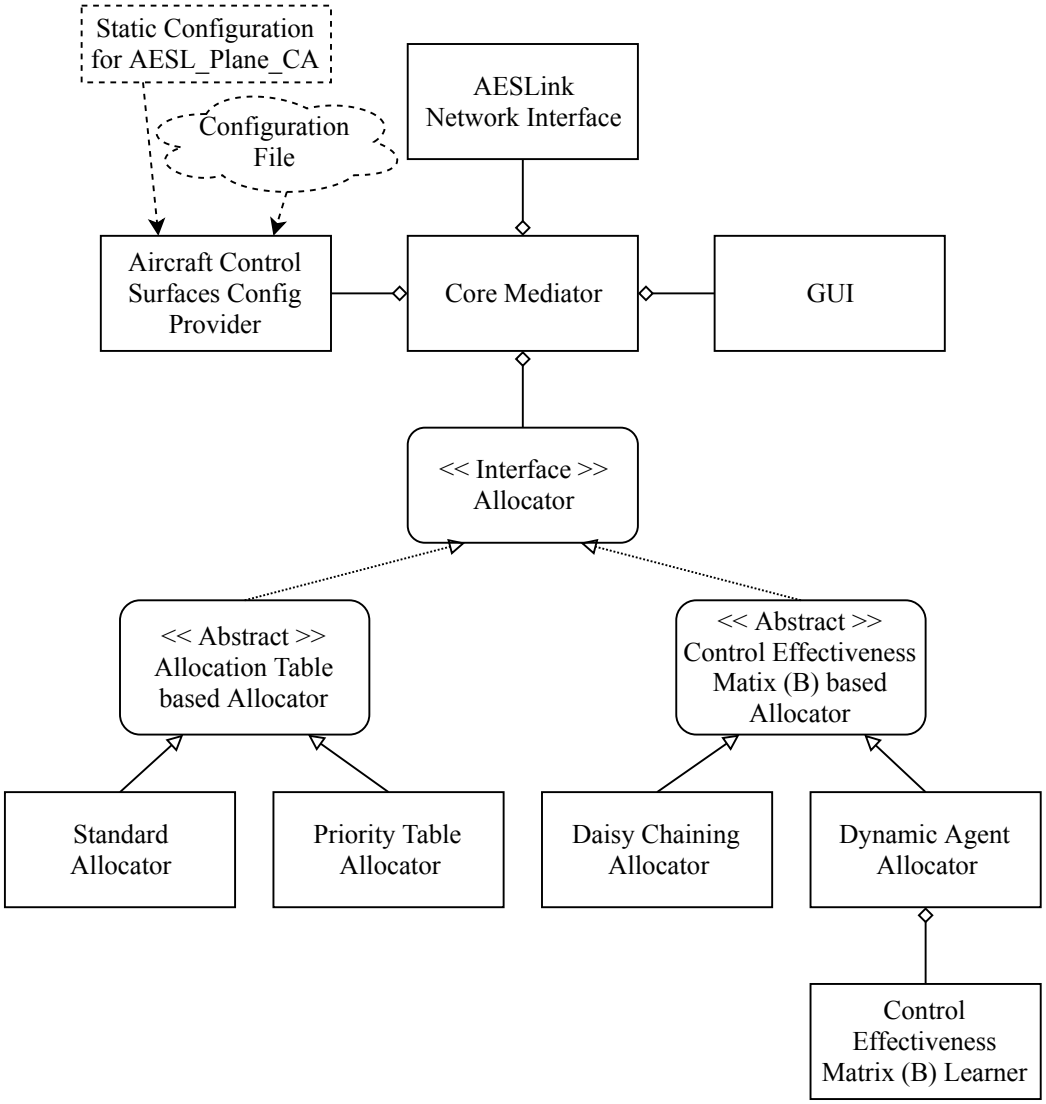

Abbildung 7.1.: Softwarearchitektur des Control Allocators

diese sind die Allokatoren parametrierbar. Die Datenstruktur soll durch eine Konfigurationsdatei abgespeichert und eingelesen werden können. Jedoch wurde das Einlesen zum aktuellen Stand noch nicht implementiert. Es wird daher eine statische Datenstruktur eingelesen, welche das in Abschnitt 5.4.1 beschriebene Flugmodell widerspiegelt.

7.2. Benutzeroberfläche

In Abbildung 7.2 ist die Benutzeroberfläche der Anwendung dargestellt. In der ComboBox *Allocator Type* kann der zu verwendende Allokator ausgewählt werden. In den Textboxen *Vector U* sind die Ausgangswerte des Allokators und damit der Anwendung selbst dargestellt. In der Tabelle *Allocation Table / Control Effectiveness Matrix (B)* sind Interna der Allokatoren dargestellt. Auf die Unterschiede wird im folgenden Abschnitt näher eingegangen. Gleiches gilt für die Tabelle *Vector V to U Mapping and Contribution*. Mit den Schaltflächen im unteren Bereich *Dynamic Agent Configuration* können Zustände und Parameter für den Allokatortyp, welcher den dynamischen Agenten implementiert, eingesehen und eingestellt werden. Die einzelnen Schaltflächen werden nachfolgend im Fließtext erläutert.

7.3. Allokatoren

Für die Evaluierung des Ansatzes, sowie der einfachen Wiederverwendung in anderen Projekten wurden insgesamt fünf verschiedene Allokatoren implementiert. Der einfachste ist statisch einprogrammiert und führt eine simple Zuordnung durch, welche lediglich zum verwendeten Flugmodell passt. Dieser statische Allokator kann als Mockup für die Weiterentwicklung von anderen Komponenten der Simulationsinfrastruktur AESLink dienen und stellt die Referenz für den Entwicklungsstand der Flugsteuerung vor der vorliegenden Arbeit dar.

Die restlichen vier teilen sich in die zwei Gruppen *Allocation Table based* und *Control Effectiveness Matrix based*, welche nachfolgend erläutert werden.

7.3.1. Zuordnungstabellen-basiert

Die beiden Allokatoren *Standard* und *PriorityTable* führen die Zuordnungen anhand von vordefinierten Tabellen durch. Die Tabellen sind jeweils in der ACCSC enthalten und werden daher extern bestimmt.

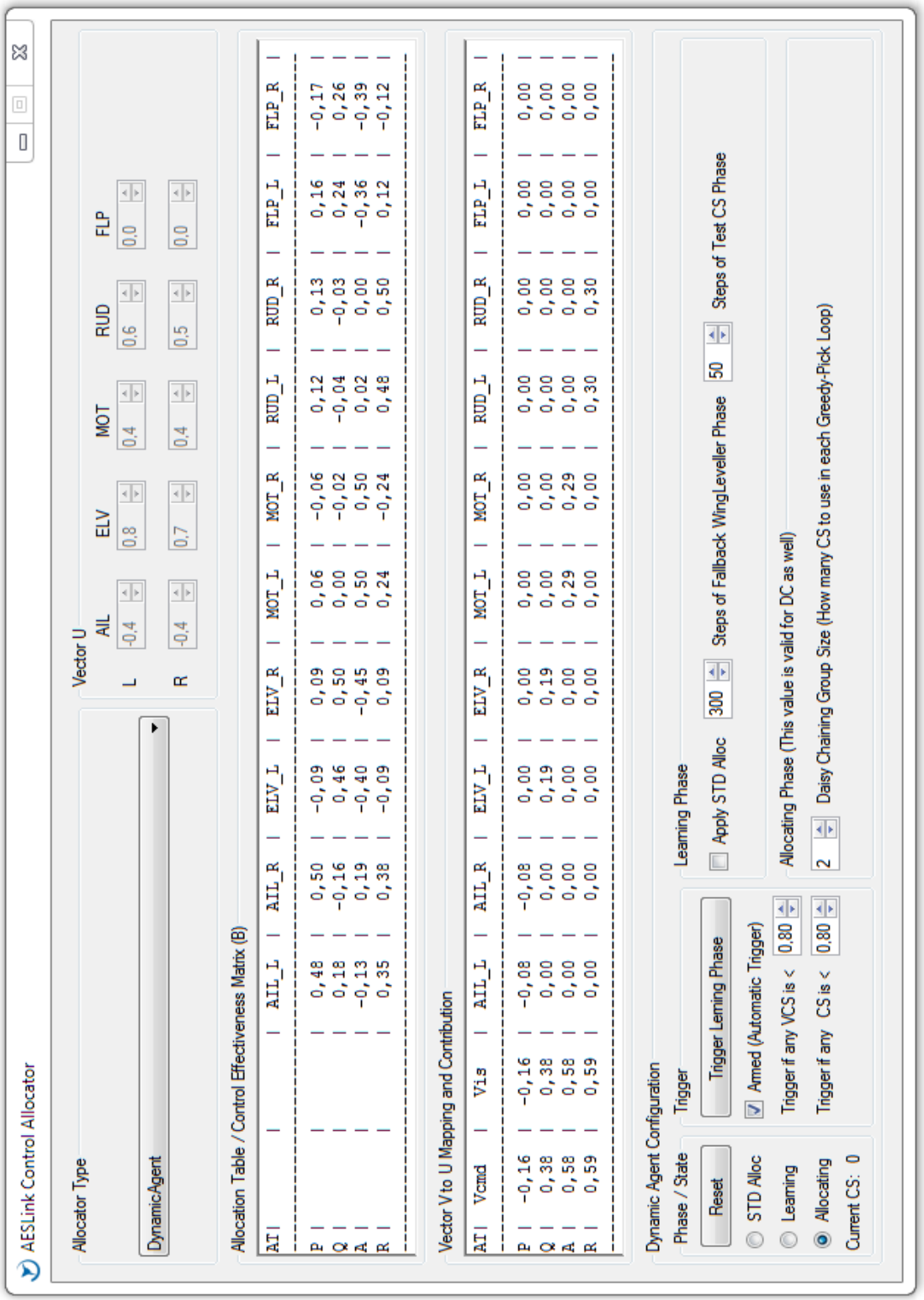

Abbildung 7.2.: Benutzeroberfläche des Control Allocators

Aircraft Control Surfaces Configuration

Die Datenstruktur ACCSC ist wie folgt aufgebaut:

```
1 // List of all Control Surfaces of this Aircraft
2 IList<ControlSurfaceConfiguration>  controlSurfaces;
3 // List of all DOFs that the algorithms shall control
4 IList<DegreeOfFreedomConfiguration> degreesOfFreedom;
5 // Parameters for the Dynamic Agent
6 DynamicAgentConfiguration           dynamicAgentConfiguration;
```

Listing 7.1: Datenstruktur *Aircraft Control Surfaces Configuration*

Die Liste *controlSurfaces* enthält alle vordefinierten Parameter aller Steuerflächen, die von den Algorithmen verwendet werden sollen. Jede einzelne enthält wiederum allgemeine Informationen, wie etwa die Bezeichnung oder deren Indices in den Vektoren u und i. Weiterhin sind für jede Steuerfläche Parameter für die Allokatoren vorhanden, wie etwa die Zugehörigkeit zu den Freiheitsgraden für die Tabellen-basierten Allokatoren oder die Werte der Effektivitätsmatrix B.

In der Liste *degreesOfFreedom* sind Informationen über den Freiheitsgrad hinterlegt, wie etwa ein Namensstring, der Index im Vektor v sowie der Index im Vektor x. In der statischen Struktur *AESL_Plane_CA* sind die vier Freiheitsgrade Rollen, Nicken, Gieren und Beschleunigung mit den Bezeichnungen P, Q, R und A hinterlegt. Wenn nachfolgend der Begriff Parametrierung verwendet wird, ist damit die Einstellung der Datenstruktur ACCSC gemeint.

Standard Allokator

Der Standard Allokator stellt eine statische Steuerflächenzuordnung dar und realisiert somit keine adaptive Flugsteuerung. Es wird daher auch nicht der Integritätszustand des Fault Detectors einbezogen. Dieser Typ soll als Referenz dienen, gegen welche die adaptiven Allokatoren evaluiert werden. In seiner Funktionalität bietet er die gleichen Möglichkeiten wie der statische Allokator, jedoch kann dieser durch die ACCSC parametriert werden, falls ein anderer Träger verwendet wird.

Ein Ausschnitt der in der statischen ACCSC hinterlegten *Allocation Table* ist in Tabelle 7.1 dargestellt. Eine Zelle mit $+$ bedeutet, dass wenn die zugehörige Komponente des Vektors v positiv ist, ebenso eine positive Ansteuerung der Komponente des Vektors u erfolgen soll. Bei einem $-$ wird die Ansteuerung negiert. Ein x stellt dar, dass keine Zuordnung zwischen Freiheitsgrad und Steuerfläche besteht.

AT	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	+++++	+++++	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx
Q	xxxxx	xxxxx	+++++	+++++	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx
A	xxxxx	xxxxx	xxxxx	xxxxx	+++++	+++++	xxxxx	xxxxx	xxxxx	xxxxx
R	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	+++++	+++++	xxxxx	xxxxx

Tabelle 7.1.: Allocation Table des Standard Allokators

PriorityTable Allokator

Der *PriorityTable* Allokator stellt eine Erweiterung des Standard Allokators dar. Dieser realisiert eine adaptive Flugsteuerung, da er den Integritätszustand der Steuerflächen und Freiheitsgrade einbezieht. Der Allokationsalgorithmus entscheidet anhand der in der ACCSC hinterlegten Allocation Table, welcher Freiheitsgrad durch welche Steuerfläche bei welchen Integritätszuständen umgesetzt werden soll.

AT	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	+ 0 +	+ 0 +	- 1 -	+ 1 +	xxxxx	xxxxx	xxxxx	xxxxx	+ 2 +	- 2 -
Q	+ 1 +	- 1 -	+ 0 +	+ 0 +	xxxxx	xxxxx	xxxxx	xxxxx	+ 2 +	+ 2 +
A	xxxxx	xxxxx	- 1 -	- 1 -	+ 0 +	+ 0 +	xxxxx	xxxxx	xxxxx	xxxxx
R	+ 2 +	+ 2 +	xxxxx	xxxxx	+ 1 +	- 1 -	+ 0 +	+ 0 +	xxxxx	xxxxx

Tabelle 7.2.: Allocation Table des Priority Table Allokators

Eine beispielhafte Allocation Table ist in Tabelle 7.2 dargestellt. Die hinterlegten Daten geben vor, dass für den Freiheitsgrad Rollen (*P*) die beiden Querruder (*AIL_L* und *_R*) primär, die Elevator sekundär, usw. verwendet werden sollen. Die Querruder sind für das Nicken (*Q*) die sekundären und für das Gieren (*R*) die tertiären Steuerflächen. Die Prioritäten werden durch jeweilige Grenzwerte der Integritätszustände vorgegeben. Die nachfolgende Datenstruktur ist für jede Steuerfläche vorhanden. Die Elemente der Listen sind dabei die Zugehörigkeiten zu den Freiheitsgraden. Die Anzahl der Einträge geben die Anzahl der Freiheitsgrade vor, für welche diese Steuerfläche verwendet werden soll. Die **thresholds* geben die Grenzwerte der Integritätszustände dieser Zuordnung an. Durch sie werden somit die Prioritäten kodiert.

```
1 IList<int>    indexOfVectorV;         // This entries index of v
2 IList<int>    priorityForVectorV;     // For easy printing
3 IList<CS_Dir> directionIfVisPositive; // Positive or Negative
4 IList<float>  upperThresholdForUsage; // Use *me* for v, if i < x
```

```
5 IList<float>  lowerThresholdForUsage; // Use *me* for v, if i > x
```

Listing 7.2: Parameter eines Eintrags in der Priority Table

Der *PriorityTable* Allokator stellt somit eine adaptive Steuerflächenzuordnung dar. Jedoch werden die jeweiligen Effektivitäten der Steuerflächen für die Freiheitsgrade und die verbliebenen Integritätszustände nicht berücksichtigt. Daher ist die Flugsteuerung zwar adaptiv und stellt eine gewisse Fehlertoleranz gegenüber Ausfällen dar, jedoch ist die Zuordnung nicht dynamisch, da sie komplett statisch vorgegeben wird.

7.3.2. Steuermatrix-basiert

Die Allokatoren *DaisyChaining* und *DynamicAgent* führen die Zuordnungen anhand der Steuermatrix B durch. Die Koeffizienten geben die Effektivitäten der Steuerflächen für den jeweiligen Freiheitsgrad an. Hieraus muss ein geeignetes u gefunden werden, welches das gewünschte v am Träger umsetzt. Ist die Matrix B quadratisch, kann die Inverse von B verwendet werden, um u zu berechnen: $u = inv(B) * v$ [Durham u. a. (2017)]. Jedoch ist dies typischerweise nicht der Fall, da für gewöhnlich mehr Steuerflächen (Spalten) als Freiheitsgrade (Zeilen) vorhanden sind. In diesem Fall muss das *Control Allocation Problem* (siehe Abschnitt 3.2.5) gelöst werden, was auch die Kernaufgabe des Allokators darstellt.

Es existieren numerische Verfahren um die *Pseudoinverse* zu bilden. Für eine gegebene Matrix B existieren eine Vielzahl an unterschiedlichen Pseudoinversen, wodurch sich eine Vielzahl an unterschiedlichen Lösungen des CAP ergibt. Das CAP wird zwar mathematisch gelöst, jedoch nicht physikalisch. Das Verfahren verwendet keine Informationen über die physikalischen Gegebenheiten des Systems, was unweigerlich zu Unstetigkeiten in der Ansteuerung führt [Frost u. a. (2010)]. Für die Invertierung und die darauffolgende Allokation müssen Randbedingungen definiert werden, damit die physikalischen Gegebenheiten berücksichtigt werden. Weiterhin muss eine Kostenfunktion bestimmt werden, nach welcher Strategie allokatiert werden soll. Diese können etwa sein, dass so viele oder so wenige Steuerflächen wie möglich verwendet werden soll, dass möglichst geringe Drehgeschwindigkeiten der Aktuatoren vorkommen oder etwa, dass bestimmte Aktuatoren für bestimmte Freiheitsgrade präferiert werden sollen.

Das CAP kann optimaler gelöst werden, je mehr Informationen über das zu steuernde System bekannt sind, bzw. vorgegeben werden. Das Ziel der vorliegenden Masterthesis ist es, eine gute Lösung mit möglichst wenigen Informationen und Randbedingungen zu erreichen. Das CAP wird durch die Implementierung eines einzelnen Algorithmus gelöst. Dieser verzichtet auf komplexe Randbedingungen, da diese nicht zur Verfügung stehen, bzw. die Aufbereitung zu

komplex gewesen wäre. Es findet somit zwar keineswegs eine *optimale Steuerflächenzuordnung* (siehe Abschnitt 3.2.5) statt, jedoch wird dennoch versucht, diese zu erreichen.

Der Allokationsalgorithmus ist für beide Steuermatrix-basierten Allokatoren der gleiche. Beim Daisy Chaining werden die Koeffizienten vorgegeben, beim dynamischen Agenten werden diese zur Laufzeit angelernt. Zunächst wird der Algorithmus anhand des Daisy Chainings erläutert und auf die Probleme eingegangen, die durch die Abstriche bei der Vorgabe von Informationen aufkommen. Daraufhin werden die Unterschiede und Besonderheiten für das Anlernen der Koeffizienten beim dynamischen Agenten beschrieben.

Daisy Chaining

Für das Daisy Chaining wird die Steuermatrix statisch in der ACCSC kodiert. Die Datenstruktur ist eine leicht abgewandelte Form der Datenstruktur der PriorityTable. Statt vorkonfigurierter Prioritäten für die einzelnen Freiheitsgrade werden die Effektivitäten vorgegeben.

```
1 IList<int>    indexOfVectorV;         // This entries index of v
2 IList<CS_Dir> directionIfVisPositive; // Positive or Negative
3 IList<float>  effectivenessForV;      // Coeff. for this CS and DOF
```

Listing 7.3: Definition eines Koeffizienten der Effektivitätsmatrix für das Daisy Chaining

Ist ein Freiheitsgrad nicht in der Liste, gilt für diesen automatisch die Effektivität 0. Aus diesen Informationen wird die Steuermatrix gebildet. Eine beispielhafte Steuermatrix ist in Tabelle 7.3 dargestellt.

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,50	0,50	-0,15	0,15	0,00	0,00	0,00	0,00	0,10	-0,10
Q	0,20	-0,20	0,50	0,50	0,00	0,00	0,00	0,00	0,10	0,10
A	0,00	0,00	-0,20	-0,20	0,50	0,50	0,00	0,00	0,00	0,00
R	0,10	0,10	0,00	0,00	0,25	-0,25	0,50	0,50	0,00	0,00

Tabelle 7.3.: Steuermatrix des Daisy Chaining Allokators

Interpretation der Koeffizienten

Die Aufgabe des Algorithmus ist es, alle Komponenten des Vektors v durch die zur Verfügung stehenden Komponenten des Vektors u zu erfüllen. Die erste Problemstellung ergibt sich beim Festlegen, bzw. bei der Allokation, insofern, dass die Koeffizienten korrekt interpretiert werden

müssen. Durch die alleinige Darstellung von einzelnen Zahlenwerten kann nicht komplett abgeleitet werden, wie u angesteuert werden muss, um v zu erfüllen. Anders ausgedrückt ergibt sich die Frage danach, wie viel Steuerleistung nötig ist, um die vom Flugregler gewünschte Zustandsänderung umzusetzen. Dies ist ein Problem, welches durch die strickte Trennung von Flugregler und Flugsteuerung sowie den Verzicht der Vorgabe von Systemwissen aufkommt.

Die nötige Steuerleistung kann durch einen einzelnen Wert nicht vollständig kodiert werden. Was ist die Aussage von B_{uv}, wie ist diese Effektivität zu interpretieren? Da es sich bei v und u um normierte Werte handelt und durch die fehlende Einbeziehung der Intention des Flugreglers und der Aktuatordynamiken, kann diese Frage nicht eindeutig beantwortet werden. Folgendes Beispiel soll das Problem verdeutlichen. Der Flugregler kommandiert ein v_{roll} von 0.5 (normiert auf 1). Es ist keine Information vorhanden, was dieser Wert 0.5 aussagen soll und wie er zu interpretieren ist. So kann er u. A. auf die folgenden Arten interpretiert werden:

- Verwende alle Aktuatoren dahingehend, dass das Flugzeug die Hälfte seiner maximal möglichen Rollrate erzeugt
- Verwende alle Aktuatoren dahingehend, dass die Hälfte einer voreingestellte Rollrate erzeugt wird
- Verwende die für die Rollrate präferierten Aktuatoren und steuere diese zur Hälfte an
- Nehme an, dass es x für das Rollen präferierte Steuerflächen gibt. Wenn diese x zu 0.5 angesteuert werden, ist das v_{roll} von 0.5 erfüllt

Wie bereits beschrieben, führt die fehlende Integration über die Intention des Flugreglers zu diesen Interpretationsproblemen. Das Fehlen des Einbeziehens der Aktuatordynamiken kann dazu führen, dass der Flugregler unnötig aggressiv oder defensiv regelt. Für das Daisy Chaining sind die Werte dahingehend kodiert, dass sie das nachfolgend beschriebene Verhalten widerspiegeln. Das gegebene Flugzeug besitzt für jeden Freiheitsgrad zwei dafür ausgelegte Aktuatoren. Wird eine statische Zuordnung verwendet, wird jeder für diesen Freiheitsgrad ausgelegte Aktuator mit diesem Wert angesteuert. Somit ergibt sich, dass ein Querruder 50% der Rollleistung erbringt. Im nachfolgenden Abschnitt wird die resultierende Kodierung deutlicher beschrieben.

Die in Tabelle 7.3 dargestellte Steuermatrix zeigt alle Koeffizienten. Die Werte der nicht-primären Zuordnungen wurden frei geschätzt, so dass etwa die Höhenruder auf das Rollmoment jeweils die $30\%ige$ Wirkung der Querruder erzeugen können. Somit basieren die Koeffizienten auf vordefiniertem Expertenwissen.

Allokationsalgorithmus

Die beschriebene Kodierung der Koeffizienten hat unmittelbaren Einfluss auf den Algorithmus. Dieser muss die gewünschte Steuerleistung v erzeugen. Die Berechnung erfolgt so, dass die Summe aus der Multiplikation der einzelnen Koeffizienten mit v, v ergeben muss. Die nachfolgende Gleichung und der folgende Pseudocode sollen dies verdeutlichen:

$$v_{is} = \sum_{i=0}^{v_{is} >= v_{cmd}} v_{cmd} * B_{v_i} \tag{7.3.1}$$

```
// B_v_cs is the element of B for this cs and this v
// u_cs contains the control value for this cs
// +/- are not shown, but have to be accounted for
while(v_is < v_cmd && notAllCSsAlreadyUsed)
    cs = pickNextCS(arrayOfAllCSs)
    hasToProvideToV = v_cmd - v_is
    u_cs += hasToProvideToV / B_v_cs
    if u_cs > 1 // cs saturated
        u_cs = 1
    thisCSprovidesToV = (u_cs * B_v_cs)
    v_is = v_is + thisCSprovidesToV
```

Listing 7.4: Skizze des Allokationsalgorithmus

Die Schleife wird so lange durchlaufen, bis das gewünschte v_{cmd} erfüllt wird ($v_{is} >= v_{cmd}$) oder durch alle Steuerflächen iteriert wurde. Es muss entschieden werden, in welcher Reihenfolge die Steuerflächen mit $pickNextCS()$ auszuwählen sind.

So könnte man einfach der Reihe nach durch die Liste der Steuerflächen iterieren. Aus mathematischer Sicht wird auf diese Weise das gewünschte v durch die vorhanden Steuerflächen erzeugt. Aus physikalischer Sicht ist dies jedoch nicht sinnvoll, da es so immerwährend zu einer unpassenden Zuordnung kommen würde. So etwa, dass immer zuerst die Querruder[1] verwendet werden würden, um ein mögliches Nickmoment zu erzeugen. Sie würden nicht selten bis zum Anschlag ausgefahren werden, ohne dass die Höhenruder überhaupt ausgeschlagen werden würden. Daher ist die Auswahl nach Reihenfolge nicht geeignet. Es muss eine Kostenfunktion gefunden werden, nach der die Auswahl erfolgt.

Als Kostenfunktion wird ein *Greedy-Pick* verwendet. Bei diesem werden zuerst die Steuerflächen mit dem höchsten Koeffizienten ausgewählt. Dies sorgt dafür, dass möglichst wenige

[1] Die Reihenfolge der Steuerflächen ist beliebig durch die ACCSC konfigurierbar.

und dass typischerweise die für den Freiheitsgrad ausgelegten Steuerflächen priorisiert verwendet werden. Jedoch ergibt sich hier das Problem, dass für kleine v ausschließlich eine der beiden Aktuatoren verwendet wird. Der zweite wird erst dann hinzugezogen, wenn die erste Steuerfläche gesättigt ist.

Um dieses Problem zu lösen, musste eine zweite Kostenfunktion definiert werden. Diese verletzt die Anforderung, dass der Algorithmus keine Informationen über den Träger besitzen soll. Die bisherigen Erläuterungen beschreiben einen Algorithmus, der lediglich eine Liste von Freiheitsgraden, eine Liste von Stellgrößen und eine Tabelle mit Effektivitätszuordnungen zwischen diesen kennt. Diese Informationen sind nicht spezifisch für ein Flugzeug. Der bisher beschriebene Algorithmus könnte daher jegliches technisches Vehikel steuern.

Die zweite Kostenfunktion besteht darin, dass Gruppen von Steuerflächen definiert werden. So können gekoppelte Beziehungen und ein Gleichlauf (engl. *ganging*) abgebildet werden. Durch diese Gruppierung wurde der Name Daisy Chaining gewählt (siehe Abschnitt 3.2.4). Für den gegebenen Träger werden Gruppen zu je zwei Steuerflächen definiert, da jede Steuerfläche symmetrisch zur X-Achse des Flugzeugs jeweils gespiegelt zu ihrer *"Schwestersteuerfläche"* angebracht ist. Jeweils zwei Steuerflächen erzeugen daher stets die gleiche Wirkung für einen gegebenen Freiheitsgrad.

Die Schleife verändert sich durch die zweite Kostenfunktion dahingehend, dass jeweils zwei Steuerflächen ausgewählt werden. Die erforderliche Steuerleistung wird zur Hälfte unter ihnen aufgeteilt. Somit werden zuerst alle primären Steuerflächen verwendet, daraufhin die sekundären, usw.

Um den Allokator adaptiv zu gestalten, müssen noch die Integritätszustände einbezogen werden. Diese vom Fault Detector gelieferten Werte geben die verbliebenen Effektivitäten der Steuerflächen an. Der Algorithmus wird dahingehend erweitert, dass zunächst die Koeffizienten der Steuermatrix entsprechend der verbliebenen Effektivitäten angepasst werden. Hierfür wurde eine weitere Annahme über den Integritätszustand und das System Flugzeug definiert, welche den Allokator weiter in seiner Generalität einschränkt. Die Annahme besagt, dass ein Gleichlauf immer noch möglich ist, solange die Aktuatoren nicht gesättigt sind. Wenn z. B. das linke Querruder lediglich 20% verbliebene Effektivität besitzt, muss es 5 mal so stark ausgeschlagen werden, wie das rechte Querruder. Eine eingeschränkte Dynamik des Aktuators wird auf diese Weise nicht berücksichtigt, ausschließlich die Sättigung. Um die eingeschränkte Integrität auszugleichen, wird in einer Gruppe zuerst die Steuerfläche mit der geringsten verbliebenen Effektivität ausgewählt. Diese muss weiterhin die Hälfte der nötigen Steuerleistung aufbringen. Anhand der angepassten Effektivität wird berechnet, wie hoch die Ansteuerung dafür sein muss. Ist die Steuerfläche gesättigt, muss die nächste Steuerfläche in der

Gruppe einen entsprechend höheren prozentualen Anteil an der Gesamtleistung übernehmen. Schaffen alle Steuerflächen der Gruppe es nicht, das gewünschte v zu erfüllen, wird die nächste Gruppe ausgewählt.

Der finale Allokationsalgorithmus wird durch folgenden Pseudocode abgebildet:

```
// len(group) can be defined in ACCSC
while(v_is < v_cmd && notAllGroupsAlreadyUsed)
    group = pickNextGroup(arrayOfAllCSs, arrayOfGroups)
    B_adjusted = adjustB(B, i)
    for j=0:len(group)
        cs = findLowest_B_v_cs_InGroup(group, B_adjusted)
        hasToProvideToVpercent = 1 / (len(group) - j)
        hasToProvideToV = (v_cmd - v_is) * hasToProvideToVpercent
        u_cs += hasToProvideToV / B_adjusted_v_cs
        if u_cs > 1 // cs saturated
           u_cs = 1
        thisCSprovidesToV = (u_cs * B_adjusted_v_cs)
        v_is = v_is + thisCSprovidesToV
```

Listing 7.5: Finaler Allokationsalgorithmus zur adaptiven Flugsteuerung

Die Gruppen teilen sich die benötigte Steuerleistung somit nicht länger gleich auf. Die Aufteilung findet linear anhand der verbliebenen Effektivitäten statt. Hierdurch ist die Steuerflächenzuordnung adaptiv und fehlertolerant gegenüber Ausfällen. Solange die Steuerflächen nicht gesättigt werden, ergeben sich in der Theorie stets die gleichen Handling und Flying Qualities für einen Piloten und den Flugregler.

Live Darstellung der Zuordnung

Zur einfachen Übersicht über die Zustände der Allokatoren wird die Zuordnung von v auf u zur Laufzeit auf der Benutzeroberfläche in der Textbox *Vector V to U Mapping and Contribution* dargestellt. Die Tabellen 7.4, 7.5 und 7.6 zeigen die Zuordnungen beispielhaft für einen Regelschritt, bei welchem auf allen Freiheitsgraden Bewegungen erzeugt werden sollen.

In Tabelle 7.4 sind alle Steuerflächen intakt und es findet keine adaptive Zuordnung statt. Die Spalten V_{cmd} und V_{is} zeigen jeweils die vom Flugregler gewünschte und die vom Allokator erreichte Erfüllung der Ansteuerungen. Die restlichen Spalten geben an, wie viel jede einzelne Steuerfläche zu welchem v beiträgt. Es kann eingesehen werden, dass von jeder Steuerfläche jeweils die Hälfte der gewünschten Steuerleistung geliefert wird.

VU	Vcmd	Vis	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,50	0,50	0,25	0,25	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Q	-0,40	-0,40	0,00	0,00	-0,20	-0,20	0,00	0,00	0,00	0,00	0,00	0,00
A	0,40	0,40	0,00	0,00	0,00	0,00	0,20	0,20	0,00	0,00	0,00	0,00
R	0,60	0,60	0,00	0,00	0,00	0,00	0,00	0,00	0,30	0,30	0,00	0,00

Tabelle 7.4.: Steueranteile bei voller Integrität

VU	Vcmd	Vis	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,50	0,50	0,20	0,30	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Q	-0,40	-0,40	0,00	0,00	-0,20	-0,20	0,00	0,00	0,00	0,00	0,00	0,00
A	0,40	0,40	0,00	0,00	0,00	0,00	0,30	0,10	0,00	0,00	0,00	0,00
R	0,60	0,60	0,00	0,00	0,00	0,00	0,00	0,00	0,30	0,30	0,00	0,00

Tabelle 7.5.: Steueranteile bei leicht eingeschränkter Integrität

In Tabelle 7.5 wurde ein leicht eingeschränkter Integritätszustand i simuliert. Das linke Querruder (*AIL_L*) kann lediglich 42% der erforderlichen Steuerleistung beitragen, der rechte Motor lediglich (*MOT_R*) 25%. Diese Relationen spiegeln sich in der Tabelle wieder. Der Allokator steuert die jeweils anderen Steuerflächen in größerem Ausmaß an, wodurch er das gewünschte v dennoch erreichen kann.

VU	Vcmd	Vis	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,50	0,41	0,25	0,10	0,02	0,05	0,00	0,00	0,00	0,00	0,00	0,00
Q	-0,40	-0,34	-0,10	-0,04	-0,05	-0,15	0,00	0,00	0,00	0,00	0,00	0,00
A	0,40	0,28	0,00	0,00	0,02	0,06	0,05	0,15	0,00	0,00	0,00	0,00
R	0,60	0,47	0,05	0,02	0,00	0,00	0,03	0,08	0,20	0,10	0,00	0,00

Tabelle 7.6.: Steueranteile bei stark eingeschränkter Integrität

In Tabelle 7.6 wurde ein stark eingeschränkter Integritätszustand i simuliert, wodurch eine adaptive Zuordnung stattfindet. Keine der Komponenten aus v kann vollständig erfüllt werden. Hinzukommt, dass sich die Angaben in den Spalten gegenseitig kompensieren. So resultiert die Ansteuerung von z. B. dem rechten Höhenruder (*ELV_R*) zu $0,05 - 0,15 + 0,06 = -0,04$, was einem sehr geringen Ausschlag entspricht. Es erfolgt keine Priorisierung, welcher Freiheitsgrad favorisiert werden sollen.

Durch die stark eingeschränkten Zustände der Steuerflächen können die Handling und Flying Qualities nicht in vollem Maße erfüllt werden, da das Flugzeug an seine verbliebenen Grenzen stößt.

Schwächen

Es wurde gezeigt, dass der vorgestellte Allokator keine beliebig komplexen Ausfallsituation kompensieren kann. Nachfolgend werden einige analysierte Schwächen präsentiert und Situationen aufgezeigt, bei welchen der Allokator selbst an seine Grenzen stößt.

Es werden feste Gruppen definiert werden, welche nacheinander solange zur Ansteuerung ausgewählt und zugeordnet werden, bis das gewünschte v erreicht ist. Daraus ergeben sich einige Probleme. Wo etwa, wenn dem linken Höhenruder lediglich wenige Prozente Effektivität verbleiben und das rechte noch über volle Integrität verfügt. Solange das linke nicht gesättigt ist, kann ein Gleichlauf stattfinden. Bei höheren Werten von v erfolgt eine Sättigung des linken Höhenruders und der Algorithmus verlagert die Steuerleistung auf das rechte, wodurch sich ein größerer Ausschlag für dieses ergibt. In der Theorie wird, solange das rechte nicht auch gesättigt ist, nach wie vor das gewünschte Nickmoment erzeugt. Jedoch wird hierdurch unweigerlich auch ein Rollmoment induziert, welches durch die Querruder ausgeglichen werden muss. Es ist eine weitere Kostenfunktion nötig, welche das beschriebene Problem verhindert. Durch diese würde sich eine *"bessere"* Lösung des Control Allocation Problems ergeben, da eine *"optimalere"* Steuerflächenzuordnung umgesetzt wird.

Weiterhin werden die einzelnen Kopplungen der Freiheitsgrade untereinander nicht berücksichtigt. Wenn etwa die Höhenruder für die Erzeugung eines Rollmoments herangezogen und hiermit beinahe gesättigt werden, bleibt keine Steuerleistung mehr für das Nickmoment übrig, bzw. die Ansteuerungen für das Rollen werden überschrieben. Dies kann dazu führen, dass sich die Ansteuerungen gegenseitig ausschließen, woraus sich ein Deadlock oder ein Schwingen ergeben kann. Auch hier fehlt dem Algorithmus eine Verhinderung des Problems. Gelöst werden könnte dies durch eine weitere Kostenfunktion, welche bestimmte Freiheitsgrade priorisiert.

Eine pauschale Vorgabe der Gruppengröße ist für asymmetrische oder unkonventionelle Träger ungeeignet. Statt einer festen Vorgabe von z. B. zwei eignet es sich hierfür, die Gruppengrößen pro *hinzuschaltbarer* nächster Gruppe einzeln konfigurieren zu können. Daraus findet eine Berücksichtigung dahingehend statt, dass bei unterschiedlichen Gruppengrößen eine unnötige Überansteuerung einer aus flugmechanischer Sicht nicht zur Gruppe gehörende Steuerfläche verhindert wird. Weiterhin würde hierdurch verhindert werden, dass ein für die Gruppen asymmetrisches i zu einem ungewollten einseitigen Ansteuern und damit Schwingen der Steuerflächen führt.

Die Motivation hinter der Umsetzung in dieser Masterthesis ist jedoch genau dieses Expertenwissen über die Gruppengrößen nicht vorgeben zu müssen, und dennoch eine möglichst optimale Steuerflächenzuordnung zu realisieren, welche die Handling und Flying Qualities für einen (Auto-) Piloten auch im eingeschränkten Zustand des Flugzeugs konstant hält. Der Ansatz, lediglich die nominalen Effektivitäten, die geschätzten Verluste dieser sowie die Annahme, dass immer Gruppen von zwei Steuerflächen gekoppelt sind, als einzige feste Vorgaben einzubeziehen, wurde aufgrund der Anforderung an eine hohe Generalität mit möglichst wenigen festen Vorgaben in der Umsetzung dieser Masterthesis vorgezogen.

7.4. Dynamisches Anlernen der Steuermatrix

Zur weiteren Verminderung der benötigten Vorgabe von Expertenwissen in Bezug auf die Beschaffenheit und Eigenschaften der Steuerflächen wurde ein Lernalgorithmus realisiert, welcher die Koeffizienten der Steuermatrix B zur Laufzeit anlernt. Für diesen wurde der in Abschnitt 4.3.3 präsentierte Ansatz umgesetzt.

In Abschnitt 4.3.3 wurde beschrieben, dass die Dauer der Lernphase und die Menge an Testsignalen von der verbliebenen potentiellen und kinetischen Energie abhängt. Diese wird für das gegebene System UAV als sehr gering eingeschätzt. Aus diesem Grund fällt die Systemidentifikation vom Umfang her sehr gering aus. Als verwendetes Testsignal wurde zunächst lediglich der *Sprung* umgesetzt und es wird jede Steuerfläche nur ein mal angesteuert, um die Systemreaktion zu erfassen und die Koeffizienten abzuschätzen.

7.4.1. Phasen

Der Allokator *Dynamic Agent* verwendet einen Zustandsautomaten. Dessen Zustände spiegeln die Phasen wider, in welchem sich der Allokator befindet. Die Begriffe Phase und Zustand sind nachfolgend somit gleichzusetzen. Die aktuelle Phase wird in der Benutzeroberfläche aus Abbildung 7.2 in der Groupbox *Phase / State* dargestellt.

Zunächst befindet sich der Allokator in der Phase *Standard Allocation* (*STD Alloc*). Die Standardallokation kann analog wie beim Daisy Chaining durch die ACCSC vorgegeben werden. Wie in Abschnitt 4.3.2 beschrieben, soll die Lernphase in dem Moment eintreten, wenn vom Fault Detector eine Ausfallsituation, bzw. eingeschränkte Integrität, festgestellt wurde. Die Grenzwerte, wann der Lernvorgang angestoßen wird, können in der Groupbox *Trigger* zur Laufzeit eingestellt werden. Die Werte bestimmen die unteren Grenzen der Integritätszustände. *VCS* steht hierbei für *Virtual Control Signals* und entspricht i_v, *CS* steht für *Control Surface*

und entspricht i_u. Zum einfachen Testen und zur Evaluierung kann der Lernvorgang über den Button *Trigger Learning Phase* händisch angestoßen werden.

7.4.2. Systemidentifikation und Anlernen

Die Phase *Learning* besteht wiederum aus einem eigenen Zustandsautomaten. In der Phase *Fallback WingLeveller* wird die Standardallokation verwendet. Diese soll den Flugzustand eines Horizontalflugs einstellen. Die Länge der Phase kann über die Textbox *Steps of Fallback WingLeveller Phase* als Anzahl von Regelschritten eingestellt werden.

In der Phase *Test CS* wird ein Sprung von 1 auf die erste Steuerfläche gegeben, wobei die restlichen auf Neutralstellung, bzw. neutraler Ansteuerung, bleiben. Für die Länge der Phase wird der Systemzustand aufgezeichnet, wobei die relevanten Freiheitsgrade über die ACCSC vorgegeben werden. Am Ende der Phase wird für jeden der auszuwertenden Freiheitsgrade der Durchschnitt der Komponenten des aufgezeichneten Systemzustands gebildet. Für den Freiheitsgrad Rollen wird z. B. der Durchschnitt der Rollrate berechnet, welcher für die Länge der Phase erzeugt wurde. Die Durchschnittswerte werden zwischengespeichert. Es wird in die Phase Fallback WingLeveller gewechselt. Nach Abschluss dieser wird ein Sprung auf die nächste Steuerfläche gegeben. Diese Schleife wiederholt sich solange, bis alle Steuerflächen einmal angesteuert wurden. Der aktuelle Index, welcher die Nummer der Steuerfläche angibt, wird in der Benutzeroberfläche mit dem Label *Current CS* angezeigt.

Am Ende der Lernphase existiert eine Abschätzung von Auswirkungen der Steuerflächen auf die Freiheitsgrade. Tabelle 7.7 zeigt solch eine nach einer beispielhaft durchgeführten Lernphase.

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	32,89	33,90	-6,23	6,23	4,04	-4,09	8,34	8,72	10,98	-11,11
Q	5,63	-4,85	13,99	15,14	0,21	-0,48	-1,11	-0,73	8,29	8,23
A	-0,36	0,51	-1,09	-1,24	1,36	1,36	0,06	0,00	-1,08	-1,06
R	5,68	6,17	-1,47	1,41	3,80	-3,83	7,69	8,01	1,88	-2,08

Tabelle 7.7.: Aufgezeichnete Systemreaktionen nach der Lernphase

7.4.3. Bilden der Koeffizienten

Im nächsten Schritt müssen aus diesen die Koeffizienten der Steuermatrix B abgeleitet werden. Zunächst werden die Werte aus der angelernten Tabelle pro Zeile nach dem höchsten Wert

der Zeile auf 1 normiert. Aus der in Tabelle 7.7 dargestellten Durchschnittswerte ergibt sich hieraus die in 7.8 dargestellte normierte Tabelle.

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,97	1,00	-0,18	0,18	0,12	-0,12	0,25	0,26	0,32	-0,33
Q	0,37	-0,32	0,92	1,00	0,01	-0,03	-0,07	-0,05	0,54	0,53
A	-0,26	0,37	-0,80	-0,91	1,00	1,00	0,05	0,00	-0,80	-0,78
R	0,71	0,77	-0,18	0,18	0,47	-0,48	0,96	1,00	0,24	-0,26

Tabelle 7.8.: Normierte Steuermatrix

Die resultierenden Werte können nicht direkt als Koeffizienten übernommen werden, da hierbei die oben beschriebenen Interpretationsprobleme auf die gleiche Weise zur Geltung kommen. Anhand dieser Werte kann nicht bestimmt werden, welche Intention mit den gewünschten virtuellen Steuersignalen des Flugreglers verfolgt wird. Daher wird ein weiteres Mal die Annahme angewandt, nach welcher mehrere gekoppelte Steuerflächen zu einer Gruppe gehören. Die Gruppengröße kann analog zum Daisy Chaining über die Textbox *Daisy Chaining Group Size* vorgegeben werden.

Im letzten Schritt wird jeder einzelne Wert durch die eingestellte Gruppengröße geteilt. Die resultierende Steuermatrix ist in Tabelle 7.9 dargestellt. Die Werte drücken die Effektivitäten somit wie folgt aus: *Diese* Steuerfläche hat bei Vollausschlag *diesen* Einfluss auf *diesen* Freiheitsgrad. Es ergeben sich somit analog zum Daisy Chaining Koeffizienten, welche vom Allokationsalgorithmus verwendet werden können. Der Unterschied zum Daisy Chaining liegt darin, dass die Koeffizienten bei diesem statisch durch Expertenwissen vorgegeben werden müssen. Für den vorgestellten Lernalgorithmus kann auf diese Vorgabe verzichtet werden, da die Koeffizienten dynamisch von einem Softwareagenten angelernt werden. Diese Eigenschaft macht die umgesetzte Realisierung im Sinne der Generalität äußerst attraktiv.

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,49	0,50	-0,09	0,09	0,06	-0,06	0,12	0,13	0,16	-0,16
Q	0,19	-0,16	0,46	0,50	0,01	-0,02	-0,04	-0,02	0,27	0,27
A	-0,13	0,19	-0,40	-0,45	0,50	0,50	0,02	0,00	-0,40	-0,39
R	0,35	0,39	-0,09	0,09	0,24	-0,24	0,48	0,50	0,12	-0,13

Tabelle 7.9.: Resultierende Steuermatrix nach der Einteilung in Daisy Chaining Gruppen

7.4.4. Sonderfall Motoren

Bei der Implementierung hat sich gezeigt, dass die beschriebene Methode nicht zur Erkennung des oder der Motoren geeignet ist. Durch die langsame Dynamik der horizontalen Beschleunigung durch die Motoren, werden durch diese lediglich sehr langsame Veränderungen in der Beschleunigung erzeugt. Die restlichen Steuerflächen haben jeweils einen deutlich größeren Einfluss gezeigt. So z. B., wenn für eine Sekunde eines der Höhenruder voll angesteuert wird, erzeugt dieses einen Nickwinkel, welcher das Flugzeug deutlich mehr beschleunigt, bzw. verlangsamt, als ein für eine Sekunde voll angesteuerter Motor. Dies resultiert darin, dass den Höhenrudern eine deutlich größere Effektivität für den Freiheitsgrad Beschleunigung zugesprochen wird, als den Motoren selbst. Durch die niedrig angelernten Koeffizienten werden die Motoren somit vom Allokator mit der geringsten Priorität behandelt. Praktisch und mathematisch ist dies auch korrekt, jedoch für das System Flugzeug inakzeptabel. Schließlich sind die Motoren die einzigen Aktuatoren, welche dem Flugzustand Energie zufügen können. Ein Auslassen der Motoren durch den Allokator führt je nach verbliebener Energie unweigerlich zum Absturz. Aufgrund des Systems Flugzeug musste dem Lernalgorithmus daher ein Sonderfall für die Motoren implementiert werden. Die Idee und die Umsetzung für diesen ist nachfolgend vorgestellt.

Es gelten zunächst folgende Annahmen, welche vom zu steuernden System Flugzeug abgeleitet wurden:

- Es existiert ein Satz von Aktuatoren, welche dem System kinetische Energie hinzufügen (Motoren)
- Beim System ist eine direkte Wechselwirkung zwischen kinetischer und potenzieller Energie vorhanden
- Somit hat die Zunahme von potenzieller Energie die sofortige Abnahme von kinetischer Energie zur Folge

Die Annahmen drücken aus, dass durch die Erzeugung eines positiven Nickwinkels an Fluggeschwindigkeit verloren wird.

Zur Erkennung der Motoren wurde der Lernalgorithmus um einen Schleifendurchlauf erweitert. In diesem werden zunächst die Steuerflächen gesucht und ausgewählt, welche den größten Einfluss auf den Freiheitsgrad Nicken ausüben. Durch die Tatsache, dass die Motoren eine deutlich geringere Dynamik als die restlichen Steuerflächen erzeugen, kann ausgeschlossen werden, dass bei dieser Suche die Motoren ausgewählt werden. Stattdessen kann davon ausgegangen werden, dass die Höhenruder oder äquivalente Steuerflächen ausgewählten werden. Auf diese wird für die angegebene Dauer der Test CS Phase ein Sprung gegeben. Für die

restlichen Steuerflächen wird die Standardallokation verwendet. Die Ausgabe dieser wird für die Dauer der Phase aufgezeichnet. Der Flugregler, welcher versucht einen Horizontalflug mit gleichbleibender Geschwindigkeit zu regeln, wird zur Einhaltung einer konstanten Fluggeschwindigkeit eine volle Ansteuerung der Motoren kommandieren. Am Ende der Test CS Phase wird die aufgezeichnete Standardallokation dahingehend durchsucht, welche der Aktuatoren mit den größten Werten angesteuert wurden, wobei die zuvor ausgewählten Höhenruder exkludiert werden. Für die gefundenen Aktuatoren wird angenommen, dass es sich bei diesen um die Motoren handelt. Die Koeffizienten dieser Aktuatoren werden für den Freiheitsgrad Beschleunigung auf 1, bzw. $1/len(group)$ gesetzt.

Die Motoren werden durch diese Anpassung des Lernalgorithmus für den in Abschnitt 5.4.1 vorgestellten Träger zuverlässig erkannt. Jedoch wird durch die Erweiterung und die Annahmen wiederum mehr Expertenwissen vom Lernalgorithmus verlangt, wodurch dieser an Generalität eingebüßt. Eine direkte Übertragung auf andere Vehikel ist somit nicht länger ohne eine Anpassung möglich.

7.5. Theoretische Analyse

Abschließend folgt für die realisierten Allokatoren eine Kurzzusammenfassung. Es wird die zur Verwendung nötige Vorgaben von Expertenwissen beschrieben und es erfolgt eine Einordnung nach dem Grad ihrer Adaptivität.

Für den *Standard Allocator* muss eine einfache Allokationstabelle vorgegeben werden. Er ist nicht adaptiv und verwendet keine Informationen über den Integritätszustand der Steuerflächen. Daher bietet er keinerlei Fehlertoleranz. Der Allokator ist **nicht adaptiv**.

Für den *Priority Table Allocator* wird eine komplexe Allokationstabelle vorgegeben. In dieser ist Expertenwissen über die Prioritäten eingebettet. Es ist dabei vorzugeben, in welchen Grenzen des Integrationszustands welche Steuerfläche für welchen Freiheitsgrad genutzt wird. Es wird dabei auf den Integrationszustand der Freiheitsgrade, nicht aber den der Steuerflächen, zurückgegriffen. Daher muss lediglich dieser verfügbar sein. Die Ansteuerung der Steuerflächen erfolgt linear zu der Ansteuerung der Freiheitsgrade. Der Allokator kann somit nicht einschätzen, ob eine nicht-primäre Gruppe an Steuerflächen eine höhere Ansteuerung zur Erfüllung der Steuerleistung benötigt. Der Allokator ist damit adaptiv und tolerant gegen Ausfälle, jedoch ist die Adaptivität auf eine fixe Ansteuerung und feste Grenzen beschränkt. Der Allokator ist **statisch adaptiv**.

Für den *Daisy Chaining Allocator* müssen die Koeffizienten der Effektivitäten zwischen Freiheitsgrad und Steuerflächen sowie die Gruppengröße für gekoppelte Steuerflächen als

Expertenwissen vorgegeben werden. Unter Einbeziehung der Integritätszustände der Steuerflächen wird eine passende Ansteuerung aus allen vorhandenen Steuerflächen gesucht, welche zur gewünschten Änderung des Freiheitsgrades führt. Es wird dabei stets eine passende Ansteuerung gefunden, solange die Integritäten der Steuerflächen dies noch zulassen. Der Allokator ist damit adaptiv und fehlertolerant. Die Adaptivität weist einen hohen Grad auf, da stets alle Aktuatoren für die Ansteuerung einbezogen werden. Für die Berechnung müssen lediglich die Effektivitäten und die Gruppengrößen vorgegeben werden, wodurch der Allokator eine hohe Generalität durch wenige Vorgaben von Expertenwissen aufweist. Der Allokator ist **variabel adaptiv**.

Der *Dynamic Agent Allocator* erweitert das Daisy Chaining dahingehend, dass die Effektivitäten nicht länger als Expertenwissen vorgegeben werden müssen. Stattdessen werden diese zur Laufzeit angelernt. Durch den Sonderfall der Erkennung der Motoren muss ein Teil der Generalität eingebüßt werden. Jedoch kann der präsentierte Lernalgorithmus durch Einschätzung des Autors auch für andere Klassen von Vehikeln verwendet werden, wodurch wieder an Generalität gewonnen wird. Der Allokator ist **dynamisch adaptiv**. Hiermit erfüllt er die Anforderung nach dem Verzicht auf die Vorgabe von Expertenwissen weitestgehend.

8. Evaluierung

Im vorliegenden Kapitel wird die Realisierung des Ansatzes evaluiert. Zunächst sei jedoch darauf hingewiesen, dass das Ziel hinter der Durchführung in dieser Masterthesis nicht nur darin bestand, eine adaptive Flugsteuerung zu entwerfen, sondern gleichzeitig auch eine Arbeitsumgebung mitzuliefern, mit welcher sich die Ansätze leicht auf andere Träger übertragen lassen. Die realisierte Arbeitsumgebung bietet dem Anwender gegenüber durch die Auswahl der Erzeugung an Trainingsdaten, der Auswahl der Netze und *NetTasks* sowie die Parametrierung der Steuerflächenzuordnung eine äußerst große Vielfalt an unterschiedlichen Varianten zur Übertragung auf den eigenen Träger. Aus diesem Grund kann hier lediglich ein Bruchteil der möglichen Kombinationen und Permutationen präsentiert werden.

Zunächst wird die Erzeugung von Trainingsdaten mit dem Pilot Simulator evaluiert. Daraufhin wird die mit dem Pilot Simulator und der in Matlab realisierten Arbeitsumgebung dafür verwendet, um den präsentierten Ansatz der Ausfallerkennung zunächst generell zu evaluieren. Hierbei wird die Performance der zur Verfügung stehenden Netze verglichen. Es folgt eine Auswahl, mit welchen Netzen die Integritätsabschätzung im Fault Detector stattfinden soll. Die Erzeugung und Evaluierung der Restwerte wird präsentiert. Darauffolgend wird das dynamische Anlernen der Koeffizienten der Steuermatrix evaluiert. Im letzten Abschnitt erfolgt ein Vergleich der Allokatoren und es wird die Performance der adaptiven Flugsteuerung durch eine Reallokation im Flugbetrieb präsentiert und diskutiert.

Das für die Evaluierung verwendete Testsystem ist mit einem *Intel Core-I5 4460*, $8GB$ Arbeitsspeicher sowie einer *Nvidia 960 GTX 2GB* ausgestattet. Als Betriebssystem ist *Windows 7 Professional x64* installiert. Alle Simulationen wurden mit einer Frequenz von 50Hz durchgeführt.

8.1. Trainingsdatenerzeugung

Im Folgenden wird die Erzeugung von Trainingsdaten mit dem Pilot Simulator evaluiert. Zunächst ist jedoch positiv hervorzuheben, dass sich dieser auch bei der Entwicklung des Control Allocators und dessen Algorithmen als sehr hilfreiches Werkzeug erwiesen hat. Durch die verbindungslose Kommunikation im AESLink Softwarebus konnte dieser nahtlos während

einer laufenden Simulation entwickelt werden, bei welcher der Pilot Simulator den (Auto-) Piloten emuliert hat. So konnte die Implementierung des Control Allocators unter realen Bedingungen getestet werden, ohne dass die restlichen Anwendungen im Regelkreis händisch gestartet oder konfiguriert werden mussten.

8.1.1. Abdeckung Flight Envelope

Der Signalgenerator des Pilot Simulators soll mit dem erzeugten Flight Envelope eine möglichst hohe Abdeckung der Flugzustände zum realen Betrieb des Flugreglers bieten. Reale mit der FCU gesteuerte Flüge stehen zum aktuellen Zeitpunkt (Mai 2018) jedoch in keiner Weise zur Verfügung. Daher kann keine analytische Analyse, wie etwa ein Vergleich der erzeugten Frequenzen oder Grenzwerte, erfolgen. Aus diesem Grund sei hier auf die eigene Einschätzung aus [Hasberg (2017b)] und Abschnitt 6.1.4 verwiesen.

8.1.2. Reproduzierbarkeit

Der Verbund aus Flugsimulator und Pilot Simulator soll möglichst reproduzierbare Flugzustände erzeugen, um z. B. bei verschiedenen Trainingsdurchläufen oder einer live Simulation, welche vom Pilot Simulator geflogen wird, konsistente Daten zu erhalten. Abbildung 8.1 zeigt den vom Flugsimulator produzierten Rollwinkel aus zwei verschiedenen Flügen. Der Pilot Simulator war dahingehend konfiguriert, dass er die Geschwindigkeit, die Gierrate und den Nickwinkel konstant hält. Erzeugt wurden als Testsignale der Sprung, die Linear- sowie eine S-Funktion. Als Input-Type war *Attitude*, als Input Method *Curve with WingLeveller* ausgewählt. Im Flugsimulator wurde der Einfluss von Wind deaktiviert, um ein möglichst konstantes Verhalten in dessen Berechnung des Flugzustands zu erhalten. Die zwei Durchläufe liefen jeweils 10 Minuten, dargestellt sind jeweils die Minuten 6 bis 8 (ergibt 6000 Regelschritte). Die Abbildung zeigt insgesamt eine hohe Reproduzierbarkeit. Es lässt sich jedoch erkennen, dass die Einstiegspunkte in die Flugkurven leicht versetzt erzeugt wurden. Besonders deutlich ist dies im Bereich zwischen den Regelschritten 4000 und 5000 zu erkennen.

Der Zeitversatz kann durch die unzureichende Echtzeitfähigkeiten des Testsystems und des Flugsimulators zustande kommen. In Abbildung 8.2 ist ein Vergleich der erzeugten Rollwinkel zweier weiterer Simulationen dargestellt. Bei diesen hat der Pilot Simulation auf den Freiheitsgraden Rollen und Nicken jeweils komplett zufällige Steuereingaben erzeugt. Ein zeitlicher Versatz lässt sich bei dieser Messung nicht feststellen. Die Messungen lassen sich stattdessen kaum voneinander unterscheiden.

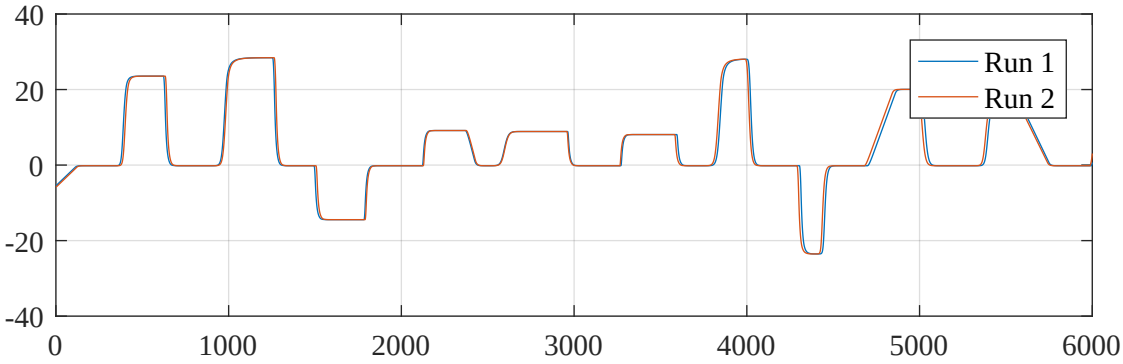

Abbildung 8.1.: Vergleich zweier Simulationsdurchläufe mit erzeugten Sprung-, Linear- sowie S-Funktionen für Verläufe der Rollwinkel

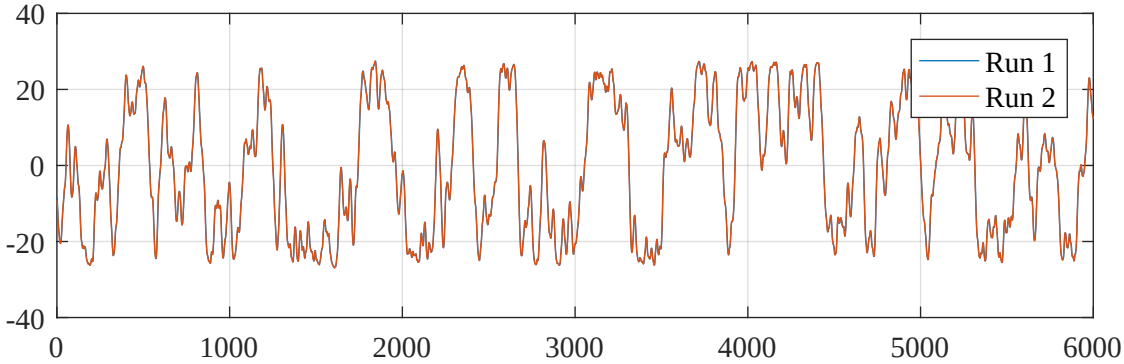

Abbildung 8.2.: Vergleich zweier Simulationsdurchläufe mit komplett zufälligen Verläufen der Rollwinkel

Wie gezeigt, ist der Pilot Simulator dazu in der Lage, den Flugsimulator reproduzierbare Flugzustände erzeugen zu lassen. Es kann durch die fehlende Echtzeitfähigkeit jedoch zu einem zeitlichen Versatz kommen. Anhand der Einschätzungen zur Eignung der erzeugten Signale aus [Hasberg (2017b)] und Abschnitt 6.1.4 sowie der gezeigten Reproduzierbarkeit, erfüllt der Pilot Simulator die an ihn gestellten Anforderungen und ist zur Erzeugung der Trainingsdaten geeignet.

8.2. Ausfallerkennung

In den folgenden Abschnitten werden die Netze zunächst hinsichtlich ihrer Performance im Trainingsprozess und in der Klassifizierung verglichen. Daraufhin folgt eine Analyse der Restwerterzeugung. Abschließend wird die Integritätsabschätzung durch den Fault Detector

evaluiert. Aufgrund der inhaltlichen Komplexität wird hier lediglich ein Integritätsverlust des Freiheitsgrades Rollen und der Querruder analysiert, bzw. es wird gezeigt, wie dieser erkannt werden kann.

8.2.1. Erzeugte Datensätze

Zur Evaluierung der generellen Performance der Nachbildung des Flugverhaltens durch die Netze und der darauffolgenden Integritätsabschätzung wurden insgesamt acht Datensätze erzeugt. Zunächst lassen sich diese in die zwei in Tabelle 8.1 dargestellten Gruppen **A** und **B** einteilen, welche die groben Einstellungen des Pilot Simulators für die jeweiligen Simulationen auflistet.

	Input Type	**Input Method**	**Curve Functions**	**Freiheitsgrade**
A	Autopilot Control	Curve /w WL	Sprung + Linear + S	Roll
B	Autopilot Control	Curve /w WL	Sprung + Linear + S	Roll, Pitch

Tabelle 8.1.: Gruppeneinteilung der Trainingsdatensätze zur Evaluierung der Ausfallerkennung

Die Datensätze wurden so gewählt, da sie eine reale Mission am ehesten widerspiegeln. In diesen ist zu erwarten, dass die Längs- (Rollen) und die Querachse (Nicken) primär zur Veränderung des Flugzustands eingesetzt werden. Die Fluggeschwindigkeit und die Gierrate werden zumeist nicht direkt verwendet, sondern lediglich konstant gehalten.

Die acht erzeugten Datensätze sind in Tabelle 8.2 aufgelistet. Jede der Gruppen wurde jeweils etwas über 1 und 10 Minuten simuliert. Diese vier erzeugten Datensätze wurden reproduziert, wobei jeweils nach einer gewissen Zeit ein Teilausfall des linken Querruders um jeweils 50% in beide Ausschlagsrichtungen durch den Fault Injector getriggert wurde. Für den Freiheitsgrad Rollen verbleibt somit eine Effektivität von insgesamt 75%.

Datensatz	**Dauer (Sek.)**	**Ausfallzeit (Sek.)**	**Ausfalltyp**
A1	70	-	-
A1F	70	40	AIL_L 50%
A10	610	-	-
A10F	610	170	AIL_L 50%
B1	70	-	-
B1F	70	30	AIL_L 50%
B10	610	-	-
B10F	610	170	AIL_L 50%

Tabelle 8.2.: Erzeugte Trainingsdatensätze zur Evaluierung der Ausfallerkennung

8.2.2. NetTasks

Die Auswahl der Netze und deren Parameter, wie etwa die Anzahl der Neuronen und Layer, die Aktivierungsfunktionen, die Netztypen, ob ein einzelnes Netz oder mehrere einzelne Netze verwendet werden, die verwendeten Trainingsparameter, die Auswahl der NetTasks, usw., hängen stark von der Dynamik des verwendeten Trägers ab. Aus diesem Grund wurde versucht, die Arbeitsumgebung in dieser Hinsicht so flexibel wie möglich zu gestalten. Aufgrund des praktisch unendlich großen Zustandsraums kann daher in dieser Arbeit keine umfassende Analyse erfolgen, welche Netze und Parameter generell empfohlen werden könnten.

Für den jeweiligen Datensatz, bzw. die durchzuführende Integritätsabschätzung, müssen zunächst die NetTasks festgelegt werden. Diese geben an, aus welchen Features welche Labels approximiert werden sollen und demnach, welche Struktur der nachzubildenden Übertragungsfunktion unterliegt. Weiterhin ist festzulegen, ob ein einzelnes Netz mit mehreren Ausgangsneuronen alle zu überwachenden Systemzustände approximieren soll oder jeweils eigene Netze mit lediglich einem Ausgangsneuron verwendet werden.

In den Vorarbeiten [Hasberg (2017a)] und [Hasberg (2017b)] wurde jeweils die Annahme aufgestellt, dass sich die zum aktuellen Zeitschritt ergebene Rollrate $x(t)_P$ am besten aus der Rollrate $x(t-1)_P$ und der Steuereingabe des Rollens $v(t-1)_{roll}$ aus dem letzten Zeitschritt abschätzen lässt.

Während der Evaluierung hat sich diese Annahme für den gegebenen Träger und seine Dynamik als korrekt erwiesen. Jedoch ließen sich durch dieses Modell keine Ausfälle erkennen. Der beschriebene NetTask lieferte im fehlerfreien Zustand und bei degradierter Integrität eine sehr gute Approximation der Drehrate. Wird z. B. ein Querruder in seiner Bewegung eingefroren, ergibt sich im Simulator mit relativ hoher Genauigkeit lediglich noch die Hälfte der bisher möglichen Rollrate. Ein Netz hat die sich ergebene Rollrate weiterhin gut approximiert, wodurch der Restwert weiterhin nahe im Bereich des Rauschens lag. Ein Ausfall wurde somit nicht detektiert, da das Netz keine signifikante Differenz in dessen Approximation des Flugverhaltens zeigte. Daraus lässt sich schließen, dass die Rollrate selbst einen zu großen Einfluss auf die Rollrate des nächsten Zeitschritts aufweist und der Einfluss der Ansteuerung v bei der Dynamik des gegebenen Trägers und diesem NetTask vom Netz vernachlässigt wird. Es muss demnach eine andere Konfiguration von Featuren und Labeln gewählt werden, um einen Ausfall detektieren zu können.

Hieraus lässt sich schließen, dass die Übertragungsfunktion, bzw. die Koeffizienten der in Abschnitt 2.2 dargestellten Bewegungsgleichungen, des gegebenen Trägers falsch eingeschätzt wurden. Uhlig et al. verwenden zur Erkennung von Querruderausfällen ein komplexeres Model der Übertragungsfunktion (siehe [Uhlig u. a. (2010)]). Die Features werden aus den

folgenden Werten gebildet: Anstellwinkel, Schiebewinkel, Fluggeschwindigkeit, Rollwinkel, Nickwinkel, Gierwinkel sowie die vier virtuellen Steuersignale. Als Labels werden die Rollrate, der Rollwinkel und die Veränderung des Schiebewinkels approximiert. Auf Basis dieses Modells wurden eigene erstellt, welche nachfolgend beschrieben werden.

Die Werte für den Anstellwinkel und den Schiebewinkel sind im verwendeten Flugsimulator AeroSimRC nicht verfügbar. Die Fluggeschwindigkeit wurde bei den erzeugten Datensätzen konstant geregelt. Es verblieben folgende Komponenten der Vektoren: Rollwinkel, Nickwinkel, Gierwinkel und Ansteuerungen der Aktorik als Eingang und Rollrate sowie Rollwinkel als Ausgang. Daraufhin wurden einige Versuche mit den Datensätzen durchgeführt, um ein gutes Modell zu finden. Für z. B. den Datensatz **A** hat sich gezeigt, dass auf den Nickwinkel, den Gierwinkel und die drei anderen virtuellen Steuersignale im Eingang sowie auf den Rollwinkel im Ausgang verzichtet werden kann.

Für die erzeugten Datensätze haben sich bei der Dynamik des gegebenen Systems in den Flugzuständen der Datensätze die in Gleichung 8.2.1 dargestellten Modelle, bzw. NetTasks, als geeignet erwiesen, um Ausfälle der Querruder zu erkennen.

$$A = \begin{bmatrix} x(t-1)_{roll} \\ v(t-1)_{roll} \end{bmatrix} \rightarrow \begin{bmatrix} x(t)_P \end{bmatrix} \qquad B = \begin{bmatrix} x(t-1)_{roll} \\ x(t-1)_{pitch} \\ x(t-1)_{speed} \\ v(t-1)_{roll} \\ v(t-1)_{pitch} \end{bmatrix} \rightarrow \begin{bmatrix} x(t)_P \end{bmatrix} \tag{8.2.1}$$

Die Frage danach, ob sich diese Modelle für andere Datensätze, Flugzustände, usw. eignen, hängen von der Übertragungsfunktion des Systems ab. Ob sich diese Modelle generell auch für andere Träger eignen, kann vom Autor nicht beantwortet werden. Für diese Fragestellung sei hier auf Literatur aus der Flugmechanik, wie etwa [Brockhaus u. a. (2011)] und [McLean (1990)], verwiesen.

8.2.3. Trainingsprozess und Performance

Es folgt ein Vergleich der von der Arbeitsumgebung zur Verfügung gestellten Netztypen. Trainiert wurde mit den fehlerfreien Datensätzen **A**. Die Analyse der Performance der Ausfallerkennung erfolgte mit den Datensätzen **AxF**.

Für den Vergleich anhand des Datensatzes **A** wurde als NetTask wie oben beschrieben die Auswirkung des virtuellen Steuersignals des Rollens auf die Rollrate gewählt. Eingangsneuro-

nen waren somit: $x(t-1)_{roll}$ und $v(t-1)_{roll}$. Das Ausgangsneuron schätzt die Rollrate des nächsten, bzw. aktuellen, Zeitpunkts $x(t)_P$.

Während der Entwicklung der Arbeitsumgebung hat sich in erster Exploration gezeigt, dass das *Function Fitting Neural Network* (*FitNet*) mit einem Sliding-Window sehr gute Ergebnisse in der Funktionsapproximation des Flugverhaltens erzeugt. Um den Zustandsraum einzuschränken, wurden die restlichen Netztypen entsprechend ähnlich parametriert. Die nachfolgende Evaluierung erfolgt demnach dahingehend, welche Performance die restlichen Netze im Vergleich zum FitNet bieten.

Alle Netze verwenden einen Hidden-Layer bestehend aus 20 Neuronen. Als Aktivierungsfunktionen wurden im Hidden-Layer der Tangens hyperbolicus und im Output-Layer die Linear-Funktion verwendet. Die FF-Typen verwenden ein Sliding-Window von 10 Regelschritten, was einer Zeit von 0.2 Sekunden entspricht. Die TS-Typen verwenden ein Rückkopplungsfenster von einem Regelschritt. Die Daten werden jeweils automatisch von Matlab normiert. Trainiert wurde jeweils mit der CPU und der GPU. Bei der CPU wurde der *Levenberg-Marquardt backpropagation*-Algorithmus verwendet. Bei der GPU die *Scaled conjugate gradient backpropagation*, was bei Matlab eine Vorgabe beim Training mittels Grafikprozessor ist. Das Training wird von Matlab erfolgreich beendet, wenn sich für 6 Epochen kein weiterer Trainingsfortschritt einstellt oder die Fehlerrate 0 wird. Ein Abbruch erfolgt, wenn die maximale Anzahl an Epochen (hier 500) erreicht wurde, der Gradient durch eine *Explosion* zu große Werte annimmt oder das Momentum zu kleine Werte annimmt.

Jeder Netztyp wurde jeweils drei mal per CPU und GPU trainiert. Die Tabelle 8.3 führt die Ergebnisse auf. Dargestellt ist der jeweils beste Trainingsdurchlauf gemessen an der Regressions-Performance, welche von Matlab direkt berechnet wird und das Verhältnis der Werte der Labels und des Netz-Outputs darstellt.

Zunächst wurden alle Netze mit dem einminütigen Datensatz **A1** trainiert. Dabei konnten bereits einige Beobachtungen gemacht werden. Die RBF Netze lieferten in allen Testläufen die exakt gleichen Ergebnisse. Sie unterliegen somit keinem zufälligen Initialisierungsprozess. Sie sind hier weiterhin nicht zu empfehlen, da sie im Matlab Trainingsprozess einen deutlich höheren Speicherverbrauch aufwiesen. Es musste je nach Anzahl der Neuronen und Größe des Sliding-Windows eine bis zu 60GB große Auslagerungsdatei am Testsystem eingerichtet werden, damit der Matlab-Systemprozess nicht abstürzt. Für die NARX Netze konnte kein GPU-Training angewandt werden, da dies ausnahmslos zu Matlab-Exceptions führte. Das GPU-Training der LSTMs wird von Cortexsys nicht unterstützt. Bei den Netzen Fit, TimeDelay, DistDelay und Elman funktionierte das GPU-Training lediglich sporadisch, da es oft zu einer Explosion des Momentums oder des Gradienten kam. Hierdurch mussten einige der

NetzTyp	CPU/GPU	Epochen	Zeit	Performance
		A1		
Fit	CPU	15	00:00	0.000 701
Fit	GPU	60	00:00	2.47
RBF	CPU	500	00:26	207.0
RBF	GPU	500	00:13	236.0
TimeDelay	CPU	19	00:01	0.000 952
TimeDelay	GPU	27	00:27	0.344
DistDelay	CPU	12	00:07	1.19
DistDelay	GPU	130	01:04	13.6
Elman	CPU	500	01:43	0.000 831
Elman	GPU	500	02:25	0.0845
NARX	CPU	33	00:01	0.000 304
NARX	GPU	−1	−:−	−1.0
LSTM	CPU	500	08:13	0.97
LSTM	GPU	−1	−:−	−1.0
		A10		
Fit	CPU	500	01:16	0.000 882
Fit	GPU	441	00:05	0.433
TimeDelay	CPU	500	06:56	0.000 990
TimeDelay	GPU	208	07:56	0.350
DistDelay	CPU	96	10:09	2.28
Elman	CPU	312	10:43	0.001 12
Elman	GPU	305	15:34	1.13
NARX	CPU	500	02:46	0.000 397

Tabelle 8.3.: Ergebnisse der Trainingsprozesse der verschiedenen Netztypen

Durchläufe wiederholt werden, um vergleichbare Resultate zu erlangen. Die RBF und LSTM Netze wurden von der weiteren Evaluierung ausgeschlossen, da diese jeweils eine deutlich schlechtere Performance als die restlichen Netze ablieferten.

Unter Ausschluss der genannten Problemfälle wurde der zehnminütige Datensatz **A10** trainiert. Beim GPU-Training der DistDelay Netze ist der Matlab-Systemprozess wiederholt abgestürzt. Für die TimeDelay und Elman Netze ist das Training per GPU nicht zu empfehlen, da der Trainingsprozess sowohl mehr Zeit benötigt, als auch schlechtere Performancewerte liefert. Beim Fit Netz ergibt sich ein Trade-Off. Das Training wird durch die GPU deutlich beschleunigt, liefert jedoch eine schlechtere Performance. Es steht dabei die Frage im Raum,

ob das Training per GPU in neueren Versionen von Matlab als die verwendete (R2016a x64) besser umgesetzt ist und es zu weniger Problemen kommt[1]

Werden die benötigte Trainingszeit und die Performance der Netze verglichen, stechen das Fit und das NARX Netz hervor. Für beide Trainingsdatensätze lieferten sie gute Resultate in jeweils kurzer Zeit. Für das gegebene Problem der Nachbildung des Flugverhaltens eignen sich diese demnach und werden an dieser Stelle weiterverwendet und außerdem für eine erste Exploration bei der Übertragung auf andere Träger empfohlen.

8.2.4. Restwerterzeugung und Integritätsabschätzung

Die Ausfallerkennung wurde aufgrund der durchgeführten Vergleiche mit Fit und NARX Netzen untersucht. Zunächst wird die Performance der Funktionsapproximation präsentiert.

Hierfür wurde ein NARX Netz per CPU mit dem Datensatz **A1** und dem zugehörigen NetTask trainiert. Abbildung 8.3 zeigt die Approximation der Rollrate für den Datensatz **A1F** im Bereich von Sekunde 35 bis 65. In diesem wurde der Ausschlag des linken Querruders nach 40 Sekunden auf 50% reduziert.

In den Sekunden 35 bis 40 schätzt das Netz die Rollrate mit sehr hoher Genauigkeit ab. Die tatsächliche Funktion und die Approximation können nicht voneinander unterschieden werden. Dies bestätigt den hohen Performancewert beim Training.

Ab Sekunde 40 können mehrere Kurven beobachtet werden, bei welchem das Netz eine deutlich zu hohe Rollrate schätzt. In den Sekunden 43, 46.5 und 52.5 schätzt das Netz weiterhin eine Rollrate von etwa $80°/s$, wobei die tatsächliche Rollrate bei lediglich etwa $60°/s$ liegt. Hier kann am Diagramm bereits abgelesen werden, dass potentiell eine Restintegrität von 75% abgeschätzt werden kann. Die Kurven in Sekunde 42 und 62 können je nach Parametrierung als Rauschen oder als Ausfall gewertet werden.

In den Kurven der Sekunden 48, 51, 54, usw. ist zu erkennen, dass ein Ausfall nur durch eine ausreichend große Ansteuerung der Steuerflächen detektiert werden kann. Befinden sich die Steuerflächen in Neutralstellung, ist eine Abschätzung der verbliebenen Integrität nicht möglich, da diese auf der Beobachtung der Systemdynamik basiert.

Abbildung 8.4 zeigt den Restwert für den genannten Zeitraum. Von den drei in Abschnitt 4.2.1 beschriebenen Verfahren liefern das direkte Fehlerquadrat R_{direct} und die Fehlerquadratsumme $R_{windowed}$ die besten Ergebnisse, um den Restwert zwischen geschätztem und tatsächlichem Zustand zu berechnen. Ein Rauschen, bzw. Impulsspitzen, sind aufgrund der

[1] Für weitere Informationen sei hier auf die folgende Hilfeseite von MathWorks verwiesen: https://de.mathworks.com/help/nnet/ug/neural-networks-with-parallel-and-gpu-computing.html [Stand: 25.05.18]

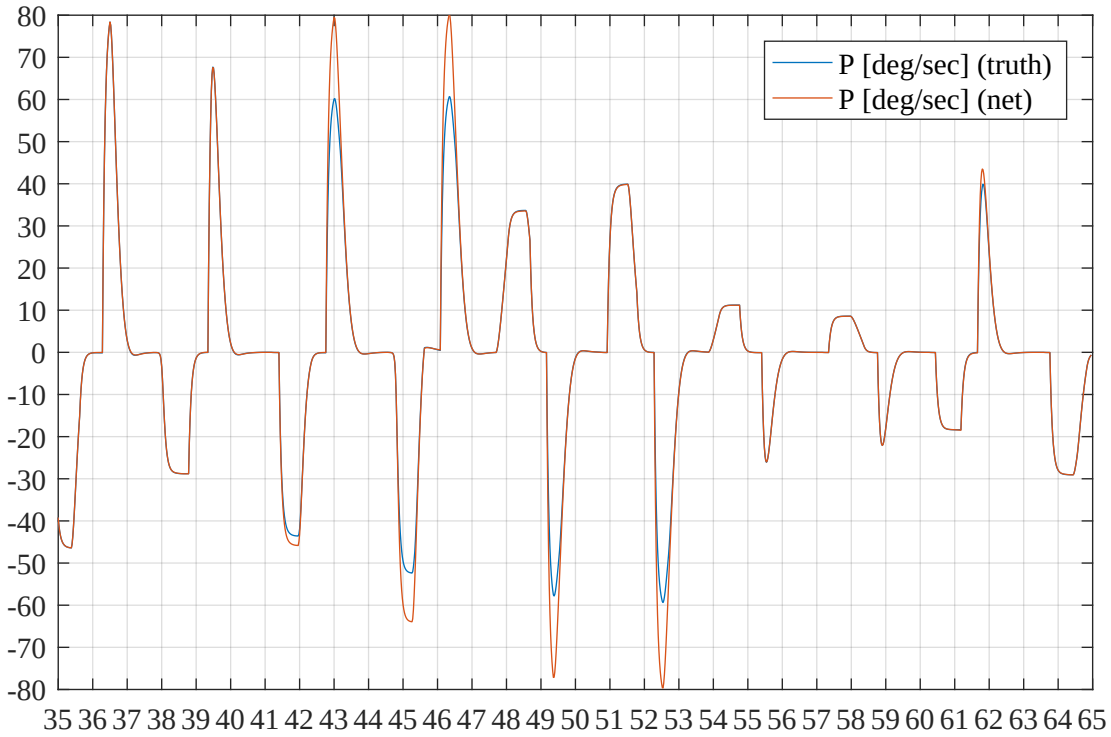

Abbildung 8.3.: Approximation des Datensatzes A1F durch ein NARX Netz

guten Approximation nicht vorhanden. Die Abbildung 8.5 zeigt den Restwert für einen simulierten Flug mit Windeinflüssen. Hier ergeben sich durchgängig Rauschen und Impulsspitzen. Diese müssen durch den Restwertanalysator (siehe Abschnitt 4.2.2) gefiltert werden, so dass es zu keinen False-Positives kommt.

Anhand der Messwerte aus den Abbildungen 8.3, 8.4 und 8.5 können die für den Restwertanalysator benötigten Parameter abgeleitet werden, welche sich für einen gegebenen Träger eignen. Es müssen hierbei Parameter bestimmt werden, welche einen guten Kompromiss aus der Unterdrückung von Rauschen und Impulsspitzen sowie einer zeitlich frühen Erkennung der Ausfälle darstellen.

Für den hier gegebenen Träger wurden für den Wechsel in den Zustand *fault observed* ein Grenzwert des Restwerts von 100 und ein Maximum des Zählers von 5 Regelschritten verwendet. Mit diesen konnte der verbliebene Integritätszustand effektiv abgeschätzt werden.

Es wurden je vier Fit und NARX Netze für die vier fehlerfreien Datensätze trainiert und in einer .NET-DLL eingebettet, welche in den Fault Detector geladen wurde. Daraufhin wurden die erzeugten Datensätze mit dem AESLink Wiedergabe-Tool abgespielt. Die Ergebnisse der Integritätsabschätzung sind in den Tabellen 8.4 und 8.5 dargestellt.

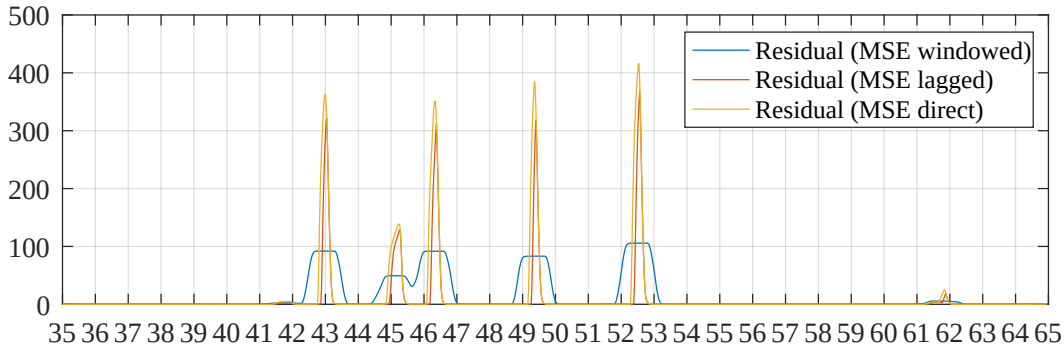

Abbildung 8.4.: Restwerte bei der Approximation des Datensatzes A1F durch ein NARX Netz

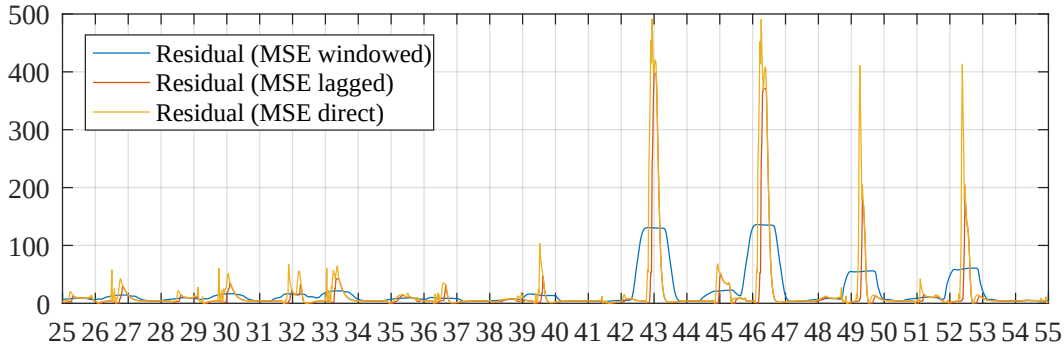

Abbildung 8.5.: Rauschen und Impulsspitzen im Restwert bei einem simulierten Flug mit Windeinflüssen

Tabelle 8.4 zeigt die Resultate für den Datensatz **A**. Die verbliebene Integrität wird von beiden Netztypen gut abgeschätzt. Der präsentierte Ansatz zur Ausfallerkennung, bestehend aus neuronalen Netzen und dem Restwertanalysator, liefert anhand dieser Daten gute Ergebnisse und wird daher als geeignet bewertet.

Tabelle 8.5 zeigt die Resultate für den Datensatz **B**. Hier zeigen sich Schwächen in der Abschätzung. Beim Datensatz **B1F** hat das NARX Netz für die Höhenruder eine degradierte Integrität abgeschätzt, was als False-Positive zu werten ist. Den Datensatz **B10F** konnten beide Netztypen nicht gut approximieren. Eine Analyse des Verhaltens hat gezeigt, dass es zu einer Unstetigkeit bei der Reproduktion des Datensatzes **B10** durch **B10F** kam, wodurch es in **B10F** eine Flugsituation gab, welche in **B10** nicht auftauchte. Dadurch konnte diese nicht approximiert werden, wodurch bereits vor dem Triggern des Ausfalls ein fataler False-Positive von 0% verbliebener Integrität berechnet wurde. Es wird erwartet, dass dies durch einen

Datensatz	Netztyp	$i_{soll_{roll}}$	$i_{ist_{roll}}$
A1	Fit	1.00	1.00
A1F	Fit	0.75	0.74
A10	Fit	1.00	1.00
A10F	Fit	0.75	0.72
A1	NARX	1.00	1.00
A1F	NARX	0.75	0.74
A10	NARX	1.00	1.00
A10F	NARX	0.75	0.74

Tabelle 8.4.: Integritätsabschätzungen für den Datensatz A

Datensatz	Netztyp	$i_{soll_{roll}}$	$i_{ist_{roll}}$	$i_{soll_{pitch}}$	$i_{ist_{pitch}}$
B1	Fit	1.00	1.00	1.00	1.00
B1F	Fit	0.75	0.61	1.00	1.00
B10	Fit	1.00	1.00	1.00	1.00
B10F	Fit	0.75	0.00	1.00	1.00
B1	NARX	1.00	1.00	1.00	1.00
B1F	NARX	0.75	0.68	1.00	0.65
B10	NARX	1.00	1.00	1.00	1.00
B10F	NARX	0.75	0.00	1.00	1.00

Tabelle 8.5.: Integritätsabschätzungen für den Datensatz B

erweiterten NetTasks, wie etwa das Modell von Uhlig et al. [Uhlig u. a. (2010)], hätte verhindert werden können. Jedoch wurde dies hier nicht weiter untersucht.

Aufgrund der präsentierten False-Positives wird an dieser Stelle darauf hingewiesen, dass für den produktiven Einsatz eine ausreichend große Datenbasis zur Überprüfung der Funktionsfähigkeit vorhanden sein muss. Diese muss alle Flugzustände des Flight Envelopes enthalten, um keine False-Positives zu erzeugen. Hier zeigen sich Schwächen des maschinellen Lernens, da eine analytische Verifikation, ob alle Fälle abgedeckt sind, nicht möglich ist.

8.3. Dynamisches Anlernen der Koeffizienten

Für die Evaluierung des Lernalgorithmus wird zunächst die Tabelle 7.9 herangezogen, da diese während einer live-Simulation angelernt wurde. In dieser ist zunächst zu erkennen, dass die Koeffizienten in ihrem Verhältnis zueinander relativ genau erkannt werden und sich die Aktuatorgruppen jeweils voneinander unterscheiden lassen. Werden die Koeffizienten in jeder Reihe durch aufsteigende Prioritäten mit einer Daisy Chaining Gruppengröße von 2 ersetzt,

ergibt sich die in Tabelle 8.6 dargestellte statische Prioritätstabelle. Die Prioritäten der primären Steuerflächen und die symmetrischen Aktuatorgruppen wurden korrekt erkannt.

```
AT|AIL_L|AIL_R|ELV_L|ELV_R|MOT_L|MOT_R|RUD_L|RUD_R|FLP_L|FLP_R|
-------------------------------------------------------------
P |+ 0 +|+ 0 +|- 3 -|+ 3 +|+ 4 +|- 4 -|+ 2 +|+ 2 +|+ 1 +|- 1 -|
Q |+ 2 +|- 2 -|+ 0 +|+ 0 +|+ 4 +|- 4 -|- 3 -|- 3 -|+ 1 +|+ 1 +|
A |- 3 -|+ 3 +|- 1 -|- 1 -|+ 0 +|+ 0 +|+ 4 +|+ 4 +|- 2 -|- 2 -|
R |+ 1 +|+ 1 +|- 4 -|+ 4 +|+ 2 +|- 2 -|+ 0 +|+ 0 +|+ 3 +|- 3 -|
-------------------------------------------------------------
```

Tabelle 8.6.: Aus den dynamisch angelernten Koeffizienten abgeleitete statische Prioritätstabelle

8.3.1. Gleichlauf und Kopplung

Jeweils zwei der Steuerflächen sind durch die symmetrische Position am Flugzeug miteinander gekoppelt. Sie unterliegen damit entweder einem symmetrischen oder einem differentiellen Gleichlauf. Der Lernalgorithmus des Control Allocators regt in der *Test_CS*-Phase eine einseitige positive Ansteuerung der Steuerflächen an. Es kommt aufgrund der Kopplungen der Freiheitsgrade je nach positiver Auslenkungsrichtung zu asymmetrischen Rotationen in allen Achsen. Durch die einseitige Ansteuerung wird die Systemreaktion des Flugzeugs nicht optimal identifiziert, da sich ungewollte Kopplungen in den restlichen Freiheitsgraden ergeben.

Eine bessere Systemidentifikation lässt sich durchführen, wenn die einzelnen Koeffizienten isoliert identifiziert und ungewollte Kopplungen unterdrückt werden. Hierfür müssen für jeden zu messenden Koeffizienten die drei jeweils restlichen Freiheitsgrade stabil gehalten werden. Wenn also z. B. der Effekt des linken Querruders auf das Nicken untersucht werden soll, müssen die restlichen Steuerflächen dafür verwendet werden, um die anderen Freiheitsgrade zu stabilisieren. Durch den bewussten Verzicht auf einprogrammiertes Vorwissen über die Steuerflächenkonfiguration kann und *soll* der Lernalgorithmus dies jedoch zunächst nicht leisten.

8.3.2. Isolierte Identifikation

Nachfolgend wird eine isolierte Analyse der Koeffizienten für das Rollen präsentiert. Es wurde für jede der gekoppelten Gruppen ein Testlauf durchgeführt. In diesen wurde die Steuerflächenzuordnung statisch per Allocation Table durchgeführt. Die Gruppen sind jeweils als primäre Steuerflächen für das Rollen verwendet worden. Der Flugregler hat zunächst einen stabilen

Horizontalflug erzeugt. Daraufhin wurde ein Sprung auf den Sollwert der Kursregelung gegeben. Die zu testenden Steuerflächen führten die Kurve durch ein Rollen aus, die restlichen wurden zur Stabilisierung und Dämpfung der anderen Freiheitsgrade verwendet.

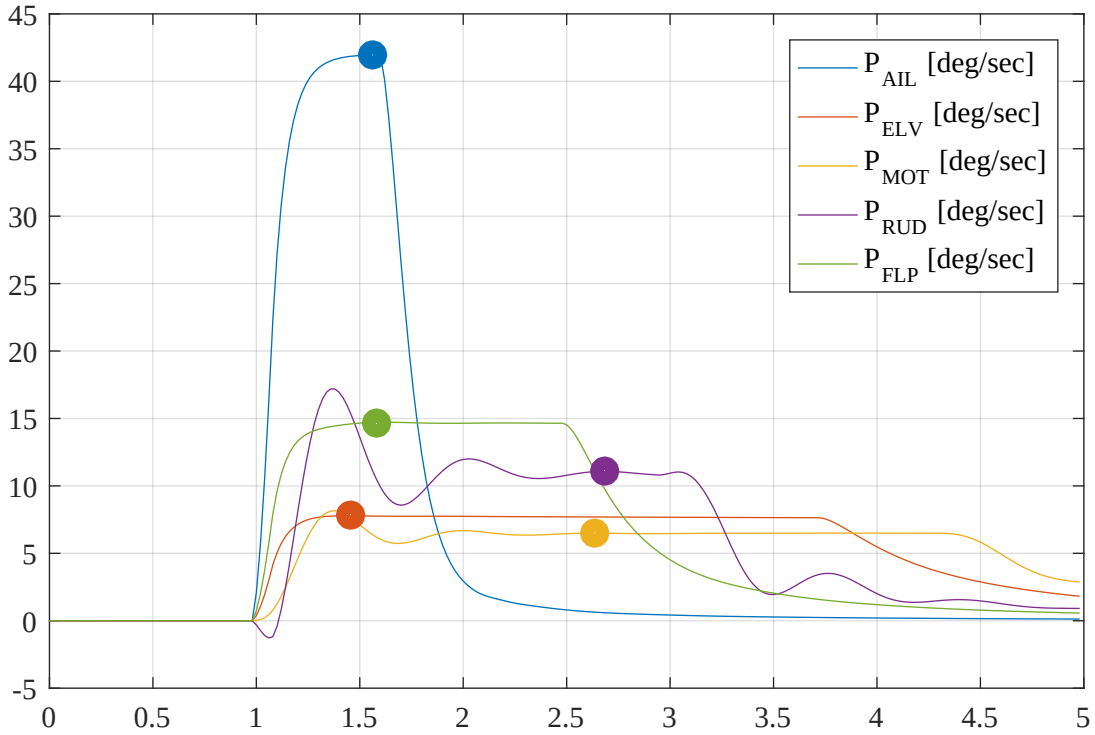

Abbildung 8.6.: Resultierende Rollraten bei gleichlaufender Ansteuerung der Gruppen, wenn Sprung auf Sollwert des Kursreglers gegeben wird

Die jeweils gemessenen Rollraten sind in Abbildung 8.6 dargestellt. Die Querruder liefern die eindeutig höchste Drehrate. Die Motoren zeigen ein leichtes, die Seitenruder ein starkes Überschwingen. Dieses kommt durch ungeeignete Parametrierung des Reglers zustande, da dieser für die Testmessungen nicht angepasst wurde. Dies zeigt bereits auf, dass eine adaptive Flugsteuerung nicht optimal arbeiten kann, wenn ein Steuerflächenausfall vor dem Regler maskiert wird und dieser sein Flight Control Law und seine Parameter nicht anpasst.

Die maximal möglichen Drehraten wurden jeweils an den dargestellten Punkten abgelesen. Für die Motoren wurde ein Zeitpunkt nach Abklingen der Schwingung gewählt. Für die Seitenruder ein mittiger Punkt innerhalb der abklingenden Schwingung.

Die Tabelle 8.7 zeigt die resultierenden Rollraten bei einseitiger Ansteuerung (P_{ea}) und bei Gleichlauf (P_{gl}). P_{gl} stellt die an den in Abbildung 8.6 markierten Punkten gemessenen Drehraten dar. Durch die isolierte Messung stellen diese die tatsächlichen maximal erzeugbaren

Steuerfläche	P_{ea}	P_{gl}	P_{ea}/P_{gl}
AIL_L	32.98	41.95	0.79
AIL_R	33.90	41.95	0.81
ELV_L	6.23	7.78	0.80
ELV_R	6.23	7.78	0.80
MOT_L	4.04	6.49	0.61
MOT_R	4.09	6.49	0.62
RUD_L	8.34	11.08	0.75
RUD_R	8.72	11.08	0.78
FLP_L	10.98	14.71	0.75
FLP_R	11.11	14.71	0.76

Tabelle 8.7.: Verhältnis der Rollraten bei einseitiger und gleichlaufender Ansteuerung

Drehraten der einzelnen Steuerflächen dar. Es sind lediglich die Absolutbeträge der Werte dargestellt, da sich die gemessenen Werte bei umgekehrter Ansteuerung umdrehen.

Die bei einseitiger Ansteuerung gemessenen Werte liegen innerhalb der Gruppen nah beieinander. Die Querruder und die Höhenruder erreichen bei einseitiger Ansteuerung ein Verhältnis von etwa 80% gemessener Drehrate zur gleichlaufenden Ansteuerung. Die Motoren erreichen Durchschnitt etwa 60%, die Landeklappen und die Seitenruder jeweils etwa 75%. Eine quantitative Aussage über die Ergebnisse kann an dieser Stelle nicht erfolgen, da keine Vergleiche oder andere Indikatoren zur Bewertung vorliegen.

8.3.3. Ableiten der Koeffizienten

In den Abschnitten 7.3.2 und 7.4.3 wurde die Problemstellung aufgezeigt, dass eine Interpretation der gemessenen Werte fehlt, um aus diesen die korrekten Koeffizienten abzubilden. Die gemessenen Verhältnisse in Tabelle 8.7 ermöglichen diese Interpretation. Die in Tabelle 7.9 angelernten Koeffizienten müssen mit den in Tabelle 8.7 dargestellten Verhältnissen skaliert werden. So ergeben sich Koeffizienten, durch welche der Allokationsalgorithmus eine optimale Ansteuerung und Verteilung der benötigten Steuerleistung vornehmen kann. Jedoch soll dieser, wie bereits beschrieben, ohne diese Vorgabe arbeiten.

Weiterhin kann anhand der Tabelle 8.7 abgeleitet werden, wie gut die Koeffizienten in Relation zueinander abgeschätzt wurden. Die Unterschiede der jeweils symmetrischen Steuerflächen liegen innerhalb des Messrauschens.

Die Tabelle 8.8 stellt die jeweiligen Verhältnisse der Koeffizienten im Verhältnis zu den Referenzkoeffizienten der Querrudern dar. V_{gl} gibt die Verhältnisse zu den Querrudern bei Gleichlauf, V_{ea} bei einseitiger Ansteuerung das Verhältnis zum jeweils besseren Querruder an.

Steuerfläche	C_{ea}	V_{ea}	C_{gl}	V_{gl}	V_{ea}/V_{gl}
AIL_L	0.4864	0.9729	0.5000	1.0000	0.9729
AIL_R	0.5000	1.0000	0.5000	1.0000	1.0000
ELV_L	0.0919	0.1838	0.0928	0.1856	0.9903
ELV_R	0.0919	0.1838	0.0928	0.1856	0.9903
MOT_L	0.0596	0.1192	0.0774	0.1548	0.7700
MOT_R	0.0603	0.1206	0.0774	0.1548	0.7791
RUD_L	0.1230	0.2460	0.1321	0.2642	0.9311
RUD_R	0.1286	0.2572	0.1321	0.2642	0.9735
FLP_L	0.1619	0.3289	0.1754	0.3508	0.9376
FLP_R	0.1639	0.3277	0.1754	0.3508	0.9342

Tabelle 8.8.: Verhältnis der angelernten zu den tatsächlichen Koeffizienten zum maßgebenden Querruder

Die Werte von V_{ea}/V_{gl} korrelieren mit den Verhältnissen P_{ea}/P_{gl} aus Tabelle 8.7. Diese geben ein Maß dafür, ob der Allokationsalgorithmus, unabhängig von tatsächlich benötigter Steuerleistung, die Steuerflächen untereinander korrekt priorisieren würde. Werden die Koeffizienten C_{gl} in eine Zeile einer Prioritätstabelle umgewandelt, ergeben sich die gleichen Daisy Chaining Gruppen und Prioritäten zu den angelernten aus Tabelle 8.6. Die Prioritäten untereinander wurden somit korrekt angelernt. Es sei angemerkt, dass sich durch die verwendete isolierte Systemidentifikation geeignete Koeffizienten für das statische Daisy Chaining identifizieren und die statischen Prioritätstabellen abgeleitet werden können. Sie stellt damit ein Verfahren dar, um Expertenwissen über das System zu generieren.

8.3.4. Degradierte Integritäten

Zur Evaluierung der Performance im Ausfallzustand wurde ein Flug mit stark degradierten Integritätszuständen simuliert. Folgende verbliebene Ausschlagswinkel, bzw. Ansteuerungen der Motoren, wurden am Flugsimulator getriggert: *AIL_L* 50%, *ELV_R* 25%, *MOT_L* 0%, *RUD_R* 50%, *FLP_L* 75%. Die aus der Lernphase resultierenden Koeffizienten sind in Tabelle 8.9 dargestellt. Tabelle 8.10 zeigt das Verhältnis zu den im fehlerfreien Flug aufgezeichneten Koeffizienten aus Tabelle 7.9. Die mit $*$ markierten Werte wurden aufgrund von zu geringen Basiskoeffizienten nicht gewertet.

Für die nicht beeinträchtigten Steuerflächen ergeben sich zumeist höhere Werte. Die Koeffizienten werden im Control Allocator pro Zeile dem höchsten Wert nach skaliert. Verliert dieser an Integrität, werden für die Skalierung der nachfolgend größere Wert genommen, welcher zuvor z. B. einen Betrag von 95% des höchsten Wertes aufwies.

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	0,26	0,50	-0,09	0,02	0,00	-0,06	0,12	0,07	0,13	-0,16
Q	0,12	-0,11	0,50	0,11	0,04	0,04	0,01	0,03	0,22	0,28
A	-0,11	0,14	-0,45	-0,08	0,50	0,50	-0,01	-0,04	-0,30	-0,41
R	0,21	0,39	-0,08	0,00	-0,01	-0,24	0,50	0,26	0,10	-0,13

Tabelle 8.9.: Angelernte Koeffizienten bei mehrfach degradierten Integritäten

CM	AIL_L	AIL_R	ELV_L	ELV_R	MOT_L	MOT_R	RUD_L	RUD_R	FLP_L	FLP_R
P	53,06	100,00	100,00	22,22	* 0,00	100,00	100,00	46,15	100,00	81,25
Q	42,11	106,25	108,70	18,00	* 0,00	* 0,00	* 0,00	* 0,00	96,30	70,37
A	53,85	105,26	112,50	15,56	100,00	100,00	* 0,00	* 0,00	100,00	70,00
R	60,00	102,56	88.89	* 0,00	* 0,00	100,00	104,00	52,00	107,69	83,33
Ø	52.26	103.52	102.53	18.59	100.00	100.00	102.00	49.08	101.00	76.24

Tabelle 8.10.: Integritätsabschätzungen in Prozent anhand der angelernten Koeffizienten

In der untersten Zeile sind jeweils die Durchschnittswerte der Spalten dargestellt. Die Ergebnisse für das Querruder, das Seitenruder und die Landeklappe spiegeln die verbliebenen Integritäten mit hoher Genauigkeit wider. Der Wert für das Höhenruder liegt um etwa 25% daneben. Problematisch ist die Erkennung von Motorausfällen. Der in Abschnitt 7.4.4 beschriebene Sonderfall zur Identifizierung der Motoren vernachlässigt das Vorhandensein eines Ausfalls. Der Control Allocator muss dahingehend erweitert werden, dass bei degradierter Integrität der Längsbeschleunigung, bzw. der Motoren, nicht auf die angelernten Koeffizienten vertraut werden darf.

8.3.5. Einschätzung

Wie vorab erläutert kann keine Aussage darüber getroffen werden, ob die in den Tabellen 8.7 und 8.8 präsentierten Resultate gute Ergebnisse darstellen. Durch den Lernalgorithmus werden die Effektivitäten der Steuerflächen in den Bereichen von 60% bis 80% geschätzt. Die Dauer der Lernphase betrug in allen Messungen jeweils insgesamt 64 Sekunden. Kürzere oder längere Lernphasen wurden nicht untersucht. Diese erzielten Resultate bleiben daher ohne Wertung.

Betrachtet man die vorgestellten Schwächen der fehlenden Informationen über die Symmetrien und den benötigten Gleichlauf, der nicht isoliert durchgeführten und im Umfang sehr

kleinen Systemidentifikation sowie der fehlenden Interpretation der Koeffizienten, wurden zunächst schlechtere Ergebnisse erwartet. Die realisierte Umsetzung des dynamischen Lernens der Effektivitäten wird daher als geeigneter Ansatz eingeschätzt.

Das Anlernen bei degradierter Priorität zeigt mit Schwächen bei den Höhenrudern und einer eindeutigen Problemstellung bei den Motoren sehr gute Resultate. Die verbliebenen Effektivitäten werden mit Ausnahme der genannten mit hoher Genauigkeit abgeschätzt. Somit bleiben die Reihenfolgen der Prioritäten gleich, wodurch dem Allokationsalgorithmus die korrekten Verhältnisse zur Verteilung der Steuerleistung zur Verfügung stehen.

8.4. Adaptive Flugsteuerung durch Reallokation

Eine theoretische Analyse der Stärken und Schwächen der verschiedenen Allokatoren wurde bereits in Abschnitt 7.5 präsentiert. Es folgt ein Vergleich der Allokatoren anhand eines simulierten Fluges, bei welchem die Integrität beider Querruder auf 20% verringert wurde. Um Messunsicherheit des Fault Detectors auszuschließen und so eine bessere Vergleichbarkeit der Allokatoren zu ermöglichen, wurden die Integritätszustände statisch mit einem *Packet Simulator* im Softwarebus verteilt.

8.4.1. Simulierter Flug und Allokatoren

Es wurden insgesamt 8 Flüge durchgeführt, von denen einer die Referenz der Flugzustände für den fehlerfreien Flug darstellt. Sie wurden jeweils vom Pilot Simulator mit den Einstellungen des Datensatzes **A** aus Tabelle 8.1 gesteuert. Es fand demnach bewusst keine Kursregelung statt, um die Performance der Allokatoren anhand der Abweichungen vom Referenzflug besser analysieren zu können. Geregelt wurde lediglich das Wiederherstellen eines Horizontalflugs. Simuliert wurde jeweils 40 Sekunden. Das Triggern des Ausfalls sowie der Integritätsverlust wurde während eines Horizontalflugs im Zeitfenster zwischen Sekunde 20 und 21 im Softwarebus verteilt. Dargestellt sind jeweils die Sekunden 19 bis 37.

Verglichen werden alle in Kapitel 7 vorgestellten Allokatoren[2]. *Static* stellt die bisherige Flugsteuerung im AESLink Framework und im Flugsimulator dar und beinhaltet somit keine Adaptivität. Die *PriorityTable* ist so konfiguriert worden, dass das Rollmoment im gewählten Integritätszustand mit den Höhenrudern erzeugt wurde. Dem *DaisyChaining* wurden als Effektivitäten für das Rollen die in Tabelle 8.8 dargestellten Koeffizienten einprogrammiert. Um die Ergebnisse zu vergleichen, wurden die Koeffizienten des *DynamicAgents* für den degradierten

[2]Nachfolgend werden diese ihrem Typ nach wie sprechende Namen verwendet.

Integritätszustand vorher angelernt. Bis zum Eintreffen der Nachricht über den Integritätsverlust ist mit der Standardzuordnung gesteuert worden. Ab diesem Zeitpunkt wurde auf die zuvor angelernte Steuermatrix umgeschaltet.

Die Resultate für die gekoppelte Ansteuerung der Seitenruder aus Abbildung 8.6 zeigen ein starkes Schwingen in der Rollrate. Daher wurden weitere Flüge durchgeführt, bei welchen dem DaisyChaining (*DCwoRUDs*) und dem DynamicAgent (*DAwoRUDs*) die zugehörigen Koeffizienten auf 0 erzwungen wurden. In einem weiteren Flug wurden die angelernten Koeffizienten des DynamicAgents (*DA42*) nicht mit dem höchsten angelernten Wert skaliert, sondern mit dem aus der Messung des Gleichlaufs erhaltenen Maximalwert von 42 (siehe Tabelle 8.7).

8.4.2. Ergebnisse

Durch die vielen verschiedenen Anwendungen im simulierten Regelkreis, welche durch das verwendete Testsystem lediglich eine bedingte Echtzeitfähigkeit aufweisen, konnte keine perfekte Reproduktion erreicht werden. Die Funktionsverläufe in den nachfolgenden Diagrammen wurden auf der X-Achse so verschoben, dass die letzte Kurve im fehlerfreien Flug bei Sekunde 19.5 jeweils übereinanderliegen. Die maximale Verschiebung liegt bei 7 Regelschritten, was 140 Millisekunden entspricht. Die Ergebnisse können daher nicht vollständig bewertet werden.

Erzeugung von Rollmoment

In den Abbildungen 8.7 und 8.8 sind die erreichten Rollraten und Rollwinkel dargestellt. *Static* hat weiterhin die Querruder zur Steuerung verwendet. *PriorityTable* steuerte mit den Höhenrudern. Beide können daher nicht das volle Rollmoment erzeugen und erzielen damit deutlich geringere Rollraten und -Winkel. *DAwoRUDs* liefert eine ebenso geringe Rollrate. Dieser hatte in seiner Lernphase den Landeklappen die höchsten Koeffizienten von jeweils etwa 0.50 zugeordnet, wonach die restlichen Koeffizienten nach diesen skaliert wurden. Der Allokationsalgorithmus interpretiert diese so, als würde das durch die Landeklappen erzeugte Rollmoment das Maximum darstellen. Wird ein virtuelles Steuersignal v_{cmd} von 1 zugeordnet, werden die Landeklappen voll angesteuert, wodurch sich algorithmisch ein v_{is} von 1 ergibt. Hier zeigt sich die fehlende Interpretation der Koeffizienten, was als eindeutige Schwäche zu bewerten ist.

DA42 erzeugte in jeder Kurve zu große Drehraten. Diese werden in der fehlenden Berücksichtigung der gleichzeitigen Verwendung der Steuerflächen in mehreren Freiheitsgraden vermutet (siehe Abschnitt 7.3.2). Wird diese Annahme zugrunde gelegt, liegt die Bestätigung

der Erwartung vor, dass der Allokationsalgorithmus das Control Allocation Problem in seiner aktuellen Realisierung nicht optimal löst.

Die restlichen Allokatoren liefern gute Resultate. Das Einleiten einer Kurve wird vom Signalgenerator des Pilot Simulators *gesteuert*. Die Wiederherstellung des Horizontalfluges erfolgt durch den eingebetteten *WingLeveller* und wird *geregelt*. Betrachtet man die Rollraten zum Einleiten einer Kurve und den resultierenden Rollwinkel, liegen die Allokatoren *DaisyChaining*, *DCwoRUDs* und *DynamicAgent* nah am Referenzzustand. Lediglich die letzte dargestellte Kurve wurde in allen drei Fällen zu stark angesteuert.

Bei Wiederherstellung des Horizontalflugs zeigt sich bei diesen jedoch ein deutliches Schwingen. Der WingLeveller ist in seiner Parametrierung weiterhin auf die erzeugte Dynamik der Querruder eingestellt. Durch den hohen K_p-Anteil des implementierten PID-Reglers wird hierdurch ein Sprung des Steuersignals erzeugt. Im Flugsimulator rotieren alle Steuerflächen mit der gleichen Geschwindigkeit. Werden mehrere Steuerflächen zugleich angesteuert, ergibt sich in Summe eine höhere Dynamik des Systems, als durch das alleinige Rotieren der Querruder. Diese Nichtlinearität wird weder im Regler, noch im Allokationsalgorithmus kompensiert. Durch das Vernachlässigen der Dynamik und das Linearisieren der Steuermatrix in einem Arbeitspunkt passen Reglerparameter und Steuerflächenzuordnung nicht länger optimal zusammen. Eine zu hohe geforderte Dynamik des Systems, wie etwa beim Sprung zur Wiederherstellung des Horizontalflugs, ergibt somit Überschwingen. Hier zeigt sich erneut, dass das Maskieren der degradierten Effektivität vor dem Flugregler keine optimale Lösung darstellt.

Die Resultate zeigen, dass eine reine Steuerung von diesen drei Steuermatrix-basierten Allokatoren gut umgesetzt wird. Die Regelung erfordert eine zu hohe Dynamik, welche von der entkoppelten Flugsteuerung nicht optimal umgesetzt wird.

Gierwinkel und Position

In den Abbildungen 8.9 und 8.10 sind die Verläufe der Gierwinkel und der Positionen in der XY-Ebene dargestellt.

Static kann dem Referenzgierwinkel nur relativ, nicht jedoch absolut folgen. Dieses Ergebnis wurde erwartet, da weiterhin die Querruder mit ihren verbliebenen 20% Effektivität verwendet werden. *PriorityTable* und *DAwoRUDs* können der hohen Frequenz der vom Pilot Simulator erzeugten Kurven nicht folgen und entfernen sich damit zunehmend vom Referenzzustand. *DA42* liegt aufgrund seiner zu hohen Dynamik phasenweise ebenfalls sehr weit vom Referenzgierwinkel entfernt.

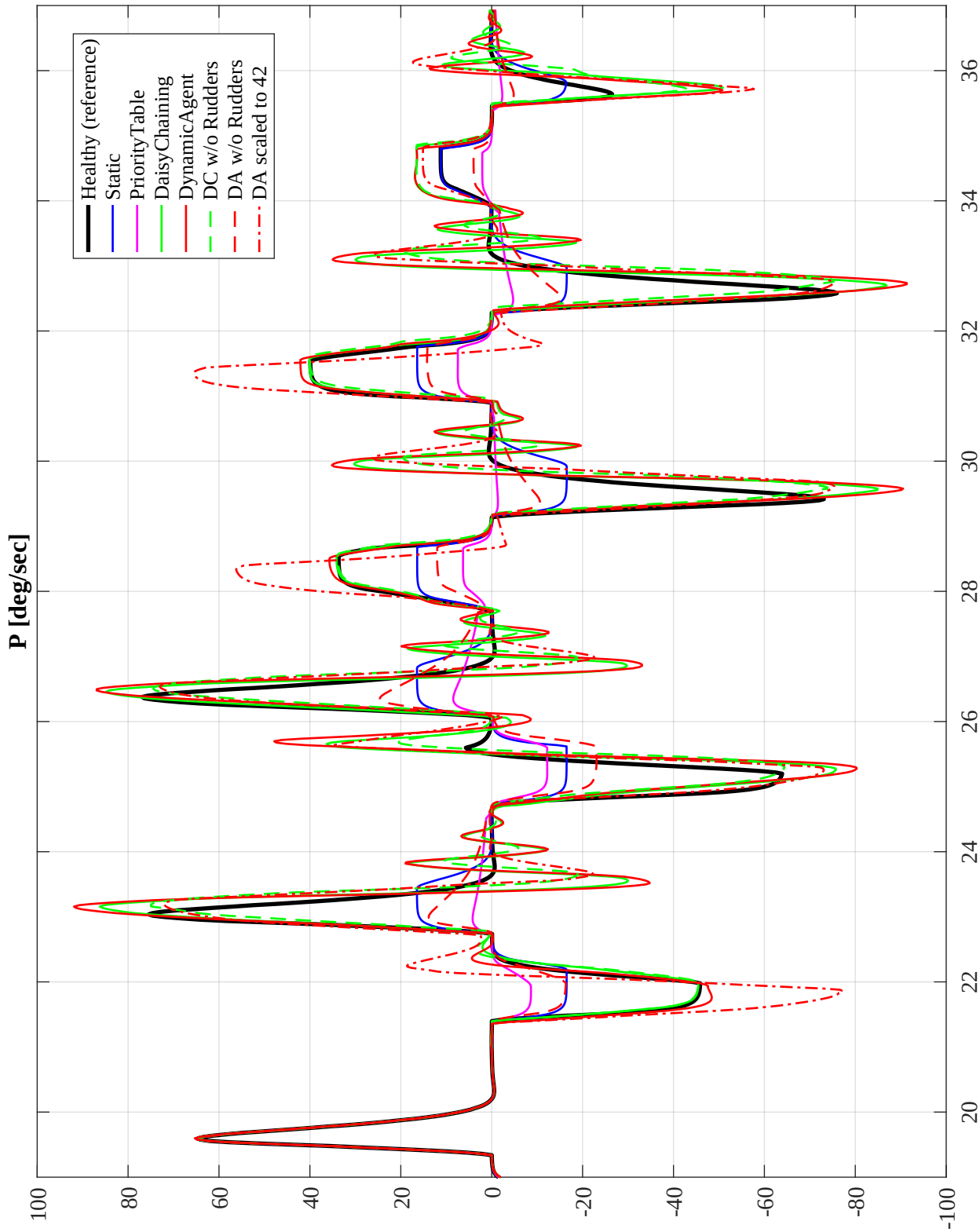

Abbildung 8.7.: Rollraten der simulierten Flüge bei degradierter Integrität der Querruder

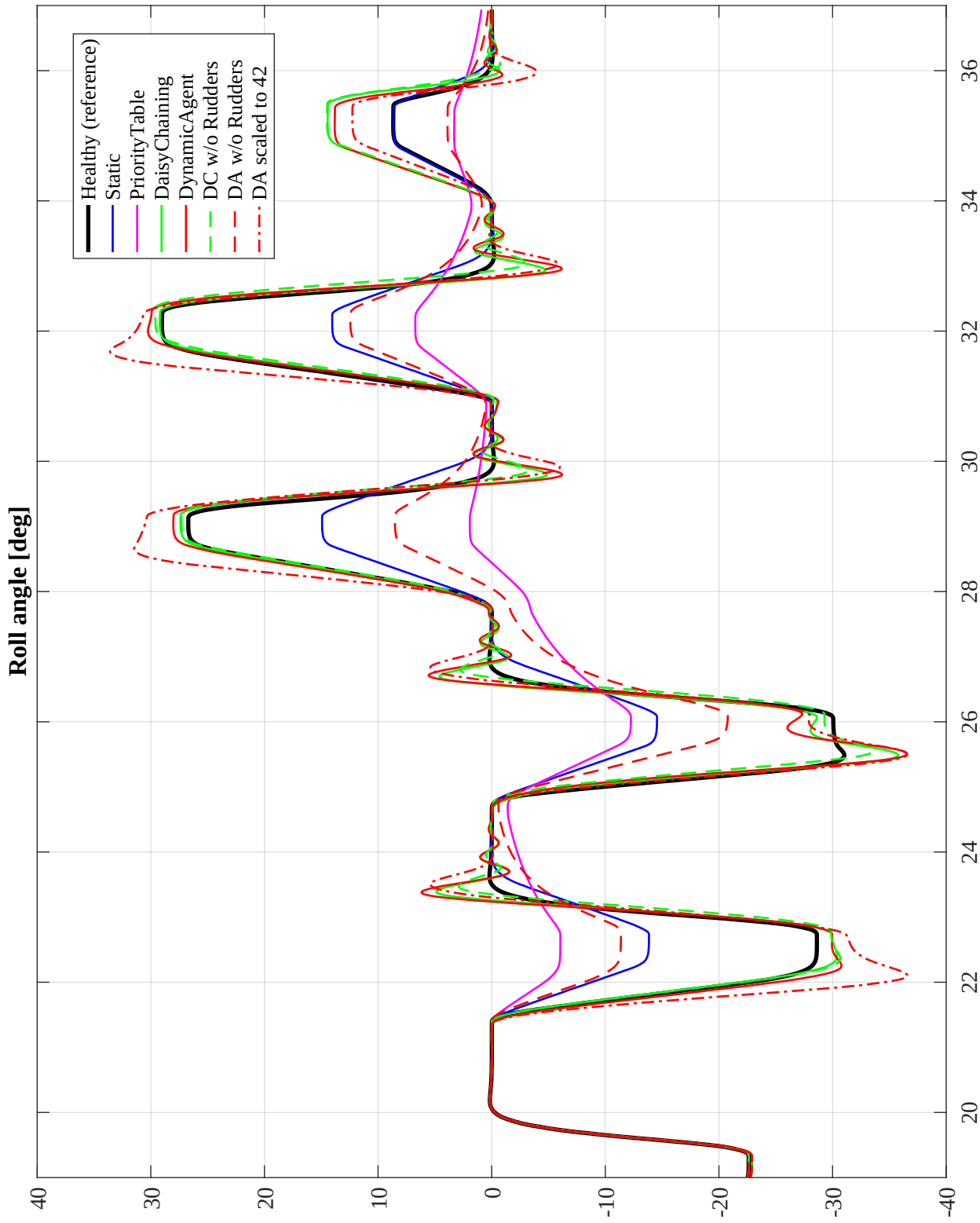

Abbildung 8.8.: Rollwinkel der simulierten Flüge bei degradierter Integrität der Querruder

Die Gierwinkel beschreiben den Vektor entlang der Längsachse im flugzeugfesten Koordinatensystem. Daher überträgt sich das im letzten Abschnitt beschriebene Schwingen auch auf den Verlauf dieser Vektoren. Abgesehen von diesem temporären Schwingen liefern die übrigen drei Allokatoren ein gutes Ergebnis. Dem Referenzgierwinkel wird mit annehmbarer Genauigkeit gefolgt.

Der Ausschnitt der Flugphasen wurde so gewählt, dass sich zu Beginn und am Ende der gleiche Referenzgierwinkel ergibt. Es werden somit gleich viele Links- und Rechtskurven geflogen. Durch die zu geringe Steuerleistung der Allokatoren *Static*, *PriorityTable* und *DAwoRUDs* kann die vom Pilot Simulator geforderte Dynamik nicht umgesetzt werden und die Gierwinkel entfernen sich vom Referenzzustand. Für *DA42* ist positiv zu erwähnen, dass dieser sowohl in Links-, als auch in Rechtskurven eine zu hohe Dynamik erzeugt und dadurch am Ende wieder den Referenzgierwinkel erreicht. Die übrigen Steuermatrix-basierten Allokatoren folgen dem Referenzzustand während der Flugphase mit Ausnahme der Schwingungen gut. Auch zum Ende der Flugphase liegen ihre Gierwinkel nahe am Referenzzustand, was positiv zu bewerten ist.

Bei der Evaluierung der Positionstreue kann ein ähnliches Verhalten beobachtet und somit die gleichen Schlüsse gezogen werden. Wird von der zu Beginn vorhandenen Differenz abgesehen und die Funktionsverläufe am ersten Punkt in der Y-Achse übereinandergelegt, ergibt sich für die drei *"guten"* Allokatoren eine hohe Übereinstimmung mit dem Positionsverlauf der Referenzmessung. Die beiden Anpassungen des dynamischen Agenten brachten nicht die erwarteten Verbesserungen. Stattdessen erzeugen diese deutlich schlechtere Resultate, als die allgemeine Version.

Werden die Ergebnisse der Rollwinkel und -raten vernachlässigt und ausschließlich die Gierwinkel und Positionen betrachtet, liefert der Allokationsalgorithmus in Zusammenhang mit den angelernten Koeffizienten sehr gute Ergebnisse. Bei Ausfall folgt durch den Verlust der Steuerbarkeit in der Rollachse mit der nicht adaptiven Standardzuordnung ein unausweichlicher Absturz. Durch die Reallokation der benötigten Steuerleistung auf die verbliebenen Steuerflächen ist weiterhin die Steuerbarkeit in allen Freiheitsgraden gegeben. Die umgesetzte adaptive Steuerflächenzuordnung wird somit als Erfolg gewertet.

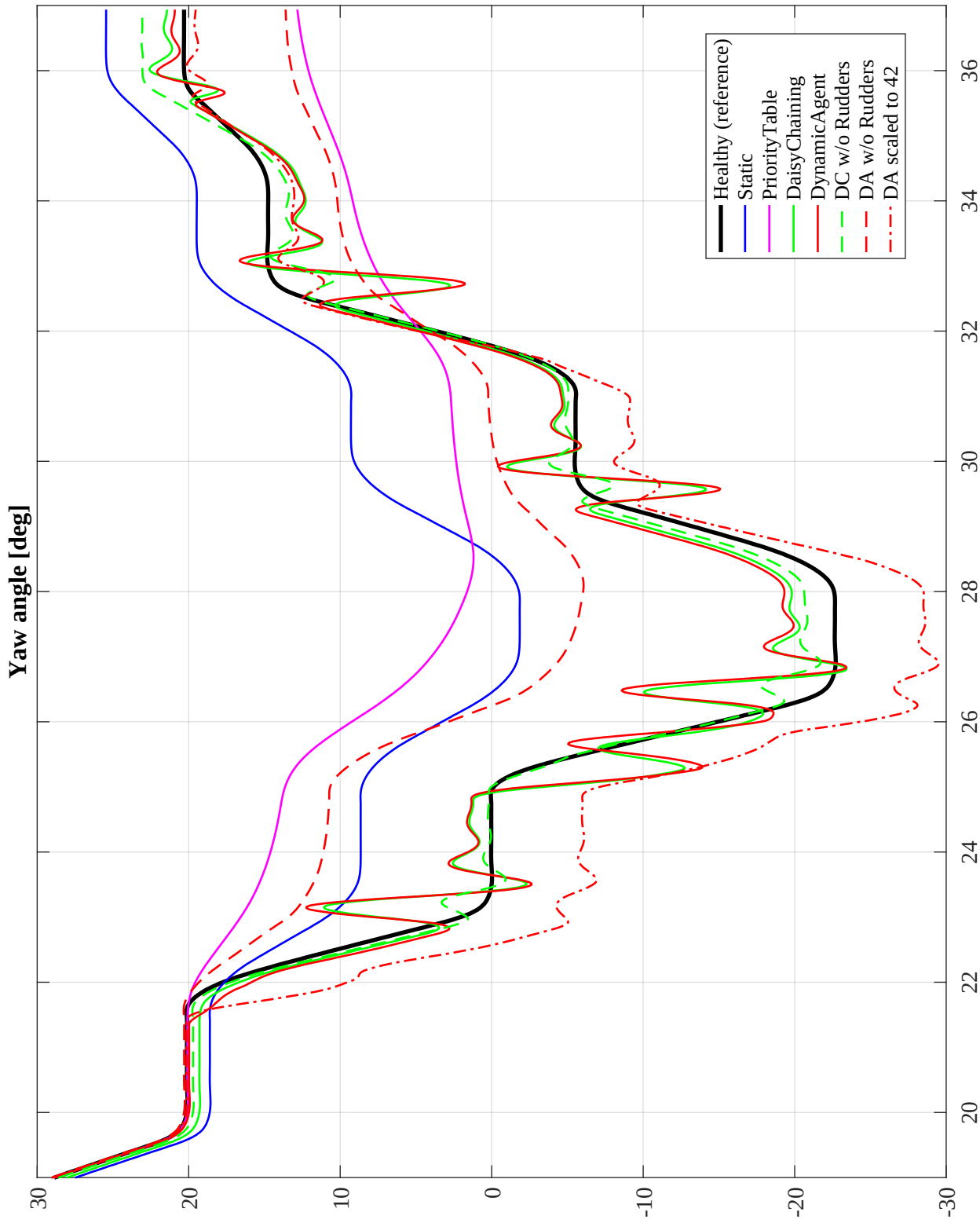

Abbildung 8.9.: Gierwinkel der simulierten Flüge bei degradierter Integrität der Querruder

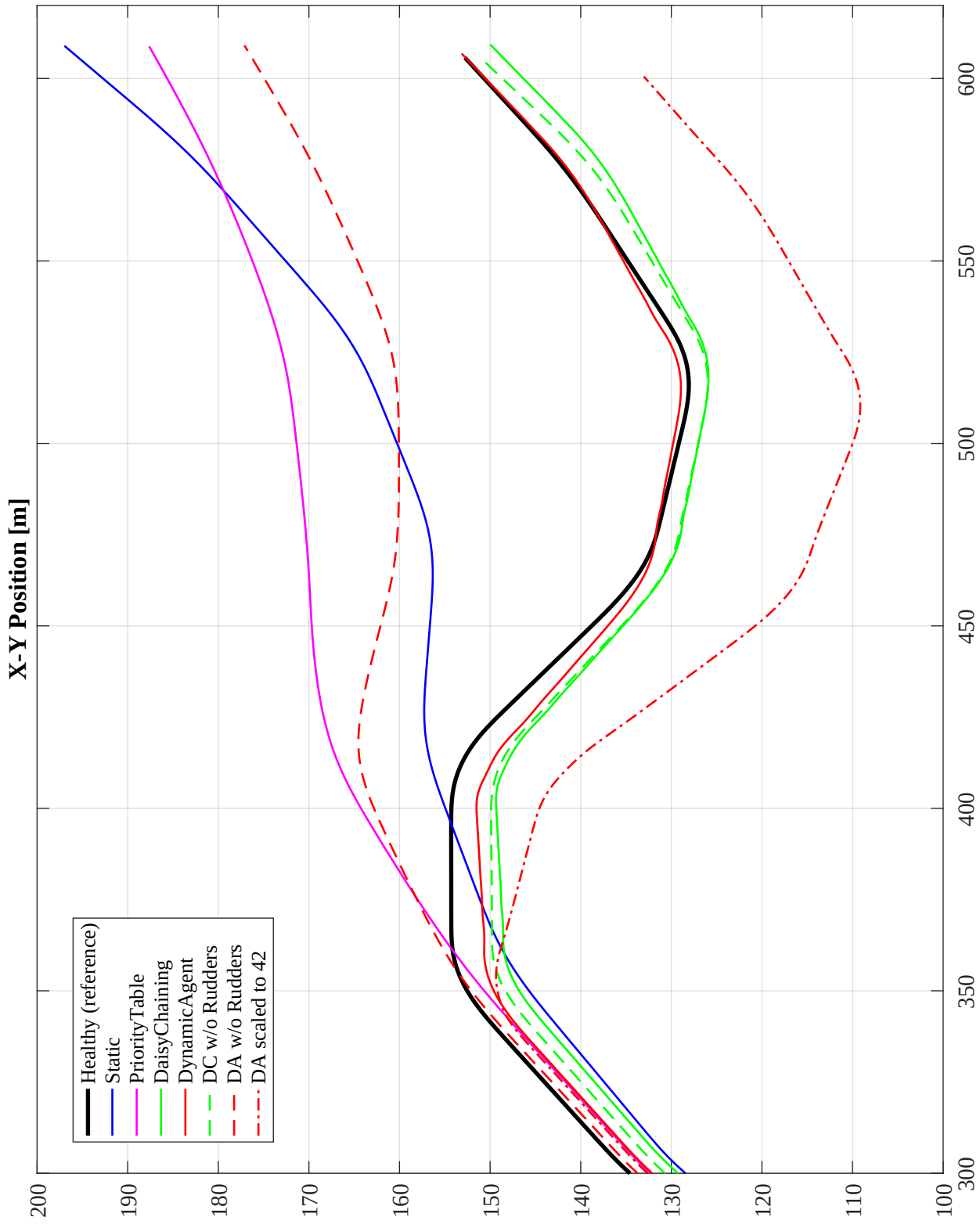

Abbildung 8.10.: XY-Positionen der simulierten Flüge bei degradierter Integrität der Querruder

9. Schluss

Zum Abschluss der Arbeit erfolgt zunächst eine Zusammenfassung der durchgeführten Arbeitsschritte und der realisierten Systemteile. Im letzten Abschnitt wird ein Fazit über die erzielten Ergebnisse gezogen und im Zuge dessen ein Ausblick dargelegt, welche Erweiterungen und Arbeitspakete die entwickelten Softwarekomponenten verbessern können.

9.1. Zusammenfassung

Das Ziel dieser Masterthesis lag darin, eine fehlertolerante Flugsteuerung mit inhärenter Adaptivität zu entwickeln. Diese soll die Autonomie der in der AES Arbeitsgruppe entwickelten *Flight Control Unit* erhöhen und das Risiko eines Absturzes minimieren. Es stand zunächst die grundlegende Idee im Raum, den Ausfall einer Steuerfläche durch eine dynamische Reallokation der intakten Steuerflächen zu kompensieren. An das Konzept und die Umsetzung wurden verschiedene eigene Anforderungen gestellt.

Zum einen sollte sich der Ansatz leicht auf andere Trägersysteme übertragen lassen können. Hierzu gehörten zwei Teilaspekte. Es sollte zunächst möglich sein, dass ein Dritter eine Übertragung ohne tiefgreifende Kenntnisse aus den Fachgebieten der Flugmechanik und Regelungstheorie durchführen kann. Diese Anforderungen brachten mit sich, dass bei der Übertragung in jeder Hinsicht auf eine analytische Modellbildung des Flugverhaltens verzichtet wird.

Zum anderen wurde eigens festgelegt, dass ein Ausfall vor dem Flugregler maskiert und die Adaptivität durch ein eigenes Teilsystem umgesetzt wird. Durch die Trennung der Zuständigkeiten für die *Flugregelung* und die *Flugsteuerung* sollte eine bessere Wiederverwendbarkeit der einzelnen Komponenten erreicht werden. Wird der Träger von einem menschlichen Piloten *geregelt*, kommt der Maskierung eine deutlich höhere Bedeutung zu. Für diesen ist es wichtig, dass er sich bei einem Ausfall nicht umorientieren muss und er weiterhin seine gewohnten Steuerkanäle verwenden kann.

Die dritte primäre Anforderung bestand darin, dass für eine Einbettung in die Flight Control Unit möglichst wenig a priori Systemwissen über den zu steuernden Träger einprogrammiert werden soll. Zunächst sollte hierdurch der Aufwand der Übertragung auf einen neuen Träger

minimiert werden. Des Weiteren sollte ein möglichst generisches Steuersystem entstehen, welches konzeptionell auf andere technische Systeme als Flugzeuge übertragen werden kann. Ebenso sollte hierdurch evaluiert werden, auf wie viel a priori einprogrammiertes Systemwissen verzichtet werden kann, um das Vorhaben dennoch realisieren zu können.

Unter Beachtung der gegebenen Anforderungen wurde eine umfangreiche Analyse über mögliche Techniken und Konzepte durchgeführt. Daraufhin wurde ein Ansatz entwickelt und präsentiert, welcher aus zwei getrennten Teilsystemen Ausfallerkennung und adaptive Steuerflächenzuordnung besteht. Um den Anforderungen der nicht benötigten Modellierung zu genügen, setzen beide Komponenten auf ein Anlernen des Flugverhaltens. Zur Evaluierung des Ansatzes wurde dieser zunächst in einem simulierten Regelkreis realisiert.

Anfänglich wurde die Simulationsinfrastruktur AESLink dahingehend erweitert, das diese das Entwickeln und Evaluieren einer simulierten adaptiven Flugsteuerung ermöglicht. Hierfür wurden der Pilot Simulator und der Fault Injector realisiert sowie das Plugin des verwendeten Flugsimulators erweitert. Der Pilot Simulator ermöglicht es, ein im Flugsimulator gewünschtes Flugverhalten zu reproduzieren. Der Fault Injector in Kombination mit dem Plugin triggert einen Ausfall der Steuerflächen im Flugsimulator.

Den Kern der Ausfallerkennung bilden neuronale Netze. Diese werden angelernt und bilden das Flugverhalten nach, ohne dass eine analytische Modellbildung nötig ist. Ein Ausfall, bzw. die verbliebene Integrität dieser, wird anhand der Differenz zwischen geschätztem und tatsächlichem Systemzustand erkannt. Zur einfachen Übertragung auf eigene Träger durch Dritte wurde eine umfangreiche Arbeitsumgebung geschaffen, welche das Anlernen des Flugverhaltens durch die Netze ohne tiefgreifende Kenntnisse aus der Flugmechanik erlaubt. Die angelernten Netze werden zur Ausführung des simulierten Regelkreises in die eigenständige Anwendung Fault Detector eingebettet. Dieser realisiert die Softwarekomponente Ausfallerkennung.

Eine Steuerflächenzuordnung führt eine Abbildung der vom Flugregler vorgegebenen Steuersignale auf die vorhandene Aktorik durch. Eine Adaptivität in dieser Abbildung kommt zustande, wenn die benötigte Steuerleistung in Ausfallzuständen auf die intakten Steuerflächen verteilt, bzw. reallokiert, wird. Für die adaptive Steuerflächenzuordnung wurde ein dynamisch lernender Allokator konzeptioniert und realisiert. Dieser lernt das Systemverhalten während eines Fluges und ist daher lediglich in geringem Maße auf eine Vorgabe von System- oder Expertenwissen angewiesen. Zur Evaluierung des Ansatzes wurden weiterhin verschiedene statische Allokatoren entwickelt, bei welchen ein Wissen über das Systemverhalten vorgegeben wird. Zur Realisierung im simulierten Regelkreis wurden die Allokatoren in der Anwendung Control Allocator umgesetzt.

9.2. Fazit und Ausblick

Im Fault Injector und Flugsimulator wurde die Umsetzung verschiedener Ausfalltypen implementiert. Das Triggern von Ausfällen, bzw. einer eingeschränkten Integrität, war für den simulierten Regelkreis unverzichtbar und kann einfach durchgeführt werden. Zur Evaluierung wurden lediglich ein Einfrieren und eine eingeschränkte Ansteuerung verwendet. Interessant wäre die Analyse der Performance der Integritätsabschätzung des Restwertanalysators bei eingeschränkter Rotationsgeschwindigkeit gewesen. Jedoch konnte dies im Flugsimulator vorerst nicht umgesetzt werden, da hierfür Informationen über die Dynamik der Aktuatoren fehlten und der Flugsimulator selbst keine Programmierschnittstelle zur Einschränkung bietet. Zur weiteren Analyse der Performance der Ausfallerkennung sollte dieser Ausfalltyp getriggert werden können. Die Umsetzung stellt damit ein zukünftiges Arbeitspaket zur Weiterentwicklung der Simulationsinfrastruktur dar.

Während der Evaluierung hat sich gezeigt, dass die Kombination aus Flugsimulator und Pilot Simulator eine nicht ausreichende Reproduzierbarkeit bietet. Sowohl bei der Analyse der Integritätsabschätzungen, als auch beim Vergleich der Allokatoren zeigten sich Abweichungen im Flugverhalten. Es ist zu analysieren, ob dies durch einen zu hohen Workload am System, fehlende Echtzeit des Flugsimulators oder eine fehlerhafte Implementierung des Pilot Simulators zustande kommt.

Bei der Entwicklung der Teilkomponenten reichte die Reproduzierbarkeit jedoch bei weitem aus. Daher lässt sich konstatieren, dass sich die Entwicklung des Pilot Simulators in vollem Umfang ausgezahlt hat. Während des Debuggens konnte sich in eine laufende Simulation mit *"realen"* und dabei wahlweise zufälligen oder bestimmten Steuereingaben eingeklinkt werden, ohne dass die Steuereingaben oder Sollwerte der Regler händisch angepasst werden mussten.

Das Fazit, Matlab als Werkzeug zur Entwicklung der neuronalen Netze zu verwenden, fiel zu Beginn der Entwicklungen zwiespältig aus. Beim Umgang mit den zur Evaluierung verwendeten Daten, der fließenden Erstellung von Diagrammen, den unzähligen Möglichkeiten zur mathematischen Analyse, den eingebetteten numerischen Verfahren, usw. zeichnet sich Matlab deutlich aus. Jedoch bieten die vordefinierten Netztypen keine große Auswahl. Erst in neueren Versionen der Neural Network Toolbox können eigene Netzarchitekturen bestimmt werden[1]. Andere Frameworks, wie z. B. *TensorFlow*, bieten schon länger die Definition eigener Strukturen. Zum Abschluss fällt das Resümee durchweg positiv aus. Die Möglichkeit zur einfachen Einbettung in eigene Anwendungen durch das Matlab Compiler SDK bietet einen enormen Zugewinn an möglicher Portabilität. Die in einfachen Skripten persistierten Netze

[1] https://de.mathworks.com/products/neural-network/features.html [Stand: 28.05.2018]

können mit dem Matlab Compiler in ANSI-C Code umgewandelt werden. Dadurch können diese sehr einfach in einem eingebetteten System wie der Flight Control Unit verwendet werden.

Die geschaffene Arbeitsumgebung zur Entwicklung der neuronalen Netze bietet einen umfangreichen Funktionsumfang bei gleichzeitig einfacher Bedienung. Für die Evaluierung wurde das Training mit verschiedenen Datensätzen automatisiert. Besonders bei größeren Datensätzen und der Analyse verschiedener NetTasks hat sich diese Automatisierung bewährt. Ein neuer Datensatz kann mit einem beliebigen Netztyp und NetTask angelernt werden. Hierfür müssen lediglich drei Strings zur Vorgabe dieser in den Fassaden-Skripten eingetragen werden. Zum aktuellen Stand der Entwicklung können im Skript zur Live-Klassifizierung ausschließlich FF-Typen verwendet werden. Zur Vorabanalyse kann sich diese unter Umständen lohnen, da das verwendete Netz zur Laufzeit aktualisiert werden kann und hierdurch die Zustände und Variablen nicht verloren gehen. Die Erzeugung der DLL hat bei 30 einzubettenden Netzen etwa eine Minute gedauert. Weiterhin muss der Fault Detector in jedem Fall neu gestartet werden, wodurch die internen Zustände nicht länger verfügbar sind.

Über die in der Arbeitsumgebung enthaltenen Netztypen kann kein umfassendes Fazit gezogen werden. Aufgrund der Komplexität einer umfassenden Evaluierung, wurde lediglich ein Teilausfall der Querruder evaluiert. Bei dieser haben sich die Fit und NARX Netze bewährt. Für andere Datensätze, NetTasks, Anzahl Neuronen, usw. könnten sich die anderen Netztypen als geeignet erweisen. Daher lässt sich nicht abschätzen, ob die investierte Zeit zur Unterstützung der restlichen Netztypen vergebens war. Bei der eigenen Recherche über mögliche Techniken zur Ausfallerkennung wurden mehrere Arbeiten gesichtet, in welchen der Einsatz von RBF Netzen zur Nachbildung des Flugverhaltens präsentiert wurde. Die Totalausfälle der RBF Netze in der eigenen Evaluierung werden daher als eigener Implementierungsfehler, bzw. in einer falschen Verwendung der Toolbox, vermutet. In der aktuellen Version (R2018a) der Neural Network Toolbox sind Implementierungen von LSTMs enthalten. Es wäre interessant diese zu analysieren, um zu erfahren, ob sich LSTMs für das gegebene Problem generell nicht eignen oder sich der Einsatz von Cortexsys nicht bewährt hat. Zum Ende der Arbeit ist die eigene Einschätzung, dass diese sich weniger eignen, da in der Flugmechanik keine Abhängigkeiten zu Zuständen und Sequenzen weit zurückliegender Zeitpunkte bestehen. Auf der anderen Seite werden regelmäßig Integratoren verwendet, welche den Zustand speichern. Es ist daher eine interessante Fragestellung.

Aufgrund der aufgestellten Anforderungen, zum einen bei der Übertragung der Flugsteuerung auf andere Trägersysteme auf jeglicher Modellbildung verzichten zu können und zum

anderen, dass lediglich ein Minimum an System- und Expertenwissen vorhanden sein soll, wurden auf Lernmethoden basierende Verfahren gewählt. Die Anforderungen sind daher in vollem Umfang erfüllt worden.

Für die neuronalen Netze konnte keine umfassende Analyse über das abzubildende Modell der Übertragungsfunktion (NetTask) durchgeführt werden. Demnach, ob *ein* Modell existiert, welches bei allen Datensätzen und Trägersystemen gute Ergebnisse liefert. Dies wird nicht erwartet, da das vorgestellte analytische Modell der Bewegungsgleichung auch nur ein generelles ist. Wie beschrieben, wird dieses je nach zu lösender Problemstellung erweitert oder es wird nur ein Teilmodell verwendet. Daher muss dieses in Form von NetTasks bei der Nachbildung des Flugverhaltens bestimmt werden. Es muss bei der Übertragung also schon eine gewisse Kenntnis von Flugverhalten und Flugmechanik mitgebracht werden, jedoch steht dies in keinem Vergleich zur analytischen Systemidentifikation und der Suche nach optimalen Parametern. Wird die Komplexität des gegebenen technischen Systems betrachtet, welches von den Netzen abstrahiert wird, stellt die Vorgabe der NetTasks einen minimalen Aufwand dar. Daher erfüllen die Wahl für und die Umsetzung der neuronalen Netze die gegeben Anforderungen vollständig.

Bei der Evaluierung wurde präsentiert, dass neuronale Netze kein Wunder- oder Allheilmittel darstellen. Wie bei der analytischen Modellbildung müssen auch hier alle möglichen Systemzustände während der Modellierung verfügbar sein. Dieses Problem ist bei jeglicher Form der Modellbildung vorhanden und wird durch die Wahl des Modellierungverfahrens in seiner Komplexität nicht verringert.

Wird davon abgesehen, dass die zur Evaluierung vorhandene Datenbasis zu gering war, ließen sich durch die Netze sehr gute Ergebnisse erzielen. In Kombination mit dem präsentierten Restwertanalysator konnte der verbliebene Integritätszustand mit hoher Genauigkeit abgeschätzt werden. Zur Weiterentwicklung des präsentierten Systems sollten auch andere Analysatortypen implementiert werden, um ein größeres Spektrum an Fehlern erkennen zu können.

Das dynamische Anlernen der Koeffizienten kann nicht vollständig bewertet werden, da Vergleichsergebnisse fehlen. Angesichts der in sehr geringem Umfang durchgeführten Systemidentifikation, werden die erzielten Resultate als sehr gut bewertet. Diese eignet sich ähnlich gut zur Abschätzung der verbliebenen Integrität, wie die Ausfallerkennung selbst. Der Ansatz sah zum Informationsaustausch zwischen Ausfallerkennung und Steuerflächenzuordnung ausschließlich die Integritätsabschätzung in einem einzigen Prozentwert vor. Hier liegt eine Vielzahl an Erweiterungsmöglichkeiten vor. Wird die Ausfallerkennung um zusätzliche Analysatortypen erweitert, kann ein besseres Zusammenspiel zwischen den Komponenten erfolgen und so die Fehlertoleranz des Systems erhöht werden. Wird die Performance des

Gesamtsystems im Zuge der geringen Kopplung betrachtet, ist diese jedoch bereits jetzt als sehr gut zu werten.

Während der eigenen Evaluierung haben sich eine Vielzahl an Verbesserungsmöglichkeiten für das dynamische Anlernen gezeigt. Im aktuellen Entwicklungsstand können bereits jetzt die symmetrischen und gekoppelten Steuerflächen erkannt werden. Es könnte eine zweite Lernphase mit gleichlaufender Ansteuerung der symmetrischen Aktuatoren erfolgen, um die Koeffizienten noch besser abschätzen zu können.

Dem Lernalgorithmus ist unbekannt, in welche Richtung die Steuerfläche gedreht wird. Beim verwendeten Träger wurden zum Beispiel die beiden Querruder jeweils unterschiedlich bewegt. Dadurch wird in beiden Fällen ein Rollmoment nach rechts und ein Nickmoment nach oben erzeugt. Somit mussten beide der Gravitation entgegen wirken und es ergeben sich geringere Drehraten. Um dies zu verhindern, kann wie oben eine zweite Lernphase mit entgegengesetzter Ansteuerung durchgeführt werden. Diese beiden Verbesserungen der gleichlaufenden und jeweils entgegengesetzter Ansteuerung kann in einer gemeinsamen zweiten Lernphase erfolgen.

Für die Interpretation der Koeffizienten kann keine Lösung präsentiert werden. Durch den zuvor beschriebenen Ansatz der erweiterten Lernphase werden höhere Drehraten erreicht. Jedoch ergeben sich hierdurch weiterhin Koeffizienten mit dem Maximalwert 0.5, bzw. der Daisy Chaining Gruppengröße entsprechend. In der Evaluierung wurde präsentiert, dass die Effektivitäten und die daraus resultierenden Koeffizienten in Relation zueinander zum aktuellen Entwicklungsstand bereits gut abgeschätzt werden. Werden alle Steuerflächen auf zum Beispiel 80% verbliebene Integrität degradiert, bleiben die angelernten Relationen weiterhin gleich. Der Allokationsalgorithmus interpretiert die Koeffizienten dahingehend, dass diese der vollen Effektivität entsprechen. So werden auch bei geringen Drehraten lediglich 80% der vom Regler erwarteten Drehrate erzeugt, obwohl die Steuerflächen nicht an ihrem Limit arbeiten und eine 20% höhere Ansteuerung möglich wäre, um die vom Regler geforderte Drehrate zu 100% zu erzeugen. Dieses Problem hat sich beim dynamischen Allokator ohne Verwendung der Seitenruder gezeigt.

Die Interpretation fehlt aufgrund der aufgestellten Anforderung, eben dieses Expertenwissen nicht in der Control Allocation vorgeben zu müssen. Der Allokationsalgorithmus liefert in seiner aktuellen Ausprägung hierdurch ein durchwachsenes Ergebnis. Zwar konnte dieser in der finalen Evaluierung dem Referenzzustand relativ gut und deutlich besser als die statischen Allokatoren folgen, jedoch traten massives Schwingen und teilweise deutlich zu hohe oder zu geringe Ansteuerungen auf.

Das Control Allocation Problem wird demnach noch nicht optimal gelöst. Für einen produktiven Einsatz in der Flight Control Unit ist er somit nur bedingt geeignet. Es müssen weitere Bedingungen und Kostenfunktionen implementiert werden, die das Verwenden einer Steuerfläche für mehrere Freiheitsgrade berücksichtigt. Die Ergebnisse des auf den Wert 42 skalierten dynamischen Allokator haben dieses Problem verdeutlicht.

Wird ein einseitigen Querruderausfall simuliert, werden das verbliebene Querruder und die effektivere der beiden Landeklappen als eine Gruppe aufgefasst. Dies erzeugt durch die höhere Ansteuerung der Landeklappen ebenfalls ein deutliches Überschwingen. Der Algorithmus ist dahingehend zu erweitern, dass die Gruppengröße nicht statisch vorgegeben, sondern dynamisch erkannt wird. So lassen sich auch Trägersysteme mit asymmetrischer und exotischer Steuerflächenkonfiguration steuern, ohne dass per se Schwingungen auftreten.

Zusammenfassend lässt sich sagen, dass sich das Steuern rein auf der einer gemessenen Effektivität als brauchbar, in der Form jedoch noch nicht als optimal erwiesen hat. Werden die beschrieben Problemstellungen und die Anforderung des kompletten Verzichts auf Expertenwissen (mit Ausnahme des Sonderfalls für die Motoren) betrachtet, liefert die eigene Umsetzung der Control Allocation jedoch bereits jetzt ein sehr gutes Ergebnis. Das System ist in der Lage, bei Ausfall einen sonst unausweichlichen Kontrollverlust und dadurch folgenden Absturz zu verhindern und ein kontrolliertes Weiterfliegen zu ermöglichen. Die Zielsetzung der Arbeit, den Autopiloten um eine Fehlertoleranz durch Adaptivität zu erweitern, erfüllt das realisierte System somit vollständig. Mit der Wahl des Ansatzes, hierfür zum Großteil auf lernende Methoden zu setzen und die Control Allocation generisch zu implementieren, wurde aus persönlicher Sicht die richtige Entscheidung getroffen.

A. Inhalte der Begleit-DVD

Die beiliegende DVD enthält die nachfolgend aufgeführten Daten und Artefakte. Die folgende Aufzählung spiegelt dabei die Verzeichnisstruktur wider:

- **MA_HasbergHagen_DynamischeAdaptiveFlugsteuerung.pdf**: PDF-Datei des vorliegenden Dokuments.
- **AeroSimRC**: Ausführbare Demoversion von AeroSimRC, sowie der Source Code des AeroSimRC Plugins. Enthält für die Evaluierung verwendeten Träger *AESL_Plane_CA*.
- **AESLink**: Utilities aus AESLink (Wiedergabe/Aufzeichnung/Visualisierung/Flugregler))
- **CSharp**: Source Code von Control Allocator, Pilot Simulator, Fault Detector und Fault Injector.
- **Exe**: Executables der C# Anwendungen.
- **Matlab**: Arbeitsumgebung zur Entwicklung der neuronalen Netze. Enthält einige persistierte Netze und die zur Evaluierung verwendeten Datensätze. Weiterhin sind die Skripte zur Erzeugung der in der Evaluierung präsentierten Diagramme und Auswertungen enthalten.

B. Verwendete Werkzeuge

Nachfolgend werden Werkzeuge und Programme aufgeführt, die in der Arbeit verwendet wurden.

- **AeroSimRC**: Version 4.3

  Flugsimulator zur Evaluierung der Realisierung

- **Matlab**: Version R2016a_x64, Neural Network Toolbox Version 9.0

  Entwicklungen: Neuronale Netze für Fault Detector. Auswertung und Plots der Evaluierung

- **Cortexsys**: Version 3.1

  LSTM-Toolbox für Matlab

- **Visual Studio**: Visual Studio 2010 Ultimate

  Entwicklungen (C++): Erweiterung des AeroSimRC AESLink-Plugins

  Entwicklungen (C#): Control Allocator, Pilot Simulator, Fault Detector, Erweiterung PacketSimulator

  Alle C#-Anwendungen verwenden das .NET Framework 2.0.

Literaturverzeichnis

[Abadi u. a. 2015] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dan ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. – URL http://tensorflow.org/. – Software available from tensorflow.org

[Barton 2012] Barton, Jeffrey D.: Fundamentals of Small Unmanned Aircraft Flight. In: *Johns Hopkins APL Technical Digest* 31 (2012), Oktober, Nr. 2, S. 132–149

[Behera und Rana 2014] Behera, Santosh K. ; Rana, Debaraj: System Identification Using Recurrent Neural Network. In: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* Bd. 3, IJAREEIE, March 2014, S. 8111–8117. – URL http://www.rroij.com/open-access/system-identification-using-recurrent-neuralnetwork.php?aid=42486

[Bengio u. a. 1994] Bengio, Y. ; Simard, P. ; Frasconi, P.: Learning Long-term Dependencies with Gradient Descent is Difficult. In: *Trans. Neur. Netw.* 5 (1994), März, Nr. 2, S. 157–166. – URL http://dx.doi.org/10.1109/72.279181. – ISSN 1045-9227

[Bengio 2012a] Bengio, Yoshua: Practical recommendations for gradient-based training of deep architectures. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7700 LECTURE NO (2012), S. 437–478. – ISBN 9783642352881

[Bengio 2012b] Bengio, Yoshua: Practical recommendations for gradient-based training of deep architectures. In: *CoRR* abs/1206.5533 (2012). – URL http://arxiv.org/abs/1206.5533

[Berg u. a. 1996] Berg, Jordan M. ; Hammett, Kelly D. ; Schwartz, Carla A. ; Banda, Siva S.: An analysis of the destabilizing effect of daisy chained rate-limited actuators. In: *IEEE Transactions on Control Systems Technology* 4 (1996), Nr. 2, S. 171–176. – ISSN 10636536

[Bosworth und Williams-Hayes 2007] Bosworth, John ; Williams-Hayes, Peggy: Flight Test Results from the NF-15B Intelligent Flight Control System (IFCS) Project with Adaptation to a Simulated Stabilator Failure. In: *AIAA Infotech@Aerospace 2007 Conference and Exhibit* (2007), Nr. December. – URL http://arc.aiaa.org/doi/10.2514/6.2007-2818. ISBN 978-1-62410-017-8

[Brockhaus u. a. 2011] Brockhaus, Rudolf ; Alles, Wolfgang ; Luckner, Robert: *Flugregelung*. 3. Springer, 2011

[Buffington und Enns 1996] Buffington, James ; Enns, Dale: Daisy chain control allocation - Lyapunov stability analysis. In: *Guidance, Navigation, and Control Conference* 19 (1996), Nr. 6. – URL http://arc.aiaa.org/doi/10.2514/6.1995-3341

[Burger 2016] Burger, Christopher: *Google DeepMind's AlphaGo: How it works*. Online. March 2016. – URL https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/. – [Stand: 06.05.18]

[Calise und Rysdyk 1998] Calise, A.J. ; Rysdyk, R.T.: Nonlinear adaptive flight control using neural networks. In: *IEEE Control Systems Magazine* 18 (1998), Dec, Nr. 6, S. 14–25. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=736008. – ISSN 1066-033X

[Calise u. a. 2004] Calise, Aj ; Shin, Yoonghyun ; Johnson, Md: A comparison study of classical and neural network based adaptive control of wing rock. In: ... *Guidance, Navigation and Control* ... (2004), Nr. August, S. 1–17. – URL http://arc.aiaa.org/doi/pdf/10.2514/6.2004-5320

[Casavola und Garone 2010] Casavola, Alessandro ; Garone, Emanuele: Fault-tolerant adaptive control allocation schemes for overactuated systems. In: *International Journal of Robust and Nonlinear Control* 20 (2010), Nr. 17, S. 1958–1980. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1561. – ISSN 1099-1239

[de Castro 2001] Castro, Helena de: *Flying and Handling Qualities of a Fly-By-Wire Blended-Wing-Body Civil Transport Aircraft*. 2001. – URL http://linkinghub.elsevier.com/retrieve/pii/S0738059301000165

[Connor u. a. 1992] Connor, Jerome T. ; Atlas, Les E. ; Martin, Douglas R.: Recurrent Networks and NARMA Modeling. In: *Advances in Neural Information Processing Systems 4* (1992), S. 301–308

[Cork u. a. 2005] Cork, Lennon R. ; Walker, Rodney A. ; Dunn, Shane: Fault Detection, Identification and Accommodation Techniques for Unmanned Airborne Vehicles. In: *Australian International Aerospace Congress*. Melbourne : Australian International Aerospace Congress (AIAC), 2005, S. 18. – URL https://eprints.qut.edu.au/1729/

[Cox 2016] Cox, Jonathan A.: *Cortexsys v. 3.0*. Mar 2016. – URL http://www.osti.gov/scitech/servlets/purl/1311620

[Cuéllar u. a. 2006] Cuéllar, M.P. ; Delgado, M. ; Pegalajar, M.C.: AN APPLICATION OF NON-LINEAR PROGRAMMING TO TRAIN RECURRENT NEURAL NETWORKS IN TIME SERIES PREDICTION PROBLEMS. In: Chen, Chin-Sheng (Hrsg.) ; Filipe, Joaquim (Hrsg.) ; Seruca, Isabel (Hrsg.) ; Cordeiro, José (Hrsg.): *Enterprise Information Systems VII*. Dordrecht : Springer Netherlands, 2006, S. 95–102. – ISBN 978-1-4020-5347-4

[Cybenko 1989] Cybenko, G.: Approximation by superpositions of a sigmoidal function. In: *Mathematics of Control, Signals and Systems* 2 (1989), Dec, Nr. 4, S. 303–314. – URL https://doi.org/10.1007/BF02551274. – ISSN 1435-568X

[Danke 2005] Danke, Kevin: *Flugerprobung mit einem BWB Modell*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, November 2005

[Drozeski 2005] Drozeski, Graham R.: *A fault-tolerant control architecture for unmanned aerial vehicles*, Georgia Institute of Technology, Dissertation, 2005

[Ducard 2009] Ducard, Guillaume J.: *Fault-tolerant Flight Control and Guidance Systems: Practical Methods for Small Unmanned Aerial Vehicles*. 1. Springer-Verlag London, 2009 (Advances in Industrial Control). – URL http://gen.lib.rus.ec/book/index.php?md5=590B155920199B72C86563E1045A1F32. – ISBN 9781441903037

[Ducard 2007] Ducard, Joseph Jacques G.: Fault-Tolerant Flight Control and Guidance Systems for a Small Unmanned Aerial Vehicle. In: *Springer Science & Business Media* (2007), Nr. 17505, S. 1–265

[Durham u. a. 2017] Durham, Wayne ; Bordignon, Kenneth A. ; Beck, Roger: *Aircraft Control Allocation*. 1. Wiley, 2017. – ISBN 978-1-118-82779-6

[Efimov u. a. 2013] Efimov, Denis ; Cieslak, Jérôme ; Zolghadri, Ali ; Henry, David: Actuator fault detection in aircraft systems: Oscillatory failure case study. In: *Annual Reviews in Control* 37 (2013), Nr. 1, S. 180–190. – URL http://dx.doi.org/10.1016/j.arcontrol.2013.04.007. – ISSN 13675788

[Elman 1990] Elman, Jeffrey L.: Finding Structure in Time. In: *Cognitive Science* 14 (1990), Nr. 2, S. 179–211. – ISSN 1551-6709

[Fekih u. a. 2007] Fekih, A ; Xu, H ; Chowdhury, F N.: Neural networks based system identification techniques for model based fault detection of nonlinear systems. In: *International Journal of Innovative Computing Information and Control* 3 (2007), Nr. 5, S. 1073–1085

[Frost u. a. 2010] Frost, Susan ; Taylor, Brian ; Jutte, Christine ; Burken, John ; Trinh, Khanh ; Bodson, Marc: A Framework for Optimal Control Allocation with Structural Load Constraints. In: *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, August 2010, S. –. – URL http://dx.doi.org/10.2514/6.2010-8112

[Funahashi und Nakamura 1993] Funahashi, K ; Nakamura, Y: Approximation of dynamical systems by continuous time recurrent neural networks. In: *Neural Networks* 6 (1993), Nr. 6, S. 801–806. – URL http://linkinghub.elsevier.com/retrieve/pii/S089360800580125X. – ISBN 0893-6080

[Grundmann 2007] Grundmann, Prof. Dr.-Ing. R.: *Flugmechanik*. 2007. – Vorlesungsskript

[Gundy-Burlet 2004] Gundy-Burlet, Karen: Augmentation of an Intelligent Flight Control System for a Simulated C-17 Aircraft. In: *Journal of Aerospace Computing, Information, and Communication* 1 (2004), Nr. 12, S. 526–542. – URL http://arc.aiaa.org/doi/10.2514/1.13246. – ISSN 1542-9423

[Gundy-Burlet und Bryant 2003] Gundy-Burlet, Karen ; Bryant, Don: Control Reallocation Strategies for Damage Adaptation in Transport Class Aircraft. (2003), Nr. August, S. 1–10. ISBN 9781563479786

[Härkegård und Glad 2005] Härkegård, Ola ; Glad, S. T.: Resolving actuator redundancy - Optimal control vs. control allocation. In: *Automatica* 41 (2005), Nr. 1, S. 137–144. – ISBN 0005-1098

[Hasberg 2014] Hasberg, Hagen: *Ein Testkonzept für Flugregler.* Bachelorthesis. June 2014. – URL http://edoc.sub.uni-hamburg.de/haw/volltexte/2014/2614/

[Hasberg 2017a] Hasberg, Hagen: *Recurrent Neural Networks zur Fault Detection von Steuerflächen.* Grundprojekt. August 2017

[Hasberg 2017b] Hasberg, Hagen: *Trainingsdatenerzeugung und Fault Injection für eine adaptive Flugsteuerung.* Hauptprojekt. Oktober 2017

[Heaton 2015] Heaton, Jeff: *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks.* CreateSpace Independent Publishing Platform, 2015. – URL http://www.heatonresearch.com/book/aifh-vol3-deep-neural.html. – ISBN 1505714346

[Hinton u. a. 2006] Hinton, Geoffrey E. ; Osindero, Simon ; Teh, Yee-Whye: A Fast Learning Algorithm for Deep Belief Nets. In: *Neural Computation* 18 (2006), Nr. 7, S. 1527–1554. – URL https://doi.org/10.1162/neco.2006.18.7.1527. – PMID: 16764513

[Hochreiter und Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), 12, Nr. 8, S. 1735–1780. – URL http://dx.doi.org/10.1162/neco.1997.9.8.1735

[Horvath 2002] Horvath, G.: Neural networks in system identification. In: *Neural Networks in Measurement Systems* 185 (2002), S. 43–78

[Hovakimyan u. a. 2001] Hovakimyan, Naira ; Rysdyk, Rolf ; Calise, Anthony J.: Dynamic neural networks for output feedback control. In: *International Journal of Robust and Nonlinear Control* 11 (2001), Nr. 1, S. 23–39. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/1099-1239%28200101%2911%3A1%3C23%3A%3AAID-RNC545%3E3.0.CO%3B2-N

[Ippolito u. a. 2007] Ippolito, Corey ; Yeh, Yoo-Hsiu ; Kaneshige, John: Neural Adaptive Flight Control Testing on an Unmanned Experimental Aerial Vehicle. In: *AIAA Infotech at Aerospace 2007 Conference and Exhibit* (2007), S. 2827. ISBN 1563478935 | 9781563478932

[Johansen 2004] Johansen, TA: Optimizing nonlinear control allocation. In: *Decision and Control, 2004. CDC. 43rd IEEE ...* 4 (2004), Nr. 4, S. 3435–3440 Vol.4. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?

arnumber=1429240{%}5Cnhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1429240. – ISBN 0-7803-8682-5

[Johansen und Fossen 2013] Johansen, Tor A. ; Fossen, Thor I.: Control allocation - A survey. In: *Automatica* 49 (2013), Nr. 5, S. 1087 – 1103. – URL http://www.sciencedirect.com/science/article/pii/S0005109813000368. – ISSN 0005-1098

[Karpathy 2015] Karpathy, Andrej: *The Unreasonable Effectiveness of Recurrent Neural Networks*. Online Blog. 05 2015. – URL https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[Kim und Calise 1997] Kim, Byoung S. ; Calise, Anthony J.: Nonlinear Flight Control Using Neural Networks. In: *{AIAA} Journal of Guidance, Control, and Dynamics* 20 (1997), Nr. 1, S. 26–33. – URL http://arc.aiaa.org/doi/10.2514/2.4029. – ISSN 07315090

[Kim u. a. 2013] Kim, Jiyeon ; Yang, Inseok ; Lee, Dongik: Daisy Chain Method for Control Allocation Based Fault-Tolerant Control. 8 (2013), Nr. 5, S. 265–272

[Lecun u. a. 1998] Lecun, Yann ; Bottou, Leon ; Orr, Genevieve B. ; Müller, Klaus-Robert: *Efficient BackProp*. 1998

[McLean 1990] McLean, Donald: *Automatic Flight Control Systems*. 1. Prentice-Hall, 1990

[Meisel 2008] Meisel, Andreas: *Künstliche neuronale netze*. 2008

[Meisel 2017] Meisel, Prof. Dr.-Ing. A.: *Robot Vision*. 2017. – Vorlesungsskript

[Müller 2008] Müller, Prof. Dr.-Ing. K.: *Regelungstheorie 2*. 2008. – Vorlesungsskript

[NASA 2014] NASA: *NASA Armstrong Fact Sheet - Intelligent Flight Control System*. Online. 2014. – URL https://www.nasa.gov/centers/armstrong/news/FactSheets/FS-076-DFRC.html. – [Stand: 21.05.2018]

[National Transportation Safety Board 1990] National Transportation Safety Board: United Airlines Flight 232, McDonnell Douglas DC-10-10, N1819U. In: *Aircraft Accident Report* (1990), S. 126

[Nguewo 2007] Nguewo, Danyck: *Erstellung und Optimierung der Skalierungsgesetze zur Abschätzung der Aerodynamik und der Eigendynamik eines Flugzeugs auf der Basis von frei fliegenden Modellen*, Universität Stuttgart, Dissertation, 2007

[Nguewo 2014] Nguewo, Danyck: *Flugmechanik 2*. 2014. – Vorlesungsskript WS13/14

[Olah 2015] Olah, Christopher: *Understanding LSTM Networks*. Online Blog. 08 2015. – URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[OpenAI 2017a] OpenAI: *Dota 2*. Online. August 2017. – URL https://blog.openai.com/dota-2/. – [Stand: 06.05.18]

[OpenAI 2017b] OpenAI: *More on Dota 2*. Online. August 2017. – URL https://blog.openai.com/more-on-dota-2/. – [Stand: 06.05.18]

[Park und Rokhsaz 2003] Park, Paul ; Rokhsaz, Kamran: Effects of a Winglet Rudder on Lift-to-Drag Ratio and Wake Vortex Frequency. In: *Fluid Dynamics and Co-located Conferences*. American Institute of Aeronautics and Astronautics, Juni 2003, S. –. – URL http://dx.doi.org/10.2514/6.2003-4069

[Pascanu u. a. 2012] Pascanu, Razvan ; Mikolov, Tomas ; Bengio, Yoshua: On the difficulty of training Recurrent Neural Networks. (2012), Nr. 2. – URL http://arxiv.org/abs/1211.5063. – ISBN 08997667 (ISSN)

[Patan 2008] Patan, Krzysztof: *Artificial Neural Networks for the Modelling and Fault Diagnosis of Technical Processes*. Springer, 2008. – 377 S. – URL http://link.springer.com/10.1007/978-3-540-79872-9. – ISBN 978-3-540-79871-2

[Pedregosa 2011] Pedregosa, F. et. a.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[Peng u. a. 2017] Peng, Xue B. ; Berseth, Glen ; Yin, KangKang ; Panne, Michiel van de: DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)* 36 (2017), Nr. 4

[Reimann 2004] Reimann, Holger: Vergleich unkonventioneller Flugzeugkonfigurationen / Universität der Bundeswehr München. August 2004. – Studienarbeit. Studienarbeit

[Schneider u. a. 2012] Schneider, Thomas ; Ducard, Guillaume ; Rudin, Konrad ; Strupler, Pascal: Fault-tolerant Control Allocation for Multirotor Helicopters using Parametric Programming. In: *International Micro Air Vehicle Conference and Flight Competition* (2012), Nr. July 2012

[Schulz u. a. 2017a] Schulz, Stephan ; Büscher, René ; Hasberg, Hagen: *Scalable FPGA-based parallel Flight Controller for SUAV*. 2017. – Manuscript in preparation

[Schulz u. a. 2017b] Schulz, Stephan ; Hasberg, Hagen ; Büscher, René: *Hardware-in-the-loop Testbed for Autopilot Development using Flight Simulation and Parallel Kinematics*. 2017. – Manuscript in preparation

[Schumann u. a. 2003] Schumann, Johann ; Gupta, Pramod ; Nelson, Stacy: On verification & validation of neural network based controllers. In: *Proc. of International Conf on Engineering Applications of Neural Networks, EANN* 3 (2003), S. 401–47. – URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.155.4718{&}rep=rep1{&}type=pdf

[Shan 2016] Shan, Shangqiu: Neural Network NARMAX Model Based Unmanned Aircraft Control Surface Reconfiguration. (2016), Nr. 2. ISBN 9781509035588

[Shin u. a. 2006] Shin, Jong-Yeob ; Belcastro, Christine M. ; Khong, Thuan H.: *Closed-Loop Evaluation of An Integrated Failure Identification And Fault Tolerant Control System for A Transport Aircraft*. 2006

[Silver u. a. 2016] Silver, David ; Huang, Aja ; Maddison, Chris J. ; Guez, Arthur ; Sifre, Laurent ; Van Den Driessche, George ; Schrittwieser, Julian ; Antonoglou, Ioannis ; Panneershelvam, Veda ; Lanctot, Marc ; Dieleman, Sander ; Grewe, Dominik ; Nham, John ; Kalchbrenner, Nal ; Sutskever, Ilya ; Lillicrap, Timothy ; Leach, Madeleine ; Kavukcuoglu, Koray ; Graepel, Thore ; Hassabis, Demis: Mastering the game of Go with deep neural networks and tree search. In: *Nature* 529 (2016), Nr. 7587, S. 484–489. – URL http://dx.doi.org/10.1038/nature16961. – ISBN 1476-4687 (Electronic)\r0028-0836 (Linking)

[Silvestri u. a. 1994] Silvestri, G. ; Verona, F.Bini ; Innocenti, M. ; Napolitano, M.: Fault detection using neural networks. In: *IEEE International Conference on Neural Networks - Conference Proceedings* 6 (1994), Nr. January 2018

[Smaili u. a. 2009] Smaili, H M. ; Breeman, J ; Lombaerts, T J J. ; Joosten, D a.: *Fault Tolerant Flight Control - A Benchmark Challenge*. 2009. – 171–212 S. – ISBN 185233410X

[Soares u. a. 2006] Soares, F ; Burken, J ; Marwala, T: Neural network applications in advanced aircraft flight control system, a hybrid system, a flight test demonstration. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture*

Notes in Bioinformatics) 4234 LNCS (2006), S. 684–691. – URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-33750722416{&}partnerID=40{&}md5=0f1e6f2ea56e4555ede54b32bad0b2e6

[The MathWorks, Inc. 2017] The MathWorks, Inc.: *Neural Network Toolbox*. 2017. – URL https://de.mathworks.com/products/neural-network.html. – [Stand: 26.07.2017]

[Trischler und D'Eleuterio 2015] Trischler, Adam P. ; D'Eleuterio, Gabriele M. T.: Synthesis of recurrent neural networks for dynamical system simulation. In: *CoRR* abs/1512.05702 (2015). – URL http://arxiv.org/abs/1512.05702

[Uhlig u. a. 2010] Uhlig, Daniel V. ; Selig, Michael S. ; Neogi, Natasha: Health Monitoring via Neural Networks. In: *Neural Networks* (2010), Nr. April, S. 12

[Wagner 2018] Wagner, Stefan S.: *Entwicklung eines Reinforcement Learning basierten Flugzeugautopiloten unter der Verwendung von Deterministic Policy Gradients*. Bachelorthesis. February 2018

[Williams-Hayes 2005] Williams-Hayes, PS: Flight test implementation of a second generation intelligent flight control system. In: *infotech@ Aerospace, AIAA-2005-6995* (2005), Nr. November. – URL http://arc.aiaa.org/doi/pdf/10.2514/6.2005-6995. ISBN 1563477394

[Yu u. a. 2014] Yu, Dong ; Eversole, Adam ; Seltzer, Mike ; Yao, Kaisheng ; Kuchaiev, Oleksii ; Zhang, Yu ; Seide, Frank ; Huang, Zhiheng ; Guenter, Brian ; Wang, Huaming ; Droppo, Jasha ; Zweig, Geoffrey ; Rossbach, Chris ; Gao, Jie ; Stolcke, Andreas ; Currey, Jon ; Slaney, Malcolm ; Chen, Guoguo ; Agarwal, Amit ; Basoglu, Chris ; Padmilac, Marko ; Kamenev, Alexey ; Ivanov, Vladimir ; Cypher, Scott ; Parthasarathi, Hari ; Mitra, Bhaskar ; Peng, Baolin ; Huang, Xuedong: An Introduction to Computational Networks and the Computational Network Toolkit / Microsoft Research. Microsoft Research, October 2014. – Forschungsbericht. – URL https://goo.gl/TrnpL2. An Introduction to Computational Networks and the Computational Network Toolkit

[Zhang und Zhang 2009] Zhang, Zhengdao ; Zhang, Weihua: Neural network based fault detection and identification for fighter control surface failure. In: *2009 Chinese Control and Decision Conference* (2009), S. 5256–5261. ISBN 9781424427239

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 30. Mai 2018 Hagen Hasberg