Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Hagen Hasberg

Ein Testkonzept für Flugregler

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hagen Hasberg

Ein Testkonzept für Flugregler

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Thomas Lehmann
Zweitgutachter : Prof. Dr.-Ing. Thomas Netzel

Abgegeben am 10. Juni 2014

Hagen Hasberg

Thema der Bachelorthesis

Ein Testkonzept für Flugregler

Stichworte

Flugregler, Autopilot, Simulation, Avionik, Echtzeit, Hardware-In-the-Loop, realitätsnah, Flugsimulator, UAV, Simulink

Kurzzusammenfassung

Gegenstand dieser Arbeit ist die Konzeptionierung und Realisierung einer Infrastruktur zum Entwickeln und Testen von Flugreglern für unbemannte Flugzeuge. Die Infrastruktur soll eine realitätsnahe Simulationsumgebung auf Basis von PC-Flugsimulatoren darstellen. Zunächst werden Konzepte vergleichbarer Systeme evaluiert und geeignete Flugsimulatoren ausgewählt. Auf Basis dieser Recherche werden Anforderungen an die Infrastruktur definiert. Anschließend wird aus den Anforderungen ein Konzept entworfen und präsentiert, welches daraufhin realisiert wird. Abschließend wird das umgesetzte System evaluiert und auf Brauchbarkeit analysiert.

Hagen Hasberg

Title of the paper

A test concept for flight controllers

Keywords

flight controller, autopilot, simulation, avionics, real-time, hardware-in-the-loop, realistic, flight simulator, UAV, Simulink

Abstract

The subject of this thesis is the conception and realization of an infrastructure for the development and testing of automatic flight control systems for unmanned aircrafts. The insfrastructure shall constitute a realistic simulation environment based on a pc flight simulator software. First, concepts from similar systems are evaluated and suitable flight simulators are chosen. Based on these investigations, the requirements for the infrastructure are defined. Afterwards, a concept that covers the requirements is designed and presented, followed by the implementation of the concept. Finally, the implemented system is evaluated and the general usability is analyzed.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Flugregelung hat schon wenige Jahre nachdem die Gebrüder Wright ihre ersten erfolgreichen Flugversuche durchgeführt haben ihren Weg in die Luftfahrt gefunden. Der erste durch einen Regler unterstützte Flug wurde der Öffentlichkeit bereits 1914 durch Lawrence Sperry vorgestellt [PopularScienceMonthly (1930)]. In den vergangenen 100 Jahren haben sich Flugregler immens weiterentwickelt. Die Flugregelung ist heutzutage nicht mehr aus Flugzeugen und anderen Fluggeräten wegzudenken.

Zum Testen von Flugreglern und Algorithmen zur autonomen Steuerung eines Flugzeugs eignen sich sogenannte *Model-In-the-Loop*- (*MIL*) und *Hardware-In-the-Loop*- (*HIL*) Tests. Bei MIL-Tests sind die Flugregler als mathematisches Modell vorhanden und werden beispielsweise in Simulink ausgeführt. Bei HIL-Tests hingegen sind die Flugregler auf der eingebetteten Hardware implementiert, die das Flugzeug autonom steuern soll. Das Flugzeug muss in beiden Fällen simuliert werden, wozu ein ausführbares Modell des physikalischen Flugverhaltens erforderlich ist.

Am Markt verfügbare PC-Flugsimulatoren, wie zum Beispiel X-Plane, enthalten Simulationsmodelle, die das Bewegungsverhalten eines Flugzeugs nachbilden. Weiterhin visualisieren die Flugsimulatoren das Trägersystem in einer real wirkenden Umgebung. Das Simulationsmodell eines Flugsimulators kann als Basis für den Entwurf von MIL- und HIL-Tests dienen.

Die vorliegende Arbeit bringt Aspekte der Flugregelung und Simulation zusammen. Sie beschäftigt sich mit der Konzeptionierung und Entwicklung einer Test- und Simulationsinfrastruktur für Flugregler.

1.1. Organisatorisches Umfeld

Im Jahr 2001 wurde die Projektgruppe *Blended Wing Body* (*BWB*) an der HAW Hamburg gegründet[1]. Seit jeher befassen sich Studierende aus dem Department Fahrzeugtechnik und Flugzeugbau mit der Erforschung von unkonventionellen BWB Flugzeugkonfigurationen [Reimann (2004)], welche sich langfristig als neuer Standard für Großraumflugzeuge etablieren könnten. In der konventionellen Bauweise ist mit dem Airbus A380 bereits jetzt die Grenze des

[1] Inzwischen "*Projekt BWB AC20.30*", www.ac2030.de

Machbaren erreicht. Steigende Passagierzahlen verlangen auf absehbare Zeit jedoch noch größere Passagierflugzeuge. Weiterhin erhofft sich die Fachwelt nach aktuellem Stand einen bis zu 30% verringerten Treibstoffverbrauch gegenüber konventionellen Verkehrsflugzeugen [Liebeck (2004), VELA].

Die Forschungsschwerpunkte des BWB-Teams sind unter anderem der Entwurf von Passagier- und Frachtkabinen, sowie die Analyse der Flugleistungen und der Flugdynamik einer BWB Konfiguration. Das Gebiet der Blended Wing Body Flugzeuge ist noch relativ unerforscht. Aus diesem Grund hat das BWB-Team zur Analyse und Demonstration der flugmechanischen und aerodynamischen Eigenschaften einer solchen Konfiguration bereits mehrere flugfähige Versuchsträger konstruiert und gebaut. Die Entwürfe der Träger stammen aus einem internationalen Forschungsprojekt mit dem Namen *VELA 2* (*Very Efficient Large Aircraft 2*) [VELA].

Die erste Version des Trägers mit dem Namen *AC2030* wurde im Jahr 2003 gebaut und hatte seinen Erstflug im Dezember desselben Jahres. Das darin verwendete Messsystem lieferte nur unbefriedigende Ergebnisse [vgl. Danke (2005), Seite 68]. Danke bemängelt außerdem das Fehlen eines Autopiloten, der unter anderem selbständig die Flughöhe oder Fluggeschwindigkeit beibehält. Er erhofft sich durch einen autonom gesteuerten Flug genauere Messungen und dadurch bessere Aussagekraft über die Flugeigenschaften des AC2030 [vgl. Danke (2005), Seite 87].

Die zweite Version wurde im Jahr 2008 gebaut. Abbildung 1.1 zeigt diese bei einem Flugversuch im August 2013. Für die zweite Version wurde ein neues Messsystem (*Flugdatenlogger*, *FDL*) angeschafft. Dieses wurde im Auftrag der Universität Stuttgart von der Firma SFL-GmbH entwickelt und gefertigt. Die Universität Stuttgart verwendete das Messsystem in ihrem eigenen Demonstrator-Modell, dass ebenfalls auf dem Entwurf VELA 2 basiert [Nguewo (2007)]. Die Ähnlichkeit beider Versuchsträger ermöglichte die Wiederverwendung des Messsystems im AC2030 ohne großen Änderungsaufwand. Dem FDL fehlt allerdings ebenfalls die Funktion eines Autopiloten.

Verschiedene Faktoren und neue Anforderungen führten zu dem Entschluss, in den kommenden Semestern eine dritte Version des Versuchsträgers zu bauen. Nachfolgend sind maßgebliche Faktoren aufgeführt:

- Einsatz des AC2030 als autonome Drohne (*UAV*), zum Beispiel für Einsätze im Katastrophenschutz.
- Verringerung des Gewichts.
- Anhaltende Probleme und Ausfälle der zweiten Version.
- Umsetzung gewonnener Erkenntnisse aus früheren Versionen.
- Verwendung neuer Systeme und Werkstoffe.

Abbildung 1.1.: 1:30 Modell des AC2030 bei einem Flugversuch im August 2013

Für die dritte Version soll ein neues Messsystem (*Flight Control Unit*, *FCU*) inklusive autonomer Flugsteuerung vom Department Informatik entwickelt werden. Hierfür wurde im Sommersemester 2013 das Projekt *Airborne Embedded Systems* (*AES*) gegründet. Die AES Projektgruppe beschäftigt sich mit der Entwicklung und der Fertigung der FCU, sowie der dafür erforderlichen Systeme. Die vorliegende Arbeit ist im Kontext von AES entstanden und behandelt den Entwurf und die Entwicklung einer Testinfrastruktur für die FCU und der zu entwickelnden Flugregler zur autonomen Steuerung des AC2030.

1.2. Motivation

Für die Entwicklung von sicherheitskritischen Systemen sind frühzeitige Maßnahmen zur Verifikation der Systeme unerlässlich. Ausgiebige Tests der Einzelkomponenten sowie des Gesamtsystems helfen dabei, Personen- und Sachschäden durch unausgereifte Produkte zu vermeiden bzw. zu minimieren. Als besonderer Faktor kommt hinzu, dass sich das Department Informatik zuvor nicht mit der Entwicklung von eingebetteter Hardware zur autonomen Steue-

rung von Flugzeugen beschäftigt hat und somit erst Erfahrung mit solchen Systemen aufbauen muss. Dieser Umstand hebt die enorme Wichtigkeit von frühzeitigen Tests ebenfalls an.

Ein Vorteil der angestrebten *test-first* Mentalität ist die zu erwartende Kostenersparnis, da Fehler in Entwurf und Umsetzung frühzeitig entdeckt und behoben werden können. Das folgende Zitat aus der Ardupilot Community soll diese Aussage verdeutlichen:

> "*Crashing software planes is a lot cheaper than crashing real ones!*"[2]

Unfälle sind jedoch nicht der einzige Kostenfaktor bei Flugversuchen. Zur Durchführung wird neben Geld auch Zeit benötigt. Ein vom BWB-Team organisierter Flugtag verursacht typischerweise die folgenden Aufwendungen in Zeit und Geld:

- Vorzeitige Planungsphase
- Anmeldung am Flugfeld
- Kraftstoffkosten für die Fahrt zum Flugfeld
- Beladen und Entladen eines Anhängers
- Preflight Checks im Labor
- Einladen eines externen Piloten

Weiterhin muss eine gewisse Anzahl von Studierenden zur Durchführung beim Flugtag anwesend sein. Sollte sich beim Flugversuch herausstellen, dass die FCU durch einen Fehler keine Messwerte aufzeichnen kann, bleibt nichts anderes übrig als den Flugtag ohne Ergebnisse abzubrechen. Die Aufwendungen und Mühen der Studierenden wären in so einem Fall wertlos, was sich negativ auf die Moral der Beteiligten auswirkt.

Realitätsnahe Simulationen der Umgebung eines Systems ermöglichen es, das System auszulegen, ohne dass es sich in seiner realen Umgebung befinden muss. Besonders bei sicherheitskritischen Systemen ist das Entwickeln in der realen Umgebung teuer und gefährlich. Viele Teilsysteme eines Flugzeugs werden heutzutage mit Hilfe von Simulationen entwickelt und ausgelegt. Am 14.06.2013 fand der Jungfernflug der MSN1 des Airbus A350 XWB statt. Der kommandierende Testpilot Peter Chandler lobte die gute Vorarbeit der Ingenieure wie folgt [Chandler (2013), 05:21':30"]:

> "*I think the biggest compliment I can give to what we did on the flight is that, after the first few minutes, it didn't feel like we were doing a first flight, it felt like we would fly an airplane at the end of a test program, not at the beginning of a test program. It was so relaxed and so predictable. And I think it's the best I can say for the airplane.*"

[2] ArduPilot - Code testing with Simulation, http://dev.ardupilot.com/wiki/simulation-2/ [Stand: 30.05.2014]

Seine Aussage unterstreicht, wie gut sich realitätsnahe Simulationen zur kostengünstigen Entwicklung eines Flugzeugs und insbesondere der zu entwickelnden Flugregler eignen.

Das frühzeitige Testen trägt zur Sicherheit von Mensch und Maschine, sowie zur Reduktion von Kosten und Zeitaufwänden bei Entwicklung und Verwendung des Systems bei. Simulationen bieten die Möglichkeit, Entwicklungen, sowie Tests in einer risikofreien Umgebung durchzuführen. Die vorliegende Arbeit soll Möglichkeiten und Konzepte aufzeigen und umsetzen, die das frühzeitige Testen und Entwickeln durch Simulationen ermöglichen.

1.3. Aufgabenstellung

Die Aufgabe dieser Thesis besteht darin, eine Infrastruktur auf Basis eines Flugsimulators für eine MIL- und HIL-Simulation zu entwickeln, die mit Simulink bzw. der FCU interagieren kann. Die MIL-Simulation soll dazu dienen, den Entwicklungsaufwand der Flugregler und Algorithmen erheblich zu verkürzen und zu vereinfachen. Die HIL-Simulation soll der Verifikation der implementierten Regler dienen.

Die Abbildungen 1.2 und 1.3 zeigen jeweils Skizzen der MIL- und HIL-Simulation. Die MIL-Simulation läuft vollständig auf einem PC ab. Sie enthält den Flugsimulator, Simulink, sowie Möglichkeiten Piloteneingaben zu berücksichtigen. Die HIL-Simulation läuft nur teilweise auf dem PC ab. Die Flugregler werden hier nicht länger in Simulink, sondern direkt auf der FCU ausgeführt. Die FCU ist eine eingebettete Hardware mit speziellen Schnittstellen. Daher müssen physikalische Umsetzungen von PC- auf FCU-Schnittstelle und umgekehrt durchgeführt werden.

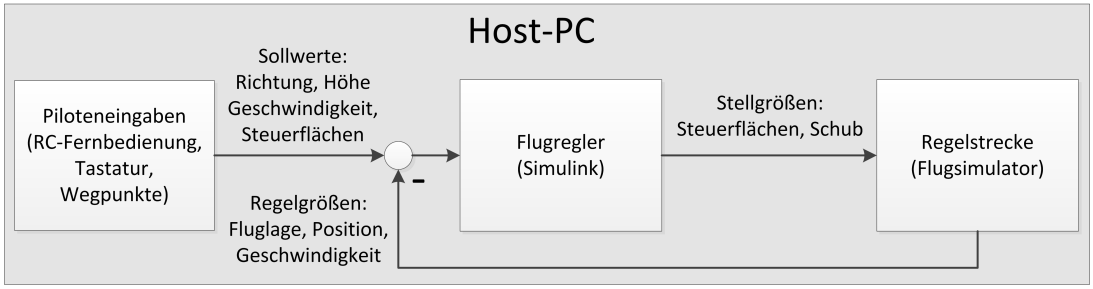

Abbildung 1.2.: Skizze der Model-In-the-Loop-Simulation

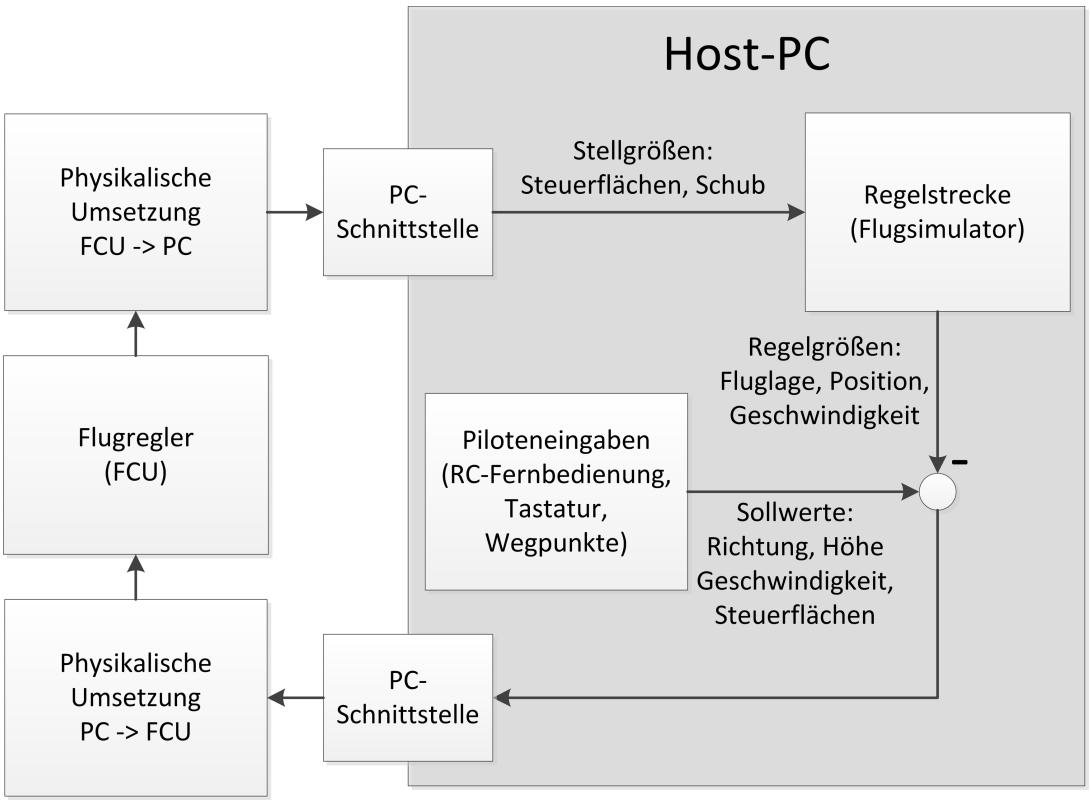

Abbildung 1.3.: Skizze der Hardware-In-the-Loop-Simulation

Teilaufgaben

Im ersten Teil dieser Arbeit sollen ähnliche Lösungen und Vorgehensweisen von Open Source Projekten und anderen Quellen recherchiert und evaluiert werden. Die gefundenen Ansätze sind auf Übertragbarkeit auf die gegebene Problemstellung zu prüfen. Weiterhin sind geeignete Flugsimulatoren zu identifizieren. Basierend auf den Recherchen sollen die Anforderungen an die Testinfrastruktur erfasst und dokumentiert werden. Daraufhin soll ein geeigneter Flugsimulator für die Umsetzung der Aufgabenstellung ausgewählt werden. Hierbei steht eine möglichst realistische Flugphysik im Vordergrund. Der grafische Detailgrad der simulierten Welt kann vernachlässigt werden.

Im zweiten Schritt sollen mit dem gesammelten Wissen und den erstellten Anforderungen ein Konzept für die Kommunikation der Komponenten untereinander entworfen werden. Dieses beinhaltet neben der Art der Kommunikation auch die Datensätze, die zwischen Flugregler

und Flugsimulator ausgetauscht werden. Des Weiteren sollen Komponenten identifiziert und spezifiziert werden, die eine Entwicklung der Regler vereinfachen.

Im dritten Teil der Arbeit werden die Konzepte gemäß den Anforderungen umgesetzt und implementiert. Zur Verifikation der gesamten Infrastruktur wird ein vereinfachter Regler entworfen. Dieser soll sowohl in Simulink modelliert, sowie auch in der FCU implementiert werden. Ziel dabei ist nicht der Entwurf eines optimalen Reglers, sondern die Verifikation und Validierung der Kopplungen der Komponenten innerhalb der Infrastruktur.

Im letzten Arbeitsschritt werden die Ergebnisse der Arbeit dargestellt. Hier sollen besonders Aussagen über den Realitätsgrad der Umgebung für den Flugregler in Simulink und für die Implementierung auf der FCU getroffen werden.

1.4. Gliederung der Thesis

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

Die Einleitung macht den Leser mit dem Umfeld und der gegebenen Aufgabenstellung vertraut.

Das zweite Kapitel gibt dem Leser ein Grundwissen über verschiedene Aspekte der Aufgabenstellung, so dass er die einzelnen Arbeitsschritte nachvollziehen kann.

Im dritten Kapitel werden ähnliche Lösungen und Vorgehensweisen von Open Source Projekten und anderen Quellen dargestellt. Weiterhin werden hier geeignete Flugsimulatoren beschrieben.

Basierend auf den Recherchen werden im vierten Kapitel Anforderungen an die zu entwickelnde Infrastruktur aufgestellt.

In Kapitel fünf wird das aus den Anforderungen und Recherchen erarbeitete Konzept vorgestellt.

Das sechste Kapitel beschreibt die Realisierung des Konzepts. Dabei werden die Infrastruktur, sowie die einzelnen Teilkomponenten beschrieben.

Im siebten Kapitel werden die Ergebnisse präsentiert, welche einen Aufschluss über die Brauchbarkeit des entwickelten Systems geben.

Das achte Kapitel gibt einen Ausblick über ausstehende Arbeiten.

Im neunten und letzten Kapitel wird abschließend ein Fazit gezogen.

2. Grundlagen und Definitionen

In diesem Kapitel werden einige Grundlagen der verwendeten Konzepte und Begriffe, die in der vorliegenden Arbeit Anwendung fanden, beschrieben. Diese Grundsteine sollen dem Leser eine solide Basis für das Verständnis der darauffolgenden Kapitel geben. Neben den Grundlagen werden hier auch verwendete Begrifflichkeiten definiert.

2.1. AC2030 und FCU

Dieser Abschnitt beschreibt den technischen Systemkontext der Arbeit. Es werden Begriffe erläutert, sowie die FCU beschrieben.

Begriffe

Zunächst werden einige ausgewählte Begriffe aus dem Arbeitsumfeld erläutert.

- **Unmanned Aerial Vehicle (UAV)**: *Unbemanntes Luftfahrzeug*, welches ohne Besatzung autonom navigieren und agieren kann. Zu einem UAV gehört der Träger, die Nutzlast (z.B. Kamera), sowie alle weiteren Komponenten, die sich im Träger befinden. Ein unbemanntes Luftfahrzeug wird umgangssprachlich auch *Drohne* genannt.
- **Ground Control Station (GCS)**: *Bodenkontrollstation*, mit der ein UAV vom Boden aus überwacht, kontrolliert und gesteuert werden kann. Im Kontext der Arbeit ist hiermit eine PC Anwendung gemeint.
- **Unmanned Aerial System (UAS)**: Das Gesamtsystem aus UAV, GCS, sowie weiterer für den Betrieb benötigter Teilsysteme (Funkverbindung, Startsystem, etc.).

Barton hat eine ausführliche Einleitung zu UAVs und dazugehörigen Systemen zusammengestellt [Barton (2012)].

Achsensystem eines Flugzeugs

Für die vorliegende Arbeit gilt das folgende Achsensystem für Flugzeuge (körperfestes Koordinatensystem):

- **Ursprung**: Der *Ursprung* liegt im Schwerpunkt des Flugzeugs.
- **Längsachse (positiv nach vorne)**: Die *Längsachse* verläuft entlang der *Symmetrieachse*. Das Drehen um die Längsachse wird *Rollen* (engl. *roll*) genannt.
- **Querachse (positiv nach rechts)**: Die *Querachse* verläuft (in der Ebene) im rechten Winkel zur Längsachse. Das Drehen um diese Achse wird *Nicken* (engl. *pitch*) genannt.
- **Hochachse (positiv nach oben)**: Die *Hochachse* bildet zusammen mit Längsachse und Querachse ein Rechtssystem. Eine Drehung wird *Gieren* (engl. *yaw*) genannt.

Hardwareumgebung der FCU

Die FCU ist Teil eines *Flight Control Computers*, welcher mehrere Hardwarekomponenten enthält. Jedoch können einige der Hardwarekomponenten vernachlässigt werden, da der Fokus der vorliegenden Arbeit auf Tests der implementierten Flugregler liegt.

Die FCU hat verschiedene Ein- und Ausgänge, die in Abbildung 2.1 dargestellt sind. Die Eingänge bilden drei unterschiedliche Sensoren (*Sensorpalette*), sowie die Steuereingaben des Piloten. Die Ausgänge bilden die vom Flugregler berechneten Steuersignale. Nachfolgend werden die einzelnen Komponenten erläutert.

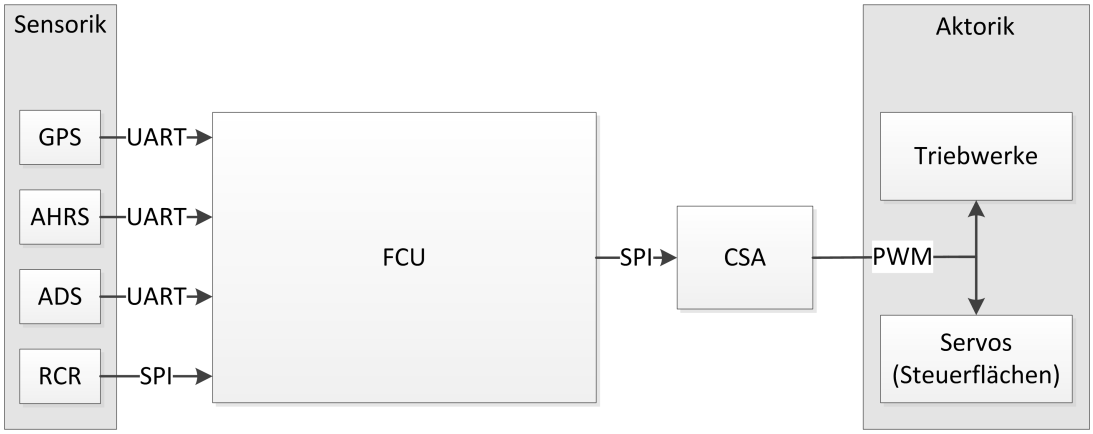

Abbildung 2.1.: Hardwareumgebung der FCU. Dargestellt sind Sensorik, Aktorik, sowie die physikalischen Schnittstellen.

GPS
Ein GPS Sensor liefert Daten über die Position, sowie Geschwindigkeiten des Flugzeugs. Er ist ein handelsüblicher GPS Sensor, wie er auch in Navigationsgeräten oder Smartphones zum Einsatz kommt. Seine Datenrate beträgt 5 Hz. Physikalisch ist er über eine UART Schnittstelle verbunden. Er wird vorwiegend zur autonomen Navigation über definierte Wegpunkte eingesetzt.

AHRS
Das Attitude Heading Reference System ist ein inertiales Messsystem zur Messung von Fluglage, Drehraten und Beschleunigungen des Flugzeugs im körperfesten Koordinatensystem. Der AHRS Sensor liefert seine Daten mit 50 Hz über eine UART Schnittstelle. Er wird verwendet, um eine bestimmte Fluglage einnehmen und beibehalten zu können.

ADS
Der Air Data Sensor (dt. *Luftdatensonde*) liefert Daten über die umgebene Luft. Diese Daten beinhalten die Geschwindigkeit gegenüber der umströmenden Luft (*Airspeed*), den Anstellwinkel und den Schiebewinkel. Der Sensor ist ebenfalls über eine UART Schnittstelle verbunden und sendet mit einer Frequenz von 50 Hz. Er wird verwendet, um kritische Flugzustände, wie etwa einen Strömungsabriss (engl. *Stall*) zu vermeiden.

RCR
Der RC-Empfänger (Radio Control Receiver) stellt die Verbindung zum Piloten dar. Der Pilot steuert das Flugzeug mittels einer RC-Fernbedienung. Der Sensor ist per SPI Schnittstelle angeschlossen und liefert seine Daten mit 50 Hz.

CSA
Der Control Surface Allocator ist das Bindeglied zwischen der FCU und der Aktorik. Er bildet die Steuersignale von Pilot und FCU auf die Aktorik des AC2030 ab. Er ist wie der RCR mit einer SPI Schnittstelle verbunden und wird mit einer Frequenz von 50 Hz von der FCU angesteuert.

Virtuelle Steuersignale

Ein Pilot hat zur Steuerung eines Flugzeugs typischerweise vier primäre Eingabemöglichkeiten. Im einfachsten Fall steuert der Pilot direkt den Triebwerksschub, den Querruderausschlag, den Höhenruderausschlag und den Seitenruderausschlag, wodurch sich die gewünschten Bewegungen ergeben. Der AC2030 besitzt jedoch keine dedizierten Querruder (engl. *Aileron*) und Höhenruder (engl. *Elevator*). Stattdessen wird auf jeder Tragflächenseite je ein Ruder eingesetzt, das eine Kombination aus beiden darstellt und deren Funktionen bietet. Der englische Begriff für eine aus Quer- und Höhenruder kombinierte Steuerfläche lautet *Elevon*.

Die Direktansteuerung eines Elevons ist für einen Piloten schwierig bis unmöglich, da eine zweidimensionale Bewegung durch eine eindimensionale Ansteuerung abgebildet werden müsste. Statt der Direktansteuerung werden die Eingabesignale für Querruder und Höhenruder gemischt. Der Pilot verwendet weiterhin seine zwei gewohnten und vertrauten Eingabekanäle. Der Ansatz des Mischens ermöglicht es einem Piloten jegliche Art von Flugzeug steuern zu können, ohne dessen explizite Steuerflächen kennen zu müssen. Diese Abstraktion wird auch bei Flugzeugen ohne Modellcharakter verwendet. Unabhängig von der Anzahl und Art der Steuerflächen verwendet der Pilot zur Bewegung des Flugzeugs auch hier lediglich seine vier primären Eingabemöglichkeiten.

Durch die Abstraktion stellen die Eingabemöglichkeiten nicht länger den Ausschlag der Steuerflächen, sondern die Bewegungen des Flugzeugs dar. Die abstrakten Eingabemöglichkeiten sind somit nicht länger direkte, sondern *virtuelle Steuersignale*. Sowohl der Pilot, als auch der Flugregler in der FCU verwenden die folgenden acht virtuellen Steuersignale zur Steuerung des AC2030:

- **Rollen (roll)**: Drehen um die Längsachse
- **Nicken (pitch)**: Drehen um die Querachse
- **Gieren (yaw)**: Drehen um die Hochachse
- **Schub (throttle)**: Steuern des Triebwerkshubs
- **Landeklappen (flaps)**: Ein- und Ausfahren der Landeklappen
- **Fahrwerk (gear)**: Ein- und Ausfahren des Fahrwerks
- **Rettungssystem (rescue system)**: Auslösen des Rettungssystems
- **Sonstiges (aux)**: Undefiniert, vorgesehen für Erweiterungen

Steuerflächenkonfiguration

Die Anzahl, Art, Position und Größe der Steuerflächen eines Flugzeugs wird nachfolgend als *Steuerflächenkonfiguration* bezeichnet. Der AC2030 hat die folgende Steuerflächenkonfiguration (vereinfacht):

- **2** Elevons
- **2** Landeklappen (engl. *Flaps*)
- **2** Seitenruder (*Rudder*)
- **2** Triebwerke (*Engines*)

Die Aktorik des AC2030 besteht aus weiteren Komponenten, wie etwa dem Rettungssystem und der Fahrwerksteuerung. Jedoch sind diese für die vorliegende Arbeit nicht relevant und werden deshalb vernachlässigt.

Steuerflächenzuordnung

Das Abbilden (Mischen) der virtuellen Steuersignale auf die vorhandenen Steuerflächen wird nachfolgend *Steuerflächenzuordnung* genannt. Diese Aufgabe wird vom CSA übernommen. Die Auslagerung der Steuerflächenzuordnung steigert die Sicherheit des Systems. Bei Ausfall der FCU können die Steuersignale des Piloten direkt an den CSA geleitet werden. Hierfür wird eine elektrische Umschaltung in einer externen Komponente vorgenommen [Büscher (2014)].

Steuerflächenkonfiguration und Steuerflächenzuordnung sind untrennbar miteinander verbunden und müssen zueinander passen.

Hardware der FCU

Die FCU ist ein eingebettetes System. Als Hardwarebasis der FCU wird ein *STM32F4* Mikrocontroller der Firma STMicroelectronics verwendet. Für die Zukunft ist geplant, dass eine eigene Platine mit diesem Mikrocontroller entwickelt wird. Jedoch wird derzeit noch eine Demonstrationsplatine des Herstellers verwendet. Bei dieser handelt es sich um das *STM32F4-Discovery Evaluation Board*[1].

Firmware der FCU

Innerhalb der Firmware werden die zu erfüllenden Aufgaben in Komponenten unterteilt. Nachfolgend werden die wichtigsten Firmwarekomponenten erläutert:

- **Flugregler**: Autonome oder den Piloten unterstützende Steuerung des AC2030. Ersteres kann unter anderem über Wegpunkte erfolgen. Letzteres enthält zum Beispiel eine Stabilisierung der Fluglage.
- **Sensordatenerfassung**: Einlesen der Sensoren. Abstrahiert die restlichen Firmwarekomponenten der FCU von der verwendeten Sensorik.

[1] STM32F4-DISCOVERY - http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419 [Stand: 30.05.2014]

- **Sensordatenaufbereitung (State Estimation)**: Filtern der gemessenen Daten, sodass diese eine höhere Qualität aufweisen. Es wird auch eine *Sensordatenfusion* durchgeführt. Diese ermöglicht das Konstruieren von nicht messbaren physikalischen Größen. Die Sensordatenaufbereitung ist neben dem Flugregler die wichtigste Komponente der FCU.
- **Datenaufzeichnung**: Ähnelt einem *Flugschreiber* aus großen Flugzeugen. Die gemessenen Daten, sowie interne Zustände der FCU werden auf einer Speicherkarte aufgezeichnet.
- **Telemetrie**: Live-Übertragung verschiedener Messwerte zur Bodenstation.
- **Telecontrol**: Empfangen von Wegpunkten und Flugplänen von der Bodenstation
- **Scheduler**: Steuert die zeitlichen Abläufe innerhalb der FCU. Ein Scheduler kann als eine Vorstufe eines Betriebssystems verstanden werden.
- **System Health Monitoring**: Überwacht die FCU sowie die Sensorik auf Ausfälle und Fehlverhalten.
- **Aktorikansteuerung**: Übertragung der Steuersignale an Aktorik, bzw. CSA.

Die Firmware der FCU ist derzeit praktisch nicht existent, da noch keine Komponente umgesetzt wurde. Die geplante Softwarearchitektur ist detailliert in [Rohrer (2014)] beschrieben.

FCU-Modes

Die FCU soll drei unterschiedliche Modes beherrschen. Diese wirken sich jedoch nur auf das Verhalten des Flugreglers aus. Die restlichen Komponenten sind nicht vom verwendeten Mode abhängig. Nachfolgend werden die drei Modes erläutert:

- **manuell**: Der Pilot fliegt das Flugzeug. Die Steuersignale des Piloten werden direkt und ohne Verfälschung an den CSA durchgereicht.
- **unterstützend**: Der Pilot fliegt und die FCU fliegen das Flugzeug. Die FCU soll den Piloten dahingehend unterstützen, dass das Flugzeug in seiner Fluglage stabilisiert wird. Weiterhin können über die Bodenstation Sollwerte wie Höhe und Geschwindigkeit angegeben werden.
- **autonom**: Die FCU fliegt das Flugzeug. Der Flugregler führt einen autonom navigierten Flug durch. Hierfür werden von der Bodenstation Wegpunkte vorgegeben.

2.2. Regelungstechnik

Die Regelungstechnik ist eine Ingenieursdisziplin und bildet ein Teilgebiet der Automatisierungstechnik. Ihre Aufgabe ist das gezielte Beeinflussen von dynamischen Systemen, welche typischerweise technischer Natur sind. Erreicht wird dies durch die Regelung einer oder mehrerer Zustandsgrößen eines Systems. Der Begriff Regelung ist nach DIN IEC 60050-351 wie folgt definiert:

> "*Vorgang, bei dem fortlaufend eine variable Größe, die Regelgröße, erfasst, mit einer anderen variablen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.*"

Ein häufig verwendetes Beispiel für die Einführung in die Regelungstechnik ist der Tempomat eines Fahrzeugs. Dieses soll nachfolgend zum besseren Verständnis der Definition veranschaulicht werden.

Ein Tempomat hat die Aufgabe, eine konstante Geschwindigkeit eines Fahrzeugs zu halten oder zu erreichen. Der Tempomat spiegelt dabei den *Regler* wider. Das Fahrzeug stellt das technische System und damit die *Regelstrecke* dar.

Die vom Fahrer gewünschte Geschwindigkeit ist in diesem Beispiel die *Führungsgröße*. Sie wird häufig auch *Sollwert* genannt. Die tatsächliche Geschwindigkeit des Fahrzeugs ist die *Regelgröße*. Die Differenz der beiden Größen ist die *Regelabweichung*. Diese Regelabweichung wird dem Tempomaten zugeführt. Dieser hat die Aufgabe, die Differenz der Geschwindigkeiten allmählich zu minimieren, bzw. sie bei null zu halten. Hierfür muss er eine Möglichkeit besitzen, die tatsächliche Geschwindigkeit des Fahrzeugs zu verändern. Typischerweise kann er die Drosselstellung des Motors variieren, wobei die Drosselklappe das *Stellglied* darstellt. Das Signal, welches vom Tempomaten zur Drossel gegeben wird ist die *Stellgröße*. Je nach Fahrzeug kann dieses Signal mechanisch oder elektrisch übertragen werden. Auf die Regelstrecke wirken *Störgrößen*. Im Falle des Fahrzeugs wären dies zum Beispiel Windeinflüsse oder Reibungen zwischen Reifen und Straße. Je nach Störgrößen und physikalischem Verhalten des Fahrzeugs wird sich die Geschwindigkeit durch die Variation der Drosselklappe verändern. Die tatsächliche Geschwindigkeit wird vom *Messglied* erfasst. Dieses ist bei Fahrzeugen ein Sensor, der die Ausgangsdrehzahl des Getriebes misst und anschließend in eine Geschwindigkeit umrechnet. Durch die Rückführung der Geschwindigkeit auf den Tempomaten ergibt sich eine geschlossene Schleife und der Prozess beginnt von Neuem.

Die Gesamtheit aller Teile wird *Regelkreis* genannt. Abbildung 2.2 zeigt die abstrakten und tatsächlichen Komponenten des Regelkreises Tempomat.

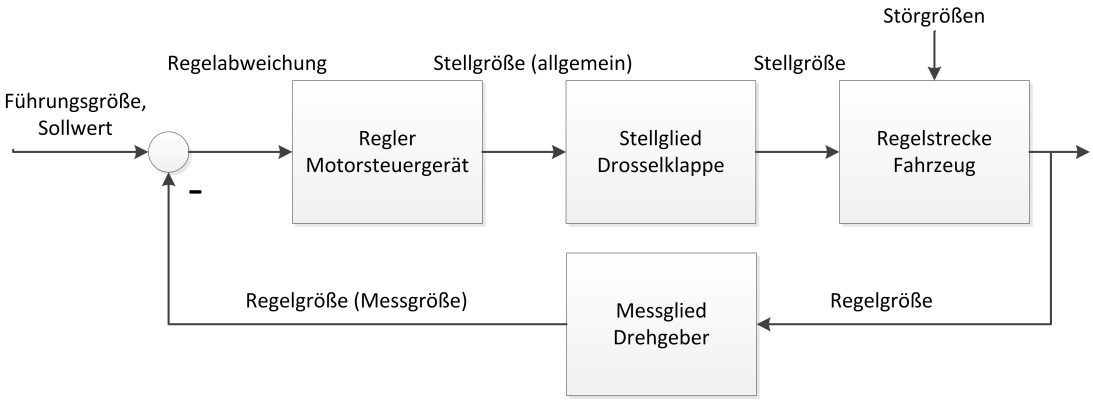

Abbildung 2.2.: Regelkreis eines Tempomaten

Ein Regler hat die Aufgabe, die Regelgröße allmählich der Sollgröße anzupassen. Je nach verwendetem Kontext ist der Regler dabei als mathematisches Modell oder als reale Umsetzung vorhanden. Die Realisierung erfolgt typischerweise durch Mechanik, elektronisch durch Operationsverstärker oder digital durch Computersysteme. Im Folgenden ist mit dem Begriff *Regler* das mathematische Modell und nicht die reale Umsetzung gemeint.

Seine Aufgabe kann ein Regler gut oder schlecht erfüllen. So kann er bei schlechter Auslegung zum Beispiel ein stabiles System instabil werden lassen, zu langsam auf Regelabweichungen reagieren, bei Erreichen des Sollwertes überschwingen, eine konstante Regelabweichung aufweisen oder eine gewünschte Dämpfung des Systems nicht bieten. Wie gut er diese Aufgabe erfüllen muss, hängt jeweils von den Anforderungen des Anwendungsfalls ab. Zum Beispiel ist die Regelung der Steuerstäbe eines Kernreaktors um ein Vielfaches kritischer als die Regelung der Temperatur eines Zimmers.

Das Modell des Reglers besteht aus einer Struktur und dazugehörigen Parametern. Das Bilden des mathematischen Modells wird auch *Reglerentwurf*, *Reglerauslegung* oder *Reglersynthese* genannt. Ein computergestützter Entwurf erfolgt häufig mit dem Simulationswerkzeug Simulink, welches in 3.2.6 näher betrachtet wird. Das Modell kann durch zwei verschiedene Verfahren gebildet werden, welche im Nachfolgenden erläutert sind.

Experimenteller Entwurf
Die zweifelsfrei einfachste Methode einen Regler auszulegen, ist das simple Ausprobieren verschiedener Strukturen und Parameter, bis eine zufriedenstellende Kombination gefunden wurde. Dabei wird das reale technische System verwendet. Diese Methode sollte bei sicherheitskritischen Systemen jedoch vermieden werden.

Analytischer Entwurf
Die weitaus schwierigere Methode ist, den Regler anhand des mathematischen Modells des

zu regelnden Systems auszulegen. Hierfür muss das Modell der Regelstrecke bekannt sein. Es ergibt sich aus den physikalischen Zusammenhängen der Systemkomponenten. Diese Zusammenhänge können entweder aufgrund von bekannten Gesetzmäßigkeiten (*Systemanalyse*) oder durch das experimentelle Messen (*Systemidentifikation*) des Systemverhaltens auf eine Eingabe bestimmt werden. Das Ergebnis ist in beiden Fällen eine *Übertragungsfunktion*, die das Verhalten des Systems mathematisch beschreibt. Das System kann ein *lineares* oder ein *nichtlineares* Übertragungsverhalten aufweisen, wobei ein nichtlineares System zunächst linearisiert werden muss. Bei jeweils einem Eingang und einem Ausgang des Systems spricht man von einem Eingrößensystem, ansonsten von einem Mehrgrößensystem. Nichtlineare und Mehrgrößensysteme weisen eine ungleich höhere mathematische Komplexität auf. Bei einem hohen Komplexitätsgrad kann das Systemverhalten möglicherweise nur näherungsweise bestimmt werden. Der Realitätsgrad, den das Modell widerspiegeln muss, hängt ebenfalls von den Anforderungen ab. Dieser Realitätsgrad wird im Folgenden auch als Güte einer Übertragungsfunktion bezeichnet. Aus dem Modell der Regelstrecke kann ein Regler entworfen werden, der den Anforderungen des gegebenen Problems genügt.

Je nach Komplexität der Regelstrecke können Systemidentifikation und Reglerentwurf sehr zeitaufwändige Prozesse sein. Als Kompromiss verwendet man häufig eine Kombination aus experimentellem und analytischem Entwurf. Dabei wird die Systemidentifikation bis zu einem gewissen Grad durchgeführt. Das Ergebnis wird verwendet, um den Regler in seiner Grundstruktur festzulegen. Im zweiten Schritt werden die Parameter durch experimentelle Ermittlung am realen System optimiert. Dies wird als *Regleroptimierung* bezeichnet.

Falls genau die vier Komponenten Regler, Regelstrecke, Messglied und Stellglied im Regelkreis vorkommen, wird im Folgenden von einem *Standardregelkreis* gesprochen. Im Kontext der vorliegenden Arbeit ist die FCU der Regler, die Steuerflächen das Stellglied, die Sensorik das Messglied und der AC2030 als Flugzeug die Regelstrecke.

Die Strukturen, bzw. die Modelle der einzelnen Komponenten werden durch sogenannte *Übertragungsglieder* mit jeweils eigenen Übertragungsfunktionen dargestellt. Die Übertragungsglieder weisen in den meisten Fällen proportionales (P), integrierendes (I), differenzierendes (D) oder ein verzögerndes (auch *Totzeit* (T) genannt) Verhalten auf. Häufig kommen diese Verhaltensweisen auch in Kombination miteinander vor. Ein klassischer Regler besteht aus P, I und D Anteilen. Alle drei Anteile zusammen ergeben einen *PID-Regler*.

Der vorliegende Abschnitt kann nur einen kurzen Überblick über das weite Feld der Regelungstechnik geben. Der interessierte Leser sei auf [Föllinger u. a. (2013)], [Schulz (2010)] oder [Unbehauen (2008)] verwiesen.

Der Entwurf eines Reglers ist durch folgende Annahme geprägt:

> *Eine realitätsnahe Simulation erlaubt einen experimentellen Entwurf des Reglers ohne das Risiko der realen Umgebung eines sicherheitskritischen Systems. Je*

realitätsnaher die Umgebung eines zu entwerfenden Reglers nachgebildet und simuliert wird, desto einfacher lässt sich ein qualitativ hochwertiger Regler entwerfen.

Dieser Grundsatz zieht sich durch die gesamte vorliegende Arbeit.

2.3. Flugregelung

Dieser Abschnitt soll einen kurzen Einblick in die Flugregelung geben. Typische Aufgaben von Flugreglern gehen von einfachen, wie etwa Fluglage halten, Geschwindigkeit halten oder Kurs halten bis hin zu komplexeren, wie etwa autonome Navigation oder autonome Landung.

Begriffe

Als Nomenklatur im Zusammenhang mit Flugregelung sollen in dieser Arbeit folgende Begrifflichkeiten gelten:

- **Flugregelung**: Disziplin der Regelungstechnik, die sich mit der Regelung von Fluggeräten beschäftigt.
- **Flugregler**: Das mathematische Modell des Reglers. Typischerweise aus mehreren Einzelreglern zusammengesetzt.
- **Autonome Flugsteuerung**: Tätigkeit des Reglers als Gesamtfunktion.
- **Autopilot**: Die konkrete Umsetzung des Flugreglers in einem oder mehreren vernetzen Computersystemen.
- **Avionik, Avioniksystem**: Bezeichnet die Gesamtheit aller elektrischen und elektronischen Systeme eines Fluggeräts. In der vorliegenden Arbeit sind damit lediglich die FCU und der CSA gemeint.
- **fly-by-wire**: Als fly-by-wire Flugzeuge werden diejenigen bezeichnet, die über einen Autopiloten verfügen.

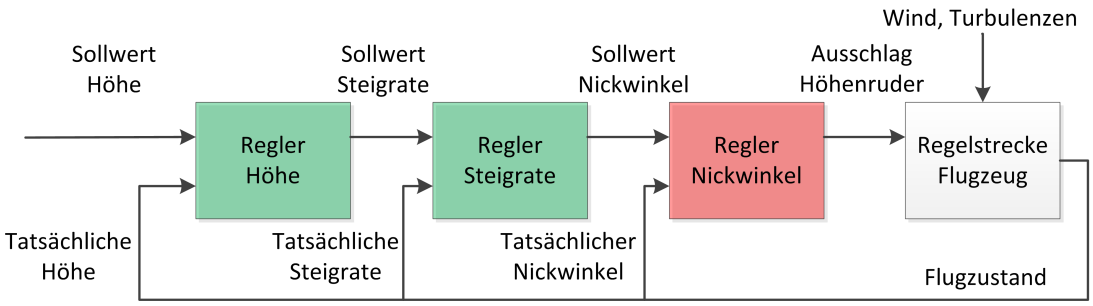

Abbildung 2.3.: Kaskadierter Flugregler zum Regeln der Flughöhe

Struktur eines einfachen Flugreglers

Die Abbildung 2.3 zeigt einen einfachen Regelkreis, der ein Flugzeug auf einer gewünschten Sollhöhe halten kann.

Das Flugzeug stellt die Regelstrecke dar. Als (Gesamt-)Regler werden drei einzelne hintereinander geschaltete Regler verwendet. Es wird dabei auch von einem kaskadierten Regler oder einer *Reglerkaskade* gesprochen. Der äußerte Regler versucht, die tatsächliche Höhe der vom Piloten eingegebenen Sollhöhe anzupassen. Die Ausgabe ist eine gewünschte Steigrate. Der nächstinnere Regler versucht diese zu erreichen. Hierfür kommandiert er den letzten Regler mit einem gewünschten Nickwinkel. Der letzte Regler verwendet als Stellglied das Höhenruder des Flugzeugs, um dieses in die gewünschte Fluglage zu bringen. Die Rückführung der Werte erfolgt durch im Flugzeug verbaute Sensoren. Analoge Reglerkaskaden werden auch für die Kurs- oder Geschwindigkeitsregelung verwendet.

Komplexe Flugregler werden ihrer Aufgabe nach in die Kategorien *Guidance*, *Navigation* und *Control (GNC)* eingeteilt. Die innere Schleife (rot), ist dabei der Control Teil. Er hat die Aufgabe, das Flugzeug zu steuern. Die äußere Schleife (grün), stellt den Guidance Teil dar. Dessen Aufgabe ist es, das Flugzeug entlang einer Trajektorie zum gewünschten Zielpunkt zu führen. Der Navigation Teil (nicht eingezeichnet) hat die Aufgabe, Trajektorien für den Guidance Teil zu generieren. Die Trajektorien werden aus vom Piloten eingegebenen Wegpunkten gebildet. Die Grenzen zwischen den GNC Teilen des Flugreglers sind nicht festgelegt und können je nach Auffassung variieren.

Eine weitere Art, die einzelnen Schichten des Flugreglers einzuteilen, basiert auf dem Grad der Autonomie, den er bietet. Wobei von innen nach außen ein immer höherer Grad erreicht wird. Die typischen Schichten sind (aufsteigend nach Grad der Autonomie): *Stability Augmentation System*, *Attitude Control System*, *Flight Path Control System* und *Flight Management System*.

Stability Augmentation Systems
Innerhalb der Control Schleife können sogenannte *Stability Augmentation Systems (SAS)* eingesetzt werden. Diese verändern künstlich die Reaktion des Flugzeugs auf Eingaben von Pilot und Autopilot. Sie werden verwendet, um die Stabilität und Dämpfung der Bewegung zu verändern. Dies kann mit unterschiedlichen Fahrwerkseinstellungen in Mittel- und Oberklasse Fahrzeugen verglichen werden (Sportlich, Komfort, Eco, etc.).

Flight Control Laws
Die *Flight Control Laws (FCL)* bestimmen die Reglerphilosophie, mit der Pilot und Autopilot das Flugzeug steuern. Eine Richtungsänderung des Flugzeugs kann zum Beispiel durch Rollen oder Gieren erreicht werden. Das FCL bestimmt auch, wie die Eingaben der Piloten interpretiert werden. So kann eine Eingabe für das Rollen als gewünschter Rollwinkel, Rollrate oder Querruderausschlag umgesetzt werden. Je nach Systemzustand (Fehlerfall) kann der Autopilot automatisch in ein anderes FCL umschalten. Eine Übersicht der Flight Control Laws moderner Airbus Flugzeuge findet sich in [AirbusFCL].

Flight Envelope Protection
Eine *Flight Envelope Protection (FEP)* ist ein in Flugreglern enthaltener Schutzmechanismus. Sie kann eingreifen, falls das Flugzeug kritische Flugzustände einnimmt, bzw. es vor diesen bewahren. Die Autopiloten moderner Airbus Flugzeuge lassen es unter anderem nicht zu, dass kritische Roll- oder Nickwinkel erreicht werden oder das Flugzeug in einen Stall gerät.

Das deutsche Standardwerk zum Thema Flugregelung ist der *Brockhaus* [Brockhaus u. a. (2011)]. Für ausführliche Informationen sei hier auf diesen verwiesen. Aus der englischen Literatur ist McLean [McLean (1990)] zu empfehlen.

2.4. Das Flugzeug als Regelstrecke

Für den qualitativ hochwertigen Entwurf eines Flugreglers muss das mathematische Modell und damit die Übertragungsfunktion des Flugzeugs bekannt sein. Dieser Abschnitt soll den Leser in Aspekte der mathematischen Modellbildung eines Flugzeugs einführen.

Das Bilden der Übertragungsfunktion eines so hochkomplexen Systems wie das eines Flugzeugs ist alles andere als trivial. Das physikalische Verhalten (die Bewegung im Raum) wird durch viele Einflussfaktoren bestimmt. Dabei spielen Flugzeugeigenschaften wie Geometrie, Massenträgheit und Trägheitsmoment, Aeroelastizität, Masse, Steuerflächengröße und -anordnung, Triebwerksleistung, Flügelprofile und einige weitere eine Rolle. Die Disziplin, die sich unter anderem mit diesen Fragestellungen auseinandersetzt, heißt Flugmechanik.

Nach aktuellem Stand der Wissenschaft existieren zwei Ansätze, die Übertragungsfunktion eines Flugzeugs zu bilden.

Analytische Modellbildung

Dieses Vorgehen stellt die analytische Modellbildung, wie sie im Abschnitt zur Regelungstechnik beschrieben wurde, dar. Die Übertragungsfunktion wird anhand einzelner Übertragungsglieder aufgestellt. Die analytische Modellbildung ist im Gegensatz zum zweiten Ansatz weitaus aufwendiger, da jedes einzelne Übertragungsglied in seinem Verhalten bekannt sein muss. Dies kann ein generischer Flugsimulator nicht leisten. Solche die hier betrachtet werden, verwenden den zweiten Ansatz. Daher wird in dieser Arbeit nicht weiter auf dieses Verfahren eingegangen.

Aufstellen der linearisierten Bewegungsgleichungen

Ein Flugzeug besitzt drei translatorische und drei rotatorische Freiheitsgrade, wenn es sich im dreidimensionalen Raum bewegt. Dabei ist es vier natürlichen Kräften ausgesetzt: Schub, Auftrieb, Widerstand, Gewicht. Das Aufstellen einer einzigen Bewegungsgleichung ist hochkomplex. Zur Vereinfachung wird diese daher in zwei von einander unabhängige Bewegungsgleichungen unterteilt. Die *Längsbewegung* beschreibt das Verhalten entlang der Längsachse und der Hochachse. Die *Seitenbewegung* entlang der Querachse. Beide Bewegungsgleichungen sind dabei hochgradig nichtlinear und besitzen mehrere Ein- und Ausgangsgrößen. Sie werden linearisiert und in die *Zustandsraumdarstellung* gebracht. Dies ist eine geeignete Darstellungsform für Übertragungsfunktionen mit solchen Eigenschaften. Sie hat die allgemeine Form:

$$\underline{\dot{x}}(t) = \underline{\underline{A}} * \underline{x}(t) + \underline{\underline{B}} * \underline{u}(t) + \underline{\underline{E}} * \underline{z}(t) \qquad (Zustandsgleichung)$$
$$\underline{y}(t) = \underline{\underline{C}} * \underline{x}(t) + \underline{\underline{D}} * \underline{u}(t) \qquad (Ausgangsgleichung)$$

Die Elemente *x*, *u*, *y* und *z* sind zeitabhängige Vektoren. Dabei ist x der Zustand (Zustandsvektor), u der Eingang (Eingangsvektor oder Steuervektor), y der Ausgang (Ausgangsvektor) und z die Störgröße (Störgrößenvektor) des Systems. *A*, *B*, *C*, *D*, *E* sind Matrizen, die Koeffizienten des Systems enthalten. A wird als *Systemmatrix* oder *Zustandsmatrix*, B als *Steuermatrix*, C als *Ausgangsmatrix*, D als *Durchgangsmatrix* und E als *Störgrößenmatrix* bezeichnet. Der neue Zustand ergibt sich aus dem aktuellen Zustand, der aktuellen Eingabe und der anliegenden Störung. Der Ausgang ergibt sich aus aktuellem Zustand und Eingabe. Ein in der Informatik zur Zustandsraumdarstellung vergleichbares Konzept ist mit einem Moore Automaten gegeben. Bei diesem ergeben sich Ausgang und neuer Zustand ebenfalls aus aktuellem Eingang und Zustand.

Die Form des Zustandsvektors hängt von der erforderlichen Güte des mathematischen Modells ab. Typische Größen sind Fluglage, Geschwindigkeiten und Drehraten. Die Form der System-

matrix leitet sich aus der Form des Zustandsvektors ab. Die *Zustandsgleichung* beschreibt das Verhalten des Systems. Der erste Summand gibt das Systemverhalten aufgrund des aktuellen Zustands wider. Der zweite aufgrund der Eingänge, welches beim Flugzeug die Stellpositionen der Steuerflächen und der erzeugte Triebwerksschub sind. Der dritte Summand aufgrund der wirkenden Störungen. Beim Flugzeug sind dies atmosphärische Werte wie Wind und Turbulenzen. Man kann die Zustandsgleichung als Antwort folgender Frage auffassen: "*Wie verhält sich das Flugzeug zum Zeitpunkt t, bei den Ruderstellungen und Triebwerksschub u, dem Zustand x und den Windeinflüssen z?*"

Beim technischen System Flugzeug sind die Ausgänge typischerweise auch die Zustände des Systems. Daher ist C häufig die Einheitsmatrix und D gleich null. Die Koeffizienten der Matrizen A, B und E spiegeln die oben genannten Eigenschaften des Flugzeugs wider. Sie werden auch *Ersatzgrößen* genannt und ergeben sich aus den aerodynamischen Beiwerten, wie etwa Auftriebsbeiwert oder Widerstandsbeiwert, des Flugzeugs. Für die Gesamtheit aller Koeffizienten wird fortan der Begriff *Derivativa* verwendet. Beim Ermitteln der Derivativa kann auf drei generelle Ansätze zurückgegriffen werden.

Bekannte Geometrien und Erfahrungen aus ähnlichen Flugzeugen
Anhand von bekannten Geometrien, Gewichten und Eigenschaften der einzelnen Bauteile können die Derivativa abgeschätzt werden. Hierfür eignet es sich auch, Flugzeuge mit ähnlichen Eigenschaften zu verwenden.

Windkanaltests
Die zweite Möglichkeit besteht darin, ein verkleinertes Modellflugzeug in einem Windkanal zu testen. Die ermittelten Werte können auf das echte Flugzeug hochskaliert werden. Nguewo hat sich mit der Methode der Hochskalierung beschäftigt [Nguewo (2007)].

Flugversuche
Die einfachste Möglichkeit besteht darin, die Derivativa in Flugversuchen zu ermitteln. Sie ist die genaueste der drei Methoden. Dafür hat sie den Nachteil, dass sie erst durchgeführt werden kann, wenn das Flugzeug bereits gebaut ist. Diese Methode wird vom BWB-Team für den AC2030 verwendet.

Die drei Methoden schließen sich gegenseitig nicht aus und werden häufig auch in Kombination miteinander eingesetzt. So wird ein Verkehrsflugzeug anhand von Erfahrungen und sich daraus ergebenen Erwartungen entwickelt. Daraufhin wird ein kleineres Modell im Windkanal getestet. Im letzten Schritt werden Flugversuche unternommen. Die Derivativa und damit das mathematische Modell des Flugzeugs werden in jedem Arbeitsgang sukzessive verfeinert. Bei ausreichender Genauigkeit kann das Verhalten in Flugsimulatoren nachgebildet und zur Ausbildung von Piloten eingesetzt werden.

Für dieses Gebiet der Flugmechanik eignen sich als Literatur die im vorangegangen Abschnitt empfohlenen Werke von Brockhaus [Brockhaus u. a. (2011)] und McLean [McLean (1990)].

2.5. Flugsimulation

Der Realitätsgrad einer Simulationsumgebung entscheidet maßgeblich darüber, wie brauchbar sie für Entwicklung und Test von Systemen ist. Neben dem Vorhandensein des mathematischen Modells des Flugzeugs muss eine weitere Voraussetzung erfüllt sein: Das Modell muss in einer Simulation ausgeführt werden. Im Folgenden werden einige Grundlagen der Flugsimulation dargelegt.

Mess- und Stellglied

Ein Regelkreis besteht neben Regler und Regelstrecke auch aus einem Stellglied und einem Messglied. Jede dieser Komponenten hat eine eigene Übertragungsfunktion. Die des Reglers gilt es zu entwerfen. Die der Regelstrecke wurde im vorherigen Abschnitt beschrieben. Idealerweise sollten Stellglied und Messglied den Eingang unverändert auf den Ausgang weitergeben und sich demnach so verhalten, als wären sie nicht vorhanden. Dies ist in der Realität nicht der Fall, da Sensoren und Aktoren nicht perfekt sind und es auch nicht sein können.

Messglied
Eine physikalische Größe kann nicht unverfälscht ermittelt werden. Folgende Eigenschaften verfälschen das Messergebnis:

- Jeder Sensor benötigt eine gewisse Zeit zum Messen einer Größe.
- Die Größen lassen sich nicht beliebig genau messen.
- Es treten Effekte wie Rauschen und Drift auf. Diese sind durch Fertigungstoleranzen unvermeidbar.
- Das Verhalten elektrischer Bauteile ist temperaturabhängig.
- Die Signalübertragung von Sensor zu Regler benötigt Zeit (Totzeit).

Stellglied
Die Stellglieder eines Flugzeugs sind seine Triebwerke und Steuerflächen. Bewegt werden sie durch Servomotoren mit hydraulischer, pneumatischer oder elektrischer Funktionsweise. Die folgenden Eigenschaften müssen dabei beachtet werden:

- Kein Motor kann unendlich schnell beschleunigen.
- Begrenzte Stellgeschwindigkeit der Motoren.
- Endlich hohe Auflösung der Stellpositionen.
- Flattern der Ruder aufgrund von hohen aerodynamischen Kräften.

- Wirkung auf die Umwelt (erzeugtes Drehmoment).
- Die Signalübertragung von Regler zu Aktor benötigt Zeit (Totzeit).

Die Eigenschaften der Sensoren und Aktoren können als Übertragungsfunktionen nachgebildet werden. Im Folgenden werden das Übertragungsverhalten des Messglieds als *Sensormodell* und die des Stellglieds als *Aktuatormodell* bezeichnet. In der Zustandsraummodellierung stellt die Steuermatrix B das Aktuatormodell dar.

Für den Entwurf eines qualitativ hochwertigen Reglers wirken sich somit nicht nur die Güte der Regelstrecke, sondern auch die des Aktuatormodells und Sensormodells vorteilig aus. Eine realitätsnahe Simulation der Umgebung eines Reglers erfordert demnach, dass das Übertragungsverhalten der Regelstrecke, des Sensormodells und des Aktuatormodells bekannt ist und realitätsnah nachgebildet wird.

Nomenklatur

Nachfolgend werden einige Begriffe beschrieben, die in dieser Arbeit im Zusammenhang mit Simulation und insbesondere Flugsimulation verwendet werden.

Begriffe der Flugsimulation

- **Flugphysik**: Die Übertragungsfunktion des Flugzeugs wird fortan als *Flugphysik* bezeichnet.
- **Ausgeführte Flugphysik**: Mit *ausgeführter Flugphysik* ist gemeint, dass die Flugphysik zur Laufzeit in einer Software berechnet wird.
- **Simulator-Engine, Simulator-Backend**: Der Softwareteil des Flugsimulators, der die Flugphysik ausführt, wird als *Simulator-Engine* oder *Simulator-Backend* bezeichnet.
- **Simulationsschritt**: Flugsimulatoren arbeiten in diskreten Zeitschritten. In jedem Schritt wird die Flugphysik einmal berechnet. Solch ein Zeitschritt wird fortan als *Simulationsschritt* bezeichnet.
- **Eine Simulation**: Von *einer Simulation* wird gesprochen, wenn mehrere Komponenten der Infrastruktur zur Laufzeit ausgeführt werden und sie somit gemeinsam eine Simulation bilden.
- **Echtzeit**: Zeitverhalten mit *harten* Echtzeitanforderungen.
- **Realzeit**: Physikalisches Zeitverhalten, jedoch ohne Echtzeitanforderungen. In der Arbeit wird hierfür auch der Begriff *live* verwendet.

Datensätze und Signale einer Simulation
Für die Kopplung von Regler und Regelstrecke sind grundsätzlich zwei Signalvektoren von Nöten. Je nach Betrachtungsweise sind dies die Ein- und Ausgänge des Reglers oder der Regelstrecke. Sie können als zwei Datensätze zusammengefasst werden, indem man lediglich die Signalvektoren Regler zu Regelstrecke und von Regelstrecke zu Regler als solche definiert. In der vorliegenden Arbeit werden hierfür folgende Begriffe verwendet:

- **Flugzustand**: Enthält die Daten, die von Regelstrecke zu Regler gesendet werden. Bei realen Flügen wird der *Flugzustand* über Sensoren gemessen und ggf. aufbereitet.
- **Steuereingaben, Steuersignale**: Die Ausgaben des Reglers werden als *Steuereingaben* oder *Steuersignale* bezeichnet. Weiterhin werden diese Bezeichnungen auch für die Eingaben des Piloten verwendet.
- **Simulationsdaten**: Bezeichnet generell die Daten, die während einer Simulation zwischen den Komponenten ausgetauscht werden.

2.6. Testverfahren eingebetteter Systeme

Dieser Abschnitt soll einen Einblick in sogenannte *X-in-the-Loop*-Testverfahren für eingebettete Systeme geben. Weiterhin wird die in dieser Arbeit verwendete Nomenklatur der in diesem Zusammenhang auftretenden Begriffe definiert.

Zum Testen eingebetteter Systeme eignen sich X-in-the-Loop-Simulationen. Wobei *X* stellvertretend für eine Reihe von verwandten Testverfahren steht. Generell kann jedes eingebettete System mit diesen Verfahren getestet werden. Da der Fokus der vorliegenden Arbeit auf dem Testen von Flugreglern liegt, wird ein solcher zur Verdeutlichung herangezogen.

Das *in-the-Loop* bedeutet, dass sich der Regler in einer simulierten Umgebung befindet. Also dass Regelstrecke, Aktuatormodell und Sensormodell auf einem oder mehreren Computersystemen ausgeführt werden. Das stellvertretende *X* steht dabei für den *Ort*, auf dem der Regler ausgeführt wird. Die Einteilungen und Grenzen sind in der Fachwelt nicht immer eindeutig. So verwendet Kirner die folgende Einteilung [Kirner (2009)]:

- **Model-in-the-Loop (MIL)**: Der Regler wird in der Software ausgeführt, in der er modelliert wird. Diese Software ist meist Simulink. Der Regler kann somit bereits in einer sehr frühen Phase der Entwicklung getestet und ausgelegt werden.
- **Software-in-the-Loop (SIL)**: Der Regler wird in einer Programmiersprache implementiert und auf einem PC ausgeführt. Dieser Schritt stellt die Verifikation der Implementierung dar. Die Umsetzung kann per Hand oder mittels automatischer Codegenerierung

erfolgen. Dieser Schritt kann bei automatischer Codegenerierung ausgelassen werden, sofern der Codegenerator der Modellierungssoftware korrekt arbeitet.

- **Processor-in-the-Loop (PIL)**: Der Regler wird auf dem Zielprozessor mittels der Zielprogrammiersprache auf einer Prototypenhardware umgesetzt. Die Ein- und Ausgabe der Werte erfolgt nicht über die geplanten physikalischen Schnittstellen, sondern über Platzhalter-Schnittstellen. Dieser Schritt gibt Aufschluss darüber, ob der geplante Prozessor die erforderliche Leistung bietet. Ab dieser Phase kann die Simulationsumgebung in Echtzeit ausgeführt werden.
- **Hardware-in-the-Loop (HIL)**: Der Regler arbeitet auf dem Zielsystem und nicht länger auf einem Prototyp. Es werden die vorgesehenen physikalischen Schnittstellen verwendet. Es werden mindestens die benötigten Firmwarekomponenten ausgeführt, um den Regler vom Rest der Hardware zu abstrahieren. Je nach Entwicklungszeitpunkt können diese Firmwarekomponenten lediglich Platzhalter sein. In dieser Phase muss die Simulationsumgebung in Echtzeit ausgeführt werden. Dies ist erforderlich, damit das Zeitverhalten der Umgebung nachgebildet werden kann. Die Firmware kann somit nicht zwischen realer und simulierter Umgebung unterscheiden.

  Eine HIL-Simulation ermöglicht auch den Test des gesamten eingebetteten Systems. In diesem Fall kann keine der Hard- und Softwarekomponenten zwischen Realität und Simulation unterscheiden.

Nach Kirner können die Schritte direkt auf verschiedene Phasen des V-Modells abgebildet werden. So findet MIL während des Softwareentwurfs, SIL während des Integrationstests statt. PIL und HIL werden in der Phase Systemtest angewendet. Demnach bilden Schritte eine natürliche Reihenfolge, in der sie angewendet werden **können**. Es ist nicht unüblich, dass einzelne oder mehrere Schritte keine Anwendung finden.

In dieser Arbeit wird eine von Kirner leicht abgewandelte Definition der einzelnen Verfahren verwendet.

- **MIL**: Wie bei Kirner
- **SIL**: Der Regler und gegebenenfalls weitere Firmwarekomponenten werden auf einem PC ausgeführt.
- **PIL**: Der Regler und weitere Firmwarekomponenten werden auf einem eingebetteten System ausgeführt. Dabei wird als Hardware entweder ein Prototyp oder das Zielsystem verwendet. Die Kommunikation zur Außenwelt wird über Platzhalter-Schnittstellen abgehandelt. Aus diesem Grund wird *zwingend* nicht exakt die Firmware eingesetzt, die in der realen Umgebung verwendet wird. Es ist jedoch nicht festgelegt, wie sehr sich die Firmware von der tatsächlichen unterscheidet. Die Simulationsumgebung wird nicht in Echtzeit, sondern in Realzeit, ausgeführt.

- **HIL**: Der Regler und weitere Firmwarekomponenten werden auf dem Zielsystem ausgeführt. Zur Kommunikation mit der Außenwelt werden die vorgesehenen physikalischen Schnittstellen verwendet. Die Firmwarekomponenten können teilweise noch aus Platzhaltern bestehen, jedoch können diese aufgrund der Verwendung der vorgesehenen physikalischen Schnittstellen nicht zwischen Realität und Simulation unterscheiden. Die Simulationsumgebung muss in Echtzeit ausgeführt werden.

Zwischen den Begriffen Testumgebung und Simulationsumgebung wird im Weiteren nicht unterschieden, da sie im Kontext dieser Arbeit die gleichen Bedeutungen haben. Als Abkürzungen werden (*M*/*S*/*P*/*H*)*IL* für die jeweilige *in-the-Loop* Simulation verwendet.

Eine ausführliche Beschreibung von HIL-Testverfahren wurde von Martin Gomez verfasst [Gomez (2001)]. Er betont insbesondere auch die Vorteile dieses Verfahrens. Lizarraga, Elkaim und Curry vom *Santa Cruz Low-cost GNC System* (*SLUGS UAV*) Projekt der University of California, Santa Cruz führen die Vorzüge ihres Systems weitestgehend auf die Nutzung von X-In-the-Loop-Testverfahren zurück [Lizarraga u. a. (2013)].

3. Analyse

Das in dieser Arbeit angestrebte System ist keinesfalls das erste dieser Art. Vielmehr sollen Konzepte und Ideen von vergleichbaren Systemen übernommen und verbessert werden. Dieses Kapitel gibt eine Übersicht über vergleichbare Systeme und Projekte, sowie über am Markt verfügbare Flugsimulatoren.

3.1. Vergleichbare Systeme und Projekte

Im Folgenden werden Konzepte von Open Source Projekten, sowie von vergleichbaren Systemen aus Industrie und Wissenschaft für einen Vergleich herangezogen und beschrieben.

3.1.1. Open Source Projekte

Die Open Source Szene beschäftigt sich bereits seit einigen Jahren mit der Entwicklung frei verfügbarer autonomer Flugsteuerungen und dazugehöriger Komponenten für UAV Systeme. Im Rahmen der Recherche wurden verschiedene Open Source Projekte analysiert. Die hier beschriebenen Systeme bieten die ausgereiftesten Test- und Simulationsmöglichkeiten. Eine generelle Übersicht über vergleichbare Open Source Projekte ist in [Hasberg (2013)] gegeben.

Paparazzi

Das *Paparazzi* Projekt [Brisset u. a. (2006)] wurde 2003 von Studierenden der französischen Hochschule für zivile Luftfahrt (*ENAC*) gegründet und stellt das älteste der hier analysierten Open Source Projekte dar. Die Zielgruppe von Paparazzi sind weniger Privatpersonen, als professionelle Anwender wie zum Beispiel wissenschaftliche Einrichtungen. Die entwickelten Systeme sind ausgereift und bieten umfangreiche Möglichkeiten zum Testen der verwendeten Hard- und Firmware. Die korrekte Umsetzung des Flugreglers wurde in [Albert (2005)] durch formale Methoden verifiziert. Die gesamte Entwicklungs- und Arbeitsumgebung läuft lediglich

auf Linux. Teile dieser können auch auf MacOS ausgeführt werden. Windows wird nicht unterstützt. Paparazzi verwendet eine Vielzahl verschiedener Hardwareplattformen. Für die Betrachtung der Simulationsmöglichkeiten sind diese allerdings nicht wichtig und werden deshalb nicht beschrieben. Abbildung 3.1 zeigt eine Systemübersicht des gesamten UAS bei realen Flügen.

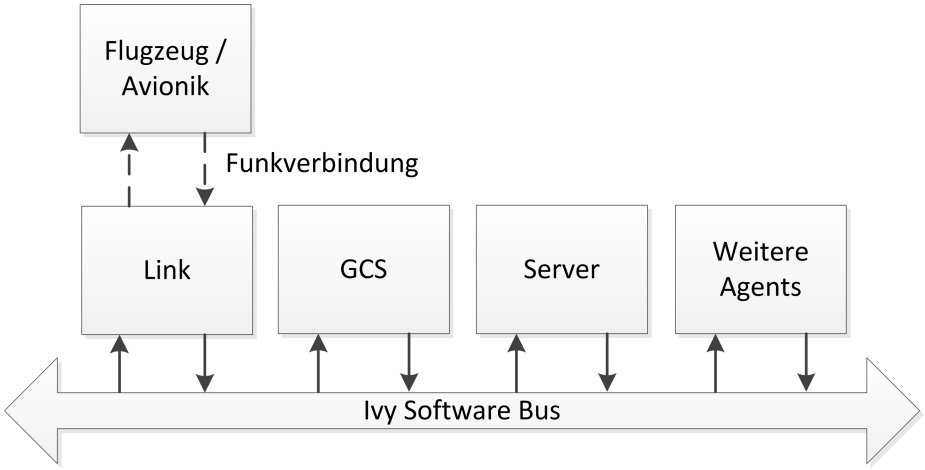

Abbildung 3.1.: Systemarchitektur von Paparazzi bei realen Flügen

Die Verbindung der Komponenten untereinander wird über eine Middleware namens *Ivy*[1] realisiert, welche die Verteilung der Nachrichten an andere Teilnehmer übernimmt. Dies vereinfacht die Anwendungslogik einzelner Komponenten, da keine Netzwerkprogrammierung vorgenommen werden muss. Durch die Verwendung eines lose gekoppelten Bussystems können im laufenden Betrieb weitere Anwendungen nach dem *Plug-and-Play* Prinzip zur Laufzeit hinzugefügt und entfernt werden. Jeder Teilnehmer kann sich für den Empfang von bestimmten Nachrichten registrieren. Die Umgebung unterstützt bei Simulation und realen Flügen das Arbeiten mit mehreren UAVs.

Paparazzi nennt seine Komponenten *Agents*. Die drei Hauptagents sind die GCS (Ground Control Station), der Server, sowie der Link inklusive des Flugzeugs.

- Das *Flugzeug* (die *Avionik*) sendet kontinuierlich Messwerte und Daten per Telemetrie zum Boden. Diese werden vom Link Agent empfangen und in das lokale Netzwerk weitergeleitet.
- Der *Server* empfängt alle über das Netzwerk gesendeten Daten und speichert sie in einer Logdatei. Je nach Einstellung im Server werden aus den Telemetriedaten neue Nachrichten synthetisiert und auf den Bus gelegt.

[1] The Ivy software bus - http://www.eei.cena.fr/products/ivy/ [Stand: 30.05.2014]

- In der *Ground Control Station* können alle relevanten Messwerte und Systemparameter eingesehen werden. Ebenfalls können Parameter für die autonome Navigation gesetzt werden. Sie werden im Netzwerk verteilt und als Telecontroldaten vom Link Agent zur Avionik gesendet.

Neben diesen Hauptagents ist eine Vielzahl weiterer Agents vorhanden. Durch diese gewinnt das System zahlreiche weitere Funktionalitäten. Nachfolgend ist eine Auswahl interessanter Agents gegeben:

- Ein Kommandozeilenprogramm zum Anzeigen aller im Netzwerk übermittelten Nachrichten zu Debug-Zwecken.
- Ein Log-Plotter, der aus den vom Server aufgezeichneten Nachrichten Diagramme erstellt.
- Ein live-Plotter, der die Daten und Parameter live in Diagrammen anzeigt.
- Ein Joystick Interface, mit dem Flugzeug und eventuelle Nutzlast in Realzeit gesteuert werden können.
- Ein Google Earth Plugin, welches die Wegpunkte und zurückgelegten Strecken live in Google Earth darstellen kann.
- Ein Flugplan Editor, mit dem Wegpunkte für die autonome Navigation erstellt und an das Flugzeug gesendet werden können.
- Ein Replay Agent, der alle vom Server aufgezeichneten Nachrichten ins Netzwerk einspeisen kann. Dies erlaubt die Wiedergabe eines vorher aufgezeichneten Fluges in Realzeit.

Simulation
Die oben genannten Agents bilden zusammen mit dem Netzwerkbus ein solides Grundgerüst zur Simulation und Vorabtests von Flügen und Fluggeräten. Abbildung 3.2 zeigt die Infrastruktur in einem Simulationsszenario.

Hier entfallen Flugzeug und Link Agent. Sie werden durch eine ausgeführte Flugphysik und eine Repräsentation der Avionik (*Microjet* Agent) ersetzt. Die restlichen Agents sind auch bei einer Simulation vorhanden und unterstützen weiterhin das Plug-and-Play Prinzip. Die Agents können dabei nicht zwischen einem realen und einem simulierten Flug unterscheiden. Speziell für die Simulation stehen weitere Agents und Anwendungen zur Verfügung. So kann zum Beispiel ein Agent namens *gaia* hinzugefügt werden. Gaia kann Umwelteinflüsse wie Wind, Luftfeuchtigkeit oder Luftdruck simulieren. Neben gaia kann ein sogenannter *Weather Agent*, welcher die aktuelle Wetterlage eines beliebigen Ortes auf der Erde in die Simulation einspeisen kann, eingesetzt werden. Zum Ausführen der Flugphysik stehen drei Flugsimulatoren zur Verfügung:

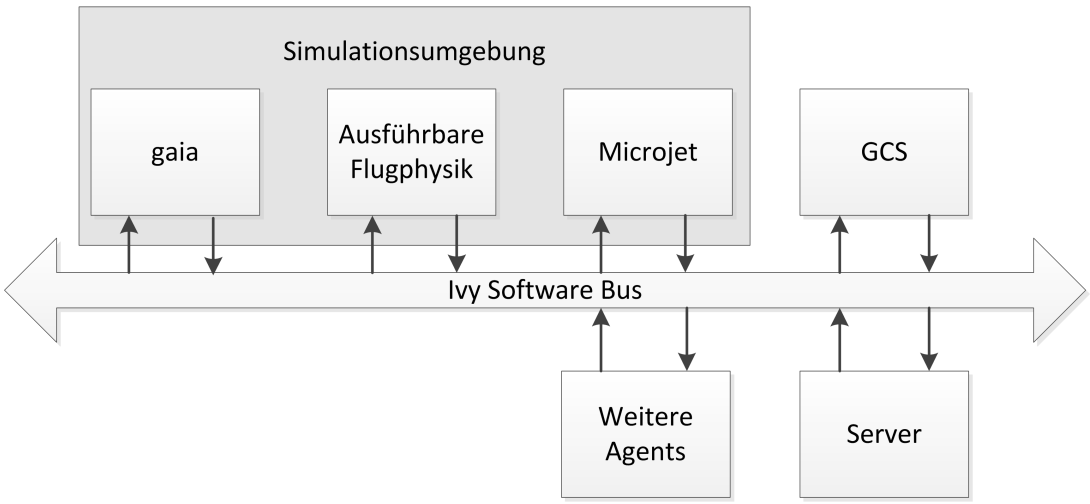

Abbildung 3.2.: Systemarchitektur von Paparazzi in der Simulation

- **sim**: einfache Simulationen von Starrflüglern
- **jsbsim**: erweiterte Simulationen von Starrflüglern
- **nps**: erweiterte Simulationen von Drehflüglern

Die Flugsimulatoren speisen den aktuellen Flugzustand in den Software-Bus. Welcher Flugsimulator geeignet ist und ob Daten der Atmosphäre bei der Simulation Verwendung finden, muss ein Benutzer bei seinen gegebenen Anforderungen und seinem Vorhaben selbst entscheiden.

Zur besseren Visualisierung des Simulationsgeschehens insgesamt kann der Flugsimulator FlightGear als Datensenke verwendet werden. Dieser dient allerdings lediglich zur Visualisierung der Fluglage und Position. Diese Daten werden durch einen Proxy Agent vom Bus an FlightGear weitergereicht.

Im Folgenden wird erläutert, wie die jeweiligen Simulationen ablaufen und wie sich einige Agents im Gegensatz zu einem realen Flug anders verhalten.

SIL-Szenario
Für die SIL-Simulation[2] wird ein Agent namens *Microjet* verwendet. Dieser enthält den Flugregler, sowie weitere Komponenten der Firmware. Die vom Flugsimulator eingespeisten Daten sind bereits aufbereitet, so dass die Firmwarekomponente zur Sensordatenaufbereitung nicht

[2]Paparazzi - Simulation, http://wiki.paparazziuav.org/wiki/Simulation [Stand: 30.05.2014]

ausgeführt wird. Der Microjet Agent wird mit zu erfliegenden Wegpunkten aus der Ground Control Station versorgt. Die von Microjet berechneten Steuersignale werden an den Flugsimulator gesendet, welcher daraufhin den nächsten Simulationsschritt berechnet.

Durch die Gesamtheit der Komponenten ergibt sich ein Regelkreis. Der Microjet Agent bildet den Regler, der Flugsimulator und gaia bilden die Regelstrecke. Die Sollwerte gibt der Benutzer mit der GCS und dem Flugplan Editor vor.

PIL-Szenario
Bei der PIL-Simulation[3] wird der Microjet Agent durch die reale Avionik ersetzt. Es handelt sich dabei um eine PIL-Simulation, da die Ein- und Ausgabe über den Telemetriekanal und nicht über die tatsächlichen Schnittstellen der Avionik ausgetauscht werden. Aus diesem Grund muss die Firmware speziell kompiliert werden. Mindestens die Firmwarekomponenten zur Sensordatenerfassung, Sensordatenaufbereitung und Aktorikansteuerung werden dabei nicht mitkompiliert.

Wie bei der SIL Simulation ergibt sich im PIL Szenario ein Regelkreis. Regelstrecke und Sollwertvorgabe bilden weiterhin der Flugsimulator und die GCS. Lediglich der Regler wird nicht durch den Microjet Agent, sondern durch die reale Avionik repräsentiert.

Zusammenfassung
Das Paparazzi Projekt bietet durch seine offene Busstruktur und seine vielen für spezielle Aufgaben dedizierten Agents sehr flexible Möglichkeiten. Viele der Agents können dem verteilten System im laufenden Betrieb hinzugefügt oder entfernt werden. Diese Eigenschaften ermöglichen umfangreiche Simulationen des Gesamtsystems, sowie auch einzelner Agents im Speziellen. Als positiver Nebeneffekt ergibt sich, dass SIL- und PIL-Simulation identisch aufgebaut sind und für den Benutzer dementsprechend auch ähnlich ablaufen.

ArduPilot

Das *ArduPilot* Projekt wurde 2007 gegründet und hat in den letzten Jahren eine große Community aufgebaut. Im Gegensatz zu Paparazzi richtet sich ArduPilot eher an Hobbybastler und Privatpersonen, als an Wissenschaftler und wissenschaftliche Einrichtungen. Dies spiegelt sich auch im Umgang des Systems wider. ArduPilot setzt viele Kenntnisse aus der Programmierung im Allgemeinen und im Umgang mit Linux voraus, während Paparazzi einfacher zu handhaben ist.

Als Hardwareplattform wird eine kontinuierlich weiterentwickelte Arduino Platine verwendet. Die Komponenten zur Sensordatenerfassung und Sensordatenaufbereitung können auf einer externen Arduino Platine (*ArduIMU*) ausgelagert werden. Das System teilt sich in diesem Fall

[3] Paparazzi - HITL, http://wiki.paparazziuav.org/wiki/HITL [Stand: 30.05.2014]

in die Subsysteme Autopilot und Sensorboard. Auf dem Sensorboard werden die Firmwarekomponenten zur Sensordatenerfassung und Sensordatenaufbereitung ausgeführt. Durch Aufteilung können beide Subsysteme isoliert voneinander getestet werden, wodurch sich die Komplexität der Testumgebung für den Flugregler verringert.

Wie auch Paparazzi unterstützt ArduPilot SIL- und PIL-Simulationen. Jedoch unterscheidet sich die Systemarchitektur des verteilten Systems zwischen Flugzeug und Ground Control Station erheblich. Paparazzi verwendet ein Software-Bussystem auf der Bodenstation, während bei ArduPilot viele der Funktionen zentralisiert in der Ground Control Station vorhanden sind.

SIL-Szenario

Bei der SIL-Simulation[4] wird im Gegensatz zu Paparazzi nicht nur eine Teilmenge, sondern die komplette Firmware auf dem Host System (Linux Betriebssystem) ausgeführt. Dadurch ergibt sich ein Test der gesamten Firmware. Ein Test der kompletten Firmware erzwingt, dass die Hardware des in der Avionik verwendeten Prozessors auf Registerebene emuliert wird. Firmware und Prozessoremulation werden in einem *Desktop Executable* (*DE*) genannten Programm ausgeführt.

Abbildung 3.3 zeigt die Systemarchitektur inklusive der Kommunikationsstruktur der SIL-Simulation.

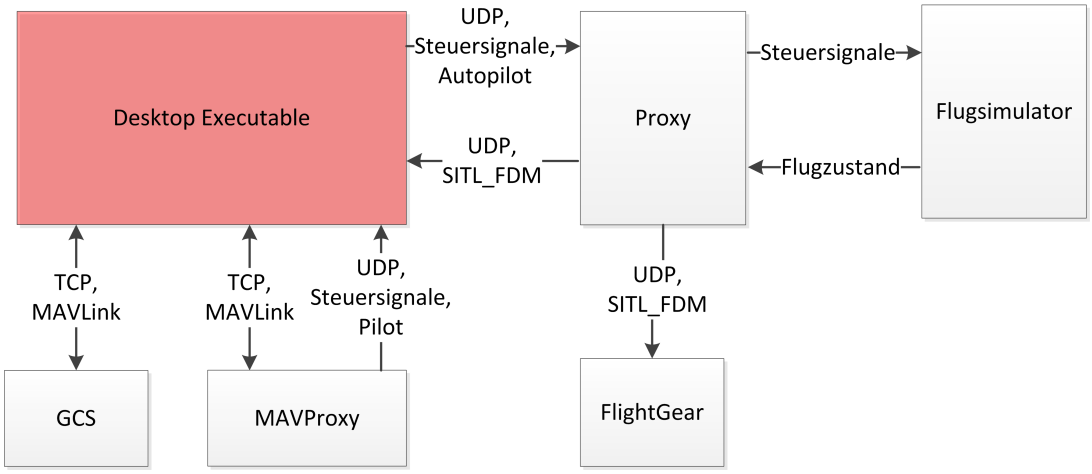

Abbildung 3.3.: SIL Simulationsumgebung von ArduPilot

Die zentrale Komponente bildet das in rot dargestellte Desktop Executable. Über die links unten abgebildete GCS können wie im echten Flug in Realzeit Daten visualisiert und Wegpunkte

[4] ArduPilot - Software In the Loop Simulator, http://plane.ardupilot.com/wiki/common-sitl-software-in-the-loop-simulator/ [Stand: 30.05.2014]

editiert werden. Die *MAVProxy* Anwendung ermöglicht die Eingabe von RC-Steuersignalen an die Simulation. An diesen MAVProxy kann auch eine weitere GCS verbunden werden. Für die logische Kommunikation der Komponenten untereinander wird das *MAVLink* Protokoll[5], welches unabhängig von ArduPilot entwickeltes wird, verwendet. Es dient der Kommunikation zwischen beliebigen Autopiloten und Ground Control Stations. Das offene Protokoll entkoppelt beide Systeme voneinander. GCS und MAVProxy sind jeweils über eine TCP Verbindung mit dem Desktop Executable verbunden. Im realen Flug wird für beide Verbindungen RS232, bzw. UART verwendet.

Die Flugphysik wird je nach Trägerkonfiguration von verschiedenen Flugsimulatoren berechnet. Für Multicopter Simulationen steht ein Python Skript zur Verfügung, welches eine rudimentäre Flugphysik simuliert. Für Starrflügler steht wie auch bei Paparazzi das Kommandozeilenprogramm jsbsim zur Verfügung. Bei beiden Lösungen wird je eine Adapteranwendung zur Kommunikation zwischen Flugsimulator und Desktop Executable verwendet. Die ausgetauschten Daten sind in Richtung Simulator die vom Flugregler berechneten Steuersignale für die Servomotoren (als emulierte PWM Werte, keine virtuellen Steuersignale). In Richtung Executable wird der Flugzustand (*SITL_FDM*) übertragen. Der Zustand beinhaltet unter anderem die Fluglage, die Position im Weltkoordinatensystem, die Drehraten und Beschleunigungen im körperfesten Koordinatensystem, sowie die Airspeed. Das Desktop Executable muss daraus die Sensordaten emulieren und der eingebetteten ArduPilot Firmware auf Register Ebene zur Verfügung stellen.

Um Simulationen zu reproduzieren steht ein Python Skript namens *FakePos* zur Verfügung. Dieses kann genutzt werden, um deterministische Sensordaten zu generieren. Als Steigerung dazu kann mit einem Python Skript namens *AutoTest* ein automatischer Test durchgeführt werden. Dieses startet alle benötigten Komponenten und lässt das System eine vordefinierte Route abfliegen. Das Skript gibt am Ende des Tests das Testergebnis aus. Dieses trifft allerdings keine Aussagen über die Qualität des Fluges. Beide Python Skripte haben jedoch begrenzten Nutzen, da sie lediglich rudimentäre Funktionen bietet und nur sehr statisch eingesetzt werden können.

Zur besseren Visualisierung der Simulation steht wie bei Paparazzi auch FlightGear zur Verfügung. Auch hier dient FlightGear lediglich der Visualisierung und kann nicht als Regelstrecke verwendet werden.

PIL-Szenario

Bei ArduPilot unterscheidet sich die PIL-Simulation[6] in vielerlei Hinsicht von der SIL-Simulation. Anders als beim SIL-Szenario werden die Sensorwerte hier nicht über die tatsäch-

[5] MAVLink - Micro Air Vehicle Communication Protocol, http://qgroundcontrol.org/mavlink/start [Stand: 30.05.2014]

[6] ArduPilot - How to set up a full hardware-in-the-loop simulation with Xplane, http://code.google.com/p/ardupilot-mega/wiki/Xplane [Stand: 30.05.2014]

lichen Schnittstellen eingelesen. Stattdessen werden die aufbereiteten Sensordaten über den Telemetriekanal an die Avionik gesendet. Somit ist die Firmware beim SIL-Szenario paradoxerweise näher an der in realen Flügen verwendeten, als beim PIL-Szenario.

Die Systemarchitektur der PIL-Umgebung ist in Abbildung 3.4 dargestellt. Die Avionik bildet den Regler. Der Flugsimulator repräsentiert die Regelstrecke. Die Sollwerte werden von der RC-Fernbedienung und der GCS durch zu erfliegende Wegpunkte geliefert. Die GCS vermittelt zwischen Regler und Strecke.

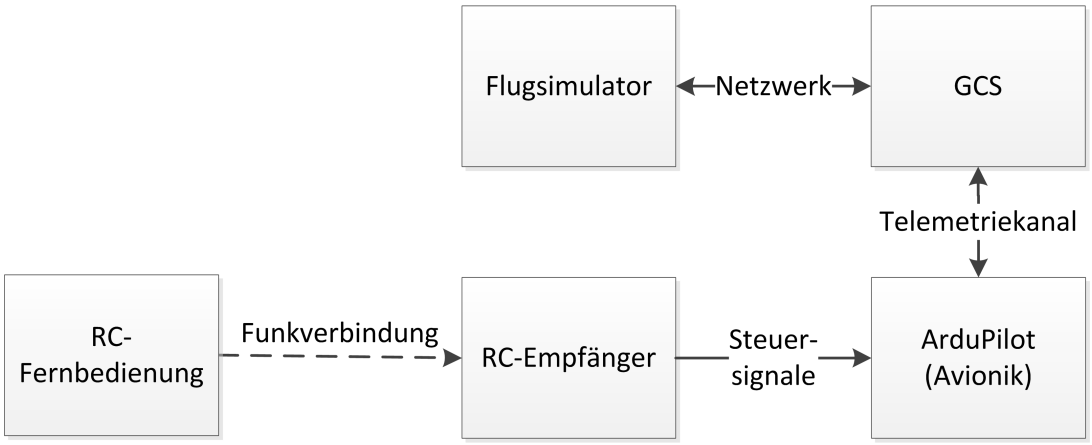

Abbildung 3.4.: PIL Simulationsumgebung von ArduPilot

Als Flugsimulatoren können unter Anderem X-Plane und FlightGear verwendet werden. Beide Simulatoren werden durch Netzwerkverbindungen mit der GCS verbunden. FlightGear unterstützt TCP und UDP, X-Plane hingegen nur UDP. Bei keinem der Simulatoren wird die Verbindung über Plugins realisiert. Stattdessen werden in beiden Fällen die nativen Netzwerkschnittstellen der Simulatoren verwendet. Neben X-Plane und FlightGear kann der Modellflugsimulator AeroSimRC genutzt werden. Dieser bietet keine native Netzwerkschnittstelle zur Kommunikation an und wird daher um ein selbst entwickeltes Plugin erweitert. Als vierte Option kann jsbsim genutzt werden. Jede der vier Flugsimulatoren muss von der GCS unterschiedlich behandelt werden, wodurch sich die Anwendungslogik der GCS verkompliziert.

Zusammenfassung
Die Simulationsmöglichkeiten von ArduPilot können ein weites Feld an Anforderungen und Wünschen der Benutzer abdecken. Durch die verschiedenen Flugsimulatoren ist er frei in der Wahl der verwendeten Regelstrecke.

Was im Gegensatz zu Paparazzi negativ auffällt, ist die enge Kopplung der zur Simulation benötigten Komponenten. Jede einzelne dieser Kopplungen ist als eine Punkt-zu-Punkt Verbindung realisiert. Eine neue Anwendung erfordert so auch eine neue Verbindung zu den bereits

vorhandenen Anwendungen. Besonders im SIL-Szenario erkennt man die Ausmaße dieses Umstands. Hier sind viele Handgriffe des Benutzers nötig, um das verteilte System aufzubauen.

OpenPilot

Das *OpenPilot* Projekt wurde 2009 gegründet und richtet sich primär an Privatanwender. Die Avionik steht in zwei verschiedenen Hardwareplattformen mit jeweils unterschiedlicher Firmware und Konzepten zur Verfügung. Die vorliegende Beschreibung basiert auf der aktuelleren *Revolution* Plattform.

Die logische Kommunikation zwischen der GCS und der Avionik wird über *UAVTalk*[7] abgehandelt. Dieses Protokoll wurde von OpenPilot für die Kommunikation von Komponenten innerhalb eines UAS entwickelt und ist für jedermann frei zugänglich. Es dient der Übertragung von *UAVObjects*. Ein UAVObject ist ein Datenobjekt, welches unterschiedliche Daten enthalten kann. Dies können zum Beispiel gemessene Sensorwerte, Wegpunkte oder interne Zustände sein.

SIL-Szenario
Die Firmware der Avionik besteht aus zwei übereinanderliegenden Softwareschichten. Die wie das Projekt genannte *OpenPilot* Schicht enthält hardware- und plattformunabhängigen Source Code. Die darin enthaltenen Softwaremodule zur autonomen Navigation, Stabilisierung, Sensordatenverarbeitung usw. setzen auf die darunterliegenden Schicht *PiOS* auf. PiOS ist Hardwareabstraktionsschicht und Echtzeitbetriebssystem zugleich.

Die Trennung der Logik und Algorithmen vom Hardwarezugriff ermöglicht es, die gesamte OpenPilot Schicht auf einfache Art und Weise in einer SIL-Simulation[8] zu testen. Die PiOS Schicht wurde für den auf dem Board verwendeten Mikrocontroller STM32F4, sowie in Linux, Windows und Mac OS X implementiert. Somit können die kritischen Firmwarekomponenten bereits auf einem PC getestet werden. Aus der OpenPilot Schicht zusammen mit der entsprechenden Hardwareabstraktionsschicht des Betriebssystems kann für alle drei Betriebssysteme ein *SIL Executable* kompiliert werden. Die GCS ist ebenfalls plattformunabhängig und kann auf allen drei Betriebssystemen verwendet werden.

Ähnlich wie bei ArduPilot dient die GCS als Vermittler zwischen Flugsimulator und SIL Executable. Genauso ist auch Kommunikation vom Flugsimulator abhängig. GCS und SIL Executable kommunizieren über drei einzelne Punkt-zu-Punkt UDP Verbindungen miteinander. Hierbei dient ein Kanal der Übertragung von Telemetriedaten. Ein anderer übermittelt den Flugzustand in Richtung SIL Executable, sowie die berechneten Steuersignale in die Gegenrichtung

[7] UAVTalk, http://wiki.openpilot.org/display/Doc/UAVTalk [Stand: 30.05.2014]
[8] OpenPilot - OpenPilot SiTL, http://wiki.openpilot.org/display/Doc/OpenPilot+SiTL [Stand: 30.05.2014]

zur GCS. Ein dritter Kanal wird verwendet, um die simulierten GPS Daten an das SIL Executable zu senden. Jeder Kanal stellt eine an der Avionik verwendete physikalische Schnittstelle dar. An die Avionik werden das Telemetriemodul und der GPS Empfänger jeweils über RS232 verbunden. Die Steuersignale werden mittels PWM an die Servomotoren und die Triebwerke geleitet.

Als Flugsimulatoren stehen FlightGear, AeroSimRC und X-Plane zur Verfügung. Alle drei Simulatoren verwenden eine TCP bzw. UDP Schnittstelle zur Kommunikation. Bei AeroSimRC wird dies über ein eigens entwickeltes Plugin realisiert.

PIL-Szenario
Bei einer PIL-Simulation[9] wird lediglich das SIL Executable gegen die Avionik ausgetauscht. In der Benutzung weist sie keine Unterschiede zur SIL-Simulation auf. Aus technischer Sicht fallen die Kanäle zur Anbindung des GPS Empfängers und die Rückführung der Steuersignale weg. Stattdessen werden zwischen Avionik und GCS alle Daten über den Telemetriekanal ausgetauscht. Dabei wird das UAVTalk Protokoll verwendet.

PX4

Das *PX4* Projekt wurde 2009 an der Eidgenössischen Technischen Hochschule Zürich gegründet. Es hat zum Ziel, ein Open Source Autopilotensystem für akademische und industrielle Zwecke, sowie Hobby-Benutzer bereitzustellen. Die derzeitig aktuellste Hardwareplattform ist der *pixhawk Autopilot*.

Als Grundlage der Firmware wird das POSIX konforme Echtzeitbetriebssystem NuttX verwendet. Es unterstützt eine Vielzahl von Dateisystemen, die Berkeley Socket API, verschiedene Scheduling Algorithmen, USB als Host und Device Schnittstelle, die graphische Ausgabe auf LCD Displays und bietet Treiber für verschiedene Kommunikationsschnittstellen für eingebettete Systeme an (CAN, I2C, SPI, etc.). Es kann auf verschiedenen Mikrocontrollern, wie zum Beispiel *ARM*, *Atmel* oder *Renesas* basierten Mikrocontrollern ausgeführt werden. Ein Benutzer kann über eine Shell mit dem System arbeiten.

Die Softwaremodule der Firmware von PX4 laufen als eigenständige Prozesse von NuttX. Dies führt dazu, dass eine Interprozesskommunikation (*IPC*) benötigt wird. Diese ist im Allgemeinen schwerer zu realisieren als eine Interthreadkommunikation. Die IPC wird mittels eines Softwarebusses inklusive eines Object Request Brokers realisiert. Aufgrund der hohen Komplexität der Portierung von NuttX inklusive der benötigten IPC unterstützt pixhawk, bzw. das PX4 Projekt aktuell keine SIL Simulation.

[9] OpenPilot - HiTL Plugin, http://wiki.openpilot.org/display/Doc/HiTL+Plugin [Stand: 30.05.2014]

PIL-Szenario
Die PIL-Simulation[10] ähnelt dem Konzept von ArduPilot, bei dem die GCS zwischen Flugsimulator und Avionik vermittelt. Zur Kommunikation zwischen GCS und Avionik wird logisch das MAVLink Protokoll und physikalisch der Telemetriekanal (RS232) verwendet. Als Flugsimulatoren werden X-Plane, FlightGear, sowie jsbsim unterstützt. Die Anbindung an X-Plane und FlightGear ist über die nativen Netzwerkschnittstellen der Simulatoren realisiert. Zur Vermittlung zwischen GCS und jsbsim wird eine Adapteranwendung verwendet, die MAVLink Pakete verarbeiten kann.

Die Simulation kann in zwei Varianten ausgeführt werden:

- **sensor-level**: Das Firmwaremodul zur Sensordatenerfassung wird ersetzt, das zur Sensordatenaufbereitung bleibt erhalten.
- **state-level**: Das Firmwaremodul zur Sensordatenerfassung wird ersetzt, das zur Sensordatenaufbereitung wird ebenfalls ersetzt.

Je nach Variante kann ein Benutzer somit entscheiden, ob das Firmwaremodul zur Sensordatenaufbereitung mitgetestet wird oder nicht. Diese Auswahlmöglichkeit bietet gegenüber der Konkurrenz einen Vorteil.

3.1.2. Anmerkung zu Open Source Projekten

Die oben beschriebenen Projekte Paparazzi, ArduPilot, OpenPilot und PX4 bieten eine Reihe von Testmöglichkeiten an. In den Projekten werden die Systeme **von** einer Community, **für** eine Community entwickelt. Jeder Benutzer eines solchen Autopilotensystems hat sein eigenes oder sogar mehrere eigene Modellflugzeuge, mit jeweils entsprechend unterschiedlicher Flugphysik. Von den Flugsimulatoren werden jedoch nur eine geringe Auswahl an Flugzeugen und deren entsprechender Flugphysik angeboten. Lediglich PX4 und Ardupilot empfehlen für die Simulationen bestimmte Modelle, bzw. liefern eigene für die Flugsimulatoren mit. Zum Beispiel bietet PX4 die Modelle EasyStar und Zagi Flying Wing für Starrflügler, sowie ein QRO X Quadrocopter Modell für die Simulation mit X-Plane an. Neben der Flugphysik sind auch die Aktuator- und Sensormodelle für jedes Flugzeug unterschiedlich.

Für den Entwurf hochwertiger Reglerparameter müssen den Benutzern Flugphysik, Sensormodell und Aktuatormodell ihrer Modellflugzeuge bekannt sein. Dies ist jedoch meist nicht der Fall, da die Benutzer in den seltensten Fällen über die nötigen Kenntnisse verfügen. Dieser Umstand führt dazu, dass die Simulationen für die meisten Benutzer lediglich als Validierung der Firmware und nicht zur Ermittlung von hochwertigen Reglerparametern dienen können.

[10] PX4 Autopilot - Hardware in the Loop Simulation Setup, http://pixhawk.org/users/hil [Stand: 30.05.2014]

Weiterhin ist anzumerken, dass die Projekte den Begriff HIL-Simulation verwenden. Nach der in 2.6 gegebenen Definitionen von XIL-Umgebungen handelt es sich allerdings um PIL-Simulationen. Daher wurden sie hier auch als solche bezeichnet.

3.1.3. Projekte in Wissenschaft und Forschung

Aufgrund der interdisziplinären Natur von UAVs werden solche Systeme mit wachsender Beliebtheit von vielen Universitäten zu Lehr- und Forschungszwecken entwickelt und eingesetzt. Die Avionik der Flugsysteme wird dabei häufig selbst entwickelt.

Nachfolgend werden einige Avioniksysteme und deren Testkonzepte vorgestellt. Die Gesamtsysteme weisen aus abstrahierter Sicht meist die gleichen Konzepte und Komponenten sowohl untereinander, als auch zu den in 3.1.1 aufgeführten Open Source Projekten auf. Aus diesem Grund wird nur auf Unterschiede zu bereits beschriebenen Konzepten eingegangen. Einige weitere hier nicht dargestellte Systeme findet der Leser in [Adiprawita u. a. (2007)], [Zhang und Li (2013)], [dos Santos u. a. (2011)] oder [Ribeiro und Oliveira (2010)].

Bei Projekten aus Wissenschaft und Forschung ist zu bedenken, dass zur Umsetzung solcher Systeme meist ein größeres finanzielles Kapital und mehr Know How seitens der Anwender zur Verfügung steht. Neben der Avionik werden auch die Träger selbst entwickelt. Diese sind häufig Unikate und dienen wissenschaftlichen Zwecken. Die Testsysteme werden anders als in 3.1.2 auch zum Entwurf von Flugreglern verwendet. Aus diesem Grund setzen die hier vorgestellten Projekte zum Teil hochwertige HIL-Testsysteme, anstelle der einfacheren PIL-Umgebungen ein.

IFSys

IFSys (Intelligentes Fliegendes System) ist ein 2006 von Studierenden der TU Berlin gegründetes Forschungsprojekt, dessen Ziel es ist, ein UAS zu entwickeln. Dazu gehört neben dem Träger selbst auch eine Hardware- und Softwareplattform inklusive einer entsprechenden Bodenstation. Der Name des Versuchsträgers lautet *Airborne Laboratory for EXperiments on Inflight Systems* (*ALEXIS*).

Die Avionik (Flight Control Computer, *FCC*) basiert auf einem Embedded PC mit x86 Prozessor [Hamann und Hoffmann]. Als Betriebssystem wird Windows XP Embedded eingesetzt. Das verwendete Sensorkonzept ähnelt dem von AES in vielerlei Hinsicht. Es werden ebenfalls ein AHRS, ein GPS-Empfänger sowie eine Luftdatensonde eingesetzt. Außerdem kommt ein Ultraschall-Entfernungsmesser für präzisere Höhenmessungen während Landungen zum Einsatz. Die Sensoren sind allesamt über RS232 Schnittstellen verbunden. Die Eingabe der

Steuersignale des Piloten erfolgt über PWM. Die Servos zum Bewegen der Steuerflächen werden ebenfalls über PWM angesteuert.

Die Testumgebung ist deutlich fortgeschrittener als bei den oben beschriebenen Open Source Projekten. Bei einer Simulation kommt kein handelsüblicher Flugsimulator zum Einsatz. Stattdessen wird eine in Simulink modellierte Flugphysik eingesetzt.

SIL-Szenario

Die SIL-Simulation [Hamann und Hoffmann (2011)] besteht aus vier Softwarekomponenten. Diese entsprechen denen eines Standardregelkreises. Abbildung 3.5 zeigt eine Übersicht der Komponenten. Neben dem Regler (*Flugreglersoftware*) und der Regelstrecke (*Flugmechanische Simulation*) beinhaltet die Simulation auch das Sensormodell (*Sensoremulation*) und das *Aktuatormodell*. Durch die Sensoremulation und das Aktuatormodell müssen in der Flugreglersoftware, neben den Reglern selbst, auch die Komponenten zur Sensordatenerfassung, Sensordatenaufbereitung und Steuerflächenzuordnung ausgeführt werden. Sie werden dadurch unweigerlich mitgetestet.

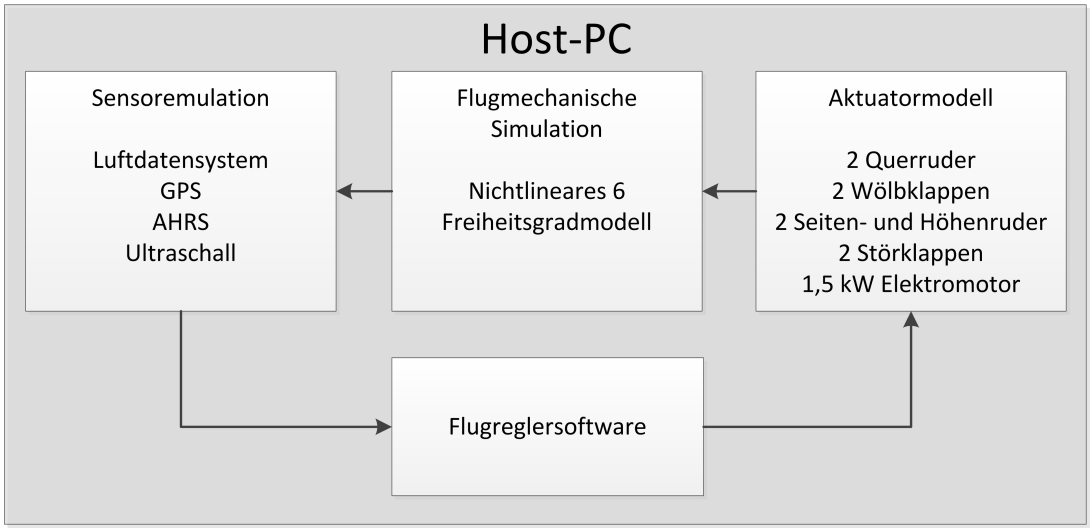

Abbildung 3.5.: SIL Simulationsumgebung von ALEXIS

HIL-Szenario

Im HIL-Szenario wird die Flugreglersoftware durch die reale Avionik ersetzt. Wie in 2.6 beschrieben, muss die Testumgebung in Echtzeit ausgeführt werden. Hierfür wird das Echtzeitcomputersystem *dSPACE PX10 Expansion Box* der Firma dSPACE eingesetzt. Diese Box liest die vom FCC generierten PWM Steuersignale ein und gibt die emulierten Sensordaten jeweils per RS232 aus. Der FCC kann bei diesem Ansatz nicht zwischen realem Flug und Simulation

unterscheiden. Die Testumgebung erfüllt damit alle Voraussetzungen, dem FCC eine realitätsnahe Simulation bieten zu können.

Das Konzept wurde mit einem Iron Bird umgesetzt, welcher auf der ILA 2012 vorgestellt wurde[11].

ROCS

Das Institut für Flugmechanik und Flugregelung der Universität Stuttgart hat ein Autopilotensystem mit dem Namen *Research Onboard Computer System* (*ROCS*) entwickelt[12]. Dieses kann zur autonomen Steuerung von Starrflügler-, Multicopter- und Helikopter-UAVs, sowie Über- und Unterwasser Fahrzeugen verwendet werden. Das Hardwaresystem besteht aus den drei Modulen *Onboard Computing Module* (*OCM*), *Interface Module* (*IM*) und *Hardware in the Loop Module* (*HILIM*). Wobei OCM und IM die Avionik bilden.

Das OCM beinhaltet einen 720MHz schnellen Prozessor (ARM Cortex A8 Kern) und ein Xilinx Spartan3 FPGA. Die Firmware kann je nach Anforderung des Benutzers auf beide Bauteile verteilt werden. Das FPGA ist mit einer generischen 80-Pin Schnittstelle ausgestattet. Über diese werden OCM und IM miteinander verbunden. Das IM beinhaltet die applikationsspezifischen Sensoren, bzw. diese werden an das IM angeschlossen. Je nach Anwendungsanforderung werden unterschiedliche IM Platinen verwendet.

Das HILIM ist mit einem FPGA ausgestattet und kommt bei Hardware-In-the-Loop-Simulationen zum Einsatz. Es verbindet den Host PC mit der Avionik. Die in Simulink modellierte Flugphysik wird auf dem Host PC ausgeführt. Die Ein- und Ausgabedaten werden über eine USB Verbindung mit dem HILIM ausgetauscht. Dieses stellt die Daten in Echtzeit dem IM zur Verfügung. Hierbei wird ebenfalls eine Sensoremulation, der im Flug real existierenden Sensoren durchgeführt. Aufgrund dieser Eigenschaft kann das ROCS ähnlich wie der FCC von ALEXIS in einer realitätsnahen Umgebung getestet werden.

Zur besseren Visualisierung der Simulation wird der Flugzustand auf dem Host PC selbst, bzw. auf einem zweiten PC in FlightGear dargestellt.

[11] IFSys - IFSys auf der ILA 2012, http://www.fmra.tu-berlin.de/menue/forschung/projekte/flugregelung/ifsys_menue/news/ifsys_auf_der_ila_2012/ [Stand: 30.05.2014]

[12] ROCS - iFR Research Onboard Computer System, http://www.ifr.uni-stuttgart.de/webseiten/computersystem/computersystem.html [Stand: 30.05.2014]

UAV System der NASA

Im Folgenden wird die Umsetzung einer Hardware-in-the-Loop Simulationsumgebung eines namentlich nicht benannten UAV Systems der NASA beschrieben [Mueller (2007)].

Der entwickelte Träger ist ein Otto-Motor betriebenes UAV, das bis zu vier Stunden per Tankfüllung fliegen kann. Die Spannweite beträgt etwa vier Meter. Den Kern der Avionik bildet ein Prozessorboard mit einem Pentium x86 Prozessor und 256MB RAM. Dieser führt das Echtzeitbetriebssystem QNX aus. Zur Bestimmung der Fluglage kommt ein AHRS zum Einsatz. Die Position wird mittels GPS erfasst. Beide Sensoren sind in einem *WAAS* (*Wide Area Augmentation System*) Empfänger vereint. Dieser WAAS Empfänger ist mittels RS232 mit der Prozessorplatine verbunden. Die vom Autopiloten berechneten Steuerwerte der Servos werden über eine RS232 Verbindung an ein Servocontroller Board gesendet und dort in Form von PWM Signalen an die Servos weitergegeben. Die Servoausschläge sowie die Motordrehzahl werden mittels Hall Sensoren gemessen. Diese geben die gemessenen Werte als analoge Spannungen an eine Analog-Digital-Wandler Karte weiter. Die Karte ist über einen PC104 Bus mit dem Prozessorboard verbunden. Anstellwinkel und Schiebewinkel werden durch Windfahnen gemessen. Die Airspeed durch ein Staurohr. Zusätzlich wird der statische Luftdruck über ein Pitotrohr ausgelesen. Dieser ermöglicht die Bestimmung der Höhe. Airspeed, Luftdruck, Anstellwinkel und Schiebewinkel ergeben zusammen die Luftdaten. Sie werden ebenfalls allesamt von der AD-Wandler Karte eingelesen.

Simulation

Zur Simulation wurde eine nichtlineare Flugphysik des Trägers in Simulink entworfen. Dieses beinhaltet auch ein Sensormodell und ein Aktuatormodell. Aus dem nichtlinearen Modell wurde ein lineares Modell entworfen, welches der Entwicklung des Flugreglers diente. Das lineare Modell und die Regler wurden durch SIL-Simulationen in iterativen Schritten sukzessiv verfeinert, bis Regler und Regelstrecke den Anforderungen genügten.

Für eine HIL Simulationsumgebung wurden drei Alternativen evaluiert. Die erste bestand darin, die Avionik und Simulink mittels RS232 zu verbinden. Dieser Ansatz wurde verworfen, da in der verwendeten Simulink Version weder ein entsprechender RS232 Treiber, noch die benötigte Echtzeitausführung unterstützt wurde. Der zweite Ansatz sah vor, das gesamte Simulationsmodell in Matlab Skripten auszuführen. Dieser wurde zwar umgesetzt, jedoch aufgrund einer unzureichenden Performance fallengelassen. Der letztendlich realisierte Ansatz unterscheidet sich in vielerlei Hinsicht von den bisher beschriebenen Konzepten. Bei diesem wurde das Simulationsmodell in C geschrieben und für QNX kompiliert. Die hohe Prozessorleistung ermöglichte das simultane Ausführen von Regler und Flugphysik auf der Avionik. Dieses Vorgehen behebt die Probleme der Schnittstellen, sowie der nicht vorhandenen Echtzeit von Simulationen in Simulink. Ein weiterer Vorteil dieses Ansatzes ergibt sich durch die Schleife bestehend aus Servos und Hall Sensoren. Die Hall Sensoren mussten nicht simuliert werden, da hierfür die reale Schleife verwendet werden konnte.

3.2. Flugsimulatoren

Für das gegebene Problem muss eine adäquate Regelstrecke entwickelt, bzw. ausgewählt werden. Hierfür eignen sich am Markt verfügbare Flugsimulatoren, da diese die erforderlichen mathematischen Modelle der Flugphysik bereits beinhalten. Im Folgenden werden geeignete Flugsimulatoren beschrieben.

Die Eignung als Regelstrecke stellt zwei fundamentale Anforderungen an die Flugsimulatoren. Zum einen müssen diese unterstützen, dass der interne Flugzustand ausgelesen werden kann und dass die Steuersignale des Reglers zur Steuerung des Modells innerhalb des Flugsimulators verwendet werden können. Hierfür muss der Simulator eine Schnittstelle zur Interprozesskommunikation bieten. Diese kann etwa über eine native Netzwerkschnittstelle oder über ein Plugin realisiert sein. Weiterhin muss der Simulator eine Möglichkeit bieten, das physikalische Verhalten des simulierten Flugzeugs zu konfigurieren. Ohne diese Möglichkeiten kann das simulierte Modell nicht an den AC2030 angepasst werden, wodurch sich ein vorzeitiger Reglerentwurf erschwert.

Viele Modellflugsimulatoren wie etwa *Phoenix RC*, *RealFlight* oder *Reflex XTR* erfüllen diese Anforderungen nicht oder nur unzureichend und scheiden damit schon in der Vorauswahl aus. Microsoft hat die Entwicklung seiner Flugsimulatoren Reihe (u.a. *Microsoft Flight Simulator 2004*, *Microsoft Flight Simulator X*, *Microsoft Flight*) vor einigen Jahren eingestellt. Daher scheiden diese ebenfalls im Voraus aus.

3.2.1. FlightGear

FlightGear ist ein weltweit verbreiteter Open Source Flugsimulator. Das bei der Entwicklung verfolgte Ziel ist ein: "*... anspruchsvolles und freies Flugsimulator-Framework für wissenschaftliche und akademische Zwecke, Pilotentraining, sowie als Werkzeug für industrielle Entwicklungen ...*"[13] bereitzustellen. Die Entwicklung begann 1996. Unterstützt werden Windows, Linux und Mac OS X Betriebssysteme.

Die aktuelle Version 3.0 bietet einige für das gegebene Problem interessante Funktionen und Eigenschaften:

- **Erweiterbarkeit**: Das komplette Projekt ist Open Source und damit frei verfügbar. Falls erforderlich oder gewünscht kann FlightGear an beliebige Anforderungen angepasst werden.

[13] vgl. FlightGear - Introduction to FlightGear, http://www.flightgear.org/about/ [Stand: 30.05.2014]

- **Native Netzwerkunterstützung**: FlightGear kann komplett über eine Netzwerkverbindung gesteuert werden. Es können alle internen Simulatorzustände ausgelesen und bis auf wenige Ausnahmen (für diese Arbeit bis auf weiteres uninteressante) können alle Werte von außen verändert werden. Somit wird für die geforderte Interprozesskommunikation kein Plugin benötigt.
- **Verschiedene Simulator-Backends**: Das grafische Frontend von FlightGear ist vom verwendeten Simulator-Backend getrennt, wobei das am häufigsten verwendete *jsbsim* ist. Weitere sind *yasim* und *uiuc*. Außerdem besteht die Möglichkeit, eine externe Flugphysik zu verwenden. Diese kann über die Netzwerkschnittstelle mit FlightGear kommunizieren.
- **Umfangreiches Wettermodell**: Das Wettermodell kann über viele Parameter eingestellt werden, sodass sich Regen, Sturm oder ein Gewitter ergibt. Es wurde von Deters et al. verwendet, um Effekte von *Icing* simulieren und analysieren zu können [Deters u. a. (2006)].
- **Netzwerkpartien**: Mehrere FlightGear Instanzen können an einer gemeinsamen Netzwerkpartie teilnehmen.
- **Screen Relay**: Eine Simulation kann auf mehreren Monitoren dargestellt werden. Diese müssen nicht notwendigerweise am gleichen Computer angeschlossen sein. Verschiedene Kameraansichten oder Teile einer großen Ansicht können über das Netzwerk auf andere FlightGear Instanzen verteilt werden.

jsbsim

Bei *jsbsim* handelt es sich um ein eigenständiges Softwareprojekt und nicht um einen Teil von FlightGear. Jedoch wird es von diesem als standardmäßiges Simulator-Backend verwendet. Wie auch FlightGear ist jsbsim Open Source und kann frei verwendet werden.

Aus technischer Sicht ist jsbsim eine Funktionsbibliothek zur Berechnung der Flugphysik von Flugzeugen, Helikoptern, Raketen oder anderer Fluggeräte. Die aerodynamischen Derivativa eines Flugmodells können als XML-Dateien bereitgestellt werden. Über diese werden auch die Steuerflächenkonfigurationen und die damit zusammenhängende Steuerflächenzuordnung angegeben. Weiterhin kann das Aktuatormodell mittels der XML-Dateien konfiguriert werden.

Jsbsim kann als statische Softwarebibliothek in eine Anwendung integriert werden. Dies ist zum Beispiel bei FlightGear der Fall. Weiterhin steht ein Kommandozeilen-Frontend zur Verfügung. Auf diese Weise kann jsbsim eigenständig ausgeführt werden. Das Frontend bietet verschiedene Möglichkeiten der Interprozesskommunikation an. Unter anderem werden TCP und UDP unterstützt.

Aufgrund der umfangreichen Funktionen und Möglichkeiten, die FlightGear in Verbindung mit jsbsim aufweist, wird die Kombination von vielen wissenschaftlichen und akademischen Projekten für XIL-Simulationen verwendet [Chao u. a. (2013), Zhang und Li (2013), Abdunabi (2006), Park u. a. (2008).

3.2.2. X-Plane

X-Plane ist ein hochgradig professioneller Flugsimulator des US-amerikanischen Softwareentwicklers Austin Meyers. Dieser hat ihn speziell als Industriewerkzeug und weniger als Unterhaltungssoftware entwickelt[14]. Die Kosten für die aktuelle Version 10 liegen bei 59.99 US-Dollar. X-Plane steht für die Betriebssysteme Windows, Linux und Mac OS X zur Verfügung.

Der besondere Unterschied zu anderen Flugsimulatoren liegt in der Berechnung der Flugphysik. Anders als etwa jsbsim benötigt X-Plane keine aerodynamischen Derivativa des Flugmodells. Stattdessen findet die sogenannte *Blade element theory* Anwendung [Meyers (2013), Adiprawita u. a. (2007)]. Bei dieser werden keine vorher festgelegten Derivativa des Fluggeräts herangezogen. Stattdessen werden diese in jedem Simulationsschritt dynamisch selbst berechnet. Das simulierte Flugzeug wird von X-Plane in einzelne Komponenten zerteilt. Bei Flugzeugen sind dies meist die Tragflächen, die Leitwerke, die Steuerflächen, die Triebwerke und der Rumpf. Diese Komponenten werden weiter in einzelne Segmente unterteilt. In einem Simulationsschritt werden für jedes Segment die individuell auftretenden Kräfte und Momente anhand dessen Geometrie und Gewicht berechnet und pro Komponente aufsummiert. Aus den Kräften und Momenten werden Drehraten und Beschleunigungen um den Schwerpunkt des Flugzeugs gebildet. Diese bestimmen, wie sich das Flugzeug bewegt.

Über dieses Verfahren kann das physikalische Verhalten des Flugzeugs allein anhand seiner Geometrie und Masse bestimmt werden. Es hat in dieser Hinsicht Ähnlichkeiten mit einer *numerischen Strömungssimulation* (engl. *Computational Fluid Dynamics*, *CFD*). X-Plane eignet sich somit, das physikalische Verhalten eines Flugzeugs in gewissem Maße vorauszusagen. Aus diesem Grund kann X-Plane früh in der Entwicklungsphase eines Flugzeugs, beziehungsweise eines beliebigen Fluggeräts, eingesetzt werden. Beispiele erfolgreicher Einsätze sind unter anderem das *SpaceShipOne* von Scaled Dynamics[15], ein Autogyro von Carter Aviation Technologies[16], sowie [Adiprawita u. a. (2007)] und [Ribeiro und Oliveira (2010)]. Austin Meyers führt auf seiner Webseite auf, dass X-Plane von namhaften Luft- und Raumfahrtunternehmen

[14] vgl. X-Plane - What is X-Plane?, http://www.x-plane.com/desktop/meet_x-plane/ [Stand: 30.05.2014]

[15] X-Plane - Overview of X-Plane, http://wiki.x-plane.com/Chapter_1:_About_X-Plane [Stand: 30.05.2014]

[16] X-Plane - X-Plane as an Engineering Tool, http://www.jefflewis.net/x-plane_engineering.html [Stand: 30.05.2014]

wie der NASA, Boeing, Cessna oder Piper verwendet wird. Die US-amerikanische Luftfahrtaufsichtsbehörde (*Federal Aviation Administration*, *FAA*) hat X-Plane für die Verwendung in realen Flugsimulatoren zur Ausbildung von Piloten zugelassen.

Neben dem reinen Simulator werden weitere Werkzeuge mitgeliefert. So bietet der Editor *PlaneMaker* die Möglichkeit, ein komplettes Flugzeug von Grund auf zu entwerfen. 2012 hat Tim von Ahlen (Student der HAW Hamburg) ein unkonventionelles Box-Wing Flugzeug mithilfe von PlaneMaker entworfen und mit diesem Flugtests durchgeführt [von Ahlen (2012)]. Nach der Auswertung seiner Testergebnisse kam er zu einigen Verbesserungsvorschlägen für das Flugzeug.

Ein weiteres mitgeliefertes Werkzeug ist *WorldMaker*. Mit diesem können benutzerdefinierte Szenerien erstellt werden. Die simulierte Welt innerhalb des Simulators ist ohne Skalierung nachgebildet. Somit dauert ein Flug von Hamburg nach New York wie in der realen Welt etwa neun Stunden.

X-Plane bietet eine native Netzwerkschnittstelle zur Interprozesskommunikation an. Diese Methode wird von Paparazzi, ArduPilot, sowie [Ribeiro und Oliveira (2010)], [dos Santos u. a. (2011)], [Abdunabi (2006)] und [Adiprawita u. a. (2007)] zur X-in-the-Loop-Simulation von UAV Systemen verwendet. Bis zu 20 X-Plane Instanzen können an einer Netzwerkpartie teilnehmen.

Das *X-Plane Plugin SDK* ermöglicht das Erweitern von X-Plane durch Plugins. Über die native Netzwerkschnittstelle kann nur eine reduzierte Menge an internen Simulatorzuständen abgefragt und verändert werden. Das SDK bietet die Möglichkeit, auf etwa 4000 interne Zustände zuzugreifen. Weiterhin können durch Plugins grafische Bedienelemente wie Checkboxen, Listen, Texteingabefeldern oder Radiobuttons auf den Bildschirm gezeichnet werden. Über diese Bedienelemente kann sowohl das Plugin, als auch der Simulator gesteuert werden.

X-Plane unterstützt die Simulation von komplexem Wetterverhalten. Dieses führt von Sonnenschein bis Hagel, Gewitter und Sturm. Alle Effekte wirken sich durch die Blade element theory auch auf das Flugverhalten aus.

Eine geflogene Trajektorie kann durch eine dreidimensionale Wand, die von Flughöhe bis zum Boden reicht, visualisiert werden. Je nach Einstellung kann diese mehrere Kilometer weit reichen.

Der Entwickler hat einen nativen Support der *Virtual Reality* (*VR*) Brille Oculus Rift für die kommende Beta Version angekündigt. Derzeit werden von Sebastian Böhle einige Ansätze untersucht [Böhle (2014)], wie eine VR Brille im AES Projekt Anwendung finden könnte.

3.2.3. AeroSimRC

AeroSimRC ist ein etwa 70,- Euro teurer Modellflugsimulator des spanischen Softwareentwicklers Manuel Guillén. Er richtet sich hauptsächlich an Hobbyanwender und Modellflugpiloten. Seit der Version 3.3 werden allerdings auch Plugins unterstützt. Hierdurch erweitert sich die Zielgruppe um akademische und industrielle Anwender. Auf der Webseite des Herstellers wird dies explizit beworben: "*... for developers of autopilots, stabilizers, IMUs, etc.*"[17].

Eigene Flugzeuge lassen sich mit einem eingebauten Editor hinzufügen. Auch mitgelieferte Modelle können mit diesem editiert werden. Die Anpassung der Flugphysik und des Aktuatormodells ist nicht so umfangreich wie bei jsbsim. Nach Aussage von Andrey Shtrakbayn (BWB-Team) eignet sich der Editor lediglich zur rudimentären Beschreibung der Flugphysik[18].

Eine Wettersimulation ist zwar vorhanden, jedoch nicht so umfangreich wie die von FlightGear oder X-Plane. Es können lediglich Windstärke, Windrichtung, sowie ein prozentualer Anteil an Turbulenzen eingestellt werden.

3.2.4. AeroFly

AeroFly ist wie AeroSimRC ebenfalls ein Modellflugsimulator. Im Sommersemester 2013 hat Tim Backes (Student der HAW Hamburg) eine Studienarbeit angefertigt [Backes (2013)]. Seine Aufgabe war, das 3D Modell des AC2030 in einen beliebigen Flugsimulator zu integrieren. Hierfür hatte er Aerofly ausgewählt. Abbildung 3.6 zeigt einen Screenshot des digitalen AC2030 in Aerofly.

Warum Aerofly ausgewählt wurde, hat Backes nicht erläutert. Der Simulator besitzt keinerlei Möglichkeiten der Interprozesskommunikation. Auch das Erweitern durch Plugins wird nicht unterstützt. Daher scheidet Aerofly trotz des bereits integrierten Modells aus den Betrachtungen dieser Arbeit aus.

3.2.5. Prepar3D

Prepar3D ist ein Simulator des US-amerikanischen Rüstungs- und Technologiekonzerns Lockheed Martin. Er basiert auf Microsofts Simulationsframework *ESP* und steht explizit als

[17] AeroSimRC - New in 3.3 (March 2010), http://www.aerosimrc.com/en/download.htm [Stand: 30.05.2014]

[18] Persönliches Gespräch, Januar 2014

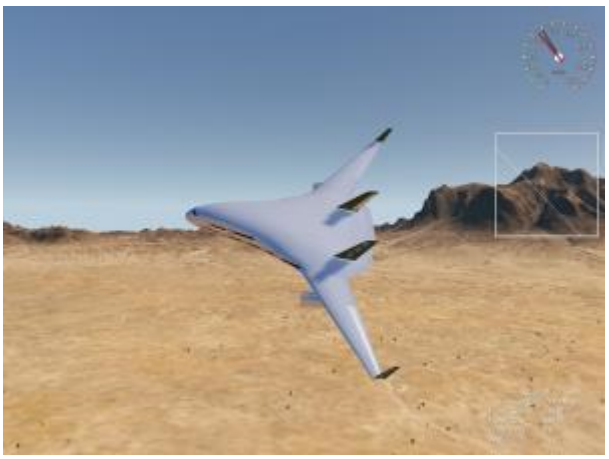

Abbildung 3.6.: Screenshot des AC2030 in AeroFly

Ausbildungs-, Trainings- und Lernsoftware und nicht als Unterhaltungssoftware zur Verfügung[19]. Eine Lizenz für den akademischen Gebrauch kostet 59.95 US-Dollar. Neben der Simulation von Flugzeugen und Hubschraubern können auch Wasser- und Landfahrzeugen simuliert werden. Diese sind häufig, jedoch nicht ausschließlich, militärischer Natur. Die im April dieses Jahres erschienene Version unterstützt nativ die Ausgabe der Bilddaten an eine Oculus Rift. Die Atmosphäre kann mit einem Wettersystem verändert werden.

Durch das ESP Framework ist implizit ein umfangreiches SDK vorhanden. Dieses kann zum Entwurf von in Flugphysik und Visualisierung detaillierten Flugmodellen verwendet werden. Eine native Interprozesskommunikation unterstützt Prepar3D nicht. Stattdessen können mithilfe des SDK Plugins entwickelt werden, wodurch sich Prepar3D als vollwertige Regelstrecke eignet.

Da Microsofts Flugsimulatoren ebenfalls auf ESP basieren, können viele für Microsofts Flugsimulator Reihe entwickelte Plugins auch in Prepar3D verwendet werden. Dadurch steht bereits eine große Anzahl an möglicherweise nützlichen Plugins zur Verfügung.

3.2.6. Matlab/Simulink

Die Matlab Erweiterung *Simulink* ist ein mächtiges Werkzeug zur Modellierung und Simulation von dynamischen Systemen aus Natur, Technik und Wirtschaft. Sofern das mathematische Modell eines noch so komplexen Systems bekannt ist, kann es in Simulink nachgebildet werden.

[19] Prepar3D - Prepar3D Product Overview, http://www.prepar3d.com/product-overview/ [Stand: 30.05.2014]

Daher eignet es sich auch zur mathematischen Nachbildung eines Flugzeugs. ROCS, IFSys, die NASA [Mueller (2007)], sowie [Chao u. a. (2013)] und [Lizarraga u. a. (2013)] verwenden Simulink zur Modellierung der Flugphysik ihrer Trägersysteme.

Bei verfügbaren Flugsimulatoren sind die mathematischen Formeln zur Berechnung des Flugverhaltens fest vorgegeben, lediglich die Parameter können variiert werden. Dies kann unter Umständen ein Problem sein oder eines werden, etwa wenn der Flugsimulator die benötigte Realität nur unzureichend nachbildet. Zum Beispiel ist keiner der oben genannten Simulatoren in der Lage, einen Flug durch einen Hurrikan oder einen Tornado nachzubilden. Auf der anderen Seite kann es vorkommen, dass ein Simulator zu viele Aspekte der Realität nachbildet und man ein simpleres Modell der Realität benötigt. Für beide Fälle stellt die Verwendung von Simulink eine geeignete Alternative dar.

Simulink selbst kann mit Hilfe so genannter *Blocksets* erweitert werden. Diese kommen teilweise vom Hersteller Mathworks selbst. Viele domänenspezifische Blocksets werden allerdings häufig von Dritten geliefert. Ein von Mathworks kostenpflichtig bereitgestelltes Blockset ist das *Aerospace Blockset*. Dieses umfasst eine Vielzahl vorgefertigter Blöcke zur Modellierung von (Teil-)Systemen aus dem Bereich der Luft- und Raumfahrt. Eine kostenlose, im Umfang ähnliche, Alternative ist mit dem nicht mehr weiter entwickelten *AeroSim Blockset* der mittlerweile aufgelösten Firma Unmanned Dynamics[20] verfügbar. Dem Mathworks eigenen Blockset liegen Beispielmodelle verschiedener Fluggeräte bei[21]. Unter Anderem sind der *DeHavilland Beaver* [Rauw (1993)] oder das *HL-20 Personnel Launch System* der NASA modelliert. Chao et. al. [Chao u. a. (2013)] und Nguewo [Nguewo (2014)] haben Simulink mithilfe des AeroSim Blocksets zur Berechnung der Flugphysik verwendet. Zur Modellierung einer Flugphysik sind anspruchsvolle Kenntnisse aus den Disziplinen Aerodynamik, Flugmechanik und Flugdynamik von Nöten. Die vorgestellten Blocksets können dabei eine große Hilfe darstellen.

Des Weiteren kann Simulink durch sogenannte *SFunctions* erweitert werden. Diese erlauben das Einbetten von eigenem Source Code in das Simulationsmodell. Unterstützt werden dabei C, C++ und Fortran als Programmiersprachen. Durch SFunctions lässt sich ein Simulink Modell um beliebige Schnittstellen zur Interprozesskommunikation erweitern. Weiterhin können mittels *Embedded Matlab* Blöcken Matlab Skripte eingebettet werden.

[20] Unmanned Dynamics, http://web.archive.org/web/20090512224330/http://www.u-dynamics.com/aerosim/default.htm [Stand: 30.05.2014]

[21] Aerospace Blockset Examples, http://www.mathworks.de/de/help/aeroblks/examples/index.html [Stand: 30.05.2014]

4. Anforderungen

Im Folgenden werden die selbst definierten Anforderungen genannt und beschrieben, welche die zu entwickelnde Infrastruktur erfüllen soll. Die Anforderungen wurden aufgestellt, nachdem die in 3 beschriebenen Konzepte und Ansätze evaluiert wurden. Sie unterteilen sich in Anforderungen an den Flugsimulator und an das Gesamtsystem. Jedoch wird zuerst beschrieben, welche Szenarien die Infrastruktur bieten soll.

4.1. Szenarien und Ziele

In der Motivation und der Aufgabenstellung wurde das angestrebte Ziel der Infrastruktur und die damit möglichen Szenarien nur abstrakt betrachtet. Nachdem die Grundlagen gegeben und eine Analyse durchgeführt wurde, sollen die Ziele und Szenarien hier konkret formuliert werden. Zunächst wird jedoch darauf eingegangen, wann ein hoher Realitätsgrad erforderlich ist und wann dieser vernachlässigt werden kann.

Notwendiger Realitätsgrad

Für eine realitätsnahe Simulation ist es unabdingbar, dass sich das simulierte Flugzeug nicht wie ein beliebiges, sondern wie der AC2030 verhält. Jedoch ist eine möglichst realitätsnahe Simulation nicht immer zwingend erforderlich. Im Folgenden wird erläutert, wann der Realitätsgrad der Simulation eine wichtige Rolle spielt und wann er vernachlässigt werden kann.

Wie in 2.3 dargestellt, werden Flugregler typischerweise als kaskadierte Regelkreise aufgebaut. Die inneren Schleifen (Control) regeln die Fluglage und dämpfen die Bewegungen. Die äußeren Schleifen (Guidance und Navigation) führen das Flugzeug entlang einer Trajektorie von der aktuellen Position zu einem gewünschten Wegpunkt.

In den folgenden zwei Fällen ist eine hohe Anforderung an den Realitätsgrad der Simulation gegeben:

1. Verwendet man die Simulation nun zur Identifikation der Reglerparameter im gesamten Regelkreis, muss das physikalische Modell aller Elemente möglichst realitätsnah sein. Nur so ist gewährleistet, dass die ermittelten Parameter der GNC Regler auch im realen Flug ein wünschenswertes Ergebnis liefern. Sollte der Realitätsgrad der Simulation weit von der Realität abweichen, sind die Simulation, bzw. die identifizierten Parameter und damit das Ergebnis der Simulation wertlos.

2. Soll die Simulation als vollständiger Systemtest zur Validierung der FCU eingesetzt werden, ist ebenfalls ein hoher Realitätsgrad erforderlich. Ziel dieses Tests ist es, die Funktion der kompletten Firmware der FCU zu prüfen. Nach solch einem Test soll die Firmware ohne Neukompilation in realen Flügen eingesetzt werden können. Dies erfordert es, dass das simulierte Flugzeug das Flugverhalten des AC2030 aufweist.

In weiteren zwei Fällen spielt der Realitätsgrad des simulierten Modells eine untergeordnete Rolle:

1. Sollte die Simulation nicht der Identifikation der Reglerparameter dienen, kann der Realitätsgrad vernachlässigt werden. Dies ist der Fall, wenn Firmwarekomponenten der FCU entwickelt und getestet werden sollen. Zum Beispiel ist es beim Aufzeichnen von Daten auf der Speicherkarte oder dem Parsen von Daten des GPS Sensors unerheblich, ob es sich beim simulierten Flugzeug um den AC2030, eine Cessna 172P oder einen Airbus A380 handelt.

2. Falls die Simulation zur Verifikation der GNC Reglerstrukturen eingesetzt wird, kann der Realitätsgrad ebenfalls vernachlässigt werden. Dieser Schritt erfolgt vor der Parameteridentifikation. Mit diesem kann geprüft werden, ob die Regler in ihrer Struktur korrekt entworfen und umgesetzt wurden. Ein Vorzeichenfehler oder eine falsche Rückführung innerhalb des Regelkreises lässt für gewöhnlich jedes Flugzeug von Modell bis Verkehrsflugzeug abstürzen. Eine korrekte Umsetzung kann hingegen jedes Flugzeug in dieser Bandbreite führen. Zwar kann die Bewegung schwingen, schlecht gedämpft oder langsam sein, jedoch ist sie in groben Zügen korrekt.

Szenarien

Mit der Infrastruktur sollen Model-In-the-Loop- und Hardware-In-the-Loop-Tests, bzw. Simulationen durchgeführt werden können. In beiden Fällen sollen alle drei Modes der FCU (manuell, stabilisiert, autonom) abgedeckt werden.

MIL-Szenario
Das MIL-Szenario soll es einem Anwender ermöglichen, einen Flugregler durch experimentelle

Ermittlung auszulegen. Hierfür müssen die Flugphysik, das Aktuatormodell und das Sensormodell des AC2030 realitätsnah nachgebildet werden. Es ist dabei nicht das Ziel der vorliegenden Arbeit, die drei Modelle zu entwerfen. Stattdessen sollen diese als Platzhalter vorliegen.

Eine Anforderung an die Infrastruktur ist, dass ein Anwender keine Programmierkenntnisse benötigen soll, um die Platzhalter mit adäquaten Modellen zu füllen.

Die Infrastruktur soll dahingehend erweiterbar sein, dass die Komponente zur Sensordatenaufbereitung in die Schleife integriert werden kann.

Die Zeitverhalten der FCU kann vernachlässigt werden, da dieses erst bei der konkreten Umsetzung des Flugreglers auf der verwendeten eingebetteten Hardware zum Tragen kommen.

HIL-Szenario
Die HIL-Simulation ist in zwei Szenarien unterteilt:

- **Entwicklung der FCU**: Die HIL-Simulation soll die Entwicklung der FCU Firmware unterstützen, in dem eine simulierte Umgebung geschaffen wird. Der Realitätsgrad kann bis zu einem gewissen Maße vernachlässigt werden. Dieses Szenario wird fortan als *HIL_Dev* bezeichnet.
- **Systemtest der FCU**: Die HIL-Simulation dient dem Systemtest der FCU. Das Szenario soll den Test der gesamten Firmware der FCU ermöglichen. Dabei soll der Realitätsgrad von Flugphysik, Aktuatormodell und Sensormodell nicht unter dem des MIL-Szenarios liegen. Die FCU soll nicht zwischen Realität und Simulation unterscheiden können. Hierfür ist es erforderlich, dass die exakt gleiche Firmware für Simulationen und später reale Flüge ohne Neukompilation genutzt werden kann. Dieses Szenario wird fortan als *HIL_Test* bezeichnet.

4.2. Flugsimulator

Die Anforderungen an den Flugsimulator unterteilen sich in primäre, sekundäre und optionale Anforderungen.

Anmerkung zur Berechnung der Flugphysik

Für eine realitätsnahe Simulation des Flugverhaltens reicht es nicht aus, lediglich das simulierte Modell an die Parameter des Systems anzupassen. Die im Simulator-Backend verwendeten mathematischen Formeln zur Berechnung von Bewegung und Verhalten des Flugzeugs müssen die Realität entsprechend widerspiegeln. Bei großen Flugzeugen müssen Eigenschaften wie etwa strukturelle Flexibilität und Aeroelastizität beachtet werden. Bei Modellflugzeugen mit

wenigen Metern Spannweite können diese Eigenschaften in der Simulation jedoch vernachlässigt werden, da sie bei Modellflugzeugen nicht oder nur vernachlässigbar gering auftreten[1].

Zur Beurteilung der Güte sind neben dem Zugriff auf die verwendeten Formeln auch jahrelange Erfahrungen in der mathematischen Beschreibung von Flugzeugen nötig. Lediglich bei FlightGear können die verwendeten Formeln im Source Code eingesehen werden, da die restlichen Flugsimulatoren nicht quelloffen sind. Jedoch erfüllt der Autor dieser Arbeit die zweite Anforderung nicht, da er keine Erfahrung in der mathematischen Beschreibung von Flugzeugen vorweisen kann. Aus diesem Grund wird die Güte der Simulator-Backends der beschriebenen Flugsimulatoren als gut, bzw. für die gegebene Aufgabenstellung geeignet, angenommen.

Primäre Anforderungen

Die zwei Mindestanforderungen an den Flugsimulator wurden in 3.2 bereits erwähnt. Diese werden nachfolgend definiert.

Interprozesskommunikation
Der Flugsimulator muss Möglichkeiten der Interprozesskommunikation bieten, so dass er als Regelstrecke verwendet werden kann.

Folgende Werte müssen mindestens als Flugzustand ausgelesen werden können:

- **Position**
- **Fluglage**
- **Drehraten**
- **Luftdaten** (Anstellwinkel, Schiebewinkel, Airspeed)

Diese Werte stellen in der Zustandsraumdarstellung den Zustandsvektor **x(t)**, bzw. den Ausgangsvektor **y(t)** dar.

Die Eingaben umfassen die virtuellen Steuersignale (Abschnitt 2.1) des Flugreglers. In der Zustandsraumdarstellung stellen die Eingaben den Steuervektor **u(t)** dar. Die Eingabekanäle für Rettungssystem und Aux können beliebig umgesetzt werden (beispielsweise Schubumkehr, *Speedbrakes* oder *Slats*). Bei Rückführung der Eingaben ist es wichtig, dass diese nicht mit den Benutzereingaben am Simulator selbst kollidieren. Die Benutzereingaben müssen entweder ignoriert werden oder sich ergänzen. In zweiten Fall ergibt die Addition beider Datensätze das Gesamtergebnis der Eingaben am Simulator.

[1] vgl. Prof. Dr. Detlev Schulze, Perönliches Gepräch, Dezember 2013

Parametrierung
Der Simulator muss die Möglichkeit bieten, das simulierte Modell in seinem Verhalten anzupassen. Die folgenden Konfigurationsmöglichkeiten müssen mindestens geboten werden:

- **Derivativa**: Das physikalische Verhalten des simulierten Flugzeugs muss anpassbar sein.
- **Steuerflächenkonfiguration**: Anzahl, Position, Größe, etc. der Steuerflächen müssen konfigurierbar sein.
- **Steuerflächenzuordnung**: Die virtuellen Steuersignale müssen auf die mittels der Steuerflächenkonfiguration definierten Steuerflächen umgesetzt werden.
- **Aktuatormodell**: Die dynamischen Eigenschaften der Aktoren müssen konfigurierbar sein. Mindestens das Vorgeben der Stellgeschwindigkeiten, sowie die erzeugten Drehmomente muss unterstützt werden.

Sekundäre Anforderungen

Neben den beiden primären Anforderungen wurden weitere an den Flugsimulator gestellt. Diese werden im Folgenden erläutert.

Reine Datensenke zur Visualisierung
Neben der Verwendung des Flugsimulators als Regelstrecke soll er auch als Komponente zur rein visuellen Darstellung dienen können. Dies ermöglicht es, einen aufgezeichneten Flug live zu visualisieren oder das Flugzeug bei einem echten Flug live in 3D darzustellen. Für beide Fälle ist es nötig, dass mindestens die Position und die Fluglage vorgegeben werden können. Diese beschreiben das Flugzeug in seinen sechs Freiheitsgraden. Das Simulator-Backend muss entweder deaktiviert sein oder es beeinflusst die Darstellung nicht durch eigene berechnete Bewegungen. Durch letzteres könnte sich eine nicht zufriedenstellende Visualisierung, zum Beispiel durch Springen oder Teleportation des Flugzeugs innerhalb der simulierten Welt, ergeben.

Bedienung
Die Bedienung des Flugsimulators sollte einfach und intuitiv sein, damit ein angenehmes Arbeiten mit diesem möglich ist. Als Eingabegeräte sollen RC-Fernbedienungen und Tastaturen unterstützt werden.

Wettermodell
Es soll ein minimales Wettermodell unterstützt werden. Es genügt, wenn Windgeschwindigkeiten, Windrichtung und Turbulenzen eingestellt werden können. Hierdurch kann getestet werden, wie gut die Regler auf Störungen reagieren. Das Wettermodell stellt in der Zustandsraumdarstellung den Störgrößenvektor **z(t)** dar.

Optionale Anforderungen

Einige der aufgestellten Anforderungen sind optional, da die dadurch vergrößerten Möglichkeiten in der vorliegenden Arbeit nur einen geringen Mehrwert bieten. Dennoch sollte bei der Auswahl des Flugsimulators wenn möglich auf diese geachtet werden, damit später nicht auf einen anderen ausgewichen werden muss oder gar zwei verschiedene Flugsimulatoren verwendet werden müssen.

3D Modell
Das gerenderte 3D Modell muss nicht dem AC2030 entsprechen. Dies ist wünschenswert, jedoch nicht erforderlich und daher eine optionale Anforderung an den Flugsimulator.

Simulationsgeschwindigkeit / Bildwiederholfrequenz
Es ist vorgesehen, die diskreten Regelalgorithmen mit einem Takt von 50Hz zu betreiben. Für einen realitätsnahen Test muss die Regelstrecke ihre Daten auch mit dieser Frequenz bereitstellen. Um dies zu gewährleisten, muss der Flugsimulator die interne Komponente zur Berechnung des physikalischen Zustands (*Physikengine*) mindestens 50 Mal pro Sekunde ausführen.

Die Physikengine ist bei Videospielen oder ähnlicher 3D Simulationssoftware typischerweise mit der Renderkomponente verzahnt [Gregory (2009)]. So dass die Bildwiederholfrequenz (engl. *Framerate*, frames-per-second, *FPS*) und die Updatefrequenz der simulierten Physik unveränderlich aneinander gekoppelt sind. Selbiges gilt auch für die Komponente der 3D Engine, welche für die Interprozesskommunikation zuständig ist. Dabei ist es irrelevant, ob diese über eine native Netzwerkunterstützung oder ein Plugin-Mechanismus abgehandelt wird.

Damit der Simulator seine Daten mit 50Hz bereitstellt, muss er mindestens mit dieser Bildwiederholfrequenz ausgeführt werden. Dies erfordert je nach Flugsimulator mehr oder weniger potente Hardware. Er sollte somit nicht zu hohe Anforderungen an die Hardware stellen. Die Anforderung ist allerdings optional, da das Problem immer durch den Einsatz potenterer Hardware behoben werden kann.

Die Grafikqualität der gerenderten Bilder muss keiner Anforderung entsprechen, da diese beim Entwurf von Flugreglern vernachlässigt werden kann.

Bereitstellung des gerenderten Bildes
Ein langfristiges Ziel im AES Projekt ist, eine automatische Kollisionsvermeidung zu entwickeln. Nach aktuellem Stand der Technik könnte hierfür eine Echtzeit-Videokamera als Basis dienen. Ein weiteres langfristiges Ziel ist, eine Bilddaten-gestützte Navigation im 3D Raum zu entwickeln [Weiss (2012)]. In beiden Fällen könnte der Flugsimulator zur Testbildgenerierung verwendet werden, da dieser auch Wettereffekte wie zum Beispiel Regen, welche sich negativ auf die Bilderkennungsalgorithmen auswirken, simulieren kann. Auch für Böhles Experimente mit der Oculus Rift könnten Testbilder aus dem Flugsimulator verwenden werden.

Die Bilder müssen in den drei Fällen einer externen Anwendung zur Verfügung gestellt werden. Hierfür eignen sich drei verschiedene Ansätze. Zum einen kann der Grafikspeicher der Grafikkarte ausgelesen werden [Díaz-Tula u. a. (2010)]. Als Alternative kann eine Screen-Capture Software wie das kostenlose CamStudio[2] verwendet werden. Die dritte Möglichkeit besteht darin, das gerenderte Bild mit einer vom Flugsimulator bereitgestellten Applikationsschnittstelle auszulesen. Dies wäre die programmiertechnisch einfachste Alternative. FlightGear unterstützt dies zum Beispiel über dessen Screen Relay Funktion.

Da die Arbeiten an den beiden ersten Entwicklungen voraussichtlich erst in einigen Semestern beginnen werden und die Oculus Rift nicht zwingend live Bilder benötigt, wäre solch eine Schnittstelle zum Abgreifen des gerenderten Bildes zwar wünschenswert, jedoch nicht zwingend erforderlich. Aus diesem Grund ist diese Anforderung an den Flugsimulator optional.

Verbundflug in Netzwerkpartien
Verschiedene Anwendungsszenarien eines autonom fliegenden AC2030, wie zum Beispiel Katastrophenschutz, Rettungs- oder Versorgungsmissionen, sehen den Einsatz von mehreren, kooperierenden UAVs vor. Hierfür eignet es sich mehrere Simulatorinstanzen zu einer Netzwerkpartie, ähnlich wie in Mehrspieler-fähigen Videospielen, zusammenzuführen. Durch diese Möglichkeit kann eine visuelle Repräsentation des Gesamtbildes einer Mission erfolgen. Die Mehrspielerfähigkeit des Flugsimulators ist allerdings optional, da der kooperierende Einsatz von mehreren UAVs im AES Projekt erst in einigen Jahren an Bedeutung gewinnt.

4.3. Gesamtsystem

Im Folgenden werden die an die gesamte Testinfrastruktur gestellten Anforderungen beschrieben. Falls nicht explizit angegeben, gelten sie für die MIL- und die HIL-Szenarien gleichermaßen.

Laufzeitverhalten
Die Laufzeit der Simulationsdaten zwischen Regler und Regelstrecke soll nach Möglichkeit gering sein. Sollte dies nicht der Fall sein, müssen die Laufzeiten im Regler als Totzeiten berücksichtigt werden. Eine Simulation (MIL- und HIL-Szenario) soll ein realitätsnahes Laufzeitverhalten aufweisen.

Aufzeichnung
Die Simulationsdaten sollen aufgezeichnet werden, so dass Eingabe und Ergebnis eines Testfalls im Nachhinein analysiert werden können.

Visualisierung
Durch die Nutzung eines Flugsimulators als Regelstrecke steht implizit eine Visualisierung des

[2] CamStudio - Free Streaming Video Software, www.camstudio.org [Stand: 30.05.2014]

simulierten Flugzeugs zur Verfügung. Daneben sollen die Simulationsdaten in Diagrammen geplottet werden. Durch diese Möglichkeit kann die Güte des Reglers detaillierter bestimmt werden. Diagramme sind hierfür geeigneter als ein Flugsimulator, da unerwünschte Eigenschaften wie etwa konstante Abweichungen von Sollwerten, schwache Dämpfungen oder Aufschwingen in Zahlenwerten abgelesen und ausgewertet werden können. Die Visualisierung soll folgende Datenquellen unterstützen:

- live Daten einer zur Laufzeit durchgeführten Simulation
- live Daten einer Aufzeichnung
- statische Daten einer Aufzeichnung

Bei der live Anzeige sollen die Diagramme laufend aktualisiert werden. Bei der statischen soll ein kompletter Datensatz einer Aufzeichnung in den Diagrammen dargestellt werden.

Bedienung
Einem Anwender sollen Details der Umsetzung verborgen bleiben. Besonders der Entwurf der Regler in Simulink soll einfach und ohne Programmierkenntnisse möglich sein, da die ausführende Person nicht notwendigerweise über einen Informatikhintergrund verfügt. Das Einrichten und Durchführen einer Simulation, die Verwendung der Funktionen zur Aufzeichnung und Visualisierung der Daten, sowie der Entwurf der Regler soll intuitiv möglich sein. Das zu Rate ziehen einer Benutzerdokumentation soll nur in Spezialfällen nötig sein. Als Gegenbeispiel sei ArduPilot genannt. Bei diesem erfordert das Einrichten einer Simulation Netzwerk- und Programmierkenntnisse.

Die Funktionen zur Aufzeichnung und Visualisierung sollen per Plug-and-Play Prinzip verwendet werden können. Eine Simulation soll möglich sein, ohne dass Aufzeichnung und Visualisierung ausgeführt werden. Die Visualisierung der aufgezeichneten Daten soll durchgeführt werden können, ohne dass andere Funktionen der Infrastruktur benötigt werden.

Erweiterbarkeit
Die Infrastruktur soll um weitere Funktionen erweitert werden können. So soll etwa für die Integration eines weiteren Flugsimulators oder das Hinzufügen von Möglichkeiten zur detaillierteren Visualisierung der Simulationsdaten keine Änderung an bereits existierenden Komponenten nötig sein.

5. Konzept

Auf Basis der Grundlagen und der Analyse wurde ein Konzept erstellt, welches den im letzten Kapitel aufgestellten Anforderungen genügen soll. Dieses Kapitel erläutert das Konzept der Infrastruktur und dessen Teilkomponenten.

5.1. Auswahl der Flugsimulatoren

Als Flugsimulatoren wurden X-Plane, FlightGear, AeroSimRC, sowie Simulink ausgewählt. Im Folgenden wird diese Entscheidung begründet.

X-Plane
Aufgrund der Möglichkeiten, die X-Plane bietet, scheint es für die gegebene Problemstellung die beste Lösung darzustellen. Besonders die Fähigkeit, dass die Flugphysik ohne Kenntnisse über die Derivativa des AC2030 akkurat nachgebildet wird, macht es zum primären Werkzeug der Wahl. Das Flugzeug wird dabei in PlaneMaker entworfen. Es werden lediglich die Geometrien und Gewichte der einzelnen Komponenten, sowie Kenntnisse über Triebwerksdaten benötigt.

Sofern die von X-Plane verwendete Blade element theory zur Berechnung der Flugphysik sich als so akkurat wie beworben herausstellt, können die Derivativa des AC2030 mit X-Plane ermittelt werden. Allerdings kann der Autor dies nicht beurteilen und stützt sich daher auf die Werbeversprechen des Herstellers, sowie auf die positiven Erfahrungen von [dos Santos u. a. (2011)], [Adiprawita u. a. (2007)] und [Ribeiro und Oliveira (2010)]. Davon ausgehend, dass die Aussage korrekt ist, eignet sich X-Plane am besten für den Entwurf eines qualitativ hochwertigen Flugreglers.

FlightGear
Neben X-Plane wurde FlightGear als Alternative ausgewählt. Die Entscheidung wurde aufgrund der Erweiterbarkeit von FlightGear getroffen. Sollten sich in Zukunft neue Anforderungen an die Infrastruktur und an den Flugsimulator ergeben, können diese durch die Quelloffenheit gegebenenfalls selbst umgesetzt werden, ohne dabei auf den Hersteller von X-Plane angewiesen zu sein. Zu nennen sei hier die Screen Relay Funktion von FlightGear. Diese ermöglicht das Abgreifen des gerenderten Bildes aus einem externen Programm heraus. X-Plane selbst bietet keine äquivalente Funktion an.

FlightGear ermöglicht es Szenarien, wie etwa das Simulieren von Tornados oder Waldbränden, nachzubilden. Diese Szenarien können für das AES Projekt in Zukunft von Interesse sein.

AeroSimRC
Während der Evaluierung der verschiedenen Flugsimulatoren wurden Proof-of-Concept Versuche durchgeführt. Bei diesen hat sich gezeigt, dass sich AeroSimRC am besten für die Umsetzung der von den Flugsimulatoren unabhängigen Infrastrukturkomponenten eignet. So ist zum Beispiel die allgemeine Geschwindigkeit von AeroSimRC besser als die der Konkurrenz. Gegenüber X-Plane vergeht vom Starten der Anwendung bis zum Beginn der Rollphase im Durchschnitt 95% weniger Zeit. Die Bildwiederholfrequenz ist bei niedrigen Grafikeinstellungen etwa vier bis fünf Mal höher als bei X-Plane. AeroSimRC kann somit auch auf älterer und leistungsschwacher Rechnerhardware als Regelstrecke verwendet werden. Die Bedienung ist für einen Laien subjektiv einfacher als bei X-Plane und FlightGear, da AeroSimRC einen deutlich geringeren Funktionssatz bietet.

Simulink
Langfristig ist geplant, die Flugphysik des AC2030 in Simulink nachzubilden. Daher soll neben den Flugsimulatoren auch Simulink als Regelstrecke verwendet werden können. Eine selbst entworfene ausführbare Flugphysik bietet eine größere Flexibilität als verfügbare Flugsimulatoren. Zwar bietet FlightGear die Möglichkeit, ein eigenes Simulator-Backend zu verwenden, jedoch ist hierfür eine hohe Einarbeitungsphase in den Source Code von FlightGear nötig. Besonders kritisch ist dies aus dem Aspekt, da die Entwickler des Simulink Modells höchstwahrscheinlich nicht über einen Informatikhintergrund verfügen werden. Kurzfristig eignet sich Simulink zum Erzeugen von Fehlersituationen. Zum Beispiel kann der Ausfall eines bestimmten Sensors durch das Ausgeben von Zufallswerten oder konstanten Werten simuliert werden. Simulink eignet sich dadurch zum Testen der Firmwarekomponenten zur Fehlererkennung innerhalb der FCU.

5.1.1. Geeignete Verwendungszwecke / Leitfaden

Die ausgewählten Flugsimulatoren erfüllen allesamt die geforderten primären und sekundären Anforderungen. Jedoch eignet sich nicht jeder Flugsimulator für jede Problemstellung oder Aufgabe gleichermaßen. Die nachfolgende Aufzählung soll einen Leitfaden für die Verwendung der einzelnen Flugsimulatoren geben:

- **X-Plane**: Eignet sich für Szenarien, in denen ein hoher Realitätsgrad der berechneten Flugphysik erforderlich ist. Somit stellt er die primäre Wahl sowohl zur Reglerauslegung (MIL-Szenario), als auch zum Testen (HIL_Test-Szenario) der FCU dar.
- **FlightGear**: Eignet sich langfristig für die Entwicklung exotischer Szenarien, für die X-Plane nicht die erforderliche Erweiterbarkeit bietet.

- **AeroSimRC**: Eignet sich für Szenarien, in denen der Realitätsgrad der berechneten Flugphysik vernachlässigt werden kann (HIL_Dev-Szenario). Durch die einfache Bedienung und die hohe Gesamtgeschwindigkeit stellt es die primäre Lösung zur Verbesserung und Anpassung der Infrastruktur dar und ist somit auch dem Selbstzweck der Infrastruktur geschuldet.
- **Simulink**: Eignet sich für die einfache Simulation von Fehlersituationen, sowie langfristig für die Modellierung einer eigenen Flugphysik.

Der Leitfaden gibt die Empfehlung des Autors wieder und ist keinesfalls verpflichtend.

5.2. Architektur der Infrastruktur

In diesem Abschnitt wird die Architektur und die Kommunikationsstruktur der Infrastruktur vorgestellt.

5.2.1. Übersicht

Die von der Infrastruktur geforderten Funktionen werden durch einzelne Komponenten realisiert. Diese sollen nach Möglichkeit jeweils nur eine Zuständigkeit haben, wodurch das Prinzip von *Separation of Concerns* gestärkt werden soll. Das zentrale Element der Infrastruktur bildet ein *Software-Bus*. Dieser Bus entkoppelt die einzelnen Komponenten voneinander. Abbildung 5.1 zeigt eine Übersicht der Infrastruktur inklusive der einzelnen Komponenten.

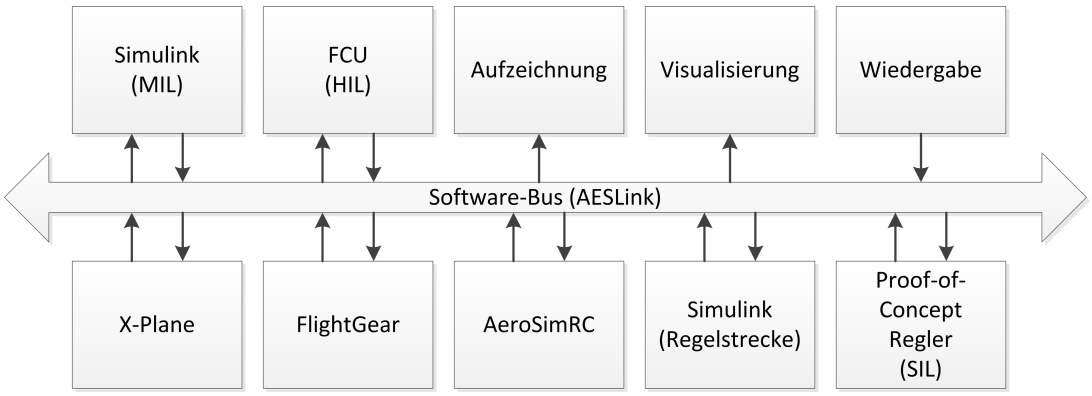

Abbildung 5.1.: Architektur der Infrastruktur, bestehend aus einem Software-Bus (AESLink) und den einzelnen Komponenten

Die Verwendung eines Software-Bus bringt folgende Vorteile gegenüber einzelnen Punkt-zu-Punkt Verbindungen:

- Die Komponenten haben keine Kenntnis über andere Busteilnehmer.
- Komponenten können sich nach dem Plug-and-Play Prinzip zur Laufzeit einhängen. Die Einrichtung durch den Benutzer entfällt, da der Bus immer die zentrale Anlaufstelle darstellt.
- Die Verteilung der Nachrichten wird vom Bus übernommen.
- Die Komponenten registrieren sich für den Empfang von bestimmten Nachrichten. Dabei ist es unabhängig, welche Komponente die Nachrichten versendet.
- Die Infrastruktur kann ohne Anpassung um weitere Komponenten erweitert werden.
- Die Kommunikation zwischen den Komponenten ist nachrichtenbasiert. Somit können die einzelnen Komponenten auf verschiedene Computer verteilt werden.
- Softwarebibliotheken, die den Zugriff auf den Bus abstrahieren, müssen pro Programmiersprache nur einmal implementiert werden und können von mehreren Komponenten verwendet werden.

5.2.2. Komponenten

Im Folgenden soll kurz auf die Funktionen der einzelnen Komponenten eingegangen werden. Auf die Realisierung dieser wird in 6 eingegangen.

- **X-Plane, FlightGear, AeroSimRC**: Die drei Flugsimulatoren berechnen die ausführbare Flugphysik und können zur rein visuellen Darstellung verwendet werden.
- **Simulink (Regelstrecke)**: Stellt eine ausführbare Flugphysik bereit. Eine visuelle Anzeige wie bei den Flugsimulatoren ist nicht erforderlich.
- **Aufzeichnung**: Soll alle über den Bus versendeten Nachrichten ohne Filterung speichern.
- **Visualisierung**: Soll eine im Gegensatz zu den Flugsimulatoren detailliertere Visualisierung des Flugzustands bieten.
- **Wiedergabe**: Soll die aufgezeichneten Daten live über den Bus versenden. Es soll eine Filterung der Nachrichten vorgenommen werden können.
- **Simulink (MIL)**: Der in Simulink zu entwerfende Regler.

- **FCU (HIL)**: Die reale FCU, da diese letztendlich das im AES Projekt zu entwickelnde System darstellt.
- **Proof-of-Concept Regler (SIL)**: Ein zu Test-, Debug- und Demonstrationszwecken implementierter Flugregler.

5.2.3. Einbindung des Piloten

Zur Einbindung der Eingabedaten des Piloten standen zwei verschiedene Ansätze zur Verfügung. Zum einen können die Benutzereingaben des Flugsimulators verwendet werden. Als Alternative dazu können die Eingaben durch eine weitere Komponente eingelesen und über den Bus verteilt werden. Die zweite Herangehensweise hat den Vorteil, dass eine beliebige Quelle für Eingaben verwendet werden kann. Beispiele sind Tastatur, (emulierte) RC-Fernbedienungen, (emulierte) Joysticks, Aufzeichnungen, statische Testmuster, etc. Für die erste Variante spricht, dass die geforderten Eingabequellen (Tastatur und RC-Fernbedienung) von den Flugsimulatoren bereits unterstützt werden. Daher wurde der erste Ansatz gewählt.

Ob die Daten in der FCU verwendet werden oder nicht ist vom anliegenden FCU Mode abhängig. Dies entspricht der Situation in realen Flügen, da der Pilot unabhängig vom anliegenden FCU-Mode Eingaben tätigen kann. Die Infrastruktur soll keine Kenntnisse über den anliegenden Mode haben und sendet die Piloteneingaben daher in jedem Fall an den Flugregler, bzw. an die FCU.

5.3. Software-Bus (AESLink)

AESLink ist ein Kommunikationsprotokoll, das zur Kommunikation der Komponenten innerhalb der Infrastruktur spezifiziert wurde. Es behandelt die logische und teilweise auch die physikalische Kommunikation der Komponenten untereinander. Die Verwendung vorhandener Protokolle wie MAVLink oder UAVTalk wurde erwogen. Da diese jedoch ein wesentlich größeres Aufgabengebiet abdecken als die Infrastruktur erfordert, wurde gegen die Verwendung entschieden.

5.3.1. Logische Kommunikation

Zur Kommunikation der Komponenten untereinander wurden zwei Nachrichtentypen definiert. Diese spiegeln den Flugzustand (*DataFromAircraft*), und die Steuereingaben des Flugreglers (*DataFromAutopilot*) wider. Beim Versenden wird jeder Nachricht ein Nachrichtenkopf (*MessageHeader*) vorangestellt.

Die nachfolgende Darstellung bezieht sich auf die zum Zeitpunkt der Fertigstellung dieser Arbeit (Mai 2014) angelangte Protokollversion 4.

MessageHeader

Besonders bei der Kommunikation über Stream-basierte Kanäle ergeben sich durch die Verwendung eines Nachrichtenkopfes mehrere Vorteile. Unter anderem bietet ein Nachrichtenkopf einen Synchronisationspunkt an. Bei Datenpaketen unterschiedlicher Länge kann sich die Empfangslogik auf die noch zu erwartenden Daten einstellen. Sowohl bei Stream- als auch bei Block-orientierten Kanälen kann die nachfolgende Nachricht anhand eines Typs identifiziert werden. Weiterhin können die verwendete Protokollversionen eingebettet werden. Der bei AESLink verwendete MessageHeader hat eine Länge von 16 Byte. Der Aufbau ist in Abbildung 5.2 dargestellt. Die Zahlen repräsentieren die relative Speicheradresse der einzelnen Felder.

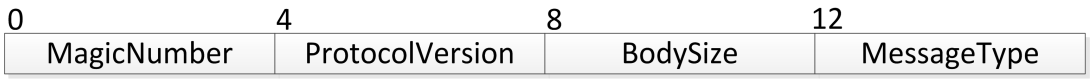

Abbildung 5.2.: Aufbau des Nachrichtenkopfes einer AESLink Nachricht

Die einzelnen Felder haben dabei die folgenden Inhalte und Aufgaben:

- **MagicNumber**: Identifiziert die Nachricht als seine AESLink Nachricht.
- **ProtocolVersion**: Enthält die verwendete Protokollversion des Senders.
- **BodySize**: Gibt die Länge der enthaltenen Nachricht in Bytes an.
- **MessageType**: Identifiziert den Typ der enthaltenen Nachricht.

Der als MagicNumber verwendete Wert ist die ASCII Repräsentation der Zeichenkette „AESL“. Dieser Wert sollte in kommenden Versionen von AESLink nicht verändert werden.

DataFromAircraft

Nachrichten dieses Typs werden von Flugsimulator an Flugregler gesendet und beschreiben den Flugzustand. Die einzelnen Felder und Werte wurden vom Autor nach bestem Wissen und Gewissen ausgewählt. Der übertragene Flugzustand erhebt dadurch keinesfalls den Anspruch auf Vollständigkeit. Eine Nachricht hat eine Länge von 112 Byte. Der Aufbau ist in Abbildung 5.3 dargestellt. Die Flugsimulatoren verwenden für die einzelnen Werte unterschiedliche Einheiten

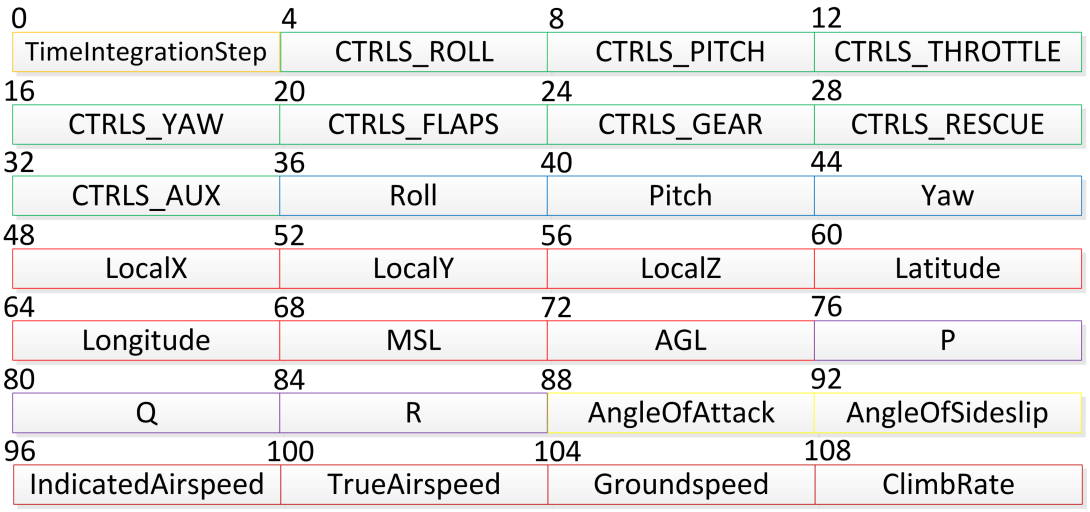

Abbildung 5.3.: Aufbau einer *DataFromAircraft* Nachricht

oder Wertebereiche. Daher ist vor dem Einspeisen der Daten eine Normierung der Werte nötig. Die Wertebereiche werden in Anhang A dargestellt.

Die einzelnen Datenfelder sind in Gruppen eingeteilt. Nachfolgend wird der Inhalt der Felder erläutert.

Simulationszeit (orange)
Das Feld *IntegrationTimeStep* gibt die Zeit zwischen dem aktuellen und dem letzten Simulationsschritt innerhalb des Flugsimulators an. Der Inhalt der Nachricht stellt den Flugzustand des aktuellen Simulationsschrittes dar.

Piloteneingaben (grün)
Die Felder *CTRLS_X* sind ein Array variable Länge. Wobei *variabel* lediglich innerhalb des Protokolls zu verstehen ist. Die Länge muss zum Zeitpunkt der Kompilation bestimmt werden. Jedes Feld des Arrays enthält jeweils ein virtuelles Steuersignal.

Fluglage (blau)
Die Fluglage wird durch Eulerwinkel (*Roll*, *Pitch*, *Yaw*) im körperfesten Koordinatensystem beschrieben. Wobei der Gierwinkel bei 0 Grad Richtung Nordpol zeigt.

Position (rot)
Die Position wird durch sieben verschiedene Felder beschrieben. Drei der Felder (*LocalX*, *LocalY*, *LocalZ*) geben die Position des Flugzeugs im lokalen Koordinatensystem an. Weitere drei Felder (*Latitude*, *Longitude*, *MSLIZ*) beschreiben die Position im globalen Koordinatensystem. Das Feld *AGL* entspricht der Höhe über Grund.

Drehraten (lila)
Diese Felder enthalten die anliegenden Drehraten in den drei körperfesten Achsen. Wobei *P* um die Längsachse (Rollen), *Q* um die Querachse (Nicken) und *R* um die Hochachse (Gieren) dreht.

Luftdaten (gelb)
Das Feld *AngleOfAttack* beschreibt den Anstellwinkel, in welcher die Strömung auf die Tragflächen trifft. Das Feld *AngleOfSideslip* beschreibt den Schiebewinkel, dem das Flugzeug ausgesetzt ist.

Geschwindigkeiten (dunkelrot)
Die Felder *IndicatedAirspeed* und *TrueAirspeed* beschreiben die Fluggeschwindigkeit gegenüber der umströmenden Luft. Die erstere verliert mit steigender Höhe an Genauigkeit, da hier die Werte der Luftdichte und der Temperatur nicht mit eingerechnet werden. Das Feld *Groundspeed* gibt die Geschwindigkeit gegenüber dem Boden an. Die *ClimbRate* beinhaltet die Steiggeschwindigkeit, auch Steigrate genannt.

DataFromAutopilot

Nachrichten dieses Typs werden von Flugregler an Flugsimulator gesendet. Die enthaltenen Felder entsprechen den *CTRLS* Feldern (virtuelle Steuersignale) aus der DataFromAircraft Nachricht. Der einzige Unterschied besteht darin, dass diese Daten vom Flugregler aus gesendet werden. Die Länge der Nachricht beträgt 32 Byte.

5.3.2. Physikalische Kommunikation

Bei der physikalischen Kommunikation muss zwischen Host PC und FCU unterschieden werden. Letztere wird in 6.7.1 erläutert. Auf dem Host PC basiert AESLink auf mehreren UDP Multicast Verbindungen.

Die beiden Nachrichtentypen werden jeweils über eine eigene Multicast-Adresse verschickt. Im Gegensatz zur Verwendung einer einzelnen Adresse für beide Nachrichtentypen, entfällt durch diese Maßnahme das Filtern der Nachrichten in den einzelnen Komponenten. Ein Empfänger registriert sich für den Empfang eines Nachrichtentyps, indem er einen Multicast Socket für die jeweilige Netzwerkadresse öffnet. Die Nachrichten des Typs DataFromAircraft werden über die Adresse *230.0.0.1* und die des Typs DataFromAutopilot über die Adresse *230.0.0.2* versendet. Beide Multicast-Verbindungen verwenden den Port *14658*.

Die Verwendung von statischen Adressen bietet einen zentralen Zugriffspunkt für die Komponenten. Dadurch entfällt das Einrichten der Verbindungen durch den Benutzer. Als Gegenbeispiel sei hier das SIL-Szenario von ArduPilot genannt. Bei dieser muss ein Benutzer jede

einzelne Verbindung (Netzwerkadresse und Port) manuell einrichten. Dies erhöht zwar die Flexibilität der einzelnen Punkt-zu-Punkt Verbindungen, jedoch kann es gleichzeitig zu einer unübersichtlichen „Frickelei“ ausarten.

5.4. Koordinatensysteme

Der in der Nachricht DataFromAircraft enthaltene Flugzustand enthält Positionsangaben in zwei verschiedenen Koordinatensystemen, dem globalen und dem lokalen.

5.4.1. Globales Koordinatensystem

Positionsangaben im *globalen Koordinatensystem* beschreiben die Position relativ zu einem fixen Punkt auf der Erde. Der *Breitengrad* (engl. *Latitude*) beschreibt die relative Position zum Äquator in Grad. Wobei der Nordpol durch +90° nördliche Breite und der Südpol durch +90° südliche Breite beschrieben wird. Fehlt die Angabe der Richtung, liegt der Südpol bei -90°. Der *Längengrad* (engl. *Longitude*) beschreibt die relative Position zum Nullmeridian in Grad. Wobei östliche Länge positiv und westliche Länge negativ gezählt wird. Die Höhe über Meeresspiegel (engl. *above Mean Sea Level*) wird in Metern angegeben.

Jedoch beschreibt das globale Koordinatensystem alleine noch nicht die Position. Dieses muss vorerst auf ein geodätisches Referenzsystem abgebildet werden. Die Referenzkoordinatensysteme werden *Erdmodelle* oder *Referenzellipsoide* genannt. GPS Empfänger verwenden das *WGS 84* Erdmodell. Dieses bildet die Positionen der Erde auf ein Ellipsoid ab. FlightGear verwendet ebenfalls das WGS 84 Erdmodell. X-Plane bildet die Positionen auf eine Kugel ab. Die Krümmung der Erde kann vernachlässigt werden, falls nur eine kleine Fläche betrachtet wird. AeroSimRC verwendet eine planare Oberfläche ohne Krümmung, da die Flächen der Szenarien (in Videospielen *Level* oder *Map* genannt) von AeroSimRC nur wenige Quadratkilometer betragen.

Für die Simulation kann das verwendete Erdmodell vernachlässigt werden, da der AC2030 vorerst nur auf Flugplätzen fliegen wird. Sollten in den kommenden Jahren größere Strecken mit autonomer Navigation abgeflogen werden, sollten diese Punkte jedoch beachtet werden.

Umfangreiche Informationen zum Thema Koordinatensysteme im Zusammenhang mit Navigationssystemen werden in [Wendel (2011)] gegeben.

5.4.2. Lokales Koordinatensystem

Positionsangaben im *lokalen Koordinatensystem* beschreiben die Position relativ zum Startpunkt des Flugzeugs innerhalb des Flugsimulators. Der Ursprung liegt im Schwerpunkt des Flugzeugs, nachdem der Flugsimulator vollständig initialisiert ist und die Rollphase begonnen werden kann. Die X-Achse führt entlang der Längsachse des Flugzeugs, die Y-Achse entlang der Querachse und die Z-Achse entlang der Hochachse. Sollte der Flugsimulator das Flugzeug bei seiner Initialisierung auf einer Startbahn platzieren (typisch), liegen die drei Achsen analog zu der Längs-, Quer- und Hochachse der Startbahn. Positive Werte in den drei Achsen liegen an, wenn sich das Flugzeug nach vorne, nach rechts und nach oben bewegt hat.

Das lokale Koordinatensystem wurde eingeführt, um einen aufgezeichneten Flug in jedem Flugsimulator und auf jeder Startbahn nachfliegen zu können. Unabhängig davon, von welchem Flugsimulator der Flug aufgezeichnet wurde.

5.5. Anbindung der FCU

In diesem Abschnitt wird erläutert, wie die FCU an die Infrastruktur angebunden wird. Hierfür wird ein externer Echtzeitcomputer verwendet. Dieser kommuniziert über eine Proxy Anwendung mit AESLink. Nachfolgend sind Echtzeitcomputer und Proxy beschrieben.

5.5.1. Externer Echtzeitcomputer

Für eine realitätsnahe HIL-Simulation müssen die Schnittstellen der FCU in Echtzeit mit Daten versorgt werden. Ein handelsüblicher PC besitzt weder ein Echtzeitbetriebssystem, noch bietet er die geforderten Schnittstellen der FCU. Aus diesem Grund muss externe Hardware verwendet werden, die diese Anforderungen erfüllen kann. In diesem Bereich etablierte Firmen wie Mathworks, dSPACE oder National Instruments bieten spezielle HIL-Testsysteme an. Diese Systeme sind für das gegebene Problem jedoch hoffnungslos überdimensioniert. Dies spiegelt sich auch in den Preisen der Systeme wider. Die Kosten dieser Systeme liegen weit über dem preislichen Rahmen, der dem AES Projekt zur Verfügung steht. Die für die externe Hardware erforderliche Prozessorleistung ist als gering zu bewerten, da derzeit nur Schnittstellenumsetzung betrieben werden muss. Mehr Leistung wird erst benötigt, wenn auch die Flugphysik auf der externen Hardware und nicht länger mittels eines Flugsimulators auf einem Host PC berechnet wird. Jedoch ist auch hierbei fraglich, wie viel Rechenleitung tatsächlich gefordert ist und ob sich die Anschaffung eines solchen Systems lohnt.

Mit der Verwendung eines Mikrocontrollers oder einer programmierbaren Logik (CPLD, FPGA, etc.) stehen zwei weitaus günstigere Alternativen zur Verfügung. Hier sollen nicht die Unterschiede und Vorteile der einzelnen Bausteine diskutiert werden. Hierfür sei der Leser auf [Lange und Bogdan (2012)] verwiesen.

Ein FPGA hätte den Vorteil, dass sich eine hohe Anzahl an Schnittstellen synthetisieren lassen. Das gegebene Problem erforderte jedoch lediglich drei UART und zwei SPI Schnittstellen. Daher wurde ein Mikrocontroller ausgewählt. Der Prototyp der FCU verwendet das STM32F4-Discovery Board der Firma STMicroelectronics. Die Verwendung dieses Boards liegt nahe, da Source Code in beiden Systemen (FCU und HIL-Testsystem) wiederverwendet werden kann. Ein weiterer Vorteil besteht darin, dass keine eigene Hardware entwickelt und gefertigt werden muss. Das STM32F4-Discovery Board kostet im deutschen Onlinehandel etwa 20,- Euro und stellt damit eine kostengünstige Alternative gegenüber FPGA Evaluation Boards, sowie den oben genannten HIL-Testsystemen dar.

Das Testsystem (im Folgenden *HIL-IO-Board* genannt) hat die Aufgaben, die Schnittstellen der FCU zu bedienen. Für die Eingänge der FCU ist eine Sensoremulation erforderlich. Diese Sensoremulation liefert die Daten so, wie die realen Sensoren sie liefern würden. Dabei müssen die Daten sowohl periodisch in der verwendeten Frequenz, als auch im korrekten Datenformat geliefert werden. Die Ausgänge der FCU senden periodisch Steuersignale an die Aktorik. Diese Steuersignale müssen vom HIL-IO-Board eingelesen werden.

Derzeit steht praktisch keine Firmware der FCU zur Verfügung. Die Komponenten zum Einlesen der Sensorik und Ausgeben der Steuersignale sind noch nicht vorhanden. Daher sollen für die Kommunikation zwischen FCU und HIL-IO-Board Platzhalter eingesetzt werden. Im HIL-IO-Board zählen zu den Platzhaltern die Sensoremulation und das Einlesen der Steuersignale. Für die FCU zählen die Sensordatenerfassung, die Sensordatenaufbereitung, der Flugregler und das Ausgeben Steuersignal zu den Platzhaltern.

5.5.2. Kommunikation mit AESLink

Die über den Bus versendeten Nachrichten müssen physikalisch mit dem HIL-IO-Board ausgetauscht werden. Der Einfachheit halber wird ein USB-to-UART Adapter (Abbildung 5.4) verwendet. Auf der HIL-IO-Board Seite stellt dieser eine UART Verbindung dar. Beim PC ist die Schnittstelle physikalisch eine USB Verbindung. Sie kann programmiertechnisch allerdings wie eine COM-Schnittstelle verwendet werden. Zur Umsetzung der physikalischen AESLink Schnittstelle (Multicast Netzwerk) zur HIL-IO-Board Schnittstelle (COM-Schnittstelle) wird eine Proxy Anwendung verwendet. Die Nachrichten werden von diesem Proxy lediglich durchgereicht. Somit werden bleiben die logischen AESLink Nachrichten erhalten.

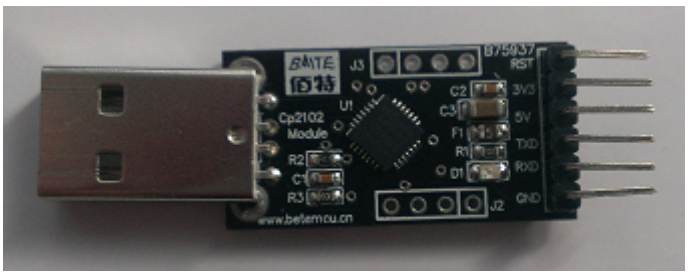

Abbildung 5.4.: Foto des verwendeten USB-to-UART Adapters

5.6. Nebenläufigkeit und Laufzeitverhalten

Dieser Abschnitt beschreibt, wie das Zeitverhalten zwischen den Flugsimulatoren und der FCU, bzw. dem in Simulink modellierten Flugregler konzipiert ist.

In realen technischen Systemen arbeiten Regler, Regelstrecke, Messglied und Stellglied nebenläufig. Die FCU arbeitet sowohl in der realen, als auch in der simulierten Umgebung implizit nebenläufig, da sie in beiden Fällen die tatsächlichen Schnittstellen verwendet, und somit asynchron zur Außenwelt arbeitet. Für das MIL-Szenario muss die Nebenläufigkeit künstlich nachgebildet werden. Das Simulink Modell des Reglers soll daher asynchron zum Flugsimulator laufen.

Die FCU und der in Simulink modellierte Regler arbeiten asynchron im 50 Hz Takt. Sie sollen dabei jeweils auf dem aktuellsten Flugzustand arbeiten. Die Idee ist, dass der Flugsimulator während einer Simulation der *Treiber* ist. Er sendet eine DataFromAircraft Nachricht und wartet synchron auf den Empfang einer DataFromAutopilot Nachricht. Im Simulink Modell, sowie im HIL-IO-Board sollen für jeden Nachrichtentyp ein asynchroner Buffer eingesetzt werden. Diese enthalten jeweils den aktuellen Flugzustand, sowie die zuletzt berechneten Steuersignale. Falls eine DataFromAircraft Nachricht ankommt, soll diese zwischengespeichert werden und ohne Verzögerung die zuletzt berechneten Steuersignale in einer DataFromAutopilot Nachricht zurückgesendet werden. Somit wird eine DataFromAutopilot Nachricht nur in dem Fall gesendet, dass unmittelbar zuvor eine DataFromAircraft Nachricht empfangen wurde.

5.7. Verteilung der Zuständigkeiten in den Szenarien

Nachfolgend wird erläutert, an welchem Ort welche Komponente ausgeführt werden soll. Für eine realitätsnahe Umgebung müssen Sensormodell und Aktuatormodell simuliert werden.

Das Aktuatormodell soll aufgrund der Arbeitsweise der Flugsimulatoren in diesen ausgeführt werden. Daneben sollen auch die Steuerflächenkonfiguration und -zuordnung (Funktion des CSA, Abschnitt 2.1) von den Flugsimulatoren übernommen werden.

Das Sensormodell soll im MIL-Szenario direkt im Simulink Modell ausgeführt werden. Die geforderte Asynchronität zwischen Regler und Sensormodell muss im Modell selbst abgehandelt werden. Hierfür eignen sich *Sample-And-Hold* oder *Delay* Blöcke. Diese Blöcke können ein Signal verzögern und somit eine Totzeit erzeugen.

Im HIL-Szenario wird das Sensormodell im HIL-IO-Board ausgeführt, welches implizit asynchron zur FCU arbeitet. Somit müssen für das HIL-Szenario keine besonderen Vorkehrungen bezüglich der geforderten Asynchronität getroffen werden.

6. Realisierung

In diesem Kapitel wird die Realisierung der Infrastruktur dargestellt. Die im Konzept beschriebenen Komponenten wurden jeweils in einzelnen oder mehreren Anwendungen umgesetzt. Hier soll vorwiegend auf die realisierten Konzepte und weniger auf die Softwarearchitektur der einzelnen Programme eingegangen werden.

6.1. Netzwerkabstraktion von AESLink

Um den Programmieraufwand für die Anwendungen zu verringern, wurde der Zugriff auf AESLink mittels Softwarebibliotheken abstrahiert. Dabei wurden je eine Bibliothek für C++ (*cpplinklib*), C# (*cslinklib*), sowie Simulink entwickelt.

6.1.1. C++ und C#

Die beiden Bibliotheken bieten nach außen die gleichen Funktionalitäten an. Aufgrund der technischen Unterschiede der beiden Sprachen werden diese Funktionalitäten intern jedoch verschieden umgesetzt. Ebenso sind die öffentlichen Methodenköpfe unterschiedlich und erwarten der Sprache entsprechend andere Parameter.

Bei C++ sind die AESLink Nachrichten als Structs, bei C# als Klassen umgesetzt. Intern verwenden beide Bibliotheken jeweils einen UDP Multicast Socket. Die cslinklib steht für andere Anwendungen als .NET DLL zur Verfügung. Die cpplinklib wurde nicht als statische .lib Datei umgesetzt, sie steht als einzelne .cpp/.h Dateien zur Verfügung, welche von den Anwendungen inkludiert, bzw. mitkompiliert werden.

Die Funktionalitäten wurden jeweils in einer Klasse umgesetzt. In beiden Sprachen wird diese Klasse *AESLink_Connection* genannt. Nachfolgend werden deren öffentlichen Methoden beschrieben. Hierbei wird auf die cpplinklib eingegangen. Es werden lediglich nennenswerte technische Unterschiede zur cslinklib genannt.

Subscribe / Unsubscribe
Mit diesen Methoden kann eine Anwendung sich für den Empfang von Nachrichten anmelden,

bzw. abmelden. Als Parameter wird jeweils der Nachrichtentyp übergeben. Technisch sind diese Funktionen so umgesetzt, dass der Socket der jeweiligen Multicast Gruppe beitritt, bzw. diese verlässt.

Broadcast
Diese Methode wird zum Senden von Nachrichten verwendet. Dabei kann als Parameter entweder eine Nachricht ohne MessageHeader oder ein Byte-Buffer inklusive MessageHeader übergeben werden. Bei der ersteren Möglichkeit wird der MessageHeader automatisch angehängt.

Asynchrones Receive
Beim Erzeugen der AESLink_Connection Objekte kann spezifiziert werden, ob Nachrichten synchron oder asynchron empfangen werden. Für das asynchrone Empfangen werden intern Threads verwendet, welche in Endlosschleifen auf eintreffende Nachrichten warten. Beim Eintreffen einer Nachricht wird eine Callback-Methode aufgerufen. Bei der cpplinklib muss beim Erzeugen des Objekts ein Funktionspointer übergeben werden. In der C# Bibliothek werden *C# Events* verwendet. Nach dem Erzeugen des Objekts muss sich eine Anwendung auf diese Events registrieren.

Das Bereitstellen von synchronen und asynchronen Empfangsmechanismen soll eine größtmögliche Flexibilität beim Einbinden und Verwenden der Bibliotheken bieten.

Synchrones Receive
Die *Receive* Methode erwartet einen Byte-Buffer als Parameter. Sie arbeitet nicht-blockierend. Falls mehrere UDP Datagramme ausgelesen werden können, wird lediglich das letzte Datagramm in den Buffer geschrieben. Die vorherigen Datagramme werden verworfen.

WaitForMessage
WaitForMessage wartet auf das Eintreffen einer Nachricht. Hierfür kann ein Timeout übergeben werden. Intern wird der Poll-Mechanismus des Sockets verwendet. Diese Methode kann nur im synchronen Betrieb verwendet werden.

6.1.2. Simulink

In den Abschnitten 6.8 und 6.9 werden Simulink Blöcke und SFunctions beschrieben, die in dieser Arbeit entwickelt wurden. Die Artefakte wurden zu einer Simulink Library, bzw. einem Simulink Blockset zusammengefasst. Durch diese Bibliothek können die Blöcke in anderen Simulink Modellen wiederverwendet werden.

6.2. Anbindung der Flugsimulatoren

Der vorliegende Abschnitt beschreibt die Anbindung von X-Plane, FlightGear und AeroSimRC an AESLink. In Abschnitt 6.2.4 wird auf die Umsetzung der lokalen Koordinatensysteme eingegangen. Screenshots der Flugsimulatoren sind in Anhang C dargestellt.

Definition der Simulator-Modes
Jeder Flugsimulator arbeitet in einem von zwei Modes. In *Mode 1* verteilt ein Flugsimulator den Flugzustand als DataFromAircraft Nachricht über den Bus und empfängt Steuersignale in Form von DataFromAutopilot Nachrichten. Diese Steuersignale dienen dem Simulator-Backend als Steuereingaben für den nächsten Simulationsschritt. In *Mode 2* werden DataFromAircraft Nachrichten empfangen. Die darin enthaltene Position (lokales Koordinatensystem) und Fluglage wird in 3D visualisiert. Das Simulator-Backend ist dabei abgeschaltet, bzw. interferiert nicht mit den empfangenen Werten.

6.2.1. AeroSimRC

Für AeroSimRC wurde ein Plugin in C++ entwickelt. Die cpplinklib arbeitet dabei im synchronen Empfangsmodus. Das Plugin liegt nach dem Kompilieren in Form einer DLL vor. Es muss im Menü von AeroSimRC gestartet werden. Nach dem Starten wird das in Abbildung 6.1 dargestellte Overlay angezeigt. Über den Button *OSD* (On-Screen Display) können verschiedene Werte wie Geschwindigkeit, Position oder Kurs über das Bild gerendert werden. Mit den Buttons *Mode 1* und *Mode 2* kann zur Laufzeit zwischen den Simulator-Modes gewechselt werden. Der Button *Connected* dient lediglich zur Anzeige, ob eine entfernte Komponente verbunden ist. Dies ist in dem Moment der Fall, wenn im letzten Simulationsschritt eine Nachricht empfangen wurde. Der Zustand, ob eine Komponente verbunden ist oder nicht wird in der der Variable *ClientConnected* festgehalten.

Abbildung 6.1.: Overlay des AeroSimRC Plugins

Nach jedem Simulationsschritt wird die Hauptfunktion des Plugins aufgerufen. Die Übergabeparameter des Aufrufs sind zwei referenzierte Datenstrukturen. Die erste Datenstruktur enthält den Flugzustand. Die zweite die Rückgabewerte des Plugins, welche ein neuer Flugzustand

oder Steuersignale des (Auto-)Piloten sein können. Per Flags in der zweiten Datenstruktur kann gesteuert werden, welche Daten im Simulator-Backend durch das Plugin überschrieben werden sollen. Innerhalb des Plugins arbeitet ein Zustandsautomat. Je nach verwendetem Mode werden von diesem unterschiedliche Aktionen ausgeführt.

Mode 1
In Mode 1 wird der Flugzustand und die Steuereingaben des Anwenders (Tastatur oder RC-Fernbedienung) in einer DataFromAircraft Nachricht über den Bus verteilt.

- Falls ClientConnected wahr ist, wird synchron auf eine Antwort gewartet. Im Falle einer empfangenen Nachricht werden die Steuersignale an AeroSimRC zurückgegeben und ClientConnected auf wahr belassen. Falls nach einem Timeout keine Nachricht empfangen wurde, werden keine Steuersignale an AeroSimRC zurückgegeben und ClientConnected wird auf unwahr gesetzt.
- Falls ClientConnected unwahr ist, wird ohne Wartezeit überprüft, ob während des letzten Simulationsschrittes eine Nachricht empfangen wurde. Im Falle einer empfangenen Nachricht wird die ClientConnected auf wahr gesetzt und die empfangenen Steuersignale an AeroSimRC zurückgegeben. Falls keine Nachricht empfangen wurde, wird ClientConnected auf unwahr belassen und keine Steuersignale an AeroSim zurückgegeben.

Durch das synchrone Warten ergibt sich die in 5.6 geforderte Asynchronität zwischen Regler und Regelstrecke.

Mode 2
In Mode 2 verhält sich der Zustandsautomat analog zu Mode 1. Jedoch wird in diesem nicht der Flugzustand über den Bus verteilt. Weiterhin sind die Rückgabewerte keine Steuersignale, sondern eine vorgegebene Position und Fluglage. Hierbei wird der vom Simulator-Backend berechnete Flugzustand überschrieben.

In beiden Modes wird nur eine bestimmte Wartezeit (per Konfigurationsdatei einstellbar) auf eine Nachricht einer entfernten Komponente gewartet. Falls keine Nachricht eintrifft, wird im Durchlauf des Plugins gar nicht gewartet. Sofern die Wartezeit nur wenige Millisekunden beträgt, ist der Übergang zwischen verbundenem und unverbundenem Zustand fließend und somit für den Anwender transparent. Dadurch ergibt sich ein angenehmes Arbeiten ohne Verzögerung oder kurzzeitiges Einfrieren des Flugsimulators.

6.2.2. X-Plane

Für X-Plane wurde ebenfalls ein Plugin in C++ entwickelt. Hierfür wurde das X-Plane SDK von Sandy Barbour verwendet[1]. Nach dem Starten des Plugins wird ein Overlay Fenster (Abbildung 6.2) oberhalb des gerenderten Bildes angezeigt. Die Checkboxen *Mode 1*, *Mode 2* und *Client Connected* haben die gleichen Funktionen wie die Steuerelemente des AeroSimRC Plugins. Auch die interne Verarbeitung des X-Plane Plugins gleicht in vielerlei Hinsicht der des AeroSimRC Plugins.

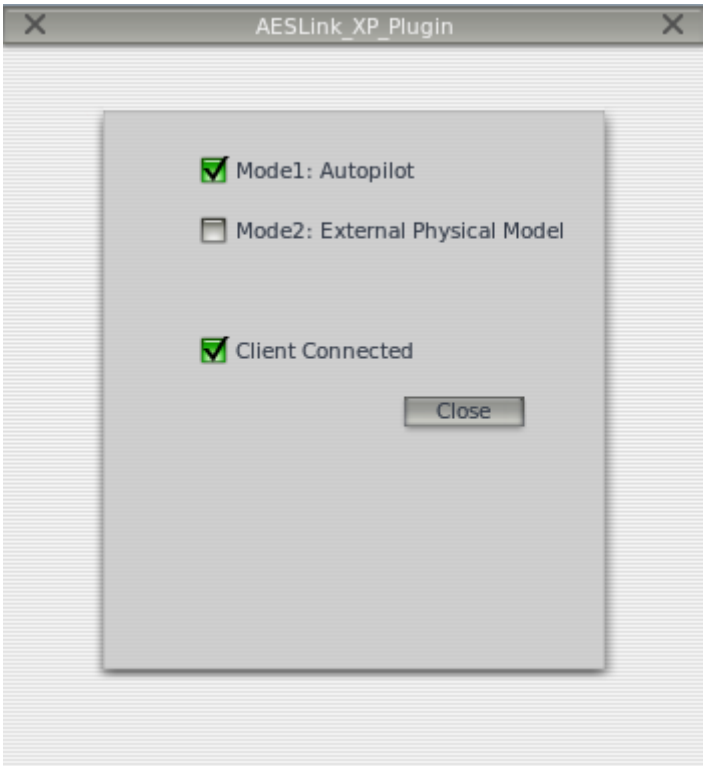

Abbildung 6.2.: Overlay des X-Plane Plugins

Die Kommunikation und der Datenaustausch zwischen Plugin und Simulator werden über Funktionen des SDKs abgehandelt. Beim Starten des Plugins wird am Flugsimulator eine Callback-Funktion registriert. Diese wird bei jedem Simulationsschritt aufgerufen. Im Plugin wird ein zu AeroSimRC funktionsgleicher Zustandsautomat verwendet, welcher beim Aufruf der Callback-Funktion durchlaufen wird. Für den Datenaustausch werden beim Starten des

[1] X-Plane - X-Plane SDK, http://www.xsquawkbox.net/xpsdk/mediawiki/FrontPage [Stand: 30.05.2014]

Plugins Referenzen anhand von Strings erstellt. Die Referenzen werden im Zustandsautomaten gelesen, bzw. geschrieben.

Durch den zu AeroSimRC funktionsgleichen Zustandsautomaten ergibt sich die in 5.6 geforderte Asynchronität zwischen Regler und Regelstrecke.

6.2.3. FlightGear

Anders als für AeroSimRC und X-Plane wurde für die Anbindung an FlightGear kein Plugin entwickelt. Stattdessen wird die native Netzwerkschnittstelle von FlightGear verwendet. Hierfür wurde eine Proxy Anwendung entwickelt, welche die Daten zwischen AESLink und FlightGear vermittelt. Als Programmiersprache wurde C# verwendet. Abbildung 6.3 zeigt die Anwendung.

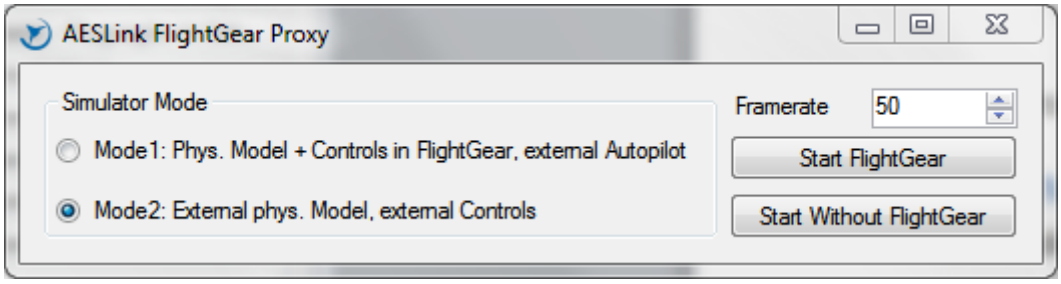

Abbildung 6.3.: Screenshot der FlightGear Proxy Anwendung

Der logische Datenaustausch zwischen FlightGear und einer externen Anwendung kann auf zwei Arten erfolgen. Die erste Möglichkeit[2] (*Native-Protocol*) besteht darin, vorgegebene Datenpakete auszutauschen. Ein *Net_FDM* Datenpaket enthält den vom Simulator-Backend berechneten Flugzustand und ein *Net_CTRLS* Paket die Steuersignale. Beide Pakettypen können sowohl eingelesen, als auch ausgegeben werden. Dies ermöglicht es, FlightGear sowohl als Regelstrecke, als auch als reine Datensenke zur Visualisierung zu verwenden. Dieser Ansatz hat sich jedoch als nicht praktikabel herausgestellt, da er einen zu großen Overhead erzeugte und die Datenpakete nicht alle für das AESLink Protokoll benötigten Daten enthalten. Aus diesem Grund wurde der zweite Ansatz[3] (*Generic-Protocol*) gewählt. Dieser erlaubt es, beliebige Daten vom Simulator-Backend zu empfangen. Hierfür werden mithilfe von XML-Dateien eigene Datenpakete angefordert, bzw. FlightGear kann für den Empfang von beliebigen Datenpaketen konfiguriert werden. Zur Umsetzung beider Simulator-Modes wurden drei

[2] FlightGear - Property Tree/Sockets, http://wiki.flightgear.org/Property_Tree/Sockets [Stand: 30.05.2014]

[3] FlightGear - Generic protocol, http://wiki.flightgear.org/Generic_protocol [Stand: 30.05.2014]

XML-Dateien erstellt. In Mode 1 wird ein Datenpaket von FlightGear empfangen und eines zurückgesendet. Das empfangene stellt den Flugzustand und das gesendete die Steuersignale dar. In Mode 2 werden lediglich die Werte der sechs Freiheitsgrade (Position und Fluglage) an FlightGear gesendet.

Für eine Netzwerkkommunikation mit einer externen Anwendung muss FlightGear mit speziellen Parametern gestartet werden. Die Parameter enthalten, mit welcher Frequenz FlightGear die Datenpakete senden, bzw. empfangen und einlesen soll, sowie die für die Kommunikation verwendeten Netzwerkadressen. Die Datenrate kann im Proxy über das *Framerate* Steuerelement konfiguriert werden. Die Netzwerkadressen sind im Proxy fest vorgegeben und bedürfen keiner Anpassung durch einen Anwender. Beim Starten mit Mode 2 wird FlightGear mitgeteilt, dass das Simulator-Backend deaktiviert werden soll.

Intern verwendet der Proxy die cslinklib im asynchronen Empfangsmodus. Nachrichten, die der Proxy von AESLink empfängt, werden in den mittels der XML-Dateien spezifizierten Datenpaketen an FlightGear weitergeleitet. Für Mode 1 wird ein spezieller asynchroner Empfangsthread verwendet. Dieser bildet aus den von FlightGear empfangenen Daten eine DataFromAircraft Nachricht und sendet sie über den Bus.

In Mode 1 wird das Simulator-Backend beim Warten auf Steuersignale eines Autopiloten nicht angehalten. Die kurz darauf empfangenen Steuersignale werden dadurch frühestens im nachfolgenden Simulationsschritt verarbeitet. Dies kann zu Problemen bei der Steuereingabe durch einen Anwender führen, da sich die Daten zeitversetzt (um einen Simulationsschritt) gegenseitig überschreiben können. Bei diskreten Steuereingaben wie etwa das Ein- oder Ausfahren des Fahrwerks kann dies dazu führen, dass sich der Zustand hin- und herschaltet, ohne dass der Anwender eine weitere Eingabe tätigt. Von diesem Problem sind lediglich die FCU-Modes *manuell* und *unterstützend* betroffen.

6.2.4. Lokales Koordinatensystem in AESLink

AeroSimRC und X-Plane verwenden intern ebenfalls lokale Koordinatensysteme. Der Einfachheit halber wurden diese für die Realisierung verwendet. Der Koordinatenursprung liegt bei keinem der beiden Flugsimulatoren in der Startposition der Flugzeuge. Daher wird eine Koordinatentransformation zwischen Ursprung des Koordinatensystems und Startposition des Flugzeugs durchgeführt. Beim Starten der Plugins werden jeweils die Offsetvektoren von Flugzeug und Ursprung, sowie der Rotationswinkel zwischen lokalem Koordinatensystem und Flugzeugausrichtung gespeichert.

In Mode 1 wird mithilfe des Offsets das Koordinatensystem des Flugsimulators auf das Koordinatensystem der Startposition des Flugzeugs transformiert. Die daraus resultierenden Positionsdaten bilden die Werte für die Felder *LocalX*, *LocalY* und *LocalZ*. In Mode 2 werden

die übermittelten Positionsdaten anhand der gespeicherten Offsets in das Koordinatensystem des Flugsimulators rücktransformiert. Hierbei ist eine zusätzliche Rotation erforderlich, welche die Positionsdaten um den beim Start gespeicherten Rotationswinkel dreht. Ohne diese Maßnahme würde das Flugzeug zwar der korrekten Flugbahn folgen, jedoch wäre es um den Rotationswinkel in der Gierachse verdreht. Visuell würde sich dadurch ein seitliches Schieben des Flugzeugs ergeben.

Für FlightGear wird im Proxy gleich verfahren. Jedoch besitzt FlightGear kein eigenes lokales Koordinatensystem. Aus diesem Grund wird das globale Koordinatensystem mit dem Ursprung Äquator, Nullmeridian und Meeresspiegel verwendet.

6.3. Aufzeichnung

Für die Aufzeichnung wurde mittels C# und der cslinklib die Kommandozeilenanwendung *Logger* entwickelt. Dieses registriert sich für beide Nachrichtentypen am Bus und legt eine leere Datei an, welche von den Komponenten aus 6.4 und 6.5.1 verwendet werden kann. Die empfangenen Nachrichten werden mit einem Zeitstempel versehen. Der Zeitstempel und die Nachricht werden binär in der angelegten Datei gespeichert. Die cslinklib Bibliothek arbeitet im asynchronen Empfangsmodus. Dies erleichtert das saubere Beenden der Anwendung. Durch ein Drücken der ESC-Taste beendet der Logger und gibt eine Statistik über die Zeit und die Anzahl der empfangenen Nachrichten aus.

6.4. Wiedergabe

Zur live Wiedergabe aufgezeichneter Daten wurden zwei Anwendungen entwickelt. Die Kommandozeilenanwendung *ReplayCLI*, sowie eine grafische Anwendung *Replay*. Beide Anwendungen wurden in C# unter Zuhilfenahme der cslinklib entwickelt. Sie bieten die Möglichkeit, nach einem der beiden Nachrichtentypen zu filtern. Zum Einlesen der aufgezeichneten Daten wird eine gemeinsame Codebasis verwendet.

6.4.1. ReplayCLI

Das Ziel der *ReplayCLI* Anwendung ist es, die Daten nach Möglichkeit in den gleichen Zeitabständen über den Bus zu versenden, wie sie vom Logger empfangen wurden. Beim Start wird die gesamte Datei eingelesen und nach Wunsch des Anwenders gefiltert. Nach dem Einlesen wird ein Thread gestartet. Dieser sendet die erste eingelesene Nachricht über den Bus. Daraufhin schläft er, bis er die nächste Nachricht senden soll. Diese Schleife läuft solange, bis die

letzte Nachricht gesendet wurde. Der Anwender hat außer beim Starten der Anwendung keine Konfigurationsmöglichkeiten.

6.4.2. Replay

Die *Replay* Anwendung besitzt im Gegensatz zu ReplayCLI eine grafische Benutzeroberfläche. In Abbildung 6.4 ist ein Screenshot der Anwendung dargestellt. Die Steuerelemente sind an Multi Media Wiedergabe Anwendungen wie etwa dem VLC Media Player[4] angelehnt. Die aus der aufgezeichneten Datei eingelesenen Nachrichten werden nachfolgend als *Datenstrom* bezeichnet. Ein Datensatz innerhalb des Datenstroms beinhaltet genau eine Nachricht.

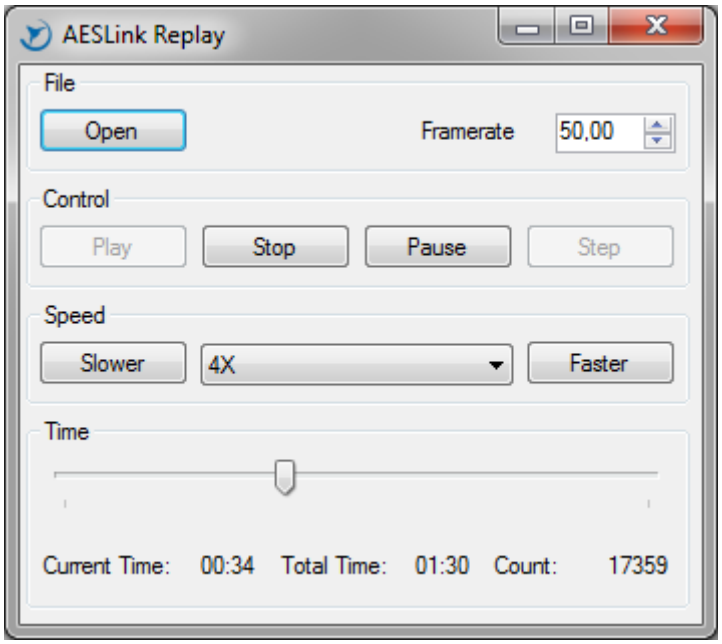

Abbildung 6.4.: Screenshot der Replay Anwendung

Folgende Steuerelemente wurden umgesetzt:

- **Open**: Öffnen einer aufgezeichneten Datei
- **Play, Stop**: Starten und Stoppen der Wiedergabe
- **Pause, Step**: Pausieren und schrittweises Fortführen der Wiedergabe
- **Slower, Faster**: Schrittweises Verändern der Abspielgeschwindigkeit

[4] VLC media player, http://www.videolan.org/vlc/ [Stand: 30.05.2014]

- **Speed (ComboBox)**: Direktes Konfigurieren der Abspielgeschwindigkeit
- **Timeline (Trackbar / Slider)**: Anzeigen und Verändern der Position innerhalb des Datenstroms
- **Current Time**: Zeigt die aktuelle Position innerhalb des Datenstroms in Minuten und Sekunden an
- **Total Time**: Zeigt die Gesamtabspieldauer in Minuten und Sekunden an
- **Count**: Die Anzahl der Nachrichten im Datenstrom
- **Framerate**: Zum Einstellen der Senderate

Intern (*ReplayEngine*) wird ein Thread verwendet, der mit einer bestimmten Frequenz die eingelesenen Nachrichten über den Bus verteilt. Die Frequenz (Senderate) kann über die Benutzeroberfläche verändert werden. Es wird dabei immer die Nachricht verteilt, die zu dem Zeitpunkt am kürzesten zurückliegt. Die Abspielgeschwindigkeit kann in neun exponentiell steigenden Stufen (Basis 2) von 0,125-facher bis 32-facher Geschwindigkeit variiert werden. Mit dieser Geschwindigkeit wird die Schrittweite eingestellt. Diese Schrittweise gibt an, wie viele Datensätze der Thread überspringen und damit nicht senden soll.

Zur Verdeutlichung wird ein Beispiel herangezogen. Mit einer Framerate von 20 werden pro Sekunde 20 Nachrichten über den Bus verteilt. Bei einer 1-fachen Geschwindigkeit verteilt der Thread die Nachricht über den Bus, die 50msec später im Datenstrom liegt. Bei 4-facher Geschwindigkeit beträgt die Schrittweite 4*50msec. Bei 0,25-facher Geschwindigkeit 0,25*50msec und so weiter.

6.4.3. Erweiterung FDLgcs

Die für die Replay Anwendung entwickelte ReplayEngine wurde in einem älteren Programm (*FDLgcs*) wiederverwendet. Bei diesem handelt es sich um die Ground Control Station des alten Messsystems (Flugdatenlogger). Die FDLgcs wird verwendet, um die aufgezeichneten Daten (Flugversuche) der letzten Jahre zu visualisieren. Sie wurde dahingehend erweitert, dass aus den aufgezeichneten Daten AESLink Nachrichten generiert werden und diese über den Bus verteilt werden. Dies ermöglicht die Visualisierung der Flugversuche, die in den letzten Jahren vom BWB-Team durchgeführt wurden.

6.5. Visualisierung

Zur besseren Visualisierung des Flugzustands wurden zwei Anwendungen entwickelt. Der *Plotter* zeigt den Flugzustand in Diagrammen an. Die *Gauges* Anwendung verwendet hingegen analoge Cockpitinstrumenten zur Visualisierung.

6.5.1. Plotter

Der Plotter ist durch ein Matlab Skript realisiert. Er besitzt zwei Funktionalitäten. Zum einen kann er live Daten, die in Realzeit über den Bus gesendet werden, plotten. Die zweite Funktionalität besteht darin, dass alle Nachrichten einer aufgezeichneten Datei geplottet werden können. Matlab unterstützt die Verwendung von .NET Bibliotheken. Um den Zugriff auf AES-Link nicht in Matlab umsetzen zu müssen, wird hierfür die cslinklib verwendet.

Die Daten werden in insgesamt 16 verschiedenen Diagrammen angezeigt. Hierbei kann wahlweise entschieden werden, ob diese in einem großen oder vier kleineren Fenstern dargestellt werden. Bei der Verwendung der kleinen Fenster besteht für den Anwender die Möglichkeit, vereinzelt Fenster nicht anzeigen zu lassen. Abbildung 6.5 zeigt das große Fenster mit allen Diagrammen. Die Diagramme zeigen primär den Flugzustand an, die Steuersignale vom Autopiloten werden überlagert angezeigt. Durch diese Überlagerung können Sprünge und Aussetzer entstehen, falls für einen Zeitraum keine DataFromAutopilot Nachricht empfangen, bzw. eingelesen wird.

Live Daten
Zum Zugriff auf den Bus werden zwei Instanzen der AESLink_Connection im asynchronen Empfangsmodus verwendet, wobei pro Nachrichtentyp eine Instanz verwendet wird. Zur Aktualisierung der Diagramme wird ein Matlab Timer genutzt. Der Timer prüft periodisch beide Instanzen nach empfangenen Nachrichten. Zur Wahrung der Übersicht wird die Anzahl der angezeigten Datensätze begrenzt, wobei jeweils die aktuellsten Datensätze angezeigt werden. Durch dieses Konzept ergeben sich zu jedem Zeitpunkt saubere Kurven ohne Sprünge und Aussetzer. Die Frequenz des Timers und die Anzahl der anzuzeigenden Datensätze sind konfigurierbar. Hier müssen je nach Rechnergeschwindigkeit gegebenenfalls Abstriche gemacht werden.

Aufgezeichnete Daten
Diese Funktionalität dient dazu, alle Datensätze einer aufgezeichneten Datei zu plotten. Zum Einlesen einer Datei werden Klassen aus den in 6.4 beschriebenen Anwendungen wiederverwendet. Es werden alle aufgezeichneten Nachrichten geplottet. Bei zeitlich langen Simulationsdurchläufen können aufgrund der beschränkten Bildschirmgröße nicht alle Datensätze zur gleichen Zeit betrachtet werden. Allerdings erlaubt Matlab das kontinuierliche Hereinzoomen, durch welches das Problem umgegangen werden kann.

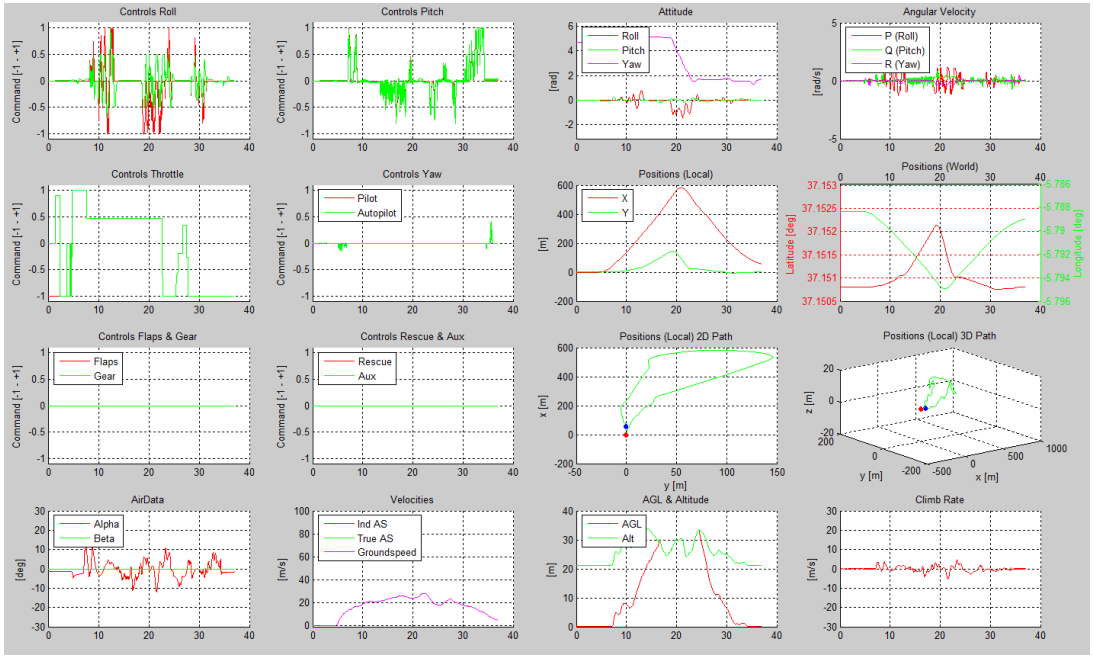

Abbildung 6.5.: Screenshot des Matlab Plotters

Bei live Daten ist es nicht möglich, dass zwei DataFromAutopilot Nachrichten aufeinander folgen. Beim Plotten von aufgezeichneten Daten wird dies jedoch durch eine Filterung der Nachrichtentypen beim Einlesen der Datei ermöglicht. Anders als bei der live Variante wird hier kein asynchron arbeitender Timer verwendet. Dieser Umstand führt dazu, dass Sprünge und Lücken anders kompensiert werden müssen. Falls zweimal in Folge der gleiche Nachrichtentyp eingelesen wurde, wird für den jeweils anderen Nachrichtentyp die letzte eingelesene Nachricht als Datensatz verwendet. Hierdurch ergeben sich saubere Kurven, jedoch geht so auch die Information verloren, wann die letzte Nachricht eines jeweiligen Typs über den Bus versendet wurde. Mit einem zweiten Blick auf ein Diagramm lässt sich dies allerdings erahnen, da fortan keine Veränderung mehr zu erkennen ist. Ob sich diese Unzulänglichkeit im Betrieb negativ auswirkt, muss ein Anwender selbst für sich entscheiden.

6.5.2. Gauges

Die in C# entwickelte Gauges Anwendung zeigt Teile des aktuellen Flugzustands anhand analoger Cockpit-Instrumente an. Zu den Instrumenten zählen zwei künstliche Horizonte, ein Kompass, ein Altimeter, ein Variometer, ein Fahrtmesser, sowie ein Wendezeiger. Eine mittels OpenGL dargestellte Schachtel im dreidimensionalen Raum zeigt die Fluglage in allen

drei Achsen an. Zur Anzeige von Piloten- und Autopilotensteuersignalen werden selbst erstellte Progress Bars verwendet, wobei je zwei von diesen die Steuersignale für Rollen, Nicken, Schub und Gieren anzeigen. Abbildung 6.6 zeigt einen Screenshot der Anwendung.

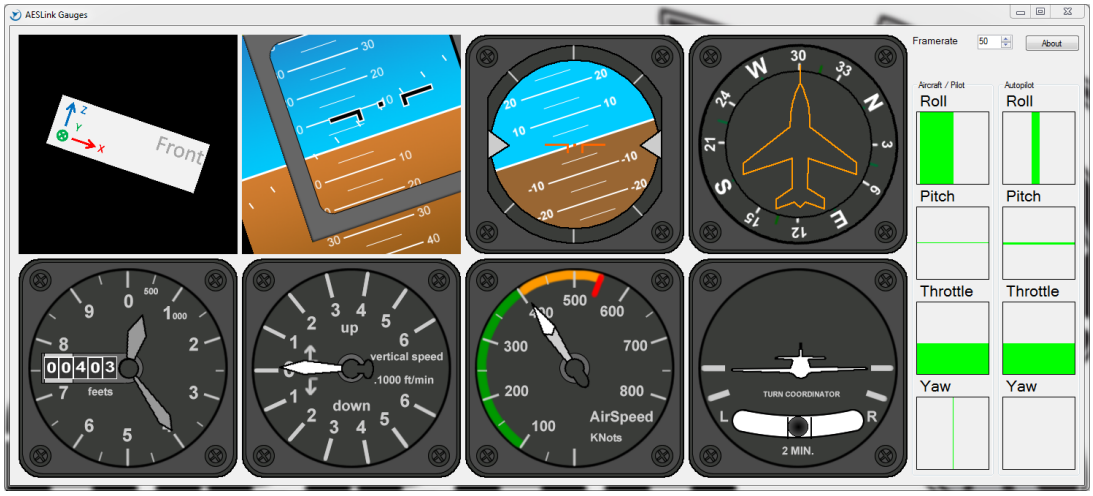

Abbildung 6.6.: Screenshot der Gauges Anwendung

Aus Performancegründen werden die Anzeigeelemente nicht bei jeder eintreffenden Nachricht aktualisiert. Stattdessen wird ein Timer mit zur Laufzeit konfigurierbarer Frequenz, bzw. gewünschter Updaterate verwendet. Die pro Nachrichtentyp jeweils letzte eingetroffene Nachricht wird zwischengespeichert. Bei jedem Update des Timers werden die Anzeigeelemente mit den beiden zwischengespeicherten Nachrichten aktualisiert.

Die Gauges Anwendung diente während der Entwicklung der Infrastruktur als Proof-of-Concept- und Demonstrationsanwendung. Weiterhin wurde sie zur Normierung der Daten der Flugsimulatoren verwendet. Inwiefern sich die Anwendung produktiv einsetzen lässt, muss ein Anwender selbst entscheiden.

6.6. SIL-Szenario / Proof-of-Concept Autopilot

Mithilfe von C# und der cslinklib wurde ein Autpilot entwickelt. Dieser diente ähnlich wie die Gauges Anwendung zu Demonstrations- und Testzwecken, sowie zur Normierung des Flugzustands. Der interne Flugregler verwendet mehrere PID-Regler, wobei die Reglerparameter auf ein Flugmodell aus AeroSimRC abgestimmt sind. Abbildung 6.7 zeigt schematisch den Aufbau des internen Flugreglers. Der Flugregler deckt Funktionalitäten aus den Kategorien Guidance und Control ab. Dabei können verschiedenen Sollwerte wie Kurs, Höhe, Geschwindigkeit oder

Anstellwinkel vorgegeben werden. Für jede Drehachse des Flugzeugs können Dämpfer eingesetzt werden, welche gemeinsam ein Stability Augmentation System bilden. Ein Anwender kann zwischen drei verschiedenen Flight Control Laws wählen. Die Piloteneingaben können dabei als gewünschter Steuerflächenausschlag, gewünschte Drehraten oder gewünschte Fluglage verwendet werden. Bei den beiden letztgenannten Flight Control Laws werden die Laws lediglich für Rollen und Nicken umgesetzt, ein Gieren bewirkt Steuerflächenausschläge und verhält sich somit wie das erste Flight Control Law. Für die ersten beiden Flight Control Laws kann eine Flight Envelope Protection verwendet werden, welche zu große Lagewinkel in der Längs- und der Querachse verhindert. Der Abbildung 6.8 können weitere Einstellmöglichkeiten entnommen werden.

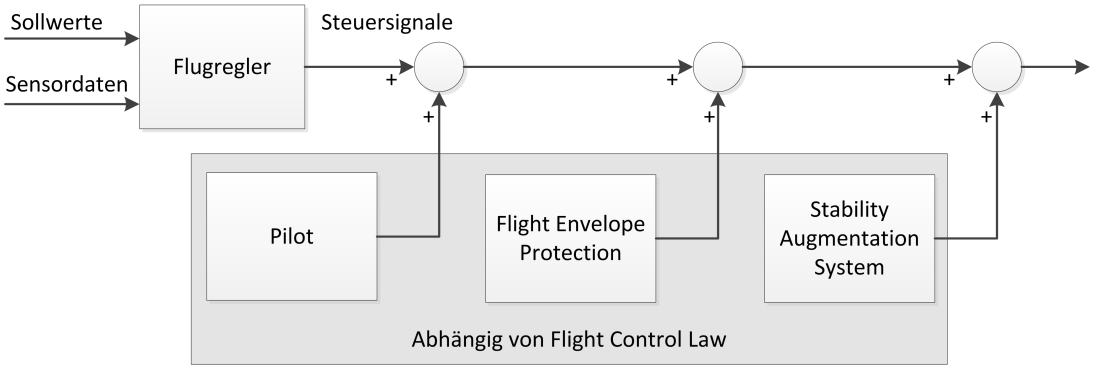

Abbildung 6.7.: Struktur des in Software implementierten Flugreglers. Piloteneingabe, Flight Envelope Protection und Stability Augmentation System sind vom verwendeten Flight Control Law abhängig

6.7. HIL-Szenario / FCU

Zur Anbindung der FCU sind zwei Komponenten erforderlich. Zum einen das externe HIL-IO-Board, welches die physikalischen Schnittstellen der FCU bedient. Zum anderen die Proxy Anwendung, welche die Nachrichten zwischen AESLink und HIL-IO-Board vermittelt.

6.7.1. HIL Proxy

Der HIL Proxy ist eine in C# entwickelte Anwendung. Abbildung 6.9 zeigt die Benutzeroberfläche. Ein Anwender muss die COM-Schnittstelle wählen, mit welcher der USB-to-UART Adapter verbunden ist. Der Button *Connect and Start* baut die Verbindung zwischen HIL-IO-Board und AESLink auf. Der Proxy registriert sich beim Bus für DataFromAircraft Nachrichten. Wenn

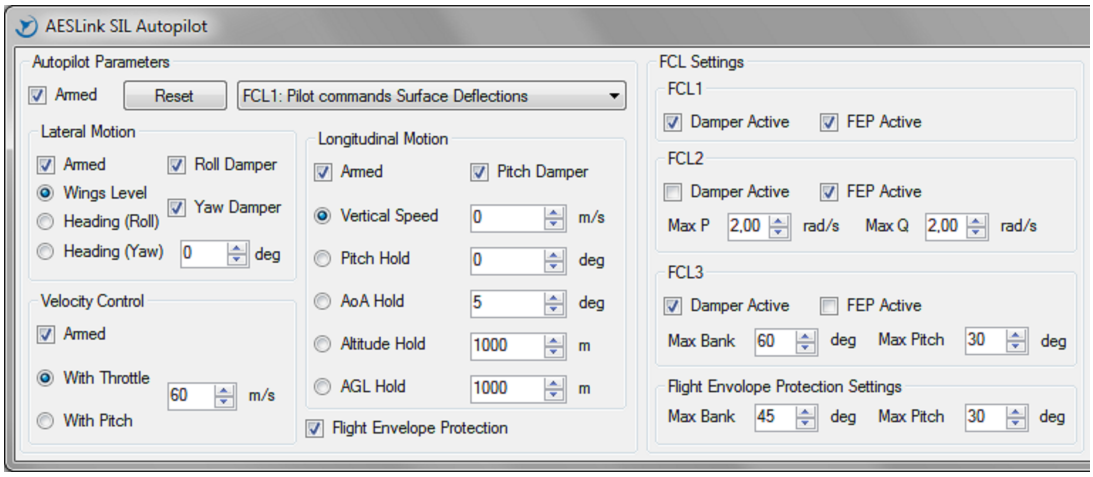

Abbildung 6.8.: Screenshot der Benutzeroberfläche des in Software implementierten Flugreglers

eine Nachricht vom Bus empfangen wurde, werden die Daten unverändert über die COM-Schnittstelle weiterleitet. Vom HIL-IO-Board werden logische DataFromAutopilot Nachrichten empfangen. Diese werden ebenfalls unverändert weitergeleitet. Bei Empfang von Nachrichten über die COM-Schnittstelle muss sich der Proxy (auf-)synchronisieren. Diese Maßnahme ist nötig, da die Verbindung Stream-basiert aufgebaut ist. Für diesen Zweck wurde das Magic-Number Feld im MessageHeader eingeführt.

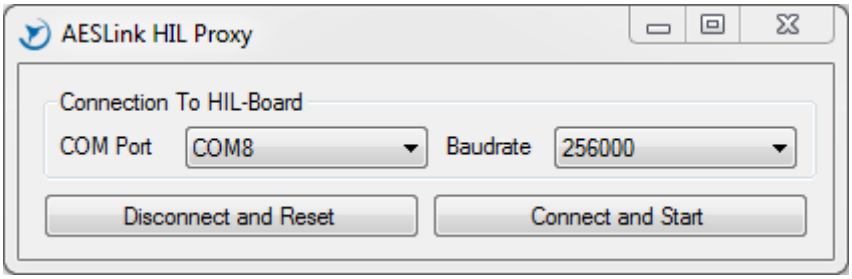

Abbildung 6.9.: Screenshot der HIL Proxy Anwendung

6.7.2. FCU und HIL-IO-Board

Als Hardware der FCU und des HIL-IO-Boards wird als erster Prototyp jeweils ein STM32F4-Discovery Board verwendet. In späteren Iterationen des AES Projekts sollen die darauf enthaltenen STM32F4 Mikrocontroller auf selbst entworfenen Platinen verwendet werden.

Für die Firmware der beiden Boards wurde zunächst eine objektorientierte Hardwareabstraktionsschicht (*HAL*) entwickelt. Diese enthält Klassen zur Verwendung von UART, SPI, I2C, GPIO, Timer, PWM und weiteren Peripheriekomponenten des STM32F4. Neben der HAL selbst wurde eine Initialisierungsumgebung für diese entwickelt. Die HAL und die Initialisierungsumgebung zusammen ermöglichen es, dass FCU und HIL-IO-Board auf der gleichen Codebasis ausgeführt werden können. In der jeweiligen Main-Funktion der beiden Firmwares werden lediglich unterschiedliche Klassen instanziiert, welche die eigentliche Anwendungslogik der jeweiligen Firmware enthalten.

FCU

Die Firmware der FCU wurde von Grund auf selbst entwickelt, da praktisch noch kein Source Code für diese vorlag. Die vorliegende Arbeit hatte nicht zum Ziel, die Firmwarekomponenten der FCU vollständig zu entwickeln. Es soll lediglich verifiziert werden, dass das entwickelte Konzept lauffähig ist. Daher wurden Platzhalter entwickelt, die in den folgenden Semestern mit gewünschter Funktionalität gefüllt, bzw. ausgetauscht werden können.

Wie in 2.1 beschrieben, besitzt die FCU als Eingänge drei UART und eine SPI Schnittstelle, sowie als Ausgang eine weitere SPI Schnittstelle. Das Einlesen der vier verschiedenen Eingänge erfolgt jeweils mit asynchronen Interrupt Service Routinen (*ISR*). Für jeden Sensor wurde eine Klasse entwickelt, welche das Einlesen der Werte übernehmen. Zusammen ergeben die Klassen die Firmwarekomponente Sensordatenerfassung.

Für das Einlesen der Daten werden jeweils zwei unterschiedliche Buffer verwendet, von denen einer in den ISRs mit Daten gefüllt wird. Dies geschieht unabhängig von der Taktfrequenz der Sensorik. Beim Durchlaufen der Hauptschleife werden die einzelnen Sensorklassen abgefragt. Diese enthalten jeweils ihre eigenen Buffer, bzw. Datenstrukturen, welche die Messwerte enthalten. Beim Aufruf einer Sensorklasse wird der Buffer der ISR dahingehend überprüft, ob ein kompletter Datensatz eines Sensors empfangen wurde. In diesem Fall wird der Datensatz aus dem ISR Buffer ausgelesen, interpretiert und in den eigenen Buffer verschoben. Durch diesen Ansatz ist das Einlesen der Sensordaten nicht an die Taktfrequenz der Sensoren gebunden.

Die Hauptschleife führt mit einer Frequenz von 50Hz nacheinander die Komponenten Sensordatenerfassung, Sensordatenaufbereitung und Flugregelung aus. Die Sensordatenaufbereitung enthält aktuell dabei keine Logik und reicht die Daten unverändert hindurch. Der Flugregler besteht aus zwei einfachen Regelschleifen, welche die Fluglage in der Längs- und Querachse stabilisieren. Die vom Flugregler berechneten Steuersignale werden am Ende der Hauptschleife per SPI ausgegeben. Zur besseren Nachvollziehbarkeit werden die eingelesenen und ausgegebenen Daten über einen weiteren UART Kanal als ASCII-Texte ausgegeben. Mittels

eines zweiten USB-to-UART Adapters können die Daten am Host PC dargestellt und analysiert werden, für welches sich das Programm HTerm[5] eignet.

HIL-IO-Board

Das HIL-IO-Board hat die Aufgabe, die Schnittstellen der FCU zu bedienen. Die Anforderung an das HIL-IO-Board ist dabei, dass die FCU nicht zwischen simulierter und realer Umgebung unterscheiden kann. Die Eingabedaten der FCU müssen in den in 2.1 genannten Zyklen bereitgestellt werden und die berechneten Steuersignale der FCU eingelesen werden.

Für die Kommunikation mit der FCU müssen die Sensoren im Zeitverhalten und im korrekten Datenformat emuliert werden. Jeder Sensor wird über eine eigene Klasse emuliert. Die Gesamtheit der Klassen bildet die Komponente zur Sensoremulation. Sie ist der vorgesehene Platzhalter für die Implementierung des Sensormodells. Für diese Arbeit wurde die Sensoremulation lediglich rudimentär durchgeführt. Die Datenformate spiegeln dabei nicht die der tatsächlichen Sensoren wider. Stattdessen werden die rohen Daten in einem Platzhalter-Format übertragen. Folgende Daten werden dabei ausgetauscht:

- **AHRS**: Roll-, Nick-, Gierwinkel
- **GPS**: Globales Koordinatensystem (Breitengrad, Längengrad, Höhe über Meeresspiegel)
- **ADS**: Airspeed, Anstellwinkel, Schiebewinkel
- **Pilot**: Virtuelle Steuersignale (acht Kanäle)
- **Autopilot (Ausgabe)**: Virtuelle Steuersignale (acht Kanäle)

Die Kommunikation zwischen FCU und HIL-IO-Board ist asynchron, wohingegen die Kommunikation zwischen AESLink (HIL Proxy) und HIL-IO-Board synchron verläuft. Beim Empfang einer Nachricht vom Proxy werden die darin enthaltenen Daten den Klassen der Sensoremulation übergeben. Diese beinhalten jeweils einen Buffer, in dem die Daten vorgehalten werden. Zur isochronen Übertragung der Daten aus den Buffern wird jeweils ein Timer verwendet. Beim Auslösen der Timer werden die Daten über die jeweiligen Schnittstellen versendet. Die Rückgabewerte der FCU werden asynchron eingelesen und ebenfalls in einem Buffer gespeichert. Die Hauptschleife wartet synchron auf DataFromAircraft Nachrichten. Beim Empfang werden die Buffer zur Emulation gefüllt. Die Daten aus dem Empfangsbuffer der Steuersignale werden direkt in einer DataFromAutopilot an den HIL Proxy zurückgesendet.

Durch den verwendeten Ansatz ergeben sich möglichst kurze Laufzeiten zwischen Regler und Regelstrecke.

[5] HTerm, http://www.der-hammer.info/terminal/ [Stand: 30.05.2014]

6.8. MIL-Szenario / Flugregler in Simulink

Für das MIL-Szenario wurde ein Simulink Blockschaltbild modelliert. Die oberste Hierarchieebene ist in Abbildung 6.10 dargestellt. Die abgebildeten Blöcke stellen jeweils Subsysteme dar. Die Blöcke *Sensor Model*, *State Estimator* und *Autopilot* sind jeweils Platzhalter, die in den kommenden Semestern mit Funktionalität gefüllt werden können. Aktuell enthält lediglich der Block Autopilot Logik in Form eines vereinfachten Flugreglers. Dieser stabilisiert das Flugzeug in seiner Längs- und Querachse und dient zur Verifikation der Funktionalität des Simulink Modells. Der *Cockpit Indicators* Block enthält verschiedene Scopes und Displays zur Anzeige des Flugzustands, sowie der Steuereingaben des Flugreglers. In den Blöcken Sensor Model und State Estimator können das Sensormodell und die Komponente zur Sensordatenaufbereitung modelliert werden.

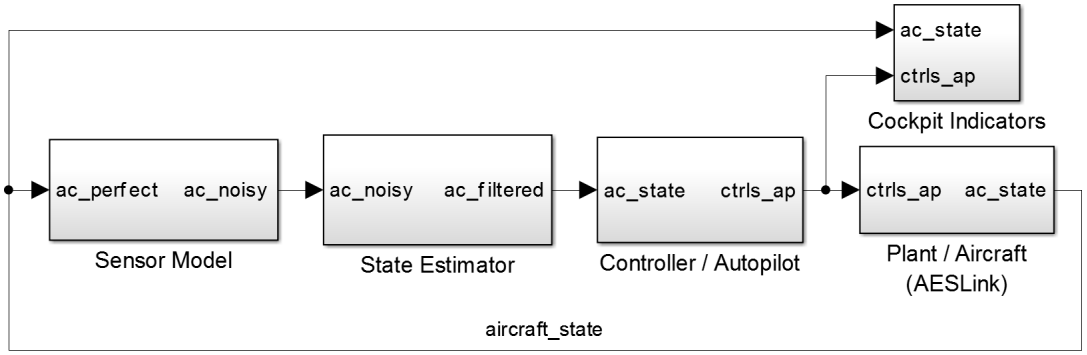

Abbildung 6.10.: Oberste Hierarchieebene des MIL Blockschaltbildes (Simulink)

Plant / Aircraft Block

Der Block *Plant / Aircraft* abstrahiert den Rest des Modells von der Kommunikation mit AESLink. Er ist als *Variant Subsystem* mit vier verschiedenen Varianten aufgebaut. Die jeweilige Variante gibt an, aus welcher Quelle die Piloteneingaben stammen. Die erste Variante verwendet als Piloteneingaben jeweils leere Werte. Mit diesem kann der Flugregler für den autonomen FCU Mode entwickelt werden. Die zweite Variante verwendet einen Simulink Block, der einen USB-GameController (Joystick, RC-Fernbedienung, etc.) auslesen kann. Die dritte Variante verwendet die vom Flugsimulator gesendeten Piloteneingaben. Die vierte Variante verwendet ein in C# entwickeltes Dialogfenster (Abbildung 6.11), welches eine RC-Fernbedienung emuliert. Die Verbindung zum Dialogfenster ist mittels einer Matlab SFunction realisiert.

SFunction zur Kommunikation mit AESLink

Alle vier Varianten verwenden für die Kommunikation mit AESLink eine in C++ entwickelte SFunction. Für den Zugriff auf den Bus wird die cpplinklib im asynchronen Empfangsmodus verwendet. Innerhalb des SFunction Blocks werden zur asynchronen Kommunikation zwischen

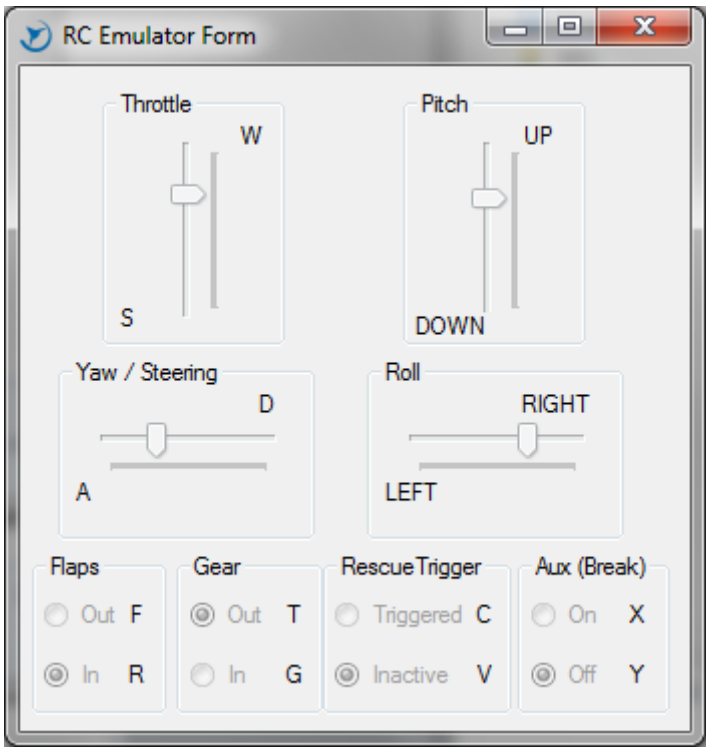

Abbildung 6.11.: Screenshot des RC-Emulator Dialogfensters

Flugregler und Flugsimulator zwei Buffer verwendet. Der *BufferAircraft* speichert den Flugzustand und der *BufferAutopilot* die vom Flugregler berechneten Steuersignale.

Die SFunction wird pro Simulations-/Integrationsschritt von Simulink genau einmal aufgerufen. Als Parameter werden die vom Flugregler berechneten Steuersignale übergeben, welche im BufferAutopilot zwischengespeichert werden. Der Rückgabewert des Aufrufs ist der Flugzustand, der asynchron im BufferAircraft gespeichert wurde.

Wenn innerhalb der cpplinklib eine DataFromAircraft Nachricht eintrifft, wird eine Callback-Funktion innerhalb des SFunction Blocks aufgerufen, welche den empfangenen Flugzustand im BufferAircraft zwischenspeichert. Die im BufferAutopilot gespeicherten Steuersignale werden als Antwort in einer DataFromAutopilot Nachricht über den Bus verschickt.

Dieses Verfahren entkoppelt Flugsimulator von Flugregler, wodurch sich die in 5.6 geforderte Nebenläufigkeit zwischen den beiden Komponenten ergibt.

6.9. Flugphysik in Simulink

Zur Verwendung von Simulink als Regelstrecke wurde ein Blockschaltbild modelliert. Die oberste Hierarchieebene ist in Abbildung 6.12 dargestellt. Anders als beim Blockschaltbild des MIL-Szenarios ist hier der *Autopilot* Block die externe Komponente. Der Block *Aircraft* ist ein Variant Subsystem mit drei Varianten. Die erste Variante ist ein Platzhalter, in welchem eine eigene Flugphysik modelliert werden kann. Er ist des Weiteren dafür vorgesehen, Fehlerszenarien wie etwa Sensorausfälle zu simulieren. Die zweite Variante wurde mit Hilfe des AeroSim Blocksets modelliert und enthält ein mathematisches Modell des *Aerosonde UAV* [Unmanned-Dynamics]. Die dritte Variante enthält eine rudimentäre Flugphysik des AC2030. Diese wurde durch die in 2.4 beschriebenen Bewegungsgleichungen modelliert. Die Derivativa und Ersatzgrößen stammen aus Windkanalversuchen der ersten Version des AC2030 [Neubacher (2008)], bzw. aus [de Castro (2003)]. Für jede der drei Varianten kann ausgewählt werden, ob die Piloteneingaben aus dem in 6.8 beschriebenen Emulator Dialog oder einem USB-GameController stammen.

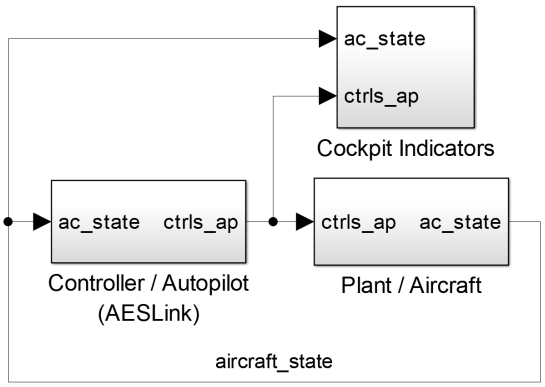

Abbildung 6.12.: Oberste Hierarchieebene des Simulink Modells, welches eine eigene Flugphysik enthält

SFunction zur Kommunikation mit AESLink

Der Autopilot Block abstrahiert das Modell von AESLink. Wie auch im MIL-Szenario wird hier eine in C++ entwickelte SFunction verwendet. Diese SFunction wird verwendet, um den Flugzustand über den Bus zu verteilen und die Steuersignale des Autopiloten zu empfangen.

Im Gegensatz zur SFunction im MIL-Szenario arbeitet die hier entwickelte SFuntion synchron. Beim Aufruf der SFunction wird der Flugzustand versendet und synchron auf eine DataFrom-Autopilot Nachricht gewartet. Die darin enthaltenen Steuersignale des Autopiloten bilden den Rückgabewert der SFunction.

Die in Simulink modellierte Regelstrecke bietet während einer Simulation demnach die gleiche Nebenläufigkeit, wie die Flugsimulatoren.

7. Ergebnisse

In diesem Kapitel wird das entwickelte System evaluiert. Es werden die mit der Infrastruktur möglichen Szenarien, sowie Nachteile gegenüber der in 3 analysierten Konzepte dargestellt. Des Weiteren wird das Zeitverhalten der Kommunikationen innerhalb der Infrastruktur betrachtet.

7.1. Frameraten der Flugsimulatoren

Nachfolgend wird die Eignung der Flugsimulatoren anhand der Geschwindigkeit der Berechnung der Flugphysik analysiert. Die Güte der verwendeten mathematischen Formeln zur Berechnung der Flugphysik wird, wie in 4.2 bereits geschrieben, bei allen Flugsimulatoren als mindestens geeignet angesehen.

Für die Entwicklung und das Testen der Infrastruktur wurde ein PC mit Windows 7 Professional Betriebssystem verwendet. Die Eckdaten der Hardware sind folgende:

- Intel Core i7-3770, acht logische Prozessorkerne, Taktfrequenz von 3.40 GHz
- 8 Gigabyte Arbeitsspeicher
- Intel HD Graphics 4000 Grafikprozessor (integriert in Prozessor)

Der integrierte Grafikprozessor stellt gegenüber dem Prozessor selbst einen deutlichen Flaschenhals dar, was sich in der Geschwindigkeit der Flugsimulatoren widerspiegelt. Nachfolgend sind die durchschnittlichen Frameraten der Flugsimulatoren dargestellt.

AeroSimRC
AeroSimRC stellt geringe Anforderungen an die Hardware. Auf dem verwendeten System läuft der Flugsimulator mit durchschnittlichen 250 FPS, wobei der Wert nicht unter 200 FPS fällt. Dadurch eignet sich AeroSimRC für die Verwendung in den Szenarien MIL, HIL_Dev und HIL_Test.

X-Plane
X-Plane stellt wesentlich höhere Anforderungen an die Hardware des Systems. Selbst auf den niedrigsten Grafikeinstellungen wurden selten mehr als 45 FPS erreicht. Für die Szenarios MIL

und HIL_Test werden jedoch mindestens 50 FPS benötigt. Bevor X-Plane für diese verwendet werden kann, muss für das Entwicklungssystem eine leistungsstärkere Grafikkarte verbaut werden.

Die Anforderung an das HIL_Dev Szenario sind jedoch nicht so stringent gegeben. Daher eignet sich X-Plane auch mit einer Framerate von 45 FPS für die Verwendung in diesem Szenario.

FlightGear
FlightGear benötigt eine noch höhere Grafikleistung als X-Plane. Die durchschnittliche Framerate beträgt bei den niedrigsten Grafikeinstellungen lediglich 22 FPS. Für FlightGear gelten daher die gleichen Aussagen, wie für X-Plane.

7.2. Laufzeitverhalten

Eine Anforderung an die Infrastruktur war, die Latenz der Datenübertragung zwischen den Flugsimulatoren und der FCU, sowie dem in Simulink modellierten Flugregler möglichst gering zu halten. Im Folgenden wird auf das erreichte Laufzeitverhalten eingegangen.

7.2.1. HIL-Szenario

Es soll betrachtet werden, wie viel Zeit ein synchroner Datenaustausch zwischen Flugsimulator und FCU benötigt. Zunächst soll noch einmal der Pfad, den die Daten durchlaufen, dargestellt werden. Abbildung 7.1 zeigt den Pfad der Daten zwischen AeroSimRC und der FCU, bzw. dem HIL-IO-Board.

Die Nachrichten laufen durch mehrere Anwendungen und Komponenten, bis sie beim HIL-IO-Board angekommen sind. Die synchronen Antworten durchlaufen den gleichen Pfad, allerdings in umgekehrter Richtung. Der Regler soll mit einer Frequenz von 50Hz neue Daten erhalten. Somit hat ein Flugsimulator 20ms Zeit um die Daten zu versenden, synchron auf eine Antwort zu warten und das Bild zu rendern. Der letztere Punkt skaliert unmittelbar mit der Grafik- und Prozessorleistung des Host PCs.

Werden Flugsimulator und Proxy auf demselben Host PC ausgeführt, ergibt sich eine *Paketumlaufzeit* der Nachrichten von etwa 3ms. Somit muss auf dem Host PC Hardware eingesetzt werden, die, falls kein Flugregler verbunden ist, knapp 60 FPS erreicht.

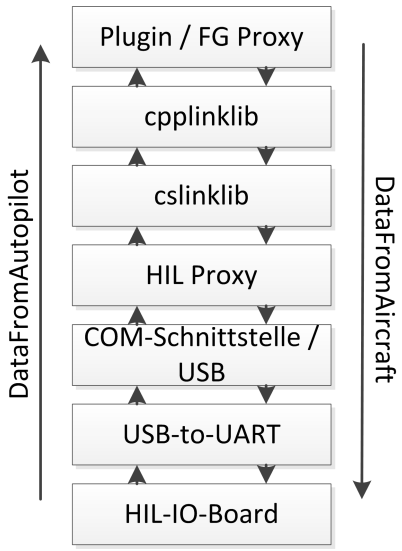

Abbildung 7.1.: Pfad der Nachrichten im HIL-Szenario

7.2.2. MIL-Szenario

Das MIL-Szenario läuft komplett auf einem Host PC ab, daher ergibt sich ein entsprechend kürzerer Pfad als im HIL-Szenario. Abbildung 7.2 zeigt den Pfad der Daten im MIL-Szenario. Die Paketumlaufzeit zwischen Flugsimulator und dem Simulink Modell, bzw. der SFunction beträgt im Durchschnitt unter 1ms. Ein Flugsimulator muss somit eine Framerate von etwa 52 FPS erreichen.

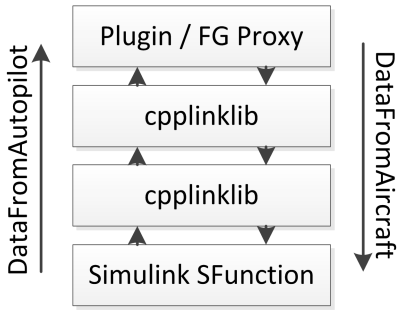

Abbildung 7.2.: Pfad der Nachrichten im MIL-Szenario

7.2.3. Einschätzung

Die Paketumlaufzeit von 3ms im HIL-Szenario machen sich einem Anwender subjektiv nicht bemerkbar. Beide geforderten Szenarien erwecken subjektiv einen guten Eindruck. Wie sehr die Laufzeiten objektiv ins Gewicht fallen, kann vom Autor nicht beurteilt werden. Letztendlich muss dies ein fachkundiger Regelungstechniker, der die Infrastruktur zur Entwicklung von Flugreglern verwendet, beurteilen.

Messungen, wie nah das Lauf- und Totzeitverhalten der realen Umgebung der FCU nachgebildet wurde, sind nicht durchgeführt worden. Diese können erst erfolgen, wenn das Laufzeitverhalten der FCU bekannt und vorhanden ist. Die in 5.6 geforderte Nebenläufigkeit zwischen Regelstrecke und Regler wurde nachgebildet. Die verwendeten Platzhalter können mit künstlicher Totzeit versehen werden, sofern die Simulationsumgebung damit einen höheren Realitätsgrad erreicht. Eine Verringerung der Totzeit kann die Infrastruktur aktuell nicht bieten. Dies kann lediglich mit leistungsstärkerer Hardware erreicht werden.

7.3. Schwächen gegenüber der in Kapitel 3 analysierten Systeme

Für die Infrastruktur wurden Konzepte und Ideen von ähnlichen Systemen übernommen. Hier werden die Schwächen und Nachteile gegenüber diesen Systeme dargestellt.

Einsetzbarkeit in Simulation und Realität
Der markanteste Unterschied zu den analysierten Systemen besteht darin, dass das entwickelte System lediglich eine Simulationsumgebung darstellt. Bei Projekten wie etwa Paparazzi oder ArduPilot können die Werkzeuge häufig auch für reale Flüge eingesetzt werden. Die Infrastruktur wurde hingegen speziell als Simulations- und Testumgebung entwickelt. Je nach Betrachtungsweise kann man dies als Stärke oder als Schwäche ansehen. Zum einen ist die Infrastruktur (zum gegenwärtigen Entwicklungsstand) nicht so flexibel einsetzbar. Zum anderen erfüllt sie jedoch genau die Anforderungen an eine Testumgebung besser als die analysierten Systeme, da sie speziell hierauf ausgelegt ist. Dies lässt sich wie folgt ausdrücken: "*Die Infrastruktur ist keine eierlegende Wollmilchsau, jedoch ist sie für die vorgesehenen Verwendungszwecke besonders gut zu gebrauchen.*"

Statisch durch vorgegebene Nachrichtentypen
Durch die zwei vorgegebenen Nachrichtentypen ist die Infrastruktur relativ statisch. Zum Beispiel ist es nicht ohne weiteres möglich, die Piloteneingaben von einer anderen Quelle als den Flugsimulatoren zu verwenden. Bei Paparazzi können etwa die Piloteneingaben und das Wetter von speziellen Programmen in das Netzwerk eingespeist werden. Bei der realisierten Infrastruktur müssen alle Daten des Flugzustands aus einer Quelle kommen, bzw. in einer zusammengefassten Nachricht verschickt werden.

Beschränktheit auf ein Flugzeug
Eine Simulation ist auf ein einzelnes UAV beschränkt. Paparazzi erlaubt die Simulation von mehreren kooperierenden UAVs zur gleichen Zeit.

Lediglich eine Simulation zur gleichen Zeit
Als weiterer Nachteil ergibt sich, dass im gesamten lokalen Netzwerk lediglich eine Simulation ablaufen darf. Wenn mehrere Nachrichten eines bestimmten Typs bei einem Empfänger eintreffen, dieser jedoch lediglich eine Nachricht erwartet, kann es zu unerwünschtem Verhalten für den Anwender führen. Somit kann etwa im MIL- oder HIL_Dev-Szenario lediglich ein Entwickler zur gleichen Zeit arbeiten.

7.4. Mögliche Szenarien und Verwendbarkeit

Das entwickelte System lässt sich in vielerlei Hinsicht produktiv einsetzen. Die gegebene Aufgabenstellung lautete, eine Infrastruktur für die Entwicklung, Auslegung und das Testen von Flugreglern umzusetzen. Jedoch sind mit dem entwickelten System weit mehr Szenarien denkbar. Dies ist zu großen Teilen der losen Kopplung der Komponenten untereinander geschuldet.

7.4.1. Szenarien und Erweiterbarkeit

Die zwei Hauptkomponenten des Regelkreises sind nicht an spezielle Anwendungen, Systeme oder Orte gebunden. Sowohl Flugregler, als auch ausführbare Flugphysik lassen sich beliebig innerhalb des Systems verteilen. Neben den eingebundenen Flugsimulatoren lassen sich ohne Anpassung der Infrastruktur weitere Anwendungen hinzufügen, die eine Flugphysik berechnen und ausführen können. Dies wurde bereits durchgeführt, da Simulink als Regelstrecke schon jetzt integriert ist. Die Implementierung des SIL-Autopiloten hat gezeigt, dass der Flugregler nicht notwendigerweise an Simulink oder die FCU gebunden ist und sich an einem beliebigen Ort innerhalb der Infrastruktur befinden kann. Jegliche Kombination aus Regelstrecke und Regler sind möglich. Wobei ein Anwender frei in der Wahl des Realitätsgrades der Umgebung ist. So kann er selbst entscheiden, ob, wie und wo Sensormodell und Aktuatormodell eingesetzt werden.

Durch die Verwendung von X-Plane als Flugsimulator ist es nicht nur möglich, einen Flugregler hochwertig auszulegen, sondern sogar die Derivativa des AC2030 ohne reale Flugversuche in gewissem Maße zu bestimmen. Dies wurde erst kürzlich ein weiteres Mal gezeigt. Auf der ILA 2014 hat Airbus eine unkonventionelle Mischung aus Starrflügler und Quadrocopter vorgestellt.

Bei der Entwicklung der Aerodynamik und der Avionik des UAVs wurde maßgeblich auf X-Plane vertraut[1].

Die Infrastruktur lässt sich um beliebige weitere Komponenten zur Analyse und Visualisierung des Simulationsgeschehens erweitern. Hier sei an erster Stelle eine live Anbindung an Google Earth genannt. Besonders bei der Entwicklung einer autonomen Navigation könnte sich eine Visualisierung in Google Earth als nützlich erweisen. Vorstellbar wäre auch die Visualisierung der Daten mittels eines Webservers oder einer Smartphone App.

Durch die Aufzeichnung aller über AESLink verschickten Nachrichten können weit umfangreichere Analyse- und Visualisierungswerkzeuge als die in 6.5 vorgestellten entwickelt werden. Diese könnten Kennzahlen der Flugleistung, wie etwa maximale Steigrate, maximale Geschwindigkeit oder die benötigte Startstrecke bestimmen.

Ein abstraktes Ziel der Infrastruktur ist es, Entwicklungen im AES Kontext zu vereinfachen. Aktuell ist der Zielträger der AC2030. Die Infrastruktur und die einzelnen Komponenten sind jedoch in keiner Weise auf den AC2030 beschränkt. Für die vier Elemente Flugregler, Flugphysik, Sensormodell und Aktuatormodell sind Platzhalter vorhanden, welche an beliebige Träger angepasst werden können.

Das HIL-Szenario kann weit mehr Tests abdecken, als gefordert waren. Mit der Infrastruktur steht eine Umgebung für die gesamte FCU und nicht nur die implementierten Flugregler zur Verfügung. Firmwarekomponenten wie Sensordatenerfassung, Sensordatenaufbereitung, Scheduler, Datenaufzeichnung, Telemetrie, Flugregelung, etc. können mit dem entwickelten System effektiv entwickelt und getestet werden.

7.4.2. Bedienung und Verwendbarkeit

Wird das entwickelte System mit den in 3.1.1 analysierten Open Source Systemen verglichen, so ist das entwickelte System für informatik-fremde Anwender bei weitem einfacher zu bedienen. Beispielsweise wird ein Anwender an keiner Stelle dazu aufgefordert, eine Netzwerkadresse oder einen Port anzugeben. Für den Entwurf eines Flugreglers in Simulink werden keinerlei Programmierkenntnisse benötigt.

Wird die Lösung mit anspruchsvolleren Systemen, wie etwa dem HIL-Testsystem des IFSys Projekts, verglichen, ist das hier entwickelte HIL-Testsystem um Größenordnungen günstiger. Zählt man die Kosten für den Flugsimulator (zum Beispiel X-Plane) hinzu, ergibt sich eine Gesamtsumme von etwa 100,- Euro. Wobei sich die Kosten zu 70,- Euro auf X-Plane, 20,- Euro auf ein STM32F4-Discovery Board, 3,- Euro auf einen USB-to-UART Adapter und die restliches Kosten auf Kabel verteilen.

[1] Golem.de - Airbus Quadcruiser - Quadrocopter mit Pusher für schnellen Vorwärtsflug, http://www.golem.de/print.php?a=106696 [Stand: 30.05.2014]

8. Ausblick

Nachfolgend werden ausstehende, sowie bereits laufende Arbeiten im Zusammenhang mit dieser Thesis genannt. Die grundlegende Aufgabe besteht darin, die Infrastruktur auf Brauchbarkeit zu prüfen. Dies erfolgt implizit dadurch, dass die vorgesehenen Platzhalter nach und nach mit Leben gefüllt werden. Erst darauffolgend können Aussagen darüber erfolgen, ob der erreichte Realitätsgrad im Zeitverhalten, sowie in der Flugphysik den Ansprüchen genügt.

8.1. Flugregler

Derzeit führt Yannic Beyer vom BWB-Team ein Studienprojekt durch. In seiner Arbeit beschäftigt er sich mit der Entwicklung und Auslegung eines Flugreglers. Das übergeordnete Ziel ist dabei, Erfahrung in dem Themengebiet zu sammeln, sowie einen generischen Prototyp zu entwerfen. Teile der in der vorliegenden Arbeit entwickelten Infrastruktur werden von ihm bereits verwendet. Am Ende seiner Arbeit kann er Aussagen darüber treffen, ob und wie gut sich die Infrastruktur im MIL-Szenario eignet.

8.2. Flugmodell in X-Plane

Im nächsten Semester wird von Christian Hornemann (ebenfalls BWB-Team) eine weitere Studienarbeit im Kontext der vorliegenden Thesis ausgearbeitet. Die Studienarbeit hat zum Ziel, ein Flugmodell des AC2030 in PlaneMaker für X-Plane zu entwerfen. Hierdurch kann die Flugphysik des AC2030 erfasst werden. Weiterhin kann das Flugmodell einem Piloten zum Training dienen.

8.3. Reale Flüge und Ground Control Station

Es ist zu prüfen, ob sich die Infrastruktur, bzw. Komponenten dieser, für reale Flüge eignen. Derzeit lässt sich nicht abschätzen, welche Teile ohne bzw. mit geringen Änderungen über-

nommen werden können. Besonders im Vordergrund steht dabei das verwendete Kommunikationsprotokoll.

In realen Flügen ist eine Ground Control Station unerlässlich. Sie übernimmt Aufgaben der Telemetrie und des Telecontrols. Das AESLink Protokoll wurde im ersten Schritt nicht für diese Zwecke konzipiert. Es ist daher zu prüfen, ob AESLink für diese Zwecke erweitert wird oder auf bereits vorhandene Protokoll wie UAVTalk und MAVLink zurückgegriffen werden sollte. Für die Verwendung von MAVLink würde beispielsweise sprechen, dass mit der freien GCS *qGroundControl*[1] bereits eine mächtige Ground Control Station zur Verfügung steht. UAVTalk ist hingegen dynamischer und könnte einfacher erweitert werden.

Die Visualisierung eines realen Fluges kann ohne Anpassung erfolgen, solange die Telemetriedaten in DataFromAircraft Nachrichten umgewandelt und über den Bus verteilt werden.

8.4. HIL-Teststand inklusive Iron Bird

Derzeit ist bereits ein HIL-Teststand inklusive eines Iron Birds im Aufbau. Er wird im Rahmen eines Semesterprojekts von vier Mechatronik Studenten der HAW Hamburg entwickelt. Als Hardware enthält er das HIL-IO-Board, das FCU Board, ein Modellflugzeug, sowie eine Platine (CSA-Platzhalter) zur Ansteuerung der Servos des Modellflugzeugs. Der Teststand soll als Grundlage für Entwicklungen im Zusammenhang der FCU, sowie zu Präsentations- und Demonstrationszwecken dienen.

8.5. Ermitteln der Derivativa des AC2030

Die Arbeit von Christian Hornemann kann für erste Simulationen und Tests genutzt werden, ohne dass der reale AC2030 in die Luft steigen muss. Jedoch ist X-Plane selbst ebenfalls nur eine Simulation mit begrenztem Realitätsgrad. Bessere Derivativa werden durch reale Flugversuche ermittelt. Diese müssen weiterhin stattfinden, um den Realitätsgrad der Simulation zu steigern. Aus diesem Grund bleiben reale Flugversuche in der Zukunft nicht aus. Bisher haben sich im BWB-Team unter anderem Danke [Danke (2005)], Neubacher [Neubacher (2008)] und Shtrakbayn [Shtrakbayn (2013)] mit den Derivativa des AC2030 beschäftigt.

Aus den ermittelten Derivativa kann eine Flugphysik in Simulink modelliert werden. Diese kann flexibler als die Flugsimulatoren auf eigene Bedürfnisse angepasst werden. Neben dem Forschungsaspekt ist dadurch auch ein Lehraspekt gegeben. Das Modellieren einer komplexen

[1] qGroundControl - Ground Control Station, http://www.qgroundcontrol.org/ [Stand: 30.05.2014]

Flugphysik ist nicht trivial und kann Stoff für mehrere Bachelor- und Masterarbeiten für Studierende des Studiengangs Flugzeugbau an der HAW Hamburg bieten.

8.6. Entwicklung der FCU

Den AES Projektteilnehmern, sowie kommenden Bachelor- und Masterstudierenden obliegt die Aufgabe, die FCU zu entwickeln. Während dieser Arbeiten wird sich entscheiden, wie gut sich die Infrastruktur im HIL-Szenario tatsächlich eignet.

Parallel zu der vorliegenden Arbeit haben René Büscher (*Ein Safety-Konzept für Airborne Embedded Systems*) [Büscher (2014)] und Alexander Rohrer (*Softwarearchitektur für Airborne Embedded Systems*) [Rohrer (2014)] ihre Bachelorthesen im Kontext des AES Projekts angefertigt. Kürzlich hat auch Kaoutarre Bedda mit ihrer Bachelorthesis begonnen. Sie beschäftigt sich mit der Entwicklung von Algorithmen und Filtern zur Sensordatenaufbereitung.

8.7. VR Brille

Im Zuge der Betrachtungen von Sebastian Böhle könnten sich Anwendungsbereiche für eine VR Brille ergeben. FlightGear kann das gerenderte Bild durch seine Screen Relay Funktion ausgeben. Der Entwickler von X-Plane hat für die kommende Version die native Unterstützung der Oculus Rift angekündigt. Prepar3D unterstützt die Oculus Rift bereits zum aktuellen Zeitpunkt nativ. Die Infrastruktur könnte um den Anwendungsbereich der Verwendung von VR Brillen erweitert werden.

8.8. SIL-Szenario

Projekte wie IFSys, Paparazzi, Ardupilot und Openpilot verwenden SIL-Simulationen zum Testen der Firmware. In der vorliegenden Arbeit wurde lediglich die Komponente Flugregler für ein SIL-Szenario implementiert (Abschnitt 6.6). Eine umfangreiche SIL-Simulation ermöglicht es, die Firmware auch ohne vorhandene Hardware testen und entwickeln zu können.

Falls in Zukunft der Wunsch besteht, bzw. die Anforderung gegeben ist, umfangreichere SIL-Tests durchzuführen, kann die Infrastruktur ohne Anpassung um diesen Anwendungsfall erweitert werden.

8.9. Multi UAV Simulationen

In den Schwächen der Infrastruktur wurde bereits erwähnt, dass AESLink aktuell nicht für die Verwendung mehrerer UAVs zur gleichen Zeit ausgelegt ist. Für ein Zukunftsszenario, in dem mehrere UAVs für Zwecke des Katastrophenschutzes simuliert werden sollen, muss die Infrastruktur angepasst werden.

8.10. Simulationen mit jsbsim

Die Infrastruktur könnte um den Kommandozeilen-Flugsimulator jsbsim erweitert werden. Dieser ist durch das Fehlen einer visuellen Darstellung des Flugmodells nicht notwendigerweise an die Realzeit gebunden. Die Realzeit und die simulierte Zeit können unterschiedlich sein. Dies kann besonders bei Regressionstests von Vorteil sein, da die benötigte Realzeit direkt an die Computerleistung gebunden ist. Sorton und Hammaker gehen hierbei von Stunden, bis sogar Tagen an Zeitersparnis aus [Sorton und Hammaker (2005)].

8.11. Erstellen einer Benutzerdokumentation

Die kommende Aufgabe für den Autor besteht darin, eine Benutzerdokumentation zu erstellen. Diese soll die Funktionen, Möglichkeiten und Szenarien der Infrastruktur detaillierter als hier dargestellt beschreiben.

8.12. Steuereingaben bei FlightGear

In Abschnitt 6.2.3 wurde beschrieben, dass es bei FlightGear im Zusammenhang mit Steuereingaben zu Problemen kommen kann. Diese sollten behoben werden, um auch die FCU-Modes *manuell* und *unterstützend* effektiv mit FlightGear behandeln zu können.

9. Fazit

An diesem Punkt möchte ich die vorliegende Arbeit mit einem persönlichen Fazit abschließen.

Hintergrund der Arbeit war es, Aspekte aus Flugregelung und Simulation zu vereinen: Es sollte eine Infrastruktur geschaffen werden, die das Entwickeln und Testen von Flugreglern durch Simulation ermöglicht. Zu Beginn stand eine vage Skizze dessen im Raum, wie solch eine Infrastruktur aufgebaut sein könnte. Es wurden ähnliche Systeme recherchiert und auf Basis dieser ein Konzept für eine eigene Infrastruktur entworfen. Dieses Konzept wurde realisiert, sowie subjektiv als produktiv einsetzbar eingeschätzt.

Die Basis der Infrastruktur bilden verschiedene Flugsimulatoren. Ich habe einen Leitfaden gegeben, bei welchen Anforderungen welcher Flugsimulator eingesetzt werden sollte. Die Flugregler können in Simulink entworfen werden. Die Infrastruktur ermöglicht die Entwicklung der FCU in einer risikofreien Simulationsumgebung. Das entwickelte System bildet somit eine solide Grundlage zum Entwickeln und Testen von Flugreglern, sowie weiterer Firmwarekomponenten.

Ich habe mich bereits in der Vergangenheit mit der Entwicklung von Testsystemen beschäftigt. Weiterhin bin ich Anhänger einer *test-first* Mentalität. Daher war es für mich ein logisches Vorgehen, zuerst eine Testinfrastruktur zu schaffen, bevor mit der effektiven Entwicklung der FCU begonnen werden kann.

Die größte Schwierigkeit für mich bestand darin, ein geeignetes Laufzeitkonzept aufzustellen. Hingegen hat mir die Arbeit mit Simulink und Matlab sehr viel Spaß gemacht, da ich hierbei eine Menge dazulernen konnte. Auch das Gebiet der Flugmechanik fasziniert mich. In meinem Masterstudium möchte ich weiter in dieses eintauchen. Vorstellbar wäre eine adaptive Flugsteuerung zu entwickeln, welche bei Ausfällen der Aktorik dynamisch die Steuerflächenzuordnung rekonfiguriert. Für diese Aufgabenstellung kann ich als Basis die hier entwickelte Infrastruktur verwenden.

Abschließend hoffe ich, dass sich das entwickelte System in Zukunft so produktiv einsetzen lässt, wie ich es persönlich einschätze.

Literaturverzeichnis

[Abdunabi 2006] ABDUNABI, Tarek: *Modelling and Autonomous Flight Simulation of a Small Unmanned Aerial Vehicle*, The University of Sheffield, Diplomarbeit, August 2006

[Adiprawita u. a. 2007] ADIPRAWITA, Widyawardana ; AHMAD, Adang S. ; SEMBIRING, Jaka: Hardware in the Loop Simulation for Simple Low Cost Autonomous UAV (Unmanned Aerial Vehicle) Autopilot System Research and Development. In: *Proceedings of the International Conference on Electrical Engineering and Informatics* Institut Teknologi Bandung, Indonesia (Veranst.), Juni 2007, S. 218–221

[von Ahlen 2012] AHLEN, Tim von: Modellierung eines Box-Wing-Flugzeuges mit PlaneMaker für den Flugsimulator X-Plane / Hochschule für Angewandte Wissenschaften Hamburg. Mai 2012. – Projekt

[AirbusFCL] AIRBUSFCL: *Airbus Flight Control Laws*. Online. – URL http://www.airbusdriver.net/airbus_fltlaws.htm. – [Stand: 30.05.2014]

[Albert 2005] ALBERT, Nicolas: *Certification du code embarqu d'un micro-drone*, University of Toulouse, Diplomarbeit, 2005

[Backes 2013] BACKES, Tim: Integration des AC20.30 in einem Modellflugsimulator / Hochschule für Angewandte Wissenschaften Hamburg. April 2013. – Projekt im Flugzeugbau

[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

[Böhle 2014] BÖHLE, Sebastian: First Person View in Unmanned Aircraft Systems / Hochschule für Angewandte Wissenschaften Hamburg. März 2014. – Studienarbeit

[Brisset u. a. 2006] BRISSET, Pascal ; DROUIN, Antoine ; GORRAZ, Michel ; HUARD, Pierre-Selim ; TYLER, Jeremy: The Paparazzi Solution. In: *Micro Air Vehicle competition 2006*, 2006

[Brockhaus u. a. 2011] BROCKHAUS, Rudolf ; ALLES, Wolfgang ; LUCKNER, Robert: *Flugregelung*. 3. Springer, 2011

[Büscher 2014] BÜSCHER, René: *Ein Safety-Konzept für Airborne Embedded Systems*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorthesis, Juni 2014

[de Castro 2003] CASTRO, Helena V. de: *Flying and Handling Qualities of a Fly-By-Wire Blended Wing Body Civil Transport Aircraft*, Cranfield University, Dissertation, Dezember 2003

[Chandler 2013] CHANDLER, Peter: *A350 XWB First Flight*. Video (www.youtube.com). Juni 2013. – URL https://www.youtube.com/watch?v=DC9qmo7roWc. – [Stand: 30.05.2014]

[Chao u. a. 2013] CHAO, YUN ; XIAO-MIN, LI ; ZONG-GUI, ZHENG: DESIGN OF UAV SIMULATOR BASED ON MAN-IN-LOOP SIMULATION PLATFORM. In: *International Journal of Science, Environment and Technology* 2 (2013), Juni, Nr. 3, S. 449–456

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

[Díaz-Tula u. a. 2010] DÍAZ-TULA, Antonio ; CASTAÑEDA-GARAY, Miguel ; BELMONTE-FERNÁNDEZ Óscar: Reading the Visible Frame Buffer to a Pixel Buffer Object. In: *GraVisMa - 2nd International Conferences on Computer Graphics, Computer Vision and Mathematics - Workshop on Computer Graphics, Computer Vision and Mathematics* Bd. 2, 2010

[Deters u. a. 2006] DETERS, Robert W. ; DIMOCK, Glen A. ; SELIG, Michael S.: Icing Encounter Flight Simulator. In: *Journal of Aircraft* 43 (2006), September - Oktober, Nr. 5

[Föllinger u. a. 2013] FÖLLINGER, Otto ; KONIGORSKI, Ulrich ; LOHMANN, Boris ; ROPPENECKER, Günter ; TRÄCHTLER, Ansgar: *Regelungstechnik: Einführung in die Methoden und ihre Anwendung*. 11. VDE VERLAG GmbH, 2013

[Gomez 2001] GOMEZ, Martin: Hardware-in-the-loop simulation. In: *EE Times-India* (2001), Dezember

[Gregory 2009] GREGORY, Jason ; LANDER, Jeff (Hrsg.): *Game Engine Architecture*. Bd. 1. Taylor & Francis Ltd, April 2009

[Hamann und Hoffmann] HAMANN, Alexander ; HOFFMANN, Arndt: *Systemübersicht ALEXIS*

[Hamann und Hoffmann 2011] HAMANN, Alexander ; HOFFMANN, Arndt: Friedliche Drohne - Flugversuchsträger mit Embedded-PC-Steuerung. In: *Elektronik* 21 (2011), Oktober, S. 35–39

[Hasberg 2013] HASBERG, Hagen: *Vergleich von Open Source Autopiloten Systemen*. Oktober 2013. – Interne Veröffentlichung

[Kirner 2009] KIRNER, Raimund: *Testen von Embedded Systems - Hardware in the Loop (HIL) Testing*. 2009. – Vorlesungsskript WS08/09

[Lange und Bogdan 2012] LANGE, Walter ; BOGDAN, Martin: *Entwurf und Synthese von Eingebetteten Systemen: Ein Lehrbuch*. 1. Oldenbourg Wissenschaftsverlag, Februar 2012

[Liebeck 2004] LIEBECK, R. H.: Design of the Blended Wing Body Subsonic Transport. In: *Journal of Aircraft* 41 (2004), Januar - Februar, Nr. 1

[Lizarraga u. a. 2013] LIZARRAGA, M. ; ELKAIM, G.H. ; CURRY, R.: SLUGS UAV: A flexible and versatile hardware/software platform for guidance navigation and control research. In: *American Control Conference (ACC), 2013*, Juni 2013, S. 674–679. – ISSN 0743-1619

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

[Meyers 2013] MEYERS, Austin: *X-Plane 10 Desktop Manual*. Laminar Research (Veranst.), 2013

[Mueller 2007] MUELLER, Eric: Hardware-in-the-loop Simulation Design for Evaluation of Unmanned Aerial Vehicle Control Systems. In: *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, August 2007, S. –. – URL http://dx.doi.org/10.2514/6.2007-6569

[Neubacher 2008] NEUBACHER, Christoph: Flight Dynamic Investigations of a Blended Wing Body Aircraft / Hochschule für Angewandte Wissenschaften Hamburg. Oktober 2008. – Project Thesis

[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

[Park u. a. 2008] PARK, Hansol ; KIM, Doo-Hyun ; CHANG, Chun-Hyon ; KIM, Jung-Guk: Experimental Evaluation of Unmanned Aerial Vehicle System Software Based on the TMO Model. In: *Journal of Computing Science and Engineering* 2 (2008), Dezember, Nr. 4, S. 357–374. – URL http://jcse.kiise.org/PublishedPaper/year_abstract.asp?idx=30

[PopularScienceMonthly 1930] POPULARSCIENCEMONTHLY: Now - The Automatic Pilot. In: *Popular Science Monthly* 116 (1930), Februar, Nr. 2, S. 22

[Rauw 1993] RAUW, M. O.: *A Simulink environment for Flight Dynamics and Control analysis - application of the DHC-2 Beaver*, Delft University of Technology, Diplomarbeit, Juli 1993

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

[Ribeiro und Oliveira 2010] RIBEIRO, L.R. ; OLIVEIRA, N. M F.: UAV autopilot controllers test platform using Matlab/Simulink and X-Plane. In: *Frontiers in Education Conference (FIE), 2010 IEEE*, Oktober 2010, S. S2H–1–S2H–6. – ISSN 0190-5848

[Rohrer 2014] ROHRER, Alexander: *Softwarearchitektur für Airborne Embedded Systems*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorthesis, Mai 2014

[dos Santos u. a. 2011] SANTOS, Sérgio Ronaldo B. dos ; JUNIOR, Sidney Nascimento G. ; JÚNIOR, Cairo Lúcio N. ; BITTAR, Adriano ; OLIVEIRA, Neusa Maria F. de: Modeling of a Hardware-in-the-Loop Simulator for UAV Autopilot Controllers. In: *Proceedings of COBEM 2011, 21st Brazilian Congress of Mechanical Engineering*, 2011

[Schulz 2010] SCHULZ, Gerd: *Regelungstechnik 1: Lineare und Nichtlineare Regelung, Rechnergestützter Reglerentwurf*. überarbeitete Auflage. Oldenbourg Wissenschaftsverlag, 2010

[Shtrakbayn 2013] SHTRAKBAYN, Andrey: Analyse der statischen Stabilität und der Steuerbarkeit des AC 20.30 / Hochschule für Angewandte Wissenschaften Hamburg. März 2013. – Projekt

[Sorton und Hammaker 2005] SORTON, Eric ; HAMMAKER, Sonny: Simulated Flight Testing of an Autonomous Unmanned Aerial Vehicle Using FlightGear. In: *Infotech@Aerospace Conferences*. American Institute of Aeronautics and Astronautics, September 2005, S. –. – URL http://dx.doi.org/10.2514/6.2005-7083

[Unbehauen 2008] UNBEHAUEN, Heinz: *Regelungstechnik I: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. 15., überarb. und erw. Auflage. Vieweg+Teubner Verlag, 2008

[Unmanned-Dynamics] UNMANNED-DYNAMICS: *AeroSim Users Guide 1.2*

[VELA] VELA: *Project VELA*. – URL http://www.dlr.de/as/desktopdefault.aspx/tabid-188/379_read-636/. – [Stand: 30.05.2014]

[Weiss 2012] WEISS, Stephan M.: *Vision Based Navigation for Micro Helicopters*, Eidgenössische Technische Hochschule Zürich, Dissertation, 2012

[Wendel 2011] WENDEL, Jan: *Integrierte Navigationssysteme: Sensordatenfusion, GPS und Inertiale Navigation*. überarbeitete Auflage. Oldenbourg Wissenschaftsverlag, Februar 2011

[Zhang und Li 2013] ZHANG, Huamin ; LI, Li: A Flight Evaluating System using Flight Gear. In: *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering* Bd. 2, Atlantis Press, Januar 2013

A. Einheiten der DataFromAircraft Nachricht

Das nachfolgende Listing zeigt die Datentypen und normierten Einheiten der DataFromAircraft Nachricht.

```
1  struct AESLink_MessageDataFromAircraft
   {
3      //------------------------
       // Time
5      //------------------------
       float time_integrationTimeStep; // integration time step in seconds.
           The time between 'now' and the last calculation of the physical
           model (the last sample)
7
       //------------------------
9      // Controls
       //------------------------
11     float ctrls[RC_NUMBER_OF_CHANNELS]; // [-1, 1] Pilot input (currently 8
           channels)

13     //------------------------
       // Attitude
15     //------------------------
       float attitude_roll;  // rad [-PI,   PI] Positive roll when right wing
           down
17     float attitude_pitch; // rad [-PI,   PI] Positive pitch when nose up
       float attitude_yaw;   // rad [  0, 2*PI] 0 = North, PI/2 = East, PI =
           South, 1.5*PI = West
19
       //------------------------
21     // Position
       //------------------------
23     float position_X; // m Model absolute position in scenario, relative to
           the starting point. Positive when position is front
       float position_Y; // m Model absolute position in scenario, relative to
           the starting point. Positive when position is right
25     float position_Z; // m Model absolute position in scenario, relative to
           the starting point. Positive when position is up
```

```
27      float position_latitude;  // deg Model absolute position in world/
            navigation coordinates (relative to equator)
        float position_longitude; // deg Model absolute position in world/
            navigation coordinates (relative to prime meridian)
29      float position_altitude;  // m  Model absolute position in world/
            navigation coordinates (relative to mean sea level)

31      float position_agl; // m Model absolute position over ground

33      //----------------------
        // Angular Velocity
35      //----------------------
        float angVel_P; // rad/s Model angular velocity in body frame
            coordinates. Positive when roll  increases
37      float angVel_Q; // rad/s Model angular velocity in body frame
            coordinates. Positive when pitch increases
        float angVel_R; // rad/s Model angular velocity in body frame
            coordinates. Positive when yaw   increases
39
        //----------------------
41      // Air Data
        //----------------------
43      float angle_of_attack;    // deg [-180, 180] Positive when wind attacks
            from bottom
        float angle_of_sideslip; // deg [-180, 180] Positive when wind attacks
            from right
45
        //----------------------
47      // Velocities
        //----------------------
49      float velocity_airspeed_indicated; // m/s Speed relative to surrounding
             air
        float velocity_airspeed_true;      // m/s Speed relative to surrounding
             air
51
        float velocity_groundspeed;        // m/s Speed over ground
53      float velocity_climb_rate;         // m/s Positive when position_Z
            increases
};
```

B. Screenshots des Matlab Plotters

Nachfolgend sind Screenshots des Matlab Plotters (Abschnitt 6.5.1) dargestellt.

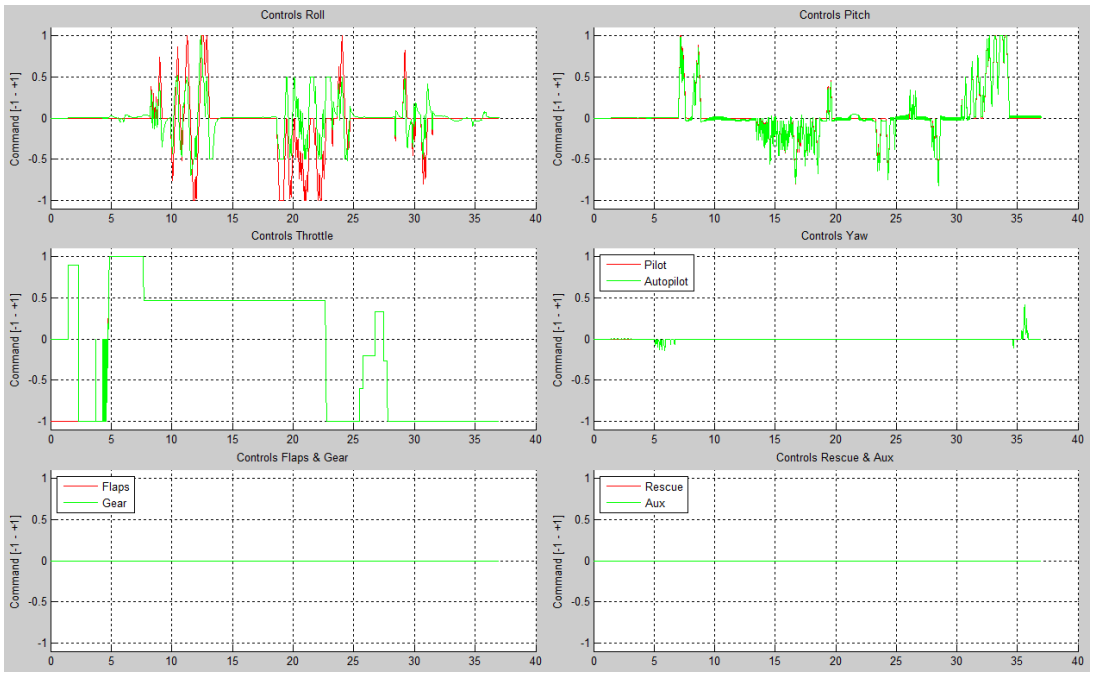

Abbildung B.1.: Diagramme der Eingeben des Piloten (rot) und des Autopiloten (grün). Bei den Kanälen Flaps, Gear, Rescue System und Aux werden jeweils nur die Eingaben des Autopiloten angezeigt

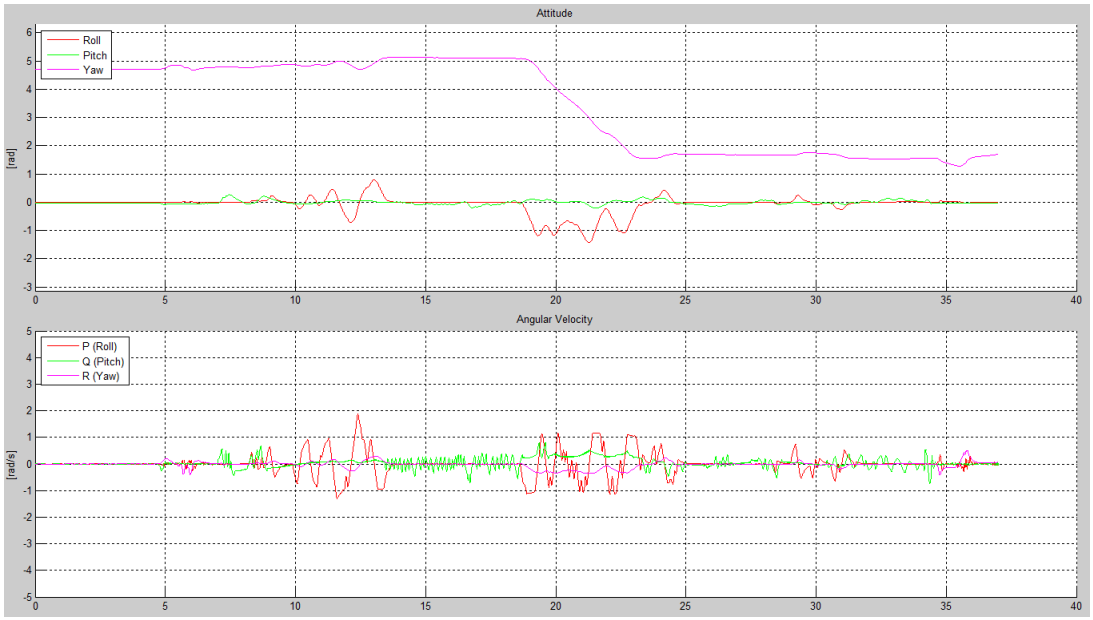

Abbildung B.2.: Diagramme der Fluglage und der Drehraten (beide im körperfesten Koordinatensystem)

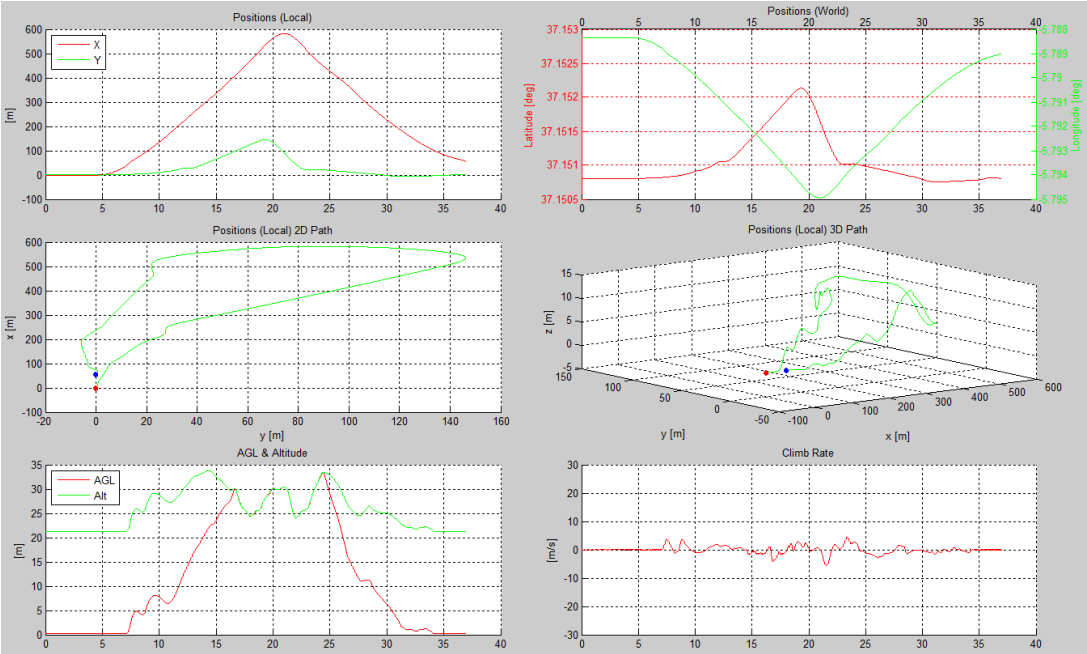

Abbildung B.3.: Diagramme der Positionen. Die mittleren beiden Diagramme sind 2D (X-Y) und 3D (X-Y-Z) Plots, ohne Zeitachsen. Der rote Punkt zeigt die Startposition, der blaue die aktuelle Position des Flugzeugs

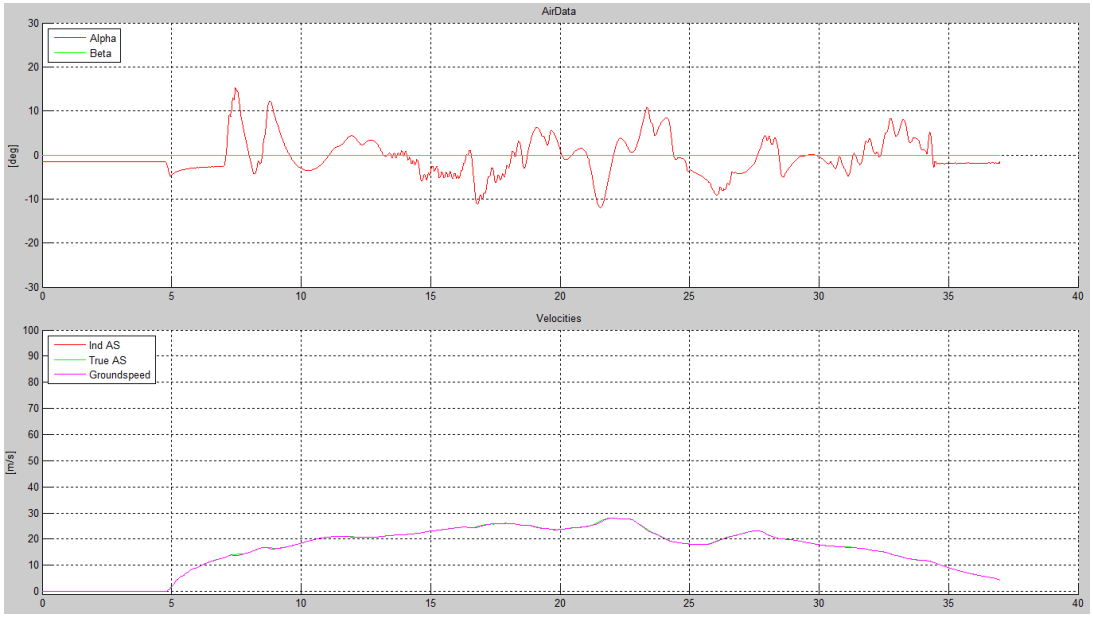

Abbildung B.4.: Diagramme zur Anzeige von Luftdaten (Anstell- und Schiebewinkel) und diversen Geschwindigkeiten

C. Screenshots der Flugsimulatoren

Nachfolgend sind Screenshots aus den Flugsimulatoren dargestellt.

X-Plane

Abbildung C.1.: Screenshot aus X-Plane mit einer Piaggio P.180 Avanti

Oben links im Bild sind interne Werte des Simulators dargestellt. Oben rechts ist das Overlay des Plugins zu sehen. Weiterhin ist eine dreidimensional Wand dargestellt, welche die geflogene Trajektorie repräsentiert.

FlightGear

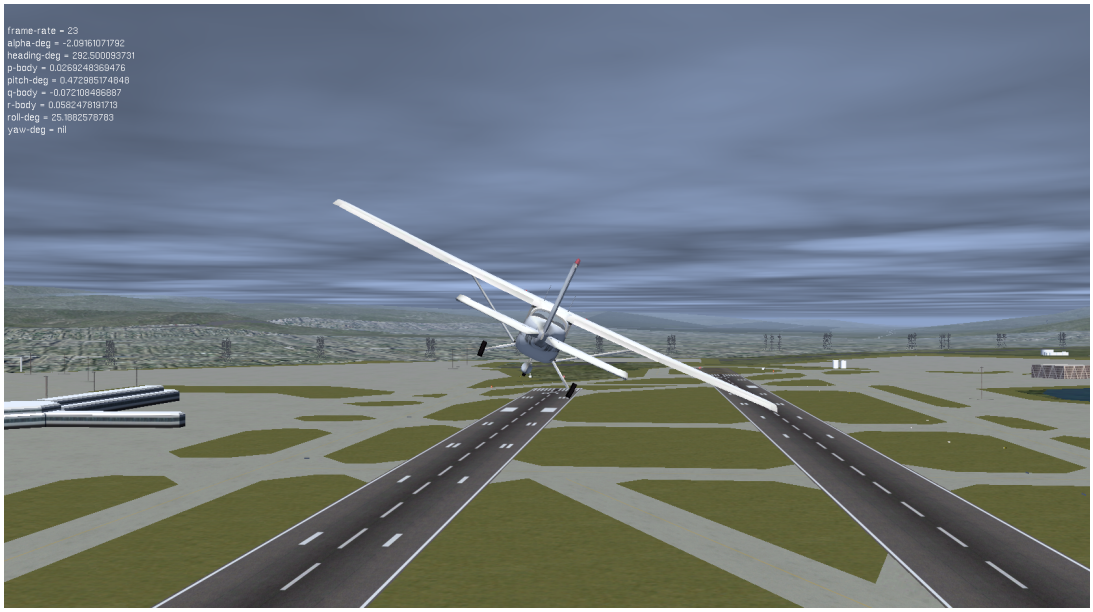

Abbildung C.2.: Screenshot aus FlightGear mit einer Cessna 172P

Oben links im Bild sind interne Werte aus FlightGear dargestellt.

AeroSimRC

Abbildung C.3.: Screenshot aus AeroSimRC mit einem einfachen Trainermodell

Im unteren Bildabschnitt werden verschiedene analoge Cockpit-Instrumente angezeigt. Am linken Bildrand ist das Overlay des Plugins dargestellt. Die Buttons mit den Abkürzungen können geklickt werden, um den Flugzustand zu verändern. Das Flugzeug wird dabei jeweils in 100 Metern Höhe über der Startbahn positioniert.

Die zwei Buchstaben vor dem Unterstrich betreffen die Fluglage und haben folgende Bedeutung:

- **RA, Random Attitude**: Zufällige Fluglage.
- **TA, Trimmed Attitude**: Getrimmte/stabilisierte Fluglage.
- **KA, Keep Attitude**: Fluglage beibehalten.

Die zwei Buchstaben nach dem Unterstrich betreffen die Kräfte und haben folgende Bedeutung:

- **KF, Keep Forces**: Kräfte beibehalten.

- **CF, Correct Forces**: Korrekte Kräfte. Die Kraft greift nur am Heck an und beschleunigt das Flugzeug lediglich nach vorne. Ein Schieben zur Seite kommt nur durch die Schwerkraft zustande.
- **RF, Random Forces**: Zufällige Kräfte.
- **NF, No Forces**: Keine Kräfte. Das Flugzeug fällt aufgrund der Schwerkraft herunter.

D. Verwendete Werkzeuge

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

- **X-Plane**: Version 10.25 (Demo)
- **FlightGear**: Version 3.00
- **AeroSimRC**: Version 4.1
- **Matlab**/**Simulink**: Version R2013b (x64)

  Entwicklungen: MIL Flugregler, Simulink Flugphysik, Plotter
- **CooCox CoIDE**: Version 1.75

  Entwicklungen (C/C++): Firmware der FCU und des HIL-IO-Boards
- **Visual Studio**: Visual Studio 10 Premium

  Entwicklungen (C/C++): Plugins für AeroSimRC und X-Plane, SFunctions der Simulink Modelle, cpplinklib
- **SharpDevelop**: Version 4.31

  Entwicklungen (C#): FlightGear Proxy, HIL Proxy, Gauges Anwendung, Logger, Replay, ReplayCLI, SIL Autopilot, RC-Emulator Dialog, cslinklib

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

E. Inhalte der Begleit-CD

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

- **Thesis**: PDF-Datei des vorliegenden Dokuments.
- **Literatur**: Die referenzierte Literatur als PDF-Dateien, bzw. die URLs der Bücher (Webseite des Verlags).
- **Realisierung**: Batch Skript (siehe unten).

  AeroSimRC: Ausführbare Demoversion von AeroSimRC, sowie der Source Code des AeroSimRC Plugins.

  AESLink: Source Code der cslinklib und der cpplinklib.

  CSharp: Source Code der in C# entwickelten Anwendungen.

  Exe: Executables der C# Anwendungen.

  Avionics: Source Code der FCU und des HIL-IO-Boards.

  XPlane: Source Code des X-Plane Plugins.

  Simuink: Simulink Modelle, Simulink Blockset/Library, Matlab Skripte des Plotters.

  Demo: Demonstrative Simulationsdurchläufe als aufgezeichnete Dateien.

Es werden vorwiegend relative Pfade verwendet. Jedoch müssen bei den Matlab Skripten, sowie den Simulink SFunctions absolute Pfade angegeben werden. Hierfür liegt ein simples Batch Skript bei, welches das Verzeichnis *Realisierung* als Basisverzeichnis für die absoluten Pfade definiert.

F. Begriffe aus der Informatik

Nachfolgend werden einige Begriffe aus der Informatik erläutert, die im Zusammenhang dieser Arbeit verwendet wurden.

Netzwerkprogrammierung

- **User Datagram Protocol (UDP)**: Verbindungsloses Netzwerkprotokoll zur Kommunikation zwischen Netzwerkteilnehmern.
- **Transmission Control Protocol (TCP)**: Verbindungsorientiertes Netzwerkprotokoll.
- **Socket**: Programmierkonzept, welches die Netzwerkprogrammierung durch Abstraktion betriebssystemabhängiger Funktionen vereinfacht.
- **Adresse und Port**: Beschreiben einen Socket. Mit der *Adresse* kann ein Computers im Netzwerk erreicht werden. Durch den *Port* kann ein spezifischer Prozess auf dem Computer angesprochen werden.
- **Multicast** / **Software-Bus**: Netzwerkkommunikation, an der mehrere Computer zur gleichen Zeit teilnehmen können. Eine Nachricht wird nicht an einen spezifischen Empfänger, sondern stattdessen werden die Nachrichten an alle Teilnehmer gesendet. Diese entscheiden selbst, ob die Nachricht an einen Prozess weitergeleitet oder verworfen wird.
- **Paketumlaufzeit**: Zeit, die ein Datenpaket benötigt, um von der Quelle zum Ziel und zurück zu reisen.

Threadprogrammierung

- **Thread**: Ein *Thread* ist ein Programmkonstrukt, welches nebenläufige Ausführungen innerhalb eines Prozesses erlaubt.
- **schlafen** / **sleep**: Ein Thread kann auf ein Ereignis, zum Beispiel den Ablauf einer Zeitspanne, warten. Dieses Warten wird *schlafen* (engl. *sleep*) genannt.
- **Interprozesskommunikation**: Informationsaustausch zwischen zwei oder mehreren Prozessen.

- **Interthreadkommunikation**: Informationsaustausch zwischen zwei oder mehreren Threads.

Schnittstellen

- **Serial Peripheral Interface (SPI)**: Serieller Datenbus zur Verschaltung von Bauteilen auf einer Hardwareplatine.
- **Universal Asynchronous Receiver Transmitter (UART)**: Punkt-zu-Punkt Verbindung von zwei Bauteilen auf einer, oder zwischen zwei Platinen.
- **RS232**: Wie UART, jedoch höhere Spannungen.
- **Pulsweitenmodulation (PWM)**: Modulationsart, bei der die Pulslänge dem Wert des Signals entspricht.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 10. Juni 2014
Ort, Datum

Unterschrift