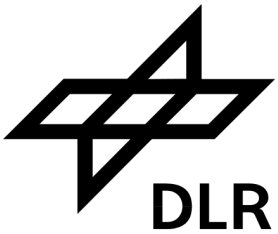

Entwicklung eines Hardware-in-the-Loop-Testsystems zur Validierung des IPS-Navigationssystems

Bachelorarbeit

für die Prüfung zum
Bachelor of Engineering

des Studienganges Informationstechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Jonathan Jost

16.09.2019

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	6056322,TINF16ITIN
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt, Berlin-Adlershof
Betreuer der Ausbildungsfirma	Dr. Maximilian Buder
Gutachter der Dualen Hochschule	Prof. Dr. Heinz Juergen Mueller

Zusammenfassung

Das Integrated Positioning System (IPS) ist ein Gerät zur Selbstlokalisierung. Es ist dazu in der Lage die Eigenposition relativ zu einem Startpunkt zu bestimmen, ohne dazu externe Annahmen über die Umgebung zu besitzen. Das IPS erhält diese Fähigkeit durch Sensorfusion. Es besitzt eine Inertialmesseinheit, die Beschleunigung und rotatorische Bewegung misst und eine Stereokamera, welche Bildpunkte verfolgt und so Bewegungen nachvollziehen kann. Kombiniert man die Informationen der beiden Sensoren heben sich die Nachteile beider gegenseitig auf. So ist es möglich millimetergenaue Trajektorien zu erzeugen, entlang derer sich das Gerät bewegt hat.
Das IPS ist Gegenstand kontinuierlicher Entwicklung. Die Bestrebung des Deutschen Zentrums für Luft- und Raumfahrt ist es das Gerät in absehbarer Zeit in einem kommerziellen Markt unterzubringen. Dafür ist es nötig das System nach einem Entwicklungszyklus auf seine Eigenschaften zu überprüfen. Ziel ist es, die Zuverlässigkeit der Sensordaten sowie des Geräts selbst aber auch die Nutzbarkeit und Verfügbarkeit sicherzustellen. Eine beliebte Maßnahme hierzu ist das Hardware-in-the-Loop-Verfahren (HiL). Bei diesem Verfahren werden die Inputdaten von eingebetteter Hardware simuliert, die darauffolgenden Ausgangssignale analysiert und mit erwarteten Daten verglichen. Auf diese Art und Weise wird die Nutzbarkeit und Zuverlässigkeit des Gerät geprüft.
Gegenstand dieser Arbeit ist die Konzeption eines HiL-Systems für das IPS. Es werden die Analyse der Rahmenbedingungen, die Konzeption sowie die Implementierung betrachtet.

Abstract

The Integrated Positioning System (IPS) is a device for self-localization. It is able to determine its own position relative to a starting point without external assumptions about the environment. The IPS obtains this capability through sensor fusion. It has an inertial measurement unit that measures acceleration and rotatory motion and a stereo camera that tracks pixels and can thus track motion. Combining the information of the two sensors cancels out the disadvantages of both. Thus, it is possible to generate trajectories accurate to the millimeter along which the device has moved.
The IPS is the subject of continuous development. The German Aerospace Center's goal is to place the device in a commercial market in the foreseeable future. Therefore it is necessary to test the system after a development cycle. The aim is to ensure the reliability of the sensor data and the device itself as well as its usability and availability. A popular measure for this is the hardware-in-the-loop (HiL) method. This method simulates the input data of embedded hardware, analyzes the following output signals and compares them with expected data. In this way, the usability and reliability of the device is tested.
The subject of this thesis is the design of a HiL system for the IPS. The analysis of the basic conditions, the conception as well as the implementation are considered.

Inhaltsverzeichnis

Abkürzungverzeichnis

HiL	Hardware-in-the-Loop
JSON	JavaScript Object Notation
DLR	Deutsches Zentrum für Luft und Raumfahrt
USB	Universal Serial Bus
CAN	Controller Area Network
FPGA	Field Programmable Gate Array
PWM	Pulse width modulation
VR	Virtual Reality
GPS	Global Positioning System
IMU	Inertialsensor
IPS	Integrated Positioning System

Erklärung

Hiermit versichere ich, dass ich
die vorliegende Arbeit selbstständig und nur unter
Verwendung der angegebenen Quellen und
Hilfsmittel angefertigt habe.

Berlin-Adlershof, der 8. Oktober 2019

1 Einleitung

Navigation ist seit jeher ein wichtiger Bestandteil der Menschheit[16] .Ob auf der Suche nach Nahrung, Wasser oder einer sicheren Bleibe. Das Zurechtfinden in unbekanntem Terrain stellt den Menschen immer aufs Neue vor Herausforderungen. Hierzu zählen unter anderem die Bestimmung der eigenen Position in einem größeren Kontext sowie die Festsetzung der Bewegungsrichtung zum Zielort. In der Vergangenheit verließen sich die Menschen zunächst auf ihre Instinkte um ihren Zielort zu erreichen. Mit der Zeit erlernten sie, die natürliche Gegebenheiten zu nutzen, wie beispielsweise den Sternenhimmel oder den Sonnenstand. Ein bekanntes Instrument zur Navigation mithilfe der Sonne ist der Sextant[10].

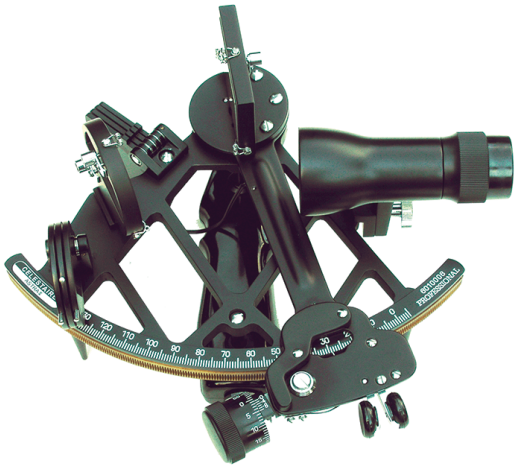

Abbildung 1.1: Sextant zur Navigation mittels Sonnenstand[2]

Mit der Exploration des Weltalls und der einhergehenden Verwendung von Satelliten, wurde das GPS erfunden. GPS steht für Global Positioning System. Es verwendet

codierte Radiosignale, welche von Satelliten ausgestrahlt werden zur Positionsbestimmung. Die Kenntnis über Signallaufzeiten und der Position von Satelliten, sowie derer Uhrzeit ermöglicht es einem GPS-Empfänger seine genaue Position zu bestimmen[13]. GPS ist beinahe auf der gesamten Welt verfügbar und dadurch essentiell in der modernen Navigation. Voraussetzung für die Verwendung von GPS ist allerdings, dass der GPS-Empfänger zeitgleich ein Verbindung mit vier Satelliten aufbauen kann. In Bereichen, in denen keine Sicht zu GPS-Satelliten besteht, wie beispielsweise in Bergwerken oder Schiffsrümpfen ist es aus diesem Grund nicht immer möglich, mittels GPS zu navigieren. Diese Problematik ist die Motivation für die Entwicklung des IPS.

Abbildung 1.2: Das Intergrated Positioning System[4]

Diese Arbeit ist im Projekt Integrated Positioning System (IPS) des Deutschen Zentrums für Luft- und Raumfahrt Berlin-Adlershof angesiedelt. Das IPS ist ein Gerät zur Selbstlokalisierung. Es kann sowohl im Freien, als auch innerhalb geschlossener Umgebung verwendet werden. Um die Eigenposition festzustellen, nutzt das IPS die Informationen aus einer Stereokamera und fusioniert diese mit den Daten eines Inertialsensors. Auf diese Weise werden die Nachteile beider Sensoren weitestgehend ausgeglichen[1]. Folgend wird aus den fusionierten Sensordaten eine Trajektorie erzeugt, entlang derer das IPS während eines Messlaufs bewegt wurde. Diese Auswertung wird durch die IPS-Applikation durchgeführt und findet auf einem externen Rechner statt.
Das IPS und die IPS-Applikation sind Gegenstand konstanter Weiterentwicklung. Dabei muss sichergestellt werden, dass für beide Instanzen weiterhin Verfügbarkeit und Nutzbarkeit gewährleistets sind.

1.1 Motivation

Computer und elektronische Elemente halten heutzutage immer häufiger Einzug in konventionelle Gebiete. Technische Entwicklungen wie das Internet-of-Things zeigen eine steigende Tendenz, alle Aspekte unseres Lebens digitalisieren zu wollen[8]. Als ein Beispiel dazu dient die Automobilbranche. Während Autos vor 50 Jahren höchstens einen elektrischen Anlasser besaßen, sind elektrische Bauteile aus dem modernen Auto nicht mehr wegzudenken. Probleme die früher mechanisch gelöst wurden, haben heutzutage in der Regel eine elektronische Lösung, wie beispielsweise die Getriebe- oder Motorsteuerung. Ebenfalls steigt die Anzahl der Assistenzsysteme. Diese können verhältnismäßig unwichtig sein oder von großer Bedeutung für die Sicherheit der Fahrzeuginsassen[15].
Da es sich teilweise um sicherheitskritische Komponenten handelt, ist die fehlerfreie Funktion dieser Komponenten unter allen Umständen sicher zu stellen. Mit der einhergehenden Digitalisierung wächst allerdings auch die Komplexität dieser Systeme und derer Software. Es ist sicherzustellen, dass die Komponenten weiterhin korrekt miteinander agieren. Aus diesem Grund ist es wichtig, diese in einem möglichst realitätsnahem Umfeld zu testen.
Die zu testenden Komponenten in ein reales System zu verbauen und dort zu überprüfen bringt Probleme mit sich. Würde man eine experimentelle Motorsteuereinheit in einem realen Auto verbauen, könnte das bei Fehlfunktionen fatale Folgen für die Testbeteiligten haben. Ebenfalls ist es sehr kostspielig Fehler zu beheben, welche erst entdeckt werden, während das Zielsystem der Komponente bereits montiert ist, da man sich zum Zeitpunkt des Testens schon in einer späten Entwicklungsphase befindet. Es muss ein Weg gefunden werden, Komponenten gefahrlos zu testen und Fehler bereits früh im Entwicklungszyklus zu identifizieren[9].
Sinnvoll ist es, die Systemumgebung von zu testender Hardware zu simulieren. Dieser Gedanke ist die Basis für das Hardware-in-the-Loop-Testverfahren (HiL).
HiL-Testverfahren generieren zu erwartende Inputs auf Testkomponenten und analysieren die Outputs auf Richtigkeit[7]. Ein simples Beispiel hierfür ist der Belastungstest einer Türklinke. In diesem Test wird die in einer Testvorrichtung angebrachte Türklinke maschinell tausendfach hintereinander betätigt. Auf diese Weise will der Hersteller der Klinke herausfinden, welche Belastungen die Klinke unter realen Bedingungen aushält.
Analog dazu kann man Komponenten im Auto auf diese Weise testen. Eine Getriebesteuereinheit wird dabei nicht im realen Auto verbaut, sondern an ihren Schnittstellen

mit einem Testsystem verbunden, welches die Motordrehzahl simuliert. Im späteren Verlauf dieser Arbeit wird dieses Beispiel konkretisiert.
Das IPS ist ein Produkt in der Entwicklung. Zum jetzigen Zeitpunkt gibt es allerdings kein Verfahren mit, dem es möglich ist, das Zusammenspiel zwischen Hardware und Software des IPS zu überprüfen. Die Zuverlässigkeit des Systems soll auch nach vielen Entwicklungszyklen gegeben sein. Hierzu muss kontrolliert werden, ob das Gerät und die Software gemeinsam nach einer Weiterentwicklung weiterhin sinnvolle Daten produzieren. An dieser Stelle bietet sich das Hardware-in-the-loop-Testverfahren an. Die folgende Arbeit beschreibt die Entwicklung eines solchen Testverfahrens für das IPS-Navigationssystem.

1.2 Aufgabenstellung

Das DLR verfügt bereits über eine Testinfrastruktur. Diese lässt sich über eine interne Website abrufen, der IPS-Website. Auf der IPS-Website existieren diverse Buttons für verschiedene Tests, darunter Softwaretests oder Performancetest für die IPS-Applikation. Bei den Softwaretests werden die Komponenten der Appliaktion mit Integrationstests überprüft. Bei Performancetests werden mit dem IPS aufgezeichnete Testläufe von der Applikation prozessiert. Daraufhin werden die Ergebnisse der IPS-Applikation auf Genauigkeit geprüft und es wird verglichen, ob diese im Vergleich zu vergangenen Performancetest Verbesserungen oder Verschlechterungen aufweisen. Diese Test werden täglich mit der neusten Version der IPS-Applikation durchgeführt, können allerdings auch jederzeit manuell innerhalb des Intranets aufgerufen werden.

Eine HiL-Testumgebung soll auf ähnliche Art und Weise in diese Testinfrastruktur integriert werden. Das Hauptziel der HiL-Testumgebung ist es die Zuverlässigkeit des IPS bei konstanter Entwicklung zu gewährleisten. Das beinhaltet, das IPS-System auf seine Nutzbarkeit und Genauigkeit zu überprüfen. Was bedeutet, nach einer Weiterentwicklung werden die Eigenschaften des IPS und seiner Sensoren überprüft und mit Soll-Daten, einem sogenannten Testkatalog, verglichen. Anhand des Testkatalogs kann entschieden werden, ob das System weiterhin für seinen Zweck geeignet ist. Um dieses Ziel zu erreichen sind einige Zwischenschritte notwendig. Zunächst müssen umfangreiche Analysen des IPS-Systems und der Systemumgebung vorgenommen werden. Es muss klar sein welche Möglichkeiten sich im DLR zur Realisierung einer HiL-Umgebung ergeben.

Dazu gehört eine Recherche zu bereits etablierten Technologien in der Industrie sowie die Analyse der DLR-Testinfrastruktur , der IPS-Hardware und der IPS-Applikation. Anhand der Analysen lässt sich ein Konzept für eine konkrete Hil-Umgebung und die Einbindung in die DLR-Testinfrastruktur aufstellen. Fürs Erste ist es das Ziel, das Konzept der HiL-Umgebung für das IPS zu bestätigen (Proof of Concept). Hierzu soll ein einzelner simpler Testfall implementiert werden. Ein Testfall ist ein zu testender Aspekt an der jeweiligen Hardware. Die gesamte Verarbeitungskette einer HiL-Umgebung soll für diesen Testfall durchlaufen werden. Zum Schluss soll das Ergebnis ausgegeben werden. Wird dieser Testfall erfolgreich abgeschlossen, kann der Testkatalog mit anderen Testfällen erweitert werden. Anschließend soll die HiL-Umgebung in die IPS-Website eingekoppelt werden. Analog zu den Software- und Performancetests soll die Hardware täglich in Kombination mit der Software kontrolliert werden.

1.3 Arbeitsplan

Recherche bezüglich etablierter Technologien Bei der Recherche zu verfügbaren Technologien geht es darum, einen Überblick über den Stand der Technik zu gewinnen. Zudem soll ein Überblick über etablierte Anbieter von HiL-Umgebungen sowie deren Anwendungsbereiche geschaffen werden. Es werden verschiedene HiL-Umgebungen betrachtet. Daraus soll eine Abschätzung getroffen werden, ob bereits entwickelte Geräte im Rahmen der IPS-Testinfrastruktur Verwendung finden könnten. Ebenso lassen sich daraus Erkenntnisse zur Konzeption eigener Umgebungen ziehen.

Analyse der IPS-Systemumgebung Zunächst ist es wichtig, einen Überblick über die Funktionsweise der IPS-Systemumgebung zu gewinnen. Dabei ist zu verstehen, wie das IPS mit der dazugehörigen Software kommuniziert und wie der Datenfluss innerhalb der IPS-Applikation funktioniert. Ziel dieser Analyse ist, Schnittstellen für die HiL-Umgebung zu identifizieren.

Analyse der IPS-Datensätze Aufgezeichnete Datensätze des IPS sollen ausgewertet werden. Anhand dieser Datensätze lässt sich erkennen, welche Werte bei einem normalen Messlauf zu erwarten sind. Es werden sowohl Messläufe mit hoher Genauigkeit als auch Messläufe mit vielen Fehler analysiert. Es ist darauf zu achten, in welchem Wertebereich

die zu erwartenden Daten liegen sollten. Ziel dieser Analyse ist, aus den Datensätzen konkrete Testfälle abzuleiten.

Analyse der DLR-Testinfrastruktur Eines der Ziele ist, den Zugriff auf die HiL-Umgebung innerhalb des Intranets zu gewährleisten. Dazu ist es nötig, die gegebene DLR-Testinfrastruktur zu betrachten. Die Testumgebung für die IPS-Applikation ist bereits in dieser Infrastruktur enthalten und vom Intranet aus verfügbar. Analog zur Softwaretestumgebung soll auch die HiL-Umgebung in das Intranet eingebunden werden. Einbindungsmöglichkeiten zu identifizieren gilt als Ziel dieser Analyse.

Ableitung von Testfällen und Testkatalog Nach der Analyse der Datensätzen, sollen Testfälle abgeleitet werden. Zuvor wird entschieden, welche Sensorwerte getestet werden sollen. Anschließend müssen Referenzwerte bestimmt werden, beziehungsweise ein Bereich in dem sich die empfangenen Daten bewegen können. Entsprechen die gemessenen Werte nicht den Refferenzwerten ist das Gerät in jenem Testfall nicht zuverlässig. Neben Sensorwerten wird die korrekte Funktionsweise des Geräts geprüft. Das beinhaltet beispielsweise das korrekte initialisieren all seiner Komponenten. Oberste Priorität haben die Testfälle, welche unbedingt erfolgreich abgeschlossen werden müssen, um die Funktionalität des IPS zu gewährleisten. Anhand der Testfälle wird der Testkatalog erstellt. Der Testkatalog stellt die Gesamtheit aller Testfälle dar.

Ableitung von Möglichkeiten im DLR Mit Hilfe der Ergebnisse aus den vorangegangenen Analysen soll abgeleitet werden welche Möglichkeiten sich zur Umsetzung dieser Aufgabe im DLR ergeben. Aus Analyse der Testinfrastruktur sowie der Analyse der Systemumgebung der IPS lassen sich die Anforderungen an die HiL-Umgebung ermitteln. Die Anforderungen umfassen die Art und Weise wie Daten ausgelesen werden und wie der Zugriff auf die HiL-Umgebung beziehungsweise die Einbindung in die DLR-Testinfrastruktur gewährleistet wird. Die Recherche bezüglich etablierter Technologien soll Aufschluss darüber geben, ob es sich lohnt vorgefertigte Hardware für dieses Projekt zu nutzen. Es wird eine Kosten-Nutzen-Abwägung eines solchen Geräts vorgenommen. Ziel des Zwischenschritts ist, den Grundstein für die Konzeption der HiL-Testumgebung zu legen. Hier werden die Rahmenbedingungen für das Projekt gelegt.

Konzeption von Software & Hardware Die eigentliche Konzeption ist der wichtigste Schritt in der Entwicklung der HiL-Umgebung. Hier geht es um die Zusammensetzung der aus den Analysen gewonnenen Erkenntnisse bezüglich des IPS. Anhand der Systemanalyse ist bekannt wie die Schnittstelle zum IPS aussieht und wie diese verwendet werden kann. Nach der Analyse ist deutlich, welche Testfälle in den Testkatalog aufgenommen werden müssen und demnach zu überprüfen beziehungsweise zu implementieren sind. Des Weiteren ist aus der Analyse der DLR-Testinfrastruktur bekannt wie die HiL-Umgebung eingebunden werden kann. Aus den genannten Erkenntnissen lässt sich dann ein grobes Testszenario konzipieren aus dem sich dann Software- sowie Hardwarekomponenten ergeben.

Implementierung und Evaluierung der HiL-Testumgebung In der Evaluierung geht es darum die zuvor implementierte HiL-Umgebung auf ihre Nutzbarkeit zu überprüfen. Es wird festgestellt, ob mit Hilfe des entwickelten System eine Aussagen über die Qualität des IPS getroffen werden kann.

1.4 Aufbau der Arbeit

Zunächst werden Grundlagen erklärt. Diese Grundlagen sollen Wissen bereitstellen um die Entwicklung einer HiL-Umgebung aufnehmen zu können. Allgemeinheiten von HiL-Systemen sowie deren Vorteile werden erläutert. Zudem werden typische Anwendungsfälle anhand von Beispielen vorgestellt, um ein gewisses Verständnis zu Hardware-in-the-Loop zu vermitteln und HiL-Systeme in den Produktentwicklungszyklus einzuordnen zu können.
Anschließend wird auf das IPS-Navogationsystem eingegangen. Um eine HiL-Umgebung zu konzipieren ist die Kenntnis über die Spezifikationen der zu testenden Hardware Bescheid erforderlich. Es werden Funktionsweise sowie Systemumgebung und Schnittstellen erklärt. Die Identifizierung der Anforderungen an die Testumgebung stellt hierbei das Ziel dar.
Nachfolgend werden die Analyseergebnisse mitgeteilt. Aus ihnen ergibt sich ein konkreter Leitfaden für ein Konzept. Das ausgewählte Konzept wird im Kapitel 'Konzeption' genauer erklärt. Anschließend folgen dann Implementierung und Evaluation. Es wird die Umsetzung des Konzepts beschrieben und bewertet wie sehr es sich im Unternehmenskontext eignet.

2 Grundlagen

2.1 Grundlegende Begriffe

In diesem Abschnitt werden grundlegende Begrifflichkeiten definiert . Diese Begriffe werden in der Arbeit vermehrt Verwendung finden. Um Missverständlichkeiten zu vermeiden, also an dieser Stelle ein Einordnung der Begriffe in den Kontext dieser Arbeit.

Testsubjekt Als Testsubjekt wird die Komponente bezeichnet, welche Gegenstand des Tests ist. In dieser Arbeit betrifft es beispielsweise das IPS.

Testprojekt Beim Testprojekt handelt es sich um das Projekt zu dem die zu testende Komponente gehört, beziehungsweise das System in das das Testsubjekt integriert werden soll. Ist eine Motorsteuereinheit das Testsubjekt, so ist das Auto, in welches diese eingebaut werden soll, das Testprojekt.

Testszenario Das Testszenario beschreibt die Abfolge an Aufgaben , die ausgeführt werden, um einen Hardware-in-the-loop-Testlauf zu absolvieren. Dazu zählt das Ausführen von Testfällen aber auch die Einrichtung der für den Testlauf nötigen Umgebung.

Testfall Der Begriff Testfall beschreibt einen überprüfbaren Aspekt am Testsubjekt . Ein Testfall kann ein Sensorwert sein der auf Korrektheit und Verfügbarkeit überprüft wird.

Testkatalog Der Testkatalog ist die Sammlung aller Testfälle.

Testmethodik Hiermit ist die konkrete Art und Weise, wie ein Testfall überprüft wird, gemeint. In der Regel handelt es sich um einen Algorithmus.

Passives & aktives System Die Bezeichnungen aktiv und passiv im Bezug auf eingebetteter Hardware beschreibt die Art, wie diese Daten oder Signale aufnimmt. Als passiv wird ein System bezeichnet welches keine direkte Stimulation durch seine Anschlüsse erfährt. Es existiert kein direkter Input. Trotzdem kann Output generiert werden. Im Gegensatz dazu haben aktive Systeme einen Anschluss, über den sie direkt von außen angesprochen werden müssen.

Statische & dynamischer Test Mit statisch und dynamisch wird die Art des Tests beschrieben. Es wird angegeben, ob die Daten, mit denen das Gerät stimuliert wird, veränderlich, also dynamisch oder gleichbleibend, also statisch sind.

2.2 HiL-Systeme

Der folgende Abschnitt bezieht sich auf die Quellen von National Instruments [7] sowie der Bachelorarbeit von Daniel Lorenz [12].
Moderne Systeme sind Gegenstand konstantem Wachstums in ihrem Umfang sowie ihrer Komplexität. Sie bestehen zu einem großen Teil aus digitalen Komponenten, weshalb die Software dieser Systeme ebenfalls eine essentielle Rolle spielt. Auch die Software ist in den letzten Jahren in Umfang und Komplexität gewachsen. Das Testen dieser Systeme hat sich aus diesem Grund zu einer Herausforderung entwickelt und stellt einen festen sowie auch wichtigen Schritt in der Entwicklung dar. Allumfassendes und zuverlässiges Testen ist absolut notwendig, um das Konzept eines Designs zu bestätigen.
An dieser Stelle bietet es sich an, ein Beispiel zu betrachten. Der Gedanke aus der Einleitung, Autos hätten immer mehr elektronische Komponenten,wird hier weiter geführt. Die Anzahl an Assistenzsystemen wächst stetig. Um im modernen Markt mit anderen Autoherstellern in den Wettbewerb einzutreten, ist es unerlässlich diesem Trend zu folgen. Die Assistenzsysteme umfassen verschiedenste Komponenten, darunter Kameras, Steuergeräte für Motor und Getriebe oder Radarsysteme. Wie bereits in der Einleitung angeschnitten, sind einige dieser Komponenten für die Sicherheit der Fahrzeuginsassen von großer Bedeutung. Aus diesem Grund müssen sie Gegenstand ausgiebigem Testens werden.
Der wohl erste Gedanke zur Überprüfung einer Komponente wäre das Einbauen in das finale Testprojekt. So hätte man die Möglichkeit, das Zusammenspiel mit den anderen Komponenten sowie auch das Verhalten in der Realität zu untersuchen.

Allerdings birgt diese Art zu testen einige Probleme in sich. Zunächst wird ein hoher Testaufwand erzeugt, um alle Komponenten zu checken. Man müsste das reale Testprojekt in jede Situation bringen, für die es ausgelegt ist und gegebenenfalls sogar Fehlerzustände durch nichtsachgemäße Bedienung hervorrufen. Außerdem ist im realen Umfeld nicht bestätigt, dass alle Komponenten des Testprojekts gleichzeitig korrekt funktionieren. Sind nun einige Komponenten voneinander abhängig, lässt sich beispielsweise ein erfolgreich abgeschlossener Testfall nicht mehr nur auf das Testsubjekt selbst zurückführen. Das Ergebnis des Testfalls ist damit nicht eindeutig.
Ein weiteres großes Problem ist die zeitliche Verzögerung. Bis ein Testsubjekt getestet werden kann, müssen alle Komponenten des Testprojekts bereits angefertigt und verbaut worden sein. Sollte sich zu diesem Zeitpunkt ein Änderung aufgrund von fehlgeschlagenen Testfällen auftun, entsteht ein zusätzlicher Zeitaufwand. Zu einem späten Zeitpunkt müssen in der Entwicklung neue Komponenten designt werden. Das erzeugt zusätzliche Testkosten und eine ungewisse Verzögerung bis zur Marktreife des Testprojekts.
Durch beschriebene Herausforderungen stellt sich berechtigter Weise die Frage, ob es möglich ist, gründliche, zuverlässige und realitätsnahe Tests an einem Testsubjekt durchzuführen, während Kosten- sowie auch Zeitaufwand realistisch bleiben. Die Lösung für dieses Problem muss es ermöglichen Komponenten im Zusammenspiel mit dem Testprojekt zu testen ohne letzteres vollständig zusammenbauen zu müssen. Würde man die Systemumgebung von eingebetteter Hardware simulieren und dann deren Funktion überprüfen, könnte man so frühzeitig im Entwicklungszyklus Tests durchführen. Auf diesem Konzept bauen Hardware-in-the-Loop-Testsysteme auf.

2.2.1 Einordnung

Dieser Abschnitt definiert HiL-Systeme. Dieser sowie der nächste Abschnitt beziehen sich auf die publizierten Texte von National Instruments[7]
Grundlegend sei gesagt, dass es sich bei Hardware-in-the-Loop um ein Testverfahren. Dieses Testverfahren ist in erster Linie für eingebettete Hardware konzipiert. Hierbei wird die besagte Hardware mit einem HiL-System verbunden. Das HiL-System erfüllt dabei exakt die Schnittstellenanforderungen des Testsubjekts. Das HiL-System erzeugt dann reale Signale für das Testsubjekt. Es erhält Eingangssignale, die im realen Testprojekt auch zu erwarten wären [12]. Dem Testsubjekt wird auf diese Weise suggeriert, es befände sich in einem realen Umfeld. So können tausende realitätsnahe Tests durch-

geführt werden ohne die Kosten und den Zeitaufwand zu verursachen,die mit einem Testlauf in der Realität verbunden wären. Zum besseren Verständnis wird an dieser Stelle ein weiteres Beispiel aus der Automobilbranche herangezogen. Das Testsubjekt dieses Beispiels soll die Getriebesteuereinheit eines Automatikautos sein.

Abbildung 2.1: Steuereinheit ist Bestandteil eines höheren Systems[7]

Die Getriebesteuereinheit ist dafür zuständig, die Gänge zu wechseln. Sie steuert Aktuatoren an, welche den nächsten, beziehungsweise vorherigen Gang, einlegen. Um die Entscheidung treffen zu können, wann ein Schaltvorgang initiiert wird, erhält die Getriebesteuereinheit zu jedem Zeitpunkt Information über die Drehzahl des Motors sowie die Getriebeausgangsgeschwindigkeit. Mit diesen Informationen ist die Steuereinheit in der Lage die Gänge nach einem festgelegten Muster zu wechseln.

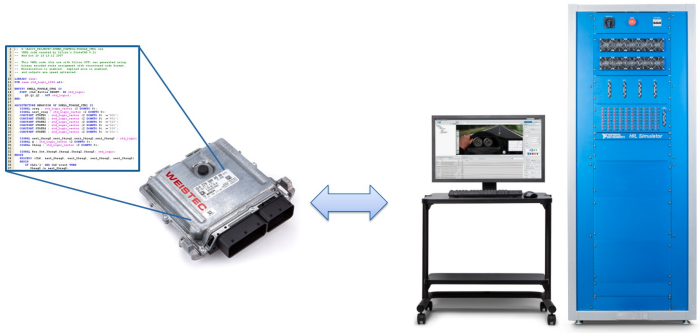

Abbildung 2.2: Höheres System wird mit HiL-Umgebung substituiert[7]

Das Testsubjekt wird nun an ein HiL-System angeschlossen. Das HiL-System ersetzt den Motor sowie das Getriebe. Es verwendet allerdings die vom Testsubjekt vorgegebenen Schnittstellen und das Kommunikationsprotokoll. Der Getriebesteuereinheit kommt es dennoch so vor, als befände sie sich in einem Auto. Es wurde eine reale Testumgebung

geschaffen, ohne einen echten Motor oder ein echtes Getriebe zu verwenden. Nun wird das Testsubjekt mit Drehzahlwerten als Input stimuliert und es wird kontrolliert, ob die Schaltvorgänge wie vorgesehen funktionieren.

2.2.2 Herausforderungen

An dieser Stelle bietet es sich an auf die Herausforderungen einzugehen, welche mit der Entwicklung einer HiL-Umgebung verbunden sind, um Vorbereitungen für die eigene Entwicklung treffen zu können. Unerlässlich für die Qualität der Tests, und somit für die Qualität des HiL-Systems, ist eine hohe Qualität der verwendeten Software. Die Software muss akribisch auf die Eigenschaften und Anforderungen des Testsubjekts abgestimmt sein. Dies bezieht sich neben den Eigenheiten der Software des Testsubjekts, den Hardware-Spezifikationen oder den konkreten Input- und Outputsignale, ebenfalls auf die korrekte Fehlereinfügung. Die Möglichkeit, gewollt Fehlerzustände im Testsubjekt hervorzurufen, muss gegeben sein. Selbst wenn diese im normalen Betrieb nicht vorgesehen werden. Fehler können sowohl durch Software als auch durch Hardware ausgelöst werden. Zum Beispiel könnte man gezielt falsche Signale übertragen, um die Software zu testen oder automatisch Hardwareverbindungen im laufenden Betrieb trennen.
Ebenfalls wichtig zu erwähnen ist, dass es keine universelle HiL-Plattform gibt. Jede HiL-Umgebung ist auf eine spezifische eingebettete Hardware ausgelegt. Es müssen Hardwareschnittstellen für das Testsubjekt individualisiert werden, so wie auch Modelle zur simulierung des Testsubjekts. Testsysteme, welche man sich kaufen kann, sind zwar oft modular und einstellbar konzipiert, die Tests für die jeweilige Hardware müssen allerdings selbst angepasst werden. Hersteller für HiL-Systeme können nur eine anpassbare Plattform anbieten, welche dann von den Entwicklern an das Testsubjekt angepasst werden muss. Aus diesem Grund ist eine der großen Herausforderungen bei der Entwicklung, die Balance zwischen einer hohen Testabdeckung durch Anpassung auf das Testsubjekt und dem damit verbundenen Zeitaufwand zu finden.
Zudem ist bei dem Einstellen auf die Zielhardware wichtig, die sich ändernden Anforderungen des Marktes zu beachten. Ein HiL-system sollte deswegen nicht nur für die derzeitige Version eines Testsubjekts ausgelegt sein. Neuentwicklungen des Testsubjekts sollten genauso mit der HiL-Plattform testbar sein. Demnach müssen diese Testplattformen erweiterbar und nachträglich individualisierbar sein.

2.2.3 Vorteile

Trotz der nicht zu vernachlässigenden Herausforderungen bringen HiL-Systeme signifikante Vorteile mit sich, welche im Folgenden erläutert werden. Die angeführten Vorteile basieren auf den Erkenntnissen von National Instruments [7], IMC [6] sowie Controllab[9].

Wie bereits erwähnt, simulieren HiL-Systeme in der Regel die Systemumgebung von eingebetteter Hardware. Daraus ergibt sich zunächst der Vorteil, nicht mit realen Objekten (abgesehen des Testsubjekts selbst) interagieren zu müssen. Durch intensive und experimentelle Tests könnte reale Hardware beschädigt werden und in manchen Fällen eine Gefahr für den Menschen darstellen. HiL-Systeme erhöhen die Sicherheit während des Tests. Das Testen von schwerer Maschinerie beinhaltet beispielsweise diverse Sicherheitsmaßnahmen. Diese werden durch HiL vollständig umgangen. Wie anfangs beschrieben, könnten manche Tests Schäden am Testprojekt bewirken, diese Tests lassen sich dank HiL trotzdem ausführen. So lässt sich das Testsubjekt über den Horizont der eigentlichen Operationreichweite untersuchen. Situationen, die in der Realität unter normalen Umständen nicht eintreten können, werden ebenfalls mit Tests abgedeckt. Durch Hil kann auch die Systemreaktion auf diverse Fehlersituationen getestet und analysiert werden. Diese Methodik ist eng mit Sicherheitsanalyse verwandt.

Ein weiterer Vorteil ist die Erhöhung der Qualität des Testsubjekts. Bei modellbasierten Designprozessen finden HiL-Systeme bereits zu sehr frühen Designphasen Anwendung. Einzelne Komponenten des Testprojekts können als Modell mit Software wie Simulink simuliert werden. Mit HiL-Systemen lässt sich danach direkt das Zusammenspiel zwischen simulierten Komponenten und den konkreten Testsubjekten austesten. Zwar wird das HiL-System mit wachsender Anzahl der Komponenten im Testprojekt ebenfalls komplexer , allerdings findet es während des gesamten Designprozesses Anwendung. Dadurch ist es HiL-systemen möglich, in frühen Designphasen Fehler zu entdecken, auf die umgehend eingegangen werden kann. Ohne HiL würden diese Fehler erst sehr spät erkannt werden.

Aus soeben genannten Gründen lässt sich der nächste Vorteil ableiten. HiL-Systeme lassen sich sehr stark in den Designprozess integrieren. HiL-Tests könnten mit Hilfe von Skripten ausgeführt werden. Manuelle Ausführung würde einen viel größeren Aufwand

mit sich bringen. Man könnte HiL ebenfalls in das Buildsystem für das Testprojekt integrieren. Die Testumgebung würde dann automatisch neuste Version von Software mit der Hardware testen. Das könnte man so konfigurieren, dass es nach jeder Änderung an der Software automatisch geschieht. Die Ergebnisse des Tests könnten gespeichert werden und anschließend auf beliebige Art visualisiert werden. Als Medium könnte man Dokumente oder Websites nehmen. Dann kann ausgewertet werden, ob das System nach besagter Änderung immer noch den Vorstellungen der Entwickler entspricht. Die Auswirkung jeder Änderung lässt sich so schnell und effektiv bestimmen.

Der nächste positive Aspekt von HiL-Systemen ist die enorme Zeitersparnis, welche sich bereits in vorherigen Vorteilen angedeutet hat. Die Zeitkosten von Fehlern erhöht sich mit fortschreitender Entwicklung. Fehler, welche erst während der Produktion oder erst bei der Inbetriebnahme entdeckt werden, haben schwerwiegende zeitliche Verzögerungen als Folge. Beispielsweise könnte es sich im Testprojekt um eine Fertigungsanlagen handeln, welche weit entfernt vom Entwicklerunternehmen montiert werden soll. Sollten sich Fehler aufzeigen, kann nur mit sehr begrenzten Mitteln auf diese eingegangen werden. Es ist zudem oft so, dass Maschinen bzw. Aktuatoren parallel entwickelt werden und deren eigentliches Zusammenspiel erst bei Inbetriebnahme erprobt werden kann. HiL-systeme umgehen diesen Sachverhalt vollständig. Fehler können bereits sehr früh entdeckt werden. Aus diesem Grund können HiL-systeme ein starkes Mittel für Zeitersparnis sein.

Neben großer Zeitersparnis birgt ein HiL-Testsystem ebenfalls eine große Kostenersparnis. Wie bereits erwähnt, sparen frühzeitig entdeckte Fehler Kosten ein, aber ebenfalls das Testen an simulierten anstatt realen Umgebungen. Die Produktion von Prototypen, das Betreiben der Anlage während des Tests aber auch zu treffende Sicherheitsmaßnahmen stellen hohe Kostenfaktoren dar. Durch HiL-Systeme werden diese komplett obsolet. Des Weiteren entfallen Arbeitsstunden. Zwar werden intensive Arbeitsstunden in die Entwicklung einer HiL-Testumgebung sowie in die Integrierung des HiL-Systems in den frühen Designprozess investiert, allerdings wird so immens viel Zeit wärend der Implementierung der Testprojekts gespart. Arbeitsstunden während der Implementierung des Systems sind wesentlich kostspieliger, zum Beispiel wenn man erst zum Montageort reisen müsste, um zu arbeiten.

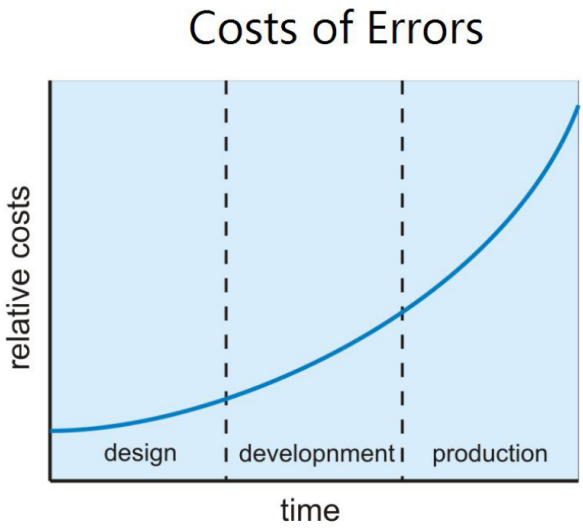

Abbildung 2.3: Steigende Kosten von Fehlern [9]

Der letzte große Vorteil bezieht sich auf die Fähigkeit, Dinge mit Hilfe von HiL simulieren zu können. Er befindet sich in der menschlichen Interaktion mit dem Testprojekt. Zwar sind menschliche Interaktionen im Testszenario in der Regel nicht vorgesehen, können allerdings als Test genutzt werden, um die Reaktion des Geräts herauszufinden. HiL-Systeme, welche auf menschliche Interaktion fixiert sind, nennen sich Trainingssimulatoren. Ein einfacher Flugsimulator ist nichts anderes als ein HiL-Simulator, in dem das Testsubjekt der Pilot ist. In Simulatoren können theoretisch auch Aspekte visualisiert werden, die im realen System nicht einsehbar sind. Zudem kann der Simulator durch falsche Bedienung nicht zerstört werden, was bedeutet, dass Training auch ohne Betreuung stattfinden kann.

2.3 Stand der Technik

In diesem Abschnitt werden etablierte Technologien vorgestellt . Zweck dessen ist es, eine Abschätzung treffen zu können, inwiefern ein bereits existentes System in diesem Projekt Anwendung finden könnte. Es werden Systeme verschiedener Anbieter sowie deren Vorteile vorgestellt. Ebenfalls wird auf die verschiedenen Varianten von HiL-Systemen eingegangen.

2.3.1 Einfache Frameworks

Zunächst existieren HiL-Umgebungen in Form von einfachen Frameworks. Diese benötigen keine zusätzliche Hardware, abgesehen eines normalen Desktoprechners. Das Testsubjekt wird in diesem Fall auf einfach Art und Weise mit dem Computer verbunden. Dafür könnte beispielsweise USB oder Ethernet genutzt werden. Getestet werden verhältnismäßig simple Subjekte wie zum Beispiel FPGAs oder Steuereinheiten. Der Test wird durch die einfache Schnittstelle zu den zu testenden Systemen ermöglicht sowie auch durch die Tatsache, dass es sich nur um eine einzelne Testkomponente handelt. Es ist nicht nötig das Zusammenspiel mehrerer Komponenten zu simulieren. Die Simulationsumgebung wird direkt auf dem Entwicklungsrechner ausgeführt. Ein Beispiel für diese Technologie wäre der DSP Builder der Firma Altera. Dieses Tool ist ausgelegt für FPGAs, die via USB verbunden werden. Es unterstützt auch die Verwendung von Simulink zur Simulation von Modellen. Abgesehen von diesen einfachen Frameworks wird bei HiL-Umgebungen zwischen monolithischen und verteilten Systemen unterschieden.

2.3.2 Monolithische HiL-Systemen

Bei monolithischen HiL-systemen handelt es sich um ein einzelnes dediziertes Testgerät. Dieses Testgerät bietet alle Schnittstellen der zu testenden Hardware an. Diese Systeme sind sehr modular und komponentenbasiert aufgebaut. Das bedeutet, dass der Entwickler der HiL-Umgebung diese Geräte zu seinen Bedürfnissen gestalten kann. Der Entwickler kann diese mit verschiedensten Echtzeitprozessoren oder auch I/O-Karten ausstatten. So kann der Monolith an mehrere Bussystheme angepasst werden. Es gibt I/O-Karten für den CAN-Bus, PWM, USB oder auch RS-232. Das Aufzeichnen von Bildern so wie auch die Interaktion mit FPGAs werden ebenfalls unterstützt. Ein bekannter Anbieter für

monolithische Systeme ist die Firma dSpace. dSpace bietet seine Monolithen in diversen Größen an. Die größten Tower bieten die meisten Individualisierungsmöglichkeiten.

Abbildung 2.4: Der Full-Size Tower von dSpace [3]

2.3.3 Verteilte HiL-Systeme

Verteilte HiL-Systeme bestehen im Gegensatz zu monolithischen Systemen aus mehreren Knotenpunkten. Jeder der Knotenpunkte simuliert einen Teil des Testprojekts. Dabei muss es nicht einmal ein konkretes Testsubjekt geben. Die einzelnen Knotenpunkte können auch miteinander kommunizieren und gegebenenfalls nur Software testen. Auf diese Weise lassen sich mehrere Teile des Systems unabhängig voneinander simulieren. Ebenso lässt sich der Datenaustausch zwischen den Komponenten besser nachvollziehen. [14]

2.3.4 Simulationsumgebung

Die Simulationsumgebung ist ebenfalls ein wichtiger Bestandteil von HiL-Systemen. Der Zweck von Simulationsumgebungen ist die Modellierung sowie Simulierung von mathematisch darstellbaren Systemen. Dabei handelt es sich dann in der Regel um die zu simulierenden Komponenten. Die modellierbaren Systeme können physikalisch, technisch oder theoretisch sein, sie sind entweder diskret oder kontinuierlich. Ein beliebtes Tool hierfür ist das zuvor öfter erwähnte Simulink. Es handelt sich dabei um eine Erweiterung von MATLAB und wurde von The Mathworks entwickelt. Simulink genießt hohe Beliebtheit und wird von beinah allen angebotenen HiL-Umgebungen unterstützt[14]

2.4 IPS

Der jetzt folgende Abschnitt wird sich auf das Testsubjekt dieses Projekts beziehen , das IPS. Um eine Hil-Testsystem für diese spezifische Hardware zu entwickeln, ist es nötig sich mit den Eigenschaften und Anforderungen dieser auseinander zu setzen.
Die vorgestellten Erkenntnisse basieren auf den Publikationen von Denis Greisbach [4], Hongmou Zhang [16] sowie einer wissenschaftlichen Publikationen des DLR [1]. Zunächst sei gesagt, dass es sich beim IPS um ein passives System handelt. Es wird nicht aktiv mit Input stimuliert, sondern generiert seinen Input selbst. Wie bereits in der Einleitung erwähnt, handelt es sich bei dem IPS um ein Navigationssystem. Um seine Eigenposition zu bestimmen, werden keine externen Informationen benötigt. Es geht dabei nach dem Prinzip der menschlichen Augen in Verbindung mit dem Gleichgewichtssinn vor. Die Bewegung , die das Innenohr wahrnimmt, wird mit den visuellen Informationen aus den Augen bestätigt[11]. Betrachten kann man dieses Phänomen beispielsweise bei sogenannter Motionsickness, ein Problem bei virtueller Realität (VR). Bei VR ist es oft der Fall, dass das Gesehene nicht mit dem übereinstimmt, was der Gleichgewichtssinn wahrnimmt. In Folge dessen kommt es zu Schwindel[5].
Wie dieses Prinzip technisch umgesetzt wird, ist im Folgenden grob erklärt. Ein genaues Verständnis der Algorithmik dahinter ist für dieses Projekt nicht nötig. Zusätzlich wird auf die Datensätze des Geräts sowie auf Anwendungsbereiche eingegangen.

2.4.1 Funktionsweise

Die Grundkomponenten des IPS sind eine Inertialmesseinheit (IMU), eine Stereokamera und eine FPGA. Optional kann das IPS mit GPS erweitert werden. Analog zu dem menschlichen Beispiel: die IMU stellt den Gleichgewichtssinn dar und die Stereokamera das Paar Augen des Menschen. FPGA steht für Field Programmable Gate Array. Die FPGA kontrolliert den Datenfluss innerhalb des Geräts und übernimmt noch weitere Aufgaben. Dazu aber an späterer Stelle mehr.
Ziel der Datenverarbeitung ist es, mit Hilfe der Informationen aus den Sensoren eine Trajektorie zu erzeugen, um den Weg, den das Gerät zurückgelegt hat, nachvollziehen zu können. Zu diesem Zweck gilt es mit den Sensordaten Bewegungsinformationen abzuleiten, aus denen schlussendlich Lagekoordinaten relativ zum Startpunkt gebildet werden können. Wie diese Informationen gewonnen werden, soll nun erklärt werden. Zunächst sei

aber gesagt,das das IPS selbst keine Verarbeitung dieser Daten vornimmt. Diese findet auf einem externen Rechner statt. Das IPS selbst generiert lediglich die nötigen Daten zur Navigation.
Als erstes wird erklärt wie Bewegunsinformationen aus Stereokameras generiert wird. Die Stereokamera nimmt Bildpaare mit einer Frequenz von 10 Hz auf. Ausgelöst werden die Kameras durch ein Triggersignal, um beide Kameras absolut gleichzeitig auszulösen. Das ist wichtig für spätere Verarbeitungsschritte. Die FPGA versieht die Bildpaare mit einem Zeitstempel. Dieser Zeitstempel ist der Zeitpunkt des Auslösens und wird später zur Einordnung der Bilder in den Messlauf genutzt. Die Bilder werden direkt auf dem verarbeitenden System abgespeichert. Der Zeitstempel wird den Bild als Dateinamen verliehen. Zur Gewinnung von Bewegungsinformationen aus diesen Bildern sind grob gesagt drei Schritte notwendig: Featurematching, Triangulation und Featuretracking.
Beim Featuretracking geht es darum, ein und denselben Punkt aus der Realität in mehreren Bildern wieder zu finden. Dieser Vorgang ist eine Voraussetzung für Triangulation. Pro Bilderpaar werden ca. 60 markante Bildpunkte gefunden, welche sich in beiden Bildern der Stereokamera wiederfinden lassen. Sehr gut funktioniert dieses Verfahren für Ecken oder Kanten mit hohem Kontrast. Gefundene Feature-Paare werden in Form von Koordinaten abgespeichert und für den nächsten Schritt verwendet.

Abbildung 2.5: Visualisierung von gefundenen Feature-Paaren

Triangulation bezeichnet ein Verfahren zur Distanzbestimmung. Es soll genutzt werden, um die reale Distanz von Bildpunkten zum Messgerät zu bestimmen. Um Triangulation durchzuführen, muss zunächst der Basisabstand (T in der Abbildung) und die Brennweite bzw. der Abstand zu Abbildungsebene (f in Abb. 2.6) der Stereokamera bekannt sein. Da ein definiertes Messgerät vorliegt. ist das der Fall. Ebenfalls wird die

Position der gefundenen Featurepaare in beiden Bilder. benötigt. Diese sind durch den vorherigen Schritt gegeben. Mit dem bekannten Wissen kann dann analog zur Abbildung die reale Distanz berechnet werden (Z in der Abb. 2.6).Dazu wird der Strahlensatz genutzt. Eine Rechnung hierzu findet sich im Anhang dieser Arbeit (A.1). Wird zu einem Feature-Paar eine Distanz gefunden, wird es zusammen mit der gefundenen Distanz abgespeichert und für den letzten Schritt genutzt.

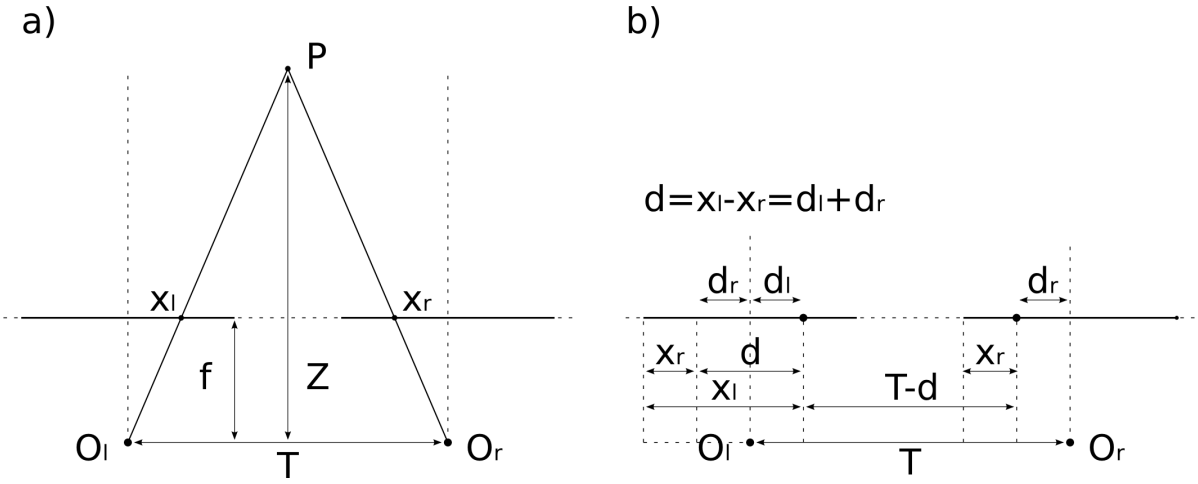

Abbildung 2.6: Triangulation mithilfe des Strahlensatzes

Der letzte Schritt ist das Feature-Tracking. Dabei wird versucht, die gefundenen Feature-Paare mit bekannter Distanz in folgenden Bildpaaren wieder zu finden. Sollte dies für ein Feature-Paar erfolgreich durchgeführt werden, kann nachvollzogen werden, wie sich besagtes Feature im Verhältnis zum Gerät bewegt hat. Anhand dieser Bewegung lässt sich die Eigenbewegung des IPS ableiten.

Probleme treten bei dieser Methode der Bewegungsermittlung auf, sofern nicht genügend Feature gefunden werden. Es gibt dafür mehrere Auslöser.Zum einen kann das Bildpaar vollkommen über- oder unterbelichtet sein, was sämtliche Kontraste zu nichte machen würde und weswegen es keine markanten Punkte mehr gebe. Zum anderen könnte es sein, dass die aufgezeichnete Ebene zu weit vom IPS entfernt ist, als dass Triangulation mit einem zuverlässigen Ergebnis durchgeführt werden könnte. Durch die Brennweite bzw. dem Abstand der Abbildungsebene zur Basisweite wird beim IPS festgelegt, in welchem Distanzbereich es zuverlässig Features entdeckt.

Die nächste Methode zur Gewinnung von Bewegungsinformationen ist die Verwertung der Daten aus dem Inertialsensor. Die IMU misst Beschleunigung in eine Richtung entlang drei Achsen, die senkrecht aufeinander stehen. Vergleichbar wäre ein dreidimensionales Koordinatensystem. Zudem misst die IMU die Drehrate um diese Achsen. Also misst sie insgesamt Beschleunigung und Geschwindigkeit auf sechs verschiedenen Achsen. Mit ca. 420 Hz werden die Werte des Inertialsensors ausgemessen. Anschließend kann die gemessene Beschleunigung zweifach integriert werden. Dadurch erhält man erst die Geschwindigkeit und danach den zurückgelegten Weg zwischen zwei Auslesungen der IMU. Das Ergebnis, wenn man alle diese Werte zusammen addiert, ist der vollständig gemessene Weg.
Die gemessene Drehrate wird ebenfalls nach der Zeit integriert. Nach der Integration erhält man die Ausrichtung des Geräts. Analog zur Berechnung des Weges, können diese Werte ebenfalls aufeinander addiert werden, um die finale Ausrichtung zu erhalten. Visualisiert ist diese Abfolge in Abbildung 2.7.

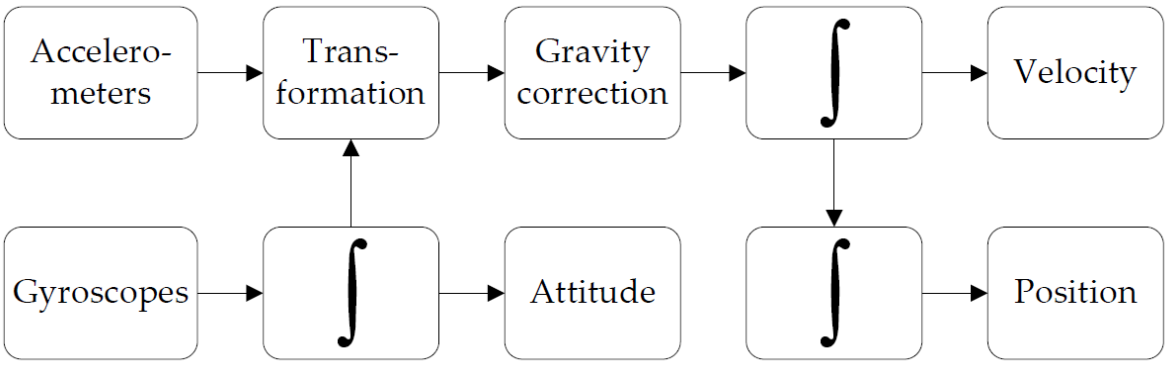

Abbildung 2.7: Schema der Integration

IMU sowie Stereokamera allein haben allerdings Messfehler. Würde man allein Daten aus einem der beiden Hauptsensoren beziehen, würde die entstehende Trajektorie anfangen zu driften. Um das Driften zu verhindern, werden die Trajektorien beider Sensoren miteinander kombiniert. Dafür wird ein Kalman-Filter herangezogen. Auf folgendem Bild lassen sich die einzeln gemessenen Trajektorien betrachten. Beim abgebildeten Messlauf handelt es sich um einen Rundgang durch das DLR-Gebäude in Berlin-Adlershof. Die grün gestrichelte Trajektorie ist eine Referenztrajektorie. Diese soll schlussendlich nach der Messung entstanden sein, beziehungsweise wird an ihr entlang gemessen. Die rote Trajektorie erhält man, wenn lediglich IMU-Daten für die Bewegungsbestimmung bezo-

gen werden. Diese driftet sehr schnell ab, da sich nicht eindeutig bestimmen lässt, welche Messungen der wahren Bewegung entsprechen und welche lediglich Fehler durch Erschütterungen am Gerät sind. Die gelbe Trajektorie repräsentiert den zurückgelegten Weg, sollte man lediglich Kameradaten verwenden um diese zu erzeugen. Sie driftet ebenfalls ab. Grund dafür ist das fehlende Wissen über die Ausrichtung des Geräts. Kombiniert man beide Trajektorien mittels Kalman-Filter erhält man die blaue Trajektorie, welche ungefähr der Referenztrajektorie entspricht.

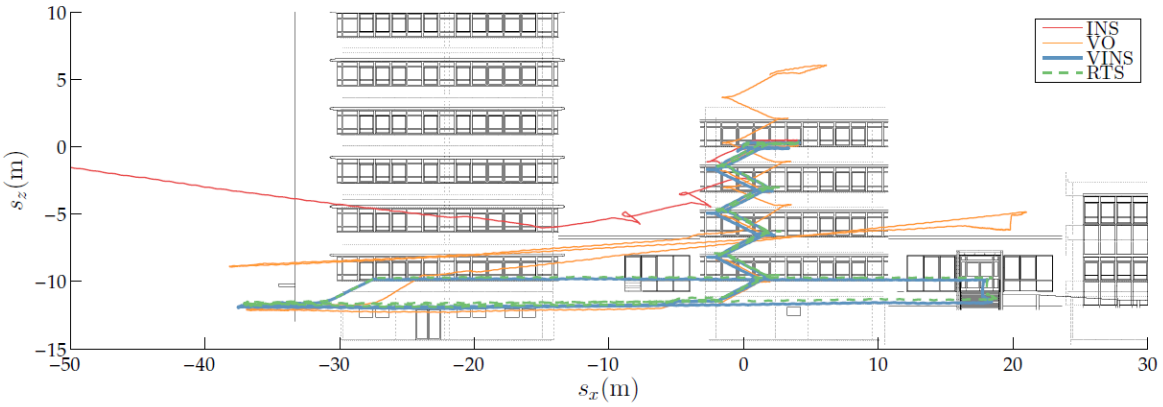

Abbildung 2.8: Seitenansicht auf Messlauf im DLR-Gebäude

Im Anhang dieser Arbeit befindet sich zusätzlich noch eine Abbildung mit einer Draufsicht dieser Trajektorien(A.2).

2.4.2 Systemumgebung

Wie bereits erwähnt nimmt das IPS selbst keine Verarbeitung seiner Daten vor. Diese Rolle übernimmt die IPS-Applikation. Sie kann entweder mit einem angeschlossenen IPS-Gerät verwendet werden oder bereits aufgezeichnete Messläufe verarbeiten. Das IPS allein lässt sich nicht ohne die Applikation verwenden.
Aufgebaut ist die IPS-Applikation in einer Pipeline-Architektur. Solch eine Prozessierungskette ist nötig um high-level Informationen, wie beispielsweise Trajektorien, aus low-level Daten, wie Beschleunigungswerten zu bestimmten Zeitpunkten, abzuleiten. Die einzelen Verarbeitungsschritte dieser Pipeline sind in verschiedene Container gekapselt. Diese Container werden in der IPS-Applikation "Feeder"genannt.
Feeder stellen die Grundstruktur zur Realisierung von Datenaustausch, -aufnahme so-

wie auch -verarbeitung dar. Die Einstellungen für einzelne Feeder werden per standardisiertem XML vorgenommen. Diese Konfigurationsdatei wird später genauer erläutert. Vergleichbar wären "Feeder"mit den Verarbeitungsstationen in einer Fabrik.

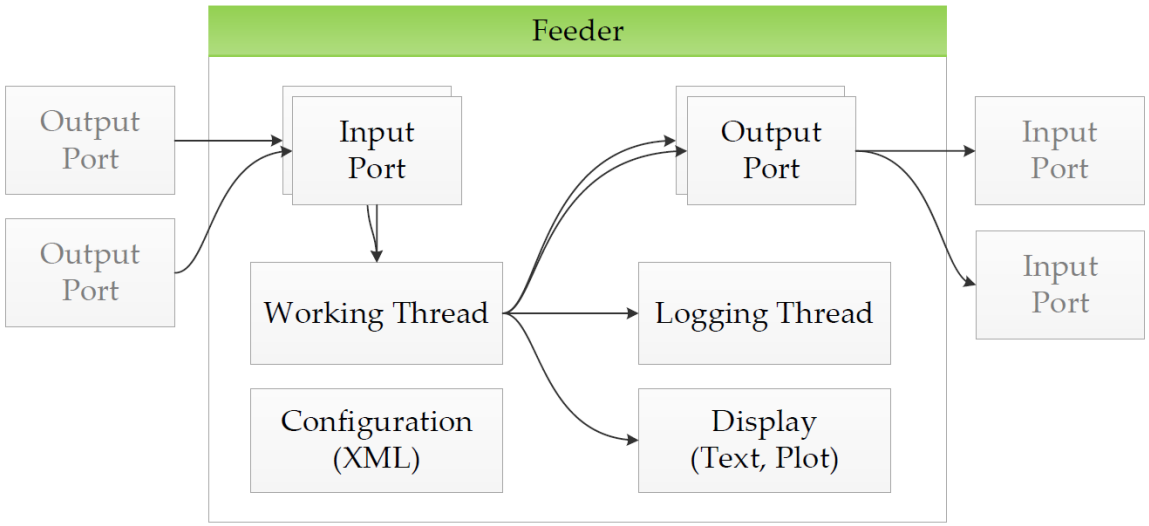

Abbildung 2.9: Feeder-Konzept

Die einzelnen Feeder bieten eine Schnittstelle, um die Daten innerhalb des Feeders einzusehen. Dies kann graphisch aber auch testbasiert geschehen. Optional lassen sich diese Daten auch direkt als Dateien abspeichern. Jeder Feeder kann mehrere Outputs sowie auch mehrere Inputs besitzen. Die Outputs eines Feeders sind mit den Inputs des Folgefeeders verbunden. Welche Feeder miteinander verbunden sind wird durch das XML festgelegt. Dadurch lässt sich die Verarbeitungskette individualisieren.
Im Allgemeinen sieht das konkrete Applikationsnetzwerk der IPS-Applikation aber aus wie in Abbildung 2.10.

Das gesamte System (Hard - und Software) kann in drei Teile unterteilt werden: Hardware, Middleware und das Applikationsnetzwerk. Mit Hardware werden die konkreten Sensoren bezeichnet. Daten der Sensoren werden mit Hilfe von Treibern extrahiert. Die Daten werden allerdings nicht direkt in die Applikation geleitet. An dieser Stelle kommt die Middleware zum Einsatz. Bei der Middleware handelt es sich praktisch um virtuelle Sensoren. Sie sind für das Auslesen der Daten aus den realen Sensoren zuständig. Sie nehmen Vorverarbeitungen vor, um die Daten für die Applikation nutzbar zu machen. Die Middleware stellt die Schnittstelle zum Applikationsnetzwerk dar. Die Feeder der

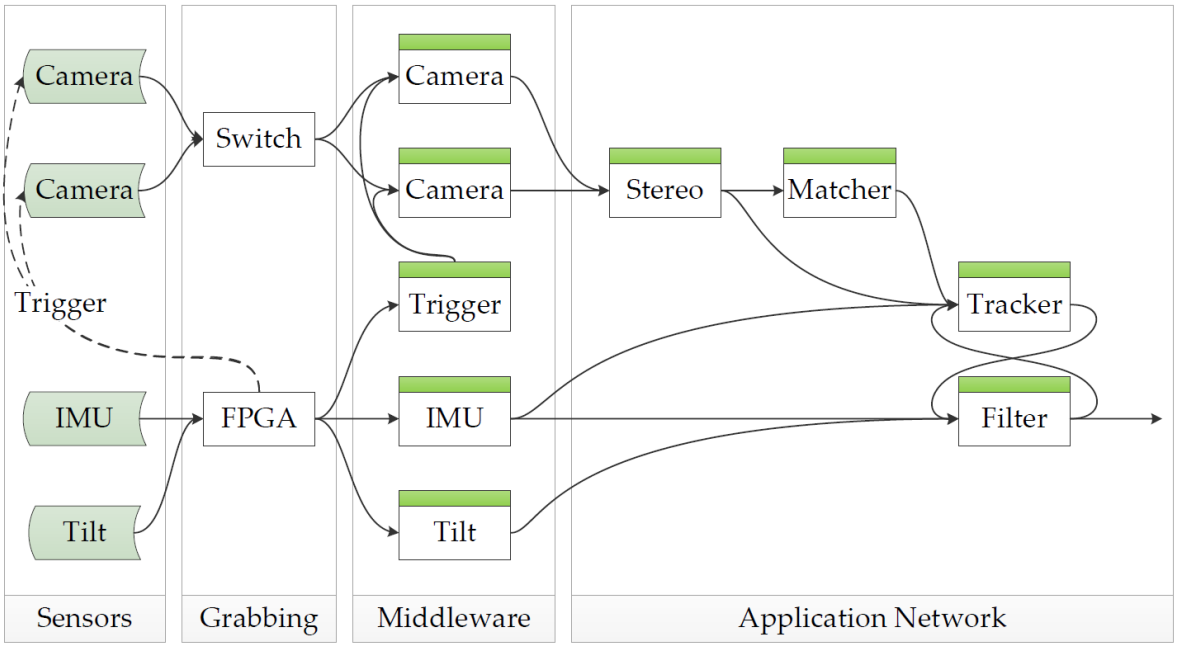

Abbildung 2.10: Das Feeder-Netzwerk der IPS-Applikation

Middleware können auch durch ihr Dateingegenstück ausgetausch werden und so bereits aufgezeichnete Messläufe in die Applikation speisen. Alle Sensoren welche mit der FPGA verbunden sind, werden durch einen FPGA-Feeder in die Application geleitet. Der FPGA-Feeder übernimmt die Initialisiereung der Kommunikation zwischen den Sensoren und derer Feedern. Sorgt also dafür dass die einzelnen Feeder den Sensor überwachen und Daten einziehen können. Das eigentliche Kommunikationsprotokoll der Sensoren ist in den Feedern implementiert. Speicherzuteilung sowie die Strukturierung der Outputdaten wird ebenfalls von den sensorpezifischen Feedern übernommen. Die Kamerafeeder sind zuständig für die Kommunikation mit den jeweiligen Kameras. Die Konfiguration dieser Feeder wird ebenfalls im XML vorgenommen. Sie starten die Aufzeichnung und extrahieren die Bilddaten aus den Kameras. Mit Aufzeichung ist hierbei gemeint, dass der Bildsensor zwar Bilder wahrnimmt aber nicht ausgelesen wird. Der Auslöser der Kameras wird durch ein Triggerevent der FPGA betätigt. Der Trigger zählt einen Zähler hoch, welcher dem ausgelöstem Bild angeheftet wird. Über die sensorspezifischen Feeder werden aufgenommene Daten in das Applikationsnetzwerk geleitet. Die eigentliche Datenverarbeitung, also zum Beispiel die Erzeugung von Trajektorien oder das Finden von Featurepaaren, findet hier statt. Zunächst werden die Bilddaten in den Stereofeeder

gegeben. Die Aufgabe des Stereofeeders ist es, Bilder mit gleichen Zeitstempel zusammen zu führen und mit den Metadaten des IPS-Geräts zu versehen. Die Metadaten umfassen beispielsweise die Basisweite des Geräts oder Brennweite der Objektive. Diese Daten sind für Folgefeeder wichtig, um weitere Verarbeitung durchzuführen. Die gepaarten Bilder werden in den Matcher geleitet, welcher das namensgebende Featurematching übernimmt. Er sucht markante Featurepaare. Diese werden in Form von Pixelkoordinaten extrahiert und in einer Liste an den Nachfolge-Feeder weitergegeben. Der nächste Feeder ist der Tracker. Er soll das Featuretracking übernehmen. Er verfolgt die Featurepaare in folge Bildern und errechnet aus deren Distanz zum IPS deren Bewegung. Zusätzlich zieht er IMU-Daten heran, um die errechnete Eigenbewegung zu bestätigen. Der letzte Feeder ist der Filterfeeder. In ihm werden die Daten aus IMU sowie aus dem Tracker zusammengeführt. Als Ergebnis erhält man die Trajektorie.
Ein anderer wichtiger Bestandteil der Systemumgebung des IPS ist die Konfigurationsdatein. Dabei handelt es sich, wie bereits erwähnt, um eine standardisierte XML-Datei. Mit Hilfe dieser Datei werden sämtliche Konfigurationen am IPS sowie auch der IPS-Applikation vorgenommen. Alle Metadaten der Feeder sind hierin enthalten. Es wird beispielsweise festgelegt, welche Kameras verwendet werden, deren Brennweite und andere Eigenschaften. Ebenfalls wird hierin die Datenverarbeitungsstrecke bestimmt sowie welcher Feeder darin enthalten ist und welcher nicht. Das gesamte Applikationsnetwerk ist definiert. Es ließe sich so zum Beispiel auch ein Netzwerk einrichten, welches nur Daten aufzeichnet diese aber nicht in eine Trajektorie überführt.

2.4.3 Daten

Die Daten der einzelnen Feeder werden in MAT-Datein abgespeichert. MAT-Dateien sind das Speicherformat von The Mathworks MATLAB. Es handelt sich bei denen vom IPS abgespeicherten MAT-Dateien um große Tabellen. Jede Zeile in dieser Datei stellt einen anderen spezifischen Feederwert dar. Jede Spalte stellt eine Aufnahme aus dem Feeder dar. Iteriert man durch die Spalten so erhält man den zeitlichen Verlauf der Feederdaten.
Eine Übersicht über den Inhalt der Zeilen sowie der Werteeinheit in diesen Zeilen wird bei jeder Aufnahme mit dem IPS automatisch generiert. Diese Übersicht nennt sich Mat-Format und wird als Textdatei abgespeichert. Für den Beschleunigungssensor des IPS sieht diese Mat-Format beispielsweise so aus:

- 1 |valid time |µs
- 2 |acquisition time |µs
- 3 |angular rate x|deg/s
- 4 |angular rate y|deg/s
- 5 |angular rate z|deg/s
- 6 |acceleration x|m/s²
- 7 |acceleration y|m/s²
- 8 |acceleration z|m/s²
- 9 |magnetometer x|gauss
- 10 |magnetometer y|gauss
- 11 |magnetometer z|gauss
- 12 |temperature |°C
- 13 |pressure |mBar
- 14 |delta angle x|deg
- 15 |delta angle y|deg
- 16 |delta angle z|deg
- 17 |delta velocity x|m/s
- 18 |delta velocity y|m/s
- 19 |delta velocity z|m/s

Die komplette Mat-Format-Datei findet sich im Anhang (A.4) Anhand dieser Textdatei lässt sich dann beispielsweise erkennen , dass sich der Zeitpunkt zu dem diese Spalte aufgenommen wurde in der ersten Zeile steht oder die Beschleunigung entlang der X-Achse in der sechsten Zeile steht. Auf die Daten wird in einem späteren Kapitel (Konzeption der Testfälle) genauer eingegangen. Die Daten der Stereokamera bilden allerdings eine Ausnahmen. Sie werden als rohe Bilddateien im TIFF-Format abgespeichert.

2.4.4 IPS-Testinfrastruktur

Die IPS-Testinfrastruktur ist onlinebasiert und über eine Website gebündelt, der IPS-Website. Diese bietet Benutzern eine Oberfläche zur Durchführung von Tests an der IPS-Software. Zugriff auf diese Website besteht abteilungsintern.

Auf dieser Website befinden sich Buttons mit diversen Funktionen. Mit Hilfe von CTest, einem Software-Testframework lassen sich Komponententests durchführen, deren

Abbildung 2.11: IPS-Website mit Buttons zur Testdurchführung

Ergebnisse sich mittels CDash visualisieren lassen. Softwaretests können manuell aber auch automatisch durchgeführt werden. Projektmanagement via Redmine wird ebenfalls untertützt. Eine weitere Funktion auf der IPS-Website ist das einsehen von Testergebnissen für Performancetests. Jede Nacht wird die IPS-Applikation neu kompiliert und die neueste Version wird zum Herunterladen bereitgestellt. Anschließen werden Benchmarkdaten genutzt, um die Genauigkeit der neuen Applikationsversion zu ermitteln. Ergebnisse vergangener Performancetests werden in Diagrammen visualisiert. So ist leicht einzusehen, welche Versionen bzw. Veränderungen an der Software positive oder negative Folgen hatten.

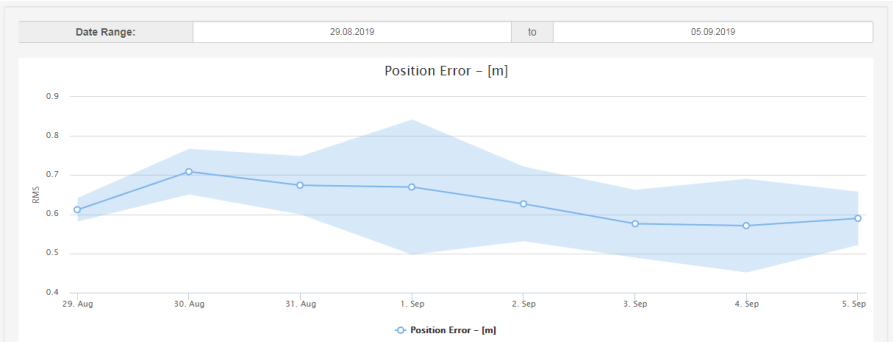

Abbildung 2.12: Visualisierung des Positionserrors von Versionen über ein Woche

3 Konzeption des HiL-Systems

Nachdem im vorherigen Kapitel die Eigenschaften von HiL-Systemen sowie die Spezifikationen des IPS und der IPS-Umgebung präsentiert wurden, soll in diesem Kapitel jenes Wissen dazu genutzt werden, um ein konkretes Konzept eines Hil-Systems für das IPS-Navigationssystem zu erstellen. Zunächst werden die Anforderungen an das HiL-System präsentiert und es wird abstrakt erklärt, wie diese zu erfüllen sind. Anschließen werden anhand der Analyseergebnisse der Systemumgebung sowie des IPS selbst die Lösungsansätze für die Anforderungen konkretisiert. Mit Hilfe der konkreten Lösungsansätze wird ein finales Konzept erstellt.

3.1 Anforderungen

Die ursprüngliche Intention eine HiL-Umgeung für das IPS zu entwickeln, liegt darin die Zuverlässigkeit des Geräts nach Veränderungen an Software oder Hardware beizubehalten bzw. zu erhöhen. Hierfür müssen Aspekte identifiziert werden, anhand derer die Zuverlässigkeit beurteilt werden kann. Diese Aspekte sind die Testfälle. Später wird auf diese Testfälle konkret eingegangen.
Zunächst sei aber gesagt, dass zur Überprüfung der Zuverlässigkeit eine Überprüfung der IPS-Daten nötig ist. Nach einem Updaten des Systems kann anhand der Qualität dieser Daten eine Bewertung über die Zuverlässigkeit des Gesamtsystems getroffen werden. Die Bewertung soll ähnlich stattfinden wie bei Performancetests auf der IPS-Website. Der Unterschied besteht darin, dass die HiL-Umgebung Daten aus einem realen IPS-System bezieht und keine Benchmarkdaten nutzt. Es wird also das Zusammenspiel von IPS und IPS-Applikaiton mit in den Test einbezogen.
Das Bewerten von Daten setzt voraus, dass die HiL-Umgebung in der Lage ist, Daten zu beziehen. Es muss eine sinnvolle Schnittstelle zum IPS-System gefunden werden, welche die eigentliche Geschäftslogik des Systems nicht beeinflusst. Schließlich soll das

Gesamtsystem in seiner vorgesehenen Funktion getestet werden. Zur Bewertung der Daten sind ebenfalls Anhaltspunkte nötig, an denen die Qualität bewertet werden kann. Das HiL-System benötigt Kenntnis darüber, wie korrekte beziehungsweise gewünschte Daten aussehen.
Eine weitere Anforderung an die HiL-Umgebung ist der Zugriff. Die HiL-Umgebung wird mit einem realen IPS-System verbunden. Sollten nun mehrere Benutzer HiL-Tests durchführen wollen, müsste entweder die Umgebung den Standort wechseln, um zum neuen Benutzer zu kommen oder der Benutzer müsste sich zum Ort der HiL-Umgebung begeben. Stattdessen soll es möglich sein die HiL-Umgebung abteilungsintern über das Netzwerk zu erreichen. Hierzu soll sie in die Testinfrastruktur eingebunden werden. Das beinhaltet ebenfalls dass die Umgebung auf dedizierter Hardware implementiert werden muss, welche über das Netzwerk erreichbar ist.
Zuletzt ist die Erweiterbarkeit und Modularität der Umgebung eine Anforderung. Wie bereits in den Grundlagen erwähnt, sollen Hil-Umgebungen nicht nur mit einer Version der Hardware , sondern ebenfalls mit modernisierten Versionen funktionieren.

3.2 Analyseergebnisse

In diesem Abschnitt werden Analyseergebnisse präsentiert. Diese Ergebnisse sollen Aufschluss darüber geben wie die Anforderungen an das HiL-System am besten umgesetzt werden können.

3.2.1 Systemumgebung

Als erstes wird die Systemanalyse durchgeführt. Ziel dieser Analyse ist es, Schnittstellen im IPS-System zu identifizieren, mit derer Hilfe Daten ausgelesen werden können. Die Grundlagen zur IPS-Applikation wurden im vorherigen Kapitel geklärt. Im Folgenden werden Möglichkeiten aufgezeigt, eine Schnittstelle zum IPS-System herzustellen.
Auf unterster Ebene, noch vor den Middleware-Feedern, kommuniziert das IPS mittels Treiber mit dem angeschlossenen Rechner. Es würde sich die Möglichkeit bieten, Daten direkt aus dem Treiben also direkt aus dem Gerät zu extrahieren. Der Vorteil hierin liegt dabei, dass es keinen direkteren Weg gäbe um an Daten zu kommen. Die Daten wurden durch keine Instanz berührt und sind absolut unverfälscht.

Allerdings liegt eben in dieser Eigenschaft auch ein Nachteil. Die rohen Sensorwer-

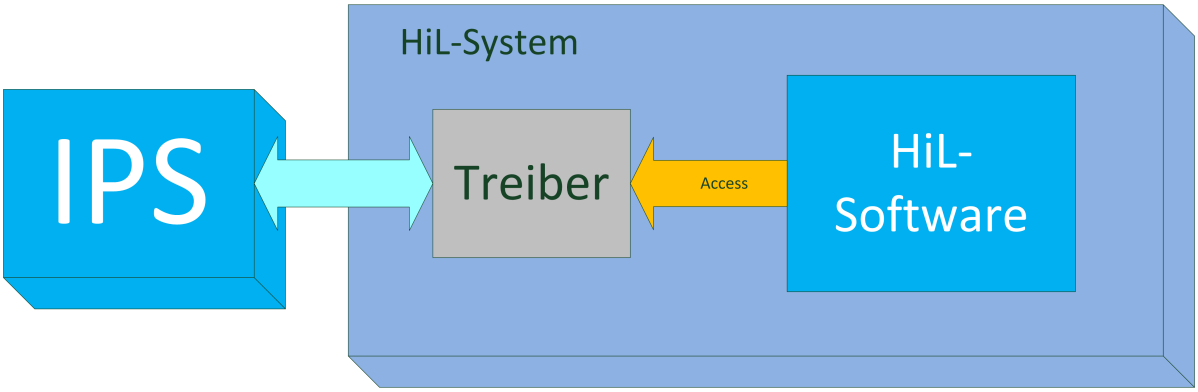

Abbildung 3.1: Visualisierung der Treiber-Methode

te müssen erst noch aufgearbeitet werden, bevor sie bewertet werden können, da die Middleware des IPS-System umgangen wird. Das Aufbereiten beinhaltet beispielsweise das Zusammensetzen der Werte aus verschiedenen Registern usw. Damit geht mit dieser Schnittstelle ein hoher programmatischer Aufwand einher, welcher den zeitlichen Rahmen der Bachelorarbeit sprengen würde. Der allerdings größte Nachteil und auch der Grund weshalb diese Methode ausscheidet ist das fehlende Zusammenspiel mit der IPS-Applikation. Die Applikation ist fester Bestandteil des IPS-Systems und soll beim HiL-Test ebenfalls mit einbezogen werden.

Eine weitere Möglichkeit wäre die Erweiterung des Application-Networks um einen weiteren Feeder, einen HiL-Feeder . Wie in den Grundlagen erwähnt, wird der Datenfluss innerhalb der Applikation durch die Konfigurationsdatei festgelegt. Mithilfe dieses XML-Files ließe sich festlegen, dass beliebige Feeder einen weiteren Output besitzen. Dieser Output könnte dann in den Feeder der HiL-Umgebung geleitet werden. Im HiL-Feeder werden dann die nötigen Tests durchgeführt.

Ein Vorteil dieser Methode ist, dass Daten bereits zur Laufzeit analysiert werden können. Auf diese Weise lässt sich dynmaisch auf Fehlerzustände eingehen. Fehler können sofort erkannt werden und es muss beispielsweise nicht auf die Beendigung der Aufnahme gewartet werden bevor Daten analysiert werden können. Die Tatsache, dass ein HiL-Feeder an jedem Punkt in der Verarbeitungskette der IPS-Applikation eingebunden werden kann, stellt einen weiteren Vorteil dar. So kann jeder Verarbeitungsschritt auf die Korrektheit kontrolliert werden. Die Nachteile dieser Methode sind ebenfalls von zeitlicher Natur. Einen HiL-Feeder zu implementieren würde vorraussetzen C++ für

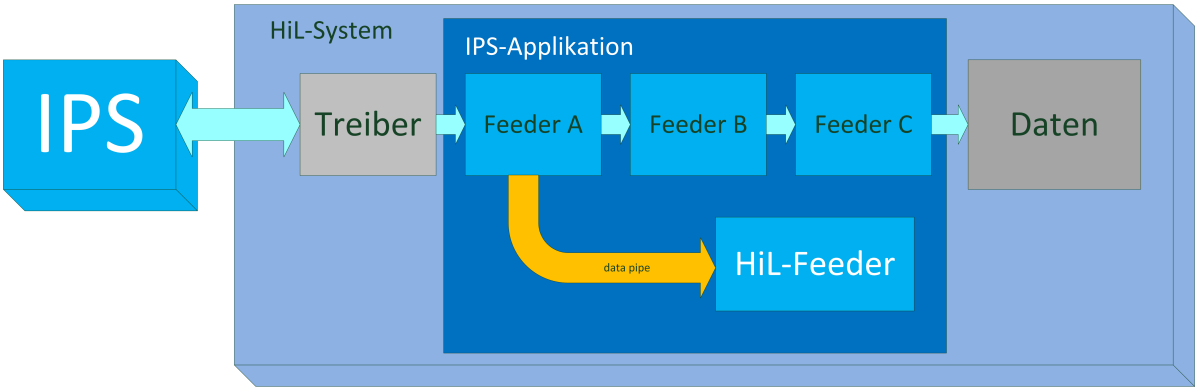

Abbildung 3.2: Visualisierung der Feeder-Methode

dessen Implementierung zu verwenden. Die Daten der Applikation werden in der Regel in MATLAB-Dateien gespeichert. Zwar ist die Handhabung von MATLAB-Dateien in C++ möglich aber benötigt wesentlich mehr Zeilen Code und stellt sich als wesentlich komplexer heraus als in anderen Programmiersprachen. Ein Beispiel hierzu findet sich im Anhang. Der C++-Code im Anhang ist um ein vielfaches länger sowie auch komplexer als der Vergleichscode in Python. Dementsprechend gibt es weniger zeitintensive Programmiersprachen als C++ zur Umsetzung.
Der größere Nachteil ist aber die Einrichtung der XML-Datei. Da das System nach Veränderung getestet werden soll, wird immer die neuste Version der Software geladen. Diese kommt mit einer neuen XML-Datei. In dieser Konfigurationsdatei ist ein Standardapplikationsnetzwerk definiert. Dieses beinhaltet die Aufnahme sowie Prozessierung von Daten. Der HiL-Feeder müsste bei jeder neuen Version automatisch in dieses XML eingefügt werden. XML-Dateien in C++ zu bearbeiten ist ebenfalls eine sehr komplexe Aufgabe. Aus zeitlichen Gründen wird auf diese Möglichkeit, eine Schnittstelle zu schaffen, verzichtet.

Die Daten vieler Feeder lassen sich, bevor diese in den folge Feeder geleitet werden, abspeichern. Der HiL-Test ist statischer Natur. Das bedeutet der Input variiert nicht. Der Input in diesem Fall ist die konstante Erdbeschleunigung sowie ein gleichbleibendes Kamerabild. Es gibt keinen dynamischen Input dessen Systemreaktion zur Laufzeit betrachtet werden muss. Dadurch wäre es möglich eine Aufnahme durchzuführen und diese anschließend auf Qualität zu überprüfen. Das heißt es wäre nur eine automatische Interaktion mit der IPS-Applikation nötig und eine Kontrolle der entstandenen Daten.

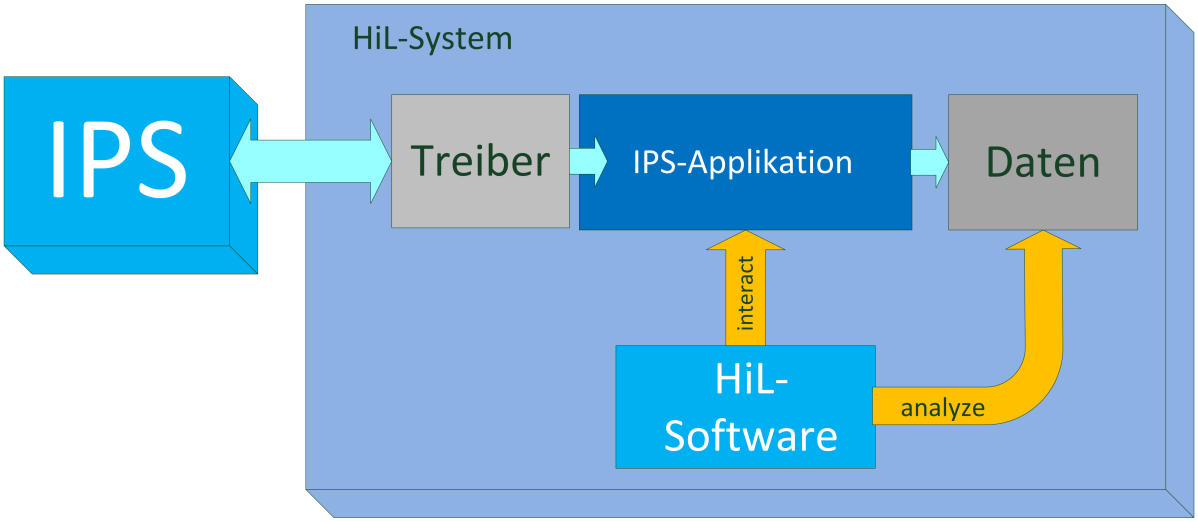

Abbildung 3.3: Visualisierung der Interaktions-Methode

Ein Vorteil der darin besteht ist der geringere Programmieraufwand im vergleich zu den vorherig aufgezeigten Möglichkeiten. Dieser Umfang wäre angebracht für den Projektzeitraum von 12 Wochen. Da ein statischer Test durchgeführt wird, ist diese Methode zur Systemüberprüfung angemessen. Aus diesen Gründen wird diese Methode zur Umsetzung des HiL-Systems in diesem Projekt genutzt.

3.2.2 Testinfrastruktur

Eine Anforderung ist es einen abteilungsweiten Zugriff auf die HiL-Umgebung zu gewährleisten. Gegebenenfalls sollen Tests automatisch nächtlich durchgeführt werden. Die erste Schlussfolgerung daraus ist, dass die Hardware der HiL-Umgebung im Netzwerk verfügbar sein muss.
Als Plattform auf welcher die Tests gestartet werden wird die IPS-Website genutzt. Von dort aus wird ein Signal an das HiL-System gesendet, welches diesem vermittelt einen Testlauf durchzuführen. Die Hardware der HiL-Umgebung wartet dazu kontinuierlich auf ein Signal der IPS-Website und führt dann den Testlauf durch. Die Ergebnisse des Testlaufs werden zurück an die IPS-Website gesendet und dort visualisiert.

3.2.3 Testfälle und Daten

Das IPS selbst erhält, wie bereits erwähnt, nur passiven input über sein Sensoren. Es existiert kein direkt steuerbarer Input für das Gerät. Bei dem Test, der durchgeführt werden soll, handelt es sich um einen statischen Test, es werden keine veränderlichen Daten eingespeist. Für dieses Testszenario bedeutet das, das IPS wird still an einer Stelle stehen und Daten produzieren. Aus diesem Grund ist es wichtig zu wissen, wie korrekte Daten beim Stillstand des Geräts aussehen.
Zudem ist es wichtig, Methoden zur Überprüfung von Testfällen zu verallgemeinern. Das bedeutet, mit einer Methodik mehrere Testfälle prüfen zu können. Dazu werden die Testfälle kategorisiert. Die Kategorien sind IMU-Testfälle, Kamera-Testfälle und Spezial-Testfälle. Im Kapitel Testmethodik wird erläutert, wie diese Unterteilung zu Stande kommt. Dieses Kapitel soll zunächst dazu dienen, konkrete Testfälle zu identifizieren. Zunächst sollen die Testfälle der Hauptkomponenten des IPS betrachtet werden. Die Hauptkomponenten sind IMU und Stereokamera. Die Daten der IMU werden, wie in den Grundlagen erklärt, von einem spezifischen Feeder für die IMU extrahiert und dann in einer MAT-Datein abgespeichert. In der MAT-Datei für die IMU befinden sich Werte für Beschleunigung entlang der x-, y- und z-Achse sowie der Drehrate um diese Achsen herum. Zusätzlich produziert die IMU Daten zur Temperatur, der Geschwindigkeitsdifferenz sowie der Drehratendifferenz zur vorherigen Aufnahme. Für jede Aufnahme wird ebenfalls der Zeitpunkt der Aufnahme abgespeichert. Alle Sensorwerte der IMU stellen einen testbaren Aspekt dar und sind somit Testfälle.
Die Kameradaten sind im TIFF-Format abgespeichert. Die Werte dieser Bilder sind in den Metadaten des TIFF-Bildes abgespeichert aus denen sie sich einfach extrahieren lassen. Die Werte umfassen unter anderem Auflösung des Bildes. Belichtungszeit oder ISO. Das sind die Testfälle für die Kamera

Da die zu überprüfenden Testfälle festgesetzt sind, muss nun noch ein Sollwert beziehungsweise ein Sollbereich dieser Werte festgelegt werden. Hierzu wird ein Messlauf analysiert, welcher etwa dem Testszenario entspricht. Es handelt sich um einen Messlauf, bei dem das IPS-Gerät zwar still steht, aber die Messung draußen vorgenommen wurde. Dieser Messlauf sollte die Temperaturveränderung außerhalb von Gebäuden bei gemäßigten Temperaturen zeigen. Da das Gerät dennoch still steht, lassen sich die Beschleunigungsdaten als Referenz nutzen.

Abbildung 3.4: Testaufbau des Referenztestlaufes

Erwartet wird bei dieser Analyse, dass alle Sensorwerte bei 0 verharren, da sich das Gerät nicht bewegt. Ausgenommen sind davon eine Beschleunigungsachse, welche die Erdbeschleunigung mit 1g beziehungsweise ca. 9,82 m/s^2, sowie die Temperatur welche sich im laufenden Betrieb des IPS verändert. Da die IMU ein rauschen aufweist, schwanken die Werte der Sensoren. Die Analyse ergab, dass alle Werte der IMU bei stillstand des Geräts ungefähr auf null bleiben mit Ausnahme einer Beschleunigungsachse ,welche die Erdbeschlaunigung misst. Das Rauschen des Sensors beeinflusst die Werte nur geringfügig. Bei der Beschleunigung schwanken die Werte um ca. 0.03 m/s^2. Bei der Drehrate schwanken die Werten um ca. 0.002 deg/s. Damit sind ist der Sollbereich für IMU-Testfälle festgelegt. Zur Überprüfung ist für alle Testfälle lediglich der Sollbereich sowie die Position in der MAT-Datein nötig. Aus diesem Grund lassen sich all diese Testfälle mit der selben Methodik überprüfen.
Die Sollwerte für die Kameradaten werden in der Konfigurationsdatei, dem XML, festgelegt. Zwar existieren keine Solldaten für den konkreten Inhalt der Bilder, aber die Metadaten der Bilder lassen sich dennoch überprüfen. Diese müssen nämlich mit den festgelegten Werten im XML übereinstimmen. Die Sollwerte der Testfälle im XML können mithilfe von Schlüsseln herausgelesen werden. Damit sind Testfälle für Kameradaten

sowie für IMU-Daten definiert. Alle Testfälle benötigen zur Überprüfung lediglich das Wissen über den XML-Schlüssel und können deshalb mit der selben Methodik überprüft werden.
Allerdings bedürfen Bilder noch weitere Überprüfung, um als nutzbar zu gelten. Zwar kann die Kontrolle der Auflösung erfolgreich sein, was aber nicht bedeutet ,dass dieses Bild zum Beispiel nicht nur schwarz ist. Zudem kann die Frequenz der Sensoren nicht direkt aus den Daten abgelesen werden. Sie muss errechnet werden. Diese Aspekte werden durch die Spezial-Testfälle abgedeckt. Bei den Spezial-Testfällen handelt es sich um Testfälle, welche nicht direkt einem Sensor zugeordnet werden können oder zu denen sich keine Methodik findet, mit der sich mehrere dieser Testfälle auf einmal testen lassen. Dazu zählt zunächst die Überprüfung der Frequenz der Sensoren, weil diese nicht direkt in den Daten abgespeichert ist, sondern anhand von Zeitstempel errechnet werden muss. Das äußert sich in der Überprüfungsmethodik. Des weiteren ist die Überprüfung auf Bildinhalt ein besonderer Testfall. Es wird kontrolliert, ob es sich um ein leeres Bild, also komplett schwarz, handelt. Sollte Bildinhalt vorhanden sein, kann der Featurezähler überprüft werden, da es sich hierbei um keinen eigenen Sensor handelt, aus dem diese Daten bezogen werden, ist das ebenfalls ein besonderer Testfall. Der letzte spezielle Testfall ist das korrekte Hochfahren des Geräts. Zur Sicherstellung der korrekten Initialisierung der IPS-Hardware, muss der Konsolenoutput der IPS-Applikation betrachtet werden. Hier werden Statusmeldungen ausgegeben, anhand derer sich der Zustand des Systems bestimmen lässt . Dabei ist zwischen fünf Zuständen zu unterscheiden. Leider lässt sich nicht bei allen die konkrete Ursache bestimmen.

- Die Applikation startet UND IPS ist aus -> Konsole: ïmu not connected"
- Die Applikation startet UND IMU ist nicht verbunden -> Konsole: ïmu not connected"
- Die Applikation startet UND Kamera ist nicht verbunden -> Konsole: "cam (left/-right) not connected"
- Die Applikation läuft UND Kamera wird getrennt -> Konsole: "cam time out"; beendet IPS-Applikation
- Die Applikation läuft UND IMU wird getrennt -> App friert ein und muss beendet werden

3.3 Konzeption der IPS-HiL-Testumgebung

In diesem Abschnitt sollen die Erkenntnisse aus den Analysen Anwendung finden. Sie sollen zu Erstellung von Architekturkonzepten, Testmethodiken sowie eines konkreten Testablaufs genutzt werden. Es geht darum ein konkretes Konzept für ein HiL-System für das IPS-Navigationssystem zu bilden

3.3.1 HiL-Infrastruktur

An dieser Stelle müssen die Möglichkeiten im DLR abgeschätzt werden. Zunächst wird die Auswahl der Hardware für das HiL-System betrachtet. Die Hardware, welche in den wissenschaftlichen Grundlagen als Beispiel für etablierte Technologien genannt wurde, ist nicht geeignet für die Verwendung im DLR. Das liegt einmal an dem horrenden Preis solcher System so wie auch an der Komplexität der verfügbaren Schnittstellen im Vergleich zu denen des Testsubjekts. Das IPS benötigt lediglich eine USB sowie eine Ethernetschnittstelle. Dementsprechend hat ein großer Tower wie dSpace ihn beispielsweise anbietet, zu umfangreich für die Verwendung mit dem IPS. Ein weiteres Problem könnte zudem die Lieferzeit darstellen.
Aus diesen Gründen wird für die Hardware ein gewöhnlicher Desktop-PC genutzt. Dieser wird mit einer zusätzlichen Netzwerkkarte ausgestattet. So kann er gleichzeitig mit dem Intranet des DLR sowie mit der Ethernetschnittstelle des IPS verbunden sein. Eine USB-Schnittstelle verfügt dieser ohnehin. Der Rechner wird dann mit einer statischen IP im Intranet angeschlossen.
Die Software des HiL-Systems wird in Python geschrieben. Ein Grund dafür ist Handhabung von MAT-Datein. Das Bearbeiten von MAT-Dateine gestaltet sich in Python sehr einfach (siehe Anhang A.2.2). Zusätzlich existieren bereits Python-Module zur Interaktion mit der IPS-Applikation die in diesem Projekt Wiederverwendung finden können.

3.3.2 Testszenario

Das Testszenario stellt den Ablauf des HiL-Tests dar. Anhand dieses Ablaufs soll festgelegt werden, über welche Funktionalitäten die Software verfügen muss, um einen HiL-Test durchzuführen. Folgender Abschnitt bezieht sich auf Abbildung 3.5.

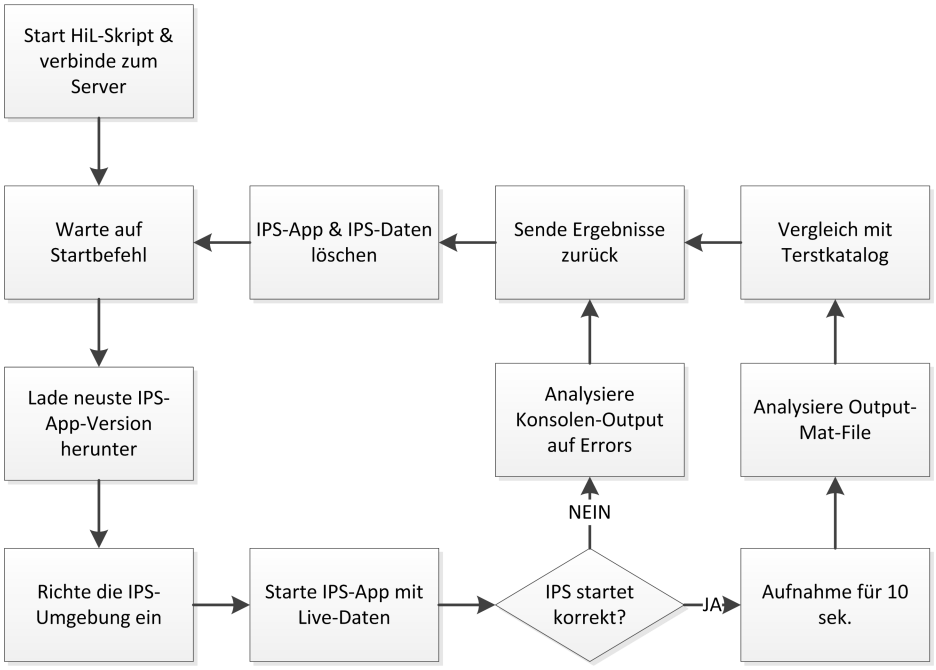

Abbildung 3.5: Testablauf eines HiL-Systems für das IPS

Der Einstiegspunkt für den Testlauf ist das Starten des Pythonskripts auf der HiL-Hardware. Startet das Skript, soll es sich mit dem zentralen Server der IPS-Website verbinden. Bei erfolgreicher Verbindung fängt das Skript an auf ein eingehendes Kommando zu warten.
Wenn nun ein Benutzer auf der IPS-Website den jeweiligen Button betätigt, wird der Startbefehl an die HiL-Umgebung versandt. Hiermit wird der eigentliche Testlauf ausgeführt.
Zunächst lädt die HiL-Umgebung die neuste Version der IPS-Applikation herunter. Hierzu wird der Umgebung zusammen mit dem Startkommando ebenfalls ein Link übertragen. Anschließend wird die IPS-Applikation entpackt und an einer individualisierbaren Stelle im Dateisystem abgelegt. Dann werden die nötigen Vorkehrungen zur Einrichtung der IPS-Umgebung getroffen. Das beinhaltet die Lizenzaktivierung und die Verwendung einer korrekten Konfigurationsdatei.
Wurden diese Schritte ohne Fehler ausgeführt, startet die IPS-Applikation. Es wird die Konsolenversion der Applikation verwendet. Die Daten werden wie es in HiL-Testverfahren vorgesehen ist aus realer Hardware bezogen. Diese werden dann von der IPS-Applikation, wie in der Konfigurationsdatei festgelegt, verarbeitet.

Zunächst analysiert die HiL-Umgebung allerdings den Output der IPS-Applikation. Es soll kontrolliert werden ob das IPS korrekt initialisiert wurde. Hierzu sucht die Umgebung in der Standaartausgabe der Konsole nach spezifischen Phrasen, welche auf einen Fehlerzustand deuten. Falls die Konsole nach geraumer Zeit keinen Output generiert wird sie beendet. Geschieht das, wird der Testlauf beendet und es wird ein Fehlercode zurück an den Benutzer der HiL-Umgebung gesendet. Dieser erhält den Fehler auf der IPS-Website. Anschließend geht das HiL-System wieder in den wartenden Zustand über.

Sollte beim Hochfahren des IPS kein Fehler auftreten, so wird eine Aufnahme gestartet. Die Länge dieser Aufnahme beträgt 10 Sekunden. Für den statischen Test sind 10 Sekunden völlig ausreichend, das keine große Veränderung der Daten erwartet wird. Nach 10 Sekunden beendet sich die Applikation. Die IPS-Applikation hat Daten erzeugt und die HiL-Umgebung geht dazu über, diese zu überprüfen.
Zur Analyse der Daten erhält die HiL-Umgebung eine Liste mit Testfällen. Diese Liste wird ebenfalls über das Netzwerk an die HiL-Umgebung übertragen. Es wird über die Liste iteriert und kontrolliert, ob die jeweiligen Testfälle erfolgreich sind. Zur Überprüfung werden die jeweiligen Testmethodiken genutzt. Die Einträge in den Listen werden um den Teststatus ergänzt. Dieser ist entweder positiv oder negativ.
Hinterher wird die HiL-Umgebung in ihren Ursprungszustand gebracht. Aufgezeichnete Datensätze sowie die heruntergeladene Version der IPS-Applikation müssen gelöscht werden.
Im letzten Schritt wird die Liste der Testfälle zurück an den Benutzer auf der IPS-Website gesendet. Dort werden die Ergebnisse visualisiert, sodass der Benutzer diese für sich auswerten kann. Danach geht die HiL-Umgebung wieder in den wartenden Zustand über. Der Loop wurde somit geschlossen.

3.3.3 Testmethodik

Zweck dieses Abschnittes ist es, Methodiken zu entwickeln, mit denen sich möglichst viele Testfälle überprüfen lassen. So soll ein hoher Programmieraufwand vermieden werden.Aus diesem Grund werden Gemeinsamkeiten der Testfälle gesucht.
Die gemessenen IMU-Daten befinden sich in MAT-Dateien (Abschnitt 2.4.3) . Jeder Testfall besitzt eine konkrete Nummer. Diese Nummer repräsentiert die jeweilige Zeile in der MAT-Datei. Zudem müssen alle Werte der IMU mit einem Sollbereich überprüft

werden. Grund dafür ist das Rauschen des Sensors.
Aus diesen Tatsachen ergibt sich folgende Testmethodik. Es gibt eine Kontrollfunktion, welcher die Nummer des jeweiligen Testfalls beziehungsweise der jeweiligen Zeile der MAT-Datei und der Sollbereich des Testfalls übergeben werden. Diese Funktion iteriert durch die Spalten der MAT-Datei und überprüft, ob alle Werte im angegebenen Bereich liegen. Mit dieser Funktion lassen sich nicht nur Werte der IMU sondern aller MAT-Dateien überprüfen. Es muss dazu nur der Dateipfad zur MAT-Datei, auf die Zugegriffen wird, spezifiziert werden. Die Funktion wäre also wiederverwendbar.
Diese Methodik bringt einige Vorteile mit sich. Die Implementierung der HiL-Umgebung benötigt keinerlei Kenntnis über die konkreten Testfälle. Das heißt, sie muss nicht wissen, in welcher Zeile welche Werte liegen oder in welchem Bereich sich diese zu befinden haben. Die Anzahl der Testfälle und der jeweilige Sollbereich kann extern gesetzt werden. Das bringt Erweiterbarkeit mit sich. Die Parameter des Sollbereichs können gegebenenfalls auch dynamisch verändert werden. Zudem bietet das die Möglichkeit via Webinterface Werte zu setzen.

Das Testen von Testfällen der Kamera gestaltet sich etwas anders. Die Werte der Bilddaten weisen kein Rauschen auf. Beispielsweise sollte die ISO aller Bilddaten konstant sein. Gleich verhält es sich mit der Auflösung der Bilder. Daraus folgt, dass die im XML festgelegten Parameter mit denen in den Metadaten der finalen Bilder übereinstimmen müssen. Es ergibt sich folgende Methodik.
Eine Kontrollfunktion erhält den Pfad zum jeweiligen XML sowie eine Liste mit den Namen der XML-Schlüssel. Anhand dieser Schlüssel liest die Funktion die Sollwerte aus der Konfigurationsdatei aus. Anschließend iteriert sie über alle Bilder die bei dem Testlauf aufgenommen wurden und überprüft deren Metadaten.
Der Vorteil in dieser Methodik liegt wie bei den IMU-Daten auch darin, dass die HiL-Umgebung keinerlei Wissen über die eigentlichen Parameter benötigt. Sie erhält den Namen eines Schlüssels und extrahiert den Sollwert aus der Konfigurationsdatei. Mit Hilfe dieser Methode ließen sich ebenfalls die Parameter anderer Feeder überprüfen und mit deren Resultaten abgleichen.

Ein weiterer wichtiger Aspekt zum Testen ist das korrekte initialisieren des Testsubjekts, dem IPS. Hierbei handelt es sich um einen Spezial-Testfall.
Wie im vorherigen Abschnitt angedeutet, wird hierzu die Standardausgabe der IPS-

Applikation überprüft. Zeile für Zeile wird überprüft ob eine Phrase enthalten ist welche auf einen Hardwarefehler hindeutet. Das funktioniert für vier der fünf aufgezeigten Fälle . Für den Fall das die IMU wärend des Messlaufs die Verbindung verliert muss ein Timeout gesetzt werden, nachdem die IPS-Applikation geschlossen wird. Der Grund dafür ist, dass die Applikation bei diesem Fehler einfriert und sich nicht wie vorgesehen nach 10 Sekunden beendet. Ein Alternative zu dieser Methode wäre es den Output der Konsolenapplikation zuspeichern und erst zu überprüfen, wenn nach der Beendigung der Applikation keine sinnvollen Daten existieren. Problematisch könnte die Methode werden, wenn die Konsolenapplikation einfriert.

Die Überprüfung der Frequenz von IMU sowie Stereokamera ist ebenfalls ein Spezial-Testfall. Hierzu wird der Zeitstempel von IMU beziehungsweise der Stereokamera genutzt. Die Bilddaten erhalten ihren Zeitstempel als Dateinamen und die IMU-Daten verfügen über diesen in der ersten Zeile der jeweiligen MAT-Datei. Die Einheit der Zeitstempel ist Microsekunde. Zu Bestimmung der Frequenz wird immer die Differenz zweier Aufeinanderfolgender Aufnahemen genommen. Anschließend wird geschaut welche Frequenz sich aus daraus ergibt. Die Zeitstempel sind immer der Zeitpunkt des Auslösens einer Aufnahme. Die Auslöse Frequenz kann allerding variieren, weshalb es nötig ist die Frequenz zwischen allen Bilder zu betrachten. Die Frequenz bewegt sich deshalt auch in einem Sollbereich.

3.3.4 Softwarearchitektur

Um eine Softwarearchitektur für die HiL-Umgebung zu konzipieren, wird das Testszenario betrachtet. Es ist zu identifizieren, welche Verantwortlichkeiten in welchen Softwaremodulen zusammen zu fassen sind. Dieser Vorgang wird unternommen, um die Komplexität des Testlaufs auf elementare Aufgaben herunter zu brechen. Zusätzlich wird darauf geachtet, die Modularität der Software zu gewährleisten. Eine Komponente hat nur eine Aufgaben. Ebenfalls soll Wiederverwendbarkeit gegeben sein. Die Komponenten sollen gegebenenfalls ihren Zweck auch außerhalb dieses Projekts erfüllen können. Eine Visualisierung der Softwarearchitektur ist in Abbildung 3.6 zu finden.

Der Testablauf ist im Endeffekt die prozedurale Aneinanderreihung von Aufgaben. Diese lassen sich grob wie folgt unterteilen:

- Serververbindung einrichten und lauschen
- IPS-Umgebung einrichten
- Mit IPS-Konsolenapplikation interagieren
- Daten auswerten/ Testmethodiken anwenden
- IPS-Umgebung säubern
- Ergebnisse versenden

Diese Aufgaben sollen nun einzelnen Komponenten zugeordnet werden. Klassifizieren lassen sich diese Aufgaben in 4 Bereiche:

- Internetverwaltung
- Dateioperationen
- Bedienen einer Konsolenanwendung
- Testmethodiken ausführen

Mit dieser Einteilung wurde eine Teilung der Verantwortung vorgenommen und es ergeben sich daraus die vier im Folgenden beschriebenen Komponenten.

Connection Manager Bei der ersten Komponente handelt es sich um den Connection Manager. Dieser ist für die Internetverwaltung zuständig. Er stellt gewissermaßen die Main-Funktion der HiL-Umgebung dar. Der Connection Manager ist für die Kommunitkation mit der IPS-Testinfrastruktur zuständig. Er richtet die Verbindung zum Server der IPS-Website ein und sendet Ergebnisse zurück. Desweiteren nimmt er die Testfälle und den Downloadlink entgegen, welche mit eingehendem Startkommando übertragen werden. Er übergibt diese an die zuständigen Komponenten und startet den eigentlichen Testlauf. Hierarchisch ist der Connection Manager am höchsten angesiedelt.
Der eigentliche Testlauf wird von den drei darunter angeordneten Komponenten durchgeführt.

Application Deployer Er ist für Interaktion mit dem Dateisystem zuständig. Das bedetet er läd die neuste IPS-Applikation herunter, richtet deren Umgebung ein undsetz die HiL-Umgvebung anschließend wieder zurück.

IPS Application Interface Diese Komponente ist für die Interaktion mit der Konsolenanwendugn zuständig. Sie soll Aufnahmen starten und Aufnahmen durch die IPS-Applikation prozesieren lassen.

HiL-Tester In der Letzten der drei Komponenten werden die konkreten Testmethodiken angewendet. Hier sind die Kontrollfunktionen implementiert und es werden konkrete Testfälle anhand der IPS-Daten durchgeführt. Der HiL-Tester gibt die Ergebnisse der Testfälle zurück an den Connection Manager.

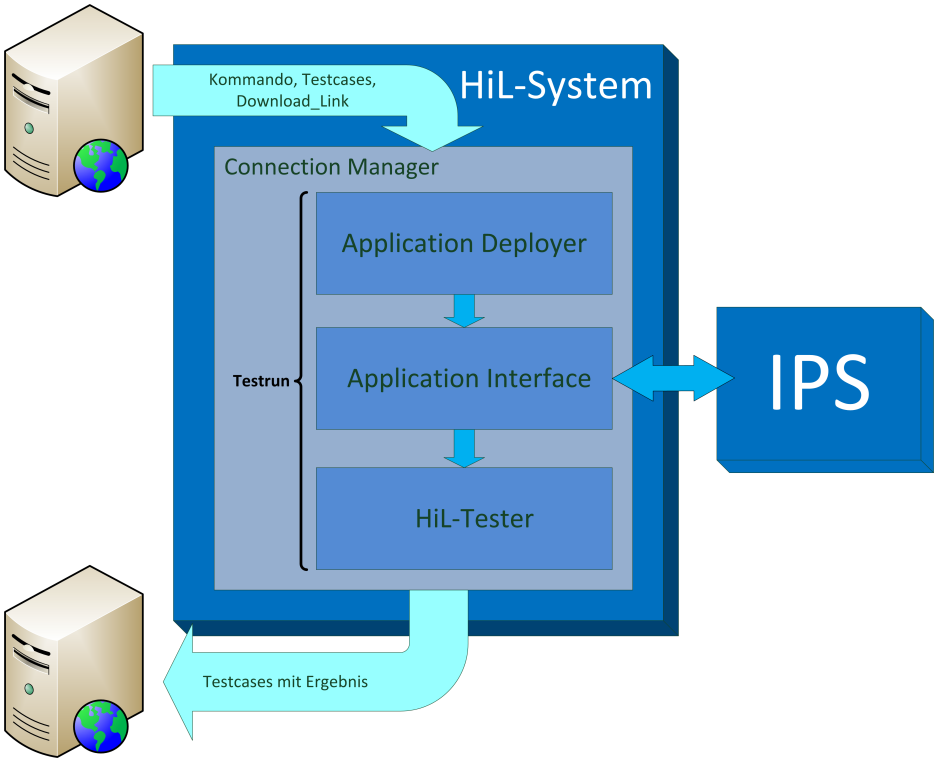

Abbildung 3.6: Visualisierung des Softwarearchitekturkonzepts

4 Implementierung der IPS-HiL-Testumgebung

Dieses Kapitel beschäftigt sich mit der konkreten Umsetzung der aufgestellten Konzepte. Es wird beschrieben, wie die Softwarekomponenten miteinander agieren und das Testszenario umsetzen.

4.1 Connection Manager

Der Connection Manager ist dafür zuständig die HiL-Umgebung bei Start mit dem Server zu verbinden und auf das Startkommando zu warten.
Hierzu initialisiert der Connection Manager einen TCP-Socket und verbindet sich mit dem Server. Anschließend geht er in einen ewigen While-Loop über. Innerhalb des While-Loops wird die Recieve-Funktion des Sockets (data = s.recv(1024)) aufgerufen. Diese blockiert den Prozess bis eine Nachricht ankommt. Wird eine Nachricht empfangen, überprüft der Connection Manager deren Inhalt auf das Startkommando(if data == “command”). Enthält die empfangene Nachricht besagtes Kommando, blockiert der Prozess erneut und wartet auf die Übertragung eines Python-Dictionaries. Dieses Dictionary enthält die Testfälle für zu überprüfende Sensoren sowie einen Downloadlink für Software.

```
1 testcases_imu = [{"name" :"temp",   "nbr" : 11,  "range" :[9.5 , 23.5],
            "tolerance": 2},
2                {"name" :"acc Z",  "nbr" : 7,   "range" :[9.6 , 9.97]}]
```

Die Testfälle der IMU sind Listen aus Dictionaries. Der erste Schlüssel gibt den Namen des Testfalls an. Der Zweite zeigt an welcher Stelle im MAT-File dieser zu finden ist und der Dritte in welchem Bereich sich der Wert zu befinden hat. Optional kann auch eine Fehlertoleranz als vierter Schlüssel mitgegeben werden. Die Toleranz gibt den

prozentualen Anteil an fehlerhaften Werten im Verhältnis zur Gesamtheit aller Werte an.

```
testcases_cam = [{"name" : "height", "xml_key": "height"}]
```

Die Testfälle für die Kamera gestalten sich simpler. Hier muss lediglich der Name des XML-Schlüssels, unter dem der Sollwert zu finden ist, übergeben werden.

```
download_link = "http://ipsmain.dhcp.ws.ba.dlr.de/release/
    nightly_builds/ips_2019-08-07_sfslib_ver5.0.4.zip"
download_dest = ".\\release"

#Dictionary welches vom Connection manager empfange wird
jsondict = {
    "testcases_imu": testcases_imu,
    "testcases_cam": testcases_cam,
    "download_link": download_link,
    "download_dest": download_dest
}
```

Mit diesen Parametern wird die einzige Funktion des Connection Managers aufgerufen: start_testrun().

start_testrun(dload_link, dload_dest, testcases_imu, testcases_cam): Diese Funktion führt den gesamten Testlauf durch. In ihr werden die anderen drei Komponenten initialisiert. Diesen wird der für sie bestimmte Parameter übergeben. Anschließend gehen sie ihrer Aufgabe nach. Als Rückgabewert hat diese Funktion die Ergebnisse des HiL-Testlaufes. Diese Ergebnisse sind erneut in Form einer Liste auf Testfällen gegeben. Allerdings wurde jeder Testfall um einen weiteren Schlüssel ergänzt. Dieser gibt den Erfolg des Testfalls an.

```
testcases_imu = [{"name" :"temp",   "nbr" : 11,  "range" :[9.5 , 23.5],
        "tolerance": 2, "status": "Successful"},
                {"name" :"acc Z",  "nbr" : 7,   "range" :[9.6 , 9.97],
    "status": "Failed"}]
```

Diese Liste wird anschließend zurück an den Server versendet.

4.2 Application Deployer

Der Application Deployer ist dafür zuständig die Systemumgebung für das IPS-System einzurichten und im Anschluss an den Testlauf wieder aufzuräumen. Dazu muss er die neueste Version der Software herunterladen und diese an einer bekannten Stelle entpacken. Der Applicaiton Deployer ist eine Klasse, die zur Initialisierung den Downloadlink und das Downloadzielverzeichnis benötigt. Diese beiden Werte werden innerhalb des initialisierten Objekts abgespeichert.
Er besitzt nur zwei Funktionen. Zum einen die Funktion deploy() und zum anderen die Funktion cleanUp().

deploy(): Diese Funktion erhält keine Übergabeparameter. Es wird die Request-Library von Python genutzt, um eine Datei von einer URL zu laden. Diese Datei wird dann mit Hilfe des open()-Operator abgespeichert. In diesem Fall handelt es sich um ein komprimiertes Verzeichnis, welches die IPS-Software enthält. Danach wird die Zipfile-Bibliothek genutzt, um die heruntergeladene Datei im Zielverzeichnis zu entpacken. Als Rückgabewert hat deploy() ein Dictionary, welches den Namen des heruntergeladenen XMLs und den Pfad zur IPS-Software enthält.

cleanUp() Diese Funktion erhält keine Übergabeparameter. Es wird lediglich der gesamte Inhalt der Downloadzielverzeichnisses sowie die komprimierte Datei der Software gelöscht.

4.3 Application Inteface

Das Application Interface ist dafür zuständig mit der IPS-Software zu kommunizieren und diese zu bedienen. Es nutzt dazu drei Funktionen: record(), process() und call_ips_app().

record(do_sessions = []) Die Zuständigkeit dieser Funktion ist es Live-Messläufe mit dem IPS und der dazugehörigen Software aufzuzeichnen. record() erhält als Übergabeparameter eine Liste mit Nummern. Diese Nummer gibt den Namen der aufzuzeichnenden Messläufe an. Sollte keine Liste übergeben werden, nutzt record() eine leere Liste. Zunächst überprüft record(), ob die Messläufe, welche aufgezeichnet werden sollen,

bereits existieren. Ist dies der Fall werden diese nicht erneut aufgezeichnet und aus der Liste der aufzunehmenden Sessions gestrichen. Bei einer leeren Liste nimmt die Funktion die nächst größere freie Nummer. Anschließend stellt record() eine Parameterliste zusammen. Diese Parameterliste enthält den Befehl zum Aufzeichnen eines Messlaufs. Mit dieser Liste wird die Funktion call_ips_app() aufgerufen. Der Messlauf endet automatisch nach 10 Sekunden. Durch die Konfiguration im XML werden die aufgezeichneten Läufe direkt prozessiert. Anschließend schaut record() welche Messläufe dazugekommen sind und gibt die Pfade der prozessierten Läufer als Rückgabewert aus.

process(do_sessions = []) process() macht genau das gleiche wie record() mit dem Unterschied, dass keine Livedaten bezogen werden, sondern auf bereits aufgezeichnete Messläufe zurückgegriffen wird. Diese werden dann lediglich prozessiert. process() gibt ebenfalls die Pfade der prozessierten Läufe zurück.

call_ips_app(param_list) Diese Funktion hat den Zweck einen Subprozess mit der übergebenen Parameterliste zu starten. Sie gibt zusätzlich den Output dieses Subprozesses aus. Dieser Output wird zeitgleich auf Fehlermeldungen der IPS-Software kontrolliert und abgebrochen, wenn ein Fehlerzustand identifiziert worden sein sollt.

4.4 Hardware-in-the-Loop Tester

Der HiL-Tester führt die Testfälle durch. Zur Initialisierung werden ihm der Dateipfad zu den IPS-Daten übergeben, der Name des XMLs, sowie die zu testenden Aufnahmesessions. Diese Parameter werden innerhalb des HiL-Tester-Objekts gespeichert. Für Dateipfad und XML verfügt der Tester über Setter-Funktionen. Die auszuführenden Testfälle werden nicht zur Initialisierung übergeben, sondern per Setter-Funkton gesetzt.
Der HiL-Tester verfügt nur über eine einzige Funktion, execute_all().

execute_all() Diese Funktion ist dazu da alle Testfälle durchzuführen. Sie erhält keine Eingabeparameter. Die nötigen Werte kann sich die Funktion durch privaten Zugriff innerhalb des Testers besorgen. Werden keine Sessions spezifiziert, so führt execute_all() die Überprüfungen für alle Sessions durch. Die eigentliche Testmethodik ist

allerdings nicht im HiL-Tester implementiert. Hierzu existieren 2 untergeordnete Objekte: Imu_tester, Cam_tester. Jede dieser Objekte verfügt über eine Funktion namens execute(). Diese Funktion wird innerhalb von execute_all() für alle drei Objekte ausgeführt. Als Rückgabewert hat execute_all() eine Liste aller Testfälle welche, um das Ergebnis derer Überprüfung ergänzt wurden.

4.4.1 Imu_tester

Der Imu_tester ist für die Testfälle bezüglich des Inertialsensors zuständig. Zur Initialisierung werden der Pfad zur betreffenden MAT-Datein und der Testkatalog für die IMU übergeben. Zusätzlich wird die MAT-Datein anhand des Pfades geöffnet. Der IMU_tester besitzt zwei Funktionen: execute() und check_value().

execute() Execute() führt die Testmethodik für jeden Testfall durch. Die Funktion iteriert durch den Testkatalog, entnimmt die Parameter der jeweiligen Testfälle und übergibt sie an die Testmethodik für IMU-Daten. Die Testfälle im Testkatalog werden durch execute() mit dem Ergebnis der Testmethodik ergänzt. Der Rückgabewert von execute() ist der ergänzte Testkatalog.

check_value(val_range, case_nbr, tolerance = 0) Hierbei handelt es sich um die Implementierung der Testmethodik. Sie erhält den Sollbereich des Testfalls sowie die Zeilennummer der MAT-Datei, in der sich der zu messende Wert befindet. Optional kann der Methode eine Toleranz übergeben werden. Dabei handelt es sich um den prozentualen Anteil an Werten außerhalb des Sollbereichs, welcher noch als korrekt angesehen wird. Der Standardwert für die Toleranz ist null. Check_value() iteriert über die jeweilige Zeile der MAT-Datein und notiert alle Werte außerhalb des Bereichs. Anschließend wird überprüft, ob die Toleranz eingehalten wurde. Abhängig hiervon wird dann der Status des Testfalls gesetzt.

4.4.2 Cam_tester

Der Cam_tester übernimmt die Überprüfung von Kamera-Testfällen. Um diesen zu initialisieren wird der Pfad zur Konfigurationsdatei und der Kamera-Testkatalog benötigt. Er verfügt wie der IMU_tester über die Funktionen execute() und checkValue(). Die execute()-Funktion des Cam_tester macht praktisch das gleiche wie die des

IMU_testers, mit dem Unterschied, dass die Testmethodik der Kamera ausgeführt wird und der Testkatalog der Kamera um Ergebnisse ergänzt und zurückgegeben wird. Aus diesem Grund wird auf einen Paragraphen für execute() verzichtet.

check_value(xml_key) Wie beim IMU_tester ist check_value die Implementierung der Testmethodik. Im Falle des Cam_tester wird allerdings nur der XML-Schlüssel für den zu überprüfenden Wert übergeben. Die Funktion sucht den konkreten Wert für den Testfall anhand des XML-Schlüssel aus der Konfigurationsdatei heraus. Dann wird dieser Wert dem Testfall im Testkatalog hinzugefügt. Es handelt sich dabei um den Sollwert.

```
1 testcases_cam = [{"name" : "height", "xml_key": "height"}] #before
     reading xml
2 testcases_cam = [{"name" : "height", "xml_key": "height", "value": "600
     px"}] # after
```

Anschließend wird durch alle Bilder iteriert und kontrolliert ob die Werte in den Metadaten des jeweiligen Bildes denen des Sollwerts entsprechen.

5 Evaluierung und Ausblick

Das Ziel dieser Arbeit war es, eine HiL-Umgebung für das IPS-Navigationssystem zu entwickeln. In diesem Kapitel soll bewertet werden, wie die Arbeitsschritte zu diesem Ziel abgeschlossen wurden. Zudem wird beschrieben inwiefern die Implementierung der HiL-Umgebung ihrem Zweck nachkommt. Anschließend liefert der Ausblick einen Eindruck auf Erweiterungsmöglichkeiten dieses Projekts.

5.1 Evaluation

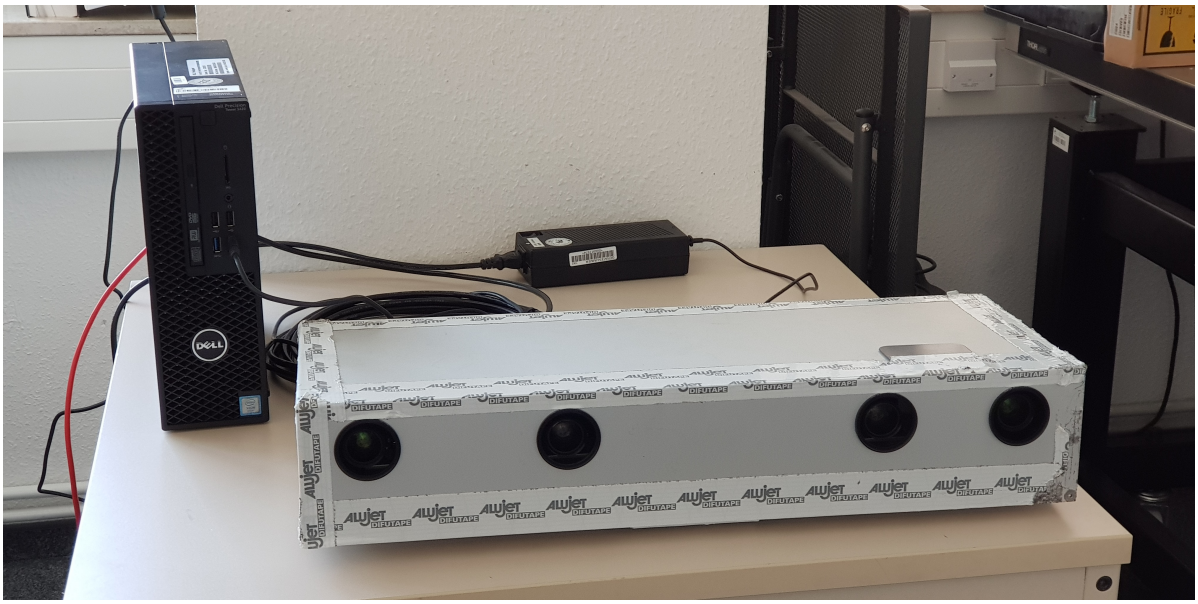

Abbildung 5.1: Aufbau der HiL-Umgebung

Die Schritte, welche im Abschnitt 'Arbeitsplan' erläutert wurden, sind alle erfolgreich abgeschlossen. Die Entwicklung einer HiL-Umgebung für das IPS umfasst eine Auseinandersetzung mit HiL-Systemen im Allgemeinen. Es ist anhand der gesammelten Erkenntnisse möglich, HiL-Systeme in den Kontext von Testverfahren und in den

Entwicklungszyklus einzuordnen. Es ist bekannt, welche Eigenschaften ein HiL-System besitzen muss um seinen Zweck zu erfüllen und wie Anforderungen an das HiL-System gestellt werden müssen. Außerdem wurden bereits etablierte Technologien betrachtet.
Die Testhardware wurde ausgiebig analysiert. Es wurden Hardware-Schnittstellen zum System identifiziert. Die Spezifikationen des IPS wurden genaustens betrachtet, um die HiL-Umgebung auf diese einstellen zu können. Die Untersuchung der Systemumgebung hat mehrere Eingliederungsmöglichkeiten in das IPS-System ergeben. An diesen Schnittstellen kann ein HiL-System eingebunden werden. Aus den aufgezeigten Eingliederungsmöglichkeiten konnte anhand des zeitlichen Aufwandes die am meisten geeignetste dieser Möglichkeiten ausgewählt werden.
Das Betrachten des Referenzmesslaufs und dessen Daten hat es ermöglicht, testbare Aspekte am IPS zu identifizieren und daraus konkrete Testfälle abzuleiten. Weitere Testfälle konnten zudem aus den gewonnenen Kenntnissen der Hardwareanalyse gebildet werden.
Die Betrachtung der DLR-Testinfrastruktur hat es ermöglicht, Einbindungsmöglichkeiten zu finden.
Die zuvor gewonnenen Erkenntnisse wurden genutzt um ein konkretes Konzept einer HiL-Umgebung für das IPS-Navigationssystem zu erstellen.Zunächst wurde ein Testszenario konzipiert. Die Schritte dieses Testszenarios konnten klassifiziert werden und so helfen, Softwarekomponenten zu bilden. Anhand der Recherche zu HiL-Technologien in der Wirtschaft sowie der Analyse der DLR-Testinfrastruktur konnte eine dedizierte Hardware für die zu entwickelnde HiL-Umgebung gefunden werden.
Die aufgestellten Konzepte konnten erfolgreich umgesetzt werden. Softwarekomponenten, welche zuvor vom Testszenario abgeleitet wurden, konnten zielführend implementiert werden. Die Funktion der Komponenten wurde durch Integrationstests bestätigt. Die ausgewählte Hardware für die Umgebung konnte in das Intranet des DLR eingebunden werden und ist dort erreichbar. Das IPS kann mit dieser Hardware kommunizieren und die implementierte Software ist darauf lauffähig.
Ebenfalls wurde eine Bestätigung des Gesamtkonzepts durchgeführt. Hierzu wurde ein Ende-zu-Ende Test mit realer Testhardware auf der dedizierten Hardware durchgeführt. Ein einzelner Testfall konnte erfolgreich via TCP-Socket an die HiL-Umgebung gesendet werden. Die Umgebung konnte diesen Testfall mit dem erwarteten Ergebnis abschließen und das Ergebnis zurücksenden.
Jede Komponente der HiL-Umgebung wurde einmal durchlaufen. Das System ist funk-

tionsfähig.

```
#Testfall des End-to-End Test
testcases_imu = [{"name" :"temp",   "nbr" : 11,  "range" :[9.5 , 23.5],
        "tolerance": 2}]
```

```
============================EXECUTING ALL TESTS==============================
Executing all tests for session:  0005
---------------------- executing testcase:  temp -------------------------
!!!! some data values are outside of the tolerated range, test case No:  11
tolerated percentage of values out of range:  2
actual percentage of values out of range:  38.738552368555624
values outside of range:  499983
================================RESULTS=====================================
[[{'name': 'temp', 'nbr': 11, 'range': [9.5, 23.5], 'tolerance': 2, 'status': 'failed'}]]
```

Abbildung 5.2: Ergebnis des End-To-End Tests

Nun soll genauer auf das Softwarekonzept eingegangen werden und ob dieses zielführend umgesetzt wurde.
Das Testszenario wurde mit allen Komponenten vollständig abgedeckt. Die Ausführung des Testlaufs bei Empfangen eines definierten Kommandos ist möglich. Die Übertragung des Testkatalogs und eines Downloadlinks wurde ebenfalls gewährleistet. Zudem werden die Ergebnisse des Testlaufs wieder an den Server versendet. Durch das JSON-Format der Nachricht können diese dort beliebig weiterverarbeitet werden. Der Connection Manager ist damit funktionstüchtig.
Der Downlaodlink kann an den Application Deployer übergeben werden. Dieser richtet die Umgebung der IPS-Applikation zuverlässig ein und ist im Anschluss an den Testlauf ebenso fähig die entstandenen Daten wieder zu löschen und die HiL-Umgebund wieder in den Ursprungszustand zu bringen. Der Application Deployer erfüllt seinen Zweck.
Mit Hilfe des IPS Application Interface können entweder Messläufe mit einem realen IPS-System durchgeführt oder bereits vorhandene Aufnahmen mit dem Applikationsnetzwerk der IPS-App prozessiert werden. Das Application Interface ist modular konzipiert, sodass es nicht zwangsweise in diesem Projekt verwendet werden muss.
Der HiL-Tester ist in der Lage die ihm übergebenen Testfälle durchzuführen. Es wird ein korrektes Ergebnis zurückgeliefert. Die Testfälle können anhand von Parametern nachträglich eingestellt werden. Die Implementierung des HiL-Testers ist absolut unabhängig vom Format der MAT-Dateien oder der XML-Dateien, die überprüft werden. Alle nötigen Informationen werden im Testkatalog übertragen. Es wurde erreicht, dass

mehrere Testfälle mittels derselben Testmethodik überprüft werden können. Damit sind alle Anforderungen an die Softwarekomponenten erfüllt.

An dieser Stelle soll auf die Anforderungen an die HiL-Umgebung im Allgemeinen eingegangen werden. Zunächst sollte Zuverlässigkeit des IPS sichergestellt werden. Das wird durch Überprüfung von Daten aus ein angeschlossenem IPS-System erreicht. Hierzu wurde eine Schnittstelle zum IPS-System identifiziert und genutzt. Diese Anforderung ist somit erfüllt.
Des Weiteren war abteilungsweiter Zugriff auf die HiL-Umgebung eine der Anforderungen. Diese wurde durch Nutzung einer dedizierten Hardware gewährleistet,welche im Netzwerk verfügbar ist. Zudem wurde durch die strukturierte Form des Testkataloges und der Ergebnisse aus einem HiL-Test eine saubere Schnittstelle geschaffen, mit der andere Entwickler arbeiten können. Mithilfe von TCP-Sockets wurde die HiL-Umgebung im Netzwerk verfügbar gemacht. Zwar ist sie im Netzwerk verfügbar, aber nicht in die IPS-Website eingebunden. Die HiL-Umgebung ist eine Konsolenanwendung und kann per Fernzugriff gestartet werden.
Die letzten Anforderungen waren Erweiterbarkeit und Modularität. Mit der vorhin erwähnten sauberen Schnittstelle ist Erweiterbarkeit gegeben. Modularität wird durch die Trennung von Zuständigkeiten der Softwarekomponenten gewährleistet. Abgesehen vom Special_tester konnten alle Komponenten implementiert werden Beinahe alle Anforderungen an die HiL-Umgebungen wurden erfüllt.

5.2 Ausblick

Da Erweiterbarkeit und Modularität gegeben sind, bietet das Projekt Potential zum Ausbau. Wie bereits erwähnt, ist die Implementierung der HiL-Umgebung unabhängig vom eingesetzten IPS-Gerät und unabhängig vom Format der Daten. Testfälle werden von außen an die Umgebung gegeben. Dadurch können diese nachträglich angepasst werden. Es wäre möglich individuelle Testkataloge zu entwerfen und diese durchzuführen.
Durch die einheitliche Strukturierung der Ergebnisse wurde eine klar definierte Schnittstelle zur weiteren Verarbeitung der Daten geschaffen. Man könnte die IPS-Website dahingehend erweitern, dass Testfälle vom Benutzer definiert werden können und dass die Ergebnisse ähnlich wie bei Performancetests visualisiert werden.

Eine weitere Erweiterungsmöglichkeit ist das dynamische Einstellen der Umgebung auf

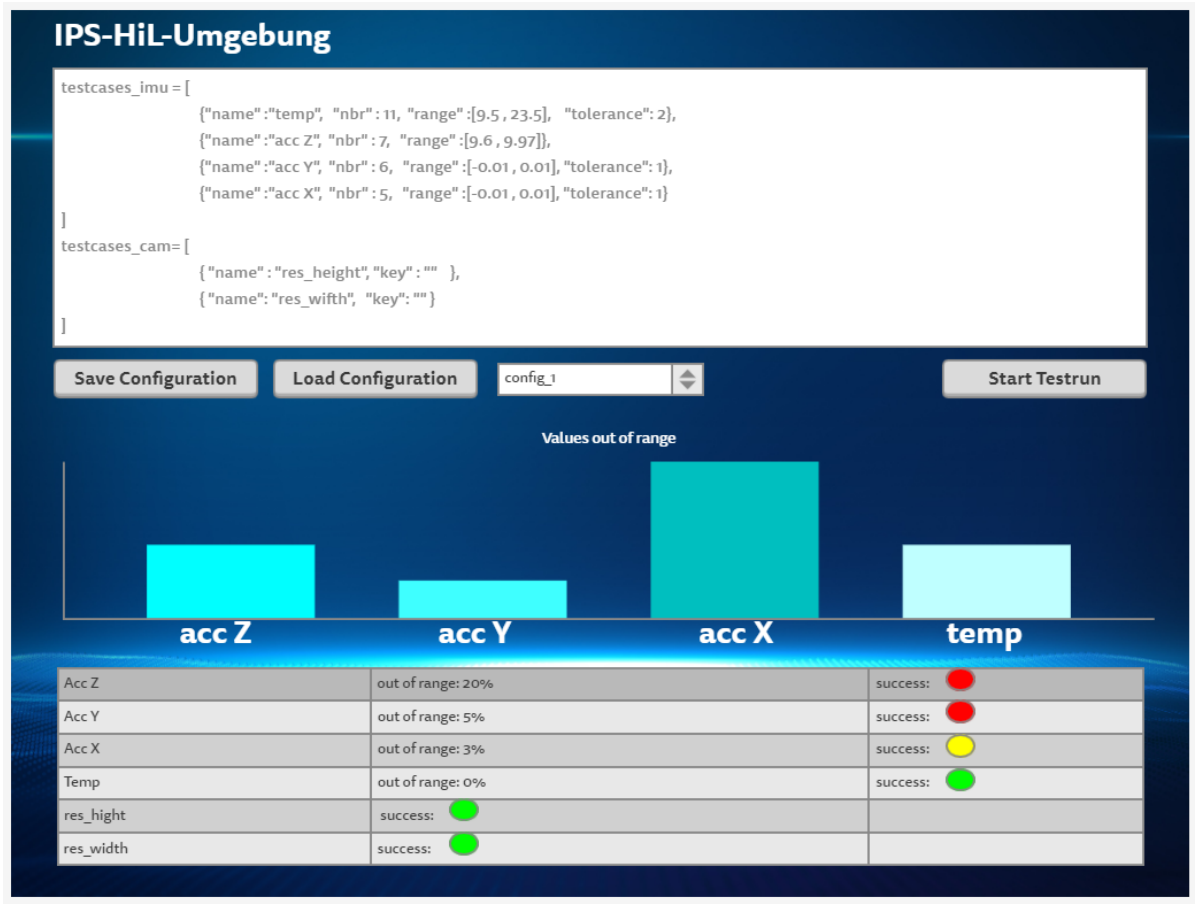

Abbildung 5.3: Mockup der HiL-Umgebung auf der IPS-Website

unterschiedliche IPS-Versionen. Zwar ist der Gebrauch nur mit einer dieser Versionen vorgesehen, aber es wäre möglich die XML-Dateien automatisch auf ein neues Gerät anzupassen.

Literaturverzeichnis

[1] Anko Börner, Dirk Baumbach, Maximilian Buder, Andre Choinowski, Ines Ernst, Eugen Funk, Denis Grießbach, Adrian Schischmanow, Jürgen Wohlfeil, and Sergey Zuev. IPS-a vision aided navigation system. *Advanced Optical Technologies*, 6(2):121–129, 2017.

[2] celestair. `https://www.celestaire.com/product/astra-iii-professional-sextant/`.

[3] dSpace. `https://www.dspace.com/en/pub/home/products/hw/simulator_hardware/dspace_simulator_full_size.cfm`.

[4] Denis Grießbach. *Stereo-Vision-Aided Inertial Navigation.* PhD thesis, Freie Universität Berlin, 2015.

[5] Lawrence J Hettinger and Gary E Riccio. Visually induced motion sickness in virtual environments. *Presence: Teleoperators & Virtual Environments*, 1(3):306–310, 1992.

[6] IMC. Vorteile des hardware-in-the-loop verfahrens. `https://www.imc-tm.de/loesungen/allgemeine-loesungen/simulation-hil/vorteile-von-hil/`; accessed: 12.09.2019.

[7] National Instruments. What is Hardware-in-the-Loop. pages 1–3, 2019. `https://www.ni.com/de-de/innovations/white-papers/17/what-is-hardware-in-the-loop-.html`; accessed: 12.09.2019.

[8] Kevin Asthon. That ' Internet of Things ' Thing. *RFID Journal*, page 4986, 2010.

[9] C Kleijn. Introduction to Hardware-in-the-Loop Simulation. In *Control Lab., 7521 AN Enschede.* 2014.

[10] Bedford A Lampkin and Robert J Randle. Investigation of a manual sextant-sighting task in the ames midcourse navigation and guidance simulator. 1965.

[11] Peter H Lindsay and Donald A Norman. *Human information processing: An introduction to psychology.* Academic press, 2013.

[12] Daniel Lorenz. HIL basing calibration on basis of HAWKS racing car. 2008.

[13] Pratap Misra and Per Enge. Global positioning system: signals, measurements and performance second edition. *Global Positioning System: Signals, Measurements And Performance Second Editions,*, 2006.

[14] Roman Obermaisser and Wilfried Elmenreich. A Framework for Hardware-in-the-Loop Testing of an Integrated Architecture Martin. Number June. 2014.

[15] Steven Peters, Jung-Hoon Chun, and Gisela Lanza. Digitalization of automotive industry–scenarios for future manufacturing. 2015.

[16] Hongmou Zhang and Doktor Der Ingenieurwissenschaften. *Optical Navigation for Mobile Platforms Based on Camera Data.* PhD thesis, Technische Universiät Berlin, 2018.

A Anhang

A.1 Triangulation

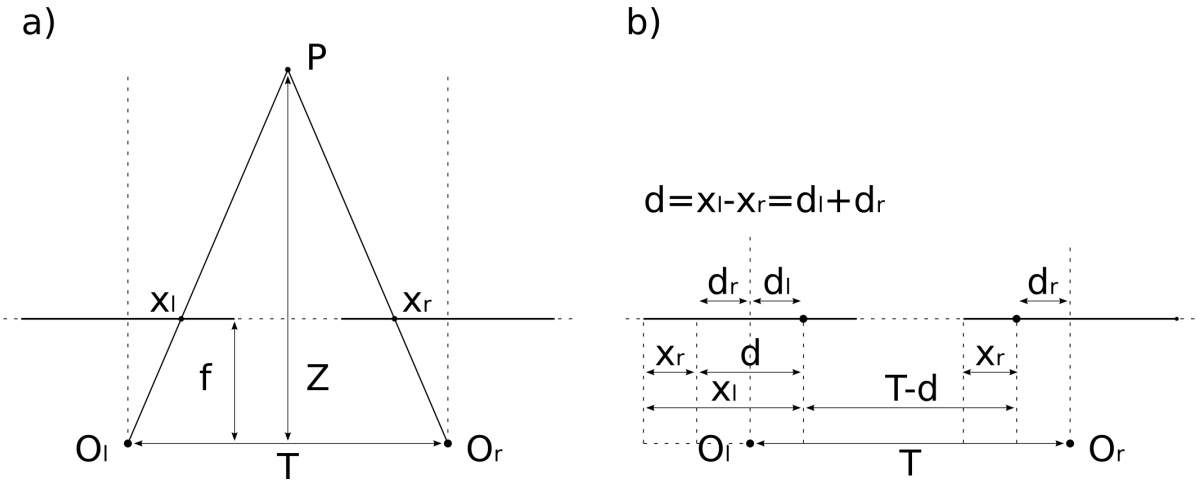

Abbildung A.1: Triangulation mithilfe des Strahlensatzes

$$\frac{Z}{T} = \frac{h}{b} \tag{A.1}$$

$$h = Z - f \tag{A.2}$$

$$b = T - d \tag{A.3}$$

$$d = dL + dR \tag{A.4}$$

$$\frac{Z}{T} = \frac{Z - f}{T - d} \tag{A.5}$$

Formel nach Z umgestellt:

$$Z = T \cdot \frac{Z - f}{T - d} \tag{A.6}$$

A.2 Laden von Matfiles

A.2.1 CPP CODE FÜR DAS EINLESEN EINER MAT-DATEI

```
1
2  #include "mat.h"
3  #include <iostream>
4  #include <vector>
5  void matread(const char *file, std::vector<double>& v)
6  {
7      // open MAT-file
8      MATFile *pmat = matOpen(file, "r");
9      if (pmat == NULL) return;
10
11     // extract the specified variable
12     mxArray *arr = matGetVariable(pmat, "LocalDouble");
13     if (arr != NULL && mxIsDouble(arr) && !mxIsEmpty(arr)) {
14         // copy data
15         mwSize num = mxGetNumberOfElements(arr);
16         double *pr = mxGetPr(arr);
17         if (pr != NULL) {
18             v.reserve(num); //is faster than resize :-)
19             v.assign(pr, pr+num);
20         }
21     }
22
23     // cleanup
24     mxDestroyArray(arr);
25     matClose(pmat);
26 }
27
28 int main()
29 {
30     std::vector<double> v;
31     matread("data.mat", v);
32     for (size_t i=0; i<v.size(); ++i)
```

```
33          std::cout << v[i] << std::endl;
34      return 0;
35  }
```

A.2.2 PYTHON CODE FÜR DAS EINLESEN EINER MAT-DATEI

```
1  mat_path = "path_to_your_mat_file"
2  mat_contents = sio.loadmat(mat_path)
3  for i in mat_contents:
4      print(i)
```

A.3 Trajektorien

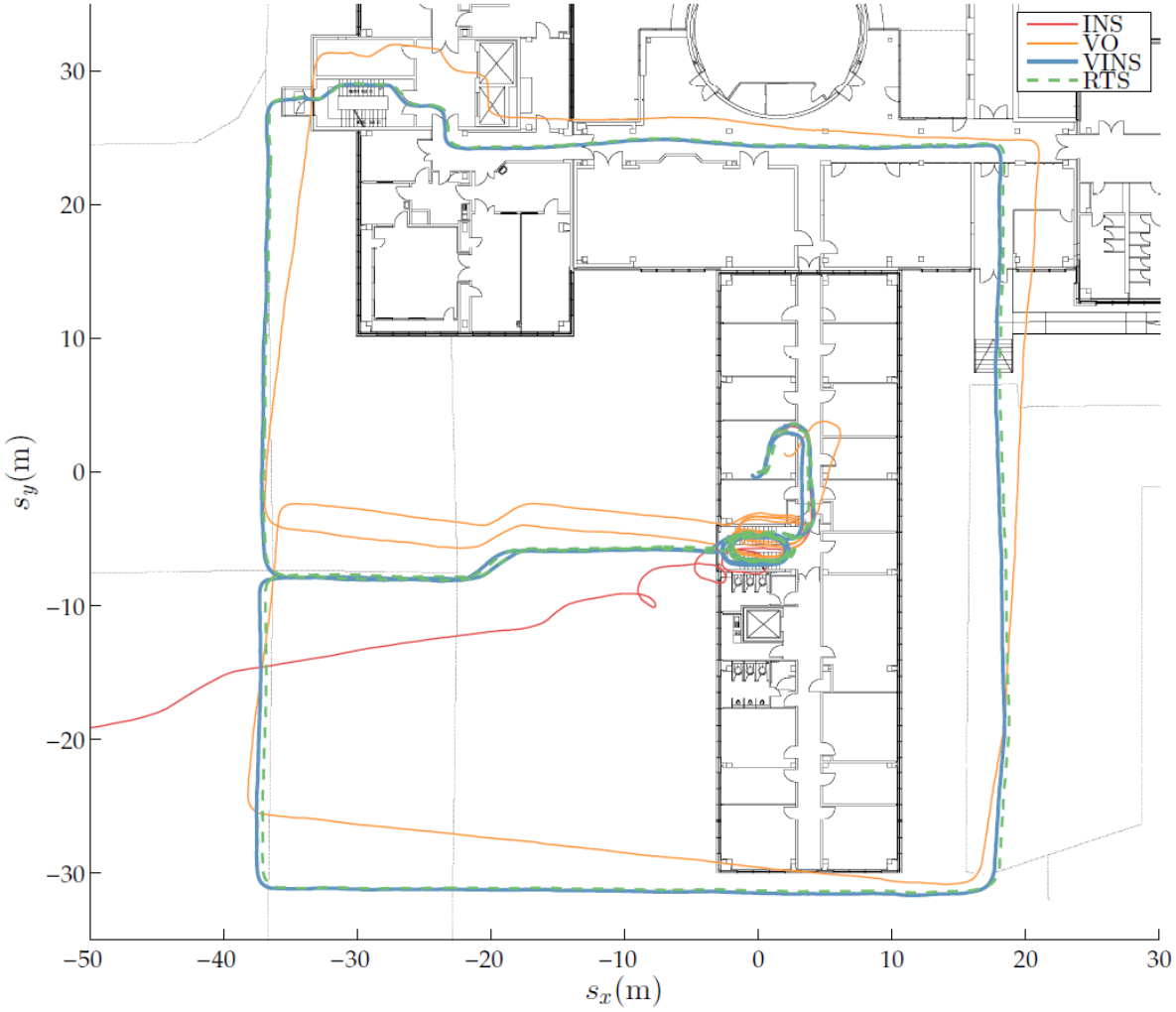

Abbildung A.2: Trajektorien mit Messfehler von oben

A.4 Mat-Format

triggerOut

---

1 |valid time |µs
2 |acquisition time |µs
3 |frame count |
4 |trigger edge(1=fall/2=rise) |
5 |trigger edge frame count |

gpsUblox

---

1 |valid time |µs
2 |acquisition time |µs
3 |position latitude|deg
4 |position longitude|deg
5 |position altitude ELL|m
6 |position altitude MSL|m
7 |positionStd latitude|m
8 |positionStd longitude|m
9 |positionStd altitude|m
10 |heading of motion |deg
11 |heading accuracy |deg
12 |speed over ground |m/s
13 |GNSS fix type |3:=3D fix
14 |heading valid flag |1:=valid
15 |Velocity North |m/s
16 |Velocity East |m/s
17 |Velocity Down |m/s
18 |horizontal accuracy |m
19 |vertical accuracy |m
20 |speed accuracy |m/s
21 |satellites used |
22 |time of week |ms

23 |UTC time |hhmmss.sss
24 |year (UTC) |yyyy
25 |month (UTC) |mm
26 |day (UTC) |dd
27 |pDOP |
28 |FPGA clock drift |usec

tiltADIS

---

1 |valid time |µs
2 |acquisition time |µs
3 |inclination x|deg |57.2958
4 |inclination y|deg |57.2958
5 |rotation angle z|deg |57.2958
6 |acceleration x|g
7 |acceleration y|g
8 |temperature |°C
9 |z-axis direction(1=up/-1=down) |

imuADIS16488

---

1 |valid time |µs
2 |acquisition time |µs 3 |angular rate x|deg/s |57.2958
4 |angular rate y|deg/s |57.2958
5 |angular rate z|deg/s |57.2958
6 |acceleration x|m/s^2
7 |acceleration y|m/s^2
8 |acceleration z|m/s^2
9 |magnetometer x|gauss
10 |magnetometer y|gauss
11 |magnetometer z|gauss
12 |temperature |°C
13 |pressure |mBar
14 |delta angle x|deg |57.2958

15 |delta angle y|deg |57.2958
16 |delta angle z|deg |57.2958
17 |delta velocity x|m/s
18 |delta velocity y|m/s
19 |delta velocity z|m/s

CamLeft

---

1 |valid time |µs
2 |acquisition time |µs
3 |frame count |
4 |exposure time |ms
5 |gain |dB
6 |timestamp |usec
7 |logger capacity |

CamRight

---

1 |valid time |µs
2 |acquisition time |µs
3 |frame count |
4 |exposure time |ms
5 |gain |dB
6 |timestamp |usec
7 |logger capacity |

stereoBS

---

1 |valid time |µs
2 |acquisition time |µs

imageScalerStereo2

---

1 |valid time |µs

2 |acquisition time |µs
3 |frame count |

stereoMatcher

---

1 |valid time |µs
2 |acquisition time |µs
3 |featureCount extract|
4 |featureCount intra|
5 |featureDistAvg z|m

tracker

---

1 |valid time |µs
2 |acquisition time |µs
3 |feature count interLL|
4 |feature count interRR|

visualOdometry

---

1 |valid time |µs
2 |acquisition time |µs
3 |delta quaternion scalar part w|
4 |delta quaternion vector part x|
5 |delta quaternion vector part y|
6 |delta quaternion vector part z|
7 |delta position x|m
8 |delta position y|m
9 |delta position z|m
10 |delta quaternion/position cov (1,1)|
11 |delta quaternion/position cov (2,1)|
12 |delta quaternion/position cov (3,1)|
13 |delta quaternion/position cov (4,1)|
14 |delta quaternion/position cov (5,1)|

15 |delta quaternion/position cov (6,1)|
16 |delta quaternion/position cov (7,1)|
17 |delta quaternion/position cov (1,2)|
18 |delta quaternion/position cov (2,2)|
19 |delta quaternion/position cov (3,2)|
20 |delta quaternion/position cov (4,2)|
21 |delta quaternion/position cov (5,2)|
22 |delta quaternion/position cov (6,2)|
23 |delta quaternion/position cov (7,2)|
24 |delta quaternion/position cov (1,3)|
25 |delta quaternion/position cov (2,3)|
26 |delta quaternion/position cov (3,3)|
27 |delta quaternion/position cov (4,3)|
28 |delta quaternion/position cov (5,3)|
29 |delta quaternion/position cov (6,3)|
30 |delta quaternion/position cov (7,3)|
31 |delta quaternion/position cov (1,4)|
32 |delta quaternion/position cov (2,4)|
33 |delta quaternion/position cov (3,4)|
34 |delta quaternion/position cov (4,4)|
35 |delta quaternion/position cov (5,4)|
36 |delta quaternion/position cov (6,4)|
37 |delta quaternion/position cov (7,4)|
38 |delta quaternion/position cov (1,5)|
39 |delta quaternion/position cov (2,5)|
40 |delta quaternion/position cov (3,5)|
41 |delta quaternion/position cov (4,5)|
42 |delta quaternion/position cov (5,5)|
43 |delta quaternion/position cov (6,5)|
44 |delta quaternion/position cov (7,5)|
45 |delta quaternion/position cov (1,6)|
46 |delta quaternion/position cov (2,6)|
47 |delta quaternion/position cov (3,6)|
48 |delta quaternion/position cov (4,6)|

49 |delta quaternion/position cov (5,6)|
50 |delta quaternion/position cov (6,6)|
51 |delta quaternion/position cov (7,6)|
52 |delta quaternion/position cov (1,7)|
53 |delta quaternion/position cov (2,7)|
54 |delta quaternion/position cov (3,7)|
55 |delta quaternion/position cov (4,7)|
56 |delta quaternion/position cov (5,7)|
57 |delta quaternion/position cov (6,7)|
58 |delta quaternion/position cov (7,7)|
59 |feature count |
60 |ransac count |
61 |reprojection error avg L0L1+R0R1|
62 |reprojection error std L0L1+R0R1|
63 |input feature count L0L1+R0R1|

navigationFilter

---

1 |valid time |µs
2 |acquisition time |µs
3 |quaternion scalar part w|
4 |quaternion vector part x|
5 |quaternion vector part y|
6 |quaternion vector part z|
7 |position x|m
8 |position y|m
9 |position z|m
10 |quaternion/position cov (1,1)|
11 |quaternion/position cov (2,1)|
12 |quaternion/position cov (3,1)|
13 |quaternion/position cov (4,1)|
14 |quaternion/position cov (5,1)|
15 |quaternion/position cov (6,1)|
16 |quaternion/position cov (7,1)|

17 |quaternion/position cov (1,2)|
18 |quaternion/position cov (2,2)|
19 |quaternion/position cov (3,2)|
20 |quaternion/position cov (4,2)|
21 |quaternion/position cov (5,2)|
22 |quaternion/position cov (6,2)|
23 |quaternion/position cov (7,2)|
24 |quaternion/position cov (1,3)|
25 |quaternion/position cov (2,3)|
26 |quaternion/position cov (3,3)|
27 |quaternion/position cov (4,3)|
28 |quaternion/position cov (5,3)|
29 |quaternion/position cov (6,3)|
30 |quaternion/position cov (7,3)|
31 |quaternion/position cov (1,4)|
32 |quaternion/position cov (2,4)|
33 |quaternion/position cov (3,4)|
34 |quaternion/position cov (4,4)|
35 |quaternion/position cov (5,4)|
36 |quaternion/position cov (6,4)|
37 |quaternion/position cov (7,4)|
38 |quaternion/position cov (1,5)|
39 |quaternion/position cov (2,5)|
40 |quaternion/position cov (3,5)|
41 |quaternion/position cov (4,5)|
42 |quaternion/position cov (5,5)|
43 |quaternion/position cov (6,5)|
44 |quaternion/position cov (7,5)|
45 |quaternion/position cov (1,6)|
46 |quaternion/position cov (2,6)|
47 |quaternion/position cov (3,6)|
48 |quaternion/position cov (4,6)|
49 |quaternion/position cov (5,6)|
50 |quaternion/position cov (6,6)|

51 |quaternion/position cov (7,6)|
52 |quaternion/position cov (1,7)|
53 |quaternion/position cov (2,7)|
54 |quaternion/position cov (3,7)|
55 |quaternion/position cov (4,7)|
56 |quaternion/position cov (5,7)|
57 |quaternion/position cov (6,7)|
58 |quaternion/position cov (7,7)|
59 |velocity x|m/s
60 |velocity y|m/s
61 |velocity z|m/s
62 |velocity cov (1,1)|
63 |velocity cov (2,1)|
64 |velocity cov (3,1)|
65 |velocity cov (1,2)|
66 |velocity cov (2,2)|
67 |velocity cov (3,2)|
68 |velocity cov (1,3)|
69 |velocity cov (2,3)|
70 |velocity cov (3,3)|
71 |true angular rate x|deg/s |57.2958
72 |true angular rate y|deg/s |57.2958
73 |true angular rate z|deg/s |57.2958
74 |true angular rate cov (1,1)|
75 |true angular rate cov (2,1)|
76 |true angular rate cov (3,1)|
77 |true angular rate cov (1,2)|
78 |true angular rate cov (2,2)|
79 |true angular rate cov (3,2)|
80 |true angular rate cov (1,3)|
81 |true angular rate cov (2,3)|
82 |true angular rate cov (3,3)|
83 |true acceleration x|m/s^2
84 |true acceleration y|m/s^2

85 |true acceleration z|m/s²
86 |true acceleration cov (1,1)|
87 |true acceleration cov (2,1)|
88 |true acceleration cov (3,1)|
89 |true acceleration cov (1,2)|
90 |true acceleration cov (2,2)|
91 |true acceleration cov (3,2)|
92 |true acceleration cov (1,3)|
93 |true acceleration cov (2,3)|
94 |true acceleration cov (3,3)|
95 |angular rate bias x|deg/s |57.2958
96 |angular rate bias y|deg/s |57.2958
97 |angular rate bias z|deg/s |57.2958
98 |angular rate bias cov (1,1)|
99 |angular rate bias cov (2,1)|
100|angular rate bias cov (3,1)|
101|angular rate bias cov (1,2)|
102|angular rate bias cov (2,2)|
103|angular rate bias cov (3,2)|
104|angular rate bias cov (1,3)|
105|angular rate bias cov (2,3)|
106|angular rate bias cov (3,3)|
107|acceleration bias x|m/s²
108|acceleration bias y|m/s²
109|acceleration bias z|m/s²
110|acceleration bias cov (1,1)|
111|acceleration bias cov (2,1)|
112|acceleration bias cov (3,1)|
113|acceleration bias cov (1,2)|
114|acceleration bias cov (2,2)|
115|acceleration bias cov (3,2)|
116|acceleration bias cov (1,3)|
117|acceleration bias cov (2,3)|
118|acceleration bias cov (3,3)|