Entwicklung eines Force Cueing Algorithmus für Sitzpolster Druckaktuatoren zum Einsatz im DLR Robotic Motion Simulator

Development of a force cueing algorithm for seat pad pressure actuators for use in the DLR Robotic Motion Simulator

Wissenschaftliche Arbeit zur Erlangung des Grades B.Sc.
an der Fakultät für Elektrotechnik und Informationstechnik der
Technischen Universität München.

Betreut von	Univ.-Prof. Dr.-Ing./Univ. Tokio habil. Martin Buss PD Dr.-Ing. habil. Marion Leibold Lehrstuhl für Steuerungs- und Regelungstechnik
Eingereicht von	Simon Dammann Pürgener Straße 28 86899 Landsberg am Lech 08191 - 9476544
Eingereicht am	München, den 02.10.2023

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
UNIV.-PROF. DR.-ING./UNIV. TOKIO HABIL. MARTIN BUSS
DR.-ING. MARION LEIBOLD

25. Mai 2023

B A C H E L O R A R B E I T
für
Simon Dammann
Mat.-Nr. 03739198, Studiengang EI

Entwicklung eines Force Cueing Algorithmus für Sitzpolster Druckaktuatoren zum Einsatz im DLR Robotic Motion Simulator

(**Development of a force cueing algorithm for seat pad pressure actuators for use in the DLR Robotic Motion Simulator**)

Problembeschreibung:

Flug oder Fahrsimulatoren nutzen unterschiedliche Wege, um einen Bewegungseindruck bei den Insassen zu erzeugen. Diese werden als Motion Cues bezeichnet. Als Beispiel hierfür können visuelle Cues durch die Visualisierung von Fenstern oder vestibulare Cues durch die Bewegungen einer Bewegungsplattform (z.B. Industrieroboter oder Hexapod) genannt werden. Ein weiterer Mechanismus zum Hervorrufen eines Bewegungseindrucks sind nonvestibulare Cues. Sogenannte G-Seats nutzen dabei verschiedene Aktuatoren wie z.B. Gurtstaffersysteme, Vibrationsaktuatoren, Druckpolster oder Klappensysteme [2]. G-Seats können ergänzend zum visuellen und vestibularen Motion Cueing eingesetzt werden. Entscheidend für die Qualität des Bewegungseindrucks ist der Motion Cueing Algorithmus (im Fall von G-Seats auch als Force Cueing bezeichnet). Dieser bestimmt die Aktuatorzustände (z.B. Gurtlänge, Druck im Druckpolster) aus den Zuständen des simulierten Vehikels. Es existieren unterschiedliche Ansätze für das Motion Cueing von G-Seats, wie z.B. Washout-Filter [3] oder Model Predictive Control [1].
Im DLR Robotic Motion Simulator (RMS) wird bisher visuelles und vestibulares Motion Cueing genutzt. Ergänzend sollen zukünftig auch Aktuatoren im RMS Cockpit Klappen in den Sitzpolstern bewegen um G-Kräfte zu simulieren. Dazu ist bereits ein Hardware Prototyp konstruiert. Dieser wird in den ersten Wochen der BA von Mitarbeitenden des DLR fertig aufgebaut. Die in den Aktuatoren genutzten Motoren und die zur Regelung notwendigen Drives sind bereits vorhanden und können ab Beginn der BA genutzt werden.

Leistungen:

- Literaturrecherche zu G-Seats und Motion/ Force Cueing Algorithmen
- Inbetriebnahme der Motoren des Hardwareprotoyps
- Entwicklung eines Sicherheitskonzepts für die Ansteuerung der Motoren (z.B. Momentenbeschränkung)
- Modellidentifikation des Aktuators
- Entwicklung eines Motion-Cueing Algorithmus für die G-Seat Aktuatoren

Literatur

[1] M. Bruschetta, Y. Chen, D. Cunico, E. Mion, and A. Beghi. A nonlinear MPC based motion cueing strategy for a high performance driving simulator with active seat. In *International Workshop on Advanced Motion Control (AMC)*, 2018.
[2] S. de Groot, J.C.F. de Winter, M. Mulder, and P.A. Wieringa. Nonvestibular motion cueing in a fixed-base driving simulator: Effects on driver braking and cornering performance. *Presence*, 20(2):117–142, 2011.
[3] T.W. Showalter. *The effects of motion and g-seat cues on pilot simulator performance of three piloting tasks*. National Aeronautics and Space Administration, 1980.

Betreuer/-in:	M. Leibold (TUM), T. Treichl (DLR)
Beginn:	01.06.2023
Zwischenbericht:	03.08.2023
Abgabe:	05.10.2023

(M. Leibold)
Akad. Rat

Zusammenfassung

Im Rahmen der Arbeit wurde ein Ansteuerungsalgorithmus für die Aktuatoren eines G-Seats für den DLR Robotic Motion Simulator (RMS) entwickelt. Bei den Aktuatoren handelt es sich um bewegliche Klappen unter den Sitzpolstern, die Druck auf den Körper des Insassen ausüben. Dadurch wird ein Bewegungseindruck erzeugt. In der Arbeit wurden zunächst die Motoren eines bestehenden Hardware Prototyps mithilfe des echtzeitfähigen Bussystems EtherCat in Betrieb genommen. Außerdem wurde ein Force Cueing Algorithmus implementiert, der aus simulierten Beschleunigungen des Flug- oder Fahrzeugmodells geeignete Aktuatorpositionen berechnet, um diese mit dem G-Seat darzustellen. Die Datenkommunikation zwischen dem Force Cueing Algorithmus und den Aktuatoren funktioniert dabei über einen Server, der nach dem publish and subscribe Prinzip arbeitet. Der Force Cueing Algorithmus arbeitet mit einem klassischen Washout-Filter. Die durch die Klappenmechanik veränderlichen Kraftrichtungen der Aktuatoren werden im Force Cueing berücksichtigt. Der Algorithmus lässt sich sowohl als Stand-alone Lösung als auch parallel zu Motion Cueing des RMS nutzen. Anhand einer simulierten Flugtrajektorie konnte gezeigt werden, dass der implementierte Force Cueing Algorithmus plausible Werte für die Aktuatorpositionen berechnet.

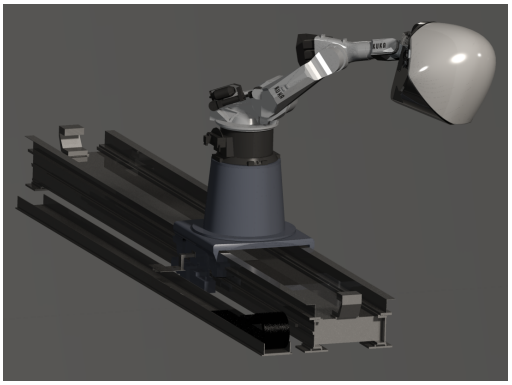
Inhaltsverzeichnis

Kapitel 1

Motivation

Bewegungssimulatoren sind ein wesentlicher Bestandteil der Forschung in der Luft- und Raumfahrt, sowie auch in der Automobilbranche und anderen Bereichen des Verkehrssektors. Eine reine visuelle Simulation der Umgebung ist hierbei oftmals nicht ausreichend, da das menschliche Wahrnehmungssystem auch auf anderen Wegen Eindrücke von Bewegung erhält, beispielsweise durch das Otholith-Vestibularsystem oder das Wahrnehmen von Kräften auf den Körper [Bel14]. Eine zu große Abweichung zwischen den Bewegungseindrücken durch die unterschiedlichen Kanäle kann bei dem Insassen zu Übelkeit, sogenannter "motion sickness"führen [BMB17].
Der klassische Weg, um diesen vestibularen Teil der menschlichen Wahrnehmung ebenfalls zu stimulieren, ist die Verwendung einer beweglichen Plattform oder einer durch einen Roboterarm gesteuerten Kapsel, wie sie auch im DLR Robotic Motion Simulator (RMS) vorhanden ist. Da solch ein Simulator nur einen begrenzten Arbeitsraum besitzt, ist es notwendig die gewünschten Beschleunigungen so zu simulieren, dass sie zum einen möglichst realistisch vom Insassen wahrgenommen werden, zum anderen der Simulator aber seinen Arbeitsbereich nicht verlässt, wozu sogenannte Motion Cueing Verfahren verwendet werden. Wünschenswert hierfür ist es möglichst viele Freiheitsgrade zu besitzen und unterschiedliche Aktuatoren zu verwenden, die für unterschiedliche Arten von Bewegung geeignet sind. Beispielsweise lassen sich lang anhaltende Beschleunigungen nur begrenzt durch eine tatsächliche geradlinige Bewegung des Simulators simulieren, da hierfür der Arbeitsraum nicht ausreicht.
Eine mögliche Erweiterung für Simulatoren sind G-Seats. Dabei werden zusätzlich zur Kabinenbewegung durch den Sitz selbst Kräfte auf den Insassen ausgeübt. Dies kann durch die Verwendung verschiedener Aktuatoren erreicht werden, z.B. sich straffende Gurte, wie in [BCC+18] oder [SGW+09], oder aufblasbare Luftkissen im Sitz [Sho80]. Anders als bei der Bewegung einer Kapsel entstehen die Kräfte, um die gewünschte Bewegungswahrnehmung zu erreichen, durch Druck auf den menschlichen Körper. Das Verfahren, um die passenden Aktuatorbewegungen für die gewünschte Wahrnehmung zu bestimmen, wird in diesem Fall als Force Cueing

(a) Cockpit des RMS



(b) Rendering des RMS

Abbildung 1.1: Robotic Motion Simulator des DLR

bezeichnet. G-Seats können außerdem alleinstehend als low-cost Alternative zu den sehr teuren bewegten Plattformen oder Roboterarmen verwendet werden [SGW+09].

In dieser Arbeit sollen Aktuatoren für einen G-Seat für den Robotic Motion Simulator (RMS) des DLR entwickelt werden. Durch den Einbau eines G-Seats in den RMS wird sich erhofft Bewegungen besser darstellen zu können, die für den Simulator aktuell sehr schwierig auszuführen sind. Dabei handelt es sich um die besonders hochfrequenten Bewegungen, für welche der RMS aktuell zu langsam ist, und die besonders niedrigfrequenten Bewegungen, die für den RMS ein Problem darstellen, da der Simulator Limitierungen besitzt und sich nicht beliebig lange in eine Richtung bewegen kann. Diese langen Bewegungen können zwar über die sogenannte Tilt Coordination abgebildet werden. Allerdings funktioniert diese nur begrenzt für Beschleunigungen über 1G. Im Gegensatz zu anderen Arbeiten zu diesem Thema sind die Aktuatoren hier bewegliche Klappen, welche einen Bewegungseindruck erzeugen sollen. Diese sollen an verschiedenen Stellen im Sitz angebracht werden können, z.B. unter der Sitzfläche oder in der Rückenlehne. Der Force Cueing Algorithmus, durch welchen die Klappen angesteuert werden, muss dabei diese spezielle Form der Aktuatoren berücksichtigen. Dadurch entsteht die Herausforderung, dass die Richtung der Kraft, die eine einzelne Klappe ausübt, im Algorithmus berücksichtigt werden muss.
In dieser Arbeit werden zunächst einige Studien evaluiert, die den positiven Nutzen eines G-Seats bestätigen (Kapitel 2.1). Danach werden verschiedene Verfahren vorgestellt, welche bereits existieren um Motion Cueing, bzw. Force Cueing, durchzuführen, und auf deren Vor- und Nachteile eingegangen (Kapitel 2.2). Basierend darauf werden die Verfahren für die Anwendung auf den G-Seat bewertet und eine der Strategien ausgewählt (Kapitel 2.3). Im nächsten Kapitel wird dann die für den Prototypen des G-Seats verwendete Hard- und Software beschrieben (Kapitel 3). Dabei wird auch das Systemverhalten der Klappen analysiert (Kapitel 3.2). Ebenso

wird die Software vorgestellt, die einen Echtzeitbetrieb des G-Seats gewährleistet (Kapitel 4). Anschließend wird die Implementierung eines Force Cueing Algorithmus für den genannten Anwendungsfall beschrieben (Kapitel 5) und dessen Plausibilität anhand von Tests von verschiedenen Beschleunigungsvorgaben für den Algorithmus getestet (Kapitel 6). Abschließend werden die erhaltenen Ergebnisse nochmals zusammengefasst und bewertet, sowie ein Ausblick auf mögliche zukünftige Arbeiten zu diesem Thema gegeben (Kapitel 7).

Kapitel 2

Stand der Technik

In diesem Kapitel wird Literatur zum Thema G-Seats und Motion Cueing Algorithmen zusammengetragen, woraus Schlüsse für die Entwicklung des Force Cueing Algorithmus in dieser Arbeit gezogen werden. Zunächst werden Studien zu G-Seats betrachtet, um den Nutzen dieser zu validieren. Anschließend werden verschiedene bereits in der Literatur vorhandene Motion Cueing Verfahren mit deren Vor- und Nachteilen vorgestellt. Abschließend wir darauf eingegangen, wie sich Motion Cueing auf den G-Seat anwenden lässt und basierend darauf ausgewählt, welches der vorgestellten Arten von Motion Cueing, bzw. im Anwendungsfall dann Force Cueing, für den G-Seat sinnvoll ist.

2.1 Nutzen von G-Seats

Studien zeigen, dass G-Seats eine sinnvolle und hilfreiche Erweiterung für Bewegungssimulatoren darstellen und in Kombination mit einer bewegten Plattform für eine weitere Verbesserung des Bewegungseindrucks sorgen können.
In der Studie von Chung et al. [CBP+01] wurden verschiedene Konfigurationen eines Simulators mit Beteiligung eines G-Seats getestet. Piloten steuerten hierbei Blackhawks (militärische Transporthubschrauber) in einem Bewegungssimulator. In diesem wurden ein G-Seat allein, sowie ein G-Seat in Kombination mit Plattformbewegung verwendet. Die Piloten berichteten, dass der G-Seat in Kombination mit der Plattform den Bewegungseindruck verbesserte im Vergleich zur Plattform allein. Der G-Seat alleine war allerdings nicht wirklich überzeugend. Ebenfalls wurde der G-Seat mit nur einem und mit zwei Freiheitsgraden eingesetzt, wobei, die Variante mit mehr Freiheitsgraden mehr überzeugte.
Die Studie von White [Whi89] führte zu ähnlichen Ergebnissen. Dort wurden ebenfalls Helikopterflüge simuliert. Auch hier fiel das Fazit der Piloten so aus, dass die Erweiterung einer bewegten Plattform mit einem G-Seat zu einer signifikanten Verbesserung des Bewegungseindruck führte. Die Piloten berichteten auch, dass sie weniger Konzentration benötigten und mehr Selbstvertrauen hatten, wodurch wesentlich aggressiver geflogen werden konnte. Der G-Seat alleine stellte allerdings auch

hier keine Verbesserung zur bewegten Plattform dar.

2.2 Motion Cueing Algorithmen

Ein Bewegungssimulator unterliegt Einschränkungen. Das heißt, er kann sich nicht wie ein reales Fahrzeug frei bewegen, sondern ist durch seine Beschaffenheit, beispielsweise die maximale Auslenkung eines Roboterarmes, eingeschränkt. Der Bereich, in welchem sich der Simulator tatsächlich bewegen kann wird als Arbeitsraum bezeichnet. Es besteht also die Schwierigkeit, dass die Bewegungen eines realen Fahrzeugs im beschränkten Arbeitsraum des Simulators abgebildet werden müssen. Um dies zu erreichen werden Motion Cueing Algorithmen, bzw. im Fall des G-Seats Force Cueing Algorithmen verwendet. Das Grundprinzip ist also, wie von Zimmermann et al. [ZKGT15] beschrieben, die Umrechnungen von Beschleunigungen des realen Fahrzeugs in Stellungen der einzelnen Aktuatoren des Simulators, um einen möglichst realistischen Bewegungseindruck zu erzeugen. Dabei muss allerdings sichergestellt werden, dass die einzelnen Aktuatoren ihren Arbeitsraum nicht verlassen.
Für die Durchführung von Motion Cueing existieren verschiedene Ansätze. Die relevantesten davon sollen nun vorgestellt werden.

2.2.1 Washout-Filter

Die klassische Ansatz zum Motion Cueing sind sogenannte Washout-Filter. Dabei handelt es sich um Hoch- und Tiefpassfilter, die dafür sorgen, dass die Aktuatoren sich während des Betriebs immer wieder langsam in ihre Ausgangsposition zurückbewegen. Dies ist wünschenswert, damit der Simulator möglichst weit von seinen Grenzen entfernt bleibt und dadurch der Simulator für zukünftige Bewegungseingänge möglichst viel Bewegungsfreiheit besitzt.

Klassischer Washout-Filter

Im Folgenden wird auf den grundlegenden Aufbau eines Washout-Filters eingegangen, wie er beispielsweise von Asadi et al. [ABQ+23] oder Bellmann [Bel14] beschrieben wird, siehe auch Abbildung 2.1.
Das Motion Cueing durch klassische Washout-Filter findet über drei Kanäle statt. Die Eingänge für den Algorithmus sind die translatorischen Beschleunigungen und die Winkelgeschwindigkeiten, die im simulierten Vehikel auftreten und möglichst realistisch durch den Simulator dargestellt werden sollen.
Die translatorischen Beschleunigungen werden zunächst skaliert und dann durch einen Hochpass zweiter Ordnung der folgenden Form mit der Dämpfung ζ_h und der Grenzfrequenz ω_h gefiltert:

$$G_{\mathrm{HPTrans}} = \frac{s^2}{s^2 + 2\zeta_h\omega_h s + \omega_h^2} \tag{2.1}$$

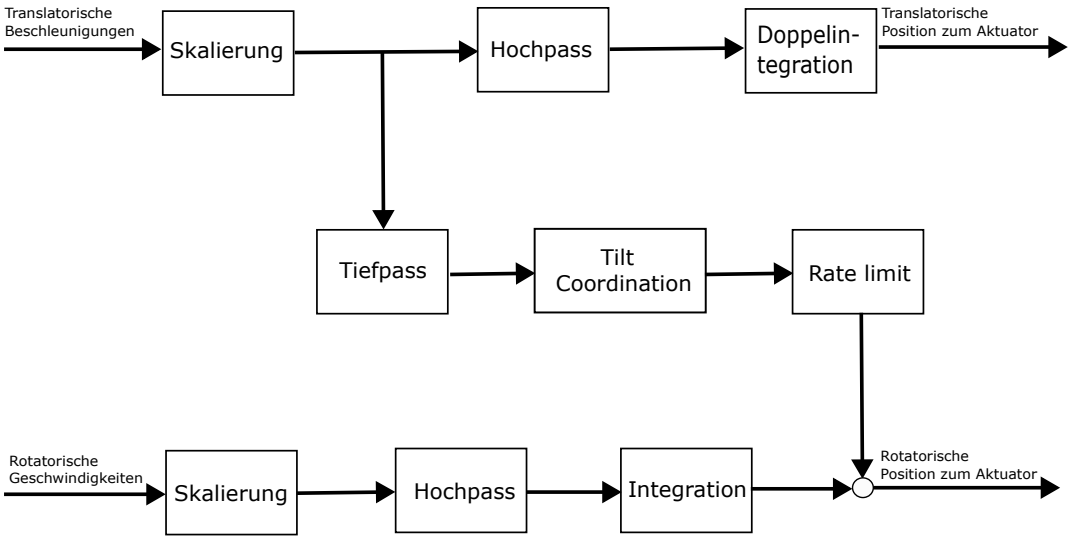


Abbildung 2.1: Struktur des klassischen Washout-Filter

Dieser sorgt dafür, dass nur die hochfrequenten Anteile, also kurze, ruckartige Beschleunigungen, durch eine tatsächliche translatorische Bewegung des Simulators abgebildet werden. Eine doppelte Integration nach der Filterung liefert direkt einen Positionsreferenzwert für den Simulator.
Sehr ähnlich funktioniert auch der direkte Kanal für die rotatorischen Geschwindigkeiten. Diese werden ebenfalls skaliert und hochpassgefiltert, hier nur durch einen Filter erster Ordnung mit Grenzfrequenz $\omega_{h,\theta}$:

$$G_{\mathrm{HPRot}} = \frac{s}{s + \omega_{h,\theta}} \tag{2.2}$$

Das Ergebnis wird dann allerdings nur einmal integriert, um einen Vorgabewert für die rotatorische Position zu erhalten.
Beim dritten Kanal handelt es sich um die sogenannte Tilt Coordination. Das Grundprinzip dahinter ist, dass die niederfrequenten Anteile der translatorischen Beschleunigungen durch eine Neigung des Simulators abgebildet werden. Die Idee ist, dass durch die Verdrehung des Simulators mithilfe der Gravitationskraft der menschlichen Wahrnehmung eine niederfrequente translatorische Beschleunigung vorgetäuscht werden kann. Dafür werden zunächst die eingehenden translatorischen Beschleunigungen tiefpassgefiltert mit Dämpfung ζ_l und Grenzfrequenz ω_l:

$$G_{\mathrm{LPTilt}} = \frac{\omega_l^2}{s^2 + 2\zeta_l\omega_l s + \omega_l^2} \tag{2.3}$$

Hierbei gibt es neben der Implementierung eines separaten Tiefpasses auch folgende Realisierung [Lor08]:

$$G_{\mathrm{LPTilt}} = 1 - G_{\mathrm{HPTrans}} \tag{2.4}$$

Dadurch wird sichergestellt, dass alle Anteile, die nicht durch den translatorischen Kanal (Gleichung 2.1) abgebildet werden, durch die Tilt Coordination repräsentiert werden und somit keine Frequenzen verloren gehen. Aus den gefilterten Beschleunigungen wird nun der Winkel berechnet, um welchen der Simulator geneigt werden muss, sodass sich die Neigung für den Insassen wie die vorgegebene niederfrequente Bewegung anfühlt. Des Weiteren ist es notwendig die aufgrund der Tilt Coordination auftretende Bewegung zu begrenzen, sodass sie sich unterhalb der menschlichen Wahrnehmungsschwelle befindet und nicht fälschlicherweise als rotatorische Bewegung wahrgenommen wird. Die niederfrequenten Anteile der rotatorischen Geschwindigkeiten müssen nicht berücksichtigt werden, da der Mensch konstante Drehraten nicht wahrnehmen kann.
Abschließend müssen noch die Koeffizienten der einzelnen Filter, sowie die Skalierungen eingestellt werden. Dafür gibt es kein festes Optimierungsverfahren. Das finale Einstellen der Parameter muss also per Trial and Error Prinzip in Zusammenarbeit mit erfahrenen Piloten oder Fahrern und deren Rückmeldung, welche Einstellungen sich am realistischsten anfühlen, durchgeführt werden [GR97].
Der größte Vorteil des klassischen Washout-Filters ist die relativ einfache Implementierung durch Übertragungsfunktionen. Das Einstellen der Filterkoeffzienten stellt allerdings eine größere Herausforderung dar, da dies nur experimentell in Zusammenarbeit mit Testpersonen durchgeführt werden kann.

Weiterentwicklungen des Washout-Filters

Eine Weiterentwicklung des klassischen Washouts ist der adaptive Washout-Filter, beispielsweise implementiert in [NRK92]. Das Prinzip dahinter ist die Gewichtungen der Größen während des Betriebs anzupassen. Der Fehler zwischen der idealen Bewegung aus der Simulation und der durch den Simulator dargestellten Bewegung wird durch eine Kostenfunktion repräsentiert, welche dann minimiert werden soll. Dadurch ergeben sich dann die sich in Echtzeit ändernden Skalierungen. Die Filterparameter müssen allerdings weiterhin mit demselben Prinzip festgelegt werden, wie beim klassischen Washout-Filter.

Der optimale Washout-Filter beschrieben in [GB10] benutzt einen ganz anderen Ansatz. Hierbei wird das Motion Cueing als Tracking Problem betrachtet, siehe Abbildung 2.2, bei dem die im Simulator wahrgenommene Bewegung möglichst genau der idealen wahrgenommenen Bewegung, die durch die Simulation vorgegeben wird, folgen soll. Der Unterschied dieser beiden Bewegungen bildet dann den Fehler des Systems. Um aus der absoluten die wahrgenommene Bewegung zu erhalten, wird ein Modell des menschlichen vestibularen Systems integriert. Für die Berechnung der idealen wahrgenommenen Bewegung werden die idealen Beschleunigungen aus der Simulation lediglich in das Vestibularmodell gegeben. Das Prinzip des Washout-

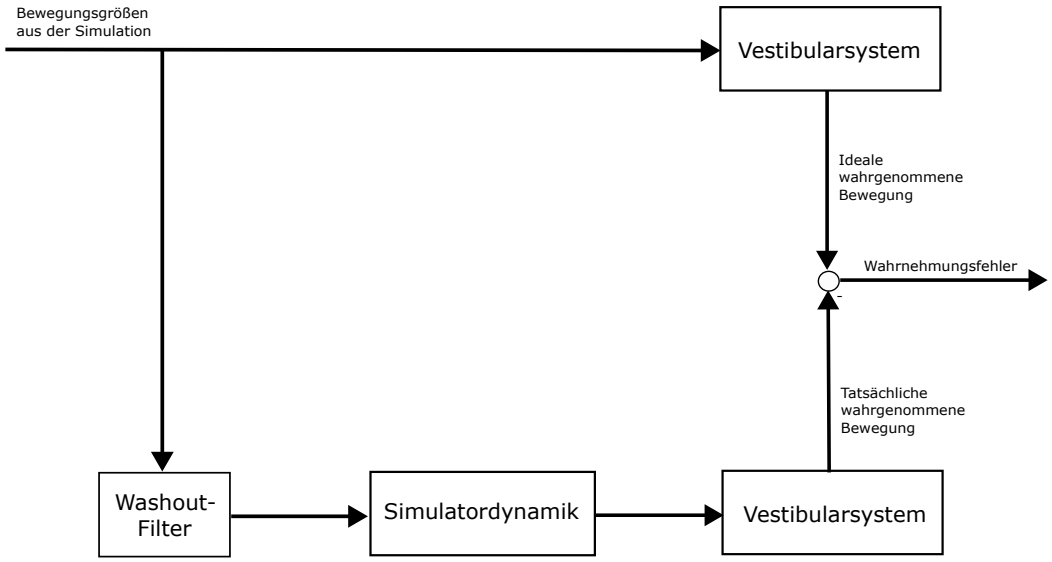


Abbildung 2.2: Motion Cueing als Tracking Problem für den optimalen Washout-Filter

Filters bleibt allerdings gleich. Er bekommt als Eingang die idealen Bewegungsgrößen und filtert diese. Die erhaltenen Größen werden dann als Referenzwerte für die Aktuatoren verwendet. Um letztendlich den Wahrnehmungsfehler berechnen zu können, müssen die absoluten Größen des Simulators ebenfalls noch in das Vestibularmodell gegeben werden.
Der optimale Washout-Filter ergibt sich dann aus der Minimierung einer Kostenfunktion, welche den Wahrnehmungsfehler, die Zustände des Simulators und die für die Aktuatoren vorgegebene Bewegung mit entsprechenden Gewichtungsmatrizen beinhaltet.
Ein wesentlicher Vorteil im Vergleich zu den anderen vorgestellten Washout-Filtern liegt darin, dass keine Filterparameter eingestellt werden müssen, sondern lediglich die Gewichtungen für die Kostenfunktion gewählt werden müssen. Dies vereinfacht den ganzen Prozess, da die Gewichtungen der Kostenfunktion wesentlich intuitiver als die Filterparameter sind und lediglich ein Kompromiss zwischen möglichst idealer Wahrnehmung und Begrenzung der Simulatorbewegung gefunden werden muss.
Im Vergleich der drei Varianten von Shao et al. [SGW+09] ist zu erkennen, dass die Weiterentwicklungen des Washout-Filters, insbesondere der optimale Washout-Filter, die Bewegungswahrnehmung verbessern. Sie bringen allerdings auch eine größere Komplexität mit sich. Beim adaptiven Washout-Filter entsteht ein zusätzlicher in Echtzeit durchzuführender Rechenaufwand und der optimale Washout-Filter benötigt eine Modellierung der menschlichen Wahrnehmung.

False cues

Ein Problem der Washout-Filter sind false cues. False cues sind generell falsche Bewegungen des Simulators, die nicht mit der Bewegung übereinstimmen, die der Insasse beispielsweise durch seine visuelle Wahrnehmung erwarten würde [GR97]. Bei

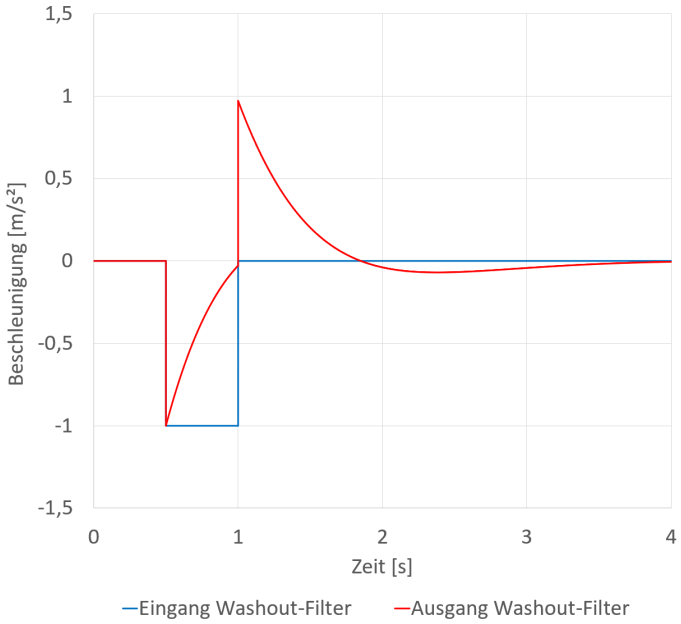


Abbildung 2.3: False cue bei Verwendung eines klassischen Washout-Filters

Washout-Filtern werden false cues vor allem durch die Hochpassfilterung hervorgerufen. In [RK00] wird die Antwort eines Hochpasses auf ein abruptes Bremsmanöver betrachtet. In Abbildung 2.3 ist dieser Vorgang mit dem Eingang des Washout-Filters u in blau und dem Ausgang y in rot, mit welchem dann der Simulator bewegt wird, dargestellt. Da es sich um einen Hochpass handelt, haben nur die steilen Flanken des Bremsmanövers einen Einfluss auf den Ausgang. Dadurch entsteht der Ausschlag der Ausgangskurve in den positiven Bereich beim Beenden des Bremsvorgangs. Das bedeutet, der Simulator beschleunigt in die entgegengesetzte Richtung, was nicht mit der vorgegebenen Beschleunigung übereinstimmt. Der Mensch im Simulator nimmt also eine falsche Bewegung wahr, die von seinen visuellen Eindrücken abweicht.
Es wird in [RK00] allerdings auch eine Möglichkeit aufgezeigt dieses Problem einzuschränken. Nach der Filterung wird eine Logik eingefügt, die überprüft, wann ein false cue erzeugt wird. Ein variabler Faktor wird dann so angepasst, dass die Auswirkungen des false cues auf die Bewegungswahrnehmung möglichst gering gehalten werden. Die Beschleunigung in entgegengesetzte Richtung ist allerdings trotzdem notwendig um den Washout-Effekt zu erzeugen und darf daher nicht komplett unterdrückt werden, da die Aktuatoren sich sonst nicht in ihre Ausgangsposition zurückbewegen könnten, was dem Grundprinzip des Washout-Filters entspricht.

2.2.2 Model Predictive Control

Bei Model Predictive Control (MPC) handelt es sich um einen anderen Ansatz, der nicht auf Filter zurückgreift. Gerade neuere Veröffentlichungen seit ca. 2010 zum Thema Motion Cueing behandeln häufig eine Implementierung mithilfe von MPC. Die grundlegende Idee, wie z.B. in [BBB+11], ist es, durch das Vorhersagen der Bewegungen in der Zukunft die ideale Simulatorbewegung zum aktuellen Zeitpunkt zu berechnen. Diese ideale Bewegung wird durch Minimierung einer Kostenfunktion erreicht, welche auch direkt die Beschränkungen des Simulators berücksichtigt. Ebenfalls wird das menschliche Wahrnehmungssystem in die Berechnung miteinbezogen. Nachdem dann die ideale Bewegung zum aktuellen Zeitpunkt getätigt wurde, geschieht der komplette Prozess erneut zum nächsten Zeitschritt. Abbildung 2.4 zeigt die grundlegende Struktur von Motion Cueing durch MPC.

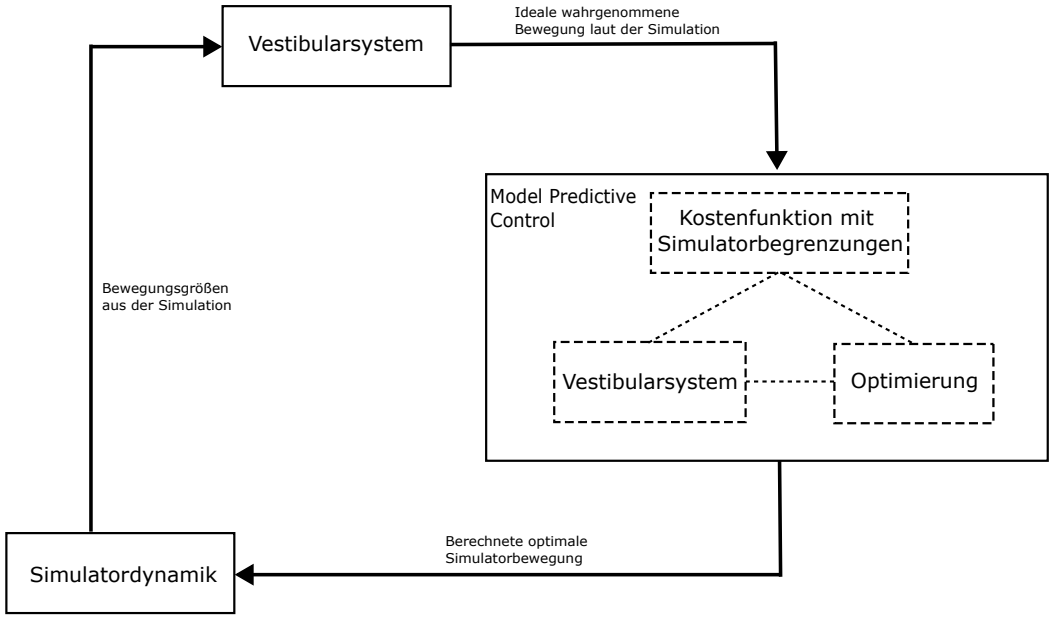


Abbildung 2.4: Grundlegende Struktur von Motion Cueing durch MPC

Die wesentlichen Vorteile von MPC in Motion Cueing Anwendungen liegen darin, dass zum einen der Arbeitsraum des Simulators ideal ausgenutzt wird, da die Beschränkungen direkt von der Kostenfunktion berücksichtigt werden und das Einstellen der Parameter, wie beim optimalen Washout-Filter, lediglich das Bilden eines Kompromisses durch das Gewichten der Kostenfunktion ist.
Ein Nachteil von MPC ist, dass durch die ständige neue Optimierung zu jedem Zeitschritt ein sehr hoher Rechenaufwand entsteht, der möglicherweise verhindert, dass ein Simulator mit einer gewünschten Echtzeitrate betrieben werden kann. Um diesem Problem entgegenzuwirken, gibt es bereits Ansätze um Teile der Optimierung

off-line durchzuführen, wodurch der verbleibende on-line Teil schneller berechnet werden kann [BMB17] [BCC+18].
Des Weiteren stellt die Prädiktion eine große Herausforderung dar. Offensichtlich ist es nicht möglich das Verhalten des simulierten Systems in der Zukunft ideal vorherzusagen, da dieses auch von der Steuerung des Insassen in der Zukunft abhängt. Wie gut eine Vorhersage erstellt werden kann hängt stark von der Art der Simulation ab. Bei der Simulation einer Fahrt auf einer Rennstrecke ist es beispielsweise deutlich leichter passende Vorhersagen zu erstellen, da der Fahrer vermutlich dem Verlauf der Strecke folgen wird. Bei einer Flugzeugsimulation wiederum gibt es deutlich mehr mögliche Bewegungen, die der Pilot als Nächstes durchführen könnte, sodass eine Prädiktion deutlich schwieriger wird. Allerdings kann auch eine nicht ideale Prädiktion bereits zu einer verbesserten Wahrnehmung im Vergleich zu den Washout-Filtern führen [ZKGT15]. Auch der Zeitraum der Prädiktion muss passend gewählt werden. Er darf nicht zu kurz sein, da sonst der Bewegungseindruck schlechter werden kann als bei den Washout-Filter, siehe Vergleich von [ZKGT15]. Allerdings darf der Prädiktionszeitraum auch nicht zu groß gewählt werden, da dann die Vorhersage des Verhaltens für die weiter entfernten Zeitpunkte kaum möglich ist [BMB17] und auch eine Annahme von konstanten Beschleunigungen, wie in [ZKGT15], zu großen Abweichungen führen würde.

2.2.3 Anwendung auf den G-Seat

Im Falle des G-Seats sollen die von der Simulation vorgegebenen Beschleunigungen durch Druck auf den Körper abgebildet werden. Das verwendete Verfahren wird als Force Cueing bezeichnet. Dieses funktioniert sehr ähnlich wie das Motion Cueing. Nur wird hier Druck von den Aktuatoren auf den menschlichen Körper ausgeübt. Besonders muss dabei die Position der Aktuatoren, bzw. deren Kraftrichtung berücksichtigt werden, damit am Ende auch die Richtungen der wahrgenommenen Beschleunigungen korrekt sind. Des Weiteren existiert keine Tilt Coordination, da der G-Seat nicht mit Drehung arbeitet sondern nur noch mit der Ausübung von Druck auf den Körper. Für die Cueing-Strategien, die ein Wahrnehmungsmodell des Menschen verwenden, muss zusätzlich ein Modell für das Empfinden von Druck miteinbezogen werden. Implementierungen von Force Cueing existieren bereits sowohl mit dem optimalen Washout-Filter, z.B. in [SGW+09], als auch mit MPC, z.B. in [BCC+18]. In letzterer Implementierung wurde der G-Seat als Ergänzung zu einer sich bewegenden Plattform realisiert, während in der Implementierung durch den optimalen Washout-Filter der G-Seat allein als Aktuator verwendet wurde.

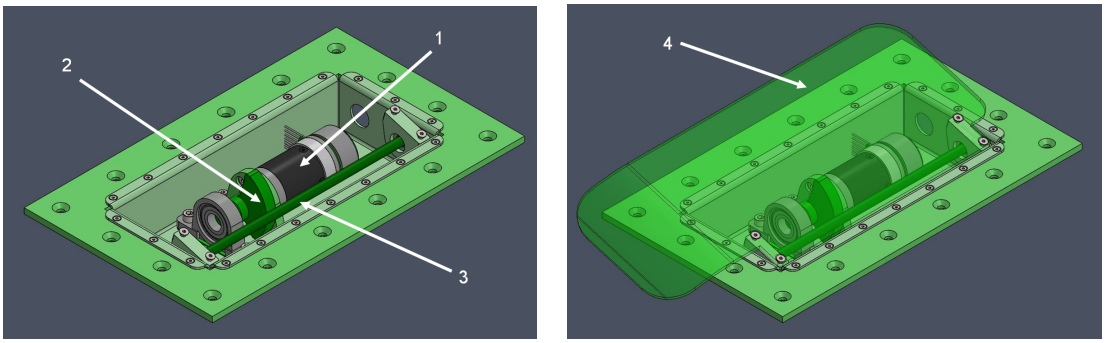
2.3 Schlussfolgerungen für diese Arbeit

Bei einer Klappe, die sich nur in einer Dimension bewegen kann (siehe Abbildung 3.1 im nächsten Kapitel), handelt es sich um ein sehr simples System im Vergleich zu größeren Bewegungssimulatoren mit mehreren Freiheitsgraden. Generell ist MPC ein sehr interessanter Ansatz, der auch in den meisten Fällen bessere Ergebnisse in der Wahrnehmung liefert als Washout-Filter. Speziell für den G-Seat scheinen die Vorteile von MPC allerdings nicht so stark ins Gewicht zu fallen, da ein ideales Ausnutzen des Arbeitsraums bei einer Klappe, die Druck auf den Körper ausüben soll, nicht so wichtig erscheint wie in einem größeren Simulator mit vielen Freiheitsgraden, welche größtenteils das Anwendungsgebiet von MPC-Algorithmen in der Literatur sind. Da MPC zusätzlich noch eine höhere Komplexität mit sich bringt, wie die aufwendige Prädiktion und den hohen Rechenaufwand, wird MPC für diese Arbeit nicht in Betracht gezogen.
Auch der adaptive und der optimale Washout-Filter würden eine Komplexität mit sich bringen, die für die Anwendung auf die Klappensysteme nicht verhältnismäßig ist. Insbesondere der optimale Washout-Filter wird für diesen Anwendungsfall sehr kompliziert, da zusätzlich zu einem Modell des vestibularen Systems noch ein Modell für das Empfinden von Druck des menschlichen Körpers einbezogen werden müsste. Dieses Druckmodell führt durch den Druck der Aktuatoren dazu, dass sich die Systemmatrizen der beiden Pfade des optimalen Washout-Filters unterscheiden [SGW+09]. Dadurch kann das Zustandsraummodell für den Wahrnehmungsfehler nicht einfach durch eine Subtraktion der Zustände der beiden Teilsysteme hergeleitet werden, wie dies in Anwendungen ohne Druckaktuatoren gemacht wird, z.B. in [AMN+15] . Stattdessen müssen die Systeme auf eine komplexere Art verknüpft werden, indem das kombinierte System durch das Entfernen aller nicht steuerbaren Eigenwerte in eine minimale Realisierung umgeschrieben wird [SGW+09].
Daher wird im Force Cueing Algorithmus in dieser Arbeit ein klassischer Washout-Filter verwendet.

Kapitel 3

Funktionsweise des G-Seat

Um die Effekte des Force Cueings zu testen und die Performanz einer Klappe zu analysieren stand bereits ein Prototyp eines Aktuators zur Verfügung. Dieser ist in Abbildung 3.1 zu sehen. In diesem Kapitel wird die Funktionsweise des Prototypen erklärt und auf die Ansteuerung eingegangen, über welche die Klappe in die gewünschte Position gebracht wird.



(a) Ohne Klappe (b) Mit Klappe

Abbildung 3.1: Aufbau des Prototypen

3.1 Mechanische Funktionsweise

Die verwendeten Klappen werden jeweils über einen Elektromotor (1 in Abblidung 3.1) gesteuert. Dieser Motor bekommt eine Sollposition in Grad vorgegeben. Der eingebaute Treiber erhält die mit einem inkrementellen Encoder gemessene Istposition des Motors als Eingang und regelt den Motor auf die Sollposition. Der Motor bewegt dann eine bewegliche Kurvenscheibe (2). Zwischen Motor und Scheibe sitzt ein Planetengetriebe. Die Übersetzung zwischen Motor- und Scheibenwinkel ist somit linear. Die Scheibe drückt von unten gegen die Klappe (4) und lenkt sie dadurch aus. Die Klappe selbst ist an der in der Aktuatorbox zu sehenden Achse (3) wie in

Abbildung 3.1(b) befestigt und lässt sich somit um diese Achse drehen. Je nach Größe der verwendeten Scheibe ändert sich die maximale Auslenkung der Klappe. Bei der in dieser Arbeit verwendeten Scheibe beträgt diese ca. 16,5°, was einer Auslenkung der Scheibe von 136,5° entspricht. Welche maximale Klappenauslenkung für den G-Seat am geeignetsten ist, kann durch Versuche noch ermittelt werden. Die Übersetzung zwischen Scheiben- und Klappenwinkel ist nichtlinear und kann durch eine Interpolation von Messwerten ermittelt werden. Auf die Implementierung wird später im Kapitel 5.2 noch genauer eingegangen.
Die Motorposition wird mit einem Hall-Sensor gemessen, der die Position inkrementell ausgibt. Dies hat zur Folge, dass zu Beginn des Betriebs eine Position als Nullposition definiert werden muss. Erreicht wird dies durch eine Referenzfahrt nach jedem Einschalten des Motors, in welchem sich der Motor in eine Richtung dreht bis er einen Anschlag erkennt. Dieser wird durch das Überschreiten eines Referenzwertes für den Motorstrom detektiert. An dem erkannten Anschlag wird dann der Nullpunkt gesetzt. Des Weiteren besitzt der Hall-Sensor nur eine Auflösung von 7,5°. Da der Motor allerdings für eine komplette Auslenkung der Sitzklappe von 0° auf 16,5° mehr als 10 Umdrehungen benötigt (3675°), ist die Abweichung von 7,5°, was in Sitzklappenauslenkung im Schnitt ca. 0,03° entspricht, praktisch nicht spürbar und somit kein Problem für den Betrieb.

3.2 Identifikation des Systemverhaltens

Für einen späteren Einsatz der Klappen ist es notwendig zu wissen, bis zu welchen Frequenzen der Motor seiner Positionsvorgabe folgen kann. Um das zu analysieren wird dem Motor eine Sinus-Funktion als Positionswert vorgegeben, die sich zwischen 0° und 136° Scheibenwinkel, also in nahezu dem kompletten Arbeitsbereich, bewegt. Für unterschiedliche Frequenzen wurde dabei bestimmt, welcher maximalen Amplitude der Sinus-Funktion der Motor bei der jeweiligen Frequenz noch folgen kann. Die Ergebnisse sind in Abbildung 3.2 für Frequenzen zwischen 0,1Hz und 6Hz zu sehen. Darin ist zu erkennen, dass der Motor bis zu einer Frequenz von 2Hz in der Lage ist die komplette Amplitude zu befahren. In diesem Frequenzbereich kann der Aktuator also uneingeschränkt arbeiten. Mit steigender Frequenz wird dann allerdings die maximale abbildbare Amplitude immer geringer, sodass bei 3Hz nur noch ungefähr die Hälfte des Arbeitsbereichs abgedeckt werden kann. Bei noch höheren Frequenzen ab ca. 4Hz ist dann neben der nur noch sehr geringen möglichen Amplitude auch kaum mehr eine kontrollierte Darstellung eines Sinus-Signals möglich. Die Amplituden der einzelnen Schwingungen variieren und weichen teilweise stark voneinander ab. Für den dargestellten Graphen wurde daher jeweils ein ungefährer Mittelwert als Referenzpunkt gewählt. Ein Betrieb der Klappen in diesem Frequenzbereich ist somit nicht sinnvoll.

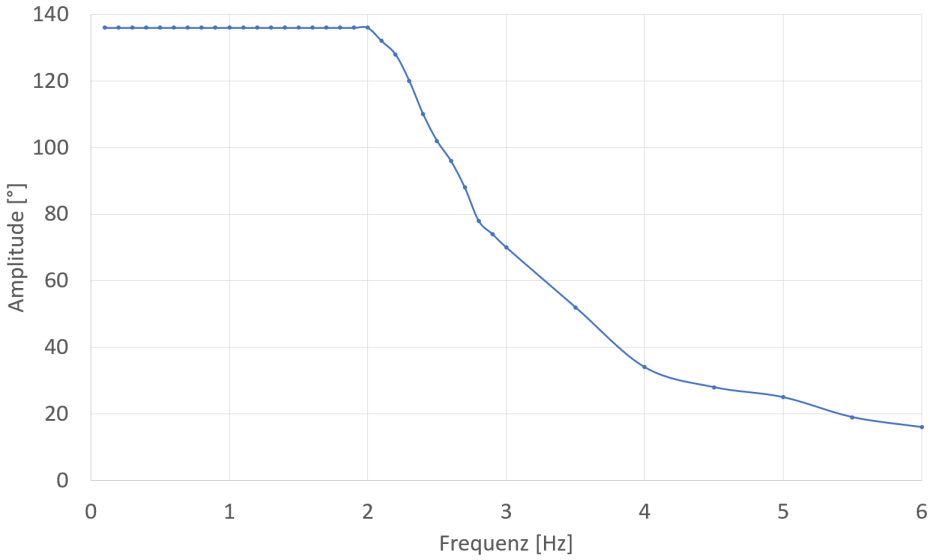


Abbildung 3.2: Verhältnis zwischen Frequenz und erreichter Amplitude bei Vorgabe einer Sinus-Funktion

3.3 Ansteuerung der Aktuatoren

Im Folgenden wird die gesamte Struktur der Ansteuerung des G-Seats beschrieben (zu sehen auch in Abbildung 3.3).
Wenn der G-Seat später im Betrieb ist, sollen die Beschleunigungen einer Flug- oder Fahrsimulation auf dem Modelnet-Server verfügbar sein. Diese werden vom Force Cueing als Eingänge verwendet. Das Force Cueing findet in dieser Arbeit über die gleichungsbasierten Modellierungssprache Modelica im Programm Dymola statt. Der Algorithmus berechnet aus den Beschleunigungen die Sollpositionen für die Aktuatoren und veröffentlicht sie wieder auf den Modelnet-Server.
Der Datenaustausch mit den Motoren findet über das echtzeitfähige Bussystem EtherCat statt. Mehr Informationen zu EtherCat sind im Anhang A aufgeführt. Die EtherCat-Schnittstelle wurde in dieser Arbeit über ein C++ Programm implementiert.
Das C++ Programm hat ebenfalls Zugriff auf den Modelnet-Server und liest die vorher vom Force Cueing Algorithmus berechneten Sollpositionen ein und übergibt sie mithilfe von EtherCat an die Motoren. Zusätzlich werden sämtliche Rückgabewerte der Motoren auf den Modelnet-Server geschrieben, um die Daten des laufenden Betriebs überwachen und analysieren zu können. Dazu gehören z.B. aktuelle Position, Geschwindigkeit und Beschleunigung der einzelnen Motoren oder deren aktueller Zustand. Zusätzlich werden zu Beginn des Betriebs Konfigurationswerte über eine Ini-Datei eingelesen, die dann während des Echtzeitbetriebs konstant bleiben.
Im folgenden Kapitel wird nun dieses C++ Programm behandelt, welches den Betrieb des G-Seat in Echtzeit gewährleistet.

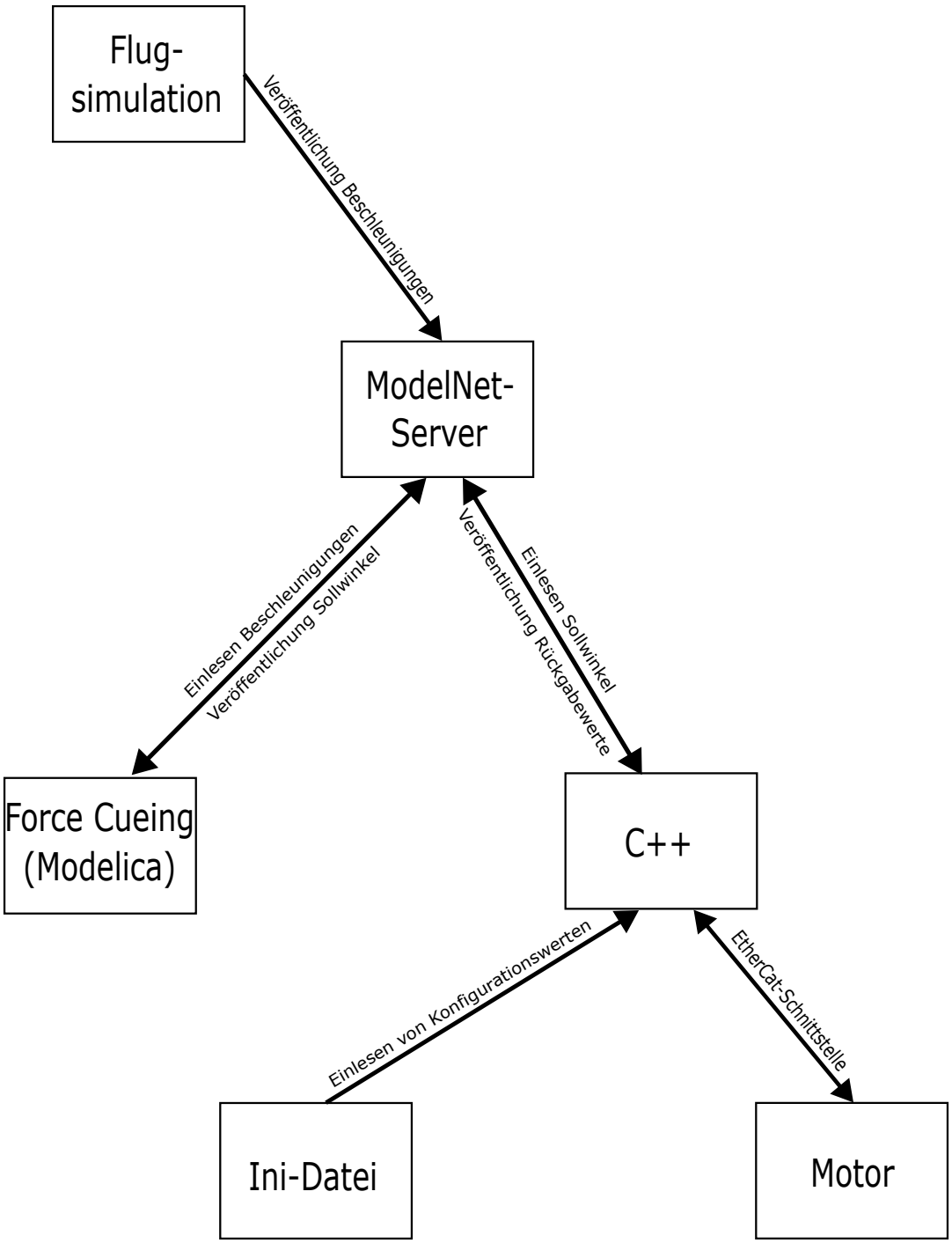


Abbildung 3.3: Gesamtstruktur der Ansteuerung

Kapitel 4

Struktur des C++ Programms für den Echtzeitbetrieb

Die Implementierung der EtherCat-Schnittstelle, der Verarbeitung der Daten und der Echtzeitschleife wurde in C++ durchgeführt. Auf die Struktur des erstellten Programms soll an dieser Stelle eingegangen werden.

4.1 Klassenstruktur

Die verschiedenen Aufgaben des C++ Programms sind in verschiedene Klassen unterteilt. Das Klassendiagramm ist in Abbildung 4.1 zu sehen.
Für jede Klappe existiert ein Objekt der Klasse Actuator. Über die main-Methode des Programms wurde implementiert, dass beliebig viele Aktuatoren angelegt und gesteuert werden können. Dies erfolgt mithilfe eines Vektors mit variabler Länge, der die Actuator-Objekte verwaltet. In diesem werden auf jedem der enthaltenen Objekte die benötigten Funktionen aufgerufen.
Die Actuator-Objekte sind lediglich für das Einlesen und Veröffentlichen von Daten auf den Modelnet-Server zuständig. Abhängig vom Namen des Aktuators und der Art der Variable, werden in dieser Klasse die Namen der Variablen erzeugt, sodass sie auf dem Modelnet-Server einheitlich benannt sind und alle anderen Objekte, die mit dem Server interagieren, korrekt mit den Werten arbeiten können. Jedes Actuator-Objekt besitzt außerdem ein Objekt der Klasse Motor. Auf dessen Variablen greift das Actuator-Objekt beim Einlesen und Ausgeben von Daten zu.
Die Klasse Motor führt den Datenaustausch mit dem tatsächlichen Motor durch. Die Vorgabewerte für den Motor, die von dem zugehörigen Actuator-Objekt in die entsprechenden Variablen des Motor-Objekts geschrieben wurden, übergibt das Motor-Objekt mithilfe von EtherCat an den in die Klappe eingebauten Elektromotor. Ebenfalls werden die Rückgabewerte in Variablen geschrieben, die wiederum das Actuator-Objekt auf den Server veröffentlicht. Des Weiteren verwaltet die Klasse Motor den Zustand des Motors und aktiviert, bzw. deaktiviert ihn abhängig von verschiedenen Kontrollvariablen. Auf diese Variablen wird später im Kapitel 5.3

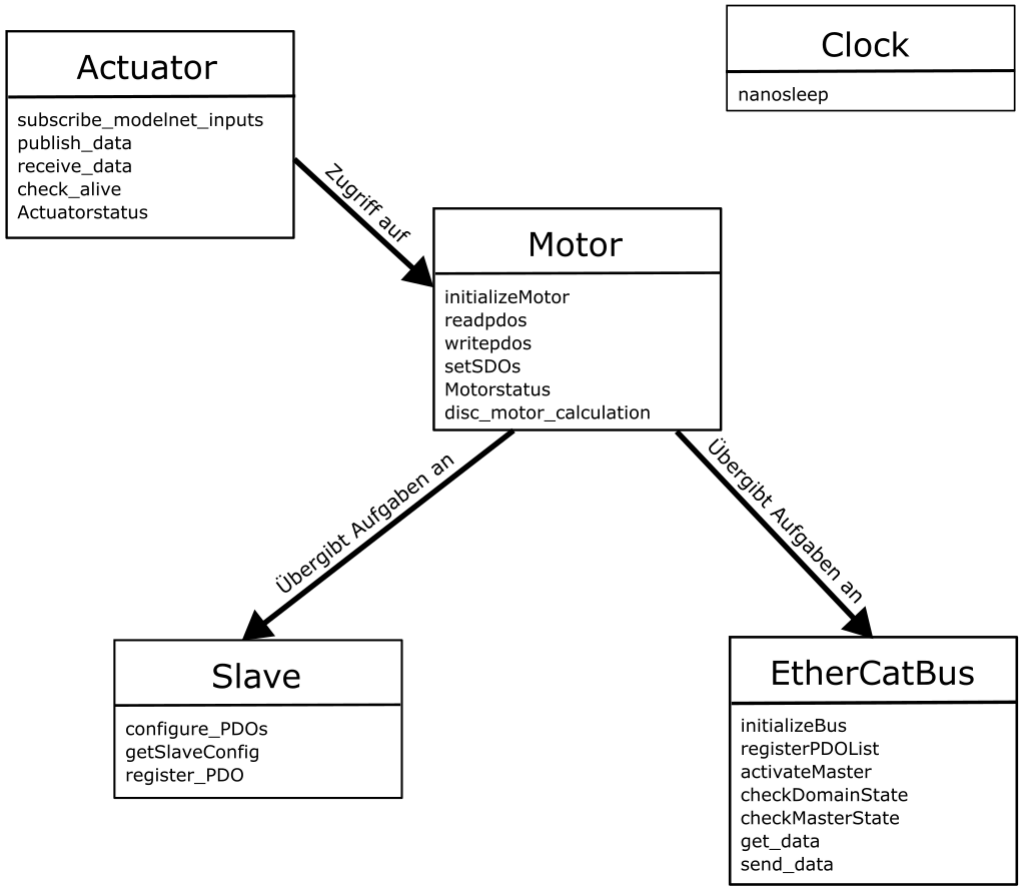


Abbildung 4.1: Klassendiagramm des C++ Programms mit Funktionen

noch eingegangen. Jedes Motor-Objekt besitzt auch ein Objekt der Klasse Slave. Die Slave-Objekte dienen zusammen mit einem Objekt der Klasse EthercatBus zum Aufbau und Verwalten der EtherCat-Struktur, die für den Echtzeitbetrieb vorhanden sein muss. Dabei wird unter anderem die Beziehung zwischen dem Master und seinen Slaves festgelegt und die Parameter des Motortreibers sowie die zyklisch über den EtherCat-Bus auszutauschenden Daten werden konfiguriert.
Zuletzt existiert noch ein Clock-Objekt, welches für alle Zeiten und Timer, die in der Echtzeitschleife benötigt werden, zuständig ist.

4.2 Einlesen von Konfigurationswerten

Neben den sich durchgehend ändernden Variablen, die durch die Simulation vorgegeben werden, müssen auch noch einige Werte vor Start des Betriebs eingestellt werden, die dann währenddessen konstant bleiben. Dafür erhält das C++ Programm neben der Serververbindung auch den Zugriff auf eine Ini-Datei. Dort können alle benötigten Werte parametriert werden.

Die Einstellungen über die Anzahl der verwendeten Aktuatoren finden über die Ini-Datei statt. Neben der Angabe der Anzahl der Aktuatoren, kann jedem Aktuator ein individueller Name gegeben werden, unter welchem die Variablen dieses Aktuators auf dem Modelnet-Server erscheinen. Zusätzlich muss dessen Position im EtherCat-Netzwerk angegeben werden.
Außerdem werden Begrenzungen für Position, Geschwindigkeit und Beschleunigung eingegeben, die der Motor während des Betriebs nicht über- oder unterschreiten soll. Weitere einzustellende Parameter sind IP-Adresse, UDP-Port und Name des Servers, mit welchem sich das C++ Programm verbinden soll, die Reglerparameter für den Regler im Motor, die notwendigen Referenzwerte für die Referenzfahrt zu Beginn des Betriebs zum Setzen des Nullpunkts und verschiedene für den Echtzeitbetrieb benötigten Zeitwerte.
Alle eingegebenen Werte werden an den Stellen im Code ausgelesen, an welchen sie benötigt werden. Davon werden die meisten Werte in der Klasse Motor direkt als Parameter auf dem Drive gesetzt, ohne dass sie nochmals in C++ zwischengespeichert werden.

4.3 Echtzeitschleife

Die Echtzeitschleife (siehe Abbildung 4.2) befindet sich in der main-Methode des C++ Programms. Diese wird mit einer Frequenz betrieben, die über die Ini-Datei eingelesen wird. Das Einhalten der Frequenz wird durch das clock-Objekt realisiert, welches dafür sorgt, dass nach einem Durchlauf der Schleife für die passende Zeit pausiert wird.
Zu Beginn jedes Durchlaufs werden zunächst die Prozessdaten vom EtherCat-Bus empfangen. Danach findet mithilfe einer Kontrollvariable eine Überprüfung statt, ob die Werte auf dem Modelnet-Server noch aktuell sind (genauere Beschreibung in Abschnitt 5.3). Die Zeit zwischen zwei Überprüfungen kann separat gewählt werden. Allerdings sind dabei nur Vielfache der Zeit möglich, mit der die Echtzeitschleife sich wiederholt. Anschließend werden die zurückgegebenen PDOs von den Motoren ausgelesen und die Werte vom Modelnet-Server empfangen. Auch für das Empfangen der Modelnet-Daten kann eine eigens eingestellte Zeit zwischen zwei Aufrufen eingestellt werden. Auch hier sind wieder nur Vielfache der Echtzeitschleifenzeit möglich. Aus den empfangenen Positionen, die als Scheibenwinkel eingelesen werden, werden dann die Vorgabewerte für die Motoren berechnet. Dies geschieht durch den linearen Zusammenhang durch eine Multiplikation mit der Getriebeübersetzung. Außerdem werden die aktuellen Zustände der Motoren überprüft und bei Bedarf geändert. Dann werden die PDOs für die Motoren geschrieben und die Rückgabewerte im selben Intervall wie das Einlesen auf den Server geschrieben. Abschließend werden die Prozessdaten an den EtherCat-Bus gesendet.
Danach folgt die Pause durch die Clock und die Schleife wird erneut durchlaufen.

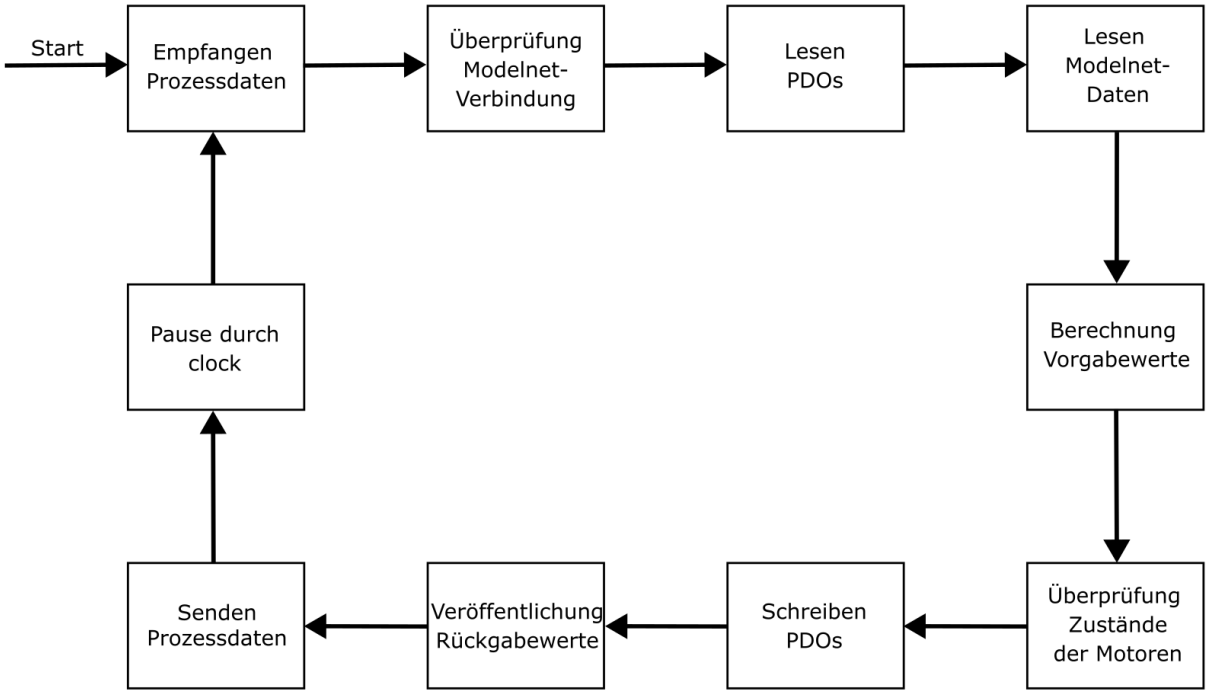


Abbildung 4.2: Ablauf der Echtzeitschleife

Kapitel 5

Implementierung eines Force Cueing Algorithmus

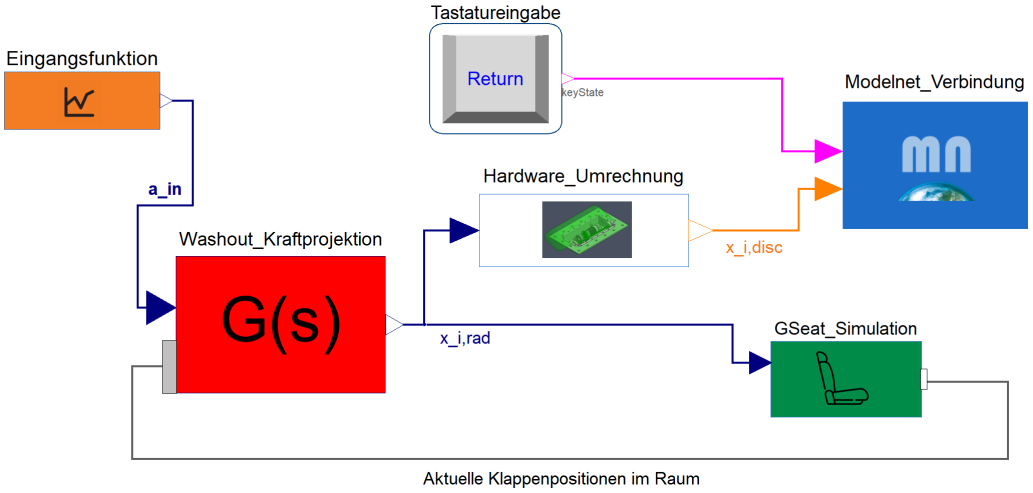


Abbildung 5.1: Aufbau des Modelica-Modells

In diesem Kapitel wird ein Force Cueing Algorithmus zur Ansteuerung der Aktuatoren des G-Seats entwickelt. Dabei wird auf die verschiedenen Komponenten des Algorithmus eingegangen. Abbildung 5.1 zeigt den Aufbau des Modelica-Modell, welches für das Force Cueing zuständig ist. Der Hauptteil des Force Cueings findet im roten Block, welcher den Washout-Filter und die Einbindung der Kraftrichtungen beinhaltet statt. Im grünen Block wird die gewünschte Konfiguration für den Sitz eingestellt und der G-Seat simuliert. Die restlichen Blöcke sind zuständig für das Erzeugen von Eingangsfunktionen (orange), das Umrechnen der Positionswerte in die für die Hardware benötigte Einheit (weiß) und die Verbindung mit einem Server (blau). Nachfolgend werden alle Blöcke und die darin stattfindenden Prozesse nacheinander beleuchtet.

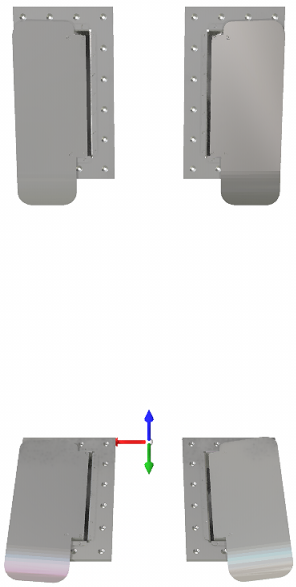


Abbildung 5.2: Beispielhafte Klappenanordnung, simuliert mit dem DLR SimVis-Tool

5.1 Generierung der Aktuatorpositionen

Der rote Block ist für das Erzeugen der Positionen für die Aktuatoren zuständig. Er erhält als Eingang einen Beschleunigungsvektor $\boldsymbol{a}_{\text{in}}$, den der G-Seat abbilden soll. Außerdem geht die aktuelle Position jedes Aktuators in den Block ein. Diese werden direkt aus dem simulierten G-Seat (siehe Abschnitt 5.4) gewonnen.

5.1.1 Kraftprojektion

Zunächst folgt die Projektion von $\boldsymbol{a}_{\text{in}}$ auf die einzelnen Kraftrichtungen der Aktuatoren f_i. Dafür müssen die Achsen der auftretenden Koordinatensysteme definiert werden, damit die Konzepte anschaulich erklärt werden können. Das Weltkoordinatensystem, zu sehen in Abbildung 5.2, ist aus der Sicht des Insassen definiert mit der x-Achse nach rechts (rot), der y-Achse nach vorne (grün) und der z-Achse nach oben (blau). Die Kraftrichtung einer Klappe ist definiert als ein Einheitsvektor, der senkrecht auf der Klappenoberfläche steht (siehe Abbildung 5.3). Im Koordinatensystem der Klappe ist dies als die z-Richtung definiert.

Konzepte für die Kraftrichtungen

Die Festlegung der Kraftrichtungen stellt in diesem Anwendungsfall eine Herausforderung dar, da sich die Klappen nicht linear, anders als in der Implementierung

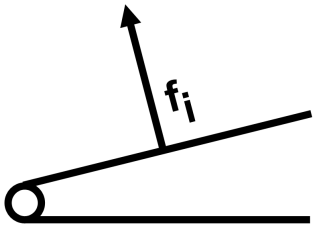


Abbildung 5.3: Definition der Kraftrichtung eines Aktuators

von z.B. Shao et. al. [SGW+09], in eine Richtung bewegen und sich die Richtung, in welche die Klappe ihre Kraft ausübt, im Weltkoordinatensystem ständig ändert. Um damit umzugehen wurden zwei Konzepte in Betracht gezogen.
In der ersten Überlegung wurden die Kraftrichtungen als konstant angenommen. Dafür wurde jeweils die Mittelstellung jeder Klappe als Grundlage für die konstante Kraftrichtung gewählt. Dieses Konzept bildet zwar die Variabilität der Klappen nicht ausreichend ab, bietet aber einen sehr einfachen Weg um die Projektion durchzuführen.
Das zweite Konzept berücksichtigt die variablen Kraftrichtungen. Dafür werden die aktuellen Positionen der Klappen aus dem simulierten G-Seat (siehe Abschnitt 5.4) per Feedback zurückgeführt. Somit wird in jedem Zeitschritt immer auf die neuen aktuellen Kraftrichtungen projeziert. Dadurch werden die Kraftrichtungen zwar realitätsnäher dargestellt, es ergeben sich allerdings andere Probleme. Beispielsweise ist es dann in der in Abbildung 5.2 gezeigten Konfiguration nicht möglich eine Beschleunigung in x-Richtung darzustellen, wenn sich alle Klappen in der Nullposition befinden, da die Kraftrichtungen in dieser Stellung alle senkrecht zur x-Achse stehen. Im praktischen Einsatz des G-Seat in einer Simulation, in der der G-Seat durchgehend unterschiedliche Beschleunigungen darstellen muss, wird diese Situation, in der keine der Klappen ausgelenkt ist, aber vermutlich nur mit kleiner Wahrscheinlichkeit eintreten. Des Weiteren ließe sich das Problem ebenfalls durch eine andere Positionierung der Aktuatoren einschränken. Daher wurde entschieden das zweite Konzept mit der Berücksichtigung der variablen Kraftrichtungen umzusetzen.

Ermittlung der Kraftrichtungen

Um nun final die Kraftrichtungen zu ermitteln muss der Einheitsvektor in z-Richtung aus dem jeweiligen Klappenkoordinatensystem in das Weltkoordinatensystem transformiert werden. Diese Transformation ist abhängig von der Position des Aktuators im Raum und der aktuellen Auslenkung dessen Klappe ϕ_i, die während des Betriebs variabel ist. In Modelica ist es möglich sich aus der Position eines Objekts im Raum direkt die Transformationsmatrix $\boldsymbol{T}$ ausgeben zu lassen, welche für die Transformation vom Weltkoordinatensystem in das entsprechende Objektkoordinatensystem benötigt wird. Daher muss die Transformationsmatrix nicht manuell aufgestellt werden, sondern kann direkt aus der von der Simulation zurückgegebenen

Position ausgelesen werden. Da hier ein Vektor vom Objekt- ins Weltkoordinatensystem transformiert werden soll, wird die Transponierte der ausgelesenen Matrix benötigt. Somit ergibt sich für die Kraftrichtung des i-ten Aktuators:

$$\boldsymbol{f_i} = \boldsymbol{T_i}(\phi_i)^\top \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.1}$$

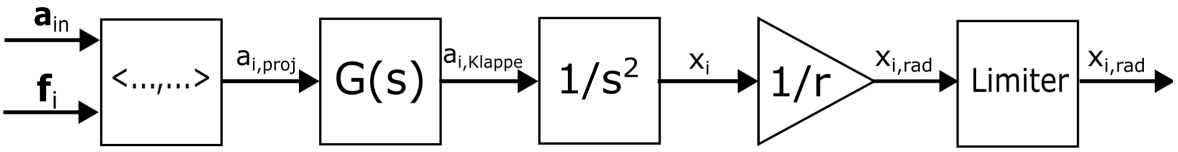


Abbildung 5.4: Ablauf des Washout-Blocks mit Kraftprojektion

Berechnung der projezierten Beschleunigungen

Nachdem nun die Kraftrichtungen berechnet wurden, auf welche der Beschleunigungseingang projeziert wird, kann die Projektion der Eingangsbeschleunigungen $\boldsymbol{a}_{\text{in}}$ durchgeführt werden. Die auf die Kraftrichtungen der einzelnen Aktuatoren abgebildeten Beschleunigungen $a_{i,\text{proj}}$ werden durch das Bilden des Skalarprodukts ermittelt:

$$a_{i,\text{proj}} = \boldsymbol{a}_{\text{in}} \circ \boldsymbol{f_i} \tag{5.2}$$

Die erhaltenen Beschleunigungen dienen dann als Eingang für den Washout-Filter.

5.1.2 Washout-Filter

Im nächsten Schritt werden die Beschleunigungen der Klappe $a_{i,\text{Klappe}}$ mithilfe eines Washout-Filters ermittelt. Wie bereits in Abschnitt 2.3 diskutiert, wird dafür ein klassischer Washout-Filter verwendet, siehe Gleichung 2.1. Die Filterparameter werden zunächst so gewählt, dass die Ergebnisse von Shao et al. [SGW+09] reproduziert werden können. Dabei handelt es sich um die Dämpfung $\zeta_h = 0,82$, die Grenzfrequenz $\omega_h = 2$ und eine zusätzliche Skalierung mit $K = 0,048$. Die Koeffizienten sind zwar auf einen G-Seat mit anderen Aktuatoren optimiert, dienen aber aufgrund fehlender anderer Referenzen erst einmal als Richtwerte, die noch auf die Ausführung des G-Seat mit Klappen angepasst werden können. Der verwendete Washout-Filter ist für alle Aktuatoren gleich und besitzt somit die Übertragungsfunktion:

$$G(s) = \frac{a_{i,\text{Klappe}}}{a_{i,\text{proj}}} = \frac{0,048s^2}{s^2 + 3,28s + 4} \tag{5.3}$$

Anschließend werden die erhaltenen Werte doppelt integriert, um aus den Beschleunigungen direkt Positionsvorgaben x_i zu erhalten.

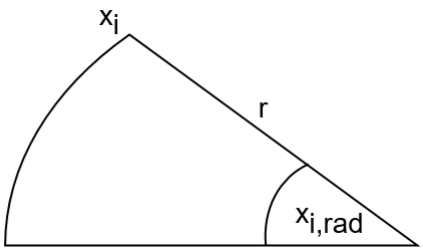


Abbildung 5.5: Klappenbewegung als Kreisbogen zur Umrechnung von Meter in Radiant

5.1.3 Einheitenumrechnung und Limitierung

Aus der Annahme, dass die eingehenden Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ vorlagen, folgt, dass die x_i nun die Einheit Meter besitzen. Da für den Betrieb der Klappen ein Winkel als Zielposition benötigt wird, müssen die x_i umgerechnet werden. Für die Umrechnung wird die Bewegung der Klappe als eine Kreisbewegung auf einem Kreisbogen um die Drehachse betrachtet (siehe Abbildung 5.5). Somit kann jeder Position in Metern auf dem Kreisbogen eindeutig ein Winkel zugeordnet werden. Der Radius des Kreisbogens kann hierbei variabel zwischen dem nächsten und dem entferntesten Punkt der Klappe zur Drehachse gewählt werden. In dieser Arbeit wurde der Mittelpunkt der Klappe gewählt, was für den Prototypen einem Radius von $r = 0.1\mathrm{m}$ entspricht. Aus dem Verhältnis zwischen Winkel und Kreisbogenlänge ergibt sich für die Zielposition im Bogenmaß:

$$x_{i,\mathrm{rad}} = \frac{x_i}{r} \tag{5.4}$$

Letztendlich führt die Wahl des Radius also nur zu einer weiteren Skalierung, wie sie bereits im Washout-Filter vorhanden ist. Von daher ist es ausreichend bei einer Optimierung nur die Washout-Koeffizienten zu verändern und den Radius nach einmaligem Wählen konstant zu halten. Die Skalierung durch den Radius hätte somit auch direkt in den Washout-Filter miteinbezogen werden können. Um die nacheinander ablaufenden Schritte aber besser darstellen zu können und hervorzuheben, dass in dieser Anwendung im Vergleich zu anderen eine Positionsvorgabe in Grad benötigt wird, wurde darauf verzichtet.
Abschließend müssen die Positionen noch auf die Limits der Klappen beschränkt werden. Zwar ist eine Beschränkung der Zielpositionen auf der C++-Seite bereits implementiert, jedoch sollen auch im Modelica-Modell die Grenzen der Klappen berücksichtigt werden, sodass die Limits auch in der Simulation der Klappen aktiv sind. Für das Berücksichtigen der minimalen und maximalen Auslenkung einer Klappe wurde ein einfacher Begrenzer genutzt, der Werte, die zu groß, bzw. zu klein für den Arbeitsraum der Klappe sind, auf die maximale, bzw. minimale Auslenkung der Klappe abbildet.
Damit sind alle Prozesse in diesem Block abgeschlossen und die Zielpositionen im Bogenmaß $x_{i,\mathrm{rad}}$ werden aus dem Block ausgegeben.

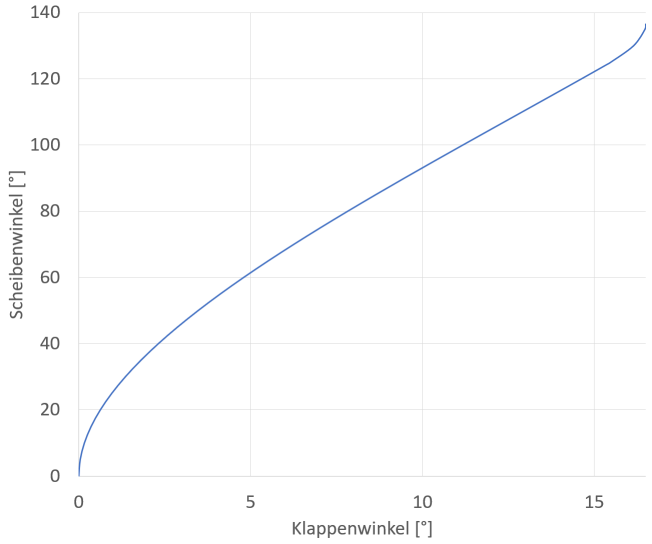


Abbildung 5.6: Abbildung Klappen- auf Scheibenwinkel

5.2 Umrechnungen für die Hardware

Einer der beiden Blöcke, die die $x_{i,\text{rad}}$ als Eingabe erhalten, soll diese in die für die Hardware notwendige Form bringen. C++, bzw. EtherCat benötigen eine Eingabe der Zielpositionen in Zehntel Grad. Die Umrechnung findet in diesem Block statt. Der Simulationsblock (Abschnitt 5.4) benötigt dies nicht, da er ohne Probleme mit der Vorgabe des Klappenwinkels in Radiant arbeitet. Zunächst findet die Umrechnung der Klappenwinkel in Grad statt:

$$x_{i,\text{deg}} = \frac{180}{\pi} x_{i,\text{rad}} \tag{5.5}$$

Des Weiteren müssen die Vorgabewerte für C++ als Winkel der Scheibe, die die Klappe bewegt, und nicht als Winkel der Klappe vorliegen. Das nichtlineare Verhältnis zwischen Klappen- und Scheibenwinkel wird im nächsten Schritt implementiert. Dafür wurden im CAD-Modell des Prototypen für 30 Positionen jeweils Klappen- und Scheibenwinkel gemessen. Mit Hilfe einer lookup-Tabelle kann in Modelica zwischen diesen Wertepaaren interpoliert werden. Daraus ergibt sich eine Funktion (siehe Abbildung 5.6), die den vorgegebenen Klappenwinkel auf einen Scheibenwinkel abbildet. Zu erkennen ist, dass die ermittelte Funktion im mittleren Arbeitsbereich nahezu linear ist und nur an den Grenzen Nichtlinearitäten aufweist. Dort muss die Scheibe deutlich weiter ausgelenkt werden für eine entsprechende Klappenauslenkung als im mittleren Teil.
Danach werden die erhaltenen Winkel mit 10 multipliziert, um die gewünschte Darstellung in Zehntel Grad zu erhalten. Abschließend werden die Werte auf ganze Zahlen gerundet, da EtherCat den Datentyp Integer benötigt. Daraus folgt, dass die Zielpositionen für die Scheibe nur auf Zehntel Grad genau vorgegeben werden

können. Dies ist allerdings ausreichend, da bei der aktuell verwendeten Scheibe im Durchschnitt ein Zehntel Grad der Scheibe nur etwas mehr als einem Hundertstel Grad der Klappe entspricht.
Die berechneten Zielpositionen $x_{i,\text{disc}}$ in der korrekten Einheit und dem korrekten Datentyp werden dem Block, der für die Serververbindung zuständig ist übergeben.

5.3 Serververbindung und Kontrollvariablen

Dieser Block erhält die zu veröffentlichenden Zielpositionen $x_{i,\text{disc}}$ als Eingang und schreibt sie auf den Modelnet-Server. Dafür muss jeweils der korrekte Name der Variable angegeben werden, sodass er mit dem Namen in C++ übereinstimmt. Andernfalls werden nicht die korrekten Werte an die C++-Seite weitergegeben. Neben den Positionswerten schreibt der Block auch drei Kontrollvariablen auf den Server.
Die erste ist das Alive-Signal. Die Idee hinter diesem ist zu überprüfen, ob das Modelica-Modell Werte auf den Server schreibt oder nicht. Falls Modelica keine Werte mehr auf den Server schreibt oder die Verbindung abbricht, würde C++ ohne das Alive-Signal weiterhin Werte einlesen und an den Motor übergeben. Diese Werte sind aber aufgrund der fehlenden Verbindung zu Modelica nicht aktuell, weshalb ein Betrieb in diesem Fall verhindert werden soll. Das Alive-Signal ist eine sich durchgehend ändernde Variable. Realisiert ist dies in Modelica durch eine Rampenfunktion, die bis zu einem Maximum steigt und dann wieder von vorne beginnt. Auf der C++-Seite wird das Alive-Signal empfangen und mit dem vorherigen empfangenen Wert verglichen. Stimmen die beiden Werte überein, bedeutet dies, dass das Alive-Signal auf dem Server nicht aktualisiert wurde und somit Modelica keine Werte mehr auf den Server schreibt. Sollte dies eintreten, fahren die Klappen des G-Seat in ihre Nullposition zurück und die Motoren werden deaktiviert. Wird die Verbindung zu Modelica wieder aktiv, werden die Motoren wieder aktiviert.
Die zweite Kontrollvariable ist der Safety Shutdown. Diese dient als Sicherheitsmechanismus um die Motoren jederzeit abschalten zu können. Es handelt sich um eine bool-Variable, die durch das Drücken der Return-Taste auf der Tastatur geändert wird. Daher besitzt der Block auch den Status der Return-Taste als Eingang. Durch das Drücken der Return-Taste werden die Motoren abgeschaltet, nachdem sie, wie bei fehlendem Alive-Signal, in die Nullposition zurückgefahren sind. Ein erneutes Drücken lässt die Motoren wieder starten.
Die dritte Kontrollvariable ist für den Start des Betriebs zuständig. Die Start Homing Variable ist ebenfalls ein bool-Wert. Wenn Start Homing nach Start des Modells zum ersten Mal auf 1 gesetzt wird, beginnt erst der Prozess der Referenzfahrt der bei jedem Neustart des C++-Programms automatisch durchgeführt wird. Somit kann der Start des Betriebs beliebig und unabhängig vom Start des C++-Programms mit dem Modelica-Modell bestimmt werden.
Durch diese Variablen wird also ein sicherer und gut steuerbarer Betrieb gewährleistet.

5.4 Simulation der Klappen

Im grünen Block finden alle Prozesse statt, die für die Simulation der Klappen benötigt werden. Dadurch ist es möglich das Verhalten des G-Seats zu testen und zu visualisieren ohne dafür den Prototypen benutzen zu müssen. Außerdem können beliebige Konfigurationen des G-Seats mit beliebig vielen Aktuatoren an beliebigen Positionen in der Simulation erzeugt werden. Während den Tests in dieser Arbeit wurde mit der Konfiguration gearbeitet, welche in Abbildung 5.2 zu Beginn des Kapitels zu sehen ist. Diese besteht aus zwei Klappen für die Sitzfläche und zwei für die Rückenlehne. Die relative Position der Aktuatoren kann dabei beliebig über Variablen eingestellt werden. Eingestellt werden kann die Höhe der Rückenklappen über den Sitzklappen, der Abstand zwischen Rücken- und Sitzklappen in y-Richtung, der Abstand zwischen den beiden Sitz-, bzw. Rückenklappen selbst und der Winkel, mit welchem die Rückenklappen im Vergleich zu den Sitzklappen gedreht sind, sodass die Rückenlehne variabel ist und nicht zwingend im 90°-Winkel zur Sitzfläche stehen muss. In den Tests dieser Arbeit wurde ein Winkel von 120° eingestellt.
Die Visualisierung findet mithilfe des DLR SimVis-Tools statt [KHB21]. Abhängig von den eingegebenen Parametern werden die Aktuatoren im Raum platziert. Die simulierte Bewegung der Klappen wird durch das Platzieren von Drehgelenken realisiert, sodass sich jede Klappe um die korrekte Achse dreht. Die Drehgelenke bewegen sich nach der Vorgabe der Positionen in Radiant $x_{i,\mathrm{rad}}$, welche die Eingänge des Blocks bilden. Die aktuelle Position jeder Klappe wird dann während der Simulation aus dem Block zum Washout-Block zurückgeführt (siehe Abbildung 5.1), welcher diese dann nutzt um die aktuellen Kraftrichtungen zu bestimmen.

5.5 Erzeugen von Eingangsfunktionen

Der letzte verbleibende Block generiert Testwerte für den Beschleunigungsvektor als in Eingang in das Modelica-Modell. Wenn der G-Seat später im Einsatz ist, soll der Beschleunigungsvektor $\boldsymbol{a}_{\mathrm{in}}$ durch eine Bewegungssimulation vorgegeben und über Modelnet eingelesen werden. Im aktuellen Stadium müssen die Eingänge allerdings noch separat erzeugt werden.
Über den Block können einige Testfunktionen eingestellt werden, wie z.B. eine Sprungfunktion, Rechteckfunktion oder ein Sinus. Dabei können Parameter, wie Amplitude, Frequenz bei Sinusfunktionen oder Start- und Endzeitpunkte, eingestellt werden. Außerdem kann die Achse ausgewählt werden, auf welche die Eingangsfunktion wirken soll. Hierbei können auch mehrere Achsen gleichzeitig gewählt werden.
Aus den getroffenen Eingaben erzeugt der Block dann den Eingangsvektor $\boldsymbol{a}_{\mathrm{in}}$, welcher in den Washout-Block eingeht und die Reaktion des Systems auf die gewählte Eingangsfunktion kann beobachtet werden.
Des Weiteren können durch diesen Block komplexere Flugtrajektorien vorgegeben werden. Dafür werden die Beschleunigungen aus der Log-Datei eines simulierten Flugmanövers ausgelesen. Ebenso besteht hier nun die Möglichkeit die Anteile der

Beschleunigungen, die vom RMS dargestellt werden, abzuziehen und nur die übrigen Beschleunigungen zu verwenden. Wird der G-Seat als stand alone Lösung verwendet, wie auch in den Tests dieser Arbeit, werden in diesem Schritt nur die unterschiedlichen Gravitationsvektoren verrechnet. Das ist notwendig, da bei einem feststehenden G-Seat der Gravitationsvektor konstant in negative z-Richtung zeigt, während er im simulierten Flugzeug durch das Rollen und Nicken im Cockpitkoordinatensystem in eine andere Richtung zeigt. Dies geschieht, indem zunächst die im Cockpit des Flugzeugs durch den Gravitationsvektor wahrgenommenen Beschleunigungen addiert werden und dann die Beschleunigungen, die im feststehenden G-Seat durch die Gravitation empfunden werden, wieder subtrahiert werden.

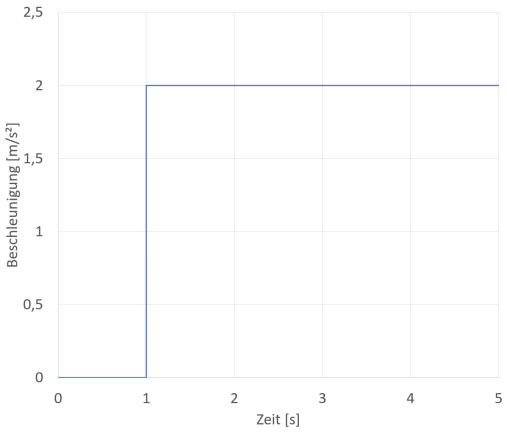
Kapitel 6

Testen des Force Cueing Algorithmus

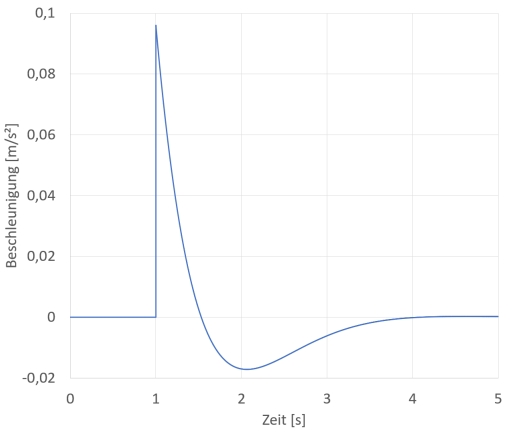
Nun soll der entworfene Force Cueing Algorithmus mithilfe von verschiedenen Eingängen getestet werden. Dafür werden die Aktuatorpositionen analysiert und abhängig von der jeweiligen Eingangsfunktion auf deren Plausibilität geprüft.

6.1 Antwort auf einfache Eingangsfunktionen

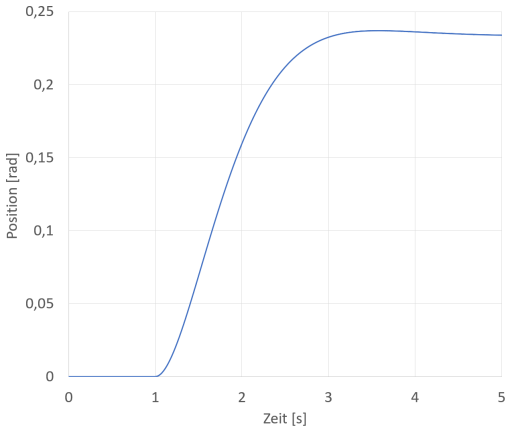
Zunächst wird die Antwort des Algorithmus auf eine Sprungfunktion (Abbildung 6.1) und eine Rechteckfunktion (Abbildung 6.2) betrachtet. Die Testfunktion wird dabei jeweils nur auf die z-Achse gegeben, da die beiden Sitzklappen größtenteils durch die z-Beschleunigungen ausgelenkt werden und somit das Verhalten des Algorithmus, insbesondere des Washout-Filters, sehr gut analysiert werden kann. Die betrachteten Plots sind somit jeweils die Positionen und Beschleunigungen einer Sitzklappe bei Vorgabe der Testfunktion nur in z-Richtung.
Die verwendete Sprungfunktion springt bei 1s auf eine konstante Beschleunigung von $2\frac{\mathrm{m}}{\mathrm{s}^2}$. Die resultierende Aktuatorbeschleunigung, die durch den Washout-Filter erzeugt wird, zeigt das für diesen Filter typische Verhalten mit einem Sprung zum selben Zeitpunkt wie die Eingangsbeschleunigung und einem anschließenden Abklingen. Die Sitzklappe wird dementsprechend ab diesem Zeitpunkt ausgelenkt. Da nach dem Sprung weiterhin eine konstante Beschleunigung vorhanden ist, bewegt sich die Klappe nicht in ihre Ausgangslage zurück, sondern verweilt im ausgelenkten Zustand.
Auch die Rechteckfunktion springt bei 1s auf $2\frac{\mathrm{m}}{\mathrm{s}^2}$ und springt dann nach weiteren 0,5s wieder zurück. Es tritt also nur eine konstante Beschleunigung für eine halbe Sekunde auf. Der erste Sprung der Aktuatorbeschleunigung verläuft dementsprechend gleich wie bei der Sprungfunktion. Da es sich bei dem Washout-Filter um einen Hochpass handelt, hat auch die fallende Flanke der Eingangsbeschleunigung eine Auswirkung auf die Aktuatorbeschleunigungen und erzeugt den gleichen Sprung wie die positive Flanke, nur in den negativen Bereich. Durch die negative



(a) Eingangsfunktion in z-Richtung



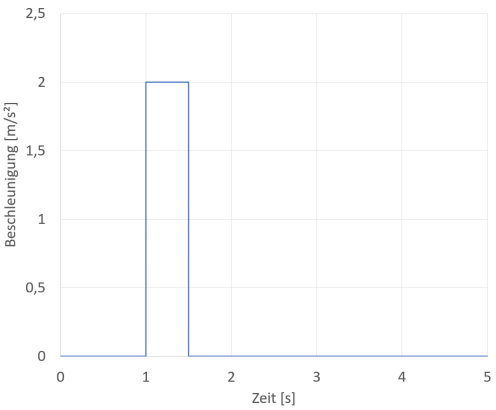
(b) Aktuatorbeschleunigung der rechten Sitzklappe



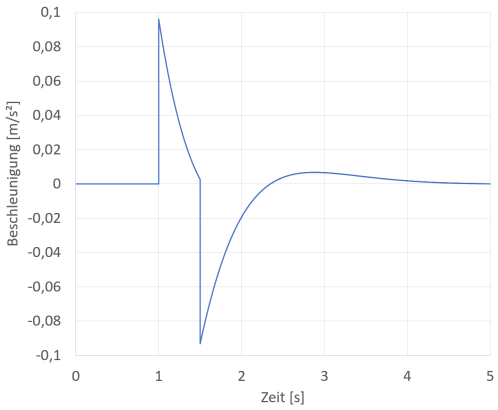
(c) Aktuatorposition der rechten Sitzklappe

Abbildung 6.1: Verwendung einer Sprungfunktion in z-Richtung als Eingang

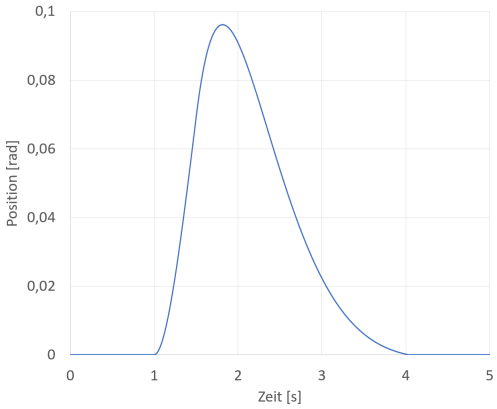
Beschleunigung ist dann auch der Washout-Effekt im Graphen der Position zu beobachten. Die Klappe bleibt nicht mehr in ihrer ausgelenkten Position wie bei der Sprungfunktion, sondern bewegt sich wieder in ihre Ausgangslage zurück und kann sich daher bei kommenden Beschleunigungseingängen wieder neu auslenken, um den entsprechenden Bewegungseindruck beim Insassen zu erzeugen.



(a) Eingangsfunktion in z-Richtung



(b) Aktuatorbeschleunigung der rechten Sitzklappe



(c) Aktuatorposition der rechten Sitzklappe

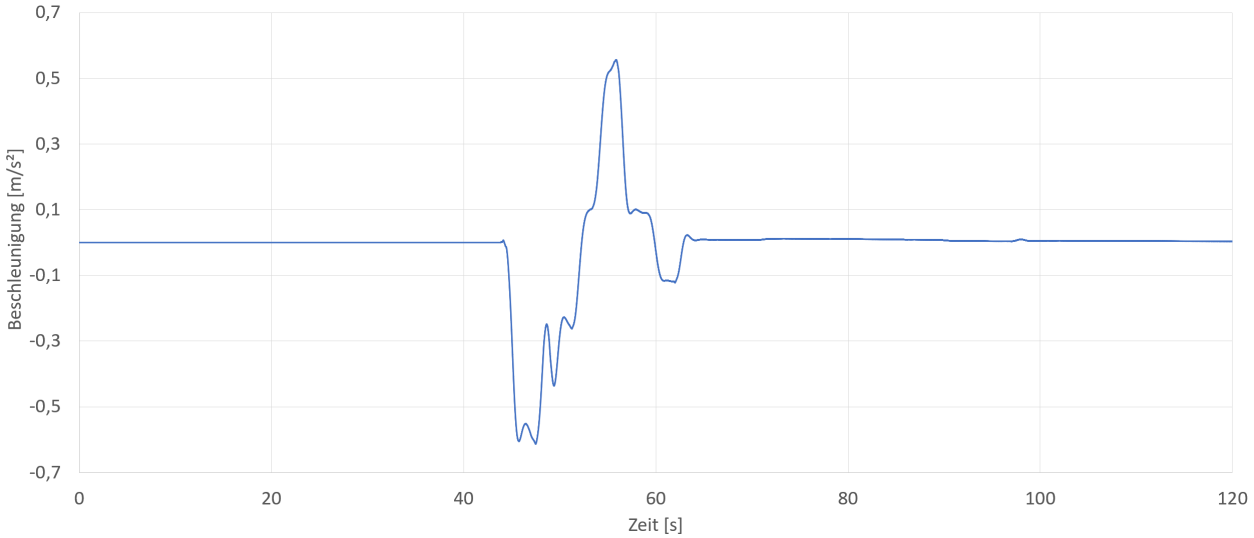
Abbildung 6.2: Verwendung einer Rechteckfunktion in z-Richtung als Eingang

6.2 Test mit einer Flugtrajektorie

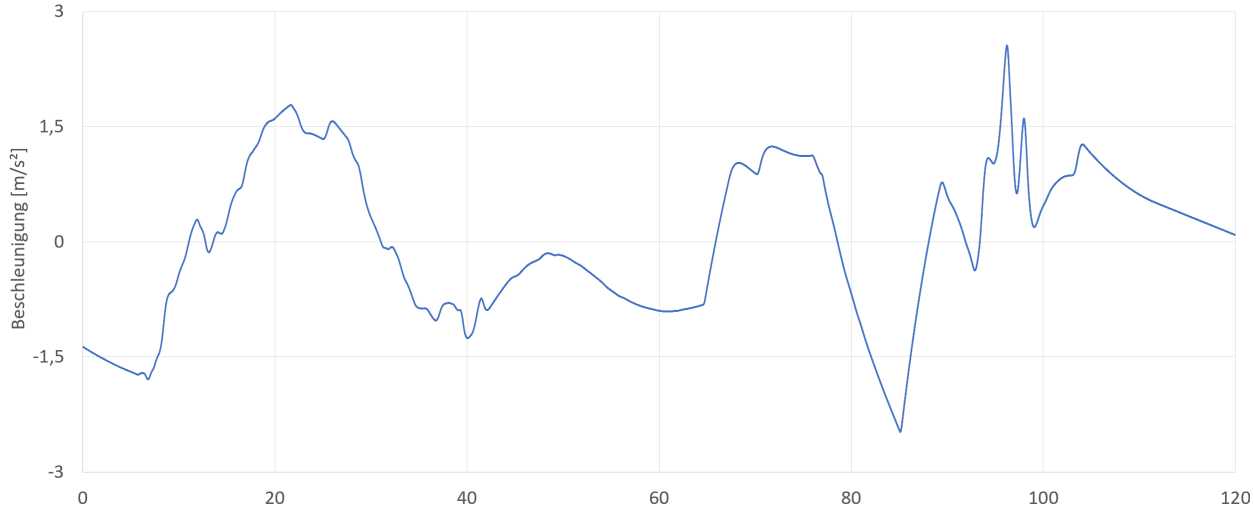
Als weiterer Test der Klappen wurde nun keine einfache Eingangsfunktion mehr erzeugt, sondern der Eingangsvektor für den G-Seat wurde aus einer Flugsimulation gewonnen. Die daraus resultierenden Beschleunigungen, die der G-Seat abbilden soll, sind in Abbildung 6.3 dargestellt.

6.2.1 Beschreibung des Manövers

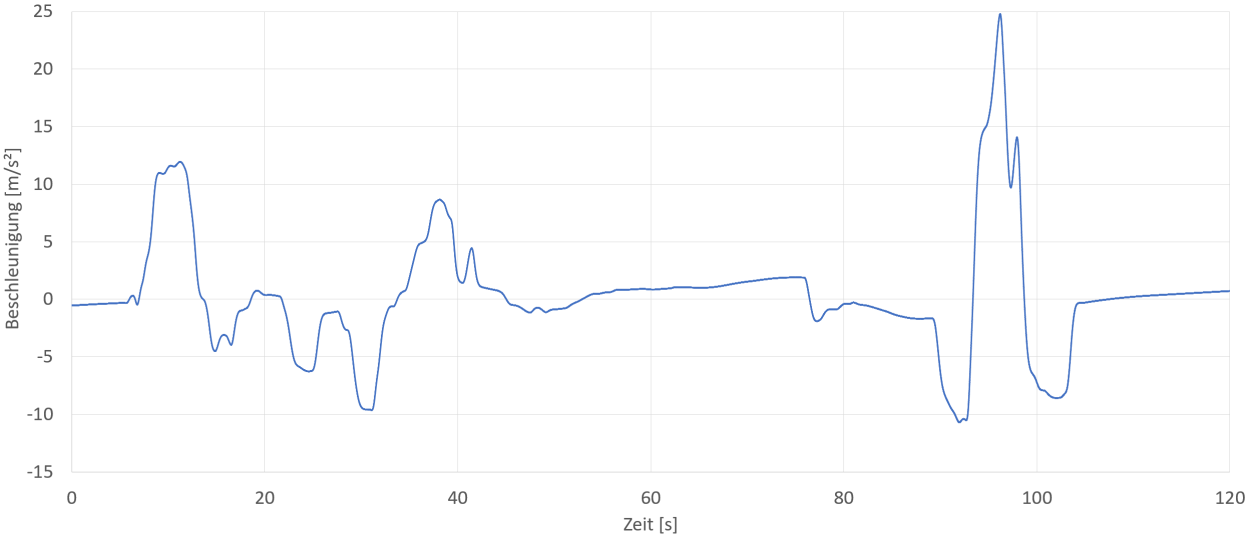
Anhand der Beschleunigungen in die x-, y- und z-Richtung kann der Verlauf des Manövers sehr gut abgelesen werden. Zu erkennen durch die Kurve der z-Beschleunigung ist, dass der Pilot zu Beginn im Bereich bis ca. 40s das Flugzeug steigen und wieder sinken lässt. Der Verlauf der Kurve der y-Richtung, die aus



(a) Beschleunigung in x-Richtung



(b) Beschleunigung in y-Richtung



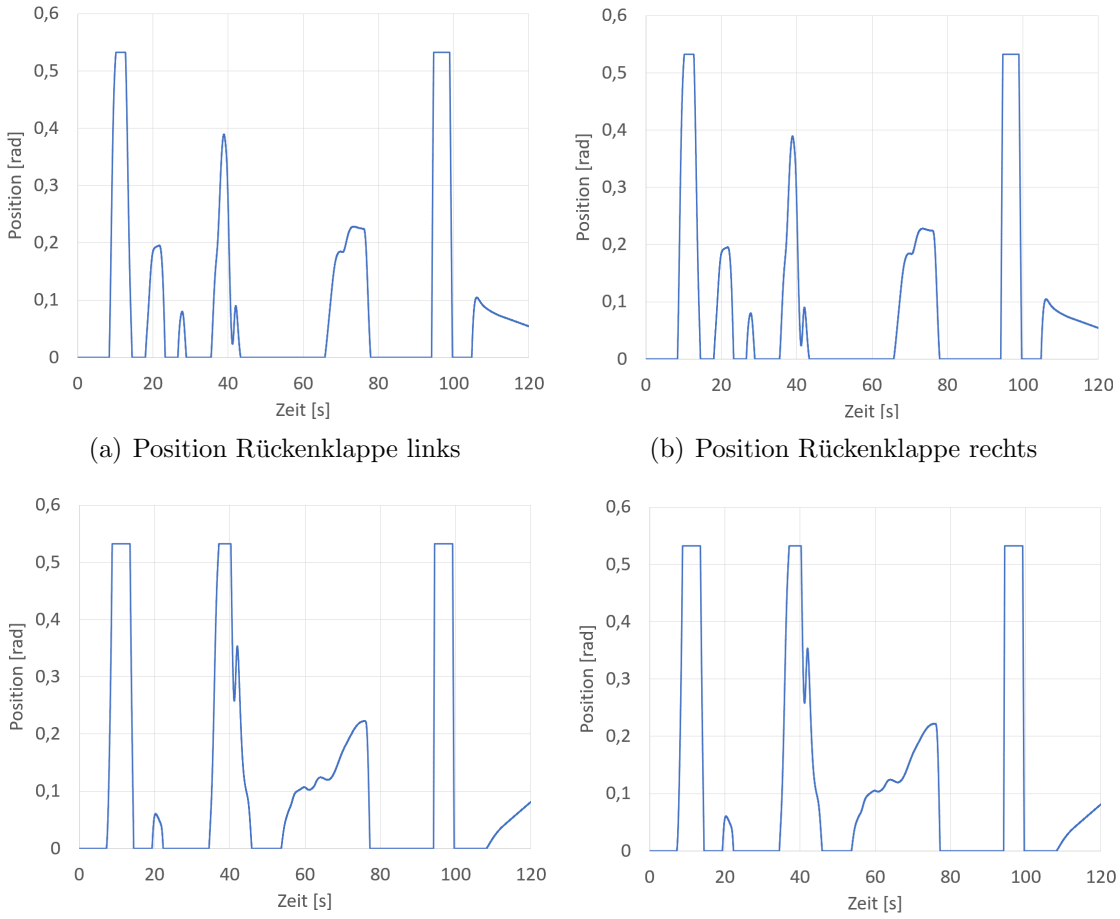
(c) Beschleunigung in z-Richtung

Abbildung 6.3: Darzustellende Beschleunigungen bei einer Flugsimulation

der Sicht des Piloten nach vorne definiert ist, liefert Informationen über die Geschwindigkeitsänderungen des Flugzeugs. Daran ist zu erkennen, dass das Flugzeug im Teil bis ca. 40s zuerst durch den Steigflug abbremst, dann durch das Sinken wieder beschleunigt und danach wieder an Geschwindigkeit verliert. Anschließend zeigt die Kurve der x-Beschleunigung seinen einzigen von 0 abweichenden Verlauf. Daraus kann gefolgert werden, dass hier Kurven geflogen werden. Allerdings sind diese x-Beschleunigungen sehr gering im Vergleich zu den anderen beiden Richtungen. Das liegt daran, dass sich die Beschleunigung, die man aufgrund des Kurvenflugs in x-Richtung wahrnehmen würde, fast komplett durch den Gravitationsvektor aufhebt. Im nächsten Abschnitt von ca. 65-90s gibt es keine großen Ausschläge in den x- und z-Kurven. Zu erkennen an der y-Richtung ist, dass der Pilot hier zunächst beschleunigt und dann etwas stärker abbremst. Gleichzeitig erfährt der Insasse auch leichte Beschleunigungen in z-Richtung, da die Höhe in diesem Abschnitt nicht ganz konstant gehalten wurde. Zum Abschluss des Manövers wird noch ein aggressives Pitch-Manöver geflogen, bei welchem zunächst gesunken und sofort wieder gestiegen wird. Die starke positive Beschleunigung in z-Richtung beim Fliegen des Manövers ist deutlich erkennbar. Ebenfalls ist eine größtenteils positive Geschwindigkeitsänderung zu erkennen.

6.2.2 Analyse der Aktuatorpositionen

Nun sollen die Auslenkungen der Aktuatoren im Lauf der Flugtrajektorie betrachtet werden. Dabei wurde die in Kapitel 5.4 beschriebene Konfiguration verwendet. Abweichend von der maximalen Auslenkung, die durch die aktuell im Prototyp eingebaute Scheibe möglich ist, wurde für diesen Test das Limit auf 30,5° gesetzt, was ungefähr 0,53 rad entspricht, um die Bewegungen besser analysieren zu können. Die Positionen der Aktuatoren sind in Abbildung 6.4 zu sehen. Da die Kraftrichtung der beiden Sitzklappen in ihrer Ausgangslage nur in z-Richtung zeigt, reagieren die beiden Sitzklappen auch nahezu nur auf die positiven z-Beschleunigungen. Die negativen z-Beschleunigungen können daher mit dieser Konfiguration nicht dargestellt werden, wenn eine Klappe sich in der Ausgangslage befindet, und haben somit auch keine Auswirkung auf die Position der Sitzklappen. Bei positiven z-Beschleunigungen ist die erwartete Auslenkung der Sitzklappen zu erkennen. Betrachtet man alle Zeitpunkte, zu denen die Sitzklappen ausgelenkt werden, so stimmen diese mit allen Zeitpunkten überein, zu denen eine positive z-Beschleunigung vorhanden ist. Ebenfalls auffällig ist, dass der Verlauf der Kurven der Sitzklappen fast identisch ist. Der Grund hierfür ist, dass sie sich bei z-Eingängen exakt gleich verhalten. Lediglich die x-Beschleunigungen könnten ein unterschiedliches Verhalten verursachen, wenn sie im ausgelenkten Zustand der Klappen auftreten. Da diese aber kaum auftreten und gleichzeitig sehr klein im Vergleich zu den z-Beschleunigungen sind, nehmen die x-Beschleunigungen nahezu keinen Einfluss auf die Position der Sitzklappen.
Die beiden Rückenklappen können aus ihrer Ausgangslage heraus durch die schräge Positionierung bereits Beschleunigungen in y- und z-Richtung abbilden. Bei Be-



(a) Position Rückenklappe links

(b) Position Rückenklappe rechts

(c) Position Sitzklappe links

(d) Position Sitzklappe rechts

Abbildung 6.4: Positionen der Aktuatoren bei Verwendung der Flugtrajektorie

schleunigungen in diese Raumrichtungen verhalten sie sich exakt gleich. Eine unterschiedliche Auslenkung wird auch hier nur durch x-Eingänge verursacht. Aus denselben Gründen wie bei den Sitzklappen ist dies aber kaum der Fall. Die Position der Rückenklappen ist daher auch nahezu identisch. Die Auslenkungen durch die positiven z-Beschleunigungen sind ebenfalls deutlich zu erkennen. Zusätzlich dazu reagieren die Rückenklappen auch auf das Beschleunigen, bzw. Bremsen des Flugzeugs, wodurch sich der Kurvenverlauf von dem der Sitzklappen unterscheidet. Sehr gut zu erkennen ist dies an den Auslenkungen bei 20s und 40s. Bei 20s liegt bei den Sitzklappen nur ein leichter Ausschlag vor, während die Rückenklappen aufgrund der Geschwindigkeitszunahme zu diesem Zeitpunkt stärker ausgelenkt werden. Bei 40s ist es dann andersherum. Da hier das Flugzeug während der positiven z-Beschleunigung langsamer wird, öffnen sich die Rückenklappen weniger als die Sitzklappen. Die durch die z-Beschleunigung verursachte Bewegung der Sitzklappen bei 60s wird bei den Rückenklappen von der gleichzeitig auftretenden negativen y-Beschleunigung sogar komplett kompensiert.

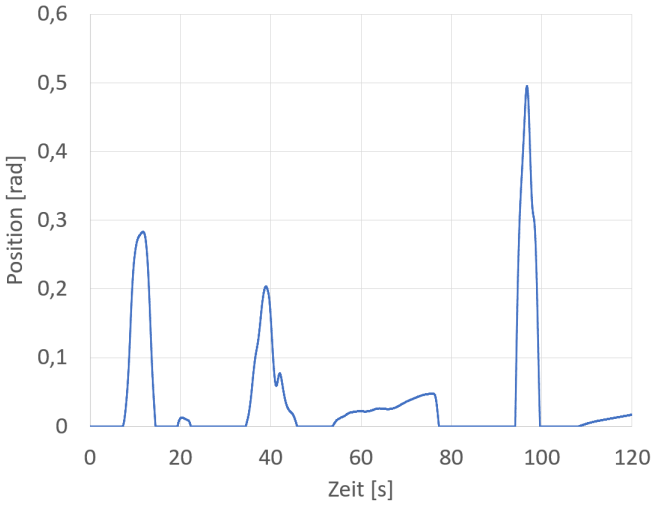


Abbildung 6.5: Aktuatorposition Sitzklappe rechts bei Verwendung der Flugtrajektorie mit angepasster Skalierung

Des Weiteren ist auffällig, dass die Kurven sehr häufig den eingestellten Grenzwert erreichen. Zum einen kann man dadurch schlussfolgern, dass die in den Algorithmus eingebauten Limitierungen ihren Zweck erfüllen, zum anderen bedeutet dies aber auch, dass die Klappen dort in ihre maximale Auslenkung fahren und dort verweilen, obwohl sie, wenn es möglich wäre, sich eigentlich noch weiter auslenken sollten.

6.2.3 Skalierung des Washout-Filters

Um dem häufigen Erreichen der Grenzen entgegenzuwirken kann der verwendete Washout-Filter entsprechend skaliert werden, sodass möglichst alle berechneten Auslenkungen auch in den Arbeitsbereich passen. Andererseits werden dadurch kleinere Auslenkungen ebenfalls herunterskaliert, wodurch sie möglicherweise nur noch wenig oder gar nicht mehr wahrgenommen werden. Bisher wurde ein konstanter Faktor von $K = 0,048$ verwendet. Dieser wurde nun auf $K = 0,01$ reduziert um die komplette Klappenbewegung im Arbeitsraum abbilden zu können. Abbildung 6.5 zeigt die Position der rechten Sitzklappe bei Verwendung derselben Flugtrajektorie mit geänderter Skalierung des Washout-Filters. Dabei zeigt sich auch das Problem, dass kleine Auslenkungen kaum mehr auftauchen. Die Auslenkung bei ca. 20s beispielsweise beträgt im skalierten Fall nur noch weniger als 1°. Welche Einstellung eine bessere Wahrnehmung liefert, kann daraus nicht abgelesen werden. Letztendlich muss ein Kompromiss zwischen idealem Ausnutzen des Arbeitsraums und besserer Wahrnehmung kleinerer Auslenkungen gefunden werden und entschieden werden, welche Einstellung zu einer möglichst realitätsnahen Wahrnehmung durch den Piloten führt.

6.3 Bewertung der Ergebnisse

Betrachtet man alle Tests aus diesem Kapitel, so verhalten sich die Klappen für alle betrachteten Fälle plausibel. Sowohl für einfache Eingangsfunktionen, als auch für den komplexeren Verlauf einer Flugtrajektorie, können alle durch den Algorithmus berechneten Klappenbewegungen anhand der Eingänge nachvollzogen werden und führen, soweit ohne Probandentest beurteilbar, zu den gewünschten Bewegungseindrücken. Das Verändern von Filterkoeffizienten hat ebenfalls den erwarteten Effekt. Sie können daher sehr gut verwendet werden, um den Algorithmus für verschiedene Anwendungsfälle zu optimieren. Es kann somit gefolgert werden, dass alle Komponenten des Force Cueing Algorithmus, insbesondere der Washout-Filter, ihren Zweck erfüllen und sinnvolle Werte für die Aktuatorpositionen berechnen. Es wurden aber auch Bereiche identifiziert, in denen die Klappen nicht den gewünschten Bewegungseindruck erzeugen können, wie negative y- oder z-Beschleunigungen oder jegliche x-Beschleunigungen. Die Ursachen dafür konnten anhand der Kraftrichtungen und Grenzen der Aktuatoren erklärt werden.

Kapitel 7

Zusammenfassung und Ausblick

Zusammenfassung

Im Rahmen der Arbeit wurde ein Ansteuerungsalgorithmus für die Aktuatoren eines G-Seats für den DLR Robotic Motion Simulator (RMS) entwickelt. Bei den Aktuatoren handelt es sich um bewegliche Klappen unter den Sitzpolstern, die Druck auf den Körper des Insassen ausüben. Dadurch wird ein Bewegungseindruck erzeugt.
Zunächst wurden dafür die Motoren eines Hardware Prototyps mithilfe des echtzeitfähigen Bussystems EtherCat in Betrieb genommen. Es konnte gezeigt werden, dass die Motoren bis zu einer Frequenz von 2Hz den kompletten Arbeitsbereich ohne Einschränkung befahren können. Bei höheren Frequenzen fällt die maximal befahrbare Amplitude deutlich ab.
Zusätzlich wurde die Datenkommunikation in Echtzeit zwischen dem Force Cueing Algorithmus und dem C++ Programm, über welches die EtherCat-Schnittstelle realisiert wurde, über einen Modelnet-Server implementiert.
Der entwickelte Force Cueing Algorithmus arbeitet mit einem klassischen Washout-Filter. Außerdem projeziert er den eingehenden Beschleunigungsvektor auf die Kraftrichtungen der einzelnen Aktuatoren. Diese verändern sich aufgrund der Klappendynamik während des Betriebs ständig. Der Algorithmus kann den G-Seat sowohl als alleinigen Aktuator als auch in Kombination mit dem Motion Cueing des RMS ansteuern. In zweiterem Fall bildet der G-Seat nur jene Beschleunigungen ab, welche nicht durch den RMS ausgeführt werden.
Beim Testen des Algorithmus mit einfachen Eingangsfunktionen und mit einer Flugtrajektorie konnte gezeigt werden, dass plausible Klappenbewegungen erzeugt werden können. Außerdem wurde festgestellt, dass der G-Seat abhängig von der verwendeten Konfiguration und der Position der Klappen manche Beschleunigungen nicht darstellen kann.
Zusammenfassend lässt sich sagen, dass die entwickelte Struktur aus Force Cueing Algorithmus und Ansteuerung des Prototyps eine sehr gute Grundlage für weitere Arbeiten am G-Seat liefert.

Ausblick

In der Zukunft muss noch das Tuning des Force Cueing Algorithmus durchgeführt werden. Die aktuell verwendeten Koeffizienten der Washout-Filter sind nicht auf diese Form eines G-Seats angepasst. Diese können mithilfe von Testpersonen, beispielsweise erfahrenen Piloten, noch optimiert werden. Dafür müssen verschiedene Parametrierungen des Filters getestet und die Werte anhand des Feedbacks der Piloten angepasst werden, sodass eine möglichst gute Wahrnehmung der Bewegung entsteht. Zu den einzustellenden Größen zählt auch auch die im vorherigen Kapitel angesprochene Skalierung. Auch können die in Kapitel 3.2 gewonnenen Informationen über das Systemverhalten des Motors miteingearbeitet werden, sodass Frequenzen, die der Motor nicht abbilden kann, bereits vom Algorithmus herausgefiltert werden.
Des Weiteren ist die Entwicklung von Konzepten denkbar, die Nullposition der Klappen in die Mitte ihres Arbeitsraums zu verlegen und die Klappen dementsprechend anders im Raum zu positionieren. Dies würde den Effekt mit sich bringen, dass die Klappen aus ihrer Ausgangslage heraus sich sofort in positive und negative Richtung bewegen könnten. Dadurch wäre beispielsweise ein Sinkflug aus der Flugtrajektorie aus Kapitel 6.2 besser darstellbar, auch wenn sich die Klappen in ihrer Ausgangslage befinden. Andererseits würde man dadurch an Amplitude in positiver Richtung einbüßen, da der Arbeitsraum natürlich gleich groß bleibt.
Auf lange Sicht gesehen soll dann eine Konfiguration eines solchen G-Seats in den RMS des DLR eingebaut werden. Der G-Seat soll dann parallel zum Motion Cueing arbeiten und den Bewegungseindruck für den Insassen verbessern, indem er Bewegungen repräsentiert, die für den RMS schwer auszuführen sind.

Anhang A

EtherCat

EtherCat ist ein echtzeitfähiges Bussystem. Über dieses System kann ein Master mit seinen Slaves kommunizieren. Zu Beginn des Betriebs können der Master und alle Slaves am Bus mithilfe ihrer Position registriert und dadurch später angesteuert werden.
Der Datenaustausch findet durch das Lesen und Schreiben von den SDOs (Service Data Objects) und PDOs (Process Data Objects) statt. Bei den SDOs handelt es sich um Werte die zu Beginn des Betriebs eingestellt werden und dann währenddessen konstant bleiben. Beispiele hierfür sind die Eingabe von Motordaten, die angegeben werden müssen, um einen Motor als Slave zu benutzen, oder die Festlegung von Konstanten, wie Begrenzung verschiedener Größen, z.B. Motorstrom, Drehmoment, Position oder Geschwindigkeit. Die PDOs hingegen sind Variablen, die sich während des Betriebs in Echtzeit ändern. Dabei wird unterschieden zwischen den Receive-PDOs, die vom Master geschrieben werden und dem Slave als Information dienen, beispielsweise Zielposition oder zu verwendender Betriebsmodus, und den Transmit-PDOs, auf die der Master keinen Schreibzugriff besitzt und die lediglich als Rückgabewerte von den Slaves dienen. Beispiele hierfür sind die Ausgaben des aktuellen Status, Position, Geschwindigkeit oder Beschleunigung des Slaves. Die PDOs werden zwar nur während des Echtzeitbetriebs gelesen und geschrieben, allerdings müssen alle gewünschten PDOs, auf die zugegriffen werden soll, zu Beginn am Bus registriert werden.
Der Zugriff auf die einzelnen Objekte erfolgt durch das Aufrufen entsprechender C++ Befehle, für die es eine eigene C++ Bibliothek gibt, in der diese definiert und beschrieben sind. Mithilfe dieser Bibliothek kann die EtherCat-Struktur auch über C++ aufgebaut werden. Jedes PDO/SDO besitzt eine eindeutige Speicheradresse, über welche es durch die EtherCat-Befehle je nach Zugriffsrecht gelesen oder geschrieben werden kann.
Somit wird durch ständiges Lesen und Schreiben der PDOs der Betrieb der Motoren in Echtzeit gewährleistet.

Abbildungsverzeichnis

Literaturverzeichnis

[ABQ+23] Houshyar Asadi, Tobias Bellmann, Mohammadreza Chalak Qazani, Shady Mohamed, Chee Peng Lim, and Saeid Nahavandi. A novel decoupled model predictive control-based motion cueing algorithm for driving simulators. *IEEE Transactions on Vehicular Technology*, 2023.

[AMN+15] Houshyar Asadi, Shady Mohamed, Kyle Nelson, Saeid Nahavandi, and Maysam Oladazimi. An optimal washout filter based on genetic algorithm compensators for improving simulator driver perception. In *DSC: Proceedings of the Driving Simulation Conference & Exhibition*, 2015.

[BBB+11] Mauro Baseggio, Alessandro Beghi, Mattia Bruschetta, Fabio Maran, and Diego Minen. An MPC approach to the design of motion cueing algorithms for driving simulators. In *International IEEE conference on intelligent transportation systems (ITSC)*, 2011.

[BCC+18] Mattia Bruschetta, Yutao Chen, Daniel Cunico, Enrico Mion, and Alessandro Beghi. A nonlinear mpc based motion cueing strategy for a high performance driving simulator with active seat. In *IEEE International Workshop on Advanced Motion Control (AMC)*, 2018.

[Bel14] Tobias Bellmann. *Optimierungsbasierte Bahnplanung für interaktive robotische Bewegungssimulatoren*. PhD thesis, Universität der Bundeswehr München, 2014.

[BMB17] Mattia Bruschetta, Fabio Maran, and Alessandro Beghi. A fast implementation of mpc-based motion cueing algorithms for mid-size road vehicle motion simulators. *Vehicle system dynamics*, 2017.

[CBP+01] William W.Y. Chung, Norman J. Bengford, Charles H. Perry, Bob Nicholson, and Colin Wilkinson. *Investigation of effectiveness of the dynamic seat in a Black Hawk flight simulation*. 2001.

[GB10] Nikhil J.I. Garrett and Matthew C. Best. Driving simulator motion cueing algorithms–a survey of the state of the art. 2010.

[GR97] Peter R Grant and Lloyd D Reid. Motion washout filter tuning: Rules and requirements. *Journal of aircraft*, 1997.

[KHB21] Sebastian Kümper, Matthias Hellerer, and Tobias Bellmann. Dlr visualization 2 library-real-time graphical environments for virtual commissioning. In *Proceedings of 14th Modelica Conference*, 2021.

[Lor08] Tobias Lorenz. *Implementierung, Test und Bewertung eines zeitvarianten Algorithmus zur Ansteuerung einer Bewegungsplattform*. PhD thesis, Technische Universität Dresden, 2008.

[NRK92] M.A. Nahon, L.D. Reid, and J. Kirdeikis. Adaptive simulator motion software with supervisory control. *Journal of Guidance, Control, and Dynamics*, 1992.

[RK00] Gilles Reymond and Andras Kemeny. Motion cueing in the renault driving simulator. *Vehicle System Dynamics*, 2000.

[SGW+09] Hua Shao, Liwen Guan, Jinsong Wang, Liping Wang, and Yang Fu. An optimal force cueing algorithm for dynamic seat. In *IEEE International Conference on control and automation*, 2009.

[Sho80] Thomas W. Showalter. *The effects of motion and g-seat cues on pilot simulator performance of three piloting tasks*. 1980.

[Whi89] A.D. White. G-seat heave motion cueing for improved handling in helicopter simulators. In *Flight Simulation Technologies Conference and Exhibit*, 1989.

[ZKGT15] Daniel Zimmermann, Andreas Kohlstedt, Sandra Gausemeier, and Ansgar Trächtler. Entwicklung eines prädiktiven Motion-Cueing-Verfahrens für den ATMOS-Fahrsimulator. *Augmented & Virtual Reality in der Produktentstehung*, 2015.

License