Hochschule **Rosenheim**
University of Applied Sciences

Untersuchung verschiedener Modellierungsansätze anhand des Ball auf Kugel Experiments

Andreas Bernatzky 818849

Projektarbeit II im Masterstudiengang Angewandte Forschung und Entwicklung

Angefertigt im Labor für Regelungstechnik

Erstkorrektor der HS Rosenheim: **Prof. Dr.-Ing. Peter Zentgraf, M.Sc.**

Zweitkorrektor der HS Rosenheim: **Prof. Dr.-Ing. Stefan Schinagl**

Abgegeben am: _____

ERKLÄRUNG:

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim den, _____

Andreas Bernatzky

Kurzfassung und Zielsetzung:

Diese Projektarbeit wurde an der Hochschule Rosenheim im Sommersemester 2018 angefertigt. Das Labor für Mess- und Regelungstechnik besitzt einen Demonstrationsversuch, welcher in der Lage ist einen Ball auf einer Kugel, mithilfe von zwei Antriebsmotoren zu balancieren. Dieser Demonstrationsversuch wurde im Wintersemester 2015/16 von Herrn Leonhard Holzner im Rahmen seiner Projektarbeit I angefertigt. Die Ziele der jetzigen Projektarbeit lassen sich in drei Punkte unterteilen. Es sollte ein digitaler Zwilling in MATLAB/Simulink zu diesem Demonstrationsversuch erstellt werden. Eine Schnittstelle zur MATLAB/Beckhoff CPU soll geschaffen werden. Die Dokumentation des Prüfstands soll nachgeholt werden. Am 26.Juni.2018 wurde der Demonstrationsversuch auf der MATLAB Expo 2018 präsentiert.

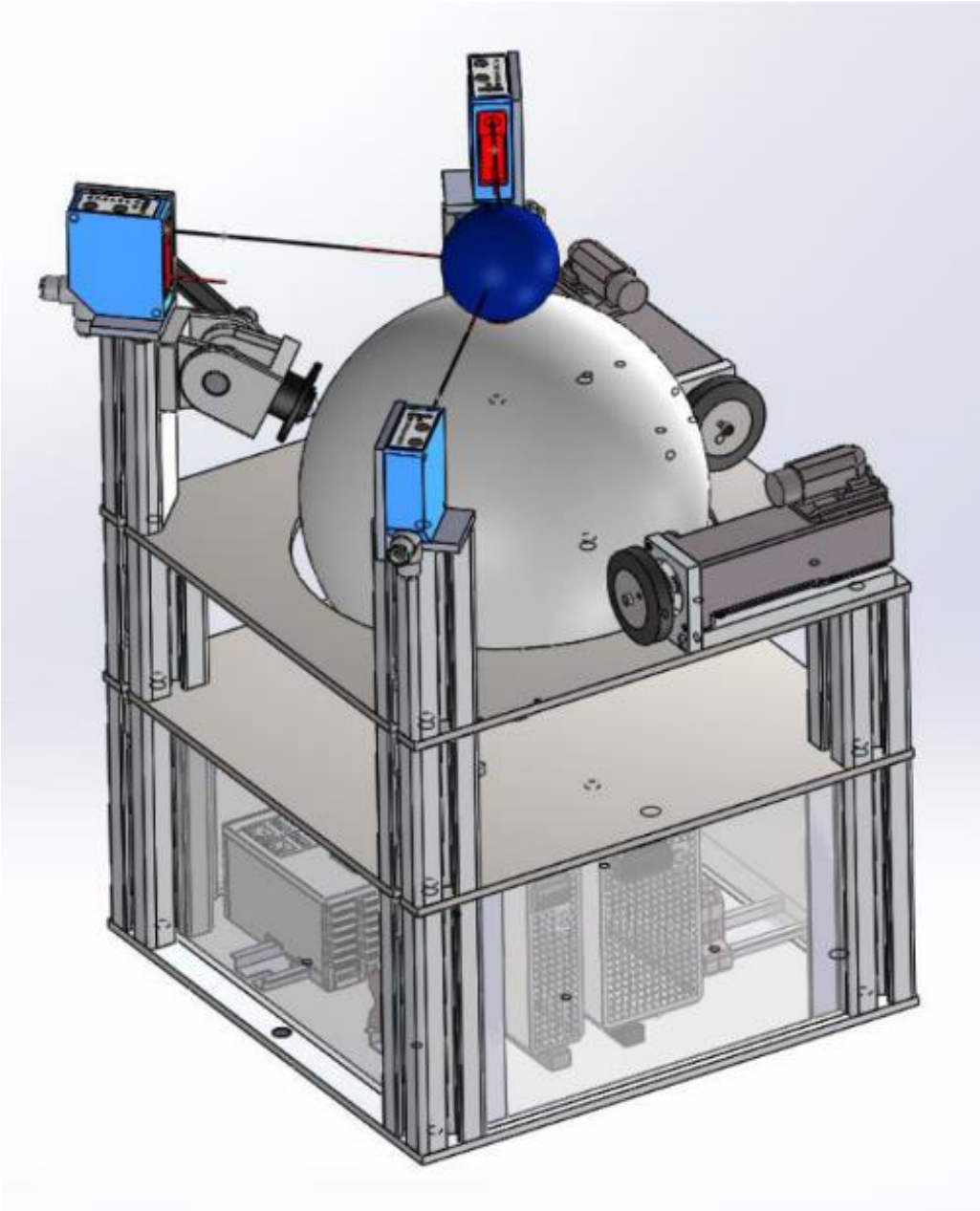

Abbildung 1: Ball auf Kugel Demonstrationsversuch in CAD (Holzner)e

Inhalt

Abbildungsverzeichnis

Abkürzungsverzeichnis:

J	Massenträgheitsmoment	kgm^2
m	Masse	kg
r	Radius	m
I_N	Nennstrom	mA
c_t	Motorkonstante	$\frac{mNm}{A}$
M_N	Nenndrehmoment	Nm
θ	Winkel des Balls	rad
θ'	Winkeländerungsgeschwindigkeit des Balls	$\frac{rad}{s}$
γ'	Kugeldrehzahl	$\frac{rad}{s}$
r_B	Radius Ball	m
r_K	Radius Kugel	m

1. Demonstrationsversuch Hardware Komponenten

Der Demonstrationsversuch kann in seiner Gesamtheit in folgende Komponenten unterteilt werden:

- Kugel
- Ball
- Motoren
- Laserabstandssensoren
- Beckhoff CPU

Es soll im Folgenden kurz auf diese 5 Punkte eingegangen werden, da diese die Kernkomponenten darstellen.

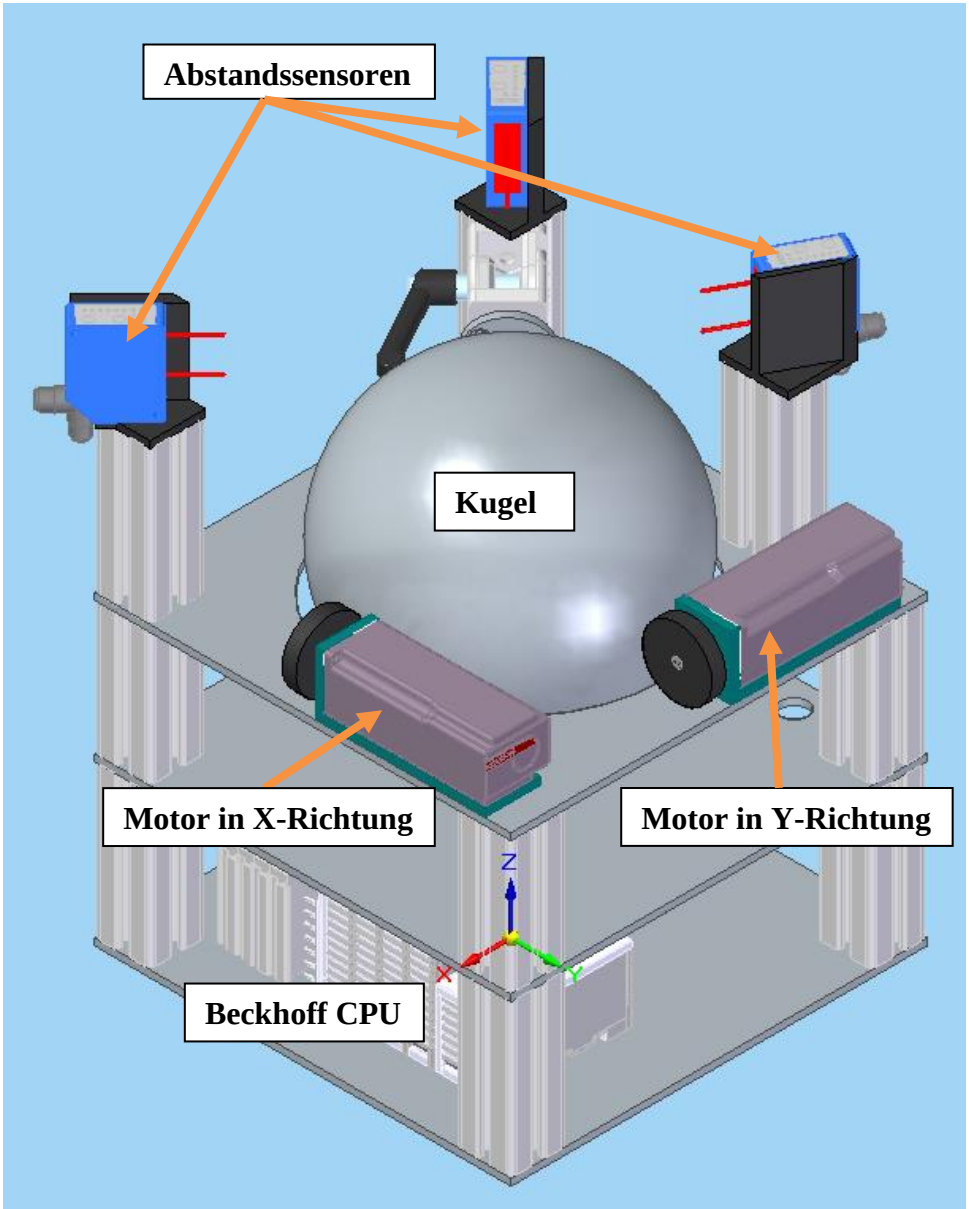

Abbildung 2: In Solid Edge für Dokumentationszwecke neu gezeichneter Demonstrationsversuch

1.1 Kugel und Ball

Bei der Kugel handelt es sich um eine im Pulversinterverfahren gefertigte Hohlkugel. Der Durchmesser der Kugel beträgt 210 mm, und Sie besitzt eine Wandstärke von 3 mm. Das Gewicht der Kugel beträgt 380,4 g. Die Drehmoment Übertragung, der Motoren erfolgt über die Kugel auf den Ball wirkend. Das Massenträgheitsmoment einer Vollkugel lässt sich mit nachfolgender Formel bestimmen:

$$J_{Voll} = \frac{2}{5} * m_K * {r_k}^2$$

Da es sich hier um eine Hohlkugel handelt muss das resultierende Massenträgheitsmoment für zwei Vollkugeln berechnet werden, welche voneinander abgezogen werden. Da das Material der Kugel nicht eindeutig bekannt ist, wird eine Dichte von 1,04 $\frac{g}{cm^3}$ angenommen. Diese Dichte entspricht der von ABS-Kunststoff (Filamentworld, 2018).

$$J_{Kugel} = J_{Voll_{außen}} - J_{Voll_{innen}}$$

$$J_{Kugel} = \left(\frac{2}{5} * m_{Kugel_{aussen}} * r_{Kugel}^2{}_{aussen}\right) - \left(\frac{2}{5} * m_{Kugel_{innen}} * r_{Kugel}^2{}_{innen}\right)$$

$$J_{Kugel} = 0{,}0030\ kgm^2$$

Es existiert eine Näherungsformel (Schumann, 2016) für Kugelschalen, welche mit dem Ergebnis $J_{Kugel} = 0{,}0028\ kgm^2$ eine gute Näherung darstellt.

Bei dem verwendeten Ball wurde ein Vollgummiball mit einem Durchmesser von 60 mm und einem Gewicht von 110,6 g verwendet.

1.2 Motor

Es existieren zwei Motoren vom Typ AM8112-wFyz. Diese Motoren besitzen ein Nenndrehmoment von 0,38 Nm. Die Motoren werden über eine Servo-Motorklemme vom Typ EL72x1 angesteuert.

Der verwendete Servomotor bildet zusammen mit einem Servoverstärker (Klemme EL72x1) und einem integrierten Resolver als Messeinheit, einen Servoantrieb. Servomotoren können in einem geschlossenen Regelkreis positions-, moment- oder geschwindigkeitsgeregelt betrieben werden. Der hier vorliegende Motor bietet folgende Regelmöglichkeiten (Beckhoff, 2015):

CST – cyclic synchronous torque
CSTCA – cyclic synchronous torque with commutation angle
CSV – cyclic synchronous velocity
CSP – cyclic synchronous position

Die im Demonstrationsversuch verwendeten Motoren befinden sich im CST-Betrieb. Hier kann den Motoren ein Drehmoment vorgegeben werden. Bei den Betriebsarten CSV und CSP handelt es sich um Geschwindigkeits- bzw. Positionsregelungen. Die CSTCA Betriebsart erlaubt es ein definiertes Drehmoment vorzugeben und einen Motorwinkel, welcher bei dem eingestellten Winkel gehalten wird. Genaue Daten zum Motor können im Anhang A auf der beiliegenden CD gefunden werden.

Während der Simulation war lange Zeit unklar, wie das eigentliche Stellmoment errechnet wird. Über die CPU kann in der Variable „Target torque“ ein definiertes Drehmoment vorgegeben werden, auf welches der Motor regeln soll. Das Moment wird hierbei als Tausendstel des Nennstroms angegeben. Die einstellbaren Motorstromwerte können je nach Einstellung, als Scheitelwert oder Effektivwert interpretiert werden.

$$M = (\frac{Torque\ Actual\ Value}{1000} * \frac{I_N}{\sqrt{2}}) * c_t$$

Bei einem vorgegebenen Wert von 1000 in der einheitenlosen Variablen „Torque Actual Value“, erreicht der Motor sein Nenndrehmoment.

Der Bemessungsstrom errechnet sich folgendermaßen:

$$I_N = \frac{M_N}{c_t}$$

$$I_N = \frac{0{,}38\ Nm}{0{,}08\ \frac{Nm}{A}} = 4{,}8\ A$$

Während der Projektarbeit war lange Zeit nicht klar, wie der Demonstrationsversuch seine Stellgröße generiert. Dies lag zum einen an der hier etwas unklaren Variablenbenennung anhand der Tatsache, dass im Programm viele vermeintliche Drehmomente definiert werden. Naheliegende Namen wie „Torque“ und „Moment“ lassen zwar auf Drehmomente schließen, jedoch handelt es sich hier, bei einigen um einheitenlose Stellgrößen. Tabelle 1 bietet eine Übersicht über diese Größen.

Tabelle 1: Übersicht aller als vermeintliches Drehmoment deklarierter Variablen

Bezeichnung im Programm	Verwendung	Einheit
My	Nein	[-]
Set_Moment1	Ja bei Sprung	[-]
Moment_M1_ist	Nein	[-]
Set_Torque1	Ja	[-]
Set_Moment1_geregelt	Ja	[-]

Mithilfe der Messdaten und einem einfachen MATLAB-Skript konnten die einzelnen Variablen in einander überführt werden.

Abbildung 3 zeigt die Umrechnung der einzelnen Größen. Die Umrechnung erfolgte über den Umrechnungsfaktor UF.

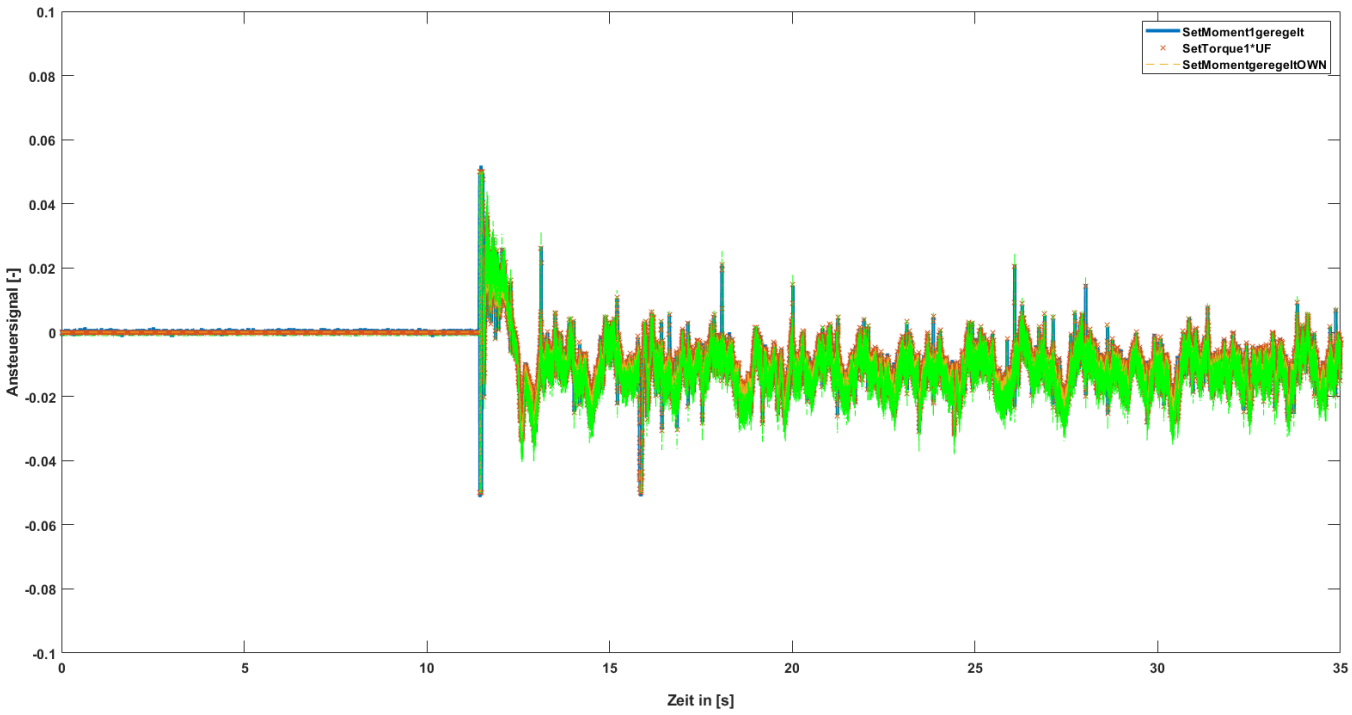

Abbildung 3: Umrechnung der einzelnen Stellwerte

$$UF = \frac{I_N * c_t}{1000 * \sqrt{2}}$$

$$UF = \frac{4{,}5 * 0{,}08}{1000 * \sqrt{2}}$$

Ebenfalls ist in Abbildung 3 zu sehen, dass das aus den Messsignalen u1x-u4x (siehe Kapitel 2) rekonstruierte Stellsignal „*SetMomentgeregeltOWN*“ sich ohne Abweichungen in die anderen Veriablen überführen lässt. Das verwendete MATLAB-Skript samt Testdatensatz befindet sich in Anhang B (Signale.m).

1.3 Laserabstandssensoren und Ermittlung der Punkte

Die drei Laserabstandssensoren von der Firma SICK Typ OD2-P135W75I0 ermöglichen eine Erfassung von der Distanz von den Sensoren zum Ball. Durch die drei ermittelten Distanzen lässt sich eine Kugel durch 3 Punkte bilden.

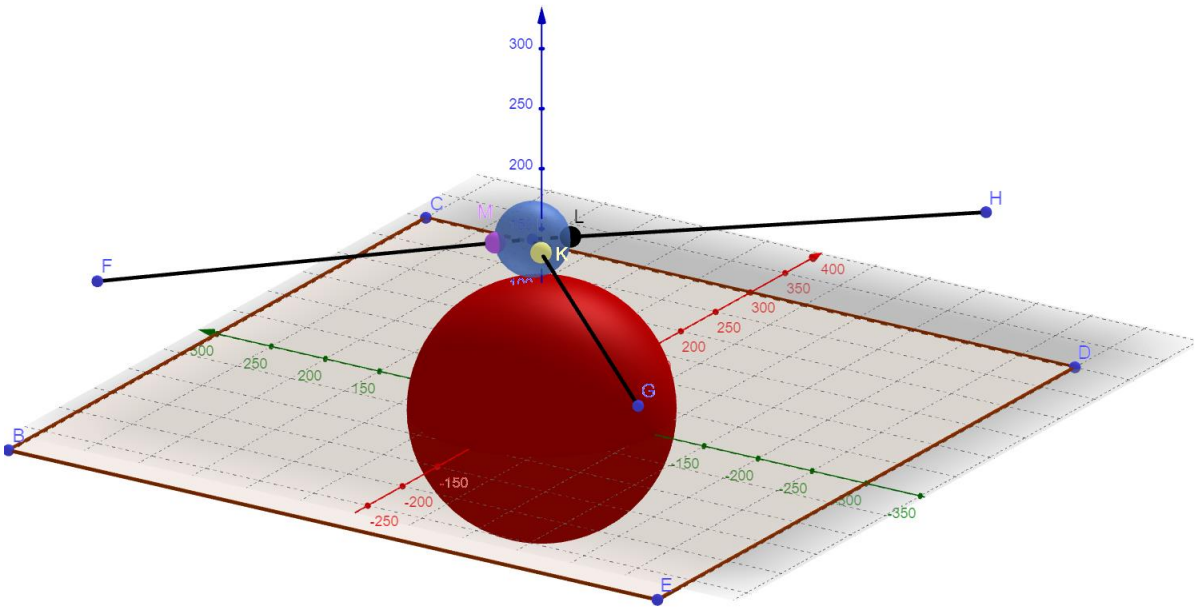

Abbildung 4: Kreis durch drei Punkte

Abbildung 4 zeigt die Ermittlung einer Kugel durch drei Punkte. Hierbei sind die Punkte F, G und H sinnbildlich für die drei Sensoren und die Punkte M, K und L die Punkte, auf die der Laserstrahl auf den Ball auftrifft.

Die Punkte M, K und L müssen aus den Längen der Strecken$\overrightarrow{FM}$, $\overrightarrow{GK}$ und $\overrightarrow{HL}$ und der Rotationsmatrix für den R^2 errechnet werden.

$M_x = \overrightarrow{FM} * \cos(-\frac{\pi}{4})$ $K_x = \overrightarrow{GK} * \cos(\frac{\pi}{4})$ $L_x = \overrightarrow{HL} * \cos(\frac{pi*115°}{180°})$

$M_y = \overrightarrow{FM} * -\sin(-\frac{\pi}{4})$ $K_y = \overrightarrow{GK} * -\sin(\frac{\pi}{4})$ $L_y = \overrightarrow{HL} * -\sin(\frac{pi*115°}{180°})$

Im Folgenden soll der Mittelpunkt der Kugel bzw. des Kreises so erklärt werden, wie es als Standardbeispiel vorkommt. Hier wird der Mittelpunkt über das Aufstellen eines linearen Gleichungssystems ermittelt.

$$I.: A + B * (-M_x) + C * (-M_y) = -(M_x^2 + M_y^2)$$
$$II.: A + B * (-K_x) + C * (-K_y) = -(K_x^2 + K_y^2)$$
$$III.: A + B * (-L_x) + C * (-L_y) = -(L_x^2 + L_y^2)$$

Die Koordinaten für den Mittelpunkt P ergeben sich hierbei folgendermaßen:

$$P_x = \frac{B}{2} \; und \; P_y = \frac{C}{2}$$

Aus dem so erhaltenen momentanen Kreismittelpunkt, welcher gleichzeitig für den Mittelpunkt des Balls steht, wird auf die zwei Regelgrößen $\theta_x$und θ_y geschlossen.

$$\theta_x = \sin^{-1}(\frac{P_x}{r_{Kugel} + r_{Ball}})$$

$$\theta_y = \sin^{-1}(\frac{P_y}{r_{Kugel} + r_{Ball}})$$

Dies ist die Berechnungsgrundlage, so wie sie im Demonstrationsversuch angewandt wird. Teilweise konnte diese Berechnungsgrundlage, wie sie im Programmcode existiert anfangs nicht komplett nachvollzogen werden, daher wurde eine GUI geschrieben um diese überprüfen zu können. Hierzu soll anhand eines Beispiels für P_x und P_y die kartesischen Werte 30 und 20 angenommen werden.

$$\theta_x: \sin^{-1}\left(\frac{30}{105 + 30}\right) = 12{,}83°$$

$$\theta_y: \sin^{-1}(\frac{20}{105 + 30}) = 8{,}52°$$

Die GUI „Position Visualizer“ basiert auf Berechnungen der Raumgeometrie. Die Position des Balls wird wie folgt ermittelt:

1. Schnittpunkt Gerade/Kugel
2. Erzeugen einer Ebene durch Schnittpunkt in 1. (Ebene ist parallel zur XZ bzw. YZ Ebene)
3. Schnittkreis Ebene/Kugel
4. Schnittpunkte Schnittkreis/Schnittkreis
5. Logischer Ausschluss eines der Ergebnisse

Die GUI ist samt Code für genauere Betrachtung in Anhang B zu finden. Das Ergebnis der GUI ist auf nachfolgender Seite in Abbildung 5 zu sehen.

Die GUI liefert für die errechneten Winkel des Demonstrationsversuchs andere Koordinaten für P_x und P_y. Es ist daher anzunehmen, dass die Positionsbestimmung des Winkels θ auf einem etwas vereinfachteren Modell beruht. Dies ist für den Betrieb des Demonstrationsversuchs irrelevant. Es sollte hierauf nur der Vollständigkeit halber hingewiesen werden um zu verhindern, dass die gemessenen Winkelpositionen nicht exakt den Winkelpositionen entsprechen, wie sie in der Simulation interpretiert werden. Es sei auch darauf hingewiesen, dass der Demonstrationsversuch nur verhältnismäßig kleine θ Winkel abbilden kann, da die Laserabstandssensoren auf einer fixen Höhe montiert sind und daher logischerweise nicht mit dem Ball die Kugel herabwandern können. Der Positionsfehler hält sich daher in Grenzen und wurde in diesem Beispiel eher übertrieben dargestellt.

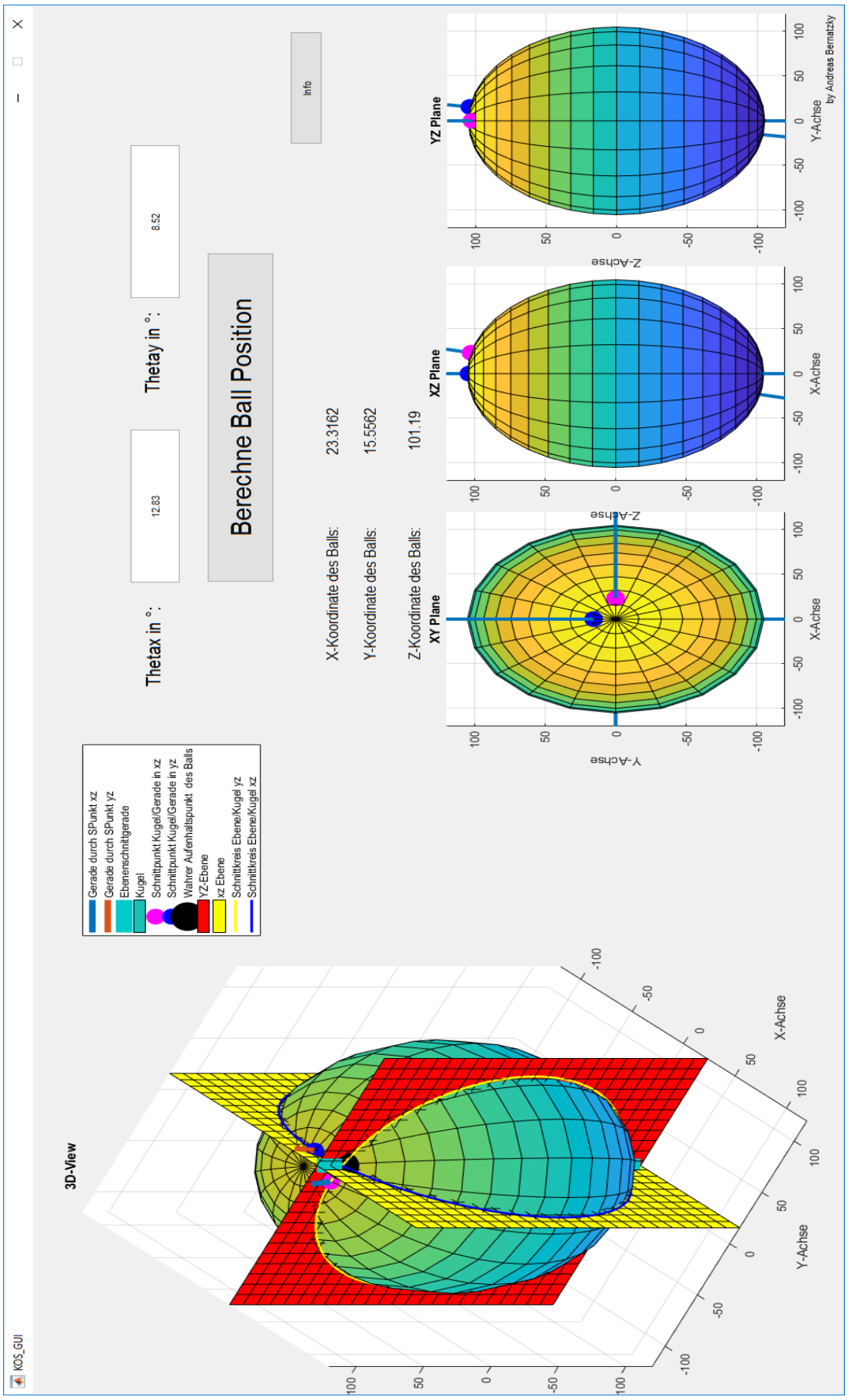

Abbildung 5: Position Visualizer GUI

1.4 Beckhoff CPU und Software

Der Regelungsalgorithmus und die Signalverarbeitung des Demonstrationsversuchs wird von einer Beckhoff CPU vom Typ CX5130-0130 / 4 GB ausgeführt. Ein Datenblatt zu dieser CPU findet sich in Anhang A. Da die Software des Demonstrationsversuchs nie dokumentiert wurde, soll der Programmablauf anhand eines UML-Diagramms dargestellt werden. Das Programm selbst gliedert sich hierbei in eine Initial-Function und in eine Main-Function. In der Initial-Function werden alle Variablen deklariert, die Initial-Function wird nur einmal durchlaufen. Die Main-Function hingegen wird immer wieder zyklisch durchlaufen. Beckhoff bietet die Möglichkeit ein Programm auf verschiedene Arten zu schreiben. Diese sind der Funktionsplan (FUP), die Anweisungsliste (AWL), der Kontaktplan (KOP) und der strukturierte Text (ST). Der Demonstrationsversuch wurde in der Form eines strukturierten Textes geschrieben. ST ist stark an die Programmiersprache PASCAL angelegt und bietet die Möglichkeit ein übersichtliches Programm zu schreiben, welches einfache gängige Funktionen wie z.B. IF-Else ausführen kann, ohne dass diese überkompliziert aus Logikbausteinen zusammengesetzt werden müssen. Sollten Änderungen am Programmcode vorgenommen werden, sollte diese auch in der momentanen Form erfolgen. Eine Konvertierung in eine der anderen genannten Programmierformen z.B. FUP könnte zu Kompatibilitätsproblemen führen. Das Hauptprogramm in der Main-Function wurde im Laufe der Projektarbeit mit Kommentaren ergänzt und soll auf nachfolgender Seite in einem UML-Diagramm dargestellt werden.

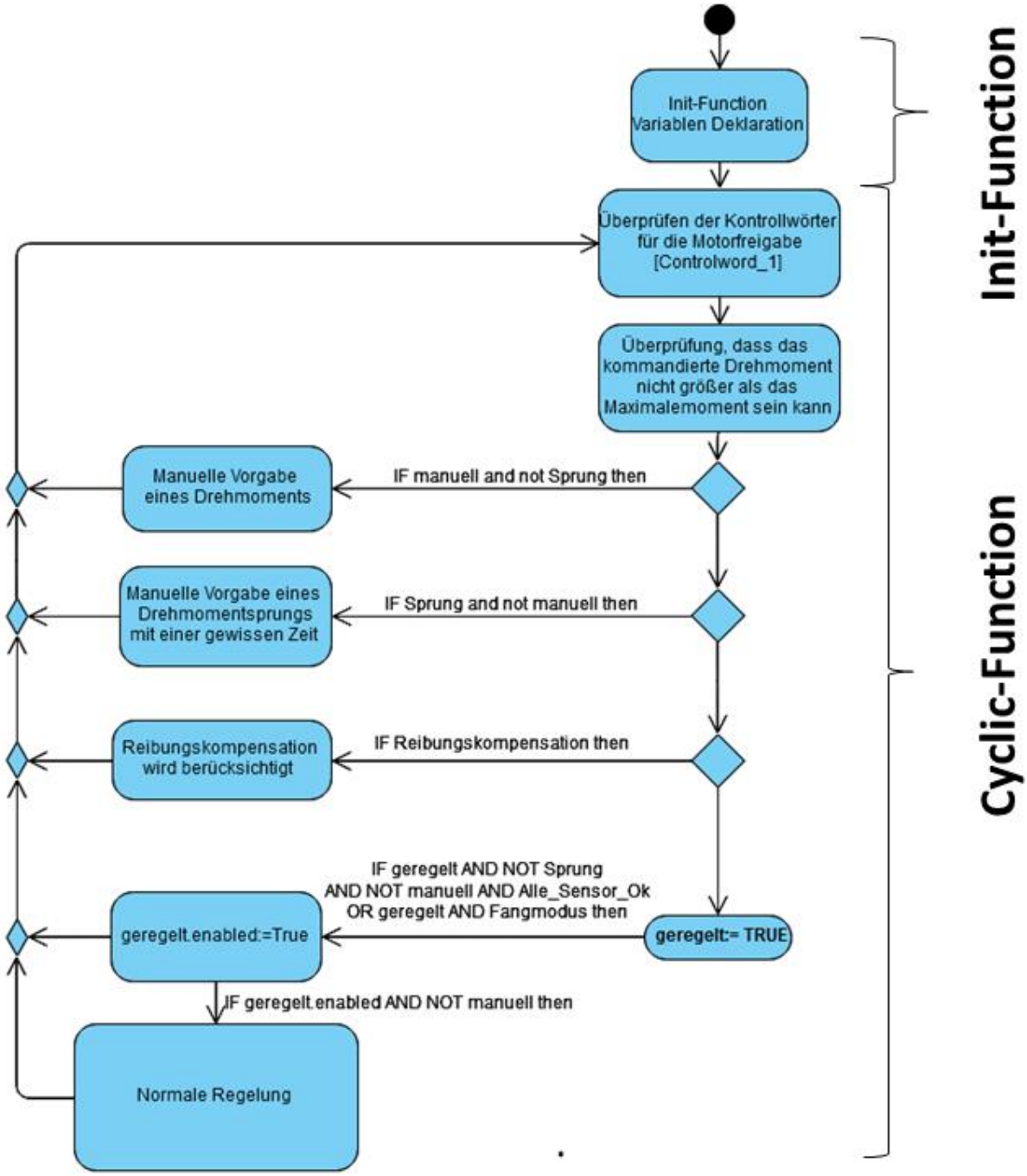

Abbildung 6: UML Diagramm des Demonstrationsversuches

2. Regler

Um ein instabiles System, wie den Ball auf Kugel Versuch stabilisieren zu können bedarf es einer Regelung. Das in dieser Projektarbeit erstellte Simulink Modell soll für weitere Reglerentwürfe herangezogen werden um einen einfacheren Regler zu implementieren, als den derzeit verwendeten Regler.

2.1 Derzeit verwendet Regler

Der Demonstrationsversuch wird über einen Zustandsregler geregelt. Es handelt sich hierbei um eine Mehrgrößenregelung. Für den Regler sind als Input folgende Größen von Relevanz:

Tabelle 2 Regelgrößen

Formelzeichen	Regelgröße	Einheit
θ	Positionswinkel des Balls	rad
θ'	Winkelgeschwindigkeit des Positionswinkels	$\frac{rad}{s}$
γ'	Kugeldrehgeschwindigkeit	$\frac{rad}{s}$

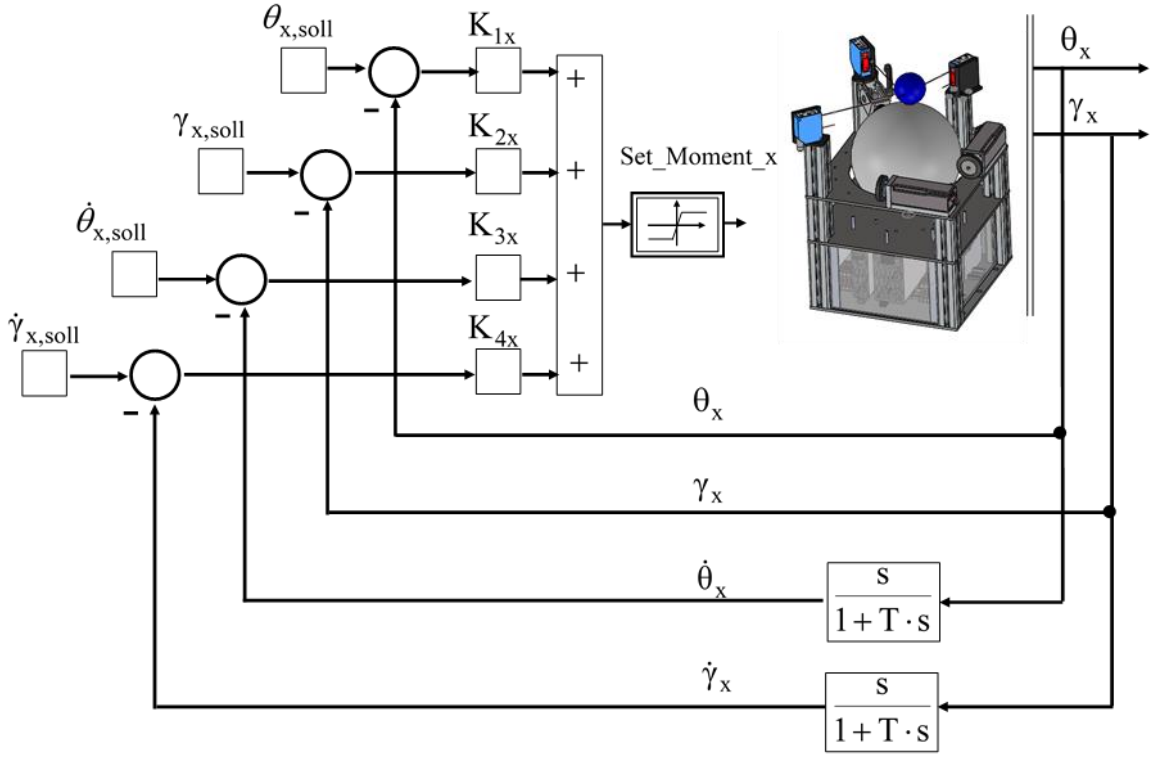

Abbildung 7: Derzeit verwendeter Zustandsregler (Huber/Zeitler, 2017)

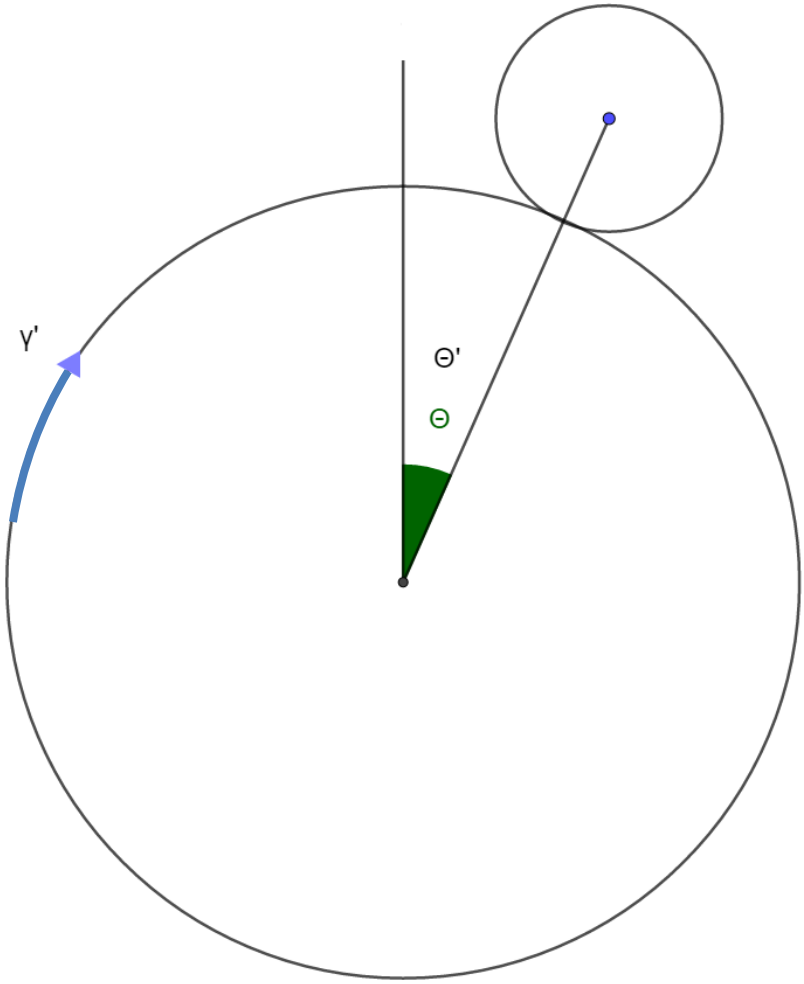

Abbildung 8 Regelgrößen

Der Stellwert für die Motoren wird folgendermaßen für jede Achse errechnet:

$$u1_x = (\theta_{soll} - \theta_{ist}) * K1$$
$$u2_x = (\gamma_{soll} - \gamma_{ist}) * K2$$
$$u3_x = (\theta'_{soll} - \theta'_{ist}) * K3$$
$$u4_x = (\gamma'_{soll} - \gamma'_{ist}) * K4$$

$$\mathrm{Set}_{\mathrm{Moment1}_{\mathrm{geregelt}}} = (\mathrm{u1_x} + \mathrm{u2_x} + \mathrm{u3_x} + \mathrm{u4_x})/3{,}5$$
$$\mathrm{SetTorque} = \frac{\mathrm{Set}_{\mathrm{Moment1}_{\mathrm{geregelt}}}}{UF}$$

Der Parameter $u2_x$ ist nur der Vollständigkeitshalber gelistet, da dieser mit $K2$=0 multipliziert wird.

Erwähnenswert hier ist noch, dass die Kugeldrehzahl γ' aus der Motordrehzahl und dem sich zwischen Kugel und Motorreibrad herrschendes Übersetzungsverhältnis errechnet. Sie wird daher nicht mit einer zusätzlichen Messeinrichtung direkt ermittelt. Die Erfassung der

Winkelposition θ des Balls erfolgt nach dem Schema, wie es in Kapitel 1.3 beschrieben wird. Die Winkeländerungsgeschwindigkeit θ' des Balls erfolgt in Beckhoff aus der Ableitung der Winkelposition. Die Ableitung wird in Kapitel 6.6 beschrieben.

2.2 PID-Regler

Da der derzeit verwendete Zustandsregler etwas unpraktisch ist, wurde versucht diesen durch einen einfachen PID-Regler zu ersetzen. Es wurden Testweiseversuche unternommen den Demonstrationsversuch nur nach der Winkelposition θ des Balls zu regeln. Diese Regelung reicht jedoch nicht dazu aus um den Demonstrationsversuch zu stabilisieren. Dennoch wurde in Beckhoff ein Programm mit dem Namen „Main_PID“ angelegt um dieses für spätere Regler eventuell aufgreifen zu können.

3. Rauschcharakterisierung Sensorik

Um eine realitätsgetreue Simulation nachbilden zu können ist es wichtig, das Messrauschen charakterisieren zu können und nachzubilden. Für den Demonstrationsversuch gilt es, das Signalrauschen der Aktorik und der Sensorik zu charakterisieren und nachzustellen. Für die Charakterisierung des Sensorrauschens wurde der Ball in einem sich auf der Kugeloberfläche befindenden kleinen Loch fixiert. Das so aufgenommene Positionssignal des Balls ist unverändert und es kann das reine Messrauschen betrachtet werden. Abbildung 9 zeigt das Positionsrauschens für eine fixierte Winkelposition des Balls.

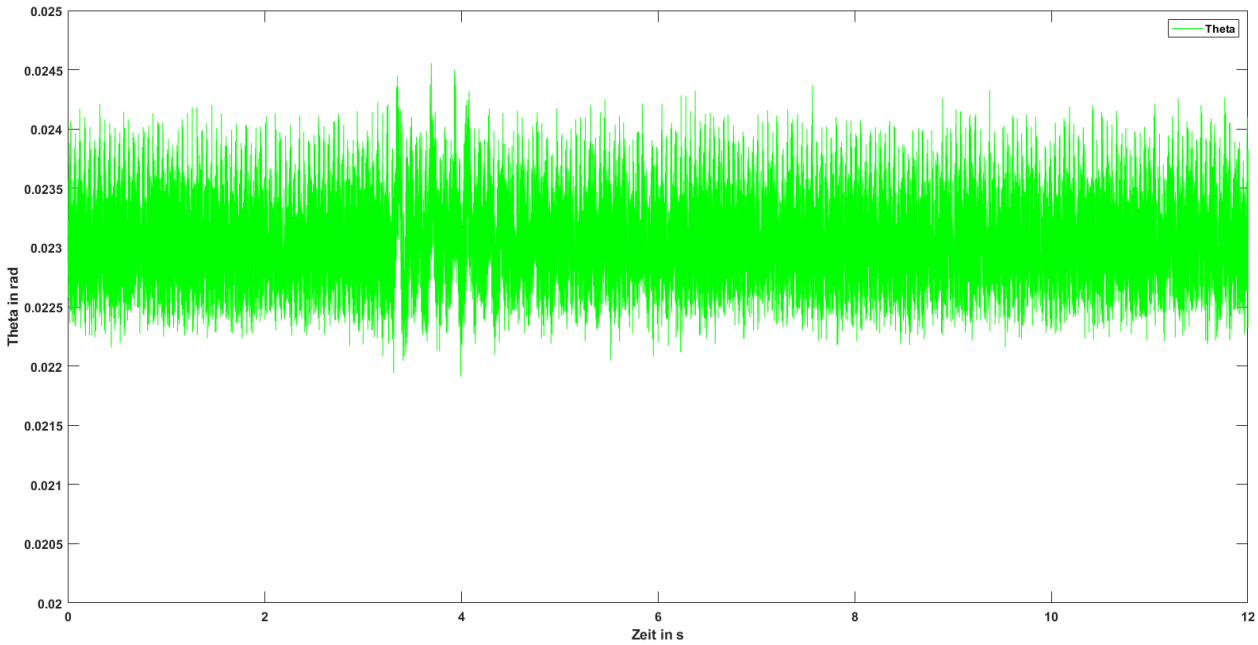

Abbildung 9: Sensorrauschen für einen Ball in fixierter Position

Um das tatsächliche Positionssignal zu ermitteln wurde das verrauschte Signal mit einem FIR-Filter[1] gefiltert (Hermann). Hierzu wurde in MATLAB der gleitende Mittelwert über jeweils 400 Messpunkte gebildet.

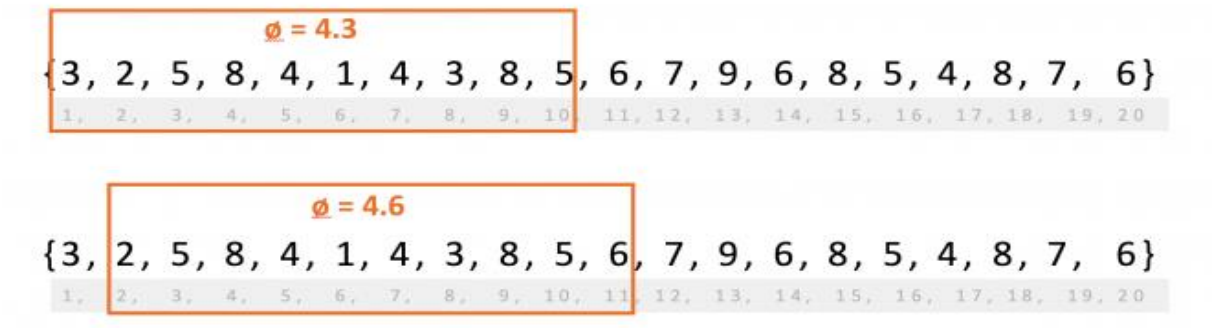

Abbildung 10: Prinzip des gleitenden Mittelwertes über 10 Messpunkte (Hofmann, 2017)

[1] Finite Impulse Response

Da der Ball während dieses Tests fixiert war wurde nochmals der Mittelwert über das gefilterte Signal gebildet, was als wahrer Wert der Ballposition interpretiert werden kann.

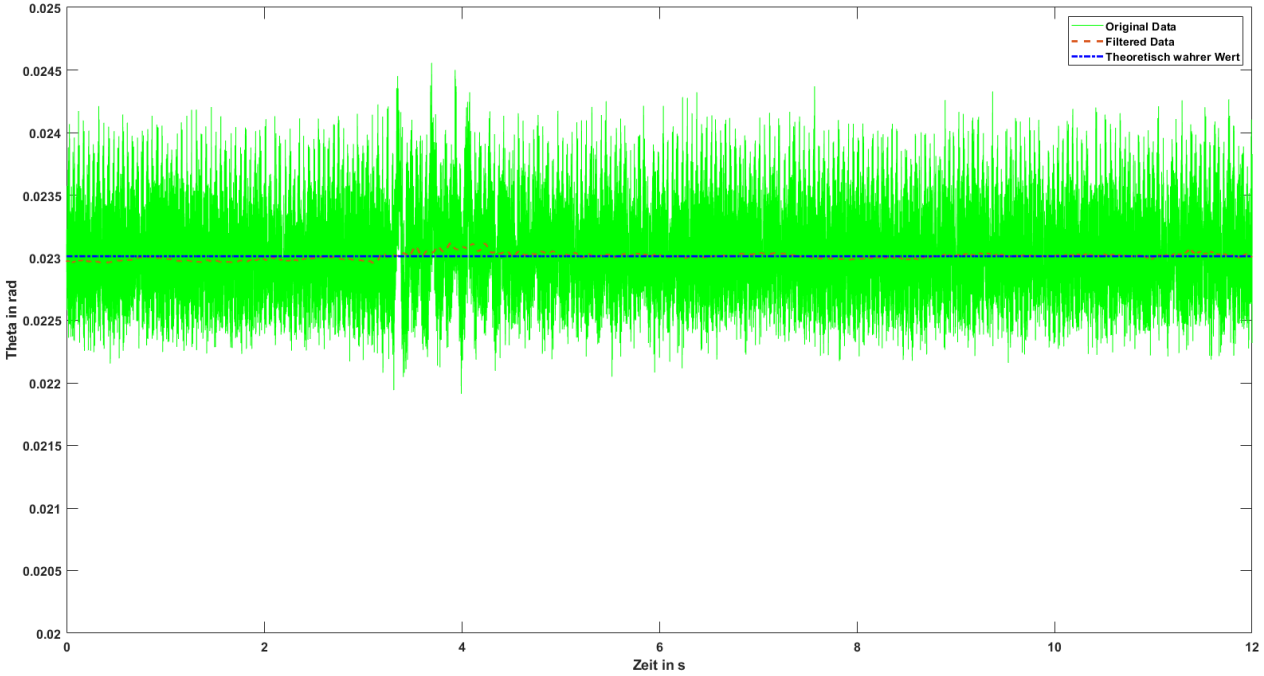

Abbildung 11: Der theoretische wahre Wert der Ballposition kann für die Bestimmung des korrekten Signal-Noise-Ratio herangezogen werden

Mit der MATLAB-Funktion `awgn()` wurde ein Gaußsches Rauschen erzeugt und über den Theoretisch wahren Wert gelegt. Es konnte hiermit ein SNR von 67 bestimmt werden.

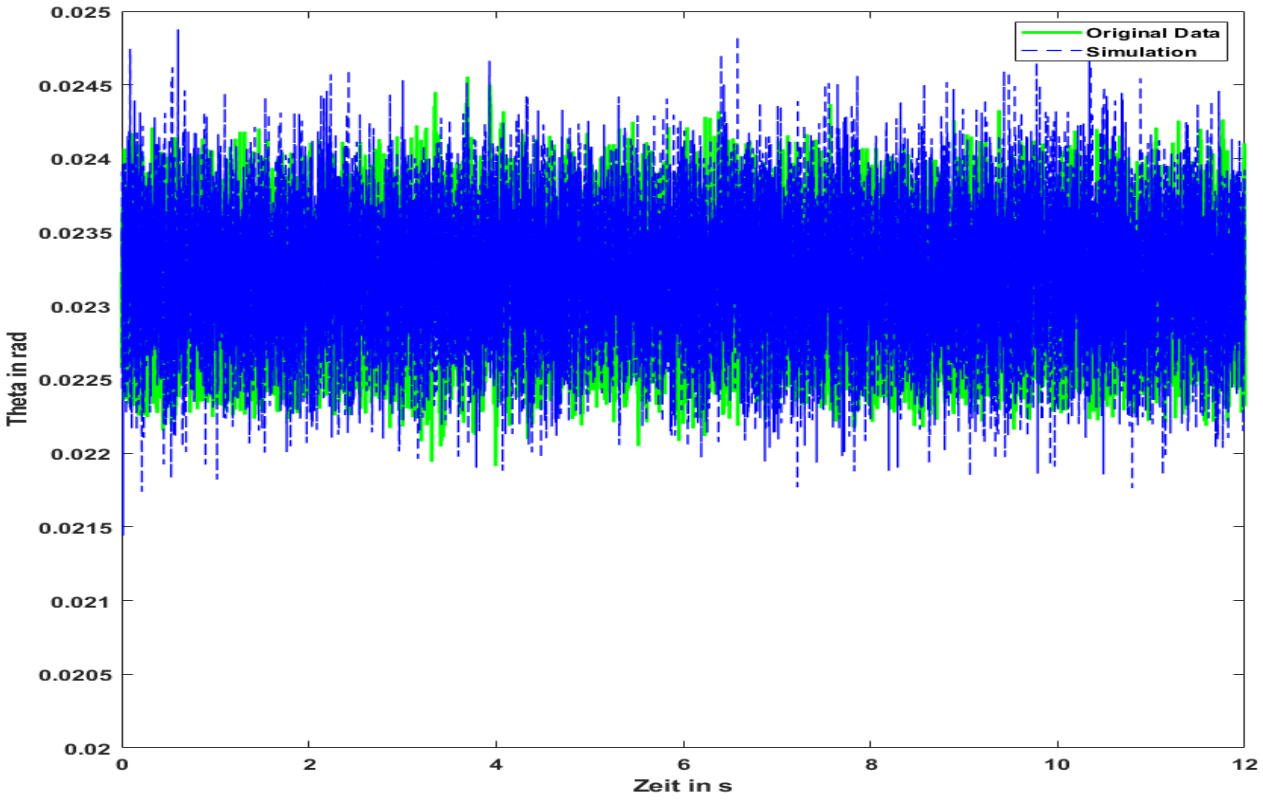

Abbildung 12: Vergleich Simuliertes Signal mit dem echten Signal

4. Rauschcharakterisierung Aktorik

Um das Messrauschen des Drehmoments charakterisieren zu können, wurden am Demonstrationsversuch Drehmomentsprünge von 0,03 Nm bis 0,05 Nm vorgegeben und diese analysiert. Abbildung 14 zeigt einen Drehmomentsprung von 0,035 Nm

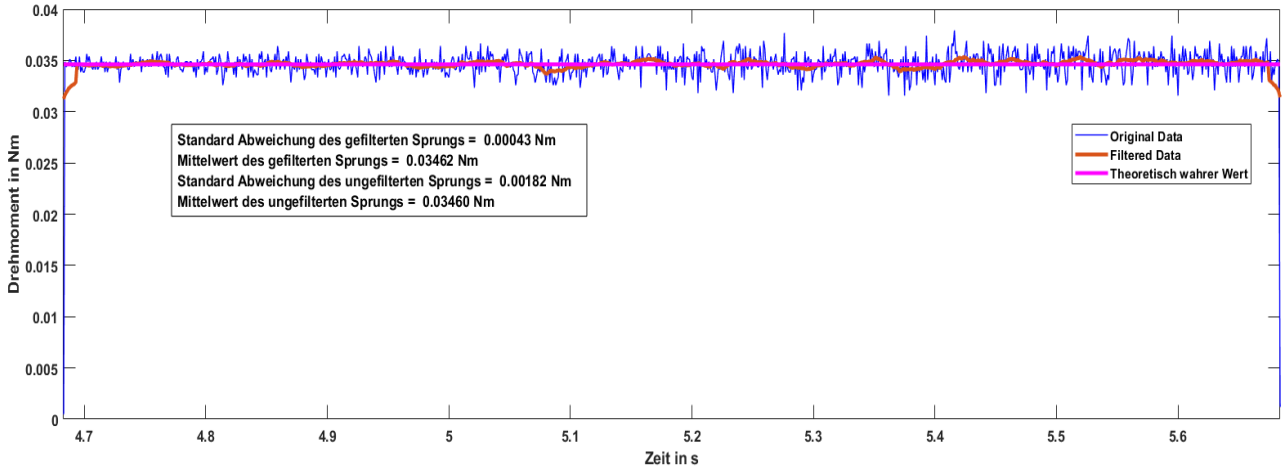

Abbildung 13: Drehmomentrauschen 0,035 Nm mit gefilterten Signal

Dieser Drehmomentsprung wurde anschließend ähnlich wie in Kapitel 3 gefiltert. Das Rauschen des Drehmoments wurde über die Standardabweichung charakterisiert. Das Ergebnis hierzu ist in Tabelle 3 zu sehen.

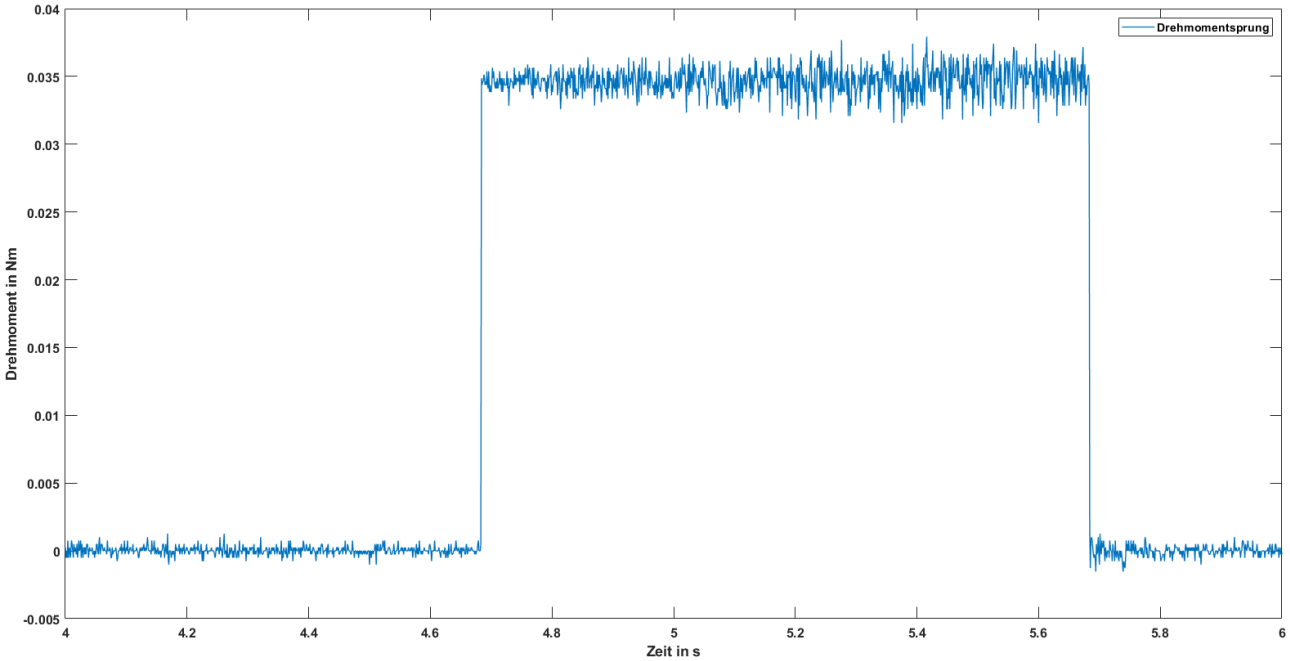

Abbildung 14: Drehmomentsprung für 0,035 Nm

Tabelle 3: Drehmomentsprünge mit den dazugehörigen Standardabweichungen

Drehmomentsprung [Nm]	Standardabweichung [Nm]
0,03	0,00154
0,035	0,00182
0,04	0.00216
0,045	0.00236
0,05	0.00273

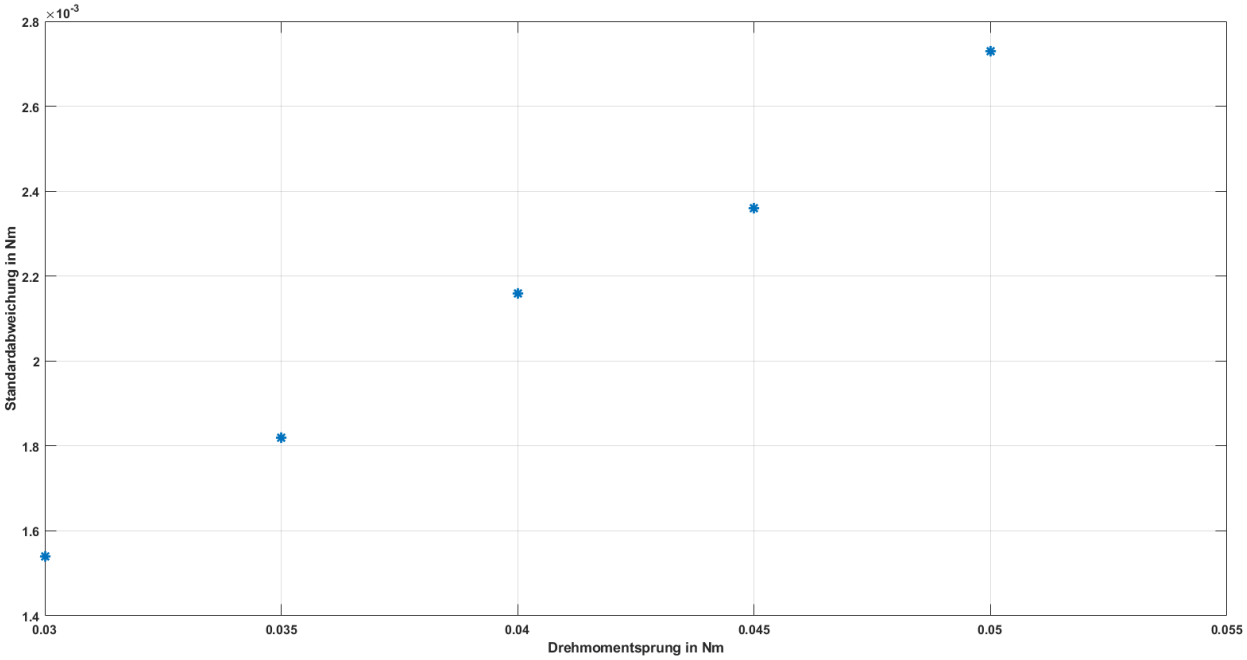

Abbildung 15: Das Ergebnis von Tabelle 3 lässt einen linearen Zusammenhang vermuten

Aus diesem Zusammenhang konnte ein SNR zwischen 62 (für 0,03 Nm) und 55 (für 0,05 Nm) ermittelt werden. Es wurden auch negative Drehmomente untersucht, welche auf ähnliche Signal to Noise Ratios schließen ließen. Der Bereich von 0 bis 0.03 Nm konnte nicht untersucht werden, da hier das Reibungsdrehmoment zu groß ist um ein Anlaufen der Motoren zu ermöglichen. Es wurde daher für diesen Bereich interpoliert.

Die Signal to Noise Ratio folgt somit einem einfach Polynom erster Ordnung:

$$SNR_{positiv} = -350 * Drehmoment + 72{,}5$$

Bzw. für negative Drehmomente:

$$SNR_{negativ} = 350 * Drehmoment + 72{,}5$$

5. Experimentelle Identifikation

Die experimentelle Systemidentifikation stellt eine schnelle und einfache Möglichkeit dar ein lineares Modell zu erzeugen. Hierzu wird das zeitliche Ein- und Ausgangsverhalten eines dynamischen Systems untersucht und mithilfe eines Parameterschätzverfahrens, wie z.B. Maximum-Likelihood oder der Methode, der kleinsten Quadrate, die Parameter für eine gegebene Systemstruktur ermittelt. Es wird hierbei immer davon ausgegangen, dass die anzunehmende mathematische Struktur des Modells, das reale System tatsächlich gut wiederspiegelt. Die experimentelle Identifikation wurde mit pzMove (version 81) durchgeführt.

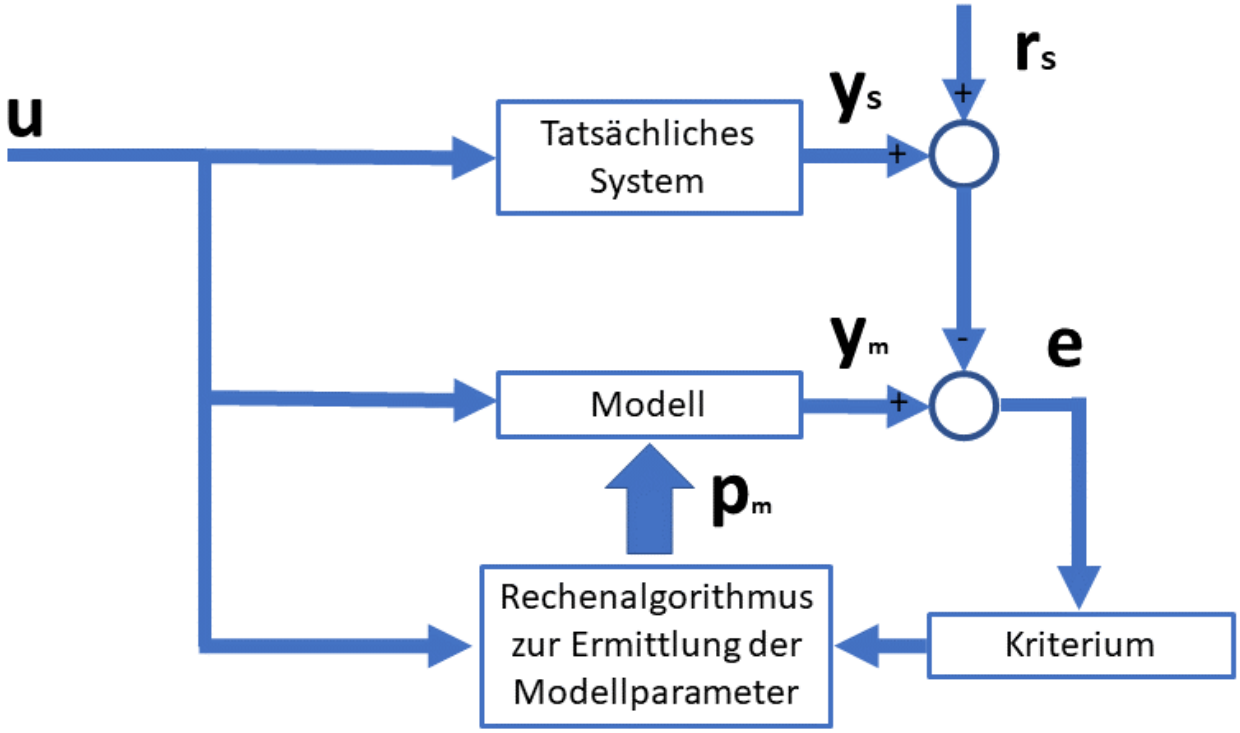

Abbildung 16: Schematischer Ablauf von Parameterschätzungen (Bohn, 2016)

5.1 Experimentelle Identifikation der Ballposition

Zu Beginn der Projektarbeit wurden viele Versuche unternommen, den Demonstrationsversuch mit Hilfe der experimentellen Identifikation, am offenen Kreis zu identifizieren. Hierzu wurde der Ball wie in Kapitel 3 beschrieben fixiert und ein Drehmomentsprung auf das System gegeben. Der Ball wurde hierbei schlagartig von der Kugel geschleudert.

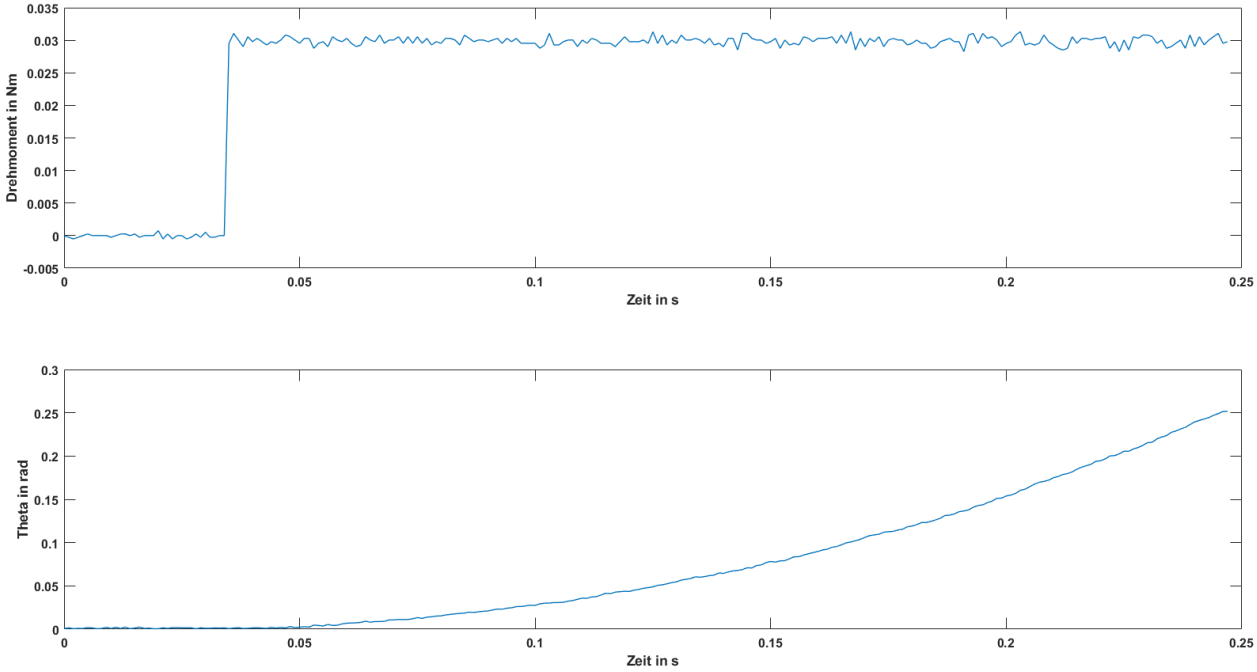

Abbildung 17: Drehmomentsprung mit Sprungantwort, wie er in pzMove verwendet wurde

Als zu identifizierendes System wurde ein Doppelintegrator angenommen. Im geschlossenen Kreis in Simulink lieferte das System jedoch ein nicht zufrieden stellendes Ergebnis. Als problematisch wird hier erachtet, dass die Sprungantwort an einem instabilen System durchgeführt wird und mit dem Verhalten im Normalbetrieb wenig bis gar nichts gemeinsam hat. Die Vermutung liegt daher auch nahe, das ein Sprung als Testsignal ungeeignet ist. Besser geeignet wäre vermutlich ein Sinussignal auf beide Motoren zu geben, welches den Ball in einer Kreisbahn hin und her oszillieren lässt. Hierzu wurden Versuche unternommen, jedoch stellt es sich in der Praxis als sehr umständlich heraus einen Sinus zu finden, welcher das System so anregt, dass dieses sich wie ein grenzstabiles System verhält. Eine experimentelle Systemidenfikation der Ballposition, auf der Kugel konnte daher nicht stattfinden.

5.2 Experimentelle Identifikation der Kugel

Die Drehzahl der Kugel konnte mithilfe von pzMove und einer aufgenommenen Sprungantwort gemessen werden. Anfangs wurde hierbei, mit den Sprungantworten aus Abbildung 18 und durch Ableiten versucht auf die Kugeldrehzahl zu schließen.

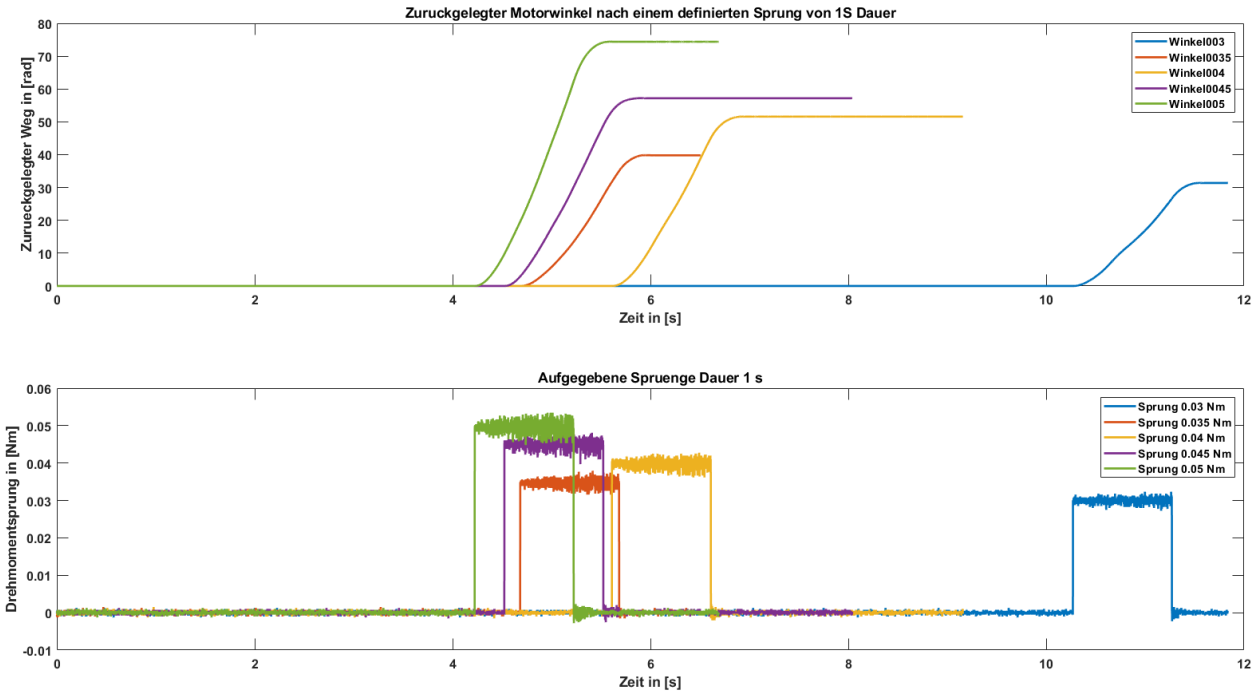

Abbildung 18: Verwendete Drehmomentsprünge und Sprungantworten

Da der zurückgelegte Motorwinkel eine Integration der Kugeldrehzahl darstellt und es bei Integrationen naturgemäß zum „Informationsverlust“ kommt, wurde auch hier die mechanische Unwucht der Kugel und Motoren zuerst nicht bemerkt. Die Unwucht ist sehr wahrscheinlich darauf zurück zu führen, dass die Motoren nicht perfekt verbaut sind, oder dass die Kugel nicht ideal rund ist.

Eine genauere Betrachtung zeigt, dass die Kugel für verschiedene Drehmomente, wie zu erwarten nicht nur in ihrem Endwert der Drehzahlvariabel ist, sondern dass diese auch mit einer Schwingung überlagert ist, welche in ihrer Frequenz und Amplitude variabel ist. Nachfolgende Abbildung 19 bis Abbildung 21 zeigen die Kugeldrehzahl, welche durch eine mechanische Schwingung überlagert ist.

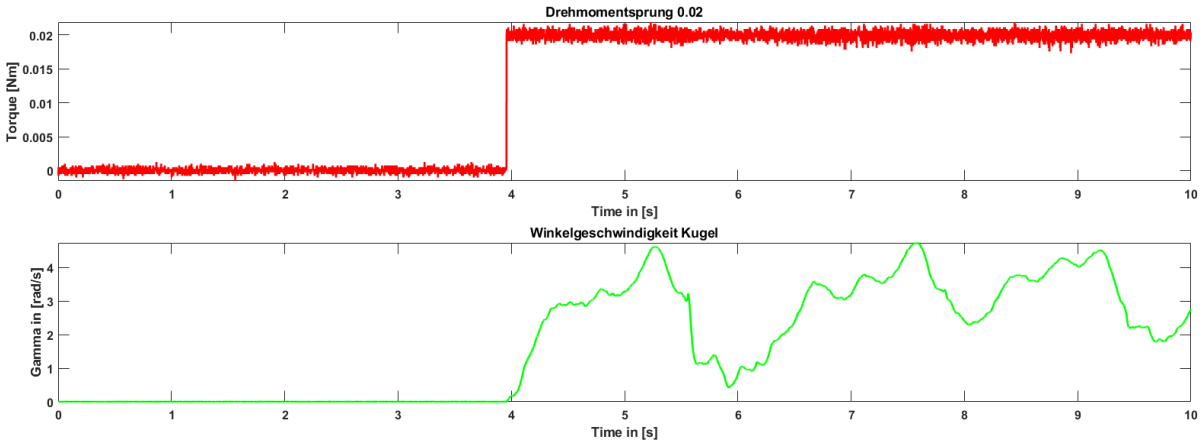

Abbildung 19: Schwingfrequenz ≈ 0,6 Hz

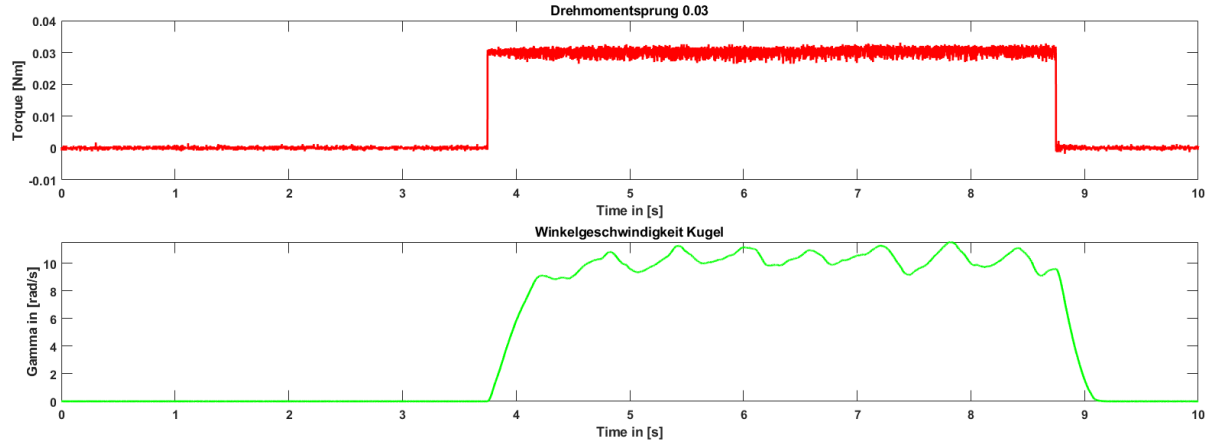

Abbildung 20: Schwingfrequenz ≈ 1,6 Hz

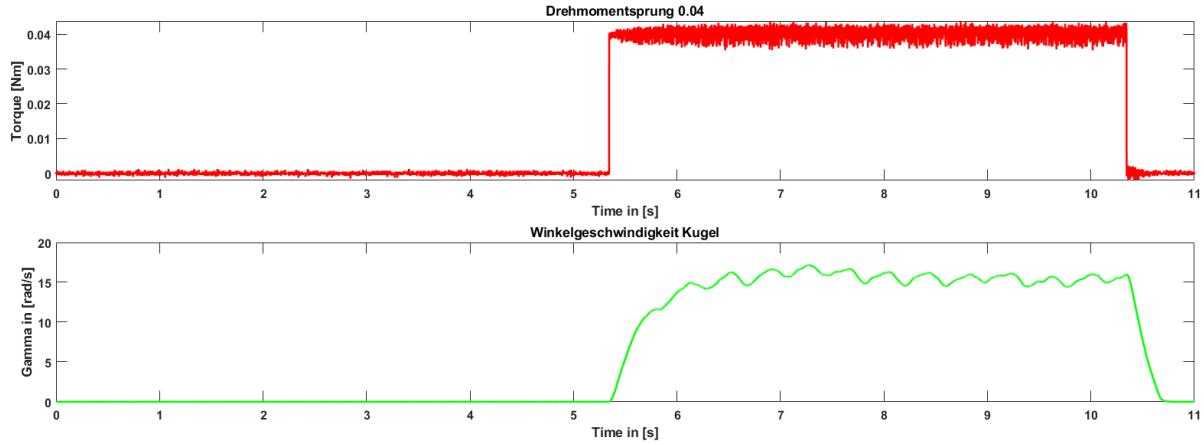

Abbildung 21: Schwingfrequenz ≈ 2,6 Hz

5.4 Fazit zur experimentellen Identifikation

Für das Simulationsmodell zum Demonstrationsversuch konnte mithilfe des Ein- und Ausgangsverhalten, des Versuchs keine brauchbare Systemidentifikation vorgenommen werden. Bei der Identifikation, der Ballposition θ wurden verschiedenste Sprungantworten durchgetestet, eine Identifikation mit Messdaten aus dem geschlossenen Kreis blieb ebenfalls erfolglos. Die experimentelle Identifikation, stellt bei Systemen, welche auf ein beschränktes Eingangssignal mit einer beschränkten Systemantwort reagieren, gewiss eine bequeme und schnelle Möglichkeit dar ein Modell zu bilden. Im Fall der Kugeldrehzahl γ' konnte für den Endwert der Kugeldrehzahl eine lineare Übertragungsfunktion gebildet werden. Jedoch stellte sich hier als problematisch heraus, dass das in Abbildung 19 bis Abbildung 21 zu sehende nichtlineare Verhalten nicht mit abgebildet werden kann (zumindest nicht mit einer Übertragungsfunktion allein). Ein weiterer Nachteil für die experimentelle Modellbildung im Fall des Demonstrationsversuchs ist, dass das System nur mit Testsignalen angeregt werden konnte, welche von ihrer Größer her nicht dem Betriebsfall wiederspiegeln (zum Überwinden der Haftreibung musste das Testsignal entsprechend hoch gewählt werden). Das schlechte Verhalten der experimentellen Kugeldrehzahl, welches in Abbildung 22 zu sehen ist, wird daher auf die ungünstig hohen Testsignale zurückgeführt. Daher wurde für das abschließende Modell, die experimentelle gewonnene Übertragungsfunktion für die Kugeldrehzahl nicht verwendet. Abbildung 22 lässt vermuten, dass die experimentell gewonnene Übertragungsfunktion nur eine etwas zu hohe Verstärkung besitzt, dem ist auch so und die Verstärkung wurde nachträglich nachjustiert. Die experimentell gewonnene Übertragungsfunktion ist dennoch nachfolgend zu gelistet:

$$G(s)_{Kugel} = \frac{2460{,}4 * s + 5074{,}5}{s^2 + 8{,}5355 * s + 14{,}851}$$

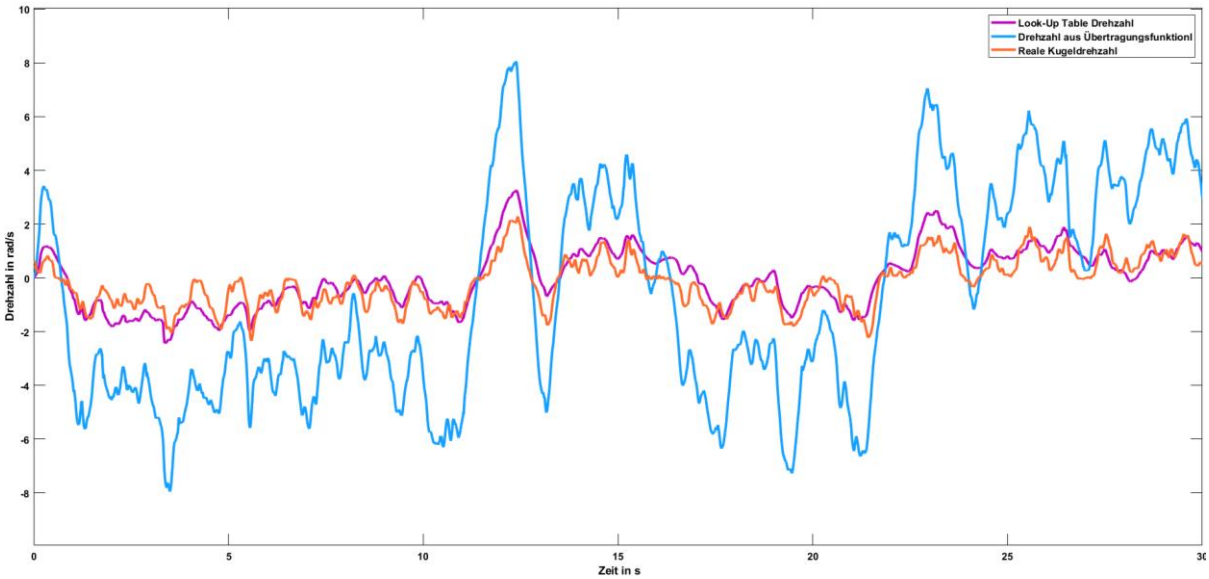

Abbildung 22 Vergleich der experimentell gewonnen Kugeldrehzahl, mit der Look-Up Table Methode

Die Modellierung mithilfe von Look-Up Tables wird in Kapitel 6.2 behandelt.

6. Analytische Simulation des Demonstrationsversuchs

Wie in Kapitel 5 gezeigt, liefert der Ansatz über die experimentelle Identifikation für die Simulation keine idealen Ergebnisse. Daher wurde parallel zu den Versuchen der experimentellen Systemidentifikation die Modellierung des Demonstrationsversuches, als White-Box Modell vorgenommen. Bei der experimentellen Identifikation liegt die Systemgleichung in Form einer Übertragungsfunktion vor. Diese Übertragungsfunktion spiegelt exakt nur dieses System wider, wie es in der Realität auch existiert. Um Vorhersagen mit veränderlichen Parametern[2] treffen zu können sind solche Systeme nicht in der Lage. White-Box Modelle hingegen bilden das System mit all ihren physikalischen Gegebenheiten ab und können logisch anhand dieser nachvollzogen werden. Problematisch bei White-Box Modellen ist hingegen, dass manche physikalische Zusammenhänge nur unter hohem Aufwand analytisch simuliert werden können, zum Beispiel das fließen von Kunststoffen unter Last.

Für das in Simulink erstellte Modell wurden verschiedene physikalische Ansätze erprobt und getestet. In den nächsten Abschnitten werden daher auf die einzelnen Simulink Systeme eingegangen und deren Wirken anhand von physikalischen Gleichungen erläutert.

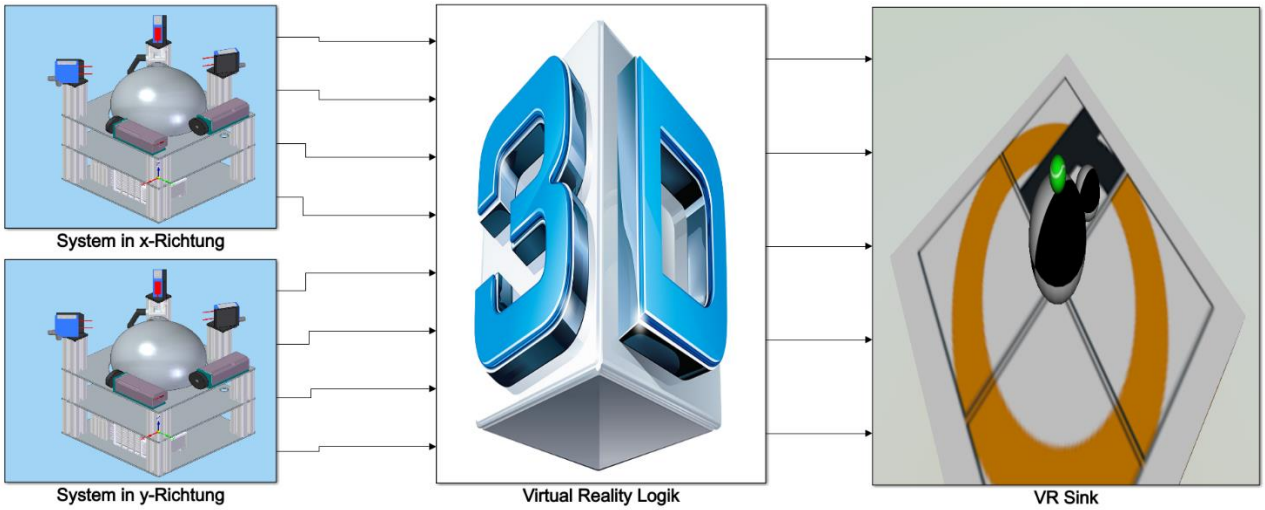

Abbildung 23: Topansicht des Modells. Das Modell wurde voneinander unabhängig für beide Richtungen simuliert.

[2] Zum Beispiel ein schwerer Ball

6.1 Simulink Modell des Reglers

Der Zustandsregler wurde komplett dem Programmcode in Beckhoff nachempfunden. Die Regelgrößen sind die Größen, welche in Kapitel 2 erläutert wurden. Diese sind nochmal die Winkelposition des Balls θ, die Winkeländerungsgeschwindigkeit des Balls θ' und die Kugeldrehzahl γ'. Abbildung 24 zeigt den implementierten Regler als Simulink Modell. Der hinterlegte Saturation-Block riegelt, sowohl die Simulation als auch den Prüfstand auf ein maximales Ansteuersignal von 0,05 bis -0,05 ab.

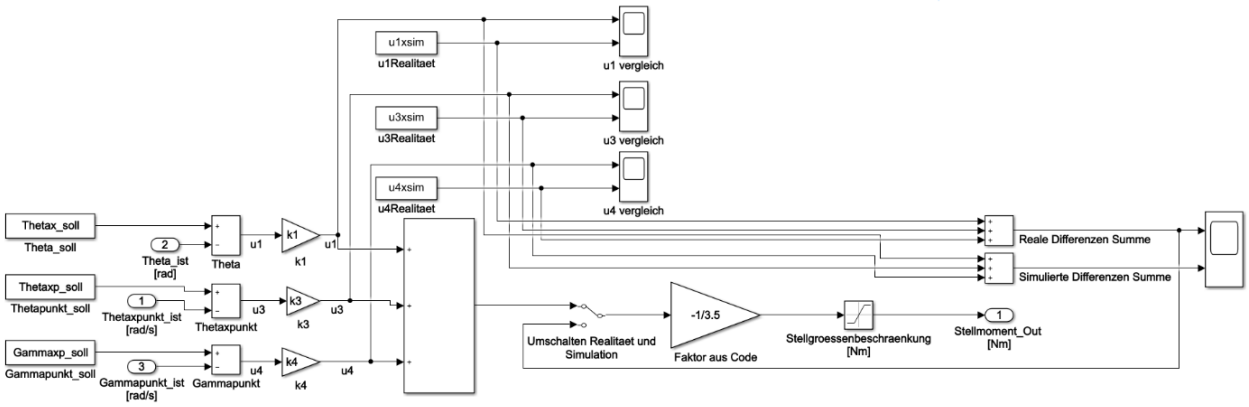

Abbildung 24: Simulink Modell des Reglers

6.2 Simulink Modell der Kugel

Die Kugeldrehzahl wurde mithilfe von Lookup-Tabellen modelliert. Hierfür wurde sich entschieden, da die Kugeldrehzahl von mechanischen Schwingungen überlagert ist, wie sie zum Beispiel in Abbildung 19 zu sehen sind. Diese mechanische Schwingung stellte sich wie in Kapitel 5.2 schon zuvor beschrieben als nichtlinear in ihrer Frequenz und in ihrer Amplitude heraus. Daher wurden verschiedene Ansteuersignale in 0,05er Schritten erfasst und bezüglich ihres Endwertes in der Drehzahl, ihrer Schwingungsamplitude und ihrer Schwingungsfrequenz erfasst und anschließend in MATLAB ausgewertet. Es wurden hierbei sowohl positive, als auch negative Drehmomente erfasst. Die beiden Achsen reagieren, wie zu erwarten identisch auf Drehmomentanregungen. Diese drei jeweils vom Drehmoment abhängigen Größen wurden separat in Lookup-Tabellen nachgebildet. Die Größen der Schwingungsfrequenz und der Schwingungsamplitude werden zusammengehörend innerhalb einer MATLAB-Function mithilfe eines einfachen Sinus verrechnet. Dieser wird wiederum auf den Endwert der Kugeldrehzahl addiert. Diese Art der Modellierung war vor Beginn der Projektarbeit nicht wirklich bekannt. Sie stellte aber für diesen Fall, eine überraschend effektive Methodik dar, wie Abbildung 26 zeigt.

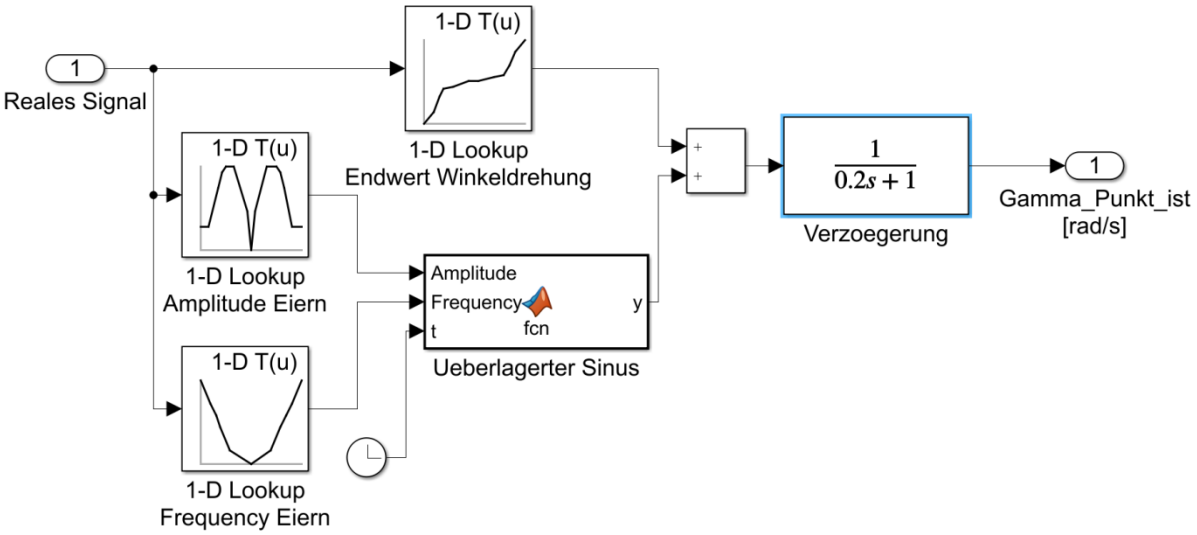

Abbildung 25: Simulink Modell der Kugeldrehzahl

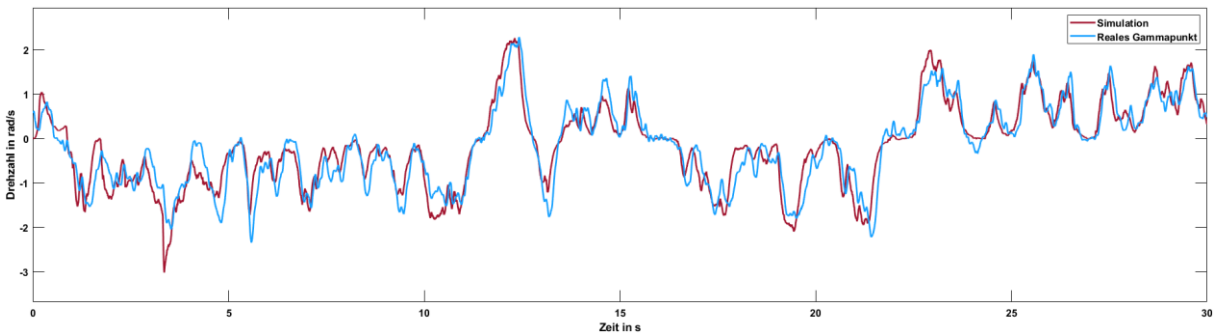

Abbildung 26 Vergleich zwischen Realität und Simulation

6.3 Simulink Modell des Balls

Das Simulink Modell des Balls beinhaltet den instabilen Teil des Modells. Hier wird das Herabrollen des Balls simuliert und die mechanische Kopplung zwischen Ball und Kugel ist hier hinterlegt. Die Entwicklung des Simulink Modell des Balls durchlief verschiedene Entwicklungsschritte mit unterschiedlichen physikalischen Ansätzen. Anfangs wurde hier versucht die Kopplung darüber zu erreichen, dass das durch die Hangabtriebskraft F_H erzeugte Drehmoment, durch ein vom Motor kommendes Drehmoment kompensiert wird. Dies liefert augenscheinlich eine korrekte Simulation für den Positionsverlauf θ. Der Winkel Theta erfolgt hierbei aus einer Multiplikation des zurückgelegten Drehwinkels des Balls multipliziert mit dem Übersetzungsverhältnis zwischen Ball und Kugel. Bei genauer Betrachtung des Demonstrationsversuchs fällt jedoch auf, dass der Ball kontinuierlich auf der großen Kugel abrollt, dies war bei dem Ansatz der Drehmomente nicht gegeben. Hier hätte die Kugel zwar ein plausibles θ dargestellt, jedoch ohne, dass Kugel und Ball sich kontinuierlich drehen müssen, wie in Abbildung 28 zu sehen ist. Demzufolge war dies auch die Ursache für die unzureichende Drehgeschwindigkeit der großen Kugel, bei den ersten Modellierungen. Daher werden keine Drehmomente versucht zu kompensieren, sondern die

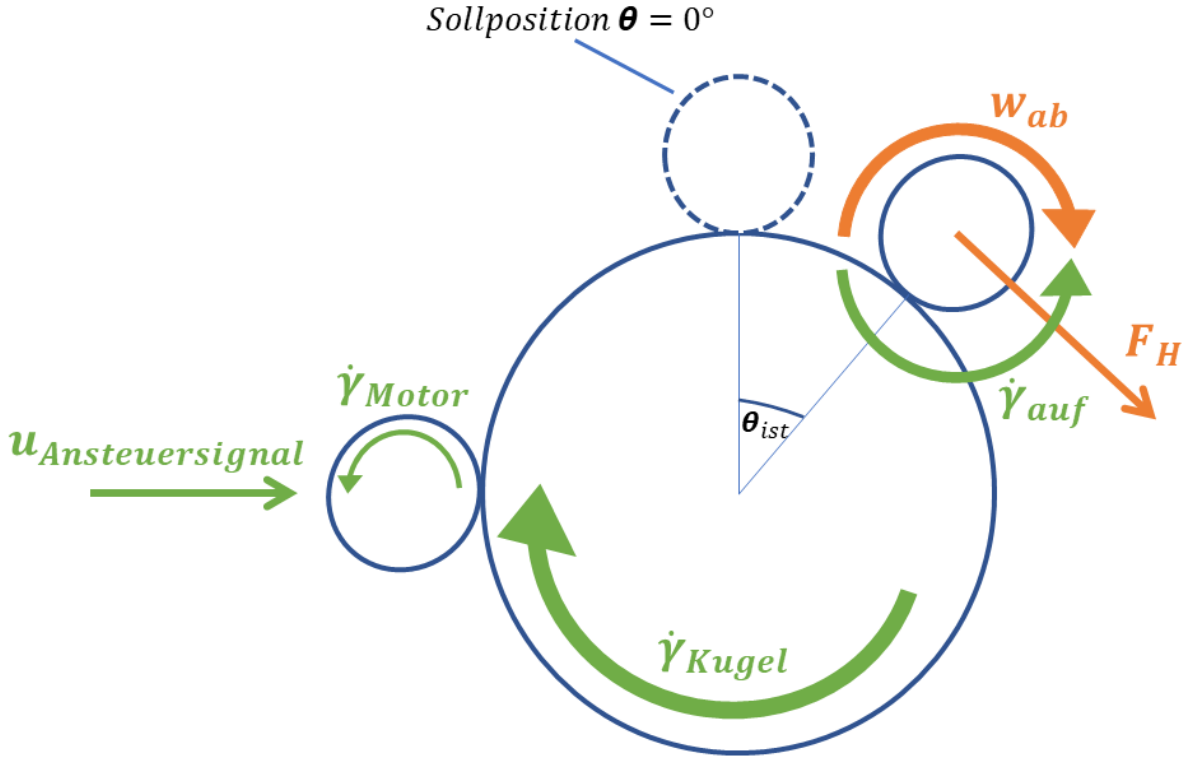

Abbildung 27: Physikalischer Ansatz der Simulation

Kugeldrehzahl wird auf den Ball übersetzt ($\dot{\gamma}_{auf}$) und kompensiert die Drehzahl des Balls (ω_{ab}), welches diesen versucht her abzuzwingen. Abbildung 27 veranschaulicht das Vorgehen.

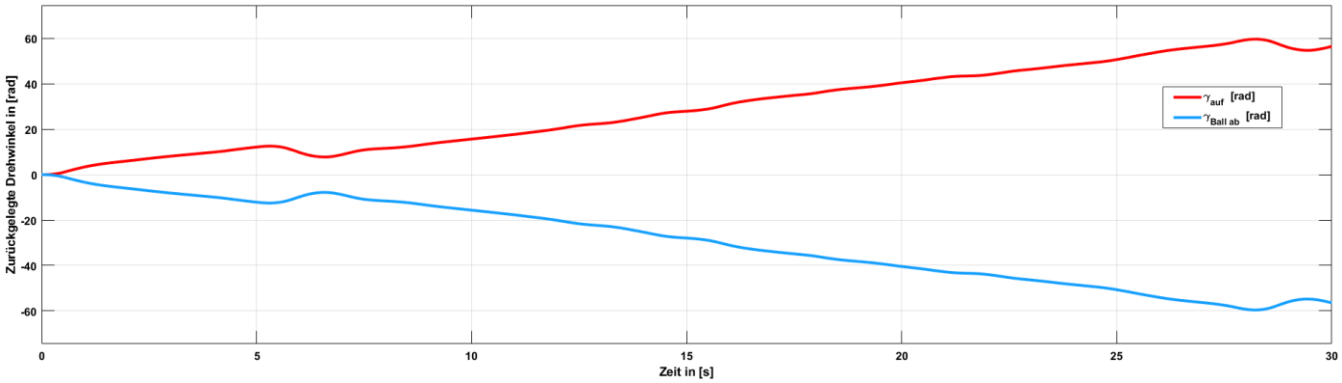

Abbildung 28: Die beiden Drehwinkel kompensieren sich für γ_{auf}und γ_{ab}

Nachfolgende Formel stellt den mathematischen Zusammenhang zwischen den beiden sich kompensierenden Drehwinkeln dar.

$$\theta = \frac{r_B}{r_K} * (\omega_{\text{ab}} + \gamma_{\text{auf}})$$

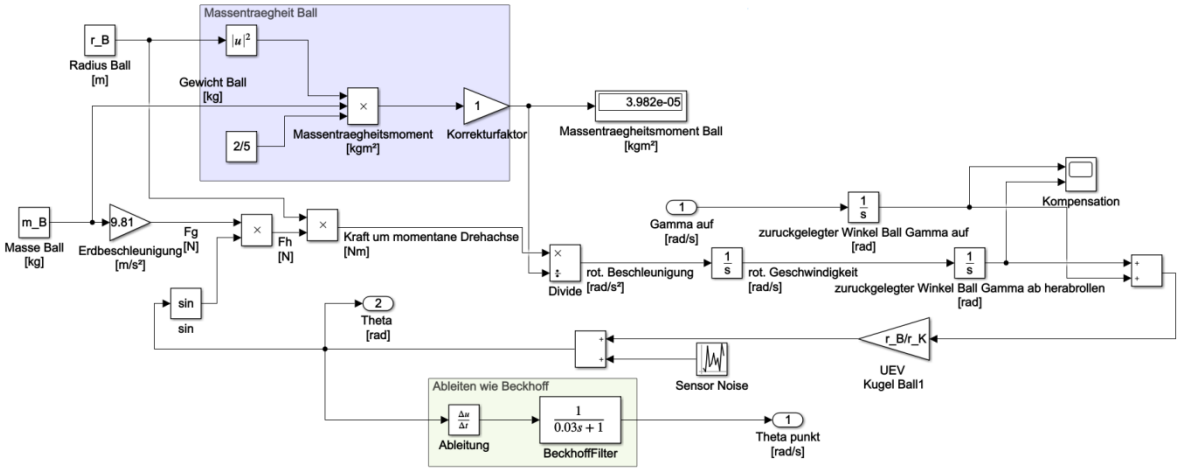

Abbildung 29: Simulink Modell welches die Kopplung zwischen Ball und Kugel simuliert

6.4 Validierung des herabrollenden Balls

Um das Simulink Modell Ball auf Kugel überhaupt validieren zu können, musste dieses anhand von realen Messdaten abgeglichen werden. Die Validierung des herabrollenden Balles konnte nicht so einfach erfolgen, wie zum Beispiel die Validierung der Kugeldrehzahl in Kapitel 6.2. Dies liegt vor allem daran, dass der herabrollende Ball in der Simulation Anfangsbedingungen ungleich 0 besitzt. Abhängig wie die Anfangsbedingung für den zurückgelegten Winkel des Balls und der anfänglichen Ball Winkelgeschwindigkeit gewählt sind kommt es selbstverständlich zu einem unterschiedlichen Ergebnis der Simulation. Um reale Messdaten des herabrollenden Balls erzeugen zu können, wurde dieser in ein kleines Loch[3] der großen Kugel platziert und durch einen kleinen Kraftstoß aus seiner Ruheposition gebracht. Durch diesen kleinen Kraftstoß sind die Anfangsbedingungen für das herabrollen, bezüglich Position und Winkelgeschwindigkeit ungleich null. Mithilfe von pzMove konnten die Anfangsbedingungen bestimmt werden. Hierzu wurde in pzMove als Modellstruktur ein Doppel-Integrator vorgegeben, so wie er auch im realen Simulink-Modell existent ist. PzMove bestimmt aus den vorgegebenen Werten die Anfangsbedingungen der beiden Integratoren bezüglich Winkel und Winkelgeschwindigkeit zum Zeitpunkt t=0 welche, die Modellstruktur des Doppel-Integrators am besten nähert. Eine Validierung des physikalischen Ansatzes erfolgt in Simulink. Hierzu werden die von pzMove bestimmten Anfangsbedingungen in die Integratoren in Simulink eingesetzt und das Ergebnis mit der realen Kurve verglichen.

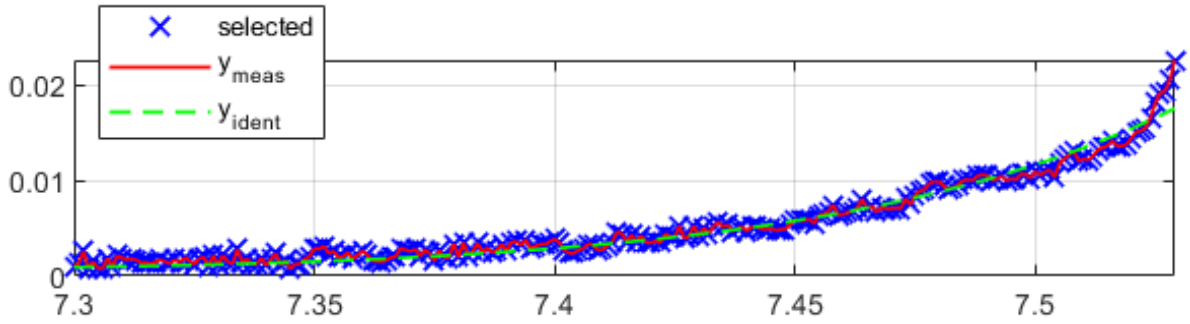

Abbildung 30: Herabrollen des Balls mit korrekten Anfangsbedingungen für den Doppel-Integrator (Zentgraf P. , 2018)

[3] Fertigungsbedingtes Loch

6.5 Kritische Stellungnahme zu den Drehmomenten und der Kugeldrehzahl

In Kapitel 6.1 wird erklärt, dass der Regler ein Drehmoment dem Motor kommandiert. Diese Annahme wird dadurch unterstrichen, dass der Motor sich im CST – cyclic synchronous torque-Betrieb befindet (siehe Kapitel 1.2). Ebenfalls wird geschrieben, dass der Motor bzw. auf die Kugel ein konstantes Drehmoment übersetzt. Eine logische Schlussfolgerung wäre demnach, dass ein konstantes Drehmoment eine konstante Beschleunigung hervorruft, welche ein ununterbrochenes Beschleunigen der Kugel zufolge hätte (soweit in der Theorie). In der Praxis läuft für ein bestimmtes Drehmoment die Drehzahl der Kugel gegen einen festen Wert. Anfangs wurde versucht durch den Einbau einer Systemdämpfung dieses Verhalten nach zu modellieren. Problematisch stellte sich hierbei dar, dass die Systemdämpfung in der Praxis oftmals eine nicht konstante und nichtlineare Größe darstellt. Daher wurde dieses Prinzip verworfen und die Kugel, wie in Kapitel 6.2 mit Look-Up-Tables beschrieben. Dies hatte zum einen den Vorteil, dass die Ungereimtheiten mit dem konstanten Drehmoment nicht mehr von Belang sind. Dieses wird daher als Ansteuersignal interpretiert, wobei dieser Annahme widerspricht die Tatsache, dass die Globale Variable für das gemessene Drehmoment von Beckhoff in der Einheit Nm vorliegt. Andererseits konnte hier auf ein kompliziertes Verfahren zur Ermittlung eines nicht linearen Systemdämpfungskoeffizienten verzichtet werden. Die Simulation wurde daher aufgrund des experimentellen Ansatzes der Kugeldrehzahl so ausgelegt, dass es unerheblich ist, ob es sich um ein Drehmoment oder um ein Ansteuersignal handelt.

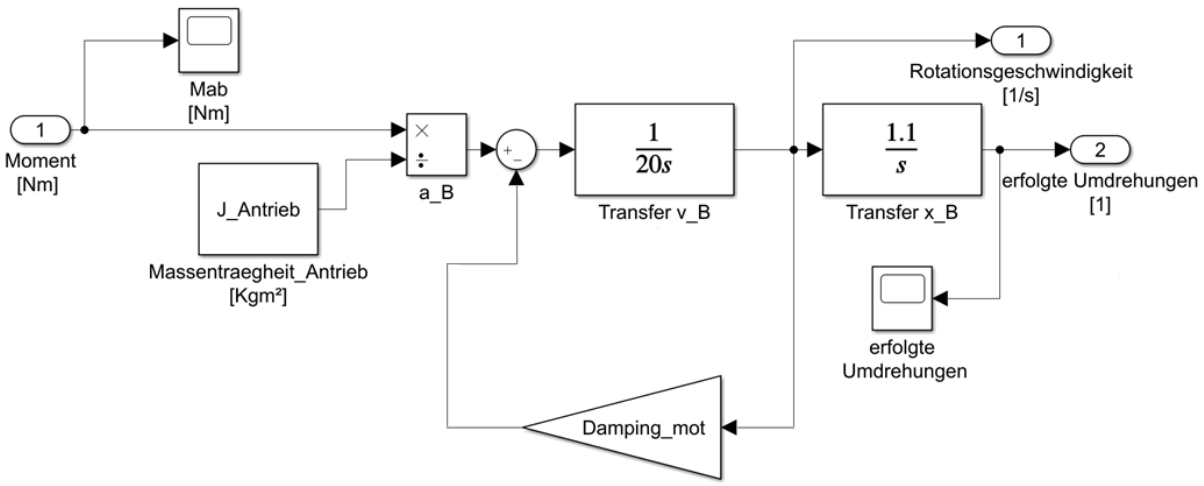

Abbildung 31: Eine der frühen ersten Versuche, einer Realisierung über einen konstanten Dämpfungskoeffizienten

6.6 Numerisches Problem bei der Bildung von θ' und Lösung mit pzMove

Um vom Positionswinkel θ auf die Positionsänderungsgeschwindigkeit θ' zu schließen ist eine Differentiation von Nöten. Für die Simulation sollte die Ableitung exakt so nachgestellt werden, wie Beckhoff diese vollzieht. Daher wurde in Simulink vorerst ein simpler Ableitungsblock ohne Filterung durch ein nachfolgendes PT1-Glied verwendet. Das so erzeugte Signal ist ohne Filter entsprechend verrauscht. Daher wurde mit pzMove eine Übertragungsfunktion gebildet, welche das Ableiten übernimmt. Hierzu wurde als Eingangssignal θ und als Ausgangssignal θ' verwendet.

$$G(s)_{Ableitung} = \frac{32{,}796s + 1{,}82e - 05}{s + 33.33}$$

Dennoch soll der Demonstrationsversuch so detailgetreu wie möglich nachgestellt werden, daher wurde diese Möglichkeit wieder verworfen, obwohl Sie eine brauchbare Alternative darstellt. Ein Durchsuchen der Beckhoff-Function Datenbank lieferte eine Zeitkonstante für den nachgeschalteten Filter. Abbildung 32 vergleicht die beiden Ableitungen mit der echt gemessenen Ableitung. Wie zu sehen ist, ist hier so gut wie kein Unterschied ersichtlich.

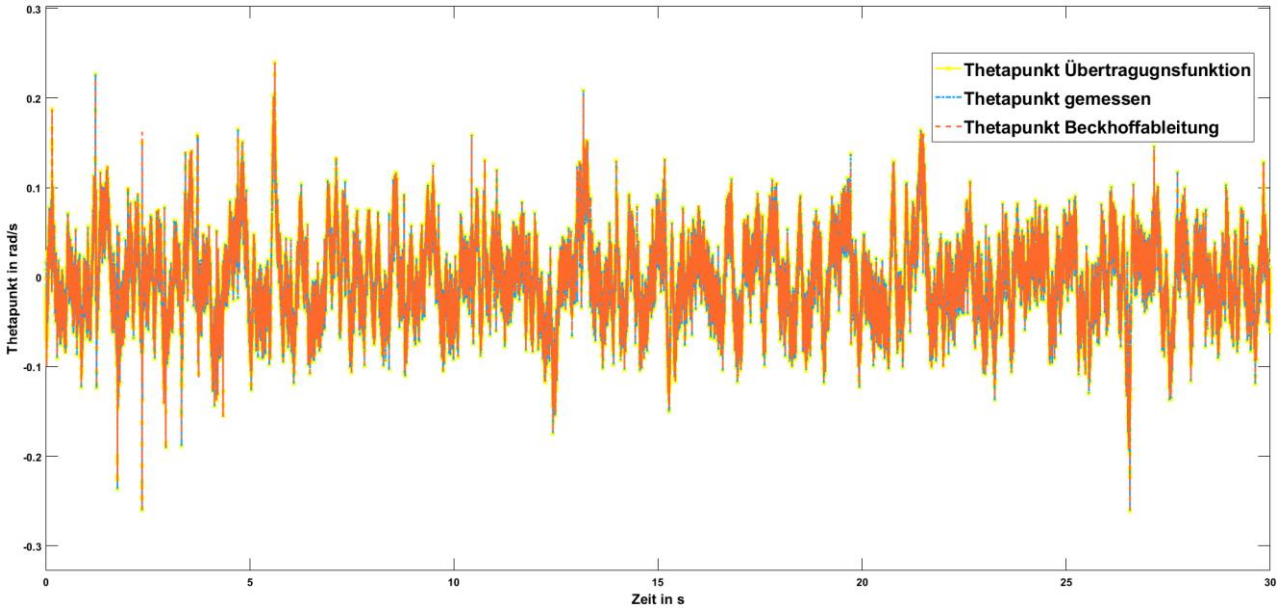

Abbildung 32: Vergleich der Beckhoff Ableitung und der Übertragungsfunktionableitung an der real gemessenen Ableitung

6.7 Ergebnisse der Simulation

Das Simulink Modell kann für γ' im offenen Kreis simuliert werden. Für die beiden instabilen Systemgrößen θ und θ' erfolgt eine Simulation und Validierung parallel zum realen Aufbau, siehe hierzu Abbildung 33. Es wurde getestet, ob auf eine Messung der Kugeldrehzahl verzichtet werden kann und anstelle der Messung das in Simulink erzeugte Modell für die Kugeldrehzahl verwendet werden kann. Für kurze Zeiten (ca. 5-7 Sekunden) kann die

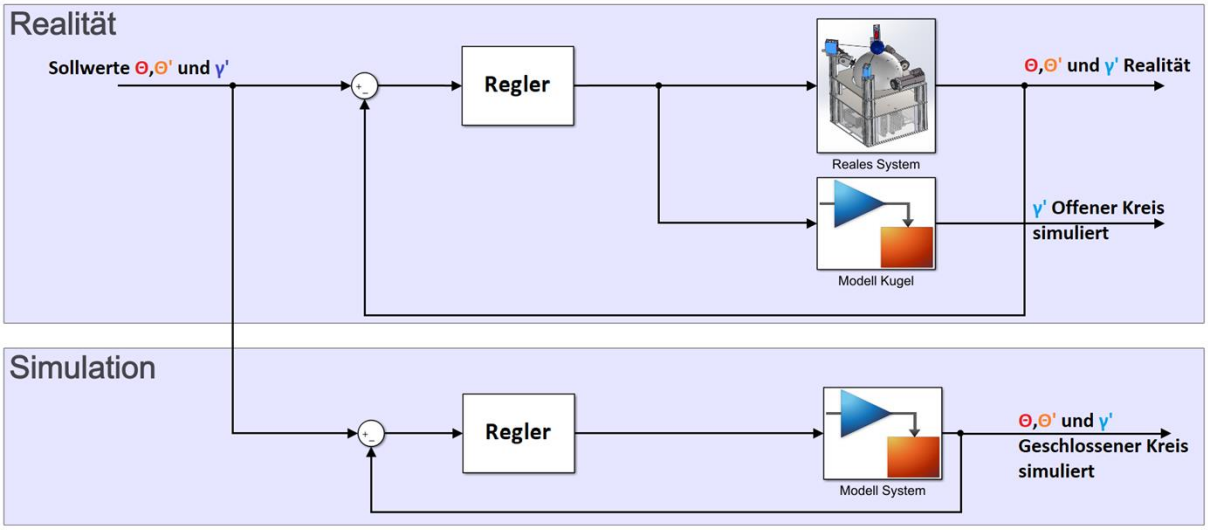

Abbildung 33: Simulationsschema für die Kugeldrehzahl in Open-Loop und im Closed-Loop die instabilen Größen

Rückführung der Kugeldrehzahl durch das Modell erfolgen. Eventuell kann die Kugeldrehzahl sogar durch einen Beobachter ersetzt werden, dies ist noch zu prüfen.

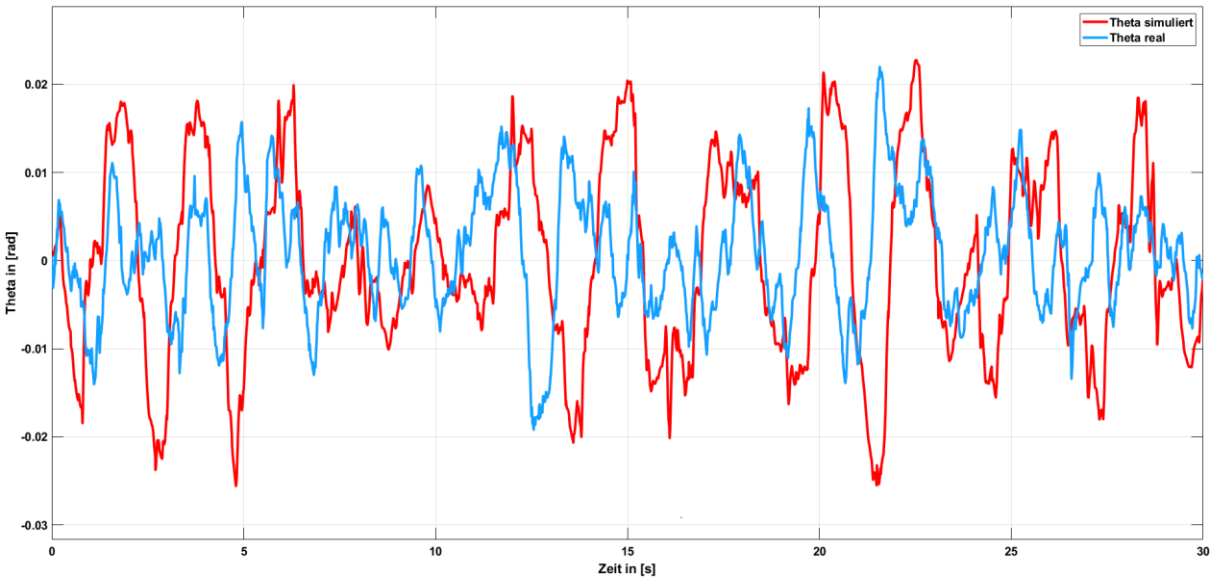

Abbildung 34: Vergleich Simulation und Realität für Θ

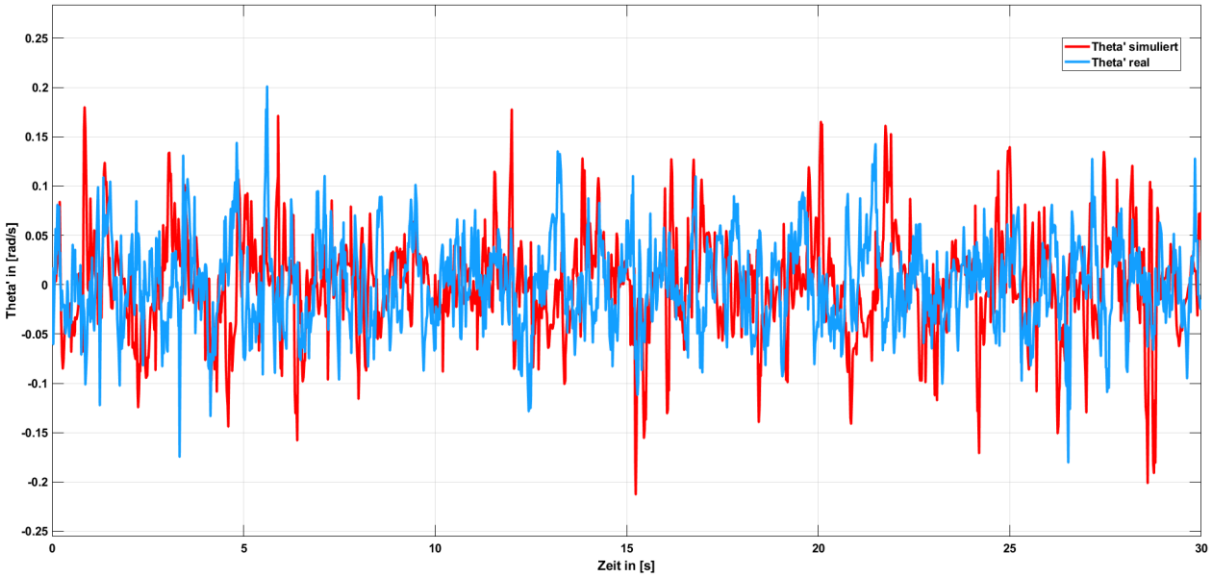

Abbildung 35: Vergleich Simulation und Realität für Θ'

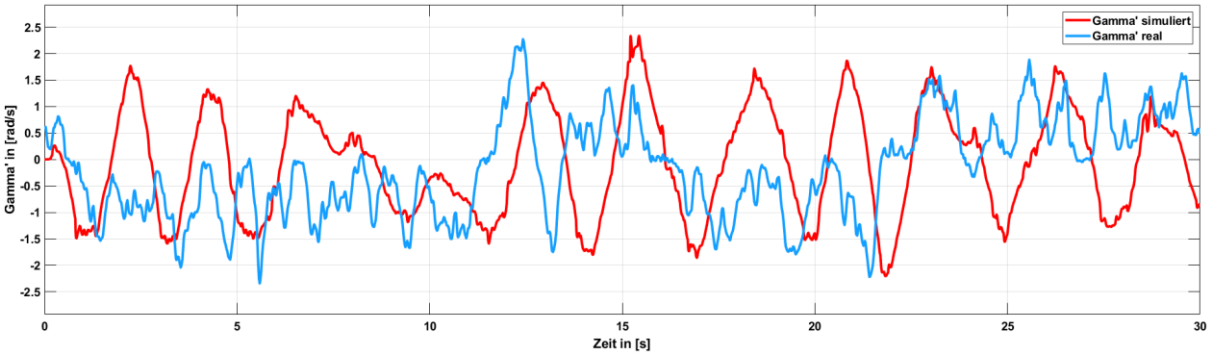

Abbildung 36 Vergleich Simulation und Realität für γ'

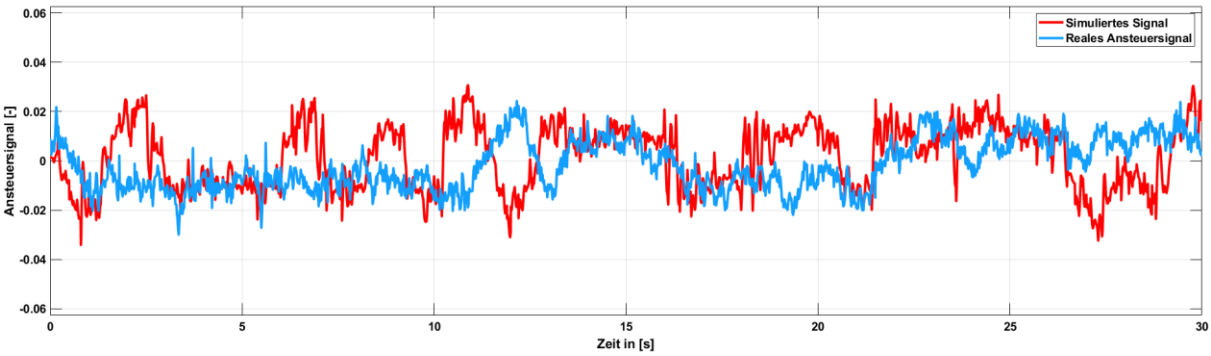

Abbildung 37: Vergleich Simulation und Realität für das Ansteuersignal für die zu resultierende Kugeldrehzahl

7. Virtual Reality Engine

Da der Demonstrationsversuch auf der MATLAB Expo 2018 vorgestellt wurde, war es naheliegend die Simulationsergebnisse in einer möglichst ansprechenden und anschaulichen Form zu präsentieren. Dies könnte selbstverständlich über das reine Aufzeigen von Simulationsergebnissen erfolgen, welche aber nicht denselben ansprechenden Effekt wie eine 3D-Animation besitzen. Daher wurden Überlegungen gemacht die Simulationsergebnisse von Simulink in Echtzeit an die Spiel-Engine Unity zu senden, wie es auch in der Masterarbeit (Zagler, 2013) von Herrn Zagler gemacht wurde. Da es hierfür, aber weder eine offizielle Schnittstelle von Mathworks gibt, hätte auf ein Workaround in Form eines Python Skripts (Bartlett, 2009) zurückgegriffen werden müssen. Da dieses aber nicht in der Lage ist eine annähernd echtzeitfähige Übertragung zu gewährleisten, wurde diese Möglichkeit wieder verworfen. Anstelle dessen wurde auf die eigene Virtual Reality Toolbox von MATLAB zurückgegriffen. Diese bietet zwar nicht den gleichen beeindruckenden Funktionsumfang von Unity, aber es stellte sich sehr schnell heraus, dass dieser auch nicht benötigt wird. Das Arbeiten mit der VR-Toolbox benötigt etwas erweiterte Theorie in Bezug auf Drehungen im 3-dimensionalen Raum, welche davor nicht bekannt war und sich als sehr interessant herausgestellt hat. Um mit dem Virtual World Editor selbst arbeiten zu können wird empfohlen die ersten Beispielvideos (Mahapatra, 2015) von Mathworks durchzuarbeiten.

7.1 Virtual Reality Builder

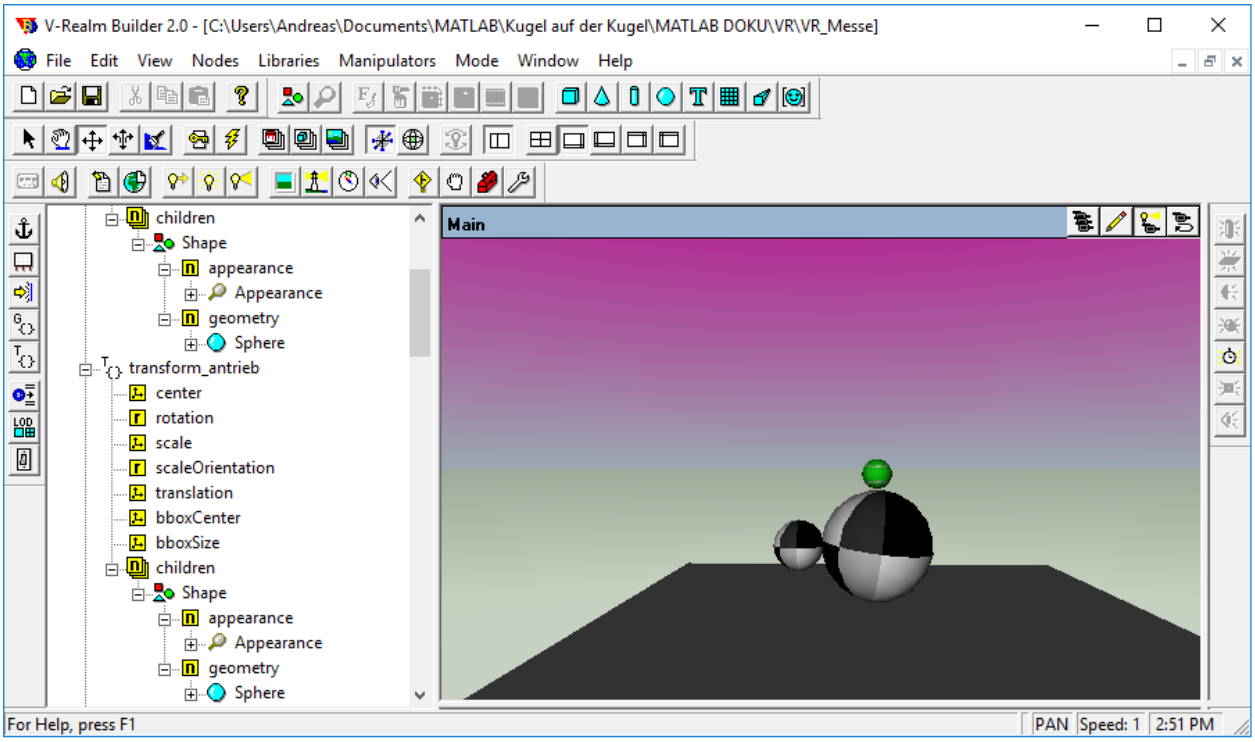

Abbildung 38: V-Realm Builder Umgebung

Bei dem verwendeten Editor zum Erstellen der VR-Umgebung handelt es sich um eine simple CAD Umgebung, welche ermöglicht grundlegende Körper wie Kugeln, Quader und Zylinder zu erstellen. Jeder Körper hat mehrere Parameter, wie zum Beispiel Translation, Rotation und Scale mit denen der Körper im Raum mithilfe von Simulink Signalen manipuliert werden kann.

7.2 Eulersche Drehwinkel und Quaternionen

Rotationen im 3-dimensionalen Raum erfolgen am einfachsten mithilfe von Euler Winkeln. Hierbei wird ein Objekt um einen bestimmten Winkelwert um die X-, Y- oder Z-Achse gedreht. Die Reihenfolge der Drehungen ist hier von relevanter Bedeutung um die korrekte Endlage eines Objekts im Raum festlegen zu können. Eine gängige Rotationsvorschrift erfolgt hier nach der ZYX-Konvention, welche in der DIN70000 Koordinatensystem des Fahrzeugs festgelegt ist (Bernd Heißing, 2011).

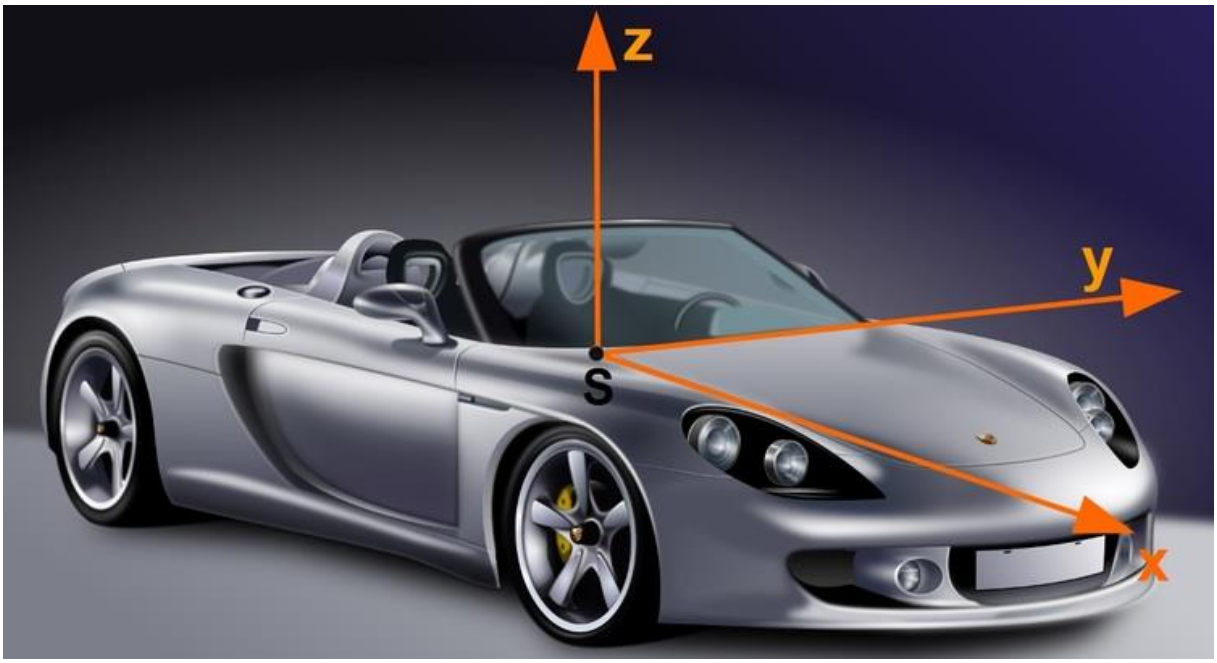

Abbildung 39: Fahrzeugkoordinatensystem nach der DIN70000 (Ingenieurkurse, 2017)

Die Rotationsmatrix für Euler Winkel setzt sich aus drei Matrizen zusammen, welche die drei Achsen repräsentieren.

Z-Achse für das Gieren[4] ψ
Y-Achse für das Nicken[5] ζ
X-Achse für das Wanken [6] ϕ

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{pmatrix} * \begin{pmatrix} cos\zeta & 0 & sin\zeta \\ 0 & 1 & 0 \\ -sin\zeta & 0 & cos\zeta \end{pmatrix} * \begin{pmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Durch Multiplizieren eines beliebigen Punktes, oder Körperkoordinatensystems mit der Rotationsmatrix, kann das Objekt gedreht werden.

[4] Engl. Yaw
[5] Engl. Pitch
[6] Engl Roll

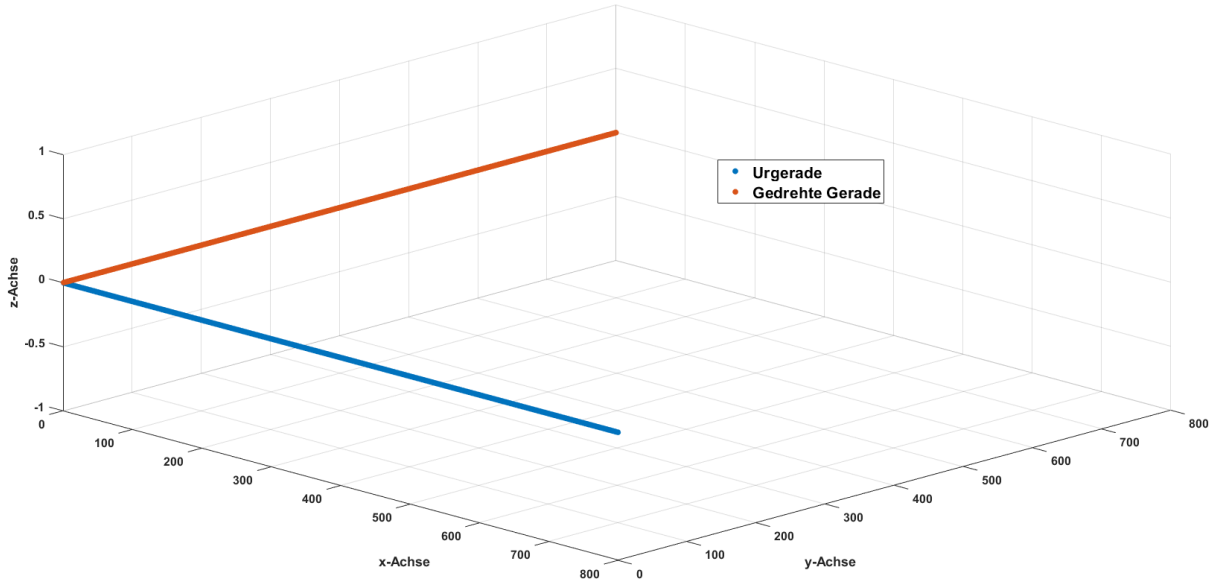

Abbildung 40: Ursprungsgerade wurde um 90° um die Z-Achse gedreht

Die Drehung mit Euler Winkeln ist einfach nachzuvollziehen, bringt aber die sogenannte Problematik der Kardanischen-Blockade[7] mit sich. Diese entsteht, wenn zwei der möglichen drei Drehachsen sich in einer Ebene befinden. Beim Auftreten einer Kardanischen-Blockade verliert das System einen Freiheitsgrad. Das Resultat ist, dass es zu keiner weiteren Drehung kommen kann. Das zu drehende Objekt würde seine Lage nicht ändern können. Eine gute Veranschaulichung bietet folgendes Video (CG, 2009).

Um diese Problematik mit Euler Winkeln zu vermeiden wird auf die Angabe der Drehung als Quaternion zurückgegriffen. Quaternionen sind in der Theorie eng mit komplexen Zahlen „verwandt“. Die Multiplikation einer komplexen Zahl mit der Länge 1 entspricht hierbei einer Drehung in einer Ebene. Nachfolgend beispielhaft zu sehen:

$$q_0 + q_1 i$$

Um eine Drehung im dreidimensionalen Raum mithilfe einer Quaternion beschreiben zu können, bedarf es also 4 Parameter[8]. Eine Quaternion folgt daher folgender Struktur:

$$q_0 + q_1 i + q_2 i + q_3 i$$

Die Simulink VR-Umgebung erwartet für Rotationen eine Eingabe, als Quaternion. Eine Umwandlung von Euler Winkeln kann in MATLAB einfach mit dem Befehl `eul2quat()`[9] erfolgen.

[7] Engl. Gimbal Lock
[8] Realteil+3 mögliche Drehungen
[9] Die Robotic Toolbox ist für diesen Befehl erforderlich

Um Euler Winkel als Quaternion darzustellen, bedarf es einer Transformation der Euler Winkel ϕ, ζ und ψ. Nachfolgende Transformationsmatrix (Kuipers, 2002) folgt wieder der ZYX Konvention.

$$\phi: p = \begin{pmatrix} \cos(-\frac{p}{2}) \\ \sin(-\frac{p}{2}) \\ 0 \\ 0 \end{pmatrix}$$

$$\zeta: h = \begin{pmatrix} \cos(-\frac{h}{2}) \\ 0 \\ -\sin(-\frac{p}{2}) \\ 0 \end{pmatrix}$$

$$\psi: b = \begin{pmatrix} \cos(-\frac{b}{2}) \\ 0 \\ 0 \\ -\sin(-\frac{b}{2}) \end{pmatrix}$$

Die Spaltenvektoren p,h und b müssen noch ausmultipliziert werden und ergeben anschließend das gesuchte Quaternion für die gewünschten Euler Winkel

$$q = b * h * p$$

$$q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

$$q = q_0 + q_1 + q_2 + q_3$$

7.3 Simulink Virtual Reality Logik in Simulink

Wie in Kapitel 7.1 beschrieben wurde, können die Körpereigenschaften der Objekte im 3-dimensionalen Raum mithilfe von Simulink manipuliert werden. Hierzu wurde ein Logik-Subsystem erstellt, welches die Parameter aus der Simulation so verarbeitet, dass diese korrekt interpretiert werden können. Dieses Subsystem beinhaltet die Umrechnung von den beiden Positionswinkeln θ_x und θ_y in kartesische Koordinaten. Hierzu wurde fast identisch der Code, der Position Visualizer GUI (siehe Kapitel 1.3) übernommen und auf das nötigste reduziert. Die Umrechnung der Positionswinkel in Simulink erfolgt durch eine hier implementierte MATLAB-Function in Simulink. Die Rotation des Balls und der Kugel von Euler Winkel in Quaternionen wird auch hier von einer MATLAB-Function ausgeführt. Die Animation der Antriebsräder kann in Euler-Winkel erfolgen, da hier nur um eine Achse gedreht werden muss.

Um die Animation und die Simulation parallel zum realen Demonstrationsversuch ausführen zu können wurde noch ein „Failure-Trigger“ erstellt. Mithilfe der in Kapitel 1.3 beschriebenen Schnittstelle, können Messdaten von der Beckhoff CPU in Simulink eingelesen und verarbeitet werden. Sollte es zu einem stoppen bzw. Versagen des Prüfstands kommen, so stoppt auch die Simulation und in der VR-Umgebung kommt es zu einem Herabfallen des Balls. Bei einem wiederaufnehmen der Regelung ist es auch erforderlich die Simulation neu zu starten.

```
function y = fcn(u,v)
%
% u= 0.18;% Radiant von beckhoff in bogenmass fuer x achse
% v= 0.18; % Radiant von beckhoff in bogenmass fuer y achse
thetaxRAD= (pi/2)-u;
thetayRAD= (pi/2)-v;
r= 105; % Fiktiver Polarer Radius
polarx= r*cos(thetaxRAD); % Liefert mir die Kartesische XKoordinate
polarz= r*sin(thetaxRAD); % Liefert mir die Kartesische YKoordinate
polary= r*cos(thetayRAD);
polarz_Y= r*sin(thetayRAD);
%% Koordinatenschnittpunkt
%Schnittgerade aus den beiden Ebenen
EsStuetz=[polarx;polary;0]; %Schnittgerade Stuetzpunkt
% EsRichtung=[0;0;1];%Richtungsvektor Schnittgerade
%x²+y²+z²=r² hier wird einfach Es eingesetzt fuer die xyz komponenten
zBall= sqrt(r^2-(EsStuetz(1)^2+EsStuetz(2)^2)); %Mathematisch gesehen existieren hier zwei loesungen
%(Schnittpunkte gerade/Kugel) die zweite Loesung hat einen negativen y wert und ist unrelevant
KartesianBall=[polarx,polary,zBall];
y = KartesianBall;
```

Abbildung 41: MATLAB-Function "Polar2Kartesian". Wandelt die beiden θ Winkel in kartesische Koordinaten um.

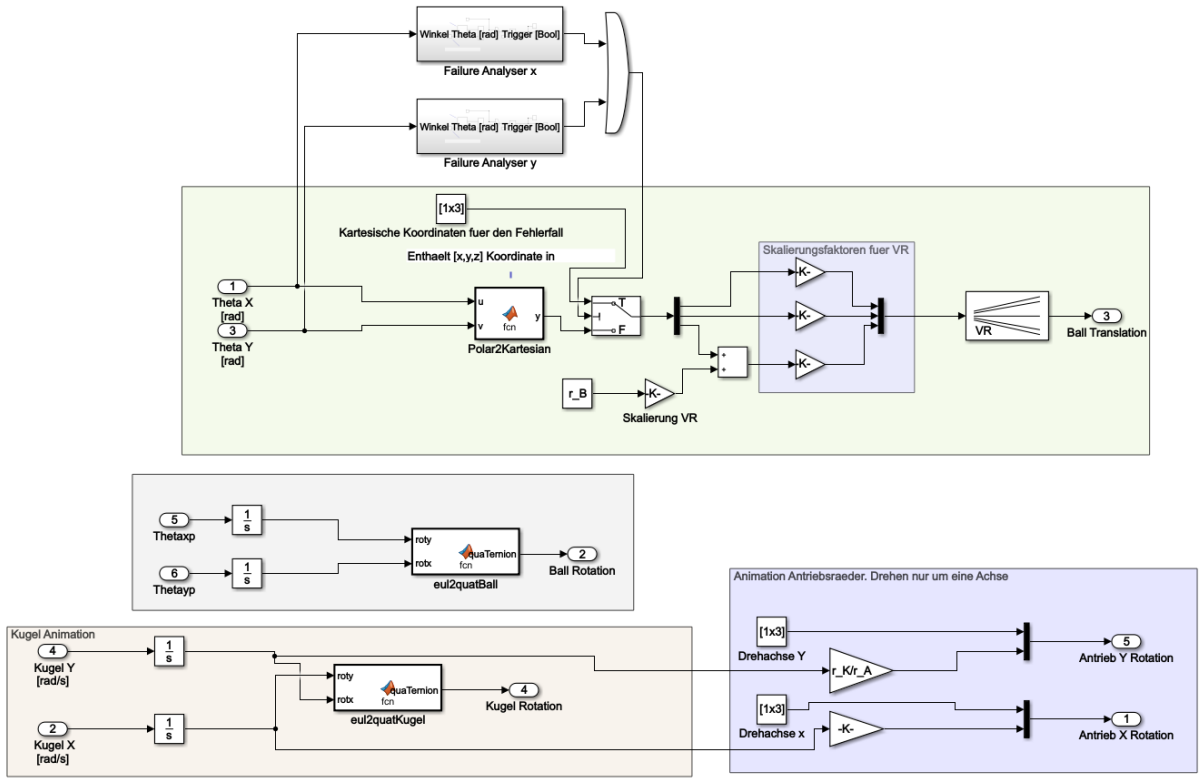

Abbildung 42: Simulink System, welche die Logik der VR darstellt.

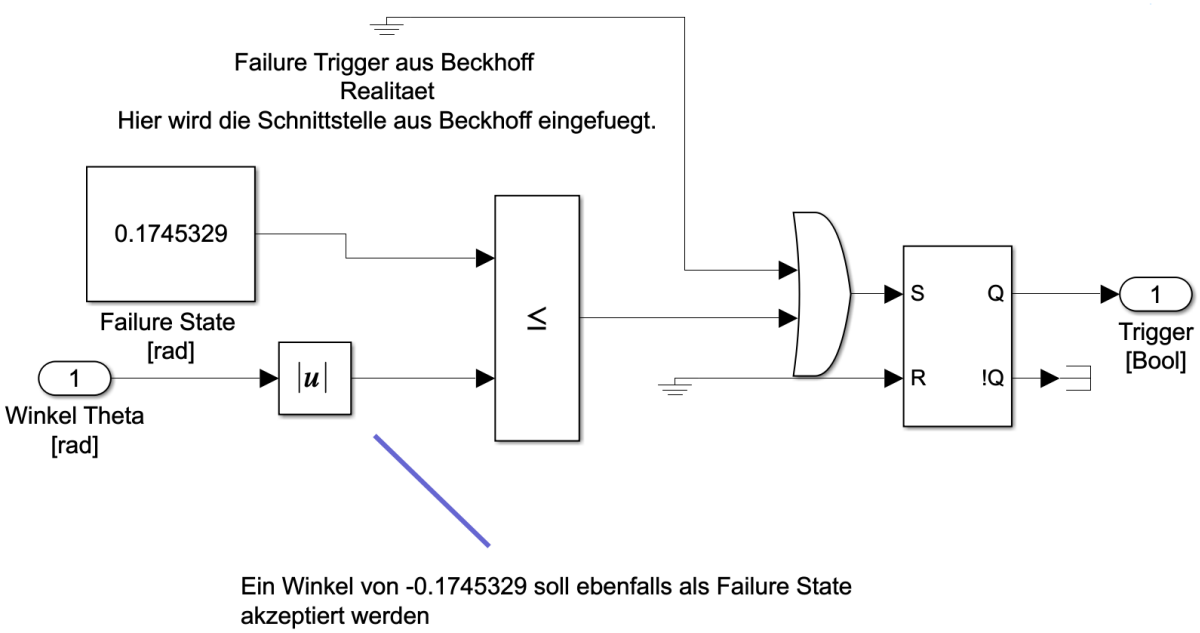

Abbildung 43: Failure Analyzer um zu erkennen, ob der Ball in der Realität ebenfalls von der Kugel fällt.

8. Schnittstellen

Die komplette Regellogik und Messwerterfassung des Demonstrationsversuches wird von der Beckhoff CPU übernommen. Um bequem verschiedene Regelungsalgorithmen ausprobieren zu können soll der Demonstrationsversuch komplett auf eine Regelung durch MATLAB umgestellt werden. Das Ziel ist es daher Beckhoff nur noch als Hardware Target zu nutzen, welche die in Simulink generierten Ansteuersignale in ein reales Signal für die Motoren umwandelt. Um dies realisieren zu können, bedarf es eine der beiden Erweiterungen für Beckhoff, für welche zuerst eine Lizenz[10] angefordert werden muss. Die beiden hierfür benötigten Lizenzen sind die TE1400[11] oder die TE1410[12], welche separat in Beckhoff unter *License* hinzugefügt werden müssen.

8.1 TE1410

Um Daten von Beckhoff in die Simulink-Entwicklungsumgebung zu transferieren, oder von Simulink Daten nach Beckhoff ist es erforderlich die Erweiterung TE1410 zu installieren. Diese ist zum einen in der Beckhoff Umgebung unter „Licensing“ hinzuzufügen und in MATLAB zu installieren. Eine detaillierte Installationsanleitung findet sich hier in Anhang C[13]. Die TE1410 ermöglicht einen Datenaustausch zwischen der Beckhoff CPU und MATLAB. Hierzu wird in MATLAB eine ADS-Schnittstelle generiert, welche die von Beckhoff gesendeten Daten in Simulink überträgt. Zum Zeitpunkt der Projektarbeit, war die Schnittstelle noch nicht für die MATLAB Version R2018a verfügbar, daher wurde diese für die Version R2017b installiert. Sollte die Schnittstelle genutzt werden wollen, so ist es auch unbedingt erforderlich ein Simulink Modell mit der MATLAB Version R2017b zu erstellen.

Nach erfolgreicher Installation der TE1410 sollte im Simulink Library Browser eine neue Library auftauchen mit dem Namen „TwinCAT ADS“.

Der Baustein „TC ADS Symbol Interface“ erlaubt das einlesen von Beckhoff Variablen in Simulink.

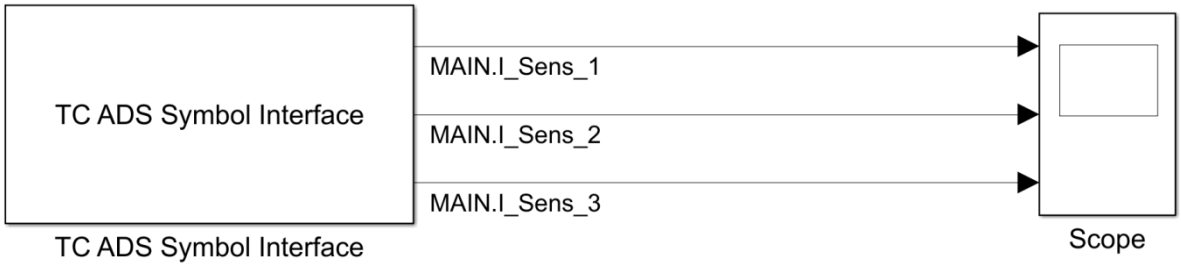

Abbildung 44: Beispielhaftes Modell zum Einlesen von Beckhoffvariablen in Simulink

[10] z.B.7-Days-Trial-License
[11] TC3 Target for MATLAB/SIMULINK
[12] TC3 Interface for MATLAB/Simulink
[13] PDF: MATLAB_Doku_V1 beginnend ab S.71 detaillierte Installationsanleitung

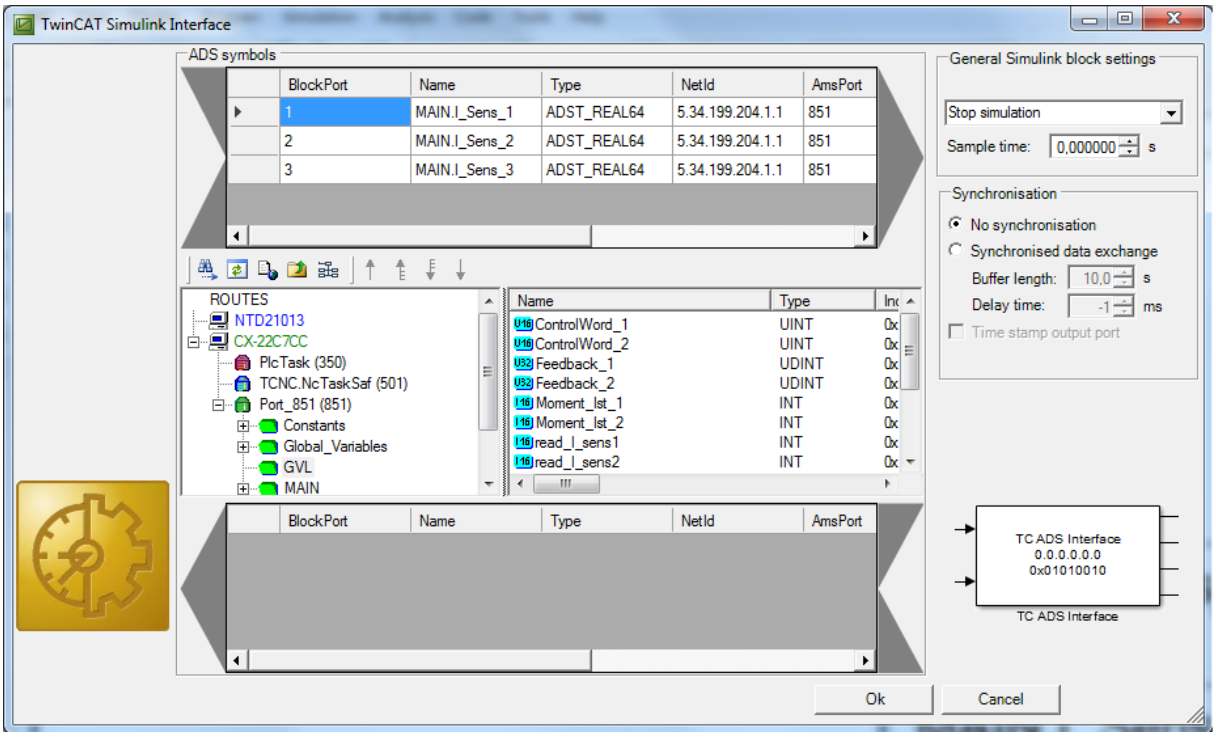

Abbildung 45: Detailansicht des TC ADS Interfaces

Abbildung 45 zeigt die Detailansicht des TC ADS Interfaces, welches entsprechend konfiguriert werden muss. Hierzu sind bei korrekter Verbindung des Demonstrationsversuchs mit dem Computer unter dem Reiter *Port_851* alle im Demonstrationsversuch beschriebenen Variablen gelistet. Diese können per Drag & Drop in das Feld ADS Symboles gezogen werden. In Abbildung 45 wurden zum Beispiel die drei Sensorabstände hineingezogen. Durch ein klicken des Ok-Buttons werden drei Ausgänge am TC ADS Interface Block erzeugt (siehe Abbildung 44). In Simulink kann nun die Simulation Time auf *Inf* eingestellt werden, welches ein anzeigen der Daten in Simulink Scopes ermöglicht. Im Anhang B findet sich ein vorkonfiguriertes Beispielprojekt „Simple ADS“. Dieses Beispielprojekt steuert die beiden Motoren mit einer Sinusschwingung an. Auf die Beckhoff CPU muss davor, das Programm „Standalone_Simulink“ geladen werden.

8.2 TE1400

Mit der Erweiterung TE1400 kann eine Codegeneration von MATLAB für Beckhoff erfolgen. Ein in Simulink erstelltes Modell wird hierbei mithilfe des von MATLAB integrierten C++-Compilers, als Programm direkt in der Beckhoff CPU generiert. Ein Versuch der Implementierung der TE1400 führte bis jetzt immer zu einem unkontrollierten Verhalten des Demonstrationsversuchs, sodass dieser wieder mit einem alten Backup wiederhergestellt werden musste. Zum momentanen Stand[14] dieser Projektarbeit, konnte eine Schnittstelle von Simulink zu Beckhoff mit der TE1400 noch nicht vollständig erzeugt werden. Die Implementierung der TE1400 soll dennoch hier soweit wie möglich erläutert werden, da diese sich nicht als so trivial darstellt, wie die Implementierung der TE1410 (siehe Kapitel 8.2).

Als getestete MATLAB Version wurde der Release 2017b verwendet, da dieser sich als kompatibel mit dem Visual Studio C++ Compiler erwies, welcher von Beckhoff benötigt wird. Der Visual Studio C++ Compiler ist nicht für jede MATLAB Version aufwärts[15], bzw. abwärts kompatibel. Der Visual Studio C++ Compiler befindet sich in Anhang C. Ansonsten kann dieser auch in MATLAB unter der Sektion *Get Addons* hinzugefügt werden. Mit der Anweisung `mex -setup` kann der gewünschte Compiler als Standard Compiler für die MATLAB Codegeneration für C++ Code verwendet werden.

Um für die Beckhoff CPU überhaupt Code generieren zu können, ist es erforderlich spezifische Gerätehardware Treiber für Windows zu installieren. Hierfür muss von Microsoft das „Windows Driver Kit“ Version 7.1.0 heruntergeladen werden und als ISO mit entsprechender Software „gemountet“ werden. Das „Windows Driver Kit“ Version 7.1.0“ befindet sich in Anhang C, es handelt sich hierbei um eine Version für ein 64-Bit Betriebssystem. Bei der Installation des „Windows Driver Kit“ ist es ausreichend lediglich die „Build Environments“ zu installieren. Es ist unbedingt darauf zu achten den Installationspfad des Driver Kit auf dem Laufwerk C:\ anzulegen. Der vorgeschlagene Dateipfad könnte folgendermaßen aussehen: C:\WinDDK\7600.16385.1

Im Anschluss ist in Windows unter Systemsteuerung->System->Erweiterte Systemeinstellungen eine neue Umgebungsvariable zu definieren, welche die entsprechenden Gerätetreiber beinhaltet. Durch klicken auf die Schaltfläche „Neu“, bei den Systemvariablen kann hier eine neue hinzugefügt werden. Es ist wichtig, dass der Wert der Variable dem Installationsverzeichnis entspricht siehe Abbildung 46.

[14] Stand 04.09.2018
[15] Stand Version 2018b

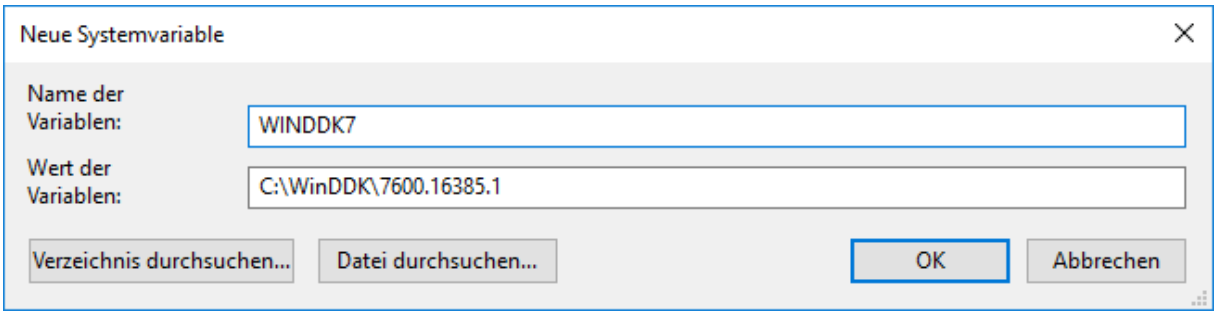

Abbildung 46: beispielhaftes Anlegen einer neuen Systemvariable in Windows

Der Computer muss im Anschluss neugestartet werden. Abschließend kann die TE1400 unter MATLAB, wie die TE1410 durch ein ausführen der TE1400.py installiert werden. Um die TE1400 korrekt ausführen zu können wird die Visual Studio Professional Version benötigt. Die Shell Variante ist hier nicht ausreichend.

9. Fehlerbehebungen am Demonstrationsversuch

Im Laufe der Projektarbeit kam es oft zu dem Fall, dass der Demonstrationsversuch keine Verbindung zum Computer herstellen konnte. Eine entsprechende Fehlermeldung verweist hier auf den Fehler, dass die ADS-Kommunikation zwischen Computer und Demonstrationsversuch nicht hergestellt werden konnte. Momentan wird sich hierbei immer noch mit dem Ein- und Ausschalten des Demonstrationsversuchs beholfen. Eine Anfrage des Beckhoff Supports verwies hierbei auf eine Asynchronität der ADS-Kommunikation. Um einen neuen Verbindungsversuch zu initialisieren muss die Spannungsversorgung getrennt werden. Wenn alle Status LEDs der CPU erloschen sind, kann die Spannungsversorgung wiederhergestellt werden. Alternativ hierzu kann zuerst die CPU neugestartet werden und im Anschluss kann das momentane Projekt Neugeladen werden. Weitere Dokumentation zur Netzwerkkonfiguration ist der Doku von Frau Michaela Huber und Herrn Adrian Zeitler zu entnehmen. Diese findet sich im Anhang D.

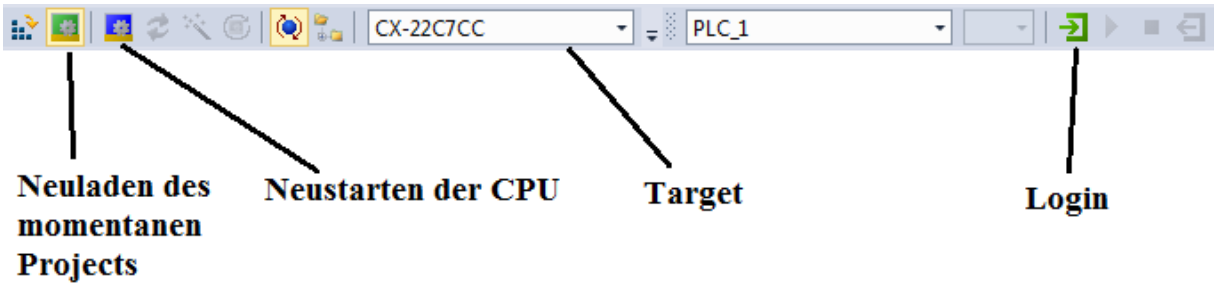

Abbildung 47: Menüleiste der TwinCat Umgebung

Wenn eine korrekte Verbindung zwischen Demonstrationsversuch und Computer hergestellt werden konnte, dann kann über den „Login Button“ welcher in Abbildung 47 zu sehen ist eine Visualisierung der vom Demonstrationsversuch gemessenen Größen erfolgen. Alle für die Installation benötigten Dateien wurden in Anhang C zusammengefasst.

10. Fazit und Ausblick

Die Projektarbeit „Untersuchung verschiedener Modellierungsansätze anhand des Balls auf Kugel Experiments“ stellte sich als äußerst anspruchsvoll heraus. Insbesondere die Tatsache, dass bezüglich der Funktionsweise und Dokumentation des Demonstrationsversuches ein hohes Maß an „Reverse Engineering“ betrieben werden musste. Zu Beginn der Projektarbeit wurde zu lange versucht aus Sprungantworten eine Übertragungsfunktion für den Ball auf der Kugel zu generieren. Dies war nicht von Erfolg und lag vermutlich daran, dass die Zeit, die gemessen werden kann zu kurz ist. Persönlich hat sich hier für mich gezeigt, dass für instabile Systeme, welche nur „extrem“ kurz gemessen werden können eine experimentelle Systemidentifikation nicht immer gelingen muss und hier eher der analytische Ansatz gewählt werden sollte. Dies kann jedoch nicht pauschal verallgemeinert werden, sondern kommt ganz auf das vorliegende instabile System an. Generell ist der analytische Modellierungsansatz nach persönlicher Einschätzung für dieses System die bessere Vorgehensweise und würde bei ähnlichen Problemstellungen von mir abhängig von deren Komplexität favorisiert werden. Da ohnehin zu Beginn der Projektarbeit eine „Black-Box“ aufgefunden wurde ist hier ein ganz klarer Vorteil eines analytischen Systems, dass dieses den Demonstrationsversuch in seiner Arbeitsweise verständlicher macht.

Das Erstellen des analytischen Simulink Modells war mit Problemen behaftet, da manche Ansteuersignal des Demonstrationsversuches nicht komplett verstanden wurden, bzw. logischen Ansätzen widersprechen (siehe Kapitel 6.5).

Bezüglich der Schnittstellen konnte die TE1410 vollständig implementiert werden. Das komplette Beckhoff Programm wurde Stück für Stück nach MATLAB/Simulink geportet und die Schnittstelle konnte erfolgreich implementiert werden. Die TE1400 konnte noch nicht zur vollen Funktionsfähigkeit gebracht werden. Hier ist jedoch noch zu prüfen, ob die TE1400 überhaupt vollständig gebraucht wird. Es ist für die Regelung auch ausreichend Daten mithilfe der TE1410 über die ADS-Schnittstelle zu manipulieren und somit nur einen Regelkreis zu erstellen, welcher keine Echtzeitfähigkeit aufweist. Das Arbeiten mit der TE1410 stellt auch hinsichtlich der Aussicht auf Praktikumsversuche zum Demonstrationsversuche, die komfortablere Erweiterung dar. Hierzu wurde zusätzlich implementiert das Eingangsgrößenstörungen auf den Demonstrationsversuch aufgebracht werden können.

Das Arbeiten mit der Virtual Reality Engine von Simulink war anfangs nicht als Zielsetzung der Projektarbeit definiert. Die VR Engine stellte sich jedoch als interessantes Werkzeug heraus um Modelle einfach und schnell visuell darstellen zu können. Daher wurde insbesondere die Theorie zu Drehungen im Raum mit Quaternionen als interessant empfunden.

Es sei an dieser Stelle mein persönlicher Dank an Herrn Professor Zentgraf ausgesprochen für die Möglichkeit den Demonstrationsversuch auf der MATLAB Expo 2018 vorstellen haben zu dürfen.

Literaturverzeichnis

Bartlett, K. (11. Juni 2009). *www.mathworks.com*. Von https://de.mathworks.com/matlabcentral/fileexchange/24418-py_udp abgerufen

Beckhoff. (30. November 2015). *Beckhoff.* Von https://download.beckhoff.com/download/Document/io/ethercat-terminals/el72x1-0010de.pdf abgerufen

Bernd Heißing, M. E. (2011). *Fahrwerkhandbuch.* Wiesbaden: Springer Verlag.

Bohn, U. (2016). *Identifikation dynamischer Systeme.* Hannover und Bochum: Springer Vieweg.

CG, G. (14. Januar 2009). *Youtube*. Von https://www.youtube.com/watch?v=zc8b2Jo7mno&t= abgerufen

Filamentworld. (04. Juni 2018). *https://www.filamentworld.de/3d-druck-wissen/was-ist-abs/*. Von Filamentworld: https://www.filamentworld.de/3d-druck-wissen/was-ist-abs/ abgerufen

Hermann, E. D. (kein Datum). *www.ti.tuwien.ac.at.* Von https://ti.tuwien.ac.at/cps/teaching/courses/dspv/files/FIRFilter.pdf abgerufen

Hofmann, P. D. (31. März 2017). *www.profhof.com.* Von http://profhof.com/sensor-daten-aufbereitung/ abgerufen

Holzner, L. (kein Datum). *Dokumentation Leonhard Holzner Unterlagen.*

Huber/Zeitler. (2017). *Ball auf Kugel Dokumentation.* Rosenheim.

Ingenieurkurse. (27. Oktober 2017). *Ingenieurkurse*. Von https://www.ingenieurkurse.de/fahrzeugtechnik/fahrzeugklassen/koordinatensysteme-in-der-fahrzeugtechnik.html abgerufen

Kuipers, J. B. (2002). *Quaternions and Rotation Sequences.* United States of America: Princeton University Press.

Mahapatra, S. (2015). *www.mathworks.com*. Von https://de.mathworks.com/products/3d-animation/videos.html abgerufen

Schumann. (22. Oktober 2016). *Schumann.* Von Uni-Leipzig: http://www.math.uni-leipzig.de/~schumann/calc/moment.pdf abgerufen

Zagler, S. (9. Januar 2013). *Fh Rosenheim.* Von https://www.fh-rosenheim.de/fileadmin/user_upload/Fakultaeten_und_Abteilungen/Fakultaet_ING/Laboratorien/Labor_fuer_Mess-_und_Regelungstechnik/Abschlussarbeiten/Entwicklung-einer-3D-

Simulationsumgebung-zur-Darstellung-von-Hagelabwehr-Fluegen-Zagler_S.pdf abgerufen

Zentgraf. (12. Oktober 2016). *FH-Rosenheim.* Von Praktikumsbeschreibung Systemidentifikation. abgerufen

Zentgraf, P. (2018).