Lehrstuhl für Raumfahrttechnik

Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Technische Universität München

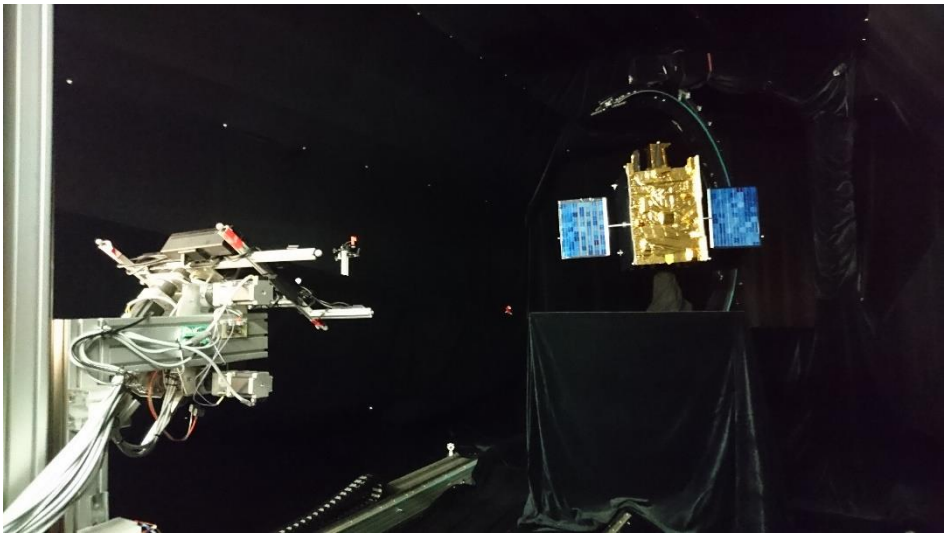
Bachelorarbeit

Überarbeitung und Implementieren einer intuitiven LabVIEW-Steuersoftware für eine robotische Testumgebung

Restructuring and Implementation of an intuitive LabVIEW Control Software for a Robotic Testbench

RT-BA 2018/13

Autor: Elias Plohmann



Betreuer: Dipl.-Ing. Martin Dziura
Lehrstuhl für Raumfahrttechnik / Institute of Astronautics
Technische Universität München

Danksagung

Ich möchte mich an dieser Stelle bei alldenjenigen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Insbesondere bedanke ich mich bei meinem Betreuer Dipl.- Ing. (Univ.) Martin Dziura für eine interessante Aufgabenstellung und seine fachliche Unterstützung während des gesamten Bearbeitungszeitraums.

Ein besonderer Dank gilt meiner Familie für den Rückhalt und die Unterstützung während des gesamten Studiums.

Zusammenfassung

Das RACOON-Lab ist ein Simulator für Nahbereichsoperationen von Satelliten im geostationären Erdorbit. Dessen Hardware-in-the-Loop Testumgebung ist eine robotische Anlage, bestehend aus drei Subsystemen mit insgesamt 13 steuerbaren Freiheitsgraden. Das Ziel dieser Arbeit ist das Implementieren von neuem Code in die bestehende LabVIEW-Steuerungssoftware dieser Anlage. Bestehende Funktionen und Eigenschaften sollen dabei beibehalten oder verbessert werden.

Um diese Vorgaben erreichen zu können, ist über das bloße Implementieren hinaus ein Überarbeiten des bestehenden Codes erforderlich, um die Systemressourcen eines embedded Controller, der die Anlage steuert, effizienter zu nutzen. Der Hauptfokus liegt dabei auf das Einführen einer asynchronen Steueroberfläche auf einem Host-PC. Dadurch werden Programmteile vom Controller auf den Host-PC übertragen, und so der Controller im Betrieb weniger ausgelastet.

Die neuen Funktionen beinhalten das Echtzeit-Logging von internen Programmwerten, eine Schnittstelle für Closed-Loop-Feedback von Anlagenparametern, einen Betriebsmodus zum Nachfahren benutzerdefinierter sinusförmiger Bewegungen, sowie einen Betriebsmodus zum Einlesen und Ausführen individueller Trajektorien-Dateien. Darüber hinaus wurde auch die Benutzeroberfläche überarbeitet, um diese übersichtlicher und deren Bedienung intuitiver zu machen.

Es sind alle oben genannte Funktionen implementiert worden, allerdings teilweise unter kleinen Einschränkungen von deren ursprünglich geplanten Eigenschaften. Durch das Überarbeiten ist die CPU-Last des embedded Controller im Betrieb von rund 99 % Auslastung auf rund 60 % gesunken. Erste Messungen lassen auf eine Anlagenpräzision von unter 1 mm bzw. 0,3 Grad vermuten. Bestätigt werden kann, je nach Achse, jedoch lediglich minimal 3,5 mm bzw. 8,1 Grad. Durch frei gewordene Systemressourcen können nun weitere neue Funktionen im embedded Controller implementiert werden. Zusätzlich steht jetzt mit dem Host-PC eine zusätzliche Plattform mit deutlich mehr Kapazität für nichtzeitkritische, aufwändigere Funktionen zur Verfügung.

Abstract

The RACOON Lab is a simulator for proximity operations of satellites in geostationary orbit. Its hardware-in-the-loop test environment is a robotic system consisting of three subsystems with a total of 13 controllable degrees of freedom. The goal of this work is to implement new code into the existing LabVIEW control software of the simulator. Existing functions and properties should be maintained or even improved.

In order to achieve these goals, not only implementation but reworking of the existing code is required to more efficiently utilize the system resources of the embedded controller operating the unit. The focus while doing so is on introducing an asynchronous control surface on the host PC. This means part of the original code now runs on the Host-PC, which results in a less busy controller during operation.

The newly implemented features include real-time logging of internal program values, an interface for closed-loop feedback of equipment parameters, an operating mode for tracing user-defined sinusoidal movements, and an operating mode for reading and executing individual trajectory files. In addition, the user interface has been redesigned to make it clearer and more intuitive to use.

All the above functions have been implemented, but with some limitations on their originally planned features. As a result of the revision, the CPU load of the embedded controller was improved from approx. 99% utilization to approx. 60% in operation. Measurements suggest a facility precision of under 1 mm and 0,3 degree. Varying by axis, only a minimum of 3,5 mm and 8,1 degrees can be confirmed. By freeing up system resources, additional new functions can now be implemented in the embedded controller. In addition, the host PC now provides an additional platform with significantly more capacity for non-time critical tasks.

Inhaltsangabe

1 EINFÜHRUNG 1

1.1 Ziel der Arbeit 1

1.2 Vorgehen 1

2 GRUNDLAGEN UND AUSGANGSLAGE 3

2.1 RACOON und ORION 3

2.2 Steuerung der Anlagen 6

2.3 LabVIEW 7

2.4 Anforderungen 8

3 RESTRUKTURIERUNG 11

3.1 Analyse des ursprünglichen Aufbaus 11
3.1.1 RACOON 11
3.1.2 ORION 14
3.1.3 Auswertung 16

3.2 Konzeptentwicklung 17

3.3 Implementierung 20
3.3.1 cRIO 20
3.3.2 Host-PC 22

3.4 Ergebnisse 26

4 NEUE FUNKTIONEN 28

4.1 Implementierung neuer Funktionen 28
4.1.1 Positions-Rückkopplung 28
4.1.2 Logging 31
4.1.3 Trajectory Follow 36
4.1.4 Sine Sweep und Manuelle Steuerung 40

4.2 Graphical User Interface 41
4.2.1 Ausgangslage 41
4.2.2 Konzepte 42
4.2.3 Implementierung 44
4.2.4 Ergebnisse 61

5 ANLAGENPARAMETER UND BEWERTUNG DER ANLAGE 64

5.1 Step-Conversion 64

5.2 Anlagenpräzision 67

5.3 Referenzfahrt 68

6 ERGEBNISSE 69

6.1 Bewertung und Diskussion 69

6.2 Empfehlungen für weitere Entwicklung 72
6.2.1 Zeitpunkt der FPGA-Kommunikation 72
6.2.2 Motorsignale 74
6.2.3 Sonnensimulator 75

6.3 Fazit 75

A LITERATURVERZEICHNIS 77

B ANHANG 78

Abbildungsverzeichnis

Abb. 1–1: Logo des RACOON-Laboratory [3] ... 1
Abb. 2–1: RACOON-Anlage – Servicer ... 3
Abb. 2–2: RACOON-Anlage – Target ... 4
Abb. 2–3: RACOON-Anlage – Sonnen- und Erdalbedo-Simulator [3] ... 5
Abb. 2–4: Aufbau des ORION-Lab [5] ... 6
Abb. 2–5: Schema: Aufbau und Geräte der RACOON-Steuerung ... 7
Abb. 2–6: typisches LabVIEW-Blockdiagramm ... 8
Abb. 2–7: typisches Frontpanel ... 8
Abb. 3–1: ursprüngliche Programmstruktur der RACOON-Lab-Steuerung ... 11
Abb. 3–2: Programmstruktur der ORION-Lab-Steuerung ... 15
Abb. 3–3: Richtung des Signalflusses abhängig von Ereignissen ... 19
Abb. 3–4: Code zum verringerten Aufrufen der Netzwerkschnittstellen ... 21
Abb. 3–5: Control Loop In: Aufrufen der Shared Variables nur bei Bedarf ... 22
Abb. 3–6: Schema Funktion des Compare-VI ... 22
Abb. 3–7: Ausschnitt Compare.vi – Start Init ... 23
Abb. 3–8: Ausschnitt Compare.vi – Target ... 24
Abb. 3–9: Ausschnitt Compare.vi – Operating Mode ... 24
Abb. 3–10: Comm Interface schematische Struktur ... 25
Abb. 3–11: neue Programmstruktur des RACOON-Labs ... 26
Abb. 4–1: Schema Datenübertragung Optitrack ... 28
Abb. 4–2: Reflexionen als Störquelle beim Messen mit Optitrack ... 30
Abb. 4–3: Log Interface ... 33
Abb. 4–4: Logging Loop ... 34
Abb. 4–5: Transformate Logging Tool ... 35
Abb. 4–6: Trajektorien-Datei lesen ... 38
Abb. 4–7: Streaming der Trajektorien-Datei ... 39
Abb. 4–8: Lesen und Nachfahren von Trajektorien ... 39
Abb. 4–9: alte Benutzeroberfläche des RACOON-Labs ... 41
Abb. 4–10: automatischer Programmablauf auf Host-PC und cRIO ... 45
Abb. 4–11: Übersicht GUI Loop ... 45
Abb. 4–12: Setup Window ... 46
Abb. 4–13: GUI Loop Init – Variableninitialisierung und Control Loop Aufruf ... 46
Abb. 4–14: GUI Loop Init – Wait for Control Loop Setup ... 47
Abb. 4–15: GUI Loop Init – Wait for Control Loop Start ... 48
Abb. 4–16: Control Loop Setup ... 49
Abb. 4–17: Stop Button und Logik ... 49

Abb. 4–18: Control Loop Stop 50
Abb. 4–19: GUI Loop Input – Vereinheitlichen der Basic Controls 51
Abb. 4–20: GUI Output – Individualisierung der einzelnen Bedienelemente 52
Abb. 4–21: All Axes Bedienfeld 52
Abb. 4–22: All Axes Handler 53
Abb. 4–23: Axis Mode while locked in der GUI Comm Loop 53
Abb. 4–24: Der Apply Button 54
Abb. 4–25: Funktion Apply Button für Target im Compare VI 54
Abb. 4–26: Funktion Apply Button für SWG im Comm Interface 55
Abb. 4–27: Veränderung des Erscheinungsbilds des Apply Buttons 56
Abb. 4–28: Veränderung des Erscheinungsbilds des LateCount 56
Abb. 4–29: InteractiveGUI.vi 56
Abb. 4–30: Caution and Advisory Display 57
Abb. 4–31: Erstellung der Warnung Network Connection Lost 58
Abb. 4–32: Bündeln und Anzeigen aller Warnungen im CAD 58
Abb. 4–33: Erstellung des Hinweises Soft Limit reached 59
Abb. 4–34: Ergebnis Erstellung des Hinweises Soft Limit reached 60
Abb. 4–35: Erstellung des Hinweises Soft Limit reached für eine Achse 60
Abb. 4–36: Das neue Graphical User Interface 62
Abb. 5–1: Herleitung des epm-Wertes für Drehungen 66
Abb. 6–1: Übertragungszeitpunkte der FPGA-Informationen bei zwei unterschiedlichen Programmstrukturen 73
Abb. 6–2: Zeit zwischen Steps für drei Versionen der Steuerung 74

Tabellenverzeichnis

Tab. 2–1: benötige Software auf dem Host-PC 7
Tab. 2–2: Anforderungsliste 8
Tab. 3–1: Betriebsmodi des alten RACOON-Codes 12
Tab. 3–2: Achsen des RACOON-Lab 13
Tab. 3–3: Vergleich cRIO 9014 und cRIO 9024 [8], [9] 16
Tab. 3–4: Zu übertragende Daten zwischen cRIO und Host-PC 18
Tab. 4–1: Aufbau einer Logging-Datei 34
Tab. 4–2: Alle im CAD implementierten Warnungen und Hinweise 61
Tab. 5–1: Ergebnisse Bestimmung m_to_steps 65
Tab. 6–1: ausgewertete Anforderungsliste 69

Symbole und Formelzeichen

$k_{m_to_steps}$	Faktor m_to_steps [-]
s_{steps}	Strecke in Steps [-]
s_{meter}	Strecke in Meter [m]
s_{rad}	Strecke im Bogenmaß [rad]
Δx_{epm}	Error per Marker [mm]
$\Delta\varphi_{epm}$	Error per Marker, in Rotation übertragen [rad]
$\Delta s_{meter,rad}$	Standardabweichung [m], [rad]
Δs_{gesamt}	Gesamtfehler
d_{min}	Abstand Drehpunkt zu am nächsten gelegenen Marker [m]
$d_{min,Servicer}$	s.o., für Servicer
$d_{min,Target}$	s.o., für Target

Abkürzungen

RACOON	Real-Time Attitude Control and On-Orbit Navigation Laboratory
ORION	Orbital Robotic Interaction, On-orbit servicing, and Navigation Laboratory
LRT	Lehrstuhl für Raumfahrttechnik
cRIO	compactRIO
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
VI	Virtual Instrument
HMI	Human-Machine Interface
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
FIFO	First In, First Out
FTP	File Transfer Protocol
DOF	Degrees of Freedom
CSV	Comma-separated values
epm	Error per Marker

1 Einführung

Das Real-Time Attitude Control and On-Orbit Navigation Laboratory (RACOON-Lab) ist eine Simulationsumgebung für On-Orbit Servicing Missionen und Nahbereichsoperationen im Erdorbit [1]. On-Orbit Servicing ist das Reparieren, Betanken oder Aufrüsten von Satelliten oder ähnlichen Konstruktionen im Weltall [2]. Als bedeutendes Forschungsfeld in der Raumfahrt wird weltweit daran geforscht. So auch am Lehrstuhl für Raumfahrttechnik (LRT) der Technischen Universität München, an dem sich das RACOON-Lab befindet. Wie die Technologie selbst, unterliegen auch die Werkzeuge, mit denen an dieser geforscht wird, einem ständigen Wandel. So gilt es, auch diesen Simulator weiterzuentwickeln, um die Forschung an zukünftigen Technologien zu erleichtern.

Abb. 1–1: Logo des RACOON-Laboratory [3]

1.1 Ziel der Arbeit

Das Ziel dieser Arbeit ist das Erweitern des Funktionsumfangs, sowie die Verbesserung der Benutzerfreundlichkeit der Steuersoftware des RACOON-Lab. Im Zuge dessen wird ebenfalls ein Restrukturieren des bestehenden Codes erforderlich sein. Eigenschaften der Anlage und der Software, wie Präzision oder Performance, sollen dabei mindestens gleichbleiben oder verbessert werden.

Die neu zu implementierenden Funktionen basieren auf Features, die im Rahmen einer vorangegangenen Masterarbeit [4] bereits in das ORION-Lab, einer Anlage ähnlich des RACOON, implementiert wurden.

Eine grundlegend neue Benutzeroberfläche soll ein möglichst einfaches und intuitives Bedienen der Anlage gewährleisten.

1.2 Vorgehen

Zum Differenzieren der Aufgabenstellung soll zu Beginn eine Anforderungsliste erarbeitet werden. Anhand dieser wird auch das Endergebnis bewertet werden.

Der erste Arbeitsschritt ist die Analyse der bestehenden, auf embedded Controller ablaufenden Steuerprogramme der beiden Simulatoren. Darauf aufbauend erfolgt eine Restrukturierung der Software, mit dem Ziel, sie im Hinblick auf Systemressourcen effizienter zu machen.

In Kapitel 4 erfolgt dann die eigentliche Implementierung von neuem Code. Zum einen sollen dabei die neuen Funktionen an das RACOON angepasst und in den überarbeiteten Code implementiert werden. Zum anderen werden Konzepte für eine intuitivere Benutzeroberfläche entworfen und in die Steuersoftware eingearbeitet.

Zuletzt werden mithilfe des Optitrack-Systems Messungen zur Bestimmung von Anlagenparametern gemacht. Außerdem sollen Aussagen zu Referenzfahrten und Anlagenpräzision gemacht werden.

2 Grundlagen und Ausgangslage

2.1 RACOON und ORION

Das RACOON-Lab (Real-Time Attitude Control and On-Orbit Navigation Laboratory) ist eine modular aufgebaute Simulationsumgebung zur Simulation von Nahbereichsoperationen im Erdorbit an der Technischen Universität München. Die Umgebung besteht aus mehreren Soft- und Hardwaremodulen, unter anderem einer nach dem HIL-Prinzip (Hardware in the Loop) eingegliederten robotischen Anlage [3].

Diese kann sowohl als Teil der Gesamtsimulation als auch als eigenständiger Teststand, beispielsweise zum Testen von optischen Sensoren, verwendet werden.

Die Anlage ist etwa 10 m lang, 5.5 m breit und 4.5 m hoch. Sie lässt sich in 3 mechanische Subsysteme aufteilen: *Servicer*, *Target* und *Sonnen-/Albedo-Simulator*.

Der Servicer-Teil, oder auch *Chaser* genannt, besteht aus einem Schienensystem für translatorische Bewegungen sowie Drehgelenken für Rotationen. Dadurch kann sich das eigentliche Servicer-Modell, in Abb. 2–1 gelb umkreist, in alle sechs Raumrichtungen bewegen. Dieses Modell stellt die Vorderseite des zu steuernden Chaser-Satelliten dar und kann mit verschiedensten Kameras, Sensoren und Messinstrumenten für Experimente ausgestattet werden.

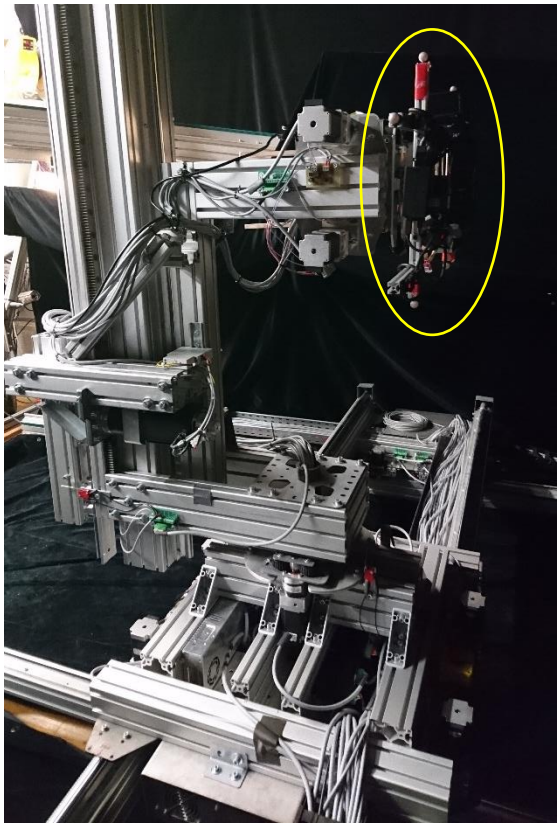


Abb. 2–1: RACOON-Anlage – Servicer

Der Target-Teil, zu sehen in Abb. 2–2, besteht aus einer um eine vertikale Achse drehbaren C-Schiene, auf deren Innenseite wiederum ein Schlitten translatorisch bewegt werden kann. Auf diesem Schlitten befindet sich die eigentliche Ziel-Satellitenattrappe.

Diese kann sich ebenfalls um eine horizontale Achse drehen, sowie auch in die beiden anderen Raumrichtungen neigen. Effektiv erlaubt diese Konfiguration der Attrappe, ohne mechanische Einschränkungen endlos in alle drei Raumrichtungen zu rotieren, um etwa außer Kontrolle geratene, taumelnde Ziele zu simulieren.

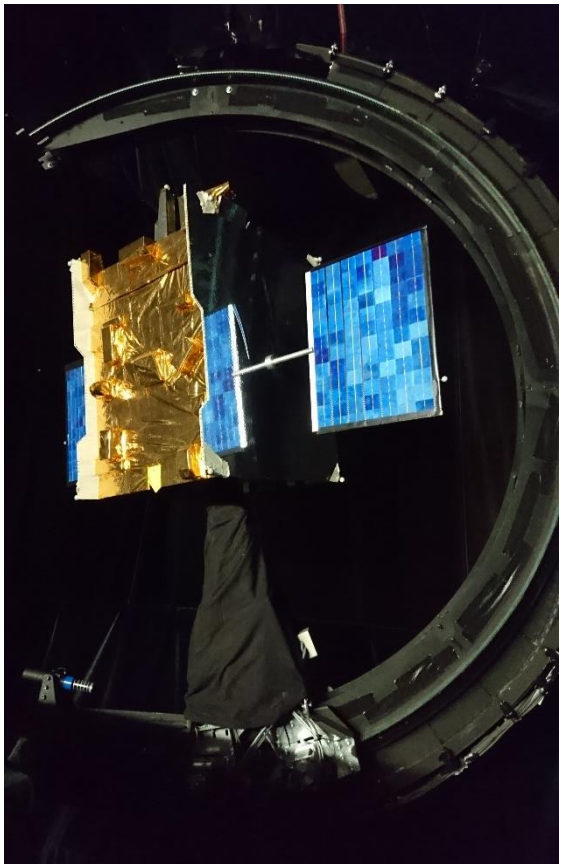


Abb. 2–2: RACOON-Anlage – Target

Die Sonnen- und Albedo-Simulatoren bestehen jeweils aus einer Lampe, die Licht, ähnlich dem des zu simulierenden Spektrums, ausstrahlt. Zu sehen sind beide in Abb. 2–3. Die Lampen sind jeweils horizontal, von Hand verstellbar gelagert. Diese gesamte Elektronik eines Simulators befindet sich auf einem um die Gierachse drehbaren Teller, der wiederum auf einem Schlitten angebracht ist. Dieser Schlitten kann sich linear entlang eines Schienensystems bewegen, das einmal um die ganze Anlage herum verläuft. Aktuell ist nur der Sonnensimulator steuerbar.

Abb. 2–3: RACOON-Anlage – Sonnen- und Erdalbedo-Simulator [3]

Zur Messung von einzelnen Anlagenteilen können reflektierende Marker an Komponenten angebracht werden. Diese werden von einem Infrarot-Kamerasystem von der Firma Natural Point, genannt Optitrack, gefilmt und aus den erhaltenen Bildern Orts- und Lageinformationen berechnet. Das RACOON-Lab hat dafür acht solcher Infrarotkameras verbaut.

Das Orbital Robotic Interaction, On-orbit Servicing, and Navigation Laboratory (ORION-Lab) am Florida Institute of Technology ist ebenfalls eine Testumgebung zur Simulation von Operationen im Weltall [5]. Auch das ORION-Lab besteht aus einem Servicer, einem Target und einem stationären Sonnensimulator.

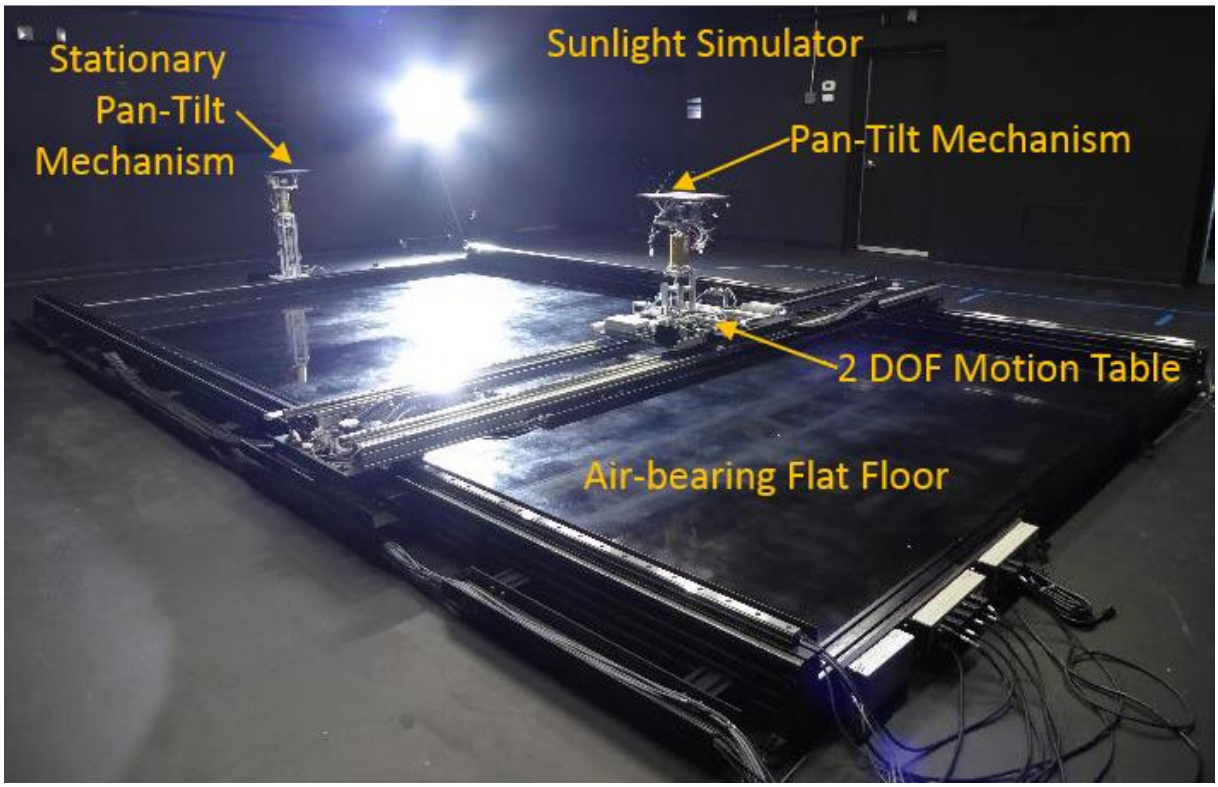


Abb. 2–4: Aufbau des ORION-Lab [5]

Der Servicer kann sich hier zweidimensional auf einer Ebene bewegen und sich zusätzlich in zwei Raumrichtungen neigen. Das Target ist stationär und kann ebenfalls in zwei Richtungen gekippt und geneigt werden. Auch das ORION-Lab hat ein Optitrack-System verbaut. Hier kommen sogar 12 Kameras zum Einsatz.

Angetrieben werden die verschiedenen Achsen in beiden Anlagen von elektronischen Schrittmotoren, welche ihre Signale direkt von einer Steuersoftware erhalten. Das ORION verfügt insgesamt über 6 Achsen, das RACOON über 13 Achsen, die von einer Software kontrolliert werden müssen.

2.2 Steuerung der Anlagen

Für beide Anlagen wird diese Steuersoftware über zwei Orte verteilt ausgeführt: Am Host-Computer und auf dem compactRIO (cRIO), ein echtzeitfähiger embedded Controller der Firma National Instruments. Die Steuersoftware läuft auf beiden Geräten innerhalb der LabVIEW-Umgebung, ebenfalls von National Instruments.

Auf dem Host-Computer erfolgt die Eingabe von Befehlen einer bedienenden Person mittels einer grafischen Benutzeroberfläche. Auf dieser werden auch Warnungen und Informationen zur Anlage angezeigt. Das verwendete Betriebssystem beim RACOON ist Windows 10 Education. Um die Anlage über den Host-PC steuern zu können, müssen die in Tab. 2–1 angegebenen Softwarepakete installiert sein.

Tab. 2–1: benötige Software auf dem Host-PC

NI LabVIEW Version 17.0 (32 Bit)
FPGA Module
MathScript RT Module
Real-Time Module
NI CompactRIO Device Driver

Mittels Ethernet-Kabel erfolgt die Übertragung von Informationen zwischen PC und dem cRIO. Das cRIO besteht aus einem Controller (Modellnummer cRIO 9014 für das RACOON), einer logischen Schaltung, genauer gesagt einem *Field Programmable Gate Array* (FPGA), und E/A-Modulen (Chassis-Modellnummer: cRIO 9104 für das RACOON). Der Großteil des in dieser Arbeit beschriebenen Steuercodes läuft im Controller ab. Ziel der Berechnungen in diesem sind im Wesentlichen Bewegungs-Spline-Informationen, die dann an das FPGA übergeben werden. Dieses verarbeitet die Splinedaten dann in Signale für die Motoren. Über das FPGA erhält der Controller Informationen zu Schaltern der Anlage und zum Zustand des FPGAs selbst.

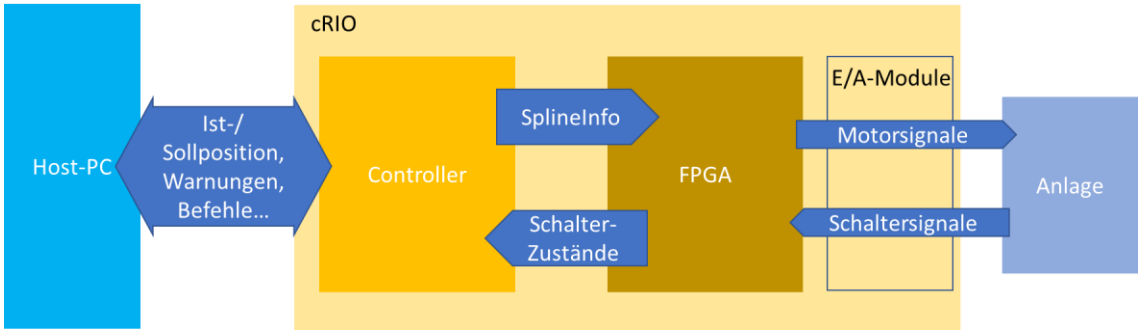


Abb. 2–5: Schema: Aufbau und Geräte der RACOON-Steuerung

Der Teil der Steuersoftware auf dem Mikrocontroller des cRIO verarbeitet die Steuereingaben des Graphical User Interface (GUI) und errechnet aus diesem 3 Vektoren pro Achse, SplineInfo genannt, mithilfe derer das FPGA ein Bewegungsspline und daraus schließlich Steuersignale für die Schrittmotoren berechnet.

Als internes Längenmaß im Programm werden sogenannte *Steps* verwendet. Ein Step ist dabei ein Schritt von einem der Schrittmotoren, die jede Achse antreiben. Jede Achse verfügt über einen anderen Umrechnungsfaktor zwischen Steps und Meter, sprich Steps sind für unterschiedliche Achsen nicht unbedingt gleich. Dieser Umrechnungsfaktor ist für jede Achse im Wert *m_to_steps* festgelegt.

2.3 LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench), von National Instruments Corporation entwickelt, ist sowohl eine Entwicklungsumgebung als auch eine

grafische Programmiersprache. Sie verwendet Funktionsblöcke, die durch sogenannte Drähte miteinander verbunden sind. Entlang dieser Drähte, die verschiedenste Datentypen führen können, werden die Blöcke nach dem Datenflussprinzip aufgerufen [6].

Alle diese Blöcke sind immer innerhalb eines Programms, im LabVIEW-Jargon Virtual Instrument (VI) genannt, angeordnet, wie etwa in Abb. 2–6.

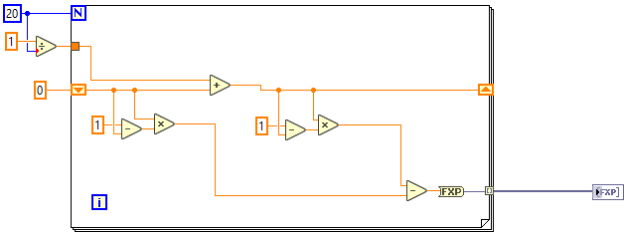


Abb. 2–6: typisches LabVIEW-Blockdiagramm

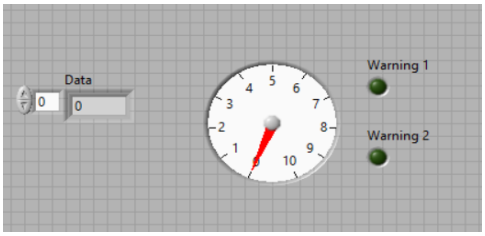


Abb. 2–7: typisches Frontpanel

Ein VI kann dabei auch andere VIs enthalten, welche dann SubVIs genannt werden. Jedem Standard-LabVIEW-Programm ist ein sogenanntes Frontpanel zugeordnet, ein Beispiel ist in Abb. 2–7 zu sehen. Von auf diesem angeordneten Bedienelementen können Werte für Berechnungen innerhalb eines VIs ausgelesen und das Ergebnis wieder in diese hineingeschrieben werden. Gleichzeitig fungiert es als Benutzeroberfläche zur Steuerung der im VI verwendeten Daten.

2.4 Anforderungen

Zum Konkretisieren der zu erreichenden Ziele wurde mit Tab. 2–2 eine Anforderungsliste erstellt, die alle wesentlichen Funktionen und Eigenschaften auflistet, die das fertige Steuerungsprogramm aufweisen soll. Sie soll helfen, passende Lösungen zu finden, Over-Engineering oder Mangel zu identifizieren, und ermöglicht während des Arbeitens die Kontrolle des Zielfortschritts. Der Erfolg der Arbeit soll am Ende anhand dieser gemessen werden.

Die erstellte Anforderungsliste basiert zum Teil auch auf Erkenntnissen, die aus einer vorangegangenen Masterarbeit am LRT gewonnen wurden.

Eine in der Priorität-Spalte mit einem Plus (+) markierte Anforderung sollte erfüllt werden. Zwei Plus (++) markieren eine Anforderung, die für die Erfüllung der Aufgabenstellung zwingend erforderlich sind. Mit drei Plus (+++) markierte Anforderungen besitzen die höchste Priorität und stellen das Mindestmaß für einen sinnvoll einsetzbaren Steuercode dar.

Tab. 2–2: Anforderungsliste

ID	Beschreibung	Verifikation	Priorität
	Allgemein		
1	Die Anlage soll mittels eines Steuerungscodes geschrieben mit dem Programm NI LabVIEW Version 17.0 (32-bit) gesteuert werden	Inspizierung	++
2	Der Steuerungscode soll auf dem bereits verbauten NI cRIO 9014 ausführbar sein	Inspizierung	+++
3	Es soll möglichst viel Code aus dem vorhandenen Steuerungscode der ORION-Anlage übernommen werden	Inspizierung	+

4	Der Steuerungscode soll alle 13 Achsen der Anlage steuern können: • Translation Servicer in X-, Y- und Z-Richtung • Rotation Servicer in X-, Y- und Z-Richtung • Translation Target auf C-Schiene • Rotation C-Schiene des Targets • Rotation Target in X-, Y- und Z-Richtung • Translation Sonnensimulator • Rotation Sonnensimulator	Demonstration	+++
5	Der Steuerungscode soll folgende Betriebsmodi (Operating Modes) zulassen: • Direkte Steuerung am PC vor Ort • Steuerung via internes LRT-Netzwerk • Individuell generierbare sinusförmige Bewegungen • Steuerung via Trajektorien-Datei (CSV-Datei)	Demonstration	++
6	Der Steuerungscode soll ein Überschreiten von Endschaltern und individuell eingestellten Limits erkennen und die entsprechende Achse stoppen	Demonstration	+++
7	Der Steuerungscode soll aktuelle Befehle an das FPGA jederzeit mit einer Frequenz von 100 Hz senden	Messung mittels Get Timestamp.vi	+++
	Positions- und Lagebestimmung		
8	Der Steuerungscode soll zwei verschiedene Methoden zur Orts- und Lagebestimmung verwenden können (außer für Sonnen-/Albedo-Simulator): • Open Loop: Bestimmung mittels vom FPGA gezählter „Motor-Steps“ • Closed Loop: zusätzliche Orts- und Lagebestimmung mittels Infrarotkameras	Demonstration	++
9	Die Daten für den Closed-Loop-Betrieb sollen mittels der vorhandenen Hardware erhoben werden • 8 Kameras • Marker • Computer zur Auswertung • Vorhandene Netzwerkumgebung	Inspizierung	++
10	Die Daten für den Closed-Loop-Betrieb sollen mittels der vorhandenen Software und nur minimal angepassten Codes ausgewertet werden • Motive V2.0 von Optitrack • Export-Code in C++ der ORION-Anlage • LabVIEW-Code der ORION-Anlage	Inspizierung	++
11	Verbesserung der Positioniergenauigkeit der Anlage zu Werten ähnlich der ORION-Anlage (max. Abweichung von > 1 mm bzw. > 1 Grad) im Closed-Loop-Betrieb	Abmessung mittels Optitrack	+
	Datenlogging		
12	Es soll die Möglichkeit des Daten-Logging innerhalb des LabVIEW-Programms geben	Demonstration	++
13	Es soll das Logging von Position, Geschwindigkeit und Beschleunigung laut LabVIEW-Code sowie die Position laut Optitrack-Software sowie die SplineInfo jeder Achse möglich sein	Inspizierung	+

14	Das Logging soll in Echtzeit und sollte mit einer Taktzahl von 10 ms geschehen	Kontrolle der Log-Dateien	+
	GUI		
15	Das GUI sollte ohne Scrollen auf den Bildschirm des Host-PC passen	Inspizierung	+
16	Es sollte das Steuern und Überwachen aller 13 Achsen gleichzeitig möglich sein	Inspizierung	+
17	Es sollen mindestens folgende Dinge ablesbar sein: • Momentane Position jeder Achse • Momentane Geschwindigkeit jeder Achse • Momentane Beschleunigung jeder Achse • Fehler, Hinweise, ausgelöste Switches	Inspizierung	+++
18	Es sollen mindestens folgende Dinge steuerbar sein: • Zielposition jeder Achse • Maximale Geschwindigkeit und maximale Beschleunigung jeder Achse • Betriebsmodus • Positions- und Lagebestimmung • Logging an/aus • Amplitude und Frequenz einer Sinus-Trajektorie • Individuelle Limits für jede Achse einstellbar • Lock/Unlock jeder Achse • Dateipfad eines Trajectory Input File	Inspizierung	++
19	Weniger wichtige Anzeige- und Steuerelemente sollen nicht bei den Basic Controls auftauchen	Inspizierung	+
20	Fehler, Hinweise und Warnungen sollen gebündelt in einem „Caution and Advisory Display“ angezeigt werden	Inspizierung	+
21	Servicer-, Target- und Lampen-Achsen sollen mittels Farbgebung geclustert werden	Inspizierung	+
22	Distanz soll in Meter, Rotation in Grad und Zeit in Sekunden angegeben werden	Inspizierung	+

Zum Erreichen der Ziele dieser Arbeit soll der Steuercode auf dem Controller und dem Host-PC überarbeitet werden. Code auf dem FPGA wird nicht verändert. Auch beschränkt sich diese Arbeit auf Änderungen an der Software, physische Teile der Anlage sollen ebenfalls nicht verändert werden.

3 Restrukturierung

3.1 Analyse des ursprünglichen Aufbaus

3.1.1 RACOON

Der Steuercode des RACOON-Labs wird innerhalb einer zeitgesteuerten Schleife, der *Control Loop,* alle 10 ms ausgeführt. Zu Beginn jeder Iteration werden Daten aus den Bedienelementen eines Frontpanel gelesen. Entsprechend werden am Ende eine Iteration alle neu erzeugten Daten wieder in diese Bedienelemente geschrieben.

Das Frontpanel wird auf dem Host-PC angezeigt. Entsprechend werden am Ende einer Iteration neu berechnete Daten wieder in diese Kontrollelemente geschrieben. Genau genommen existieren keine Frontpanels auf Real Time Targets wie dem cRIO. Es wird lediglich automatisch das Abbild eines solchen von LabVIEW zu Debug-Zwecken auf dem Host-PC erstellt [7], vorausgesetzt die LabVIEW-Entwicklungsumgebung ist auf diesem installiert.

Die Struktur des ursprünglichen Steuerprogramms ist in Abb. 3–1 dargestellt. Die eben beschriebene Datenübertragung zwischen cRIO und Host-PC ist im Diagramm mit zwei parallelen schwarzen Strichen dargestellt. Der Code lässt sich in mehrere Abschnitte, beziehungsweise Unterprogramme aufteilen.

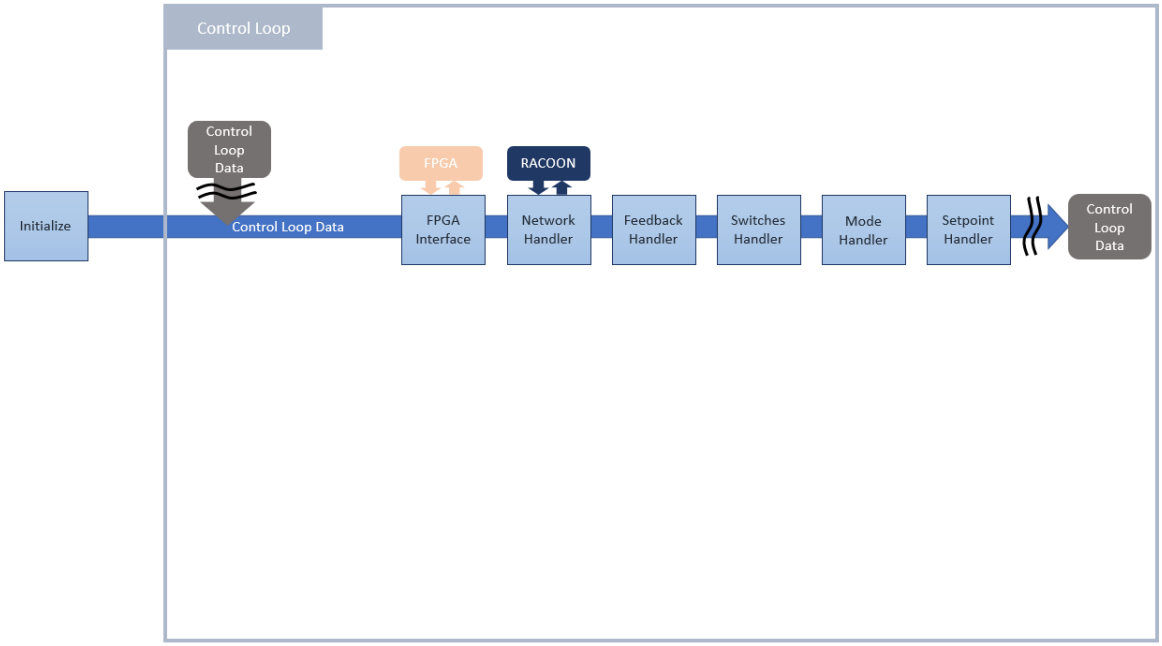


Abb. 3–1: ursprüngliche Programmstruktur der RACOON-Lab-Steuerung

Beim Starten der Steuersoftware werden Startwerte und Konfigurationswerte geladen und berechnet, sowie die Netzwerkverbindung und die Verbindung zum FPGA hergestellt.

Das *FPGA-Interface* erledigt den Großteil der Kommunikation zwischen Controller und FPGA. Es werden die Informationen zum Bewegungsspline an das FPGA übergeben, im Gegenzug erhält der Controller Informationen zu Status von Switches und Daten,

die der Synchronisation beider Geräte dienen. Außerdem findet hier die Synchronisation zwischen Controller und FPGA statt. Für das Generieren von stetigen Motorsignalen ist es von großer Wichtigkeit, dass dieses Sub-Vi regelmäßig mit gleich bleibender Periode aufgerufen wird.

Im *Network Handler* werden Informationen aus der restlichen RACOON-Simulation, in erster Linie Steuerbefehle, mittels User Datagram Protocol (UDP) empfangen und in verwertbare Datentypen gewandelt. Außerdem werden hier Informationen zum Zustand des HIL-Teils gesammelt und ebenfalls per UDP an die RACOON-Simulation zurückgesandt.

Im *Feedback Handler* werden kleine Schrittkorrekturen, die im FPGA berechnet wurden, mit der momentanen Position verrechnet.

Der *End Switch Handler* sorgt dafür, dass die Anlage stoppt, wenn ein Endschalter ausgelöst wird, indem der Betriebsmodus auf *Stopping* geschaltet wird.

Im *Mode Handler* werden, falls nicht schon vorher geschehen, Steuerbefehle und Signale zur Erzeugung genauer Positions- und Geschwindigkeitswerte verwendet. Je nach achsenweise eingestelltem Betriebsmodus, im Programm *Operating Mode* genannt, werden diese Werte anders berechnet. Alle aktuell im RACOON implementierten Betriebsmodi sind in Tabelle Tab. 3–1 dargestellt.

Tab. 3–1: Betriebsmodi des alten RACOON-Codes

Network	Erlaubt Steuerung der Anlage über eine UDP-Verbindung. Wird verwendet, wenn Anlage zusammen mit der restlichen Simulationsumgebung verwendet wird
RefSearch	Automatisch ablaufende Referenzfahrt, um Achsen in fest definierte Positionen zu bringen
Stopping	Achsen werden gestoppt
Docked	Nicht verwendet Erlaubt manuelle Eingaben von Zielwerten für Bewegung
PathDemo	Nicht verwendet

Im *Setpoint Handler* werden diese Werte schließlich zu Bewegungsspline-Informationen verarbeitet, die dann in der darauffolgenden Iteration an das FPGA übergeben werden.

Die wichtigsten internen Werte innerhalb der Control Loop sind in Abb. 3–1 als die *Control Loop Data* bezeichnet. Sie sind in drei verschiedenen Datensträngen, sogenannten *Cluster,* zusammengefasst. *RealTime* enthält alle Informationen, die für die Kommunikation und die Synchronisation mit dem FPGA verantwortlich sind. In *Network* sind alle wichtigen Daten zur Kommunikation zwischen cRIO und anderen Geräten via Ethernet-Kabel festgehalten. *AxisData* sind genau genommen mehrere Cluster, die in einem sogenannten *Array,* einem Vektor oder eine Matrix mit Elementen gleichen Datentyps, zusammengefasst sind. Welcher Index oder Achsennummer dabei zu welcher Achse gehört, ist in Tab. 3–2 zu sehen.

Tab. 3–2: Achsen des RACOON-Lab

0	Servicer Translation X
1	Servicer Translation Y
2	Servicer Rotation Z
3	Servicer Translation Z
4	Servicer Rotation Y
5	Servicer Rotation X
6	Target C-Schiene Rotation Z
7	Target Translation entlang C-Schiene (entspricht einer Rotation des Targets entlang der Y-Achse der C-Schiene)
8	Target Rotation X
9	Target Rotation Y
10	Target Rotation Z
11	Sonnensimulator Translation entlang Kreisbahn
12	Sonnensimulator Rotation Z

Bei der Benutzung von dieser ursprünglichen Steuersoftware fallen schnell die teils mehrere Sekunden langen Verzögerungen beim Bedienen und Stocken des GUI bis hin zu einem Nichterkennen von Eingaben auf. Der Grund dafür ist die Auslastung des Prozessors des Controllers, welche selbst im Stillstand bei rund 99 % liegt. Gemessen wurde sowohl mit dem von National Instruments bereitgestellten *Distributed Systems Manager* als auch mit dem in LabVIEW integrierten VI *RT Get CPU Loads*.

Die Control Loop selbst scheint davon jedoch nicht allzu stark beeinflusst zu werden. Mittels des integrierten VI *RT Get Timestamp* wurde die Periodendauer im Stillstand der Anlage über 3000 Perioden gemessen. Dabei konnten keine signifikanten Abweichungen von der vorgesehenen Periodendauer festgestellt werden. Ob der voll ausgelastete Prozessor dieses Messergebnis beeinflusst hat, lässt sich nicht völlig ausschließen.

Eine Begründung für die scheinbar funktionierende Control Loop, eine aber höchstens mangelhafte Bedienbarkeit könnte sein, dass die Control Loop durch präemptives Scheduling trotz hoher Prozessorauslastung immer noch pünktlich und ungestört ablaufen kann. Die restliche Auslastung resultiert womöglich beispielweise aus den Prozessen, die das Übertragen der Frontpanel-Daten von cRIO zum Host-PC und zurück steuern. Durch mangelnde Systemressourcen eingeschränkt, können diese Prozesse nicht mehr ungehindert ablaufen, und es kommt zu den beschriebenen Problemen bei Benutzung des GUI.

Ob dies so wirklich der Fall ist, lässt sich jedoch leider nicht mit Sicherheit sagen. Weder ist ein Blick in den Code dieser Prozesse möglich, noch konnte eine detaillierte Beschreibung von National Instruments gefunden werden.

3.1.2 ORION

Der ORION-Code wurde auf dem RACOON-Code aufbauend entwickelt. Große Teile des Codes sind ähnlich oder identisch. Es wurden jedoch einige zusätzliche Funktionalitäten hinzugefügt [4]. Programmteile, die wesentlich geändert wurden, sind in Abb. 3–2 farblich hervorgehoben.

- *Closed Loop Feedback* (Positions-Rückkopplung): Ziel ist es, die Anlagenpräzision zu erhöhen. Mittels des Optitrack-Systems werden Orts- und Lageinformationen von Chaser und Target ermittelt. Diese werden in den Steuercode der RACOON-Anlage gesendet, mit den programminternen Werten verglichen und wenn nötig Korrekturen eingeleitet. Diese neue Funktion ist im Feedback Handler untergebracht.

- *Sine Sweep:* Ein neuer Betriebsmodus, der benutzerdefinierte, sinusförmige Bewegungen ermöglicht. Der Benutzer kann dabei Amplitude und Frequenz festlegen.

- *Trajectory Follow:* Ebenfalls ein neuer Betriebsmodus. Dieser erlaubt es, Trajektorien-Dateien nachzufahren. Die als Comma-separated values (CSV) gespeicherten Dateien erhalten dabei für jede Iteration einzeln die gewünschten Achsenpositionen.

- *Logging:* Eine aktivierbare Funktion, bei der programminterne Werte, wie etwa Achsenposition, -geschwindigkeit und -beschleunigung in jeder Schleifeniteration ausgelesen und in eine Textdatei gespeichert werden.

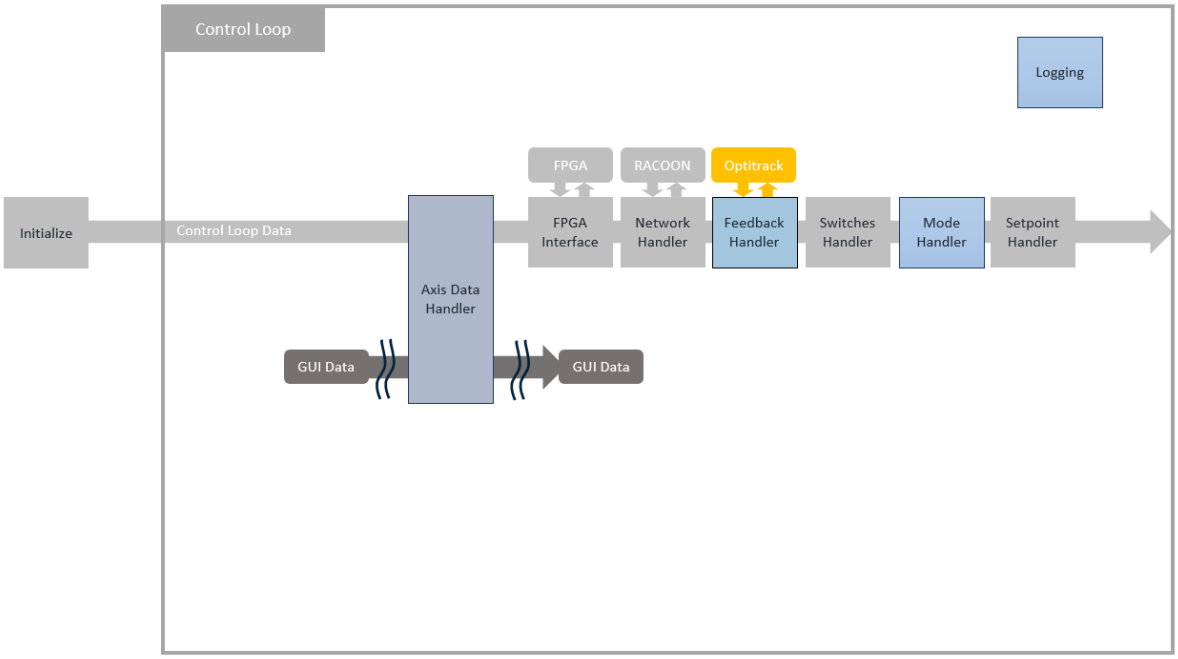


Abb. 3–2: Programmstruktur der ORION-Lab-Steuerung

Ein weiterer wesentlicher Unterschied zum alten RACOON-Code ist die Trennung von Steuer- sowie Anzeigewerten und programminternen Werten, die der Benutzer nicht sehen muss. Zum Zusammenführen der beiden Informationsflüsse wurde das Sub-VI *Axis Data Handler* eingeführt. Für den Großteil der internen Werte werden nun sogenannte Schieberegister verwendet, die Werte von einer Iteration auf die darauffolgende übergeben. Dadurch entfallen für einen Großteil der in der Control Loop verwendeten Werte das Lesen und das Schreiben in Bedienelemente, einschließlich des Übertragens zum Host-PC. So sinkt die Auslastung des RACOON-Prozessors im Stillstand der Anlage auf ~70%, wenn eine angepasste Version des ORION-Codes mit dreizehn Achsen verwendet wird. Sobald jedoch die Anlage bewegt wird oder etwa einer der neuen Funktionen verwendet wird, steigt die Auslastung stark an, bzw. stürzt das Programm ab. Zusätzlich kommt es teils zu Überschreitungen der maximal erlaubten Periodenzeit.

Laut der vorangegangenen Arbeit, in der am ORION die nun neu zu implementierenden Funktionen entwickelt wurden, funktioniert der Steuerungscode allerdings fast ohne Probleme. Genaue Angaben zu Performance gibt es jedoch leider nicht.

Eine mögliche Erklärung dafür, dass ein fast identischer Code auf den beiden Anlagen so unterschiedlich gut arbeitet, ist die Anzahl der Achsen. Der Großteil der zu erledigenden Rechenleistung steht im direkten Zusammenhang zu der Achsenanzahl. Dreizehn Achsen im RACOON-Lab ergeben somit in etwa den doppelten Aufwand im Vergleich zu sechs Achsen im ORION-Lab. Wird die Achsenanzahl virtuell beim RACOON verringert, so sinkt die Prozessorauslastung auch deutlich ab.

Eine andere Erklärung kann sein, dass im RACOON-Lab schlichtweg weniger Systemressourcen zur Verfügung stehen. Das RACOON-Lab hat mit dem CRIO 9014 einen Controller mit höchstwahrscheinlich schlechterer Leistung verbaut hat, siehe Tab. 3–3.

Tab. 3–3: Vergleich cRIO 9014 und cRIO 9024 [8], [9]

Modell	NI cRIO-9014	NI cRIO-9024
DRAM	128 MB	512 MB
Taktrate CPU	400 MHz	800 MHz
Persistenter Speicher	2 GB	4 GB

Der cRIO 9024 Controller des ORION ist eine neuere Version mit besseren Leistungsmerkmalen. Zu beachten ist jedoch, dass eine endgültige Aussage über die Leistung nicht möglich ist, da die angegebenen Merkmale für eine eindeutige Bewertung nicht ausreichen und entsprechende Benchmark-Tests nicht vorliegen.

3.1.3 Auswertung

Als Fazit aus diesen beiden kurzen Analysen lässt sich festhalten, dass sich keiner der beiden Codes in ihren derzeitigen Zuständen als Grundlage zur Weiterentwicklung und zur Implementierung neuer Funktionen eignet. Davor müssen zuerst grundlegende Probleme beseitigt werden und ein stabil laufender Code mit genug freien Systemressourcen für neue Funktionen geschaffen werden.

Das Hauptproblem beider Codes ist eine fehlende Differenzierung zwischen *zeitkritischen* und *nicht-zeitkritischen* Operationen. *Zeitkritisch* sind dabei diejenigen Prozesse, die direkt für das Berechnen und Übertragen der SplineInfo verantwortlich sind, oder bei Zwischenfällen die Anlage durch die Änderung des Betriebsmodus stoppen. National Instruments bezeichnet in seiner Hilfe zum LabVIEW Real Time Module die Task Separation als einen Kernaspekt der meisten Echtzeitanwendungen [10].

Alle Operationen innerhalb der Programme werden mit der gleichen Häufigkeit und der gleichen Priorität aufgerufen. Eine Ausnahme ist vermutlich das automatische Übertragen und Anzeigen der Frontpanel-Werte auf einem Host-PC. Diesem Prozess ordnet LabVIEW automatisch scheinbar eine niedrigere Priorität zu.

Im Idealfall sollten Prozesse wie das Erstellen und Übergeben der SplineInfo und das Verarbeiten von Endschalter-Signalen deterministische Prozesse sein. Da das Betriebssystem auf dem cRIO mit dem Namen VxWorks echtzeitfähig ist, eine Unterteilung des Codes in verschiedene Einzelteile möglich ist, und sich auch die Kommunikation zwischen diesen Einzelteilen deterministisch realisieren lässt, ist rein theoretisch für viele Prozesse Determinismus erreichbar.

Allerdings sind zur Erfüllung mancher Vorgaben der Anforderungsliste zwingend Operationen wie Festplattenzugriff nötig, die immer nicht-deterministisch sind [11].

Da der Controller lediglich einen Prozessorkern zur Verfügung hat, kann also kein vollständig deterministischer Code geschrieben werden. Trotzdem lohnt es sich natürlich, für möglichst viele Programmteile Determinismus anzustreben, um eventuell auftretende Schwankungen in Zeitbedarf und Gerät-Ressourcen wie etwa CPU-Rechenkapazität zu minimieren.

Als erster Arbeitsschritt gilt es nun, dass Grundgerüst des bestehenden Codes grundlegend umzubauen, um Systemressourcen des Controllers nicht zu überlasten und einen stabilen Programmablauf zu gewährleisten.

3.2 Konzeptentwicklung

Als Lösungsmöglichkeit wurden drei grundlegende Konzepte erarbeitet, die auch gleichzeitig angewendet werden können. Sie werden nun im Folgenden vorgestellt werden:

Selektive Ansteuerung von Achsen

Wie bereits erwähnt, korrelieren Bedarf an Systemressourcen und Achsenzahl für den Großteil des Steuercodes. Dieser Ansatz besteht im Grunde daraus, nicht genutzte Achsen ganz aus den nötigen Berechnungen herauszunehmen.

Dieser Ansatz bietet sich an, da nur in seltenen Fällen alle Achsen gleichzeitig bewegt werden. Meist sind sogar nur einzelne Achsen in Bewegung. Berechnungen sind nur während einer Bewegung erforderlich, im Stillstand kann also an Systemressourcen gespart werden. Ein Nachteil jedoch liegt in dem zusätzlichen Overhead, der durch das selektive Aufrufen nötig wird, auch wenn dieser relativ gesehen eher gering ausfällt. Das Hauptargument gegen diese Lösung ist wohl die Einschränkung der Fähigkeiten der Anlage. Da laut Zielvorgaben alle Fähigkeiten und Eigenschaften der Anlage mindestens gleichbleiben oder verbessert werden sollen, sollte die selektive Achsansteuerung lediglich als Notlösung gesehen werden. Als solche ist sie jedoch sehr effektiv beim Einsparen von etwa CPU-Leistung und Rechenzeit.

Diese Lösung löst auch nicht die grundlegenden Probleme der bestehenden Codes, sondern schwächt lediglich deren Symptome ab.

Verringertes und/oder selektives Aufrufen von Teilprozessen

Die Grundidee hinter diesem Ansatz ist die, Teilprozesse in verschiedene Kategorien bezüglich ihrer nötigen Aufruffrequenz einzuteilen. So müssen selbstverständlich bei jeder Iteration aktuelle SplineInfo-Vektoren an das FPGA übergeben werden, damit die Anlage ohne Stocken fahren kann. Ein Update der Informationen der GUI mit einer Frequenz von 100 Hz ist jedoch verschwendete Rechenleistung, da neue Informationen überhaupt nicht so schnell von einem Bildschirm angezeigt werden können, geschweige denn sinnvoll von einem Menschen verarbeitet werden können.

Für diese Idee spricht, dass sie recht einfach zu implementieren ist, die Funktionalität der Anlage nicht einschränkt und die CPU-Last verringert. Allerdings kann es weiterhin passieren, dass Iterationen durch einzelne Prozesse länger als angedacht dauern, und so kritische Prozesse nicht rechtzeitig ausgeführt werden.

Separate Schleifen

Dieser Ansatz baut auf dem vorherigen auf und verschiebt nicht-kritische Teilprozesse in separate Schleifen, welche mit einer niedrigeren Frequenz ausgeführt werden. Gleichzeitig erhalten nicht-kritische Schleifen eine niedrigere Priorität, sodass immer

die kritische Schleife Vorrang hat. Im Prinzip wird so die von National Instruments empfohlene Task Separation [10] verwirklicht. Solange nun die Ausführdauer kritischer und nicht-kritischer Prozesse innerhalb einer Periode des kritischen Prozesses deren Dauer nicht übersteigt, wird die kritische Schleife immer ohne Verzögerung ausgeführt werden. Ein zusätzlicher Vorteil dieser Unterteilung des Codes liegt darin, dass Programmteile auch auf andere Geräte ausgelagert werden können, um so zusätzliche Systemressourcen einzusparen. Nachteile hingegen sind die zusätzlich erforderliche Kommunikation zwischen den verschiedenen Schleifen und der generell hohe Aufwand bei der Implementierung.

Da das Konzept der separaten Schleifen den größtmöglichen Nutzen verspricht, soll der Fokus bei der Restrukturierung der Steuersoftware vor allem auf diesem liegen.

Der erste Schritt soll das Herauslösen der Benutzeroberfläche aus der Control Loop sein. Es wird eine separate *GUI Loop* eingerichtet, die sämtliche Eingaben in die Benutzeroberfläche aufbereitet und nur die wesentlichen Signale an die Control Loop überträgt. Da ein Benutzer kaum Unterschiede zwischen einer Updaterate von 100 Hz und 10 Hz merkt, kann die Ausführfrequenz dieser neuen Schleife deutlich reduziert werden und somit Systemressourcen eingespart werden. Allerdings müssen für die Datenübertragung zwischen beiden Schleifen zusätzlicher Aufwand betrieben und somit Systemressourcen beansprucht werden. Bei einem Ausführen der GUI Loop auf dem cRIO müssen außerdem weiterhin sämtliche Daten der Steuerelemente an den Host-PC übertragen werden, sprich es findet gleich zweimal eine Datenübertragung statt.

Um also wirklich großen Nutzen aus diesem Konzept zu ziehen, soll die gesamte GUI Loop auf den Host-PC übertragen werden. Dadurch fällt die gesamte Belastung durch die Benutzeroberfläche weg, lediglich die Datenübertragung macht noch Aufwand. Diese kann aber jetzt genau an die Gegebenheiten angepasst werden, und ist nun nicht mehr länger eine Funktion mit unbekannten Eigenschaften.

Um Datenverkehr im Netzwerk und Nutzung von Systemressourcen des cRIO gering zu halten, sollen Netzwerk-Operationen nur so häufig wie nötig ausgeführt werden.

Von der Anforderungsliste lässt sich ablesen, welche Daten übertragen werden müssen, um die Anlage bedienen zu können. In Tab. 3–4 sind die zu übertragenden Werte, vom cRIO aus betrachtet, in zu sendende Daten und zu empfangende Daten unterteilt.

Tab. 3–4: Zu übertragende Daten zwischen cRIO und Host-PC

Senden	**Empfangen**
Current-Vektor (Position, Geschw., Beschl.)	
Target-Vektor (Position, Geschw., Beschl.)	Target-Vektor (Position, Geschw., Beschl.)
Operating Mode	Operating Mode
Network Warnings (RACOON-Network und Optitrack)	Tracking Mode

Switch-Status (Reference und End Switches)	Sine Sweep Daten (Amplitude und Frequenz)
LateCount (der Control Loop)	Endless Rotation
	Soft Limits
	Max V, Max A
	Min. Optitrack Deviation
	Max. Correction

Beim Großteil der zu übertragenden Daten ist lediglich ein Senden an das jeweils andere Gerät nötig. Eine Ausnahme bilden hier jedoch der Target-Vektor und der Operating Mode, in der Tabelle grau markiert: Sowohl Host-PC als auch cRIO senden hier einen Wert an das andere Gerät. Es muss also festgelegt werden, welcher Wert für beide Geräte gilt. Hier sollen immer die Signale des cRIO verwendet werden, so lange kein bestimmtes Ereignis vorliegt. So wird sichergestellt, dass am Host-PC immer die Werte angezeigt werden, mit denen das cRIO rechnet.

Sobald eine Eingabe des Benutzers am Host-PC erfolgt, sollen stattdessen die eingegebenen Werte des Host-PCs verwendet werden, um ein Steuern des cRIO zu ermöglichen.

Wenn allerdings im cRIO ein Alarm, beispielsweise ein End Switch, ausgelöst wird, werden die Host-PC-Eingaben wiederum vom cRIO überschrieben. So kann man sichergehen, dass sicherheitskritische Operationen, wie etwa das Stoppen der Anlage, beim Erreichen eines Endschalters nicht von Benutzereingaben blockiert werden. Diese Priorisierung ist in Abb. 3–3 schematisch dargestellt.

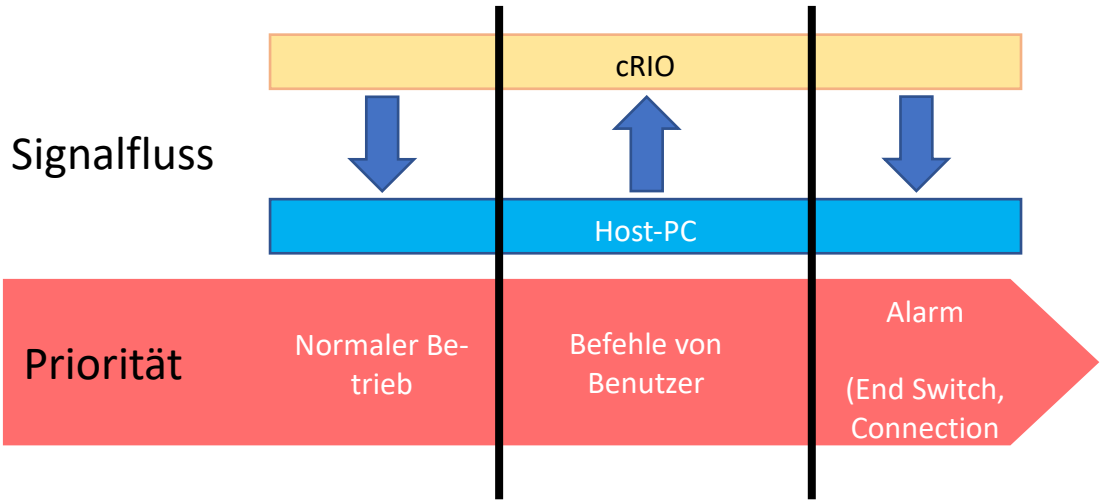


Abb. 3–3: Richtung des Signalflusses abhängig von Ereignissen

Bevor mit der Implementierung begonnen wird, gilt es noch das geeignetste Übertragungsmittel zu wählen. Das cRIO ist mittels Ethernet mit dem Host-PC verbunden. Die IP-Adresse des cRIO ist 129.187.61.250. Es stehen vier Möglichkeiten zur Auswahl: User Datagram Protocol (UDP) und Transmission Control Protocol (TCP), Network Streams und Shared Variables. Das Übertragungsmittel muss fehlerfreie Übertragung garantieren können. Auch sollte es zu möglichst keinem Datenverlust kommen. Da neue Funktionen auch in Zukunft zum Steuercode hinzufügbar sein sollen, sollte das Hinzufügen neuer Übertragungswerte möglichst einfach sein.

UDP wird bereits für die Kommunikation zwischen dem HIL-Teil und der restlichen RACOON-Umgebung verwendet. TCP hat vergleichsweise ähnliche Eigenschaften, besitzt aber den Vorteil gegenüber UDP, dass es verlustlos ist. Beide Protokolle sind effizient und benötigen deutlich weniger Systemressourcen als die anderen beiden Übertragungsmittel. Allerdings lassen sich mit beiden nur der Datentyp String übertragen, sprich Daten müssen erst aufbereitet werden, bevor sie gesendet oder nach dem Empfangen verwendet werden können, was wiederum Ressourcen kostet. Außerdem steigt der Programmieraufwand deutlich an [12].

Network Streams und Shared Variables (auch Umgebungsvariablen genannt) sind LabVIEW-interne Übertragungsmöglichkeiten, die von National Instruments entwickelt wurden. Sie erlauben verlustlose Übertragung auch von LabVIEW-eigenen Datentypen. Während ein Network Stream Punkt-zu-Punkt-Kommunikation in eine Richtung erlaubt, sind Shared Variables auf einer sogenannten Shared Variable Engine gespeichert, welche auf einem beliebigen Gerät gehostet wird. Beim Schreiben wird der Wert einer Variablen in dieser Engine geändert, beim Lesen wird ebenfalls direkt aus dieser gelesen. Der Nachteil beider Möglichkeiten, vor allem der Shared Variables, ist der vergleichsweiße hohe Bedarf an Systemressourcen. Die LabVIEW-Hilfe empfiehlt im Falle von der Zurverfügungstellung des immer neuesten Wertes die Verwendung von Shared Variables, da diese für diesen Zweck optimiert wurden [12].

Da der Steuercode des RACOON-Lab auch in Zukunft immer weiterentwickelt wird, sollte das Implementieren von neuen Funktionen, und damit wahrscheinlich auch neuen Werten, in die Übertragung möglichst einfach gehalten werden. Daher werden für die Datenübertragung zwischen den Schleifen Shared Variables verwendet.

3.3 Implementierung

3.3.1 cRIO

Als Ausgangsmaterial wurde vor allem der ORION-Code verwendet, da dieser deutlich besser strukturiert ist. Unterschiedliche Programmteile, wie etwa Teile des Setpoint Handlers wurden vom RACOON-Code in den ORION-Code integriert.

Als Schnittstelle zur Datenübertragung wurden in die Control Loop zwei neue SubVIs integriert. *Control Loop In* empfängt Daten und führt, falls notwendig, einfache Berechnungen für neu definierte Einstellungen durch. *Control Loop Out* sammelt und überträgt alle benötigten Daten über die Shared Variable Engine an den Host-PC.

Da die GUI Loop auf dem Host-PC mit einer deutlich niedrigeren Frequenz aufgerufen wird als die Control Loop auf dem cRIO, bietet es sich an, das im Abschnitt 3.2 erwähnte Konzept des verringerten Aufrufens anzuwenden. Vor den Netzwerkschnittstellen wird mittels der Nummer der aktuellen Iteration, des Funktionsblocks „Quotient und Rest“ sowie eines an den „Rest“ Ausgang angeschlossenen „=0“-Vergleichs ein boolescher Wert erzeugt, siehe Abb. 3–4. Dieser ist wiederum mit einer booleschen Case-Struktur verbunden, in dem sich eine Netzwerkschnittstelle befindet. Da nur jede vierte Zahl auch durch 9 teilbar ist, wird der Boole-Wert an der Case-Struktur auch nur alle vier Iterationen wahr.

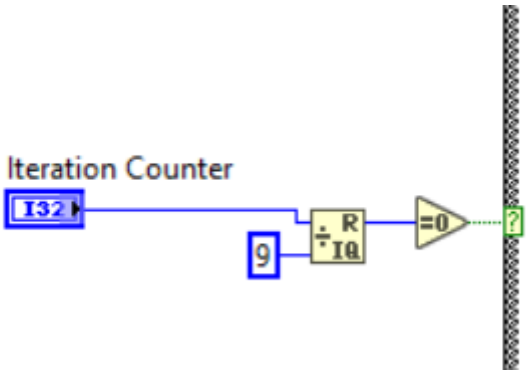


Abb. 3–4: Code zum verringerten Aufrufen der Netzwerkschnittstellen

Da vom Netzwerk sowohl das Lesen als auch das Schreiben, relativ zur gesamten Schleifendauer, lange dauern, soll sichergegangen werden, dass beide Operationen nicht in derselben Iteration erfolgen. Das Ganze wird mithilfe der Trennung der beiden Operationen voneinander, sowie einer einfachen booleschen Variablen *I/O Selector* erreicht. Durch eine Switch-Case-Struktur wird *Control Loop Out* nur bei TRUE, *Control Loop In* nur bei FALSE ausgeführt. Nach dem Ausführen einer der Operationen wird die Variable in den jeweils anderen Zustand geschaltet.

Dieses wechselseitige und das eben beschriebene verringerte Aufrufen hat zur Folge, dass die einzelnen Netzwerkschnittstellen jeweils alle 18 Iterationen, beziehungsweise alle 180 ms aufgerufen werden. Da die GUI Loop am Host-PC alle 200 ms ausgeführt wird, kann jederzeit ein schnelles Verwerten von Befehlen garantiert werden.

Beim Austesten der Netzwerkkommunikation wurde festgestellt, dass das Lesen von Daten innerhalb der Control Loop im cRIO deutlich mehr Zeit in Anspruch nimmt als das Schreiben. Außerdem hängt dieser Aufwand in beiden Fällen direkt mit der Größe der zu übertragenden Arrays ab.

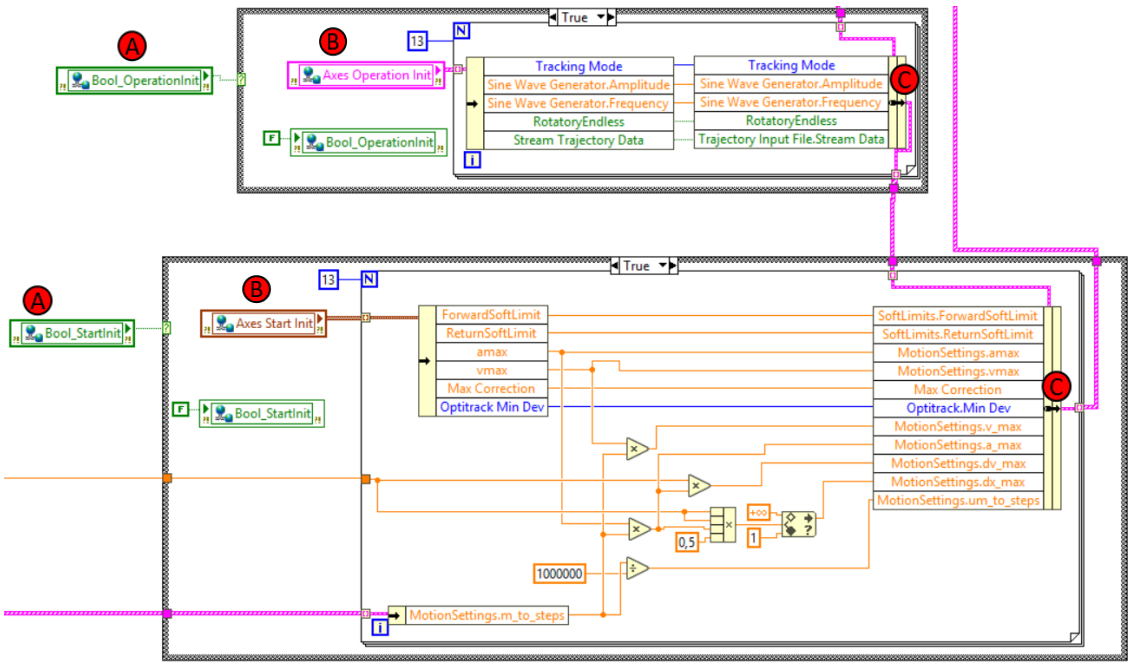


Abb. 3–5: Control Loop In: Aufrufen der Shared Variables nur bei Bedarf

Aus diesem Grund wurde ein System für das Aufrufen des Leseprozesses im cRIO eingebaut, dass diesen Prozess nur wenn benötigt aufruft. Anstatt bei jedem Ausführen der Netzwerk-Schnittstelle immer die gesamten empfangenen Daten aufzurufen, wird lediglich ein Boole-Wert abgefragt, der angibt, ob es eine Änderung im Vergleich zu den gesendeten Daten gibt. Bei TRUE werden die neuen Daten gelesen, bei FALSE wird das Lesen übersprungen, da das Auslesen keine neuen Daten liefern würde und somit unnötig ist. In Abb. 3–5 sind zwei solcher Systeme zu sehen, welche sich im VI *Control Loop In* befinden. Mit einem A markiert sind die booleschen Umgebungsvariablen, die die nachfolgenden Switch-Case-Strukturen steuern, in denen sich die mit B markierten Daten enthaltenen Shared Variables befinden. Nach einem Aufruf werden die Boole-Variablen wieder auf FALSE geschalten. Bei C werden die empfangenen Daten in den alle Achsendaten enthaltenen Cluster *AxisData* geschrieben.

3.3.2 Host-PC

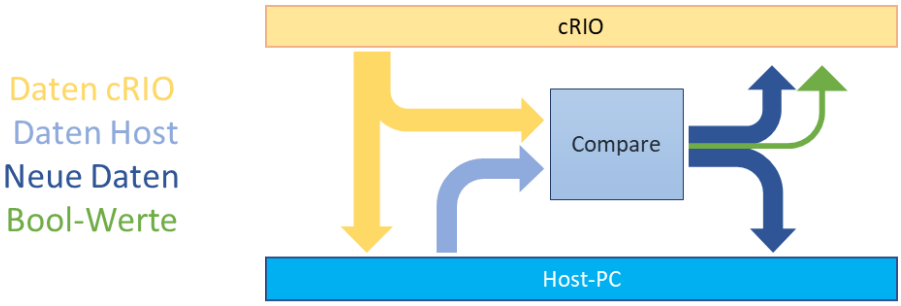


Abb. 3–6: Schema Funktion des Compare-VI

Das Vergleichen von Daten und Generieren der steuernden Boole-Werte, verdeutlicht in Abb. 3–6 findet auf dem Host-PC im SubVI *GUI Loop Compare* statt und läuft so ab: Alle Werte, die der Host-PC an das cRIO überträgt, werden auch vom cRIO an den

Host-PC übertragen. Vor dem Senden der Host-PC-Werte werden nun die zu sendenden Daten auf dem Host-PC mit den Daten vom cRIO verglichen. Gibt es einen Unterschied, so werden die neuen Daten an Host-PC und cRIO gesendet und so als neuer Wert festgelegt. Außerdem wird die bereits erwähnte Boole-Variable auf TRUE geschalten. Ein Beispiel hierfür ist in Abb. 3–7 zu sehen.

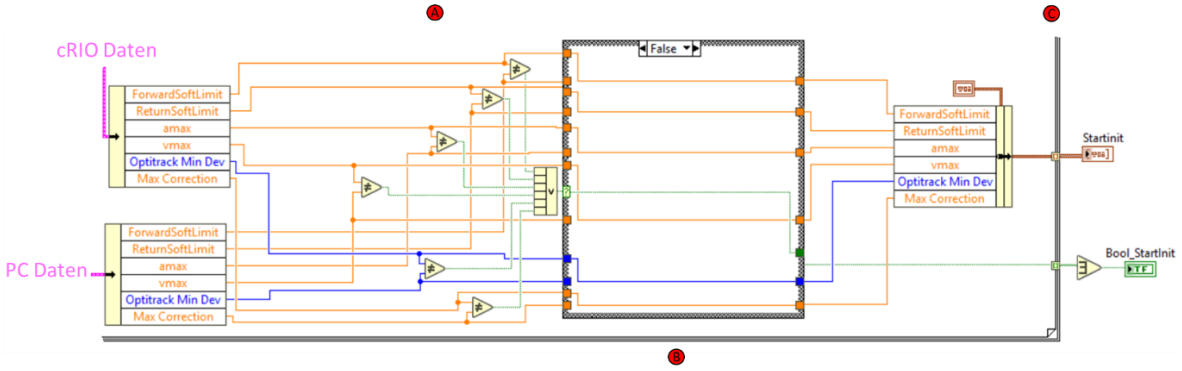


Abb. 3–7: Ausschnitt Compare.vi – Start Init

Die PC-Daten aus der GUI Loop sowie die empfangenen cRIO-Daten werden in Bereich A miteinander verglichen. Sollte bei mindestens einem Wert eine Diskrepanz festgestellt werden, bedeutet dies ein neuer Befehl aus der GUI Loop. Dann wird über die ODER-Verknüpfung die Switch-Case-Struktur auf TRUE geschalten. Anschließend werden die neuen Daten aus der Benutzeroberfläche an die Umgebungsvariable zum Übertragen weitergeleitet, sowie auch der zugehörige Steuerungs-Boole-Wert auf TRUE geschalten.

Um noch mehr von diesem System zu profitieren, sind die zu übertragenden Daten in vier einzelne Kanäle mit ihren jeweils eigenen booleschen Variablen aufgeteilt. So müssen nicht alle Daten neu aus dem Netzwerk gelesen werden, wenn sich nur ein Wert ändert, sondern lediglich die im jeweiligen Kanal. Unterteilt sind die Daten dabei folgendermaßen:

Start Init: Dies ist die größte Datenmenge. Es sind alle Größen, die im Normalfall während des Betriebs der Anlage nicht geändert werden.

Operation Init: Beinhaltet solche Größen, die während des Betriebes zu Beginn einer Operation, etwa dem Definieren einer Sinus-Bewegung, geändert werden.

Target-Vektor und *Operation Mode* sind die mit Abstand am häufigsten geänderten Größen, weshalb sie jeweils über einen eigenen Kanal verfügen.

Für die letzten beiden Kanäle werden im Sub-VI *GUI Loop Compare* auch gleich die im Abschnitt 3.2 erwähnten Prioritäten kontrolliert. Beim Target-Vektor lässt sich dies relativ einfach umsetzen, da hier nur in drei Fällen die Benutzereingaben überschrieben werden sollen. Die programmierte Logik ist in Abb. 3–8 zu sehen.

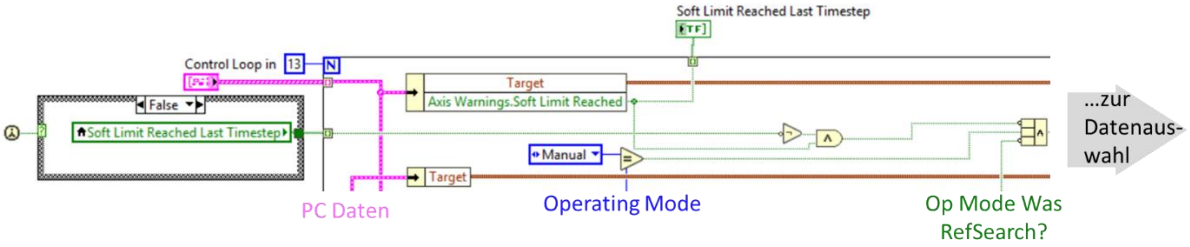


Abb. 3–8: Ausschnitt Compare.vi – Target

Der erste Fall ist der, wenn ein Soft Limit erreicht wird. Dann müssen die Befehle des cRIO Vorrang haben. Allerdings nur für genau diese Iteration, in der das Soft Limit erreicht wird. Andernfalls wird durch das ständig ausgelöste Soft Limit die Steuerung des Benutzers blockiert und dieser kann nicht mehr aus dem Soft Limit herausfahren. Darum wurde der zusätzliche Wert *Soft Limit Reached Last Timestep* eingeführt. Dieser wird negiert und mit dem aktuellen *Soft Limit Reached* durch eine Konjunktion verknüpft. Das Ergebnis von dieser ist nur wahr, wenn in der letzten Iteration *Soft Limit Reached* unwahr war, in der aktuellen aber wahr ist. Diese Konfiguration wandelt den dauerhaften *Soft Limit Reached*-Wert in einen bloßen Impuls beim ersten Ändern um. Die zweite Bedingung für das Schreiben von Target-Werten ist, dass der aktuelle Betriebsmodus *Manual* ist. Der dritte Fall, bei dem das cRIO Vorrang hat, ist der, wenn gerade eben in der letzten Iteration ein RefSearch abgeschlossen wurde. Wird in diesem Fall dem cRIO kein Vorrang gewährt kann es passieren, dass die Target-Werte, die während eines RefSearch vom cRIO geändert werden, nicht wieder zurück auf null geschaltet werden.

Beim Operating Mode ist die Umsetzung komplizierter. Um das gewünschte Verhalten des Compare-VI zu erhalten, muss zuerst festgelegt werden, in welchen Fällen der cRIO-Teil des Steuercodes Benutzereingaben überschreiben soll. Die Anlage schaltet in den *Stopping*-Modus für folgende Ereignisse: das Auslösen eines End Switches, Verlust der Netzwerkverbindung im *Network*-Betriebsmodus, Verlust der Optitrack-Netzwerkverbindung im *Closed-Loop*-Trackingmodus und beim Erreichen des Endes eines Trajectory Files. Außerdem soll beim Abschließen einer Referenzfahrt im Ref-Search-Betriebsmodus die Anlage automatisch in den Manual-Modus schalten. Der Code zur Steuerung der Operating-Mode-Übertragung ist in Abb. 3–9 zu sehen.

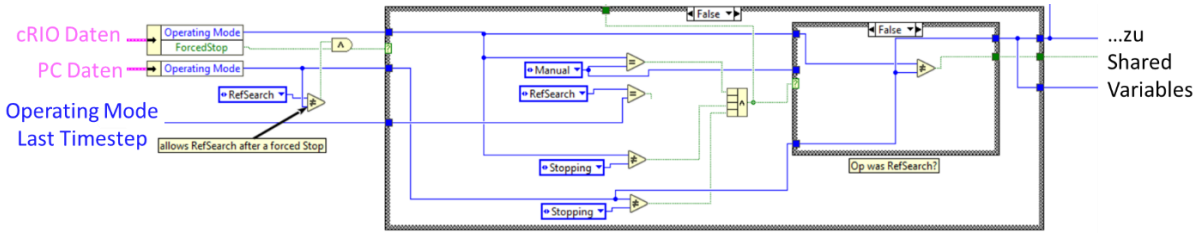


Abb. 3–9: Ausschnitt Compare.vi – Operating Mode

Für die Ereignisse, die zum Stoppen der Anlage führen, wurde eine neue Variable *ForcedStop* eingeführt. Sie ist vom Dateityp Boolean und für jede Achse im Cluster-array AxisData enthalten. Tritt nun ein Ereignis ein, weswegen der cRIO-Steuercode die Anlage automatisch in den Stopping-Modus schaltet, wird diese Variable auf TRUE gesetzt. Im Sub-VI Control Loop Output wird sie zusammen mit den restlichen Daten

an die GUI Loop übertragen. Nach jedem Übertragen wird ForcedStop wieder auf FALSE geschalten.

Liegt nun auf dem Host-PV im Sub-VI Compare ein ForcedStop mit dem Wert TRUE vor, wird statt einer Benutzereingabe immer der Befehl aus dem cRIO übertragen. Ein Überschreiben solcher sicherheitskritischer Stopp-Signale aufgrund von ungünstigem Timing bei der Kommunikation zwischen cRIO und Host-PC ist nicht möglich, da solche Signale dauerhaft anhalten, und direkt vor jedem neuen SplineInfo-Berechnen erneuert werden. Um ein Bewegen der Anlage nach einem Forced Stop zu erlauben, kann der *Stopping*-Befehl mit einem Befehl zum Starten einer RefSearch überschrieben werden.

Um das automatische Umschalten nach erfolgreichem Beendigen einer Referenzfahrt zu ermöglichen, wurde in Compare-VI eine zusätzliche Rückkopplung des Operation Mode-Wertes der vorherigen Iteration hinzugefügt. Um das gewünschte Verhalten zu erreichen, wurden folgende Vergleiche mit einem logischen AND verknüpft:

- Der aktuelle Betriebsmodus des cRIO ist Manual
- Der aktuelle Betriebsmodus des cRIO ist nicht Stopping
- Der aktuelle Betriebsmodus laut GUI ist nicht Stopping
- Der Betriebsmodus des cRIO in der vorherigen Iteration war RefSearch

Sind alle Bedingungen wahr, was nur nach dem erfolgreichen Beendigen einer RefSearch der Fall sein kann, wechselt der Betriebsmodus auf beiden Geräten zu Manual.

Das VI *GUI Loop Compare* befindet sich innerhalb des *GUI Loop Comm Interface.* Dieses ist direkt in der GUI Loop untergebracht, ein schematischer Aufbau ist in Abb. 3–10 zu sehen.

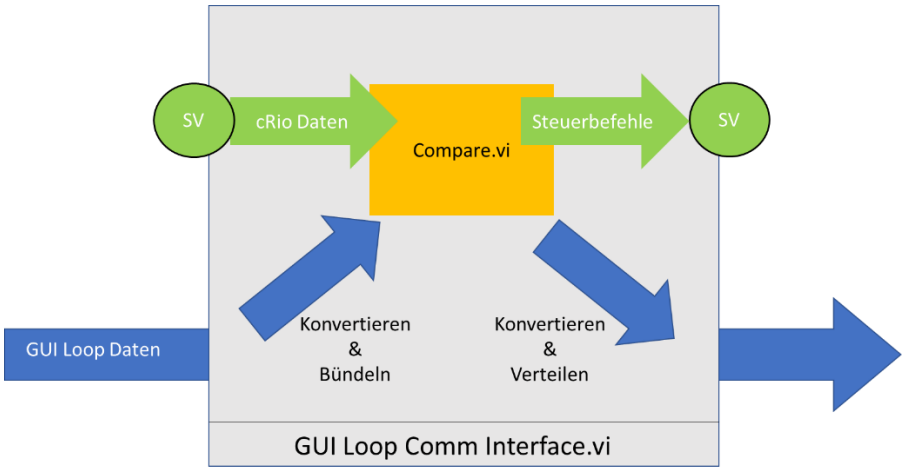


Abb. 3–10: Comm Interface schematische Struktur

In diesem werden Daten aus der GUI Loop von den Anzeigeeinheiten Meter und Radianten in die interne Einheit Steps konvertiert, um ein Vergleichen mit den cRIO-Daten innerhalb des *Compare*-VI zu erlauben. Im *Comm Interface* werden auch die Shared Variables zum Empfangen von Daten und Senden von Steuerbefehlen aufgerufen. Nach erfolgtem Vergleichen werden Vergleichsergebnisse und weitere vom cRIO erhaltene Werte wieder in Meter und Radianten konvertiert und weiter in die GUI Loop geschickt.

3.4 Ergebnisse

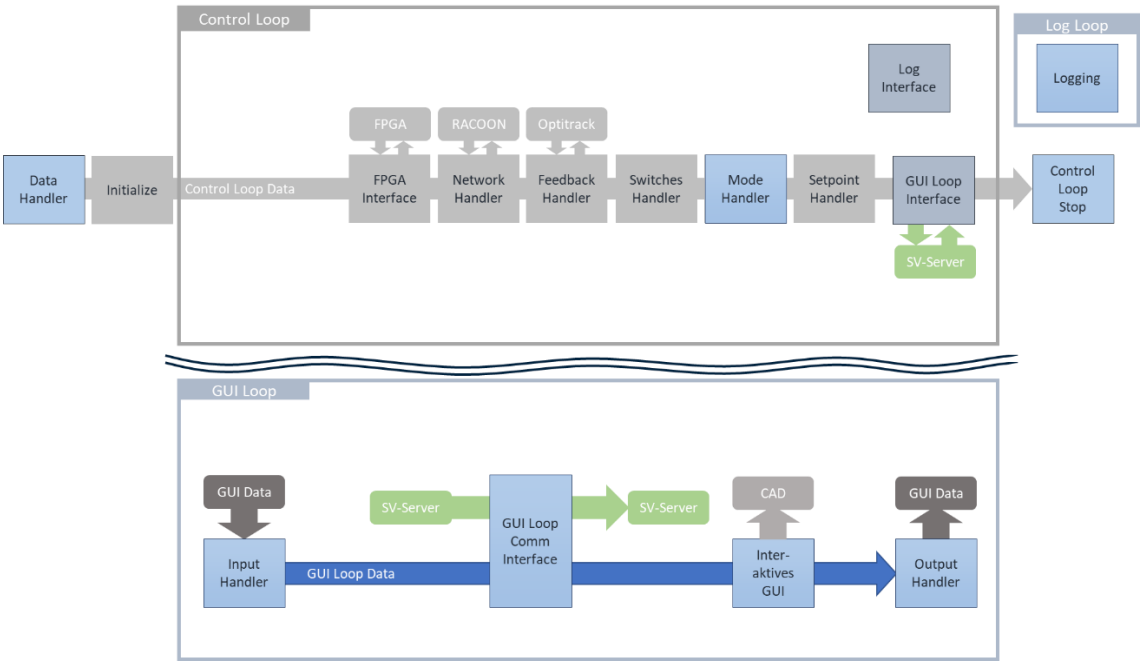


Abb. 3–11: neue Programmstruktur des RACOON-Labs

Die Struktur des nun neu geschriebenen Codes ist in Abb. 3–11 schematisch dargestellt. Auch hier stellen die beiden parallelen Linien die Trennung zwischen Host-PC und cRIO dar und wesentliche Änderungen sind wieder in Farbe gehalten. Die größte Änderung im Vergleich zu den beiden vorherigen Steuerprogrammen ist wohl die Verwendung von mehreren Schleifen.

Auf dem cRIO-Controller läuft die Control Loop, die zum Großteil aus den ursprünglichen beiden Codes besteht und weiterhin den wesentlichen Kern des Programms darstellt. Ergänzt wurden hier die beiden Schnittstellen zum Senden und Empfangen von Daten.

Neu ist die GUI Loop, die gänzlich auf dem Host-PC läuft. Sie ist in erster Linie für das asynchrone User Interface als *Human Machine Interface* (HMI) zuständig. Dieser Programmteil wird im Abschnitt 4.2 weiter erläutert.

Der einfachste Weg zum Hinzufügen neuer zu übertragende Daten in die Shared Variables ist, die entsprechenden Type Definitions mit dem neuen Wert zu ergänzen. Jeder komplexeren Umgebungsvariable ist eine solche Definition mit gleichem Namen zugeordnet. Anschließend die Eigenschaften der zu ändernde Variable aufrufen, im Data-Type-Menü *Custom Controls* auswählen und die eben geänderte Typendefinition auswählen.

Werte zu den Einstellungen wie dem RefSearch können nun wie beim ORION-Lab im SubVI *Initialize* festgelegt und gespeichert werden. Es gilt jedoch, dass alle Werte, die auch in der GUI Loop festgelegt werden können, die Werte in *Initialize* überschreiben.

Funktionen, die in diesem Schema bereits zu sehen sind, werden im nachfolgenden Kapitel 4 beschrieben. Zum Beispiel das in einer weiteren, eigenen Schleife ausgelagerte Logging. Die notwendigen Daten hierfür werden im Log Interface an diese Schleife übertragen. Da Trajektorien-Daten im *Data Handler* vor der Ausführung der

Control Loop eingelesen werden, muss so innerhalb der Control Loop kein Festplattenzugriff erfolgen, ein CPU-intensiver und nicht-deterministischer Prozess.

Leistungstechnisch wurden Verbesserungen erzielt. Im Ruhezustand der Anlage liegt nun mit der erneuerten Version, inklusive aller zusätzlichen Funktionen aus den nachfolgenden Kapiteln, die CPU-Auslastung bei rund 60 %, bei Volllast zwischen 60 % und 70 %. Die maximale Periodendauer wird eingehalten. So stehen nun trotz der bereits implementierten Funktionen immer noch Systemressourcen für weitere Neuerungen zur Verfügung. Bei den entscheidenden Übertragungszeitpunkten der FPGA-Informationen können keine nennenswerten Schwankungen gemessen werden. Bei der Benutzung der Bedienoberfläche, auch über längere Zeit, treten keinerlei spürbaren Verzögerungen auf.

Für zukünftige Entwicklungen sollte unbedingt der Grundgedanke der Funktionstrennung und Priorisierung im Kopf behalten werden. Zeitkritische Prozesse müssen immer ungehindert von anderen Tasks ablaufen können. Selbst in seinem jetzigen Zustand ist die Control Loop noch lange nicht auf Geschwindigkeit und Effizienz optimiert. Beispielsweise ist die Vorbelegung von Speicherplatz nur teilweise umgesetzt. Auch kann es durch das Teilen desselben Prozessors auch bei einer optimal programmierten Control Loop zu verzögertem Ausführen kommen, wenn ein anderer Prozess diese blockiert. Ein Multi-Core-Prozessor könnte in diesem Fall Abhilfe schaffen.

4 Neue Funktionen

4.1 Implementierung neuer Funktionen

Da nun Dank der neuen Grundstruktur des Steuerprogramms wieder genug freie Systemressourcen zur Verfügung stehen, sollen nun die neuen Funktionalitäten implementiert werden. Zum Großteil basieren diese auf den neuen Funktionen der ORION-Anlage, wie sie in der einer vorangegangen Arbeit beschrieben werden [4].

Implementiert werden sollen dabei folgende Funktionen:

- Positions-Rückkopplung mit Optitrack
- Logging
- Trajectory-Follow-Betriebsmodus
- Sine-Sweep-Betriebsmodus
- Manuelle Steuerung

4.1.1 Positions-Rückkopplung

Das Ziel ist es, von Infrarotkameras ermittelte Orts- und Lageinformationen zurück in den Steuercode der RACOON-Anlage zu senden, um mithilfe von diesem Feedback die Anlagenpräzision zu erhöhen.

Das Konzept wurde bereits erfolgreich im ORION-Lab umgesetzt und soll nun auch in die RACOON-Anlage implementiert werden. Dieses Rückkopplungssystem besteht sowohl aus Hardware-, als auch Softwareteilen. Eine Übersicht ist in Abb. 4–1 dargestellt.

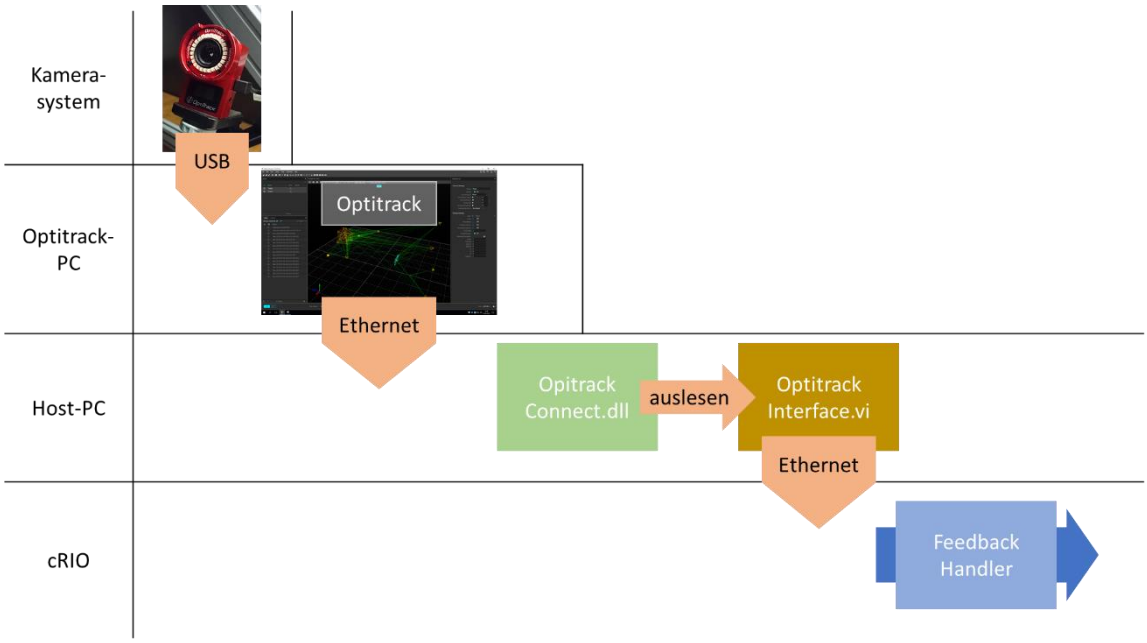


Abb. 4–1: Schema Datenübertragung Optitrack

Ein Kamerasystem filmt auf Anlagenelemente angebrachte reflektierende Marker und schickt diese so aufgenommenen Bilder über USB an den Optitrack-PC. Die

zweidimensionalen Lagepunkte der Marker auf den einzelnen Kamerabildern werden durch die Software *Motive* in dreidimensionale Punkte im Raum umgerechnet. Unter der Annahme, dass einzelne Marker-Gruppen auf Festkörper angebracht wurden, kann *Motive* so die Position und Lage dieser Festkörper bestimmen.

Diese Daten werden über Ethernet an den Host-PC übertragen, auf dem ein in C++ geschriebener Code diese Werte liest. Dieser C++-Code fungiert dabei als Schnittstelle zwischen Optitrack- und LabVIEW-Software. Mittels LabVIEW-Funktionen zum Auslesen von DLL-Dateien werden dann Orts- und Lagedaten in dem VI *OptitrackInterface.vi* eingelesen, gefiltert, achsenweise in das Anlagen-Koordinatensystem umgerechnet und als String mittels UDP weitergeschickt. Diese Daten erhält dann über Ethernet das cRIO. Im Sub-VI *FeedbackHandler.vi* werden diese Informationen mit den programmintern berechneten Positionen der einzelnen Achsen verglichen. Falls es Abweichungen gibt, werden entsprechende Korrekturen ausgeführt [4].

4.1.1.1 Probleme beim Verwenden von Optitrack

Leider ist das RACOON-Lab für präzise, erfolgreiche Optitrack-Messungen suboptimal ausgestattet. Ein Problem etwa ist die Art, wie die Kameras im Raum angebracht sind. So kommt es vor, dass einige der hinteren Target-Kameras im Bild der vorderen Chaser-Kameras zu sehen sind. Dies ist insofern schlecht, dass alle Kameras mit Infrarot-LEDs ausgestattet sind, um die Marker hell reflektieren zu lassen. Diese LEDs der Target-Kameras sind dann im Bild der Chaser-Kameras als Störquelle zu sehen. Aus demselben Grund sollten auch die zur Überwachung genutzten, zusätzlichen Kameras in der Anlage ausgeschaltet werden.

Auch wäre es vorteilhafter, die insgesamt acht Kameras nicht gleichmäßig verteilt auf jeweils eines der beiden Anlagenteile auszurichten. Stattdessen sollten diese in unregelmäßigen Höhen so im gesamten Raum verteilt werden, dass ein möglichst großer Bereich des Kamera-Sichtfeldes den Raum erfasst. Leider ist mit der momentanen Optitrack-Ausstattung des RACOONs ein Neuausrichten der Kameras nur schwer umzusetzen, da das verwendete Flex-3-Modell nur über USB verbunden werden kann. Deren USB-Kabel besitzen eine maximale Länge von 5 Meter [13].

Ein anderes Hindernis zu präzisen, zuverlässigen Messungen ist die Target-Attrappe. Sie ist, um einem echten Satelliten zu ähneln, auf allen 4 Seiten mit glatter oder zerknitterter, reflektierender Folie bedeckt. Zwar simuliert diese auch in Wirklichkeit auftretende Störquellen und erhöht die Nähe zur Realität, etwa beim Testen von optischen Messsystemen. Allerdings ist das Problem, dass Optitrack eben auch genauso ein System ist. Die verwendeten Folien reflektieren im Infrarotbereich sehr ähnlich wie sie es im sichtbaren Bereich tun. Dabei stellen die zerknitterte, goldene Folie, sowie die glatte, silberne Folie jeweils auf eigene Art ein Problem dar. Die goldene Folie reflektiert die Infrarotstrahlung der Kameras mit ihren vielen kleinen Flächen in viele verschiedene Richtungen. Aus Sicht der Kameras wirkt dies wie viele reflektierende Marker, ein solches Beispiel ist in Abb. 4–2 zu sehen. Die eigentlichen Marker sind auf dem linken, von Optitrack erstellten Bild gelb eingefärbt. Alle anderen orangen Marker sind fälschlicherweise registrierte Reflexionen. Zwar gibt es einen softwareinternen Filter in *Motive*, der etwa nach Größe und Rundheit ein Großteil dieser Störungen filtert. Entschärft wird das Problem dadurch allerdings nicht. So werden einige Flächen trotzdem als Marker erkannt. Problematisch ist auch, wenn aus der zweidimensionalen Sicht einer Kamera durch Rotation der Attrappe ein Marker in

ein Gebiet mit goldener Folie wandert. Dann überlagern sich Marker und eigentlich gefilterte Folienreflektionen, und das Gesamtgebilde wird durch den Rundheitsfilter herausgefiltert.

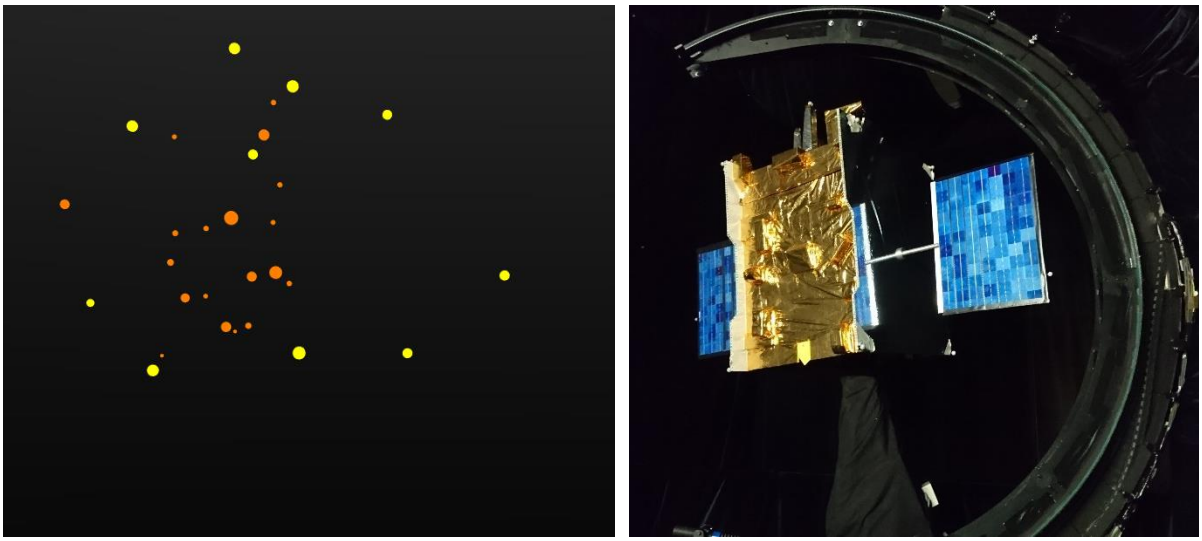


Abb. 4–2: Reflexionen als Störquelle beim Messen mit Optitrack

Die glatte, silberne Folie hingegen wirkt wie ein Spiegel. Dadurch kann es gelegentlich auftreten, dass Marker in der Folie reflektiert werden und als zusätzlicher Ghost-Marker von Optitrack erkannt wird. Durch die oft nur um wenige Millimeter abweichende Lage werden diese Reflektionen teils fälschlicherweise als Marker des definierten Festkörpers erkannt. Auch kann es passieren, dass Marker und Reflektion sich überlagern und so vom Rundheitsfilter herausgelöscht werden.

Durch Anbringen von zusätzlichen Markern an den weiter entfernten Solarpanelen konnten die erwähnten Probleme bei der Attrappe zwar etwas entschärft werden. Messungen in Bewegung des Targets werden allerdings trotzdem weiterhin durch relativ starke Schwankungen verzerrt.

Zwar sind Optitrack-Messungen mit der gegebenen Hardware gerade bei Messungen in der Bewegung, unter anderem aus den oben genannten Gründen, Unsicherheiten und Abweichungen ausgesetzt. Nutzbar ist das System jedoch trotzdem. So treten die oben genannten Probleme beim Chaser-Teil der Anlage nur sehr eingeschränkt auf. Außerdem kann auch auf zusätzliche Filter oder bei Messungen im Stillstand auf Mittelwerte zurückgegriffen werden.

Erfahrungsgemäß sollte für das RACOON der *ray-based tracking algorithm* verwendet werden. Hier sind die auftretenden Messfehler in der Regel kleiner.

4.1.1.2 Implementierung der Software

Dank der großen Ähnlichkeit zwischen ORION und RACOON konnten weite Teile der bestehenden Software einfach übernommen werden. Kleine Änderungen, etwa zu Übertragungsarten und IP-Adressen, mussten im C++- und im LabVIEW-Code gemacht werden. Auf dem cRIO im Feedback Handler wurde die String-Umwandlung ausgetauscht. Grund war ein fehlerhaftes Verhalten des Funktionsblocks *Spreadsheet*

String to Array. Dieser versteht das übertragene Dezimaltrennungs-Komma als Spaltentrennung. Dieses Verhalten tritt nur auf dem cRIO auf, nicht auf dem Host-PC. Eigentlich sollte man für den Funktionsblock definieren können, welche Zeichen als Spaltentrennung verwendet werden. Allerdings wurde bei verschiedenen Zeichen auch immer das Komma als Spaltentrennung übersetzt, auch wenn es nicht als solches definiert wurde. Die Ursache konnte nicht endgültig herausgefunden werden, da der zu Grunde liegende Code von solchen Funktionsblöcken nicht öffentlich zugänglich ist. Vermutlich gibt es allerdings auf dem cRIO Probleme mit der Unterscheidung zwischen deutscher und amerikanischer Schreibweise der Dezimaltrennung. In Amerika und wohl auch auf dem cRIO wird ein Punkt anstelle eines Kommas verwendet. Auf diesen Unterschied sollte beim zukünftigen Entwickeln geachtet werden, um Fehler zu vermeiden.

Die einzige größere erforderliche Änderung ist die Umrechnung von Orts- und Lageinformationen, angegeben im Optitrack-Koordinatensystem in Lagepunkten der einzelnen Achsen. Hier treten nun auch die Unterschiede zwischen ORION und RACOON in Erscheinung. Während Chaser- und Target-Koordinaten im ORION-Lab lediglich mit einfachen Rotationen der Koordinatenachsen umgerechnet werden können, gestaltet sich diese Transformation bei der RACOON-Anlage deutlich komplizierter. Ein Grund ist der, dass Rotationsachsen sich nicht überall in einem Punkt schneiden. In Optitrack lässt sich jedoch nur ein Rotationspunkt festlegen. Das hat zur Folge, dass etwa bei einer Rotation des Chaser um seine z-Achse es gleichzeitig immer zu einer translatorischen Bewegung in x- und y-Richtung kommt. Durch Abmessen des Hebelarms zwischen definiertem Rotationspunkt und wirklicher Rotationsachse, sowie dem Einsatz der Sinus- und Kosinus-Funktion kann dieser Fehler jedoch beseitigt werden.

Beim Target kommt zusätzlich erschwerend noch die Tatsache hinzu, dass dieses Teilsystem überbestimmt ist. Durch die C-Schiene werden zwei zusätzliche Rotations-Freiheitsgrade hinzugefügt. Um überhaupt sinnvoll die Lage der einzelnen Achsen bestimmen zu können, muss mindestens der Schlitten, auf dem die Satellitenattrappe sich translatorisch auf dem Radius der C-Schiene bewegt, mit zusätzlichen Markern ausgestattet werden, und als eigener Festkörper mit in die Lageberechnungen einfließen.

Da im Laufe dieser Arbeit deutlich mehr Zeit als ursprünglich geplant in das Erstellen einer funktionierenden, stabilen Programmstruktur geflossen ist, war leider nicht mehr genug Zeit, um die eben beschriebenen Probleme beim Umrechnen von Orts- und Lagevektoren anzugehen. Alle anderen Programmteile der Positions-Rückkopplung sind jedoch erfolgreich implementiert worden. Nicht umgerechnete Werte können in die Steuersoftware gestreamt werden.

4.1.2 Logging

Das Logging von internen Daten ist eine weitere Neuheit im ORION-Code. Ziel ist es, ohne Einflussnahme auf den Programmablauf in jeder Iteration Daten wie Ort oder Geschwindigkeit der Achsen zu speichern, und ihre Auswertung zu einem späteren Zeitpunkt zu ermöglichen.

Die ORION-Version des Logging kann auf dem RACOON-cRIO die Bedingung *ohne Einflussnahme* nicht erfüllen. Während des Logging kommt es zur hohen CPU-Auslastung und immer wieder zu Überschreitung der maximalen Periodendauer. Sehr ungünstig ist außerdem die Tatsache, dass der Festplattenzugriff in der gleichen Schleife wie zeitkritische Prozesse stattfindet.

4.1.2.1 Lösungsansätze

Um den Festplattenzugriff außerhalb der zeitkritischen Control Loop zu verschieben, soll der Ansatz mit separaten, priorisierten Schleifen verwendet werden. Es gibt zwei Möglichkeiten, wo das Abspeichern erfolgen soll. Direkt auf dem cRIO werden Systemressourcen für den Prozess des Festplattenschreibens in Anspruch genommen. Auf dem Host-PC ist dies nicht der Fall. Allerdings müssen dafür Ressourcen für den Netzwerkzugriff aufgewandt werden. Außerdem entsteht dadurch zusätzlicher, nicht zu priorisierender Datenverkehr in derselben Leitung, die etwa auch für Steuerbefehle genutzt werden.

Um das Problem der CPU-Last zu vermindern ist eine Möglichkeit, den Vorgang des Festplattenzugriffs seltener auszuführen. Wenn mehr Text auf einmal geschrieben wird, verringert sich relativ gesehen der Overhead und das Schreiben wird effizienter. Allerdings wird die Dauer eines einzelnen Schreibvorgangs dadurch erhöht.

Durch das Abspeichern der gewünschten Daten in einer Binärdatei anstatt in einer Textdatei lassen sich sowohl CPU-Last als auch Ausführdauer verringern. Durch das direkte Abspeichern der Bits entfällt die Umwandlung der verschiedenen Datentypen in String. Außerdem wurde beim Austesten der verschiedenen Speichermöglichkeiten festgestellt, dass das Schreiben auf die Festplatte mit Binärdaten effizienter ist als das Abspeichern in ASCII-Zeichen. Der Nachteil dieser Methode ist jedoch, dass dann die Daten in einer .bin-Datei vorliegen, die nur von einem LabVIEW-Programm gelesen werden kann. Zusätzlich müssen auch noch das genaue Format und der Datentyp der gespeicherten Daten bekannt sein.

Für eine kurze Logging-Dauer mit großem Datendurchsatz bietet sich die zusätzliche Verwendung eines Puffers an. Durch das Schreiben der Daten in den Arbeitsspeicher statt direkt auf die Festplatte können so, zeitlich begrenzt, mehr Daten abgespeichert werden, als in derselben Zeit auf die Festplatte geschrieben werden können.

Wenn nur bestimme Achsen von Interesse sind, kann auf das Abspeichern der übrigen Achsen verzichtet werden. Da auch beim Logging erforderlicher Aufwand in direktem Zusammenhang mit der Achsenanzahl steht, können auch hier auf diesem Weg Zeit und Ressourcen gespart werden.

4.1.2.2 Implementierung

Das nun neu entwickelte Logging basiert lose auf dem des ORION-Labs, allerdings wurden einige wesentliche Teile hinzugefügt oder abgeändert. Als Ausführungsort für das neu programmierte Logging wurde das cRIO gewählt. Der Grund hierfür war der, dass die Systemressourcen-Einsparung durch Auslagern des Logging und Einrichten einer Netzwerkverbindung kaum merkbar war, dadurch die Netzwerkauslastung erhöht und das Gesamtsystem verkompliziert werden würde. Ein kleiner Nachteil der lokalen Abspeicherung auf dem cRIO liegt darin, dass Logging-Daten im Normalfall

auf dem Host-PC weiterverarbeitet werden. Sprich: durch das lokale Abspeichern müssen Logging-Dateien von Hand auf den Host-PC geholt werden. Um über den Host-PC auf das cRIO zuzugreifen, bietet sich die Verwendung des *File Transfer Protocol* (FTP) an. Für seine Verwendung wird lediglich der *Windows Explorer* benötigt. Durch Eingabe von *ftp://xxx,* wobei *xxx* die IP-Adresse des cRIO ist, kann direkt auf die Festplatte des cRIO zugegriffen werden. Die aktuelle IP-Adresse des RACOON-cRIO ist 129.187.61.250.

Ist das Logging aktiviert, werden in jeder Control-Loop-Iteration für jede Achse die aktuelle Position, Geschwindigkeit und Beschleunigung in das Log Interface übertragen. Dessen Code ist in Abb. 4–3 zu sehen. Hier erfolgt, je nach Wunsch des Benutzers, eine Umrechnung der Werte von Steps zu Meter bzw. Radianten oder nicht. Aus diesen Werten wird unter Hinzunahme der Zeit ab Logging-Start ein eindimensionales Array erstellt. Dieses Array ist dabei wie eine Zeile der Tabelle Tab. 4–1 aufgebaut. Die Zeit ab Start wird mittels der Systemzeit berechnet, indem die aktuelle Systemzeit von der Systemzeit bei Logging-Beginn abgezogen wird.

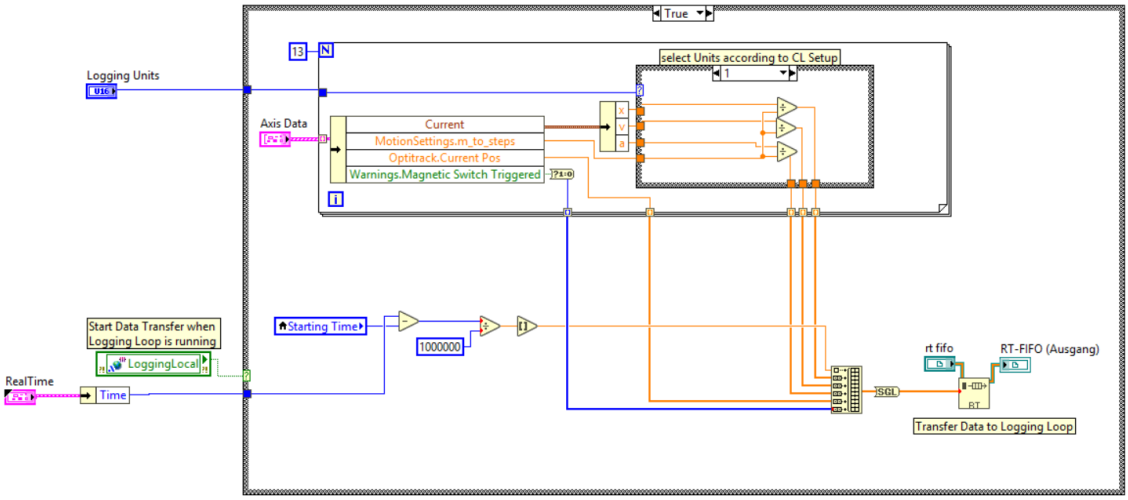


Abb. 4–3: Log Interface

Danach wird das erstellte Array in ein Real Time FIFO geschrieben, welches sowohl als Übertragungsmittel als auch als Puffer dient. In FIFOs (First in First out) können zuvor definierte Daten geschrieben und an einer anderen Stelle im Programm, etwa einer anderen Schleife, wieder ausgelesen werden. Sobald ein Wert ausgelesen wurde, wird er aus dem Speicher gelöscht. In einem FIFO können mehrere Werte eines gleichen Datentyps gespeichert werden. Real Time FIFO eignen sich für ein deterministisches Übertragen von Informationen innerhalb eines Programms [14].

Ausgelesen wird das FIFO in einer separaten, zeitgesteuerten Logging-Schleife, die alle 80 ms ausgeführt wird. Diese ist in Abb. 4–4 zusehen. Leider konnten keine Informationen zu einem optimalen Verhältnis von Datenmenge zu Schreibfrequenz beim Dateischreiben auf einem cRIO gefunden werden. Die 80 ms wurden nach Ausprobieren verschiedener Werte als bester Wert festgelegt.

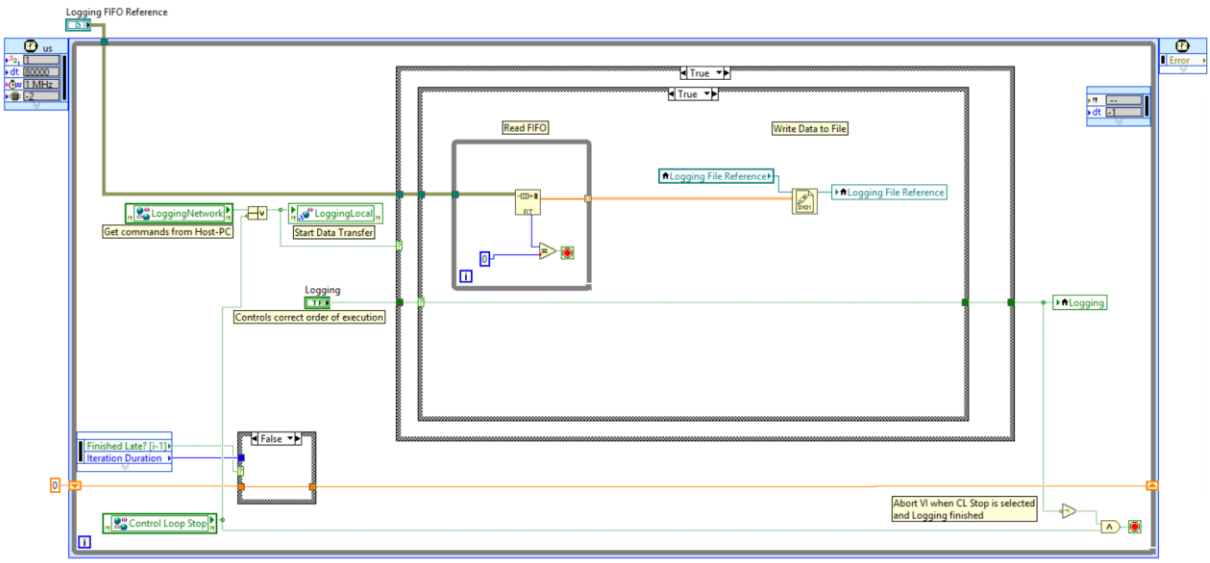


Abb. 4–4: Logging Loop

Das Lesen selbst findet innerhalb einer weiteren Schleife ohne festgelegter Ausführfrequenz statt. Stattdessen wird diese Schleife einfach schnellstmöglich ausgeführt, bis die Abbruchbedingung erfüllt ist, welche hier ein leeres FIFO ist. Wenn diese Bedingung erfüllt ist, dann werden alle ausgelesenen, eindimensionalen Arrays, die bei den vorherigen Iterationen gelesen wurden, untereinander in Zeilen angeordnet, so dass ein neues, nach der Zeit ansteigend geordnetes, zweidimensionales Array entsteht. Dieses wird dann mittels des Funktionsblocks *Write to Binary File* in eine erstellte und geöffnete Datei geschrieben. Bei jeder Logging-Loop-Iteration werden also acht bis neun eindimensionale Arrays auf einmal untereinander in eine Datei geschrieben, so dass letzten Endes eine chronologische Logging-Tabelle entsteht, wie sie in der Tab. 4–1 angedeutet ist. Es wurde als Dateiformat eine Binärdatei gewählt, da beim Schreiben in einer Textdatei es zu gelegentlichem Überschreiten der maximalen Periodendauer in der Control Loop kam.

Tab. 4–1: Aufbau einer Logging-Datei

(Zeit in ms)	Position Achsen			Geschwindigkeit Achsen			Beschleunigung Achsen		
t0	x0…	…	x12	v0	…	v12	a0	…	a12
t1	x0…	…	x12	v0	…	v12	a0	…	a12
…	…	…	…	...	…	…	…	…	…

Die verschachtelte Case-Struktur, die das Erstellen, Schreiben und Schließen der Logging-Datei verwaltet, wurde vom ORION-Code übernommen.

Damit bei einem deaktivierten Logging keine Systemressourcen verschwendet werden, wurde eine weitere Switch-Case-Struktur um das Logging herumgebaut, das den

Großteil der Schleife deaktiviert. Zusätzlich wurde eine lokale Umgebungsvariable eingeführt, die auch das Logging Interface in der Control Loop deaktivieren kann.

Zum Aktivieren des Logging muss auf dem Host-PC der entsprechende Schalter auf TRUE geschalten werden. Dadurch wird eine Netzwerk-Umgebungsvariable ebenfalls wahr, welche in der Logging-Schleife ausgelesen wird, und dort die im vorherigen Absatz erwähnte Switch-Case-Struktur und damit das Logging aktiviert.

Aufgrund der unterschiedlichen Ausführfrequenzen von Control und Logging Loop kann es passieren, dass auch nach dem Beenden des Logging noch ein paar Einträge im FIFO verbleiben. Damit diese nicht beim nächsten Aktivieren in eine neue Datei geschrieben werden, wird das FIFO bei einer Logging-Deaktivierung beim Schließen der Logging-Datei wieder geleert.

Da die wichtigsten Werte nun auch über längere Zeit stabil geloggt werden können, wurde auf einen über die Zeit anwachsenden Puffer verzichtet, wie er in den Lösungsansätzen vorgeschlagen wurde.

Um erhaltene Binärdateien in lesbare Textdateien umzuwandeln, wurde das VI *TOOL TransformateLogging* programmiert. Der Code des im Grunde recht einfachen Programms ist in Abbildung Abb. 4–5 zu sehen.

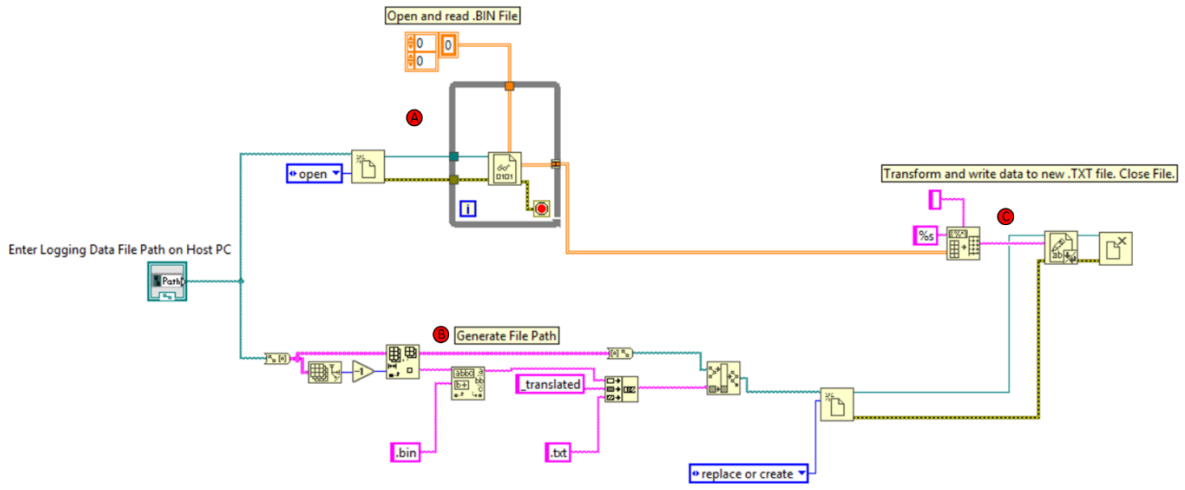


Abb. 4–5: Transformate Logging Tool

Bei A wird eine Datei, die auf einem auf dem Frontpanel angegebenen Pfad liegt, geöffnet, mithilfe des gleichen Datentyps wie beim Schreiben, Double Array, gelesen und bei C in eine neue TXT-Datei geschrieben. Als Dezimalseparator wird ein Komma verwendet, als Spaltentrennung ein Leerzeichen. Bei B werden Pfad und Name der neuen Datei gebaut. Der Pfad ist immer gleich dem der geöffneten Datei, der Dateiname ist gleich der geöffneten Datei plus *_translated* am ursprünglichen Namen angehängt.

Die Neuentwicklung des Logging auf dem RACOON-cRIO ist leider durch die erforderliche Umwandlung umständlicher als die des ORION. Auch wurde das Loggen der SplineData bei der Übertragung auf das RACOON-Lab gestrichen. Allerdings wurde das Ziel, das Erstellen einer Logging-Funktion ähnlich der des ORION, bei der die

wichtigsten Daten in jeder Iteration ohne Beeinflussung des restlichen Programmablaufs in einer Datei gespeichert werden können, erreicht.

4.1.3 Trajectory Follow

Der *Read-Trajectory-File*-Betriebsmodus erlaubt dem cRIO das Lesen und Nachfahren einer Trajektorien-Datei, in der für jede Achse pro Iteration ein Positionssollwert angegeben ist.

Die Trajectory-Follow-Funktion wurde beim ORION leider nicht optimal umgesetzt. Das Einlesen der Trajektorien-Datei erfolgt hier, während das Programm läuft. Dies stellt ein großes Risiko bezüglich stabiler Ausführung dar, da das Lesen größerer Textdateien sehr arbeitsintensiv ist. Außerdem ist der Befehl des Dateilesens innerhalb einer For-Schleife platziert, sodass diese Aktion für jede Achse einzeln durchgeführt wird, wobei jedoch ein einziges Lesen genügen würde.

Das Ausführen des Read-Trajectory-File-Modus des ursprünglichen ORION-Codes führt bei etwas größeren Dateien zu einem Absturz des RACOON-cRIO. Selbst wenn per Hand ohne Datei-Lesen das Trajektorien-Array in den Code eingefügt wird, kommt es trotzdem immer noch zu starken Performance-Einbrüchen mit hoher CPU-Last und Überschreiten der maximalen Periodendauer. Das zeigt, dass das cRIO wohl sehr wahrscheinlich nicht mit den großen Datenmengen im laufenden Programm zurechtkommt.

Eine zuvor weniger beachtete Systemressource stößt beim Trajektorien-Dateien-Lesen nun ebenfalls an ihr Limit. Die relativ wenigen 128 MB Arbeitsspeicher des RACOON-cRIO werden sehr schnell von diesen Dateien gefüllt. Zu beachten ist hier auch, dass Trajektorien-Dateien im Arbeitsspeicher wesentlich mehr Platz benötigen als auf einer Festplatte. Die Ursache hierfür konnte leider nicht festgestellt werden.

4.1.3.1 Lösungsansätze

Der wohl wichtigste Schritt beim Erstellen einer effizienteren Trajectory-Follow-Funktion ist das Auslagern des Datei-Lesens aus der Control Loop. Wie bereits erwähnt ist ein Festplattenzugriff ein relativ langsamer, ressourcenintensiver Prozess, der auf keinen Fall mit zeitkritischen Prozessen zusammen in einer Schleife ausgeführt werden sollte. Es gibt zwei prinzipielle Ansätze für diesen Schritt: Parallelisierung und Sequenzierung.

Mit Parallelisierung sind ein gleichzeitiges Lesen und Nachfahren der Trajektorien-Werte gemeint. Die Idee ist, dass immer nur ein begrenzter Teil der Trajektorien-Datei ausgelesen wird, und so sich immer nur eine begrenzte Menge an Daten im Arbeitsspeicher befindet. Voraussetzung für diesen Ansatz ist ein ausreichend schnelles Lesen, sowie genügend freie Systemressourcen, um dies zu ermöglichen. Der größte Vorteil dieses Ansatzes ist der, dass gelesene Dateien theoretisch unendlich groß sein können.

Sequenzierung, sprich das Lesen vor dem Nutzen der Daten, wurde im Prinzip im ORION-Code schon umgesetzt. Allerdings soll bei diesem Ansatz nun strikter getrennt werden und das Daten-Einlesen vor dem Ausführen der Control Loop stattfinden.

Da es bei diesem Ansatz im Prinzip keine zeitlichen Beschränkungen gibt, eignet er sich gut für Systeme mit niedriger Leistungsfähigkeit. Der große Nachteil von diesem Ansatz ist die Beschränkung der lesbaren Dateigröße durch den begrenzten Arbeitsspeicher.

Im Prinzip findet beim Parallelisieren eine räumliche, beim Sequenzieren eine zeitliche Trennung statt.

Wie bereits beim Logging kann auch beim Lesen von Trajektorien-Dateien von einer Binär-Version von diesem Gebrauch gemacht werden, um ein effizienteres Auslesen zu ermöglichen. Der Nachteil ist auch hier, dass Textdateien vor der Benutzung transformiert werden müssen.

Möglich ist auch ein komplett neues Konzept für das Nachfahren vordefinierter Trajektorien. Beim aktuellen Konzept wird alle 10 ms ein genauer Wert in Steps oder Meter definiert, zu dem die einzelnen Achsen fahren. Dadurch entsteht auch für im Grunde einfache Trajektorien über eine längere Strecke eine große Menge an Daten. Ein anderer Ansatz wäre beispielsweise, nur die absolute Zielposition einer Bewegung zusammen mit der Zeit anzugeben, wie lange dieses Ziel gültig ist. Bei dieser Variante müssten dann bei einer Fahrtzeit von 30 Sekunden statt 3000 mal 14 Zahlen nur 2 Zahlen angegeben und ausgelesen werden. Möglicherweise ist diese Variante aber nicht ganz so präzise wie das aktuelle Konzept. Außerdem ließe sich das *Recording*-Konzept wie am Ende dieses Kapitels beschrieben, wenn überhaupt, nur mit deutlich mehr Aufwand umsetzten.

4.1.3.2 Implementierung

Als Strukturmodell wurde die Sequenzierung gewählt. Grund hierfür war der, dass sich eine Parallelisierung schlichtweg nicht umsetzen ließ. Bei einer größeren, auf einmal gelesener Datenmenge war die erforderliche Ausführzeit trotz Verwendung eines FIFOs als Puffer zu lange. Bei kleiner werdenden Datenmengen stieg die CPU-Auslastung bis 100% an, ohne dass davor jedoch das Problem des zu langsamen Auslesens gelöst wurde. Die Verwendung von Binär-Dateien würde hier weiterhelfen, womöglich ein paralleles Einlesen sogar ermöglichen. Paralleles Einlesen kann aber nur umgesetzt werden, wenn eine bestimmte Zeile angegeben werden kann, ab der in einer Datei gelesen werden soll. Dies ist beim Einlesen von Binärdateien jedoch nicht möglich. Stattdessen lässt sich immer nur vom Anfang der Datei anlesen. Es gibt zwar die Möglichkeit, über Referenz-Manipulation einen Versatz des Startpunktes des Lesens in Bit zu verändern. Da jedoch je nach Inhalt eine Zeile eine unterschiedliche Bit-Länge besitzt, war es nicht möglich, ein gezieltes Auslesen von Binärdateien umzusetzen.

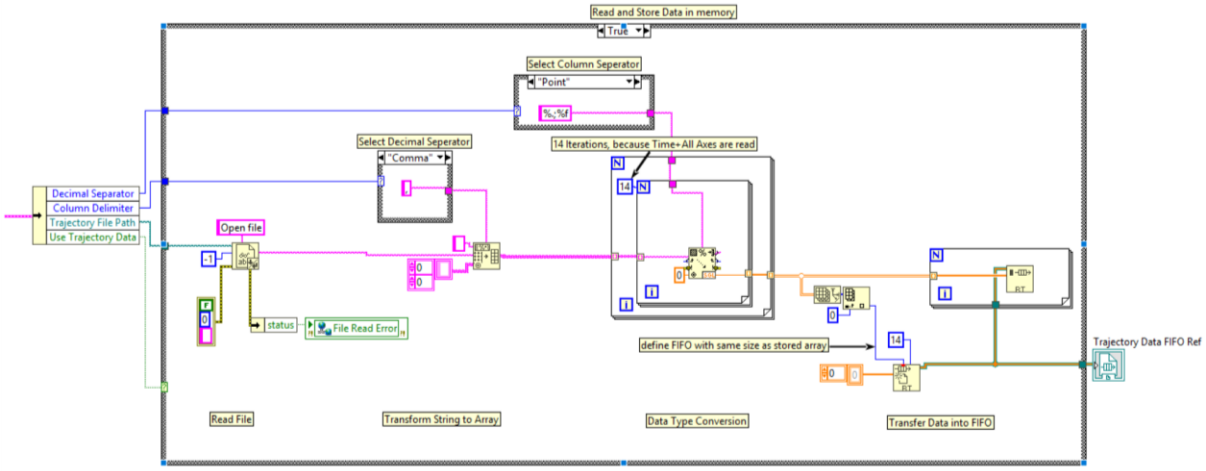


Abb. 4–6: Trajektorien-Datei lesen

Der neue Betriebsmodus wurde zeitlich in zwei Teile geteilt, Lesen und Ausführen. Der Prozessteil zum Einlesen der Trajektorien-Daten außerhalb von der Control Loop ist in Abbildung Abb. 4–6 dargestellt. Der Benutzer kann vor dem Start der Control Loop mittels des booleschen Wertes *Use Trajectory Data* angeben, ob Daten eingelesen werden sollen oder nicht. Entsprechend wird dieser Prozess, der sich innerhalb einer Case-Struktur befindet, dann ausgeführt oder nicht.

Zuerst wird die Datei geöffnet und gelesen. Der Aufbau der Datei soll dabei gleich sein wie die ersten 14 Spalten beim Logging (siehe Tab. 4–1). Sprich es muss die Zeit in 10 Millisekunden-Schritten angegeben werden, sowie die Position der einzelnen Achsen. Fehler beim Lesen werden mittels der Netzwerk-Umgebungsvariablen *File Read Error* an den Host-PC kommuniziert. Der ausgegebene String wird unter Berücksichtigung des vom Benutzer gewählten Dezimalseparator (*Decimal Separator*) in einen String-Array umgewandelt, bei dem jeder Zahl ein Element im Array zugeordnet ist. Danach erfolgt die Datentypumwandlung in den numerischen Datentyp *Single*-Array, ebenfalls unter Berücksichtigung von Benutzereingaben, diesmal zum angegebenen Zeichen für Spaltentrennung (*Column Delimiter*). Hierbei werden auch sämtliche Spalten nach Spalte 14 abgeschnitten, sprich Zeitangaben und Ortsangaben der einzelnen Achsen, sodass etwa beim Nachfahren aufgezeichneter Trajektorien, wie am Ende des Kapitels beschrieben, keine unnötigen zusätzlichen Daten übernommen werden. Single-Zahlen in LabVIEW sind Fließkommazahlen mit einfacher Genauigkeit, sprich einer Größe von 32 Bit [15].

Anschließend wird dieses numerische Array zeilenweise in ein Echtzeit-FIFO eingelesen. Dieses FIFO dient als ein von überall auf dem cRIO einfach zugänglicher Speicherplatz der Trajektorien-Daten. Das FIFO passt sich automatisch an das eingelesene Array an, indem bei dessen Definierung die Länge dieses Arrays verwendet wird. Die Referenz zu diesem Speicher-FIFO wird als letzter Schritt in den Mode Handler übertragen, welcher sich in der Control Loop befindet.

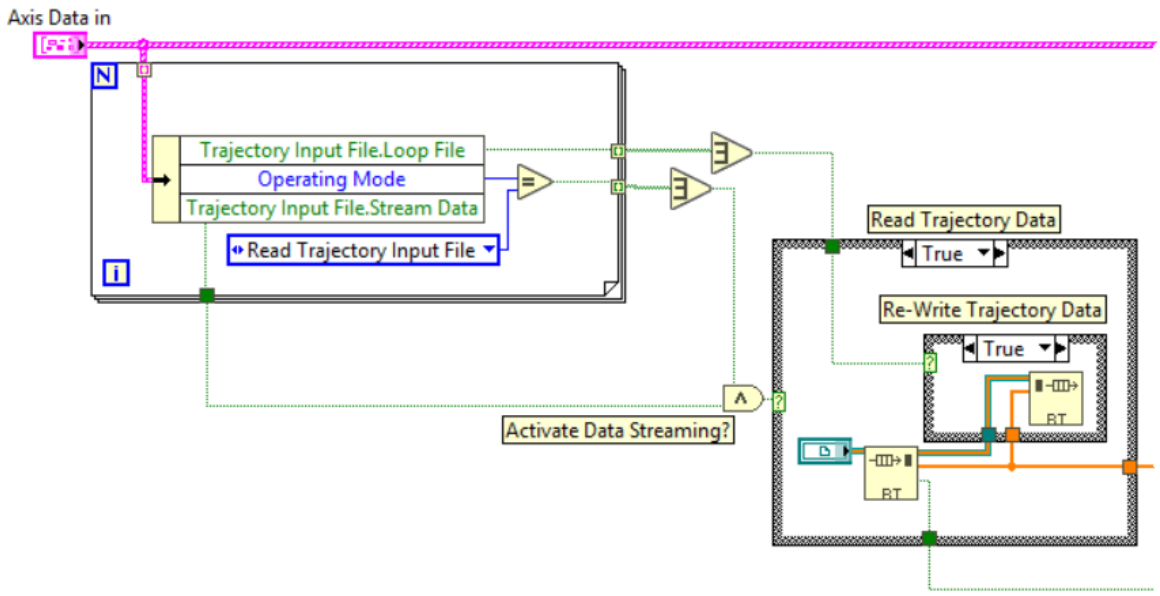


Abb. 4–7: Streaming der Trajektorien-Datei

Das Ausführen findet im Mode Handler statt. Noch vor der Schleife für alle einzelnen Achsen findet das Streaming statt, in Abb. 4–7 zu sehen. Hier wird überprüft, ob der Benutzer sowohl den booleschen Wert *Stream Data* auf wahr, als auch bei mindestens einer Achse den Betriebsmodus auf *Read Trajectory File* geschalten hat. Ist das der Fall, dann wird eine Zeile des gespeicherten Trajektorien-Arrays gelesen und in der nachfolgenden For-Schleife weiterverwendet.

Nach dem Lesen eines Elements, in diesem Fall eine Arrayzeile, wird dieses aus einem FIFO gelöscht. Wenn nun jedoch der Boole-Wert *Loop File* vom Benutzer auf TRUE geschalten wird, wird der ausgelesene Wert wieder in das FIFO geschrieben. Da immer der jüngste Wert in einem FIFO hintenangestellt wird, ergibt sich über viele Iterationen ein Kreislauf der Array-Zeilen und somit ein unendliches Streamen der Trajektorien-Daten. Bei Verwendung von dieser Funktion sollte jedoch darauf geachtet werden, dass Start- und Endwert der einzelnen Achsen gleich sind.

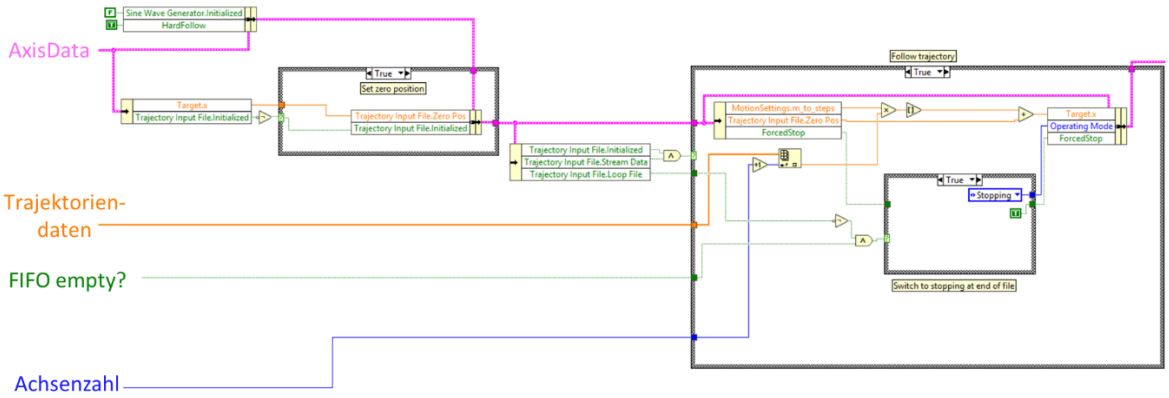


Abb. 4–8: Lesen und Nachfahren von Trajektorien

In jeder Iteration wird nun mithilfe des sich in Abb. 4–8 befindlichen Codes eine Zeile ausgelesen. Dies geschieht in der nachfolgenden For-Schleife, in der achsenweise Positions- und Geschwindigkeitswerte bestimmt werden. Die Schleife ist der Hauptteil des Mode Handler. Für jede Achse wird der entsprechende Positionswert aus dem Array abgelesen, mittels des entsprechenden m_to_steps-Faktor umgerechnet und die so erhaltenen Motor-Steps mit einem definierten Nullpunkt *Zero Pos* verrechnet. Beim Aktivieren von *Stream Trajectory Data* wird bei der jeweiligen Achse der aktuelle Positionswert intern als dieser Nullpunkt *Zero Pos* festgehalten. Die Trajektorie wird also immer bezüglich von diesem Anfangswert abgefahren. Die so erhaltenen Werte werden im nachfolgenden *Setpoint Handler* in SplineInfo-Daten umgewandelt. Wird das Ende des Trajektorien-Arrays im FIFO erreicht, wechselt das cRIO automatisch in den Stopping-Modus.

4.1.3.3 Ergebnisse

Der fertig implementierte Betriebsmodus *Read Trajectory Data* weist alle grundlegenden Fähigkeiten auf, wie auch der des ORION. Allerdings gibt es Einschränkungen. So ist die Größe und somit die Dauer der Trajektorien-Daten begrenzt. Auch muss bereits vor dem Ausführen der Control Loop gewartet werden, bis eine Datei eingelesen wurde. Allerdings wurden auch einige Eigenschaften verbessert. Laut der ursprünglichen Arbeit [4], in der diese Funktion das erste Mal implementiert wurde, gab es beim ORION teils kleinere Verzögerungen im Programmablauf beim Abfahren von Trajektorien. Diese sind beim Austesten dieser neuen Version nicht aufgetreten. Neu zum ORION ist auch das Auswählen der Dezimal- und Spaltenseparatoren. Außerdem können nun automatisch Trajektorien-Dateien in einer Endlos-Schleife wiedergegeben werden.

Der nun eben implementierte Betriebsmodus erlaubt zusammen mit dem neuen Logging ein neues Feature: *Recording*, das Aufzeichnen und Nachfahren von Trajektorien. Da beim Logging unter anderem die genaue Achsenposition bei jeder einzelnen Iteration aufgezeichnet wird, kann nun die so aufgezeichnete Fahrt nach einem Umwandeln und Einlesen in den Arbeitsspeicher nachgefahren werden. Wichtig ist hier, dass bei dem in Abschnitt 4.2.3.1 beschriebenen Startfenster folgende Eingaben gemacht werden:

- Decimal Separator: Comma
- Column Separator: Space

Auch sollte beim Aufzeichnen die Beschränktheit des Arbeitsspeichers beachtet werden.

4.1.4 Sine Sweep und Manuelle Steuerung

Der *Sine-Sweep*-Betriebsmodus erlaubt dem Anlagenoperator benutzerdefinierte, sinusförmige Bewegungen zu erstellen. Festgelegt werden kann dabei sowohl die Amplitude als auch die Frequenz der Bewegung.

Der *Manual*-Betriebsmodus erlaubt ein manuelles Steuern der Achsen über die Eingabe von Zielwerten in die *Target*-Bedienfelder. Theoretisch war dies bereits über Umwege auch bei der alten Version der RACOON-Steuerung möglich, nun wurde aber ein dafür vorhergesehener Betriebsmodus eingerichtet.

Das Implementieren von diesen beiden Betriebsmodi verlief ohne Probleme, der vorhandene ORION-Code konnte ohne Änderungen einfach in den neuen RACOON-Code eingefügt werden.

4.2 Graphical User Interface

4.2.1 Ausgangslage

Die grafische Benutzeroberfläche, oder *Graphical User Interface* im Englischen, dient als Mensch-Maschine-Schnittstelle zur Bedienung der Anlage durch einen Benutzer. In LabVIEW wird dieses Fenster auch als *Frontpanel* eines VI bezeichnet.

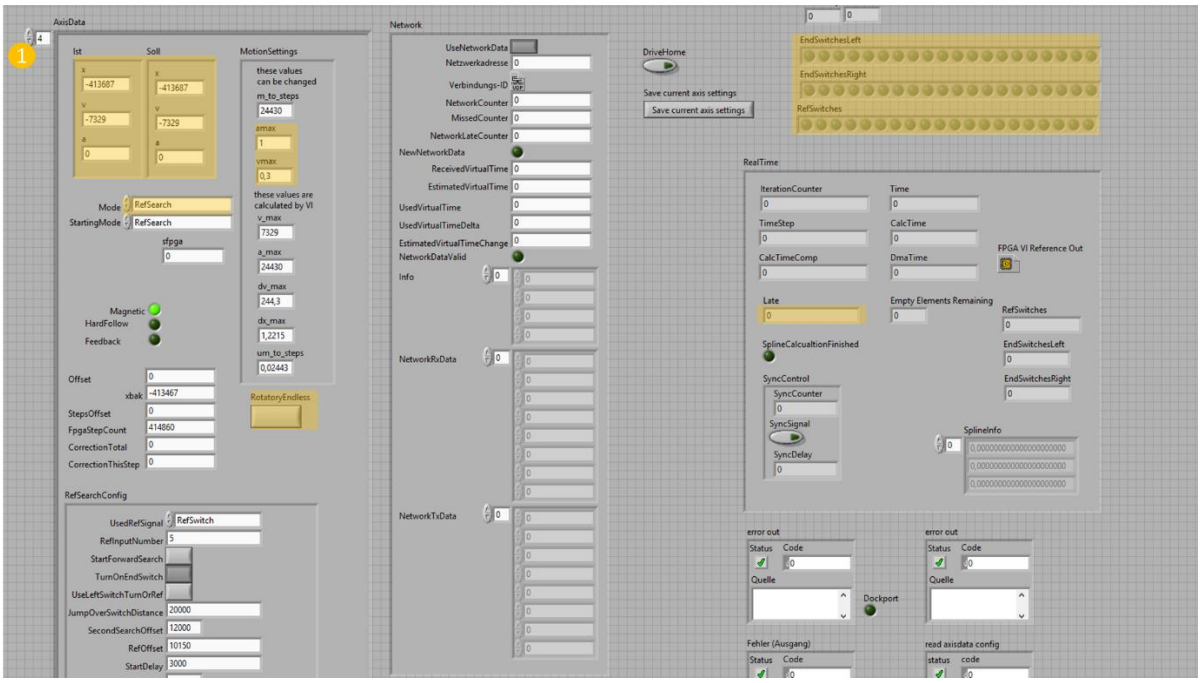


Abb. 4–9: alte Benutzeroberfläche des RACOON-Labs

Ein Ausschnitt des ursprünglichen GUI ist in Abb. 4–9 zu sehen. In dieser Abbildung wurden alle Bedienelemente orange eingefärbt, die im normalen Betrieb benötigt werden und auch im neuen GUI in derselben oder ähnlichen Form vorhanden sind. Auffällig ist, dass der Großteil der Elemente keinen Nutzen für den Bediener hat. Dadurch gehen die relevanten Steuerelemente in der Masse unter, die Oberfläche wirkt überladen.

Es ist immer nur eine Achse auf einmal steuerbar. Um verschiedenen Achsen zu überwachen und Befehle zu erteilen, muss in dem mit einer orangen 1 markiertem Feld die entsprechende Achse ausgewählt werden. Dadurch wird das Bedienen umständlich,

da zur Eingabe eines einfachen Befehls, etwa das Fahren einer Achse zu einem bestimmten Punkt, deutlich mehr Eingaben erforderlich sind als der bloße Befehl.

Ein wesentlicher Störfaktor beim Bedienen der alten Version sind außerdem Verzögerungen von teils mehreren Sekunden und Nichterkennen von Eingaben. Diese Effekte treten vor allem bei längerem Ausführen des Steuerprogramms auf. Sie werden vermutlich von den meisten Benutzern als sehr störend empfunden werden. Bei Unkenntnis über diese Eigenschaften kann es zusätzlich leicht zum mehrfachen Eingeben eines Befehls kommen.

Ein Grund für die fehlende Benutzerfreundlichkeit könnte sein, dass beim ursprünglichen Code noch keine direkte Steuerung per GUI am Host-PC angedacht war. Stattdessen wurde womöglich bis dato nur die Steuerung via Netzwerk mit dem restlichen RACOON implementiert.

Neben der Verbesserung der Performance und dem Hinzufügen neuer Funktionen soll beim neuen Steuercode nun auch diese Benutzeroberfläche überarbeitet werden. Sie soll laut Aufgabenstellung möglichst intuitiv und benutzerfreundlich sein. Es müssen auch alle in der Anforderungsliste angegebenen Eigenschaften erfüllt sein.

4.2.2 Konzepte

In diesem Abschnitt werden nun einige Konzepte betrachtet, die zur eine übersichtlicheren Benutzeroberfläche beitragen können.

4.2.2.1 Darstellung der Achsen

Für ein leichteres Steuern und Überwachen sollten möglichst viele Achsen gleichzeitig sichtbar sein, um zusätzliches Hin- und Herschalten zu vermeiden. Der daher wohl am besten geeignete Ansatz ist also das gleichzeitige Anzeigen aller Achsen. Allerdings wird dafür sehr viel Platz benötigt. Beim ORION-Lab wurde bereits dieser Ansatz gewählt, hier nahm schon bei nur 6 Achsen die Benutzeroberfläche den gesamten Bildschirm ein. Bei einer Umsetzung dieses Konzepts muss daher sehr viel Platz eingespart werden.

Eine Alternative hierzu ist, Achsen beispielsweise nach ihrer Zughörigkeit zu den Anlagenteilen, Servicer, Target und Strahlungssimulatoren, zu clustern. Der Benutzer kann dann zwischen den einzelnen Clustern wählen, und kann dadurch immerhin die Teilbereiche auf einmal steuern und überwachen. Dieses Konzept lässt sich gut mit dem im Abschnitt 3.2 erwähnten Ansatz der selektiven Achsansteuerung verbinden.

4.2.2.2 Angepasste Bedienfelder

Damit ist gemeint, dass für jede Achse ein individuell gestaltetes Bedienfeld ermöglicht wird. Sowohl beim alten RACOON-Code als auch beim ORION-Code sind alle Bedienfelder in einem Array zusammengefasst. Dadurch muss jedes Feld die gleichen Eigenschaften besitzen, lediglich die Werte innerhalb eines Bedienelements sind variabel. Durch das Herauslösen aus dieser strikten Struktur können für jede Achse einzeln Bedienelemente hinzugefügt oder entfernt werden, individuelle Einheiten eingefügt und das Erscheinungsbild frei verändert werden. Weitere daraus folgende Vorteile sind

unter anderem Platzeinsparungen und Vermeiden von Verwirrung durch funktionsunfähige Bedienelemente.

4.2.2.3 Steuerbefehle für alle Achsen

Dieses Konzept fügt dem GUI ein zusätzliches Bedienfeld hinzu, mit dem alle Achsen gleichzeitig gesteuert werden können. So muss beispielsweise für das Ändern des Betriebsmodus aller Achsen nur einmal statt dreizehnmal geklickt werden. In das ORION-Lab ist dieses Feature bereits erfolgreich implementiert.

4.2.2.4 Apply Button

Bei der ursprünglichen RACOON-Steuerung werden Eingaben in Bedienfelder nach dem Bestätigen mit der Enter-Taste sofort ausgeführt. Ein gleichzeitiges Fahren zweier Achsen ist somit nicht möglich. Abhilfe kann hier ein zusätzlicher Apply-Button schaffen. Erst durch ein Bestätigen mittels Klicken auf diesem werden dann alle zuvor eingegebenen Befehle ausgeführt. Allerdings fungieren einige Bedienelemente des GUI sowohl als Steuer- als auch als Anzeigeelement, beispielsweise das Feld *Operating Mode*. Ohne Apply-Schalter macht diese Eigenschaft keine Probleme, da Änderungen sofort als Ist-Zustand übernommen werden. Durch Einführung eines Apply-Buttons geht die Anzeige-Eigenschaft verloren. Es kann beim Ändern z.B. des Operating Modes zu einer Diskrepanz zwischen Anzeigewert und Ist-Wert der Anlage, und dadurch zu Fehlern und Unklarheiten beim Bedienen kommen. Abhilfe lässt sich durch eine Aufteilung dieser Hybrid-Elemente schaffen. Allerdings wird dadurch der von diesem beanspruchten Platz verdoppelt.

4.2.2.5 Caution and Advisory Display

Über diese Anzeige soll ein Großteil der Informationen über die Anlage an den Benutzer kommuniziert werden. Die besondere Eigenschaft dieser Anzeige ist dabei, dass Informationen nur sichtbar werden, wenn eine Situation dies erfordert. Die normalen LabVIEW-Anzeigeelemente sind hingegen immer sichtbar und können nur ihren Wert ändern, etwa indem sie eine andere Zahl anzeigen oder aufleuchten. Durch Implementierung dieses Caution- and Advisory-Display (CAD) lässt sich so zum einen viel Platz auf der Benutzeroberfläche einsparen. Zum anderen wird die Oberfläche dadurch deutlich übersichtlicher.

4.2.2.6 Setup Window

Durch die Restrukturierung und Implementierung neuer Funktionen ist die Bedienung der Anlage komplexer geworden. Um diesem entgegenzusteuern kann ein Startfenster eingeführt werden, welche alle erforderlichen Konfigurationen vor dem Start der Anlage bündelt.

4.2.2.7 Automatisches Aufrufen

Zusätzlich zum Setup Window kann auch noch das Aufrufen von Prozessen und VIs an den verschiedenen Orten, auf dem der Code laufen soll, automatisiert werden. Auf einem einzelnen Gerät geschieht das automatisch, bei mehreren Geräten müsste der Benutzer dies von Hand tun.

4.2.2.8 Interaktive GUI

Durch das Auslagern der Berechnungen zur Benutzeroberfläche auf den Host-PC können nun auch deutlich komplexere Operationen in den Programmablauf eingebaut werden. Ein Beispiel sind sogenannte *Property* Nodes, Invoke *Nodes* und *Event Structures*. Diese ermöglichen etwa das Verändern von Programmelementen und somit auch von Anzeigeelementen auf dem GUI bei bestimmen Ereignissen. Dies erlaubt das Gestalten einer interaktiven GUI, bei der beispielsweise Elemente und zusätzliche Fenster nach Belieben angezeigt, deaktiviert oder versteckt werden können. Eine andere Möglichkeit ist das Signalisieren oder Hervorheben einer Warnung durch Änderung der Farbe eines Steuerelements. Für einige der oben erwähnten Konzepte kommen solche Methoden zum Einsatz. Die Gestaltungsmöglichkeiten gehen jedoch noch weiter über das aktuell Implementierte hinaus.

4.2.3 Implementierung

Die Umsetzung der eben aufgezählten Konzepte wird nun im Folgenden erläutert.

4.2.3.1 Setup Window und Automatisches Aufrufen

Das Programm sowie dessen Steuerung ist wegen der eingeführten Neuerungen wesentlich komplexer geworden. Das Setup Window soll erforderliche Aktionen und Einstellungen, die vor dem Start durchzuführen und festzulegen sind, bündeln oder ganz automatisch durchführen. Ein wichtiger Bestandteil dieses Startfenster-Konzepts ist dabei das automatische Aufrufen von VIs und Prozessen. Ein vereinfachter Programmablauf ist in Abbildung Abb. 4–10 dargestellt. Quadrate stellen Zustände des Programmcodes dar, Pfeile bedeuten Signalfluss. Mit Ausnahme der ersten Kommunikation mittels Invoke Node findet die Kommunikation zur Ablaufsteuerung zwischen den Geräten mit Netzwerk-Umgebungsvariablen statt. Die Namen dieser sind in der Abbildung über den jeweiligen Pfeilen zu sehen. Unterteilt wurde die Abbildung in vier Bereiche. In Prozesse, die auf dem Host-PC und auf dem cRIO ablaufen, sowie diese nochmals unterteilt in *Preparing*-Prozesse und in die des *Main Program,* in dessen Zuständen sich die Anlage zum Großteil der Zeit befindet.

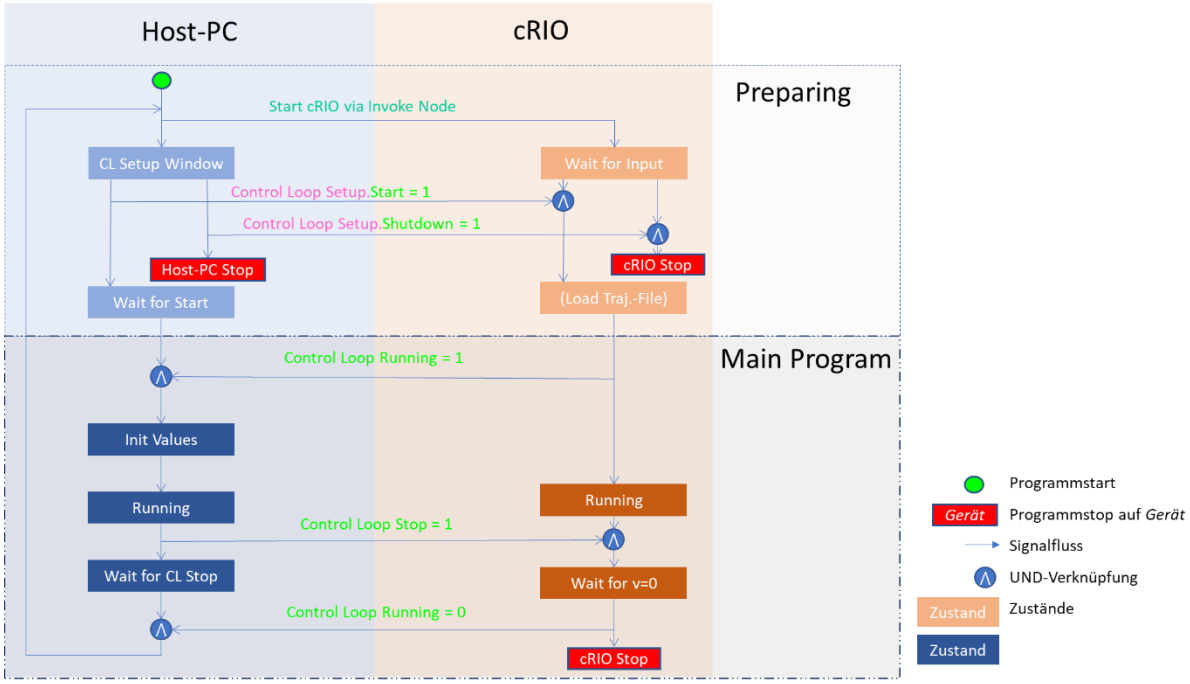


Abb. 4–10: automatischer Programmablauf auf Host-PC und cRIO

Der Programmablauf mit seinen einzelnen Zuständen wird nun chronologisch vorgestellt. Der Ablauf wird nach jedem Durchlaufen des Main Program wiederholt, um beispielsweise Änderungen der Starteinstellungen vorzunehmen, oder um eine neue Trajektorien-Datei zu laden. Erreicht wird dies mittels einer den gesamten Code umgebende While-Schleife, zu sehen als grauer Kasten um den restlichen Code in Abb. 4–11. Sie und damit das Programm wird erst beendet, wenn im Startfenster *Shutdown* gewählt wird. In Abb. 4–11 grün markiert sind alle Setupfunktionen, orange markiert sind die zentralsten Programmteile in der GUI Loop und rot sind alle Funktionsblöcke, die zum Beenden des gesamten Programms dienen.

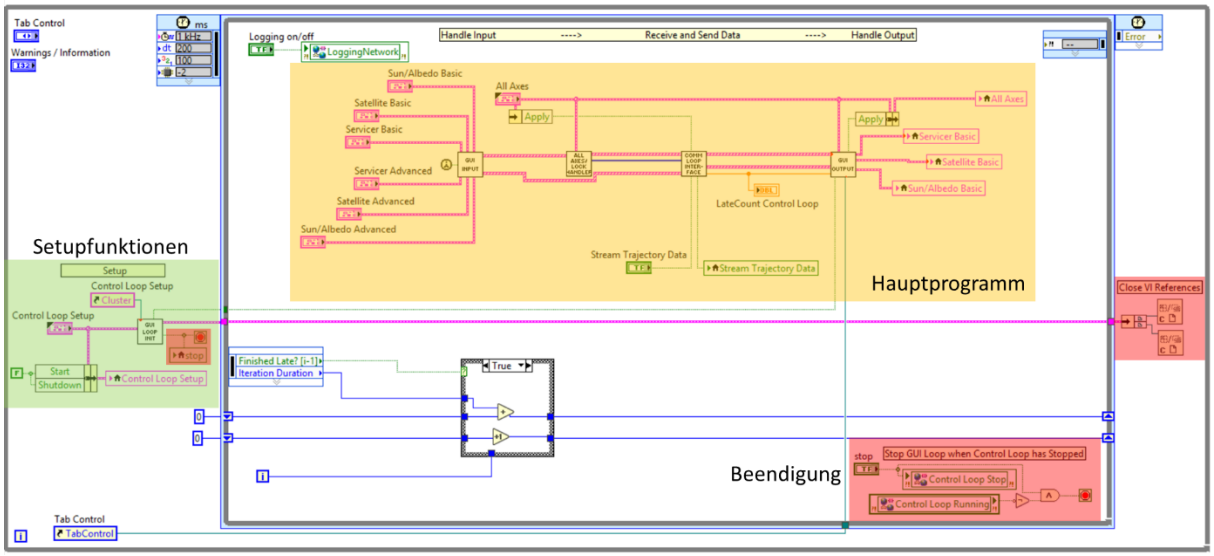


Abb. 4–11: Übersicht GUI Loop

Dank des automatischen Aufrufens muss vor dem Start zur Steuerung der Anlage lediglich das VI *RACOON GUI* geöffnet werden. Direkt nach dem Programmstart erscheint ein Dialogfenster, das Setup Window, zu sehen in Abbildung Abb. 4–12.

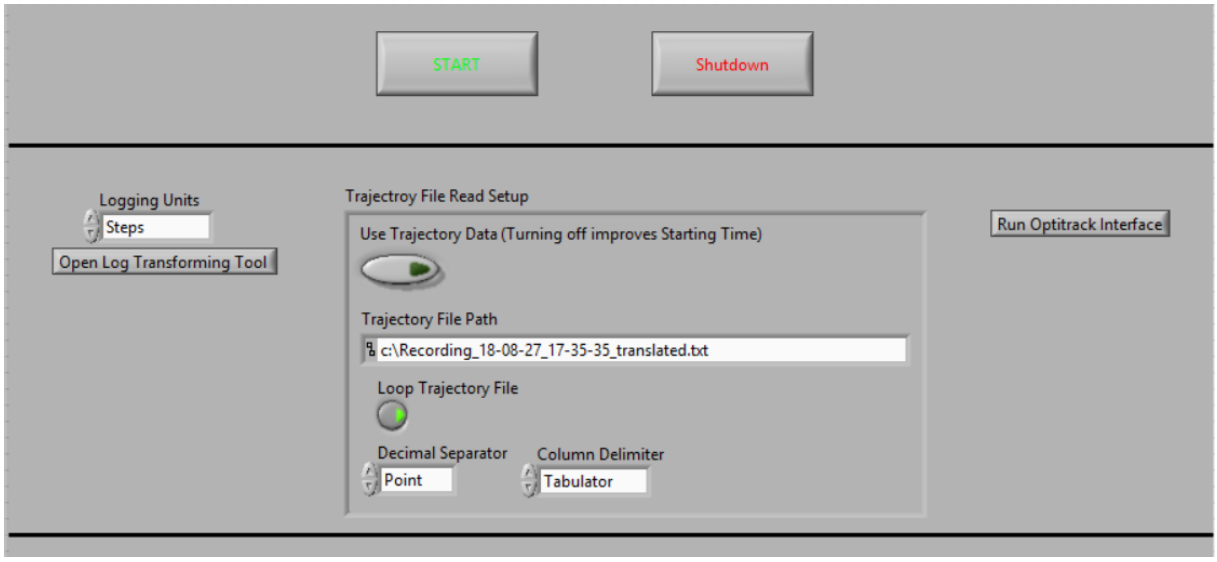


Abb. 4–12: Setup Window

Mithilfe von diesem können die Steuerung gestartet oder das Programm beendet, das Transforming Tool oder das Optitrack Interface aufgerufen oder Einstellungen zum Logging und Trajectory Follow geändert werden.

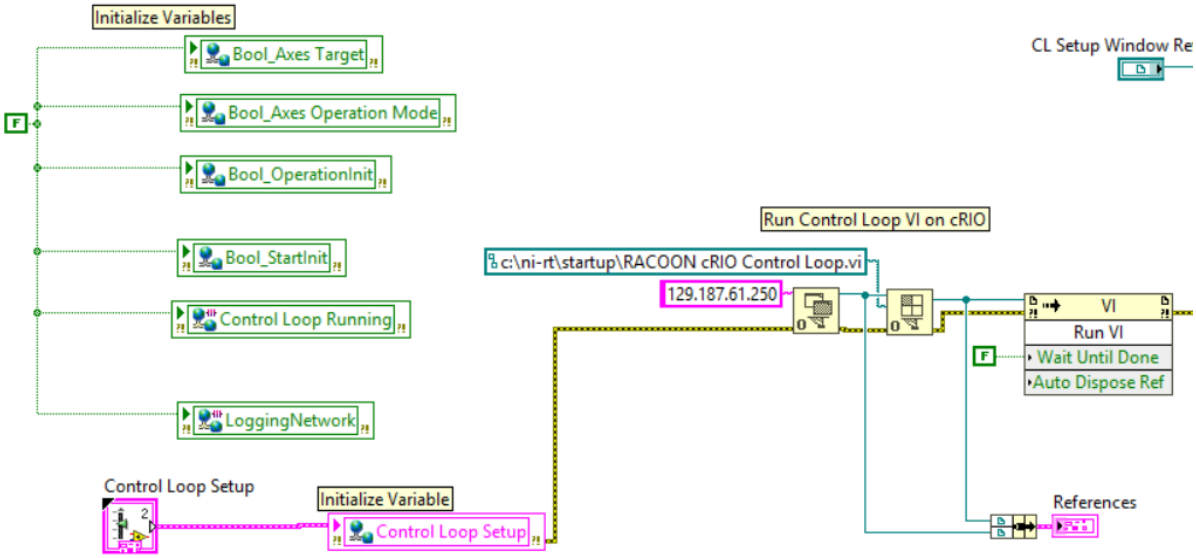


Abb. 4–13: GUI Loop Init – Variableninitialisierung und Control Loop Aufruf

Der dem Fenster zugrunde liegende Code ist im SubVI *GUI Loop Init* gespeichert, und ist in Abb. 4–13 zu sehen. Das SubVI wird gleich zu Beginn des Hauptprogramms *RACOON GUI* automatisch ausgeführt. Im ersten Schritt werden einige Umgebungsvariablen initialisiert, die hauptsächlich die Kommunikation zwischen cRIO und Host-PC steuern. Danach wird mittels Invoke Node das Hauptprogramm auf dem cRIO gestartet, RACOON *cRIO Control Loop*. Hier sollte darauf geachtet werden, dass der

angegebene Pfad mit dem tatsächlichen Ort des VIs im Verzeichnis des cRIO übereinstimmt. Kontrolliert werden kann das mit einer FTP-Verbindung. Das Ausführen des VIs entspricht dem Erreichen des Zustands *Wait for Input* auf dem cRIO in Abb. 4–10. Abläufe auf dem cRIO werden später erläutert.

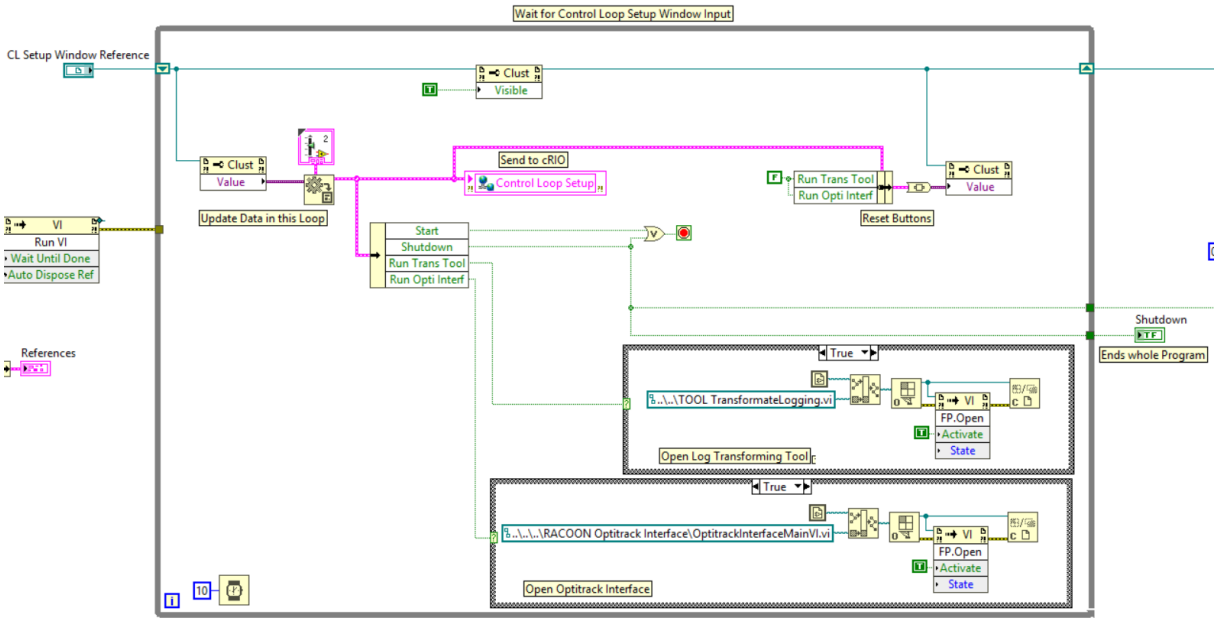


Abb. 4–14: GUI Loop Init – Wait for Control Loop Setup

Die nun nachfolgende Schleife, zu sehen in Abb. 4–14, dient zur Verarbeitung von Eingaben in das Startfenster. Im Ablaufschaubild Abb. 4–10 entspricht sie dem Zustand *CL Setup Window*. Das Setup Window wird in dieser Schleife mittels einer Property Node auf sichtbar geschalten. Genau genommen ist dieses Starfenster einfach nur ein Cluster, der über die normale Steuerung eingeblendet wird, und in dem die einzelnen Bedienelemente des Startfensters untergebracht sind. Auch der Zugriff auf die Steuerelement-Werte erfolgt mittels Property Node. So können in jeder Iteration diese Elemente ausgelesen werden, ohne dass das Haupt-VI *RACOON GUI* erneut aufgerufen werden muss. Je nach Eingabe werden entsprechende Befehle ausgeführt. So können über einen Klick des Benutzers die Boole-Werte der beiden Case--Strukturen geändert und so mithilfe von Invoke Nodes das VI zur Logging-Umrechnung sowie das Optitrack-Interface aufgerufen werden. Damit diese nur einmal pro Klick geöffnet werden, werden diese booleschen Werte nach jeder Iteration wieder auf FALSE geschalten. Zum Aufrufen werden relative Pfade verwendet, sprich so lange die Struktur der Programmordner nicht verändert wird, kann das Programmverzeichnis auf einer beliebigen Stelle am PC gespeichert werden. Es empfiehlt sich jedoch, das gesamte Verzeichnis auf einer möglichst hohen Ebene, beispielsweise direkt unter *C:* zu speichern, da LabVIEW mit sehr langen Programmpfaden Probleme bekommen kann.

Die Werte, die zur Steuerung von Prozessen auf dem cRIO dienen, sprich *Logging Units* und alle Werte in *Trajectory File Read Setup,* werden mithilfe der Netzwerk-Umgebungsvariable *Control Loop Setup* an das cRIO übertragen. In dieser Variablen sind auch die beiden Boole-Werte *Start* und *Shutdown* enthalten. Sowohl Start als auch Shutdown beenden die Schleife, wenn sie TRUE werden.

Um Systemressourcen nicht unnötig zu belasten, wurden in dieser und den nachfolgenden Schleifen eine Wartezeit von 10 ms pro Iteration hinzugefügt.

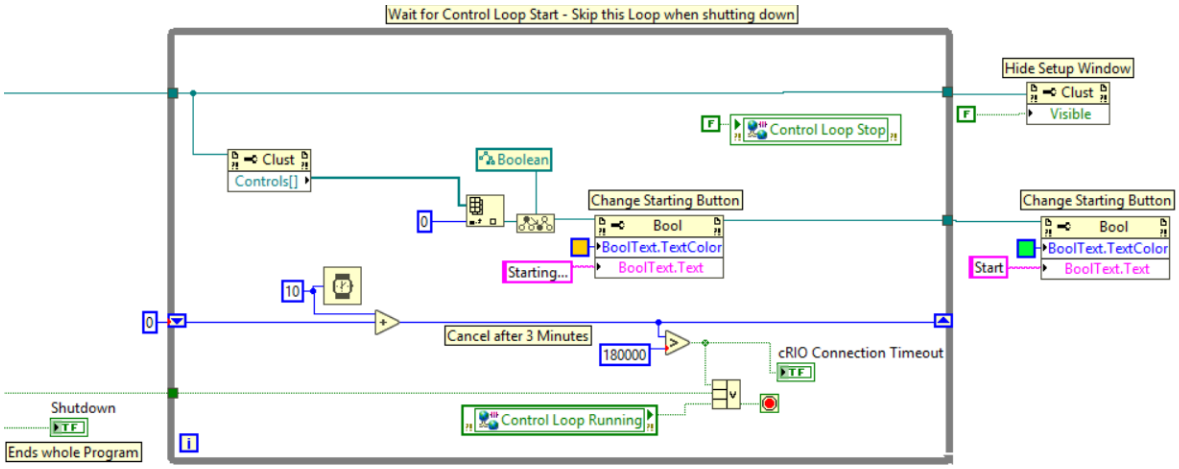


Abb. 4–15: GUI Loop Init – Wait for Control Loop Start

Die direkt nachfolgende Schleife entspricht dem Zustand *Wait for Start* auf der Host-PC-Seite. In dieser in Abbildung Abb. 4–15 zu sehenden Schleife gibt es drei Abbruchkriterien. Entweder die Netzwerk-Umgebungsvariable *Control Loop Running* gibt vom cRIO aus kommend das boolesche Zeichen, dass die Control Loop läuft. Dies führt im Anschluss dazu, dass das Programm und somit die GUI Loop ganz normal startet. Wurde jedoch in der vorherigen Schleife jedoch *Shutdown* ausgewählt, wird das gesamte Programm direkt nach einer Iteration beendet. Das dritte Kriterium ist Überschreiten einer Wartefrist von 3 Minuten. Dieser Timer wurde eingeführt, um bei Problemen mit der cRIO-Kommunikation oder Trajektorien-Einlese-Fehlern die Wartezeit zu begrenzen. Wegen des begrenzten Speicherplatzes werden Trajektorien-Dateien immer innerhalb dieser 3 Minuten eingelesen. Bei einem Überschreiten wird im Anschluss im Caution and Advisory Display (siehe Abschnitt 4.2.2.5) der Fehler *„cRIO Connection timed out“* angezeigt. Während die Schleife ausgeführt wird, wird mittels einer Property Node die Start-Button-Beschriftung geändert, um dem Benutzer zu signalisieren, dass seine Eingabe erkannt wurde.

Nach dem Beenden der Wait-for-Control-Loop-Schleife, unabhängig vom Auslöser, wird der Start-Button wieder in den Ausgangszustand zurückgesetzt und das Setup Window wird ausgeblendet.

Nachdem das VI *RACOON cRIO Control Loop* aufgerufen wurde, wird gleich als erster Schritt das SubVI *Control Loop Setup* ausgeführt. Die gleich zu Beginn des SubVI ausgeführte Schleife entspricht dem Zustand *Wait for Input* in der Übersicht (Abb. 4–10). In dieser wird die Umgebungsvariable *Control Loop Setup*, die zuvor in der Setup-Window-Schleife mit Benutzereingaben beschrieben worden ist, ausgelesen. Wird dabei der darin enthaltene Wert *Shutdown* als TRUE gelesen, wird die in der Switch-Case-Struktur enthaltene Invoke Node ausgeführt, welche das Haupt-VI auf dem cRIO beendet. Wird hingegen *Start* wahr, wird die Schleife beendet und das Programm weiter ausgeführt. In diesem Fall werden die aktuellen Werte dieser Iteration zu Logging und Trajektorien-Datei-Lesen an das restliche Programm weitergegeben.

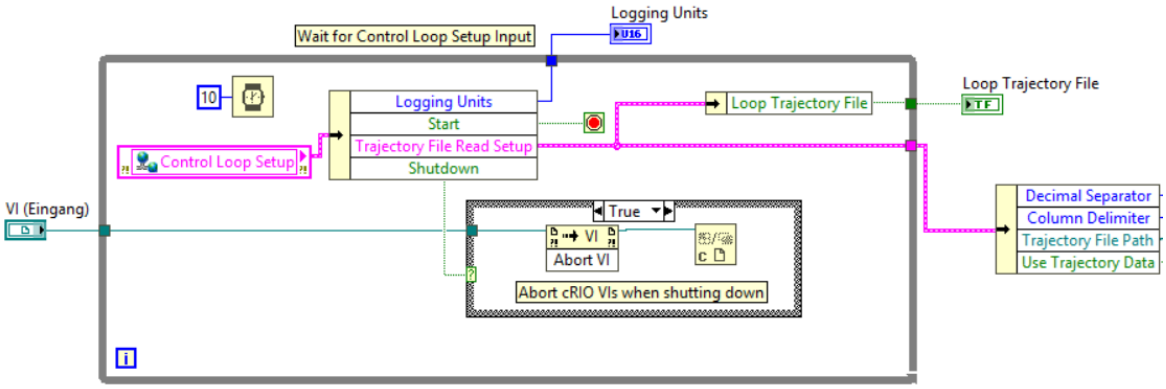


Abb. 4–16: Control Loop Setup

Falls vom Benutzer ausgewählt, wird nachfolgend zu dieser Schleife das in Abschnitt Abb. 4–16 beschriebene Lesen der Trajektorien-Datei ausgeführt. Nach diesem werden alle übrigen Anlagenwerte initialisiert und die Control Loop sowie die Logging Loop ausgeführt. Bei der ersten Iteration der Control Loop wird im SubVI *cRIO Control Loop Stop* der Wert von *Control Loop Running* auf wahr geschalten.

Der Zustand *Init Values* in der Übersicht (Abb. 4–10) entspricht der ersten Iteration der GUI Loop. In dieser wird die GUI-Loop initialisiert.

Zum Beenden des Programms muss der im GUI am unteren Rand platzierte *Stop*-Button gedrückt werden, siehe in der Abb. 4–17 links.

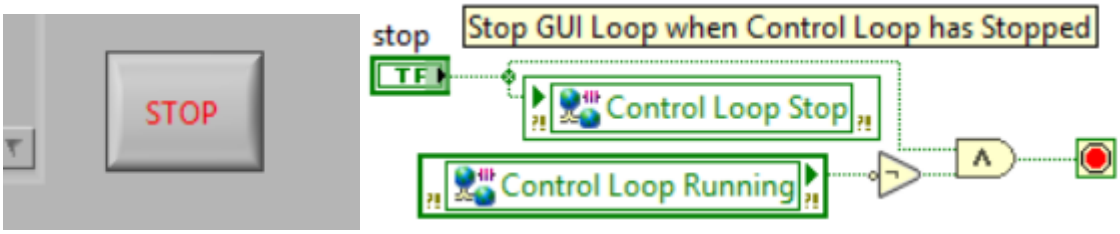


Abb. 4–17: Stop Button und Logik

Dies hat, wie in der oberen Abbildung rechts zu sehen, die Folge, dass die Shared Variable *Control Loop Stop* wahr wird. Vorbeugend wurde eine zusätzlich erforderliche Rückmeldung des cRIO eingeführt, ohne die die GUI Loop nicht beendet wird. Es könnte theoretisch ein Kontrollverlust über die Anlage entstehen, wenn durch die beendete Benutzeroberfläche keine Steuereingaben mehr möglich sind, durch einen Fehler in der Kommunikation allerdings der Code auf dem cRIO weiterläuft und die Motoren ansteuert.

Auf dem cRIO ist der Code zum Anlagenstopp im SubVI *cRIO Control Loop Stop* untergebracht (siehe Abb. 4–18).

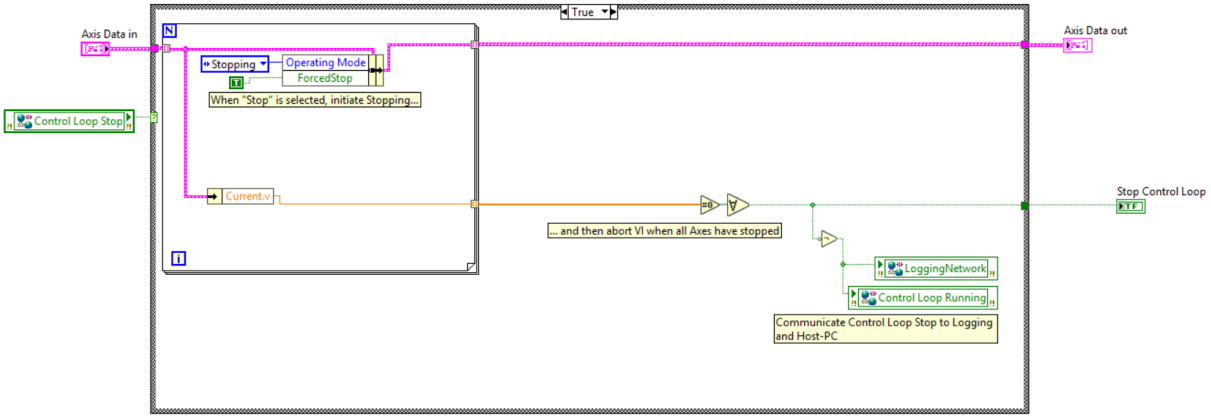


Abb. 4–18: Control Loop Stop

Erhält das cRIO über die Variable *Control Loop Stop* den Befehl zum Stoppen, werden alle Achsen in den *Stopping*-Modus geschalten. Erst wenn alle Achsen die Geschwindigkeit 0 erreicht haben, wird ein eventuell noch laufendes Logging mit der Shared Variable *LoggingNetwork* beendet, *Control Loop Running* auf FALSE geschaltet und anschließend die Control Loop beendet.

Control Loop Stop gleich TRUE hat außerdem das Beenden der Logging Loop zur Folge, sobald der Logging-Prozess abgeschlossen ist.

Wenn dann so auf dem Host-PC die GUI Loop beendet wurde, beginnt dank der den gesamten Code umgebenden While-Schleife der Ablauf von neuem mit dem Anzeigen des Setup Windows.

4.2.3.2 Achsensteuerung

Dank anderer Maßnahmen zur Platzeinsparung konnte das gleichzeitige Anzeigen aller Achsen ermöglicht werden. Um das GUI möglichst übersichtlich zu gestalten, sind die jeweils zu einem Anlagenteil gehörenden Achsen beschriftet und in einem Cluster zusammengefasst. Auf ein Zusammenfassen mittels eines Arrays wurde verzichtet, da sonst die ebenfalls implementierten angepassten Bedienfelder nicht realisierbar gewesen wären.

Für das Programmieren ist es jedoch sehr hilfreich, wenn die Bedienfeldbefehle in einem Array vorliegen, da für alle Achsen sehr oft die gleichen Operationen durchgeführt werden müssen, und beispielsweis For-Schleifen nur für Arrays umsetzbar sind. In einem Array müssen alle Elemente jedoch genau den gleichen Datentyp, beziehungsweise in diesem Fall die einzelnen Cluster dieselben Inhalte haben. Sprich um individuelle Bedienfelder umsetzen zu können, müssen die Daten aus den Bedienfeldern erst vereinheitlicht werden. Dies geschieht im SubVI *GUI Input*. In diesem werden die einzelnen Cluster zuerst in den Datentyp Variant umgewandelt. Dies erlaubt dann das Erstellen eines Clusters, sodass ab hier alle Operationen in For-Schleifen ablaufen können. Für die Basic Controls ist dieser Vorgang in Abb. 4–19 dargestellt. Für die Advanced Controls ist der verwendete Code sehr ähnlich

Eine weitere Ergänzung ist ein nicht sichtbares, zusätzliches Steuerelement auf jedem Achsen-Bedienfeld. Mit diesem *Unit*-Steuerelement wird festgehalten, wie ein Bedienfeld-Typ umgewandelt werden soll. Dieses wird zu Beginn des VI *GUI Input* für jede

Achse einzeln ausgelesen und ebenfalls in einem Array festgehalten. Dabei wird einem Element im Variant-Array immer derselbe Index zugewiesen, wie im *Unit-Array.*

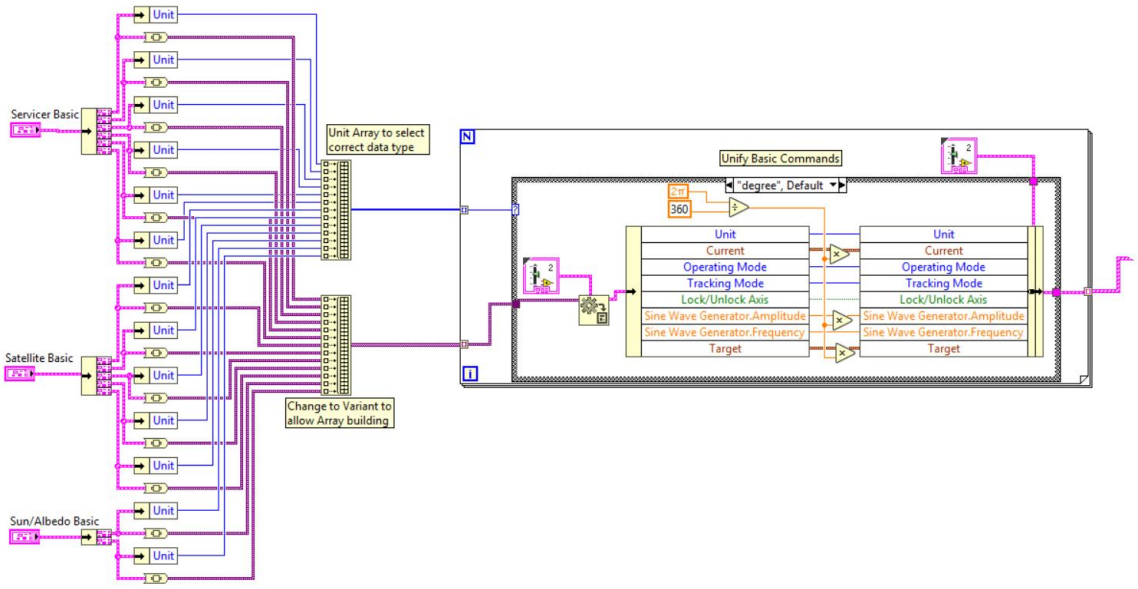


Abb. 4–19: GUI Loop Input – Vereinheitlichen der Basic Controls

In der nachfolgenden Zählschleife wird mittels diesem *Unit*-Array für jedes Variant-Element der angegebene Datentyp ausgewählt. Entsprechend wird das Variant-Element zurück in einen Cluster umgewandelt und gegebenenfalls eine Einheitenumrechnung von Grad zu Radianten durchgeführt. Anschließend werden alle Elemente in ein Array mit einheitlichen Datentypen geschrieben. Dieses Array muss alle verschiedenen Elemente der einzelnen Achsen-Bedienfelder enthalten, damit alle Benutzereingaben weitergegeben werden. Wenn eine Bedienfeldart ein Element oder einen Datentyp nicht besitzt, kann dieser leer gelassen oder mithilfe einer Konstanten überschrieben werden. Selbstverständlich sollte dann darauf geachtet werden, dass dieses Element im nachfolgenden Code entsprechend bearbeitet oder ignoriert wird.

Dieser Vorgang findet sowohl für die Basic Controls auf der ersten Registerkarte des GUI statt, als auch für die Advanced Settings der zweiten Karte.

Um die neu berechneten Daten am Ende einer GUI-Loop-Iteration wieder in die Bedienelemente schreiben zu können, müssen diese wieder zurück in ihre entsprechenden Cluster-Datentypen gewandelt werden. Dies geschieht im VI *GUI Output*, das entsprechende SubVI ist in Abb. 4–20 zu sehen.

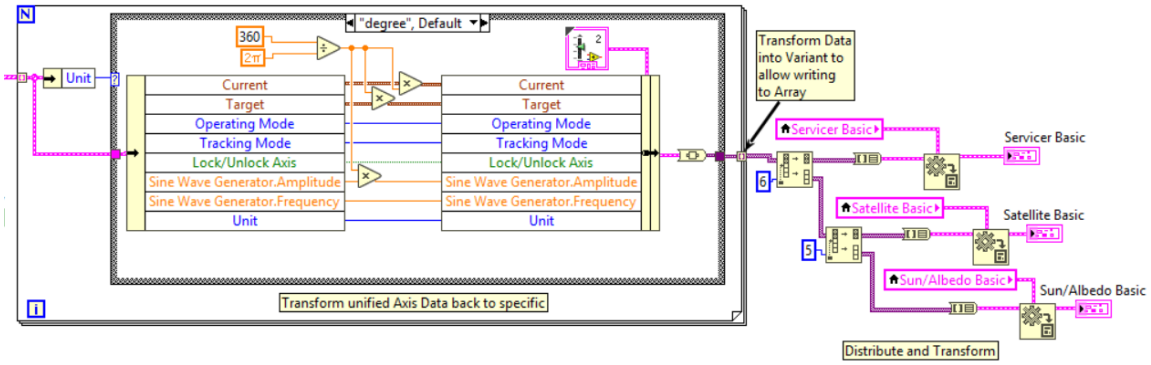


Abb. 4–20: GUI Output – Individualisierung der einzelnen Bedienelemente

Hier findet der eben beschriebene Prozess umgekehrt statt. Zuerst werden die Daten aus dem vereinheitlichten Array ausgelesen, gegebenenfalls umgerechnet und ihrem Unit-Wert entsprechend in einen Cluster mit den entsprechenden Datentypen geschrieben. Um eine For-Schleife verwenden zu können, werden die einzelnen Elemente wieder in den Datentyp Variant gewandelt und als Array aus der Schleife ausgegeben. Dieses Array wird in einen Servicer-, einen Target- und einen Simulator-Teil aufgespalten. Die Teilarrays werden in Cluster gewandelt und die darin enthaltenen Daten ebenfalls in ihren ursprünglichen Datentyp transformiert.

Ein anderes Konzept, welches von der ORION-Bedienoberfläche übernommen wurde, ist das gleichzeitige Steuern aller Achsen.

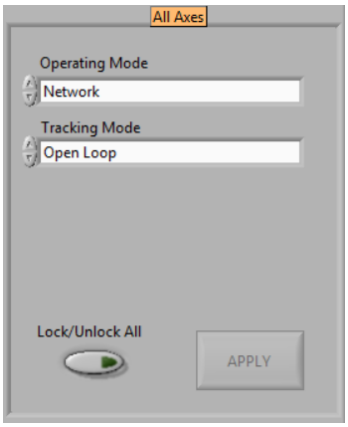


Abb. 4–21: All Axes Bedienfeld

Das dafür implementierte Bedienfeld befindet sich auf dem GUI oben rechts, und ist in Abbildung Abb. 4–21 zu sehen. Alle Eingaben, die hier getätigt werden, werden für alle Achsen gleichzeitig übernommen. Der Apply-Button wird im nachfolgenden Abschnitt beschrieben.

Eingaben in dieses Bedienfeld werden im SubVI *GUI Loop All Axes Handler* verarbeitet, zu sehen in Abbildung Abb. 4–22. Das Bedienfeld ermöglicht ein Steuern des Operating Modes, Tracking Modes und das Sperren aller Achsen mittels des Buttons *Lock/Unlock All*. Beim Sperren einer Achse wird der Betriebsmodus intern auf Stopping geschaltet und die Achse so angehalten. Für den Benutzer wird allerdings

weiterhin der ursprüngliche Betriebsmodus angezeigt. Wird das Sperren aufgehoben, wechselt der Betriebsmodus überall wieder in den ursprünglichen Modus, und die Achse kann sich wieder bewegen. Diese Funktion ist ebenfalls im *All Axes Handler* untergebracht.

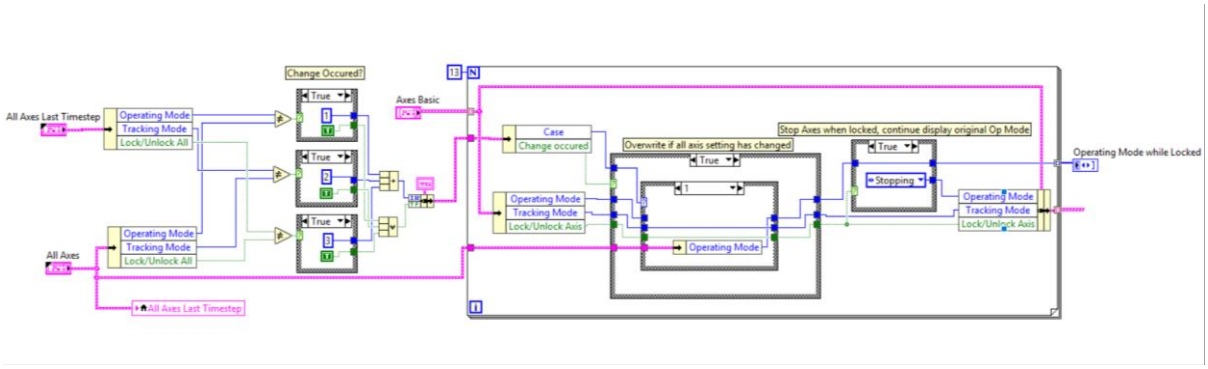


Abb. 4–22: All Axes Handler

Zu Beginn des SubVI werden die Daten von *All Axes* in dieser Iteration in ein internes Anzeigeelement *All Axes Last Timestep* geschrieben. Dieses dient als Speicher, um in der aktuellen Iteration Zugriff auf die Daten der letzten Iteration zu haben. Die Daten der aktuellen und der letzten Iteration werden verglichen, um so feststellen zu können, ob eine Änderung aufgetreten ist. Ist dies der Fall, wird der Wert *Change occurred* wahr und der Wert von *Case* wird je nach festgestellter Änderung geändert. In der nachfolgenden For-Loop wird dann in einer verschachtelten Case-Struktur zuerst der Wert *Change occurred* abgefragt. Falls er wahr ist, wird anschließend der Wert von *Case* überprüft und je nach Zustand die Werte der einzelnen Achsen von Operating Mode, Tracking Mode oder Lock/Unlock Axis zu den *All Axes* Werten geändert.

Die nachfolgende Case-Struktur ist für den Lock/Unlock-Prozess zuständig. Je nachdem, ob *Lock/Unlock Axis* wahr oder falsch ist, wird der Operating Mode auf Stopping geändert oder nicht. Bei einem *Lock/Unlock Axis = TRUE* werden zwei verschiedene Werte für den Operating Mode an das nachfolgende *GUI Comm Loop Interface* weitergegeben. Die Werte im *Axis Basic* Array werden an das cRIO übertragen. Im Falle eines aktivierten Axis Lock sind diese Werte auf Stopping geändert. Die Werte im Array *Operating Mode while locked* sind die nicht geänderten Eingangswerte von Operating Mode. Diese werden, nachdem die Daten einer Iteration an das cRIO gesendet wurden, im VI *GUI Comm Loop Interface* wiederverwendet.

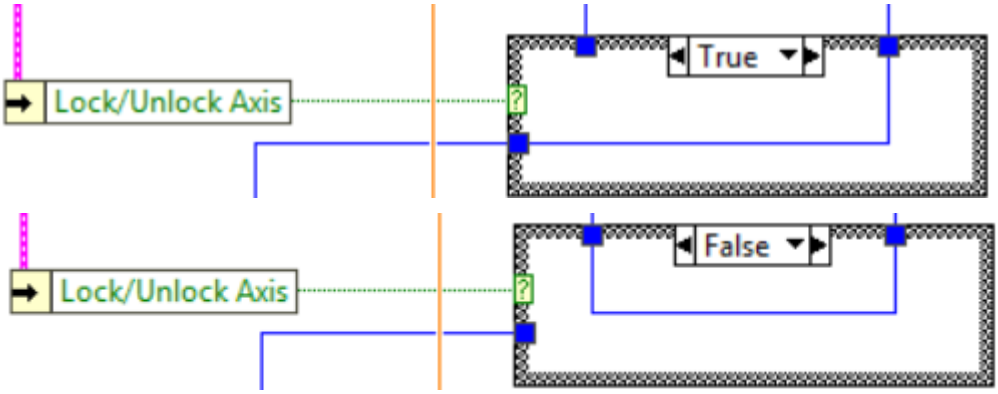


Abb. 4–23: Axis Mode while locked in der GUI Comm Loop

Hier wird, wie in der Abbildung Abb. 4–23 zu sehen, nochmals der Status von *Lock/Unlock Axis* abgefragt und dann bei einem Wahr die überbrückten Werte von *Operating Mode while locked* verwendet, oder bei einem Falsch die Werte aus dem Compare-VI. Wird nach einem Sperren der Achsen der Lock-Befehl aufgehoben, entfällt das auf Stopping Setzen und die Anlage kehrt zu dem ursprünglichen Betriebsmodus zurück.

4.2.3.3 Apply Button

Wegen der bereits im Konzepte-Abschnitt erwähnten Probleme beim Umsetzen der Apply-Button-Idee wurde dieser nur für die Eingabe von Zahlen als Bestätigung übernommen. Sprich beim Benutzen des Manual Modes und des Sine Wave Generator setzt sich die Anlage erst in Bewegung, wenn alle eingegebenen Zahlen mit dem Apply Button bestätigt werden. Dieser befindet sich im *All Axes* Bedienfeld.

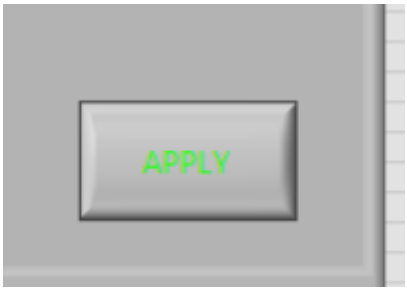


Abb. 4–24: Der Apply Button

Das Erscheinungsbild des Buttons ändert sich bei der Eingabe von Zahlenwerten und beim Klicken auf diesen Button. Diese Funktion wird im VI *GUI Output* gesteuert und wird im Abschnitt *Interaktive GUI* erklärt. Die Eingaben des Buttons hingegen werden im VI *GUI Comm Loop Interface* und dem darin enthaltenem SubVI *GUI Loop Compare* verarbeitet. Dieses wird nun hier erklärt.

Die Funktionsweise für die Manual-Mode-Eingaben, sprich den Eingaben aus den *Target*-Feldern, ist anders als die für den Sine Wave Generator. Für den Manual Mode wird im *GUI Loop Compare* VI mithilfe einer simplen und-Konjunktion das Weiterleiten eines wahren booleschen Compare-Ergebnis (siehe Abschnitt 3.3) an die Datenübertragungssteuerung verzögert, bis der Apply Button durch einen Klick auf TRUE geschalten wird. Erst dann wird ein TRUE-Signal an das cRIO übertragen.

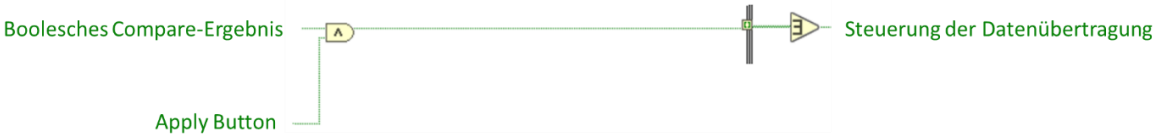


Abb. 4–25: Funktion Apply Button für Target im Compare VI

So werden erst die neuen Manual-Mode-Eingaben aus dem GUI auf dem Host-PC an das cRIO gesendet, wenn der Apply Button gedrückt wird.

Für Sine-Wave-Generator-Eingaben ist das Verfahren etwas komplizierter, da diese nicht auf einem eigenständigen Shared-Variables-Kanal übertragen werden, sondern in der Variable *Operation Init* untergebracht sind. In Abb. 4–26 ist der Codeausschnitt zu sehen, der für dieses zuständig ist. Wenn der Apply Button nicht gedrückt ist, sprich sein Wert FALSE ist, ist das Programm in dem oberen Zustand. Andere Werte im

Cluster *Operation Init* werden ganz normal übertragen, alle Sine-Wave-Generator-Werte werden jedoch von *SWG Last Apply* überschrieben. *SWG Last Apply* sind diejenigen Werte, die bei dem letzten Drücken des Apply Buttons übertragen wurden. Werte in *Operation Init* können also einfach passieren, neue Sine-Wave-Generator-Werte können jedoch nicht übertragen werden.

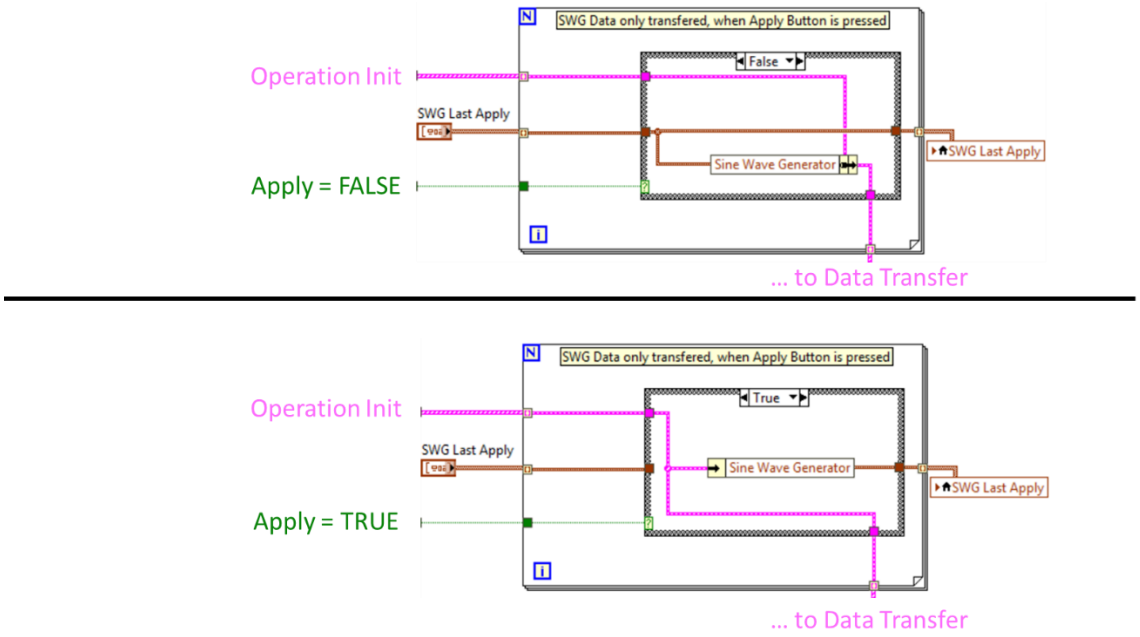


Abb. 4–26: Funktion Apply Button für SWG im Comm Interface

Erst wenn der Apply-Button-Wert durch einen Klick auf diesen TRUE wird, werden die aktuell eingegebenen Werte des Sine Wave Generator an die Shared Variable und somit an das cRIO übertragen. Gleichzeitig wird in *SWG Last Apply* ein neuer Wert geschrieben.

Der Apply Button verhindert in seiner nun implementierten Form ein ungewolltes Fahren der Anlage und ermöglicht ein einfaches, gleichzeitiges Bewegen von verschiedenen Achsen. Durch diese Bestätigung soll dem Benutzer klar werden, ab wann genau die Anlage denn nun fährt. Allerdings ist es wiederum ein zusätzliches Bedienelement, welches die Bedienoberfläche komplexer macht, und welches eine Erklärung der Anlagenbedienung verkompliziert. Um dem entgegenzuwirken, wurde eine automatische Änderung des Aussehens des Buttons implementiert, die im folgenden Abschnitt beschrieben wird.

4.2.3.4 Interaktive GUI

Durch das Verschieben der GUI-Steuerung auf den Host-PC stehen nun deutlich komplexere Funktionen und Befehle zur Verfügung, die auf dem cRIO nicht möglich waren. Dies erlaubt das Gestalten einer interaktiven Benutzeroberfläche, die durch das Ein- und Ausblenden sowie durch Verändern von Bedienelementen eine Kommunikation von Ereignissen und erforderlichen Befehlen verbessern kann. Neben dem Setup Window und dem Caution and Advisory Display, die jeweils in ihren eigenen Kapiteln beschrieben werden, wurden folgende beiden Funktionen implementiert:

Hervorheben des Apply Button

Der im vorangegangenen Kapitel beschriebene Apply Button ist für den Großteil der Zeit ausgegraut und nicht auswählbar. Erst wenn in eines der Target-Felder oder der SWG-Felder ein Wert eingegeben wird, wird der Button auswählbar und farblich hervorgehoben.

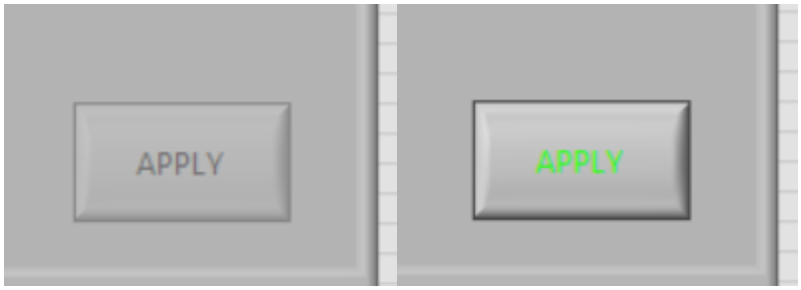


Abb. 4–27: Veränderung des Erscheinungsbilds des Apply Buttons

Hervorheben von LateCount Control Loop

Der *LateCount*-Wert gibt an, wie oft die Control Loop auf dem cRIO zu spät beendet wurde. Da dies im Normalfall nicht auftreten sollte und die Anlagenpräzision beeinflussen kann, wird bei einer Erhöhung dieses Werts das entsprechende Zahlenfeld mit Rot umrahmt.

Abb. 4–28: Veränderung des Erscheinungsbilds des LateCount

Beide Funktionen werden im SubVI *GUI Loop Interactive GUI* gesteuert, welches im SubVI *GUI Loop GUI Output* aufgerufen wird. Dieses ist in Abbildung Abb. 4–29 zu sehen.

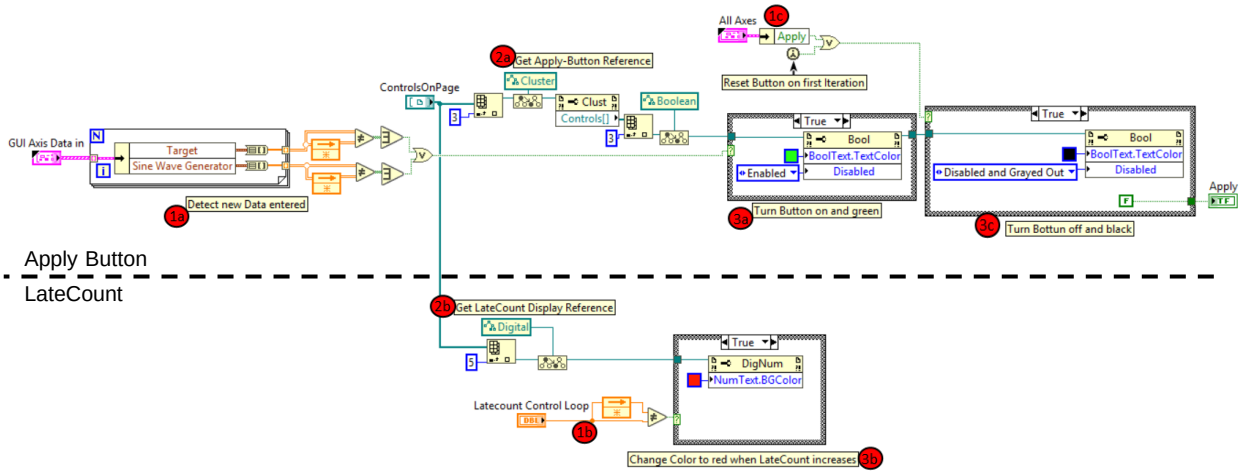


Abb. 4–29: InteractiveGUI.vi

Die obere Hälfte beinhaltet den Code für den Apply Button, die untere Hälfte den für das LateCount-Textfeld.

Bei 1a und 1b werden Signale aus der GUI Loop mithilfe von Feedback Nodes mit den Signalen aus der jeweils vorherigen Iteration verglichen. So können Änderungen festgestellt werden, und anschließend ein TRUE-Signal an die jeweils nachfolgenden Switch-Case-Strukturen gesendet werden.

Bei 2a und 2b wird jeweils eine Referenz zu den beiden Bedienelementen erzeugt, welche in den nachfolgenden Switch-Case-Strukturen verwendet werden.

Mit einer solchen Referenz wird bei 3b mithilfe einer Property Node im Falle eines TRUE-Wertes aus dem vorangegangenen Vergleich die Farbe des LateCount-Feldes in Rot geändert. Ist das Signal aus dem Vergleich FALSE, wird dieses Nummernfeld wieder grau.

Bei 3a wird bei einem wahren Ergebnis aus dem Vergleich bei 1a der Apply Button aktiviert, und die Schriftart in grün geändert. Im Falle eines FALSE-Ergebnisses passiert nichts. Erst wenn über 1c ein Drücken des Apply Buttons registriert wird und der hier nachfolgende Switch Case (3c) auf TRUE wechselt, wird der Button wieder in seinen deaktivierten Normalzustand zurückgesetzt.

Beide Funktionen sorgen für ein einfacheres Verständnis der Benutzeroberfläche bzw. heben einen Handlungsbedarf hervor. Es sind allerdings auch noch deutlich komplexere Funktionen möglich.

4.2.3.5 Caution and Advisory Display

Ebenfalls Gebrauch von den nun möglichen Codebausteinen macht das im Folgenden vorgestellte Anzeigeelement. Das Caution and Advisory Display (CAD) dient zum Ein- und Ausblenden relevanter Informationen und ist im rechten unteren Eck des GUI zu finden. Ein beispielhafter Zustand von diesem ist in Abbildung Abb. 4–30 dargestellt. Kritische Warnungen werden in weißer Schrift, rot hinterlegt und immer ganz oben angezeigt. Hinweise und sonstige Informationen sind in schwarz geschrieben.

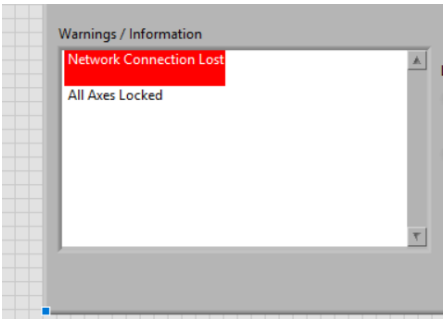


Abb. 4–30: Caution and Advisory Display

Die Steuerung des Caution and Advisory Display ist im VI *GUI Loop CAD* untergebracht. Das Anzeigen einer Warnung soll nun beispielhaft mit der kritischen Warnung *Network Connection Lost* gezeigt werden, dies funktioniert mit allen anderen Meldungen prinzipiell gleich.

Zuerst wird festgestellt, ob ein Signal angezeigt werden soll. Mittels Daten aus der GUI Loop wird ein boolescher Wert ermittelt, der mit einer Switch-Case-Struktur verbunden, entscheidet, ob eine Mitteilung angezeigt wird oder nicht. In unserem Beispiel wird der

boolesche Wert *Network Connection Lost* an einen der Eingänge einer UND-Verknüpfung geführt. An das andere Ende angeschlossen ist das Ergebnis der Abfrage, ob mindestens eines der Achsen den Operating Mode *Network* verwendet. Eine Warnung über eine verlorene Netzwerkverbindung ist schließlich nur notwendig, wenn dieses auch verwendet wird.

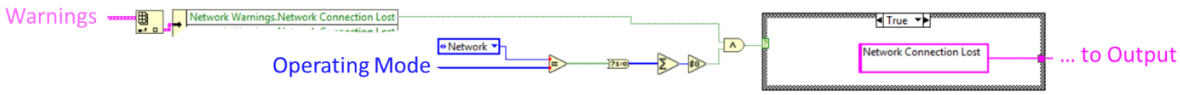


Abb. 4–31: Erstellung der Warnung Network Connection Lost

Ist das Ergebnis der Konjunktion wahr, wird der String *Network Connection Lost \n* an den nachfolgenden Output-Teil zum Anzeigen der Meldung gesendet. Wichtig ist hierbei der Zeilenumbruch \n der dafür sorgt, dass alle Meldungen in jeweils einer eigenen Zeile angezeigt werden. Ist das Ergebnis unwahr, wird ein leerer String weitergegeben, der dann letzten Endes nicht zu sehen ist.

Im unten abgebildeten Output-Teil des CAD-VI wird nun zwischen kritischen Warnungen und nicht-kritischen Hinweisen unterschieden. Bei beiden Arten von Meldungen werden zuerst alle Strings aneinandergehängt, sodass jeweils ein String entsteht, in dem dank dem eben erwähnten, hinten angestellten Zeilenumbruch alle Meldungen in jeweils einer eigenen Zeile stehen. Beide dieser Strings werden anschließend zusammen in einem Array gebündelt.

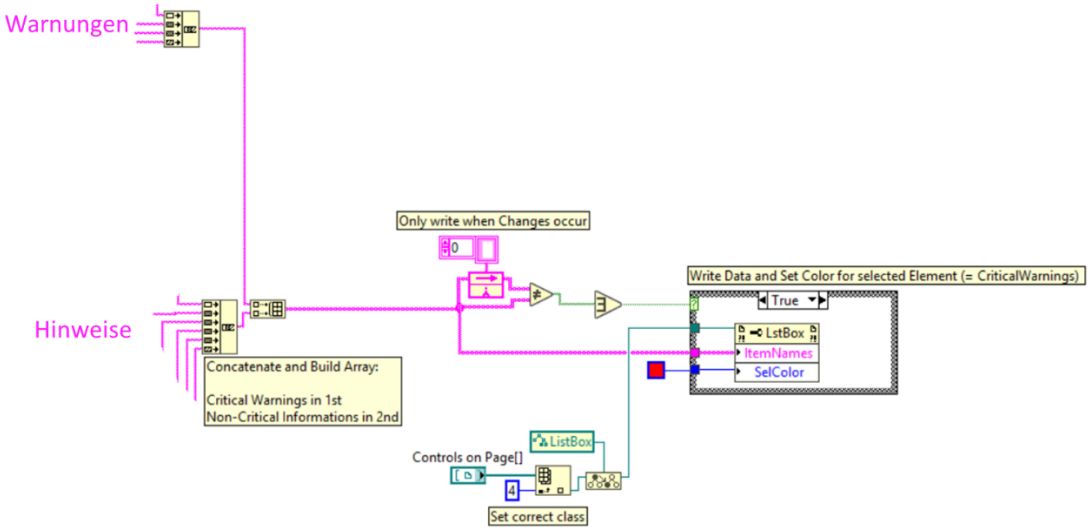


Abb. 4–32: Bündeln und Anzeigen aller Warnungen im CAD

Dieses wird mittels einer Feedback Node mit seiner Version aus der vorangegangenen Iteration verglichen. Nur bei einer festgestellten Änderung wird der nachfolgende Switch Case aktiviert und der String-Array mithilfe einer Property Node in eine ListBox geschrieben.

Diese ListBox ist das Caution and Advisory Display. ListBoxes sind eigentlich Steuerelemente zum Auswählen von Elementen in einer Aufzählung [15]. Durch ein Deaktivieren von Eingaben wurde dieses Steuerelement in ein bloßes Anzeigeelement

verwandelt. Der String-Array, der mit der Property Node in die ListBox geschrieben wird, ändert einfach nur die Namen der beiden Auswahlelemente in der ListBox. Sie besteht im Grunde nur aus zwei Elementen. Durch die Zeilenumbrüche wirkt die Liste jedoch deutlich länger. Ausgewählte Elemente in einer ListBox werden normalerweise blau hinterlegt angezeigt. Auch dies wurde beim Caution and Advisory Display ausgenutzt. Im Prinzip ist einfach nur das Element der ListBox, in dem die kritischen Warnungen geschrieben werden, dauerhaft ausgewählt und deshalb farblich hinterlegt. Um diese Warnungen zusätzlich noch hervorzuheben, wurde die Farbe der Hinterlegung in Rot geändert.

Es können jedoch auch deutlich komplexere Meldungen angezeigt werden. Beispielsweise werden bei einem Erreichen eines Soft Limits nicht nur eine entsprechende allgemeine Meldung angezeigt, sondern auch die entsprechende Achse oder Achsen, für welche diese Meldung zutrifft. Ein beispielhafter Code für das Erstellen von achsenspezifischen Meldungen ist in Abbildung Abb. 4–33 zu sehen.

Links am Eingang, mit einem A markiert, liegt ein Array, bestehend aus booleschen Werten, die angeben, ob eine Achse ein Soft Limit erreicht hat oder nicht. Dieses wird in ein Zahlen-Array umgewandelt, wobei ein TRUE einer 1 entspricht und ein FALSE einer 0. Außerdem befindet sich bei A ein weiterer Array, der in seinen 13 Elementen die Zahlen von 0 bis 12 enthält, welche die einzelnen Achsen wiederspiegeln sollen. Gleich zu Beginn werden alle diese Zahlen um eins erhöht. Der Grund hierfür ist, dass später zwischen der nullten Achse und Null als negatives Signal unterschieden werden muss. Beide Arrays werden miteinander multipliziert, sodass das Ergebnis ein Array mit Nullen und denjenigen Achsen ist, welche ein Soft Limit erreicht haben.

Bei B wird dieses Array zuerst aufsteigend sortiert und danach umgekehrt, sodass nun alle Nullen unten angeordnet sind. Anschließend wird in einer parallelen Bahn nach demjenigen Index gesucht, ab dem eine Null auftritt.

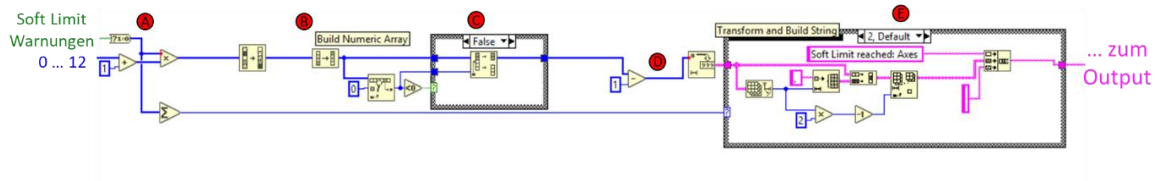


Abb. 4–33: Erstellung des Hinweises Soft Limit reached

Bei C werden ab diesem Index alle Arrayelemente gelöscht, sodass nur noch ein Array übrigbleibt, in dem alle anzuzeigenden Achsen enthalten sind. Zusätzlich wird bei C noch eine Fallunterscheidung durchgeführt. Denn wenn bei B keine Nullen im Array gefunden wurden, gibt dieser Funktionsblock statt eines Index den Wert -1 zurück. Leider löscht der Funktionsblock bei C bei einem Index von -1 das gesamte Array. Daher wird zuerst überprüft, ob dieser Index kleiner als 0 ist. In diesem Fall wird dann kein Löschen durchgeführt.

Bei D werden alle Elemente wieder um eins verringert. Anschließend werden alle Integer-Zahlen im Array in Strings umgewandelt.

Bei E wird wieder eine Switch-Case-Struktur verwendet, um zwischen einem Anzeigen und einem nicht Anzeigen zu wechseln zu können. Zusätzlich wird aber hier noch ein dritter Fall verwendet, wenn nur eine Achse ein Soft Limit erreicht hat. Gesteuert wird

dieser Switch Case mithilfe des Einsen-und-Nullen-Array von A. Dessen Summe entscheidet über den verwendeten Case. Eine Null bedeutet keine Achse, eine Eins eine Achse und alle anderen Werte mehrere Achsen.

Im Falle mehrerer Achsen, in Abbildung Abb. 4–33 bei E zu sehen, wird die Länge des Arrays gemessen und entsprechend viele Kommas mit nachfolgendem Leerzeichen generiert. Mit dem Funktionsblock *Interleave 1D Arrays* wird ein neuer Array erstellt, in dem immer abwechselnd eine Achsenzahl und ein Komma eingefügt wird. Durch Verdoppeln der eben gemessenen Arraylänge kann dann präzise am Ende ein überflüssiges Komma entfernt werden. Der String, nun bestehend aus den anzuzeigenden Achsen, jeweils mit Komma und Leerzeichen getrennt, wird dann zwischen den String *Soft Limit reached: Axes* und einem Zeilenumbruch-String eingefügt. Eine so entstandene Meldung ist in Abbildung Abb. 4–34 zu sehen.

Abb. 4–34: Ergebnis Erstellung des Hinweises Soft Limit reached

Der Case für eine einzelne Achse sieht folgendermaßen aus:

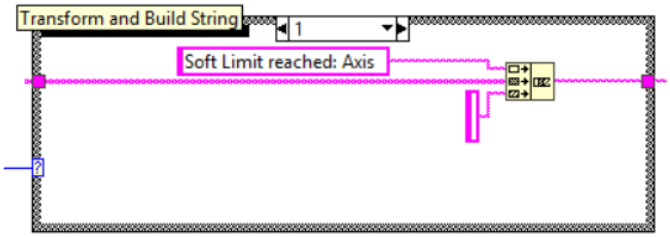


Abb. 4–35: Erstellung des Hinweises Soft Limit reached für eine Achse

Hier wird der Achsenzahl-String einfach direkt in den restlichen Text eingefügt.

Alle implementierten Meldungen sind in Tabelle Tab. 4–2 zusammengefasst. Es können jedoch beliebig neue Meldungen hinzugefügt werden, wobei sich beim Implementieren die Verwendung eines der beiden eben beschriebenen Muster empfiehlt. Eventuell muss ein neuer Eingang an das VI *GUI Loop CAD* gelegt werden oder eine neue Umgebungsvariable in das VI eingefügt werden.

Tab. 4–2: Alle im CAD implementierten Warnungen und Hinweise

Warnungen	
cRIO Connection Timeout	Einfache Warnung
Network Connection Lost	Einfache Warnung
Optitrack Connection Lost	Einfache Warnung
End Switches Triggered	Achsenspezifische Warnung
Hinweise	
Axes Locked	Achsenspezifischer Hinweis
All Axes Locked	Einfacher Hinweis
Soft Limit reached	Achsenspezifischer Hinweis
Ref Switch Triggered	Achsenspezifischer Hinweis
Streaming Trajectory Data	Einfacher Hinweis
File Read Error	Einfacher Hinweis
Logging-Status	Einfacher Hinweis

4.2.4 Ergebnisse

Die neue Benutzeroberfläche ist in Abb. 4–36 zu sehen. Dank des neu entwickelten Designs sind jetzt nicht nur alle Achsen gleichzeitig steuerbar, es ist darüber hinaus sogar noch Platz für weitere Bedienelemente für zukünftige Weiterentwicklungen der Software. So ist beispielsweise orange markiert ein Bereich für die geplante Erdalbedo-Steuerung freigelassen.

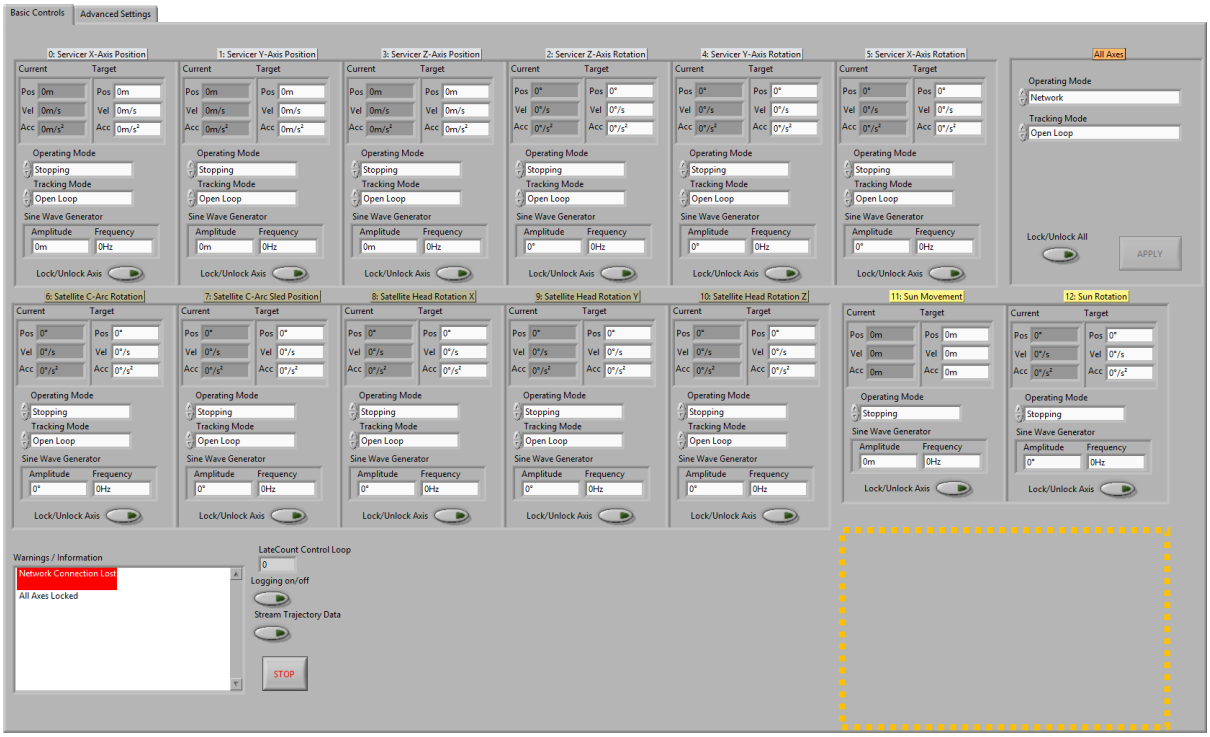


Abb. 4–36: Das neue Graphical User Interface

Da Benutzerfreundlichkeit letztendlich nur subjektiv bewertet werden kann, bleibt eine Beurteilung der Benutzeroberfläche in dieser Hinsicht immer ein Stück weit subjektiv. Es wäre theoretisch möglich, etwa mithilfe von Definitionen aus der Norm ISO 9241 eine Analyse des GUI oder auch eine Evaluation mit Testpersonen durchzuführen. Dies wäre jedoch mit einem großen und in diesem Fall nicht gerechtfertigtem Aufwand verbunden.

Eine Aussage dazu, ob das Ziel einer intuitiven Steuerung erreicht wurde, bleibt also wohl letzten Endes dem jeweiligen Benutzer überlassen. Es können jedoch die wesentlichen Neuerungen nochmals genannt und ihre potentiellen Beiträge zu einer Verbesserung aufgezeigt werden:

Gleichzeitiges Überwachen und Steuern aller Achsen

Zum einen fällt dadurch ein umständliches Durchklicken bis zur gewünschten Achse weg, um Eingaben zu machen. Zum anderen kann man nun mit einem Blick alle wesentlichen Informationen zu den einzelnen Achsen erhalten.

Bündeln und Hervorheben von Informationen

Durch das Entfernen oder Verschieben unnötiger Steuerelemente wurde der Großteil der ablenkenden Instrumente entfernt. Da zusätzlich ein großer Teil der übrig gebliebenen wichtigen Anzeigen für den Benutzer nur sichtbar ist, wenn ein Ereignis dies erfordert, konnte die Menge an Informationen auf dem Bildschirm noch weiter verringert und die Übersichtlichkeit erhöht werden. So ist nun das Fokussieren auf wesentliche Signale möglich. Durch Manipulieren von Bedienelementen können kritische Informationen hervorgehoben oder zusätzlicher Handlungsbedarf verdeutlicht werden.

Verbesserte Reaktionszeit

Es kommt beim Bedienen zu keinen Verzögerungen mehr und alle Eingaben werden auf Anhieb erkannt. Diese Verbesserung ist auf die in Kapiteln 3 durchgeführten strukturellen Änderungen zurückzuführen.

5 Anlagenparameter und Bewertung der Anlage

Ziel dieses Abschnitts sind zum einen das Bestimmen des Umrechnungsfaktors m_to_steps. Zum anderen soll eine Empfehlung ausgesprochen werden, wie die Anlage für eine möglichst gleichbleibende Präzision verwendet werden sollte. Dies beinhaltet ein Untersuchen der Referenzfahrt sowie eine grobe Abschätzung zur Anlagenpräzision. Anhand dieser Ergebnisse wird dann entschieden, ob es für einen zuverlässigen Anlagenbetrieb besser ist, regelmäßig Referenzfahrten durchzuführen, oder ob dies womöglich sogar von Nachteil ist.

Für alle Aufgaben sind Ortsmessungen von Anlagenkomponenten erforderlich, welche mithilfe des Optitrack-Systems durchgeführt werden. Dieses nimmt mit einer Frequenz von 100 Hz Infrarotbilder von den auf der Anlage angebrachten Markern auf. Über die bekannte Position der einzelnen Kameras im Raum, sowie aus den zweidimensionalen Ortsinformationen aus den aufgenommenen Bildern errechnet Optitrack eine dreidimensionale Position von den Markern. Durch die Definition von Festkörpern, auf denen die Marker angebracht sind, lässt sich aus der Lage der einzelnen Marker Orts- und Lageinformationen dieser Festkörper ermitteln. So kann die Position und Orientierung von mit Markern versehenen und als Festkörper definierten Anlagenteilen erhalten werden. Gemessen werden können selbstverständlich nur diejenigen Achsen, die von den Kameras erfasst werden können, sprich Achse 0 bis 10. Um Ergebnisse aus diesen Messungen sinnvoll bewerten zu können, soll zuerst die zulässige Genauigkeit für diese festgelegt werden.

Neben den erwähnten Orts- und Lagevektoren kann man von Optitrack auch den Wert *Error per Marker* (epm) erhalten. Dieser wird für jedes aufgenommene Frame und für jeden definierten Körper angegeben. Er entspricht der Distanz, die zwischen der gemessenen und einer erwarteten Position eines Markers liegt [16]. Was genau ein erwarteter Wert ist, und wie er bestimmt wird, ist nicht bekannt, vermutlich aus überschüssigen Informationen aus den Kameraaufnahmen. So erhält Optitrack schon aus zwei Kamerabildern eines Markers vier Werte zur Bestimmung der dreidimensionalen Lage im Raum. Der Wert *Error per Marker* gibt also einen im Messgerät festgestellten Fehler an. Folglich ist eine Messgenauigkeit unter dem Wert für *Error per Marker* nicht zulässig. Wie genau *Error per Marker* innerhalb Optitrack berechnet wird, ist nicht bekannt. Trotzdem sollte der Wert bei der Angabe von Messwerten berücksichtigt werden.

Da festgestellt wurde, dass je nach Achse und Ort zufällige Fehler unterschiedlich groß sein können, wurde für jede Messung dieser einzeln bestimmt.

5.1 Step-Conversion

Eine wichtige Größe für das korrekte Berechnen von Anlagenwerten ist der Faktor m_to_steps. Dieser dient zur Umrechnung von Distanzen in der internen Einheit Steps zu Metern oder Radianten. Ein Step stellt dabei einen Schritt der in der Anlage verbauten Schrittmotoren dar. Da für die unterschiedlichen Antriebe der Achsen teils unterschiedliche Mechanik verbaut ist, besitzt jede Achse einen eigenen Faktor.

Es existieren bereits m_to_steps-Werte für alle Achsen, allerdings ist ihr Ursprung unbekannt. Um also eine verlässliche Größenumrechnung garantieren zu können, sollen im Rahmen dieser Arbeit diese überprüft und falls nötig, neue Werte für diesen Faktor

bestimmt werden. Das Bestimmen soll folgendermaßen ablaufen: Mittels des neuen Trajectory Follow werden Trajektorien nachgefahren. Diese Fahrten werden sowohl vom Optitrack-System als auch vom neu implementierten, internen Logging aufgezeichnet. Optitrack misst die tatsächlich gefahrene Strecke. Das Logging misst die Steps, die dafür zurückgelegt wurden. Durch Verrechnung dieser beiden Größen kann dann m_to_steps bestimmt werden. Die dabei verwendeten Trajektorien-Dateien sollen so definiert werden, dass eine gleichbleibende Strecke mindestens 6 Mal abgefahren wird. Messungen in der Bewegung unterliegen größeren Abweichungen. Darum pausiert die Anlage zwischen den einzelnen Fahrten für mindestens 1 Sekunde. Nur während dieses Stillstands gemessene Werte werden verwendet. So werden für jede zu messende Position mindestens 100 Frames aufgenommen und aus diesen der Mittelwert gebildet, wodurch Messrauschen minimiert wird.

Konkret berechnet wird *m_to_steps* folgendermaßen. Der Betrag aus der Differenz zwischen zwei zeitlich hintereinander folgenden Positionen ergibt ein Intervall. Die Summe dieser Intervalle ist, je nach Achse, die insgesamt gefahrene Strecke s_{meter} oder s_{rad}. Die intern bei Logging gemessene Strecke ist s_{steps}. Die Definition des Wertes *m_to_steps,* beziehungsweise des Faktors $k_{m_to_steps}$ ist wie in (5–1) zu sehen.

$$k_{m_to_steps} = s_{steps} \div s_{meter,rad} \qquad \textbf{(5–1)}$$

Die gemittelt zurückgelegte Strecke mit ermitteltem Fehler, die gesamte Steps-Zahl sowie die endgültigen neuen m_to_steps-Werte sind in Tab. 5–1 zu sehen.

Tab. 5–1: Ergebnisse Bestimmung m_to_steps

Achse	$s_{meter,rad}$	Δs_{gesamt}	s_{steps}	m_to_steps
0	13,82 m	0,035 m	1181280	85500
1	6,616 m	0,001 m	564960	85400
2	19,03 rad	0,048 rad	511680	26900
3	4,200 m	0,00076 m	950400	226300
4	10,00 rad	0,068 rad	256100	25600
5	12,59 rad	0,019 rad	306960	24400
6	7,05 rad	0,024 rad	230220	32700
7	15,9 rad	0,14 rad	2559900	161000
8	5,63 rad	0,04 rad	193600	34400
9	3,914 rad	0,0054 rad	102348	26100
10	31,6 rad	0,12 rad	921000	29100

Der Gesamtfehler Δs_{gesamt} setzt sich aus der Standartabweichung $\Delta s_{meter,rad}$ von $s_{meter,rad}$ sowie aus *Error per Marker* zusammen. *Error per Marker* für Drehungen $\Delta\varphi_{epm}$ wird dabei aus dem bekannten *Error per Marker* für Translation Δx_{epm} berechnet.

$$\Delta\varphi_{epm} = \sin^{-1}(2 * \Delta x_{epm} \div d_{min}) \qquad (5\text{–}2)$$

Die Herleitung für die Formel (5–2) ist in Abb. 5–1 zu sehen. Drehungen erfolgen immer um den Drehpunkt, auch Optitrack-intern wird hierfür ein Punkt definiert, mit dem das Programm dann rechnet. Der epm-Wert gibt die durchschnittliche Abweichung eines Markers an. Der Drehpunkt wird in Abhängigkeit aller Marker eines Festkörpers ermittelt. Sprich ein Worst-Case-Szenario beim Messen wäre, wenn wie in Abb. 5–1 zu sehen, der dem Drehpunkt naheliegendste Marker in eine Richtung abweicht, alle anderen Marker, und damit der Drehpunkt, in die andere Richtung. Es wurde konservativ der Marker mit dem kleinsten Abstand d_{min} gewählt, da hier ein Fehler den größten Einfluss hat. Die Distanz von Marker mit kleinstem Abstand zum Drehpunkt ist etwa $d_{min,Servicer}$ = 0,2 m für den Servicer und $d_{min,Target}$ = 0,6 m für das Target. Mithilfe des Dreiecks *abc* und dem Sinussatz kann so das größtmögliche $\Delta\varphi_{epm}$ bestimmt werden.

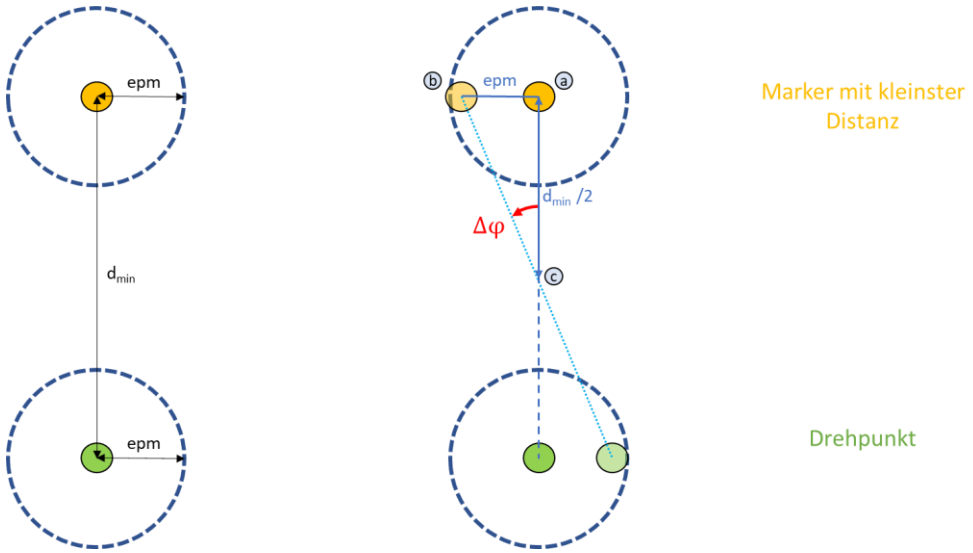


Abb. 5–1: Herleitung des epm-Wertes für Drehungen

Leider ist *Error per Marker* für den Großteil der Messung sehr groß, die Standardabweichung fällt beim Gesamtfehler kaum ins Gewicht. Grund für den im Allgemeinen recht hohen epm-Wert können die im Abschnitt 4.1.1.1 beschriebenen grundlegenden Probleme beim RACOON-Optitrack sein. Manche hohen Werte lassen aber auch andere, vermeidbare Erklärungen zu.

Der epm-Wert von Achse 0 steigt beim Fahren in Richtung Target, sprich von den messenden Kameras weg, sprunghaft an. Eine Erklärung hierfür könnte sein, dass möglicherweise manche Kameras das freie Sichtfeld auf einige der Marker verlieren, weil dieses durch den restlichen Servicer-Teil verdeckt werden. Diese Effekte lassen sich durch ein sorgfältiges Auswählen des Messbereichs vermeiden. Gerade im Hinblick einer geplanten Positions-Rückkopplung sollte auf dieses Problem aber womöglich mit grundlegenden Änderungen reagiert werden.

Bei der Achse 7 fällt ebenfalls ein sprunghafter Anstieg auf sehr hohe Fehlerwerte auf. Hier tritt er im Gebiet um das Überschreiten von 90° Rotation auf. Eine Erklärung in

diesem Fall kann sein, dass in dieser Situation einige Marker, vom Target selbst verdeckt, aus dem Kamerasichtfeld verschwinden. Durch die bereits erwähnten grundlegenden Probleme bei Optitrack-Messungen wird das Erkennen der stattdessen neu erscheinenden Marker behindert. Bei Achse 10 lässt sich ein ähnlicher Effekt beobachten. Falls möglich, kann zur Vermeidung auf Messungen mit Bewegungen unter 90° zurückgegriffen werden.

Weitere Fehlerquellen bei der Rotation könnten Optitrack-intern liegen. Etwa das Überschreitungen von 180° Drehungen, bei denen Optitrack zu -180° springt, oder mathematische Beschränkungen der verwendeten Eulerwinkel sein. Das Ergebnis sind Sprünge und Wendungen der eigentlich linearen Verläufe der zurückgelegten Drehungen, welche das Auswerten erschweren. Messfehler lassen sich für diese Phänomene mit großer Sicherheit ausschließen, da sie regelmäßig und immer an den genau gleichen Punkten auftreten.

Die nun neu berechneten Werte sind teils sehr ähnlich zu den Ausgangswerten, oft ist der Unterschied kleiner als 1 %. Stark aus der Reihe fällt Achse 8, hier hatte sich das Target für eine angebliche Drehung um 7° um etwa das Vierfache bewegt.

5.2 Anlagenpräzision

Aus den eben gemessenen Werten lassen sich auch Rückschlüsse zur Anlagenpräzision ziehen. Eine verlässliche Anlage sollte bei den eben gemessenen Fahrten immer zu den gleichen Positionen zurückkehren. Durch Vergleichen von diesen kann also eine Aussage zur Präzision der Anlage getroffen werden.

Hierbei ist für die meisten Achsen die Präzision der Messung nicht hoch genug, um eine Positionsabweichung festzustellen. In diesen Fällen kann die vermutete Anlagenpräzision also nur bis zur Genauigkeit der Messung angegeben werden. Die genaueste Aussage für Translation lässt sich hier wohl für Achse 3 machen. Selbst mit einem Gesamtmessfehler von kleiner 1 mm konnte hier keine Abweichung zwischen den einzelnen Haltepunkten gemessen werden. Für rotatorische Bewegungen gilt das gleiche für Achse 9, mit einem Messfehler von etwa 0,3 Grad. Einzige Ausnahme ist die Achse 7, deren stetiges Zunehmen der gefahrenen Strecke auffällig ist. Die Änderungen liegen jedoch innerhalb des sehr hohen Messfehlers von 0,14 rad. Bei einer Vergleichsmessung mit einer kleineren Bewegung von etwa 14,5°, welche auch einen deutlich niedrigeren epm-Wert aufwies, trat dieser Effekt jedoch nicht auf.

Ob die gefahrene Strecke lang genug ist, hängt von dem Anwendungsfall ab, für den die Präzision bestimmt werden soll. Für das Testen eines Sensors, bei dem stückweise die Position variiert wird, um aus verschiedenen Richtungen Messungen vorzunehmen, reicht die bewertete Strecke aus. Für dauerhaftes Fahren über mehr als 10 Minuten können kein Aussagen getroffen werden.

Eine genaue Angabe zur Anlagenpräzision ist nicht möglich, da hierfür die Messungen nicht präzise genug sind. Wie die Achse 3 und Achse 9 vermuten lassen, liegt sie wohl bei unter einem Millimeter beziehungsweise unter 0,3 Grad. Für die gesamte Anlage bestätigen lassen sich jedoch nur eine Genauigkeit von unter 3,5 mm beziehungsweise 8,1 Grad.

5.3 Referenzfahrt

Es soll herausgefunden werden, ob Referenzfahrten von jeweils verschiedenen Positionen der einzelnen Achsen immer zum gleichen Nullpunkt führen, oder ob Abweichungen zu sehen sind. Die fünf gewählten Positionen sind dabei bei jeder Achse nahe den beiden Enden der fahrbaren Distanzen, kurz vor, sowie kurz nach dem Nullpunkt und am Nullpunkt selbst. Die Punkte werden dabei in der folgenden Reihenfolge abgefahren:

1. Negativ, große Distanz zum Nullpunkt
2. Negativ, kleine Distanz zum Nullpunkt
3. Nullpunkt
4. Positiv, kleine Distanz zum Nullpunkt
5. Positiv, große Distanz zum Nullpunkt

Sobald eine Referenzfahrt abgeschlossen ist, wird direkt zum nächsten Ausgangspunkt gefahren, und von dort eine neue Fahrt begonnen. Um Fehler zu vermeiden, werden dabei alle Achsen einzeln abgefahren. Während dieser Fahrten wird die Anlage mithilfe des Optitrack-Systems gefilmt, anschließend werden die Orts- und Lageinformationen der einzelnen Achsen exportiert. Durch ein Vergleichen der Endpositionen nach den Referenzfahrten kann festgestellt werden, ob eine Referenzfahrt auch wirklich zum gleichen Ausgangspunkt zurückführt und, falls es Abweichungen gibt, wie groß diese sind. Gemessen werden die Endpositionen in Ruhelage über mindestens einer Sekunde, und aus diesen Messwerten der arithmetische Mittelwert gebildet.

Dabei konnten keine Abweichungen innerhalb der zulässigen Messgenauigkeit festgestellt werden. Häufiges Durchführen von Referenzfahrten sollte also keine negativen Auswirkungen mit sich bringen. Da jedoch auch keine Abweichungen beim Ermitteln der Anlagenpräzision festgestellt werden konnten, sind häufige Referenzfahrten nicht unbedingt erforderlich. Lediglich zu Beginn der Benutzung sind diese zwingend durchzuführen.

6 Ergebnisse

6.1 Bewertung und Diskussion

Der Großteil der zu Beginn der Arbeit festgelegten Anforderungen wurde erfüllt, wie in Tab. 6–1 zu sehen ist. Kleine Abstriche im Vergleich zum ORION wurden beim Logging und Trajectory Follow gemacht. Das Logging erfolgt nicht in Textdateien, sondern in Binärdateien, die anschließend noch umgewandelt werden müssen. Außerdem wurde das weniger wichtige Logging der SplineInfo, sowie der nicht vollständig implementierten Optitrack-Position gestrichen. Beim Trajectory Follow wurde das Einlesen von Dateien vor den Programmstart gelegt. Grund für alle Änderungen war immer ein zu hoher Bedarf an Systemressourcen.

Tab. 6–1: ausgewertete Anforderungsliste

ID	Beschreibung	Verifikation	Status
	Allgemein		
1	Die Anlage soll mittels eines Steuerungscodes geschrieben mit dem Programm NI LabVIEW Version 17.0 (32-bit) gesteuert werden	Inspizierung	erfüllt
2	Der Steuerungscode soll auf dem bereits verbauten NI cRIO 9014 ausführbar sein	Inspizierung	erfüllt
3	Es soll möglichst viel Code aus dem vorhandenen Steuerungscode der ORION-Anlage übernommen werden	Inspizierung	Nicht erfüllt, große Teile der neuen Funktionen mussten neu geschrieben werden
4	Der Steuerungscode soll alle 13 Achsen der Anlage steuern können: • Translation Servicer in X-, Y- und Z-Richtung • Rotation Servicer in X-, Y- und Z-Richtung • Translation Target auf C-Schiene • Rotation C-Schiene des Targets • Rotation Target in X-, Y- und Z-Richtung • Translation Sonnensimulator • Rotation Sonnensimulator	Demonstration	erfüllt
5	Der Steuerungscode soll folgende Betriebsmodi (Operating Modes) zulassen: • Direkte Steuerung am PC vor Ort • Steuerung via internes LRT-Netzwerk • Individuell generierbare sinusförmige Bewegungen • Steuerung via Trajektorien-Datei (CSV-Datei)	Demonstration	erfüllt
6	Der Steuerungscode soll ein Überschreiten von Endschaltern und individuell eingestellten Limits erkennen und die entsprechende Achse stoppen	Demonstration	erfüllt

7	Der Steuerungscode soll aktuelle Befehle an das FPGA jederzeit mit einer Frequenz von 100 Hz senden	Messung mittels Get Timestamp.vi	erfüllt
	Positions- und Lagebestimmung		
8	Der Steuerungscode soll zwei verschiedene Methoden zur Orts- und Lagebestimmung verwenden können (außer für Sonnen-/Albedo-Simulator): • Open Loop: Bestimmung mittels vom FPGA gezählten „Motor-Steps“ • Closed Loop: zusätzliche Orts- und Lagebestimmung mittels Infrarotkameras	Demonstration	nicht erfüllt, fehlende Koordinatenumrechnung, ansonsten Funktionsbereit
9	Die Daten für den Closed-Loop-Betrieb sollen mittels der vorhandenen Hardware erhoben werden • 8 Kameras • Marker • Computer zur Auswertung • Vorhandene Netzwerkumgebung	Inspizierung	erfüllt
10	Die Daten für den Closed-Loop-Betrieb sollen mittels der vorhandenen Software und nur minimal angepassten Codes ausgewertet werden • Motive V2.0 von Optitrack • Export-Code in C++ der ORION-Anlage • LabVIEW-Code der ORION-Anlage	Inspizierung	nicht erfüllt, fehlende Koordinatenumrechnung, ansonsten Funktionsbereit
11	Verbesserung der Positioniergenauigkeit der Anlage zu Werten ähnlich der ORION-Anlage (max. Abweichung von > 1 mm bzw. > 1 Grad) im Closed-Loop-Betrieb	Abmessung mittels Optitrack	nicht erfüllt, ohne Closed Loop eine geschätzte Genauigkeit zwischen unter 1 mm bzw. 0,3° bis 3,5 mm bzw. 8,1°
	Datenlogging		
12	Es soll die Möglichkeit des Daten-Logging innerhalb des LabVIEW-Programms geben	Demonstration	erfüllt
13	Es soll das Logging von Position, Geschwindigkeit und Beschleunigung laut LabVIEW-Code sowie die Position laut Optitrack-Software sowie die SplineInfo jeder Achse möglich sein	Inspizierung	Erfüllt, SplineInfo-Logging und Optitrack-Position gestrichen
14	Das Logging soll in Echtzeit und sollte mit einer Taktzahl von 10 ms geschehen	Kontrolle der Log-Dateien	erfüllt
	GUI		
15	Das GUI sollte ohne Scrollen auf den Bildschirm des Host-PC passen	Inspizierung	erfüllt
16	Es sollte das Steuern und Überwachen aller 13 Achsen gleichzeitig möglich sein	Inspizierung	erfüllt
17	Es sollen mindestens folgende Dinge ablesbar sein: • Momentane Position jeder Achse • Momentane Geschwindigkeit jeder Achse • Momentane Beschleunigung jeder Achse • Fehler, Hinweise, ausgelöste Switches	Inspizierung	erfüllt

18	Es sollen mindestens folgende Dinge steuerbar sein: • Zielposition jeder Achse • Maximale Geschwindigkeit und maximale Beschleunigung jeder Achse • Betriebsmodus • Positions- und Lagebestimmung • Logging an/aus • Amplitude und Frequenz einer Sinus-Trajektorie • Individuelle Limits für jede Achse einstellbar • Lock/Unlock jeder Achse • Dateipfad eines Trajectory Input File	Inspizierung	Erfüllt, Dateipfad bei Neustart änderbar
19	Weniger wichtige Anzeige- und Steuerelemente sollen nicht bei den Basic Controls auftauchen	Inspizierung	erfüllt
20	Fehler, Hinweise und Warnungen sollen gebündelt in einem „Caution and Advisory Display“ angezeigt werden	Inspizierung	erfüllt
21	Servicer-, Target- und Lampen-Achsen sollen mittels Farbgebung geclustert werden	Inspizierung	erfüllt
22	Distanz soll in Meter, Rotation in Grad und Zeit in Sekunden angegeben werden	Inspizierung	erfüllt
23	Die Bedienung sollte subjektiv flüssig und ohne spürbare Verzögerungen möglich sein	Inspizierung	erfüllt

Intuitivität bei der Bedienung ist nicht objektiv messbar, sondern eine subjektive Einschätzung. Eine Quantifizierung der gemachten Fortschritte wäre wohl nur mit einer aufwändigen Evaluation unter anderem mit Versuchspersonen möglich. Da dies im Rahmen dieser Arbeit nicht möglich war, kann aber sehr gut auch argumentativ erörtert werden, wieso nun die neue Benutzeroberfläche als intuitiver und benutzerfreundlicher einzuordnen ist. Ein wesentlicher Punkt ist die Tatsache, dass für gleiche Aktionen nun deutlich weniger Aktionen erforderlich sind. So muss beispielsweise vor der Eingabe eines Befehls nicht mehr über ein Zahlenfeld die gewünschte Achse ausgewählt werden, da alle Achsen gleichzeitig steuerbar sind. Zwar könnte argumentiert werden, dass dadurch die Komplexität der Oberfläche vergrößert wurde; dem wurde aber durch ein räumliches Anordnen und durch eine geschickte Farbgebung entgegengewirkt. Die Benutzerfreundlichkeit wird durch die auf Bildschirmgröße zugeschnittene Fenstergröße erhöht und damit das Scrollen unnötig. Aufwendiger wurden das Ändern von Einstellungen zu Referenzfahrten und manche Einstellungen zu Soft Limits und einige weitere Bedienelemente. Da diese jedoch im normalen Betrieb nicht benötigt werden, beziehungsweise nur bei grundlegenden Änderungen an der Anlage selbst bearbeitet werden müssen, fällt dies kaum ins Gewicht. Durch die bessere Anordnung von Bedienelementen, können diese schneller gefunden werden. Dies hat weitere Zeitersparnis beim Bedienen zur Folge. Diese Tatsache führt auch gleich zu einem weiteren bedeutenden Argument: Die angezeigten Informationen sind nun nur auf die Wichtigsten beschränkt und zusätzlich durch visuelle Unterschiede priorisiert. Durch das eingeführte Caution and Advisory Display werden Informationen nur angezeigt, wenn eine Situation dies auch erfordert.

Durch interaktive Bedienelemente wird zusätzlicher Handlungsbedarf signalisiert, oder werden Warnungen hervorgehoben.

Nicht vollständig implementiert werden konnte das Closed-Loop-Feedback. Hier fehlt die Umrechnung vom Optitrack-Koordinatensystem zum Anlagen-Koordinatensystem. Es wurde anfangs der Aufwand unterschätzt, der mit dem Entwickeln dieser Umrechnung verbunden ist. Da die Kinematik der RACOON-Anlage sehr komplex ist, muss hier viel Zeit verwendet werden. Darum war die vollständige Umsetzung des Closed-Loop-Feedbacks im Rahmen dieser Arbeit nicht mehr möglich. Ebenfalls aus Zeitgründen gestrichen wurde die Weiterentwicklung der Sonnensimulator-Steuerung. Informationen dazu sind im Abschnitt 6.2.3 zu finden.

Grund für die mangelnde Zeit war in erster Linie das nicht vorhergesehene, erforderliche und sehr aufwendige Überarbeiten der Ausgangscodes. Auch musste der Großteil der zu implementierenden Funktionen neu geschrieben werden, da diese zu viel Systemressourcen beanspruchten.

Trotz dieser kleinen Abstriche ist diese Arbeit erfolgreich abgeschlossen worden, da die wichtigen Ziele Funktionsumfangserweiterung und Verbesserung der Benutzerfreundlichkeit erreicht wurden. Auch die Vorgabe, bestehende Eigenschaften nicht zu verschlechtern, wurde klar eingehalten, sogar übertroffen. Die neue Steuersoftware läuft deutlich effizienter und das Bedienen ist spürbar flüssiger und intuitiver.

Die gemessenen Werte sind leider weniger belastbar als erhofft, da aufgrund der epm-Werte eine nicht sehr hohe Messpräzision erreicht wurde. Wie genau Error per Marker innerhalb von Optitrack ermittelt wird, ist unbekannt. Daher kann letzten Endes nicht mit absoluter Sicherheit gesagt werden, ob die hier angewandte Interpretation zulässig ist. Dies gilt speziell auch für die Übertragung von Error per Marker als die Angabe einer Distanz auf Rotationen. Hier sollte zusätzlich nochmals betont werden, dass bei diesem Umrechnen konservativ die kleinste Markerdistanz verwendet wurde. Außerdem hat sich gezeigt, dass Optitrack für Rotationen in eigentlich lineare Verläufe Sprünge und Wendungen einbringt. Da in den gemachten Messungen nur feste Punkte und deren absolute Distanzen berechnet wurden, konnten diese herausgerechnet werden. Gerade im Hinblick auf eine geplante Positionsrückkopplung, bei der die Ortsbestimmung kontinuierlich ablaufen soll, sollte die Ursache der aufgetretenen Phänomene jedoch gefunden und unbedingt beseitigt werden.

6.2 Empfehlungen für weitere Entwicklung

Im Folgenden sind nun ein paar Ideen und Empfehlungen im Hinblick auf eine zukünftige Entwicklung der Steuersoftware aufgelistet.

6.2.1 Zeitpunkt der FPGA-Kommunikation

Beim Testen einer früheren Version des neuen Codes wurde festgestellt, dass die Anlage teils sehr unruhig lief. Mit einem Logic Analyzer wurde herausgefunden, dass die Schrittmotoren ihre Steuersignale nicht regelmäßig erhielten, sondern mit kleinen Pausen, in denen keine Signale übertragen wurden.

Als Grund für diesen Fehler wurde der Übertragungszeitpunkt der SplineInfo-Daten an das FPGA identifiziert. Zwar lief der damalige Code aus Sicht des cRIO-Controllers fehlerfrei, regelmäßig und ohne Überschreiten der maximalen Periodendauer. Allerdings änderte sich alle 8 Control-Loop-Iterationen der Übertragungszeitpunkt innerhalb der einzelnen Iterationen. Die Ursache hierfür war das VI *Control Loop Output*, das zum damaligen Zeitpunkt vor dem FPGA-Interface ausgeführt wurde. Durch das Schreiben von Netzwerkdaten verzögerte sich so die Datenübertragung an das FPGA um etwa 1,5 ms. Die Differenz von der 10 ms langen Periode ist in Abb. 6–1 dargestellt. Auffällig sind bei der fehlerhaften Struktur die extremen Schwankungen, welche die unstetigen Motorsignale verursachten. Zum Vergleich sind auch die zeitlichen Differenzen des aktuellen Codes angegeben.

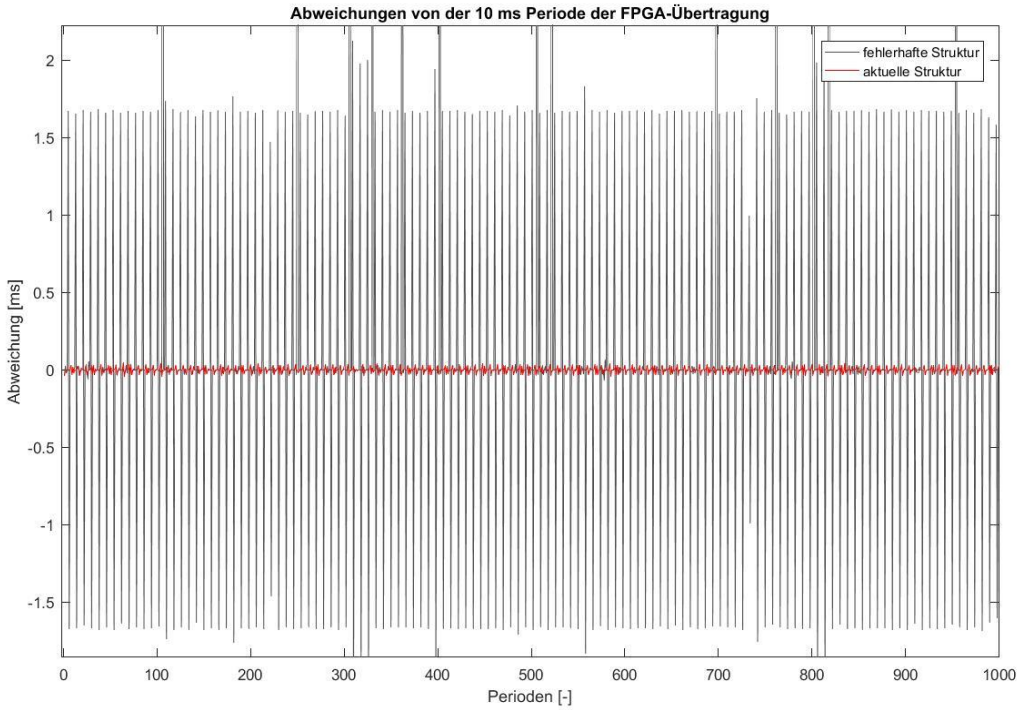


Abb. 6–1: Übertragungszeitpunkte der FPGA-Informationen bei zwei unterschiedlichen Programmstrukturen

Da es keine Pufferung zwischen Controller und FPGA gibt, fehlten dann dem FPGA die SplineInfo-Daten und es konnte deshalb keine neuen Motorsignale erzeugen.

Durch eine Umstrukturierung des Codes, bei der das Sub-VI Control Loop Output in den hinteren Teil der Schleife versetzt wurde, konnte dieses Problem gelöst werden.

Bei zukünftigen Weiterentwicklungen des Steuercodes sollte unbedingt darauf geachtet werden, dass die FPGA-Übertragungszeitpunkte immer in einem konstanten Abstand zueinander erfolgen. Sprich zwischen Iterationsbeginn und FPGA-Übertragung dürfen keine zeitlich variablen Operationen stattfinden. Eventuell kann auch ein klein gehaltener Puffer für die SplineInfo-Übertragung zwischen cRIO und FPGA implementiert werden.

6.2.2 Motorsignale

Gerade im Hinblick auf die Verbesserung der Anlagenpräzision sollte in Zukunft ein Blick auf die Steuersignale an die Motoren geworfen werden. Speziell die Step-Werte des Steuersignals an die Schrittmotoren ist interessant. Deren Regelmäßigkeit hängt unter anderem von der eben erwähnten Übertragungsperiode des FPGA-Signals statt.

In Abb. 6–2 sind die Differenz zwischen den Zeitpunkten der Übertragung von Motorsteps geplottet. Das Diagramm stellt nur einen Ausschnitt dar, bei dem die Punkte dichter zusammenliegen, auf der y-Achse weiter nach oben sind noch mehr, feiner verteilte Punkte. Es wurde diese Darstellungsweise gewählt, da so am deutlichsten die einzelnen Anhäufungen gleicher Zeitdifferenzen zu sehen sind. Für jede Programmversion gibt es in unterschiedlichen Bereichen eine solche Anhäufung.

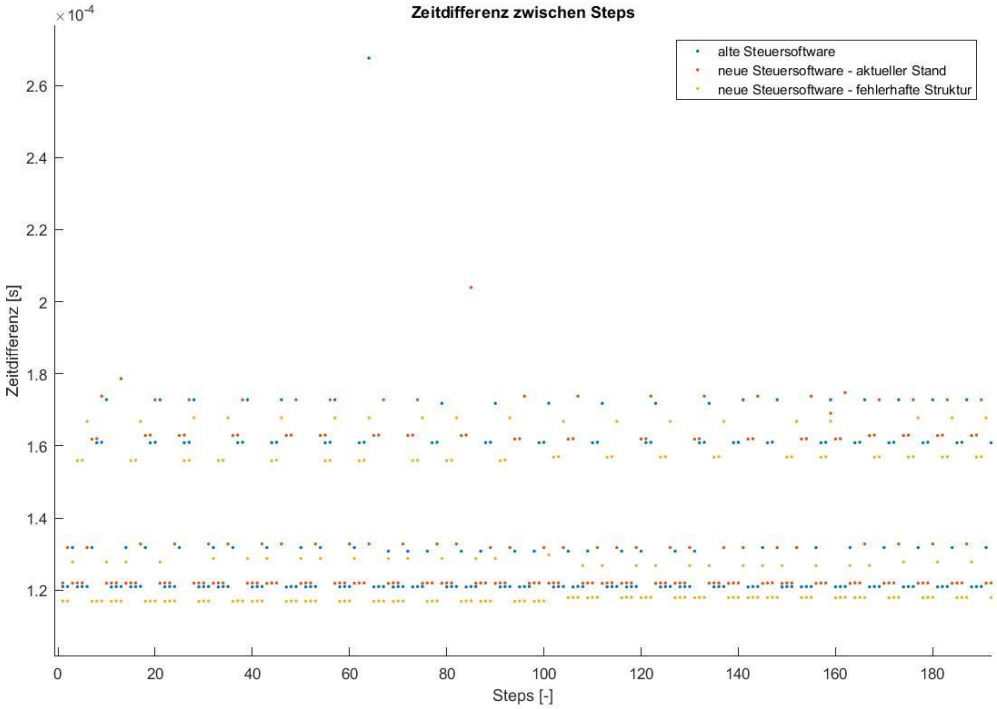


Abb. 6–2: Zeit zwischen Steps für drei Versionen der Steuerung

Auffällig ist dabei, dass die Reihenfolge, in denen diese Anhäufungen auftreten, immer gleich bleibt. Die gelben Punkte stellen die Differenzen der fehlerhaften Programversion mit einer schwankenden FPGA-Übertragung dar, welche im vorrangegangenen Abschnitt erwähnt wurde. Diese besitzt in diesem Diagrammausschnitt tendenziell die niedrigsten Werte, im selben Intervall weiter oben besitzt sie jedoch die mit Abstand größten Zeitdifferenzen. Ein möglichst nahes Beieinanderliegen der Punkte ist also erstrebenswert.

Wieso nun die Werte immer in Gruppen auftreten ist unbekannt, eine Untersuchung hierzu könnte aber wertvolle Erkenntnisse im Hinblick auf ein regelmäßigeres Motorsignal, und damit auf eine präzisere Anlage liefern.

6.2.3 Sonnensimulator

Die Bedienung des Sonnensimulators ist aufgrund der ausgefallenen Bahn um die Anlage herum sehr kompliziert. Zusätzlich ist neben der bloßen translatorischen Bewegung für jede gefahrene Strecke auch immer ein rotatorisches Nachjustieren der Gier- und Nickachse erforderlich. Letztere ist nicht einmal mit einem Motor versehen.

Im Laufe dieser Arbeit wurde bereits mit der Entwicklung eines Steuerungsmoduls begonnen, welches das Nachjustieren der Gierachse automatisch übernehmen kann, und für die Nickachse einen einzustellenden Winkel angibt. Leider konnte diese Funktion nicht vollendet werden. In seinem aktuellen Zustand gibt das Modul für eine Position auf der Ringbahn um die Anlage den einzustellenden Winkel relativ zu einem starren Koordinatensystem an, und gibt entsprechende Werte für die Rotationsachsen des Simulators aus. Eingangswert ist AxisData, Ausgangswerte sind AxisData mit neuen Zielwerten für Achse 11 und der Nickwinkel für die Lampe. Um das Modul einsetzten zu können, muss jedoch noch ein Teil ergänzt werden, der die Rotation des Simulators durch das Fahren von Kurven auf der Bahn berücksichtigt.

6.3 Fazit

In dieser Arbeit wurden nun erfolgreich drei neue Betriebsmodi implementiert: *Read Trajectory File* zum Nachfahren in einem CSV-File definierter Trajektorien, *Sinus Wave Generator* zum Ausführen von benutzerdefinierten sinusförmigen Bewegungen, außerdem ein einfacher, dedizierter *Manual*-Modus.

Der Code für ein Closed-Loop-Feedback wurde implementiert. Von Optitrack aufgenommene Positionsdaten können über eine Schnittstelle in die Steuersoftware übertragen werden. Allerdings fehlt noch eine Koordinatenumrechnung, sodass der *Closed Loop* Trackingmodus aktuell noch nicht funktionsfähig ist.

Es ist nun das Logging der aktuellen Position, Geschwindigkeit und Beschleunigung jeder Achse möglich. Das Abspeichern erfolgt zunächst in eine Binärdatei, welche mithilfe des VIs *TOOL TransformateLogging* in eine Textdatei gewandelt werden kann.

Durch Restrukturieren des Codes und Einführen einer asynchronen Benutzeroberfläche auf dem Host-PC konnte die CPU-Auslastung von rund 99 % auf zwischen 60 % bis 70 % in Volllast reduziert werden. Das Prinzip der Task Separation sollte unbedingt auch in zukünftigen Entwicklungen berücksichtigt werden, um ein stabiles, effizientes Ablaufen der Software gewährleisten zu können. Durch die ergriffenen Maßnahmen stehen nun wieder Systemressourcen für neue Funktionen auf dem cRIO zur Verfügung. Weiterer Vorteil dieser Überarbeitung des Codes ist eine nun deutlich flüssiger zu bedienende Anlage auf dem Host-PC.

Dies ist nicht der einzige Beitrag zu einer benutzerfreundlichen Bedienung. Durch ein neues Design des GUI können nun alle Achsen gleichzeitig gesteuert werden. Das Anzeigen von Informationen wurde auf die nötigsten Daten begrenzt. Ein zusätzliches Bündeln und Hervorheben von Meldungen ist dabei durch das ebenfalls neue Caution and Advisory Display möglich. Um der erhöhten Komplexität durch neue Strukturen und neue Funktionen entgegen zu wirken, wurde ein Startfenster, sowie das erleichterte oder automatische Ausführen von Prozessen und VIs eingeführt.

Es wurden aktualisierte Umrechnungsfaktoren für die Umrechnung von Metern und Radianten nach Motorschritten erarbeitet.

Häufige Referenzfahrten sind nicht erforderlich, beeinflussen die Anlagenpräzision aber auch nicht negativ. Die Präzision der Anlage liegt zwischen 1 mm bzw. 0,3° und 3,5 mm bzw. 8,1°.

A Literaturverzeichnis

[1] A. Fleischner, M. Wilde und U. Walter, „RACOON – a hardware-in-the-loop simulation environment for teleoperated proximity operations“ in *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (I-SAIRAS 2012)*, 2012.

[2] Pelton J., Madry S., Camacho-Lara S., „On Orbit Servicing and Retrofitting“ in *Handbook of Satellite Applications*, New York: Springer, 2015.

[3] M. Dziura, J. Harder und S. Haberl, „Future Technologies for Operating Robots in Space“, *International Astronautical Congress*, 2017.

[4] J. Gutsmiedl, „Design and Implementation of a Closed-Loop Motion Control Interface using Optitrack in a Simulation Environment for Spacecraft Proximity Operations“. Masterarbeit, Technische Universität München, München, 2018.

[5] Florida Institute of Technology, *Orion Research Lab Website.* [Online] Verfügbar unter: https://research.fit.edu/orion/. Zugriff am: Nov. 06 2018.

[6] W. Georgi und E. Metin, *Einführung in LabVIEW: Mit … 157 Aufgaben,* 5. Aufl. 4030 München: Hanser, 2012.

[7] National Instruments, LabVIEW Real-Time Module Help: *Network the User Interface,* 2017.

[8] National Instruments, *cRIO-9014.* [Online] Verfügbar unter: http://www.ni.com/de-de/support/model.crio-9014.html. Zugriff am: Nov. 05 2018.

[9] National Instruments, *cRIO-9024.* [Online] Verfügbar unter: http://www.ni.com/de-de/support/model.crio-9024.html. Zugriff am: Nov. 05 2018.

[10] National Instruments, LabVIEW Real-Time Module Help: *Separate Your Tasks (Real-Time Module),* 2017.

[11] National Instruments, LabVIEW Real-Time Module Help: *Creating Deterministic Applications with the Real-Time Module (Real-Time Module),* 2017.

[12] National Instruments, LabVIEW Real-Time Module Help: *Using the Right Networking Protocol.* [Online] Verfügbar unter: http://www.ni.com/white-paper/12079/en/. Zugriff am: Nov. 06 2018.

[13] NaturalPoint Inc., *Cabling and Wiring.* [Online] Verfügbar unter: https://v20.wiki.optitrack.com/index.php?title=Cabling_and_Wiring#USB_Camera_System. Zugriff am: Nov. 13 2018.

[14] National Instruments, LabVIEW Real-Time Module Help: *RT FIFO Functions,* 2017.

[15] National Instruments, „LabVIEW 2017 Help“, 2017.

[16] NaturalPoint Inc., *Data Export: CSV.* [Online] Verfügbar unter: https://v20.wiki.optitrack.com/index.php?title=Data_Export:_CSV. Zugriff am: Nov. 14 2018.

B Anhang

Folgende Daten sind dieser Arbeit beigelegt.

- Ausarbeitung
  - Ausarbeitung als .docx und .pdf
  - Sämtliche verwendete Bilder
- Literatur
  - Alle als PDF verfügbaren Quellen
- Messdaten
  - Alle Messdaten in Rohformat sowie Ergebnisse
- Programmcode
  - Motion Control Projekt mit sämtlichen Dateien
  - Quellcodepaket für cRIO
  - Optitrack Interface Projekt mit sämtlichen Dateien
- Software
  - Logic Setup
- Sonstiges
  - Tabelle Achsendaten laut altem PathPlanner21
  - nicht einsatzbereites VI: Automatic Sun Movement