Lehrstuhl für Raumfahrttechnik
Prof. Prof. h.c. Dr. Dr. h.c. Ulrich Walter

Technische Universität München

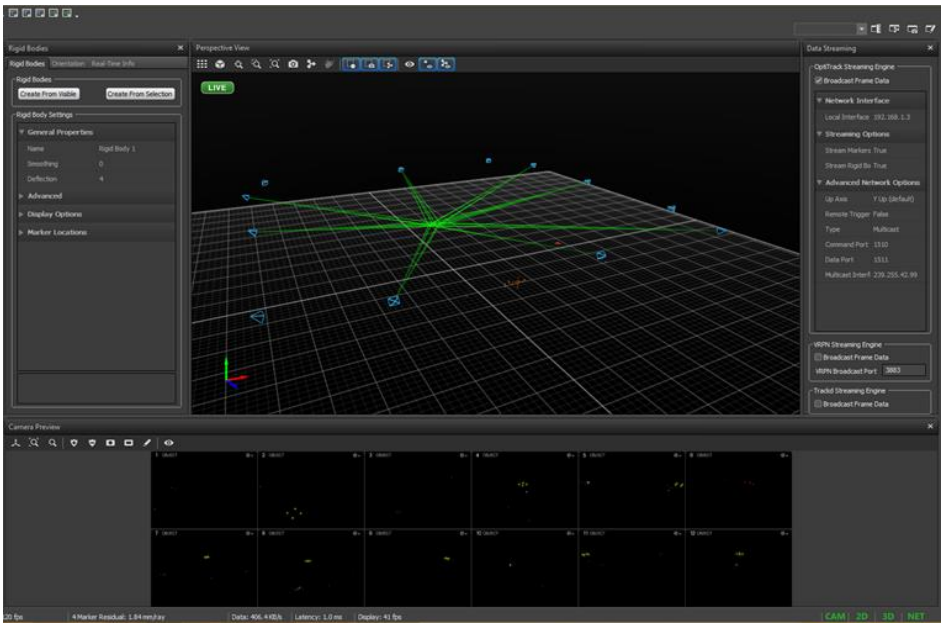
Masterarbeit

Design and Implementation of a Closed-Loop Motion Control Interface using Optitrack in a Simulation Environment for Spacecraft Proximity Operations

RT-MA 2018/04

Author:

Johannes Gutsmiedl



Betreuer: Martin Dziura
Lehrstuhl für Raumfahrttechnik / Institute of Astronautics
Technische Universität München

Acknowledgments

I would like to thank Dr. Markus Wilde and Martin Dziura for providing me the opportunity to work on this project at FIT and for always providing great advice and feedback.

I would also like to thank my family and friends for their support during this thesis and my entire time at university. Special thanks goes to WARR and its members for making the past years such an enjoyable and rewarding experience.

Zusammenfassung

Das Florida Institute of Technology (FIT) und die Technische Universität München (TUM) betreiben Hardware-Simulatoren für ferngesteuerte Nahbereichs- und Dockingoperationen mit Satelliten. Beide Simulatoren benutzen eine gemeinsame Steuersoftware auf Basis von National Instruments Labview. In dieser Arbeit wurde diese Software komplett überarbeitet und mehrere wichtige neue Features hinzugefügt. Die neuen Features beinhalten eine neue Benutzeroberfläche, eine interne Restrukturierung des Programms, zwei neue Betriebsmodi und eine Funktion zur Datenaufzeichnung. Zusätzlich wird ein geschlossener Regelkreis mit absoluter, externer Positionsbestimmung implementiert. Diese Upgrades vereinfachen die Bedienung des Systems gerade für neue Nutzer und eröffnen neue Möglichkeiten für Experimente. Die Arbeit fokussiert sich auf den Hardwaresimulator am FIT, enthält aber auch eine Liste der nötigen Änderungen für eine Nutzung der neuen Software an der TUM.

Die Änderungen an Benutzeroberfläche und Softwarestruktur basieren auf Feedback von bisherigen Nutzern und auf dem Labview Style Guide. Die Software wird dabei in einem Top-Down-Ansatz entwickelt, wobei zunächst eine Anforderungsdefinition durchgeführt wird. Anschließend wird die Nutzeroberfläche festgelegt und dann die Subsysteme von höherer hin zu niedrigerer Ebene eingebaut. Schließlich werden die Anforderungen durch Tests verifiziert. Es wird auch ein Beispielexperiment beschrieben, in dem einige der neuen Features zum Einsatz kommen. Durch diese Tests wird gezeigt, dass die neue Software ausreichend weit entwickelt ist um von Nutzern für andere Projekte am Hardware-Simulator eingesetzt zu werden.

Das komplexeste neue Feature ist der neue geschlossene Positionsregelmodus. Er nutzt bereits vorhandene optische Positionierungssysteme an beiden Anlagen um den Positionsvektor des Systems in einem externen, absoluten Koordinatensystem zu messen. Die neue Software führt diesen Vektor zurück in die Logikschleife für die Positionierung der Anlage und nutzt die Messdaten um die intern errechnete Position periodisch zu korrigieren. Im bisher genutzten inertialen Positionierungsmodus ergibt sich bei längerer Operation der Anlage eine erhebliche Drift. Wie erwartet kann dieses Problem durch den geschlossenen Regelkreis eliminiert werden. Zusätzlich ermöglicht die externe Positionsbestimmung auch eine exakte Bestimmung der Positioniergenauigkeit der Anlage. Diese liegt bei etwa 1 mm in Translation und 0.1 ° in Rotation.

Abstract

The Florida Institute of Technology (FIT) and Technical University of Munich (TUM) both operate hardware-in-the-loop simulators for remotely operated satellite proximity flight and docking. These simulators share a common operating software implemented in National Instruments Labview. This thesis describes the implementation of several important upgrades to this software. Namely, the upgrades include a complete rework of the user interface and internal software structure, two new operating modes, and data logging functionality. In addition, a closed loop tracking circuit with external absolute position measurement and feedback is implemented. The upgrades allow much faster user training and enable new options for experiments. Most work in this thesis centers on the hardware simulator at FIT, but instructions are provided for implementing the same changes for the system at TUM.

The user interface and software structure changes are based on feedback from previous operators as well as the Labview style guide. A top-down process is used for development, with a requirement derivation first and then successive development starting with the interface and progressing through subsystems. The requirements are then verified by testing, and finally an example experiment utilizing some of the new software capabilities is described. Through this testing, the software is shown to be in a sufficiently developed state to be ready for use in future projects incorporating the hardware-in-the-loop simulator.

The most complex new feature is the closed loop tracking mode. It utilizes a pre-existing optical tracking system available at both facilities to determine the system state vector in an absolute external coordinate system. The new software provides a way to feed this position data back into the control loop and use it to apply corrections. The previously used inertial tracking mode causes slow mechanical drift over long-term use. As expected, this problem is shown to be removed with the new closed-loop solution. The external measurement also allows determining the positioning accuracy of the system. It is found to be approximately 1 mm in translation and 0.1° in rotation.

Table of Contents

List of Figures

List of Tables

Symbols

Symbol	Unit	Description
A	[m]	Amplitude
f	[Hz]	Frequency
t	[s]	Time
x	[m]	Position
ϕ	[rad]	Roll Angle
θ	[rad]	Pitch Angle
ψ	[rad]	Yaw Angle
q_x	[-]	Quaternion

Abbreviations

LRT	Lehrstuhl für Raumfarttechnik
TUM	Technical University of Munich
FIT	Florida Institute of Technology
ORION	Orbital Robotic Interaction, On-orbit Servicing, and Navigation
RACOON	Real-Time Attitude Control and On-Orbit Navigation Laboratory
EPOS	European Proximity Operations Simulator
COTS	Commercial Off The Shelf
DLR	Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)
POSEIDYN	Proximity Operation of Spacecraft: Experimental hardware-In-the-loop Dynamic simulator
ADAMUS	Advanced Autonomous Multiple Spacecraft
SOSC	Space Operations Simulation Center
RSGS	Robotic Servicing of Geosynchronous Satellites
DARPA	Defense Advanced Research Projects Agency
CRIO	Compact Reconfigurable Input/Output Module
FPGA	Field Programmable Gate Array
USB	Universal Serial Bus
UDP	User Datagram Protocol
VI	Virtual Instrument
(G)UI	(Graphical) User Interface
SDK	Software Development Kit
DLL	Dynamic Link Library
XML	Extensible Markup Language
CSV	Comma Seperated Values
WLAN	Wireless Local Area Network
IP	Internet Protocol

1 Introduction

Remote docking of satellites is a topic currently being researched at many institutions around the world. Two of them are the Technical University of Munich (TUM) in Germany and the Florida Institute of Technolgy (FIT) in the United States. The universities have an ongoing cooperation between their projects ORION (Orbital Robotic Interaction, On-orbit Servicing, and Navigation) at FIT and RACOON (Real-Time Attitude Control and On-Orbit Navigation Laboratory) at TUM (Figure 1-1).

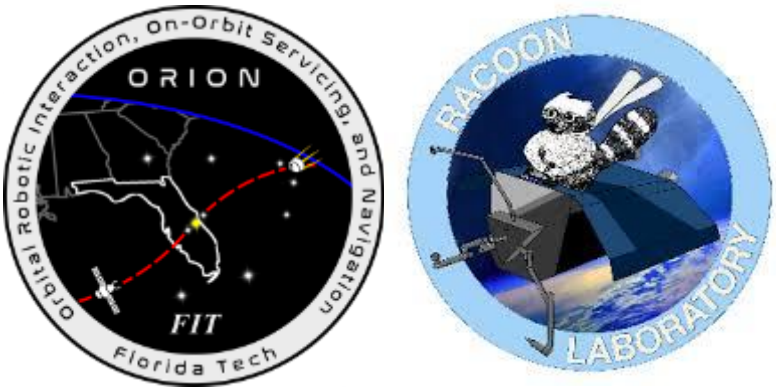


Figure 1-1 Mission Patches for ORION (Left) and RACOON (Right)

Both projects operate facilities for simulating proximity operations of spacecraft in hardware, used for sensor and algorithm testing as well as human-in-the-loop experiments. The simulators share a common development history and use the same software to operate. The goal of this thesis is to add a closed loop position control mode to this software which uses an external optical positioning system to obtain the feedback necessary to close the control loop. In addition, the software is reworked and updated with a new user interface and some additional features.

1.1 Literature Research

Many laboratories around the world operate hardware simulators for research ito spacecraft proximity operations and docking. To correctly evaluate the results from this thesis, it is of interest to consider the performance of these systems regarding positioning and position determination. The existing facilities can be broadly split into two categories: those using mechanical guidance systems like gantries and robot arms and those using autonomous vehicles on air bearings, usually controlled by cold gas thrusters. ORION and RACOON fall in the former category, although ORION also includes a flat floor for air bearing vehicles which is beyond the scope of this thesis. Most other mechanically guided simulators utilize commercial off the shelf (COTS) industrial robots. Examples for such facilities are the German Aerospace Center (DLR)'s European Proximity Operations Simulator (EPOS) [1] and the National Aeronautics and Space Administration (NASA)'s Space Operations Simulation Center (SOSC) [2]. They typically do not use external position determination systems and instead rely on the robot arm's internal controls. COTS robots typically achieve position repeatability on the order of 0.1 mm [3].

ORION and RACOON do not use COTS robots and instead rely on custom-built gantry-based guidance systems. While this approach is lower cost and allows more customization for both hardware and software, the custom-built systems also tend to

be less accurate in position repeatability than commercial robots. To solve this issue, both simulators have been equipped with external optical sensor systems based on infrared cameras, which enables precise monitoring and should be able to allow precise control of the simulator position. This approach is typically used in air bearing based docking simulators such as the Naval Postgraduate School's Proximity Operation of Spacecraft: Experimental hardware-In-the-loop Dynamic simulator (POSEIDYN) [4] or the Advanced Autonomous Multiple Spacecraft laboratory (ADAMUS) at Rensselaer Polytechnic Institute [5]. Both laboratories use different optical tracking systems. POSEIDYN achieves a three-sigma tracking error of about 3 mm in translation and 0.2° in rotation using a 10-camera VICON system [4], while ADAMUS only publishes an accuracy of 1-5 mm using 8 PhaseSpace cameras [5]. Positioning accuracy for air bearing vehicles is not comparable to mechanically guided systems, with a mean position error of 0.3 m and a mean attitude error of 20° for POSEIDYN [4]. ORION and RACOON are the only satellite docking simulators using Optitrack. A laboratory at Universitá de Bologna using Optitrack for a different but comparable robotics application reports position accuracy of 0.3 mm and a rotation tracking error below 0.06° [6]. The publication does not contain detailled specifications of camera count and setup. These findings provide a benchmark for ORION and RACOON.

1.2 Problem Statement

Both the ORION and RACOON hardware simulators are equipped with optical camera-based tracking systems which enable determination of the absolute positions of objects within the facilities. The goal of this thesis is to use these tracking systems to create a closed feedback loop for position control, enabling higher positioning accuracy and preventing drift during operation compared to the previous open-loop system based on counting motor steps. In addition, the software used to operate both facilities is to be updated to make it more user friendly and introduce additional capabilities such as data logging and a sine-wave motion mode for testing.

1.3 Methodology

The bulk of work in this thesis was undertaken at FIT in the ORION laboratory. Section 2 provides some necessary background information. The first step in the thesis is the development of an interface for streaming Optitrack data to a Labview application, which is meant to be as generic as possible to enable reuse in other projects. This work is described in section 3. The larger part of the thesis is the reworking of the hardware simulator operating software to integrate closed loop tracking and other features and to rework the user interface. Section 4 documents this work. Finally, section 5 describes the additional work required to get the developed software to run on RACOON, and section 6 summarizes the results of the thesis and gives an outlook on further work required.

2 Background Information

This section provides some background necessary to understand the work executed for this thesis. This includes a brief description of the reasoning and mission concept for on-orbit servicing (section 2.1). The hardware simulators RACOON and ORION used at TUM and FIT respectively, as well as the Optitrack system are presented in sections 2.2-2.4. Section 2.5 provides an overview of the existing Labview code used to run both hardware simulators, which formed the basis for the work in this thesis.

2.1 On-Orbit Servicing

On-orbit servicing generally has three applications:

- Repair of damaged satellites
- Life extension of satellites (generally refueling)
- Space debris removal

Because of the complex nature of these mission scenarios and the large number of unknown parameters involved, successful on-orbit servicing has so far only been performed on crewed flights using the Space Shuttle. In recent years, however, multiple research efforts have focused on remotely controlled robotic servicers. Important projects include the Japanese mission ETS-VII (Engineering Test Satellite No. 7), launched in 1997, and the Orbital Express mission flown by NASA in cooperation with the Defense Advanced Research Projects Agency (DARPA) in the United States in 2007. Both of these missions demonstrated capture and docking operations in orbit. Figure 2-1 shows a concept image for the Robotic Servicing of Geosynchronous Satellites (RSGS) mission currently planned by DARPA [7].

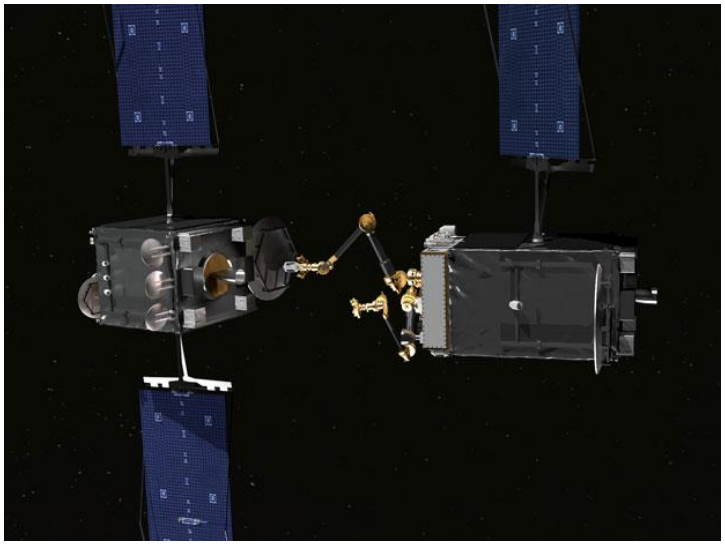


Figure 2-1 Example for a Remote On-Orbit Servicing Mission [7]

Challenges for robotic on-orbit servicing include the mechanical capture and interfacing with non-cooperative targets which have not been designed with servicing in mind, as well as the selection of an effective sensor suite to provide the teleoperator with all necessary information while meeting the cost, weight, and durability requirements of a typical space mission. Researchers working on solving these problems typically use computer simulations as well as hardware simulators, such as those available at FIT and TUM.

2.2 ORION

The ORION laboratory at FIT uses several experiment setups to study spacecraft proximity operations and docking. This thesis concerns the ORION gantry system, a hardware simulator designed as a two-dimensional testbed for sensors and teleoperation procedures. The gantry consists of a so-called chaser, simulating the servicing satellite, which can move in two translational (X, Y) and two rotational axes (pitch ϕ, yaw ψ) as well as a target which is stationary but can be rotated in pitch and yaw. The axes are all driven by stepper motors. All axes have magnetic reference switches used to set their respective zero positions, and all axes except the infinitely rotating yaw axes are also equipped with mechanical end switches. The target carries a sub-scale mockup of a satellite, while the chaser is set up as a multi-purpose platform for mounting sensors and other test hardware. An on-board PC is available to be placed on the chaser as a sensor interface. The space between the translational axes is covered by an acrylic flat floor allowing the operation of vehicles with air bearings in concert with test articles on the gantry. Figure 2-2 shows the current configuration of the ORION gantry system.

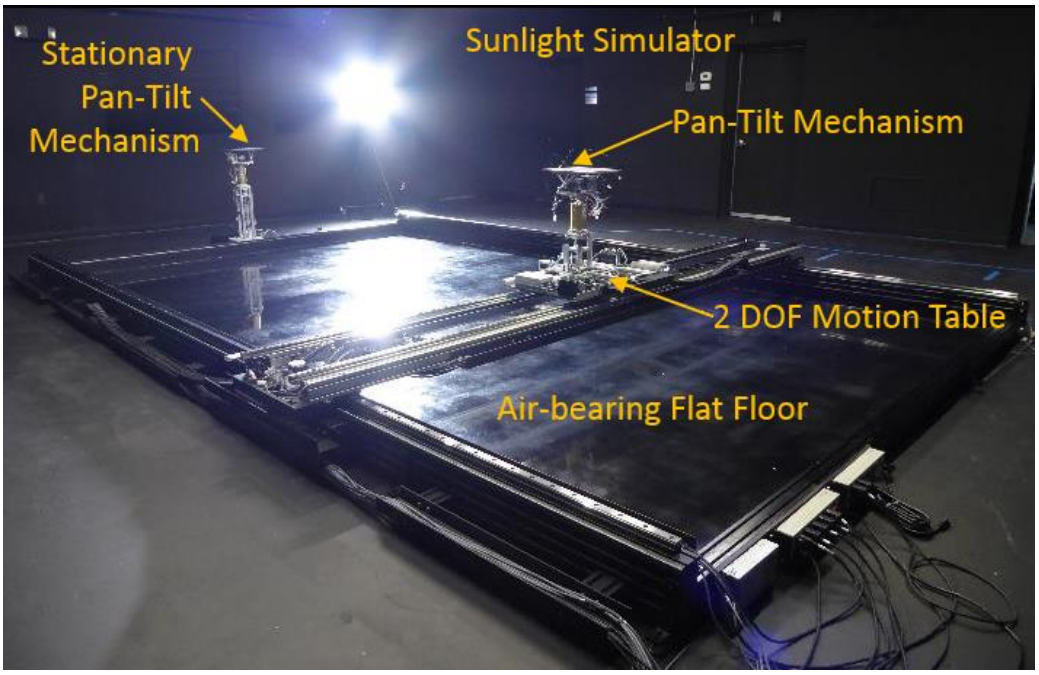


Figure 2-2 ORION Gantry Hardware Configuration [8]

The gantry is controlled by a National Instruments Compact Reconfigurable Input/Output Module (CRIO) system, which features both a microcontroller with a real-time operating system and a Field Programmable Gate Array (FPGA). All sensors and actuators are connected to the CRIO, which forwards data to and receives commands from the user through a PC running a Graphical User Interface (GUI) implemented in National Instruments' proprietary Labview programming environment. A second computer is set up with multiple screens for teleoperation experiments and communicates with the operating PC, and a third one is responsible for collecting and forwarding data from the Optitrack positioning system (section 2.4). All three computers are located in a control room next to the main laboratory. The control room setup is shown in Figure 2-3. A layout of the entire laboratory is shown in Figure 2-4.

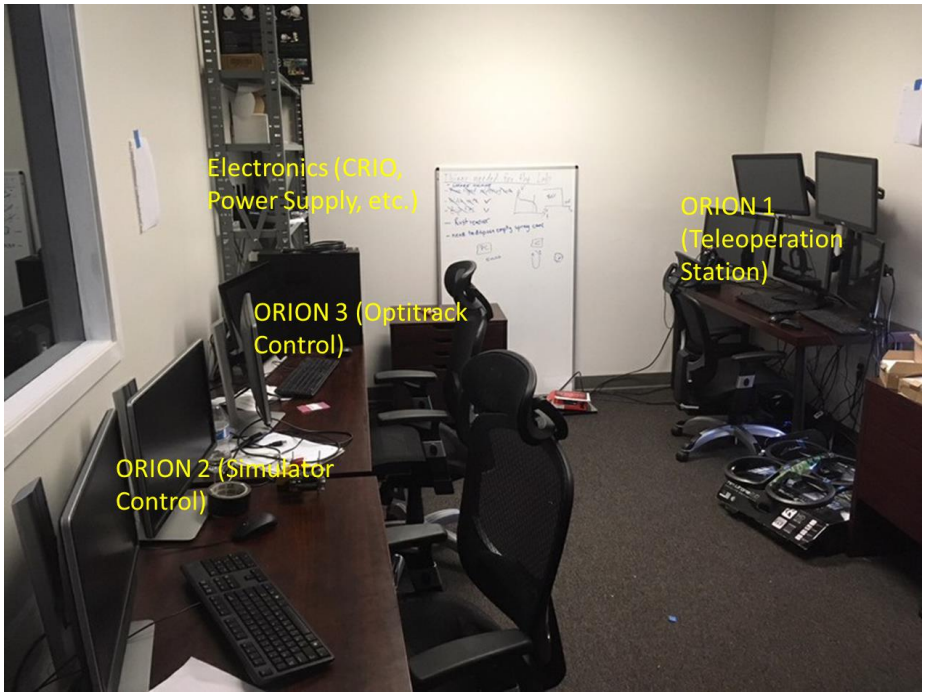


Figure 2-3 ORION Control Room

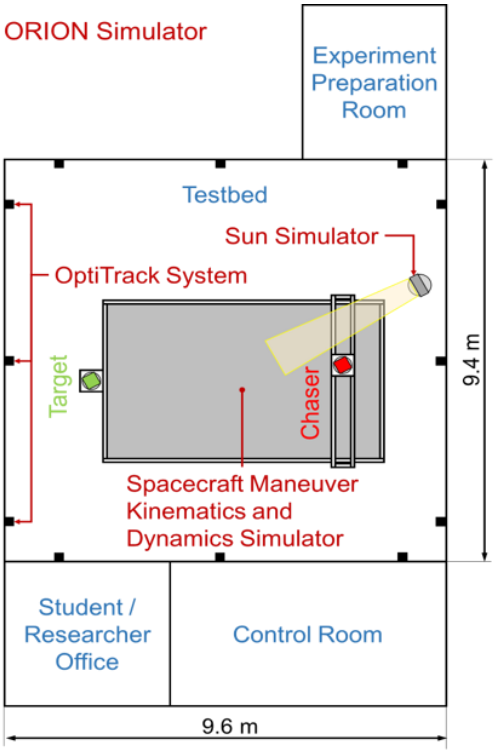


Figure 2-4 ORION Laboratory Top Down View

2.3 RACOON

The ORION Gantry system is based on the original configuration of the RACOON-Lab (Real-Time Attitude Control and On-Orbit Navigation Laboratory) hardware simulator at TUM [9]. While RACOON originally featured the same two-dimensional six-axis concept that ORION now uses, it has since been upgraded to a three-dimensional configuration as shown in Figure 2-5.

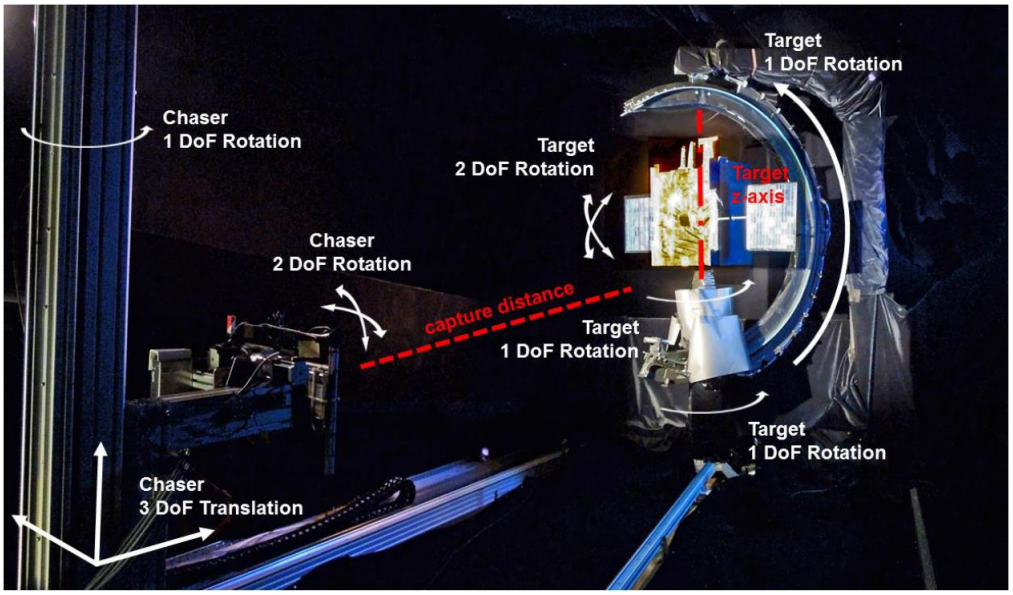


Figure 2-5 RACOON-Lab Hardware Configuration [9]

The chaser can move in all six spatial degrees of freedom, while the target can only move in the vertical axis and has five rotational degrees of freedom to avoid gimbal lock. In addition, the facility features movable lighting sources matched to the solar and Earth albedo spectrum. Figure 2-6 provides a top-down view of the simulator, with the mountings for target and servicer as well as the orbital rails carrying the sun and albedo lamps clearly visible.

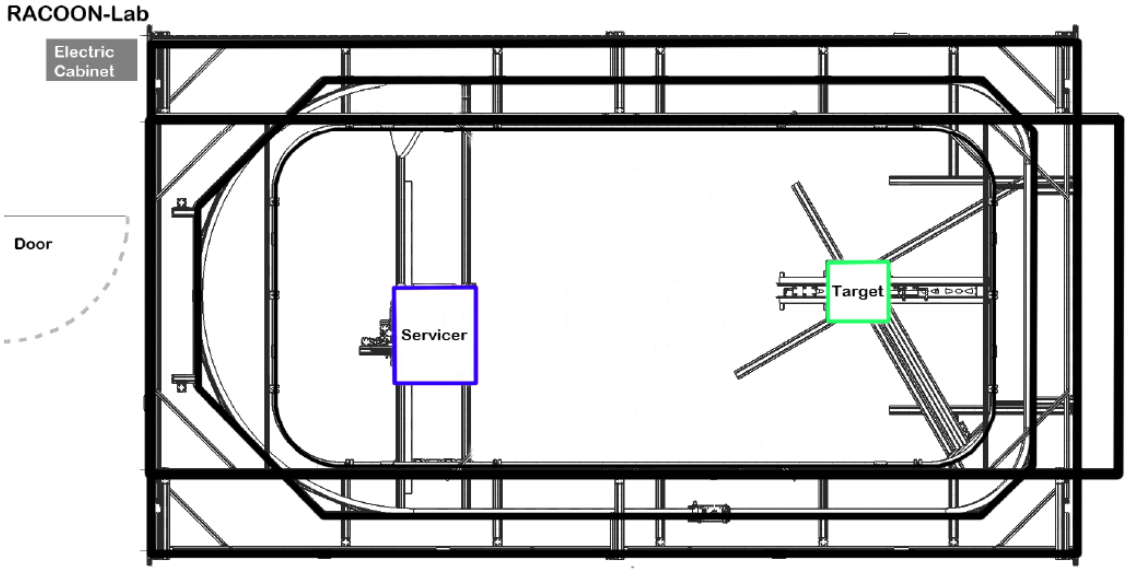


Figure 2-6 RACOON-Lab Top Down View [9]

As with the ORION gantry, the chaser is a multi-purpose sensor platform while the target usually carries a sub-scale model of a satellite. Despite the mechanically much more complex setup, RACOON still uses the same Labview software as ORION, with a CRIO real-time system and FPGA controlling the facility. The network setup is otherwise the same as in the ORION lab, with one computer each for commanding the hardware simulator, teleoperations, running the chaser's on-board sensors and evaluating and forwarding Optitrack data. Figure 2-7 shows a comparison between the conceptual configurations of ORION and RACOON.

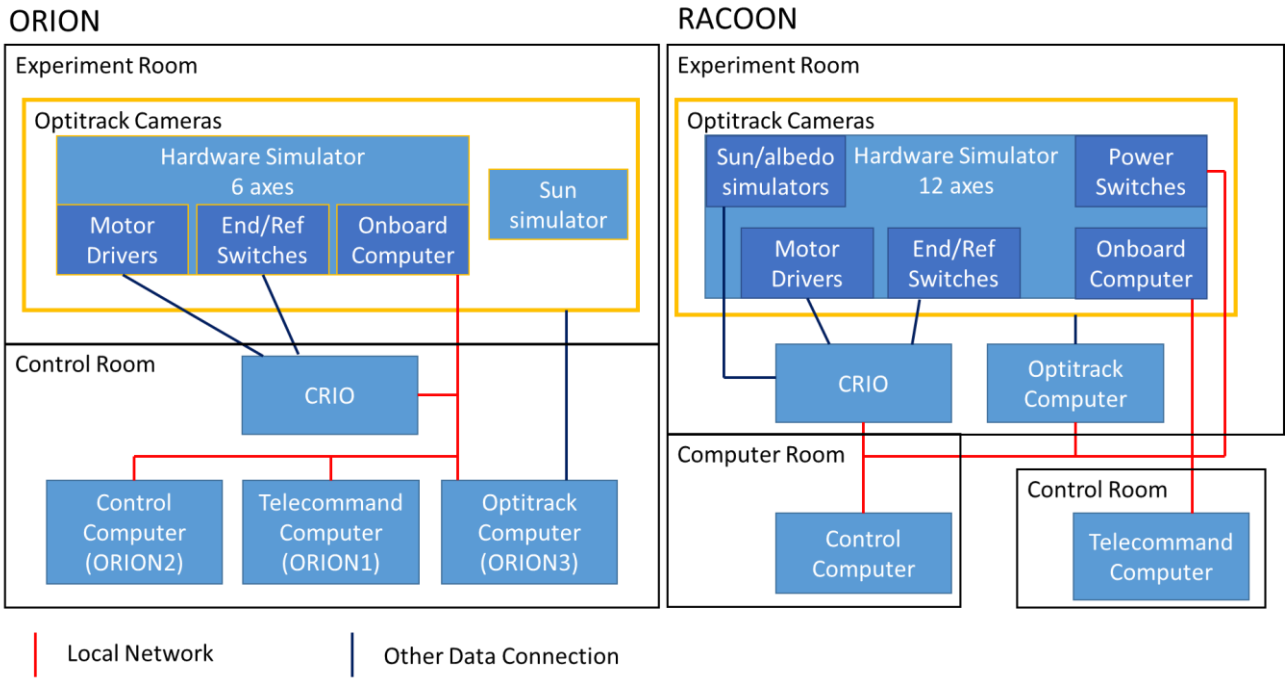


Figure 2-7 System Diagram Comparison between ORION and RACOON

2.4 Optitrack

Both ORION and RACOON use an optical tracking system called Optitrack produced by NaturalPoint to determine the position of moving objects in a fixed reference frame. Optitrack was originally developed for motion capturing in the entertainment industry. The system uses an array of infrared cameras imaging a scene from different angles. Tracked objects need to carry spherical reflective markers. Figure 2-8 shows the Chaser Interface Plate for experiment mounting on the ORION gantry with four tracking markers placed on it. If the camera positions relative to each other are known, this allows reconstruction of the marker positions in three-dimensional space. Optitrack allows for very simple setup by providing a calibration procedure that allows the cameras to determine their own relative positions, so it is not necessary to place them at predefined locations. Documentation for Optitrack is available on NaturalPoint's Wiki site [10].

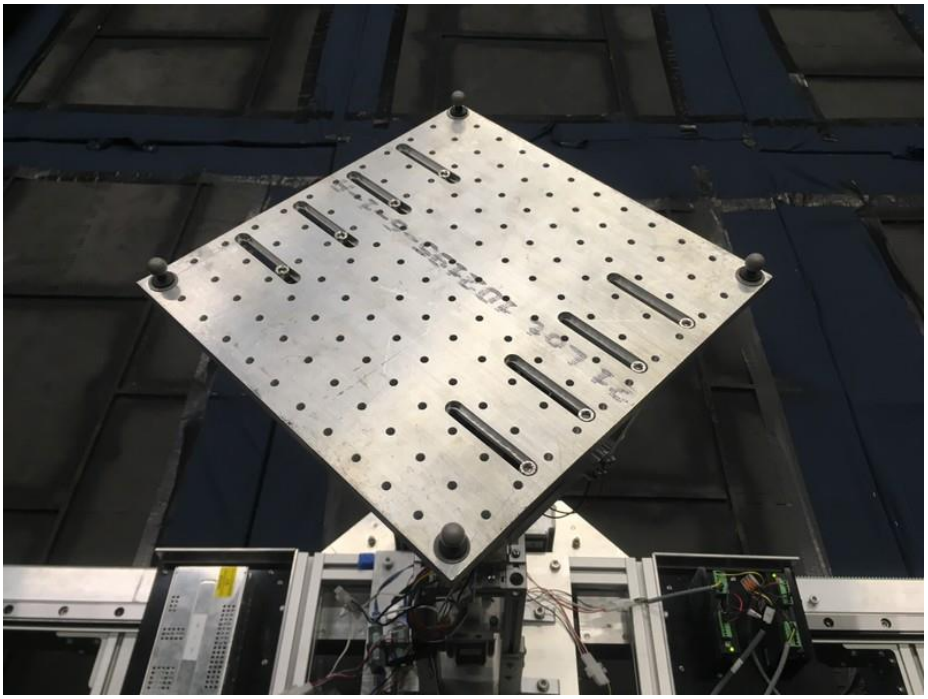


Figure 2-8 ORION Chaser with Tracking Markers

The Optitrack setups on RACOON and ORION are quite different. RACOON uses eight Flex3 USB cameras, with two sets of four cameras each connected to an Optihub unit, which ensures correct synchronization of the images [11]. Camera positioning is limited by the length maximum of 5 m for USB cables. This causes coverage problems in some areas of the experiment setup, particularly close to the target. ORION, meanwhile, uses 12 Prime 17W network cameras all connected to a network switch via Ethernet. These cameras do not suffer from the same cable length limitation as the USB cameras, so all twelve can be set up at equal spacing around the top corner of the laboratory, covering the entire room very well. An overview and comparison of both setups is provided in Table 2-1.

Table 2-1 Optitrack Setup Comparison between RACOON and ORION [12]

System	**ORION**	**RACOON**
		
Camera Type	Prime 17W	Flex 3
Number of Cameras	12	8
Data Connection	Ethernet	USB

In both ORION and RACOON, a dedicated computer handles data from the Optitrack system using the proprietary Motive software from NaturalPoint. A typical screenshot of Motive is shown in Figure 2-9. It provides a GUI showing all tracked markers in three-dimensional space and provides many post-processing options. For robotics applications, the most important one is the capability to define a rigid body object from multiple markers and thus track the object's position and rotation in space. Both RACOON and ORION typically use two rigid bodies, one each for chaser and target. In the ORION lab the first tracked rigid body is always the chaser and the second one the target. It should be noted that while the position of the body is provided in absolute coordinates in the Optitrack reference frame, the rotations are relative to the body's orientation when it was created. Data from Motive can be streamed to other computers on the network using Natural Point's proprietary NatNet protocol. In the ORION laboratory, all computers are connected on a local network without a firewall. Optitrack data is broadcast via UDP in multicast mode, allowing multiple receivers to listen to and use the data at the same time. The normally used broadcast settings in Motive are: IP address 192.168.1.3, command port 1510, data port 1511.

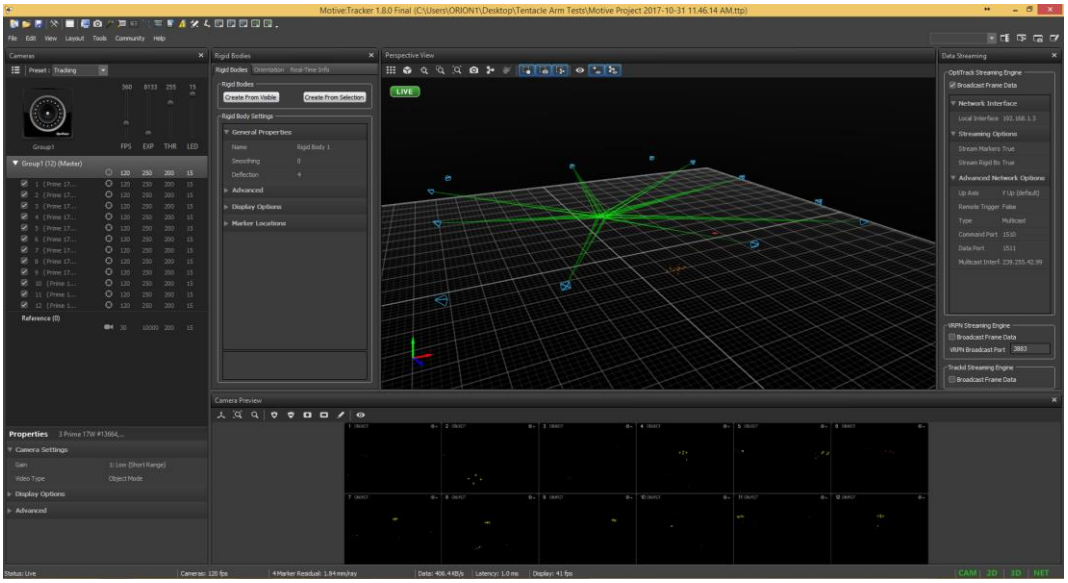


Figure 2-9 Motive User Interface

2.5 Heritage Labview Code Overview

The goal of this thesis is to update the Labview software used in both ORION and RACOON by adding new features and an improved user interface. As the original software is mostly undocumented, this section provides an overview of its design and functionality. The Labview software runs on a National Instruments CRIO using a real-time operating system, with the GUI and the Labview code itself visible to the user on a desktop PC. Labview internally handles Ethernet communications between the PC and the CRIO. The CRIO microcontroller forwards commands to an FPGA, which in turn actuates the hardware simulator's motor controllers and feeds data from magnetic reference switches and mechanical limit switches back to the microcontroller. In addition, the software can communicate with an external computer via a UDP (User Datagram Protocol) network connection for teleoperation experiments. Figure 2-10 shows a component diagram of this system.

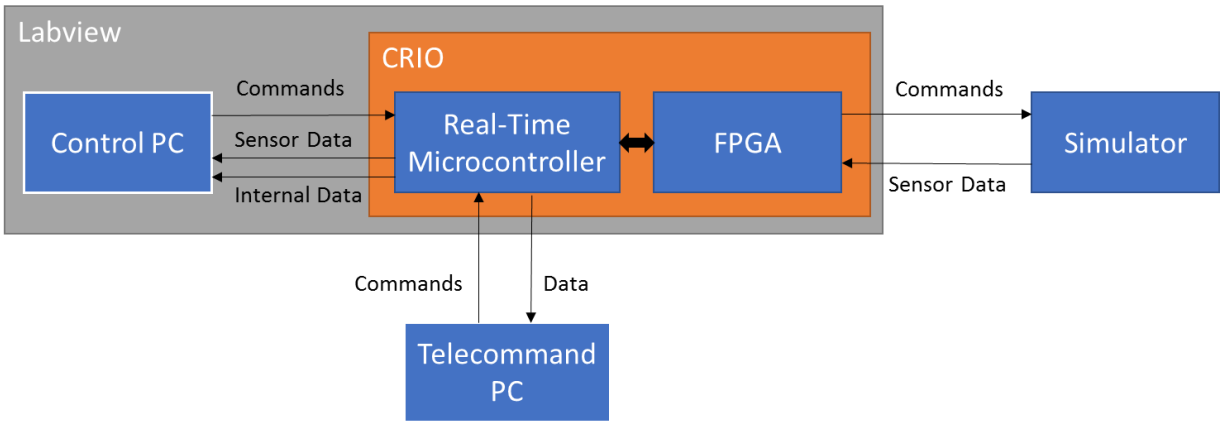


Figure 2-10 Simulator Control System and Data Flow Diagram

The real-time code itself is contained in one single VI. Upon startup, an initialization sequence is executed to set all variables to their correct starting values. Once this is complete, the main program loop runs as a time-controlled loop containing a large flat sequence at a rate of 100 Hz performing all main functions of the program until the user terminates execution. A flat sequence is a Labview code structure consisting of several frames which are executed sequentially. Each frame waits until all operations within the previous frame have finished before it executes. The necessary variables for the system are stored in three different clusters which are always visible to the user in the GUI and get updated once per main loop cycle. These are:

- Axis State: contains all information that differs for each axis, including current and targeted state vector (position/velocity/acceleration), operating mode, limits, and conversion factors. State vectors and other data related to distances and angles are given in motor steps, and the user is responsible for conversion to and from other units.
- Network: contains all information required for the UDP connection to the telecommand PC, such as IP address and the internal Labview connection reference, but also internal data like timers and the actual received data.
- Realtime: contains all the data transferred to and from the FPGA in the current cycle as well as additional information for synchonization etc. Transferred data includes movement commands in the format of set points as well as the status of magnetic and mechanical switches on the gantry.

All calculations involving the axis state are performed by iterating through all axes in for-loops, and the data is displayed in the GUI in a cluster array. The iterative approach enables easy adaptation if axes are removed or added. Due to the large size of the axis state cluster, only one axis is visible to the user at any time, and the user needs to page through the array to view different ones. Figure 2-11 shows part of the original GUI (it does not fully fit on the screen without scrolling).

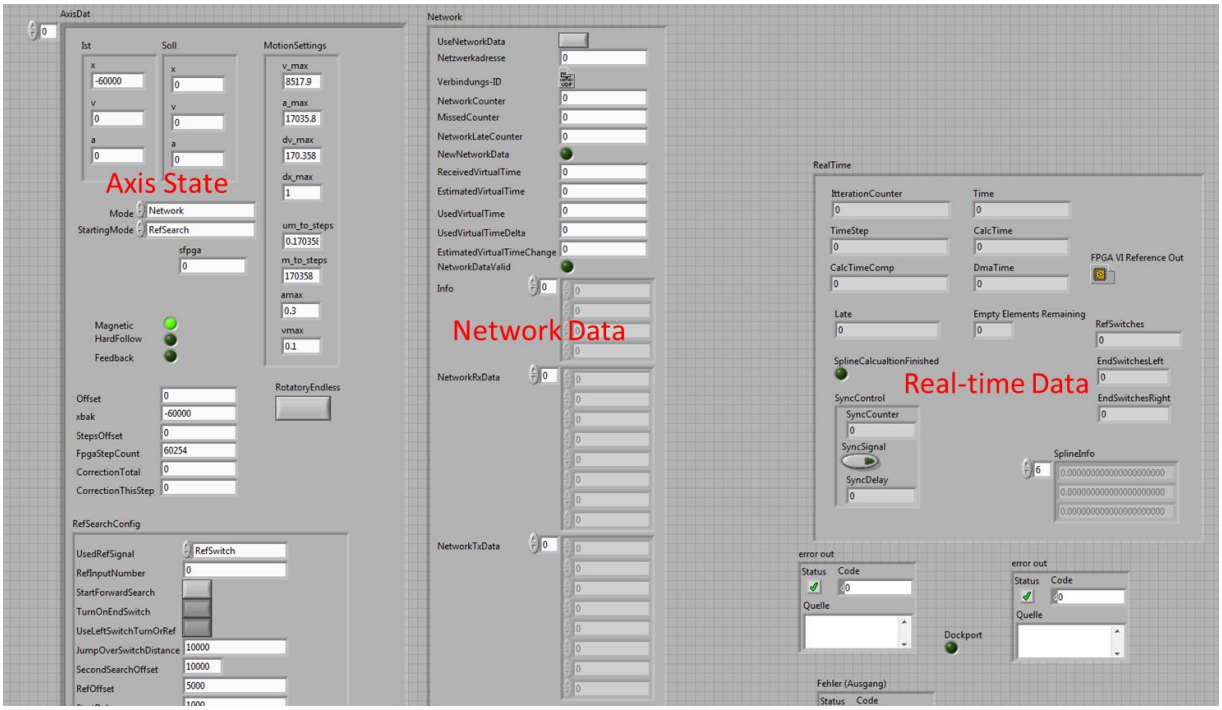


Figure 2-11 Heritage Labview GUI

The software supports the following operating modes, which are set for each axis individually, i.e.different axes can run in different modes:

- Network: moves the hardware simulator based on commands received through the UDP connection. This mode is also used for manually moving the system by directly entering target positions.
- Reference Search: executes an automated sequence to slowly drive the axis until a reference switch is encountered, then sets the switch position as a new zero position. Positioning is otherwise completely open-loop and achieved by counting motor steps.
- Stopping: when engaged, slows the axis down to full stop in a controlled manner, then prevents further movement.
- Docked, PathDemo: the modes exist, but no functionality was implemented

As the entire code is contained in a single large VI, it is hard to display in images. A top-level diagram is shown in Figure 2-12. On each program cycle, the main VI executes the following sequence of functions:

- Timing check: ensures the last cycle was finished within its allowable execution time of 0.01 s, if not, the function increases a counter.
- FPGA Synchronization: checks if the FPGA finished its last cycle on time, calculates any delays.
- FPGA Data Transfer: sends movement data for the next main program cycle to the FPGA and receives reference and limit switch states.
- Network Handling: Exchanges command and telemetry data with the telecommand PC via the UDP connection.
- Feedback Calculation: corrects the current position for rounding errors introduced by the splining process based on information returned fom the FPGA.

- Limit Switches: decodes the limit switch status information from the FPGA and puts any axis that has hit a limit switch into stopping mode.
- Mode Handling: implements all operating modes in a switch-case structure.
- Set Point Handling: this mode contains another flat sequence with multiple steps. It uses the current and target state vector of each axis in addition with velocity and acceleration limits to define a set point for where to move the axis within the next time step. It then calculates coefficients for a spline interpolation between the current state vector and the desired result at the end of the current timestep and passes them on to the FPGA. The previously calculated feedback is applied in this step, after the setpoint calculation is done.

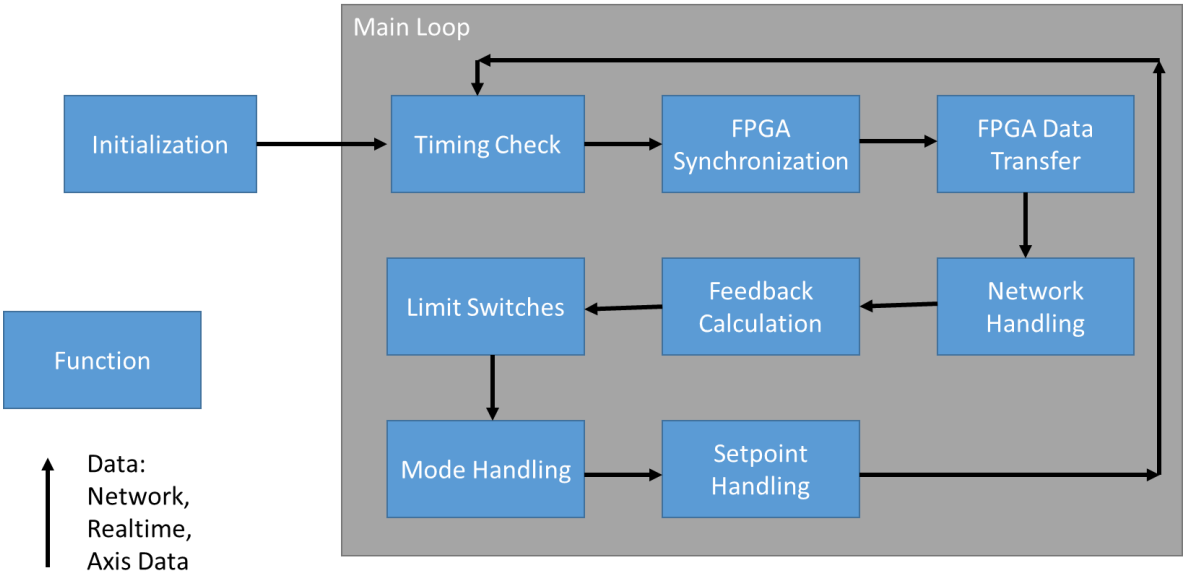


Figure 2-12 Heritage Labview Code Top-Level Flow Chart

The FPGA runs its own loop at a frequency of 1000 Hz, so it executes 10 times for every main program loop. The FPGA receives the spline data from the real-time system and uses them to interpolate the target position for the stepper motors for each of its own time steps, then commands the motor controllers to execute these steps. The FPGA software is precompiled and loaded onto the CRIO during the initialization phase of the main program. No changes to the FPGA software were made for this thesis.

3 Optitrack-Labview-Interface Development

One of the core new features to be implemented in this thesis is an Optitrack interface for the Labview software used to operate the ORION and RACOON hardware simulators. This interface can then be used to control the system in a closed loop with absolute position feedback, instead of the previous open loop mode relying on counting motor steps. This section describes the development of the Optitrack interface with section 3.1 giving a conceptual overview and section 3.2 going into more detail on the implementation.

3.1 Conceptual Overview

As mentioned in section 2.4, Optitrack allows streaming data over a network. It uses the proprietary NatNet protocol for this communication. Natural Point offers a software development kit (SDK) for building applications that receive data via this protocol [13]. The core of the SDK is a precompiled C++ dynamic link library (DLL), which contains all necessary functions, as well as several header files declaring the functions and variables used. Implementation examples are also provided. The SDK can be used to create a C++ program which acts as a client in the NatNet connection, while the Motive software acts as the server broadcasting data. Motive is set up to calculate position and rotation data for two tracked rigid bodies, one for the chaser and one for the target. Positions are transmitted in meters, with rotations in unit quarternions.

The SDK can be used to create C++ functions which are then called within Labview to feed data into the Labview VI. This is complicated by the fact that the NatNet library requires a large amount of Windows specific functions which are not available on the CRIO real time system. It is thus necessary to create a Labview VI that is independent of the hardware simulator operating software and runs directly on either the Optitrack PC (with Motive set to broadcast on a local loopback) or on the hardware simulator control PC. Both methods provide equal results (similar latency), and the latter is used in this thesis as it is often helpful to have access to the interface VI on the control computer for troubleshooting. This VI receives, filters and converts the data and maps it to the correct coordinate system in case the Optitrack coordinate system is different from the simulator coordinate system. The data is then passed on to the CRIO via a UDP connection. Latency in the local network is typically well below 1 ms, so no significant delay is introduced by this. The networking concept for the Optitrack interface is illustrated in Figure 3-1.

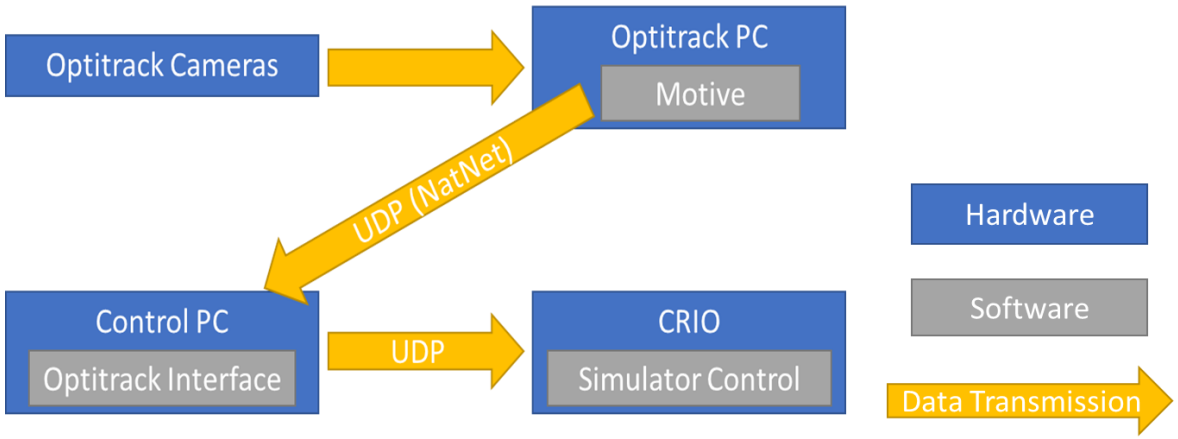


Figure 3-1 Optitrack Interface Networking Concept

In the simulator control software itself, the pre-existing feedback portion of the program is used to incorporate the closed loop control mode. Optitrack position and rotation data is compared to the internal position information in the VI. If the difference is above a certain value the difference is applied as a correction once the setpoint calculation is complete. This part of the implementation is described in more detail in section 4.2.3.

3.2 Implementation Details

Due to the proprietary nature of the NatNet SDK, the easiest way of implementing an interface in Labview is to write a wrapper function for NatNet in C++, which Labview then calls. Labview provides two options for running code written in external text-based languages: the *call library function* node, and the *code interface* node [14]. The *call library function* node interfaces with precompiled DLLs, while the *code interface* node directly calls C code from Labview. According to the Labview manual, using the *code interface* node requires more programming experience and restricts compiler choice, but has the advantage of running faster and being able to directly pass complex data structures to and from Labview. These advantages do not outweigh the higher effort required for writing the code for this project, so *call library function* nodes were used instead.

One problem with *call library function* nodes is that they only allow primitive data types like *float*, *int* or *char* to be passed to and from Labview. For the Optitrack interface, it would be convenient to pass an array containing all position and rotation data for the two tracked elements for a total of 14 elements (three for position, four for rotation per tracked element). In addition, to keep the interface scalable and to enable future growth, the number of tracked bodies should be able to change, so it would be helpful to pass a two-dimensional array with each line representing position and rotation for one body. This can be achieved by using pointers. Labview provides a built-in VI called *GetValueByPointer*, which returns the value stored at the memory location indicated by the pointer. A two-dimensional array can be implemented as a double pointer, meaning a pointer to the first element of a one-dimensional array of pointers which in turn each point to the first element of a one-dimensional array of numbers. The C++ part of the interface was created and compiled in Visual Studio 2013, and the Labview part was created using Labview 2016 32 bit.

3.2.1 C++ Code Description

The C++ DLL is based on one of the example programs provided with the NatNet SDK. The source code consists mainly of one header file and one code file. Several additional files automatically created by Visual Studio are required to compile the result into a single DLL. For later use with RACOON, the dependencies also include the *tinyxml2* library for interfacing with XML files. This allows reading certain parameters from the RACOON configuration file. For the ORION implementation, this feature is not used. *Tinyxml2* is compiled into the *OptitrackConnect.dll*. All components of the *OptitrackConnect.dll* source code are listed in

Table 3-1, and the full source code is part of the files appended to this thesis. The main external dependency is the *NatNetLib.dll*. Otherwise, all dependencies should be included in a normal install of Windows and Visual Studio. *OptitrackConnect.dll* can be compiled both as a 32 bit and a 64 bit version, depending on the Labview version used.

The correct version of *NatNetLib.dll* must be placed in the same folder as *OptitrackConnect.dll* for it to run successfully.

Table 3-1 *OptitrackConnect.dll* Source Code Files

Name	**Purpose**	**Source**
dllmain.cpp	Entry point to DLL, no further functionality	Auto-created by Visual Studio
NatNetCAPI.h	Setup code for NatNet	NatNet SDK
NatNetClient.h	Function prototypes for NatNetLib.dll	NatNet SDK
NatNetTypes.h	Declares custom data structures used in NatNetLib.dll	NatNet SDK
OptitrackConnect.cpp	Main source code file	New code
OptitrackConnect.h	Function prototypes, includes and global constants for OptitrackConnect.dll	New code
stdafx.cpp	unused	Auto-created by Visual Studio
stdafx.h	Includes for any system libraries used	Auto-created by Visual Studio
targetver.h	Can be used to define includes for different Windows versions	Auto-created by Visual Studio
tinyxml2.cpp	Code for the Tinyxml2 library	Tinyxml2
tinyxml2.h	Function prototypes and includes for Tinyxml2	Tinyxml2

The main static variables declared in *OptitrackConnect.h* define the default connection properties. They can only be changed by recompiling the DLL and include:

- Connection Type: Set to Multicast for ORION.
- Default IP: Fallback IP address of the NatNet server (i.e. the computer running Motive). For ORION this is set to 129.187.61.235. It is used if no other IP is provided when the main DLL function is called.
- Configuration File Path: Indicates the path to an XML configuration file using the RACOON format. Not used for ORION.

The header file also provides prototypes for the functions used in the DLL. They are split into export functions, which are accessible to any program using the DLL (such as the Labview *call library function* node), and helper functions, which are only used inside the DLL. The exported functions are:

- *ListenForOptitrackData*: this function provides the main functionality of the DLL. It requires the IP address of the NatNet server and the number of bodies to track as an input and continually writes data it receives from Motive into memory. It does not return values. This function is designed to run in a background thread so it does not block the other functions from executing.
- *ReturnOptitrackData*: when called, this function returns a double pointer to the array containing the data received from Motive.
- *KillListeningProcess*: when called, this function terminates the *ListenFor OptitrackData* function.
- *OptitrackStatus*: when called, this function returns an integer value indicating the current status of the *ListenForOptitrackData* function.

The communication between these functions is enabled by global variables. While non-static globals are generally viewed as bad practice in C++ programming, in this application they keep the code simple, and due to the small size of the overall program (Labview cannot access these global variables) the danger of inadvertently changing their values is low. The main global variables used are:

- *TrackingData*: array expressed as a double pointer, indicates memory location of data received from Motive.
- *ContinueListening*: Boolean that is queried regularly by the *ListenForOptitrackData* function, if it is set to zero the function terminates.
- *OptitrackStatusID*: integer indicating the current status of *ListenForOptitrackData*, has the following values:
  - 0: Connection to NatNet server not yet established
  - 1: Successfully connected to NatNet server and receiving data
  - 2: Unable to connect to NatNet server
  - -1: Function successfully terminated
- *BodyNumber*: Number of rigid bodies the *ListenForOptitrackData* function returns data on. If this exceeds the number of bodies tracked by Motive, the positions and rotations of all excess bodies are shown as 0.

An illustration of the interactions between externally accessible functions and main global variables in the DLL is provided in Figure 3-2.

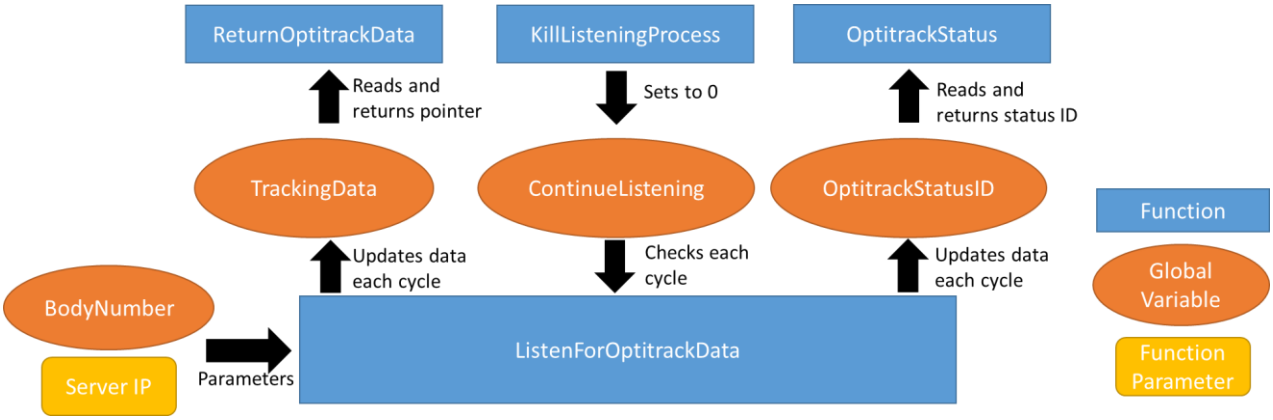


Figure 3-2 Overview of *OptitrackConnect.dll* Functions and Global Variables

Most of the functionality of the DLL is implemented in *ListenForOptitrackData*. It is set up as a mostly linear structure leading into a loop which can only be broken by an external command sent via the *ContinueListening* variable. When it is first called, *ListenForOptitrackData* initializes the global variables to their default values and fills the initially NULL double pointer *TrackingData* with an empty array. It then sets up a callback function which handles the reading of data sent from the NatNet server. The callback function executes whenever a new data frame is received from Motive via UDP, which depends on the framerate set in Motive and usually happens 100 times per second. *ListenForOptitrackData* then establishes a connection to the NatNet server and enters a while loop which checks the *ContinueListening* variable at a frequency of 4 Hz. The callback function remains active during this time, executing on each new data frame and writing the received data into the *TrackingData* array. If *ContinueListening* is found to not equal 1 (which only happens if the *KillListeningProcess* function is called), the while loop is terminated. The function then disconnects from the server, frees all variables and sets the status ID to -1 to indicate that it has terminated.

The callback function is called *DataHandler*. Each time it is called, it iterates through as many rigid bodies as indicated by the global variable *BodyNumber*. Position data, which is already provided in meters, is stored directly in the *TrackingData* variable. Rotation data is first transformed from quaternions to Euler angles using Equations 3-1 to 3-3 [15]:

$$\phi = \arctan\left(\frac{2(q_0 q_1 + q_2 q_3)}{1-2(q_1^2+q_2^2)}\right) \qquad (3\text{-}1)$$

$$\theta = \arcsin(2(q_0 q_2 - q_3 q_1)) \qquad (3\text{-}2)$$

$$\psi = arctan\left(\frac{2(q_0 q_3 + q_1 q_2)}{1-2(q_2^2+q_3^2)}\right) \qquad (3\text{-}3)$$

ϕ, θ and ψ are the roll, pitch and yaw angles, repectively. q_0, q_1, q_2 and q_3 are the quaternions. The implementation in C++ uses the atan2 function instead of arctan to obtain angles outside the (-90°, +90°) range. The Euler angles, in radians, are then also stored in *TrackingData*. The data array is overwritten each time the callback function is executed.

Labview does not support debugging of precompiled C++ code executed by a *call library function* node. The node is treated as a black box. An unhandled error within the DLL will usually cause Labview to crash without detailed indication of the error cause. During development, it is thus advisable to create a C++ test environment calling the functions within the DLL which can be run with the debugging tools of Visual Studio enabled. This has the additional advantage of showing outputs of the *printf* function in a console window, enabling the use of debugging messages in the DLL. Printf outputs are suppressed if functions from the DLL are called in Labview. A debugging environment for *OptitrackConnect.dll* exists and is part of the appended files to this thesis.

3.2.2 Labview Code Description

While the C++ side of the Labview-Optitrack interface was kept as general and simple as possible to allow usage in other projects, the Labview side has some application-specific components. The concept for the Labview VI is illustrated in Figure 3-3. Two threads are used, one of them running the *ListenForOptitrackData* function from the C++ DLL in a *call library function* node. As discussed in section 3.2.1, it updates the Optitrack tracking data stored in memory at the framerate set in Motive (usually 100 Hz). The second thread first gets and de-references the tracking data double pointer from the C++ DLL, thus obtaining the memory location of the tracking data. It then enters a timed loop executing at 100 Hz, which reads the stored tracking data, filters it, maps it from the Optitrack coordinate system to the hardware simulator coordinate system and then transmits it to the CRIO via UDP. The C++ data listener and the Labview main loop run asynchronously. The data read and filter functions can be reused for other projects wthout changes, but the mapping function needs to be updated whenever either the Optitrack or the project coordinate system changes and the data transmission function may need changes if the target software is running on a different system.

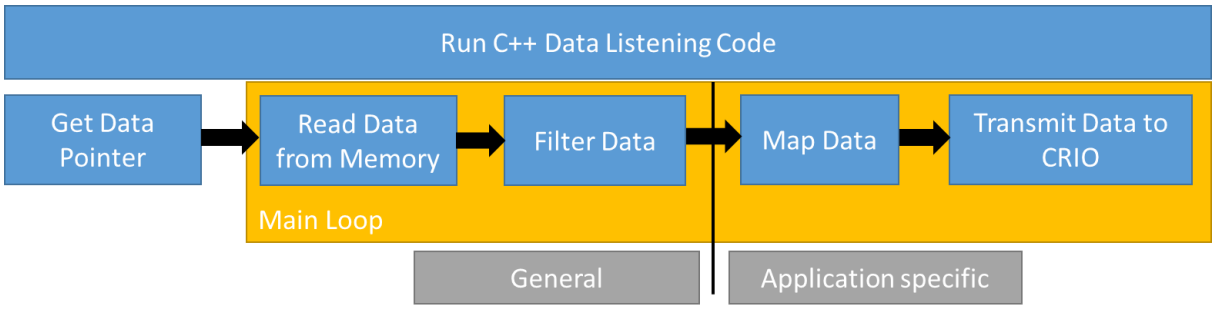


Figure 3-3 Optitrack Interface Labview Code Concept

The user interface for the Labview VI is shown in Figure 3-4. The elements indicated in the figure are:

1. This two-dimensional array shows the tracking data as received from Optitrack. Each tracked body has its own line. The coordinate system is the one used by Optitrack.
2. This section has the controls for the filter, with a toggle to turn the filter on or off. In addition, the filter cutoff frequencies can be set here.
3. This array contains the mapped and filtered data, which is transmitted to the CRIO in this format. Positions are indicated in meters, rotations in degrees, and the coordinate system is the one used by the hardware simulator.
4. The graph shows the values for one of the axes, taken from the axis position array. It has a control for switching through the six axes.
5. These clusters show the internal status of the Optitrack connection using the OptitrackStatusID discussed in the previous section, as well as any errors within the Labview UDP functions.
6. The Stop button is used to stop the VI execution. The C++ background thread must be internally shut down to stop Labview from crashing when exiting the program, so the Labview abort VI button is disabled and this button must be used instead.

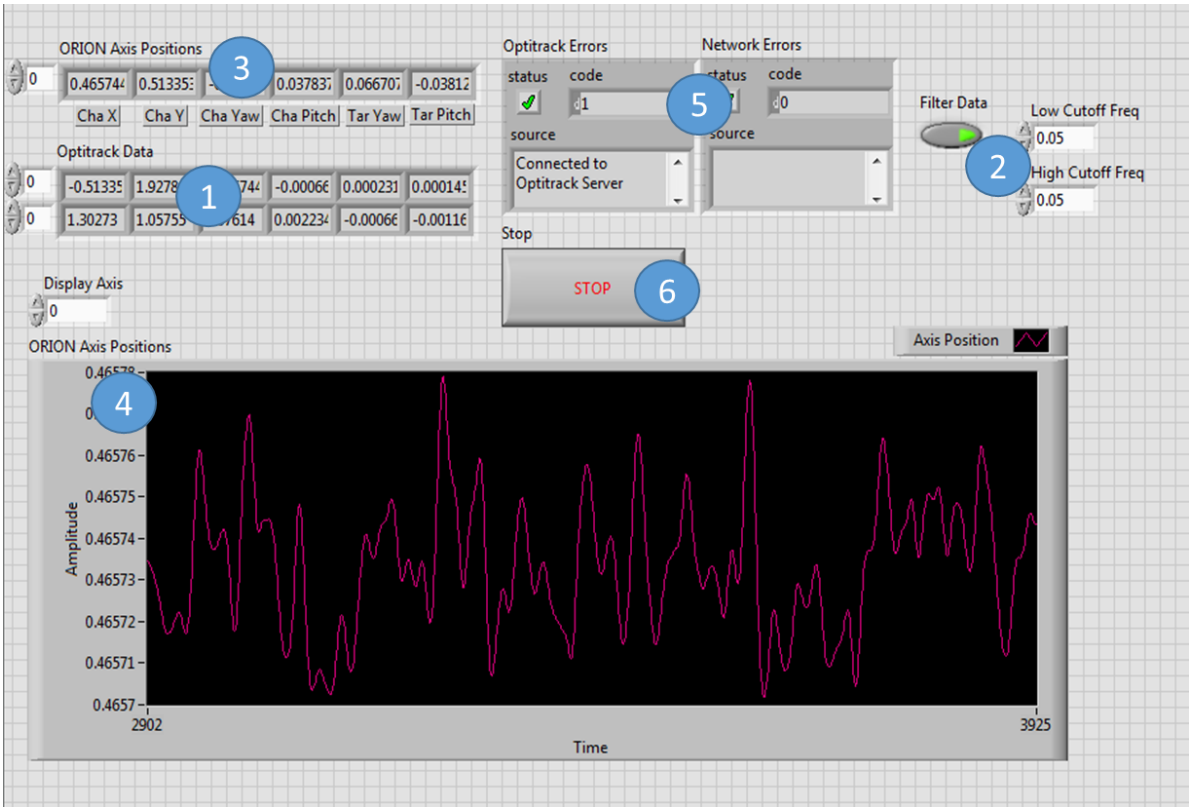


Figure 3-4 Optitrack Interface GUI

Figure 3-5 shows the main block diagram of the Optitrack interface VI. Most of the logic is contained in sub-VIs, the main diagram only handles the data displays in the GUI, the data transmission, and the main program structure. The numbered markers in the figure indicate:

1. This is the *call library node* function running the C++ data listener. It is configured to run in a background thread and takes the Motive server IP address and the tracked body count of two (chaser and target) as preset inputs.
2. This sub-VI, called *CheckOptitrackStatus*, periodically runs the *OptitrackStatus* function from the DLL until it either times out after 5 s, receives an error from the DLL or reads that the *ListenForOptitrackData* function has successfully connected to Motive and is reading data. If this function encounters a connection error or timeout, the main loop of the interace VI is skipped and the GUI indicates the encountered problem.
3. This sub-VI, called *GetFilterReferenceArray*, sets up the filtering function used in the main loop.
4. This sub-VI, called *GetDataPointers*, executes the *ReturnOptitrackData* function from the VI and performs the first de-referencing step. The pointers for the individual rows of the two-dimensional data array stay constant throughout the runtime of the program, so they are de-referenced here before entering the main loop to save resources.
5. The first sub-VI to execute after entering the main loop is *GetOptitrackData*. With each iteration, it de-references the pointer array that *GetDataPointers* has read from memory and combines the data for all tracked bodies into a two-dimenional data array in Labview. This array is displayed to the user via the GUI.

6. This sub-VI filters the received data. It can be toggled on or off, and the cutoff frequencies can be set in the GUI, although changing the cutoff frequencies requires restarting the interface VI. More information regarding filtering can be found in section 3.2.3.
7. After filtering, the data is mapped from the Optitrack coordinate system to the one used by the ORION gantry in this sub-VI.
8. When the interface VI is shut down using the Stop button, the sub-VI *OptitrackShutdown* is executed. It runs the *KillListeningProcess* function from the DLL and then periodically checks OptitrackStatusID until it indicates that *ListenForOptitrackData* has terminated. When this sub-VI returns, execution of the interface VI stops.
9. Located below the sub-VIs, this section handles the data transmission to the CRIO. The UDP connection is set up before entering the main loop, then the *UDP Write* function is executed on each loop iteration and finally the connection is terminated after exiting the main loop.

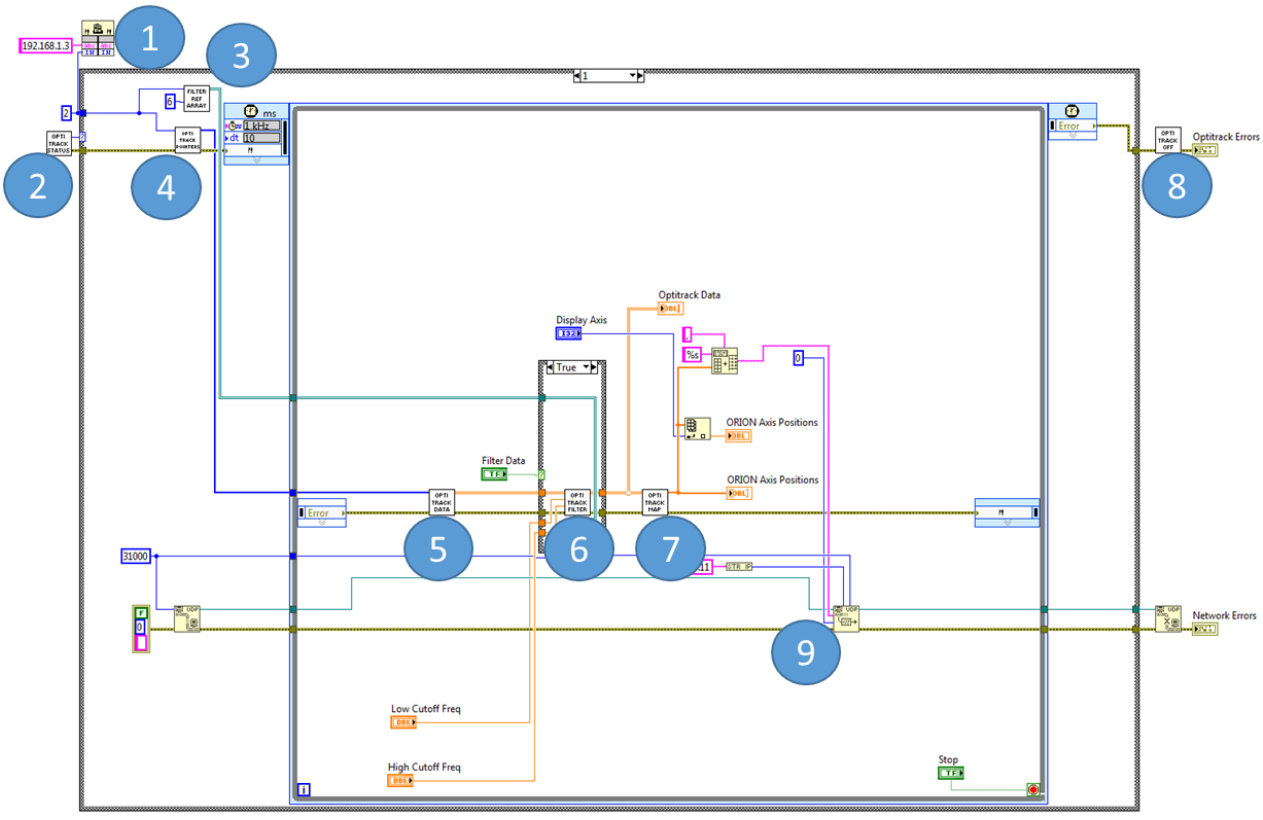


Figure 3-5 Optitrack Interface Main Block Diagram

The sub-VIs used in de-referencing the pointers are shown in Figure 3-6, with the *GetDataPointers* VI on top and the *GetOptitrackData* VI below. The first of these VIs may require changes if the Optitrack interface is adapted to a different system since it requires incrementing through memory locations in its for-loops. The *GetDataPointers* VI receives the pointer to the first row in the data array from a call library function node executing *GetOptitrackData*. It then de-references it to obtain the pointer to the array containing the status vector of the first tracked body. As final step, the VI then adds

four to the pointer to get the pointer for the second body by shifting the pointer address by 4 bytes, which is the size of a pointer in a 32-bit system. To port the interface VI to a 64-bit version of Labview, the shift must be by 8 bytes to account for the greater length of a pointer. The *GetOptitrackData* sub-VI returns a two-dimensional array, with the two nested for-loops each responsible for one dimension. The outer loop iterates through the array of pointers returned by the *GetDataPointers* VI, with each representing one of the tracked bodies. The inner loop then de-references the six elements of each status vector, incrementing the pointers by eight because the state vector elements are stored as double precision floating point numbers (doubles) with a length of 8 bytes each.

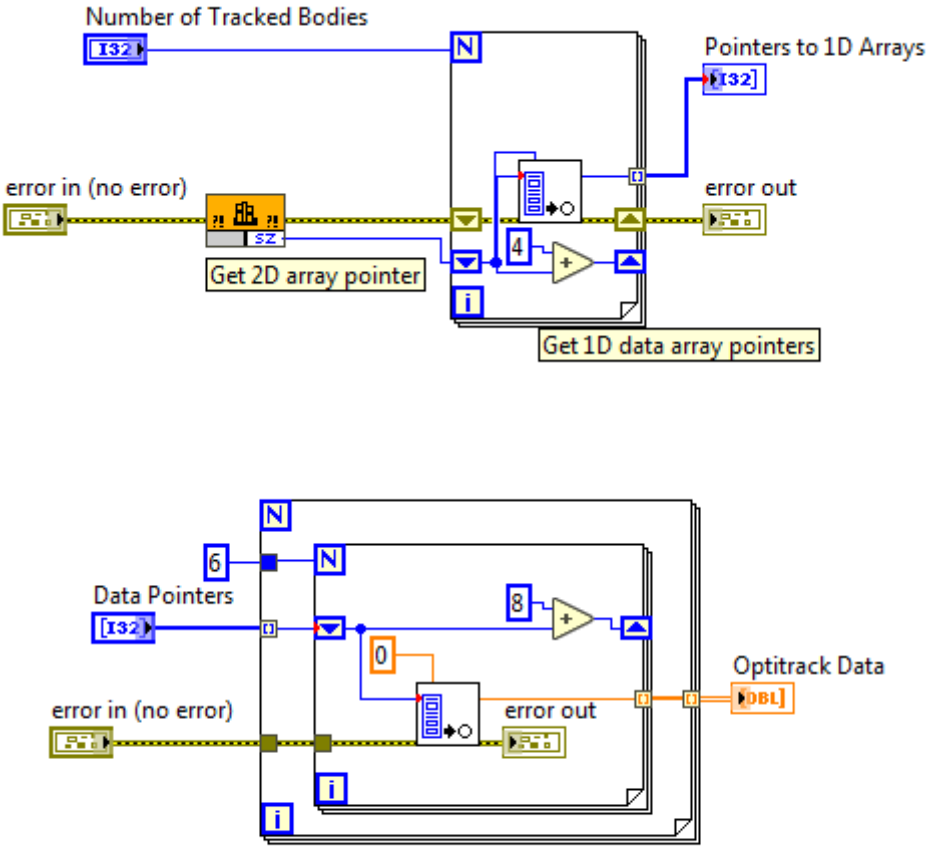


Figure 3-6 Pointer Dereferencing Sub-VI Block Diagrams

The filtering function utilizes Labview point-to-point filters. Their settings and effects are described in more detail in section 3.2.3, but the implementation is shown in Figure 3-7. The filter needs to process all incoming data channels (number of tracked bodies multiplied with six) seperately, but if the data is simply fed into a point-to-point function contained in a for-loop, the function will receive data from all channels. To prevent this, the setup VI *GetFilterReferenceArray* (top in Figure 3-7) is executed before the main loop and uses the open VI reference function to create an array of Labview objects, each referencing the point-to-point VI, one for each data channel. The *FilterOptitrackData* function (bottom in Figure 3-7) then uses these references in

a call by reference node to create one instance of the point-to-point function per data channel. These instances then get called in turn as the data array is parsed, and each of them only sees one data stream from one channel. This allows the point-to-point function to filter the data correctly.

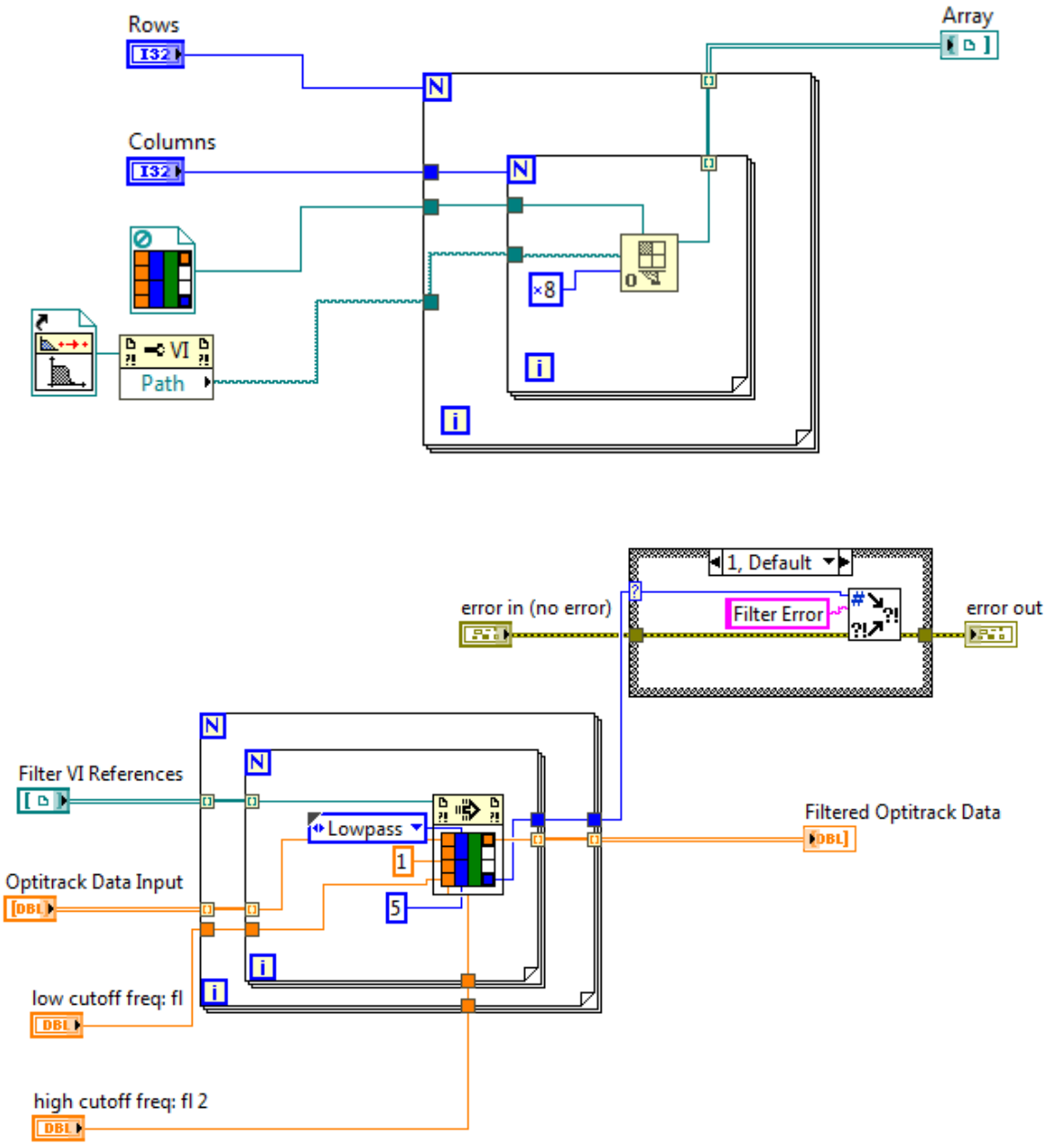


Figure 3-7 Filter Setup and Execution Sub-VI Block Diagrams

The data mapping sub-VI is shown in Figure 3-8. It iterates through all six axes of the hardware simulator and assigns a value from the two-dimensional Optitrack data array to it, converting units and changing signs in the process where necessary. As every axis is treated differently, this is done using a switch-case structure. The output is a one-dimensional array containing the position data for all axes of the simulator. This

function is specific to the Optitrack and hardware setup used in the ORION lab and must be updated if the lab configuration changes or the interface VI is used for a different project. The mapping configuration used for this thesis is shown in Table 3-2. Figure 3-9 shows the origins and axes of the Optitrack and the gantry coordinate systems within the laboratory. The origin of the gantry coordinate system is defined by the settings for its reference search mode, while the Optitrack coordinate system's origin is determined during calibration.

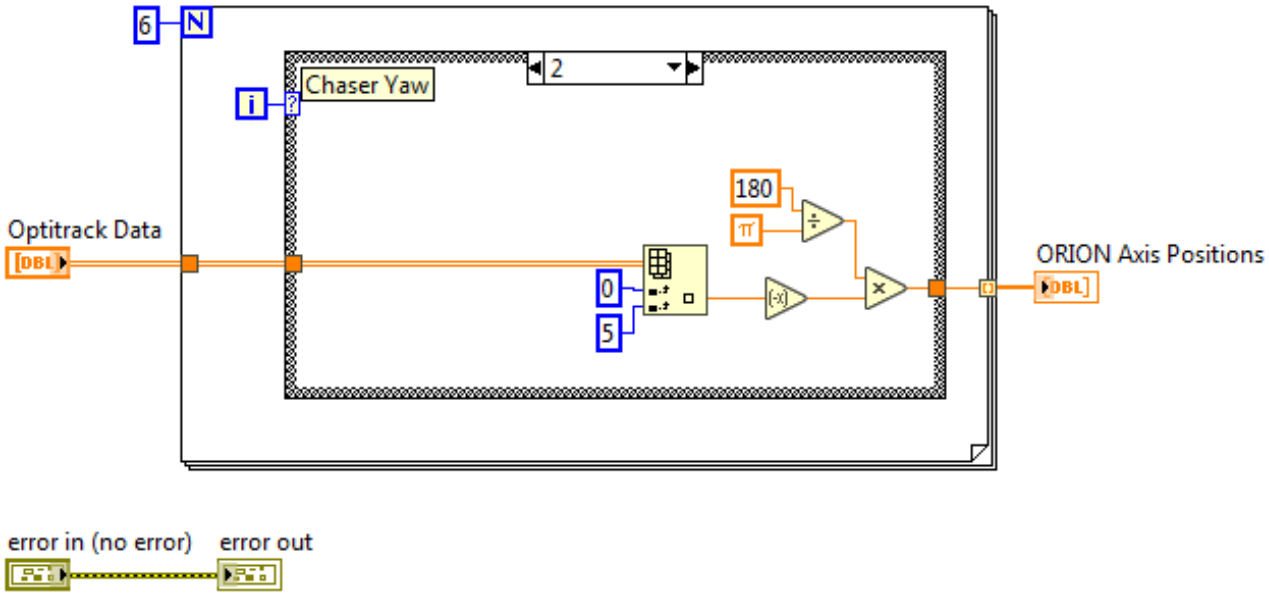


Figure 3-8 Data Mapping Sub-VI Block Diagram

Table 3-2 Data Mapping Configuration

ORION axis	Chaser X	Chaser Y	Chaser Yaw	Chaser Pitch	Target Yaw	Target Pitch
Optitrack axis	Body 1 Z	Body 1 X	Body 1 Yaw	Body 1 Roll	Body 2 Yaw	Body 2 Roll
Unit Conv.	-	-	rad to deg	rad to deg	rad to deg	rad to deg
Sign Change	no	yes	yes	yes	yes	yes

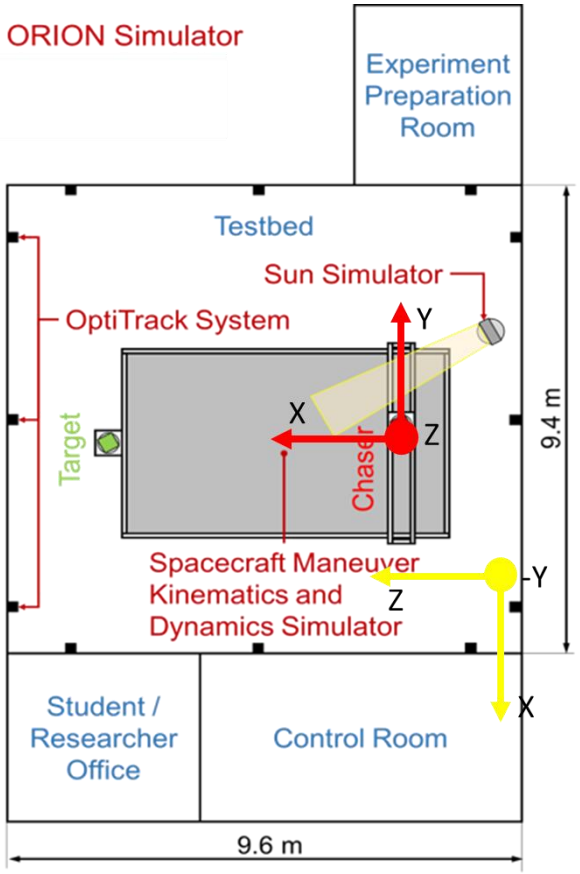


Figure 3-9 Optitrack Coordinate System (Yellow) and Gantry Coordinate System (Red), Vertical Vectors Point out of Plane

The full Labview project structure including all Sub-VIs is as follows:

Project: *OptitrackInterface-ORION.lvproj*

OptitrackInterfaceMainVI.vi (Main GUI)

- *CheckOptitrackStatus.vi*
- *FilterOptitrackData.vi*
- *GetDataPointers.vi*
- *GetFilterReferenceArray.vi*
- *GetOptitrackData.vi*
- *MapOptitrackToGantryCoordinates.vi*
- *OptitrackShutdown.vi*

3.2.3 Data Filtering

While data filtering is not a core focus of this thesis, a filtering function was implemented and tested in the Optitrack interface. The filter parameters were found empirically by testing and comparing various configurations. A lowpass 5th order Butterworth filter is used, with a cutoff frequency of 0.05 Hz. Movements of the gantry are usually slow and not periodical, so a low pass filter with a very low cutoff frequency is a good fit for the application. This configuration was found to provide satisfactory performance after testing various options including bandpass filtering, varying filter order and cutoff frequencies and trying other filter types such as Chebychev. The high cutoff frequency shown in the GUI is unused, but still present in case future users want to apply bandpass instead of lowpass filtering. Only the cutoff frequency can be changed in the GUI, while order and filter band can be updated in the sub-VI *FilterOptitrackData*. To change the filter type (for example from Butterworth to Chebyshev), the sub-VI *GetFilterReferenceArray* must be updated. Both the static VI reference function and the VI type specifier (showing the VI connector panel) must be changed. If the connection panel is altered by this update, the connections in *FilterOptitrackData* may need to be repaired. Only point-to-point filters can be used without significantly altering the entire program structure of the interface.

To test the performance of the filters and the interface VI over an extended period, a data logger was added to a debug version of the software. The debug version was used to track a stationary body for a period of several hours. The standard deviation of the tracking data was calculated at each time step (100 times per second) for the past 100 timesteps. The standard position deviations of the X and Roll axis were then plotted using Matlab. Figure 3-11 and Figure 3-11 show the results. Conservatively, a 3-σ error of about 1 mm can be assumed for position and about 2 mrad (0.11°) for rotation on the unfiltered signal, provided the Optitrack system has good visibility on all tracked markers and no or only slow movement takes place. The filtering VI cuts this down to 0.3 mm for position and 0.75 mrad (0.04°) for rotation, an improvement of almost 70%. The slow increase of the error over time can likely be attributed to heating of the infrared cameras. It is unknown why the deviation sometimes drops to a lower value for a short time, as visible most clearly in Figure 3-11 between the 3 and 3.5 h marks. The peaks before and after this drop can likely be attributed to mathematical error due to the large value change. The signal delay introduced by the filter is around 0.15 s, which is acceptable if no fast movements (much faster han the gantry) are being tracked.

The tracking performance is comparable to optical tracking systems used in other laboratories as described in section 1.1. Performance is similar to the Optitrack installation in Bologna [6] and even compares well to the more expensive VICON and PhaseSpace systems used by the Naval Postgraduate Research Center [4] and the Rensslaer Polytechnic Institute [5]. Comparable performance data for RACOON is not available. Data filtering has shown very promising results even without much research put into it and should be examined further in the future.

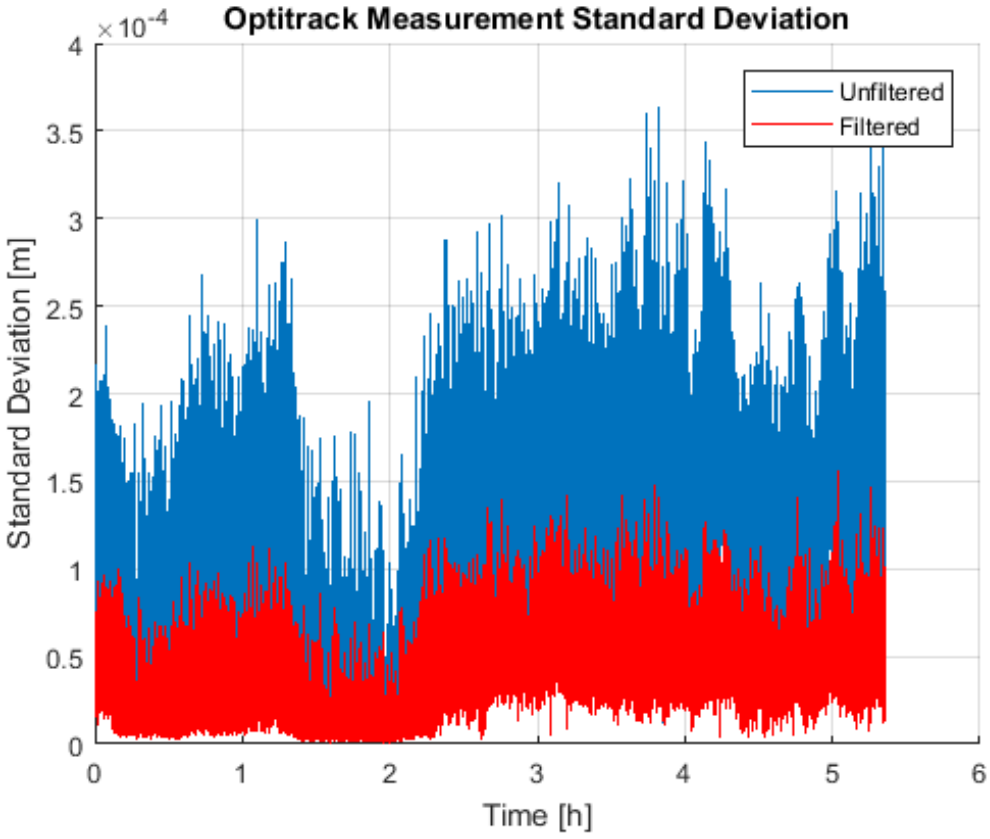


Figure 3-10 Measured Optitrack Standard Deviation in X

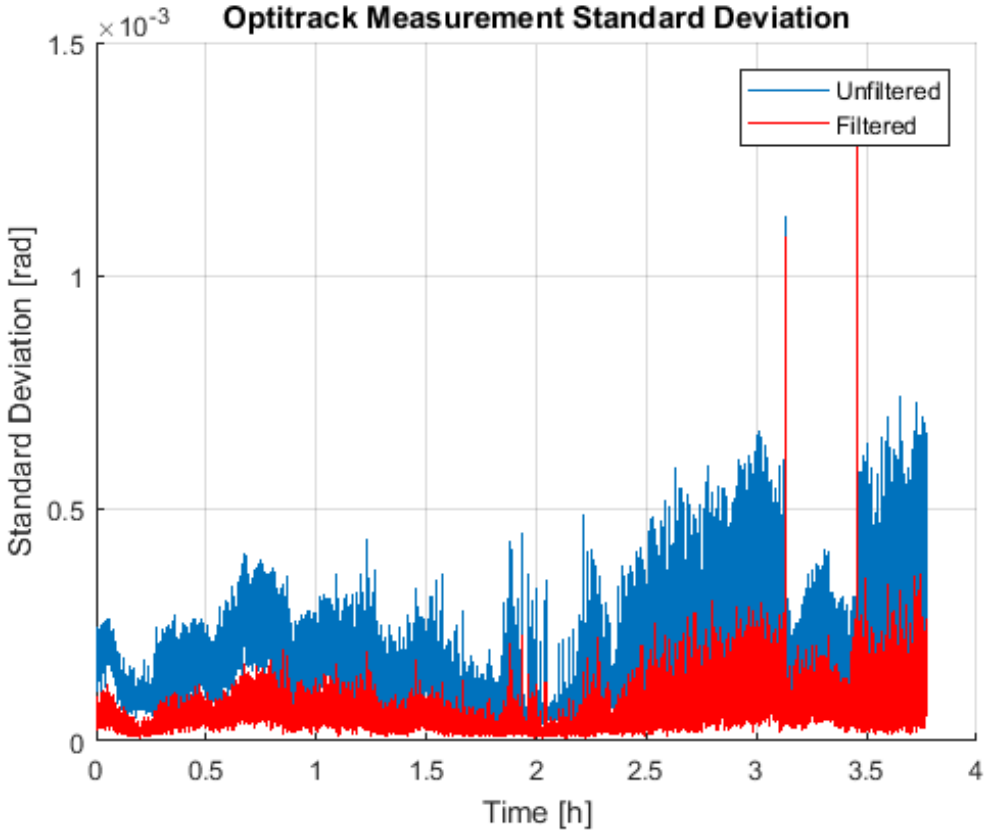


Figure 3-11 Measured Optitrack Standard Deviation in Roll

4 ORION Software Updates

After completing the Optitrack-Labview interface and successfully transmitting data to the CRIO, the next step and core of this thesis is updating the ORION hardware operating software. This section documents the new software, from requirements (section 4.1.) to implementation (section 4.2) and testing (section 4.4). Some updates to the simulator hardware were made in the process of the software development as explained in section 4.3. Finally, an example use case of the new software capabilities is described in section 4.5, and section 4.6 examines how well the requirements were fulfilled and how its capabilities compare to other similar systems. The operation procedures for the new software are provided in Appendix B.1.

4.1 Requirement Definition

Defining requirements is a critical step in any complex engineering project. It is important to compile all stakeholder expectations and define a success state for the development. Most requirements for new features are defined by current and potential future users of the ORION gantry. Table 4-1 shows the requirements defined for the new gantry control software at the beginning of the upgrade project, including prioritization and verification methods.

Table 4-1 Requirements for the Updated ORION Software

ID	Requirement	Priority	Verification
1	The software shall be programmed using NI Labview 2016.	Must	Inspection
2	The real-time control software shall be implemented on the existing NI cRIO 9024 real time controller.	Must	Inspection
3	The control software shall be able to command all axes of the ORION gantry system: Chaser X and Y position, Chaser pitch and yaw, Target pitch and yaw	Must	Demonstration
4	The software shall provide the following information to the operator at all times in a GUI on the host computer (ORION2): 1. Current and commanded position, velocity and acceleration of all axes 2. Critical system status information (Connection status, errors etc.) 3. Status of end and reference switches	Must	Demonstration
5	The operator shall be able to make the following adjustments in the GUI on the host computer (ORION2) as needed: 1. Target position for all axes 2. Maximum velocity and acceleration for all axes 3. Software defined limit positions for all axes 4. Position tracking mode 5. Positioning mode	Must	Demonstration

6	The GUI provided by the software shall use the following units: meters for position, degrees for angles	Must	Inspection
7	In case of any critical error, the software shall inform the operator via the GUI on the host computer (ORION2) and return the system to a safe state.	Must	Demonstration
8	The software shall support controlling the gantry using the following methods for position determination and enable the operator to switch between them for each axis: 1. Open-loop guidance, calculating the state vector internally 2. Optitrack closed-loop guidance, determining the state vector from position data gathered through the Optitrack system and streamed in from the ORION3 computer	Must	Demonstration
9	The software shall support the following gantry motion commands and enable the operator to switch between them: 1. Move axis to set position 2. Move axis at set velocity	Must	Demonstration
10	The software shall be able to accommodate the following modes of setting the target position for each axis: 1. Direct input in the GUI on the host computer (ORION2) 2. Constantly updating input stream from the simulation computer (ORION1) 3. A configurable sine wave generator in the GUI on the host computer (ORION2) 4. Input (and loop) any pregenerated command sequence on the host computer (ORION2)	Must	Demonstration
11	The software shall account for the following methods of determining the maximum motion range and enable the operator to switch between them: 1. Software defined limit positions 2. Magnetic reference switches on each axis	Must	Demonstration
12	The software shall be able to command any axis of the gantry system to follow a user-specified sine shape curve.	Must	Demonstration
13	The software shall be able to command the gantry to follow an arbitrary trajectory input by the user in a specified format.	Must	Demonstration
14	The software shall provide an option to record a time series of any internal variables for later analysis.	Must	Demonstration

15	The software shall detect the activation of the axis limit switches and stop any movement of the system if they are triggered at any time.	Must	Demonstration
16	The software shall enable repeatable positioning accuracy to within 1 mm in all translational and 1 degree in all rotational axes within an external reference frame when operating in closed-loop mode.	Must	Measurement
17	The software shall send updated position commands to the FPGA at a rate of 100 Hz in all operating modes.	Must	Demonstration
18	All Labview VI front panels visible to the operator in normal operation should fit on a 1920 x 1080 pixel monitor without scrolling.	Should	Inspection
19	All Labview VI block diagrams should fit on a 1920 x 1080 pixel monitor without scrolling.	Should	Inspection

4.2 Implementation

The software implementation process is based on the Labview Style Guide [16], which recommends a top-down approach to program development. First, the user interface is designed, as described in section 4.2.1. Next, the top-level block diagram is created, initially made up of empty sub-VIs. This process is repeated for lower tiers as necessary. The resulting program structure is explained in section 4.2.2. Once this structure is finalized, the actual code implementation can begin. A lot of code is reused from the first version of the ORION software, but it is reorganized to improve readability and simplify debugging. The Optitrack integration allowing for closed loop control of axis positions is described in section 4.2.3. Trajectory follow and sine wave operating modes are implemented to meet the stakeholder requirements, as described in section 4.2.4. Finally, section 4.2.5 explains how data loging functionality is realized in the new software.

4.2.1 User Interface

The first step in updating the software is the new user interface. As Labview lacks a zoom feature, it is highly recommended by the Style Guide to keep the GUI small enough to fit on the screen intended for normal use with this program without scrolling. The user should have all relevant information for the system operation available at once, while unimportant information should be hidden to prevent cluttering of the interface. As explained in section 2.5, the original software displays all information to the user in three clusters. This includes internal variables, warnings and important parameters alike and makes it very difficult for a new user to understand the system state. In addition, only data for one axis is visible at any time. For the new GUI, it is thus necessary to look at the data contained in the original software and sort the information into different categories of importance. In the final implementation, three layers of information exist: basic controls, advanced settings and internal variables.

The basic control panel is displayed when first starting the software (Figure 4-1). It has all information required for normal operation of the gantry. Identical data clusters for all six axes are shown at the same time, and there is an additional section for quickly switching key settings on all axes simultaneously. Controls for the data logging system are located below the axis data block.

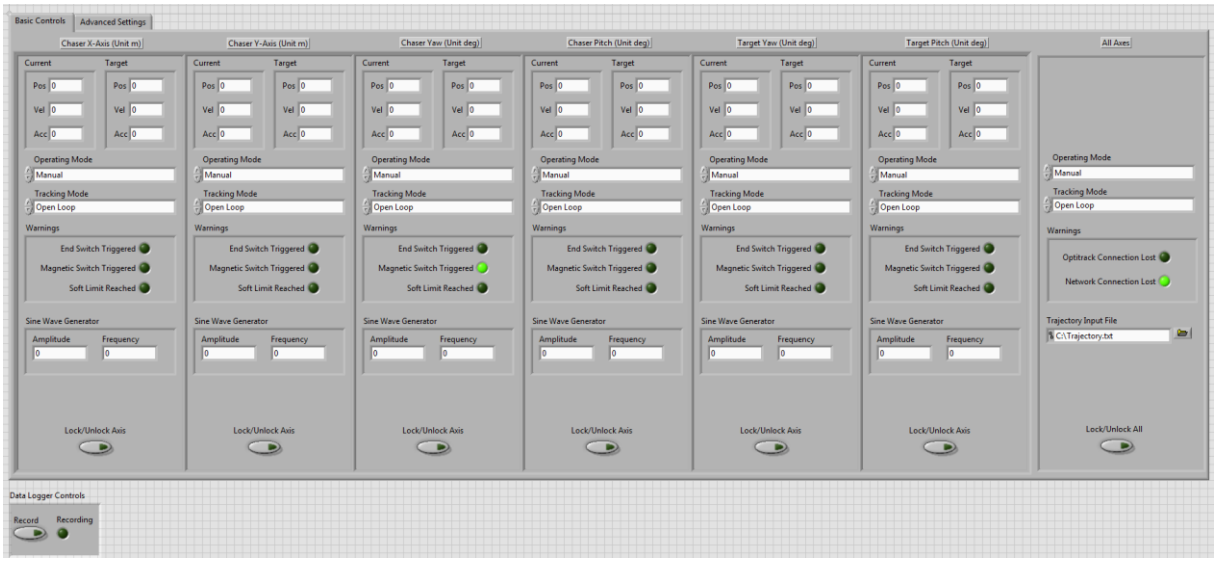


Figure 4-1 ORION Gantry Basic Control Panel

The variables on the basic control panel are:

- Current and target state vector, including position, velocity and acceleration. They are shown in meters for translational and degrees for rotational axes. The current state vector cannot be edited even in manual mode.
- Operating mode selector, allowing the selection via a drop-down menu. This setting can be changed for all axes simultaneously.
- Tracking mode selector, allowing use of the Optitrack system by switching into closed loop mode. This setting can be changed for all axes simultaneously.
- Warning cluster, indicating on each axis whether a limit switch, reference switch or soft limit has been triggered. This helps the user determine the cause of an automatic stop of the system.
- Sine wave generator settings, for input of amplitude and frequency when using the sine wave operating mode. Amplitude is in meters for translational and degrees for rotational axes, frequency is in Hz.
- Lock button, puts the axis into stopping mode when toggled on and returns the axis to the previous operating mode when toggled off. This setting can be changed for all axes at the same time.

Additionally, the all axes section has a warning cluster indicating whether the UDP connections for Optitrack and network data streaming have been lost.

Parameters which do not need frequent updates during operations but may be changed from time to time based on the application are contained in the advanced settings tab. It is highly recommended to make sure the gantry is not moving while the interface is switched to the advanced settings tab since the axis controls are only quickly accessible from the basic control panel. The advanced settings panel is shown in Figure 4-2. The main part of the advanced settings panel is again an array of six

identical data clusters, one for each gantry axis. The advanced settings panel does not contain an all axes panel. The only information not contained in the cluster array is the late counter, which indicates the number of times since program start that the time-controlled main loop of the real-time VI has not finished within the planned time of 0.01 s. If the program responds sluggishly or the gantry itself moves slower than expected, this parameter should be checked to make sure that the system is executing the software fast enough.

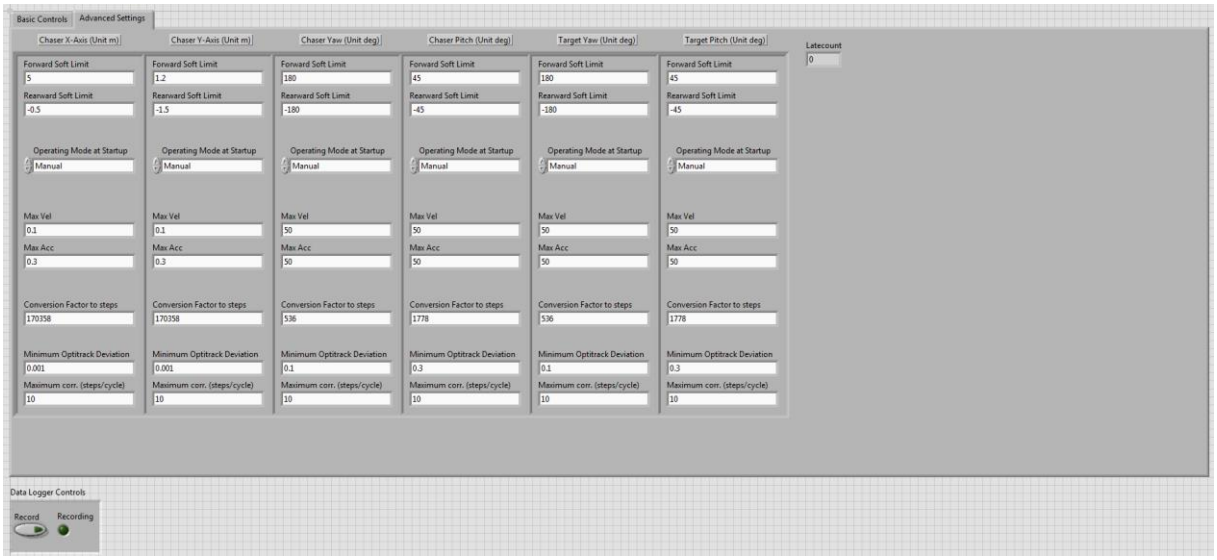


Figure 4-2 ORION Gantry Advanced Settings Panel

The information contained in the main part of the advanced settings panel includes:

- Forward and reversesoft limit. The axis will switch into stopping mode and a warning light in the basic control panel will illuminate if an axis reaches these points. Units are meters for translational and degrees for rotational axes.
- Operating mode at startup: allows selecting which operating mode the selector on the basic control panel is initialized to upon program start.
- Max. velocity and acceleration settings: the axis will not go over these values unless sine wave or trajectory follow mode are used. Units are m/s and m/s² for translational and deg/s and deg/s² for rotational axes.
- Conversion factor to steps indicates the multiplication factor to get motor steps from the position, velocity and acceleration data of the axis.
- Minimum Optitrack deviation indicates how big the difference between the internally calculated position and the value delivered by Optitrack can be before corrective measures are taken. This helps to eliminate noise in the Optitrack position data. The units are meters for translational and degrees for rotational axes. This value only applies to closed loop tracking.
- Maximum correction: indicates how many motor steps can be added to or subtracted from the internally calculated position in each main loop cycle to minimize the deviation from the Optitrack measurement. This only applies to closed loop tracking. This setting is further explained in section 4.2.3.

The final category of information is internal variables. They do not need to be visible to the user but are vital for the program to function. To allow code reuse from the original version, the set of real-time, axis and network data clusters was kept, with some additional parameters added by the new modes. They are still accessible for

debugging by opening sub-VI front panels during the program runtime. Some variables are initialized to specific values, notably for the reference search mode. While these should never be changed by the user, it may be necessary to access them for maintenance or future development. They can be found in the ORION Initialize sub-VI and edited on its front panel.

4.2.2 Updated Block Diagram Structure

After finishing the GUI, the next step in developing the new software is creating the top-level block diagram for the main VI. As the goal is to keep internal variables hidden from the user and to reuse as much as possible of the preexisting code, an information exchange is required between the axis data shown in the GUI and the data being used in the program main loop. The general concept is shown in Figure 4-3.

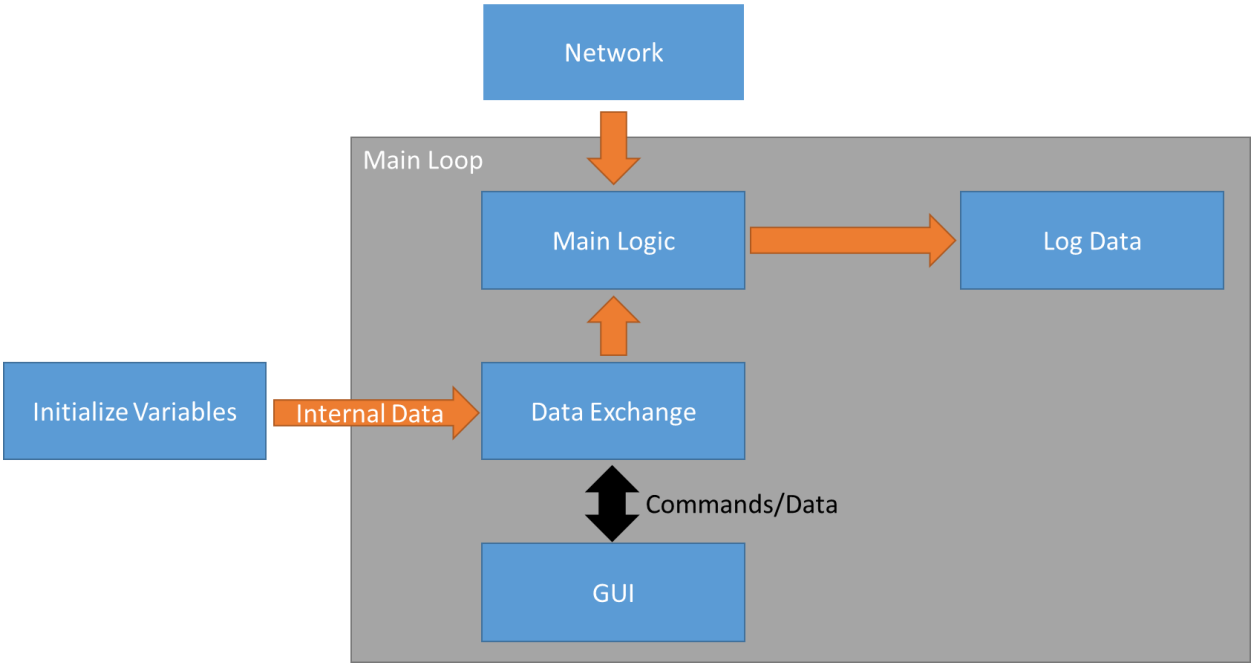


Figure 4-3 ORION Software Concept Flow Chart

The GUI is initialized on the first main loop iteration, setting target and current values to zero. Internal data in the diagram refers to the network, real-time and axis data clusters which were also used in the original code. They are initialized in the sub-VI *ORION Initialize* and then transferred to the *AxisDataHandler* VI, which reads out data to be displayed in the GUI and updates all variables in the internal axis data affected by the basic control or advanced settings panel in the GUI. In addition, this sub-VI implements the functionality of the lock axis switch and overwrites the individual axis settings if a parameter in the all axes block of the GUI is changed. The unit conversion between meters and degrees for the GUI and motor steps for the internal data takes place in this VI. After this exchange of data between the user interface layer and the internal data layer has taken place, the main logic of the program can be run. This is the part that was adopted from the original version of the software, with the addition of closed loop tracking and new operating modes. After the main logic has finished, the internal data is handed over to the data logger and is then stored in a shift register. On the next main loop iteration, this shift register is read and the next cycle now starts with the results of the previous iteration instead of the data provided by the ORION

initializer. Just like in the original code, the main loop is hard coded to run at 100 Hz. This is monitored using some additional code that increments a counter on the advanced settings panel whenever the main loop does not finish within 0.01 s, informing the user about the loss of real-time performance.

The Labview implementation of this concept is shown in Figure 4-4. The real-time and network data clusters are currently not wired through the data handler VI because none of the data contained in them is displayed or affected by the GUI, except for the network connection state. The two UDP blocks visible at the top of the block diagram set up connections to Optitrack and to the network on program start and then feed the connection references into the main logic. None of the function blocks used in the code require special actions to terminate them, so the program can simply be stopped using the Labview abort execution button.

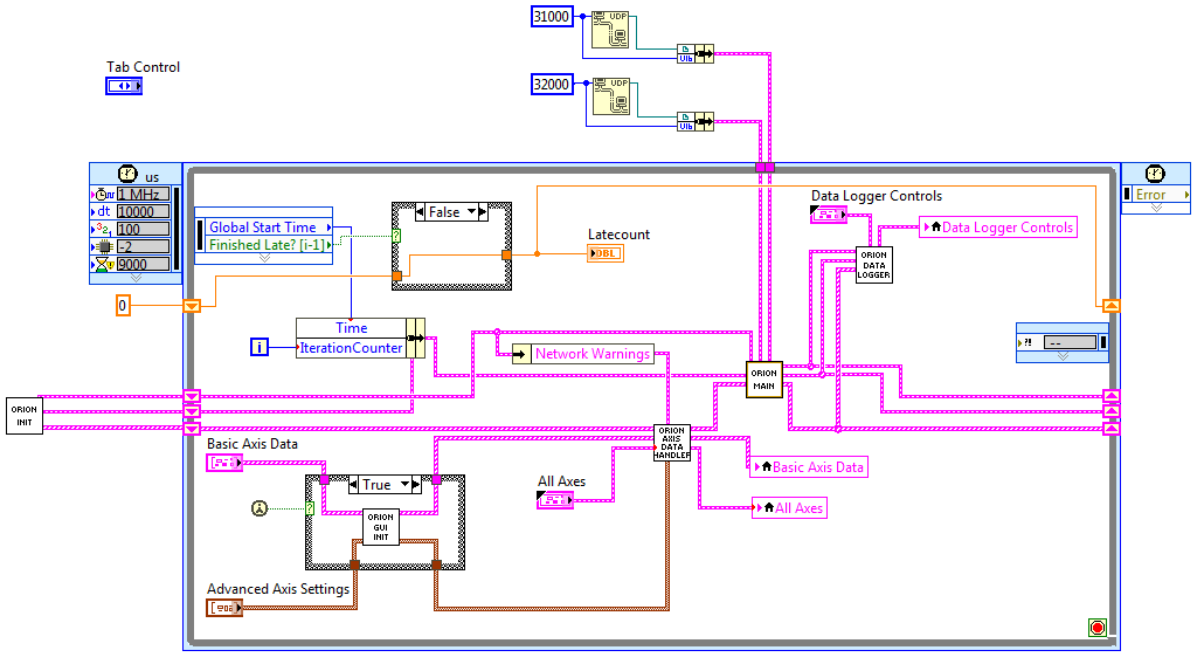


Figure 4-4 ORION Software Labview Block Diagram

The main logic top-level diagram is displayed in Figure 4-5. As mentioned before, it is based on the original software. For better readability and to facilitate debugging, it is separated into sub-VIs, which conform to the frames of the previously used flat sequence structure. The sub-VIs are executed in sequence on each loop. The functionality is described in more detail in section 2.5. As a quick overview the names and main purposes of each sub-VI are listed here:

1. *Convert to Steps*: Converts the state vector input data from meters and degrees into motor steps.
2. *FPGA Interface*: Exchanges data between the FPGA and the real-time software, mostly spline data for movement and end switch status.
3. *Network Interface*: Handles data transmission to and from the ORION1 computer used for teleoperation experiments.

4. *Feedback Handler*: Implements handling of FPGA feedback, which covers motor steps missed due to rounding errors, and contains the logic for the Optitrack interface.
5. *End Switch Handler*: Reads limit switch status, puts any axis reaching a limit switch into stopping mode and controls the corresponding warning indicators.
6. *Mode Handler*: Implements the logic for the various operation modes. The new software supports the following modes:
    a. Manual, for moving the gantry directly using inputs in the GUI.
    b. Network, for using data sent from ORION1 (will now go into stopping mode if connection is lost).
    c. Sine Sweep, for following a sine wave defined by the amplitude and frequency fields in the GUI.
    d. Trajectory Follow, for following a trajectory defined via an input file specified in the GUI.
    e. Stopping, for bringing the respective axis to a halt in a controlled manner and then locking it.
    f. Docked, which is currently not implemented.
7. *Setpoint Handler*, for calculating the spline data which is then passed on to the FPGA to create the input for the motor controllers.

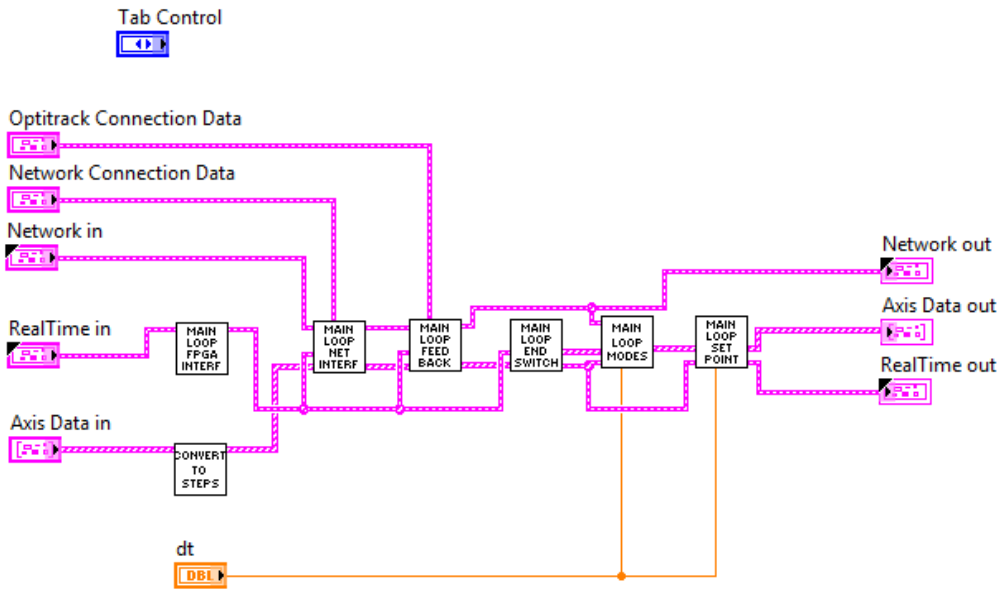


Figure 4-5 ORION Software Main Logic Sub-VI Block Diagram

The full control software project including all sub-VIs is structured as follows:

Project*: ORION Gantry Software 2.0.lvproj*

ORIONGUI.vi (Main GUI)

- *ORION Initialize.vi*
- *GUI Initialize.vi*
- *ORION Axis Data Handler.vi*
    - *ORION All Axes Handler.vi*
- *ORION Main Loop.vi*
    - *Convert to steps.vi*
    - *Main Loop FPGA Interface.vi*
    - *Main Loop Network Interface.vi*
    - *Main Loop Feedback Handler.vi*
    - *Main Loop End Switch Handler.vi*
    - *Main Loop Mode Handler.vi*
    - *Main Loop Setpoint Handler.vi*
- *ORION Data Logger.vi*

4.2.3 Optitrack Interface

Section 3 explains how the tracking data from Optitrack is processed and sent to the CRIO. This section describes how the data is then used to correct the internally calculated positions of all axes, which is implemented in the *FeedbackHandler* VI within the main logic. Figure 4-6 illustrates the concept for the sub-VI responsible for calculating the correction required in each step. The data streamed in from Optitrack via UDP is already mapped to the gantry axes. Therefore, it is an array of six floating point values describing X and Y position of the chaser in meters as well as pitch and yaw position of chaser and target in degrees. All calculations in the program are executed for each axis individually, thus enabling to run any subset of axes in closed or open loop mode as desired by the user. Because UDP by itself does not do any checks on the connection quality, the first step in the program is to check whether the arrived data packet is valid. Whenever the built-in Labview function used to read data from the UDP port does not receive a valid packet, it returns an empty string, and when this is detected in the program a counter is incremented. The counter is reset whenever a valid packet is received. If the counter reaches 100, meaning there have been no valid packets received for an entire second, the axis is switched into stopping mode. In addition, empty packets are discarded and the Optitrack position is assumed to be unchanged compared to the previous main loop iteration.

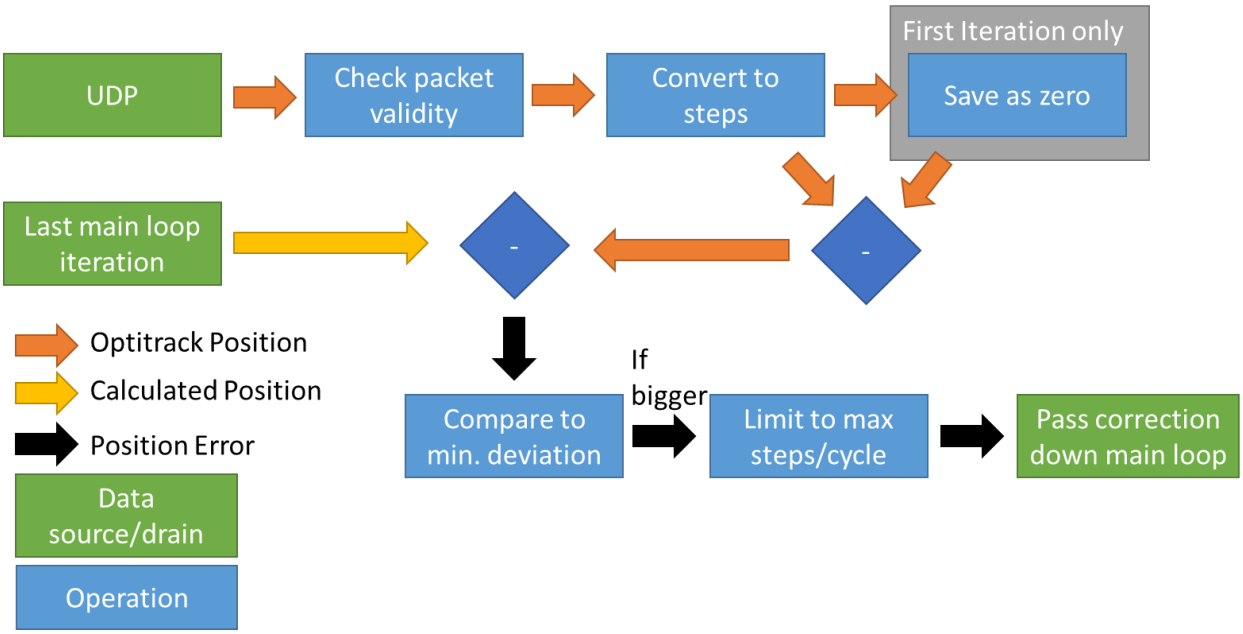


Figure 4-6 Optitrack Correction Calculation Concept

After checking for validity, the position is converted to steps using the user specified conversion factor for the respective axis. On the first iteration after switching from open loop to closed loop tracking (the software is always initialized in open loop mode at startup), the position obtained from the first valid packet detected is saved as a zero position. This way, the program can compensate for changes in Optitrack's internal coordinate origin. It is recommended to always use the reference search mode in the ORION software and drive all axes to zero before switching them into closed loop tracking mode to make sure the Optitrack zero aligns with the gantry zero.

After this initialization step on the first iteration, on all following iterations the zero position is subtracted from the newly acquired Optitrack position to aquire a position within the gantry coordinate system, and then the internally calculated position is subtracted from the result to obtain the error between internal and Optitrack position. If the Optitrack signal was directly used for correcting the gantry position, its noise would cause constant very small oscillations (on the order of a few steps per main loop cycle). This is prevented by comparing the error to a user specified value for minimum deviation required before corrections are applied. If the magnitude of the error is not larger than the minimum deviation, it is set to zero. In a final step, the resulting desired correction is then compared to the maximum allowable correction per main loop iteration, which is also specified by the user in the GUI. If the error exceeds this value the correction is set to its maximum allowed value, otherwise the error is passed through as the correction value.

The actual correction takes place within the set point handler sub-VI of the main logic. After calculating the desired state vector for the next main loop iteration and before calculating spline data for the FPGA, the correction is subtracted from the calculated new position value. Figure 4-7 shows the implementation in Labview.

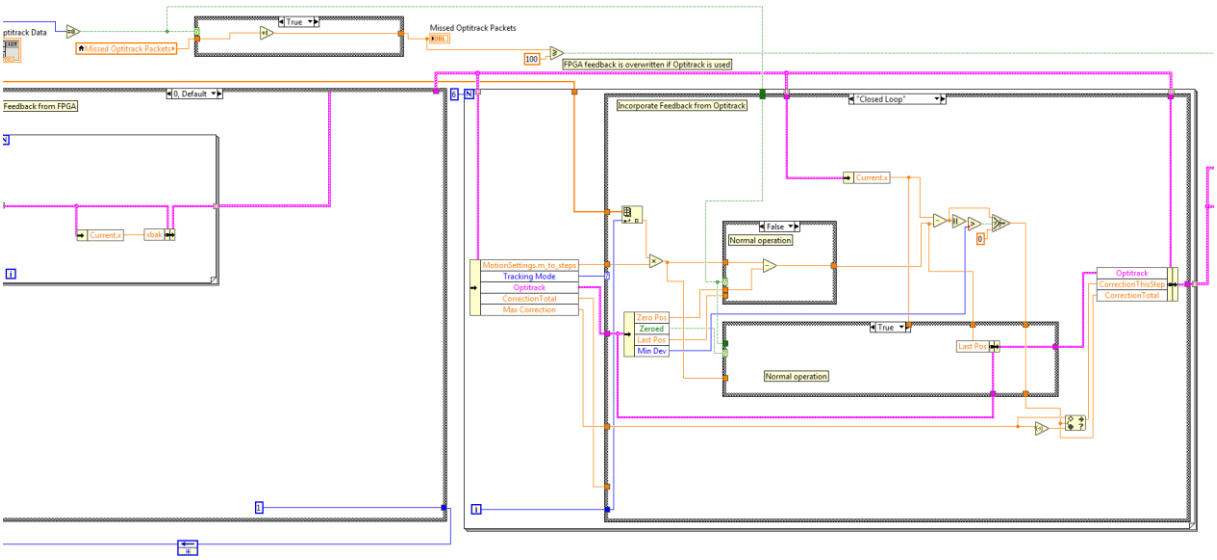


Figure 4-7 Closed Loop Feedback Calculation in Labview

4.2.4 New Operating Modes

Three new operating modes are added by the new software version: manual, sine sweep and trajectory follow. The manual mode is simply a copy of the previous network mode with the network interfacing logic removed. The network mode still exists and will now automatically switch into stopping if the network connection is lost. Connection status is determined the same way as in the Optitrack interface described in section 4.2.3, by checking the validity of the last received UDP packet and declaring the connection as lost if no valid packets have been detected for one second.

Sine sweep and trajectory follow mode use a feature of the setpoint handling part of the software that was previously implemented but not used, namely the hard follow setting. Whenever hard follow is active, the first part of the setpoint handler, which calculates the new state vector based on the last iteration, is skipped and the input state vector is directly used for spline parameter generation. While the setpoint handler normally ensures a smooth acceleration and deceleration to the target, hard follow mode causes more abrupt movements if used to cover large distances. The sine sweep and trajectory follow modes can avoid this by updating the target state vector in small increments on every main loop iteration.

The implementation of the sine sweep mode is very straightforward. The user can set an amplitude and frequency individually for each axis, and the axis will begin oscillating around the position where it is located when sine sweep mode is engaged. Switching into a different operating mode will cause the axis to return to the zero position of the oscillation. To achieve this behavior, the current position of the axis and the current system time are saved whenever sine sweep mode is entered. The target position for the current iteration is then calculated as follows (Equation 4-1):

$$x(t) = A * \sin\big(2\pi f(t - t_0)\big) + x_0 \quad (4\text{-}1)$$

Where $x(t)$ is the position at time t, x_0 is the zero position, t_0 is the start time, A is the amplitude and f is the frequency. Zero position and start time are reset each time the operation mode is switched to sine sweep. The Labview implementation of the sine sweep mode is displayed in Figure 4-8.

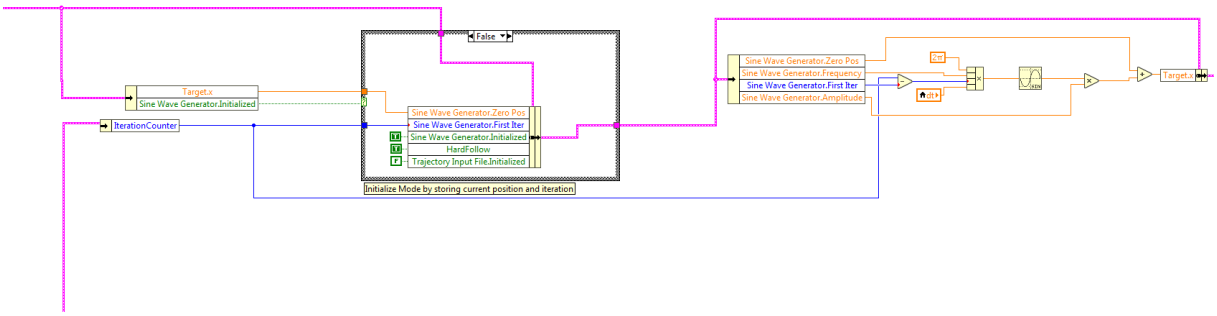


Figure 4-8 Sine Sweep Mode Labview Block Diagram

If more complex preprogrammed trajectories or coordination between different axes are required, the trajectory follow mode can be used. It requires a user generated comma-seperated value (CSV) file to be manually transferred to the CRIO. The CSV files used for testing the mode were created in Matlab, but other programs can also produce valid files. The ORION software expects a file with time indicated in the first column and then six values indicating position of each axis at that time, separated by semicolons (illustrated in Table 4-2). The sequence of columns from left to right is time, chaser X translation, chaser y translation, chaser yaw, chaser pitch, target yaw, target pitch. The data type is expected to be double precision floating point values (doubles). The time step must be 0.01 s as the built-in Labview interpolation funtions are not fast enough to operate at real-time on more than two axes at the same time. Once the trajectory file has been created, the file must be manully transferred to the CRIO. On ORION2, the on-board memory of the real-time system is accessible as a network drive in the Windows Explorer. The file path must then be indicated in the all axes block of the GUI. It is important to note that the file selection box in the GUI looks at the host PC's data system and not at the CRIO, so the user must manually state the path to the trajectory file. The root directory of the CRIO, which is the one visible when using the network drive, is indicated as C:\ in the GUI.

Table 4-2 Trajectory Input File Format (Columns Seperated by Semicolons, Rows as Separate Lines)

Time [s]	Chaser X	Chaser Y	Chaser Yaw	Chaser Pitch	Target Yaw	Target Pitch
0	Value [m]	Value [m]	Value [°]	Value [°]	Value [°]	Value [°]
0.01	Value [m]	Value [m]	Value [°]	Value [°]	Value [°]	Value [°]
0.02	…	…	…	…	…	…
…	…	…	…	…	…	…

When trajectory mode is entered, the entire CSV file is read into Labview on the first iteration, creating a two-dimensional array. On the next iteration, a smaller array is created for each axis in trajectory follow mode containing the time data and the position data for the respective axis. In addition, the current axis position and the current iteration count of the main loop are saved. In every subsequent iteration, the new target

position is determined by subtracting the initial main loop iteration count from the current count and using the result to access the corret line in the position array. The iteration count difference is also multiplied by the time step of 0.01 s and compared to the last value in the time column of the CSV file. Once the end of the file has been reached the axis is switched into stopping mode.

Before forwarding the new target position to the setpoint handler, the zero position is added to it, meaning that the trajectory follow mode always uses the position of the gantry when it was engaged as its starting point. Therefore, it is important to ensure that the trajectory can be started from the current position without running into limit switches or soft limits. Ideally, all trajectories should assume the gantry starting out at the reference search zero position since it is most easily reproducable. The Labview implementation of the trajectory follow mode is shown in Figure 4-9.

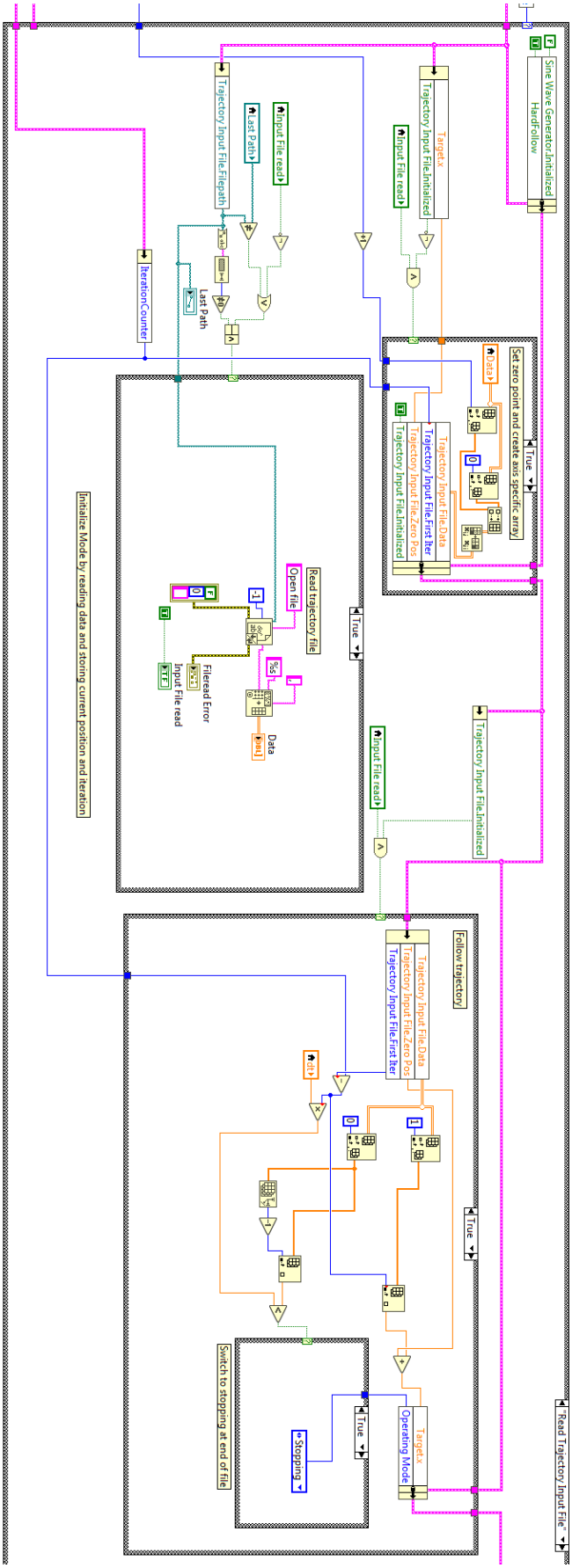


Figure 4-9 Trajectory Follow Mode Labview Block Diagram

4.2.5 Data Logging

The final major new feature of the software is data logging. All its logic is contained in a seperate VI, named *ORION Data Logger*. All internal variables are wired into it and thus are accessible for logging. Currently the VI is set up to log the current state vectors of all axes, the spline data transmitted to the FPGA, the elapsed time since the start of the recording in milliseconds and the current Optitrack position of all axes as illustrated in Table 4-3. The data is saved in a CSV file on the root directory of the CRIO. As for placing trajectory input files, it is accessible as a network drive on ORION2. The user only needs to press the record button in the GUI to start a recording and press it again to stop. The data file created will automatically be given a name indicating the date and time at the start of the recording. The user can then analyze the data in external programs. For testing purposes, a Matlab script was created for quickly plotting state vector and spline data. The Labview block diagram of the data logger is displayed in Figure 4-10.

Table 4-3 Data Reording File Format (Columns Seperated by Semicolons, Rows as Separate Lines)

Time [ms]	Current State Vectors (18 Columns)	Splining Data (18 Columns)	Optitrack Positions (6 Columns)
0	Position, Velocity, Acceleration for each axis	Three parameters per axis	One position per axis
10	…	…	…
20	…	…	…
…	…	…	…

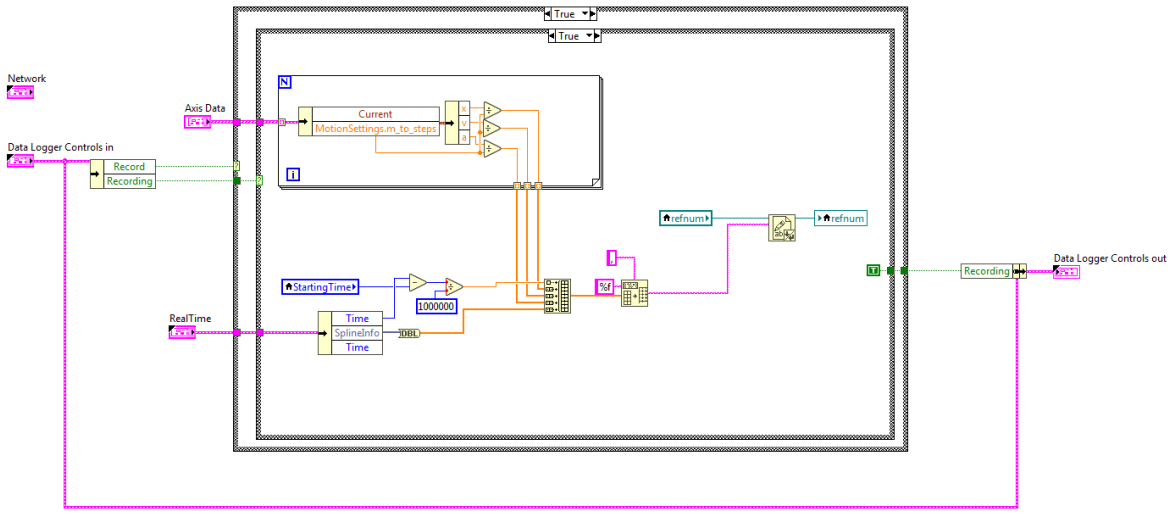


Figure 4-10 Data Logger Labview Block Diagram

For sufficient performance in a real-time application, opening and closing the file in each main loop iteration is too slow, which is what happens if the simplified Labview save functions are used. Instead, it is required to open the file on the first iteration and obtain a reference which is then passed to all subsequent iterations and enables them to write into the open text file. At the end of the recording the file is then should be closed to ensure all data has been successfully written and the file is not corrupted. This is complicated slightly by the fact that the data logger VI is contained within the main loop. The implementation requires a nested switch structure and a status variable in addition to the toggle switch. When the switch is toggled on, the new file is created and opened and its reference ID is stored as a local variable. The status variable is then set to true. While the button is toggled on and the status variable is true, data is written to the file in each main loop iteration. If the button is toggled off and the status variable is still true, the file is closed and the status variable set to false. If the button is off and the status variable is false, the VI does nothing.

4.3 Hardware Changes

The main translational axes (X and Y) of the ORION gantry are equipped with mechanical limit switches to prevent the chaser assembly from running off the rails. The original hardware simulator operating software reads the status of these switches from the FPGA on each iteration and stops the corresponding axis if an end switch is triggered. This means that if the software does not correctly recognize the triggering of the switch, or does not correctly enter stopping mode, the chaser may overrun the end switch, causing damage to the facility. While this is generally not an issue during normal operation with a well tested and documented operating software, it makes code testing during development riskier. An issue was in fact encountered where the chaser overran the end switch and derailed on the X axis during work on this thesis, and while the root cause was discovered and fixed, this prompted an upgrade to the gantry hardware.

Three normally open solid-state relays are now wired into the main power supply of the simulator in series. The end switches for the X and Y axis are each connected in series to one of the relays, while the third one is activated by an emergency stop switch located next to the operating PC. If one of the end switches is triggered, or if the operator presses the emergency stop button, the power supply to the gantry is cut, stopping all axes in place immediately. The operating software can still detect a triggered emergency stop and inform the user, although it can no longer differentiate between front and back emergency switches on an axis since they are now connected in series instead of the previous parallel setup. The relays are shown in Figure 4-11 and the wiring diagram in Figure 4-12.

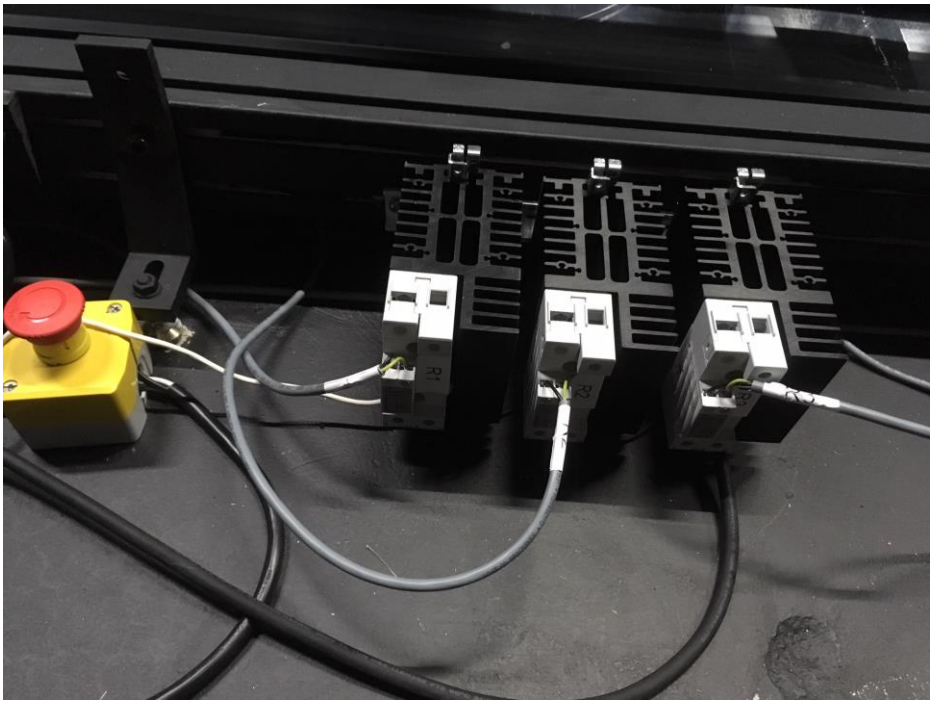


Figure 4-11 ORION Emergency Stop Relays

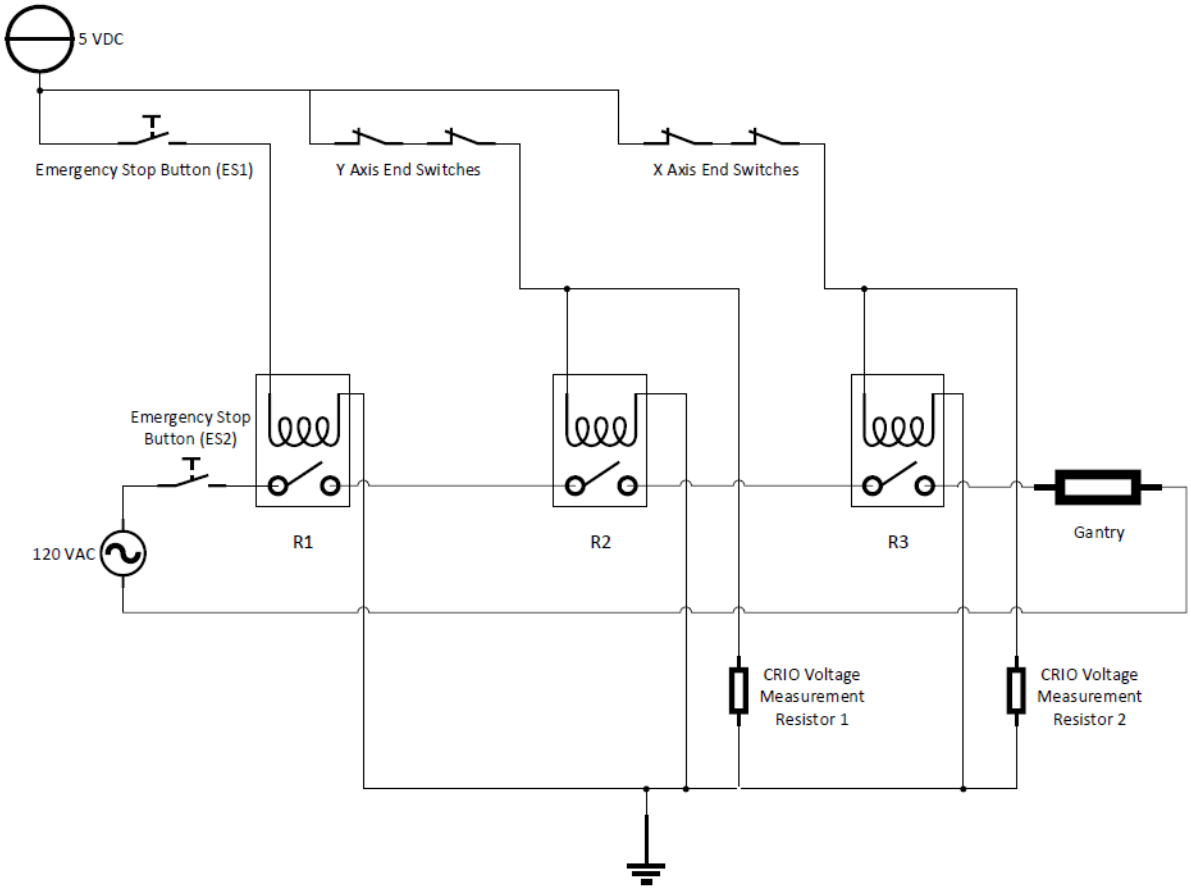


Figure 4-12 ORION Emergency Switch Wiring Diagram

4.4 Testing

The test program run on the ORION gantry for this thesis has two objectives: verifying the requirements and using the new system capabilities, mainly the Optitrack interface, to quantify several important parameters that could only be estimated so far. These include system positioning accuracy and conversion factors into motor steps. As Table 4-1 in section 4.1 shows, most requirements are to be verified by qualitative demonstration and do not require measurements. The main challenge is the large number of potential operating mode combinations, which requires a systematic approach to ensure no required mode combination is overlooked. One such approach is using a validation matrix as shown in Table 4-4. All operating modes are tested in both open and closed loop tracking configuration. The test program for the various operating modes is:

- Manual: move each axis forward to its soft limit and then back to zero.
- Reference Search: run the reference search algorithm for all axes.
- Network: command gantry movements by commands streamed via UDP in real-time.
- Trajectory Follow: load a test file as described in section 4.2.4 and activate the mode on all axes, wait until the system has run through the entire test file.
- Sine Sweep: set up all axes with reasonable values for amplitude and frequency and run the mode for a few minutes.
- Stopping/Lock Axis: with a gantry axis moving in any other mode, switch over to stopping or engage the axis lock and check if it stops correctly.

Table 4-4 Mode Validation Matrix

Operating Mode	Open Loop	Closed Loop
Manual	✓	✓
Ref Search	✓	Not tested
Network	✓	✓
Trajectory Follow	✓	✓
Sine Sweep	✓	✓
Stopping	✓	✓
Docked	Not implemented	Not implemented
Lock Axis	✓	✓

The target pitch and yaw axes can currently not be tested in closed loop mode because the target lacks Optitrack markers. A wooden test box fitted with some markers was used to create the correct number of tracked bodies and test the Optitrack interface. Otherwise, all operating modes currently seem to work fine in both closed and open loop modes for all chaser axes.

The conversion factor between external position units (meters and degrees) and internal motor steps is a very important but previously unverified parameter of the system. The values used before were gained from measurements taken on the hardware. With the Optitrack interface now available, it is possible to command the gantry to move by a certain distance on one axis, then use Optitrack to verify the actual

distance travelled and apply corrections to the conversion factor. Only small corrections were required, but some improvements were still made. The new default conversion factors are listed in Table 4-5. Translational axes show conversion from meters to steps, rotational axes from degrees to steps. If the user wishes to adjust these factors for any reason, they are accessible in the advanced options panel of the GUI.

Table 4-5 Conversion Factor to Motor Steps (Cha: Chaser, Tar: Target)

Cha X	Cha Y	Cha Yaw	Cha Pitch	Tar Yaw	Tar Pitch
170358	170358	536	1778	536	1778

Using the Optitrack interface, it is now also possible to measure the system accuracy. It is limited by the remaining noise in the filtered Optitrack signal, which must be suppressed by the minimum Optitrack deviation as explained in section 4.2.3 to pevent oscillations. Position corrections are only applied if the deviation is larger in magnitude than this minimum, so it provides an upper limit to system accuracy. It is therefore important to find a value for minimum deviation that is as small as possible while still suppressing noise. Usable values can be found by observing one of the main loop sub-VI GUIs, which shows the correction in the next step as a variable. The minimum Optitrack deviation can be lowered using the main GUI advanced options panel with the gantry stationary until the correction for the next step is no longer zero. An ideal minimum deviation value is just high enough to not cause corrections to be applied while the system is zeroed and stationary. The default values found for minimum deviation are listed in Table 4-6. To find them, the gantry was first put into closed loop mode while stationary with the minimum deviation set to zero. This causes constant small movements of the gantry as the algorithm tries to correct for the measurement noise. The minimum deviation is then slowly increased in steps until the gantry stops moving. The smallest vaue that does not lead to position corrections with the gantry stationary is recommended for minimum Optitrack deviation.

Table 4-6 Minimum Optitrack Deviation (Translational Axes: Meters, Rotational Axes: Degrees)

Cha X	Cha Y	Cha Yaw	Cha Pitch	Tar Yaw	Tar Pitch
0.001	0.001	0.1	0.3	0.1	0.3

The main advantage of using a closed loop tracking system relying on external position data is that position drift over time can be avoided. Mechanical slip and other disturbance forces in the system are usually not exactly equal for both directions of movement. Whenever an axis moves forward and then back in open loop mode, it never returns to the exact starting position since the internal disturbances cannot be accounted and corrected for by simply counting motor steps. In a closed loop system, an external reference is available to quantify and correct these errors.

To demonstrate this capability, it is first necessary to determine drift in open loop mode. It can be measured by getting position data before and after a movement operation from the Optitrack interface. The gantry system drifts relatively little in a single motion. In a test running the x axis forward and back by 2 m ten times, the zero position drifted by just 0.4 mm, which is below the normal Optitrack correction range. In this case, the closed loop performance would have been the same. Drift is mostly a problem for long, continuous movements. In a second test, the Y axis was put into sine sweep mode with an amplitude of 0.5 m and a frequency of 0.05 Hz. After five minutes, the axis was returned to zero and the drift was measured. In open loop mode, the zero position drifted by 1.7 mm. In closed loop mode, it only changed by 0.2 mm. It is important to point out that while an even longer runtime in open loop mode would cause an even larger drift, in closed loop mode the error magnitude remains constant over time. Drift was roughly characterized for all four chaser axes. It should be noted that different movement profiles will produce different results, but it is at least possible to find the magnitude of expected drift after a five-minute running time. While the Y axis and the pitch axis are noticeably affected, there is no measureable drift in the yaw axis, and X axis drift is very small. These finding, shown in more detail in Table 4-7, can be used when designing future experiments.

Table 4-7 Chaser Axis Position Drift After Five Minutes of Uninterrupted Sine Wave Motion

Axis	Amplitude	Frequency	Drift
X	0.3 m	0.05 Hz	0.3 mm
Y	0.5 m	0.05 Hz	1.7 mm
Yaw	30°	0.1 Hz	0°
Pitch	12°	0.32 Hz	0.45°

4.5 Example Use Case

The first experiment run on the ORION gantry using the new software provides a good use case for the new software capabilities. In a different Master's thesis at FIT, a control algorithm for autonomously landing a drone on a ship's deck is being developed. In the laboratory setup, the drone used is a small quadcopter, with positioning provided by Optitrack. The drone is shown in Figure 4-13. The gantry chaser is used to simulate the ship's deck in this experiment. A mockup landing platform is fitted to the instrument carrier platform, and the chaser motors are then used to simulate deck movement due to waves. A rough approximation is possible using the new Sine Sweep operating mode, and a more exact reproduction of realistic deck movement including random elments can be achieved using the Trajectory Follow mode.

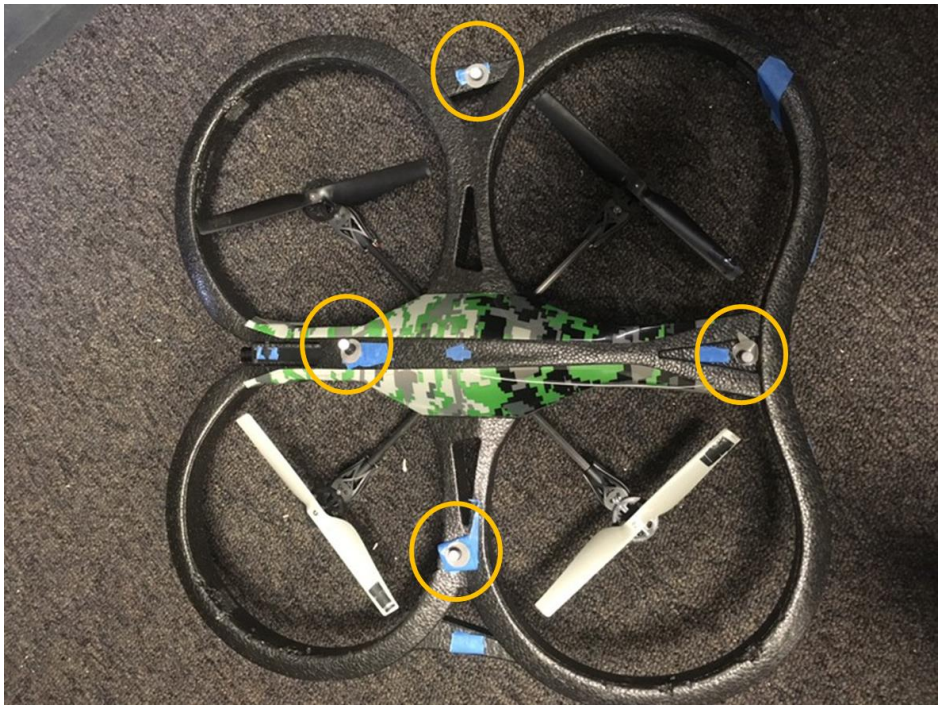


Figure 4-13 Test Drone with Optitrack Markers (Circled)

The drone itself is controlled by a Matlab/Simulink script running on the gantry control computer, using Wireless Local Area Network (WLAN) for data and command transfer. The mockup landing platform was not finished in time for this thesis so the actual landing tests could not be carried out, but it is still possible to demonstrate several new features in a preparatory test case. They include the sine wave operating mode, the data logger and the Optitrack data feedback. An amplitude of 12° and a frequency of 0.32 Hz are required for the pitch axis for the drone landing test. First, the gantry is run in open loop mode. Using the data logger, the internally determined position can be plotted against the Optitrack measurement as shown in Figure 4-14. Some Optitrack data packets are lost during transmission, most notably visible around the six second mark, but the loss rate is considered acceptable. The Optitrack signal is also delayed by about 0.05 s, which provides an estimate for the system latency introduced by capturing, processing and transmitting the position data. In open loop mode, the Optitrack measurement shows that the gantry does not fully reach the commanded amplitude of 12°. It instead stops short by about 0.5° at each inversion point. The reason for this is likely internal mechanical slip in the system which the open loop mode can not correct for.

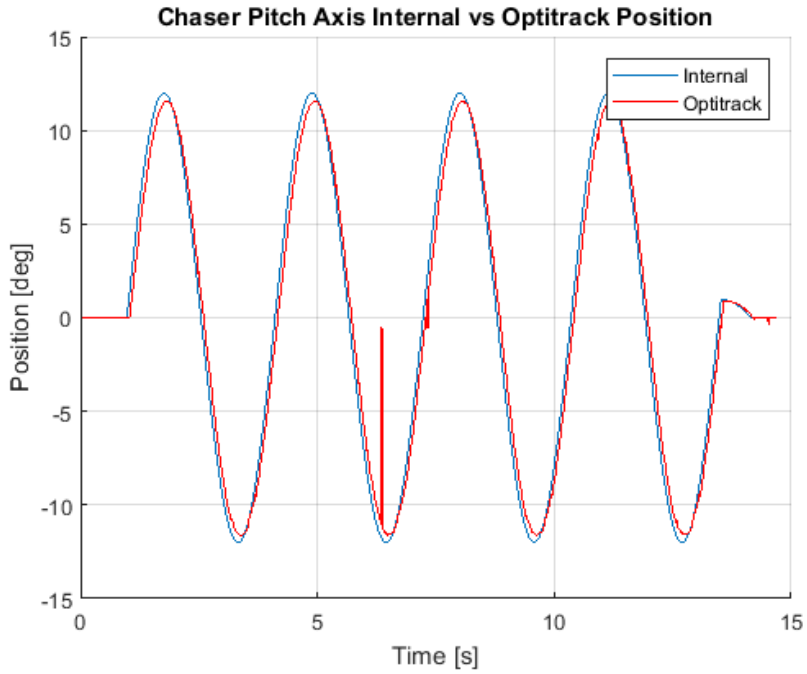


Figure 4-14 Chaser Pitch Axis Position Comparison Between Internal Data and Optitrack Measurement in Open Loop

For comparison, the same test was repeated with the gantry operating in closed loop mode. Using the external position feedback, the mechanical slip can now be corrected and the axis correctly moves to the commanded positions (Figure 4-15). While the open loop accuracy is certainly sufficient for the drone landing experiment, this example illustrates the new capabilities of the system enabled by closed loop Optitrack feedback.

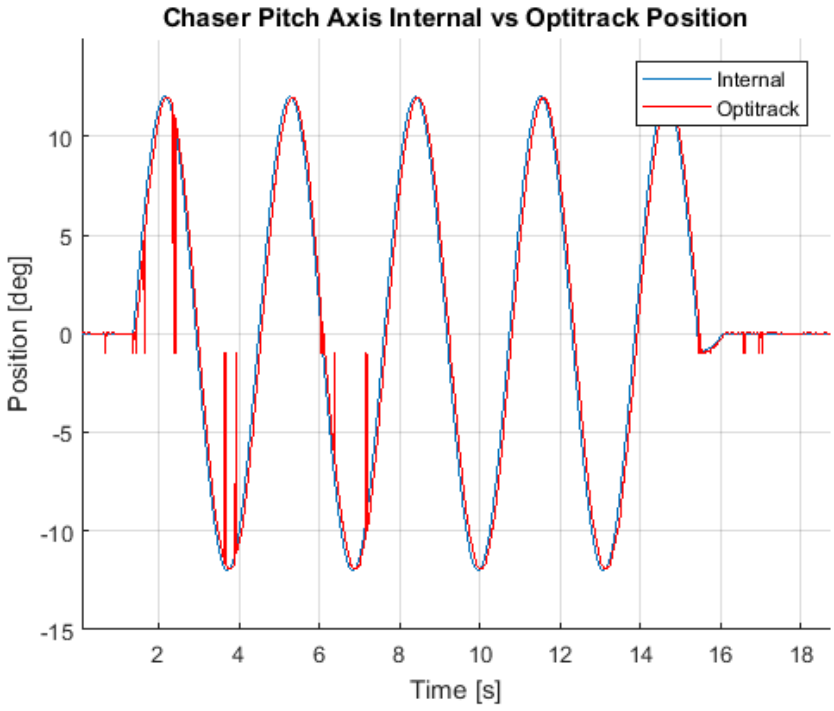


Figure 4-15 Chaser Pitch Axis Position Comparison Between Internal Data and Optitrack Measurement in Closed Loop

4.6 Verification and Discussion

Having finished the software development process, it is time to look back at the requirements defined in section 4.1 and check how well they are fulfilled by the finished product. Table 4-8 displays the verification results for each requirement, based on the system description and the tests explained in the previous sections. As the table shows, all requirements are at least partially fulfilled. The initially planned option to command a constant velocity motion on any axis until an end switch is reached was cancelled during development because the required added software complexity was not deemed worth the limited additional capability. The requirement was kept, however, in case this feature is selected to be developed in the future. Getting every sub-VI on one screen without scrolling would require an additional level of subdivision, which would not help readability and was thus omitted. The only known issue impacting performance is a slowdown in trajectory follow mode that occurs if multiple axes are switched into trajectory follow mode. Sometimes this causes the software main loop to run at less than 100 Hz even if the axes are subsequently switched to a different mode. The reason for this is unknown and the error is hard to reproduce. It can be fixed by restarting the software.

Table 4-8 Requirement Verification for the Updated ORION Software

ID	Requirement	Fulfilled	Comment
1	The software shall be programmed using NI Labview 2016.	Yes	
2	The real-time control software shall be implemented on the existing NI cRIO 9024 real time controller.	Yes	
3	The control software shall be able to command all axes of the ORION gantry system: Chaser X and Y position, Chaser pitch and yaw, Target pitch and yaw	Yes	
4	The software shall provide the following information to the operator at all times in a GUI on the host computer (ORION2): 1. Current and commanded position, velocity and acceleration of all axes 2. Critical system status information (Connection status, errors etc.) 3. Status of end and reference switches	Yes	
5	The operator shall be able to make the following adjustments in the GUI on the host computer (ORION2) as needed: 1. Target position for all axes 2. Maximum velocity and acceleration for all axes 3. Software defined limit positions for all axes 4. Position tracking mode 5. Positioning mode	Yes	

6	The GUI provided by the software shall use the following units: meters for position, degrees for angles	Yes	
7	In case of any critical error, the software shall inform the operator via the GUI on the host computer (ORION2) and return the system to a safe state.	Yes	Fault cases tested: soft limit reached, end switch reached, Optitrack/Network connection loss
8	The software shall support controlling the gantry using the following methods for position determination and enable the operator to switch between them for each axis: 1. Open-loop guidance, calculating the state vector internally 2. Optitrack closed-loop guidance, determining the state vector from position data gathered through the Optitrack system and streamed in from the ORION3 computer	Yes	
9	The software shall support the following gantry motion commands and enable the operator to switch between them: 1. Move axis to set position 2. Move axis at set velocity	Partially	Constant velocity mode cancelled
10	The software shall be able to accommodate the following modes of setting the target position for each axis: 1. Direct input in the GUI on the host computer (ORION2) 2. Constantly updating input stream from the simulation computer (ORION1) 3. A configurable sine wave generator in the GUI on the host computer (ORION2) 4. Input (and loop) any pregenerated command sequence on the host computer (ORION2)	Yes	
11	The software shall account for the following methods of determining the maximum motion range and enable the operator to switch between them: 1. Software defined limit positions 2. Magnetic reference switches on each axis	Yes	
12	The software shall be able to command any axis of the gantry system to follow a user-specified sine shape curve.	Yes	

13	The software shall be able to command the gantry to follow an arbitrary trajectory input by the user in a specified format.	Yes	
14	The software shall provide an option to record a time series of any internal variables for later analysis.	Yes	Currently configured to record target and current state vector and spline coefficients
15	The software shall detect the activation of the axis limit switches and stop any movement of the system if they are triggered at any time.	Yes	
16	The software shall enable repeatable positioning accuracy to within 1 mm in all translational and 1 degree in all rotational axes within an external reference frame when operating in closed-loop mode.	Yes	Tested in manual mode
17	The software shall send updated position commands to the FPGA at a rate of 100 Hz in all operating modes.	Partially	Sometimes slowdowns experienced in trajectory follow mode
18	All Labview VI front panels visible to the operator in normal operation should fit on a 1920 x 1080 pixel monitor without scrolling.	Yes	
19	All Labview VI block diagrams should fit on a 1920 x 1080 pixel monitor without scrolling.	Partially	Some main loop VIs require scrolling

It is hard to compare the usability of the new software to other laboratories, both because it is difficult to define a suitable metric and because the exact specifications for GUIs and source code are usually not published. The new, clearer structure of GUI and code should, however, help in utilizing one of the main advantages that RACOON and ORION have over simulators based on COTS robots, namely the full control over and customizability of the custom software and hardware system. It should be easier to add and debug new features in the future. The positioning accuracy in closed loop tracking mode, at 1 mm for translation and 0.3 degrees for rotation is well below the capabilities of simulators based on modern industrial robots as described in section 1.1, but sufficient for all experiments planned at the ORION laboratory. As described in section 3.2.3, the positioning accuracy of the ORION Optitrack system was found to be comparable to other implementations of this and other optical tracking systems, and signal filtering was identified as an interesting avenue for further research which might further increase accuracy.

5 Adaptation for RACOON

While the operation software for ORION and RACOON shares a common base, the differences in configuration of both systems need to be reflected in the programs. This section lists all known changes required to adapt all software developed and tested in this thesis on ORION to RACOON. Section 5.1 explains how to adapt the Optitrack interface, and section 5.2 the gantry control software.

5.1 Optitrack-Labview-Interface

The interface between Optitrack and Labview is designed to be as application independent as possible, as explained in section 3. Still, there are some fundamental differences in the network topology between the two laboratories which unfortunately require changes on the DLL level of the interface. In the ORION lab, all computers involved in running the simulator are connected on a separated network without firewall protection. The laboratory is a multi-user environment, and especially during development the Optitrack system is often not exclusively used to control the gantry. To allow streaming Optitrack data to multiple users, the multicast option is used in Motive, allowing several clients to receive data from the same server. In the RACOON lab, on the other hand, the control and telecommand computers are located physically distant from each other and are connected via university-wide intranet. The connection is protected by a firewall which does not allow multicast UDP connections. In addition, the Optitrack system at TUM is used exclusively for the RACOON simulator, so unicast is the better solution. Switching from multicast to unicast requires editing a variable in the OptitrackConnect.h source code file and then recompiling the DLL.

The RACOON system makes use of a configuration file written in the Extensible Markup Language (XML). It is recommended to use this file for obtaining the IP address of the Optitrack server to keep all important system parameters in one place. Reading the configuration file can be implemented in Labview in the *Optitrack Connect Main VI*, or the C++ code can be edited. A function called *ReadFromXMLFile* is already included in the source code, specifically adapted to the RACOON configuration file format, but is currently unused.

The largest change needed in Labview is in the mapping function, which must be updated for the larger number of axes used in RACOON (12 as opposed to 6 in ORION). The mapping function is always application specific. It is important to thoroughly test the mapping function before using it for closed loop control since a false axis assignment or even just a wrong sign can be hard to diagnose from observing the feedback in the control software alone.

In addition to these larger changes, the UDP connection setup (IP address and ports) in Labview for forwarding data to the CRIO must be changed in Labview. If the software is to be executed in a 64-bit version of Labview, it may also be necessary to update the size of pointers used in some sub-VIs as described in section 3.2.2.

5.2 Control Software

Adapting the new ORION control software to RACOON requires deeper changes compared to the Optitrack interface. The larger number of axes of the RACOON system (12 as opposed to 6 in ORION) requires updates to the user interface as it may not be possible to fit all of them on screen at one time. It is, however, still recommended to implement this solution if possible. One option may be splitting the data for target and chaser into two arrays arranged on top of each other on the screen and combining them in the axis data handler. In many places where axis data is processed in a for-loop, the number of loop iterations is currently hard coded to 6 for maximized performance. This must be changed to 12 instead. IP addresses and ports for both UDP connections must also be updated, they are defined either in the initialization file or in the GUI block diagram.

6 Conclusion

Within this thesis, a new version of the operating software for the ORION hardware simulator has been developed. Closed loop control is now available using the Optitrack system installed in the laboratory for absolute position determination, increasing the achievable positioning accuracy of the system. Position control is now possible to within 1 mm for translational and 0.3 degrees for rotational axes without drift in long-term operation. The updated user interface should make it much easier for new users to understand and operate the system. New capabilities are enabled by the added sine sweep and trajectory following operation modes. The system is now also safer for testing software changes due to the addition of an emergency off switch at the operator's station and due to making the operation of the end switches software independent. This thesis documents all changes made and should serve as an introduction to the system for future operators.

So far, the ORION gantry system has not been used much in experiments, so further changes and updates may be required down the line. It is recommended to keep documenting these changes so new operators and programmers always have a reference of the current system state to work with. The lack of existing documentation was a major problem in the creation of this thesis. If more software changes are planned in the future, it is also recommended to implement a content management system to ensure the current software version is always in use, document changes and enable backtracking if major problems are encountered with a new version.

While all currently planned changes for the ORION laboratory have been implemented, the planned use of the same new software to operate the RACOON facility remains a major work package which will require changes to the software as detailed in section 5. It also remains to be seen how well the current RACOON Optitrack installation will work with the software developed in this thesis. Hardware upgrades may be necessary to achieve comparable precision to the system at FIT. It is also recommended to search for a further optimized filtering solution for the Optitrack position signal, as even a very simple filter can produce a significant reduction in error.

A References

[1] O. Ma, A. Flores-Abad und T. Boge, „Use of industrial robots for hardware-in-the-loop simulation of satellite rendezvous and docking," *Acta Astronautica,* pp. 335-347, December 2012.

[2] Lockheed Martin, „SOSC Facility Description," 2018. [Online]. Available: https://www.lockheedmartin.com/us/ssc/sosc.html. [Zugriff am 23 03 2018].

[3] KUKA Roboter GmbH, „Indutrial Robotics - Medium Payloads," 2017.

[4] R. Zappulla, J. Virgili-Llop, C. Zagaris, H. Park, A. Sharp und M. Romano, „Floating Spacecraft Simulator Test Bed for the Experimental Testing of Autonomous Guidance, Navigation, and Control of Spacecraft Proximity Maneuvers and Operations," in *SPACE Conferences and Exposition*, Long Beach, California, USA, 2016.

[5] K. Saulnier, D. Pérez, R. Huang, D. Gallardo, G. Tilton und R. Bevilacqua, „A six-degree-of-freedom hardware-in-the-loop simulator," *Acta Astronautica,* pp. 444-462, October 2014.

[6] B. Milosevic, R. Naldi, E. Farella, L. Benini und L. Marconi, „Design and validation of an attitude and heading reference system for an aerial robot prototype," in *Proceedings of the American Control Conference*, Montréal, Canada, 2012.

[7] IEEE, „IEEE Spectrum," 17 03 2017. [Online]. Available: https://spectrum.ieee.org/aerospace/satellites/inside-darpas-mission-to-send-a-repair-robot-to-geosynchronous-orbit.

[8] Florida Institute of Technology, „ORION Website," 27 02 2018. [Online]. Available: https://research.fit.edu/orion/.

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

[10] Natural Point, „Optitrack Documentation Wiki," 28 11 2017. [Online]. Available: https://v20.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki.

[11] J. Otto, „Development of an Absolute Reference Measurement System in the Racoon Hardware Lab," Technische Universität München, 2016.

[12] Natural Point, „Optitrack Hardware Overview," 27 02 2018. [Online]. Available: http://optitrack.com/hardware/.

[13] Natural Point, „NatNet User's Guide," 2016.

[14] National Instruments, „Using External Code in Labview," 2000.

[15] J.-L. Blanco, „A tutorial on Se(3) Transformation Parameterization and On-Manifold Optimization," University of Malaga, 2010.

[16] National Instruments, „Labview Development Guidelines," 2003.

B Appendices

B.1 Procedures

This section provides procedures for the main tasks required to operate the ORION gantry. It should help new operators to avoid errors and to speed up their learning process. Tasks described include starting up the gantry system, setting up Optitrack (required for closed loop tracking mode) and shutting the system down.

Startup

1. Visual inspection: Make sure no foreign objects that might interfere with the gantry motion are located anywhere on the hardware simulator, check that the system is in working order.
2. Plug the power supply cable for the gantry into a wall socket.
3. Disengage the emergency stop button located next to the emergency stop relays on the hardware simulator. This requires twisting the switch to unlock it.
4. Start up the operating PC (ORION 2).
5. Turn on the power supply for the CRIO.
6. When the operating PC has finished booting, load Labview 2016 32 bit.
7. Open the ORION Software 2.0 Project in Labview.
8. Run the ORION GUI VI contained in the project. This should deploy all required software to the CRIO.
9. Make sure the GUI has started up correctly and is not commanding any axes to move.
10. Disengage the emergency stop button located next to the operating PC by pulling it upwards. You should see the indicator LEDs on all motor controllers of the gantry turn on. The system is now ready to execute movement commands.
11. Put all axes you intend to use into reference search mode and wait until the algorithm has finished and the system has returned them to stopping mode.
12. The system is now initialized and ready to operate.

Optitrack Setup

1. Start up the Optitrack PC (ORION 3).
2. Turn on the power supply for the Optitrack cameras.
3. Turn on the Wifi router on the shelf with the CRIO and power supplies.
4. Once ORION 3 has finished booting, run the Motive software.
5. In Motive, open the latest calibration file or run a new calibration process.
6. Find the markers for the ORION chaser and target in the 3D environment shown in Motive and designate them as part of rigid bodies. If only one body is required proceed as follows:
    a. If only the chaser is tracked, no second body is required. Make sure you do not switch the target axes into closed loop mode in the ORION GUI.
    b. If only the target is tracked, use a dummy body in place of the chaser. Make sure you do not switch the chaser axes into closed loop mode in the ORION GUI.

7. Ensure that the chaser is listed in the Optitrack project panel as Rigid Body 1 and the target as Rigid Body 2.
8. Activate frame broadcasting in the Motive streaming panel. Make sure the broadcat IP is 192.103.1.3. This concludes setup on ORION 3.
9. On ORION 2, open the OptitrackUDP Labview project in Labview 2016 32 bit.
10. Run the OptitrackInterfaceMain VI. Confirm that Optitrack data is being received (no errors and the plot is updating).
11. Change the filter cutoff frequencies if necessary.
12. Activate data filtering.
13. Check that the CRIO is receiving data from Optitrack (Optitrack Connection Lost warning light is off in the GUI)
14. The Optitrack connection is now set up and the system can be switched into closed loop tracking mode. It is recommened to drive an axis to the zero position determined in reference search before engaging closed loop tracking, otherwise the soft limits should be updated in the advanced settings panel of the ORION GUI to prevent the axis from running into the limit switches.

Shutdown

1. Switch all axes into open loop tracking mode.
2. Move all axes back to their zero positions to speed up the reference search on the next startup.
3. Copy any data files from the CRIO to a storage you want to keep them in.
4. Press the emergency stop button next to the operating PC (ORION 2)
5. Stop execution of the ORION GUI VI using the abort execution button.
6. Stop execution of the Optitrack interface VI using the stop button.
7. Close all Labview windows.
8. Close Motive on ORION 3.
9. Turn off the power supply for the CRIO.
10. Turn off the power suppy for the Optitrack cameras.
11. Shut down the ORION 2 and ORION 3 PCs.
12. Press the emergency stop button located next to the gantry emergency stop relays.
13. Pull the gantry power plug from the wall socket.

B.2 Appended Files

The appended files to this thesis contain:

- Ausarbeitung:
  - Thesis text in both docx and pdf format
  - All figures displayed within the thesis
- Literatur:
  - All references used in this thesis in pdf format
- Messdaten:
  - Raw data recording files from running the ship deck movement test described in section 4.5
- Programmcode:
  - The source code and assorted files for the Optitrack Interface DLL
  - The source code and assorted files for the C++ test environment for the Optitrack Interface DLL
  - The Labview code for the Optitrack Interface
  - The Labview code for the gantry control software
  - The Matlab scripts for generating an example trajectory file and for reading the result files