Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

NATIONAL INSTRUMENTS™

Thema:

Realisierung eines Instrumententreibers zur Einbindung der Microsoft Kinect in LabVIEW.

Verfasser: Florian Abry

Betreuer: Prof. Dr.-Ing. Günther Kastner

Dipl.-Ing. Jochen Klier

Studiengang	Elektrotechnik und Informationstechnik
Studienrichtung	Automatisierungstechnik
Verfasser	Florian Abry
Matrikel-Nr.	18789
Geburtsdatum	30.09.1987
Betreuer Hochschule	Prof. Dr.-Ing. Günther Kastner
Betreuer bei der Firma	Dipl.-Ing. Jochen Klier
Abgabetermin	31.08.2011

Erklärung

Die vorliegende Diplomarbeit wurde von mir selbst verfasst und nur mit den angegebenen Quellen und Hilfsmittel angefertigt.

München, im August 2011

_____

Florian ABRY

Danksagung

Diese Diplomarbeit wurde in der Zeit vom 01.06.2011 bis 31.08.2011 bei der Firma National Instruments, in der Abteilung Applications Engineer angefertigt. Für die Aufgabenstellung und die Bereitstellung des Arbeitsplatzes möchte ich mich bei National Instruments bedanken.

Ein besonderer Dank gilt meinem Betreuer Herrn Dipl.-Ing. Jochen Klier, der mich während dieser Zeit unterstützt hat. Ebenso möchte ich mich bei Herrn Dipl.-Ing. Daniel Riedelbauch, Herrn Dipl.-Ing. Robert Hallas und Herrn Dipl.-Ing. Christian Mergl sowie allen Mitarbeitern der Abteilung Application Engineer und den anderen Abteilungen bedanken, die sich für Fragen und Probleme meinerseits Zeit nahmen und Anregungen für Lösungen gaben.

Vonseiten der Hochschule Ravensburg-Weingarten bedanke ich mich bei meinem Betreuer Herrn Prof. Dr.-Ing. Günther Kastner für seine Unterstützung.

Florian Abry, im August 2011

Inhaltsverzeichnis

1 Kurzfassung

Die vorliegende Arbeit liefert die Ergebnisse einer Untersuchung zur Entwicklung eines Instrumententreibers zur Einbindung der Microsoft Kinect in LabVIEW.

Die Arbeit befasst sich zuerst mit dem Verständnis von der Kinect Technologie und seine mehrere Einbindungsmöglichkeiten auf einem Windows Computer.

Danach wird mit C++ und Visual Studio versucht, die Instrumententreibers als DLL zu entwickeln, die von LabVIEW benutzbar ist. Diese versuch war leider ein Sackgasse.

Im nächsten Teil der Arbeit werden die Treiber mit Hilfe eine .NET Assemby in LabVIEW entwickeln. Diese Treibers werden auch veröffentlicht.

Als Abschluss der Arbeit werden diese Treibers in einer praktischen Aufgabe getestet. Ein Steppermotor sollte mit den Menschen Positionsdateien der Kamera gesteuert.

2 Aufgabenstellung und Ziel der Bachelorarbeit

Um mein Studium an der Hochschule Ravensburg-Weingarten beenden zu können, ist das Verfassen einer Bachelorarbeit notwendig. Im Wesentlichen wird diese Arbeit basierend auf einem 3-montatigen Projekt sein, das entweder an der Hochschule oder in einer Firma durchgeführt wird. Ich habe mich entschieden, das Projekt bei der Firma National Instrument, einem amerikanischen Mess- und Automatisierungstechnik-Unternehmen, zu absolvieren. Dieses Dokument soll die grundsätzliche Aufgabe von meinem 3-monatigen Projekt beschreiben, was die Entwicklung einer LabVIEW Schnittstelle für Kinect sein wird um ein Newton Pendel steuern zu können. In diesem Dokument werde ich zuerst die Kinect Technologie sowie das Newton Pendel vorstellen und dann erklären, was ich damit machen soll.

Abbildung 2-1: Kinect Kamera(http://www.xbox.com/kinect)

Kinect ist eine von Microsoft entwickelte Kamera, die sowohl als 3D Kamera als auch als Spielcontroller fungiert. Ursprünglich war es als alternativer Controller ohne Manipulatoren für die XBOX 360 gedacht (XBOX 360 ist die aktuellste Spielkonsole von Microsoft). Aufgrund der hohen Stückzahlen ist dieses Gerät die billigste am Markt verfügbare 3D Kamera und deswegen sehr attraktiv für Forschungsprojekte. Seine Funktion macht es perfekt für die Bewegungs- und Objekterkennung im Raum.

Abbildung 2-2: Ein Newton Pendel, auch Kugelstoßpendel genannt (http://www.istockphoto.com)

Das Newton Pendel wurde ursprünglich nicht von Isaac Newton erfunden, sondern nur nach ihm benannt. Dieses Gerät demonstriert den Impuls- und Energieerhaltungssatz. Es besteht aus einer Reihe von Metallkugeln, die alle an einem Rahmen aufgehängt sind, sodass sie einander berühren. Wenn eine Kugel am Ende der Reihe hochgehoben und anschließend losgelassen wird, so wird die potentielle Energie in kinetische Energie umgewandelt und an die Nachbarkugel weitergegeben. National Instruments besitzt ein motorisiertes Pendel, welches die Funktion eines echten Newton-Pendels z. B. bei variabler Schwerkraft simulieren kann. Dieser mechanische Aufbau soll für mein Projekt benutzt werden. Das Ziel ist nun, das Pendel über einfache Bewegungen der Körperextremitäten und anhand der Kinect Kameraaufzeichnungen steuern zu können.

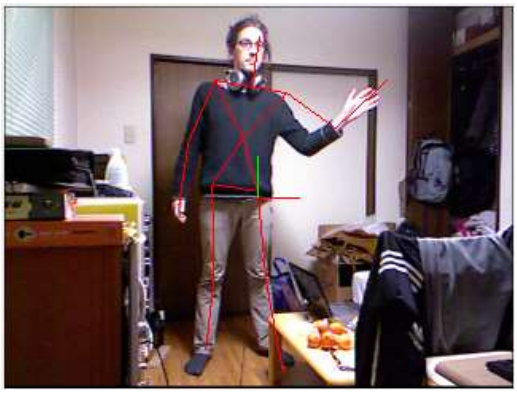


Abbildung 2-3: 3D Skelett Erkennung anhand von Kinect (http://greyviper.com)

Das Projekt wird in verschiedene Aufgaben unterteilt sein. Zuerst ist es notwendig, die offizielle Microsoft Software Development Kit Bibliothek für Kinect in LabVIEW zu integrieren. Da die offizielle Bibliothek nicht rechtzeitig veröffentlicht wurde, musste der

Umweg über ein öffentlich verfügbares Hack[1] genommen werden, der in der Forschungsgemeinde bereits weite Verbreitung findet. Zuerst muss der Zugriff auf die Kamera-Rohdaten gewährleistet sein. Diese Rohdaten bestehen prinzipiell aus 3D Bilddaten des Raums. Darin ist keinerlei Objekt- oder Mustererkennung enthalten. Diese Daten könnten in Zukunft nützlich sein, um eigene Erkennungsalgorithmen zu entwickeln. Industriell könnten sie zum Beispiel in der Fertigungstechnologie benutzt werden, um Lötkomponenten besser zu prüfen.

Der zweite Schritt wird sein, die Skelettinformationen zu ermitteln (z.B. Bild oben) um anhand einer Bewegungsbibliothek Gesten erkennen zu können.

Abbildung 2-4: Kinect als Spielkontrolle(http://www.xbox.com/kinect)

Diese ersten beiden Schritte werden der Datenerfassungsteil des Projektes sein. Nachfolgend besteht die Aufgabe diese Daten zu verwerten. Die Grundidee besteht darin, Hände und Arme zu benutzen um die Kugel des Newton Pendels zu steuern. Dieses motorisierte Pendel existiert schon und ist nicht Teil meines Projekts. Meine Aufgabe wird

[1] Ein Hack ist eine inoffizielle Bibliothek für Microsoft-Produkte, die nicht von Microsoft entwickelt wurde.

sein, die Kommunikationsschnittstelle zwischen dem Rechner, der die Daten von der Kamera erfasst und dem Pendel-Controller, zu entwickeln.

3 Über National Instuments

Abbildung 3-1: Logo von National Instruments (http://www.ni.com/)

3.1 Das Unternehmen

National Instruments (NI) wurde 1976 von drei jungen Herren in der Garage des heutigen Firmenleiters Dr. James Truchard in Austin/Texas gegründet. Als Gründungsidee galt es, verschiedenartige Messgeräte von verschiedensten Herstellern effektiv zu vernetzen und zentral zu steuern um Messaufgaben automatisiert zu lösen. 1977 bringt NI daher auch als erstes Produkt einen GPIB[2] Controller für Mikrocomputer auf den Markt. GPIB ist auch heute noch ein Standbein der Firma.

1986 veröffentlicht NI die erste Version seines heutzutage bekanntesten Produkts: die grafische Programmiersprache LabVIEW, die das Prinzip der virtuellen Instrumente verkörpert. Die Produktpalette erweiterte sich dann 1988 um den ersten Rechner mit interner Messdaten-Erfassungskarte für IBM PCs.

In den Achtzigerjahren wurde NI dann auch schon international: 1987 eröffnet die Niederlassung in Japan, 1988 in Frankreich und in 1989 die Niederlassung in Italien.

Im Jahre 1990 wird NI Schweiz in Ennetbaden bei Zürich gegründet, 1991 folgt NI Germany (NIG) mit der Gründung in München und schließlich 1993 noch NI Österreich.

Diese drei Niederlassungen mit ca. 230 Mitarbeitern sind in der Firmenstruktur zur „Central European Region“ zusammengefasst. Da es in den drei Ländern eine sprachliche Überschneidung gibt, arbeiten diese eng zusammen. Beispielsweise werden Service Fälle

[2] General Purpose Interface Bus auch bekannt al IEEE-488 oder IEC-625-Bus. Es ist die internationale Normbezeichnung für einen externen parallelen Datenbus, der vorrangig zur Verbindung von Messgeräten und Peripheriegeräten wie Plottern und Druckern mit einem Computer eingesetzt wird

auch über Ländergrenzen hinweg durch das Call Center vermittelt, denn NI Österreich hat keine eigene Support Abteilung und wenn bei NI Schweiz alle Leitungen belegt sind, helfen die Applikationsingenieure in Deutschland aus.

1994 veröffentlicht NI ihre erste Webseite unter www.natinst.com als Ergänzung zu den Vertriebswegen und als Wissensdatenbank für die Ingenieure, die mit NI Produkten arbeiten. Unter dem Börsenzeichen „NATI“ ist NI seit 1995 an der NASDAQ Börse zu erwerben und 1997 beginnt NI mit dem Vertrieb von Industrie PCs speziell für die Messtechnik, so genannten PXI Systemen.

Abbildung 3-2: Die "Central European Region" (http://www.ni.com/)

Seit 2002 werden die NI Produkte in Ungarn gefertigt und Reparaturen und Austausch des europäischen Raumes laufen über diesen Standort. Die US Geräte verbleiben auf dem amerikanischen Kontinent und werden in der Firmenzentrale in Texas repariert.
Im Jahre 2009 ist NI mit über 5000 Mitarbeitern weltweit einer der führenden Hersteller von Hard- und Softwareprodukten für PCs und Workstations. Diese werden weltweit von Wissenschaftlern und Ingenieuren zum Aufbau virtueller Mess- und Automatisierungssysteme für Messtechnik, Automatisierung, Motorensteuerung und Bildverarbeitung eingesetzt. NI steht mittlerweile in 47 Ländern mit Niederlassungen für ihre Kunden direkt zur Verfügung. Im Jahr 2008 zählten mehr als 25 000 Firmen in über 90 Ländern der Welt zu den Kunden von National Instruments.

Neben den Hard- und Softwareprodukten bietet NI auch Trainings- und Schulungskurse, sowie Zertifizierungen zur fachgerechten Verwendung der Produkte an.
Bereits zehnmal in Folge wurde National Instruments in den USA als "Great Place to Work" ausgezeichnet. Seit 2004 hat NI Germany sich an der deutschen Version dieses Titels beteiligt und wurde bereits sechsmal mit dem Titel: "Deutschlands bester Arbeitgeber" ausgezeichnet.

NI bietet viele Produkte für einen breiten Markt an, der sich nicht mehr nur auf das ehemalige Kerngeschäft, der Messtechnik, stützt. Der Hauptsitz der CER in München, wo ich mein Praktikum absolviert habe, ist als Vertriebs- und Support-Niederlassung konzipiert. Alle Support-Anfragen und technischen Machbarkeitsstudien aus Deutschland werden hier bearbeitet. Die Entwicklung in Deutschland ist in der National Instruments Engineering GmbH & Co. KG (ehemals GFS) in Aachen angesiedelt, mit dem Schwerpunkt auf das Datenverwaltungs- und Analyse Werkzeug DIAdem.

Als amerikanische Firma ist es für National Instruments im Alltagsgeschäft üblich, viele amerikanische Begriffe zu benutzen. Eine Übersetzung dieser Begriffe wäre nicht sachgemäß und wird daher in diesem Bericht nicht gemacht.

3.2 Produktpalette

3.2.1 Hardwareprodukte

Abbildung 3-3: National instruments Datenerfassungkarten (http://www.ni.com/daq)

Datenerfassung

National Instruments ist einer der Marktführer auf dem Gebiet der PC-basierten Datenerfassung und bietet eine umfassende Familie an Datenerfassungsprodukte für Desktop, portable, industrietaugliche und Embedded-Anwendungen. Anwender haben die Wahl zwischen verschiedenen Bussystemen, u. a. USB, PCI, PCI Express, PXI, PXI Express, Wireless und Ethernet. Die einfach zu handhabende Treibersoftware ist für eine große Auswahl an Betriebssystemen verfügbar, z. B. Windows, Linux, Mac OS X und verschiedene Echtzeitbetriebssysteme.

Modulare Messgeräte

Modulare Messgeräte von NI sind die Grundbausteine für effiziente und vielseitig einsetzbare, automatisierte Prüfsysteme. Anwender können dabei aus Mess-, Signalerzeugungs-, RF-, Stromversorgungs- und Schaltmodulen auswählen, die für die jeweiligen Messaufgaben in Software konfiguriert werden können. Da die Geräte modular aufgebaut sind und ihre Funktionen über Software definiert werden, können sie schnell ausgetauscht und neu konfiguriert werden, so dass sie neuen Anwendungsanforderungen jederzeit gerecht werden. Darüber hinaus ermöglichen die modularen Messgeräte von NI hohe Geschwindigkeiten bei der Anwendungsausführung, da sie sich die Leistung von Industriestandard-PCs und anspruchsvollen Timing- und Synchronisationstechnologien zunutze machen. Die Produktpalette von NI umfasst Module für die unterschiedlichsten Plattformen, u. a. für die Industrie-PC Bauform PXI.

Abbildung 3-4: Ein PXI Chassis von National Instruments (http:// ni.com/)

PXI

PXI ist die offene, PC-gestützte Plattform für die Mess-, Prüf- und Automatisierungstechnik. Als Plattform bietet PXI industrieweit die besten Übertragungsraten und Latenzzeiten. Die PXI-Architektur ist modular aufgebaut und umfasst I/O-Module für hochauflösende Messungen, die von der Erfassung von Gleichstrom bis hin zur Erfassung von RF-Signalen mit 6 GHz reichen. Über 1200 PXI-Produkte von mehr als 70 Herstellern sind z. Z. auf dem Markt erhältlich. National Instruments stellt für diesen Bus alle benötigten Komponenten her, von Gehäusen, Controllern, Einsteckkarten bis hin zu MXI-Bridges die den PC eigenen PCI Bus in das PXI Chassis erweitert.

Messen, Steuern und Regeln in Echtzeit

Die Echtzeitprodukte von National Instruments verbinden die grafische Entwicklungsumgebung LabVIEW mit der LabVIEW-Embedded-Technologie, um die deterministische Leistungsfähigkeit dedizierter Echtzeit- und FPGA-Zielgeräte zu erreichen. Damit lassen sich Anwendungen erstellen, die vorhersagbar reagieren und zuverlässig sowie autark arbeiten.

Verteilte I/O

Die Produkte für verteilte I/O von National Instruments wurden für Remote-Messsysteme sowie industrielle Steuer-, Regel- und Datenprotokollierungsanwendungen konzipiert. Sie sind widerstandsfähig und für den Einsatz in Industrieumgebungen zertifiziert, so dass sie für Messungen mit verschiedenen Sensoren und Aktoren geeignet sind, unabhängig von Umgebung oder Entfernung. NI bietet deterministische Ethernet-I/O und Ethernet basierte I/O-Erweiterungen sowie intelligente Steuer-, Regel- und Datenerfassungssysteme, die als Stand-Alone-Systeme eingesetzt werden können. Zum einfachen Anschluss an bereits vorhandene Geräte nutzen verteilte I/O Geräte von NI übliche industrielle Protokolle. Mithilfe der verteilten I/O können selbst in elektromagnetisch empfindlichen und rauen Umgebungen Daten einfach und zuverlässig erfasst werden.

Programmable Automation Controller (PAC)

Um die Vorteile eines Industrie-PCs und einer SPS zu kombinieren, bietet National Instruments so genannte Programmable Automation Controller (PAC) an. PAC-Systeme verbinden Robustheit mit der Funktionalität von PCs in einer offenen, flexiblen Softwarearchitektur. Mit diesen Geräten lassen sich fortschrittliche Systeme entwerfen, die nicht nur Softwarefunktionen wie z. B. eine erweiterte Regelung, Kommunikation, Datenprotokollierung und Signalverarbeitung, sondern auch robuste Hardware umfassen, die Logik, Motorsteuerung, Prozesssteuerung und Bildverarbeitung ausführt.

Gerätesteuerung

National Instruments bietet eine große Auswahl an Hard- und Softwarewerkzeuge für Gerätesteuerungssysteme. Die Produktpalette umfasst Hardware für GPIB, VXI, RS-232/485, USB, Ethernet, WLAN, sowie Softwarewerkzeuge wie NI LabVIEW und Gerätetreiber.

Industrielle Bildverarbeitung und -visualisierung

National Instruments bietet auch Hardware und Softwarepakete für industrielle und wissenschaftliche Bildverarbeitung. Werden dabei die Kameras von Drittherstellern verwendet, so erfolgt die Bildverarbeitung nebst Digitalisierung (falls noch nötig) auf entweder verschiedenen Hardware Plattformen von National Instruments (wie z.B. PAC oder PXI) oder auch auf normalen Windows Rechnern. Die Softwarepakete bieten dabei verschiedenste bildverarbeitende Algorithmen für die Bildverarbeitung in LabVIEW oder auch anderen Programmiersprachen an. Einfache Assistenten helfen dabei auch schwierige Algorithmen in den Griff zu bekommen. Für kompakte und autonome Bildverarbeitung bietet NI auch die so genannte SmartCamera an. Dies ist ein Gesamtsystem aus Kamera, CPU, Festspeicher und I/O Modulen zur Anbringung auch auf engstem Raume.

3.2.2 Softwareprodukte (Auszugsweise)

Abbildung 3-5: Logo von LabVIEW (http://www.ni.com/)

LabVIEW

Seit über 20 Jahren revolutioniert die grafische Entwicklung mit NI LabVIEW die Arbeitsweise von Ingenieuren und Wissenschaftlern bei der Erstellung skalierbarer Mess-, Prüf-, Steuer- und Regelanwendungen. LabVIEW ist eine grafische Programmiersprache, die es dem Anwender ermöglicht obig genannte Hardware mit selbst geschriebenen Programmen zu verwenden. Dabei spielt das verwendete Betriebssystem vorrangig keine Rolle, ein Echtzeitsystem kann genauso einfach wie eine Windows Applikation geschrieben werden. Die Integration von Hardware in Programmarchitekturen wird durch vorgefertigte Funktionsbibliotheken, Beispielen und Wizards unterstützt.

DIAdem

DIAdem ist die Standardsoftware von National Instruments für die interaktive und automatisierte Datenverwaltung, Datenanalyse und Berichterstellung. Zielsetzung hier ist es, leistungsstark und zeitsparend große Mengen an Daten zu analysieren, bearbeiten und zu präsentieren. Dank der Unterstützung unterschiedlichster Dateiformate und Datenbanken ermöglicht DIAdem die Auswertung von Messdaten und Simulationsdaten nahezu beliebiger Herkunft. Der integrierte DataFinder bietet zudem eine fertige Lösung für die Datenrecherche in umfangreichen Dateibeständen. Zusätzliche Module für die Datenerfassung und Schnittstellen zu anderer Software ermöglichen Komplettlösungen, angefangen bei der Datenerfassung bis hin zur Datenauswertung.

LabWindows/CVI

Bei LabWindows/CVI von National Instruments handelt es sich um eine bewährte, auf ANSI C basierende Entwicklungsumgebung, die Anwendern ein umfassendes Paket an Programmierwerkzeugen für die Entwicklung von Prüf-, Steuer- und Regelanwendungen zur Verfügung stellt. LabWindows/CVI kombiniert die Langlebigkeit und Wiederverwendbarkeit von ANSI C mit Funktionen, die speziell für die Gerätesteuerung, Datenerfassung, Analyse und Entwicklung von Benutzeroberflächen konzipiert sind.

Multisim

Die National Instruments Electronics Workbench Group (ehemals Electronics Workbench) stellt Entwicklungsingenieuren, Lehrenden und Studierenden eine leistungsstarke und bedienerfreundliche Software für Schaltungserfassung, interaktive SPICE-Simulation, Leiterplattenlayout und Designvalidierung zur Verfügung.

TestStand

NI TestStand von National Instruments ist eine sofort ablauffähige Prüfumgebung, welche die zügige Entwicklung automatisierter Prüf- und Validierungssysteme ermöglicht. NI TestStand kann eingesetzt werden, um Prüfsequenzen, die in beliebigen Programmiersprachen geschrieben wurden, zu entwickeln, zu verwalten und auszuführen. Die Integration in unternehmensweite Systeme ist mit NI TestStand ebenfalls möglich.

4 Das Gerät : Microsoft Kinect.

4.1 Überblick

Abbildung 4-1: Eine XBOX 360 (Mitte) mit dem Kinect Spielcontroller (links) und einem traditionellen Spielcontroller (rechts) (http://reviews.cnet.com/)

Ehe man mit einem Projekt anfängt, sollte man die Technologie vorstellen. Kinect ist ein Spielcontroller für die XBOX 360 Spielkonsole von Microsoft. Kinect ist eine 3D Kamera, die Menschen erkennen kann und ihre Bewegungen messen kann. Das Gerät wurde im November 2010 veröffentlicht, 5 Jahre nach der XBOX 360. Es ist eine Antwort auf die Monopolstellung von Nintendo im „Motion Gaming“ Bereich[3].

Da dieses Gerät für die Unterhaltungsindustrie produziert wird, ist die Produktionsanzahl höher beziehungsweise der Preis günstiger als bei ähnlichen Geräten die für die technische-Industrie. Dieser Preisunterschied ist so, dass Kinect ca. zehn- bis hundert-mal günstiger wie eine Industrie 3D Kamera ist. Da das Gerät auch von Leuten, die nicht in die Industrie arbeiten, bekannt ist, kein Komma haben die Projekte, die dieses Gerät benutzen, eine bessere Sichtbarkeit.

[3] „Motion Gaming“ ist ein Bereich der Videospiel-Industrie. Spiele werden nicht mehr mit Spielcontroller kontrolliert, sondern mit Bewegungen der Spieler. Die Nintendo Wii war die erste erfolgreiche Konsole, die diese Technologie benutzt hat. Seinen Konkurrenten Sony und Microsoft haben dann auch „Motion Gaming“ orientierte Gerät veröffentlicht um das Nintendo Monopol zu zerschlagen.

4.2 Die Technologie.

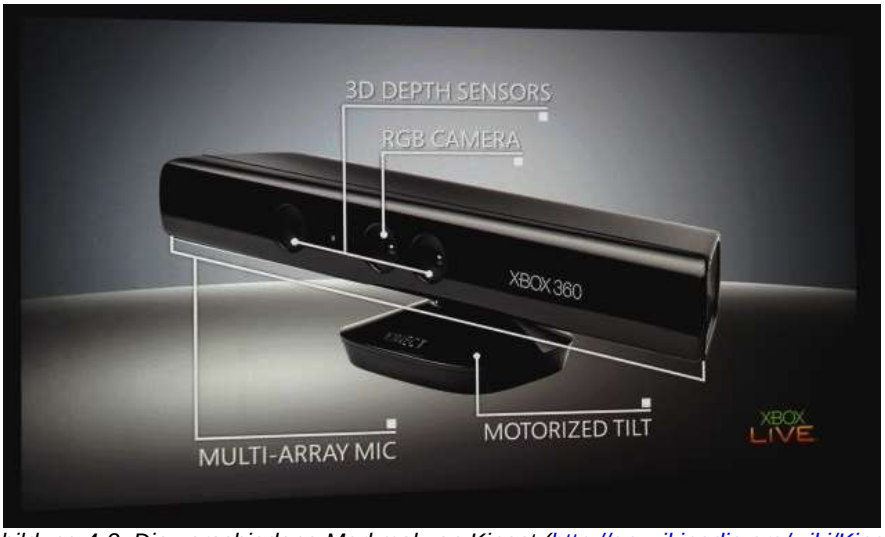


Abbildung 4-2: Die verschiedene Merkmal von Kinect (http://en.wikipedia.org/wiki/Kinect)

Das Gerät verwendet ein paar Sensoren: Abstandsensor, RGB Kamera und Mikrophone (cf. Abbildung 4-2). Wichtig für mein Projekt ist der 3D Abstand-Sensor („3D Depth Sensor“ im Bild). Diese arbeitet nach dem Streifenprojektionsprinzip mit einem Infrarotgitternetz. Die Idee ist, dass ein Infrarot Laser ein vordefiniertes Muster von Punkten vorgibt (cf. Abbildung 4-3). Der Reflektionswinkel dieses Lasers wird gemessen und damit der Abstand berechnet.

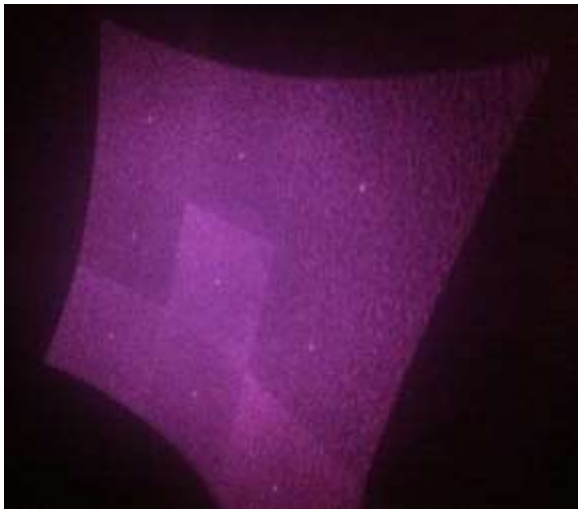


Abbildung 4-3: Infrarotmuster von das Kinect Laser (http://www.anandtech.com/)

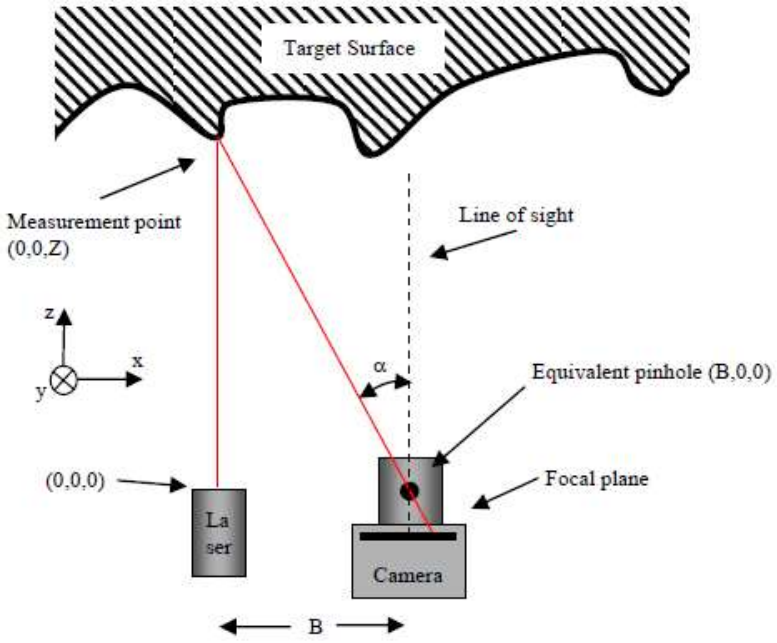


Abbildung 4-4: Abstand-Messung in Idealfall (http://trs-new.jpl.nasa.gov/)

In Abbildung 4-4 kann man das Prinzip der Abstandmessung erkennen. Im Abstand kann man sagen, dass gilt:

$$Z = B*\cot an(\alpha).$$

Das ist nur im Idealfall so, wo der Laser parallel mit der z Achse ist. In der Realität ist die Berechnung komplizierter aber auf dem gleichen Prinzip basiert. Um weitere Informationen zu diesem Thema zu erhalten, gibt es PDF-Dokument auf der NASA Webseite. Der Link kann in [4] gefunden werden.

4.3 Applikationen

Wegen seines niedrigen Preises und des „Sichtbarkeitsbonus“ des Gerätes haben sich ein paar industrieller und medizinischer Projekte haben wir uns für das Kinect als Sensor entschieden. ABB hat zum Beispiel angefangen mit Kinect (cf. Literaturverzeichnis [5]) auf industrieller Roboterkontrolle zu arbeiten. EVT in Karlsruhe arbeitet auch an die Kompatibilität von Kinect mit seinem Tool „Eyescan 3D“ für die Palettierung und die Depalletierung in Bereich Robotik (cf. Literaturverzeichnis [6]). Das Gerät wird auch

Medizinapplikation haben für Taubstummheit oder Hellersche Demenz (Literaturverzeichnis [7] und [8]).

5 Die Treiber

Als ich mein Projekt angefangen habe, sollte ich mich für eines der Hacks von Kinect entscheiden. Bei der Suche sind mir zwei Hack bekannt geworden: OpenKinect und OpenNI. Die folgenden Seiten werden diese zwei Hack vorstellen.

5.1 OpenKinect

5.1.1 Überblick

Abbildung 5-1: Logo von Openkinect (http://openkinect.org/wiki/Main_Page)

OpenKinect ist eine Open Source Gesellschaft, die von Joshua Blake ausgegründet wurde, nachdem er dem Codierungscode von Kinect in drei Stunden gehackt hatte. Die Gesellschaft hat jetzt mehr als 2000 Mitglieder, die von verschiedenen anderen Hackgesellschaften kamen. Sie konzentrieren sich mehr auf der Datenerfassung in dem Raum als auf der Bewegungsanerkennung von den Menschen. Man kann viele Infos auf ihnen finden: http://openkinect.org/wiki/Main_Page.

5.1.2 Installation

Die OpenKinect Treiber sind ziemlich leicht zu installieren. Die Treiber sind nicht von Windows anerkannt. Man muss die Treibers selbst runterladen: http://www.blokmodular.com/dev/Kinect-v16-withsource.zip (Version 16 die Treibers). Anschließend muss man dann manuell die Treiber installieren: Unten „Systemsteuerung => Geräte Manager“ sieht man die unbekannten Geräte. Man muss den Pfad zum Treiber selbst eingeben: <Unzipped Ordner>\Drivers. Nach der Installation sieht Man zwei neue unbekannte Geräte: Camera und Audio. Der gleiche Treiberpfad soll für beide manuell angegeben sein.

Man kann dann direkt probieren, ob es funktioniert mit der mitgelieferten Demo-Applikation: <Unzipped Ordner>\Kinect-Demo\Kinect-Demo.exe . Wenn es nicht funktioniert, bedeutet es, dass die Grafikkarte nicht genug Grafikspeicher hat. Es kann zwei Gründe haben: entweder ist es ein „On-board graphic chip“ oder die Graphikkarte ist zu alt.

5.1.3 Verwendung

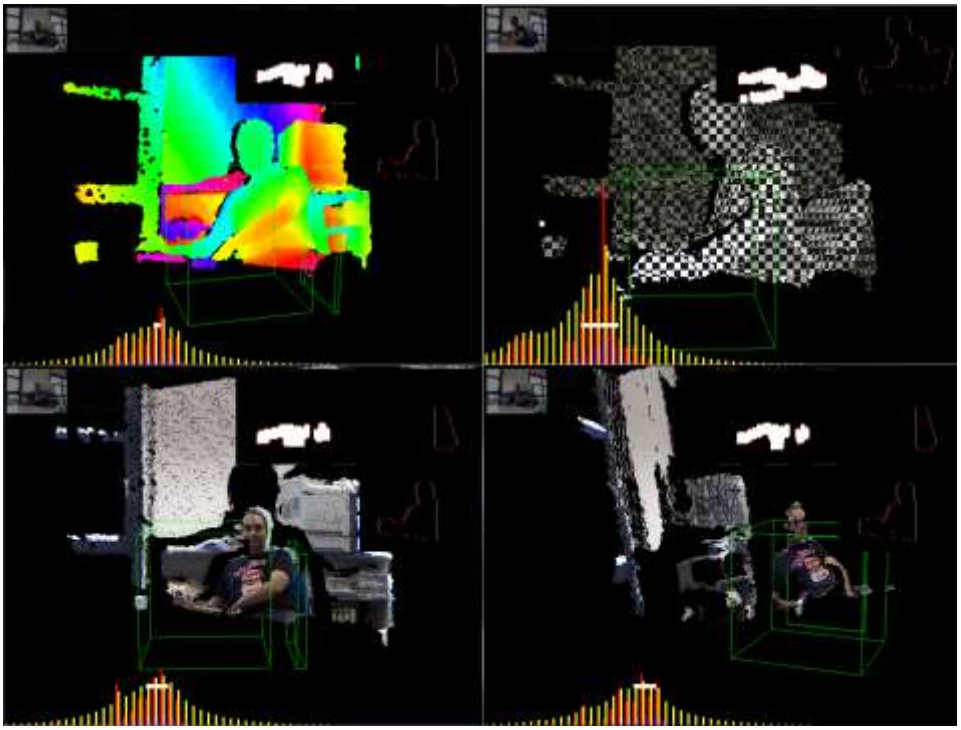


Abbildung 5-2: Verschiedene Anzeigemodi mit die KinectDemo

Wie schon erwähnt, haben die Kinecttreiber Kompatibilitätsprobleme mit den On-Board graphischen Chip. Dies war auch bei mir der Fall, da meine Motherboard einen integrierten Chip verwendet. Um es zum Laufen zu bringen, sollte ich eine graphische Karte einbauen. Nach Einbau einer NVIDIA GeForce 6800 hat das Beispielprogramm funktioniert. Abbildung 5-2 zeigt verschiedene Anzeigmodus von die OpenKinect mitgelieferte Demo-Applikation.

Die OpenKinect Treiber haben soweit für meine Aufgabe Nach- und Vorteile. Sie sind sehr einfach zu installieren und haben nicht viele Funktionen. Sie sind auch deswegen weniger

flexibel. Der Focus auf 3D ist auch ein Nachteil, weil meine Projektaufgabe mehr mit Bewegungserfassung zu tun hat. Deswegen habe ich versucht mit anderen Treibern zu arbeiten.

5.2 OpenNI

5.2.1 Überblick

Abbildung 5-3: Logo von OpenNI (http://www.openni.org/)

Open NI eine industrielle und nicht gewinnorientierte Organisation, die Naturell Interaktion Gerät fördert. Dieses Naturell Interaktion Gerät – auch Naturell Schnittstelle genannt - sind Geräte, die Bewegungen und Klang einfangen, um eine Naturelle Interaktion mit Computer zu erlauben. Diese Organisation wurde von vier Firmen ausgegründet: ASUS, Willow Garage, Side Kick und Prime Sense. Prime Sense ist auch die Firma, welche die 3D Messung für Kinect entwickelte. Deswegen haben Prime Sense Kinect auch OpenNI kompatibel gemacht.

5.2.2 Installation

Die Installation von dieser Umgebung ist komplizierter als die von Open Kinect. Zuerst muss man andere Kinect Treiber deinstallieren und folgende installieren:

1) https://github.com/avin2/SensorKinect. Der Pfad zum Treiber ist <unzipped Ordner>\Platform\Win32\Driver\. Nachher muss man OpenNI und NITE installieren. Hier sind die Links zum Runterladen.
2) OpenNI:http://www.openni.org/downloadfiles/openni-binaries/latest-unstable/108-openni-unstable-build-for-windows-x86-32-bit-v1-1-0/download
3) NITE:http://www.openni.org/downloadfiles/openni-compliant-middleware-binaries/latest-unstable/115-primesense-nite-unstable-build-for-windows-x86-32-bit-v1-3-1/download

Nach der Installation von diesen zwei Programmen muss man wieder in den Treiberordner gehen und <unzipped Ordner>\Bin\ SensorKinect-Win32-5.0.0.exe ausführen. Diese Installer funktioniert nur, wenn eine aktuelle Version von NITE installiert ist. Sein Inhalt ist die Komponente, die die Anerkennung von Kinect mit OpenNI erlaubt.

Mit den Versionen von NITE ältere als 1.4 sind die Treiber nicht benutzbar. Man muss noch die Konfiguration von NITE ein ändern. Es gilt drei xml Files zu modifizieren:

- Sample-Scene.xml,
- Sample-Tracking.xml und
- Sample-user.xml.

Diese Files kann man unten C:\Program Files\PrimeSense\NITE\Data finden. Die Modifikation sind in Anhang 15.1, 15.2 und 15.3 verfügbar.

5.2.3 Verwendung

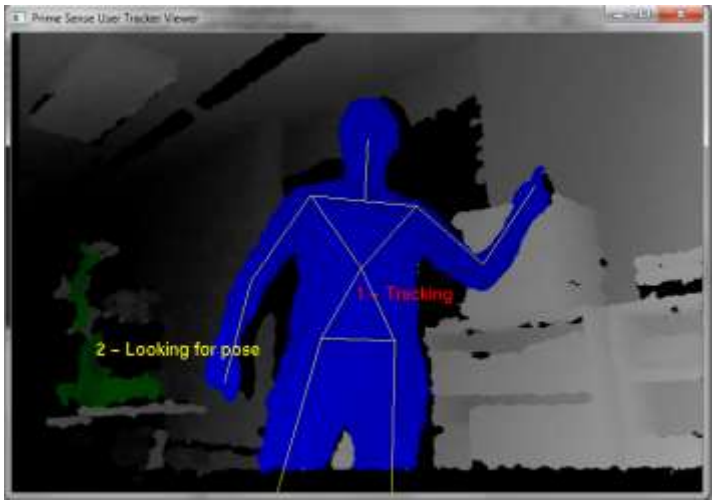


Abbildung 5-4: Ein Anzeigemodus von OpenNI mit Skelett-Erkennung

Nach der Installation kann man testen, ob die Treiber funktionieren. OpenNI hat dafür vorbereite und vorkompilierte Beispiele. Man kann dieses Beispiel üblicherweise unter C:\ProgramFiles\OpenNI\Samples\Bin\Release finden. Das Beispiel, um die Skelett-Erkennung zu probieren, heißt NIUserTracker.exe und sieht wie in Abbildung 5-4 aus. Um die Kalibration des Skeletts zu realisieren, muss man ein U mit seinem Arm machen – wie es auf dem Abbildung 5-5 gezeigt ist.

Abbildung 5-5: Arm Haltung um die Skelett-Erkennung anzufangen

Zusammenfassung/Bewertung

Die OpenNI Treiber scheinen besser für die Aufgabe zu sein, da sie flexibler und „bewegungsorientiert“ sind. Jedoch sind sie auch komplizierter zu benutzen, weil die Treiber, kein Komma der NITE Module und die OpenNI-Oberfläche anders aussehen. Meine nächste Aufgabe besteht darin, DLL auf diese Treibe zu bauen um Kinect in LabVIEW zu benutzen.

6 LabVIEW und DLLs

6.1 Implementierung von DLLs in dem Projekt

6.1.1 Überblick

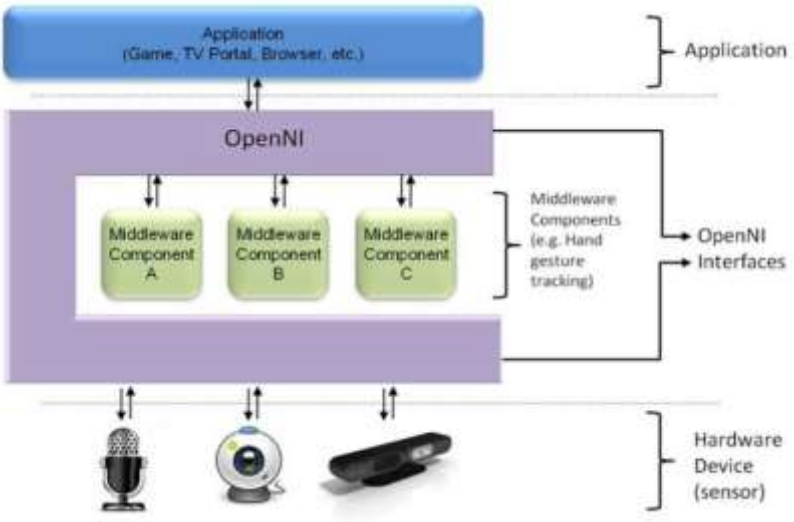


Abbildung 6-1: Architektur von OpenNI (http://www.openni.org/)

Am Anfang haben wir gedacht, dass wir von LabVIEW direkt auf die DLLs-Treiber von Kinekt zugreifen können um die Daten zu nutzen. Leider ist es nicht so einfach. Zwischen den Geräten und den verwertbaren Daten gibt es die Treibers aber auch die OpenNI Oberfläche. Diese Oberfläche integriert auch „Middleware Components“ von anderen Erstellern (Prime Sense in unserem Fall). Die gesamte Architektur der OpenNI Framework ist in Abbildung 6-1 gezeigt. Der einfachste Weg verwertbare Daten zu erhalten, ist sich nur auf den obersten Schnitt von OpenNI zu konzentrieren. Leider ist dieser Schnitt nur in C++ und C# programmierbar. Die Lösung würde die Programmierung ein Wrapper sein.

6.1.2 Ein Wrapper für LabVIEW

Wikipedia definiert einen Wrapper wie folgend: *„Als Wrapper bezeichnet man in der Informationstechnik ein Stück Software, welches ein anderes Stück Software umgibt. Dies kann sich sowohl auf ganze Programme, als auch nur auf einzelne Programmteile bis Klassen beziehen. […] Beispielsweise sind Wrapper behilflich, wenn Programmteile einer anderen Programmiersprache verwendet werden sollen, oder auch um den Zugriff auf bestimmte Programmteile einzuschränken (da das Programm so nur innerhalb des Wrapper läuft).“* In meinen Fall brauche ich ein Wrapper der die C++ Funktion zum LabVIEW gibt. Um es umzusetzen würde ich DLLs erzeugen als DLLs die einzige Art von C++ Code dass, LabVIEW verstehen kann. Eine Schwierigkeit wird, dass alle OpenNI verwirklicht, Methode von einem Objekt sind. Ein C++ Objekt kann nicht in LabVIEW gelesen oder übertragen sein. Eine potentielle Lösung würde der pointer zu diesem Objekt weitergeben zu jede Aufruf die DLL.

6.2 Realisierung in LabVIEW

6.2.1 Überblick

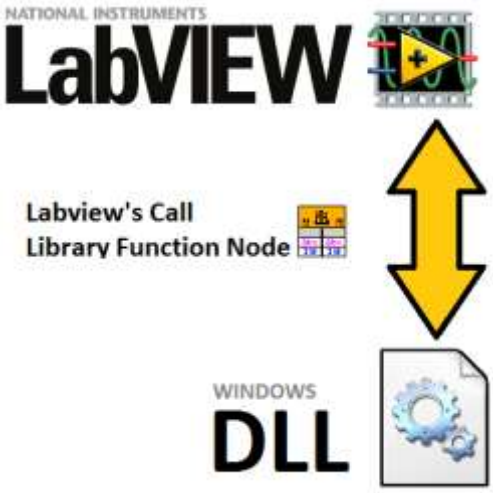


Abbildung 6-2: Dialog zwischen LabVIEW und eine C++ DLL mit die Call Library Function Node

LabVIEW erlaubt C++ DLL einzubinden durch die so genannte „Call Library Function Node“. Dieser Funktionsblock soll wie in C++ parametriert sein: Zuerst den Name die DLL auswählen, dann dem Rückdatentyp definieren und zuletzt die Argumente.

Meine Idee war, zuerst die Einbindung einiger Beispielprogramme zu testen und dann zu versuchen den Wrapper zu programmieren. Ich habe versucht ein DLL zu programmieren, das ein paar einfache Funktion ausführt (addieren, dividieren). Der Plan war es in Visual Studio in C++ zu programmieren, als DLL zu compilieren und in LabVIEW zu benutzen.

6.2.2 Programmierung einer DLL in C++

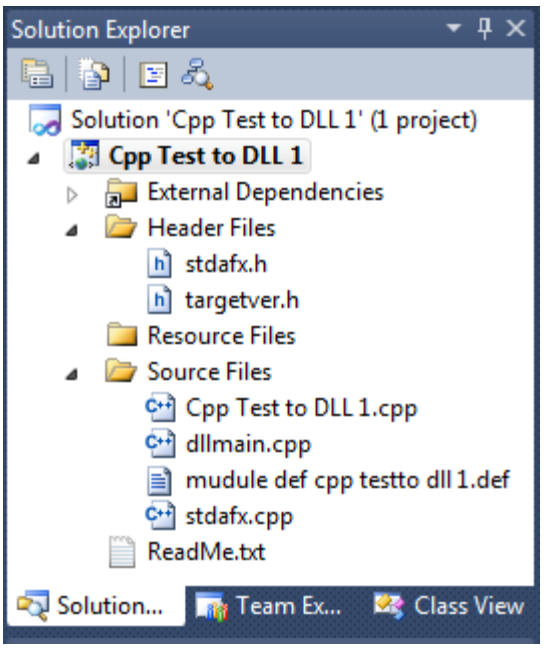


Abbildung 6-3: Architektur meines Projekt

Die Entwicklung einer DLL in C++ ist nicht viel anders als in einem normalen Programm. Der Unterschied ist, dass die Funktionen nicht im Mainprogramm aufgerufen und benutzt sind. Zwei Headers sollten immer da sein: `stdafx.h` und `targetver.h` (cf. Abbildung 6-3). Der erste listet alle andere Headers, die im Projekt benutzt werden und der zweite der Code in falls die Einsetzungsumgebung nicht die gleiche als die Entwicklungsumgebung ist. In `stdafx.cpp` sollen wieder die Headers gelistet sein.

In `dllmain.cpp` soll das Standard DLL Mainprogramm sein sowie die Liste alle dem Funktion der DLL. Man kann den Code dieser Files im Anhang 15.4 finden. Meine drei Funktionen sind in den Zeilen 20, 21 und 22 gelistet. „extern "C“ und „__stdcall“ sind da,

dass der Compiler diese Funktionen sichtbar macht und aufrufbar macht in der DLL. Ohne was? würden diese Funktionen unbenutzbar, außer von der DLL selbst.

`Cpp Test to DLL 1.cpp` ist das File, in dem die Funktionen definiert sind. Man kann den Code dieser Files im Anhang 15.5 finden. In der ersten Funktion Addiertfuenf (Linie 6-10), wollte ich einen ganz einfachen Parameter benutzen (einen Integer) und 5 als Integer benutzen. Der Grund von der zweiten Funktion Addieren (Linie 13-17) ist, dass ich mit kompliziertem Datentypen (float) und mehr Parametern (2) arbeiten wollte. Das dritte Beispiel Durchzwei (Linie 20-26) sollte ein Versuch mit Parameter als Referenz sein. Die Funktion nimmt zwei Zahlen, dividiert beide durch zwei und addiert das Ergebnis. Um die dividierte Zahl zu benutzen, soll man diese als Referenz definieren. Deswegen wird das Zeichnen „&“ benutzt in Zeile 20. Referenz bedeutet, dass sie variabel sind nicht nur lokal benutzt werden. Die Speicheradresse ist dem Argument gegeben, sodass die Operationen auf die Variablen einen Einfluss an die anderen Teile des Programmes haben.

Zuletzt haben wir `module def cpp testto dll 1.def`. Es ist ein .def File, das die Funktion der DLL listet, sodass die Aufrufer der DLL die Funktionen sehen kann. Sein Inhalt ist sehr klein man kann sie aber ohne nicht die DLL in LabVIEW benutzen.

```
1  EXPORTS
2     Addiertfuenf    @1
3     Addieren        @2
4     Durchzwei       @3
```

Abbildung 6-4: module def cpp testto dll 1.def

Nach der Kompilierung konnte ich die DLL in LabVIEW einbinden.

6.2.3 Einbindung meiner DLL in LabVIEW

Ich habe ein ganz einfaches Programm programmiert, um meine DLL zu benutzen. Es sieht aus wie drei „Tab Controls“, die alle eine Funktion ausführen (cf. Anhang 15.6). Den Code kann man in Anhang 15.7 sehen. Es ist nichts anders als eine while Schleife, die an jeder Iteration eine von den drei Funktionen die DLL ausführt. Der interessante Teil sind die Eigenschaften, die Call Library Function Node. Jede benutzt die „stdcall (WINAPI) convention“, weil ich es so programmiert habe (deswegen soll ich __stdcall schreiben vor alle Funktion). Man merkt, dass bei *Addiertfuenf* und *Addieren* ich den Wert meines Argumentes benutze, aber bei *Durch2* ein „Pointer to value“. Mit diesem Pointer kann ich die Adresse meiner Zahl geben und später im Programm diese Adresse lesen um die modifizierte Zahl zu lesen. So funktioniert mein Beispielprogram einwandfrei.

7 Nutzung eines C++ Objekts in einer DLL durch LabVIEW

7.1 Überblick

Im vorherigen Kapitel habe ich eine selbst gebaute DLL in LabVIEW eingebunden. In diesem Kapitel werde ich eine DLL, die objektorientierte Programmierung enthält, in LabVIEW versuchen zu benutzen. Wikipedia definiert Objektorientierung wie folgend: *„Unter Objektorientierung versteht man eine Sichtweise auf komplexe Systeme, bei der ein System durch das Zusammenspiel kooperierender Objekte beschrieben wird. Der Begriff Objekt ist dabei unscharf gefasst; wichtig an einem Objekt ist nur, dass ihm bestimmte Attribute (Eigenschaften) und Methoden zugeordnet sind und dass es in der Lage ist, von anderen Objekten Nachrichten zu empfangen beziehungsweise an diese zu senden. Dabei muss ein Objekt nicht gegenständlich sein. Entscheidend ist, dass bei dem jeweiligen Objektbegriff eine sinnvolle und allgemein übliche Zuordnung möglich ist. Ergänzt wird dies durch das Konzept der Klasse, in der Objekte aufgrund ähnlicher Eigenschaften zusammengefasst werden. Ein Objekt wird im Programmcode als Instanz beziehungsweise Inkarnation einer Klasse definiert. Objektorientierung wird hauptsächlich im Rahmen der objektorientierten Programmierung verwendet, um die Komplexität der entstehenden Programme zu verringern.“*

Es ist wichtig, in unserem Fall mit C++ Objekten in LabVIEW zu arbeiten, weil die ganze Architektur von OpenNI Objekt benutzt wird. Zuerst wird ein Objekt erstellt, das mit einfachen Methoden verwirklicht wird. Diese Methode werde ich mit DLL Funktionen aufrufen und diese DLL in LabVIEW einbinden. Das Ziel ist es, den Pointer auf das Objekt zwischen den LabView-Funktionen weiterzuleiten (cf. Abbildung 7-1). Wenn es funktioniert, wird die gleiche Architektur mit OpenNI Funktionen zum Laufen gebracht.

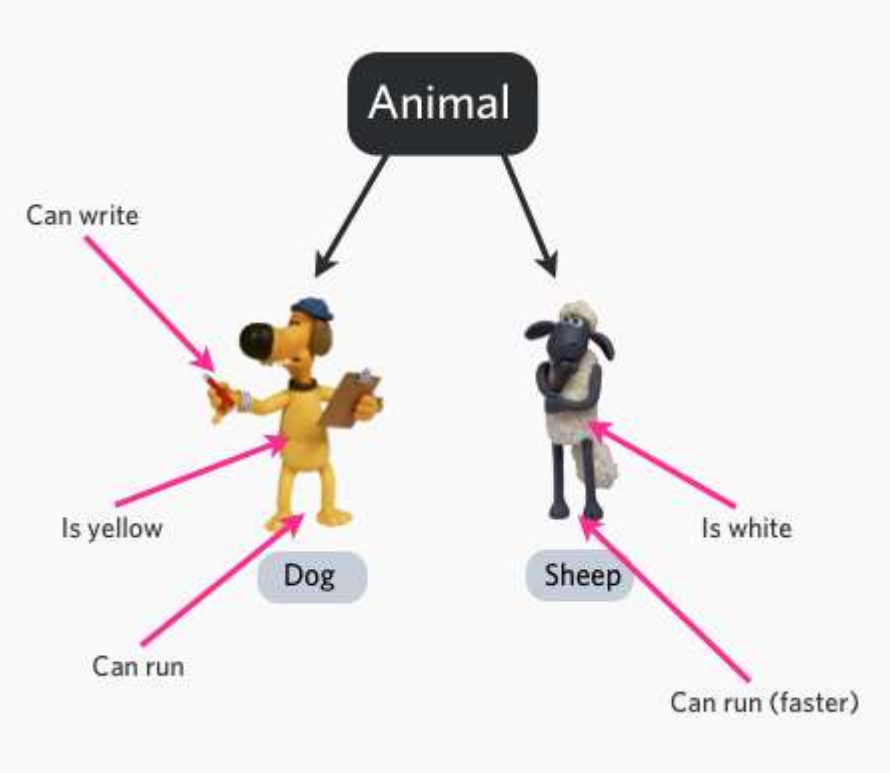


Abbildung 7-1:Ein Beispiel, was ein Objekt sein könnte: Man kann einen Hund und ein Schaf erstellen, die beide der Klasse „Tiere“ gehören. Diese werden beide mit verschiedenen Attributen definiert (Farbe, Fähigkeiten) und man kann später Methoden von der Tier-Klasse aufrufen (Rennen zum Beispiel, oder Schreiben für den Hund) (http://www.codercaste.com/)

7.2 Einfache Methode zur Einbindung einer DLL in LabVIEW

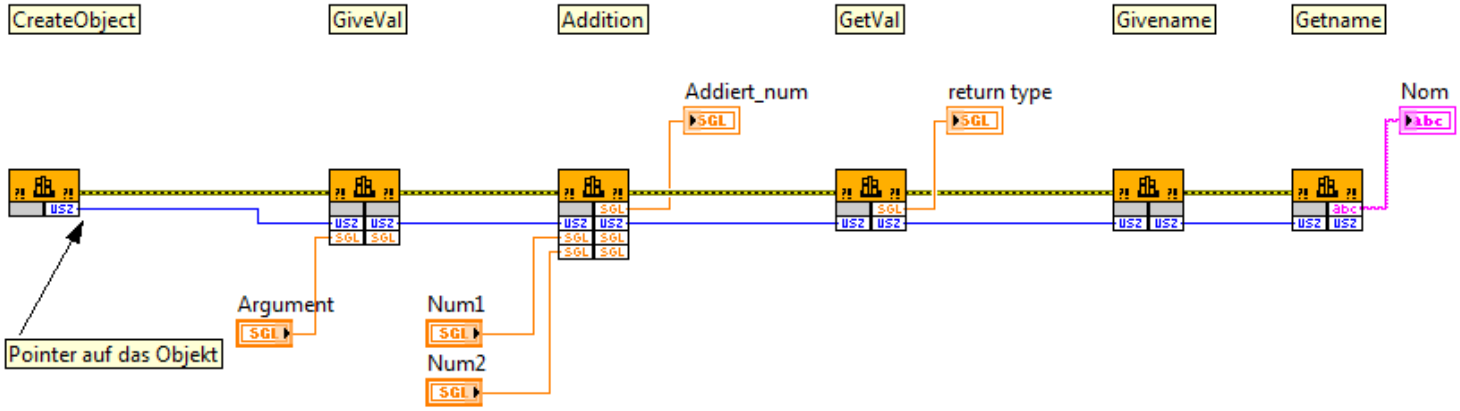


Abbildung 7-2: Ausblick auf mein LabVIEW Testprogram

Ich habe die gleiche Struktur wie beim Vorgängerbeispiel genommen (DLL ohne Objekt). Die Änderungen des Codes können als Anhang 15.8 gefunden sein. Ein Screenshot sieht man in Abbildung 7-2. Die Hauptidee ist, dass ich ein Objekt in eine Funktion einbaue und dann seine Referenz zurückgebe. Mit dieser Referenz (ein Pointer auf das Objekt) rufe ich andere Funktionen auf, welche die Methode des Objekts aufruft. Wichtig war für den Test, dass ich auch in die Attribute schreibe um zu sicher zu stellen, dass ich immer an das gleiche Objekt zugreife. Deswegen sind die Funktionen GiveVal(); GetVal(); Givename(); und Getname(); da. In diese Funktionen schreibe ich in die privaten Attribute mein Objekt. Addition addiert wie immer zwei Zahlen. In Anhang 15.8.2 merkt man, dass außer CreateObject(); alle Funktionen einen Pointer auf dem Dummi-Objekt als Argument nehmen (Zeile 14, 19, 24, 19 und 34). Deswegen werde ich einen Pfeil statt einen Punkt benutzen (Zeile 16, 21, 26, 31 und 36), um meine Methode anzuwenden (der Punkt von die anderen Programme ruft die Methode aus einem Objekt und der Pfeil ruft die Methode aus einem Pointer der auf ein Objekt zeigt). Dieser Test war erfolgreich und ich konnte von LabVIEW auf die privaten Attribute sowie auf die Methoden meines C++ Objekt zugreifen. Im nächsten Schritt wird auf die Daten der Kinect Kamera zugegriffen.

7.3 Einbindung von OpenNI Programmen in LabVIEW durch C++ DLL

Vor allem sollte ich ein Beispielprogramm in C++ schreiben. Dieses Programm ist das Fundament meiner ersten DLL. Es ist als Anhang 15.9 abgebildet. Dieses Programm misst die Tiefe des Mittelpixels der 3D Kamera. Es funktioniert so einwandfrei. In Zeile 7 dieses Programms kann man „using namespace xn;“ lesen. Es ist da, weil die meisten Objekte von OpenNI in dem Namensraum „xn“ definiert sind. Wikipedia definiert es wie folgend: *„Der Namensraum (englisch: „namespace“) ist ein Begriff aus der Programmierung. Dabei werden – vor allem bei der objektorientierten Programmierung – die Namen für Objekte in einer Art Baumstruktur angeordnet und über entsprechende Pfadnamen eindeutig angesprochen.“.* Man kann es mit einem Ordner, der Objekt enthält, vergleichen.

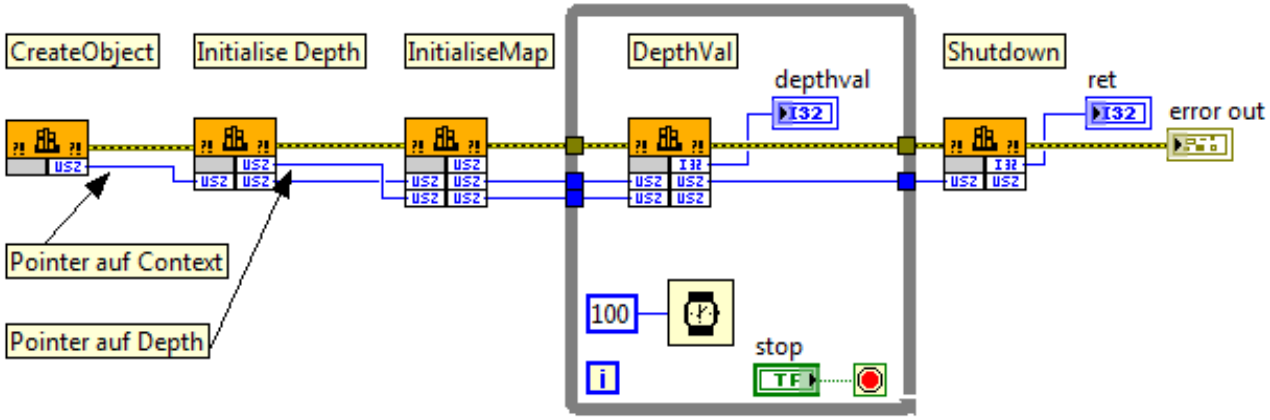


Abbildung 7-3: Das LabVIEW Programme um die Tiefe des 3D Kamera mit eine DLL zu lesen

Nach dem Beispiel mit C++ habe ich versucht, das gleiche Programm in LabVIEW zu implementieren. Ich habe zuerst das ganzes Programm (ohne die while Schleife) in eine große Funktion gepackt und diese Funktion in eine DLL kompiliert. Als ich diese Funktion in LabVIEW aufgerufen habe, könnte ich die Tiefe lesen. Es hat aber der Nachteil, dass es nur einmal aufgerufen werden konnte und dass es sehr langsam war (die Initialisierung dauert ein paar Sekunden). Um es benutzbar zu machen, sollte ich mein Programm in verschiedene Funktionen teilen (wie im Anhang 15.10) und dann diese Funktionen in LabVIEW aufrufen (wie bei die Abbildung 7-3).

Obwohl es die gleiche Architektur wie in meinem Beispiel annimmt, hat dieses Programm nie funktioniert. Schuld ist die komplexe Architektur von der OpenNI-Bibliothek. Die Objekte sind in verschiedenem Namensraum deklariert und rufen einander auf. Im Anhang 15.10 findet sich der Namensraum. Man kann dann sehen, dass zum Beispiel `Context` und `DepthGenerator` im Namensraum „xn“ deklariert sind aber `XnMapOutputMode` nicht. Diese Namensräume sind auch Eigenschaften von C++ und werden von C nicht verstanden. Verschiedene Namespace bedeutet eine komplizierte Speicherarchitektur, die bei unserem C DLL und LabView nicht gut verstanden wird. Die weitergegebenen Pointer auf das Objekt sind oft falsch oder vom Programm schlecht interpretiert. Im Internet findet man viele Beschreibungen dieses Problems. Es betrifft die mit „extern "C"“ deklarierten Funktionen, die Namensraum enthalten. Ich habe versucht, den Namensraum „xn“ der OpenNI Bibliothek zu löschen. Ohne Namensraum könnte ich das Programm nicht mehr kompilieren.

8 Nutzung von .NET Assemby in LabVIEW

Abbildung 8-1: Logo von Microsoft .NET (http://www.microsoft.com/)

8.1 Überblick

Ich habe über meine Probleme mit Kollegen diskutiert und sie haben mir gesagt, falls es ein .NET Wrapper gibt, es vielleicht einfacher wäre damit zu arbeiten als mit DLL. Ich habe mich dann entschieden mit C#.NET neu anzufangen. C# ist eine Programmiersprache, die .NET geeignet ist. Wikipedia definiert .NET als *„[Bezeichnung] eine von Microsoft entwickelte Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. Eine zunehmende Anzahl von Programmen benötigt .NET als Unterbau. .NET besteht aus einer Laufzeitumgebung (in der die Programme ausgeführt werden) sowie einer Sammlung von Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (Services). Es bietet dem Programmierer viele Lösungen für Standard-Aufgaben an, so dass er nur noch die eigentlichen Programmfunktionen erstellen muss“.* LabVIEW hat eine ganze Palette um .NET Elemente in LabVIEW einzubinden.

8.2 Tests der Einbindung.

Der .NET Wrapper für OpenNI ist bei Installierung der Software mit installiert. Es bedeutet, dass ich es nicht programmieren muss. Nach kurzem Lesen auf .NET in LabVIEW habe ich ein mitgeliefertes Beispiel geöffnet (Anhang 15.11). Dieses Beispiel macht nichts anders als mein

C++ Beispiel: es nimmt die Tiefe des Mittelpixels aus der 3D Kamera. Ich habe versucht es in LabVIEW zu übersetzen. Das Ergebnis kann man in Anhang 15.12 sehen. Obwohl die Programmierungsart ganz anders ist, kann man die gleichen Funktionen fast immer verwenden. Ich habe als Kommentar in Anhang 15.12 den C# Code geschrieben, sodass es mit LabVIEW verglichen werden kann. Der einzige Unterschied ist, dass man aufpassen muss alle Referenzen zu schließen. Insgesamt funktioniert das Programm vollkommen. Der nächste Schritt ist jetzt meinem Projekt zu verwirklichen.

9 LabVIEW Projektausblick

9.1 Kurzfassung

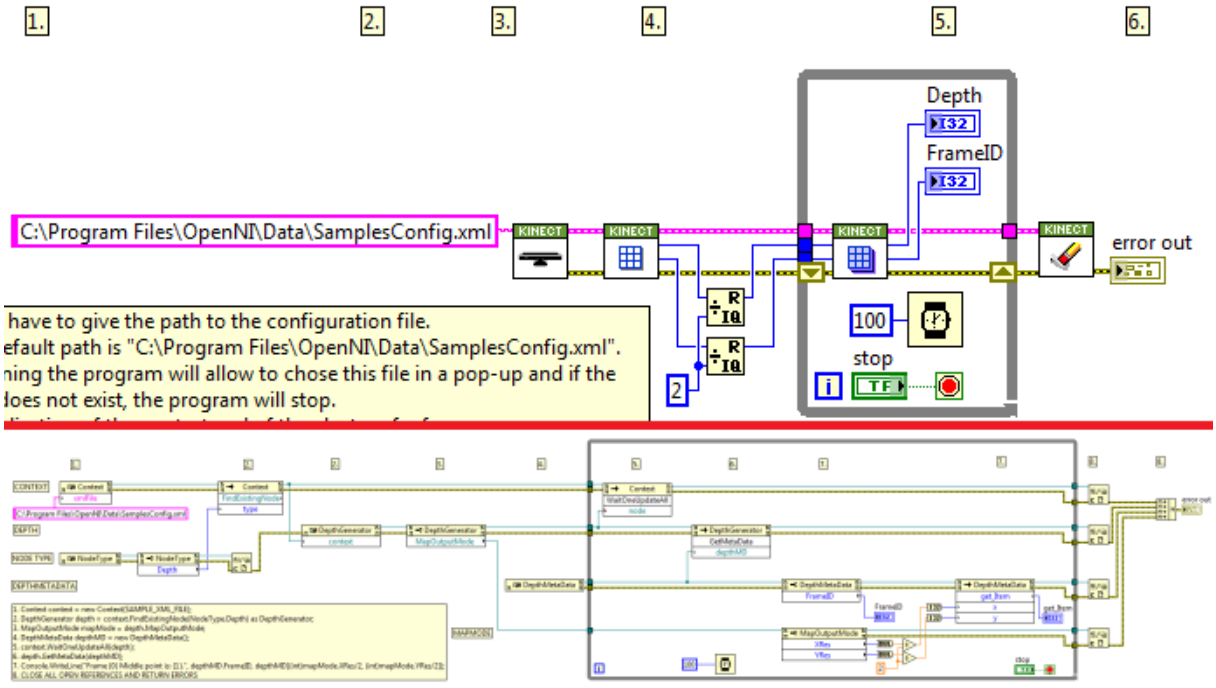


Abbildung 9-1: Die gleiche Funktion. Oben mit gepackte Unterfunktionen und unten ohne.

Vermutlich: Wie in allen Programmiersprachen es ist wichtig auch in LabVIEW die verschiedenen Funktionen im Programm zu trennen. Ich habe in letzte Kapitel ein Beispiel Programme der C# benutzt in LabVIEW programmiert. Meine erste Aufgabe war dann dieses Programm in Funktionen zu trennen. Das Ergebnis kann man in Abbildung 9-1 sehen. Das Programm ist mehr als 10-mal kleiner, einfacher zu verstehen und „benutzerfreundlicher“. Den Weg sowie die neue entwickelten Funktionen werde ich im nächsten Kapitel erklären.

9.2 Architektur des Projekts

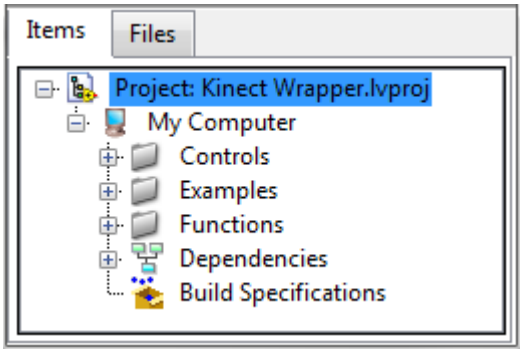


Abbildung 9-2: Ordnerarchitektur am Anfang meinem LabVIEW Projekt.

Um die Senkung von Abbildung 9-1 zu verwirklichen, wollte ich in ein Projekt entwickeln. Der Vorteil von einem Projekt ist, dass es erlaubt mich die VIs zu trennen und in virtuellen Ordner organisieren. Das Projekt erlaubt mir auch Applikation aus meinem Programm zu kompilieren. Die erste Frage, die ich mir stellte, lautete „wie kann ich die Referenz auf meinen .NET Konstrukt weiterleiten?“. Ich habe angefangen, diese Referenz als Ausgang von Funktionen weiterzuleiten. Das Hauptproblem ist, dass es viele Funktionen gibt. Die Funktionsblöcke haben keine unendliche Anzahl von Ein- und Ausgängen und meine 20 Referenzen sind für eine Funktion zu viel. Nach einigen unbefriedigenden Versuchen habe ich mich entschieden meinen Referenzen in ein Cluster zu packen. Diese Cluster habe ich als typ-definierte Kontrolle gespeichert.

Typ-definiert bedeutet, dass es als ein separater VI gespeichert ist. Es hat als großen Vorteil, dass die Änderungen auf diese Kontrolle werden in alle Funktionen die diese Kontrolle benutzen aktualisiert sein. Es war für mich wichtig, als ich am Anfang noch nicht wusste, wieviel .NET Konstrukt Referenz ich für das ganze Project benutzen würde. Die Idee ist, dass diese Cluster Kontrolle zu allem VIs weiterleitet wird und dann erst werden die zu benutzenden Referenzen ausgebündelt.

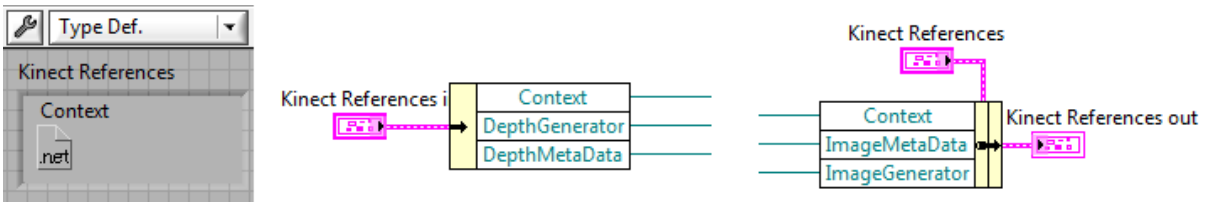


Abbildung 9-3: Kinect Reference ist mein Typ Definierte Cluster von .NET References Kontrolle. Link ist sein Definition in mein Projekt. In der Mitte und nach Recht sind die Bündeln und Ausbündeln Methode (von Cluster nach Referenz und von Referenz nach Cluster).

In Abbildung 9-2 kann man sehen, dass zusätzlich zum „Controls“ Ordner (unten „My Computer“) es noch 4 Verzeichnisse gibt: „Examples“, „Functions“, „Dependencies“ und „Build Specifications“. Ich habe in „Examples“ alle fertigen Programme geordnet. Drinnen gibt es nur die aufspielbaren Programme. Diese Programme sind auch die, die in der HTML-Dokumentation dokumentiert sind. In „Functions“ gibt es alle selbstentwickelten Unterfunktionen, die von den Beispielen aufgerufen werden plus ein paar Funktionalitäten, die noch keine Beispielfunktionen haben. Die Funktionen und Unterfunktionen sind von Ordner getrennt nur, weil es benutzerfreundlicher ist. Man soll nicht durch die ganze Liste von Unterfunktionen nach Beispielen suchen. „Dependencies“ enthaltet alle LabVIEW Funktionen und die Bibliothek, die ich in meinem Projekt benutze. Zum Beispiel kann man in den Dependencies die 3D Graph Bibliothek für das TestMap.vi Beispiel und das Vision plug-in für TestImage.vi finden. Zuletzt gibt es die „Build Specifications“, die immer noch leer sind, weil ich noch keine Applikation erstellt habe.

10 Die Beispielprogramme

10.1 Kurzfassung

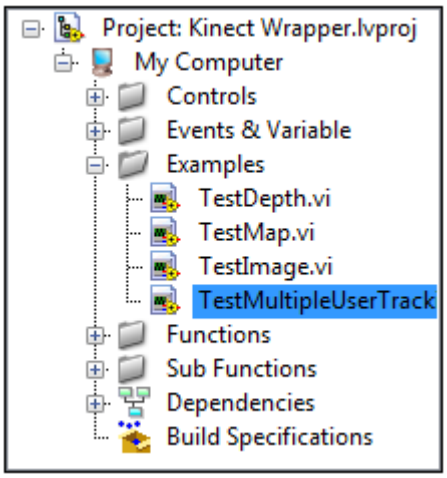


Abbildung 10-1: Das Projektbaum von den Beispiele.

Mit ein Projektarchitektur und eine .NET Datenübertragungsmethode konnte ich anfangen Unterfunktionen und Beispiel zu programmieren. Diese Beispiele sowie die Funktionen sollten genug dokumentiert sein sodass Benutzer es verstehen können. Die Dokumentation ist in HTML gespeichert in das Anhang DVD unter das „Dokumentation“ Ordner. In diese Kapitel werde ich diese Dokumentation zusammenfassen.

10.2 TestDepth.vi

Abbildung 10-2: Icon von TestDepth.vi

Dieses Beispielprogramm erlaubt die Tiefe von dem Mittelpixel das 3D Bild zu schauen. Das Front Panel und das Block Diagramm sind in Anhang 15.13). Dieses Programm Funktionalität ist fast gleich wie das erste C# basierte LabVIEW Programm. Die Unterschiede sind, dass die verschiedenen Funktionen jetzt in verschiedene VIs verpackt sind. Performanz- und Stabilitätsmäßig ist diese Version auch verbessert: jede Funktionen benutzen die gleiche

Fehlerlinie und die xml-Konfigurationsfile ist jetzt mit einer Dialogbox geprüft. Das Beispiel ist in Anhang 15.13.2 dokumentiert. Um zusammenzufassen:

- Zu [1.] ruft man die Dialogbox auf, um die Konfigurationsfile zu wählen. Wenn keine existiert, stürzt das Programm ab.
- Zu [2.] wird Context initialisiert. ~~Das~~ Context ist das globale Konstrukt von der OpenNI Bibliothek. Es wird von den anderen oft aufgerufen. Deswegen ist es in eine separate Funktion gepackt: jede Programm wird mit diesem VI anfangen.
- Zu [3.] ist das Tiefenbild initialisiert und die Bild-Auflösung wird zurückgegeben .
- Die x- und y-Auflösung wird in [4.] halbiert, um die Koordinaten des Mittelpunkts auszurechnen. Um diese Dividision durchzuführen habe ich statt das normaler dividieren Block, den „Quotient und Reminder“ benutzt. Es ist so weil die Auflösung und die Koordinaten sind integer Zähle. Das normale dividieren Block nimmt nur float Zähle als Argument. Um effektiver das Programm auszuführen und um weniger Zahlübersetzung zu habe, ist es besser mit diese „Quotient und Reminder“ Funktion zu arbeiten. Koordinaten werden
- Zu [5.] in die While Schleife weitergegeben und die Tiefe das Mittelpixel jede 100ms gemessen. Es dauert bis zum das Benutze auf das Stop Drück drückt. Dann werden bei [6.] alle Referenz geschlossen und der Error Anzeiger aktualisiert.

Das Blockdiagramm und alle Informationen auf die Unterfunktionen können in der HTML Dokumentation für TestDepth.vi gefunden werden unter „Dokumentation\HTML Report - Test Depth\TestDepth.html“ auf das CD in Anhang. In meinem Programm benutze ich vier Unterfunktionen:

- 

CreateContext.vi: Dieses VI initialisiert den Context „.NET Assembly“ mit einem XML Konfigurationsfile. Das Referenz auf diesen Context wird in den Typ~~e~~ Definiert Cluster gespeichert und zurückgegeben.

-  InitializeDepth.vi: Initialisiert alle Tiefe „.NET Assemblies“ mit dem Kontext. Die Mappe wird auch initialisiert und seine Auflösung sowie die Referenz auf die neuen Assemblies werden zurückgegeben.
-  GetDepth.vi: Dieses VI gibt die Entfernung zwischen die Kamera und ein Objekt zurück. Man muss die Koordinaten des Objekts als Argument übergeben.
-  CloseRefs.vi: Schaltet das Context aus, Schließt alle .NET Referenz. In das Block Diagramm sieht man, dass alle eine andere Fehler Linie benutzt als die Fehlereingang. Ich habe es so gemacht sodass, das Context in allen Fällen ausgeschaltet wird (wenn ich den Fehlereingang zu der Shutdown Methode verbinde, würde diese Methode nicht ausgeführt, falls es früher in das Programm einem Fehler gibt).

10.3 TestMap.vi

Abbildung 10-3: Icon von TestMap.vi

Als das vorherige Programm funktioniert hat, habe ich auf die erste Zielaufgabe mein Bachelorarbeit hingearbeitet. Ich sollte die rohen Daten der Kamera auslesen. Bei rohen Daten versteht man die Tiefenmappe und die Kamera Bild. Als ich schon die Tiefe mit TestDepth.vi lesen konnte habe ich mit der Tiefenmappe angefangen. Wie man in Anhang 15.14.2 sehen kann, ist die Architektur ähnlich wie bei TestDepth.vi. Ich habe in [1.] den Konfigurationsfilepfad angegeben statt einer Dialogbox. Den Konfigurationsfilepfad/ die Dialogbox kann der Benutzer selbst wählen. Es ist nur um das Beispielprogramm kleiner um es lesbarer zu halten. Wie bei letzte Beispiel initialisiere ich das Referenz Cluster, das Context und die Tiefenmappe in [2.] und [3.]. [4.], [5.] und [6.] sind in meine while Schleife. Nachdem ich die Auflösung des Bildes von [3.] bekomme, leite ich es weiter zu [4.] sodass mein Unterprogramm weiß wie groß das Tiefes Array sein soll. In [4.] wird dieses Array berechnet und als 2D Array von UI16 zurückgegeben. Ich muss dann dieses Array konvertieren in Single float Komma sodass, kein Komma es in [5.] nutzbar ist. [5.] ist ein VI von National

Instruments der ein 2D Array von float nach ein 3D Graph übersetzt. Dieser Graph wird in [6.] angezeigt. Dieser Schritt verlangsamt das Programm: Aktualisieren eines 3D Graph von 30722 Punkten (640*480 Standardeinstellung) dauert ca. 2,5 Sekunden auf dem Entwicklungscomputer. Deswegen gibt es den 3 Sekunden Timer in die Schleife. Das Ergebnis aus diesem Graph ist in Abbildung 10-4 sichtbar.

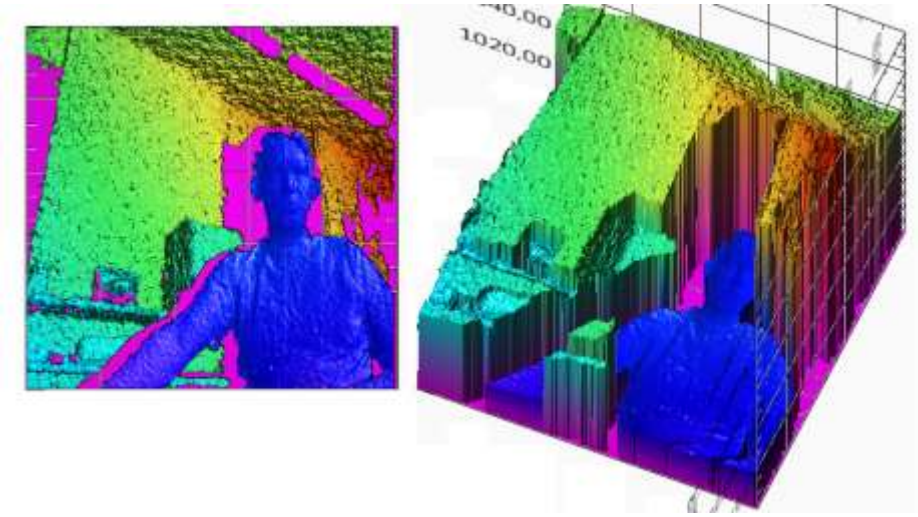


Abbildung 10-4: Die Abstandinformationen sind in einem Graph angezeigt

Dieses Anzeigenelement ist nur da, um die Daten besser zu visualisieren. Man kann sich vorstellen, dass, in der Zukunft die Benutzer die Tiefeninformationen mit mathematischen Werkzeugen benutzen werden. Deswegen ist diese Zeitbegrenzung kein wichtiges Kriterium. Allerdings, falls die Benutzer trotzdem die Tiefe schneller visualisieren möchten, habe ich ein alternatives Visualisierungsunterprogramm entwickelt, das NI-Vision benutzt. Er wird in diesem Beispiel nicht dokumentiert sein aber es funktioniert sehr ähnlich wie „DisplayImage.vi“ von Kapitel 10.4. In [7.] schließt das Programm alle Referenzen und zeigt die Fehler.

Das Blockdiagramm und alle Informationen der Unterfunktionen können in der HTML Dokumentation für TestMap.vi gefunden werden unter „Dokumentation\HTML Report - Test

Map\TestMap.html“ auf der CD im Anhang. In meinem Programm benutze ich nur eine andere selbstgemachte Funktion als in TestDepth.vi:

-  GetMap.vi: Dieses VI berechnet die Mappe von tiefe. Es nimmt als Argument die Auflösung das Tafelbild und gibt das Ergebnis als ein 2D Array. Der Aufbau dieses Unterprogramms sollte am Anfang ähnlich sein wie „GetDepth.vi“ in Kapitel 3.2 mit zwei For Schleife, um die Tiefe von jedem Punkte zu messen und in die Tabelle speichern. Das Problem ist, dass LabVIEW für .NET nicht optimiert und eine Methode 307200-mal aufrufen dauert mehr als 2 Minuten. Mein größtes Problem war, dass ich die Tabelle von zahl, die ich aus einer Methode bekam, nicht direkt benutzen konnte: der Datentyp war System.IntPtr. Nach einigen erfolglosen Experimenten habe ich eine .NET Methode aus das mscorlib Bibliothek gefunden, der ein System.IntPtr in ein 1D Array von Integer kopiert. Dann brauchte ich nur noch das Array zu ordnen und habe die Messung einer Mappe von 160 000ms auf 20 ms schnell gemacht.
-  DisplayDepth.vi: Dieses VI ist nicht direkt in das Beispiel eingefügt. Ich habe es so gemacht, weil das nächste Beispiel benutzt ein ähnliches VI und es braucht Vision, ein Toolkit der nicht kostenlos ist und der nicht bei allen LabVIEW Benutzer installiert ist. Ohne es können alle Leute, die ein Kinect installieren haben, dieses VI testen. Ein Screenshot von dem Frontpanel und die Änderungen auf die While Schleife von TestMap.vi sind in Anhang 15.16 sichtbar. Das VI nimmt das 2D Tiefentabelle und das Bildauflösung und berechnet damit eine Vision Bild.

10.4 TestImage.vi

Abbildung 10-5: Icon von TestImage.vi

Kinect ist eine 3D Kamera, aber sie kann auch eine normale Kamera sein. Der zweite Aspekt von meiner ersten Aufgabe war die Kameradaten zu lesen. Deswegen sollte ich auch ein

Beispiel programmieren um ganz normalen Kameradaten zu lesen. Der Code und das Front Panel sind in 15.15 sichtbar. Wie immer fängt das Programm an mit der Context Initialisierung in Schritt [1.] und [2.]. In [3.] wird das Bild initialisiert. Dieser Schritt ist sehr ähnlich wie die Tiefe Initialisierung, nur werden Bilder initialisiert sein statt tiefen Assemblies. Wie bei den anderen Beispielprogrammen werden das Cluster von Referenz und die Bildauflösung in die While Schleife weitergeleitet. In [4.] wird das Bild ausgenommen und in ein 2D Array gespeichert. Jedes Feld des Array ist ein Farbenpixel in RGB Kodiert (Unsigned integer 32 Bits). Dieses Array kann in [5.] in ein Bild übersetzt sein. [5.] braucht NI-Vision zum Funktionieren. Deswegen habe ich diese Funktion nicht direkt in [4.] gepackt. Das berechnete Bild wird in [6.] gezeigt. Die While Schleife ist schnell genug ausgeführt um ein ruckloses Bild zu haben. Wie immer werden in [7.] die Referenzen geschlossen.

Das Blockdiagramm und alle Informationen auf die Unterfunktionen können in der HTML Dokumentation für TestImage.vi gefunden werden unter „Dokumentation\HTML Report - Test Image\TestImage.html“ auf der CD im Anhang. In meinem Programm benutze ich drei andere selbstgemachte Funktionen wie in die anderen Beispiele:

- 

InitializeImage.vi: Diese VI initialisiert meinen Bildreferenzen. Es ist gleich gebaut wie InitializeDepth.vi. Es werden nur Bildreferenzen initialisiert statt Tiefereferenzen.
- 

GetImage.vi: Prinzipiell sollte diese VI wie GetMap.vi funktionieren. Leider ist es nicht ganz der Fall. Die Daten sind auch von die OpenNI Assemblies als System.IntPtr aber als ich die frühere Methode ausgeführt habe war das Bild qualitativ sehr schlecht. In der Dokumentation habe ich gefunden, dass der Inhalt dieses Zeigers kein Pixel ist, sondern eine Farbe. Es bedeutet, dass ein Pixel in drei Adressen gespeichert ist und dass das erste Array dreimal größer als die Auflösung sein sollte. Es bedeutet auch, dass ich das Pixel Array aus den Farben selbst erzeugen musste.

-  DisplayImage.vi: Diese Unterfunktion konvertiert ein 2D Array von Pixel in ein Vision Bild. Man muss das Array sowie die Auflösung vorgeben, sodass das Bild initialisiert und „gezeichnet“ werden kann.

10.5 TestMultipleUserTracking.vi

Abbildung 10-6: Icon von TestMultipleUserTracking.vi

Der nächste auszuführende Aufgabe ist die Implementierung von „User Tracking“ in LabVIEW. Es ist die wichtigste Aufgabe meines Projekts, weil niemand es bisher gemacht hat. Es ist auch wichtig, weil die nächste Aufgabe mit dem Pendel unabhängig von diesem ist. Ich habe ein Beispiel programmiert, wo diese „tracking“ Informationen zeigen und lesen sein könnten. Der Code und das Front Panel sind in Anhang 15.17 sichtbar. Dieses Mal fängt das Programm nicht gleich mit der Initialisierung des Contextes, sondern mit der Initialisierung eines „Rendezvous“ (Treffpunkt auf Französisch) in [1.]. Dieses Rendezvous wird mit der Konstante 2 initialisiert. Es bedeutet, dass das Programm an einem „wait at rendezvous.vi“ bloc bis eine parallele Schleife ein zweites „wait at rendezvous“ ausführen wird. Dann sind die zwei freigeschaltet und die zwei schleife gehen weiter ?.

Bei [2.] Initialisiere ich meine Funktionale Globale Variabel von Bild. Ich benutze die NI-Vision VIs um eine Image Referenz zu bauen, die später geschrieben und angezeigt wird.

In [3.] gibt es meine Anzeigeschleife. Da wird zuerst am rendezvous gewartet und dann das aktuelle Image anzeigt. Beim Stopp des Programms wird die „Press Start“ Bild wieder angezeigt.

[4.] ist die Hauptschleife meines Programms wo die Kinect Initialisierung sowie die Messungen und den Datenanzeigen ausgeführt werden.

Bei [5.] wird die Initialisierungsbild in die Image variabel gespeichert. Dann wird das Programm

in [6.] an rendezvous warten. Die rendezvous sind immer nach eine Imageänderung. Es ist so um die [3.] Unterschleife zu aufhalten (abzubrechen? unterbrechen?): Das Vision Anzeigeelement wird nur aktualisiert sein, wenn es eine Änderung bei die Vision Image gibt um Prozessorleistung zu speichern.

Erst bei [7.] wird die Kinect Kamera initialisiert sein. Für den Aufbau dieses Beispiels habe ich die Funktionsverteilung in die UnterVIs etwas geändert als ich die Synchronisierung immer mit Depth macht und nicht mehr unabhängig von dem benutzte Datentyp. Deswegen initialisiere ich meine Tiefe in [8.].

[9.] ist die Initialisierung des Benutzers. Initialisierung des Benutzer entspricht Erstellung von Benutzerreferenzen, Initialisierung der Benutzer-Events und Rückgabe der verschiedenen Artikulationsreferenzen. Nach den verschiedenen Initialisierungen fängt die Messschleife an.

Im [10.] wird das Programm warten auf das nächste update des Tiefessensors.

Im [11.] werden die Benutzerdaten gemessen. Dieses VI gibt zurück ein Cluster mit den Benutzerdaten, eine Tabelle von Pixeldaten, die Liste von erkannte Benutzer und einem Enum mit dem Tracking Status. Diese Tracking Status wird

in [12.] benutzt um die nächste Image zu entscheiden ht. Bei Waiting und Calibrating wird nur ein Informationsbild gezeigt und bei Tracking die Tabelle von Pixeldaten, die eine Repräsentation des Benutzerskelett entspricht.

In [13.] wird die Benutzeroberfläche mit den vorgewählten Eigenschaften aktualisiert. Diese Eigenschaften sind die Benutzernummer und die Artikulation, die man isoliert will.

In [14.] nimmt man den Cluster von dem richtigen Benutzer aus der Tabelle. Es zeigt auf das Front Panel. Die Benutzerliste wird auch angezeigt.

Bei [15.] wartet man wieder auf rendezvous.

In [16.] werden .NET die Referenz und die Events geschlossen und endlich

in [17.] wird das rendezvous geschlossen und die möglichen Fehler angezeigt.

Das Blockdiagramm und alle Informationen auf die Unterfunktionen können in der HTML Dokumentation für TestMultipleUserTracking.vi gefunden werden unter

„Dokumentation\HTML Report - Test Multiple User Track\TestMultipleUserTracking.html“ auf der CD im Anhang. In meinem Programm benutze ich 5 andere selbstgemachte Funktionen wie in die anderen Beispiele:

- 

Variable_Image.vi: Das ist die Funktional Globale Variabel meinem Programm. Die einhaltet die Image Informationen die auf dem Front Panel angezeigt werden. Die Variable hat 7 Zustände: Initialize, Set Start, Set Initialize, Set Wait, Set Calibrate, Set Skeleton und Get Image. Initialize initialisiert das Image, das später als eine 400 Pixels * 400 Pixel farbiges IMAQ Bild benutzt wird. Set Skeleton zeigt die Tabelle von Pixel, die im Eingang gegeben ist, an. Get Image leitet in den Ausgang das Bild, das in der Variabel gespeichert ist. Die vier anderen Zustände speichern nur in die Variable das Zustandsbild (Press Start, Initializing, Witing und Calibrating. Siehe Anhang 15.18).
- 

Initialize User.vi: Dieses VI initialisiert alle Benutzerdaten. Da werden auch die Benutzerevents registriert. Zusätzlich zur Öffnung der Benutzerreferenz wird die Artikulationsreferenz (Joints) in eine Tabelle gespeichert. Wichtig ist auch in diesem VI die Registrierung von Events. 3 werden registriert: die Anerkennung von einem neue Benutzer, die Anerkennung von eine Pose und die Ende des Kalibrierung.
- 

Wait on Update.vi: Dieses VI wartet auf eine Aktualisierung des Tiefessensors der Kamera. Das Programm läuft weiter nach dieser Aktualisierung.
- 

Get User.vi: Diese VI gibt bei jedem Aufruf die Benutzerdaten zurück. Diese Benutzerdaten sind die Tabelle von Benutzern, in der alle anerkannten Benutzer gelistet sind, die Globale Spurstatus, die Tabelle aus Pixel, und das Cluster aus Benutzereigenschaften.
- 

Get User Joint.vi: Diese VI schreibt in das Cluster von Benutzereigenschaften die 3D Position von die Vorgewählt Artikulation. Gibt das Aktualisierte Cluster zurück.

10.6 Weitere Informationen

Abbildung 10-7: Meine selbstgebaute Palette

Nach der Entwicklung dieses Beispiels war meine Arbeit mit dieser Bibliothek noch nicht ganz fertig. Ich wollte noch ein paar Sachen machen, wie zum Beispiel die Implementierung einer Palette. Aber am wichtigsten war die Veröffentlichung meiner Bibliothek. Bevor die Veröffentlichung sollte ich das Gesamte programm wieder testen – auf meinem Rechner sowie auf andere mit anderem Betriebssysteme. Jedes Mal sollten die Treibers neuinstalliert werden. Nach einiger Korrektur habe ich es auf der Entwicklergesellschaft Seite von national Instruments hochgeladen. Es ist an die folgende Addresse lesbar: https://decibel.ni.com/content/docs/DOC-16978 . Ich habe auch für die Veröffentlichung einem Beispielvideo gemacht. Damit endet sich die dritte Teil meinem Bachelorarbeit. Zuletzt gibt es die Integration mir das Pendel zu machen.

11 Das Newton Pendel

11.1 Überblick

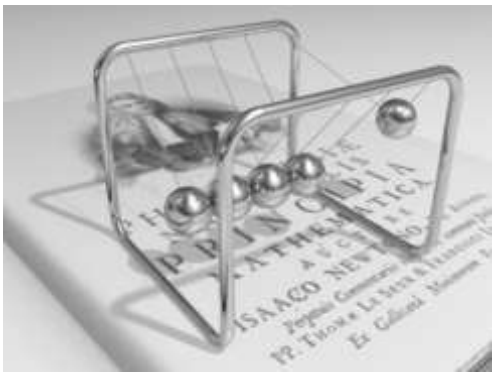


Abbildung 11-1: Einem Newtons Pendel (http://de.wikipedia.org/wiki/Kugelstoßpendel)

Das Newton Pendel wurde ursprünglich nicht von Isaac Newton erfunden, sondern nur nach ihm benannt. Dieses Gerät demonstriert den Impuls- und Energieerhaltungssatz. Es besteht aus einer Reihe von Metallkugeln, die alle an einem Rahmen aufgehängt sind, sodass sie einander berühren. Wenn eine Kugel am Ende der Reihe hochgehoben und anschließend losgelassen wird, so wird die potentielle Energie in kinetische Energie umgewandelt und an die Nachbarkugel weitergegeben.

Abbildung 11-2: Das gesteuerte Pendel von National instruments

Herr Ridelbauch von der Marketingabteilung hat einem Elektronisch gesteuert Pendel entwickelt (Abbildung 11-2). Es soll mit 5 Schrittmotoren funktionieren die allem mit NI-9501 Modulen in einem cRIO gesteuert sind. Der Aufbauplan kann unter

„Dokumentation\Newton's Cradle in Motion.pdf“ auf der CD in Anhang gefunden sein. Mit diesem Pendel sollte ich mit der Bewegung mein Arm die äußere Kugeln gleich bewegen. Leider war an der Zeit mein Bachelorarbeit der Pendel noch nicht ganz funktionierbar. Ich habe als Ersatz der gleiche cRIO mit der gleiche Modul und ein Motor gekriegt um eine Lösung trotzdem zu entwickeln.

11.2 Realisierung

Der erste Schritt diese Realisierung war ein Protokoll zu erstellen. Ich sollte definieren mit welschen Bewegungen bewegt sich der Motor, wie es sich bewegt und wann es aufhören soll meine Bewegungen zu messen. Eine alte Idee war, der ganze Zeit das Winkel zwischen das Benutzerkörper und seine Arm zu messen. Leider war diese Idee nicht machbar als der winkel würde der ganze Zeit gemessen und das Pendelkugel nie freigelassen.

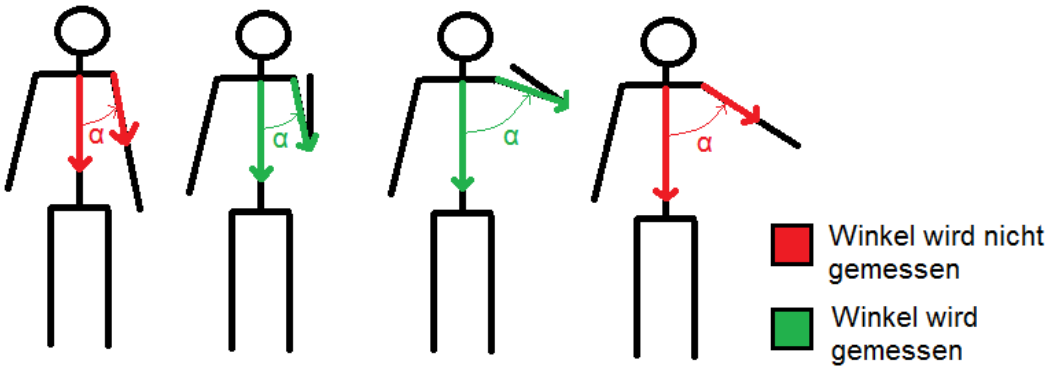


Abbildung 11-3: Die entscheidende Lösung um der Winkel zu messen

Die Lösung die ich gefunden habe ist Winkel messen nur, wenn es eine Flexion auf die Ellbogen gibt (siehe Abbildung 11-3). Wenn es keine Flexion gibt würde der Pendel einfach frei gelassen.

Eine die Herausforderung diese Aufgabe war ein Programm zu entwickeln mit eine ziemlich komplexe Architektur wo meine selbstentwickelte Treibers, Motion, LabVIEW Real Time,

FPGA und Vision zusammenspielen. Die gesamte Architektur meinem Beispielprojekt sieht wie folgend aus:

DIAGRAMM 1

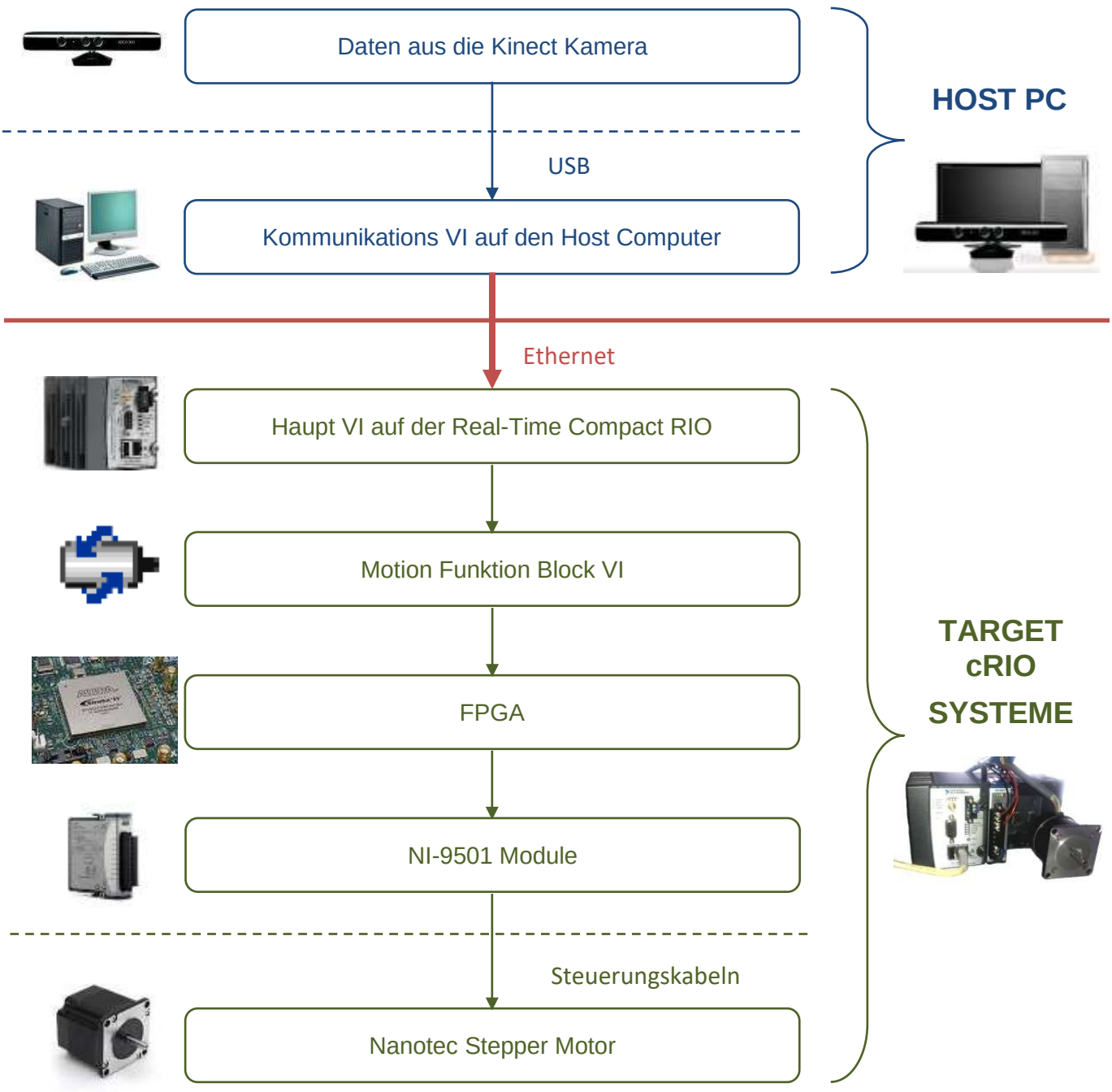


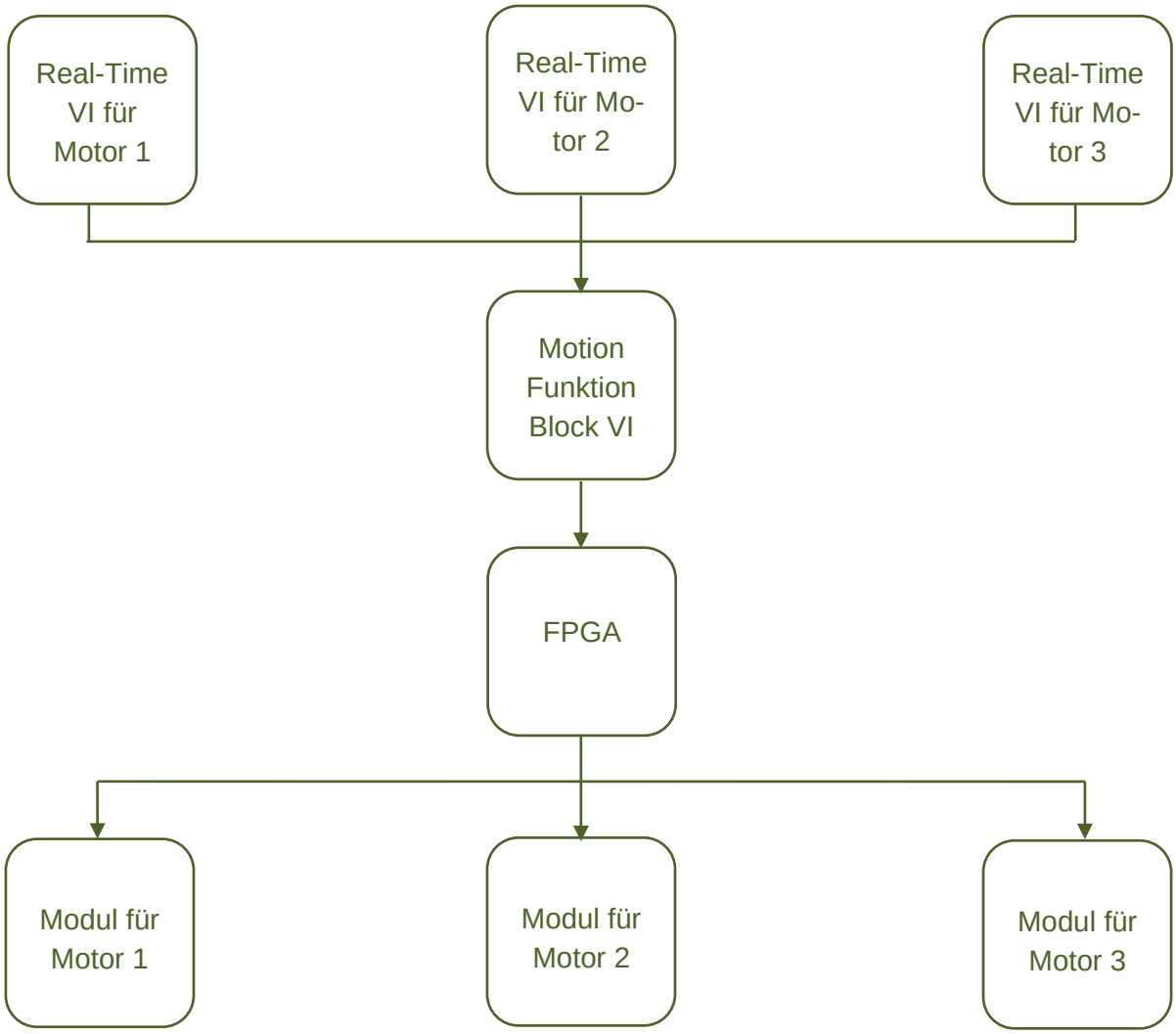
Abbildung 11-4: Dar cRIO System mit die Motorsteuerung und der Motor

Die Art und Weise wie ich die Daten von der Kamera lese war schon in den früheren Kapiteln erklärt. Ich werde dann die „Host PC" Teil des vorherigen Diagramms jetzt nicht wieder erklären. Wichtig ist zu verstehen wie der Compact RIO System funktioniert. Der cRIO Echtzeit-Kontroller wird Daten von dem PC durch Ethernet-Verbindung kriegen. Diese Daten werden cyclisch überprüfen und als Position für das Motor Übersetzt. Diese Zielposition wird geschrieben, sodass der Motion Funktion Block es verstehen kann. Diese Funktion Block funktioniert ein bisschen wie der Dirigent für Stepper- und Servomotoren. Es fast alle Informationen zusammen und leitet weiter die auf die FPGA[4] oder fast die aus die FPGA zusammen (Im Fall von Kodierungsrad, zum Beispiel). Der FPGA Chip Verteilt die Informationen zwischen der Kontroller und die Modulen. Es kümmert sich um die Ganze Zwischenkommunikation in das System. In meinem Fall benutze ich nur einen Motor, aber falls ich mehrere Motoren in separaten Modulen habe und jede Motoren einer separaten Führungs-Vi hat, würde mein Diagramm anders aussehen. Dieses Beispiel könnte helfen zu verstehen wie alle zusammenspielt:

[4] Wikipedia definiert FPGA wie „*Ein Field Programmable Gate Array (kurz: FPGA) ist ein Integrierter Schaltkreis (IC) der Digitaltechnik, in den eine logische Schaltung programmiert werden kann. Die englische Bezeichnung kann übersetzt werden als: im (Anwendungs-)Feld programmierbare (Logik-)Gatter-Anordnung.*

Anders als bei der Programmierung von Computern oder Steuerungen bezieht sich hier der Begriff Programm nur in zweiter Linie auf die Vorgabe zeitlicher Abläufe im Baustein, sondern vor allem auf die Definition von dessen Funktionsstruktur. Durch die Programmierung von Strukturvorschriften wird zunächst die grundlegende Funktionsweise einzelner universeller Blöcke im FPGA und deren Verschaltung untereinander festgelegt. Man spricht daher auch von der Konfiguration eines FPGAs." - http://de.wikipedia.org/wiki/Field_Programmable_Gate_Array

DIAGRAMM 2



Mit diesem Beispiel ist es vielleicht einfacher die Rolle des Funktionsblocks und des FPGA-Chips zu verstehen. Mein NI-9501 Modul ist speziell für Stepper-Motoren gedacht. Es hat eine externe Spannungsversorgung, um den Motor zu steuern. Es kriegt vom FPGA die Zielposition, Geschwindigkeits- und Beschleunigungsinformationen und steuert damit den Motor bis um die Zielposition erreicht ist.

11.3 Die Programme

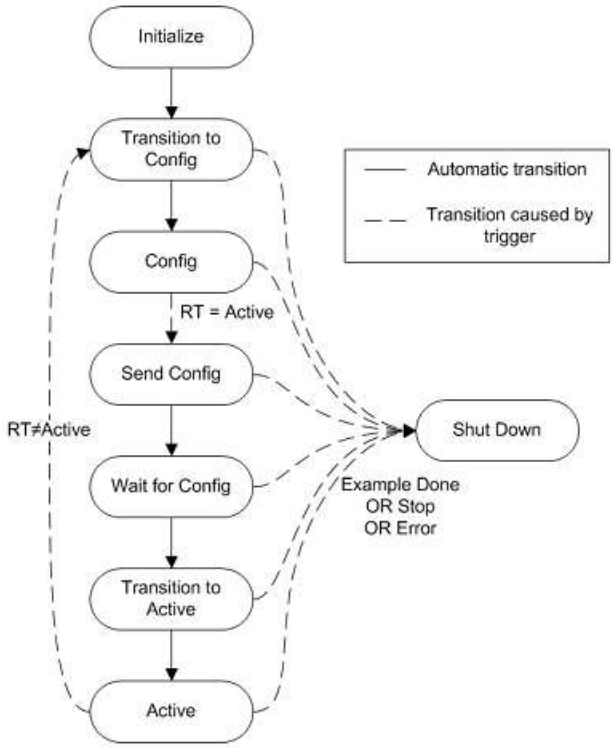


Abbildung 11-5: Diagramm von den MotionFunktion Block

Mit der Motion Toolkit für LabVIEW gibt es einem mitgelieferten Beispiel der für Stepper Motoren mit NI-9501 Modul gemeint ist. Diese Beispiel kann Standardweise unter „C:\Program Files\National instruments\LabVIEW 2010\examples\Motion\Unbound Axis\Stepper Drive (9501)\Stepper Drive (9501).lvproj“ gefunden sein. Dieses Beispiel enthält schon der FPGA Programm für meinem Modul sowie der Motion Funktion Block VI. Als diese VIs Standard sind ich kann damit Zeit sparen und mich auf die Kommunikation zwischen die Kamera und der Echtzeit Kontroller konzentrieren.

Der Diagramm 1 dieses Kapitel zeigt wie alle organisiert. Am besten schauen wir die Programme von Ende bis Anfang. Der Stepper Motor und der 9501 Modul sind Hardware Stück und enthalten kein Code.

Die FPGA Chip macht die Kommunikation zwischen der Echtzeit Kontroller und die Module. Der LabVIEW Code für die FPGA kann Man in Anhang 15.19 sehen. Die Funktionalitäten sind erklärt in dem Code und brauchen hier keine Wiederholung.

Höher in der Diagramm gibt es der Motion Funktion Block VI. Diese VI macht die Kommunikation zwischen der obere Echtzeit VI und die FPGA Chip. Das Programm mit Kommentare ist in Anhang 15.20 verfügbar. Ein Diagramm seine Funktion ist auch als Abbildung 11-5 lesbar.

Ein Schritt oben in Diagramm 1 gibt es der Haupt VI der Real-Time Compact RIO. Der habe ich ganz selbst geschrieben. Der LabVIEW Code ist in Anhang 15.21 verfügbar. Um die Kommunikation zwischen mein Echtzeit Kontroller und mein Computer zu verwirklichen, habe ich mir für eine „Shared Variable“ entscheiden. Die kann man ganz link in meinem Code finden. Am Anfang meinem Programm wird diese „Shared Variabel“ (die die Angle variabel gennant ist) mit 0 initialisiert. „Example Done“ wird auch mit False initialisiert sein. In die erste While-Schleife Steuere ich der Motor. Es ist in eine While-Schleife gepackt, weil die Motion Blöcke folgen nicht der Datenflussprinzip. Der muss dann so lang ausgeführt sein bis die „Done“ Ausgang True ist.

Dann werden 2 parallel schleifen ausgeführt. Die untere macht nichts anders wie Daten kontinuierlich lesen. Es nimmt die Position und die Geschwindigkeit und schreibt die in eine Tabelle. Die oben ist die hauptschleife des Programm. Es ist eigentlich 2 schleifen ineinander. In die aussende wird die Zielposition der Steppermotor von der Host PC gelesen. Mit die Geschwindigkeit und die Beschleunigung wird die in die innere schleife weitergeleitet. Wie für die Steuerung der Motor folgen da die Motion Blöcke der Datenflussprinzip nicht. Deswegen lauft diese schleife bis um die Zielposition erreicht ist. Die boolesche Konstante die in der Schiebregister geschrieben ist, benutzt man um in die nächste Iteration der motor wieder zu bewegen. Im Gegensatz zu Standard LabVIEW Blöcke die bei „execute == TRUE“

ausgeführt werden, werden die Motion Blöcke nur ausgeführt wenn „execute“ eine steigende Flanke am Eingang erkennt.
Zuletzt wird der „example done“ boolesche Anzeigenelement aus TRUE gesetzt und die Fehlern gezeigt.

Diese Programme habe ich vor der Haupt VI der Host PC entwickelt, weil es einfacher zu testen war (mit eine Konstante statt die Shared Variable). Die Kommunikation mit dem Host PC habe ich erst nachher entwickelt. Das Programm ist in Anhang 15.22 verfügbar. Der ist auf die gleiche Architektur wie TestMultipleUserTracking.vi basiert. Um mehrere Informationen auf diese Programm zu haben kann Man 10.5: TestMultipleUserTracking.vi und die Dokumentation unter „Dokumentation\HTML Report - Test Multiple User Track\TestMultiple-UserTracking.html“ auf das CD in Anhang lesen.

Die Unterschiede mit der Tracking Programm kann Man in die zentrale Case Structure finden. Ich lese die Position in der Raum von punkten Paar und berechnet der Vektor damit (Schulter Elben, Hand Schulter und Rumpf Kopf). Ich kann nachher die winkeln wie in Abbildung 11-3 berechnen von diesen Vektoren. Wenn der Winkel zwischen Schulter Elben und Hand Schulter zwischen 0° und -60° ist, wird die neue zielposition in der Shared Variabel geschrieben. Der Motor hat 256 Positionen die jede 100 Mikropositionen haben. Deswegen ist die Konvertierungsformel: Zielposition = Winkel/90*6400 (für die linke Hande, -6400 für die rechte).

Mit diesen 4 Programmen kann ich der zweite Aufgabe mein Bachelorarbeit verwirklichen und damit der Bachelorarbeit auch beenden.

12 Zusammenfassung und Ausblick

12.1 Zusammenfassung

Ich habe versucht, die Treiber für Kinect mit C++ als DLL für LabVIEW zu programmieren. Jedoch war es wegen der hohen Komplexität der Instrumentenbibliothek-Architektur unmöglich. Die DLLs, die man in LabVIEW einbinden kann, sollen ansi C kompatibel sein. Die Nutzung von Objekte und Namespace ist leider in C unmöglich.

Ausgewählt war dann mit .NET und C# basierte Beispiele zu arbeiten. Ich konnte die .NET Funktion von LabVIEW verwenden um das gewünschte Ergebnis zu erreichen. Ich konnte am Ende mit meiner selbstentwickelten Bibliothek auf die Kamera und den Tiefen-Sensor zugreifen. Mit den Dateien dieses Sensors konnte ich einen Mensch und sein Skelett erkennen und so mein erstes Ziel erreichen.

Ich habe nachher diese Dateien benutzt um einen Steppermotor zu steuern. Ich sollte mit einem ganzen Newtons-Pendel arbeiten, aber das war zur Zeit meiner Bachelorarbeit noch nicht fertig. Allerdings konnte ich mit dem Steppermotor das gewünschte Skript entwickeln und erreichen, dass die Position der Achsen die gleiche ist wie die Armposition des Benutzers.

12.2 Ausblick

In naher Zukunft wird das Skript des Motors auf den Pendeln implementiert. ~~Der~~ Herr Ridlebauch ist für die Pendel verantwortlich. Er hatte auch neue Ideen um die Bewegungen der Benutzer zu messen.

Die Treiber, die ich während der Bachelorarbeit entwickelte, sind auch veröffentlicht worden. Ein anderes Thema ist dann die Unterstützung für diese Treiber nach der Arbeit

weiter zu verwirklichen. National Instruments Entwickler in USA haben zum Beispiel schon damit angefangen weiter zu arbeiten um die Bewegungserkennung zu implementieren. National Instruments Schweden überlegt sich auch ein Beispielsystem für die Messe mit Kinect statt einer normalen Kamera zu wechseln um die Lichtunabhängigkeitsprobleme zu löschen.

Die Treiber und das Gerät werden auch vielleicht von einem anderen Praktikant für ein Projekt weiterbenutzt um ein Roboter zu steuern.

13 Literaturverzeichnis

[1] Gieselman, Hartmut: „Tanz der Skelette – Bewegungserkennung mit Kinect“, C’T Magazin für Computer Technik N°8, März 2011, Heise Zeitschriften Verlag

[2] Tajeddini, Damon: „Minority Report im Fernsehsessel – Windows-Programme mit Gesten steuern“, C’T Magazin für Computer Technik N°11, Mai 2011, Heise Zeitschriften Verlag

[3] BEACON eSpace at Jet Propulsion Laboratory: „Spacecraft Hazard Avoidance Utilizing Structured Light“, NASA, PDF Dokument, http://hdl.handle.net/2014/39375.

[4] Klug, Brian: „Microsoft Kinect: The AnandTech Review“, AnandTech, Webseite, http://www.anandtech.com/show/4057/microsoft-kinect-the-anandtech-review .

[5] Koziolek, Heiko: „Kinect Controlling ABB Robot“, Heiko Koziolek Blog (Principal Scientist at ABB Corporate, Germany), Webseite, http://www.koziolek.de/2011/08/11/kinect-controlling-abb-robot/

[6] Eye Vision Technology: „Palletieren und Depalletieren mit EyeScan 3D und Kinect-Sensor“, EVT, Webseite, http://tinyurl.com/evtkinect

[7] Dumitrescu, Andrei: „Minnesota University Team Adapts Kinect for Medical Use“, Softpedia, Webseite, http://news.softpedia.com/news/Minnesota-University-Team-Adapts-Kinect-for-Medical-Use-189553.shtml .

[8] Maxwell, Ben: „Kinect Will Recognise Sign Language“, Edge, Webseite, http://www.next-gen.biz/news/kinect-will-recognise-sign-language .

14 Abbildungsverzeichnis

15 Anhang

15.1 Sample-Scene.xml

```
<OpenNI>
    <Licenses>
        <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
    </Licenses>
    <Log writeToConsole="true" writeToFile="false">
        <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
        <LogLevel value="3"/>
        <Masks>
            <Mask name="ALL" on="false"/>
        </Masks>
        <Dumps>
        </Dumps>
    </Log>
    <ProductionNodes>
        <Node type="Depth">
            <Configuration>
                <Mirror on="true"/>
            </Configuration>
        </Node>
        <Node type="Scene" />
    </ProductionNodes>
</OpenNI>
```

15.2 Sample-Tracking.xml

```
<OpenNI>
    <Licenses>
        <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
    </Licenses>
    <Log writeToConsole="true" writeToFile="false">
        <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
        <LogLevel value="3"/>
        <Masks>
            <Mask name="ALL" on="false"/>
        </Masks>
        <Dumps>
        </Dumps>
    </Log>
    <ProductionNodes>
        <Node type="Depth">
            <Configuration>
                <Mirror on="true"/>
            </Configuration>
        </Node>
        <Node type="Gesture" />
        <Node type="Hands" />
    </ProductionNodes>
</OpenNI>
```

15.3 Sample-User.xml

```
<OpenNI>
    <Licenses>
        <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
    </Licenses>
    <Log writeToConsole="false" writeToFile="false">
        <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
        <LogLevel value="3"/>
        <Masks>
            <Mask name="ALL" on="true"/>
        </Masks>
        <Dumps>
        </Dumps>
    </Log>
    <ProductionNodes>
        <Node type="Image" name="Image1">
            <Configuration>
                <MapOutputMode xRes="640" yRes="480" FPS="30"/>
                <Mirror on="true"/>
            </Configuration>
        </Node>
        <Node type="Depth" name="Depth1">
            <Configuration>
                <MapOutputMode xRes="640" yRes="480" FPS="30"/>
                <Mirror on="true"/>
            </Configuration>
        </Node>
        <Node type="User" name="User1"/>
        <!--
        <Node type="Audio" name="Audio1">
        </Node>
        -->
    </ProductionNodes>
</OpenNI>
```

15.4 dllmain.cpp

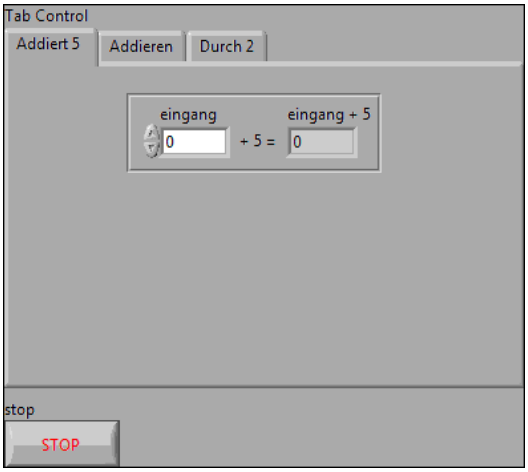
```
1  // dllmain.cpp : Defines the entry point for the DLL application.
2  #include "stdafx.h"
3
4  BOOL APIENTRY DllMain( HMODULE hModule,
5                         DWORD  ul_reason_for_call,
6                         LPVOID lpReserved
7                       )
8  {
9     switch (ul_reason_for_call)
10    {
11       case DLL_PROCESS_ATTACH:
12       case DLL_THREAD_ATTACH:
13       case DLL_THREAD_DETACH:
14       case DLL_PROCESS_DETACH:
15       break;
16    }
17    return TRUE;
18 }
19
20 extern "C" int __stdcall Addiertfuenf(int af);
21 extern "C" float __stdcall Addieren (float a1, float a2);
22 extern "C" float __stdcall Durchzwei (float& dz1, float& dz2);
```

15.5 CPP Test to DLL 1.cpp

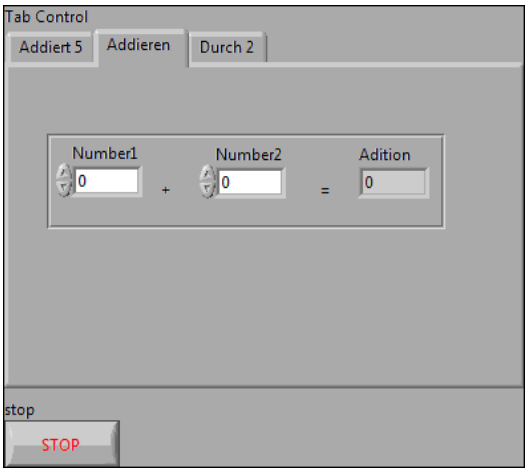
```
1  /* Cpp Test to DLL 1.cpp : Defines the exported functions for the DLL appli-
2  cation.*/
3
4  #include "stdafx.h"
5
6  extern "C" int __stdcall Addiertfuenf (int af)
7  {
8     af = af + 5;
9     return af;
10 }
11
12
13 extern "C" float __stdcall Addieren (float a1, float a2)
14 {
15    float summe = a1 + a2;
16    return summe;
17 }
18
19
20 extern "C" float __stdcall Durchzwei (float& dz1, float& dz2)
21 {
22    dz1 = dz1/2;
23    dz2 = dz2/2;
24    float summe = dz1+dz2;
25    return summe;
26 }
```

15.6 LabVIEW Test Programm: Front Panel

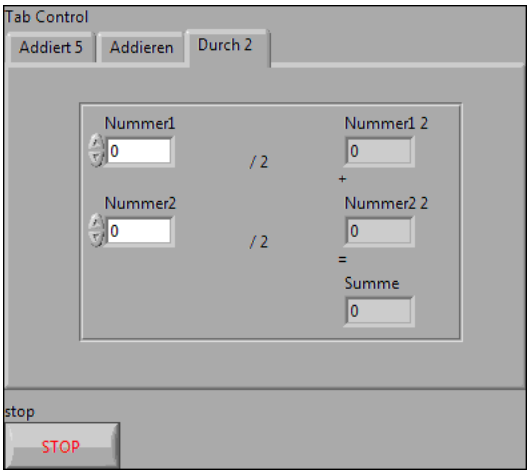
Tab1:



Tab2:



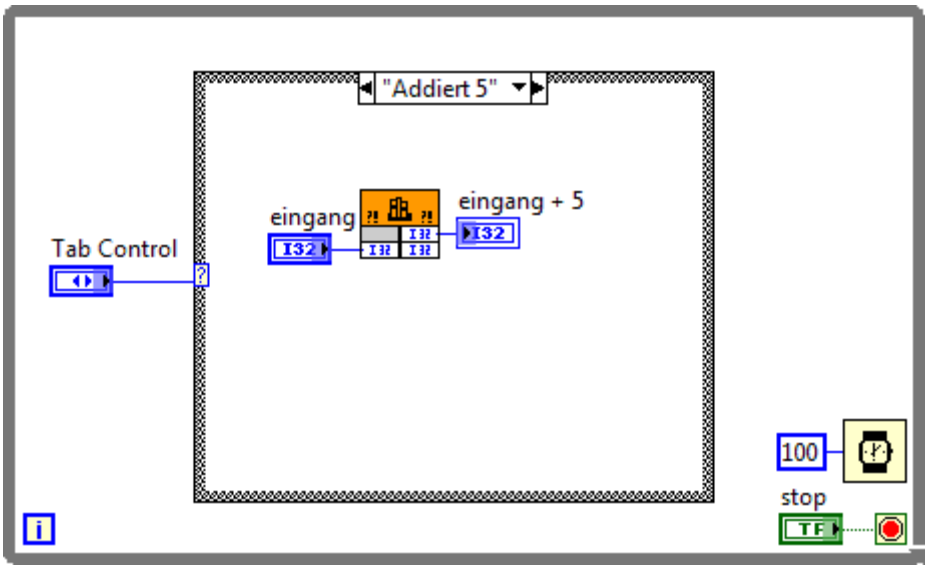
Tab3:



15.7 LabVIEW Test Programm : Block Diagramm

15.7.1 : Addiert 5

Code:



Call Library Function Eingeschärften:

Function name
Addiertfuenf

Function prototype
int32_t Addiertfuenf(int32_t eingang);

Calling convention
stdcall (WINAPI)
C

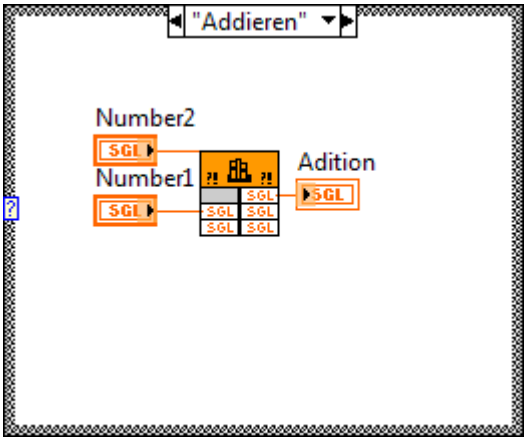
Thread
Run in UI thread
Run in any thread

Current parameter
Name eingang + 5
Type Numeric
Constant
Data type Signed 32-bit Integer

Current parameter
Name eingang
Type Numeric
Constant
Data type Signed 32-bit Integer
Pass Value

15.7.2 : Addieren

Code:



Call Library Function Eingeschärften:

Function name

Addieren

Function prototype

float Addieren(float Number1, float Number2);

Calling convention

- stdcall (WINAPI)
- C

Thread

- Run in UI thread
- Run in any thread

Current parameter

Name: Adition
Type: Numeric
Constant:
Data type: 4-byte Single

Current parameter

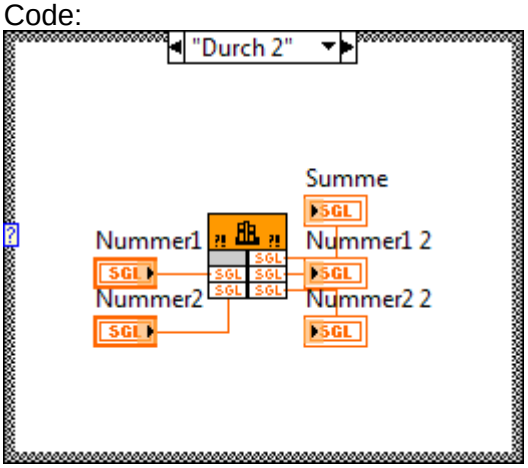
Name: Number1
Type: Numeric
Constant:
Data type: 4-byte Single
Pass: Value

Current parameter

Name: Number2
Type: Numeric
Constant:
Data type: 4-byte Single
Pass: Value

15.7.3 : Durch Zwei

Code:



Call Library Function Eingeschärften:

Function name: Durchzwei

Function prototype: float Durchzwei(float *Nummer1, float *Nummer2);

Calling convention: stdcall (WINAPI) / C

Thread: Run in UI thread / Run in any thread

Current parameter

Name	Summe
Type	Numeric
Constant	
Data type	4-byte Single

Current parameter

Name	Nummer1
Type	Numeric
Constant	
Data type	4-byte Single
Pass	Pointer to Value

Current parameter

Name	Nummer2
Type	Numeric
Constant	
Data type	4-byte Single
Pass	Pointer to Value

15.8 C++ Beispiel Programme mit Objekt

15.8.1 Definition meines Objekts

```
1  // Object Dummi.h
2
3  #include "stdafx.h"
4  #include <string>
5  #ifndef __DUMMI_H_INCLUDED__
6  #define __DUMMI_H_INCLUDED__
7
8  class Dummi {
9  public:
10
11    void Init(void)
12       {
13       _name_dummi = "aaa";
14       _valuep = 2;
15       }
16
17    float Addieren (float z1, float z2)
18    {
19       float a = z1 + z2;
20       return a;
21    }
22
23    float Durchzwei (float z1)
24    {
25       float b = z1/2;
26       return b;
27    }
28
29    void Setname (char* name_dummi)
30    {
31       _name_dummi = name_dummi;
32    }
33
34    char* Getname (void)
35    {
36       return _name_dummi;
37    }
```

```
38
39      void Setvalue(float valuep)
40      {
41          _valuep = valuep;
42      }
43
44      float Getvalue(void)
45      {
46          return _valuep;
47      }
48
49  private:
50      char* _name_dummi;
51      float _valuep;
52  };
53
54  #endif /* __DUMMI_H_INCLUDED__ */
```

15.8.2 Definition der Funktionen meiner DLL

```
1  // Projet DLL C++.cpp : Defines the exported functions for the DLL application.
2  //
3
4  #include "stdafx.h"
5
6
7  extern "C" Dummi * __stdcall CreateObject(void)
8  {
9     Dummi dum;
10    dum.Init();
11    return &dum;
12 }
13
14 extern "C" float __stdcall Addition(Dummi * d1, float a, float b)
15 {
16    return d1->Addieren(a, b);
17 }
18
19 extern "C" void __stdcall GiveName(Dummi * d1)
20 {
21    d1->Setname("myobject");
22 }
23
24 extern "C" char* __stdcall GetName(Dummi * d1)
25 {
26    return d1->Getname();
27 }
28
29 extern "C" void __stdcall GiveVal(Dummi * d1, float a)
30 {
31    d1->Setvalue(a);
32 }
33
34 extern "C" float __stdcall GetVal(Dummi * d1)
35 {
36    return d1->Getvalue();
37 }
```

15.9 Testprogramm um die Dateien der Kamera zu lesen

```
1  // OpenNI Test.cpp : Defines the entry point for the console application.
2  //
3  #include "stdafx.h"
4  #include "XnCppWrapper.h"
5
6  using namespace xn;
7
8  int _tmain(int argc, _TCHAR* argv[])
9  {
10    // Initialize the context
11    XnStatus nRetVal = XN_STATUS_OK;
12    Context context;
13    nRetVal = context.Init();
14
15    // Create a depth generator
16    DepthGenerator depth;
17    nRetVal = depth.Create(context);
18
19    // Set it to VGA maps at 30 FPS
20    XnMapOutputMode mapMode;
21    mapMode.nXRes = XN_VGA_X_RES;
22    mapMode.nYRes = XN_VGA_Y_RES;
23    mapMode.nFPS = 30;
24    nRetVal = depth.SetMapOutputMode(mapMode);
25
26    // Start generating
27    nRetVal = context.StartGeneratingAll();
28
29    // Calculate index of middle pixel
30    XnUInt32 nMiddleIndex =
31                          XN_VGA_X_RES * XN_VGA_Y_RES/2 + // start of middle line
32                          XN_VGA_X_RES/2; // middle of this line
33
34    while (1)
35    {
36       // Update to next frame
37       nRetVal = context.WaitOneUpdateAll(depth);
38       const XnDepthPixel* pDepthMap = depth.GetDepthMap();
```

```
39          printf("Middle pixel is %u millimeters away  -  Middle index is %u\n", pDep
40      }
41
42      // Clean up
43      context.Shutdown();
44  }
```

15.10 TestDLL um die Tiefe von die Kamera auszulesen

```
1  // Depht DLL.cpp : Defines the exported functions for the DLL application.
2  //
3
4  #include "stdafx.h"
5
6  XnStatus nRetVal = XN_STATUS_OK;
7
8
9  extern "C" xn::Context * __stdcall CreateObject(void)
10 {
11    static xn::Context context;
12    nRetVal = context.Init();
13    return &context;
14 }
15
16 extern "C" xn::DepthGenerator * __stdcall InitialiseDepht(xn::Context * con-
17     text)
18 {
19    static xn::DepthGenerator depth;
20    nRetVal = depth.Create(*context);
21    return &depth;
22 }
23
24 extern "C" XnMapOutputMode * __stdcall InitialiseMap(xn::Context * context,
25     xn::DepthGenerator * depth)
26 {
27    static  XnMapOutputMode mapMode;
28    mapMode.nXRes = XN_VGA_X_RES;
29    mapMode.nYRes = XN_VGA_Y_RES;
30    mapMode.nFPS = 30;
31    nRetVal = depth->SetMapOutputMode(mapMode);
32
33    nRetVal = context->StartGeneratingAll();
34
35    return &mapMode;
36
37 }
38
```

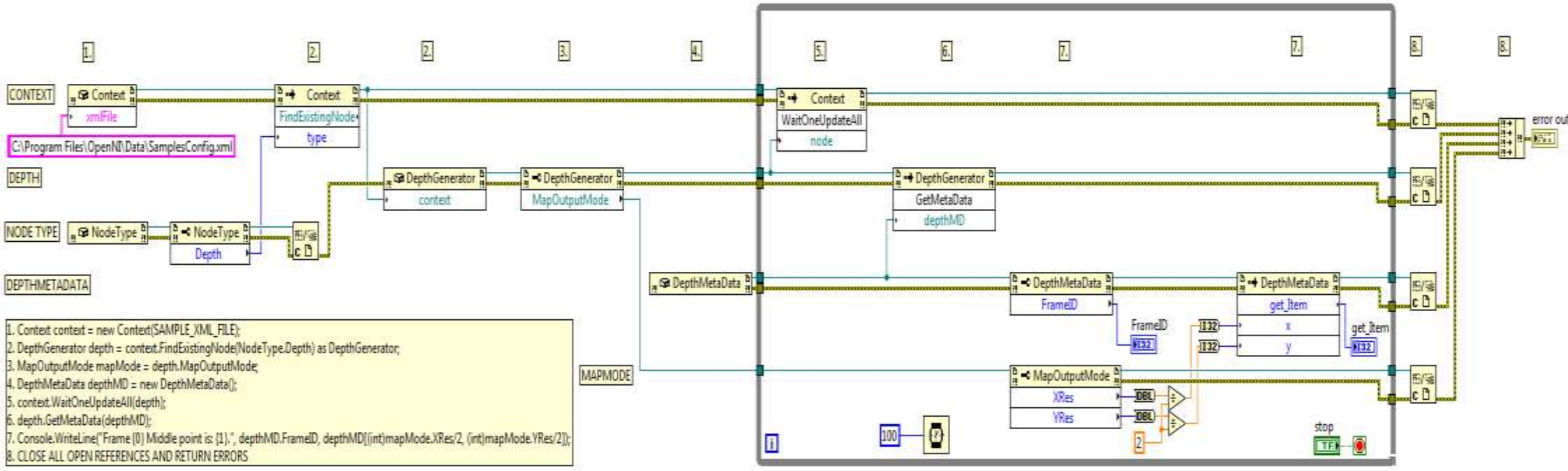
```
39  extern "C" int __stdcall DepthVal(xn::Context * context, xn::DepthGenerator *
40      depth)
41  {
42     nRetVal = context->WaitOneUpdateAll(*depth);
43
44     const XnDepthPixel* pDepthMap = depth->GetDepthMap();
45     return pDepthMap[15600];
46  }
47
48  extern "C" int __stdcall Shutdown(xn::Context * context)
49  {
50     context->Shutdown();
51     return 1;
52  }
```

15.11 Das Beispielprogramm aus C# geschrieben mit .NET Assembies

```
1  // Program.cs
2
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using OpenNI;
8
9  namespace SimpleRead.net
10 {
11    class Program
12    {
13       static void Run()
14       {
15          string SAMPLE_XML_FILE = @"../../../Data/SamplesConfig.xml";
16
17          Context context = new Context(SAMPLE_XML_FILE);
18
19          DepthGenerator depth = context.FindExistingNode(NodeType.Depth) as
20     DepthGenerator;
21          if (depth == null)
22          {
23             Console.WriteLine("Sample must have a depth generator!");
24             return;
25          }
26
27          MapOutputMode mapMode = depth.MapOutputMode;
28
29          DepthMetaData depthMD = new DepthMetaData();
30
31          Console.WriteLine("Press any key to stop...");
32
33          while (!Console.KeyAvailable)
34          {
35             context.WaitOneUpdateAll(depth);
36
37             depth.GetMetaData(depthMD);
38
```

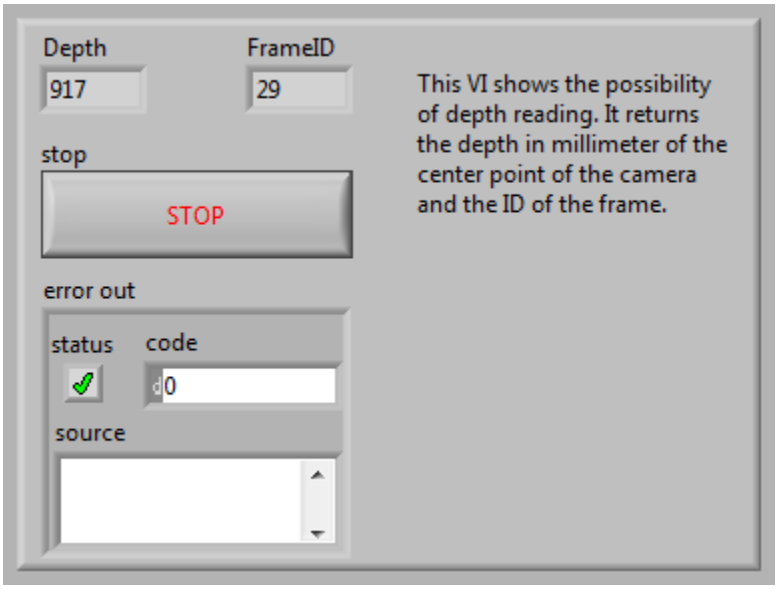
```
39              Console.WriteLine("Frame {0} Middle point is: {1}.",
40      depthMD.FrameID, depthMD[(int)mapMode.XRes/2, (int)mapMode.YRes/2]);
41          }
42       }
43
44       static void Main(string[] args)
45       {
46          try
47          {
48             Run();
49          }
50          catch (System.Exception ex)
51          {
52             Console.WriteLine("Error: {0}", ex.Message);
53          }
54       }
55    }
56 }
57
```

15.12 Das LabVIEW Programm mit .NET Assemblies

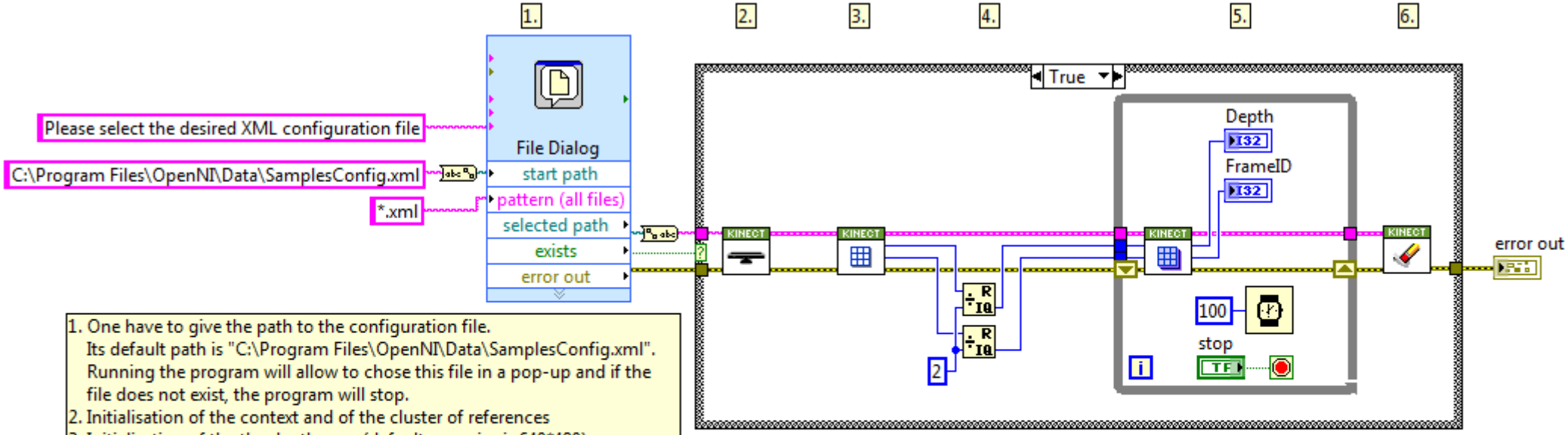


15.13 TestDepth.vi

15.13.1 Front Panel



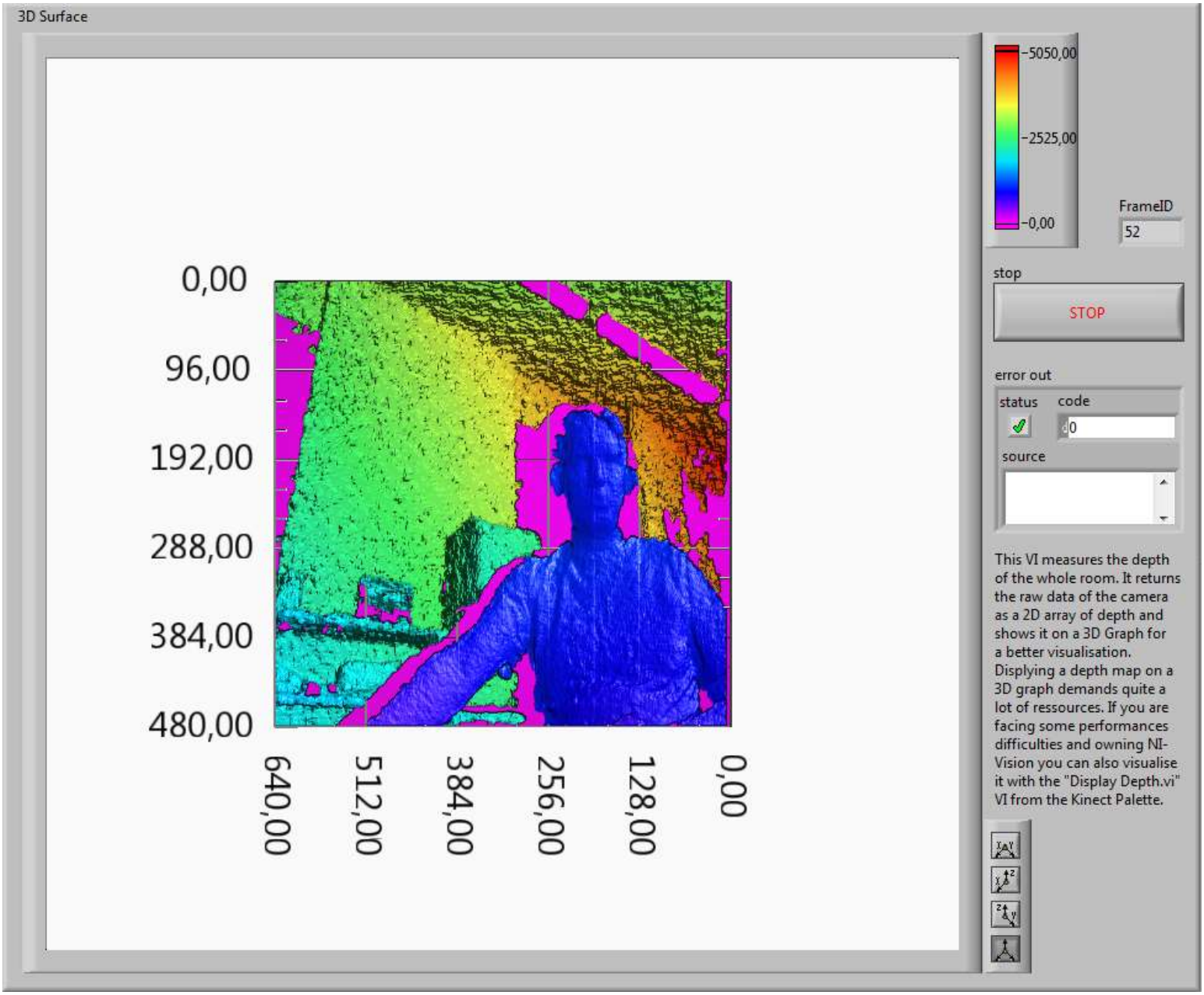
15.13.2 Block Diagramm



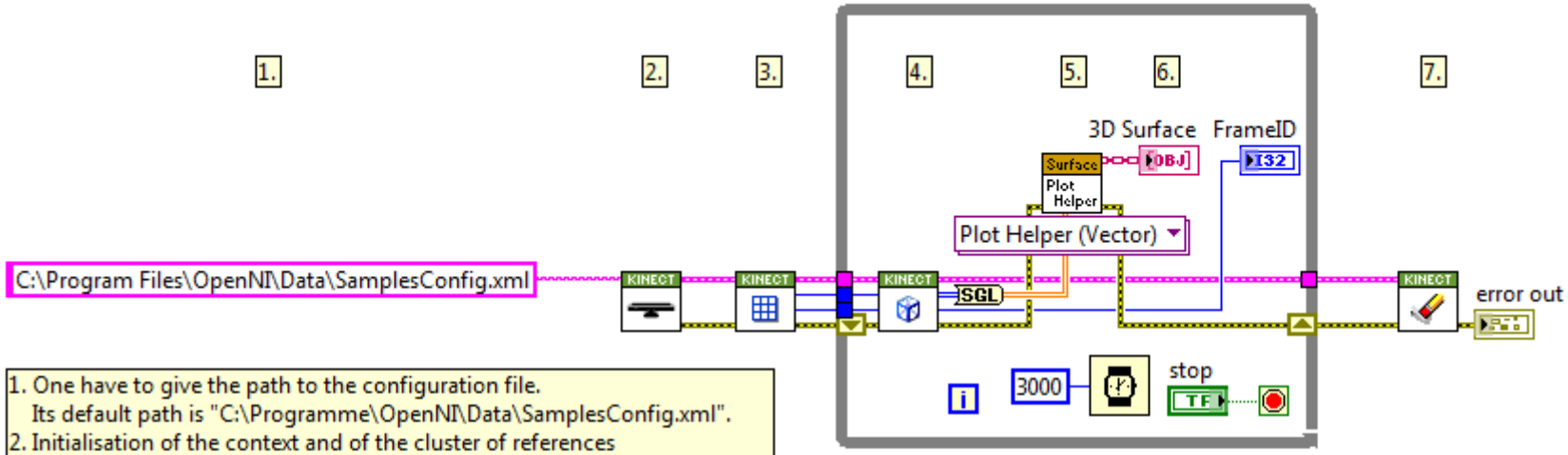
1. One have to give the path to the configuration file.
   Its default path is "C:\Program Files\OpenNI\Data\SamplesConfig.xml".
   Running the program will allow to chose this file in a pop-up and if the file does not exist, the program will stop.
2. Initialisation of the context and of the cluster of references
3. Initialisation of the the depth map (default map size is 640*480)
4. Calculation of the coordinates of the middle point of the map. This coordinate is calculated by dividing the x and the y resolution by 2.
5. Returns every 100 ms the depth of the middle point.
   You can test moving your hand in front of the camera and see it changes.
   /!\ Beware: The camera can only calculate the depth of an object further away than 50cm. Under 50cm it will return a depth of 0. The same will happend with a distance over 6,50m.
6. Shutdown the current context and close all references

15.14 TestMap.vi

15.14.1 Front Panel



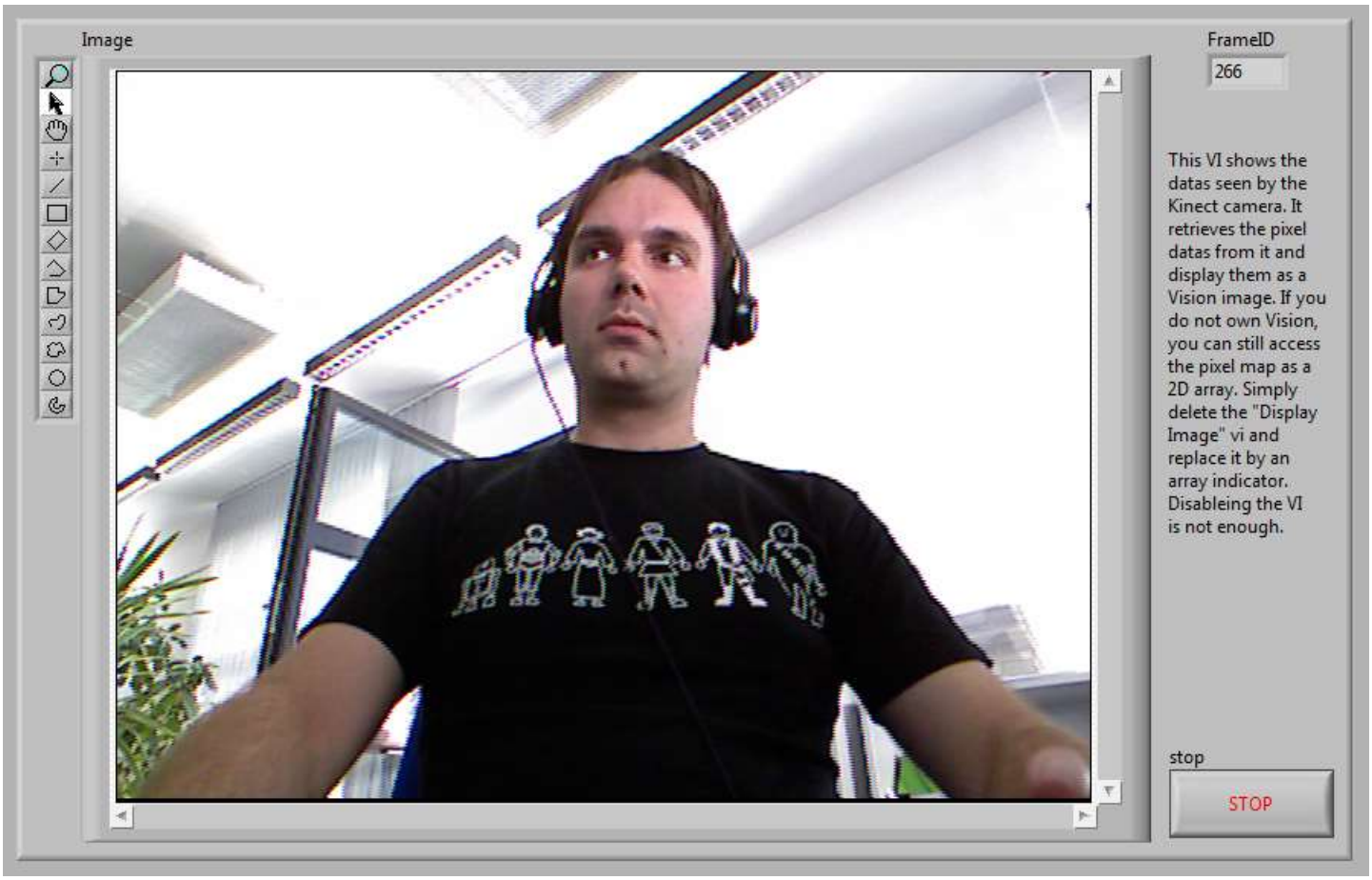
15.14.2 Block Diagramm



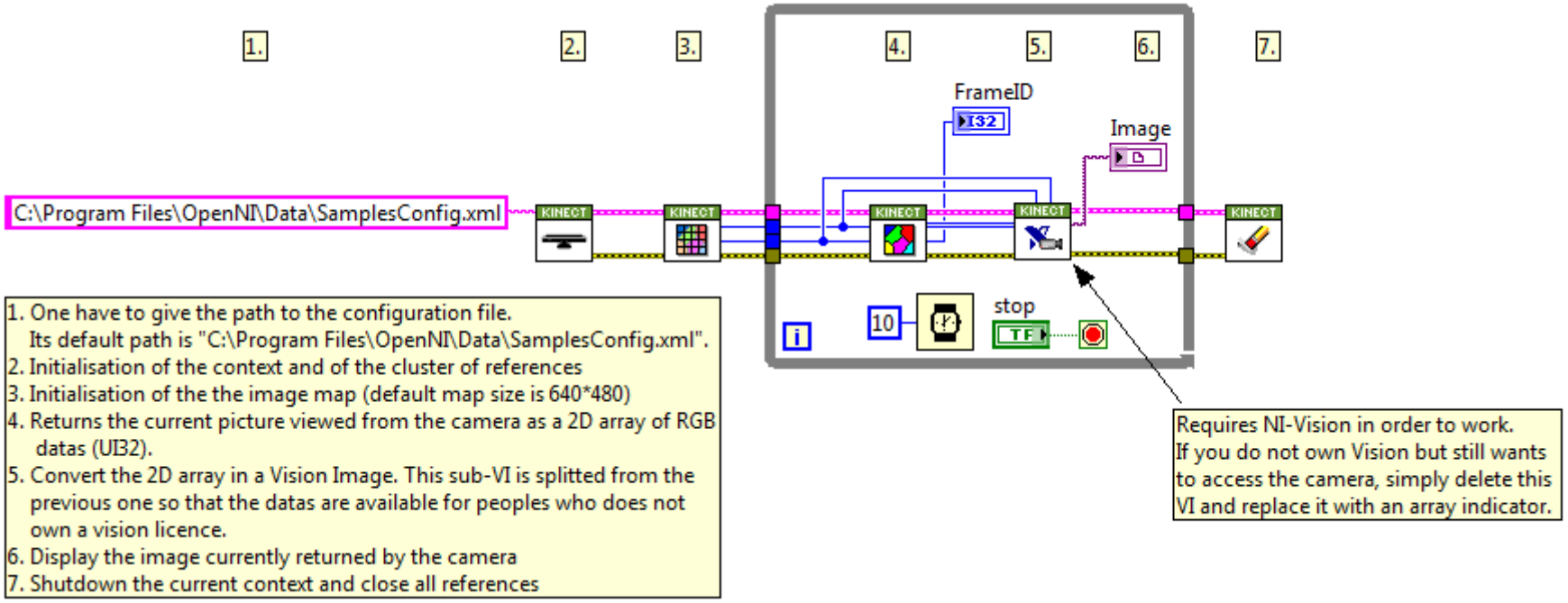
1. One have to give the path to the configuration file.
   Its default path is "C:\Programme\OpenNI\Data\SamplesConfig.xml".
2. Initialisation of the context and of the cluster of references
3. Initialisation of the the depth map (default map size is 640*480)
4. Calculation of the depth map as a 2D array of depth (mm). This VI needs the resolution of the map in order to work.
   /!\ Beware: The camera can only calculate the depth of an object further away than 50cm. Under 50cm it will return a depth of 0. The same will happend with a distance over 6,50m.
5. Convert the 2D array in a 3D Surface Graph ion order to visualise it.
6. Display of the graph of depth. This operation is demanding performances but not necessary is one yhish to york with depth. It is only for a showing purpose in the examples. The 3s timing in the loop is due to it.
7. Shutdown the current context and close all references

15.15 TestImage.vi

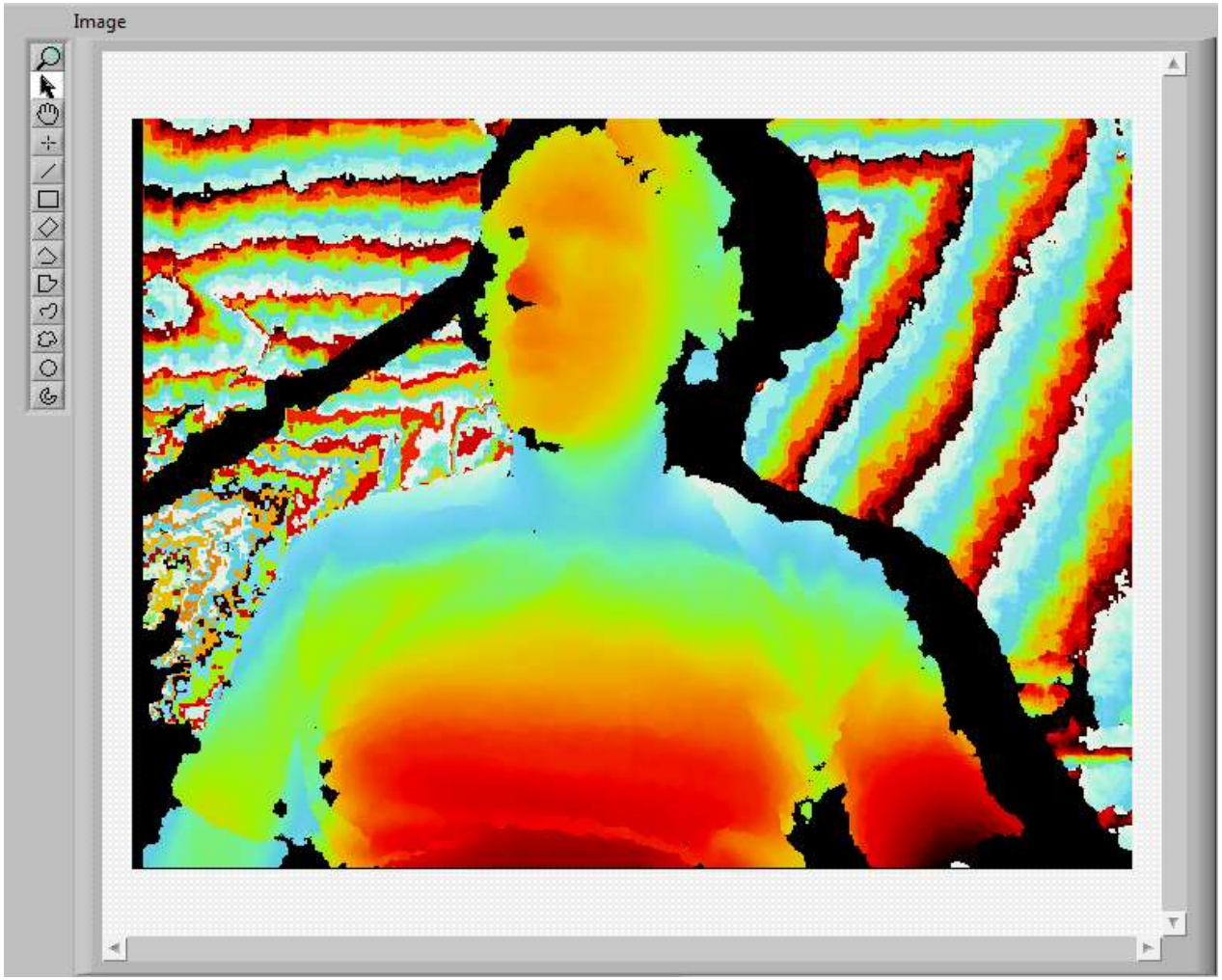
15.15.1 Front Panel

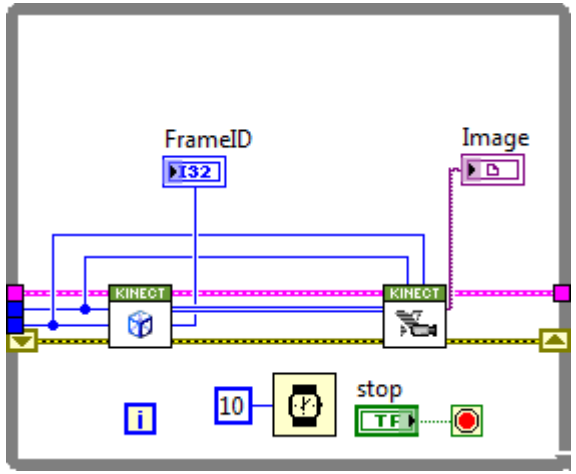


15.15.2 Block Diagramm



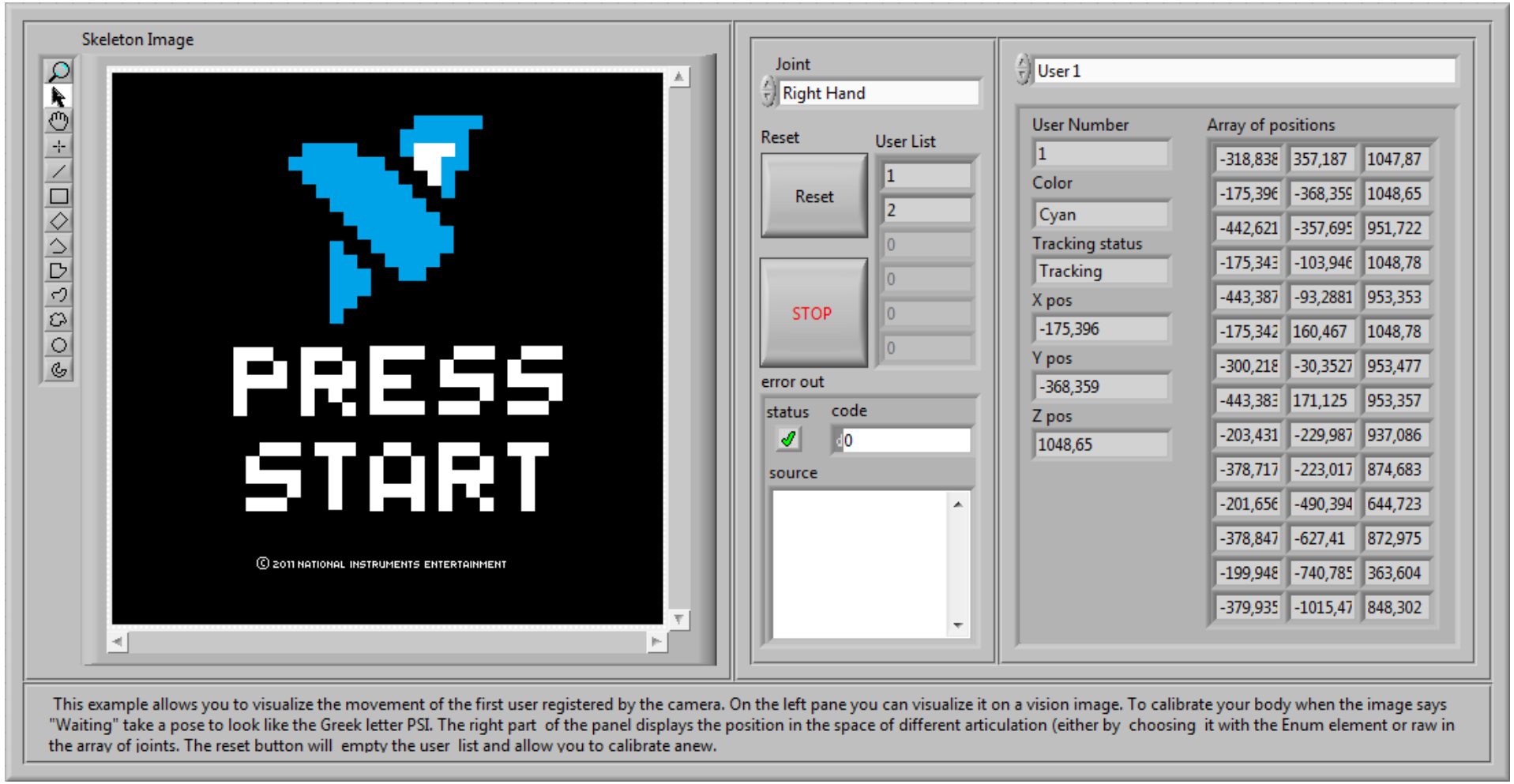
15.16 Visualisierung der Tiefe von TestMap.vi mit dem Unterprogramm, das Vision benutzt, statt 3D Graph



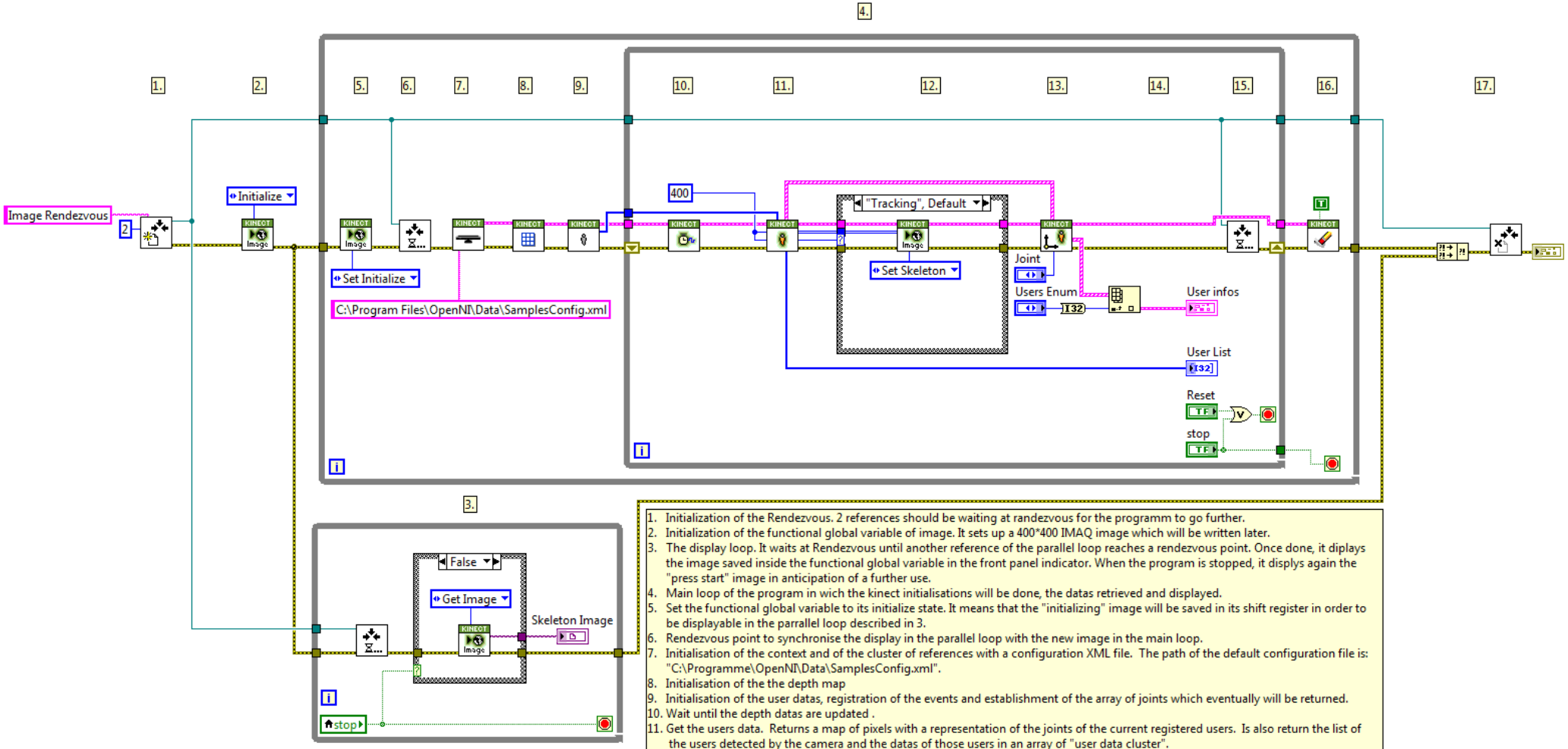


15.17 TestMultipleUserTracking.vi

15.17.1 Front Panel



15.17.2 Block Diagramm



1. Initialization of the Rendezvous. 2 references should be waiting at randezvous for the programm to go further.
2. Initialization of the functional global variable of image. It sets up a 400*400 IMAQ image which will be written later.
3. The display loop. It waits at Rendezvous until another reference of the parallel loop reaches a rendezvous point. Once done, it diplays the image saved inside the functional global variable in the front panel indicator. When the program is stopped, it displys again the "press start" image in anticipation of a further use.
4. Main loop of the program in wich the kinect initialisations will be done, the datas retrieved and displayed.
5. Set the functional global variable to its initialize state. It means that the "initializing" image will be saved in its shift register in order to be displayable in the parrallel loop described in 3.
6. Rendezvous point to synchronise the display in the parallel loop with the new image in the main loop.
7. Initialisation of the context and of the cluster of references with a configuration XML file. The path of the default configuration file is: "C:\Programme\OpenNI\Data\SamplesConfig.xml".
8. Initialisation of the the depth map
9. Initialisation of the user datas, registration of the events and establishment of the array of joints which eventually will be returned.
10. Wait until the depth datas are updated .
11. Get the users data. Returns a map of pixels with a representation of the joints of the current registered users. Is also return the list of the users detected by the camera and the datas of those users in an array of "user data cluster".
12. Set the image to display in the functional global variable. If users are neither tracked nor calibrating, the "waiting" image will be set. If one user is calibrating but none are tracked, the "calibrating" image will be set. At last, if at least one user is tracked, the pixel map returned in 11. will be set as current image.
13. Saves the joint position on the 3 axes of the whished joint inside the array of user info cluster.
14. Redeem the user informations of the whished user and display them as well as the user list.
15. Wait at the rendezvous to duisplay the new informations before to start the loop over again.
16. Shutdown the current context and close all references.
17. Closes the rendezvous reference.

15.18 Zustandsbilder:

„Set Start“ Zustand:

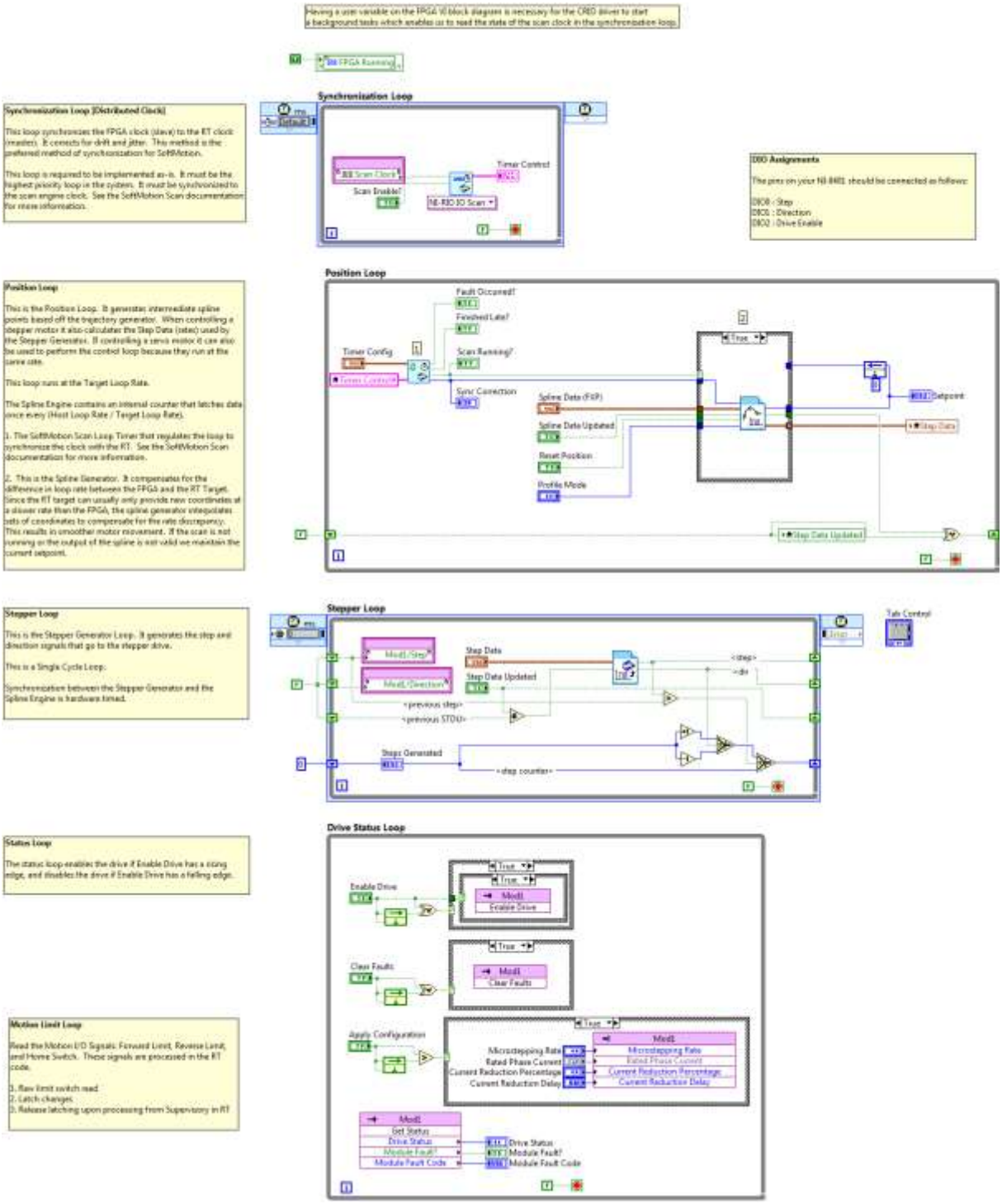
„Set Initialize“ Zutand:

„Set Wait“ Zustand:

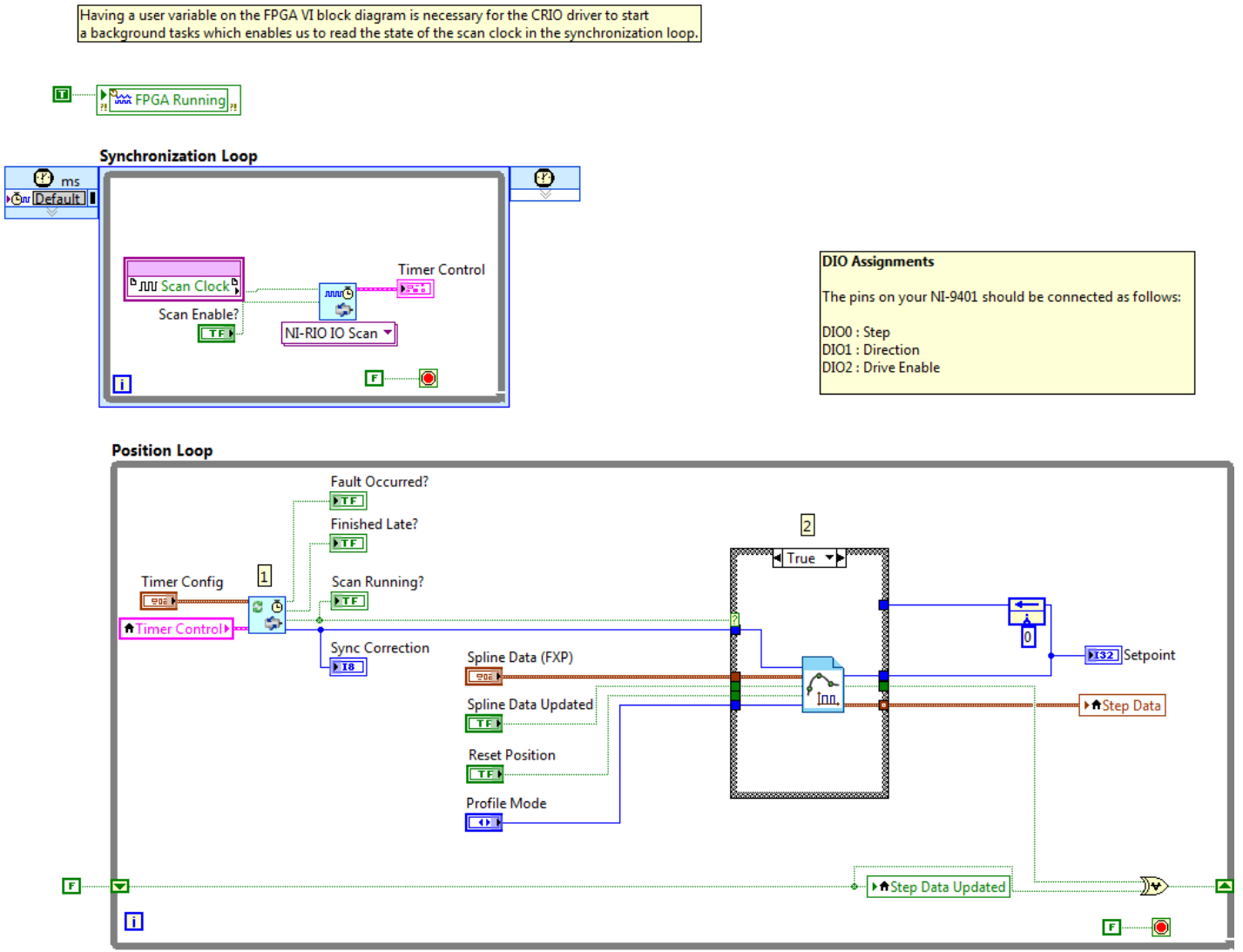
„Set Calibrate“ Zustand:

15.19 Der FPGA VI

15.19.1 Ausblick



15.19.2 Teil 1 (Synchronization Loop und Position Loop - oben)



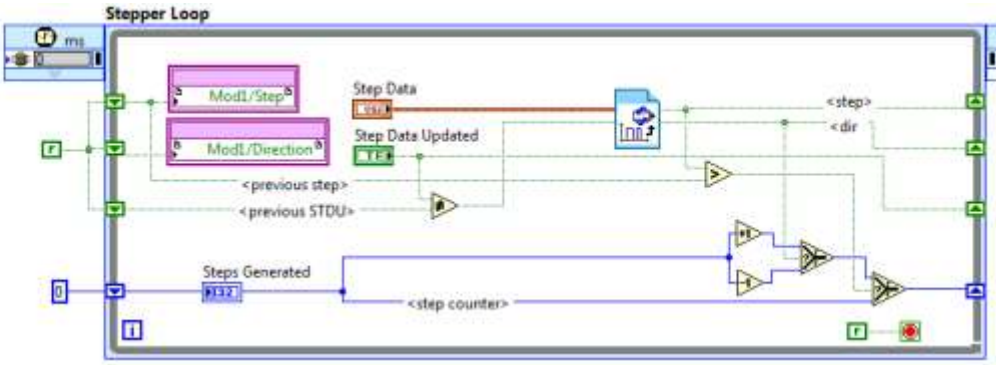
15.19.3 Teil 2 (Stepper Loop und Drive Status Loop unten)

Stepper Loop

This is the Stepper Generator Loop. It generates the step and direction signals that go to the stepper drive.

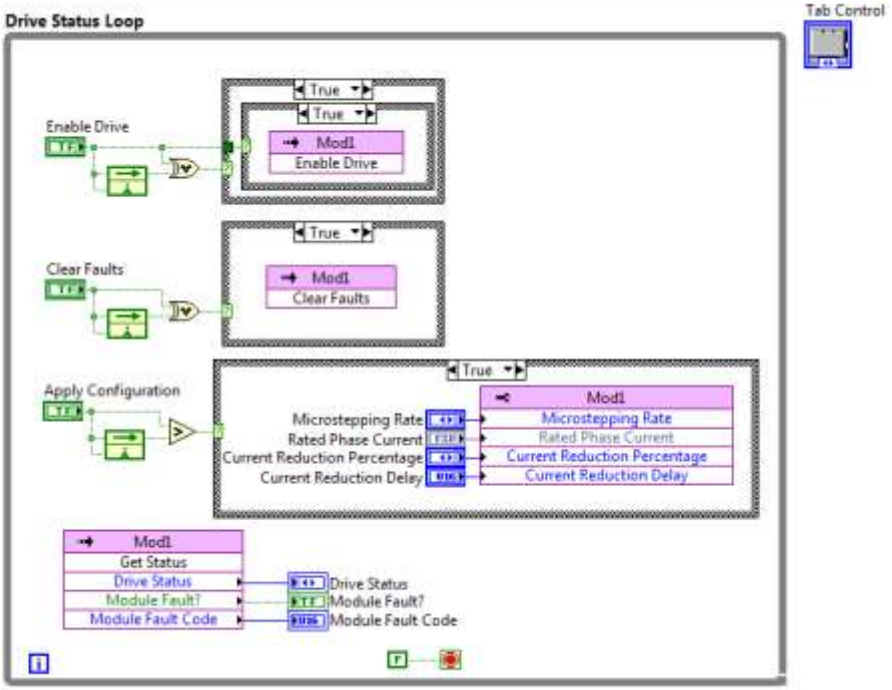
This is a Single Cycle Loop.

Synchronization between the Stepper Generator and the Spline Engine is hardware timed.



Status Loop

The status loop enables the drive if Enable Drive has a rising edge, and disables the drive if Enable Drive has a falling edge.



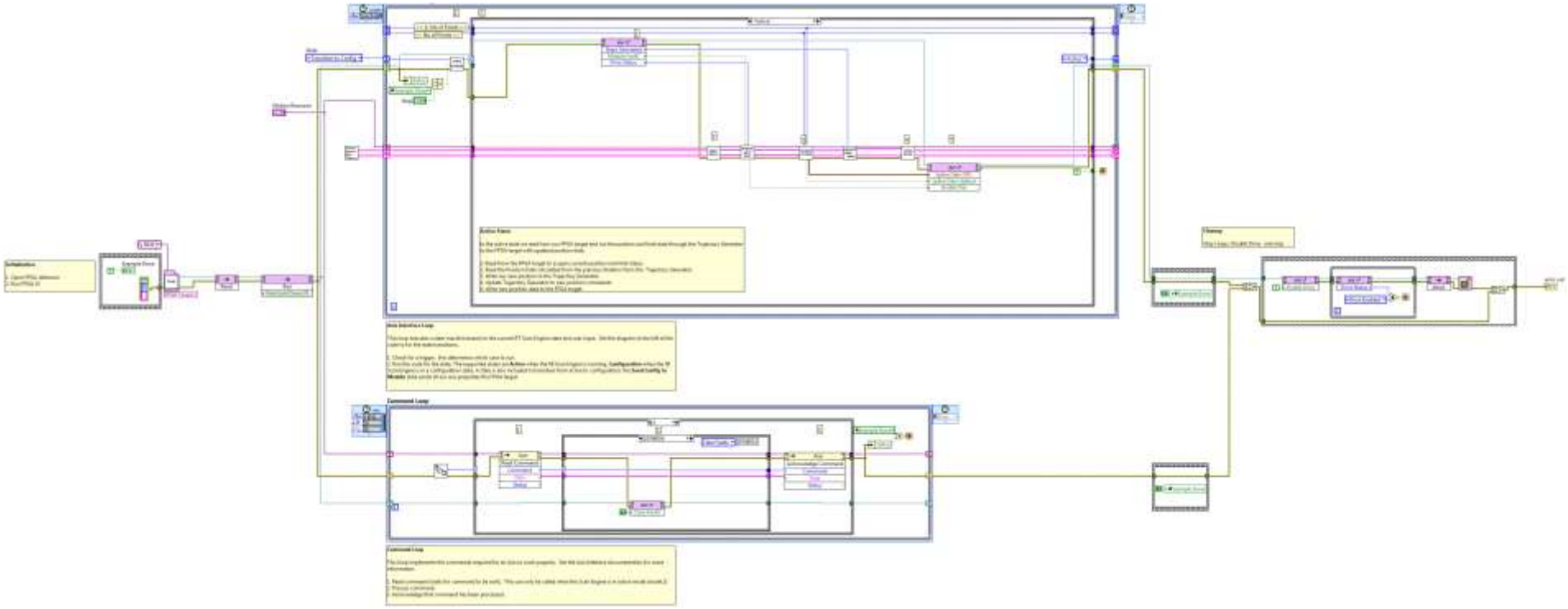
Motion Limit Loop

Read the Motion I/O Signals: Forward Limit, Reverse Limit, and Home Switch. These signals are processed in the RT code.

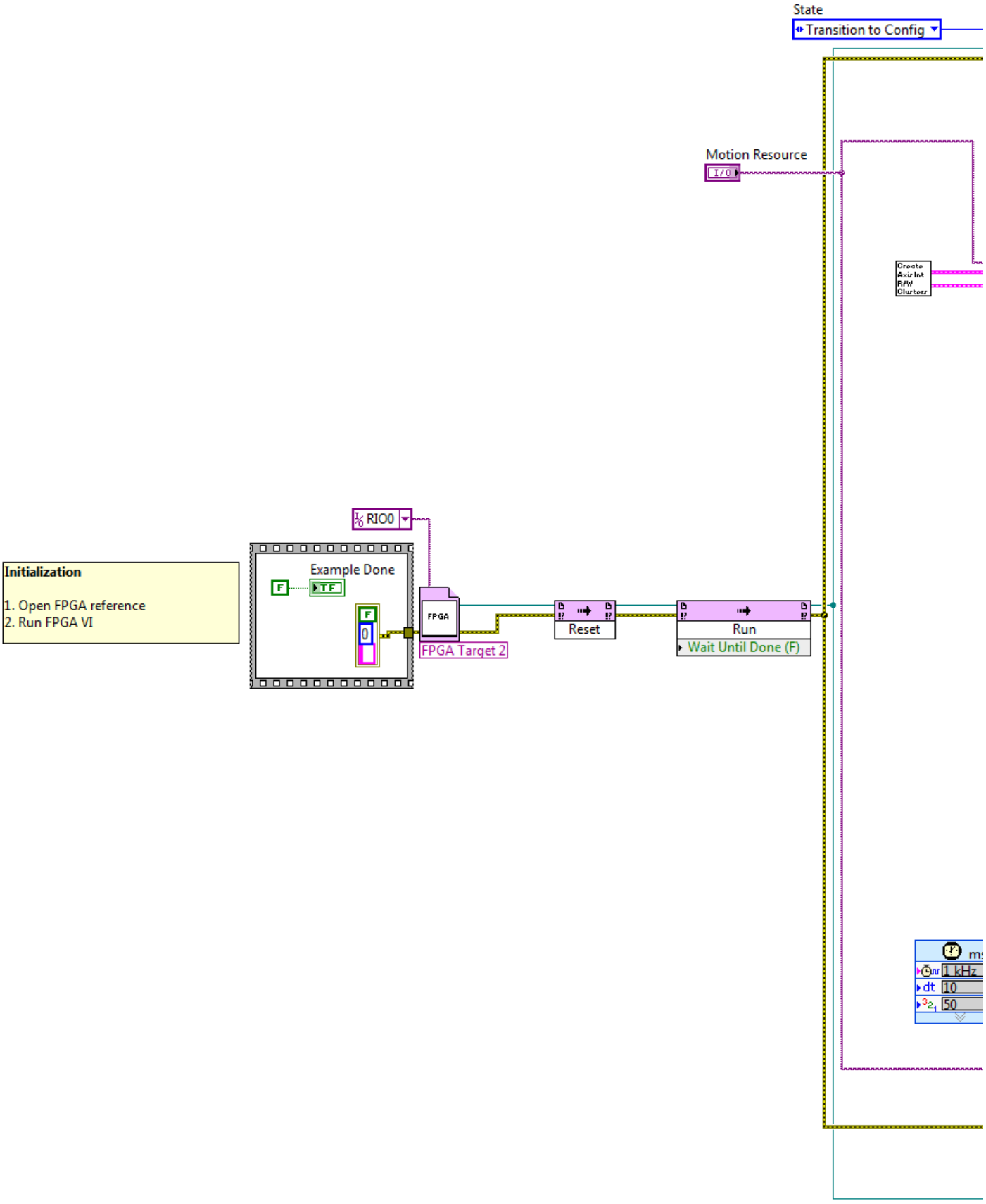
1. Raw limit switch read
2. Latch changes
3. Release latching upon processing from Supervisory in RT

15.20 Motion Funktion Block: StepperDrive.vi

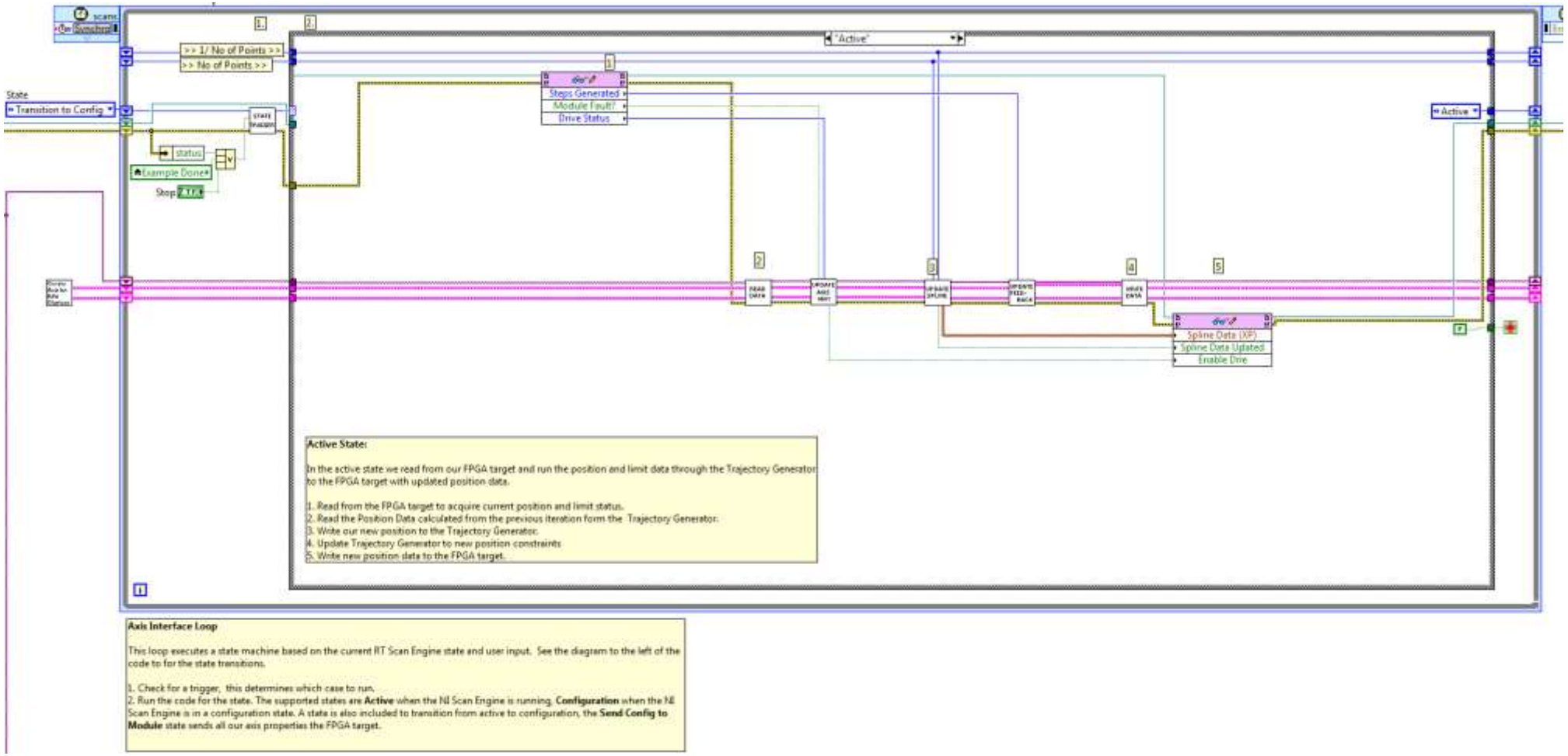
15.20.1 Ausblick



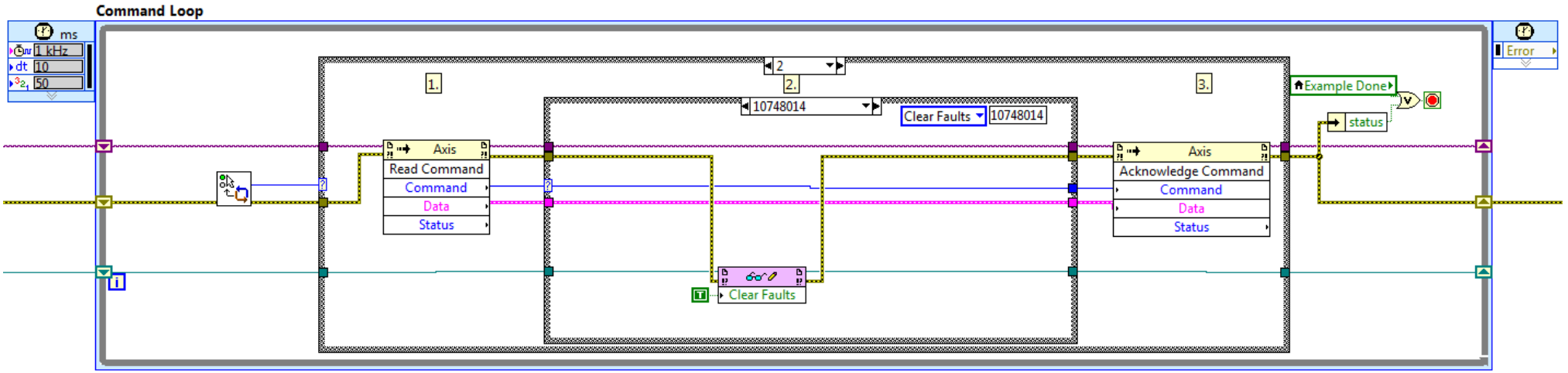
15.20.2 Teil 1 (Initialization – links)



15.20.3 Teil 2 (Axis Interface Loop – oben)



15.20.4 Teil 3 (Command Loop – unten)

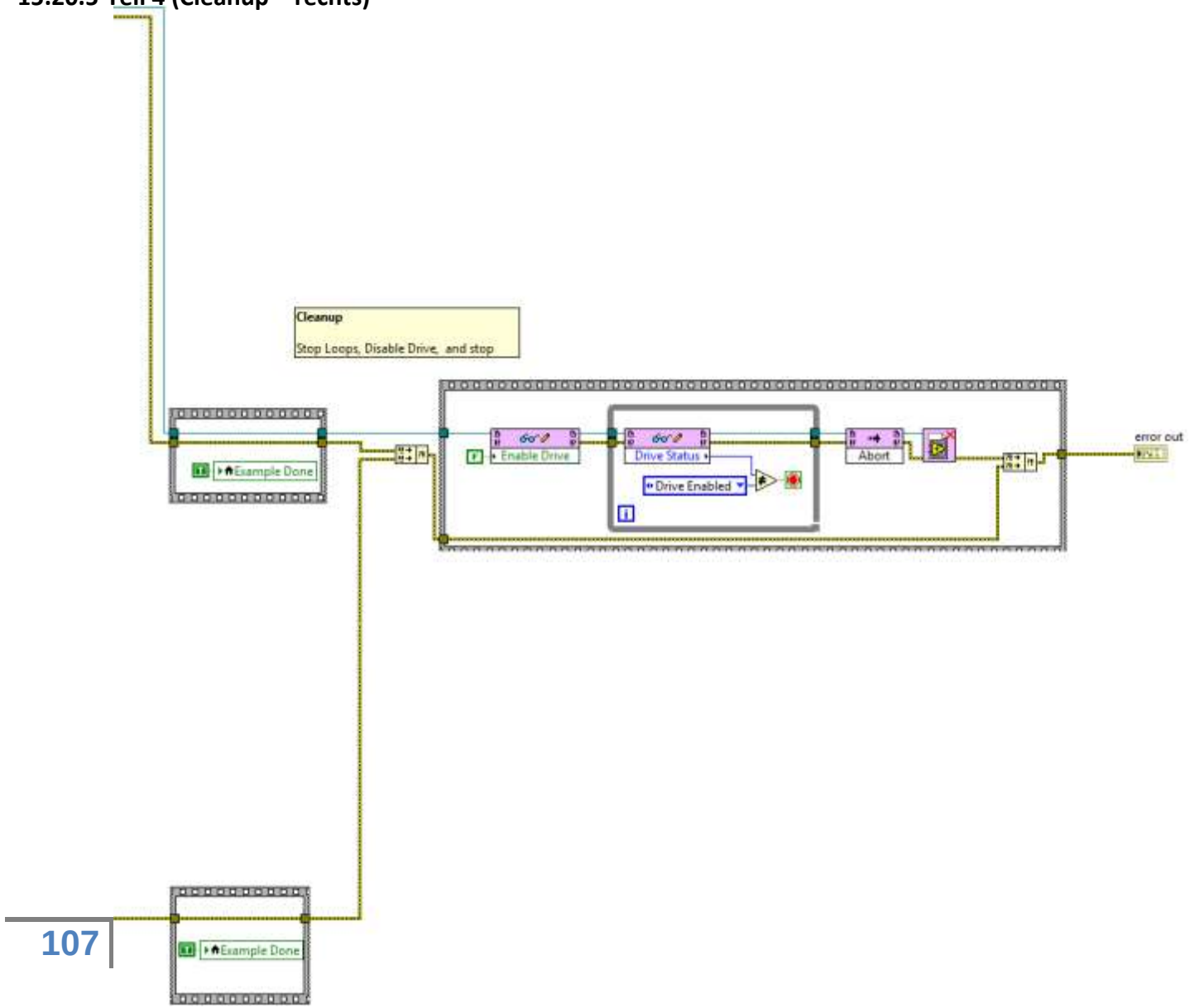


Command Loop

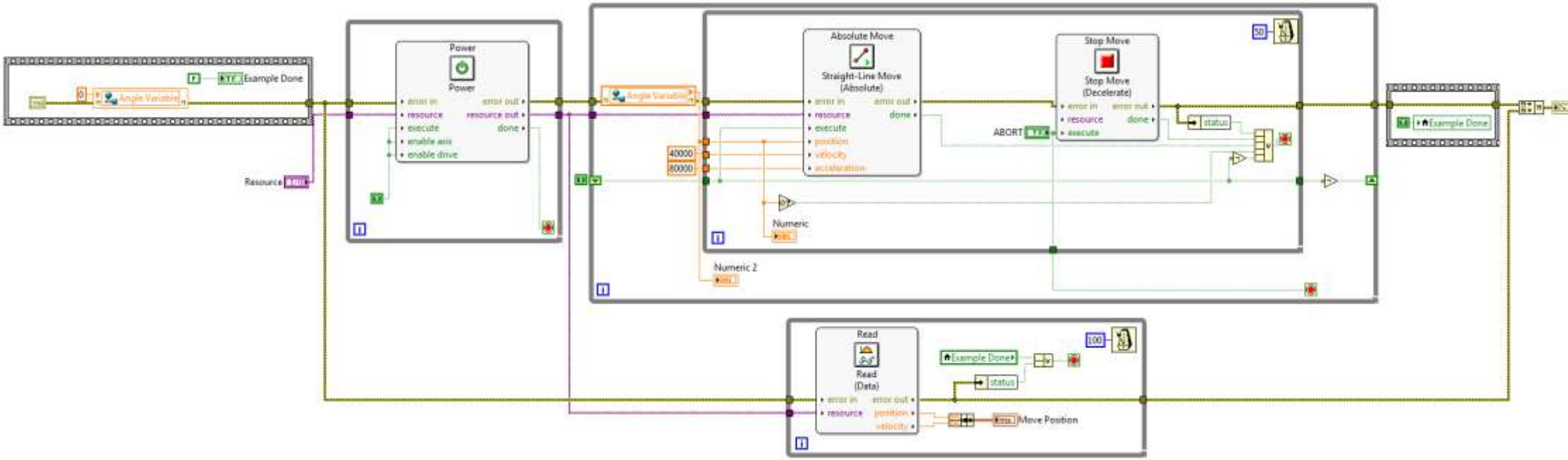
This loop implements the commands required for an Axis to work properly. See the Axis Interface documentation for more information.

1. Read command (waits for command to be sent). This can only be called when the Scan Engine is in active mode (mode 2)
2. Process command.
3. Acknowledge that command has been processed.

15.20.5 Teil 4 (Cleanup – rechts)

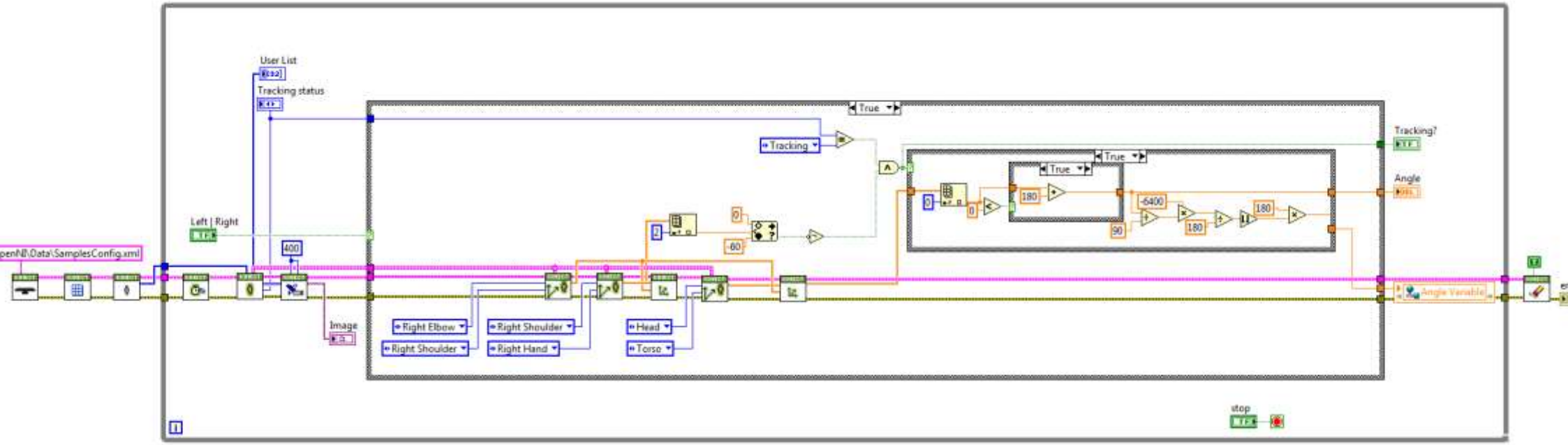


15.21 Haupt VI auf der Real-Time Compact RIO



15.22 Kommunikation VI auf der Host PC

15.22.1 Block Diagramm



15.22.2 Front Panel