DIPLOMARBEIT

Integration von Bildverarbeitung in ein industrielles Automatisierungssystem basierend auf IEC 61499

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von

Univ. Prof. Dipl.-Ing. Dr. techn. Bernard Favre-Bulle
Dipl.-Ing. Dr. techn. Alois Zoitl
Dipl.-Ing. Dr. techn. Christoph Sünder

E376
Institut für Automatisierungs- und Regelungstechnik

eingereicht an der Technischen Universität Wien
Fakultät für Elektro- und Informationstechnik

von

Felix-Stefan Chelu
Matr.-Nr.: 9925519
Dr.-Adolf-Schärf-Ring 7
3034 Maria Anzbach

Wien, im *Juni 2008* _____

Kurzfassung

Mit der vorliegenden Arbeit wird der neue zukunftsträchtige Standard für die Automatisierungstechnik, IEC 61499 auf seine Eignung zur industriellen Bildverarbeitung hin untersucht. Es soll gezeigt werden, dass komplexe, ressourcenhungrige Bildverarbeitungsanwendungen direkt und ohne Umwege in IEC 61499 entwickelt und ausgeführt werden können.

Hiermit sollen die Vorteile - im besonderen gegenüber seinem Vorgänger IEC 61131 - demonstriert werden und etwaige Nachteile bzw. Schwierigkeiten ebenfalls aufgezeigt werden. Dieses Experiment ist keinerlei eine Selbstverständlichkeit, denn die elektronische Bildverarbeitung verarbeitet große Datenmengen (Bilder, Videos) mittels aufwendigen mathematischen Algorithmen, während der IEC 61499 Standard mitunter Ressourcenökonomie und Echtzeitfähigkeit auf Mikrokontrollern anvisiert.

In diesem Zusammenhang wird auch ein neues Produkt, das *Kompaktkamerasystem SBOx-C* der Firma Festo vorgestellt, welche dem derzeitigen Entwicklungstrend folgend einen „intelligenten Sensor“ darstellt. Sie enthält sowohl einen hochauflösenden Kamerasensor als auch einen leistungsstarken Mikrokontroller zur Vorverarbeitung der Sensordaten. Sie folgt damit dem Trend der verteilten Automatisierungssysteme, indem - ermöglicht durch die niedrigen Hardwarekosten - immer mehr „Intelligenz“ in die Sensoren und Aktoren der Feldebene, hinzugenommen wird.

Es wird ein Konzept vorgestellt, wie das *Kompaktkamerasystem SBOx-C* in eine IEC 61499 Automatisierungsanlage integriert werden kann und wie passende IEC 61499 Bildverarbeitungsfunktionsblöcke - zur Entwicklung von Bildverarbeitungsanwendungen in IEC 61499 - zu Verfügung gestellt werden können. Hierzu wird auch ein neuer IEC 61499 Datentyp zur Repräsentation von Bildern eingeführt.

Die Bildverarbeitungsfunktionen selbst werden durch den Einsatz von externen Bibliotheken realisiert und deren Einbindungsmöglichkeiten in IEC 61499 werden im Detail analysiert. Weiters wird auch eine Lösung für das Problem der großen Datenmengen der Bilder vorgestellt, welche seitens IEC 61499 schwer verarbeitet werden können.

Den Abschluss der Arbeit stellt eine IEC 61499 Anwendung zur Demonstration der Ergebnisse dar. Diese zeigt sowohl die erfolgreiche Implementation der Bildverarbeitungsfunktionalität in IEC 61499, als auch die einfache Integration des Kompaktkamerasystems in ein IEC 61499 Automatisierungssystem. Sie veranschaulicht auch einen der größten Vorteile des IEC 61499 Standards, nämlich das Nebeneinander von verschiedensten Technologien wie z.B. Bildverarbeitung, Motion Control, Visualisierung, Logik usw., welche alle auf dieselbe Art - durch Funktionsblöcke - in eine Gesamtapplikation kombiniert werden.

Abstract

The present elaboration investigates the new seminal standard for industrial automation IEC 61499 about its fitness for image processing. It will be shown that complex image processing applications with a high demand on resources can be directly developed and executed within IEC 61499.

Hereby the advantages of IEC 61499 - especially over its ancestor IEC 61131 - should be demonstrated and any disadvantages and difficulties should be pointed out. This experiment is not at all self-evident, because the industrial image processing has to handle a huge amount of data by extensive mathematical operations, whereas IEC 61499 mainly focuses on microcomputers with limited resources and real-time requirements.

Within this context a new product for industrial automation will be presented: The Compact Camera System SBOx-C, engineered by the FESTO company. This product joins the actual trend of „intelligent sensors“. It contains a high resolution image sensor and further a high performance microcontroller to pre-process acquired sensor data. So it is well equipped to follow the tendency of distributed automation components by carrying more „intelligence“ into the sensor-actuator layer of automation processes.

A concept will be presented how to integrate the Compact Camera System SBOx-C into an automation plant, that is primarily based on IEC 61499 and further how an image processing application can be realised by using newly developed IEC 61499 Function Blocks.

The image processing functionality will be achieved by integrating external third party libraries and the integration possibilities of them into IEC 61499 runtimes are analysed in detail. Further a solution will be presented how to handle the huge amount of data of digital images, that causes a lot of troubles within IEC 61499 applications.

The completion of this thesis, is an IEC 61499 application to demonstrate the achieved results. It demonstrates the successful implementation of image processing into IEC 61499 applications as well as the successful integration of the Compact Camera System SBOx-C into an IEC 61499 based automation plant. It illustrates further one of the biggest advantages of the IEC 61499 standard, namely the coexistence and combination of a wide variety of technologies, like image processing, motion control, visualisation, logic, etc., that are all integrated using the same tools, viz. IEC 61499 Function Blocks.

Inhaltsverzeichnis

1 Einleitung

Die Automatisierungstechnik ist der umfassendste Bereich dem sich die nachfolgende Diplomarbeit zuordnen läßt. Dabei ist die Automatisierungstechnik keineswegs eine neue Forschungsdisziplin. Die Automatisierung von Arbeiten ist fast so alt wie die Menschheit selbst, denn seit jeher versucht der Mensch sich von Geräten und Technologien seine Arbeiten zu erleichtern bzw. sich diese zur Gänze abnehmen zu lassen.

Der Wortstamm *Automat* stammt aus dem altgriechischen Wort: αυτοματ und bedeutet in etwa „von selbst tun“ bzw. „sich selbst bewegend“. [1]

Nach [Wie98] wird das Wort αυτοματον folgendermaßen definiert: „nach eigenem Dünken handelnd“, also: von Dingen gesagt, deren Tätigkeit keiner äußeren Steuerung bedarf. Wenn man heutzutage eine moderne, hochgradig automatisierte Produktionsanlage besichtigt, so mag zwar der Eindruck entstehen, dass hier selbstätig Dinge ablaufen, jedoch gänzlich frei von jeder Steuerung?

Automatisieren läßt sich sehr vieles in sehr vielen unterschiedlichen Bereichen, z.B. automatisierte Datensicherung in der EDV (Elektronische Datenverarbeitung) oder das automatische Ein- und Ausschalten der Waschmaschine in der Haushaltstechnik. Ich möchte daher diesen umfassenden Bereich der Automatisierungstechnik etwas weiter auf die Produktionstechnik (nach Favre-Bulle [Fav04] als Anlagenautomatisierungstechnik bezeichnet) einschränken. Die Produktionstechnik schränkt die Automatisierungstechnik auf die mehr oder weniger automatisierte Produktion von Produkten ein.

Die ganz wichtige Frage, wozu überhaupt eine Produktionsanlange zu automatisieren bzw. aus welchen strategischen Gründen sich ein Unternehmen zur Automatisierung entscheidet, soll hier nicht weiter behandelt werden, sondern lediglich mit einem Verweis auf [Fav04] heutzutage grundsätzlich als gegeben angenommen werden - im Besonderen für den Produktionsstandort Europa.

Die Entwicklung der Computer- und Informationstechnologie (sowohl bei Hard- als auch bei Software) hat derart rasante Fortschritte gemacht, dass Produktionsanlagen mit diesen neuen Technologien nicht Schritt halten konn-

[1] Alle größeren Lexika, z.B. Mayers, Brockhaus aber auch Wikipedia führen diese Begriffserklärung

ten, d.h. sie sind bei den mittlerweile veralteten Technologien hängen geblieben und können die neuen Möglichkeiten noch nicht nutzen.

Bestehende Produktionsanlagen wurden sehr teuer und unter großem Aufwand aufgebaut, programmiert und in Betrieb genommen und müssen eine gewisse Zeitspanne in Betrieb sein, um sich zu amortisieren. Hinzu kommt, dass bis heute in der Anlagenautomatisierung das Duell zwischen Menge versus Vielfalt klar zugunsten der hohen Stückzahlen desselben Produkts und daher zu niedrigen Fertigungskosten entschieden wurde.

Die Vielfalt der Produktvariationen nimmt jedoch stetig zu und der Produktlebenszyklus wird immer kürzer. Der Kundenwunsch nach individuellen Produkten mit dem Grenzwert von „Losgröße 1“ [Fav04] wird immer stärker und seitens der Industrie, nur mittels einer flexiblen, dynamischen Produktion zu befriedigen. Unter dem Extremfall „Losgröße 1“ versteht man die Fertigung eines Produkts mit der Stückzahl = 1, d.h. ein Produkt ganz individuell auf die Kundenwünsche abgestimmt.

Die bis heute eingesetzte Technik in den meisten Produktionsanlagen, ist jedoch historisch bedingt sehr starr und unflexibel und noch nicht bereit der vielen aufkeimenden Möglichkeiten der erwähnten rasanten Entwicklung im Computer- und Informationstechnikbereich Herr zu werden.

Das Herzstück der meisten Produktionsanlagen ist die Speicherprogrammierbare Steuerung (SPS, engl. Programmable Logic Controller, PLC), welche bereits in den 80-er und 90-er Jahren erhältlich war und im Grunde schon den heutigen Modellen entsprach.

Die SPS hat die zuvor festverdrahtete Anordnung von Relais (verbindungsprogrammierte Steuerung (VPS)) abgelöst und ist in Kombination mit den eingesetzten standardisierten Programmiersprachen nach IEC 61131-3 die weitergeführte, computertechnische Abbildung dieser früheren festverdrahteten Relaisschaltungen. Manche der eingesetzten Bezeichnungen dieser Programmiersprachen erinnern noch an jene der Relaisschaltungen, z.B. Stromlaufplan. Zu den Programmiersprachen sei noch angemerkt, dass obwohl diese nach anfänglichen Schwierigkeiten mittlerweile durch das International Electrotechnical Commision (IEC) standardisiert sind, die meisten Hersteller zum Schutz ihrer Märkte, diese recht frei für sich auslegen und eine echte Interoperabilität praktisch nicht gegeben ist.

Die parallele Verarbeitung der einzelnen Relaisstränge bei der VPS wurde bei der SPS durch eine zyklische Abarbeitung der Geber (Sensoren) und Stellglieder (Aktoren) - im Millisekundenbereich - ersetzt. Es werden zyklisch die Eingangsbits eingelesen, mittels eines Programms verarbeitet und die Ausgangsbits hinausgeschrieben.

Obwohl eine heutige SPS eigentlich nicht viel weniger als ein industrieoptimierter PC ist, bleibt sie mangels „veralteter“ Softwarestandards hinter den Möglichkeiten moderner IT-Systeme zurück.

Genaue Statistiken über den Aufbau der Produktionsanlagen und zu der Frage wie Betriebe die Steuerungen ihrer Anlagen realisieren, gibt es keine, jedoch verrät ein Blick auf das am Markt befindliche Angebot sehr wohl die durch eben diese Betriebe bedingte Nachfrage. Die klassische SPS und der IEC 61131 Standard sind auch heute nicht wegzudenken. Die Unterstützung von Feldbussen und Netzwerken sowie die individuelle Programmierung in Standardprogrammiersprachen wie C/C++ bzw. Java sind aber auffällig oft angebotene Features[2] und bestätigen die wachsende Nachfrage nach verteilten/vernetzten Steuerungen und die nicht ausreichenden Möglichkeiten der SPS standardisierten Programmiersprachen nach IEC 61131-3. Im Abschnitt 2.4 wird etwas näher auf den IEC 61131 Standard eingegangen bzw. auf detailiertere weiterführende Literatur verwiesen.

Die vertikalen Ebenen in der Prozessleittechnik [Fav04], [Zei97b], [Zei97a], mit Prozessebene, Prozessleitebene, Unternehmensleitebene usw. rücken durch die immer billigere und leistungsfähigere Hardware näher zusammen. Die „klassischen“ Aktoren und Sensoren der nach diesen benannten Aktor/Sensorebene, welche sternförmig mit einer zentralen Intelligenz auf der Steuerungsebene verdrahtet sind, können nun selbst große Rechenkapazitäten besitzen und sind anstatt einzelverdrahtet über Feldbusse oder sogar direkt in die (Ethernet-)Büronetzwerke der Unternehmensleitebene angebunden.

Doch um diese neue Möglichkeiten besser nützen zu können, müssen auch die Softwarestrukturen angepaßt werden und obwohl auch in diesem Sektor große Anstrengungen unternommen werden, so erfolgt die Marktpenetration sehr langsam. Als besonders hervorzuheben sei hier der vielversprechende Standard IEC 61499, welcher im Jahre 2005 von der IEC standardisiert wurde, jedoch noch nicht die akademischen Testumgebungen verlassen hat um in eine Industrieanlage einzuziehen. Hierfür können zum Teil auch die Hersteller verantwortlich gemacht werden, welchen ein gewisses Desinteresse an offenen Standards unterstellt werden kann, da diese ihre alteingesessenen Cash-Cow Märkte bedrohen. Je „offener“ ein Standard desto größer das Bestreben nach echter Interoperabilität und damit nach beliebiger herstellerunabhängiger Austauschbarkeit. Dies ist natürlich nicht im Sinne der Hersteller.

[2] Das Wort Features ist aus dem Englischen entnommen und wird hier meinerseits lieber als die deutschen Entsprechungen: Eigenschaften, Funktionen bzw. Merkmale verwendet. Unter einem Feature ist die Funktionalität, das Leistungsmerkmal eines Geräts oder einer Software gemeint.

Im Abschnitt 2.5 wird der IEC 61499 Standard mit seinen verheißungsvollen neuen Möglichkeiten genauer erläutert.

Unter dem vorhin erwähnten „klassischen Sensor“ ist ein einfacher Sensor gemeint, welcher keinerlei Weiterverarbeitung der sensorischen Eingänge besitzt, sondern seine gewonnenen Messwerte nur zur Auswertung weitergeben kann. Das Gegenteil eines „klassischen Sensors“ ist ein sogenannter „intelligenter Sensor“, welcher beginnend bei einer einfachen Vorverarbeitung bis hin zur komplexen Auswertung der Messwerte gleich an Ort und Stelle vornehmen kann. Allgemein werden in der Automatisierungstechnik „intelligente Geräte“, sowohl Sensoren als auch Aktoren unter *Smart Devices* zusammen gefaßt.

Ein gutes Beispiel für einen „intelligenten Sensor“ ist der Mensch. Er vereint viele Sensoren in sich, kann sehr dynamisch parametrisiert werden, paßt sich adaptiv an geänderte Bedingungen an und kann aufgrund seiner kognitiven Informationsverarbeitung sehr komplexe Auswertungen seiner Meßwerte vornehmen und erst die Ergebnisse seiner Vorarbeitung - welche oftmals aus einem riesigen optischen und/oder akkustischen Datenstrom auf ein einfaches Ja bzw. Nein reduziert wurden - weiterreichen.

Ein weiterer „intelligenter Sensor“ welchem für die Zukunft eine gewichtige Rolle zugeschrieben wird [VDI00] ist das „intelligente elektronische Auge“, worunter man das „maschinelle Sehen“ (engl. Machine Vision) versteht. Die industrielle Bildverarbeitung (IBV) wird als eine Schlüsseltechnologie[3] bezeichnet, welche alle Bereiche der Technologie herausfordert.

Wie der Titel vorwegnimmt, beschreibt die nachfolgende Diplomarbeit die Integration von Bildverarbeitung in ein modernes, verteiltes, auf IEC 61499 basiertes Automatisierungssystem.

In den nächsten Unterabschnitten möchte ich zuerst die Motivation hierfür erläutern und aus dieser eine konkrete Aufgabenstellung und somit das Ziel dieser Diplomarbeit herauskristallisieren. Danach im letzten Unterabschnitt des Einleitungskapitels auf die weitere Struktur und den Aufbau der Diplomarbeit eingehen.

1.1 Motivation

Wie in den einleitenden Sätzen vorangestellt, ist im Bereich der Anlagenautomatisierung sehr viel Forschung und Entwicklung bereits im Gange bzw.

[3] Eine Schlüsseltechnologie hat die Kinderschuhe bereits verlassen und befindet sich schon in der Wachstumsphase. Sie ermöglicht die Erschließung neuer Technikbereiche und ist entscheidend für die Wirtschaft der Zukunft.

noch viel hievon von Nöten, um sich von den alteingesessenen Strukturen zu emanzipieren und neue Standards und Technologien einzuführen, sodass den Anforderung dieses wachsenden Marktes der Produktionsautomatisierung gerecht werden kann.

Die flexible Automation der Produktherstellung, von der individuellen Kundenbestellung beginnend und mit der Auslieferung des bestellten Produktes endend, ist noch ein sehr weiter Weg und wird von den Ingenieuren der nächsten Jahrzehnte noch einiges abverlangen. Besonders die Universitäten und akademischen Einrichtungen sind hier ebenfalls gefordert Grundlagenforschung zu betreiben und neue, offene Standards und Richtlinien für die Industrie bereitzustellen. Dies sollte selbstverständlich in gegenseitiger Zusammenarbeit und in Kooperation mit der Industrie geschehen.

Die ACON (*Agile Control*) Forschungsgruppe des Automatisierungstechnikinstituts der Technischen Universität Wien, betreibt seit Jahren Forschung an vorderster Front, insbesondere den neuen Industriestandard IEC 61499 betreffend und hat schon in Zusammenarbeit mit namhaften Unternehmen viele industrierelevante Ergebnisse erzielt.

Diese Diplomarbeit fügt sich nahtlos in die Forschungs- und Entwicklungstätigkeit der ACON-Gruppe ein, indem die Zusammenführung der beiden Schlüsseltechnologien, IEC 61499 und industrielle Bildverarbeitung, realisiert und in einem praktischen industrienahen Aufbau demonstriert werden soll.

Hierfür konnte auch seitens der Bildverarbeitung auf neueste Technologien der Firma FESTO zurückgegriffen werden, welche ebenfalls in Zusammenarbeit mit dem Automatisierungstechnikinstitut der Technischen Universität Wien eine „intelligente Kamera“, das SBOx-C [Fst08] entwickelt hat und welche alle Voraussetzungen für einen vorhin erwähnten „intelligenten Sensor“ mit sich bringt.

1.2 Ziele der Diplomarbeit

Im Folgenden werden die Ziele, welche im Rahmen dieser Diplomarbeit erreicht werden sollen, definiert, ohne etwaige technische Details vorwegzunehmen:

intelligenter Sensor: Einbindung eines „intelligenten Sensors“ in ein Automatisierungssystem.

Wie bereits im vorherigen Unterabschnitt angemerkt, hat die Firma FESTO ein neues Produkt - das *Kompaktkamerasystem SBOx-C* - entwickelt, welches eine Hochleistungskamera gemeinsam mit einem lei-

stungsstarken Rechner in einem *Smart Device* auf kleinstem Raum integriert, abgebildet in Abb. 1.1. Die genauen technischen Merkmale des *Kompaktkamerasystems SBOx* werden im Unterabschnitt 2.5.5 beschrieben.

Abbildung 1.1: FESTOs Kompaktkamerasystem SBOx-C

verteilte Applikation: Dieses „intelligente elektrische Auge“ soll in eine auf IEC 61499 basierende, verteilte Automatisierungsanlage integriert werden. Zum Zeitpunkt der Arbeiten an dieser Diplomarbeit bestand diese

Abbildung 1.2: Portalroboter

jedoch lediglich aus einem Portalroboter, abgebildet in Abb. 1.2, dessen vier Achsen - zwei getrennt angetriebene X-Achsen, eine Y- und ein Z-Achse, jeweils durch einen eigenen Mikrokontroller mittels einer IEC 61499 Anwendung gesteuert werden. Angestrebt wird (als Fernziel, das wohl noch die eine oder andere weiterführende Diplomarbeit erfordern

wird) die Integration in ein agentenbasierendes Palettentransportsytem mit mehreren Arbeits(Index-)Stationen, welche über dutzende von Mikrokontrollern mittels einer verteilten IEC 61499 Applikation gesteuert wird.

IEC 61499 Fähigkeit: Die ACON-Gruppe hat über mehrere Jahre und mehreren Versionen eine stabile Laufzeitumgebung für Mikrokontroller für IEC 61499 Anwendungen geschaffen, welche als Basis für die Integration der „nackten" Kamera in das IEC 61499 Netzwerk fungieren soll.

Bildverarbeitungsfunktionen: Es sollen Bildverarbeitungsfunktionen für IEC 61499 Anwendungen implementiert werden, welche sowohl auf dem *Kompaktkamerasystem SBOx-C*, als auch auf anderen IEC 61499 fähigen Kontrollern ausgeführt werden können.

1.3 Struktur der Diplomarbeit

Um die angestrebten Ziele erreichen zu können, bedarf es zunächst einer gründlichen Analyse des derzeitigen Standes der Technik, welche im nachfolgenden Kapitel 2 erläutert werden soll.

Zuerst wird eher kurz der derzeit vorherrschende Standard IEC 61131-3 untersucht, und Möglichkeiten zur Integration des *Kompaktkamerasystems SBOx-C* inklusive Bildverarbeitungsfunktionen in ein auf IEC 61131 basierendes Automatisierungssystem besprochen. Danach wird in ähnlicher Weise der potentielle Nachfolgestandard IEC 61499 Standard analysiert.

Etwas näher betrachtet werden auch die verfügbaren Laufzeitumgebungen für IEC 61499 Anwendungen, sowie auch die verfügbaren Toolchains[4] zur Erstellung solcher Anwendungen.

Weiters wird auf die industrielle Bildverarbeitung eingegangen und die Anforderungen des „maschinellen Sehens" an die Soft- und Hardware ausgearbeitet.

Im Kapitel 3 wird ein Konzept zur Umsetzung der Ziele der Diplomarbeit im Detail beschrieben. Hierbei wird auf die zu lösenden Probleme eingegangen und ein gangbarer Weg zur Lösung vorgestellt.

[4] Eine Toolchain ist die Summe der zur Verfügung stehenden Programme, um wiederum andere Programme erstellen zu können. Ein Tool (dt. Werkzeug) wird nach dem anderen in einer Chain (dt. Kette) benutzt, wobei die Ausgabe des Einen als Eingabe des Nächsten benutzt wird.

Im anschließenden Kapitel 4 wird die konkrete Umsetzung des Konzepts beschrieben und auf implementationstechnische Feinheiten eingegangen.

Die abschließenden Kapitel 6 und 7 geben einen kurzen Ausblick über weiterführende Aufgaben und Probleme, welche es zu lösen gilt und fassen die Ergebnisse der Arbeiten an dieser Diplomarbeit nochmals zusammen.

2 Stand der Technik

Das folgende Kapitel behandelt den Stand der Technik der von dieser Diplomarbeit betroffenen Technologien zum Zeitpunkt der Arbeiten.

Zum einen wäre dies das große Gebiet der Bildverarbeitung, im Besonderen der industriellen Bildverarbeitung. Dieser Punkt wird als erster behandelt, um die Anforderungen, welche sich aus der Anwendung von Bildverarbeitung im industriellen Einsatz ergeben, herauszuarbeiten. Diese sind hilfreich, wenn als nächstes der heutige Standard für Automatisierungssysteme IEC 61131 und der nachfolge Standard IEC 61499 analysiert werden.

Allen voran wird jedoch eine Begriffsdefinition von verteilten Steuerungen - bzw. besser ausgedrückt verteilten Automatisierungssystemen - gestellt, um für den weiteren Verlauf dieser Diplomarbeit dieses *en vogue*[1] Schlagwort einzugrenzen.

2.1 Verteilte Steuerungen

In dieser Diplomarbeit wird eine verteilte Steuerung nach Favre Bulle [Fav04] definiert, dargestellt in Abb. 2.1.

Unter einem verteilten Automatisierungssystem versteht man die dezentrale Realisierung von sowohl Hardware als auch Software, d.h. es gibt keine einfach oder mehrfach konzentrierte, zentrale Intelligenz, welche das Gehirn eines Automatisierungssystems darstellt. In diesem Zusammenhang ist Intelligenz im Sinne der Computer- und Informationsverarbeitungstechnik gemeint, d.h. hardwareseitig arithmetische Prozessorer, Arbeitsspeicher, Kommuniktionsschnittstellen usw. und softwareseitig mehr oder weniger standardisierte Programme, Betriebssysteme usw. Die Verteilung der Hardware in der Automatisierungstechnik hat bereits vor vielen Jahren eingesetzt und ist heute bereits *State of the Art*. Die „echte“ Verteilung der Software ist jedoch noch eine große Herausforderung an welcher zur Zeit viel geforscht und gearbeitetet wird. In einem „echt“ verteilten System kommunizieren *Smart Devices* - d.h.

[1] franz. „in Mode“

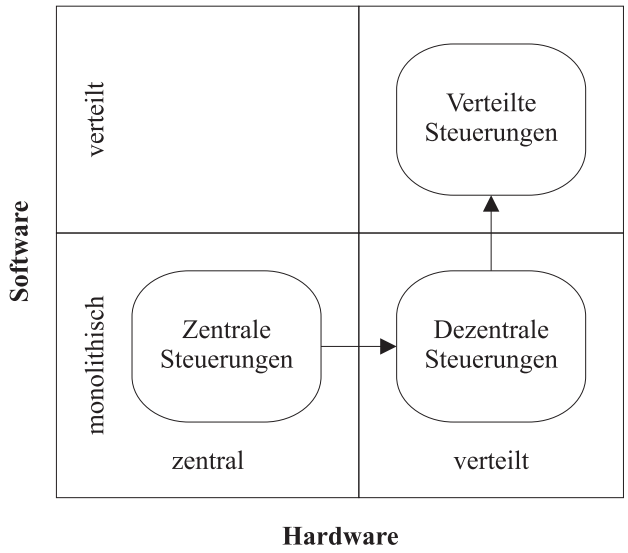


Abbildung 2.1: Zentrale, dezentrale und verteilte Steuerungen (Quelle: [Fav04], Seite 87)

„intelligente Sensoren“ und „intelligente Aktoren“ - direkt über Feldbusse bzw. sonstige standardisierte Netzwerke und standardisierte Protokolle - miteinander, ohne den Umweg über eine zentrale Intelligenz (Software), dargestellt in Abb. 2.2.

Über verteilte Automatisierungssysteme wird schon seit längerem viel gesprochen, jedoch verstand man darunter früher nur eine verteilte Hardware. Eine solche Automatisierungsanlage - hauptsächlich durch die Ära der SPSen geprägt - wird nach [Fav04] nur als dezentral jedoch nicht als verteilt bezeichnet. Erst der Aufbruch und die Modularisierung von großer „monolithischer“ Software aus „einem Guß“ in viele kleine, klar definierte und standardisierte und somit wiederverwendbare und austauschbare Funktionsblöcke öffnen den Weg von dezentralen zu „echt“ verteilten Automatisierungssystemen. Hierzu soll der IEC 61499 Standard ein Antrieb sein.

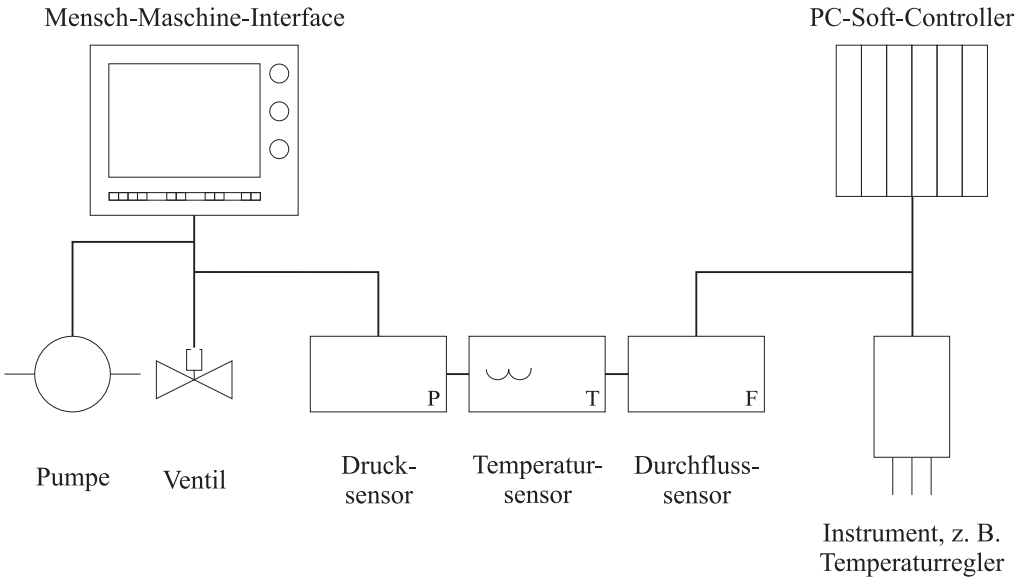


Abbildung 2.2: Verteilte Funktionalität mit verteilten Geräten (Quelle: [Fav04], Seite 90)

2.2 Bildverarbeitung

Die industrielle Bildverarbeitung, das *Maschinelle Sehen*[2] (engl. *Machine Vision*), ist schon seit langem eine seitens der Industrie sehr gefragte und vielversprechende Technologie. Jedoch zeigen sich in der praktischen Umsetzung große Schwierigkeiten, welche noch bewältigt werden müssen.

Die größte Schwierigkeit bei der Interpretation von Einzel- bzw. bewegten Bildern - aber in Analogie auch von gesprochener Sprache - ist der Bedarf an Vor- und Zusatzwissen. Die „gesehenen“ Bilder bzw. die „gehörte“ Sprache wird in Verbindung mit einer gewissen Erwartung und einer bereits vorhandenen Erfahrung in Verbindug gesetzt und erst daraus lassen sich Ergebnisse erzielen. Eine solche *kognitive* Verarbeitung erfordert sehr viele Ressourcen und ist für Computer zurzeit *noch* sehr schwierig zu bewältigen.

Die Bildverarbeitung ist eine Technologie, welche viele weitere Technologien

[2]Das *Maschinelle Sehen* ist ein sehr umfassender Begriff. So beinhaltet er z.B. auch Methoden der Informationsgewinnung mittels Laserabtastung oder Röntgenstrahlung. Im Folgenden soll aber nur die Bildverarbeitung mittels Licht im sichtbaren Bereich näher betrachtet werden.

herausfordert und zur praktischen Anwendung integriert. Dank der schnellen Entwicklung der betroffenen Technologien in den letzten Jahrzehnten kann aber auch die industrielle Bildverarbeitung mit fortschrittlichen Entwicklungen aufwarten, wodurch deren praktischer Einsatz in Industrieanlagen kontinuierlich attraktiver wird.

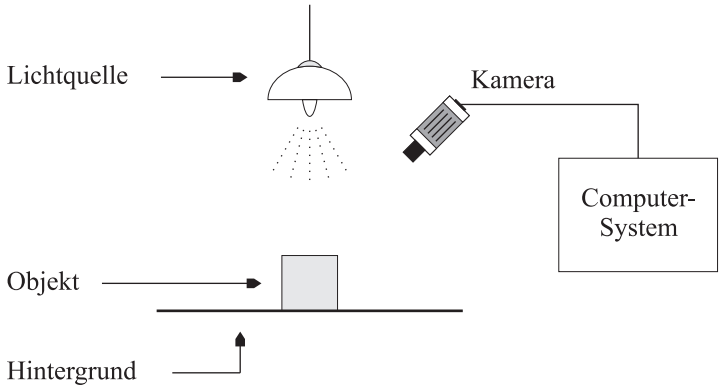


Abbildung 2.3: Komponenten einer Bildverarbeitungsanwendung

Die notwendigen Komponenten eines Bildverarbeitungssystems, skizziert in Abb. 2.3, sind:

- Kamera
  - CCD[3] oder CMOS[4]
  - Objektiv
  - digitalisierte Bilder
- Lichtquelle(n)
- Objekt(e)
  - Hintergrund
- Computersystem
  - Hardware
  - Software
  - Treiber zur Kommunikation mit der Kamera

[3] **C**harge **C**oupled **D**evice
[4] **C**omplementary **M**etal **O**xide **S**emiconductor

Die geometrische Anordnung von Kamera (inkl. gewähltem Objektiv), Lichtquelle und Objekt spielt ein sehr wichtige Rolle und kann den Aufwand der nachfolgenden Verarbeitung drastisch reduzieren bzw. überhaupt eine erfolgreiche Weiterverarbeitung erst ermöglichen. Sich ändernde Lichtverhältnisse und Reflexionen im Bild erschweren drastisch die Aufgabe einer erfolgreichen Bildverarbeitungsanwendung. Besonders die Beleuchtung ist von ganz großer Bedeutung, denn gute Aufnahmen erfordern eine gute Beleuchtung. Sie ist so zu wählen, dass das Objekt auf eine Weise kontrastiert wird, dass genau die Details für die Kamera sichtbar werden, die von anderen unterschieden werden sollen.

Trotz der Wichtigkeit dessen, soll hier der geometrische Aufbau einer Anlage nicht weiter untersucht, sondern qualitative Bildaufnahmen vorausgesetzt werden. Als weiterführende Literatur wird an dieser Stelle auf das Buch von Gerhard Weissler [Wei07] verwiesen, in welchem er besonders ausführlich auf das Zusammenspiel von Beleuchtung, Objektiv und Kamerasensor eingeht.

Sind erst gute Aufnahmen vorhanden, so erfolgt danach die Weiterverarbeitung und Informationsgewinnung mittels eines Computers. Hier wird versucht das eingangs erwähnte fehlende Vor- und Zusatzwissen zu kompensieren bzw. dieses stattdessen in eine starre Kette von Bildverarbeitungseinheiten abzubilden. Natürlich müssen hierbei Abstriche in der Flexibilität gemacht werden.

Bei der Entwicklung einer Bildverarbeitungsanwendung, d.h. zur Lösung einer vorgegebenen Problemstellung mittels des Einsatzes von Bildverarbeitung, kann leider nicht auf ein theoretisches Regelwerk zurückgegriffen werden. Vielmehr bedarf es handwerklicher Fähigkeiten und eines „Gefühls“ für die Werkzeuge (Funktionen) und ihrer Einsatzfähigkeiten. Die Theorie bietet eine endlose Fülle von Techniken, um bestimmte Ergebnisse zu erzielen, jedoch ob diese für den praktischen Einsatz vor Ort geeignet sind, bleibt dem jeweiligen Entwickler einer Bildverarbeitungsanwendung überlassen.

Die folgende Liste zeigt eine *typische* Kette von Bildverarbeitungsoperationen, wie sie in der industriellen Bildverarbeitung und didaktischen Lehrbüchern oft eingesetzt wird [Bass04] [Son08]:

- Weißabgleich
  Der Weißabgleich dient dazu die Kamera auf die Lichtsituation am Aufnahmeort korrekt anzupassen[5], da sich ansonsten verfälschte Farben er-

[5] In diesem Zusammenhang spricht man von Lichttemperatur, wobei die Farbe der Lichtquelle von einem „kühlen“ blau in ein „warmes“ rot übergeht. Gemeint ist die notwendige Temperatur eines idealen schwarzen Körpers, um mit der angegebenen Farbe zu strahlen und die physikalische Einheit ist somit Kelvin.

geben würden. Dies erfolgt meist schon vollautomatisch in der Kamera, es sei denn, diese liefert nur *RAW*[6] Daten.

Für genauere Ergebnisse wird hierzu eine bekannte Referenzfarbe in den Aufnahmebereich der Kamera positioniert, sodass der nachfolgende Weißabgleich anhand der Farbabweichung vom Referenzbild getätigt werden kann.

- Tonwertkorrektur
  Hier wird für jeden Farbkanal das Spektrum der vorkommenden Helligkeitswerte (heutzutage im Allgemeinen eine Zahl zwischen 0-255, d.h. ein Byte) so gedehnt, dass das gesamte zur Verfügung stehende Spektrum genützt wird.

- Graustufenbildung
  Die meisten Bildverarbeitungsanwendungen verzichten auf die Interpretation der Chrominanz und begnügen sich mit der Luminanzinformation eines Bildes, d.h. die Farbinformation wird verworfen und übrig bleibt ein Graustufenbild.[7]

- Binarisierung
  Ein Binärbild ist ein Schwarz-Weiß-Bild ohne weitere (Zwischen-)Graustufen. Unter Binarisierung versteht man somit die Reduktion der Farbtiefe eines Graustufenbildes auf ein Schwarz-Weiß-Bild, d.h. , dass jeder Pixel, abhängig von seinem Graustufenwert, seinen Nachbarn, der Häufigkeitsverteilung der Graustufenwerte im gesamten Bild usw. in Schwarz oder Weiß umgewandelt wird. Im Allgemeinen wird das Bild so binarisiert, dass nur noch die relevanten Information erhalten bleiben und sich diese vom Hintergrund lösen.

- Störungsfilterung
  Das Rauschen - besonders an den Kanten - welches im Binärbild zu verfransten Pixeln[8] führt, wird entfernt. Hierbei bedient man sich im Allgemeinen folgender Operationen:

  – Erosion, Dilation
  – Open, Close

[6] Dies sind die Daten direkt aus den CCD- bzw. CMOS Sensoren.

[7] Auch das menschliche Auge reagiert wesentlich empfindlicher auf Helligkeits- als auf Farbunterschiede. Ein Faktum, welches von diversen Bildkompressionsverfahren, z.B. JPEG für eine stärkere Komprimierung erfolgreich ausgenützt wird.

[8] Das Wort Pixel stammt vom englischen *picture elements* und definiert einen einzelnen Bildpunkt, welcher je nach Farbtiefe eine Farbe annehmen kann.

Bei der Erosion werden vereinzelte weiße Pixel d.h. weiße Pixel, welche von schwarzen Pixeln berührt werden, schwarz gefärbt. Dadurch verkleinert sich ein weißes Objekt auf einem schwarzen Hintergrund, jedoch werden die Kanten geglättet. Bei der Dilation geschieht das Gegenteil. Jeder schwarze Pixel, welcher einen weißen Pixel berührt wird weiß gefärbt, wodurch sich ein weißes Objekt auf einem schwarzen Hintergrund vergrößert.

Die Open und Close Operation ist die Anwendung beider Operationen hintereinander, d.h. die Erosion gefolgt von der Dilation bzw. vice versa. Dadurch bleibt die Objektgröße - im großen und ganzen - erhalten, aber die zerfransten Kanten werden geglättet.

- Segmentierung
  Unter der Segmentierung eines Binärbildes versteht man deren Aufteilung in einzelne Regionen zur nachfolgenden Merkmalsextraktion. Die einfachste Segmentierung besteht darin, zusammenhängende Regionen in einem Binärbild zu markieren.

- Merkmalsextraktion
  Unter Merkmalsextraktion versteht man die Bestimmung von relevanten Merkmalen der einzelnen Regionen des segmentierten Bildes, z.B. Fläche, Orientierung usw. Zur Merkmalsextraktion gehören aber auch Methoden der Bilderkennung bzw. Kategorisierung z.B. mittels Neuronaler Netzwerke.

- Ergebnis
  Bestimmung der Ergebnisse aufgrund der vorhin berechneten Merkmale.

Je nach konkreter Anwendung der Bildverarbeitung weicht die tatsächliche Kette von Bildverarbeitungsoperationen von der hier dargestellten ab. Jedoch ist eine schrittweise Weiterverarbeitung der Eingangsbilder im Allgemeinen typisch für eine Bildverarbeitungsanwendung.

Zusammenfassend definiert sich ein Bildverarbeitungssystem durch folgende Eigenschaften und Anforderungen:

- großer Speicherverbrauch
  Die Speicherung bzw. Aufbewahrung eines einzelnes Bildes in ausreichender Qualität benötigt viel Speicher. Jedoch je höher die Auflösung gewählt wird, umso bessere Ergenisse lassen sich erzielen. Ein Video, d.h. eine Serie von Bildern besteht wiederum aus sehr vielen Bildern pro Sekunde, im Besonderen für Hochgeschwindigkeitskameras mit z.B.

400 Bildern pro Sekunde. Auch hier gilt aber, dass je höher die Bildwiederholfrequenz gewählt wird, umso besser die Erkennung von schnellen Abläufen.

- hohe Rechenleistung
  Die vorhin beschriebene Kette von Bildverarbeitungsoperationen muss für jedes aufgenommene Bild einzeln durchgeführt werden. Dies führt zu einer sehr hohen Rechenanforderung, welche selbst heutige Hochleistungsprozessoren nicht zur Genüge erfüllen können. Die verfügbare Rechenleistung der Computersysteme ist die größte Hürde der Bildverarbeitung.

- sequentielle Verarbeitung
  Von der Bildquelle bis zur endgültigen Informationsgewinnung werden die Bilder schrittweise weiter verarbeitet. Hierbei ist es nicht notwendig jedes Zwischenbild zu behalten, sondern es kann oftmals das gleiche Bild für den nächsten Schritt weiterverarbeitet werden. Eine Parallelisierung dieser Bildverarbeitungskette ist praktisch kaum möglich bzw. sinnvoll.

Im Laufe der letzten Jahrzehnte haben sich eine - mehr oder weniger umfangreiche - Vielzahl von Bildverarbeitungswerkzeugen und -Bibliotheken - sowohl kommerzielle als auch frei verfügbare - entwickelt und die Forschung und Entwicklung in diesem Sektor wird sehr aktiv vorangetrieben.

Deshalb sollen sich die Arbeiten an dieser Diplomarbeit nicht auf die Umsetzung von Bildverarbeitungsfunktionen konzentrieren, da dies schon vielfach in weitaus umfangreicheren Projekten als einer Diplomarbeit getätigt wurde, sondern vielmehr auf die *Einbettung* vorhandener Bildverarbeitungsbibliotheken in ein modernes, verteiltes Automatisierungssystem. Dies erspart zwar den gesamten Entwicklungsaufwand der Bildverarbeitungsfunktionen, bringt jedoch das Problem der Einbindung externer Bibliotheken in die Standards für Automatisierungssysteme, IEC 61131 und IEC 61499. In den jeweiligen nachfolgenden Kapiteln wird deshalb auch auf diese Thematik kurz eingegangen.

2.3 Kompaktkamerasystem SBOx-C

Das *Kompaktkamerasystem SBOx-C* [Fst08] der Firma FESTO, welches kurz im Abschnitt 1.2 *„Ziele der Diplomarbeit“* bereits genannt wurde, ist eine Hochleistungskamera mit integrierter Rechenlogik und folgenden Eigenschaften (Angaben laut Hersteller):

- Frei programmierbare Hochgeschwindigkeitskamera
- Bis zu 185 Bilder/sek bei voller Auflösung
- CMOS-Sensor mit Global Shutter
- Intel Xscale Prozessor mit 400 MHz Taktrate
- Open Source Linux Betriebssystem
- Frei zugängliches Xilinx Spartan-3 FPGA
- Sehr kompaktes Design
- Äußerst geringe Leistungsaufnahme
- Ethernet- und CAN-Bus-Schnittstelle
- Schutzart IP65/67
- Kompatibel mit vielen Standard-Softwarepaketen

Hierbei handelt es sich noch um ein ganz neues Produkt (Stand Mai 2008) und FESTO vertreibt das *Kompaktkamerasystem SBOx-C* noch nicht aktiv, sondern nur über Drittanbieter, z.B. Videor Technical [wwwVRT].

Das Kompaktkamerasystem ist frei programmierbar und dessen Programmierung erfolgt mittels mitgelieferter Open-Source Bibliotheken in der Programmiersprache C/C++. Die Bibliotheken enthalten Funktionen für alle hardwarespezifischen Merkmale des *Kompaktkamerasystems SBOx-C*, so z.B. zur Bildakquirierung oder zum Lesen bzw. Schreiben der externen Ein- bzw. Ausgänge.

Zur Programmierung kann jede beliebige C/C++ Entwicklungsumgebung eingesetzt werden. Die Übersetzung für die ARM-Architektur des *Kompaktkamerasystems SBOx-C* erfolgt durch ein Cross-Compile[9] von einem Linux Rechner, mittels einer von FESTO bereitgestellten Tool-Chain.

Ihre Einbindung in ein Automatisierungssystem basierend auf IEC 61131 oder IEC 61499 wird am Ende des jeweiligen nachfolgenden Abschnitts besprochen.

[9] Cross-Compile bezeichnet das Übersetzen (Kompilieren) eines Programms auf einem Hostsystem für ein anderes - nicht kompatibles - Zielsystem. Besonders für eingebettete Systeme (engl. Embedded Systems) ist dies eine gebräuchliche Technik.

2.4 IEC 61131

Dieser Unterabschnitt behandelt nur sehr oberflächlich den IEC 61131 Standard und zielt hauptsächlich auf die Möglichkeiten der Integration von Bildverarbeitungsfunktionalität in Automatisierungssysteme basierend auf IEC 61131. Für weiterführende Literatur wird der Standard selbst [PLC01] und die Bücher [Neu98] und [Joh01] empfohlen.

Der Standard bestand aus anfangs fünf Teilen und wurde im Laufe der letzten Jahre auf sieben Teile ergänzt. Von Interesse seien hier nur die Teile zwei und drei, welche die Hard- und die Softwarearchitektur beschreiben.

2.4.1 Hardwaremodell nach IEC 61131

Das Hardwaremodell nach IEC 61131 bildet die Grundlage für das Verständnis der Funktionalität einer SPS. Der tatsächliche technische Aufbau ist dabei stark herstellerabhängig, jedoch ist die Funktionalität aufgrund des Standards bei allen SPSen gleich. Die Gesamtfunktionalität ist in *logische* Funktionen aufgeteilt deren Aufgaben und Beziehungen zueinander in der Abb. 2.4 dargestellt sind.

Das Kernstück jeder SPS ist die Signalverarbeitungsfunktion, welche für die Ausführung von Anwenderprogrammen verantwortlich ist und sich hierfür der Betriebssystem-, Speicher- und Anwendungsfunktionen bedient.

Die Interfacefunktion zu den Sensoren wandelt die vom Prozeß anliegenden Signale in eine für die Signalverarbeitungsfunktion verständliche Form um und vice versa.

Die Kommunikationsfunktionen sorgen für den Datenaustausch zu externen Geräten wie z.B. PCs[10]. Zwei besondere dieser Kommunikationsfunktionen sind die Mensch-Maschinen-Interfacefunktionen und die Programmier- und Debug-Funktionen über welchen die Interaktion mit dem Bediener und dem Programmierer erfolgt.

[10] engl. Personal Computers; hierunter versteht man Computerarbeitsstationen für den Menschen

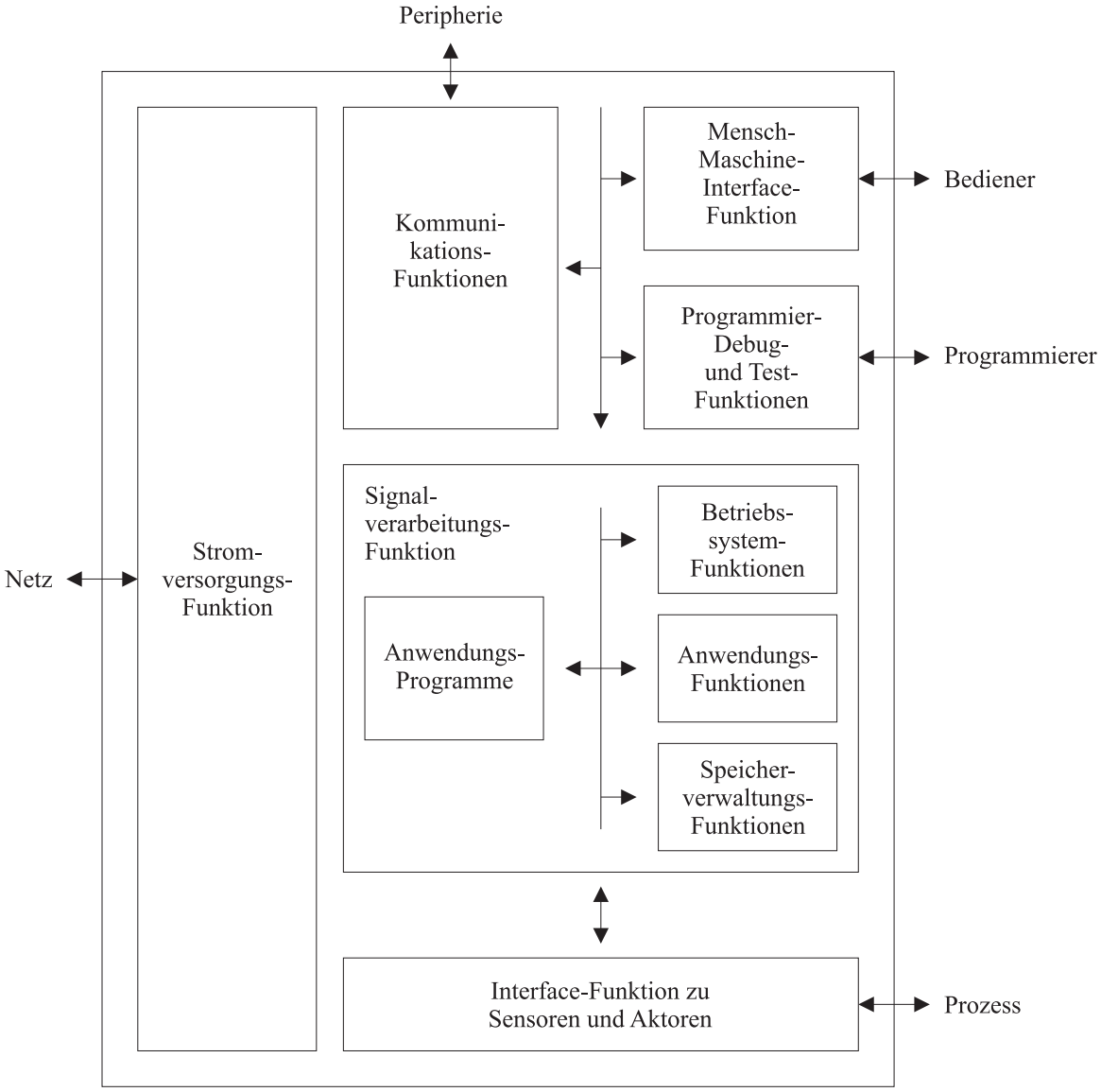


Abbildung 2.4: Hardwaremodell nach IEC 61131

2.4.2 Softwaremodell nach IEC 61131

Um den großen Funktionsumfang eines Automatisierungsgeräts[11] beherrschen zu können, teilt das Softwaremodell nach IEC 61131 - als Abbild des Hardwaremodells - ein Automatisierungsgerät in Kategorien ein, dargestellt in Abb. 2.5.

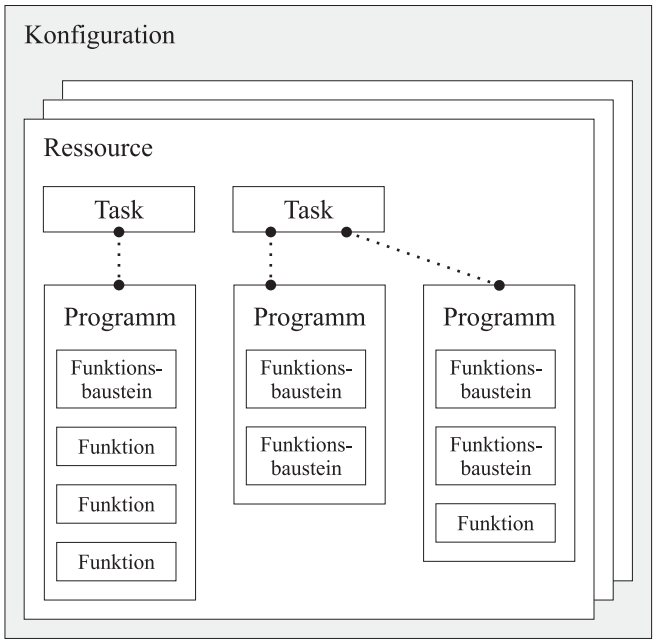


Abbildung 2.5: Softwaremodell nach IEC 61131

Konfiguration und Ressource beschreiben die grundlegende Struktur des Automatisierungsgeräts und die Abbildung der Software auf die Hardware. Beide Kategorien werden unter dem Namen *Konfigurations-Organisationseinheit* (KOE) zusammengefasst.

Programme, Funktionsbausteine und Funktionen sind Software im klassischen Sinn und haben den Oberbegriff *Programm-Organisationseinheiten* (POE). Allgemein gilt für alle POEs, dass sie nicht rekursiv aufgerufen werden dürfen.

[11] Unter Automatisierungsgerät und Speicherprogrammierbare Steuerung versteht man selbiges. Der Ausdruck Speicherprogrammierbare Steuerung ist historisch bedingt entstanden, hat sich jedoch gut durchgesetzt. Passender und ganzheitlicher ist aber der Name Automatisierungsgerät.

Konfiguration

Die Konfiguration beinhaltet die Zuordnung jeder einzelnen Ressource zu einem physischen Prozessor und einem physischen Speicherbereich und definiert globale Variablen, welche für alle Ressourcen verfügbar sind. Weiters kann sie auch sogenannte Zugriffspfade für ressourceninterne Variablen definieren, welche wiederum von anderen Ressourcen zur Interressourcekommunikation verwendet werden können.

Die Definition der E/A-Anbindung ist ebenfalls innerhalb der Konfiguration vorzunehmen da diese symbolische Namen den - für alle Ressourcen gültigen - fest-verdrahteten Ein- und Ausgängen (E/As) zuweist.

Mittels der Kommunikations-Funktionen der SPSen werden Konfigurationen auf SPSen geladen bzw. gelöscht. Beim Starten einer Konfiguration werden alle globalen Variablen initialisiert und alle enthaltenen Ressourcen werden gestartet. Beim Stoppen werden alle Ressourcen ebenfalls angehalten.

Ressource

Eine Konfiguration kann eine oder mehrere Ressourcen enthalten. Die Definition von Ressourcen ergibt sich als erste Modularisierung der zu lösenden Gesamtaufgabe und resultiert weniger aus technischen Gründen sondern ergibt sich vielmehr aus der Anwendersicht des Problems. Genaue Richtlinien, wie Ressourcen angelegt werden sollen, gibt IEC 61131 nicht vor.

Mit einer Ressource werden gewisse Speicherbereiche verbunden und Ressourcenvariablen definiert, welche für alle POEs innerhalb einer Ressource gültig sind.

Task

Eine Ressource kann eine oder mehrere Tasks enthalten. Ein Task fasst innerhalb einer Ressource ein oder mehrere Programme mit gleichem Zeitverhalten zusammen. Das Zeitverhalten kann eine periodische Aktivierung oder eine Aktivierung durch Ereignisse sein. Darüber hinaus können Prioritäten[12] und Verdrängungsmechanismen[13] für Tasks festgelegt werden. Wird ein Programm keinem Task zugeordnet, so läuft er zyklisch mit niedrigster Priorität im Hintergrund, immer wenn die Ressource gerade frei ist.

[12] zur Bestimmung welches Programm als nächstes aktiviert wird, falls im entscheidenden Augenblick mehrere Programme aktiviert werden sollten

[13] zur Festlegung, ob ein zu aktivierendes Programm ein gerade ausgeführtes unterbrechen (verdrängen) darf

Programm

Programme und Funktionsbausteine sind im großen und ganzen identisch, außer dass Programme selbst keine anderen Programme beinhalten können, sondern nur von Ressourcen aufgerufen werden dürfen. Weiters können Programme über die Ressource einem Task zugewiesen werden. Programme bestehen aus Funktionsbausteinen und Funktionen.

Funktionsbaustein

Funktionsbausteine besitzen beliebig viele Eingangs-, Ausgangs- und interne Variablen und sie dürfen vom Programm oder anderen Funktionsbausteinen aufgerufen werden. Sie können explizit einem Task zugewiesen werden, welcher von dem Task der aufrufenden POE abweichen kann.

Funktionsbausteine können (mit jeweils anderen Datensätzen) mehrfach benutzt (instanziert) werden. Sie können statische Variablen und somit ein Gedächnis besitzen, welches über mehrere Aufrufe hinweg erhalten bleibt. Somit lassen sich z.B. Zähler realisieren, welche ihren Zählerstand über mehrere Aufrufe hinweg nicht vergessen.

Funktion

Eine Funktion ist sehr ähnlich einem Funktionsbaustein, jedoch besitzen Funktionen im Gegensatz zu Funktionsbausteinen kein Gedächnis, d.h. dass dieselben Eingangswerte stets zu den selben Ausgangswerten führen, was bei Funktionsbausteinen - bedingt durch das innere Gedächnis - nicht zutrifft. Funktionen werden im Task des aufrufenden POEs abgearbeitet.

2.4.3 Programmiersprachen nach IEC-61131-3

Der IEC 61131 Standard definiert 5 sowohl textuelle als auch graphische Programmiersprachen[14]. Damit alle POEs unabhängig von der eingesetzten Programmiersprache miteinander kommunizieren können, sind für alle Sprachen die Datentypen gleich und deren Deklaration muss für allen Sprachen jeglichem Programmkode vorangestellt werden.

Auf die einzelnen Programmiersprachen wird hier jedoch nicht eingegangen, da dieses den Rahmen der Diplomarbeit sprengen würde und für diese auch

[14] Anweisungsliste (AWL), Kontaktplan (KOP), Ablaufsprache (AS), Strukturierter Text (ST), Funktionsbausteinsprache (FBS)

nicht von Relevanz sind. Im Hinblick auf die Integration von Bildverarbeitung in ein Automatisierungssystem mittels IEC 61131 sind die unterstützten Datentypen jedoch sehr wohl von Interesse, weshalb auf diese im nächsten Abschnitt näher eingegangen werden soll.

Von Interesse sind ebenfalls die Möglichkeiten zur Ein- bzw. Anbindung von Programmen und Bibliotheken, welche in anderen Programmiersprachen geschrieben wurden, z.B. Funktionsbausteine welche in C/C++ programmiert wurden.

An dieser Stelle muss gesagt werden, dass eine solche Anbindung seitens IEC 61131 nicht unterstützt wird, d.h. Softwaremodule, welche z.B. in den Programmiersprachen C/C++ oder Java geschrieben wurden, können nicht in IEC 61131-3 Programme eingebunden werden.

Moderne SPSen bzw. überhaupt Industrie-PCs unterstützen auch Programme in anderen (höheren) Programmiersprachen, parallel zu einer IEC 61131 Laufzeitumgebung (SoftPLC) zur Ausführung der IEC 61131 Programme. Auch eine Kommunikation zwischen diesen wird im Allgemeinen über herstellerspezifische Methoden unterstützt, jedoch ist diese nicht durch den IEC 61131 Standard abgedeckt.

D.h. dass vorhandene Bildverarbeitungsbibliotheken für eine der verfügbaren IEC 61131 Programmiersprachen übersetzt werden müssten, wenn Funktionsbausteine mit Bildverarbeitungsfunktionen verfügbar sein sollen. Hier wäre noch am ehesten die Programmiersprache *Structured Text* (ST) vorzuziehen, da diese am ehesten einer höheren Programmiersprache ähnelt.

Datentypen

IEC 61131-3 unterstützt alle gängigen Standard-Datentypen, wie Bits, Bytes, Integer mit oder ohne Vorzeichen, Fließkomma usw. Das Bit spielt aber im Gegensatz zu höheren Programmiersprachen wie z.B. Java usw. eine besondere Rolle. Bits können einfach ohne Maskierungsfunktionen direkt angesprochen werden. Darüber hinaus werden auch zwei explizite Datentypen für die fallenden und steigenden Flanken eines Bits definiert, d.h. für die Änderung von Bitwerten.

Auch Zeichenketten mit unterschiedlich langer Zeichenfolge werden unterstützt.

Funktionsbausteine werden ebenfalls als Datentypen gehandhabt und entsprechen im großen und ganzen dem *Objekt* in OOPs. Andere Merkmale von OOPs, wie Klassen, Polymorphismus, Vererbung oder Prototyping, werden jedoch nicht unterstützt.

Aus den verfügbaren Datentypen lassen sich neue Datentypen ableiten[15]. Aufzählungen[16] werden ebenfalls unterstützt. Mittels eines Bereichstypes lassen sich auch die Gültigkeitsbereiche einer Variablen einschränken.

Felder[17] (engl. Arrays) und Strukturen[18], welche für die Bildverarbeitung nicht fehlen dürften, werden - zwar mit leichten Einschränkungen - ebenfalls unterstützt. Die Länge der Felder, d.h. die Anzahl der Elemente, muss aber im vorhinein bekannt sein.

Ein frei verfügbarer Speicherbereich[19], in welchem Speicherblöcke beliebig reserviert bzw. freigegeben werden können, wird nicht unterstützt. Dies wäre für die Arbeit mit Bildern - welche wie schon im Unterabschnitt 2.2 erwähnt sehr speicherlastig sein kann - von großem Vorteil. Ohne diese Möglichkeit muss im vorhinein für jedes verwendete Bild der Speicher reserviert werden und könnte für nichts anderes eingesetzt werden.

Verteilte Software

Eine dezentrale Hardwarestruktur wird seitens des IEC 61131 Standards - besonders über die Feldbustechnologien - gut unterstützt. Eine dezentrale Softwarestruktur hingegen nicht. Die Verteilung einer großen Softwareapplikation auf mehrere SPSen muss vom Entwickler manuell erledigt werden, indem die Gesamtapplikation an günstigen Stellen aufgebrochen und die Kommunikation über Feldbus- bzw. sonstige Netzwerke erledigt wird.

Der Teil 5 des IEC 61131 behandelt im Detail die Kommunikation über Feldbusse und definiert auch Funktionsbausteine hierfür, welche jedoch praktisch in der Industrie nicht unterstützt werden.

2.4.4 Werkzeugarchitektur

Der IEC 61131 Standard macht weder Aussagen über die Erstellungswerkzeuge von Programmen auf dem Programmiersystem, noch über die Laufzeitumgebungen auf der SPS, d.h. die Schnittstellen von Editor, Kompiler usw. sind nicht festgelegt. Es wird lediglich festgeschrieben welchen syntaktischen Aufbau ein korrektes Programm laut Standard haben muss.

[15] ähnlich der *typedef* Deklaration in C/C++
[16] ähnlich dem *enum* in C/C++
[17] Zusammensetzung von Variablen gleichen Typs
[18] Zusammengesetzte Datentypen
[19] ähnlich dem Heap-Speicher in C/C++

Ein Nachteil hiervon ist, dass nicht einmal das Format, in welchem SPS Programme gespeichert und ausgetauscht werden, festgelegt wird, wodurch die Portierbarkeit von Programmen stark eingeschränkt ist, d.h. ein Programm, welches für eine SPS vom Hersteller A geschrieben wurde, nicht ohne weiteres für eine SPS vom Hersteller B verwendet werden kann.

Die meisten Hersteller - besonders die Marktdominierenden - bieten bereits sehr ausgereifte und hoch entwickelte Entwicklungsumgebungen an und besonders für die graphischen Programmiersprachen sehr benutzerfreundliche Werkzeuge. Jedoch müssen diese von Hersteller zu Hersteller neu erlernt werden.

2.4.5 Kompaktkamerasystem SBOx-C

Das *Kompaktkamerasystem SBOx-C* kann mittels der IEC 61131 Laufzeitumgebung CoDeSys[20] [wwwCDS] als eine SPS genutzt werden. Über einen *Shared Memory*[21] Mechanismus kann die CoDeSys-Laufzeitumgebung auch auf die Daten der Bilder zugreifen und es werden IEC 61131 Funktionsbausteine hierfür angeboten. Die eigentliche Verarbeitung und Informationsgewinnung aus den aufgenommenen Bilder erfolgt aber nicht in 61131, sondern wird in Form von C/C++ Funktionen in die CoDeSys Laufzeitumgebung eingebettet.

Ebenso können auch die digitalen Ein- und Ausgänge des Kompaktkamerasystems und der CAN Feldbus mittels der mitgelieferten IEC 61131 Funktionsbausteine genutzt werden.

2.5 IEC 61499

Der IEC 61499 Standard - *Function Blocks* - ist der Nachfolger des IEC 61131 Standards - *Programmable Controllers*. Im Folgenden ein Zitat von Robert Lewis [Lew01] - Mitarbeiter bei der Ausarbeitung des Standards - über die Motivation für einen neuen Standard für die Automatisierungstechnik:

There is currently an explosion in the use of object oriented (OO)

[20] CoDeSys - **Co**ntroller **De**velopment **Sy**stem - ist ein Programmiersystem zur Programmierung und Ausführung von IEC 61113-3 Software. Die CoDeSys Laufzeitumgebung kann für viele bzw. fast alle Zielsysteme übersetzt werden und macht somit aus jedem Rechner eine „Soft"-SPS.

[21] Unter *Shared Memory* versteht man einen Speicherblock im Arbeitsspeicher eines Rechners, welcher von mehreren Programmen gleichzeitig genutzt werden kann. Natürlich muss hierfür der Zugriff auf diesen gemeinsamen Speicher - meistens vom Betriebssystem - geregelt werden.

> software encapsulated as components. In business systems the use of software components and technology is becoming more widespread. In industrial systems, PLCsc and soft controllers based on PC architectures are also starting to adopt many of these techniques. The different worlds of factory automation and business systems are clearly starting to share the same software technology.[22]

womit er auf die Wichtigkeit der Kapselung und Wiederverwendbarkeit von Softwarebausteinen abzielt.

Als besonders gelungenes Beispiel einer solchen Software-Architektur möchte ich die ausschließlich objektorientierte Programmiersprache Java erwähnen, welche als Referenzprogrammiersprache für objektorientierte Programmierung gilt. Die Grammatik dieser Programmiersprache fördert und erzwingt die funktionsorientierte Kapselung von Softwareteilen zu Objekten und gerade diese hat sie innerhalb kürzester Zeit - trotz hoher Ressourcenanforderungen und damit verbundenen Performanceschwierigkeiten - zu einer der am häufigsten eingesetzten Programmiersprachen gemacht.

Der IEC 61499 Standard hat zwei Schwerpunkte, welche sich im Laufe der letzten Jahre bei seinem Vorgänger IEC 61131 als unzureichend berücksichtigt herausgestellt haben. Diese sind:

- der Komponenten Ansatz (*Funktionsblock*orientiert) und
- die dezentrale, verteilte Software-Architektur.

Während beim IEC 61131 Standard die eingesetzten Programmiersprachen im Teil 3 ganz genau definiert sind, beschreibt der IEC 61499 Standard nicht die Programmierung im Detail, als vielmehr ein Modell und ein Konzept zur Realisierung von funktionsblockorientierten, verteilten Programmen. Wie im einzelnen die eingesetzten Algorithmen zu programmieren sind, verbleibt beim Hersteller bzw. bei der Realisierung jeder einzelnen IEC 61499 Laufzeitumgebung.

[22] Zur Zeit findet eine regelrechte Explosion im Einsatz von als Komponenten gekapselter objektorientierter (OO) Software statt. In der elektrischen Datenverarbeitung ist der Einsatz von Software-Komponenten bereits sehr verbreitet. Nun beginnt auch die industrielle Automatisierung - SPSen und Mikrokontroller, welche auf Industrie-PCs basieren - sich dieser Techniken zu bedienen. Die unterschiedlichen Welten der Produktautomatisierung und der elektronischen Datenverarbeitung beginnen die selben Softwaretechnologien einzusetzen.

Im Gegensatz zu IEC 61131 macht der IEC 61499 Standard auch keine Aussagen über die eingesetzte Hardware. Jeder beliebige Rechner - ob ein spezieller Industrie-PC oder „Smart Device“ auf der Prozessebene oder eine Arbeitsstation (engl. Workstation) in der Leitebene - kann in eine verteilte Automatisierungsanlage einbezogen werden. Die Gesamt-Applikation wird - sowohl auf Hardware- als auch auf Softwareebene - auf viele Ausführungseinheiten aufgeteilt, wobei diese beliebig - auch direkt - untereinander kommunizieren können ohne eine zetrale Intelligenz zu benötigen, siehe Abb. 2.2, d.h. ein Sensor kann seine Sensordaten gleich selbst auswerten und damit direkt einen Aktor steuern.

2.5.1 Softwaremodell nach IEC 61499

Im Folgenden werden - in aller Kürze - die Modelle beschrieben, welche gemeinsam die IEC 61499 Architektur für Funktionsblock orientierte, verteilte Automatisierungssysteme beschreiben. Die nachfolgende Liste ist nicht vollständig, sondern es werden nur die für diese Diplomarbeit wichtigen Modelle vorgestellt. Für eine detailiertere Beschreibung wird auf das Buch von Valeriy Vyatkin [Vya07] verwiesen.

Systemmodell

Das Systemmodell, skizziert in Abb. 2.6, beschreibt die physikalischen Begebenheiten in einem Automatisierungssystem. Dies sind hauptsächlich die IEC 61499-fähigen Rechner (Devices) und deren Vernetzung.

Devicemodell

Ein Device ist eine Ausführungseinheit für IEC 61499 Funktionsblocknetzwerke. Im Allgemeinen besitzt sie auch E/A-Einheiten bzw. sonstige Peripherie, welche über die Device-Treiber gesteuert werden. Auf jeden Fall benötigt sie eine Kommunikationseinheit, z.B. Ethernet-Kontroller für die Einbindung in das Automatisierungssystem.

Die eigentliche Ausführung eines Funktionsblocksnetzwerks erfolgt nicht direkt im Device sondern in einer darunterliegenden logischen Einheit, der Ressource. Zur Laufzeit werden in einem Device Ressourcen angelegt, welche für die Kapselung und Ausführung von IEC 61499 Funktionsblöcken zuständig sind.

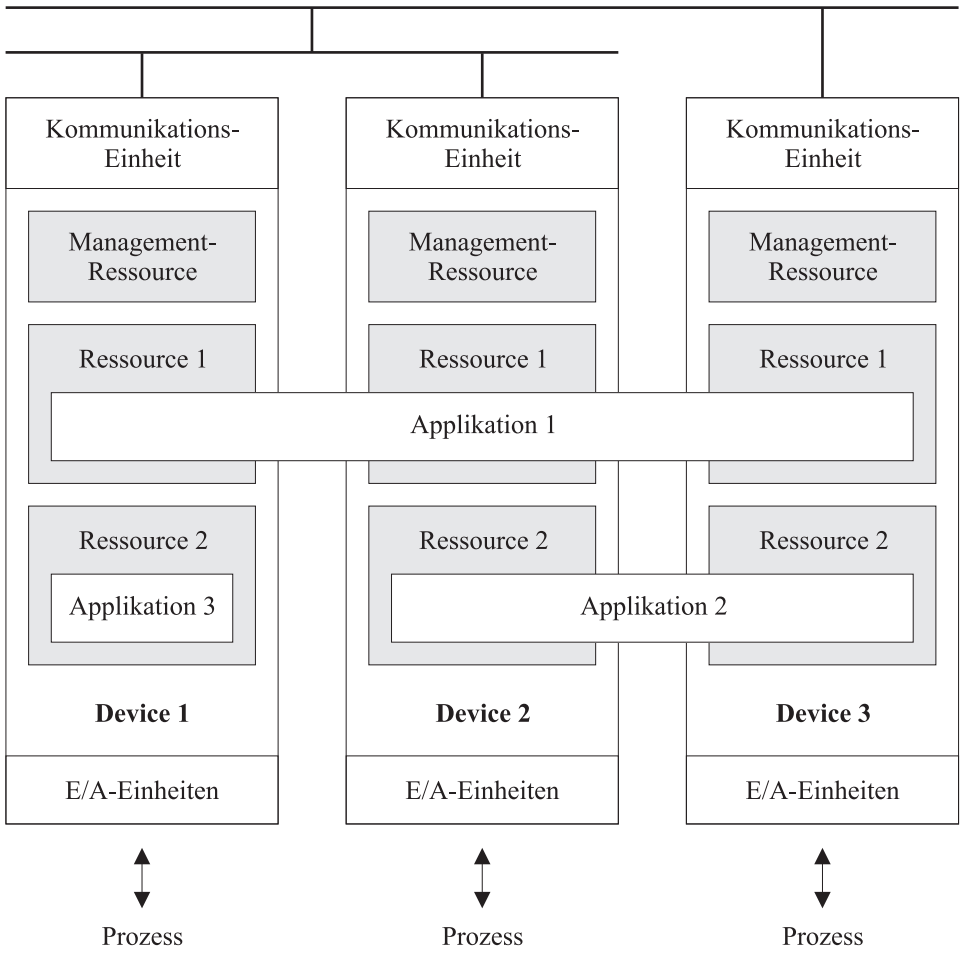


Abbildung 2.6: Systemmodell eines IEC 61499 Automatisierungssystems

Ressourcemodell

Innerhalb einer Ressource kann ein (Teil-)Funktionsblocknetzwerk - isoliert von etwaigen anderen Ressourcen im gleichen Device - ausgeführt werden. Mehrere Ressourcen teilen sich hierbei über die Treiber des Devices die E/A- bzw. sonstige Peripherieeinheiten und die Kommunikationseinheit.

Managementmodell

Jedes Device enthält eine fixe Ressource - die Managementressource, welche für die Managementaktivitäten zuständig ist. Diese Managementressource ist

zuständig für die Instanzierung sowie Beendigung von weiteren Ressourcen bzw. deren Funktionsblocknetzwerken.

Der IEC 61499 Standard definiert noch nicht im Detail alle Aufgaben der Managementressource. Sie könnte noch um viele weitere nützliche Features erweitert werden, z.B. eine Debug- Schnittstelle oder ein Reporting-Werkzeug. Oder sie könnte auf die Anfrage eines Entwicklungswerkzeugs hin mit einer Liste der unterstützten Funktionsblocktypen antworten.

Funktionsblockmodell

Ein Funktionsblock definiert sich durch seine Schnittstelle nach außen, bestehend aus sowohl Ereignisein- und -ausgängen als auch Dateneinein- und -ausgängen, und kann im eingebundenen Netzwerk wie eine Black-Box betrachtet werden, skizziert in Abb. 2.7.

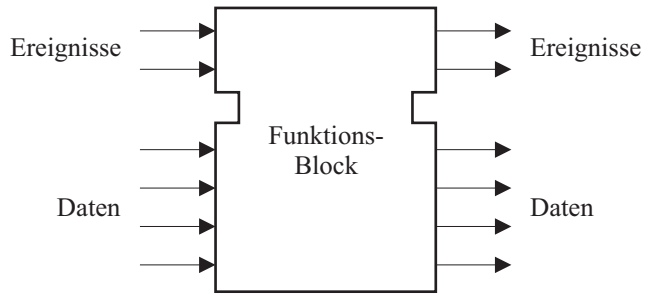


Abbildung 2.7: IEC 61499: Funktionsblockmodell

Basic Funktionsblock[23]
Ein Basic Funktionsblock besteht aus folgenden Komponenten:

- interne Daten:
  zusätzlich zu den Eingangsdaten, können interne Daten definiert werden.
- Algorithmen:
  Algorithmen sind erst die eigentlichen kodierten Programme. In welcher Programmiersprache diese programmiert sind, wird bewußt nicht vorgegeben, um den IEC 61499 Standard nicht gleich zu Beginn wieder einzuschränken. Je nach Laufzeitumgebung kann hier flexibel gearbeitet werden.

[23] deutsch: Basisfunktionsblock

Die implementierten Algorithmen müssen jedoch auf die Eingangs-, Ausgangs- und internen Daten zugreifen können und über den aktuellen Zustand und das auslösende Ereignis Bescheid wissen.

- ECC (Execution Control Chart):
  Hier wird mittels eines Zustandsgraphen der interne Zustand und die Zustandsübergänge des Funktionsblocks beschrieben. Der Wechsel zwischen den Zuständen erfolgt, ausgelöst durch eingehende Ereignisse, in Abhängigkeit von den Eingangs-, Ausgangs- bzw. internen Daten.

  An jedem Zustand kann die Ausführung von einem oder mehreren Algorithmen gekoppelt und Ausgangsereignisse angestoßen werden.

Composite Funktionsblock[24]
Ein Composite Funktionsblock ist von außen von einem Basic Funktionsblock nicht zu unterscheiden, jedoch enthält er intern ein Funktionsblocknetzwerk von weiteren Basic oder Composite Funktionsblöcken, anstatt der internen Komponenten des Basic Funktionsblocks.

Service Interface Funktionsblock
Ein Service Interface Funktionsblock ist ein reduzierter Basic Funktionsblock. Er enthält keinen ECC und keinerlei interne Vorgaben und darf aber in den auskodierten Algorithmen, hardwarespezifische Anweisungen enthalten. Somit ist ein Service Interface Funktionsblock an ein spezielles Device gebunden und kann nicht einfach so auf ein anderes übertragen werden.

Ein typisches Beispiel für Service Interface Funktionsblöcke sind die Kommunikations-Funktionsblöcke, welche auch von der Management-Ressource eingesetzt werden.

Applikationsmodell

Die Applikation stellt ein geschlossenes Funktionsblocknetzwerk dar, welches auf eines oder auf mehreren Ressourcen verteilt ausgeführt wird. Ein Funktionsblocknetzwerk besteht aus mehreren Funktionsblöcken, welche mittels Daten- und Ereignisleitungen miteinander verbunden werden. Ein Beispiel für ein ganz kleines demonstratives Netzwerk ist in Abb. 2.8 dargestellt.

Der Programmierer kann die Gesamtapplikation - das Funktionsblocknetzwerk - in einem Stück entwickeln und sich erst danach um die Verteilung auf die einzelnen Devices im Automatisierungssystem kümmern.

[24] deutsch: zusammengesetzter Funktionsblock

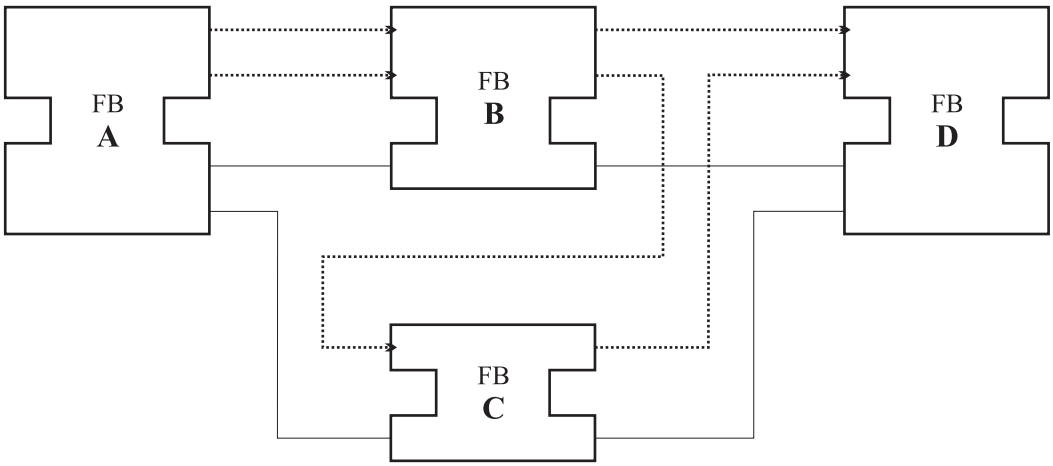


Abbildung 2.8: Beispiel für ein kleines IEC 61499 Funktionsblocknetzwerk

Sollten hierbei etwaige Daten- bzw. Ereignisleitungen aufgebrochen werden, könnte das Entwicklungswerkzeug dies detektieren und für die aufgebrochene Kante selbsttätig einen Sende- und einen Empfangsfunktionsblock auf den entsprechenden Ressourcen instanzieren.

2.5.2 Datentypen in IEC 61499

Bei der Definition der Funktionsblöcke müssen für die Daten Ein- bzw. Ausgänge die unterstützten Datentypen bekannt sein. Diese sind identisch zum IEC 61131 Standard mit einer besonderen Erweiterung, nämlich dass diese unter Einsatz von zusätzlichen sogenannten *generischen* Datentypen in eine hierarchische Struktur gebracht wurden, dargestellt in Abb. 2.9.

Ein Beispiel für einen generischen Datentyp ist z.B. `ANY_NUM`, welche die Menge aller Zahlen repräsentiert, d.h. 16-Bit, 32-Bit und 64-Bit Ganz- sowie Fließkommazahlen sowohl mit als auch ohne Vorzeichen. Ein Funktionsblock, welcher einen Eingangsparameter vom Typ `ANY_NUM` erwartet, akzeptiert somit alle Zahlendatentypen, z.B. `REAL`, `UINT` usw.

Die Einführung von neuen Datentypen ist möglich, jedoch sollten hier Erweiterungen des Standards angestrebt werden, anstatt beliebig und unkontrolliert herstellerabhängige Datentypen zu produzieren. Zur Einführung von Bildverarbeitungsfunktionen wird dies aber notwendig sein.

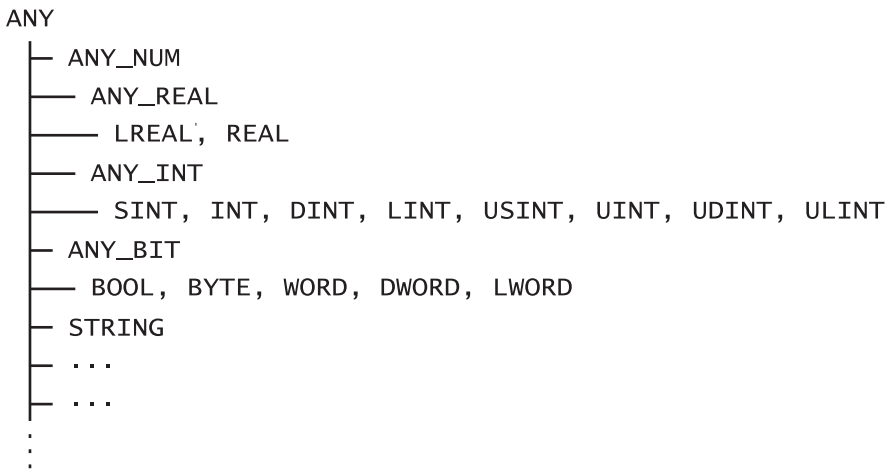


Abbildung 2.9: Hierarchische Ordnung der IEC 61499 Datentypen

2.5.3 Laufzeitumgebungen für IEC 61499

Eine IEC 61499 Laufzeitumgebung macht aus der Hardware erst ein IEC 61499 *Device*, welches zum Ausführen von IEC 61499 (Teil-)Netzwerken verwendet werden kann. Die Hardware wird hierbei völlig abstrahiert und nur noch die Service Interface Funktionsblöcke stellen die physikalische Verbindung zu dieser her.

Diese vollständige Abstraktion der Hardware ist der Grund, warum der IEC 61499 Standard keine Aussagen über die Hardware zu machen braucht. Für die vielen heterogenen Mikrokontroller und Rechnerarchitekturen müssen die Hersteller lediglich IEC 61499 Laufzeitumgebungen und hardwarespezifische Service Interface Funktionsblöcke anbieten, dann kann die Hardware vía *Plug & Play*[25] in das IEC 61499 Automatisierungssystem angeschlossen werden.

Jede Laufzeitumgebung unterstützt, limitiert durch seine Ressourcen, nur eine gewisse Anzahl von Funktionsblöcken. Eine Bildverarbeitungsanwendung könnte aber seine Berechnungen auch auf mehrere *Devices* aufteilen.

Für diese Diplomarbeit wurden die folgenden zwei IEC 61499 Laufzeitumgebungen genutzt:

- FBRT (**F**unction **B**locks **R**unt**i**me) von Holobloc Inc. [wwwHB]

[25] „Anstecken und Spielen“ bzw. „Anschließen und Loslegen“, auch „Plug 'n' Play“ genannt, ist ein Begriff aus dem Gebiet der Computertechnologie mit dem man die Eigenschaft eines Computers beschreibt, neue Geräte - meist Peripheriegeräte - anschließen zu können, ohne anschließend Treiber installieren oder andere Einstellungen vornehmen zu müssen.

- MARTE (**M**icrons[26] **A**dvanced **R**un**t**ime **E**nvironment) [wwwMNS] als Zusammenarbeit zwischen der Firma PROFACTOR, vom Automatisierungstechnikinstitut der Fachhochschule Oberösterreich - Campus Wels und vom Institut für Automatisierungs- und Regelungstechnik der Technischen Universität Wien.

Die FBRT wurde in der Programmiersprache Java programmiert und läuft hauptsächlich auf PCs (sowie für Java optimierte Mikrokontroller, die direkt Java-Bytecode interpretieren), während für die MARTE C++ verwendet wurde und sie auch auf ressourcenarme Mikrocontroller abzielt.

Die MARTE wurde mittlerweile im Rahmen des Projekts 4DIAC - *Open Source for Distributed Industrial Automation* - [www4DC] weiterentwickelt und ist unter dem Namen FORTE (4DIAC-RTE) frei verfügbar.

Ereignisabarbeitung

Über einige wichtige Fragen macht der IEC 61499 Standard zur Zeit noch keine Aussagen. Eine davon ist die Abarbeitungsreihenfolge der aufgetretenen Ereignisse bzw. über die Handhabung von nicht verarbeiteten Ereignissen, da aufgrund des internen Zustandes eines Funktionsblocks, dieser das Ereignis nicht verarbeiten konnte bzw. wollte.

Eine Priorisierungsmöglichkeit bestimmter Ereignisse in einem Funktionsblocknetzwerk oder, noch allgemeiner, auch in einem Device, sind ebenfalls noch offene Punkte. Die Frage nach Echtzeitfähigkeit, aufgrund der nicht definierbaren Durchlaufzeiten durch die einzelnen Funktionsblöcke, gehört ebenfalls noch behandelt. Zum Thema Echtzeitfähigkeit für die Laufzeitumgebung MARTE wird an dieser Stelle auf die Arbeiten von Alois Zoitl [Zoi06] verwiesen.

Die beiden erwähnten Laufzeitumgebungen verfolgen in diesem Punkt unterschiedliche Strategien.

In der FBRT wird ein aufgetretenes Ereignis solange über Funktionsblöcke hinweg verfolgt, bis es versiegt, d.h. bis ein Funktionsblock angestoßen durch ein Ereignis selbst kein neues Ereignis mehr auslöst. Dies bedingt aber, dass

[26] µCrons; Das µCrons Projekt hat sich mit der Entwicklung einer Middleware (deutsch Zwischenanwendung) für integrierte Systeme (engl. Embedded Systems) beschäftigt, um die Software-Entwicklung für heterogene Automatisierungskomponenten in der industriellen Automatisierung zu vereinfachen. Anstatt jede „intelligente“ Komponente (Smart Device) frei programmieren zu müssen, soll somit die Programmierung stark abstrahiert und funktionsorientiert geschehen. Im Rahmen dieses Projektes wurde auch die IEC 61499 Laufzeitumgebung MARTE entwickelt.

ein initiiertes Ausgangsereignis nicht als Eingang wieder zurückkommen darf, d.h. nicht im Kreis laufen darf. Löst ein Funktionsblock zwei bzw. mehrere Ausgangsereignisse aus, so wird auf diese Weise garantiert, dass zuerst der erste vollständig abgearbeitet wurde und danach erst der nächste startet.

Die MARTE wiederum verfolgt eine FIFO[27] Strategie für die Ereignisabarbeitung. Ein Ausgangsereignis wird in einer Ereignisliste hinten angehängt und die Ereignisliste selbst wird, unabhängig vom Funktionsblock, der Reihe nach abgearbeitet. Löst ein Funktionsblock mehrere Ausgangsereignisse aus, so werden diese „quasi“ parallel abgearbeitet. Durch Multithreading[28] können aber auch mehrere Ereignisse tatsächlich gleichzeitig abgearbeitet werden bzw. mehrere solcher Ereignisketten getrennt voneinander geführt werden.

Weitere aufgetretene Fragen in Bezug auf die Reihenfolge der Ereignisabarbeitung in einem IEC 61499 Funktionsblocknetzwerk werden in [Sun06] behandelt.

Datenweitergabe

Die Weitergabe von Daten von Funktionsblock zu Funktionsblock ist ebenfalls ein noch nicht genau definierter Punkt im IEC 61499 Standard. Denn die weitergegebenen Daten können als Kopie oder nur als Referenz weitergegeben werden.

Eine physische Kopie ist langsamer und speicherlastiger, da jedes weitergegebene Datum zuerst im Speicher kopiert werden muss. Dafür hält jeder Funktionsblock zu jeder Zeit seine eigenen Daten und ist unabhängig von der Gültigkeitsdauer einer Referenz. Beim Einsatz von Bildern als Daten ist diese Form der Datenweitergabe jedoch ganz unpraktikabel, da selbst für ressourcenstarke Rechner die vielen mehrfach gehaltenen Bilder schnell eine Speicherknappheit bewirken.

Die Weitergabe der Daten als Referenz ist wesentlich schneller und jedes Datum wird nur einmal im Speicher gehalten, nämlich als Ausgang eines Funktionsblocks. Dafür muss aber die Gültigkeit der Referenz auf das Datum gewährleistet werden. Ein Funktionsblock muss wissen, ob der ihm nachfolgende Funktionsblock die Daten noch benötigt oder bereits freigegeben hat, was wiederum einen extra Verwaltungsaufwand mit sich bringt.

Die verwendete Form der Datenweitergabe hängt stark von der Art der Er-

[27] **F**irst **I**n - **F**irst **O**ut

[28] logisch - bzw. für Multiprozessormaschinen echt - paralelle Ausführung von mehreren Programmteilen

eignisabarbeitung ab. Für das sequentielle Modell der FBRT ist die Weitergabe der Daten als Referenz möglich und zu empfehlen, während für die FIFO Ereignisabarbeitung der MARTE eher die Datenweitergabe als Kopie in Frage kommt.

Bei der Entwicklung neuer Funktionsblöcke - wie ihm Rahmen dieser Diplomarbeit für bildverarbeitende Funktionsblöcke gefordert wird - müssen die gewählten Strategien der darunterliegenden Laufzeitumgebungen berücksichtigt werden.

2.5.4 Werkzeugarchitektur für IEC 61499 Programmierung

IEC 61499 Programme oder Funktionsblöcke können sowohl grafisch als auch textuell erstellt werden. Über die eingesetzten Werkzeuge macht der Standard keine Aussagen, jedoch - als Verbesserung seines Vorgängers - definiert er das Speicherformat, wodurch die Austauschbarkeit der Entwicklungsumgebungen und der eingesetzten Werkzeuge gewährleistet wird.

Da der IEC 61499 Standard noch sehr jung ist, sind die angebotenen Entwicklungsumgebungen[29] noch spärlich und minimalistisch. Zum Entstehungszeitpunkt der Diplomarbeit war FBDK[30] - obwohl selbst noch in Kinderschuhen steckend - das am fortschrittlichste, frei verfügbare Programmierwerkzeug.

Von der Firma ICS Triplex - eine Tochterfirma von Rockwell Automation - unterstützt das Entwicklungswerkzeug ISaGRAF [wwwISG] ab Version 5 ebenfalls den IEC 61499 Standard.

Besonders für die grafische Programmierung ist aber das Vorhandensein von guten Entwicklungswerkzeugen absolut notwendig. Da das Speicherformat XML basierend ist, kann aber die textuelle Programmierung mittels jedes beliebigen XML Editors erfolgen.

Zum Zeitpunkt der Diplomarbeit waren noch keine Debug und Profiling Werkzeuge für IEC 61499 verfügbar. Das Debugging war nur mittels Werkzeugen der darunter liegenden Programmiersprache, also der Programmiersprache der Laufzeitumgebung selbst, möglich.

[29] engl. IDE, Integrated Development Environment
[30] **F**unction **B**locks **D**evelopment **K**it

2.5.5 Kompaktkamerasystem SBOx-C

Seitens des Kompaktkamerasystems wird zumindest zum Zeitpunkt der Auslieferung keine IEC 61499 Laufzeitumgebung mitgeliefert.

Jedoch ist das *Kompaktkamerasystem SBOx-C* ein frei programmierbarer - auf einem Linux Betriebssystem basierender - Rechner mit einer ARM Prozessorarchitektur und somit kann jedes beliebige Programm, welches sich für eine ARM Plattform mit Linux übersetzen lässt - so auch z.B. die IEC 61499 Laufzeitumgebung MARTE - kompiliert und ausgeführt werden, wodurch das *Kompaktkamerasystem SBOx-C* als ein IEC 61499 Device in ein bestehendes Automatisierungssystem integriert werden kann.

2.6 Zusammenfassung

Im vorangegangenen Kapitel wurde

- der viel benutzte Ausdruck „verteilte Steuerung“ genauer eingegrenzt,
- der Stand der Technik der industriellen Bildverarbeitung analysiert,
- die Standards IEC 61131 und IEC 61499 in Hinblick auf deren Eignung zur Bildverarbeitung beschrieben
- und ein neues Produkt (*Smart Device*) aus dem Hause FESTO - *das Kompaktkamerasystem SBOx-C* - vorgestellt.

3 Konzept zur Integration von Bildverarbeitung in IEC 61499

Bevor das Konzept zur Intergration von Bildverarbeitung in IEC 61499 ausgearbeitet wird, werden die wesentlichen definierenden Merkmale der beiden involvierten Technologien, Bildverarbeitung und IEC 61499 gegenübergestellt.

Bildverarbeitung	**IEC 61499**
Daten sind Bilder	Daten sind Standarddatentypen des IEC 61131-3 Standards
komplexe Algorithmen	einfache Operationen
sequentielle Datenverarbeitung, d.h. das Datum (also das Bild) wird schrittweise weiterverarbeitet und analysiert	parallele, verteilte, ereignisorientierte Verarbeitung
hohe Ressourcenanforderungen	Verarbeitung auch auf Mikrocontrollern mit wenig verfügbaren Ressourcen

Tabelle 3.1: Gegenüberstellung der Merkmale von Bildverarbeitungs- und IEC 61499 Anwendungen

Aus dieser Gegenüberstellung und dem im vorangehenden Kapitel dargelegten Stand der Technik ergeben sich folgende zu lösende Aufgaben:

- Einführung neuer Datentypen in IEC 61499.
  Der wichtigste neu zu implementierende Datentyp ist das Bild (engl. Image), aber auch weitere Datentypen, wie Bildmasken, Paletten usw. sind notwendig.

- Datenweitergabe der neuen Datentypen über IEC 61499 Datenverbindungen.
  Dies soll sowohl als Referenz (besonders geeignet innerhalb einer Ressource) wie auch als Kopie (notwendig für Übertragungen über das Netzwerk) implementiert werden, unabhängig von der darunterliegenden IEC 61499 Laufzeitumgebung.

- Erstellung von bildverarbeitenden Funktionsblöcken für IEC 61499.
  Diese sollen weitestgehend unabhängig von der Laufzeitumgebung implementiert werden.

Das Lösungskonzept erfolgt in folgenden Schritten.

Zuerst wird auf die Integration des *Kompaktkamerasystems SBOx-C* in ein Automatisierungssystem basierend auf IEC 61499 eingegangen.

Danach wird eine Marktrecherche nach einer externen Bildverarbeitungsbibliothek durchgeführt, um passende Bildverarbeitungsfunktionen auszuwählen. Wie bereits am Ende vom Abschnitt 2.2 angemerkt, ist es nicht Ziel dieser Diplomarbeit die Bildverarbeitungsfunktionen selbst *neu* zu entwickeln, sondern auf bereits vorhandene Bibliotheken zurückzugreifen. Wie jedoch deren Einbindung in die IEC 61499 Laufzeitumgebung erfolgt, wird im Abschnitt 3.2.1 konzipiert und die anschließende Implementation im Abschnitt 4.2 ausgeführt.

Abschließend folgen die IEC 61499 spezifischen Lösungsansätze:

1. Strategien zur Einbindung externer Bibliotheken in IEC 61499 Laufzeitumgebungen.

2. Definition notwendiger Datentypen für IEC 61499.

3. Ausarbeitung und Gruppierung der zu entwickelnden IEC 61499 Funktionsblöcke.

4. Spezifikation einer Demonstrationsanwendung.

3.1 Externe Bildverarbeitungsbibliothek(en)

Zur Zeit existiert eine sehr große Anzahl von - sowohl frei verfügbaren als auch kostenpflichtigen - Bildverarbeitungsbibliotheken, geschrieben in den unterschiedlichsten Programmiersprachen und für den Einsatz in vielen verschiedenen Gebieten. Für diese Diplomarbeit soll auf eine der frei verfügbaren zurückgegriffen werden und die einzelnen Funktionen in IEC 61499 Funktionsblöcke gekleidet werden.

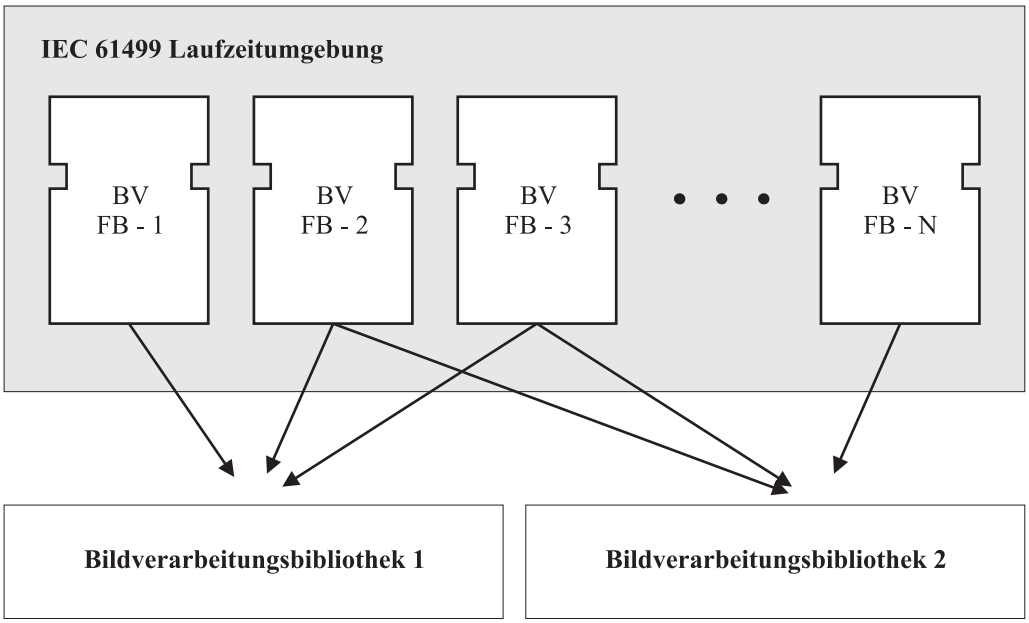


Abbildung 3.1: Einbindung einer oder mehrerer externer Bildverarbeitungsbibliotheken in eine IEC 61499 Laufzeitumgebung zur Kapselung der Bildverarbeitungsfunktionen in IEC 61499 Funktionsblöcke. (BV... Bildverarbeitung, FB... Funktionsblock)

Dies geschieht, indem die gewählte Bildverarbeitungsbibliothek - abhängig von der Programmiersprache - in die IEC 61499 Laufzeitumgebungen eingebunden wird und von den instanzierten Funktionsblöcken genutzt wird. Dadurch bleibt die gesamte Bildverarbeitungfunktionalität in den externen Bibliotheken und nur die Schnittstelle zu diesen muss definiert werden. Ein Blockschaltbild von diesem Ablauf ist in Abb. 3.1 skizziert.

Bei der Recherche nach einer passenden Bibliothek muss diese folgende Mindestanforderungen erfüllen:

- **frei verfügbar**
  Für diese Diplomarbeit, als ein akademisches Referenzprojekt zur Integration von Bildverarbeitung in eine IEC 61499 Umgebung, ist es nicht notwendig auf kommerzielle Bibliotheken zurückzugreifen. Für einen Einsatz in einer industrierelevanten Anwendung soll es aber möglich sein, die gewählte Bibliothek zu ersetzen.

- **Quellkode verfügbar**
  Sollten bei der Einbindung in die IEC 61499 Laufzeitumgebung leichte Anpassungen an der Bibliothek notwendig sein, so ist die Verfügbarkeit des Quellkodes unumgänglich.

- **kompilierbar für verschiedene Rechnerarchitekturen**
  Die IEC 61499 Laufzeitumgebung soll auf unterschiedlichsten Systemen - 16-Bit, 32-Bit und 64-Bit, ARM und x86 Prozessoren, Windows bzw. Linux Betriebssysteme usw. - lauffähig sein, weshalb auch die Bildverarbeitungsbibliothek hierfür geeignet sein muss.

- **programmiert in C/C++**
  Für den Einsatz auf Mikrocomputer ist eine plattformunabhängige[1] *low level*[2] Programmiersprache zu bevorzugen. Für IEC 61499 Laufzeitumgebungen, welche in Hochsprachen wie z.B. Java implementiert sind, z.B. FBRT, kann auf Schnittstellenfunktionen der jeweiligen Hochsprache zurückgegriffen werden, um auch auf externe plattformabhängige Bibliotheken zuzugreifen, z.B. für die Programmiersprache Java, ist dies das JNI[3] Paket.

- **gute Dokumentation**
  Die Bildverarbeitungsbibliothek und ihre Funktionen sollen gut dokumentiert sein.

- **breites Funktionsspektrum**
  Zumindest die gängigsten Standardfunktionen für eine Bildverarbeitungsanwendung sollen bereits integriert sein.

- **erweiterbar**
  Der Aufbau und die interne Struktur der Bildverarbeitungsbibliothek soll die Erweiterung um neue Funktionen ermöglichen.

- **„lebendes" Projekt, optimal mit Ansprechpartner**
  Die Bildverarbeitungsbibliothek soll ein aktives, gewartetes Projekt sein, an welchem weiterhin entwickelt wird, d.h. etwaige Fehler behoben werden und immer wieder neue Funktionen hinzugefügt werden.

[1] sofern dies überhaupt möglich ist

[2] Unter einer *low-level* Programmiersprache wie z. B. Assembler oder C, ist in diesem Zusammenhang eine Programmiersprache gemeint, welche hardwarenahe Anweisungen zulässt und nach dem Übersetzen für ein Zielsystem, ohne Betriebssystem oder einer Laufzeitumgebung, ausgeführt werden kann.

[3] Java Native Interface

> Weiters wäre ein Ansprechpartner - ein Betreuer für das Projekt - von großem Vorteil falls Fragen und Anregungen auftauchen sollten.

Hauptsächlich wurden die Recherchen über das Internet - mittels diverser Suchmaschinen, allen voran *Google* - getätigt. Aber auch Empfehlungen von Kollegen der Firma FESTO - Abteilung für Bildverarbeitung bzw. der Forschungsgruppe *Machine Vision* vom Automatisierungstechnikinstitut der Technischen Universität Wien, wurden in Betracht gezogen.

Das breite Angebot wurde aufgrund der eben genannten Anforderungen stark reduziert und in Frage kamen nur folgende Bibliotheken:

- OpenCV (Intel Open Source Computer Vision Library [wwwOCV])
- Gandalf (The Fast Computer Vision and Numerical Library [wwwGND])
- IM (An Imaging Toolkit [wwwIM])

Diese wurden im Detail betrachtet und untersucht, da sie - mit wenigen Einschränkungen - alle die gestellten Anforderungen erfüllten.

Die OpenCV Bildverarbeitungsbibliothek von Intel ist sehr komplex und bietet sehr fortgeschrittene Funktionen, z.B. Gesichtserkennung, Bewegungsverfolgung usw. Unter den frei verfügbaren ist sie jedenfalls die fortschrittlichste Bildverarbeitungsbibliothek. Dementsprechend ist sie sehr groß und ressourcenlastig und erfordert ein ausgiebiges Studium der Dokumentation bzw. der verfügbaren Beispielanwendungen.

Die Gandalf Bibliothek ist ursprünglich als Bibliothek für mathematische Funktionen, z.B. Matrizenrechnung usw. entstanden und da die Bildverarbeitungsoperationen im Allgemeinen selbst sehr mathematiklastig sind, war es naheliegend die Gandalf Bibliothek auch um Bildverarbeitungsfunktionen zu ergänzen. Sie ist besonders schnell, da anstatt der Standard mathematischen C/C++ Funktion die eigenen, für Geschwindigkeit optimierten, genutzt werden. Dafür sind die Bildverarbeitungsfunktioner eher nur für einfache Operationen ausreichend und komplexere, weiter abstrahierte stehen leider noch keine zur Verfügung.

Gewählt wurde aber die Bibliothek *IM - An Imaging Toolkit* - von Antonio Escaño Scuri [wwwIM]. Sie hat aufgrund einer ganz intuitiven Bedienung, d.h. ohne große Einarbeitungszeit, und klarer, übersichtlicher Dokumentation und eine Vielzahl von Bildverarbeitungsfunktionen überzeugt. Ergänzend zur Bildverarbeitungsbibliothek selbst wird eine ausführbare Anwendung mitgeliefert, welche die Funktionen der Bibliothek nutzt, um diese im praktischen Einsatz

zu demonstrieren. Damit hat sie auch sehr hohen didaktischen Wert und schien am besten für die Anforderungen dieser Diplomarbeit geeignet.

Eine zur Muster-(Bild-)erkennung oft eingesetzte Technik, sind die Neuronalen Netzwerke. Die gewählte Bildverarbeitungsbibliothek unterstützt jedoch keine Neuronalen Netzwerke. Um das Funktionsspektrum der Bildverarbeitungsbibliothek um diese *State of the Art* Technologie zu erweitern, wurde eine zweite externe Bibliothek für Neuronale Netzwerke, *FANN*[4] [wwwFN] hinzugenommen.

Beide Bibliotheken sind *Open-Source* nach *GPL*[5] und in den Programmiersprachen C/C++ geschrieben. Sie eignen sich durch die interne Strukturierung bzw. der einhergehenden Tool-Chain auch besonders gut für unterschiedlichste Computerplattformen.

Wie genau der Zugriff auf diese Bilbiotheken aus einer IEC 61499 Laufzeitumgebung - sowohl MARTE als auch FBRT - erfolgen soll, wird im Folgenden Abschnitt beschrieben.

3.2 IEC 61499

In diesem Abschnitt wird ein Konzept vorgelegt, wie Bildverarbeitungsfunktionsblöcke in ein Automatisierungssystem, basierend auf IEC 61499, eingeführt werden können.

Der erste Unterabschnitt behandelt die Einbindung von externen Bibliotheken in eine IEC 61499 Laufzeitumgebung, wobei hier zwischen MARTE und FBRT unterschieden wird. Der nachfolgende Unterabschnitt beschreibt ein Konzept, wie die Bilderfunktionen und -Daten in IEC 61499 eingeführt werden können, um die am Anfang von Kapitel 3 beschriebenen Probleme beherrschen zu können. Hierzu werden zwei Lösungskonzepte, eine zentrale Bilderablage und eine Abstraktionsschicht zwischen IEC 61499 und den externen Bibliotheken, vorgestellt. Danach wird ein kurzer Überblick über die zu implementierenden Bildverarbeitungsfunktionsblöcke gegeben und zum Abschluss eine mögliche Applikation zur Demonstration der Ergebnisse vorgestellt.

[4] **F**ast **A**rtificial **N**euronal **N**etworks

[5] GNU **G**eneral **P**ublic **L**icense (GPL) ist eine von der Free Software Foundation herausgegebene Lizenz für die Lizenzierung freier Software

3.2.1 Zugriff auf externe C/C++ Bibliotheken aus IEC 61499

Die Einbindung von externen Bibliotheken, muss für jede zu unterstützende IEC 61499 Laufzeitumgebung einzeln erfolgen und kann leider nicht verallgemeinert werden. Soll eine IEC 61499 Laufzeitumgebung ebenfalls für verschiedene Computerplattformen übersetzt werden, so muss auch noch zwischen den unterschiedlichen Zielplattformen unterschieden und auch die externe Bibliothek darauf vorbereitet werden.

Im Folgenden wird das Konzept zur Einbindung einer externen Bibliothek - programmiert in C/C++ - getrennt für die beiden, im Unterabschnitt 2.5.3 vorgestellten, IEC 61499 Laufzeitumgebungen, FBRT und MARTE, beschrieben.

FBRT

Die FBRT ist in der Programmiersprache Java geschrieben und ist somit plattformunabhängig und für jeden Computer mit einer kompatiblen Java Laufzeitumgebung[6] geeignet. Eine einzubindende Bibliothek, welche in C/C++ programmiert wurde, ist jedoch plattformabhängig und muss speziell für die gewählte Zielplattform übersetzt werden. Nach erfolgreichem Erstellen der Bibliothek für die gewünschte Plattform, muss diese für den Zugriff aus der Java-Laufzeitumgebung vorbereitet werden. Die Programmiersprache Java - ab der Version 1.4 - bietet für den Zugriff auf plattformabhängige externe Bibliotheken ein eigenes Paket an, das **J**ava **N**ative **I**nterface (JNI).

Nach kurzem Studium der JNI Dokumentation [Lia99] wird aber klar, dass die empfohlene Einsatzmethode des JNI Pakets für diesen konkreten Fall unpraktikabel ist. Das JNI schreibt nämlich vor, dass für jede C/C++ Funktion, welche aus einem Java Programm heraus, eingesetzt werden soll, sowohl für C/C++ als auch für Java, eine Zwischenfunktion[7] definiert werden muss. Diese „übliche“ Zugriffsvariante aus einem Java Programm heraus auf C/C++ Bibliotheken, ist in Abb. 3.2 graphisch veranschaulicht. Das native Schlüsselwort vor den Funktionen der Java Klasse bedeutet, dass es sich bei dieser Funktion um eine externe, d.h. nicht in Java implementierte Funktion, handelt.

Diese Methode würde aber erstens einen großen Zeitaufwand bedeuten, um

[6] Für Java wird eher der Ausdruck virtuelle Maschine (engl. Virtual Machine) als Laufzeitumgebung (engl. Runtime) verwendet.

[7] Im Programmierfachjargon ist für eine solche Zwischenfunktion der Ausdruck *Wrapper* gebräuchlich, für welchen es aber keine gleichwertige deutsche Entsprechung gibt.

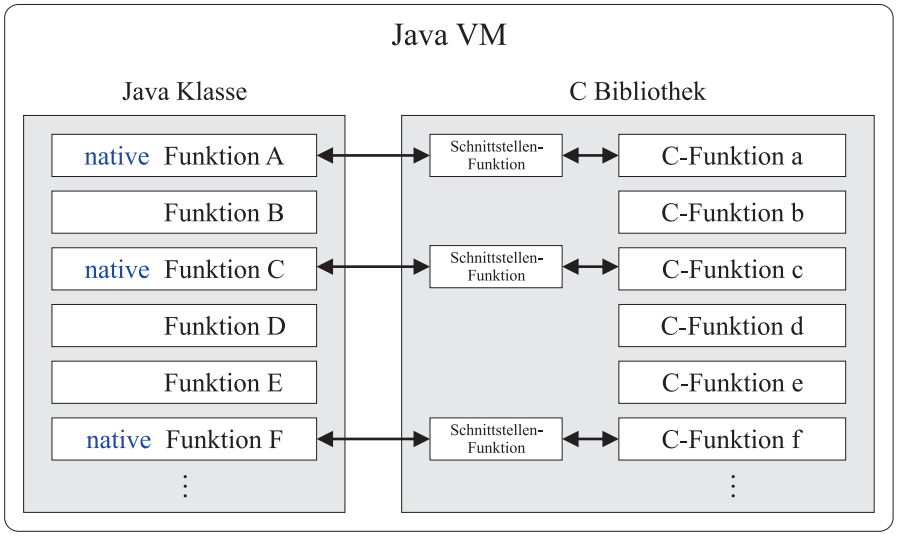


Abbildung 3.2: Zugriff auf eine C/C++ Bibliothek aus einem Java Programm heraus mittels des JNI-Pakets.

jede eingesetzte C/C++ Funktion zu umhüllen (engl. to wrap) und zweitens, was bei weitem unangenehmer ist, Änderungen im Quellkode der C/C++ Bibliotheken bedingen. Sollten weitere Versionen der eingesetzten Bibliotheken erscheinen, so müssten die getätigten Erweiterungen auch für die nachfolgenden Versionen erneut durchgeführt werden.

Um eine externe C/C++ Bibliothek in die FBRT einzubinden, wird deshalb ein anderer Weg - *Shared Stubs* [Lia99] genannt - eingeschlagen. Diese Zugriffsweise ist in Abb. 3.3 graphisch dargestellt. Mit Hilfe einer zusätzlichen externen C Bibliothek wird eine einzige Zwischenfunktion definiert, welche genutzt wird, um jede beliebige C/C++ Funktion einer jeden beliebigen C/C++ Bibliothek erst zur Laufzeit, d.h. erst bei Bedarf während der Programmausführung aufrufen zu können. Diese Zwischenfunktion bedient sich dabei „low-level“ Methoden (programmiert in Assembler), um die geforderten Funktionen bereitzustellen.

In der Java Klasse sind jetzt die externen Funktionen nicht mehr mit dem Schlüsselwort *native* deklariert, sondern bedienen sich einer Java-Hilfsklasse, welche die externen Zugriffe verwaltet. Diese Java-Hilfsklasse wurde auf Grund der besseren Übersicht nicht eingezeichnet.

Für den Zugriff mittels der JNI Schnittstelle muss somit nur noch eine einzige C/C++ Funktion vorbereitet werden, welche als Parameter den Namen

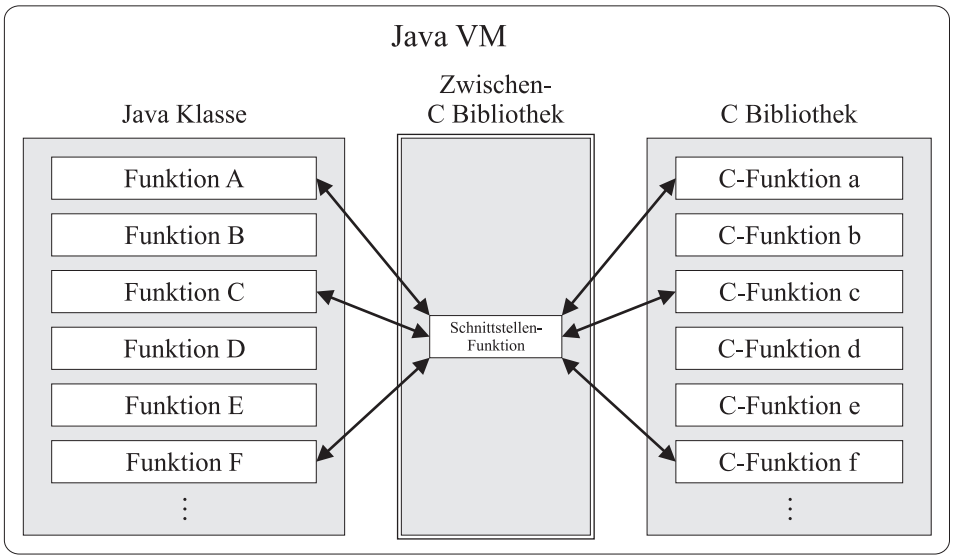


Abbildung 3.3: Zugriff auf eine C/C++ Bibliothek aus einem Java Programm heraus mittels der *Shared-Stubs* Methode.

der externen Bibliothek, den Funktionsnamen und die Funktionsparameter erhält, die genannte Bibliothek ladet, die adressierte Funktion ausführt und das Ergebnis zurückliefert.

Durch diese Technik kann vermieden werden, dass Änderungen an externen C/C++ Bibliotheken getätigt werden müssen. Besonders in den Fällen, wo ein Programmquellkode gar nicht verfügbar ist, ist dies sowieso die einzige Möglichkeit, C/C++ Funktionen für Java Programme verfügbar zu machen.

MARTE

Die MARTE ist selbst in C/C++ programmiert, wodurch der Zugriff auf weitere C/C++ Bibliotheken keine Schwierigkeit darstellt. Für jede gewählte Zielplattform müssen aber sowohl MARTE als auch die externe Bibliothek neu übersetzt werden.

3.2.2 IEC 61499 Datentyp für Bilder

Für die Representation von Bildern in IEC 61499 Funktionsblocknetzwerken muss ein neuer IEC 61499 Datentyp definiert werden.

Hierbei handelt es sich um einen zusammengesetzten Datentyp, der aus einem *Header*[8] und einem *Body*[9] besteht. Der *Header* hat - unabhängig vom Bild - eine fixe Länge und beinhaltet Meta-Informationen über das Bild wie z.B. Bildgröße. Der *Body* ist in der Länge variabel, abhängig von der Bildgröße, der Farbanzahl und Farbtiefe und enthält die tatsächlichen Bilddaten (Pixelinformationen).

Bilderablage

Die Speicherung der Bilder soll in einer zentralen Bilderablage (engl. Container) innerhalb der IEC 61499 Laufzeitumgebung erfolgen. Die Funktionsweise der Bilderablage ist in Abb. 3.4 veranschaulicht.

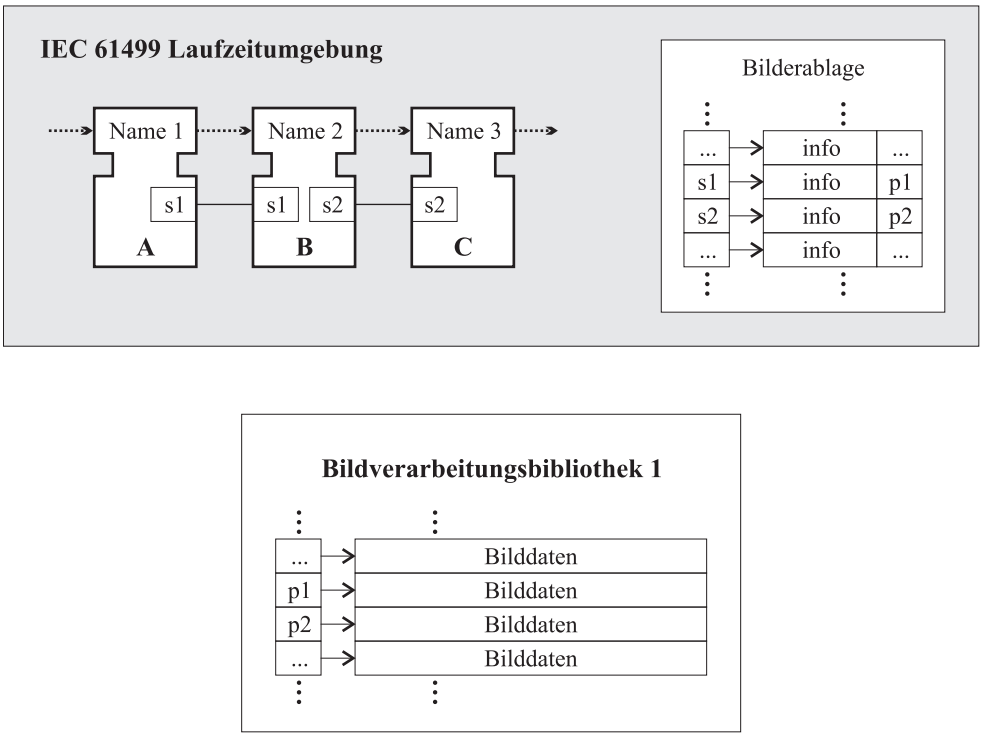


Abbildung 3.4: Funktion der zentralen Bilderablage

Kein Funktionsblock hält sich selbst Bilddaten im eigenen Speicher, sondern

[8] Auch im deutschprachigen Raum hat sich im Bereich der Softwareentwicklung das Wort *Header* anstatt der deutschen Entsprechung Kopfteil durchgesetzt.

[9] gleiches gilt für das englische Wort *Body* anstatt des deutschen Körpers

nur einen Schlüssel (engl. *Key*) in die zentrale Bilderablage (In der Abb. 3.4 mit `s1` und `s2` dargestellt). Bei der Weitergabe eines Bildes von einem Funktionsblock zum nächsten wird somit auch nur der Schüssel weitergereicht, unabhängig von der Art der Datenweitergabe der darunterliegenden Laufzeitumgebung. Die Größe der Schlüssel hängt von der Anzahl der zu erwartenden Bilder ab, aber voraussichtlich ist eine Schlüssellänge von 4 Byte (entspricht mehr als 4 Milliarden Bildern) bei weitem ausreichend.

Die Bilderablage selbst hält auch keine Bilddaten im Speicher da diese von der gewählten externen Bildverarbeitungsbibliothek verwaltet werden. Sie hält sich nur einige Metainformationen zum Bild und einen Zeiger (engl. *Pointer*) auf das Bild im Datenspeicher der externen Bibliothek (In der Abb. 3.4 mit `p1` und `p2` dargestellt).

Gibt nun ein Funktionsblock ein Ausgangsbild[10] weiter, so darf er - z.B. im Falle einer parallelen Ereignisverarbeitung - dieses nicht weiterverändern, solange der nachfolgende Funktionsblock, für welches das Bild nun als Eingangsdatum fungiert, seine Verarbeitung abgeschlossen hat. Dieser zusätzliche Verwaltungsaufwand muss ebenfalls von der Bilderablage übernommen werden.

Durch diese weitere Information der Gültigkeitsdauer bzw. Lebensdauer (Teil der Metainformationen in der Bildablage) des Schlüssel auf ein Bild ist es auch möglich, mehrere nachfolgende Operationen am gleichen Bild durchzuführen und das erste Bild der Verarbeitungskette erst für erneute Änderungen freizugeben, nachdem das letzte berechnet wurde.

Jeder Funktionsblock hat somit die extra Aufgabe, vor dem Beginn einer Ereignisabarbeitung zu prüfen, ob Ausgangsdaten schon verändert werden dürfen bzw. ob der nachfolgende Funktionsblock seine Abarbeitung eines vorangehenden Ereignisses noch nicht beendet hat und somit noch sein Eingangsbild benötigt. Die Handhabung dieser Verwaltungsaufgabe ist IEC 61499 Laufzeitumgebung spezifisch und hängt stark von dessen Strategie der Ereignisabarbeitung ab.

Diese Methode hat folgende Vor- und Nachteile:

\+ geringerer Speicherverbrauch, da jedes Bild nur einmal im Speicher gehalten wird.

\+ schnelle Datenweitergabe, da nur ein Schlüssel (4-Byte) weitergegeben wird.

\+ unabhängig von der Strategie der Datenweitergabe der IEC 61499 Laufzeit-

[10] genauer: einen Schlüssel auf das Ausgangsbild

umgebung, da es keine Rolle spielt, ob der Schlüssel selbst als Wert oder als Referenz weitergegeben wird.

+ Unterstützung einer sequentiellen Weiterverarbeitung der Bilder, unabhängig von der Strategie der Ereignisabarbeitung der IEC 61499 Laufzeitumgebung.

- für verteilte Applikationen muss jede Ressource, welche Bildverarbeitungsblöcke einsetzt, eine Bilderablage besitzen.

- erhöhter Verwaltungs- und Koordinationsaufwand.

- Einschränkung der Datenweitergabe von Bildern, da der Entwickler nun bedenken muss, dass nur Referenzen auf Bilder und nicht die Bilder selbst als Daten weitergereicht werden. Eine solche Technik ist grundsätzlich für IEC 61499 - *noch* nicht - vorgesehen.

Abstraktionsschicht

Die tatsächliche Verarbeitung der Bilder erfolgt nicht in den IEC 61499 Funktionsblöcken selbst, sondern in den darunterliegenden C/C++ Bibliotheken, welche wiederum - jede für sich - ihren eigenen Datentyp für Bilder erfordern. Die Funktionsblöcke sollen nicht direkt auf die externen Bildverarbeitungsbibliotheken zugreifen, sondern sich einer Abstraktionsschicht (Zwischenfunktionen) innerhalb der IEC 61499 Laufzeitumgebung bedienen, um die Funktionen der externen Bibliotheken zu nutzen.

Diese Abstraktionsschicht ist verantwortlich, dass IEC 61499 Bilder aus der Bilderablage so aufbereiten bzw. konvertiert werden, dass die darunterliegenden Bibliotheken diese nutzen können. Die Rolle der Abstraktionsschicht ist in Abb. 3.5 veranschaulicht; vgl. hierzu auch die Abb. 3.1, welche den Zugriff auf die externen Bibliotheken ohne Abstraktionsschicht darstellt.

Diese Methode hat folgende Vorteile:

+ Abstraktion der eingesetzten externen Bibliotheken, d.h. die IEC 61499 Funktionsblöcke haben keinerlei Abhängigkeiten von diesen.

+ Vereinfachung der Arbeiten beim Wechsel einer oder mehrerer externen Bibliotheken, da alle Änderungen nur an einer einzigen Stelle getätigt werden müssen. solche Technik ist grundsätzlich für IEC 61499 - *noch* nicht - vorgesehen.

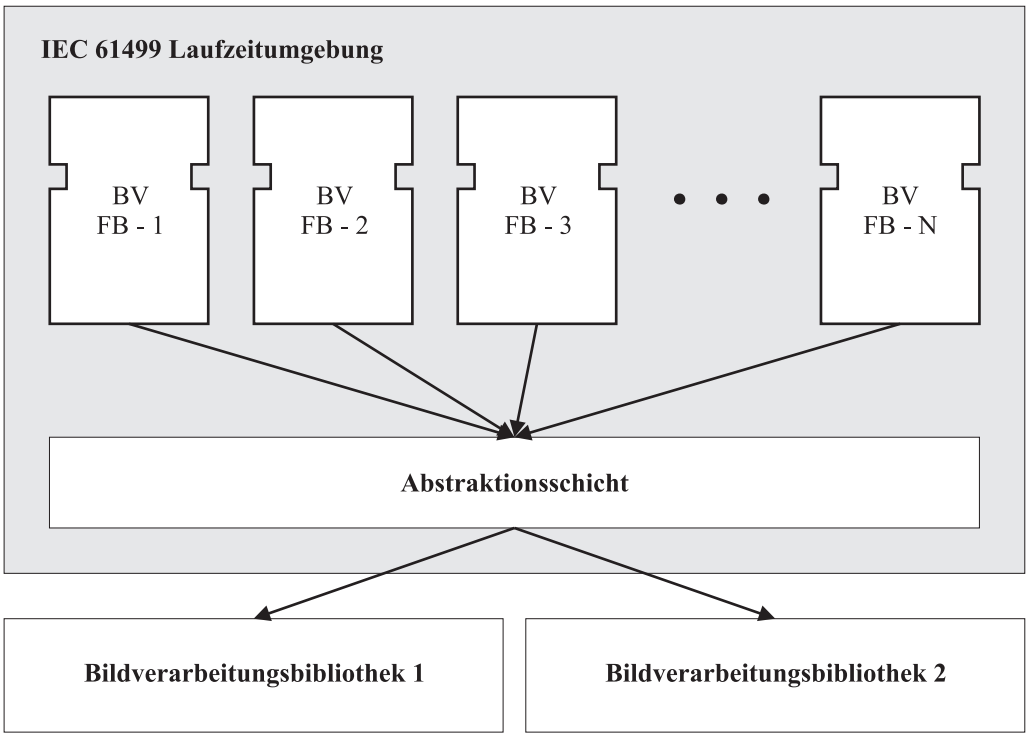


Abbildung 3.5: Zugriff der IEC 61499 Funktionsblöcke auf externe Bibliotheken mittels einer Abstraktionsschicht innerhalb der IEC 61499 Laufzeitumgebung.

3.2.3 IEC 61499 Bildverarbeitungsfunktionsblöcke

In diesem Unterabschnitt werden die benötigten Funktionsblöcke als ein Basissatz an Bildverarbeitungswerkzeugen - in aller Kürze - gruppiert nach ihrer Funktionalität vorgestellt, um einen groben Überblick über die zu entwickelnden Funktionsblöcke zu erhalten.

Funktionsblöcke zur Akquirierung von Bildern.

Sollte das IEC 61499 Device selbst eine Kamera besitzen (z.B. das *Kompaktkamerasystem SBOx-C* oder die USB-Kamera eines PCs), so soll diese - mittels Service Interface Funktionsblöcken - zur Bildakquirierung genutzt werden können.

Als zusätzliche Möglichkeit sollen Bilder mittels des HTTP[11](-Protokoll)s

[11]Das **H**yper**t**ext **T**ransfer **P**rotocol (HTTP) ist im World Wide Web (WWW) das am

akquiriert bzw. geladen werden können. Diese Möglichkeit ist besonders für IP-Kameras von Nutzen.

Funktionsblöcke zum Speichern und Laden von Bildern.

Hierbei sollten zumindest die Speicherformate BMP, JPG und GIF unterstützt werden.

JPG ist zwar ein verlustbehaftetes Format, jedoch besonders für Bilder ohne scharfe Kanten, wie z.B. Fotos, aufgrund des hohen Kompressionsfaktors sehr gut geeignet. GIF wiederum ist ein verlustfreies, aber dafür ein farbreduzierendes Format, welches wiederum schlecht für Fotos, aber gut für Computergrafiken und Zeichnungen geeignet ist. Das BMP ist auch ein verlustfreies, da unkomprimiertes Format, welches daher geringe Rechenleistung beim Laden bzw. Speichern benötigt, aber dafür eines höheren Speicherplatzes bedarf.

Funktionsblöcke zur Farbreduktion und Graustufenbildung.

Hier wird mittels unterschiedlicher Algorithmen versucht, den Informationsgehalt eines Bildes zu reduzieren, indem die Farbtiefe der Bilder reduziert wird. Dies schafft Speicherplatz und ist ein Vorfilter unnötiger Information für die Weiterverarbeitung.

Im einfachsten Fall ist dies eine Graustufenbildung.

Funktionsblöcke zur Binarisierung von Bildern.

Hierbei spielt die Wahl des Schwellwertes (engl. Threshold) für weiße und schwarze Pixel eine große Rolle. Deshalb sollen auch unterschiedliche Algorithmen zur automatischen Wahl des Schwellwertes implementiert werden. Für die Wahl des Schwellwertes wird meistens das Histogramm[12] des Bildes benötigt, weshalb auch Funktionen zur Histogrammbildung und Analyse benötigt werden.

Funktionsblöcke zur Bildsegmentierung und Merkmalsextraktion.

Im Allgemeinen wird ein Bild in Segmente eingeteilt, um die für die Applikation relevanten Bildteile zu filtern. Die Selektion erfolgt auf der Grundlage von diversen berechneten Eigenschaften der einzelnen Bildsegmente. Der häu-

häufigsten benutzte Protokoll.

[12]Das Histogramm ist die Häufigkeitsverteilung der einzelnen Helligkeitstufen eines Bildkanals, z.B. Rot, Grün oder Blau bzw. eines Graustufenbildes. Die Anzahl der unterschiedlichen Helligkeitsstufen hängt wiederum von der Farbtiefe ab. Im Allgemeinen sind dies 1 Byte, d.h. 256 unterschiedliche Helligkeitsstufen.

figste Einsatz besteht z.B. in der Trennung des aufgenommen Objektes vom Hintergrund.

Im einfachsten Fall kann die Segmentierung nach einer Binarisierung erfolgen, wodurch die einzelnen Bildsegmente quasi bereits vorliegen. Die relevanten Bildteile müssen aber nicht immer durch Helligkeitsunterschiede erkennbar sein. Oftmals sind kompliziertere Methoden, wie z.B. der Kantendetektion, notwendig, um eine verwertbare Bildsegmentierung vorzunehmen.

Funktionsblöcke zur Bilderkennung mittels Neuronaler Netze

Neuronale Netze müssen mit Referenzbildern trainiert werden, bevor sie zufriedenstellende Ergebnisse liefern können. Das Training von Neuronalen Netzen kann je nach Größe der Netzwerke und der Vielfalt der zu lernenden Bilder ein sehr zeitintensiver Vorgang sein. Im Allgemeinen kann das Training (bis auf wenige Ausnahmefälle) nicht im laufenden Betrieb erfolgen. Betreffend den IEC 61499 Funktionsblöcken, bedeutet dies, dass das Speichern und Laden von bereits trainierten Neuronale Netzwerken ebenfalls unterstützt werden muss. Konkret muss auch ein neuer Datentyp für Neuronale Netzwerke - nach dem gleichen Schema wie schon für Bilder - eingeführt werden. Weiters werden IEC 61499 Funktionsblöcke zum Trainieren und Ausführen von Neuronalen Netzwerken benötigt.

3.2.4 IEC 61499 Demonstrator

Zur Demonstration der Ergebnisse der vorliegenden Diplomarbeit - wie im Abschnitt 1.2 *„Ziele der Diplomarbeit“* festgelegt - wird im Folgenden eine Bildverarbeitungsapplikation konzipiert. Diese Applikation soll praktische Erfahrungen im Einsatz der Bildverarbeitungsblöcke in auf IEC 61499 basierende Automatisierungssysteme liefern und als Grundlage für das Kapitel 5 *„Diskussion der Ergebnisse“* dienen.

Aus den am OSL (**O**do **S**truger **L**abor des Automatisierungstechnikinstituts der Technischen Universität Wien) zur Verfügung stehenden Mitteln soll der Aufbau - skizziert in Abb. 3.6 - realisiert werden.

Das Herzstück des Aufbaus ist das *Kompaktkamerasystem SBOx-C* - ausführlich im Abschnitt 2.3 beschrieben - auf welchem die IEC 61499 Laufzeitumgebung MARTE installiert wird. Somit ist das *Kompaktkamerasystem SBOx-C* das erste IEC 61499 Device.

Das Kompaktkamerasystem selbst wird auf einem Portalroboter installiert, welcher sich horizontal - mittels zweier X- und einer Y-Achse - beliebig be-

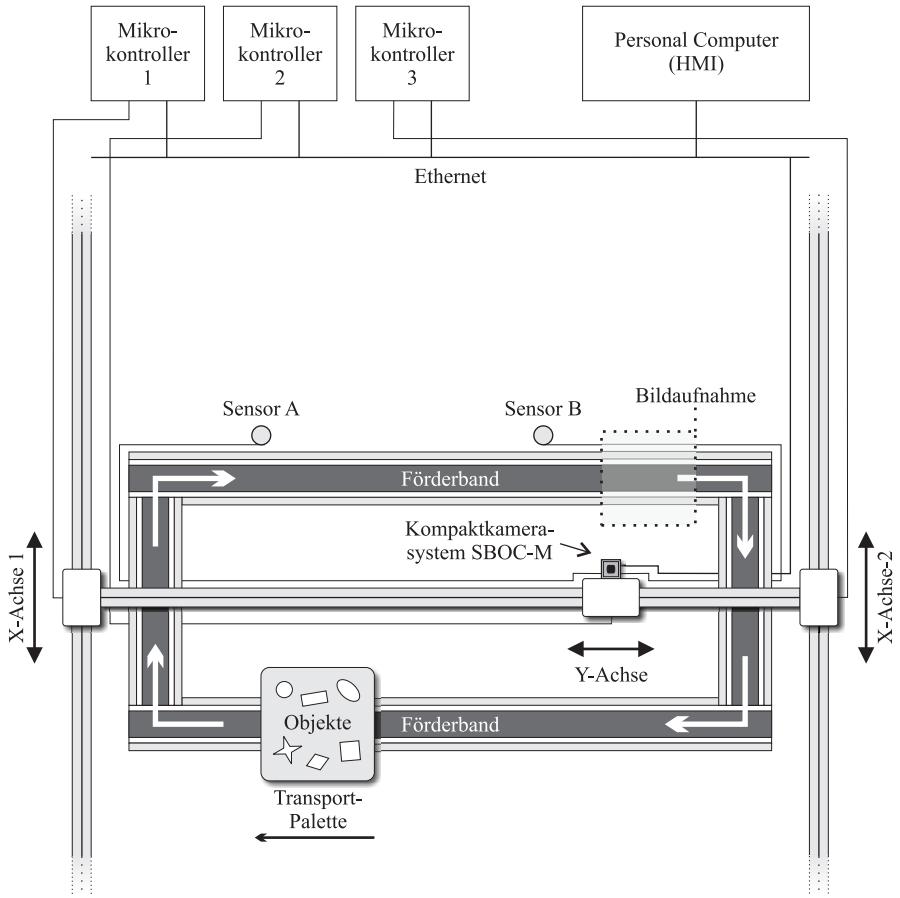


Abbildung 3.6: Konzept für Demoapplikation

wegen kann. Die Steuerung erfolgt für jede der drei Achsen mittels eines eigenen Mikrokontrollers, auf welchen ebenfalls eine IEC 61499 Laufzeitumgebung läuft. Die Synchronization der drei Achsen, um beliebige Kurvenprofile fahren zu können, wurde in der Diplomarbeit von Franz Mehofer [Meh05] realisiert. Die eben genannten Kurvenprofile werden nur durch Angabe von Stützpunkten, d.h. vorgegebene X- und Y-Koordinaten zu einem vorgegeben Zeitpunkt, definiert und selbständig ohne weitere Regelung abgefahren. Im Falle des Portalroboters kann somit auch von einem „intelligenten Aktor“ bzw. *Smart Device* gesprochen werden. Bei der zur Steuerung der Roboterachsen eingesetzten IEC 61499 Laufzeitumgebung handelt es sich um die Vorgängerversion der MARTE, entwickelt am OSL im Rahmen einer Diplomarbeit von Alois Zoitl [Zoi02]. Somit sind die drei zur Robotersteuerung eingesetzten Mikrocontroller drei weitere IEC 61499 Devices.

Das fünfte IEC 61499 Device ist ein PC auf welchem die IEC 61499 Laufzeitumgebung FBRT läuft. Diese soll hauptsächlich für die Visualisierung dienen, jedoch soll sie auch Teile der Bildverarbeitungsaufgaben übernehmen können. Weitere IEC 61499 Devices, z.B. weitere PCs, lassen sich beliebig hinzunehmen, um so die Rechenlast der Bildverarbeitungsapplikation aufzuteilen und gleichzeitig die Verteilungsfunktionalität von IEC 61499 noch besser zu demonstrieren.

Die fünf genannten IEC 61499 Devices stellen ein auf IEC 61499 basierendes, einfaches Automatisierungssystem dar.

Unter dem Portalroboter befindet sich ein Förderband, welches eine Transportpalette im Kreis befördert. Dies ist für eine praktische Anwendung natürlich nicht besonders sinnvoll, jedoch soll dieser Aufbau nur Demonstrationszwecken dienen. Selbstverständlich ließe sich der Portalroboter ohne großen Aufwand über jeden beliebigen anderen Aufbau installieren.

Zwei Induktionssensoren, welche am *Kompaktkamerasystem SBOx-C* angeschlossen sind, detektieren die vorbeifahrende Palette. Der Portalroboter wartet in der Ausgangsposition, welche so definiert ist, dass die Kamera ein gutes Bild von der unterhalb fahrenden Palette machen kann. Sobald der Sensor B die Palette erkannt hat, wird von der Kamera eine Bildaufnahme gemacht. Mittels eines IEC 61499 Funktionsblocknetzwerks sollen die auf der Palette befindlichen Objekte erkannt werden und je nach Sollwertvorgabe über die Visualisierung (das HMI[13]) ein vordefiniertes Objekt markiert werden.

Die Markierung des gewünschten Objektes soll auf der bewegten Palette zwischen dem Sensor A und dem Sensor B mittels eines Laserpointers erfolgen. Hierzu muss der Portalroboter entlang des Förderbandes mit gleicher Geschwindigkeit wie die darunter befindliche Transportpalette bewegt werden. Danach kehrt der Portalroboter in die Ausgangsposition zurück und je nach Steuervorgabe über die Visualisierung wird entweder gleich der nächste Durchlauf gestartet oder erst auf ein weiteres Kommando gewartet.

Für das Mitfahren des Portalroboters mit der Palette ist ein Feedback über die Geschwindigkeit der Palette notwendig. Die Geschwindigkeit wird einfach als konstant vorausgesetzt und über ein Eingabefeld der Applikation bekannt gegeben. Denkbar wäre aber auch ein weiteres IEC 61499 Device, welches *nur* die Geschwindigkeit des Förderbandes mißt und diese Sensorwerte den anderen IEC 61499 Devices zur Verfügung stellt. Auch die Sensoren A und B müssten nicht direkt am *Kompaktkamerasystem SBOx-C* angeschlossen werden, sondern können wiederum über ein eigenes IEC 61499 Device in die Applikation eingebunden werden.

[13] **H**uman **M**achine **I**nterface

Die Visualisierung selbst soll ebenfalls in IEC 61499 programmiert werden unter Zuhilfenahme der Visualisierungsfunktionsblöcke, welche in der IEC 61499 Laufzeitumgebung FBRT bereits implementiert sind.

Somit wird die gesamte Applikation in IEC 61499 entwickelt.

Die Applikation soll zwischen dem *Kompaktkamerasystem SBOx-C* und dem Visualisierungsrechner bzw., falls vorhanden, auch noch weiteren PCs, beliebig verteilt (aufgeteilt) werden können.

Abbildung 3.7: Objekte auf der Transportpalette sind Zifferntasten einer Computertastatur

Die Objekte auf der Transportpalette sind Zifferntasten einer Computertastatur, abgebildet in Abb. 3.7. Dies ergibt sich aus bereits früher getätigten Projekten am OSL, welche ebenfalls diese Tasten genutzt haben.

Die Applikation soll zuerst die Anwesenheit und Position der Transportpalette detektieren und danach ebenfalls die Anwesenheit und Position der Computertasten inklusive der auf diesen abgebildeten Ziffern.

Der Einfachheit halber liegen die Tasten auf der Transportpalette in einem strengen Muster auf starren Positionen und mit fixer Ausrichtung, ebenfalls abgebildet in Abb. 3.8.

Mit den beschriebenen Funktionsblöcken des vorangehenden Abschnitts 3.2.3 stellt es aber keine größere Schwierigkeit dar, beliebig positionierte und orientierte Tasten ebenfalls zu erkennen. Hier gälte es zuerst die Tastenposition und danach ihre Orientierung zu detektieren, um sie je nach Orientierungswinkel, alle in die gleiche Referenzlage zu drehen. Diese extra Aufgabe fällt jedoch in die allgemeine Kategorie der Bildverarbeitungsanwendung und würde nur vom eigentlichen Fokus der Demonstratorapplikation - nämlich einer einfachen Bildverarbeitungsapplikation im Rahmen eines IEC 61499 Automatisierungssystems - ablenken.

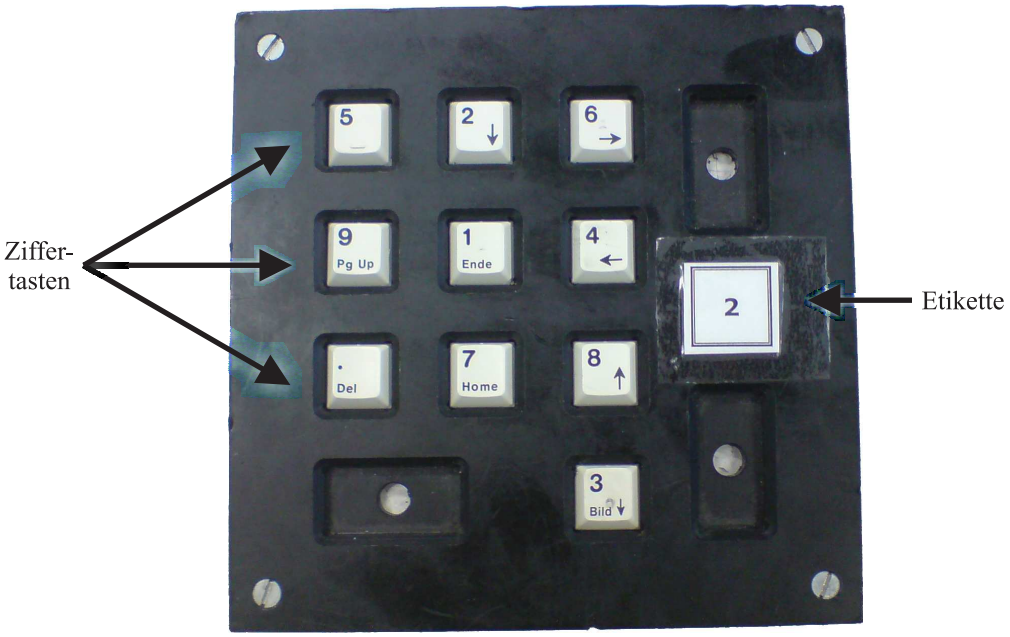


Abbildung 3.8: Positionierung und Ausrichtung der Computertasten auf der Transportpalette. Auf der rechten Seite ist die Etikette (Beschriftung) der Palette - mit der Nummer zwei - zu erkennen, während links davon zehn Zifferntasten abgebildet sind.

3.3 Integrationsaspekte

Die Fragestellung wie das *Kompaktkamerasystem SBOx-C* in ein bereits bestehendes Automatisierungssystem zu integrieren ist, zeigt eine der größten Probleme der heutigen Automatisierungstechnik auf. Gäbe es zurzeit bereits ausreichende Standards, so würde sich diese Fragestellung gar nicht stellen, sondern die Art der Integration läge direkt auf der Hand.

Ziel wäre es, Softwarewerkzeuge zur Verfügung zu haben, mittels welcher die Integration einer jeder beliebigen neuen Hardware in die bereits vorhandene Automatisierungsanlage, stets auf die gleiche Weise geschehen kann. Denn die Heterogenität einer Automatisierungsanlage erfodert gut ausgebildete Fachkräfte in vielen unterschiedlichen Gebieten und erschwert die Wartbarkeit und Instandhaltung um ein Vielfaches.

Der IEC 61499 Standard könnte die Lösung dieser Probleme sein. Egal um welchen Sensor, Aktor oder sonstiger Automatisierungskomponente es sich

handelt, wenn sie den IEC 61499 Standard unterstützt, d.h. eine IEC 61499 Laufzeitumgebung integriert hat, und die notwendigen IEC 61499 Funktionsblöcke mitgeliert werden, so kann die Integration immer auf gleicher Weise erfolgen. Ein schematisches Beispiel, welches die Vermischung von IEC 61499 Funktionsblöcken unterschiedlichster Domänen symbolisiert, ist in Abb. 3.9 dargestellt. Unabhängig vom jeweiligen Unterbereich der Automatisierungstechnik, ob Visualisierung, Logik, Datenbank, Robotersteuerung oder eben auch Bildverarbeitung, die Art der Programmierung ist gleich und es können für das gesamte Gebiet der Automatisierungtechnik homogene Systeme geschaffen werden.

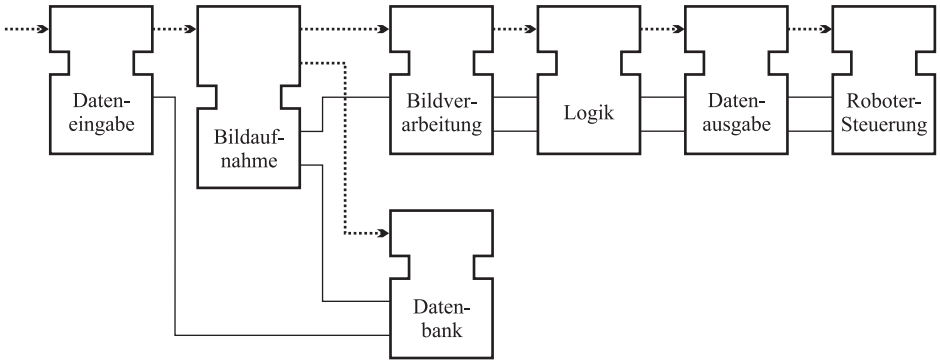


Abbildung 3.9: Ein symbolisches Funktionsblocknetzwerk zur Veranschaulichung der Homogenität eines Automatisierungssystems basierend auf IEC 61499.

Durch die Integration des *Kompaktkamerasystems SBOx-C* in ein IEC 61499 Automatisierungssystem und der Erstellung von Bildverarbeitungsfunktionsblöcken für IEC 61499 ist nun eine weitere Komponente - ein weiterer Puzzlestein - für ein homogenes Automatisierungssystem verfügbar.

Neben den Vorteilen einer vereinfachten Integration neuer Automatisierungskomponenten mittels IEC 61499, soll auch deren bessere Austausch- und Erweiterbarkeit hervorgehoben werden. Funktionsblöcke lassen sich beliebig ohne größerem Aufwand - auch zur Laufzeit - austauschen, z.B. beim Tausch eines Sensors oder Aktors. Ein solcher Tausch ist in Abb. 3.10 angedeutet. Beim Tausch einer Hardwarekomponente muss in manchen Fällen nicht einmal das Funktionsblocknetzwerk verändert werden, wenn nämlich die neue Komponente den gleichnamigen Funktionsblock mit gleicher Schnittstelle implementiert hat.

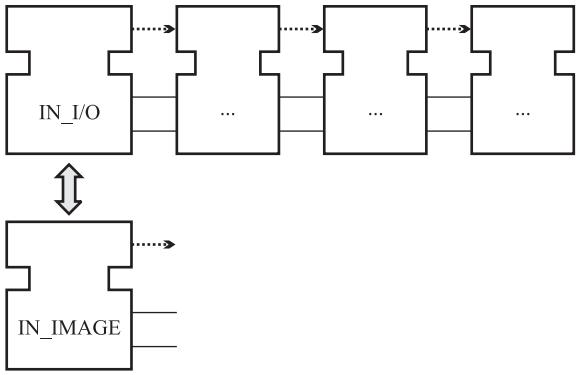


Abbildung 3.10: Wechsel eines Funktionsblocks in einem IEC 61499 Funktionsblocknetzwerk.

Hiezu ein kurzes Beispiel: In einer Automationsanlage wurde die Anwesenheit einer vorbeifahrenden Palette mittels eines Induktionssensors detektiert und damit die Weiterverarbeitung des Objektes auf der Palette eingeleitet. Nun sei diese Art der Detektion nicht ausreichend, da ebenfalls die korrekte Orientierung bzw. Bestückung der Palette für die Weiterverarbeitung vorausgesetzt werden muss, diese aber bisher nicht sicher gestellt werden konnte. Zur Verbesserung der Situation wird eine Kameraeinheit installiert, welche neben der Palette ebenfalls die korrekte Bestückung detektieren kann. Parallel zum laufenden Betrieb wird die Kamera installiert und ein Entwickler programmiert mittels 61499 einen (Composite) Funktionsblock welcher die gestellten Anforderung der korrekten Detektion erfüllt. Nach positiven Tests kann der I/O-Sensor Funktionsblock in der Gesamtapplikation durch den soeben erstellten Funktionsblock getauscht werden. Die IEC 61499 Laufzeitumgebungen FBRT und MARTE unterstützen zur Zeit noch keinen Austausch von Funktionsblöcken „on-the-Fly“[14], sodass in diesem Fall die Anlage kurz angehalten werden müsste, um die geänderte Applikation auf die jeweiligen Mikrokontroller zu kopieren. Ein Funktionsblockaustausch „on-the-Flyïst jedoch angestrebt, wodurch die neue verbesserte Komponente ganz ohne Unterbrechung in die Anlage eingefügt werden kann.

[14] d.h. zur Laufzeit

3.4 Zusammenfassung

Im vorangegangenen Kapitel wurde ein Konzept zur Integration von Bildverarbeitungsfunktionalität in ein Automatisierungssystem, welches auf dem Standard IEC 61499 beruht, vorgestellt. In Kürze zusammengefasst, werden folgende Arbeiten durchzuführen sein:

- Einbindung von externen Bibliotheken, geschrieben in der Programmiersprache C/C++, in die IEC 61499 Laufzeitumgebungen MARTE und FBRT.
- Einführung eines neuen IEC 61499 Datentypes für Bilder.
- Einführung einer zentralen Bilderablage, um die Datenweitergabe von Bildern, welche im Allgemeinen große Datenmengen beanspruchen, durch IEC 61499 zu beschleunigen.
- Einführung einer Abstraktionsschicht zwischen IEC 61499 und den externen Bibliotheken.
- Implementation von IEC 61499 Funktionsblöcken zur Bildverarbeitung.
- Erstellen einer Demonstrationsanwendung, welche die vorhin implementierten Bildverarbeitungsfunktionsblöcke einsetzt.
- Allgemeine Behandlung einiger Aspekte bei der Integration neuer Komponenten in ein Automatisierungssystem.

4 Implementation der Bildverarbeitungfunktionalität in IEC 61499

Das folgende Kapitel behandelt die implementationsspezifischen Themen bei der Umsetzung des Konzeptes aus Kapitel 3 - „Konzept zur Integration von Bildverarbeitung in IEC 61499“.

Zuerst werden die drei externen Bibliotheken, IM, FANN und DISP, und deren Einbindung in die IEC 61499 Laufzeitumgebungen, MARTE und FBRT, betrachtet. Danach folgen Details zur Bilderablage und Abstraktionsschicht und deren Umsetzung in die eben genannten IEC 61499 Laufzeitumgebungen. Vor der abschließenden Beschreibung der Demonstrationsanwendung werden in einem eigenen Unterabschnitt „Ausgewählte Themen“ einige IEC 61499 spezifische Probleme und Schwierigkeiten, welche während der Implementationsarbeiten aufgetreten sind, erläutert.

4.1 Externe Bibliotheken

4.1.1 IM - An Imaging Toolkit

Eine Bildverarbeitungsbibliothek basiert allgemein auf folgende vier Teile: Die Bildakquirierung, -darstellung, -verarbeitung und -speicherung, siehe Abb. 4.1.

Alle vier Komponenten benötigen als Parameter ein digitales Bild - die Bildrepräsentation. Es gibt viele verschiedene Möglichkeiten eine Bildverarbeitungsbibliothek nach obigem Konzept zu implementieren und es gibt hierzu keine allgemein gültige Definition in der Literatur. Es gäbe zwar den Standard ISO/IEC 12087 - PIKS[1] - welcher jedoch ziemlich schwierig und komplex in der Implementation ist und deshalb in den gängigen Bildverarbeitungsbibliotheken kaum Berücksichtigung findet, so auch nicht in der *IM* Bildverarbeitungsbibliothek.

[1] **P**rogrammer's **I**maging **K**ernel **S**ystem

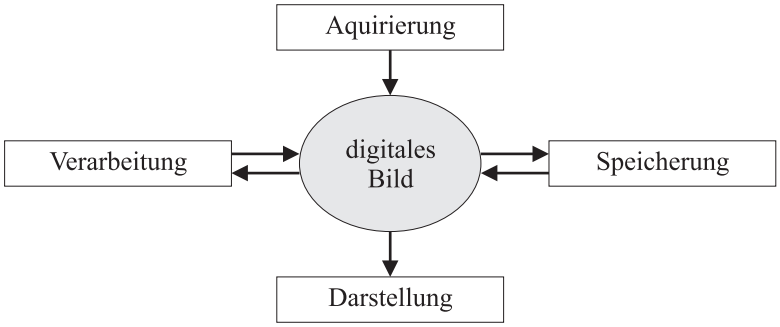


Abbildung 4.1: Komponenten einer Bildverarbeitungsbibliothek

Für die Bildrepräsentation unterstützt die *IM* [wwwIM] Bildverarbeitungsbibliothek folgende Eigenschaften für digitale Bilder:

Breite und Höhe

Bilder sind zwei dimensionale Matrizen von Pixelwerten, definiert durch ihre Breite = Spalten und Höhe = Zeilen. Eine solche Matrix stellt *einen* Farbkanal dar und farbige Bilder können auch mehrere Farbkanäle benötigen. Videos werden als eine Folge von Einzelbildern repräsentiert.

Farbmodus

Es werden die gängigsten Farbmodi unterstützt und darüber hinaus noch einige, welche hier - da nicht relevant - gar nicht erwähnt werden sollen.

RGB Drei Farbkanäle für die einzelnen Farben.

Graustufen Nur ein Farbkanal, für die Helligkeitswerte zwischen schwarz (0) und weiß (1).

Binär Ebenfalls nur ein Farbkanal mit nur boolschen Werten.

Palette Ein Farbkanal mit Farbindexwerten, welche über eine zusätzliche Palette in echte Farbwerte z.B. RGB übersetzt werden können. Diese Technik wird eingesetzt um Speicherplatz zu sparen.

Datentyp

Für die einzelnen Farbkanäle werden folgende Datentypen unterstützt:

Byte 8-Bit.

Short 16-Bit.

Integer 32-Bit.

Float 32-Bit, Fließkomma.

Komplex 2 x 32-Bit, Fließkomma für Real- und Imaginärteil. Dies wird für Fourier- und Laplacetransformierte Bilder benötigt.

Für die Repräsentation eine Binärbildes wäre der Datentyp Bit ausreichend, jedoch können die meisten Rechner nicht ohne zusätzlichen Aufwand Bitoperationen durchführen, weshalb aus Performancegründen auf diese verzichtet wird und stattdessen das Byte eingesetzt wird.

Flags[2]

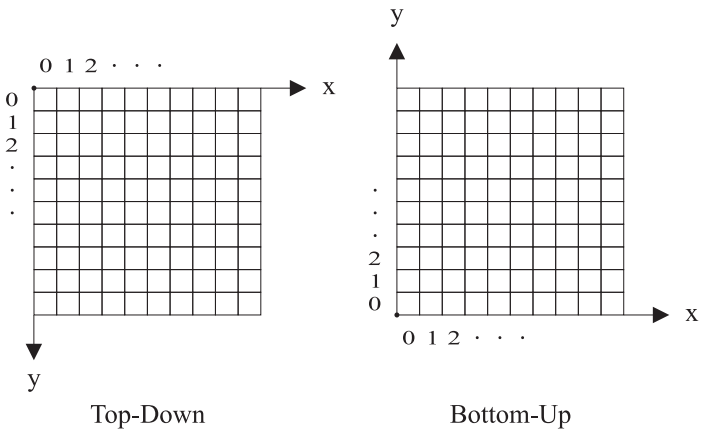
Folgende Zusatzmerkmale können für die Repräsentation der Bilder verwendet werden:

Alpha-Kanal Es wird ein zusätzlicher Alpha-Farbkanal für Transparenz verwendet.

Gepackt- bzw. Ungepackt Die Reihenfolge der Anordnung der Farbkanalpixeldaten im Speicher:

Orientierung Die Orientierung (Adressierung) der Pixelmatrix:

[2] Für das englische Wort: *flag*, welches im Computerfachjargon oft eingesetzt wird, gibt es in der deutschen Sprache keine adäquate Übersetzung, deshalb soll hier ebenfalls das englische Wort benutzt werden. Das PONDS Englisch-Deutsch Wörterbuch [PNS07] übersetzt flag mit: Markierung, Kennzeichen.



Die Darstellungs- und Akquirierungsfunktionen der Bildverarbeitungsbibliothek *IM* werden für den weiteren Einsatz in IEC 61499 nicht benötigt.

Bei den Speicher- und Ladefunktionen wurden seitens der *IM* Bildverarbeitungsbibliothek keine Mühen gescheut, denn diese unterstützt fast alle bekannten Datentypen in ihren unterschiedlichsten Versionen. Auch Videoformate können gelesen bzw. geschrieben werden, wobei entweder nur Einzelbilder oder eine Serie von Bilder geladen bzw. gespeichert wird.

Ebenfalls sind eine Fülle (> 300) von Verarbeitungsfunktionen implementiert, welche im Allgemeinen ein oder mehrere Eingangsbilder und Konfigurationsparameter erwarten und ein Ausgangsbild zurückliefern. Für die meisten Funktionen kann das Ausgangsbild auch gleich das Eingangsbild sein, wodurch Speicherplatz gespart wird.

Diese Verarbeitungsfunktionen bilden einen Basissatz an Funktionalität, welche im Allgemeinen benötigt wird, um einfache, „klassische“ Bildverarbeitungsaufgaben zu lösen. Für kompliziertere Aufgaben wie Objektverfolgung über mehrere Bilder, Gesichtsserkennung usw., muss sich der Anwender der Bibliothek selbst zu helfen wissen und sich aus dem eben genannten Basissatz an Funktionalität komplexere Funktionen entwerfen. So fehlt leider auch jegliche Unterstützung für Neuronale Netzwerke zur Bilderkennung, siehe folgender Unterabschnitt 4.1.2.

Jedoch wird zur Zeit viel am Projekt gearbeitet, denn beinahe quartalsmäßig erscheinen neue Versionen mit neuen Features und so ist es denkbar, dass im Laufe der nächsten Jahre auch komplexere Funktionen hinzugenommen werden.

Die Bibliothek kann kostenlos heruntergeladen werden [wwwIM] und beinhaltet eine eigene Kompilier-Umgebung (*tecmake*), welche die Bibliothek problemlos für alle gängigen Computerplattformen übersetzt. So auch für das

Kompaktkamerasystem SBOx-C unter Zuhilfenahme der Tool-Chain der Firma Festo.

Die Dokumentation der gesamten Bibliothek inklusive der API[3] ist sehr ausführlich dokumentiert und lässt wenige Fragen offen. Ebenfalls die direkte Unterstützung seitens der Projektmitarbeiter der Bibliothek wurde während der Implementationsarbeiten mehrmals in Anspruch genommen.

4.1.2 FANN

Da der - im vorigen Unterabschnitt beschriebenen - Bildverarbeitungsbibliothek *IM* die Unterstützung von Neuronalen Netzwerken fehlt, wird als Ergänzung zu dieser eine zweite externe Bibliothek *FANN - Fast Artificial Neural Network Library* [wwwFN] eingesetzt.

Die *FANN* Bibliothek unterstützt die Handhabung von Backpropagation[4] Neuronalen Netzwerken. Die Methode der Rückpropagierung bzw. der Fehlerrückführung ist mittlerweile ein sehr verbreitetes Verfahren zum Einlernen von künstlichen Neuronalen Netzwerken. Sie gehört zu den überwachten Lernmethoden (im Gegensatz zu selbstorganisierenden Lernmethoden, z.B. für Kohonen Neuronale Netzwerke), d.h. während einer Trainingsphase kennt ein „Lehrer“ die richtigen Zielwerte und hilft dem Neuronalen Netzwerk beim Einlernen dieser. Auf die Theorie der Backpropagation Neuronalen Netzwerke soll hier aber nicht eingegangen, sondern diese lediglich mit einem Verweis auf das Buch von Werner Kinnebrock [Kin94] abgetan werden.

Die gewählte Bibliothek unterstützt beliebig parametrisierbare Backpropagation Neuronale Netzwerke und so auch folgende wichtige Merkmale:

- Anzahl der Ein- und Ausgangsneuronen
- Anzahl der versteckten Schichten (engl. Hidden Layer) und deren Neuronenanzahl
- unterschiedliche Lernfunktionen
- adaptive Lerngeschwindigkeit
- unterschiedliche Datentypen sowohl für die Neuronenausgänge als auch für die Gewichtung der Neuronenverbindungen (als besonders nützlich

[3] **A**pplication **P**rogramming **I**nterface bzw. in Deutsch die Programmierschnittstelle.

[4] Das englische Wort Backpropagation kann ins Deutsche mit *Rückpropagierung* übersetzt werden. Diese hat sich jedoch auch im deutschprachigen Raum nicht durchgesetzt.

sei hier die Unterstützung von Festkomma-Zahlen für Prozessoren ohne Fließkomma-Recheneinheit erwähnt, wodurch sich diese Bibliothek auch gut für Mikrokontroller eignet.)

Die Bibliothek ist ebenfalls kostenlos und kann von der offiziellen Webseite [wwwFN] heruntergeladen werden. Die Übersetzung und Erstellung der Bibliothek gestaltet sich mittels des Standard Linux Tool-Chains *automake* als unproblematisch. So auch für das *Kompaktkamerasystem SBOx-C* unter erneuter Zuhilfenahme der Tool-Chain der Firma Festo.

Auch für diese Bibliothek lassen die ausführliche Dokumentation und die zahlreichen mitgelieferten Beispiele keine Fragen bei der Erstellung einer eigenen Anwendung offen.

4.1.3 Shared Stubs

Der Zugriff auf externe plattformabhängige Bibliotheken aus der Programmiersprache Java heraus erfolgt mit Hilfe des JNI Pakets, welches ab Java Version 1.4 bereits standardmäßig in der Java Laufzeitumgebung enthalten ist.

Im Folgenden wird der Zugriff auf eine einfache C-Funktion mittels des JNI Pakets aus einem Java Programm heraus vorgestellt. Die ursprüngliche C-Funktion - Teil einer C-Bibliothek[5] mit dem Namen *HelloWorld* - lautet z.B. folgendermaßen:

```
#include <stdio.h>

void print()
{
  printf("Hello World!\n");
  return;
}
```

Sie ist nicht besonders anspruchsvoll, sondern gibt nur die Zeichenkette „Hello World!“ auf die Standardkonsole aus. Um nun auf diese C-Funktion zugreifen zu können, muss ein Java Programm mindestens folgende Zeilen implementieren:

[5] Dynamische, d.h. zu Programmbeginn oder überhaupt erst zur Laufzeit ladbare Bibliotheken - im Gegensatz zu statischen Bibliotheken, welche bereits während des Kompiliervorgangs in das fertige Programm eingebunden werden - haben unter dem Betriebssystem Windows die Dateiendung `.DLL` und unter Linux die Dateiendung `.SO`.

```
class HelloWorld
{
 /*
  * Definition der externen C-Funktion, mit
  * dem Befehl: native.
  */
  private native void print();


  public static void main(String[] args)
  {
   /*
    * Aufruf der externen C-Funktion: print()
    */
    new HelloWorld().print();
  }

  static
  {
   /*
    * Einbindung der externen plattformabhängigen Bibliothek
    */
    System.loadLibrary("HelloWorld");
  }
}
```

Mittels des *System.loadLibrary* Befehls im *static* Block wird beim Laden der Java Klasse *HelloWorld* versucht, die externe, plattformabhängige Bibliothek *HelloWorld* zu laden. Diese Funktion ist naturgemäß - je nach darunterliegendem Betriebssystem - unterschiedlich implementiert, siehe Fußnote auf Seite 64. Das *native* Schlüsselwort in der Deklaration der Methode *print* besagt, dass es sich hierbei um eine externe Funktion handelt.

Weiters muss aber die ursprüngliche C-Funktion ebenfalls für den Zugriff aus einem Java Programm vorbereitet werden. Die notwendigen Modifikationen sind im Folgenden Quellkodeauszug dargestellt:

```
#include <jni.h>
#include <stdio.h>

JNIEXPORT void JNICALL
Java_HelloWorld_print(JNIEnv *env, jobject obj)
{
  printf("Hello World!\n");
  return;
}
```

Zuerst muss die Header-Datei *jni.h* eingebunden werden, um JNI-spezifische Deklarationen zu ermöglichen. Danach muss vor dem eigentlichen Funktionsnamen das Wort Java und der Bibliotheksname, jeweils durch einen Unterstrich getrennt, eingefügt werden.

Auch die Liste der Funktionsargumente muss noch angepasst werden, indem zwei weitere Argumente vorangestellt werden und etwaige vorhandene angepasst werden. Die hinzugenommenen Argumente sind JNI Metainformationen für die Schnittstelle zwischen Java und dem C-Programm. Bereits vorhandene Argumente müssen von den ursprünglichen C-Datentypen in JNI-Datenstrukturen umgewandelt werden, z.B. der Datentyp *int* in die JNI-Struktur *jint*, welche weitere Zusatzinformationen enthält.

Die notwendigen Änderungen an den Funktionen auf der C/C++ Seite durch die JNI Schnittstelle, würden für die Einbindung der externen Bibliotheken - *IM* und *FANN* - in die IEC 61499 Laufzeitumgebung FBRT bedingen, dass jede eingesetzte C/C++ Funktion angepasst werden muss. Aus den im Unterabschnitt 3.2.1 dargelegten Gründen ist diese Methode für den hier verfolgten Zweck unbrauchbar. Stattdessen wird zur Einbindung von externen Bibliotheken in die Java basierte IEC 61499 Laufzeitumgebung FBRT die Methode der *Shared Stubs*, vorgestellt in [Lia99], eingesetzt.

Durch die Methode der *Shared Stubs* müssen keine C-seitigen Änderungen am Quellkode vorgenommen werden und so können auch C/C++ Bibliotheken eingebunden werden, zu welchen kein Quellkode verfügbar ist.

Der Java Programmkode der *HelloWorld* Klasse sieht beim Einsatz der *Shared Stubs* Methode folgendermaßen aus:

```
class HelloWorld
{
 /*
  * Definition der externen C-Funktion
  * über die Hilfsklasse CFunction.
  */
  private static CFunction print = new CFunction
                  (
                    "HelloWorld", // Name der Bibliothek
                    "print",      // Name der Funktion
                  );


  public static void main(String[] args)
  {
   /*
    * Aufruf der externen C-Funktion: print()
```

```
    */
    new HelloWorld().print.callVoid();
  }
}
```

Hierbei wird die neue Java Klasse *CFunction* eingesetzt, welche eine externe C-Funktion abstrahiert. Als Instanzierungsparameter benötigt diese Klasse den Namen der externen Bibliothek und den Namen der C/C++ Funktion.

Der Aufruf der C/C++ Funktion erfolgt über Methoden der *CFunction* Klasse und zwar je nach Rückgabewert. Für jeden C/C++ Standarddatentyp hat die *CFunction* Klasse eine eigene Methode, z.B. *callVoid()* für Funktionen ohne Rückgabewert oder *callInt()* fur einen *integer* Rückgabewert. Benötigte C/C++ Funktionsparameter werden in Form eines Java Arrays als Parameter der jeweiligen *CFunction* Methode übergeben.

Der Entwickler muss beim Einsatz der *CFunction* Klasse die Rück- bzw. Übergabeparameter der gewünschten Funktion kennen und erhält keine Warnungen bzw. Fehlermeldungen des Java-Kompiliers im Falle eines Fehlers, sondern im Allgemeinen einen Absturz der Java Laufzeitumgebung und somit auch der geladenen externen C/C++ Bibliothek und des Java Programms.

Auf die Details der *CFunction* Klasse soll hier aus Platzgründen nicht eingegangen werden. Wie bereits im Unterschnitt 3.2.1 dargelegt, bedient sie sich einer weiteren externen C-Bibliothek - mit dem Namen *disp*, welche die reguläre JNI Schnittstelle implementiert, mittels der vorhin beschriebenen JNI Methoden aufgerufen wird und anschließend die C/C++ seitigen Funktionsaufrufe - ohne JNI Umwege - tätigt.

4.2 IEC 61499

Die nachfolgenden Unterabschnitte beschäftigen sich mit den Implementationsarbeiten an den beiden IEC 61499 Laufzeitumgebungen FBRT und MARTE. Die Implementation lässt sich in mehrere logisch zusammenhägende Unterbereiche gliedern:

- Implementation der Bilderablage in die IEC 61499 Laufzeitumgebungen.
- Implementation der Abstraktionsschicht zwischen IEC 61499 Laufzeitumgebung und der darunterliegenden externen Bibliotheken.
- Erstellung der Bildverarbeitungsfunktionsblöcke für IEC 61499 Anwendungen.

- Erstellung von Hilfsfunktionsblöcken für IEC 61499 Anwendungen für noch fehlende Programm Strukturierungsfunktionen, wie z.B. Zählschleifen usw. Diese werden für die anschließende Demonstratoranwendung benötigt.
- Erstellung der Demonstratoranwendung.

4.2.1 Die Bilderablage[6]

Die Bilderablage ist sowohl für die FBRT als auch für die MARTE eine Singleton-Klasse[7] - *ImLab*[8] genannt, mit folgenden Feldern:[9]

instance: Einzige Instanz der Klasse, welche beim ersten Aufruf der *getInstance* Methode instanziert und bis zum Programmende gehalten wird.

image_list: Assoziatives Array[10] zur Speicherung der Bilder. Die tatsächliche Allokation des physikalischen Speichers erfolgt aber nicht in der Abstraktionssklasse, sondern in der darunterliegenden *IM* Bildverarbeitungsbibliothek. Die Abstraktionsklasse verknüpft somit den IEC 61499 spezifischen Bildschlüssel (plus Metainformationen) mit dem Bildverarbeitungsbibliothek spezifischen Schlüssel (+ Metainformationen).

und folgenden Methoden:

[6] In Analogie zur Bilderablage gibt es auch eine Ablage für Neuronale Netzwerke. Da die Funktionalitäten beider Ablagen exakt identisch sind, außer dass in der Bilderablage Bilder und in der Ablage für Neuronale Netzwerke eben diese gespeichert werden, ist hier nur die Bilderablage beschrieben. Alle hier getroffenen Überlegungen gelten aber selbstverständlich auch für die Ablage der Neuronalen Netzwerke.

[7] Singleton (deutsch Einzelstück) ist ein Begriff aus der Softwareentwicklung, welcher eine Klasse beschreibt, die nur ein einziges Mal instanziert werden kann. Die Instanzierung selbst wird meistens über eine eigene statische Methode der Klasse übernommen, sodass deren Konstruktor im Allgemeinen gar nicht öffentlich (*public*) ist, d.h. von außerhalb der Klasse nicht aufgerufen werden kann.

[8] Der Name *ImLab* ist in Anlehnung an die *IM* Bildverarbeitungsbibliothek und der dazugehörenden Bildverarbeitungsapplikation *Image Laboratory* entstanden. Passender wäre vielleicht ein neutralerer Name, wie z.B. *ImageLibrary* gewesen.

[9] Im Zusammenhang mit Klassen und Objekten wird oft von Eigenschaften gesprochen. Hier wird jedoch die Java Terminologie verwendet, in welcher ein Objekt Felder und Methoden besitzt.

[10] Ein Assoziatives Array - auch Hashliste genannt - ist eine Liste von Schlüssel (Key) und Wert (Value) Paaren, wobei jeder Schlüssel nur eindeutig einen Wert zurückliefert.

getInstance: Statische[11] Methode, welche die einzige Instanz der Bilderablage retourniert.

createImage: Hinzufügen eines Bildes in die Bilderablage und retournieren des Schlüssels zu dem Bild. Unter Zuhilfenahme der darunterliegenden Bildverarbeitungsbibliothek wird der erforderliche Speicher - abhängig von den gewünschten Bildeigenschaften - allokiert. Durch den Aufruf von *getImage* mit dem soeben zurückgelieferten Schlüssel, kann das neu erstellte Bild geholt werden.

getImage: Retournieren des Bildes aufgrund eines vorhandenen Schlüssels. Beim „Holen" eines Bildes aus der Bilderablage wird dieses automatisch gesperrt und *muss* anschließend wieder freigegeben werden.

removeImage: Löschen eines Bildes aus der Bilderablage. Das Bild wird nur zum Löschen markiert, ist aber durch nachfolgende *getImage* Befehle nicht mehr zugreifbar. Das tatsächliche physikalische Löschen des Bildes erfolgt über einen *Garbage Collector*[12] erst nachdem alle Referenzen auf das Bild freigegeben wurden. Der Garbage Collector nutzt die Funktionen der Bildverarbeitungsbibliothek, um den von den Bilddaten allokierten Speicher wieder freizugeben.

releaseImage: Freigeben eines Bildes aufgrund eines vorhandenen Schlüssels. Erst nachdem ein Bild wieder freigegeben wurde, ist es für andere Funktionsblöcke zugreifbar.

isLocked: Abfragen, ob ein Bild - identifiziert durch seinen Schlüssel - zur Zeit gerade gesperrt ist.

Die bildverarbeitenden IEC 61499 Funktionsblöcke müssen sich zuerst die *eine* Instanz der Abstraktionsklasse über die statische *getInstance* Methode der Klasse holen. Danach können sie Bilder erstellen bzw. mit Hilfe eines - vom vorangehenden Funktionsblock - erhaltenen Schlüssels das dazugehörige Bild holen und bearbeiten.

Die soeben beschriebene Bilderablage - sowie auch die Ablage für Neuronale Netzwerke - wurde in der Programmierprache C++ in den Quellkode der IEC 61499 Laufzeitumgebung MARTE und in der Programmiersprache Java in den Quellkode der IEC 61499 Laufzeitumgebung FBRT implementiert.

[11] Eine statische Methode einer Klasse kann direkt über den Klassennamen ohne einer Instanz der Klasse aufgerufen werden.

[12] Unter einem *Garbage Collector* versteht man eine automatisch in regelmäßigen Abständen stattfindende Bereinigung des Speichers von nicht mehr benötigten Daten.

4.2.2 Abstraktionsschicht

Die zu implementierenden bildverarbeitenden IEC 61499 Funktionsblöcke sollen sich der Funktionen der externen Bildverarbeitungsbibliotheken bedienen, um ihre Aufgaben zu erfüllen. Damit die Funktionsblöcke aber unabhängig von den externen Bildverarbeitungsbibliotheken (und deren Funktionsumfang) bleiben, wird zwischen diesen eine weitere Abstraktionsschicht eingezogen. Aus der Sicht der Funktionsblöcke gibt es somit nur die Bilderablage (bzw. die Ablage für Neuronale Netzwerke) und die Schnittstelle zur Abstraktionsschicht. Wie die Abstraktionsschicht die an sie gestellten Aufgaben erfüllt, d.h. welcher Bibliotheken sie sich hierzu bedient, ist für die Funktionsblöcke nicht von Interesse.

Durch diese Abstraktionsschicht werden alle bibliotheksspezifischen Abhängigkeiten zentral und gebündelt an einer Stelle gehalten und „verschmieren“ sich nicht über alle implementierten Funktionsblöcke. Mittels dieser Methode ist die Austauschbarkeit der externen Bibliotheken gewährleistet und die damit verbundenen Arbeiten auf ein Minimum reduziert.

Beim Entwurf der Schnittstelle zur Abstraktionsschicht wird ein Top-Down Entwurfsverwahren gewählt, d.h. ausgehend vom Ergebnis, nämlich den IEC 61499 Bildverarbeitungsfunktionsblöcken, werden die Schnittstellen nach „unten“ zur Abstraktionsschicht und noch weiter nach „unten“ zu den externen Bibliotheken definiert. Dadurch wird auch sicher gestellt, dass sich die Schnittstelle der Abstraktionsschicht zu den Funktionsblöcken nicht am Funktionsumfang der externen Bilbiotheken orientiert, sondern nur anhand der *benötigten* Bildverarbeitungsfunktionen entworfen wurde. Erst danach gilt es das Problem zu lösen, welcher Funktionen - der Schnittstelle zu den externen Bibliotheken - sich die Abstraktionsschicht bedient.

Sollte eine Funktion der Schnittstelle zur Abstraktionsschicht seitens der externen Bibliotheken nicht erfüllt werden können, so gilt es seitens der Abstraktionsschicht zu „improvisieren“. D.h. sie muss entweder durch Kombination einfacher Funktionen eine komplexere Funktion erfüllen oder sich einer weiteren externen Bibliothek bedienen. Im Grunde wurde dies schon bei der Hinzunahme der externen Bibliothek für künstliche Neuronale Netzwerke getan, da ansonsten die Abstraktionsschicht ihre *logische* Funktion der Bilderkennung nicht erfüllen hätte können.

Technisch wird die Abstraktionsschicht ebenfalls in die Singleton-Klasse *ImLab*, welche im vorigen Unterabschnitt 4.2.1 eingeführt wurde, implementiert. Dies ist insofern ein guter Ansatz, da auch die Bilderablage Abhängigkeiten von den eingesetzten externen Bibliotheken besitzt.

Die Abstraktionsschicht wurde in der Programmierprache C++ in den Quellkode der IEC 61499 Laufzeitumgebung MARTE und in der Programmiersprache Java in den Quellkode der IEC 61499 Laufzeitumgebung FBRT implementiert.

4.2.3 IEC 61499 Funktionsblöcke

Dieser Unterabschnitt beschreibt die Implementationsarbeiten an den IEC 61499 Funktionsblöcken, welche im Rahmen dieser Diplomarbeit erstellt wurden.

Im Kapitel 2 - „Stand der Technik“ Abschnitt 2.5 - „IEC 61499“ wurde die Freiheit bei der Wahl der Programmiersprache für die Algorithmen von IEC 61499 Funktionsblöcken gelobt. Nun gibt es aber zur Zeit erst eine allgemein - d.h. auch von den beiden IEC 61499 Laufzeitumgebung MARTE und FBRT - unterstützte Programmiersprache, nämlich *Structured Text* (ST), spezifiziert im IEC 61131-3 Standard. Die MARTE Laufzeitumgebung unterstützt auch die Programmiersprache C/C++ für Algorithmen, während die FBRT auch LD (**L**adder **D**iagram) und FBD (**F**unction **B**lock **D**iagram) sowie die Programmiersprache Java unterstützt.

Theoretisch wäre es am besten die IEC 61499 Bildverarbeitungsfunktionsblöcke als Basic Funktionsblöcke mit einem ECC Zustandsgraphen und Algorithmen in *Structured Text* zu implementieren. Dies würde nämlich die Funktionsblöcke unabhängig von der IEC 61499 Laufzeitumgebung machen und sie ließen sich ohne größere Schwierigkeiten auch auf andere Laufzeitumgebungen - neben MARTE und FBRT - portieren.

Praktisch ist dies - zumindest zum augenblicklichen Stand der Technik - *nicht* möglich, da die Programmiersprache *Structured Text* und deren Interpretation durch die jeweiligen IEC 61499 Laufzeitumgebungen ihre Grenzen haben. Hierzu sei angemerkt, dass die Unterstützung von *Structured Text* durch die IEC 61499 Laufzeitumgebungen in zwei Schritten erfolgt. Beim Speichern (bzw. Exportieren) von Funktionsblöcken werden mittels eines passenden Exportfilters[13] alle Algorithmen, welche in *Structured Text* geschrieben sind, in die jeweilige Programmiersprache der darunterliegenden IEC 61499 Laufzeitumgebung übersetzt, d.h. der MARTE Exportfilter interpretiert den *Structured Text* Kode und „schreibt“ daraus C/C++ Kode, während der FBRT Exportfilter den *Structured Text*- in Java-Programmkode umwandelt.

[13] Die beiden Laufzeitumgebungen MARTE und FBRT bringen jeweils ihre Exportfilter als „Add-On“ (Erweiterung) für die IEC 61499 Funktionsblock-Entwicklungsumgebung FBDK mit.

Diese Konvertierung des *Structured Text* Programmkodes unterstützt - zurecht - keine Zugriffe auf außerhalb des Funktionsblocks liegende Funktionen - wie z.B. im Falle der Abstraktionsschicht.

Weiters unterstützt *Structured Text* - wie bereits in Unterabschnitt 2.4.3 ausgeführt - keine objektorientierte Programmierung und keine zusammengesetzten Datentypen mit variabler Länge, wie sie z.B. für die Bilddaten benötigt werden.

Die Algorithmen der Bildverarbeitungsfunktionsblöcke müssen aber auf die Bilderablage und die Abstraktionsschicht zu den externen Bildverarbeitungsfunktionsblöcken zugreifen können. Aus den vorhin beschriebenen Gründen wurde hierfür jedoch mittels *Structured Text* keine Möglichkeit gefunden.

Somit bleibt nichts anderes übrig, als die IEC 61499 Bildverarbeitungsfunktionsblöcke als Service Interface Funktionsblöcke auszuführen. Dies bringt zwar einige Freiheiten bei der Implementation mit sich, hat jedoch den bitteren Beigeschmack, dass die Funktionsblöcke für jede zu unterstützende IEC 61499 Laufzeitumgebung in die jeweilige Programmiersprache der Laufzeitumgebung umgeschrieben werden müssen. Im Falle der FBRT und MARTE wurden zuerst die Bildverarbeitungsfunktionsblöcke für die FBRT in der Programmiersprache Java ausprogrammiert und anschließend für die MARTE in die Programmiersprache C/C++ umgeschrieben.

Bildverarbeitungsfunktionsblöcke

In diesem Unterabschnitt wird zuerst ein abstrakter[14] Funktionsblock betrachtet, welcher den *Größten Gemeinsamen Nenner* der Ereignis- und Daten Ein- und Ausgänge beschreibt, d.h. fast alle implementierten Funktionsblöcke besitzen diese.

Danach werden einige weitere repräsentative Bildverarbeitungsfunktionsblöcke vorgestellt und ausführlicher betrachtet. Eine Liste aller implementierten Funktionsblöcke ist im Anhang A zu finden.

Allgemeiner Bildverarbeitungsfunktionsblock

Abb. 4.2 zeigt einen Funktionsblock, welcher die gemeinsamen, d.h. von allen (natürlich mit wenigen Ausnahmen) implementierten Ein- und Ausgänge, aller

[14] Abstrakt wird hier im Sinne von nicht existierend verwendet und ist an das Schlüsselwort *abstract* der Programmiersprache Java angelehnt, welche eine Klasse beschreibt, die zwingend abgeleitet werden muss und nicht selbst instanziert werden kann. Diese Methode wird in der objektorientierten Programmierung oft benutzt, um Gemeinsamkeiten von verwandten Klassen in eine eigene „Eltern"-Klasse zu vereinen.

bildverarbeitenden Funktionsblöcke abbildet. Diese sind:

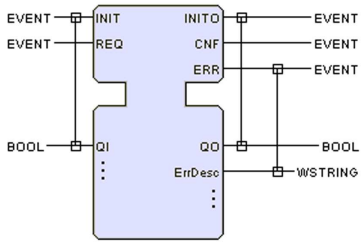


Abbildung 4.2: Abstrakter Funktionsblock mit allen gemeinsamen Ein- und Ausgängen

`INIT`: Initialisierungseingang. Es wird lediglich mittels der Abstraktionsschicht geprüft, ob die externen Bildverarbeitungsbibliotheken erfolgreich geladen werden konnten und ob die benötigten Funktionen verfügbar sind[15]. Nur wenige Funktionsblöcke - z.B. Funktionsblöcke, welche GUI[16] Elemente initialisieren müssen - erfordern eine tatsächliche Initialisierung bzw. Deinitialisierung.

`REQ`: Anforderungseingang zum Starten der eigentlichen Funktion. Beim Eintreten dieses Ereignisses müssen alle Dateneingänge gültige Werte besitzen.

Die meisten Funktionsblöcke sind in Bereitschaft (d.h. Startzustand) und warten auf diese Anforderung, um mit der jeweiligen Bildverarbeitung zu beginnen. Nach Abschluss initiieren sie ein Bestätigungsereignis (`CNF`, siehe weiter unten) und gehen in die Bereitsschaft zurück.

`QI`: Parameter für die Initialisierung bzw. Deinitialisierung. `QI=true` → initialisieren, `QI=false` → deinitialisieren.

`INITO`: Bei Eintreffen eines Initialisierungsereignisses wird dieses zuerst vom Funktionsblock abgearbeitet und anschließend ein neues, ausgehendes Initialisierungsereignis für den nachfolgenden Funktionsblock gestartet.

`CNF`: Bestätigung der Anforderung im Falle, dass keine Fehler aufgetreten sind und das Anforderungsereignis erfolgreich ausgeführt werden konnte.

[15] Dies ist nur eine Absicherung, ob die externen Bibliotheken korrekt kompiliert wurden.
[16] **G**raphical **U**ser **I**nterface

`ERR`: Fehlerindikation im Falle eines Fehlers. Sollte dieses Ereignis ausgelöst werden, so kann über den Datenausgang `ErrDesc` eine textuelle Fehlerbeschreibung ausgelesen werden.

Bei größeren Funktionsblocknetzwerken sind immer wieder auftretende Fehler - besonders während der Entwicklungsphase - leider unvermeidlich. Aufgrund von fehlenden Fehlerbehandlungsroutinen in IEC 61499, ist die „ausführliche“ Protokollierung des aufgetretenen Fehlers die nächstbeste Möglichkeit, die Fehlerquelle einzugrenzen und aufzuspüren.

`QO`: Weiterschleifen des eingegangenen Initialisierungsparameters `QI`.

`ErrDesc`: Fehlerbeschreibung. Besitzt nur im Falle eines Fehlers einen Wert, ansonsten nur eine leere Zeichenkette.

ImageSubImage

Der *ImageSubImage*[17] Funktionsblock, abgebildet in Abb. 4.3, dient dazu, aus einem Bild ein Teilbild[18] herauszukopieren.

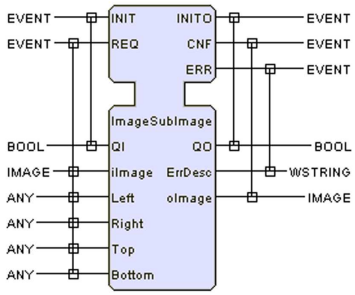


Abbildung 4.3: ImageSubImage Funktionsblock

`iImage`: Das Eingangsbild, aus welchem ein Teilbild herauskopiert werden soll.

`Left, Top`: Die Startkoordinaten links oben in Pixel oder Prozent, z.B. „20“ oder „35 %“.

[17] Alle Bildverarbeitungsfunktionsblöcke beginnen mit dem Schlüsselwort *Image*, gefolgt von einem kurzen Namen, welcher am nächsten die Funktion des Funktionsblocks beschreibt.

[18] Das heraus zu kopierende Teilbild ist meistens ein Bereich von erhöhtem Interesse [engl. Region of Interest (ROI)] und ergibt sich meistens durch die Bildsegmentierung, siehe Abschnitt 2.2.

`Right`, `Bottom`: Die Endkoordinaten rechts unten in Pixel oder Prozent.

`oImage`: Das Ausgangsbild (Teilbild aus dem Eingangsbild).

Die Positionsangaben sind vom Typ `ANY`, d.h. ein beliebiger Wert, welcher sich erst durch die Verschaltung ergibt, und der *ImageSubImage* Funktionsblock kann bei der Abarbeitung des Anforderungsereignisses zwischen Prozentangaben vom Datentyp `WSTRING` und nummerischen Ganzzahlen vom Datentyp `ANY_INT` unterscheiden.

ImageMorphBinOpen

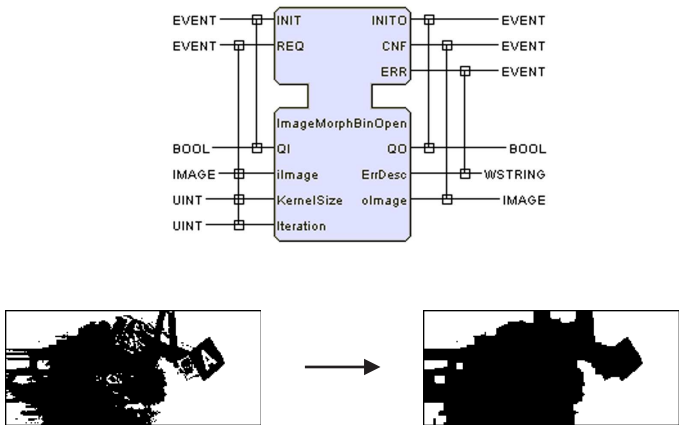


Abbildung 4.4: ImageMorphBinOpen Funktionsblock

Der *ImageMorphBinOpen* Funktionsblock, abgebildet in Abb. 4.4, entspricht der binären morphologischen Operation *Open*, welche sich wiederum aus den binären morphologischen Operationen *Erosion* und *Dilation* zusammensetzt. Die Abbildung zeigt auch ein Beispielbild und die Auswirkung der *Open* Funktion auf dieses. Wie bereits im Abschnitt 2.2 in aller Kürze ausgeführt, bewirkt die *Open* Funktion eine Reduktion der zerfransten Kanten, wie auch in der Abbildung ersichtlich.

`iImage`: Das Eingangsbild, auf welches die morphologisch binäre Operation *Open* angewendet wird.

`KernelSize`: Die Größe des quadratischen *Open* Operators, im Allgemeinen „3“, also 3x3 Pixel.

`Iteration`: Die Anzahl der Durchläufe, im Allgemeinen „1“.

`oImage`: Das Ausgangsbild.

Je größer `KernelSize` und `Iteration`, desto stärker die Reduktion der Verfransung der Kanten im Bild, aber auch desto stärker der Verlust der ursprünglichen Konturen. Die passenden Parameterwerte müssen im Allgemeinen - abhängig von der jeweiligen Anwendung - vor Ort empirisch eingestellt werden.

Ausgesuchte Themen

Bei den Implementationsarbeiten an den IEC 61499 Bildverarbeitungsfunktionsblöcken und an der Anwendung zur Demonstration dieser mussten einige weitere, im Konzept bisher nicht bedachte Hürden gemeistert werden, von denen einige ausgewählte hier in Kürze erläutert werden.

Dieser Unterabschnitt behandelt somit nicht direkt die Arbeiten an der Integration von Bildverarbeitung in eine IEC 61499 Umgebung, jedoch zeigt er wichtige Aspekte bei der IEC 61499 Programmierung auf und wurde deshalb in die Diplomarbeit aufgenommen.

Die während der Implementation gesammelten Erfahrungen können auch für andere Arbeiten betreffend den IEC 61499 Standard von Nutzen sein. Die Demonstrationsanwendung ist ein Funktionsblocknetzwerk mit über 4.000 Funktionsblöcken und beinahe 10.000 Ereignis- und Datenverbindungen und ist somit allein durch seine Größe eine Herausforderung für die Stabilität und Performance einer jeden IEC 61499 Laufzeitumgebung.

Ereignisabarbeitung

Bei der Entwicklung einer IEC 61499 Anwendung muss man die Implementationseigenheiten der IEC 61499 Laufzeitumgebung, auf welcher das IEC 61499 Programm ausgeführt werden soll, beachten und ist somit nicht ganz Device unabhängig, wie eigentlich vom IEC 61499 Standard angestrebt.

Besonders die Reihenfolge der Ereignisabarbeitung der IEC 61499 Laufzeitumgebung muss beachtet werden. Als ein Beispiel sei hier ein einfaches IEC 61499 Funktionsblocknetzwerk mit dem Funktionsblock `E_SPLIT`, skizziert in Abb. 4.5, angeführt. Der Split Funktionsbaustein von Typ `E_SPLIT` teilt ein eintreffendes Ereignis in zwei ausgehende auf. Die Reihenfolge der ausgehenden Ereignisse ist zwar für diesen Funktionsblock sehr wohl festgelegt, jodoch die weiter Ergeignisabarbeitung nicht, wodurch ein solches Netzwerk wie in Abb. 4.5 zu, von der darunterliegenden IEC 61499 Laufzeitumgebung abhängigen, Ergebnissen führt. Der Zustand der ausgehenden Daten von Funktionsblock **B**

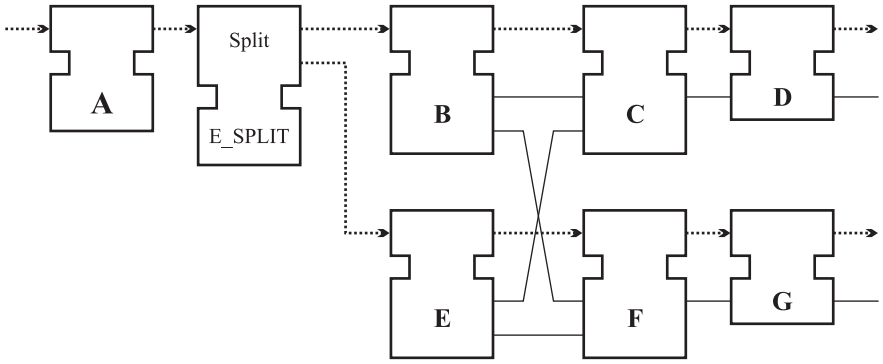


Abbildung 4.5: IEC 61499 Funktionsblocknetzwerk mit Abhängigkeiten von der Ereignisabarbeitungsstrategie der IEC 61499 Laufzeitumgebung.

bzw. Funktionsblock **E**, kann bei der Ausführung von Funktionsblock **C** bzw. Funktionsblock **F** nicht sichergestellt werden. Die IEC 61499 Laufzeitumgebung MARTE hätte folgende Abarbeitungsreihenfolge **B**, **E**, **C**, **F**, **D** und **G**, während die FBRT in der Reihenfolge **B**, **C**, **D**, **E**, **F** und **G** abarbeiten würde.

An dieser Stelle wird nochmals auf [Sun06] verwiesen, wo weitere Fragestellungen betreffend der Ereignisabarbeitungsreihenfolge in einem IEC 61499 Funktionsblocknetzwerk behandelt werden.

Als weiteres Beispiel wird an dieser Stelle auch vor blockierenden Operationen (engl. *blocking operations*) gewarnt. Wenn z.B. ein Service Interface Funktionsblock auf Hardware- bzw. Betriebssystem spezifische Schnittstellen zugreift, so treten dabei immer wieder auch lange bzw. blockierende[19] Wartezeiten auf. Da die darunterliegende Ereignisabarbeitungsstrategie unbekannt ist, und somit auch, welche sonstigen wichtigen Aufgaben der Netzwerkabarbeitung damit noch blockiert werden, so sind wartende und blockierende Konstrukte zu vermeiden bzw. auf eine andere Art zu lösen.

Als ein Beispiel für eine solche blockierende Operation, kann z.B. das Warten auf eine Eingangssignalflanke, d.h. auf den Wechsel eines digitalen Einganges von $0 \rightarrow 1$ oder umgekehrt. Für die IEC 61499 Laufzeitumgebung MARTE ausgeführt am *Kompaktkamerasystem SBOx-C*, wurde ein Service Interface Funktionsblock `IN_SIGNAL` erstellt, welcher beim Ändern eines Signaleinganges

[19] Als blockierend im Gegensatz zu lang ist hier das „unbegrenzte“ Warten auf bestimmte andere Ereignisse gemeint, welche in ungünstigen Konstellation auch zu sogenannten *Dead-Locks* - wo zwei „Beteiligte“ gegenseitig aufeinander warten - führen können.

ein ausgehendes Ereignis (`IND`) auslöst. Für die Implementierung musste das periodische Abfragen des digitalen Eingang mittels den hardwarespezifischen Funktionen der Kompaktkamera in einen eigenen Thread ausgelagert werden, um nicht eine Ereignisabarbeitungskette der MARTE zu blockieren.

Schleifen

Eine der wichtigsten Programmablauf-Kontrollstrukturen in allen Programmiersprachen sind Schleifen, welche einen gewissen Programmteil solange ausführen, bis eine vorgegebene Endbedingung eintritt.

Während der Arbeiten an der Demonstrationsanwendung ist das Fehlen einer solchen Programmstruktur in IEC 61499 als besonders schmerzlich aufgefallen. Dies wurde durch die Einführung zweier Hilfsfunktionsblöcke - eine Schleife mit Abbruchbedingung (Abb. 4.6) und eine Zählschleife (Abb. 4.7) - kompensiert. Bei einem eintreffenden `START` Ereignis beginnt die Schleife

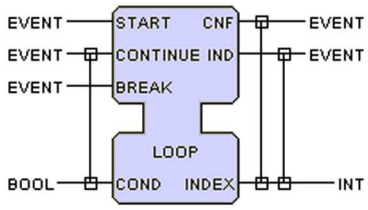


Abbildung 4.6: IEC 61499 Funktionsblock: Schleife mit Abbruchbedingung.

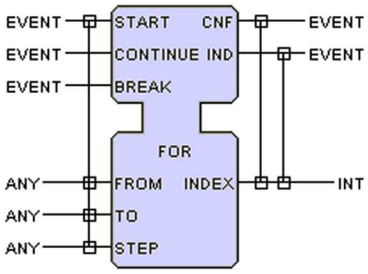


Abbildung 4.7: IEC 61499 Funktionsblock: Zählschleife.

ihre zyklische Abarbeitung, indem pro Schleifendurchgang ein `IND` (engl. indication) Ereignis initiiert wird. Der letzte Funktionsblock der Schleife sendet ein `CONTINUE` Ereignis, woraufhin der Schleifenfunktionsblock die Abbruch-

bedingung prüft und entweder einen weiteren Schleifendurchgang startet oder mittels eines `CNF` (engl. confirmation) Ausgangsereignisses die Schleife beendet.

Die Implementation eines solchen Funktionsblocks hängt abermals sehr stark von der eingesetzten IEC 61499 Laufzeitumgebung ab. Die Ereigniskette (engl. Event-Chain), welche mittels einer `START` Anforderung an ein Schleifen-Funktionsblock gestartet wird, kann - je nach Abbruchbedingung und „Körper" der Schleife - sehr lange sein.

Die IEC 61499 Laufzeitumgebung FBRT verfolgt - wie bereits in Abschnitt 2.5.3 beschrieben - ein Ereignis so lange, bis es versiegt. Im Falle der Schleife würde dies aber heißen, dass die Ereigniskette über mehrere Schleifendurchgänge gehen würde. Weiters würde dies aber bedeuten, dass das Ausgangsereignis eines Funktionsblocks im Schleifenkörper wieder als Eingangsereignis - nämlich im nächsten Schleifendurchlauf - zurückkommt. Da dies, wie bereits in Abschnitt 2.5.3 dargelegt nicht erlaubt ist, muss die FBRT Implementation der Schleife, bei jedem *CONTINUE* Ereignis die Ereigniskette unterbrechen. Beim Ende der Ereigniskette kann in umgekehrter Reihenfolge jeder beteiligte Funktionsblock die Abarbeitung seines Anforderungsereignisses beenden. Das nächste `IND` Ereignis wird dann aufgrund des sequentiellen Programmablaufs vom Schleifenfunktionsblock wieder ausgelöst, sobald alle Ausgangsereignisse aller Funktionsblöcke innerhalb des Schleifenkörpers ebenfalls versiegt sind.

Für die IEC 61499 Laufzeitumgebung MARTE kann nicht *immer* garantiert werden, dass der Schleifenkörper vollständig abgearbeitet wurde, bevor der nächste Schleifendurchgang startet. Sollte innerhalb des Schleifenkörpers ein Ereignis geteilt werden und nicht vor dem Schleifenende wieder zusammengeführt werden - d.h. die beiden „quasi" parallel laufenden Ereignisketten warten an einem bestimmten Punkt wieder aufeinander und werden nur als eine Ereigniskette weitergeführt - so kann ein weiterer Schleifendurchlauf starten, bevor der vorige vollständig beendet wurde.

Auch die Schleifenproblematik soll veranschaulichen, dass der Programmierer das Abarbeitungsmodell der Ziel IEC 61499 Laufzeitumgebung kennen muss, um stabile Anwendungen zu entwickeln.

Fehlerbehandlung

Eine weitere Schwierigkeit während der Implementationsarbeiten war die Behandlung von aufgetretenen Fehlern (Ausnahmeereignissen). Kein noch so *perfekt* entwickeltes Programm kann Fehlerfreiheit garantieren[20] und so müsste jeder Funktionsblock darauf vorbereitet werden, dass der ihm nachfolgende

[20] Häufige nicht softwaretechnisch verschuldete Fehler sind z. B. Netzwerkprobleme, welche aber gerade eine *verteilte* Software meistern können muss.

die an ihn gestellte Anforderung womöglich - aufgrund eines nicht planbaren Fehlers - nicht erfüllen kann.

Der IEC 61499 Standard sieht keine Möglichkeit vor einen solchen Fehler rückzumelden, denn eigentlich sollte der Funktionsblock, welcher die Anforderung gestellt hat und kein anderer, über den Fehler informiert werden. Der IEC 61499 Standard sieht für einen Fehlerfall vor, dass trotzdem ein Bestätigungsereignis (`CNF`) gesendet wird, aber mit dem Ausgangsdatum `QI` $= 0$. Der nachfolgende Funktionsblock müsste bei einem Anforderungseingang sein Eingangsdatum `QI` prüfen und im Falle, dass $QI = 0$ ebenfalls, ohne „echte“ Abarbeitung der Anforderung, ein Bestätigungsereignis mit $QO = 0$ senden.

Durch den fehlenden Feedback an den Auslöser der Anforderung, muss jeder Funktionsblock davon ausgehen, dass seine gestellte Anforderung erfüllt wurde und setzt mit seiner Abarbeitung fort.

Graphische Programmierung

Für Softwareentwickler, welche textuelle Programmiersprachen, wie z.B. Java, C/C++ usw. gewöhnt sind, ist der Umstieg auf die graphische Programmierung nicht immer einfach. Komplexere Programme mit vielen Funktionsblöcken und Verbindungen (wie z.B. die Demonstrationsanwendung) werden sehr schnell unübersichtlich. Der Einsatz von Verzweigungen, Schleifen und Fehlerbehandlungsroutinen tragen ebenfalls nicht gerade zur besseren Lesbarkeit bei.

4.2.4 IEC 61499 Demonstrator

Die im Abschnitt 3.2.4 konzipierte Anwendung zur Demonstration der Bildverarbeitungsfunktionalität von IEC 61499 wird in zwei getrennte Anwendungen gespalten:

- Die *Teaching*-Anwendung
- Die *Run*-Anwendung

Teaching-Anwendung

Die *Teaching*-Anwendung dient dem Einlernen der Ziffern auf den Zifferntasten, d.h. dem Trainieren der künstlichen Neuronalen Netzwerke mittels Referenzdaten. Die *Teaching*-Anwendung ist für die Ausführung auf einem PC auf der FBRT IEC 61499 Laufzeitumgebung konzipiert. Die Rechenbelastung ist während des eigentlichen Trainingsvorgangs der Neuronalen Netzwerke sehr

hoch und deshalb kommt für diese Anwendung nur ein rechenstarker Computer in Frage. Die Akquirierung der Bilder erfolgt trotzdem über das *Kompaktkamerasystem SBOx-C*, indem dieses als Web-Server[21] fungiert und die aufgenommenen Bilder über das HTTP(-Protokoll) von der *Teaching*-Anwendung geladen werden.

Das Programm ist zu 100 % in IEC 61499 geschrieben und bedient sich der Visualisierungselemente der FBRT IEC 61499 Laufzeitumgebung. Der Programmablauf ist in drei Teile geteilt:

1. Sammeln von Referenzdaten
   a) Eingabe der Sollwerte durch den Benutzer
   b) Bildaufnahme
2. Trainieren der Neuronalen Netzwerke
3. Speichern der Neuronalen Netzwerke

Im ersten Teil wird das *Kompaktkamerasystem SBOx-C* in die Startposition (Bildaufnahmeposition) gebracht und tätigt eine Aufnahme von jeder unterhalb vorbeifahrenden Palette. Vorher wurden dem System die korrekten Ziffern der Tasten auf den einzelnen Ziffernpositionen eingegeben. Dieser Vorgang wird öfters wiederholt, bis jede Ziffer auf jeder Position mindestens einige Male aufgenommen wurde.

Danach beginnt der zweite Teil: Das Einlernen der Ziffern. Für jede der zehn Tastenpositionen wird ein neues Neuronales Netzwerk mit 440 Eingangsneuronen (= 440 Pixel für ein 20 x 22 großes Bild) und 10 Ausgangsneuronen (= 10 unterschiedliche Ziffern) erstellt und danach das Training begonnen. Der Einsatz nur eines Neuronalen Netzwerks für alle Positionen hat sich in der Praxis als nicht ausreichend gezeigt.

Das Problem ist, dass eine Ziffer - representiert durch ein 20 x 22 Pixel großes Bild - auf den einzelnen Tastenpositionen sehr unterschiedlich aussieht und das Neuronale Netzwerk es in der Praxis nicht schafft, mehrere Aufnahmen der gleichen Ziffer, aber auf unterschiedlichen Tastenpositionen, als *gleich* zu kategorisieren. Hierzu sind einige Aufnahmen der gleichen Ziffer acht, aber auf unterschiedlichen Positionen in Abb. 4.8 dargestellt. Diese starken Unter-

[21] Ein Web-Server ist die serverseitige Implementation des HTTP(-Protokolls). (Eine weitverbreitete clientseitige Implementation des HTTP(-Protokolls) ist ein Web-Browser, z. B. Internet Explorer.) Der Server wartet auf eingehende HTTP Anforderungen (engl. *requests*) und sendet die geforderten Daten (im Allgemeinen HTML-Dokumente, Bilder oder sonstige Mediendaten) zurück.

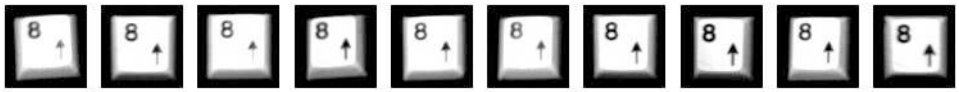


Abbildung 4.8: Aufnahme derselben Ziffer auf unterschiedlichen Tastenpositionen.

schiede werden durch die perspektivische Verzerrung der Kamera und durch die ungleichmäßige Beleuchtung der Palette hervorgerufen. Für die Demonstrationsanwendung wurde auf den Einsatz einer extra Beleuchtung - mittels welcher die Schwankungen des Umgebungslichts kompensiert werden könnten - verzichtet. Durch den Einsatz von zehn Neuronalen Netzwerken - was keinen Mehraufwand in der Rechenzeit, sondern nur etwas wenig höheren Speicherbedarf bedingt - kann aber dieses Problem gelöst werden.

Auch die gleiche Ziffer mehrfach an einer Tastenposition aufgenommen unterscheidet sich von Aufnahme zu Aufnahme, jedoch sind dies die Variationen (Störungen) der Ziffernaufnahmen, welche eben das Neuronale Netzwerke trotzdem als gleich kategorisieren kann - natürlich nur nach einem erfolgreichen Training mit ausreichenden Referenzaufnahmen. Diese kleinen Unterschiede in der Aufnahme werden durch leichte Rotationen der Transportpalette, durch sich leicht ändernde Belichtungsverhältnisse und durch das Rauschen des Kamerasensors bedingt.

Im dritten Teil der *Teaching*-Anwendung werden die trainierten Neuronalen Netzwerke auf das Dateisystem gespeichert.

Die Benutzerschnittstelle des Programms besteht aus mehreren Dialogfenstern, welche in Anhang B abgebildet sind und im Folgenden nur kurz beschrieben werden sollen.

Der *Command* Dialog, dargestellt in Abb. B.1, ist der Hauptdialog zur Steuerung der *Teaching*-Anwendung. Einerseits dient er zum Starten bzw. Fortsetzen und zum Anhalten der Aufnahme von Referenzbildern. Bevor die Aufnahme von Referenzbildern begonnen wird, müssen aber auch noch die auf der Transportpalette platzierten Tasten der Anwendung mitgeteilt werden. Zum anderen kann mittels des Hauptdialogs das Trainieren der Neuronalen Netzwerke gestartet bzw. wieder pausiert werden. Die Eingabe der zurzeit auf der Palette befindlichen Tasten (genauer der Ziffern auf den Tasten) geschieht über den Dialog *Reference* abgebildet in Abb. B.2.

Mittels eines Statusfensters (*Status* dargestellt in Abb. B.3) werden einige

Information über das gerade (zuletzt) aufgenomme Bild angezeigt. Hieraus sind die wichtigsten Merkmale der Aufnahme und die ersten Bildverarbeitungsergebnisse ersichtlich, z.B. die Anzahl der auf der Transportpalette gefundenen Tasten (*Valid Regions*). Diese zu den Referenzbildern hinzugenommenen Bilder werden auch in einem weiteren Dialog *Last Samples*, abgebildet in Abb. B.4, angezeigt.

Je mehr Referenzbilder aufgenommen werden, umso besser können die Neuronalen Netzwerke auf das, was sie während der eigentlichen Ausführung (*Run*-Anwendung) zu erkennen haben, vorbereitet werden und umso bessere Ergebnisse lassen sich erzielen. Leider dauert auch das Trainieren der Neuronalen Netzwerke umso länger. Die Anzahl der bereits aufgenommenen Referenzbilder zeigt der *Samples*-Dialog, dargestellt in Abb. B.5, an, d.h. die Anzahl der bis jetzt aufgenommen Referenzziffern für jede der zehn Tastenpositionen und jede der zehn Ziffern somit 100 Ausgabefelder. Wenn in jedes dieser Feld mind. fünf, besser aber 10 - 15 Referenzen gelistet sind, kann das Trainieren der Neuronalen Netzwerke begonnen werden.

Nach dem Beginn des Trainingsvorgangs können die Trainingsfortschritte im *Status*-Dialog, dargestellt in Abb. B.6 beobachtet werden. Das Trainieren der Netzwerke dauert selbst auf einem starken Rechner seine Zeit (z.B. auf einem Pentium IV mit 3 GHz ungefähr 10 Minuten) und kann beendet werden, sobald der MSE[22]-Wert $< 0,00001$ geworden ist. Für manche der zehn Netzwerke ist dieser Wert früher erreicht, für einige dauert dies länger.

Der letzte Dialog, *Save*, dargestellt in Abb. B.7, dient zum abschließenden Speichern der Neuronalen Netzwerke nachdem das Training abgeschlossen wurde. Hierfür wird pro Netzwerk eine Datei mit der Nummer der jeweiligen Tastenposition (1 - 10) geschrieben, wobei der Anfang und das Ende des Dateinamens durch den Benutzer eingegeben werden kann. So können auch mehrere Sätze von Neuronalen Netzwerken für stark unterschiedliche Lichtverhältnisse oder für andere Tastentypen gespeichert werden, denn es ist nicht zwingend notwendig, Ziffern auf den Tasten gedruckt zu haben. Denkbar wären auch beliebige sonstige Symbole.

Run-Anwendung

Die *Run*-Anwendung ist die eingentliche Demonstrationsanwendung und ist die Umsetzung der im Abschnitt 3.2.4 konzipierten Anwendung. Für das (mehr oder weniger) korrekte Erkennen der Ziffern auf den Computertasten bedient

[22] **M**ean **S**quare **E**rror

sie sich der vorher bereits trainierten Neuronalen Netzwerke.

Ein Programmablaufplan (PAP) nach DIN 66001 der *Run*-Anwendung für einen Programmdurchlauf ist in Abb. B.8 skizziert. Der Programmablaufplan musste aus Platzgründen eher kompakt gehalten werden, weshalb einige der Aktionen darin nicht ganz selbsterklärend sind. Diese werden in folgender Auflistung etwas genauer beschrieben:

Region ist Palettenetikette? Die Überprüfung, ob die Transportpalette in der Bildaufnahme gefunden werden konnte, erfolgt anhand einer Etikette, welche auf jeder Palette geklebt ist und nach dem Binarisieren als großes weißes Quadrat aufscheint. Die berechneten Merkmale, nämlich die Fläche, der Umfang und das Zentrum der gesuchten Region - der Palettenetikette - müssen innerhalb bestimmter Grenzwerte liegen. Ist dies erfüllt, so wird davon ausgegangen, dass die Palettenetikette und somit auch die genaue Position der Palette gefunden wurde.

Region ist Taste? Die Überprüfung, ob eine Region eine Taste ist, erfolgt nach selbigem Verfahren. Die Merkmale der einzelnen Regionen bleiben gespeichert und müssen nicht neu berechnet werden.

Position bestimmen Hier wird die Postion der gefundenen Taste auf der Palette (1-10) bestimmt, damit anschließend das korrekte Neuronale Netzwerk zur Bilderkennung herangezogen werden kann.

Erkennungsrate $>$ 50 %? Das Neuronale Netzwerk liefert an jedem Ausgangsneuron einen Wert zwischen 0 und 1. Damit eine Kategorisierung akzeptiert werden kann, muss sie wenigsten größer als ein Schwellenwert - in diesem Fall 0.5 - sein. Das Ausgangsneuron mit dem höchsten Ausgang ist die erkannte Ziffer.

Taste markieren? Bei dieser Abfrage wird überprüft, ob die erkannte Ziffer jener durch den Bediener ausgesuchten Ziffer entspricht, welche markiert werden soll.

Roboter bewegen Die Bewegung des Roboters zur Markierung einer Taste erfolgt in drei Schritten. Zuerst wird der Roboter in die Ausgangsposition gebracht, ab welcher die Taste markiert werden soll. Die Ausgangspostion ist bereits die genaue Position über der zu markierenden Taste. Sobald der Detektionssensor die vorbeifahrende Palette erkennt, beginnt auch der Portalroboter seine Bewegung synchron mit der unterhalb fahrenden Palette und ein eingeschalteter Laserpointer markiert die gewünschte

Taste. Nach einer definierten Zeit verlässt der Roboter seine synchrone Bahn über dem Förderband und geht in die Startposition zurück.

Für die Steuerbefehle des Roboters wurden IEC 61499 Motion Control Funktionsblöcke (konform zu den PLCOpen für Motion Control Vorgaben [PLC05]), welche im Rahmen der Diplomarbeit von Franz Mehofer [Meh05] entwickelt wurden, eingesetzt. Diese wurden zwar, wie bereits im Unterabschnitt 3.2.4 angeführt, für eine Vorgängerversion der IEC 61499 Laufzeitumgebung MARTE entwickelt, jedoch, da es sich um Basic Funktionsblöcke handelt, konnten sie problemlos in die IEC 61499 Laufzeitumgebungen MARTE und FBRT portiert werden. Es waren keinerlei Anpassungen bzw. Erweiterungen notwendig.

Die *Run*-Anwendung existiert in zwei Versionen, welche sich nur durch die Aufteilung auf die IEC 61499 Devices unterscheiden:

1. Ausfühung am *Kompaktkamerasystem SBOx-C*
   - Die Visualisierungselemente laufen am PC in der IEC 61499 Laufzeitumgebung FBRT.
   - Alles andere wird am *Kompaktkamerasystem SBOx-C* in der IEC 61499 Laufzeitumgebung MARTE ausgeführt.
2. Ausfühung am PC
   - Die Bildakquirierung und Detektionssensoren laufen am *Kompaktkamerasystem SBOx-C* in der IEC 61499 Laufzeitumgebung MARTE.
   - Alles andere wird am PC in der IEC 61499 Laufzeitumgebung FBRT ausgeführt.

Die Benutzerschnittstelle umfasst zwei Dialoge, einen Eingabe-Dialog (*Kommando*-Dialog) - abgebildet in Abb. 4.9 - und einen Ausgabe-Dialog (*Output*-Dialog) - abgebildet in Abb. 4.10.

Bevor der erste Programmdurchlauf ausgeführt werden kann, muss die zu markierende Taste (*Key*) ausgewählt und die Geschwindigkeit der Förderbänder (*Conveyor Velocity*) eingegeben werden. Danach kann entweder ein Einzeldurchlauf (*Single Step*) gestartet oder die Anlage in den kontinuierlichen Betrieb (*Continuous Run*) - d.h. nach einem Durchlauf wird sogleich der nächste gestartet - geschaltet werden. Nach dem Befehl, den kontinuierlichen Betrieb wieder zu beenden (*Stop Run*), wird zwar der bereits gestartete Durchlauf noch beendet, aber kein weiterer mehr gestartet.

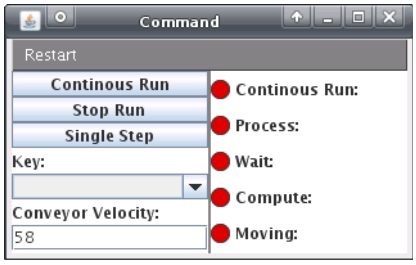


Abbildung 4.9: Dialog zur Steuerung der *Run*-Anwendung.

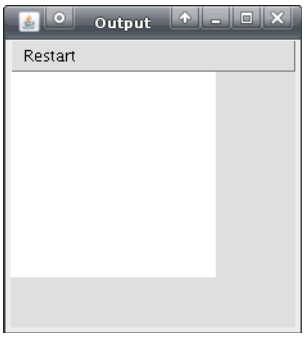


Abbildung 4.10: Ausgabefenster der *Run*-Anwendung.

Das Ausgabefenster zeigt die gefundenen Zifferntasten, wobei falls die zu markierende Ziffer ebenfalls erkannt wurde, diese in rot hinterlegt dargestellt wird.

4.3 Zusammenfassung

Im vorangegangenen Kapitel wurden die implementationstechnischen Details und die während der Implementation aufgetretenen Probleme behandelt.

Zuerst erfolgte eine kurze Beschreibung der eingesetzten externen Bibliotheken:

- IM - An Imaging Toolkit
- FANN - Fast Artificial Neuronal Network Library
- DISP - Shared Stubs

und danach die IEC 61499 spezifischen Punkte:

- Die Bilderablage
- Die Abstraktionsschicht
- Ausgewählte Themen zu IEC 61499
- Beschreibung der Demostrationsanwendung

5 Diskussion der Ergebnisse

Die generelle technische Realisierbarkeit der Aufgabenstellung, nämlich der Integration von Bildverarbeitung in ein Automatisierungssystem basierend auf IEC 61499, stand zu keiner Zeit in Frage, denn dass es „irgendwie“ gehen muss, war einleuchtend. Die Art und Weise, wie dieses aber bewerkstelligt wurde und welche Vorteile sich gegenüber anderen Alternativen ergeben, sollen im folgenden diskutiert werten.

Die erste grundlegende Entscheidung, siehe zu Beginn von Kapitel 3, war der Einsatz von externen Bibliotheken anstatt einer Eigenentwicklung der Bildverarbeitungsfunktionalität. Dies mag zwar selbstverständlich klingen, da ja die Vorteile hiervon auf der Hand liegen, jedoch gibt es *keine* Bildverarbeitungsbibliotheken für die standardisierten Programmiersprachen der Automatisierungssysteme.

Die getroffene Entscheidung war somit der Einsatz von „low-level“ externen Bibliotheken - die in großer Zahl verfügbar sind -, welche für die darüberliegenden Laufzeitumgebungen - der eben erwähnten „high-level“ Programmiersprachen - zur Verfügung gestellt werden sollen. Dies erspart einerseits die Entwicklungsarbeiten der Implementation von Bildverarbeitungsfunktionen, aber erfordert ein tiefergehendes Verständnis der Standards für Automatisierungssysteme und der eingesetzten Laufzeitumgebungen.

Trotzdem wurde diese Methode gewählt, da sie folgende Vorteile mit sich bringt:

- Durch einen einmaligen Entwicklungsaufwand wird das Know-How und das Framework zum Zugriff auf *beliebige* externe Bibliotheken (welche in der Programmiersprache C/C++ geschrieben wurden) geschaffen.
- Beim Einsatz externer Bibliotheken mit weitem Funktionsumfang ist der Arbeitsaufwand wesentlich geringer und die Arbeitszeit amortisiert sich sehr schnell.
- Die Arbeiten an Bildverarbeitungsfunktionen führen weit weg vom eigentlichen Forschungsgebiet der Automatisierungstechnik.

- Die Arbeiten führen stattdessen tiefer in das „innere“ der Standards der Automatisierungstechnik.
- Geschwindigkeitsvorteile aufgrund des Einsatzes von bereits „leistungsoptimierten“ Bibliotheken und der Programmiersprache C/C++ gegenüber einer „high-level“ Programmiersprache.
- Keine Einschränkung der Funktionalität durch die unterstützten - zeitweise sehr einfachen - Programmiersprachen der Standards.

Als Nachteil soll nicht unerwähnt bleiben, dass nun bei der Portierung auf eine neue Zielplattform nicht nur die Laufzeitumgebung, sondern auch die eingesetzten Bibliotheken für das Zielsystem übersetzt und bei Bedarf angepasst werden müssen. Wären die Bildverarbeitungsfunktionen direkt in einer vom Standard unterstützen Programmiersprache kodiert, so würde dieser Schritt entfallen.

5.1 Performancevergleich

Eine wichtige Frage, ist jene nach den Leistungseinbußen durch den Einsatz von IEC 61499 gegenüber einem C/C++ Programm. Dass hier mit Verzögerungen aufgrund des Overheads[1] des IEC 61499 Standards zu rechnen ist, bedarf keiner Erklärung, schließlich bietet der Standard sehr viele Annehmlichkeiten, welche mit Performanceverlusten erkauft werden müssen. Wie hoch diese sind, ist aber von großem Interesse.

Parellel zur Demonstrationsanwendung als IEC 61499 Anwendung, wurde selbige auch als reines C/C++ Programm implementiert. Für beide Programme wurden mehrere Programmdurchläufe mit jeweils unterschiedlicher Tastenanzahl auf der Transportpalette durchgeführt und Zeitmessungen der Rechenzeit durchgeführt. Die Anwendung nutzt einen digitalen Ausgang des *Kompaktkamerasystems SBOx-C*, um den Beginn und das Ende seiner Berechnungen zu signalisieren, welche mittels eines Oszilloskops dargestellt wurden und somit die Rechenzeit gemessen werden konnte.

Die benötigten Rechenzeiten wurden in Abb. 5.1 in einem Diagramm gegenübergestellt. Beide Programme benötigen eine gewisse Zeit (ca. 3-4 Sekunden), unabhängig von der Tastenanzahl, um das Bild zu Akquirieren, die ersten

[1] Overhead ist ein in der Informatik oftmals verwendeter Anglizismus, den man mit „Mehraufwand“ übersetzen kann.

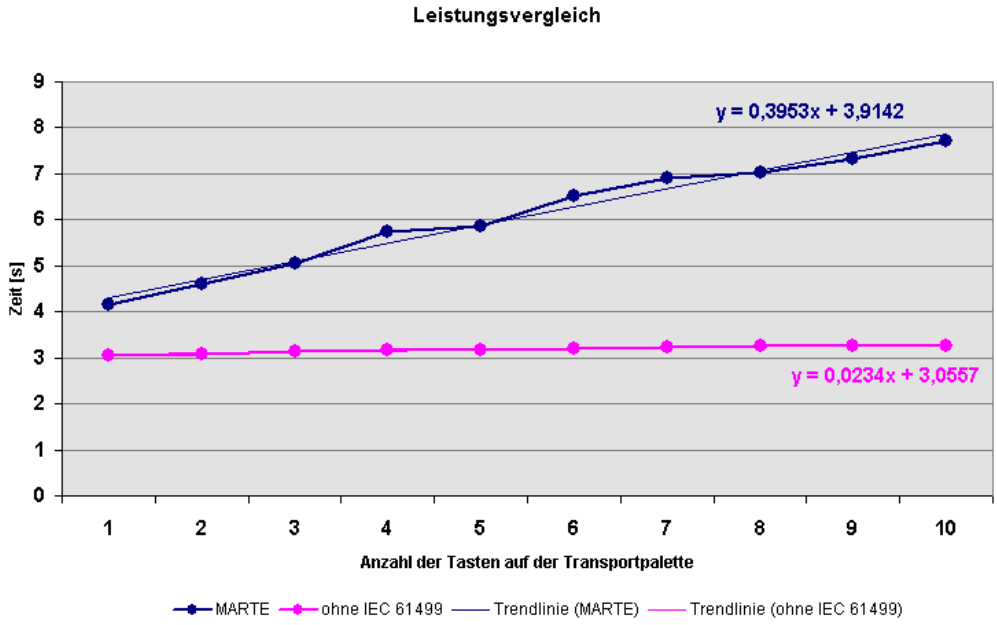


Abbildung 5.1: Laufzeitmessungen der Demonstratoranwendung mit und ohne IEC 61499 Overhead.

Vorverarbeitungen, wie z.B. Binarisierung usw. durchzuführen und die Anwesenheit bzw. Position der Transportpalette zu detektieren. Ca. 80 % dieser Zeit geht bei den Bildverarbeitungsoperationen *Close* und *FillHoles* verloren, wodurch hier noch kaum Unterschiede zwischen den beiden Programmen aufscheinen.

Danach beginnt die Analyse der einzelnen Tasten (d.h. Untersuchung der einzelnen Bildregionen nach der Bildsegmentierung), wobei für jede gefundene Taste eine Aktivierung (Berechnung) eines Neuronalen Netzwerkes erfolgt. Bei der Analyse der einzelnen Regionen zeigt sich der IEC 61499 Overhead ziemlich stark. Die Analyse einer Taste benötigt in der IEC 61499 Programmversion ca. 0,4 Sekunden, wobei nur ca. 0,03 Sekunden hiervon außerhalb der Laufzeitumgebung, nämlich in den externen Bibliotheken verloren gehen.

Dieses Ergebnis zeigt doch große Overhead Verluste seitens der IEC 61499 Laufzeitumgebung.

Anmerkung: Dieser Mehraufwand durch IEC 61499 fällt hier quantitativ etwas stärker aus als er tatsächlich ist. Das Problem, d.h. die größten Zeitver-

luste, ist die Umwandlung der Bildpixeldaten in die Eingangsdaten der Neuronalen Netzwerke. Dieses erfolgt nämlich im IEC 61499 Programmteil und nicht innerhalb einer C/C++ Bibliothek[2], weshalb alle Daten aus der Bildverarbeitungsbibliothek in IEC 61499 „heraufgeholt“ werden müssen, hier in passende Daten für die Neuronalen Netzwerke konvertiert werden, um abschließend wieder in die Neuronale Netzwerke Bibliothek „hinunterkopiert“ zu werden.

[2]Da die externen Bildverarbeitungs- und die Neuronale Netzwerke Bibliotheken voneinander unabhängige Produkte sind, bietet keine der beiden eine passende Konvertierungsfunktionen an. Eine solche ließe sich natürlich selbst ausprogrammieren und als weitere C/C++ (Hilfs-)Bibliothek hinzunehmen, was aber hier unterlassen wurde.

6 Ausblick

Der IEC 61499 Standard für die Automatisierungstechnik ist noch relativ jung - zum Zeitpunkt der Diplomarbeit waren erst drei Jahre seit der Standardisierung vergangen - und es werden sicher noch viele Jahre vergehen, bis er genauso verbreitet ist, wie sein Vorgänger der IEC 61131 Standard. Auch seitens der Industrie ist „eher abwarten, wie es sich entwickelt“ das Credo. Vorerst sind die akademischen Einrichtungen - durch diverse Forschungsprojekte, Diplom- bzw. Doktorarbeiten - die treibende Kraft bei der Weiterentwicklung dieses vielversprechenden Standards.

Durch diese Arbeit wurde ein weiteres großes Gebiet in der Automatisierung, nämlich die industrielle Bildverarbeitung, in Bezug auf IEC 61499 untersucht und weitere wertvolle Erfahrungen für das Voranschreiten des jungen Standards gesammelt.

Das Wichtigste für die Verbreitung des IEC 61499 Standards sind jetzt frei verfügbare Entwicklungs- und Laufzeitumgebungen, damit Hersteller von Automatisierungsgeräten - besonders auch kleinere Betriebe - ohne große Investitionen den Umstieg auf - oder wenigstens das „Hineinschnuppern“ in - IEC 61499 riskieren. Hierzu brauchen sie aber - ohne große Umkosten - Werkzeuge.

Betreffend dieser Diplomarbeit war dies erst der Beginn der Integration von Bildverarbeitung in IEC 61499 und damit der industriellen Bildverarbeitung mittels IEC 61499. Für industriereife Anwendungen werden aber noch einige solcher Arbeiten notwendig sein. Die umgesetzten Bildverarbeitungsfunktionen bieten aber bereits einen soliden Grundstock für weitere Bilder verarbeitende Anwendungen.

Die Eigenschaften einer Bildverarbeitungsanwendung - zu Beginn von Kapitel 3 dargelegt - stellen besondere Anforderungen an eine IEC 61499 Laufzeitumgebung und im besonderen an ihre interne Daten-(Variablen-)Verwaltung und Ereignisabarbeitung, da die elektronische Bilderverarbeitung nicht das typische Einsatzgebiet von IEC 61499 ist. Einige dieser Probleme wurden bereits im Rahmen dieser Diplomarbeit (Kapitel 3 und 4 - im Besonderen im Abschnitt 4.2.3) aufgezeigt, jedoch nicht gelöst.

Im folgenden einige Vorschläge für Weiterführende Arbeiten:

Aufgrund der großen Datenmenge von Bildern wurde gezeigt, dass diese nur mittels Zeigern[1] zwischen Funktionsblöcken weitergereicht werden können. Der IEC 61499 Standard schreibt jedoch nicht vor, wie eine IEC 61499 Laufzeitumgebung den Datentransfer zwischen den Funktionsblöcken zu verwalten hat[2]. Durch die - im Falle von Bildverarbeitung notwendige - Unterstützung von Zeigern ergibt sich aber ein weiteres Problem, nämlich die Gültigkeitsdauer eines Zeigers.

Ob IEC 61499 Daten als Wert oder als Referenz behandeln soll, ist sehr anwendungsspezifisch[3] und kann somit am besten vom Anwender, d.h. Programmierer entschieden werden. Der IEC 61499 Standard sollte diesem die Wahl überlassen, ob ein Funktionsblock seinen Ausgang direkt als Wert (d.h. als eine Kopie des Datums) oder nur als Referenz (d.h. als einen Zeiger auf das gleiche Datum) dem nachfolgenden Funktionsblock übergeben möchte. Weitere Forschungsarbeiten könnten hier von Nöten sein, um mehr Licht in diesen „Graubereich“ des IEC 61499 Standards zu bringen.

Bedingt durch die hohen Ressourcenanforderungen von Bildverarbeitungsfunktionen, z.B. Kantendetektion usw. kann es bei Bildverarbeitungsanwendungen zu größeren Verzögerungen kommen. Diese sind - neben den großen Datenmengen - ein weiteres Problem für IEC 61499, da somit ein einzelner Funktionsblock die gesamte Ereignisabarbeitung des gesamten Funktionsblocknetzwerks blockieren kann. Auf diese Problematik wurde im Rahmen dieser Diplomarbeit jedoch nicht eingegangen und nachfolgende Arbeiten könnten sich im Detail damit auseinandersetzen.

Zum augenblicklichen Entwicklungsstand des IEC 61499 Standards sollte der IEC 61499 Programmierer die Eigenheiten und implementationstechnischen Details der einzusetzenden IEC 61499 Laufzeitumgebungen verstehen und beherrschen, bevor er erfolgreich IEC 61499 Anwendungen entwickeln möchte. Der IEC 61499 Standard wird ein wenig „enger geschnürrt“ werden müssen, um eine echte Middleware-Lösung zu werden, in welcher sowohl die Hard- und Low-Level-Software abstrahiert, als auch die „Individualität“ der IEC 61499 Laufzeitumgebung selbst weitestgehend beseitigt ist.

[1] als Referenzen

[2] z.B. ob als Wert oder Referenz

[3] Im Falle von Bildverarbeitungsanwendungen, ist jedoch der Einsatz von Referenzen die einzig mögliche Lösung.

7 Zusammenfassung

Die vorliegende Diplomarbeit befasste sich mit dem neuen Standard für die Automatisierungstechnik, IEC 61499 - „Function Blocks“ und die Integration von Bildverarbeitunsfunktionalität für industrielle Bildverarbeitungsanwendungen in diesem Standard.

Hierfür wurde eine „intelligente Kamera“ - ein sogenanntes *Smart Device*, welches Bildsensor und Mikrokontroller in einem vereint - das *Kompaktkamerasystem SBOx-C* -, in ein Automatisierungssystem basierend auf IEC 61499 integriert. Diese Integration inkludierte auch die Bereitstellung von Bildverarbeitungsfunktionen für IEC 61499, sodass zum Abschluss der Arbeiten eine Anwendundung zur Demonstration der Ergebnisse entwickelt werden konnte.

Zu Beginn wurden die Gemeinsamkeiten und Unterschiede der beiden „Welten“, industrielle Bildverarbeitung und IEC 61499 Funktionsblocknetzwerke, gegenübergestellt und die Schwierigkeiten beim Zusammenführen dieser ausgearbeitet.

Es wurden verschiede Möglichkeiten untersucht, das Kompaktkamerasystem in ein Automatisierungssystem einzubinden, wobei die Standards IEC 61131 und IEC 61499 näher betrachtet wurden. Besonders der Standard IEC 61499 wurde im Detail analysiert und der Weg der Einbettung der Bildverarbeitungsfunktionalität durchgehend - vom Konzept zur Implementation - dokumentiert. Für diese Einbettung wurden externe Bibliotheken für Bildverarbeitung und Neuronale Netzwerke eingesetzt, und der Zugriff auf diese durch die IEC 61499 Laufzeitumgebung ermöglicht, wobei zwei IEC 61499 Laufzeitumgebungenen - MARTE und FBRT - im Detail analysiert wurden.

Abschließend wurden noch einige ausgewählte Probleme detailierter ausgeführt, welche während den Implementationsarbeiten aufgetreten sind und die Ergebnisse der Arbeiten - mit Vor- und Nachteilen - aufgelistet.

A IEC 61499 Bildverarbeitungs-funktionsblöcke

A.0.1 Datentypen

Bevor auf die neu implementierten Funktionsblöcke eingegangen werden soll, werden einige neue Datentypen vorgestellt, welche in diesen eingesetzt werden.

IMAGE-Datentyp

Der IMAGE-Datentyp ist nur ein Zeiger (eine Referenz) in die zentrale Bildablage der IEC 61499 Laufzeitumgebung. Auf die Speicherung und Aufbewahrung der in IEC 61499 Anwendungen eingesetzten Bilder wurde in Abschnitt 4.2.1 bereits genauer eingegangen.

NN-Datentyp

Der NN[1]-Datentyp ist, in Analogie zum IMAGE-Datentyp ein Zeiger in die zentrale Ablage für Neuronale Netzwerke.

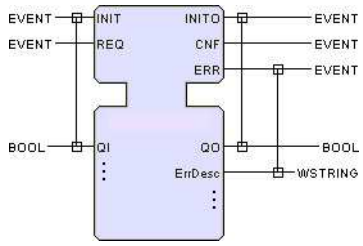
A.0.2 Funktionsblöcke

Sämtliche Funktionsblöcke wurden als Service-Interface Funktionsblöcke implementiert. Einige der Daten- bzw. der Ereignis-Ein- und Ausgänge haben alle erstellten Funktionsblöcke gemeinsam. Deswegen sollen - vor der eigentlichen Beschreibung der implementierten Funktionsblöcke - diese zuerst beschrieben werden.

Basis-Funktionsblock

Virtueller Basisfunktionsblock zur Beschreibung der gemeinsamen Ein- und Ausgänge aller kommenden Funktionsblöcke.

[1] Back-Propagation Neuronales Netzwerk



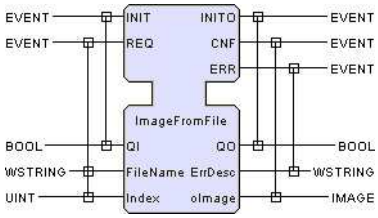
`INIT`:	Initialisierungseingang. Im allgemeinen ist keine 'echte' Initialisierung der Bildverarbeitungsblöcke notwendig. Es wird nur geprüft, ob die externe Bildverarbeitungsbibliothek erfolgreich geladen werden konnte. Nur wenige Funktionsblöcke, z.B. welche GUI Elemente initialisieren müssen, erfordern eine tatsächliche Initialisierung und Deinitialisierung.
`REQ`:	Anforderungseingang zum Starten der eigentlichen Funktion. Beim Eintreten dieses Ereignisses müssen alle Dateneingänge gültige Werte besitzen.
`QI`:	Parameter für die Initialisierung. `QI=true` → Initialisieren, `QI=false` → Deinitialisieren.
`INITO`:	Weiterschleifen des Initialisierungsereignisses beim Initialisieren.
`CNF`:	Bestätigung der Anforderung im Falle das keine Fehler aufgetreten sind.
`ERR`:	Fehlerindikation im Falle eines Fehlers. Sollte dieses Ereigniss ausgelöst werden, so kann über den Datenausgang `ErrDesc` eine Fehlerbeschreibung ausgelesen werden.
`QO`:	Weiterschleifen des Initialisierungsparameters: `QI=true` → Initialisieren, `QI=false` → Fehler oder Deinitialisieren.
`ErrDesc`:	Fehlerbeschreibung. Besitzt nur im Falle eines Fehlers einen gültigen Wert.

Im folgenden werden alle implementierten Funktionsblöcke detailiert - meistens mit Skizzen und manchmal auch mit Anwendungsbeispielen - beschrieben.

Ein- und Ausgabe Funktionsblöcke

ImageFromFile

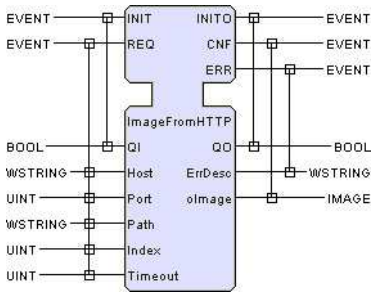
Laded eine Datei vom Dateisystem.



`Filename`:	Ein absoluter oder relativer Pfad zu der Bilddatei, welche geladen werden soll. Unterstützt werden folgende Dateitypen: `BMP`, `JPG`, `GIF`, `PNG`, `TGA` und `TIFF`. Ist die Pfadangabe relativ so ist der Bezugspfad das Startverzeichnis der **FBRT**.
`Index`:	Falls die Datei mehrere Bilder beinhaltet, z.B. Animated-GIF, so kann ein Index angegeben werden, welches Bild geladen werden soll. Normalerweise `Index=0`.
`oImage`:	Das geladene Bild.

ImageFromHTTP

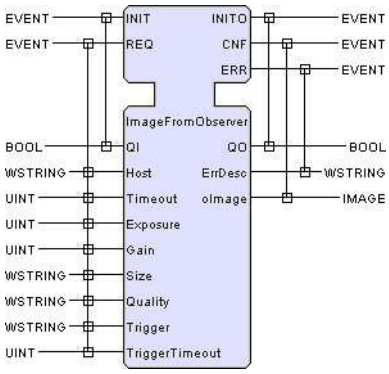
Laded eine Datei von einem beliebigen Web-Server über das HTTP Protokoll. Dieser Funktionsblock wurde nur für die FBRT IEC 61499 Laufzeitumgebung, aber nicht für die MARTE implementiert.



`Host`:	Hostname eines Webservers, z.B. `www.acin.tuwien.ac.at`.
`Port`:	Port des Webservers. Normalerweise `Port=80`.
`Path`:	Vollständiger Pfad zur Bilddatei, z.B. . Unterstützt werden folgende Dateitypen: `BMP`, `JPG`, `GIF`, `PNG`, `TGA` und `TIFF`.
`Index`:	Falls die Datei mehrere Bilder beinhaltet, z.B. Animated-GIF, so kann ein Index angegeben werden, welches Bild geladen werden soll. Normalerweise `Index=0`.
`Timeout`:	Maximale Wartezeit in `ms`.
`oImage`:	Das geladene Bild.

ImageFromObserver

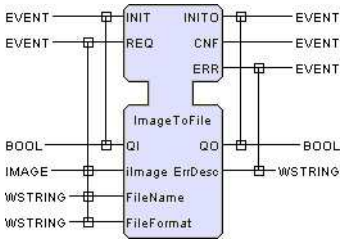
Laded ein Bild von einer Festo-Observer Kamera. Dieser Funktionsblock kann nur für IEC 61499 Anwendungen welche auf dem *Kompaktkamerasystem SBOx-C* ausgeführt werden, eingesetzt werden.



`Host`:	Hostname oder IP-Adresse der Kamera, z.B. `128.130.200.156`.
`Timeout`:	Maximale Wartezeit in `ms`.
`Exposure`:	Integrationszeit (Belichtungszeit) in `ms`. Je größer der Wert desto schwächer kann das Umgebungslicht sein. Jedoch dürfen sich dafür die Aufnahmemotive nicht allzu stark bewegen, sonst verwischen sich die Aufnahmen.
`Gain`:	Verstärkung der Bildhelligkeit. Übliche Werte zwischen `1-2` für ganz helle Räume bis `7-10` für Räume mit schwacher Beleuchtung. Zu hohe Verstärkungen führen jedoch zu großem Rauschen und sind nicht mehr praktikabel.
`Size`:	Bildgröße. Unterstützt werden drei Größen: `640x480`, `320x240` und `160x120`.
`Quality`:	Bild Qualität. Unterstützt werden folgende Eingaben: `low`, `moderate`, `good` und `excellent`.
`Trigger`:	Aufnahmetrigger kann entweder `software` sein, um gleich ein Bild aufzunehmen, oder `hardware` um auf ein Eingangssignal auf `E1` (graue Eingangsleitung) zu warten.
`Trigger`:	Maximale Zeit in `s` um auf Hardware-Aufnahmetrigger zu warten. Diese Größe ist nur relevant, wenn `Trigger=hardware` gesetzt wurde.
`oImage`:	Das geladene Bild.

ImageToFile

Speichert ein Bild in eine Datei im Dateisystem.

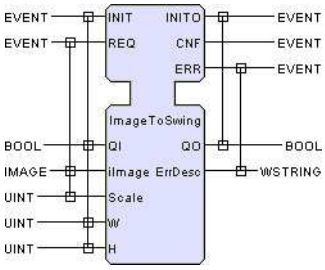


`iImage`:	Das zu speichernde Bild.
`Filename`:	Ein absoluter oder relativer Pfad zu der Datei, welche geschrieben werden soll. Ist die Pfadangabe relativ so ist der Bezugspfad das Startverzeichnis der **FBRT**.

`FileFormat`:	Dateiformat. Unterstützt werden folgende Dateitypen: `BMP`, `JPG`, `GIF`, `PNG`, `TGA` und `TIFF`.

ImageToSwing

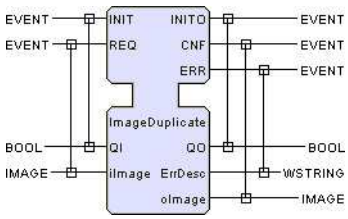
Stellt ein Bild am Bildschirm (mittels der Java-Swing-Klassen) dar. Dieser Funktionsblock muß innerhalb einer `PANEL_RESSOURCE` Resource vorkommen. Auch dieser Funktionsblock wurde nur für die FBRT IEC 61499 Laufzeitumgebung implementiert, da die MARTE keine grafische Benutzeroberfläche besitzt.



`iImage`:	Das auszugebende Bild.
`Scale`:	Verkleinerungfaktor. Zur Zeit wird dieser Parameter jedoch nicht unterstützt, d.h. `Scale=1`.
`W`:	Bildbreite.
`H`:	Bildhöhe.
	Die Parameter `W` und `H` müssen bereits bei der Initialisierung durch das `INIT` Ereignis gültig sein und können sich daher nicht dynamisch an die Bildgröße anpassen.

ImageDuplicate

Kopiert ein Bild.

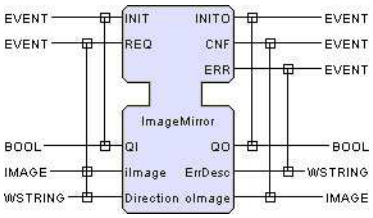


`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

Geometrische Funktionen

ImageMirror

Spiegelt ein Bild horizontal und/oder vertikal.



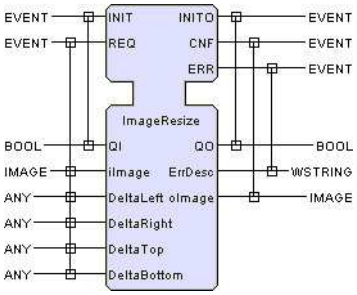
Eingangsbild:

Ausgangsbild bei horizontaler und vertikaler Spiegelung:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Direction`:	Spiegelungs-Richtung: `HORIZONTAL`, `VERTICAL` oder `BOTH`.

ImageResize

Beschneidet bzw. erweitert den Bildbereich ohne das Bild zu verändern. Im Falle einer Bilderweiterung werden die neuen Pixel mit `0=Schwarz` gefüllt.



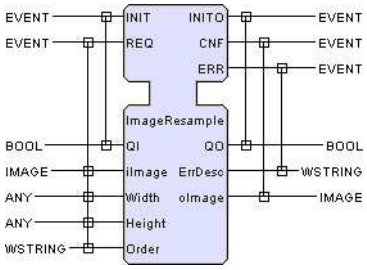
Eingangsbild:

Ausgangsbild bei `DeltaLeft=-47`, `DeltaRight=-100`, `DeltaTop=5` und `DeltaBottom=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`DeltaLeft`:	Beschneidung (=negativer Wert) bzw. Erweiterung (=positiver Wert) der linken Bildseite.
`DeltaRight`:	Beschneidung (=negativer Wert) bzw. Erweiterung (=positiver Wert) der rechten Bildseite.
`DeltaTop`:	Beschneidung (=negativer Wert) bzw. Erweiterung (=positiver Wert) der oberen Bildseite.
`DeltaBottom`:	Beschneidung (=negativer Wert) bzw. Erweiterung (=positiver Wert) der unteren Bildseite.

ImageResample

Vergrößert bzw. Verkleinert das Bild.

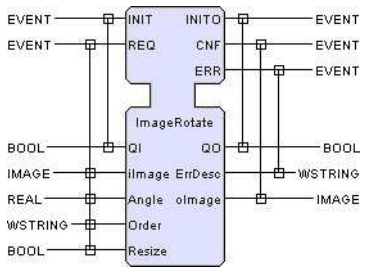


Eingangsbild:

Ausgangsbild bei `Width=150%`, `Height=50%` und `Order=BILINEAR`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Width`:	Neue Bild-Breite. Dies kann eine absolute (z.B. `150`) bzw. eine relative Eingabe (z.B. `60%`) sein.
`Height`:	Neue Bild-Höhe. Dies kann eine absolute (z.B. `150`) bzw. eine relative Eingabe (z.B. `60%`) sein.
`Order`:	Ordnung der Bildskalierung: `LINEAR`, `BILINEAR` oder `BICUBIC`. Je höher die gewählte Ordnung, desto weicher das Ausgangsbild.

ImageRotate Rotiert ein Bild und vergrößert - falls erwünscht - den Bildbereich, um keine Bildpixel durch die Rotation zu verlieren. Das Rotationszentrum ist die Bildmitte. Im Falle einer Bilderweiterung werden die neuen Pixel mit `0=Schwarz` gefüllt.



Eingangsbild:

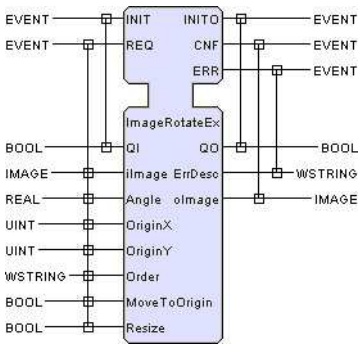
Ausgangsbild bei `Angle=-10`, `Resize=1` und `Order=BILINEAR`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Angle`:	Drehwinkel: positive Werte drehen im Uhrzeigersinn.
`Resize`:	`1` wenn das Bild vergrößert werden soll, falls sich durch die Rotation Bilddaten verloren gehen würden. Dies ist im allgemeinen der Fall, außer bei einer Drehung um $\pm 180°$.

`Order`:	Ordnung der Bildskalierung: `LINEAR`, `BILINEAR` oder `BICUBIC`. Je höher die gewählte Ordnung, desto weicher das Ausgangsbild.

ImageRotateEx

Rotiert ein Bild und vergrößert - falls erwünscht - den Bildbereich, um keine Bildpixel durch die Rotation zu verlieren. Hierbei kann im Gegensatz zum `ImageRotate`-Funktionsblock das Rotationszentrum frei gewählt werden. Im Falle einer Bilderweiterung werden die neuen Pixel mit `0=Schwarz` gefüllt.



Eingangsbild:

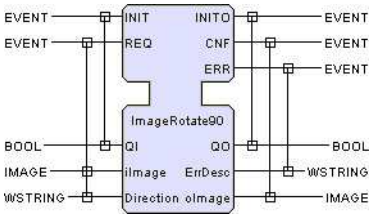
Ausgangsbild bei `Angle=-45`, `OriginX=0`, `OriginY=28`, `MoveToOrigin=1`, `Resize=1` und `Order=BILINEAR`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Angle`:	Drehwinkel: positive Werte drehen im Uhrzeigersinn.
`OriginX`:	X-Position des Drehzentrums.
`OriginY`:	Y-Position des Drehzentrums.
`Resize`:	`1` wenn das Bild vergrößert werden soll, falls sich durch die Rotation Bilddaten verloren gehen würden. Dies ist im allgemeinen der Fall, außer bei einer Drehung um $\pm 180°$.
`MoveToOrigin`:	`1` wenn nach der Drehung das Drehzentrum an die linke untere Ecke des Bildes verschoben werden soll.
`Order`:	Ordnung der Bildskalierung: `LINEAR`, `BILINEAR` oder `BICUBIC`. Je höher die gewählte Ordnung, desto weicher das Ausgangsbild.

ImageRotate90

Rotiert das Bild um 90° im oder gegen den Uhrzeigersinn.

Eingangsbild:

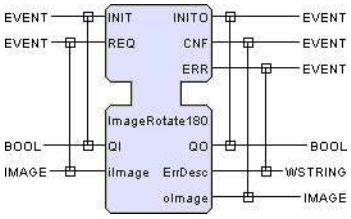


Ausgangsbild bei `Direction=CCW`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Direction`:	Drehrichtung: `CW` (Clockwise) oder `CCW` (Counterclockwise).

ImageRotate180

Rotiert das Bild um 180°.



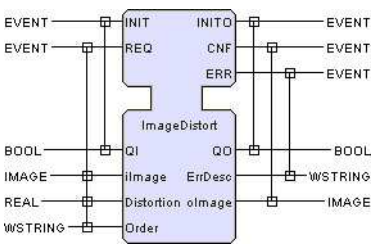
Eingangsbild:

Ausgangsbild:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

ImageDistort

Behebt linsenbedingte perspektivische Verzerrungen in einem Bild.



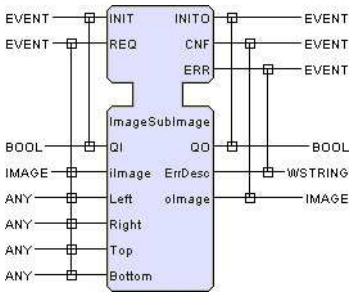
Eingangsbild:

Ausgangsbild bei `Distortion=3` und `Order=BILINEAR`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Distortion`:	Verzerrungsgrad. Negative Werte für konvexe (=nach außen) und positive für konkave (= nach innen) Verzerrungen.
`Order`:	Ordnung der Bildskalierung: `LINEAR`, `BILINEAR` oder `BICUBIC`. Je höher die gewählte Ordnung, desto weicher das Ausgangsbild.

ImageSubImage

Kopiert ein Teilbild aus einem Bild heraus.



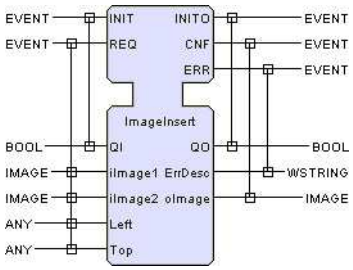
Eingangsbild:

Ausgangsbild bei `Left=5`, `Top=4`, `Right=40` und `Bottom=44`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Left`:	Linke Start-Position des Teilbildes (inkl.).
`Top`:	Obere Start-Position des Teilbildes (inkl.).
`Right`:	Rechte End-Position des Teilbildes (inkl.).
`Bottom`:	Untere End-Position des Teilbildes (inkl.).

ImageInsert

Fügt ein Teilbild in ein Bild ein.



Eingangsbilder:

= Zielbild

= einzufügendes Bild

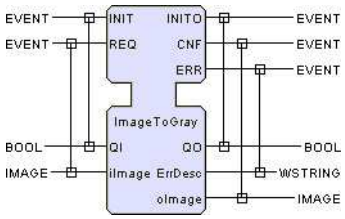
Ausgangsbild bei `Left=52` und `Top=0`:

iImage1:	Zielbild
iImage2:	Einzufügendes Bild
oImage:	Ausgangsbild
Left:	Linke Start-Position des Teilbildes.
Top:	Obere Start-Position des Teilbildes.

Farbverwaltungs-Funktionen

ImageToGray

Wandelt ein Eingangsbild in ein Graustufen-Bild um.



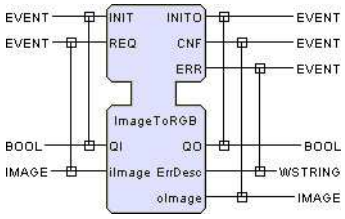
Eingangsbild:

Ausgangsbild:

iImage:	Eingangsbild
oImage:	Ausgangsbild

ImageToRGB

Wandelt ein Eingangsbild in ein RGB-Bild um.



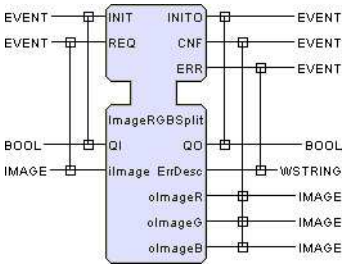
Eingangsbild:

Ausgangsbild:

iImage:	Eingangsbild
oImage:	Ausgangsbild

ImageRGBSplit

Teilt ein RGB-Bild in drei Graustufen-Bilder, wobei jedes Graustufenbild einer Farbkomponente entspricht.



Eingangsbild:

Ausgangsbild Rot:

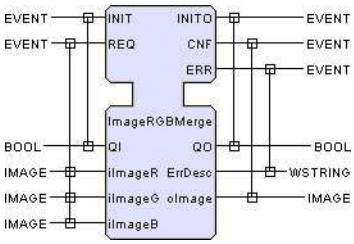
Ausgangsbild Grün:

Ausgangsbild Blau:

`iImage`:	Eingangsbild
`oImageR`:	Ausgangsbild Rot
`oImageG`:	Ausgangsbild Grün
`oImageB`:	Ausgangsbild Blau

ImageRGBMerge

Erstellt ein RGB-Bild aus drei Graustufen-Bildern, wobei jedes Graustufenbild als eine Farbkomponente interpretiert wird.



Eingangsbild Rot:

Eingangsbild Grün:

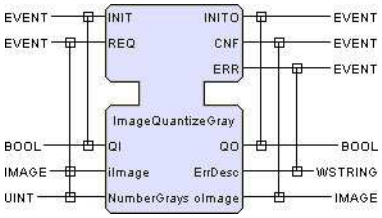
Eingangsbild Blau:

Ausgangsbild:

`iImageR`:	Eingangsbild Rot
`iImageG`:	Eingangsbild Grün
`iImageB`:	Eingangsbild Blau
`oImage`:	Eingangsbild

ImageQuantizeGray

Reduziert die Anzahl der unterschiedlichen Graustufen in einem Bild.



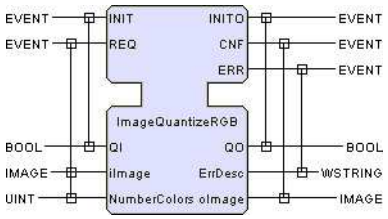
Eingangsbild:

Ausgangsbild bei `NumberColors=4`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`NumberColors`:	Anzahl der unterschiedlichen Graustufen nach der Reduktion.

ImageQuantizeRGB

Reduziert die Anzahl der unterschiedlichen Farben in einem Bild.



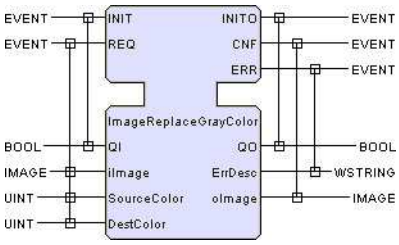
Eingangsbild:

Ausgangsbild bei `NumberColors=4`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`NumberColors`:	Anzahl der unterschiedlichen Farben nach der Farbreduktion.

ImageReplaceGrayColor

Ersetzt einen Graustufenton in einem Bild.



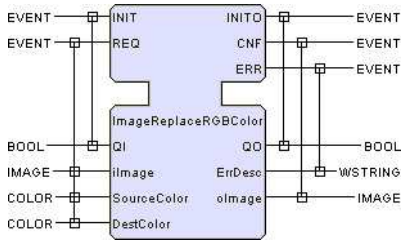
Eingangsbild:

Ausgangsbild bei `SourceColor=255` und `DestColor=0`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`SourceColor`:	Farbton der zu ersetzenden Farbe ≥ 0 und ≤ 255.
`DestColor`:	Neuer Farbton ≥ 0 und ≤ 255.

ImageReplaceRGBColor

Ersetzt eine Farbe in einem Bild.



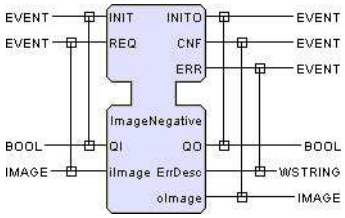
Eingangsbild:

Ausgangsbild bei `SourceColor=[255,255,255]` und `DestColor=[0,0,0]`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`SourceColor`:	Die zu ersetzende RGB-Farbe.
`DestColor`:	Neue Farbe.

ImageNegative

Erstellt das Negativ eines Bildes.



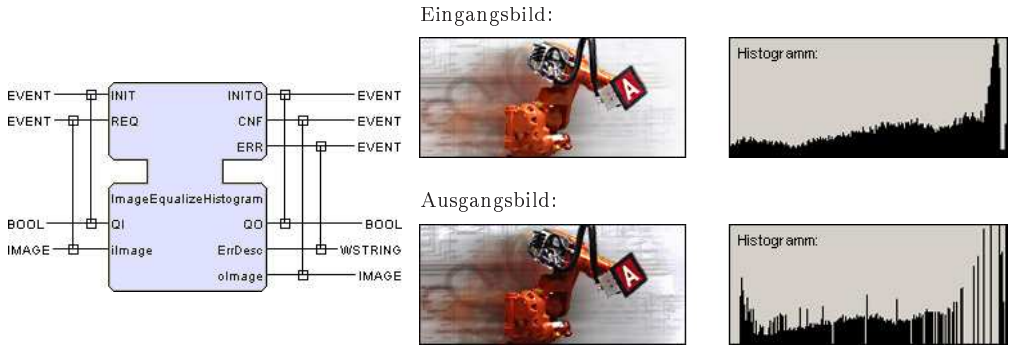
Eingangsbild:

Ausgangsbild:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

ImageEqualizeHistogram

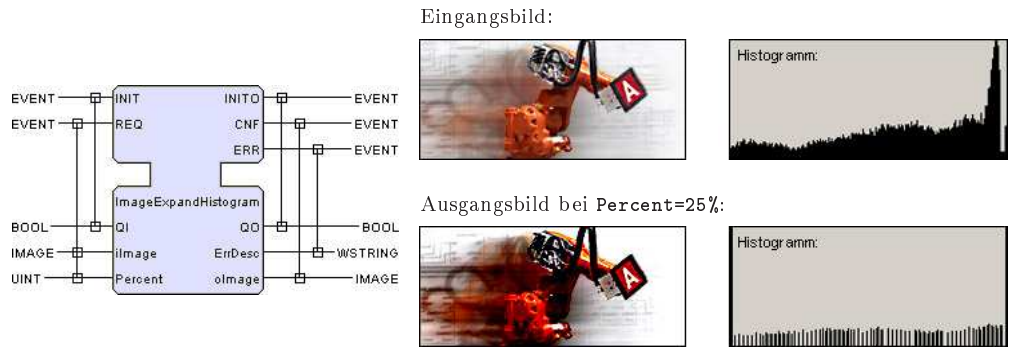
Erstellt eine automatische Tonwertkorrektur indem häufig vorkommende Farbpixel auf ein breiteres Spektrum gestreckt werden. Das Ergebnisbild nutzt den vorhandenen Farbraum besser aus.



`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

ImageExpandHistogram

Der Farbraum des Bildes wird gestreckt.

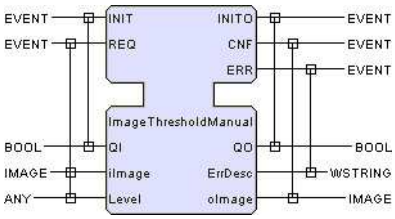


`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Percent`:	Prozentangabe für die Farbraumstreckung.

Binarisierungs-Funktionen

ImageThresholdManual

Binarisiert ein Graustufenbild mittels eines Schwellwerts.



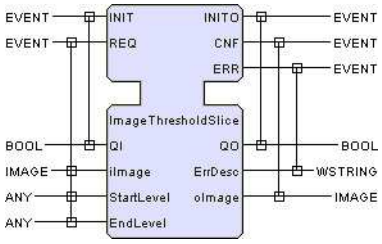
Eingangsbild:

Ausgangsbild bei `Level=128`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Level`:	Schwellwert: Grautöne unter dem Schwellwert werden schwarz, Grauwerte darüber weiß.

ImageThresholdSlice

Binarisiert ein Graustufenbild mittels zwei Schwellwerten.



Eingangsbild:

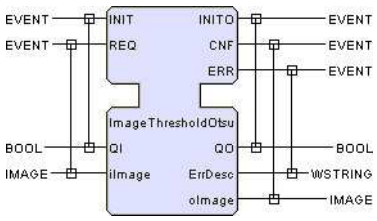
Ausgangsbild bei `StartLevel=51` (25%) und `EndLevel=191` (75%):

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`StartLevel`:	Start-Schwellwert: Grautöne unter dem Start-Schwellwert oder über dem End-Schwellwert werden schwarz, Grauwerte dazwischen weiß.

`StartLevel:`	End-Schwellwert: Grautöne unter dem Start-Schwellwert oder über dem End-Schwellwert werden schwarz, Grauwerte dazwischen weiß.

ImageThresholdOtsu

Binarisiert ein Graustufenbild automatisch ohne Schwellwertparameter nach dem Algorithmus von Nobuyuki Otsu aus dem Jahr 1979 [Ots79].



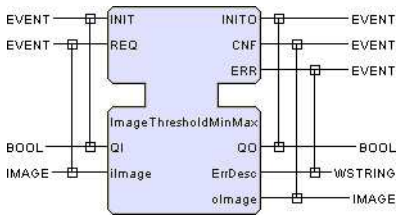
Eingangsbild:

Ausgangsbild:

`iImage:`	Eingangsbild
`oImage:`	Ausgangsbild

ImageThresholdMinMax

Binarisiert ein Graustufenbild automatisch, wobei der Schwellwert nach dem folgendem Algorithmus berechnet wird: $\frac{(max-min)}{2}$. max = der Graustufenwert mit der höchsten Häufigkeit und min = der Graustufenwert mit der niedrigsten Häufigkeit.



Eingangsbild:

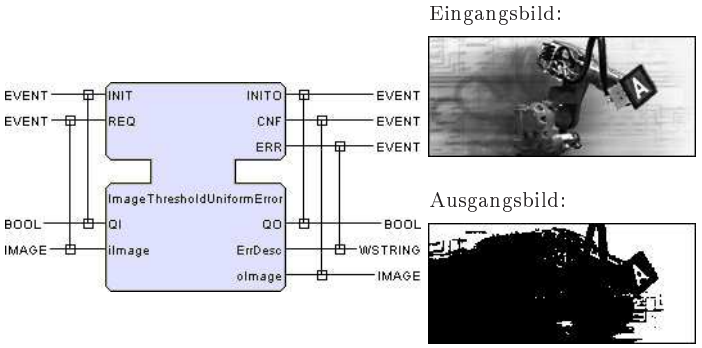
Ausgangsbild:

`iImage:`	Eingangsbild

`oImage:`	Ausgangsbild

ImageThresholdUniformError

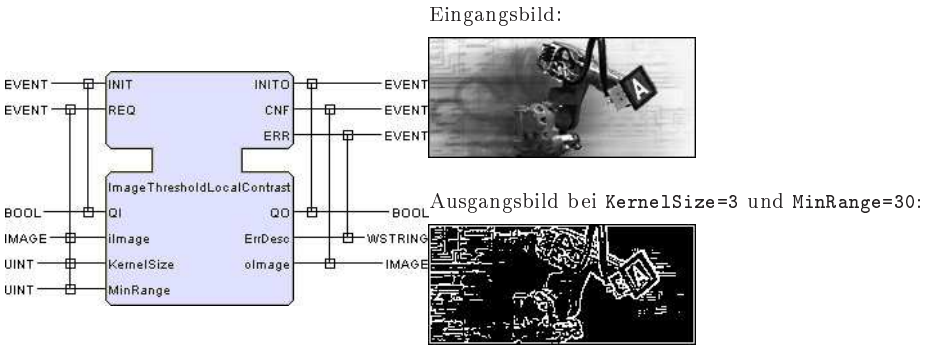
Binarisiert ein Graustufenbild automatisch ohne Schwellwertparameter nach dem [Dun84] Algorithmus.



`iImage:`	Eingangsbild
`oImage:`	Ausgangsbild

ImageThresholdLocalContrast

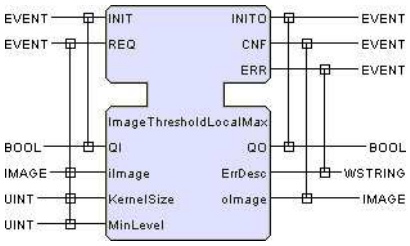
Binarisiert ein Graustufenbild nach dem [Ber86] Algorithmus und ist besonders gut zur Kantenhervorhebung geeignet.



`iImage:`	Eingangsbild
`oImage:`	Ausgangsbild
`KernelSize:`	Kernel Größe. Je kleiner desto feiner die Kontrastlinien.
`MinRange:`	Schwellwert für minimal Kontrast. Je kleiner desto körniger das Ergebnis.

ImageThresholdLocalMax

Binarisiert ein Graustufenbild, indem jeweils nur die lokalen Maxima (innerhalb einer gegeben Kernel-Größe) auf weiß gesetzt werden, ansonsten (wenn ein Pixel kein lokales Maximum darstellt) auf schwarz.



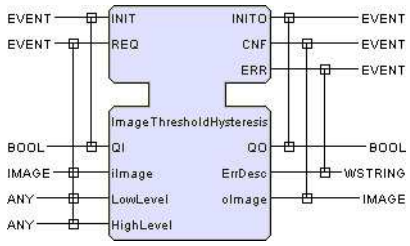
Eingangsbild:

Ausgangsbild bei `KernelSize=1` und `MinLevel=100`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für lokale Maximum Bedingung.
`MinLevel`:	Minimaler Schwellwert als Zusatzbedingung wenn die Bedingung für lokales Maximum erfüllt ist.

ImageThresholdHysteresis

Binarisiert ein Graustufenbild, indem bei einem oberen Schwellwert (Start-Schwellwert) begonnen wird, und solage benachbarte Pixel auf weiß gesetzt werden bis der untere Schwellwert unterschritten wird.



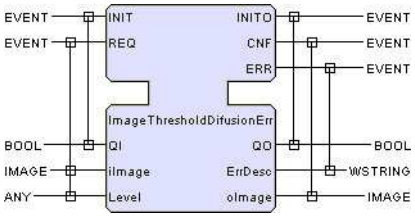
Eingangsbild:

Ausgangsbild bei `LowLevel=0` und `HighLevel=128`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`HighLevel`:	Oberer Schwellwert. Grauwerte welche über dem oberen Schwellwert liegen werden weiß gefärbt und all benachbarten Grauwerte ebenfalls, solange sie nur über dem unteren Schwellwert liegen.
`LowLevel`:	Unterer Schwellwert.

IMAGETHRESHOLDDIFUSIONERR

Binarisiert ein Graustufenbild, indem dessen Anzahl der Graustufe auf zwei reduziert wird, siehe Funktionsblock `ImageReduceGrayColor`. Dies entspricht einer Farbreduktion durch Dithering[2].



Eingangsbild:

Ausgangsbild bei `Level=64`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Level`:	Helligkeit. Je größer dieser Wert desto dünkler das Ergebnis.

Morphologische Funktionen

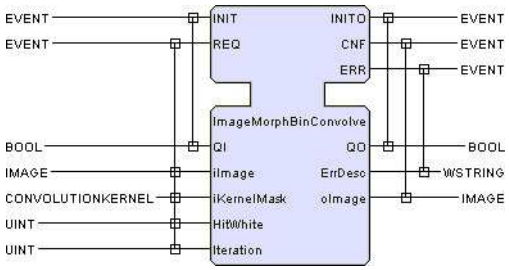
Binär

IMAGEMORPHBINCONVOLVE

Dies ist der Hauptfunktionsblock für binäre morphologische Funktionen. Als Eingabe Parameter wird eine $n \times n$ - Maske (`iKernelMask`) mit den Werten `1`, `0` oder `-1` erwartet. Über alle Pixel wird diese Maske gelegt und mit den benachbarten Pixel verglichen (am Rand wird das Bild temporär durch 0 Werte

[2]Das Dithering (englisch „to dither“: „schwanken“, „zittern“) ist eine Technik in der Computergrafik, um bei Bildern mit geringer Farbtiefe die Illusion einer größeren Farbtiefe zu erzeugen. Bei einem geditherten Bild werden die fehlenden Farben durch eine bestimmte Pixel-Anordnung aus verfügbaren Farben nachgebildet. Das menschliche Auge nimmt diese als Mischung der einzelnen Farben wahr.

erweitert). Für `-1` Werte in der Maske werden die betroffen Pixel ignoriert, d.h. sie können sowohl `1` als auch `0` sein. Kommt es zu einer Übereinstimmung, so wird der betroffene Pixel (der mittlere) entweder auf `1=Weiß` oder `0=Schwarz` gesetzt, je nach dem zweiten Eingabe Parameter `HitWhite`.



Eingangsbild:

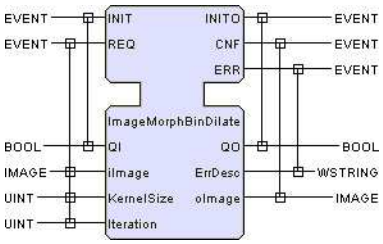
Ausgangsbild bei `iKernelMask=`

$$\left\{\begin{array}{ccccc} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{array}\right\}$$ und `HitWhite=0`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`iKernelMask`:	$n \times n$ - Maske als ein Array der Länge n^2. Die Werte im Raster dürfen `1`, `0` oder `-1` sein.
`HitWhite`:	Wenn `1` so wird bei Übereinstimmung der Pixel auf `1=Weiß` gesetzt, ansonsten auf `0=Schwarz`.
`Iteration`:	Anzahl der Wiederholungen.

ImageMorphBinDilate

Die weißen Pixel werden auf Kosten der schwarzen erweitert. Dies entspricht dem Aufruf des `ImageMorphBinConvolve`-Funktionsblocks mit den Parametern: `iKernelMask[`i`][`j`]=0` für $i = 0, 1, 2, \ldots n$ und $j = 0, 1, 2, \ldots n$ und `HitWhite=0`.



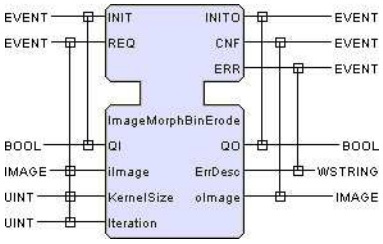
Eingangsbild:

Ausgangsbild bei `KernelSize=5` und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Erweiterung der weißen Pixel.
`Iteration`:	Anzahl der Wiederholungen.

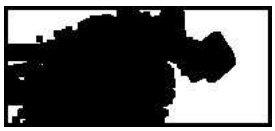
ImageMorphBinErode

Die schwarzen Pixel werden auf Kosten der weißen erweitert (die weißen werden erodiert). Dies entspricht dem Aufruf des `ImageMorphBinConvolve`-Funktionsblocks mit den Parametern: `iKernelMask[`i`][`j`]=1` für $i = 0, 1, 2, \ldots n$ und $j = 0, 1, 2, \ldots n$ und `HitWhite=1`.



Eingangsbild:

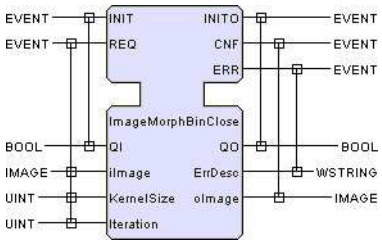
Ausgangsbild bei `KernelSize=5` und `Iteration=1`:



`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Erweiterung der schwarzen Pixel.
`Iteration`:	Anzahl der Wiederholungen.

ImageMorphBinClose

`ImageMorphBinDilate` + `ImageMorphBinErode`. Ausfransungen der schwarzen Pixel werden entfernt und die weißen Bereiche somit erweitert.



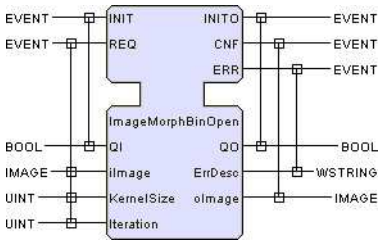
Eingangsbild:

Ausgangsbild bei `KernelSize=5` und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Reduktion der schwarzen Ausfransungen.
`Iteration`:	Anzahl der Wiederholungen.

ImageMorphBinOpen

`ImageMorphBinErode` + `ImageMorphBinDilate`. Ausfransungen der weißen Pixel werden entfernt und die schwarzen Bereiche somit erweitert.



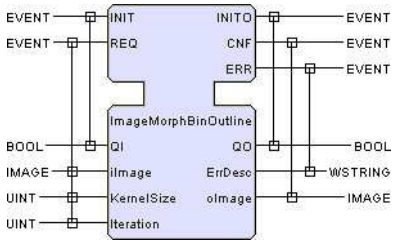
Eingangsbild:

Ausgangsbild bei `KernelSize=5` und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Reduktion der weißen Ausfransungen.
`Iteration`:	Anzahl der Wiederholungen.

ImageMorphBinOutline

`ImageMorphBinErode` + Differenz zwischen Original und `ImageMorphBinErode`.



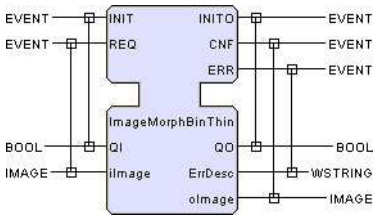
Eingangsbild:

Ausgangsbild bei `KernelSize=5` und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Hervorhebung der Kanten.
`Iteration`:	Anzahl der Wiederholungen.

ImageMorphBinThin

Morphologischer Algorithmus nach [Cyc94]. Dieser Funktionsblock ist zum Hervorheben von Kanten geeignet.



Eingangsbild:

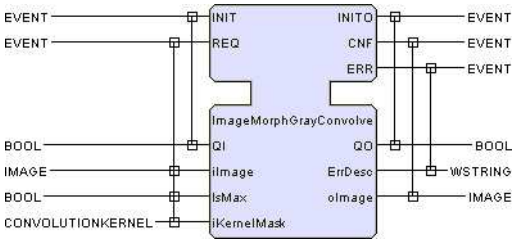
Ausgangsbild:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

Graustufen

ImageMorphGrayConvolve

Dies ist der Hauptfunktionsblock für morphologische Funktionen mit Graustufenbilder. Als Eingabe Parameter wird eine $n \times n$ - Maske (`iKernelMask`) erwartet. Über alle Pixel wird diese Maske gelegt und zu den benachbarten Pixel addiert (am Rand werden Pixel außerhalb des Bildes ignoriert). Für `-1` Werte in der Maske werden die betroffen Pixel ignoriert, d.h. es erfolgt keine Addition. Anschließend wird der betroffene Pixel (der mittlere), abhängig vom zweiten Eingabe Parameter `IsMax` das Maximum bzw. das Minimum der Maske.



Eingangsbild:

Ausgangsbild bei `iKernelMask=`

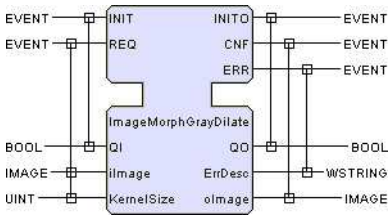
$$\begin{Bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{Bmatrix}$$

und `IsMax=0`, d.h. Minimum:

`iImage:`	Eingangsbild
`oImage:`	Ausgangsbild
`iKernelMask:`	$n \times n$ - Maske als ein Array der Länge n^2.
`IsMax:`	Wenn `1` so wird der Maximalwert der Maske genommen, ansonsten der Minimalwert.

ImageMorphGrayDilate

Die hellen Pixel werden auf Kosten der dunklen erweitert. Dies entspricht dem Aufruf des `ImageMorphGrayConvolve`-Funktionsblocks mit den Parametern: `iKernelMask[`i`][`j`]=0` für $i = 0, 1, 2, \ldots n$ und $j = 0, 1, 2, \ldots n$ und `IsMax=1`.



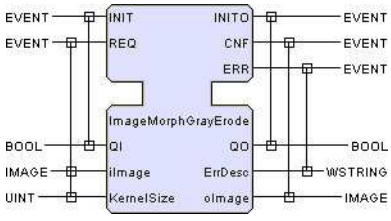
Eingangsbild:

Ausgangsbild bei `KernelSize=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Dominanz der hellen Pixel.

ImageMorphGrayErode

Die dunklen Pixel werden auf Kosten der hellen erweitert (die hellen werden erodiert). Dies entspricht dem Aufruf des `ImageMorphGrayConvolve`-Funktionsblocks mit den Parametern: `iKernelMask[`i`][`j`]=1` für $i = 0, 1, 2, \ldots n$ und $j = 0, 1, 2, \ldots n$ und `IsMax=0`.



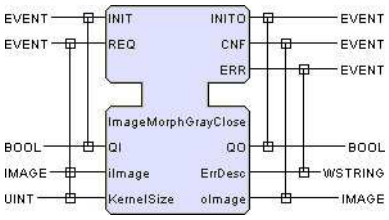
Eingangsbild:

Ausgangsbild bei `KernelSize=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Dominanz der dunklen Pixel.

ImageMorphGrayClose

`ImageMorphGrayDilate` + `ImageMorphGrayErode`. Ausfransungen der dunklen Pixel werden entfernt und die hellen Bereiche somit erweitert.



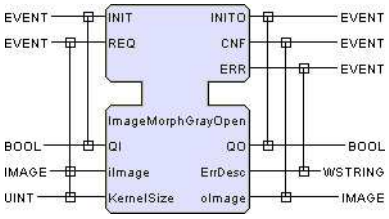
Eingangsbild:

Ausgangsbild bei `KernelSize=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Reduktion der dunklen Ausfransungen.

ImageMorphGrayOpen

`ImageMorphGrayErode` + `ImageMorphGrayDilate`. Ausfransungen der hellen Pixel werden entfernt und die dunklen Bereiche somit erweitert.



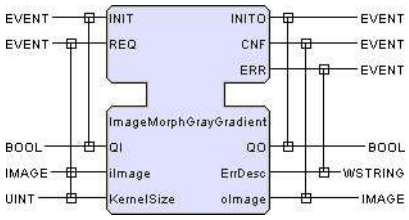
Eingangsbild:

Ausgangsbild bei `KernelSize=5` und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto stärker die Reduktion der hellen Ausfransungen.

ImageMorphGrayGradient

Differenz zwischen `ImageMorphGrayErode` und `ImageMorphGrayDilate`. Dieser Funktionsblock ist zum Hervorheben von Kanten geeignet.



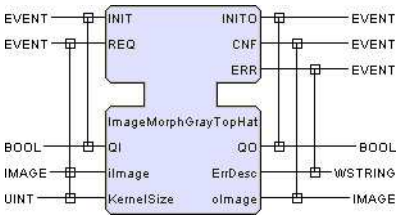
Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske. Je größer desto breiter die Hervorhebung der Kanten.

ImageMorphGrayTopHat

`ImageMorphGrayOpen` + Differenz zwischen Original und `ImageMorphGrayOpen`.



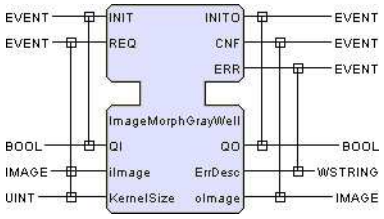
Eingangsbild:

Ausgangsbild bei `KernelSize=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske.

ImageMorphGrayWell

`ImageMorphGrayClose` + Differenz zwischen Original und `ImageMorphGrayClose`.



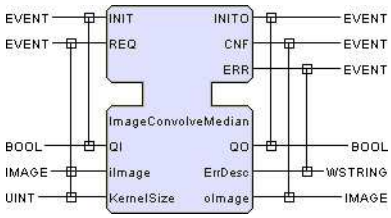
Eingangsbild:

Ausgangsbild bei `KernelSize=15`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Größe der Maske.

ImageConvolveMedian

Jeder Pixel wird der Mittelwert seiner umgebenden Pixel.



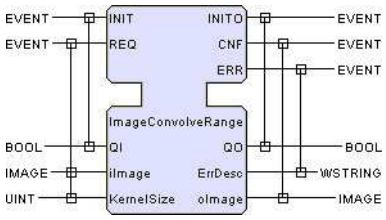
Eingangsbild:

Ausgangsbild bei `KernelSize=5`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für die Mittelwertberechnung.

ImageConvolveRange

Jeder Pixel wird die Differenz zwischen dem Maximum- und Minimumwert seiner umgebenden Pixel.



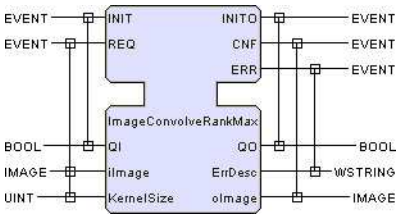
Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für die Differenzwertberechnung.

ImageConvolveRankMax

Jeder Pixel wird der Maximumwert seiner umgebenden Pixel.



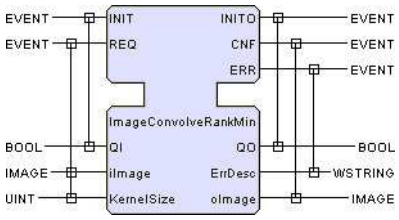
Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für die Maximumwertberechnung.

ImageConvolveRankMin

Jeder Pixel wird der Minimumwert seiner umgebenden Pixel.



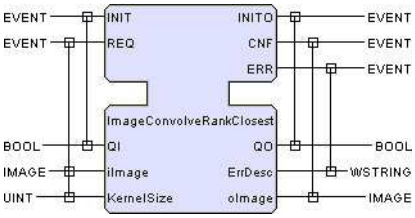
Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für die Minimumwertberechnung.

ImageConvolveRankClosest

Jeder Pixel wird entweder der Minimum- bzw. der Maximumwert seiner umgebenden Pixel. Je nachdem welches näher zum ursprünglichen Pixelwert liegt.



Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

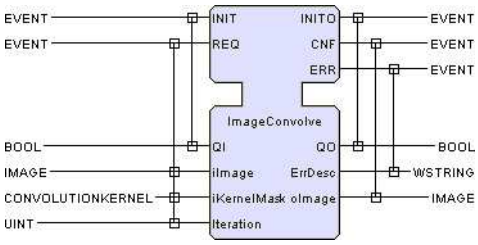
`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe für die Minimum- und Maximumwertberechnung.

Filter

ImageConvolve

Dies ist der Hauptfunktionsblock für Filter Funktionen mit Graustufenbilder. Hierbei werden zwei Graustufenbilder - das Original und eine Maske -

miteinander gefaltet. Je nach Maske lassen sich hiermit unterschiedliche Filter-Funktionen realisieren.



Eingangsbild:

Ausgangsbild bei `iKernelMask=Enhance` (siehe Nachfolgende Filter-Masken) und `Iteration=1`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`iKernelMask`:	$n \times n$ - Maske als ein Array der Länge n^2.
`Iteration`:	Anzahl der Wiederholungen.

Im folgenden werden einige oft eingesetzte Filter aufgelistet:

`Sobel` 3×3:	$\begin{Bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{Bmatrix}$
`Prewitt` 3×3:	$\begin{Bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{Bmatrix}$
`Kirsh` 3×3:	$\begin{Bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{Bmatrix}$
`Laplace 4` 3×3:	$\begin{Bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{Bmatrix}$
`Laplace 8` 3×3:	$\begin{Bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{Bmatrix}$
`Laplace 8` 5×5:	$\begin{Bmatrix} 0 & -1 & -1 & -1 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ -1 & 1 & 8 & 1 & -1 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & -1 & -1 & 0 \end{Bmatrix}$

`Laplace 48` 7×7:	$\begin{Bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 48 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{Bmatrix}$
`Gradiant` 3×3:	$\begin{Bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{Bmatrix}$
`Gradiant` 7×7:	$\begin{Bmatrix} 0 & -1 & -1 & 0 & 1 & 1 & 0 \\ -1 & -2 & -2 & 0 & 2 & 2 & 1 \\ -1 & -2 & -3 & 0 & 3 & 2 & 1 \\ -1 & -2 & -3 & 0 & 3 & 2 & 1 \\ -1 & -2 & -3 & 0 & 3 & 2 & 1 \\ -1 & -2 & -2 & 0 & 2 & 2 & 1 \\ 0 & -1 & -1 & 0 & 1 & 1 & 0 \end{Bmatrix}$
`Sculpt` 3×3:	$\begin{Bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
`Mittelwert` 3×3:	$\begin{Bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{Bmatrix}$
`Mittelwert` 5×5:	$\begin{Bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{Bmatrix}$
`Mittelwert` 7×7:	$\begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{Bmatrix}$
`Zirkularer Mittelwert` 3×3:	$\begin{Bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{Bmatrix}$
`Zirkularer Mittelwert` 5×5:	$\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}$

`Zirkularer Mittelwert` 7×7:

$$\left\{ \begin{array}{ccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array} \right\}$$

`Gauß` 3×3:

$$\left\{ \begin{array}{ccc} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{array} \right\}$$

`Gauß` 5×5:

$$\left\{ \begin{array}{ccccc} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{array} \right\}$$

`Barlett` 5×5:

$$\left\{ \begin{array}{ccccc} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 4 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{array} \right\}$$

`TopHat` 5×5:

$$\left\{ \begin{array}{ccccc} 0 & -1 & -1 & -1 & 0 \\ -1 & -1 & 3 & -1 & -1 \\ -1 & 3 & 4 & 3 & -1 \\ -1 & -1 & 3 & -1 & -1 \\ 0 & -1 & -1 & -1 & 0 \end{array} \right\}$$

`TopHat` 7×7:

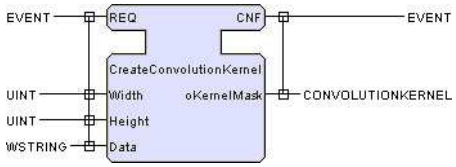
$$\left\{ \begin{array}{ccccccc} 0 & 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & -1 & -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & 3 & 3 & 3 & -1 & -1 \\ -1 & -1 & 3 & 4 & 3 & -1 & -1 \\ -1 & -1 & 3 & 3 & 3 & -1 & -1 \\ 0 & -1 & -1 & -1 & -1 & -1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 0 \end{array} \right\}$$

`Enhance` 5×5:

$$\left\{ \begin{array}{ccccc} 0 & -1 & -2 & -1 & 0 \\ -1 & -4 & 0 & -4 & -1 \\ -2 & 0 & 40 & 0 & -2 \\ -1 & -4 & 0 & -4 & -1 \\ 0 & -1 & -2 & -1 & 0 \end{array} \right\}$$

CreateConvolutionKernel

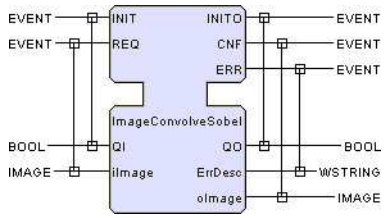
Erstellt eine Maske geeignet für Faltungsoperationen mittels des `ImageConvolution`-Funktionsblocks.



`Width`:	Breite
`Height`:	Höhe
`Data`:	Eingabe Array der Länge `Width` $*$ `Height`.
`oKernelMask`:	Maske geeignet als Eingabe-Parameter für den `ImageConvolve`-, den `ImageMorphBinConvolve`- und den `ImageMorphGrayConvolve`-Funktionsblock.

ImageConvolveSobel

Wendet den Sobel-Operator zur Kantenfindung an.



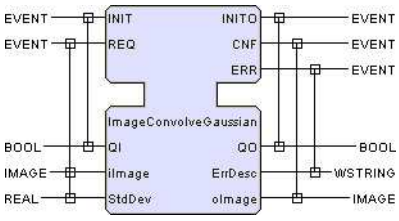
Eingangsbild:

Ausgangsbild:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild

ImageConvolveGaussian

Wendet den Gauß-Weichzeichnungs-Filter an.



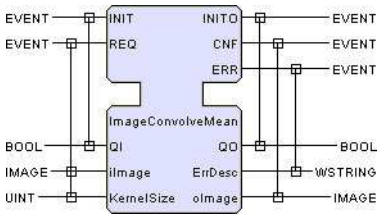
Eingangsbild:

Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe. Je größer desto stärker die Weichzeichnung des Bildes.

ImageConvolveMean

Wendet den zirkularen Mittelwert-Operator an.



Eingangsbild:

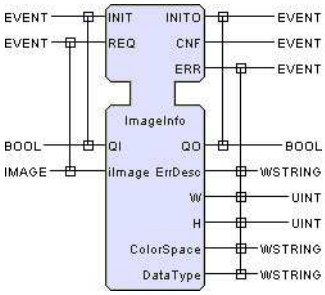
Ausgangsbild bei `KernelSize=3`:

`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`KernelSize`:	Kernel Größe.

Statistische Funktionen

ImageInfo

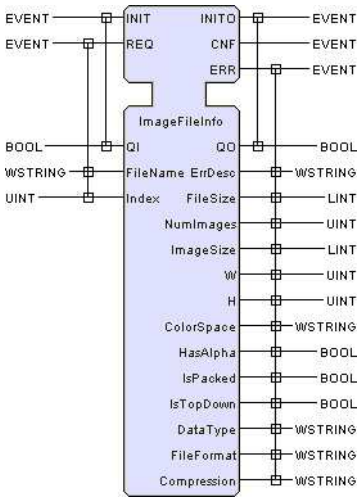
Bietet grundlegende Bildinformationen an.



`iImage`:	Eingangsbild
`W`:	Breite in Pixel
`H`:	Höhe in Pixel
`ColorSpace`:	Farbmodus: `RGB`, `MAP`, `GRAY` oder `BINARY`.
`DataType`:	Pixeldatentyp: `BYTE`, `USHORT`, `INT`, `FLOAT` oder `CFLOAT` (= komplex).

ImageFileInfo

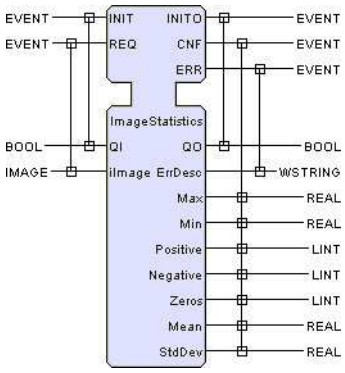
Bietet grundlegende Bildinformationen an. (Gleiche Funktion wie der `Image-Info`-Funktionsblock außer, dass als Eingabeparameter kein Bild sondern ein Dateiname erwartet wird.



`Filename:`	Ein absoluter oder relativer Pfad zu der Bilddatei, welche geladen werden soll. Unterstützt werden folgende Dateitypen: `BMP`, `JPG`, `GIF`, `PNG`, `TGA` und `TIFF`. Ist die Pfadangabe relativ so ist der Bezugspfad das Startverzeichnis der **FBRT**.
`Index:`	Falls die Datei mehrere Bilder beinhaltet, z.B. Animated-GIF, so kann ein Index angegeben werden, welches Bild geladen werden soll. Normalerweise `Index=0`.
`W:`	Breite in Pixel
`H:`	Höhe in Pixel
`ColorSpace:`	Farbmodus: `RGB`, `MAP`, `GRAY` oder `BINARY`.
`DataType:`	Pixeldatentyp: `BYTE`, `USHORT`, `INT`, `FLOAT` oder `CFLOAT` (= komplex).

ImageStatistics

Berechnet einige Bildstatistiken für ein Graustufenbild.



Eingangsbild:

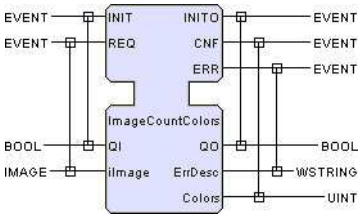
Ausgänge:

- Max: 255
- Min: 56
- Positive: 9635
- Negative: 0
- Zeros: 0
- Mean: 123.08687
- StdDev: 79.74906

`iImage`:	Eingangsbild
`Max`:	Höchster Graustufen-Ton. (`255=Weiß`)
`Min`:	Niedrigster Graustufen-Ton. (`0=Schwarz`)
`Positive`:	Anzahl der positiven Graustufen-Töne. Dies ist nur relevant für Bilder mit `Integer`-Pixel, welche auch negative Werte annehmen können.
`Negative`:	Anzahl der negativen Graustufen-Töne. Dies ist nur relevant für Bilder mit `Integer`-Pixel, welche auch negative Werte annehmen können.
`Zeros`:	Anzahl der `0` Werte im Bild.
`Mean`:	Mittelwert über alle Pixel.
`StdDev`:	Standardabweichung über alle Pixel.

ImageCountColors

Zählt die unterschiedlichen Farb- bzw. Graustufenwerte in einem Bild.



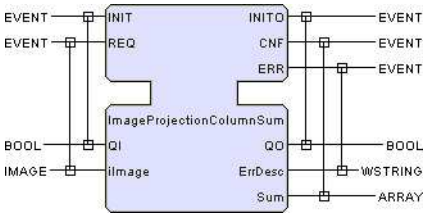
Eingangsbild:

Ausgang: 194
d.h. im Bild verstecken sich viele ähnliche Graustufentöne.

`iImage`:	Eingangsbild
`Colors`:	Anzahl der unterschiedlichen Farben.

ImageProjectionColumnSum

Berechnet die Summe der Graustufenwerte pro Spalte.



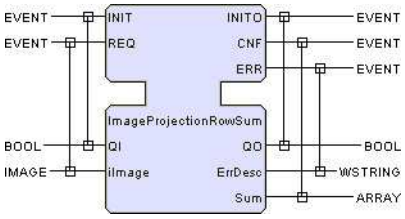
Eingangsbild:

Ausgang ist ein Array der Länge = Bildbreite. Hier jedoch als Diagramm dargestellt:

`iImage:`	Eingangsbild
`Sum:`	Array mit der Summe der Graustufen-Pixelwerte der einzelnen Bildspalten.

ImageProjectionRowSum

Berechnet die Summe der Graustufenwerte pro Zeile.



Eingangsbild:

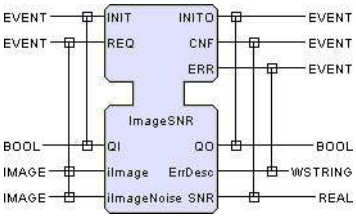
Ausgang ist ein Array der Länge = Bildhöhe. Hier jedoch als Diagramm dargestellt:

`iImage:`	Eingangsbild
`Sum:`	Array mit der Summe der Graustufen-Pixelwerte der einzelnen Bildzeilen.

ImageSNR

Berechnet die SNR[3] zwischen zwei Bilder: Signal- und Rauschbild. Das Rauschbild ergibt sich im allgemeinen durch mehrmaliges Aufzeichnen des selbigen Motives und anschließender Differenzbildung. Eigentlich sollte das selbe Motiv zum gleichen Bild führen. Die Unterschiede kann man als Rauschen bezeichnen, verursacht durch diverse Störungen.

[3] Signal-Noise-Ratio



Eingangsbilder:

= Signal

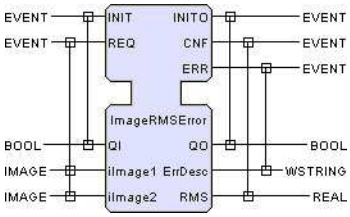
= Zufallsrauschen

Ausgang: 0.7031118 Dezibel

`iImage`:	Signalbild
`iImageNoise`:	Rauschbild.
`SNR`:	Signal-Rausch-Verhältnis in Dezibel.

ImageRMSError

Berechnet den RMS[4]-Fehler zwischen zwei Bildern.



Eingangsbilder:

(durch Zufallsrauschen etwas verändert)

Ausgang: 13.066103

`iImage1`:	Eingangsbild 1
`iImage2`:	Eingangsbild 2
`RMS`:	RMS-Fehler zwischen den eiden Eingangsbildern

ImageHistogram

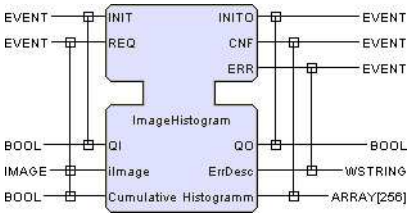
Zählt die unterschiedlichen Farb- bzw. Graustufenwerte in einem Bild.

[4] Root-Mean-Square

Eingangsbild:

Ausgang bei `Cumulative=0`:

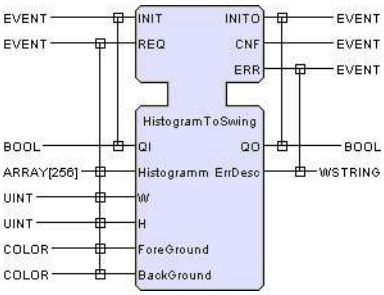
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	2	1	5	4	8	23
23	46	95	143	250	396	1003	1951
1733	418	165	101	66	51	26	11
9	2	6	5	3	2	5	5
5	6	3	5	3	3	4	5
3	6	8	11	14	10	11	8
4	5	2	6	4	3	3	0
1	2	1	0	2	2	3	3
3	4	3	2	3	2	8	26
4	7	1	3	6	15	19	7
1	4	2	2	2	6	0	4
1	1	2	1	2	4	2	1
2	1	2	2	3	2	4	0
1	4	2	2	2	1	3	7
5	2	1	1	4	3	0	3
3	7	2	18	9	4	4	2
6	8	5	2	2	6	2	5
1	7	2	2	1	1	3	3
4	3	5	2	1	3	3	1
0	7	4	4	6	3	5	3
3	2	7	4	2	6	2	5
3	17	4	5	7	5	4	4
5	4	8	3	2	5	11	9
10	6	11	6	11	15	21	33
55	67	123	173	302	379	317	909



`iImage`:	Eingangsbild
`Histogram`:	Das Histogramm des Bildes als ein Array der Länge 256.
`Cumulative`:	Wenn `1` so erfolgt die Zählung der Farben bzw. Graustufen kummulativ, d.h. für `255=Weiß` erhält man gleichzeitig auch die Pixel-Summe.

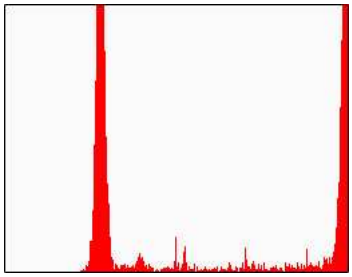
HistogramToSwing

Stellt ein Histogram am Bildschirm (mittels der Java-Swing-Klassen) dar. Dieser Funktionsblock muß ein innerhalb einer `PANEL_RESSOURCE` Resource vorkommen. Auch dieser Funktionsblock wurde nur für die FBRT IEC 61499 Laufzeitumgebung implementiert, da die MARTE keine grafische Benutzeroberfläche besitzt.



Eingangsbild:

Ausgang bei `W=300`, `H=200`, `ForeGround=[255,0,0]` und `BackGround=[250,250,250]`:

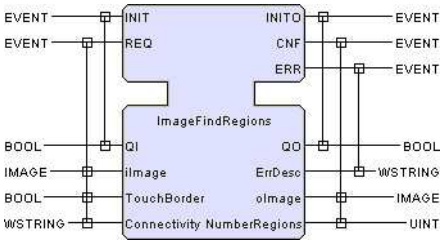


`Histogram:`	Eingangs-Histogramm
`W:`	Breite für die Darstellung.
`H:`	Höhe für die Darstellung.
`ForeGround:`	Vordergrund-Farbe.
`BackGround:`	Hintergrund-Farbe.

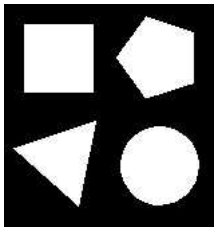
Analyse Funktionen

ImageFindRegions

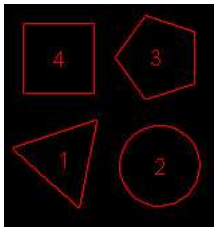
Findet zusammenhängende Regionen in einem Binär-Bild.



Eingangsbild:



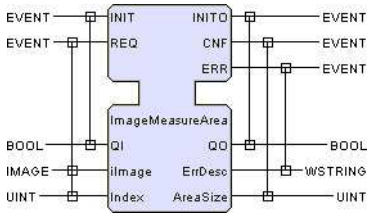
Ausgangsbild bei `TouchBorder=1` und `Connectivity=EIGHT`:
Im hier dargestellten Ausgangsbild sind die Pixel mit den Werten `1,2,3` und `4` mit Rot hervorgehoben, da ansonsten das menschliche Auge diese minimalen Graustufen Unterschiede nicht wahrnehmen kann.



`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild hat `unsigned short` Pixelwerte, wobei der Hintergrund = `0` ist und für jede gefundene Region die Pixelwerte auf `1,2,3,...` gesetzt werden.
`TouchBorder`:	Wenn `1` werden auch Regionen berücksichtigt, welche den Bildrand berühren.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EINE gemeinsame Region betrachtet.
`NumberRegions`:	Anzahl der gefundenen Regionen.

ImageMeasureArea

Berechnet die Pixelfläche einer Region.



Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

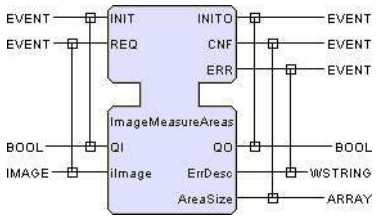
Ausgang bei `Index=0`: `1864`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.

`Index`:	Index der Region, mit `0` beginnend.
`AreaSize`:	Flächengröße in Pixel.

ImageMeasureAreas

Berechnet die Pixelflächen aller Regionen.



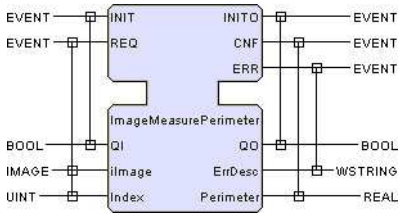
Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

Ausgang: `[1864, 2700, 2464, 2652]`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`AreaSize`:	Array der Flächengrößen in Pixel.

ImageMeasurePerimeter

Berechnet den Umfang einer Region.



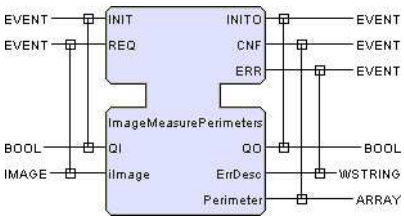
Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

Ausgang bei `Index=0`: `202.03056`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Index`:	Index der Region, mit `0` beginnend.
`Perimeter`:	Umfang der Region in Pixel.

ImageMeasurePerimeters

Berechnet die Umfänge aller Regionen.



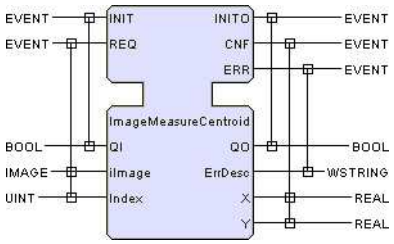
Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

Ausgang: `[202.03056, 192.16661, 195.13107, 202.0]`

`iImage:`	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Perimeter:`	Array der Umfänge in Pixel.

ImageMeasureCentroid

Berechnet den Schwerpunkt (das Zentrum) einer Region.



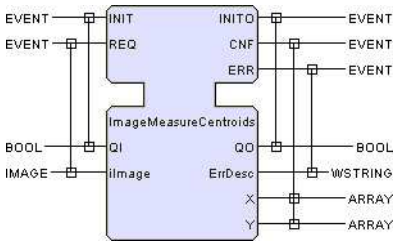
Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

Ausgänge bei `Index=0`:
`X = 43.53648`
`Y = 114.60676`

`iImage:`	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Index:`	Index der Region, mit `0` beginnend.
`X:`	X-Koordinate des Schwerpunkts der Region.
`Y:`	Y-Koordinate des Schwerpunkts der Region.

ImageMeasureCentroids

Berechnet die Schwerpunkte (die Zentren) aller Regionen.



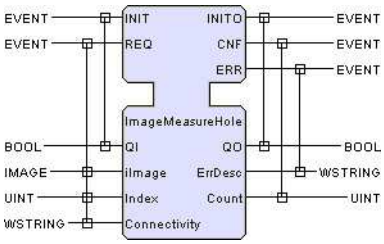
Eingangsbild:
Das Ausgangsbild aus dem Beispiel des `ImageFindRegions`-Funktionsblocks.

Ausgänge:
`X = [ 43.53648, 115.91926, 115.48011, 40.5]`
`Y = [114.60676, 119.94704, 39.48098, 40.0]`

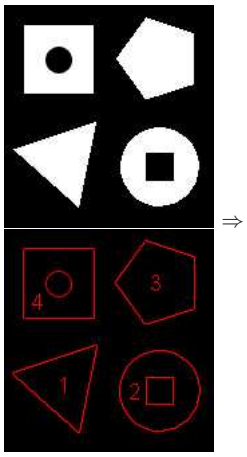
`iImage:`	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`X:`	Array der X-Koordinaten der Schwerpunkte der Regionen.
`Y:`	Array der Y-Koordinaten der Schwerpunkte der Regionen.

ImageMeasureHole

Berechnet die Anzahl der Löcher in einer Region, d.h. nicht dazugehörige Flächen innerhalb der Region.



Eingangsbild:
Als Eingangsbild wurde nochmals der `ImageFindRegions`-Funktionsblock mit den Eingangsparametern `TouchBorder=1` und `Connectivity=EIGHT` ausgeführt. Diesmal jedoch mit einem anderen Eingangsbild, welches Löcher innerhalb der Regionen enthält.

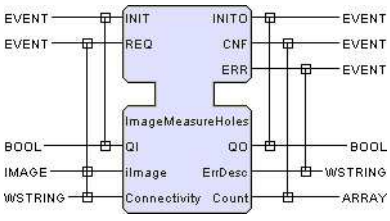


Ausgang bei `Index=1` und `Connectivity=EIGHT`: `1`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Index`:	Index der Region, mit `0` beginnend.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`Count`:	Anzahl der gefundenen Löcher in der Regionen.

ImageMeasureHoles

Berechnet die Anzahl der Löcher pro Region in allen Region, d.h. die nicht dazugehörigen Flächen innerhalb der Regionen.



Eingangsbild:
Das Eingangsbild aus dem Beispiel des `ImageMeasureHole`-Funktionsblocks.

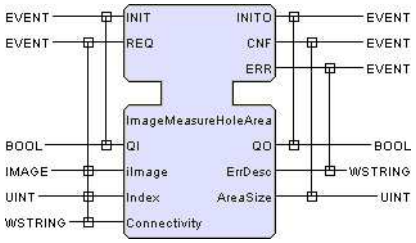
Ausgang bei `Connectivity=EIGHT`: `[0,1,0,1]`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.

`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`Count`:	Anzahl der Löcher in den Regionen.

ImageMeasureHoleArea

Berechnet die gemeinsame Pixelfläche aller Löcher in einer Region.



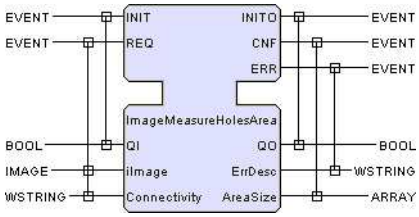
Eingangsbild:
Das Eingangsbild aus dem Beispiel des `ImageMeasureHole`-Funktionsblocks.

Ausgang bei `Index=1` und `Connectivity=EIGHT`: `441`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Index`:	Index der Region, mit `0` beginnend.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`AreaSize`:	Flächengröße der Löcher in der Region in Pixel.

ImageMeasureHolesArea

Berechnet die gemeinsame Pixelfläche aller Löcher pro Region in allen Regionen.



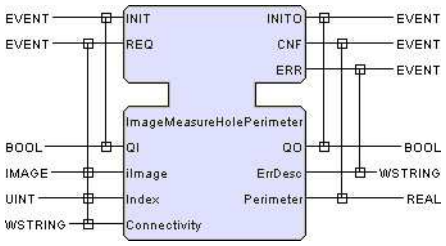
Eingangsbild:
Das Eingangsbild aus dem Beispiel des `ImageMeasureHole`-Funktionsblocks.

Ausgang bei `Connectivity=EIGHT`: `[0, 441, 0, 312]`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`AreaSize`:	Array der Flächengrößen der Löcher in den Regionen in Pixel.

ImageMeasureHolePerimeter

Berechnet den gemeinsamen Umfang aller Löcher in einer Region.



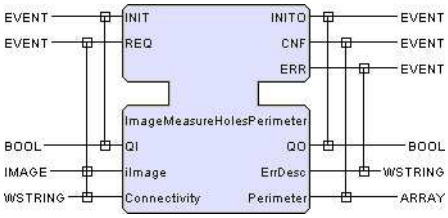
Eingangsbild:
Das Eingangsbild aus dem Beispiel des `ImageMeasureHole`-Funktionsblocks.

Ausgang bei `Index=1` und `Connectivity=EIGHT`: `80.0`

`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Index`:	Index der Region, mit `0` beginnend.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`Perimeter`:	Umfang der Löcher in der Region in Pixel.

ImageMeasureHolesPerimeter

Berechnet den gemeinsamen Umfang aller Löcher pro Region in allen Regionen.



Eingangsbild:
Das Eingangsbild aus dem Beispiel des `ImageMeasureHole`-Funktionsblocks.

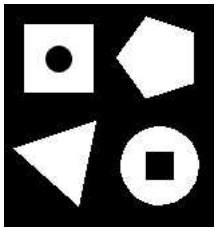
Ausgang bei `Connectivity=EIGHT`:
`[0.0, 80.0, 0.0, 62.627434]`

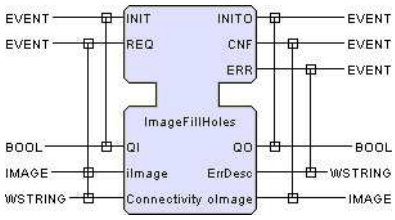
`iImage`:	Eingangsbild muss das Ausgangsbild eines vorangehenden `ImageFindRegions`-Funktionsblocks sein.
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EIN gemeinsames Loch in der Region betrachtet.
`Perimeter`:	Array der Umfänge der Löcher in den Regionen in Pixel.

ImageFillHoles

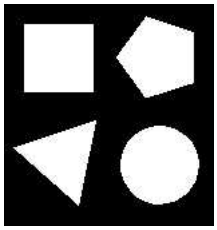
Füllt die Löcher innerhalb geschlossener Flächen in einem Binärbild aus.

Eingangsbild:





Ausgangsbild bei `Connectivity=EIGHT`:

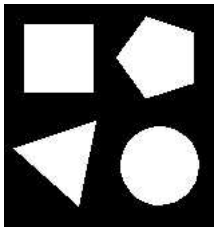


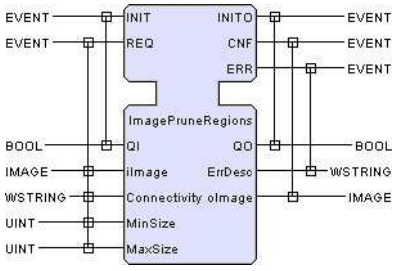
`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EINE geschlossene Region betrachtet.

ImagePruneRegions

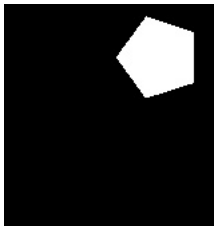
Löscht geschlossene Flächen in einem Binärbild, welche zu klein oder zu groß sind.

Eingangsbild:





Ausgangsbild bei `Connectivity=EIGHT`, `MinSize=2100` und `MaxSize=2500`:

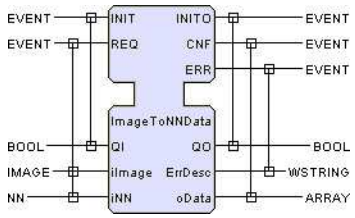


`iImage`:	Eingangsbild
`oImage`:	Ausgangsbild
`Connectivity`:	Kann die Werte `EIGHT` oder `FOUR` annehmen. Wenn `EIGHT` werden auch NUR diagonale Pixelberührungen als EINE geschlossene Region betrachtet.
`MinSize`:	Minimale Flächengröße einer Region, um diese Operation zu überleben.
`MaxSize`:	Maximale Flächengröße einer Region, um diese Operation zu überleben.

Bilderkennungs Funktionen (mittels Neuronalen-Netzwerken)

ImageToNNData

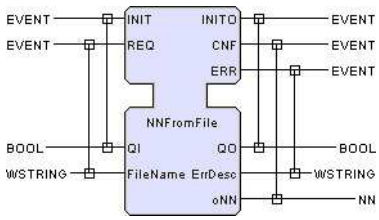
Erstellt aus einem Eingangsbild ein Daten-Array geeignet als Eingabe-Parameter für Bilderkennungs-Funktionsblöcke.



`iImage`:	Eingangsbild. Bildbreite $*$ Bildhöhe müssen mit den Eingangsneuronen des Neuronalen Netzwerks übereinstimmen.
`iNN`:	Neuronales Netzwerk.
`oData`:	Daten-Array geeignet als Eingabe-Parameter für den `NNTrain`- und den `NNRun`-Funktionsblock.

NNFromFile

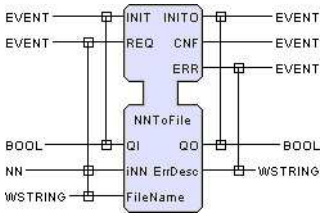
Ladet ein - zuvor trainiertes und gespeichertes - Neuronales Netzwerk.



`FileName`:	Ein absoluter oder relativer Pfad zu einer gespeicherten Neuronale-Netzwerk-Datei.
`oNN`:	Geladenes Neuronales-Netzwerk.

NNToFile

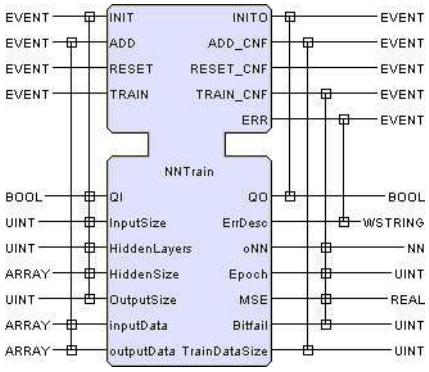
Speichert ein Neuronales Netzwerk.



`FileName`:	Ein absoluter oder relativer Pfad zu einer gespeicherten Neuronale-Netzwerk-Datei.
`iNN`:	Neuronales-Netzwerk welches gespeichert werden soll.

NNTrain

Trainiert ein Neuronales Netzwerk bzw. bereitet dessen Trainingsdaten vor.

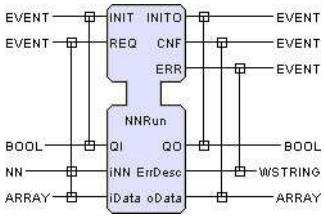


`INIT`:	Initialisiert das Neuronale Netzwerk.
`ADD`:	Fügt dem Neuronalen-Netzerk Trainingsdaten zu.
`RESET`:	Setzt das Neuronale-Netzerk auf den Initialisierungszustand zurück.
`TRAIN`:	Trainiert das Neuronale-Netzerk für einen Durchlauf.
`ADD_CNF`:	Bestätigt das Hinzufügen der Trainingsdaten.
`RESET_CNF`:	Bestätigt das Zurücksetzen des Neuronalen-Netzerks.
`TRAIN_CNF`:	Bestätigt einen Trainingsdurchlauf des Neuronalen-Netzerks. Als Ausgabe-Parameter wird der aktuelle Fehler (`MSE` bzw. `BitFail`) nach dem Trainingsdurchlauf ausgegeben. Dieser sollte sehr gering (≈ 0) sein, damit das Training abgebrochen werden kann. Ansonsten sollte ein weiterer Trainingsdurchgang erfolgen.

`InputSize`:	Anzahl der Eingagsneuronen. Diese Zahl muss gleich der Zeilen- $*$ der Spaltenzahl der zu verwendenden Bilder sein.
`HiddenLayers`:	Anzahl der versteckten Schichten im Neuronalen Netzwerk.
`InputSize`:	Anzahl der Neuronen in den versteckten Schichten. Die Eingabe erfolgt über ein Array der Länge der Anzahl der versteckten Schichten.
`OutputSize`:	Anzahl der Ausgagsneuronen.
`inputData`:	Eingangsdaten als Array. Dies sollte der Ausgang eines vorgeschalteten `ImageToNNData`-Funktionsblocks sein.
`outputData`:	Gewünschte Ausgangsdaten bei diesen angelegten Eingangsdaten.
`oNN`:	Neuronales Netzwerk.
`Epoch`:	Aktueller Trainingsdurchgang.
`MSE`:	Aktueller Fehler nach dem letzten Trainingsdurchgang.
`Bitfail`:	Aktuelle anzahl der Neuronen-Fehler nach dem letzten Trainingsdurchgang.
`TrainDataSize`:	Anzahl der hinzugefügten Trainingsdatensätze.

NNRun

Führt ein Neuronales Netzwerk aus, d.h. es wird versucht die Daten eines angelegten Bildes zu klassifizieren.



`INIT`:	Initialisiert das Neuronale Netzwerk.
`iNN`:	Das auszuführende Neuronale Netzwerk.
`iData`:	Eingangsdaten als Array. Dies sollte der Ausgang eines vorgeschalteten `ImageToNNData`-Funktionsblocks sein.
`oData`:	Errechnete Ausgangsdaten bei diesen angelegten Eingangsdaten.

B Dialoge der *Teaching*- und der *Run*-Anwendung

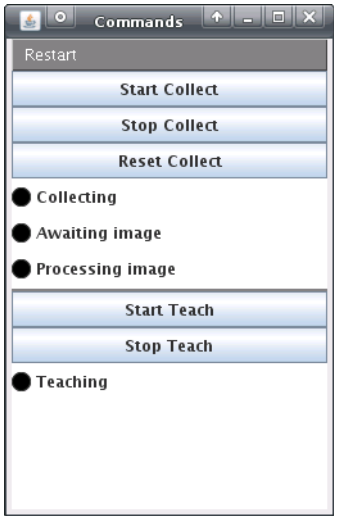


Abbildung B.1: Eingabedialog zur Steuerung der *Teaching*-Anwendung.

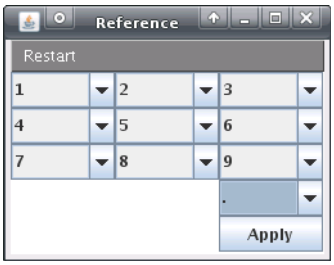


Abbildung B.2: *Teaching*-Anwendung: Eingabe der Referenz-(Soll-)werte.

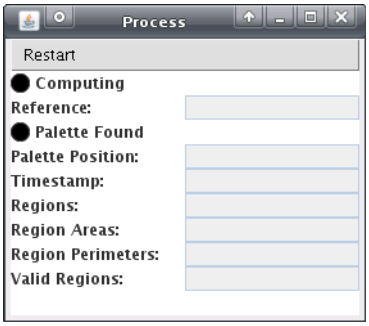


Abbildung B.3: *Teaching*-Anwendung: Anzeige der Statusinformationen der zuletzt getätigten Aufnahme.

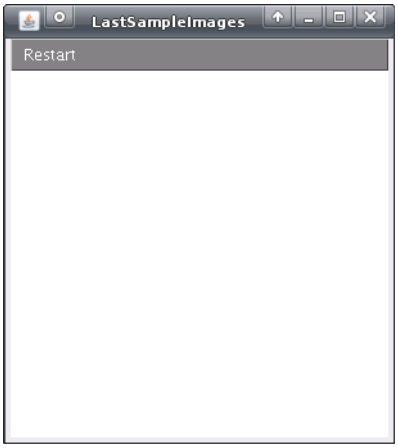


Abbildung B.4: *Teaching*-Anwendung: Anzeige der zuletzt aufgenommenen Referenzbilder.

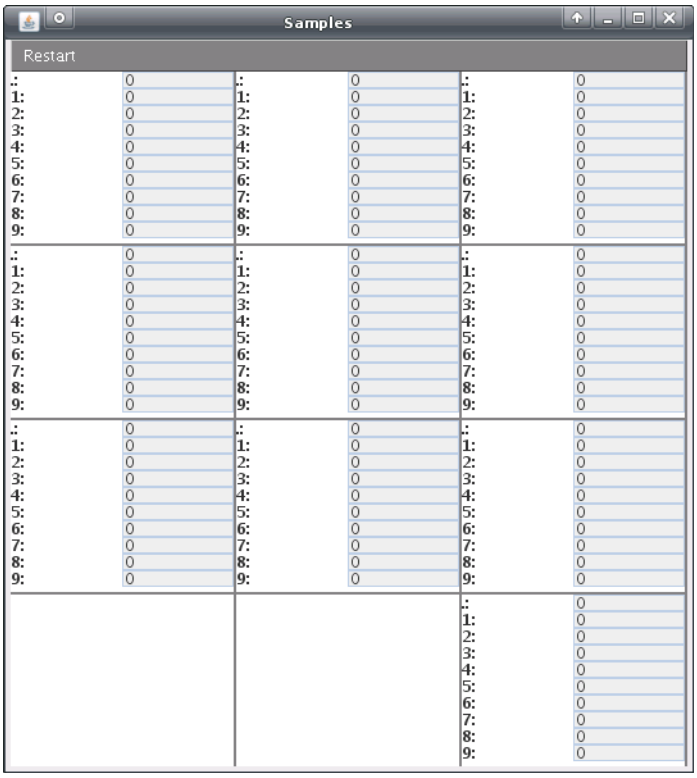


Abbildung B.5: *Teaching*-Anwendung: Anzeige der bereits vorhandenen Referenzbilder.

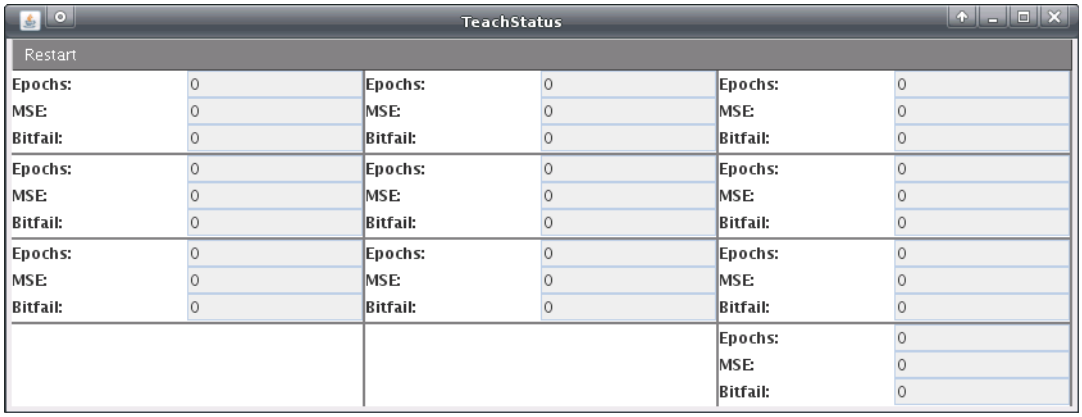


Abbildung B.6: *Teaching*-Anwendung: Anzeige des Trainingsfortschritts.

Abbildung B.7: *Teaching*-Anwendung: Speichern der Neuronalen Netzwerke.

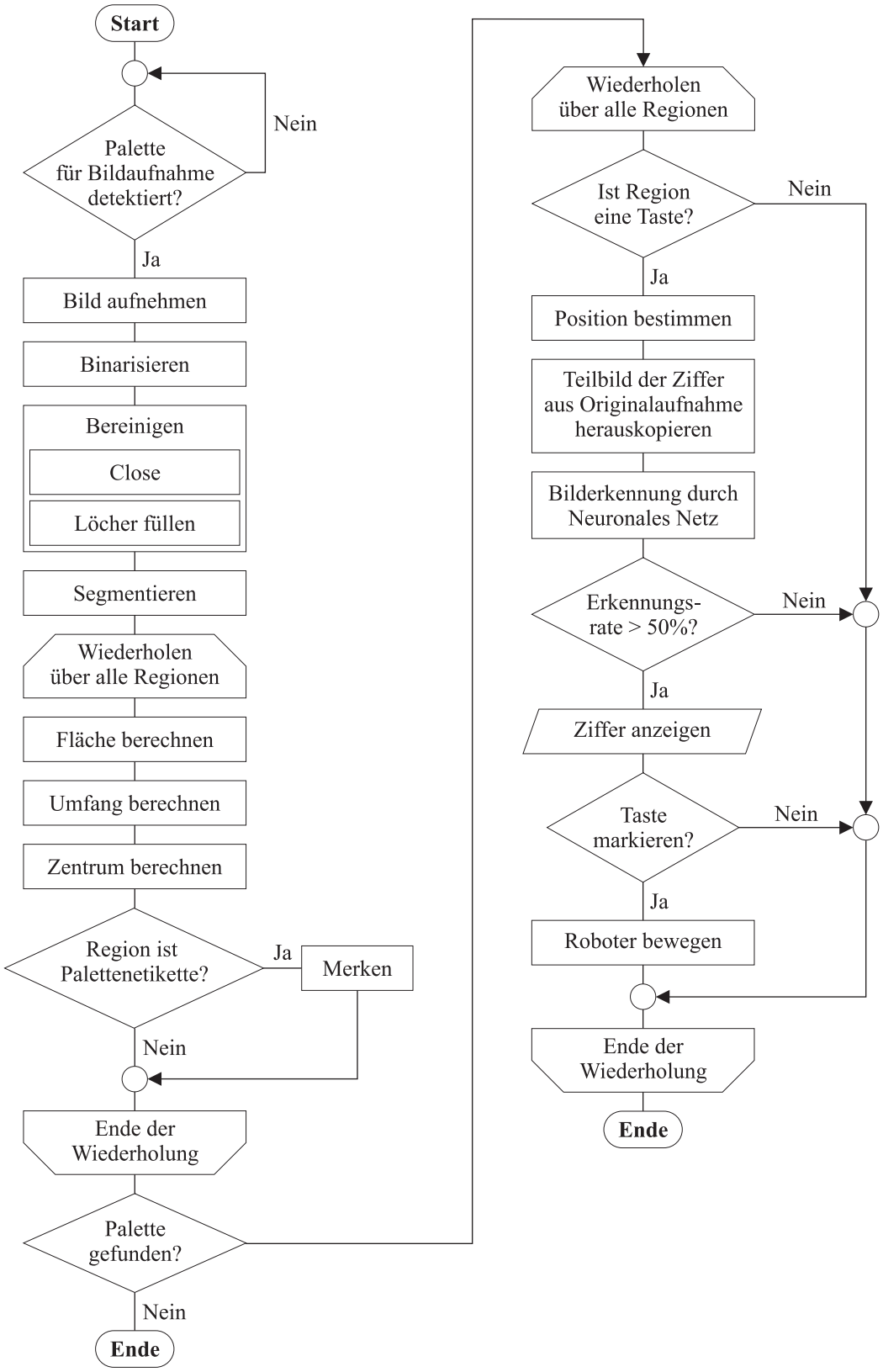


Abbildung B.8: Programmablaufplan der *Run*-Anwendung.

Literaturverzeichnis

[Bass04] Bässmann, H., Kreyss, J. *Bildverarbeitung Ad Oculos*, 4. Auflage, Springer, 2004

[Ber86] Bernsen, J. *Dynamic thresholding of grey-level images*, Proc. of the 8th ICPR, Paris, October 1986

[Cyc94] Cychosz, J. M. *Efficient Binary Image Thinning using Neighborhood Maps*, Academic Press, 1994

[Dun84] Dunn, S. M., Harwood, D., Davis, L. S. *Local Estimation of the Uniform Error Threshold*, IEEE Transactions on PAMI, November 1984

[Fav04] Favre-Bulle, B. *Automatisierung komplexer Industrieprozesse*, 1. Auflage, Springer, Wien – New York, 2004

[Fst08] *Kompaktkamerasystem SBOX-M*, FESTO, 2008. PDF Dokument von `http://www.festo.com/cat/de_de/data/doc_de/PDF/DE/SBOX-M_DE.PDF`

[Joh01] John, K.-H., Tiegelkamp, M. *IEC 61131-3: Programming Industrial Automation Systems*, Springer, Berlin, 2001

[Kin94] Kinnebrock , W. *Neuronale Netze - Grundlagen, Anwendungen, Beispiele*, 2. Auflage, Oldenbourg, 1994

[Lew01] Lewis, R. *IEC 61499: Modelling control systems using IEC 61499*, The Institution of Electrical Engineers, London, 2001

[Lia99] Liang, S. *Java(TM) Native Interface: Programmer's Guide and Specification*, Addison Wesley, 1999

[Meh05] Mehofer, F. *Cooperating Autonomous Linear Axes in Distributed Automation Systems*, Diplomarbeit, Technische Universität, Wien, 2005

[Neu98] Neumann, P., Grötsch, E., Lubkoll, C., Simon, R. *SPS-Standard: IEC 1131 - Programmierung in verteilten Automatisierungssystemen*, 2. Auflage, R. Oldenbourg, München – Wien, 1998

[Ots79] Otsu, N. *A threshold selection method from grey level histograms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, 1979

[PLC01] International Electrotechnical Commision, *Programmable Controllers*, 2001

[PLC05] *PLCOpen Motion Control*, PLCOpen, 2005. PDF Dokument von `http://www.plcopen.org/pages/tc2_motion_control/downloads/tf_mc_part1_version1_1.pdf`

[PNS07] PONS. *Großwörterbuch Englisch*, 1. Auflage, Klett, 2007

[Son08] Sonka, M., Hlavac, V., Boyle, R. *Image Processing, Analysis and Machine Vision*, 3. Edition, CENGAGE-Engineering, 2008

[Sun06] Sünder, C., Zoitl, A., Christensen, J.H., Vyatkin, V., Brennan, R.W., Valentini, A., Ferrarini, L., Strasser, T., Martinez-Lastra, J.L., Auinger, F. *Usability and Interoperability of IEC 61499 based distributed automation systems*, Industrial Informatics, IEEE International Conference 2006

[VDI00] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik *Bildverarbeitung im industriellen Einsatz*, 1. Auflage, VDI Verein Deutscher Ingenieure, Düsseldorf, 2000

[Vya07] Vyatkin, V. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*, O3NEIDA - Instrumentation Society of America, 2007

[Wei07] Weissler, G. *Einführung in die industrielle Bildverarbeitung*, 1. Auflage, Franzis, 2007

[Wie98] Wiener, O., Bonik, M., Hödicke, R. *Eine elementare Einführung in die Theorie der Turing-Maschinen*, 1. Auflage, Springer, Wien – New York, 1998

[www4DC] PROFACTOR GmbH. *4DIAC - Open Source for Distributed Industrial Automation* `http://www.fordiac.org`, Online: Zugriff erfolgte am 1. Juni 2008

[wwwCDS] Smart Software Solutions. *CoDeSys - The IEC 61131-3 Programming System* `http://www.3s-software.com/`, Online: Zugriff erfolgte am 23. Mai 2008

[wwwFN] Nissen, S. *FANN - Fast Artificial Neural Network Library* `http://leenissen.dk/fann/`, Online: Zugriff erfolgte am 22. Mai 2008

[wwwGND] McLauchlan, P. F. *Gandalf: The Fast Computer Vision and Numerical Library* `http://gandalf-library.sourceforge.net/`, Online: Zugriff erfolgte am 6. Juni 2008

[wwwHB] Christensen, H. *HOLOBLOC, Inc.* `http://www.holobloc.com`, Online: Zugriff erfolgte am 15. Mai 2008

[wwwIM] Scuri, A. E. *IM - An Imaging Toolkit* `http://www.tecgraf.puc-rio.br/im/`, Online: Zugriff erfolgte am 6. Mai 2008

[wwwISG] ICS Triplex. *Automation Solutions Worldwide* `http://www.icstriplex.com/`, Online: Zugriff erfolgte am 1. Juni 2008

[wwwMNS] PROFACTOR GmbH. *Micro Holons for Next Generation Distributed Automation and Control* `http://www.microns.org`, Online: Zugriff erfolgte am 15. Mai 2008

[wwwOCV] Intel. *OPENCV - Intel Open Source Computer Vision Library* `http://www.intel.com/technology/computing/opencv/`, Online: Zugriff erfolgte am 6. Juni 2008

[wwwVRT] *Videor Technical* `http://www.videortechnical.com/system/web/dt/products/product.php?area=ID&group=S84&subgroup=S8420&product=59701`, Online: Zugriff erfolgte am 23. Mai 2008

[Zei97a] Zeichen, G. *Prozessleittechnik*, Vorlesungsskriptum 361.317, Technische Universität, Wien, 1997

[Zei97b] Zeichen, G. *Flexible Automation*, Vorlesungsskriptum 361.042, Technische Universität, Wien, 1997

[Zoi02] Zoitl, A. *Development of an IEC 61499 Based Embedded Control Platform and Integration in a Distributed Automation System*, Diplomarbeit, Technische Universität, Wien, 2002

[Zoi06] Zoitl, A., Smodic, R., Sunder, C., Grabmair, G. *Enhanced Real-Time Execution of Modular Control Software based on IEC 61499*, Robotics and Automation, ICRA 2006. Proceedings 2006 IEEE International Conference 2006

Abbildungsverzeichnis

Abkürzungen

ACIN	Automation & Control Institute
ACON	Agile Control
API	Application Programming Interface
ARM	Acorn Risc Machine
AS	Ablaufsprache
AWL	Anweisungsliste
BMP	Bitmap
BV	Bildverarbeitung
CAN	Controller Area Network
CCD	Charge Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
ECC	Execution Control Chart
EDV	Elektronische Datenverarbeitung
FANN	Fast Artificial Neuronal Networks
FB	Funktionsblock
FBDK	Function Blocks Development Kit
FBRT	Function Blocks Runtime
FBS	Funktionsbausteinsprache
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GIF	Graphics Interchange Format
GNU	GNU's Not Unix
GPL	General Public License
HMI	Human Machine Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBV	Industrielle Bildverarbeitung
IDE	Integrierte Entwicklungsumgebung
IEC	International Electrotechnic Commision
IP	Internet Protocol
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group

JVM	Java Virtual Machine
KOE	Konfigurations-Organisationseinheit
KOP	Kontaktplan
LD	Ladder Logic
MARTE	Microns Advanced Runtime Environment
MMI	Mensch Maschine Interface
MSE	Mean Square Error
NN	Neuronales Netzwerk
OOP	Objektorientierte Programmierung
OSL	Odo Struger Labor
PAP	Programmablaufplan
PC	Personal Computer
PIKS	Programmer's Imaging Kernel System
PLC	Programmable Logic Controller
POE	Programm-Organisationseinheiten
RGB	Rot-Grün-Blau
SPS	Speicherprogrammierbare Steuerung
ST	Strukturierter Text
USB	Universal Serial Bus
VPS	Verbindungsprogrammierte Steuerung
WWW	World Wide Web
XML	Extensible Markup Language