Thomas Lauenstein

Programmierung und Test einer Systemsoftware zur Odometrischen Navigation von Robotern

BACHELORARBEIT

HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Mathematik, Naturwissenschaften, Informatik

Mittweida, 2010

Bibliographische Angaben

Lauenstein, Thomas: *Programmierung und Test einer Systemsoftware zur Odometrischen Navigation von Robotern,* 75 Seiten, Hochschule Mittweida, Fakultät Mathematik/ Naturwissenschaft/ Informatik

Bachelorarbeit, 2010

Referat

Die vorliegende Arbeit beschreibt die Entwicklung eines Selbstortungsalgorithmus für Roboter auf Basis eines von der Chemnitzer Werkstoffmechanik entwickelten digitalen Bildkorrelationsverfahrens. Dadurch werden die Nachteile bisher verwendeter Techniken zur Positionsbestimmung vermieden, was zu einer genaueren Ortung führt und damit auch zu einer besseren Navigation. Die Verminderung der Rechenzeit, benötigter Systemressourcen, Entwicklung von Korrekturverfahren und die Berechnung trigonometrischer Daten, sind die wichtigsten Bestandteile, um den bereits existierenden Korrelationsalgorithmus anzupassen.

Inhalt

A Abbildungsverzeichnis

B Formelverzeichnis

C Listingverzeichnis

D Tabellenverzeichnis

1. Einleitung

Die Chemnitzer Werkstoffmechanik GmbH (CWM), in dem die vorliegende Bachelorarbeit entstand, versteht sich als Dienstleister für Forschungs- und Entwicklungsleistungen auf dem Gebiet der Werkstoffmechanik. Die Unternehmensschwerpunkte liegen in der

- Werkstoffprüfung, Werkstoffanalytik und Bauteilpräparation (Materialographie)
- Schwingungsmesstechnik
- Finite-Elemente-Methode (FEM) Simulation
- Deformationsanalyse und korrelationsgestützte Bildverarbeitungstechnologien.

Wesentliche Produkt- und Verfahrensentwicklungen der CWM entstanden auf dem Sektor der digitalen Bildkorrelation für die Analyse von Verschiebungen und Deformationen. Diese stellen für die CWM eine Schlüsseltechnologie dar. Dafür wird das Softwaresystem *VEDDAC* und die Messsysteme *uniDAC* unter der Markenbezeichnung *microDAC®* für die digitale Bildkorrelation entwickelt und vertrieben. [CWM10]

Die Chemnitzer Werkstoffmechanik GmbH hat es sich zur Aufgabe gemacht, ein neues Verfahren zur automatischen Navigation und Positionsbestimmung mobiler Plattformen (Roboter) zu entwickeln, welches auf dem Prinzip der digitalen Bildkorrelation basiert. Mit diesem Prinzip soll eine genauere und dynamischere Navigation als mit den in Kapitel 2.1 beschriebenen herkömmlichen Methoden zur Positionsbestimmung möglich werden. Dieses Verfahren soll es ermöglichen, auf größere bauliche Veränderungen bzw. Präparationen (Anbringung aufwendiger Markierungen) am Einsatzort des Roboters, verzichten zu können. Einige mögliche Einsatzzwecke sind:

- Überwachungs- und Sicherheitsroboter
- Reinigungsroboter an für Menschen nur schwer zugänglichen oder gefährlichen orten
- Dienstleistungen wie Rasenmäher oder Lagerverwaltung.

Ein hierzu im Juli 2009 gestartetes Projekt mit trägt den Kurznamen *Correlation Robotics (CORRO)*. Es wird außer von der CWM von zwei weiteren Firmen mitbearbeitet, welche die Aufgabe haben, den mechanischen Aufbau und die Fahrwerksteuerung einer autonomen, mobilen Plattform zu entwickeln (Kolmann und Haberland GmbH Stuttgart) sowie eine Internetbasierende Software für die Fernsteuerung und Fernwartung, für den Endbenutzer (Infokom GmbH

Neubrandenburg) bereitzustellen. Die CWM forscht nach verschiedenen Umsetzungsmöglichkeiten eines Navigationskonzeptes unter Verwendung der digitalen Bildkorrelation und entwickelt dazu verschiedene Algorithmen sowie Softwareimplementationen.

1.1. Systemarchitektur

Um einen Roboter zu entwickeln sind außer der Erstellung von Navigationsalgorithmen eine ganze Reihe weiterer Steuer- und Bedienkonzepte wichtig, u.a. ein Benutzerinterface zur Steuerung und Verwaltung. Dieses Kapitel gibt Aufschluss über das gesamte Projekt *CORRO* und über die Anforderungen und Ziele des Projektes.

1.1.1. Übersicht über das Projekt *CORRO*

Für die Aufgaben des robotischen Sehens bei der Navigation von mobilen Service-Robotern sollen die Methoden und Algorithmen der so genannten digitalen Bildkorrelation (auch Grauwertkorrelation, DIC, DAC) nutzbar gemacht werden.

Die digitale Bildkorrelation beruht auf dem kreuzkorrelationsbasierten Vergleich lokaler Graustufen-Muster in unterschiedlichen Bildern (z.B. zu unterschiedlichen Zeitpunkten, in unterschiedlichen Lastzuständen) in interessierenden Bildregionen, wobei Musterverschiebungen oder Musterveränderungen quantitativ erfasst werden können.

Die digitale Bildkorrelation ordnet lokale Bildteilmatrizen unterschiedlicher Objektbilder durch Bestimmung der Kreuzkorrelationsfunktion und Aufsuchen ihres lokalen Maximums einander zu. Der Vergleich der lokalen Bildmuster erfolgt mittels Ähnlichkeitsbewertung durch Kreuzkorrelation, wodurch das Berechnungsergebnis bemerkenswert unempfindlich gegenüber typischen schwerwiegenden Störgrößen der klassischen Bildverarbeitung sein wird. In der Bildverarbeitung nutzt man so die Korrelationsfunktionen zur genauen Lokalisierung eines Musters (der Musterfunktion im Sinne der mathematischen Korrelation) in einem Bild. Eine typische Anwendung der Korrelation von Bildern ist beispielsweise die Auffindung und Ortsbestimmung von bestimmten Objekten oder Strukturen.

Der Grundgedanke des zu entwickelnden Selbstortungsalgorithmus für Roboter besteht darin, Bilder der Einsatzumgebung per Videokamera zu erfassen und in ihnen (eingelernte oder vorgegebene) Bildmuster wieder zu finden. Anhand der die Objektabbildung beschreibenden Parameter (Skalierung, Sichtwinkel) der in den aktuellen Umgebungsbildern enthaltenen Muster-

teilbilder können dann Winkel und Entfernung zum (bekannten) Musterstandort ermitteltet werden. Dies ist unmittelbar die Position in Polarkoordinaten relativ zum Muster tragenden Objekt, woraus mittels einfacher Transformation die absolute Position im so genannten metrischen Weltmodell des Einsatzumfelds berechnet werden kann.

Eine Besonderheit dieses Verfahrens ist, dass die zur Selbstortung herangezogenen Muster sowohl im einfachsten Fall natürliche Oberflächen von Umgebungsobjekten sein können (z.B. Putz an Wänden, Holzmaserung, Fußbodenbeläge), was den Installationsaufwand im Einsatzumfeld stark verringert, als auch separat für Ortungszwecke in das Einsatzumfeld integrierte künstliche Markierungen (Aufkleber, spezielle „Ortungsbaken“ etc.). Im letzteren Fall ist der entscheidende Vorteil gegenüber den bekannten klassischen Markierungen in der extremen Störsicherheit und der nahezu hundertprozentigen Identifikationssicherheit zu sehen, die bei Nutzung von Kreuzkorrelationsalgorithmen ohne weitere aufwändige Vorkehrungen und Bemühungen gewährleistetet werden kann.

In beiden Varianten kommt einer der wesentlichsten Vorteile der Grauwertkorrelationsanalyse zum Tragen, nämlich, dass an die wieder zu erkennenden Muster nur äußerst geringe Ansprüche gestellt werden. Es ist weder ein in irgendeiner Weise „sinntragendes“ Muster (also etwa eine alphanumerische Markierung oder besonders geeignete Symbolmarker) erforderlich, noch ist die Erkennung an geeignete Umfeldbedingungen gebunden (wie z. B. einheitliche, intensive Beleuchtung, Schutz vor Verschmutzung und Alterung etc.). Es muss lediglich eine mittels optische/elektronischer bildgebender Technik erfassbare optische Struktur vorliegen, die unverrückbar an die betrachtete Oberfläche gekoppelt ist und zu hinreichend kontrastierenden Abbildungen führt. „Hinreichend“ bedeutet hier eine Kontrastspannweite von wenigen Bit. In der Praxis haben sich sogar solche technischen Oberflächen als geeignet erwiesen, denen die menschliche visuelle Wahrnehmung ein „einheitliches“ Oberflächenerscheinungsbild zusprechen würde, wie weiße Papieroberflächen, matte metallische Oberflächen oder schwarzgestrichene Wandflächen.

Eher ungeeignet sind lediglich durchscheinende oder durchsichtige sowie vollständig spiegelnde Oberflächen.

Typische künstliche Marker könnten z. B. kontrastreich stochastisch gemusterte Schilder oder Fähnchen sein. Auch Markierungen mit durch den Menschen lesbaren Inhalten werden wieder erkannt, ohne dass herbei ein gewissermaßen „maschinelles semantisches Verständnis“ vorläge.

Ist ein Objekt in einem Bild zu suchen, so kann das Oberflächenbild des Objekts (bzw. ein geeignet definierter Ausschnitt daraus) selbst als Muster verwendet werden. Dieses Verfahren wird Mustervergleich, Template Matching der Detektionsfilter genannt. Dabei wird das Muster (Template) mit dem zu analysierenden Bild verglichen. Das Ziel des Verfahrens ist es, eine "Wahrscheinlichkeit" für das Vorhandensein des gesuchten Objektes auf einer gewissen Bildposition anzugeben. Für jede Position des Musters im Bild kann ein Korrelationskoeffizient angegeben werden, der ein Maß für die Übereinstimmung bzw. Ähnlichkeit von Muster und Bildausschnitt darstellt.

In der Lernphase muss zuerst das Objekt aufgenommen und aus der Bildaufnahme ein charakteristisches Objektmuster (i.A. ein lokaler Bildausschnitt) generiert werden, welches abschließend gespeichert wird. Das Erzeugen eines Musters erfolgt meist interaktiv, d.h. ein Mensch ist in irgendeiner Weise einflussnehmend daran beteiligt.

Bevor die Bildverarbeitung ihre eigentliche Arbeit aufnimmt, muss erst das Muster geladen und ggf. in problemspezifischer Weise vorkonditioniert, z.B. skaliert oder verzerrt, werden. Nach Aufnahme eines aktuellen Vergleichsbildes und seiner Vorverarbeitung erfolgt die Berechnung der Korrelationskoeffizienten für jede Position des Referenzmusters im Bild. Anschließend erfolgt die Auswertung der Korrelationskoeffizienten. Das mathematische Verfahren, das für die Berechnung der Korrelationskoeffizienten verwendet wird, ist in bekannter Weise die Kreuzkorrelation. Im Ortsbereich werden für lokale Bildteilmatrizen vorgegebener Größe an jeder interessierenden Position im Bild f(x,y) die Korrelationskoeffizienten PHI(x,y) berechnet. Die Kreuzkorrelation kann direkt oder entsprechend der bekannten mathematischen Äquivalenz mit Hilfe der diskreten Fourier-Transformation (FFT) durchgeführt werden und ist sehr rechenintensiv. Proportional mit der Anzahl und Größe der Muster wächst die Zeit, die zur Erkennung benötigt wird. Nach Möglichkeit sollte daher nur eine kleine, statistisch ausreichende Anzahl an kleinen Mustern verwendet werden.

Schwerpunkt des Entwicklungsprojekts sind sensorische Algorithmen zur Standortbestimmung und zur Analyse des navigatorischen Umfelds. Die Einordnung dieser Lösungen in einen Demonstratorroboter *CORRO* soll daher so erfolgen, dass für alle weiteren, angrenzenden robotischen Aufgabenstellungen möglichst einfache und damit aufwandsarme Lösungen genutzt werden können. Gegenstand der Realisierung eines Demonstrators sind nicht nur Anwendungen wie Überwachungsroboter, Besuchermanagementroboter etc., sondern es müssen auch erforderliche Sicherheitsfunktionen um Schäden für Personen, Haus- und Wildtiere oder Sachen sicher auszuschließen.

Aus den dargestellten Grundsatzüberlegungen kann geschlussfolgert werden, dass die Bildkorrelationsanalyse bei sachgemäßer Verwendung wesentlich einfacher als konventionelle Verfahren die Detektion relevanter Inhaltsänderungen und deren sichere Unterscheidung von nichtrelevanten Inhaltsänderungen ermöglicht, also zur Vermeidung von Fehlern und damit Fehlfunktionen beitragen kann. Derzeit noch gravierendster technischer Nachteil der digitalen Bildkorrelation ist der hohe Rechenzeitaufwand für entsprechende Berechnungen. Dies gilt selbst bei Nutzung der bei der CWM verfügbaren, bereits in deren Eigenentwicklung hochoptimierten Kernsoftware. Bei herkömmlichen Computern stellt dies auf derzeitigen Stand der PC-Technik ein großes Problem dar, da die Korrelation für Navigationszwecke in Echtzeit erfolgen sollte. Da noch deutlich schnellere Kernalgorithmen kaum mehr möglich erscheinen, wird die technische Problemlösung sich auf die Parallelisierung der Rechenprozesse, den Einsatz entsprechend mehrprozessorfähiger Hardware und die Optimierung der für entsprechende Aufgaben anzusetzenden Korrelationsparameter konzentrieren (z.B. sind reine Musterdetektions- und Identifizierungsaufgaben in der Regel so gestaltbar, dass die Referenzmatrizen nur unwesentlich größer als die Vergleichsmatrizen sein müssen, was die Berechnung ungemein beschleunigt).

Diesem Anliegen kommen mittelfristig die vorhersehbare Bereitstellung von Multikernprozessoren (Dual, Quad- und höher kernige CPU) oder auch die Tendenz zur parallelen grafischen Hochleistungsberechnung in Grafikprozessoren entgegen. (bereits verfügbar sind GPU mit 128 Kernen sowie eine C-Programmiersprache Nvidia CUDA und Neuentwicklungen zu OpenGL, die den Zugriff auf diese enormen Kapazitäten erlauben.) Auch arbeitet z.B. der traditionelle Grafikkartenhersteller Nvidia an massiven, auf seinen Grafikprozessorarchitekturen aufsetzenden Multiprozessorlösungen mit heute noch kaum vorstellbaren projektierten Leistungsparametern.

Trotzdem liegt hier eine wesentliche Entwicklungsaufgabe der Zukunft. Unter Nutzung, Weiterentwicklung und Problemanpassung des CWM eigenen Korrelationsstandardprogramms *VEDDAC* und des darin implementierten Korrelationskernels in Form einer wieder verwendbaren Programmbibliothek wird es möglich sein, auf Basis des implementierten Korrelationsalgorithmus entsprechende Ergebnisse ohne neue sehr umfangreiche auf die Korrelationsalgorithmen bezogenen Programmierarbeiten zu demonstrieren und die Funktionalität nachzuweisen.

Der Nachweis der entwickelten Funktionalitäten der digitalen Bildkorrelation für die Navigation von Mobilen Robotern wird über den Aufbau des Technologie-Demonstrators *CORRO* erbracht. Damit verbunden ist neben dem innovativen Kern, der Realisierung der digitalen Bildkorrelation, auch die Entwicklung aller weiteren Funktionen zur kompletten Einsatzbereit-

schaft eines mobilen Service Roboters für die Leistungserbringung einer vorgesehenen Service Funktion. Dieser Demonstrator wird prototypisch komplett entwickelt, aufgebaut und so gestaltet, dass ein autonomer Einsatz in einem komplexen Umfeld unter Bedienung und Fernwartung von einem räumlich abgesetzten Bedienerstandort praktisch demonstriert werden kann.

1.1.2. Anforderungen

Bei der Entwicklung des Roboters müssen mehrere Aspekte betrachtet werden:

- Positionserkennung und Navigation mittels Korrelation
- Erhöhung der Rechenleistung und Optimierung der Algorithmen für Anwendungsfall
- Nutzer kann den Roboter ortsunabhängig steuern
- Senden und Empfangen von Weltbildaten, Steuerdaten und Statusmeldungen über einen Internet-Service (Web-Client)
- einfache Uploadmöglichkeiten über den jeweiligen Internet-Browser
- Telematik Web Client sichert den Übertragungsweg zur Telematikplattform
- Telematikplattform speichert die Daten lokal zwischen
- Telematikplattform sendet die Daten per FTP direkt in das *CORRO*-System
- lokale Telematik API steht auf mobilen Serviceroboter zur Verfügung
- Telematik API stellt Verbindung zu Telematikplattform her
- Übertragung vollständig verschlüsselt
- Datenstruktur variabel und dynamisch

In Abbildung 1.1 ist eine schematische Übersicht über *CORRO* dargestellt. Die Navigation erfolgt über einen Abgleich der Soll-Position mit der Ist-Position und berechnet daraus eine Route zum Zielort. Diese Daten werden an die Telematik-API gesendet welche Low-Level Funktionen zur Steuerung der Plattform und Erfassung von Servicedaten zur Verfügung stellt. Die Daten werden verschlüsselt und zur späteren Weiterverwendung gespeichert.

Ein Client kann auf die Servicedaten über Internet zugreifen und den Roboter überwachen. Beispielsweise könnte somit ein Sicherheitsroboter einen Alarm auslösen, wenn er etwas Verdächtiges „entdeckt“ hat oder der Nutzer könnte die aktuelle Position des Roboters abfragen. Aufgrund der Vielzahl von denkbaren Anwendungsmöglichkeiten muss die Schnittstelle sehr

dynamisch gehalten werden um einfache Anpassungen für einen bestimmten Verwendungszweck vornehmen zu können.

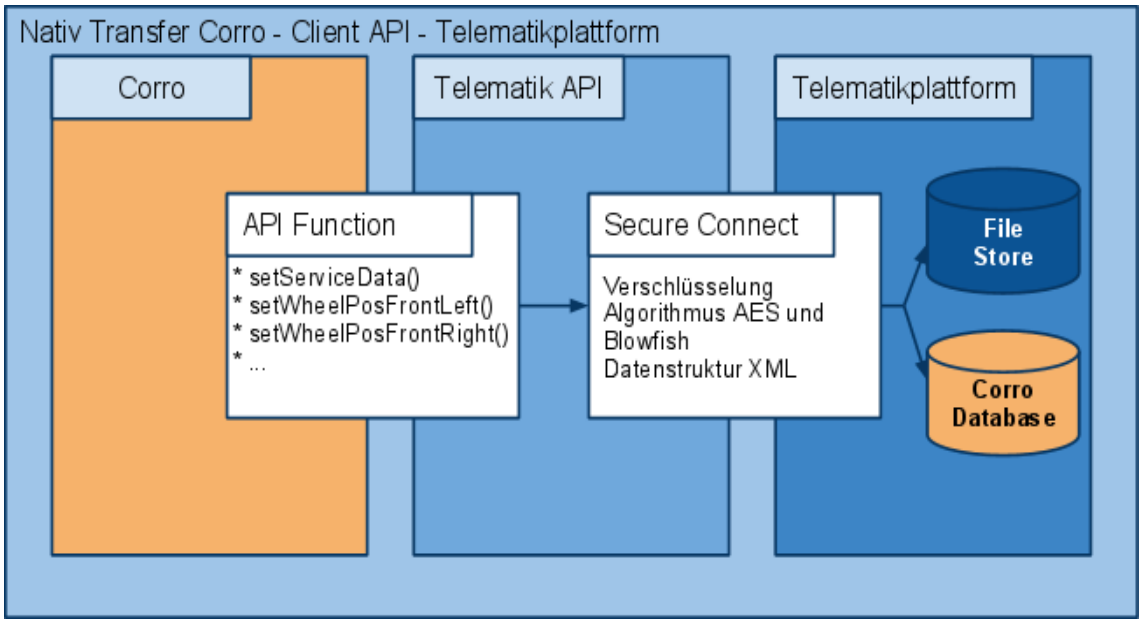


Abbildung 1.1 – Nativ Data Transfer (Quelle: Infokom GmbH Neubrandenburg)

1.1.3. Beispiel Gartenserviceroboter

Basis der Navigation ist die Beschreibung der Einsatzumfeld-Geometrie in Form einer grafischen „Landkarte“, die als Farb-Bitmap, ggf. mit mehreren Layern, im Navigations-Computer gespeichert ist. Die dafür erforderlichen Geometriedaten werden entweder durch manuelle Landvermessung oder durch Teaching-Fahrten gewonnen. Ein Beispiel solch einer Landkarte ist in Abbildung 1.2 dargestellt.

Ein interaktives Editieren der Landkarte durch den Bediener ermöglicht die flexible Gestaltung des Roboterverhaltens. Die Farbdarstellung enthält farbcodierte Informationen zu Serviceorten bzw. zu Orts- oder Umfeldabhängigen Servicefunktionen. In einem von CWM vorgeschlagenen einfachen interaktiven Nutzungskonzept, wird durch den Nutzer, von einem entfernten Standort aus, ein kleiner Initialer „Anfangsservicebereich“ (z.B. rechteckiger Bereich vor der Home-Position) festgelegt und durch schrittweise Ergänzung grüner Bereiche (d.h. für den Servicebetrieb freigegeben) eine komplexe Umfeldgeometrie iterativ erschlossen. Kritische Randbereiche wie Zäune oder Teiche können so durch schrittweise Annäherung ohne komplizierte Vermessungsarbeiten berücksichtigt werden. Diese werden als rote Markierung in die Landkarte übertragen. Dadurch ist der Roboter in der Lage, sich auf nahezu jeder Oberflächengeometrie zu bewegen.

Hindernisse wie Bäume und Sträucher müssen nicht notwendigerweise in die Landkarte integriert werden. Diese Objekte können zeitgleich zur Servicefahrt mit sensoriellen Mitteln (z.B. Abstandssensor oder Bildverarbeitungstechnisch) erkannt und umfahren werden. Dadurch ist es auch möglich, bewegliche Hindernisse wie Personen oder Tiere zu erkennen und damit eine Kollision zu verhindern.

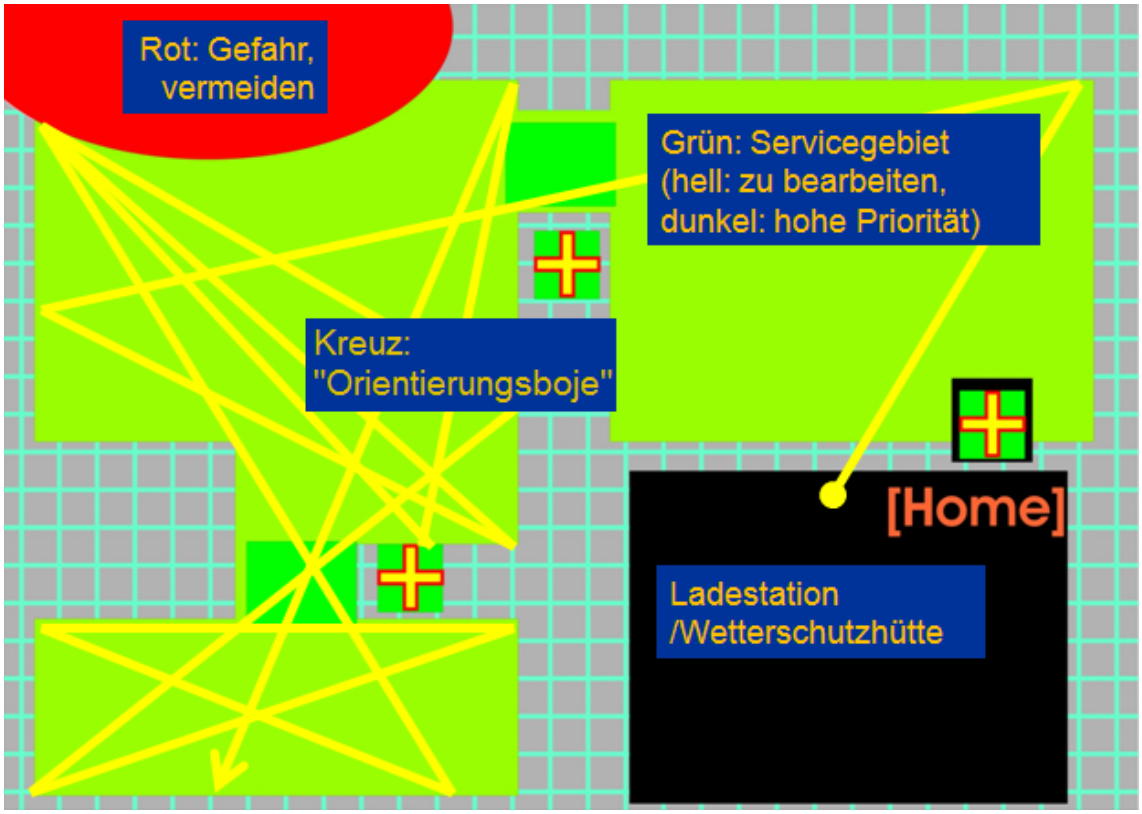


Abbildung 1.2 – Umfeldbeschreibung eines Gartenserviceroboters als farbcodiertes Bitmap

Gefahrene Wege werden in diesem Konzept als gelbe Pixel erfasst. Die Servicelogik wählt für die Planung der Route von einer bestimmten Startposition jeweils den Weg aus, der zur größten Anzahl „eingesammelter“ grüner Pixel führt. Dadurch wird nach und nach die gesamte Fläche befahren. Nachwachsendes Gras wird simuliert, indem nach einem Zufallsalgorithmus die gelben Pixel in ihrem Grünanteil inkrementiert werden. Nach einer gewissen „Nachwachszeit“ werden also früher gemähte Bereiche wieder für die Routenplanung „attraktiv“ und daher wieder für die Servicefunktion berücksichtigt.

1.2. Hauptziele der Arbeit

Diese Arbeit befasst sich mit der Erstellung von Algorithmen und Programmen auf Basis der Grauwertkorrelation, zur Selbstortung (Bestimmung des aktuellen Standortes im Einsatzumfeld) einer mobilen Plattform. Weiter werden mögliche Optimierungen und Verbesserungen von den bereits existierenden Korrelationsalgorithmen in Bezug auf die Positionsbestimmung dargestellt. Dazu gehören die Berechnung Trigonometrischer Daten (Rotationswinkel, Verschiebungsvektor), Verwendung eines Offsets zur Verringerung benötigter Rechenzeit und Korrekturverfahren für Verschiebungsmessfelder.

Auf dieses Ziel bezogen werden in Kapitel 2 einige Grundlagen erläutert. Theoretisches Basiswissen um die herkömmliche Navigation von Robotern, das Prinzip der Grauwertkorrelation, Programmbibliotheken und das sogenannte KOBOLD-Prinzip werden vorgestellt.

Darauffolgend wird in Kapitel 3 der erste Experimentträger mit der verwendeten Hard- und Software vorgestellt. Außerdem wird der Korrelationsalgorithmus zur Umsetzung des KOBOLD-Prinzips sukzessive aufgebaut und auf Details der Implementierung eines dafür entwickelten Programmes eingegangen. Weiter wird die Möglichkeit betrachtet, Webcams statt CCD-Kameras für die Bildaufnahme zu verwenden.

In Kapitel 4 wird das KOBOLD-Prinzip mit der herkömmlichen Odometrie[1] anhand neuer Referenzdaten verglichen und verifiziert.

Kapitel 5 schließt die Arbeit mit einer Zusammenfassung und einen Ausblick ab.

[1] Odometrie oder auch Hodometrie (von griech. hodós, „Weg“ und métron, „Maß“ - also: „Wegmessung“) ist die Wissenschaft von der Positionsbestimmung eines mobilen Systems anhand der Daten seines Vortriebsystems.

2. Grundlagen

2.1. Navigationsprinzipien mobiler Roboter

Navigation bedeutet, dass sich ein Objekt von einem Ausgangspunkt zu einem Zielpunkt bewegt. Damit dies möglich ist müssen drei Fragen beantwortet werden.

- **Wo bin Ich?** – Der aktuelle Standort im Einsatzumfeld (z.B. ebene Koordinaten) muss bestimmt werden.
- **Wo will ich hin?** – Berechnung eines Weges abhängig von der aktuellen Lageinformation und der Zielvorgabe in komplexen Umgebungen.
- **Bin ich da?** – Die aktuelle Lageinformation entspricht der Zielvorgabe.

Um diese Fragen zu beantworten, existieren mehrere Methoden. Eine Auswahl der am häufigsten verwendeten Navigationsprinzipien und -anordnungen wird im Folgenden kurz dargestellt. [Gut99]

- **Sonar- und Laserscanner:** Mit Sonar- und Laserscanner werden Signale im Ultraschallbereich bzw. Lichtwellen ausgesendet. Diese Signale werden von den Objekten auf denen sie treffen reflektiert und kehren zum Sensor zurück. Aus der Zeit die dieses Signal braucht bis es wieder am Ausgangsort ist, wird die Entfernung zum Objekt berechnet. Der Vorteil ist, dass diese Sensoren sehr günstig sind aber diese Technik besitzt auch mehrere Nachteile. Das ausgesendete Signal kann von sehr glatten Oberflächen abgelenkt werden und wird erst reflektiert bis es auf ein weiteres Objekt trifft (specular reflection). Es wird eine falsche Entfernung zum Objekt berechnet, d.h. Hindernisse und Begrenzungen werden nicht erkannt bzw. falsch lokalisiert. Ein ähnliches Problem tritt auf, wenn mehrere Sensoren verwendet werden. Das ausgesendete Signal wird von einem Objekt reflektiert aber trifft auf einen anderen Sensor welcher es als sein eigenes Signal fehlerkennt (cross-talk). Dadurch wird ebenfalls eine falsche Entfernung zum Objekt berechnet.
- **Videokamera:** Mit Videokameras werden Farbbilder aufgenommen. Aus diesen Bildern können ohne weiteres (z.B. stereoskopische Prinzipien wie „depth from motion“ etc.) keine Entfernungen bestimmt werden aber es lassen sich Objekte anhand ihrer Form und Farbe bestimmen. Dabei entsteht eine große Datenmenge an Bildern, sodass

ein typisches Merkmal solcher Lösungen eine sehr zeitaufwendige und algorithmisch schwierige Auswertung ist (und das grundsätzlich auch für die in dieser Arbeit genutzten Konzepte gilt) eine sehr zeitaufwendige und algorithmisch schwierige Auswertung ist.

- **Odometrie:** Mit der Odometrie wird der zurückgelegte Weg anhand der Analyse der Bewegungen des Antriebssystems (z.B. Umdrehung der Räder) berechnet. Auf kurzen Entfernungen ist dieses Prinzip sehr genau aber diese Genauigkeit sinkt mit zunehmender Fortbewegungsdistanz. Ein großes Problem ist dass die genaue Ausrichtung der Räder, welche die Richtung des Roboters vorgibt, nicht hinreichend exakt bestimmt werden kann. Weitere Probleme resultieren aus ungenau berücksichtigten, teilweise unbekannten Parametern der Radgeometrie, Bodenbeschaffenheit und Fahrzeuggewicht. Diese Fehler summieren sich mit jedem Messschritt was zu einer immer größer werdenden Abweichung (Regeldifferenz) der berechneten von der tatsächlichen Position des Fahrzeugs führt.

2.2. Korrelation

Die Korrelation ist ein statistisches Verfahren, das den Zusammenhang zwischen zwei Merkmalen ermittelt. Die Stärke des Zusammenhangs wird mittels des Korrelationskoeffizienten „r“ (weiter vorn „Phi“) (Formel 2.1) bestimmt, dessen Wertebereich von -1 bis +1 reicht. Ein Koeffizient von 0 beschreibt, dass keinerlei lineare Abhängigkeit zwischen den Merkmalen besteht. Im Falle der exakten linearen Abhängigkeit ist der Betrag des Koeffizienten gleich +1. [Haa08]

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Formel 2.1 – Empirischer Korrelationskoeffizient „r“ nach Bravaise-Pearson[2]

Das Quadrat des Koeffizienten ist das Bestimmtheitsmaß „B“. Dieses Maß beschreibt mit welcher Wahrscheinlichkeit ein Merkmal von dem anderen abhängig ist. Beispielsweise ist ein Merkmal A bei einem Bestimmtheitsmaß von 0,56 zu 56% von dem Merkmal B abhängig.

[2] Benannt nach Auguste Bravais (* 23. August 1811 in Annonay, Frankreich; † 30. März 1863 in Le Chesnay; französischer Physiker) und Karl Pearson (* 27. März 1857 in London; † 27. April 1936 in Coldharbour, Surrey; britischer Mathematiker)

2.2.1. Prinzip der Digitalen Bildkorrelation

Die digitale Bildkorrelation (engl.: digital image correlation, Abk.: DIC) nutzt die Korrelation für die berührungslose Verformungsanalyse von Werkstoffen. Industriekameras zeichnen während der Verformung Bilder auf. Dabei hinterlässt jede Oberfläche eines aufgenommenen Objektes ein eindeutiges Pixelmuster welches in einem zweiten Bild, unter Verwendung der Korrelation, gesucht wird. Es wird also „erkannt“ in wieweit sich das digitale Bild eines Objektes an lokalen virtuellen „Messpunkten“ verschoben und verformt hat. Dieses Verfahren findet u.a. Einsatz in der Werkstoffprüfung, Personenerkennung, Sicherheitsidentifikation, Katastrophenfrühwarnung und Mikrosystemtechnik. (Abbildung 2.1) [CWM10]

Abbildung 2.1 – L.: Nachweis der Identität eines Gemäldes. R.: Rissbildung und Scherverformung an einem QFN-Bauelement nach 1000 Temperatur-zyklen bei 130°C. [CWM10]

Beim Grauwertkorrelationsverfahren (engl.: Deformation Analysis by means of correlation, Abk.: DAC) werden in einem Graustufen-Referenzbild mehrere Messpunkte definiert. Die Grauwertmatrix der näheren Umgebung dieser Messpunkte wird über das Korrelationsverfahren in einem größeren zu analysierenden Bildbereiches (Vergleichsmatrix) eines weiteren Bildes (Messbild) gesucht. Die Position an der der größte Korrelationskoeffizient berechnet wurde, stimmt am besten mit der Referenzmatrix überein. Aus der Differenz der Koordinaten der gefundenen Struktur und den Koordinaten der Referenzmatrix ergibt sich der Verschiebungsvektor. (Abbildung 2.2)

Mit diesem Verfahren kann die Verschiebung pixelgenau bestimmt werden. Diese Genauigkeit wird, unter Verwendung eines Subpixel-Algorithmus, erhöht. Damit wird der Nachkommaanteil der berechneten Verschiebungen näherungsweise bestimmt. Typische erreichte Genauigkeiten liegen in der Größenordnung zwischen 1/10 und 1/100 Pixel.

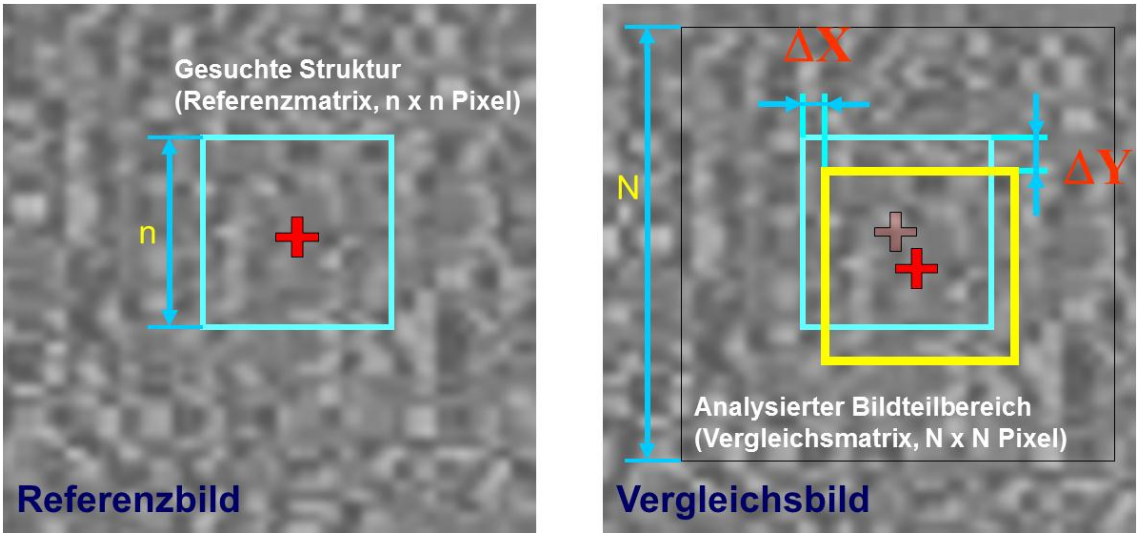


Abbildung 2.2 – Darstellung des Grauwertkorellationsverfahrens

Weiterführende Informationen und mathematische Beschreibung zur Grauwertkorrelation sind in der Diplomarbeit [Haa09b] verfügbar.

2.2.2. Korrelationsbibliotheken

Das Grauwertkorrelationsverfahren stellt für die CWM eine Schlüsseltechnologie für ihre Produkte und Dienstleistungen dar. Deshalb wurde ein Framework (Abbildung 2.3) entwickelt, welches Datenstrukturen, Funktionen und Konstanten bereitstellt die für die Ausführung der Korrelationsberechnungen nötig sind. Dieses Framework unterteilt sich in die Programmbibliotheken:

- `CorrKrnl.dll`: Kernfunktionen und Datenstrukturen für die Berechnung von Verschiebungsvektoren.
- `CorrIO.dll`: Kapselt Kern-Datenstrukturen und stellt Ein-Ausgabefunktionen zur Verarbeitung von Device Independent Bitmaps (DIB) zu Verfügung. Weiterhin gibt es Funktionen mit welchen Korrelationsberechnungen gespeichert bzw. geladen werden und auch in andere Formate (z.B. Excel-kompatible Ergebnislisten) exportiert werden können.

Durch das Framework wird erreicht, dass Programme schneller entwickelt werden können, da eine separate Programmierung des Korrelationsalgorithmus nicht nötig ist. Änderungen am Korrelationsalgorithmus, wie Geschwindigkeits-Optimierungen und beheben von Fehlern, wirken sich auf alle Programme die das Framework verwenden aus. Dadurch wird ein Patchen der Programme vereinfacht indem die Bibliotheken nur „ausgetauscht“ werden müssen. Ein Neu-

kompilieren der nutzenden Programme ist nicht nötig. Dadurch ist eine hohe Wiederverwendbarkeit, Änderbarkeit und Funktionalitätssicherheit gewährleistet.

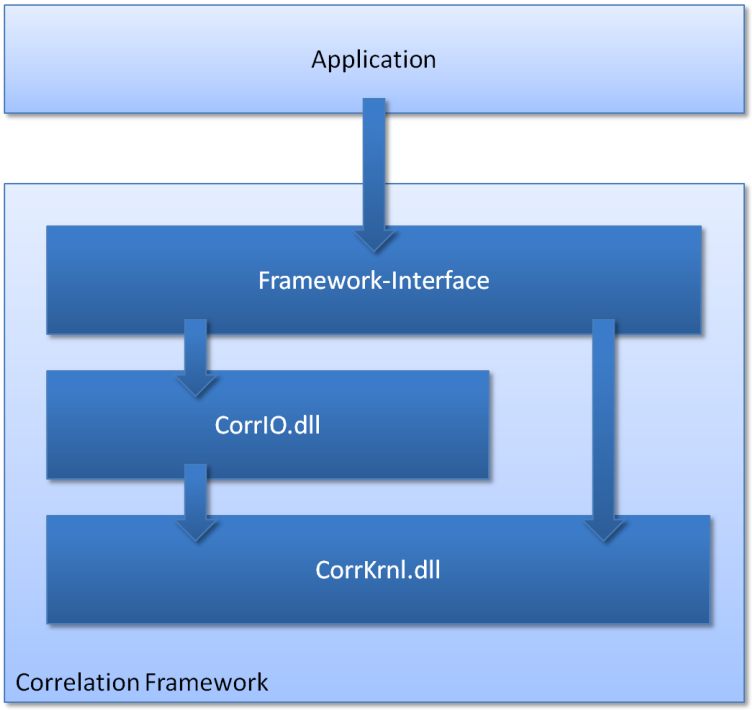


Abbildung 2.3 – Schematische Darstellung des Korrelationsframeworks

In Listing 2.1 sind die Header der wichtigsten Funktionen des Frameworks dargestellt. Mit `cioLoadCalc()` wird eine Berechnungsstruktur mit Korrelationsparametern wie Referenz- und Messfeldgröße, Koordinaten der Messpunkte und des zu verwendenden Korrelationsalgorithmus initialisiert. Das Referenzbild und Messbild wird mit `cioLoadDib()` geladen. Es wird jeweils ein Zeiger auf die Bitmapdaten erstellt, welche der Funktion `cioGetImageData()` übergeben werden, um einen Zeiger auf die Pixeldaten zu erhalten. Mit den so gesammelten Informationen über die Bilder, Messpunkte und Parameter wird die Korrelationsberechnung gestartet, indem die Funktion `CalcCorr()` aufgerufen wird. Diese Funktion berechnet die Verschiebungsvektoren jedes Messpunktes inklusive den Subpixelanteil. Dafür werden, je nach verfügbaren Prozessorkernen, bis zu 8 Threads erzeugt welche die Berechnungen Echtparallel durchführen und damit insgesamt bis zu 8-mal schneller gegenüber einer Singlethread Variante werden.

```
// Laden einer Messpunktdatei und initialisieren der Calc-Struktur
CIO_API int cioLoadCalc(PCIO_CALC pCalc, PCSTR pcPath,
   unsigned uFlags = CIO_CF_ALL);

// Laden eines Device Independent Bitmap (DIB)
CIO_API int cioLoadDib(PCIO_DIB pDib, PCSTR pcPath,
   unsigned uFlags = CIO_DF_ALL);

// Pixeldaten eines Referenz- und Messbitmap laden
CIO_API int cioGetImageData(PCK_IMAGE_DATA pImg, PCCIO_DIB pcRef,
   PCCIO_DIB pcMeas);

// Hauptfunktion zur Ausführung der Korrelationsberechnung
CK_API int CalcCorr(
   PCK_IMAGE_DATA pImageData,       // Bilddaten
   PCK_CORR_PARAM pCorrParam,       // Korrelationsparameter
   PCK_CORR_DATA pCorrData,         // Messpunkte
   CK_FCT_CALC_PROGRESS pfCalcProgress = 0,     // Callbackfunktion
   LPARAM lParam = 0          // Benutzerparameter
);
```

Listing 2.1 – Grundlegende API-Funktionen des Korrelationsframeworks

2.3. Das KOBOLD-Prinzip

Das KOBOLD[3]-Prinzip ist ein von der CWM vorgeschlagenes Konzept zur automatischen Navigation autonomer mobiler Roboterplattformen. Im Kern handelt es sich dabei um die hochexakte Messung der Fahrwege auf Basis der Bilderfassung und einer Wegmessung aufgrund korrelationsgestützter Verschiebungsanalyse. Die Auswertung der Daten erfolgt nach dem Prinzip der Odometrie. Dadurch wird eine hohe Genauigkeit bei der Positionsmessung erreicht, weil die in Abschnitt 2.1 bereits benannten fehlerträchtigen Einflussfaktoren herkömmlicher Odometrie vermieden werden. Diese Genauigkeit wird zudem erhöht, indem vorhandene Muster der realen Umgebung oder dort eingebrachte künstliche Grauwertpattern (Markierungen, „Bojen“) mittels einer Künstlichen Intelligenz erkannt und insoweit verstanden werden, dass daraus die exakte Position des Roboters berechnet wird. Ein sonstiges maschinelles Verstehen des Umfelds ist damit nicht verbunden obwohl ein solches grundsätzlich auch auf Korrelationsbasis unterstützt werden könnte.

Das KOBOLD-Prinzip ist in Abbildung 2.4 näher veranschaulicht. Ein Roboter befindet sich an einem Startpunkt mit bekannten Koordinaten. Durch Korrelationsodometrie werden die neuen Koordinaten nach fahren des Roboters bestimmt. Wenn bei der Navigation über ein Bodenmuster gefahren wird, werden die Absolutkoordinaten exakt festgestellt und die Bahnkurve

[3] KOBOLD – **K**orrelations**O**dometrische **BO**denmustergestützte **L**age**D**etektion

entsprechend korrigiert. Die Koordinaten der Bodenmuster sind zuvor im Rahmen einer Teaching-Fahrt erfasst und eingelernt worden.

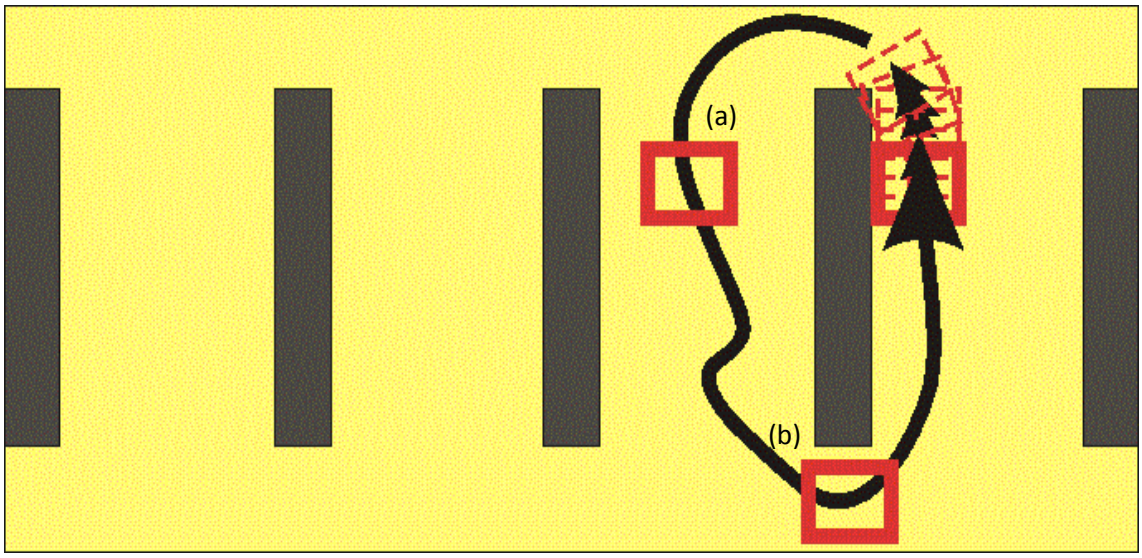


Abbildung 2.4 – Ergänzung des KOBOLD-Prinzips durch absolute Positionsbestimmung mit eingelerntem Bodenmuster (a), (b)

3. Umsetzung des KOBOLD-Prinzips

Um das KOBOLD-Prinzip umzusetzen, wurde ein korrelationsbasierter Algorithmus entwickelt, der stetig um ergänzende Funktionen und Optimierungen erweitert wurde. In diesem Abschnitt werden die einzelnen Schritte vorgestellt.

3.1. Aufbau eines Experimentträgers (ET) zur Simulation einer Mobilen Plattform

Es wurde auf einem Palettenwagen ein in Abbildung 3.1 dargestellter Experimentträger gebaut, mit welchen Bilder eines zurückgelegten Weges aufgenommen werden können. Dafür wurde am hinteren Ende der Palette eine CCD-Kamera befestigt welche auf den Boden gerichtet ist. Mit einem handelsüblichen Laptop mit installierter Software *VEDDAC Cam* wird die Kamera gesteuert. Die so aufgenommenen und gespeicherten Bilder wurden dazu verwendet um Algorithmen und Formeln zur Rückverfolgung des gefahrenen Weges zu entwickeln.

Abbildung 3.1 – Experimentträger mit CCD-Kamera *AVT Guppy 080-F*

3.1.1. Verwendete Kameras

Die Testfahrten wurden mit unterschiedlichen Kameras, Wegen und Untergründen unternommen, um die Korrelationsfähigkeit der aufgenommen Bilder mit verschiedenen Kameras (Tabelle 3.1) zu untersuchen.

Hersteller	Typ	Auflösung	fps	Shutter
AVT	Guppy 080-F	1034x768 pxl	30	54µs bis 67s, Auto
AVT	Marlin F-201B	1628x1236 pxl	15	20µs bis 67s
Hercules	Deluxe Optical Glass	VGA	30	Unbekannt, Konstant

Tabelle 3.1 – Verwendete Kameras

Bei den Testfahrten stellte sich heraus, dass eine Kamera, welche viele Bilder pro Sekunde erstellt und mit geringer Belichtungsdauer (Shutterzeit) am besten für die Korrelation geeignet ist. Viele Bilder bedeuten, dass eine starke Überlappung der Bilder vorhanden ist und damit der Weg, welchen der Roboter zwischen zwei Bildern zurückgelegt hat, sehr kurz ist. Dadurch ist die Länge der Verschiebungsvektoren geringer und somit kann ein kleineres Messfeld zur Korrelationsberechnung verwendet werden was die Rechenzeit erheblich verkürzt. Eine kürzere Shutterzeit hat zufolge, dass die Bilder bei größeren Geschwindigkeiten schärfer werden und somit Grauwertmuster besser wieder erkannt werden. Dadurch erhöht sich der Korrelationskoeffizient und die Verschiebungsvektoren werden mit besserer Genauigkeit ermittelt.

3.1.2. Webcam

Im Hinblick auf die Verringerung der Kosten des Endproduktes, wurde überprüft ob die Industriekameras durch Webcams ersetzt werden können. Dafür wurde ein Programm entwickelt, welche den Videodatenstrom der Webcams in Grauwertbitmaps umwandelt und speichert. Anforderungen an das Programm sind, dass mindesten zwei Webcams parallel betrieben und jede handelsübliche Webcam betrieben werden kann.

Treiber für Webcams basieren auf dem Windows Driver Model für Multimedia (WDM) von Microsoft. Durch diesen Standard wird ermöglicht, dass eine einheitliche Schnittstelle vorhanden ist. Dadurch ist es möglich die Treiber zu verwenden, ohne dass eine genaue Architektur des Treibers bekannt sein muss.

Der Zugriff auf WDM-Treiber erfolgt mit dem Programmierinterface *DirectShow*. *DirectShow* ist eine von Microsoft entwickelte Architektur um Multimedia Daten auf Windows Plattformen zu verwenden. Diese Architektur basiert auf dem *Component Object Model (COM)*, welche sprachunabhängig, objektorientiert und plattformunabhängig (innerhalb von Microsoft Plattformen) ist. Abbildung 3.2 zeigt eine Übersicht über die Architektur der *DirectShow* Lösung. [MSD10]

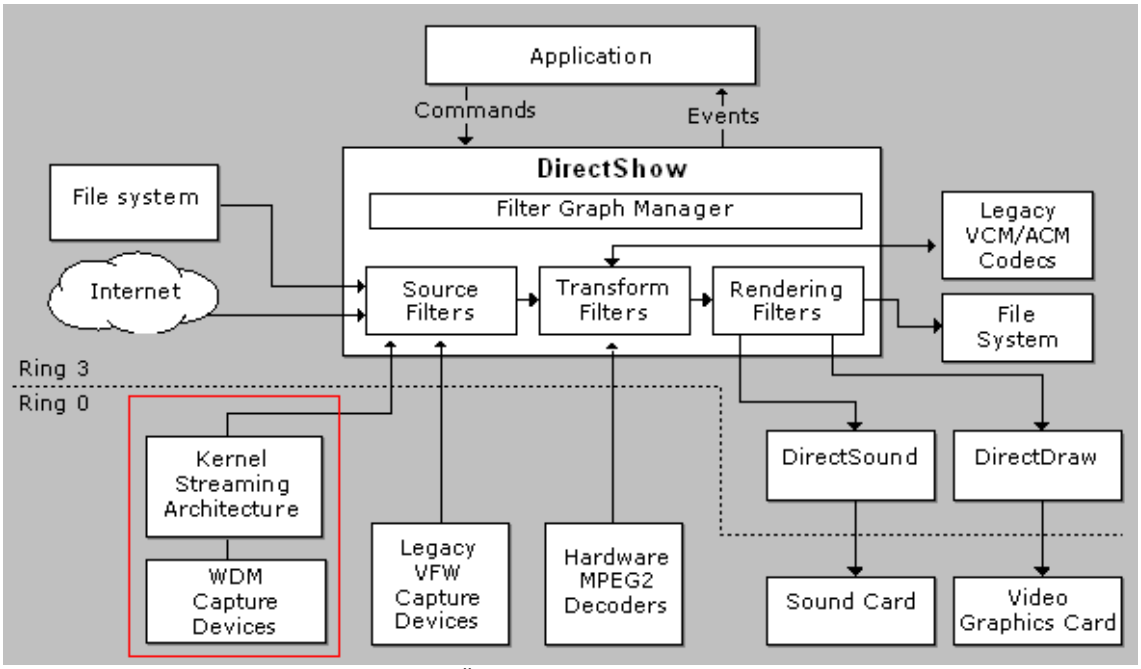


Abbildung 3.2 – Übersicht über DirectShow Architektur

DirectShow stellt eine Reihe von Filtern (COM-Objekte) bereit, welche mit einer Multimedia-Quelle (Treiber, Datei, Datenstrom) über In-/Output Pins „verkettet“ werden. Diese Kette (Filtergraph) manipuliert mit Transform Filter und Rendering Filter die Quelldaten. Abbildung 3.3 zeigt einen *Filtergraph* für ein USB-Videogerät (Webcam-Treiber). Der *Capture-Pin* des Videogerätes ist mit einem *AVI-Decompressor* (Transform Filter) verbunden, welcher den komprimierten MPEG-Video-Datenstrom dekomprimiert. Der *Sample Grabber* (Transform Filter) liest die einzelnen Bilder des Videos und wandelt diese in Grauwertbitmaps um und speichert die Bilder. Da kein Rendering Filter benötigt wird, wird stattdessen ein *Null Renderer* verwendet, welcher den Filtergraphen abschließt aber ohne Daten zu rendern. An Stelle des *Null Renderes* könnte der *Sample Grabber* auch mit einen *Video Renderer* verbunden werden. Dies würde ermöglichen, dass ein aufgenommenes Bild auch auf dem Monitor angezeigt wird. Allerdings kostet dies einen großen Teil der Rechenleistung. Deshalb wurde diese Funktion in den *Sample Grabber* implementiert, wodurch bedeutend weniger Rechenleistung benötigt wird.

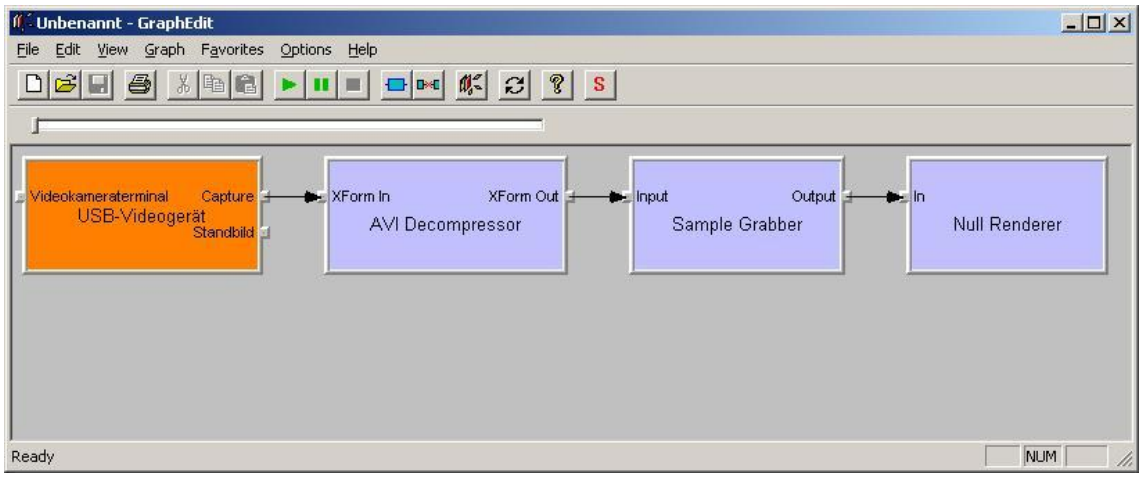


Abbildung 3.3 – Webcam Filtergraph zur Aufnahme von Grauwertbitmaps

Basierend auf diesen Graphen wurde ein Programm entwickelt[4], welches den Videodatenstrom in Grauwertbitmaps umwandelt. Es werden zunächst alle verfügbaren Webcams initialisiert indem für jede ein *Filtergraph* erstellt wird. Diese Initialisierung wird mit der Methode `BuildGraph()` von der Klasse `CVidCapture` vorgenommen. Die Klasse repräsentiert somit jeweils eine Webcam. Die Klasse `CSampleGrabberCB` implementiert das *COM-Interface* `ISampleGrabberCB` für den *Sample Grabber* Filter, welche benachrichtigt wird, wenn ein neues *Sample*, also Bild, fertig gestellt wurde. Dafür wird die Methode `BufferCB()` aufgerufen. Diese Methode erhält die Pixeldaten des *Samples*, welche von der Methode `WriteBitmap()` in Grauwerte umgerechnet und als Bitmap gespeichert werden. Dieser Vorgang läuft in einer asynchronen Schleife, solange die Webcam *Samples* liefert und die Klasse existiert. Der Nutzer kann deshalb mit der Methode `StartWriteBitmap()` bzw. `StopWriteBitmap()` das Speichern der Bilder steuern. Abbrechen, also ein Zerstören des *Filtergraphen*, ist nur durch beenden des Programmes möglich. Abbildung 3.4 zeigt ein grobes Sequenzdiagramm. In Anhang E.1 ist ein Klassendiagramm, Dokumentationen und der Quellcode für die wichtigsten Funktionen verfügbar.

[4] Entwicklung in Microsoft Visual Studio 2008 Professional; C++ unmanaged für Windows XP x86 oder neuer; Generische Zeichensatzverarbeitung („TStrings“)

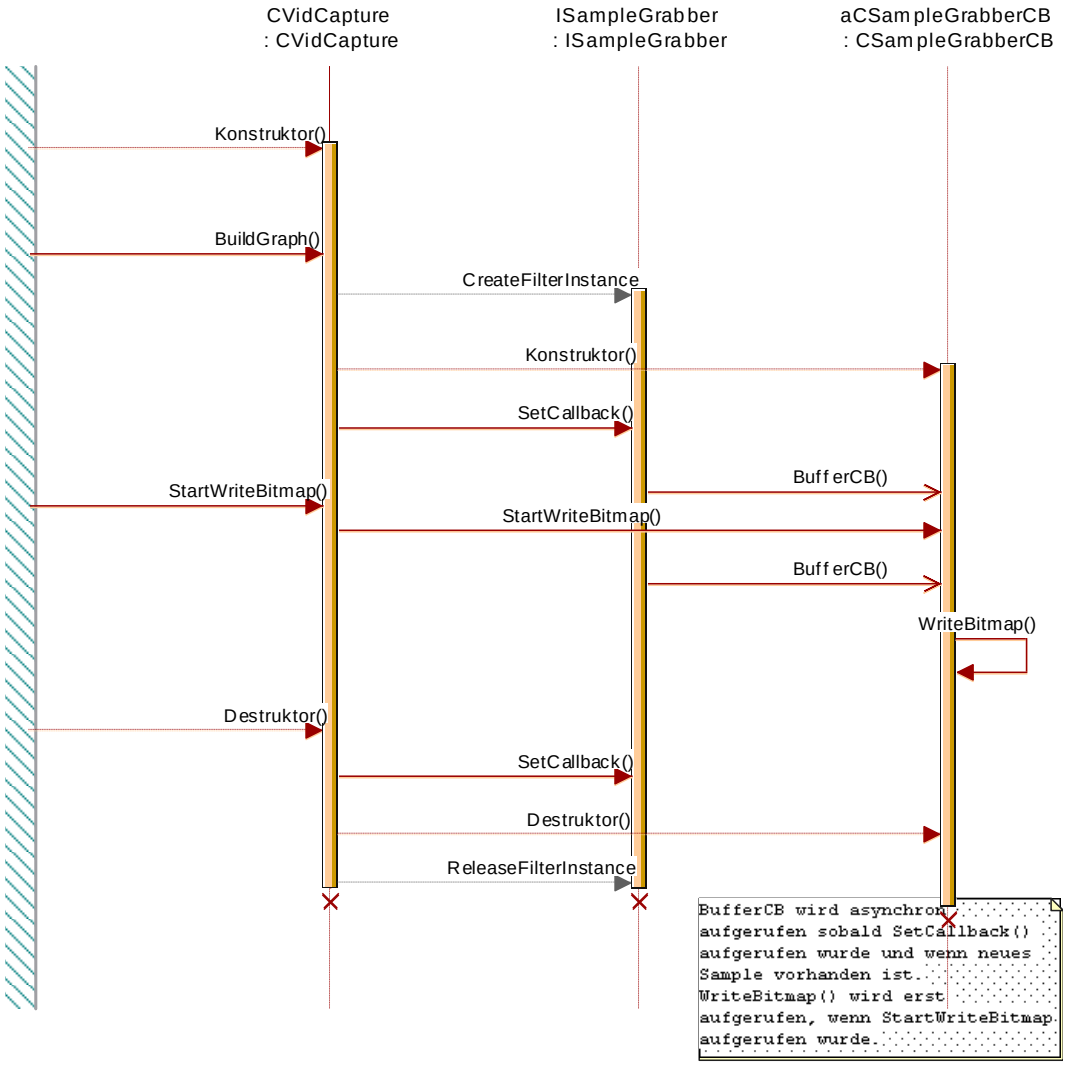


Abbildung 3.4 – Webcam Sequenzdiagramm

Mit diesem Programm wurden einige Testfahrten unternommen. Dabei stellte sich heraus, dass die Shutterzeiten der Webcams zu groß sind. Das heißt, dass die aufgenommenen Bilder, selbst bei geringen Fahrgeschwindigkeiten stark verschwommen sind und nicht mehr korreliert werden können. Ein weiteres Problem ist, dass beim zeitlich parallelen Betrieb von mehreren Webcams die Busbreite der USB-Schnittstelle zu gering ist. Dadurch reduzieren sich die Anzahl der aufgenommen Bilder pro Sekunde. Dies hat zufolge, dass Verschiebungsvektoren bei der Korrelation zu groß werden und ein immens höherer Rechenaufwand erforderlich ist. Aus diesen Gründen ist der Einsatz von Webcams – zumindest für eine Nutzung nach dem grundlegenden Kobold-Konzept – derzeit nicht möglich. Denkbare Alternativen bestünden möglicherweise in einer Verschiebungsmessung im bewegungsunscharfen Einzelbild mittels Texturanalyse oder in einer stroboskopischen Bodenbeleuchtung und einer bodenbezogenen Geschwindigkeitsmessung durch Periodenbestimmung mittels Autokorrelationsanalyse im Einzelbild. Diese Lösungen sind jedoch nicht Gegenstand der vorliegenden Aufgabenstellung.

3.2. Programm *CorroPos*

Der Korrelationsalgorithmus ist die Basis für die Navigation. Um diesen Algorithmus zu entwickeln, wurden mit dem in Kapitel 3.1 beschriebenen Experimentträger Bilder auf verschiedenen Untergründen und Wegen erstellt worden, um die Korrelationsfähigkeit und Zurückverfolgbarkeit des Weges zu prüfen. Die Bilder wurden versuchsweise zunächst mit *VEDDAC* korreliert, da dieses Programm bereits vorhanden und getestet ist. Aus den gewonnen Verschiebungsvektoren und Rotationswinkeln kann mittels Formel 3.1 die aktuellen Koordinatenpositionen der mobilen Plattform im Weltkoordinatensystem berechnet werden. [Lau10]

$$P_{x_{n+1}} = \sqrt{\Delta x_n^2 + \Delta y_n^2} * \sum_{i=0}^{n} \cos \varphi_i + P_{x_n}$$

$$P_{y_{n+1}} = \sqrt{\Delta x_n^2 + \Delta y_n^2} * \sum_{i=0}^{n} \sin \varphi_i + P_{y_n}$$

(P_n = n-ter Punkt im Weltkoordinatensystem; Δx_n, Δy_n = n-te Verschiebung; φ = lokaler Verschiebungswinkel)
Formel 3.1 – Rekursionsformel für Berechnung der Weltkoordinaten aus Lokalkoordinaten

Auf Basis dieser Formel wurde ein in Abbildung 3.5 dargestelltes Programm (*CorroPos*) entwickelt[5], welches eine Positionsbestimmung zeitgleich zur Fahrt der Plattform und in Echtzeit durchführt.

Die Grundfunktionalitäten des Programmes sind:

- Überwachen eines angegebenen Ordners auf neue Bilder
- Positionserkennung mittels des in Kapitel 2.2.2 vorgestellten, Korrelationsframeworks
- Sicherung der berechneten Koordinaten und Zwischenergebnisse in eine Logdatei und Speicherung der Verschiebungsvektoren im Standard *VEDDAC*-Format
- Ausgabe der aktuellen Absolutkoordinaten und des aktuellen Rotationswinkels für eine Weiterverarbeitung durch andere funktionsrelevante Robotersoftware
- Optimierung der Rechengeschwindigkeit
- Hohe Wiederverwendbarkeit des Korrelationsalgorithmus

[5] Entwicklung in Microsoft Visual Studio 2008 Professional; C++ unmanaged für Windows XP x86 oder neuer; Multibyte Zeichensatz

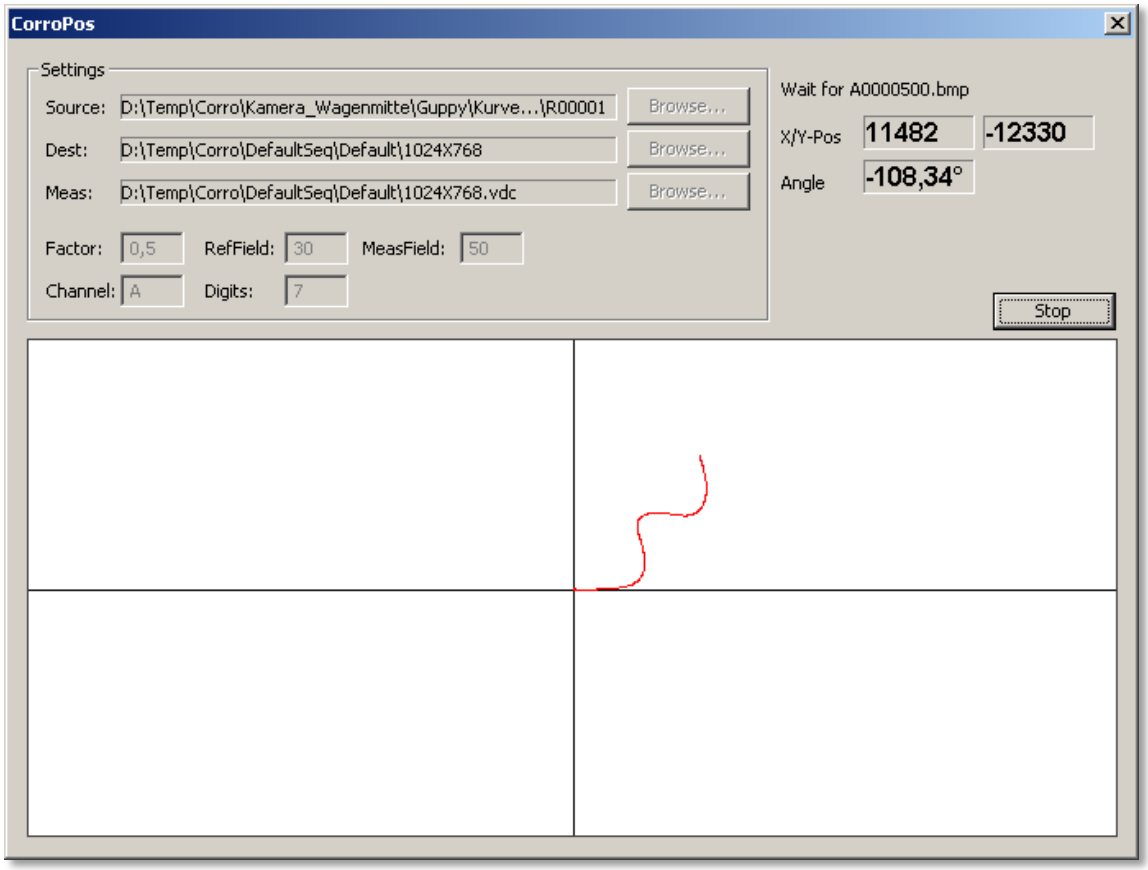


Abbildung 3.5 – Benutzeroberfläche des Programms zur Selbstortung

Die Benutzeroberfläche ist als Testumgebung für den Korrelationsalgorithmus zu betrachten und auf die wichtigsten Ein-/Ausgabefunktion beschränkt.

Über die „Browse-Buttons“ können ein Bildüberwachungsordner, ein Ordner für die Sicherung der Rechenergebnisse und eine in *VEDDAC* erstellte Messpunktdatei angegeben werden. Weiterhin können Werte für den Offsetfaktor, Referenz- und Messfeldgröße und Dateimaske der Bilder verändert werden. Mit dem „Start/Stop-Button“ werden die Korrelationsberechnungen gestartet bzw. angehalten. Im rechten Abschnitt des Programmes werden das aktuell berechnete Bild, die aktuelle Position und der Rotationswinkel der Plattform angezeigt. Der untere Teil der Oberfläche stellt den zurückgelegten Weg des Roboters, im Weltkoordinatensystem, graphisch dar. Im Anhang E.2 sind die Klassendiagramme, Dokumentationen und Quellcode für den Korrelationsalgorithmus verfügbar.

3.3. Korrelationsalgorithmus für die Weganalyse

Der Korrelationsalgorithmus ist in der Klasse `CCalcPos` gekapselt und wird von der Klasse `CMainCtrl` gehalten, welche die Steuerungsschnittstelle zwischen dem Nutzer und der Programmlogik darstellt (Controller).

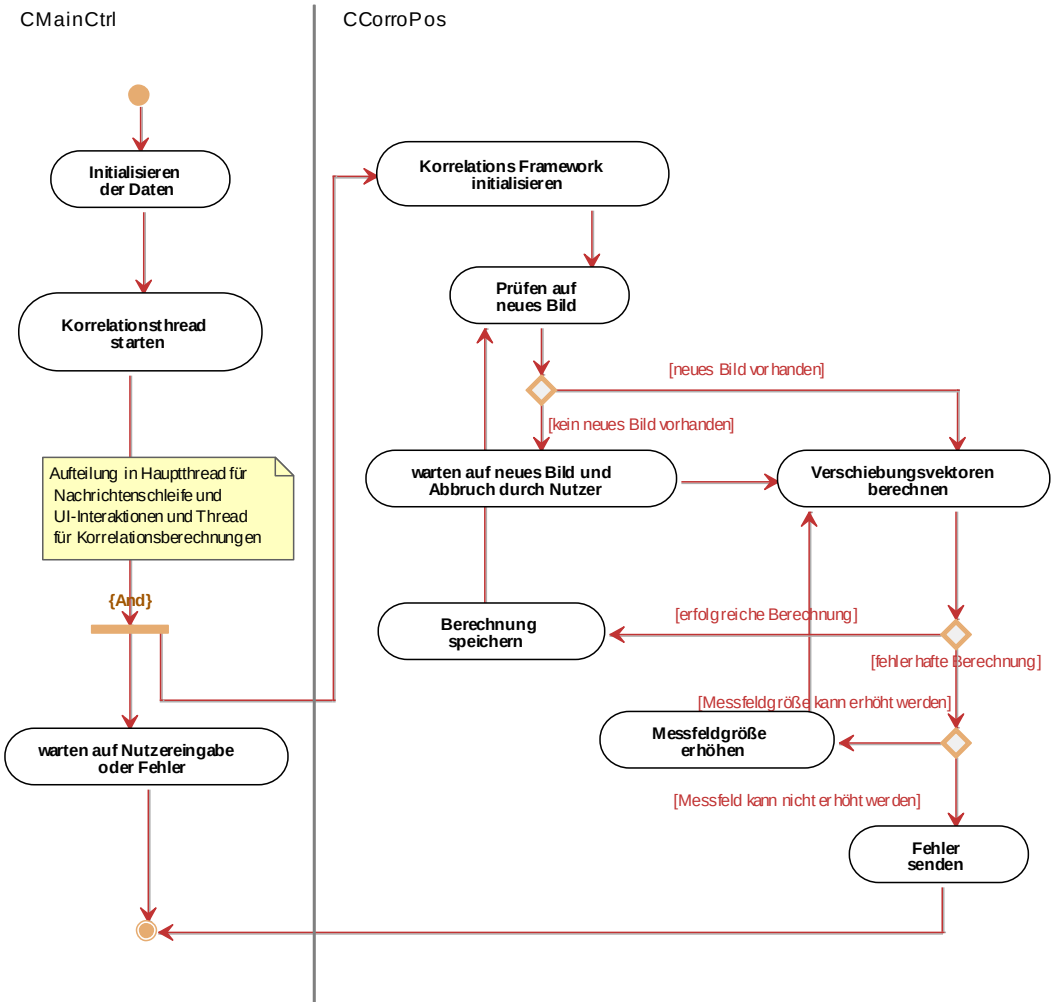


Abbildung 3.6 – Aktivitätsdiagramm des Basis Korrelationsalgorithmus

In Abbildung 3.6 ist der Korrelationsalgorithmus als Aktivitätsdiagramm dargestellt. Im Folgenden werden die Aktivitäten genauer beschrieben.

- **Initialisieren der Daten:**

  Die Klasse `CCalcPos` wird mit den Voreinstellungen des Nutzers, wie Messfeldgröße, Quell-und Zielpfad, initialisiert. Dabei werden mit der Methode `validate()` aus der Controllerklasse die Daten auf Gültigkeit überprüft. Beispielsweise darf die Messfeldgröße nicht kleiner als die Referenzfeldgröße sein. Falls ein Fehler gefunden wurde, wird der Nutzer mit einer entsprechenden Fehlermeldung informiert und der Algorithmus wird abgebrochen.

- **Korrelationsthread starten:**

  Die Klasse `CCalcPos` ist von `CThread` abgeleitet. `CThread` ist eine abstrakte Klasse, welche es ermöglicht, separate Threads zur Ausführung in einer Anwendung zu erstellen. Weiterhin stehen Methoden zum sauberen Abbruch und Abfragen des Ausführungsstatus des Thread bereit. Eine Dokumentation zu dieser Klasse befindet sich im Anhang E.2.2.

  Nach Aufruf der Methode `Start()` wird der Algorithmus in einem Thread gestartet. Der Thread verhindert, dass die GUI blockiert wird und somit das Programm weiter vom Anwender steuerbar bleibt.

- **Korrelationsframework initialisieren:**

  Der Thread initialisiert zunächst das Korrelationsframework mit Standardeinstellungen und Einstellungen des Nutzers. Außerdem wird auf Fehler und Gültigkeit von Daten geprüft, die erst nach der Initialisierung des Frameworks festgestellt werden können. Z.B. ist eine Messpunktdatei mit zu wenig oder keinen Angaben von Messpunkt Koordinaten ungültig. Der Nutzer wird mit einer entsprechenden Fehlermeldung informiert und der Algorithmus wird abgebrochen.

- **Prüfen auf neues Bild, warten auf neues Bild und Abbruch durch Nutzer:**

  Unter Verwendung der Windows API-Funktionen `FindFirstChangeNotification()`, `FindNextChangeNotification()`, und `FindCloseChangeNotification()` wird der vom Nutzer angegebene Quellordner auf Änderungen überwacht. Windows „informiert“ damit über Änderungen in dem Ordner ermittelt aber nicht welche Datei neu erstellt wurde. Da die Dateinamen der Bilder mit einer laufenden Nummer erstellt werden, kann über einen mitgeführten Zähler der erwartete Dateiname bestimmt werden. Bei jeder Änderung im Quellordner wird geprüft, ob die erwartete Datei existiert. Somit fungieren die ChangeNotification-Funktionen, unter Anwendung der API-Funktion `WaitForMultipleObjects()`, nur als „Blocker“ um ein stetiges Abfragen, z.B. in einer Endlosschleife (*polling*), zu vermeiden. Diese Aktivität ist in Listing 3.1 als Pseudocode dargestellt.

```
HANDLE hNotify =
   FindFirstChangeNotification(m_sSourcePath.c_str(), false,
   FILE_NOTIFY_CHANGE_LAST_WRITE);
HANDLE hEvents[2] = {m_hQuit, hNotify};
DWORD dwFileNum = FIRST_FILE_NUM;

if (hNotify != INVALID_HANDLE_VALUE)
{
   while (!Terminated())
   {
      // Prüfen ob Bild existiert
      if (CheckFileExists(dwFileNum))
      {
         // Bild existiert -> Zähler erhöhen und nächstes
         doCorrelation();
         dwFileNum++;
      }
      else
      {
         // Bild existiert nicht -> Auf Änderung warten
         FindNextChangeNotification(hNotify);
         WaitForMultipleObjects(2, hEvents, false,
            INFINITE);
      }
   }
   FindCloseChangeNotification(hNotify);
}
```

Listing 3.1 – Pseudocode der „Prüfen auf neues Bild“ Aktivität

- **Verschiebungsvektoren berechnen:**

  Wenn ein neues Bild in den Quellpfad geschrieben wurde, werden zunächst unter Verwendung des Korrelationsframeworks die Verschiebungsvektoren der virtuellen Messpunkte zum vorhergehenden erfassten Bild berechnet. Diese Vektoren werden zunächst mit einer Korrekturmethode (hier symbolisch die Methode `CorrectCorrData()`) korrigiert und die durchschnittliche Verschiebung aller Messpunkte ermittelt. Der Durchschnittswert dient zur Berechnung des Offsets und der Absolut-Koordinaten der Plattform. Weiterhin wird aus den Vektoren mit der Klasse `CRotation` der Rotationswinkel berechnet. Somit können die Werte in die im Kapitel 3.2 beschriebene Formel 3.1 zur Berechnung der Koordinaten eingesetzt werden. Die Koordinaten werden mit der Methode `AddCoord()` der Klasse `CRouteDraw` auf die Benutzeroberfläche gezeichnet. Dadurch zeichnet sich der zurückgelegte Weg der Plattform, zeitlich parallel zur Korrelation, ab. Listing 3.2 zeigt einen Quellcodeausschnitt dieser Aktivität.

```
if ((res = CalcCorr(&img, &m_calc.CorrParam,
       m_calc.pCorrData)) == CK_SUCCESS)
{
  double dXAverShift;
  double dYAverShift;
  if (CorrectCorrData(m_calc, &dXAverShift, &dYAverShift) > 0)
  {
    // Offset berechnen
    m_calc.CorrParam.dXShiftOffset += m_dOffsetFactor *
      (dXAverShift - m_calc.CorrParam.dXShiftOffset);
    m_calc.CorrParam.dYShiftOffset += m_dOffsetFactor *
      (dYAverShift - m_calc.CorrParam.dYShiftOffset);

    // Winkel berechnen
    double dPhi = m_pRotation->CalculateAngle(&m_calc);
    m_dAngleSum += dPhi;

    // Koordinaten berechnen
    double dHypo = sqrt(dXAverShift * dXAverShift +
      dYAverShift * dYAverShift);
    m_dLastXPos += dHypo * cos(m_dAngleSum);
    m_dLastYPos += dHypo * sin(m_dAngleSum);

    // Neue Koordinate anzeigen
    m_pRouteDraw->AddCoord(m_dLastXPos, m_dLastYPos);
  }
}
```

Listing 3.2 – Auszug der Berechnung der Absolutkoordinaten (verkürzt)

- **Messfeldgröße erhöhen:**

  Wenn eine Korrektur der berechneten Verschiebungsvektoren, durch Fehlen von korrekt berechneten Vektoren nicht möglich ist, wird angenommen, dass die angegebene Messfeldgröße zu klein ist. Deshalb wird die Größe verdoppelt und die Korrelation erneut ausgeführt. Dies geschieht solange bis eine Mindestzahl von korrekten Verschiebungsvektoren berechnet wurde und somit angenommen werden kann, dass die Korrelation erfolgreich war. Wenn die Messfeldgröße das Vierfache (Wert änderbar; hat sich in vielen Tests als effizient erwiesen) des Ausgangswertes übersteigt, wird angenommen, dass ein anderer Fehler vorliegt und die Korrelation wird mit einer entsprechenden Meldung an den Nutzer abgebrochen.

- **Berechnung speichern:**

  Nachdem die Koordinaten berechnet wurden, werden diese und die Zwischenergebnisse in eine Logdatei im Textformat abgelegt. Somit können die Ergebnisse in einem beliebigen Tabellenkalkulationsprogramm ausgewertet werden. Beispielsweise können die Koordinaten in einem Diagramm dargestellt werden um den gefahrenen Weg zurück zu verfolgen. Außerdem werden die Verschiebungsvektoren zusätzlich als *VEDDAC* Speicherformat abgelegt um mit dem *VEDDAC*-Programmsystem weiter Untersuchungen vornehmen zu können.

3.4. Geschwindigkeitsoptimierung durch Offset

Die Korrelationsberechnung beansprucht den größten Teil der verfügbaren Rechenleistung. Obwohl das Korrelationsframework bereits eine Multithreading Unterstützung von bis zu 8 Prozessorkernen bietet, dauern die Berechnungen, durch einen Aufwand von $O(m*n^2)$ (bedingt durch die Suche des Referenzfeldes im Messfeld, was für jeden Messpunkt wiederholt werden muss) bei einer Messfeldgröße von 200 und 30 Messpunkten, bereits rund eine Sekunde auf aktuellen Rechnersystemen. Entsprechend der Bildrate soll jedoch der Korrelationsalgorithmus mindestens 30-mal pro Sekunde ausgeführt werden, sodass dies kein akzeptabler Wert für eine Echtzeitnahe Berechnung darstellt.

Eine mögliche Lösung, um die Geschwindigkeit zu erhöhen, ist die Verringerung der Messfeldgröße. Der Unterschied der Verschiebungsvektoren zwischen zwei aufeinanderfolgenden Korrelationen ist sehr gering, da der zeitliche Abstand zwischen zwei Bildern, bei Aufnahmen von 30fps, bei rund 33 Millisekunden liegt. In dieser Zeit ist nicht zu erwarten, dass signifikante Änderungen an der Geschwindigkeit und Richtung des Roboters vorhanden sind. Folglich kann mit einem nur kleinen Fehler vorab geschätzt werden wohin der als nächstes berechnete Vektor zeigen wird. Es reicht dann in der Umgebung dieses Zielpunktes ein kleines Referenzumfeld zu betrachten. Somit braucht das Suchfeld nur diese „nähere Umgebung“ abdecken und kann deshalb stark verkleinert werden. Dies wird umgesetzt, indem das Referenzbild um den berechneten Vektor „verschoben“ wird, d.h. der im Vorabschnitt berechnete Verschiebungsvektor wird als Offset der Folgeberechnung verwendet.

Wie Versuche zeigen, kann mit dem Offset eine weitere Optimierung vorgenommen werden, indem er, unter Verwendung der Formel 3.1, nachgeführt wird. D.h., dass nicht einfach der vorhergehend berechnete Vektor als Offset dient, sondern dieser Wert langsam um einen angegebenen Faktor verändert wird. Dies hat zufolge, dass zufallsbedingte Schwankungen in der Richtung der berechneten Vektoren ausgeglichen werden. [Lau10]

$$O_{n+1} = O_n + f(\Delta x - O_n)$$

f = Offsetfaktor; O=Offset; Δx = Durchschnittsverschiebung
Formel 3.2 – Rekursionsformel zur Berechnung des Offset

In Abbildung 3.7 sind die Auswirkungen der Größe des Offsetfaktors anhand des Vergleiches zwischen berechneter Verschiebung und unterschiedlich „gefilterten“ Verschiebungen (Faktor 0,2 und 0,6) dargestellt. Es ist zu erkennen, dass ein kleiner Wert die Schwankungen besser ausgleicht aber der Unterschied zu den tatsächlich berechneten Verschiebungsvektoren sehr

groß ist. Das bedeutet, dass ein größeres Messfeld benötigt wird. Ein Faktor von 0,6 gleicht Unterschiede ausreichend aus um die Messfeldgröße weiter zu verkleinern.

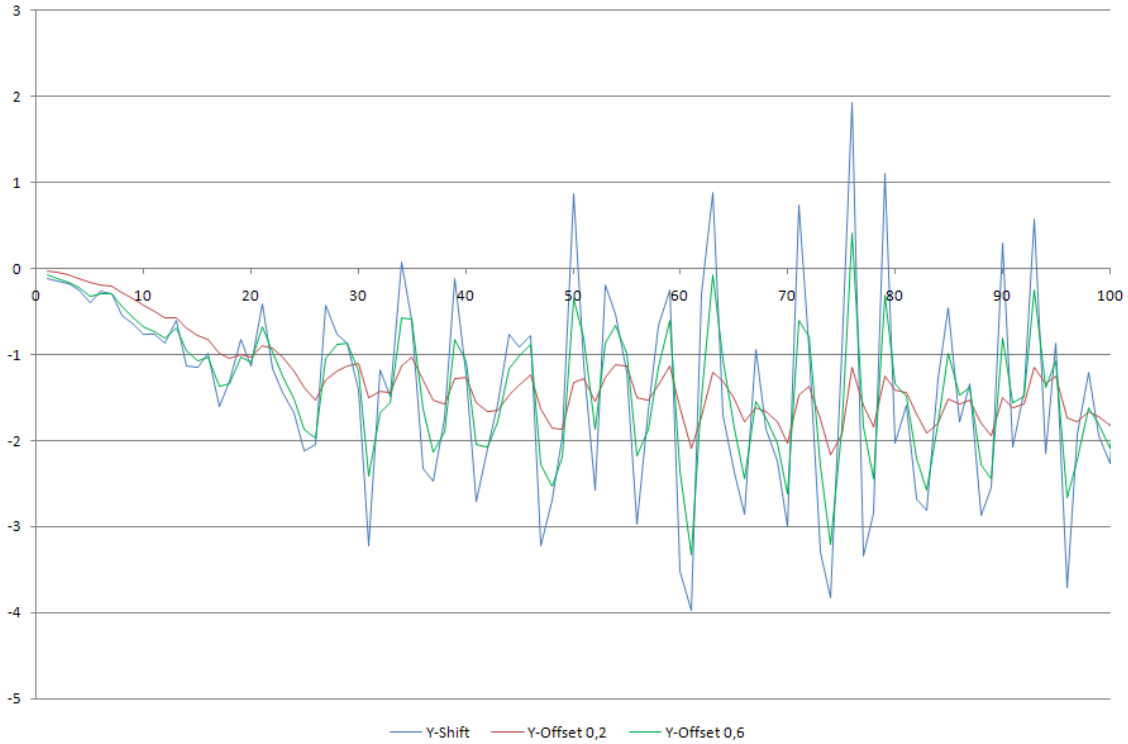


Abbildung 3.7 – Vergleich zwischen zwei Offsetfaktoren und Verschiebung in Y-Richtung

3.5. Rotation

Mit Hilfe des Rotationswinkels kann die Drehung des Roboters bestimmt werden welche wichtig für die Bestimmung der Position nach Formel 3.1 benötigt wird. Abbildung 3.8 zeigt eine Skizze zur Berechnung der Rotation. In diesem Beispiel wurde mit Hilfe eines Bildbearbeitungsprogrammes, künstlich ein Rotationswinkel von 10° erstellt, welcher im Folgenden, unter Verwendung der Grauwertkorrelation, berechnet wird.

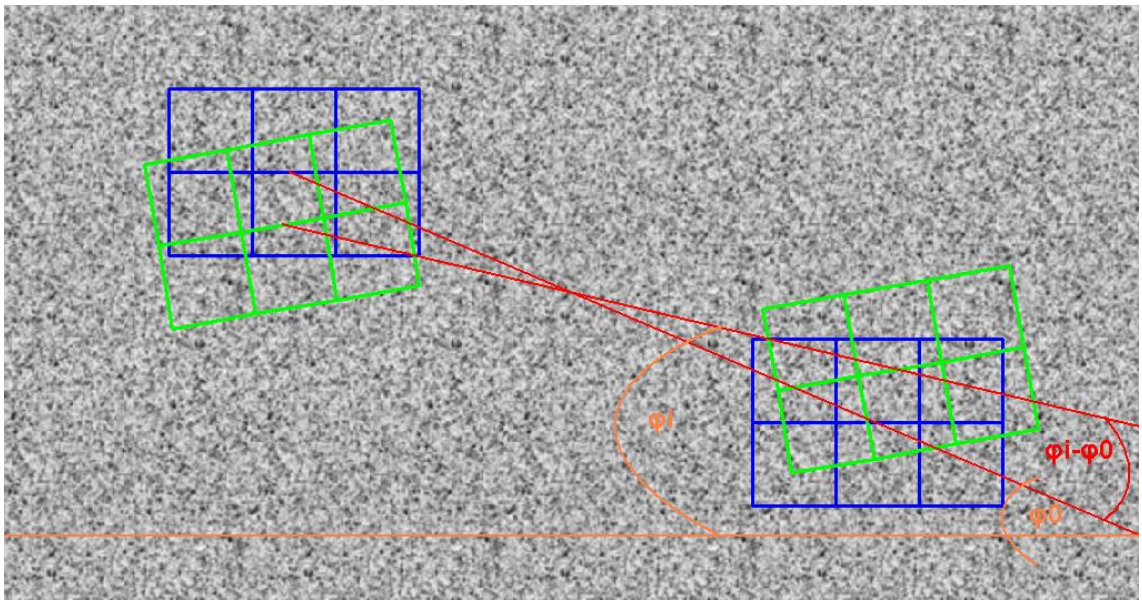


Abbildung 3.8 – Skizze zur Berechnung des Rotationswinkels

Um den Rotationswinkel φ zu berechnen, werden zunächst die Menge aller Messpunkte in einen linken und rechten Teil gesplittet (bzw. in einen oberen und unteren Teil). Aus einer gedachten Linie durch den Mittelpunkt der beiden Messpunkthälften (blaues Raster) und zu den Bildrand, wird der Winkel φ_0 berechnet, welcher die Grunddrehung der Messpunkte beschreibt. Da die Koordinaten der Messpunkte nicht verändert werden, ist dieser Winkel immer konstant. Nach der Korrelation wird wiederum mit einer gedachten Linie durch den Mittelpunkt der neu berechneten lokalen Koordinaten (grünes Raster) zu, der Winkel φ_i berechnet. Aus der Differenz dieser beiden Winkel ergibt sich der Rotationswinkel φ.

	linkes Messfeld		**rechtes Messfeld**		**Winkel ϕ**	
	X-Shift (ΔX_l)	**Y-Shift (ΔY_l)**	**X-Shift (ΔX_r)**	**Y-Shift (ΔY_r)**	**(Bogenmaß)**	**(Grad)**
Referenz	262,5	300	787,5	525	0,4048	23,198
Messung	251,5966	347,0266	807,9181	477,0826	0,2296	13,158
					- 0,1752	10,04

$$\varphi = \arctan(\Delta X_r - \Delta X_l;\ \Delta Y_r - \Delta Y_l)$$

Tabelle 3.2 – Berechnung des Rotationswinkels aus Abbildung 3.8

In Tabelle 3.2 wird der Rotationswinkel aus Abbildung 3.8 berechnet. Das Bild wurde zuvor um 10° gedreht, welches mit dem Ergebnis aus der Korrelationsberechnung übereinstimmt.

3.6. Korrektur der Verschiebungsvektoren

Da die Richtung der Vektoren mit Hilfe eines Wahrscheinlickeitswertes bestimmt werden und es keinen „richtigen“ oder „falschen“ Vektor gibt, können manche Vektoren in eine andere

Richtung zeigen, als der Roboter fährt (Abbildung 3.9). Das hat zur Folge, dass der Durchschnittsvektor aller Vektoren stark von der tatsächlichen Richtung abweicht, was die Bestimmung der Position und des Rotationswinkels verfälscht. Dies kann durch Befahren von unebenem oder ungünstig strukturiertem Untergrund wie Schotter oder Rasen auftreten. Im Folgenden werden mehrere Alternativen dargestellt, wie man die abweichenden Vektoren erkennen und korrigieren kann.

Abbildung 3.9 – Verschiebungsvektoren vor und nach Korrektur

3.6.1. Verschiebungskorrektur mittels *Backcorrelation*

Eine Möglichkeit die abweichenden Vektoren zu erkennen ist, dass die Bilder ein zweites Mal in umgekehrter Reihenfolge korreliert werden (*Backcorrelation*). Dazu werden zunächst die Verschiebungsvektoren der ersten Korrelation auf die Messpunktkoordinaten addiert. Diese neuen Messpunkte werden nun korreliert. Die so erhaltenen Vektoren müssen exakt in entgegengesetzte Richtung der ersten Berechnung zeigen. Wenn das nicht der Fall ist und ein Vektor in eine andere Richtung zeigt, bedeutet das, dass dies ein „falscher“ Vektor ist. Dieser wird bei der Berechnung des Durchschnittsvektors ausgelassen.

Im Programm ist dies in der Methode `CorrectCorrBackcorr()` umgesetzt, welche direkt nach der Korrelationsberechnung ausgeführt wird (Abbildung 3.10). Dabei wird zunächst das Referenzbild und Messbild getauscht und die Verschiebungsvektoren auf die Messpunkte addiert. Da die erwartete Richtung eines Vektors bekannt ist, kann ein umgekehrter Offset und ein sehr kleines Messfeld verwendet werden, um die Korrelation zu beschleunigen. Danach wird mit den neuen Parametern korreliert. Die Subtraktion eines Vektors aus der ersten Korrelation mit dem Zurück berechneten Vektor ergibt genau null, wenn keine Korrektur vorgenommen werden muss. Da es in der Praxis nahezu unmöglich ist, dass dieser Wert genau Null ergibt, wird komponentenweise geprüft, dass das Ergebnis der Subtraktion innerhalb der Standardabweichung aller Verschiebungen liegt. Ist dies nicht der Fall, wird die Durchschnittsver-

schiebung aller „korrekten“ Vektoren eingesetzt und dieser Vektor für die Wegfindung verworfen. Ansonsten wird er in die Berechnung der Durchschnittsverschiebung einbezogen.

Wenn keiner der Vektoren innerhalb der Abweichung liegt, kann keine Durchschnittsverschiebung berechnet werden und es wird angenommen, dass das Messfeld zu klein ist. Eine neue Korrelation mit größerem Messfeld ist nötig.

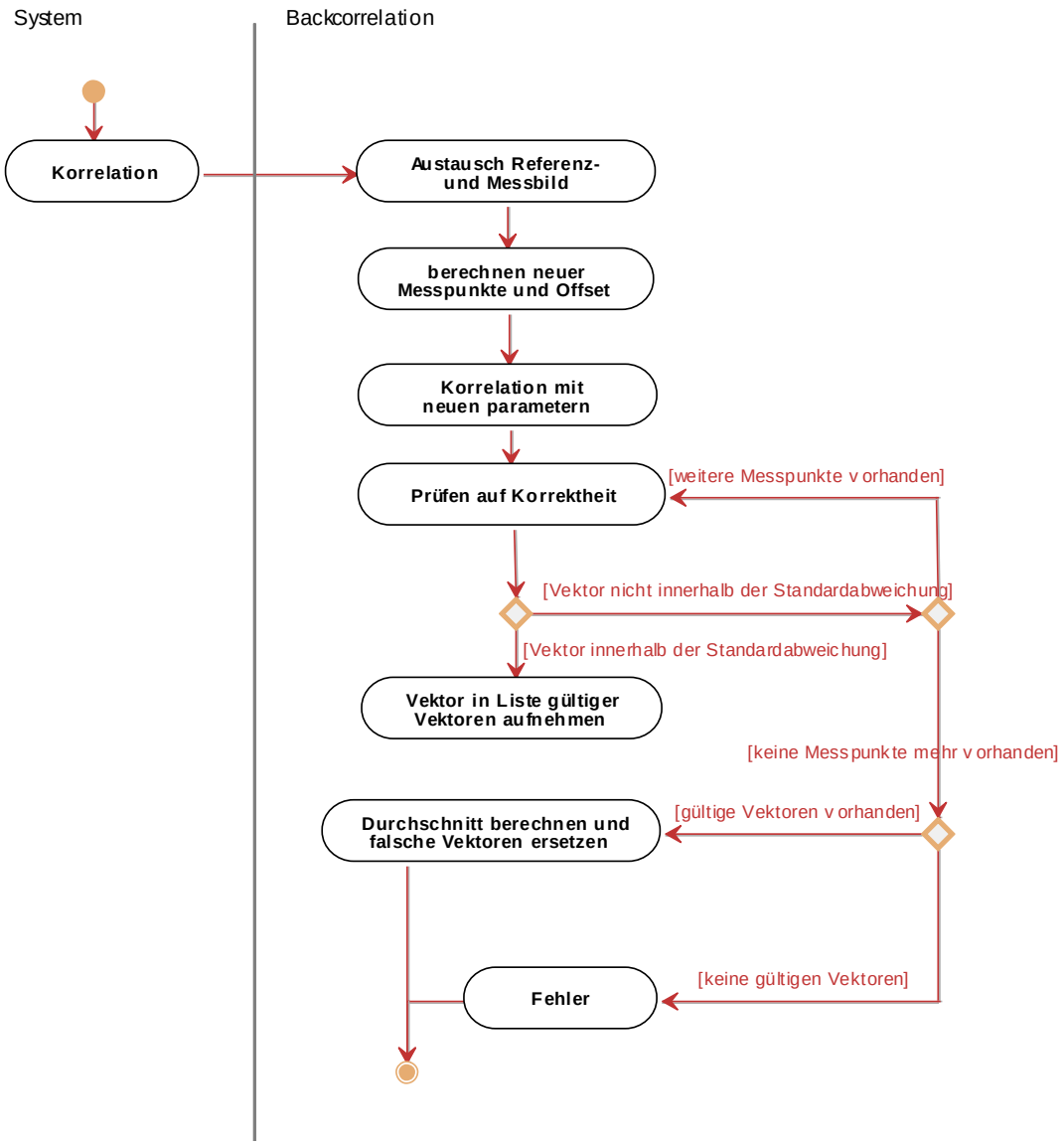


Abbildung 3.10 – Aktivitätsdiagramm *Backcorrelation*

3.6.2. Verschiebungskorrektur mittels Auswahl nach Korrelationskoeffizient

Eine weitere Möglichkeit Vektoren zu korrigieren ist, dass ein hoher Korrelationskoeffizient gefordert wird, damit ein Vektor gültig ist. Allerdings hat diese Methode hat den Nachteil, dass eine gute Oberflächenstruktur, wie z.B. Asphalt oder Kies, vorhanden sein muss, da sonst der Koeffizient abnimmt und auch korrekte Vektoren als falsch erkannt werden.

Im Programm wird dazu die Methode `CorrectCorrKoeff()` aufgerufen, welche jeden Vektor überprüft, ob er einen Mindestkoeffizienten nicht unterschreitet. Falls er unterschritten wird, wird dieser Vektor nicht für die Berechnung des Durchschnittsvektors verwendet.

3.6.3. Verschiebungskorrektur mittels Median-Abweichung

Der Mittelwert aller Verschiebungen kann verwendet werden, um stark abweichende Vektoren als „Ausreißer“ zu erkennen und zu eliminieren.

Die Methode `CorrectCorrMid()` setzt dies um, indem zunächst der Durchschnittswert und die Standardabweichung mittels der Formel 3.3 berechnet wird. Danach wird geprüft, ob die Vektoren innerhalb dieser Abweichung liegen. Wenn das der Fall ist, wird der Vektor zur Berechnung der Durchschnittsverschiebung verwendet, ansonsten eliminiert.

$$\sigma_x^2 = \mathrm{VAR(X)} = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n * \overline{x}^2\right)$$

Formel 3.3 – Standardabweichung σ_x

Abbildung 3.11 zeigt dieses Verfahren Graphisch. Der rote Punkt ist der Mittelpunkt aller Vektoren um den die Standardabweichung ein Rechteck bildet. Alle Messpunkte (blau markiert) welche innerhalb des grünen Rechteckes liegen, werden als korrekte Verschiebung angenommen. Alle anderen Punkte werden eliminiert und nicht für die Berechnung der Verschiebung verwendet.

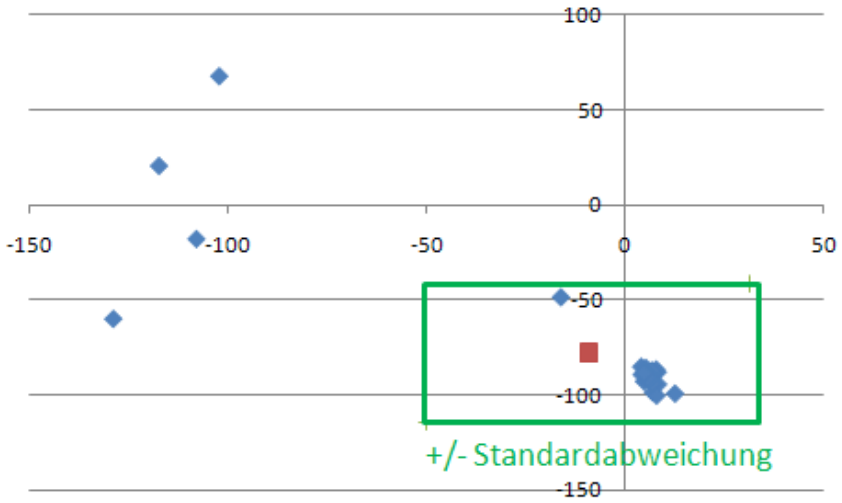


Abbildung 3.11 – Auswahl der Messpunkte in bestimmten Bereich

3.6.4. Verschiebungskorrektur mittels absoluter Häufigkeit

Die Verschiebungskorrektur mittels absoluter Häufigkeit kommt dem intuitiven Erkennen, fehlerhafter Vektoren, durch den Menschen in ihrer Funktionsweise und im Ergebnis sehr nahe. Wie in Abbildung 3.11 der Mediankorrektur auch zu erkennen ist, häufen sich Vektoren in einem bestimmten Bereich, während sich einige wenige in anderen Bereichen befinden. Die Auswertungsvoraussetzung hierbei ist, ein Radial-Koordinatensystem in Winkel-Bereiche zu unterteilen und die Häufigkeit der Vektoren in diesen Bereich zu zählen. Aus dem so entstandenen Histogramm werden nunmehr nur noch diejenigen Vektoren verwendet, die in den Bereichen mit dem höchsten Häufigkeiten liegen (Abbildung 3.12).

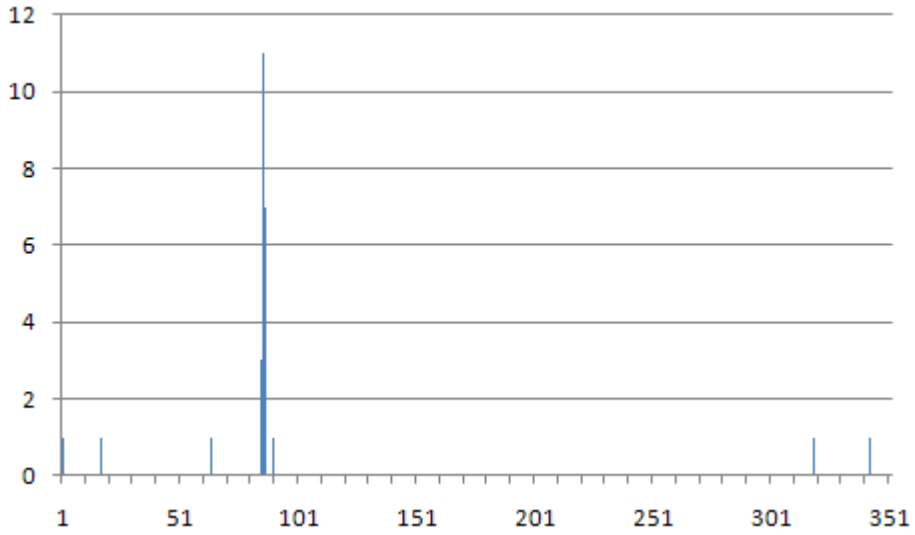


Abbildung 3.12 – Histogramm Korrekturmethode „Absolute Häufigkeit“ mit der Klassenbreite 1° und willkürlichen Diskriminatorwert 2

Die Methode `CorrectCorrAbsoluteFreq()` setzt dieses Prinzip um. Es werden aus den Verschiebungsvektoren die Winkel zum Nullpunkt berechnet. Die Vektoren häufen sich in einigen (den „gültigen“) Klassen während andere Klassen keine oder nur sehr wenige Vektoren („Ausreißer“) enthalten. Die Vektoren einer Klasse werden gezählt und aus der nachfolgenden Auswertung eliminiert, wenn die Anzahl unter einem an sich willkürlich vorgegebenen Schwellwert (hier: 2) liegt. Nur die übriggebliebenen Vektoren aus den gültigen Klassen gehen in die Berechnung des resultierenden Durchschnittswinkels ein.

3.6.5. Vergleich der Methoden und Auswahlbegründung

In Tabelle 3.3 sind die Korrelationsergebnisse aus Abbildung 3.9 ohne Korrektur und mit den vorstehen beschriebenen Korrekturmethoden gegenüber gestellt. Die gelb markierten Messpunkte stellen diejenigen Vektoren dar, welche nach intuitiver Betrachterwertung des Autors

stark von den Werten der anderen Vektoren abweichen und dadurch den durchschnittlichen Verschiebungswert signifikant verfälschen. Die grün markierten Vektoren sind von dem jeweiligen Korrekturalgorithmus als falsche erkannt und mit den Durchschnittswert aus allen anderen Vektoren ersetzt wurden. Die rot markierten sind entweder nicht als falsch erkannt wurden oder korrigiert, obwohl die Korrelationsberechnung korrekte Werte lieferte.

In der Tabelle 3.3 ist zu erkennen:

- Die Korrektur über den Median und die *Backcorrelation* Methode ergeben praktisch die gleichen Durchschnittswerte. Der Unterschied der Ergebnisse beider Methoden zwischen den X und Y Verschiebungskomponenten liegt unter einem Pixel und ist damit zu vernachlässigen.

- *Backcorrelation* erkennt korrekterweise alle gelb markierten Messpunkte als falsch, eliminiert aber zusätzlich einen weiteren Messpunkt, der jedoch nur einen geringen Einfluss auf die Gesamtdurchschnittsberechnung hat.

- Die Median-Methode erkennt alle „Ausreißer“ außer Einem korrekterweise als falsch.

- Die Methode über den Korrelationskoeffizienten eliminiert 3 von 5 fehlerhaften Vektoren korrekt, korrigiert aber auch 2 nicht und erkennt zusätzlich 3 Fehler.

- Bei der Korrektur unter Nutzung der Winkelklassenhäufigkeiten wurden alle Fehler korrekt eliminiert und 2 zusätzlich erkannt (was jedoch das Ergebnis nicht wesentlich nachteilig beeinflusst).

Der Grund dafür, dass die Koeffizienten-Korrektur nur mäßige Ergebnisse erzielt ist, dass der Schwellwert von 0,85 zu hoch für die Struktur der aktuellen Bilder ist. Ein Schwellwert von 0,8 würde (allerdings nur für diese konkreten Bilder) bessere Ergebnisse erbringen. In der Praxis muss damit gerechnet werden, dass der Roboter auf verschiedenen Untergründen zum Einsatz kommt und damit verschiedene Bildqualitäten existent sind. Dadurch ist eine Bestimmung eines allgemeingültigen Koeffizienten- Schwellwertes nicht möglich.

Die *Backcorrelation* Methode erfordert durch den zusätzlich notwendigen Korrelationsvorgang einen deutlich höheren Rechenaufwand. Außerdem ist es derzeit nicht möglich im Korrelationsframework ein individuelles Offset für jeden einzelnen Messpunkt anzugeben, sondern nur für die gesamte Menge der Messpunkte. Das heißt, dass eine Korrelation mit nahezu allen falschen Vektoren das Offset so stark verändert, dass die (rechenzeitfreundliche) kleine Messfeldgröße bei der *Backcorrelation*-Berechnung nicht ausreicht und damit keine verwertbaren

Ergebnisse liefert. Dies kann umgangen werden, indem die ursprünglich für die Hin-Berechnung genutzte Messfeldgröße in der Back-Korrelationsrichtung unverändert beibehalten wird, was aber zur Folge hat, dass der Rechenaufwand doppelt so groß wie bei der einfachen Korrelationsberechnung wird.

Mess-punkt	ohne Korrektur		Backcorrelation		Koeffizient (r=0.85)		Median		Winkelklassen-Häufigkeit		max-Koeff
	ΔX	ΔY	ΔX	ΔY	ΔX	ΔY	ΔX	ΔY	ΔX	ΔY	
1	7,82	-88,34	7,82	-88,34	7,82	-88,34	7,82	-88,34	7,82	-88,34	0,88
2	6,79	-91,77	6,79	-91,77	6,79	-91,77	6,79	-91,77	6,79	-91,77	0,92
3	8,46	-94,78	8,46	-94,78	8,46	-94,78	8,46	-94,78	8,46	-94,78	0,92
4	7,12	-89,16	7,12	-89,16	eliminiert		7,12	-89,16	7,12	-89,16	0,83
5	-117,5	20,64	eliminiert		-117,5	20,64	eliminiert		eliminiert		0,87
6	4,87	-93,23	4,87	-93,23	4,87	-93,23	4,87	-93,23	4,87	-93,23	0,94
7	-129,1	-60,21	eliminiert		eliminiert		eliminiert		eliminiert		0,80
8	4,13	-85,48	4,13	-85,48	4,13	-85,48	4,13	-85,48	4,13	-85,48	0,96
9	8,34	-88,27	8,34	-88,27	8,34	-88,27	8,34	-88,27	8,34	-88,27	0,90
10	7,03	-98,97	7,03	-98,97	7,03	-98,97	7,03	-98,97	7,03	-98,97	0,96
11	7,31	-94,26	7,31	-94,26	7,31	-94,26	7,31	-94,26	7,31	-94,26	0,98
12	-16,12	-48,78	eliminiert		-16,12	-48,78	-16,12	-48,78	eliminiert		0,88
13	6,95	-89,79	6,95	-89,79	6,95	-89,79	6,95	-89,79	6,95	-89,79	0,92
14	-108,1	-18,02	eliminiert		eliminiert		eliminiert		eliminiert		0,79
15	5,33	-87,66	5,33	-87,66	5,33	-87,66	5,33	-87,66	5,33	-87,66	0,89
16	4,26	-89,61	4,26	-89,61	4,26	-89,61	4,26	-89,61	eliminiert		0,88
17	7,93	-86,89	7,93	-86,89	7,93	-86,89	7,93	-86,89	7,93	-86,89	0,98
18	8,06	-100,7	8,06	-100,7	8,06	-100,7	8,06	-100,7	8,06	-100,7	0,98
19	6,92	-94,26	6,92	-94,26	6,92	-94,26	6,92	-94,26	6,92	-94,26	0,98
20	5,92	-92,11	5,92	-92,11	5,92	-92,11	5,92	-92,11	5,92	-92,11	0,98
21	6,23	-92,08	6,23	-92,08	6,23	-92,08	6,23	-92,08	6,23	-92,08	0,96
22	6,34	-89,17	6,34	-89,17	6,34	-89,17	6,34	-89,17	6,34	-89,17	0,97
23	-102,3	67,97	eliminiert		eliminiert		eliminiert		eliminiert		0,80
24	5,31	-86,13	5,31	-86,13	5,31	-86,13	5,31	-86,13	5,31	-86,13	0,98
25	7,06	-88,04	7,06	-88,04	7,06	-88,04	7,06	-88,04	7,06	-88,04	0,94
26	6,77	-87,46	6,77	-87,46	6,77	-87,46	6,77	-87,46	6,77	-87,46	0,92
27	5,54	-88,73	eliminiert		eliminiert		5,54	-88,73	5,54	-88,73	0,74
28	5,67	-88,73	5,67	-88,73	eliminiert		5,67	-88,73	5,67	-88,73	0,80
29	5,76	-91,18	5,76	-91,18	5,76	-91,18	5,76	-91,18	5,76	-91,18	0,89
30	5,20	-88,64	5,20	-88,64	5,20	-88,64	5,20	-88,64	5,20	-88,64	0,96
31	12,69	-99,70	12,69	-99,70	12,69	-99,70	12,69	-99,70	eliminiert		0,97
32	4,81	-88,73	4,81	-88,73	4,81	-88,73	4,81	-88,73	4,81	-88,73	0,87
Med.:	-9,21	-77,88	6,66	-90,97	1,02	-85,21	5,80	-89,38	6,47	-90,58	

Legende:

fehlerhaft berechnete Messpunkte nach intuitiver visueller Einschätzung („Ausreißer“)

Richtig eliminierte Messpunkte

Messpunkte, die zwar „fehlerhafte“ Vektoren liefern, jedoch unkorrekter weise eliminiert bzw. nicht korrigiert wurden

Tabelle 3.3 – Vergleich der Korrekturmethoden

Weitere Tests mit Bildsequenzen auf unterschiedlichen Untergründen haben gezeigt, dass die Median Berechnung, *Backcorrelation* und Häufigkeit ähnliche Endergebnisse ergeben und die

Koeffizienten-Methode nur bei sehr guter Bildqualität und gut strukturiertem Untergrund akzeptable Werte liefert, was mit dem hier gezeigten Test-Beispiel übereinstimmt.

Das Verfahren über die absolute Häufigkeit bewährte sich im Test insbesondere, wenn (im Unterschied zu Abbildung 3.9) eine Anordnung von zwei Messpunktclustern mit nur wenigen Einzelmesspunkten jeweils nahe dem rechten und linken Bildrand eingesetzt wurde. Hierdurch werden nur wenige Berechnungen und eine kurze Rechenzeit benötig, während die Winkelberechnung wegen der relativ großen Entfernung der Clusterschwerpunkte voneinander relativ genau wird. Somit ist dieses Verfahren derzeit am besten geeignet, da es sehr gute Ergebnisse bei nur geringem zusätzlichem Rechenaufwand liefert.

4. Referenzfahrten

Um die Genauigkeit der Positionsbestimmung mit unterschiedlichen korrekturverfahren zu ermitteln, wurden mehrere Testfahrten im Labor unternommen. Die CCD-Kamera *AVT Guppy 080-F*, welche auch in den bisherigen Testfahrten zur Verwendung kam, wurde auf einer beweglichen, schienengestützten Vorrichtung angebracht. In das Aufnahmefeld wurde ein Maß gelegt, um die zurückgelegte Strecke metrisch ermitteln zu können.

Als Software kamen zum einen *VEDDAC Cam* zum Einsatz, um die Bilder zu erstellen und zum anderen das in Kapitel 3.2 beschriebene Programm *CorroPos* zur Positionsbestimmung mit den alternativen Korrekturalgorithmen, zum Einsatz.

Mit den so gesammelten Daten und durch das im Aufnahmebereich angebrachte Maß, kann die berechnete Position mit der tatsächlichen Position verglichen werden um somit eine Abweichung zu berechnen.

4.1. Auswertung der Daten

Für die Positionsbestimmung nach der Formel 3.1 sind zwei Werte für die Berechnung essentiell: die durchschnittliche Verschiebung und der Rotationswinkel. Es gilt genau diese zwei Werte zu überprüfen.

Dazu wurde zur Bestimmung der Verschiebung eine gerade Strecke von ca. 60cm gefahren von welcher genau 50cm betrachtet werden. D.h. dass die Bildsequenz von dem ersten Bild bis zu dem Bild korreliert wird, indem die Messpunkte plus Offset auf 50cm zeigen (Abbildung 4.1, links).

Um die Berechnung des Winkels zu überprüfen wurde eine kreisförmige Strecke gefahren. Die Bildsequenz von einer angebrachten Markierung bis zur selben Markierung muss einen Winkel von genau 360° ergeben (Abbildung 4.1, rechts).

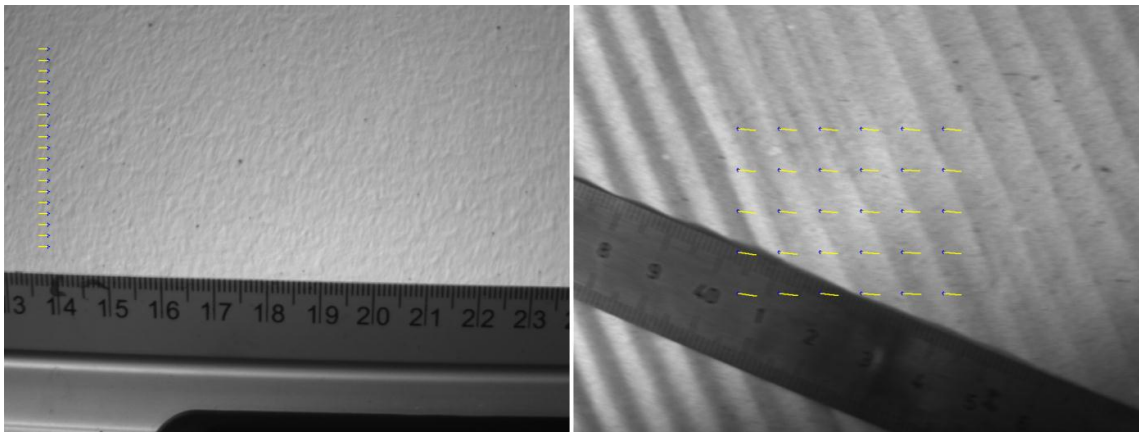


Abbildung 4.1 – Auszug aus Bildsequenz der Testfahrten. L.: Gerade; R.: Kreis

Mit Hilfe des Satzes von Pythagoras kann die Entfernung vom Koordinatenursprung zur Position in Pixel berechnet werden. Die Umrechnung von Pixel in cm erfolgt mit einem Umrechnungsfaktor, welcher sich aus der Anzahl der cm pro Pixel multipliziert mit dem Seitenverhältnis der Bilder ergibt. Da die Entfernung der Kamera zur Aufnahmeoberfläche feststehend ist, ist auch der Umrechnungsfaktor in der Ausgewählten Bildsequenz konstant.

Tabelle 4.1 zeigt die Ergebnisse der Umrechnung der berechneten Koordinaten in den zurückgelegten Weg der Referenzfahrt auf gerader Strecke. Es ist zu erkennen, dass bei der geraden Strecke, ein zurückgelegter Weg von 50,42cm annähernd unabhängig von der verwendeten Korrekturmethode, berechnet wurde. Dies entspricht einer Abweichung von nur 0,86% die in der Größenordnung der theoretischen Fehlergrenze bei der Bestimmung des Maßstabes aus einer Strecke von 10cm des Maßbandes liegt und daher als vernachlässigbar betrachtet werden kann. Zu bemerken ist, dass die Referenzfahrt unter optimalen Laborbedingungen (Beleuchtung, Untergrundstruktur) erstellt wurde. Inwieweit diese geringe Abweichung auf größeren Distanzen von mehreren Kilometern und im tatsächlichen Einsatz ebenfalls erreicht werden kann, muss zu einer späteren Projektphase ermittelt werden.

Fahrt	Backcorrelation	Median	Koeffizient	Häufigkeit	ohne Korrektur
Gerade	50,42cm (0,86%)	50,43cm (0,87%)	50,43cm (0,87%)	50,43cm (0,87%)	50,42cm (0,86%)
Kreis 1	357,57° (0,68%)	350,84° (2,54%)	---	351,23° (2,44%)	356,65° (0,93%)
Kreis 2	354,36° (1,57%)	350,95° (2,51%)	---	351,40° (2,39%)	339,17° (5,79%)
Kreis 3	349,92° (2,80%)	351,64° (2,32%)	---	354,00° (1,67%)	343,21° (4,66%)
Kreis 4	350,72° (2,58%)	349,18° (3,01%)	---	349,03° (3,05%)	341,16° (5,23%)

Tabelle 4.1 – Ergebnisse der Referenzfahrten

Die Ergebnisse für den Rotationswinkel, welche mit der geschlossenen Referenzkreisfahrt ermittelt wurden, sind ebenfalls in Tabelle 4.1 gezeigt. Für die Berechnung wurden, ausgehend von der angebrachten Markierung, alle Bilder welche einen vollständigen Kreis von 360° bilden für die Korrelation verwertet.

Da die Oberflächenstruktur bei den Kreisfahrten nicht optimal war und nur niedrige Korrelationskoeffizienten erzielt wurden, wird die Korrektur mittels Koeffizienten hier nicht betrachtet.

Die übrigen Korrekturmethoden ergeben eine Abweichung von maximal 3,05%. Der Fehler wird u.a. verursacht, durch die sehr kleinen Verschiebungsvektoren am Anfang der Bildsequenz. Diese liegen im Subpixelbereich und sind deshalb kaum für die Berechnung des Rotationswinkels zu verwerten. Außerdem schließen die Bildsequenzen, aus technischen Gründen, nicht genau auf den Punkt bei 360° ab sondern 2 bis 3° davor. D.h. der tatsächliche Fehler ist um diesen Winkel vermindert.

Der Unterschied der jeweiligen Kreisfahrten bestand in der Geschwindigkeit des Fahrzeuges. Diese seigt von Kreis 1 zu Kreis 4 an. Alle drei betrachteten Korrekturmethoden liefern annähernd die gleichen Auswertungsergebnisse von ca. 350°. Die Abweichung deutet wohl auf einen systematischen Fehler hin, ggf. noch näher untersucht wird, jedoch noch deutlich unter der Größenordnung der in den Einzelberechnungen aus Fehlern der Subpixelberechnung theoretisch zu erwartenden Winkelfehler liegt und daher als vernachlässigbar angesehen werden darf.

4.2. Vergleich mit herkömmlicher Odometrie

Bei der herkömmlichen Odometrie wird nach [Gut96] die zurückgelegte Strecke über die Umdrehungen der Räder und den Lenkwinkel ermittelt. Radgeometrie, Bodenbeschaffenheit, Fahrgestellgeometrie und Fahrzeuggewicht sind Einflussfaktoren welche sich auf die Positionsberechnungen auswirken und zu Regeldifferenzen führen. Diese Differenzen werden mit zunehmender Entfernung größer und führen zu immer ungenaueren Ergebnissen. Es existieren in der Robotik verschiedene Methoden, wie Ortung per GPS, Echoortung und mathematische Modelle, um derartige Fehleinflüsse zu minimieren. Sie sind aber praktisch nicht vollständig zu vermeiden.

Das KOBOLD-Prinzip ist unabhängig von Rad-, Fahrgestellgeometrie und Fahrzeuggewicht, sodass diese Faktoren keinen Einfluss auf die Selbstortung mehr haben. Allein die Bodenbeschaffenheit hat eine geringe Bedeutung für die Bestimmung der Verschiebungsvektoren. Ein Einflussfaktor kann die Dreidimensionalität der aufgenommenen Objekte (wie z.B. Grasbüschel oder kleine Steine) sein, deren Abbildung sich perspektivisch zwischen Referenz- und Messbild ändert und dadurch zu ungenauen Ergebnissen führt. Die in Kapitel 3.6 dargestellten Korrekturmethoden bereinigen jedoch teilweise auch derartige Fehlereinflüsse.

Die Qualität der aufgenommen Bilder ist eine mögliche weitere Fehlerquelle, welche bei herkömmlicher Odometrie keine Rolle spielt. Bei zunehmender Geschwindigkeit entsteht eine Bewegungsunschärfe die zu inkorrekten Ergebnissen führen kann. Allerdings kann dies durch hochwertige Kameras, angemessene Aufnahmetechniken und reproduzierbare künstliche Bodenbeleuchtung vermieden werden.

Die Regeldifferenzen die beim KOBOLD-Prinzip entstehen, sollen in einem weiteren Entwicklungsschritt durch eine Absolutortung auf Basis des Vergleichs der aktuellen Bodenstrukturen mit „eingelernten“ Bodenmustern korrigiert werden. Der Roboter fährt hierzu nach einer gewissen Zeit eine bestimmte „eingelernte“ Position im Einsatzbereich an. Sobald diese Position erreicht wurde, kann über die bekannten Koordinaten des „eingelernten“ Bodenmusters die genaue Position neu bestimmt werden. Auf diese Weise wird also ein „Reset“ der odometrisch berechneten Koordinaten durchgeführt.

5. Zusammenfassung und Ausblick

Die vorgelegte Arbeit behandelt die Erstellung eines Algorithmus zur Selbstortung einer autonomen mobilen Plattform auf Grundlage des von der Chemnitzer Werkstoffmechanik GmbH entwickelten digitalen Bildkorrelationsverfahrens. Dazu wurden zuerst die Systemarchitektur erläutert und theoretische Grundlagen vermittelt. Dazu gehört die Grauwertkorrelation und das Korrelationsframework, welches essenziell für die Entwicklung des in dieser Arbeit beschrieben Verfahrens zur Positionsbestimmung ist. Dieses Verfahren wird KOBOLD-Prinzip genannt und ist eine Erweiterung der Odometrie. Danach werden die einzelnen Schritte des Selbstortungsalgorithmus beschrieben und ein Vergleich mit Odometrie durchgeführt.

Im Laufe der Forschungsarbeit wurden mehrere Testprogramme entwickelt, welche einzelne Konzepte und Ideen zur Rückverfolgung des Weges aber technische Umsetzungen, wie die Verwendung einer Webcam, getestet werden konnte. Nach erfolgreichem Test wurden diese Konzepte in den gesamten Algorithmus implementiert für den wiederum ein Testprogramm, genannt *CorroPos* (siehe dazu auch Abbildung 5.1), entwickelt wurde. Das Programm wurde also stetig um einzelne Funktionen erweitert welche vorher ausführlich getestet wurden. Die Wiederverwendbarkeit des so entwickelten Algorithmus war ein wichtiger Aspekt auf den bei der Implementierung zu achten war.

Um eine Positionsbestimmung in Echtzeit durchführen zu können, war der bereits existierende Korrelationsalgorithmus in der Rechengeschwindigkeit zu optimieren. Dafür wird ein nachgeführtes Offset verwendet, welcher die bisher benötigte Rechenleistung für Grauwertkorrelation auf einen Bruchteil reduziert ohne Berechnungen zu verfälschen. Außerdem musste ein Verfahren entwickelt werden, welches den Rotationswinkel ermittelt. Dies wird erreicht indem die Drehung der gemessenen Verschiebung zwischen zwei Bildern bestimmt wird. Das Prinzip kann den Winkel subpixelgenau bestimmen und ist essenziell für die Erstellung der Trajektorie. Da die Korrelation ein mathematisch-statistisches Verfahren ist, können Abweichungen („Ausreißer“) einzelner Verschiebungsvektoren zu der tatsächlichen Verschiebung auftreten. Dies beeinflusst die berechnete durchschnittliche Verschiebung und somit die gemessene Bahnkurve. Deshalb wurden mehrere Verfahren zur Korrektur der Vektoren entwickelt.

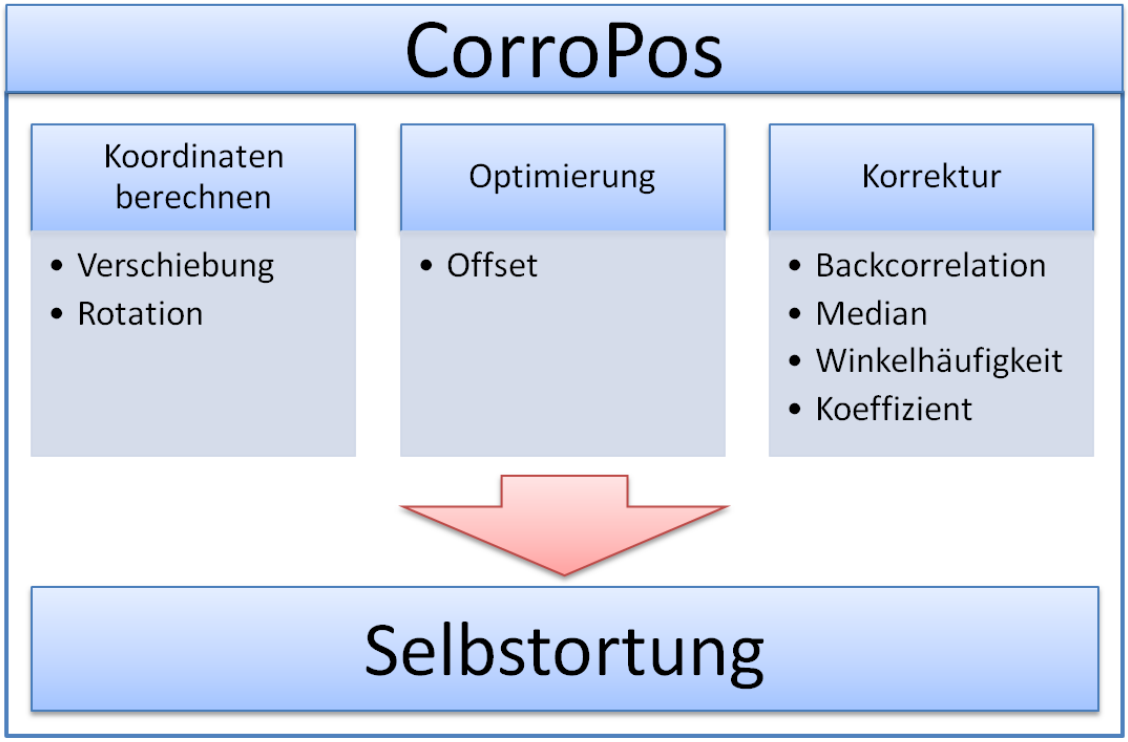


Abbildung 5.1 – Elemente zur Selbstortung im Programm *CorroPos*

Die bisher entwickelten Korrekturalgorithmen erzielen verwertbare Ergebnisse und stellen eine geeignete Basis für Weiterentwicklungen dar. Ein Ansatz für eine derartige Weiterentwicklung ist das Schärfemaß nach der in [Haa09a] beschriebenen Methode *Laplacian-of-Gaussian*. Nach diesem Konzept werden Messpunkte nur in Bildbereiche gesetzt, in denen dieser Wert am höchsten ist. Genau an diesen Stellen hat das Bild den höchsten Informationsgehalt und ist demnach auch am besten für Grauwertkorrelation geeignet. Ähnliche Methoden zum Herausfiltern von für Korrelation geeignete Bildbereiche sind, wie in [Sch98] beschrieben, die Ermittlung der lokalen Varianz oder Schätzung des lokalen Bildinformationsgehaltes (Entropie). Dabei werden über ein Schwellwertfahren die Bildbereiche herausgefiltert, welche eine besonders informationshaltige Struktur aufweisen und damit gut für Korrelation geeignet sind. (Abbildung 5.2)

Abbildung 5.2 – Grauwertbild, lokale Varianz über (7x7) Bildpunkte, Segmentierung von Bereichen mit hoher lokaler Varianz durch ein Schwellwertverfahren [Sch98]

Die Fortführung des Projektes wird darin bestehen, den Roboter selbständig zu befähigen, ein bestimmtes Ziel anzusteuern. Dazu muss ein Wegfindungsalgorithmus (voraussichtlich A*-

Algorithmus bzw. D*-Algorithmus oder Dijkstra Algorithmus; siehe auch [Kra05]) entwickelt werden, welcher mit den Daten der Positionsbestimmung verglichen wird. Es ergeben sich dadurch Soll-Ist-Informationen nach welchen das Fahrwerk der Plattform angesprochen werden muss (Lenkwinkel, Geschwindigkeit) um zu dem vorbestimmten Ziel zu navigieren. Weiter soll ein Verfahren zur Erkennung von im Einsatzumfeld angebrachten virtuellen oder künstlichen „Markierungsbojen“ entwickelt werden, an dem sich der Roboter neu absolut ausrichten kann um Differenzen in der Wegfindungsroutine auszugleichen.

E Anhang

E.1. Webcam Programm

E.1.1. Klassendiagramm

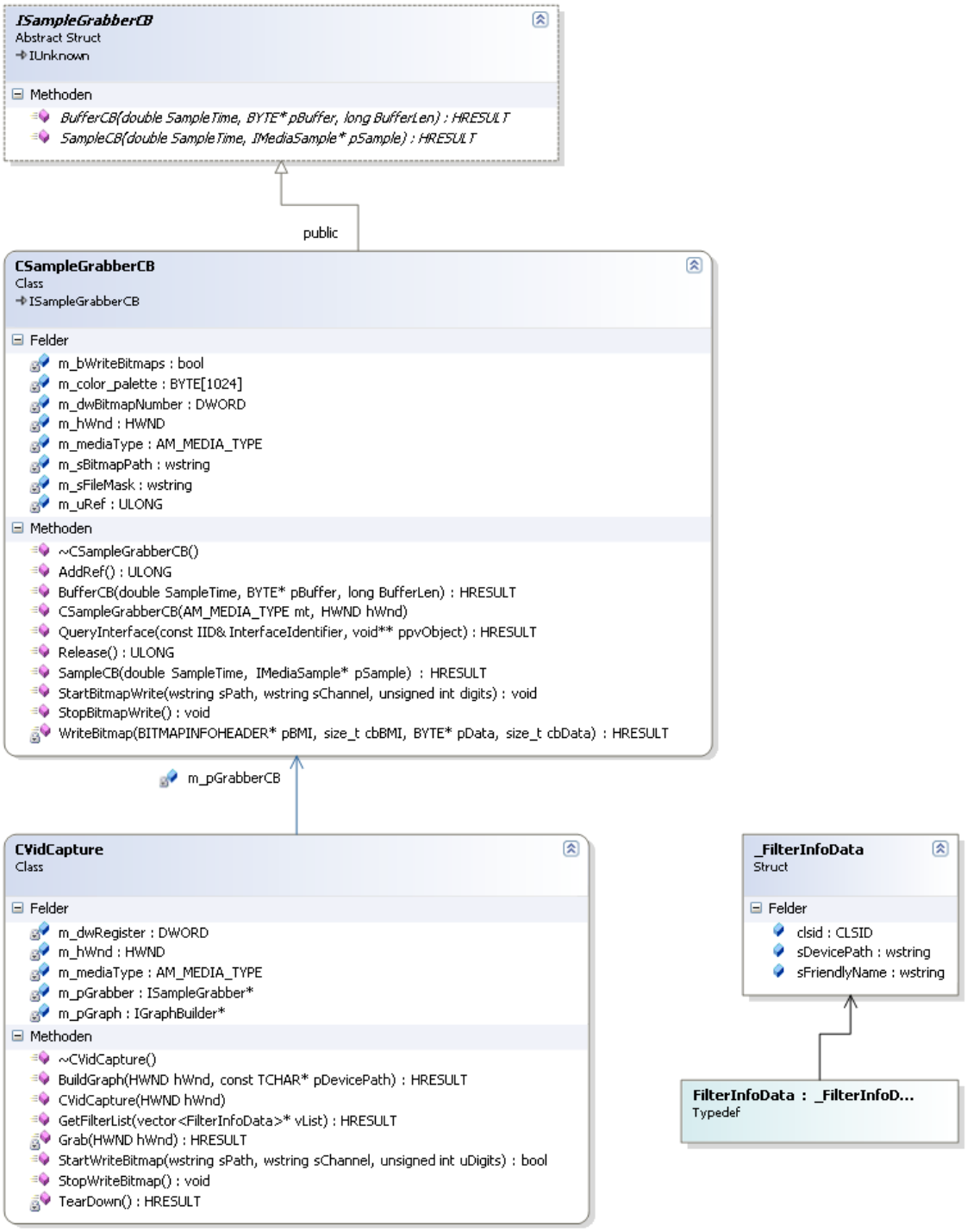
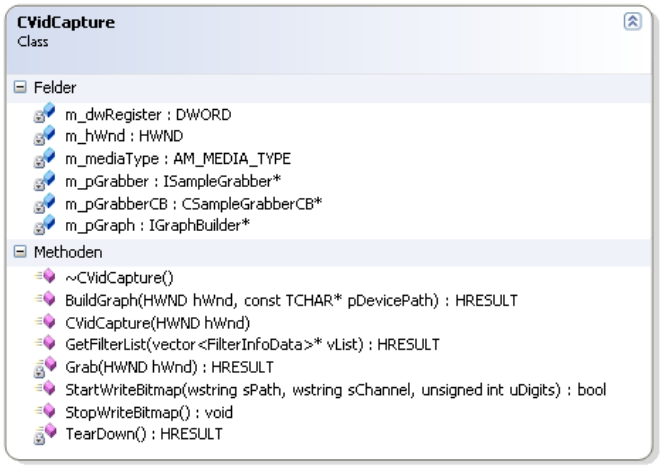


Abbildung E.1 - Klassendiagramm Webcam Programm

E.1.2. Dokumentation



Erstellt einen Graphen für einen angegebenen Gerätepfad einer WDM-Webacam und initialisiert den Samplegrabber.

`CVidCapture(HWND hWnd)`

Erzeugt Instanz des VidCapture-Objektes.

Parameter:

hWnd: Fensterhandle

`~CVidCapture()`

Zerstört den Graphen und Grabber und gibt die Instanz frei.

`BuildGraph(HWND hWnd, const TCHAR* pDevicePath): HRESULT`

Baut einen Graphen für die Verwendung einer Webcam zusammen.

Parameter:

hWnd: Handle zu einem Control zur Anzeige des Webcam Datenstromes.
pDevicePath: Gerätepfad zum WDM-Treiber der Webcam.

Rückgabe:

S_OK – Erfolgreich, VFW_E_XXX – Fehler.

`static GetFilterList(vector<FilterInfoData>* vList): HRESULT`

Erstellt eine Liste mit Filterdaten aller im System verfügbaren Webcams.

Parameter:

*vList: Zeiger auf einen Vector zur Speicherung der Filterdaten

Rückgabe:

S_OK – Erfogreich, S_FALSE – Erfolgreich, aber keine Geräte gefunden, VFW_E_XXX – Fehler

`StartWriteBitmap(wstring sPath, wstring sChannel, unsigned uDigits): bool`

Startet das Schreiben der Grauwertbilder auf einen Datenträger.

Parameter:

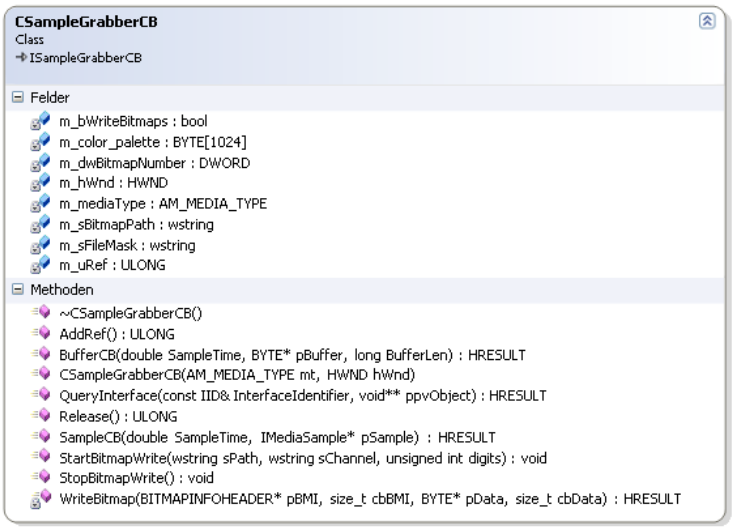
wstring sPath: Zielpfad
wstring sChannel: Kanal des Bilddateinamens
uDigits: Anzahl der Stellen der laufenden Nummer des Bilddateinamens.

Rückgabe:

true, wenn erfolgreich gestartet, sonst Fehler oder Speicherung der Bilder läuft bereits.

`StopWriteBitmap(): void`

Hält das Speichern der Bilder auf Datenträger an.



Implementierung des ISampleGrabberCB interfaces. Wandelt Bilder aus Datenstrom in 8-Bit RGB Grauwertbilder um und schreibt diese auf einen Datenträger.

`CSampleGrabberCB(AM_MEDIA_TYPE mt, HWND hwnd)`

Erstellt Objektinstanz.

Parameter:

mt: Medientypinformationen des Datenstromes

hWnd: Handle zum Control zur Anzeige des Datenstromes

`~CSampleGrabberCB()`

Beendet schreiben der Bilder und zerstört Objektinstanz.

`BufferCB(double SampleTime, BYTE* pBuffer, long BufferLen): HRESULT`

Implementierung der Interfacemethode. Ruft Methoden zum Schreiben, umwandeln der Bilder und Anzeige des Datenstromes auf. Diese Methode wird vom SamplerGrabber-Filter automatisch aufgerufen und sollte nie manuell aufgerufen werden.

`SampleCB(double SampleTime, IMediaSample* pSample): HRESULT`

Implementation der Interface Methode. Nicht verwendet.

`StartBitmapWrite(wstring sPath, wstring sChannel, unsigned digits): void`

Startet das schreiben und umwandeln der Bilder auf einen Datenträger.

Parameter:

sPath: Zielpfad

sChannel: Kanal des Bilddateinamens

digits: Anzahl der Stellen der laufenden Nummer des Bilddateinamens.

`StopWriteBitmap(): void`

Hält das Schreiben der Bilder an.

E.1.3. Quellcodeauszüge

- CSampleGrabberCB

```
HRESULT CSampleGrabberCB::BufferCB
   (double SampleTime, BYTE *pBuffer, long BufferLen)
{
   HRESULT hr = S_OK;
   if ((m_mediaType.formattype == FORMAT_VideoInfo) &&
      (m_mediaType.cbFormat >= sizeof(VIDEOINFOHEADER)) &&
      (m_mediaType.pbFormat != NULL))
   {
      VIDEOINFOHEADER *pVih = (VIDEOINFOHEADER*)m_mediaType.pbFormat;
      // Draw every second DIB only
      static BYTE a;
      a = (1 - a) % 2;
      if (a)
      {
         BITMAPINFO bmi = {0};
         bmi.bmiHeader = pVih->bmiHeader;
         // Create temporary device context
         HDC hDC = GetDC(m_hWnd);
         RECT ClientRect = {0};
         GetClientRect(m_hWnd, &ClientRect);
         HDC hMemDC = CreateCompatibleDC(hDC);

         // Create DIB
         BYTE *pBits;
         HBITMAP hBitmap = CreateDIBSection(0, &bmi, DIB_RGB_COLORS,
            (void**)&pBits, NULL, 0);
         CopyMemory(pBits, pBuffer, BufferLen);
         HBITMAP hOldBitmap = (HBITMAP)SelectObject(hMemDC, hBitmap);

         // Draw DIB
         SetStretchBltMode(hDC, STRETCH_HALFTONE);
         StretchBlt(hDC, ClientRect.left, ClientRect.top,
            ClientRect.right, ClientRect.bottom, hMemDC, 0, 0,
            bmi.bmiHeader.biWidth, bmi.bmiHeader.biHeight, SRCCOPY);

         // Free memory
         SelectObject(hMemDC, hOldBitmap);
         DeleteDC(hMemDC);
         DeleteObject(hBitmap);
      }

      // Write Bitmap
      if (m_bWriteBitmaps)
      {
         hr = WriteBitmap(&pVih->bmiHeader, m_mediaType.cbFormat -
            SIZE_PREHEADER, pBuffer, BufferLen);
      }
   }
   return hr;
}
```

Listing E.1 – CSampleGrabberCB::BufferCB: Zeichnet Sample und ruft WriteBitmap() auf

```
HRESULT CSampleGrabberCB::WriteBitmap(BITMAPINFOHEADER *pBMI, size_t
  cbBMI, BYTE *pData, size_t cbData)
{
  if (pBMI->biBitCount < 8) return E_FAIL;
  HRESULT hr = S_OK;
  BYTE cbColor = pBMI->biBitCount / 8;
  size_t cbSize = cbData / cbColor;
  TCHAR szFile[MAX_PATH];
  TCHAR szPath[MAX_PATH];

  StringCchPrintf(szFile, MAX_PATH, m_sFileMask.c_str(),
    m_dwBitmapNumber++);
  PathCombine(szPath, m_sBitmapPath.c_str(), szFile);
  FILE *pFile;
  errno_t e = _tfopen_s(&pFile, szPath, _T("wb"));
  if (e == 0) {
    // Write Fileheader
    BITMAPFILEHEADER bmf = {0};
    bmf.bfType = (WORD)'MB';
    bmf.bfSize = cbBMI + sizeof(bmf) + cbSize + SIZE_COLOR_PALETTE;
    bmf.bfOffBits = cbBMI + sizeof(bmf) + SIZE_COLOR_PALETTE;
    fwrite(&bmf, sizeof(bmf), 1, pFile);

    // Write InfoHeader
    BITMAPINFOHEADER bmi;
    CopyMemory(&bmi, pBMI, cbBMI);
    bmi.biBitCount = 8;
    bmi.biSizeImage = 0;
    bmi.biClrUsed = 256;
    bmi.biClrImportant = 256;
    fwrite(&bmi, cbBMI, 1, pFile);

    // Write color palette
    fwrite(m_color_palette, sizeof(m_color_palette), 1, pFile);

    // Calculate and write Grayscale
    if (pBMI->biBitCount > 8) {
      BYTE *pGrayData = (BYTE*)malloc(cbSize);
      for (size_t i = 0; i < cbSize; i++)
      {
        int gray = 0;
        for (int j = 0; j < cbColor; j++)
        {
          gray += pData[i*cbColor+j];
        }
        pGrayData[i] = gray / 3;
      }
      fwrite(pGrayData, cbSize, 1, pFile);
      free(pGrayData);
    } else {
      fwrite(pData, cbSize, 1, pFile);
    }
    fclose(pFile);
  } else {
    hr = HRESULT_FROM_WIN32(e);
  }
  return hr;
}
```

Listing E.2 – CSampleGrabberCB::WriteBitmap: Konvertiert Farbwerte in Graustufen und speichert sie als 8bit Grayscale Bitmap

- CVidCapture

```
HRESULT CVidCapture::BuildGraph(HWND hWnd, const TCHAR *pDevicePath)
{
  TearDown();
  HRESULT hr = S_OK;
  // Create Filtergraph and Capturegraph Object
  hr = CoCreateInstance(CLSID_FilterGraph, NULL,
    CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&m_pGraph));

  // Init Capturegraph-Builder
  ICaptureGraphBuilder2 *pCapture;
  hr = CoCreateInstance(CLSID_CaptureGraphBuilder2 , NULL,
    CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pCapture));
  hr = pCapture->SetFiltergraph(m_pGraph);

  // Create and Init Source, Grabber and Null Renderer Filter
  IBaseFilter *pGrabberF;
  hr = AddFilter(m_pGraph, CLSID_SampleGrabber,
    _T("Sample Grabber"), &pGrabberF);
  hr = pGrabberF->QueryInterface(IID_PPV_ARGS(&m_pGrabber));

  // Init source Mediatype
  AM_MEDIA_TYPE mt = {0};
  mt.majortype = MEDIATYPE_Video;
  mt.subtype = MEDIASUBTYPE_RGB24;
  hr = m_pGrabber->SetMediaType(&mt);
  hr = m_pGrabber->SetOneShot(FALSE);
  hr = m_pGrabber->SetBufferSamples(TRUE);

  // Init Sourcefilter
  IBaseFilter *pSource = NULL;
  TCHAR szFilterName[MAX_PATH];
  hr = GetFilter(AM_KSCATEGORY_CAPTURE, pDevicePath, szFilterName,
    &pSource);
  hr = m_pGraph->AddFilter(pSource, szFilterName);

  // Init Null-Renderer
  IBaseFilter *pNULL;
  hr = AddFilter(m_pGraph, CLSID_NullRenderer, _T("Null Renderer"),
    &pNULL);

  // Render and run Graph
  hr = pCapture->RenderStream(NULL, NULL, pSource, pGrabberF,
    pNULL);
  IMediaControl *pControl;

  // Get Control Interface and try to run the Graph
  hr = m_pGraph->QueryInterface(IID_PPV_ARGS(&pControl));
  hr = pControl->Run();
  if (hr == S_FALSE)
  {
    // If least one filter not Ready, wait and try run again
    FILTER_STATE fs;
    hr = pControl->GetState(INFINITE, (OAFilterState*)&fs);
    if (SUCCEEDED(hr) && fs == State_Running)
      hr = pControl->Run();
  }

  // Get full Media informations and register Sample Grabber Class
  hr = m_pGrabber->GetConnectedMediaType(&m_mediaType);
```

```
   hr = Grab(hWnd);

   // Free COM objects
   pControl->Release();
   pNULL->Release();
   pSource->Release();
   pGrabberF->Release();
   pCapture->Release();
   return hr;
}
```

Listing E.3 – CVidCapture::BuildGraph: Erstellt Filtergraphen (ohne Prüfung der Rückgabewerte[6])

```
HRESULT CVidCapture::Grab(HWND hWnd)
{
   HRESULT hr = S_FALSE;
   if (m_pGrabber)
   {
      if (!m_pGrabberCB)
      {
         m_pGrabberCB = new CSampleGrabberCB(m_mediaType, hWnd);
         m_pGrabber->SetCallback(m_pGrabberCB, 1);
      }
      else
      {
         m_pGrabber->SetCallback(NULL, 1);
         delete m_pGrabberCB;
         m_pGrabberCB = NULL;
      }
      hr = S_OK;
   }
   return hr;
}
```

Listing E.4 – CVidCapture::Grab: (De)Registriert ein SampleGrabberCB Objekt bei Grabber-Filter

[6] Rückgabewerte sind DirectShow `HRESULT` - Fehlercodes, welche mit den Makros `SUCCEEDED` oder `FAILED` geprüft werden. Aus Gründen der Übersichtlichkeit sind diese Abfragen entfernt worden, aber im Quellcode vorhanden.

- DShowHelpers.cpp

Die DShowHelpers.cpp stellt einige Hilfsfunktionen zur Erstellung, Verbinden und Finden von Filtern zu Verfügung.

```
HRESULT GetFilter(const CLSID deviceClass, const TCHAR *pDevicePath,
   TCHAR *pFilterName, IBaseFilter **pFilter)
{
   ICreateDevEnum *pDevEnum = NULL;
   IEnumMoniker *pEnum = NULL;

   // Create the System Device Enumerator.
   HRESULT hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
      CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pDevEnum));

   // Create an enumerator for the category.
   hr = pDevEnum->CreateClassEnumerator(deviceClass, &pEnum, 0);
   if (hr == S_OK) // S_FALSE means succeeded but nothing to enum!
   {
      IMoniker *pMoniker = NULL;
      while (pEnum->Next(1, &pMoniker, NULL) == S_OK)
      {
         IPropertyBag *pPropBag = NULL;
         hr = pMoniker->BindToStorage(0, 0,
            IID_PPV_ARGS(&pPropBag));

         // Find the description or friendly name.
         VARIANT varName;
         VariantInit(&varName);
         hr = pPropBag->Read(_T("DevicePath"), &varName, 0);
         if (_tcscmp(varName.bstrVal, pDevicePath) == 0)
         {
            hr = pMoniker->BindToObject(0, 0,
               IID_PPV_ARGS(pFilter));
            VariantClear(&varName);
            hr = pPropBag->Read(_T("FriendlyName"), &varName,
               0);
            StringCchCopy(pFilterName, MAX_PATH,
               varName.bstrVal);
            pPropBag->Release();
            pMoniker->Release();
            VariantClear(&varName);
            pPropBag->Release();
            pMoniker->Release();
            break;
         }
         pEnum->Release();
      }
      pDevEnum->Release();
   }
   return hr;
}
```

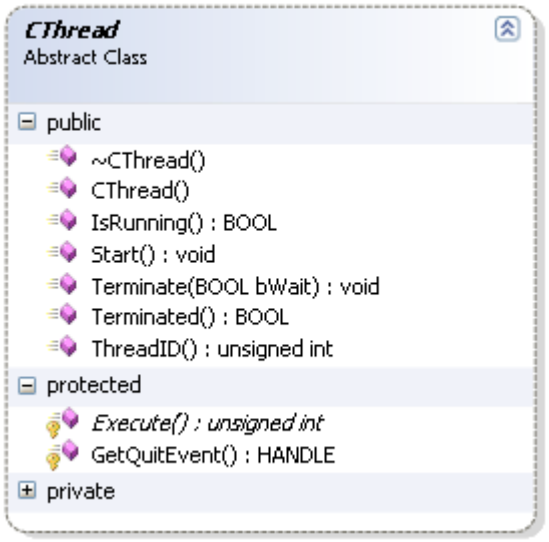
Listing E.5 – GetFilter: Erstellt ein Filter Objekt aus einem angegebenen Gerätepfad und Geräteklasse (ohne Prüfung von Rückgabewerten)

E.2. Programm *CorroPos*

E.2.1. Klassendiagramm

Abbildung E.2 – Klassendiagramm *CorroPos*

E.2.2. Dokumentation



Abstrakte Klasse welche ermöglicht, separate Threads in einer Anwendung Objektorientiert zu erstellen.

`CThread()`	Erzeugt eine Instanz des Threadobjektes.
`virtual ~CThread()`	Gibt Threadobjekt frei, setzt Terminate-Event und wartet auf Beendigung der Execute() Methode, falls erforderlich.
`IsRunning(): BOOL`	Prüft ob Thread läuft.
	Rückgabe: true, Thread läuft, sonst läuft nicht.
`Start()`	Startet die Codeausführung der Execute() Methode.
`Terminate(BOOL bWait = true)`	Setzt Event, dass der Thread sobald wie möglich beendet werden sollte. **Anmerkung:** Im Gegensatz zur Windows-API-Funktion Terminate(), die den Thread sofort abbricht, fordert die Methode Terminate() nur an, dass der Thread beendet wird. So kann der Thread vor Beendigung noch sämtliche Bereinigungen durchführen.
	Parameter: bWait: true (default), Methode kehrt erst zurück, wenn Thread beendet ist, sonst kehrt sie sofort nach setzen des Terminate-Events zurück.
`Terminated(): BOOL`	Prüft ob Thread im Abbruch Status ist. (Terminate-Event gesetzt).
	Rückgabe: true, Thread ist am Beenden, sonst Abbruchevent ist nicht gesetzt.
`ThreadID(): unsigned int`	Liefert Thread ID zurück.
	Rückgabe: ThreadID.
`Execute(): unsigned int`	Stellt eine abstrakte bzw. rein virtuelle Methode bereit, die den auszuführenden Code enthält. Execute prüft die Rückgabewert der Methode Terminated() und ermittelt dadurch, ob der Thread beendet werden muss. **Anmerkung:** Der Code wird mit der Methode Start() ausgeführt. Rufen Sie die Methode Execute() niemals direkt auf.
	Rückgabe: Benutzerdefinierter Rückgabewert.

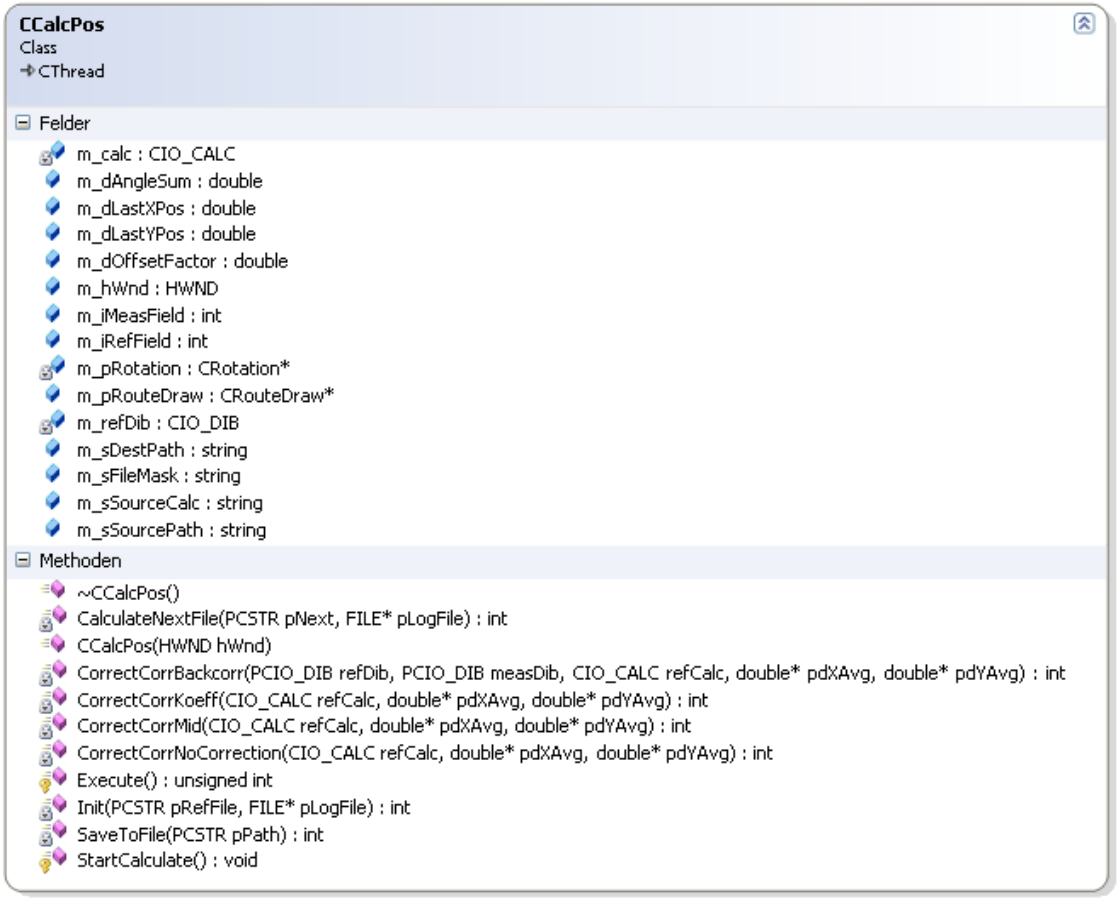
GetQuitEvent(): HANDLE	Gibt das Handle des Terminate-Events zurück.
	Rückgabe: Terminate-Event Handle.

Beispiel:

```
// overrided Execute()-Method from CThread
unsigned CMyThread::Execute(void)
{
   // Check user want to terminate
   while (!Terminated())
   {
      doSomethingNeedLongTime(m_iAMember, m_sAOtherMember);
   }
   return 0;
}

int Main()
{
   CMyThread *myThread = new CMyThread();
   myThread->m_iAMember = 1234;
   myThread->m_sAOtherMember = "this is a string";
   myThread->Start();

   WaitForSingleObject(myThread->GetQuitEvent(), INFINITE);
   delete myThread;
}
```



Hauptklasse des Korrelationsalgorithmus. Steuert die Verarbeitung aller Korrelationsdaten und

Parametern. Berechnung blockiert nicht den Hauptthread

Felder:

m_dAngleSum: Derzeitiger Winkel.
m_dlastXPos, m_dlastYPos: Derzeitige Koordinate im Weltkoordinatensystem.
m_dOffsetFactor: Angabe des Offsetfaktors.
m_hWnd: Fensterhandle.
m_iMeasField: setzt oder liest die Größe des Messfeldes.
m_iRefField: setzt oder liest die Größe des Referenzfeldes.
m_pRouteDraw: Zeiger auf CRoutDraw.
m_sDestPath: Zielpfad der Speicherdateien und Logdateien.
m_sFileMask: Dateinamenmaske. Beispiel: „A%.7d.bmp“.
m_sSourceCalc: liest oder setzt die Messpunktdatei.
m_sSourcePath: liest oder setzt den Quellpfad der Bilder.

`CCalcPos(HWND hWnd)`
Erzeugt Instanz und initialisiert Felder mit 0 bzw. Standardwerten

`~CCalcPos()`
Bereinigt und Zerstört Instanz.

`Execute(): unsigned`
Implementation der abstrakten Methode. Führt Korrelationsberechnung aus. Wird Automatisch aufgerufen und sollte nicht manuell gestartet werden. Zum Start der Korrelationberechnungen `Start()` der abgeleiteten CThread Klasse verwenden.

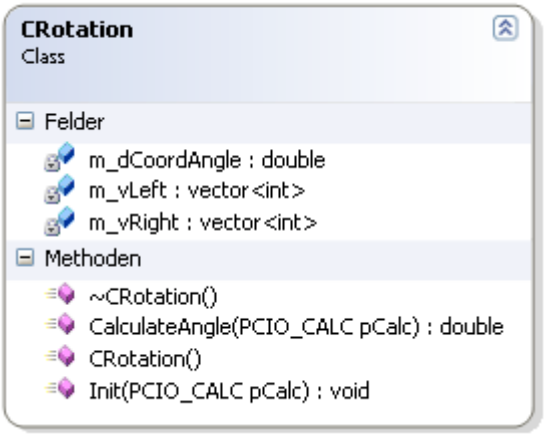
`StartCalculate(): void`
Startet die Korrelationsberechnungen mit den angegebenen Parametern im „Felder“-Abschnitt als Thread.

Beispiel:

```
void CMainCtrl::Start()
{
  CCalcPos *m_pCalcPos = new CCalcPos(m_hWnd);

  m_pCalcPos->m_sSourcePath = "C:\\temp";
  m_pCalcPos->m_sDestPath = "C:\\temp\\out";
  m_pCalcPos->m_sSourceCalc = "C:\\temp\\meas.vdc";
  m_pCalcPos->m_sFileMask = "A%.7d.bmp";
  m_pCalcPos->m_dOffsetFactor = 0.5;
  m_pCalcPos->m_iRefField = 30;
  m_pCalcPos->m_iMeasField = 50;

  m_pCalcPos->Start();
}
```



Berechnet Rotationswinkel aus gegebenen Korrelationsdaten

`CRotation()`
Instanziiert das Objekt.

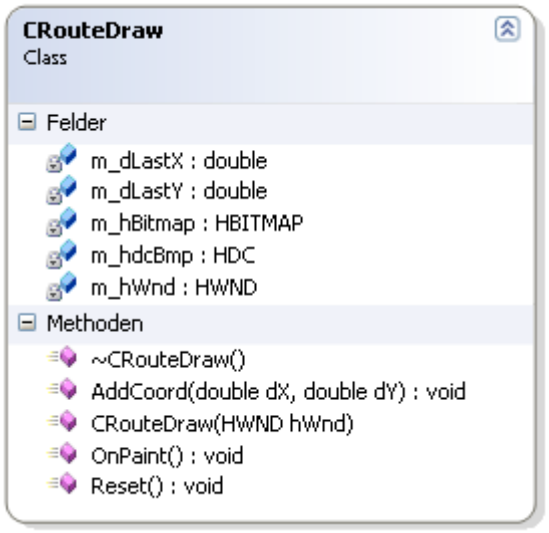
`~CRotation()`
Zerstört das Objekt.

`Init(PCIO_CALC pCalc): void`
Berechnet die Verteilung der Messpunkthälften und den Startwinkel. Muss vor der ersten Verwendung der Methode CalculateAngle() aufgerufen werden.

Parameter:
pCalc: Zeiger auf Berechnungsstruktur

`CalculateAngle(PCIO_CALC_ pCalc): void`
Berechnet den Rotationswinkel aus Verschiebungswerten.

Parameter:
pCalc: Zeiger auf Berechnungsstruktur mit den Verschiebungswerten.



Erstellt ein Memory Bitmap auf den übergebene Koordinaten skaliert als Weg gezeichnet werden.

`CRouteDraw(HWND hWnd)`
Instanziiert das Objekt.

Parameter:
hWnd: Handle auf ein Control auf dem gezeichnet werden soll.

`~CRouteDraw()`
Zerstört das Objekt.

`Reset(): void`
Löscht den gezeichneten Weg.

`AddCoord(double dX, double dY): void`
Setzt den nächsten Koordinatenpunkt zum Zeichnen des Weges.

Parameter:
dX: Koordinatenpunkt X
dY: Koordinatenpunkt Y

`OnPaint(): void`
Zeichnet das Bitmap auf ein Control. Sollte beim Eintreten der WM_PAINT Message aufgerufen werden.

E.2.3. Quellcodeauszüge

- CCalcPos

```
void CCalcPos::StartCalculate()
{
  int iImage = FIRST_IMAGE;
  char szImage[MAX_PATH];
  char szImagePath[MAX_PATH];
  bool bIsInitialized = false;

  HANDLE hNotify =
    FindFirstChangeNotification(m_sSourcePath.c_str(), false,
    FILE_NOTIFY_CHANGE_LAST_WRITE);
  HANDLE hEvents[2] = {GetQuitEvent(), hNotify};

  if (hNotify != INVALID_HANDLE_VALUE)
  {
    // Logdatei anlegen
    FILE *pLogFile;
    if (fopen_s(&pLogFile,
      generateLogFilename(m_sDestPath.c_str()), "w") == 0)
    {
      // Dateimaske erstellen
      string sFileMask(m_sFileMask + FILE_EXTENSION);
      StringCchPrintf(szImage, MAX_PATH, sFileMask.c_str(),
        iImage);
      PathCombine(szImagePath, m_sSourcePath.c_str(),
        szImage);

  while (!Terminated())
  {
    // Berechnung initialisieren
    int res;
    if (bIsInitialized)
    {
      res = CalculateNextFile(szImagePath, pLogFile);
      if (res == CIO_SUCCESS)
      {
        char szOutFile[MAX_PATH];
        char szOutPath[MAX_PATH];
        char szBuf[MAX_PATH];

        MESSAGE(m_hWnd, WM_OFFSET_PROGRESS, iImage);
        StringCchPrintf(szBuf, MAX_PATH, SAVEFILE_MASK,
          m_sFileMask.c_str(), m_sFileMask.c_str());
        StringCchPrintf(szOutFile, MAX_PATH, szBuf,
          iImage-1, iImage);
        PathCombine(szOutPath, m_sDestPath.c_str(),
          szOutFile);
        SaveToFile(szOutPath);
      }
    }
    else
    {
      res = Init(szImagePath, pLogFile);
      bIsInitialized = (res == CIO_SUCCESS);
    }
```

```
      switch (res)
      {
      case CIO_SUCCESS: // Create next Image Filename
        StringCchPrintf(szImage, MAX_PATH, sFileMask.c_str(),
          ++iImage);
        PathCombine(szImagePath, m_sSourcePath.c_str(), szImage);
        break;
      case CIO_ERROR_FILE_LOAD: // Wait for new File
        DWORD e;
        e = GetLastError();
        if (e == ERROR_FILE_NOT_FOUND || e ==
          ERROR_SHARING_VIOLATION)
        {
          MESSAGE(m_hWnd, WM_OFFSET_WAIT, iImage);
          WaitForMultipleObjects(2, hEvents, false,
            INFINITE);
          FindNextChangeNotification(hNotify);
          break;
        } // else fall through
        default: // Critical Error
        MESSAGE(m_hWnd, WM_OFFSET_ERROR_CALC, res);
        break;
      }
    }

        fclose(pLogFile);
      } else MESSAGE(m_hWnd, WM_OFFSET_ERROR, GetLastError());
      FindCloseChangeNotification(hNotify);
    }

    MESSAGE(m_hWnd, WM_OFFSET_THREADEND, 0);
}
```

Listing E.6 – Steuert Korrelationsalgorithmus, sendet Informationsnachrichten an Hauptthread.

```
int CCalcPos::Init(PCSTR pRefFile, FILE *pLogFile)
{
  m_dAngleSum = 0.0;
  m_dLastXPos = 0.0;
  m_dLastYPos = 0.0;
  m_calc.CorrParam.dXShiftOffset = 0.0;
  m_calc.CorrParam.dYShiftOffset = 0.0;
  m_pRouteDraw->Reset();

  int res;
  if ((res = cioLoadCalc(&m_calc, m_sSourceCalc.c_str(),
    CIO_CF_CORR_DATA)) == CIO_SUCCESS)
  {
    if (m_calc.CorrParam.iNumberData <= MIN_NUMBER_MEASPOINTS)
    {
      SetLastError(ERROR_INVALID_DATA);
      res = CIO_ERROR_INVALID_CALC_STRUCT;
    }
    else
    {
      // CorrParam setzen
      StringCchCopy(m_calc.pMeasImage, MAX_PATH, pRefFile);
      m_calc.CorrParam.iRefField = m_iRefField;
      m_calc.CorrParam.iMeasField = m_iMeasField;
      m_calc.CorrParam.iCorrAlg = CA_CORRECT;
      m_calc.CorrParam.iSubPixAlg = SP_PARABOL;
      if ((res = cioLoadDib(&m_refDib, pRefFile)) == CIO_SUCCESS)
      {
        fprintf(pLogFile, LOG_SETTINGS_MASK,
          m_calc.CorrParam.iRefField,
          m_calc.CorrParam.iMeasField,
          m_dOffsetFactor);
        fprintf(pLogFile, LOG_HEADER_MASK);
      }

      m_pRotation->Init(&m_calc);
    }
  }
  return res;
}
```

Listing E.7 – Initialisierung des ersten Bildes der Sequenz

```
int CCalcPos::CalculateNextFile(PCSTR pNext, FILE *pLogFile)
{
   int res;
   CIO_DIB measDib = {0};
   if ((res = cioLoadDib(&measDib, pNext)) == CIO_SUCCESS) {
      CK_IMAGE_DATA img = {0};
      if ((res = cioGetImageData(&img, &m_refDib, &measDib)) ==
         CIO_SUCCESS) {
         m_calc.CorrParam.iMeasField = m_iMeasField;
         char szError[MAX_PATH] = {0};

do {
   if ((res = CalcCorr(&img, &m_calc.CorrParam, m_calc.pCorrData))
      == CK_SUCCESS)
   {
   // Offset auf Verschiebungen addieren
   for (int i = 0; i < m_calc.CorrParam.iNumberData; i++)
   {
      m_calc.pCorrData[i].dXShift += m_calc.CorrParam.dXShiftOffset;
      m_calc.pCorrData[i].dYShift += m_calc.CorrParam.dYShiftOffset;
   }

   double dXAverShift;
   double dYAverShift;

   // Auswahl der Korrekturmethoden derzeit nur im Code
   //int n = CorrectCorrBackcorr(&m_refDib, &measDib, m_calc,
   //&dXAverShift, &dYAverShift);
   int n = CorrectCorrNoCorrection(m_calc, &dXAverShift,
      &dYAverShift);
   //int n = CorrectCorrKoeff(m_calc, &dXAverShift, &dYAverShift);
   //int n = CorrectCorrMid(m_calc, &dXAverShift, &dYAverShift);

   // weniger als 50% der Messpunkte korrekt. -> Neuberechnung mit
   // größeren Messfeld sonst zu ungenau.
   // oder max Messfeld erreicht -> berechnete Werte verwenden
   if ((n > (int)(m_calc.CorrParam.iNumberData * 0.5)) ||
      (m_calc.CorrParam.iMeasField >= 4 * m_iMeasField)) {

      // Offset berechnen
      m_calc.CorrParam.dXShiftOffset += m_dOffsetFactor *
         (dXAverShift - m_calc.CorrParam.dXShiftOffset);
      m_calc.CorrParam.dYShiftOffset += m_dOffsetFactor *
         (dYAverShift - m_calc.CorrParam.dYShiftOffset);

      // Winkel berechnen
      double dPhi = m_pRotation->CalculateAngle(&m_calc);
      m_dAngleSum += dPhi;

      // Koordinaten berechnen
      double dHypo = sqrt(dXAverShift * dXAverShift + dYAverShift *
         dYAverShift);
      m_dLastXPos += dHypo * cos(m_dAngleSum);
      m_dLastYPos += dHypo * sin(m_dAngleSum);

      // Neue Koordinate anzeigen
      m_pRouteDraw->AddCoord(m_dLastXPos, m_dLastYPos);

      // Logline schreiben
      fprintf(pLogFile, LOG_LINE_MASK, PathFindFileName(pNext),
```

```
        m_calc.CorrParam.dXShiftOffset,
        m_calc.CorrParam.dYShiftOffset, dXAverShift,
        dYAverShift, dPhi, m_dLastXPos, m_dLastYPos,
        m_dAngleSum, szError);
      StringCchCopy(m_calc.pRefImage, MAX_PATH, m_calc.pMeasImage);
      StringCchCopy(m_calc.pMeasImage, MAX_PATH, pNext);
    } else {
      res = CK_CALC_CORREL_FAILED;

      // Bei Fehler Messfeld vergrößern und neu berechnen
      if (m_calc.CorrParam.iMeasField >= 4 * m_iMeasField)
        break;
      m_calc.CorrParam.iMeasField *= 2;
      StringCchPrintf(szError, MAX_PATH, LOG_ERROR_MASK,
        m_calc.CorrParam.iMeasField);
    }

    }
  } while (res != CIO_SUCCESS);

        cioFreeImageData(&img);
      }
      cioFreeDib(&m_refDib);
      memcpy(&m_refDib, &measDib, sizeof(measDib));
    }
    return res;
  }
```

Listing E.8 – Hauptteil des Korrelationsalgorithmus

- CRotation

```
void CRotation::Init(PCIO_CALC pCalc)
{
  // Mittelpunkt, Breite und Höhe der Punktwolke berechnen
  double dXAvgCoord = 0.0;
  double dYAvgCoord = 0.0;
  double xMin, xMax, yMin, yMax;
  int iNum = 0;
  for (int i = 0; i < pCalc->CorrParam.iNumberData; i++)
  {
    // ignorieren eliminierter Vektoren
    if (pCalc->pCorrData[i].dXShift != 0.0 &&
      pCalc->pCorrData[i].dYShift != 0.0)
    {
      if (iNum == 0)
      {
        xMin = pCalc->pCorrData[i].dXCoord;
        xMax = pCalc->pCorrData[i].dXCoord;
        yMin = pCalc->pCorrData[i].dYCoord;
        yMax = pCalc->pCorrData[i].dYCoord;
      } else {
        if (pCalc->pCorrData[i].dXCoord < xMin)
          xMin = pCalc->pCorrData[i].dXCoord;
        if (pCalc->pCorrData[i].dXCoord > xMax)
          xMax = pCalc->pCorrData[i].dXCoord;
        if (pCalc->pCorrData[i].dYCoord < yMin)
          yMin = pCalc->pCorrData[i].dYCoord;
        if (pCalc->pCorrData[i].dYCoord > yMax)
          yMax = pCalc->pCorrData[i].dYCoord;
      }
      dXAvgCoord += pCalc->pCorrData[i].dXCoord;
      dYAvgCoord += pCalc->pCorrData[i].dYCoord;
      iNum++;
    }
  }
  dXAvgCoord /= iNum;
  dYAvgCoord /= iNum;
  if (iNum > 0)
  {
    // Messpunkte aufteilen
    double dxl = 0.0;
    double dyl = 0.0;
    double dxr = 0.0;
    double dyr = 0.0;
    m_vLeft.clear();
    m_vRight.clear();
    if (xMax - xMin > yMax - yMin)
    {
      for (int i = 0; i < pCalc->CorrParam.iNumberData; i++)
      {
        if (pCalc->pCorrData[i].dXShift != 0.0 &&
          pCalc->pCorrData[i].dYShift != 0.0)
          if(pCalc->pCorrData[i].dXCoord<dXAvgCoord)
          {
            m_vLeft.push_back(i);
            dxl += pCalc->pCorrData[i].dXCoord;
            dyl += pCalc->pCorrData[i].dYCoord;
          } else {
            m_vRight.push_back(i);
```

```
                dxr += pCalc->pCorrData[i].dXCoord;
                dyr += pCalc->pCorrData[i].dYCoord;
              }
            }
          } else {
            for (int i = 0; i < pCalc->CorrParam.iNumberData; i++)
            {
              if (pCalc->pCorrData[i].dXShift != 0.0 &&
                pCalc->pCorrData[i].dYShift != 0.0)
                if(pCalc->pCorrData[i].dYCoord<dYAvgCoord)
                {
                  m_vLeft.push_back(i);
                  dxl += pCalc->pCorrData[i].dXCoord;
                  dyl += pCalc->pCorrData[i].dYCoord;
                } else {
                  m_vRight.push_back(i);
                  dxr += pCalc->pCorrData[i].dXCoord;
                  dyr += pCalc->pCorrData[i].dYCoord;
                }
            }
          }

          // Ausgangswinkel
          m_dCoordAngle = atan2(dyr / m_vRight.size() - dyl /
            m_vLeft.size(), dxr / m_vRight.size() - dxl /
            m_vLeft.size());
        }
      }
```

Listing E.9 – Initialisierung der Rotationswinkelberechnung

```
double CRotation::CalculateAngle(PCIO_CALC pCalc)
{
  double dXAvgL = 0.0;
  double dYAvgL = 0.0;
  for (SIZE_T i = 0; i < m_vLeft.size(); i++)
  {
    dXAvgL += pCalc->pCorrData[m_vLeft[i]].dXCoord +
      pCalc->pCorrData[m_vLeft[i]].dXShift;
    dYAvgL += pCalc->pCorrData[m_vLeft[i]].dYCoord +
      pCalc->pCorrData[m_vLeft[i]].dYShift;
  }

  double dXAvgR = 0.0;
  double dYAvgR = 0.0;
  for (SIZE_T i = 0; i < m_vRight.size(); i++)
  {
    dXAvgR += pCalc->pCorrData[m_vRight[i]].dXCoord +
      pCalc->pCorrData[m_vRight[i]].dXShift;
    dYAvgR += pCalc->pCorrData[m_vRight[i]].dYCoord +
      pCalc->pCorrData[m_vRight[i]].dYShift;
  }

  return m_dCoordAngle - atan2(dYAvgR / m_vRight.size() - dYAvgL /
    m_vLeft.size(), dXAvgR / m_vRight.size() - dXAvgL /
    m_vLeft.size());
}
```

Listing E.10 – Berechnung des Rotationswinkels

- CRouteDraw

```
void CRouteDraw::AddCoord(double dX, double dY)
{
  // Skalierung berechnen
  RECT r = {0};
  GetClientRect(GetDlgItem(m_hWnd, IDC_PIC), &r);
  const int offsetX = DEFAULT_SCALE;
  const int offsetY = (int)((double)r.bottom / (double)r.right *
    (double)offsetX);

  double dScaleX = ((dX + offsetX / 2) * r.right) / offsetX;
  double dScaleY = ((dY + offsetY / 2) * r.bottom) / offsetY;

  // Neue Linie vom letzten Punkt aus zeichnen
  HPEN hPen = CreatePen(PS_SOLID, 1, RGB(255,0,0));
  SelectObject(m_hdcBmp, hPen);
  MoveToEx(m_hdcBmp, (int)m_dLastX, (int)m_dLastY, NULL);
  LineTo(m_hdcBmp, (int)dScaleX, (int)dScaleY);
  DeleteObject(hPen);

  m_dLastX = dScaleX;
  m_dLastY = dScaleY;

  // Änderung der Oberfläche mitteilen
  InvalidateRect(m_hWnd, &r, false);
}
```

Listing E.11 – Zeichenroutine für zurückgelegten Weg

F Literaturverzeichnis

[CWM10] Chemnitzer Werkstoffmechanik GmbH: http://www.cwm-chemnitz.de, 21.09.2010

[Gut99] GUTMANN, J.-S.: *Robuste Navigation autonomer mobiler Systeme* – 1999, Dissertation, Universität Freiburg

[Gut96] GUTMANN, J.-S.: Vergleich *von Algorithmen zur Selbstlokalisierung eines mobilen Roboters* – 1996, Diplomarbeit, Universität Ulm

[Haa08] HAASE, S.: *Mathematische Aspekte der Grauwertkorrelation* – 2008, Erster Zwischenbericht im Rahmen der Berufspraktischen Ausbildung, Hochschule Mittweida, Fachbereich MPI, Mittweida

[Haa09a] HAASE, S: *Depth-From-Focus* – 2009, Zweiter Zwischenbericht im Rahmen der Berufspraktischen Ausbildung, Hochschule Mittweida, Fachbereich MPI, Mittweida

[Haa09b] HAASE, S: *Mathematische Formulierung von Objektverformungen* – 2009, Diplomarbeit, Hochschule Mittweida, Fachbereich MPI, Mittweida

[Kra05] KRASCHL, G.: *Routenplanung für mobile Roboter* – 2005, Magisterarbeit, Technische Universität Graz, Institut für Navigation und Satellitengeodäsie

[Lau10] LAUENSTEIN, T.: *Praktikumsbeleg* – 2010, Hochschule Mittweida, Fachbereich MNI, Mittweida

[MSD10] Microsoft Developer Network: *DirectShow System Overview (Windows),* http://msdn.microsoft.com/en-us/library/dd375470.aspx, 21.09.2010

[Sch98] SCHNITGER, T; HANDMANN, U.: *Fusion von Bildanalyseverfahren mittels einer neuronalen Kopplungsstruktur* – 1998, Ruhr-Universität, Institut für Neuroinformatik, Bochum.

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Chemnitz, den 21.09.2010