



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Semesterarbeit Digitale Medien

Relative Bewegungsschätzung für Roboternavigation mittels Videokamera



Studenten

Dani Hofmann
Christian Lutz

Betreuer

Prof. Dr. Guido M. Schuster

Ergänzungen / Änderungen im Dokument

| Version | Datum | Seite / Kapitel | Ergänzungen / Änderungen | Wer |
|---------|----------|---------------------------------------------------------|-------------------------------------------------------|---------|
| V1.0 | 11.12.03 | Zwischendoku | Aufbau auf Zwischendoku | DH |
| V1-1 | 08.02.03 | Theoretische Grundlagen | Rotation | CL |
| V1-2 | 11.02.03 | Praktische Grundlagen | USB WebCam / WirelessCam / MFC | CL / DH |
| V1-3 | 12.02.03 | Realisierung | sämtliche Unterkapitel überarbeitet oder neu erstellt | CL / DH |
| V1-4 | 13.02.03 | Optimieren der Parameter Schlussfolgerungen Fazit | Schlussfolgerungen von Fazit separiert | CL / DH |
| V1-5 | 15.02.03 | Anhang | Verzeichnisse ergänzt | CL / DH |

| | |
|----------------------|----------------------------------------------------------------------------------|
| Autor: | Dani Hofmann, Christian Lutz |
| Dokumentname: | Schlussdokumentation der Semesterarbeit |
| Version: | 1-5 |
| Dateiname: | DokuV1-5.doc |
| Status: | <i>freigegeben</i> [geplant, in Bearbeitung, vorgelegt, akzeptiert, freigegeben] |
| Verteiler | |

Inhaltsverzeichnis

| | | |
|----------|-------------------------------------------------|-----------|
| 1 | ABSTRACT..... | 5 |
| 2 | AUFGABENSTELLUNG..... | 6 |
| 3 | EINFÜHRUNG | 7 |
| 4 | KONZEPT | 7 |
| | 4.1 Übersicht | 7 |
| | 4.2 Algorithmus..... | 8 |
| | 4.3 GUI | 8 |
| | 4.4 Traking..... | 9 |
| 5 | THEORETISCHE GRUNDLAGEN..... | 10 |
| | 5.1 Motion Estimation..... | 10 |
| | 5.2 Diskrete Cosinus-Tranformation (DCT)..... | 11 |
| | 5.3 Kreuzkorrelation | 12 |
| | 5.4 Schwellwertschalter / Objektsuche | 13 |
| | 5.5 Bereichskorrelation | 13 |
| | 5.6 Rotation | 15 |
| | 5.6.1 Nearest-neighbour Interpolation | 17 |
| | 5.6.2 Bilineare Interpolation..... | 19 |
| | 5.7 Downsampling | 20 |
| 6 | PRAKTISCHE GRUNDLAGEN | 21 |
| | 6.1 MatLab | 21 |
| | 6.1.1 MEX-Schnittstelle | 22 |
| | 6.1.2 WebCam unter MatLab | 23 |
| | 6.2 USB WebCam | 23 |
| | 6.3 WirelessCam | 24 |
| | 6.3.1 Konfiguration der WirelessCam..... | 24 |
| | 6.4 MSVC++ | 26 |
| | 6.4.1 Bilder einlesen mittels USB WebCam | 26 |
| | 6.4.2 Bilder einlesen mittels WirelessCam | 26 |
| | 6.4.3 Zeitmessung..... | 26 |
| | 6.4.4 MFC..... | 27 |
| 7 | REALISIERUNG | 30 |
| | 7.1 GUI | 30 |
| | 7.1.1 Verschiebungslinie | 31 |
| | 7.1.2 Pfeil | 32 |
| | 7.2 Algorithmus..... | 34 |
| | 7.2.1 Downsampling..... | 34 |
| | 7.2.2 Abrunden..... | 34 |
| | 7.2.3 Rotation | 35 |



| | | |
|------------|--------------------------------------|-----------|
| 7.2.4 | Bereichskorrelation..... | 37 |
| 7.3 | Fehlerkorrektur | 38 |
| 7.4 | Skalierung..... | 38 |
| 7.5 | Fahrzeug..... | 39 |
| 7.6 | Spannungsregler | 39 |
| 7.6.1 | Schema | 40 |
| 7.6.2 | Beschreibung | 40 |
| 7.7 | Programmstruktur | 41 |
| 8 | OPTIMIEREN DER PARAMETER..... | 42 |
| 9 | ZEITPLAN..... | 43 |
| 10 | SCHLUSSFOLGERUNGEN | 44 |
| 11 | PERSÖNLICHES FAZIT | 44 |
| 12 | ANHANG..... | 45 |
| 12.1 | Abbildungsverzeichnis | 45 |
| 12.2 | Formelverzeichnis..... | 46 |
| 12.3 | Tabellenverzeichnis | 46 |
| 12.4 | Quellenverzeichnis..... | 46 |
| 12.5 | Arbeitsmittel | 46 |
| 12.6 | Glossar | 47 |

1 Abstract

Im Rahmen dieser Studienarbeit wurde gezeigt, dass es möglich ist, die Position eines Roboters zu bestimmen, mittels einer WebCam, die senkrecht zur Decke schaut. Das Projekt wurde in einer Windowsumgebung (Windows 2000/XP) realisiert. Die auf dem Roboter oder einem anderen Gefährt montierte WebCam, liefert laufend Bilder der Decke über WLAN. Der Computer empfängt diese Bilder und berechnet die Verschiebung der einzelnen Frames. Dies ergibt die relative Bewegung der WebCam. Die Werte der Berechnung, sowie der zurückgelegte Weg werden im GUI visualisiert. Die berechnete Bewegung enthält die Translation in x- und y- Richtung, sowie den Rotationswinkel in Grad. Die Berechnung der Translation beruht auf der Einheit Pixel, d.h. es können Bewegungen von einem Pixel detektiert werden. Da der Faktor cm/Pixel von der Distanz zwischen der Kamera und der Decke abhängig ist, beträgt in unserer Versuchsanordnung die kleinste detektierbare Bewegung des Roboters etwa 1cm. Die Genauigkeit der Rotation beträgt 2° . Dieser Wert ist ein Kompromiss zwischen Nutzen und Rechenaufwand. Eine Berechnung zwischen zwei Frames beträgt ca. 30ms. Das Projekt umfasste die Erarbeitung der Grundlagen in Bildverarbeitung, den verwendeten Werkzeugen, die Entwicklung des Algorithmus, den Bau einer Versuchsanordnung und die Visualisierung der Berechnungen.

2 Aufgabenstellung

Thema: Relative Bewegungsschätzung für Roboternavigation mittels einer Videokamera

Studenten: Daniel Hofmann und Christian Lutz
Betreuer: Prof. Dr. Guido M. Schuster

Kurzbeschreibung

Das Problem das es in dieser SA zu lösen gilt, ist dies der Positionsbestimmung eines Roboters mittels einer Kamera, welche auf dem Roboter fix montiert ist. Die Kamera ist so montiert, dass sie direkt nach oben schaut, und damit ein Bild der Decke erhält. Wenn sich der Roboter bewegt, dann ändert sich das Videobild der Kamera. Anhand dieser Änderung ist die relative Bewegung des Roboters zu bestimmen. Der Roboter kann sich Vorwärts, Rückwärts, nach Links und nach Rechts bewegen. Erschwerend kommt hinzu, dass der Roboter sich auch um seine eigene Achse drehen kann. Als Resultat der Bewegungsschätzung wird die relative Position $(x(t), y(t))$ und der relative Winkel $(\alpha(t))$ erwartet.

Aufgabenstellung

- Erarbeitung und Darstellung der erforderlichen Grundlagen
- Erarbeitung eines Algorithmus mit VC++ und einer WirelessCam
- (Inbetriebnahme der Daughtercard und der mitgelieferten Software)
- Echtzeitimplementierung der Positionserkennung

Erwartete Ergebnisse

- Dokumentation der Grundlagen
- Dokumentation des Problems und der Lösung(en)
- C/C++ Implementierung des Algorithmus
- Dokumentation der Implementierungen

Arbeitsweise

Sie führen ein persönliches Laborbuch, wo Sie aufschreiben wann Sie was für wie lange machen und was die Ergebnisse sind.

Sie schicken mir vor jedem Treffen eine Agenda und nach dem Treffen ein Protokoll.

3 Einführung

Die HSR nimmt jedes Jahr an den EUROBOT¹-Wettkämpfen teil. Deshalb wurden an dieser Schule schon viele Roboter für diese Einsätze entwickelt und gebaut. Eines der grossen Probleme, welches jedes Team zu lösen hatte, ist die Positionsbestimmung, die für die Navigation gebraucht wird. Der Roboter wird von einem definierten Ort gestartet. Nach dem Startzeichen muss der Roboter autonom die Aufgaben lösen. Ab diesem Zeitpunkt ist der Roboter auf sich alleine gestellt. Die Positionsbestimmung wurde bis anhin häufig mittels Inkrementalzähler an den Rädern bestimmt. Diese Zähler weisen jedoch einen Fehler auf, der sich mit jeder Bewegung aufaddiert. Der Fehler ist insbesondere dann gravierend, wenn der Roboter durch den Gegner geschubst wird oder in ein Hindernis fährt. Daraus ist die Idee entstanden, die Position des Roboters optisch zu bestimmen. In dieser Semesterarbeit wird nun untersucht, ob es möglich ist, die Positionsbestimmung mit einer WebCam zu realisieren.

4 Konzept

4.1 Übersicht

Folgende Darstellung gibt einen Überblick über die gesamte Arbeit:

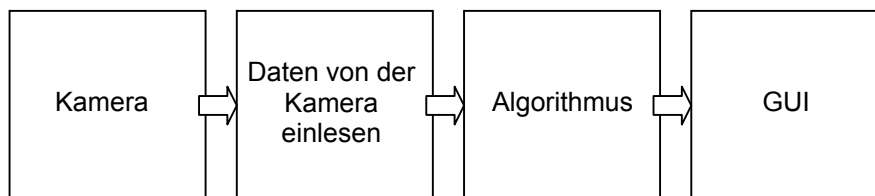


Abbildung 4.1 Grobkonzept

Unsere Arbeit gliedert sich in vier Teilbereiche.

Zu Beginn wurde mit einer einfachen WebCam herumexperimentiert, um Erfahrungen zu sammeln. Später hatten wir die WirelessCam zur Verfügung, die für unser Projekt gedacht ist. Da die Bilder in unterschiedlicher Weise von den Kameras kommen, muss das Einlesen in einem separaten Block geschehen. Der Block Algorithmus soll so realisiert werden, dass er unabhängig von den anderen Modulen ist, damit er möglichst einfach auf einen DSP portiert werden kann.

Die Übergänge von Block zu Block bilden jeweils klar definierte Schnittstellen. Dem Block Algorithmus werden die Bilddaten übergeben und dem Block GUI den Verschiebungsvektor und den Winkel.

¹ Eurobot: www.robotik.com
Swissbot: www.robocup.ch

4.2 Algorithmus

An den Algorithmus stellen wir folgende Anforderungen:

- Unser Positionierungssystem soll autonom sein. Die Berechnungen sollen deshalb ohne jegliche feed-backs des Roboters auskommen.
- Der Algorithmus soll möglichst robust sein. Das heisst, er muss möglichst einfach gehalten werden. Denn je komplexer die Methoden sind, desto eher sind sie anfällig auf Störungen.
- Der Algorithmus soll immer gleich schnell sein, egal wie die Situation gerade ist.
- Sowohl der PC als auch der DSP² haben nur beschränkte Rechenleistungen. Abhängig von der Framerate besteht ein Zeitfenster zwischen 20ms und 200ms, um die Verschiebung und Rotation zu berechnen und darzustellen. Die Berechnungsmethoden müssen also sehr schnell und einfach sein. Die optimale Framerate und Komplexität des Algorithmus müssen wahrscheinlich experimentell ermittelt werden.

4.3 GUI

Das GUI ist ein Dialogfeld aus der MFC³. In der linken Hälfte ist das Livebild der Kamera zu sehen. In der rechten Hälfte sieht man, wie sich der Roboter relativ zu seinem Startpunkt bewegt. Der Roboter wird als Pfeil dargestellt. Dieser Pfeil zeigt stets in die Richtung in die der Roboter `schaut`. Seine Fahrrichtung muss also nicht mit seiner Winkelposition übereinstimmen, da er sich an Ort drehen kann. Weiter gibt es noch ein paar Buttons mit denen man die Aufzeichnung beginnen, stoppen und löschen kann. Das GUI wird ungefähr wie auf der Abbildung 4.2 aussehen:

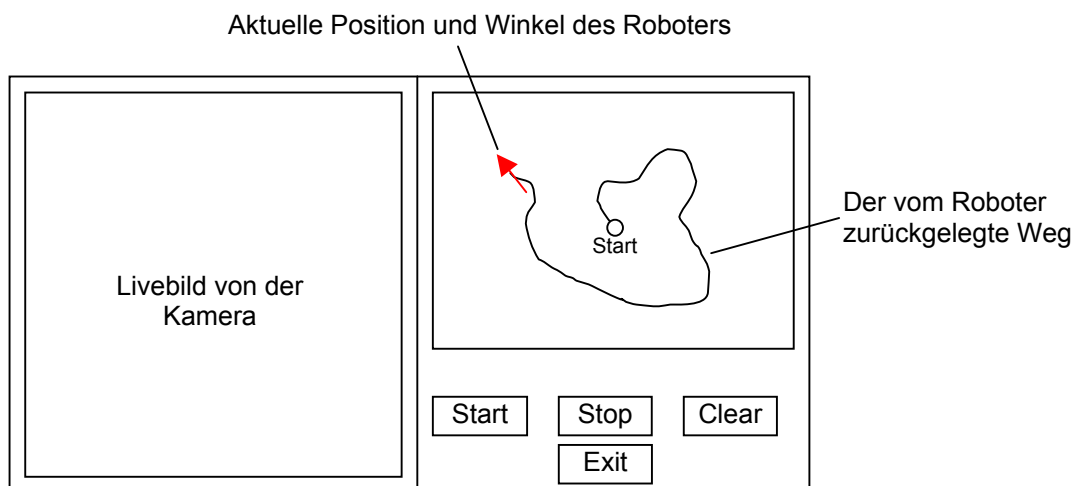


Abbildung 4.2 GUI

² DSP: Digital Signal Processor

³ MFC: Microsoft Foundation Class

Als Option besteht die Idee, nach einer bestimmten Zeit die Linie von hinten her wieder zu löschen. So kann ein Fadengewirr verhindert werden.

4.4 Traking

Der Umstand, dass der Roboter nicht innerhalb weniger Millisekunden an einem völlig anderen Ort sein kann, könnte für uns noch nützlich sein. Es ist möglich, ein dynamisches Modell des Roboters zu erzeugen. Mit diesem Modell kann mit einer bestimmten Wahrscheinlichkeit vorausgesagt werden, wo sich der Roboter im nächsten Augenblick befinden wird.

Mit dem Traking könnte man beispielsweise einen kurzen Verbindungsunterbruch der WirelessCam mit dem PC überbrücken oder die geschätzte zukünftige Position, mit der vom Algorithmus berechneten, vergleichen. Wenn die Resultate stark voneinander abweichen, erkennt man, dass etwas nicht stimmt. So könnten Fehler vermieden werden.

Der Nachteil dieser Methode ist der zusätzliche Rechenaufwand, der den Rahmen des gegebenen Zeitfensters höchstwahrscheinlich sprengen würde.

5 Theoretische Grundlagen

In der ersten Phase der Semesterarbeit, wurden die erforderlichen Grundlagen erarbeitet. Für die erste Informationsbeschaffung, wurden Bücher, das Internet und Diplomanden herbeigezogen. Soweit dies zu diesem Zeitpunkt schon möglich war, wurde entschieden, welche Themen für den Gebrauch in dieser Arbeit tauglich sind und welche nicht.

5.1 Motion Estimation

Bei diesem Thema geht es darum, Bewegungen von einem Bild zum nächsten zu detektieren und den Verschiebungsvektor zu berechnen. Hierzu wird meistens das Originalbild in Makroblöcke zerteilt. Nun sucht man im zweiten Bild nach den Blöcken vom ersten Bild. Dieses Verfahren wird Blockmatching genannt. Die Bewegungen zwischen zwei Frames sind in der Regel gering. Deshalb kann man den Suchbereich einschränken. In unserem Fall rechnen wir mit einer maximalen Geschwindigkeit des Roboters von 1m/s. Nehmen wir nun an, dass die Kamera 25 Bilder/s schießt, dann ergibt dies eine maximale Verschiebung S_{\max} von 4 cm. Die allgemeine Formel lautet:

$$S_{\max} = v \cdot \Delta t = v \cdot \frac{1}{F} \quad v : \text{Geschwindigkeit} \quad F : \text{Framerate}$$

Formel 5.1 max Verschiebung

Der Verschiebungsvektor V lässt sich mit folgender Formel berechnen:

$$V = [i - x, j - y]$$

Formel 5.2 Verschiebungsvektor

Der Referenzblock kann nun also maximal um S_{\max} in jede Richtung verschoben sein und muss deshalb auch nur in diesem Bereich gesucht werden (siehe Abbildung 5.1).

Falls die Änderung zwischen zwei Bildern zu gering ist, d.h. wenn die Verschiebung kleiner als die Breite eines Pixels ist, dann kommt die Long-Term Motion Estimation zum Zuge. In diesem Fall wird das Frame zum Zeitpunkt t mit dem Frame zum Zeitpunkt $t+n$ verglichen, wobei n das n te Frame ist, bei dem eine Verschiebung festgestellt wurde.

Blockmatching ist in unserem Fall keine ideale Lösung. Da sich die Kamera bewegt und nicht die gefilmten Objekte, verschieben sich alle Pixel. Würde man die Verschiebung aller Makroblöcke berechnen, wäre der Rechenaufwand dafür viel zu gross. Nämte man nur einzelne Makroblöcke, wäre das Verfahren weniger robust.

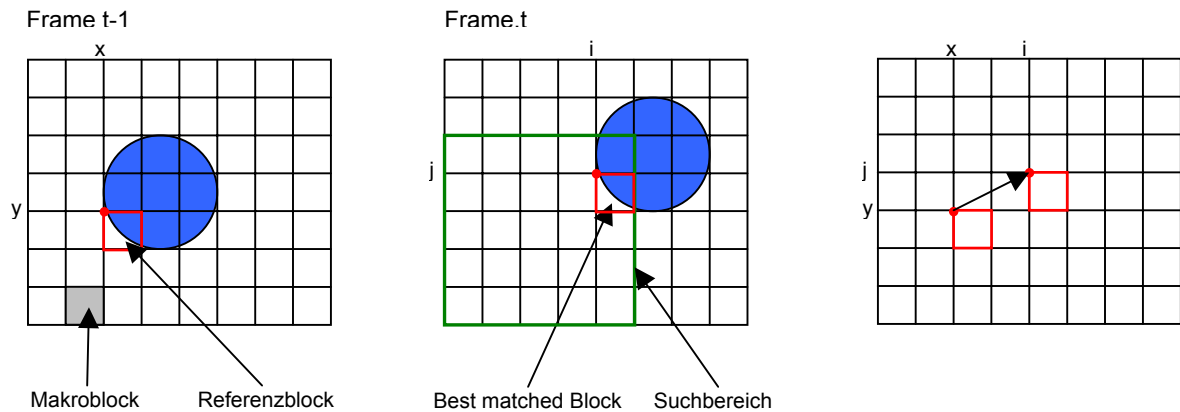


Abbildung 5.1 Motion Estimation

5.2 Diskrete Cosinus-Transformation (DCT)

Die DCT kann sowohl zum Komprimieren von Bildern, als auch zum Ermitteln von scharfen Kanten verwendet werden. Zunächst wird das Bildmaterial in Blöcke von beispielsweise 8x8 Pixel unterteilt. In jedem Block wandelt die Diskrete Cosinus-Transformation die enthaltenen Helligkeits- und Farbinformationen in Frequenzen um, die den Helligkeits- und Farbverlauf widerspiegeln. Dabei schlagen sich einerseits Flächen mit feinen Details, die das menschliche Auge kaum wahrnimmt, andererseits auch abrupte Farbwechsel (Kanten), in hohen Frequenzen nieder. Beim Komprimieren filtert die anschließende Quantisierung diese hohen Frequenzen heraus und reduziert so drastisch die Datenmenge. Ist man an den Kanten und Linien des Bildes interessiert, konzentriert man sich auf die hohen Frequenzen.

Wir haben mit bestehenden Filtern von MatLab experimentiert und diverse Kantenbilder erzeugt. Wenn man Pech hat und das Bild zu geringe Farbunterschiede aufweist, dann ist das Kantenbild einfach schwarz. Dies kommt bei Bildern von Decken ziemlich häufig vor. Aus diesem Grund wird die DCT nicht verwendet.

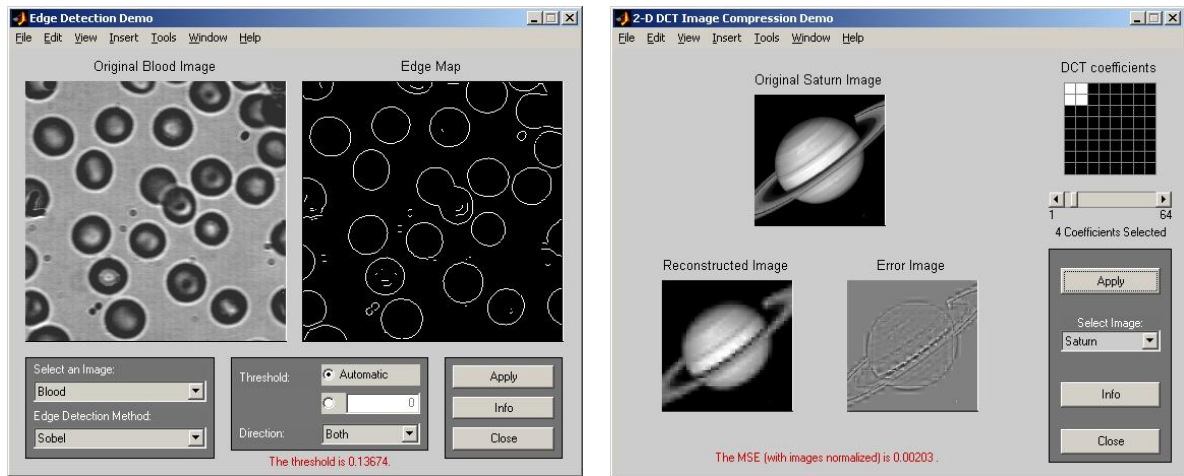


Abbildung 5.2 DCT-Demo: links Kanten, rechts Komprimierung⁴

5.3 Kreuzkorrelation

In der Welt der elektrischen Signale benutzt man die Kreuzkorrelationsfunktion um zwei Signale auf Ähnlichkeit zu testen. Sind beide Signale identisch, ist die Kreuzkorrelation identisch mit der Autokorrelation, welche ihr Maximum bei 0 hat. Ist das eine der beiden gleichen Signale zeitverschoben um T , verschiebt sich die Kreuzkorrelationsfunktion und hat ihr Maximum nun bei T .

$$R_{xy} = \int_{-\infty}^{\infty} x(t) \bullet y(t + \tau) dt$$

Formel 5.3 Korrelationsfunktion

Von der WebCam kommen in regelmässigen Zeitabständen Bilder, welche sich nur sehr gering voneinander unterscheiden. Zwei aufeinanderfolgende Bilder sind bis auf den Randbereich identisch. Das eine ist verschoben zum anderen. Mit diesen beiden Bildern wird nun eine 2D-Kreuzkorrelation berechnet. Die Verschiebung des Maximums zum Ursprung (0,0) ergibt die Verschiebung der zwei Bilder. Natürlich braucht es für diese Berechnung eine 2-Dimensionale-Kreuzkorrelation.

Die Kreuzkorrelation ist sehr langsam, da sehr viele Rechenoperationen nötig sind. Versuche mit MatLab haben ergeben, dass die Kreuzkorrelation so nicht brauchbar ist. Bessere Rechenzeiten wurden erreicht, indem die Kreuzkorrelationsfunktion in C geschrieben und von MatLab aus die Funktion aufgerufen wurde.

Folgende Rechenzeiten haben sich nun für die Kreuzkorrelation ergeben:

Zwei Bilder mit je 100x100 Pixel brauchen um die 300ms, 50x50 Pixel nur noch 30ms (Testsystem: Intel Pentium IV, 2.6GHz, 512MB RAM).

⁴ Demoprogramme bereits in MatLab integriert

5.4 Schwellwertschalter / Objektsuche

Um diese enorme Rechenzeit bei der Kreuzkorrelation zu minimieren, muss das Bild verkleinert werden. Der Ausschnitt muss aber so gewählt werden, dass nicht nur die weisse Decke darin vorkommt. Es sollte mindestens ein Objekt im Bildausschnitt vorkommen. Damit ergibt sich aber ein neues Problem, nämlich die der Objektsuche. Mittels eines Schwellwertschalters wird ein 8-Bit-Graustufen Bild zu einem s/w-Bild konvertiert. In diesem s/w-Bild muss jetzt ein Objekt gesucht werden. Um dieses Objekt wird dann einen Ausschnitt gelegt, welcher mit dem Folgebild korreliert wird. Die Probleme bestehen hauptsächlich darin, im Bild ein Objekt zu suchen und was zu machen ist, wenn kein Objekt gefunden wird. Da gleichzeitig die Idee mit der Bereichskorrelation entstanden ist, wurde diese Idee verworfen, weil sie wegen den Fallunterscheidungen komplizierter ist.



Abbildung 5.3 Schwellwertschalter: links Originalbild, rechts s/w-Bild

5.5 Bereichskorrelation

Da die Verschiebung von Bild zu Bild nur sehr gering ist, muss gar nicht die ganze Kreuzkorrelationsfunktion berechnet werden. Es interessieren nur jene Punkte um den Mittelpunkt der Ergebnismatrix. Mit einem Bereichsfenster wurde die Rechenzeit nochmals verkürzt. Ein Bild mit 100x100 Pixel mit einem Bereichsfenster von 20x20 Pixel ergibt eine Rechenzeit von 40ms. Die Grösse des Bereichsfensters sollte so gross wie möglich gehalten werden. Deshalb wird die endgültige Grösse erst am Schluss bestimmbar sein. Es werden immer noch die gleichen Bilder miteinander verglichen. Es wird kein Ausschnitt aus den Bildern herausgeschnitten, sondern es werden nur jene Rechenoperationen weggelassen, welche wegen der geringen Änderung von Bild zu Bild sowieso keine Rolle spielen. Das Maximum der Kreuzkorrelation wird immer um den Mittelpunkt zu finden sein. Das untere Beispiel zeigt die komplette Kreuzkorrelation zwischen zwei 3x3 Matrizen. Bei jedem Schritt, wird die grüne Matrix um eine Position nach rechts geschoben. Ist die äusserste Überlappung ganz rechts erreicht, wird die grüne Matrix eine Position nach unten verschoben und fängt ganz links wieder an. Bei jedem Schritt werden die zwei sich genau überlappenden Positionen multipliziert und zu den anderen Produkten dieser Überlappung addiert. Diese Summe wird nun

an die gleiche Position der Ergebnismatrix geschrieben, in welcher der Wert ausgerechnet wurde.

Bei der Bereichskreuzkorrelation werden nur diejenigen Werte innerhalb des roten Rechtecks ausgerechnet.

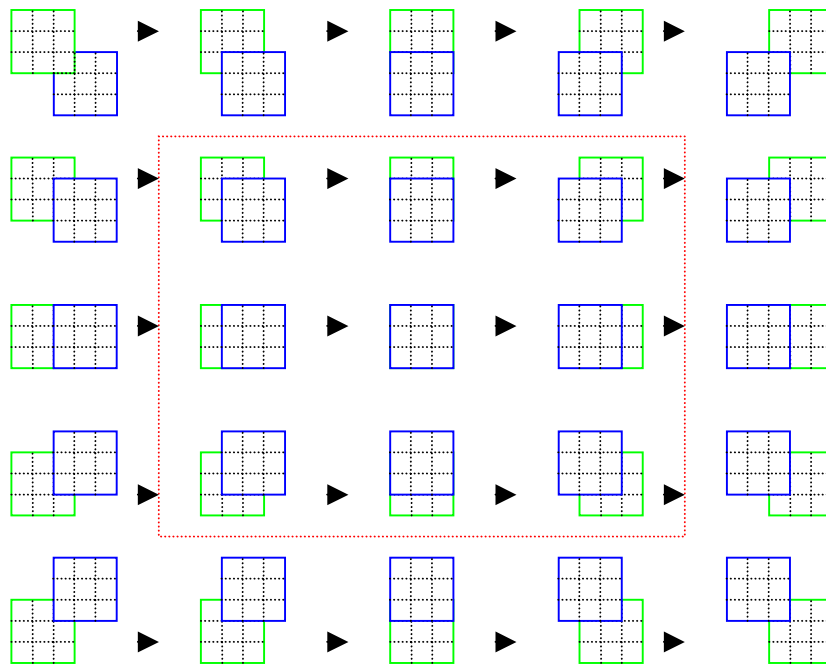


Abbildung 5.4 Bereichskorrelation

Die Minimierung des Rechenaufwandes ist bei zwei 3x3 Matrizen nicht gross. Bei zwei 100x100 Matrizen mit einem Bereichfenster von 20x20 sieht man den Gewinn viel besser. In der Ergebnismatrix sind die Dimensionen immer doppelt so gross, wie in den Ausgangsmatrizen. Aus diesem Grund ist das Bereichsfenster in dem untenstehenden Bild auch 40x40.

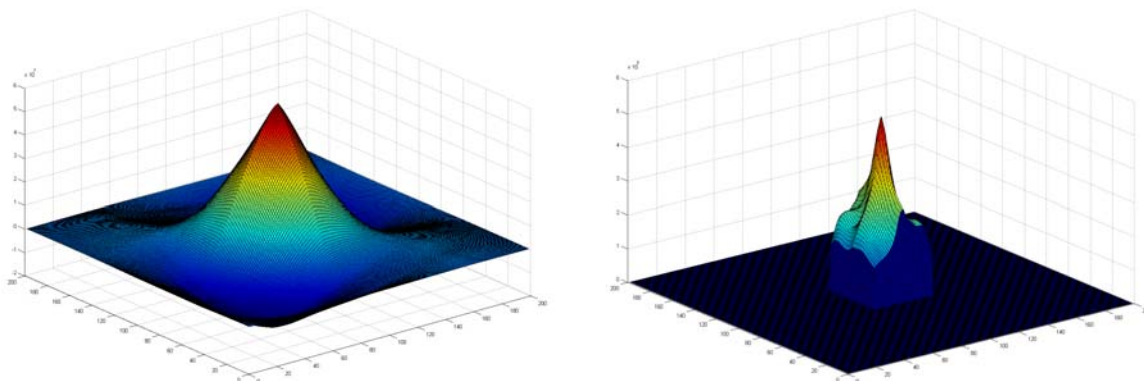


Abbildung 5.5 Korrelation und Bereichskorrelation

5.6 Rotation

Neben der Translation, kann sich die WebCam respektive der Roboter auch drehen. Da sich der Roboter von Bild zu Bild nur geringfügig dreht, muss daher nur ein kleiner Winkel betrachtet beziehungsweise berechnet werden.

Ein Bild der WebCam muss somit nur mit wenigen Winkeln gedreht und mit dem vorherigen Bild der WebCam korreliert werden. Die "beste" Kreuzkorrelation ergibt dann den Winkel, welcher der tatsächlichen Drehung am nächsten kommt. Die Translation wie auch die Rotation sind beide Bestandteile der affinen Transformation. Aus diesem Grund ist es naheliegend, beide Operationen gemeinsam zu benutzen. Genau diese Idee lässt sich auch auf die Kreuzkorrelation anwenden. Anstatt einer 2D- Berechnung, wird noch eine dritte Dimension eingeführt, die Drehung. Somit werden die Bilder nicht nur in der Translation korreliert, sondern auch noch in der Rotation. Die Ergebnismatrix ist natürlich jetzt 4-Dimensional. Jetzt muss noch das Maximum aus dieser Matrix gefunden werden. Die drei Koordinaten ergeben dann die Translation und Rotation.

Wenn man die maximale Winkelgeschwindigkeit des Roboters kennt, kann man den maximalen Drehwinkel, der von einem Bild zum andern auftreten kann, wie folgt berechnen:

$$\varphi = \omega \cdot \Delta t = \omega \frac{1}{F} \quad \omega : \text{Winkelgeschwindigkeit} \quad F : \text{Framerate}$$

Formel 5.4 Winkelberechnung

Eine andere Möglichkeit den Drehwinkel zu ermitteln, könnte mit zwei Bildblöcken der Grösse A x A Pixel erfolgen. Der eine Block befindet sich im Bildmittelpunkt und der andere möglichst am Rand des Bildes, um die grösstmögliche Genauigkeit zu gewährleisten. Dieser Block muss aber einen Mindestabstand vom Bildrand haben, damit er bei einer schnellen Bewegung nicht aus dem Bildbereich des Folgebildes verschwindet. Werden diese zwei Referenzblöcke im folgenden Frame lokalisiert, kann so relativ einfach der Drehwinkel berechnet werden (siehe Abbildung 5.6):

$$\omega = \tan\left(\frac{q-j}{p-i}\right)$$

Formel 5.5 Drehwinkel

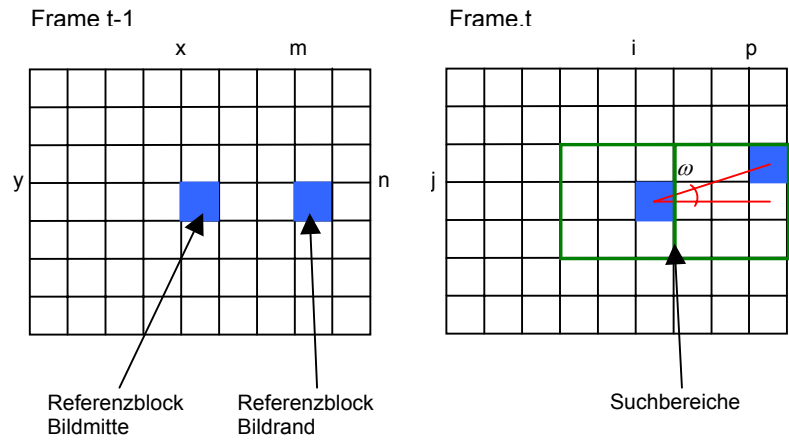


Abbildung 5.6 Winkelberechnung

Je nach Lösungsansatz muss man wissen, wie ein Bild, bestehend aus Pixel, gedreht werden kann.

Man stelle sich das Bild als eine zweidimensionale Matrix vor. Diese Matrix wird einer Zahlenebene gleichgesetzt. Die Distanzen der einzelnen Mittelpunkte der Pixel zum Drehpunkt werden als Vektoren angesehen. Diese Vektoren können mit einer Rotationsmatrix mit einem beliebigen Winkel um den Drehpunkt an einen neuen Ort gedreht werden.

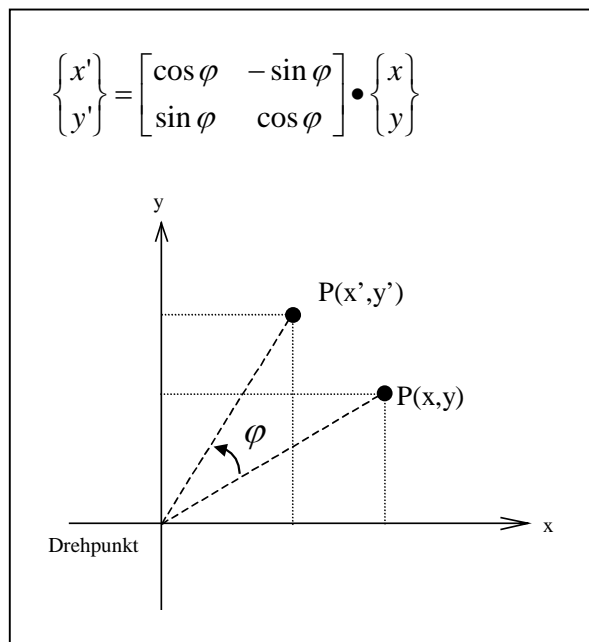


Abbildung 5.7 Rotationsmatrix

Es ergibt sich folgendes Problem:

Die Rasterpunkte (mögliche Positionen eines Pixel) in (x,y) werden nicht exakt auf Rasterpunkte in (x',y') abgebildet.

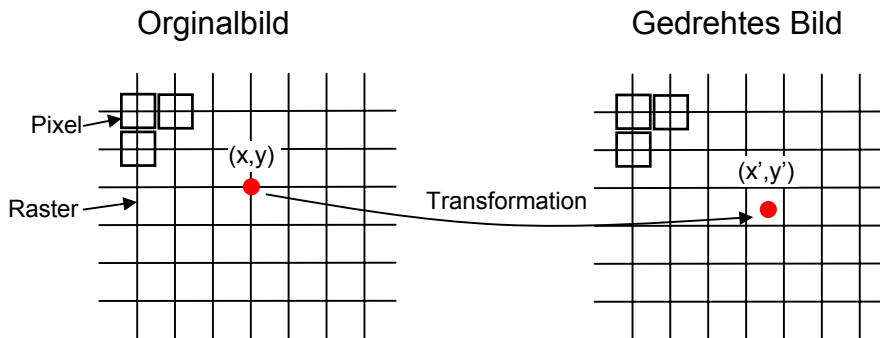


Abbildung 5.8 Rasterpunkte

Die mittels der Rotationsmatrix berechneten Pixelkoordinaten (x',y') sind keine natürlichen Zahlen mehr, sondern reelle. Sie müssen aber für das gedrehte Bild wieder den gegebenen Pixelpositionen des Bildrasters zugeordnet werden. Dieses Problem kann man mit einer Interpolation lösen. Dazu gibt es verschiedene Möglichkeiten. Zwei davon werden in den folgenden Unterkapiteln vorgestellt.

5.6.1 Nearest-neighbour Interpolation

Die berechneten Pixelkoordinaten werden einfach auf- oder abgerundet. Der Farbwert des gedrehten Pixels ist gleich dem Farbwert des Originalpixels.

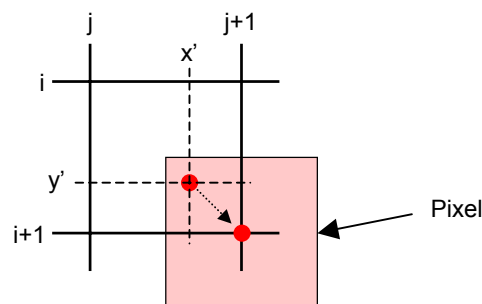


Abbildung 5.9 Pixelberechnung nearest neighbour

Damit alle Pixel des gedrehten Bildes eindeutig berechnet werden können, wird das Target-to-Source Mapping verwendet. Dabei geht man vom gedrehten Bild aus

und fragt sich, von wo der Pixel hergekommen ist. Dazu ist eine inverse geometrische Abbildung nötig.

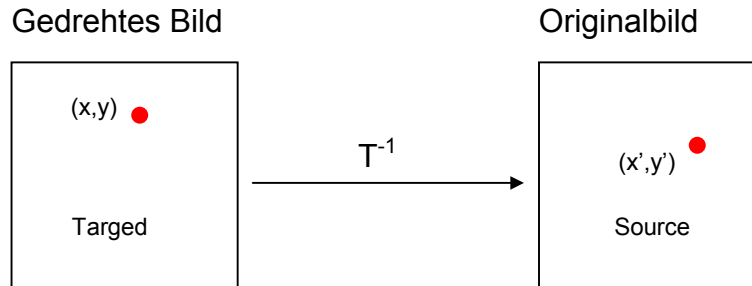


Abbildung 5.10 Inverse Abbildung

Dieses Verfahren ist relativ ungenau, da es wie ein Tiefpass wirkt. Besonders bei Bildern mit hohen Frequenzen gehen Informationen verloren. Die Bildqualität verschlechtert sich auch durch Blockbildung und Aliasing. Der Vorteil ist die einfache und schnelle Berechnung.

Original

gedreht



Abbildung 5.11 Rotation nearest neighbour

5.6.2 Bilineare Interpolation

Der Farbwert des gedrehten Pixels ist ein gewichteter Durchschnitt von 2x2 Pixel der näheren Umgebung, von der der Pixel herkommt.

$$\text{Farbwert} = (j + 1 - x') * (i + 1 - y') * A + (x' - j) * (i + 1 - y') * B + (j + 1 - x') * (y' - i) * C + (x' - j) * (y' - i) * D$$

Formel 5.6 bilineare Berechnung des Farbwerts

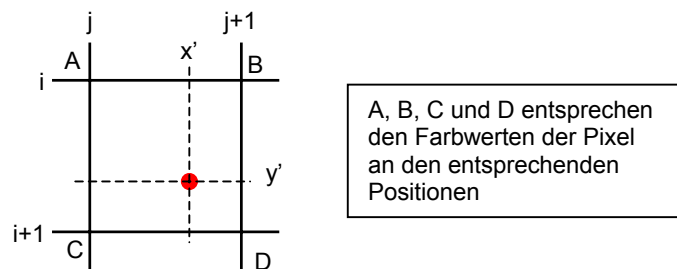


Abbildung 5.12 Pixelberechnung Bilinear

Bei dieser Methode werden die Bilder qualitativ besser als mit Nearest-neighbour. Sie ist jedoch ein wenig rechenintensiver.

Original

gedreht



Abbildung 5.13 Rotation Bilinear

Es gibt noch eine bikubische Methode. Dort ist der Farbwert des gedrehten Pixels ein gewichteter Durchschnitt von üblicherweise 4x4 Pixel. Dies ist nahezu eine optimale Interpolation. Feine Details gehen nicht verloren aber der Nachteil ist, dass der Rechenaufwand zu gross ist. Diese Interpolation kommt deshalb nicht in Frage.

5.7 Downsampling

Das Bild, welches die WebCam liefert, ist zu gross um daraus die Kreuzkorrelation zu berechnen. Würde man einfach einen Ausschnitt aus dem Bild nehmen, wird zwar die Anzahl Pixel verkleinert, jedoch gehen alle Bildinformationen verloren, welche ausserhalb des Ausschnittes liegen, was zu einem schlechteren Ergebnis führen könnte. Gleichzeitig stellt sich auch wieder die Frage, wo der Ausschnitt im originalen Bild der WebCam sein soll. Wegen diesen Nachteilen suchten wir nach einer anderen Methode um die Anzahl Pixel zu verringern und bekamen den Tipp des Downsamplings.

Mit dem Downsampling wird das Bild verkleinert, indem man das Originalbild in Blöcke der Grösse $N \times N$ Pixel unterteilt. Der Mittelwert des Blockes (x,y) ergibt dann den Grauwert des Pixels (i,j) des neuen Bildes (Aus $N \times N$ Pixel ergibt sich ein Pixel). Der Mittelwert hat leider auch ein Tiefpassverhalten, mit welchem kleine Details im neuen Bild verschwinden.

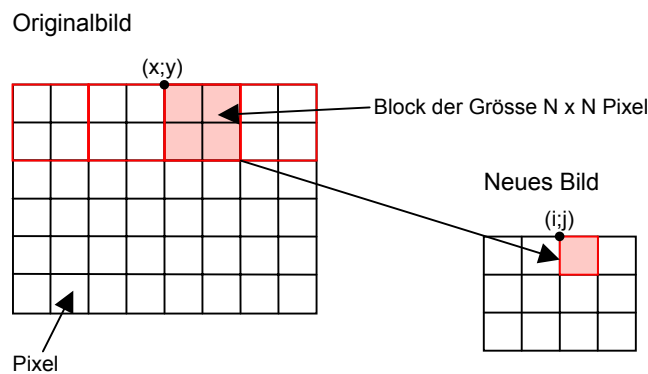


Abbildung 5.14 Downsampling



Abbildung 5.15 Beispiel Downsampling

6 Praktische Grundlagen

Neben den theoretischen Grundlagen, wurden auch die praktischen Grundlagen erarbeitet. Diese beinhalten hauptsächlich die Einarbeitung in die Tools MatLab und Microsoft Visual Studio sowie auch die Repetition der Hochsprache C / C++.

6.1 MatLab

MatLab⁵ ist ein Programm für das numerische Rechnen. Es ist ein universelles Werkzeug für technische und wissenschaftliche Berechnungen und Visualisierung von Daten. Wichtigster Datentyp in MatLab ist die Matrix.

Neben dem einfachen Handling mit Bildern, bringt MatLab noch viele fertige Funktionen, welche schnell erste Versuche zulassen. Mit MatLab können auch kleinere Programme geschrieben werden. Aus diesem Grund diente MatLab als Hilfsmittel bei den ersten Bildverarbeitungen. Auch allfällig geeignete Algorithmen wurden in MatLab getestet. Die als brauchbar erscheinenden Ergebnisse wurden dann in C++ implementiert.

Mit folgendem Befehl kann ein Bild vom aktuellen Verzeichnis gelesen werden:

```
bild = imread('dateiname.tif');  
bild = double(bild);
```

Das eingelesene Bild kann mittels folgenden Befehlen in einem Fenster ausgegeben werden:

```
figure(1);  
imagesc(bild);  
colormap(gray)
```

Die Kreuzkorrelation ist in MatLab bereits integriert. Für Bilder wird aber eine 2-dimensionale Kreuzkorrelation gebraucht. Auch diese ist bereits vorhanden:

```
bild_corr = xcorr2(bild1,bild2);
```

Dieses Resultat kann nun als 3D Grafik angezeigt werden:

```
[r,c] = size(bild_corr);  
[X,Y] = meshgrid(1:r, 1:c);  
surf(X,Y,bild_corr)
```

Je nach Bildern kann aus diesem 3D-Graphen leider nicht die genaue Verschiebung ausgelesen werden. Der Grund besteht darin, dass beide Bilder einen Mittelwert haben. Deshalb muss vor der Kreuzkorrelation der Mittelwert subtrahiert werden (DC subtrahieren):

```
bild1 = bild1 - mean(mean(bild1));  
bild2 = bild2 - mean(mean(bild2));
```

⁵ MatLab: www.mathworks.com

Jetzt kann die Verschiebung zwischen den beiden Bildern direkt aus dem Graphen ausgelesen werden (Die Abweichung des absoluten Maximum von der Mitte, ergibt die Translation).

6.1.1 MEX-Schnittstelle

Die MEX-Schnittstelle erlaubt, in C geschriebene Funktionen in MatLab aufzurufen. Mit einem C Compiler wird eine *.dll Datei erzeugt, welche unter MatLab aufgerufen werden kann.

Anleitung für das Erstellen eines MEX-files mittels MSVC:

- Microsoft Visual C++ installieren
- MatLab installieren
- Unter Windows NT/2k/XP Lokale Benutzerrechte auf PowerUser ändern (Hauptbenutzer)
- MatLab starten, im Command Window folgendes eingeben:
 - `mex -setup`
 - ↵ drücken
 - MSVC Compiler auswählen
 - **y** drücken
- MSVC starten
 - Menu Tool → Customize, Add-ins and Makro Files
 - MatLab Add-in selektieren
 - close
- Neues mex-Project unter MSVC erzeugen:
 - Menu File -> New
 - Projects, MatLab Project Wizard
 - Project Name und Pfad eingeben
 - OK
 - C-MEX DLL wählen
 - Finish
 - Ins Projekt Verzeichnis wechseln und `projectname.c` file wählen
 - open drücken, files werden erstellt

Die unter MSVC++ kompilierte dll muss im gleichen Verzeichnis sein, wie das aktuelle Verzeichnis unter MatLab (resp. wie das m-file).

Unter MatLab kann die dll folgendermassen aufgerufen werden (dll-file heisst `user_add.dll`):

```
matrix3 = user_add(matrix1, matrix2);  
close user_add;
```

6.1.2 WebCam unter MatLab

Mit Hilfe des Programms vfm⁶ (Vision for MatLab) kann direkt im MatLab ein Bild von der WebCam aufgenommen werden.

Die Dateien vfm.dll und vfm.m müssen im selben Verzeichnis sein, wie das eigene m-file. Folgender Befehl liest das Bild von der WebCam ein, konvertiert das RGB-Bild in ein Graustufenbild und generiert einen Array namens bild1 mit Double-Werten.

```
bild1 = double(rgb2gray(vfm('grab',1)));
```

Damit auch noch andere Programme auf die WebCam zugreifen können, muss der Zeiger entfernt werden:

```
vfm('show',0);
```

6.2 USB WebCam

Als wired WebCam wurde eine USB WebCam verwendet.

Hersteller: Creative⁷

Typ: WEBCAM PRO eX

Die Installation der USB WebCam erfolgt über die Originaltreiber von Creative. Nach der Installation kann sowohl unter MatLab als auch unter MSVC++ mit der USB WebCam gearbeitet werden



Abbildung 6.1 USB WebCam

⁶ vfm: (Vision for MatLab) <http://www2.cmp.uea.ac.uk/~fuzz/vfm/default.html>

⁷ Creative: www.creative.com

6.3 WirelessCam

Als Wireless WebCam wurde diejenige verwendet, welche auch schon bei anderen Studien- und Diplomarbeiten verwendet wurde.

Hersteller: D-Link⁸

Typ: DCS-1000W 2.4GHz Wireless Internet Camera.

Diese WebCam hat bereits einen Webserver integriert. Die Einstellungen können also bequem mit einem Browser vorgenommen werden. Dank dem Schalter auf der Rückseite der Camera kann zwischen normalem wired LAN und WLAN umgeschaltet werden. Beim Betätigen des Resetknopfes auf der Rückseite der Camera werden alle Einstellungen auf die Werkseinstellungen zurückgesetzt. Nach dem Betätigen des Resetknopfes sollte die WebCam über wired-LAN konfiguriert werden.

Um mit der WirelessCam mobil zu sein, sollte eine direkte Adhoc WLAN Verbindung gewählt werden und nicht über einen AccessPoint. Dabei wird die Wireless-Cam der AdHoc Master sein. Damit kann die WirelessCam auch mit verschiedenen Computern einfach verwendet werden.

6.3.1 Konfiguration der WirelessCam

- WirelessCam über wiredLAN verbinden. Lokale IP-Einstellungen:
IP: 192.168.0.40 SubNet 255.255.255.0
- WirelessCam IP eingeben im Browser: 192.168.0.20



Abbildung 6.2 WebServer WirelessCam

⁸ D-Link: www.dlink.com

- Setup anklicken, folgende Einstellungen übernehmen

The screenshot displays the 'Advanced' tab of the WirelessCam configuration interface. The 'System Setting' section is active, showing the following configurations:

- Camera Name: CS-4A98D01
- Location: 6004
- IP Assignment: Manually Assign
 - IP Address: 192.168.0.20
 - Subnet Mask: 255.255.255.0
 - Default Gateway: 0.0.0.0
- Assign Automatically Using: (with sub-options for RARP, BOOTP, DHCP, and PPPoE)
- DNS IP Address: 1: 0.0.0.0, 2: 0.0.0.0
- Wireless Interface: Connection Mode: 802.11 Adhoc, Network Name: D-Link WebCam (SSID), Wireless Channel: 12, Encryption WEP Key: HEX
- LED Control: Normal
- Loading ActiveX From: (empty field)
- Open Second Port: Yes
 - 1: Web Server - Default 80 (0)
 - 2: Transfer Image - Default 8481 (0)

Buttons for 'Apply' and 'Cancel' are visible at the bottom right.

Abbildung 6.3 Netzwerkeinstellungen WirelessCam

- Apply drücken
- Alle anderen Optionen müssen nicht verändert werden

Nun kann man die WirelessCam auf WLAN umschalten. Der verwendete Computer muss noch konfiguriert werden:

- lokale IP der WLAN-Netzwerkkarte einstellen:
IP: 192.168.0.X SubNet: 255.255.255.0 (X zwischen 0 und 255, ausser 20)

Jetzt kann mit dem Computer die Verbindung zur WirelessCam hergestellt werden. Als Kontrolle IP des WirelessCam im Browser eingeben. Wenn die Startseite des embedded Webserver angezeigt wird, funktioniert die Funkverbindung.

6.4 MSVC++⁹

Das Programm wird mit der Entwicklungsumgebung von Microsoft programmiert. Es macht keinen Sinn die Programmiersprache zu wechseln, da der Berechnungsalgorithmus später auf einem DSP laufen soll, welcher nur mit C programmiert werden kann.

6.4.1 Bilder einlesen mittels USB WebCam

Die Implementation erfolgt mit der Library Video for Window (vfw). Als Vorlage diente das Praktikumsprogramm aus dem Modul DigMed. Ein Thread kopiert jeweils das aktuelle Bild der WebCam in ein Array, und gibt es auf den Bildschirm aus.

6.4.2 Bilder einlesen mittels WirelessCam

Mit der WirelessCam wird das Bild automatisch mittels eines ActiveX-Treibers angezeigt. Das Bild muss nun direkt aus dem Bildschirmspeicher kopiert werden. Der ActiveX-Treiber befindet sich auf der Treiber-CD der WirelessCam. Den C-Code für das Kopieren des Bildes aus dem Bildschirmspeicher haben wir von Thomas Balcarczyk bekommen. -> Vielen Dank!

6.4.3 Zeitmessung

Um die verschiedenen Algorithmen miteinander vergleichen zu können, benötigen wir eine Zeitmessung. Diese wurde mit den gleichen Funktionen programmiert, wie in den Übungen des Moduls DigMed:

```
__int64 freq, start, stop;  
double diff;
```

⁹ MSVC++: Microsoft Visual C++ www.microsoft.com

```

QueryPerformanceFrequency((LARGE_INTEGER*)&freq);
QueryPerformanceCounter((LARGE_INTEGER*)&start);
{
}
QueryPerformanceCounter((LARGE_INTEGER*)&stop);
diff = 1000*(double(stop-start))/freq;
printf ("\nDiff: %fms\n", diff);           //Ausgaben in ms
  
```

6.4.4 MFC

Für die Programmierung des GUI wurde die MFC verwendet.

Die MFC ist eine Ansammlung von C++ Klassen (Werkzeuge, Datentypen, Exception, Dialoge etc.) die das API¹⁰ kapseln, um es objektorientiert nutzen zu können.

Ab Version 2.0 kam u.a. die Dokument-Ansicht Architektur hinzu. Damit stellt die MFC dem Programmierer ein Gerüst für Anwendungen bereit (Application Framework). Mit dem MFC-Application Wizard können schnell und einfach Anwendungsgerüste erstellt werden.

Für unsere Arbeit waren die GDI-Objektklassen¹¹ besonders wichtig. Sie werden von der abstrakten Basisklasse CObject abgeleitet. Mit CBrush und Cpen wurden die Verschiebungslinien und Rotationspfeile des GUI's erstellt.

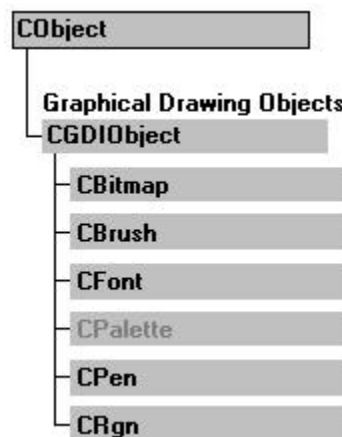


Abbildung 6.4 GDI - Objektklassen¹²

Instanzen der GDI-Objektklassen müssen in einem Device Context verwendet werden. Die Version 6.0 der MFC-Bibliothek unterstützt eine Reihe von Device

¹⁰ API: Application Programming Interface

¹¹ GDI: Graphics Device Interface

¹² Quelle: MSDN (Microsoft Developer Network)

Context-Klassen. Die Basisklasse CDC enthält alle zum Zeichnen erforderlichen Member-Funktionen. Für Bildschirmausgaben werden für gewöhnlich von CClientDC und CWindowDC abgeleitete Klassen verwendet. Für andere Geräte wie Drucker und Puffer im Arbeitsspeicher verwendet man Instanzen der Basisklasse CDC.

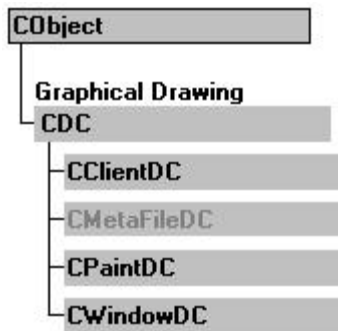


Abbildung 6.5 Device Context Klassen¹³

In einem Device Context ist je ein Objekt der oben genannten GDI-Klassen aktiv. Sind keine eigenen GDI-Objekte definiert worden, werden die Standardobjekte (beispielsweise schwarzer Pen und weisser Pinsel) verwendet. Diese Standardobjekte können mit SelectObject durch Selbstdefinierte ersetzt werden. Folgendes Beispiel zeigt, wie ein Pen und ein Brush definiert werden:

```

CPen bluePen(PS_SOLID, 1, RGB(0, 0, 255));
CBrush yellowBrush;
yellowBrush.CreateSolidBrush(RGB(255, 255, 0));

```

Dummerweise kann man die Auswahl eines GDI-Objektes nicht aufheben, ohne ein neues Objekt auszuwählen. Eine einfache Lösung dieses Problems besteht darin, das alte Objekt zu speichern, wenn das eigene GDI-Objekt in den Device Context selektiert wird. Wird das eigene Objekt nicht mehr gebraucht, kann man es durch das alte wieder ersetzen. Diese werden automatisch beim Verlassen der Funktion gelöscht. So ist sichergestellt, dass die eigenen GDI-Objekte den Speicher wieder freigeben und der Device Context sich wieder im ursprünglichen Zustand befindet. Folgendes Beispiel zeigt, wie ein GDI-Objekt in den Device Context selektiert, das alte Objekt gespeichert und am Schluss wieder zurückselektiert wird:

```

pOldBrush = cdc.SelectObject(&yellowBrush);
/*      irgendwas zeichnen      */
cdc.SelectObject(pOldBrush);

```

¹³ Quelle: MSDN (Microsoft Developer Network)

Auf der Document-View-Architecture basierende Programme sollte die Grafikausgabe grundsätzlich in der Ansichtsklasse (View) erfolgen. Diese stellt die Funktion `OnDraw()` bereit. Diese wird immer dann aufgerufen, wenn die Ansicht oder ein Teil davon neu gezeichnet werden muss.

Wenn man sich ein MFC-Programm ansieht, wird man sich vielleicht wundern, dass keine `main`-Funktion vorhanden ist. Diese ist innerhalb der MFC versteckt. Der Programmierer bekommt sie nicht zu sehen. Das MFC-Programm wird gestartet, indem man eine globale Instanz des Applikationsobjekts erzeugt. Diese Instanz hält den Rest des Programms zusammen und ist der Startpunkt für den Haupt-Thread der Anwendung. Innerhalb der Member-Funktion `Run` befindet sich die Nachrichtenschleife der MFC-Anwendung. Aus dieser Funktion wird zudem die Funktion `OnIdle` aufgerufen, wenn keine Nachrichten aus der Nachrichtenschleife zu verteilen sind. `OnIdle` kann von Programmen überschrieben werden, um Operationen im Hintergrund durchzuführen, ohne einen eigenen Thread dafür verwenden zu müssen. Man sollte darauf achten, die Nachrichtenschleife nicht allzu lange zu blockieren, denn wenn das Programm keine Nachrichten mehr verarbeiten kann, reagiert es auch nicht mehr auf Benutzereingaben. Folgendes Bild illustriert den Ablauf einer MFC-Anwendung:



Abbildung 6.6 Ablauf einer MFC-Anwendung¹⁴

¹⁴ Quelle: MSDN (Microsoft Developer Network)

7 Realisierung

Als Grundlage diente das MFC Programm CaptureExample aus dem Praktikum des Moduls DigMed. Mit diesem Programm wurden auch erste Erfolge erzielt. Dazu gehörten das Einlesen der Bilder von der USB WebCam sowie erste Erfahrungen mit dem GUI. Da mit diesem Programm keine ActiveX Komponenten funktionieren, wurde ein neues MFC - Projekt erstellt, bei dem man beim Schritt 3 des MFC-Assistenten beim Kontrollkästchen ‚ActiveX-Steuererelemente‘ ein Häkchen setzte. Mit diesem Projekt gelang es, auch Bilder von der WirelessCam einzulesen. Beim Starten des Programms wird das GUI gestartet. nach dem Drücken auf den Button "Start", werden alle Variablen, etc. initialisiert und es wird ein Thread gestartet. Der Thread läuft in einer Endlosschleife bis der Button "Stop" gedrückt wird. In dieser Schleife wird das Bild von der richtigen WebCam kopiert und dann die Funktion Algorithmus gestartet, welche alle Berechnungen und Visualisierungen vornimmt. Nach dem Drücken des Button "Stop" beendet sich der Thread.

7.1 GUI

Das Erscheinungsbild des GUI ist in Abbildung 7.1 ersichtlich:

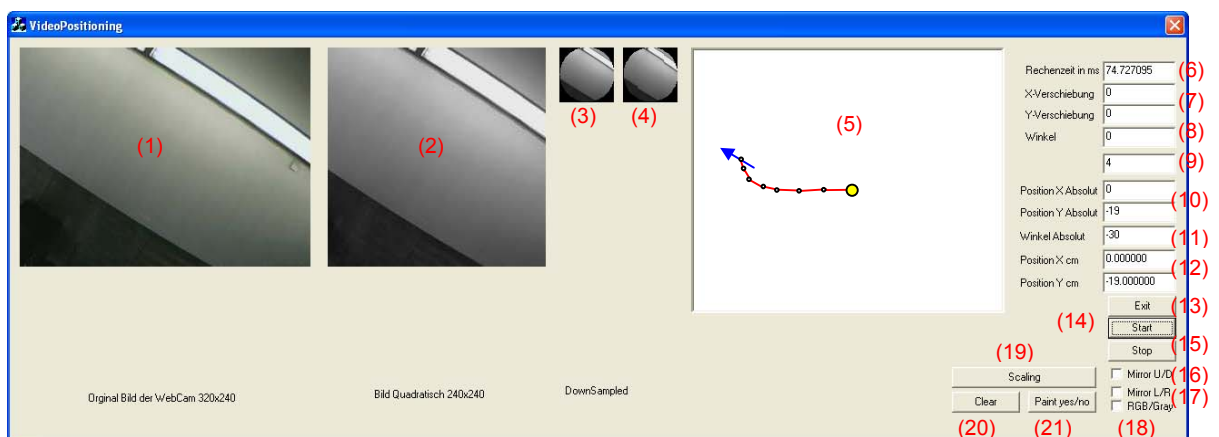


Abbildung 7.1 GUI

Legende:

- (1) Originalbild der Kamera
- (2) Bild quadratisch zugeschnitten und in Graustufen
- (3) downgesampeltes Bild
- (4) um -4° , -2° , 2° , 4° gedrehtes Bild
- (5) Visualisierung der Verschiebung und Rotation
- (6) Rechenzeit für eine Berechnung in Millisekunden
- (7) relative X- und Y-Verschiebung in Pixel
- (8) Rotationswinkel in Grad zwischen zwei Bildern
- (9) Anzahl Berechnungen zwischen jeder Korrektur

- (10) absolute Position X und Y in Pixel
- (11) absoluter Winkel in Grad,
- (12) absolute Position X und Y in cm
- (13) Programm verlassen
- (14) Berechnung starten
- (15) Berechnung stoppen
- (16) Originalbild horizontal spiegeln
- (17) Originalbild vertikal spiegeln
- (18) Originalbild in Graustufen wandeln
- (19) Skalierung
- (20) Zeichnung löschen und Werte zurücksetzen
- (21) Zeichnen Ja/Nein

Die auf dem GUI visualisierte Verschiebung und Drehung funktioniert wie folgt: Zu Beginn der Bewegungsschätzung wird ein gelber Startpunkt gezeichnet. Die relative Verschiebung wird mit roten Linien dargestellt. Diese Linien verbinden die berechneten Punkte, die jeweils weiss markiert werden. Die Rotation wird durch einen blauen Pfeil visualisiert. Die Pfeilspitze markiert die Richtung, in der der Roboter bzw. der Lastwagen schaut.

In den folgenden zwei Unterkapiteln wird im Detail erklärt, wie die rote Verschiebungslinie und der Pfeil realisiert wurden.

7.1.1 Verschiebungslinie

Die rote Verschiebungslinie wird vom letzten berechneten Punkt zum aktuellen Punkt gezogen. Dazu wird eine Variable `topicalPoint` verwendet. Diese Variable enthält immer die aktuelle Position und wird beim Programmstart mit dem Startpunkt initialisiert. Mit dem Befehl `MoveTo(topicalPoint)` wird der Stift auf den Startpunkt gesetzt. Anschliessend wird die Verschiebung dazu addiert und der neue Wert wieder in der Variable `topicalPoint` abgespeichert. Mit dem Befehl `LineTo(topicalPoint)` wird nun die Linie zum neuen aktuellen Punkt gezeichnet und so weiter.

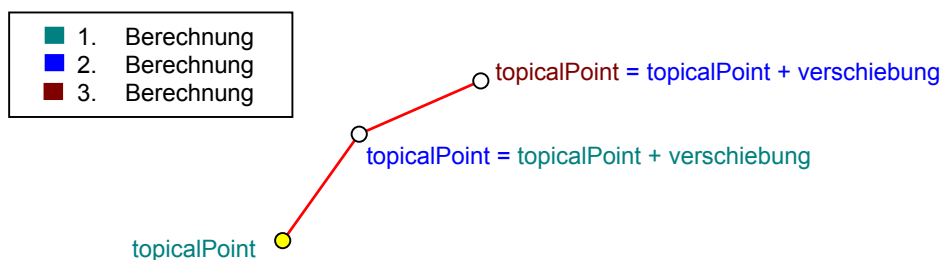


Abbildung 7.2 Verschiebungslinie

7.1.2 Pfeil

Die halbe Länge des Pfeils wird in einer Konstanten *length* definiert (10 Pixel). In Abhängigkeit des Rotationswinkels wird die *x*- und *y*-Koordinate des Start- und Endpunktes des Pfeils berechnet. Diese beiden Punkte werden mit einer blauen Linie verbunden, welche mit den Funktionen *MoveTo(startPointt)* und *LineTo(endPoint)* gezeichnet wird. Die Pfeilspitze wird mit einem Polygon gebildet. Der Start- und Endpunkt wird folgendermassen berechnet:

$$\text{versatzX} = \text{length} * \cos(\varphi)$$

$$\text{versatzY} = \text{length} * \sin(\varphi)$$

$$\text{startPoint} = (\text{topicalPoint} - \text{versatzX}, \text{topicalPoint} + \text{versatzY})$$

$$\text{endPoint} = (\text{topicalPoint} + \text{versatzX}, \text{topicalPoint} - \text{versatzY})$$

Formel 7.1 Start- Endpunkt

Es ist zu beachten, dass die *start*- und *endPoints* bezüglich der *j*- und *i*-Koordinaten berechnet werden. Der Ursprung dieser Koordinaten liegt in der linken oberen Ecke des Clientbereichs des Dialogfensters.

Der *versatzX* und $-\text{Y}$ hingegen werden bezüglich der *x*- und *y*-Koordinaten berechnet, dessen Ursprung stets im aktuellen Messpunkt, dem *topicalPoint*, liegt. Diese Koordinatentransformation von (*i,j*) nach (*x,y*) vereinfacht die Berechnung wesentlich.

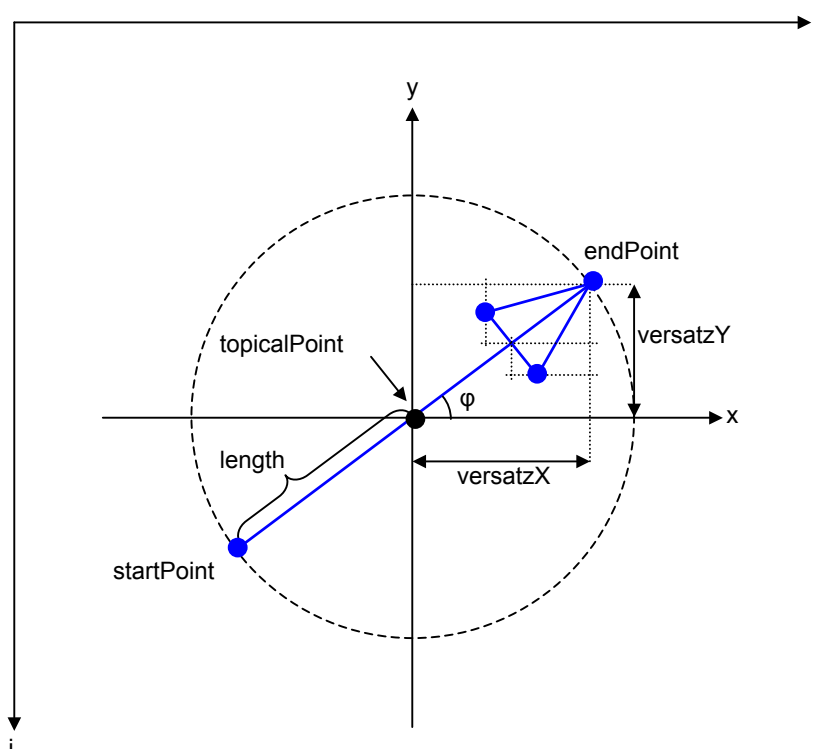


Abbildung 7.3 Darstellung Pfeil

Die Pfeilspitze wird mittels einem Polygon gebildet, das blau ausgefüllt wird. Dazu muss ein Array mit drei Eckpunkten berechnet werden. Der endPoint ist schon bekannt. Die anderen Punkte lassen sich wie folgt berechnen:

$$\begin{aligned}
 p1X &= \text{endPointX} - 6 * \cos(\varphi) + 2 * \cos(90^\circ - \varphi) \\
 p1Y &= \text{endPointY} + 6 * \sin(\varphi) + 2 * \sin(90^\circ - \varphi)
 \end{aligned}$$

$$\begin{aligned}
 p2X &= \text{endPointX} - 6 * \cos(\varphi) - 2 * \cos(90^\circ - \varphi) \\
 p2Y &= \text{endPointY} + 6 * \sin(\varphi) - 2 * \sin(90^\circ - \varphi)
 \end{aligned}$$

Formel 7.2 Punkte des Polygon

Das Polygon wird mit der Funktion *Polygon(Array, 3)* gezeichnet.

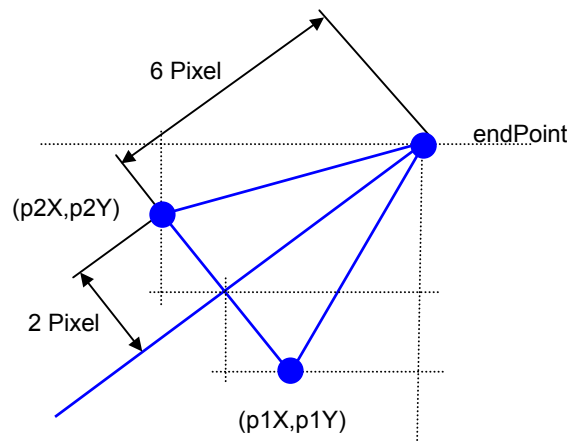


Abbildung 7.4 Darstellung Pfeilspitze

Jedes Mal, wenn eine neue Position berechnet wird, muss an dieser Stelle ein Rotationspfeil gezeichnet werden. Damit die Visualisierung übersichtlich bleibt, muss der vorhergehende Pfeil gelöscht werden. Dazu wird die alte Position des Pfeils gespeichert und bevor man den neuen zeichnet, wird der alte mit der Hintergrundfarbe übermalen.

Es gibt noch einen eleganteren Ansatz, der aber nicht mehr rechtzeitig umgesetzt werden konnte. Die MFC stellt eine Funktion *InvalidateRect()* bereit. Mit dieser Funktion kann man nun das orthogonale Rechteck, das durch den Rotationspfeil aufgespannt wird, für ungültig erklären. Bei der nächsten Berechnung wird dieser Vorgang wiederholt. Windows merkt sich diese beiden ungültigen Rechtecke und berechnet ein neues ungültiges Rechteck, das sich aus der Schnittmenge des Client-Rechtecks (Bereich des Dialoges ohne Menu- und Statuszeile) mit der Vereinigung der beiden ungültigen Rechtecke ergibt. Somit würde der alte Pfeil gelöscht.

7.2 Algorithmus

Vor dem Aufruf der Funktion Algorithmus, wird von der richtigen WebCam das aktuelle Bild in einen Array kopiert. Da die Dimension von diesem Bild nicht quadratisch ist, wird in der Breite auf beiden Seiten je ein Streifen von 40 Pixel weggeschnitten. Aus einem 320x240 Bild wird ein 240x240 Bild.

7.2.1 Downsampling

Durch das Downsampling werden die Dimensionen des Bildes nochmals verkleinert. Mit der Konstante BLOCK kann der Downsamplingfaktor angegeben werden. Ein Faktor von zwei bedeutet, dass immer 2x2 Pixel zu einem neuen Pixel zusammengefasst werden. Dadurch wird das neue Bild vier mal kleiner. Je grösser der Downsamplingfaktor ist, desto grösser wird das Tiefpassverhalten. Das neue Bild hat keine scharfen Kanten mehr. Kleinere Objekte können verschwinden. Die besten Ergebnisse wurden mit einem Downsamplingfaktor zwischen vier und sechs erreicht.

7.2.2 Abrunden

Damit alle Pixel vom Originalbild einem Pixel des gedrehten Bildes zugeordnet werden können, werden die Bilder abgerundet. Sonst hätte man das Problem, dass Pixel aus dem Bildbereich verschwinden und andere, nicht existierende, ins Bild hineinrutschen. Alle Pixel ausserhalb dieses Kreises werden auf Null (Schwarz) gesetzt. Die anderen erhalten den Farbwert vom Originalbild.

Der Kreis wird folgendermassen berechnet:

$$s = \sqrt{x^2 + y^2}$$
$$x = \text{radius} - j$$
$$y = \text{radius} - i$$

Formel 7.3 Bild abrunden

Falls s grösser ist als der Radius wird der Pixel schwarz, wenn s kleiner oder gleich dem Radius ist, bleibt der Farbwert wie er ist (siehe Abbildung 7.5)

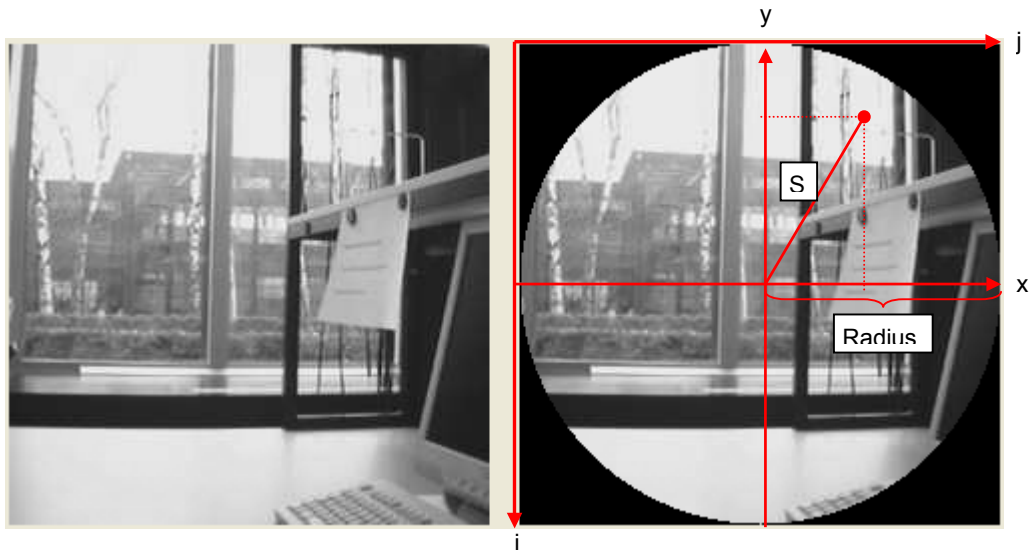


Abbildung 7.5 Bild abrunden

7.2.3 Rotation

Um die Rotation zu detektieren, wurde die Methode des Vergleichs des aktuellen Bildes mit einem dazu gedrehten Bild umgesetzt. Diese Idee wird im Kapitel 5.6 erwähnt.

Aus den Überlegungen zur Winkelgeschwindigkeit des Roboters, hat sich herausgestellt, dass eine Auflösung von 2° ausreichend ist. Wenn man davon ausgeht, dass sich der Roboter innerhalb vom einen Frame zum anderen maximal 5° gedreht haben kann, muss man das aktuelle Frame mit dem vorhergehenden, welches um -4° , -2° , 2° und 4° gedreht wurde, korrelieren.

Da für die Berechnung der Bewegungsschätzung nur ein begrenztes Zeitfenster zur Verfügung steht, musste man für die Drehung der Bilder auf den Gebrauch der Winkelfunktionen Sinus und Cosinus innerhalb des Thread verzichten, da sie extrem rechenintensiv sind. Das Zeitproblem wurde gelöst, indem man beim Initialisieren des Programms Umsetztabelle erstellt, in denen die Positionen der Pixel stehen. Da man für die Referenzkorrelation einen grösseren Winkelbereich benötigt, werden Tabellen für den Bereich von -50° bis $+50^\circ$ in Zweierschritte (nur die geraden Winkel) berechnet.

Der Einfachheit wegen wurde der Drehpunkt in die Mitte des Bildes gesetzt. Dies würde zwar voraussetzen, dass sich die Achse der Kameralinse genau in der Drehachse befinden muss. Das ist in der Versuchsanordnung aber nicht der Fall und es funktioniert trotzdem. Der Abstand dieser Achsen darf einfach nicht zu gross sein.

Für das Festlegen des Drehpunktes ist eine Koordinatentransformation nötig. Die Bildkoordinate (i,j) mit dem Ursprung in der oberen linken Bildecke wird auf die

Koordinate (x,y) mit dem Ursprung im Drehpunkt transformiert. Dazu werden folgende Formeln verwendet:

$$\begin{aligned} \text{center} &= \text{bildbreite}/2 \\ x &= j_ziel - \text{center} \\ y &= \text{center} - i_ziel \end{aligned}$$

Formel 7.4 Koordinatentransformation

Für die Berechnung der Tabellen verwendet man die in Kapitel 5.6.1 beschriebene Source-to-Target Methode. Für die Bildkoordinaten i und j wird je eine Tabelle erstellt, wobei jede Tabelle die Werte zu allen Winkeln und allen Pixelpositionen enthalten.

Die dazu verwendeten Formeln lauten

$$\begin{aligned} \text{resultJ}[\text{winkel}] &= \cos(\text{winkel}) * x + \sin(\text{winkel}) * y + \text{center} \\ \text{resultI}[\text{winkel}] &= \text{center} - \cos(\text{winkel}) * y - \sin(\text{winkel}) * x \end{aligned}$$

Formel 7.5 Berechnung der Umsetztabeln

wobei resultJ und resultI zweidimensionale Matrizen darstellen, in denen pro Winkel eine Zeile mit den Pixelkoordinaten zugeordnet wird.

Damit der Drehsinn des Gefährts mit dem des Pfeils im GUI übereinstimmt, muss die transponierte Rotationsmatrix verwendet werden. Deshalb sind in der Formel 7.5 die Vorzeichen etwas anders, als man es vielleicht erwarten würde.

Mit diesen Umsetztabeln kann man nun ziemlich schnell die gedrehten Bilder generieren. Dazu wurde sowohl die nearest-neighbour als auch die bilineare Interpolation implementiert (Kapitel 5.6.1 und 5.6.2).

7.2.4 Bereichskorrelation

Das aktuelle Bild wird abgerundet. Mit diesem Bild und den gedrehten Bildern, welche aus dem Bild des vorhergehenden Durchgangs berechnet wurden, führt man nun die Bereichskorrelation durch. Doch vorher muss beim aktuellen und dem des vorherigen Durchgangs der Mittelwert abgezogen werden.

Für jeden der fünf Winkel (-4° , -2° , 0° , 2° , 4°) wird eine Bereichskorrelation durchgeführt. Die Ergebnisse der Korrelationen werden in einem 3-dimensionalen Array gespeichert. In diesem Array wird das Maximum gesucht. Das Maximum bestimmt die drei Koordinaten der Bewegung.

1. Koordinate: x - Richtung
2. Koordinate: y - Richtung
3. Koordinate: Winkel

Die Einheiten der Bewegungen sind in Pixel. Beim Winkel ist die Einheit Grad.

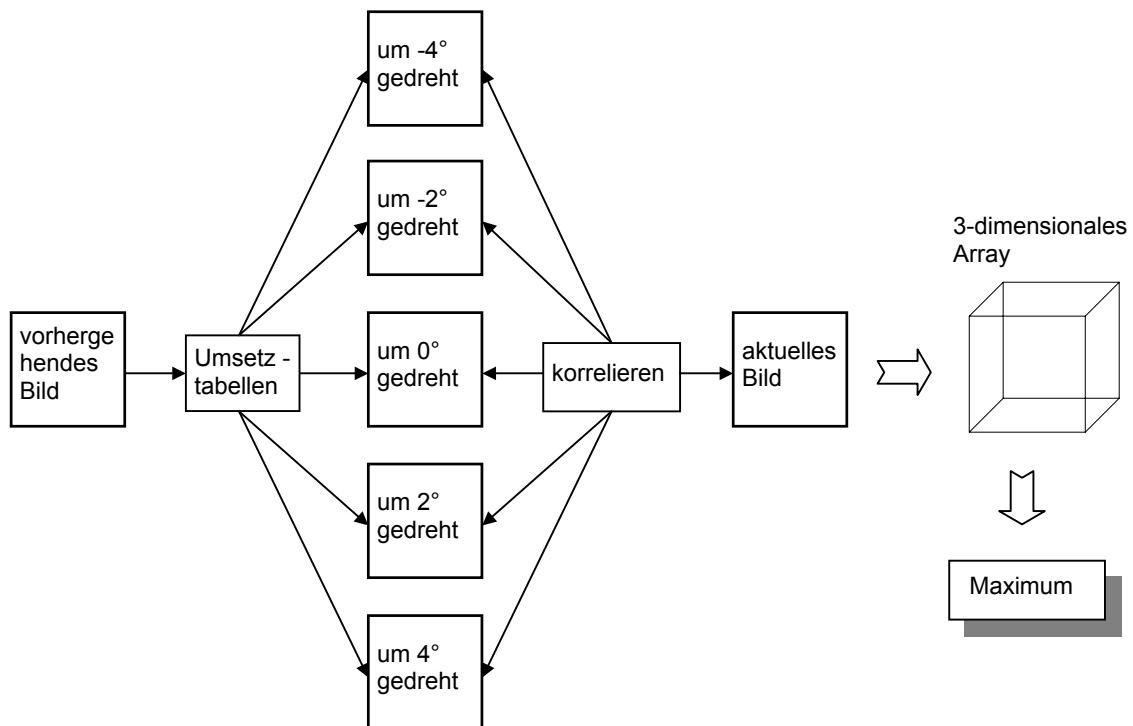


Abbildung 7.6 Berechnungsvorgang

Sowohl die aktuellen Bewegungen, als auch die absoluten Positionen werden numerisch im GUI angezeigt.

7.3 Fehlerkorrektur

Da sich das Maximum nicht immer genau bei einem Pixel befinden muss, treten Quantisierungsfehler auf. Um diese zu minimieren, wird nach fünf detektierten Bewegungen eine Referenzkorrelation gemacht.

Das Referenzbild wird in einem separaten Array gespeichert. Nach den fünf detektierten Bewegungen wird die nächste Bereichskorrelation mit dem Referenzbild gemacht. Zusätzlich wird nicht nur mit fünf Winkeln, sondern mit sieben Winkeln korreliert. Das ergibt einen grösseren Bereich, um auch grosse Fehler zu korrigieren. Aus der Bereichskorrelation ergibt sich eine Differenz, mit welcher die absoluten Positionen korrigiert werden.

7.4 Skalierung

Die berechneten Bewegungen der Translationen sind jeweils in der Einheit Pixel. Mit einem Skalierungsfaktor soll auf die gebräuchliche Einheit Meter umgerechnet werden können. Da der Skalierungsfaktor abhängig ist von der Höhe der Decke, ist es nicht möglich, einen festen Wert zu definieren. Damit es trotzdem möglich ist, die Änderungen der Translation in Meter anzuzeigen, wurde eine Skalierungsprozedur geschaffen. Diese ermöglicht es, den Skalierungsfaktor direkt vor Ort zu bestimmen. Im ersten Teil der Skalierungsprozedur muss die WebCam um 20cm in eine Richtung verschoben werden. Die Anzahl Pixel wird zwischengespeichert. Nun werden alle Werte auf null zurückgesetzt, danach muss die WebCam wieder zum Ursprung zurückbewegt werden. Am Ursprung angelangt, wird wieder die Pixeldifferenz zwischengespeichert. Das Mittel der Beträge der beiden Zwischenwerte gibt an, wieviel Pixel 20cm entsprechen. Nun wird dieser Wert noch durch 20 dividiert. Das Ergebnis ist der Skalierungsfaktor mit der Einheit Pixel pro cm.

Anleitung für Skalierung:

1. Die WebCam darf nicht bewegt werden
 2. Eine Strecke von 20cm abmessen
 3. Den Button "Scaling" drücken
 4. Die WebCam mit konstanter Geschwindigkeit um diese 20cm verschieben
 5. Den Button "20cm, dann drücken" drücken
 6. Die WebCam mit konstanter Geschwindigkeit wieder zum Ursprung zurückschieben
 7. Den Button "zurück zum Ursprung" drücken
- Skalierung abgeschlossen.

Ab diesem Zeitpunkt wird im GUI die Verschiebung nicht nur in Pixel sondern auch in der Einheit cm angezeigt.

7.5 Fahrzeug

Um unseren Algorithmus zu testen, wurde ein Spielzeuglastwagen verwendet, auf dem die WirelessCam montiert wurde. Diese Anordnung entspricht ungefähr dem eigentlichen Verwendungszweck auf einem Roboter.

Auf der Laderampe wurde mittels vier Distanzschrauben ein Brett montiert. Auf diesem wurde die mit der WirelessCam mitgelieferte Halterung festgeschraubt. Der Akku und der Spannungsregler befinden sich ebenfalls auf dem Brett. Damit die Kamera vertikal positioniert werden kann, benötigt es noch ein Zwischenstück, damit die Antennen das Brett nicht berühren.



Abbildung 7.7 Fahrzeug mit WebCam

7.6 Spannungsregler

Die WirelessCAM übermittelt die Bilddaten zwar über WLAN, die Speisung ist aber über ein Steckernetzteil gelöst. Damit die WirelessCAM ihrem Namen gerecht wird, wurde eine Akkuspeisung gebaut. Da der Stromverbrauch recht hoch ist (max. 2.5A) muss mit Akkus aus dem Modelbaugewerbe gearbeitet werden.

Die Speisung besteht aus einem LM7805C¹⁵ Längsregler. Der Spannungsregler wird wegen der hohen Stromaufnahme im TO-220 Gehäuse verwendet.

¹⁵ LM7805C: 5V Spannungsregler. <http://www.national.com/ds/cgi/LM/LM7512C.pdf>

7.6.1 Schema

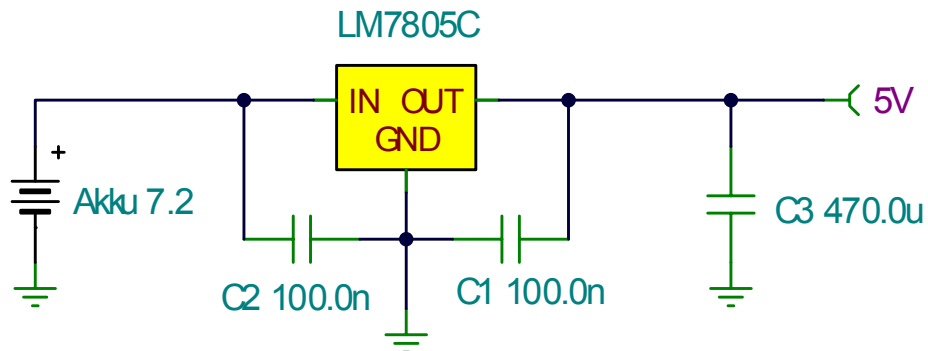


Abbildung 7.8 Spannungreglung

7.6.2 Beschreibung

C1 und C2 sind vorgegeben, gemäss Datenblatt des Spannungsreglers. C3 dient als Stützkondensator. Der Spannungsregler trägt noch einen Kühlkörper, um die Wärme besser an die Umgebung abzugeben.

7.7 Programmstruktur

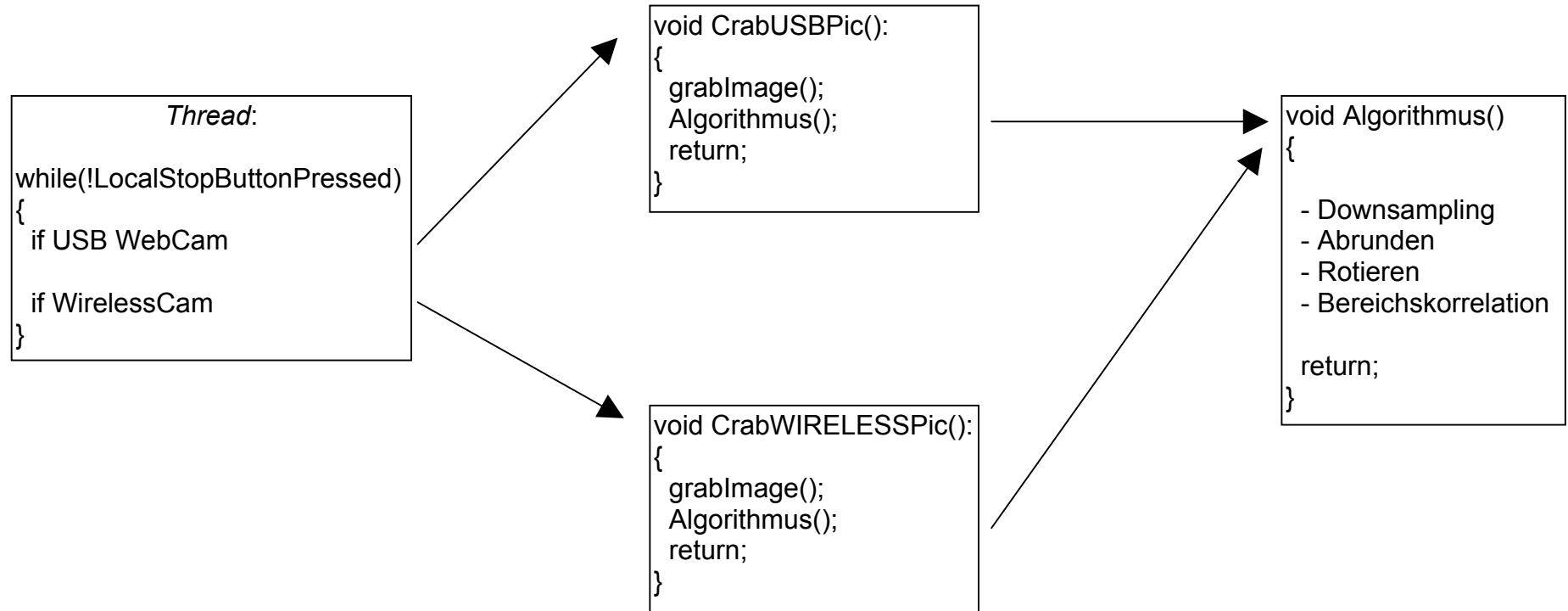


Abbildung 7.9 Programmstruktur

8 Optimieren der Parameter

| Parameter | Name der Konstante im C - Code | Wert | Ergebnis |
|---------------------------------|--------------------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Blockgrösse für Downsampling | BLOCK | 3 | Wert zu klein. Downgesampeltes Bild zu gross und Rechenzeit zu lang. |
| | | 4 | Optimaler Wert, wenn der Kontrast der Objekte nicht besonders stark ist. |
| | | 5 | Optimaler Wert, wenn gute Kontrastverhältnisse vorliegen. Helles Objekt auf dunklem Hintergrund (Lampe) |
| | | 6 | Wert zu gross. Tiefpasswirkung zu stark. Bild verschwommen, deswegen keine guten Ergebnisse mit der Korrelation. |
| Bereichsfenster für Korrelation | BEREICH | $a < 15$ | Die maximal berechenbare Verschiebung ist unter a Pixel. Bei der Referenzkorrelation können grössere Verschiebungen vorkommen. Nur geeignet, wenn sich die WebCam langsam bewegt. |
| | | $a = 15$ | Fenster optimal für normale Bewegungen. 15 Pixel genügen für die Referenzkorrelation. |
| | | $a = 20$ | Optimaler Wert, falls sich die WebCam schneller bewegt. |
| | | $a > 20$ | Einstellung für sehr schnelle Bewegungen. Erhöht die Rechenzeit enorm. |

Tabelle 8.1 Optimierung der Parameter

9 Zeitplan

Es wurde kein genauer Zeitplan erstellt, aber wir setzten drei Meilensteine:

1. Meilenstein 04.12.2003

- Einen oder mehrere funktionierende Algorithmen unter MatLab

2. Meilenstein 08.01.2004

- Das erste GUI, welches Bilder einliest, die Verschiebung ausrechnet und den Vektor ausgibt. Eventuell bereits graphisch. (Algorithmen in C)

3. Meilenstein 25.01.2004

- Endgültiges Programm

4. Meilenstein Ende Semesterarbeit

- Algorithmus auf DSP lauffähig

Für den 4. Meilenstein müsste die Schnittstelle DSP zu DSP-Kamera funktionieren, an welcher in einer anderen parallel laufenden Semesterarbeit gearbeitet wurde. Anfänglich war geplant, dass diese Person an unseren Sitzungen teilnimmt, um die Umsetzung unseres Algorithmus gemeinsam voranzubringen. Da aber die benutzten Daughterkarten relativ neu sind und die Kenntnisse darüber gering, wurde auf die Echtzeitimplementation des Algorithmus auf der Daughterkarte, in Absprache mit Herrn Schuster, verzichtet. Das Erfüllen dieser Zielvorgabe hätte den Rahmen dieser Semesterarbeit gesprengt.

10 Schlussfolgerungen

In dieser SA wurde gezeigt, dass es möglich ist mit einer WebCam die Position des Roboters zu erfassen. Die erarbeitete Lösung um die Bewegung der WebCam resp. des Roboters zu bestimmen, ist eine gute Basis, um in einer späteren SA die Umsetzung auf einen DSP zu realisieren. Es werden sowohl die Translation wie auch die Rotation berechnet und angezeigt. Der gesamte Algorithmus wurde so geschrieben, dass dieser ohne grosse Probleme auf den DSP portiert werden kann. Die Feineinstellungen können aber erst auf dem endgültigen System mit der richtigen WebCAM, dem richtigen DSP usw. vorgenommen werden. Die Erfahrungen haben gezeigt, dass je besser die Kontrastunterschiede der Objekte auf dem Bild sind, desto besser funktioniert die Berechnung. Es gibt noch diverse Möglichkeiten, wie der Algorithmus verbessert werden könnte. Das in Kapitel 4.4 beschriebene Tracking beispielsweise, könnte das Verfahren verbessern. Aus Zeitgründen konnte diese Möglichkeit nicht implementiert werden.

11 Persönliches Fazit

Zu Beginn der Arbeit stürzte man sich in ein beinahe gänzlich unbekanntes Gebiet. Man musste sich nach und nach an das Problem herantasten und den Brocken in übersichtliche Teile unterteilen. Mit zunehmenden Kenntnissen und Erfahrungen wurde die Aufgabe immer klarer, aber auch vielfältiger. Die einzelnen Aspekte wurden immer zahlreicher und es tat sich ein riesiges Betätigungsfeld auf. Doch bis es soweit war, blieb nicht mehr viel Zeit und man musste sich auf das Wesentliche beschränken, obwohl noch monatelang weiter daran gearbeitet werden könnte, bis man das Potential ausgeschöpft hätte. Aber innerhalb der gegebenen Zeit konnte eine Lösung gefunden werden, die funktioniert.

Eine Semesterarbeit gibt einem die Möglichkeit, sich wirklich in ein Thema zu vertiefen, was im normalen Unterricht kaum möglich ist. Man muss auch Kenntnisse diverser Hilfsmittel wie VC++, Matlab etc. erarbeiten, die man in Zukunft sowieso brauchen wird.

Endlich kann man nach einer langen Studienzeit auch seine Fähigkeiten und das Gelernte anwenden. Während der SA müssen die Studenten selber Ansätze und Lösungen suchen. Im normalen Unterricht werden die Lösungen meistens serviert. Die SA ist der erste Schritt Richtung Arbeitswelt. Es war sehr interessant und auch motivierend, selber das Problem der SA lösen zu dürfen. Vielen Dank an Herrn Guido M. Schuster, der uns hervorragend durch diese SA geführt hat.

12 Anhang

12.1 Abbildungsverzeichnis

| | |
|------------------------------------------------------------------------|----|
| Abbildung 4.1 Grobkonzept | 7 |
| Abbildung 4.2 GUI | 8 |
| Abbildung 5.1 Motion Estimation | 11 |
| Abbildung 5.2 DCT-Demo: links Kanten, rechts Komprimierung | 12 |
| Abbildung 5.3 Schwellwertschalter: links Originalbild, rechts s/w-Bild | 13 |
| Abbildung 5.4 Bereichskorrelation | 14 |
| Abbildung 5.5 Korrelation und Bereichskorrelation | 14 |
| Abbildung 5.6 Winkelberechnung | 16 |
| Abbildung 5.7 Rotationsmatrix | 16 |
| Abbildung 5.8 Rasterpunkte | 17 |
| Abbildung 5.9 Pixelberechnung nearest neighbour | 17 |
| Abbildung 5.10 Inverse Abbildung | 18 |
| Abbildung 5.11 Rotation nearest neighbour | 18 |
| Abbildung 5.12 Pixelberechnung Bilinear | 19 |
| Abbildung 5.13 Rotation Bilinear | 19 |
| Abbildung 5.14 Downsampling | 20 |
| Abbildung 5.15 Beispiel Downsampling | 20 |
| Abbildung 6.1 USB WebCam | 23 |
| Abbildung 6.2 WebServer WirelessCam | 24 |
| Abbildung 6.3 Netzwerkeinstellungen WirelessCam | 25 |
| Abbildung 6.4 GDI - Objektklassen | 27 |
| Abbildung 6.5 Device Context Klassen | 28 |
| Abbildung 6.6 Ablauf einer MFC-Anwendung | 29 |
| Abbildung 7.1 GUI | 30 |
| Abbildung 7.2 Verschiebungslinie | 31 |
| Abbildung 7.3 Darstellung Pfeil | 32 |
| Abbildung 7.4 Darstellung Pfeilspitze | 33 |
| Abbildung 7.5 Bild abrunden | 35 |
| Abbildung 7.6 Berechnungsvorgang | 37 |
| Abbildung 7.7 Fahrzeug mit WebCam | 39 |
| Abbildung 7.8 Spannungreglung | 40 |
| Abbildung 7.9 Programmstruktur | 41 |

12.2 Formelverzeichnis

| | |
|-----------------------------------------------|----|
| Formel 5.1 max Verschiebung | 10 |
| Formel 5.2 Verschiebungsvektor | 10 |
| Formel 5.3 Korrelationsfunktion | 12 |
| Formel 5.4 Winkelberechnung | 15 |
| Formel 5.5 Drehwinkel | 15 |
| Formel 5.6 bilineare Berechnung des Farbwerts | 19 |
| Formel 7.1 Start- Endpunkt | 32 |
| Formel 7.2 Punkte des Polygon | 33 |
| Formel 7.3 Bild abrunden | 34 |
| Formel 7.4 Koordinatentransformation | 36 |
| Formel 7.5 Berechnung der Umsetztabelle | 36 |

12.3 Tabellenverzeichnis

| | |
|---------------------------------------|----|
| Tabelle 8.1 Optimierung der Parameter | 42 |
|---------------------------------------|----|

12.4 Quellenverzeichnis

Eurobot www.robotik.com
 Swissbot www.robocup.ch
 Diverse Demoprogramme in MatLab
 Diverse Bilder aus MSDN (Microsoft Developer Network)

12.5 Arbeitsmittel

| Bezeichnung | Hersteller | Version | Link |
|-------------------------|------------------------------|---------|-----------------------------------------------------------------------------------------------------------------|
| WebCam PRO eX | Creative | | www.creative.com |
| WirelessCam DCS-1000W | D-Link | | www.dlink.com |
| MatLab | MathWorks | 6.5 R13 | www.mathworks.com |
| Microsoft Visual C++ | Microsoft | 6.0 | www.microsoft.com |
| vfm (Vision For MatLab) | University of East Anglia UK | | http://www2.cmp.uea.ac.uk/~fuzz/vfm/default.html |

12.6 Glossar

| | |
|--------|-----------------------------------|
| API | Application Programming Interface |
| DCT | Diskrete Cosinus-Transformation |
| DSP | digitaler Signalprozessor |
| GDI | Graphics Device Interface |
| GUI | Graphical User Interface |
| MFC | Microsoft Foundation Class |
| MSDN | Microsoft Developer Network |
| MSVC++ | Microsoft Visual C++ |
| SA | Studienarbeit |
| vfm | Vision for MatLab |