Karlsruhe Institute of Technology

Fraunhofer IOSB

Markerbasiertes Assistenzsystem für 3D-Radarbildsimulation

Bachelorthesis
von

Marcel Finger

Institut für Anthropomatik
Fraunhofer IOSB

Gutachter:	Prof. Dr.-Ing. J. Beyerer
Betreuer:	Dipl.-Inf. Alexander Streicher, Fraunhofer IOSB
	Dipl.-Inf. Anton Berger, Fraunhofer IOSB

Bearbeitungszeit: 15. Mai 2013 – 31. August 2013

Zusammenfassung

Im Rahmen dieser Bachelorthesis wird ein Assistenzsystem entworfen, das auf Basis eines Tangible User Interface den Aufbau einer Szene und deren Manipulation anhand von markierten Stellvertreterobjekte ermöglicht. Die extrahierten Daten werden im 3D-Simulationsprogramm ViSAR ("Visualisierung von geometrischen Radareffekten") dargestellt.

Hierzu wird ein Verfahren zur automatischen visuellen Identifikation von vorgegebenen Stellvertreterobjekten, die mit einem QR-Code markiert sind, entwickelt. Zunächst durchläuft das Kamerabild das Good Features to Track Verfahren. Die so berechneten Keypoints weisen eine hohe Dichte im Bereich der QR-Codes auf. Sie werden mit Hilfe des DBSCAN-Verfahren in Cluster eingeteilt um damit die Region des QR-Codes bestimmen zu können. Diese Regionen werden extrahiert und deren Bildbereiche skaliert damit ZXing den QR-Code dekodieren kann. Zusätzlich wird die Position des QR-Codes in der Ebene bestimmt und an ViSAR übertragen.

Anhand von Experimenten wird in Erfahrung gebracht, dass die Erkennungsrate eines 2,3 cm großen QR-Codes auch unter störenden Einflüssen bei 87% liegt. Zudem wird eine mittlere Laufzeit zwischen 6 und 8 Frames pro Sekunde erreicht.

Aus der beschriebenen Erkennungsrate und Laufzeit lässt sich schließen, dass das Assistenzsystem unter den verlangten Bedingungen betrieben werden kann und zu den gewünschten Ergebnissen führt.

Abstract

Within the framework of this bachelor thesis a assistance system is designed to create and manipulate a scene via substitution objects using a tangible user interface. The extracted data are simulated in ViSAR.
In order to accomplish this quest a method is needed which can identify the qr code placed on the substitution objects. Firstly the good feature to track procedure is used on the camera images. A high density of the found key points indicates a potential substitution object with a qr code. The DBSCAN procedure clusters the points and thus finds the precise position of the qr code on the image. This region has to be extracted and scaled before ZXing can decode the code. Furthermore the precise position of the qr code on the plate is determined and transferred to ViSAR.
Experiments show the rate of identification is 87% even with adverse effects. In addition the average duration counts 6 to 8 frames per second.
Resuming, based on the results of the experiments, the system works under the required conditions.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, 29.08.2013

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das am Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung IOSB entwickelte 3D-Simulationsprogramm ViSAR ("Visualisierung von geometrischen Radareffekten") dient zur Visualisierung von geometrischen Effekten im Bereich der Radarbildauswertung. Hierzu wird von einer zuvor erstellten Szene ein Radarbild simuliert. Dies bietet Radarbildauswertern die Möglichkeit Effekte, die bei der Aufnahme des Radarbildes entstehen, kennenzulernen und so ihre Fähigkeiten in diesem Bereich zu erweitern.

Um die zuvor angesprochenen Fähigkeiten zu verbessern, stehen ein Lehrer und mehrere Schüler vor zwei großen Displays auf denen das Simulationsprogramm ViSAR zu sehen ist. Der Aufbau der Szene wird aktuell durch die Verwendung von Maus und Tastatur bewerkstelligt. Diese Art der Nutzung ist nicht sehr intuitiv. Aus diesem Grund wird in dieser Thesis ein Assistenzsystem entwickelt. Das System ermöglicht es die Szene in ViSAR aufzubauen sowie ihre Manipulation.

Die grundlegende Idee dabei ist, das Objekte, die in ViSAR verwendet werden, durch sogenannte Stellvertreterobjekte in der realen Welt vertreten werden. Durch die Veränderung der Position der Stellvertreterobjekte auf einer Ebene kann der Nutzer die Lage der Objekte in ViSAR beeinflussen. Um die Szene aus allen Blickwinkeln betrachten zu können, ist sie auf einer drehbaren Glasplatte aufgebaut. Die Veränderungen werden durch Kameras erfasst und von dem in dieser Arbeit entwickelten Assistenzsystem namens ViSAR.OBJ verarbeitet um sie in ViSAR anzuwenden. Die Stellvertreterobjekte werden dabei durch einen QR-Code identifiziert. Als Größe für den QR-Code wird 2,5 cm angestrebt.

1.1 Motivation

Seit den 60er Jahren werden Maus und Tastatur als Eingabegeräte für Computer genutzt [Ech09]. Da der Mensch auf Grund seiner Physiologie nicht gewohnt ist auf diese Weise zu arbeiten, muss diese Art der Handhabung zunächst erlernt werden.

Schön wäre es wenn Modellgebäude in der Szene platziert werden, diese dann automatisch vom System erkannt und dem richtigen Objekt in ViSAR zugeordnet werden. Da dies in der aktuellen Situation noch nicht praktikabel ist, wurde das in dieser Thesis vorgestellte Assistenzsystem entwickelt.

In dieser Bachelorthesis wird ViSAR durch ein Tangible User Interface erweitert. Ein Tangible User Interface ermöglicht dem Nutzer eine natürlichere Interaktion mit dem Computer als mit Maus und Tastatur. Dabei werden zur Manipulation des Computers mehrere physische Objekte verwendet. Somit ist es möglich die Szene, die in ViSAR dargestellt wird, nicht wie zuvor mit Maus und Tastatur, sondern in der realen Welt durch sogenannte Stellvertreterobjekte aufzubauen und zu manipulieren. Ein großer Vorteil ist, dass die Szenenkomposition einfacher wird. Es ermöglicht dem Lehrer der Radarbildauswerter sich auf die Wissensvermittlung zu konzentrieren und nicht wie zuvor auf die Bedienung des Systems. Die Szene kann zudem sehr schnell den Bedürfnissen angepasst werden, so dass zum Beispiel auf Fragen bezüglich einer Änderung der Szene und der daraus resultierenden Effekte, welche im SAR-Bild zu sehen sind, didaktisch eingegangen werden kann und die Belastung durch die Bedienanforderung in den Hintergrund rückt.

1.2 Ziele dieser Arbeit

Die Ziele dieser Bachelorarbeit sind die Entwicklung eines Assistenzsystems, das den Aufbau der Szene durch Stellvertreterobjekte sowie ihrer Veränderung in ViSAR ermöglicht. Hierzu wird ein Verfahren zur automatischen visuellen Identifikation von vorgegebenen Stellvertreterobjekten, die mit einem QR-Code markiert sind, entwickelt. Hinzu kommt, dass eine schnelle Erfassung deren Lage auf einer vorgegebenen Ebene ermöglicht wird. Des Weiteren soll das entwickelte Programm robust im Simulationslabor (SimLab) laufen, so dass es bei der Präsentation vor Kunden verwendet werden kann.

1.3 Stand der Forschung und Technik

Die Entwicklung von Assistenzsystemen, oder auch Unterstützungssystemen genannt, gewinnt seit 1990 immer mehr an Bedeutung. Timpe beschreibt “ein Unterstützungssystem als ein informationsverarbeitendes technisches Gebilde, das die Aufgabenerfüllung eines Operateurs in einem Mensch-Maschine-System dadurch fördert, dass es bestimmte, für die Zielerreichung notwendige, Teilaufgaben innerhalb seiner Gesamtaufgabe übernimmt und/oder ausführt” [KPT00].

Einige Assistenzsysteme, wie auch das in dieser Arbeit entwickelte, verwenden für die Identifikation von Objekten künstliche visuelle Marker. Diese Marker werden an den Objekten befestigt, die sich in einer betrachteten Szene befinden. Beim Entwurf der Marker wird darauf geachtet, dass sie durch Algorithmen sehr leicht erkannt und dekodiert werden können. In ihrem Paper geben F. Bergamasco

et al. eine Übersicht über einige Marker (siehe Abbildung 1.3.1) [BAT11]. Des Weiteren stellen sie eine weitere Methode vor, um den in Abbildung 1.3.1 abgebildeten PI-Tag zu erkennen.

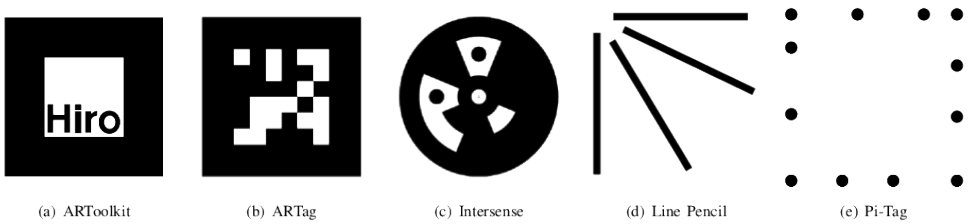


Abbildung 1.3.1: Marker Übersicht [BAT11]

Eine weitere Bibliothek um Marker zu erkennen, wurde von der Universität von Cordoba auf Basis von OpenCV entwickelt [ArU13]. Die ArUco genannte Bibliothek ist aktuell unter der BSD-Lizenz[1] in Version 1.2.4 verfügbar. Durch sie ist es möglich sehr schnell bis zu $2^{10} = 1024$ unterschiedliche Marker zu erkennen. Der in Abbildung 1.3.2 dargestellte Marker besteht aus weißen und schwarzen Feldern, in denen die Informationen durch eine Art Hamming Code codiert sind.

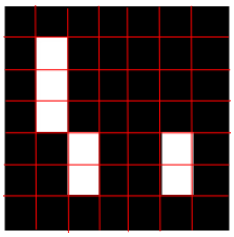


Abbildung 1.3.2: Aufbau eines ArUco Marker bestehend aus weißen und schwarzen Feldern [Swi11]

Eine Möglichkeit Marker zu nutzen fand in der Bachelorthesis von C. Swiontek ihre Anwendung. Hier wurde anhand von aktiven Markern (IR-LEDs) die Position eines Tricopters bestimmt. Dazu wurde das Bild, das von einer Webcam aufgenommen wurde, zunächst in ein Graustufenbild umgewandelt, anschließend geglättet und die interessanten Bildregionen vergrößert. Im Anschluss daran wurde durch OpenCV die Kontur der IR-LEDs bestimmt [Swi11]. In dieser Bachelorthesis wurde zuerst ein ähnliches Verfahren angewendet, um die Regionen der QR-Codes zu finden. Wie in Abschnitt 3.4 beschrieben führt dies jedoch zu keinem brauchbaren Ergebnis.

Des Weiteren werden Assistenzsysteme auch in Industrie bei der Montage eingesetzt. In der Arbeit "Kognitive Assistenzsysteme in der manuellen Montage" wird der Monteur durch ein Assistenzsystem bei den Abläufen der einzelnen Schritte eines Fertigungsprozesses unterstützt. Dabei kommt ein markerbasiertes Infrarottracking zum Einsatz um die Bewegungen der Hand des Arbeiters zu erkennen [MFZ07]. Die eingesetzten Verfahren werden in dem Buch "Static and Dynamic Hand-Gesture Recognition for Augmented Reality Applications" beschrieben [RWA+07].

[1] Die Berkeley Software Distribution oder auch BSD ist eine freie Softwarelizenz [DGG+08].

Eine andere Anwendung befindet sich in der Medizin. C. Westendorff beschreibt in seiner Dissertation "Experimentelle Untersuchungen zur computergestützten Planung und bilddatengestützten Navigation bei enossalen Implantatlagerpräparationen" ein System, welches dem Arzt bei seiner Planung und der darauf folgenden Operation helfen soll. Hierbei werden Marker an bestimmten Stellen der Instrumente und des Patienten befestigt, um zum Beispiel den Arzt während einer Operation mit eingeschränkter Sicht zu unterstützen [Wes06]. Auch M. Brell nutzt in ihrer Dissertation Marker zur Positionserkennung von Instrumenten [Bre09]. In dieser Arbeit werden, wie auch in den zwei zuvor genannten Arbeiten, die Marker zur Positionserkennung, sowie zur Identifikation von realen Objekten genutzt.

Während die Ansätze von C. Westendorff und M. Brell in der Medizin anzusiedeln sind, geht es in der Dissertation "Leistungserhöhung durch Assistenz in interaktiven Systemen zur Szenenanalyse" von E. Peinsipp-Byma um ein Assistenzsystem, das den Menschen bei der Bildauswertung entlastet[PB07]. Dies spielt eine wichtige Rolle im Bereich der Luftbildauswertung, da hier eine sehr große Menge an Daten anfällt. Um nun die Aufgabe des Bildauswerters etwas zu vereinfachen, wurde das System RecceMan entwickelt. Es bietet dem Bildauswerter die Möglichkeit anhand von Luftbildern und bestimmten Merkmalen den Typ eines militärischen Fahrzeugs zu klassifizieren. Dabei werden die Daten, welche für die Erkennung benötigt werden, aus einem festen Datensatz entnommen. Die Masterthesis von N. Bekri verbesserte RecceMan durch die Möglichkeit Objekte zu identifizieren, für die es bislang noch keine speziellen Datensätze gibt [Bek12].

Für eine sachgerechte Auswertung von Satellitenbildern mit dem Ziel Objekte in diesen Bild zu identifizieren, ist bisher eine intensive Ausbildung erforderlich. Zu diesem Zweck wurde am Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung (IOSB) in Karlsruhe der SAR-Tutor entwickelt, der das Lernprogramm ViSAR enthält. Diese Software soll dem Luftbildauswerter die Möglichkeit bieten spezifische Effekte von SAR-Sensoren kennen zu lernen. Hierzu werden die jeweiligen Effekte simuliert und grafisch dargestellt [Pir12]. In ViSAR wird die entsprechende Szene durch Maus und Tastatur aufgebaut und verändert. Zudem kann ViSAR, durch Einbindung einer Microsoft Kinect, mit Gesten gesteuert werden [Bür11].

Diese Bachelorthesis erweitert das System in sofern, dass es möglich ist durch markierte Stellvertreterobjekte eine Szene auf einer Platte aufzubauen. Durch Kameras werden die Stellvertreterobjekte erfasst und in dem hier beschriebenen Programm analysiert. Anschließend werden die extrahiert Koordinaten an ViSAR übertragen.

Tangible User Interface

Wie bereits erwähnt wird in dieser Thesis ein System entwickelt, um das Programm ViSAR zu steuern. Hierzu wird ein Tangible User Interface (TUI) genutzt. In diesem Abschnitt wird auf die grundsätzliche Funktion eines Tangible User Interface eingegangen sowie zwei TUIs vorgestellt.

Grundsätzlich bezeichnet ein Tangible User Interface eine Schnittstelle zwischen dem Benutzer und einem Computer, die es erlaubt den Computer durch physische Objekte zu beeinflussen. Die physischen Objekte werden von dem Computersystem erkannt und durch deren Manipulation kann der Benutzer mit dem Computer interagieren. Hierzu können unterschiedliche Eingabegeräte verwendet werden zum Beispiel Datenhandschuhe oder berührungssensitive Oberflächen. Ebenso können genutzt werden, die durch Bilderkennung oder andere Sensorik erkannt werden. Im Folgenden werden exemplarisch zwei seit 2002 entwickelte Tangible User Interfaces vorgestellt [Koe11]:

SandScape

Die Tangible Media Group des MIT hat das Projekt SandScape 2002 ins Leben gerufen, um Landschaften zu erstellen und zu simulieren [YW03]. Dabei wird eine Vertiefung mit Sand gefüllt und darüber ein Projektor sowie eine Kamera angebracht. Sobald in der Szene eine Veränderung eintritt, zum Beispiel durch Graben eines Loches in den Sand, ändert sich das Bild, das durch den Projektor auf die Szene geworfen wird. Es bildet sich in dem Loch eine blaue Fläche, die Wasser darstellen soll. Diese Situation ist in Abbildung 1.3.3 wiedergegeben.

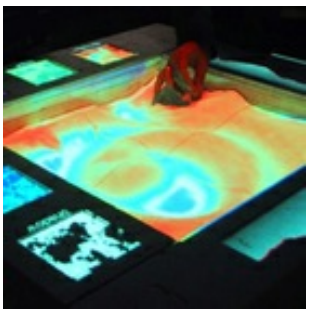


Abbildung 1.3.3: In der Abbildung ist das System, welches im Projekt SandScape entwickelt wurde, zu sehen [YW03].

Reactable

Das Projekt Reactable startete im Februar 2003 und hatte zum Ziel das beste computerbasierte Musikinstrument zu entwickeln [Bar03]. Ein wesentliches Augenmerk wurde auf die intuitive Bedienbarkeit gelegt, um Anfängern die Interaktion und den Systemaufbau zu erleichtern. Zudem sollte das System auch für Komponisten geeignet sein, die es zum Beispiel in einem Konzert einsetzen wollen. Wesentlich dabei ist, dass das System komplett vom Nutzer kontrolliert werden kann und kein zufälliges Verhalten auftritt. Um diese Ziele zu erreichen wurde zunächst ein Konzept entwickelt, welches dann in einem weiteren Schritt umgesetzt wurde. Das Ergebnis ist in Abbildung 1.3.4 zu sehen.

Abbildung 1.3.4: In der Abbildung ist das System, welches im Projekt Reactable entwickelt wurde, zu sehen [Bar03].

1.4 Gliederung

In diesem Teil, der *Einleitung*, wird beschrieben in welchen Gebieten gerade geforscht wird und welche Möglichkeiten sich durch die aktuelle Technik ergeben.

Der folgende zweite Teil geht auf die *Grundlagen* ein, die im weiteren genutzt werden. Es werden die Verfahren aufgezeigt und an Beispielen demonstriert.

Das darauf folgende dritte Kapitel mit dem Namen *Markerbasiertes Assistenzsystems für 3D-Radarbildsimulation*, widmet sich der Beschreibung des entwickelten Verfahrens sowie dem Assistenzsystem.

Das so entwickelte System wird im vierten Kapitel, *Experiment*, analysiert. Dazu zählen die Erkennungsrate, die Auflösung der verwendeten Kameras, die QR-Code Größe sowie die Laufzeit.

Im fünften Teil der These, Szenario, wird das entwickelte Assistenzsystem anhand eines Szenarios veranschaulicht.

Abschließend wird im letzten Kapitel, *Zusammenfassung und Ausblick,* die hier vorgestellte Arbeit zusammengefasst sowie Ideen unterbreitet, die in Zukunft verfolgt werden können.

Kapitel 2

Grundlagen

Im Folgenden werden die grundlegenden Methoden vorgestellt, die im Assistenzsystem ViSAR.OBJ ihre Anwendung finden. Dabei orientiert sich die Reihenfolge der einzelnen Unterkapitel an dem Programmablauf der implementierten Software.

2.1 Good Features to Track (GFTT)

Shi und Tomasi beschreiben in ihrer Veröffentlichung "Good Features to Track" (GFTT) ein Verfahren um Merkmale zu finden, welche sehr gut zu Tracken sind [ST94]. Das vorgestellte Verfahren basiert auf den Arbeiten von Lucas und Kanade "An Iterative Image Registration Technique with an Application to Stereo Vision" und "Detection and Tracking of Point Features" [LK81, TK91].

In dem von Tomasi und Kanade entwickelten Verfahren wird für das Tracking ein reines Verschiebungsmodell verwendet. Da dies nach Ansicht von Shi und Thomasi nicht ausreichend ist, werden bei ihnen auch affine Änderungen in der lokalen Nachbarschaft berücksichtigt. Dazu wird die Verschiebung d, welche den Punkt $\chi = (x, y)$ aus Bild I zu einem anderen Punkt im darauf folgenden Bild J verschiebt, wie folgt berechnet:

$$J(A\chi + d) = I(\chi)$$

wobei

$$A = 1 + D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix} = \begin{bmatrix} 1 + d_{xx} & d_{xy} \\ d_{yx} & 1 + d_{yy} \end{bmatrix}$$

Wird die Deformationsmatrix D auf 0 gesetzt, so spiegelt es das reine Verschiebungsmodell wieder. In den Experimenten zeigte sich, dass das reine Verschiebungsmodell eine höhere Ausfallsicherheit

und Exaktheit aufweist, während das affine Modell besser geeignet ist, um die Qualität der Merkmale zwischen den Bildern zu überwachen.

Um die gut geeigneten Merkmale zu extrahieren, müssen die Parameter A und d so bestimmt werden, dass die folgende Funktion minimiert wird:

$$\epsilon = \int \int_W [J(A\chi + d) - I(\chi)]^2 w(\chi) \mathrm{d}\chi$$

Wird der Wert der durch $w(\chi)$ gewichteten Formel zu groß, so wird der Keypoint verworfen. Wird er hingegen hinreichend klein, kann angenommen werden, dass er sehr gut verfolgbar ist. Falls nur der Wert von d bestimmt werden soll, ist es ausreichend das folgende Gleichungssystem zu lösen:

$$Zd = e \Leftrightarrow \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} d = \begin{bmatrix} g_x \\ g_y \end{bmatrix},$$

wobei g bei der Linearisierung durch die Taylorentwicklung entsteht. Werden nun die Eigenwerte von Z bestimmt so können Rückschlüsse auf das Bild gezogen werden. Zum Beispiel können zwei hohe Eigenwerte Ecken repräsentieren, die bei affinen Transformationen sehr gut zu verfolgen sind. Ein hoher und ein niedriger Eigenwert repräsentieren unidirektionale Texturen. Weitergehende Informationen können den genannten Veröffentlichungen entnommen werden.

Als Beispiel wird das Verfahren an dem Bild von Lena und einem QR-Code verdeutlicht. Dabei wurden, wie in Abbildung 2.1.1 zu sehen, die GFTT-Keypoints durch grüne Punkte dargestellt.

Abbildung 2.1.1: Veranschaulichung des Good Features to Track Algorithmus am Beispiel von Lena (links) und an einem QR-Code (rechts). Die Stellen, die in der Abbildung grün eingezeichnet sind, spiegeln die errechneten GFTT-Keypoints wieder.

OpenCV

In dieser Bachelorthesis wird für den Zugriff auf die Kameras, die Interpolation der Bilder sowie für das GFTT-Verfahren Open Source Computer Vision (OpenCV [Bra00]) genutzt. Dies ist eine quelloffene Programmbibliothek, welche unter der BSD-Lizenz[1] verfügbar ist und viele Algorithmen für die Bildverarbeitung und das Maschinelle Sehen bereitstellt.

Die Bibliothek ist in C++ geschrieben und bietet neben C++, C und Python Interfaces auch die Anbindung an Java. Des Weiteren kann sie auf unterschiedlichen Plattformen wie Windows, Linux, Mac und Android genutzt werden.

Die Algorithmen sind in mehrere Module unterteilt. Von diesen werden über den JavaCV-Wrapper folgende verwendet:

- core-Modul: Beinhaltet die grundlegenden Datenstrukturen und Funktionen von OpenCV.
- imgproc: Bildbearbeitungsmodul für lineare und nicht lineare Bildfilterung, geometrische Transformation, Farbraumkonvertierung und Histogramme.
- calib3d: Basisalgorithmen für Multiple-View Geometrie, Einzel- und Stereokamera Kalibrierung.

JavaCV

Die Java Schnittstelle, welche von OpenCV bereitgestellt wird, beinhaltet nicht alle benötigten Funktionen. Aus diesem Grund wird JavaCV[2] eingesetzt. Sie steht aktuell in der Version 0.5 bereit und ist unter der GNU GPL v2 Lizenz[3] verfügbar.

2.2 Density-based Clustering

Clustering ist im Bereich der Klassifikation von Objekten ein sehr wichtiges Hilfsmittel. Es wird unter anderem dazu verwendet Daten in großen Geodatenbanken einzuteilen. Dies ist zum Beispiel bei der Klassifikation von Häusern in Satellitenbildern hilfreich. Zu diesem Zweck wurde DBSCAN (Density Based Spatial Clustering of Applications with Noise) entwickelt [EKSX96]. Durch das Verfahren ist es möglich eine Menge von Punkten in eine Menge von ähnlichen Objekte, sogenannte Cluster, einzuordnen. Dabei kann es vorkommen, dass es Punkte gibt, die keinem Cluster zugeordnet werden können. Diese werden als Rauschen oder Noise bezeichnet.

Die grundlegende Idee ist, dass jeder Punkt eines Clusters von einer bestimmten minimalen Anzahl (MinPts) von Punkten, die in einer $\epsilon - Nachbarschaft$ liegen, umgeben ist. Dabei unterscheiden die

[1] Die Berkeley Software Distribution oder auch BSD ist eine freie Softwarelizenz [DGG+08].
[2] http://code.google.com/p/javacv/
[3] Die General Public Licence oder auch GPL genannte Lizenz ist eine freie Softwarelizenz [DGG+08]

Autoren zwei Arten von Punkten; Kernpunkte (core points) und Randpunkte (border points). Diese müssen unterschiedlich behandelt werden. Für Kernpunkte gilt, dass in ihrer $N_\epsilon = \epsilon - Nachbarschaft$ mit

$$N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$$

eine bestimmte Anzahl (MinPts) an Nachbarpunkten überschritten wird, so dass gilt:

$$|N_\epsilon(q)| \geq MinPts$$

Dabei beschreibt D eine Menge von Punkten deren Euklidische Distanz $d(p,q)$ der oben beschrieben Gleichung entspricht.

Für Randpunkte gilt die $\epsilon - Nachbarschaft$ nicht, sie müssen nur innerhalb einer $\epsilon - Nachbarschaft$ eines Kernpunktes liegen. Dieser Sachverhalt wird durch Abbildung 2.2.1 verdeutlicht. Es ist eine Menge von Punkten zu sehen und für vier Punkte wurde die $\epsilon - Nachbarschaft$ eingezeichnet. Wenn zum Beispiel $MinPts = 5$ wäre, so sind R_1 und R_2 die zwei äußeren Randpunkte wobei K_1 und K_2 Kernpunkte repräsentieren.

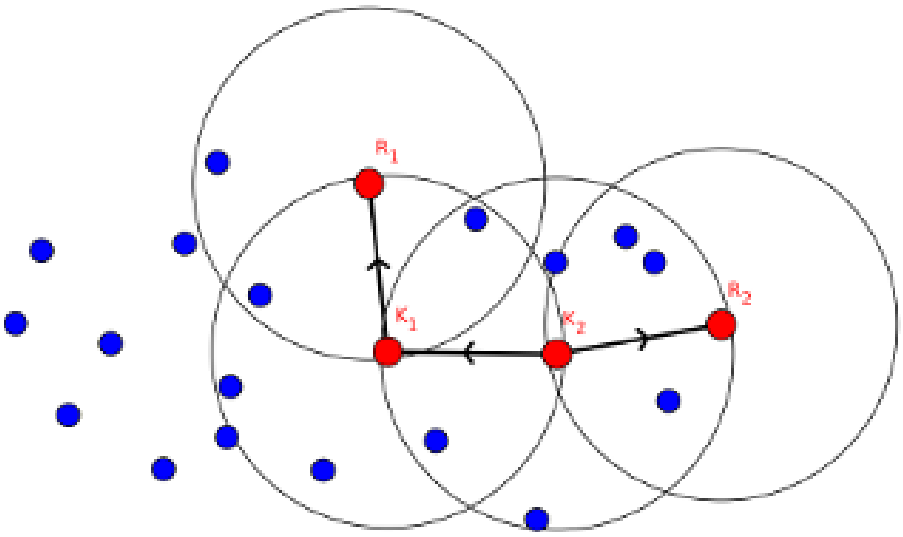


Abbildung 2.2.1: Die Grafik zeigt die $\epsilon - Nachbarschaft$ von vier Punkten (angelehnt an [EKSX96]).

Des Weiteren definieren M. Ester et al. in ihrer Arbeit Cluster und Noise wie folgt:

Definition 1. Sei D eine Datenmenge von Punkten. Ein Cluster C mit ϵ und $MinPts$ ist eine nicht leere Untermenge von D welche die folgenden Bedingungen erfüllt:

- $\forall p, q$: Falls $p \in C$ und q ist Dichte-erreichbar von p mit ϵ und $MinPts$, dann ist auch $q \in C$ (Maximalität).
- $\forall p, q \in C : p$ ist Dichte-verbunden zu q mit ϵ und $MinPts$ (Konnektivität).

Definition 2. Seien $C_1, ..., C_k$ die Cluster der Datenbasis D mit den Parametern ϵ_i und $MinPts_i$, $i = 1, ..., k$. Dann ist Rauschen definiert als eine Menge von Punkten in der Datenbasis D, welche zu keinem der Cluster C_i gehören:

$$Rauschen = \{p \in D | \forall i : p \notin C_i\}$$

In dieser Bachelorthesis werden die Punkte, die durch das in Abschnitt 2.1 beschriebene Verfahren erzeugt wurden, mit DBSCAN in Cluster eingeordnet. Dieser Algorithmus weist eine mittlere Komplexität von $O(n*log(n))$ auf und liefert gute Ergebnisse. Das heißt, dass alle Keypoints eines QR-Codes einem Cluster zugehörig sind. Um das Verfahren zu veranschaulichen, wurde es auf eine Menge von Punkten angewendet (siehe Abbildung 2.2.2). Dabei zeigt sich im dritten Beispiel, dass die vereinzelt auftretenden Punkte zu keinem Cluster gehören.

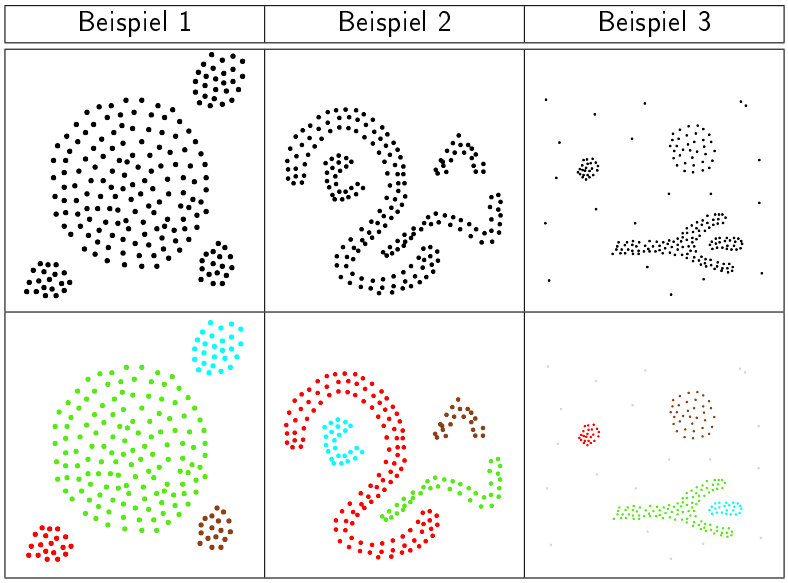


Abbildung 2.2.2: Veranschaulichung des DBSCAN Verfahrens an 3 Beispielen. Zusehen sind eine Menge von Punkten die durch das DBSCAN Verfahren geclustert wurden. Dabei zeigt sich im dritten Beispiel, dass die vereinzelt auftretenen Punkte zu keinem Cluster gehören [EKSX96].

java-statistical-analysis-tool

Eine Java-Bibliothek, die diesen Algorithmus zur Verfügung stellt, ist die von Edward Raff entwickelte Software *java-statistical-analysis-tool*. Die unter GNU GPL v3[4] stehende Bibliothek implementiert

[4]Die General Public Licence oder auch GPL genannte Lizenz ist eine freie Softwarelizenz [DGG+08]

zusätzlich die in Tabelle 2.1 zusammengefassten Clustering Algorithmen [Raf13].

PAM	CLARA	k-Means	Mini-Batch k-Means	EM-Gaussian Mixture	DBSCAN
LSDBC	OPTICS	Mean Shift		HAC	

Tabelle 2.1: Übersicht der Implementierten Clustering Algorithmen von java-statistical-analysis-tool [Raf13].

2.3 Projektive Verzerrungsabbildungen

Wird mit einer Kamera eine Szene aufgenommen und dann die Position der Kamera verändert und die selbe Szene erneut aufgenommen, so entstehen Bilder, die die Objekte der Szene aus unterschiedlichen Blickwinkeln darstellen. Die Überführen vom ersten zum zweiten Bild werden als projektive Verzerrungsabbildungen bezeichnet. Sie bestehen aus Kombinationen von Rotations-, Verzerrungs- und Skalierungsabbildungen [AGD07].

Allgemeine Definition

Seien die Koordinaten $x = (u, v)$ eines beliebigen Bildpunktes gegeben, können die neuen Koordinate des zweiten Bildes $x' = (u', v')$ berechnet werden. Hierzu wird die ursprüngliche Koordinate mit der Transformationsmatrix multipliziert:

$$x' = Ax \leftrightarrow \begin{bmatrix} u's \\ v's \\ s \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Je nachdem wie die Matrix A aufgebaut ist, lässt sich die Rotation, Verzerrung und Skalierung eines beliebigen Bildes beschreiben.

In dem, in dieser Bachelorarbeit, relevanten Fall ist eine Skalierung erforderlich um die QR-Codes mit ZXing zu dekodieren. Wendet man hierzu die oben stehende Abbildung A auf ein Bild an, so würden im resultierenden Bild Stufeneffekte auftreten. Um diese zu verringern kann zwischen den Werten, für die keine Information im Originalbild vorhanden sind, interpoliert werden [AGD07]. Die gängigsten und in OpenCV implementierten Verfahren werden im Folgenden kurz erklärt.

Nearest-Neighbor Interpolation

Die Nearest-Neighbor Interpolation oder auch Pixelwiederholung genannte Methode ordnet dem aktuell betrachteten Pixel immer den nächstgelegenen Pixel im Originalbild zu. Bei der Vergrößerung des Bildes entstehen somit größere einfarbige Blöcke. Das Verfahren führt bei der Verkleinerung des Bildes zu Aliasing-Effekten.

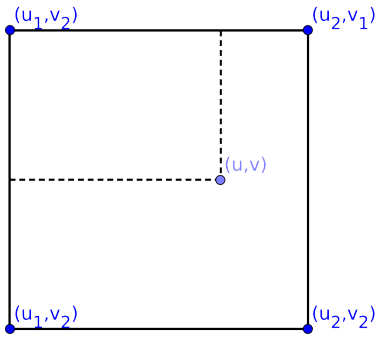


Abbildung 2.3.1: Veranschaulichung der Koordinaten für die bilineare Interpolation (angelehnt an [AGD07]).

Bilineare Interpolation

Bei der bilinearen Interpolation wird jeder Pixel aus den vier benachbarten Pixeln des Originalbilds berechnet. Das Pixel mit den Koordinaten u, v im resultierenden Bild I berechnet sich aus den folgenden Koordinaten des Originalbildes:

$$u_1 := \lfloor u \rfloor, \quad u_2 := u_1 + 1 \ \wedge \ v_1 := \lfloor v \rfloor, \quad v_2 := v_1 + 1$$

Diese vier Koordinaten, wie in Abbildung 2.3.1 zu sehen, stellen die Eckpunkte im Originalbild da. Des Weiteren wird noch die folgende Differenz benötigt:

$$\Delta u := u - u_1 \ \wedge \ \Delta v := v - v_1$$

Nun kann der Wert $I(u, v)$ des skalierten Bildes mit der folgender Formel berechnet werden:

$$I(u,v) = \begin{bmatrix} 1 - \Delta u & \Delta u \end{bmatrix} \begin{bmatrix} I(u_1, v_1) & I(u_1, v_2) \\ I(u_2, v_1) & I(u_2, v_2) \end{bmatrix} \begin{bmatrix} 1 - \Delta v \\ \Delta v \end{bmatrix}$$

Wendet man das Verfahren nun auf das zu Anfang gezeigte Originalbild an, so entsteht das in Abbildung 2.3.3 gezeigte Bild [AGD07].

Bikubische Interpolation

Bei der bikubischen Interpolation wird jeder neue Pixel durch eine 4 x 4 Nachbarschaft des Originalbildes berechnet. Die lokale Werte werden durch die Approximation eines bikubischen Polynom berechnet. Allgemein kann man die bikubische Interpolation wie folgt beschreiben:

$$I_{u,v} = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}u^i v^j = a_{00} + a_{10}u + a_{01}v + a_{20}u^2 + a_{11}uv + a_{02}v^2 + a_{21}u^2v + a_{22}u^2v^2$$

$$+a_{30}u^3 + a_{03}v^3 + a_{31}u^3v + a_{13}uv^3 + a_{32}u^3v^2 + a_{23}u^2v^3 + a_{33}u^3v^3$$

Wie in Abbildung 2.3.2 zu sehen, ergibt sich ein Netz, bei dem in beide Richtungen u und v der Gradient bestimmt werden muss. Daraus resultieren 16 Gleichungen mit 16 Koeffizienten (a_{ij}) [AT07].

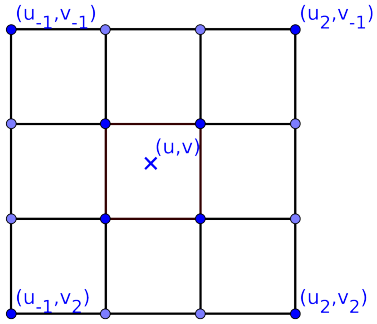


Abbildung 2.3.2: Veranschaulichung der Koordinaten für die bikubische Interpolation.

Lanczos Interpolation

Bei der Lanczos Interpolation können entweder 4 x 4, 6 x 6 oder 8 x 8 Nachbarschaften betrachtet werden. OpenCV nutzt die 8 x 8 Nachbarschaft die im Folgenden erklärt wird.

Dabei ist die Funktion $K(t)$ wie folgt definiert:

$$K(t) = \begin{cases} sinc(t)sinc(\frac{t}{4}) & |t| < 4 \\ 0 & sonst \end{cases}$$

Um nun einen Wert an der Stelle u, v zu interpolieren, kann folgende Formel verwendet werden:

$$I(u,v) = \sum_{m,n} v_{m,n}K(u-m)K(v-n),$$

wobei v die Elemente des Originalbilds widerspiegelt [Get11].

Abbildung 2.3.3: Vergleich der einzelnen Interpolationsverfahren.

2.4 Barcodes

In dieser Bachelorthesis werden Barcodes, speziell QR-Codes, für die Identifikation von Objekten sowie für die Bestimmung deren Lage im Raum genutzt.

Durch Barcodes oder auch Strichcodes können numerische und alphanumerische Zeichen codiert werden. Sie bestehen aus unterschiedlich langen und breiten Strichen, in denen die Information enthalten ist. Um diese Information wieder abzurufen sind Scan-/Lesegeräte erforderlich.

Alle im weiteren Verlauf beschriebenen Codes sind in Abbildung 2.4.1 zusammengefasst.

1D - Codes

Das grundlegende Prinzip der Barcodetechnologie wurde im Jahr 1949 entwickelt. In den folgenden Jahren wurden durch sinkende Preise der elektronischen Bauelemente die Leser immer preiswerter. Dies führte zu einer Steigerung in der Nutzung dieser Technik. Die Tabelle 2.2 zeigt eine Übersicht über die ersten entwickelten Barcodes und deren Anwendungsbereiche.

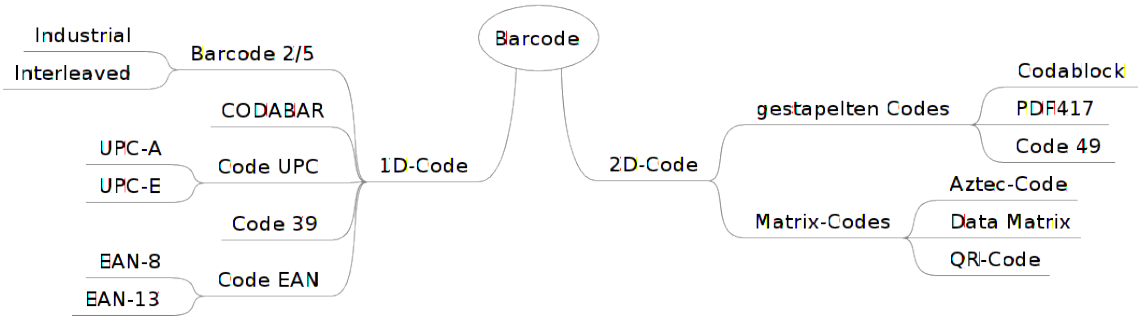


Abbildung 2.4.1: Übersicht Codes

Barcode	Jahr	Anzahl an Zeichen	Zeichensatz	Anwendungsbereich
Barcode 2/5 Industrial	1968	variabel	Ziffern 0 bis 9	Fördertechnik
Code 2/5 interleaved		variabel (gerade Anzahl)	Ziffern 0 bis 9	Produktion, Paketdienste, Lebensmittel Branche, Elektronik Industrie, Pharma-Industrie, Transport und Logistik
CODABAR	1972	variabel	Ziffern 0 bis 9, 6 Sonderzeichen	
Code UPC (Universal Product Code)	1973	12 Stellen	Ziffern 0 bis 9, Zwei Zeichensätze (A und C)	Handel
Code 39	1974	variabel	Ziffern 0 bis 9, 26 Buchstaben, 7 Sonderzeichen 1 Leerzeichen	Produktion, Paketdienste, Elektronik Industrie, Automobil Industrie, Transport und Logistik
Code EAN (European Article Numbering) (EAN 13)	1976	13 Stellen	Ziffern 0 bis 9, Drei Zeichensätze	Produktion, Lebensmittel Branche, Einzelhandel, Elektronik Industrie, Transport und Logistik

Tabelle 2.2: Barcode Übersicht [PJ93]

In Tabelle 2.2 sind einige Anwendungsbereiche aufgeführt in denen Barcodes zum Einsatz kommen. Um Objekte zu markieren werden unterschiedliche Technologien verwendet. Zum einen kann der Barcode durch das Ausdrucken auf Papier erzeugt werden, zum anderen kann er auf einem Bildschirm angezeigt werden.

Um den Barcode wieder aus dem jeweiligen Medium auszulesen werden Lesegeräten verwendet. Das Prinzip beruht im Normalfall auf einem gebündelten Lichtstrahl, der über den Barcode geführt wird.

Dieser reflektiert das Licht, wobei die weißen Stellen stärker reflektieren. Ein lichtempfindliches Bauelement registriert das empfange Licht, welches dann ausgewertet wird.

Eine weitere Möglichkeit ist, den Barcode optisch als Bild aufzunehmen und dieses dann zu analysieren. Hierzu muss der Barcode nur in einen Scanner gelegt oder von einer Kamera oder einem Smartphone abfotografiert werden [PJ93].

2D-Codes

Bei 2D-Codes bzw. zweidimensionalen Barcodes können Informationen nicht nur wie bei 1D-Barcodes horizontal codiert werden, sondern auch vertikal (siehe Abbildung 2.4.2). Somit können Informationen sehr kompakt abgelegt und die Größe des Barcodes verkleinert werden.

Abbildung 2.4.2: Der 1D Barcode sowie der 2D Barcode enthalten den gleichen Inhalt [WAV13].

Es werden echte Matrix-Codes von gestapelten Codes unterschieden. Letztere werden durch das aneinanderdrucken von eindimensionalen Barcodes erreicht. Zu ihnen zählen Codablock, PDF417 und Code 49 die in Abbildung 2.3 abgedruckt sind.

Barcode	Grafik	Anwendungsgebiete
Codablock		Pharma-Industrie
PDF417		Transport und Logistik
Code 49		

Tabelle 2.3: Übersicht gestapelte Barcodes [IBM13, TI13]

Bei den echten Matrix-Codes spielt im Gegensatz zu den gestapelten Codes die Leserichtung keine Rolle. Dieses bietet Vorteile in der Handhabung, weil ein mit diesem Code gekennzeichnetes Objekt unabhängig von seiner Lage erfasst werden kann. Zu den gebräuchlichen Matrix-Codes zählen der QR-Code, die DataMatrix, der MaxiCode und der Aztec-Code welche in Abbildung 2.4 abgedruckt sind.

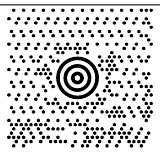
Barcode	Grafik	Anwendungsgebiete
QR-Code		
DataMatrix		Produktion, Elektronik Industrie, Pharma-Industrie
MaxiCode		Paketdienste, Transport und Logistik
Aztec-Code		

Tabelle 2.4: Übersicht der echten Matrix-Codes (erstellt mit [TI13])

QR-Code

QR-Codes sind ein zentraler Bestandteil dieser Thesis da sie zur Identifikation der Stellvertreterobjekte sowie für die Bestimmung deren Lage eingesetzt werden. Ein QR-Code ist ein zweidimensionaler Barcode der von der Firma *DENSO WAVE INCORPORATED* entwickelt wurde. Die Idee ist, wie zuvor schon erwähnt, Informationen nicht nur horizontal sondern auch vertikal zu codieren.

Somit ist es möglich eine sehr hohe Kapazität (bis zu 7089 Ziffern, 4296 Zeichen) auf einer kleinen Fläche zu erreichen. Des Weiteren bietet der QR-Code die Möglichkeit *verschmutzte* oder *beschädigte Inhalte* zu rekonstruieren. Zudem kann der QR-Code aus jeder Richtung gelesen werden und es ist möglich die Lage des Barcodes in der Ebene zu bestimmen. In Tabelle 2.4.3 ist eine Übersicht von QR-Codes zu sehen.

Die in dieser Arbeit verwendeten QR-Codes besitzen 21 x 21 Module und sind unter dem Namen QR-Code 2005 bekannt, der in der Norm ISO/IEC 18004:2006 beschrieben wird. Um den QR-Code in Bildern aufzufinden kann das in Abbildung 2.4.4 gezeigte Suchmuster verwendet werden. Die drei großen Quadrate dienen zusätzlich der Lagebestimmung. Für den Fall, dass die Barcodes verschmutzt oder beschädigt sind, kann durch Reed Solomon auf Datenebene eine gewisse Anzahl an Daten wiederhergestellt werden [Len12].

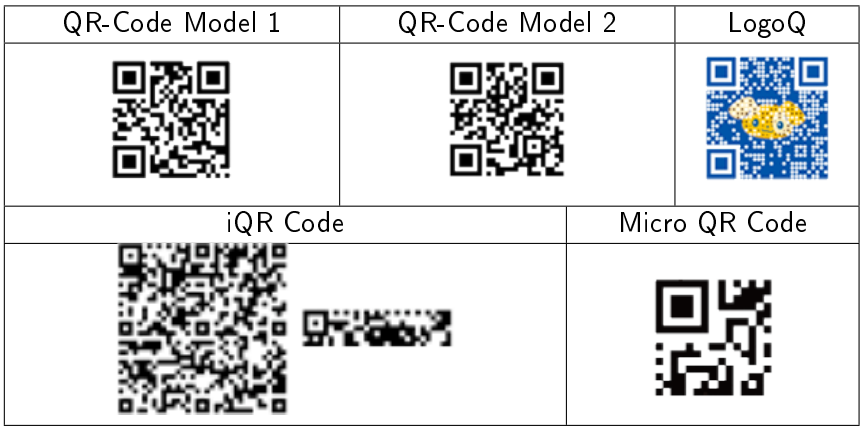


Abbildung 2.4.3: Übersicht über die Typen von QR-Codes [WAV13]

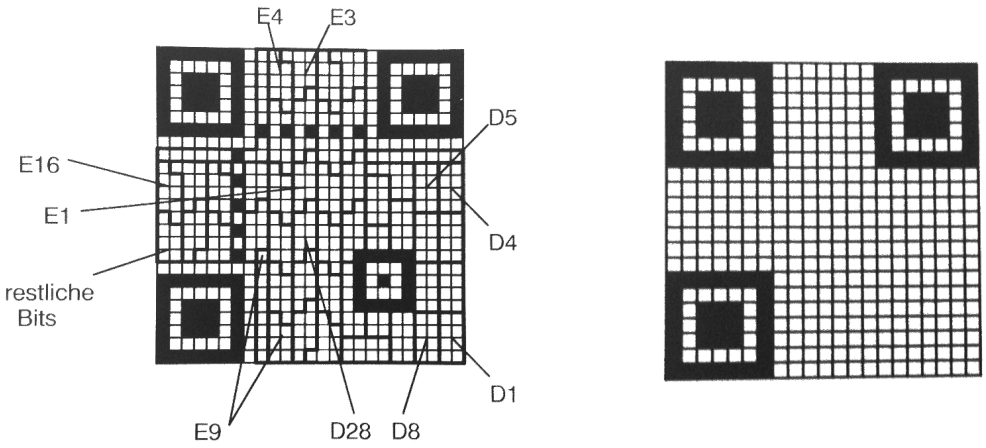


Abbildung 2.4.4: Aufteilung der Codeworte (links) und Suchmusters (rechts) des QR Code 2005 [Len12].

ZXing ("Zebra Crossing")

ZXing ist eine freie Bibliothek um 1D und 2D Barcodes zu verarbeiten. Sie ist in Java implementiert und bietet Schnittstellen für weitere Sprachen, wie zum Beispiel C++ und C# an. Als Eingabe dient meist eine Kamera, wie sie zum Beispiel auch in Smartphones zu finden ist. In dieser Thesis wird eine Netzwerkkamera verwendet. Des Weiteren kann ZXing auch genutzt werden, um Barcodes zu erzeugen. Dabei unterstützt die Software alle Formate, die in Tabelle 2.5 aufgeführt sind [5].

[5] http://code.google.com/p/zxing/

UPC-A and UPC-E	Code 93	Codabar	Data Matrix
EAN-8 and EAN-13	Code 128	RSS-14 (all variants)	Aztec ('beta' quality)
Code 39	ITF	RSS Expanded (most variants)	PDF 417 ('alpha' quality)
		QR Code	

Tabelle 2.5: ZXing unterstützte Formate

Kapitel 3

Markerbasiertes Assistenzssystemes für 3D-Radarbildsimulation

In diesem Kapitel wird das Assistenzsystem ViSAR.OBJ, sowie das Verfahren zur Erkennung und Extraktion der Koordinaten, die an ViSAR gesendet werden, vorgestellt. Zunächst wird ein Überblick über den Aufbau gegeben. Im Anschluss daran, wird auf das Verfahren und die einzelnen Bestandteile des Systems eingegangen. Ein Handbuch, welches das implementierte Programm beschreibt, ist im Anhang zu finden.

3.1 Aufbau

Das Assistenzsystem zur Manipulation der in ViSAR angezeigten Objekte besteht aus drei Komponenten, welche in Abbildung 3.1.1 schematisch dargestellt sind.

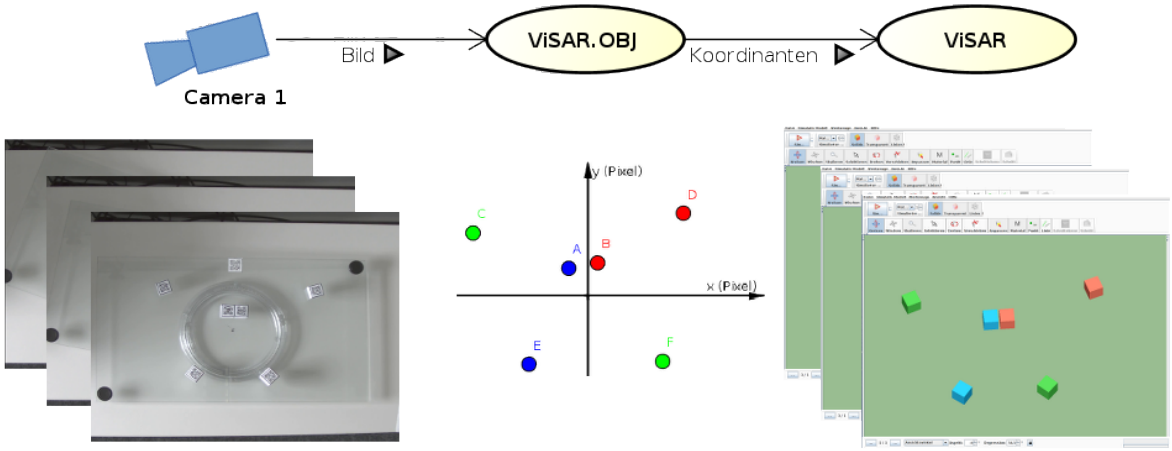


Abbildung 3.1.1: Der Aufbau des gesamten Systems.

Die Szene, welche in der Abbildung 3.1.1 zu sehen ist, befindet sich auf einer drehbaren Glasplatte. Es sind die Stellvertreterobjekte mit ihren QR-Codes erkennbar, ein weiterer QR-Code (oben in der Mitte) kennzeichnet bzw. repräsentiert die Glasplatte. Dadurch ist es möglich die Drehung der Platte zu verfolgen und das Koordinatensystem in ViSAR um diesen Aspektwinkel zu drehen.

Die Szene wird von einer Kamera, welche an der Decke angebracht ist, erfasst. Die erzeugten Bilder werden dann von dem hier entwickelten Assistenzsystem ViSAR.OBJ abgegriffen und verarbeitet. Nachdem die Objekte, anhand ihrer QR-Codes erkannt wurden, kann ihre Lage im Bildraum bestimmt werden. Bevor ihre Lage an ViSAR übertragen wird, erfolgt eine Umrechnung in das Koordinatensystem von ViSAR. Der beschriebene Ablauf ist in Abbildung 3.1.2 dargestellt.

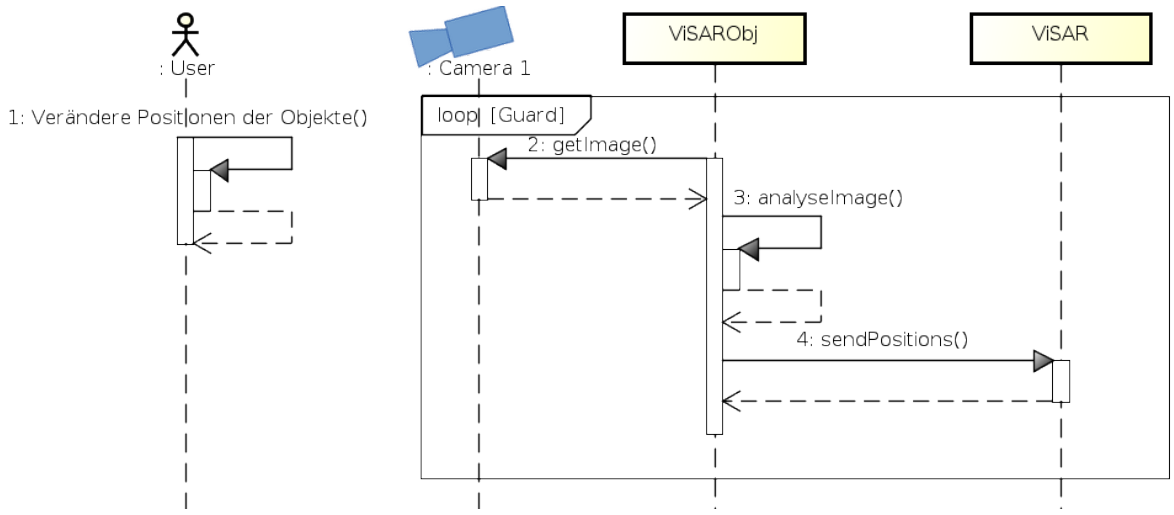


Abbildung 3.1.2: Interaktion des Benutzers mit dem Gesamtsystem.

3.2 ViSAR.Obj

Das Programm ViSAR.Obj ist in fünf Pakete unterteilt (siehe Abbildung 3.2.1). Sie erfüllen dabei jeweils unterschiedliche Aufgaben und sind durch die Klasse ViSAROBJ miteinander verbunden. In den nächsten Abschnitten wird jeweils auf die Pakete näher eingegangen und deren Aufgabe und Funktion beschrieben. Dabei orientiert sich die Reihenfolge an dem Programmablauf (siehe auch Abschnitt 2).

Bildbereitstellung

Zunächst müssen die Bilder bereitgestellt werden. Dies wird durch das Paket *camera* ermöglicht. Es enthält das Interface CameraContoller, welches von dem MonoCameraController implementiert wird. Dieser ist so gestaltet, dass er über das Beobachtermuster (Observer) alle angemeldeten Klassen

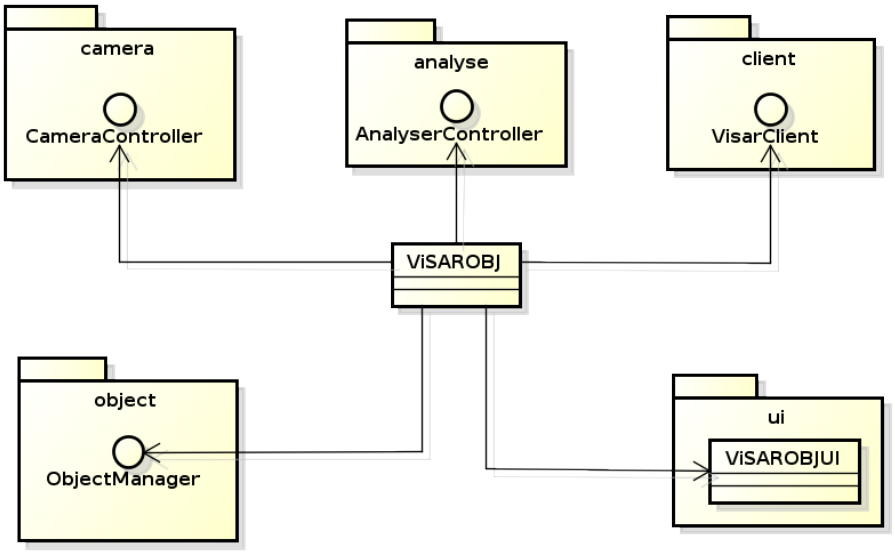


Abbildung 3.2.1: Übersicht der verwendeten Pakete und Schnittstellen.

benachrichtigt, sobald ein neues Bild von der Kamera vorliegt. Der Ablauf ist in Abbildung 3.2.2 veranschaulicht. Des Weiteren dient der MonoCameraController auch als Fassade um den Zugriff auf die Kameras zu kapseln und zu vereinfachen.

In dieser Thesis wird nur eine Kamera über den MonoCameraController angesprochen, da dies für die gegebene Aufgabe ausreicht. Falls jedoch in Zukunft das hier entwickelte Assistenzsystem erweitert werden soll und eine zweite Kamera verwendet wird, kann der StereoCameraController zum Einsatz kommen.

Der CameraController stellt nicht nur die Bilder der Kamera zur Verfügung, sondern ermöglicht es auch diese anhand von Kalibrierungsparametern, zu transformieren. Hierzu wurde die Möglichkeit geschaffen Kalibrierungsparameter einzulesen. Diese können durch das Tool Calib3d[1] erzeugt werden.

Ist ein neues Bild der Kamera verfügbar, so wird die Klasse ViSAROBJ benachrichtigt. Das in Abbildung 3.2.3 gezeigte Kamerabild dient als Eingabebild für das entwickelte System und alle folgenden Verfahren werden auf den daraus erzeugten Daten angewandt.

Extraktion der QR-Codes und deren Analyse

Das Paket *analyse* stellt eine Schnittstelle zur Analyse der aufgenommenen Bilder bereit. Hier werden die Bilder vorverarbeitet und in einem weiteren Schritt analysiert. Sobald die QR-Codes extrahiert und dekodiert wurden, werden sie durch den Resolver mit ViSAR-Objekten verknüpft.

[1] Calib3d ist in den Beispielen von OpenCV zu finden (http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html).

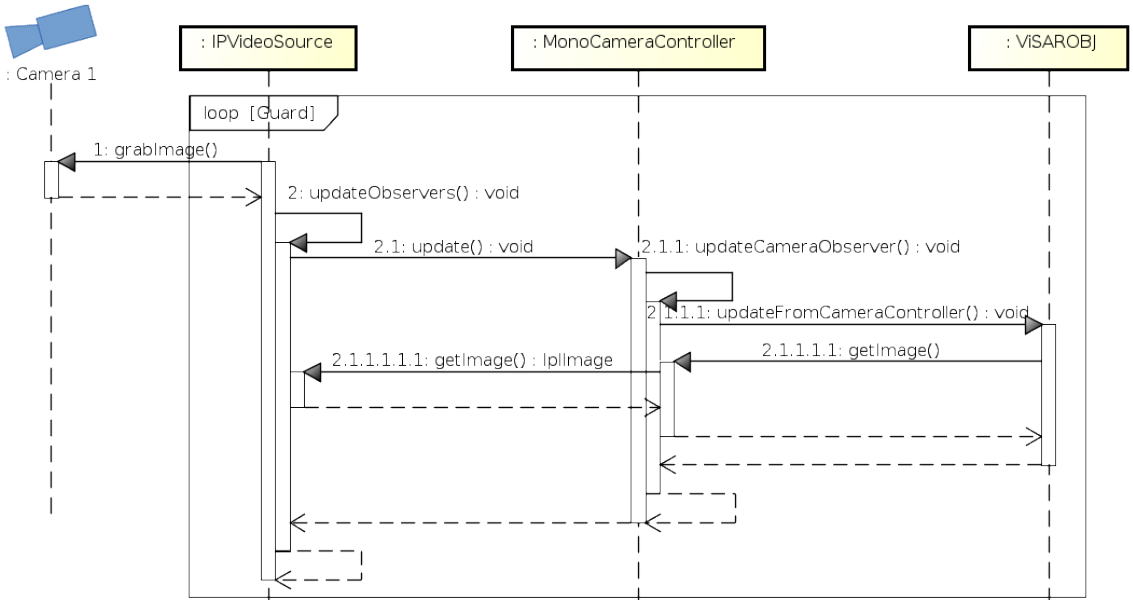


Abbildung 3.2.2: Der Ablauf für die Bereitstellung eines Bildes.

Soll nun das in Abbildung 3.2.3 gezeigte Kamerabild verarbeitet werden, sind im nächsten Schritt die relevanten Bildausschnitte zu extrahiert. Dies geschieht in der Vorverarbeitung. In der Vorverarbeitung durchläuft das Bild zunächst das Good Features To Track Verfahren, welches in Abschnitt 2.1 beschrieben wurde. Das Resultat ist in Abbildung 3.2.4 zu sehen. Es zeigt, dass es bei allen QR-Codes eine sehr hohe Dichte an GFTT-Keypoints gibt. Dies resultiert aus dem Umstand, dass die QR-Codes viele Ecken aufweisen, welche wie zuvor beschrieben, sehr gut zu Tracken sind.

Die so extrahierten GFTT-Keypoints werden in einem weiteren Schritt durch das DBSCAN Verfahren geclustert. DBSCAN, wie in Abschnitt 2.2 beschrieben, arbeitet auf Basis von $MinPts$ und ϵ. Dabei wird auch der Umstand einbezogen, dass ein Rauschen vorhanden ist. Dies ist in Abbildung 3.2.4 auf dem Drehkranz der Glasplatte zu sehen. Durch DBSCAN werden nun Kernpunkte und Randpunkte ermittelt und die GFTT-Keypoints in Cluster eingeteilt. Wobei die zuvor angesprochenen GFTT-Keypoints auf dem Drehkranz der Glasplatte als Rauschen interpretiert werden. Hierfür müssen jedoch $MinPts$ und ϵ passend gewählt sein. Während der Arbeit stellte sich heraus, dass die Werte $MinPts = 10$ und $\epsilon = 20$ am besten dazu geeignet sind die Keypoints durch DBSCAN zu gruppieren.

Die in Abbildung 3.2.4 dargestellten GFTT-Keypoints sind mit diesem Verfahren in Cluster eingeteilt (siehe Abbildung 3.2.5). Es ist zu erkennen, dass durch die passende Wahl der Parameter nur die GFTT-Keypoints auf den QR-Codes übrig geblieben sind.

In einem weiteren Schritt wird die Region der QR-Codes aus dem Originalbild herausgeschnitten. Bevor jedoch die Teilbilder herausgeschnitten werden, wird von jedem Cluster der Median der X und Y Koordinate berechnet. Dadurch wird verhindert, dass Ausreißer die ausgeschnitten Bereiche

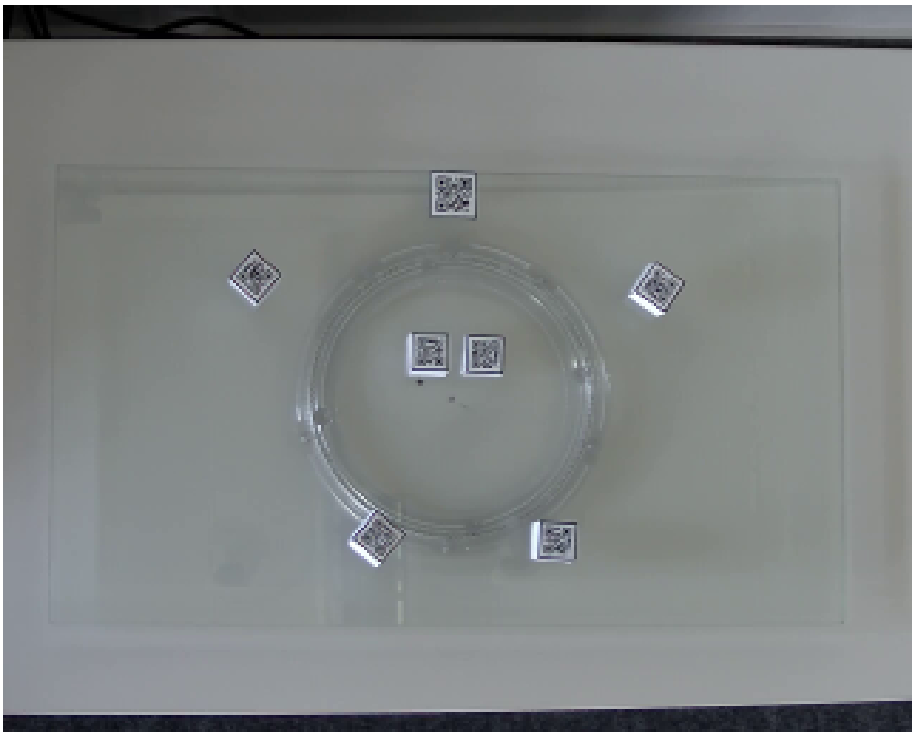


Abbildung 3.2.3: Kamerabild welches aus dem Modul camera an die Klasse ViSAR.OBJ geliefert wird.

verfälschen.

Diese so heraus getrennten Teilbilder, welche in Tabelle 3.1 zu sehen sind, werden mit der Basiskoordinate weiter verarbeitet. Die Basiskoordinate ist die obere linke Koordinate des Teilbildes. Diese sind notwendig um nachher die Position der QR-Codes im Originalbild zu ermitteln.

Nun müssen die QR-Codes analysiert werden. Hierzu werden die einzelnen QR-Codes skaliert. Ohne eine Skalierung wird von ZXing kaum ein QR-Code erkannt. Der Grund hierfür sind die kleinen Strukturen des QR-Codes. Ein QR-Code ist zum Beispiel bei einer Auflösung von 1280 x 1024 Pixeln nur ca. 60 Pixel groß. Der verwendete QR-Code-Leser, der in Abschnitt 2.4 vorgestellt wurde, nutzt ein Verfahren, welches mit den Verhältnissen zwischen schwarzen und weißen Modulen arbeitet. Bei kleinen QR-Codes ist der Kontrast zwischen den schwarzen und weißen Modulen nicht besonders hoch, wodurch die QR-Codes nicht gut bestimmbar sind. Aus dem zuvor genannten Grund müssen die Teilbilder skaliert werden. Hierzu wurden alle in OpenCV implementierten Interpolationsverfahren, welche in Abschnitt 2.3 vorgestellt wurden, zunächst getestet (siehe Abschnitt 4.2). Es zeigt sich, dass die Lanczos4 sowie die bikubische Interpolation die besten Ergebnisse liefern, da die Erkennungsrate im Vergleich zu den anderen Interpolationen am höchsten ist. Nachdem die Bilder der QR-Codes skaliert sind können sie mit ZXing dekodiert werden. Das Resultat ist zum Einen der Inhalt des QR-Codes und zum Anderen seine Lage im Bild.

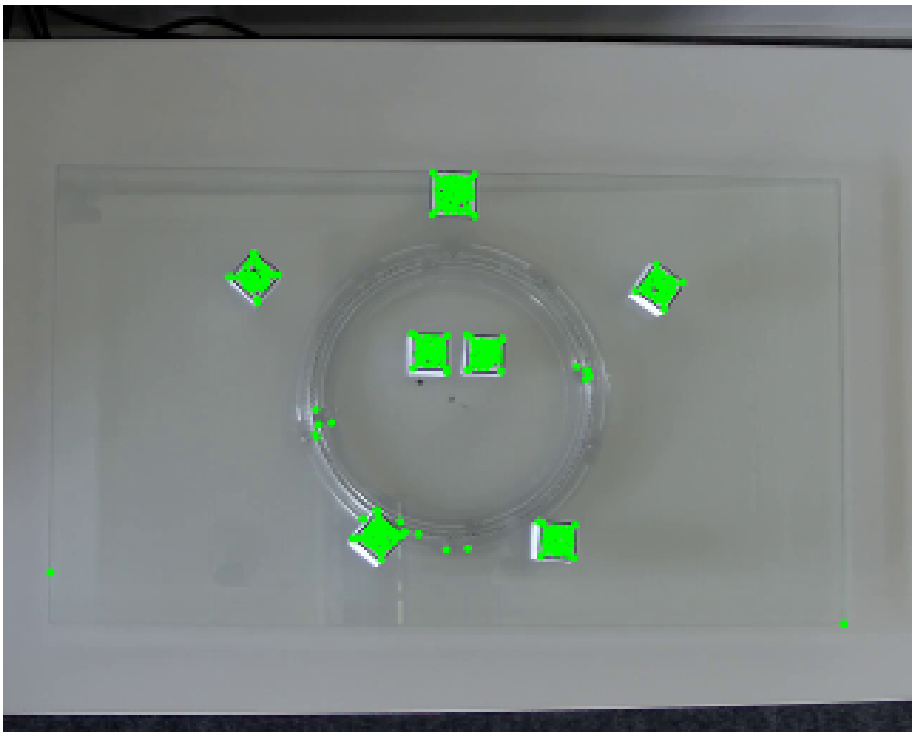


Abbildung 3.2.4: Visualisierung der aus dem Verfahren Good Features to Track zurückgelieferten GFTT-Keypoints.

In einem weiteren Schritt muss geprüft werden, ob für die einzelnen QR-Codes ein Mapping vorliegt. Das Mapping beinhaltet die Zuordnung von einem QR-Code zu einem Objekt in ViSAR. Zusätzlich ist ein Offset enthalten, so dass der QR-Code nicht in der Mitte des Objektes platziert werden muss. Dadurch ist es möglich ein QRObject mit allen relevanten Koordinaten und Informationen zu füllen und dies an den gegebenen ObjectManager, der die Objekte verwaltet, weiterzugeben.

Datenhaltung und Koordinatentransformation

Sobald die QRObjects an den Objectmanager, welcher sich im Paket object befindet, übergeben werden, ist es möglich, die Daten an ViSAR zu übertragen. Um die Kommunikation zwischen ViSAR.OBJ und ViSAR zu verringern, werden nicht immer alle Daten übertragen. Hierzu bietet der ObjectManager und das QRObject die Möglichkeit zu überprüfen, ob die Daten übertragen werden müssen oder nicht.

Der ObjectManager liegt in unterschiedlichen Ausführungen vor. Die relevante und im weiteren verwendete Ausführung ist der AdvancedObjectManger. Dieser bietet die Möglichkeit eine gewisse Historie zu speichern. Denn es kann zum Beispiel vorkommen, dass ein QR-Code in einem Bild erkannt wird, in dem darauf folgenden jedoch nicht. Falls sich dieses Verhalten öfters wiederholt, führt dies zu ei-

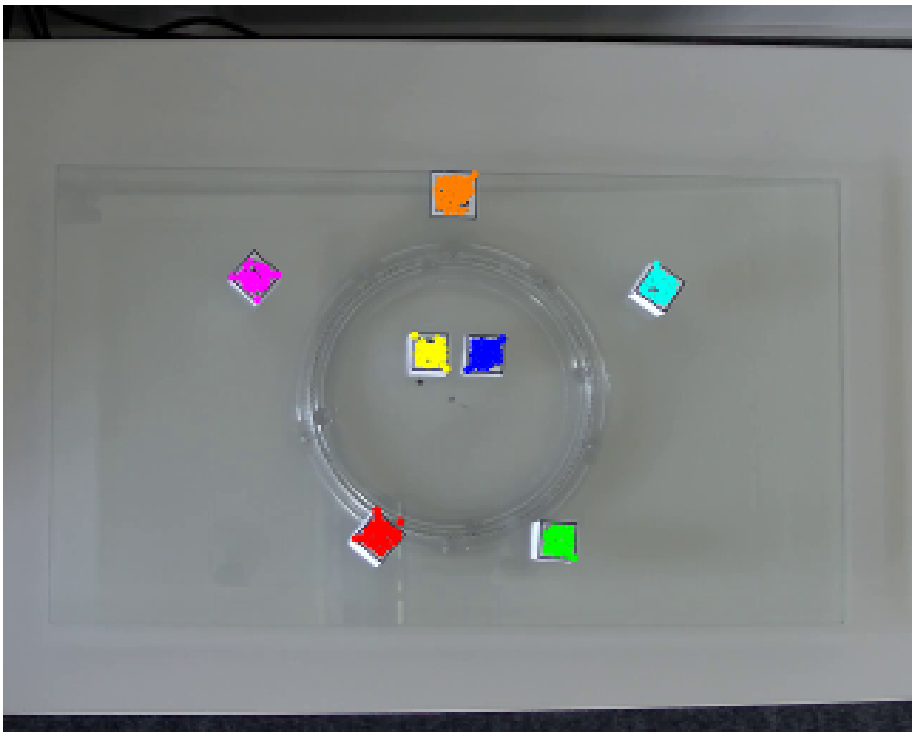


Abbildung 3.2.5: Visualisierung der Cluster die Anwendung des DBSCAN Verfahrens entstehen.

nem Flackern in ViSAR. Außerdem verursacht dieses Verhalten zusätzlichen Kommunikationsaufwand. Aus diesem Grund wurde eine Historie eingeführt, die dieses Verhalten kompensiert. Die Anzahl der Elemente in der Historie kann beliebig gewählt werden. Des Weiteren ist es möglich, die Anzahl der Abschnitte zu definieren, in denen der QR-Code auftauchen muss bevor er gelöscht wird.

Das QRObject merkt sich intern ob seine Position im Raum verändert wurde oder nicht. Des Weiteren berechnet das Objekt auch die Position, die es in ViSAR hat. Nachdem die Koordinate im Bild bekannt ist, wird die Koordinate relativ zum vorher festgelegten Ursprung berechnet. Um die Drehung der Glasplatte zu berücksichtigen, wird die Rotation dieser mit in die Berechnung einbezogen, um das Koordinatensystem von ViSAR nach zuführen. Dieser Sachverhalt ist in Abbildung 3.2.6 dargestellt. Die Koordinaten berechnen sich wie folgt:

- $Bildkoordinate = (X_B, Y_B)$
- $Ursprungskoordinate = (X_{M_R}, Y_{M_R})$
- $Drehung_{Glasplatte} = \alpha$

$$Relativkoordinate = Bildkoordinate - Ursprungskoordinate = \begin{bmatrix} X_B \\ Y_B \end{bmatrix} - \begin{bmatrix} X_{M_R} \\ Y_{M_R} \end{bmatrix}$$

Tabelle 3.1: Herausgeschnittene Teilbilder. Hier sind die einzelnen aus dem Bild extrahierten QR-Codes dargestellt.

$$Visarkoordinate = \begin{bmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{bmatrix} * Relativkoordinate = \begin{bmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{bmatrix} * \begin{bmatrix} X_B - X_{M_R} \\ Y_B - Y_{M_R} \end{bmatrix}$$

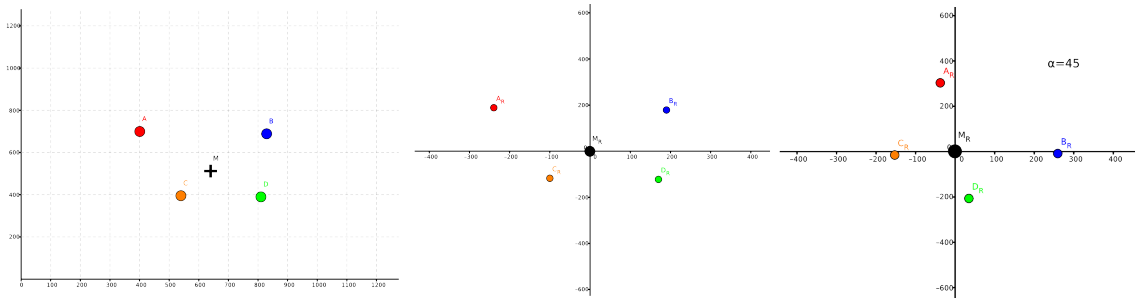


Abbildung 3.2.6: Überführung der Bildkoordinaten(links) in relative Koordinaten (mitte) und schlussendlich in das Koordinatensystem von ViSAR(rechts).

Übertragung an ViSAR

Zum Schluss müssen die berechneten Koordinaten an ViSAR übertragen werden. Hierfür wurde eine vom IOSB bereitgestellte Schnittstelle genutzt, die um die benötigen Komponenten im Paket *client* erweitert wurde.

3.3 ViSAR

ViSAR ("Visualisierung von geometrischen Radareffekten"), in Abbildung 3.3.1 zu sehen, ist ein Lernprogramm für die Weiterbildung vom Bildauswerter zum Radarbildauswerter und wurde am Fraunhofer IOSB entwickelt.

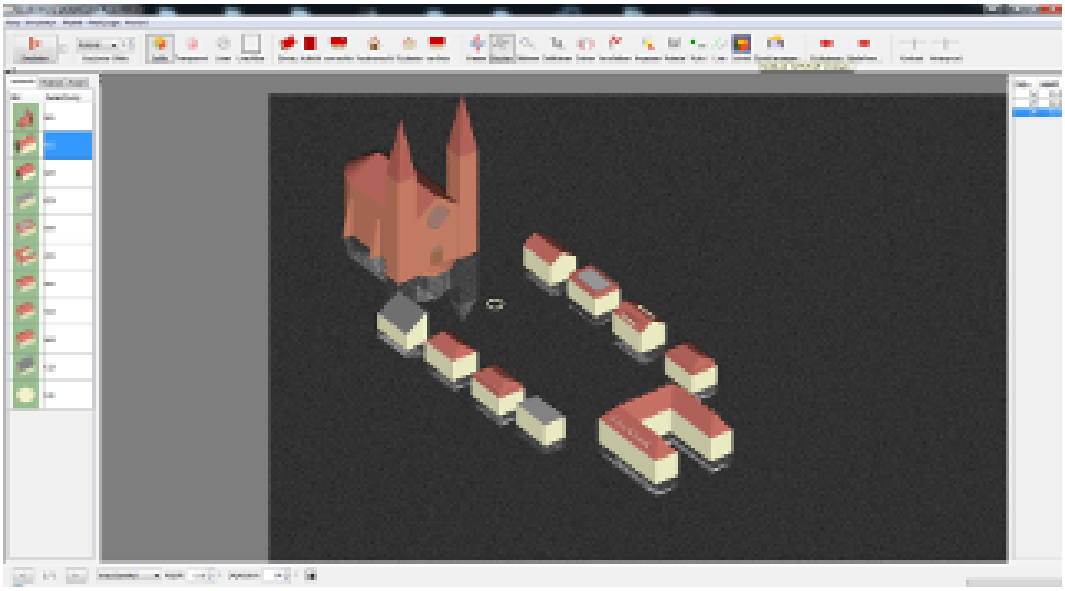


Abbildung 3.3.1: Benutzeroberfläche von ViSAR mit der Simulation einer Marktplatzszene

Aktuell werden im simulierten Radarbild ausschließlich diffuse und spiegelnde Einfachreflektionen, sogenannte Single Bounce angezeigt. Durch bestimmte Dach- und Gebäudeformen entstehen, je nach Aufnahmegeometrie und Materialeigenschaften der Gebäudeteile, typische Radarbildzeichen, die durch affine Projektionen und Überlagerungen geprägt sind. Somit ist es dem Schüler möglich, verschiedene Szenen zu laden, unterschiedliche Dach- und Gebäudeformen im Radarbild zu erkennen und den tatsächlichen Ort der Gebäude im Radarbild zu bestimmen [Pir12].

3.4 Diskussion

Im folgenden werden zwei Ansätze vorgestellt, welche zunächst als erfolgsversprechend eingestuft wurden, die jedoch nicht zielführend waren.

Vorverarbeitung durch morphologische Operationen

Der erste Schritt besteht darin, die QR-Codes durch morphologische Operationen zu extrahieren. Die Idee dabei ist, die Strukturen des QR-Codes so weit zu verschmelzen, dass weitgehend rechteckige Boxen entstehen. Dazu wird das Originalbild zunächst in ein Schwarzweißbild umgewandelt und im

Anschluss die Lücken im Bild durch Erosion geschlossen. Die Konturen der Elemente im Bild werden durch OpenCV bestimmt und nur diejenigen heraus gefiltert, die rechteckig sind. Diese werden in einem weiteren Schritt vergrößert und durch ZXing dekodiert. Das Verfahren funktioniert, solange der Untergrund (weißes Blatt) hinter dem QR-Codes weiß ist. Befindet sich der QR-Code aber auf einer Glasplatte können die Konturen oberhalb des Kugellagers nicht mehr vernünftig extrahiert werden.

Um dieses Problem zu umgehen, kann ein schwarzer Rahmen um jeden QR-Code gezogen werden, den das Programm dann sucht. Das Verfahren funktioniert, solange die QR-Codes keine Drehung in sich aufweisen. Bei bestimmten Drehungen kommt es vor, dass der Rahmen im Kamerabild nicht gut abgebildet wird. Das hatte zur Folge, dass abermals die Konturen nicht gut erkannt werden.

ASIFT: An Algorithm for Fully Affine Invariant Comparison

Die Scale Invariant Feature Transform (SIFT) wurde 1999 von David Lowe veröffentlicht [Low99]. Es nutzt lokale Bild Features die invariant zu Skalierung, Translation, Rotation, teilweise invariant für Helligkeitsunterschiede und affine oder 3D Projektionen sind.

Der Algorithmus "An Algorithm for Fully Affine Invariant Comparison" oder kurz ASIFT bezieht nicht nur Skalierung, Rotation und Translation ein, sondern zusätzlich die Winkel der Kameraachse [YM11].

Die Idee war ein QR-Code zu nutzen, der vom Aufbau sehr allgemein gehalten war. Das bedeutet, dass die Module aus schwarzen und weißen Kästchen relativ gleichmäßig über den Code verteilt sind. In einem weiteren Schritt wurden die Keypoints aus dem Bild der Kamera und aus dem zuvor beschriebenen QR-Code durch das ASIFT- und das SIFT-Verfahren ermittelt und verglichen. Die Vergleiche führten zu keinem brauchbaren Ergebnis, denn es wurde keine Übereinstimmung gefunden. Als Software wurde die verfügbare Demo von der Website ASIFT (http://www.ipol.im/pub/art/2011/my-asift/) verwendet.

Kapitel 4

Experimente

In diesem Kapitel werden die durchgeführten Tests beschrieben. Dabei wird auf die Bedingungen eingegangen unter denen das Assistenzsystem verwendet werden kann. Zusätzlich wird die Laufzeit analysiert und gezeigt welche Module verbessert werden können. Als Versuchsort stand das SimLab, welcher in Abbildung 4.0.1 zu sehen, zur Verfügung.

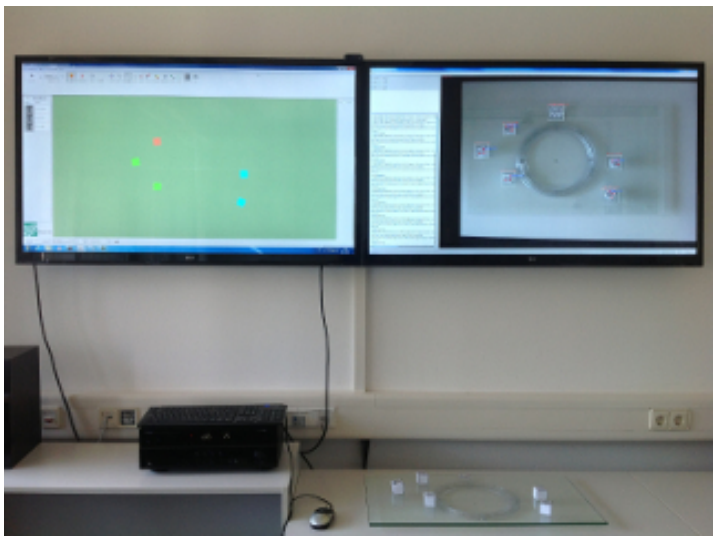


Abbildung 4.0.1: Der Versuchsort an dem die Experimente durchgeführt wurden.

4.1 Verwendete Hardware

Für alle Versuche wurde die Axis P1347 Network Camera eingesetzt. Sie überträgt die aufgenommenen Bilder im H.264 Format an das Programm ViSAR.OBJ. Damit die Kamera unter OpenCV genutzt werden kann, muss der Codec des Videostreams angegeben werden, da sonst die Bilder nicht verarbeitet werden können:

<IP>/mjpg/video.mjpg?<Parameter>*&.mjpg*

Durch die in Tabelle 4.1 aufgeführten Parameter kann das Verhalten der Kamera beeinflusst werden. Genauere Informationen kann dem Handbuch[1] der Kamera entnommen werden.

Parameter	Beschreibung	Beispiel
resolution	Gibt die Auflösung an.	http://192.168.0.1/mjpg/video.mjpg?resolution=640x480
rotation	Gibt an um wie viel Grad das aufgenommene Bild gedreht werden soll.	http://192.168.0.1/mjpg/video.mjpg?rotation=90
streamprofile	Falls ein Stream Profil hinterlegt ist kann diese gewählt werden.	http://192.168.0.1/mjpg/video.mjpg?streamprofile=ViSAR
videocodec	Gibt an welcher Videocodec von der Kamera verwendet werden soll.	http://192.168.0.1/mjpg/video.mjpg?videocodec=h264

Tabelle 4.1: Zusammenfassung einer Parameter die beim Zugriff auf die Kameras von Bedeutung sind.

Als Computer stand ein Desktop PC mit Windows 7 Professinal und Java 7 zur Verfügung. Die Hardware besteht aus einem Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz 3.40 GHz mit 8GB Arbeitsspeicher und einer NVIDIA GeForce 9300 GE. Es wurde OpenCV in Version 2.4.4 und JavaCV in Version 0.4 eingesetzt.

4.2 Bewertung der Interpolation zur Vergrößerung der QR-Codes

Wie zuvor in Abschnitt 3.2 beschrieben, müssen die extrahierten Bilder von ZXing aufgrund ihrer kleinen Strukturen skaliert werden. Um die beste Interpolation wählen zu können, werden die in Kapitel 2.3 beschriebenen Verfahren auf die extrahierten Teilbilder angewendet.

Zunächst wurde jede Interpolation auf ein Bild angewendet, welches eine Auflösung von 1280 x 1024 Pixeln hatte und in dem sechs QR-Codes angeordnet sind. Die so skalierten Teilbilder sind mit einer Vergrößerung der oberen rechten Ecke des QR-Codes in Tabelle 4.2 zu sehen. Bei diesen Tests stellte sich heraus, dass die Strukturen des QR-Codes bei dem Area und Nearest-Neighbor Verfahren besonders stark verschwimmen. Dabei verschmelzen benachbarte Pixel mit ähnlicher Farbe zu größeren Blöcken. Das führt dazu, dass die weißen Module des QR-Codes bei Drehungen von den benachbarten schwarzen Modulen überlagert werden. Dieses ist bei der bilinearen Interpolation nicht der Fall, jedoch verschwimmen die Module des QR-Codes auch. Bei den anderen Verfahren tritt dieses Verhalten nur zum Teil auf.

[1] http://www.axis.com/files/manuals/um_p1347_45969r2_en_1212.pdf

Area Interpolation	Nearest-Neighbor Interpolation
	
Bilineare Interpolation	**Bikubische Interpolation**
	
Lanczos4 Interpolation	
	

Tabelle 4.2: Vergleich der Interpolation, die in OpenCV implementieren sind.

Um die Erkennungsrate für die unterschiedlichen Verfahren zu ermitteln, wird als Quelle ein Referenzvideo verwendet, das eine Auflösung von 1280 x 1024 Pixel aufweist. In dem Videobild werden sechs QR-Codes mit einer Größe von 2,3 cm dargestellt. Nachdem die Teilbilder aus dem Video extrahiert sind, werden sie skaliert. Nutzt man das selbe Referenzvideo für jede Interpolation, so kann man diese vergleichen. Dazu werden die Anzahl der erkannten QR-Codes über die Bilder des Videos aufsummiert.

Abbildung 4.2.1 zeigt die Anzahl der erkannten QR-Codes für jedes Verfahren. Dabei zeigte sich, dass ZXing mit bikubisch und Lanczos4 interpolierten Bildern die besten Ergebnisse liefert. Danach folgte die lineare Interpolation, gefolgt von der Nearest-Neighbor und die Area Interpolation. Obwohl das Lanczos4 Verfahren etwas bessere Ergebnisse lieferte, wurde dennoch die bikubsiche Interpolation gewählt, da sie eine kürzere Laufzeit aufweist.

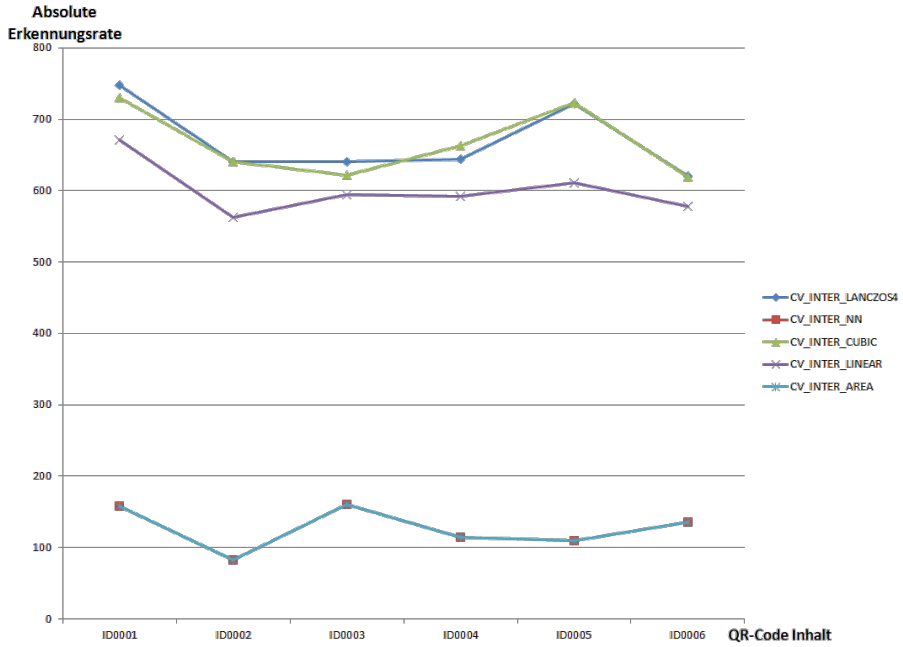


Abbildung 4.2.1: Vergleich der Verfahren zur Interpolation, bei einer Auflösung von 1280 x 1024 und einer QR-Code Größe von 2,3 cm in einem aufgenommen Referenzvideo. Aufgetragen sind die Inhalte sowie die Anzahl der erkannten QR-Codes für jede Interpolation.

4.3 Erkennungsraten bei Verwendung unterschiedlich dimensionierter QR-Codes und Kamera Auflösungen

Mit diesem Testaufbau soll festgestellt werden, welche QR-Code Größen mit welcher Kamera Auflösungen die besten Ergebnisse liefern. Hierzu wird dem Programm ViSAR.Obj jeweils ein Kamerabild, auf dem die QR-Codes enthalten sind, in unterschiedlichen Auflösungen übergeben.

Wie zuvor beschrieben, sollen die QR-Codes so klein wie möglich sein und dabei gleichzeitig noch gut erkannt werden. Als Variablen wird eine Bildauflösung zwischen 640 x 480 Pixel und 1920 x 1080 Pixel verwendet, wobei die Größe der QR-Code zwischen 2 cm und 3 cm liegen. Zu berücksichtigen ist, dass bei einer sehr hohen Auflösung auch die Laufzeit zunimmt und demnach im gleichen Zeitraum weniger Bilder verarbeitet werden können.

Erkennungsrate bei Einzelbild

Das Abbildung 4.3.1 zeigt die Testanordnung. Auf der Glasplatte befinden sich insgesamt acht QR-Codes. Dabei haben jeweils zwei QR-Codes die gleiche Größe. Um ein gutes Ergebnis zu erhalten, werden die einzelnen Bilder in den unterschiedlichen Auflösungen zur gleichen Zeit von der Kamera abgegriffen. Werden die Erkennungsraten der QR-Codes gleicher Größe addiert und mit der Gesamtzahl

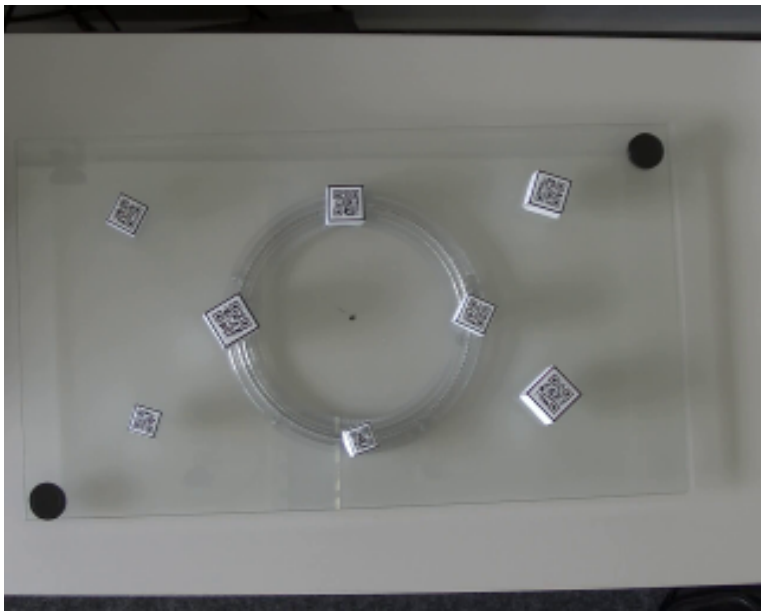


Abbildung 4.3.1: In der Szene sind acht QR-Codes in vier unterschiedlichen Größen zu sehen. Die Bilder wurden mit einer Auflösung von 1280 x 1024 Pixel aufgenommen.

der Bilder normiert, erhält man die Erkennungsrate in Prozent:

$$\text{Erkennungsrate}_{\text{Bild}} = \frac{\text{Anzahl erkannter QR-Codes gleicher Größe}}{\text{Anzahl der QR-Codes gleicher Größe}}$$

In der Abbildung 4.3.2 sind die Ergebnisse zusammengefasst. Auf der untereren linken Achse sind die unterschiedlichen Auflösungen in Pixel, auf der rechten unteren Achse die Größen des QR-Codes und auf der vertikalen Achse ist die Erkennungsraten aufgetragen. Die Werte können an dem Gitter, dass durch die Auflösung und die Größe der QR-Codes entstehen abgelesen werden. Dabei ist darauf zu achten, dass die Werte zwischen den Stützstellen interpoliert sind.

Es lässt sich ablesen, dass je nach Auflösung und Größe die QR-Codes unterschiedlich gut erkannt werden. Bei der Wahl der Parameter muss also darauf geachtet werden, dass nur Werte im oberen Bereich (hier grün eingezeichnet) gewählt werden, da hier die Erkennungsrate in dem Bild bei 100% liegt. In Abbildung 4.3.2 zeigt sich, dass bei einer QR-Code Größe von 2,3 cm und einer Auflösung von 1920 x 1080 Pixel die Erkennungsrate sinkt. Gleiches gilt auch für den 3 cm großen QR-Code. Die Ursache liegt entweder beim QR-Code Detektor (Abschnitt 2.4) oder an der Vorverarbeitung. Durch die Erhöhung der maximalen Anzahl der Keypoints, die von dem GFTT-Verfahren verwendet werden, konnten auch im 1920 x 1080 Pixel großen Bild alle QR-Codes erkannt werden. Somit ist es sinnvoll die maximale Anzahl der Keypoints mit der zuvor erkannten Anzahl der QR-Codes zu gewichten.

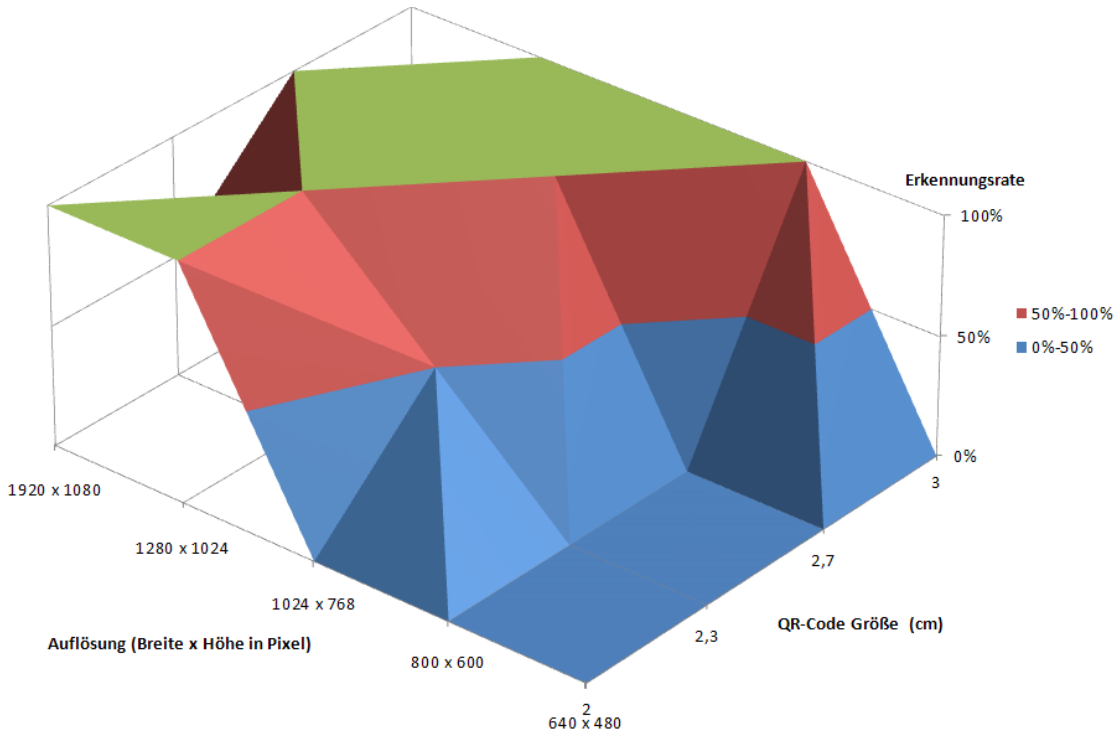


Abbildung 4.3.2: Erkennungsrate bei einem Aufgenommenen Bild bei jeweils unterschiedlicher Auflösung. Das Bild enthielt dabei acht QR-Codes, bei der jeweils zwei QR-Codes die gleiche Größe hatten.

Erkennungsraten bei Video

Der vorige Absatz beschreibt wie die Erkennungsraten an einem einzigen unbewegten Bild ermittelt wird. Bei der konkreten Anwendung werden dem Programm über die Videokamera laufend neue Bilder zu Verfügung gestellt. Daher ist es notwendig das Verhalten unter realistischen Bedingungen zu testen. Hierzu werden mehrere Videos aufgenommen. Die Videos sind jeweils 2 Minuten lang und wurden, mit unterschiedlicher Auflösungen, zur selben Zeit von derselben Kamera aufgenommen. Dabei sind die in Tabelle 4.3 aufgeführten möglichen Szenarien mit einbezogen.

Durchlaufen die aufgenommenen Videos das Programm, so entstehen bei der Auswertung der einzelnen Bilder die in Abbildung 4.3.3 gezeigte Grafik über die Anzahl der erkannten QR-Codes. Es fällt direkt auf, dass die meisten QR-Codes bei einer Auflösung von 800 x 600 Pixel und einer Größe von 3 cm erkannt werden. Wird jedoch die Gesamtzahl der analysierten Bilder berücksichtigt, zeigt sich aber ein anderes Ergebnis. Mit einer Auflösung von 800 x 600 Pixel werde von der Kamera 5,5 Bilder geliefert während bei einer Auflösung von 1920 x 1080 Pixel lediglich ein Bild geliefert wird.

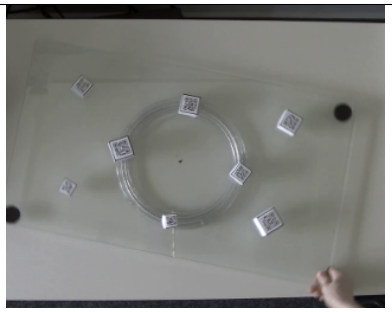
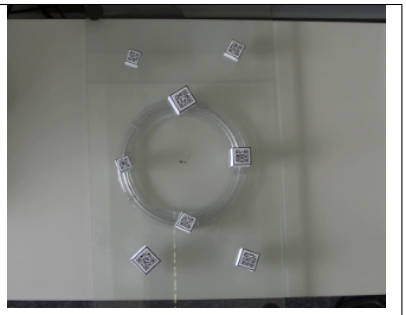
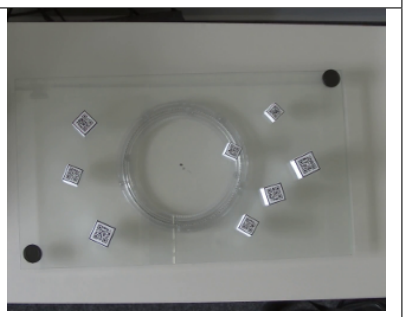
Bezeichnung	Grafik	Bezeichnung	Grafik
Störende Bewegungen z.B. durch Hände		Rotation des Drehtellers	
Person die Objekte bewegt		Veränderte Szene	

Tabelle 4.3: Mögliche Szenarien, welche bei der Verwendung von ViSAR.Obj auftreten können.

Die absoluten Erkennungsraten müssen mit der Gesamtzahl der Bilder normiert werden:

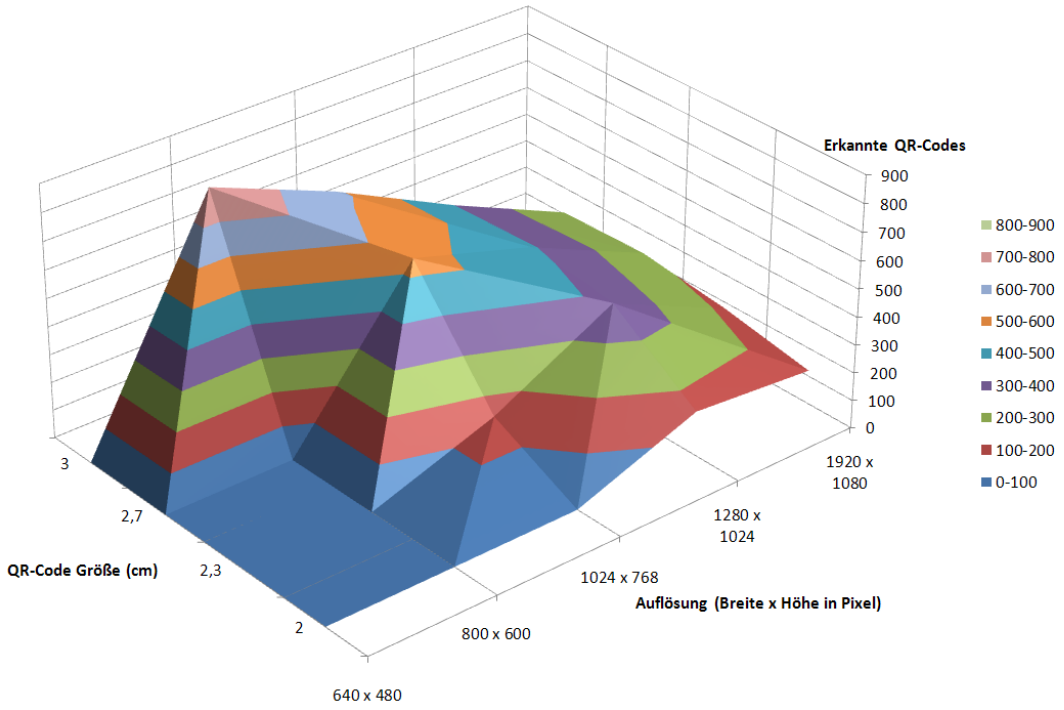
$$\text{Erkennungsrate}_{\text{Video}} = \frac{\sum_{\text{Bilder}} \text{Erkennungsrate}_{\text{Bild}}}{\text{Anzahl der analysierten Bilder}}$$

Das daraus entstehende Diagramm ist in Abbildung 4.3.4 zu sehen. Die maximale Erkennungsrate liegt nicht wie zuvor bei den Parametern 3 cm und 800 x 600 Pixel, sondern bei 1024 x 768 und einer QR-Code Größe von 3 cm. Werden die Werte des Diagramm betrachtet, so unterscheiden sich die Erkennungsraten von

- 1024 x 768 Pixel mit 3 cm,
- 1280 x 1024 Pixel mit 2,3 cm bis 3 cm und
- 1920 x 1080 Pixel mit 2,7 cm

um maximal 9%. Sie liegen jedoch alle zwischen 87 und 96 Prozent.

Der QR-Code soll so gut wie möglich erkannt und gleichzeitig sehr klein sein. Aus diesem Grund muss entweder der QR-Code mit 2,7 cm oder mit 2,3 cm und einer Auflösung von 1280 x 1024 Pixel



erden

Abbildung 4.3.3: Diese Grafik zeigt die absoluten Erkennungsraten. Jeweils in Abhängigkeit von der QR-Code Größe in cm und der Auflösung Pixeln

verwendet werden. Da im Video der 2,3 cm große QR-Code öfter von einem Körper verdeckt wurde als der 2,7 cm Große und die Größe eine entscheidende Rolle spielt wurde der 2,3 cm große Code mit einer Auflösung von 1280 x 1024 gewählt.

4.4 Einfluss der Lichtverhältnisse auf die Erkennungsrate

Zusätzlich zur Auflösung wird in diesem Abschnitt bewertet, wie resistent das Programm gegenüber der Veränderung der Lichtverhältnisse ist. Hierzu wurden drei Videos mit einer Auflösung von 1280 x 1024 Pixel aufgezeichnet. Diese drei Videos wurden unter folgenden Bedienungen erstellt:

1. Raum Tageslicht
2. Abgedunkelter Raum
3. Abgedunkelter Raum mit eingeschalteter Raumbeleuchtung

Die sechs 2,3 cm großen QR-Codes waren bei jedem Video an der selben Position auf der drehbaren Platte. Des Weiteren wurden bei allen Videos die gleichen Bewegungssequenzen durchgeführt, um vergleichbare Resultate zu erzielen.

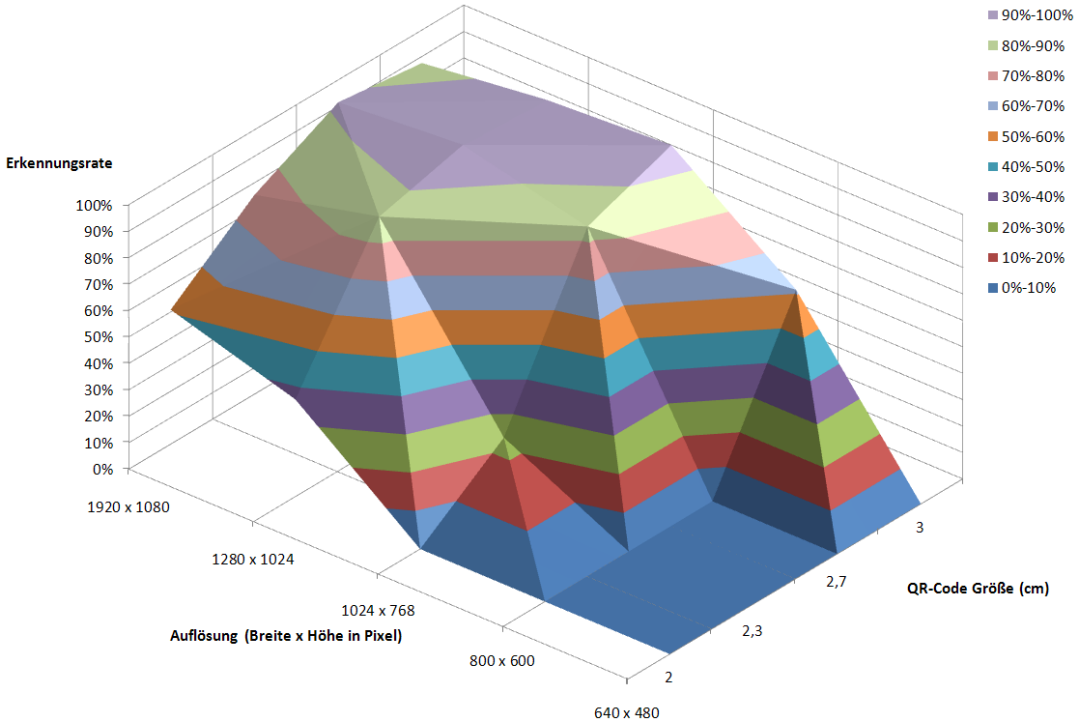


Abbildung 4.3.4: Die Grafik enthält die ermittelten Erkennungsraten, welche anhand eines Videos gemessen wurden. In den Video waren unterschiedlich große QR-Codes zu sehen, welche jeweils mit einer Kamera in unterschiedlichen Auflösungen aufgenommen wurden.

Um die drei Videos zu vergleichen, wurden die absoluten Erkennungsraten ermittelt (siehe Abbildung 4.4.1). Wie erwartet, werden die QR-Codes am besten unter Tageslicht erkannt. Unter diesen Bedingungen fällt genug Licht auf die QR-Codes, so dass die Module des QR-Codes nicht verschwimmen. Im abgedunkelten Raum mit eingeschalteter Raumbeleuchtung wurden, im Vergleich mit den Werten bei Tageslicht, im Durchschnitt 150 QR-Codes weniger erkannt. Weniger Licht bedeutet auch weniger Kontrast bei dem Aufnahmesystem. Die Module sind nicht mehr gut unterscheidbar. Bei einem abgedunkelten Raum geht die Erkennungsrate gegen Null. Unter diesen Bedienungen werden die Module sehr stark verschwommen dargestellt. Die Übergänge zwischen den weißen und schwarzen Bereichen weisen keinen hohen Kontrast auf, sondern gehen in Graustufen ineinander über.

Damit auch unter schlechten Lichtverhältnissen die QR-Codes erkannt werden, sollte eine zusätzliche Lichtquelle installiert werden.

4.5 Evaluation des Laufzeitverhaltens

Da das entwickelte Programm annähernd echtzeitfähig sein soll, wird im Folgenden die Laufzeit der relevanten Funktionen analysiert. Grundsätzlich lässt sich das System in drei Bereiche unterteilen.

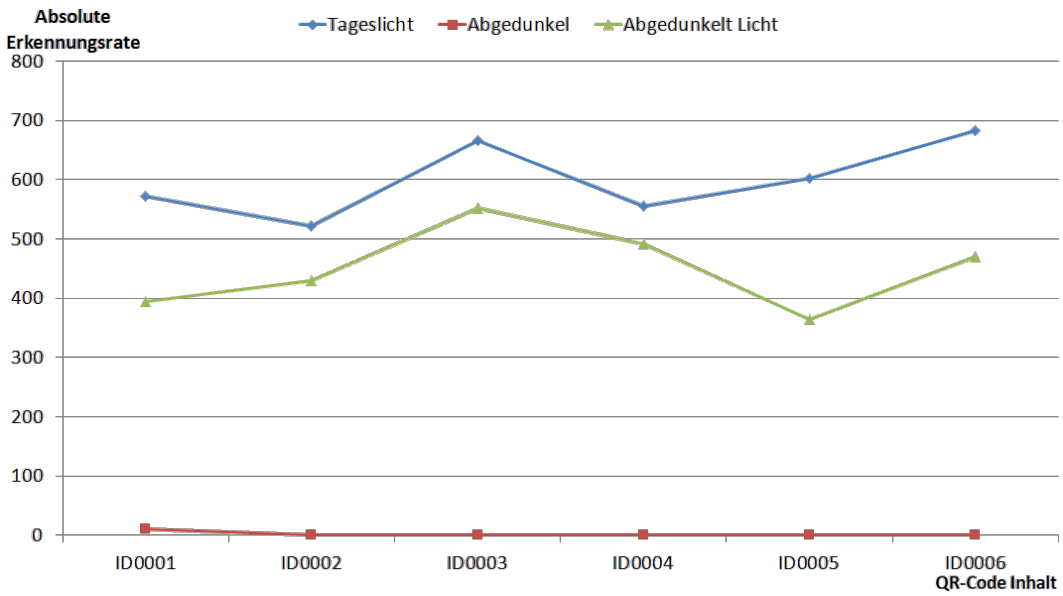


Abbildung 4.4.1: Ermittelte Erkennungsraten bei unterschiedlichen Lichtverhältnissen.

Zum einen die Vorverarbeitung, die aus der Berechnung der GFTT-Keypoints und dem Clustering dieser besteht, sowie der Extraktion der Bildbereiche. Als Zweites die QR-Code Analyse, die auf den extrahierten Bildbereichen stattfindet. Der dritte Bereich ist die Übertragung der Koordinaten an ViSAR.

Im Folgenden wird angenommen, dass nur ViSAR.OBJ und ViSAR auf dem Computer gestartet sind. Beim Ablauf des Videos wird im Programmcode eine Klasse mit den jeweiligen Zeiteinheiten gespeist. Die so ermittelten Laufzeiten sind in Abbildung 4.5.1 als Diagramm dargestellt. Das Diagramm gibt die Dauer der Operation in ms über die Bilder des Videos wieder. Dabei zeigte sich, dass die Dauer der Funktionen zur Erzeugung der GFTT-Keypoints, des Clustering und der QR-Code Analyse einigermaßen konstant verlaufen. Im Gegensatz dazu dauert die Übertragung der Daten an ViSAR zwischenzeitlich sehr lange.

Werden die in Abbildung 4.5.1 gezeigten Werte für jedes Bild aufsummiert, so erhält man die Gesamtdauer der Operationen pro Bild. Der errechnete Mittelwert liegt bei 87,25 ms. Das bedeutet, dass im Mittel ein Bild in 87,25 ms abgearbeitet werden kann. Somit können theoretisch jede Sekunde 11,46 Bilder bearbeitet werden. Durch zusätzlichen Aufwand der durch das Programm entsteht werden im Schnitt zwischen 6 bis 8 Bilder pro Sekunde verarbeitet. Der ermittelte Wert kann je nach Szene stark schwanken. Wobei ein Großteil dieser Schwankungen auf die Kommunikation zurückzuführen ist.

Um dies zu verdeutlichen wurden die in Abbildung 4.5.1 dargestellt Laufzeiten in ein Boxplot Diagramm überführt (siehe Abbildung 4.5.2). Im Boxplot ist zu sehen, dass die Laufzeit der Kommunikation stark vom Median abweichen kann.

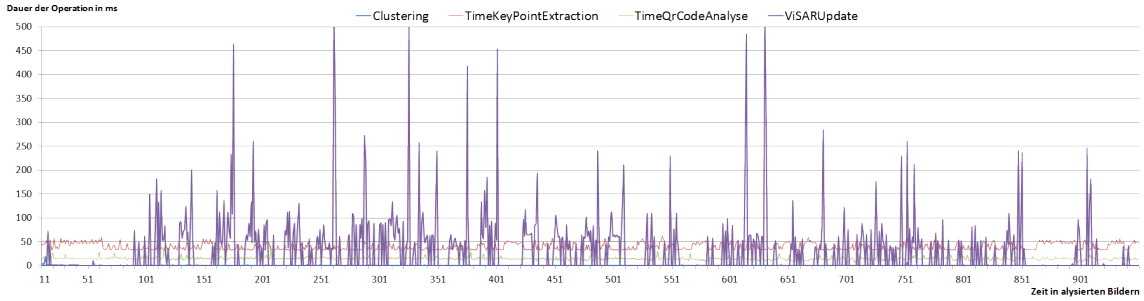


Abbildung 4.5.1: Dargestellt ist die Laufzeit der einzelnen Aufgaben die für die Analyse eines Bildes nötig sind.

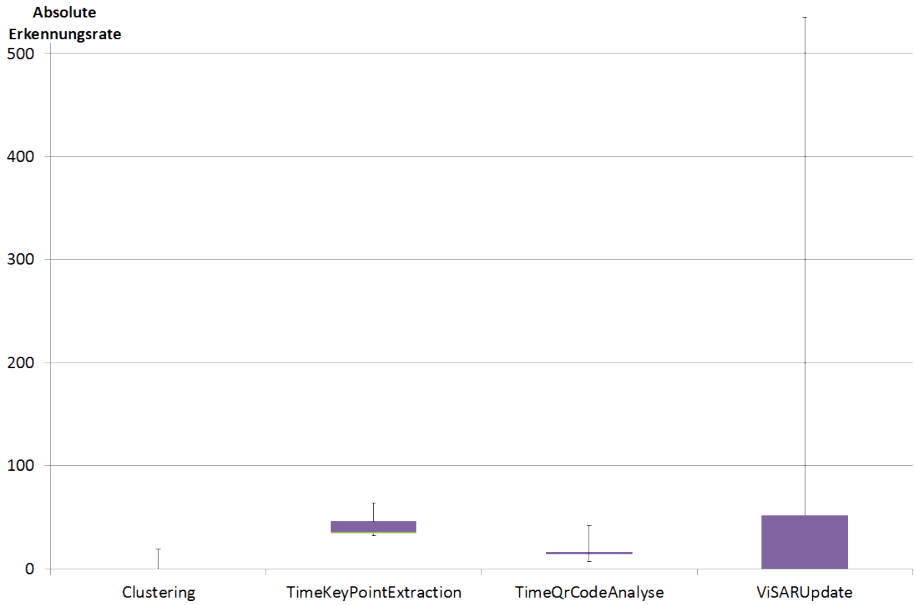


Abbildung 4.5.2: Veranschaulichung der Laufzeit der relevanten Operationen anhand eines Boxplot-Diagramms.

Die Ursache, für die starke Schwankung der Laufzeit von ViSARUpdate, ist zum einen auf die Kommunikationszeiten zwischen den zwei Programmen ViSAR und ViSAR.Obj, sowie auf interne Optimierungen zurückzuführen. Wie schon in Kapitel 3.2 beschrieben, werden die Objekte nur übertragen, wenn sich ihre Lage im Raum ändert. Falls nur ein kleiner Teil seine Lage ändert, ist die Laufzeit geringer. Ist es notwendig in einem Schritt alle Objektkoordinaten an ViSAR zu übertragen, nimmt dieser Prozess mehr Zeit in Anspruch.

Es gibt mehrere Möglichkeiten den Gesamtprozess zu beschleunigen. Zum einen besteht die Möglichkeit ViSAR in Intervallen zu aktualisieren. Zum anderen könnte ein größerer Toleranzbereich, indem sich die Objekte bewegen können ohne an an ViSAR übertragen zu werden, eingestellt werden. Die

dritte Möglichkeit wäre, die Schnittstelle welche die Kommunikation der beiden Programme übernimmt zu optimieren.

4.6 Diskussion

In dem Assistenzsystem besteht die Möglichkeit die Kamera zu kalibrieren. Dies wurde während der Bachelorthesis getestet, jedoch führte es nur zu einer Verbesserung von ca. 1%. Der Grund hierfür liegt vermutlich darin, dass sich die Kalibrierung nur bei Kameras mit schlechten Eigenschaften bemerkbar macht. Die im SimLab eingesetzte Kamera ist qualitativ hochwertig.

Kapitel 5

Szenario

In diesem Kapitel wird das entwickelte Assistenzsystem anhand eines Szenarios präsentiert. Das Ziel ist eine Abwandlung der Marktplatzszene, die in Abschnitt 3.3 zu sehen ist. Die Szene wird Schritt für Schritt aufgebaut. Zunächst wird eine Kirche und ein Haus in der Szene positioniert. Danach wird die Szene erweitert bis in ihr der Marktplatz zu sehen ist.

Die Abbildung 5.0.1 zeigt oben links das Kamerabild, darauf wird das GFTT-Verfahren angewendet. Die GFTT-Keypoints sind oben rechts in grün eingezeichnet. Danach werden diese gruppiert (unten links zu sehen) und die QR-Codes dekodiert sowie ihre Lage auf der Glasplatte bestimmt. Im Anschluss werden die Koordinaten an ViSAR übertragen und dort die Objekte, die durch die QR-Codes vertreten werden, dargestellt.

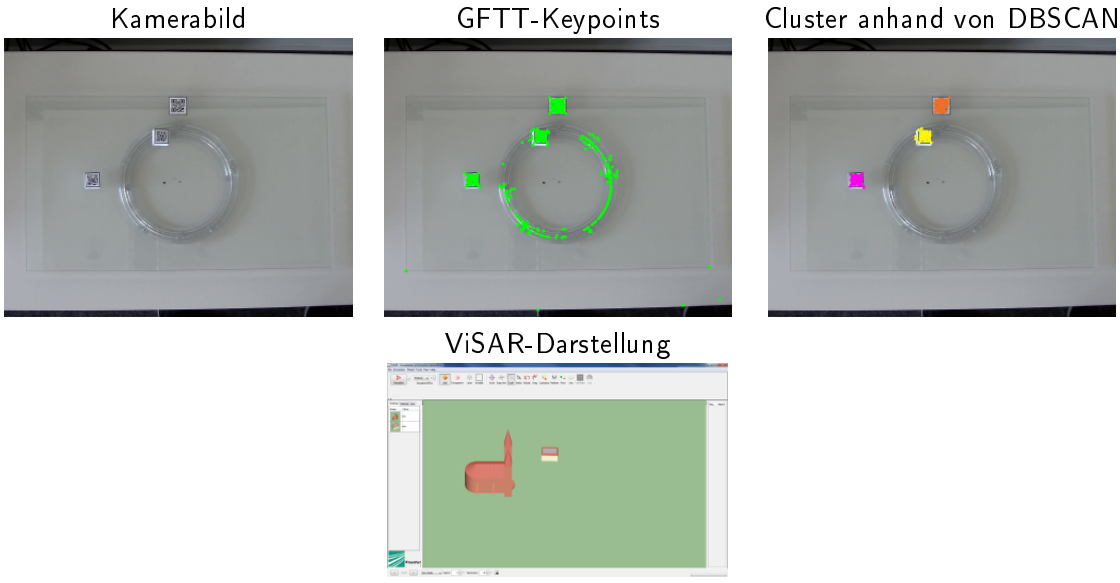


Abbildung 5.0.1: Veranschaulichung des Szeneaufbaus und der angewandten Verfahren.

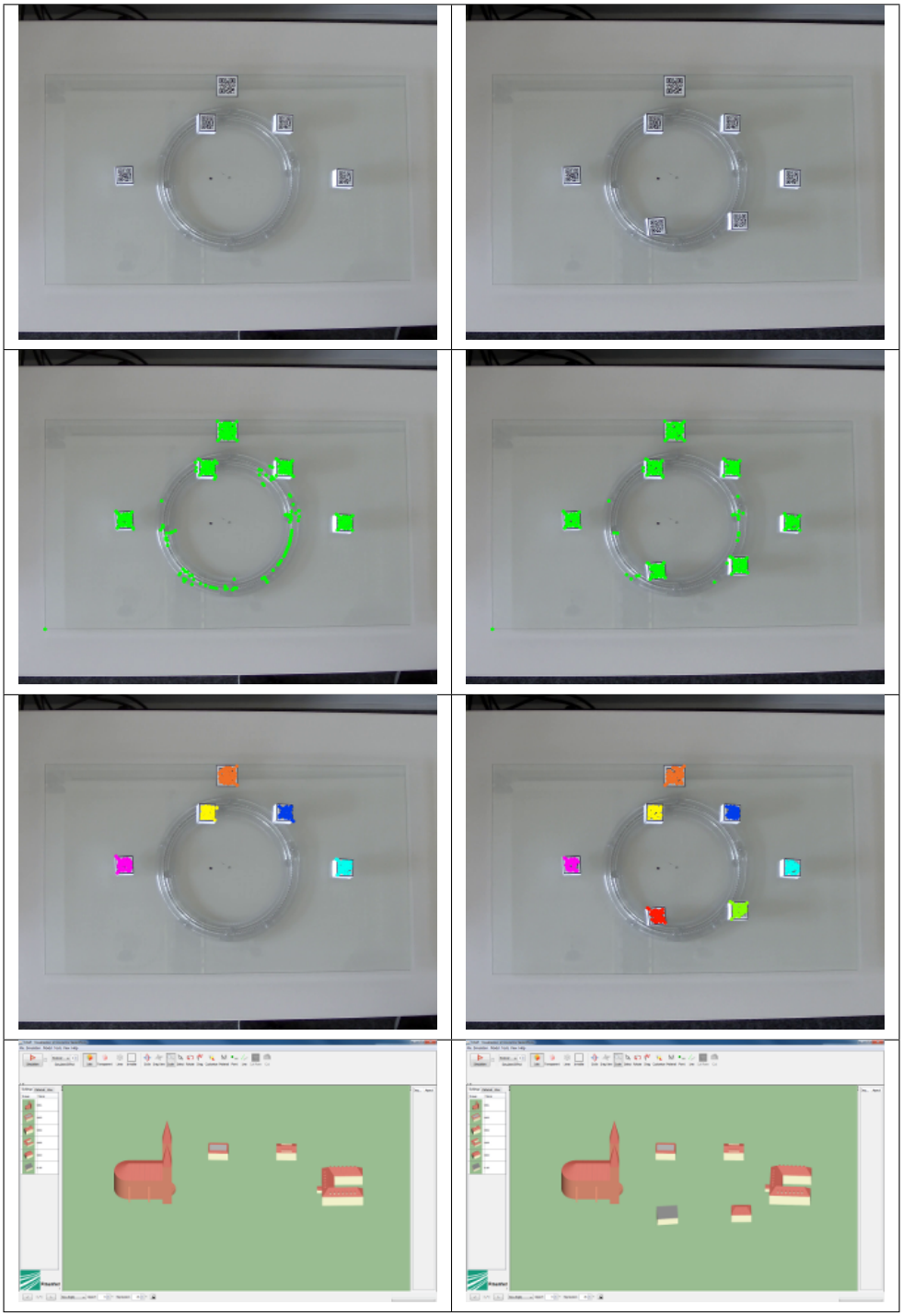


Abbildung 5.0.2: Vervollständigung der Markplatzszene.

Kapitel 6

Zusammenfassung und Ausblick

Mit dieser Bachelorarbeit wird ein Assistenzsystem für ViSAR entwickelt. Dabei wird für den Aufbau der Szene sowie deren Veränderung ein Tangible User Interface eingesetzt. Das entwickelte Interface versetzt den Anwender in die Lage das Programm ViSAR intuitiver zu bedienen.

Das Assistenzsystem ist in Java implementiert und nutzt die drei Bibliotheken OpenCV, java-statistical-analysis-tool und ZXing. Beim Design, sowie bei der Implementierung stand die Erweiterbarkeit im Vordergrund, so dass durch den modularen Aufbau Änderungen sehr einfach und schnell durchgeführt werden können.

Die Verarbeitung der Bilder erfolgt in fünf Schritten. Zunächst werden durch das Good Features to Track (GFTT) Verfahren Keypoints berechnet. Diese Keypoints sind hauptsächlich auf den QR-Codes zu finden. Im zweiten Schritt werden die GFTT-Keypoints durch den DBSCAN-Algorithmus gruppiert. Von diesen Clustern wird dann der Median der X und Y Koordinate bestimmt. Dieser wiederum wird als Mittelpunkt einer Box mit einer Größe von 70 Pixeln gewählt. Der somit komplett umschlossene QR-Code wird im dritten Schritt skaliert, damit der QR-Code durch ZXing dekodiert werden kann. Im Anschluss daran wird die Lage im Koordinatensystem von ViSAR bestimmt und an ViSAR übertragen.

In Experimenten wurde das entwickelte System anhand von Referenzvideos getestet. Die Experimente zeigen, dass die angestrebte Größe der QR-Codes erreicht wird. Die Erkennungsrate beträgt für einen 2,3 cm großer QR-Code auch unter Einbeziehung von Störungen mindestens 87%. Wird eine geringere Fehlertoleranz benötigt, muss der QR-Code zwangsläufig größer werden. Bei einer Größe von 2,7 cm ist eine Erkennungsrate von 95% nachgewiesen. Zudem kann die Rate durch eine bessere Beleuchtung weiter erhöht werden. Wird die Beleuchtung geringer, verschwimmen die Module des QR-Codes und es ist nicht mehr möglich ihn zu dekodieren. Des Weiteren wurde gezeigt, dass für die Verarbeitung des Bildes eine theoretische Geschwindigkeit von 11,46 Frames pro Sekunde erreicht wird. In der Praxis hängt diese Geschwindigkeit stark von dem Aufbau der Szene ab und wie schnell sie geändert wird. Dabei Schwanken die Werte zwischen 6 und 8 Frames pro Sekunde.

Das Assistenzsystem erfüllt die Anforderung die Szene in ViSAR aufzubauen und zu manipulieren. Beim Einsatz von 2,3 cm bis 2,7 großen QR-Codes liegt die Erkennungsrate zwischen 87% und 95%, wobei die Laufzeit zwischen 6 und 8 Frames pro Sekunde beträgt. Die Tests haben gezeigt, dass das Assistenzsystem für die Präsentation im Simulationslabor eingesetzt werden.

Ausblick

Durch die Anordnung der Kamera können Verzerrungen in den Bildern auftreten die einen negativen Einfluss haben. Es ist möglich die Anwendung dahingehen zu erweitern, dass die erzeugten Teilbilder entzerrt werden und somit die Erkennungsrate verbessert wird. Wie in Abschnitt 4.4 beschrieben, hat die Szenenbeleuchtung einen erheblichen Einfluss auf die Erkennungsrate. Deswegen sollte eine zusätzliche Lichtquelle montiert werden.

Es wurde festgestellt, dass die Kommunikation zwischen ViSAR.OBJ und ViSAR sehr langsam ist, obwohl nur die Veränderungen übertragen werden. Daher ist zu empfehlen, dieses Modul zu optimieren, damit die Kommunikation auch annähernd echtzeitfähig ist.

Um die Robustheit für den Einsatz bei der Ausbildung zu verbessern und unnötigen Rechenaufwand zu vermeiden, könnte zum Beispiel eine Kinect in das System integriert werden. Dadurch wäre es möglich zu prüfen ob sich die Szene überhaupt ändern kann. Dies ist der Fall wenn eine Person vor der aufgebauten Szene steht. In diesem Fall muss die Analyse der Bilder aktiviert werden. Sobald sich die Person, aus einem vorher definierten Bereich, entfernt wird sie deaktiviert. Somit wird die aktuelle Szene eingefroren, da keine Änderungen mehr auftreten können.

Von Vorteil wäre es das System dahingehend zu erweitern, dass keine Marker benötigt werden, sondern die Stellvertreterobjekte durch Modellhäuser ersetzt werden. Die Modellhäuser müssen dann automatisch von dem System erkannt werden. Durch den modularen Aufbau des Assistenzsystems kann die Software in einem weiteren Schritt durch dieses Modul ergänzt und optimiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AGD07] Azad, Pedram ; Gockel, Tilo ; Dillmann, Rüdiger: *Computer Vision - das Praxisbuch*. Aachen : Elektor-Verl., 2007 http://www.amazon.de/Computer-Vision-Praxisbuch-Pedram-Azad/dp/3895761656/ref=pd_sim_b_1. – ISBN 9783895761652 3895761656

[ArU13] ArUco: *ArUco*. http://www.uco.es/investiga/grupos/ava/node/26. Version: August 2013

[AT07] Acharya, Tinku ; Tsai, Ping-Sing: Computational foundations of image interpolation algorithms. In: *Ubiquity* 2007 (2007), Oktober, Nr. October, 4:1–4:17. http://dx.doi.org/10.1145/1317487.1317488. – DOI 10.1145/1317487.1317488. – ISSN 1530–2180

[Bar03] Barcelona, Music Technology Group Universitat Pompeu F.: *Reactable*. http://mtg.upf.edu/project/reactable. Version: 2003

[BAT11] Bergamasco, F. ; Albarelli, A. ; Torsello, A.: Image-Space Marker Detection and Recognition Using Projective Invariants. In: *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, 2011, 381-388

[Bek12] Bekri, Nadia E.: *Konzept zur Adaption eines Assistenzsystems zur Erkennung von unbekannten Objekten*, Hochschule Karlsruhe - Technik und Wirtschaft, Diplomarbeit, 2012. http://fhgonline.fhg.de/bibliotheken/iosb/12_El_Bekri_hld.pdf

[Bra00] Bradski, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[Bre09] Brell, Melina: *Eine vibrotaktile Mensch-Maschine-Schnittstelle für chirurgische Applikationen*, Universität Oldenburg, Diss., Juli 2009. http://www.informatik.uni-oldenburg.de/download/Promotionen/Dissertation_MBrell_lq.pdf

[Bür11] Bürger, Stefan: *Gestensteuerung mit Microsoft Kinect*. http://fhgonline.fhg.de/bibliotheken/iosb/11_Buerger_stc.pdf. Version: Juli 2011

[DGG+08] DJORDJEVIC, Valie (Hrsg.) ; GEHRING, Robert A. (Hrsg.) ; GRASSMUCK, Volker (Hrsg.) ; KREUTZER, Till (Hrsg.) ; SPIELKAMP, Matthias (Hrsg.): *Schriftenreihe der Bundeszentrale für politische Bildung*. Bd. 655: *Urheberrecht im Alltag. Kopieren, bearbeiten, selber machen*. 2. Aufl. Bonn : Bundeszentrale für Politische Bildung, 2008 http://www.bpb.de/publikationen/OJVZDZ,0,0,Urheberrecht_im_Alltag.html. – ISBN 3–89331–812–7

[Ech09] ECHTLER, Florian: *Tangible Information Displays*. München, Technische Universität München, Dissertation, 2009

[EKSX96] ESTER, Martin ; KRIEGEL, Hans peter ; S, Jörg ; XU, Xiaowei: A density-based algorithm for discovering clusters in large spatial databases with noise, AAAI Press, 1996, S. 226–231

[Get11] GETREUER, Pascal: Linear Methods for Image Interpolation. In: *Image Processing On Line* 2011 (2011). http://dx.doi.org/10.5201/ipol.2011.g_lmii. – DOI 10.5201/ipol.2011.g_lmii

[IBM13] IBM: *IBM i Labeling Software - Barcode Basics - Symbologies*. http://www.tlashford.com/TLA/pages/Basic_sym/Symbol_overview.htm. Version: August 2013

[Koe11] KOELLE, Marion: Tangible User Interfaces - Ein kurzer Überblick über Forschungsfeld und Literatur. (2011), 2011. http://www.medien.ifi.lmu.de/lehre/ws1011/mmi2/mmi2_uebungsblatt1_loesung_koelle.pdf

[KPT00] K.-P. TIMPE, H. K. T. Jürgensohn J. T. Jürgensohn: *Mensch-Maschine-Systemtechnik Konzept, Modellierung, Gestaltung, Evaluation*. 1. K.-P. Timpe, T. Jürgensohn, H. Kolrep, 2000. – 364 S.

[Len12] LENK, Bernhard: *QR Code : [mit 26 Tabellen]*. 1. Aufl., Stand 1. Okt. 2012. Kirchheim unter Teck : Lenk, 2012. – ISBN 978–3–935551–10–6

[LK81] LUCAS, Bruce D. ; KANADE, Takeo: An Iterative Image Registration Technique with an Application to Stereo Vision, 1981, 674–679

[Low99] LOWE, D.G.: Object recognition from local scale-invariant features. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* Bd. 2, 1999, 1150-1157 vol.2

[MFZ07] M. F. ZÄH, F. Engstler F. Friesdorf A. Schubö S. Stork A. Bannat F. W. M. Wiesbeck W. M. Wiesbeck: Kognitive Assistenzsysteme in der manuellen Montage - Adaptive Montageführung mittels zustandsbasierter, umgebungsabhängiger Anweisungsgenerierung. In: *wt Werkstattstechnik online* (2007). http://www.mmk.ei.tum.de/publ/pdf/07/07zae1.pdf

[PB07] Peinsipp-Byma, Elisabeth: *Leistungserhöhung durch Assistenz in interaktiven Systemen zur Szenenanalyse*. Universitätsverlag Karlsruhe, Karlsruhe, 2007. http://dx.doi.org/http://dx.doi.org/10.5445/KSP/1000007038. http://dx.doi.org/http://dx.doi.org/10.5445/KSP/1000007038

[Pir12] Pirk, Bastian: *Entwicklung eines Prototyps zur Analyse von Mehrfachreflektionen in Radarbildern zum Lernprogramm ViSAR*. http://fhgonline.fhg.de/bibliotheken/iosb/12_Pirk_ber.pdf. Version: 2012

[PJ93] Pötter, Mathias ; Jesse, Ralf: *Barcode : Einführung und Anwendungen*. 2. Hannover : Heise, 1993 http://www.bsz-bw.de/cgi-bin/ekz.cgi?SWB03314472. – ISBN 3–88229–010–2

[Raf13] Raff, Edward: *java-statistical-analysis-tool*. http://code.google.com/p/java-statistical-analysis-tool/. Version: Juli 2013

[RWA+07] Reifinger, Stefan ; Wallhoff, Frank ; Ablassmeier, Markus ; Poitschke, Tony ; Rigoll, Gerhard: Static and dynamic hand-gesture recognition for augmented reality applications. In: *Proceedings of the 12th international conference on Human-computer interaction: intelligent multimodal interaction environments*. Berlin, Heidelberg : Springer-Verlag, 2007 (HCI'07). – ISBN 978–3–540–73108–5, 728–737

[ST94] Shi, J. ; Tomasi, C.: Good features to track. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, 1994. – ISSN 1063–6919, S. 593–600

[Swi11] Swiontek, Cynthia: *Optische Positionsregelung eines Miniaturfluggeräts mit aktiven Landmarken*. http://fhgonline.fhg.de/bibliotheken/iosb/11_Swiontek_brk.pdf. Version: 2011

[TI13] Tec-IT: *Online Barcode Generator*. http://barcode.tec-it.com/barcode-generator.aspx. Version: August 2013

[TK91] Tomasi, Carlo ; Kanade, Takeo: Detection and Tracking of Point Features / International Journal of Computer Vision. Version: 1991. http://www.lira.dist.unige.it/teaching/SINA/slides-current/tomasi-kanade-techreport-1991.pdf. 1991. – Forschungsbericht

[WAV13] WAVE, DENSO: *QR-Code.com - DENSO WAVE, the Inventor of QR Code*. http://www.qrcode.com/en. Version: August 2013

[Wes06] Westendorff, Carsten: *Experimentelle Untersuchungen zur computergestützten Planung und bilddatengestützten Navigation bei enossalen Implantatlagerpräparationen*. Wilhelmstr. 32, 72074 Tübingen, Medizinischen Fakultät der Eberhard Karls Universität zu Tübingen, Diss., 2006. http://tobias-lib.uni-tuebingen.de/volltexte/2006/2340/pdf/Diss_Westendorff.pdf

[YM11] Yu, Guoshen ; Morel, Jean-Michel: ASIFT: An Algorithm for Fully Affine Invariant Comparison. In: *Image Processing On Line* 2011 (2011). http://dx.doi.org/10.5201/ipol.2011.my-asift. – DOI 10.5201/ipol.2011.my–asift

[YW03] Yao Wang, Ben Piper Carlo Ratti Professor Hiroshi I. Assaf Biderman B. Assaf Biderman: *SandScape*. http://tangible.media.mit.edu/project/sandscape/. Version: 2003

Anhang

Handbuch des markerbasierten Assistenzsystem für 3D-Radarbildsimulation

Marcel Finger

26. August 2013

Inhaltsverzeichnis

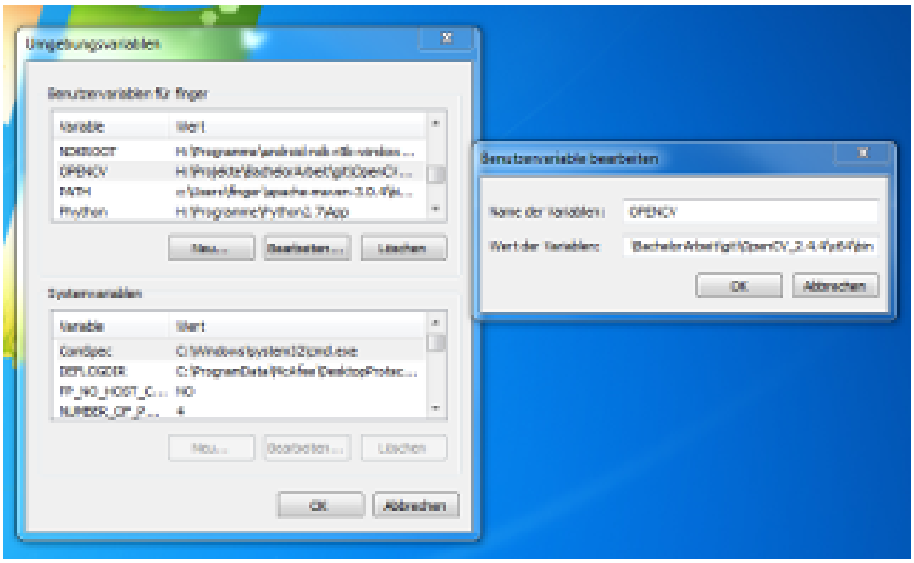


Abbildung 1.1: Eingestellte Umgebungsvariablen

1 Setup

Um das Programm nutzen zu können, müssen Sie zunächst die Umgebungsvariable PATH anpassen oder die OpenCV Libs auf eine andere weise für das Programm zugänglich macgen. Falls die Umgebungsvariablen geändert werden sollen, gehen Sie bitte wie folgt vor:

1. Start→Suche →"Umgebungsvariablen"
2. Dort eine neue Variable mit dem Namen OpenCV erstellen und den Pfad zu den Libs eintragen
3. Im Anschluss daran den PATH um "%OpenCV% erweitern

Nachdem Sie diese Schritte befolgt haben, sollte die OpenCV Variable ähnlich wie in Abbildung 1.1 aussehen.

2 Konfiguration

Um das Programm zu konfigurieren kann beim Aufruf des Programms mit dem Argument –config die Konfiguration angegebenen werden. Zum Beispiel:

java -jar ViSAR.OBJ.jar –config=../config.xml

Falls dieses Argument nicht vorhanden ist, wird davon ausgegangen, dass die Konfiguration im Ordner von ViSAR.OBJ.jar zu finden ist. Diese Standartkonfiguration, welche im folgenden angegeben ist, kann beliebig angepasst werden:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- config.xml -->
<configuration>
        <showui>true</showui>
        <visar>
                <update>false</update>
                <address>localhost</address>
                <port>9090</port>
                <updateinterval>0</updateinterval>
                <measurefactor>0.1</measurefactor>
        </visar>
        <camera>
                <stereo>false</stereo>
                <grabinterval>200</grabinterval>
                <address1>http://IP/mjpg/video.mjpg?resolution=1280x1024&amp;.mjpg
                    </address1>
                <address2>http://IP/mjpg/video.mjpg?resolution=1280x1024&amp;.mjpg
                    </address2>
                <calibration>
                        <usecalibration>false</usecalibration>
                        <intrinsicmatrixpath>Intrinsics.xml</intrinsicmatrixpath>
                        <distortioncoeffspath>Distortion.xml</distortioncoeffspath
                            >
                </calibration>
        </camera>
        <origin>
                <X>568</X>
                <Y>509</Y>
        </origin>
        <analyse>
                <analyse>true</analyse>
                <maximumtime>1000</maximumtime>
                <highlight>true</highlight>
        </analyse>
        <debug>
                <saveruntime>false</saveruntime>
                <showobjectstatistic>true</showobjectstatistic>
        </debug>
        <tolerance>
                <aspectangle>1.5</aspectangle>
                <object>
                        <coordinate>5.0</coordinate>
                        <angle>4.0</angle>
                </object>
        </tolerance>
</configuration>
```

Abbildung 3.1: Hauptfenster des entwickelten Assistenzsystems.

3 User Interface

In dem Programm sind zwei User Interface enthalten. Zum einen kann das Programm in der Kommandozeile gestartet werden oder mit einer Grafischen Oberfläche.

Kommandozeile

Es wurde die Möglichkeit geschaffen in der Konfiguration diese Ausgabe zu deaktivieren da die Software auch ohne grafische Ausgabe laufen soll. Dazu muss in der Konfiguration der Eintrag von *showui* auf *false* gesetzt werden.

GUI

Das Programm kann auch komfortabel mit einer GUI aufgerufen werden (siehe Abbildung 3.1). Sie bietet die Möglichkeit die Software zu kalibrieren, hierzu zählt die Kamerakalibrierung und die Festlegung des Drehmittelpunktes, sowie einige weitere Einstellungsmöglichkeiten.

1. In diesem Abschnitt werden die erkannten Objekte angezeigt.
2. Dieser Bereich beinhaltet die Konsolen-Ausgaben.
3. Das aufgenommene Kamerabild.
4. Der Schalter *"Calibrate Input"* lädt die in der Einstellung definierten Kalibrierungsparamter. Diese werden dann im weiteren dazu genutzt das Kamerabild zu transformieren.

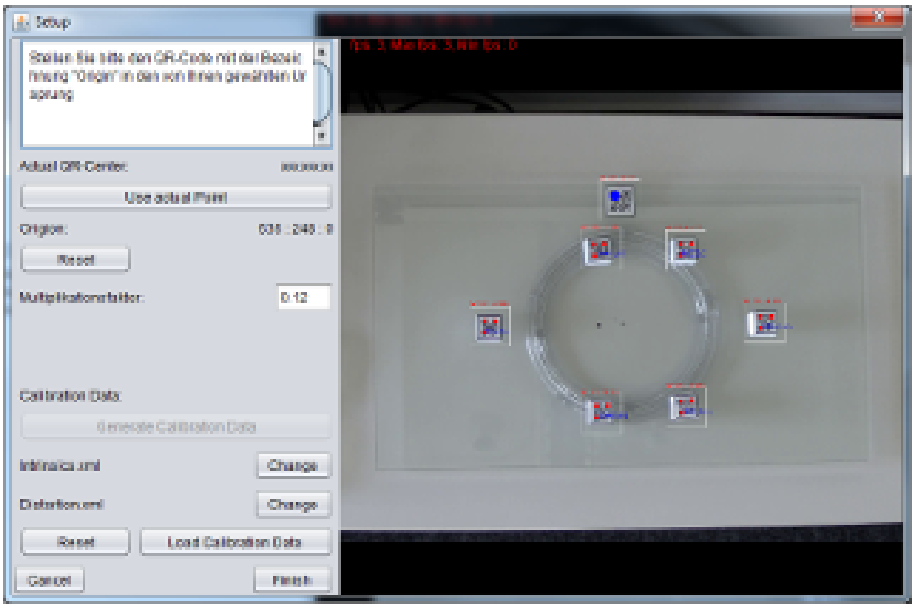


Abbildung 3.2: Das Fenster indem die Einstellungen vorgenommen werden können.

5. Dieser Schalter aktiviert die Analyse der aufgenommenen Bilder.

6. Der Schalter *"Setup"* wird genutzt um die Software einzustellen . Wird dieser gedrückt, öffnet sich das in Abbildung 3.2 gezeigte Fenster.

Setup

Im Setup können wie zuvor beschrieben einige Einstellungen vorgenommen werden (siehe Abbildung 3.2).

Mittelpunkt bestimmen Der Mittelpunkt der Glasplatte kann eingestellt werden wobei der blaue Punkt oberhalb diesem liegen muss. Klicken Sie auf die Schaltfläche mit der Beschriftung *"Use actual Point"*. Ein grüner Strich erscheint, der den Mittelpunkt schneidet. Danach muss die Platte um ca. $45°$ im Uhrzeigersinn gedreht und der Schalter noch einmal betätigt werden. Es sind nun zwei grüne Linien und ein roter Punkt zu sehen (siehe Abbildung 3.3). Dieser Punkt stellt den berechneten Mittelpunkt der Platte da.

Mit dem Schalter *"Reset"* wird der Mittelpunkt auf $(0, 0)$ gesetzt, der der oberen linken Ecke des Bildes entspricht.

Multiplikationsfaktor Der Multiplikationsfaktor wird mit jedem QR-Code Mittelpunkt multipliziert. Dies ist notwendig um die unterschiedlichen Größenverhältnisse in ViSAR abzubilden.

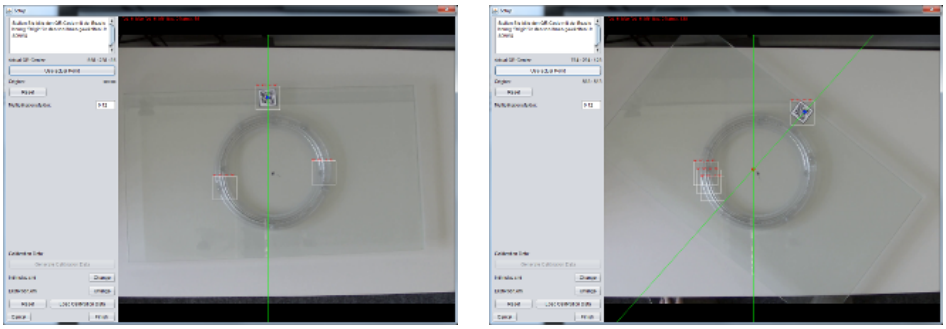


Abbildung 3.3: Bestimmung des Mittelpunktes.

Kamerakalibrierung Die Pfade der Kalibrierungsparameter können mit den zwei Schaltern “Change”, welche in Abbildung 3.2 zu sehen sind, eingestellt werden. Mit einem Klick auf die Schaltfläche *“Load Calibration Data”* werden die Parameter geladen.