Automatische Einrichtung von Multi-Monitor-Systemen mithilfe von optischen Bezugsmarkern

Eva-Maria Brucker

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im November 2016

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 21. November 2016

Eva-Maria Brucker

Inhaltsverzeichnis

Kurzfassung

Das manuelle Einrichten eines Multi-Monitor-Systems ist ein langwieriger und aufwändiger Vorgang. Die Distanzen zwischen einzelnen Bildschirmen und deren Rotation müssen mithilfe von verschiedensten Messwerkzeugen mühsam bestimmt werden. Dieser Prozess kann mittels optischer Tracking-Methoden vereinfacht und beschleunigt werden. Eine ähnliche Problemstellung findet sich im Bereich der Augmented Reality, bei der die Position von virtuellen Objekten mithilfe von optischen Bezugsmarkern bestimmt wird. Der Lösungsansatz zum Problem der Multi-Monitor-Kalibrierung orientiert sich deshalb an bestehenden AR-Libraries.

Diese Arbeit gibt einen Überblick über existierende Markertypen und deren Merkmale, sowie über einige Augmented Reality-Libraries. Der implementierte Lösungsansatz wird im Detail beschrieben. Es wird überprüft, ob dieser ein geeignetes Mittel zur Kalibrierung von Multi-Monitor-Systemen darstellt.

Abstract

Manual calibration of a multi-monitor system is a rather cumbersome process and takes a lot of time. For each screen the relative distance and orientation to its neighbouring screens has to be measured using several measuring tools. This task could be simplified by using optical tracking methods. Augmented Reality applications face a similar problem. They need to measure the position and orientation of fiducial markers to be able to correctly place virtual objects. Therefore, simplifying the calibration of a multi-monitor system follows a similar approach.

This thesis gives an overview of different types of fiducial markers and Augmented Reality libraries. The implemented approach is described in detail. It is evaluated whether it provides a feasible alternative to manually calibrating a multi-monitor system.

Kapitel 1

Einleitung

Die automatisierte Erfassung von Positions- und Rotationsdaten verschiedener realer Objekte hat in den letzten Jahrzehnten immer wieder neue Anwendungsgebiete gefunden. Vor allem im Bereich *Computer Vision* ist das sogenannte *Tracken* von Objekten zu einer immer wichtigeren Aufgabe geworden. Dies ist auf den steigenden Bedarf an automatisierter Videoanalyse, die hohe Verfügbarkeit von günstiger Kameraausrüstung und die steigende Rechenleistung von Computern zurückzuführen.

Diese Arbeit beschäftigt sich damit, wie diese Technologie für das Kalibrieren des *Multiplayer Multiple Unit*-Systems verwendet werden kann. Zuerst wird dieses und die genaue Problemstellung mitsamt Lösungsansätze näher beschrieben. Anschließend werden einige bestehende Systeme zur Positions- und Orientierungsbestimmung relativ zur Kamera im Bereich Augmented Reality näher beschrieben. Weiters wird die für dieses Projekt verwendete Methode im Detail Schritt für Schritt erklärt. Als nächstes wird diese auf ihre Praxistauglichkeit überprüft. Abschließend wird ein kurzer Überblick über die Ergebnisse und zukünftige Schritte gegeben.

1.1 Multiplayer Multiple Unit (MuMu)

Das Multiplayer Multiple Unit-System (kurz MuMu) ist ein von *Quantum Reboot*[1] entwickeltes Konzept für Computerspiele oder ähnliche Anwendungen. Das System besteht aus einem Server, welcher ein Spiel hostet und zu dem sich mehrere Clients verbinden können. Diese Clients zeigen jedoch nicht, wie es üblicherweise der Fall ist, einen Ausschnitt des Spiels für einen bestimmten Spieler an, sondern jeweils einen kleinen Teil der gesamten Spielwelt. Alle Clients werden dann in einer beliebigen Anordnung neben- und übereinander aufgebaut, wie es zum Beispiel in Abbildung 1.1 zu sehen ist und zeigen zusammen eine große Spielwelt an. Ein Client besteht aus einem

[1] http://quantumreboot.com/

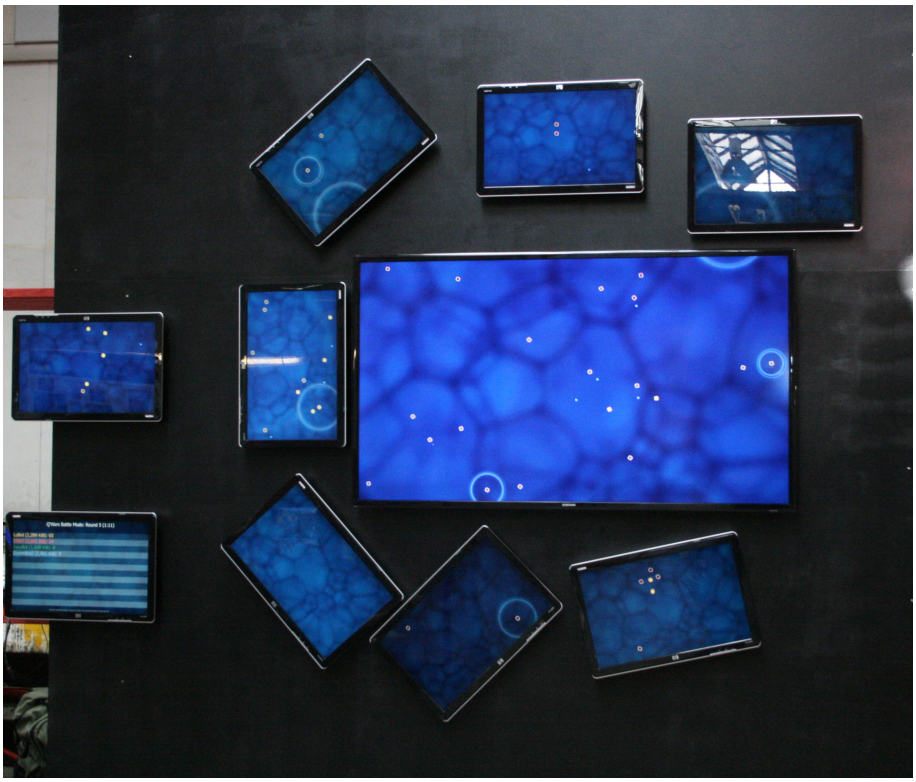


Abbildung 1.1: Live-Setup des MuMu-Systems. © Quantum Reboot 2016.

Monitor und einem *Raspberry Pi* Mikrocomputer, was das Setup kostengünstig und leicht erweiterbar macht, und wird als *Unit* bezeichnet.

Beim Aufbau dieses Systems müssen am Server die Positions- und Rotationsdaten der Bildschirme aller Units eingestellt werden. Dies kann in der Serveranwendung direkt durchgeführt werden, wie in Abbildung 1.2 zu sehen ist. Die *Viewports* aller verbundenen Units scheinen dort auf und können verschoben, rotiert und skaliert werden. Als Viewport wird der auf dem Monitor angezeigte Bildausschnitt bezeichnet.

1.2 Problemstellung

Momentan ist die Kalibrierung des MuMu-Systems aufwändig und umständlich. Die relativen Abstände zwischen den einzelnen Bildschirmen und deren Rotation müssen händisch gemessen oder ungefähr abgeschätzt werden und manuell bei der Serveranwendung eingetragen werden. Diese Werte sind oft noch ungenau und müssen dementsprechend Bildschirm für Bildschirm angepasst werden. Für diese Messungen werden verschiedenste Messwerkzeuge, wie Maßbänder, Wasserwagen oder Winkelmesser benötigt. Außerdem kann

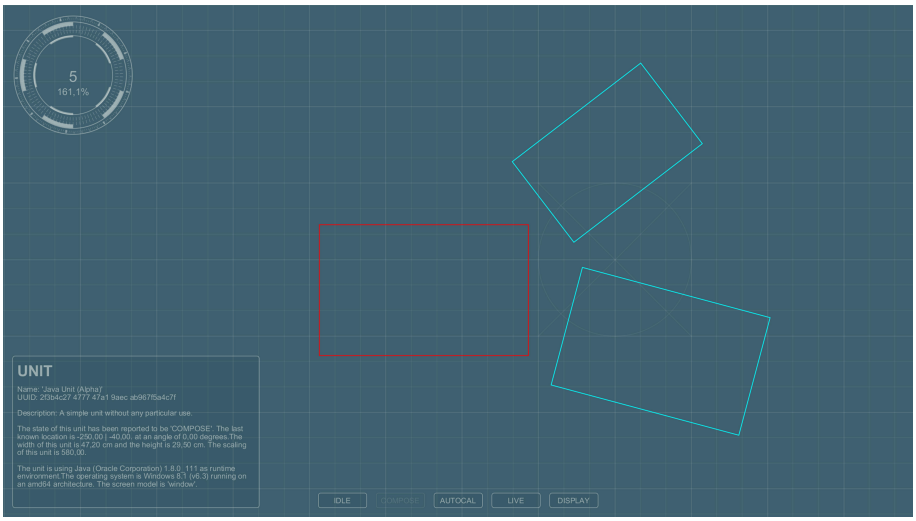


Abbildung 1.2: Server-Anwendung des MuMu-Systems. Die Positions- und Rotationsdaten der einzelnen Bildschirme können hier eingestellt werden. © Quantum Reboot 2016.

die Kalibrierung alleine meist nur sehr umständlich durchgeführt werden, vor allem wenn Bildschirme weit voneinander entfernt sind. Je nach Anzahl der verwendeten Bildschirme, nimmt dieser Vorgang außerdem viel Zeit in Anspruch.

Aus diesen Gründen soll eine einfachere und schnellere Möglichkeit für den Kalibrierungsvorgang des MuMu-Systems entwickelt werden. Es soll eine Lösung gefunden werden, die es Benutzern des Systems erlaubt, die Viewports der einzelnen Bildschirme ohne großen Aufwand auf der Server-Anwendung einzustellen.

1.3 Lösungsansätze

In [17] sind verschiedene mögliche Methoden für das Tracken von realen Objekten beschrieben. Eine Möglichkeit wäre, die einzelnen Bildschirme des MuMu-Systems mithilfe verschiedener Sensoren, wie zum Beispiel magnetischen, mechanischen oder Trägheitssensoren, auszustatten. Eine gute Übersicht über verschiedene Sensoren ist in [13] gegeben. Je ein Sensor pro Bildschirm könnte dessen Rotation messen und weitere Sensoren könnten verwendet werden, um den Abstand zwischen den einzelnen Monitoren zu ermitteln. Diese Lösung würde jedoch je nach Größe des Aufbaus viel zusätzliche Ausrüstung benötigen. Dies ist einerseits mit höheren Kosten verbunden, andererseits ist es aufwändiger den aktuellen Aufbau eines MuMu-Systems zu erweitern, da für jeden neuen Bildschirm weitere Sensoren benötigt wer-

den.

Eine weitere Möglichkeit, den Aufbau eines MuMu-Projekts automatisch zu erkennen, und welche für diesen Anwendungsfall geeigneter wäre, ist optisches Tracking. Bildschirme können in einem mit einer beliebigen Kamera aufgenommenen Bild erkannt und anhand eines darauf platzierten Markers identifiziert werden. Sind Kalibrierungsdaten zur verwendeten Kamera und die Abmessungen der einzelnen Bildschirme bekannt, so kann deren relative Position und Orientierung zueinander berechnet und vom MuMu-System verwendet werden. Diese Kalibrierungsmethode soll es erlauben, ohne viel zusätzlichen Aufwand alle Bildschirme möglichst genau einzurichten. Bis auf eine kalibrierte Kamera wird keine zusätzliche Ausrüstung benötigt. Weiters ist es möglich jederzeit eine beliebige sich ändernde Anzahl an Bildschirmen zu erkennen, ohne diese vorher speziell ausstatten zu müssen. Es müssen lediglich deren Bildschirmflächen abgemessen werden.

Aufgrund der oben erwähnten Vorteile von optischem Tracking wurde diese Methode umgesetzt und evaluiert. Es soll festgestellt werden, ob mittels optischem Trackings brauchbare und praxistaugliche Kalibrierungswerte für die Bildschirme eines MuMu-Systems ermittelt werden können.

Kapitel 2

State of the Art

2.1 Optische Bezugsmarker

Unter optischen Bezugsmarkern, im Weiteren als Marker bezeichnet, versteht man, wie in [1] beschrieben, ein Objekt, welches im Sichtfeld einer Kamera platziert wird und später im aufgenommenen Bildmaterial sichtbar ist. Dieses wird anschließend als Referenz- oder Messpunkt für verschiedenste Berechnungen verwendet. Um die einzelnen Bildschirme des MuMu-Systems in einem Bild identifizieren zu können, können solche Marker auf diesen platziert und später im Bild erkannt werden. Nachstehend werden Kriterien für optische Marker sowie verschiedene Markertypen und deren Anwendungen beschrieben.

2.1.1 Kriterien für Markerqualität

In [1] sind Kriterien für die Qualität eines Markers aufgelistet. Diese werden nachstehend näher beschrieben.

Form

Die Aufgabe eines Markers ist es, eine Verbindung zwischen der realen dreidimensionalen Welt und dem zweidimensionalen Bild herzustellen. Um die Position und Rotation eines Objektes relativ zur Kamera bestimmen zu können, werden mindestens vier koplanare Punkte benötigt, um ein eindeutiges Ergebnis zu erhalten. Mit nur drei Punkten gibt es mehrere mögliche Lösungen, wie in [7] beschrieben ist. Damit stellen Vierecke eine optimale Form für Marker dar. Da bei unterschiedlichen Perspektiven die Kanten eines Markers verzerrt dargestellt werden, können Marker mit unterschiedlichen Kantenlängen je nach Betrachtungswinkel andere Ergebnisse liefern. Quadratische Marker sind deshalb zu bevorzugen, da deren Kanten keine Perspektive begünstigen.

Farbe

Die Verwendung von farbigen Markern erhöht die Anzahl der möglichen unterschiedlichen Marker. Für diese müssen jedoch auch komplexere Algorithmen zur Erkennung und Unterscheidung einzelner Marker verwendet werden, welche eine höhere Laufzeit aufweisen. Weiters unterstützen verschiedene Kameras unterschiedliche Farbräume, was die Identifizierung farbiger Marker zusätzlich erschwert. Ein weiteres Problem ist, dass die Anzahl an Farben, die eindeutig voneinander identifiziert werden können sehr stark variiert. Diese hängt von der Beleuchtung der aktuellen Umgebung ab und ist meist eher gering. Außerdem können Reflexionen Farbtöne verändern, so dass diese nicht mehr eindeutig zugewiesen werden können. Zusätzlich muss darauf geachtet werden, dass die verwendeten Farben sich von der Umgebung abheben. Aus all diesen Gründen ist es praktikabler monochrome Marker zu verwenden.

Positionierung

Um Marker in einem Bild leichter erkennen zu können, sollten diese auf einer dazu kontrastreichen Oberfläche platziert werden. Deswegen werden schwarze Marker meist auf weißem Hintergrund gedruckt.

Identifizierung

Um Marker identifizieren zu können, ist es von Vorteil wenn diese ein eindeutiges Merkmal enthalten. Würden mehrere Marker gleich aussehen, so könnten diese nur durch ihre relative Anordnung zueinander identifiziert werden. Marker können durch die Verwendung verschiedener Farben und Muster voneinander unterschieden werden. Außerdem sollen Marker auch bei unterschiedlichen Auflösungen noch eindeutig identifiziert werden können. Weiters sollte der Unterschied zwischen den einzelnen Markern möglichst hoch sein, um Verwechslungen zu vermeiden. Idealerweise können Marker auch noch erkannt werden, wenn sie teilweise verdeckt werden oder das aufgenommene Bild einen hohen Rauschanteil aufweist.

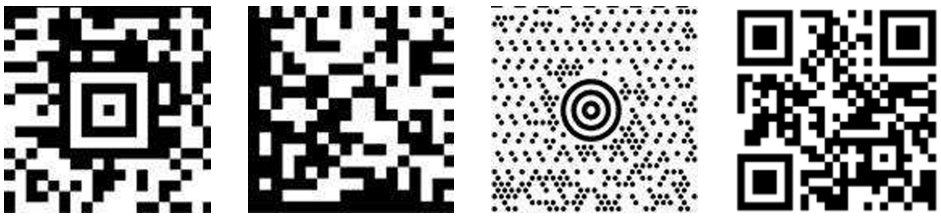
2.1.2 Markertypen

Die verschiedenen Markertypen basieren auf den zur Erkennung dieser verwendeten Tracking-Algorithmen. In [9] und [1] werden nachstehende Markertypen unterschieden.

ID-Marker

ID-Marker sind rechteckige 2D-Marker für simple Tracking-Anwendungen. Sie verfügen über eine fixe Struktur und haben meist einen charakteristischen schwarzen Rahmen. Dadurch können sie leicht und robust erkannt

Abbildung 2.1: Beispiele für ID-Marker aus [9] und [1].



(a) (b) (c) (d)

Abbildung 2.2: Beispiele für die Datenmarker Aztec Code (a), Data Matrix (b), Maxi Code (c) und QR Code (d) aus [1].

und getracked werden. Das Innere eines ID-Markers besteht aus schwarzen und weißen Datenzellen, welche die ID des jeweiligen Markers kodieren. Beim Erkennen der Marker werden die inneren schwarzen und weißen Quadrate binär kodiert. In Abbildung 2.1 sind Beispiele für ID-Marker zu sehen.

Datenmarker

Datenmarker sind ähnlich wie ID-Marker, können jedoch mehr Informationen enthalten. Sie können ebenfalls über einen schwarzen Rahmen verfügen und ihr Inneres stellt eine maschinenlesbare 2D-Repräsentation von Daten dar. Einige der meistgenutzten Datenmarker sind *Aztec Code*, *Data Matrix*, *Maxi Code* und *QR Code*. Diese sind in Abbildung 2.2 zu sehen. Da Datenmarker dazu entwickelt wurden, mehr Daten als eine einfache ID zu beinhalten, sind diese größer und benötigen mehr Pixel in einem Bild, um erkannt werden zu können. Aus diesem Grund können Datenmarker meist nur aus geringer Distanz abfotografiert werden.

Template-Marker

Als Template-Marker werden Graustufen-Marker mit einem einfachen Bild im Inneren und einem schwarzen Rahmen bezeichnet. Bei der Verwendung dieses Markertyps wird eine Datenbank mit allen ausgewählten Marker-Templates angelegt. Bei der Identifizierung eines gefundenen Markers, wird

Abbildung 2.3: Beispiele für Template-Marker aus [1]

Abbildung 2.4: Beispiele für Kreismarker aus [1].

dieser mit allen in der Datenbank enthaltenen Markern verglichen. Das Template mit der größten Übereinstimmung wird dann mit dem gefundenen Marker assoziiert. Beispiele für Template-Marker sind in Abbildung 2.3 ersichtlich. Zusammen mit ID-Markern sind Template-Marker die meistgenutzten Marker im Bereich *Augmented Reality*[1].

Kreismarker

Kreismarker werden hauptsächlich im Bereich der *Photogrammetrie*[2] verwendet. Photogrammetrische Anwendungen benötigen eine hohe Genauigkeit, jedoch ist die Laufzeit des verwendeten Algorithmus kaum relevant. Um die Position eines Objektes relativ zur Kamera genau zu berechnen, werden mindestens vier Punkte benötigt. Kreismarker liefern nur eine Position, diese jedoch mit hoher Genauigkeit. Aus diesem Grund werden diese Marker meist nicht zur relativen Positionsberechnung verwendet, da mindestens vier einzelne Marker für die Berechnung eines Punktes platziert werden müssten. In Abbildung 2.4 sind Beispiele für Kreismarker zu sehen.

[1] Unter Augmented Reality versteht man die Platzierung virtueller Objekte in der realen Welt [11].

[2] Photogrammetrie bezeichnet eine Anzahl an Messmethoden und Auswerteverfahren, die dazu dienen aus genauen Messbildern die dreidimensionale Form und Lage eines Objektes zu bestimmen [18].

Abbildung 2.5: Beispiele für Bildmarker aus [9].

Unauffällige Marker

Optische Bezugsmarker sind gut für die Berechnung der relativen Position zur Kamera geeignet. Manchmal sind jedoch keine visuellen Marker erwünscht, weil diese das Umgebungsbild stören würden. Falls keine der oben erwähnten schwarz-weiß Marker mit hauptsächlich für den Computer erstellten Mustern verwendet werden sollen, können auch beliebige Farbbilder als Marker definiert werden. Sollte auch kein Farbbild sichtbar sein, so existieren weitere Möglichkeiten für Menschen nicht wahrnehmbare Marker zu verwenden. Diese werden im Folgenden näher beschrieben.

Bildmarker: Für Bildmarker können beliebige Farbbilder verwendet werden. Diese besitzen meist entweder eine scharfe Bildkante oder andere Merkmale, um die Erkennung zu erleichtern. Für die Identifizierung einzelner Marker werden Template- oder Feature-Matching Algorithmen verwendet. Der Vorteil von Bildmarkern besteht darin, dass diese in einer existierenden Umgebung funktionieren, ohne diese verändern zu müssen, da zum Beispiel ein Fotoausschnitt der Umgebung als Marker registriert werden kann. Bildmarker mit Rahmen können schneller erkannt werden als ohne. Rahmenlose Bildmarker sollten ein möglichst simples Bild mit charakteristischen 2D-Features enthalten, um eine robuste Erkennung der Marker zu ermöglichen. Beispiele für Bildmarker sind in Abbildung 2.5 zu sehen.

Infrarotmarker: Für das menschliche Auge sind Infrarotmarker nicht sichtbar. Es können selbstleuchtende Marker, reflektierendes Material, Infrarot-Spotlights oder Infrarot-Projektoren verwendet werden, um Marker auf einer Oberfläche zu platzieren. Mit einer speziellen Infrarotkamera können diese erkannt werden. Diese Technologie funktioniert jedoch nicht unter freiem Himmel, da dort die Sonne eine unkontrollierbare Infrarotlichtquelle darstellt.

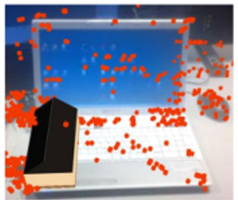


Abbildung 2.6: Beispiel für einen markerlosen 3D-Marker aus [9] mit im Bild platzierten 3D-Inhalten.

Kleine Marker: Eine weitere Möglichkeit Marker zu verstecken, ist diese so klein zu skalieren, bis sie vom Menschen nicht mehr wahrgenommen werden. Ein Beispiel hierfür ist das Marker-System *Bokode* des *MIT Media Lab* aus [19]. Die darin verwendeten Marker sind unauffällig gestaltet, können jedoch von handelsüblichen Kameras aus einigen Metern Entfernung erkannt werden.

Markerlose 3D-Marker: Die leicht irreführende Bezeichnung *Markerlose 3D-Marker* steht für eine fortschrittliche optische Trackingmethode, welche das Tracken beliebiger realer Objekte ermöglicht. Um ein Objekt erkennen zu können, muss eine Sammlung aus dessen charakteristischen Features erstellt werden, um diese später in einem Bild wieder erkennen zu können. Das verwendete Objekt muss jedoch genug visuelle Features aufweisen, um robust erkannt werden zu können. Weiters muss es zuvor aus mehreren Perspektiven gescannt werden. Ein Beispiel hierfür ist in Abbildung 2.6 zu sehen.

2.2 Augmented Reality Libraries mit planaren Markern

Bei bestehenden Augmented Reality Libraries werden meist ID-Marker zur Berechnung der relativen Kameraposition verwendet. Da diese auch für die Kalibrierungsanwendung des MuMu-Systems der geeignetste Markertyp sind, wird nachstehend die Funktionsweise einiger solcher Libraries beschrieben.

Wie in [5] und [3] beschrieben, bestehen Bezugsmarkersysteme aus Markermustern, die sich je nach System unterscheiden, und Computer Vision-Algorithmen, um diese zu erkennen. Die Muster werden dabei an beliebigen Stellen in der Umgebung angebracht und später in den aufgenommenen Kamerabildern erkannt und identifiziert. Bezugsmarkersysteme werden hauptsächlich im Bereich der Augmented Reality und für die Navigation von Robotern verwendet. Sie können jedoch für alle Anwendungen, bei denen die

relative Position zwischen der Kamera und einem beliebigen Objekt im Bild benötigt wird, genutzt werden.

Wie in [5] beschrieben ist, werden die Marker in solchen Systemen meist in zwei Schritten erkannt. Zuerst wird versucht, das einzigartige Feature eines Markers im Bild zu erkennen. Anschließend werden die so gefundenen Marker anhand ihres Musters identifiziert. Viele Markersysteme verwenden einen vierseitigen Rahmen als Erkennungsfeature ihrer Marker, da deren vier Eckpunkte für die Entzerrung des inne liegenden Markermusters verwendet werden können.

In [15] sind die Voraussetzungen für Augmented Reality Libraries aufgelistet. Diese müssen die genaue Erkennung der Positionen ihrer Marker in Echtzeit ermöglichen. Die dazu verwendete Ausrüstung soll mit möglichst niedrigen Kosten verbunden sein und trotzdem robuste Ergebnisse in unterschiedlichen Umgebungen liefern können. Vor allem für Anwendungen für mobile Geräte müssen diese Ansprüche erfüllt werden, da hier nur limitierte technische Ressourcen verfügbar sind. Es empfiehlt sich die Nutzung von ID-Markern, da diese eine robuste und recheneffiziente Methode darstellen, die gleichzeitig kostengünstig und schneller als zum Beispiel die Erkennung von Features in der natürlichen Umgebung ist.

In [3] sind folgende wichtige Parameter für die Bewertung solcher Markersysteme beschrieben:

- die *False-Positive-Rate*, sie gibt an, wie viele Marker fälschlicherweise als solche erkannt werden,
- die *False-Negative-Rate*, sie gibt an, wie viele existierende Marker im Bild nicht gefunden werden,
- die *Inter-Marker-Confusion-Rate*, sie gibt an, wie viele Marker zwar richtig erkannt, jedoch falsch identifiziert und mit anderen Markermustern verwechselt werden,
- die Erkennungsrate unter verschiedenen Lichtbedingungen,
- die Erkennungsrate bei unscharfen Bildern,
- die kleinstmögliche erkennbare Größe eines Markers, sie gibt an, wie viele Pixel mindestens für eine verlässliche Erkennung des Markers notwendig sind. Je kleiner dieser Wert ist, desto größere Bildausschnitte der Umgebung können aufgenommen werden.

2.2.1 ARToolKit

ARToolKit[3] wurde erstmals 1999 vorgestellt und 2001 veröffentlicht. Es ist eines der weit verbreitetsten Augmented Reality-Markersystemen, da es einfach zu verwenden und frei verfügbar ist. Außerdem unterstützt es alle gängigen Plattformen. Wie in [5] beschrieben ist, verwendet dieses System

[3] https://artoolkit.org/

Abbildung 2.7: Beispiele für typische ARToolKit-Marker aus [4] und [1].

Template-Marker. Diese bestehen aus einem quadratischen schwarzem Rahmen und einer darin enthaltenen ID auf weißem Hintergrund. Die ID eines Markers ist ein Graustufenbild, welches vom Benutzer beliebig festgelegt werden kann. Beispielmarker dieses Systems sind in Abbildung 2.7 zu sehen.

Die zwei Schritte zur Erkennung eines Markers funktionieren bei ARToolKit, wie in [5] und [4] beschrieben, wie folgt: Zuerst wird bei der Erkennung des Markers im Bild nach vierseitigen Formen mit schwarzem Rand, welche Marker sein könnten, gesucht. Hierzu wird das zu durchsuchende Bild mittels Thresholding zuerst in ein Binärbild, welches nur noch schwarze und weiße Pixel enthält, umgewandelt. Der dabei verwendete Grenzwert ist der einzige Parameter, der für die Erkennung konfiguriert werden kann. Er wird sowohl für die Erkennung der Marker im Bild, als auch für die Identifizierung des Markermusters verwendet. Nach der Umwandlung in ein Binärbild wird versucht, Kanten im Bild zu erkennen. Hierzu werden Gruppen aus zusammenhängen unter dem Grenzwert liegenden Pixeln gebildet. Anschließend werden die Konturen dieser Gruppen gesucht. Jene Konturen, welche aus vier geraden zusammenhängenden Linien bestehen, werden als potenzielle Marker identifiziert.

Beim zweiten Schritt wird das Innere der gefundenen möglichen Marker untersucht, um herauszufinden, ob es sich tatsächlich um einen Marker handelt. Die vier Eckpunkte des gefundenen Markers werden verwendet, um die perspektivische Verzerrung vom Bild zu entfernen und das innere Markermuster in eine Frontalansicht zu bringen. Anschließend wird dieses in einem 16×16 Raster abgetastet, so dass jedes Rasterfeld am Ende einen Graustufenwert enthält. Um eine höhere Genauigkeit bei der Unterscheidung zwischen verschiedenen Mustern zu ermöglichen, wird dieses Raster manchmal auf eine Größe von 32×32 erhöht. Das Ergebnis dieses Schrittes ist ein 16×16 oder 32×32 großer Vektor. Für diesen wird mit den Vektoren aller möglichen gespeicherten Markermustern ein normalisiertes Skalarprodukt, der sogenannte *Confidence Faktor*, berechnet. Dabei werden auch die möglichen Rotationen eines Markers beachtet, so dass dieser Schritt pro Marker vier Mal, je einmal für jede Richtung, durchgeführt werden muss. Liegt der berechnete Wert über oder unter einem bestimmten Grenzwert, so wurde ei-

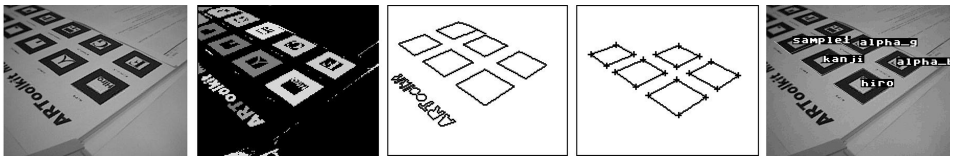


Abbildung 2.8: Markererkennung und Identifizierung von ARToolKit aus [4]. Von einem aufgenommenen Bild wird mittels eines festgelegten Grauwert-Grenzwerts ein Binärbild erstellt. Von den darin zusammenhängenden Pixelgruppen, welche entweder über oder unter dem Grenzwert liegen, werden die Konturen gesucht. Aus diesen werden jene herausgefiltert, welche vier zusammenhängende gerade Kante besitzen. Das innere dieser Marker wird identifiziert und den Bildern können passende Muster zugeordnet werden.

ne Übereinstimmung gefunden oder nicht. Die einzelnen Erkennungsschritte von ARToolKit sind in Abbildung 2.8 zu sehen.

Um einen eigenen Marker mit ARToolKit verwenden zu können, müssen, wie in [4] beschrieben, dessen Referenzbilder zuerst manuell der Markerdatenbank hinzugefügt werden. Dazu muss der neue Marker ausgedruckt und abfotografiert werden. Dies sollte optimalerweise unter den gleichen Lichtbedingungen und mit der selben Kamera erfolgen, wie beim später verwendeten tatsächlichen Setup. Insgesamt müssen zwölf Versionen des neuen Markers erstellt werden, jeweils drei pro möglicher Ausrichtung. Diese drei Versionen sollen dabei unter unterschiedlichen Lichtbedingungen und aus verschiedenen Distanzen aufgenommen werden. Dadurch, dass mehrere mögliche Perspektiven eines Markers in der Datenbank abgespeichert werden, kann die Robustheit des Identifizierungsprozesses erhöht werden. Werden jedoch zu viele Markermuster zur Datenbank hinzugefügt, kann dies, wie in [3] erwähnt ist, sowohl die Inter-Marker-Confusion-Rate erhöhen, als auch eine Verschlechterung der Laufzeit mit sich bringen, da ein Marker bei seiner Identifizierung mit allen anderen in der Datenbank gespeicherten Markern verglichen werden muss.

Die Schwächen und Stärken von ARToolKit werden in [4] und [5] aufgelistet. Der verwendete Algorithmus liefert beim Finden der Marker robuste Ergebnisse. Jedoch weist er höhere False-Positive- und Inter-Marker-Confusion-Raten auf. Marker werden häufig miteinander verwechselt oder fälschlicherweise im Hintergrund erkannt. Dieses Problem kann durch die Verwendung von möglichst unterschiedlichen Markern abgeschwächt werden. Weiters liefert das System unter kontrollierten Lichtbedingungen bessere Ergebnisse. Die False-Positive- und False-Negative-Rate ist außerdem durch den Benutzer direkt manipulierbar, da dieser den für die Berechnungen verwendeten Grenzwert einstellen kann. Die Verwendung eines einzelnen Grenzwertes limitiert jedoch das Einsatzgebiet des Systems, da kontrollierbare Lichtverhältnisse vorausgesetzt werden. Deswegen ist die am weitesten

Abbildung 2.9: Beispiele für typische ARTag-Marker aus [4].

verbreitete Adaption dieses Systems auch ein zusätzlicher Pre-Processing-Schritt für automatisches Thresholding, welches bessere Ergebnisse liefert, jedoch auch die Laufzeit des Algorithmus erhöht. Eine weitere Lösung für dieses Problem ist, ein gesamtes Bild mehrmals mit verschiedenen Grenzwerten zu überprüfen. Ein Vorteil dieses Systems im Vergleich zu anderen, ist das ansprechendere Aussehen der Marker. Da beliebige Graustufenbilder verwendet werden können, können diese zum Beispiel der Verwendung entsprechend gestaltet werden.

2.2.2 ARTag

ARTag wurde 2004 im *National Reasearch Council of Canada* entwickelt. Wie in [4] erwähnt, war es das Ziel, eine robustere Version der vorher beschriebenen Marker-Library ARToolKit zu erstellen. ARTag wurde von dieser beeinflusst, die Schritte zur Erkennung und Identifizierung der Marker wurden jedoch anders gelöst. Die Marker sind ebenfalls quadratisch und besitzen einen Rahmen, dieser kann jedoch schwarz oder weiß sein. Anstatt eines Template-Markers wurden ID-Marker verwendet. Beispiele für ARTag-Marker sind in Abbildung 2.9 zu sehen.

Die einzelnen Schritte der Markererkennung und -identifzierung sind in [4] und [5] beschrieben. Zuerst wird, wie bei ARToolKit, versucht Vierecke in einem Bild zu erkennen. Anstatt Thresholding wird hier jedoch ein kantenbasierter Ansatz verwendet. Dieser ist robuster unter verschiedenen Lichtbedingungen und kann selbst partiell verdeckte Marker erkennen. Der eigentliche Vorteil gegenüber ARToolKit liegt jedoch im Identifizieren der Marker. Da das Innere der Marker durch eine digitale Kodierungsform ersetzt wurde, können Marker besser voneinander unterschieden werden. Diese Änderung führt zu einer verschwindend geringen False-Positive- und Inter-Marker-Confusion-Rate des Systems. Die Wahrscheinlichkeit einen Marker mit einem anderen zu verwechseln liegt bei weniger als 0,0039%, wie in [3] zu sehen ist.

Wie in [4] beschrieben ist, besteht das Innere eines ARTag-Markers aus einem 6×6 großen Raster, es stehen somit 36 Bits für die Kodierung der ID zur Verfügung. Der Rand eines Markers ist zwei Rastereinheiten breit und

kann entweder schwarz oder weiß sein. Bei der Kodierung der ID werden zusätzliche Maßnahmen für eine robustere Identifizierung getroffen. Diese sind in [4] genauer beschrieben. 26 der Bits werden nicht für die Kodierung der ID, sondern für Prüfsummen und Vorwärtsfehlerkorrektur verwendet. Dies ermöglicht die Erkennung des ID-Musters trotz einiger verdeckter oder falsch erkannter Bits im Inneren des Markers. Die Verwendung solcher Methoden zur Fehlerkorrektur verbessert zwar die False-Negative-Rate, erhöht jedoch die False-Positive-Rate, da Teile des Bildes, welche keine Marker beinhalten als solche erkannt werden können, wenn nur wenige Bits Unterschied zu existierenden Markermustern bestehen. Dieser Fehler kann bei einem schlecht gesetzten Grenzwert zur Identifizierung eines Markermusters, Reflexionen oder Rauschen im Bild auftreten.

Die Vorteile ARTags gegenüber ARToolKit sind in [4] beschrieben. Durch die Verbesserungen bei der Identifzierung der ID liegt die Rate der fälschlicherweise erkannten Markern bei unter 0,0079%. Wie in [5] zu sehen ist, liegt die False-Negative-Rate von ARTag unter der von ARToolKit und ARToolKit Plus. Weiters müssen die Markermuster nicht in einer Datenbank gespeichert werden. Dies führt zu einer verbesserten Performance bei der Identifizierung eines Markers. Insgesamt können 2046 verschiedene Markermuster verwendet werden. Diese Anzahl ist hauptsächlich der zusätzlichen Verwendung weißer Rahmen zuzuschreiben, da dadurch die Anzahl möglicher Marker verdoppelt werden kann, ohne die Größe oder Komplexität der Marker verändern zu müssen.

2.2.3 ARToolKit Plus

Das in 2005 entwickelte *ARToolKit Plus* wurde von der Robustheit beim Erkennen und Identifizieren der Marker von ARTag inspiriert, wie in [5] beschrieben ist. Es wurden jedoch nicht alle Features von ARTag in ARToolKit Plus übernommen. Die Erkennung einzelner Marker erfolgt wie bei ARToolKit und ist nicht kantenbasiert. Bei der Identifizierung eines Markers werden auch keine Methoden zur Fehlerkorrektur verwendet.

Wie in [15] beschrieben, fokussiert sich ARToolKit Plus auf die Optimierung für mobile Geräte, da zu dieser Zeit ein starker Anstieg in der Verbreitung von mobilen Geräten mit eingebauten Kameras zu verzeichnen war. Das Ziel war es, eine möglichst gute Performance auf einer großen Zahl verschiedener Geräte zu erreichen. In [15] werden die Anpassungen, welche für eine mobile Umsetzung nötig waren näher beschrieben.

Die Marker von ARToolKit Plus ähneln denen von ARTag. Wie in [5] beschrieben ist, bestehen diese aus einem quadratischen schwarzen Rahmen mit einem 6×6 großen darin enthaltenen schwarz-weißen ID-Muster. Die ID wird jedoch anders kodiert als bei ARTag. Der Grund für die Verwendung des ID-Musters anstelle des Template-Musters ist, wie in [15] beschrieben, dass die Anzahl gleichzeitig zur Laufzeit verwendbarer Marker durch die

höhere Suchzeit in der Template-Datenbank auf mobilen Geräten stark eingeschränkt wäre. Insgesamt werden 4096 verschiedene Markermuster unterstützt.

Da sich mobile Anwendungen, wie in [15] beschrieben ist, auf größere Bereiche, sowohl im Innen- und Außenbereich, erstrecken können, muss sich der Markererkennungs-Algorithmus auf ändernde Lichtbedingungen anpassen können. Aufgrund der bei ARToolKit verwendeten konstanten Grenzwerte ist dies jedoch nicht möglich. Deswegen wurde bei ARToolKit Plus eine simple aber effektive Heuristik zur Ermittlung eines passenden Grenzwertes implementiert, welche keinen messbaren Performanceverlust verursacht. Nachdem ein Marker im Bild gefunden wurde, wird für den nächsten Durchlauf der Median von allen Pixeln des gefundenen Markers als Grenzwert verwendet. Konnte kein Marker gefunden werden, so wird für jeden Durchlauf so lange ein zufälliger Wert gesetzt, bis wieder ein Marker gefunden wird. Eine weitere Optimierung für mobile Geräte bezieht sich auf deren Kameras, die häufig eine Vignettierung aufweisen. Marker, die in den äußeren Bereichen des aufgenommenen Bildes liegen, können dadurch manchmal nicht erkannt werden. Deshalb wurde ein manuell einstellbarer Farbabfall vom Zentrum ausgehend eingebaut, welcher der Vignettierung entgegen wirken soll.

Kapitel 3

Entwicklung

Für das Multiplayer Multiple Unit System sollte ein Tool entwickelt werden welches die Kalibrierung der Viewports der einzelnen Bildschirme erleichtert und so weit wie möglich automatisiert. Hierfür wurde eine visuelle Tracking-Methode verwendet. Die aufgebauten Bildschirme sollen mit einer Kamera abfotografiert oder gefilmt werden. Anschließend kann aus den erhaltenen Bildern und den Kameradaten die relative Position und Rotation zwischen den einzelnen Bildschirmen berechnet und an den MuMu-Server gesendet werden. Diese Methode ist nicht nur günstiger und einfacher zu verwenden als wie zum Beispiel die Verwendung von verschiedensten Sensoren, sie ist auch wesentlich flexibler. Es wird nicht für jeden der Bildschirme eigene Hardware benötigt, wie es beispielsweise bei der Verwendung von magnetischen oder elektromagnetischen Sensoren der Fall wäre. So können jederzeit Bildschirme zum Aufbau des aktuellen MuMu-Setups hinzugefügt oder entfernt werden ohne großen zusätzlichen Aufwand bei der Kalibrierung der Viewports.

3.1 Bildschirmerkennung

Für die Automatisierung des Kalibrierungsvorgangs müssen zuerst die einzelnen Bildschirme in einem Bild erkannt werden. Hierfür wurde ein Verfahren von Martin Hirzer [8] zur Markererkennung implementiert, welches normalerweise für Augmented Reality Anwendungen verwendet wird. Es basiert auf der Augmented Reality Library ARTag und ist somit ein auf Kantendetektion basierendes Verfahren. Dies ermöglicht eine stabile und gute Erkennung der Marker auch unter verschiedenen sich ändernden Lichtbedingungen. Weiters können auch teilweise verdeckte Marker noch erkannt werden. Hierfür werden zuerst stichprobenartig Pixel auf einem Raster ausgewählt, um die Berechnungsdauer des Algorithmus zu reduzieren. Dies soll es ermöglichen Marker in Echtzeit zu erkennen. Aus den ausgewählten Pixeln werden jene ermittelt, welche Teil einer Kante sein könnten. Diese werden

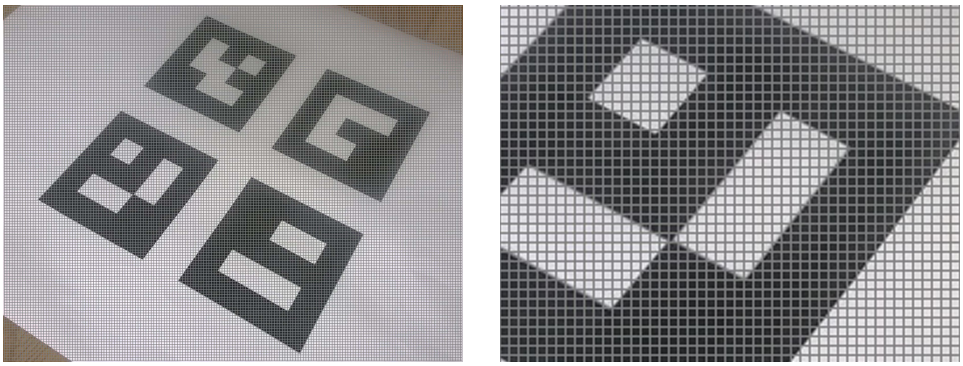


Abbildung 3.1: Es wurde für ein 5×5 Pixel großes Raster für die Erkennung der Kantenpunkte verwendet. Alle Pixel, die sich auf diesem Raster befinden werden überprüft.

anschließend zu Linien zusammengefügt. Aus den Linien werden dann Vierecke gebildet, welche als Marker verwendet werden können. Die so erkannten Vierecke werden zum Schluss noch in Bildschirme, ID-Marker und ungültige Vierecke eingeteilt. Im folgenden Abschnitt werden die einzelnen Schritte noch genauer beschrieben.

3.1.1 Kantendetektion

Für die Erkennung der Bildschirme in einem Bild wird eine Methode zur Augmented Reality Markererkennung von Martin Hirzer [8] verwendet. Die Grundlagen des Kantendetektionsalgorithmus stammen dabei von Clarke et al. [2].

Berechnung der Edgels

Um Kanten in einem Bild zu erkennen, werden zuerst alle Pixel ermittelt, die Teil einer Kante sein könnten. Diese Pixel werden auch als Edgels (kurz für *edge pixels*) bezeichnet. Um die Laufzeit des Algorithmus zu verbessern wird jedoch nicht das ganze Bild nach Edgels abgesucht, sondern nur jene entlang eines groben Rasters. Wie in Abbildung 3.1 zu sehen, besteht das hier verwendete Raster aus rechtwinklig und in gleichmäßigen Abständen von 5×5 Pixel angeordneten horizontalen und vertikalen Linien. Eine andere Anordnung des Rasters wäre ebenfalls möglich. Je nach Orientierung des Rasters könnten somit bestimmt orientierte Linien bewusst seltener oder häufiger erkannt werden. Das hier verwendete rechtwinklige horizontale und vertikale Raster bevorzugt dementsprechend auch horizontale und vertikale rechtwinklige Linien, da nur Pixel entlang des Rasters auf einen Kantenpunkt überprüft werden. Wie in Abbildung 3.2 ersichtlich könnten deshalb diagonale Linien bei der Kantenerkennung übersehen werden, falls das Ras-

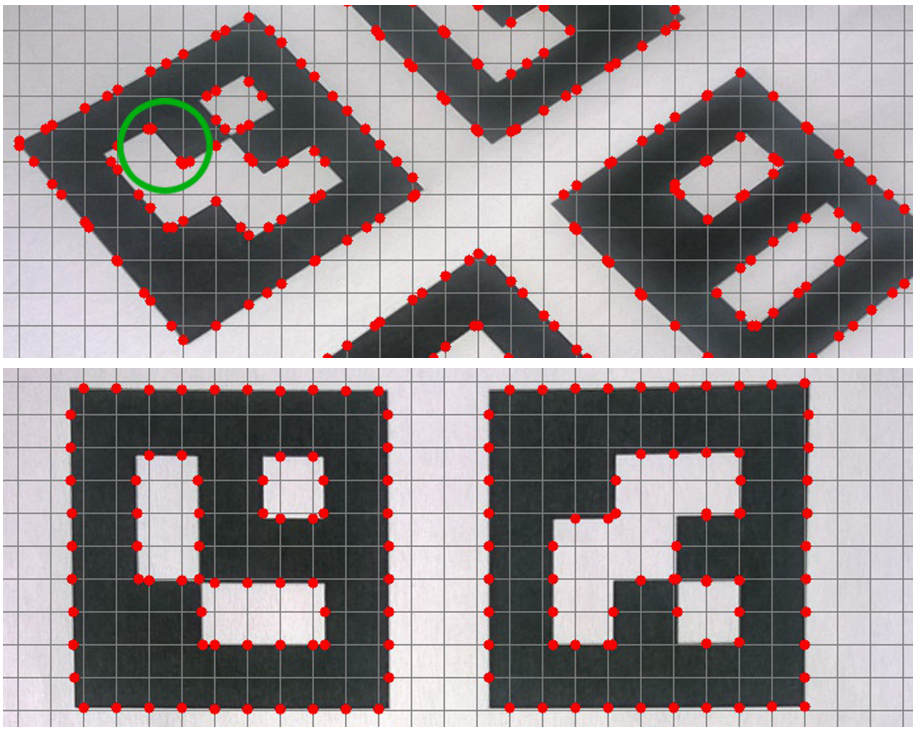


Abbildung 3.2: In diesem Beispiel wurde ein 25×25 Pixel großes Raster für die Berechnung der Edgels verwendet. Stimmt die Orientierung der Linien im Bild nicht mit der Orientierung des Rasters überein, so werden Edgels nur in unregelmäßigen Abständen erkannt. Dabei besteht die Möglichkeit, dass ganze Linien übersehen werden wenn das Raster zu grob gewählt wurde, wie an der markierten Stellen ersichtlich ist. Im unteren Beispiel tritt dieses Problem nicht auf, da das Raster und die Linien eine ähnliche Orientierung haben.

ter zu grob gewählt wurde. Dieses leicht anisotrope Verhalten ist zwar ein Nachteil des verwendeten Algorithmus, tritt jedoch hier nur in geringem Maße auf, da das Raster klein genug gewählt wurde. Die Größe des Rasters wurde nach den Ergebnissen in Abschnitt 5.1 gewählt.

Nachdem ein passendes Raster ausgewählt wurde, wird für jedes Pixel entlang diesem überprüft, ob es sich um einen Kantenpunkt handelt. Dazu wird auf alle Rasterpixel eine Faltungsoperation mit einer eindimensionalen Abwandlung der Gauß'schen Funktion angewandt um dessen Intensitätsgradienten zu berechnen. Der Gradient gibt an, ob es sich bei einem Pixel um ein Edgel handelt oder nicht. Er ist ein Vergleichswert zwischen den Nachbarpixeln des aktuellen Rasterpixels. Je niedriger der Wert ist, desto weniger Unterschied gibt es zwischen zwei Nachbarseiten, wie in den Beispielen in Abbildung 3.3 zu sehen ist. Ein Wert über einem bestimmten Grenz-

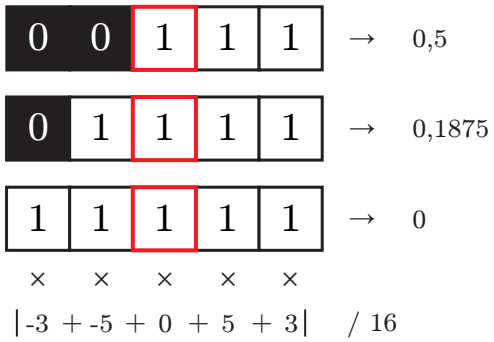


Abbildung 3.3: Drei vereinfachte Beispiele für die Berechnung des Intensitätsgradienten mit schwarzen und weißen Pixeln. Für das aktuelle Pixel (rot markiert) auf einer vertikalen Rasterlinie wird dessen Intensitätsgradient mit dem Faltungskern $0{,}0625 \cdot \begin{bmatrix} -3 & -5 & 0 & 5 & 3 \end{bmatrix}$ ermittelt. Der größtmögliche Unterschied zwischen den beiden Nachbarseiten ergibt einen Maximalwert von 0,5. Sind die Farbwerte der Pixel auf beiden Seiten gleich, so ist der Intensitätsgradient 0.

wert (hier 0,1) definiert also einen Edgel. Für diese Berechnung werden, je nachdem ob es sich um eine horizontale oder vertikale Rasterlinie handelt, jeweils die darauf rechtwinklig stehenden Nachbarpixel in Richtung der Y- oder X-Achse verwendet. Wie viel Einfluss welche Nachbarpixel auf den Intensitätsgradienten haben wird vom verwendeten Faltungskern bestimmt. Hier wurden

$$h^x = 0{,}0625 \cdot \begin{bmatrix} -3 & -5 & 0 & 5 & 3 \end{bmatrix}, \tag{3.1}$$

$$h^y = 0{,}0625 \cdot \begin{bmatrix} -3 \\ -5 \\ 0 \\ 5 \\ 3 \end{bmatrix} \tag{3.2}$$

verwendet, wobei 0 die Gewichtung des aktuellen Pixels ist. Pixel direkt neben dem aktuellen Pixel haben also mehr Einfluss darauf ob es sich beim aktuellen Pixel um eine Kante handelt als weiter entfernte Pixel. Dadurch ergeben sich die Gleichungen

$$f(x,y) = |\sum_{i=0}^{l-1} h_i^x \cdot I_c(x + i - \lfloor \frac{l}{2} \rfloor, y)|, \tag{3.3}$$

$$g(x,y) = |\sum_{i=0}^{l-1} h_i^y \cdot I_c(x, y + i - \lfloor \frac{l}{2} \rfloor)| \tag{3.4}$$

zur Berechnung des Intensitätsgradienten für einen Farbkanal des Pixels an der Position (x, y), wobei Gleichung 3.3 zur Berechnung in horizontaler Richtung und Gleichung 3.4 zur Berechnung in vertikaler Richtung verwendet wird. l beschreibt die Größe des verwendeten Faltungskerns. I_c ist die Bildmatrix des Farbkanals c.

Weiters muss für die Berechnung des Intensitätsgradienten festgelegt werden, welche Farbkanäle der Pixel verwendet werden sollen. Dies muss vor allem auf die Farbe der Marker und deren Hintergrundfarbe abgestimmt werden. Zwei Möglichkeiten wurden für dieses Projekt für das Finden von schwarzen Markern auf weißem Hintergrund in Betracht gezogen. Eine Möglichkeit wäre, für jeden Farbkanal einen eigenen Intensitätsgradienten zu berechnen. Für jeden einzelnen Gradienten muss dann überprüft werden, ob dieser über dem festgelegten Grenzwert liegt, um das Finden farbiger Marker möglichst zu unterbinden. Diese Methode wurde auch von Martin Hirzer in [8] verwendet. Die zweite Möglichkeit wäre, das zu verarbeitende Bild zuerst in ein Graustufenbild umzurechnen. Dadurch gäbe es nur mehr einen Farbkanal, für den ein Intensitätsgradient berechnet werden muss. Wie in Abschnitt 5.1 zu sehen ist, ergaben Messungen, dass dadurch eine Laufzeitverbesserung von bis zu 30% bei der Edgelberechnung möglich ist. Ein Nachteil dieser Methode ist jedoch, dass bei der Umrechnung in ein Graustufenbild die Information über die einzelnen Farbkanäle verloren geht. Wie in Abbildung 3.4 ersichtlich, ist es deshalb nicht mehr möglich zwischen schwarz-weißen und farbigen Kanten zu unterscheiden, was zu einer größeren Anzahl an gefundenen irrelevanten Edgels führt. Dies verlangsamt die späteren Schritte des Algorithmus und führt insgesamt zu einer Verschlechterung der Performance, was ebenfalls in Abschnitt 5.1 genauer erklärt ist.

Nachdem ein Pixel als Edgel identifiziert wurde, muss noch eine letzte Überprüfung durchgeführt werden. Da, wie in Abbildung 3.5 ersichtlich, an einer Kante im Bild meistens mehrere nebeneinander liegende Pixel als Edgel erkannt werden, müssen hier noch die weniger relevanten Kantenpunkte aussortiert werden. Dafür wird der Intensitätsgradient des aktuellen Pixels mit dem jeweils vorherigen und nächsten Nachbarpixel verglichen. Hat das aktuelle Pixel den höchsten Intensitätsgradienten der drei, so wird es zu einer Liste mit gefundenen Edgels hinzugefügt. Das endgültige Ergebnis der Edgelberechnung ist in Abbildung 3.6 zu sehen.

Kanten aus Edgels berechnen

Nachdem alle Edgels in einem Bild berechnet wurden, müssen diese zu Linien zusammengefügt werden. Da es viel Rechenzeit benötigen würde, für jedes Edgel mit jedem anderen Edgel im Bild zu überprüfen, ob diese eine Linie bilden, wird dieser Schritt zuerst auf kleinere Regionen im Bild angewandt. Dies ist insofern sinnvoll, da sich Edgels, die eine Linie bilden, mit hoher Wahrscheinlichkeit nah beieinander befinden. Hierfür wurden 40×40 Pixel

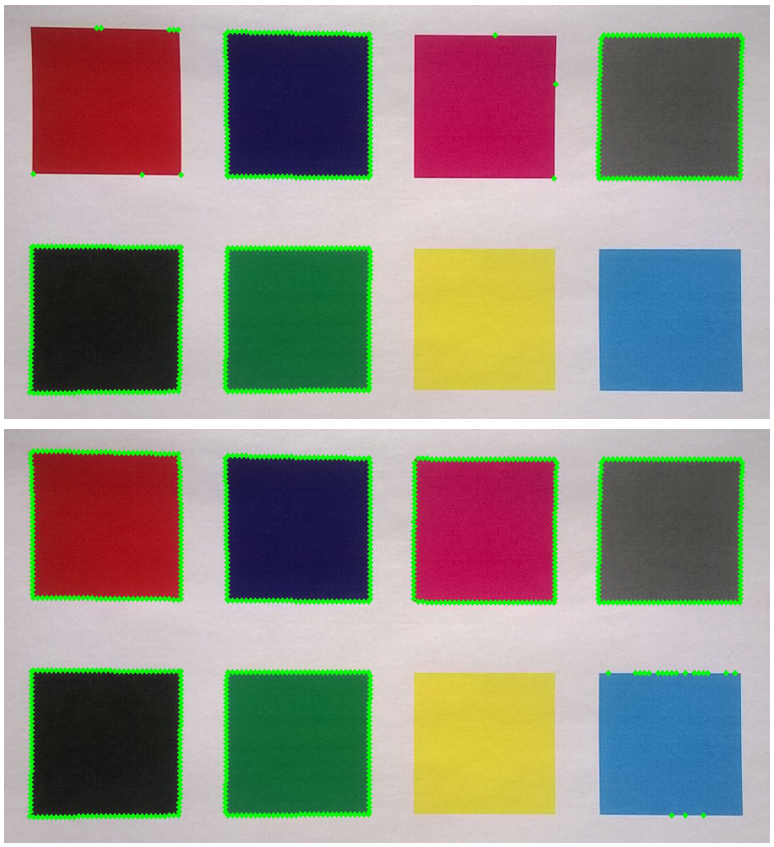


Abbildung 3.4: Im oberen Bild wurde bei der Edgelberechnung der Intensitätsgradient für jeden Farbkanal einzeln berechnet. Im unteren Beispiel wurde das Bild vor der Edgelberechnung in ein Graustufenbild umgewandelt. Daher kann nicht mehr gut zwischen farbigen und schwarz-weißen Kanten unterschieden werden.

große Regionen gewählt.

In jeder einzelnen Region des Bildes müssen nun Edgels gefunden werden, welche sich auf einer Linie befinden. Hierfür wurde der RANSAC-Algorithmus (***ran**dom **sa**mple **c**onsensus*) [6] verwendet. Dabei werden zuerst zufällig zwei der Edgels in einem Bereich ausgewählt, deren Steigung übereinstimmt. Die Steigung eines Pixels kann mithilfe des Sobel-Algorithmus berechnet werden. Bei diesem wird auf einen 3×3 Pixel großen Bild-

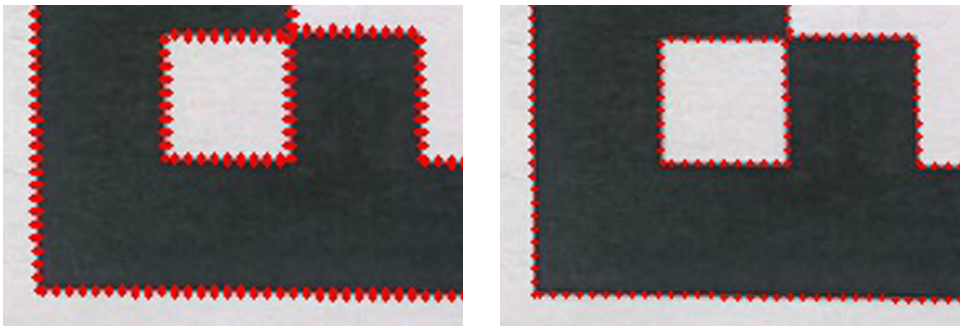


Abbildung 3.5: Im linken Bild wurde nicht überprüft, ob ein Pixel der Kantenpunkt mit dem höchsten Intensitätsgradient in seiner Umgebung ist, daher wurden einfach alle in der Nähe einer Kante zur Liste an Edgels hinzugefügt. Rechts wurden nur die Punkte mit der höchsten Relevanz, also diejenigen, die sich am nächsten zur Kante befinden, verwendet.

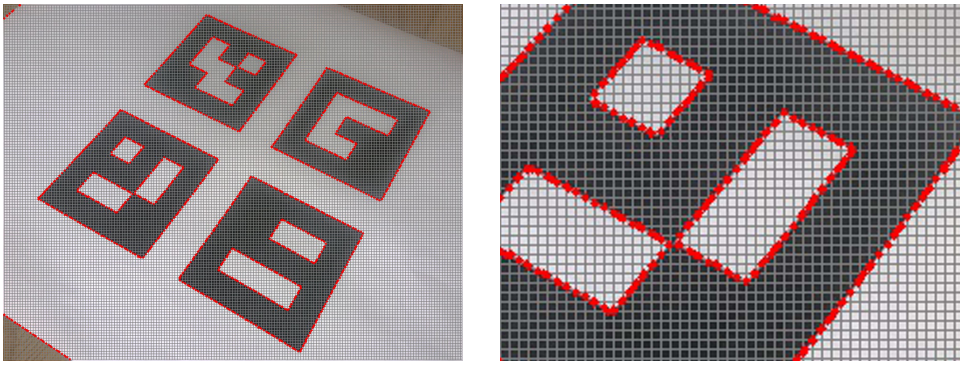


Abbildung 3.6: Das Ergebnis der Edgelberechnung.

ausschnitt eine Faltung mittels der Sobeloperatoren

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \tag{3.5}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.6}$$

angewandt. Die Herleitung dieser beiden Operatoren ist in [14] genauer erklärt. Das Ergebnis der Faltung ist ein Vektor, dessen Komponenten Intensitätsgradienten in Richtung der X- und Y-Achse sind. Die Normale dieses Vektors ist die Steigung des mittleren Pixels. Die beiden Sobeloperatoren geben an, wie viel Einfluss die umliegenden acht Nachbarpixel auf diese

p_3	p_2	p_1
p_4	X	p_0
p_5	p_6	p_7

Abbildung 3.7: Für das mit X markierte Pixel werden seine acht Nachbarpixel als Parameter p_0 bis p_7 für die Gleichungen 3.7 und 3.8 zur Berechnung seiner Steigung definiert.

Berechnung haben, wobei Gleichung 3.6 zur Ermittlung des X-Wertes und Gleichung 3.5 zur Ermittlung des Y-Wertes der Steigung verwendet wird. Dadurch ergeben sich die Gleichungen

$$k_x = p_7 - p_3 + 2 \cdot (p_0 - p_4) + p_1 - p_5, \tag{3.7}$$
$$k_y = p_3 - p_7 + 2 \cdot (p_2 - p_6) + p_1 - p_5 \tag{3.8}$$

zur Ermittlung der X- und Y-Komponenten des Steigungsvektors k für das aktuelle Pixel. p_0 bis p_7 sind wie in Abbildung 3.7 zu sehen zugeordnet und beschreiben dessen Nachbarpixel in der Bildmatrix I wie folgt:

$$p_j(x,y) = \begin{cases} I(x+1,y) & \text{für } j = 0, \\ I(x+1,y-1) & \text{für } j = 1, \\ I(x,y-1) & \text{für } j = 2, \\ I(x-1,y-1) & \text{für } j = 3, \\ I(x-1,y) & \text{für } j = 4, \\ I(x-1,y+1) & \text{für } j = 5, \\ I(x,y+1) & \text{für } j = 6, \\ I(x+1,y+1) & \text{für } j = 7. \end{cases} \tag{3.9}$$

Nachdem zwei Edgels mit passender Steigung gefunden wurden, wird für alle restlichen Edgels im Bereich überprüft, ob diese auf der Verbindungslinie zwischen den beiden liegen. Dies ist der Fall, wenn das zu überprüfende Edgel ebenfalls die gleiche Orientierung hat und innerhalb einer bestimmten Distanz zur Linie liegt, wie in Abbildung 3.8 dargestellt ist. Je nachdem wie viele Edgels die potenzielle Linie unterstützen, ist es wahrscheinlicher, dass es sich um eine echte Kante im Bild handelt. Jede potenzielle Kante muss auch eine bestimmte Mindestanzahl an unterstützenden Edgels besitzen, um als Kante zu gelten. Dieser Vorgang wird einige Male mit immer neu zufällig ausgewählten Edgels wiederholt. Die Linie, welche am Ende die meistens unterstützenden Edgels besitzt, wird einer Liste an gefundenen Kanten hinzugefügt. Die zwei Ausgangspunkte dieser Kante und alle ihre unterstützenden Edgels werden anschließend von der Gruppe an Edgels im

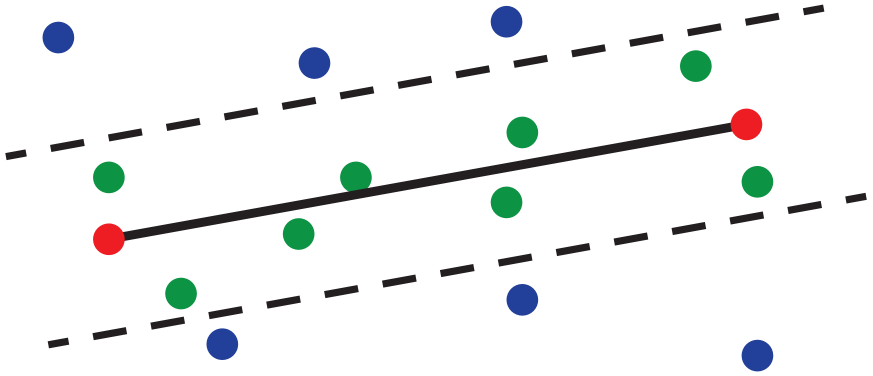


Abbildung 3.8: Vereinfachte Darstellung des RANSAC-Algorithmus. Die zwei roten Edgels wurden zufällig ausgewählt und es wird überprüft ob diese eine Kante im Bild darstellen. Alle Edgels innerhalb eines bestimmten Abstands mit gleicher Orientierung wie die potenzielle Linie (hier in grün dargestellt) unterstützen diese Kante.

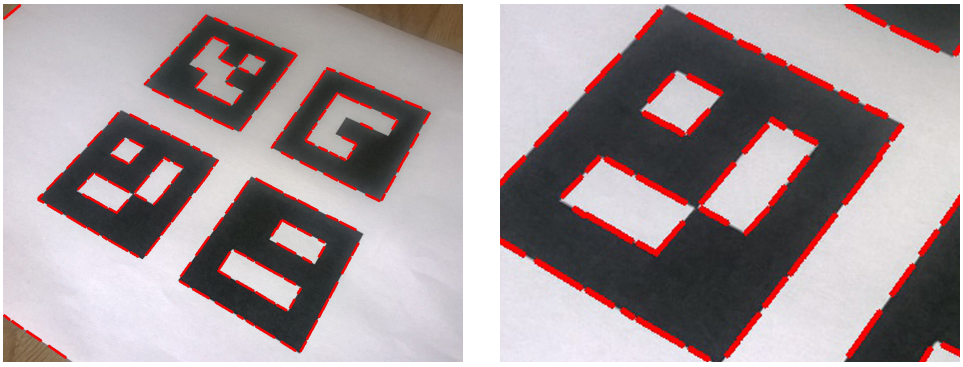
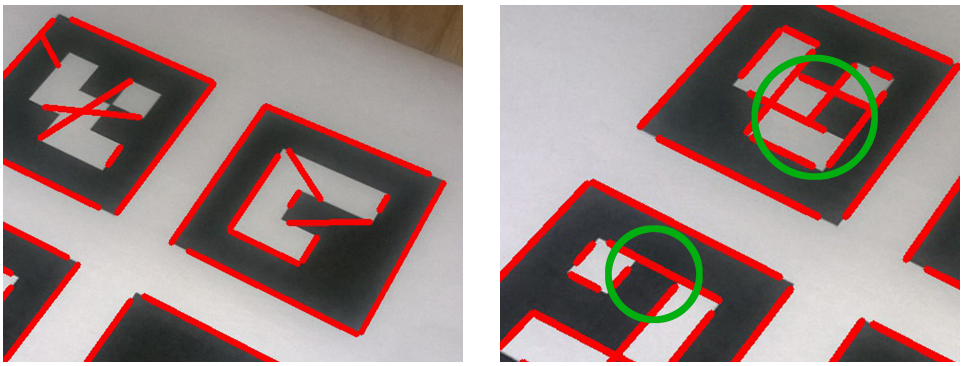


Abbildung 3.9: Mithilfe des RANSAC-Algorithmus wurden die gefundenen Edgels zu Kanten zusammengefügt.

Bereich entfernt und der gesamte Prozess wird erneut mit weniger Edgels wiederholt, um noch mehr potenzielle Linien im selben Bereich zu finden. Der gesamte Vorgang wird solange wiederholt, bis entweder nur noch eine kleine Anzahl an Edgels im Bereich übrig ist oder eine maximale Anzahl an Durchläufen ausgeführt wurde. Das Ergebnis des RANSAC-Algorithmus ist in Abbildung 3.9 zu sehen.

Kanten zusammenfügen

Für den nächsten Schritt müssen die gefundenen Kantenstücke zu längeren Kanten zusammengefügt werden. Dieser Schritt wird zunächst wieder nur



(a) (b)

Abbildung 3.10: Verschiedene Fehler bei der Verbindung von Kanten. Im ersten Beispiel (a) wurden Linien verbunden, deren Steigung zwar übereinstimmt, aber nicht die deren Verbindungslinie. Im zweiten Bild (b) wurde nicht jedes Pixel zwischen zwei Kanten auf eine ähnliche Steigung wie die der Kanten überprüft, sondern nur ob ein bestimmter Mindestabstand zwischen den zwei Kanten liegt. Dadurch werden auch falsche Kanten verbunden wenn sie zu nah beieinander liegen.

in kleineren Bereichen des Bildes durchgeführt, um die Performance zu verbessern. Sobald in einem Bereich im letzten Schritt mindestens zwei Kanten gefunden wurden, werden alle Kanten darin miteinander verglichen. Zuerst wird überprüft ob die Steigung einer Kante deren der anderen ähnelt. Alle Kanten, bei denen dies zutrifft, werden in einer Liste als potenzielle Kandidaten zum Zusammenfügen gespeichert. Wurden alle geprüft und mindestens zwei passende Linien gefunden, so wird die Liste aufsteigend nach Abstand zur Ausgangslinie geordnet. Dies stellt sicher, dass die nächsten Berechnungen zuerst für die Linien mit kürzerer Distanz zur Ausgangslinie durchgeführt wird. Im nächsten Schritt wird überprüft, ob auch die Steigung der Verbindungslinie zwischen zwei Kanten mit denen der Kanten selbst übereinstimmt. Diese Überprüfung stellt sicher, dass gleich orientierte Kanten, die nicht nebeneinander liegen, nicht verbunden werden, wie in Abbildung 3.10(a) zu sehen ist. Stimmt die Steigung der Verbindungslinie mit deren der Kanten überein, so wird auch noch die Orientierung jedes einzelnen Pixels zwischen den beiden Kanten überprüft, um falsche Verbindungen wie in Abbildung 3.10(b) dargestellt zu vermeiden. Bei diesem Berechnungsvorgang macht sich der Vorteil der Sortierung nach Distanz bemerkbar. Wie in Abbildung 3.11 zu sehen ist, reduziert diese Sortierung die Anzahl an Pixel, die auf ihre Steigung überprüft werden müssen.

Wenn eine Linie sämtliche der Kriterien erfüllt, so wird die längste Verbindungslinie zwischen den vier möglichen Eckpunkten berechnet. Diese neue längere Linie wird anschließend anstatt der zwei Ursprungslinien wei-

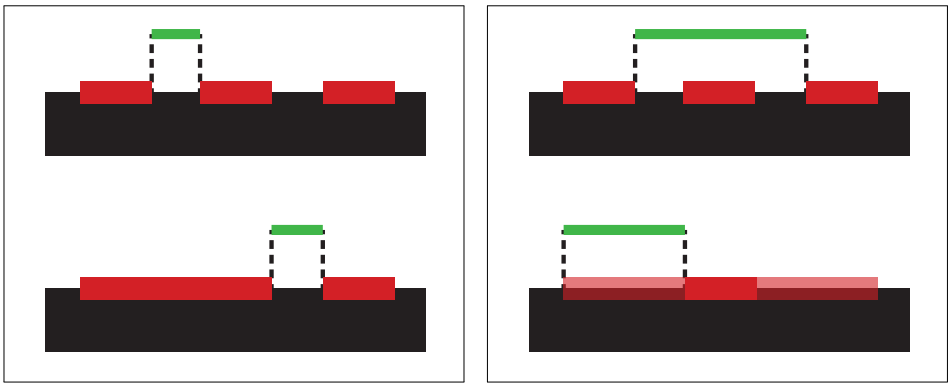


Abbildung 3.11: Gefundene Kantenteile (rot) werden ausgehend vom Kantenteil links zusammengefügt. Im linken Bild wurden die restlichen Kantenteile nach Distanz zum linken Kantenteil geordnet und in dieser Reihenfolge weiter überprüft. Daher muss im nächsten Schritt nur die Steigung der fehlenden Pixel (grün) überprüft werden. Im Beispiel rechts wurden die Kantenteile nicht geordnet und die Überprüfung wird zuerst bei den zwei äußersten Kantenteilen durchgeführt, daher werden manche Pixel unnötig oder sogar mehrmals auf ihre Steigung überprüft.

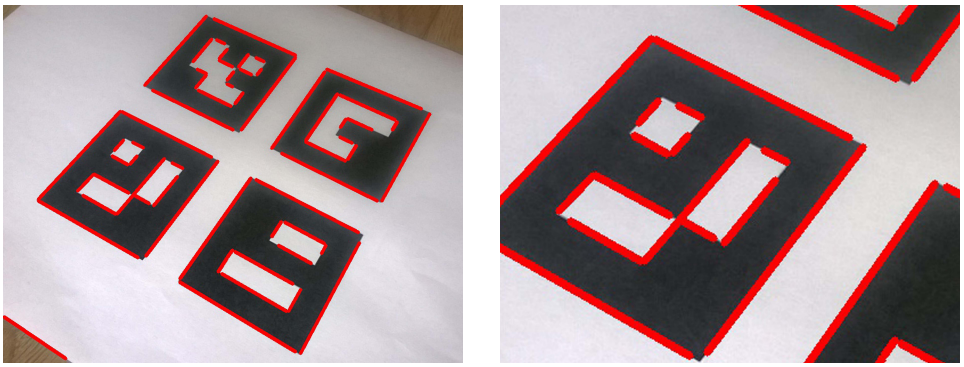


Abbildung 3.12: Alle passenden Kantenstücke wurden zu langen Kanten zusammengefügt.

terverwendet. Der gesamte Vorgang wird so lange auf einen Bereich des Bildes angewandt, bis keine passenden Linien mehr vorhanden sind. Nachdem so alle Bereiche des Bildes überprüft worden sind, wird der gesamte Vorgang noch ein letztes Mal auf alle Kanten im kompletten Bild angewandt. Das Ergebnis dieses Schrittes ist in Abbildung 3.12 zu sehen.

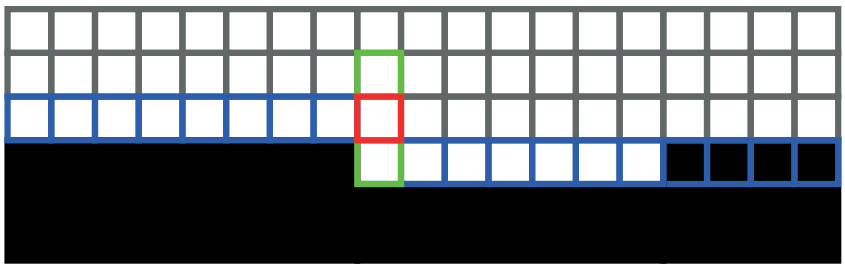


Abbildung 3.13: Bei der Fortführung einer Kante wird Pixel für Pixel überprüft, ob dieser zur Kante hinzugefügt werden kann. Erfüllt ein Pixel die Kriterien nicht (rot), so werden auch die dazu rechtwinklig stehenden Pixel (grün) überprüft. Das untere grün markierte Pixel wurde als Kantenpunkt identifiziert und somit wird die Kantenerweiterung bei ihm fortgesetzt um weitere passende Pixel (blau) zu finden.

Kanten erweitern

Da nach dem Zusammenfügen der Linien die Kanten im Bild nicht komplett erkannt wurden, sondern teilweise kleine Stücke fehlen, wie in Abbildung 3.12 zu sehen ist, müssen alle Linien noch bis zum tatsächlichen Endpunkt der Kante verlängert werden. Dafür wird von den beiden Endpunkten einer Linie ausgehend in Richtung ihrer Steigung und entgegengesetzter Steigung Pixel für Pixel überprüft, ob dieser die gleiche Orientierung wie die Linie selbst hat. Die Steigung eines Pixels kann mit den bereits erwähnten Gleichungen 3.7 und 3.8 berechnet werden. Dies wird so lange wiederholt, bis die Steigung eines Pixels nicht mehr mit der der Kante übereinstimmt. Das zuletzt überprüfte Pixel ist aber nicht immer der Endpunkt der Kante. Aufgrund von Verzerrungen im Bild sind Kanten nicht immer gerade, sondern manchmal leicht gekrümmt. Um auch diese Kanten möglichst gut erkennen zu können, wird die Überprüfung, wie in Abbildung 3.13 dargestellt, bei den beiden im rechten Winkel stehenden Pixeln fortgeführt, nachdem ein unpassendes Pixel gefunden wurde. So können leichte Verzerrungen im Bild bei der Kantenfindung gehandhabt werden, wodurch der Algorithmus robuster wird. Das Ergebnis dieses Schrittes ist in Abbildung 3.14 zu sehen.

3.1.2 Erkennung von Vierecken

Nachdem alle Kanten im Bild erkannt wurden, müssen passende zu Vierecken gruppiert und alle restlichen aussortiert werden. Hierfür wird für eine Linie nach der anderen von einem ihrer Endpunkte ausgehend mit allen restlichen Linien überprüft, ob diese das Eck eines Vierecks bilden. Dafür müssen zwei Kriterien erfüllt werden. Einerseits müssen die Endpunkte zweier Linien nah beieinander liegen. Hierfür werden zuerst alle zu prüfenden Li-

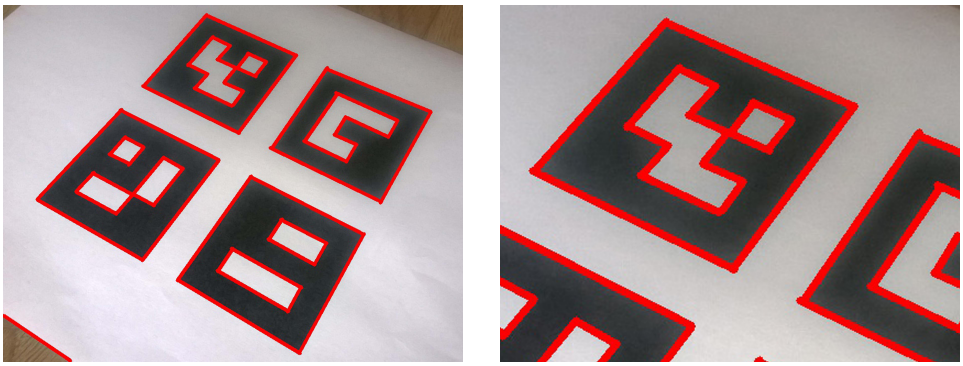


Abbildung 3.14: Alle gefundenen Linien wurden bis zum Endpunkt der Kanten im Bild erweitert.

nien nach deren Distanz zum zuvor gewählten Endpunkt der Ausgangslinie aufsteigend geordnet. Außerdem werden alle Linien, deren Endpunkte nicht nah genug beim gewählten Endpunkt der Ausgangslinie liegen, aussortiert. Der zweite Überprüfungsschritt wird dann auf die restlichen nach Distanz geordneten Linien ausgeführt. Die Reihung nach Distanz soll verhindern, dass zuerst Linien überprüft werden, welche zwar auch innerhalb der gültigen Distanz liegen, jedoch nicht die eigentliche Kante des Vierecks sind. Dies ist zum Beispiel bei der Erkennung von dünnen Bildschirmrändern der Fall, wie in Abbildung 3.15 zu sehen ist, bei denen Außen- und Innenrand der erkannten Bildschirme vertauscht wurden. Für alle Linien muss noch deren Orientierung überprüft werden, da diese nicht parallel zur Ausgangslinie liegen dürfen. Aufgrund dieser Überprüfung werden zwar extrem verzerrte Vierecke im Bild nicht erkannt, diese könnten jedoch für die weiteren Berechnungsschritte sowieso nicht verwendet werden, da deren Inhalt bei zu starker Verzerrung nicht mehr auswertbar ist.

Erfüllen zwei Linien diese beiden Kriterien, so bilden sie eine Ecke, und es wird von der gefundenen passenden Linie aus weiter gesucht. Dieser Vorgang wird so lange rekursiv wiederholt, bis 4 zusammenhängende Kanten gefunden wurden. Schlägt die Überprüfung bei einer Kante bei allen anderen Linien fehl, so wird der Vorgang bei der Ausgangslinie in Richtung des zweiten Endpunktes fortgeführt.

Manchmal kann es vorkommen, dass Marker oder Bildschirme im Bild leicht verdeckt wurden. Mit obiger Methode werden diese jedoch immer noch erkannt, wenn nur maximal eine Ecke und zwei Kanten zum Teil verdeckt sind, wie in Abbildung 3.16 zu sehen ist. Da jedoch die Eckpunkte der Kanten dann nicht den wahren Eckpunkten der Vierecke entsprechen, müssen diese nach Erkennung aller Vierecke im Bild noch neu berechnet werden. Hierfür wird für alle vier Eckpunkte der Schnittpunkt zweier aneinander folgender Linien berechnet und gespeichert. Das Ergebnis dieses

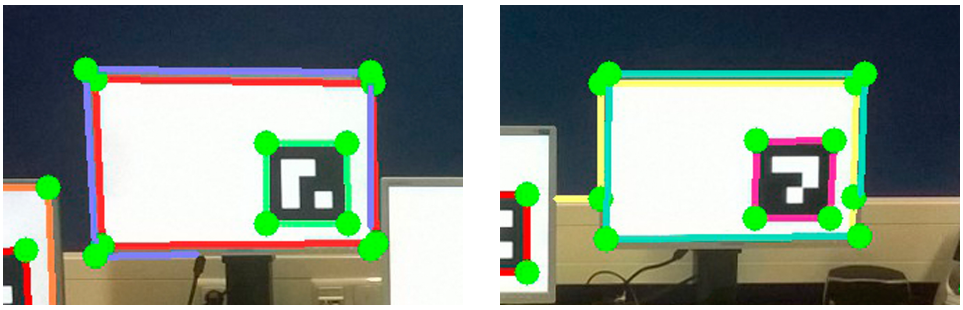


Abbildung 3.15: Fehler bei der Erkennung von Bildschirmen. Beim Verbinden der einzelnen Linien zu Vierecken wurden Linien verbunden, welche zwar den Kriterien einer Ecke entsprechen, jedoch nicht die richtige bilden. Deshalb wurden die Kanten des inneren und äußeren Bildschirmrandes verwechselt. Dies kann verhindert werden, indem zuerst die Kanten mit geringster Entfernung miteinander überprüft werden.

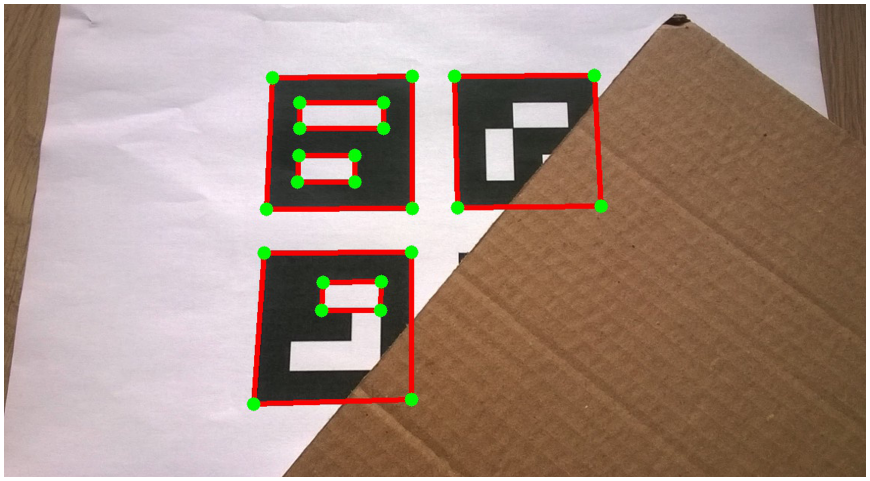


Abbildung 3.16: Auch wenn ein Teil der Marker verdeckt war, können diese immer noch erkannt werden, wenn maximal ein Eckpunkt und zwei Kanten zum Teil verdeckt sind.

Berechnungsschrittes ist in Abbildung 3.17 ersichtlich. Alle Vierecke, die so im Bild gefunden wurden, könnten nun Bildschirme, Marker oder andere irrelevante Objekte sein und müssen im nächsten Schritt noch sortiert werden.

3.1.3 Unterscheidung in Bildschirme und Marker

Nachdem alle Vierecke und somit mögliche Marker oder Bildschirme in einem Bild erkannt wurden, müssen diese noch voneinander unterschieden und

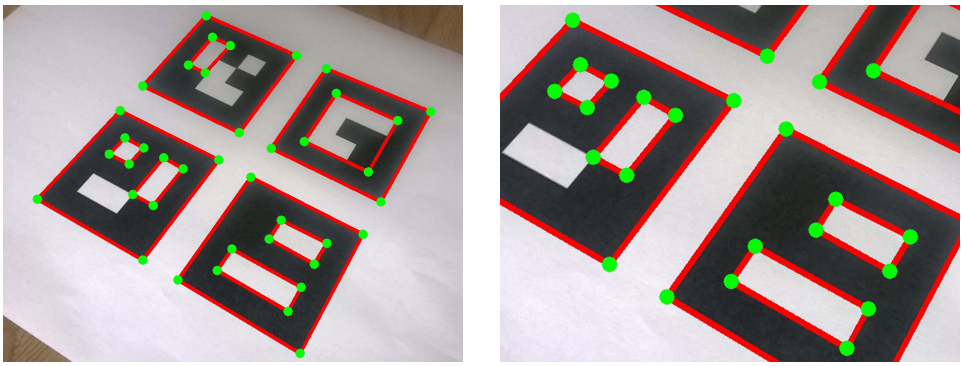


Abbildung 3.17: Aus den gefundenen Linien im Bild wurden Vierecke gebildet.

irrelevante Vierecke aussortiert werden. Dazu werden zuerst die Eckpunkte aller gefundenen Vierecke im Uhrzeigersinn rund um deren Mittelpunkt geordnet. Dies hilft den nächsten Abschnitt, das Erkennen der Marker-IDs, zu vereinfachen. Außerdem erleichtert es die nächsten Schritte für die Positions- und Rotationsberechnung der Bildschirme. Wenn deren Eckpunkte bereits einheitlich geordnet angegeben sind, muss die Ordnung der Eckpunkte bei allen späteren Berechnungsschritten nicht mehr vorher überprüft werden.

Für das Ordnen der Eckpunkte eines Vierecks im Uhrzeigersinn wird vorausgesetzt, dass diese bereits in richtiger Reihenfolge, entweder im oder gegen den Uhrzeigersinn, angegeben sind. Dies wird durch den vorherigen Schritt der Markererkennung, das Verbinden aller gefundenen Linien zu Vierecken, sichergestellt, da die Linien reihum zu einem Viereck zusammengefügt werden. Für die Eckpunkte eines solchen Vierecks kann dann überprüft werden, ob diese bereits im Uhrzeigersinn angeordnet sind oder nicht, indem zuerst ihr Mittelpunkt berechnet wird. Danach werden Vektoren von zwei aufeinanderfolgenden Eckpunkten des Vierecks zu diesem Mittelpunkt gebildet. Von diesen beiden Vektoren wird dann das Kreuzprodukt berechnet. Das Ergebnis dieser Berechnung ist ein Skalar, dessen Vorzeichen die Ordnung der Eckpunkte bestimmt. Ist das Ergebnis positiv, so sind die Punkte schon im Uhrzeigersinn angeordnet, ist es negativ, so waren die Punkte gegen den Uhrzeigersinn geordnet und müssen umgedreht werden.

Nachdem die Eckpunkte aller Vierecke im Uhrzeigersinn geordnet wurden, wird überprüft, welche Vierecke innerhalb oder außerhalb anderer Vierecke liegen. Da Bildschirm-Marker außerhalb von ID-Markern liegen, können diese im späteren Schritt dadurch identifiziert werden. Es wird für jedes Viereck mit allen anderen überprüft, ob dessen Eckpunkte darin oder außerhalb liegen. Nachdem jedes Viereck die Vierecke gespeichert hat, welche innerhalb von ihm liegen und welche es umschließen, wird für alle überprüft,

Abbildung 3.18: Ein verzerrter Marker im aufgenommenen Bild und nachdem er mithilfe der Homographie-Matrix entzerrt und in ein 150×150 Pixel großes Bild gezeichnet wurde.

ob ihr Bildbereich das Muster eines ID-Markers beinhaltet. Hierzu muss ein Marker zuerst entzerrt werden. Alle Verzerrungen, die bei der Aufnahme eines Bildes entstehen, können mit der sogenannten *Homographie-Matrix* entfernt werden. Diese kann berechnet werden, wenn mindestens vier koplanare Punkte des Objektes im Bild und die entsprechenden Koordinaten des echten Objektes bekannt sind. In [10] ist dies näher beschrieben. In diesem Fall werden die Koordinaten der vier Eckpunkte eines Vierecks im Bild den vier Eckpunkten eines Quadrats zugewiesen. Mit der Homographie-Matrix kann dann jeder Punkt des entzerrten Markers aus dem verzerrten Bild berechnet werden. Da nicht bekannt ist, welches Eck des gefundenen Vierecks welchem Eck des eigentlichen Markers entspricht, also zum Beispiel, welches Eck das linke obere des Markermusters ist, kann ein Marker beim Zuweisen der Eckpunkte von den verzerrten zu den entzerrten rotiert werden. Durch die vorherige Ordnung der Eckpunkte im Uhrzeigersinn kann sichergestellt werden, dass das resultierende Marker-Bild nicht gespiegelt ist, was dazu führt, dass dies beim späteren Erkennen des ID-Musters nicht mehr beachtet werden muss. In Abbildung 3.18 ist ein Beispiel für das Entzerren eines stärker verzerrten Markers zu sehen.

Nachdem ein möglicher Marker im Bild entzerrt wurde, muss überprüft werden, ob dieser ein gültiges ID-Muster enthält, wodurch sichergestellt wird, dass dieser tatsächlich ein Marker ist. Hierfür wird für jedes Feld im Marker mittels *Thresholding* festgestellt, ob dieses schwarz oder weiß ist. Für jedes Pixel im Marker-Bild wird überprüft ob dieses über einem bestimmten Grenzwert liegt. Ist dies der Fall, so wird es als weiß markiert, ansonsten als schwarz. Wichtig für diese Überprüfung ist, einen geeigneten Grenzwert zu finden. Ein fix gesetzter Grenzwert funktioniert nur in bestimmten Situationen und führt zum Beispiel bei unterschiedlichen Lichtverhältnissen

(a) (b) (c)

Abbildung 3.19: Für einen gefundenen entzerrten Marker (a) wird seine ID mittels Thresholding ermittelt. Wird der Grenzwert fix auf zum Beispiel den Mittelwert 128 gesetzt, so kann es bei manchen Lichtverhältnissen zu falschen Ergebnissen kommen (b). Wird der Grenzwert jedoch jedes Mal abhängig vom Ausgangsbild neu gewählt, so kann die ID eines Markers korrekt bestimmt werden (c).

zu falschen Ergebnissen, wie in Abbildung 3.19(b) zu sehen ist. Um dies zu verhindern wurde die Methode von Otsu [12] verwendet, welche für ein Grauwertbild je nach Intensitätsverteilung zweier Farbklassen den idealen Grenzwert liefert. Hierfür muss zuerst ein Grauwert-Histogramm vom Marker-Bild erstellt werden. Der genaue Vorgang mitsamt der Herleitung der Methode von Otsu ist in [12] genau erklärt.

Nachdem ein Markerbild mittels Thresholding in schwarze und weiße Pixel unterteilt wurde, muss noch herausgefunden werden, ob es ein valides ID-Muster enthält. Je nach gewählter Größe des Musters (hier 3×3) wird das Bild in gleich große quadratische Bereiche unterteilt (hier 5×5 30 Pixel große Bereiche für Muster und Rand). Für jeden der Bereiche wird anschließend überprüft, ob dieser mehr schwarze oder weiße Pixel beinhaltet. Dementsprechend wird dann auch die gefundene ID-Matrix festgelegt.

Nachdem das innere Bild eines gefundenen Vierecks in eine ID-Matrix umgewandelt wurde, muss noch überprüft werden, ob es sich dabei auch um ein echtes ID-Muster handelt. Als erstes wird dazu überprüft, ob die gefundene Matrix den schwarzen Rand eines Markers enthält. Ist dies der Fall, wird das innere Muster mit allen möglichen gültigen Mustern verglichen. Dabei muss darauf geachtet werden, dass das gefundene Muster auch die rotierte Version eines echten Musters sein könnte. Wie schon erwähnt, muss es jedoch aufgrund der vorherigen Sortierung der Eckpunkte im Uhrzeigersinn, nicht mehr gespiegelt werden. Durch die Anzahl an Rotationen, die durchgeführt werden müssen, bis ein gefundener Marker mit dem festgelegten Muster übereinstimmt, kann festgestellt werden, welches Eck des gefundenen Markers zum Beispiel dem linken oberen Eck des echten Markermusters entspricht. Wurde bei dieser Überprüfung keine passende ID gefunden, so

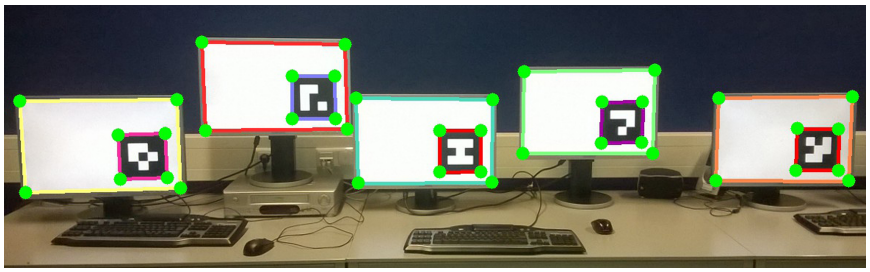


Abbildung 3.20: Das entgültige Ergebnis der Maker- und Bildschirmerkennung.

wird der Schritt mit den restlichen Vierecken fortgeführt. Wurde jedoch eine Übereinstimmung gefunden, so wird nach dem zugehörigen Bildschirm gesucht. Dieser muss ein Viereck sein, welches den gefundenen Marker beinhaltet. Da jedoch andere viereckige Objekte im Bild auch erkannt werden könnten, kann dies bei mehreren Vierecken der Fall sein. Von allen möglichen Vierecken kann jedoch nur jenes einen Bildschirm markieren, welches selbst die wenigsten anderen Vierecke umschließt. Nachdem so alle Bildschirme und Marker im Bild identifiziert wurden, wie es in Abbildung 3.20 zu sehen ist, können alle restlichen gefundenen Vierecke für die weiteren Schritte ignoriert werden.

3.2 Berechnung der relativen Positions- und Rotationsdaten

Nachdem alle ID-Marker und die Bildschirme, welche sie beinhalten, in einem Bild erkannt wurden, kann mithilfe deren realen Abmessungen und gefundenen Positionen im Bild, deren reale relative Positions- und Rotationsdaten zueinander berechnet werden. Dafür müssen zuerst die intrinsische Kameramatrix und die Verzerrungskoeffizienten der verwendeten Kamera ermittelt werden, um anschließend die Homographiematrix berechnen zu können. Mit dieser kann jeder zweidimensionale Punkt im Bild seinem äquivalenten dreidimensionalen Punkt relativ zur Kamera zugewiesen und deren Ausrichtung ermittelt werden. Aus diesen Werten können anschließend die relativen Positions- und Rotationsdaten zwischen den einzelnen Bildschirmen berechnet werden.

3.2.1 Kamerakalibrierung

Bei der Berechnung der Positions- und Rotationsdaten relativ zur Kamera muss auch die Verzerrung im aufgenommenen Bild miteinberechnet werden. Diese hängt von der verwendeten Kamera ab. Um die Verzerrung auszu-

gleichen, werden, wie in [10] erwähnt, bestimmte Kameraparameter bei der Berechnung der Homographiematrix benötigt. Diese müssen für jede Kamera einzeln bestimmt werden. Für die Kalibrierung einer Kamera gibt es verschiedene Möglichkeiten, welche zum Beispiel in [16] aufgelistet sind. Für dieses Projekt wurde ein bereits existierendes Tool zur Kamerakalibrierung verwendet, die *GML C++ Camera Calibration Toolbox*[1]. Dieses verwendet eine häufig benützte Kalibrierungsmethode mithilfe der Erkennung eines Schachbrettmusters in mehreren mit der gleichen Kamera aufgenommenen Bildern. Das Tool liefert als Ergebnis die *intrinsische Kameramatrix*[2]

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

und die Verzerrungskoeffizienten. f_x und f_y beschreiben die Brennweiten der Kamera in Richtung der X- und Y-Achse. s gibt die Schräge der Achsen an, beträgt jedoch normalerweise 0. Die Hauptachse der Kamera ist jene Linie, welche rechtwinklig zur Bildebene steht und durch das *Pinhole* der Kamera führt. Jener Punkt, an dem diese Linie die Bildebene schneidet, ist der sogenannte *Bildmittelpunkt*. x_0 und y_0 geben, wie in Abbildung 3.21 zu sehen ist, den Abstand zu diesem Punkt relativ zum Ursprungspunkt des Bildes an. Mithilfe dieser Kameradaten kann anschließend die Homographiematrix für jedes aufgenommene Bild dieser Kamera berechnet werden.

3.2.2 Berechnung der Transformationsdifferenzen

Um die relative Position und Rotation zur Kamera für jeden Bildschirm berechnen zu können, müssen als nächstes jedem im Bild erkannten Bildschirm seine realen Abmessungen zugewiesen werden. Diese wurden zuvor händisch abgemessen und in den Konfigurationsdateien der einzelnen MuMu-Einheiten gespeichert. Da jede Einheit auch seine Marker-ID kennt und diese der Kalibrierungsanwendung mitsamt seiner Abmessungen zur Verfügung stellt, können jedem erkannten Bildschirm seine realen Abmessungen zugewiesen werden. Es ist jedoch nicht sichergestellt, dass die Rotation der Vierecke mit der der echten Bildschirme übereinstimmt, das heißt die zum Beispiel linke obere Ecke eines gefundenen Vierecks könnte jeder Ecke eines echten Bildschirmes entsprechen. Deshalb müssen die Ecken der gefundenen Marker zuerst noch richtig sortiert werden. Dies kann aufgrund

[1] http://graphics.cs.msu.ru/en/node/909

[2] Die intrinsischen Kameraparameter beschreiben verschiedene interne Eigenschaften einer Kamera, wie etwa Brennweite oder Bildmittelpunkt. Sie dienen dazu den Zusammenhang zwischen den zweidimensionalen Bildkoordinaten und den dreidimensionalen Kamerakoordinaten wieder herzustellen. Im Gegensatz dazu stehen die extrinsischen Kameraparameter. Diese beschreiben die Position und Orientierung einer Kamera im Raum, und dienen somit dazu den Zusammenhang zwischen dem Kamera- und Weltkoordinatensystem wieder herzustellen.

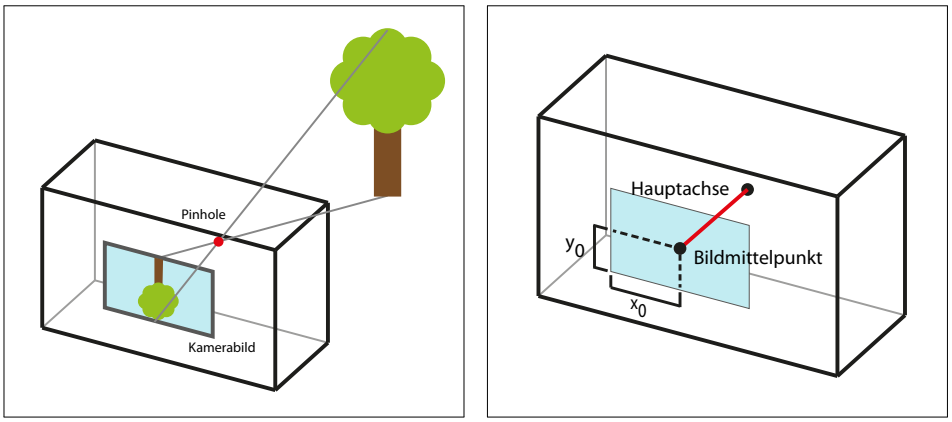


Abbildung 3.21: Darstellung einiger intrinsischer Kameraparameter anhand eines vereinfachten Kameramodells. Grafik erstellt nach [21].

Abbildung 3.22: Da die Ausgangsrotation der Markermuster bekannt ist, können den gefundenen Bildschirmeckpunkten ihre realen Eckpunkte aufgrund der im Bild gefundenen Rotation der Markermuster zugewiesen werden. Die realen linken oberen Eckpunkte der Bildschirme sind im Bild rot markiert. Die restlichen Eckpunkte sind im Uhrzeigersinn geordnet.

der Rotation des inne liegenden Markers bestimmt werden, da bei diesem die Ausgangsrotation bekannt ist. Der Eckpunkt eines gefundenen Bildschirmes, welcher die geringste Distanz zum linken oberen Eckpunkt seines Markers aufweist, entspricht somit der linken oberen Ecke des realen Bildschirmes. Dieses Ergebnis ist in Abbildung 3.22 zu sehen.

Da die realen dreidimensionalen Eckpunkte eines Bildschirmes nun den gefundenen zweidimensionalen Koordinaten Punkt für Punkt zugewiesen werden können und die intrinsischen Daten der verwendeten Kamera bekannt sind, kann von jedem Bildschirm die Position und Rotation relativ zur Kamera berechnet werden. Diese Problemstellung wird als *PnP-Problem* (*Perspective-n-Point*) bezeichnet, bei dem die Position einer kalibrierten Ka-

mera mithilfe von n dreidimensionalen Punkten im Weltkoordinatensystem und n korrespondierenden zweidimensionalen Punkten im Bild angenähert werden kann. Probleme mit $n \geq 3$ sind lösbar. In diesem Projekt wurde für diese Berechnung die Funktion `solvePnP` aus der Computer Vision-Library *OpenCV*[3] verwendet, welche drei verschiedene Ansätze für die Lösung eines PnP-Problems implementiert:

- `CV_ITERATIVE`,
- `CV_P3P`,
- `CV_EPNP`.

Nähere Informationen zu den einzelnen Ansätzen sind in [20] zu finden. Bei einem Vergleich dieser drei Implementierungen lieferte die Methode CV_ITERATIVE die genaueren Ergebnisse, wie in Abschnitt 5.1 näher beschrieben ist. Deshalb wurde diese für das Projekt verwendet.

Die Funktion `solvePnP` liefert als Ergebnis einen Translations- und einen Rotationsvektor, welche Position und Orientierung eines Bildschirmes im Kamerakoordinatensystem angeben. Für jeden der gefundenen Bildschirme werden so seine relativen Transformationsdaten berechnet. Die Distanz zwischen zwei beliebigen Bildschirmen entspricht dann dem Betrag des Verbindungsvektors der Translationsvektoren dieser Bildschirme. Für die Berechnung der Rotationsunterschiede zwischen den einzelnen Bildschirmen wurde nachstehender Lösungsansatz verfolgt. Hierfür wird deren Rotation um die Z-Achse (jene Achse in Richtung des Bildschirmbetrachters) im Koordinatensystem des jeweiligen Bildschirmes benötigt. Dazu wurde zuerst der Normalvektor zur Bildschirmoberfläche eines jeden Monitors im Kamerakoordinatensystem berechnet. Anschließend wird der Rotationsvektor jedes Bildschirmes in ein Quaternion umgewandelt. Mithilfe dessen kann die Rotation um eine bestimmte Achse (hier der zuvor berechnete Normalvektor) berechnet werden. Somit wird die Rotation eines Bildschirmes um die Z-Achse in dessen eigenen Koordinatensystem bestimmt, und nicht jene um die Z-Achse im Kamerakoordinatensystem. Die Differenz der einzelnen berechneten Winkel gibt somit den Rotationsunterschied der einzelnen Bildschirme an. Da diese Berechnungsschritte jedoch schlechtere Ergebnisse lieferten als der Rotationswert um die Z-Achse des Kamerakoordinatensystems, wurden letztere für die Differenzberechnung verwendet.

[3] http://opencv.org/

Kapitel 4

Evaluierung

4.1 Testaufbau

Um herauszufinden, ob das automatische Kalibrieren des MuMu-Systems mittels optischen Trackings bei einer Live-Umgebung praktikabel ist, werden verschiedene Aspekte des entwickelten Projektes getestet. Die wichtigste Voraussetzung für das automatische Kalibrierungssystem ist, dass dieses den Kalibrierungsvorgang vereinfacht und beschleunigt. Die so ermittelten Einstellungswerte sollen dabei die gleiche Genauigkeit, oder sogar eine bessere, aufweisen wie die der manuell gemessenen.

Um gültige möglichst exakte Vergleichswerte für die berechneten Kalibrierungsdaten zu erhalten, werden für die ersten durchgeführten Tests keine manuell gemessenen Daten verwendet, da diese selbst nicht vollkommen genau sein können. Deshalb könnte, falls unterschiedliche Ergebnisse vorliegen, nicht festgestellt werden, ob der Messfehler nicht schon beim manuellen Ausmessen der Bildschirme verursacht wurde. Aus diesem Grund werden bei den ersten Tests statt Bildschirmen ausgedruckte Quadrate auf Papier verwendet, deren Abmessungen in einem Bildbearbeitungsprogramm auf bestimmte Werte eingestellt werden. Bei Ungenauigkeit des verwendeten Druckers besteht zwar immer noch die Möglichkeit auf nicht exakte Vergleichswerte, jedoch ist die Ausprägung dieser Fehler geringer als bei den manuell gemessenen Daten und zu vernachlässigen. In den anfänglichen Tests werden einfache Quadrate verwendet und noch nicht die ID-Marker, da vorerst nur die Erkennung von Vierecken und deren Abstandsberechnung, und nicht die Identifizierung einzelner Marker getestet werden soll.

Die Analyse der gedruckten Quadrate mit bestimmten Abmessungen ermöglicht die Evaluierung der Genauigkeit des entwickelten Systems. Dabei werden folgende Punkte beachtet: Einerseits soll überprüft werden, ob das System bei unterschiedlichen Abständen der Kamera zu den Quadraten immer dieselben Rotations- und Positionsdifferenzen zwischen den Quadraten liefert. Andererseits soll auch die Auswirkung des Neigungswinkels der Ka-

mera auf das Ergebnis analysiert werden.

Die nächsten Tests werden mit Bildschirmen durchgeführt auf denen die zugehörigen ID-Marker angezeigt werden. Für den verwendeten Testaufbau wurden fünf Monitore auf einer Ebene nebeneinander platziert und verschieden rotiert. Die aufgenommenen Bilder für die Kalibrierung des System enthalten immer alle diese fünf Bildschirme. Es wurden Bilder aus möglichst mittiger Position vor der Bildschirmreihe aufgenommen. Dies ist der Optimalfall, bei dem angenommen wird, dass dieser die besten Kalibrierungsergebnisse liefert.

Als nächstes werden die Daten eines bereits verwendeten Aufbaus des MuMu-Systems analysiert. Die Daten der Bildschirme wurden damals manuell gemessen und für die Einstellungen verwendet. Nun soll überprüft werden, ob das automatische Kalibrierungssystem ähnliche Daten liefert.

Weiters wird auch die Erkennung des Markermusters getestet. Es wird überprüft, ob dieses aus verschiedenen Abständen und mit starker Verzerrung noch identifiziert werden kann. Weiters soll überprüft werden, wie viele Pixel mindestens für die Darstellung eines Markers benötigt werden.

Außerdem soll die Genauigkeit der Marker- und Bildschirmerkennung auch unter verschiedenen Lichtverhältnissen getestet werden. Ein Vorteil des verwendeten Markererkennungs-Algorithmus ist, dass dieser bei unterschiedlicher Beleuchtung robuste Ergebnisse liefert. Bei der Erkennung von Bildschirmen muss insbesondere überprüft werden, wie sich die Markererkennung bei direktem Licht verhält, da dieses Reflexionen verursachen kann. Wenig Licht ist hier kein Problem, da die Bildschirme von sich aus leuchten.

Zuletzt wird überprüft, wie schnell ein Aufbau des MuMu-Systems im Vergleich zur herkömmlichen manuellen Methode mithilfe des automatischen Kalibrierungssystems eingerichtet werden kann. Dazu wird zuerst ein Aufbau mit zwei Bildschirmen verwendet. Die Positions- und Rotationsdaten werden dann sowohl manuell als auch mithilfe der Kalibrierungsanwendung gemessen und eingestellt. Bei beiden Methoden wird die benötigte Zeit notiert. Dieser Vorgang wird dann mit einem Aufbau mit drei Monitoren wiederholt. So kann festgestellt werden, wie viel Zeit ungefähr pro zusätzlichem Monitor bei der manuellen oder automatischen Kalibrierung benötigt wird.

4.2 Testergebnisse

Für alle durchgeführten Tests wurde die Kamera eines *Nokia Lumia 620* mit einer Auflösung von 1277×717 Pixel verwendet.

4.2.1 Tests auf Papier

Für die ersten Tests wurden Quadrate mit 5cm Seitenlänge in bestimmten Abständen voneinander auf Papier ausgedruckt. Diese sind in Abbildung 4.1 zu sehen. Das dort rot umrandete Quadrat wird im Folgenden als Marker

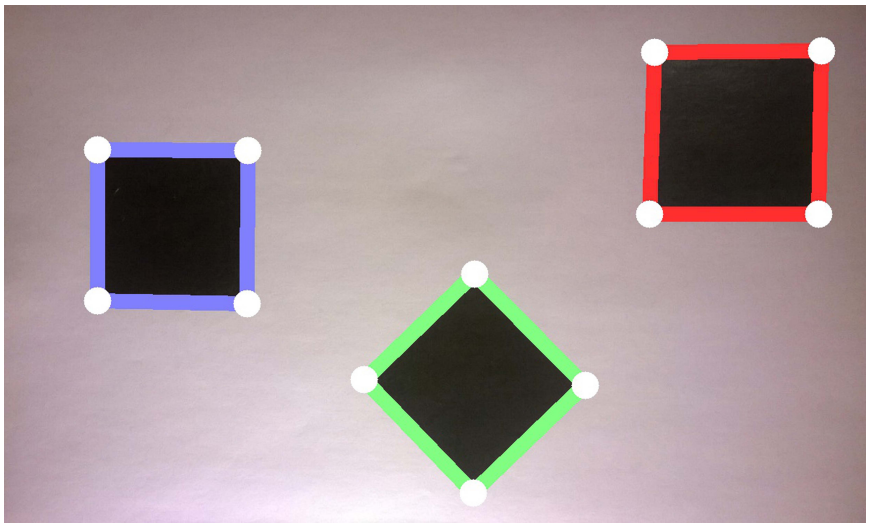


Abbildung 4.1: Quadrate mit 5cm Seitenlänge, welche für die Tests auf Papier verwendet wurden. Zwischen Marker 1 (rot) und Marker 2 (blau) liegen 18,25cm, zwischen Marker 1 und Marker 3 (grün) liegen 11,31cm und zwischen Marker 2 und Marker 3 liegen 11,18cm. Diese Werte geben den Abstand zwischen den Mittelpunkten zweier Quadrate an. Marker 1 und Marker 2 wurden nicht rotiert, Marker 3 jedoch um 45°.

1 bezeichnet, das blau umrandete als Marker 2 und das grün umrandete als Marker 3. Von diesem Bild wurden aus verschiedenen Abständen Fotos aufgenommen. Die Kamera wurde dabei möglichst parallel zum Papier gehalten. Der geringst mögliche Abstand der Kamera zum Papier, mit dem alle drei Quadrate auf einem Bild der Kamera vollständig sichtbar sind, beträgt 25cm. Dieser Abstand wurde schrittweise bis zu 175cm erhöht. Ab einem Abstand von 180cm konnten die Quadrate von der Kalibrierungsanwendung nicht mehr erkannt werden, und somit auch keine Distanzberechnung durchgeführt werden.

Die aufgenommenen Bilder wurden anschließend von der entwickelten Kalibrierungsanwendung analysiert. Dabei wurden für jedes Bild 1000 Berechnungsdurchläufe durchgeführt, um aussagekräftige Durchschnittswerte für die Distanz- und Rotationsunterschiede der Quadrate zu erhalten.

Die Ergebnisse der Distanzabweichung sind in Abbildung 4.2 zu sehen. Wurde die Kamera in einem Abstand von 40 bis ungefähr 130cm gehalten, so sind die Abweichungen von den tatsächlichen Distanzwerten verschwindend gering. Wird die Kamera jedoch zu nah zum Bild gehalten, so erhöht sich die Abweichung rapide. Bei größeren Entfernungen nehmen die Abweichungen langsam zu. Es konnte jedoch nicht getestet werden, ob sich dieser Trend gleichmäßig fortsetzt, da zuvor die Erkennung der Vierecke im Bild scheitert.

Abbildung 4.3 zeigt die Ergebnisse der Rotationsabweichung bei ver-

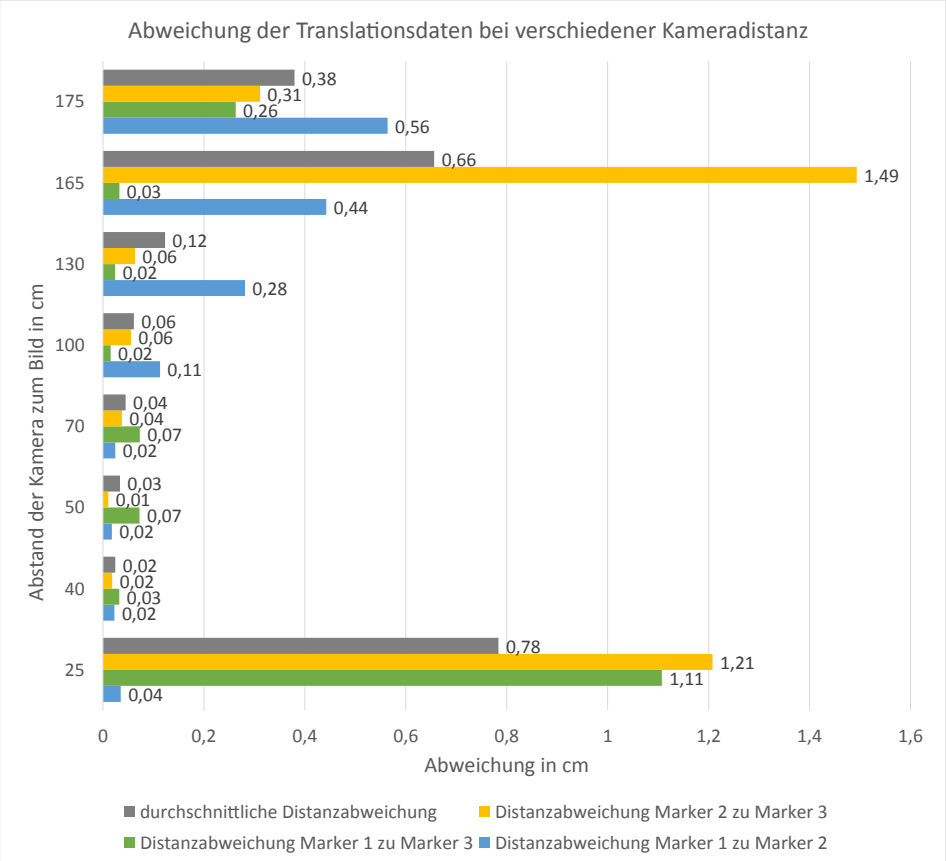


Abbildung 4.2: Durchschnittliche Abweichung bei 1000 Durchgängen für die Berechnung der Abstände zwischen zwei Quadraten aus der Abbildung 4.1. Die gelben, grünen und blauen Balken geben jeweils die durchschnittliche Abweichung zwischen zwei bestimmten Quadraten an. Die grauen Balken stellen dabei die Durchschnittswerte aus diesen drei Werten dar.

schiedenen Abständen der Kamera zum Papier. Mit zunehmenden Abstand kann ein leicht irregulärer Trend der Werte nach oben beobachtet werden. Der Erkennungsfehler steigt jedoch nie über 1.5°.

Für die nächsten Tests wurde das selbe Bild mit den drei Quadraten verwendet. Es wurden jedoch nicht die Einflüsse des Abstandes zur Kamera, sondern die des Neigungswinkels der Kamera gemessen. Da aus den obigen Ergebnissen hervorgeht, dass ein Abstand der Kamera zum Bild von 40cm die besten Ergebnisse liefert, wurde dieser für folgende Tests verwendet. Die Kamera wird dabei von oben ausgehend um 90° mit diesem fixen Abstand zum Bild rotiert.

In Abbildung 4.4 sind die Ergebnisse der Distanzabweichung bei verschiedenen Kameraneigungswinkeln ersichtlich. Bis zu einem Winkel von etwa

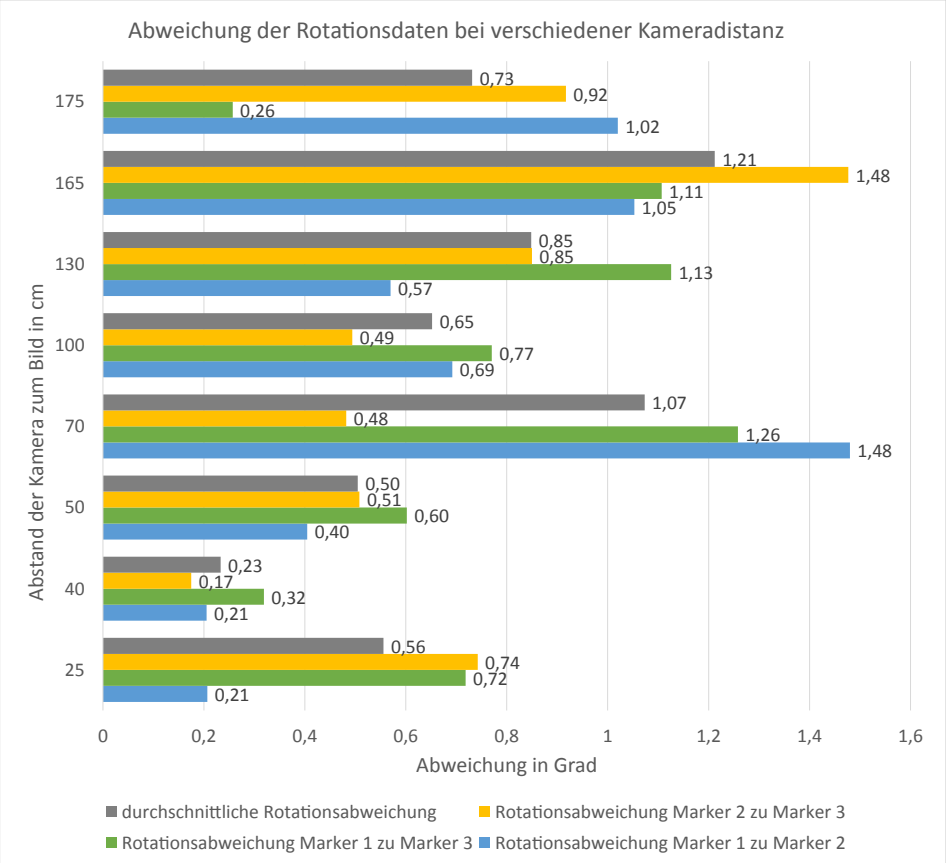


Abbildung 4.3: Durchschnittliche Abweichung bei 1000 Durchgängen für die Berechnung der Rotationsunterschiede von zwei Quadraten aus der Abbildung 4.1. Die gelben, grünen und blauen Balken geben jeweils die durchschnittliche Abweichung bei zwei bestimmten Quadraten an. Die grauen Balken stellen dabei die Durchschnittswerte aus diesen drei Werten dar.

50° bleiben die Ergebnisse annähernd gleich. Danach steigt die Abweichung pro 10° nahezu exponentiell an. Ab einem Neigungswinkel von mehr als 80° können die Quadrate nicht mehr erkannt werden. Der Distanzunterschied zwischen Marker 1 und Marker 2 weicht jedoch vom restlichen Ergebnis ab. Diese Distanz kann meist gut erkannt werden, die Translationsabweichung bleibt bei allen Winkeln annähernd konstant. Dies lässt sich darauf zurückführen, dass diese beiden Quadrate aus Sicht der Kamera hauptsächlich in Richtung der X-Achse verschoben liegen und nicht wie Marker 3 in Richtung der Z-Achse.

In Abbildung 4.5 sind die Ergebnisse der Rotationsabweichung bei unterschiedlichem Neigungswinkel zu sehen. Diese steigt mit zunehmendem Winkel stark an. Schon ab einem Neigungswinkel der Kamera von 30° liegt

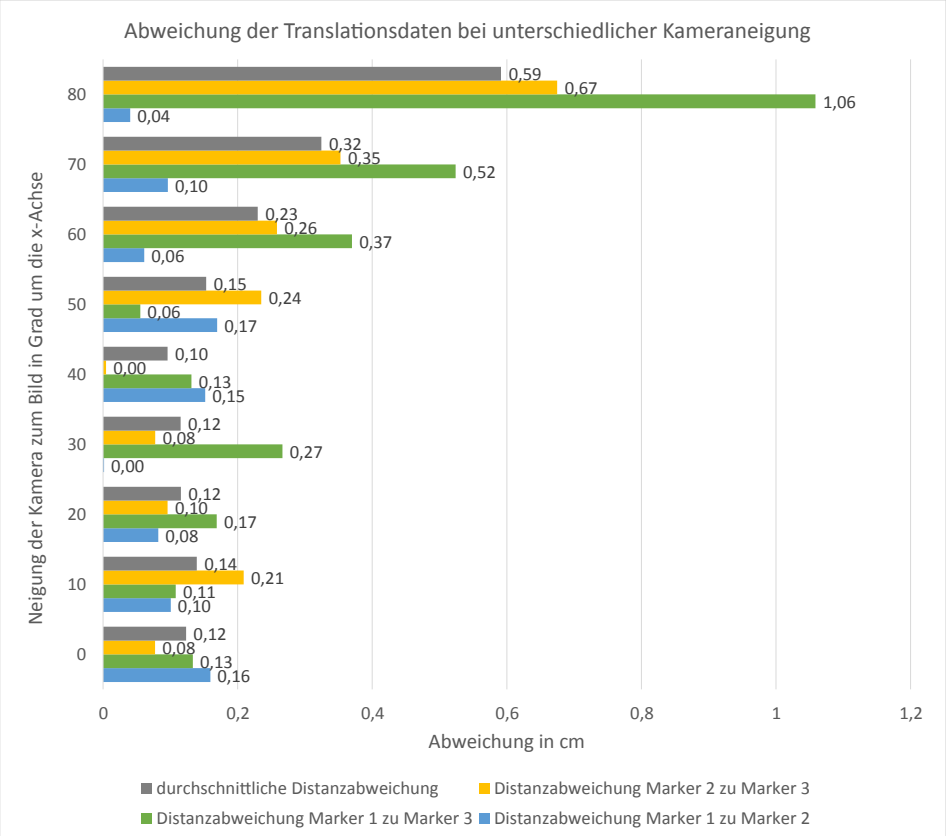


Abbildung 4.4: Durchschnittliche Abweichung bei 1000 Durchgängen für die Berechnung der Abstände zwischen zwei Quadraten aus der Abbildung 4.1. Die gelben, grünen und blauen Balken geben jeweils die durchschnittliche Abweichung zwischen zwei bestimmten Quadraten an. Die grauen Balken stellen dabei die Durchschnittswerte aus diesen drei Werten dar.

der durchschnittliche Fehler bei etwa 4,5° und die Ergebnisse werden schnell unbrauchbar. Dieser Fehler steigt auf bis zu 28° bei einem Neigungswinkel von 80° an. Die Ausnahme bildet hier wiederum die Abweichung der Rotation zwischen Marker 1 und Marker 2. Diese steigt zwar ebenfalls mit zunehmendem Neigungswinkel an, jedoch nur auf einen Maximalfehler von 3.6°.

4.2.2 Tests zur Erkennung der Marker-ID

Um die Erkennung der Markermuster zu testen, wurden von vier nebeneinander platzierten Markern mit einer Seitenlänge von 5,4cm Bilder aus verschiedenen Abständen und Winkeln aufgenommen. Um ein gültiges Ergebnis zu erhalten, wurden diesmal nur 10 Markererkennungs-Durchläufe

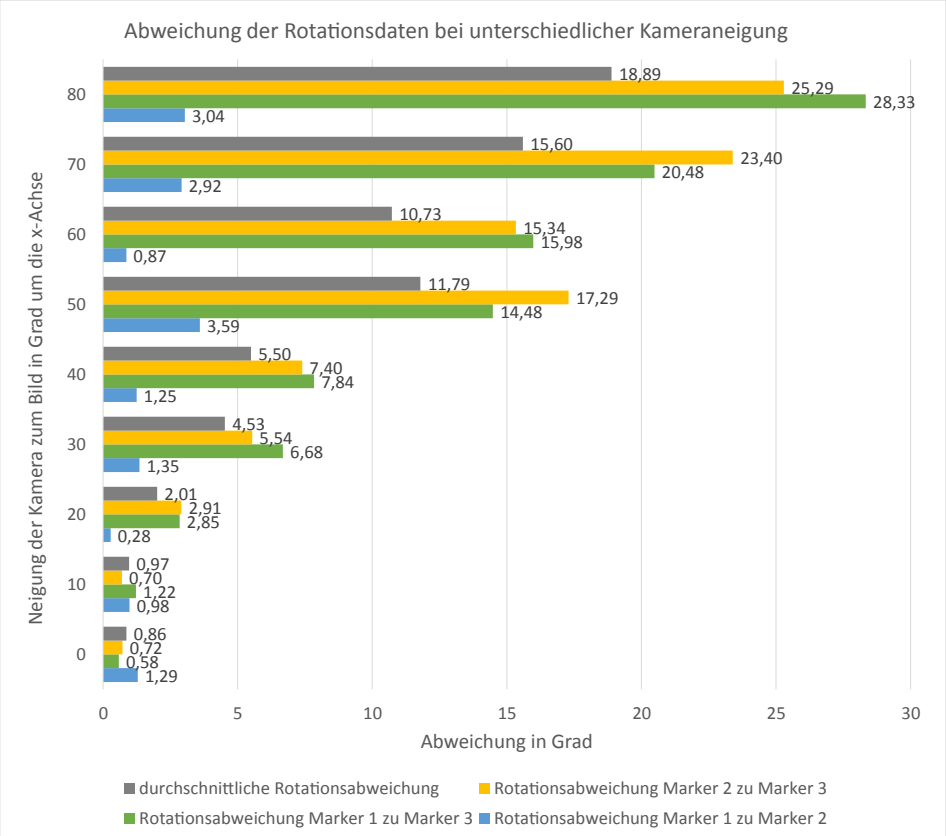


Abbildung 4.5: Durchschnittliche Abweichung bei 1000 Durchgängen für die Berechnung der Rotationsunterschiede von zwei Quadraten aus der Abbildung 4.1. Die gelben, grünen und blauen Balken geben jeweils die durchschnittliche Abweichung bei zwei bestimmten Quadraten an. Die grauen Balken stellen dabei die Durchschnittswerte aus diesen drei Werten dar.

pro aufgenommenen Bild durchgeführt, da diese manuell auf die Anzahl gefundener Marker überprüft werden müssen, um falsch erkannte Marker, welche das Ergebnis verfälschen könnten, ausschließen zu können.

Zuerst wurde die Auswirkung der Verzerrung durch den Neigungswinkel der Kamera getestet. Hierfür wurden von den Markern Bilder aus einem Abstand von 40cm mit einem Neigungswinkel von 0 bis 90° aufgenommen. Wie bei den Ergebnissen in Abbildung 4.6 zu sehen ist, ist die Erkennungsrate bis zu einem Neigungswinkel von 60° fehlerlos. Ab 70° können einige Marker nicht mehr erkannt werden, bis bei einem Neigungswinkel von knapp 90° keiner mehr gefunden wird.

Als nächstes wurde die Erkennung der Marker aus zunehmender Entfernung getestet. Die Ergebnisse dieser Tests sind in Abbildung 4.7 zu sehen. Der geringst mögliche Abstand zur Kamera, bei dem alle Marker vollstän-

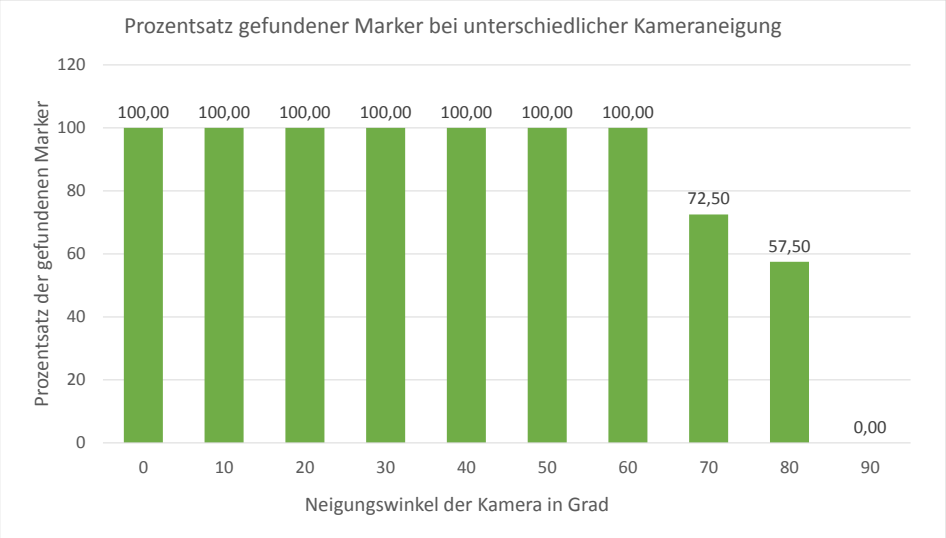


Abbildung 4.6: Durchschnittlicher Prozentsatz gefundener Marker bei Aufnahmen mit unterschiedlichem Kameraneigungswinkel bei 10 Erkennungsdurchläufen pro Aufnahme.

dig im Bild sichtbar sind, beträgt 20cm. Es wurden mehrere Bilder in regelmäßigen Abständen aufgenommen. Bis zu einer Distanz von etwa 70cm ist die Erkennung mit dem verwendeten Setup nahezu fehlerlos und stabil. Danach schwanken die Ergebnisse und verschlechtern sich stetig, bis ab einem Abstand von 160cm keine Marker mehr erkannt werden. Diese Werte hängen stark von der bei der Kantendetektion verwendeten Rastergröße in Abschnitt 3.1.1 ab. Würde dieses Raster kleiner gewählt werden, so könnten bei höheren Distanzen bessere Ergebnisse erzielt werden. Darunter würde jedoch die Geschwindigkeit des Algorithmus leiden.

Bei einem Abstand von 70cm beträgt die Größe eines Markers im Bild ungefähr 80×80 Pixel. Dies ist die Mindestgröße eines Markers, welcher mit der verwendeten Rastergröße robust erkannt werden kann. Bei einem Abstand von 160cm ist ein Marker im Bild etwa 35×35 Pixel groß. Marker mit gleicher oder niedrigerer Auflösung werden nicht mehr erkannt.

4.2.3 Tests mit Monitoren

Nachdem die Genauigkeit des Systems mittels ausgedruckter Quadrate und ID-Markern auf Papier getestet wurde, werden diese Tests mit Monitoren und darauf angezeigten ID-Markern fortgeführt.

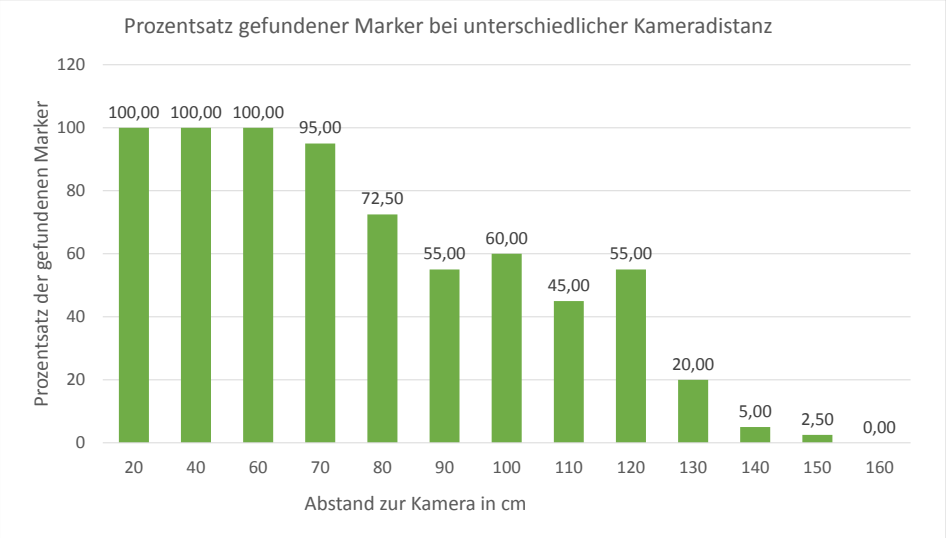


Abbildung 4.7: Durchschnittlicher Prozentsatz gefundener Marker bei Aufnahmen aus unterschiedlicher Entfernung bei 10 Erkennungsdurchläufen pro Aufnahme.

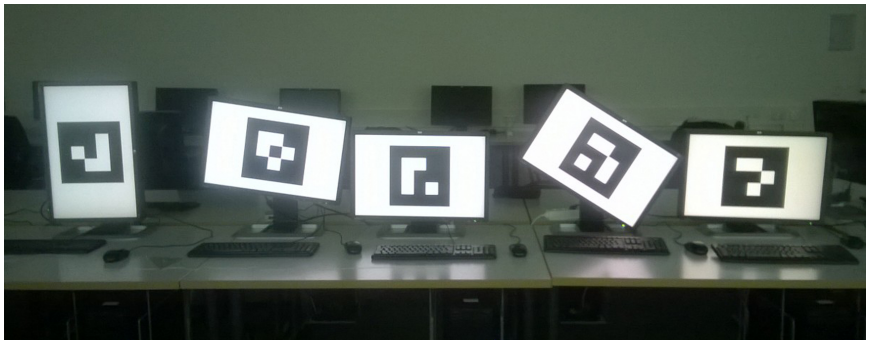


Abbildung 4.8: Fünf Monitore mit einer Abmessung von 51,9 × 32,45cm, welche für den Testaufbau verwendet wurden.

Testaufbau mit fünf Bildschirmen

Bei dem in Abbildung 4.8 gezeigten Testaufbau wurden sowohl die Distanz- als auch die Rotationsunterschiede zwischen den einzelnen Bildschirmen manuell gemessen. Anschließend wurden dieselben Daten mittels der automatischen Kalibrierungsanwendung berechnet. Ein Vergleich dieser Ergebnisse ist in Abbildung 4.9 zu sehen. Das Bild, welches für die Kalibrierungsberechnungen verwendet wurde, wurde aus einer möglichst mittigen Position ohne großen Neigungswinkel aufgenommen, da bei diesem die besten Ergebnisse erzielt werden können, wie aus den vorherigen Tests hervorgeht. Es

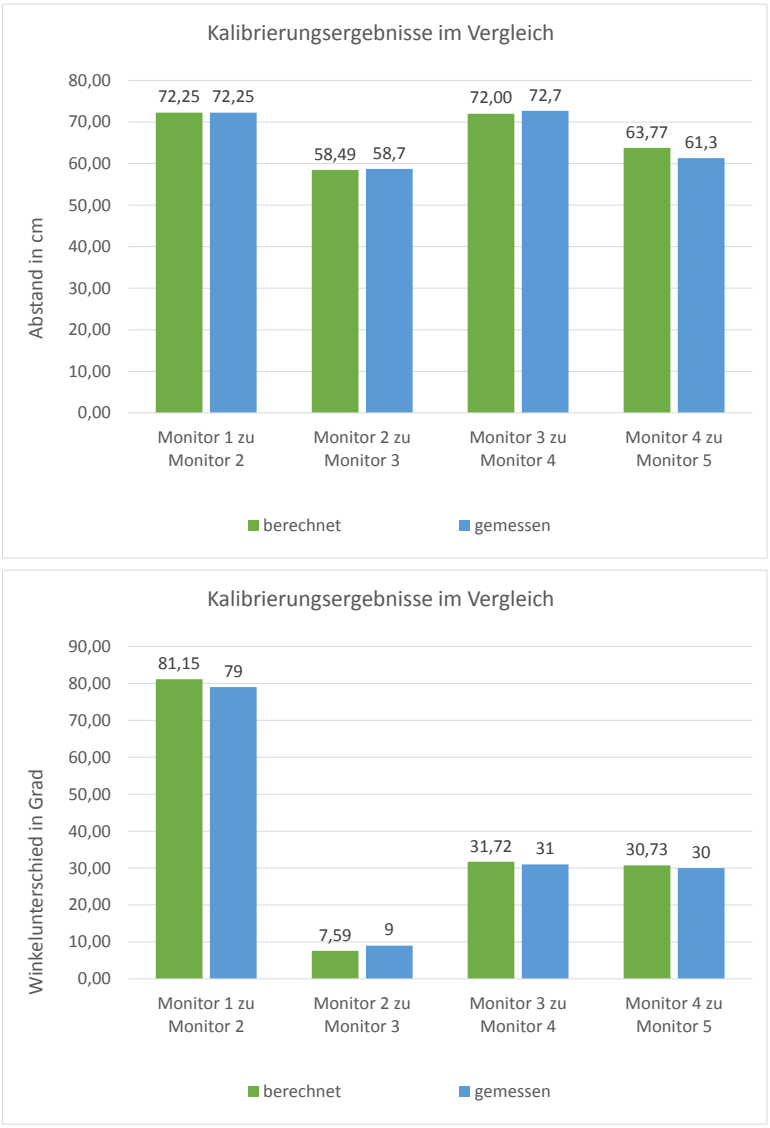


Abbildung 4.9: Unterschiede zwischen manuell gemessenen und automatisch berechneten Distanz- und Rotationswerten für ein Setup mit fünf Monitoren. Die berechneten Ergebnisse zeigen Durchschnittswerte aus 1000 Durchläufen.

wurde versucht, beim manuellen Messen der Daten so genau wie möglich vorzugehen. Fehler können jedoch nicht ausgeschlossen werden. Die Ergebnisse beider Kalibrierungsmethoden weisen keine großen Unterschiede auf, deswegen könnte das berechnete Ergebnis auch genauer sein, als das manuell gemessene.

Da bei den Tests zur Erkennung des Markermusters festgestellt wurde,

dass mindestens 80×80 Pixel für eine robuste Erkennung benötigt werden, kann daraus abgeleitet werden, wie viele Pixel für einen Bildschirm mitsamt seinem inne liegenden Marker benötigt werden. Für die bei diesen Tests verwendeten Bildschirme ergibt sich eine ungefähre Mindestgröße von 160×100 Pixel, bei der die automatische Kalibrierung noch stabile Ergebnisse liefert und die Bildschirme mitsamt Marker gut erkennen kann. Die so ermittelten Werte ergeben je nach verwendeter Kamera und deren Auflösung eine andere Maximaldistanz, aus der Bilder des Aufbaus gemacht werden können. Daraus ergibt sich auch die maximale Anzahl an Bildschirmen, die mit nur einem Bild gleichzeitig kalibriert werden können.

Live Setup

Um zusätzlich zum vorher beschrieben simulierten Testaufbau noch reale Ergebnisse für die automatische Kalibrierungsanwendung zu erhalten, wurde das verwendete MuMu-Setup bei einer Ausstellung nachträglich analysiert. Die Abweichungen der berechneten Werte zu den bei der Ausstellung verwendeten Werte sind in Abbildung 4.10 zu sehen.

4.2.4 Tests unter verschiedenen Lichtbedingungen

Um herauszufinden, inwieweit Lichtverhältnisse das Ergebnis der Kalibrierung beeinflussen, wurden Tests unter extremen Bedingungen durchgeführt. Wenig Beleuchtung hat auf das Ergebnis der Bildschirmerkennung keinen Einfluss, da Bildschirme selbstleuchtend sind. Das andere Extrem, direktes Sonnenlicht, beeinflusst die Erkennung von Bildschirmen jedoch stark. Wie in Abbildung 4.11 ersichtlich ist, treten bei heller Umgebung Reflexionen auf. Je nach Reflexionsstärke können manche Bildschirme aber noch erkannt werden.

4.2.5 Zeitmessung

Um herauszufinden, ob und in welchem Ausmaß sich die automatische von der manuellen Kalibrierungszeit unterscheidet, wurde ein Testszenario mit drei Monitoren aufgebaut. Die Kalibrierung wurde jeweils einmal mit zwei und einmal mit allen drei Monitoren durchgeführt. Für die manuelle Kalibrierung wurden folgende Schritte in die Zeitmessung miteinbezogen:

- Messung der X- und Y-Distanzen zwischen den Monitoren,
- Messung der Rotationsunterschiede zwischen den Monitoren,
- Einstellen der gemessen Daten,
- eventuelle Nachbesserungen.

Bei der automatischen Kalibrierung wurden folgende Schritte gemessen:

- Anzeigen der Marker auf den Bildschirmen mittels MuMu-System,

Abbildung 4.10: Unterschiede zwischen manuell eingestellten und berechneten Positions- und Rotationsdaten einzelner Bildschirme eines MuMu-Aufbaus. Je ein Balkenpaar steht für den Abstand eines Monitors zum Referenzmonitor.

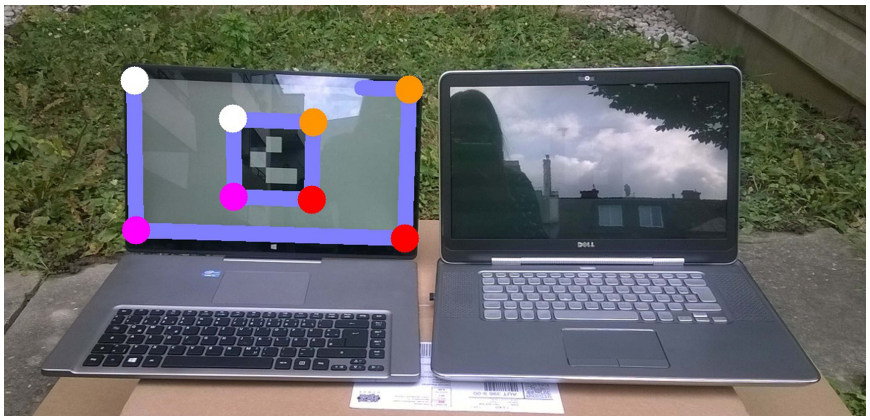


Abbildung 4.11: Test der Bildschirmerkennung bei nicht optimalen Lichtbedingungen unter freiem Himmel. Der linke Monitor kann trotz leichter Reflexionen noch erkannt werden. Die obere linke Ecke kann korrekt aus dem Schnittpunkt der linken Kante und dem oberen gefundenen Liniensegment berechnet werden, auch wenn die gesamte Oberkante des Monitors aufgrund der Reflexionen nicht vollständig sichtbar ist. Beim rechten Monitor sind die Reflexionen zu stark, um einen Marker zu erkennen.

- Fotografieren der Bildschirme,
- Berechnung der Translations- und Rotationswerte,
- Einstellen der berechneten Daten,
- eventuelle Nachbesserungen.

Die dadurch gemessenen Daten sind in Abbildung 4.12 ersichtlich. Es sollte herausgefunden werden, wie viel Zeit pro zusätzlichem Monitor durchschnittlich für die Kalibrierung des Systems benötigt wird. Bei der manuellen Kalibrierung sind dies 2 Minuten und 31 Sekunden. Bei der automatischen nur 27 Sekunden. Diese ergeben sich bei letzterer hauptsächlich durch das Einstellen der Daten und eventuelle Nachbesserungen für den zusätzlichen Monitor, da die restlichen Schritte nahezu unabhängig von der Anzahl der verwendeten Monitore sind. Da das manuelle Einstellen der berechneten Daten in einer späteren Version der Kalibrierungsanwendung noch automatisiert werden kann, kann der Zeitaufwand generell und pro zusätzlichem Bildschirm noch weiter verringert werden.

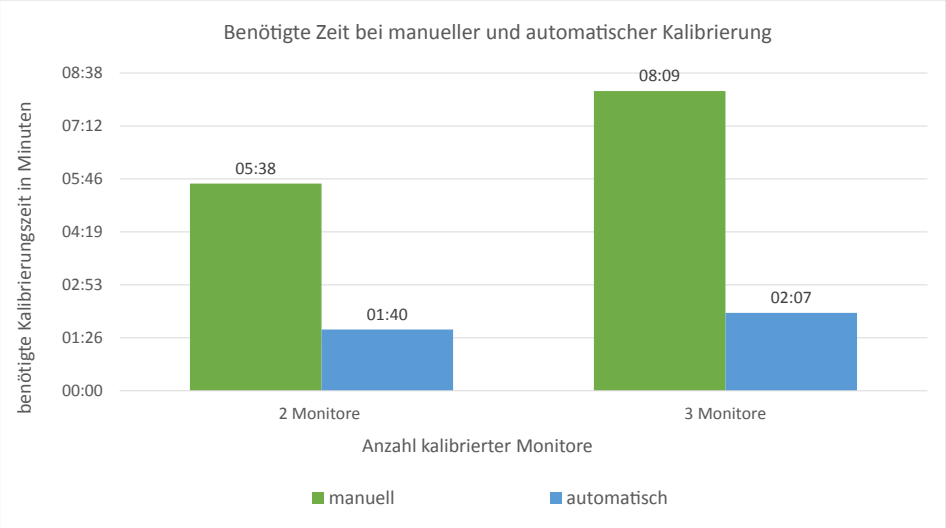


Abbildung 4.12: Zeitmessung bei manuellem und automatischem Kalibrierungsvorgang. Durchschnittswerte bei fünf Durchgängen.

Kapitel 5

Ergebnisse

5.1 Performance

Die Laufzeit des Algorithmus und die Qualität der Ergebnisse sind stark von verschiedenen Faktoren abhängig. Es wurde versucht, diese so gut wie möglich auf die Problemstellung anzupassen. Einer dieser Faktoren ist zum Beispiel die Größe des verwendeten Rasters in Abschnitt 3.1.1.

Um zu überprüfen, welche Rastergröße gewählt werden soll, wurden mehrere Testdurchläufe mit verschiedenen Werten mit Abbildung 4.8 durchgeführt. Deren Ergebnisse sind in Abbildung 5.1 zu sehen. Je größer das Ras-

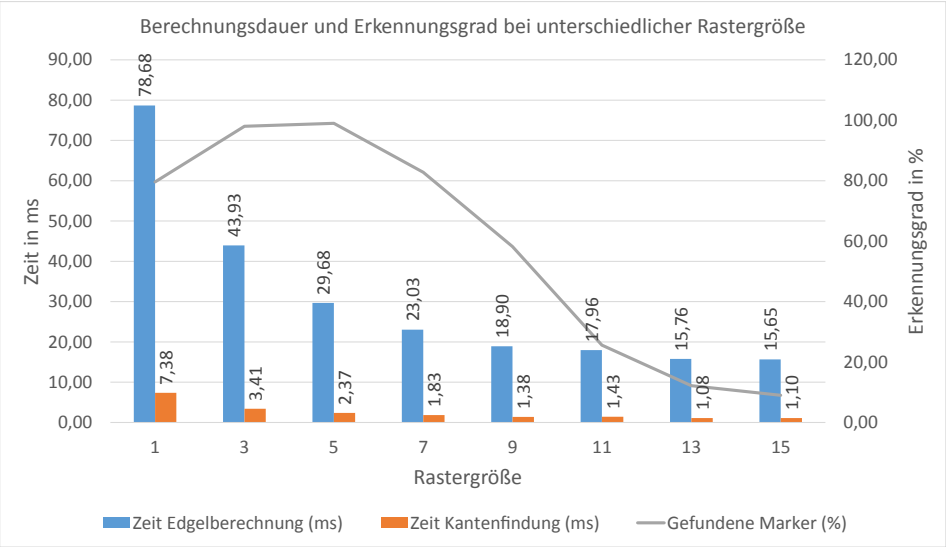


Abbildung 5.1: Berechnungsdauer und Erkennungsgrad bei unterschiedlicher Rastergröße. Die Ergebnisse sind Durchschnittswerte aus 100 Durchläufen.

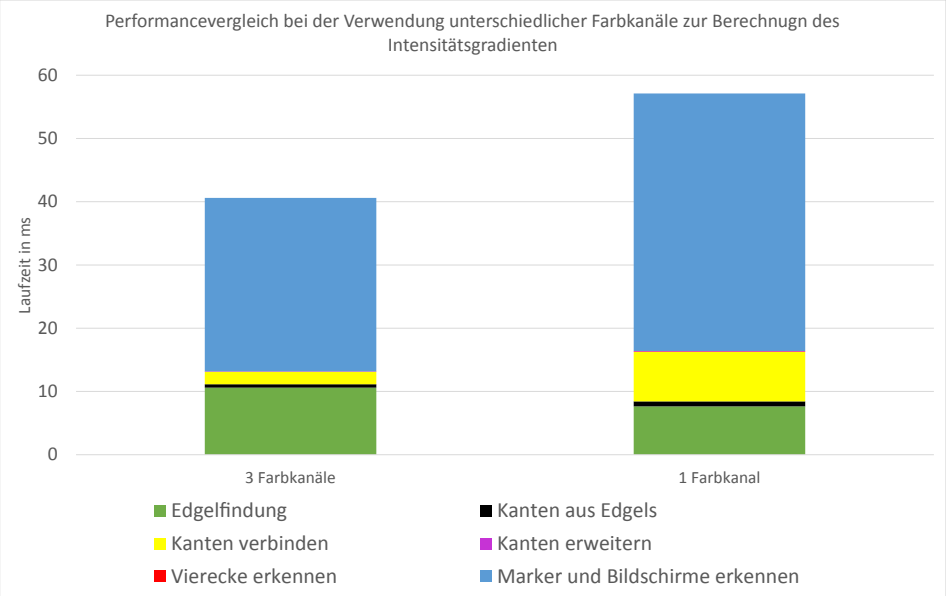


Abbildung 5.2: Performancevergleich der einzelnen Schritte der Kalibrierungsanwendung bei der Verwendung unterschiedlicher Farbkanäle zur Berechnung des Intensitätsgradienten. Die Ergebnisse sind Durchschnittswerte aus 1000 Durchläufen.

ter gewählt wird, desto schneller wird der Algorithmus, da weniger Pixel überprüft werden. Darunter leidet aber auch das Ergebnis der Markererkennung, da immer weniger der Marker im Bild gefunden werden können. Ist die Rastergröße zu klein gewählt, so können ebenfalls nicht alle Marker im Bild erkannt werden. Durch die hohe Anzahl an Edgels, wovon viele nicht Teil einer Markerkante sind, werden öfter falsche Linien gebildet und verschlechtern so das Endergebnis. Wie in der Grafik zu sehen ist, liefert eine Rastergröße von 5 Pixel die stabilsten Ergebnisse bei guter Laufzeit.

Zusätzlich wurde die Auswirkung der verwendeten Farbkanäle zur Berechnung des Intensitätsgradienten aus Abschnitt 3.1.1 überprüft. Je nachdem, ob dieser für jeden Farbkanal eines RGB-Pixels einzeln oder nur für den daraus berechneten Graustufenwert ermittelt wird, hat dies andere Auswirkungen auf die Performance des übrigen Algorithmus. Die Berechnung des Intensitätsgradienten, und somit der Edgels, ist bei der Verwendung von nur einem Farbkanal um etwa 30% schneller, wie bei den Ergebnissen in Abbildung 5.2 zu sehen ist. Da so, wie in Abbildung 3.4 ersichtlich ist, jedoch auch mehr Edgels gefunden werden, erhöht sich die Laufzeit der übrigen Schritte des Algorithmus so sehr, dass dieser insgesamt mehr Zeit benötigt. Diese Werte hängen jedoch auch stark vom zu analysierenden Bild und dessen Farben ab. Für diese Tests wurde die bereits erwähnte Abbildung 3.4 ver-

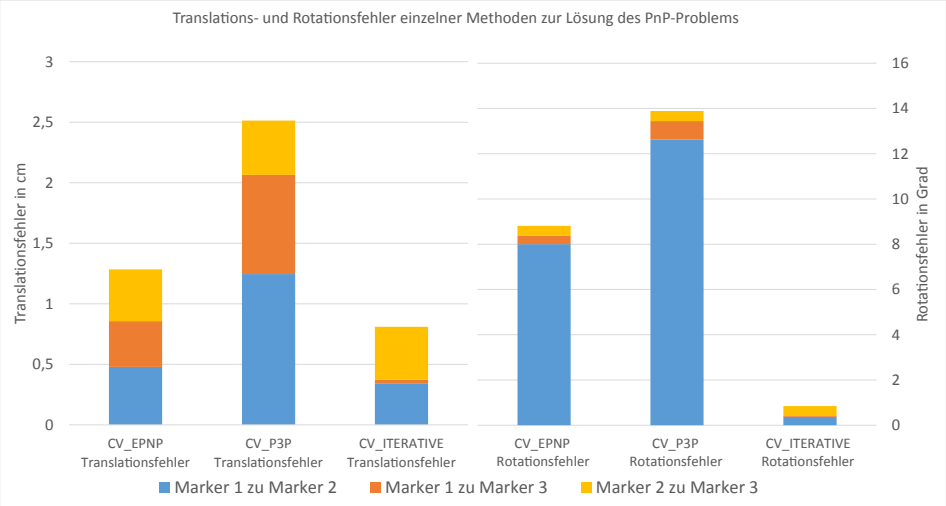


Abbildung 5.3: Translations- und Rotationsfehler der verschiedenen Implementierungen der Computer Vision-Library OpenCV `CV_EPNP`, `CV_P3P` und `CV_ITERATIVE`. Die Bezeichnungen Marker 1, Marker 2 und Marker 3 referenzieren die in Abbildung 4.1 zu sehenden Quadrate. Die Ergebnisse sind Durchschnittswerte aus 1000 Durchläufen.

wendet, welche Vierecke in einigen verschiedenen Farben zeigt. Damit kann ein Extremfall überprüft werden. Das andere Extrem wäre ein Bild mit äußerst niedriger Sättigung. Die beiden Methoden würden hierbei eine ähnliche Anzahl an Edgels liefern und deswegen wäre die Methode, welche den Graustufenwert zur Berechnung des Intensitätsgradienten verwendet, insgesamt schneller. Da jedoch nicht davon ausgegangen werden kann, dass in der Praxis nur farblose Umgebungen fotografiert werden, wurden die RGB-Kanäle für die Berechnung verwendet.

Möglichkeiten zur Lösung des PnP-Problems

Die verwendete Funktion `solvePnP` aus der Computer Vision-Library OpenCV bietet drei verschiedene Implementierungen zur Lösung des PnP-Problems. Um herauszufinden, welche die besten Ergebnisse liefert, wurden die Translations- und Rotationsabstände zwischen den einzelnen Vierecken aus Abbildung 4.1 mit allen drei Methoden berechnet. Die Unterschiede zu den tatsächlichen Werten sind in Abbildung 5.3 ersichtlich. Anhand dieser Tests wurde die Methode `CV_ITERATIVE` für das Projekt ausgewählt.

5.2 Fazit

Die Ergebnisse aus Kapitel 4 belegen, dass die Kalibrierung des MuMu-Systems mittels des entwickelten Projekts praktikabel ist. Vor allem bei einem Aufbau mit vielen Bildschirmen lohnt sich die automatisierte Kalibrierung, da der Vorgang um ein Vielfaches schneller ist als das manuelle Kalibrieren. Die berechneten Werte liegen außerdem sehr nahe an den tatsächlichen Abstands- und Rotationswerten der einzelnen Bildschirme, wenn das Bild, welches für die Kalibrierung verwendet wird, aus einer möglichst frontalen Perspektive mit möglichst geringem Neigungswinkel der Kamera aufgenommen wird.

Bei der automatischen Kalibrierung treten noch Probleme auf, wenn das Bild mit einem hohen Neigungswinkel der Kamera aufgenommen wird oder starke Spiegelungen auf den Bildschirmen sichtbar sind. Die Genauigkeit der Ergebnisse nimmt dann vor allem bei der Rotationsberechnung rapide ab. Bei zu starken Spiegelungen können manche Bildschirme nicht mehr erkannt werden.

Trotz der erwähnten Schwächen des entwickelten Projektes, stellt es eine enorme Verbesserung zum manuellen Kalibrierungsvorgang dar. Wenn das Kalibrierungsbild aus der korrekten Perspektive aufgenommen wird, können gleichwertige Ergebnisse schneller und ohne zusätzlich benötigte Messwerkzeuge oder Personen erzielt werden.

5.3 Ausblick

In Zukunft sind noch weitere Verbesserungsschritte für das umgesetzte Projekt geplant. Zunächst sollen die ermittelten Daten automatisch an den Server übertragen werden. Dazu soll das Projekt für mobile Geräte mit Kamera umgesetzt werden. Es bestehen zwei Möglichkeiten wie der Server die Positions- und Rotationsdaten erlangt. Diese könnten einerseits direkt auf dem verwendeten mobilen Gerät berechnet werden und anschließend an den Server gesendet werden. Sollte sich jedoch herausstellen, dass der damit verbundene Rechenaufwand zu hoch für viele gängige Geräte sein sollte, könnten auch nur die aufgenommenen Bilder an den Server übertragen werden und dieser könnte die Daten daraus selbst errechnen.

Außerdem könnte die Kalibrierungsanwendung dem Benutzer Hinweise zum optimalen Erstellen der Kalibrierungsbilder anzeigen. Auf mobilen Geräten könnte zum Beispiel deren Gyrosensor verwendet werden, um den Benutzer darauf hinzuweisen, wenn er das verwendete Gerät mit zu starkem Neigungswinkel hält. Weiters könnte aus den aufgenommenen Bilddaten und den Seitenverhältnissen der einzelnen Bildschirme bestimmt werden, ob der Benutzer frontal oder in einem zu hohen Winkel zum MuMu-Aufbau steht. Diese Informationen können den Benutzern helfen, ein besseres Kalibrie-

rungsergebnis zu erzielen.

Weiters soll versucht werden, die Ergebnisse der Kalibrierungsanwendung bei hohem Neigungswinkel der Kamera zu verbessern. Dies hilft auch bei Spiegelungen auf den Monitoren, da diese meist aus anderen Perspektiven weniger stark auftreten. Wären die Messergebnisse aus jedem Neigungswinkel gleich genau, könnten die Bilder immer aus jenem Winkel aufgenommen werden, bei dem die Spiegelungen am geringsten auftreten.

Quellenverzeichnis

Literatur

[1] Vipulkumar Chauhan und Manish Kayasth. „Augmented Reality Markers, It's Different Types, Criterion for Best Fiducially Marker and Necessary Requirements to Selecting Application Oriented Markers". *International Journal of Engineering Sciences & Research Technology* 4.1 (2015), S. 550–559 (siehe S. 5–8, 12).

[2] John C. Clarke, S. Carlsson und Andrew Zisserman. „Detecting and Tracking Linear Features Efficiently". In: *Proceedings of the British Machine Vision Conference*. (Edinburgh). Hrsg. von Bob Fisher und Manuel Trucco. Durham: BMVA Press, Sep. 1996, S. 8.1–8.10 (siehe S. 18).

[3] Mark Fiala. „ARTag, a Fiducial Marker System Using Digital Techniques". In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. (San Diego). Bd. 2. Washington: IEEE Computer Society, Juni 2005, S. 590–596 (siehe S. 10, 11, 13, 14).

[4] Mark Fiala. *ARTag, an Improved Marker System Based on ARToolKit*. Techn. Ber. 47166. Ottawa: National Research Council of Canada, 2004 (siehe S. 12–15).

[5] Mark Fiala. „Comparing ARTag and ARToolKit Plus Fiducial Marker Systems". In: *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and their Applications*. (Ottawa). New York: IEEE, Okt. 2005, S. 148–153 (siehe S. 10–15).

[6] Martin A. Fischler und Robert C. Bolles. „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Communications of the ACM* 24.6 (Juni 1981), S. 381–395 (siehe S. 22).

[7] Xiao-Shan Gao u. a. „Complete Solution Classification For the Perspective-Three-Point Problem". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.8 (2003), S. 930–943 (siehe S. 5).

[8] Martin Hirzer. *Marker Detection for Augmented Reality Applications*. Techn. Ber. ICG–TR–08/05. Graz: Institute for Computer Graphics und Vision, Graz University of Technology, Okt. 2008 (siehe S. 17, 18, 21).

[9] Christian Koch u. a. „Natural Markers for Augmented Reality-Based Indoor Navigation and Facility Maintenance“. *Automation in Construction* 48 (2014), S. 18–30 (siehe S. 6, 7, 9, 10).

[10] Ezio Malis und Manuel Vargas. *Deeper Understanding of the Homography Decomposition for Vision-Based Control*. Techn. Ber. RR-6303. Le Chesnay Cedex: French National Institute for Computer Science und Applied Mathematics, Sep. 2007 (siehe S. 32, 35).

[11] Paul Milgram und Fumio Kishino. „A Taxonomy of Mixed Reality Visual Displays“. *IEICE Transactions on Information and Systems* 77.12 (1994), S. 1321–1329 (siehe S. 8).

[12] Nobuyuki Otsu. „A Threshold Selection Method from Gray-Level Histograms“. *IEEE Transactions on Systems, Man and Cybernetics* 9.1 (1979), S. 62–66 (siehe S. 33).

[13] Jannick P. Rolland, Yohan Baillot und Alexei A. Goon. „A Survey of Tracking Technology for Virtual Environments“. In: *Fundamentals of Wearable Computers and Augmented Reality*. Hrsg. von Woodrow Barfield und Thomas Caudell. Mahwah: Lawrence Erlbaum Associates, 2001. Kap. 3, S. 67–112 (siehe S. 3).

[14] Irwin Sobel. „An Isotropic 3× 3 Image Gradient Operator“. In: *Machine Vision for Three-Dimensional Scenes*. Hrsg. von Herbert Freeman. New York: Academic Press, 1990, S. 376–379 (siehe S. 23).

[15] Daniel Wagner und Dieter Schmalstieg. „ARToolKitPlus for Pose Tracking on Mobile Devices“. In: *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*. (St. Lambrecht). Hrsg. von Michael Grabner und Helmut Grabner. Graz: Graz Technical University, Feb. 2007 (siehe S. 11, 15, 16).

[16] Zhengyou Zhang. „Camera Calibration“. In: *Emerging Topics in Computer Vision*. Hrsg. von Gerard Medioni und Sing Bing Kang. Upper Saddle River: Prentice Hall PTR, 2004. Kap. 2, S. 5–44 (siehe S. 35).

[17] Feng Zhou, Henry Been-Lirn Duh und Mark Billinghurst. „Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR“. In: *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. (Cambridge). Hrsg. von Mark A. Livingston, Oliver Bimber und Hideo Saito. Washington: IEEE, Sep. 2008, S. 193–202 (siehe S. 3).

Online-Quellen

[18] URL: https://de.wikipedia.org/wiki/Photogrammetrie (besucht am 19. 11. 2016) (siehe S. 8).

[19] URL: http://www.media.mit.edu/research/groups/1467/bokode-imperceptible-visual-tags-camera-based-interaction-distance (besucht am 20. 11. 2016) (siehe S. 10).

[20] OpenCV. *Camera Calibration and 3D Reconstruction*. URL: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (besucht am 02. 11. 2016) (siehe S. 37).

[21] Kyle Simek. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix*. URL: http://ksimek.github.io/2013/08/13/intrinsic/ (besucht am 02. 11. 2016) (siehe S. 36).