Fraunhofer
FOKUS

TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV - Elektrotechnik und Informatik
Fachgebiet DCAITI

Projektdokumentation
DCAITI Projekt

Konzeption und Implementierung einer Augmented Reality basierten Applikation für historische Gebäude und Baustellen

vorgelegt von: Herbert Potechius und Linh Kästner
Betreuer: Konstantin Klipp
eingereicht am: 17. August 2018

Eidesstattliche Erklärung

Wir, Herbert Potechius und Linh Kästner, versichern hiermit an Eides statt, dass wir unsere Projektdokumentation - *DCAITI Projekt* mit dem Thema

> *Konzeption und Implementierung einer Augmented Reality basierten Applikation für historische Gebäude und Baustellen*

selbständig und eigenhändig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Berlin, den 17. August 2018

_____

Herbert Potechius und Linh Kästner

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

1.1 Einleitung

Der stetige Fortschritt im Bereich der Informationstechnik setze in den vergangenen Jahren in nahezu allen Bereichen der Gesellschaft innovative Möglichkeiten in Gang. Das Konzept einer erweiterten Realität (Augmented Reality - AR) in der die reale Umgebung mittels virtuellen Instanzen erweitert wird, erlangt dabei, aufgrund seiner vielfältigen Einsatzmöglichkeiten, zunehmende Wichtigkeit. AR kann die Nutzerinteraktion erheblich verbessern und Aufgaben wie Wartung, Steuerung oder Kontrolle effizienter und einfacher gestalten. Durch die Visualisierung innerhalb der realen Umgebung trägt AR ferner zur Unterstützung des räumlichen Verständnisses bei und hat damit einhergehend positive Auswirkungen auf das Nutzererlebniss. Wesentliche Probleme beziehungsweise Herausforderung beim Einsatz von AR ist der Koordinatenabgleich zwischen Weltkoordinaten und dem verwendeten Darstellungsgerät - meist das Smartphone oder sogenannte Head Mounted Devices wie die Microsoft Hololens um die Illusion zu schaffen, dass Daten und Hologramme sich innerhalb der echten Welt befinden. Dazu wurden in den vergangenen Jahren vielfältige Forschung und Methoden entwickelt auf welche im späteren Verlauf noch eingegangen wird. Weitere Herausforderungen siedeln sich im Bereich der Computervision zur intelligenten Erkennung und Auswertung der Pixeldaten Computergrafik und der Bildverarbeitung, beispielsweise das Rendern von Objekten in die Szene oder die korrekte Darstellung der Objekte an.

1.2 Zielstellung

Ziel dieses Projektes war es eine AR basierte Applikation zu implementieren, welche es dem Nutzer erlaubt innerhalb der realen Umgebung, virtuelle Instanzen zu platzieren. Konkret soll es sich dabei um Gebäude und Gegenstände handeln, um somit eine Erweiterung von Baustellen oder Ruinen durch jene virtuellen Instanzen zu erreichen und dabei unterstützend bei der Konstruktion, Planung und Design von neuen Gebäuden zu dienen. Außerdem kann, beispielsweise durch die Erweiterung von Ruinen und der Platzierung von historischen Objekten ein interessantes Nutzererlebniss ermöglicht werden.

Kapitel 2

Theorie

Um ein optimales AR Erlebnis bieten zu können, müssen, wie bereits eingangs erwähnt, Aspekte aus verschiedenen Teildisziplinen miteinander verknüpft werden. Wichtige Disziplinen sind die Computervision sowie die Computergrafik. Außerdem ist der Abgleich er Koordinatensysteme von essentieller Bedeutung. Im folgenden wird auf die Theorie und Ansatzpunkte dieser Problemstellungen eingegangen.

2.1 Augmented Reality

kurze Erklärung was das ist hauptprobleme und aspekte bildverarbeitung rendering usw auch ein paar bilder zu anwendungsgebieten usw reinmachen

übergang koordinatenabgleich wichtigste

2.2 Die Detektion der Kamerapose im Raum

Wie bereits vorher erwähnt ist ein Kernaspekt um ein angemessenes AR Erlebnis zu ermöglichen, der sogenannte Koordinatenabgleich der Weltkoordinaten mit denen der Kamera. Grundsätzlich geht es darum die Position des Nutzers und dessen Sichtfeld und Blickrichtung innerhalb der Weltkoordinaten zu lokalisieren. Praktisch entspricht dies der Orientierung und Position der Kamera des AR Geräts (Smartphone, HMD, etc.). Wichtig hierbei ist, dass die Kamera individuell kalbibriert werden muss. Forschungen, welche z.T noch hochaktuell sind haben zahlreiche Algorithmen und Lösungsansätze hervorgebracht, wobei die wichtigsten Paradigmen im folgenden aufgelistet sind:

- **Visuelle Erkennungsmethoden**. Hier wird die Position der Kamera anhand von visuellen Informationen geschätzt. Das System tätigt Rückschlüsse auf Position der Kamera anhand von Informationen, welche es visuell mitbekommt. Nachteile dieser Methoden ergeben sich bei unbekannten Umgebungen, womit vorher erst genügend Daten gesammelt werden müssen um eine korrekte Poseschätzung zu erreichen. Der Markerbasierte Ansatz gehört zu dieser Art von Erkennung, welcher es durch vordefinierte Muster, jenes Problem löst. Dies wird im weiteren Verlauf detailliert erläutert. Weitere Methoden der visuellen Erkennung sind beispielsweise die Modellbasierten Erkennungsmethoden, welche einen Abgleich zwischen visuellen Daten mit vordefinierten Modellen (bspw. CAD Modellen) bewerkstelligen.

- **Sensorbasierte Erkennungsmethoden** Hier werden verschiedene, dem System verfügbare Sensordaten wie beispielsweise, Tiefendaten oder Lokalisierungsdaten bei der Positionsbestimmung verwendet. Dafür werden spezielle Geräte verwendet, welche diese zusätzlichen Informationen bereitstellen können, beispielsweise Gyroskope, GPS oder Tiefenkameras. Sensordaten können entweder die Position oder Orientierung des Nutzers wiedergeben. Die Kombination aus Position und Orientierung wird auch **Pose** genannt.

- **hybride Ansätze** Hybride Ansätze vereinen oben genannte Methoden um eine optimierte/genauere Schätzung zu erreichen Ein Ansatz beispielsweise die Kombination zwischen einem GPS Signal, welches die Position der Kamera vermitteln kann und visuellen Informationen, welche dann die Orientierung der Kamera festlegen.

In der Praxis hat sich unter den oben vorgestellen Methoden der Markerbasierte Ansätz als Kompromiss zwischen Genauigkeit und Einfachheit in der Implementierung als weit verbreitet herausgestellt. Durch die vordefinierten Markerinformationen wird erheblich Zeit und Rechenleistung bei der Erkennung eingespart. Viele der führenden AR Frameworks nutzen die Markerbasierte Erkennung (ARToolkit, ARTag, Aruco). Die vordefinierten Informationen beispielsweise eine MarkerID zu welchem bestimmte Objekte eingeblendet werden sollen sind weitere Vorteile des Ansatzes. Nachteile sind, dass Marker für einige Anwendungsfälle nicht geeignet sind, beispielsweise bei Anwendungen in gefährlichen Gebieten in welchem dann auf die Sensorbasierten bzw der Kombination aus visuellen Informationen mit zusätzlichen Senrdaten gesetzt wird.

Im Rahmen des Projektes wurde aufgrund obiger Erkenntnisse und der Empfehlung durch den Betreuer, der Markerbasierte Abgleich mittels Aruco genutzt, welche Teil der OpenCV Bibliothek ist. Auf diesen wird im folgenden detailliert eingegangen.

2.3 Die markerbasierten Erkennung mit Aruco

Trotz der erwähnten Vorteile der Marker gibt es einige Aspekte zu beachten. Beispielsweise können ungünstige Lichtverhältnisse oder schlechter Kontrast dazu führen dass der Marker nicht erkannt wird. Aufgrund von Farbveränderungen, welche durch ungünstiges Licht auftreten können werden grundsätzlich nur schwarz-weiss Marker benutzt. Optimalerweise reichen 4 bekannte Punkte aus um die Pose des Markers zu erkennen. Daher eignen sich Quadratische Formen als Marker am besten. Es gibt zahlreiche bekannte Bibliotheken, welche diese Art von Markern verwendet. Eine davon ist die Aruco Bibliothek, welche teil des OpenCV Frameworks ist, welches im späteren Verlauf noch erläutert wird. Aruco benutzt Schachbrettmuster um individuelle Marker zu erzeugen. Diese Marker sind durch Ihre individuelen Muster mit einer ID versehen. Dies ist nützlich um beispielsweise bestimmte Objekte mit einem bestimmten Marker zu versehen.

Das Aruco Modul ist Teil der Aruco Bibliothek und wurde von Rafael Munoz und Sergio Garrido entwickelt. Für die Markererkennung und idendifikation nutzt es spezielle Muster, welche dekodiert werden. Der Marker besteht aus einem schwarzen Rand sowie einem Muster im inneren, welches einer binären Matrix gleichkommt. Dabei bestimmt die Größe des Markers auch die Bitzahl der Innenmatrix. Entsprechend hat ein 4cm x 4 cm großer Marker eine 16 Bit Matrix. Auf diese Matrix wird im späteren Verlauf noch eingegangen. In Abbildung 2.1 sind Beispiele dieser Marker zu sehen.

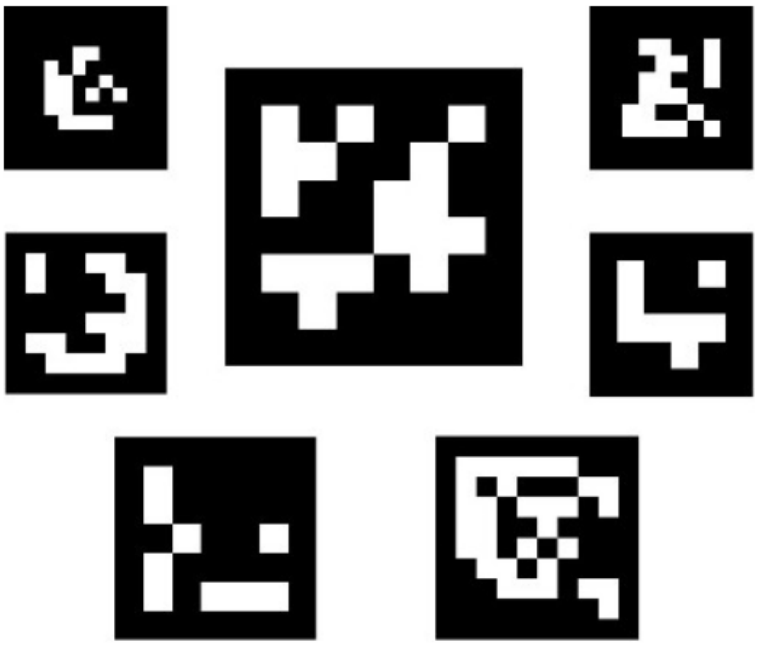


Abbildung 2.1: Koordinatensysteme mit Freiheitsgraden [1] (aus opencv.com)

Im folgenden werden die einzelnen Schritte bei der Markererkennung besprochen.

- Umwandlung des Bildes in ein Greyscale Bild, welches die Intensitätswerte beinhaltet.
- Erkennung der Kanten und Ecken im Bild und Bestimmung potentieller Markerkandidaten
- Herausfilterung und Verbesserung
- Dekodierung der Marker durch das Auswerten des Innenmusters des Markes.
- Berechnung der Position und Orientierung des Markers (Posebestimmung)

Im folgenden wird auf die wichtigsten Punkte genauer eingegangen.

Greyscaling und Tresholding

Aus dem aktuellen Bild der Kamera muss als erstes ein Grayscale Bild, also ein Bild aus Intensitätswerten bestimmt werden, da weitere Operationen auf diese Werte aufbauen. Um aus diesem Bild die Konturen zu erkennen, wird es 'getresholdet'. Ein für einen Bespielmarker generiertes Treshold Bild ist in Abb. zu sehen.

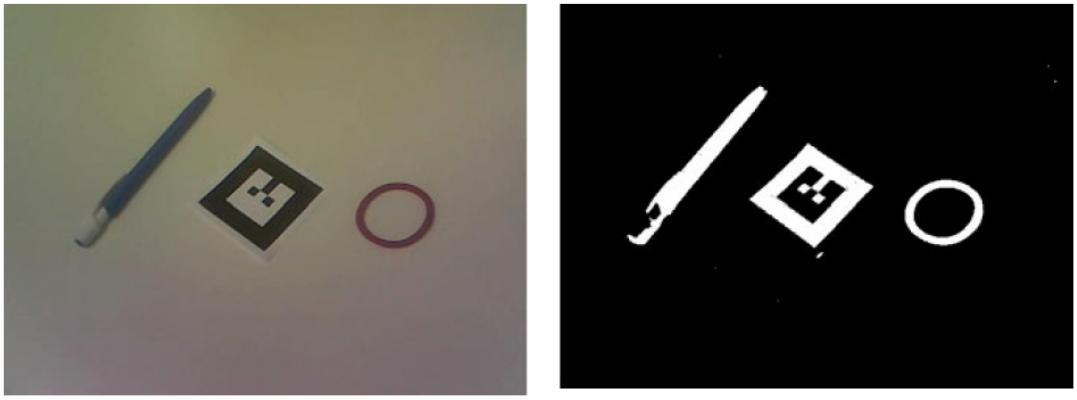


Abbildung 2.2: links: Originalbild, rechts: Treshold Bild [1] (aus grundquelle)

Der Treshold Ansatz erweisst sich als stabil gegen mögliche Helligkeitsschwankungen. Nachdem 'getresholdet' wurde liegt ein binäres Bild, bestehend aus Hintergrund und Objekten vor, aus denen dann die Konturen erkannt werden können. Dafür werden die Kanten aller potentiellen Marker im Bild markiert und geprüft ob genau 4 gerade Linien erkannt wurden. Sind diese Vorraussetzungen erfüllt, werden die Ecken des potentiellen Markers analysiert. Wichtig ist, dass davor die Verzerrung durch die inverse Verzerrungsfunktion, welche beim Kalibrieren der Kamera gewonnen wird, vorgenommen wird.

Filterung und Verbesserung

Nachdem Tresholding wurden Konturen, welche auf einen potentiellen Marker hindeuten erkannt. Jedoch sind nicht alle erkannten Konturen tatsächlich Marker. Daher ist der Filterungschritt ein essentieller Schritt um nur tatsächliche Marker zu berücksichtigen. Dabei werden alle jene Konturen herausgefiltert, welche offentsichtlich keine Marker sind oder zu nah beieinander liegen. Dafür existieren in Aruco diverse Funktionen und Parameter um die Filterung individuell strikt zu gestalten. Wichtig ist es dabei die Markerparameter, wie Markergröße oder Maximale Distanz, welche zwei Marker zueinander besitzen dürfen vorher zu definieren um die Filterung effizient zu gestalten.

Markerdekodierung und Identifikation

Nachdem die Markerkandidaten detektiert wurden kann die Detektierung des Markers beginnen um zu prüfen ob es sich tatsächlich um einen Aruco basierten Marker handelt. Dazu wird das Bitmuster im Inneren des Markers mittels dem sogenannten Otsu Algortihmus extrahiert um schwarze und weisse Bildpunkte zu differenzieren. Anschließend wird das Bild mittels eines Gitters aufgeteilt, welches erlaubt jene einzelnen schwarz-weiss Punkte exakt auszumachen. Dies ist in Abb. 2.3 zu erkennen.

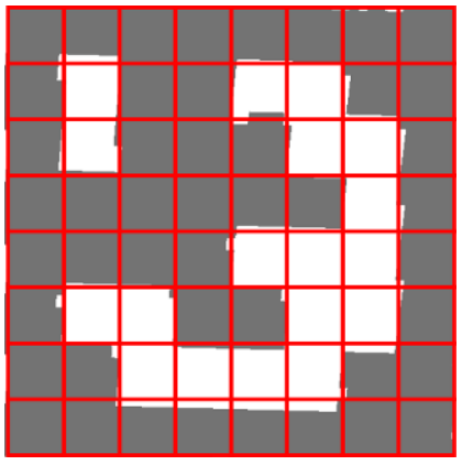


Abbildung 2.3: Bitmuster im Gitter [1] (OpenCV.org)

Anschließend werden weisse und schwarze Bits gezählt. Dabei wird für jede Zelle nur die Mitte beachtet, um somit Störungen zu vermeiden, da es auch Zellen gibt in denen beide Farbein existieren und somit nicht eindeutig ist um was für eine Zelle es sich handelt. Dies wird in Abbildung 2.4 deutlich.

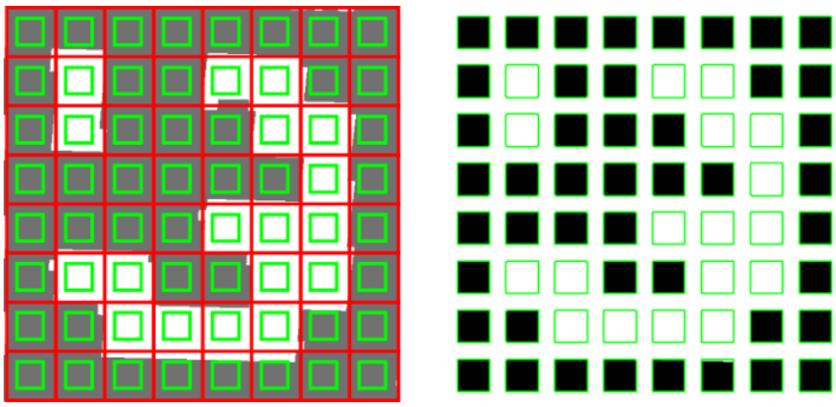


Abbildung 2.4: verbessertes Bitmuster im Gitter [1] (OpenCV.org)

Nachdem die Bits extrahiert und gezählt wurden, können diese mit dem Dictionary verglichen werden, in welchem sich alle vordefinierten Marker befinden.

2.4 Der Koordinatenabgleich

Ist der Marker erkannt, kann der eigentliche Koordinatenabgleich zwischen der Kamera, welche den Marker detektiert hat und dem Marker selbst stattfinden. Dies wird ist im Fachjargon auch unter dem Perspective-n-Point Problem bekannt. Zuvor muss jedoch erst eine Kamerakalibrierung stattfinden, in welchem die Kameramatrix, bestehend aus intrinsischen Parametern, wie Brennweite,... und Verzerrungselemente der Kamera gefunden werden. OpenCV und Aruco bieten dazu einige Funktionen an mit denen die Kamera kalibriert werden kann. Hauptziel des Koordinatenabgleiches ist die Bestimmung der Position der Kamera im Raum basierend auf der Markerposition. In Abbildung 2.5 ist eine Übersicht der beiden Koordinatensysteme veranschaulicht.

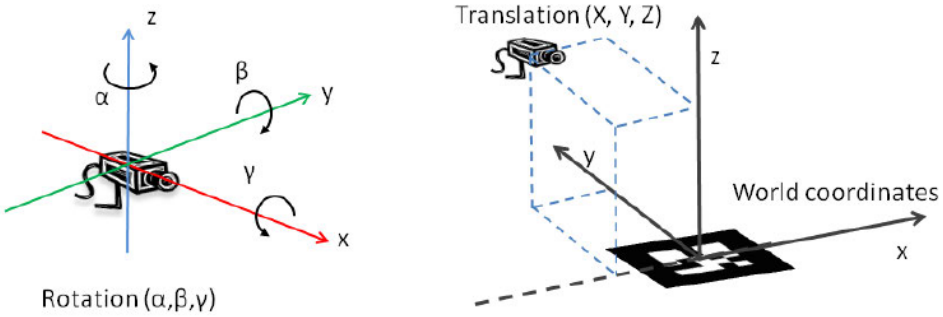


Abbildung 2.5: Koordinatensysteme mit Freiheitsgraden [1] (aus grundquelle)

Position und Orientierung der Kamera erzeugen somit 6 Freiheitsgrade. Die Koordinaten (x,y,z) beshreiben dabei die Translation und die Winkel (α, β, γ) die Rotation. Um diese 6 Freiheitsgrade zu finden, werden die zuvor detektierten Punkte des Markers verwendet, welche eine Transformationsmatrix T liefern, welche den Koordinatenabgleich ermöglicht. Diese ist definiert als

$$x = T \cdot X$$

Oder umgeschrieben

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}.$$

Diese Matrix beinhaltet die Translation und Rotation der Kamera in Relation zum Marker basierend auf den gefundenen Punkten des zuvor detektierten Markers und wird daher auch als Posematrix bezeichnet. Durch Anwenden dieser Transformation geschieht der Abgleich zwischen Welt und Kamera und es wird eine virtuelle Kamera auf den Marker gesetzt. Dies ermöglicht das positionsgenaue Rendern der Objekte auf oder in Relation zur Markerpose. Dafür muss eine weitere Transformation mit der Kameramatrix getätigt werden, welche dann dann die 2D Szene im sichtbaren Bild (Image Plane) liefert. Dies ist in Abbildung 2.6 veranschaulicht.

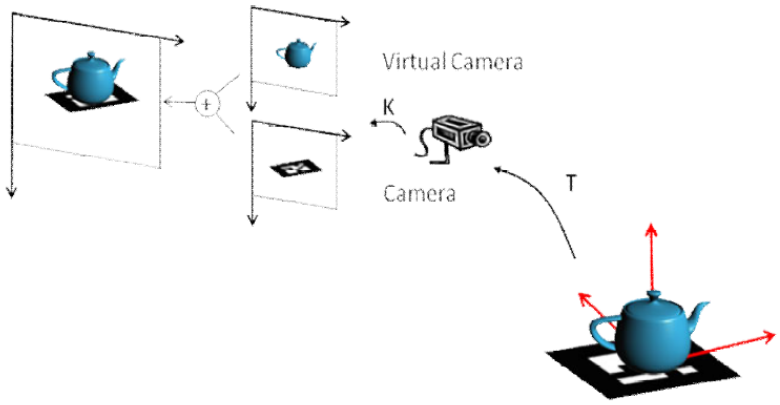


Abbildung 2.6: Koordinatensysteme mit Freiheitsgraden [1] (aus grundquelle)

Soll das Objekt in Relation zum Marker platziert werden, muss eine weitere Transformationsmatrix T_object angegeben werden in welcher Translation und Rotation definiert sind (Abbildung 2.7) .

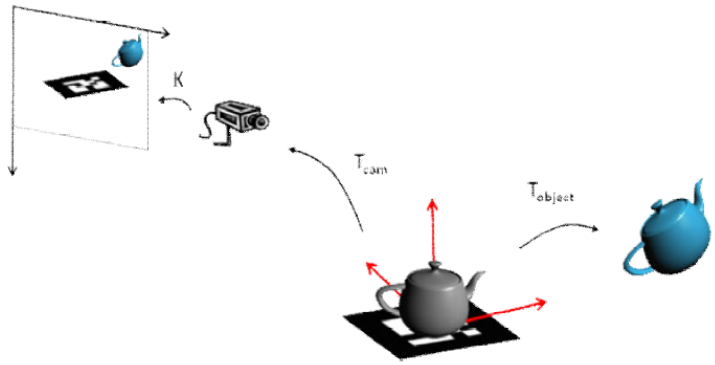


Abbildung 2.7: Koordinatensysteme mit Freiheitsgraden [1] (aus grundquelle)

Im Aruco wird eine erfolgreiche Detektion meist mit einem gerenderten Koordinatensystem in die Szene gezeigt wie dies in Abbildung 2.8 zu sehen ist.

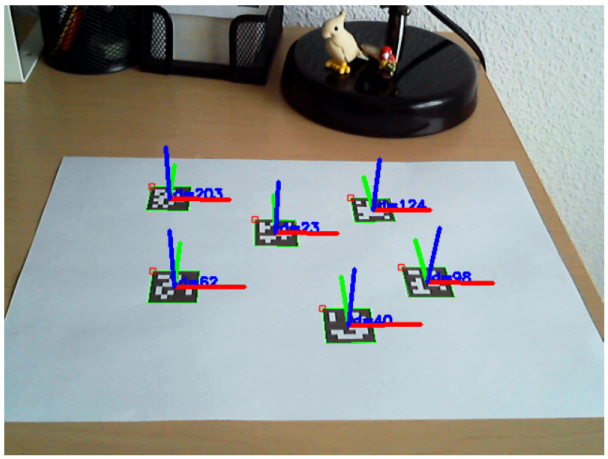


Abbildung 2.8: Erfolgreiche Markererkennung und Objektplatzierung mit Aruco [1] (aus aus opencv.com)

2.5 Chromaticity-Bild

Sind Bilder invariant gegenüber Helligkeitsveränderungen, kann ein und das selbe Objekt bei unterschiedlichen Helligkeiten besser identifiziert werden. Um diese Invarianz zu erreichen, wird das RGB-Bild in den rgb-Farbraum überführt. Die Transformation erfolgt nach folgenden Gleichungen:

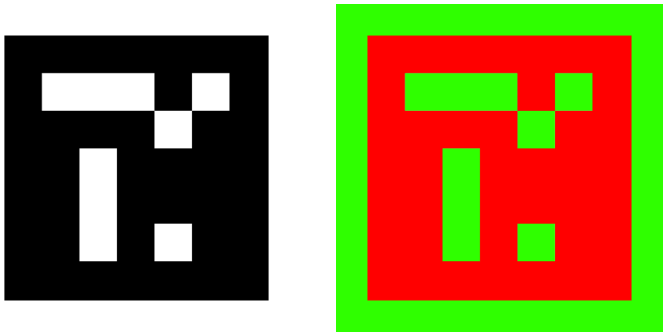
$$r = \frac{R}{R+G+B} \qquad g = \frac{G}{R+G+B} \qquad b = \frac{B}{R+G+B}$$

Die Farbe des Pixels definiert sich nicht mehr durch die Intensität der einzelnen Farbkanäle, sondern durch das Verhältnis der Farbkanäle untereinander. Somit bilden z.B. die beiden Farbwerte (255,0,0) und (150,0,0) aus dem RGB-Farbraum auf den selben Farbwert (1,0,0) im rgb-Farbraum ab.

(a) RGB-Farbraum (b) rgb-Farbraum

Abbildung 2.9: Vergleich: Originalaufnahme mit Chromaticity-Bild

Damit dieses Verfahren zur Markererkennung verwendet werden kann, müssen die Marker farblich angepasst werden, sodass Unterschiede in der Farbart erkennbar sind.



(a) schwarz-weiß Marker (b) rot-grün Marker

Abbildung 2.10: Anpassung der Markerfarben

Kapitel 3

Setup

3.1 Verwendete Software und Frameworks

Im folgenden soll auf die benutze Software zur Realisierung obig genanter Aspekte eingegangen werden.

3.1.1 OpenCV und Aruco

Für die Markererkennung im Raum wurde die Aruco Bibliothek benutzt, welche auf der markerbasierten Erkennung basiert, wie diese bereits in der Theorie besprochen wurde. Die Kernfunktionalitäten der Bibliothek sind im folgenden aufgelistet:

3.1.2 OpenGL

OpenGL ist eine von .. bereitgestellt Bibliothek, welche Funktionen für das Rendern von 3D Objekten bereitstellt. Sie ist gut mit OpenCV integrierbar.

3.1.3 Android Studio

Zur Entwicklung der Applikation wurde Android Studio benutzt und somit als Zielgruppe ausschließlich Android Geräte. Dies hat den Grund einer offeneren Plattform und der leichteren Einbindung von externen Bibliotheken wie OpenCV oder OpenGl.

3.1.4 Integration von OpenCV und OpenGL in Android Studio

....

Kapitel 4

Implementierung

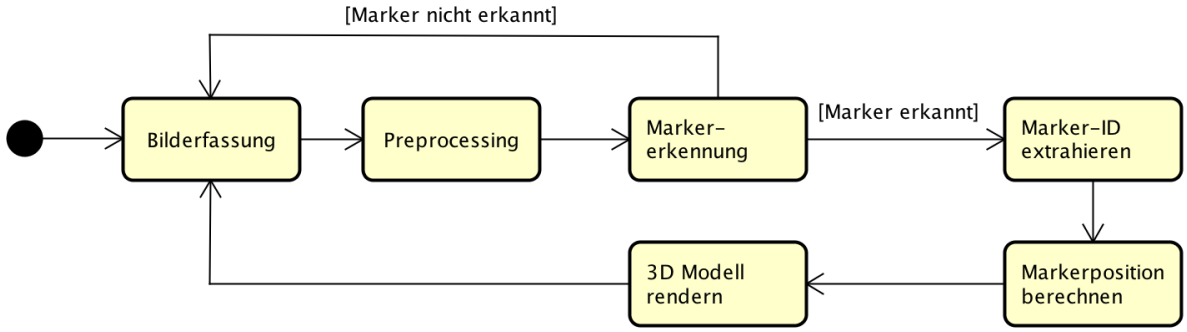


Abbildung 4.1: Grobe Darstellung der Arbeitsschritte

4.0.1 Bilderfassung

Durch Instanziierung der von Android bereitgestellten Klasse *JavaCameraView* kann auf die Aufnahmen der Kameras zugegriffen werden. Diese werden für eine spätere Analyse gespeichert.

Funktion	Beschreibung
init_camera()	Initialisiert die Rück-Kamera des Smartphones mit einer vordefinierten Auflösung.
onCameraFrame(CvCameraViewFrame inputFrame)	Wird pro Frame aufgerufen und liefert das Kamerabild als *CvCameraViewFrame*

Tabelle 4.1: Funktionen zur Bilderfassung

4.0.2 Preprocessing

Das als *CvCameraViewFrame* vorliegende Bild wird zur weiteren Bearbeitung in eine Matrix gespeichert. Aus performancetechnischen Gründen wird das Preprocessing in C++ ausgeführt. Dieser Schritt sorgt dafür, dass der Marker unter verschiedensten Lichtverhältnissen erkannt wird.

Funktion	Beschreibung
analyzeMarker(...)	Ruft die C++-Funktion auf, die die eigentliche Preprocessing-Funktionalität beinhaltet
preprocessing(Mat mRgb, Mat mChrom)	Die Matrix *mRgb* wird in ein Chromaticity-Bild transformiert und in *mChrom* gespeichert.

Tabelle 4.2: Funktionen zum Preprocessing

4.0.3 Markeranalyse

Die Markeranalyse enthält die Schritte *Markererkennung*, *Marker_ID extrahieren* und *Markerposition berechnen*. Diese werden auch aus performancetechnischen Gründen in C++ ausgeführt. Alle drei Schritte werden von der Funktion *detect*(...) aus der Aruco-Library durchgeführt.

Funktion	Beschreibung
detect(Mat input, vector<Marker> detectedMarkers, CameraParameters camParams, float markerSize)	Sucht innerhalb des Bildes *input* nach Markern und speichert diese in *detectedMarkers*. Anhand der Kameraparameter *camParams* und der Markergröße *markerSize* kann die Orientierung des Markers bezüglich der Kamera berechnet werden. Wie in der Theorie beschrieben wird hier die Marker-ID bestimmt.

Tabelle 4.3: Funktionen zur Markeranalyse

4.0.4 Rendering

Durch die Initialisierung des Renderers wird pro Frame die Orientierung des Markers verwendet um das 3D-Modell zu positionieren.

Funktion	Beschreibung
initOpenGL()	Initialisiert den Renderer und sorgt dafür, dass die onDrawFrame-Methode pro Frame ausgeführt wird.
onDrawFrame(GL10 gl)	Wendet die Transformationsmatrizen des Markers auf das 3D-Modell an und projiziert dieses auf die Bildebene.

Tabelle 4.4: Funktionen zum Rendern des 3D-Modells

Kapitel 5

Zusammenfassung

Insgesamt wurden die Konzepte der Theorie verstanden und im Projekt erfolgreich angewandt. Damit konnten die anfangs definierten Ziele für das Projekt alle erreicht werden. Es konnte eine Web basierte Applikation zur Visualisierung der Kommunikation des betrachteten Prozesses implementiert werden, welche auch die Möglichkeit bietet, alle Netzwerkkomponenten zu sehen. Zusätzlich dazu wurde eine Filter-Funktion implementiert, um die Kommunikation aus Sicht einer Komponente anzuzeigen. Entsprechend wurde für diesen Fall auch das Design des Chat Raums angepasst. Die Funktionalität der Applikation konnte im Labor erfolgreich getestet werden. Weiterhin konnte für Testzwecke eine emulierte Maschinenumgebung mithilfe der verfügbaren Demo Applikationen geschaffen werden, welche es erlaubt auch außerhalb des Labors, die Funktionalität zu überprüfen, womit gleichzeitig die Grundlage für weitere Projekte und Implementierungen geschaffen wurde.

Kapitel 6

Ausblick

Die existierende App bietet viel Freiraum für zukünfitge Erweiterungen. Eine wichtige Erweiterung ist das Füllen der internen Datenbank mit neuen 3D Modellen. Dazu können verschiedene MarkerIds verschiedenen Objekten zugeordnet werden. Ein weiterer Punkt ist eine erweiterte Benutzerinteraktion durch Buttons. Beispielsweise lass sich eine Map in die App integrieren mit der alle in der Stadt verfügbaren Marker sichtbar sind. Dafür ist die Google Maps API zu empfehlen welche sichproblemlos in Android Studio einbetten lässt. Von seiten des Preprocessings der Bilder könnten weitere Algorithmen probiert werden ,welche eine noch robustere Markererkennung bei schlechten Verhältnissen bietet. Hier wären beispielsweise der ... oder .. auszuprobieren.

....

Literaturverzeichnis

[1] ZVEI SG Modelle und Standards

[2] openmtc.org

[3] w3.org/TR/jsonld

[4] devcentral.f5.com

[5] Bildquellen Setup und Konzeption the-mtc.org iot.do fotolia.com

[6] Microsoft Office 365
https://www.microsoft.com/de-de/store/d/office-365-home

[7] Flat Chat Widget UI
https://designshack.net/design/flat-chat-widget-ui/

[8] Dock
https://zurb.com/playground/osx-dock

[9] Wallpaper/Icons
Wood: http://www.kinyu-z.net/data/wallpapers/147/1219741.jpg
Wood2: http://getwallpapers.com/wallpaper/full/0/9/9/404732.jpg
Up: https://images2.alphacoders.com/437/thumb-350-437561.jpg
Farbverläufe: https://uigradients.com
Icons: https://www.shareicon.net/

[10] jQuery
http://jquery.com/

[11] Ablaufpläne
https://www.draw. io/