

## Einblendung von kontextsensitiven Inhalten auf der Google Glass

### Bachelorarbeit

am Fachgebiet Informationsmanagement und Wirtschaftsinformatik,  
Universität Osnabrück

zur Erlangung des Grades  
Bachelor of Science (B. Sc.)  
im Studiengang  
Wirtschaftsinformatik

**Themensteller:** Prof. Dr. Oliver Thomas  
**Betreuer:** Dirk Metzger, M.Sc. with Honors

**Vorgelegt von:** Jannik Hoffjann  
Jahnplatz 6 W-169  
49080 Osnabrück

Matrikelnummer: 945592  
E-Mail-Adresse: [jhoffjann@uni-osnabrueck.de](mailto:jhoffjann@uni-osnabrueck.de)

**Abgabetermin:** 2015-01-22

## Zusammenfassung

In der Arbeit wurden die Möglichkeiten der Einblendung von kontextsensitiven Inhalten als Lernhilfsmittel auf einem in das Sichtfeld integrierten Gerät am Beispiel der Google Glass erprobt. Dabei wird nach Definition von Kontextsensitivität und Erörterung der verschiedenen Möglichkeiten das Gerät und der mitgelieferte Software vorgestellt.

Nach Abwägung der einzelnen Möglichkeiten wurde eine der Methoden der kontextsensitiven Inhaltsgewinnung beispielhaft auf der Google Glass implementiert. Es wurde getestet, inwieweit sich das Medium Google Glass als agierendes Objekt eignet und wo durch die gegebene Hard- und Software eventuelle Grenzen entstehen.

Dies wurde durch eine Implementation auf Basis der Computer Vision Bibliothek OpenCV und der von ihr zur Verfügung gestellten Methoden beispielhaft gezeigt. Mithilfe vielfältiger Alternativen des Matchings bietet diese eine Wiedererkennung und Auswertung von Grafiken, welche die Umsetzung einer kontextsensitiven Anwendung auf der Google Glass ermöglicht.

Durch diese abschließende Implementation und eine Auswertung der Ergebnisse wird ein erster erfolgreicher Versuch der Einblendung von kontextsensitiven Inhalten durch Keypointerkennung und -matching anhand von Realobjekten auf der Google Glass erbracht. Die zeigt das Potenzial der Plattform Google Glass und mit ihr die Möglichkeiten einer Lernunterstützung durch Geräte der Augmented Reality.

## Inhaltsverzeichnis

Zusammenfassung .....	II
Abbildungsverzeichnis .....	IV
Tabellenverzeichnis.....	V
Codeverzeichnis .....	VI
Abkürzungsverzeichnis .....	VII
1 Einleitung.....	1
2 Kontextsensitivität .....	3
2.1 Definition.....	3
2.2 Möglichkeiten der Kontextsensitivität .....	4
2.2.1 QR Codes .....	5
2.2.2 Location-Based Services .....	6
2.2.3 Objekt- und Bilderkennung.....	7
3 Google Glass.....	9
3.1 Die Google Glass als Vertreter der Augmented Reality .....	9
3.2 Spezifikationen und Besonderheiten der Google Glass .....	10
3.2.1 Hardwarespezifikationen .....	10
3.2.2 Softwarespezifikationen.....	11
4 Einblendung von kontextsensitiven Inhalten auf der Google Glass.....	13
4.1 Idee und Funktionsweise der kontextsensitiven Applikation .....	13
4.2 Vorstellung von OpenCV und der verwendeten Algorithmen .....	15
4.2.1 OpenCV und JavaCPP .....	15
4.2.2 SURF und Fast Approximate Nearest Neighbor Matching .....	16
5 Umsetzung einer kontextsensitiven Applikation mit OpenCV .....	20
5.1 Vorstellung der Implementation und ihrer Komponenten.....	20
5.2 Glass Client .....	22
5.3 OCV Server .....	25
5.3.1 Hinzufügen eines neuen Objekts.....	25
5.3.2 Analyse eines gesendeten Bildes .....	27
5.4 Auswertung der Applikation .....	29
6 Fallstudie .....	32
6.1 Einführung in die Fallstudie .....	32
6.2 Beispieldurchführung .....	33
7 Fazit und Ausblick .....	38
Literaturverzeichnis.....	40

## Abbildungsverzeichnis

Abb. 2.1	QR Code für den Titel dieser Arbeit .....	6
Abb. 2.2	Foursquare auf iOS 8.1.1.....	7
Abb. 3.1	Beschriftete Google Glass (in Anlehnung an Feng et al. 2014, S. 3070).....	10
Abb. 4.1	Darstellung der Funktionsweise der kontextsensitiven Applikation .....	13
Abb. 4.2	SURF-Keypointerkennung auf einem Logo.....	16
Abb. 4.3	SURF-Keypointerkennung auf einem Foto.....	17
Abb. 4.4	Neares Neighbor Matching der beiden Bilder.....	18
Abb. 4.5	Filterung des Matchings .....	18
Abb. 5.1	UML-Darstellung des Glass Clients.....	22
Abb. 5.2	UML-Darstellung des OCV Servers .....	25
Abb. 5.3	Ein Bild mit hoher Fehleranfälligkeit.....	30
Abb. 6.1	Die Bilder der auf dem Server hinterlegten Objekte .....	32
Abb. 6.2	Der Startbildschirm der Google Glass.....	33
Abb. 6.3	Die beiden Startmöglichkeiten der Applikation .....	34
Abb. 6.4	Der Standardbildschirm der Applikation.....	34
Abb. 6.5	Bestätigung des Fotografierbefehls .....	35
Abb. 6.6	Vorschaubild mit Akzeptanzanfrage .....	35
Abb. 6.7	Die beiden Ladebildschirme der Applikation.....	36
Abb. 6.8	Das Ergebnis der Anfrage .....	36
Abb. 6.9	Das negative Ergebnis einer anderen Anfrage .....	37

**Tabellenverzeichnis**

Tabelle 2.1 Häufig genutzte physische Sensoren (in Anlehnung an (Baldauf et al. 2007, S. 266) ) .....	4
Tabelle 5.1 Übersicht der genutzten Komponenten .....	20

## Codeverzeichnis

Code 5.1	Asynchroner Uploadprozess .....	23
Code 5.2	Erstellung der Anfrage und Auswertung der Antwort.....	24
Code 5.3	Aktualisierung der Benutzeroberfläche .....	24
Code 5.4	Extrahierung des ersten Absatzes eines Wikipedia-Artikels .....	26
Code 5.5	Keypointerkennung und Deskriptorextraktion .....	26
Code 5.6	Serialisierung einer Deskriptormatrix.....	27
Code 5.7	Beispiel eines abgespeicherten Objekts .....	27
Code 5.8	Matching der Deskriptoren und Filtern der Matches .....	28
Code 5.9	Beispiel einer Antwort des OCV Servers .....	29

## Abkürzungsverzeichnis

Abb.	Abbildung
aGPS	assisted Global Positioning System
API	Application Programming Interface
BoF	Bag of Features
BoW	Bag of Words
BRISK	Binary Robust Invariant Scalable Keypoints
bspw.	beispielsweise
bzw.	beziehungsweise
ca.	circa
engl.	englisch
FREAK	Fast Retina Keypoint
GB	Gigabyte
GDK	Glass Development Kit
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HUD	Head-up Display
JSON	JavaScript Object Notation
Mhz	Mega Hertz
OCR	Optical Character Recognition
OCV-Server	OpenCV-Server
OpenCV	Open Computer Vision
PDA	Personal Digital Assistant

QR	Quick Response
REST	Representational State Transfer
SDK	Software Development Kit
SIFT	Scale Invariant Feature Transform
SURF	Speeded-Up Robust Features
Tablet	Tablet Personal Computer
UMPC	Ultra Mobile PC
Wh	Wattstunden
XML	Extensible Markup Language
z.B.	zum Beispiel

## 1 Einleitung

Die Nutzung von Geräten der *Virtual Reality* (VR) und der *Augmented Reality* (AR) hat in den vergangenen zwei bis drei Jahren mit der Google Glass und der Oculus Rift sowie den beiden hinter ihnen stehenden Großunternehmen Google und Facebook einen Aufschwung erhalten.

Dabei hat insbesondere die Google Glass mit ihrer leichten Bauweise und Ungebundenheit zu nahestehenden Computern die Möglichkeit, den Bereich des Ubiquitous Computing zu verändern. Anders als Smartphones, Tablets oder konkurrierende Wearables ist die Google Glass durchgängig in das Sichtfeld des Trägers integriert. So bieten sich neue Möglichkeiten der Nutzerinteraktion, die den Gedanken des anywhere und anytime auf eine neue Ebene bringen könnten, da die benötigte Interaktion durch den Nutzer minimiert wird.

Anders als die Oculus Rift wurde die Google Glass nicht zum Anzeigen virtueller Welten bzw. der virtuellen Darstellung realer Umgebungen entwickelt. Vielmehr bietet sie die Möglichkeit, ähnlich einem Interface aus Computerspielen, sich über die Wahrnehmung des Nutzers zu legen und diese mit kontextsensitiver Information anzureichern.

Diese Erweiterung in der Wahrnehmung bietet zahlreiche Möglichkeiten. Anders als bei bekannten mobilen Geräten bietet die AR-Brille die Chance, dem Nutzer zusätzliche Informationen zu dem von ihm Betrachteten zu liefern, ohne dabei seine Handlung zu unterbrechen. Es wäre also zum Beispiel möglich, für den Arbeitsfluss dringend benötigte Informationen zu integrieren, ohne dabei die Arbeit unterbrechen zu müssen.

Diese Arbeit ist ein Teil des GLASSROOM Projekts, das vom Lehrstuhl für Informationsmanagement und Wirtschaftsinformatik (IMWI) der Universität Osnabrück in Kooperation mit namhaften Partnern wie der AMAZONEN-Werke H. Dreyer GmbH & Co. KG, dem Fraunhofer-Institut für Arbeitswirtschaft und Organisation oder der Universität des Saarlandes durchgeführt wird. Im Rahmen dieses Projekts sollen die Möglichkeiten der Nutzung von Brillen der *Augmented Reality* (AR) und *Virtual Reality* (VR) zur Schulung und Nutzung im Anlagen- und Maschinenbau erforscht werden. Bisher wurden die konkreten technischen Fähigkeiten zur Ausführung einer Tätigkeit in diesem Bereich vor allem durch Methoden aus dem Bereich *eLearning* und *Blended Learning* oder durch konkretes Training vor Ort vermittelt. Dieser Schritt soll für die Lernenden vereinfacht und gezielt um kontextsensitive Anwendungen der AR und VR erweitert werden.

*Inwieweit ist eine solche kontextsensitive Erweiterung mit der Google Glass möglich?*

Diese Forschungsfrage soll am Beispiel des heute am weitesten verbreiteten und bekanntesten Gerät der Augmented Reality beantwortet werden.

Dazu werden in dieser Arbeit nach einer Definition von Kontextsensitivität die Google Glass als Vertreter der Augmented Reality vorgestellt und in die Funktionsweise sowie die Idee der implementierten Applikation eingeführt. Die Implementation dieser Applikation wird daraufhin dargelegt und einige Schlüsselpunkte der Programmierung werden erklärt. Im Anschluss soll eine kurze Fallstudie mit einer beispielhaften Anwendung der implementierten Applikation durchgeführt werden. Damit wird ein Beitrag zur Implementierung umfangreicherer kontextsensitiver Applikationen zum Augmented Reality gestützten Lernen und Arbeiten geleistet. Die Arbeit schließt mit einem Fazit und gibt einen Ausblick auf die weiteren Forschungsschwerpunkte.

## 2 Kontextsensitivität

### 2.1 Definition

*Kontextsensitivität (engl. Context Awareness)* ist ein in der Wissenschaft langjährig diskutierter Begriff, der erstmals von Schilit und Theimer (1994, S. 23) erwähnt, aber in seinen Grundzügen bereits in den frühen 90er Jahren von Want et al. (1992) beschrieben wurde. Die Autoren entwickelten damals ein System, mit dem zur Koordination einer Belegschaft der Aufenthaltsort der einzelnen Mitarbeiter ermittelt wurde. Dafür wurden die Mitarbeiter mit *Active Badges* ausgestattet und mithilfe dieser geortet. Mit dieser Information konnten die Abläufe verbessert werden, um zum Beispiel das Telefonystem zu steuern (Want et al. 1992, S. 92–94).

Um aber zu einer Definition für Kontextsensitivität zu kommen, ist es notwendig, zunächst den Kontext-Begriff zu erläutern. Die heute in der Wissenschaft weitgehenden anerkannten Definitionen für Kontext und Kontextsensitivität kommen von Abowd et al. (1999). Sie definieren Kontext wie folgt:

*„[Context is] any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves.“* (1999, S. 3)

Mit dieser Definition bestimmten die Autoren den Kontext nicht nur als Information, die für den Nutzer in seiner Interaktion relevant ist, sondern lieferten zugleich auch eine Definition für kontextsensitive Systeme:

*„A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.“* (Abowd et al. 1999, S. 6)

Zwar gab es auch später Versuche, kontextsensitive Systeme zu definieren (Dey 2001, S. 5). Die meisten wissenschaftlichen Arbeiten zu dem Thema bauen aber auf der oben genannten Definition auf (Dey et al. 1999, S. 2; Lee et al. 2010, S. 1; Baldauf et al. 2007, S. 264; Perera et al. 2014, S. 414). Daher soll diese soll die Definition von Abowd et al. genutzt werden.

Heute wird die Kontextsensitivität als Unterfeld des Ubiquitous (bzw. Pervasive) Computing gesehen (Baldauf et al. 2007, S. 263–264; Bellavista et al. 2012, S. 2; Perera et al. 2014, S. 414). Kontextsensitivität nimmt hier eine wichtige Rolle ein, da die durch

Sensoren gewonnene Datenmenge in den mobilen Endgeräten stetig steigt und die Kontextsensitivität als Chance gesehen wird, die relevanten Informationen hieraus zu gewinnen (Perera et al. 2014, S. 414). Indulska und Sutton gehen sogar soweit, Kontextsensitivität als die Voraussetzung für ein „anywhere, anytime“ Computing zu sehen ( 2003, S. 1),

## 2.2 Möglichkeiten der Kontextsensitivität

Bei näherer Betrachtung der einzelnen Möglichkeiten von kontextsensitiven Systemen fällt auf, dass es sich bei Kontextsensitivität um ein enorm vielschichtiges Thema handelt.

Baldauf et al. ( 2007) und Perera et al. ( 2014) haben eine große Zahl wissenschaftlicher Arbeiten der letzten zwei Jahrzehnte (1990-2014) analysiert und dabei eine Vielzahl von Kategorisierungsmöglichkeiten für kontextsensitive Systeme und Applikationen festgehalten. Da eine Eingrenzung und Betrachtung aller was? den Rahmen dieser Arbeit deutlich sprengen würde, wird der Fokus im Folgenden auf zwei Schwerpunkte gelegt.

<i>Kontextart</i>	<i>Verfügbare Sensoren</i>
Licht	Fotodioden, Farbsensoren, Infrarot und UV-Sensoren etc.
Visueller Kontext	Verschiedene Kameras
Audio	Mikrofone
Bewegung, Beschleunigung	Quecksilberschalter, Neigungssensoren, Beschleunigungssensoren, Bewegungssensoren, Magnetfelder
Ort	Freiluft: Global Positioning System (GPS), assisted GPS, Global System for Mobile Communications (GSM); Indoor: Active Badge system, etc.
Berührung	Berührungssensoren
Temperatur	Thermometer
Physische Attribute	z.B. Biosensoren zur Hautberührungsmeßung oder Blutdruckmessung

**Tabelle 2.1** Häufig genutzte physische Sensoren

(in Anlehnung an Baldauf et al. 2007, S. 266)

Chen ( 2004) schlägt eine Einteilung der kontextsensitiven Systeme in drei Kategorien vor, die sich besonders in der Verarbeitung der gewonnenen kontextsensitiven Informationen unterscheiden. Die drei von ihm vorgeschlagenen Kategorien sind: *Direct sensor access*, *Middleware infrastructure* und *Context server*. Die Verarbeitungsstelle der Information wechselt bei den verschiedenen Kategorisierungen. Sie geht dabei von direkt im Gerät verankerten Verarbeitung, über eine Verarbeitung auf innerhalb des Gerätes implementierten Zwischenebenen hin zu einer Client-Server-Struktur, bei der jegliche Verarbeitung auf einem kontaktierten Server stattfindet (Baldauf et al. 2007, S. 264–265; Perera et al. 2014, S. 428).

Ein weiterer Ansatz ist die Einteilung der kontextsensitiven Systeme nach Kontextart und den zugehörigen Sensoren (**TABELLE 1.1**). Die Idee ist, dass verschiedene Umwelteinwirkungen und Kontexte durch verschiedene Sensoren wahrgenommen werden können. So kann ein Lichtsensor genutzt werden, um zu bestimmen, ob es hell oder dunkel ist, eine Uhr kann dem Gerät die Zeit mitteilen und ein GPS-Sensor den Standort auf der Welt. Weiterführend lassen sich diese Informationen verknüpfen, um so sehr präzise Ergebnisse zu liefern (Schmidt und van Laerhoven 2001, S. 67–68),

Im Rahmen dieser Arbeit soll eine Kombination der beiden Kategorisierungen zum Einsatz kommen. Dabei wird ein besonderer Schwerpunkt auf Middleware Infrastructure, Context Server und auf die visuellen Kontextarten gelegt.

Im Folgenden wird die Nutzung der verschiedenen Sensortypen anhand von Beispielen aufgezeigt, um eine Veranschaulichung der einzelnen Bereiche zu bieten.

### 2.2.1 QR Codes

Als spezielle Art der visuellen Kontextsensitivität (siehe Tabelle 2.1) sollten Identifizierungen von Objekten und Orten anhand von Markern gesehen werden. Die dafür am häufigsten genutzten Möglichkeiten sind die *QR (Quick Response) Codes* (Ashford 2010, S. 1).

QR Codes sind zweidimensionale, matrixbasierte Barcodes die Informationen sowohl in vertikaler als auch horizontaler Richtung enthalten (siehe **ABB. 2.1**). Sie wurden entwickelt, um schnell Informationen mithilfe geeigneter Scannersoftware auszulesen und dem Nutzer zur Verfügung zu stellen. Im Gegensatz zu eindimensionalen Barcodes, wie man sie zum Beispiel aus dem Einzelhandel kennt, haben sie dabei ein stark gesteigertes Speichervermögen. Zudem ist es nicht möglich, die Codes mit dem menschlichen Auge nachzuvollziehen (Chang et al. 2007, S. 231; Rouillard 2008, S. 52).



**Abb. 2.1** QR Code für den Titel dieser Arbeit

QR Codes wurden 1994 von dem japanischen Unternehmen Denso-Wave entwickelt. Es existieren 40 verschiedene Versionen des QR Codes, die sich vor allem in ihrer Größe und dadurch in ihrem Speichervolumen, aber auch in ihrer Auslesbarkeit trotz teilweiser Beschädigung unterscheiden (DENSO WAVE 2014).

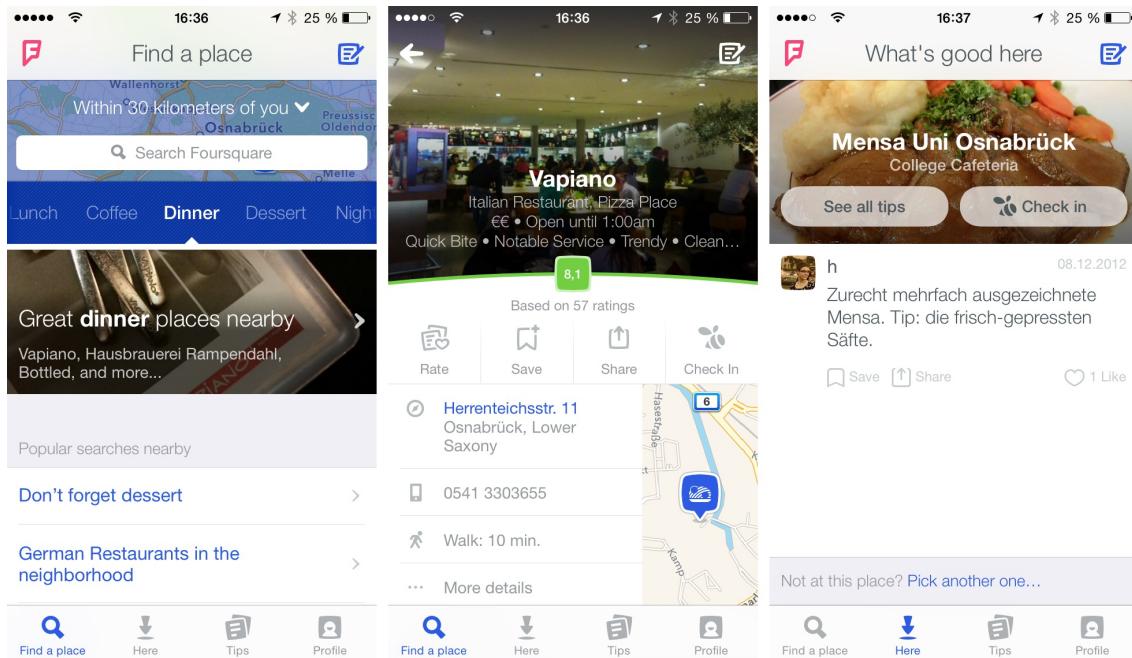
QR Codes finden heute vielfältige Anwendung. Sie sind in der Werbung, auf Visitenkarten, in Bibliotheken (Walsh 2010, S. 57) oder – einfach in ihrer ursprünglichen Bestimmung – in der Industrie aufzufinden. Dabei haben sie immer den Zweck, Objekte oder Orte zu identifizieren und weitere kontextsensitive Informationen zum Gescannten anzuzeigen. Sei es die Internetseite eines Unternehmens in der Werbung oder ein informativer Text über ein Ausstellungsstück in einem Museum. Zusätzlich ist es möglich, die vom QR Code zur Verfügung gestellte Information mit von weiteren Sensoren des Geräts festgestellten Informationen zu verknüpfen, um so ein ganzheitliches kontextsensitives System zu erreichen (Rouillard 2008, S. 52–53).

## 2.2.2 Location-Based Services

Ortsbasierte Dienste (engl. *Location-Based Services*) sind unter Normalnutzern von mobilen Endgeräten die heute wohl am verbreitesten und bekanntesten Beispiele unter den kontextsensitiven Anwendungen. Dafür spricht, dass neben einem in den USA registrierten Patent (Portman et al. 2005) auch empirische Studien zu der Akzeptanz dieser Dienste unter Endkunden (Kölmel und Yellowmap AG 2003; Junglas und Watson 2008) zum Thema erschienen sind. Zudem fällt auf, dass ein Großteil der Forschung zum Thema aus dem vergangenen Jahrzehnt stammt. Dies lässt auf eine so weitgehende Etablierung dieser Technik schließen, dass grundlegende Forschungsarbeiten verzichtbar sind. Neuere Veröffentlichungen beschäftigen sich eher mit der Verknüpfung von bestehenden Location-Based Services mit weiteren Technologien (Gao et al. 2012; Cho et al. 2011).

Auch die immense Fülle an reinen ortsbasierten Applikationen – von Foursquare (siehe **Abb. 2.2**) und Yelp bis hin zu Applikationen, die wie bspw. Facebook ihrem Kerndienst

Extrainformationen durch ortsbasierte Dienste hinzufügen – sprechen für die Annahme, dass Location-Based Services akzeptierte Anwendungen in der Welt der mobilen Endgeräte sind.



**Abb. 2.2** Foursquare auf iOS 8.1.1

So beschreibt das UMTS-Forum Location-Based Services bereits in seinem 13. Report von 2001 als einen Service, der es Nutzern oder Geräten ermöglicht, andere Personen, Fahrzeuge, Ressourcen, Dienste oder Maschinen anhand ihrer Positionen zu ermitteln. Zudem ermöglicht es dem Nutzer, seine eigene Position zu ermitteln (UMTS Forum 2001, S. 35).

Diese Lokalisierung geschieht durch verschiedene Techniken, die sich stark in ihrer Funktionsweise unterscheiden. Beispiele hierfür sind – wie bereits in der **TABELLE 1.1** genannt – physische Sensoren wie GSM, GPS und assisted GPS (aGPS) (Djuknic und Richton 2001, S. 123). Darüber hinaus ist eine Ermittlung anhand von Nutzereingaben oder durch andere Applikationen denkbar (Indulska und Sutton 2003, S. 1). Hierbei ist jedoch zu unterscheiden, ob die Lokalisierung unter freiem Himmel und so über Satelliten oder Sendemasten geschehen kann oder ob es sich um eine Lokalisierung in einem Gebäude handelt, welche dann zum Beispiel über die bereits genannten QR Codes geschehen könnte (Chang et al. 2007, S. 231).

### 2.2.3 Objekt- und Bilderkennung

Objekt- und Bilderkennung (engl. *Object Recognition*) sind als Unterbereich der Computer Vision zu sehen (Swain und Ballard 1991, S. 1). Bei Computer Vision

handelt es sich um einen Forschungsbereich der Informationstechnik, über den erste wissenschaftliche Veröffentlichungen bereits Mitte der 70er Jahre verfasst wurden (Baumgart 1974). Heute ist die Computer Vision eng in viele Anwendungsbereiche der Informationstechnologie, wie der Automation, Robotik oder auch der Medizininformatik eingebunden. Sie kommt zum Beispiel zur Steuerung von Montagearmen in der Industrie zum Einsatz (Szeliski 2010, S. 1–6). Aber auch im Endnutzerbereich lassen sich Applikationen der Computer Vision finden.

So existieren Applikationen, welche Texterkennung (engl. *Optical Character Recognition (OCR)*) (Cipolla et al. 1982, S. 1) zum Scannen von Textdokumenten nutzen und diese so automatisch in editier- und durchsuchbaren Text formatieren. Oder etwa Anwendungen wie Google Goggles, welches in der Lage ist, fotografierte Sudokurätsel zu lösen oder Informationen zu bekannten Gemälden in Museen zu suchen.

Objekt- und Bilderkennung nutzt Licht- und Bildsensoren (siehe Tabelle 2.1) zur Erkennung und Wiedererkennung von Objekten. Für die Gewinnung der Beschreibung gibt es verschiedene Ansätze, die von Kantenerkennung (engl. *Edge Detection*) (Canny 1986, S. 679) und Eckenerkennung (engl. *Corner Detection*) (Mehrotra et al. 1990, S. 1223) bis zur Blob Erkennung (engl. *Blob Detection*) (Shneier 1983, S. 345) reicht. All diese Erkennungs- und Beschreibungsalgorithmen sind aber grob unter den Algorithmen der Merkmalserkennung einzuordnen und haben gemein, dass sie entwickelt wurden, um bestimmte, einmalige Bildmerkmale zu erkennen.

Insgesamt ist der Bereich der Objekt- und Bilderkennung der Bereich in dem sich am wenigsten Anwendungen finden lassen. Er bietet sich daher für eine Umsetzung in dieser Arbeit an.

### 3 Google Glass

#### 3.1 Die Google Glass als Vertreter der Augmented Reality

Bereits seit Mitte der 90er Jahre des letzten Jahrhunderts ist die erweiterte Realität (engl. Augmented Reality) ein intensiv beachteter Forschungsbereich. Hierbei geht es um die Integration von virtuellen Elementen in die Realität und im Gegensatz zur virtuellen Realität nicht um die künstliche Darstellung von Räumen und Objekten (Azuma 1997, S. 2). In dieser Weise wird die Augmented Reality genutzt, um dem Nutzer die Möglichkeit zu bieten, mit der ihm umgebenden Umwelt zu interagieren, während diese gleichzeitig um virtuelle Elemente erweitert wird (Huang et al. 2013, S. 1–2).

Allerdings gibt es Uneinigkeit darüber, ab welchem Zeitpunkt eine Erweiterung als Augmented Reality anerkannt werden kann. So bezeichnet Azuma (1997, S. 2) Augmented Reality noch als Systeme, die Virtualität und Realität kombinieren, in Echtzeit interaktiv agieren und dreidimensional dargestellt werden. Er schließt zudem 2D-Einblendungen konsequent als Teil der Augmented Reality aus. Spätere Definitionen wie die von Huang et al. (2013, S. 1) heben diese Beschränkung auf und akzeptieren mobile Geräte als Vertreter der Augmented Reality, wenn sie mit dieser interagieren und um Elemente gleich welcher Natur erweitern. Sie bezeichnen Vertreter dieser Gerätetypen als Mobile Augmented Reality (MAR) und unterteilen diese weiter in sechs Unterkategorien:

1. *Notebooks*
2. *Personal Digital Assistants (PDAs)*
3. *Tablet Personal Computer (Tablets)*
4. *Ultra Mobile PCs (UMPCs)*
5. *Mobiltelefone*
6. *AR-Brillen*

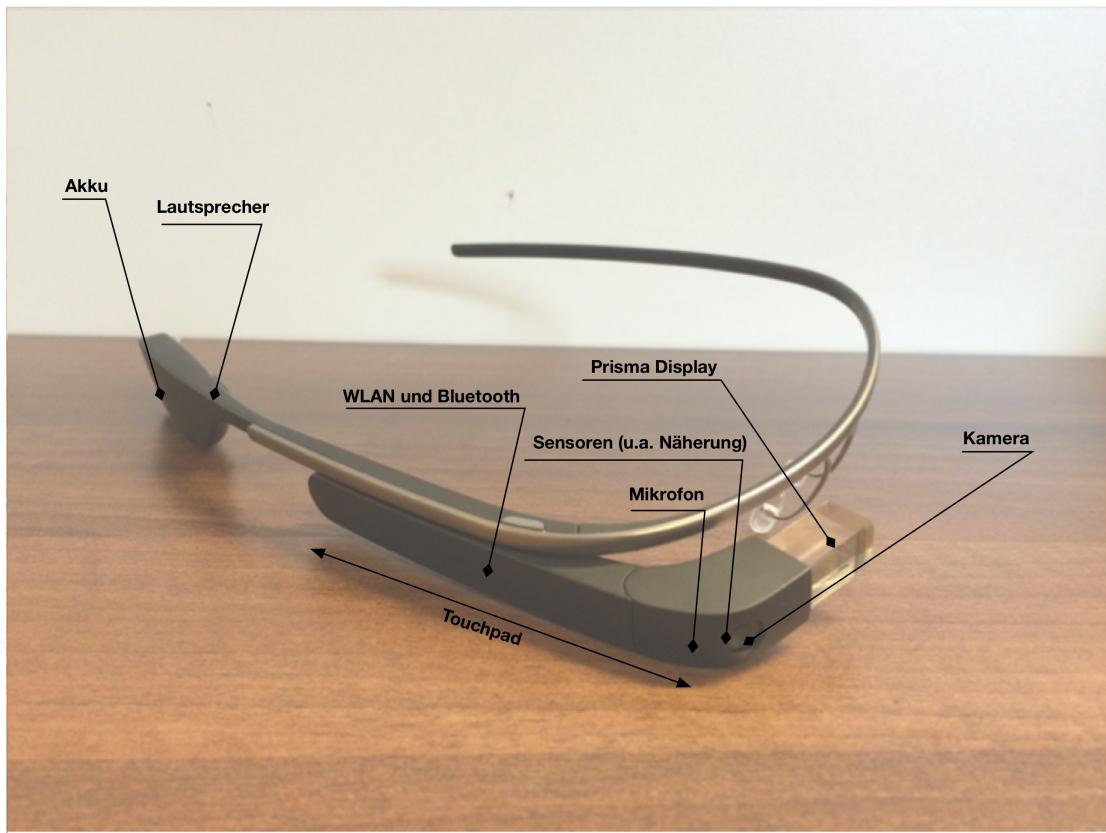
Die AR-Brille von Google, die Google Glass, wurde erstmals im Februar 2012 (Bilton 2012) erwähnt und im Juni während der Unternehmensmesse Google I/O 2012 im Rahmen des Project Glass offiziell vorgestellt (Google 2012). In den Verkauf ging das Gerät im Frühjahr 2013, gleichzeitig mit dem Glass Development Kit für Entwickler und anerkannte Tester (Stevens 2013). Bis heute befindet sich Project Glass in der Open

Beta und ist daher nur in sehr kleiner Auflage für Entwickler und Forschungszwecke verfügbar (Google 2014a).

### 3.2 Spezifikationen und Besonderheiten der Google Glass

Im Folgenden wird die Google Glass vorgestellt und ihre Besonderheiten auf Hard- und Softwareseite aufgezeigt.

#### 3.2.1 Hardwarespezifikationen



**Abb. 3.1** Beschriftete Google Glass (in Anlehnung an Feng et al. 2014, S. 3070)

Bei der Google Glass (siehe **ABB. 3.1**) handelt es sich um eine über Sprachbefehle oder ein Touchpad bedienbare AR-Brille. Sie ist mit WLAN- und Bluetoothmodulen ausgestattet, welche für die Konnektivität zu WLAN-Netzwerken oder aber die Verbindung mit einem Smartphone sorgen (Feng et al. 2014, S. 3069).

Allerdings stellt sie ihre Inhalte nicht durch Projektion auf Brillengläsern dar, sondern integriert ein kleines Display in die obere rechte Ecke des Sichtfeldes des Trägers. Es handelt sich hier also um ein *Head-up-Display (HUD)* (Lins et al. 2014, S. 167–168).

Dabei hat es durch das Prisma Display den Eindruck, als würde die Benutzeroberfläche in einiger Entfernung vor dem Nutzer schweben (Torborg und Simpson 2012).

Im hinteren Teil des Gerätes befinden sich ein einzelliger Lithium Polymer Akku mit 2.1 Wattstunden (Wh) und einem Bone Conduction Speaker. An der rechten Seite ist ein Touchpad zu finden, welches die Eingabe ähnlich einem Notebook-Touchpad ermöglicht. In Verlängerung des Touchmoduls befinden sich die Recheneinheit mit Mikrofon, GPS-Einheit, einer 5 Megapixel-Kamera, sowie Beschleunigungs- und Erschütterungssensoren. Das User Interface (UI) wird dem Träger auf einem Prisma Display mit einer nativen Auflösung von 640x360 Pixel angezeigt. Das Gerät hat eine Speicherkapazität von 16 GB Flash-Speicher (Torborg und Simpson 2012; Google 2014b). Gesteuert wird das Gerät von einem ARM-Prozessor mit bis zu 1000 Megahertz (MHz) (Texas Instruments 2012, S. 125).

Das Gerät hat bei starker Beanspruchung – wie zum Beispiel dem Aufnehmen eines 720p-Videos oder der durchgehenden Berechnung von aufwändigen Algorithmen – eine Akkulaufzeit von ca. einer Stunde. Dies lässt darauf schließen, dass es nicht als allzeit eingeschaltetes, sondern als nur im Bedarfsfall gefragtes Gerät geeignet ist (Lins et al. 2014, S. 168–169).

### **3.2.2 Softwarespezifikationen**

Die Glass benutzt Googles hauseigenes Betriebssystem für den mobilen Markt, *Android*. Dies ermöglicht es Entwicklern, in einer gewohnten Umgebung zu arbeiten. Bei Android handelt es sich um ein frei erhältliches Betriebssystem von Google, welches am 23. September 2008 in der ersten Major Version 1.0 erschien (Morril 2008). Zum Stand dieser Arbeit ist die aktuellste Version die Major Version 5.0 (Google 2014c).

Android basiert in hohem Maß auf der Programmiersprache Java. Diese wird allerdings anders als auf dem Desktop nicht von einer *Virtual Machine (VM)* von Oracle, sondern von einer Eigenentwicklung von Google ausgeführt, der Dalvik VM. Dalvik wurde mit besonderer Aufmerksamkeit für die mobile Plattform entwickelt und führt automatisch grundlegende Aufgaben wie Speichermanagement und Multithreading für den Nutzer aus. Die Verwendung der Java-Plattform ermöglicht es Entwicklern, auf bekannte Methoden und Bibliotheken zurückzugreifen (Saha 2008, S. 49).

Zum Ansprechen der speziellen Komponenten der AR-Brille hat Google zudem eine Erweiterung des Android *Software Development Kit (SDK)*, das Glass Development Kit

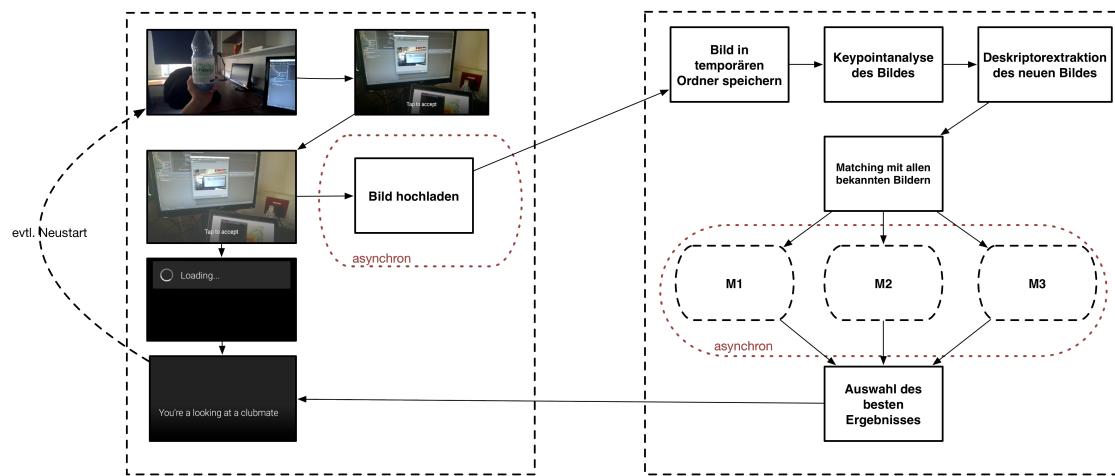
(GDK), veröffentlicht (Lins et al. 2014, S. 169). Zum Zeitpunkt dieser Arbeit lag GDK in der Version XE22.0 vom 14. Oktober 2014 vor (Google 2014d).

GDK übernimmt auf der Glass im Wesentlichen die Verwaltung der Glass-eigenen Komponenten. Zudem werden Methoden zur Verfügung gestellt, um ein Erstellen der speziell für die AR Brille entwickelten UI-Elemente, den Cards, zu ermöglichen (Lins et al. 2014, S. 173).

## 4 Einblendung von kontextsensitiven Inhalten auf der Google Glass

In diesem Kapitel soll in die Idee und Funktionsweise der implementierten Applikation sowie der genutzten Bibliotheken eingeführt werden.

## 4.1 Idee und Funktionsweise der kontextsensitiven Applikation



**Abb. 4.1** Darstellung der Funktionsweise der kontextsensitiven Applikation

Bringt man die Google Glass als Vertreter eines Geräts der Augmented Reality und Kontextsensitivität zusammen, fällt der hohe Überschneidungsgrad der beiden Themenbereiche auf. Selbst eine kleine auf der Glass vorinstallierte Anwendung wie der Kompass erfüllt streng genommen bereits die Anforderungen, um als kontextsensitive Applikation akzeptiert zu werden. Hier werden anhand der Position des Nutzers über GPS die Himmelsrichtungen ermittelt, es werden also ortsbasierte Informationen angezeigt und somit die Wahrnehmung des Nutzers um in seinem Kontext relevante Informationen ergänzt, was sowohl der in **KAP. 2.2.1** angesprochenen Definition eines Location-Based Services, als auch der in **KAP. 2.1** genannten eines kontextsensitiven Systems genügt.

Auch einige von freien Entwicklern veröffentlichte Anwendungen gehen den Weg der Kontextsensitivität und nutzen die Möglichkeiten der Google Glass, um die über Sensoren ermittelten Umwelteinflüsse auszuwerten und so dem Nutzer kontextsensitive Informationen anzubieten. So sind Applikationen erhältlich, die anhand der Position und Blickrichtung des Nutzers relevante Wikipedia-Artikel anzeigen.

Die Idee der für diese Arbeit entwickelten Applikation entstand direkt aus dem angeschlossenen Forschungsprojekt Glassroom. Durch Nutzung einer AR-Brille, in diesem Fall der Google Glass, soll es einem Mitarbeiter im technischen Kundendienst ermöglicht werden, allein durch Interaktion mit der AR-Brille zusätzliche Informationen zu der vor ihm liegenden Aufgabe und den ihm eventuell bisher unbekannten Komponenten zu erhalten. Durch einfaches Fotografieren mit der Brille soll ermöglicht werden, zusätzliche Informationen zu bereits bekannten Objekten anzubieten und diese dem Nutzer über Texteinblendung zur Verfügung zu stellen.

Die Brille schickt dazu das Foto an einen Server, auf dem die Übereinstimmung des Bildes mit allen bereits bekannten Bildern ermittelt wird. Bei Erfolg werden die Informationen zu dem am besten übereinstimmenden Bild an die Google Glass zurückgesendet. Erreicht der Matchingprozess mit keinem der bekannten Bilder eine vorher festgelegte Akzeptanzgrenze, wird der Prozess abgebrochen und dem Nutzer eine entsprechende Fehlermeldung angezeigt. Die Architektur der Applikation lässt sich in **ABBILDUNG 4.1** nachvollziehen.

Wie bereits in **3.2.1** genannt handelt es sich bei der Google Glass um kein leistungsstarkes Gerät, um daher diese rechenintensiven und aufwendigen Prozesse von der AR-Brille zu nehmen, wurde jegliche Rechenarbeit auf einen Server in der *Cloud* ausgelagert. Auf dem Gerät selber verbleiben nur die den User direkt betreffenden Prozesse wie UI-Darstellung, Fotografieren, Upload des Bildes und Anzeigen der Ergebnisse. Ein weiterer Grund für die serverseitige Berechnung entstand durch die Auslagerung der benötigten Methoden zur Keypoint-Erkennung und Deskriptorextraktion in der angewandten Computer Vision Bibliothek in ein Nonfree-Modul. Dieses ist bis heute nur über Umwege in der Android-Version verfügbar (opencv dev team 2014).

Der implementierte Matchingprozess erfolgt dabei in vier Schritten:

1. Keypointerkennung im hochgeladenen Bild
2. Extraktion der Deskriptoren, die das Bild beschreiben
3. Matching der Deskriptoren gegen bereits ermittelte, abgespeicherte Deskriptoren
4. Auswahl des besten Ergebnisses und Rückgabe des Ergebnisses an die Glass

Diese Vorgehensweise hat besonders im Bereich der Skalierbarkeit enorme Vorteile. Zudem verschwindet durch die serverseitige Pflege der Vergleichsdaten, das Problem einer inkonsistenten und redundanten Datenhaltung, welche durch Pflege der Daten auf

Clientseite entstehen würde. Es erleichtert aber auch den nachträglichen Austausch der AR-Brille oder sogar die Integration eines weiteren mobilen Gerätes, da der Server über eine einfache *REST-API (Representational State Transfer - Application Programming Interface)* verfügt.

Bei REST handelt es sich um ein Programmierparadigma, welches durch Standardisierung der Client-Server-Kommunikation einen vereinheitlichten Zugang zu Webressourcen ermöglicht (Fielding 2000, S. 86).

Um dem Nutzer neben der reinen Erkennung eines Objektes auch weitergehende kontextsensitive Informationen anzubieten wurde zudem eine Logik zur Abfrage von Informationen zu dem dargestellten Objekt aus der freien Internet-Enzyklopädie Wikipedia eingebaut. Diese greift beim Hinzufügen eines neuen Objektes automatisch ein und ermittelt den ersten Absatz zu einem zusätzlich angegebenen Schlüsselwort.

## **4.2 Vorstellung von OpenCV und der verwendeten Algorithmen**

Im Folgenden werden die für die Bilderkennung essenzielle Bibliothek OpenCV und die dahinterstehenden genutzten Algorithmen vorgestellt und beispielhaft erläutert.

### **4.2.1 OpenCV und JavaCPP**

Zur Implementation der Kernlogik der Applikation wurden OpenCV beziehungsweise der Wrapper JavaCV von Bytedeco genutzt. Bei OpenCV (<http://opencv.org>) handelt es sich um eine in C++ und C implementierte, frei erhältliche Bibliothek, die gängige Methoden der Computer Vision in sich vereint und so Forschern und interessierten Nutzern einen Einstieg bietet. OpenCV wird aktiv für C++, Java, Python, Ruby, Matlab und einigen weiteren Programmiersprachen entwickelt (Bradski und Kaehler 2008, S. 1). Auch für Android findet man Implementierungen. Die Umsetzungen für die verschiedenen Programmiersprachen sind allerdings verschieden weit entwickelt und so kann es vorkommen, dass wichtige Grundfunktionen in der genutzten Sprache noch gar nicht zur Verfügung stehen und aus diesem Grund auf die Kernimplementierung in C++ zurückgegriffen werden muss.

Um solche Probleme zu umgehen wurde für die Implementierung des hier vorgestellten Prototypen auf JavaCPP zurückgegriffen. Dabei handelt es sich um Interfaces zu gängigen C++ Bibliotheken, die unter anderem auch eine Umsetzung von OpenCV für Java beinhalten. Umgesetzt wurden diese von Bytedeco, einer Gemeinschaft von Entwicklern, die sich dem Ziel verschrieben haben, C++ Bibliotheken in Java zugänglich zu machen. Durch Nutzung dieser konnte ein gut konfigurierbarer Zugang

zu OpenCV geschaffen werden. Zur Implementierung dieser Arbeit wurde OpenCV in der Version 2.4.9 und JavaCPP in der Version 0.9 genutzt.

#### 4.2.2 SURF und Fast Approximate Nearest Neighbor Matching

Zur Erkennung und zum Abgleich der auf dem Server hinterlegten Bilder und der von der Google Glass fotografierten Objekte wurden *Speeded-Up Robust Features (SURF)* und *Fast Approximate Nearest Neighbor Matching* genutzt.



**Abb. 4.2** SURF-Keypointererkennung auf einem Logo

Bei SURF handelt es sich um einen Keypointerkennungs- und Beschreibungsalgorithmus. Er wird als ein als Algorithmus zur Blob Erkennung (siehe 2.2.3) gezählt (Bay et al. 2008, S. 8). Dieser ist in der Lage, trotz Skalierung und Drehung Bilder und Objekte wiederzuerkennen. Der Algorithmus wurde von Bay et al. (2008, S. 1) vorgestellt und gilt seitdem als fehlerunanfälligster und effizientester Algorithmus zur Keypointerkennung und -beschreibung (Sidla et al. 2011, S. 7–8). Zwar gibt es mit Binary Robust Invariant Scalable Keypoints (BRISK) (Leutenegger et al. 2011, S. 1) und Fast Retina Keypoints (FREAK) (Alahi et al. 2012, S. 1) moderne Algorithmen, welche bei einer ihnen angepassten Anwendung deutlich effizienter sind (Schaeffer 2013, S. 5), doch bei einigen eigenen Testläufen stellte sich SURF als Keypointerkenner und -beschreiber am zuverlässigsten heraus.

Die Umsetzung von SURF ist eine Weiterentwicklung des von Lowe (1999, 2004) entwickelten Scale Invariant Feature Transform (SIFT) (Bay et al. 2008, S. 3). Bei beiden ist hervorzuheben, dass sie zum korrekten Arbeiten Bilder in Graustufen

benötigen (Schaeffer 2013, S. 2). Anders als SIFT, welches eine Bildpyramide baut, um diese dann mit Gaussfilter mit ansteigendem Sigmawert zu bearbeiten und so aus der Differenz der einzelnen Ebenen relevante Bildpunkte zu erkennen, nutzt SURF einen Stack, in dem verschiedene Skalierungen des Bildes vorliegen. Diese werden mit Mittelwertfiltern bearbeitet. Durch die Verwendung von Integralbildern kann eine Berechnung in konstanter Zeit geschehen, was SURF eine höhere Effizienz gegenüber SIFT bringt (Juan und Gwun 2009).



**Abb. 4.3** SURF-Keypointerkennung auf einem Foto

Nach Erkennung und Berechnung der einzelnen Bildpunkte – wie in **ABB 4.2** und **4.3** zu sehen – wird anhand dieser von SURF das Bild in Quadrate aufgeteilt und für diese dann jeweils die Ausrichtung bestimmt (Bay et al. 2008, S. 7). Dieser Schritt wird im Folgenden als Beschreibung des Bildes genutzt und anhand dieser kann eine Wiedererkennung des dargestellten Objekts geschehen.



**Abb. 4.4** Neares Neighbor Matching der beiden Bilder

Dieser Vorgang geschieht bei einem Fast Approximate Nearest Neighbor Matching für beide Bilder (siehe **ABB 4.4**). Zur Geschwindigkeitssteigerung wird die Suche nach einem Match zweier Punkte in den zu vergleichenden Datensätzen nicht durch lineare Suche, sondern durch Annäherung erreicht. Dies hat zur Folge, dass auch nicht eindeutige, „schlechte“ Ergebnisse dabei entstehen (Muja und Lowe 2009, S. 1).



**Abb. 4.5** Filterung des Matchings

Um den Geschwindigkeitsvorteil des Fast Approximate Nearest Neighbor Matchings, trotz teilweise abweichender Ergebnisse nutzen zu können werden die Ergebnisse des Vorgangs im Nachhinein gefiltert (siehe **ABB. 4.5**).

Dazu wird der Quotient aus der Distanz des besten Matches und der Distanz des zweitbesten Matches gezogen. Bei einem Quotient  $< 0.7$  wird ein Match als „gut“ eingestuft und somit als im Quellbild wiedererkannt bezeichnet. Dieser Quotient als hinreichende Bedingung für einen guten Match wurde von Lowe ( 2004, S. 19–20) umfangreich erarbeitet. So ist es möglich, Bilder anhand der SURF-Bildpunkte und Deskriptoren zu vergleichen und lediglich über die Anzahl der „guten“ Matches eine Übereinstimmung der beiden festzustellen.

## 5 Umsetzung einer kontextsensitiven Applikation mit OpenCV

In diesem Kapitel werden zuerst die verwendeten technischen Komponenten der umgesetzten prototypischen Applikation vorgestellt. Im Anschluss wird in die Implementation eingeführt um diese abschließend auszuwerten.

### 5.1 Vorstellung der Implementation und ihrer Komponenten

Die Implementierung der hier vorgestellten Applikation besteht aus zwei wesentlichen Teilen: einerseits dem Client auf der Google Glass, welcher die Darstellung des User Interfaces, das Fotografieren des Objektes und den Upload zum Server übernimmt, und andererseits dem Server, welcher jegliche Aufgaben im Bereich des Matchings, der Verwaltung und der Informationsabfrage übernimmt. Es handelt sich also um eine Applikation der Kategorie Context Server (siehe 2.2), welche Sensoren des visuellen Typs (siehe Tabelle 2.1) nutzt. Der Übersicht halber sind die wichtigsten genutzten technischen Komponenten in der **TABELLE 5.1** zu finden

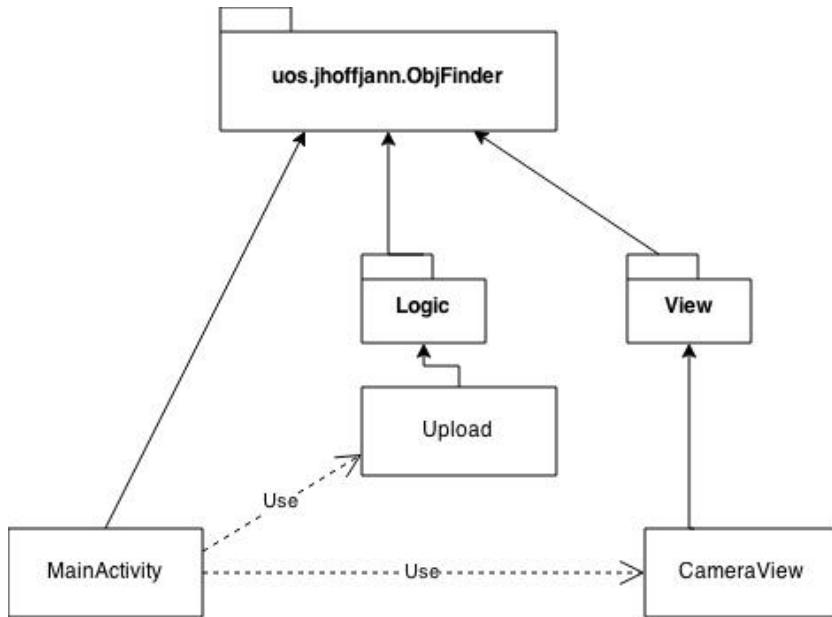
<i>Komponente</i>	<i>Einsatzbereich</i>	<i>Versionsnummer</i>	<i>Webseite</i>
Glass Development Kit (GDK)	Glass Client	XE 22.0	<a href="https://developers.google.com/glass/develop/gdk/index">https://developers.google.com/glass/develop/gdk/index</a>
Gradle	Glass Client	2.1	<a href="http://gradle.org">http://gradle.org</a>
Apache HTTP Components	Glass Client & OCV Server	4.3.5	<a href="https://hc.apache.org">https://hc.apache.org</a>
Jetty	OCV Server	6.1.10	<a href="http://www.eclipse.org/jetty/">http://www.eclipse.org/jetty/</a>
Spring Framework	OCV Server	4.1.3	<a href="http://spring.io">http://spring.io</a>
Maven	OCV Server	3.2.3	<a href="https://maven.apache.org">https://maven.apache.org</a>
Log4j	OCV Server	2.1	<a href="http://logging.apache.org/log4j/2.x/">http://logging.apache.org/log4j/2.x/</a>
GSON	OCV Server	2.2.4	<a href="https://code.google.com/p/google-gson/">https://code.google.com/p/google-gson/</a>
JavaCPP	OCV Server	0.9	<a href="https://github.com/bytedeco/javacpp">https://github.com/bytedeco/javacpp</a>
OpenCV	OCV Server	2.4.9	<a href="http://opencv.org">http://opencv.org</a>

**Tabelle 5.1** Übersicht der genutzten Komponenten

Der Glass Client wurde mit der zu dem Zeitpunkt dieser Arbeit aktuellsten Version des Glass Development Kits (XE 22.0) programmiert, welche auf der Android Version 4.4.2 basiert. Die Verwaltung der Bibliotheken geschieht auf Clientseite mit Gradle, einer Paketverwaltung und Kompilierautomatisierung, die von Google zur Entwicklung von Android-Applikationen empfohlen wird (Google 2014e). Die Standardbibliotheken von Android wurden durch den gezielten Einsatz der Apache HTTP Components in der Version 4.3.5 erweitert. Dabei handelt es sich um in der Java-Welt etablierte Bibliotheken zur HTTP-Kommunikation auf Client- und Server-Seite. Apache stellt zudem eine eigene Version der Bibliothek für das Android-System bereit, was der dem mobilen Betriebssystem eigenen Rechteverwaltung Aufmerksamkeit schenkt.

Der OpenCV-Server (OCV-Server) nutzt als Grundgerüst einen Jetty-Server. Bei Jetty handelt es sich um ein frei erhältliches Server-Framework der Eclipse Foundation (Eclipse Foundation 2014). Dieser wurde zur einfachen Umsetzung des Model-View-Controller-Prinzips (MVC) um das Spring Framework erweitert. Aufgrund der guten Integration mit Jetty kommt zur Bibliotheksverwaltung und Kompilierungsautomatisierung Maven zum Einsatz. Das Logging der Serveraktivitäten und -fehler geschieht über die Log4j-Bibliothek von Apache. Jegliche Nutzung von JSON-Dateien wird mit der GSON-Bibliothek von Google realisiert, welche eine Erstellung von JSON aus Java Objekten und vice versa ermöglicht (Singh et al. 2014). Die Rechenlogik des Servers nutzt JavaCPP in der Version 0.9, welches OpenCV in der Version 2.4.9 zur Verfügung stellt. Auch auf dem Server werden zur Behandlung von HTTP-Anfragen und Antworten die Apache HTTP Components genutzt.

## 5.2 Glass Client



**Abb. 5.1** UML-Darstellung des Glass Clients

Der auf der Google Glass ausgeführte Client (siehe **ABB. 5.1**) hat zwei Hauptkomponenten: die *CameraView-Klasse*, welche die Verwaltung der Kamera gegenüber dem Android-System übernimmt und die *MainActivity-Klasse*, welche den Großteil der Logik der Applikation darstellt. Daneben gibt es noch eine Hilfsklasse, *Upload*, welche die *MainActivity-Klasse* beim Hochladen der Bilder an den Server unterstützt, sowie zwei versteckte, interne Klassen innerhalb von *MainActivity*..

Der Prozess der Einblendung von kontextsensitiven Informationen beginnt auf der Google Glass. Der Nutzer startet die Applikation und bringt das zu identifizierende Objekt in den Fokus, durch einfaches Tippen auf das berührungsempfindliche Bedienfeld der Glass wird das Foto aufgenommen. Nach erfolgreichem Fotografieren und Abspeichern des Bildes findet der Upload an den Server statt, welcher automatisch im Hintergrund als asynchroner Prozess läuft (siehe **CODE 5.1**). Dies garantiert eine weitere Verfügbarkeit der Benutzeroberfläche bei gleichzeitiger, verlässlicher Ausführung des Uploads.

```

private class asyncUploading extends AsyncTask<Void,
Void, Void> {
    private final ProgressDialog dialog = new
    ProgressDialog(MainActivity.this);
    private String strResult = null;

    protected void onPreExecute() {
        this.dialog.setMessage("Loading...");
    }
}
  
```

```

this.dialog.setCancelable(false);
this.dialog.show();
}

@Override
protected Void doInBackground(Void... params) {
strResult = Upload.upload(URL, image);
return null;
}

protected void onPostExecute(Void result) {
if (dialog.isShowing()) {
dialog.dismiss();
}
updateMainUi(strResult);
}
}

```

#### Code 5.1 Asynchroner Uploadprozess

Der asynchrone Prozess greift für die Ausführung des Uploads auf eine eigene Uploadverwaltung (**CODE 5.2**) zurück, welche Apache HTTP Components nutzt, um die Anfrage an den Server vorzubereiten, zu stellen und die Antwort im Anschluss zu empfangen, auszuwerten und zu interpretieren. Für eine erfolgreiche Rückmeldung des Servers müssen der HTML-Statuscode 200, sowie ein Ergebnisinhalt vorliegen. Ist dies nicht gegeben, wird der Prozess an diesem Punkt abgebrochen und dem Nutzer wird eine Fehlermeldung angezeigt.

```

try {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(URL);
    MultipartEntityBuilder entity =
    MultipartEntityBuilder.create();
    entity.addTextBody("name", new Date() + "");
    entity.addBinaryBody("file", image);
    httpPost.setEntity(entity.build());
    HttpResponse response = httpClient.execute(httpPost);
    int statusCode =
    response.getStatusLine().getStatusCode();
    if (statusCode != 200) {
        return "Error: " + statusCode + " Something in the
        uploading process went wrong";
    }
    if (response.getEntity() != null) {
        HttpEntity responseEntity =
        response.getEntity();
        String resStr = EntityUtils.toString(responseEntity);
        // parse to JSON
    }
}

```

```
JSONObject result = new JSONObject(resStr);
String token = result.getString("message");
responseEntity.consumeContent();
return token;
}
return "Something went terribly wrong";
}
```

**Code 5.2** Erstellung der Anfrage und Auswertung der Antwort

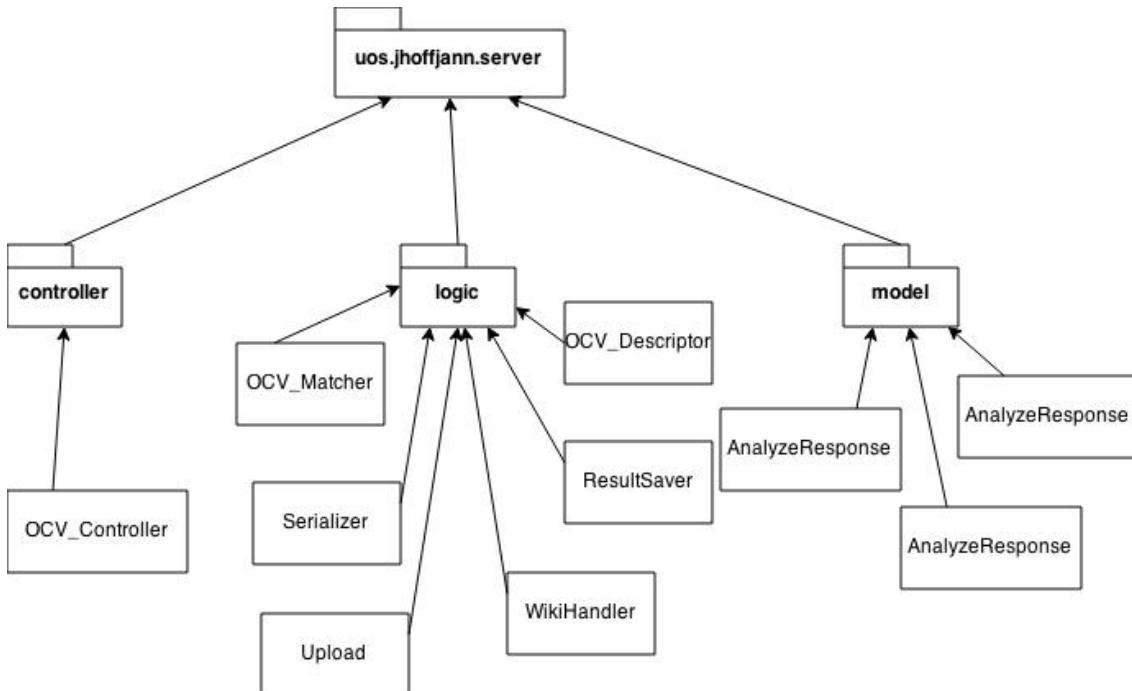
Nach der Antwort der Uploadverwaltung wird das Ergebnis an eine Methode zur Anpassung der Benutzeroberfläche (**CODE 5.3**) weitergeleitet. Diese erstellt zur Anzeige des Ergebnisses eine Card und befüllt diese mit dem Ergebnis, um sie dann für den Nutzer sichtbar in den Vordergrund zu bringen.

```
public void updateMainUi(String result) {
    CardBuilder cardBuilder = new CardBuilder(this,
CardBuilder.Layout.TEXT);
    cardBuilder.setText(result);
    View resultView = cardBuilder.getView();
    cameraView.releaseCamera();
    this.setContentView(resultView);
}
```

**Code 5.3** Aktualisierung der Benutzeroberfläche

Im Anschluss kann der Vorgang durch einfaches Tippen auf das Touchfeld neu gestartet werden.

### 5.3 OCV Server



**Abb. 5.2** UML-Darstellung des OCV Servers

Die Anwendungsmöglichkeiten des OCV Server erstrecken sich auf zwei Bereiche:

1. Das Hinzufügen von Objekten zur Datenbasis des Servers
2. Die Analyse eines gesendeten Bildes mit Matching gegen die Datenbasis

Die Ausführung der beiden Grundfunktionen geschieht grundsätzlich unabhängig voneinander, allerdings wird dabei auf die gleichen Klassen – insbesondere im Bereich der SURF-Analyse der Bilder und der Datenverwaltung – zurückgegriffen (siehe **ABB. 5.2**).

#### 5.3.1 Hinzufügen eines neuen Objekts

```

public static String getPlainSummary(String url) {
    try {
        Document doc = Jsoup.parse(new
URL(url).openStream(), "UTF-8", url);
        Elements paragraphs = doc.select("#mw-content-text
p");
        Element firstParagraph = paragraphs.first();
        logger.debug(firstParagraph.text());
        return firstParagraph.text();
    } catch (Exception e) {
        logger.error(e.getMessage());
        e.printStackTrace();
    }
}
    
```

```

return "Nothing found here!";
}
}

```

#### Code 5.4 Extrahierung des ersten Absatzes eines Wikipedia-Artikels

Das Erstellen eines neuen Objekt kann über ein POST-Request an die URL des Servers (bspw. localhost:8080/opencvserver-server/add) geschehen. Diese Anfrage wird vom OCV\_Controller entgegengenommen und weiter bearbeitet.

Nach erfolgreichen Hochladen und Überprüfung des hochgeladenen Bildes wird eine Anfrage an den WikiHandler (**CODE 5.4**) geschickt. Dieser führt eine Suche nach dem mitgelieferten Namen des Objektes in der freien Internetencyklopädie Wikipedia durch und extrahiert im Anschluss den ersten Absatz des Ergebnisses.

```

public static opencv_core.Mat getDescriptor(File
image, boolean debug) {
opencv_features2d.KeyPoint keypoints = new
opencv_features2d.KeyPoint();
opencv_core.Mat descriptors = new opencv_core.Mat();
opencv_core.Mat mImage =
opencv_highgui.imread(image.getAbsolutePath());
opencv_imgproc.cvtColor(mImage, mImage,
opencv_imgproc.COLOR_BGR2GRAY);
surfFeatureDetector.detect(mImage, keypoints);
surfDescriptorExtractor.compute(mImage, keypoints,
descriptors);
return descriptors;
}

```

#### Code 5.5 Keypointerkennung und Deskriptorextraktion

Danach wird durch den OCV\_Descriptor (**CODE 5.5**) eine Keypointanalyse und Deskriptorextraktion mit Hilfe des von OpenCV bereitgestellten SURF-Algorithmus durchgeführt. Diese Ergebnisse werden in einer Matrix zurückgeliefert, vom Serializer gelesen und in eine Extensible Markup Language Datei (XML-Datei) überschrieben, um die Ergebnisse für zukünftige Analyseanfragen verfügbar zu machen.

```

public static String serializeMat(String name,
opencv_core.Mat sMat) {
    File dir = new File(root + File.separator +
"object_xml");
    if(!dir.exists())
        dir.mkdirs();
    String filePath = dir.getAbsolutePath() +
File.separator + UUID.randomUUID() + ".xml";
    opencv_core.FileStorage storage = new
opencv_core.FileStorage(filePath,

```

```

    opencv_core.FileStorage.WRITE);
    opencv_core.CvMat cvMat = sMat.asCvMat();
    storage.writeObj(name, cvMat);
    storage.release();
    return filePath;
}

```

#### Code 5.6 Serialisierung einer Deskriptormatrix

Der Serializer (**CODE 5.6**) nutzt zum Schreiben der Matrix in eine XML-Datei die C++ Klasse FileStorage, welche über JavaCPP zugänglich gemacht wird. Aufgrund der Komplexität der von OpenCV genutzten Matrizen war dies der effektivste Weg, um die ermittelten Deskriptoren für die spätere Nutzung zwischen zu speichern.

```

{
  "name": "Club-Mate",
  "descriptorPath": "/root/Bachelorarbeit/dev/OCV_Serve
r/opencvserver-server/object_xml/d51.xml",
  "creationDate": "Nov 28, 2014 4:30:28 PM",
  "description": "Club-Mate ist ein koffeinhaltiges,
  alkoholfreies Erfrischungsgetränk der Brauerei
  Loscher aus Mönchsteinach. Club-Mate basiert auf der
  Pflanze Mate und hat einen Koffeingehalt von
  20 Milligramm pro 100 Milliliter.[1]"
}

```

#### Code 5.7 Beispiel eines abgespeicherten Objekts

Nach erfolgreicher Durchführung der genannten Teilprozesse werden die das Objekt beschreibenden Daten (Name, Wikipediaauszug, absoluter Pfad des Deskriptordokuments) zusammengetragen und gesammelt in einer JavaScript Object Notation–Datei (JSON-Datei) abgespeichert (**CODE 5.7**). Tritt bei einem der Teilschritte ein Fehler auf, wird dem Nutzer eine Fehlermeldung zurückgegeben und ein entsprechender Fehlerbericht in die Log-Datei des Servers geschrieben.

#### 5.3.2 Analyse eines gesendeten Bildes

Die Analyse eines gesendeten Bildes funktioniert ähnlich dem Hinzufügen eines neuen Bildes. Die Anfrage zur Analyse wird vom Controller über ein POST (bspw. localhost:8080/opencvserver-server/analyze) entgegengenommen und das Bild zur Analyse wird vorläufig in einem temporären Ordner auf dem Server abgespeichert.

Im Anschluss werden die Keypoints und Deskriptoren für das hochgeladene Bild ermittelt, um diese mit den gesammelten Objekten vergleichen zu können.

Der Matchingprozess erfolgt aus Gründen der Effizienzsteigerung für jedes Objekt in einem Thread. Alle Threads werden während dieser Zeit von einem Threadpool verwaltet. Während des Matchingprozesses werden zunächst die Deskriptoren des Objektes aus der XML-Datei vom Serializer zurück in eine Matrix geschrieben. Diese Objektmatrix und die Matrix mit den Deskriptoren des hochgeladenen Bildes werden dann vereint an den OCV\_Matcher (**CODE 5.8**) gegeben.

```

@Override
public Result call() {
    opencv_features2d.DMatchVectorVector matches = new
    opencv_features2d.DMatchVectorVector();
    matcher.knnMatch(descriptors[0], descriptors[1],
    matches, 2);
    ArrayList<Double> goodMatches =
    getGoodMatches(matches);

    return new Result(name, goodMatches, path);
}

private ArrayList<Double>
getGoodMatches(opencv_features2d.DMatchVectorVector
matches) {
    ArrayList<Double> goodMatches = new
    ArrayList<Double>();
    for (int j = 0; j < matches.size(); j++) {
        double mRatio = matches.get(j, 0).distance() /
        matches.get(j, 1).distance();
        if (mRatio <= RATIO)
            goodMatches.add(mRatio);
    }
    return goodMatches;
}

```

#### **Code 5.8** Matching der Deskriptoren und Filtern der Matches

Dort wird ein Matching nach dem Fast Nearest Neighbor Verfahren, wie in Kap. [4.2.2](#) beschrieben, durchgeführt. Die Auswahl des besten Ergebnisses und somit eines wiedererkannten Objekts erfolgt, wie im gleichen Kapitel beschrieben, lediglich über die Auszählung der besten Treffer.

Nach Abschluss aller Einzelthreads wird überprüft, ob ein Objekt gefunden wurde. Bei vorliegen von mehr als einem Ergebnis wird das Objekt mit der höchsten Trefferrate ausgewählt. Für eine Antwort an den Server werden zuletzt der Name und die Wikipedia-Beschreibung des ermittelten Objekts in eine JSON-Datei geschrieben und

diese dann an den Client zurückgegeben. Das Beispiel einer solchen Antwort ist in **CODE 5.9** zu sehen.

```
{"message":"Das MacBook Pro (MBP) ist ein Macintosh-Notebook des Unternehmens Apple. Die Produktreihe wurde von Steve Jobs am 10. Januar 2006 auf der Macworld Expo vorgestellt. Die neuen Laptops lösten das 15- und 17-Zoll-Modell des PowerBook G4 ab. Das Design basiert auf dem von Jonathan Ive entwickelten Design der G4-PowerBooks. 2005 kündigte Apple an, von PowerPC-Prozessoren zu Intel CPUs zu wechseln. Das MacBook Pro markierte den ersten mobilen Mac, der mit dem Core Duo einen Intel-Prozessor besaß. [1]","createdOn":1418565277417,"name":"MacbookPro"}
```

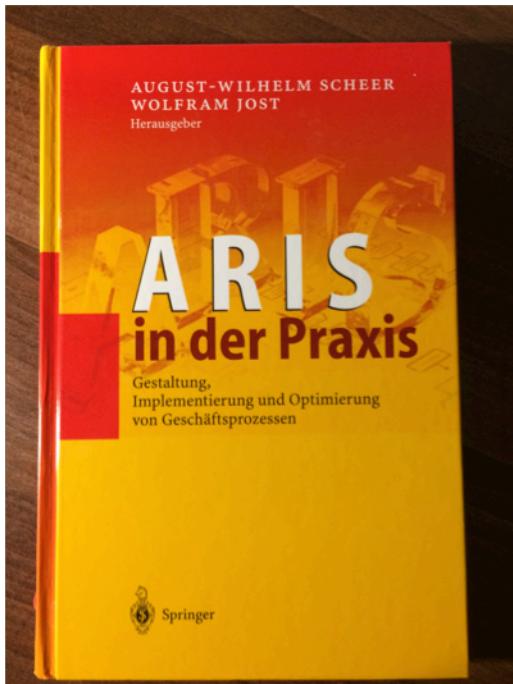
#### **Code 5.9** Beispiel einer Antwort des OCV Servers

Falls kein Objekt gefunden werden sollte oder während der Ausführung der Einzelprozesse ein Fehler auftritt, wird auch dies dem Client über eine Meldung mitgeteilt und ein entsprechender Fehlerbericht in die Log-Datei des Servers geschrieben.

#### **5.4 Auswertung der Applikation**

Die für die Applikation umgesetzte und entwickelte Applikation zur Objekterkennung und Ermittlung relevanter kontextsensitiver Informationen hat insgesamt gute Ergebnisse erbracht. Bekannte Objekte konnten in den meisten Fällen wiedererkannt werden. Nur selten erkannte der Algorithmus Objekte gar nicht wieder oder gab falsche Ergebnisse zurück.

Hierbei ist allerdings zu beachten, dass bis zu diesem Endergebnis insbesondere bei dem Hinzufügen neuer Objekte zum Server bestimmte Regeln befolgt werden müssen um dieses Ergebnis zu erreichen. So kam es in einigen Fällen immer wieder dazu, dass Objekte auf fotografierten Bildern erkannt wurden, die sich nicht auf diesen befanden. Ein Beispiel für ein solches Objekt ist das in der nachfolgenden **ABBILDUNG 5.3** gezeigte Buch.



**Abb. 5.3** Ein Bild mit hoher Fehleranfälligkeit

Der Gründe für diese Fehleranfälligkeit liegen in der Beschaffenheit des SURF-Algorithmus und diesem Bild. Beim Matching mit nachfolgend hochgeladenen Fotos wurde nicht etwa das Buch wiedererkannt sondern die das Objekt umgebende Maserung des Holztisches. Da dieser Tisch während der Anfertigung der Arbeit auf vielen Fotos immer wieder am Rand auftauchte und durch die Struktur für den Algorithmus einfach zu analysieren ist, gelangte der Matchingprozess bei diesem Objekt immer wieder zu einer auffallend hohen Übereinstimmung. Diesem Problem konnte bei später durchgeführten Versuchen Einhalt geboten werden, indem die Bilder der Objekte großzügig beschnitten wurden, damit sie tatsächlich nur das wiederzuerkennende Objekt enthalten.

Eine weitere Schwäche des Programms ist das Erkennen mehrerer Objekte auf einem Foto. So wird im Normalfall nur das signifikanteste Objekt mit der höchsten Übereinstimmungsrate auf dem Bild erkannt. Für eine Realanwendung könnte hier das Programm durch eine Liste aller erkannten Objekte erweitert werden, welche dem Nutzer die Möglichkeit gibt, eine weitere Auswahl zu treffen.

Eine zusätzliche Schwäche der Applikation liegt in der Wahl des SURF-Deskriptors, welcher auf die reine Erkennung von in Graustufen hinterlegten Bildern beschränkt ist. Dies zerstört viel Potenzial bei farbenfrohen Objekten und senkt den Wiedererkennungswert. Hier gibt es bereits Ansätze, welche den in **4.2.2** genannten SIFT-Deskriptor um Farberkennung erweitern (Van De Sande et al. 2010).

Ein Programmieransatz, welcher nicht beachtet wurde, aber bei der Suche in Veröffentlichungen zur Objekterkennung immer wieder auffiel, war der Bag of Words (BoW) oder Bag of Features (BoF) Ansatz. Dieser Ansatz ermöglicht es durch aufwendige mathematische Methoden in sehr kurzer Zeit zu bestimmen, ob sich ein fotografiertes Objekt in dem Datensatz der bekannten Objekte befindet, aber nicht um welches Objekt es sich handelt (Nowak 2006, S. 1).

Durch diesen Ansatz wäre es möglich Objekte in verschiedene Gruppen zu kategorisieren (bspw. Zahnräder, Schläuche, Batterien) und erst innerhalb dieser Gruppen dann den in dieser Arbeit vorgestellten Algorithmus zu starten. So würden gleichzeitige Geschwindigkeit und Genauigkeit des Programms gesteigert werden.

## 6 Fallstudie

In diesem Abschnitt soll nach einer kurzen Einführung in die Fallstudie beispielhaft durch eine Anwendung der implementierten Applikation geführt werden.

### 6.1 Einführung in die Fallstudie

Die hier entwickelte Applikation ist im Rahmen des GLASSROOM Projektes des Lehrstuhls Informationsmanagement und Wirtschaftsinformatik an der Universität Osnabrück entstanden. Im Rahmen des Projektes sollen die Möglichkeiten von AR- und VR-Brillen im Bereich des Maschinen- und Anlagenbaus erforscht werden.

Dafür wurde eine Applikation entwickelt, welche durch Wiedererkennung von Objekten und Bildern kontextsensitive Informationen in die Wahrnehmung des Trägers integriert und diese so für den Nutzer aufwandsarm zur Verfügung stellt.



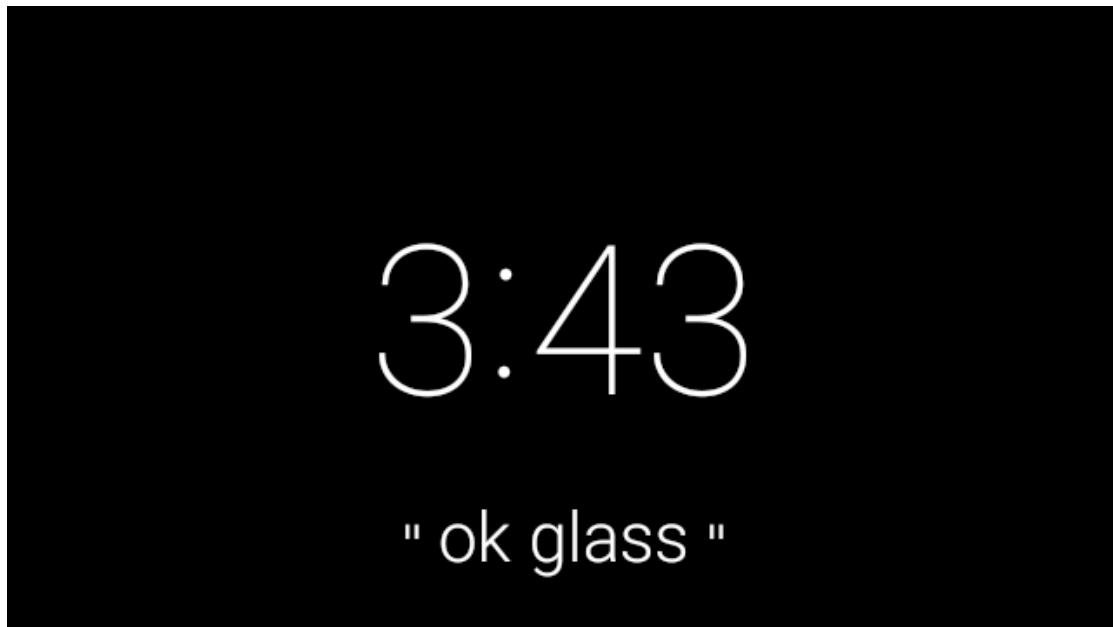
**Abb. 6.1** Die Bilder der auf dem Server hinterlegten Objekte

Für die Durchführung der Fallstudie wurden auf dem vorgestellten OCV-Server die Bilder von fünf verschiedenen Objekten hinterlegt (siehe **ABB. 6.1**). Diese wurden beim Hochladen benannt und stehen so zur Wiedererkennung zur Verfügung. Im Idealfall sollen die Objekte beim erneuten Fotografieren mit der Google Glass durch einen

Anwender wiedererkannt und die verknüpfte kontextsensitive Information dem Nutzer angezeigt werden.

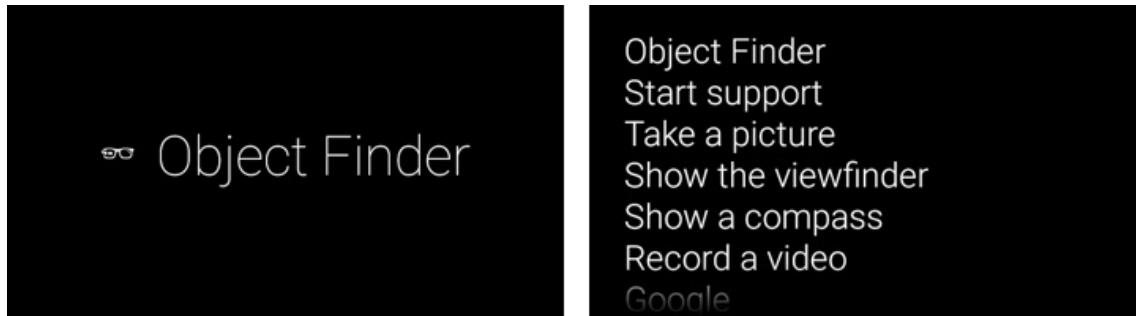
## 6.2 Beispieldurchführung

Möchte ein Nutzer nun die Applikation auf einer damit ausgestatteten Google Glass nutzen, hat er bei Ansicht des Startbildschirms (**ABB. 6.2**) zwei verschiedene Möglichkeiten, dies zu erreichen.



**Abb. 6.2** Der Startbildschirm der Google Glass

Das Programm lässt sich einerseits über den integrierten Applikationslauncher der Glass finden. Dieser wird durch ein einfaches Tippen auf das Touchpad erreicht und lässt sich durch Scrollen in horizontaler Richtung durchblättern. Die gewünschte Applikation lässt sich dann durch ein weiteres Tippen auf das Touchpad starten. Eine weitere Möglichkeit, in die Applikation zu gelangen, ist die Spracherkennung der Google Glass zu nutzen. Dafür öffnet sich nach Ansage des Startbefehls „ok glass“ ein Menü mit verschiedenen Auswahlmöglichkeiten für Anschlussbefehle. Auch in diesem Menü ist die Applikation Object Finder zu finden, welche die hier implementierte Anwendung wiederspiegelt. Beide Möglichkeiten sind in der **ABB. 6.3** aufgeführt.

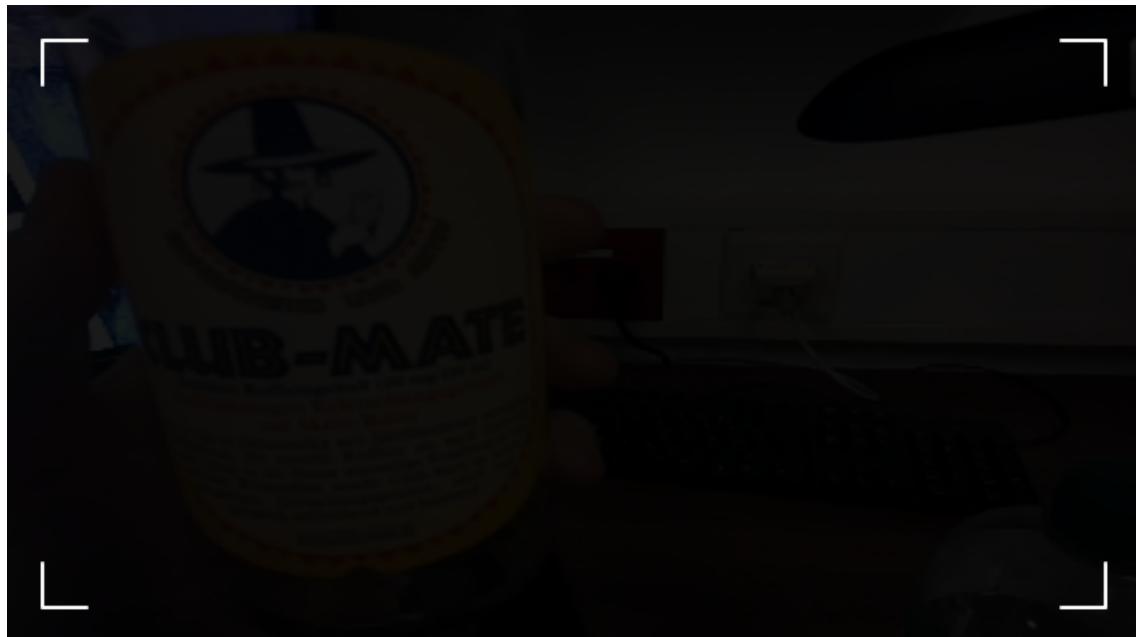


**Abb. 6.3** Die beiden Startmöglichkeiten der Applikation



**Abb. 6.4** Der Standardbildschirm der Applikation

Nach Start der Applikation gelangt der Nutzer in eine einfache Ansicht, in der in Echtzeit das Bild der Gerätekamera auf dem Bildschirm angezeigt wird (siehe **ABB 6.4**). Dieser Bildschirm ist als Sucher der Applikation zu verstehen. Durch ein einfaches Tippen auf das Touchfeld des Gerätes kann der Nutzer ein Foto aufnehmen.



**Abb. 6.5** Bestätigung des Fotografierbefehls

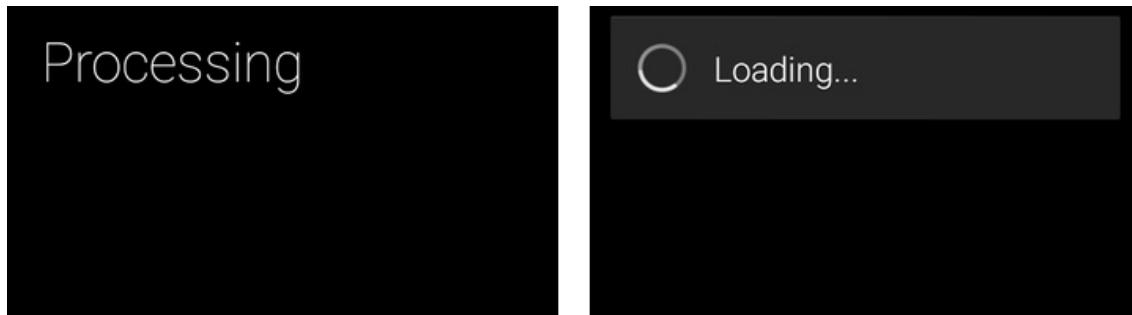
Die Nutzereingabe wird von der Applikation durch ein kurzes Einblenden eines imitierten Suchfeldes bestätigt (siehe **ABB. 6.5**). Nach der Aufnahme des Bildes wird dem Nutzer das aufgenommene Bild zur Bestätigung angezeigt. Hier ist es dem Nutzer freigestellt, das Bild durch erneutes Tippen zu akzeptieren oder aber durch einfaches vertikales Wischen von oben nach unten zu verwerfen und ein neues Bild aufzunehmen.



**Abb. 6.6** Vorschaubild mit Akzeptanzanfrage

Akzeptiert der Nutzer das aufgenommene Bild, wird die Nutzerinteraktion gesperrt und die Applikation beginnt wie in **5.2** vorgestellt mit dem asynchronen Abspeichern des

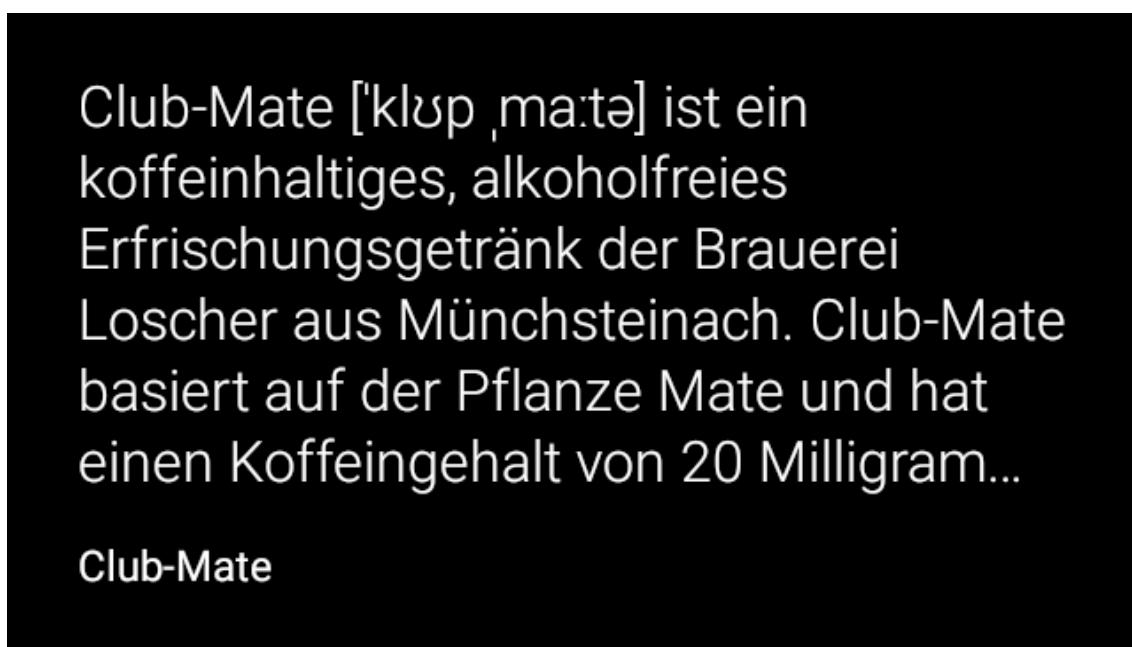
Bildes, um dieses im Anschluss in einem weiteren asynchronen Prozess direkt auf den Server hochzuladen. Im Verlauf dieser beiden Prozesse werden dem Nutzer zwei unterschiedliche Ladebildschirme gezeigt, welche die beiden Teilabschnitte des Prozesses signalisieren sollen. Der erste zeigt dabei den Vorgang des Abspeicherns und der zweite den des Hochladens (siehe **Abb. 6.7**).



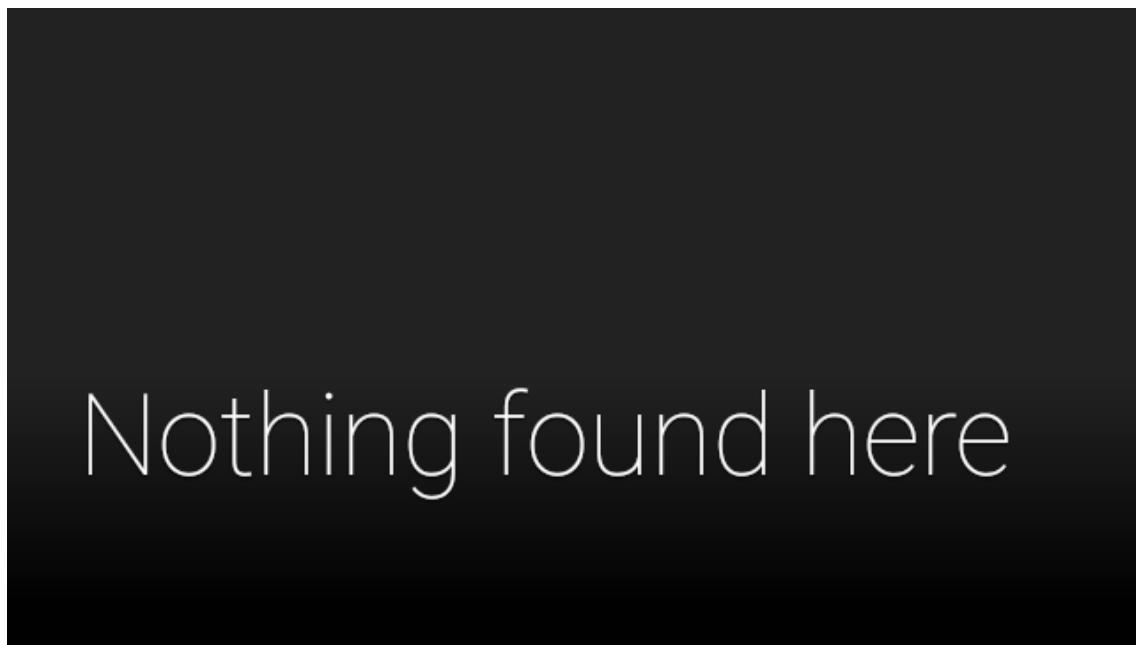
**Abb. 6.7** Die beiden Ladebildschirme der Applikation

Nach Antwort des Servers wird diese unverzüglich von der Applikation ausgewertet und das – in diesem Fall richtige – Ergebnis aufbereitet und dem Nutzer angezeigt (siehe **Abb. 6.8**). Sollte keines der hinterlegten Objekte die minimale Anzahl der Matches überschritten haben, würde dem Nutzer der Bildschirm aus **Abb. 6.9** angezeigt.

Der Nutzer hat im Anschluss die Möglichkeit, entweder durch einfaches Tippen einen neuen Durchlauf der Applikation zu starten oder aber die App durch vertikales Streichen zu beenden und so zum Hauptbildschirm der AR-Brille zurück zu gelangen.



**Abb. 6.8** Das Ergebnis der Anfrage



**Abb. 6.9** Das negative Ergebnis einer anderen Anfrage

## 7 Fazit und Ausblick

In der vorliegenden Arbeit wurde ausführlich auf die erarbeitete Applikation zur Erkennung von Objekten und der damit verbundenen Gewinnung und Anzeige von kontextsensitiver Information eingegangen. Dazu wurde Kontextsensitivität definiert und danach beispielhaft auf verschiedene Vertreter von kontextsensitiven Systemen eingegangen. Im Anschluss wurde die Google Glass als Vertreter eines Geräts der Augmented Reality vorgestellt, um die beiden Forschungsbereiche der Kontextsensitivität und der Augmented Reality im Anschluss übereinander zu legen und so die Ideenfindung der umgesetzten Applikation darzulegen. Diese Applikation wurde im Anschluss umrissen, um dann genauer auf die genutzten Algorithmen, Bibliotheken und – in Auszügen – auf die Implementation einzugehen. Die Beispielanwendung der Applikation wurde zum Schluss der Arbeit in einer kurzen Fallstudie dargelegt, um eine solche Nutzung zu verdeutlichen.

Abschließend ist zu konstatieren, dass ein Potenzial in der Nutzung von kontextsensitiven Informationen in der Industrie vorhanden ist. Durch die Nutzung von darauf spezialisierten Algorithmen ist es möglich, Objekte in Fotos und somit der den Nutzer umgebenden Realität wiederzuerkennen und diese mit Informationen zu verknüpfen. So können auf dieser Basis weitergehende Information hinterlegt oder bspw. anschließende Arbeitsschritte eingeleitet werden.

Auch wurden die Schwächen der Plattform Google Glass insbesondere im Bereich der Akkulaufzeit und Rechenleistung aufgezeigt und die Vorteile einer Client- / Serverlösung für eine kontextsensitive Anwendung der Augmented Reality erläutert.

Damit ist ein Beitrag geleistet worden um im Rahmen des GLASSROOM Projektes ein Gerät der Augmented Reality zukünftig als Lernunterstützung zu nutzen und durch Objekt- und Bilderkennung dem Träger und Nutzer wichtige kontextsensitive Informationen zu liefern.

Für zukünftige Arbeiten ist vor allem eine Erweiterung der Applikation in Richtung noch genauerer Information möglich. So ist zum Beispiel die Nutzung von zusätzlichen Sensoren denkbar, um zum Beispiel auf Grundlage der Ortsdaten des Nutzers eine Vorauswahl der hinterlegten Objekte zu treffen. Auch im Bereich der Objekterkennung sind Erweiterungen insbesondere mit Blick auf die Wahl des Matching- und Analysevorganges denkbar. Hier sollten auf Dauer modernere Algorithmen des Matchings genutzt werden und dazu die Forschung im Bereich der Computer Vision berücksichtigt werden. Hier wären vor allem eine Erweiterung des Match- und

Suchprozesse durch neue Technologien, Vorauswahl durch den genannten BoF-Ansatz oder sogar ein Einbinden und Abgleichen über 3D-Modelle denkbar.

Aber auch eine Ergänzung um weitere Informationsträger wie QR Codes neben der reinen Objekterkennung wären Ansätze, die als Verbesserungsvorschlag für eine Anwendung dieser Art denkbar wären.

## Literaturverzeichnis

- Abowd, Gregory D.; Dey, Anind K.; Brown, Peter J.; Davies, Nigel; Smith, Mark; Steggles, Pete (1999) Towards a better understanding of context and context-awareness. In Gellersen, Hans-W. (Hrsg) Handheld and ubiquitous computing. Berlin, Heidelberg, Springer Berlin Heidelberg, 304–307.
- Alahi, Alexandre; Ortiz, Raphael; Vandergheynst, Pierre (2012) FREAK: Fast Retina Keypoint. 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 510–517.
- Ashford, Robin (2010) QR codes and academic libraries. College & Research Libraries News, 71 (10):526–530.
- Azuma, Ronald T. (1997) A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments, 6 (4):355–385.
- Baldauf, Matthias; Dustdar, Schahram; Rosenberg, Florian (2007) A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, 2 (4):263–277.
- Baumgart, Bruce Guenther (1974) Geometric Modeling for Computer Vision. Stanford University, 136.
- Bay, Herbert; Ess, Andreas; Tuytelaars, Tinne; Gool, Luc Van (2008) Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding, 110 (3):346–359.
- Bellavista, Paolo; Corradi, Antonio; Fanelli, Mario; Foschini, Luca (2012) A survey of context data distribution for mobile ubiquitous systems. ACM Computing Surveys (CSUR), 44 (4):24.
- Bilton, Nick (2012) Google to Sell Heads-Up Display Glasses by Year's End. <http://bits.blogs.nytimes.com/2012/02/21/google-to-sell-terminator-style-glasses-by-years-end/?ref=technology>, abgerufen am 20.11.2014.
- Bradski, Gary; Kaehler, Adrian (2008) Learning OpenCV Computer Vision with the OpenCV Library. 1. Auflage, Sebastopol, CA, O'Reilly, Inc.

Canny, John (1986) A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8 (6):679–698.

Chang, Yao-Jen; Tsai, Shih-Kai; Chang, Yao-Sheng; Wang, Tsen-Yung (2007) A novel wayfinding system based on geo-coded QR codes for individuals with cognitive impairments. *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility - Assets '07*. New York, New York, USA, ACM Press, 231–232.

Chen, Harry L. (2004) An intelligent broker architecture for pervasive context-aware systems. University of Maryland, .

Cho, Eunjoon; Myers, Seth A.; Leskovec, Jure (2011) Friendship and mobility: user movement in location-based social networks. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*. New York, New York, USA, ACM Press, 1082.

Cipolla, Thomas M; Lake, Ballston; Mundy, Joseph L (1982) Optical Character Recognition. United States, .

DENSO WAVE (2014) QRcode.com. <http://www.qrcode.com/en/index.html>, abgerufen am 16.10.2014.

Dey, Anind K. (2001) Understanding and Using Context. *Personal and Ubiquitous Computing*, 5 (1):4–7.

Dey, Anind K.; Salber, Daniel; Abowd, Gregory D.; Futakawa, Masayu (1999) The Conference Assistant: combining context-awareness with wearable computing. *Digest of Papers. Third International Symposium on Wearable Computers*. IEEE Comput. Soc, 21–28.

Djurknic, GM; Richton, RE (2001) Geolocation and assisted GPS. *Computer*, 34 (3):123–125.

Eclipse Foundation (2014) Chapter 1. Introducing Jetty. <http://www.eclipse.org/jetty/documentation/current/introduction.html#what-is-jetty>, abgerufen am 12.12.2014.

Feng, Steve; Caire, Romain; Cortazar, Bingen; Turan, Mehmet; Wong, Andrew; Ozcan, Aydogan (2014) Immunochromatographic diagnostic test analysis using Google Glass. *ACS nano*, 8 (3):3069–3079.

Fielding, Roy Thomas (2000) Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 162.

Gao, Huiji; Tang, Jiliang; Liu, Huan (2012) Exploring Social-Historical Ties on Location-Based Social Networks. *ICWSM*. 114–121.

Google (2012) Google I/O 2012. <https://developers.google.com/events/io/2012/>, abgerufen am 07.12.2014.

Google (2014a) The Glass Explorer Programm. <http://www.google.com/glass/start/>, abgerufen am 29.12.2014.

Google (2014b) Tech specs. <https://support.google.com/glass/answer/3064128?hl=de>, abgerufen am 26.11.2014.

Google (2014c) Android Lollipop. <https://developer.android.com/about/versions/lollipop.html>, abgerufen am 29.12.2014.

Google (2014d) Glass Platform Release Notes. <https://developers.google.com/glass/release-notes>, abgerufen am 02.12.2014.

Google (2014e) Configuring Gradle Builds. <https://developer.android.com/tools/building/configuring-gradle.html>, abgerufen am 09.01.2015.

Huang, Zhanpeng; Hui, Pan; Peylo, Christoph; Chatzopoulos, Dimitris (2013) Mobile augmented reality survey: a bottom-up approach. *Arxiv.Org*. 35.

Indulska, Jadwiga; Sutton, Peter (2003) Location management in pervasive systems. *ACSW Frontiers '03 Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21*. Adelaide, Australia, Australian Computer Society, 143–151.

Juan, L; Gwun, O (2009) A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3 (4):143–152.

- Junglas, Iris A.; Watson, Richard T. (2008) Location-based services. *Communications of the ACM*, 51 (3):65–69.
- Kölmel, Dr. Bernhard; Yellowmap AG (2003) Location Based Services. Workshop Mobile Commerce. 88–101.
- Lee, Sangkeun; Chang, Juno; Lee, Sang-goo (2010) Survey and Trend Analysis of Context-Aware Systems. *Information-An International Interdisciplinary Journal*, 14 (2):527–548.
- Leutenegger, Stefan; Chli, Margarita; Siegwart, Roland Y. (2011) BRISK: Binary Robust invariant scalable keypoints. 2011 International Conference on Computer Vision. IEEE, 2548–2555.
- Lins, Caio Novaes; Teixeira, João Marcelo; Rafael, Alves Roberto; Teichrieb, Veronica (2014) Development of Interactive Applications for Google Glass. *Tendências e Técnicas em Realidade Virtual e Aumentada*. 4. Auflage, 167–188.
- Lowe, David G. (1999) Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision. IEEE, 1150–1157 vol.2.
- Lowe, David G. (2004) Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60 (2):91–110.
- Mehrotra, Rajiv; Nichani, Sanjay; Ranganathan, N. (1990) Corner detection. *Pattern Recognition*, 23 (11):1223–1233.
- Morril, Dan (2008) Announcing the Android 1.0 SDK, release 1. <http://android-developers.blogspot.be/2008/09/announcing-android-10-sdk-release-1.html>, abgerufen am 02.12.2014.
- Muja, Marius; Lowe, David G. (2009) Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. *VISAPP* (1). 331–340.
- Nowak, Eric (2006) Computer Vision – ECCV 2006. Berlin, Heidelberg, Springer Berlin Heidelberg.
- opencv dev team (2014) OpenCV API Reference. <http://docs.opencv.org/modules/refman.html>, abgerufen am 09.10.2014.

- Perera, Charith; Zaslavsky, Arkady; Christen, Peter; Georgakopoulos, Dimitrios (2014) Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16 (1):414–454.
- Portman, Eric A.; Gailey, Michael L.; Holmes, Chad S.; Burgiss, Michael J.; Smith, Angela King; Pitts III, Ashton F.; Dempsen, Stephen L.; Che, Vinny Wai-yan (2005) Location-based services. USA, 16.
- Rouillard, José (2008) Contextual QR Codes. 2008 The Third International Multi-Conference on Computing in the Global Information Technology (iccgi 2008). IEEE, 50–55.
- Saha, Amit Kumar (2008) A Developers First Look At Android. *Linux For You* (January), (January):48–50.
- Sande, Koen E. A. Van De; Gevers, Theo; Snoek, Cees G. M. (2010) Evaluating color descriptors for object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 32 (9):1582–1596.
- Schaeffer, Cameron (2013) A comparison of keypoint descriptors in the context of pedestrian detection: freak vs. surf vs. brisk. Stanford University, 5.
- Schilit, Bill N.; Theimer, Marvin M. (1994) Disseminating active map information to mobile hosts. *IEEE Network*, 8 (5):22–32.
- Schmidt, Albrecht; Laerhoven, Kristof van (2001) How to build smart appliances? *IEEE Personal Communications*, 8 (4):66–71.
- Shneier, Michael (1983) Using Pyramids to Define Local Thresholds for Blob Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5 (3):345–349.
- Sidla, Oliver; Kottmann, Michal; Benesova, Wanda (2011) Real-time pose invariant logo and pattern detection. *Intelligent Robots and Computer Vision XXVIII: Algorithms and Techniques*, 78780C. 8.
- Singh, Inderjeet; Leitch, Joel; Wilson, Jesse (2014) gson. <https://sites.google.com/site/gson/gson-user-guide#TOC-Goals-for-Gson>, abgerufen am 12.12.2014.

- Stevens, Tim (2013) Google announces Glass Developer Kit, will enable offline apps and direct hardware access. <http://www.engadget.com/2013/05/16/google-glass-developer-kit/>, abgerufen am 30.11.2014.
- Swain, Michael J.; Ballard, Dana H. (1991) Color indexing. *International Journal of Computer Vision*, 7 (1):11–32.
- Szeliski, Richard (2010) *Computer Vision*. London, Springer London.
- Texas Instruments (2012) OMAP4430 Multimedia Device. 443.
- Torborg, Scott; Simpson, Star (2012) What's inside Google Glass? <http://www.catwig.com/google-glass-teardown/>, abgerufen am 16.10.2014.
- UMTS Forum (2001) Report No. 13. London, 100.
- Walsh, Andrew (2010) QR Codes – using mobile phones to deliver library instruction and help at the point of need. *Journal of Information Literacy*, 4 (1):55–65.
- Want, Roy; Hopper, Andy; Falcão, Veronica; Gibbons, Jonathan (1992) The active badge location system. *ACM Transactions on Information Systems*, 10 (1):91–102.

## **Abschließende Erklärung**

Ich versichere hiermit, dass ich diese Bachelorarbeit *Einblendung von kontextsensitiven Inhalten auf der Google Glass* selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel und Quellen angefertigt habe, sowie den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Osnabrück, den 9. Januar 2015

Jannik Hoffjann