

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

Abgabetermin: 2015-01-22

Zusammenfassung / Expose

In der Arbeit sollen die Möglichkeiten der Einblendung von kontextsensitiven Inhalten auf einem in das Sichtfeld integrierten Gerät am Beispiel der Google Glass erprobt werden. Dabei sollen nach Einführung und Vorstellung des Geräts und der mitgelieferten Software verschiedene Möglichkeiten der Kontextsensitivität (engl. Context-Awareness) erörtert werden und ihren Nutzbarkeit auf dem Bereich der tragbaren Geräte erfragt werden.

Es sollen die verschiedenen Möglichkeiten der Context-Awareness erläutert werden und dabei versucht werden die Anforderungen der Einzelnen mit den Möglichkeiten der Google Glass abzugleichen. Beispielsweise sollen diese Möglichkeiten an bereits erhältlichen Applikationen erläutert werden. Durch sorgfältige Auswahl sollen die verschiedenen Arten dargelegt werden um so eine Auswahl der am besten Geeigneten zu ermöglichen.

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

Vorstellbar wäre an dieser Stelle zum Beispiel die Implementation einer mobilen Applikation auf Grundlage von Open CV (opencv dev team 2014) und einer der implementierten Keypointerkennungen wie zum Beispiel SURF (Bay et al. 2008), FREAK (Alahi et al. 2012) und BRISK (Leutenegger et al. 2011). Diese bieten durch vielfältige Möglichkeiten des Matchings, Möglichkeiten der Wiedererkennung und Auswertung von Grafiken, welche die Umsetzung einer kontextsensitiven Anwendung auf der Google Glass ermöglichen könnten.

Durch diese abschließende Implementation und eine Auswertung der Ergebnisse soll ein erster Versuch der Einblendung von kontextsensitiven Inhalten auf der Google Glass erbracht werden und Möglichkeiten zu weiteren Nutzung des Geräts aufgezeigt werden.

Inhaltsverzeichnis

Zusammenfassung / Expose	II
Abbildungsverzeichnis	V
Tabellenverzeichnis.....	VI
Codeverzeichnis	VII
Abkürzungsverzeichnis	VIII
Symbolverzeichnis	X
1 Einleitung / Motivation.....	1
2 Kontextsensitivität	3
2.1 Definition.....	3
2.2 Möglichkeiten der Kontextsensitivität	4
2.2.1 Marker / QR Codes	5
2.2.2 Location-Based Services.....	6
2.2.3 Objekt- und Bilderkennung.....	8
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	19
5.1 Vorstellung der Implementation und ihrer Komponenten.....	19
5.2 Glass Client	21
5.3 OCV Server	24
5.3.1 Hinzufügen eines neuen Objekts.....	24
5.3.2 Analyse eines gesendeten Bildes	26
5.4 Auswertung der Applikation	28
6 Fallstudie	30
6.1 Einführung in die Fallstudie	30
6.2 Beispieldurchführung	31
7 Fazit und Ausblick.....	36
Literaturverzeichnis.....	37
8 Anhang.....	1
A Unterkapitel des Anhangs.....	1
B Zweites Unterkapitel des Anhangs	1

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. 4.3 Fast Approximate Neighbor Matching der beiden Bilder	18
Abb. 5.1 UML-Darstellung des Glass Clients.....	21
Abb. 5.2 UML-Darstellung des OCV Servers	24
Abb. 5.3 Ein Bild mit hoher Fehleranfälligkeit.....	29
Abb. 6.1 Die Bilder der auf dem Server hinterlegten Objekte	30
Abb. 6.2 Der Startbildschirm der Google Glass.....	31
Abb. 6.3 Die beiden Startmöglichkeiten der Applikation	32
Abb. 6.4 Der Standardbildschirm der Applikation	32
Abb. 6.5 Bestätigung des Fotografierbefehls	33
Abb. 6.6 Vorschaubild mit Akzeptanzanfrage	33
Abb. 6.7 Die beiden Ladebildschirme der Applikation	34
Abb. 6.8 Das Ergebnis der Anfrage	35
Abb. 6.9 Das negative Ergebnis einer anderen Anfrage	35

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	19

Codeverzeichnis

Code 5.1	Asynchroner Uploadprozess	22
Code 5.2	Erstellung der Anfrage und Auswertung der Antwort.....	23
Code 5.3	Aktualisierung der Benutzeroberfläche	23
Code 5.4	Extrahierung des ersten Absatzes eines Wikipedia-Artikels	25
Code 5.5	Keypointerkennung und Deskriptorextraktion	25
Code 5.6	Serialisierung einer Deskriptormatrix.....	26
Code 5.7	Beispiel eines abgespeicherten Objekts.....	26
Code 5.8	Matching der Deskriptoren und Filtern der Matches.....	27
Code 5.9	Beispiel einer Antwort des OCV Servers	28

Abkürzungsverzeichnis

aGPS	assisted Global Positioning System
API	Application Programming Interface
BRISK	Binary Robust invariant scalable keypoints
bspw.	Beispiele
bzw.	beziehungsweise
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

Symbolverzeichnis

1 Einleitung / Motivation

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 und den beiden hinter ihnen stehenden Großunternehmen Google und Facebook erneuten 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 bietet die Google Glass dabei die Chance, durchgängig in das Sichtfeld des Trägers integriert zu sein. So bieten sich neue Möglichkeiten von Nutzerinteraktion die den Gedanken des anywhere und anytime auf eine neue Ebene bringen könnten, da die benötigte Handlung durch den Nutzer minimiert wird.

Anders als die Oculus Rift ist die Google Glass dabei ein Gerät, welches nicht zum Anzeigen virtueller Welten bzw. der virtuellen Darstellung realer Umgebungen entwickelt wurde. 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 vielzählige 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 dringend notwendige Information für den Arbeitsfluss zu integrieren, ohne dabei die Arbeit unterbrechen zu müssen.

Diese Arbeit ist ein Teil des Glassroom Projekts, welches vom Lehrstuhl für Informationsmanagement und Wirtschaftsinformatik (IMWI) der Universität Osnabrück in Kooperation mit namhaften Partnern 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. So sollen bisherige, veraltete Methoden in diesen Bereichen auf Dauer ersetzt werden.

Inwieweit eine solche kontextsensitive Erweiterung mit der Google Glass, dem heute am weitesten verbreiten und bekanntesten Beispiel für ein Gerät der Augmented Reality, möglich ist, soll Bestandteil dieser wissenschaftlichen Arbeit sein. Dafür werden nach Definition von Kontextsensitivität und der Vorstellung der Google Glass als Vertreter der Augmented Reality in die Funktionsweise und die Idee der

implementierten Applikation eingeführt. Die Implementation dieser 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. Die Arbeit schließt mit einem Fazit und gibt Ausblick auf die weiteren sich ergebenden 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 erdachten 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 anhand dieser geortet. Mit dieser Information konnten dann 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 dem Kontext Aufmerksamkeit zu schenken.

Die heute in der Wissenschaft weitgehenden anerkannten Definitionen für Kontext und Kontextsensitivität kommen von Abowd und Dey (1999; Perera et al. 2014, S. 414); sie beschreiben 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)

Anhand dieser Definition, mit der die Autoren damals klar den Kontext als Information abgrenzten, die für den Nutzer in seiner Interaktion relevant ist, gelang es ihnen auch eindeutig eine Definition für kontextsensitive Systeme bzw. Applikationen zu nennen.

„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 danach noch Versuche kontextsensitive Systeme zu definieren (Dey 2001, S. 5), die meisten wissenschaftlichen Arbeiten zu dem Thema bauen aber auf der oben genannten Definition auf (Baldauf et al. 2007, S. 264; Perera et al. 2014, S. 414; Lee et al. 2010, S. 1; Dey et al. 1999, S. 2); diese soll daher hier genutzt werden.

Heute wird die Kontextsensitivität als Unterfeld des Ubiquitous (bzw. Pervasive) Computing gesehen (Baldauf et al. 2007, S. 263–264; Perera et al. 2014, S. 414; Bellavista et al. 2012, S. 2) und spielt in diesem eine wichtige Rolle, 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 fällt auf, dass es sich bei Kontextsensitivität um ein enorm vielschichtiges Thema handelt.

Baldauf et al. (2007) und Perera et al. (2014) eine Großzahl 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 all dieser den Rahmen dieser Arbeit deutlich sprengen würde, wird im Folgenden die Kategorisierung nach der Stelle der Informationsverarbeitung sowie die Art des Sensors zur Informationsgewinnung beispielhaft betrachtet.

<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 daher 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 Verarbeitung der Information geht dabei von direkt im Gerät verankert, über eine Verarbeitung auf implementierten Zwischenebenen, aber immer noch im Gerät 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).|

Jannik Hoffmann 20.12.14 12:12

Kommentar [1]: PDF-Dokument fehlt noch

Ein weiterer Ansatz ist die Einteilung der kontextsensitiven Systeme nach Kontextart und den zugehörigen Sensoren ([TABELLE 1.1](#)). Die Idee hierhinter 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 eine GPS-Sensor den Standort auf der Welt. Weiterführend lassen sich diese Informationen verknüpfen, um so sehr genaue Ergebnisse zu liefern. (Schmidt und van Laerhoven 2001, S. 67–68)

Jannik Hoffmann 20.12.14 12:12

Kommentar [2]: Die Priorität dieser Arbeit wird dabei insbesondere auf den Kategorien der *Middleware infrastructure* und der *Context server* liegen.

Jannik Hoffmann 20.12.14 12:12

Kommentar [3]: Unklar

Im Rahmen dieser Arbeit soll eine Kombination der beiden Kategorisierungen zum Einsatz kommen. Dabei wird ein besonderer Schwerpunkt auf Middleware Infrastructure und 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 Marker / QR Codes

Als spezielle Art der visuellen Kontextsensitivität sollten Identifizierungen von Objekten und Orten anhand von Markern gesehen werden. Die dafür am häufigsten genutzten Möglichkeiten sind die QR (Quick Response) Codes (Quelle benötigt).|

Jannik Hoffmann 20.12.14 12:12

Kommentar [4]: Quelle!

QR Codes sind zweidimensionale matrixbasierte Barcodes die Information sowohl vertikaler als auch in horizontaler Richtung enthalten (siehe [ABB. 2.1](#)). Sie wurden entwickelt um schnell Informationen durch geeignete Scannersoftware auszulesen und diese dem Nutzer zur Verfügung zu stellen. Im Gegensatz zu eindimensionale Barcodes, wie man sie zum Beispiel aus dem Einzelhandel kennt, haben sie dabei ein stark gesteigertes Speichervermögen und es ist 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 Lesbarkeit trotz teilweiser Beschädigung unterscheiden. (QRcode.com | DENSO WAVE)

QR Codes finden heute vielerlei 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 aber 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, festgestellte 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 weitesten verbreiteten und bekanntesten Beispiele unter den kontextsensitiven Anwendungen. Dafür spricht auch, 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 2005; Junglas und Watson 2008) zum Thema erschienen sind. Zudem fällt auf, dass ein Großteil der Forschung zum Thema aus dem vergangenen Jahrzehnt stammen, was darauf schließen lässt, dass die Technik heute so etabliert ist, dass keine weiteren grundlegenden Forschungsarbeiten benötigt werden. 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, wie bspw. Foursquare (siehe [ABB 2.2](#)) oder Yelp und Applikationen die ihrem Kerndienst Extraintformationen

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

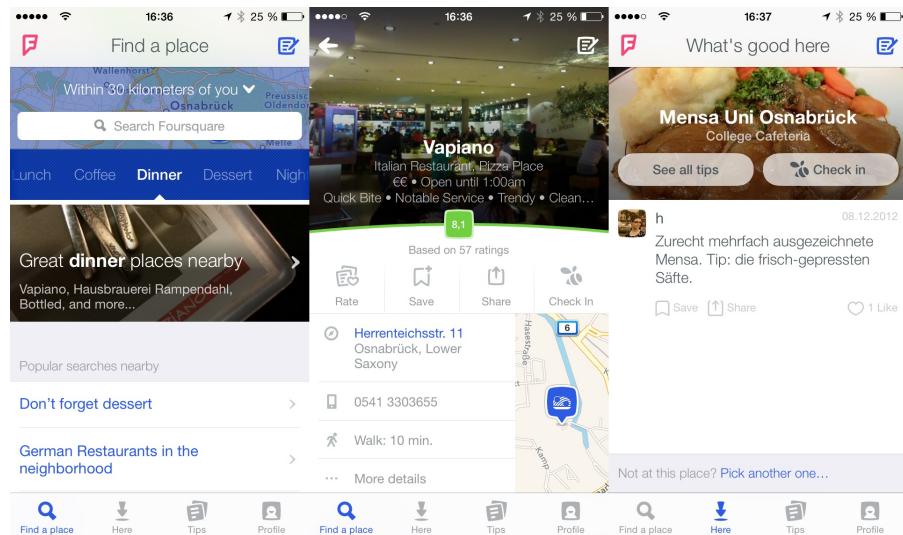


Abb. 2.2 Foursquare auf iOS 8.1.1

Das UMTS-Forum beschreibt 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 Personen zu ermitteln. Zudem ermöglicht es dem Nutzer, seine eigene Position zu ermitteln. (UMTS Forum 2001, S. 35).

Diese Lokalisierung geschieht dabei durch verschiedene Techniken, die sich sehr 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), aber auch eine Ermittlung anhand von Nutzereingaben oder durch andere Applikationen ist denkbar (Indulska und Sutton 2003, S. 1). Hierbei ist allerdings zu unterscheiden ob die Lokalisierung unter freiem Himmel und so über Satelliten oder Sendemasten geschehen kann. Oder ob 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 aus Mitte der 70er Jahre zu finden sind (Baumgart 1974). Heute ist die Computer Vision eng in viele Anwendungsbereich der Informationstechnologie, wie der Automation, Robotik oder auch der Medizininformatik eingebunden und wird dort beispielsweise zur Steuerung von Montagearmen in der Industrie genutzt (Szeliski 2010). Aber auch im Endnutzertbereich lassen sich Applikationen der Computer Vision finden.

So existieren Applikationen, welche Texterkennung (engl. Optical Character Recognition (OCR)) 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 Sudoku Rätsel zu lösen oder Informationen zu bekannten Gemälden in Museen zu suchen.

Objekt- und Bilderkennung nutzt Licht- und Bildsensoren zur Erkennung und Wiedererkennung von Objekten. Sie alle haben gemein, dass sie Bilder anhand eines speziellen Algorithmus analysieren und beschreiben und in diesen Beschreibungen dann Übereinstimmungen mit den Beschreibungen anderer Bilder zu suchen. Für die Gewinnung der Beschreibung gibt es verschiedene Ansätze, die von Kantenerkennung und Eckenerkennung bis zur sogenannten Blob Erkennung reicht. All diese Erkennungs- und Beschreibungsalgorithmen sind aber grob unter den Algorithmen der Merkmalserkennung einzuordnen und sind zum Teil nicht alleinig zur Objekt- und Bilderkennung entwickelt worden.

Jannik Hoffmann 20.12.14 12:12

Kommentar [5]: Beispiel + Quelle

Jannik Hoffmann 20.12.14 12:12

Kommentar [6]: Quelle

Jannik Hoffmann 20.12.14 12:12

Kommentar [7]: Quellen!

3 Google Glass

3.1 Die Google Glass als Vertreter der Augmented Reality

Bereits seit der Mitte der 90er Jahre ist die erweiterte Realität (engl. Augmented Reality) ein intensiv beachteter Forschungsbereich. Hierbei geht es um die Integration von virtuellen Elementen in die Realität und anders als in der 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 Geräte Kategorie als Mobile Augmented Reality (MAR) und unterteilen diese weiter in 6 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 (Google to Sell Heads-Up Display Glasses by Year's End - NYTimes.com) 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 dann 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.

Jannik Hoffmann 20.12.14 12:12
Kommentar [8]: Quelle

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

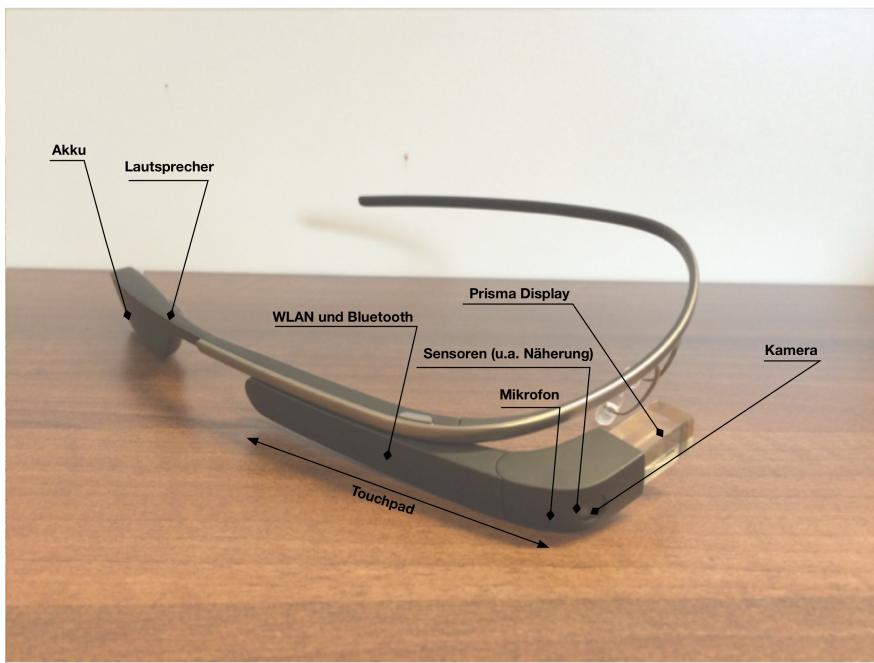


Abb. 3.1 Beschriftete Google Glass (in Anlehnung an (Feng et al. 2014, S. 3070))

Bei der Google Glass handelt es sich um eine über wahlweise Sprachbefehle oder ein Touchpad bedienbare AR Brille. Sie ist mit WLAN und Bluetooth Modulen 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 Geräts 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. An der 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 2014a). Gesteuert wird das Gerät von einem ARM-Prozessor mit bis zu 1000 Megahertz (MHz) (Texas Instruments 2012, S. 125).

Jannik Hoffmann 20.12.14 12:12

Kommentar [9]: Kurzerklärung

Das Gerät hat bei starker Beanspruchung, wie zum Beispiel dem Aufnehmen eines 720p-Videos oder der durchgehende Berechnung von aufwändigen Algorithmen eine ungefähre Akkulaufzeit von einer Stunde. Dies lässt darauf schließen, dass es nicht als allzeit eingeschaltetes, sondern als nur im Bedarfsfall gefragtes Gerät konzipiert wurde (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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [10]: Quelle

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 bekannten Methoden und Bibliotheken zurückzugreifen (Saha 2008, S. 49).

Zum Ansprechen der speziellen Komponenten der AR-Brille hat Google zu dem 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 2014b).

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-Elementen, den Cards, zu ermöglichen.

Jannik Hoffmann 20.12.14 12:12

[Kommentar \[11\]](#): Quelle

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

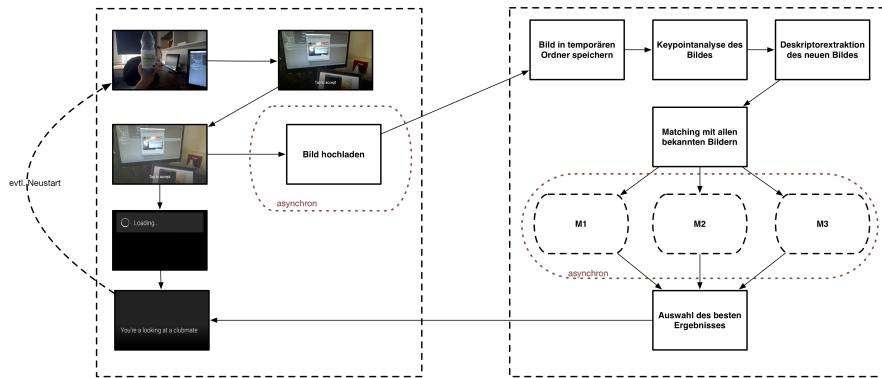


Abb. 4.1 Darstellung der Funktionsweise der kontextsensitiven Applikation

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 werden zu können. Hier wird anhand der Position des Nutzers über GPS die Himmelsrichtungen ermittelt, es werden also ortsbasierte Informationen angezeigt und somit die Wahrnehmung des Nutzer um in seinem Kontext relevante Informationen ergänzt, was sowohl der in 2.2.1 angesprochenen Definition eines Location-Based Services, als auch der in 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 Glas, um die über Sensoren ermittelten Außenwelteinflüsse auszuwerten und so dem Nutzer kontextsensitive Informationen anzubieten. So sind Applikationen erhältlich, die Filmtrailer beim Betrachten eines Posters einblenden, oder anhand der Position des Nutzers relevante Wikipediaartikel anzeigen.

Jannik Hoffmann 20.12.14 12:12

Kommentar [12]: Beispiel für solch eine App (<http://glass-apps.org/google-glass-application-list>)

Die Idee zur Applikation dieser Arbeit 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 Information 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.

Um diese rechenintensiven und aufwendigen Prozesse von der Glass zu nehmen, wurde jegliche Rechenarbeit auf einen Server in der Cloud ausgelagert. Auf der Glass selber verbleiben nur die den User direkt betreffenden Prozesse wie UI-Darstellung, Fotografieren, Upload des Bildes und Anzeigen der Ergebnisse. Eine weitere Einschränkung entstand durch die Auslagerung der benötigten Methoden zur SURF Keypointerkennung und Deskriptorextraktion in der angewandten Computer Vision Bibliothek in ein Nonfree-Modul (opencv dev team 2014). Diese ist bis heute nur über Umwege in der Android-Version verfügbar.

Der implementierte Matchingprozess erfolgt dabei in 4 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, sowie das Problem der inkonsistenten und redundanten Datenhaltung. 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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [13]: http://web.guohuiwang.com/technical-notes/opencv_nonfree_android_jni_demo

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

Um dem Nutzer neben einer 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 zu der Objektdatenbank automatisch ein und ermittelt den ersten Absatz zu einem mitangegebenen 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 wurde 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). 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 (<https://github.com/bytedeco/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 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-Keypointerkennung auf einem Logo

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 hier 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 Gaussfiltern mit ansteigendem Sigma-Wert 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. Aufgrund der Verwendung von Integralbildern kann eine Berechnung in konstanter Zeit geschehen, was SURF eine höhere Effizienz gegenüber SIFT bringt (Juan und Gwun 2009).

Jannik Hoffmann 20.12.14 12:12

Kommentar [14]: Evtl. bessere Quelle?

Bei SURF handelt es sich um einen Keypointerkennungs- und Beschreibungsalgorithmus, welcher in der Lage ist, trotz Skalierung und Drehung Bilder und Objekte wiederzuerkennen. Der Algorithmus wurde von Bay et al. (2008, S. 1) vorgestellt und gilt seitdem als fehlerunempfindlicher 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) modernere Algorithmen,

welche bei einer ihnen angepassten Anwendung deutlich effizienter sind (Schaeffer 2013, S. 5), doch bei einigen eigenen Testläufen stellte sich die Kombination aus SURF als Keypointerkennner und –beschreiber als am zuverlässigsten heraus.]



Jannik Hoffmann 20.12.14 12:12

Kommentar [15]: Unterschied zu anderen Algorithmen, Feature Detection, Binary Features, Corner Detection etc. aufzeigen

Abb. 4.3 SURF-Keypointerkennung auf einem Foto

Nach Erkennung und Berechnung der einzelnen Bildpunkte wird anhand dieser von SURF das Bild in Quadrate aufgeteilt und für diese 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.

Dieser Vorgang geschieht bei einem Fast Approximate Nearest Neighbor Matching für beide Bilder. 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. 4.4 Fast Approximate Neighbor Matching der beiden Bilder

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

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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [16]: Evtl. genauere Erklärung

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 im Anschluss auszuwerten.

5.1 Vorstellung der Implementation und ihrer Komponenten

Wie bereits erwähnt besteht die Implementierung der hier vorgestellten Applikation aus zwei wesentlichen Teilen: Dem Client auf der Google Glass, welcher die Darstellung des User Interfaces, das Fotografieren des Objektes und den Upload zum Server übernimmt und dem Server, welcher jegliche Aufgaben im Bereich des Matchings, der Verwaltung und der Informationsabfrage übernimmt. 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	https://developers.google.com/glass/develop/gdk/index
Gradle	Glass Client	2.1	http://gradle.org
Apache HTTP Components	Glass Client & OCV Server	4.3.5	https://hc.apache.org
Jetty	OCV Server	6.1.10	http://www.eclipse.org/jetty/
Spring Framework	OCV Server	4.1.3	http://spring.io
Maven	OCV Server	3.2.3	https://maven.apache.org
Log4j	OCV Server	2.1	http://logging.apache.org/log4j/2.x/
GSON	OCV Server	2.2.4	https://code.google.com/p/google-gson/
JavaCPP	OCV Server	0.9	https://github.com/bytedeco/javacpp
OpenCV	OCV Server	2.4.9	http://opencv.org

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. 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 Kommunikationen 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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [17]: Checken

Jannik Hoffmann 20.12.14 12:12

Kommentar [18]: Quelle

Jannik Hoffmann 20.12.14 12:12

Kommentar [19]: Quelle

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. Zur Bibliotheksverwaltung und Kompilierautomatisierung und wegen der guten Integration mit Jetty kommt Maven zum Einsatz. Das Logging der Serveraktivitäten und -fehler geschieht über die Log4j-Bibliothek von Apache und jegliche Nutzung von JSON-Dateien wird mit der GSON-Bibliothek von Google, welche eine Erstellung von JSON aus Java Objekten und vice-versa realisiert (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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [20]: Quelle

5.2 Glass Client

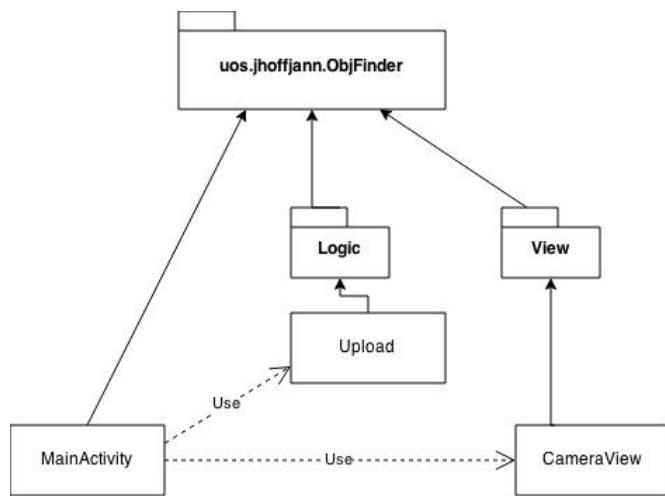


Abb. 5.1 UML-Darstellung des Glass Clients

Der auf der Google Glass ausgeführte Client 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 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 Fokus, durch einfaches Tippen auf das Touchbedienfeld der Glass wird das Foto aufgenommen. Nach erfolgreichem Fotografieren und Abspeichern des Bildes findet dann 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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [21]: Erklärung?

```

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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [22]: Quelle

```

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 terrible 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 ist der Vorgang durch einfaches Tippen auf das Touchfeld neustartbar.

5.3 OCV Server

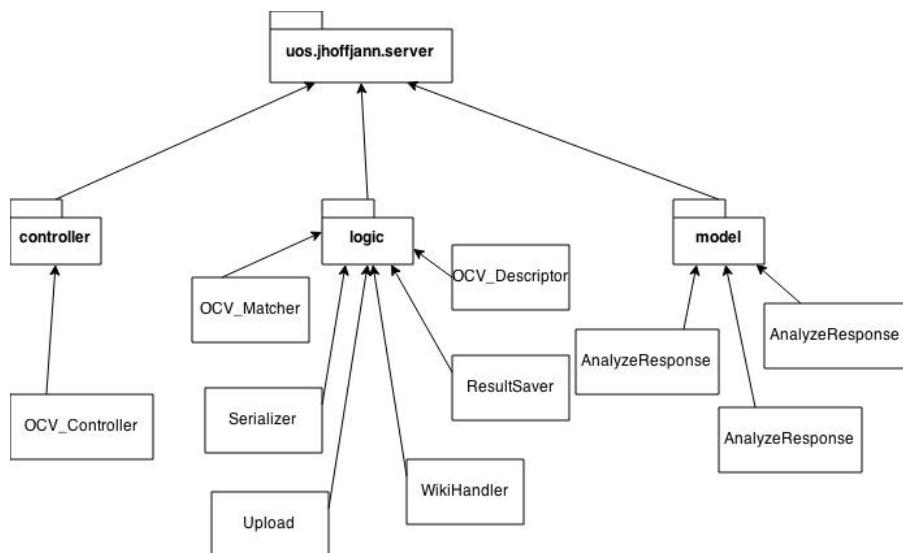


Abb. 5.2 UML-Darstellung des OCV Servers

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 Grundfunktion 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

Ein neues 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, der eine Suche in Suche in der freien Internetencyklopädie Wikipedia nach dem mitgelieferten Namen des Objektes durchführt und im Anschluss den ersten Absatz des Ergebnisses extrahiert.

```

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 und 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 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 an einer 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 [4.2.2](#) beschrieben, durchgeführt. Die Auswahl des besten Ergebnisses und somit einem wiedererkannten Objekt erfolgt wie im gleichen Kapitel beschrieben lediglich über die Auszählung der besten Treffer.

Nach Abschluss aller Einzelthreads wird überprüft ob ein Objekt gefunden wurde und bei mehr als einem Ergebnis, 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 hier 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 konnte insgesamt gute Ergebnisse bringen. Bekannte Objekte konnten in den meisten Fällen wiedererkannt werden und 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 sollten um dieses Ergebnis zu erreichen. So kam es bei einigen Fällen immer wieder dazu das 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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [23]: Schwäche: Mehrere Objekte auf einem Foto

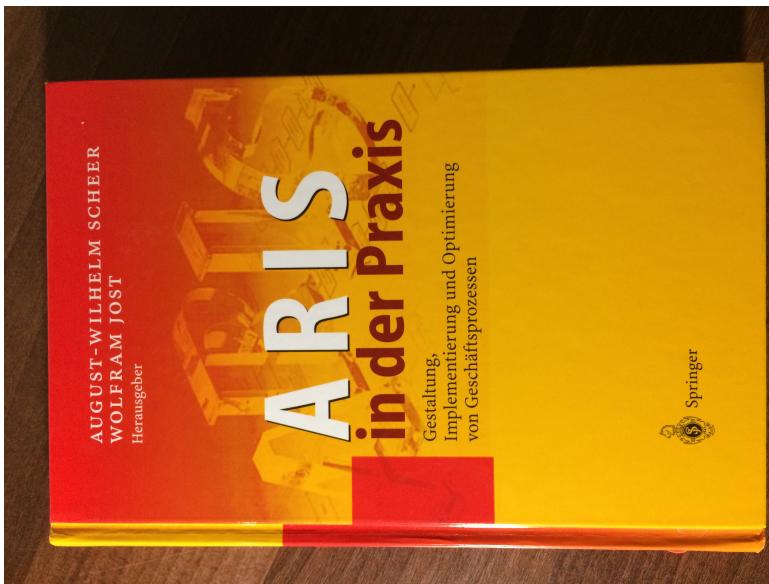


Abb. 5.3 Ein Bild mit hoher Fehleranfälligkeit

Der Grund für diese Fehleranfälligkeit liegt in der Beschaffenheit des SURF-Algorithmus und diesem Bild. Beim Matching mit danach hochgeladenen Fotos wurde nämlich 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 um wirklich nur das wiederzuerkennende Objekt zu enthalten.

Eine weitere Schwäche der Applikation liegt in der Wahl des SURF-Deskriptors, welcher auf die reine Erkennung von in Graustufen hinterlegten Bildern beschränkt ist. Die lässt viel Potenzial bei farbenfrohen Objekten fallen und senkt den Wiedererkennungswert.

Jannik Hoffmann 20.12.14 12:12

Kommentar [24]: Bag of Words Ansatz

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 Information in die Wahrnehmung des Trägers integriert und diese so 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 wurde beim Hochladen benannt und liegen so zur Wiedererkennung zur Verfügung. Im Idealfall sollen die Objekte beim erneuten Fotografieren mit der Google Glass durch einen

Anwender wiedererkannt werden und die verknüpfte kontextsensitive Information soll 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.

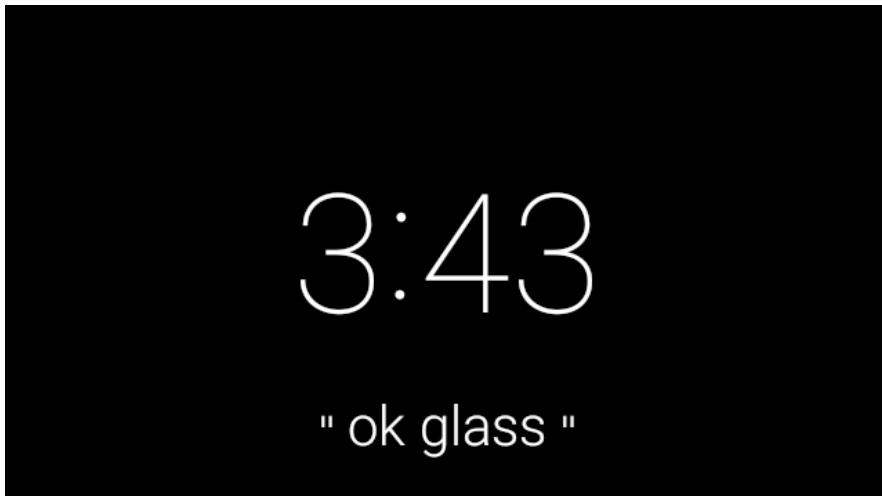


Abb. 6.2 Der Startbildschirm der Google Glass

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. Die andere Möglichkeit in die Applikation zu gelangen ist die Spracherkennung der Google Glass zu nutzen. Dafür taucht nach Ansage des Startbefehls „ok glass“ ein Menü mit verschiedenen Auswahlmöglichkeiten für Anschlussbefehle auf. Auch in diesem Menü ist die Applikation Object Finder zu finden, die die hier implementierte Anwendung wiederspiegelt. Beide Möglichkeiten sind in der **ABBILDUNG 6.3** zu sehen.

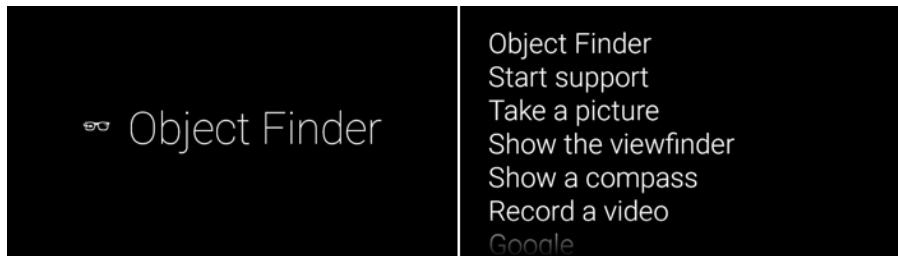


Abb. 6.3 Die beiden Startmöglichkeiten der Applikation



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**) und nach dem 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.

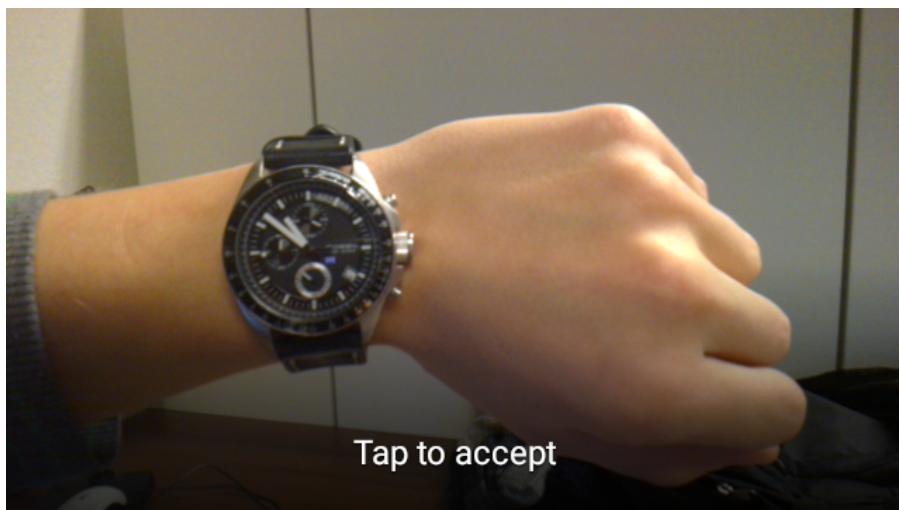


Abb. 6.6 Vorschaubild mit Akzeptanzanfrage

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**).

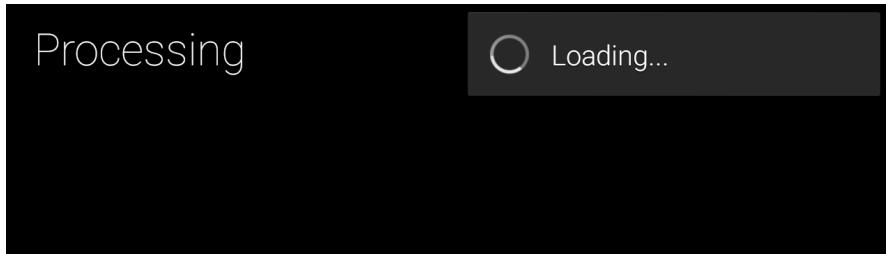


Abb. 6.7 Die beiden Ladebildschirme der Applikation

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

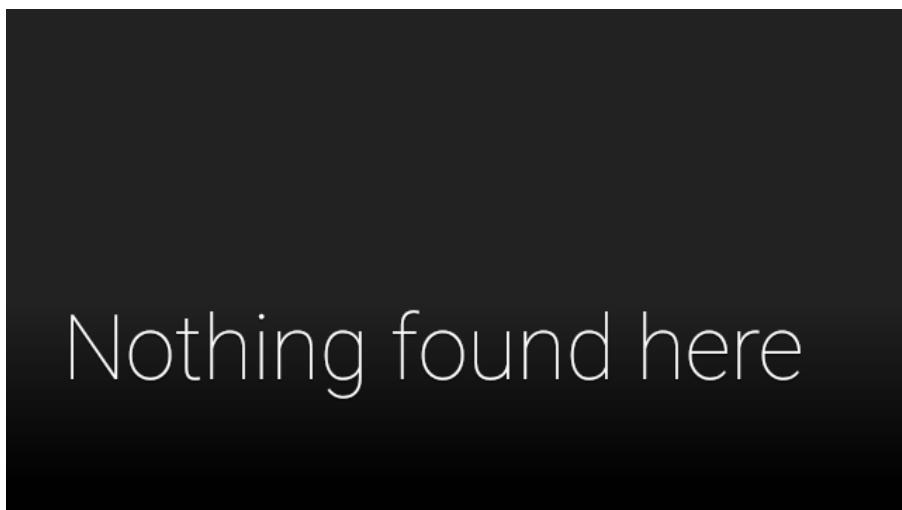


Abb. 6.9 Das negative Ergebnis einer anderen Anfrage

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 Beispielnutzung der Applikation wurde zum Schluss der Arbeit in einer kurzen Fallstudie dargelegt um eine solche Nutzung zu verdeutlichen.

Abschließend lässt sich sagen, dass mit dieser Arbeit gezeigt wurde, dass ein Potenzial in der Nutzung von kontextsensitiven Informationen in der Industrie vorhanden ist. Es ist möglich durch die Nutzung von darauf spezialisierten Algorithmen, Objekte in Fotos und somit der, den Nutzer umgebenden Realität, wiederzuerkennen und diese mit Information zu verknüpfen um auf Basis dieser weitergehende Information zu hinterlegen oder beispielsweise anschließende Arbeitsschritte einzuleiten.

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.

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 vermutliche modernere Ansätze des Matchings genutzt werden und dazu die Forschung im Bereich der Computer Vision beobachtet werden. 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.

Jannik Hoffmann 20.12.14 12:12

Kommentar [25]: 3D Modelle zur Berechnung

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, A.; Ortiz, R.; Vandergheynst, P. (2012) FREAK: Fast Retina Keypoint. 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 510–517.
- 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.
- Bradski, Gary; Kaehler, Adrian (2008) Learning OpenCV Computer Vision with the OpenCV Library. 1. Auflage, Sebastopol, CA, O'Reilly, Inc.
- 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, HL (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.
- Dey, A.K.; Salber, D.; Abowd, G.D.; Futakawa, M. (1999) The Conference Assistant: combining context-awareness with wearable computing. Digest of Papers. Third International Symposium on Wearable Computers. IEEE Comput. Soc, 21–28.
- Dey, Anind K. (2001) Understanding and Using Context. Personal and Ubiquitous Computing, 5 (1):4–7.
- 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):3069–79.
- 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) Tech specs. <https://support.google.com/glass/answer/3064128?hl=de>, abgerufen am 26.11.2014.
- Google (2014b) Glass Platform Release Notes. <https://developers.google.com/glass/release-notes>, abgerufen am 02.12.2014.
- Huang, Zhanpeng; Hui, Pan; Peylo, Christoph; Chatzopoulos, Dimitris (2013) Mobile augmented reality survey: a bottom-up approach. Arxiv.Org. 35.

- Indulska, J; Sutton, P (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 (2005) 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 :167–188.
- Lowe, D.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.
- 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, DG (2009) Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. VISAPP (1), .

- opencv dev team (2014) OpenCV API Reference — OpenCV 2.4.9.0 Documentation. <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, .
- 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.
- Schaeffer, Cameron (2013) A comparison of keypoint descriptors in the context of pedestrian detection: freak vs. surf vs. brisk. Stanford University, 5.
- Schilit, B.N.; Theimer, M.M. (1994) Disseminating active map information to mobile hosts. *IEEE Network*, 8 (5):22–32.
- Schmidt, A; Laerhoven, K. van (2001) How to build smart appliances? *IEEE Personal Communications*, 8 (4):66–71.
- 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.
- QRcode.com | DENSO WAVE. <http://www.qrcode.com/en/index.html>, abgerufen am 16.10.2014.
- Google to Sell Heads-Up Display Glasses by Year's End - NYTimes.com.
<http://bits.blogs.nytimes.com/2012/02/21/google-to-sell-terminator-style-glasses-by-years-end/?ref=technology>, abgerufen am 20.11.2014.

8 Anhang

A Unterkapitel des Anhangs

B Zweites Unterkapitel des Anhangs

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 20. Dezember 2014

Jannik Hoffjann