Bachelorarbeit

Interaktive Lokalisierung durch Objekterkennung

Adrian Batzill

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik
Lehrstuhl für Algorithmen und Datenstrukturen

Bearbeitungszeitraum

08. 07. 2013 – 08. 10. 2013

Gutachterin

Prof. Dr. Hannah Bast

Betreuer

Prof. Dr. Hannah Bast

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Abstract

Objekterkennung ist ein zentrales Thema der Computer Vision. Diese Arbeit beschreibt den Stand der Technik bezüglich Objekterkennung, bei der nur ein einziges zuvor gelerntes Bild als Ausgangspunkt für die Erkennung dient. Dazu werden drei bekannte Algorithmen - SIFT, SURF und ORB - miteinander verglichen und deren Tauglichkeit zur Verwendung in Lokalisierungsanwendungen, wie beispielsweise Augmented Reality, evaluiert. Des Weiteren wird diskutiert, wie sich diese Algorithmen sinnvoll in eine Augmented Reality Anwendung integrieren lassen, um eine exakte Augmentierung zu gewährleisten.

1 Einleitung und Motivation

Viele moderne Anwendungen bieten ihren Nutzern lokalitätsabhängige Dienste an. So wird bei Webanwendungen oftmals der Standort des Benutzers für personalisierte Werbeeinblendungen bezüglich der aktuellen Position verwendet. Insbesondere durch die zunehmende Verbreitung von Smartphones und anderen Mobilsystemen bieten sich lokalitätsabhängige Dienste an: von der App, die Sehenswürdigkeiten in der Umgebung anzeigt, bis zum kompletten Navigationssystem. Für viele Dienste ist eine reine Satellitenlokalisierung durch GPS (Global Positioning System) und ähnliche Anbieter ausreichend. Benötigt eine Anwendung jedoch eine exakte Position sind diese Dienste oft zu ungenau. So sind Ungenauigkeiten von 30 Metern bei GPS in Deutschland keine Seltenheit. Für solche Anwendungen kann sich stattdessen eine Lokalisierung durch Objekterkennung anbieten, sofern die exakten Positionen dieser Objekte bekannt sind. Dabei wird die Kamera des Smartphones dazu verwendet bekannte Objekte automatisch wiederzuerkennen und sich anhand der gespeicherten Position dieser Objekte zu orten. Dafür ist es nötig, dass die verwendeten Objekterkennungsalgorithmen schnell genug sind um eine interaktive Lokalisierung mit geringen Latenzen gewährleisten zu können.

Insbesondere für Augmented Reality Anwendungen eignet sich die Lokalisierung durch Objekterkennung. Bei Augmented Reality wird auf dem Smartphonedisplay das aktuelle Kamerabild angezeigt. Erkannte Objekte in diesem Bild können dann mit zusätzlichen Informationen versehen werden, die zuvor in eine Datenbank eingegeben wurden. Da bei diesen Anwendungen also bereits Objekte mit ihren exakten Positionen gespeichert wurden, können diese Informationen auch dazu verwendet werden die Kameraposition zu bestimmen.

In letzter Zeit bieten neuere Entwicklungen von Mobilsystemen, wie etwa die mit einem Display versehene Brille *Google Glass*, Benutzern neue Möglichkeiten Augmented Reality wahrzunehmen.

1.1 Stand der Technik

Es existieren einige Arbeiten die ähnliche Themen wie das dieser Arbeit behandeln. In [KOTY00] geht es ebenfalls um die Lokalisierung der Kamera durch Objekterkennung. Dort wird dazu jedoch eine Stereokamera, und somit 3D Bilddaten, verwendet um zusätzlich die Distanz und eine exakte Ausrichtung zum Objekt zu erhalten. So ist eine exaktere Lokalisierung möglich.

In [RSF+06] wird untersucht, inwiefern sich die Kameralokalisierung auf Basis von Bilddaten für die Naviagation in Gebäuden eignet. Dazu wird eine Datenbank mit vielen Bildern des Gebäudes und den Positionen der Kamera beim Aufnehmen dieser Bilder erstellt. Um die Kameraposition später zu bestimmen wird dann überprüft, welches Bild in der Datenbank dem Kamerabild am ähnlichsten ist. Zudem wird die Position der Kamera über die Zeit verfolgt. Da die Fortbewegung in Gebäuden meist langsam ist, können zur Performanceverbesserung so frühzeitig weit entferte Bilder ausgeschlossen werden.

Ähnlich wie in dieser Arbeit wird in [HS97] ebenfalls ein Triangulierungsverfahren verwendet, um die Position einer Kamera anhand der Positionen von Bilddaten zu ermitteln. Dabei wird jedoch nur der Fall untersucht, in dem sich exakt zwei zuvor aufgenommene Objekte im Sucher befinden. Auch wird dort nicht darauf eingegangen, wie die Bildpositionen im Sucher gefunden werden können, sondern es wird davon ausgegangen, dass diese bekannt sind.

In [Hof12] wird ebenfalls untersucht, wie eine exakte Lokalisierung, insbesondere für Augmented Reality Anwendungen, erreicht werden kann. Dabei werden hier GPS-Daten zusammen mit einem Pedometer verarbeitet, um die Positionsgenauigkeit zu verbessern.

2 Bild- und Objekterkennung

Bild- und Objekterkennung sind ein zentrales Thema im Bereich der Computer Vision. So haben sich im Laufe der Zeit verschiedene Verfahren entwickelt, dem Computer das Interpretieren von Bilddaten beizubringen. Hierbei hat sich gezeigt, dass es schwierig scheint ein einziges System zu konstruieren, welches in der Lage ist ein menschenähnliches Interpretationsvermögen zu zeigen. Somit beinhalten moderne Computer Vision Bibliotheken eine Vielzahl an Algorithmen für unterschiedliche Einsatzzwecke. Bei der für diese Arbeit verwendeten Bibliothek OpenCV[1] sind dies derzeit über 2500 Algorithmen für Gesichtserkennung, Kreiserkennung, Machinelearning und viele weitere Einsatzgebiete, sowie Vorverarbeitungsalgorithmen und Filter um andere Algorithmen effizienter zu machen. Für die generische, exakte Objekterkennung, bei der nur ein einzelnes zuvor gelerntes Bild in einem neuen Bild gefunden werden soll, wurden diverse Verfahren entwickelt, die zunächst in beiden Bildern wichtige Features - in der Regel Ecken von Objekten, aber auch Kanten und Blobs - finden (Feature Detector), Informationen über diese Punkte extrahieren (Descriptor Extractor) und diese anschließend miteinander vergleichen (Feature Matcher). Die meisten Algorithmen, die nach diesem Verfahren arbeiten, bieten Vorteile wie:

- Skalierungsinvarianz - die selben Punkte werden auch in einem skalierten Bild wieder gefunden.
- Rotationsinvarianz - die selben Punkte werden auch wieder gefunden wenn das Bild gedreht wurde.
- Robustheit gegenüber geometrischen Transformationen - Das Bild wird auch wieder erkannt, wenn sich der Blickwinkel, aus dem das Foto aufgenommen wurde, verändert.
- Robustheit gegenüber Beleuchtung - das Bild wird auch erkannt, wenn es dunkler oder heller ist, oder wenn sich der Lichteinfall verändert.

[1] Siehe `http://opencv.org/`.

- Robustheit gegenüber Noise - Bildfehler (Noise) sollten nicht als Feature eines Bildes erkannt werden.

Insbesondere sei bei diesen Verfahren anzumerken, dass nicht alle Algorithmen jeweils einen Detector, einen Extractor und einen Matcher mitbringen. Die verschiedenen Elemente der Algorithmen können hingegen - bis zu einem gewissen Grad - frei kombiniert werden[2], sodass beispielsweise die von einem ORB Detector erkannten Keypoints von einem SURF Extractor extrahiert werden[3]. Dies ermöglicht eine große Anzahl an Kombinationsmöglichkeiten um ein für den gegebenen Anwendungsfall optimales Verfahren zu entwickeln.

2.1 Feature Detection

Die Feature Detection ist der Teil des Verfahrens in dem sich die unterschiedlichen Algorithmen hauptsächlich unterschieden. Ein Feature Detector muss in der Lage sein, Punkte in einem Bild zu finden die das Bild besonders gut charakterisieren. Im Optimalfall ist der Detector in der Lage, auch bei einer stark veränderten Version des selben Bildes immer noch die selben Keypoints zu finden. Insbesondere unter den Aspekten der Skalierung, Rotation, geometrischer Transformation und Beleuchtungsveränderungen sollte ein Feature Detector möglichst robust sein. Er ist somit der zentrale Bestandteil des Verfahrens.

2.2 SIFT Feature Detector

SIFT steht für **s**cale-**i**nvariant **f**eature **t**ransform und wurde 1999 von David Lowe in [Low99] vorgestellt. Er ist bis heute einer der am weitesten verbreiteten Feature Detektoren. Um Features zu finden verwendet SIFT eine Reihe von Verfahren.
Zunächst wird das Bild auf verschiedene Größen skaliert (Scale Space). Anschließend wird auf jedes dieser Bilder ein Gauss-Filter angewandt, der unterschiedlich stark verschwommene Kopien des Bildes erstellt (Octaves), wie in Abb. 2.1 zu sehen ist.

[2] Es ist natürlich nur sinnvoll, gleichartige Algorithmen zu kombinieren, also beispielsweise Eckendetektoren mit Eckenextraktoren, nicht jedoch mit Blockextraktoren.
[3] ORB und SURF werden in den folgenden Kapiteln genauer erklärt.

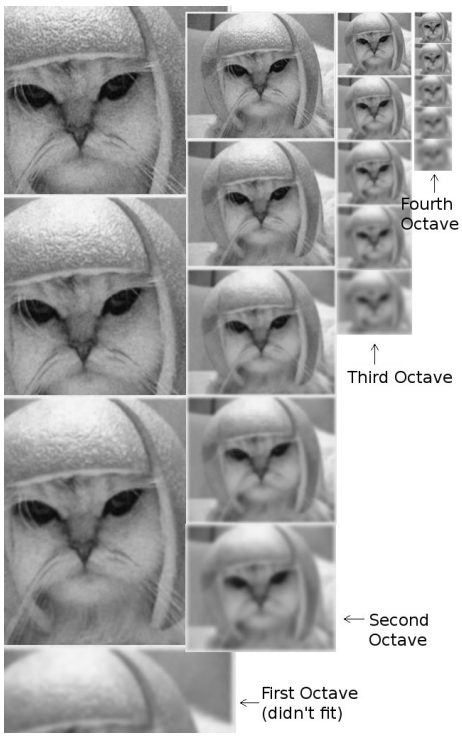

Abbildung 2.1: Der SIFT Scale Space und Octaves nach [Sin10]

Der Gauss-Filter ist definiert durch $L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y)$.

- L ist das verschwommene Ausgabebild.
- x und y sind die Bildkoordinaten.
- k ist die Octave, also der aktuelle Skalierungsfaktor, sodass der Gauss-Filter immer den selben Bildinhalt unabhängig von der Skalierung gleich behandelt.
- σ bestimmt wie stark das Bild verschwommen werden soll (Breite der Gauss-Funktion).
- G ist der Gauss-Filter Operator, welcher die Gauss-Funktion mit Breite σ an der Stelle (x, y) definiert.
- I ist das Originalbild.
- $*$ ist der Faltungsoperator der G auf jeden Pixel aus I anwendet.

Der Gauss-Filter dient dazu, unwichtige Details und Noise zu entfernen, während große, globale Features erhalten bleiben. Dabei werden insbesondere sehr kontrastreiche Regionen im Bild stark verändert, während sich kontrastarme Regionen und

Flächen kaum verändern.

Nun erfolgt die Kantenerkennung im Bild. Dies kann über einen Laplacian of Gaussian (LoG) Filter erreicht werden. Dabei wird zunächst ein Gauss-Filter über das Bild gelegt um Noise zu verwischen. Anschließend werden die zweiten partiellen Ableitungen des Bildes, also die Hesse Matrix H für jeden Pixel, errechnet.

$$H(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix},$$

wobei $L_{xx}(x, y, \sigma)$ (und analog L_{xy} und L_{yy}) die Faltung der zweiten partiellen Ableitung $\frac{\partial^2 g(\sigma)}{\partial x^2}$ des Gauss-Bildes entspricht. Nun kann für jeden Pixel mit Hilfe der Determinanten der Hesse Matrix errechnet werden, ob es sich beim untersuchten Pixel um ein lokales Maximum oder Minimum handelt.

In der Praxis wird jedoch oft auf ein schnelleres Verfahren zurück gegriffen, in dem verschieden stark verschwommene Gauss-Bilder nur voneinander subtrahiert werden (Dies wird auch Difference of Gaussian, oder DoG, genannt) [Low04].

Somit werden nun innerhalb jeder Octave die unterschiedlich verschwommenen Bilder voneinander subtrahiert: $D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_{i-1}\sigma)$. Dadurch werden alle Stellen, an denen sich die Bilder stark verändert haben, und somit kontrastreiche Regionen, wie Kanten und Ecken, im Bild gefunden.

Anschließend werden die Scale Space Extremas ermittelt. Hierzu wird jeder Pixel mit seinen acht Nachbarpixeln in der selben Octave, sowie mit den jeweils neun Nachbarpixeln in den Octaves darüber und darunter verglichen, wie in Abb. 2.2 dargestellt. Bildet der untersuchte Pixel ein Minimum oder ein Maximum unter seinen Nachbarn, wird er als möglicher Keypoint vorgemerkt. Als Verbesserung zur Stabilität der Keypoints schlägt Lowe in [Low04] vor, die Keypoints mit Hilfe einer quadratischen Taylor Expansion auf Subpixel Genauigkeit anzunähern[4]. Dies erleichtert auch das entfernen von Keypoints mit einem geringen Kontrast.

Anschließend werden diese Keypoints wieder gefiltert, um nur möglichst aussagekräftige Features im Ergebnis zu haben. Die zuvor beschriebene Gauss-Filter Methode ergibt insbesondere an Kanten viele Keypoint Kandidaten, welche eliminiert werden

[4]Es wird also eine Funktion gefunden, die den lokalen Bildausschnitt approximiert um von dieser dann das Maximum zu ermitteln.

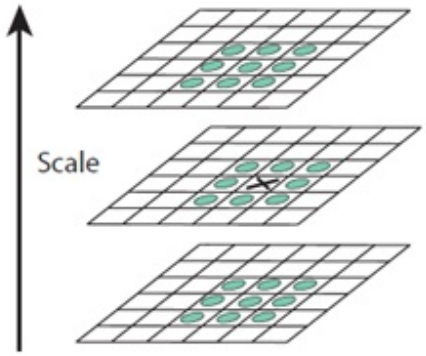

Abbildung 2.2: SIFT Vergleich, um zu prüfen, ob es sich bei einem Pixel um ein Maximum handelt, siehe [Sin10].

sollen. Dafür werden für jeden Keypoint zwei orthogonale Gradienten auf Basis der umgebenden Pixel des Keypoints berechnet [Sin10].

- Handelt es sich beim Keypoint um eine Fläche, sind die Gradienten klein und der Keypoint sollte entfernt werden.
- Handelt es sich um eine Kante, ist ein Gradient groß (Orthogonal zur Kante) und der andere klein (auf der Kante). Auch diese Keypoints sollten entfernt werden.
- Handelt es sich um eine Ecke, sind beide Gradienten groß. Ecken ergeben die besten Features und sollten daher beibehalten werden.

Um Rotationsinvarianz in SIFT zu erreichen, wird für jeden Keypoint eine Richtung in Form eines Winkels berechnet. Diese Richtung wird aus lokalen Orientierungshistogrammen aus der Nachbarschaft des Keypoints ermittelt. Zunächst wird für jeden Pixel in der Nachbarschaft des Keypoints eine Gradientstärke

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2},$$

sowie eine Gradientrichtung

$$\theta(x,y) = tan^{-1}(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)})$$

ermittelt. Nun wird ein Histogramm über die verschiedenen Gradientrichtungen, gewichtet nach ihrer Stärke, erstellt, sodass ermittelt werden kann in welche Richtung die lokale Nachbarschaft allgemein den stärksten Gradient aufweist. Diese Rich-

tung wird nun dem Keypoint zugewiesen. Sollten mehrere Richtungen ähnlich stark vertreten sein, können hier auch mehrere Keypoints ausgegeben werden um Nichterkennung bei geringen Abweichungen vorzubeugen [Low99][Low04].

2.3 SIFT Descriptor Extractor

SIFT bringt auch einen eigenen Descriptor Extractor mit um die gefundenen Keypoints möglichst aussagekräftig zu speichern. Bei der Extraktion der Features aus den Keypoints wird ein 16 mal 16 Pixel großes Feld um den Keypoint ausgeschnitten. Dieses wird wiederum in 4 mal 4 Felder mit jeweils 16 Pixeln aufgeteilt. Für jedes dieser 4 mal 4 Felder werden nun wie zuvor die Gradientrichtung und -Stärke ausgerechnet, wie in Abb. 2.3 deutlich wird.

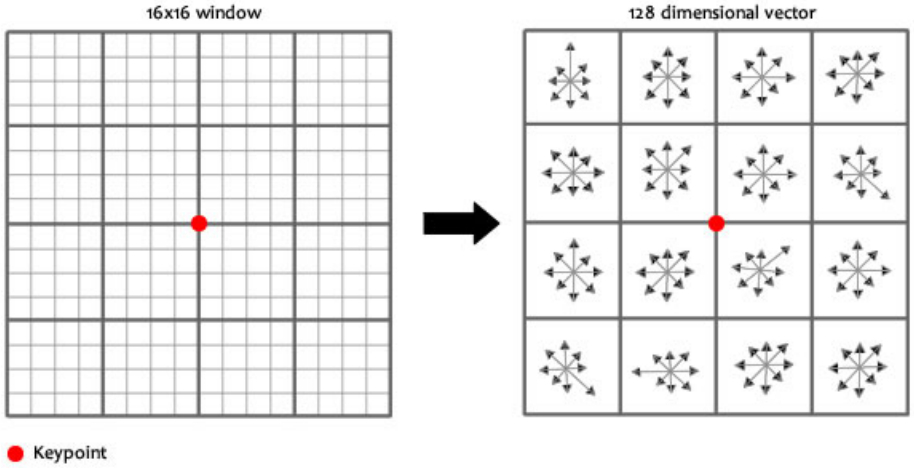

Abbildung 2.3: SIFT Vergleich, um zu prüfen, ob es sich bei einem Pixel um ein Maximum handelt, siehe [Sin10].

Zusätzlich wird die Gradientstärke in Abhängigkeit zur Distanz zum Keypoint mit einer Gauss-Funktion multipliziert um entfernten Pixeln weniger Gewicht zu geben. Schließlich werden die Gradientstärken noch normalisiert um bessere Beleuchtungsinvarianz zu erreichen.

Um trotz der Gradientrichtungen Rotationsinvarianz zu gewährleisten wird von den Gradientrichtungen schließlich noch die Richtung des Keypoints selbst subtrahiert, sodass sich alle Richtungen im Deskriptor relativ zu der des Keypoints verhalten.

Die Histogramme der einzelnen Regionen ergeben schließlich den Deskriptor (16 Regionen mit jeweils acht Werten ergeben einen Deskriptor mit 128 Werten).

2.4 SURF Feature Detector

SURF steht für **s**peeded **u**p **r**obust **f**eatures und wurde im Jahr 2006 in [BTVG06] vorgestellt. Im Gegensatz zu SIFT wird SURF als schneller beschrieben, ohne dabei negative Auswirkungen auf die Qualität zu haben. Zusätzlich wird im Verleich zu SIFT, der einen 128-dimensionalen Deskriptorvektor erzeugt, nur ein 64-dimensionaler Vektor zur Beschreibung von Keypoints verwendet, um das spätere Matching effizienter zu machen [BTVG06]. SURF arbeitet im Vergleich zu SIFT nicht auf Scale Spaces, sondern auf dem Integralbild

$$I_\Sigma(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x,y).$$

Für die Erkennung von Keypoints wird ein *Fast-Hessian* Verfahren verwendet [Eva09].

Wie bei der SIFT Optimierung wird bei Fast-Hessian ebenfalls die Differenz von Gauss-Bildern verwendet um Kanten zu erkennen, anstelle der Kantenerkennung durch den Laplacian of Gaussian. Hier wird jedoch zusätzlich eine Optimierung vorgenommen, indem nicht der tatsächliche Gauss-Filter verwendet wird um das Bild zu glätten, sondern lediglich ein einfacher Box-Filter, der sich insbesondere für große Filtermasken deutlich effizienter verhält. Auf dem zuvor berechneten Integralbild ist die Auswertung einer Box-Filter Maske sogar in konstanter Zeit möglich [Eva09].

Um Skalierungsinvarianz zu erreichen wird statt eines echten Scale-Spaces wie bei SIFT dieser nur implizit erstellt. Anstelle von Bildern unterschiedlicher Größe wird lediglich die Größe des Box-Filters bei Bedarf angepasst um unterschiedlich große Teile des Bildes abzudecken und so unterschiedliche Skalierungen des Bildes zu simulieren. Die Determinante der Hesse-Matrix jedes Pixels kann dadurch über

$$det(H_{approx}) = D_{xx} \cdot D_{yy} - (0.9 \cdot D_{xy})^2$$

approximiert werden, wobei D_{xx} die durch Box-Filter approximierte zweite partielle Ableitung in x-Richtung darstellt und analog D_{yy} und D_{xy} die anderen zweiten partiellen Ableitungen [BTVG06]. Da der Box-Filter auf dem Integralbild in konstanter

Zeit ausgewertet wird, ergibt sich somit kein nennenswerter Einfluss auf die Rechenzeit. Außerdem erlaubt dieses Vorgehen mehrere Ebenen des Scale-Spaces parallel abzuarbeiten.

Nachdem die Kanten erkannt wurden wird nun, analog zu SIFT, für jeden Kantenpunkt ermittelt ob es sich bei ihm um ein lokales Maximum unter seinen 8 Nachbarn in der selben Octave und seinen jeweils 9 Nachbarn in den Octaves darüber und darunter handelt.

Um die exakte Subpixelposition des Keypointkandidaten zu ermittelt wird eine quadratische 3D-Funktion gesucht, die den Bildausschnitt approximiert. Dies kann mit Hilfe einer Taylor Expansion mit dem ermittelten Maximum als Ursprung erreicht werden. Durch Gleichsetzen dieser Funktion mit Null kann so das exakte Maximum ermittelt werden [Eva09].

2.5 SURF Descriptor Extractor

Ähnlich dem SIFT Deskriptor beschreibt auch der SURF Deskriptor wie die Pixelintensitäten in der Nachbarschaft jedes Keypoints verteilt sind. Dabei werden jedoch Integralbilder in Kombination mit einem Haar Wavelet Filter verwendet um die Berechnungszeit zu verringern. Haar Wavelets sind einfache Filter um Gradienten in x und y Richtung zu finden. Für SURF werden die beiden in Abb. 2.4 gezeigten Wave-

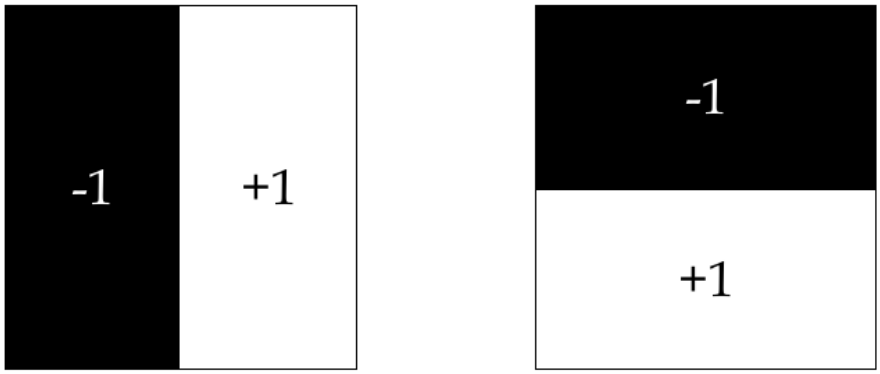

Abbildung 2.4: SURF Wavelets zur Richtungsbestimmung. Das linke Wavelet dient zur Bestimmung von Gradienten in x-Richtung, das rechte zur Bestimmung von Gradienten in y-Richtung. Die dunklen Stellen werden mit -1, die hellen mit +1 gewichtet, siehe [BTVG06].

lets verwendet und mit dem Bildinhalt an der Stelle des Keypoints verglichen, was mit Hilfe von Integralbildern nur sechs Operationen für jedes Wavelet beansprucht [Eva09].

Zunächst werden also die Haar Wavelet Antworten in x und y Richtung innerhalb eines Radius von $6k$ errechnet, wobei k die Bildskalierung darstellt in welcher der Keypoint gefunden wurde. Die Haar Wavelets selbst besitzen dabei eine Größe von $4k$ [BTVG06]. Anschließend werden die Wavelet Antworten mit einer Gauss-Funktion mit Zentrum beim Keypoint und Parameter $\sigma = 2k$ gewichtet. Mithilfe eines Sliding Windows der Größe $\frac{\pi}{3}$ wird nun über die gefundenen Wavelet Antworten iteriert und die gewichteten Intensitäten aufsummiert. Dies wird in Abb. 2.5 gezeigt.

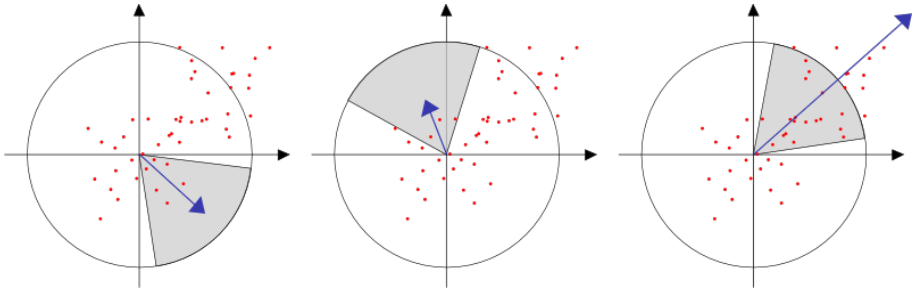

Abbildung 2.5: Sliding Window zur Berechnung der Keypointrichtung bei SURF. In der Mitte befindet sich der Keypoint. Die graue Region ist das sliding Window. Die einzelnen Punkte sind die Waveletantworten, siehe [Eva09].

Die Richtung für die der Wert des Sliding Windows am größten ist bestimmt die Richtung des Keypoints. Nun wird ein rechteckiges Fenster um den Keypoint mit der Größe $20k$ erstellt, welches parallel zur zuvor berechneten Richtung des Keypoints ausgerichtet ist, sodass alle nachfolgenden Operationen relativ zu dieser Richtung gemacht werden. Das Fenster wird nun in 16 gleich große, rechteckige Subregionen aufgeteilt. In jeder Subregion werden dann wieder Haar Wavelets der Größe $2k$ für 25 gleichverteilte Samplepoints berechnet und erneut mit einer Gaussverteilung mit $\sigma = 3.3k$ und Zentrum am Keypoint gewichtet. Für jede Subregion wird dann ein Featurevektor der Form

$$v_{subregion} = (\Sigma dx, \Sigma dy, \Sigma|dx|, \Sigma|dy|)$$

extrahiert, wobei dx und dy die Waveletantworten in x und y Richtung beschreiben [Eva09]. Durch die 16 Subregionen ergibt sich somit ein Desktriptorvektor der Länge $4 \cdot 16 = 64$.

2.6 ORB Feature Detector

Der neuste Algorithmus zur Feature Detektion der breite öffentliche Anerkennung erhalten hat wurde 2011 von Ethan Rublee et al. in [RRKB11] vorgestellt. Der Fokus bei der Entwicklung lag dabei vor Allem darauf, den Algorithmus möglichst effizient zu gestalten um interaktive Objekterkennung selbst auf mobilen Endgeräten zu ermöglichen. Die Abkürzung ORB steht dabei für *Oriented FAST and Rotated BRIEF*. Es handelt sich bei ORB nicht um eine eigenständige Implementierung von Grund auf, sondern um eine Erweiterung eines bestehenden Feature Detector namens FAST, sowie eine Erweiterung eines bestehenden Deskriptors namens BRIEF.

Um Keypoints in einem gegebenen Bild zu finden geht FAST wie folgt vor: Für jeden Pixel p mit der Intensität I_p des Bildes wird überprüft, ob dieser einen guten Keypoint darstellt. Dazu wird, wie in Abb. 2.6 gezeigt, ein Kreis um p gebildet, der exakt 16 Pixel enthält[5].

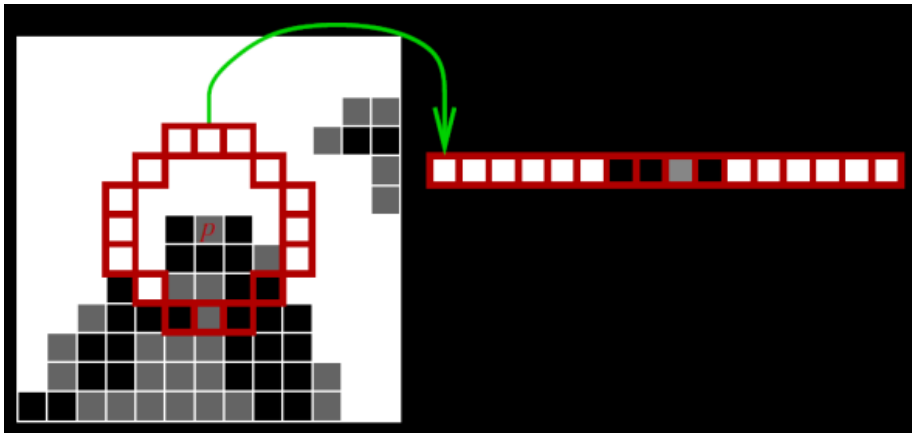

Abbildung 2.6: Der FAST Kreis zum Erkennen von Ecken. Sind viele aufeinander folgende Pixel im resultierenden Vektor sehr hell oder sehr dunkel, handelt es sich um eine Ecke, Siehe [RPD10].

Nun wird geprüft, ob insgesamt mindestens N aufeinander folgende Pixel des Kreises entweder über, oder unter $I_p \pm T$ sind, wobei T ein Threshold Parameter (für gewöhnlich 20%) ist. Für N wird in [Vis] ein Wert von 12 empfohlen, für ORB wird jedoch $N = 9$ verwendet [RRKB11].

Zur Beschleunigung des Algorithmus wird zunächst die Intensität der Pixel 1, 5, 9 und 13 mit I_p verglichen. Wie in Abb. 2.6 deutlich wird müssen mindestens drei

[5]Dies entspricht einem Bresenham Kreis vom Radius 3.

dieser vier Pixel das Thresholdkriterium erfüllen. Sollte dies gegeben sein, müssen nun alle 16 Pixel darauf geprüft werden, ob mindestens N konsekutive Pixel das Kriterium erfüllen.

Zusätzlich wird in [Vis] ein Machinelearningverfahren vorgestellt um die 16 Kreispixel als Ecken zu klassifizieren. Dies wird jedoch bei ORB nicht verwendet.

ORB erweitert FAST nun, indem zunächst eine Scalepyramide erstellt wird und auf jedes Bild der Pyramide FAST mit $N = 9$ aufgerufen wird. Der Schwellwert T wird dabei zunächst gering gewählt um möglichst viele Keypoint Kandidaten zu erhalten.

Da laut [RRKB11] FAST jedoch häufig auf Kanten und nicht nur auf Ecken anspricht, werden diese Keypoints nun nach ihrem *Harris Corner Maß* sortiert [6]. Anschließend werden nur die besten K Keypoints verwendet[7].

Des Weiteren besitzt FAST keine Rotationsinvarianz. Um diese in ORB zu gewährleisten wird ein *Intensity Centroid* genanntes Verfahren verwendet. Die Idee basiert darauf, dass das Intensitätszentrum der näheren Umgebung eines Keypoints nicht die Ecke selbst ist, also nicht direkt auf dem Keypoint liegt. Es wird also das Intensitätszentrum in der näheren Umgebung ermittelt. Ein Vektor vom Keypoint zum Intensitätszentrum bestimmt dann die Orientierung des Keypoints [RRKB11].

2.7 ORB Descriptor Extractor

Wie bereits erwähnt ist der ORB Descriptor Extractor eine Erweiterung des BRIEF Extractors. Bei BRIEF besteht der Deskriptor aus einem Bitstring, welcher die Intensitätsverteilung eines Bildausschnitts mit Hilfe von binären Intensitätschecks beschreiben soll. Für einen weichgezeichneten Bildausschnitt p ist dieser Test τ definiert durch

$$\tau(p, x_1, y_1, x_2, y_2) := \begin{cases} 1, & \text{falls } p(x_1, y_1) < p(x_2, y_2) \\ 0, & \text{falls } p(x_1, y_1) \geq p(x_2, y_2) \end{cases},$$

[6] Dies funktioniert ähnlich dem in SIFT verwendeten Verfahren über die Determinante der Hesse Matrix. Siehe [HS88] für weitere Details.

[7] Frei wählbarer Parameter, bestimmt die maximale Anzahl an Keypoints die generiert werden sollen.

wobei $p(x, y)$ die Intensität des Pixels (x, y) definiert. Ein Feature Vektor für das Feature an der Stelle (x, y) wird dann durch n dieser Tests definiert als:

$$f_n(p) := \sum_{1 \leq i \leq n} 2^{i-1} \cdot \tau(p, x, y, x_i, y_i).$$

Für ORB wird ein Deskriptor der Länge 256 verwendet. Der Bildausschnitt besitzt also eine Größe von $16 \cdot 16$ Pixeln.

ORB erweitert diesen Deskriptor zu *Steered BRIEF*, welcher die zuvor ermittelte Orientierung des Features beinhaltet um Rotationsinvarianz zu gewährleisten. Dazu wird der Bildausschnitt lediglich um den zuvor berechneten Winkel rotiert und erst anschließend der BRIEF Deskriptor berechnet.

2.8 Feature Matching

Sind die Features zweier Bilder extrahiert, werden im nächsten Schritt die gefundenen Features zu vergleichen. Ziel ist dabei, zu jedem Feature in Bild I_1 den besten Match in Bild I_2 zu finden. Dazu wird zunächst eine Distanzfunktion eingeführt, welche zwei Deskriptoren vergleicht. Für SIFT und SURF kann dafür einfach die Vektordistanz

$$d(x, y) = \sqrt{\sum i = in(x_i - y_i)^2}$$

verwendet werden[8]. Diese allein kann jedoch bei Bildern in denen sich gewisse Objekte wiederholen zu mehrdeutigen Ergebnissen führen[9]. In der Praxis wird deshalb häufig die Ratiodistanz $r = d(x, y)/d(x, y')$ verwendet, wobei y der beste Match und y' der zweitbeste Match zu x in I_2 ist. Somit werden ambigen Regionen in Bildern kleinere Werte zugeordnet und es ergibt sich die Wahrscheinlichkeit r, dass der gefundene Match korrekt ist. Die Vektordistanz beschreibt somit ein Maß für die Ähnlichkeit zweier Features. Mithilfe einer Nearest Neighbour Suche kann so ermittelt werden welches Feature aus I_2 am besten zu einem bestimmten Feature aus I_1 passt. Um die Nearest Neighbour Suche zu optimieren können auch *k-d Trees* oder andere Suchbäume verwendet werden [Low99]. Für den BRIEF Deskriptor von ORB wird in der Regel die Hamming-Distanz der Bitstrings der Deskriptoren verwendet. Um die

[8] Hier zeigt sich auch warum der kürzere Deskriptor von SURF beim Matching Performance Vorteile gegenüber dem von SIFT bringt [BTVG06].

[9] Beispielsweise bei einem Zaun wiederholt sich das selbe Muster immer wieder.

Nearest Neighbour Suche zu optimieren wird *Locality Sensitive Hashing* verwendet, sodass ähnliche Bitstrings so gehasht werden, dass sie mit großer Wahrscheinlichkeit in den selben Buckets landen.

3 Lokalisierung durch Bilderkennung

Im Rahmen dieser Bachelorarbeit wurde eine Augmented Reality Anwendung für Android[1] entwickelt. Ziel dieser Anwendung war, dass insbesondere im Nahbereich zuvor aufgenommene Objekte erfolgreich wiedererkannt werden. Es sollte dem Benutzer also möglich sein ein beliebiges Objekt mit einem festen Standort in eine Datenbank aufzunehmen. Kehrt der Benutzer später an diesen Standort zurück und fokussiert das Objekt mit dem Sucher einer Kamera, sollte das Objekt wiedererkannt und auf dem Display eine zuvor eingegebene Bezeichnung für das Objekt angezeigt werden.

3.1 Lokalisierung von Smartphones

Um die Lokalisierungsfunktionalität zu gewährleisten bieten moderne Smartphones eine Vielzahl an Sensoren, wie

- einen GPS- und/oder GLONASS-Chip zur Lokalisierung,
- einen Gravitationssensor, der die Erdbeschleunigung in drei Achsen misst, um die Lage des Geräts festzustellen,
- einen Magnetfeldsensor, der als Kompass dient um die Absolutausrichtung des Geräts festzustellen,
- sowie Software Implementierungen, welche verschiedene Sensoren vereinen um größere Genauigkeit zu erhalten[2] [and].

[1] Mobilbetriebssystem für Smartphones, `http://www.android.com/`.

[2] Beispielsweise der Fused Location Provider, welcher GPS, Wifi Signale, und Funksignale vereinigt um die Geräteposition so genau wie möglich zu berechnen.

In der Praxis zeigt sich jedoch, dass die vorhandenen Sensoren häufig sehr ungenau sind. Wifi und Funksignale sind insbesondere in ländlichen Regionen nicht genügend vorhanden um eine genaue Lokalisierung zu ermöglichen. Für GPS- und GLONASS-Chips wurde festgestellt, dass selbst bei guten Bedingungen Abweichungen von mehr als zehn Metern häufig auftreten. Auch der Magnetfeldsensor weißt häufig Abweichungen von mehr als 5 Grad gegenüber dem magnetischen Nordpol auf, oder sogar mehr, wenn sich Störquellen in der Nähe des Smartphones befinden. Diese Ungenauigkeiten haben sich in einer ersten Version der Anwendung gezeigt, welche sich lediglich auf die Smartphone-internen Sensoren verließ. Während bei großen Entfernungen von mehr als 200 Metern von Objekten immer korrekt eine grobe Richtung festgestellt werden konnte, in der sich das gesuchte Objekt befindet, waren im Nahbereich von weniger als 50 Metern sehr große Ungenauigkeiten vorhanden.

3.2 Berechnung des augmentierten Bildes

Die zuvor genannten Sensoren reichen theoretisch um das augmentierte Bild für den Benutzer zu erzeugen. Wie in Abb. 3.1 deutlich wird sind hier nur einige Winkelberechnungen nötig.

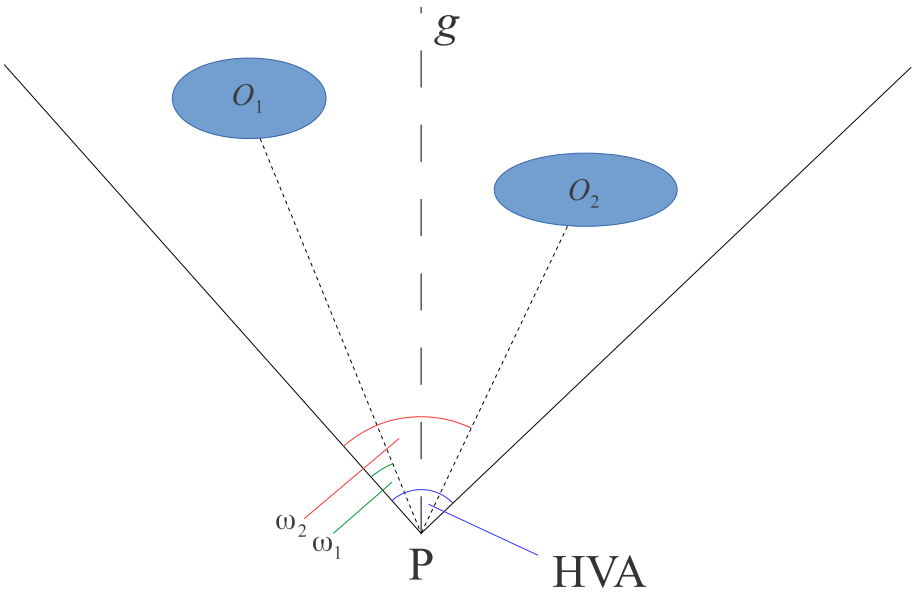

Abbildung 3.1: Die benötigten Werte zur Berechnung von Labelpositionen

O_1 und O_2 stellen Objekte in der Welt mit festen Koordinaten dar. Außerdem sei c der Kompasswinkel der vom Gerät an der Position P gemessen wurde, also der Absolutwinkel in Richtung der Geraden g mit Richtungsvektor $\overrightarrow{N}$. Der Winkel HVA steht für *Horizontal Viewing Angle*, also den horizontalen Öffnungswinkel der Kamera. Dieser unterscheidet sich von Kamera zu Kamera und muss daher ebenfalls in die Berechnung einfließen. Der Kompasswinkel des linken Bildrandes lässt sich somit durch $c' = c - \frac{\text{HVA}}{2}$ berechnen. Des Weiteren lassen sich Winkel ω_n von der Kamerablickrichtung zu den beiden Objekten O_1 und O_2 berechnen[3]. So ergibt sich

$$\omega_n = c \pm acos \frac{(\overrightarrow{O_n} - \overrightarrow{P}) \cdot \overrightarrow{N}}{|\overrightarrow{O_n} - \overrightarrow{P}| \cdot |\overrightarrow{N}|} - c'.$$

Nun können anhand der Objektwinkel ω_n, des Viewing Angles HVA, der Blickrichtung c, sowie der Bildschirmbreite x in Pixeln die horizontalen Labelpositionen berechnet werden, indem die Winkelratio in Pixel umgerechnet wird. Es ergibt sich also für jedes Objekt in der Szene ein

$$\chi_n = \frac{\omega_n}{\text{HVA}} \cdot x,$$

welches das Zentrum des Objektes im Bild in Pixeln darstellt. Um das Label auch in vertikaler Richtung auszurichten, kann der Gravitationssensor anstelle des Kompasses, sowie die vertikale anstelle der horizontalen Bildschirmgröße verwendet werden. Außerdem muss der *Vertical Viewing Angle* VVA anstelle des HVA verwendet werden. Für den implementierten Prototyp wurde auf die Eingabe von Objekthöhen verzichtet und für alle Objekte eine feste Höhe, die der Höhe des Gerätes über dem Boden entspricht, angenommen. Die Berechnung erfolgt analog zur Berechnung der horizontalen Position.

Während der Kompasswinkel c und die Position P von den Sensoren des Gerätes geliefert werden, und die Objekte O_n zuvor eingegeben wurden, müssen der HVA und der VVA ermittelt werden. Android stellt die Funktionen $getHorizontalViewAngle()$ und $getVerticalViewAngle()$ bereit, diese liefern jedoch ausschließlich Daten über den verbauten Kamerachip. Der tatsächliche Aufnahmewinkel der Kamerabilder, die von einem Entwickler verarbeitet werden, kann nicht über die Android API ermittelt werden. Daher ist eine Kamerakalibrierung, wie sie im nachfolgenden Unterkapitel

[3]Hierfür wird von einem kartesischen Koordinatensystem ausgegangen. Für den Prototyp wurde eine Mercatorprojektion, die sogenannte *Google Projection*, verwendet.

erklärt wird, notwendig, um diese Daten zu erhalten.

3.3 Kamera Kalibrierung

Es existieren eine Vielzahl von Möglichkeiten eine Kamera zu kalibrieren. Je nach Anforderung werden unterschiedliche Ziele mit der Kalibrierung verfolgt, wie beispielsweise

- das Ermitteln der optischen Achsen,
- das Ermitteln von Linsenverzerrungen im Kamerabild um das Bild später zu normalisieren oder
- das Ermitteln von Kamerawinkeln.

Für die hier verwendeten Algorithmen SIFT, SURF und ORB wird angenommen, dass die meisten Kalibrierungsarten keine ausschlaggebenden Auswirkungen auf die Qualität der Ergebnisse haben, da diese Algorithmen dafür konzipiert sind robust gegenüber geringen Verzerrungen zu sein. Deshalb werden hier lediglich die Kamerawinkel ermittelt, da sich diese stark von Gerät zu Gerät unterscheiden und einen relevanten Einfluss auf die Berechnung des augmentierten Bildes haben. Ein wichtiges Kriterium für die breite Anwendung in Augmented Reality Anwendungen ist, dass die Kalibrierung möglichst einfach mit Hausmitteln umsetzbar ist, da diese aufgrund der Vielzahl an unterschiedlichen Geräten im Regelfall vom Endbenutzer durchgeführt werden muss. Somit wurden im Rahmen der Arbeit zwei praktische Methoden implementiert, um den Vorgang möglichst einfach und auch für Endbenutzer umsetzbar zu gestalten.

3.4 Ermittlung der Kamera Öffnungswinkel mit Hilfe der Sensoren

Die erste Implementierung verwendet die im Smartphone vorhandenen Sensoren um die Kamerawinkel zu ermitteln. Es werden keine anderen Hilfsmittel benötigt. Um die Kalibrierung durchzuführen muss der Nutzer einen beliebigen Fixpunkt im Bild wählen. Anschließend wird dieser Punkt durch Drehen des Smartphones in vertikaler

und horizontaler Richtung ein mal auf jede Seite des Kamerabildes gebracht, wie in Abb. 3.2 gezeigt wird.

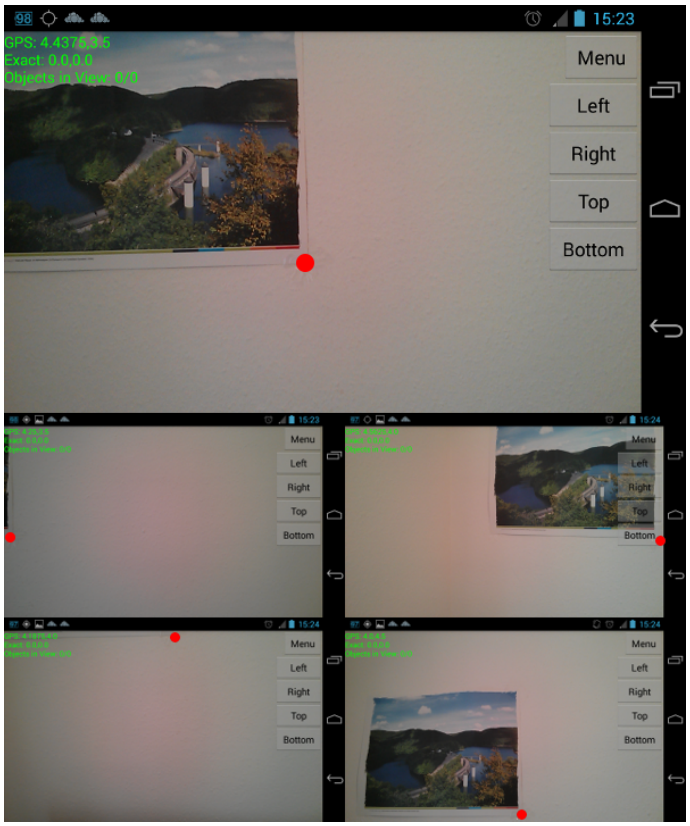

Abbildung 3.2: Kalibrierung der Kamera mit Hilfe der Android Sensoren. Ein zuvor gewählter Punkt wird durch Kamerarotation an jede Bildschirmkante gebracht und der entsprechende Button wird gedrückt.

Jedes mal wenn der Benutzer den ausgesuchten Punkt an eine Bildschirmkante gebracht hat, drückt er den entsprechenden Button. Die Anwendung merkt sich dann für den *Left* und den *Right* Button die Kompassrotation des Geräts. Für den *Top* und den *Bottom* Button, merkt sich die Anwendung die Lage des Geräts mit Hilfe des Gravitationssensors. Hat der Benutzer den Kalibrierungsvorgang abgeschlossen, den gesuchten Punkt also durch Kamerarotation an jede Bildschirmkante gebracht, kann nun einfach durch die Differenz zwischen Bottom und Top der vertikale Kamerawinkel und analog aus der Differenz zwischen Right und Left der horizontale Kamerawinkel berechnet werden. Um so weiter der gewählte Punkt von der Kamera entfernt ist, desto exakter lassen sich die Winkel mit diesem Verfahren berechnen. Insbesondere für den horizontalen Öffnungswinkel zeigte sich in Versuchen jedoch

wieder die Ungenauigkeit des Kompasses, der Abweichungen bis zu 10 Grad verursachte. Für den vertikalen Öffnungswinkel konnten hingegen gute Werte mit diesem Verfahren ermittelt werden.

3.5 Ermittlung der Kamera Öffnungswinkel mit Hilfe von Hausmitteln

Bereits mit wenigen Hausmitteln lässt sich das obige Verfahren verbessern um Öffnungswinkel zu ermitteln, die sich um weniger als zwei Grad vom tatsächlichen Wert unterscheiden. Die benötigten Hilfsmittel belaufen sich auf ein Blatt Papier, einen Stift und ein Lineal. Zunächst werden zwei parallele Linien mit einem Abstand von n Zentimetern auf das Papier gezeichnet. Nun wird das Smartphone direkt auf die eine Linie gelegt und das Lineal auf die andere Linie, wie es in Abb. 3.3 gezeigt wird. Durch die Kamera des Smartphones wird nun abgelesen, wie viele Zentimeter des Lineals sichtbar sind. Dies ergibt die sichtbare Breite l_1. Nun wird die Kamera um 90 Grad gedreht und die sichtbare Höhe l_2 gemessen.

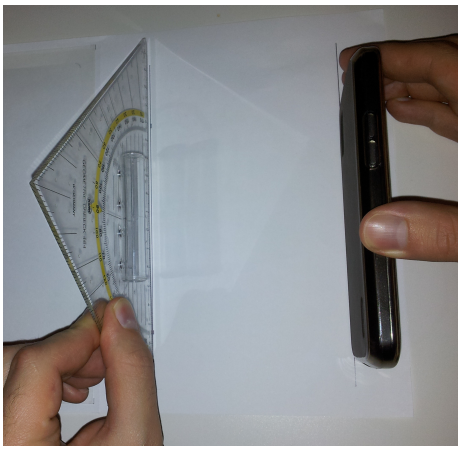

Abbildung 3.3: Kamera Öffnungswinkel mit einem Lienal ermitteln. Hier mit $n = 10cm$

Der horizontale und vertikale Öffnungswinkel ergibt sich dann durch die Berechnung von HVA $= tan\frac{l_1}{n}$, beziehungsweise VVA $= tan\frac{l_2}{n}$.

3.6 Lokalisierung der Kamera

Wie bereits beschrieben sind die in derzeitigen Smartphones verbauten Sensoren zu ungenau um mit den obigen Mitteln ein brauchbares augmentiertes Bild im Nahbereich zu erhalten. Aus diesem Grund wurden in in Kapitel 2 beschriebenen Algorithmen verwendet um die Genauigkeit zu erhöhen. Um die Objekterkennung in eine Augmented Reality Anwendung zu integrieren gibt es zwei grundlegende Herangehensweisen.

- Die triviale Implementierung erkennt Objekte im Bild. Sobald ein Objekt erkannt wurde, wird die entsprechende Bezeichnung darüber gezeichnet (1).
- Alternativ können die erkannten Objekte zusammen mit Hilfe des Kompass verwendet werden, um eine exakte Position der Kamera zu berechnen. Die Position der Objektbezeichnung wird dann wie zuvor mit Hilfe des Kompass berechnet (2).

Beide Lösungen bieten einige Vor- und Nachteile, wie sie in Kap. 3.6 kurz beschrieben werden. False Positives steht dabei dafür, dass der Objekterkennungsalgorithmus ein Objekt erkannt hat das nicht existiert. False Negatives steht dafür, dass ein existierendes Objekt nicht erkannt wurde.

Tabelle 3.1: Vor- und Nachteile der Verfahren zur Integration von Objekterkennung

Merkmal	**Label über gefundenem Objekt (1)**	**Berechnung der Kameraposition (2)**
False Positives	Eine Bezeichnung wird fälschlicherweise eingeblendet.	Kameraposition wird falsch berechnet. Ein False Positive kann zu vielen falschen Bezeichnungen führen.
False Negatives	Die Bezeichnung des Objekts fehlt.	Ungefähre Position der Bezeichnung kann mit Hilfe von Kompass oder anderen, korrekt gefunden Objekten im Bild eingezeichnet werden.
Echtzeit	Nicht gegeben. Selbst bei schnellen Algorithmen (ORB) hängt die Anzeige.	Echtzeitgefühl für den Benutzer. Die Kameraposition ist nicht immer korrekt, die Bezeichnungen verhalten sich jedoch flüssig.
Genauigkeit	Exakt. Bezeichnungen können korrekt platziert werden.	Geringe Ungenauigkeiten, da immer noch mit Kompass und Öffnungswinkel gerechnet wird. Diese fallen hier jedoch deutlich weniger ins Gewicht als bei der Variante ohne Bilderkennung.

Insbesondere auf Grund der Echtzeitwirkung und dem Ausgleich von False Negatives des Kamera Lokalisierungsverfahrens (2), wurde für die Implementierung diese Variante gewählt.

3.6.1 Berechnung der Kameraposition

Um die Kameraposition mit Hilfe von Objekterkennung zu errechnen wurde das folgende Verfahren gewählt:

1. Wurde kein Objekt auf dem Kamerabild gefunden wird die GPS oder GLONASS Position für die Kamera verwendet.
2. Wurde ein einzelnes Objekt auf dem Kamerabild gefunden wird das Lot von

der GPS/GLONASS Position auf die Objektrichtung[4] gefällt. Der Lotfußpunkt stellt dann die Position der Kamera dar.

3. Wurden mehrere Objekte gefunden wird für jedes Objekt die Objektrichtung als Richtungsvektor ermittelt. Anschließend werden die Schnittpunkte aller Richtungsvektoren berechnet und von diesen Schnittpunkten wiederum der Mittelwert berechnet. Dieser bildet dann die Kameraposition

Ein Objektvektor stellt dabei die globale Kompassrichtung dar, in der ein Objekt liegt. Um diese zu berechnet wird zunächst der Kompasswinkel der linken Bildkante $c' = c - \frac{\text{HVA}}{2}$ ermittelt. Der horizontale Objektwinkel für ein Objekt n ergibt sich dann durch

$$\omega_n = c' + \text{HVA} \cdot \frac{n_x}{X},$$

wobei n_x den Bildpixel in x-Richtung darstellt an dem das Objekt gefunden wurde, und X die Bildbreite in Pixeln definiert. Analog lässt sich so auch der vertikale Objektwinkel durch Verwendung des VVA, der Bildschirmhöhe Y und des Lagesensors anstelle der Kompassrichtung berechnen.

3.7 Implementierung eines Prototyps

Die hier vorgestellten Verfahren wurden zur Implementierung eines Prototyps für Android Smartphones verwendet. Der Prototyp bietet eine Benutzeroberfläche mit der der Benutzer die Kamera kalibrieren und Objekte aufnehmen kann. Außerdem ist ein Augmented Reality Borwser vorhanden in dem die zuvor aufgenommenen Objekte dann mit Bezeichnungen versehen werden, sobald die Kamera das Objekt in der Welt fokussiert hat.

3.7.1 Aufnehmen von Objekten

Die Aufnahme von Objekten erfolgt in mehreren Schritten durch den Benutzer.

1. Der Benutzer öffnet das Programm Menü und drückt auf *Add Object*
2. Daraufhin öffnet sich ein Dialog in dem der Benutzer eine Bezeichnung des Objekts angeben kann, die später im augmentierten Bild über dem Objekt gezeichnet werden soll.

[4] Ermittelt mit Kompass und Öffnungswinkel, wie in Kap. 3.2 beschrieben.

3. Anschließend öffnet sich eine Karte mit Satellitenbildern und einem Marker. Der Marker befindet sich initial an der GPS Position des Benutzers. Der Benutzer kann diesen nun an die tatsächliche Position des aufzunehmenden Objekts verschieben.
4. Nun öffnet sich die Kamera Anwendung von Android und der Benutzer kann ein Foto des Objekts erstellen, welches zur späteren Wiedererkennung dient.
5. Schließlich wird das Objekt mit den gegebenen Informationen auf einem Server gespeichert.

3.7.2 Kommunikation und Speicherung auf dem Server

Für die Kommunikation zwischen der Augmented Reality Anwendung und dem Server dient eine REST Schnittstelle, die in PHP implementiert wurde. Die Schnittstelle bietet dabei die folgenden Funktionen:

- *createObject* mit den Parametern *lat*, *lon*, *label*, *photo* und *keypoints* fügt ein neues Objekt in die Datenbank ein. *lat* und *lon* bestimmen die Koordinaten, *label* die Objektbezeichnung und *photo* enthält die aus dem Foto extrahierten Deskriptoren. Um die benötigte Bandbreite zu reduzieren werden diese Deskriptoren zusätzlich per *GZIP*[5] komprimiert. Der Parameter *keypoints* enthält die Positionen an denen die jeweiligen Deskriptoren im Bild extrahiert wurden. Diese werden benötigt um die exakte Position eines Matches im Kamerabild zu errechnen.
- *getObjects* mit den Parametern *lat*, *lon* und *distance* gibt eine Liste aller Objekte zurück die sich innerhalb des Radius *distance* um den Punkt (lat, lon) befinden.
- *setCalibration* mit den Parametern *device*, *hva* und *vva* dient zum Speichern von Kalibrierungsdaten für ein bestimmtes Gerät.
- *getCalibration* mit dem Parameter *device* gibt die gespeicherten Kalibrierungsdaten für ein bestimmtes Gerät zurück. Somit muss die Kalibrierung für jedes Gerät nur ein einziges Mal vollzogen werden.

[5] Verbreitetes Kompressionsverfahren, siehe `http://www.gzip.org/`.

Zur Speicherung der Daten wurde eine PostgreSQL[6] Datenbanksystem mit der Erweiterung PostGIS[7] gewählt. PostGIS ist eine Erweiterung für PostgreSQL Server, die diverse Funktionen, Datentypen und Indizes für Geoinformationssysteme und geometrische Berechnungen bietet. Mithilfe von PostGIS lässt sich insbesondere das Auffinden von Objekten innerhalb eines Radius um einen Punkt, wie es für die *getObjects* Anfrage benötigt wird, effizient erreichen.

[6]Datenbanksystem, siehe `http://www.postgresql.org/`.
[7]Siehe `http://postgis.net/`.

4 Evaluation

Um die Leistungsfähigkeit der Augmented Reality Anwendung zu evaluieren wurden die drei Algorithmen SIFT, SURF und ORB miteinander bezüglich verschiedener Kriterien verglichen. Als Testrechner für den Performancevergleich wurde ein PC mit der folgenden Hardware Konfiguration verwendet:

- CPU: Intel Core i7 2600K mit 3.4 Ghz
- Arbeitsspeicher: 16 GB
- Betriebssystem: Ubuntu Linux 13.04 mit Linux Kernel 3.11

Die für die Tests verwendeten Bilddaten wurden mit einem Google/LG Nexus 4 erstellt, um eine Signal-to-Noise Ratio zu erhalten wie sie auch in der Praxis vorzufinden ist. Außerdem wurden die Bilder auf eine Größe von 1280x960 Pixel skaliert, da größere Bilder nur schwer in annehmbarer Zeit auf Mobilgeräten verarbeitet werden können. Als Implementierungsgrundlage wurden die Algorithmen der OpenCV Bibliothek verwendet. Um die Ergebnisse dieser Evaluation nachzuvollziehen können die Testdaten, sowie die verwendeten Programme und der Augmented Reality Prototyp, herunter geladen werden[1].

Um die Qualität Q der Algorithmenausgabe E bezüglich der Ground Truth G zu bewerten, wurde das Standardverfahren

$$Q = \frac{|E \cap G|}{|E \cup G|} \cdot 100$$

verwendet.

[1] Siehe `http://www.abatzill.de/ar-thesis`.

4.1 Skalierungsinvarianz

Um Skalierungsinvarianz bei den genannten Algorithmen zu erreichen ist es vor Allem wichtig die selben Keypoints in verschiedenen Skalierungen zu finden. Um die Skalierungsinvarianz zu überprüfen wurde ein Objekt aus den Distanzen 0,5, 1, 2, 3 und 4 Metern aufgenommen und überprüft wie gut die 0,5 Meter Aufnahme in den anderen Aufnahmen gefunden wurde.

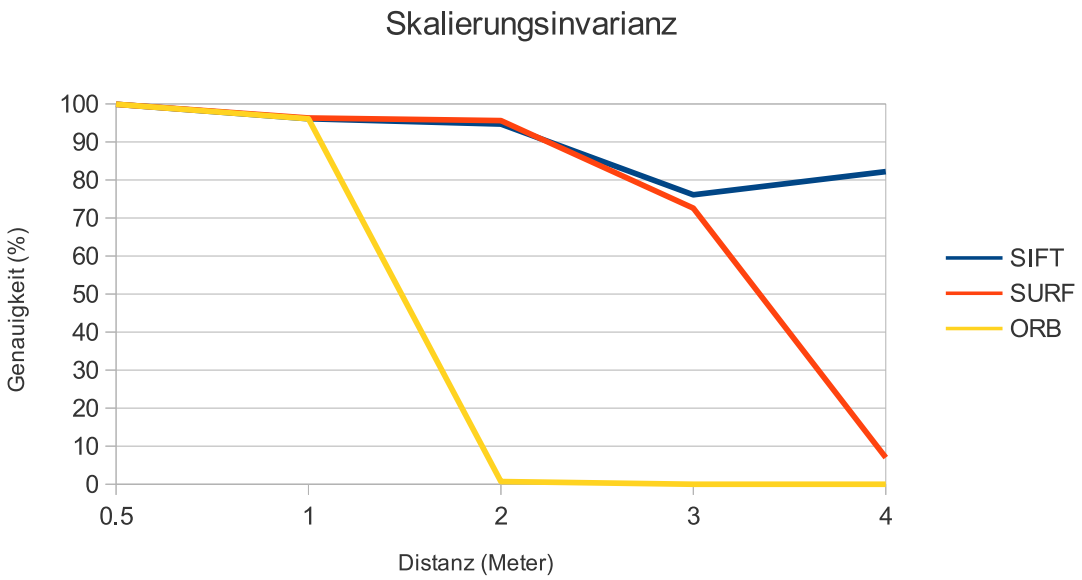

Abbildung 4.1: Algorithmenvergleich bei Skalierungen von 0,5 bis 4 Metern.

Das Diagramm in Abb. 4.1 zeigt, dass insbesondere ORB größere Probleme bei der Skalierungsinvarianz aufweist, während SURF bis zu einem Skalierungsfaktor von 6, also bei einer Distanz von 3 Metern, brauchbare Ergebnisse ($Genauigkeit > 70\%$) und SIFT sogar bei einem Skalierungsfaktor von 8, also bei einer Distanz von 4 Metern, noch gute Ergebnisse liefert.

4.2 Rotationsinvarianz

Um Rotationsinvarianz zu evaluieren wurde eine Szene zunächst normal fotografiert. Anschließend wurde die Kamera schrittweise zwischen 0 und 180 Grad gedreht um die Szene erneut mit unterschiedlicher Rotation aufzunehmen. Wie in Abb. 4.2 zu

sehen ist weisen alle untersuchten Algorithmen ein hohes Maß an Rotationsinvarianz auf. Bei den vorhandenen Abweichungen wird davon ausgegangen, dass diese durch Rauschen im Bild oder geringe Ungenauigkeiten bei der Bestimmung der Ground Truth entstanden sind. Rotationsinvarianz ist vor Allem dafür wichtig, dass Objekte auch korrekt erkannt werden, wenn der Benutzer die Kamera nicht exakt gerade hält.

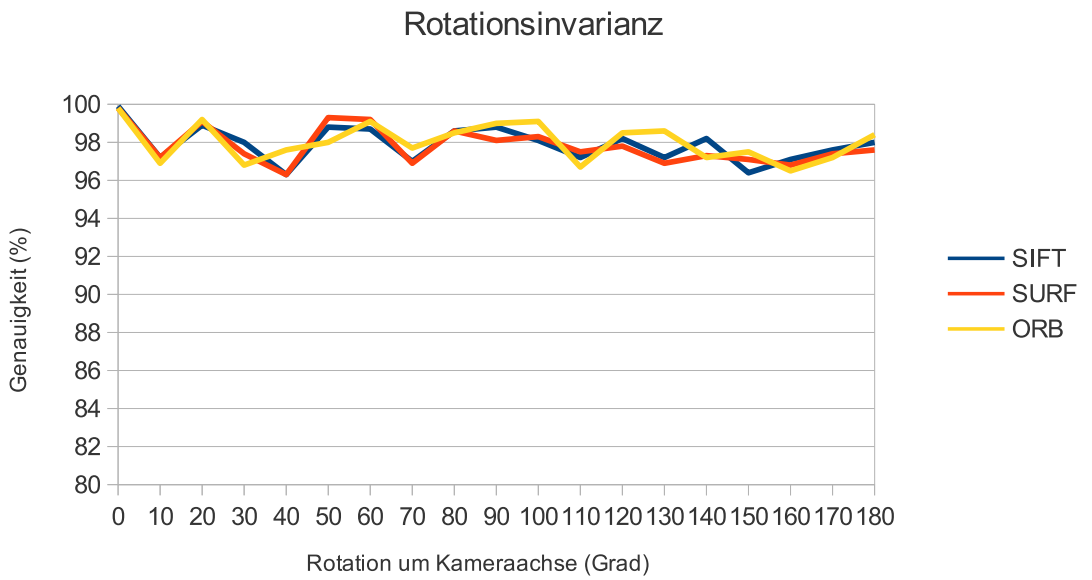

Abbildung 4.2: Algorithmenvergleich bei Rotationen zwischen 0 und 180 Grad.

4.3 Einfache Transformation

Unter einfacher Transformation wird hier verstanden, dass eine weitgehend flache Szene (also beispielsweise eine Häuserfassade) aus unterschiedlichen Winkeln aufgenommen wird. Es handelt sich dabei um eine affine Transformation in der realen Welt. Durch die Fluchtpunktprojektion der Kamera ist die Transformation im Bild jedoch nicht mehr affin, da durch die verschiedenen Winkel der Fluchtpunkt des Bildes verschoben wird und somit parallele Linien im Bild nicht parallel bleiben. Die Ergebnisse sind in Abb. 4.3 zu sehen. Während SIFT und SURF hier bis zu einem Winkel von 40 Grad sehr gute Ergebnisse liefern, wird ORB ab einem Winkel von etwa 30 Grad instabil. und bietet somit ein geringeres Maß an Robustheit gegenüber einfachen geometrischen Transformationen.

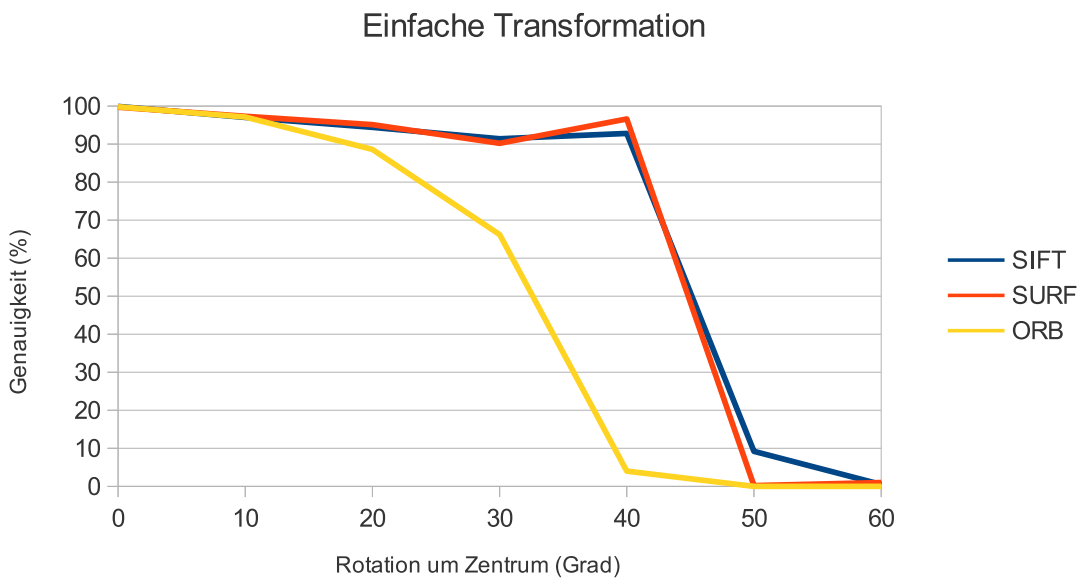

Abbildung 4.3: Algorithmenvergleich bei einfachen Transformationen.

4.4 Komplexe Transformation

Komplexe Transformation bedeutet hier, dass die fotografierte Szene keine Annäherung an eine Fläche darstellt, sondern dass sich mehrere Objekte in verschiedenen Tiefen im Bild befinden. Durch die anschließende Rotation der Kamera um den Szenenmittelpunkt verschieben sich Objekte, die nahe an der Kamera sind, in eine andere Richtung als Objekte, die weit von der Kamera entfernt sind, was die Erkennung deutlich erschwert. Wie in Abb. 4.4 zu sehen ist hat insbesondere SURF Schwierigkeiten mit dieser Art von Transformation und erkennt bereits bei einem Rotationswinkel von 20 Grad keine Bildinhalte mehr. SIFT und ORB hingegen bleiben bis zu einem Winkel von 20 Grad stabil, sind bei 30 Grad jedoch bereits unbrauchbar.

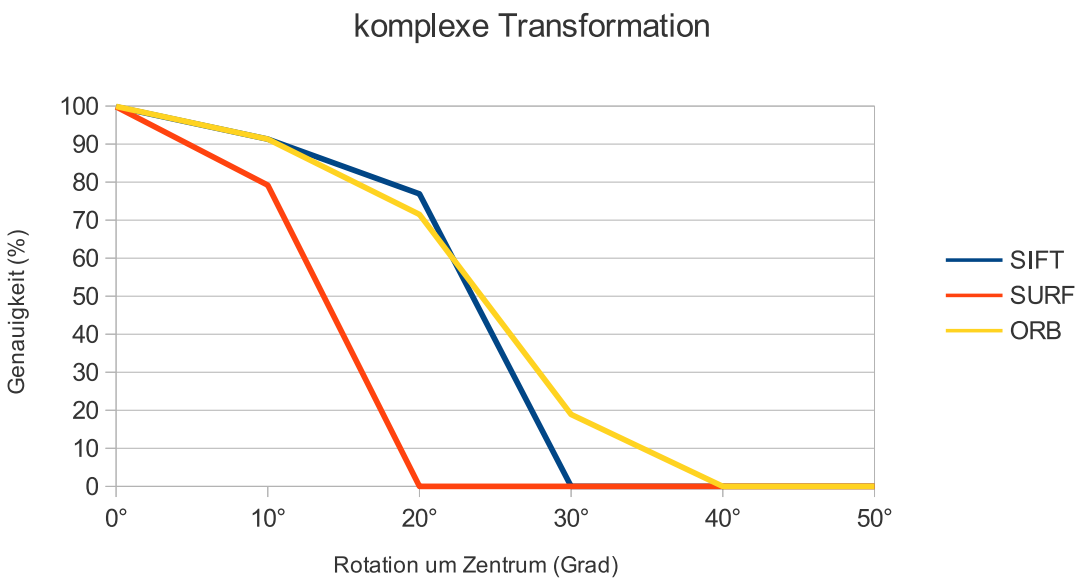

Abbildung 4.4: Algorithmenvergleich bei komplexen Transformationen.

4.5 Performance

Schließlich wurde noch die Performance der unterschiedlichen Algorithmen auf der in Kapitel 4 genannten Hardware überprüft. Dazu wurde untersucht, wie viele Bilder der jeweilige Algorithmus in einer Sekunde verarbeiten kann. Die Verarbeitung eines Bildes wird dabei als kompletten Durchlauf aller Stadien des Algorithmus, also Feature Detection, Extraction und Matching, beschrieben. Die in Abb. 4.5 gezeigten Ergebnisse bestätigen, was auch in [RRKB11] und [Low99] ermittelt wurde: Während SURF um einen Faktor von 3,75 schneller arbeitet als SIFT, ist ORB um einen Faktor von 4,44 schneller als SURF und bietet somit trotz den Einbußen bei Skalierungs- und Transformationsinvarianz eine berechtigte Alternative, insbesondere für Mobil- und Echtzeitsysteme.

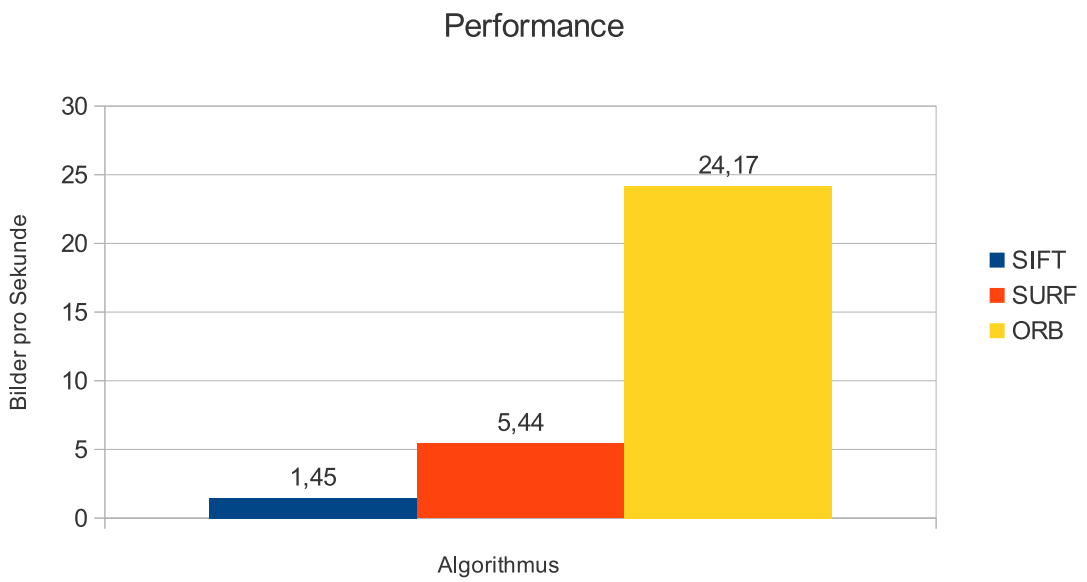

Abbildung 4.5: Algorithmenvergleich bezüglich ihrer Geschwindigkeit.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Verwendung von den modernen Computer Vision Algorithmen SIFT, SURF und ORB für Augmented Reality Anwendungen evaluiert und die Integration der Algorithmen in solche Anwendungen beschrieben. In Kapitel 2 wurden zunächst alle Algorithmen kurz beschrieben um ein Verständnis für die Arbeitsweisen zu vermitteln. Dabei wurde eine Unterscheidung zwischen Feature Detection, Descriptor Extraction und Matching eingeführt. In Kapitel 3 wurde schließlich erörtert, wie sich diese Algorithmen sinnvoll in eine Augmented Reality Anwendung integrieren lassen. Dazu wurde beschrieben, wie sich die Position einer Kamera möglichst exakt berechnen lässt, indem Daten von GPS, einem integrierten Magnetfeldsensor und der Kamera verrechnet werden. Auch wurde diskutiert welche Vorteile die Lokalisierung der Kamera gegenüber der reinen Objekterkennung bieten. Anhand eines für diese Arbeit implementierten Prototyps wurde erläutert, wie sich die Kamera eines Smartphones mit möglichst einfachen Hausmitteln kalibrieren lässt um die Öffnungswinkel zu errechnen. Kapitel 4 evaluiert die vorgestellten Bilderkennungsalgorithmen nach ihrer Leistung bezüglich Skalierungsinvarianz, Rotationsinvarianz, Robustheit gegenüber einfachen und komplexen Transformationen, sowie deren Performance.

Dabei hat sich gezeigt, dass insbesondere ORB sich gut für Mobilsysteme eignet und schnell genug war, eine interaktive Lokalisierung ohne große Latenzen zu gewährleisten.

Bei der Entwicklung des Prototyps sind verschiedene Möglichkeiten ermittelt worden um die Qualität des Prototyps zu verbessern. Zunächst wäre es denkbar, Algorithmen zu verwenden die keine Rotationsinvarianz gewährleisten. Da diese Algorithmen keine Annahme über die Ausrichtung eines Bildfeatures machen, könnte damit insgesamt eine höhere Erkennungsrate erreicht werden. Um die Augmented Reality Anwendung dennoch robust gegenüber Rotationen zu machen, könnte der Lagesensor des Smartphones verwendet werden um die Bildrotation zu normalisieren. Die

Annahme ist dabei, dass eine Rotation bei Augmented Reality Anwendungen nicht dadurch auftritt, dass sich die Szene selbst rotiert, sondern dadurch dass die Kamera rotiert wird. Mithilfe des Lagesensors könnte diese Kamerarotations wieder entfernt werden, sodass nicht-rotationsinvariante Algorithmen verwendet werden können.

Auch wäre denkbar, dass der Benutzer bei der initialen Aufnahme eines Objektes zusätzlich die Höhe dieses Objektes angeben muss. Unter der Annahme, dass ein Benutzer bei der späteren Rückkehr zu diesem Objekt (also wenn das Objekt automatisch im Bild gefunden werden soll) die Mitte des Objektes fokussiert, lässt sich so mit Hilfe des Lagesensors eine Distanz vom Benutzer zum Objekt errechnen. Diese könnte zusätzlich helfen die Lokalisierung der Kamera zu verfeinern. In Abb. 5.1 ist der grundlegende Aufbau zu sehen. Die benötigten Werte zum Berechnen der Distanz sind beispielhaft eingezeichnet.

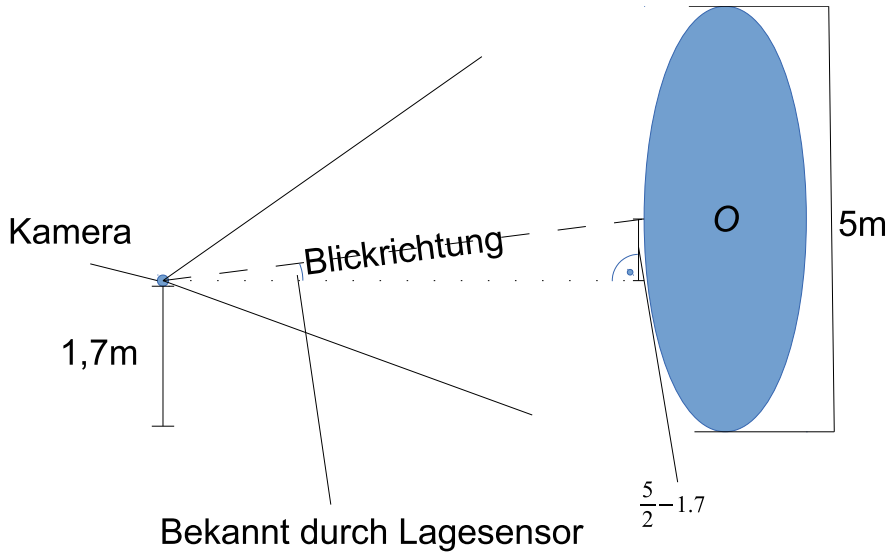

Abbildung 5.1: Berechnung der Distanz zu einem Objekt mit Hilfe des Lagesensors

In Kap. 3.6 wurden die Vor- und Nachteile von der Berechnung der Kameraposition durch Objektlokalisierung mit der direkten Anzeige des Labels über gefundenen Objekten verglichen. Als weitere Verbesserung wäre hier ein Hybrid aus beiden Lösungen denkbar. Wenn ein Objekt im Bild gefunden wurde, wird das Label direkt eingezeichnet. Zusätzlich wird die Position der Kamera berechnet um gegebenenfalls nicht gefundene Objekte approximativ einzuzeichnen. Dieses Hybridvorgehen hätte den Vorteil, dass alle gefundenen Objekte exakte Label erhalten. Gleichzeitig wird jedoch der Nachteil eingeführt, dass die Anwendung nicht mehr interaktiv

wirkt, wenn der Objekterkennungsalgorithmus nicht schnell genug ist. Ein solches Vorgehen wäre daher nur auf sehr schnellen Smartphones in Kombination mit ORB denkbar.

Literaturverzeichnis

[and] *Sensors Overview*. `http://developer.android.com/guide/topics/sensors/sensors_overview.html`

[BTVG06] Bay, Herbert ; Tuytelaars, Tinne ; Van Gool, Luc: Surf: Speeded up robust features. In: *Computer Vision–ECCV 2006*. Springer, 2006, S. 404–417

[Eva09] Evans, Christopher: Notes on the OpenSURF Library / University of Bristol. 2009 (CSTR-09-001). – Forschungsbericht

[Hof12] Hoffmann, Alexandre: Augmented Reality: Navigation und standortbezogene Dienste mit Android-Smartphones. (2012)

[HS88] Harris, Chris ; Stephens, Mike: A combined corner and edge detector. In: *Alvey vision conference* Bd. 15 Manchester, UK, 1988, S. 50

[HS97] Hartley, Richard I. ; Sturm, Peter: Triangulation. In: *Computer vision and image understanding* 68 (1997), Nr. 2, S. 146–157

[KOTY00] Kanbara, Masayuki ; Okuma, Takashi ; Takemura, Haruo ; Yokoya, Naokazu: A stereoscopic video see-through augmented reality system based on real-time vision-based registration. In: *Virtual Reality, 2000. Proceedings. IEEE* IEEE, 2000, S. 255–262

[Low99] Lowe, David G.: Object recognition from local scale-invariant features. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* Bd. 2 Ieee, 1999, S. 1150–1157

[Low04] Lowe, David G.: Distinctive image features from scale-invariant keypoints. In: *International journal of computer vision* 60 (2004), Nr. 2, S. 91–110

[RPD10] Rosten, Edward ; Porter, Reid ; Drummond, Tom: Faster and better: A machine learning approach to corner detection. In: *Pattern Ana-*

lysis and Machine Intelligence, IEEE Transactions on 32 (2010), Nr. 1, S. 105–119

[RRKB11] Rublee, Ethan ; Rabaud, Vincent ; Konolige, Kurt ; Bradski, Gary: ORB: an efficient alternative to SIFT or SURF. In: *Computer Vision (ICCV), 2011 IEEE International Conference on* IEEE, 2011, S. 2564–2571

[RSF+06] Ravi, Nishkam ; Shankar, Pravin ; Frankel, Andrew ; Elgammal, Ahmed ; Iftode, Liviu: Indoor localization using camera phones. In: *Mobile Computing Systems and Applications, 2006. WMCSA'06. Proceedings. 7th IEEE Workshop on* IEEE, 2006, S. 49–49

[Sin10] Sinha, Utkarsh. *SIFT: Scale Invariant Feature Transform.* `http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/`. 2010

[Vis] Viswanathan, Deepak G. *Features from Accelerated Segment Test (FAST).* `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/AV1FeaturefromAcceleratedSegmentTest.pdf`