Institut für Mensch-Computer-Interaktion
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Fachstudie Nr. 176

Personenerkennung über verschiedene Geräte

David Krauss, Mathias Landwehr,
Paul Brombosch

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Albrecht Schmidt
Betreuer:	Tilman Dingler, Markus Funk
begonnen am:	15.04.2013
beendet am:	15.10.2013
CR-Klassifikation:	H.1.2

Kurzfassung

Im Laufe der Fachstudie wurde ermittelt welches Kamera-/Softwaresystem sich am besten zur Erkennung und Wiedererkennung von Personen eignet. Der Grundgedanke ist es, personenbezogene Inhalte auf einem Display anzuzeigen. Nach den Recherchen haben sich zwei APIs zum genaueren Vergleich heraus kristallisiert, nämlich Kinect SDK und OpenCV. Um einen Vergleich durchführen zu können, wurde jeweils eine Anwendung implementiert, die Gesichter in einem Video erkennt. Der Vergleich zeigte deutlich die Stärken und Schwächen der Implementierungen in mehreren Kategorien. Das Kinect SDK zeichnete sich durch viele Vorteile bei den Personenwahrnehmung aus, wo hingegen OpenCV bei der Wiedererkennung hervorstechen konnte. Das endgültig implementierte System verwendet deshalb Kinect SDK zur Gesichtswahrnehmung und OpenCV zur Wiedererkennung und Kategorisierung.

Inhaltsverzeichnis

1. Aufgabenbeschreibung

In unserer heutigen Gesellschaft, in der Informationen immer schneller zur Verfügung stehen müssen, sind Anzeigeflächen nicht mehr wegzudenken. Um unnötige Informationsflut zu vermeiden, sollten Inhalte auf digitalen Anzeigeflächen passend auf den Benutzer zugeschnitten werden. Personalisierung von Inhalten findet bereits in großem Maße im Web und auf mobilen Endgeräten statt. Im Bereich der „Public Display“ Netzwerken stellt sich die Frage danach, wie Passanten erkannt werden können, um personalisierte Informationen in Echtzeit darzustellen. Wenn Passanten darüber hinaus über mehrere Geräte hinweg erkannt werden, können Nutzerprofile generiert werden, welche Bewegungsmuster beinhalten sowie Buchführen über Inhalte, die Nutzern bereits angezeigt wurden.

Somit können ähnliche oder weiterführende Informationen über mehrere Displays hinweg angezeigt werden. Ein denkbares Szenario wäre ein klassischer Museumsbesuch: In Museen soll den jeweiligen Besuchern an Hand der bereits gesehenen Ausstellungsstücke ähnliche Informationen präsentiert werden, wie zum Beispiel Flyer mit weiteren Ausstellungsstücken, die im Interessensbereich des Besuchers liegen.

Ein weiteres Szenario wäre die zielgruppenorientierte Anzeige von Werbeinhalten: Während ein Passant auf einem Display ein Kinoplakat angezeigt bekommt, könnte auf dem nächsten Display, welches dieser passiert, beispielsweise ein Trailer des Films oder ein Rabattgutschein für das nahe gelegene Kino zu sehen sein.

1.1. Zweck des Dokuments

Dieses Dokument wurde in Verbindung mit einer Fachstudie erarbeitet. Es dokumentiert den Verlauf und die Entscheidung bei der Analyse der Studie und Entwicklung eines Tools für das VIS der Fakultät Informatik der Universität Stuttgart.

1.2. Überblick über den weiteren Inhalt

Im Kapitel 2 wird ein grober Überblick über das gesamte Projekt und die damit verbundene Zeitplanung gegeben. Im weiteren Verlauf wird auf die einzelnen APIs eingegangen. Kapitel 4 spiegelt die Vergleichsphase unseres Projekts wider, in der wir die Anforderungen an die jeweiligen APIs testen und abwägen. Somit folgen in Kapitel 5 unsere gewonnen Erkenntnisse und die daraus resultierenden Folgen für das weitere Projekt. Der technische Hintergrund und die Architektur unseren Anwendung werden in Kapitel 6 erläutert. In letzten zwei Kapitel wir auf den bestehenden so wie auf den erweiterbaren Funktionsumfang eingegangen.

2. Projektüberblick

Dieses Kapitel dokumentiert die Problematik und die strukturelle Herangehensweise der gesamten Fachstudie. Zu Beginn wird die ursprünglichen Aufgabenstellung und anschließend die verschiedenen Phasen des Projekts repräsentiert.

2.1. Aufgabenstellung

Aufgabe der Fachstudie ist es geeignete Systeme für die visuelle Analyse von Passanten zu vergleichen und zu bewerten. Mit Hilfe von Bilderfassungssystemen sollen Passanten klassifiziert und wiedererkannt werden. Public Displays sollen dann benutzerspezifische Inhalte anbieten.

Außerdem soll ein lauffähiges System implementiert werden, in Form einer verteilten Anwendung mit einem zentralen Server. Dieser soll in der Lage sein, Nutzerprofile zu speichern und eine Schnittstelle bieten, um diese Profile abzufragen, zu modifizieren, sowie Bilder entgegenzunehmen und zu analysieren.

2.2. Vorgehensweise

Auf Grundlage des Aufbaus einer Fachstudie teilt sich die die Arbeit in vier Phasen:

- Einarbeitungsphase
- Vergleichsphase
- Entscheidungsphase
- Entwicklungsphase

2.2.1. Einarbeitungsphase

Da dem Team bis dato keinerlei Erfahrungen mit bestehenden Technologien zur Gesichtserkennung bekannt waren, recherchierten wir erstmals nach bestehenden Lösungen und Bibliotheken.

Leider standen keine kostenfreie Anwendungen, die unsere Aufgabenstellung bereits realisieren, zur Verfügung. Wir haben uns deshalb auf folgende drei APIs fixiert:

- Kinect SDK Version 1.6
- OpenCV Version 2.4.6.0
- OpenNI

2.2.2. Vergleichsphase

In dieser Phase des Projekts stand der Vergleich der APIs im Mittelpunkt.
So stellten wir bereits bei der Zusammensetzung der Vergleichsparameter fest, dass dies nicht nicht wie vorgestellt laufen kann. Ursprünglich basierte unser Vergleich auf der Durchführung der Tests mit Hilfe von Bildern, leider konnten wir dies nicht für das Kinect SDK realisieren. Auf Grund dessen griffen wir auf Kinect-Studio zurück um mit Kinect-Sequenzen arbeiten zu können. Als weiteres Problem stellten wir fest, dass das Kinect SDK über keinerlei Algorithmen zur Wiedererkennung von Personen verfügt, wodurch wir unsere Vergleichsstudie auf die Wahrnehmung von Personen begrenzten.

2.2.3. Entscheidungsphase

Die Ergebnisse der Vergleichsphase wurden nun auf geeignete Weise verglichen und daraus die Struktur und die Zusammensetzung der einzelnen Komponenten für die Anwendung festgelegt. Aufgrund der bereits aufgewandten Arbeitszeit legten wir an dieser Stelle fest, den Umfang unserer Anwendung auf die Personen sowie Geschlechtserkennung zu begrenzen.

2.2.4. Entwicklungsphase

Durch die erfassten Erkenntnisse erstellen wir ein lauffähiges System und begannen mit der Erfassung unterschiedlicher Profile. Hierfür wurden Gesichter erfasst und Profilen sowie Geschlechtern zugeordnet.

Die Fachstudie endet schließlich mit der Fertigstellung der Anwendung und der schriftlichen Ausarbeitung sämtlicher erfassten Erkenntnisse.

2.3. Zeitplanung

Für die zeitliche Planung wurden zu Beginn des Projekts Arbeitspakete und deren zeitlichen Rahmenbedingungen abgeschätzt.

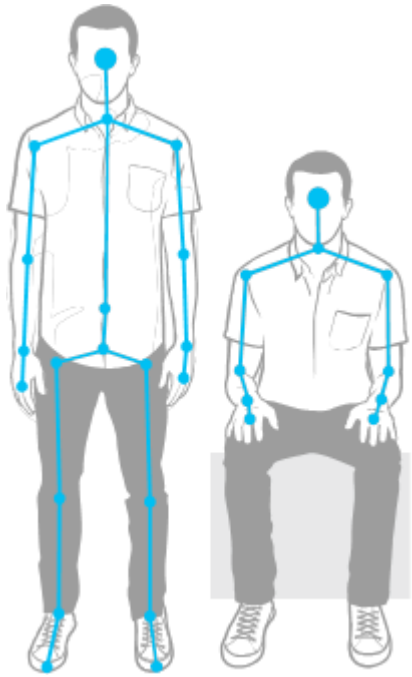
Meilenstein	Name	Beschreibung	Termin	Ziele
1	Treffen	Kurze Präsentation bisheriger Programme	28.05.13	- Fertiges Alphaprogramm
2	Bildererkennung	Gesichtswarhnehmung in Streams	20.06.13	- Wahrnehmung von Gesichtern in Streams - Speichern der Gesichtern
3	Datenbank	Datenbank mit Bildern aufbauen	30.06.13	- Datenbank mit Gesichtern von Personen
4	Gesichtserkennung	Streams aufnehmen und durch Gesichtserkennung jagen und testen	15.07.13	- Vergleich von OpenCV und KinectSDK
5	Bildvergleich	Vergleich von Datenbankbildern und Gesichtern aus Streams	01.08.13	- Vergleich von Gesichtern aus Streams und Datenbank - Erkennung von bereits vorhandenen Gesichtern
6	Gesichtserkennung 2	Gesichter von Streams erkennen	20.08.13	- Erkennung von Gesichter in Streams - Hinzufügung von noch nicht erkannten Gesichtern
7	Kategorisierung	Gesichtsdaten in einer Datenbank speichern	05.09.13	- Datenbank erstellen mit Kategorien für Gesichtsdaten - Einfügen von neu erhaltenen Daten aus den Streams
9	Ausarbeitung	Ausarbeitung anhand der Studie erstellen	05.10.13	- Fertige Ausarbeitung im PDF Format

3. Einarbeitung

3.1. Kinect SDK [MSDN]

Das Kinect SDK wird von Microsoft angeboten und richtet sich an Entwickler, die eine Kinect für Windows (im Folgendem kurz Kinect) besitzen. Diese ist von der Kinect für die Xbox zu unterscheiden und wie der Name schon sagt, für die einfache Programmierung unter Windows geeignet. Die Kinect verfügt über eine Farbkamera, einen Infrarotstrahler, einen Infrarot-Tiefensensor, mehrere Mikrofone und einen Neigungsmotor.

Die Kinect ist für die Interaktion mit einem Computer in mittelgroßen Räumen vorgesehen. Der Abstand der Benutzer sollte dabei zwischen 3,5m und 0,5m betragen. Die Personenerkennung des Kinect SDK ermittelt mit Hilfe der Tiefeinformationen die Position eines Skelettes. Zu einem Skelett gehören mehrere Skelett-Punkte, die die Software an markanten Punkten eines Menschen anbringt. Im Standardmodus werden einem Benutzer 20 Gelenkpunkte zugeordnet. Ein so genannter Seated-Mode reduziert die Anzahl der Gelenkpunkte. Es werden dann nur noch 10 Gelenkpunkte aus der Schulter-, Arm- und Kopfpartie auf dem Benutzer gesucht.



(Quelle: Tracking Modes (Seated and Default) - http://msdn.microsoft.com/en-us/library/hh973077.aspx)

Die Abstandsgrenzen können ebenfalls verändert werden. Im Near-Mode können Benutzer ab einer praktischen Entfernung von 0,4m bis zur Maximalentfernung von 3,0m erkannt werden. Im Standardmodus von 0,8m bis 4,0m.

Für das Szenario unserer Fachstudie eignet sich der Seated-Mode in Verbindung mit dem Near-Mode besonders, weil ein Benutzer wegen seiner geringen Interaktionsdistanz (zu einem public display) nicht immer vollständig im Sichtfeld der Kinect ist.

3.2. OpenCV [OPCV]

Bei OpenCV handelt es sich um eine freie Programmbibliothek welche unter den BSD-Lizenz Bedingungen steht und somit kostenlos für akademische und kommerzielle Nutzungen einsetzbar ist. Das „CV“ im Namen steht für englisch „Computer Vision“. OpenCV beinhaltet Schnittstellen für C++, C, Python sowie JAVA und unterstützt Windows, Linux, Mac OS, iOS und Android.

OpenCV wurde mit einem starken Fokus auf Echtzeit-Anwendungen konzipiert. Die Bibliothek ist im Stande, unter der Nutzung von C++ bzw. C, die Vorteile der Mehrkernprozessoren anzuwenden. Die Bibliothek verfügt über mehr als 2500 optimierte Algorithmen. Das Einsatzspektrum reicht von interaktiver Kunst bis hin zur fortschrittlichen Robotik.

3.2.1. Projekt relevante Funktionen

Gesichtserkennung:
Die Gesichtserkennung wird mit Hilfe von Beschreibungsdateien realisiert, in unserem Fall kam die von OpenCV zur Verfügung gestellte *harrcascade_frontface_default* zum Einsatz. Für diese Beschreibungsdateien wurden die notwendigen Informationen aus mehreren tausend Bildern berechnet und schließlich als .xml abgespeichert.

Der Algorithmus vergleicht mit Hilfe von Rechtecken das Eingabebild mit der Beschreibungsdatei.

Dabei wird das Eingabebild systematisch, Zeile für Zeile, von oben-links bis unten-rechts abgescannt. Diese Prozedur wird für unterschiedliche Rechteckgrößen wiederholt.

Abbildung 1: Quelle: Vorlesung Computergrafik Universität Stuttgart

Bildinformation verwenden:

OpenCV bietet auch die Möglichkeit eigene Beschreibungsdateien zu erstellen, welche in unserem Fall bei der Gesichtswiedererkennung (siehe Kapitel 6) sowie der Geschlechtserkennung (siehe Kapitel 7) zum Einsatz kamen.

3.2.2. EmguCV [EMCV]

Um die Fähigkeit der unterschiedlichen Bibliothek zu erfassen, wollten wir den Vergleiche auch in der selben Programmsprache, in unserem Fall C#, durchführen. An dieser stelle kam EmguCV zum Einsatz. Bei EmguCV handelt es sich um einen plattformübergreifenden .NET Wrapper , womit OpenCV Funktionen in .NET-kompatiblen Sprachen wie C#, VB, VC++ und IronPython aufgerufen werden können.

3.3. OpenNI [OPNI]

OpenNI steht für "*Open Natural Interaction*" . Das Framework ist ein Open Source SDK, welches von einer Gruppe von verschiedenen Entwicklern ins Leben gerufen wurde. Es wird dafür genutzt um 3D Sensor Middleware-Bibliotheken und Anwendungen zu entwickeln. Das Framework stellt hauptsächlich Schnittstellen zwischen den Geräten und der Middleware bzw. den Anwendungen zur Verfügung. Auf der OpenNI Internetseite selbst können auch bereits fertige Projekte, die mit Hilfe von OpenNI entwickelt wurden heruntergeladen werden. Hierbei erkennt man die

vielen Einsatzmöglichkeiten. Diese reichen von Scannen von Personen bis hin zur Animation von Ork-Gesichtern. OpenNI ist also eine sehr vielseitiges SDK das in Verbindung mit anderen Frameworks zum Einsatz kommt.

Da OpenNI allerdings keine eigenständige Gesichtswahrnehmung und -erkennung besitzt und die meisten, von OpenNI zur Verfügung gestellten, Projekte zur Gesichtswahrnehmung ohnehin OpenCV verwenden, haben wir uns dazu entschieden OpenNI nicht zu benutzen. Aber da OpenNI ohnehin für kompliziertere Anwendungen ausgelegt ist und es sich bei unserer um eine einfache Gesichtserkennung handelt, für die das Kinect SDK und OpenCV ausreichen, lohnt es sich nicht OpenNI zu verwenden.

4. Vergleich Vorgehensweise (Versuche)

Für den Vergleich der beiden Programme nehmen wir mit der Software Kinect Studio kurze Videosequenzen auf. Diese Videosequenzen können wir in unsere Programme einfügen und dadurch die Gesichtswahrnehmung vergleichen. Hierbei messen wir bei beiden Programmen die Anzahl der Frames, in denen Gesichtern erfasst wurden. Das Ergebnis besteht aus zwei Zahlen besteht:

1. Die Anzahl an Frames auf denen Gesichtern erfasst wurden im Verhältnis zu der insgesamten Anzahl an Frames

2. Das Verhältnis zur Anzahl der insgesamt aufgenommenen Frames in Prozent. Bei diesen Versuchen ist eine maximale Prozentzahl von 100% nicht erreichbar. Um 100% erreichen zu können, müsste auf jedem Frame ein Gesicht zu sehen sein, das im passenden Abstand zur und direkt in die Kinect schauen müsste. Dies ist aber nicht immer der Fall. Aus diesem Grund können die Prozentzahlen stark von 100% abweichen. Da aber bei beiden Programmen die selbe Videosequenz analysiert wird spielt das keine Rolle. Denn wenn die Zahlen beim Kinect SDK niedrig sind, sind sie es meistens auch bei OpenCV. Das Programm welches bei den Prozentzahlen den höheren Wert erreicht gewinnt. Allerdings sind die Zahlen allein nicht immer ausreichend, weshalb noch eine Gewichtung der Kategorien und persönliche Bewertung mit einfließt.

Die Sequenzen werden im Labor B im Simtech-Gebäude aufgenommen.

Folgende Materialien wurden für die Aufnahmen benötigt:

- Kinect: Für die Aufnahme der Videosequenzen
- Leiter: Um die Kinect auf Augenhöhe zu montieren.
- Dumbledore-Computer: Auf dem Kinect-Studio läuft.
- Klebeband: Um Abstände auf dem Boden zu markieren.
- Meterstab: Zur Abmessung.
- Papier und Stift: Für Notizen und Malen von Gesichtern.
- Taschenrechner: Um Abstände und Winkel zu berechnen.
- Smartphone: Für Bilder von Personen.

Verwendete Software:

- Kinect Studio: für die Aufnahmen. (http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx)
- Eigene Implementierung: Für Erfassung von Videosequenz mit Hilfe von Kinect Studio.

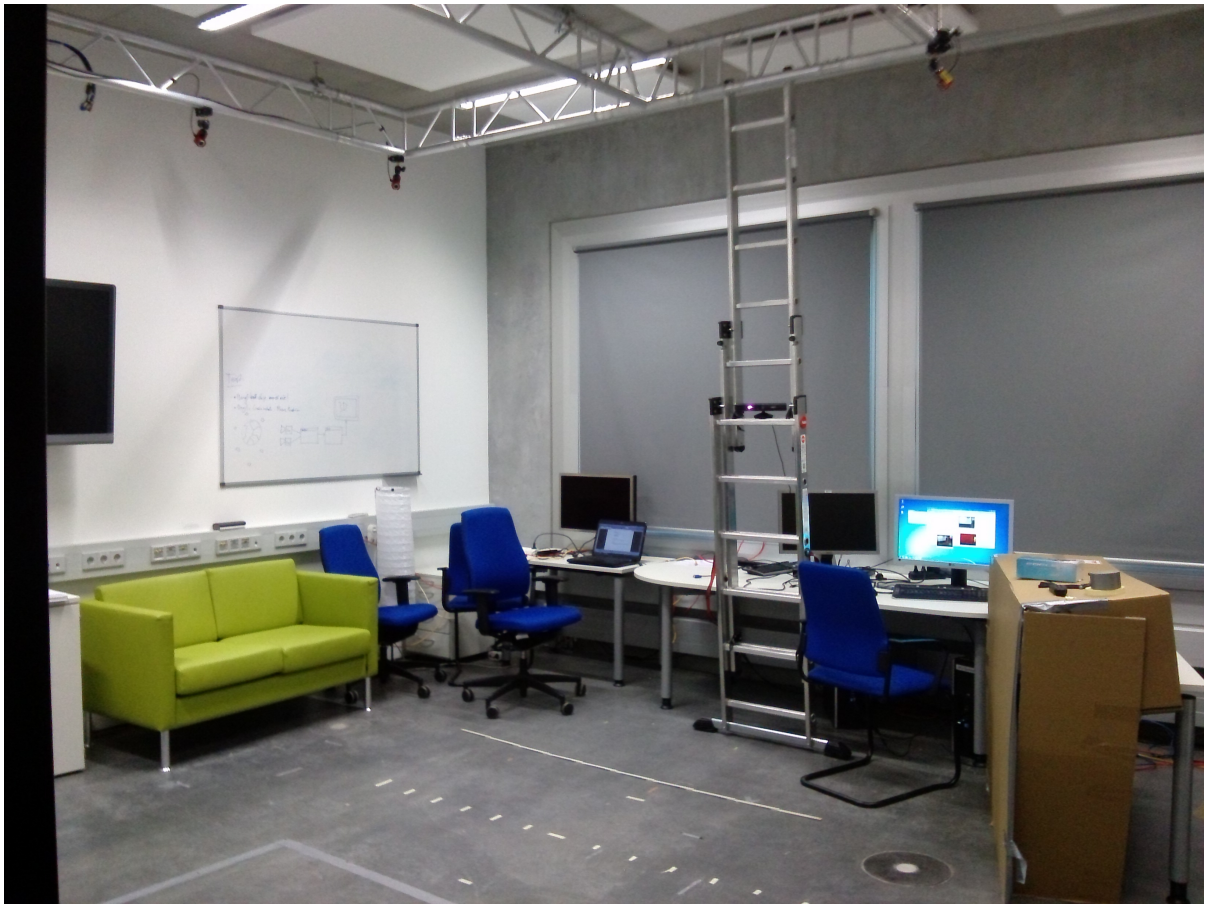


Abbildung 2: Raum für die Aufnahmen

Raum Vorbereitung:

Zunächst stellen wir die Leiter auf, damit wir die Kinect auf Augenhöhe anbringen können.

Danach werden die Markierungen auf dem Boden angebracht. Diese werden so positioniert, dass der maximale Kinectaufnahmebereich markiert ist. Die Kinect bildet mit den Markierungen ein Dreieck, bei dem die Kinect die Spitze K (Kinect) bildet. Die beiden Eckpunkte am linken und rechten Rand des maximalen Aufnahmebereichs heißen A (links) und B (rechts). Der Mittelpunkt zwischen A und B heißt M. Die Maximale Reichweite für das Labor B (Strecke KM) beträgt: 3,7m.

Der Abstand zwischen AM und BM ist jeweils ca. 2,3m lang. Daraus folgt für den maximalen Kinectaufnahmewinkel: $\alpha \approx 2 * \arctan\left(\frac{2,3}{3,7}\right) \approx 2 * 30° \approx 60°$

Der Raum wird von der Decke beleuchtet. Bei allen Aufnahmen, abgesehen von „verschiedene Beleuchtungen“, bleibt die Beleuchtung immer gleich.

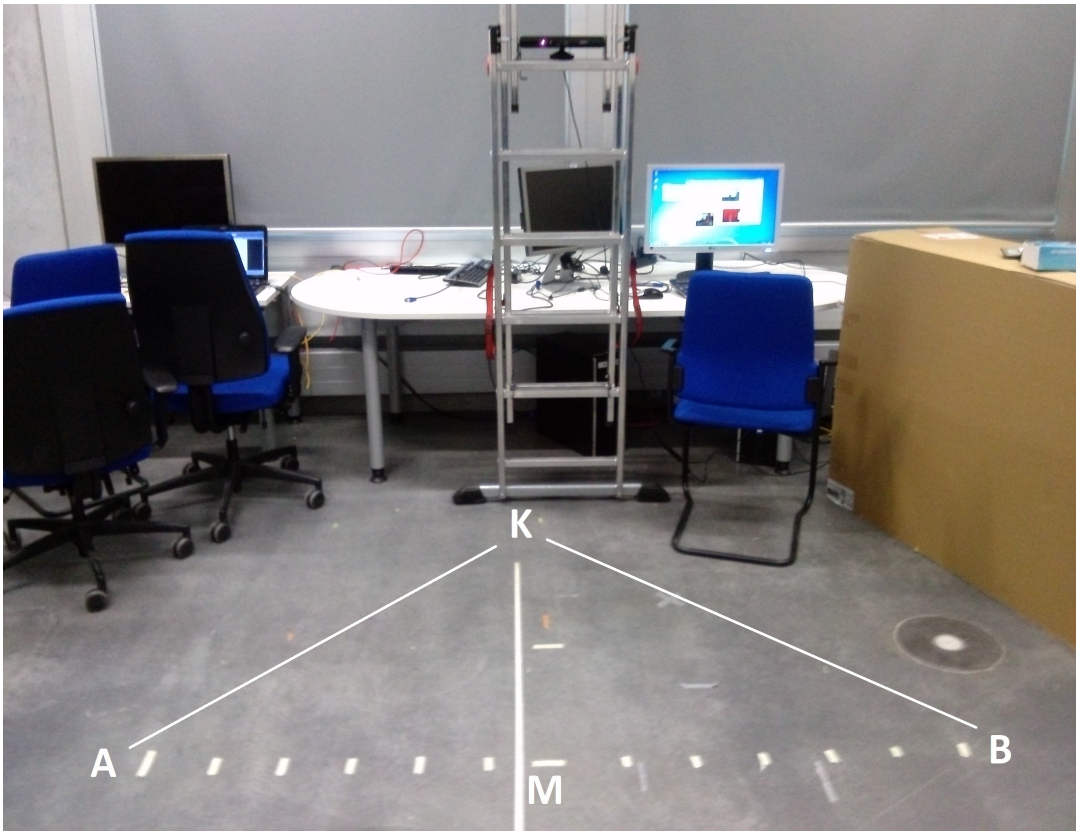


Abbildung 3: Raum mit Markierungen

Für die Aufnahmen werden beide Programme mit einer Auflösung von 640x480 konfiguriert. Des Weiteren haben wir für die Gesichtswahrnehmung des OpenCV Programms als Parameter für die „detectMultiscale“- Methode folgende Parameter verwendet:

1. das Bild (640x480)
2. Scalefactor: 1,5
3. Anzahl Nachbarn: 6
4. kleinster Rahmen: 30x30
5. Größter Rahmen: empty

Das OpenCV Programm haben wir, aus perfomancetechnischen Gründen, über die GPU der Grafikkarte laufen lassen. Dies ist nur mit einer Nvidia Grafikkarte möglich. Wir haben dafür eine Nvidia Geforce GT 750M benutzt. Wenn man das Programm über die CPU hätte laufen lassen, wären die Framewerte noch schlechter gewesen als sie ohnehin schon waren. Da die Framewerte bei OpenCV wesentlich geringer sind als bei KinectSDK werden die Gesamtzahlen nicht beachtet, sondern lediglich die Prozentzahlen.

4.1. Minimale und Maximale Reichweite

Hintergrund: Dieser Test ist dazu notwendig um die minimale und maximale Reichweite, in der die zwei APIs Gesichter wahrnehmen können, ausfindig zu machen. Ein Gesicht sollte auch dann noch wahrgenommen werden, wenn die Testperson sich in einem minimalen Abstand von 50cm von der Kinect entfernt befindet, da bei realen Public Displays die Kinect einen ähnlichen Abstand zum Gesicht der bedienenden Person aufweist. Außerdem wird der normale Use-Case im Alltag sich hauptsächlich auf das Annähern an die Kinect beschränken.

Durchführung:
Die Testperson geht in 50cm Schritten auf die Kinect hinzu und entfernt sich anschließend wieder. (Strecke KM). Gestartet wird hierbei bei M. Bei jeder 50cm Markierung wird ein paar Sekunden gestoppt, um ordentliche Aufnahmen zu gewährleisten. Der Versuch wird mit allen 3 Testpersonen durchgeführt.

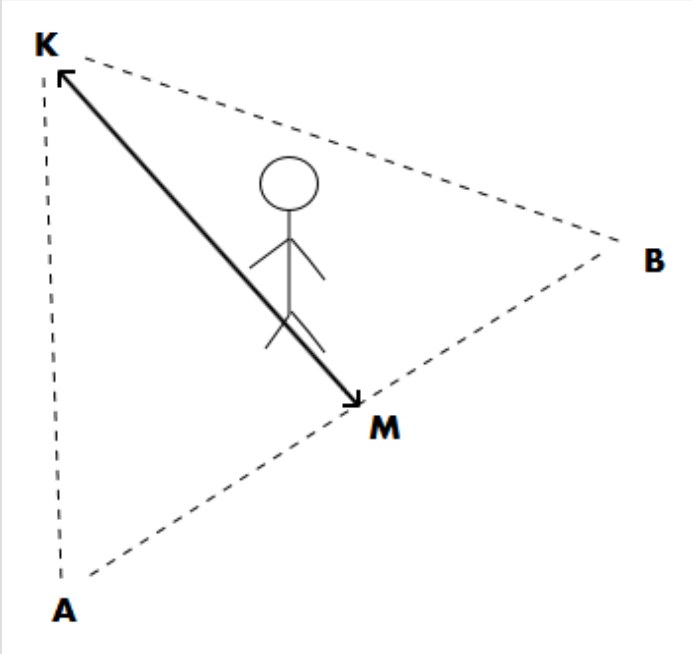


Abbildung 4: Vor- und Zurücklaufen

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1	323/542	59,50%	27/87	31,00%
Person2	335/625	53,60%	30/122	24,50%
Person3	459/779	58,90%	90/169	53,20%

Ergebnis:
Man sieht auf dem ersten Blick, dass Kinect SDK wesentlich besser abschneidet als OpenCV. Das liegt hauptsächlich an den Arbeitsweisen der beiden Algorithmen von Kinect SDK und OpenCV. Da OpenCV bei jedem Frame das Gesicht neu entdecken muss und, aufgrund des Vor- und Zurücklaufens die Größe der Rechtecke stark variiert, die Performance dadurch noch einmal stark beeinträchtigt wird, schneidet OpenCV wesentlich schlechter ab als Kinect SDK.

4.2. Erkennung von Gesichtern bei verschiedenen Winkeln

Hintergrund:
Die APIs sollten auch Gesichter, die nicht frontal zur Kinect zeigen wahrnehmen, da der Bedienende vermutlich oft auf dem Public Display hin- und herschaut. Folglich muss die Kinect Gesichter auch aus unterschiedlichen Winkeln noch erfassen können.

4.2.1. Einzelne 5° Messungen

Durchführung:
Die Testperson bewegt sich seitlich auf AM bzw. BM, wobei AM = BM = 1m. Der Abstand zur Kinect KM beträgt hierbei 1,5m. Die Testperson stellt sich zunächst auf den Punkt A/B und schaut stets geradeaus. Nun bewegt sich die Person Richtung M und zwar in einem Abstand, der im Kinect Aufnahmebereich einem Winkel von 5° entspricht.

Der Abstand eines 5° Winkels beträgt hierbei ca. 16,6666cm (1m = 30°).

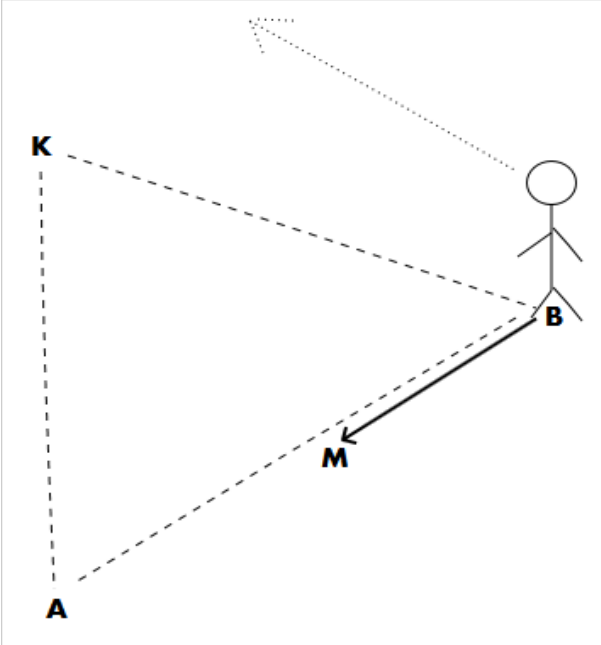


Abbildung 5: Seitlich laufen in 5° Schritten

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1	132/318	41,50%	16/62	25,80%
Person2	160/343	46,60%	36/75	48,00%
Person3	216/393	54,90%	19/78	24,30%

Ergebnis:
Bei Kinect SDK wird bei allen Testpersonen das Gesicht zwischen einem Winkel von 0 – 25° erkannt. OpenCV verhält sich hier wesentlich instabiler. Die Erkennungswinkel reichen von 25° bei einer Person bis zu 10° bei einer anderen. Außerdem wir teilweise auch die Hand der Personen als Gesicht erkannt. Da KinectSDK bei diesem Versuch wesentlich stabiler gearbeitet hat und auch von den Prozentzahlen besser abschneidet, gewinnt hier Kinect SDK.

4.2.2. Komplette durchgängige Drehung

Durchführung: Eine Testperson setzt sich auf einen drehbaren Stuhl in einem Abstand von (1m) zur Kinect. Danach wird der Stuhl um 360° gedreht. Die Testperson schaut dabei immer geradeaus.

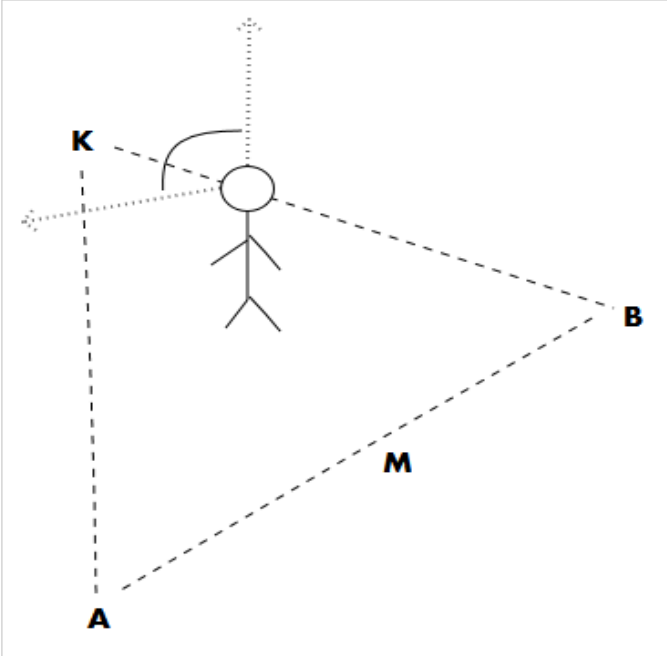


Abbildung 6: Kopfdrehung

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1	11/84	13%	2/17	11,7%

Ergebnis:
Anhand der Prozentzahlen schneiden Kinect SDK und OpenCV ungefähr gleich gut ab. Da es bei diesem Versuch keine besonderen Auffälligkeiten gab können wir hier aufgrund der höheren Framezahlen Kinect SDK bevorzugen.

4.3. Vorbeilaufen an der Kinect

Hintergrund:
Dieser Versuch dient dazu herauszufinden, in welcher Geschwindigkeit die APIs ein Gesicht erkennen. Das kann von Vorteil sein, wenn eine Person an dem Public

Display vorbei läuft und flüchtig in die Kinect schaut, worauf der Inahlt des Displays reagieren und der Passant mit Inhalten gelockt werden kann.

Durchführung: Abstand KM = 1,5m, AB = 2m. Die Testpersonen laufen von A nach B in unterschiedlichen Geschwindigkeiten und schauen dabei immer in die Kinect.

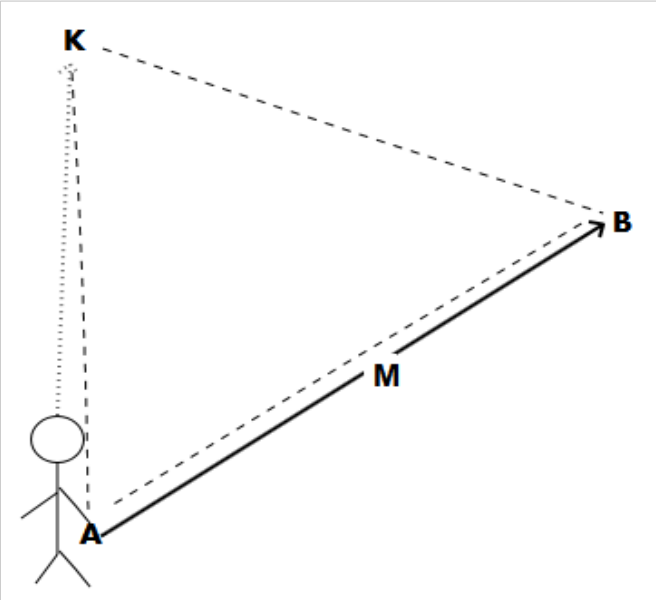


Abbildung 7: Vorbeilaufen an der Kinect

Messdaten:

Probanten	Zeiten	KinectSDK		OpenCV	
		Frames	Prozent	Frames	Prozent
Person1					
langsam	3,1s	9/92	9,7%	2/21	9,5%
mittel	1,5s	0/59	0%	0/18	0%
schnell	1,2s	0/50	0%	2/20	10%
Person2					
langsam	2,6s	8/80	10%	1/11	9%
mittel	2,2s	0/73	0%	1/17	5,8%
schnell	1,0s	0/41	0%	1/10	10%

Ergebnis:
Man sieht hier anhand der Zahlen, dass OpenCV eindeutig besser abschneidet. Das liegt vor allem an den Wahrnehmungsalgorithmen. Da sich die Personen für das Skelett zu schnell bewegt haben, konnte KinectSDK keine guten Werte erzielen. OpenCV sucht auf jedem Frame ein Gesicht, was in diesem Fall besser funktioniert. Da OpenCV aber stark von dem Frame abhängt den das Programm gerade verarbeitet handelt es sich hierbei aber um eine Glückssache. Bei Person1 mit mittelschnellem Tempo sieht man zum Beispiel, dass 0 Frames gemessen wurden. Daher ist OpenCV mehr oder weniger vom Glück der Framewahl abhängig, was aber auch nicht zu zuverlässigen Ergebnissen führt.

4.4. Mehrere Probanden

Hintergrund:
Dieser Versuch soll testen, ob auch mehrere Personen von den APIs erfasst werden können. Es ist nämlich möglich, dass mehrere Menschen gleichzeitig vor dem Display stehen und anderen Personen zuschauen bzw. mit ihnen zusammen das Display bedienen.

4.4.1. Personen stillstehend

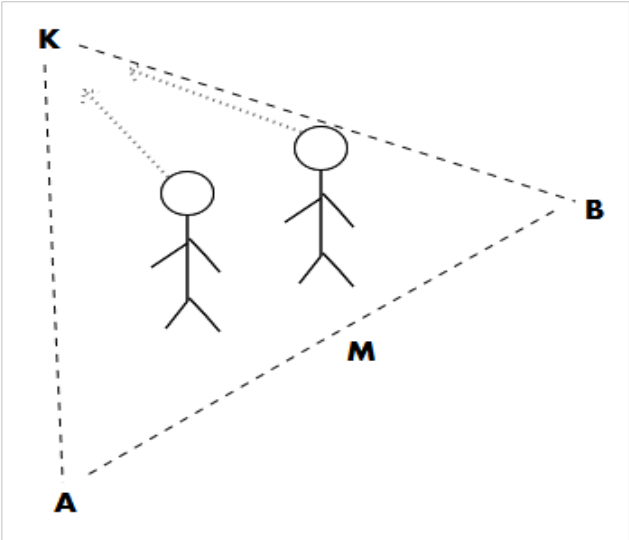


Abbildung 8: Mehrere Probanten

Durchführung:
Drei Personen stellen sich gleichzeitig vor die Kinect.

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1/2/3	73/88	82,90%	14/38	36,80%

Ergebnis:
Das Kinect SDK weist hier wesentlich höhere Werte auf. Allerdings liegt das auch daran, dass immer nur eine Person erkannt wird. OpenCV erkennt alle drei Personen. Demnach gewinnt hier OpenCV trotz schlechterer Prozent- und Framezahlen.

4.4.2. Personen bewegend

Durchführung:
Drei Personen laufen wild durcheinander vor der Kinect herum.

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1/2/3	222/286	77,60%	52/65	80,00%

Ergebnis:
Bei diesem Versuch dominiert OpenCV. Nicht nur höhere Prozentzahlen, sondern auch mehrere Personen werden erkannt. Allerdings wurden hier nur 2 von maximal drei Personen gleichzeitig erkannt. Trotzdem gewinnt bei diesem Versuch ebenfalls OpenCV.

4.5. Fakegesichter

Hintergrund:
Hier soll getestet werden, ob auch falsche Gesichter, wie zum Beispiel Bilder mit Gesichtern von den APIs erfasst werden. Das sollte im Normalfall nicht passieren, da

sonst eventuell statt dem Gesicht des Bedieners, ein Gesicht, das sich eventuell auf dem T-Shirt befindet erfasst und gespeichert wird. Falls danach eine Person mit dem selben T-Shirt den Display bedient wird unter Umständen dieselbe Person auf dem T-Shirt statt der Person, welche das T-Shirt trägt, erkannt.

4.5.1. Gezeichneter Smiley

Durchführung:
Smiley malen und das Bild vor die Kinect halten.

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Bild	0/122	0,00%	0/24	0,00%

Ergebnis:
Bei diesem Versuch haben beide Programme gleich abgeschnitten. Der Smileywurde bei beiden nicht erkannt, wie es auch sein sollte.

4.5.2. Gemaltes Gesicht mit Körper

Durchführung:
Gemaltes Gesicht vor das Gesicht einer Person halten und die Person vor die Kinect stellen.

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1 mit Bild	22/165	13,30%	0/33	0,00%

Ergebnis:
Kinect SDK hat, wenn auch nur kurz, ein Gesicht erkannt, da das Skelett zwar vorhanden aber das Gesicht eine Attrappe war. OpenCV hat hierbei nichts erkannt. Folglich gewinnt hier OpenCV.

4.5.3. Bilder von Personen

Durchführung:
Portraitbilder von Personen mit Smartphone aufnehmen und die Bilder vor die Kinect halten.

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Bilder	0/87	0,00%	8/20	40,00%

Ergebnis:
Kinect SDK hat aufgrund eines fehlendes Skeletts nichts erkannt. OpenCV jedoch hat die Gesichter erkannt. Das sollte allerdings nicht passieren. Gerade wenn zum Beispiel gegenüber des Public Displays ein Plakat mit einem Gesicht aufgehängt wird, wird die ganze Zeit dieses Gesicht erkannt. Außerdem könnte man, wenn man das nötige Hintergrundwissen über das System verfügt, Bilder von anderen Personen machen und vor den Display halten, um so zu sehen, wofür sich diese Person interessiert. Deshalb ist der Gewinner hier eindeutig Kinect SDK.

4.6. verschiedene Beleuchtungen

Hintergrund:
Da die Public Displays, wie der Name schon sagt, public, also öffentlich sind, befinden diese sich teilweise auch an der frischen Luft. Sie sind also der Witterung und der Sonne ausgesetzt. Da sich die Sonne leider nicht immer an der selben Stelle befindet, ist es wichtig verschiedene Beleuchtungswinkel zu überprüfen.

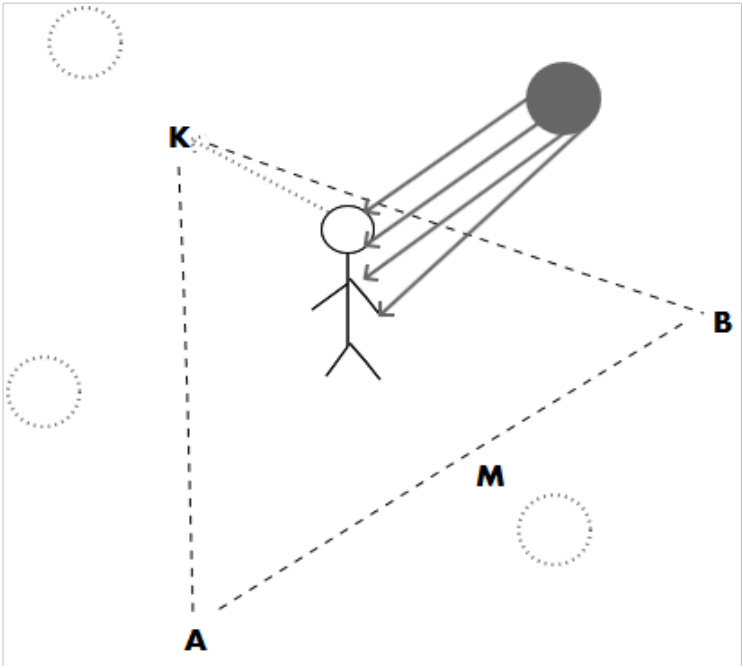


Abbildung 9: Beleuchtung in 45°

Durchführung:
Testperson stellt sich im Abstand von ca. 1,5m vor die Kinect im abgedunkelten Raum. Das Gesicht wird dann für ein paar Sekunden in 45° Winkeln beleuchtet. (0°, 45°, 90°, 135°, 180°).

Messdaten:

Probanten	KinectSDK		OpenCV	
	Frames	Prozent	Frames	Prozent
Person1	199/239	83,20%	42/47	89,30%

Ergebnis:
Bei diesem Versuch schneidet OpenCV gering besser ab. Beim Kinect SDK gab es außerdem geringe Verschiebungen der Skelettpunkte beim Lichtwechsel. Hier gewinnt OpenCV.

5. Ergebnis Vergleich

Der Gewinner unsere Versuche ist Kinect SDK. OpenCV hat zwar mehr, Kinect SDK aber die wichtigeren Kategorien gewonnen. Die Hauptvergleiche: Vor- und Zurücklaufen und Kopfdrehen, konnte Kinect SDK für sich verbuchen. Die Bildattrappen von Gesichtern werden durch die Skelettmechanik nicht erkannt, was ein enormer Vorteil für die Public Displays sein kann. Der größte Nachteil des Kinect SDK ist, dass mehrere Personen nicht erkannt werden. Aber von diesem Punkt abgesehen ist Kinect SDK wesentlich besser geeignet als OpenCV.

OpenCV hat den Nachteil, dass es bei Streams generell dem Kinect SDK in Hinsicht auf Perfomance unterlegen ist.

Ein weiterer Schwachpunkt von OpenCV ist die niedrige Framezahl auf die wir in unserem Vergleich nicht eingegangen sind, da OpenCV allein schon aus diesem Grund nicht geeignet wäre. Es gibt zwar verschiedene Möglichkeiten die Framezahl von OpenCV wesentlich zu erhöhen, allerdings würde das den Vergleich verfälschen. Man könnte zum Beispiel die Auflösung herunterschrauben. Dadurch würden aber keine guten Vergleichsmöglichkeiten zwischen OpenCV und Kinect SDK mehr herrschen. Eine weitere Möglichkeit bestünde darin die Parameter der Methode „DetectMultiScale“ je nach Versuch anzupassen. Dies würde ebenfalls die Framezahl stark erhöhen, allerdings verfälscht das wieder die Ergebnisse, da im Endprogramm die Parameter nicht geändert werden können.

Man kann also zu beiden API's folgendes über die Vor- und Nachteile sagen:

Kinect SDK: Ist für Streams und Videos ausgelegt und aufgrund der Skelettmechanik gut um einzelne Personen zu verfolgen. Außerdem wird dadurch verhindert, dass Gesichtsattrappen erkannt werden, was, je nach Situation auch ein Nachteil sein kann. Wenn zum Beispiel die Anwendung darauf ausgelegt ist, Personen auf Bildern zu erfassen. In unserem Fall ist es jedoch positiv zu betrachten. Des Weiteren ist Kinect SDK präziser bei der Gesichtswahrnehmung, da, wenn das Skelett einmal registriert wurde, die Position gespeichert wird, im Gegensatz zu OpenCV. Aufgrund der oben genannten Punkte läuft Kinect SDK wesentlich schneller als OpenCV.

Allerdings gibt es auch viele Nachteile für Kinect SDK. Diese wurden in unserem Vergleich nicht mit einbezogen, weil sie sich hauptsächlich auf die Wiedererkennung beziehen. Das Kinect SDK benötigt ein Skelett und ist dadurch nicht in der Lage Gesichter auf Bildern zu erkennen, wie bereits angesprochen. Allerdings kann man deshalb auch keine erkannten Gesichtern mit Bildern aus der Datenbank vergleichen, was eine Wiedererkennung unmöglich macht. Das spielt allerdings auch keine große Rolle, da das Kinect SDK von sich aus sowieso keine Funktionen zur Gesichtswiedererkennung bereitstellt.

OpenCV: Aus technischen Gründen eignet sich OpenCV besser für die Gesichtserkennung auf Bildern. Hierbei kommt die Performance nicht so sehr zum tragen. Allerdings funktioniert die Erkennung auf Bildern dafür besser, was bei Kinect SDK nicht einmal unterstützt wird. Ein weiterer Vorteil ist, dass OpenCV Gesichtswiedererkennung unterstützt, was für die nachfolgende Arbeit unerlässlich ist. Zusätzlich zur Wiedererkennung gibt es hier auch Möglichkeiten verschiedene andere Persondendaten zur Kategorisierung zu erfassen die mit Hilfe der Trainierbarkeit realisiert werden. Ein weiterer Vorteil von OpenCV ist die bereits angesprochene Konfigurierbarkeit mit der man situationsabhängig das Programm konfigurieren kann. Auf diese Weise kann man das Programm auf bestimmte Situationen anpassen, in denen es gute Ergebnisse erzielen kann.

Die bereits angesprochene Konfigurierbarkeit kann zwar für einzelne Situation sehr von Vorteil sein, da unser Programm aber einen universelleren Einsatz als Ziel hat, wird daraus ein Nachteil der sehr an der Performance zehrt.

Aus bereits genannten Gründen eignet sich Kinect SDK wesentlich besser zur Gesichtswahrnehmung. Da es allerdings keine Unterstützung seitens Gesichtserkennung anbietet können wir uns nicht komplett auf Kinect SDK verlassen. Da die bereits genannten Schwächen zur Wiederkennung von Kinect SDK gleichzeitig die Stärken von OpenCV sind haben wir beschlossen das Kinect SDK zur Personenerkennung und OpenCV zur Wiedererkennung zu verwenden.

6. Gesichtswidererkennung

Die Aufgaben der Wiedererkennungssoftware sind:

- Bekannte Gesichter verwalten
- Neue Gesichter hinzufügen
- Gesichter wiedererkennen

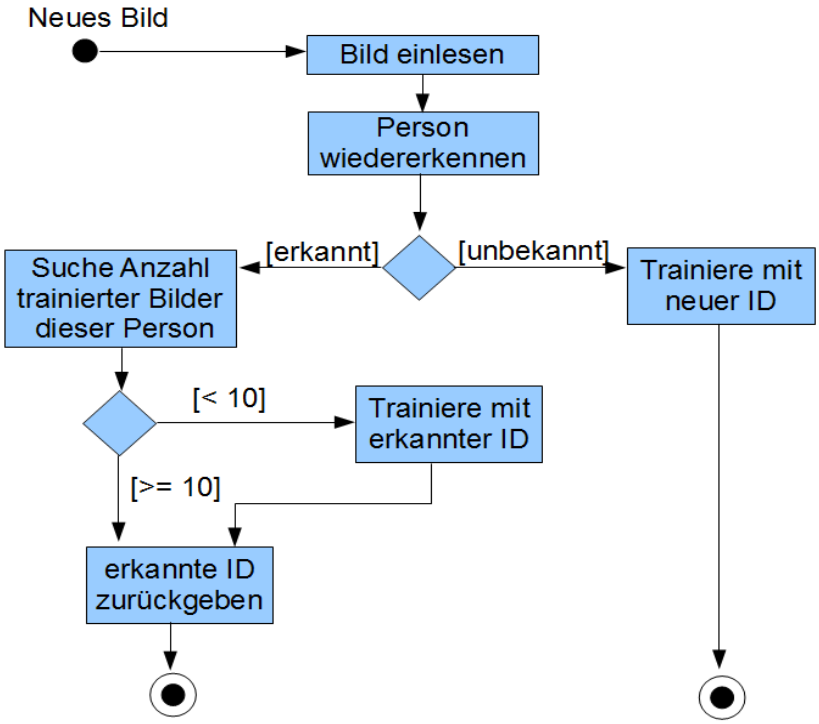
Darüber hinaus soll sie später über weitere Features erweitert werden können. (siehe Kapitel 7: Geschlechtserkennung)

6.1. Bilderverwaltung

Die Bilderverwaltung wird über eine Liste realisiert, die den vollständigen Pfad der Bilddatei und die zugehörige ID dieser Person hält. Bei einer Änderung der Liste, wird diese sofort XML-serialisiert und abgespeichert. Ebenso wird sie beim Programmstart wieder eingelesen.

6.2. Hinzufügen neuer Bilder

Beim Hinzufügen neuer Bilder wird zunächst überprüft ob das Gesicht auf dem neuen Bild bereits bekannt ist. Als Ergebnis liefert der Fisherface-Algorithmus die ID der Klasse, welcher das Gesicht zugeordnet werden konnte. Bei Gesichtern die keiner der trainierten Klassen zugeordnet werden konnten, gibt der Algorithmus -1 als Ergebnis zurück. In diesem Fall wird das neue Gesicht in eine neue Klasse (mit neuer ID) abgelegt und trainiert. Bereits bekannte Gesichter werden nur dann trainiert, wenn noch nicht genügend Bilder trainiert wurden.



Aus Gründen der Skalierbarkeit werden zu einer Klasse maximal 10 Gesichter trainiert.

6.3. Gesichter wiedererkennen

Bilder müssen für die Wiedererkennung mit OpenCV alle die selbe Größe haben. Aus diesem Grund speichern wir alle aufgenommenen Bilder im 100 px * 100 px Format auf. Da die Größe des Gesichtes allerdings abhängig von dem Abstand unterschiedlich groß ist, müssen wir dir Bilder verkleinern, bzw. vergrößern, sodass sie in das Format passen.

Der Fisherface-Algorithmus ordnet ein Bild in eine der trainierten Klassen ein. Dabei wählt er diejenige Klasse als Ergebnis, die die geringste Distanz zum Testobjekt hat. Dabei gibt es drei mögliche Fehler:

1. Ein bekanntes Gesicht wird der falschen Klasse zugeordnet
2. Ein bekanntes Gesicht wird als unbekannt eingestuft („false negative“)
3. Ein unbekanntes Gesicht wird einer Klasse zugeordnet („false positive“)

Der Konstruktor des Recognizer-Klasse erhält 2 Parameter, die die Fehlerquote maßgeblich beeinflussen.

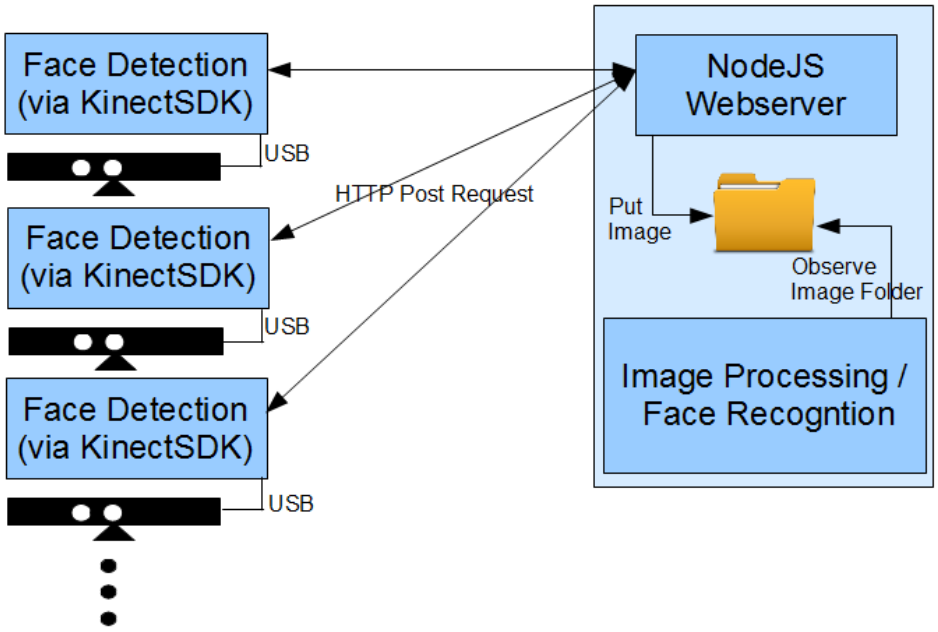
```
public FisherFaceRecognizer(
        int numComponents,
        double threshold
)
```

Der zweite Parameter gibt die Schwelle an, ab welcher Distanz ein zu testendes Gesicht als unbekannt eingestuft wird. Ist die Schwelle

- zu hoch eingestellt, wird Fehler Nummer 3 öfter auftreten
- zu niedrig eingestellt, wird Fehler Nummer 2 öfter eintreten

6.4. Architektur zur verteilten Wiedererkennung

Das Szenario erfordert die verteilte Erfassung von Bildern, aber eine zentralisierte Verwaltung, Kategorisierung und Untersuchung. Hier wird auf das Client-Server Muster zurückgegriffen. Jeder Klient ist dabei mit einer Microsoft Kinect und einer Netzwerkverbindung ausgerüstet. Der Server stellt einen leichtgewichtigen Webserver zu Verfügung, der die empfangenen Bilder an die Wiedererkennungssoftware weitergibt.



6.5. Implementierung

6.5.1. Client Application

Abhängigkeiten
Die Anwendung verwendet die folgenden Assebmlys:

Microsoft.Kinect.dll
Microsoft.Kinect.Toolkit.dll
Microsoft.Kinect.Toolkit.FaceTracking.dll

Als erstes soll eine Verbindung zur Kinect aufgebaut werden. Dazu wird ein EventHandler für das Event KinectSensorChooser.KinectChanged implementiert. An dieser Stelle werden nun die Einstellungen für die Kinect gesetzt. Dazu zählen Auflösung (sowohl für Farb- als für Infrarotbilder), Near mode und Seated mode.

Ein weiterer EventHandler wird ausgeführt sobald die Kinect neue Frames bereithält. Hier erhält der Entwickler bereits ein Feld mit Skelettobjekten, die die Kinect erkannt hat. Für jedes erkannte Skelett wird mit Hilfe des FaceTracker Objektes versucht ein Gesicht zu finden. Jedoch kann die Anwendung nicht ohne Weiteres entscheiden, ob ein erkanntes Gesicht zu einer bereits bekannten Person gehört.

Über die Oberfläche kann der Benutzer den Bild-Ausgabepfad festlegen. Dieser wird in den Benutzereinstellungen gespeichert. Eine konfigurierbare natürliche Zahl gibt den Abstand als Anzahl von Frames an, in dem Gesichter gespeichert und an den Server gesendet werden sollen. Darüber hinaus vergibt die Anwendung jedem Gesicht eine ID-Nummer. Sollte über einen konfigurierbaren Zeitraum kein Gesicht erkannt worden sein, wird beim nächsten Mal eine um eins erhöhte ID vergeben. Der Dateiname beinhaltet dabei alle für den Server interessanten Informationen:

237-165753678.jpg

x-y.jpg, wobei x die automatisch inkrementierte ID des erkannten Gesichts ist. Y ist eine Zufallszahl, die dafür sorgt, dass ein Dateiname einmalig ist.

Die gespeicherten Bilder werden in der Methode ServerConnection.TrainImage an den Webserver gesendet:

```
webClient.UploadFileAsync(
new Uri(Properties.Settings.Default.ServerUploadURL), "POST", FilePath);
```

Die Server URL wird dabei in den Anwendungseinstellungen (KinectApp.exe.config) konfiguriert.

6.5.2. NodeJS Webserver

Der NodeJS Webserver lauscht auf Port 3000 und registriert sich auf zwei URLs:

app.post('/train', train.start);
app.post('/upload', upload.begin);

Sollte ein Post-Request an eine der beiden URLs gesendet werden, nimmt die zugehörige Funktion train.start oder upload.begin diesen entgegen. Das darin enthaltene Bild wird dann entweder in .\uploads oder in .\train abgelegt.

Es gibt zwei getrennte Ordner, damit der Recognition-Server zwischen Bildern unterscheiden kann, die er trainieren soll und denen, die er nur prüfen soll. Diese Unterscheidung wurde im Laufe des Projekts hinfällig. Neuester Stand ist, dass jedes Bild zuerst geprüft wird und dann entschieden wird, ob neu trainiert wird oder nicht. Dazu wird jedes Bild an die /train-Route gesendet.

6.5.3. Server Application

Zunächst werden in einem neuen Thread zwei FileSystemWatcher Objekte konfiguriert. Diese überwachen zwei konfigurierbare Ordnerpfade. Sobald eine Datei mit .jpg-Endung erstellt wird, wird ein EventHandler ausgeführt. Dieser startet wiederum einen neuen Thread, der sich um die Behandlung des Bildes kümmert. Dazu gehört die Wiedererkennung und das Training des FisherFaceRecognizers.

Beim Start der Anwendung wird versucht im Windows-Anwendungsdatenverzeichnis die Datei \RecognizerServer\TrainingData.xml zu lesen. In ihr werden die Pfade zu allen trainierten Bildern verwaltet. Zu jedem Bild gehört außerdem ein Label, das anzeigt zu welcher Klasse das jeweilige Bild gehört. Im Quellcode ist dies die Liste:

```
List<ImageSet> ImageStorage.TrainingData
```

Diese wird bei Änderungen XML-serialisiert und abgespeichert. Ein Auszug:

```
<ImageSet>
<Filename>C:\Uni\Fachstudie\FS_Server\train\269-1667962847.jpg</Filename>
<Label>269</Label>
</ImageSet>
<ImageSet>
<Filename>C:\Uni\Fachstudie\FS_Server\train\269-845899570.jpg</Filename>
<Label>269</Label>
</ImageSet>
<ImageSet>
<Filename>C:\Uni\Fachstudie\FS_Server\train\271-722060412.jpg</Filename>
<Label>271</Label>
</ImageSet>
```

Außerdem wird zur Geschlechtserkennung eine weitere XML-Datei eingelesen. (Siehe Kapitel 7)

Das Trainieren des Fisherface-Algorithmus geschieht über die Methode

```
FisherFaceRecognizer.Train(...)
```

Die Einordnung eines Bildes in eine Klasse geschieht über die Methode

```
FisherFaceRecognizer.Predict(...)
```

6.6. Codemetriken

Hierarchie	Wartbarkeitsindex	Zyklomatische Komplexität	Vererbungstiefe	Klassenkopplung	Codezeilen
KinectOpenCVApp (D	81	47	9	68	140
KinectSDKApp (Debu	77	49	9	74	151
RecognizerServer (De	73	43	1	30	146
RecordApp (Debug)	79	35	9	59	118

Die in C# implementierten Systemteile bestehen aus nur wenigen Zeilen Code und haben eine gute Wartbarkeit. Das zeigt, dass die Implementierung durch das Kinect Toolkit und OpenCV stark vereinfacht wurden.

6.7. Skalierbarkeit

Eine Bilddatei in Graustufen mit einer Auflösung von 100 px * 100 px hat durchschnittlich die Größe von 4 kB. Für 1.000.000 Bilder benötigt man also etwa 4 GB Speicherplatz.

Durch die verteilte Architektur des Systems, skaliert die Erfassung der Bilder ebenfalls sehr gut. Es stellt kein Problem dar ca. 5 Bilder pro Sekunde, pro Maschine zu erfassen.

Der Teil, der am meisten Zeit beansprucht ist das Trainieren der Wiedererkennung. Bei der Geschlechtserkennung spielt dies allerdings keine Rolle, weil lediglich zum Programmstart eine feste Anzahl an Bildern trainiert wird (konstante Komplexität). Bei fortschreitender Laufzeit erhöht sich jedoch die Anzahl der zu trainierenden Gesichter für die Wiedererkennung. Hier gibt es unterschiedliche Verbesserungsmöglichkeiten:

- Es werden immer nur die neuesten n Bilder trainiert. (konstante Komplexität)
- Es werden mehrere FisherFaceRecognizer-Objekte erstellt, die sich die „Trainingslast“ aufteilen. (Load Balancing) Diese können nebenläufig trainiert werden.
- Das Training wird auf mehrere Maschinen gleichzeitig aufgeteilt. (Hoher Entwicklungsaufwand nötig)

7. Geschlechtserkennung

Ähnlich wie die Wiedererkennung funktioniert die Geschlechtserkennung. Dabei werden zwei Klassen von Gesichtern trainiert.

Ein Ausschnitt aus der XML-Datei TrainingData.xml:

```
<?xml version="1.0"?>
<ArrayOfImageSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<ImageSet>
<Filename>.\Training\Frauen\s35\9.jpg</Filename>
<Label>0</Label>
</ImageSet>
<ImageSet>
<Filename>.\Training\Frauen\s35\10.jpg</Filename>
<Label>0</Label>
</ImageSet>
<ImageSet>
<Filename>.\Training\Männer\s1\1.jpg</Filename>
<Label>1</Label>
</ImageSet>
<ImageSet>
<Filename>.\Training\Männer\s1\2.jpg</Filename>
<Label>1</Label>
</ImageSet>
</ArrayOfImageSet>
```

Bilder mit Frauengesichtern werden mit dem Label 0 trainiert und Männergesichter mit dem Label 1. Ein zu testendes Bild wird einer der beiden Klassen zugeordnet. Um zu verhindern, dass zu testende Gesichter als unbekannt eingestuft werden, wird die Schwelle auf einen möglichst hohen Wert eingestellt:

```
genderDetector = new FisherFaceRecognizer(90, Int32.MaxValue);
```

Die Aussagekraft der Geschlechtserkennung erhöht sich dabei mit der Anzahl der trainierten Bilder. Beim Start der Server Software werden eine feste Anzahl an Bildern eingelesen und trainiert. (40 Männergesichter und 40 Frauengesichter) Im Hinblick auf die Skalierbarkeit ist es kein Problem, mehr Gesichter zu trainieren, da dies nur einmal zum Start der Anwendung ausgeführt wird.

8. Erweiterbarkeit

8.1. Gesichtsmerkmale

Wie schon im Kapitel "Geschlechtserkennung" erwähnt, basiert die Klassifizierung auf Gesichtsmerkmale und den zugehörigen Attributen. So ist es möglich durch das Erweitern der Gesichtsdatenbank weitere Gruppierungen zu erstellen, wie Beispielsweise Altersgruppen oder ethnische Gruppen.

8.2. Architektur

Durch die verteilten Komponenten des Systems lassen sich beliebig viele Systeme mit einer Kinect anbinden.

Zur Verbesserung der Performance gibt es außerdem mehrere Möglichkeiten die Serversoftware zu erweitern. (Siehe Kapitel 5.7 Skalierbarkeit)

9. Weiterführende Arbeiten

Für weiterführende Arbeiten würde sich ein Vergleich der Wiedererkennungsalgorithmen anbieten. In OpenCV sind vier Algorithmen implementiert:

- FisherFaces
- EigenFaces
- Local Binary Patterns Histogram
- Principle Component Analysis

Interessante Fragestellungen dazu wären:

- Wie viele Bilder sollten sinnvollerweise zu einer Person trainiert werden?
- Wie verhält sich die Performance?
- Welcher Algorithmus hat die meisten korrekten Treffer?

Anhand der Ergebnisse könnte das vorhandene System angepasst werden. Außerdem wären Verbesserungen in Hinsicht auf die Erfassung von mehreren Personen zum selben Zeitpunkt denkbar.

10. Zusammenfassung

Abschließend möchten wir das gesamte Projekt noch einmal Revue passieren lassen.

Für den Vergleich haben wir die APIs Kinect SDK und OpenCV verwendet. Hierbei ging es darum herauszufinden, welche der beiden APIs sich besser dafür eignet, Gesichter von Personen wahrzunehmen. Um das herauszufinden, haben wir verschiedene Versuche durchgeführt. Dabei haben wir versucht, die APIs auf Alltagssituationen zu testen, wie zum Beispiel das Zulaufen zur Kinect, sowie Bewegungen des Kopfes und das Anzeigen von Bilder mit Gesichtern von Personen.

Kinect SDK hat bei diesen Versuchen wesentlich besser abgeschnitten, weshalb wir uns dazu entschieden haben, es für die Gesichtswahrnehmung zu verwenden. OpenCV hatte allgemein eine schlechtere Bildrate als Kinect SDK. Außerdem gab es bei OpenCV auch einige Probleme bei der Kopfdrehung und dem Erkennen der Bilder von Personen. Da Kinect SDK allerdings keine eigenständige Gesichtserkennung unterstützt, haben wir uns dazu entschlossen, OpenCV für die Wiedererkennung zu verwenden. Als abschließendes Statement kann man sagen, dass beide APIs gewisse Vor- und Nachteile bieten, weshalb wir uns auch dazu entschieden haben, beide in Verbindung zu verwenden.

Quellverzeichnis

[EMCV] EmguCV; http://www.emgu.com/wiki/index.php/Main_Page

[MSDN] Miscrosoft MSDN; http://msdn.microsoft.com/en-us/library/jj131033.aspx

[OPCV] Intel, Willow Garage; http://opencv.org/

[OPNI] PrimeSense , Willow Garage, ASUS; http://www.openni.org/

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 15. Oktober 2013 _____

_____

_____