INDEPENDENT COURSEWORK

Independent Production

Fingergestenerkennung
mit Microsoft Kinect für Windows

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Verfasser: Samuel Stein, s0534332
Internationale Medieninformatik Master
SS 2013

Betreuer: Prof. Dr. Carsten Busch

Inhalt

Abstract

This project tries to find out how the Microsoft Kinect for Windows could help to recognize and translate Sign language from the depth image of the sensor to ordinary speech or text. The independent coursework is separated into two parts: evaluation and prototypal implementation of software who transcribes sign language symbols which are recorded before.

Keywords: Kinect, 3D hand- and finger detection, gesture recognition, sign language, nui, real-time, c#

Quellcode und Binärdateien sind unter
https://github.com/samuelstein/FingerSpellingKinect zu finden.

Zitiert wird im Nachfolgenden Text nach IEEE 2006.

Einleitung

„*After the Command Line Interface (CLI) and the Graphical User Interface (GUI) the Natural User Interface (NUI) is the next big thing in human-machine interaction."*

Stefan Stegmueller, Creator of Candescent NUI Library [1]

Die technische Evolution schreitet voran und bietet neue Möglichkeiten der Interaktion und Kommunikation. Oftmals gehen damit auch Arbeitserleichterungen einher. Das anfangs genannte Zitat von Stefan Stegmüller spiegelt dabei treffend die Entwicklungen der letzten 40 Jahre in der IT-Branche wieder. Dabei ersetzt die Einführung einer neuen Technik bzw. Schnittstelle nicht zwangsläufig die vorangegangenen. Häufig etabliert sie sich in einem bestimmten Anwendungsbereich. Als prominentes Beispiel für ältere und nicht aus der Mode gekommene User Interface-Technologien sei die Maus oder das unverzichtbare Terminal für UNIX-Systemadministratoren genannt.

Die Motivation für diese Arbeit resultierte sowohl aus der Idee die Grenzen der Kinect, als ein Vertreter der Natural User Interfaces, auszuloten als auch aus einem interessanten Gespräch mit einer Freundin, die Dolmetscherin für Gebärdensprache ist. Des Weiteren fasziniert mich der Ansatz Technik für die Überwindung von Barrieren der Mensch-Maschine-Schnittstelle einzusetzen. Beispielsweise können körperlich beeinträchtigte Menschen mit den richtigen technischen Hilfsmitteln den Computer benutzen und somit mit anderen Menschen kommunizieren.

Die vorliegende Arbeit konzentriert sich wegen der technischen Möglichkeiten und Limitation der Kinect nicht auf die Erkennung vollständiger Gebärden sondern des deutschen Einhand-Fingeralphabets (siehe Anhang).

Gebärdensprache

Die Gebärdensprache besteht aus kombinierten Zeichen (Gebärden), die vor allem mit den Händen, in Verbindung mit Mimik und Mundbild (lautlos gesprochene Wörter oder Silben) und zudem im Kontext mit der Körperhaltung gebildet werden [2].

Die Gebärdensprache variiert von Land zu Land. In Deutschland wird sie offiziell Deutsche Gebärdensprache (DGS) bezeichnet. Am meisten verbreitet ist die American Sign Language (ASL). Sie ist eine eigenständige, visuelle, natürliche (Minderheiten-) Sprache.

Geschichte

Die Gebärdensprache blickt auf eine lange und bewegte Geschichte zurück. Die Anfänge gehen bereits auf die Prärie-Indianer (Plains-Indianer) zurück, die den Norden Amerikas und Teile Kanadas besiedelten. Diese sogenannte „Plains Indian Sign Language (PISL)“ entwickelte sich lange bevor Kolumbus den Kontinent entdeckte und fand eine große Verbreitung vor allem unter den Stämmen der Blackfoot, Cheyenne, Sioux Kiowa und Arapaho [3]. Sie diente der Kommunikation innerhalb der Stämme, aber auch darüber hinaus war sie wichtig um Handel zu treiben und Verträge zu schließen. Eine Schlüsselrolle kam der Zeichensprache vor allem bei der Jagd und bei Kriegszügen zu. Dabei konnte die Sprache, je nach regionaler Ausprägung, bis zu 3000 verschiedene Zeichen [4] besitzen.

Heutzutage ordnet man die Zeichensprache PISL in die Klasse der Lautsprachbegleitenden Gebärden (LBG) ein. Sie werden durch die vier Parameter Position, Bewegung, Form und Richtung der Hand bestimmt.

In Europa kann man die Entstehungsgeschichte bis zum Beginn des 18. Jahrhunderts zurückverfolgen. Der französische Geistliche Abbé Charles Michel de l'Epée gründete bereits 1755 die erste öffentliche Schule für taube Kinder in seinem Privathaus in Paris. Zu dieser Zeit waren Taubstumme nicht als vollwertige Mitglieder der feudalen Gesellschaft anerkannt und mussten mit Ausgrenzung rechnen. Auch war die Französische Revolution und die damit einhergehende Zeit der Aufklärung noch nicht absehbar. Damals ging die Wissenschaft von einer fehlenden intellektuellen Fähigkeit der Taubstummen aus und ordnete diese Menschen in die Kategorie der geistig Behinderten ein [5].

Die erste deutsche Gehörlosenschule eröffnete 1788 in Berlin. Ende des 19. Jahrhunderts erlitt die Gebärdensprache einen großen Rückschlag durch den Beschluss des Mailänder Kongresses im Jahre 1880 der besagte, dass die Sprache generell aus dem Unterricht verbannt werden sollte und die tauben Kinder zum Sprechen erzogen werden sollten.

Zur Zeit des dritten Reiches wurden Gehörlose reihenweise zwangssterilisiert und geächtet.

Den großen Durchbruch und die breite gesellschaftliche Akzeptanz der Gebärdensprache fing in den 80er und 90er Jahren des 20. Jahrhunderts an, als Schweden und die USA die Sprache offiziell anerkannten [5]. In Deutschland wurde durch das Inkrafttreten des Behindertengleichstellungsgesetzes im Jahre 2002 die politische Grundlage für die Gleich-

berechtigung von Hörbehinderten geschaffen. Es besagt, dass gehörlose Menschen nicht bei offiziellen Behördenbesuchen benachteiligt werden dürfen. Sie haben Anspruch auf einen Gebärdensprachdolmetscher.

Charakteristik und Grammatik

Die Deutsche Gebärdensprache besitzt wie das gesprochene Deutsch klare Regeln für den Satzbau [5]. Dabei ist der auffälligste Unterschied die Position des Verbs. Es steht nicht wie beim gesprochenen Satz in der Mitte sondern am Schluss.

Beispiel: „Die Katze springt auf den Tisch."

DGS: „Tisch Katze springen"

Quelle: http://www.kinderschutz.de/angebote/alphabetisch/aeh.gehoerlose/kommunikationsformen.dgs.lbg.gebaerdensprach-dolmetschen

Das Beispiel verdeutlicht auch einen zweiten wesentlichen Unterschied: Es werden nur die Schlüsselwörter des Satzes gebärdet.

Adjektive werden dabei nicht übersetzt, da sie durch die Mimik des Gesprächspartners transportiert werden.

Eine weitere wichtige Regel besagt, dass Zeitangaben immer am Anfang eines Satzes stehen. Es gibt in der DGS nur drei Zeitformen: Vergangenheit, Gegenwart und Zukunft. Ortsangaben werden direkt nach der Zeitform gebärdet.

Generell gilt, dass eine Gebärde durch den entsprechenden Gesichtsausdruck ausdrucksstärker ist.

Die wichtigsten Faktoren für die Bedeutung der Gebärde sind:

- Die Handstellung
- Die Handbewegung
- Der Ausführungsort
- Der Blickkontakt

Weiterführende Informationen sowie Beispiele enthält das Gebärdenlexion (GebLex) [6] und das freie kollaborative Lexikon der deutschen Gebärdensprache [7].

Fingeralphabet

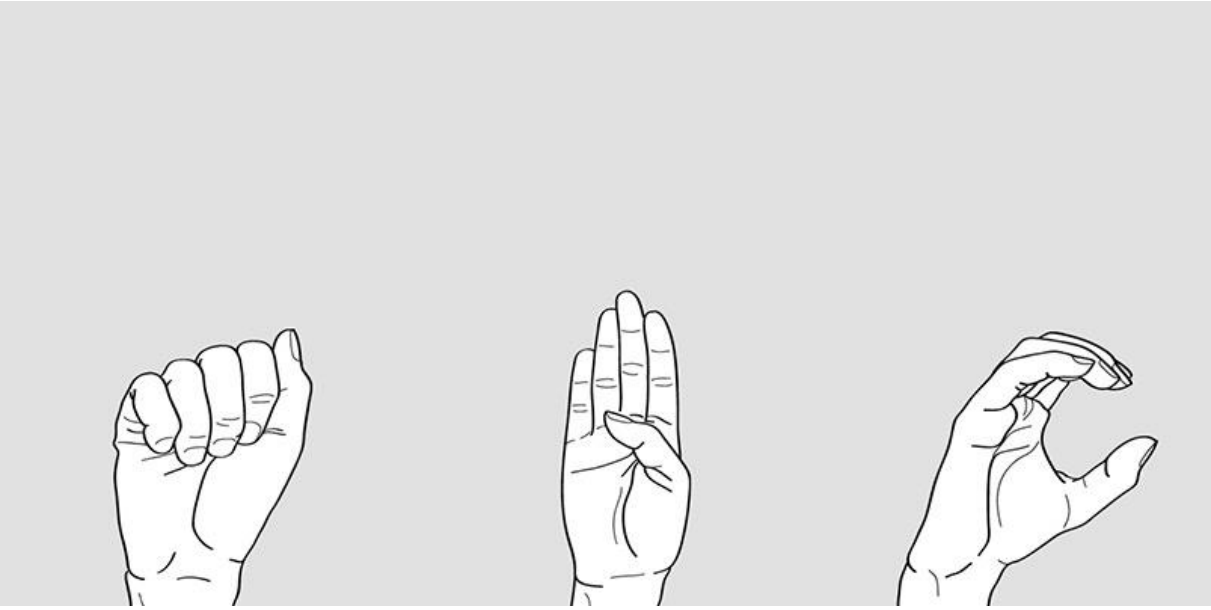
Das Fingeralphabet (auch als Fingersprache oder Daktylologie bezeichnet) dient dazu, die Schreibweise eines Wortes oder Eigennamens mit Hilfe der Finger zu buchstabieren.

Die älteste bekannte Veröffentlichung stammt vom Spanier Juan Pablo Bonet aus dem Jahre 1620.

Am weitesten verbreitet ist das Einhand-Fingeralphabet. Dabei werden alle Buchstaben durch die Finger einer Hand nachgebildet. Es wird nicht nach Groß- und Kleinschreibung unterschieden. Die Buchstaben des Fingeralphabets werden mit der rechten (bei Linkshändern mit der linken) Hand vor der Brust oder etwas seitlich vom Rumpf ausgeführt.

Weltweit sind verschiedene Versionen des Fingeralphabets im Gebrauch. Beispielsweise gibt es das zweihändige Fingeralphabet, das internationale Fingeralphabet (unterscheidet sich vom deutschen Alphabet nur durch die Umlaute und ß) und weitere nationale Abwandlungen.

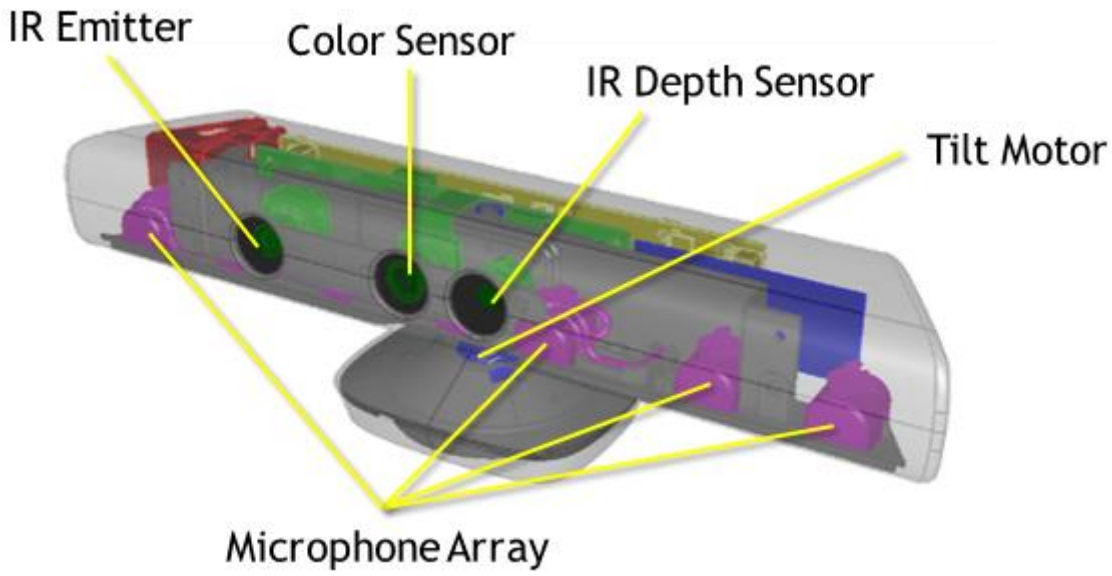
Beispiel: a b c



Quelle: http://www.vektorrausch.de/2008/07/fingeralphabet-wenn-haende-sprechen/

Im Anhang befindet sich eine Übersicht des deutschen Fingeralphabets, ausgeführt von einem Rechtshänder.

Natural User Interface: Kinect



Quelle: http://msdn.microsoft.com/en-us/library/jj131033.aspx

Framerate	30 FPS (Tiefen- und Farbsensor)
Audioformat	16-kHz, 24-bit mono (PCM)
Farbsensor	640 x 480 Pixel, 32-Bit Farbtiefe
Tiefensensor	320 x 240 16-Bit Farbtiefe
Schnittstelle	USB 2.0
Sichtbereich	43° vertikal, 57° horizontal, 27°vertikal kippen

In diesem Projekt wird die Kinect für Windows benutzt .Sie unterscheidet sich von dem ursprünglichen für die Xbox entwickelten Sensor durch den „Near Mode“ (erlaubt Erkennung von Objekten durch Tiefensensor bei minimal 40 Zentimetern Entfernung), bessere Spracherkennung, Unterstützung von mehreren (bis zu vier) Sensoren an einem Gerät und Vorhandensein eines Software-Development-Kits (SDK).

Die Kinect ist für den Massenmarkt ausgelegt und wurde ursprünglich von der Firma PrimeSense entwickelt. Vorteilhaft sind der kostengünstige Anschaffungspreis und das Vorhandensein zahlreicher Frameworks und Bibliotheken.

Frameworks und Bibliotheken

Im Folgenden werden die Rechercheergebnisse und Bewertungen zu verschiedenen Frameworks und Bibliotheken abgegeben, die für die Kinect benutzt werden können.

- **3 Gear Systems SDK**
  http://www.threegear.com
  Die Firma aus San Francisco startete ursprünglich ein Projekt zur Hand- und Fingergestenerkennung mit zwei 3D-Sensoren, welches nur unter einer kommerziellen Lizenz erhältlich war [8]. Mittlerweile bieten sie eine kostenlose SDK-Version (60 Tage) an und setzen beim Tracking nur noch auf einen 3D Sensor (muss nicht Kinect sein). Mit dem SDK lassen sich eine festgelegte Anzahl an Fingergesten erkennen.

- **FORTH 3D Hand Tracking Library**
  http://cvrlcode.ics.forth.gr/handtracking/
  Bibliothek, die am Institut „FORTH Institute of Computer Science" in Griechenland entwickelt wurde. Sie ist relativ rechenintensiv und langsam, da zur Berechnung der Hand das Kamerabild der Kinect herangezogen wird. Leider ist die Bibliothek nicht gut dokumentiert.

- **Microsoft Kinect für Windows SDK 1.5**
  https://www.microsoft.com/en-us/kinectforwindows/
  Offizielles SDK für die Kinect. Das SDK funktioniert gut und bringt eine Vielzahl an kleinen Beispielprogrammen mit.
  Unterstützt bis dato leider keine Finger- und Handgestenerkennung. Jedoch erlaubt das neue SDK 1.8 die Kinect per HTML5 und JavaScript anzusprechen [9].

- **OpenNI + NITE**
  http://www.openni.org/files/nite/
  Gute Alternative zur Candescent NUI Bibliothek. Nutzt für die Hand- und Skeletterkennung entweder das Tiefenbild oder das Kamerabild. Es ist plattformunabhängig und in mehreren Sprachen (C#, JAVA) benutzbar. Die Bibliothek wird von PrimeSense entwickelt.

- **Candescent NUI**
  https://candescentnui.codeplex.com/
  Bibliothek, die Hand- und Fingererkennung ermöglicht. Nutzt OpenNI als Grundlage. Funktioniert stabil, ressourcenschonend und ist gut dokumentiert. Die Bibliothek ermöglicht den Zugriff auf alle relevanten Daten, die bei der Handerkennung entstehen.

Sie lässt sich auch zusammen mit dem Microsoft SDK nutzen.

Experimente und Restriktionen

Gebärdensprache mit der Kinect in Echtzeit (Antwortzeit max. 10 ms) zu erkennen ist nahezu unmöglich, da die Auflösung und Verarbeitungsgeschwindigkeit der Kinect zu gering ist. Das liegt zum einen daran, dass der Tiefen- und Bildsensor nur mit einer Bildwiederholfrequenz von 30 FPS arbeitet. Zum anderen benötigen die Algorithmen zum Erkennen des Skelettes und der Entfernung auch Rechenzeit. Da dies alles softwareseitig geschieht kommt man in der Regel nur auf eine Bildrate von 20-25 Frames pro Sekunde.

Bei der Entwicklung mit dem Tiefenbild der Kinect stellt man schnell die starke Abhängigkeit des Sensors von der Helligkeit der Umgebung und des Kontrastes fest. Der Sensor funktioniert am besten bei einer Entfernung von ca. einem Meter und normalem Umgebungslicht.

Störend ist auch, dass das Sichtfeld des Tiefensensors nur 57° beträgt. Dadurch ist man in seiner Bewegung eingeschränkt. Für ein größeres Sichtfeld bzw. genauerer (Tiefen) Erkennung bietet sich die Verwendung einer zweiten Kinect an.

Aufgefallen ist weiterhin, dass sich bei der Erkennung von Handgesten nur eine Person im sichtbaren Bereich aufhalten sollte und die zu erkennende Hand vor direkter Sonneneinstrahlung abgeschirmt ist. Weiterhin ist es vorteilhaft, wenn an der Hand keine Ringe sind. Den Arm sollten auch möglichst keine Bänder oder Ringe schmücken und er sollte bis zum Ellenbogen nicht bedeckt sein.

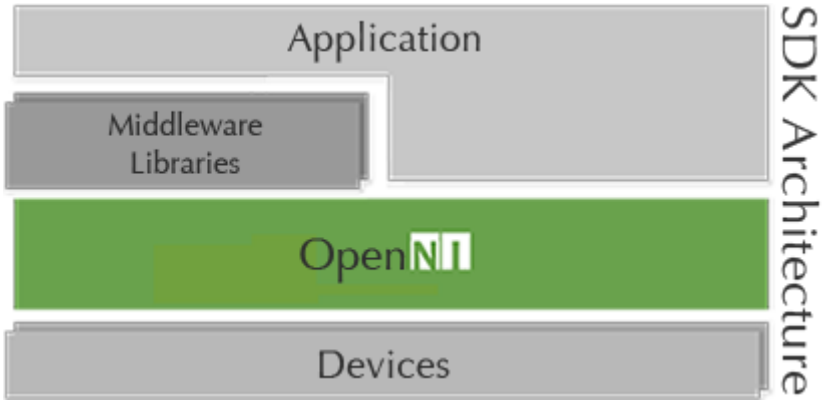
Implementierung

Softwarearchitektur

Vor dem Beginn der Entwicklung von Software mit der Kinect bieten sich grundsätzlich zwei Möglichkeiten an, um diese anzusprechen.

1. Offizielles Microsoft Kinect SDK + Treiber
   Nachteil: kein Finger- und Handtracking möglich
2. Experimenteller PrimeSense Treiber mit OpenNI
   Vorteil: Unterstützung von Handtracking. Generische Softwareschnittstelle

Der entstandenen Applikation liegt nicht das Microsoft Kinect SDK zugrunde sondern die Open-Source Programmbibliothek OpenNI (http://www.openni.org/). Sie zeichnet sich besonders durch die Abstraktion der Schnittstelle zur Kinect aus, was den Vorteil hat, das auch Sensoren von anderen Herstellern wie Asus und PrimeSense verwendet werden können. Das ist vorteilhaft, da die Applikation nicht der Politik des Herstellers ausgeliefert und von dessen Interessen abhängig ist. Weiterhin bietet OpenNI noch zahlreiche Middleware Bibliotheken zur Erkennung von Händen, Objekten, Volumina an. Siehe: http://www.openni.org/software/?cat_slug=file-cat1.



Quelle: http://www.openni.org/

Durch das Wählen der 2. Möglichkeit bietet sich in der Entwicklung eine große Flexibilität, da alle Komponenten Open-Source sind und der Quellcode frei zugänglich ist. Oftmals musste direkt der Quellcode studiert oder die Entwickler kontaktiert werden, da die Dokumentation teilweise sehr dürftig ist.

Die Anwendung ist ein Reaktives System, d.h. dass sie auf die Benutzereingaben (Fingergesten) reagiert, sie verarbeitet (Vergleich mit aufgenommenen Gesten) und dann das ermittelte Ergebnis (Geste x gefunden) zurückgibt. Es handelt sich um ein weiches Echtzeitsystem.

Die Applikation ist folgendermaßen aufgebaut:

Für die Implementierung wurde als Programmiersprache C# und als Entwicklungsumgebung Microsoft Visual Studio 2012 gewählt.

Die Anwendung verwendet als grafische Benutzeroberfläche (GUI) das Template einer Windows Form Applikation. Visual Studio bietet hierfür komfortable Funktionen und einen Designer, der es erlaubt einfach Steuerelemente zu erstellen und zu bearbeiten.

Soweit es möglich war, wurde auf eine strikte Trennung zwischen der GUI, der Applikationslogik und der Datenhaltung geachtet, um dem Model-View-Controller Modell zu entsprechen. Die einzelnen Klassen sind lose gekoppelt und kommunizieren über Events miteinander (event driven). Das heißt, dass z.B. der Klick auf den Button „Start Detection" ein Event auslöst, das von einer abonnierenden Klasse gefangen wird und den zugehörigen Quellcode ausführt. In diesem Fall würde es sich mit der Kinect verbinden und die Fingererkennung starten.

Klassen, die Ressourcen verwalten, wie z.B. die Klasse `DataPersister.cs`, die den Zugriff auf die Datenbank, sowie den XML-Export regelt und verwaltet sind als statische Klassen umgesetzt, da sie nicht instanziiert werden sollen. Auch die Hilfsklasse `MathHelper.cs` die verschiedene mathematische Berechnungen durchführt, ist nur ein Service bzw. eine Containerklasse und deshalb statisch. Weiterhin werden Datenbankzugriffe und Exporte asynchron in einem extra Thread (BackgroundWorker) behandelt, damit die Reaktion der GUI nicht beeinträchtigt wird.

Die wichtigste Klasse ist der `GestureDetector.cs`. Er ist gewissermaßen das Herzstück der Anwendung und für das Auswerten und Analysieren der Tiefeninformationen der Kinect zuständig. Er ist als Singleton implementiert, da es zur Laufzeit des Programms nur eine Instanz geben soll. In ihr wird versucht aus den erkannten Händen und Fingern die zugehörigen Gesten (mit Wahrscheinlichkeit) zu berechnen. Der Controller wirft folgende Events:

- Hand gefunden
- Keine Hand gefunden
- Geste gefunden

Diese werden von der GUI entsprechend ausgewertet und die Informationen (Handmittelpunkt, Fingeranzahl, erkannter Buchstabe, ...) angezeigt.

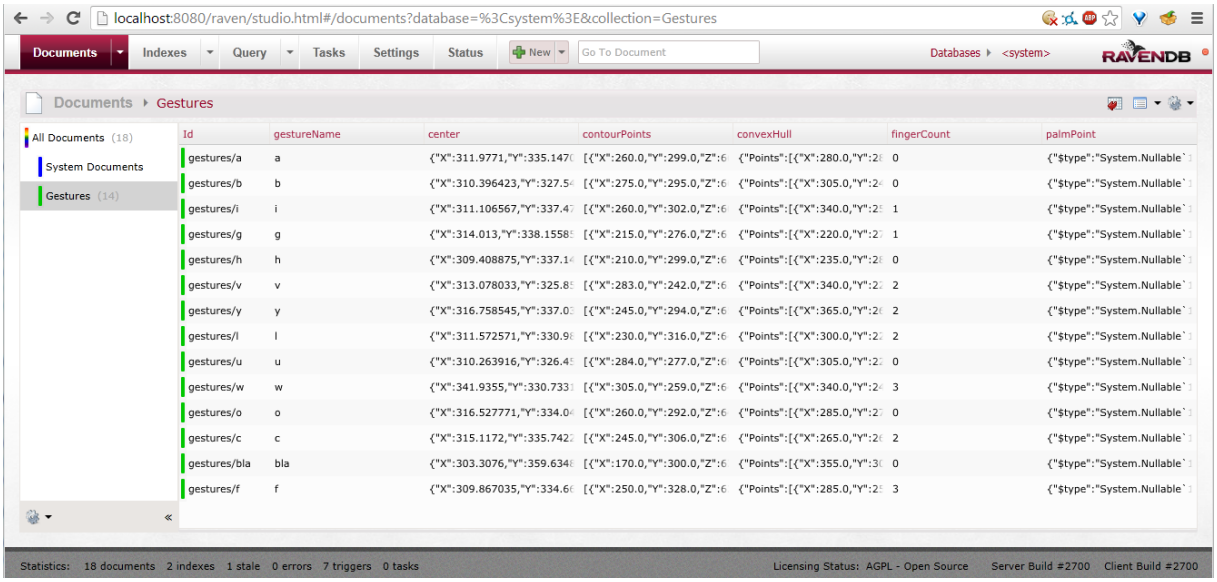
Die wichtigste Modellklasse ist `Gesture.cs`. Sie beherbergt die Daten der Geste:

- Name des Buchstabens/Geste
- 2D-Punkte der Kontur
- 2D-Punkte der umgebenden Hülle
- Fingeranzahl
- Handflächenmittelpunkt
- Mittelpunkt
- Volumenangaben

Da die Software nur vorher aufgenommene Gesten erkennt, kann mit der Software eine Geste aufgenommen werden. Sie wird mit dem in .NET eingebauten Serialisierungsmechanismus automatisch zu XML (Beim Export) oder JSON beim Speichern in der Datenbank.

Als Datenbank wurde auf die dokumentenbasierte NoSQL-Datenbank „RavenDB“ (http://ravendb.net/) gesetzt. Sie bietet eine gute Integration in C# und ist als eingebettete Variante ressourcenschonend und leichtgewichtig. Sie ist Open-Source und über github frei erhältlich. Die großen Vorteile sind, dass sie ohne Datenbankschema auskommt, transaktional (ACID) ist, eine eingebaute Suchmaschine (Lucene) besitzt und eine gute Performance bietet. Durch die Schemalosigkeit entstand praktisch kein Datenbankmodellierungsaufwand, da die erhaltenen Daten der Gesten einfach als neue Entität abgespeichert werden und der Name der Geste der Schlüssel zu dem Datensatz ist. Während der laufenden Applikation kann der Inhalt der Datenbank im Browser betrachtet und modifiziert werden.

Die URL zur Webview lautet:
http://localhost:8080/raven/studio.html#/documents?database=%3Csystem%3E&collection=Gestures



3D Hand- und Fingererkennung

Als Grundlage für die Erkennung der Hände und Finger fungiert die Bibliothek Candescent NUI von Stefan Stegmueller. Sie nutzt OpenNI, um die Sensordaten auszulesen. Danach führt sie eine Reihe von Berechnungen durch und bereitet die gewonnenen Daten auf.

Arbeitsweise der Candescent NUI Bibliothek [10]:

Handerkennung:

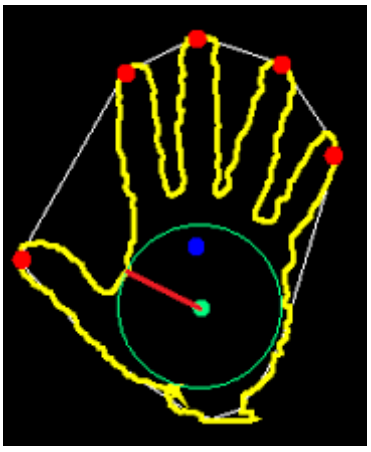
1. Pixel (der Hände) vom Tiefenbild erkennen (z-Range)
2. Projektion in 2D Raum
3. Pixel mittels K-Means Algorithmus in zwei Gruppen unterteilen bzw. clustern
4. Konvexe Hülle mit Graham Scan Algorithmus finden
5. Handkonturen mittels modifiziertem Moore-Neighbor-Tracing Algorithmus finden

Fingererkennung:

6. Fingerspitzen erkennen: Triangulation
7. Mittelpunkt der Handfläche finden
8. Hände zwischen beiden Frames synchronisieren
9. Finger erkennen

Dabei wird das von OpenNI angebotene Feature genutzt, nur die Tiefeninformation eines bestimmten Bereiches (500mm - 800mm) zu scannen und für die Hand- und Fingererkennung zu nutzen. Das spart Rechenzeit und entfernt störende Artefakte.

Hat die Tiefenkamera die Hand des Nutzers erkannt und wurde der Erkennungsalgorithmus erfolgreich ausgeführt, ist folgendes zu sehen:



Diese Daten werden von der Klasse `Gesture.cs` anschließend repräsentiert und abgespeichert. Danach stehen sie für die potentielle Gestenerkennung zur Verfügung.

Nun kommt der Hauptteil des selbst entwickelten Algorithmus: Erkennung und Zuordnung der Handgeste zu einem Buchstaben aus dem Fingeralphabet.

Hierbei werden folgende Kriterien zur Kategorisierung herangezogen:

- Anzahl der Finger
- Umriss der Hand
- Mittelpunkt der Hand
- (Skalierungsfaktor)

Innerhalb eines einstellbaren Zeitintervalls (500-5000 ms) werden die Daten des Tiefensensors aufgenommen und mit den bereits aufgenommenen und in der Datenbank gespeicherten Gesten verglichen. Dabei werden zuerst alle Gesten, die die gleiche Fingeranzahl besitzen für den Vergleich herangezogen.

Das sind meistens mehrere Möglichkeiten. Zeigt man beispielsweise ein „a“ in die Kamera, dann selektiert der Algorithmus die Zeichen „a“, „b“, „h“, „u“, „o“. Nun werden die Koordinaten der Geste noch durch eine Translation des Gestenmittelpunktes auf den Ursprung relativiert (Position der gezeigten Geste ist unwichtig). Anschließend wird für die potentiellen Zeichen mittels der Hausdorff Distanz der Abstand der Gesten zueinander berechnet. Dafür werden die Konturpunkte (gekennzeichnet durch gelbe Linie) der gerade vom Bildschirm aufgenommenen Hand mit den aus der Datenbank selektierten Gesten verglichen. Die Gesten sind sich umso ähnlicher, je kleiner der Hausdorff-Abstand ist [11].

Letztendlich wird die Geste mit der geringsten Hausdorff-Distanz gesucht und als Suchergebnis im Programm angezeigt. Basis für die Implementierung und die mathematischen Grundlagen war das Paper „Hausdorff Distance for Shape Matching“ [12].

Optimal funktioniert die Fingererkennung, wenn die Entfernung zum Gerät zwischen 500 – 800 mm beträgt.

Diese Herangehensweise zur Gestenerkennung ist sehr ressourcenschonend, da nur das Tiefenbild der Kamera (schlechter aufgelöst als Farbbild) herangezogen wird und die anschließenden Berechnungen nur anhand von Punkten im 3 bzw. 2 dimensionalen Raum durchgeführt werden. Während der Recherche ist aufgefallen, dass die meisten Lösungen zur Finger- und Handerkennung auf der Filterung des Kamerabildes basieren, was deutlich rechenintensiver ist. Weiterhin erlauben es die meisten Beispielprojekte und Bibliotheken nur Standardgesten wie Wischen nach rechts, links, oben und unten, sowie Winken durchzuführen.

Siehe: http://www.openni.org/software/?cat_slug=file-cat1, https://kinectdtw.codeplex.com/)

Herausforderungen

Natürlich bot das Projekt auch zahlreiche Herausforderungen:

- Den richtigen Algorithmus zur Berechnung der Ähnlichkeit des Bildes finden.
- Guten Kompromiss zwischen Performance und Berechnungsgenauigkeit ermitteln
- Richtige Treiber und Bibliotheksversionen einsetzen
- Eventhandler beenden
- Datenbankabfragen cachen (Gesten werden nun nur zu Programmstart und nach dem Anlegen einer neuen Geste direkt aus der Datenbank gelesen)
- Detectionrate einführen, zu der Tiefenbild auf Gesten gescannt wird
- Einfache mathematische Funktionen effizienter gestalten. Z. B.
  Euklidische Distanz Berechnung
  ursprünglich: `00:00:00.0298668`
  Anschließend: `00:00:00.0002261`

Installation

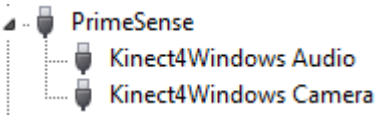
Die folgenden Schritte sollten in der vorgegebenen Reihenfolge abgearbeitet werden, damit die Software ordnungsgemäß funktioniert.

Wichtig dabei ist vor allem die Einhaltung der Versionsnummern der Bibliotheken.
Außerdem ist die Software auf die Kinect für Windows optimiert.
Der gesamten Anwendung liegt eine 64-Bit Architektur zugrunde, deswegen werden die Treiber und Bibliotheken auch nur in 64-Bit Version installiert.

1. **Installation von OpenNI SDK v1.5.4.0 x64**
   Download der Bibliothek von: http://www.openni.org/openni-sdk/openni-sdk-history-2/
   Anschließend installieren.

2. **Installation von alternativen Kinect Treiber (SensorKinect 5.1.2.1 x64)**
   Der Treiber ist unter https://github.com/avin2/SensorKinect/tree/unstable/Bin zu finden. Er sollte SensorKinect093-Bin-Win64-v5.1.2.1.msi heißen.
   Anschließend installlieren.

3. **Installation prüfen**
   Nach erfolgreicher Installation muss im Gerätemanager folgendes zu sehen sein:

   Danach am besten den NiViewer64 im Samples-Ordner von OpenNI starten (Im Programme Menü zu finden). Falls die Installation erfolgreich war sollte der NiViewer64 ein Video- und Tiefenbild anzeigen.
   Weitere gute Tipps und Vorgehensweisen sind auch unter den folgenden Adressen zu finden:
   - http://kinect-i.blogspot.com.br/2012/05/how-to-install-and-use-openni-microsoft.html
   - http://www.kinecthacks.com/guides/install-kinect-on-your-pc-and-start-developing-your-programs-disclaimer/
   - http://ramsrigoutham.com/2012/06/07/installation-of-kinect-on-windows-7-openni-sensor-kinect-and-nite/

4. **Installation des Programms FingerSpellingApp**
   Download von https://github.com/samuelstein/FingerSpellingKinect.
   Den Installer im Ordner FingerSpelling ausführen. Anschließend starten.

Ausblick

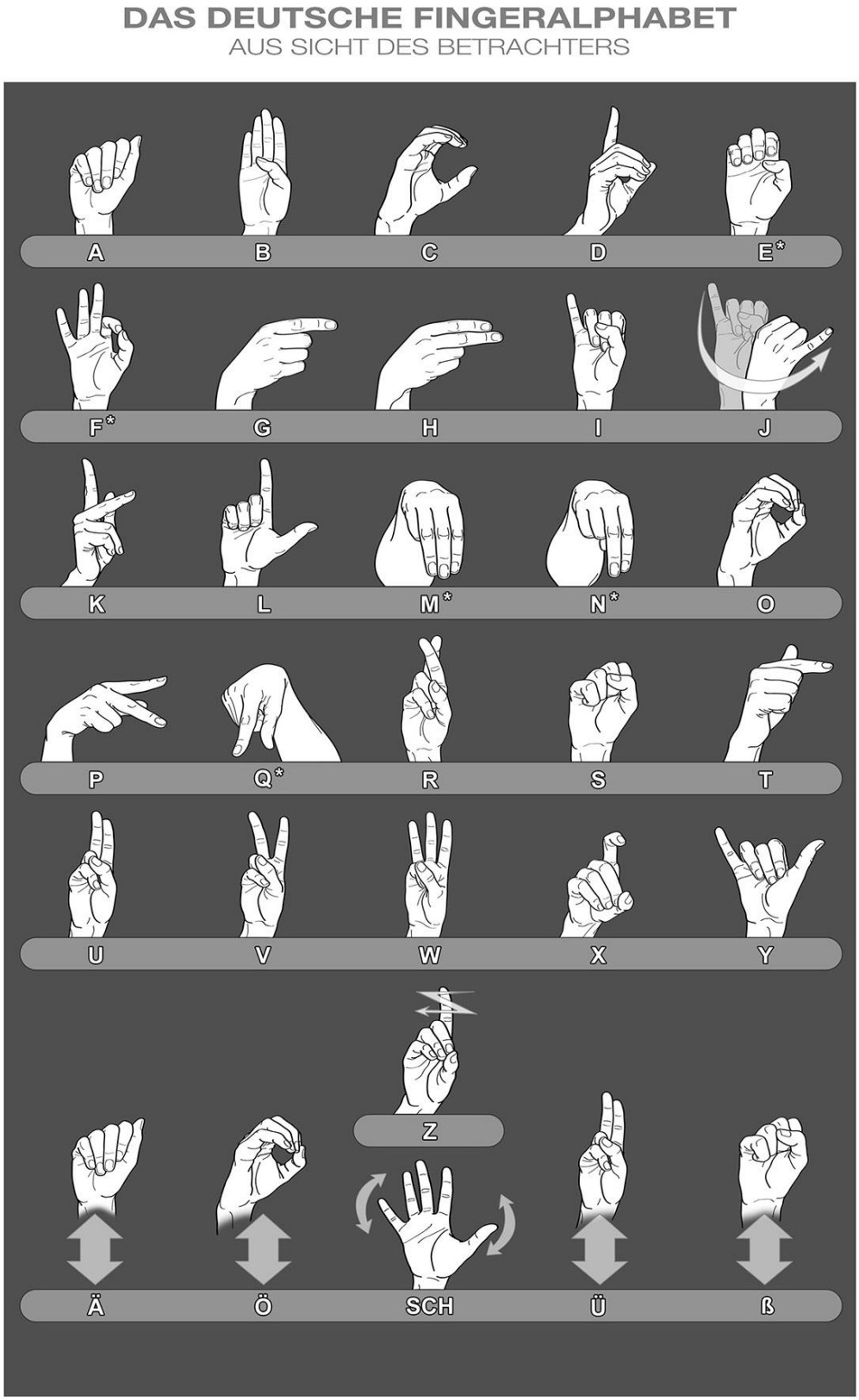
Bedauerlich ist, dass Microsoft in ihrem SDK keine Möglichkeit mitbringt, Hand- bzw. Fingererkennung durchzuführen.

Spannend wird die Veröffentlichung des neuen 3D Sensors Kinect One zu Beginn des nächsten Jahres werden. Er soll eine doppelt so große Tiefenauflösung mitbringen und um ein Drittel schneller in der Berechnung sein.

Die Software ist modular aufgebaut um eine spätere Erweiterung zu befördern. Beispielsweise wäre eine erleichterte Texteingabe mittels T9 denkbar. Über die Exportschnittstelle können die Daten der Gesten für eine weitere Verwendung aufbereitet werden.

Anhang

Deutsches Fingeralphabet



Literaturverzeichnis

[1] S. Stegmueller, „Candescent,“ [Online]. Available: http://www.candescent.ch/. [Zugriff am 26 09 2013].

[2] „Wikipedia Gebärdensprache,“ [Online]. Available: https://de.wikipedia.org/wiki/Geb%C3%A4rdensprache. [Zugriff am 29 09 2013].

[3] W. Tomkins, Indian Sign Language, 5th ed. 1969 Hrsg., New York: Dover Publications, 1931.

[4] „Die Indianer Nordamerikas,“ -. [Online]. Available: http://www.indianerwww.de/indian/verstaendigung.htm. [Zugriff am 30 09 2013].

[5] S. Wolf und S. Strixner, Kleines Wörterbuch der Gebärdensprache, 5. durchgesehene und überarbeitete Auflage Hrsg., Wiesbaden: marixverlag GmbH, 2012.

[6] M. Szczepanski, „Fachdienste für Hörgeschädigte,“ [Online]. Available: http://www.michaelszczepanski.de/geblex.html. [Zugriff am 30 09 2013].

[7] „DGS.Wikisign,“ [Online]. Available: http://dgs.wikisign.org/. [Zugriff am 28 09 2013].

[8] B. Schwan, „Heise News,“ 18 10 2012. [Online]. Available: http://www.heise.de/newsticker/meldung/Zwei-3D-Kameras-erfassen-Fingergesten-auf-Millimeter-genau-1724962.html. [Zugriff am 30 09 2013].

[9] A. Neumann, „Heise Developer,“ 19 09 2013. [Online]. Available: http://www.heise.de/developer/meldung/Kinect-versteht-HTML5-und-JavaScript-1961438.html. [Zugriff am 29 09 2013].

[10] F. T. Cerezo, „3D Hand and Finger Recognition using Kinect,“ Universidad de Granada (UGR), Spain.

[11] „Hausdorff distance between convex polygons,“ 1998. [Online]. Available: http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html. [Zugriff am 26 09 2013].

[12] H. Şengül AKAÇ und D. BİNGÜL, „HAUSDORFF DISTANCE FOR SHAPE MATCHING,“ Department of Computer Engineering, İstanbul Kültür University, 2010.