Masterprojekt 2 - Projektbericht
3D Interaktionen in Smart Homes

Edo Kriegsmann

Contents

1 Einleitung

Wie schon im Vorsemester beschäftige ich mich auch im Projekt 2 mit der Microsoft Kinect-Kamera und dem Potential der Daten, welche man mit dieser gewinnen kann. So sollen die 3D-Tiefeninformationen, sowie die Farbinformationen der RGB-Kamera gebündelt und zur Ermittlung menschlicher Aktionen, seiner Emotionen sowie dessen Alters und Geschlechts genutzt werden. Die Zielsetzung dieser Sensoraus- und Bewertung der Kamerainformationen ist die Schaffung eines vorverarbeiteten Informationspools, aus welchem sich aufgesetzte Anwendungen bedienen können. Im Gegensatz zu primitiven Bilderkennungssystemen, wie TTL-Kameras oder reguläre Farb-Kameras, entsteht so ein detaillierter Informationspool über die erkannten Personen im Sichtbereich der Kamera. Anwendungen sind in den Berei-chen von Ambient Intelligence, Ubiquitous Computing sowie Mensch-Companion-Interaktion einzuordnen, ((vgl. GI-Jahrestagung, 2011) , (vgl. Weiser, 1991)) wie beispielsweise sich selbstständig an den Bewohner anpassende Wohnräume, sich selbst individualisierende Werbetafeln oder die Steigerung des Bedienkomforts von (Fahrkarten-) Automaten.

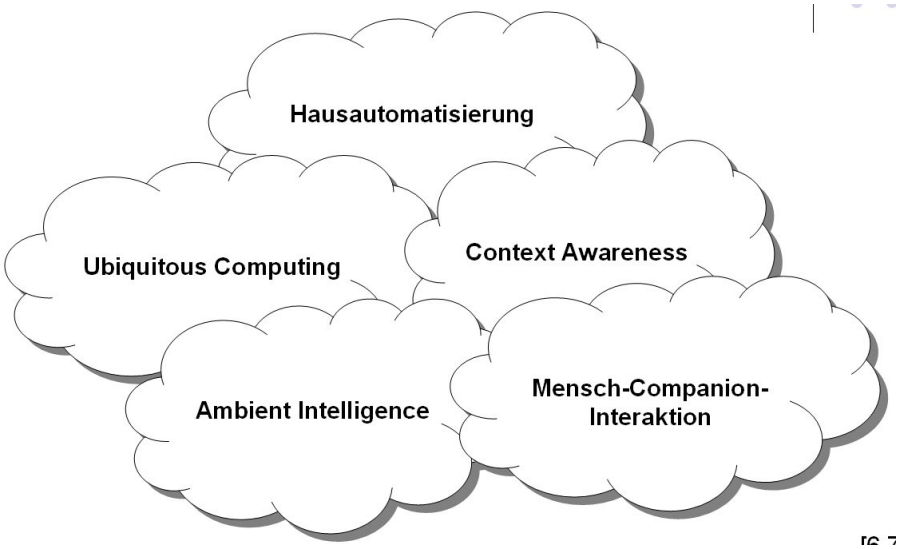

Figure 1.1: *Bereiche der Arbeit*

2 Vorarbeiten

Bereits im Projekt 1 konnten wichtige Grundlagen für diese Überlegungen geschaffen werden. So ist die Auswahl der geeigneten Hilfsmittel auf Seiten der Hard- und Software getroffen worden und es fand eine Einarbeitung in diese Bereiche statt. Im folgenden Abschnitt möchte ich kurz auf die Vorarbeiten eingehen.

2.1 Microsoft Kinect Datenerfassung

Die Verarbeitung der Tiefeninformationen habe ich im Projekt 1 mit mehreren Demoprogrammen untersucht. Mit Hilfe eines Algorithmus wurden die Tiefeninformationen in einem Programm soweit reduziert, dass auch externer, verhältnismäßig langsamer Hardware die Weiter-verarbeitung möglich gewesen wäre. In folgendem Bild links 2.1 ist ein grafisch aufgearbeiteter Filter, welcher die Tiefeninformationen der Kamera in einer gemittelten 32x32x255 Matrix ausgibt, zu erkennen.

Die Kinect-Skeleton-Engine ermöglichte es, einzelne Skelett-Punkte im 3D-Raum der erkann-ten Personen abzufragen und diese Informationen weiter zu verarbeiten. Die Erfassung und Verarbeitung dieser Informationen in Echtzeit ermöglicht es, diese Daten unmittelbar zu verwenden und auf Eingaben direkt reagieren zu können. Im Projekt 1 habe ich eine Art Spiel entwickelt, um diese Möglichkeit der Skeletterkennung zu erproben. ”Dabei las ich die Skelettinformationen der bis zu zwei Spieler aus und rechnete in das auf dem Monitor angezeigte Skelettbild ein ”Laserschwert” ein. Dies war ein um Faktor zwei verlängerter Vektor, erstellt aus der Differenz zwischen XYZ-Koordinate des Ellenbogens und des Handgelenks, addiert zur Koordinate des Handgelenks. Der Effekt war ein sich im 3D-Raum korrekt nach dem Unterarm richtendes ”Laserschwert” welches auf Bild 2.2 rechts zu sehen ist.” (vgl. Kriegsmann, 2011)

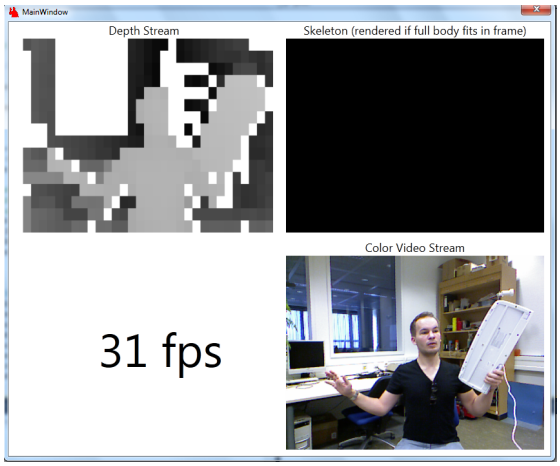

Figure 2.1: *Filterung auf 32x32px a 255-Stufigem Tiefenbild*

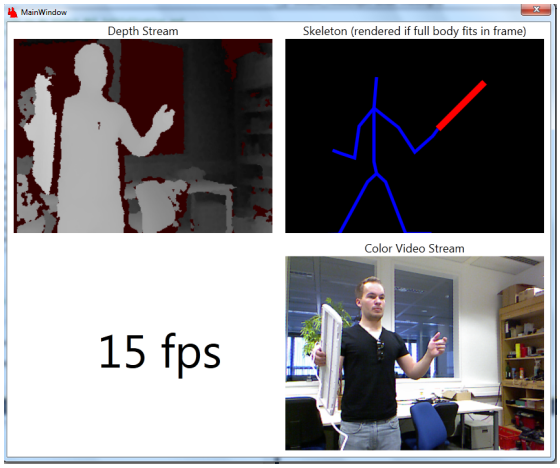

Figure 2.2: *Modifizierung der Skelettinformationen*

2.2 Erkennung von Emotionen und des Alters/Geschlechts von Personen

Zur Erkennung der Emotionen und Körpermerkmale kam im Projekt 1 die vom Frauenhofer Institut entwickelte Engine names ”SHORE” - Sophisticated High-speed Object Recognition Engine- zur Verwendung. (vgl. IIS, 2011) Diese Bibliothek ermöglichte es, in Farbbild-Informationen menschliche Gesichter und deren Emotionen zu erkennen. In Projekt 1 konnte durch eine Einarbeitung in diese Bibliothek und die Demosoftware zudem die Basis der Weiterverarbeitung dieser Informationen geschaffen werden.

2.3 Fazit der Vorarbeiten

Im Projekt 1 konnten erfolgreich die Grundlagen der Auswertung der unterschiedlichen Sensordaten geschaffen werden. Es ist sowohl eine Auswertung der Skelettmodelle/Tiefeninformationen, wie auf Seite 4 beschrieben, als auch eine Einarbeitung in die SHORE-Bibliothek möglich gewesen. Dies legte den Grundstein der gemeinsamen Verarbeitung der Informationen, auf welche ich im kommenden Abschnitt eingehen werde.

3 Weiterentwicklungen Projekt 2

In den folgenden Abschnitten möchte ich auf die Neuerungen und die Ausrichtung der Projektarbeit eingehen.

Folgender Grobentwurf (siehe Bild 3.1) umschreibt in etwa das Ziel, welches ich im Projekt 2 verfolgte und weitgehend erreichte. So sollen die Informationen der SHORE-Bibliothek und die der NUI-Bibliothek (vgl. Microsoft, 2011, Seite 14) in einer gemeinsamen Software gebündelt und ausgewertet werden. Diese Daten erzeugen so die Basis einer zweigeteilten Weiterverarbeitung. Zum einen sollen die Daten dahingehend so weit vorverarbeitet und reduziert werden, dass eine Nutzung dieser mit einfacher Hardware möglich ist - der Low-Level-Teil. Zum anderen sollen die Möglichkeiten ausgelotet werden, welche sich aus der detaillierten Gesamtheit der gebündelten Informationen eröffnen - der High-Level-Teil.

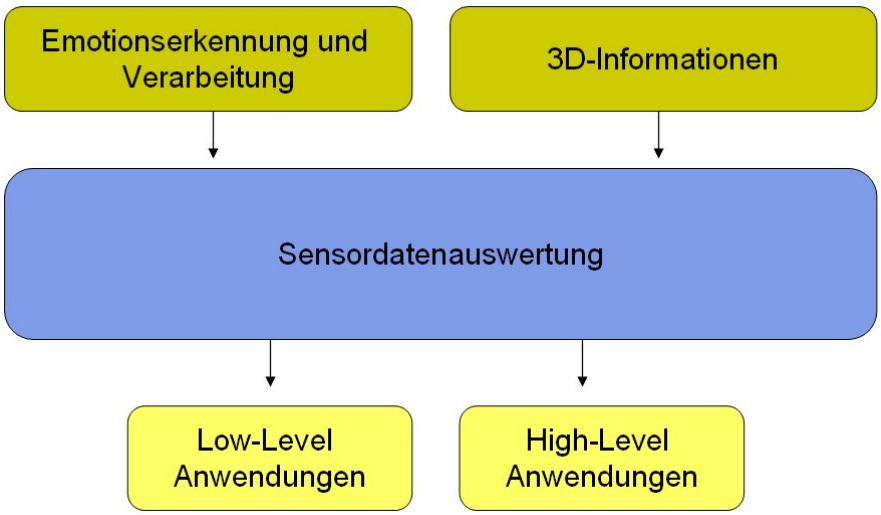

Figure 3.1: *Verarbeitung der Daten*

3.1 Verbinden der Daten

Ein großer, offener Punkt aus der Projektarbeit 1 war es, die Daten der unterschiedlichen Systeme geschickt in einer Entwicklungsumgebung zu bündeln. So standen die Informationen der SHORE-Bibliothek lediglich einer CPP-Entwicklungsumgebung zur Verfügung, während die 3D-Informationen favorisiert in einer C#-Entwicklungsumgebung verarbeitet werden sollten. Dieses Problem löste eine im Rahmen des LivingPlace entstandene Wrapper-Software, welche die Informationen der SHORE-Bibliothek in einer Active MQ-Datenbank ablegte (vgl. Foundation, 2012) . Alle Informationen, welche man aus der SHORE-Auswertung verwenden wollte, wurden zu einem String gebündelt (serialisiert) und über sogenannte ”json”-Nachrichten dem Active MQ-Dienst zur Verfügung gestellt. Im Anhang auf Seite 15 befindet sich ein Code-Fragment (gekürzt), welches das Zusammenstellen und Senden einer solchen Nachricht zeigt.

Nun war es aus einer C#-Entwicklungsumgebung heraus möglich, asynchron auf den Active-MQ-Datenpool zuzugreifen und so die verarbeiteten Informationen der SHORE-Bibliothek auszulesen. Ein Codebeispiel (gekürzt) findet sich im Anhang auf Seite 17. Diese so empfangenen ”json”-Nachrichten wurden dann wieder deserialisiert, um auf die übertragenen Objekte zuzugreifen. Im Beispiel-Bild Bild 3.2 wird das Alter durch die SCORE-Bibliothek ermittelt, übertragen und in C# wieder ausgegeben.

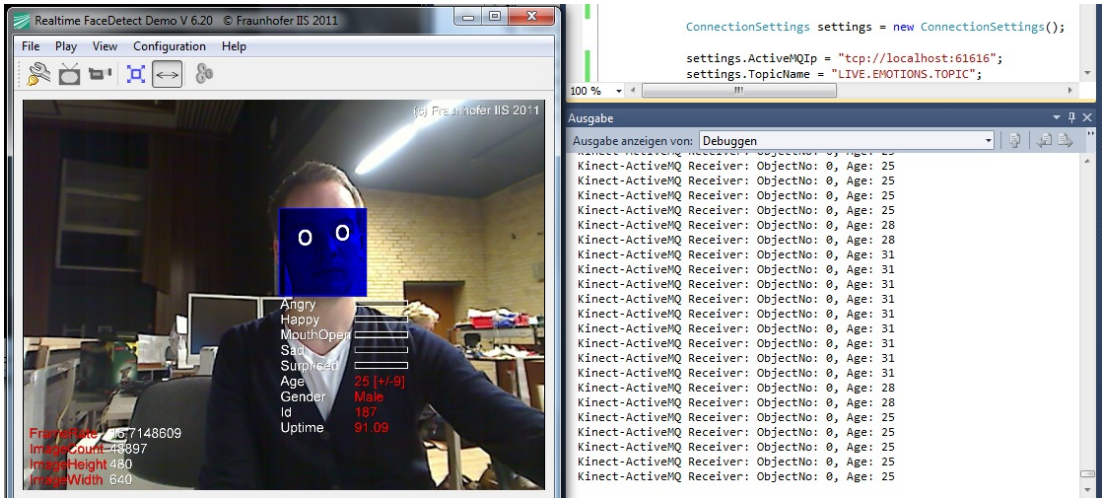

Figure 3.2: *SHORE-Ausgewertete Daten in C#*

Der Umweg über den ActiveMQ-Dienst hat den weiteren Vorteil, dass bei Einsatz dieses Systems in einer Produktivumgebung diese Daten von weiteren Systemen direkt genutzt werden können. Im Folgenden ergab sich dann ein Komponentenentwurf, welcher in Grafik 3.3 zu erkennen ist.

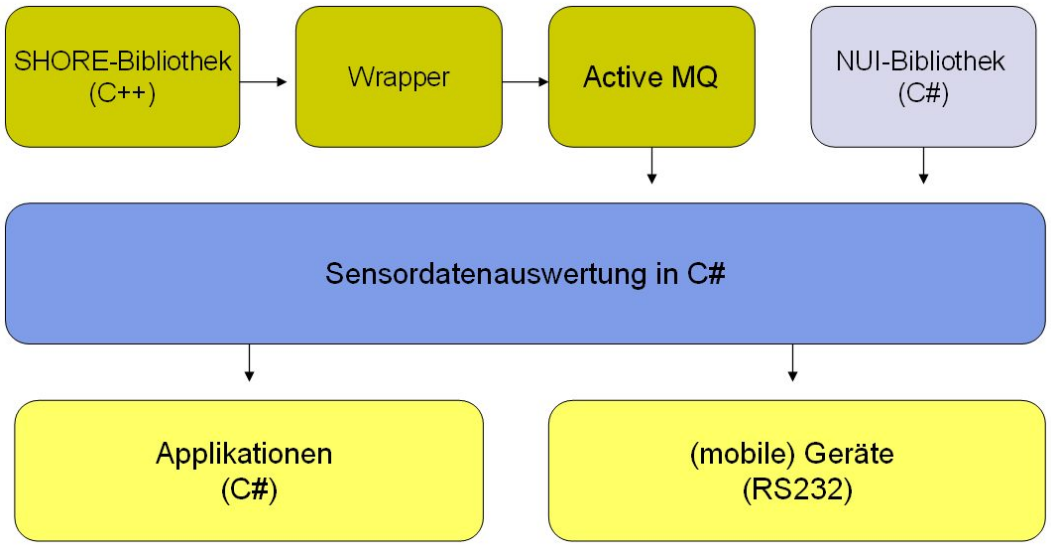

Figure 3.3: *Verbindungsmodell*

3.2 Weiterentwicklungen im Bereich der Datenerfassung

Im November 2011 erschien eine neue SDK-Version des Microsoft Kinect SDKs. Dieses behob einige wenige Fehler der ersten Version, verbesserte die Performance und Präzision der Skeleton-Verarbeitung und enthielt einige Softwarebeispiele, welche diese Verbesserungen eindrucksvoll demonstrierten.

Kinect-Audioverarbeitung

Die Kinect-Audioverarbeitung stach in der Betrachtung der Neuerungen des SDKs besonders hervor. So ist es nun über das SDK möglich, die Informationen des Mikrofons, welches in der Kinect integriert ist, umfassend zu nutzen. Es lässt sich über die Softwareschnittstelle die Richtung einer Geräuschquelle lokalisieren (Winkelangabe zur Position der Kinect) und über die integrierte Spracherkennung können Kommandos empfangen und erkannt werden. In einem mitgelieferten Demoprogramm konnte man diese Funktionen testen. Folgende Bildschirmgrafik zeigt die Software, welche das Wort "rot" aus einer Auswahl möglicher Wörter erkannte und zudem die Richtung der Geräuschquelle lokalisierte (Blaue Linie).

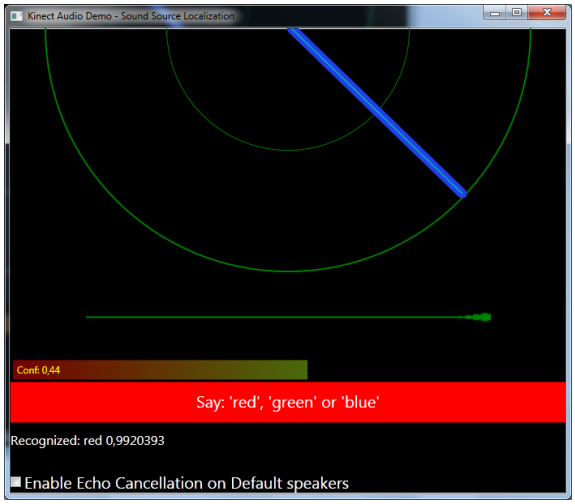

Figure 3.4: *Richtung einer Geräuschquelle inkl. Spracherkennung*

Die Möglichkeiten der Kinect-Audioverarbeitung wurden in den Vorarbeiten als unzureichend eingeschätzt und nicht ausreichend betrachtet. Diesen falschen Eindruck widerlegten die Demonstrationsprogramme schnell und zeigten vielfältige Anwendungsmöglichkeiten. So sollte die optionale Verarbeitung dieser Informationen in der Masterarbeit bedacht werden.

Skelett-Erkennung und Verarbeitung

Auch auf Seiten der Skelett-Erkennung und Verarbeitung konnten Fortschritte erzielt werden. So ist diese Erkennung nun subjektiv ein wenig präziser als in der Vorversion und arbeitet um einiges schneller. Eine Software, welche sowohl das Tiefen- und RGB-Bild, als auch die Skelettinformationen verarbeitet, läuft etwa 30% schneller als in der Vorversion[1]. Zudem ist ein Mapping des Koordinatensystems der Skelettinformationen auf das Koordinatensystems der RGB-Bildinformationen von Grund auf möglich. Dies erlaubt es, in der High-Level-Verarbeitung augmented-reality-Elemente[2] einzubringen.

Um dies zu demonstrieren, programmierte ich erneut ein virtuelles Laserschwert, welches aber im Gegensatz zu der Vorarbeit aus Projekt 1 (siehe Seite 4) im RGB-Bild platziert wurde. Hier musste die Software dahingehend umgeschrieben werden, als dass die Punkte des Skelett-Koordinatensystems in den RGB-Raum gemappt werden, um hier korrekt positioniert angezeigt zu werden. In diesem RGB-Koordinatensystem fand dann die Berechnungen der Schwertposition und korrespondierender Linien statt. Im Anhang auf Seite 16 befindet sich ein gekürzter Codeausschnitt, welcher das Auslesen und Mappen der Koordinaten, sowie das Zeichnen des ”Laserschwerts” beschreibt. Das Ergebnis dieses Programms kann man in folgender Grafik sehen:

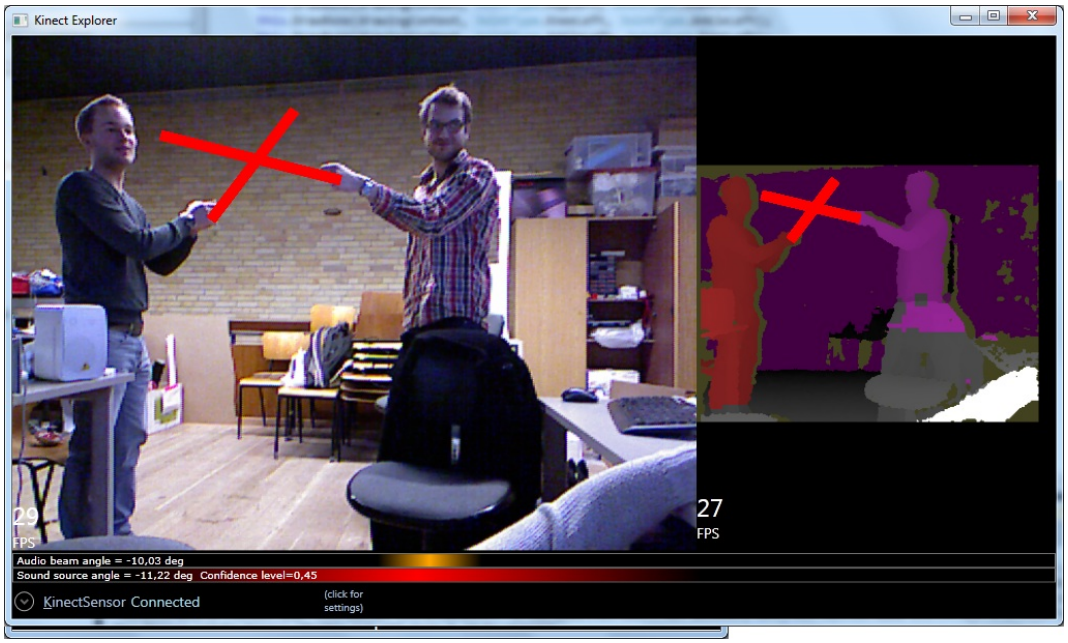

Figure 3.5: *Laserschwert in RGB-Bild*

[1] Vergleich der Bildschirmausgaben der Demonstrationsprogramme ohne analytische Auswertung.
[2] Augmented Reality ist eine Kombination aus wahrgenommener und vom Computer erzeugter Realität

Verarbeitung der SHORE-Daten

Die Erfassung der Daten erfolgt wie im Vorsemester über die Bibliothek des Frauenhofer Instituts. Diese funktioniert tadellos und stellt eine solide Datenbasis her. Eine Schwierigkeit ist jedoch die Einordnung der ermittelten Emotionen, da diese stets stark differenziert ausgeprägt und zudem kontextabhängig sind. Herr Becker-Asano beschäftigt sich in seiner Doktorarbeit ausführlich mit diesem Thema und stieß auf den gleichen Sachverhalt (vgl. Becker-Asano, 2008, Seite 23 bis 31) .

Folgende Grafik zeigt diese Kontextabhängigkeit von Emotionen anschaulich. Während auf dem linken Bild der Herr Wut vermittelt, so ist die Emotion des rechten Bildes - wohlgemerkt, dass es sich um das exakt gleiche Gesicht handelt - eher dem Ekel zuzuordnen. Mittels einer Software, welche ausschließlich Farbbildinformationen eines Gesichts analysiert, ist dieser Unterschied nicht ermittelbar.

Figure 3.6: *Ein Gesicht - Zwei Bedeutungen. (vgl. el Kaliouby R. Picard R.W. Hoque, 2009)*

Die Weiterverarbeitung der einfach einzuordnenden Werte, wie das erkannte Alter oder Geschlecht, funktioniert dank dem ActiveMQ-Wrapper in C# einwandfei. In folgendem Beispiel erfolgt eine Einfärbung des ”Laserschwertes” in Abhängigkeit des erkannten Geschlechts:

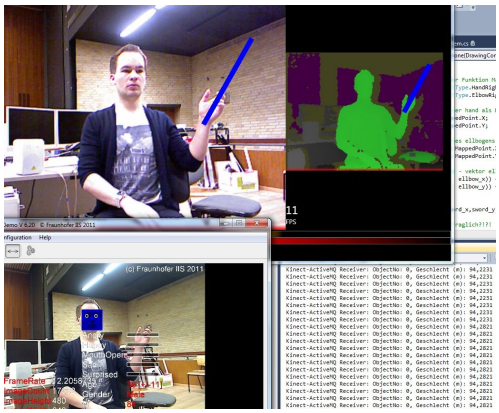

Figure 3.7: *Laserschwertfärbung Mann*

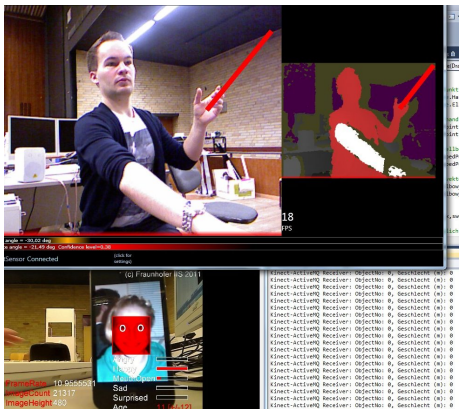

Figure 3.8: *Laserschwertfärbung Frau*

3.3 Hardwarenahe Nutzung der Informationen

Ein ursprüngliches Ziel des Masterprojektes war es, die RGBD-Daten[3] der Microsoft-Kinect mit Hilfe einer geeigneten Hardware-Plattform, ohne den Umweg über ein PC-System, auszulesen. Aufgrund der umfangreichen Funktionen des Microsoft-Kinect-SDKs und dem geringen, zu erwartenden Erfolg einer Eigenentwicklung wurde dieses Ziel jedoch aufgegeben. Anstelle dessen wurde das Ziel ausformuliert, die über ein PC-System ermittelten RGBD-Informationen soweit vorzuverarbeiten und zu reduzieren, dass diese von einfacher, externer Hardware verwendet werden können. Bei bisher genutzter Hardware waren die Möglichkeiten der Nutzung aber schnell druch Übertragungsbandbreiten und Prozessorgeschwindigkeit beschränkt.

Das **FEZ-Spyder Board** (vgl. Electronics, 2012) stellt eine interessante Möglichkeit dar, diese Flaschenhälse zu entfernen. Auf Seiten der Übertragungsgeschwindigkeit besitzt diese Plattform neben einer Onboard Ethernet-Schnittstelle auch eine USB-Verbindung, was eine sehr zügige Übertragung jeder Art von Daten ermöglicht. Mit einem 72 MHz 32Bit-Prozessor verfügt das FEZ-Board eine deutlich größere Rechenleistung als beispielsweise die Arduino-Plattform mit 16 MHz. Dies erlaubt auch komplexere Informationen in Echtzeit zu verarbeiten.

Besonders interessant ist auch die Tatsache, dass das FEZ-Spyder Board .NET Gadgeteer (vgl. Microsoft, 2012) kompatibel ist. Dies bedeutet, dass sich das Board mit einer C#-Entwicklungsumgebung programmieren lässt und damit auf der softwareseitig selben Umgebung, wie die anderen Komponenten dieses Projekts basieren. In der Masterarbeit wäre es interessant, eine auf diesem Board basierende Anwendung zu erstellen und die Nutzbarkeit zu untersuchen.

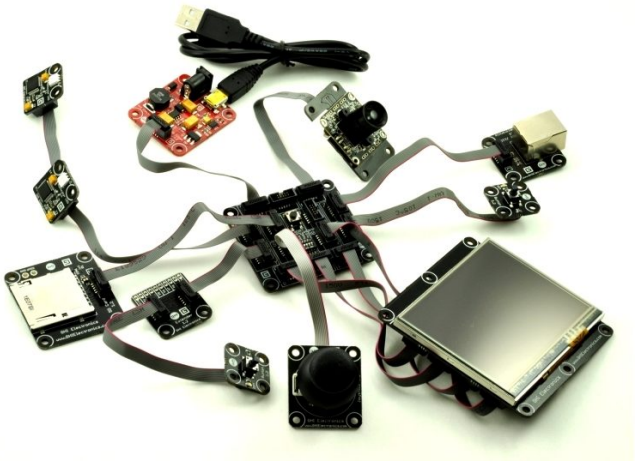

Figure 3.9: *FEZ-Spider (vgl. Electronics, 2012)*

[3] RGBD - Red Green Blue Depth, gebündelte Farb- und Tiefenbildinformationen

4 Zusammenfassung und Ausblick

Durch die im Projekt 2 ausgearbeiteten Techniken können alle relevanten Informationen in einer (C#-) Entwicklungsumgebung gebündelt und zur Verarbeitung bereitgestellt werden. Dies ermöglicht es nun, darauf aufbauend Software zu entwickeln, welche diese zusammengefassten Informationen verarbeitet.

Im Folgenden muss eine sinnvolle Einordnung und Reduzierung der gesammelten Daten erfolgen. Dies ist bei simpel quantisierbaren Messgrößen wie Entfernung zweier Punkte oder dem erkannten Alter einer Person problemlos möglich. Komplexität gewinnt diese Verarbeitung bei der Einordnung von Emotionen, wie auch das Beispiel auf Seite 11 veranschaulicht.

Durch das vorgestellte Entwicklungsboard ”FEZ Spyder” rückt das ursprüngliche Ziel der Arbeit - der Verarbeitung der Kinect-RGBD-Daten mit einer einfachen Hardwareplattform - wieder in greifbare Nähe. Es wird zwar der Weg über ein PC-System mit der Microsoft-Entwicklungsumgebung gegangen werden müssen, jedoch ist mit dem FEZ-Board auch die Weiterverarbeitung größerer Informationsmengen hardwarenah möglich. Als Vorteil dieser dualen Entwicklung lässt sich jedoch nennen, dass diese nun auch die Verwendung der Informationen aus der SHORE-Bibliothek ermöglicht, was ursprünglich nicht angedacht war.

Das Ziel der Masterarbeit ist es nun, die gesammelten Informationen aufzubereiten und in eine Anwendung zu überführen, welche dem Anwender einen Nutzen bereitstellt. Hier gilt es noch ein geeignetes Szenario auszuarbeiten. Die Überprüfung des möglichen Mehrwerts dieser Anwendung gegenüber Systemen und Lösungen, welcher weniger detaillierte Informationen zur Verfügung stehen, ist ein weiteres Teilziel der Masterarbeit.

Bibliography

[Becker-Asano 2008] BECKER-ASANO: *WASABI: Affect Simulation for Agents with Believable Interactivity*. 2008

[Electronics 2012] ELECTRONICS, GHI: *FEZ Spider Starter Kit*. 2012. – URL http://www.ghielectronics.com/catalog/product/297. – Zugriffsdatum: 02.02.2012

[Foundation 2012] FOUNDATION, Apache S.: *ActiveMQ*. 2012. – URL http://activemq.apache.org/. – Zugriffsdatum: 14.02.2012

[GI-Jahrestagung 2011] GI-JAHRESTAGUNG: *Companion-Systeme und Mensch-Companion-Interaktion*. 2011. – URL http://edu.cs.uni-magdeburg.de/EC/konferenzen-und-workshops. – Zugriffsdatum: 14.02.2012

[IIS 2011] IIS, Fraunhofer: *Fraunhofer IIS Website*. 2011. – URL http://www.iis.fraunhofer.de/bf/bv/ks/gpe/. – Zugriffsdatum: 02.07.2011

[el Kaliouby R. Picard R.W. Hoque 2009] KALIOUBY R. PICARD R.W. HOQUE, M. E. el: *When human coders (and machines) disagree on the meaning of facial affect in spontaneous videos.* 2009

[Kriegsmann 2011] KRIEGSMANN, Edo: *Projektarbeitung 1 - 3D Interaktionen in Smart Homes*. 2011. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2011-proj1/kriegsmann.pdf. – Zugriffsdatum: 18.02.2012

[Microsoft 2011] MICROSOFT: *Microsoft Kinect SDK Programming Guide*. 2011. – URL http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/docs/ProgrammingGuide_KinectSDK.pdf. – Zugriffsdatum: 02.07.2011

[Microsoft 2012] MICROSOFT: *Microsoft .NET Gadgeteer*. 2012. – URL http://www.netmf.com/gadgeteer/. – Zugriffsdatum: 02.02.2012

[Weiser 1991] WEISER, Mark: *The Computer for the Twenty-First Century*. 1991

A Anhang

A.1 ”json”-Nachrichten erstellen und versenden

```
//RatingCount = Anzahl ausgewerteter Merkmale
unsigned long ratingCount = object->GetRatingCount();

for( unsigned long r = 0; r < ratingCount; ++r )
{
json  << "\t\""
//Merkmal (zB. Alter in Jahren)
<< object->GetRatingKey(r)
<< "\":\""
//Ausprägung (in Prozent oder Jahren)
<< *(object->GetRating(r))
<< "\"";
}
json << "}";
//Nachricht zum Active MQ senden:
SendMessage(json.str());
```

A.2 Codebeispiel ”Laserschwert”

```
//Joitnts-Variablen
JointMapping joint_hand;
JointMapping joint_elbow;

 //Die erforderlichen Koordinaten abrufen/mappen
jointMappings.TryGetValue(JointType.HandRight, out joint_hand);
jointMappings.TryGetValue(JointType.ElbowRight, out joint_elbow);

//Gemappte X- und Y-Koordinate der Hand
double hand_x = joint_hand.MappedPoint.X;
double hand_y = joint_hand.MappedPoint.Y;

//Gemappte X- und Y-Koordinate des Ellenbogens
double elbow_x = joint_elbow.MappedPoint.X;
double elbow_y = joint_elbow.MappedPoint.Y;

//Koordinaten des Laserschwert-Endes berechnen
double sword_x = (2 * (hand_x - elbow_x)) + hand_x;
double sword_y = (2 * (hand_y - elbow_y)) + hand_y;

//Laserschwertende als Point
 Point sword_end = new Point(sword_x,sword_y);

//Linie in rot mit 9er-Dicke zeichnen
 Pen drawPen_laser = new Pen(Brushes.Red, 9);

//Zeiche Linie von Hand zu Schwertende (schon im neuen System)
 drawingContext.DrawLine(drawPen_laser,
       joint_hand.MappedPoint, sword_end);
```

A.3 Codebeispiel ”ActiveMQ in C#”

```
using ActiveMQWrapper.Messaging;
using ActiveMQWrapper;
using Newtonsoft.Json;

ConnectionSettings settings = new ConnectionSettings();

settings.ActiveMQIp = "tcp://localhost:61616";
settings.TopicName = "LIVE.EMOTIONS.TOPIC";
try
{
    ActiveMQSubscriber subscriber = new ActiveMQSubscriber(settings);
    subscriber.MessageReceivedEvent += new EventHandler
          <MessageEventArgs>(subscriber_MessageReceivedEvent);
 }
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
 }

void subscriber_MessageReceivedEvent(object sender,
          MessageEventArgs e1)
{
    String json = e1.Message.Text;
    ExampleMessage message = JsonConvert.DeserializeObject
          <ExampleMessage>(json);
    Console.WriteLine("Kinect-ActiveMQ Receiver: ObjectNo: {0},
          Age: {1}", message.ObjectNo, message.Age);
}
```