Bericht für Projekt 1 im Masterstudiengang Informatik

Kollaborative Mixed-Reality-Anwendungen

Christian Blank

HAW Hamburg, Technik und Informatik,
Berliner Tor 7, Hamburg, Germany
christian.blank@hawhamburg.de
http://www.haw-hamburg.de

1 Motivation

Die bisherigen Interaktionsmöglichkeiten mittels Tastatur, Maus und zweidimensionalen Gesten sind zwar sehr gut für die Arbeit mit zweidimensionalen Inhalten geeignet, aber nicht für Inhalte, die auch in der dritten Dimension dargestellt werden. Neben der Arbeit mit virtuellen Inhalten können dreidimensionale Gesten auch für die Kommunikation oder die Arbeit mit realen Objekten verwendet werden (vgl. Mixed Reality und Computer Supported Collaborative Work[1]).

1.1 Ziel

Ziel der Arbeit in der Gruppe I^2E ist die Entwicklung einer Mixed-Reality-Anwendung zur kollaborativen Konstruktion von Produkten und der Evaluierung dieser Anwendung. Verschiedene Teilnehmer können dabei zusammen an einem Produkt arbeiten, über das Produkt diskutieren und sich austauschen, auch wenn sie räumlich von einander getrennt sind. Zur Steuerung werden Gesten und reale Objekte, die beispielsweise als Werkzeuge oder Bauteile dienen, eingesetzt.

Für dieses Ziel wurde in Projekt 1 ein prototypisches Objekttracking umgesetzt und ein ausführliches Konzept für eine Gestenerkennung erstellt. Für die Gestenerkennung wurden bereits einige wichtige Grundlagen implementiert.

1.2 Aufbau

In Abschnitt 2 wird ein Konzept für die Gestenerkennung und das Objekttracking vorgestellt und es werden die Anforderungen aufgelistet. In Abschnitt 3 werden die in Projekt 1 entstandenen Implementierungen beschrieben und in Abschnitt 4 werden die Ergebnisse zusammengefasst. Zudem werden ein Ausblick auf Projekt 2 gegeben und Risiken aufgezeigt.

2 Konzept

2.1 Gestenerkennung

Die Gestenerkennung besteht aus zwei Komponenten und benutzt für die Datenaggregation eine Abstraktionsschicht, wie in Abbildung 1 dargestellt. Die Abstraktion stellt ein uniformes Skelettmodell unabhängig von der genutzten Skeletterkennung zur Verfügung. Weitere Informationen zu Trame befinden sich in Abschnitt 3.3. Der Controller und die Streampipelines bilden zusammen die Gestenerkennung. Der Controller nimmt die Skelette von Trame entgegen. Er ist darüber hinaus für die Erstellung von Jobs und die Verwaltung der Pipelines verantwortlich. Die Pipelines verarbeiten die Skelettstreams, indem sie Daten filtern und die Eingabe mit Gestentemplates vergleichen. Für die Möglichkeit, Eingaben verschiedener Nutzer zu erkennen, werden die Skelette einzelner Nutzer in separaten Pipelines verarbeitet.

[1] CSCW beschäftigt sich mit der rechnergestützten Zusammenarbeit bei zeitlichen oder räumlichen Trennung

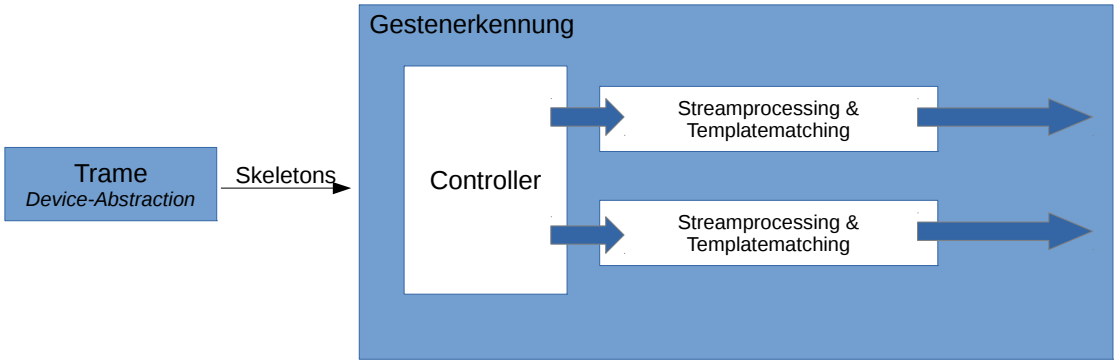

Abb. 1. Schematische Darstellung der Komponenten der Gestenerkennung und der Abstraktionsschicht Trame (Quelle: eigene Arbeit).

2.1.1 Anforderung und Abgrenzung

Die Gestenerkennung besitzt verschiedene Anforderungen, die in dem Konzept berücksichtigt wurden. Sie werden nachfolgend aufgelistet. Die Reihenfolge gibt dabei nicht Auskunft über die Priorität.

1. Ikonische Gesten können erkannt werden.
2. Zeigegesten können erkannt werden.
3. Es können gleichzeitig mehrere Nutzereingaben verarbeitet werden.
4. Die Erkennung erfolgt unabhängig von einem verwendeten Eingabegerät, sodass nicht auf ein bestimmtes Device zurück gegriffen werden muss.
5. Bei Nichterkennen einer Geste können Bewegungsdaten weitergegeben werden.
6. Die Gestenerkennung kann entscheiden, ob eine Eingabe von einem Nutzer beabsichtigt ist.
7. Die Gestenerkennung ist nicht für eine Personenerkennung verantwortlich.

Details zur Implementierung befinden sich in Abschnitt 3.2.

2.2 Objekttracking

Physikalische Objekte sollen zusammen mit virtuellen Objekten interagieren können. Dazu müssen die physikalischen Objekte von dem Computersystem erkannt und identifiziert werden.

2.2.1 Anforderungen

1. Physikalische Objekte können im Raum erkannt werden. Der Raum ist dabei auf einen kleinen Bereich vor einer Kamera begrenzt.
2. Die physikalischen Objekte können von dem Objekttracking identifiziert und verfolgt werden.
3. Das Verschwinden von Objekten und das Auftauchen von neuen Objekten kann erkannt werden.
4. Die Ausrichtung und die Position der Objekte relativ zur Kamera können ermittelt werden.

In Abschnitt 3.1 wird eine Implementierung vorgestellt, die die Anforderungen erfüllen.

2.3 Prototyp

Die beiden zuvor beschriebenen Konzepte sind Komponenten eines Prototypen, wie er in Abbildung 2 gezeigt wird. Die einzelnen Komponenten sprechen dabei über eine Middleware miteinander. Neben der Gestenerkennung und dem Objekttracking gehören der Viewer, die Applikationslogik und ein Eventlogger zu den Komponenten des Prototypen.

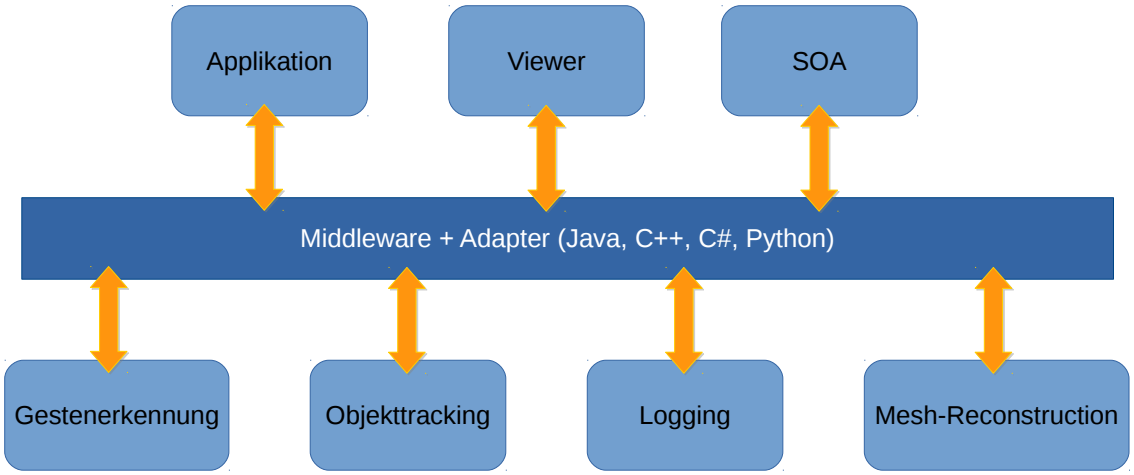

Abb. 2. Schematische Darstellung der Basiskomponenten des Prototypen (Quelle: eigene Arbeit).

Applikation/Rendering. Die Applikation und das Rendering werden durch eine Komponente übernommen, die in Unity geschrieben ist. Unity setzt auf eine Mono-Version, die eine Neuimplementierung von .Net 3.5 darstellt. Die Komponente enthält die gesamte Programmlogik und dient im Prototypen als Hauptkomponente. Alle Daten der Sensorik werden an sie geliefert, von ihr ausgewertet und dann über den Viewer ausgegeben.

Viewer. Die gerenderten Bilder können wahlweise über einen Monitor im Frontbereich des Tisches oder durch eine See-Through-Brille dargestellt werden. Für die Darstellung über eine Brille müssen die Position und die Ausrichtung des Kopfes des Betrachters bestimmt und an den Applikationsservice übertragen werden.

Objekttracking. Objekte werden mithilfe von Markern erkannt. Die Position und Ausrichtung der Marker wird genutzt, um die Objekte im Raum zu bewegen und diese Bewegung am Computer sichtbar zu machen. Hat die Komponente ein Objekt identifiziert werden alle Komponenten, die sich auf dieses Event eingeschrieben haben, benachrichtigt. Bei Änderungen der Position und der Rotation, sowie beim Entfernen eines realen Objektes werden ebenfalls Events ausgelöst.

Gestenerkennung. Die Gestenerkennung verwendet eine Kinect for Windows und das Kinect SDK von Microsoft. Mithilfe des SDKs können User getrackt werden. Wenn eine Geste erkannt wird, dann werden über einen Publish/Subscribe-Mechanismus alle Subscriber informiert. Die Interpretation der Geste erfolgt dann erst innerhalb der Applikation.

Middleware. Alle Komponenten kommunizieren über die Middleware miteinander. Sie kümmert sich um die transparente Kommunikation zwischen verschiedenen Komponenten und anderen Remote-Systemen. Malte Eckhoff ist für die Middleware verantwortlich und hat sie in seiner Projekt 1-Ausarbeitung detailiert beschrieben.

2.3.1 Anforderungen

Der Prototyp besitzt eine Reihe von Anforderungen. Aufgrund des großen Umfanges des gesamten Projektes und der terminlichen Einschränkungen durch die Fächer Projekt 1 und Projekt 2, wurde festgelegt, dass in Projekt 1 nicht alle Anforderungen umgesetzt werden können.

1. Darstellung von virtuellen Objekten auf einer See-Through-Brille.
2. Anpassung der Wiedergabe an die Blickrichtung und Position des Anwenders.
3. Überdeckung von realen Objekten durch physikalische Objekte.
4. Mehrere Anwender können gleichzeitig an einem Tisch mit einander arbeiten.
5. Mehrere Anwender können an verschiedenen Tischen mit einander arbeiten.
6. Einfache Erweiterbarkeit und Austausch von bestehenden Komponenten.

2.4 Exemplarische Anwendungen

In diesem Abschnitt werden verschiedene Ideen für Anwendungen, die prototypisch umgesetzt werden sollen, aufgelistet. Diese Anwendungen sollen alle mit einem minimalen Umfang an Komponenten realisiert werden können. Mit Fortschritt des Projektes erweitern sich auch die Anwendungsmöglichkeiten des Prototypen.

2.4.1 3D-Image-Viewer

Eine Anwendung bestehend aus den Komponenten Gestenerkennung und 3D-Wiedergabe, die zur Darstellung von einzelnen 3D-Modellen dient. Dabei kann ein Nutzer mithilfe von Gesten zwischen verschiedenen Modellen wechseln und diese skalieren und rotieren. Das Modell befindet sich mit seinem Nullpunkt auf dem Mittelpunkt der Tischplatte. Der Nutzer kann sich in einem gewissen Rahmen um das virtuelle Modell bewegen.

2.4.2 Baukasten

Eine weitere Möglichkeit ist die Umsetzung eines Baukastens. Zusätzlich zu den Komponenten Gestenerkennung und 3D-Wiedergabe wird ein Objekttracking genutzt, um Würfel zu verfolgen und mit in die virtuelle Szene zu integrieren. Mit diesen Objekten (Bausteinen) können dann größere, virtuelle Objekte (Gebilde) gebaut werden, indem Kopien von den einzelnen virtuellen Bausteinen aneinander gefügt werden. Die Positionierung der Bausteine erfolgt durch das Objekttracking. Das Anfügen an das virtuelle Gebilde geschieht durch eine Geste.

2.4.3 Murmelbahn

Eine Erweiterung der Baukasten-Applikation ist eine Murmelbahn. Hier sollen anstelle von virtuellen Bausteinen 3D-Modelle von Murmelbahnelementen verwendet werden. Diese können dann ebenso wie die Bausteine aneinander gefügt werden und ergeben somit eine komplette Bahn. Zusätzlich können noch eine oder mehrere Murmelquellen festgelegt werden. Diese produzieren in gewissen Zeitabständen neue virtuelle Murmeln und geben sie in die Bahn. Die Murmeln verhalten sich dann innerhalb der Bahn physikalisch korrekt und rollen den vorgegebenen Weg hinab. Durch Hindernisse, Weichen etc. kann ihre Bewegung beeinflusst werden.

3 Implementierungen

Es werden nun alle Implementierungen vorgestellt, die im Laufe von Projekt 1 erstellt worden sind. Es wird darauf verzichtet, zu viele Details über die exakte Umsetzung wiederzugeben und im einzelnen auf die jeweilige Dokumentation der Software verwiesen.

3.1 Objekttracking

Das Objekttracking ist ein wichtiger Teil der Benutzerschnittstelle des Prototypens. Um das Arbeiten in einer Mixed Reality zu ermöglichen, müssen die realen Objekte durch den Computer erkannt werden. Dabei sind Position, Rotation und Identität in Erfahrung zu bringen. Im Rahmen von Projekt 1 musste eine Lösung gefunden werden, die die gestellten Anforderungen erfüllt (Abschnitt 2.2.1) und dennoch möglichst schnell umgesetzt werden konnte. Die Wahl fiel auf ein markerbasiertes Tracking von Holzwürfeln. Mithilfe der Marker können die einzelnen Objekte identifiziert und ihre Ausrichtung grob bestimmt werden. Eine genaue Bestimmung der Position und der Ausrichtung wird mithilfe des POSIT-Verfahrens aus [DD95] ermöglicht (coplanar POSIT).

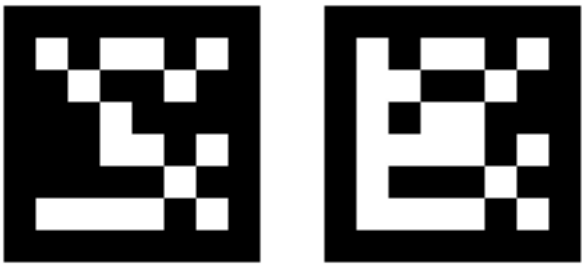

Abb. 3. Beispiele für Marker, die von dem Objekttracking verwendet werden (Quelle: `http://www.aforgenet.com/articles/glyph_recognition/glyphs.png`)

Jeder der verwendeten Marker (dargestellt in Abbildung 3) ist eindeutig und rotatationsinvariant, also gibt es keine zwei Marker, die durch eine beliebige Rotation identisch werden, außer sie waren bereits zu Anfang identisch. Ein Objekt besitzt sechs Marker. Für POSIT werden nicht die Identitäten der Marker, sondern die Bildkoordinaten ihrer Eckpunkte benötigt. Die Identifizierung der Marker ist jedoch für die Identifizierung der Objekte nötig.

Für die Verfolgung wird kein iteratives Verfahren eingesetzt, sondern jedes Bild wird für sich verarbeitet und die Ergebnisse werden mit vorherigen Werten verglichen. Bei Änderungen, die einen Schwellenwert überschreiten, werden die entsprechenden Events ausgelöst. Ein iteratives Verfahren würde die Position des Markers im vorherigen Bild in der Betrachtung des nächsten Bildes mit einbeziehen und somit die Suchzeit möglicherweise verkürzen. Die Erkennung neuer Elemente ist bei solchen Verfahren jedoch vergleichsweise schwierig umzusetzen.

3.2 Gestenerkennung

Neben dem Objekttracking ist die Gestenerkennung die zweite Komponente zur Nutzereingabe, die für den Prototypen entwickelt wird.

Für die geforderte Unabhängigkeit von einzelnen Eingabegeräten, wie etwa einer Microsoft Kinect for Windows, wird eine Abstraktionsschicht verwendet, die Skelette aus verschiedenen Devices erzeugt. Dabei spielt es keine Rolle, ob die Devices bildbasiert arbeiten oder am Körper getragen werden, solange sie Informationen zu den Gelenkpositionen des Nutzers geben. Die Ideen und die Implementierung von Trame wird in Abschnitt 3.3 beschrieben.

Die Skelette, die durch die Abstraktionsschicht erzeugt werden, werden zunächst in einem Controller in der Gestenerkennung klassifiziert. Es wird geprüft, ob ein valides Skelett vorliegt und welchem Nutzer das Skelett zugeordnet ist. Dementsprechend wird es in eine der Pipelines gelegt und durch diese weiterverarbeitet.

In den Pipelines werden die Skelette weiter verarbeitet. Es wird über die Bewegungen ein Mittelwert gebildet, damit Messfehler ausgeglichen werden können. Anschließend wird über eine Input-Zone festgestellt, ob der Nutzer eine Eingabe machen möchte. Wenn das zutrifft, dann werden die Bewegungen der Hand mit zuvor festgelegten Templates verglichen. Grundlage für die Input-Zone und den Vergleich zwischen Hand und Templates ist die Arbeit [KNQ12].

Sollte keine passende Geste für eine Eingabe gefunden werden, dann können die Bewegungsdaten des Skelettes analysiert und einer Anwendung bei Bedarf zur Verfügung gestellt werden. So können dann beispielsweise Collider erzeugt werden, die virtuelle Objekte bei Zusammenstoß bewegen.

3.3 Trame

Ein wichtiger Bestandteil der Gestenerkennung sind verlässliche Sensordaten. Die Vergangenheit hat gezeigt, dass es innerhalb von kurzer Zeit zu großen Änderungen kommen kann. So ist beispielsweise das Opensource-Projekt OpenNI von PrimeSense nicht mehr direkt verfügbar und auch die Erweiterung Nite zur Erkkenung von Skeletten kann nicht mehr verwendet werden. Bei der LEAP Motion wurden die API mit einer neuen Version des SDKs verändert. Diese Entwicklung macht deutlich, dass eine Gestenerkennung von den Sensordaten abstrahiert

werden sollte. An diesem Punkt setzt *Trame* an. Es ist eine Abstraktionsschicht für Sensordaten von verschiedenen Sensoren, die die räumliche Position von Skelettdaten ermitteln. Dabei ist es egal, ob die Sensoren am Körper getragen werden oder die Daten durch ein Kamerasystem ermittelt werden.

3.3.1 Skelettmodell

Trame stellt ein Skelett zur Verfügung, dass neben ID und Zeitstempel auch Informationen zur Validität eines Skelettes bietet. Ein Skelett, wie in Abbildung 4 dargestellt, besteht aus mindestens 18 Gelenken die in einer Baumstruktur angeordnet sind. Das Root-Gelenk befindet sich dabei im Körperschwerpunkt und besitzt eine Normale. Die Koordinaten dieses Gelenkes sind als einziges im Baum absolut. Alle anderen Koordinaten sind relativ zu ihrem Eltern-Gelenk. Ein Gelenk kann ein oder kein Elterngelenk und kein oder mehrere Kindgelenke besitzen. Viele der Gelenke besitzen genau ein Kindgelenk, wie etwa die Knie oder die Schultern.

In Listing 1.1 wird ein kleiner Ausschnitt (nur drei Gelenke) aus einem vollständigen Skelett gezeigt. Ein wichtiger Wert ist neben den Normalen und Punkten der Typ eines Gelenks. Dieser Integerwert muss für jedes Gelenk in einem Skelett eindeutig sein, denn er wird bei der Suche im Skelett als Schlüssel verwendet.

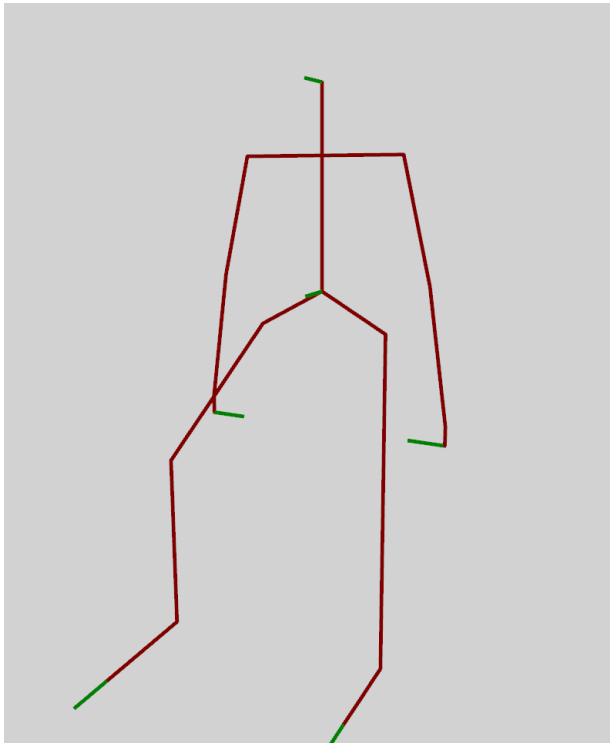

Abb. 4. Eine Visualisierung der Skelettdaten durch den in Trame enthaltenen skeleton-viewer (Quelle: eigene Arbeit).

3.3.2 Aufbau und Funktionsweise

Trame besteht aus den zwei Komponenten Device und Skeleton, die im Folgenden näher beschrieben werden.

Device. Die wichtigste Komponente von Trame ist Device. In ihr werden alle Sensoren, die in Trame verwendet werden können, abstrahiert und die erhaltenen Daten werden aufbereitet und zu Skeletten zusammengefügt. Um möglichst viele Sensoren und SDKs zu unterstützen, werden Teilaufgaben in Adaptern ausgelagert. So wird bspw. ein Skelett bei der Verwendung einer LEAP Motion aus einem Standardskelett und den beiden Händen zusammengesetzt. Die Hände werden über den Adapter generiert, da sich die Erstellung in den unterschiedlichen Versionen der SDKs geändert hat.

Skeleton. Die Skeleton-Komponente umfasst Funktionalitäten zum Erstellen, Bearbeiten und Vergleichen von Skeletten und das Skelettmodell selbst.

```
{
  "id": 1,
  "root": {
    "children": [
      {
        "children": [{
          "children": [...],
          "normal": [0, 0, 10],
          "point": [0, 180, 0], // in centimeters relative to parent
          "type": 1
        }],
        "normal": null,
        "point": [0, 350, 0],
        "type": 5 // type of joint
      },...
    ],
    "normal": [0, 0, 10],
    "point": [0, 1100, 0], // absolute coordinates
    "type": 9
  },
  "timestamp": 4215765565 // unix timestamp of creation
}
```

Listing 1.1. Ausschnitt aus serialisiertem Skelett in JSON-Format

Trame ist als Bibliothek konzipiert und gut für die Verwendung als eigenständigen Service geeignet. In Listing 1.2 wird gezeigt, wie ein valides und als Byte-Array serialisiertes Skelett geladen werden kann. Die Voreinstellung für den Sensor ist LEAP Motion SDK 1 und für die Serialisierung JSON. In dem Listing wurde die Serialisierung auf Protobuf geändert. In einer neueren Version ist die Serialisierung ausgelagert und kann über eine selbstständige Bibliothek eingebunden werden.

In dem Repository von Trame befindet sich auch ein Skript zum Anzeigen von Skeletten. Somit können Ergebnisse einfach visuell überprüft werden (zu sehen in Abbildung 4). Weitere Informationen dazu befinden sich in Abschnitt 3.4.2.

```
#include <iostream>
#include "trame.hpp"
#include <chrono>
#include <thread>

using namespace std;

int main(int argc, char* argv[])
{
  trame::trame t;
  // set output type
  t.set_output(output_type::PROTOBUF);

  // only return a valid skeleton
  while(!t.get_skeleton().valid) {
    this_thread::sleep_for(std::chrono::milliseconds(30));
  }

  for (auto& c : t.get_serialized_skeleton()) {
    cout << c;
  }
```

```
22
23   return 0;
24 }
```

Listing 1.2. Initialisierung von Trame und Änderung des Serialisierers zu Protobuf. Anschließende Ausgabe des Skelettes auf der Konsole.

3.3.3 Features

Trame besitzt eine Reihe von Features, die die Arbeit mit Skeletten im Kontext von Gestenerkennung vereinfachen. Die wichtigsten werden nachfolgend aufgelistet und kurz erläutert.

Unterstützung verschiedener Sensoren
Trame unterstützt verschiedene Sensoren und SDKs, etwa Microsoft Kinect for Windows, LEAP Motion, Microsoft Kinect 2 und PrimeSense Carmine.

Einfache Erweiterbarkeit
Die Unterstützung von zukünftigen Versionen von Sensoren können ebenso schnell integriert werden, wie die neue Sensorik (z.B. das Myo-Armband von Thalmic oder YEI Technologies PrioVR). Dafür müssen in der Device-Komponente entweder ein neuer Adapter geschrieben werden, wenn bereits ein Sensor mit einer ähnlichen Funktionsweise integriert ist oder es wird ein neues Device erstellt und in der Komponente registriert.

Nutzung mehrerer Sensoren zur Auswertung
Für eine genauere Erkennung oder einen erweiterten Aktionsbereich können mehrere Sensoren zusammen betrieben werden.

Schnelle Verarbeitung
Das System ist auf eine schnelle Verarbeitung ausgelegt und kann im Idealfall bis zu 9000 Skelette pro Sekunde erstellen. Ein wichtiger Faktor ist der interne Aufbau des Skelettes, der die Verarbeitung sehr effizient macht.

Generierung von Standardskeletten
Sensoren liefern nicht immer ein korrektes Ergebnis, sondern zum Teil invalide Daten. Trame erkennt solche Fehler und führt das Ergebnis in einen Initialwert zurück, sodass ein Programm, dass Trame verwendet, sich nicht zwangsläufig um die Validierung kümmern muss.

3.4 Helper

3.4.1 protobuf-guy

Dieses Tool[2] ist ein Konsolenprogramm zum automatischen Generieren von Sourcecode aus bestehenden Protobuf-Definitionen. Es werden dabei Dateien für C++, C#, Java und Python erstellt. Zusätzlich zu der Übersetzung wird eine Datei angelegt, die eine Liste aller definierten Messages enthält. Dabei werden auch tief verschachtelte Messages erkannt. Jeder Message wird ein eindeutiger Integer-Wert zugeordnet. Mithilfe der erzeugten Datei können später die versendeten Nachrichten ohne großen Aufwand identifiziert werden. *protobuf-guy* wurde in Ruby geschrieben und ist somit weitestgehend plattformunabhängig.

3.4.2 skeleton-viewer

Der Skeleton-Viewer ist ein in Python geschriebenes Skript zum Betrachten von Skeletten. Es befindet sich in dem Repository von Trame. Das Tool kann Daten aus einer Datei oder *stdin* einlesen. Das Skelett muss in serialisierter Form vorliegen (JSON oder protobuf). Während der Betrachtung wird das Skelett-Modell um die Z-Achse rotiert. Die Rotation kann angehalten und das Modell kann selbstständig rotiert werden.

4 Fazit

Der vorliegende Projektbericht zeigt die Arbeit, die in dem Projekt während des Sommersemesters 2014 im Projekt 1 geleistet wurde. Dabei wird deutlich, dass innerhalb eines Semesters viele Probleme gelöst wurden, aber auch, dass es noch viele Teilbereiche gibt, an denen weiter intensiv gearbeitet werden muss. In diesem Abschnitt wird zunächst ein Überblick über den aktuellen Stand von Projekt 1 gegeben und anschließend werden die Ziele für Projekt 2, dass im kommenden Semester statt findet, bestimmt. Abschließend werden mögliche Risiken identifiziert.

[2] Der Quellcode ist auf GitHub zu finden: `https://github.com/1blankz7/protobuf-guy`

4.1 Stand

In diesem Abschnitt wird aufgezeigt welchen Stand die einzelnen Komponenten nach Abschluss von Projekt 1 haben.

4.1.1 Middleware und SOA-Komponente

Für die Middleware wurde ein Konzept und eine Architektur entwickelt und der Netzwerkadapter wurde für C# vollständig und für C++ teilweise umgesetzt. Die SOA-Komponente ist nur zu einem kleinen Teil implementiert worden, da eine Verteilung im ersten Prototypen noch nicht vorgesehen ist.

Im Gegensatz zu der eigentlichen Architektur der Middleware wird in dem ersten Prototypen eine vereinfachte Version verwendet. Die Services an sich bleiben bestehen, jedoch findet die Kommunikation nicht mehr ausschließlich über Messages statt, sondern auch durch die Verwendung von nur einer Programmiersprache (C#) und der Nutzung von dynamischen Bibliotheken, gegen die dann gelinkt werden kann. Somit wird Das Message Passing zunächst durch einen Shared Memory ersetzt.

Als wichtigster Grund für diese Entscheidung ist der vorhandene Zeitdruck bei der Fertigstellung des Prototypen zu nennen. Die Aufteilung der Komponenten hat sich nicht geändert. Die genannten Komponenten werden von Iwer Peters und Malte Eckhoff entwickelt. Für Informationen wird an dieser Stelle auf die entsprechenden Projektberichte verwiesen.

4.1.2 Objekttracking

Das Objekttracking ist vollständig implementiert worden und wird in Projekt 2 nur noch bei Bedarf angepasst. Es muss noch geklärt werden, ob ein Filter eingebaut werden soll, der zu große Sprünge bei fehlerhafter Erkennung ausgleicht.

4.1.3 Gestenerkennung

Es wurde in Projekt 1 ein Konzept für eine Gestenerkennung und eine erste Anforderungsanalyse aufgestellt. Die Gestenerkennung wurde in Projekt 1 nicht implementiert, aber es wurden Vorarbeiten geleistet, die für eine erfolgreiche Durchführung von Projekt 2 und der Masterarbeit sehr wichtig sind.

4.1.4 Trame

Trame ist in der Entwicklung weit fortgeschritten und unterstützt bereits verschiedene Sensoren (Leap Motion, Microsoft Kinect for Windows, Primesense Carmine). Neben einer Implementierung in C++ wurde bereits mit einer C#-Implementierung begonnen.

4.1.5 Tools

Es wurden verschiedene Tools zur Unterstützung entwickelt, etwa der *skeleton-viewer* zur visuellen Kontrolle von Skeletten oder der *protobuf-guy* zum schnellen Umwandlung von Protobuf-Definitionen in Quellcode. Beide Tools bieten noch nicht ihren vollen Funktionsumfang. So gibt es bei der Verwendung von *protobuf-guy* unter Windows Probleme mit der Pfadangabe, die noch behoben werden müssen. Der *skeleton-viewer* lädt ein Modell nur zu Beginn und kann es danach nicht mehr updaten.

4.1.6 Anwendungs-Prototyp

Die Anwendung in der man eine Murmelbahn mithilfe von verschiedenen Bausteinen aufbauen kann, ist umgesetzt und die Objekterkennung soweit vorangetrieben, dass sie Objekte durch Marker erkennt und ihre Position und Ausrichtung ermitteln kann. Die Gestenerkennung wird noch nicht verwendet.

4.2 Ausblick

In Projekt 2 kann auf dem Erreichten aus Projekt 1 aufgebaut werden und es kann im Wintersemester 2014 zum ersten Mal auf einen funktionsfähigen Prototypen zurückgegriffen werden, um neue Umsetzungen zu testen. Es gibt mehrere wichtige Punkte, die im Projekt 2 umgesetzt werden müssen.

Für die Objekterkennung sollen nur sporadisch Verbesserungen umgesetzt werden, wenn diese notwendig werden. Eine Auslagerung in einen Service ist aber angedacht, sobald die SOA-Komponente vollständig umgesetzt wurde.

Die Gestenerkennung wird die meiste Zeit der Implementierung benötigen und soll zum Ende von Projekt 2 mit der Kernfunktionalität fertig gestellt sein (vgl. 3.2). Sie wird in C# implementiert werden. Die genaue Definition der Templates muss erarbeitet und anschließend erstellt werden.

Die Umsetzung von Trame in C# muss fertig gestellt werden und es müssen Vergleiche mit der C++-Version durchgeführt werden. In Zukunft soll die Verwendung von mehreren Sensoren zur selben Zeit integriert werden. Auch eine Identifizierung der Skelette und eine Glättung der Sensorfehler wären sinnvolle Erweiterungen. Eine Identifizierung wird aber nicht von allen Sensoren unterstützt und könnte somit nicht immer zur Verfügung gestellt werden.

Fehler, die bisher in den erstellten Tools auftreten, sollten sukzessive behoben werden. Zudem benötigt der skeleton-viewer eine Möglichkeit, das Modell, das von ihm zu Beginn geladen wird, zu aktualisieren.

4.3 Risiken

In Zusammenhang mit einem so umfangreichen Projekt können verschiedene Probleme auftreten. Die entwickelten Komponenten besitzen untereinander Abhängigkeiten und es kann zu Problemen mit externen Projekten, etwa den SDKs von Sensor-Herstellern kommen, wenn diese beispielsweise ihre API stark ändern oder überhaupt kein SDK mehr zur Verfügung stellen.

4.3.1 Abhängigkeiten

Einige der eigenen Entwicklungen sind abhängig von dem Stand der Entwicklungen von anderen Teilprojekten. So werden die SOA- und die Netwerk-Komponente nicht selbst entwickelt, sondern von Malte Eckhoff und Iwer Petersen zur Verfügung gestellt. Auf die Entwicklung dieser Komponenten kann kein direkter Einfluss genommen werden.

Die Brillen, die für den Prototypen verwendet werden sollen, müssen noch bestellt werden. Es ist nicht sichergestellt, dass sie pünktlich geliefert werden, um alle Tests mit ihnen durchführen zu können.

Neben der Nutzung der Gestenerkennung im Kontext von Mixed Reality und der direkten Einbindung in den Prototypen wäre auch eine eigenständige Installation mit einer kleinen grafischen Demo-Anwendung denkbar, sollten nicht alle Komponenten, zu denen im Prototypen Abhängigkeiten bestehen, in der vorgesehenen Zeit zur Verfügung stehen.

4.3.2 Machbarkeit

Die Entwicklung einer Gestenerkennung in dem gedachten Umfang nimmt viel Zeit für Programmierung und zum Testen in Anspruch. Auf Basis der Erfahrungen aus Projekt 1 und den guten Fortschritten kann jedoch davon ausgegangen werden, dass ein großer Teil der Wünsche und vor allem die Kernfunktionalitäten der Gestenerkennung umgesetzt werden können.

Literatur

[DD95] DEMENTHON, Daniel F. ; DAVIS, Larry S.: Model-based object pose in 25 lines of code. In: *International journal of computer vision* 15 (1995), Nr. 1-2, S. 123–141

[KNQ12] KRISTENSSON, Per O. ; NICHOLSON, Thomas ; QUIGLEY, Aaron: Continuous Recognition of One-handed and Two-handed Gestures Using 3D Full-body Motion Tracking Sensors. In: *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*. New York, NY, USA : ACM, 2012 (IUI '12). – ISBN 978–1–4503–1048–2, 89–92

Abbildungsverzeichnis