|  |  |
| --- | --- |
|  | Kinect SA |
|  | Evaluation |

Felder werden durch InStep automatisch ausgefüllt. Bei manuellen Einträgen kann mit Doppelklick der Text selektiert und geändert werden Achtung! Dabei dürfen die Textmarken nicht gelöscht werden, da ansonsten die Verweise in der Fusszeile nicht mehr stimmen.

Änderungsnachweis

| Version | Status | Datum | Beschreibung | Autor |
| --- | --- | --- | --- | --- |
| 1.0 | Erstellung | 28.09.2012 |  | Renato Bosshart |

Inhaltsverzeichnis

[1. Einleitung 6](#_Toc343246143)

[1.1 Zweck 6](#_Toc343246144)

[1.2 Geltungsbereich 6](#_Toc343246145)

[1.3 Definitionen, Akronyme, Abkürzungen 6](#_Toc343246146)

[1.4 Literaturverzeichnis 6](#_Toc343246147)

[1.5 Übersicht 6](#_Toc343246148)

[2. Zieledefinition 7](#_Toc343246149)

[2.1 Evaluation 7](#_Toc343246150)

[2.2 Entwicklung 7](#_Toc343246151)

[2.3 Stabilisierung 7](#_Toc343246152)

[3. Wochenplanung 8](#_Toc343246153)

[4. Aufgabenbeschreibung 9](#_Toc343246154)

[4.1 Technische Produktevaluation 9](#_Toc343246155)

[4.2 Evaluation von verschiedenen Bedienkonzepten 9](#_Toc343246156)

[4.3 Analyse der Gestenabhängikeiten 9](#_Toc343246157)

[4.4 Implementierung Standardmodus 9](#_Toc343246158)

[4.4.1 Pinch-Zoom 9](#_Toc343246159)

[4.4.2 Blättern 9](#_Toc343246160)

[4.4.3 Anmelden 9](#_Toc343246161)

[4.4.4 Auswahlgesten 9](#_Toc343246162)

[4.5 Implementierung des Joystick-Modus 9](#_Toc343246163)

[4.5.1 Scrollen 10](#_Toc343246164)

[4.5.2 Cursor 10](#_Toc343246165)

[4.6 Technische Grenzwertanalyse 10](#_Toc343246166)

[4.7 Präsentation bei M&F 10](#_Toc343246167)

[5. Stabilisierung 10](#_Toc343246168)

[5.1 Unterscheidung Aktiv-/Passiv-User 10](#_Toc343246169)

[5.2 Stabilisierung & Kalibrierung 10](#_Toc343246170)

[5.2.1 Bedienungsstabilität 10](#_Toc343246171)

[5.2.2 Fremdeinflüsse 10](#_Toc343246172)

[6. Hilfsmittel 10](#_Toc343246173)

[7. Anforderungskriterien 11](#_Toc343246174)

[7.1 Umfeld 11](#_Toc343246175)

[7.1.1 Störfaktoren 11](#_Toc343246176)

[7.2 Anforderungen 11](#_Toc343246177)

[7.2.1 Umgebung 11](#_Toc343246178)

[7.2.2 Workflow 11](#_Toc343246179)

[7.2.3 Auswertung Realisierbarkeit 12](#_Toc343246180)

[7.3 Erweiterungen 12](#_Toc343246181)

[8. Gesten 12](#_Toc343246182)

[8.1 Benutzte Gesten anderer Produkte und Projekte 13](#_Toc343246183)

[9. Mögliche Gesten 14](#_Toc343246184)

[9.1 Anmeldung 14](#_Toc343246185)

[9.1.1 Arme nach aussen Halten 14](#_Toc343246186)

[9.1.2 Slide to Unlock 14](#_Toc343246187)

[9.1.3 Winken 14](#_Toc343246188)

[9.1.4 Militärischer Gruss 14](#_Toc343246189)

[9.1.5 Verbeugen 14](#_Toc343246190)

[9.2 Zoom 14](#_Toc343246191)

[9.2.1 Pinch 14](#_Toc343246192)

[9.2.2 Push/Pull 14](#_Toc343246193)

[9.2.3 Kreis Zeichnen 15](#_Toc343246194)

[9.3 Scrollen/Blättern 15](#_Toc343246195)

[9.3.1 Wischen 15](#_Toc343246196)

[9.3.2 Joystick 15](#_Toc343246197)

[9.3.3 Oberkörper bewegen 15](#_Toc343246198)

[9.4 Cursor bewegen 15](#_Toc343246199)

[9.4.1 Joystick 15](#_Toc343246200)

[9.4.2 Zeigen 15](#_Toc343246201)

[9.4.3 2D Mapping der Handposition 15](#_Toc343246202)

[9.5 Auswählen 15](#_Toc343246203)

[9.5.1 Stossen 16](#_Toc343246204)

[9.5.2 Spezielle Gesten mit der anderen Hand 16](#_Toc343246205)

[9.5.3 Grab 16](#_Toc343246206)

[9.5.4 Nicken 16](#_Toc343246207)

[9.5.5 Timer 16](#_Toc343246208)

[9.5.6 Thumb up 16](#_Toc343246209)

[9.6 Spezialaktionen (spezielle Aktionen) 16](#_Toc343246210)

[9.6.1 Bestimmter Winkel zwischen Körper und Armen 16](#_Toc343246211)

[9.7 Abmeldung 16](#_Toc343246212)

[10. Entscheidung 16](#_Toc343246213)

[10.1 Anmeldung – *Winken* 16](#_Toc343246214)

[10.2 Zoom – *Pinch-Zoom* 17](#_Toc343246215)

[10.3 Scrollen/Blättern – *Swipe/Joystick* 17](#_Toc343246216)

[10.4 Cursor bewegen – *Joystick* 17](#_Toc343246217)

[10.5 Auswählen – *spezielle Gesten/Nicken* 17](#_Toc343246218)

[10.6 Spezialaktionen (spezielle Aktionen) 18](#_Toc343246219)

[11. Abhängigkeiten der Gesten 18](#_Toc343246220)

[12. Innere Architektur 19](#_Toc343246221)

[12.1 Überlegungen zur allgemeinen Architektur 19](#_Toc343246222)

[12.1.1 Events 19](#_Toc343246223)

[12.1.2 Berechnungen 19](#_Toc343246224)

[12.1.3 Hauptfunktion: Gestenerkennung 19](#_Toc343246225)

[12.2 Lösungsansätz für aufgetretene Probleme 19](#_Toc343246226)

[12.2.1 Zuweisungsalgorithmus für neue und bestehende Personen 19](#_Toc343246227)

[12.2.2 Zeitmessung in der .NET-Umgebung 19](#_Toc343246228)

[12.2.3 Event-Triggers aus Subklassen 19](#_Toc343246229)

[12.2.4 GestureChecker-Statemachine: Unterscheidung zwischen triggered und success 20](#_Toc343246230)

[12.2.5 … 20](#_Toc343246231)

[12.3 Domain 21](#_Toc343246232)

[12.4 Beispielsequenz einer Geste 22](#_Toc343246233)

[12.5 Memory Management: Speicherzuweisung für Eventhandling 23](#_Toc343246234)

[12.5.1 Code-Analyse der Zuweisungen 23](#_Toc343246235)

[12.5.2 Online-Analyse mit Mocking 24](#_Toc343246236)

[12.5.3 Online-Analyse ohne Mocking 24](#_Toc343246237)

[13. Äussere Architektur (API) 25](#_Toc343246238)

[13.1 Allgemeine Überlegungen 25](#_Toc343246239)

[13.1.1 Multi-Layer 25](#_Toc343246240)

[13.2 Schnittstellendefinition – Hoher Layer 25](#_Toc343246241)

[13.3 Schnittstellendefinition – Tiefer Layer 25](#_Toc343246242)

# Einleitung

## Zweck

Diese Planung beschreibt den Umfang und den Verlauf der Semesterarbeit über die Kinect bei M&F. Dieses Dokument soll eine kurze Übersicht über die anfallenden Arbeiten und deren Reihenfolge geben. Kritische Aspekte der Arbeit werden kurz erläutert.

## Geltungsbereich

Semesterarbeit Kinect HS12 unter Betreuung von Oliver Augenstein und in Zusammenarbeit mit M&F

## Definitionen, Akronyme, Abkürzungen

## Literaturverzeichnis

Einfügen mit Verweis->Zitate und Literaturverzeichnis -> Literaturverzeichnis->Literaturverzeichnis einfügen. Als Formatvorlage „ISO 690 Numerische Vorlage“ verwenden.

Literaturverweise mittels „Verweise“->“Zitat einfügen“ erstellen.

## Übersicht

Im Industrieumfeld kann die Bedienung von Maschinen durch herkömmliche Eingabegeräte ungeeignet sein. In der SA soll untersucht werden, ob und wie weit die Kinect von Microsoft als Eingabegerät in Frage kommt.

# Zieledefinition

Die Aufgabe der Semesterarbeit ist die Beantwortung folgender Frage:

*„Ist die Kinect für Industrielösungen einsetzbar?“*

Zur Erreichung des Ziels wird ein Prototyp entwickelt, welcher einen ausgewählten Satz an Eingabemethoden demonstriert. Der Prototyp wird hinsichtlich Stabilität analysiert und optimiert. Dazu gehört auch eine technische Grenzwertanalyse, welche die Einsetzbarkeit des Produkts in einem Industrie-Umfeld dokumentiert.

Der zeitliche Ablauf der Semesterarbeit umfasst drei Phasen:

* **Evaluation** bestehender technischer Lösungen und Bedienkonzepte
* **Entwicklung** und Implementierung eines Bedienkonzeptes für einen minimalen Funktionssatz. Das beinhaltet die folgenden Unterschritte pro Bedienkonzept:
  + Implementierung: Grundimplementierung
  + Stabilisierung: Es wird so lange auf Input stabilisiert, bis klar ist, ob das Bedienkonzept industrie-tauglich ist.
* **Dokumentation** der Ergebnisse

## Evaluation

Während der Evaluationsphase werden verschiedene bestehende Projekte betreffs deren technischer Möglichkeiten und vorhandener Bedienkonzepte analysiert. Die technische Analyse beschränkt sich auf die Erfüllbarkeit der Anforderungen von M&F. Die Analyse der verschiedenen Bedienkonzepte ist breiter gefächert. So können exotische Konzepte, deren Umsetzung die technischen Anforderungen nicht erfüllen u.U. doch berücksichtigt und ev. adaptiert werden.

## Entwicklung

Ein Satz an ausgewählten Bedienkonzepten wird implementiert. Konkret abzubilden sind folgende Funktionen:

* Blättern
* Scrollen
* Klicken
* Zoomen

Zusätzlich müssen An- und Abmeldung einer Person implementiert werden. Ebenso nicht zu vergessen ist die Kalibrierung.

## Stabilisierung

Jedes Bedienkonzept wird stabilisiert. Das bedeutet konkret, es wird darauf geschaut, dass die Bedienung der Kinect mittelst Geste stabil gegenüber äusseren Einflüssen gemacht wird. Fremdeinflüsse wie Staub oder Licht sind ebenso zu analysieren wie das Verhalten der bedienenden Personen. Am Ende werden durch die Stabilisierung gewonnenen Erkenntnisse in einer übersichtlichen Produkt-Spezifikation festgehalten.

# Wochenplanung

Die Wochenplanung soll nicht als fix verstanden werden. Je nachdem, wieviel Zeit bei der Implementierung wirklich gebraucht wird, werden die einzelnen zu entwickelnden Gesten nicht komplett entwickelt und nicht komplett stabilisiert.

|  |  |  |
| --- | --- | --- |
| Semesterwoche | Aufgaben und Artefakte | Meilensteine |
| W1 (21.9.) | Einarbeitung und Aufsetzen der Umgebung |  |
| W2 (28.9.) | Evaluation bestehender Produkte und deren Bedienkonzepte |  |
| W3 (5.10.) | Definition der zu verwendenden Bedienkonzepte (konkret)  Analyse der Gestenabhängigkeiten  **End of Elaboration am 10.10.2012** | **M1** |
| W4 (12.10.) | Implementierung Standardmodus:  Konkret: Pinch-Zoom inkl. Stabilisierung |  |
| W5 (19.10.) | "" |  |
| W6 (26.10.) | **"" Pinch-Zoom bis zum Machbarkeits-Grad\* implementiert** | **M2** |
| W7 (2.11.) | Implementierung Standardmodus:   1. Blättern 2. Anmelden 3. Auswahlgesten |  |
| W8 (9.11.) | "" |  |
| W9 (16.11.) | Zeit für Verbesserungen oder Implementierung Joystick-Modus:   1. Scrollen 2. Cursor 3. Auswahl |  |
| W10 (23.11.) | "" |  |
| W11 (30.11.) | "" |  |
| W12 (7.12.) | "" **Beide Modi umgesetzt & dokumentiert** | **M3** |
| W13 (14.12.) | Technische Grenzwertanalyse & -bericht  **Dokumente finalisiert** | **M4** |
| W14 (21.12.) | Präsentation bei M&F |  |

Farben: Evaluation, Entwicklung, Berichte

\*Machbarkeitsgrad: Entwicklungsstand, an welchem klar ist, dass das Konzept erfolgreich implementiert werden könnte.

# Aufgabenbeschreibung

Im Folgenden werden die zu erledigenden Aufgaben in chronologischer Reihenfolge kurz beschrieben.

## Technische Produktevaluation

Es ist zu untersuchen, was für Produkte für Gestenerkennung es bereits gibt. Diese Produkte werden hinsichtlich ihrer technischen Möglichkeiten mit den Anforderungen von M&F verglichen. Einige sind schon alt und ausgereift (z.B. *LightGun*), andere befinden sich noch in der Entwicklung (z.B. *The Leap*). Eventuell sind OpenSource-Projekte hilfreich für unsere Minimalimplementierung (siehe *Entwicklungsphase*).

## Evaluation von verschiedenen Bedienkonzepten

Es gibt diverse Bedienmechanismen für schon existierende Geräte. Wir werden diese Verfahren mit den Anforderungen von M&F vergleichen und ev. an die Zielplattform Kinect anpassen.

## Analyse der Gestenabhängikeiten

Es wird evaluiert, in welcher Reihenfolge die verfügbaren Bedienkonzepte implementiert werden sollen. Es nach Aufwand/Lernerfolg priorisiert. So wird mehrfache Arbeit minimiert.

## Implementierung Standardmodus

Wir haben uns entschieden, zwei Modi zu entwickeln. Den *Standardmodus* für eine Grundlegende Bedienung und den *Joystick-Modus* für eine erweiterte Bedienung mit Cursor. Begonnen wird die Implementierung mit dem *Pinch-Zoom* aus dem *Standard-Modus*. Die genauen Gesten und weshalb sie gewählt wurden, ist im *Evaluationsdokument* festgehalten.

### Pinch-Zoom

Der *Pinch-Zoom* ist die aufwendigste Geste des *Standardmodus*. Er vereint die meisten anderen Gesten in sich und verspricht so den grössten Lernerfolg für die Implementierung der anderen. Der *Pinch-Zoom* wird, wie die anderen Gesten auch, nur bis zu dem Grad entwickelt, wo klar ist, ob die Komplettierung machbar ist. So kann der Fokus weiter auf andere Bedienkonzepte gelegt werden.

### Blättern

Die Geste *Blättern* gehört zum *Standardmodus* und ist grundlegend wichtig für die Bedienung einer Industrieanwendung. Deshalb wird diese Funktion als zweites implementiert. Wir rechnen damit, viele Erfahrungen aus der Umsetzung des *Pinch-Zooms* wiederverwenden zu können.

### Anmelden

Ohne Anmeldung ist das Komplettsystem nicht praktisch einsetzbar. Deshalb wird deren Umsetzung als wichtig gewertet. Die *Anmeldung* gehört zu beiden Modi (*Standard*- und *Joystick-Modus*).

### Auswahlgesten

Auf dem Industriepanel sollen Buttons ausgewählt werden können. Im *Standardmodus* geschieht das über speziell definierte Gesten. Die Auswahl kann man mit dem Joystick nicht direkt umsetzen (ausser Timer a la Wii). Darum wird eine weitere Auswahlgeste implementiert, z.B. *Nicken*.

## Implementierung des Joystick-Modus

Der Joystick-Modus setzt eine erweiterte Bedienung des Industriepanels um. Er soll lediglich implementiert werden, falls die Zeit reicht. Der Joystick ist eine komplexe Geste, die viel Feingefühl in der Bedienung erfordert. Dafür können mit ihm alle erforderlichen Funktionen umgesetzt werden. Mit den aus dem Standardmodus gewonnenen Erkenntnissen, sollte es gut möglich sein, den Joystick umzusetzen.

### Scrollen

Mit dem Joystick kann in alle Richtungen gescrollt werden, sobald der Arm eine äussere Deadzone verlässt.

### Cursor

Mit dem Joystick kann ein ein Cursor auf dem Bildschirm bewegt werden, sobald der Arm eine innere Deadzone verlässt.

## Technische Grenzwertanalyse

Die Grenzen unseres Produktes sollen aufgezeigt werden. Dazu wird das System aus verschiedenen Entfernungen bei verschiedenen Fremdeinflüssen bedient. Die Resultate der Bedienung werden bewertet und in einer Produkt-Spezifikation dokumentiert.

## Präsentation bei M&F

Die Resultate der Semesterarbeit über Industrieanwendungen mit Kinect werden dem Kunden präsentiert.

# Stabilisierung

Während der Stabilisierungsphasen der einzelnen Gesten, müssen folgende Dinge beachtet werden.

## Unterscheidung Aktiv-/Passiv-User

Bevor das zu entwickelnde System benutzt werden kann, muss klar sein, wer es bedient. Dazu muss ein Mechanismus zur Identifikation des davorstehenden Benutzers entwickelt werden. Das System soll auf eine Anmeldung warten und dann so lange nur den angemeldeten Benutzer beachten bis dieser sich entweder manuell abgemeldet hat oder automatisch vom System abgemeldet wurde.

## Stabilisierung & Kalibrierung

Eine wichtige Anforderung von M&F ist die Robustheit der Anwendung. Es wird Wert darauf gelegt, dass die Bedienung von Industriegeräten einfach und stabil ist. Die Stabilität kann unterteilt werden in Bedienung und Fremdeinflüsse. Ziel der Semesterarbeit ist, möglichst viel Stabilität in beiden Kategorien zu erlangen.

### Bedienungsstabilität

Die Benutzung einer 3D-Erkennungssoftware lässt sich nicht trivialerweise diskret abbilden. Deshalb sind für eine stabile Anwendung gute Heuristiken wichtig. Ziel dieser Arbeit ist, den maximalen Spielraum einer diskreten Abbildung auf den zu benutzenden Funktionssatz ausreizen. So wird das Benutzen der Anwendung bequem.

*Beispiel*: Eine identifizierte Person bedient das System. Sie kratzt sich am Kopf. Danach fährt die Person mit der Bedienung fort. Das System ignorierte, dass die Person sich am Kopf gekratzt hat.

### Fremdeinflüsse

Die Kinect hat relativ beschränkte Möglichkeiten. So können z.B. Fremdlicht, Staub oder spiegelnde Oberflächen ein Problem sein. Unser Ziel ist, die Anwendung möglichst robust gegen solche Fremdeinflüsse zu machen. Dies kann u.U. mittels manueller Kalibrierung erreicht werden.

*Beispiel*: Eine fremde, nicht identifizierte Person läuft ins Bild und stört die bedienende Person. Das System beachtet die fremde Person nicht.

# Hilfsmittel

* Visual Studio 2010
* Kinect SDK 1.5

# Anforderungskriterien

## Umfeld

Die in der Semesterarbeit zu erstellende Software wird im industriellen Umfeld eingesetzt – konkret in einer grossen Halle. Man muss Bildschirme bedienen können, die bis zu 3m in der Höhe angebracht sind. Die Distanz zum User beträgt etwa 2 - 10 Meter. Die Raumbeleuchtung befindet sich an der Decke – normalerweise Neon oder Halogen. Es könnte aber auch Tageslicht von oben oder der Seite hinzukommen.

In der Halle bewegen sich normalerweise mehrere Personen, teilweise auch kleinere Fahrzeuge. Die Arbeiter haben unter Umständen Schutzkleidung an.

### Störfaktoren

Unter Umständen ist die Kinect kleinen Erschütterungen oder Vibrationen ausgesetzt. Durch Maschinen kann es zu Lärm oder Staubemissionen kommen. Ebenso können Wärmequellen in der Halle entstehen und sich allenfalls bewegen. Störende Beleuchtung kann künstlicher oder natürlicher Natur sein. Fabrikbeleuchtung oder -Heizung kann Licht im Infrarot- oder UV-Bereich freisetzen – das gilt auch für direktes Sonnenlicht. Da in Zukunft immer mehr LED-Beleuchtungen eingesetzt werden, müssen auch sie in dieser Arbeit berücksichtigt werden. Abgesehen von Personen und Fahrzeugen kann es auch andere Objekte haben, die sich bewegen können (z.B.: ein Kran).

## Anforderungen

### Umgebung

* Ca. 2-10m Distanz zur bedienenden Person
* Personen sollen sich nicht überwacht fühlen (BigBrother-Problem)
* Mehrere Personen können zuschauen, es bedient jedoch immer nur jemand gleichzeitig (offen für mehrere Bedienungen)
* Signalisierung der Einsatzbereitschaft des Systems: Sind Inputs möglich? Hat es zu viel Licht? Person zu weit entfernt, etc.
* Anmeldung der zu bedienenden Person (z.B.: vorbeilaufen soll keine Events triggern)
* Automatische und manuelle Abmeldung (z.B.: was passiert beim Schuhebinden)
* Realisierbarkeit mit einer Kinect
* Aktive und passive Nutzer
* Möglichst unabhängig von oben erwähnten Störfaktoren

### Workflow

* Ausführbare Aktionen/Gesten:
  + Blättern
  + Scrollen
  + Klicken
  + Zoomen
* Einfache Gestik: Personen wollen nicht den „Hampelmann machen“ oder sich exponieren.
  + Eindeutig
  + Einfach lernbar
  + Intuitiv
  + Einhändig bedienbar
* Gesten sollten nach maximal zwei Versuchen erkannt werden, irrtümliche Inputs sollten nicht vorkommen.
* Einfach kalibrierbar
* Workflow soll intuitiv und nicht zu träge sein, jedoch auch nicht zu empfindlich: guter Tradeoff gesucht

### Auswertung Realisierbarkeit

## Erweiterungen

# Gesten

Damit die Begrifflichkeiten in der Semesterarbeit immer klar sind, wird im Folgenden definiert welche Gesten mit welchen Aktionen verknüpft werden.

|  |  |
| --- | --- |
| Geste | Beschreibung |
| Push | Eine Bewegung, bei der die Hand schnell nach vorne bewegt wird. |
| Pull | Die Hand wird von einer Position vorne schnell zum Körper gezogen. |
| Pinch Zoom | Durch die Bewegung von beiden Händen zueinander wird herausgezoomt, wenn sich die Hände auseinanderbewegen wird hineingezoomt. |
| Push and Pull Zoom | Durch das Heranziehen eines Objektes wird es grösser, durch das wegstossen kleiner. |
| Wischen | eine schnelle Handbewegung nach links oder rechts, die Bewegung kommt aus dem Ellbogen, nicht nur aus dem Handgelenk. |
| Joystick | Es wird die relative Handposition zu einem definierten Punkt des Körpers verwendet. Darum herum gibt es eine Deadzone, in der nichts passiert. Ausserhalb von dieser wird in diese Richtung gescrollt. Die Geschwindigkeit ist abhängig von der Distanz. |
| Armwinkel | Der Winkel der zwischen Hand, Schulter und Hüftgelenk aufgespannt wird. |
| Winken | Hin und her bewegen der Hand, die sich oberhalb des Kopfes befinden muss. Die Bewegung kommt aus dem Ellbogen. |
| Ausgestreckt | Schulter, Ellbogen und Hand bilden eine Linie. Der Arm zeigt vom Körper weg. |
| Körper lehnen | Eine Position, wobei der Oberkörper nach vorne, hinten, links oder rechts geneigt ist. Die Körperachse ist dabei gekrümmt in dieser Richtung. |
| Handgeste | Eine Geste, die nur von der Hand ausgeführt wird. Diese Gesten können wir mit unserer Ausgangslage nicht erfassen. Daher werden wir nicht weiter darauf eingehen. |
| Laser Pointer | Durch Verlängerung einer Controllerachse oder Körperteils erhält man am Durstosspunkt durch die Bildebene einen Punkt wo ein Cursor dargestellt wird. |
| Absolutes Scrolling | Ein Punkt auf der Bildebene wird fixiert. Das Bild wird genau dem nachfolgenden Bewegungsmuster folgen und der Cursor(falls vorhanden) bleibt auf dem gleichen Bildpunkt bestehen. |
| Scrolling mit Momentum | Das Bild wird beim Loslassen mit der gleichen Geschwindigkeit weiterbewegt. Mit einer Dämpfung wird es während einem Zeitinterwall abgebremst. |
| Emulation | Controller wird als Maus und Tastatur emuliert |

## Benutzte Gesten anderer Produkte und Projekte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Device | Eigenheiten und Details | Auswahl | Zoom | Scrolling | Sonstiges |
| Xbox mit Kinect | Individuell für jedes Game, die meisten Games haben Tutorials. Kein allgemeines Konzept | Meist Push, teilweise Handerkennung | 2-Hand Pinch | Wischen, teilweise Joystick | Zusätzliche Aktionen mit z.B. bestimmtem Armwinkel Anmeldung mit Winken |
| Kinect am PC | Noch keine kommerzielle Software, meist spezifische Prototypen oder Emulatoren | individuell |  |  |  |
| Interactive Wall | 3 Beamer, 1 Kinect  Media Center | Push | - | Wischen | Anmeldung mit ausgestreckten Armen, 2 Cursor pro User |
| FAAST | Maus und Tastatur Emulator | Arm strecken | - | Körper lehnen oder Handposition |  |
| The Leap | PC Steuerung, auch mit Emulation, hohe Genauigkeit | Individuelle Handgeste | 2 Finger Zoom oder Push & Pull | Wischen | Noch nicht erhältlich |
| Sixense | Sehr genau, kurze Distanz, zwei Controller, nur Emulation | Button | Button | Button | Laserpointer |
| Glove Pie | Wii Controller am PC | Button | - | - | Nicht zuverlässig |
| Webcam | Objekt oder Handtracking am PC | Handgeste | - | Wischen | Ungenau, störungsempfindlich |
| 3D-Maus | Keine Gesten | Button | Push+Pull | Seitlich bewegen |  |
| Minority-Report | Handgesteuerte Navigation im 3D Raum. Aktionen mit Gesten | Handgeste | 2 Hand Pinch oder Handgeste | Handposition | Kleine Gesten |
| Wii | Ein Controller pro Spieler | Button | - | - | Laserpointer |
| PS Move | Ein Controller pro Spieler | Button | - | - | Genau und schnell |
| Eye Toy | Kollisionsdetektion mit Webcam | - | - | - |  |
| Light-Gun | Point and Click | Button | - | - | Keine Positionsdaten |
| Touchscreen | Weit verbreitet | Klick | 2-Finger Pinch | Absolutes Scrolling | Nur 2D |

# Mögliche Gesten

## Anmeldung

Diese Geste ist sehr wichtig, um dem System mitzuteilen, dass man es jetzt bedienen möchte. Es ist auch schwierig dafür eine gute Geste zu finden, die dem User klar ist. Es ist für einen User ungewohnt, dass er einem System signalisieren muss, dass er interagieren möchte. Alle anderen Inputsysteme reagieren sofort auf einen Input.

### Arme nach aussen Halten

Diese Geste ist ziemlich eindeutig, was Fehlaktivierungen minimieren würde. Der Nachteil ist, dass diese Geste relativ viel Platz braucht, was andere Personen behindern könnte. Zudem braucht es eine Information, da diese Geste nicht intuitiv ist.

### Slide to Unlock

Diese Geste wird 99% der User bekannt sein, es braucht keine Anleitung dafür. Dafür ist die Erkennung schwieriger, da die Position der Hand nicht klar definiert ist und es wird häufig zu irrtümlichen Aktivierungen kommen. Dafür braucht diese Geste wenig Platz. Apple hat darauf ein Patent, man müsste abklären, wie allgemein das formuliert ist.

### Winken

Diese Geste ist bei den meisten Systemen umgesetzt und ziemlich selbsterklärend. Ein kleiner Text: „Winken zum bedienen“ wird es allen Usern klar machen. Die Erkennung dieser Geste ist etwas einfacher als Slide to Unlock, aber auch hier können selten Fehlaktivierungen vorkommen. In den ersten Wochen des Einsatzes könnte es bei den Mitarbeitern zu Missverständnissen kommen, wenn sie eine Person winken sehen, da dies eine übliche Geste zwischen Personen ist um Aufmerksamkeit zu erlangen. Das exponiert den User und lenkt andere ab in einer grossen Halle. Nach einer gewissen Zeit wird dieses Problem aber verschwinden.

### Militärischer Gruss

Die militärischen Grüsse sind allgemein bekannt und intuitiv. Es stellt sich die Frage, ob die Benutzer bereit sind, vor einem elektronischen System zu salutieren. Das System erscheint u.U. unsympathisch und wird nicht akzeptiert.

### Verbeugen

Intuitive Geste, die jedoch u.U. unbequem sein kann (Arbeitskleidung). Zusätzlich stellt sich die Frage, ob eine Verbeugung nicht als Unterwerfung verstanden wird und somit keine Akzeptanz bei den Benutzern findet.

## Zoom

Zoomen ist etwas weit verbreitetes, hier gibt es eigentlich nur zwei intuitive Gesten.

### Pinch

Pinch Zoom ist auf Touchscreens sehr stark verbreitet und dürfte daher jedem User klar sein. Durch den Umstand, dass die Hände in sehr viele Richtungen zueinander und auseinander bewegt werden können, ist die Erkennung hierbei schwieriger und könnte mit anderen Gesten interferieren. Es wird auch zu Fehlaktivierungen kommen, was jedoch nicht kein Problem darstellen wird, da es sich nur um kleine Zoomänderungen handeln wird. Unklar ist ebenfalls, wann die Zoomfunktion aktiviert werden soll und wann der User seine Hände bewegt, damit er nachher eine Zoomfunktion auslösen kann. Zudem werden für diese Geste beide Hände benötigt, in unserem Umfeld wäre eine einhändige Bedienbarkeit sicher ein Vorteil.

### Push/Pull

Je nach Darstellung ist auch diese Geste sehr intuitiv und für jeden User logisch. Diese Geste ist sehr einfach zu erkennen und eindeutig. Dadurch wird es zu wenigen Fehlaktivierungen kommen. Jedoch ist auch hier unklar, wann der User zu einer solchen Bewegung ansetzen will und wann er sie wirklich ausführen will. Diese Geste kann jedoch nur mit einer Hand bedient werden.

Wenn man diese Geste verwendet kann man logischerweise Stossen und ziehen nicht mehr für eine Auswahl verwenden.

### Kreis Zeichnen

Hierbei kann der User mit einer Hand einen Kreis in die Luft zeichnen und je nach Richtung wird hinein der herausgezoomt. Das ist zwar nicht besonders intuitiv, dafür aber schnell erklärt und es kann ziemlich stabil erkannt werden. Es wird wenig Interferenz mit anderen Gesten haben. Eine sicherte Erkennung ist aber erst ab einem bestimmten Kreisdurchmesser möglich.

## Scrollen/Blättern

Bei den meisten evaluierten Systemen wird Scrollen dem Blättern gleichgesetzt. Blättern wird dabei als Einrastfunktion für Scrollen umgesetzt.

### Wischen

Diese Geste wird praktisch bei allen anderen Projekten verwendet. Da man die Geschwindigkeit der Bewegung analysieren kann, werden einerseits Fehlaktivierungen minimiert, andererseits kann man die Distanz und die Geschwindigkeit auf die Aktion übertragen.

### Joystick

Diese Bewegung ist nicht ganz so intuitiv wie andere, dafür hat sie den Vorteil, dass man nicht „nachgreifen“ muss. Zudem sind bei dieser Bewegung der Platzbedarf und der Bewegungsradius grösser. Je nach Gestaltung des GUIs ist auch diese Geste intuitiv und sollte allen Nutzern klar sein.

### Oberkörper bewegen

Diese Geste ist nicht intuitiv und braucht sicher eine Anleitung. Zudem ist sie für unser Umfeld mit spontanen Nutzern nicht wirklich geeignet. Unter Umständen ist sie auch unbequem. Aus technischer Sicht ist diese Geste wahrscheinlich schwierig auszuwerten falls der Benutzer weit weg steht.

## Cursor bewegen

### Joystick

Hierbei werden die Cursorfunktion und das Scrolling kombiniert. Das ist stabil erkennbar und hat keine Interferenz mit anderen Gesten. Es wird den Usern intuitiv klar sein.

### Zeigen

Dabei wird mit dem Unterarm auf einen Punkt gezeigt. Das ist sicher ungenauer in der Erkennung und benötigt wahrscheinlich eine Kalibrierung für jede neue Session. Zudem wird die Nutzung der Geste „Push“ verunmöglicht. Für den User ist diese Geste sehr intuitiv.

### 2D Mapping der Handposition

Die Handposition wird vor dem Körper mit angewinkeltem Arm ausgewertet. Diese Geste ist für User gewöhnungsbedürftig und nicht sehr stabil erkennbar. Dafür interferiert sie nicht mit anderen Gesten und es wäre möglich, pro Hand einen Cursor darzustellen. Klicken könnte dann durch Zeigen umgesetzt werden.

## Auswählen

Diese Geste ist sehr wichtig, jedoch nicht ganz einfach umzusetzen, da es bei anderen Technologien dafür immer einen Button oder eine intuitive Lösung gibt. Das ist bei unserer Lösung nicht der Fall. Durch die grosse Distanz wird es zudem unmöglich Handgesten zu erkennen. Deshalb ist es wichtig beim finalen Programm darauf zu achten, dass möglichst wenige Selektionen gemacht werden müssen.

### Stossen

Diese Geste ist bei anderen Projekten am häufigsten umgesetzt und dürfte dem User intuitiv einigermassen klar sein. Dafür ist es bei dieser Bewegung sehr wahrscheinlich, dass währenddessen der Cursor bewegt wird, was zu einer Fehlaktion führen wird.

### Spezielle Gesten mit der anderen Hand

Hier sind verschiedene Gesten denkbar z.B. Stossen, Winken oder ausstrecken. Dadurch werden Fehleingaben minimiert, jedoch ist das einiges weniger intuitiv und benötigt zudem beide Hände.

### Grab

Dabei wird ein Objekt gepackt und zu sich gezogen. Das ist etwas einfacher zu erkennen, Cursorverschiebungen werden weniger häufig vorkommen. Je nach Gestaltung des GUIs ist diese Geste intuitiv klar.

### Nicken

Diese Geste ist intuitiv relativ klar und sollte gut zu erkennen sein. Fehlaktivierungen sind denkbar, wenn sich der User gerade mit jemand anderem unterhält. Das kann jedoch anhand der Blickrichtung korrigiert werden.

### Timer

Wenn der User seinen Cursor nicht bewegt, beginnt ein sichtbarer Timer abzulaufen. Wenn dieser abgelaufen ist, gilt die Selektion, wenn er bewegt wird. Problematisch ist hierbei eine saubere Kalibrierung. Ebenso muss erkannt werden, wann der User etwas zeigen möchte und wann er wirklich etwas selektionieren möchte.

### Thumb up

Eine sehr intuitive Geste, jedoch nicht machbar mit Kinect auf diese Distanz. Zudem wäre sie nicht für spezielle Arbeitskleidung (Handschuhe) einsetzbar.

## Spezialaktionen (spezielle Aktionen)

Solche sind für unser Projekt nicht nötig, ausser man würde sie für die Anmeldung einsetzten. Dadurch haben wir uns nicht auf solche konzentriert bei der Recherche.

### Bestimmter Winkel zwischen Körper und Armen

Diese Geste ist nicht besonders intuitiv und braucht eine Anleitung. Dafür ist sie gut zu erkennen.

## Abmeldung

Die Abmeldung geschieht automatisch nach einem Timeout wenn der Benutzer das Sichtfeld der Kinect verlässt. Eine manuelle Abmeldung könnte durch das nochmalige Ausführen der Anmelde-Geste umgesetzt werden.

# Entscheidung

Wir haben uns entschieden, zwei verschiedene Bedienmodi einzuführen. Der erste ist die Standardbedienung mit den zwei Gesten *Zoom* und *Blättern* und eventuell mehreren Auswahlgesten. Der zweite Modus ist die Joystick-Bedienung. Dieser Modus kann verwendet werden um komplexere Bedienungen wie Zeigen auszuführen. Er bildet die Funktionen *Zoom*, *Scrollen*, *Zeigen*, *Auswählen* ab. Die Implementierung dieses Modus hat für uns niedrige Priorität.

Beide Modi werden ergänzt um die *Anmelde*-Geste.

## Anmeldung – *Winken*

Hier haben wir uns für Winken entschieden, da diese Geste sehr intuitiv ist und wenige Fehlaktivierungen auslösen wird. Zudem benötigt ein Hinweis für diese Geste nur sehr wenig Platz auf GUI.

Slide To Unlock wäre ebenfalls sehr intuitiv gewesen, hätte aber viel Platz auf dem GUI benötigt und die Erkennung ist unsicherer.

Die anderen sind unterwürfige Gesten, welche bei den Benutzern nicht gut ankommen werden. Wer will sich schon einem PC unterordnen?

## Zoom – *Pinch-Zoom*

Aufgrund der Intuitivität haben wir uns für Pinch-Zoom entschieden, obwohl beide Hände für die Geste benötigt werden. Es ist schwierig, zu erkennen, wann die Geste getriggert werden soll und wann der Nutzer die Geste beendet hat. Beispielsweise möchte ein User schlicht seine Hände zurückziehen.

In Kombination mit dem Joystick-Mode werden wir Push-/Pull Zoom verwenden, aufgrund der eventuellen Interferenzen.

Der Zoom-Ring war zwar früher bei analogen Devices vorhanden, ist jedoch im Moment nicht als Geste üblich und könnte deshalb nicht richtig verstanden werden.

## Scrollen/Blättern – *Swipe/Joystick*

Für einfache Applikationen, die keinen Cursor benötigen und wo jeweils nur wenig geblättert werden muss (z.B. PowerPoint) werden wir *Wischen* implementieren. Für komplexere Anwendungen werden wir, falls die Zeit reicht, den Joystick verwenden, da dieser sehr gut erkannt werden kann und wir damit zugleich auch einen Cursor erhalten würden.

Den Oberkörper zu bewegen ist zu anstrengend und nicht mit allen Arbeitskleidern gut möglich.

## Cursor bewegen – *Joystick*

Wir haben uns hier dafür entschieden, dass wir nur einen Cursor verwenden werden, wenn die Software im Joystick-Modus bedient wird. In vielen Programmen ist es nämlich nicht nötig, einen Cursor zu haben. Dabei können wir die Position innerhalb einer Deadzone einfach für Cursorbewegungen verwenden.

Das Zeigen zu implementieren wäre sicherlich interessant gewesen. Wir vermuten, dass die User wahrscheinlich mit den Fingern zeigen würden. Das Erkennen von Fingern bei unserer Bedienentfernung scheitert an den technischen Möglichkeiten der Kinect. Die Implementierung der Zeigengeste mittels Arm wäre zwar technisch u.U. machbar, aber zu komplex für den Umfang unserer Arbeit.

Ein 2D Mapping der Handposition entspricht eigentlich ziemlich stark dem Modus, den wir für den Joystick verwenden, daher ist jene Geste nicht mehr zu beachten.

## Auswählen – *spezielle Gesten/Nicken*

Auch hier sind wir davon abhängig, ob wir uns in einem Bedienmodus befinden, der einen Cursor zur Verfügung hat, oder nicht. Im Joystick-Modus haben wir keine Handgesten mehr zur Verfügung und eine Geste mit der anderen Hand empfinden wir nicht als besonders intuitiv. Deshalb haben wir uns für Nicken entschieden, da es eine gut erkennbare Geste ist.

In einem Bedienmodus ohne Cursor, wo es nur wenige Objekte mit Interaktionsmöglichkeiten geben wird, können wir wenige Gesten auf diese Objekte legen, die wir schon implementiert haben, z.B. Stossen oder Nicken. Für eine finale Software müsste man hier allenfalls noch ein paar wenige zusätzliche Gesten definieren, was den Benutzer aber nicht überfordern darf. Das heisst, dass nicht mehr als 4 Bedienelemente zur gleichen Zeit aktiv sein sollten, andernfalls wird der Benutzer die Übersicht verlieren. Mittels animierter Icons kann man die Nutzer gut auf die zu verwendende Geste hinweisen.

Diese Geste ist sehr wichtig, jedoch nicht ganz einfach umzusetzen, da es bei anderen Technologien dafür immer einen Button oder eine intuitive Lösung gibt. Das ist bei unserer Lösung nicht der Fall. Durch die grosse Distanz wird es zudem unmöglich Handgesten zu erkennen. Deshalb ist es wichtig beim finalen Programm darauf zu achten, dass möglichst wenige Selektionen gemacht werden müssen.

Einen Timer werden wir nicht verwenden, da dies ein stabiles Inputsignal voraussetzt und den uneingeschränkten Fokus des Nutzers.

Thumb-Up wäre sehr intuitiv ist aber auf diese Distanz sicher nicht mehr erkennbar und fällt daher raus.

## Spezialaktionen (spezielle Aktionen)

Werden für unser Projekt nicht benötigt, sind aber sicherlich möglich im Umfang der anderen Gesten.

# Abhängigkeiten der Gesten

Im folgenden Abschnitt wird nur auf die Gesten eingegangen, die wir für unser Projekt verwenden, da es sonst unnötig kompliziert werden würde und andere Gesten für unser Projekt nicht von Belang sind.

Wir ordnen die Gesten in Datengruppen ein. Die Datengruppen beschreiben, welche Daten(Skelett-Punkte) in die Berechnung einbezogen werden müssen.

|  |  |
| --- | --- |
| Hand rechts | * Winken * Joystick * Wischen * Pinch-Zoom * Push * Pull |
| Hand links | * Pinch-Zoom |
| Kopf | * Nicken |
| Relative Position  (Abhängig von anderem Körperpunkt) | * Winken * Joystick * Push * Pull |
| Bewegungsrichtung | * Wischen * Pinch-Zoom |
| Bewegungsgeschwindigkeit | * Winken * Wischen * Nicken |
| Unterscheidung Hin- und Rückbewegung | * Wischen (zwei verschiedene Gesten) * Pinch-Zoom (zwei verschiedene Gesten) * Nicken * Winken |
| Unterscheidung Gestenstart- und Ende | * Wischen * Pinch-Zoom |

Für Pinch-Zoom müssen viele Faktoren analysiert werden. Es ist die komplizierteste, jedoch kann man für die anderen sicher am stärksten profitieren.

Winken ist eher unabhängig zu anderen Gesten. Es besteht eine geringfügige Ähnlichkeit zu Wischen.

Joystick verwendet nur die relative Position, dafür ist man auf ein stabiles Inputsignal angewiesen. Diese Geste sollt man erst machen wenn man den Input schon stabilisiert hat.

Wischen ist ähnlich zu Pinch, aber etwas einfacher, da man nur eine Hand auswerten muss.

Nicken hat eine ähnliche Charakteristik wie Winken, jedoch müssen andere Skelett-Punkte analysiert werden.

Push\Pull gehören zum Joystick, es ist lediglich eine andere Achse, die analysiert werden muss.

# Innere Architektur

## Überlegungen zur allgemeinen Architektur

### Events

Die Architektur der Gestenerkennungssoftware legt den Schwerpunkt auf eventbasierte Programmierung. Dies hat den Vorteil, dass auf Schleifen und Threading verzichtet werden kann. Das verringert die Komplexität der Anwendung. Zur Statussynchronisation können Event-Argument-Objekte verwendet. Dies fördert die Wart- und Anpassbarkeit des Codes. Ein wichtiger Nachteil ist das Memory Management (s.u.).

### Berechnungen

Berechnungen an den 3D-Skeletten sind in eigene Klassen gekapselt, was Korrekturen vereinfacht und Duplicated Code verhindert.

### Hauptfunktion: Gestenerkennung

Gesten werden vom Gesture-Checker erkannt. Jener überpfüft, ob die Reihenfolge der zu erfüllenden Gestenteile stimmt. Falls die Erkennung erfolreich war, triggert er die vom API-Benutzer für die erfolgreiche Erkennung hinterlegte Funktion.

## Lösungsansätz für aufgetretene Probleme

### Zuweisungsalgorithmus für neue und bestehende Personen

Die Unterscheidung von mehreren Personen hat Schwierigkeiten bereitet. Die Kinect wechselt die Nummerierung der erkannten Skelette ohne erkennbares System. Deshalb muss die Zuweisung der erkannten Skelette an neue oder bestehende Personen per Software erfolgen. Dies erfordert ein Matching zwischen den bisherigen Skeletten der existierenden Personen und den Skeletten die wir jeweils neu von der Kinect bekommen.

Die Ähnlichkeit eines Skelettes zu einem anderen wird lediglich anhand des Skelettgliedes „Hüfte“ bewertet. Diese Bewertung ergibt eine 2D-Matrix mit den Abweichungen als Einträge. Das Problem war nun die Auswertung dieser Ähnlichkeiten. Wie kann man am besten auswerten, welches Skelett welcher Person zuzuweisen ist? Eine Idee war, jeweils das Minimum in der Matrix zu suchen, die Zuweisung zu machen und sowohl Skelett als auch Person aus der Match-Matrix zu löschen. Das ist aber eventuell im Durchschnitt nicht die beste Zuweisung.

Wir entschieden uns vorerst für eine naive Lösung mit Listen, die jedoch gut zu funktionieren scheint:

Es werden drei Fälle unterschieden:

1. Es hat mehr Skelette als schon bestehende Personen, d.h. es kam eine Person ins Bild, sie muss neu erstellt werden. Zudem müssen ihr die benötigten Events registriert werden.
2. Es hat mehr bestehende Personen als neue Skelette, d.h. es ging eine Person aus dem Bild. Sie muss gelöscht/vergessen werden (Sie bleibt in einem dafür vorgesehenen Cache).
3. Es hat gleich viele Personen und Skelette, d.h. Zuweisung muss neu gemacht werden, sonst nichts.

Die Zuweisung erfolgt nun einfach mittels zwei temporärer Listen, aus welchen die gematchten Elemente gelöscht werden. Was übrig bleibt muss nach den drei Fällen (s.o.) beurteilt werden.

### Zeitmessung in der .NET-Umgebung

… Ticks, millis, etc. … TODO Renato

### Event-Triggers aus Subklassen

Events, aus Subklassen können nicht direkt aufgerufen werden, sonden müssen in der Subklasse von einer protected Funktion gekapselt werden –in folgendem Stil: *fireSubclassEvent()*.

### GestureChecker-Statemachine: Unterscheidung zwischen triggered und success

Lösung: DynamicCondition erbt von Condition

### …

## Domain

Die folgende Domainanalyse ist stark vereinfacht und soll einen groben Überblick über die Gestenerkennungssoftware ermöglichen.

|  |  |
| --- | --- |
| Objekt | Beschreibung |
| Checker | Der Checker benutzt die von der *Person* abgespeicherten Skelettdaten um verschiedene zeitabhängige Berechnungen durchführen zu können, z.B:  absolute Geschwindigkeit, relative Geschwindigkeit, etc. |
| Condition | Eine *Condition* ist der eigentliche Gestenteil. Sie kann mit *check* auf Gültigkeit überprüft werden. Zudem beinhaltet sie die Events *OnSuccess* und *OnFail*. |
| Device | Das *Device* kapselt das physikalische Gerät. Es gibt *Personen* per Event zurück wenn sie aktiv werden und meldet neue *Skelette*. |
| GestureChecker | Der *GestureChecker* führt Buch über eine komplette Geste. Er speichert wie weit fortgeschritten die Erkennung einer Geste ist und feuert Events bei der erfolgreichen Beendigung oder beim Abbruch. |
| SkeletonMath | Klasse für Vektorberechnungen auf Skelettdaten. |
| SmothendSkeleton | Geglättete *Skelette* sind die Datenquelle für alle Berechnungen und damit für die Gestenerkennung. Sie werden in der *Person* gespeichert und vom *Checker* verarbeitet. |



## Beispielsequenz einer Geste

Der Ablauf einer Gestenerkennung lässt sich in drei Phasen aufteilen:

1. Initialisierung:

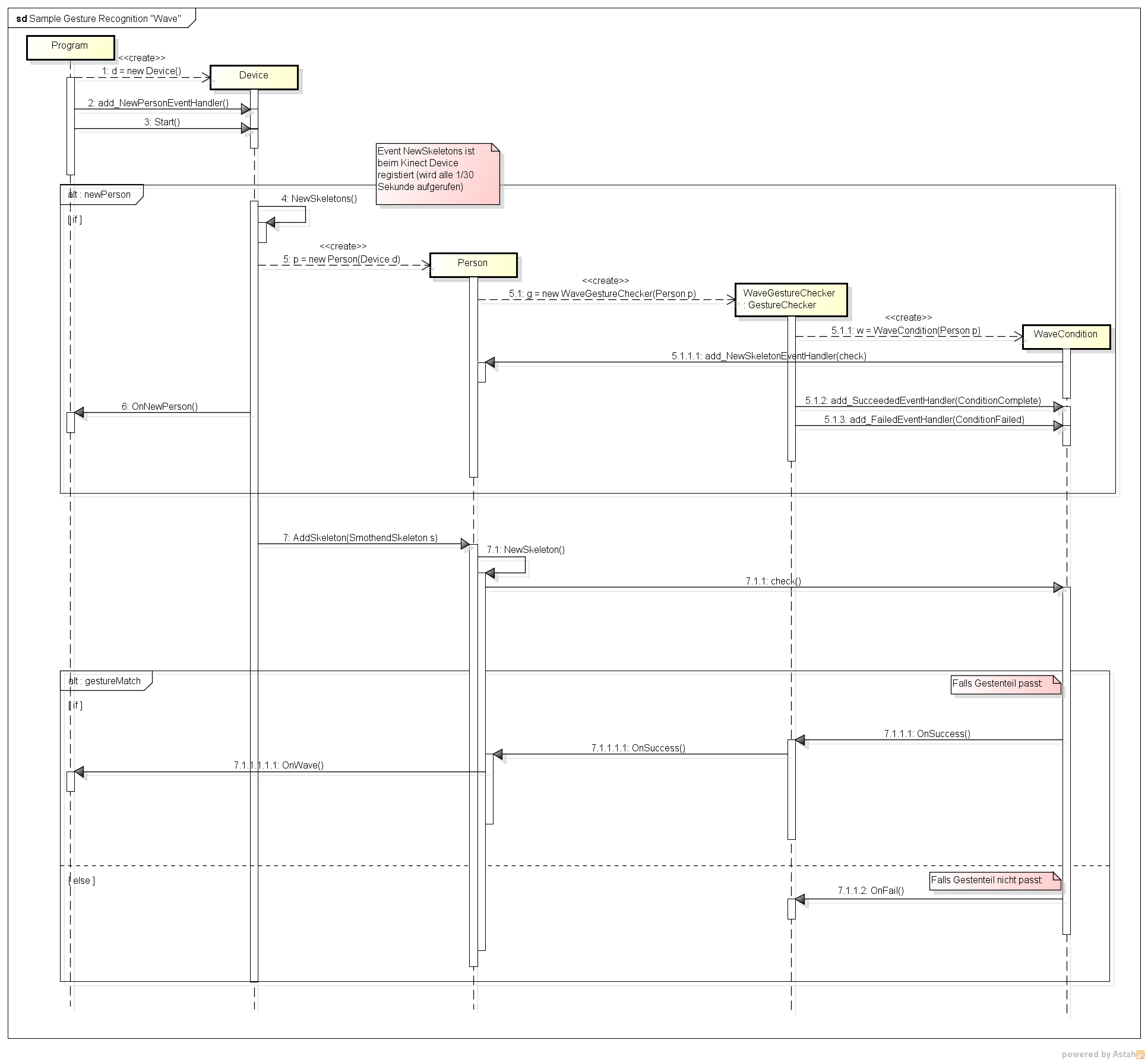
In jedem Framezyklus werden alle *Personen* vom *Device* eindeutig identifiziert. Die identifizierten *Personen* bekommen vom *Device* ihr aktuelles *Skelett* zugewiesen. Jede *Person* erhält so mit der Zeit einen Skelettcache von 10 zeitlich geordneten *Skeletten*, welche für dynamische Berechnungen verwendet werden können.

1. Prüfen:

Nach der Zuweisung vom neuen *Skelett* wird von den der *Person* zugeordneten *GestureCheckern* geprüft ob das aktuelle *Skelett* einen gültigen Kontext für die registrierten *Conditions* bildet.

1. Feedback:

Der GestureChecker führt Buch über die Reihenfolge der Gestenteile (Conditions). Fall eine Geste komplett fertig durchlaufen wird, ist sie erfolgreich und wird mit einem von der User-API registrierten Event quittiert.



## Memory Management: Speicherzuweisung für Eventhandling

Das verwenden von Events hat den Nachteil, dass long-lived Publisher ihre Referenzen zu Event-Subscribern im Speicher behalten und so zu Speicherlecks führen können. Unsere Applikation wurde auf Lecks geprüft und für stabil empfunden.

### Code-Analyse der Zuweisungen

In folgender Tabelle sind alle EventHandler-Referenzen aufgeführt. Es ist ersichtlich, dass praktisch alle EventHandler-Referenzen vom GarbaggeCollector beim Aufräumen einer *Person*-Instanz indirekt im Speicher invalidiert werden. Der Event im Device ist auch unproblematisch, da sowohl die *KinectSensor*- als auch die *Device*-Instanz long-lived sind. Zur Sicherheit wird er jedoch beim Entfernen einer *Person* aus dem Expiration-Cache deregistriert.

Da eine *Person*-Instanz jeweils nur während der Bedienungszeit existiert, sind die verwendeten Events unproblematisch. Der Speicher wird nicht zu sehr beansprucht.

|  |  |  |  |
| --- | --- | --- | --- |
| Klasse | Publisher | Subscriber | Subscriber lebt länger oder gleich lang wie Publisher |
| Condition | Person.NewSkeleton | Condition.check | ✔ |
| Device | KinectSensor.SkeletonFrameReady | Device.NewSkeletons | ✔ |
| Device | Person.OnWave | Device.personWaved | **?** |
| GestureChecker | Condition.Succeeded | GestureChecker.ConditionComplete | ✔ |
| GestureChecker | Condition.Failed | GestureChecker.ConditionFailed | ✔ |
| ~~GestureChecker~~ | ~~Timer.Elapsed~~ | ~~GestureChecker.Timeout~~ | **✘** |
| Person | GestureChecker.\* | Person.\* | ✔ |

Bemerkung: Der Publisher hält jeweils eine Referenz auf den Subscriber, nicht aber umgekehrt.

Die einzige Ausnahme bildete die Verwendung der *Timer*-Klasse im *GestureChecker*. Der *Timer* registrierte sich bei System Events um den erfolglosen Ablauf einer Geste durch einen Timeout abzubrechen. Durch das Verwenden von vielen *GestureCheckers* entstand ein Speicherleck (siehe unten). Der Timer wurde durch eine einfachere Zeitmessung ersetzt und wird gar nicht mehr verwendet.

### C:\Users\scj\Desktop\timerLeak.pngOnline-Analyse mit Mocking

Zum Testen der Stabilität des Speichers wurde ein *Device (MockingDevice)* vorgetäuscht und das Erscheinen und Verschwinden von vielen Personen vorgetäuscht. Zudem wurden kritische Klassen wie *Condition*, *GestureChecker* und EventArgs mit grossen Byte-Arrays aufgepumpt. So wurde das einzige Speichleck schnell sichtbar.

Was jedoch zum Problem wurde war die Verwendung des C#-Timers. Jener wurde benutzt um auf einen Timeout in der GestureChecker-Statemachine zu reagieren. Dabei behielt ein System-Event des jeweils praktisch alle Objekte der Applikation. Durch die automatisierte Analyse mit SciTech .NET Memory Profiler konnten die aufgepumpten Objekte zurückverfolgt werden. Im dem Allocation-Tree ist gut sichtbar, dass praktisch alle Instanzen der Applikation vom Timer gehalten werden.

Da aus der Analyse der Aufrufshierarchie zusätzlich ersichtlich war dass der *Timer* sehr viel Rechenzeit verbrauchte, wurde komplett auf diese Klasse verzichtet. Stattdessen wird jetzt eine weniger komplexe Zeitmessung aufgrund der Systemzeit verwendet um den Timeout einer Geste zu messen.

### Online-Analyse ohne Mocking

Nach den weniger laufzeitintensiven Test wurde das Programm während einer Woche blind an einem Ort ausgeführt wo mehrere Personen ein- und ausgehen.

# Äussere Architektur (API)

## Allgemeine Überlegungen

### Multi-Layer

Die User-API wird in verschiedenen Layern aufgebaut. Der Benutzer des Gestenerkennungsframework kann entscheiden, welchen Layer und damit auch welche Komplexität er benutzen will. Das hohe Layer bietet einen eingeschränkten Funktionsumfang, den man sehr einfach einbinden kann. Das tiefe Layer bietet Möglichkeit eigene Gesten zu definieren oder auf Low-Level Eigenschaften zuzugreifen. Beide Layer lassen sich kombiniert einsetzen. Beispielsweise muss sich der Benutzer der API nicht um die Aktivierung der Personen kümmern, kann aber dennoch eigene Gesten definieren – oder umgekehrt.

## Schnittstellendefinition – Hoher Layer

|  |  |
| --- | --- |
| Klasse | Beschreibung |
| Device | Von einer *Device*-Instanz bekommt man alle neuen *Personen* indem man sich beim *NewPerson*-Event registriert. Nach dem Aufruf von *Start()* initialisiert das *Device* die Kinect und beginnt mit der Erkennung von *Personen*. Die bei *NewPerson* registrierten Funktionen werden jetzt mit dem *NewPersonEventArgs*-Parameter aufgerufen. Jener enthält jeweils eine neue Person.  Der *PersonActive*-Event wird gefeuert, wenn sich eine *Person* einloggt. Im *ActivePersonEventArgs*-Parameter wird diese *Person* mitgegeben. |
| Person | Bei einer *Person*-Instanz können Gestenreaktionen registriert werden. Momentant sind dies die folgenden:   * *OnZoom*: übermittelt den Zoomfaktor * *OnSwipe*: signalisiert eine Wisch-Geste * *OnWave*: signalisiert Winken |

## Schnittstellendefinition – Tiefer Layer

…