

Technikerarbeit 2012/13

Entwicklung einer Bibliothek
zur Emulation eines Touchpads
mit Hilfe des Microsoft Kinect Sensors

Philipp Grathwohl

Technikerarbeit 2012/13

Informationstechnik

Entwicklung einer Bibliothek
zur Emulation eines Touchpads
mit Hilfe des Microsoft Kinect Sensors

Verfasser: Philipp Grathwohl

Betreuer: Manfred Modrow

Elektronikschule - Oberhofer Straße 25 - 88069 Tettnang

Kurzbeschreibung

Mit dem Kinect Sensor hat Microsoft ein Gerät herausgebracht, mit dem es möglich ist ohne Controller oder sonstige Hardware Hilfsmittel, Eingaben für Computersysteme aufgrund menschlicher Bewegungen auszuwerten. Das ist nicht nur für die Spielekonsolennutzer, für die dieses Gerät hauptsächlich gedacht ist, eine Bereicherung. Auch für Software-Entwickler und Endanwender aus anderen Bereichen bietet das völlig neue Möglichkeiten zur Interaktion mit Computern.

Die Software, die es bisher auf dem Markt gibt, beschäftigt sich hauptsächlich mit dem Erkennen und Auswerten von Daten, die für die Steuerung der XBox360 Spielekonsole - ohne weiteren Controller - nötig sind. Also mit Gesten und Bewegungen, die mit dem ganzen Körper bzw. einzelnen Körperteilen wie Händen oder Füßen ausgeführt werden. Dafür gibt es unter anderem von Microsoft bereits ein eigenes Framework, das Microsoft Kinect Software Development Kit (SDK). Zusätzlich gibt es auch ein OpenSource Projekt, das einen ähnlichen Funktionsumfang bietet. Das OpenKinect Projekt.

Diese Frameworks bieten für ihre Zwecke sehr gute Leistungen. Es ist zum Beispiel möglich mit wenig Aufwand Skelett-Koordinaten von Menschen, die sich vor der Kamera befinden, aufzuzeichnen und auszuwerten. Diese Skelett-Koordinaten beinhalten markante Punkte des Körpers wie Kopf, Schultern, bestimmte Gelenke, Hüfte und so weiter. Bei Aufgaben die davon abweichen, bieten die Frameworks zwar die Rohdaten des Sensors, aber keine weiteren Hilfen wie z.B. Hand- oder Fingererkennung. Diese müssen von Entwicklern jeweils selbst implementiert werden.

Im Rahmen dieser Technikerarbeit wurde eine Bibliothek geschaffen, die Software Entwicklern eine Basis bietet um einen flachen Untergrund in ein Touchpad, wie man es z.B. von Notebooks kennt, zu verwandeln. Dafür wird der Kinect Sensor so ausgerichtet, dass er von oben den Untergrund (z.B. einen Tisch) erfassen kann. Aus dieser Position wird der Tiefensensor der Kinect verwendet, um Berührungspunkte mit dem Tisch zu erkennen. Die daraus resultierenden Daten werden dem Entwickler zur Verfügung gestellt. Zusätzlich bietet die Bibliothek eine einfache Gestenerkennung bzw. Aufnahmefähigkeit. So können simple Bewegungen Aktionen im Programmablauf auslösen und beeinflussen. Für die Erkennung und Weiterverarbeitung der Berührungspunkte aufgrund der gegebenen Rohdaten wurden verschiedene Bildverarbeitungsmechanismen angewendet. Dabei wurde auf die Verwendung externer Bibliotheken verzichtet um die Abhängigkeiten möglichst gering zu halten. Alle verwendeten Algorithmen wurden daher selbst implementiert.

Da bei einer Bibliothek die Akzeptanz anderer Entwickler einen maßgeblichen Einfluss auf die Verbreitung derselben hat, wurde darauf geachtet, die Application Programming Interface (API) möglichst übersichtlich zu halten um die Anwendung zu erleichtern. Außerdem wurde eine API-Dokumentation erstellt, die die einzelnen Methoden, Parameter und Rückgabewerte beschreibt.

Abstract

With the Kinect Sensor Microsoft introduced a piece of hardware with which you can control computer systems only with movements of parts or your whole body without depending on any additional controllers of any kind. This is a gain not only for console players for whom the Kinect was developed in the first place but also for developers and consumers of other areas. This opens a broad range of possibilities for interactions with computers.

All currently available software focuses on the recognition and tracking of player data used for controlling the XBox360 console without a controller in the usual way. It recognizes gestures and movements of the players body or parts of it like hands and feet. For this purpose there are several frameworks available. One is Microsoft's own Kinect SDK and the other is an open source project called OpenKinect. Both have similar features and a functional volume.

Both frameworks have very good capabilities when focusing on the above mentioned tracking mechanisms. It is very easy to track skeleton data of people in front of the camera. These skeletons contain coordinates of distinctive points of the body such as head, shoulders, various joints, hip and so on. For tasks other than skeleton tracking the frameworks only provide the raw data of the sensor but no other utilities for hand- or finger tracking. Those have to be done by the developer himself.

Within the course of this thesis a shared library was created which provides a base for developers to use a plain area as a touchpad like the ones known from notebooks. Therefore the Kinect Sensor has to be mounted on top of a table (or similar) so that it is able to capture the plain area from above. The depth sensor of the Kinect will be used to recognize touch points of fingers with the table. The resulting data is provided for the developers. Additionally the library is capable of basic gesture recording and recognition that can be used by developers for the execution of actions in their programs based on gestures entered by the user. For the recognition and processing of the raw sensor data several image processing algorithms were used. All of the used algorithms were implemented by myself and no third-party library was used to keep the dependencies at a minimum.

It is quite important for a shared library that it is easy to use and well documented. This way other developers are more likely to accept and use it. Therefore I tried to keep the API as easy as possible and also created an API-Documentation that describes the methods, parameters and return values.

Inhaltsverzeichnis

1	Einleitung	2
1.1	Motivation	2
1.1.1	Ziele	2
1.1.2	Projektabgrenzung	2
1.2	Hinweise zu den Quellenangaben	3
2	Vorbereitung	4
2.1	Programmiersprache	4
2.2	Entwicklungsumgebung	4
2.2.1	Erweiterungen	4
2.3	Qualitätssicherung	5
2.3.1	Versionskontrolle	5
2.3.2	Automatische Tests	5
2.4	Dokumentation	6
2.5	Backup	7
2.6	Der Kinect Sensor	7
2.6.1	Hardware	7
2.6.2	Software	9
3	Umsetzung	12
3.1	Einarbeitung und Planung	12
3.1.1	Programmablauf	13
3.1.2	Erweiterbarkeit	14
3.2	Projektstruktur	14
3.2.1	Touchdown.Core Interface	15
3.2.2	Touchdown.MicrosoftSDK	16
3.3	Aufbau der Hardware	17
3.4	Erkennung der Berührungspunkte	18
3.4.1	Überlegungen	18
3.4.2	Hintergrundmodell	20
3.4.3	Schritte	20
3.5	Gestenerkennung	30
3.5.1	Definition	30
3.5.2	Wiedererkennen von Gesten	31
3.5.3	Speichern und Laden	36
3.6	Probleme	37
3.6.1	Memory Leaks	37

3.6.2	TouchPoint Erkennung	37
3.6.3	Performance	38
3.7	Demo Applikation	39
3.7.1	Wizzard	39
3.7.2	Hauptfenster	44
3.7.3	Anmerkungen	45
4	Fazit	46
4.1	Einsatzzwecke	46
4.2	Weiteres Vorgehen	46
Abkürzungsverzeichnis		
Glossar		
Literaturverzeichnis		
Abbildungsverzeichnis		
Anhang		
Eidesstattliche Erklärung		

Dokumentation

Dokumentation des Projekts

“ENTWICKLUNG EINER BIBLIOTHEK ZUR EMULATION EINES
TOUCHPADS MIT HILFE DES MICROSOFT KINECT SENSORS“

1 Einleitung

1.1 Motivation

Dieses Projekt habe ich mir ausgesucht, da ich als Softwareentwickler jeden Tag mit Business- und Webanwendungen zu tun habe. Als Technikerarbeit wollte ich aber etwas machen, das mich persönlich interessiert und eine neue Herausforderung für mich darstellt. Dadurch, dass ich noch nie mit Bildverarbeitung und der Analyse von Bildinformationen in Echtzeit zu tun hatte, war dieses Gebiet absolutes Neuland für mich. Den Kinect Sensor hingegen kannte ich aus Anwendersicht schon von meiner Spielekonsole und habe mir bereits länger vorgenommen, mich mit diesem genauer auseinanderzusetzen. Die Technikerarbeit hat mir dafür die ideale Voraussetzung geboten.

1.1.1 Ziele

Ziel des Projektes war es eine Basis bereitzustellen, die es Entwicklern von .NET Programmiersprachen ermöglicht schnell und ohne viel Aufwand einen Kinect Sensor benutzen zu können, um damit mit ihren Programmen zu interagieren. Es wird dabei nicht der volle Funktionsumfang des Kinect Sensors ausgeschöpft, sondern die Konzentration liegt auf dem Erkennen von Berührungspunkten auf einem flachen Untergrund. Die Weiterverarbeitung dieser Berührungspunkte soll durch eine einfache Gestenerkennung erleichtert werden. Zusätzlich sollte die Bibliothek einfach für neue Funktionen erweiterbar sein.

Die Entwicklung sollte einfachen Qualitätsanforderungen entsprechen, um zum einen die Entwicklung selbst zu erleichtern und zum anderen die Bibliothek attraktiv für andere Entwickler zu gestalten. Dazu siehe Abschnitt 2.3.

1.1.2 Projektabgrenzung

Im Rahmen des Projekts wurde lediglich die Bibliothek als Basis für andere Applikationen entwickelt. Es wurde kein Produktivsystem erstellt, das von der Bibliothek Gebrauch macht. Es war auch nicht Ziel ein (Multi-)Touchpanel inkl. Treiber zu entwickeln. Das Projekt zielt lediglich darauf ab Funktionalität für .NET Applikationen bereit zu stellen, nicht aber eine generelle Eingabemethode auf Betriebssystemebene zu sein.

Um einige Fähigkeiten der Bibliothek demonstrieren zu können, wurde aber eine DEMO Applikation erstellt.

1.2 Hinweise zu den Quellenangaben

Einige der Quellen verweisen auf Artikel im Internet, Forenbeiträge oder Beiträge bei Frage- und Antwortseiten. Oft ist hierbei das Erscheinungsjahr und der Autor nicht angegeben bzw. werden nur Pseudonyme verwendet. Das heißt, dass bei einigen Quellenangaben weder Autor, noch Erscheinungsjahr angegeben ist.

2 Vorbereitung

Es wurden für die Umsetzung und Planung der Technikerarbeit verschiedene Werkzeuge ausgewählt und eingesetzt. Die Auswahl erfolgte nach persönlicher Einschätzung der Fähigkeiten, Benutzbarkeit und der schon vorhandenen Erfahrung mit dem speziellen Produkt.

2.1 Programmiersprache

Bei der Auswahl der Programmiersprache für dieses Projekt standen von Beginn an zwei Optionen zur Auswahl. Einmal C# und zum anderen Visual Basic .NET (VB.NET). Beide Programmiersprachen stammen aus dem Hause Microsoft und benutzen das .NET Framework. Darüber hinaus ist zu beachten, dass für beide Sprachen auch der OpenSource Compiler Mono und die dazugehörige Entwicklungsumgebung MonoDevelop zur Verfügung stehen. Auch wenn beide Sprachen prinzipiell in etwa den gleichen Funktionsumfang bieten, so hat C# bei vielen Entwicklern einen besseren Ruf. Das führt dazu, dass viele Beispiele, Fragen, Artikel und sonstige Informationen auf C# zugeschnitten sind. Dadurch, dass die Sprache beliebter ist, gibt es außerdem mehr C# Entwickler als es VB.NET Entwickler gibt. Vor diesem Hintergrund und der Absicht den Quellcode nach der Technikerarbeit evtl. nicht alleine weiterzuentwickeln, sondern als OpenSource Projekt zu veröffentlichen, habe ich mich dazu entschieden C# einzusetzen.

2.2 Entwicklungsumgebung

Durch die vorher getroffene Auswahl der Programmiersprache ergaben sich auch zwei Optionen für die einzusetzende Integrated Development Environment (IDE). Zum einen MonoDevelop und zum anderen das von Microsoft vertriebene Visual Studio. Durch das gute Zusammenspiel der Microsoft Programmiersprache, der passenden Entwicklungsumgebung und guten Vorkenntnissen mit eben dieser fiel die Entscheidung schnell auf Visual Studio. Ein weiterer ausschlaggebender Punkt waren die für Visual Studio verfügbaren Erweiterungen, die die Entwicklung vereinfachen und beschleunigen. Durch den vorhandenen Microsoft Developer Network Academic Alliance (inzwischen DreamSpark) (MSDNAA) Account der Elektronikschule war die Professional Edition in Version 2012 ebenfalls verfügbar.

2.2.1 Erweiterungen

Array Visualizer

Erleichtert das Debuggen indem es Arrays (auch mehrdimensional) visuell darstellt.

DPack

Erleichtert das Navigieren durch Codedateien und Methoden durch Tastenkombinationen und Suchfunktion.

NUnit Test Adapter

Ausführen von NUnit Unittests (siehe unten) in Visual Studio.

Productivity Power Tools 2012

Zusätzliche Features für den Visual Studio Editor. Zum Beispiel eine erweiterte Scrollbar mit zusätzlichen Informationen.

SHFB

Sandcastle Helpfile Builder. Für die Erstellung der API Dokumentation.

Visual Studio Tools for Git

Visual Studio Erweiterung für das Git-Versionskontrollsystem.

2.3 Qualitätssicherung

2.3.1 Versionskontrolle

Als Versionskontrollsysteme bezeichnet man Anwendungen, die Änderungen an Dateien mitverfolgen, historisieren, rückgängig machen und spezifische Stände festschreiben können. Diese Anwendungen werden vor allem bei der Software-Entwicklung eingesetzt, um die Arbeit der Entwickler zu erleichtern. Beim Einsatz in Teams koordinieren diese Anwendungen Änderungen von Entwicklern und versuchen sich überschneidende Änderungen zu erkennen und zu lösen. Aber auch als einzelner Entwickler bietet ein Versionskontrollsystem große Vorteile. So kann z.B. der Quellcode gefahrlos geändert und ausprobiert werden ohne Gefahr zu laufen, evtl. bestehende Funktionalität zu verlieren, da die Änderungen jederzeit auf einen vorherigen Stand zurückgenommen werden können. Außerdem lassen sich sehr einfach Backups des Quellcodes anfertigen.

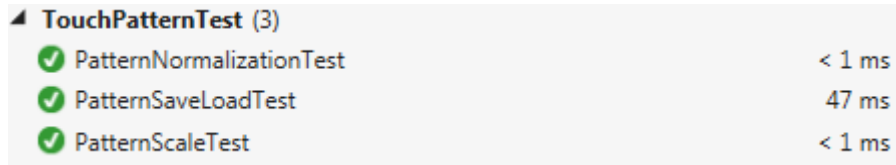
Für die Technikerarbeit wurde ein Versionskontrollsystem namens Git eingesetzt. Git ist ein sog. Distributed Version Control System (DVCS). Das bedeutet, dass es ohne einen zentralen Server auskommt. So können z.B. Entwickler untereinander Änderungen austauschen, ohne auf einen Server angewiesen zu sein. Dieses System kommt oft bei großen OpenSource Projekten zum Einsatz. Git selbst ist ebenfalls OpenSource.

Auch wenn kein Server nötig ist, wurde für das Projekt ein zentrales Repository auf einem privaten Network Attached Storage (NAS) eingerichtet. Dieses ist über das Internet erreichbar und wurde regelmäßig mit dem Repository auf dem Entwicklerrechner abgeglichen. So wurde automatisch eine Sicherung des Quellcodes auf einer separaten Maschine abgelegt. Dasselbe gilt für diese Dokumentation.

2.3.2 Automatische Tests

Zusätzlich zur Versionskontrolle wurden automatische Tests, sog. Unittests, für die Basisklassen und die Methoden des Projekts angelegt. Unittests sind programmierte Testmethoden,

die kontinuierlich (am besten nach jeder Änderung des betroffenen Quellcodes) Teile von Programmen aufrufen und deren Verhalten und Ergebnisse auswerten.



▲ TouchPatternTest (3)	
✓ PatternNormalizationTest	< 1 ms
✓ PatternSaveLoadTest	47 ms
✓ PatternScaleTest	< 1 ms

Abbildung 2.1: Erfolgreich ausgeführte Tests für den Bereich TouchPattern.

Quelle: eigene Angabe

So lässt sich schnell feststellen, ob dieser Teil des Programmes nach einer Änderung am Quellcode immer noch wie erwartet funktioniert oder nicht.

Unittests sind aus verschiedenen Gründen für eine Bibliothek unerlässlich. Zum einen können damit Teile des Programms oder der Bibliothek getestet werden, obwohl das Gesamtprojekt noch gar nicht einsatzfähig ist. Zum anderen gibt es dem Entwickler Sicherheit in der Form, dass ihm evtl. falsche Implementierungen oder Änderungen, die zu Fehlern führen, schneller bzw. überhaupt auffallen. Ein weiterer Punkt ist, dass viele Tests bzw. eine hohe Testabdeckung des Codes das Vertrauen anderer Entwickler in das eigene Produkt fördert. Des weiteren helfen Tests bzw. das schreiben dieser, den Aufbau der API übersichtlich zu halten. Denn wenn die Aufrufe der API für den Entwickler selbst (im Test) schon unübersichtlich werden, wird es für andere Entwickler auch nicht leichter.

Für die Technikerarbeit wurde das Unittest-Framework NUnit eingesetzt. Es ist ein bewährtes Framework inkl. Testrunner. Das zugehörige Plugin für Visual Studio ist auch verfügbar. Siehe Seite 4.

2.4 Dokumentation

```
/// <summary>
/// Compares all <see cref="TouchPoint"/>s of all <see cref="SimpleTouchFrame"/>s of two
/// <see cref="TouchPattern"/>s.
/// </summary>
public class SimpleTouchPatternComparer{
```

Abbildung 2.2: XML Kommentar mit Verweisen auf andere Klassen.

Quelle: eigene Angabe

Um Entwicklern den Einstieg und das Programmieren zu erleichtern, wurde für die erstellte Bibliothek eine API-Dokumentation erstellt. In dieser werden alle Klassen, Methoden, Parameter, Properties und Rückgabewerte kurz beschrieben. Diese Beschreibungen finden sich im Quellcode des Projekts wieder. Dabei handelt es sich um sog. XML-Kommentare. Diese Kommentare können zu Klassen, Methoden und Properties hinzugefügt werden und enthalten die vorhin genannten Informationen. Diese Kommentare beginnen mit „///“ vor jeder Zeile und enthalten spezielle XML tags. Dabei kann man, wie in Abbildung 2.2 zu sehen,

auch auf andere Codeobjekte (z.B. Klassen) verweisen.

Diese erstellte Hilfe ist einmal in kompilierter Form als CHM Datei vorhanden und einmal als HTML Hilfe.







	Touchdown.Core.PatternRecognition Namespace	Touchdown Touchpanel Emulator - API Help
Contains classes and methods used for creating, saving, loading and recognizing patterns.		
Classes		
	Class	Description
	SimpleTouchFrameComparer	takes two frames and compares their touchpoints. If one frame has more than one touchpoint
	SimpleTouchPatternComparer	Compares all TouchPoints of all SimpleTouchFrames of two TouchPatterns .
	TouchPatternRecognizedEventArgs	Event arguments for when a pattern was recognized.
	TouchPatternRecognizer	Tries to recognize patterns
	TouchPatternRecordingEventArgs	Event arguments for when a pattern is currently recorded. Fired when a frame was added to the patterns.

Abbildung 2.3: Ausschnitt der kompilierten API-Dokumentation.

Quelle: eigene Angabe

2.5 Backup

Dadurch, dass sowohl das Projekt selbst als auch die Dokumentation unter Versionskontrolle via Git stehen, und dadurch nach jeder Änderung zusätzlich auf ein NAS übertragen werden, wurde kein weiteres Backup angelegt.

2.6 Der Kinect Sensor

2.6.1 Hardware

Kinect ist ein Sensor, der zur Steuerung der Microsoft Spielekonsole Xbox360 entwickelt wurde. Im Gegensatz zu anderen Spielekonsolenentwicklungen ist bei der Steuerung via Kinect bei der Xbox360 kein regulärer Controller mehr nötig. Eingaben werden durch Bewegungen und Gesten, die der Spieler mit seinem Körper oder Teilen davon (z.B. nur Hände) ausführt, erkannt und umgesetzt. Auch Sprachsteuerung der Konsole ist mit dem Kinect Sensor möglich. Die Verbindung mit dem Computer oder einer Xbox360 wird über ein Kabel mit Universal Serial Bus (USB) Anschluss hergestellt.

Die Kinect besteht aus mehreren Sensoren, die im Folgenden kurz angesprochen werden. Die Technikerarbeit befasst sich allerdings hauptsächlich mit dem Tiefensensor der Kinect.



Abbildung 2.4: Frontansicht des Kinect Sensors.

Quelle: <http://msdn.microsoft.com/en-us/library/hh438998.aspx>

RGB Kamera

Die RGB Kamera der Kinect liefert Farb- bzw. Infrarotbilder, die mit der regulären Kamera aufgenommen wurden. Der Betrieb dieser kann durch verschiedene Modi eingestellt werden. Die vorhandenen Betriebsmodi sind folgende: (Auf die Eigenschaften der verschiedenen Bildformate wird nicht näher eingegangen, da sie für dieses Projekt keine Relevanz haben.)¹

InfraredResolution640x480Fps30 Liefert einen Infrarot Datenstrom mit einer Auflösung von 640x480 Pixeln bei einer Framerate von 30 Frames per Second (FPS). Die Infrarotbilder werden als Graustufenbilder dargestellt.

RawBayerResolution1280x960Fps12 Liefert einen Datenstrom im Raw-Bayer Format. Bei einer Auflösung von 1280x960 Pixeln werden 12 FPS bereitgestellt.

RawBayerResolution640x480Fps30 Wie im letzten Modus liefert auch dieser einen Datenstrom im Raw-Bayer Format allerdings mit einer niedrigeren Auflösung von 640x480 Pixeln und einer höheren Framerate von 30 FPS

RawYuvResolution640x480Fps15 Bei einer Auflösung von 640x480 Pixeln und einer Framerate von 15 FPS liefert dieser Modus Raw-YUV Daten.

RgbResolution1280x960Fps12 Der RGB Modus liefert Bildinformationen mit einer Auflösung von 1280x960 Pixeln bei einer Framerate von 12 FPS.

RgbResolution640x480Fps30 Wie der vorgehende Modus liefert auch dieser RGB Daten. Bei 30 FPS werden Bilddaten mit einer Auflösung von 640x480 Pixeln geliefert. Dieser Modus wird von der Bibliothek für den Bilddatenstrom voreingestellt. Auch wenn die Bilddaten momentan noch nicht verwendet werden.

YuvResolution640x480Fps15 Der letzte verfügbare Modus liefert YUV Bilddaten mit einer Auflösung von 640x480 Pixeln und einer Framerate von 15 FPS

¹Vgl. MSDN Kinect Informationen zu Color Image Format [1]

Mikrophon-Array

Ein weiteres Feature des Kinect Sensors ist das eingebaute Mikrophon-Array bestehend aus vier einzelnen Mikrofonen. Diese werden für Spracherkennung verwendet, wobei durch die vier Mikrophone auch die Richtung aus der gesprochen wird, erkannt werden kann. Das Microfonarray wird in diesem Projekt nicht verwendet.

Beschleunigungsmesser

Ebenfalls in der Kinect enthalten ist ein sog. Beschleunigungsmesser. Dieser gibt Werte zur aktuellen Ausrichtung des Sensors an. Der Beschleunigungsmesser wird in diesem Projekt nicht verwendet.

Neigungsmotor

Zur optimalen und automatischen Ausrichtung des Sensors ist ein kleiner Motor im Fuß des Sensors eingebaut, mit dem sich die Neigung einstellen lässt. Da der Sensor für dieses Projekt immer gleich ausgerichtet sein muss, bzw. seine Ausrichtung während des Betriebs nicht verändern sollte, wird auch der Motor nicht eingesetzt.

Tiefensensor

Der Tiefensensor selbst besteht aus zwei Komponenten. Einem Infrarot-Emitter und einem Infrarotsensor. Der Sensor ermittelt die Tiefeninformationen anhand der reflektierten Infrarotstrahlen. Die Reichweite des Tiefensensors befindet sich zwischen 800mm und 4000mm. Wie auch bei dem RGB Sensor können auch beim Tiefensensor verschiedene Betriebsmodi ausgewählt werden.

Resolution320x240Fps30 Diese Einstellung liefert einen Datenstrom für die Tiefeninformationen mit einer Auflösung von 320x240 Pixeln bei 30 FPS.

Resolution640x480Fps30 Diese Einstellung liefert einen Datenstrom für die Tiefeninformationen mit einer Auflösung von 640x480 Pixeln bei 30 FPS. Diese Einstellung wird von der Bibliothek standardmäßig verwendet.

Resolution80x60Fps30 Diese Einstellung liefert einen Datenstrom für die Tiefeninformationen mit einer Auflösung von 80x60 Pixeln bei 30 FPS.

Die Positionen der einzelnen Sensoren innerhalb der Kinect sind der Abbildung 2.5 zu entnehmen.

2.6.2 Software

Es gibt zwei Frameworks die hauptsächlich für die Entwicklung mit dem Microsoft Kinect Sensor verwendet werden.

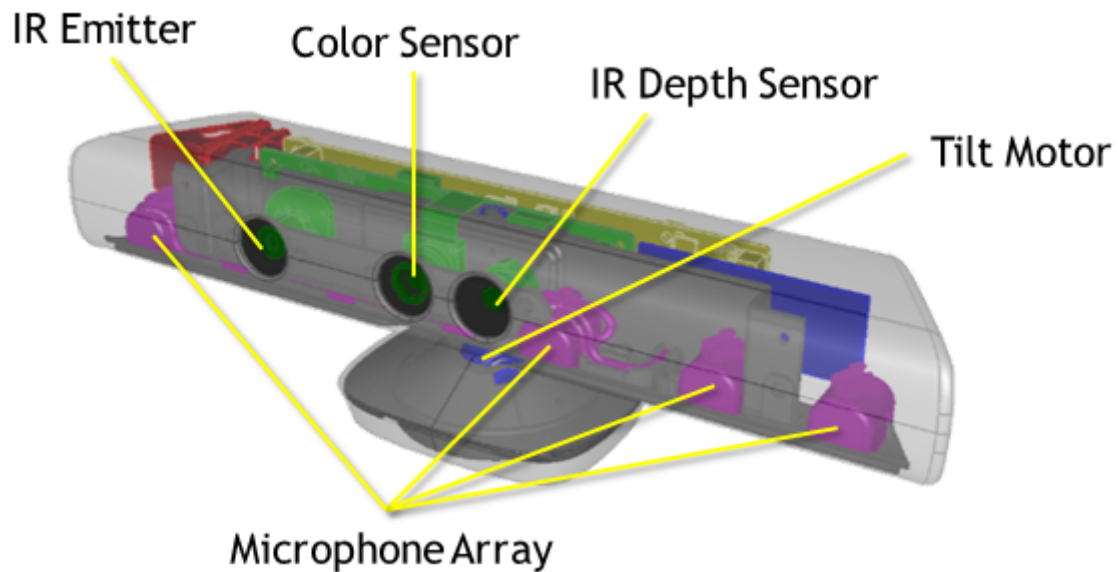


Abbildung 2.5: Aufteilung der einzelnen Sensoren in der Kinect.

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

Microsoft Kinect SDK

Das Microsoft Kinect SDK ist das Entwicklungswerkzeug, das direkt vom Hersteller des Sensors zur Verfügung gestellt wird. Es wird für .NET Programmiersprachen genutzt. Offiziell unterstützt werden nur Microsoft Betriebssysteme. Der Quellcode für das SDK ist nicht öffentlich zugänglich. Inzwischen gibt es auch das sog. Microsoft Kinect Studio, mit dem sich RGB und Tiefeninformationen aufnehmen und wiedergeben lassen, was die Entwicklung erleichtert, indem man immer den gleichen Anwendungsfall nachstellen kann. Allerdings muss der Sensor dazu immer mit dem Computer verbunden sein. Die umfangreiche Microsoft Developer Network (MSDN) Dokumentation ist in vielen Fällen hilfreich und bietet gute Beispiele und Erklärungen.

OpenKinect

OpenKinect bezeichnet eine Community, die sich zur Aufgabe gemacht hat eine Basis für die Entwicklung mit der Kinect zu ermöglichen. Dazu gehören eine Dokumentationsplattform, Mailinglisten und selbstverständlich ein Framework (libfreenect) inkl. Wrapper-Bibliotheken für viele gängige Programmiersprachen (unter anderem C#). Das Framework bietet einen ähnlichen Umfang wie das Microsoft SDK und wird von verschiedenen Projekten bereits aktiv verwendet.²

Auswahl

Da beide Frameworks inzwischen großen Andrang finden und jeweils sehr gut funktionieren, wurde die Entscheidung getroffen, die Bibliothek mit beiden kompatibel zu machen. So

²OpenKinect Project Community [2]

können durch die .NET Plattform und die Unabhängigkeit des verwendeten Frameworks möglichst viele Entwickler profitieren. Zum Aufbau der Bibliothek, die das ermöglicht siehe Abschnitt 3.2.

3 Umsetzung

Bevor mit der Umsetzung des Projektes begonnen hat, wurde zuerst ein ansprechender und markanter Projektname bzw. Codename benötigt. Aufgrund dessen, dass die Bibliothek ein Emulator für ein *Touch-Pad* sein soll, wurde der Projektname *Touchdown* gewählt. Das passende Logo, ein American Football, wie in Abbildung 3.1 zu sehen ist, war auch schnell gefunden.



Abbildung 3.1: Touchdown Project Logo.

Quelle: Jason M. Simanek <http://openclipart.org/detail/102853/football-by-simanek>

3.1 Einarbeitung und Planung

Die Planung des Projekts hat sich in diesem Fall als recht schwierig erwiesen, da die benötigten Methoden und Algorithmen für die Bildverarbeitung allesamt neu für mich waren. Daher war es schwer einzuschätzen, was alles zu tun ist und wie lange sich die jeweiligen Aufgaben hinziehen. Aus diesem Grund hat das Projekt erst mal mit langen Recherchen und der Einarbeitung in die Programmierung der Kinect und Bildverarbeitung im Allgemeinen begonnen. Viele der gelesenen Artikel, Blogs, Hilfeseiten, Foren u.s.w. lassen sich keinen speziellen Themen dieser Dokumentation zuordnen und finden sich deshalb nur als Verweise im Literaturverzeichnis.

Nachdem ein gewisses Grundverständnis vorhanden war, wurde ein UML Diagramm angelegt, das die grobe Struktur der verwendeten Klassen und deren Methoden skizzieren sollte. Dieses UML Diagramm findet sich im Anhang wieder. Es erhebt keinen Anspruch auf Korrektheit oder Vollständigkeit, es sollte lediglich als Gedächtnisstütze dienen bzw. die ersten Gedankengänge festhalten.

3.1.1 Programmablauf

Der im Vorhinein gedachte Programmablauf ist der Abbildung 3.2 zu entnehmen.

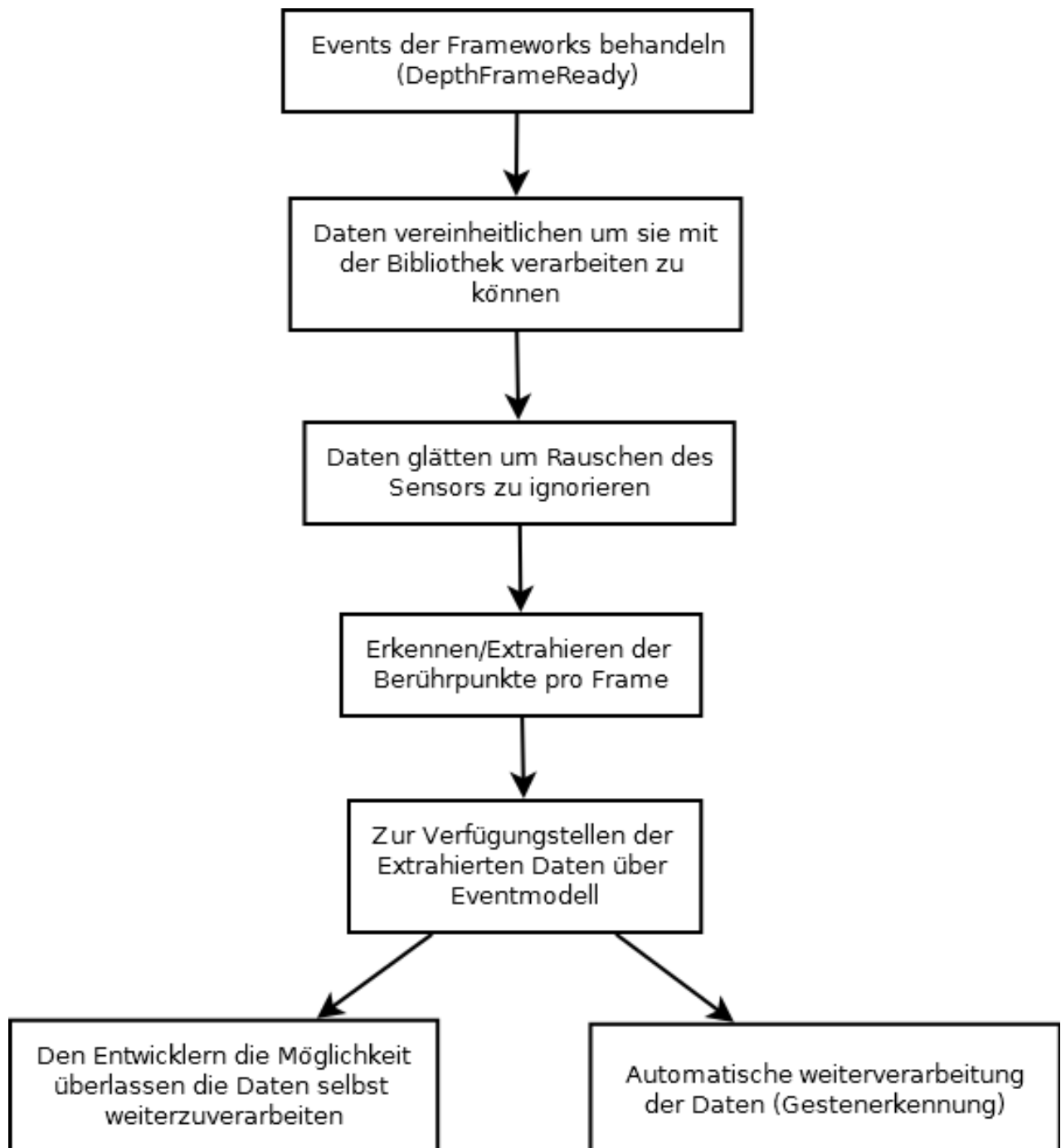


Abbildung 3.2: Skizzierter Programmablauf

Quelle: eigene Angabe.

3.1.2 Erweiterbarkeit

Ein Hauptaugenmerk der Programmierung sollte darauf liegen, die Bibliothek so zu gestalten, dass sie einfach erweiterbar ist. Es sollte also auch im Nachhinein möglich sein Methoden für z.B. die Erkennung der Berührungspunkte zu verändern, oder durch eigene Implementierungen auszutauschen. Dies gilt nicht nur für die Erkennung der Berührungspunkte, sondern für alle Bereiche der Bibliothek.

3.2 Projektstruktur

Durch die gewünschte Unterstützung der beiden Frameworks wie in Abschnitt 2.6.2 beschrieben, ist es nötig eine spezielle Projektstruktur einzuhalten. Es müssen bestimmte Methoden in eigene Assemblies ausgelagert werden. Dies ist nötig um keine ungewünschten Abhängigkeiten zu schaffen. Wenn z.B. das Microsoft SDK verwendet werden soll, ist es weder nötig noch gewünscht Abhängigkeiten zum OpenKinect Framework im Projekt zu haben. Daher wurde die Projektstruktur wie in Abbildung 3.3 gewählt.

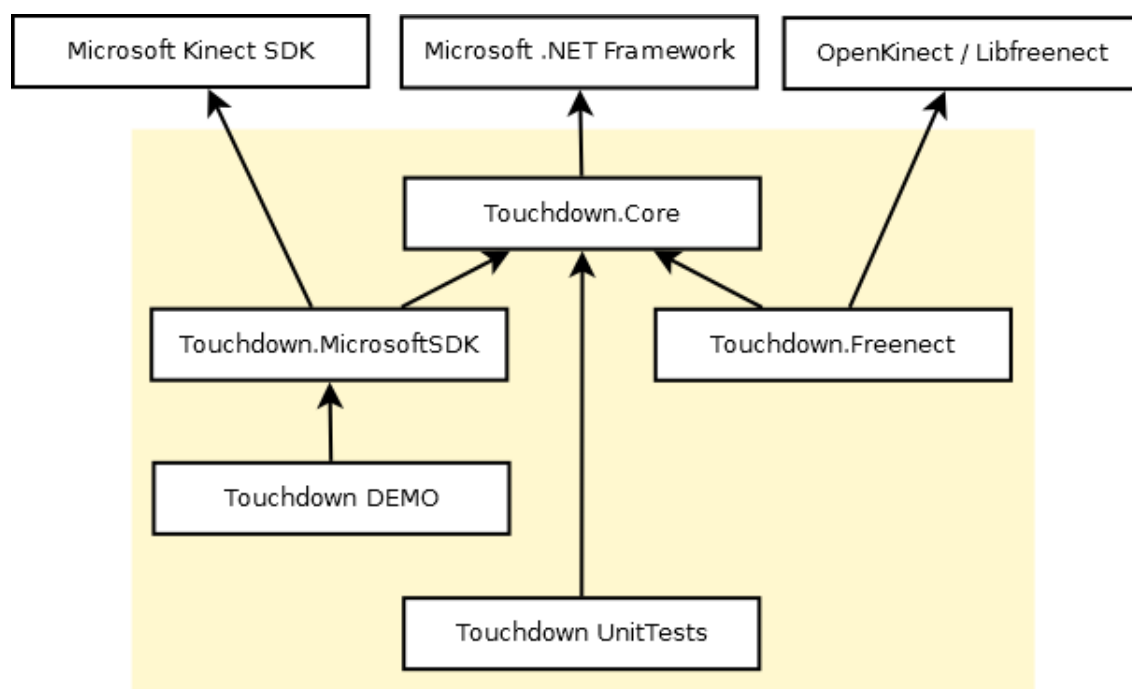


Abbildung 3.3: Schematische Darstellung der Projektstruktur

Quelle: eigene Angabe.

Diese Projektstruktur wirkt sich in sofern aus, dass jeder der benannten Bereiche zu eigenen Assemblies werden und daher jeweils separate Abhängigkeiten haben. Die Touchdown.Core Bibliothek hat lediglich Abhängigkeiten zum .NET Framework, während die Touchdown.MicrosoftSDK zusätzlich von der Touchdown.Core sowie dem Microsoft SDK abhängig ist. Die Touchdown.Freenect hingegen ist von der Touchdown.Core und der libfreenect Library abhängig.

Die Bereitstellung eines DEMO Programms erfordert, dass für mindestens eines der beiden - in Abschnitt 2.6.2 genannten - Frameworks die Schnittstellen zur tatsächlichen Verwendung nicht nur von der Bibliothek angeboten, sondern auch implementiert werden. Hierfür wurde nur das Microsoft SDK benutzt. Ebenfalls in Abbildung 3.3 erkennbar, die Unittests beziehen sich ausschließlich auf die Core Implementierung.

Die Struktur der Projektmappe in Visual Studio kann Abbildung 3.4 entnommen werden.

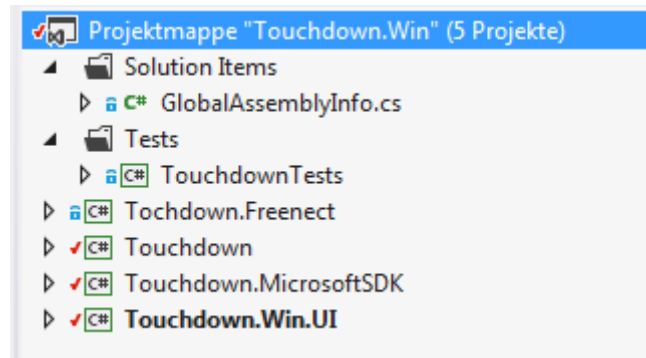


Abbildung 3.4: Projektstruktur der Visual Studio Solution

Quelle: eigene Angabe.

3.2.1 Touchdown.Core Interface

Der Projektteil Touchdown.Core beinhaltet den Großteil der Implementierungen, die für das Projekt erforderlich waren. Unter anderem auch die Grundlage dafür, dass die unterschiedlichen Frameworks eingesetzt werden können. Um dies zu erreichen, wurde eine zusätzliche Abstraktionsschicht eingeführt. Diese erfordert, dass die Unterprojekte Touchdown.MicrosoftSDK und Touchdown.Freenect jeweils ein Interface implementieren, um einen Wrapper für den tatsächlichen Sensor bereitzustellen. Diese Wrapperklassen beinhalten jeweils die von den Frameworks bereitgestellten Sensoren-Klassen. Sie sind dafür verantwortlich die Datenströme so aufzubereiten, dass sie von der Touchdown.Core Bibliothek verarbeitet werden können. Diese geht davon aus, dass die Tiefeninformationen in einem 32Bit Integer Array in Millimetern zu erhalten. Für Farbbildinformationen wird hingegen ein Byte Array erwartet. Um zusätzliche Informationen wie Erstellungszeit, Höhe und Breite festzuhalten bzw. weitere Methoden bereitzustellen, die die Daten direkt verarbeiten und visualisieren können, werden diese wiederum jeweils in Objekte vom Typ DepthFrame oder RGBFrame verpackt. Diese werden von der Core Bibliothek verarbeitet.

Die Implementierung der Wrapper übernehmen neben dem Vereinheitlichen der Datenströme zusätzlich die Aufgabe den Sichtbereich des Sensors ggf. auf einen vordefinierten Bereich einzuschränken. Dabei werden Daten, die außerhalb des Sichtbereichs liegen nicht bei der Aufbereitung der Daten für die Core Bibliothek berücksichtigt. Dies hat den Vorteil, dass bei einem kleineren zu verarbeitenden Bereich die benötigte Rechenleistung niedriger und somit die Performance besser wird. Außerdem lassen sich so Bereiche des Bildes, die anfällig für Störungen sind, ausblenden. Solche Bereiche sind z.B. Tischkanten oder Gegenstände, die sich am Rand der zu verwendenden Fläche befinden. Wird nicht explizit ein solcher Bereich

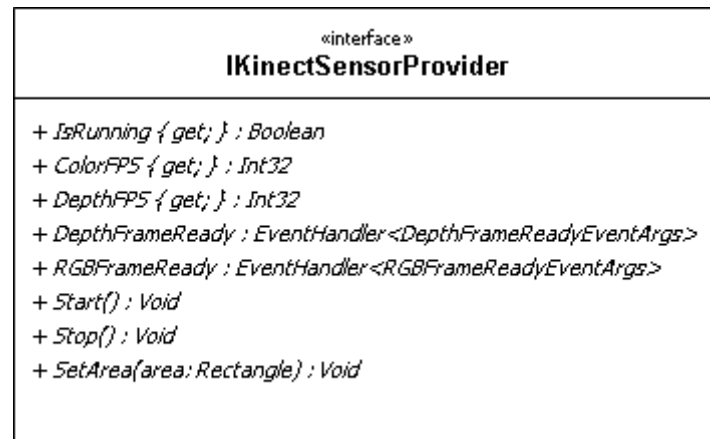


Abbildung 3.5: Interface für Wrapperklassen. Lieferant für RGB- und DepthFrames.

Quelle: eigene Angabe.

vom Entwickler definiert, wird keine Einschränkung vorgenommen. Weitere kleinere Aufgaben des Sensor Wrappers sind Zählen der Frames für Tiefen- und Bildinformationen, und Bereitstellen einer Möglichkeit die Sensoren zu Starten und zu Stoppen (=keine weiteren Frames zu liefern)

Momentan wurde nur die Wrapperklasse für das Microsoft SDK implementiert. Die Touchdown.Freenect Variante wurde nicht getestet und ist derzeit nicht funktionsfähig.

3.2.2 Touchdown.MicrosoftSDK

Um die nativ vom Sensor gegebenen Daten in RGBFrames und DepthFrames umzuwandeln, sind beim Microsoft SDK folgende Schritte nötig. Die Bildinformationen müssen vom Microsoft gegebenen Objekt in ein Byte Array kopiert werden. Danach kann der RGBFrame erstellt werden. Anpassungen der Daten sind nicht nötig. Für die Tiefeninformationen ist das anders. Diese werden vom Microsoft Framework in einem Objekt geliefert, was die Daten erst mal nur in ein 16Bit Array kopieren kann. Diese Tiefeninformationen liegen bereits als Distanz in Millimetern zum Sensor vor, mit der Besonderheit, dass sie pro Wert zusätzlich die Information enthalten ob und wenn ja zu welchem Spieler ein Wert gehört. Dies ist sinnvoll, wenn die Kinect im regulären Sinne, nämlich zum Erfassen von ganzen Körpern beziehungsweise von Spielern genutzt wird. Da in diesem Fall aber immer nur einzelne Hände, die von dem Sensor auch nicht als Spieler gewertet werden, im Bild zu sehen sind, müssen diese Informationen erst entfernt werden. Um dies zu erreichen müssen alle Werte mit dem Binärshift Operator um eine gewisse Anzahl Bits, die der Variable `DepthImageFrame.PlayerIndexBitmaskWidth` zu entnehmen ist, nach rechts gerückt werden. Der Grund hierfür ist, dass die Informationen zum Player-Index in den niederwertigen Bits des Wertes stehen. Siehe untenstehender Pseudocode.

```

1  int DistanceInMilimeter = OriginalDistance >> DepthImageFrame.PlayerIndexBitmaskWidth;

```

Sind von jedem einzelnen Wert die Player-Index Information entzogen worden und alle Werte in einen 32Bit Integer Wert umgewandelt, kann aus diesen Daten ein DepthFrame

erstellt werden.

3.3 Aufbau der Hardware

Der Sensor wird ca. einen Meter über der Fläche, die als Untergrund dient, montiert, so dass der Tiefensensor im Winkel von möglichst 90° in Richtung der Fläche zeigt. Der Abstand ist teilweise durch die Reichweite des Sensors vorgegeben, da dieser einen Mindestabstand von 800 Millimetern benötigt, um verwendbare Daten zu erhalten. Für einen Beispielaufbau mit einem Tisch, siehe Abbildung 3.6.

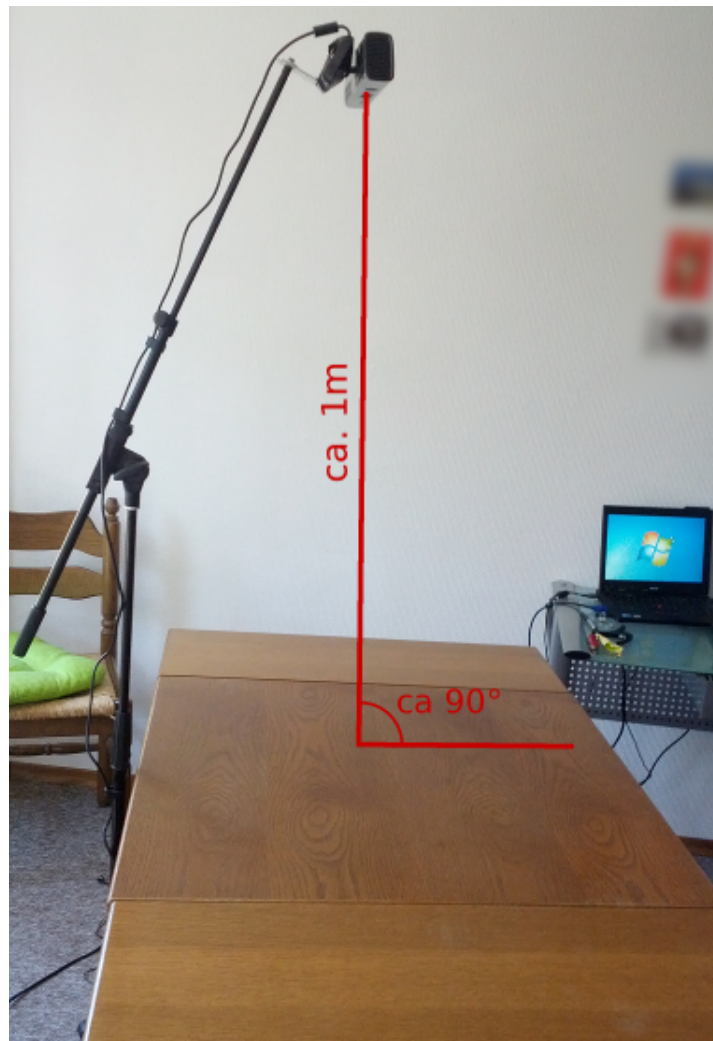


Abbildung 3.6: Aufbau des Sensors

Quelle: eigene Angabe.

Die Kinect wurde mit Hilfe eines Mikrofonständers in Position gebracht. Zur Verbindung der Kinect mit dem Mikrofonständer wurde ein handelsüblicher 90° Winkel aus dem Baumarkt verwendet.

3.4 Erkennung der Berührungspunkte

3.4.1 Überlegungen

Um Berührungspunkte der Finger mit dem Tisch erkennen zu können, muss als erstes die Überlegung angestellt werden, wie diese Bereiche aus den Tiefeninformationen ausgelesen werden können. Dafür gibt es einige Möglichkeiten. Zum einen könnte man versuchen eine Erkennung der Hände unter dem Sensor zu implementieren, um so herauszufinden wo sich die Fingerspitzen befinden. Wenn sich diese dann nahe genug an der Oberfläche befinden, könnte das als Berührungspunkt gezählt werden. Diese Methode hätte den Vorteil relativ genau die Position der Finger bestimmen zu können. Außerdem könnte man Berührungspunkte, die von Fingern ausgelöst werden von denen, die durch kleine Gegenstände entstehen, unterscheiden. Allerdings ist eine solche Erkennung der Hände nicht ganz einfach und erfordert viel Zeit für die Implementierung.

Eine andere, für den Anfang einfachere und nicht so zeitaufwendige Implementierung wäre hingegen das so genannte Thresholding (engl. Schwellenwertverfahren). Dabei werden Ober- und Untergrenzen für die Tiefenwerte, die vom Sensor geliefert werden, festgelegt. Alle Werte, die über der Obergrenze oder unter der Untergrenze liegen, werden verworfen. Alle anderen Punkte gelten als berührt.

Letztere Methode wurde in der Bibliothek implementiert, allerdings wurde auch hier dar-

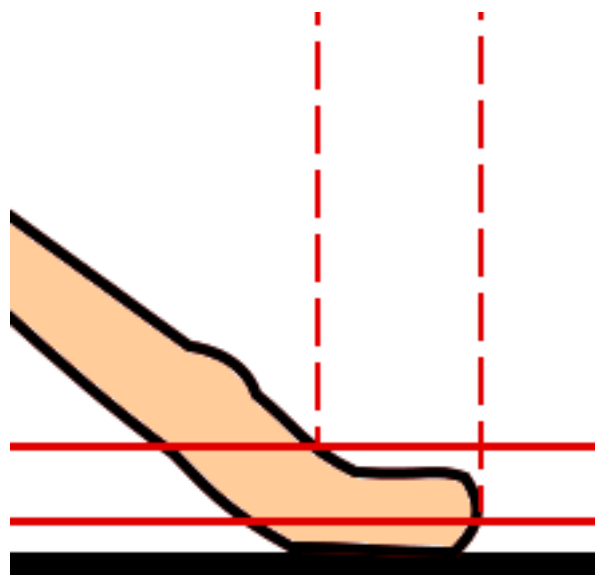


Abbildung 3.7: Skizzierter Finger. Ober- und Untergrenzen festgelegt.

Quelle: eigene Angabe.

auf geachtet, dass die erste Methode nachgerüstet werden kann. Dafür kam wiederum ein generisches Interface zum Einsatz, welches von einer Klasse implementiert werden muss, um TouchFrames zurück zu liefern. TouchFrames sind dabei, ähnlich wie Depth- oder RGBFrames, Container für einen Datenstrom, welche in diesem Fall Informationen über berührte Punkte des betrachteten Bereichs enthalten. Die Skizze 3.7 zeigt die Seitenansicht eines Fingers, mit eingezeichneten Ober- und Untergrenzen. Die Fingerspitze liegt hierbei zwischen

den Grenzen, während der Rest des Fingers darüber liegt. Die Untergrenze dient auch dazu vom Sensor verursachtes Rauschen zu ignorieren.

Die Klasse, die für das Thresholding und Extrahieren der Berührungspunkte verantwortlich ist, implementiert das oben genannte Interface und liefert dabei eine abgeleitete Version eines Frames zurück. Diese Frames sind vom Typ SimpleTouchFrame. Die Klasse ist der SimpleTouchAreaObserver. Das *Simple* Präfix soll dabei verdeutlichen, dass es sich hierbei um eine sehr einfache Variante der Erkennung handelt. Eine nachträgliche Erweiterung um eine genauere Möglichkeit ist aber dennoch denkbar.

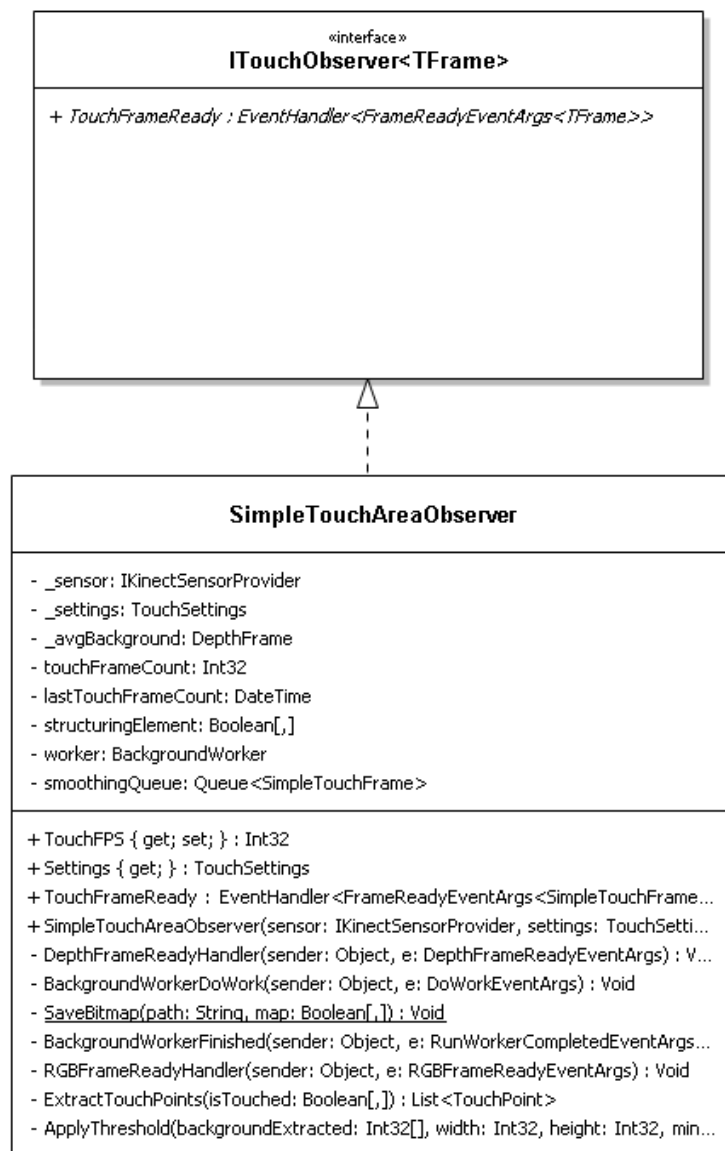


Abbildung 3.8: UML Ausschnitt - SimpleTouchAreaObserver.

Quelle: eigene Angabe.

3.4.2 Hintergrundmodell

Beide in Abschnitt 3.4.1 genannten Möglichkeiten brauchen aber - um zu erkennen ob der Tisch berührt ist - die Information wie weit der Tisch vom Sensor entfernt ist. Um dieses Problem zu beheben, wird auf ein einfaches Hintergrundmodell zurückgegriffen. Dabei wird eine gewisse Anzahl von Frames vom Programm aufgenommen und danach der Durchschnittswert pro Pixel aus allen Frames errechnet. Theoretisch würde auch ein einzelner Frame ausreichen, aber durch den Durchschnittswert spielen Störungen, die eventuell während der Aufnahme vorkommen eine kleinere Rolle.

Das Hintergrundmodell bezeichnet also im Prinzip einen DepthFrame, der den Hintergrund möglichst ohne Störung darstellt. Wie das Hintergrundmodell angewendet wird, wird im nächsten Abschnitt erläutert.

3.4.3 Schritte

Folgende Schritte wurden geplant bzw. angewendet, um die Berührungspunkte aus den jeweils aktuellen DepthFrames zu extrahieren.

Werte Glätten

Um bei den folgenden Schritten bessere bzw. klarere Ergebnisse zu erhalten, war es geplant die Frames, die verarbeitet werden, zu Beginn zu glätten. Das heißt es wurde versucht vom Sensor verursachtes Rauschen zu minimieren. Dabei wurden mehrere Ansätze verfolgt. Die Methoden sollen kurz angesprochen werden. Jedoch ist die detaillierte Funktionsweise nicht Teil der Dokumentation, da diese zu umfangreich wäre. Zu beachten ist, dass das Glätten der Werte zwar als erster Schritt der Kette vermerkt ist, aber erst nach der Implementierung der eigentlichen Erkennung ausprobiert wurde. Daher beziehen sich die angemerkten Ergebnisse auf die Qualität der extrahierten Berührungspunkte.

Weighted Moving Average Diese Methode wurde einem Artikel ¹ entnommen, welcher beschreibt, wie DepthFrames der Kinect geglättet werden können, um ein besseres Ergebnis zu erhalten. Dabei wird für jedes Pixel der Median der benachbarten Pixel gesetzt. Hierbei wird nicht nur der aktuelle Frame betrachtet, sondern eine einstellbare Anzahl von Frames, wobei der neuste Frame am höchsten gewichtet wird. Allerdings wird im Artikel auch erwähnt, dass diese Methode nur Verbesserungen im Bezug auf das Rendering der Daten zur Darstellung (z.B. eines Videos) mit sich bringt. Für die Weiterverarbeitung der Daten, wie es bei der Extraktion der Berührungspunkte der Fall ist, ist sie nur bedingt geeignet.

Nichtsdestotrotz wurde die Methode ausprobiert. Nachdem festgestellt wurde, dass dies keinerlei Verbesserungen mit sich bringt, wurde die Implementierung wieder entfernt.

Bilateral Filtering Eine weitere Methode, die in Betracht gezogen wurde, ist das bilaterale Filtern der Pixelwerte. Ähnlich wie bei der vorhergehenden Methode werden hierbei benachbarte Pixel in einem bestimmten Umkreis betrachtet. Allerdings spielen hier nicht nur die

¹Smoothing Kinect Depth Frames in Real-Time [4]

Werte der benachbarten Pixel eine Rolle, sondern auch deren Entfernung zum gerade betrachteten Pixel. Je weiter weg das Nachbarpixel ist, desto weniger Einfluss hat es darauf, was dem aktuellen Pixel am Ende für ein Wert zugewiesen wird.²

Auch diese Methode hat nach der Anwendung keine Verbesserung gezeigt. Zusätzlich wurde die Performance beeinflusst, so dass die Framerate auf bis zu 10 FPS gesunken ist. Die Implementierung wurde auch hier wieder rückgängig gemacht.

Nachdem die ausprobierten Methoden keine Besserung der Erkennung mit sich brachten, wurde versucht das Verbessern an anderer Stelle zu bewerkstelligen.

Hintergrundmodell

Um für die weitere Verarbeitung die Tiefenabstände relativ zum Untergrund anstatt relativ zum Sensor zu erhalten, kommt nun das zuvor erstellte Hintergrundmodell zum Einsatz. Jeder einzelne DepthFrame, der bearbeitet wird, wird jeweils vom Hintergrundmodell abgezogen. Das bedeutet es wird jeder Pixelwert des Frames von dem Pixelwert des entsprechenden Pixels im Hintergrundmodell abgezogen. Vereinfacht ausgedrückt würde es also so aussehen:

```
1 VordergrundFrame = Hintergrundmodell - AktuellerFrame;
```

Dabei wird der aktuelle Frame vom Hintergrundmodell abgezogen, da die Werte im aktuellen Frame in der Regel kleiner sind, als die des Hintergrundmodells. Das liegt daran, dass zum Beispiel eine Hand, die sich im Bild bewegt immer eine kleinere Distanz zum Sensor hat, als der Hintergrund. So entstehen beim daraus erzeugten Vordergrund-Frame keine negativen Werte für den Bereich der Hände. Es können zwar trotzdem negative Werte entstehen, diese werden aber durch Rauschen des Sensors verursacht, sind meist Teil des Hintergrunds und können daher also leicht herausgefiltert werden.



Abbildung 3.9: Erstellung des Vordergrundframes. Links Ausgangsbild, rechts Vordergrund.

Quelle: eigene Angabe.

In Abbildung 3.9 ist links ein Ausschnitt der Visualisierung eines DepthFrames abgebildet. Rechts daneben eine Visualisierung des Ergebnisses, der Subtraktion des Hintergrundes. Durch die geringen Distanzunterschiede, ist die Hand im Vordergrund nicht sehr gut zu erkennen. Bei diesen geringen Unterschieden und fließenden Kanten ist es sehr schwierig gute Ergebnisse mit Thresholding zu erzielen. Oft werden relevante Punkte dadurch weg

²Vgl. Bilateral Filter Algorithm. [5]

gefiltert oder unwichtige Punkte behalten, je nachdem welche Schwellenwerte man wählt. Daher war es nötig vor dem Thresholding noch einen Zwischenschritt einzulegen.

Scaling

Scaling bezeichnet eine Methode der Bildverarbeitung, die normalerweise dazu verwendet wird Bilder aufzuhellen oder zu verdunkeln. Beim Scaling werden die einzelnen Pixelwerte eines Bildes mit einer Konstante multipliziert. Ist diese Konstante kleiner als 1, so wird das Bild abgedunkelt. Ist sie größer als 1 wird das zu verarbeitende Bild aufgehellt. Scaling ist hier nicht zu verwechseln mit dem geometrischen Vergrößern oder Verkleinern von Bildern, was man teilweise ebenfalls als Scaling (Skalieren) bezeichnet.³

Im Falle des Vordergrundbildes wird die Multiplikation vorbereitend zum Thresholding angewendet, da so die Abstände zwischen Hintergrundwert und Vordergrundwert um den Faktor, um den skaliert wird, größer werden. So können einfacher Schwellenwerte gefunden werden, die ein gutes Ergebnis liefern.

Die Erhöhung der Differenz kann folgendermaßen dargestellt werden: Ein Pixel A hat den Wert 4, ein benachbartes Pixel B hat den Wert 6. Der Abstand zwischen den beiden beträgt 2. Einen Schwellenwert von 5 als untere Grenze festzulegen, würde schlechte Ergebnisse liefern, da allein durch das normale Rauschen des Sensors der Schwellenwert von jeweils Pixel A und B höher oder niedriger werden lässt. Angenommen man skaliert das Bild um den Faktor 10, so hat Pixel A den Wert 40 und Pixel B den Wert 60. Der Abstand beträgt nun 20. Hintergrundpixel können so deutlicher identifiziert werden.

In Abbildung 3.10 ist deutlich zu erkennen, dass der Vordergrund (die Hand) besser zu erkennen ist, als noch in Abbildung 3.9 (rechts), während sich der Hintergrund kaum verändert hat.



Abbildung 3.10: Skaliertes Vordergrund.

Quelle: eigene Angabe.

Thresholding

Nachdem durch das Scaling eine gute Ausgangssituation für das Thresholding geschaffen wurde, kann nun dazu übergegangen werden. Die Schwellenwerte können variabel auf dem

³Vgl. Image processing learning resources. Abschnitt Image Arithmetic. [6]

SimpleTouchAreaObserver (siehe Abbildung 3.8) eingestellt werden. Das Thresholding liefert ein boolean Array, das pro Element einen Wert enthält, der angibt ob dieses Pixel als berührt gilt oder nicht. Das resultierende Array hat die Größe des Originalbildes bzw. des zugrundeliegenden Arrays mit den Distanzinformationen.



Abbildung 3.11: Visualisierung des Ergebnisses nach dem Thresholding.

Quelle: eigene Angabe.

Abbildung 3.11 zeigt eine Visualisierung des oben genannten Arrays. Der größere Weiße Bereich ist unser gesuchter berührter Bereich, während die kleineren Flecken unerwünschte Störungen sind.

Morphology

Die kleinen Störungen, die in Abbildung 3.11 zu sehen sind, werden auch als *salt noise* bezeichnet. Um diese Störung aus dem Bild zu filtern, wird das so genannte *Opening* angewendet. Dieser morphologische Operator, ergibt sich wiederum aus den beiden Operatoren *Dilation* und *Erosion*. Die Grundlage, um diese Operatoren anwenden zu können, ist immer ein binäres Bild oder ein Graustufenbild und ein strukturierendes Element. Das Bild ist in diesem Fall das Array, das als Ergebnis des Thresholding entstanden ist. Das strukturierende Element kann sich je nach Anwendungsfall der Operatoren unterscheiden. Von der Bibliothek wird derzeit ein kreisförmiges strukturierendes Element mit einem Radius von 2 Pixeln verwendet. Ein Beispiel ist in Abbildung 3.12 vergrößert dargestellt. Die einzelnen Kästchen stellen jeweils Pixel dar. Schwarze Kästchen sind Hintergrundpixel, weiße Kästchen sind Vordergrundpixel und das hellgraue Pixel ist der Ausgangspunkt, das *Origin*.

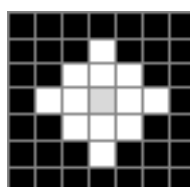


Abbildung 3.12: Vergrößerte Darstellung eines strukturierenden Elements.

Quelle: eigene Angabe.

Dilation (engl. Dilatation/Erweiterung). Bei der Dilation wird das strukturierende Element nun folgendermaßen auf das Originalbild angewendet. Das strukturierende Element wird über jedes Vordergrundpixel des Originalbildes gelegt, sodass das Vordergrundpixel und das Origin-Pixel übereinander liegen. Alle Pixel des Originalbildes, die nun von einem Vordergrundpixel des strukturierenden Elementes verdeckt werden, werden im Ergebnis automatisch zu einem Vordergrundpixel.

Vereinfacht gesprochen kann man es sich so vorstellen: Ein Pinsel mit einer bestimmten Form entspricht dem strukturierenden Element. Mit diesem Pinsel wird der Rand eines Rechtecks nachgezeichnet, so dass die Mitte des Pinsels immer auf dem Rand des Rechtecks entlang geführt wird, ohne den Pinsel zu drehen. Dabei wird das Rechteck erweitert.⁴

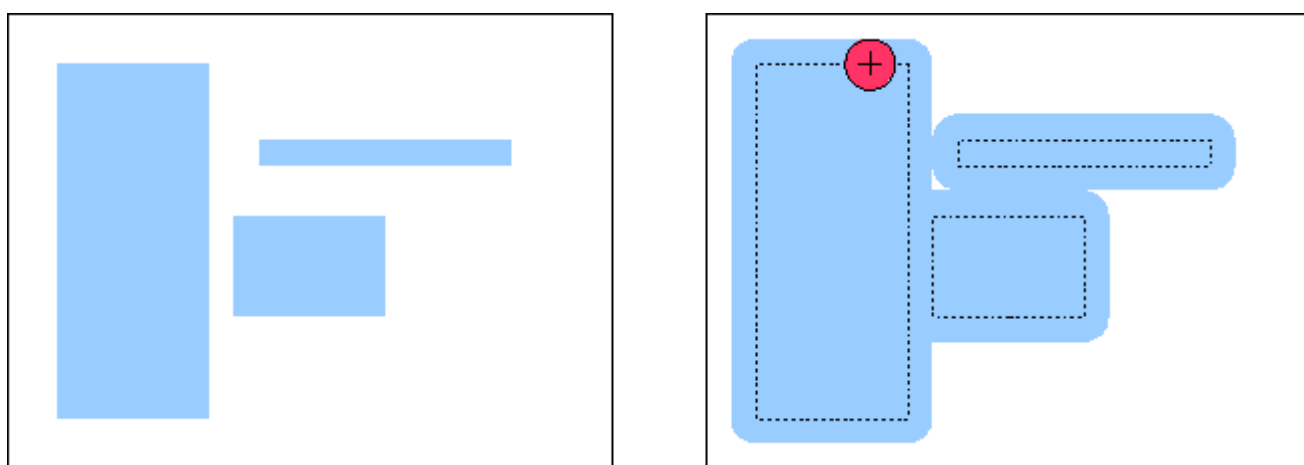


Abbildung 3.13: Dilation von Rechtecken mit einem Kreis als strukturierendes Element.

Quelle: [http://de.wikipedia.org/wiki/Dilatation_\(Bildverarbeitung\)](http://de.wikipedia.org/wiki/Dilatation_(Bildverarbeitung))

Ein Effekt der dadurch entsteht ist, dass Lücken oder Löcher im Rechteck, verkleinert werden. Nahe beieinander liegende Elemente werden so gegebenenfalls miteinander verbunden.

Erosion (engl. Erosion/Abtragung). Erosion funktioniert ähnlich wie Dilation. Es wird ebenfalls das strukturierende Element angewendet. Anders als bei der Dilation werden allerdings alle Vordergrundpixel des Originalbildes, die von Vordergrundpixeln des strukturierenden Elementes überlagert werden automatisch zu Hintergrundpixeln.

Der Vergleich mit dem Pinsel würde dann also so aussehen, dass der Pinsel wie eine Art Radiergummi funktioniert und die Rechtecke verkleinert anstatt vergrößert.⁵

Der Effekt, der hierdurch wiederum entsteht ist der, dass die bearbeiteten Elemente kleiner werden, Lücken und Löcher in den Formen aber größer.

Opening Opening ist ebenfalls ein morphologischer Operator. Dieser macht sich die bereits beschriebenen Operatoren Erosion und Dilation zu nutze, indem es die beiden in dieser Rei-

⁴Vgl. Image processing learning resources. Abschnitt Morphology - Dilation. [6]

⁵Vgl. Image processing learning resources. Abschnitt Morphology - Erosion. [6]

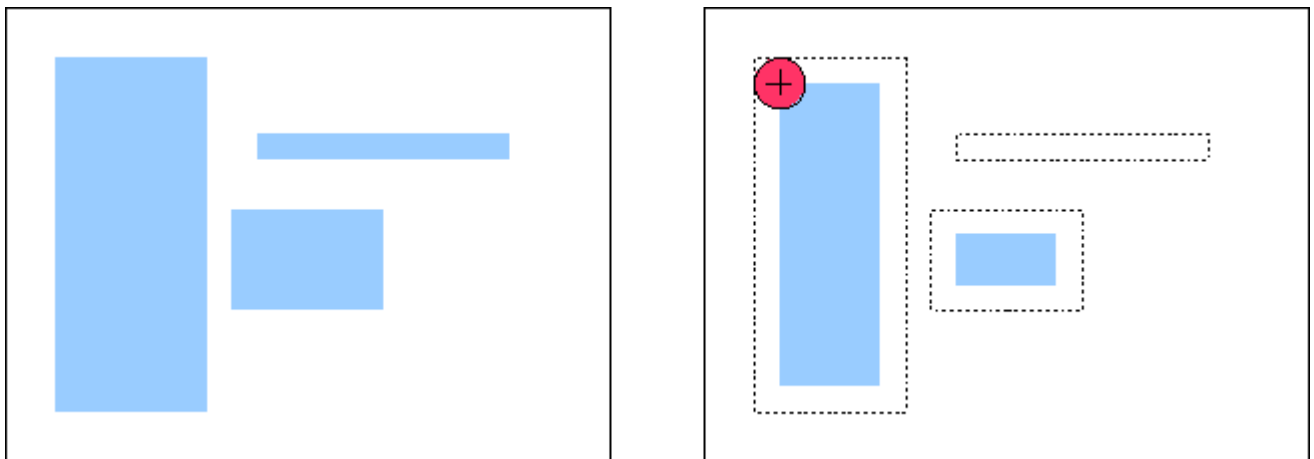


Abbildung 3.14: Erosion von Rechtecken mit einem Kreis als strukturierendes Element.

Quelle: [http://de.wikipedia.org/wiki/Erosion_\(Bildverarbeitung\)](http://de.wikipedia.org/wiki/Erosion_(Bildverarbeitung))

henfolge nacheinander mit dem gleichen strukturierenden Element angewendet. Dabei werden alle Bereiche erhalten, die eine ähnliche Form wie das strukturierende Element haben oder in die das strukturierende Element ganz hinein passt. Es werden nur die Teile einer Form behalten, die von dem strukturierenden Element abgedeckt werden können. Weiterhin werden alle Vordergrundpixel, die kleiner sind als das strukturierende Element bei dem Vorgang entfernt. ⁶So zum Beispiel die fehlerhaften Vordergrundbereiche des nach dem Thresholding entstandenen Bildes.

Um wieder die Vereinfachung mit dem Pinsel zu erhalten: Alle Bereiche eines Bildes, die von dem Pinsel erreicht werden können, ohne dass dieser den Hintergrund berührt, bleiben erhalten. Der Rest wird zu Hintergrund. Die Abbildung 3.15 zeigt ein Dreieck, welches die

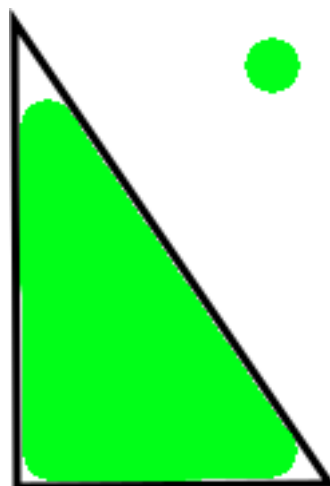


Abbildung 3.15: Opening eines Dreiecks mit einem runden strukturierenden Element.

Quelle: eigene Angabe.

⁶Vgl. Image processing learning resources. Abschnitt Morphology - Opening. [6]

ursprüngliche Form auf weißem Hintergrund darstellen soll. Die grüne Fläche zeigt das Ergebnis des Openings. Es ist zu erkennen, dass der Großteil der Fläche des Dreiecks durch das runde strukturierende Element, welches rechts oben zu sehen ist, abgedeckt werden kann. Die Ecken des Dreiecks können hingegen nicht erreicht werden können ohne über den Rand hinauszukommen. Daher sind diese frei geblieben.

Opening auf das Ergebnis des Thresholding angewendet, liefert das in Abbildung 3.16 zu sehende Bild. Man sieht, dass das der *salt noise* verschwunden ist. Außerdem hat die als berührt erkannte Fläche eine klarere Kante.

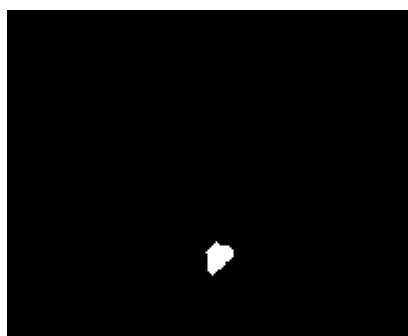


Abbildung 3.16: Ergebnis des Thresholding nachdem morphologischen Opening.

Quelle: eigene Angabe.

Konturen

Nachdem nun durch das Opening ein klarer Bereich entstanden ist, muss noch bestimmt werden, ob der Bereich groß genug ist um ein Berührungspunkt durch einen Finger darzustellen. Genauso muss der Bereich klein genug sein, um nicht zum Beispiel eine ganze Handfläche, die auf den Tisch gelegt wird als einen berührten Punkt zu werten. Außerdem kann nicht der ganze Bereich als ein Berührungspunkt gewertet werden. Daher muss zusätzlich noch das Pixel gefunden werden, dessen Koordinaten als Berührungspunkt für diesen Bereich hergenommen werden. Um diese Aufgaben zu bewerkstelligen, wird versucht Konturen um die jeweiligen Flächen zu legen. Anhand der Länge der Kontur kann entschieden werden, ob diese Flächen zu Berührungspunkten weiterverarbeitet werden sollen oder nicht. Durch die Konturen wird ebenfalls klar, welche einzelnen Pixel des Gesamtbildes zu welcher separaten Fläche gehören. Dadurch kann der ungefähre Mittelpunkt der Fläche ermittelt werden, dessen Koordinaten als Berührungspunkt gelten.

Als erstes müssen dafür alle Pixel gefunden werden, die potentiell als Kontur in Frage kommen. Das sind jeweils Vordergrundpixel, die mindestens ein benachbartes Pixel haben, welches ein Hintergrundpixel ist. Der erste Schritt ist also, die Vordergrundpixel auf Konturpixel zu filtern. Abbildung 3.17 zeigt eine berührte Fläche bzw. deren herausgefilterte Konturpunkte. Es werden also in dem Array, was durch das Opening entstanden ist, die Vordergrundpixel, die keine Kontur sein können entfernt, so dass nur noch potentielle Konturpixel vorhanden sind.

Bei mehreren berührten Flächen oder durch noch nicht herausgefilterte Störungen ist aber zu diesem Zeitpunkt noch nicht eindeutig erkennbar, welche Konturpixel zusammen gehören

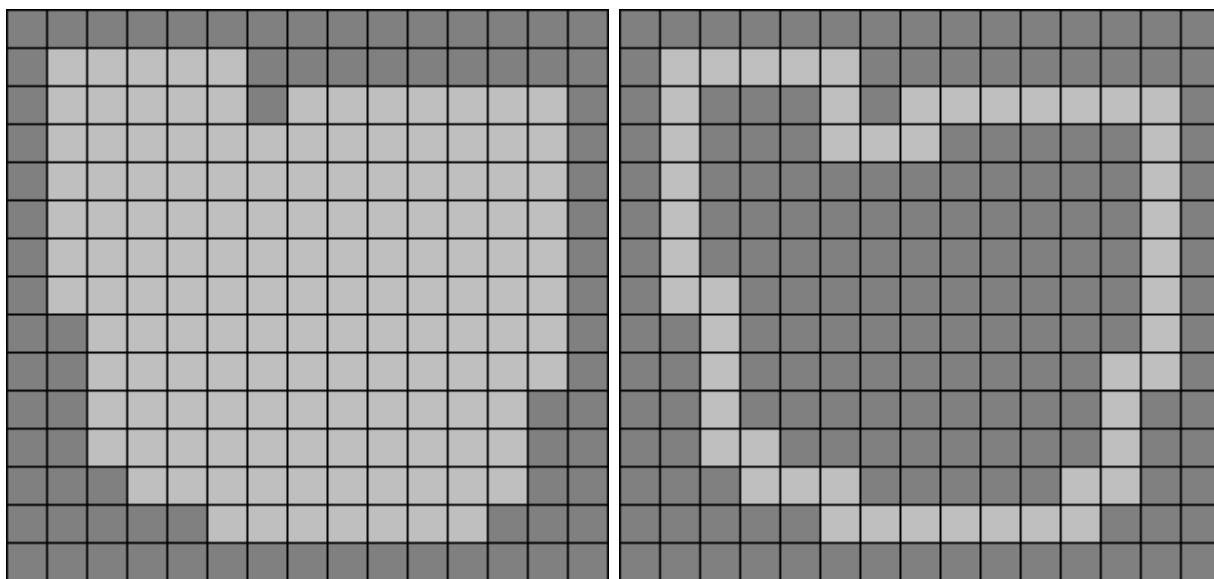


Abbildung 3.17: Schematische Darstellung einer berührten Fläche und ihrer Konturpixel.

Quelle: eigene Angabe.

bzw. eine Fläche eingrenzen, da diese momentan nur als einzelne Pixel im Array zu finden sind. Um herauszufinden, welche Pixel zusammengehören, welche Fläche sie eingrenzen und wie groß diese ist, muss eine sortierte Liste von Pixelkoordinaten pro Kontur erstellt werden. Dazu werden alle potentiellen Pixel durchgegangen und es wird nach benachbarten Pixeln gesucht, die ebenfalls Konturpixel sind. Handelt man sich so von Pixel zu Pixel kommt man irgendwann wieder am ursprünglichen Pixel an. Die Kontur ist dann komplett.

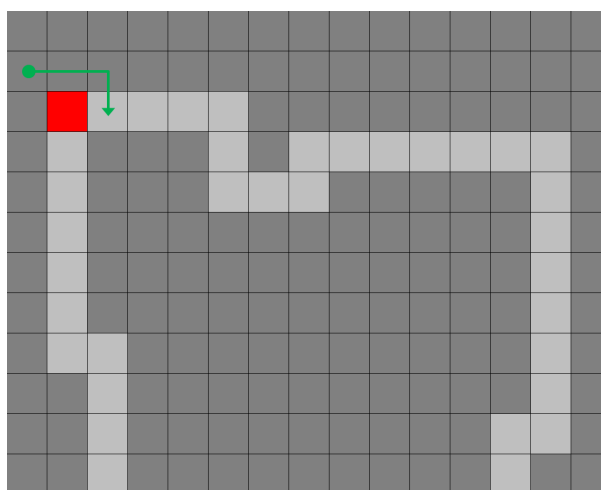


Abbildung 3.18: Erstes Pixel bei der Findung einer zusammenhängenden Kontur.

Quelle: eigene Angabe.

Es wird folgendermaßen vorgegangen: Das oben genannte Array wird so lange durchlaufen bis das erste Pixel gefunden wird, was einer Kontur angehört. In Abbildung 3.18 ist dieses Pixel zur Verdeutlichung rot markiert. Im Umkreis von diesem Pixel wird jetzt nach Nach-

barpixeln gesucht, bei denen es sich ebenfalls um ein Konturpixel handelt. Dabei wird im linken oberen Pixel begonnen und im Uhrzeigersinn weiter gesucht.

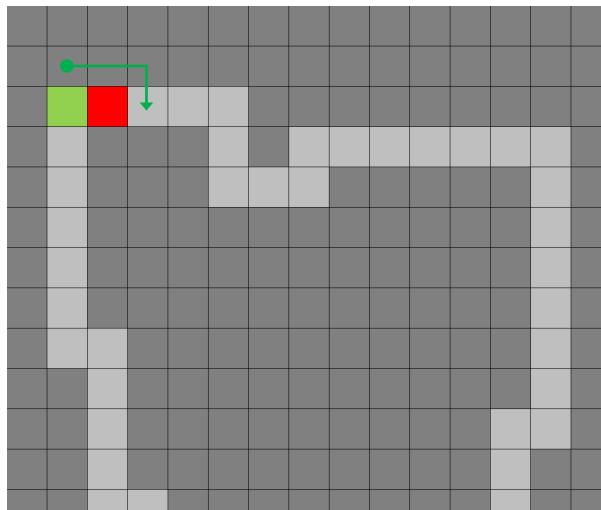


Abbildung 3.19: Zweites Pixel bei der Findung einer zusammenhängenden Kontur.

Quelle: eigene Angabe.

Wird ein weiteres Konturpixel gefunden, wird das erste Pixel zu einer Liste von sortierten Konturpunkten hinzugefügt und in dem Array als Hintergrundpixel markiert, um nicht erneut berücksichtigt zu werden. Der nächste Schritt ist in Abbildung 3.19 abgebildet. Das erste gefundene Pixel ist inzwischen grün markiert. Für das aktuelle Pixel (rot) werden jetzt wieder alle Nachbarpixel wie in Schritt eins durchsucht.

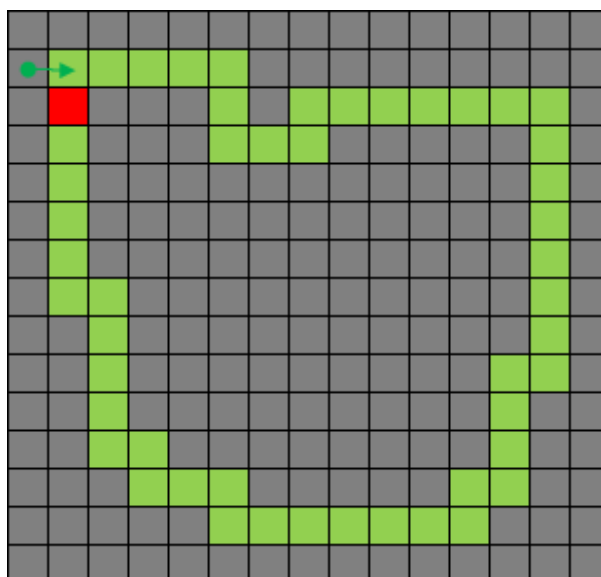


Abbildung 3.20: Vollständig gefundene Kontur einer Fläche.

Quelle: eigene Angabe.

Diese Schritte werden pro Pixel solange wiederholt, bis die Suche wieder am Ausgangspixel

angekommen ist. Ist das der Fall, so wird aus der sortierten Liste der Konturpunkte ein *Contour* Objekt erstellt und im Speicher gehalten. Erst wenn das Array der Konturpunkte ganz durchlaufen wurde, um evtl. weitere Konturen zu erkennen, wird die Verarbeitung der Konturen fortgesetzt.

Als nächstes folgt die Filterung der relevanten Konturen. Alle Konturen, die zu groß oder zu klein sind um eine Fingerspitze einzuschließen, werden verworfen. Für den Rest sollen jetzt die Mittelpunkte gefunden werden. Dafür müssen erst die äußeren Kanten der Kontur gefunden werden. Dies dient dazu, dass ein Rechteck um die Kontur gelegt werden kann, um aus dessen Diagonalen den Mittelpunkt herauszufinden. Abbildung 3.21 zeigt die Pixel der Kontur, die sich jeweils am äußersten Rand befinden in blau. Daneben sind diese äußeren Pixel so verlängert worden, dass sich daraus das einschließende Rechteck ergibt. Daraus kann dann relativ einfach ein Mittelpunkt (gelb) errechnet werden.

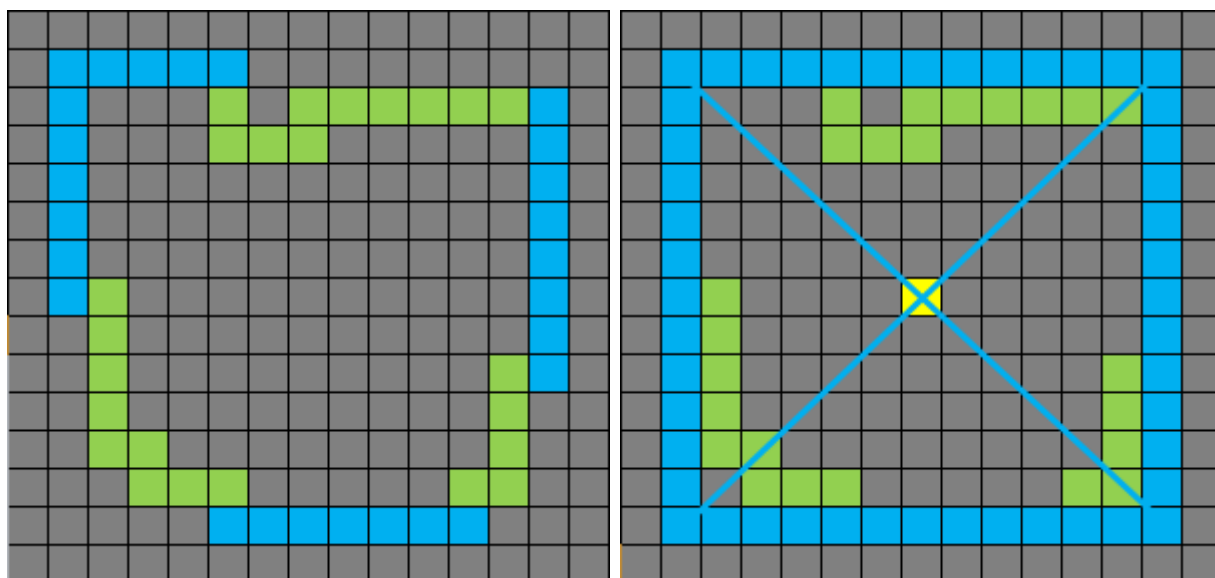


Abbildung 3.21: Äußere Pixel der Kontur. Gedachtes Rechteck und errechneter Mittelpunkt.

Quelle: eigene Angabe.

Ist dieser Schritt für alle Konturen des aktuellen Frames erledigt, können die Koordinaten der Mittelpunkte vorbereitet werden, um vom SimpleTouchAreaObserver weitergegeben zu werden. Dafür werden diese in Objekte vom Typ *TouchPoint* verpackt. Diese Klasse bietet eine X/Y Koordinate und eine Global Unique Identifier (GUID), mit der ein Punkt identifiziert werden kann. Diese GUID wird aber momentan noch nicht verwendet.

Frames

Die Liste der TouchPoints pro Frame wird vom SimpleTouchAreaObserver wiederum in einen eigenen Frame verpackt, der SimpleTouchFrame. Dieser wird vom SimpleTouchAreaObserver per Eventmodell bereitgestellt. So können die extrahierten Informationen pro DepthFrame als TouchFrame abgerufen und weiterverarbeitet werden.

Ein SimpleTouchFrame kann, wie jede Art von Frame in der Bibliothek eine visuelle Darstellung von sich erzeugen. Der passende Ausschnitt zum gerade verarbeiteten DepthFrame

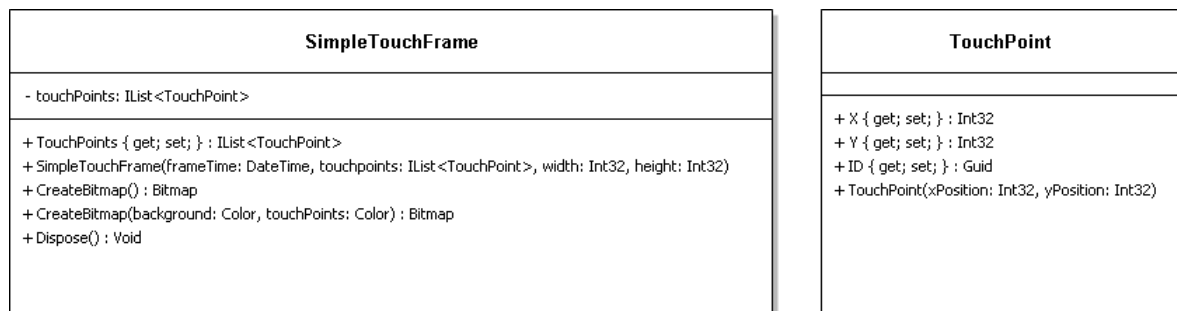


Abbildung 3.22: UML Darstellung eines SimpleTouchFrames und eines TouchPoints.

Quelle: eigene Angabe.

würde wie in Abbildung 3.23 aussehen. Der Hintergrund ist hellblau dargestellt, der erkannte TouchPoint in rot.

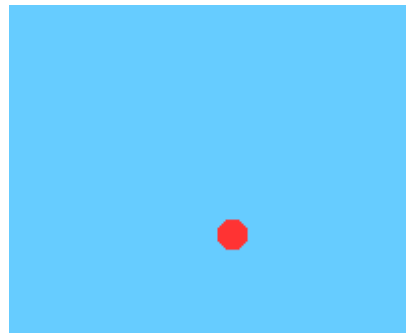


Abbildung 3.23: Ausschnitt einer visuellen Darstellung eines SimpleTouchFrames.

Quelle: eigene Angabe.

3.5 Gestenerkennung

Um Entwicklern, die die Touchdown Bibliothek benutzen wollen, nicht nur Rohdaten zu liefern, sondern gleich ein einsatzbereites Feature, wurde eine einfache Gestenerkennung implementiert. Momentan gilt noch die Einschränkung, dass nur Gesten, die mit einem Finger, sprich einem Berührungspunkt pro Frame, erkannt werden können. Dies hat die Umsetzung erheblich erleichtert. Die Bibliothek bietet neben der Erkennung auch Möglichkeiten, Gesten für spätere Verwendung abzuspeichern und natürlich auch wieder zu laden. Dazu wurde das Extensible Markup Language (XML) Format gewählt.

3.5.1 Definition

Eine Geste besteht aus einer Reihe aufeinanderfolgender SimpleTouchFrames. Dabei werden so viele Frames aufgenommen, bis eine gewisse Anzahl von Frames keine TouchPoints mehr beinhaltet. Dies gilt als Abgrenzung für eine Geste. Die Bibliothek verwendet als interne

Bezeichnung aber nicht *Gesten*, sondern aufgrund des entstehenden Musters bei aufeinanderfolgenden Frames, *TouchPattern*. Auch diese TouchPatterns haben eine Methode, die eine Visualisierung von sich selbst erzeugt. Abbildung 3.24 zeigt ein Pattern, das entsteht, wenn man mit dem Finger von rechts nach links über den Tisch fährt.



Abbildung 3.24: Visualisierung eines TouchPatterns. Finger fährt von rechts nach links über den Tisch.

Quelle: eigene Angabe.

Um darstellen zu können, wo die Geste begonnen hat, und wie ihr Verlauf war, sind Punkte, die zu Beginn der Geste aufgenommen wurden Grün und verlaufen mit weiterem Fortschritt der Geste zu Rot.

Die Patterns können aber nicht nur visuell voneinander unterschieden werden, sondern haben zusätzlich auch einen frei wählbaren Namen.

3.5.2 Wiedererkennen von Gesten

Eine Vorüberlegung für die Erkennung der Gesten war, dass es nicht möglich sein wird, Gesten exakt wiederzugeben. Damit ist gemeint, dass selbst wenn man das Gefühl hat zwei Mal genau die gleiche Geste auf den Untergrund zu zeichnen man doch immer kleine Unterschiede macht. Man beginnt mit minimaler Abweichung an einer anderen Stelle, man braucht länger oder ist schneller fertig als beim Original und so weiter. Daher müssen bei der Wiedererkennung folgende Punkte beachtet werden:

Gesten sollen erkannt werden ...

- ... unabhängig in welchem Bereich des Gesamtbildes sie ausgeführt wurden.
- ... auch wenn sie länger oder kürzer dauern als das Original.
- ... wenn ihr Größenverhältnis mit dem Original übereinstimmt, auch wenn die tatsächliche Größe abweicht.

Alles in allem bedeutet das, dass die Gesten prinzipiell auf Ähnlichkeit und nicht auf Gleichheit überprüft werden sollen. Die Klasse, die diese Aufgabe übernimmt, nennt sich *TouchPatternRecognizer* (siehe Abbildung 3.25). Dieser bezieht seine Rohdaten, nämlich die SimpleTouchFrames, vom SimpleTouchAreaObserver. Zusätzlich zu den Rohdaten müssen dem *TouchPatternRecognizer* die bekannten Patterns, also die die erkannt werden sollen, registriert werden. Das heißt, sie müssen dem Recognizer bekannt gegeben werden.

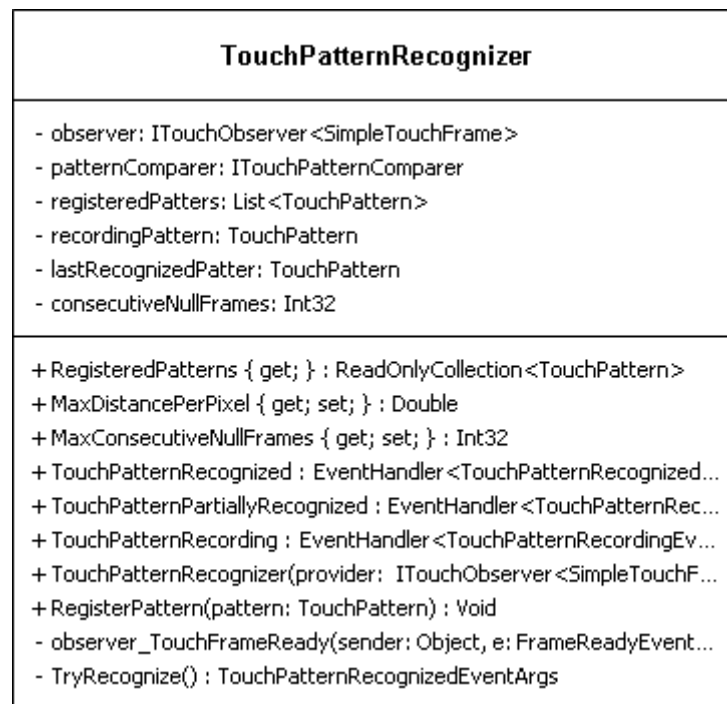


Abbildung 3.25: UML Darstellung des TouchPatternRecognizers.

Quelle: eigene Angabe.

Der Recognizer beinhaltet die registrierten Patterns und einen sogenannten Comparer. Dieser Comparer wird für das eigentliche Vergleichen der Patterns verwendet. Er implementiert ein spezielles Interface. So ist es möglich eine eigene Logik zum Vergleichen von Patterns zu verwenden, indem man dieses Interface implementiert. Gleichzeitig ist das Vergleichen auf mehrere Ebenen verteilt. Alle können jeweils durch Implementieren der entsprechenden Interfaces beeinflusst werden.

ITouchPatternComparer Nimmt zwei TouchPatterns als Parameter und implementiert spezifische Logik, um die Patterns zu vergleichen. Die Standardimplementierung der Touchdown Bibliothek ist der SimpleTouchPatternComparer, welcher jeden Frame des ersten Patterns mit jedem Frame des zweiten Patterns vergleicht. Details sind in Abschnitt 3.5.2 auf Seite 34 zu finden.

ITouchFrameComparer Vergleicht zwei SimpleTouchFrames. Dabei kann auch hier eine eigene Logik implementiert werden. Die Standardimplementierung der Touchdown Bibliothek ist der SimpleTouchFrameComparer, der nur Frames vergleicht, die genau einen Berührungspunkt haben. Alle anderen Frames erhalten einen Standardwert als Vergleichsergebnis, sodass der Vergleich auf keinen Fall als Übereinstimmung gesehen werden kann.

IDistanceProvider Vergleicht zwei TouchPoints. Dabei wird die Entfernung zwischen zwei TouchPoints ermittelt. Die Standardimplementierung der Touchdown Bibliothek ist der EuclidianDistanceProvider, welcher die Entfernung zweier Punkte in der Angabe der kürzesten Strecke macht. Dafür wird der Satz des Pythagoras verwendet.

Normalisierung

Wenn Patterns aufgenommen werden, sind die Koordinaten der TouchPoints, die das Pattern formen immer relativ zur aufgenommenen Fläche. Um Patterns aber unabhängig von ihrer Position auf der Fläche vergleichen zu können, müssen diese von der relativen Darstellung in eine absolute umgewandelt werden. Diese Umwandlung wurde *Normalisierung* genannt. Um ein Pattern zu normalisieren, wird jeweils der kleinste X Positionswert und der kleinste Y Positionswert aller TouchPoints ermittelt. Danach wird dieser Wert von allen TouchPoints abgezogen. Wenn man sich das Pattern in einem Koordinatensystem vorstellt, hat die Normalisierung den Effekt, das gesamte Pattern so weit wie möglich zum Nullpunkt zu verschieben, ohne die X- oder Y-Achse zu unterschreiten. Zusätzlich werden in dem Vorgang die Größen aller TouchFrames im Pattern angepasst, sodass sie gerade so groß sind, dass sie das gesamte Pattern beinhalten können. Die Größe und die Seitenverhältnisse des Patterns bleiben dabei unverändert.

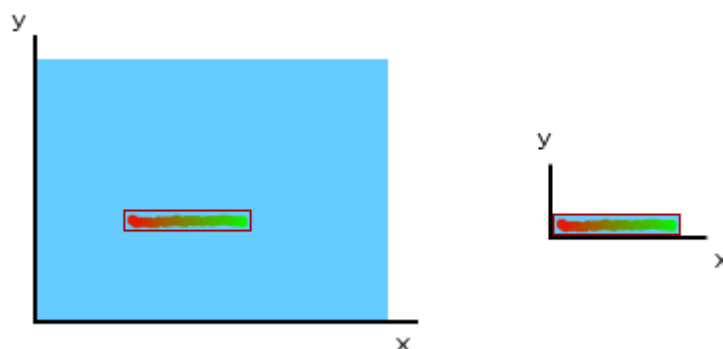


Abbildung 3.26: Normalisierung eines einfachen TouchPatterns.

Quelle: eigene Angabe.

Skalierung

Ein weiterer Schritt der nötig ist, um Patterns miteinander vergleichen zu können, ist die Skalierung auf die gleiche Größe. Anders als im Abschnitt 3.4.3 ist hier jedoch tatsächlich das Angleichen der Größe des Patterns gemeint. Damit wird die Möglichkeit geschaffen, ein Pattern, das in größerem Maßstab wie das Original gezeichnet wurde, ebenfalls zu erkennen. Umgekehrt kann das Pattern auch kleiner als das Original gezeichnet werden.

Das Skalieren nimmt immer eines der beiden Patterns und gleicht die Größe auf das jeweils andere Pattern an. Dabei werden die Koordinaten der TouchPoints des zu skalierenden Patterns neu errechnet. Der untenstehende Pseudocode zeigt, wie die Skalierung grundsätzlich stattfindet. Hierbei ist anzumerken, dass der Pseudocode nicht die tatsächliche Implementierung widerspiegelt, sondern nur vereinfacht darstellt.


```

1  TouchPattern original;
2  TouchPattern scalePattern;
3
4  scaleFactorX = original.MaxXPosition / scalePattern.MaxXPosition;
5  scaleFactorY = original.MaxYPosition / scalePattern.MaxYPostition;
6
7  foreach(var frame in scalePattern.TouchFrames) {
8      foreach(var point in frame.TouchPoints){
9          point.X *= scaleFactorX;
10         point.Y *= scaleFactorY;
11     }
12 }

```

Die Abbildung 3.27 zeigt links das dem TouchPatternRecognizer registrierte, originale Pattern. Rechts daneben, ein Pattern das eingegeben wurde und erkannt werden soll. Man sieht deutlich, dass das eingegebene Pattern größer ist und bei einem Vergleich der einzelnen Punkte eine Erkennung nicht gewährleistet wäre.

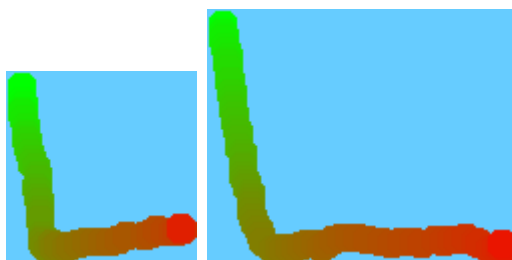


Abbildung 3.27: Original Pattern und unskaliertes Pattern, das verglichen werden soll.

Quelle: eigene Angabe.

Nachdem das eingegebene Pattern allerdings auf die Größe des original Patterns skaliert wurde - wie in Abbildung 3.28 zu sehen - sind sich die beiden Patterns schon sehr ähnlich, sodass die folgende Erkennung eine höhere Erfolgsquote verspricht.

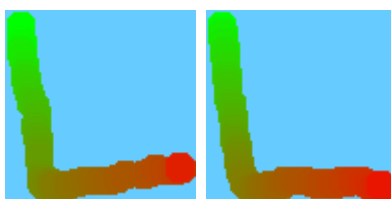


Abbildung 3.28: Original Pattern und zu vergleichendes Pattern nach Skalierung auf die Größe des Originals.

Quelle: eigene Angabe.

Vergleichen von Pattern

Auch wenn die Patterns jetzt geometrisch gesehen gleich groß sind, so können sie dennoch aus unterschiedlich vielen Frames bestehen. Eine Möglichkeit zwei Patterns mit unterschiedlicher Länge zu vergleichen bietet das *Dynamic Time Warping*. Diese Methode wird auch für Vergleiche von Audiodateien oder zur Handschrifterkennung eingesetzt.

Um das Verfahren einfacher beschreiben zu können, werden in folgendem Beispiel Listen bzw. Arrays von Integerwerten anstatt Frames verglichen. Ein Array soll analog zu einem registrierten Pattern dienen und zwei weitere jeweils mit diesem verglichen werden. Dabei soll herausgefunden werden, welches der beiden dem Original ähnlicher ist.

Die Werte sind wie folgt:

Original 2, 4, 6, 7

Erstes Vergleichsarray 5, 3, 1

Zweites Vergleichsarray 1, 3, 5, 8, 8

Die beiden Arrays werden nacheinander mit dem Original verglichen. Dabei wird eine Matrix angelegt, die die Vergleiche der jeweiligen Ziffern beinhalten. Wenn dabei i die Stelle der Ziffer im Original Array und j die Stelle der Ziffer des zu vergleichenden Arrays ist, dann ergibt sich für die einzelnen Matrixelemente folgende Formel:

```
Dist(i,j) = IDistanceProvider.Compare(OriginalArray[i], VergleichsArray[j])
D(i,j) = Dist(i,j)+MIN(D(i,j-1), D(i-1,j), D(i-1,j-1))
```

Das bedeutet, dass jedes Matrixelement nicht nur die berechnete Distanz der Einzelwerte enthält, sondern zusätzlich den kleinsten Wert, aus drei der angrenzenden Nachbarn. Dabei werden nur die drei Nachbarn berücksichtigt, welche sich links, unten oder links-unten vom aktuellen Arrayplatz befinden. Daraus ergeben sich die in Abbildung 3.29 aufgeführten Werte.⁷

Ebenfalls in der Abbildung zu erkennen ist, dass das zweite Array dem Originalen ähnlicher ist. Der Abstandswert beträgt hier nur 5, während beim ersten Array 13 anfallen. Im Bild grün eingezeichnet ist jeweils der Weg mit der kleinsten Abweichung pro Arrayelement. Analog zu dem Beispiel mit den Integerwerten funktioniert das ganze Verfahren auch mit den SimpleTouchFrames. Zusätzlich zur Überprüfung welches Pattern dem gerade eingegebenen am Ähnlichsten ist, wird nur eine Prüfung auf den entgeltigen Wert mit eingebaut. Schließlich soll nicht die Geste mit der größten Ähnlichkeit angezeigt werden, wenn sie nicht tatsächlich auch etwas mit der eingegebenen Geste gemeinsam hat. Daher wird der Wert, der am Ende des kürzesten Pfades steht, durch die Anzahl der verglichenen Werte in diesem Pfad geteilt. Es wird aber nicht mitgezählt wie viele das genau sind, daher wird die ungefähre Anzahl der Pixel so ermittelt:

$$Pixelanzahl = \sqrt{OriginalArrayElemente^2 + VergleichsArrayElemente^2}$$

Die durchschnittliche Entfernung pro Pixel zum Original berechnet sich dann so:

$$Durchschnitt = EndwertDTW / Pixelanzahl$$

Auf den Durchschnittswert lassen sich jetzt wieder Schwellenwerte festlegen, ab wann eine Geste als erkannt gilt.

⁷Vgl. StackOverflow.com. Dynamic Time Warping. [7]

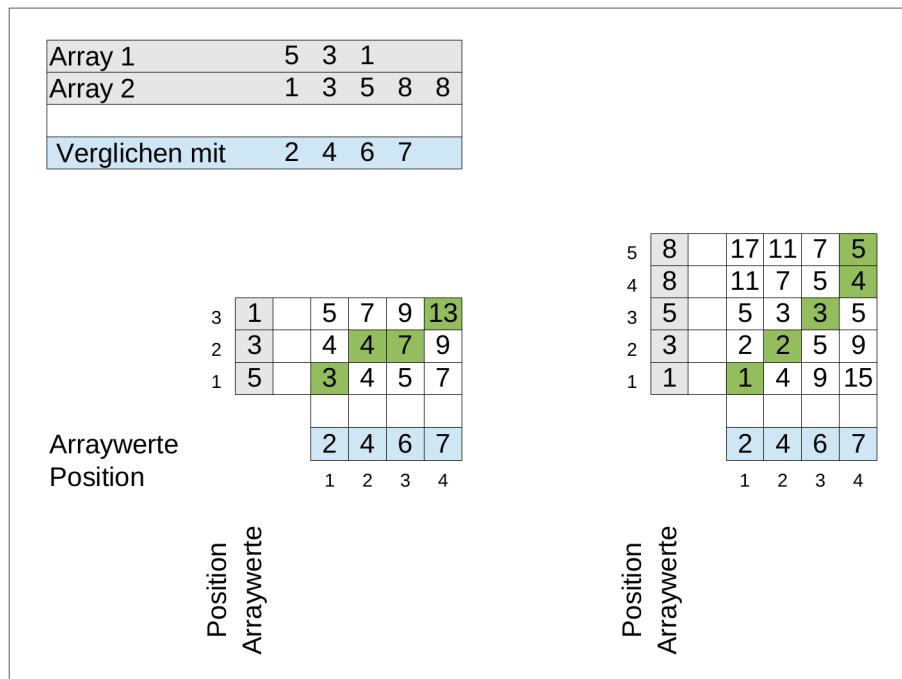


Abbildung 3.29: Dynamic Time Warping - Angewendet auf Integer Arrays.

Quelle: Vgl. <http://stackoverflow.com/a/6369947/254797>

3.5.3 Speichern und Laden

Um nicht bei jedem Programmstart selbst Gesten bzw. Patterns aufnehmen zu müssen, wurde die Möglichkeit geschaffen diese abzuspeichern und wieder zu laden. Dafür wurde der im .NET Framework integrierte *DataContractSerializer* verwendet. Mit dessen Hilfe werden die Pattern-, Frame- und TouchPointinformationen im XML Format gespeichert bzw. daraus wieder gelesen. Dafür muss auf den zu serialisierenden Klassen und Properties lediglich jeweils ein Attribut angebracht werden. Anhand dieser Attribute weiß der *DataContractSerializer* welche Werte er in die XML Zeichenfolge übernehmen muss bzw. auf welches Property der Wert aus der XML Zeichenfolge gehört. Die Serialisierung kann entweder in Streams erfolgen oder direkt als String Zeichenfolgen. Ein Beispielcode für Zeichenfolgen sieht wie folgt aus:

```
1 String serializedPattern = pattern.Save();
2 TouchPattern resultPattern = TouchPattern.Load(serializedPattern);
```

```
/// <summary>
/// Represents a sequence of touchframes.
/// </summary>
[DataContract]
public class TouchPattern : IDisposable {
```

Abbildung 3.30: Beispiel für ein DataContract Attribut auf Klassenebene.

Quelle: eigene Angabe.

3.6 Probleme

Natürlich gab es bei der Umsetzung auch Probleme, die in diesem Abschnitt kurz erläutert werden.

3.6.1 Memory Leaks

Ein Problem, das schon sehr frühzeitig aufgefallen ist, war, dass beim Erstellen der Wrapper Klassen für den Sensor Memory Leaks entstanden sind. Das sind Umstände, die das Freigeben von Speicher verhindern und so dazu führen, dass dieser soweit vollläuft, dass sich das Programm mit einer Fehlermeldung beendet. Als Grundlage hierfür muss man wissen, dass in .NET der sogenannte GarbageCollector dafür zuständig ist, Objekte im Speicher die nicht mehr gebraucht werden, freizugeben. Die genaue Funktionsweise des GarbageCollectors ist sehr komplex und übersteigt den Rahmen dieser Problembeschreibung. Was aber zu sagen ist, ist dass die Freigabe des Speichers zyklisch geschieht und nur, wenn auf die betroffenen Objekte keine Referenzen mehr verweisen. Genau das war im Falle der vom Sensor-Wrapper erstellten DepthFrames noch der Fall. D.h. DepthFrames, die schon lange verarbeitet und daher nicht mehr gebraucht wurden, waren noch im Code referenziert und wurden nicht vom GarbageCollector freigegeben. Die Menge der Daten pro DepthFrame beträgt:

$$\frac{640 \times 480 \text{Pixel} \times 32 \text{Bit}}{8 \text{Bit}} = 1228800 \text{Byte} \approx 1,17 \text{MiB}$$

Bei 30 FPS sind das ca. 35MiB die pro Sekunde. Wenn diese nicht sauber freigegeben werden, ist der Speicher schnell voll.

Die Lösung bestand darin, nicht mehr benötigte Referenzen der Arrays auf *null* zu setzen bzw. die Dispose Methode von Objekten zu implementieren und zu nutzen. Dies führt dazu, dass der GarbageCollector den Speicher ordentlich freigegeben hat und der Speicherverbrauch des Programms auf eine reguläre Menge heruntergesetzt hat.

3.6.2 TouchPoint Erkennung

Eines der größten Probleme war die Qualität der Erkennung der TouchPoints in Abschnitt 3.4. Lange Zeit während der Entwicklungsphase wurde gar nichts erkannt. Erschwerend kam hinzu, dass das Debugging nur bedingt geholfen hat. Pro DepthFrame sind es 640x480 Pixel, die sich, selbst wenn keine Bewegung vor der Kamera stattfindet, durch das Rauschen des Sensors ständig minimal verändern. Diese große Datenmenge machte es fast unmöglich einen guten Ansatzpunkt zu finden, um das Debugging sinnvoll einzusetzen. Letzten Endes war dann ein kleiner Vorzeichenfehler der Grund, warum die Erkennung nicht funktioniert hat. Allerdings war die Qualität der erkannten Punkte immer noch nicht zufriedenstellend.

Nach viel Recherche und Ausprobieren verschiedener, ebenfalls in Abschnitt 3.4 genannter, Methoden zur Verbesserung der Erkennung hat dann die Kombination der eingesetzten Verbesserungen und ständige Anpassung der Schwellenwerte zu einer annehmbaren Qualität der Erkennung geführt. Dabei war allein das Finden der Methoden zur Verbesserung schon ein großer Aufwand, da schlicht keine Vorkenntnisse in diesem Bereich vorhanden waren.

3.6.3 Performance

Als die Erkennung dann funktioniert hat, war die Performance allerdings nicht zufriedenstellend. TouchPoints konnten zwar visualisiert werden, aber mit großer Verzögerung und sehr ruckelig. Zu dem Zeitpunkt waren noch keine Maßnahmen zur parallelen Verarbeitung einzelner Schritte getroffen worden. Der ganze Ablauf wurde erheblich beschleunigt, als die *Tasks* Library von Microsoft eingebunden und benutzt wurde. Diese bietet Funktionen, die dem Entwickler die Arbeit rund um parallele Aufgaben groÙteils abnimmt und dabei sehr einfach zu benutzen ist. Sehr effektiv war an einigen Stellen wie z.B. dem Erstellen der DepthFrames aus den Rohdaten, dem Extrahieren der TouchPoints und dem Vergleichen der Patterns parallele For-Schleifen zu verwenden. Diese funktionieren im Prinzip wie reguläre Schleifen, nur werden einzelne Schleifendurchgänge in eigenen Threads ausgeführt. Dabei ist zu beachten, dass die Tasks Library selbst entscheidet ob und wann eigene Threads erstellt werden. Es wird nicht prinzipiell für jeden Schleifendurchlauf ein eigener Thread verwendet.

^{8 9} Eine Beispielverwendung der parallelen ForEach-Schleife anhand des Vergleiches eines eingegebenen TouchPatterns mit allen registrierten TouchPattern.

```
1 Parallel.ForEach(this.registeredPatters, pattern => {  
2     patternComparer.Compare(pattern, recordingPattern);  
3 });
```

⁸Vgl. Jon Skeet. Parallel Tasks [8]

⁹Vgl. MSDN Parallel.For loop [9]

3.7 Demo Applikation

Um die Fähigkeiten der Touchdown Bibliothek nicht nur zu beschreiben, sondern auch demonstrieren zu können, wurde eine kleine Windows-Forms Applikation entwickelt, welche die Touchdown Bibliothek nutzt. Die Demo Applikation ist so aufgebaut, dass zu Beginn die notwendigen Konfigurationsparameter durch einen Wizzard abgefragt werden. Anschließend wird die Haupt-Umgebung geladen und eingerichtet. Die einzelnen Schritte zur Konfiguration werden im Folgenden kurz erläutert.

3.7.1 Wizzard

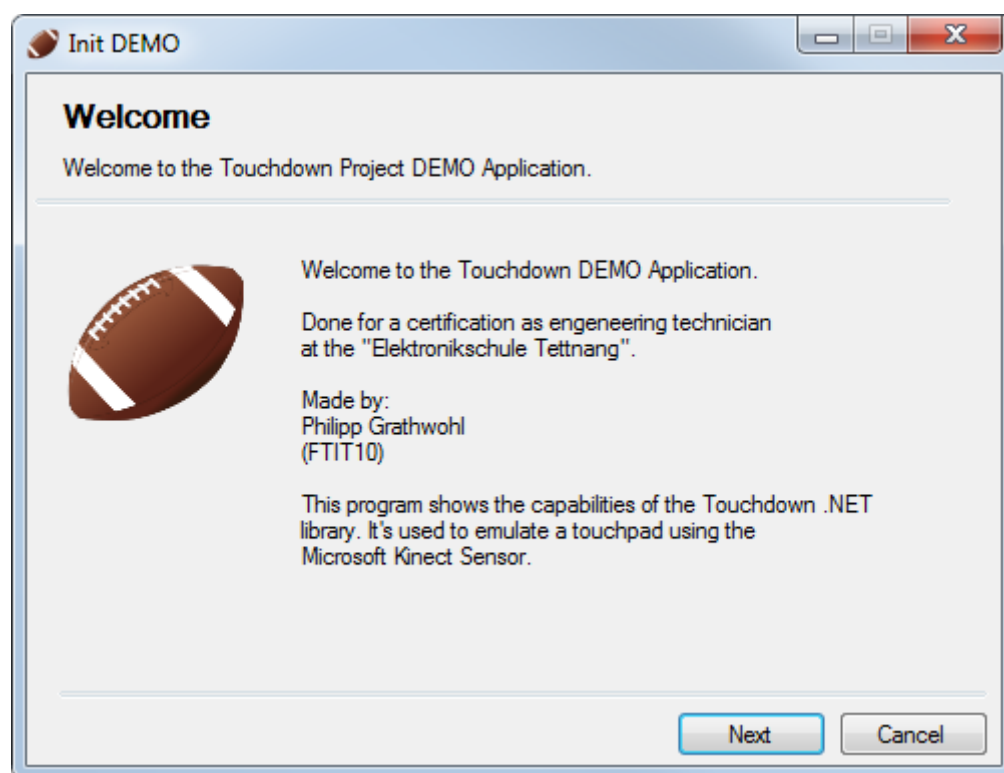


Abbildung 3.31: DEMO Programm - Wizzard - Welcome Page.

Quelle: eigene Angabe.

Die Abbildung 3.31 stellt einen Begrüßungsbildschirm dar, der eine kurze Einleitung und Informationen über das Projekt bzw. die Anwendung enthält. Zu beachten ist, dass dieser und alle weiteren Formulare der Demo Applikation auf Englisch gehalten sind, um sie gegebenenfalls auch außerhalb der Technikerarbeit zur Vorstellung der Bibliothek zu verwenden.

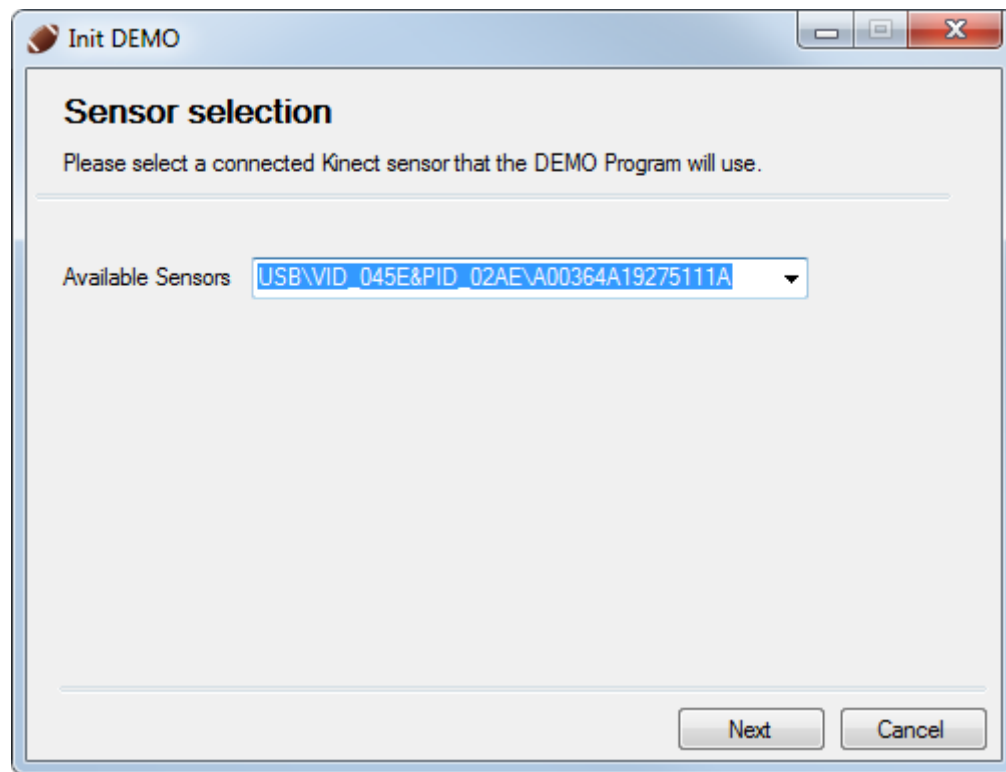


Abbildung 3.32: DEMO Programm - Wizzard - Kinect Sensor Auswahl.

Quelle: eigene Angabe.

Der nächste Schritt des Wizzards ist die Auswahl des zu verwendeten Kinect Sensors. Theoretisch ist es möglich mehrere Kinect Sensoren per USB mit dem Rechner, auf dem das Programm ausgeführt wird, zu verbinden. Die kombinierte Verwendung von mehreren Sensoren ist nicht möglich, daher muss einer der Sensoren ausgewählt werden. Ist kein Sensor angeschlossen, so kann der Wizzard nicht fortgesetzt werden. Es erscheint ein Fehler mit einer entsprechenden Meldung und das Programm kann dann beendet werden. Die Kennzeichnung des Sensors ist eine interne ID, die nicht selbst generiert wurde, sondern direkt vom Sensor selbst abgerufen wird. Diese wird lediglich angezeigt. Nach Auswahl des Sensors kann mit dem nächsten Schritt weitergemacht werden. Abbildung 3.32 zeigt das Wizzardfenster zur Auswahl des Sensors.

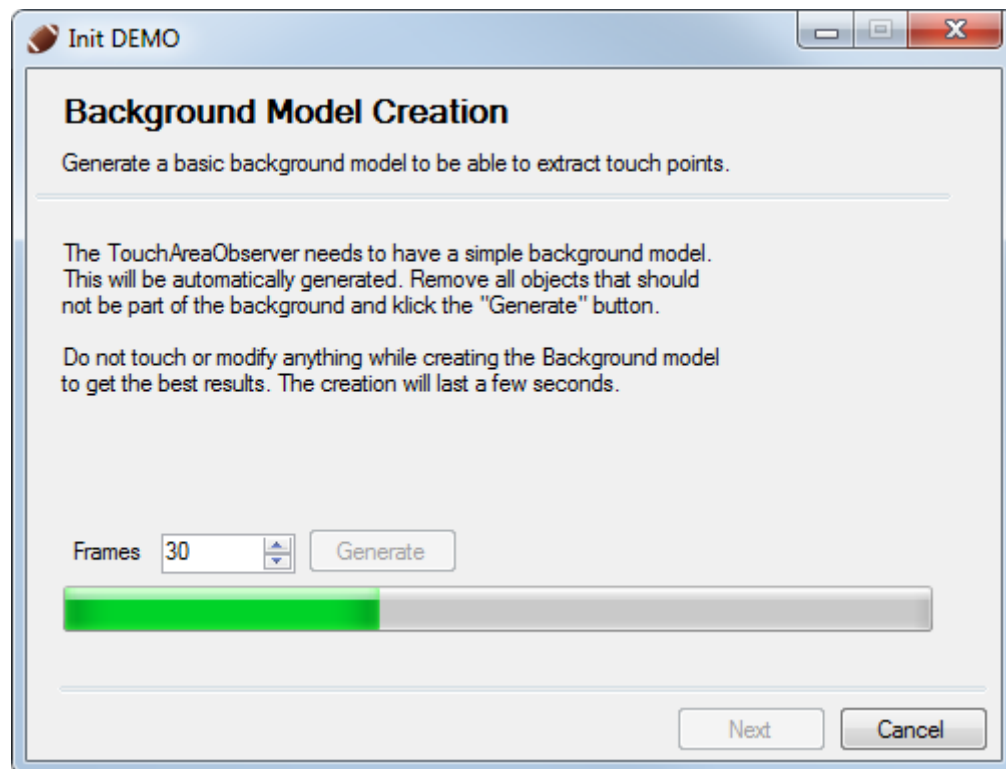


Abbildung 3.33: DEMO Programm - Wizzard - Erstellung des Hintergrundmodells.

Quelle: eigene Angabe.

Wie in Abschnitt 3.4 beschrieben, ist für die Erkennung der TouchPoints ein Hintergrundmodell nötig. Die Erstellung dieses Hintergrundmodells wird ebenfalls vom Wizzard übernommen. Dabei ist, wie in Abbildung 3.33 zu sehen, die Angabe der zu verwendenden Frames nötig. Nach dem Klick auf *Generate* wird dann das Hintergrundmodell erstellt. Dies kann mehrfach, muss aber mindestens ein Mal durchgeführt werden. Während der Erstellung zeigt die Fortschrittsanzeige an, wie weit der Prozess vorangeschritten ist. Danach kann zum nächsten Schritt übergegangen werden.

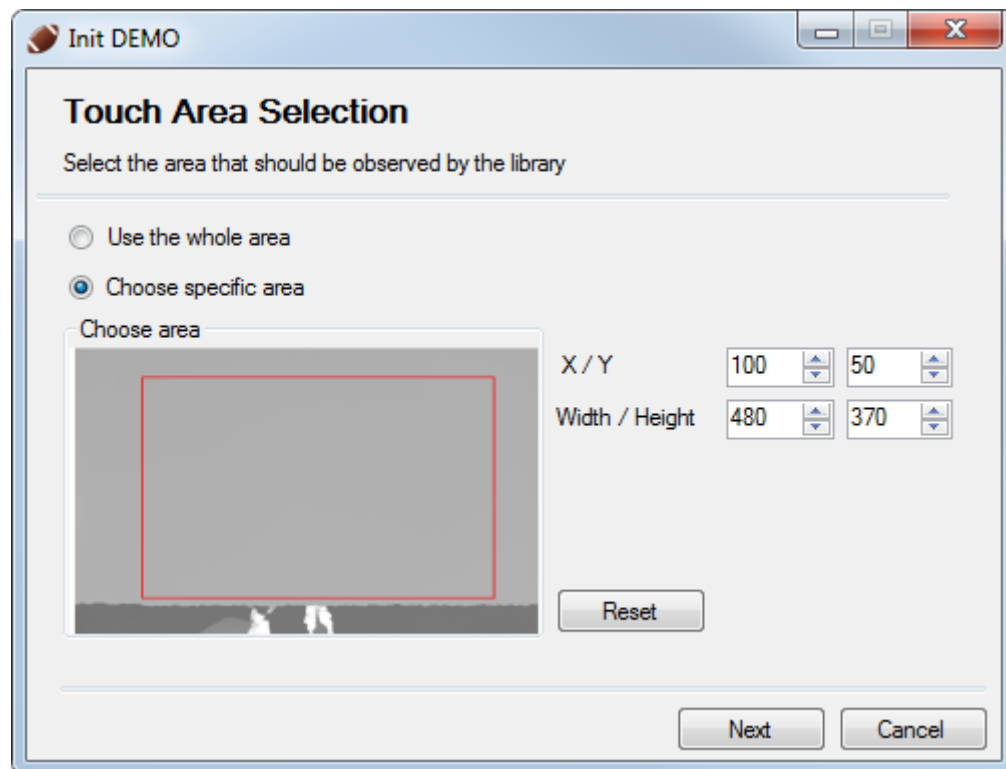


Abbildung 3.34: DEMO Programm - Wizzard - Auswahl des zu erfassenden Bereichs.

Quelle: eigene Angabe.

Als nächstes kann der zu erfassende Bereich ausgewählt werden. Dabei kann entweder der ganze vom Sensor erfassbare Bereich oder aber ein selbst festgelegter Bereich ausgewählt werden. Wird der Bereich selbst festgelegt, so wird in dem visualisierten DepthFrame ein roter Rahmen, der den Bereich eingrenzt, gezeichnet. Dieser lässt sich durch die rechts angeordneten Steuerelemente verschieben, vergrößern oder verkleinern. Dabei sollte darauf geachtet werden, dass möglichst keine Kanten (z.B. Tischkanten oder Ähnliches) im Bereich sind, um Falscherkennungen und Störungen so gering wie möglich zu halten. Abbildung 3.34 zeigt einen selbst ausgewählten Bereich eines DepthFrames.

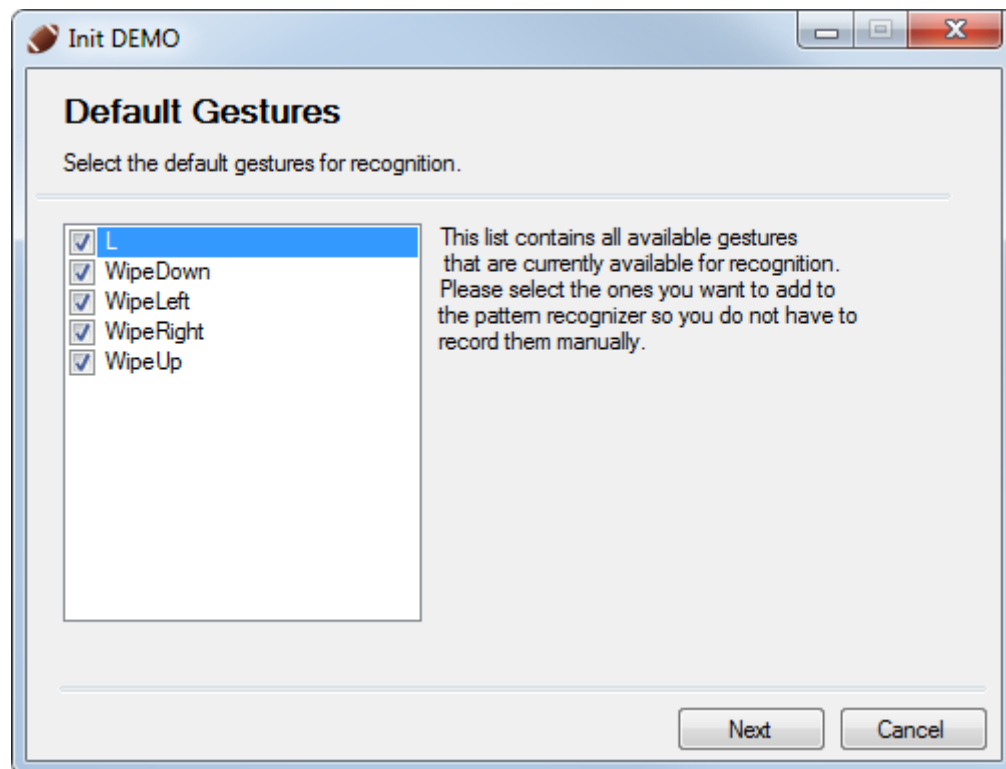


Abbildung 3.35: DEMO Programm - Wizzard - Auswahl der im Standard zu registrierenden Pattern.

Quelle: eigene Angabe.

Um nicht nach jedem Start des Demo Programms selbst alle Gesten registrieren zu müssen, folgt eine Auswahl der automatisch zu registrierenden Gesten. Dabei wird die Funktionalität des Speicherns und Ladens von TouchPatterns angewendet. Alle hier verfügbaren Patterns finden sich im Unterverzeichnis *Gestures* des Startpfades der aktuellen Anwendung wieder. Die hier ausgewählten Patterns werden alle beim später initialisierten TouchPatternRecognizer registriert und stehen ab dann der Erkennung von Gesten zur Verfügung.

3.7.2 Hauptfenster

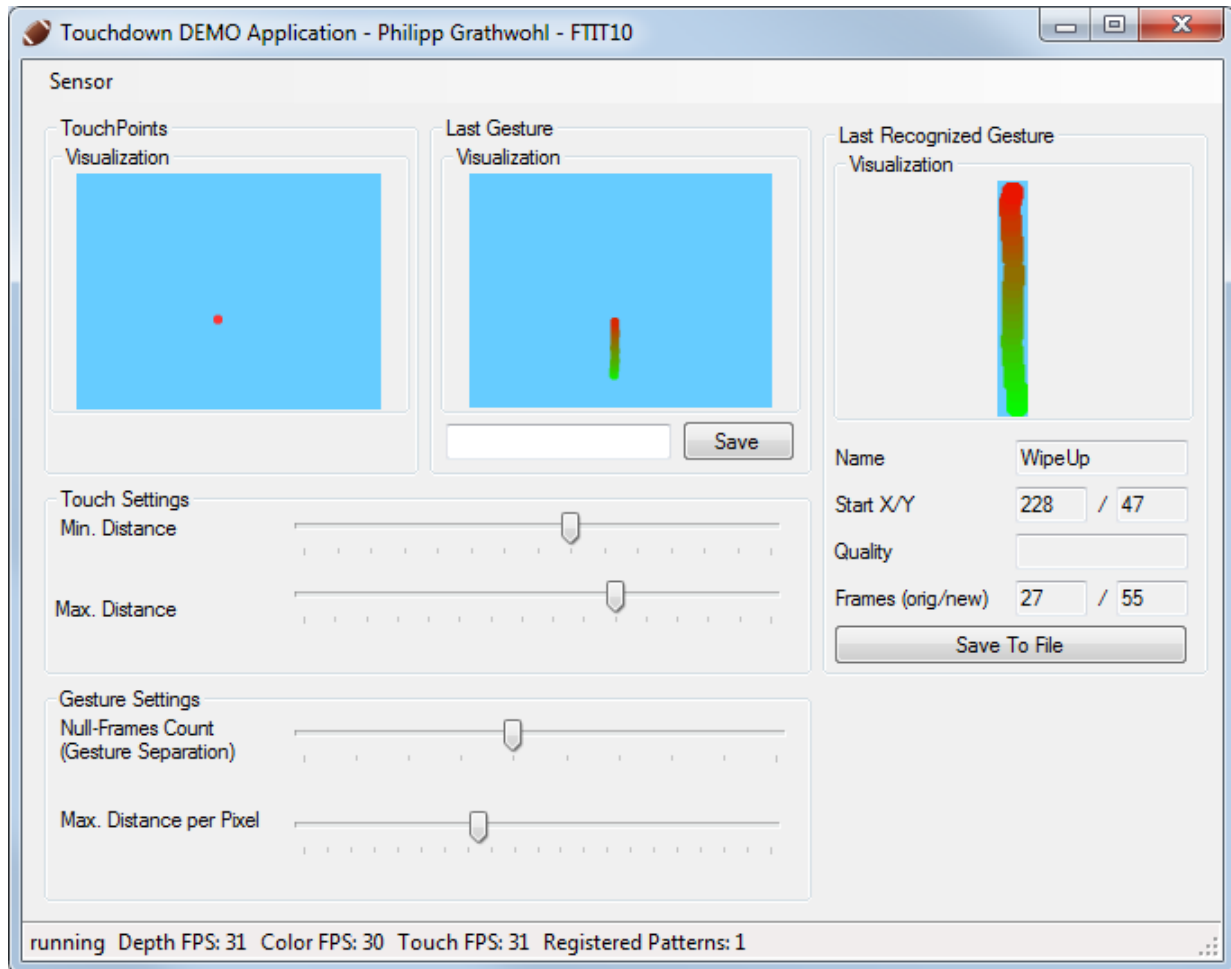


Abbildung 3.36: DEMO Programm - Hauptfenster, das die DEMO Funktionalität bereitstellt.

Quelle: eigene Angabe.

Nachdem der Wizzard abgeschlossen ist, öffnet sich das Hauptfenster wie in Abbildung 3.36 zu sehen. Dabei werden alle benötigten Objekte mit den im Wizzard gesammelten Informationen initialisiert. Dazu zählen zum Beispiel der Sensor Wrapper, der SimpleTouchAreaObserver und der TouchPatternRecognizer. In diesem Zustand ist der Sensor noch nicht gestartet. Dies kann über das Hauptmenü über *Sensor - Sensor starten* geändert werden. Über den gleichen Weg kann dieser auch wieder gestoppt werden. Die einzelnen Elemente des Hauptfensters werden im Folgenden erklärt.

TouchPoints Im linken oberen Bereich ist die Visualisierung der TouchPoints zu finden. Hier wird jeder SimpleTouchFrame, der vom SimpleTouchAreaObserver zurückgeliefert wird visualisiert. Es werden also immer die in diesem Moment erkannten TouchPoints angezeigt. Dabei ist der Hintergrund hellblau und die momentan vorhandenen TouchPoints rot. Sind keine TouchPoints erkannt, wird nur ein hellblauer Hintergrund angezeigt.

Last Gesture Rechts neben der Visualisierung der TouchPoints befindet sich der Bereich in dem die zuletzt eingegebene Geste visualisiert wird. Wenn dort eine Geste erscheint, kann in der untenstehenden Textbox ein Name gewählt werden und die eingegebene Geste beim TouchPatternRecognizer registriert werden, so dass sie ab dem Zeitpunkt auch wiedererkannt werden kann.

Last Recognized Gesture Hier wird dargestellt, wenn eine Geste wiedererkannt wurde. Dabei wird immer die Originalgeste visualisiert. Zusätzlich stehen in den Textboxen darunter Informationen zur wiedererkannten Geste zur Verfügung. So zum Beispiel der Name der Originalgeste, die Koordinaten relativ zum DepthFrame wo die eingegebene Geste begonnen hat, die Anzahl der Frames der Original- und der der eingegebenen Geste im Vergleich und die Qualität der Erkennung, die derzeit immer noch leer gelassen wird. Wenn eine Geste erkannt wurde, kann diese durch den Button *Save To File* in einer Datei gespeichert werden, um sie z.B. beim nächsten Programmstart automatisch wieder registrieren zu lassen.

Touch Settings Neben den Visualisierungen können auch noch einige Einstellungen beeinflusst werden. Der TouchSettings-Bereich beeinflusst dabei die Erkennung der Berührungspunkte aus den DepthFrames wie in Abschnitt 3.4 beschrieben. Die zwei Schieberegler setzen jeweils die Werte für den minimalen und maximalen Abstand der Finger zur Oberfläche fest.

Gesture Settings Hier können Einstellungen, die die Wiedererkennung der Gesten beeinflussen, gesetzt werden. Der erste Schieberegler gibt dabei an wie viele aufeinanderfolgende Frames keine TouchPoints enthalten dürfen, bevor die Geste als unterbrochen gilt. Der zweite Schieberegler gibt die nötige Qualität einer Geste im Vergleich zum Original an, um noch als erkannt zu gelten. Hier wird der maximale durchschnittliche Abstand pro TouchPoint angegeben. (Siehe Abschnitt `refsec:ComparePatterns`)

Statusleiste In der Statusleiste sind verschiedene Informationen zum Programmstatus angegeben. Darunter eine Angabe, ob der Sensor momentan läuft oder nicht (running / stopped), Performanceindikatoren der einzelnen Frames und die Menge der momentan registrierten TouchPatterns. Die Performanceindikatoren geben jeweils die aktuelle Framerate der Depth-, RGB- und TouchFrames an.

3.7.3 Anmerkungen

Das Demoprogramm bietet keine produktive Funktionalität und erhebt keinen Anspruch auf Vollständigkeit. Es soll lediglich zeigen, was die Touchdown Bibliothek leistet. Anstatt nur die erkannte Geste zu visualisieren, könnte aber zum Beispiel auch eine beliebige Aktion im Programmcode ausgeführt werden. Da es sich nur um eine Demo Applikation handelt, wurde diese auch nicht vollständig getestet. Daher ist es möglich, dass noch Fehlfunktionen auftauchen oder sie durch Fehler automatisch beendet wird.

4 Fazit

Die Touchdown Bibliothek ist zu einer Grundlage geworden, die für Entwickler den Einstieg erleichtert, den Microsoft Kinect Sensor als Eingabemethode in Form eines TouchPanels für ihre Applikationen zu nutzen. Neben den Rohdaten, die als TouchPoints bzw. SimpleTouchFrames geliefert werden, kann auch die integrierte Gestenerkennung genutzt werden. Laden und Speichern von Gesten kann ebenfalls verwendet werden. Die Bibliothek unterstützt neben dem Microsoft SDK für die Entwicklung auch das OpenKinect Framework Freenect. Sie ist modular aufgebaut und kann durch die Nutzung einiger Interfaces sehr einfach um eigene Funktionalität erweitert werden. Die Entwickler, die die Touchdown Bibliothek einsetzen, sind völlig frei in der Entscheidung, wie sie mit den Rohdaten umgehen wollen oder was beim Erkennen einer Geste geschehen soll.

Das Projekt hat viel Einarbeitung gebraucht, hat dadurch aber einige Einblicke in Bereiche gegeben, die man als Entwickler von Businessapplikationen eher selten hat. Die eigene Entwicklung der benötigten Algorithmen hat zwar Zeit gekostet, aber Abhängigkeiten wurden größtenteils vermieden und der Lerneffekt war dadurch am größten. Gerade bei der Entwicklung in einem Gebiet, in dem man keine Erfahrung hat, ist es manchmal frustrierend an einem bestimmten Punkt nicht weiterzukommen, allerdings ist es dann um so aufmunternder es dann doch zu schaffen. Das war das, was mich im Nachhinein am meisten begeistert hat.

4.1 Einsatzzwecke

Als potentielle Einsatzzwecke für die Touchdown Bibliothek könnte man sich zum Beispiel folgende Punkte vorstellen:

- Mausersatz oder Erweiterung bzw. Gestensteuerung für die normale Arbeit am PC.
- In Kombination mit einem großen Display eine Touchscreen oder Touchtable Implementierung ähnlich wie Microsofts Touchtable *Surface*.
- In Kombination mit einem Beamer eine Touchwall oder digitale Tafel.
- ...

4.2 Weiteres Vorgehen

Durch die viele Arbeit, die bisher in das Projekt investiert wurde, wäre es schade sie einfach auf einer Backup-CD oder Festplatte veralten zu lassen. Daher ist der Plan, den Quellcode unter einer passenden, noch auszuwählenden OpenSource Lizenz zu veröffentlichen und Entwicklern die Chance zu geben, das Projekt kennen zu lernen und möglicherweise sogar zu erweitern. Die Liste der folgenden Punkte stellen die Ideen für das weitere Vorgehen bereit:

- Gestaltung und Veröffentlichung einer Projekthomepage mit detaillierten Informationen über das Projekt Touchdown.
- Veröffentlichen des Quellcodes unter einer OpenSource Lizenz.
- Hosten in einer für OpenSource Projekten bekannten Projektseite (z.B. GitHub) inklusive BugTracking.
- Erweiterung des Freenect Sensor Wrappers, sodass dieser auch eingesetzt werden kann.
- Weitere Verbesserung der Qualität der Erkennung von TouchPoints z.B. durch miteinbeziehen der RGBFrames.
- Erweiterung der Gestenerkennung auf Multitouch-Gesten.

Abkürzungsverzeichnis

API Application Programming Interface

DVCS Distributed Version Controll System

FPS Frames per Second

GUID Global Unique Identifier

IDE Integrated Development Environment

MSDN Microsoft Developer Network

MSDNAA Microsoft Developer Network Academic Alliance (inzwischen DreamSpark)

NAS Network Attached Storage

SDK Software Development Kit

USB Universal Serial Bus

VB.NET Visual Basic .NET

XML Extensible Markup Language

Glossar

Assembly Eine Assembly bezeichnet im .NET Umfeld eine kompilierte Datei (z.B. .exe oder .dll) welche die programmierte Logik beinhaltet.

Debugging Vorgang des Fehlersuchens und -behebens im Quellcode. Oft durch unterschiedliche Werkzeuge unterstützt. Viele Entwicklungsumgebungen bringen einen eigenen Debugger mit, der es ermöglicht den Quellcode Zeile für Zeile zu durchlaufen und währenddessen Variablen auszuwerten.

Frame (engl. Rahmen) Hier: Datencontainer für einen Datenstrom der Kinect. Kann z.B. Daten der RGB Kamera enthalten oder aber Daten für Tiefeninformationen. Ein Datencontainer enthält immer Daten für eine Momentaufnahme. Beispiel Einzelbild eines Videos.

Framework Sammlung von Klassen und Methoden die Basisfunktionalität/Rahmenbedingungen für einen bestimmten Bereich bieten. Kein Programm, sonder nur ein Grundgerüst, das zur weiteren Verwendung gedacht ist.

Pseudocode Nicht lauffähiger Quellcode in komplett oder teilweise erfundener Programmiersprache. Diese Codebeispiele sollen ähnlich wie Struktogramme nur Beispiele verdeutlichen.

Wizzard (engl. Zauberer) Bezeichnung für einen geführten Programmteil. Oftmals werden solche für Konfigurationen und Installationen verwendet. Dazu sind diese aufgrund ihres sehr einfachen Aufbaus gut geeignet.

Wrapper Bezeichnung für Klassen, die eine bestimmte Funktionalität einer anderen Klasse einschließen und über ein einheitliches, definiertes Interface in anderer Form oder teilweise wieder veröffentlichen. Wird auch für Methoden verwendet, die Funktionalität einer anderen Methode unter anderem Namen oder mit abgewandelten Parametern bereitstellt. Wrapper werden auch Adapter genannt.

Literaturverzeichnis

- [1] Microsoft. Kinect Color Image Format <http://msdn.microsoft.com/en-us/library/microsoft.kinect.colorimageformat.aspx>, Zugriff 28.05.2013
- [2] OpenKinect Community. Projektseite der OpenKinect Community incl. Wiki. <http://openkinect.org/>, Zugriff 15.12.2012 - Projektende
- [3] OpenClipart.org. Gemeinfreie Clipart-Bibliothek. <http://openclipart.org>, Zugriff 22.05.2013
- [4] Karl Sanford, CodeProject. 2012. Smoothing Kinect Depth Frames in Real-Time. <http://www.codeproject.com/Articles/317974/KinectDepthSmoothing>, Zugriff 05.05.2013
- [5] StackOverflow.com. Bilateral Filtering Algorithm. <http://stackoverflow.com/questions/7512691/bilateral-filter-algorithm>, Zugriff 08.05.2013
- [6] Robert Fisher, Simon Perkins, Ashley Walker, Erik Wolfart. 2004. Image processing learning resources. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/index.htm>, Zugriff 10.12.2012 bis Projektende
- [7] StackOverflow.com. Dynamic Time Warping. <http://stackoverflow.com/a/6369947/254797>, Zugriff 10.03.2013
- [8] StackOverflow.com. Jon Skeet. Parallel Tasks Library. Thread count. <http://stackoverflow.com/a/1114402/254797>, Zugriff 03.06.2013
- [9] Microsoft Developer Network. Parallel.For. <http://msdn.microsoft.com/de-de/library/system.threading.tasks.parallel.for.aspx>, Zugriff 03.06.2013
- [10] Stan Salvador, Philip Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. <http://cs.fit.edu/~pkc/papers/tdm04.pdf>, Zugriff 04.06.2013
- [11] KinectTouch Project Home. <https://github.com/robbeofficial/KinectTouch>, Zugriff 15.12.2012
- [12] F.Trapero Cerezo. Kinect Hand and Finger tracking. <http://frantracerkinectft.codeplex.com/releases>, Zugriff 20.12.2012
- [13] Brandon Cannaday. 2011. Kinect Tutorial - Hacking 101. <http://tech.pro/tutorial/1039/kinect-tutorial-hacking-101>, Zugriff 20.12.2012
- [14] Microsoft. Getting and Displaying Depth Data in C#. <http://msdn.microsoft.com/en-us/library/jj131029.aspx>, Zugriff 13.04.2013

- [15] David Catuhe. 2011. Unleash the power of Kinect for Windows SDK. <http://blogs.msdn.com/b/eternalcoding/archive/2011/06/13/unleash-the-power-of-kinect-for-windows-sdk.aspx>, Zugriff 13.04.2013
- [16] Kinect Calibration. <http://burrus.name/index.php/Research/KinectCalibration>, Zugriff 20.12.2012
- [17] Herrera C., D., Kannala, J., Heikkilä, J. 2012. Joint depth and color camera calibration with distortion correction. <http://www.ee.oulu.fi/~dherrera/kinect/>, Zugriff 10.06.2013
- [18] Nathan Crock. 2011. Kinect Depth vs. Actual Distance. <http://mathnathan.com/2011/02/depthvsdistance/>, Zugriff 18.12.2012

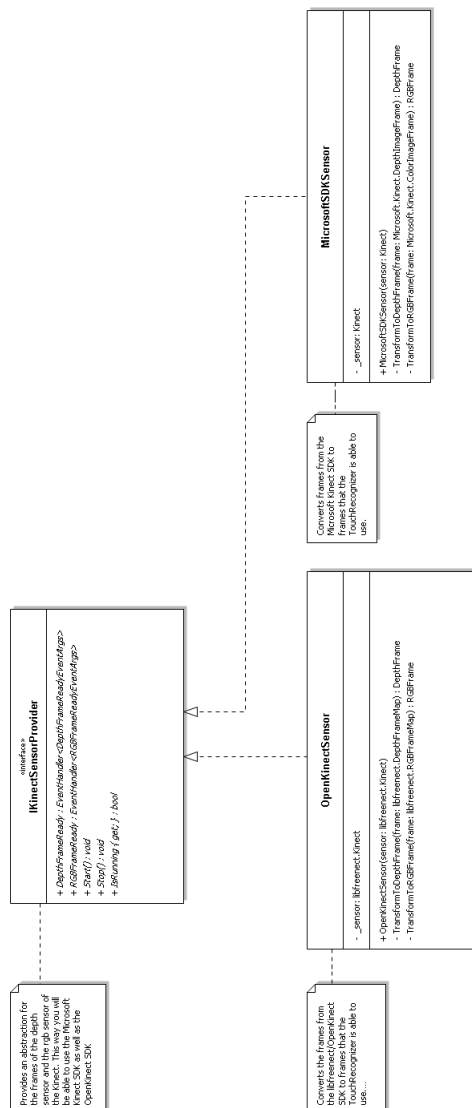
Abbildungsverzeichnis

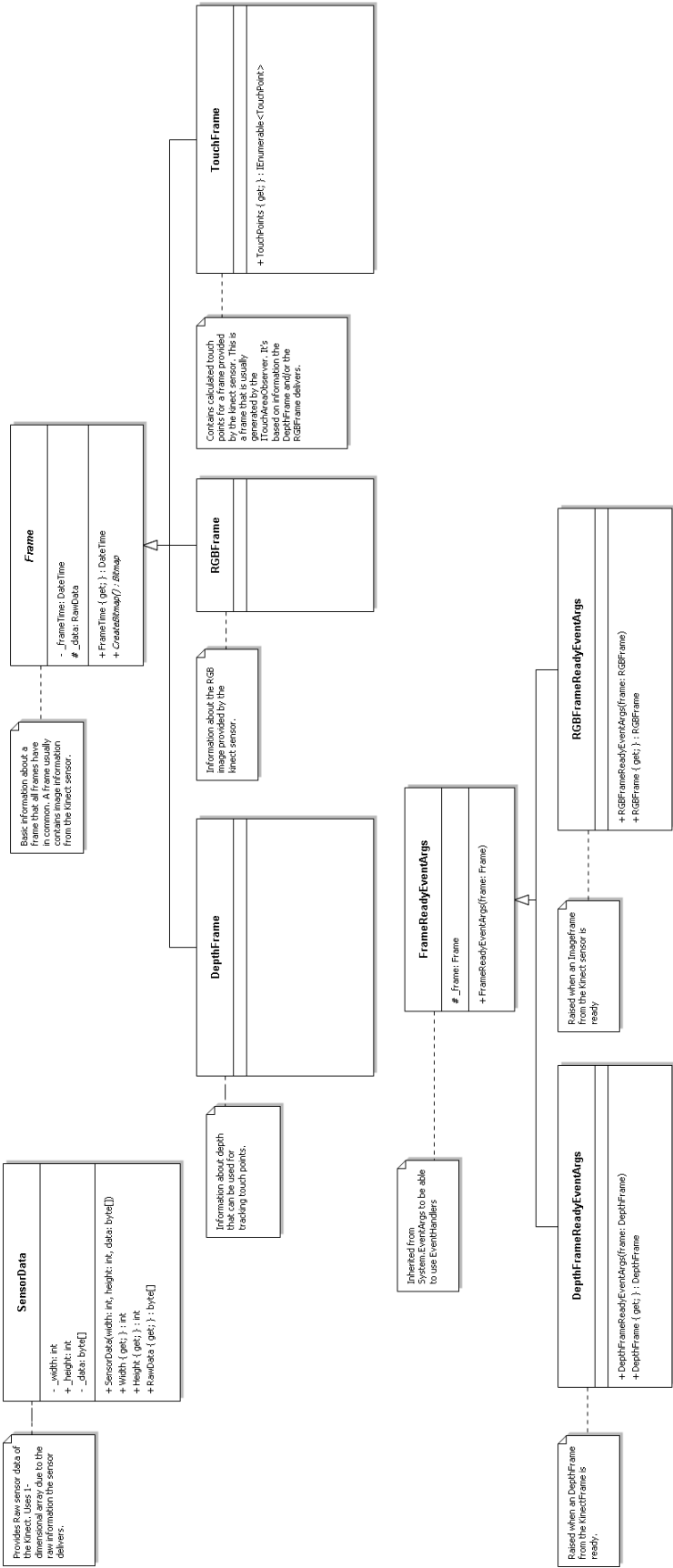
2.1	Erfolgreich ausgeführte Tests für den Bereich TouchPattern.	6
2.2	XML Kommentar mit Verweisen auf andere Klassen.	6
2.3	Ausschnitt der kompilierten API-Dokumentation.	7
2.4	Frontansicht des Kinect Sensors.	8
2.5	Aufteilung der einzelnen Sensoren in der Kinect.	10
3.1	Touchdown Project Logo.	12
3.2	Skizzierter Programmablauf	13
3.3	Schematische Darstellung der Projektstruktur	14
3.4	Projektstruktur der Visual Studio Solution	15
3.5	Interface für Wrapperklassen. Lieferant für RGB- und DepthFrames.	16
3.6	Aufbau des Sensors	17
3.7	Skizzierter Finger. Ober- und Untergrenzen festgelegt.	18
3.8	UML Ausschnit - SimpleTouchAreaObserver.	19
3.9	Erstellung des Vordergrundframes. Links Ausgangsbild, rechts Vordergrund.	21
3.10	Skalierter Vordergrund.	22
3.11	Visualisierung des Ergebnisses nach dem Thresholding.	23
3.12	Vergrößerte Darstellung eines strukturierenden Elements.	23
3.13	Dilation von Rechtecken mit einem Kreis als strukturierendes Element.	24
3.14	Erosion von Rechtecken mit einem Kreis als strukturierendes Element.	25
3.15	Opening eines Dreiecks mit einem runden strukturierenden Element.	25
3.16	Ergebnis des Thresholding nachdem morphologischen Opening.	26
3.17	Schematische Darstellung einer berührten Fläche und ihrer Konturpixel.	27
3.18	Erstes Pixel bei der Findung einer zusammenhängenden Kontur.	27
3.19	Zweites Pixel bei der Findung einer zusammenhängenden Kontur.	28
3.20	Vollständig gefundene Kontur einer Fläche.	28
3.21	Äußere Pixel der Kontur. Gedachtes Rechteck und errechneter Mittelpunkt.	29
3.22	UML Darstellung eines SimpleTouchFrames und eines TouchPoints.	30
3.23	Ausschnitt einer visuellen Darstellung eines SimpleTouchFrames.	30
3.24	Visualisierung eines TouchPatterns. Finger fährt von rechts nach links über den Tisch.	31
3.25	UML Darstellung des TouchPatternRecognizers.	32
3.26	Normalisierung eines einfachen TouchPatterns.	33
3.27	Original Pattern und unskaliertes Pattern, das verglichen werden soll.	34
3.28	Original Pattern und zu vergleichendes Pattern nach Skalierung auf die Größe des Originals.	34
3.29	Dynamic Time Warping - Angewendet auf Integer Arrays.	36
3.30	Beispiel für ein DataContract Attribut auf Klassenebene.	36

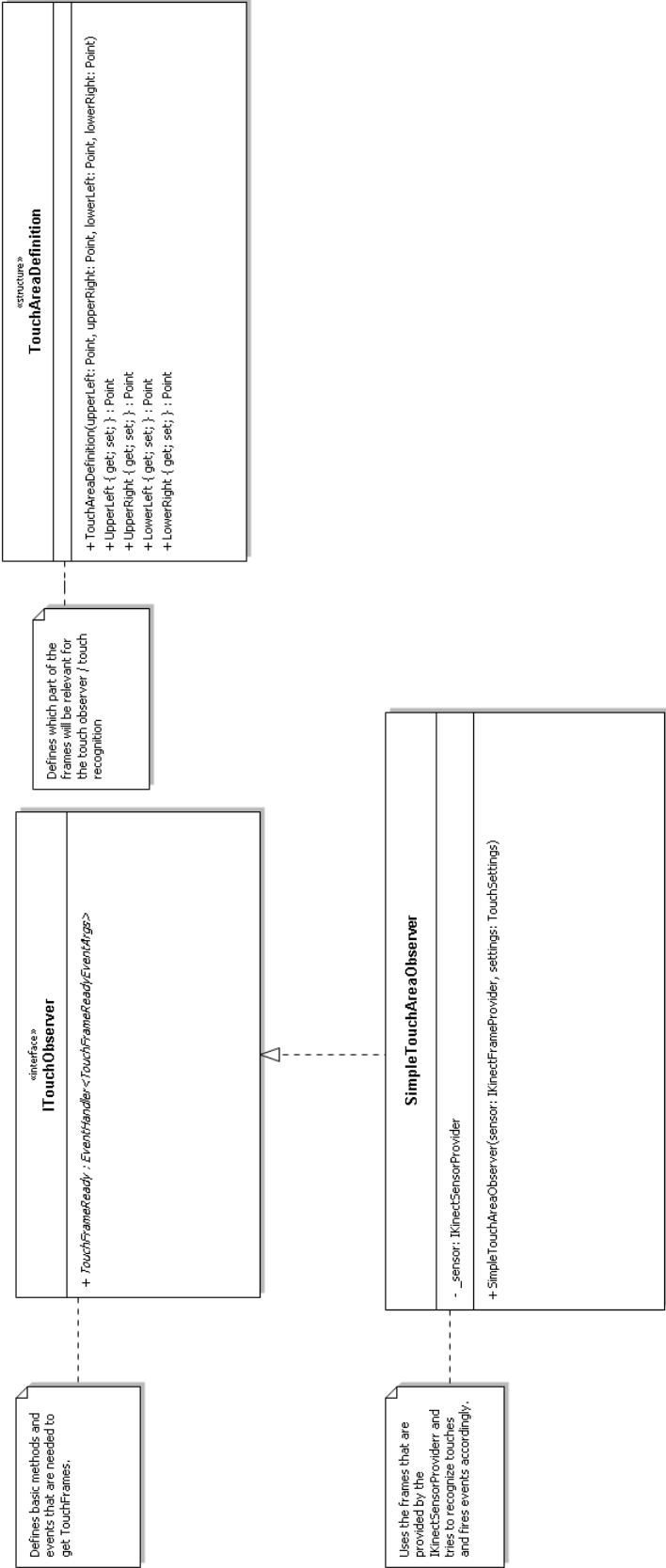
3.31 DEMO Programm - Wizzard - Welcome Page.	39
3.32 DEMO Programm - Wizzard - Kinect Sensor Auswahl.	40
3.33 DEMO Programm - Wizzard - Erstellung des Hintergrundmodells.	41
3.34 DEMO Programm - Wizzard - Auswahl des zu erfassenden Bereichs.	42
3.35 DEMO Programm - Wizzard - Auswahl der im Standard zu registrierenden Pattern.	43
3.36 DEMO Programm - Hauptfenster, das die DEMO Funktionalität bereitstellt.	44

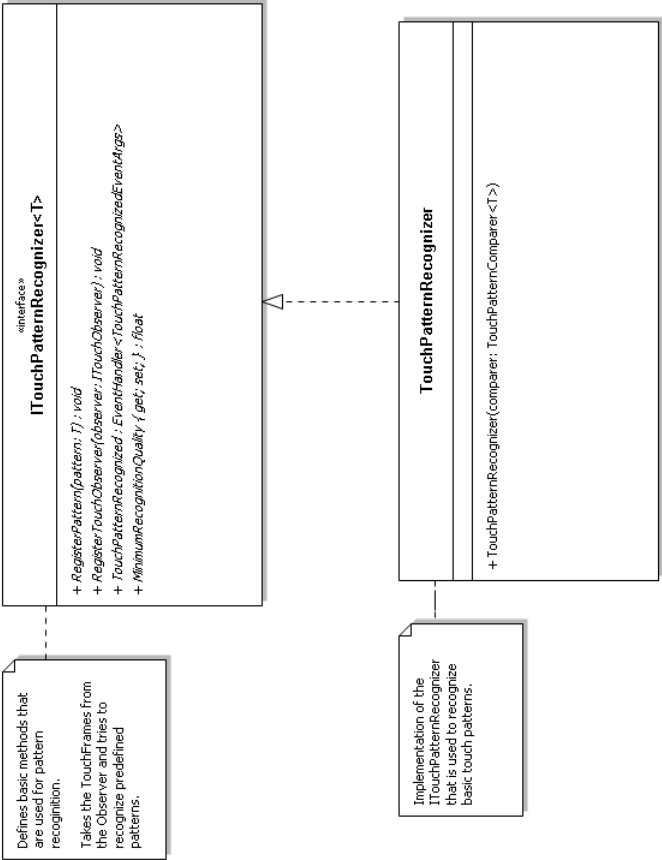
Anhang

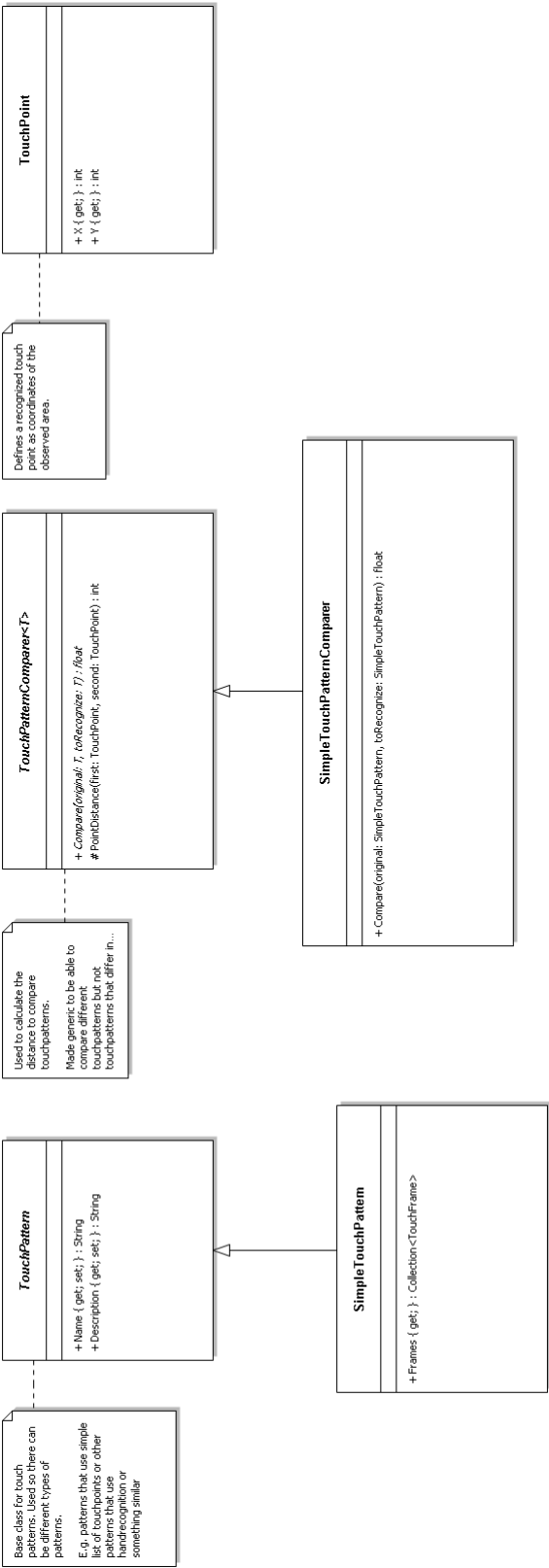
UML Entwurf Planung











Weitere Anhänge

Die weiteren Anhänge wie

- SourceCode
- Hilfe Dateien
- Assemblies der Bibliothek
- UML Diagramme
- DEMO Programm

befinden sich auf beigelegter CD-ROM.

Ich versichere, dass die Technikerarbeit von mir selbstständig angefertigt und nur die angegebenen Hilfsmittel benutzt wurden.

Alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich durch Angabe der Quellen kenntlich gemacht.

Datum

Unterschrift