UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

REALTIME VISIONS
RV realtime visions GmbH

Entwicklung einer Benutzerschnittstelle auf Basis von markerlosem Handtracking und Gestenerkennung auf Basis von räumlichen und optischen Sensordaten

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Computervisualistik

vorgelegt von

Benedikt Tschörner

Erstgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Dipl.-Inform. Johannes Erb, RV realtime visions GmbH

Koblenz, im Juli 2013

Kurzfassung

Zur Zeit ist die Microsoft Kinect in verschiedensten Anwendungsbereichen populär, da sie sowohl günstig als auch genau ist. Für die Steuerung des Mauszeigers ist sie jedoch noch ungeeignet da die Skelettdaten zittern. In meinem Ansatz wird versucht mit gängigen Methoden aus der Bildverarbeitung die Mauszeiger Position zu stabilisieren. Dabei wird als Input die Farbkamera der Kinect dienen. Aus den verschiedenen ermittelten Positionen soll anschließend eine finale Position bestimmt werden. Die rechte Hand steuert dabei die Maus. Eine einfache Klick Geste wird ebenfalls entwickelt. In der Evaluation wird gezeigt ob dieser Ansatz eine Verbesserung darstellt.

Abstract

The Microsoft Kinect is currently polular in many application areas because of the cheap price and good precission. But controlling the cursor is unapplicable due to jitter in the skeletton data. My approach will try to stabilisize the cursor position with common techniques from image processing. The input therefore will be the Kinect color camera. A final position will be calculated using the different positions of the tracking techniques. For controlling the cursor the right hand should be tracked. A simple click gesture will also be developed. The evaluation will show if this approach was succesfull.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☐	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☐	nein ☐

Koblenz, den 24. Juli 2013

Inhaltsverzeichnis

1 Einleitung **13**
- 1.1 Motivation . . . 13
- 1.2 Problemstellung . . . 13
- 1.3 Lösungsansatz . . . 14
- 1.4 Übersicht . . . 14

2 Stand der Technik **17**
- 2.1 Generelle Tracking Methoden . . . 17
- 2.2 Handtracking . . . 18

3 Tracking Ansätze **19**
- 3.1 Modell basierte Detektion . . . 19
- 3.2 Kanade-Lucas-Tomasi Feature Tracker . . . 19
- 3.3 Flocks of Features . . . 20
- 3.4 Continuosly Adaptive Mean Shift . . . 20
- 3.5 Normalized Cross Correlation . . . 21

4 Eigener Ansatz **23**
- 4.1 Aufbau . . . 23
- 4.2 Vorverarbeitung . . . 23
- 4.3 Tracking Module . . . 25
  - 4.3.1 Camshift . . . 26
  - 4.3.2 KLT . . . 27
  - 4.3.3 NCC . . . 27
- 4.4 Nachverarbeitung . . . 28

5 Maus und Gestensteuerung **29**
- 5.1 Maussteuerung . . . 29
- 5.2 Gestensteuerung . . . 29

6 Umsetzung **31**
6.1 Struktur 31
6.2 Implementierung 33

7 Evaluation **35**
7.1 Testsystem und Umsetzung 35
7.2 Testaufbau 36
7.3 Zufallstest 38
7.4 Systematischer Test 38
7.4.1 Generelle Erkenntnisse 38
7.4.2 Analyse der Testsequenzen 39

8 Zusammenfassung **45**
8.1 Ergebnisse 45
8.2 Ausblick 45

Tabellenverzeichnis

2.1 Abkürzung mit Beschreibung der Tracking-Methoden. 18

6.1 Standard-Parameter 34

7.1 Beschreibung der Testsequenzen 36

Abbildungsverzeichnis

2.1 Kinect Skelett 17
2.2 Trackerübersicht 18

4.1 Tiefenbild auf Farbbild 24

6.1 Beziehungen der Module zueinander. 31
6.2 Bearbeitungsreihenfolge 32
6.3 Verlauf Eingabedaten 33
6.4 UML-Klassendiagramm der Applikation 33

7.1 Bilder aus Testsequenz 1. 37
7.2 Bilder aus Testsequenz 2. 37
7.3 Bilder aus Testsequenz 3. 37
7.4 Bilder aus Testsequenz 4. 37
7.5 Bilder aus Testsequenz 5. 37
7.6 Bilder aus Testsequenz 6. 38
7.7 Distanz aller Tests 39
7.8 Distanz für Confidence Values 40
7.9 Distanz für Confidence Values einzeln 40
7.10 Test 1 Bewegung 42
7.11 Test 2 Bewegung 42
7.12 Test 3 Bewegung 43
7.13 Test 4 Bewegung 43
7.14 Test 5 Bewegung 44
7.15 Test 6 Bewegung 44

Kapitel 1

Einleitung

1.1 Motivation

Ende 2006 kam mit dem Erscheinen der Nintendo Wii zum ersten Mal Bewegungssteuerung in Form einer Videospiel-Konsole in normale Haushalte. Im November 2010 brachte auch Microsoft mit der Kinect für die XBox360 ein Konkurrenzprodukt auf den Markt. Diesmal wurde komplett auf Controller verzichtet, so das der Nutzer nur mit seinem Körper das Geschehen lenkt. Umgesetzt wurde dies durch eine Kombination von einem Tiefensensor, einer Farbkamera und mehreren Mikrofonen. Bei einem Preis von 149 Euro zum Einführungszeitpunkt war sie deutlich günstiger als herkömmlichen Tiefenkameras. Februar 2012 kam dann die Kinect für Windows mit passendem Software Delevopment Kit(SDK) auf den Markt. Damit konnten auch Firmen die Technik kommerziell nutzen. So auch RV realtime visions, eine auf interaktive Multimediainhalte spezialisierte Firma, bei der ich für die Zeit der Erstellung meiner Bachelorarbeit als Werksstudent an einem Projekt mitarbeitete.

1.2 Problemstellung

In einem bestehenden Projekt im sportmedizinischen Bereich nutzen Anwender eine Software zur Trainingsunterstützung. Um diese bedienen zu können ist ein Eingabegerät nötig. Eine andere Möglichkeit zur Bedienung wäre jedoch das Tracking der Anwender durch die Kinect. Doch die Genauigkeit der Kinect reicht nicht aus um mit einem Mauszeiger präzise arbeiten zu können. Dies liegt unter anderem an der niedrigen Auflösung des Tiefenbildes von 640x480 Pixeln. Ein weiterer Faktor der zu einem zittern des Mauszeigers führt ist das Rauschen der Tiefendaten.

1.3 Lösungsansatz

Um das Problem der Ungenauigkeit und des Zitterns zu kompensieren soll zusätzlich zu den Skelett aus dem Kinect SDK auch die Farbkamera der Kinect genutzt werden. Die Daten dieser werden in einer Auflösung von 640x480 Pixeln übertragen. Da diese ein deutlich geringeres von Rauschen haben eignen sie sich gut für genaues Tracking. Doch wirft die Nutzung von Farbbildern auch Probleme wie die Differenzierung von Vordergrund und Hintergrund auf. Verschiedene Tracking Methoden aus der Bildverarbeitung sollen hier ihre Vorteile nutzen und ihre Schwächen gegenseitig ausgleichen. Schlussendlich wurden folgende gewählt:

- Continuously Adaptive Mean Shift (Camshift)
- Normalized Cross-Correlation (NCC)
- Kanade-Lucas-Tomasi Feature Tracker (KLT)

Für die initiale Bestimmung der zu trackenden Hand wird auf die Skelettdaten des Kinect SDK zugegriffen. Für die finale Bestimmung der Mauszeiger Position werden die Ergebnisse der einzelnen Methoden auf Plausibilität und Stabilität geprüft und anhand dieser und vorher bekannter Daten verrechnet. Dafür wurden drei Methoden evaluiert:

- Mittelwert aller Ergebnisse
- Bestes Ergebnis
- Besten zwei Ergebnisse

1.4 Übersicht

Es folgt eine kurze Übersicht über den Inhalt der Kapitel.

1. Einleitung
   Einleitung in die Thematik mit kurzer Beschreibung der Problemstellung und des Lösungsansatz.

2. Stand der Technik
   Vorstellung für diese Arbeit relevanter Texte zum Thema Object-Tracking und ihr Einfluss auf die resultierende Anwendung.

3. Tracking Ansätze
   Auswahl an Methoden zum Hand-Tracking und theoretische Evaluation auf ihre Brauchbarkeit.

4. Eigener Ansatz
   Erläuterung des genauen Aufbaus der Anwendung mit detaillierter Beschreibung der einzelnen Komponenten.

5. Maus und Gestensteuerung
   Beschreibt die Methode zur Steuerung der Maus mithilfe der Applikation.

6. Umsetzung
   Es wird auf die genaue Implementierung und Struktur dieser eingegangen.

7. Evaluation
   Beschreibung der verschiedenen Tests mit anschließender Auswertung derer.

8. Zusammenfassung
   Schlussbetrachtung der Arbeit und ihrer Ergebnisse mit kurzem Ausblick auf den Nutzen der gezogenen Schlüsse.

Kapitel 2

Stand der Technik

2.1 Generelle Tracking Methoden

Um Objekte im Raum zu tracken ist oft eine Vordergrund-Separierung sinnvoll. Dies kann zum Beispiel direkt durch Stereo Kameras, Time of Flight Kameras oder durch eine Infrarot-Laser-Projektor-Kamera-Kombination, wie sie in der Kinect vorzufinden ist, erreicht werden. Dadurch bietet das Kinect SDK ein solides Tracking des gesamten Körpers. Bis zu sechs Personen können zugeordnet werden und von bis zu zwei Personen kann die Position des Skelettes abgefragt werden. Als Ergebnis hat man Zugriff auf 20 Punkte, die die Position des Körpers repräsentieren, siehe Figur 2.1. Andere Verfahren, die zuvor keine Vordergrund Separierung vornehmen haben oft das Problem, dass es Hintergrundstrukturen gibt die das Tracking fehl leiten können und somit der Fokus des zu trackenden Objektes an Hintergrundstrukturen hängen bleibt [Fog11]. Auch bestimmte Bewegungen können unabhängig vom Hintergrund zu einem Fehler führen. Dies geschieht oft bei Verdeckung oder zu schnellen Bewegungen. Beispiele hierfür sind in Kapitel 3 aufgeführt.

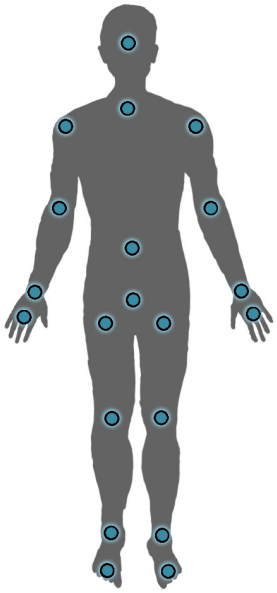


Abbildung 2.1: Position der 20 Ausgabepunkte des Kinect SDK im Bezug zu der getrackten Person.

2.2 Handtracking

Als Basis für eine Vorauswahl der Tracking Methoden wurde die Arbeit von Stenger [SWC10] herangezogen. Sie stellt eine Übersicht über verschiedene Tracking Methoden mit ihren Vor- und Nachteilen dar. Stenger zeigt außerdem, dass die Kombination verschiedener Methoden einen Vorteil bringt. Der robusteste Tracker ist dabei der ColorMotion Tracker, gefolgt vom FoF Tracker. Bei der Genauigkeit ist der NCC Tracker gefolgt von dem auf der Farbwahrscheinlichkeit basierenden Blob Tracker am genausten, siehe Figur 2.2.

Methode	Beschreibung
NCC	Normalized cross correlation
KLT	Kanade-Lucas-Tomasi mit 50 Features
FoF	Flock of Features mit 50 lokalen Features mit hoher Farbähnlichkeit und Schwarm Beschränkungen
MS	Farb Histogramm basierter mean shift Algorithmus mit Hintergrund Gewichtung
C	Farbwahrscheinlichkeits Map mit Blob Detektion
CM	Farb- und Bewegungswahrscheinlichkeit Map mit Blob Detektion

Tabelle 2.1: Abkürzung mit Beschreibung der Tracking-Methoden.

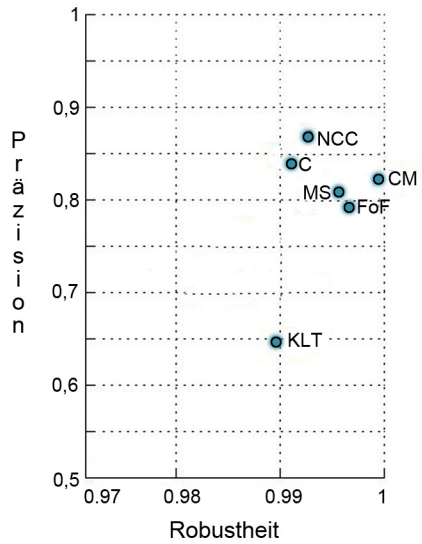


Abbildung 2.2: Übersicht über die Präzision und Robustheit einzelner Tracker nach Stenger. [SWC10, Kapitel 3.8]

Kapitel 3

Tracking Ansätze

3.1 Modell basierte Detektion

Bei der Modell basierten Detektion wird ein Modell mit einem Kamerabild verglichen. Die Kameradaten werden dabei vorverarbeitet. In Stenger [Ste04] werden dafür die Kanten extrahiert. Weiterhin wird die Farbähnlichkeit zur Hautfarbe bestimmt, so dass man als Ergebnis die hautfarbene Fläche der Hand und deren Kanten erhält. Die Kombination dieser beiden Methoden hat zum Vorteil, dass die Nachteile der Einzelnen ausgeglichen werden. Vor einem Hintergrund mit vielen Strukturen führt die Kantendetektion zu keinem brauchbaren Ergebnis. Ebenso verhält es sich bei der Farbähnlichkeit vor hautfarbenen Strukturen, wie hölzerne Türen oder Schränke [Ste04, Kapitel 5.4.1]. Für diese Arbeit wurden die Erfahrungen mit den Schwächen der Vorverarbeitungsschritte genutzt. Der Tracker selbst erschien für die Anwendung ungeeignet, da eine genaue Posen-Erkennung nicht nötig ist. Weiterhin ist die Methode mit 0,3-2 Bildern pro Sekunde auf einem 2,4 GHz Pentium IV, auch auf heutige Hardware übertragen, zu langsam [Ste04, Kapitel 7.2 Limitations].

3.2 Kanade-Lucas-Tomasi Feature Tracker

Der Kanade-Lucas-Tomasi Feature Tracker (KLT) sucht Punkte die in mindestens zwei Richtungen einen hohen Helligkeitsgradienten haben. Nach [ST94] sind diese Punkte gut zu tracken. Er arbeitet auf der Basis von Kanten und ist Farbunabhängig. Schwächen dieses Trackers sind schnelle Bewegungen und Strukturen im Hintergrund. Bleibt ein Punkt an so einer Struktur hängen verfälscht er das Tracking. Bei langem Einsatz dieser Methode sind somit ab einem bestimmten Zeitpunkt keine Punkte mehr am zu trackenden Objekt vorhanden.

3.3 Flocks of Features

Eine Erweiterung des KLT Trackers ist der Flocks of Features Tracker (FoF). Dabei werden den Feature Punkten des KLT Trackers Beschränkungen auferlegt [KT04]. Sie dürfen weder näher als ein Schwellwert aneinander liegen, noch dürfen sie zu weit von dem Mittelwert aller Punkte entfernt sein. Wird eine der Beschränkungen überschritten wird der Punkt neu gesetzt. Einzelne Punkte können leicht an Kanten die nicht zu dem getrackten Objekt gehören hängen bleiben. Im schlechtesten Fall werden alle Punkte an einer starken Kante im Hintergrund abgestreift [Fog11, Kapitel 4]. Eine geeignete Nachverarbeitung der Endposition kann ein Abstreifen der Punkte erkennen, siehe Kapitel 4.4. In der Auswertung von [KT04] schneidet der FoF Tracker trotz seiner Schwächen besser als der Camshift Tracker ab. Diese Ergebnisse stehen im Widerspruch zu [SWC10]. Unter der Annahme, dass die Methoden unterschiedlich implementiert wurden, ist daraus zu schließen, dass die gewählte Implementierung und Optimierung sich stark auf die Ergebnisse der Tracker auswirkt.

3.4 Continuosly Adaptive Mean Shift

Basierend auf dem Mean Shift Algorithmus erweitert der Continuosly Adaptive Mean Shift Algorithmus (Camshift oder CS) diesen um ein variables Suchfenster. Als Vorverarbeitungsschritt ist eine sogenannte Backprojection des Kamerabildes notwendig auf welcher der Algorithmus arbeitet. Eine Backprojection ist ein Bild, das für jedes Pixel die Wahrscheinlichkeit enthält, ob dort das zu trackende Objekt zu finden ist. In den meisten Fällen wird hierfür die Farbähnlichkeit genommen, bei Handtracking somit die Ähnlichkeit zur Hautfarbe. Vierjahn zeigt in [Vie08] dass mit dem Camshift Algorithmus solides Handtracking erreicht werden kann. Auf die Backprojection wendet er noch eine Kombination aus Dilatation und Erosion an, um die Backprojection von kleinen Störpunkten zu bereinigen. In meiner Arbeit erwies sich dieser Schritt jedoch als unnötig. Die Verbesserungen in Betrachtung zur Rechenzeit waren marginal. Die Rechenzeit selbst liegt nach [Vie08, Kapitel 4.3] bei rund 25 Bilder pro Sekunde auf einem zeitgemäßen Computersystem (2,33GHz Intel Core 2 Duo-Prozessor). Für ein korrektes Tracking ist die Startposition des Algorithmus wichtig. Das Suchfenster muss dabei den Bereich der Backprojection mit hohen Werten mindestens schneiden. Die größte Schwäche des Algorithmus, bezogen auf Handtracking mit Farbähnlichkeit, tritt bei kurzärmliger Kleidung auf. Dann kann das variable Suchfeld dazu führen, dass der ganze Arm getracked wird und somit die zurückgegebene Position nutzlos wird. Dieses Problem kann durch eine anschließende Evaluation der Position in der Nachbearbeitung gelöst werden, siehe Kapitel 4.4.

3.5 Normalized Cross Correlation

Der Normalized Cross Correlation Algorithmus (NNC) arbeitet auf Basis von Template-matching. Nach [SWC10] ist er die genaueste Trackingmethode, siehe Figur 3. Er generiert aus einem Suchbild und einem Bild auf dem das Suchbild gesucht werden soll eine Wahrscheinlichkeits-Map. Diese gibt die Wahrscheinlichkeit an, mit der der Mittelpunkt des Suchbildes auf dem Pixel des Bildes auf dem gesucht werden soll zu finden ist. Um den finalen Wert der Position zu bestimmen wird auf der Wahrscheinlichkeits-Map das Maximum gesucht. Dieses entspricht dann der Position bei der das Suchbild und das Bild am ähnlichsten sind [BH01]. Die Vorteile des NCC Algorithmus ist seine Unabhängigkeit von Farbe oder Kanten. Ein Nachteil ist, dass er wenn er nicht dynamisch implementiert wird sich nicht auf die Veränderungen des zu trackenden Objektes einstellt. Um dieses Problem zu lösen kann das Suchbild aktualisiert werden. Dies ermöglicht das Tracken von Objekten, die ihre Form oder Farbe verändern, was durch Lichtveränderungen, wie Schatten oder direktes Licht sowie bei Bewegung eintritt. Das Aktualisieren des Suchbildes birgt jedoch die Gefahr vom zu trackenden Objekt abzudriften. Dieser Effekt tritt bei kontinuierlicher Aktualisierung selbst bei statischen Bildern auf. In [WTH02] wird ein anderer Ansatz zur Aktualisierung genommen. Dabei wird das alte Suchbild mit dem neuen Suchbild, unter Nutzung des Infinite-Impulse-Response Filters, kombiniert. Dieser Ansatz wurde in meine Arbeit übernommen. Die langsame Geschwindigkeit des Algorithmus macht ihn anfangs unpraktikabel für meine Arbeit. Durch eine Optimierung konnte er jedoch trotzdem genutzt werden, siehe Kapitel 4.3.3. Ein anderer Ansatz für das Aktualisieren der Suchbildes ist die Erstellung eines Suchbild Katalogs. In [KMM09] wird dies erfolgreich vorgestellt. Durch erweitern und verkleinern des Suchkataloges wird so eine Auswahl an Suchbildern erstellt, die das zu trackende Objekt repräsentieren. Diese Methode erschien jedoch für die Arbeit zu komplex und wurde nicht weiter verfolgt.

Kapitel 4

Eigener Ansatz

4.1 Aufbau

Da meine Arbeit auf dem BvCode2-Framework aufbaut, ist sie wie dieses modular gestaltet. Der Ablauf lässt sich in 3 Phasen unterteilen, die jeweils durch ein Modul widergespiegelt werden. Die erste Phase ist die Vorverarbeitung, die Zweite das Berechnen der Handpositionen nach verschiedenen Trackingmethoden und die letzte Phase ist die Verarbeitung der Positionen zu eine finalen Position, bevor der Algorithmus von vorne beginnt.

4.2 Vorverarbeitung

Die Vorverabeitung verhindert doppelte Berechnungen und stellt allen Tracking-Modulen die notwendigen Daten zur Verfügung. Hier werden alle Vorverarbeitungsschritte der einzelnen Tracker berechnet. Als erster Schritt wird das Tiefenbild auf das Farbbild projiziert. Damit stimmen die Positionen des Tiefenbildes näherungsweise mit denen des Farbbildes überein. Für die Projektion wurde anfangs eine Skalierung und Verschiebung genutzt. Anschließend sollte eine Linsenverzerrungskorrektur die verbliebenen Fehler kompensieren. Dafür wäre jedoch eine genaue Kalibrierung der Infrarot- und Farbkamera nötig [KE12]. Wie in Figur 4.1 zu sehen, erwies sich der naive Ansatz aber als ausreichend, weswegen eine Linsenverzerrkunskorrektur unnötig erschien. Die Position der Hand aus dem Kinect SDK wird ebenfalls der entsprechenden Position auf dem Farbbild angepasst.

Für den Camshift Algorithmus und die Confidence Values (siehe Kapitel 4.3) der einzelnen Tracker wird in der Vorverarbeitung die Backprojection bereitgestellt. Diese benötigt als Eingabe ein Histogramm. Um geeignete Farbwerte für dieses zu finden wird es erst initialisiert wenn das Kinect SDK die Handposition mit einem Confidence Value von 1 liefert. Der Confidence Value liegt dabei

zwischen 0 und 1, wobei die Größe des Wertes für die Wahrscheinlichkeit eines korrekten Trackings der Position steht. Ist dieser 1 so wird in einem Radius um diese Position die Farbwerte gemittelt und ein Histogramm aus diesen erstellt. Das Histogramm beinhaltet dabei 32 Farbwerte. Die besten Ergebnisse des Histogramms konnten bei einem Radius von etwa 10 Pixeln erreicht werden. Dieser Wert hängt jedoch stark vom einzelnen Anwendungsfall ab. Für die anschließende Berechnung der Backprojection wird zuerst der Farbraum des Farbbildes von RGB zu HSV konvertiert. Weiterhin ist eine Maske für zu helle, zu dunkle und zu wenig gesättigte Farben nötig. Dies liegt daran, dass diese Farben sich schlecht zuordnen lassen. Bei dem Tracken von weißen, grauen oder schwarzen Objekten kann dieser Schritt jedoch zu Problemen führen. Eine Backprojection auf Basis von Helligkeit wäre dafür die bessere Wahl. Da das Ziel dieser Arbeit jedoch das Tracken von Händen ist, ist dieses Problem zu vernachlässigen. Aus dem HSV Bild und dem Histogramm wird dann die Backprojection berechnet. Diese spiegelt dann an jedem Pixel die Übereinstimmung der Farbe des Farbbildes mit dem Histogramm wieder. Das Maskieren der Backprojektion mit der Maske ergibt damit die finale Backprojection.

Abbildung 4.1: Tiefenbild der Kinect auf deren Farbbild projiziert

4.3 Tracking Module

Alle Tracking Module geben einen Confidence Value zurück. Dieser liegt zwischen 0 und 1, wobei größere Werte für eine höhere Verlässlichkeit der Rückgabepositionen stehen. Um diesen Wert zu berechnen wird zuerst in einem Radius von 6 Pixeln um die Rückgabeposition auf dem Tiefenbild nach Werten ungleich 0 gesucht. Die Gesamtzahl wird anschließend durch 36 geteilt und ergibt einen Wert $t \in [0, 1]$. Im Farbbild wird in einem Radius von 12 Pixeln um die Rückgabeposition nach Werten ungleich 0 gesucht. Die Gesamtzahl wird hier durch 80 geteilt da eine Übereinstimmung von rund 50% für einen Confidence Value von 1 ausreicht. Der resultierende Wert c wird auf 1 begrenzt. Ist der Confidence Value der Kinect k 1, fließt der Abstand zu der Kinect Position in den finalen Confidence Value p zu 50% ein, t zu 30% und c zu 20%. Ist dies nicht der Fall macht t 70% und c 30% in p aus. Der Abstand a ist dabei ein Wert zwischen 0 und 1 der den Abstand in Pixeln von 0 bis 80 widerspiegelt. Dies sind Erfahrungswerte die sich als geeignet erwiesen haben.

Um ein gleichmäßiges Tracking zu gewährleisten wird bei Sprüngen im Confindence Wert dieser um bis zu 0,5 bis kleinstenfalls 0 heruntergesetzt. Dafür wird zur Laufzeit bei jedem Sprung in p der größer als 0,2 ist ein Malus m vergrößert. Dieser liegt zwischen 0 und 0,5. Alle 0,1 Sekunden wird dieser Malus wieder um 0,02 verringert. Dadurch wird die Auswirkung von nicht stabilen Tracker gedämpft, was zu einem ruhigeren Tracking der Hand führt. Für jedes Tracking Module ist p somit:

$0 \leq k, t, c, p \leq 1$
$0 \leq m \leq 0,5$
$p = 0{,}7t + 0{,}3 + c - m \mid k \neq 1$
$p = 0{,}3t + 0{,}2c + 0{,}5a - m \mid k = 1$

Ist $k = 1$ und der Abstand zu einem Tracker Modul größer als 80 Pixel wird dieses neu initialisiert. Dies verhindert das Tracking von falschen Objekten über längere Zeit. Ein weiterer sicherer Anhaltspunkt für einen irregeführten Tracker ist ein Sprung seiner Rückgabeposition. Ist der Abstand zu seiner letzten Position dabei größer als 80 Pixel wird er neu inititalisiert. Der Wert von 80 Pixeln stellt dabei kein Problem für schnelle Bewegungen dar.

4.3.1 Camshift

Für den Camshift Algorithmus wurde die Implementierung aus OpenCV [Int] nach [Bra98] genutzt. Dabei arbeitet der Algorithmus auf der von der Vorverarbeitung bereitgestellten Backprojection-Map. Der Algorithmus wird erst initialisiert wenn das Kinect SDK die Position der rechten Hand mit einem Confidence Value von 1 übergibt. Geschieht dies wird das initiale Suchfeld in einem Radius von 10 Pixeln um diese Position erstellt. Damit ist gewährleistet, dass der Camshift Algorithmus die Hand und nicht etwa das Gesicht verfolgt. Die angepasste Arbeitsweise von Camshift ist damit:

1. Setze das initiale Suchfenster auf die Handposition des Kinect SDK.
2. Berechne die Backprojection-Map.
3. Finde mit Hilfe des Mean Shift Algorithmus das Zentrum und dessen Größe:
   (a) Finde das Zentrum.
   (b) Setze das Suchfeld auf die Position des Zentrums.
   (c) Solange die Änderung der Position des Zentrums größer als ein Schwellwert ist gehe zu Schritt (b).
4. Setzte das künftige Suchfeld auf die Position und Größe des Zentrum. Gehe zu Schritt 2.

[Int]

Das Zentrum c in Schritt 3(b), das dem Schwerpunkt entspricht, wird über das nullte und erste Moment in x und y Richtung bestimmt. Dabei ist nullte Moment m_0 die gesamte Masse und das erste Moment in x m_x und y m_y Richtung das statische Moment. Für ein Bild mit einem Kanal (Grauwertbild) oder eine Region of Interest (ROI) I mit Höhe h und Breite w erhält man c nach:

$$m_0 = \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} I(x,y); \qquad m_x = \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} xI(x,y); \qquad m_y = \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} yI(x,y) \tag{4.1}$$

[Int]

$$x_c = \frac{m_x}{m0}; \qquad y_c = \frac{m_y}{m0} \tag{4.2}$$

[Int]

Die resultierende Position nach dem Konvergieren des Zentrums nach Schritt 3(c) ist dann die finale Position des Camshift Moduls.

4.3.2 KLT

Ist der Confidence Value der Kinect 1 wird die Initialisierung des KLT Trackers vorgenommen. Bei dieser werden 50 Punkte in einem Radius von 15 Pixeln um die Position der Hand aus dem Kinect SDK verteilt. Anschließend wird für jeden Punkt in einem Radius von fünf Pixeln um dessen Position nach einer Ecke oder zwei aufeinander stoßenden Ecken gesucht. Die Anzahl der Punkte und der Abstände entsprechen dabei Erfahrungswerten die sich als gut erwiesen haben. Das Suchen der Ecken geschieht nach der in [Int] dokumentierten Funktion FindCornerSubPix. Dabei werden von jedem Ursprungspunkt weitere Punkte in dessen Nachbarschaft gesucht und der Vektor zwischen Ursprung und Punkt mit der Richtung der Helligkeitsgradienten auf Orthogonalität untersucht. Der Punkt mit dem kleinsten Winkel zu den orthogonalen Vektoren der Helligkeitsgradienten ergibt dann die neue Position des Ursprungspunktes.

Für die Berechnung der Bewegung der Punkte wird der Optical Flow zwischen dem aktuellen und vorhergehenden Bild berechnet. Dies geschieht nach dem Ansatz aus [Bou02, Kapitel 2.3 Iterative Optical Flow Computation (Iterative Lucas-Kanade)], welcher in der Funktion aus [Int] angewendet wird. Diese bestimmt anhand des aktuellen und vorhergehenden Bildes und einer Menge an zu trackenden Punkten auf dem vorhergegangenen Bild die Position der Punkte auf dem aktuellen Bild. Aus dieser Position wird der Mittelpunkt aller Punkte bestimmt. Um das Verschieben des Mittelpunktes bei Ausreißern zu verhindern, wird dieser anschließend um den nach Entfernung gewichteten Abstand der Punkte zum Mittelpunkt verschoben. Dieser neue Punkt ergibt dann die finale Position des KLT Trackers.

Um Punkte zu vermeiden die nicht mehr auf dem zu trackenden Objekt liegen wird auf die Entfernung von der finalen Position getestet. Ist diese größer als 50 Pixel wird der Punkt, wie am Anfang dieses Kapitels beschrieben, neu initialisiert.

4.3.3 NCC

Für den NCC Algorithmus gibt es eine effektive Umsetzung von OpenCV [Int]. Diese berechnet aus einem Bild I mit Höhe H und Breite W und einem Suchbild T mit der Höhe h und Breite w eine Wahrscheinlichkeits Map C mit der Höhe $W - w + 1$ und der Breite $H - h + 1$. Für jedes Pixel in C gilt:

$$C(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')I(x+x',y+y')}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} I(x+x',y+y')^2}} \tag{4.3}$$

[Int]

Auf C wird anschließend das Maximum M gesucht. Für die Position in I gilt $x = w + M_x$ und $y = M_y + h$. Dies ist dann die finale Position des NCC Trackers.

Der Algorithmus konnte jedoch nicht direkt genutzt werden, denn die langsame Geschwindigkeit stellt ein Problem dar, welches durch die verschachtelten Iterationen zustande kommt. Für ein Bild mit Breite x_b und der Höhe y_b und einem Suchbild mit der Breite x_s und der Höhe y_s ergeben sich $(x_b - x_s) \cdot (y_b - y_s) \cdot x_s \cdot y_s$ Iterationen. Bei einer Auflösung von 640x480 Pixeln des Bildes und 80x80 Pixeln des Suchbildes führt dies zu 1.433.600.000 Iterationen. Um dieses Problem zu beheben wurde in meiner Arbeit zuerst in auf mit 0.25 skalierten Bildern gesucht. Anschließend auf den Originalbildern in einem von den Tiefenwerten an der auf den skalierten Bildern ermittelten Stelle abhängigen Radius. Bei einem anschließendem Suchradius von 20 Pixeln führt dies zu 8.160.000 Iterationen. Damit konnte bei gleicher Präzision die Geschwindigkeit des Algorithmus um das rund 175 fache gesteigert werden. Für das Aktualisieren der Suchbilder wurden diese gewichtet addiert. Ein Wert $a \in [0, 1]$ legt dabei die Gewichtung fest. Anschließend wird über jedes Pixel in T iteriert und der mit a multiplizierte Wert mit dem mit $1 - a$ multiplizierten Wertes des Pixel des letzten Suchbildes addiert.

4.4 Nachverarbeitung

Haben alle Tracking Module eine Position ermittelt, werden diese in der Nachverarbeitung zu einer finalen Position zusammengefasst. In dieser wird auch der Confidence Value zur Gewichtung der einzelnen Ergebnisse hinzugezogen. Um im Voraus bekannte Schwächen oder Stärken einzelner Tracker zu berücksichtigen gibt es einen Global Confidence Value. Dieser wird vom Nutzer angegeben und ist statisch. Anhand dieser Gewichtungen und einer Methode zur Verrechnung der Positionen wird die finale Position bestimmt. Die genutzten Methoden sind:

- Mittelwert aller Ergebnisse
  Es werden alle Positionen entsprechend ihrer Gewichtung addiert und anschließend durch ihre Anzahl geteilt. Die finale Gewichtung ist das Produkt aus dem Confidence Value und dem Global Confidence Value.
- Bestes Ergebnis
  Es wird der Confidence Value mit dem Global Confidence Value multipliziert. Nur die Position des Trackers mit dem besten Ergebnis wird anschließend genutzt.
- Besten zwei Ergebnisse
  Anhand des Produkts aus Confidence Value und Global Confidence Value werden die besten zwei Tracker ermittelt. Deren Positionen werden anschließend entsprechend ihrem Global Confidence Value gewichtet.

Kapitel 5

Maus und Gestensteuerung

5.1 Maussteuerung

Die Steuerung der Maus erfolgt über eine Projektion eines Ausschnittes auf dem Farbbild in die Mauskoordinaten. Dieser Schritt ermöglicht eine komfortablere Steuerung, da nicht der gesamte mögliche Eingabebereich genutzt wird und somit die zurückzulegenden Strecken deutlich kürzer sind. Die Berechnung des Ausschnittes A erfolgt dabei über zwei wählbare Variablen d_x und d_y wobei d_x den Abstand des Ausschnittes zu den vertikalen Rändern darstellt und d_y den Abstand zum oberen Rand des Farbbildes I. Dieses hat die Breite I_w. Damit ergibt sich für die Breite A_w und Höhe A_h von a bei Bildschirmhöhe S_h und Breite S_w in Pixeln, $A_w = I_w - 2d_x$ und $A_h = (\frac{S_h}{S_w}) \cdot A_w$. Um die Position im Farbbild in Bildschirmkoordinaten umzuwandeln wird zuerst der Ursprung A_U von A auf I vom Punkt der finalen Position F subtrahiert, $F' = F - A_U$. Damit liegt F' nun im Koordinatensystem von A. Dabei gilt $0 \leq F'_x \leq A_w$ und $0 \leq F'_y \leq A_h$. Anschließend gilt $F''_x = (\frac{F'_x}{A_w} \cdot S_w)$ und $F''_y = (\frac{F'_y}{A_h} \cdot S_h)$. Der Punkt F'' kann nun an das Betriebssystem als Mausposition übergeben werden.

5.2 Gestensteuerung

Um einen Mausklick auszulösen ist eine schnelle Bewegung der Hand zur Kinect hin und wieder zurück erforderlich. Diese Methode erwies sich als einfach und robust. Ein Nachteil ist jedoch, dass der Klick erst wieder beim Zurückkehren der Hand in die Ausgangsposition der Bewegung ausgelöst wird. Die Umsetzung erfolgt dabei über das Speichern der letzten Tiefenwerte. Auf diesen wird getestet ob auf einen Wert ein niedrigerer und wieder ein hoher Wert folgt. Um bei einer Handbewegung nach vorne oder hinten, und dem damit verbundenen Zurückwip-

pen der Hand, einen Klicken zu verhindern, muss die Hand mindestens 3cm über den durchschnittlichen Wert der letzten gespeicherten Werte bewegt werden. Die Anzahl der gespeicherten Werte auf der die Suche nach dem Verhalten ausgeführt wird bestimmt dabei wie schnell die Bewegung sein muss.

Kapitel 6

Umsetzung

6.1 Struktur

Meine Arbeit baut auf dem von RV realtimevisions entwickelten und genutzten Bildverarbeitungsframework auf. Das BvCode2 genannte Framework ist modulbasiert weswegen sich auch diese Arbeit in Form eines Modules in das Framework integriert. Es ist in C++ geschrieben und nutzt zur Konfiguration xml-Dateien. In diesen Dateien können auch weitere Parameter gesetzt werden die das Verhalten der Module ändern, siehe Figur 6.1. Mit Hilfe derer lässt sich eine Pipeline erstellen, die aus verschiedenen Modulen besteht. Dabei kann jedes Modul auch weitere Kindmodule haben. Diese Pipeline ist in meiner Arbeit folgendermaßen definiert. Das Hauptmodule Application hat die Kindmodule SkeletonTracking und HandRefinement, siehe Bild 6.2. HandRefinement wiederum hat zum einen die Kindmodule der Basis HandTracker mit Typ CS, KLT und NCC und ein Kindmodule der Basis PositionHandler vom Typ Mean, Best oder BestTwo. Die Basis eines Moduls entspricht dabei der abgeleiteten Klasse. Zur Laufzeit werden dann alle Module solange wiederholt durchlaufen bis das System pausiert oder gestoppt wird.

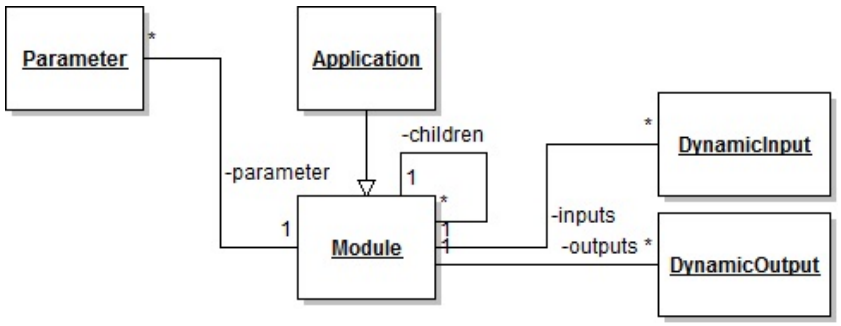


Abbildung 6.1: Beziehungen der Module zueinander.

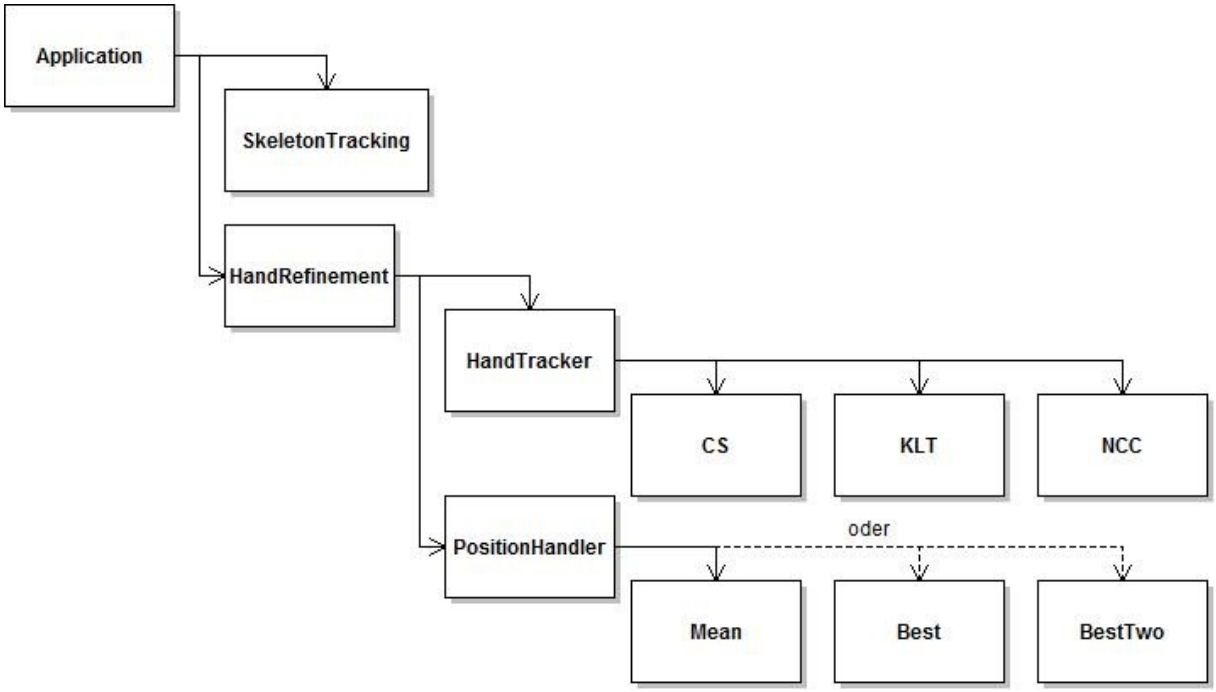


Abbildung 6.2: Reihenfolge der Bearbeitung, von oben nach unten. Kindmodule sind nach rechts versetzt.

Als Eingabedaten wird ein Farbbild, ein Tiefenbild und die Skelettdaten aus dem SkeletonTracking Modul verwendet. Um zu der finalen Handposition zu gelangen wird zuerst aus den Skelettdaten die Position der rechten Hand extrahiert. Anschließend wird diese auf das Koordinatensystem des Farbbildes projiziert. Das Tiefenbild wird ebenfalls in das Koordinatensystem des Farbbildes gebracht, so dass eine Position auf dem Tiefenbild der im Farbbild entspricht. Wie in Kapitel 4.2 beschrieben wird aus der Handposition und dem Farbbild ein Histogramm gebildet. Aus diesem wird die Backprojection berechnet, welche mit dem angepassten Tiefenbild, dem Farbbild und der angepassten Handposition an die HandTracker Module vom Typ CS, NCC und KLT weitergegeben wird. Diese berechnen ihre Handposition anhand ihres Typs. Die resultierenden Handpositionen werden anschließend im PositionHandler Modul für die Berechnung der finalen Handpositionen benutzt. Figur 6.3 veranschaulicht den Zustand der Daten mit ihren Abhängigkeiten. In anderen Applikationen kann diese dann als Eingabe genutzt werden.

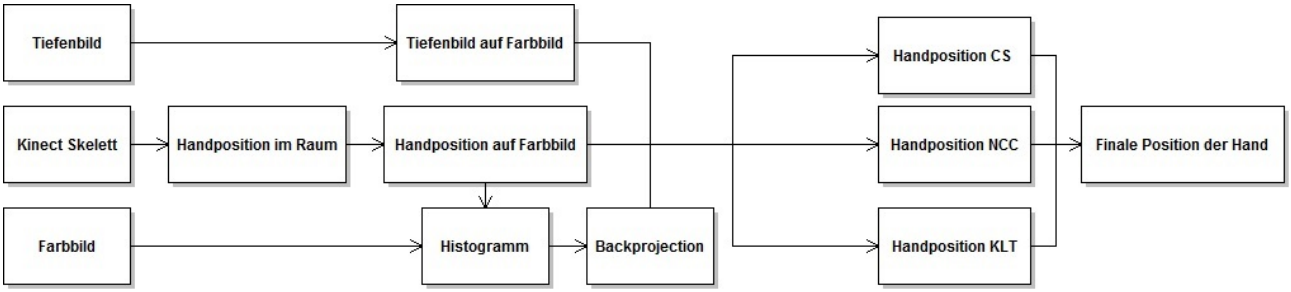


Abbildung 6.3: Verlauf des Zustandes der Eingabedaten bis zur finalen Handposition.

6.2 Implementierung

Die Anwendung ist wie das BvCode2-Framework ebenfalls in C++ geschrieben. Jedes Modul ist eine eigene Klasse. Die Klassen HandTracker und PositionHandler sind dabei Basisklassen. Von HandTracker erbt HandTrackerCS, HandTrackerNCC und HandTrackerKLT. Von PositionHandler wiederum erbt PositionHandlerMean, PositionHandlerBest und PositionHandlerBestTwo. Die von HandTracker erbenden Module sind dabei eine eins zu null bis unendlich Komposition zur HandRefinement Klasse. Die von PositionHandler erbenden Module eine eins zu eins Komposition zu HandRefinement, siehe Figur 6.4.

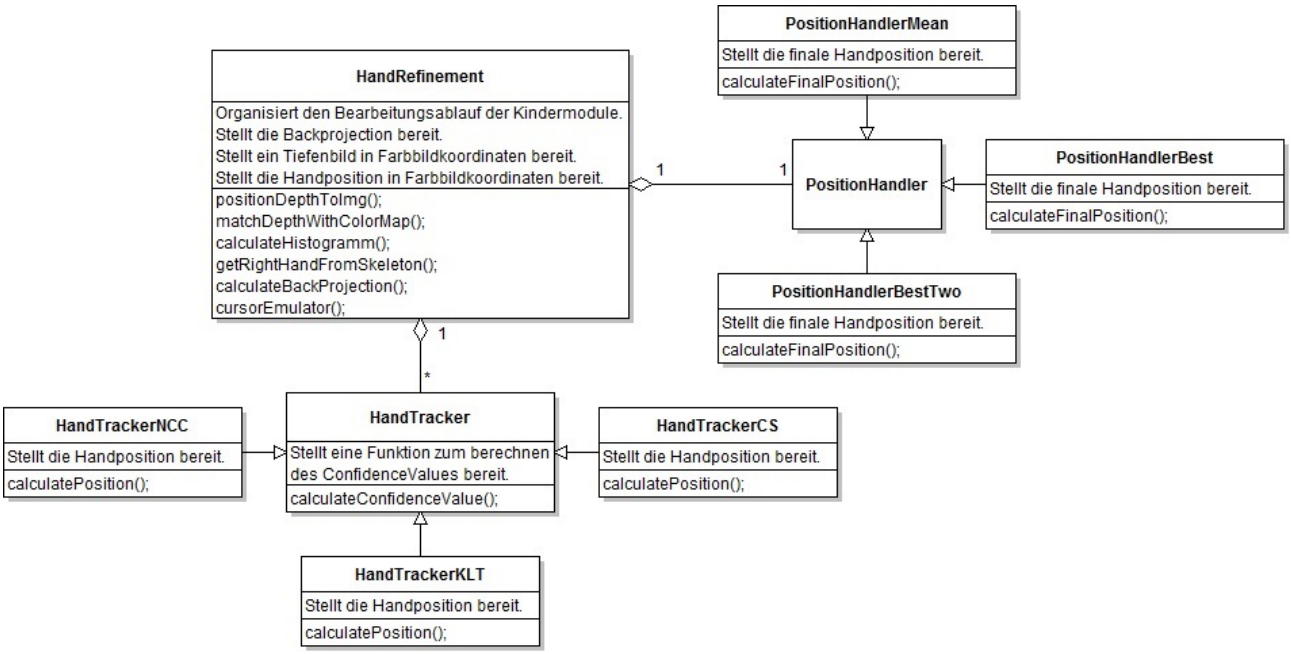


Abbildung 6.4: UML-Klassendiagramm der HandRefinement Applikation.

Für die Konfiguration der Module stehen in der xml-Datei Parameter bereit. Um die Backprojection zu kontrollieren ist es möglich die Schwellwerte der Maske zu ändern. Die genauen Parameter sind SMin welches der minimalen Sättigung entspricht und VMin und Vmax, die die Schwellwerte für minimale und maximale Helligkeit definieren. Für die Berechnung des Tiefenbildes in Farbbildkoordinaten stehen die Parameter OffsetX, OffsetY, ScaleX und ScaleY zur Verfügung. Mithilfe deren lässt sich das Tiefenbild skalieren und verschieben. Für die Maussteuerung lässt sich der Ausschnitt des Eingabebildes, der auf den Bildschirm projiziert wird mit CursDistX und CursDistY ändern. Um auch größere Auflösungen unterstützen zu können gibt es den Parameter SampleRadius. Dieser ist ein globaler Wert, anhand dessen sich Arbeitsbereiche und deren Radien orientieren. Der globale Confidence Value der einzelnen Tracking Module lässt sich ebenfalls einstellen, ebenso wie die Wahl der finalen Positionsverarbeitung. Alle Parameter sind standardmäßig sinnvoll eingestellt und lassen sich in der XML-Datei bei Bedarf deaktivieren. In diesem Fall wird dann ein im Programmcode festgesetzter Wert genutzt welcher der Standardeinstellung entspricht.

Parameter	Standard	Beschreibung
SMin	30	Sättigungsminimum der Backprojection
VMin	96	Helligkeitsminimum der Backprojection
VMax	256	Helligkeitsmaximum der Backprojection
ScaleX	34	Pixeldifferenz des Tiefenbildes zu den vertikalen Rändern des Farbbildes
ScaleY	23	Pixeldifferenz des Tiefenbildes zur den horizontalen Rändern des Farbbildes
OffsetX	-7	Verschiebung des Tiefenbildes in x Richtung
OffsetY	3	Verschiebung des Tiefenbildes in y Richtung
CursDistX	120	Pixeldifferenz des Bildausschnittes zu den vertikalen Rändern
CursDistY	50	Pixeldifferenz des Bildausschnittes zum oberen Rand
SampleRadius	10	Abhängigkeitsmaß für den Arbeitsbereich von Algorithmen
GlobalConfKinect	25	Globaler Confidence Value der Kinect Position
GlobalConfCS	25	Globaler Confidence Value des CS Moduls
GlobalConfNCC	25	Globaler Confidence Value des NCC Modul
GlobalConfKLT	25	Globaler Confidence Value des KLT Modul
PositionHandler	mean	Methode zur finalen Positionsbestimmung (mean, best, besttwo)

Tabelle 6.1: Standard-Parameter

Kapitel 7

Evaluation

7.1 Testsystem und Umsetzung

Um bei der Evaluation mit konstanten Daten zu arbeiten wurde eine Funktion erstellt die Sequenzen sowohl aufnehmen als auch abspielen kann. Dabei werden die Tiefendaten und Farbbilder der Kinect, sowie die Position der Hand gespeichert. Das Speichern der Handposition ist nötig, da das Kinect SDK bei gleichen Eingangsdaten verschiedene Ausgangsdaten liefert. Die Laufzeit der Komponenten ist bei der Evaluation zu vernachlässigen. Der einzige beschränkende Faktor der Geschwindigkeit ist die Kamera der Kinect. Diese liefert 30 Bilder pro Sekunde. Mit dieser Rate arbeitet auch die Applikation. Beim Laden von Sequenzen werden die eingehenden Daten nacheinander abgearbeitet. Hier ist der beschränkende Teil die Lesegeschwindigkeit der eingebauten Festplatte. Dennoch wird eine Geschwindigkeit von durchschnittlich 127fps erreicht. Mit minimal 66fps und maximal 176fps, einer Varianz von 317 und Standardabweichung von 17,8.

Die Daten des für die Evaluation genutzten Computers sind:

Prozessor	i7-3770K
Kerne	4 physisch, 8 virtuell
Frequenz	3,5Ghz
Arbeitsspeicher	32 GB
Betriebssystem	Windows 8 Professional 64Bit
Festplatte	512GB Intel SSD

7.2 Testaufbau

Für die Evaluation wurden sechs Testsequenzen aufgenommen. Diese sollen möglichst alle Problemsituationen aufzeigen, aber auch einen normalen Anwendungsfall simulieren. Dazu wurde sowohl die Kleidung als auch die Umgebung angepasst. Für die Tests wurde ein normaler Büroraum genutzt. Durch An- und Ausschalten der verschiedenen Beleuchtungen und Öffnen und Schließen der Jalousien wurden verschiedene Lichtsituationen getestet. Um einen schwierigen Hintergrund zu testen wurden verschiedene Objekte kurz hinter dem Bewegungsfeld der Hand platziert. Darunter waren auch hautfarbene Objekte. Mit anderen Personen wurde auch das Verhalten bei Verdeckung überprüft. Eine weitere Sequenz testet die Qualität der Ergebnisse bei verschiedenartigen Handbewegungen. Eine Übersicht aller Sequenzen ist in Tabelle 7.1 zu finden.

Sequenz	Beschreibung
1	T-Shirt kurz Normale Beleuchtung Normaler Hintergrund
2	T-Shirt lang Normale Beleuchtung Normaler Hintergrund
3	T-Shirt lang Wechselnde Beleuchtung Normaler Hintergrund
4	T-Shirt kurz Normale Beleuchtung Hautfarbe und Objekte im Hintergrund
5	T-Shirt lang Normale Beleuchtung Normaler Hintergrund Verdeckung durch andere Personen
6	T-Shirt lang Normale Beleuchtung Normaler Hintergrund Verschiedene Handbewegungen

Tabelle 7.1: Beschreibung der Testsequenzen

Zu jeder Testsequenz liegt die Position der Hand vor. Diese wurde manuell aus den Bildern bestimmt, wobei auch gespeichert wurde ob die Hand überhaupt zu sehen ist. Die gespeicherte Position der Hand ist dabei immer der Mittelpunkt der Handfläche, unabhängig ob dieser von Fingern oder dem Handrücken verdeckt ist.

Abbildung 7.1: Bilder aus Testsequenz 1.

Abbildung 7.2: Bilder aus Testsequenz 2.

Abbildung 7.3: Bilder aus Testsequenz 3.

Abbildung 7.4: Bilder aus Testsequenz 4.

Abbildung 7.5: Bilder aus Testsequenz 5.

Abbildung 7.6: Bilder aus Testsequenz 6.

7.3 Zufallstest

Um gute Werte für die Parameter (siehe Tabelle 6.1) SMin, VMin, VMax und SampleRadius zu finden wurden 1000 zufällige Kombinationen aller Parameter auf allen sechs Testsequenzen getestet und anschließend analysiert. Dazu wurde der Abstand der getrackten Handposition mit der echten Position verglichen. Messwerte von der Initialisierung wurden dabei nicht berücksichtigt, da für die Parameter nicht die Initialisierungsgeschwindigkeit sondern die Langzeitgenauigkeit im Vordergrund stand. Aus den besten Tests wurde anschließend ein Mittelwert für die Parameter gebildet. Die resultierenden Werte waren ähnlich bis identisch zu den subjektiven Erfahrungswerten. Sie sind in der Anwendung als Standardwerte gesetzt und in der Tabelle 6.1 nachzulesen.

7.4 Systematischer Test

Basierend auf den im Zufallstest ermittelten Parametern wurde ein systematischer Test aus Kombinationen der vier globalen Confidence Values durchgeführt. Es wurde jede Kombination mit einer Schrittweite der globalen Confidence Values von fünf getestet, also 20 Werte von 0-100. Die resultierenden 5313 Konfigurationen wurden anschließend auf allen Testsequenzen analysiert. Um die Testdaten analysieren zu können wurde ein Programm geschrieben das die Daten aufbereitet und Ergebnisse in Graphen darstellt.

7.4.1 Generelle Erkenntnisse

Die Auswertung zeigt, dass die Beschränkungen der Tracker, die ein Tracking von falschen Objekten verhindern greifen. So kann der mittlere Abstand zwischen getrackter Position und echter Position nie 30 Pixel überschreiten, siehe Bild 7.7. Markant sind die Qualitätsunterschiede der Auswertungsmethoden im Graph in der Reihenfolge: mean, best, bestTwo. Der Abstand ist bei mean deutlich besser als bei best und bestTwo.

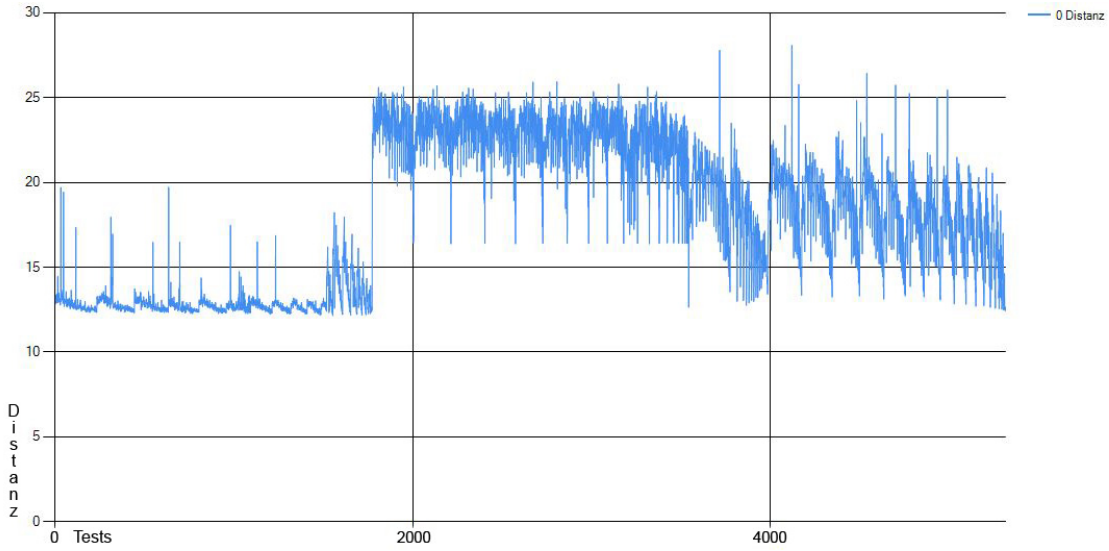


Abbildung 7.7: Mittlere Distanz zwischen Rückgabeposition und echten Position der Hand gemittelt aus allen sechs Testsequenzen pro Test. Deutlich zu sehen die Auswertungsmethoden in der Reihenfolge: mean, best, bestTwo

Bild 7.8 zeigt die Distanzen für entsprechende Global Confidence Values. Zu sehen ist die Qualität der einzelnen Tracker bei einem Confidence Value von 100. Hier liegt der CS Tracker vor der Kinect und etwas schlechter dem NCC und KLT Tracker. Ebenfalls lassen sich Tendenzen über den Einfluss der einzelnen Tracker auf die Qualität des Gesamtergebnis anhand der Werte für einen Confidence Value von 0 treffen. Sind sie niedrig kann man davon ausgehen, dass der Tracker eher das Ergebnis stört anstatt es zu verbessern. Im Gegenschluss weisen hohe Werte auf ein Fehlen des Trackers im negativen Sinne hin. So bestätigt sich die Erwartung über die Qualität des CS Trackers, siehe Bild 7.9. Er ist der genauste Tracker und für seinen Confidence Value gegen null lässt die Gesamtqualität deutlich nach. Wiedererwartens sind die Ergebnisse der Kinect mit derselben Interpretation schlecht. Bei hohem Confidence Value entstehen hohe Distanzen, bei niedrigem Confidence Value werden die Ergebnisse besser, was wiederum gegen die Qualität der Ergebnisse der Kinect spricht.

7.4.2 Analyse der Testsequenzen

Das beste Ergebnis auf Sequenz 1 ist mit den Confidence Values von: Kinect 15, CS 45, KLT 20 und NCC 20. In Bild 7.10 beginnt die Handbewegung oben, die Kinect braucht sieben Bilder bis sie die Hand gefunden hat und schließt dann auf. Links oben endet die Bewegung, hierbei verdeckt die linke Hand die Kamera, so dass die letzten fünf Bilder falsch tracken und zu vernachlässigen sind. Auffällig sind die

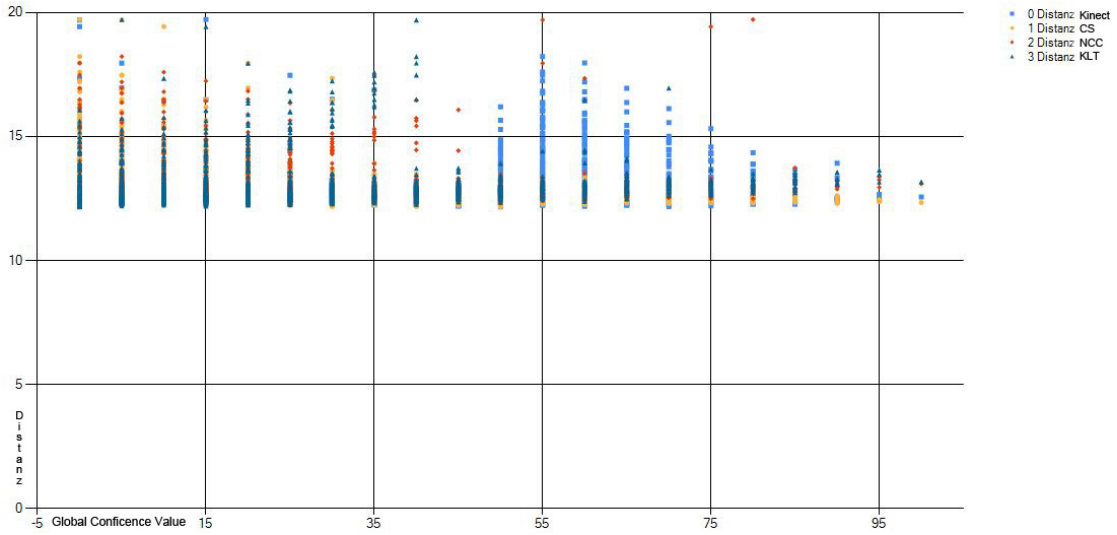
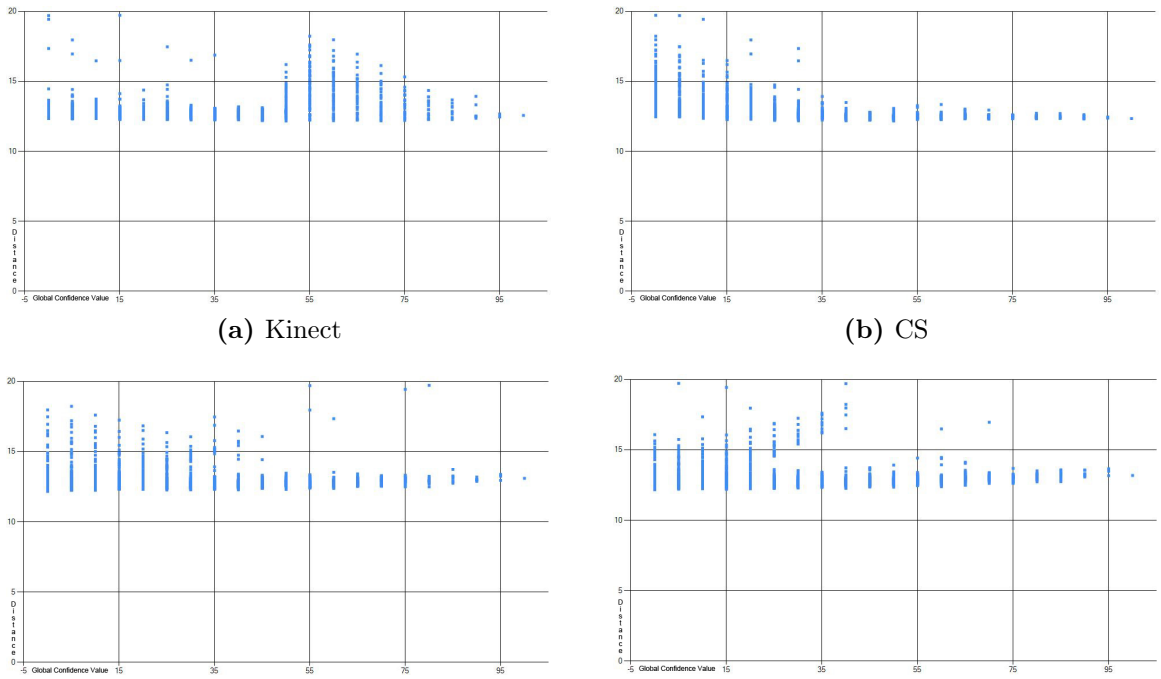


Abbildung 7.8: Mittlere Distanz der Tracker bei steigendem Global Confidence Value über alle Testsequenzen und Tests.



(a) Kinect **(b)** CS

(c) NCC **(d)** KLT

Abbildung 7.9: Mittlere Distanz der Tracker bei steigendem Global Confidene Value über alle Testsequenzen und Tests.

Ausreißer, welche durch ein Springen der Position der Kinect zustande kommen. Anderen Trackern ist es nicht möglich sich so weit von der Kinect Position zu entfernen und damit solche Ausreißer zu erzeugen. Das kurze T-Shirt welches in der Testsequenz getragen wird zeigt keinen sichtbaren Einfluss auf die Ergebnisse. Auf Sequenz 2 ist mit den Confidence Values von: Kinect 0, CS 95, KLT 0 und NCC 5 das beste Ergebnis zu erreichen, siehe Bild 7.11. Eine Betrachtung der Kurven der Ergebnisposition zeigt, dass diese glatt und gleichmäßig sind. Die wechselnden Lichtverhältnisse, siehe Bilder 7.3, in Sequenz 3 führen zu den besten Ergebnissen bei einem hohen Confidence Value der Kinect, da diese lichtinvariant ist. Das beste Ergebnis ist mit den Confidence Values von Kinect 50, CS 25, KLT 20 und NCC 5 zu erreichen. In Sequenz 4, siehe Bild 7.13, mit hautfarbenen Objekten und vielen Kanten im Hintergrund sind die Confidene Values des besten Ergebnisses Kinect 95, CS 5, KLT 0 und NCC 0. Hier verhindert der Hintergrund, siehe Bild 7.4, ein Tracking des CS, KLT und NCC Tackers. Bei Verdeckung durch andere Personen in Sequenz 5 lässt sich die Kinect täuschen. Der Graph für Sequenz 5 in Bild 7.14 zeigt ein Tracken der falschen Person. Die Confidence Values von Kinect 25, CS 20, KLT 10 und NCC 45 zeigen eine ausgewogene Verteilung. In Sequenz 6 mit verschiedenen Handbewegungen sind die korrespondierenden Bewegungen der Ergebnisse mit der Hand nicht immer eindeutig. Es ist davon auszugehen, dass teilweise die Bewegungen zu schnell oder zu ungünstig für die Tracker waren. Der KLT Tracker ist besonders anfällig für schnelle Bewegungen, da er auf dem vorhergegangenen Bild arbeitet. Das Suchen nach dem Suchbild auf dem gesamten Bild macht den NCC Tracker unabhängig von der Handgeschwindigkeit, was auch in den Confidence Values von Kinect 55, CS 5, KLT 5 und NCC 35 zu erkennen ist.

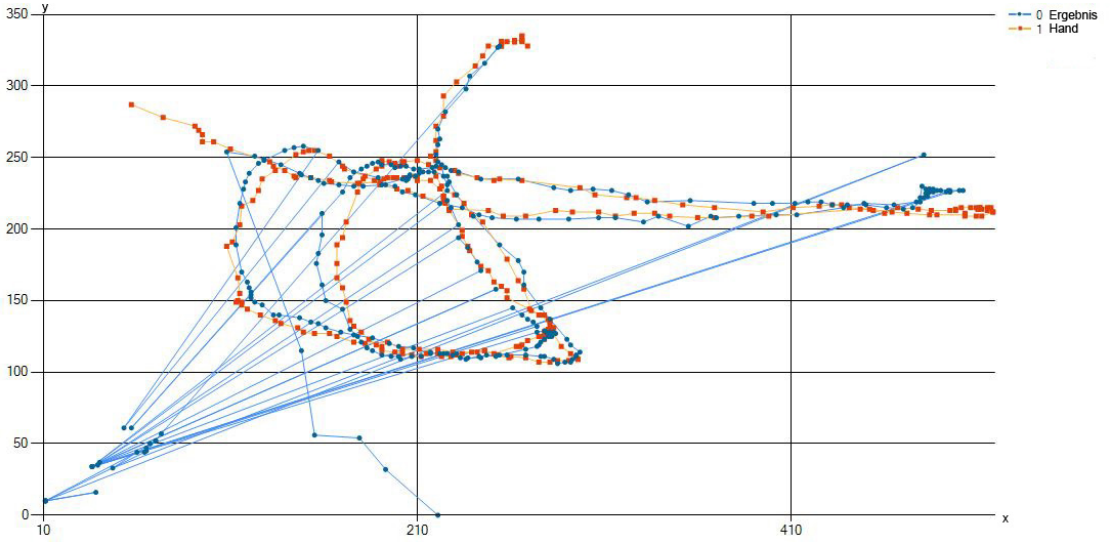


Abbildung 7.10: Bester Test für Testsequenz 1. Ergebnis des Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 15, CS 45, KLT 20, NCC 20.

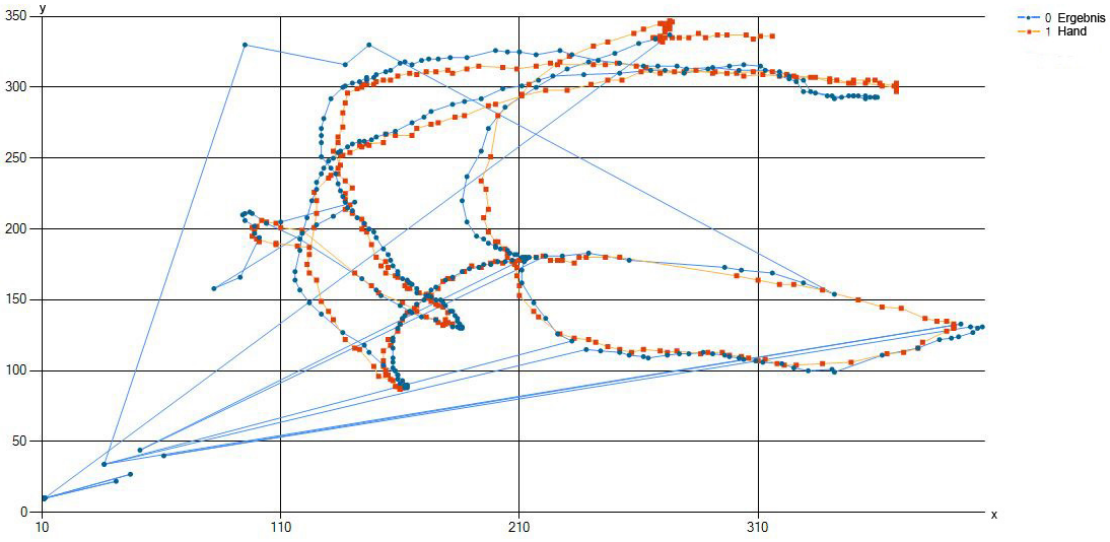


Abbildung 7.11: Bester Test für Testsequenz 2. Ergebnis des Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 0, CS 95, KLT 0, NCC 5.

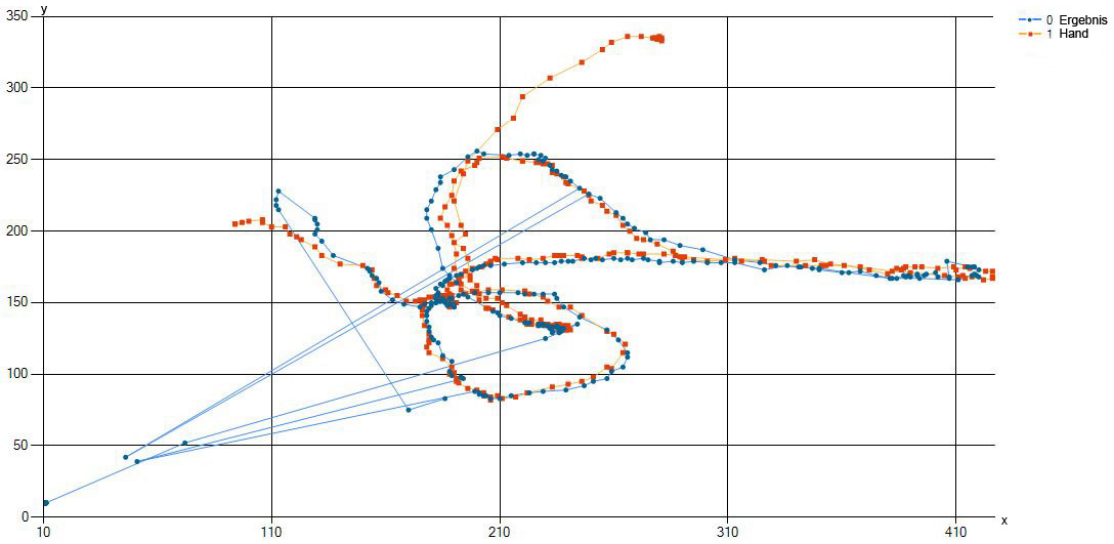


Abbildung 7.12: Bester Test für Testsequenz 3. Ergebnis des Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 50, CS 25, KLT 20, NCC 5.

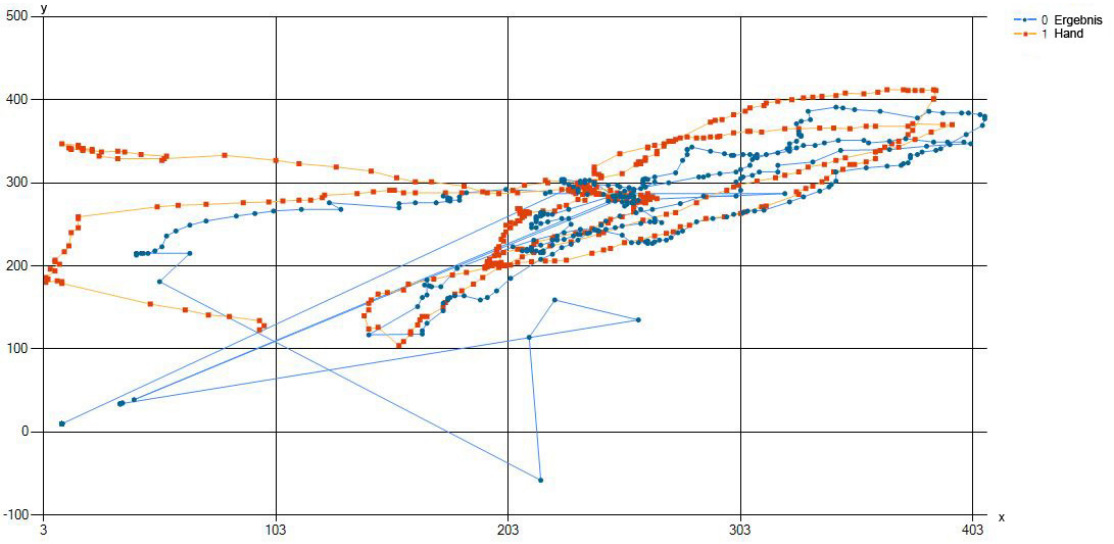


Abbildung 7.13: Bester Test für Testsequenz 4. Ergebnis des Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 95, CS 5, KLT 0, NCC 0.

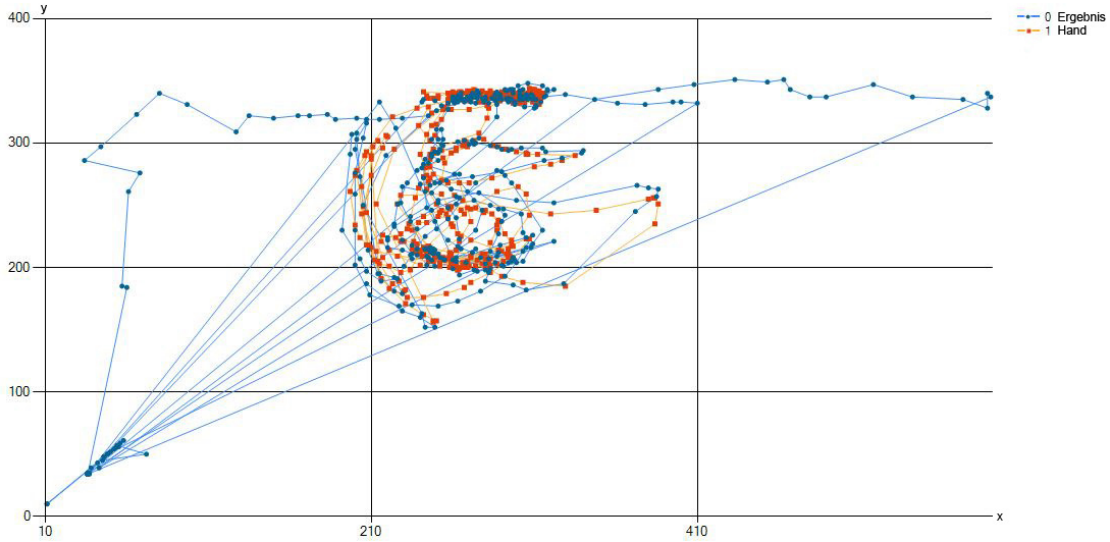


Abbildung 7.14: Bester Test für Testsequenz 5. Ergebnis des Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 25, CS 20, KLT 10, NCC 45.

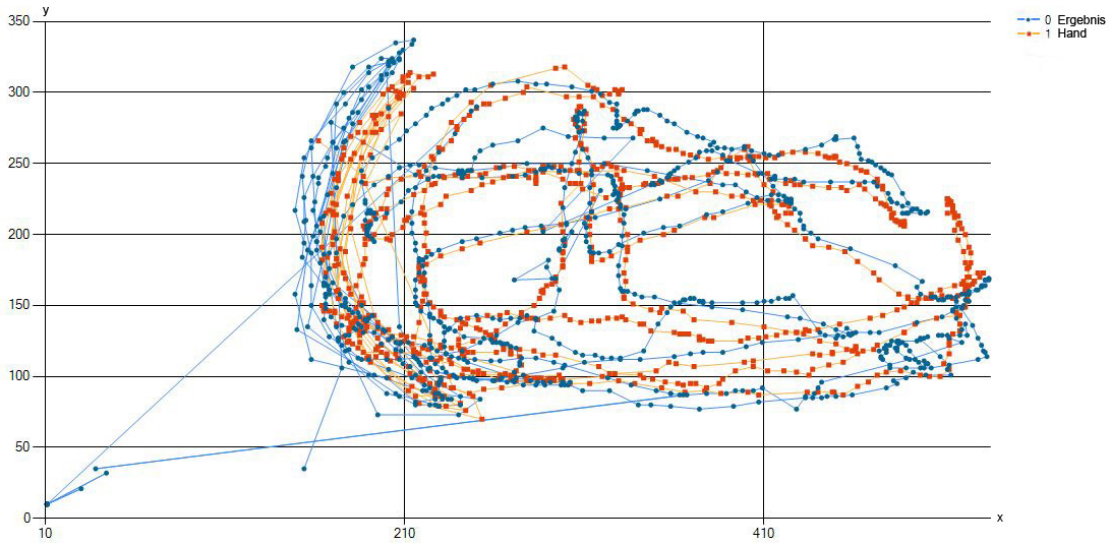


Abbildung 7.15: Bester Test für Testsequenz 6. Ergebnis der Tracker in blau, Position der Hand in rot. Confidence Values: Kinect 55, CS 5, KLT 5, NCC 35.

Kapitel 8

Zusammenfassung

8.1 Ergebnisse

Es wurde gezeigt, dass die aus der Bachelorarbeit entstandene Anwendung eine mögliche Lösung der Problemstellung darstellt. Sie fügt sich in das bestehende System ein und ist flexibel auf Anwendungsfälle anpassbar. Das modulare System macht sie leicht erweiterbar und wartungsfreundlich. Die verschiedenen Tracker nutzen ihre Stärken aus und kompensieren ihre Schwächen. Bestätigt wird dies durch die in der Analyse gefundenen Confidence Values der einzelnen Testsequenzen. Für jede Situation führten andere Konfigurationen zu den besten Ergebnissen.

8.2 Ausblick

In die Anwendung könnten weitere Tracker implementiert werden um noch umfassender alle Situationen abdecken zu können. Weiterhin wäre ein Modul denkbar das dynamisch die Global Confidence Values findet. Aber auch die implementierten Tracker haben noch viel Potential. In den CS Tracker könnte noch die Helligkeit eingebracht werden, der KLT Tracker könnte zu einem FoF Tracker [KT04] erweitert werden und der NCC Tracker könnte ein ausgefeilteres System zum aktualisieren des Suchbildes erhalten. Doch zählt schlussendlich nur das gemeinsame Ergebnis aller Tracker. So sehe ich für die Zukunft das größte Potential in der Kombination von vielen spezialisierten Trackern.

Literaturverzeichnis

[BH01] Briechle, K. ; Hanebeck, U.D.: Template matching using fast normalized cross correlation. In: *Proceedings of SPIE* Bd. 4387 SPIE, 2001, S. 95–102

[Bou02] Bouguet, Jean-Yves: *Pyramidal Implementation of the Lucas Kanade Feature Tracker*. 2002

[Bra98] Bradski, Gary R.: Computer Vision Face Tracking For Use in a Perceptual User Interface. In: *Intel Technology Journal* (1998), Nr. Q2, 1-15. `citeseer.csail.mit.edu/bradski98computer.html`

[Fog11] Fogelton, Bc A.: Real-time hand tracking using Flocks of features. In: *Proceedings of CESCG* (2011)

[Int] Intel: *Open Source Computer Vision Library (OpenCV)*. `http://www.intel.com/technology/computing/opencv/`

[KE12] Khoshelham, Kourosh ; Elberink, Sander O.: Accuracy and resolution of kinect depth data for indoor mapping applications. In: *Sensors* 12 (2012), Nr. 2, S. 1437–1454

[KMM09] Kalal, Zdenek ; Matas, Jiri ; Mikolajczyk, Krystian: Online learning of robust object detectors during unstable tracking. In: *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* IEEE, 2009, S. 1417–1424

[KT04] Kolsch, Mathias ; Turk, Matthew: Fast 2d hand tracking with flocks of features and multi-cue integration. In: *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on* IEEE, 2004, S. 158–158

[ST94] Shi, Jianbo ; Tomasi, Carlo: Good Features to Track. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*. Seattle, 6 1994, 593-600

[Ste04] STENGER, B.: *Model-Based Hand Tracking Using A Hierarchical Bayesian Filter*. Cambridge, UK, University of Cambridge, Diss., 3 2004. http://mi.eng.cam.ac.uk/~bdrs2/papers/stenger04_thesis.pdf

[SWC10] STENGER, Björn ; WOODLEY, Thomas ; CIPOLLA, Roberto: A Vision-Based Remote Control. Version: 2010. http://dx.doi.org/10.1007/978-3-642-12848-6_9. In: CIPOLLA, Roberto (Hrsg.) ; BATTIATO, Sebastiano (Hrsg.) ; FARINELLA, GiovanniMaria (Hrsg.): *Computer Vision* Bd. 285. Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–12847–9, 233-262

[Vie08] VIERJAHN, Tom: Handerkennung und -tracking im Kamerabild, 2008

[WTH02] WANG, Liang ; TAN, Tieniu ; HU, Weiming: Face tracking using motion-guided dynamic template matching. In: *Fifth Asian Conference on Computer Vision*, 2002