OST
Ostschweizer
Fachhochschule

FreeCursor MediaPipe API

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2021

Autor(en):	Lukas Schiltknecht,
	Nathanael Gall
Betreuer:	Oliver Augenstein
Projektpartner:	OST – Ostschweizer Fachhochschule
Experte:	Reto Bättig
Gegenleser:	Frank Koch

1. Abstract

Problem	Touch-Oberflächen sind für die meisten normalen Geräte üblich, ausser für Beamer. Im Moment werden Präsentationen jeweils mit einem Laserpointer bedient. Ein projiziertes Bild hat keine Möglichkeit, die Oberfläche auf eine Berührung abzufragen. Mit zusätzlichen Hilfsmitteln wäre es technisch machbar, die Koordinaten einer Berührung zu erkennen. Dies würde Kosten verursachen und möglicherweise sogar zusätzliche Infrastruktur benötigen.
Ziel	Einerseits soll geprüft werden, inwieweit die MediaPipe Library verwendet werden kann, um Beamer ohne zusätzliche Hardware durch Software und mit der eingebauten Webcam in Touchpads zu verwandeln. Und andrerseits ist es Ziel, eine Beispiel-Software zu entwickeln, die es einem ermöglicht freihändig zu präsentieren und dabei auf dem Bild des Projektors/Bildschirms direkt den Cursor zu steuern
Methode/-Vorgehen	Es geht darum, die Grenzen der MediaPipe API zu ertesten, um herauszufinden, ob dieses Tool als mögliches Mittel dienen kann, eine Hand vor dem Bild eines Projektors auf dem Videostream einer eingebauten Webcam zu erkennen. Sofern eine Hand erkannt wird, kann eine digitale Bildtransformation zur Erkennung der Bildschirmkoordinaten der Hand gemacht werden und dann die Gestik ausgelesen werden, welche zur Steuerung des Mauscursors verwendet wird.
Wesentliche Ergebnisse	In der ersten Projektphase wurden die Grenzen der MediaPipe API manuell unter vorher festgelegten Bedingungen ertestet. Diese Bedingungen beinhalteten sowohl Infrastruktur als auch projizierten Bildinhalt. Daraus ergab sich, dass Tageslicht einen grossen Einfluss auf die Stabilität der Funktionsweise der MediaPipe API hat. Weiter ergab sich daraus, dass ein projiziertes Bild, welches stark gesättigte Farbanteile hat, ein Erkennen von Händen mittels der MediaPipe API verunmöglicht. Aus diesen Ergebnissen resultierende Tests haben aufgezeigt, dass ein Lösen dieser Probleme mittels konventionellem Vorgehen in der vorgegebenen Zeit nicht möglich war. Daraufhin wurde die Idee einen Beamer zu steuern verworfen und ein gestengesteuerter Touchscreen-Maus-Treiber für den Nontouch-Bildschirm entwickelt.

Empfehlungen

Für weitere Projekte, welche dieselbe Aufgabe umsetzen möchten empfiehlt es sich, die Wahl der Trainingsdaten so zu treffen, dass die Test-Situation ungefähr den in dieser Arbeit beschriebenen Infrastruktur-Szenarien und Hintergrundbild-Dimensionen entspricht. Unsere Lösung zeigt, dass es grundsätzlich möglich ist, einen Beamer softwarebasiert in ein Touchpad zu verwandeln. Damit das Verfahren aber nicht nur für externe Bildschirme sondern auch konventionelle Beamer funktioniert, ist es erforderlich, dass das neuronale Netzwerk, welches zur Erkennung der Gesten verwendet wird, auch mit entsprechenden Bildern trainiert wird.

Schlüsselwörter

MediaPipe API, Beamer-Bedienung, Computersteuerung via Webcam

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	21.10.2021
1.1	Finalisierung	Lukas Schiltknecht	22.01.2022

Inhaltsverzeichnis

2. Aufgabenstellung im Original mit Unterschrift

In dieser Arbeit soll eine Aufgabenstellung aus einer früheren Bachelorarbeit noch einmal in abgewandelter Form aufgegriffen werden, um im Zuge des Projekts die Mediapipe-API kennen zu lernen. Das Kennenlernen dieser Bibliothek anhand einer interessanten Aufgabenstellung ist dabei eines der Hauptziele des Projekts. Ich bin offen dafür, die Bibliothek in einer anderen Anwendung einzusetzen, sofern wir uns vor Beginn der Arbeit auf die konkrete Aufgabenstellung einigen.

Falls kein eigener Projektvorschlag zur Verwendung der Mediapipe-API eingebracht wird, ist es Ziel der Arbeit, mit der in einem Laptop eingebauten Webcam das vom Laptop projizierte Beamerbild zusammen mit dem Präsentator so zu filmen, dass der Beamer vom Präsentator als Touchpad genutzt werden kann. Dazu muss vorgängig die Kamera auf das Beamerbild kalibriert werden. Hierzu kann auf die Konzepte der früheren Bachelorarbeit zurückgegriffen werden. Sie müssen allerdings je nach Wahl der Programmiersprache allenfalls neu implementiert werden.

Der erste Teil der Aufgabenstellung besteht dann daraus, die Pixelkoordinaten der Position der Hand oder eines Fingers auf dem Beamer zu identifizieren und den Wunsch des Präsentators, einen Button anzuklicken zu erkennen. Das zuverlässige Erkennen des Klicks und des darunter liegenden Buttons ist hierbei zentral. Wichtig ist auch, dass die Software nicht fälschlicherweise Klicks identifiziert. Diese Funktionalität soll unter realen Arbeitsverhältnissen (bei verschiedene Präsentatoren, Laptops (Windows), Beamer, Licht- und Raumverhältnissen) funktionieren.

Der zweite Teil der Arbeit soll dann die entsprechenden Klicks an das Betriebssystem übergeben, so dass diese Klicks tatsächlich ausgeführt werden.

Die Mindestfunktionalität am Ende der Arbeit sollte sein, dass man mit Hilfe der entstandenen Software den Laptop über die Beamerprojektion zuverlässig bedienen kann. Die Integration einer "Stiftfunktionalität" in die Software, wie sie in der vorangegangenen Arbeit entwickelt wurde, wäre schön, ist aber nicht Teil der Mindestanforderungen. Dieser Integrationsschritt wäre allenfalls Teil einer Folgearbeit.

Die angestrebte Softwarequalität soll so hoch sein, dass das Projekt nach Abschluss der Arbeit als OpenSource-Projekt weitergeführt werden kann.

Unterschrift: _____

3. Technischer Bericht

3.1 Einleitung

Nutzen und Idee

Das Projekt kam zustande auf Initiative von Herrn Professor Oliver Augenstein, der vor einigen Jahren schon einmal eine ähnliche Problemstellung als Bachelor Arbeit ausgeschrieben hatte, welche die Thematik zu einem grossen Teil löste, allerdings mit gewissen Einschränkungen, welche in dieser Arbeit mit einer neuen Herangehensweise zumindest zum Teil verringert werden sollen. Die Problemstellung beinhaltet das Bedienen eines Computersystems durch Handgestik vor einem projizierten Bild. Dies ist wichtig für das Unterrichten ohne Wandtafel, den Austausch von Ideen ohne Whiteboard und hilft entsprechend in jeder Situation, in welcher sowohl präsentiert als auch eine Computeranwendung bedient wird.

Hauptziel

Das Hauptziel der Arbeit beinhaltet die Führung des Cursors mittels Handführung und Gestik auf dem Bild eines Videoprojektors. Dies beinhaltet die Positionierung des Cursors und das Auslösen eines Klicks durch eine Handbewegung.

Zusatzziel

Das Zeichnen von Skizzen auf dem Bild eines Videoprojektors durchführen und das Führen des Cursors mit Hilfe von Handbewegungen und Gestiken.

Umgebung oder Infrastruktur

Die exakte Beschreibung der Umgebungsfaktoren unter welchen die Software getestet wird, ist im Kapitel der Anforderungsspezifikationen genau erläutert. Daher werden in diesem Kapitel die Anforderungen an die Umgebung und Infrastruktur stark vereinfacht. Grundsätzlich soll die Software in einer normalen Präsentationsumgebung mit branchenüblichen Mitteln und durchschnittlichen Lichtverhältnissen angemessene Ergebnisse liefern.

Methode / Vorgehen

Diese Arbeit geschieht iterativ. In den ersten Wochen werden nur Experimente mit den einzelnen externen Softwarekomponenten, namentlich MediaPipe API und OpenCV, gemacht, um die Machbarkeit und die entsprechenden Grenzen des momentan realistischen Ergebnisses zu eruieren. Danach folgt das Erstellen eines ersten Softwareentwurfs, der dann iterativ durch Testen in verschiedenen Umgebungen mit unterschiedlichen Mitteln optimiert wird. Am Ende wird ermittelt, wo am meisten Optimierungspotential besteht und dann schrittweise optimiert bis zu einem Zeitpunkt, wo keine neuen Funktionen mehr hinzugefügt werden (Featurefreeze).

3.2 Ausgangslage

3.2.1 Bisherige Vorgehensweisen

Maus oder Touchpad

Der Präsentierende bedient während der Präsentation das Gerät mit der Maus oder dem Touchpad. Diese Gräte sind meist vor dem Präsentierenden platziert und jeweils nicht dort, wo das Bild hin projiziert wird.

Nachteile

- Die Aufmerksamkeit des Publikums geht verloren.
- Die Maus eignet sich schlecht, um zu zeichnen oder zu schreiben.
- Die Bedienung kann schwerfällig sein.

Bisherige Bachelorarbeit

Laut den Aussagen von Herrn Professor Augenstein ist die Funktionalität der bestehenden Software sehr gut, mit dem einzigen Nachteil, dass der Benutzer über das System die Helligkeitskorrektur ausschalten muss.

Nachteil

- Der Benutzer muss die Helligkeitskorrektur über das System ausschalten.
- Basiert auf konventionellem C# und ist daher vermutlich nicht plattformunabhängig.

3.2.2 Beurteilung des Potentials

Kriterien der Optimierung

Anhand der bestehenden Lösungsansätze kann gesagt werden, dass Verbesserungspotential besteht in den folgenden Bereichen:

- Robustheit bezüglich der Lichtverhältnisse
- Anwenderfreundlichkeit
- Plattformunabhängigkeit
- Zuverlässigkeit der Bedienung

Diskussion

Der neue Lösungsansatz soll hauptsächlich auf Experimentieren und dem Austesten der externen Softwarekomponenten wie der MediaPipe API und OpenCV beruhen. Dazu werden verschiedene Testszenarien festgelegt, welche detailliert im Kapitel Anforderungsspezifikation beschrieben sind. Anhand dieser Testszenarien wird festgelegt, wie die Robustheit der Software verbessert und die Anwenderfreundlichkeit gewährleistet werden kann. Da als Technologie Python verwendet wird, was als plattformunabhängige Sprache gilt, sollte dieser Punkt schon von vorneherein kein Thema sein.

3.3 Lösungsansatz mit MediaPipe-API

Unsetzung mit MediaPipe

Unsere Lösung basiert auf dem Einsatz der MediaPipe-API von Google. Dabei wird hauptsächlich ein empirischer Ansatz verfolgt, der durch gezielte Tests iterativ verbessert werden kann. Als eine von Google zur Verfügung gestellte Library ist die Dokumentation über die API selbst sehr gut. Die Teile, welche als «künstliche Intelligenz» bezeichnet werden können, sind nur sehr schwer durch die Dokumentation nachzuvollziehen. Das liegt allerdings daran, dass auf das Neuronale-Netzwerk selbst durch den Nutzer mit den nach aussen sichtbaren Mitteln der Library nicht Einfluss genommen werden kann.

Herausforderung

Da die MediaPipe-API als Neuronales-Netzwerk in dem Sinne für das Team eine Blackbox ist, werden als erstes die Grenzen der Erkennungsalgorithmen eruiert. Sobald der Schritt der Erkennung bis an die Grenzen getestet wurde, werden Ideen gesammelt und schrittweise getestet um die Funktionsweise zu optimieren.

Fokus der Arbeit

Die Ergebnisse der Arbeit beinhalten Software, Software-Dokumentation und eine Dokumentation der Testergebnisse im Zusammenhang der Erkennung von Gestiken mit der API. Diese Arbeit dient sowohl dazu das Haupt- und Nebenziel zu erreichen, welches mit dem Bedienen eines Computersystems per Handgestik gegeben ist, als auch dazu die Funktionsweise der API auszutesten.

3.4 Ergebnisse der Arbeit

Zusammenfassung

In dieser Arbeit wurde nach ausführlichem Testen gezeigt, dass das Bedienen einer einfachen Whiteboardanwendung durch Gestik funktioniert. Während die Mediapipe alle dafür notwendigen Funktionalitäten vorweist, ist sie jedoch out of the box nicht geeignet, um zuverlässig Hände zu erkennen, welche von einem Beamer angestrahlt werden.

Ausblick

Sollte dieses Projekt wieder aufgegriffen werden, empfehlen die Autoren folgende Verbesserungen:

Das für die Usability am relevanteste Problem ist, dass diese Lösung nur mit Bildschirmen funktioniert und nicht mit einem Leinwand und Beamer Setup. Während von unseren Ansätzen mehrere Potential haben, ist der Selfie Segmentation Ansatz der Vielversprechendste. Eine weitere mögliche Idee, wäre es, die Mediapipe zu ersetzen durch eine andere Library, welche Handerkennung anbietet.

Weitere Verbesserungsansätze wären eine bezüglich Lichtverhältnissen robustere Methode, um den Bildschirm zu erkennen und das Verwenden von Deep Learning um die Vielfalt und Dynamik der möglichen Gesten zu erhöhen.

4. Projektdokumentation

4.1 Vision

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	21.10.2021

Projektform

Dieses Projekt unterscheidet sich massgeblich von Projekten, in denen die Arbeitsvorgänge und der Umfang bekannt sind. Da ein grosser Teil des Projekts mit Experimenten zusammenhängt, ist eine klassische Form der Projektplanung nicht zielführend.

Zeitumfang

Da das Projekt aufgrund einer Verletzung eines Teammitgliedes später starten musste sind die zeitlichen Vorgaben wie folgt:

Teammitglied	Stunden/Woche	Total Stunden
Nathanael Gall	18	252
Lukas Schiltknecht	18	252
Total		504

Tabelle 1: Stundenbudget

Dies entspricht den gleichen Bedingungen wie einer Bachelorarbeit, die nicht durch Krankheit in Mitleidenschaft gezogen wird. Allerdings sind die Abgabetermine um 2 Wochen nach hinten geschoben worden.

4.1.1 Arbeitsauftrag

Versuchsphase

Während der Versuchsphase war der Auftrag, sich in das Thema Mediapipe zu vertiefen und die API kennen zu lernen, um zu eruieren, ob die Aufgabe mit diesem Tool zufriedenstellend gelöst werden kann und unter welchen Umständen die Software den Zweck erfüllen soll.

Software-Entwicklung

In der Phase der Softwareentwicklung galt es die bestehenden Ansätze entsprechend den an der OST gelehrten Grundsätzen zu Software-Entwicklung auf einen Stand zu bringen, damit der erstellte Code als Ausgangslage dienen kann, um eine Weiterentwicklung dieses Tools als mögliche Folgearbeit zu bewerkstelligen.

4.2 Systemanalyse

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	[Student]	01.01.2020

Erläuterung

Dieses Kapitel umfasst mehr als nur die in Software-Projekten üblichen Ergebnisse und Darstellungen der Systemumgebung, sondern auch die Ergebnisse *der Versuchsphase,* welche hauptsächlich im Kapitel über die Mediapipe API abgehandelt werden.

4.2.1 Erläuterung des Systems

System-Diagramm

Um den groben Zusammenhang des Systems zu verstehen, ist im Folgenden eine Übersicht in Form eines Systemdiagramms sichtbar. Aus dieser Abbildung wird ersichtlich, dass ein grosser Anteil der Funktionalität des Programms abhängig ist vom Einsatz und der Qualität der damit verbundenen Bibliotheken und Komponenten.

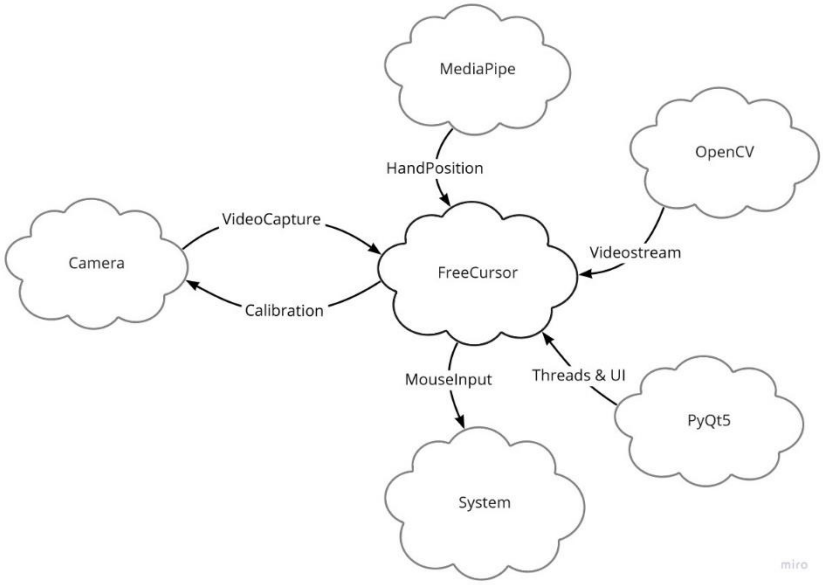


Abbildung 1: System-Diagramm

MediaPipe

Von Google zur Verfügung gestellte Software zur Bilderkennung auf Videostreams.

FreeCursor

Vom Team dieser Arbeit zur Verfügung gestellte Software zum Bedienen eines PCs via Webcam.

System	Betriebssystem auf welchem die FreeCursor Software ausgeführt wird.
OpenCV	Python Library zum Öffnen und auswerten von Video-Streams über die interne Web-Kamera.
Camera	Systeminterne oder externe Videokamera.
PyQt5	Framework zum Erstellen von UI-Komponenten.

4.2.2 Infrastruktur-Szenarien

Bestandteile der Infrastruktur

Die Infrastruktur setzt sich zusammen aus folgenden Komponenten:

- Beamer
- Laptop
- Webcam
- Raum
- Lichtverhältnisse
- Störfaktoren
- Erschwerende Umstände

Dementsprechend sind die Infrastruktur-Szenarien so ausgelegt, dass jeweils nach Möglichkeit nur eine zusätzliche Komponente auswechselbar, respektive frei wählbar wird.

Szenario (0)	Als Szenario Null wird der Laborraum betrachtet, in welchem die Software entwickelt wird. Entsprechend sind alle frei wählbaren Parameter abgestimmt auf diesen Raum und daher sollte sie dort am zuverlässigsten funktionieren.
Szenario (1)	Beim Szenario Eins wird der Raum gewechselt. Der Projektor, der Laptop und die Webcam bleiben gleich.
Szenario (2)	Beim Szenario Zwei ändern sich Raum, Projektor, Laptop und Webcam. Die Lichtverhältnisse und die Störfaktoren, sowie die erschwerenden Umstände bleiben so konstant wie möglich.
Szenario (3)	Beim Szenario Drei ändern sich die Lichtverhältnisse. Die Störfaktoren, sowie die erschwerenden Umstände bleiben so konstant wie möglich.
Szenario (4)	Beim Szenario Vier kommen Störfaktoren dazu. Zum Beispiel eine unebene Wand, ein Scheinwerfer, der das Bild zeitweise streift, oder Ähnliches.
Szenario (5)	Beim Szenario Fünf wird ein Teil des projizierten Bildes abgeschnitten. Dieses Szenario wurde von Anfang an als nicht ohne Weiteres lösbar eingestuft.

4.2.3 Kamera Kalibration

Charuco

Zur Kalibration der Kameraverzerrung wurde ein Verfahren eingesetzt, welches die Vorteile der Schachbrettmuster-Erkennung und der Aruco-Indize-Erkennung kombiniert. Dieses Verfahren hat gegenüber einem Schachbrettmuster den Vorteil, dass es auch funktioniert, wenn nur ein relativ kleiner Anteil an Bildmustern erkannt wird.

Verfahren

Erst wird ein Referenzbild *Charucoboard* erstellt, entsprechend dem Charuco-Dictionary und dann startet ein Loop zur Erkennung der Indize. Der Anteil erkannter Indize wird auf dem Bildschirm als Balken dargestellt, der manuell optimiert werden kann, mithilfe der Tasten 'W' und 'E'. Diesen Vorgang zu automatisieren, hat das Entwicklerteam ausgiebig versucht, dies wurde aber von der systeminternen Helligkeitskorrektur jeweils vereitelt. Sobald der Benutzer die Taste 'S' drückt, startet das Auslesen der Indize. Nach dem Auslesen, startet der Vorgang zur Berechnung der Kameramatrix, welche dann als Python-Dictionary zurückgegeben wird.

4.2.4 Bildschirmerkennung

Verfahrenswahl

Diese Aufgabe wurde von der letzten Gruppe mithilfe eines KNN-Algorithmus gelöst, was den Vorteil hatte, dass es anscheinend robust war. Die Möglichkeit es mit KNN zu lösen, wurde in dieser Arbeit bewusst nicht gewählt, da das mit dem KNN erkannte Viereck nicht direkt auf geometrische Konsistenzbedingungen überprüft werden kann und eine affine Transformation mit den ermittelten Koordinaten anhand einer Transformationsmatrix zu mehreren Problemen geführt hätte. Daher haben wir uns für den Algorithmus von OpenCV entschieden, der mittels eines korrigierten Bildes die Konturen des Bildschirms ermittelt, dann auf geometrische Konsistenz überprüft und ausgibt.

Ausgangslage

Als Ausgangslage diente ein OpenCV-Tutorial [1], welches auf einem einfachen Bild einer weiss projizierten Fläche beruht. Dieses Verfahren hat eigentlich nur in einem Szenario funktioniert, in welchem *keine anderen viereckigen oder sonst wie konturstarken Elemente sich auf dem Bild befanden.* Unglücklicherweise wäre dieser Nachteil, der auch bei einem KNN-Verfahren bestanden hätte, nicht tragbar.

Differenzbilder

Weil die Vorgehensweise in dem OpenCV-Tutorial für unsere Zwecke nicht genügte, wurde das Verfahren angepasst und mit zwei Bildern gearbeitet. Das erste Bild wird eingefangen von einem weissen Bildschirm und das zweite Bild wird eingefangen von einem Bildschirm, der eine 16px breite, schwarze Kontur rund um den Bildbereich abbildet. Daraus wird die Differenz errechnet und dadurch ist *theoretisch* nur noch die Kontur sichtbar. Praktisch funktioniert dieses Verfahren

gut, allerdings ist es meist nicht ein Bild, auf welchem lediglich die Konturen sichtbar sind, sondern auch noch ein mehr oder weniger starkes Bildrauschen.

Gradationskurve

Mit einem Lookup-Table wird über das resultierende Differenzbild eine sehr starke, harte Gradation gerechnet, welche lediglich die Konturen noch als Tonwerte abbilden sollte. In der Praxis funktioniert dieses Verfahren angemessen, auch wenn im resultierenden Bild nicht immer nur die Konturen sichtbar sind.

Konturerkennung

Die Konturerkennung des Bildschirms wird im Moment von der OpenCV-Funktion `cv2.findContours()` allerdings wäre es entsprechend besser, wenn man die Konturen als Linien berechnen und daraus die Eckpunkte errechnen würde. Da noch dringendere Pendenzen bestanden, wurde dies aber als niedrige Priorität angesehen.

4.2.5 Mediapipe-API

Allgemein

Die Mediapipe [2] ist eine Machine Learning Lösung zur Verarbeitung von gestreamten Videodaten. Sie beinhaltet diverse Subprojekte, welche fertige Funktionen anbieten, wie Objecttracing, Gesichtserkennung, Motiontracking und diverse andere Schnittstellen, um menschlich lesbare Informationen aus einem Videofeed zu gewinnen.

Version

Mediapipe ist momentan in der Alphaphase und in dieser Arbeit wurde die Version 0.8.7.3 verwendet.

Mediapipe Hands

Für die Zwecke dieser Arbeit wurde Mediapipe Hands benutzt. Diese Lösung bietet die Funktionalität an, um aus rohem Bildmaterial die Position von darin vorkommenden Händen zu bestimmen.

Um dies zu erreichen, erkennt die Software zuerst Handflächen und bestimmt in einem zweiten Schritt dann die relative Position der Finger zu dieser Handfläche. Der Ausgabewert sind dann X-, Y- und geschätzte Z-Koordinaten einzelner Punkte in der Hand, relativ zum ganzen Inputbild. Der genaue Ablauf dieses Vorgangs wurde im Rahmen dieser Arbeit jedoch als Blackbox behandelt. [3]

4.2.5.1 Hintergrundbild-Dimensionen

Hintergrundbilder

Das Bedienen des Laptops mit der Software sollte auf verschiedenen projizierten Hintergrundbildern möglich sein. Einige hier aufgeführte Hintergrundbilder dienen dazu, die Grenzen der Software aufzuzeigen:

a. Weiss
b. Schwarz
c. Grundfarben volle Sättigung (#ff0000, #00ff00, etc.)
d. Grundfarben auf ¾ Sättigung (#bf3f3f, #3fbf3f, etc.)
e. Grafiken und Buttons
f. Schrift und Logo (Präsentation)
g. Foto
h. Foto mit Händen (sollte fehlschlagen)
i. Video

Merke: *Die Farben wurden in Hexadezimalwerten für den RGB-Farbraum angegeben.*

4.2.5.2 Testszenarien

Motivation

Mit dem Ziel, dass das resultierende Programm in Szenario 2 laufen sollte, musste zuerst ein Bild gemacht werden, wie diverse Faktoren, die Performance der Mediapipe beeinflussen.

Wahl der Tests

Nach kurzem Herumspielen mit der API wurde schnell klar, dass kleine Veränderungen im Winkel der Hände, Wolken, welche vor der Sonne durchziehen und Tageszeit starken Einfluss auf die Testergebnisse haben würden. Dies machte es sehr schwierig, rekonstruierbare Szenarien und Tests zu schreiben. Deshalb wurde folgender Testablauf definiert:

1. Die benutzte Hardware (Beamer, Raum, Kamera und Leinwand) wurden festgehalten.

2. Die Lichtsituation, während dem Test, wurde mit einer Mobiltelefonkamera fotografiert.

3. Eine Powerpointpräsentation mit den gegebenen Hintergrunddimensionen wurde im Vollbildmodus auf die Leinwand projiziert. Dies wurde von der Kamera gefilmt und der Videofeed davon in Echtzeit in die Mediapipe eingespiesen.

4. Die testende Person bewegt ihre Hand vor jedem dieser Hintergründe und sieht an direktem Feedback, wann die Hand vor dem Hintergrund dort erkannt wird, wo sie sich tatsächlich befindet.

5. Da es nicht möglich war, vor jedem Hintergrund und jedem Testszenario die exakt selbe Bewegungen zu machen, konnte der Erfolg der API nicht danach gemessen werden, auf wie vielen Frames die Hand richtig erkannt wurde. Darum wurde jeweils vom Tester auf Deutsch beschrieben, wie gut die Erkennung funktioniert hat.
   Dies Passiert auf einer Skala von «Die Hand wurde nicht erkannt» bis zu «Funktioniert einwandfrei», was heisst, dass die Hand zuverlässig genau dort eingezeichnet war, wo sie sich befand.

6. Dann wurde, falls das Erkennen der Leinwand funktioniert hat, noch ein Frame ausgelesen und dokumentiert, welches der Mediapipe eingespiesen wurde. Dies sollte dazu dienen, dass man sieht, wie gut die Hand von Auge aus erkennbar sei.

Testszenarien

Diese Tests wurden in drei Räumen mit jeweils vier Lichtverhältnissen durchgeführt: Einmal ohne Licht, einmal mit Tageslicht, einmal mit Zimmerbeleuchtung ohne Tageslicht und einmal mit beidem.

Abbildung 2: Raum 1 | Wohnzimmer

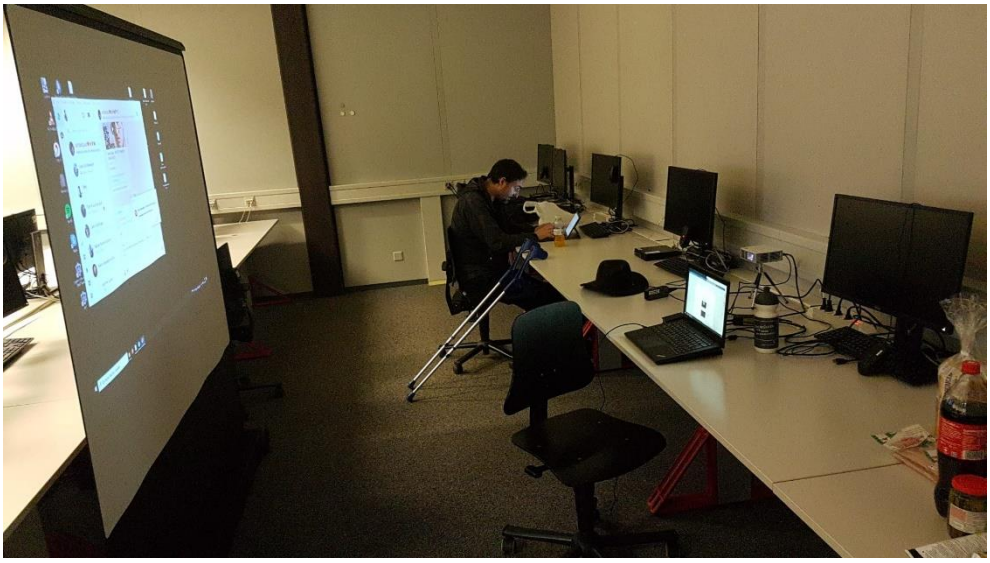


Abbildung 3: Raum 2 | Laborarbeitsplätze

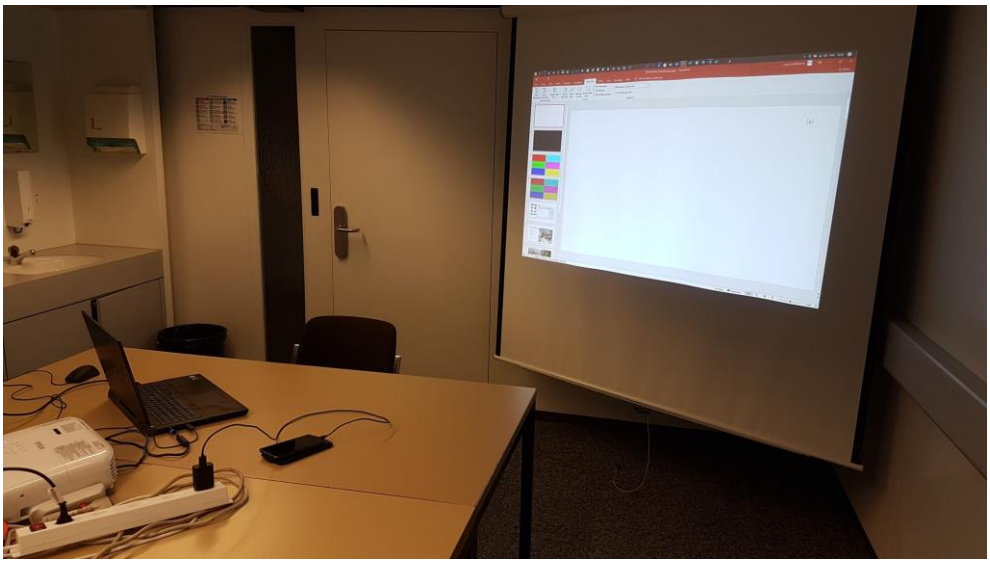


Abbildung 4: Raum 3 | Sitzungszimmer

Bei den drei Räumen handelte es sich einmal um ein einfaches Wohnzimmer mit einem Taschenbeamer, welcher die tapezierte Wand anstrahlt, einmal um die Laborarbeitsplätze, mit demselben portablen Beamer und einer portablen Leinwand, einmal um ein Sitzungszimmer der OST mit dem dort vorhandenen Setup.

Emission von Tests

Die folgenden Tests wurden ausgelassen, da die Hand vor keinem der Hintergründe ausser Weiss und Schwarz überhaupt erkannt wurde:

1. Laborarbeitsplätze: Alle Lichtsituationen
2. Sitzungszimmer: Alle Situationen ohne Tageslicht
3. Im Wohnzimmer wurden alle Tests mit Tageslicht gar nicht erst durchgeführt, weil die Wand, an welche projiziert wurde, von Tageslicht kaum erreicht werden konnte.

Nachtests

Um einzelne Hypothesen zu testen, welche durch das Auswerten der Testresultate entstanden sind, wurden noch einzelne Experimente durchgeführt:

- **a.** Auswechseln der Leinwand durch ein rotes Tuch
- **b.** Auswechseln des Beamers im Sitzungszimmer durch einen schwächeren Beamer
- **c.** Veränderung der Helligkeit eines Beamers
- **d.** Anziehen von grauen Handschuhen
- **e.** Aufstellen eines Scheinwerfers, welcher auf die Leinwand leuchtet

4.2.5.3 Ergebnisse

Allgemein

Die Durchführung der obig genannten Tests hat Aufschluss gegeben darüber, wie diverse Faktoren das erfolgreiche Erkennen von Händen in der Mediapipe beeinflussen.

Lichtsituation

Die Lichtsituation hat einen sehr starken Einfluss auf die Performance der Mediapipe. Je heller der Raum ist, in dem das Programm ausgeführt wird, desto besser werden Hände erkannt. Dieser Effekt zeichnet sich nur sehr schwach ab auf komplett schwarzem oder komplett weissem Hintergrund. Auf allen anderen Hintergründen ist er recht stark ausgeprägt. Dies geht so weit, dass wir diverse Tests in dunkleren Umgebungen komplett weglassen mussten. Dazu kommt, dass natürliches Licht besser zu sein scheint als die Beleuchtung mit Lampen.

Um diese Annahme zu testen, wurde In der Wohnzimmersituation ein Scheinwerfer auf die Hand gerichtet. Die Handerkennung funktionierte mit dem Aufdrehen der Helligkeit des Scheinwerfers immer besser, bis zu dem Punkt, an dem die Hand zwar immer erkannt wurde, das projizierte Bild im Hintergrund jedoch kaum mehr wahrzunehmen war.

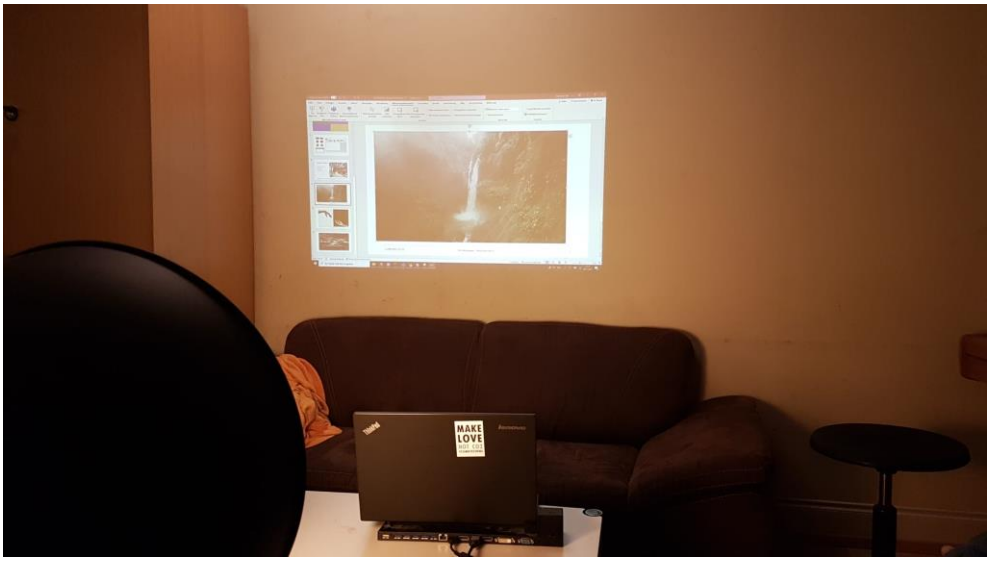


Abbildung 5: Situation mit Scheinwerfer

Beamer

Einer der Hauptunterschiede zwischen dem Wohnzimmerszenario und dem Sitzungszimmer war, abgesehen von der Beleuchtung, das Verwenden eines portablen und viel schwächeren Beamers. Die These wurde aufgestellt, dass sich die Helligkeit des Beamers umgekehrt verhält zu der Helligkeit der Umgebung. Um dies zu testen, wurde im Sitzungszimmer einerseits die Helligkeit des Beamers heruntergeschraubt und andererseits der portable Beamer aus dem Wohnzimmerszenario verwendet.

Beide dieser Experimente führten zu merkbar besserer Performance bei allen Hintergründen, ausser denen, welche komplett weiss oder komplett schwarz waren.

Leinwand

Ein weiterer Unterschied zwischen Wohnzimmer und Sitzungszimmer war die Leinwand. Bevor der Test durchgeführt wurde, gab es die Annahme, dass sich die unreine vergilbte Tapete negativ auf den Test auswirken würde. Da der Test im Wohnzimmer besser ausfiel, als der Test im Sitzungszimmer musste die Leinwand als Variable isoliert werden. Es wurde eine orangefarbene Wolldecke an die Wohnzimmerwand gehängt und dies hatte auf das Erkennen der Hände keinen messbaren Einfluss.

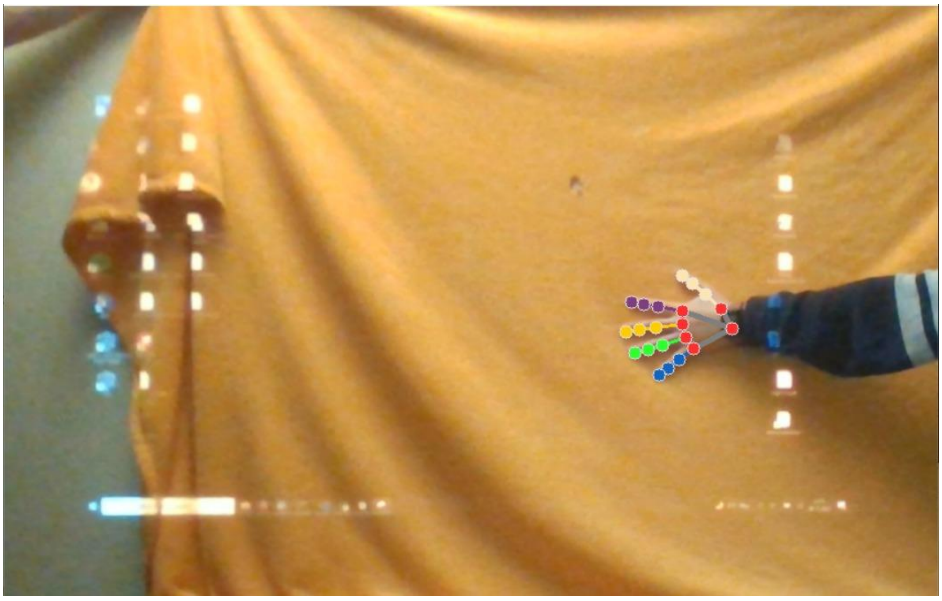


Abbildung 6: Frame von Experiment mit Wolldecke

Schatten

Da die Kamera und der Beamer nicht aus der genau gleichen Richtung auf das Bild gerichtet waren, gab es immer (ausser auf schwarzem Hintergrund) neben der Hand auch noch den Schatten der Hand. Bei weissem und grauem Hintergrund gab dies keine Fehlerkennung. Vor farbigem Hintergrund hingegen kam es vor, dass vereinzelt der Schatten der Hand anstatt der Hand erkannt wurde.

Dies waren jedoch nur vereinzelte Frames und das Experiment war nicht zuverlässig reproduzierbar. Man kann jedoch sagen, dass dieses Phänomen häufiger vorkam, wenn der Beamer im Vergleich zur Umgebung sehr hell war.

Hintergründe

Der grösste Unterschied, was das Erkennen der Hände anbelangt, waren die Hintergrundszenarien. Mit allen kontrollierbaren Variablen identisch wurde, je nach Hintergrund, die Hand zuverlässig an der richtigen Stelle erkannt oder es wurde nicht mal erkannt, dass eine Hand existierte.

Weiss

Vor einem komplett weissen Hintergrund funktionierte die API in fast allen Szenarien einwandfrei. Die einzige Ausnahme war Tageslicht und Raumbeleuchtung im Sitzungszimmer. In dieser Situation dauerte es jeweils ein paar Frames, bis die Hand erkannt wurde, wenn sie neu ins Bild kam.

Schwarz

Ein komplett schwarzer Hintergrund verhielt sich sehr ähnlich: Wenn die Hand überhaupt erkannt wurde, passierte dies sehr zuverlässig. In Szenarien mit dunkler Umgebung versagte die Mediapipe jedoch.

Farben

Vor farbigen Hintergründen wurde die Erkennung sofort merkbar schlechter. Mit weniger gesättigten Farben, schwächerem Beamer oder einem Scheinwerfer, welcher die Leinwand anstrahlt, funktionierte es wesentlich besser. Welche Farben

es waren, spielt auch eine Rolle: Gelb funktioniert recht zuverlässig, wenn auch merkbar schlechter als Weiss oder Schwarz, auf Grün und Rot wurde eine Hand ebenfalls gelegentlich erkannt und auf den anderen der Grundfarben wurde die Hand nie erkannt.

Steuerelemente

Wenn der Hintergrund nicht mehr einfarbig war, kamen noch weitere Schwierigkeiten hinzu: Auch auf Farben, auf welchen die Mediapipe Hände erkannte, wurde die Hand nicht mehr erkannt, wenn der Übergang zwischen diesen Farben (e.g. eine gerade schwarze Linie auf weissem Hintergrund) auf die Hand projiziert wurde. Die Mediapipe hat jedoch eine Option, den Videofeed als Video zu betrachten, anstatt jedes Frame separat anzuschauen. Dies ermöglichte es teilweise, dass eine Hand ausserhalb eines solchen Steuerelements erkannt wurde und dann beim Bewegen über das entsprechende Steuerelement nicht verloren ging. Dies galt jedoch nur für die Position der Hand, nicht aber der Finger, welche dann nicht mehr realitätsgerecht erkannt wurden.

Präsentationsfolie

Vor einer Präsentationsfolie im Stil, wie sie an der OST in Vorlesungen verwendet wird, wurde die Hand vor weissem Hintergrund und kleiner Schrift erkannt. Vor dem Titel der Folie und der Grafik wurde die Hand jedoch nur erkannt an Stellen, welche fast weiss waren und wenig Kontrast hatten.

Schwarzweiss Kontrast

Wurde ein Hintergrund projiziert, welcher zur Hälfte weiss und zur Hälfte Schwarz war, gab es wieder ein neues Problem: In dunkleren Umgebungen wurde gar nichts erkannt. In helleren Umgebungen kommt es auf die Stärke des Beamers an. Bei dem stärkeren Beamer aus dem Sitzungszimmer wurde auf dem weissen Teil des Bildes nichts erkannt, mit dem schwächeren Beamer auf dem schwarzen Teil.

Abgebildete Hände

Eine weitere Frage war, ob abgebildete Hände erkannt werden würden. Dies war nur in den hellsten Szenarien mit dem schwächeren Beamer der Fall.

Bewegende Hintergründe

Das letzte Hintergrundszenario sollte feststellen, ob es einen Einfluss hat, wenn sich nicht nur die Hand bewegt, sondern auch der Hintergrund. Vor einem Video von sich bewegendem Wasser wurde die Hand jedoch nicht merkbar schlechter erkannt, als wenn das Video pausiert wurde.

4.2.5.4 Probleme

Allgemein

Diese Ergebnisse zeigten ein paar Probleme auf, für welche im folgenden Kapitel Lösungsansätze aufgelistet werden.

Kontrast

Auf die Hand projizierte Muster und Linien verhindern, dass die Hand genau erkannt wurde. Was für das Bedienen von Steuerelementen und dem Anwendungsszenario einer Whiteboard-Anwendung fatal wäre.

Farben

Die meisten Farben verhindern, wenn sie auf die Hand gestrahlt werden, dass überhaupt eine Hand erkannt wird von der Mediapipe API. Dies wäre für das Anwendungsszenario Bedienen einer Powerpoint-Präsentation fatal.

4.2.5.5 Lösungsansätze

Mediapipe API

Die Mediapipe bringt in ihrer Funktionalität schon mehrere Optionen, um die Handerkennung potenziell zu verbessern. Diese wurden alle schon vor dem Durchführen der Tests optimiert.

`STATIC_IMAGE_MODE:`

Diese Option entscheidet, ob die Mediapipe auf jedem Frame neu Hände zu erkennen versucht, oder in vorhergehenden Frames gefundene Hände verfolgt, bis sie diese nicht mehr erkennen kann. Diese Option zeigt klar bessere Ergebnisse, wenn sie aktiviert ist.

`MIN_TRACKING_CONFIDENCE:`

Wenn `STATIC_IMAGE_MODE` deaktiviert ist, legt diese Variable fest, wie lange Hände getrackt werden, bevor aufgegeben wird und neue Hände gesucht werden. Da `STATIC_IMAGE_MODE` aktiviert ist, ist dieser Parameter irrelevant.

`MIN_DETECTION_CONFIDENCE:`

Die Mediapipe gibt für jede gefundene Hand einen Zuversichtswert heraus, dass die gefundene Hand auch eine Hand ist. Diese Variable definiert einen Mindestwert, den dieser erreichen muss, um im Output der Mediapipe zu landen. Wenn dieser Wert kleiner wird, erkennt die Mediapipe lange bevor sie reale Hände erkennt, welche sich zum Beispiel vor einer blauen Fläche befinden, fiktive Hände in den Falten der Kleidung des Users. Die besten Erfahrungen wurden mit dem Standardwert von 0.5 gemacht.

`MODEL_COMPLEXITY:`

Diese Variable ist beschränkt auf 0 oder 1 und gibt die Option auf Kosten von Genauigkeit die Rechenkosten zu senken. Hier gab es bessere Erfahrungen mit `MODEL_COMPLEXITY = 1.`

Scheinwerfer

Ein Ansatz, welcher beide Probleme lösen würde, ist das Aufstellen eines Scheinwerfers, der auf die Leinwand scheint. Dies zeigte beim Testen schon vielversprechende Resultate, bringt jedoch zwei weitere unlösbare Probleme mit sich: Erstens macht dies das projizierte Bild blasser und schwierig zu erkennen. Das zweite Problem ist, dass der Hauptvorteil der Arbeit sein soll, dass keine zusätzliche Hardware benötigt wird.

Selfie Segmentation

Ein weiterer Ansatz, welcher beide Probleme lösen würde, ist das Verwenden von Selfie Segmentation, um die Silhouette der Person aus dem projizierten Bild auszuschneiden, oder diesen Bereich abzudunkeln. Dies würde darin resultieren, dass vom Beamer kein Licht auf die Hand fällt, was beide Probleme lösen würde.

Low-Pass Filter

Für das Lösen des Problems mit dem Kontrast wäre ein Ansatz, dass irgendein Low-Pass Filter auf das Bild angewandt wird, welcher den scharfen Kontrast auf der Hand verwischt, die Hand jedoch so scharf stehen lässt, dass sie immer noch erkannt wird.

Hue Korrektur

Um das Problem mit dem Einfluss verschiedener Farben zu lösen, könnte man die Frames so bearbeiten, dass der Einfluss der auf die Hand projizierten Farben rückgängig gemacht wird. Einfachheitshalber wird das Bild hierfür in den HSV-Raum konvertiert und nur der Hue-Wert verändert. Um dies zu erreichen, gibt es zwei Ansätze:

1. Es wird gemessen, welcher Hue eine Hand haben muss, damit sie am besten erkannt wird, dann wird bei jedem Frame der Hue-Wert von jedem Pixel auf diesen Hue gesetzt.

2. Es wird für jede Farbe gemessen, wie sich der Hue einer Hand verändert, wenn sie mit dieser Farbe angestrahlt wird. Dann werden auf den Bereichen der Frames, welche mit dieser Farbe beleuchtet wurden, die umgekehrte Operation durchgeführt.

4.2.5.6 Neue Testergebnisse

Mediapipe API

Für die Optimierung über die Mediapipe selbst gab es keine neuen Testergebnisse, da diese Optimierung schon vor den ersten Testergebnissen stattgefunden hat. Für weitere Ideen, was die Ursachen der Probleme genau sein könnten, wurde das Team, welches die Arbeit über Mediapipe Hands veröffentlicht hat, erfolglos kontaktiert.

Low-Pass Filter

Für die Implementierung des Low-Pass Filters wurde ein Gaussfilter auf das Bild angewandt, bevor es in die Mediapipe eingespeist wurde. Die Kernelgrösse und die Varianz konnten dabei in real time auf Tastendruck angepasst werden. Dann wurde die Hand vor zunehmend dickere schwarze Linien gehalten und es wurde manuell eine Konfiguration gesucht, bei welcher sie erkannt wurde.

Low-Pass Filter Resultate

Während dies bei dünnen Linien durchaus funktioniert hat, sind zwei Probleme aufgetreten:

1. Die Parameter des Gaussfilters, mit welchem die Hand erkannt wurde, waren abhängig von der Liniendicke. Je dicker die Linie war, desto grösser musste der Kernel sein und desto kleiner musste die Varianz sein.

2. Bei Linien, welche breiter waren als ungefähr eine halbe Fingerbreite, gab es keine Kombination von Parametern, welche eine verbesserte Performance gegenüber dem Programm ohne Gaussfilter herbeiführten. Verstärkte man den Gaussfilter weiter, wurde die Hand so unscharf, dass sie auch ohne Kontrastlinien nicht mehr richtig erkannt wurde.

4.2.5.6.1 Hue Korrektur

Absicht

Falls das Problem mit den Farben sich tatsächlich daraus ergibt, dass sich der Farbton der Hand unter Beleuchtung ändert, sollte es möglich sein, die Hand erkennbar zu machen, indem der Hue im Bild verändert wird, bevor es in die Mediapipe eingespeist wird.

Testing

Zuerst wurde die Plausibilität getestet. Dafür wurde die Hand vor eine farbige Fläche gehalten, vor welcher sie nicht erkannt worden war, und dann wurde per Tastendruck der Hue des ganzen Bildes so lange erhöht, bis die Hand erkannt wurde. Dies funktionierte auf mehreren Farben. Deshalb wurden dann Testvideos gefilmt, um qualitative Aussagen über Verbesserung machen zu können. Diese sind eine Abfolge von Bewegungen einer Hand, vor den verschiedenen Farbenflächen. Getestet wurden diese Videos mit der folgenden Grösse «Gutheit»: Auf welchem Bruchteil der Frames dieser Testvideos wurde die Hand erkannt?

Die Gutheit für einen unveränderten Hue Wert beträgt 14.6%.

Strategie

Wenn man den Hue des ganzen Bildes einheitlich auf einen bestimmten Wert setzen will, muss dieser gewählt werden. Hierzu gab es zwei Ansätze: Einerseits, könnte man den Hue der Hand messen, wenn sie nur mit weissem Licht bestrahlt wird, andererseits könnte man auch den Hue Wert nehmen, welcher auf den Testvideos die beste Gutheit vorweist. Während der Vorteil des zweiten Ansatzes eine bessere Performance und Konsistenz zwischen verschiedenen Nutzern ist, ist der Nachteil, dass unklar ist, wie stark dadurch das Over Fitting auf die Testvideos ist. Es wurde entschieden, dass die Vorteile überwiegen und es wurde der optimale Hue Wert ermittelt. Um diesen zu bestimmen, wurde zuerst grob über den ganzen Farbraum gegangen, danach fein durch den Bereich, in dem das Maximum der groben Suche lag.

```
Average Score from hue 0 =  0.2153531245458086
Average Score from hue 5 =  0.2885634567648657
Average Score from hue 10 =  0.37935021350475445
Average Score from hue 15 =  0.35894473372405233
```

Average Score from hue 20 = 0.21315950890102958
Average Score from hue 25 = 0.10830822951653846
Average Score from hue 30 = 0.0624172981617128
Average Score from hue 35 = 0.07253742199551601
Average Score from hue 40 = 0.07573160434449226
Average Score from hue 45 = 0.062452760445788805
Average Score from hue 50 = 0.054576966180698724
Average Score from hue 55 = 0.04864355311725194
Average Score from hue 60 = 0.06470126192870854
Average Score from hue 65 = 0.0443612542563129
Average Score from hue 70 = 0.033699721207672284
Average Score from hue 75 = 0.03276077102779323
Average Score from hue 80 = 0.03525286117269706
Average Score from hue 85 = 0.030496161825664826
Average Score from hue 90 = 0.02466107137052854
Average Score from hue 95 = 0.02638336258491127
Average Score from hue 100 = 0.02940978371551969
Average Score from hue 105 = 0.025389197264689727
Average Score from hue 110 = 0.02258267582212275
Average Score from hue 115 = 0.03137964187487607
Average Score from hue 120 = 0.02996338754257306
Average Score from hue 125 = 0.054552822101949054
Average Score from hue 130 = 0.07126396269142128
Average Score from hue 135 = 0.08108450077667216
Average Score from hue 140 = 0.08429917696522564
Average Score from hue 145 = 0.08328728082924126
Average Score from hue 150 = 0.07940162927324876
Average Score from hue 155 = 0.09054822748666662
Average Score from hue 160 = 0.11198985361272706
Average Score from hue 165 = 0.1462528225813636
Average Score from hue 170 = 0.178171369215889
Average Score from hue 175 = 0.19424557330096714

Average Score from hue 6 = 0.3136632870453673
Average Score from hue 7 = 0.32726549433083474
Average Score from hue 8 = 0.336777387710889
Average Score from hue 10 = 0.37935021350475445
Average Score from hue 11 = 0.3813794451715976
Average Score from hue 12 = 0.3853453560371099
Average Score from hue 13 = 0.38790852076809296
Average Score from hue 14 = 0.38339754999323833

Bester Wert

Der beste Wert liegt also scheinbar bei 13.

Resultat

Für diesen Wert kann man jetzt die Güte noch weiter aufteilen, in Bezug darauf, wie sie sich vom Original auf den jeweiligen Hintergründen unterscheidet.

Bei unverändertem Bild kommen folgende Statistiken heraus:

white score: 0.14204545454545456
white-moving score: 0.05917159763313609
black score: 0.9941176470588236
black-moving score: 0.8546511627906976
blue score: 0.0
blue-moving score: 0.0
cyan score: 0.13245033112582782
cyan-moving score: 0.0
green score: 0.0
green-moving score: 0.0
magenta score: 0.0
magenta-moving score: 0.0
red score: 0.0
red-moving score: 0.0
yellow score: 0.32098765432098764

yellow-moving score: 0.016483516483516484
edges score: 0.0
blue-dimm score: 0.0
blue-moving-dimm score: 0.0
cyan-dimm score: 0.0
cyan-moving-dimm score: 0.0
green-dimm score: 0.0625
green-moving-dimm score: 0.0
magenta-dimm score: 0.0
magenta-moving-dimm score: 0.0
red-dimm score: 0.2206896551724138
red-moving-dimm score: 0.24666666666666667
yellow-dimm score: 0.7368421052631579
yellow-moving-dimm score: 0.5833333333333334
edges-dimm score: 0.009174311926605505
Average Score = 0.145970447877354

Bei Hue 13 ergeben sich diese:

white score: 0.9943181818181818
white-moving score: 0.6745562130177515
black score: 0.9941176470588236
black-moving score: 0.7209302325581395
blue score: 0.0
blue-moving score: 0.0
cyan score: 0.006622516556291391
cyan-moving score: 0.0
green score: 0.4943181818181818
green-moving score: 0.030120481927710843
magenta score: 0.4318181818181818
magenta-moving score: 0.04938271604938271
red score: 0.3146067415730337
red-moving score: 0.09375
yellow score: 0.7160493827160493
yellow-moving score: 0.12637362637362637
edges score: 0.38704581358609796
blue-dimm score: 0.0
blue-moving-dimm score: 0.0
cyan-dimm score: 0.09289617486338798
cyan-moving-dimm score: 0.0
green-dimm score: 0.7215909090909091
green-moving-dimm score: 0.011363636363636364
magenta-dimm score: 0.12418300653594772
magenta-moving-dimm score: 0.125
red-dimm score: 0.21379310344827587
red-moving-dimm score: 0.08666666666666667
yellow-dimm score: 0.5855263157894737
yellow-moving-dimm score: 0.36904761904761907
edges-dimm score: 0.06269113149847094
Average Score = 0. 38790852076809296

Während Schwarz und Weiss immer noch sehr gut funktionieren und Blau gar nicht, gibt es doch klare Verbesserungen mit vielen der anderen Farben.

Potential

Die Möglichkeit den Hue fix auf 13 zu setzen, bringt eine klare Verbesserung. Allerdings löst es das Farbenproblem nur für gewisse Farben und auch für diese ist die Handerkennung nachher noch um Einiges schlechter, als sie es bei weissem oder schwarzem Hintergrund ist. Dieser Lösungsansatz macht es auch sehr schwierig, die Werte noch weiter zu verbessern, da es beim besten, uniformen Hue Wert keinen Besseren geben kann.
Was danach versucht worden ist, war, anstatt alle Hue Werte gleich zu setzen, zu messen, um wieviel sich der Farbton der Haut ändert, wenn er vor einer

bestimmten Farbe ist. Nach dieser Kalibration wird für jedes gefilmte Frame ein Screenshot gemacht, und dann wird pro Pixel das Frame um diesen Wert verändert. Dies schien sehr vielversprechend. Bei Tests, bei denen die Veränderung des Wertes, welcher zum Hue addiert wurde, manuell verändert wurde, konnte die Hand fast immer erkannt werden. Es wurde jedoch keine performante Lösung implementiert, welche das Bild in Echtzeit dem angepasst hat, was der Beamer ausgestrahlt hatte. Dies hätte jedoch nur das Farbproblem gelöst und zur Lösung des Kontrastproblems nichts beigetragen, was darin resultierte, dass dieser Ansatz aus Zeitründen nicht weiterverfolgt wurde.

4.2.5.6.2 Selfiesegmentation

Absicht

Die beiden bestehenden Probleme *Kontrast* und *Farben* wären auf einen Schlag lösbar, wenn auf der Fläche, wo eine Person erkannt werden kann, gar keine Projektion stattfindet. Daher könnte man eine transparente Schwarzfläche abbilden, wo auch immer der Umriss einer Person erkannt wird.

Vorgehen

Die MediaPipe API bietet eine Funktion namens `mp.solutions.selfie_segmentation` [4] an. Diese ermöglicht es dem Anwender der Bibliothek auf einem beliebigen Bild die Konturen einer Person zu finden. Dies wird so angewandt, dass auf einem Kamera-Bild Konturen erkannt werden, welche dann auf einem transparenten, generierten Bild mit einem dunklen Farbton abgebildet werden, welches über das projizierte Bild gelegt wird.

Resultat

Unglücklicherweise hatte dieses Vorgehen zur Folge, dass die erkannten Konturen der Person *zu wachsen* schienen. Der Algorithmus erkannte offenbar die Konturen nun als Teil der Person und erweiterte diesen Bereich stetig. Dies führte zu einer etwas surrealen Darstellung. Wo auch immer eine Person erkannt wurde, begann ein dunkler Bereich sich auszubreiten. Da offenbar das Überlagern des Bereiches, wo eine Person erkannt wird, einen zu starken Effekt auf das Erkennen besagter Person hat, wurde dieses Verfahren nicht weiterverfolgt.

Potential

Unter der Annahme, dass für die Selfiesegmentation geeignete Trainingsbilder verwendet würden, hätte dieses Verfahren beinahe unbegrenztes Potential, da sowohl *Kontrast* als auch *Farbe* absolut keine Rolle spielen würden und auch weder Bewegung noch sonst etwas darauf einen Einfluss hätten, ob eine Hand erkannt werden kann oder nicht.

4.2.6 Analysephase Schlussfolgerungen

Vorgehen

Nach Abschluss der Analysephase, wurde in einem Meeting mit den Studierenden, dem Dozenten und den Experten festgelegt, was das Ziel der Arbeit schlussendlich sein soll.

Problem mit der Mediapipe

Die Probleme, welche sich daraus ergeben, Mediapipe Hands auf Hände anzuwenden, welche sich vor einer bestrahlten Leinwand befanden, wurden im Rahmen dieser Arbeit als unlösbar eingestuft. Obwohl Lösungsansätze vorhanden waren, wäre der geschätzte Aufwand zu gross gewesen, um diese bis Ende der Arbeit so zu implementieren, dass diese dann zuverlässig funktionieren würden.

Änderung des Mediums

Deshalb wurde beschlossen, dass die Zielsetzung dieser Arbeit abgeändert werden würde auf einen grossen Bildschirm anstatt einer Leinwand. Damit, dass das Licht nun von der Rückseite an die Hand leuchtete, sollten diese Probleme wegfallen.

Ziel der Arbeit

Das am Meeting definierte Ziel war es dann, eine Anwendung zu machen, welche die Mediapipe und eine Kamera benutzt, um mit Gestikulieren ein einfaches Programm bedienen zu können.

4.3 Anforderungsspezifikation

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	20.10.2021

Gültigkeitsbereich	Die folgenden Anforderungen beziehen sich auf den Teil der Software, welcher durch die Studenten erstellt wurde. Die Erweiterung der Funktionalität der MediaPipe API liegt ausserhalb der Aufgabenstellung.

4.3.1 Funktionsanforderungen an die Software

Funktionsumfang	Die Software soll nach Möglichkeit in den in den folgenden Kapiteln beschriebenen Szenarien einsetzbar sein. Diese Szenarien sind aufgeteilt in Infrastruktur- und Anwendungs-Szenarien. Die daraus resultierenden Test-Szenarien setzen sich zusammen aus Infrastruktur-Szenario, Hintergrundbild-Dimension und Anwendungs-Szenario.

4.3.2 Anwendungs-Szenarien

Bedienung von PowerPoint	Das Hauptanwendungsszenario ist die Verwendung von Präsentationssoftware insbesondere MS Powerpoint. Dementsprechend sind die Hintergrund-Dimensionen jeweils als Powerpoint-Präsentation gestaltet, damit in den getesteten Infrastruktur-Szenarien ein möglichst rascher und reibungsloser Testverlauf ermöglicht wird.
Diashow bedienen	Die Bedienung einer Diashow mit vorwärts und rückwärts Wechseln in einer Bildfolge.
Bedienung MS Paint	Zeichnen mit einem rudimentären Zeichenprogramm wie MS Paint. Hierbei steht im Fokus, dass der Benutzer eine Skizze machen kann von etwas, das er sonst an einem Whiteboard oder einer Wandtafel erstellt hätte.

4.3.3 Use Cases

Kamerakalibration	Durch einen Knopfdruck soll eine Kalibration der Kameraverzerrung durchgeführt werden können, alternativ dazu sollte eine Option es ermöglichen, die Software auch mit einer Default-Kalibration zu verwenden.
Bildschirm erkennen	Erkennungsprozess starten und eine Rückmeldung erhalten, ob der Bildschirm gefunden wurde oder nicht.
Linksklick	Zeigefinger und Daumen zusammendrücken sollte den Linksklick der Maus ausführen.
Rechtsklick	Die *Schwurfinger*-Gestik bewirkt einen Rechtsklick auf der Maus.
Zusatztaste	Mit der sogenannten *Spiderman*-Gestik, kann eine zusätzliche Funktion über eine Geste ausgeführt werden. Dies liefert aber keine Koordinaten sondern löst lediglich eine entsprechend im UI ausgewählte, voreingestellte Bedienung auf der Tastatur aus. (MyPaint: «Taste E» und PowerPoint: «Pfeil zurück»)

4.3.4 Implizite Funktionalität

Kamerakalibration	Die Kamerakalibration soll nur einmal gemacht werden müssen, da sie eine gewisse Zeit in Anspruch nimmt und daher sollte die Information dazu auf dem Dateisystem gespeichert werden.
Kamerawahl	Die verwendete Kamera sollte nach Möglichkeit ausgewählt werden können, so dass auch eine externe Kamera an das System angeschlossen und zum Tracking verwendet werden kann.

4.3.5 Nicht-funktionale Anforderungen

Räumliche Infrastruktur	Die Software soll in einem Präsentationsraum funktionieren, der nicht vollkommen abgedunkelt aber auch nicht unnatürlich stark ausgeleuchtet ist. Der Erfolg der Ausführung der Software wird entsprechend der Thematik stark davon abhängen, ob die Lichtsituation günstig gewählt ist. Dies wird der automatischen Lichtkorrektur geschuldet und kann von den Studierenden nur sehr schwer beeinflusst werden.
Geräte	Die dazu verwendeten Geräte sollen eine für den Markt von 2021 entsprechende Leistungsfähigkeit besitzen und über eine entsprechende Webcam verfügen. Bezüglich der Auflösung und der Videoframerate werden keine Voraussetzungen gestellt, angenommen wird eine Framerate von 15fps und eine Auflösung von 1280x720, was einer HD-Auflösung entspricht. Für den Projektor wird eine Auflösung von 1920x1080 Pixel angenommen, was der Full-HD Auflösung entspricht, allerdings werden auch hier keine Voraussetzungen gestellt.
Positionserkennung	Die erkannten Cursorpositionen sollen angemessen sanft und kontinuierlich aneinandergereiht werden, so dass eine möglichst zitterfreie Linie entsteht.

4.4 Design FreeCursor

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	19.11.2021

Kapitelinhalt

Dieses Kapitel behandelt die Designüberlegungen in Bezug auf die Software. Da der Umfang der Software etwas reduziert ist, weil ein Teil der eingesetzten Mittel erst empirisch in Erfahrung gebracht werden mussten, werden hier die relevanten Zusammenhänge dargestellt und auf den Inhalt der MediaPipe API kann wegen mangelnder Informationen nicht eingegangen werden.

4.4.1 Architektur von FreeCursor

Lokaler Client

Da im Moment kein Grund besteht, die Applikation über mehrere Maschinen und Prozesse zu verteilen, ist es am zielführendsten einen lokalen Client zu erstellen, welcher möglichst wenig Konfiguration bedarf und sich so gut es geht selbst an die Umgebung anpasst.

4.4.2 Bildschirmerkennung

Verfahren

Zur Erkennung des projizierten Bildes wird eine OpenCV Funktion genutzt namens `cv2.findContours()`. Diese funktioniert am besten auf vorbearbeitetem Bildmaterial und dementsprechend werden einige Vorgehensweisen angewendet, um ein möglichst optimales Bild zu erhalten, in welchem die Konturen der Projektion sicher zu bestimmen sind. Diese Verfahren werden im Folgenden erläutert und umfassen das Erstellen eines Differenzbildes mittels verschiedener projizierter Weiss- und Schwarzflächen, die Korrektur des Kamerainputs mittels Charuco-Indize und die Bearbeitung des Resultats mit einer LookupTable.

Differenzbild

Da der Kontrast eines mit der Webcam aufgenommenen Bildes in der Regel nicht ausreicht, um das Rechteck der Projektion zu bestimmen, weil in den meisten Fällen der umliegende Raum viel zu viele störende Konturen aufweist, wurde ein Verfahren angewendet, wo erst ein weisses Vollbild eingeblendet wird und danach dasselbe Vollbild, aber mit einem schwarzen Rand. Da nun die beiden Bilder voneinander subtrahiert werden können, ist das Endresultat jeweils nur die Kontur des Bildschirms und in wirklich günstigen Lichtverhältnissen wird beinahe der gesamte umliegende Raum ausgelöscht, respektive mit Null oder Schwarz abgebildet.

Charuco-Indize

Da das Inputmaterial, welches wir von der Kamera erhalten bestenfalls mässig verzogen ist, muss dieses mit einer ausgemessenen Transformationsmatrix korrigiert werden. Dazu kann die OpenCV-Funktion `cv.remap()` verwendet werden, sofern vorher eine Matrix mit `cv2.aruco.calibrateCameraCharuco()` erstellt wurde.

LookupTable

Obwohl in vielen Situationen die Lichtverhältnisse ausreichen, um das projizierte Rechteck zu erkennen, wird ein weiteres Verfahren darüber angewandt, um auch Bilder von Projektoren mit nur sehr schwachen Lichtquellen zu erkennen. Dazu wird eine Gradationskurve über den gesamten Bildbereich berechnet, welche die hellsten Bildpunkte zu Weiss, den Median und etwas darüber zu Schwarz umwandelt und die Bildpunkte, welche sich dazwischen befinden anhand einer linear-harten Gradation umwandelt.

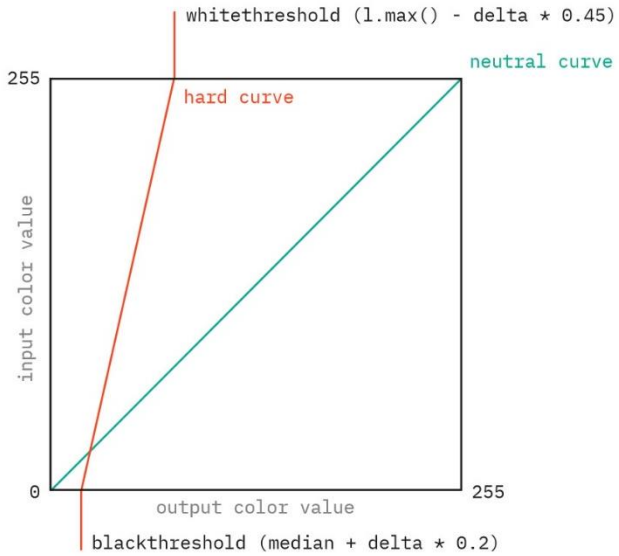


Abbildung 7: Automatische Gradationskurve

4.4.3 Objektkatalog

Tracker	Dient als Schnittstelle zur UI und startet das Tracking. Besitzt eine *ProjectionHandler*-Instanz um das Rechteck des Screens zu berechnen.
ProjectionHandler	*ProjectionHandler* verfügt über die Auflösung des Screens und die Auflösung der Kamera; hat eine *CharucoUtility*-Instanz um die Kamera zu kalibrieren und eine *FrameUtility*-Instanz um die Bilder mit Rand oder ohne Rand darzustellen.
FrameUtility	Bietet Funktionen an zur Transformation, Generierung und Darstellung von *Frames;* in diesem Zusammenhang bedeutet der Begriff Frame eine Pixelmatrix von Farbtonwerten mit einer bestimmten Auflösung.
CharucoUtility	Dieses Objekt ist verantwortlich für die Kalibrierung der Kamera.
GeometryUtility	Um die Konsistenzbedingungen des Screens zu checken, braucht es einfache geometrische Funktionen, welche von der GeometryUtility zur Verfügung gestellt werden.
HandUtility	Dieses Objekt ist verantwortlich für den Moving-Average der erkannten Koordinaten und Klickvariablen.
TrackingThread	Dieses Objekt ist verantwortlich für die Parallelisierung des Trackingvorgangs zum Mainthread, der das UI beinhaltet.
CalibrationThread	Dieses Objekt ist verantwortlich für die Parallelisierung des Kalibrationsvorgangs.
ControllerWidget	Dieses Objekt beinhaltet die UI-Konfigurationselemente und startet die Threads.
FreecursorWindow	Dieses Objekt beinhaltet das ControllerWidget und empfängt Keyboard-Interrupts.

4.4.4 Klassendiagramm FreeCursor

Klassendiagramm von FreeCursor

Im Folgenden sieht man eine Übersicht der Klassen, die für das Tracking geschrieben wurden. Entsprechend dazu wurden Tests definiert, welche automatisiert ausgeführt werden. Mehr zur Implementation finden Sie im dazugehörigen Kapitel.

Was als wichtig zu beachten gilt, ist, dass lediglich der Tracker eine Abhängigkeit zur MediaPipe aufweist, was dann vorteilhaft ist, falls der Code einmal mit einer anderen Handerkennungsbibliothek ausgeführt wird, sollte die Anpassung nur minimalen Aufwand kosten.

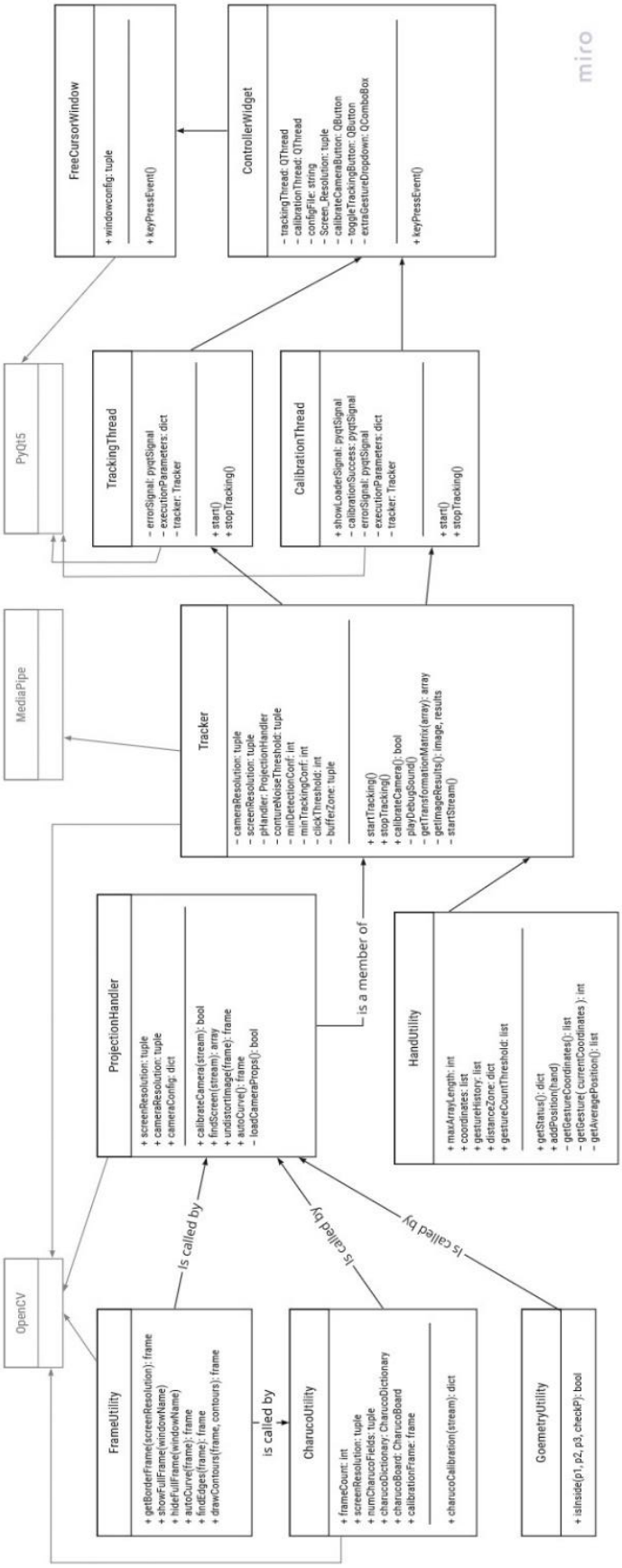


Abbildung 8: Klassendiagramm

4.4.5 Sequenzdiagramm TrackingSequenz

Trackingsequenz

Das Sequenzdiagramm einer Trackingsequenz zeigt die relevanten Funktionsaufrufe der Klassen auf, um eine Übersicht über die einzelnen Schritte zu erhalten, welche das Programm ausführt, um die Gestiken über dem projizierten Bild zu tracken.

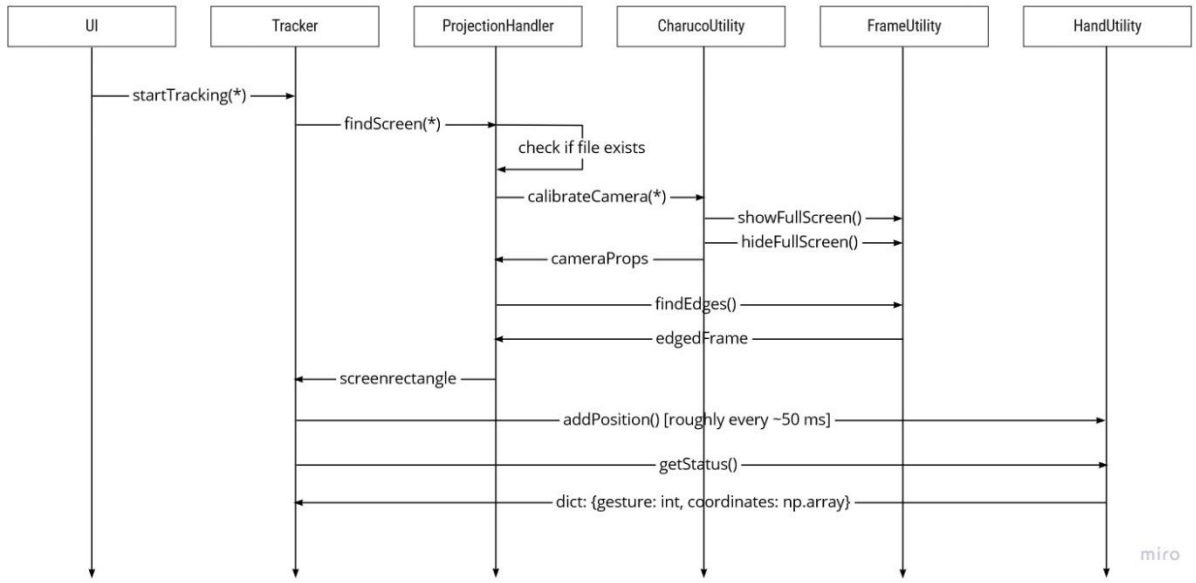


Abbildung 9: Sequenzdiagramm Trackingsequenz

4.5 Implementation und Test

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	12.11.2020

Kapitelinhalt Dieses Kapitel befasst sich mit den Einzelheiten der Implementation. Der Teil welcher als Funktionalität von der MediaPipe API übernommen wird, ist hier nicht behandelt, weil dazu die Informationen über den Aufbau fehlen.

Aufbau Der Aufbau des Kapitels ist so gehalten, dass erst die relevanten Klassen im Detail beschrieben werden, dann werden die entsprechenden Unit-Tests und die CI-CD-Pipeline behandelt und zum Schluss wird das Verfahren beschrieben, welches als manueller Test des Endresultats als Usability-Testverfahren verwendet wurde.

Anmerkung Als erste Ausgangslage für einige Funktionalitäten diente eine *Anleitung zur Erkennung von Gesichtern* [1] vor einem Projektor mittels eines RaspberryPI mit Kameramodul. In der finalen Version von FreeCursor sind nur sehr wenige Teile dieser Anleitung noch vorhanden, namentlich ist damit die Bildtransformation anhand der Kameramatrix gemeint.

4.5.1 CharucoUtility

Grund Die heute weit verbreiteten Lochkameras bringen viel Verzerrung mit sich, damit ist hier hauptsächlich radiale und tangentiale Verzerrung gemeint. Durch radiale Verzerrung erscheinen gerade Linien beliebig gekrümmt. [5]

Kalibrationsmuster Ursprünglich wurde zur Korrektur von Objektivverzerrungen jeweils ein Schachbrettmuster auf einem ebenen Brett verwendet. Daraus konnte die Krümmung berechnet werden und dementsprechend eine Transformationsmatrix zur Korrektur. Weil die Erkennung eines Schachbretts gewisse Nachteile aufweist, zum Beispiel, dass eine Kalibration nur dann möglich ist, wenn das gesamte Schachbrett gefunden wird, was bei schlechten Lichtsituationen nahezu unmöglich ist, ist Charuco-Kalibration ein sehr gutes und fortgeschrittenes Mittel zur Kalibration einer Webkamera. [6]

Vorgehen Beim Erstellen des Objekts wird ein Charucoboard erstellt in der Auflösung des Projektors, die übergeben werden muss zusammen mit der Auflösung der Webcam. Danach startet ein Loop zur Erkennung der Charuco-Indize, wo man

zuerst die Helligkeit mittels ‘w’ und ‘e’-Tasten definiert und sobald man den Ausleseprozess mittels Drücken der ‘s’ Taste startet, werden Bild für Bild Indize eingelesen über 50 Frames, welche danach dazu benutzt werden, die Transformationsmatrix und die Verzerrungskoeffizienten zu berechnen.

Herausforderung

Im Moment wird sowohl die Quote der erkannten Indize, als auch der Fehler der aus der Erkennung resultierenden Matrix, im Kamera-Properties-Objekt gespeichert. Ist dieser Fehler grösser als 0.3 wird die Erkennung des Bildschirms beinahe unmöglich sein und eine Neukalibration ist notwendig.

4.5.2 ProjectionHandler

Aufgabe

Die ProjectionHandler-Klasse ist zuständig für das Verwalten der Kamerakonfiguration, die Erkennung des Bildschirms und alle Informationen, die das projizierte Bild anbelangen.

Abhängigkeiten

Diese Klasse ist am stärksten verbunden mit der *FrameUtility*-Klasse, weil die meisten Zugriffe auf Methoden der *FrameUtility* aus dem *ProjectionHandler* aufgerufen werden. Der *ProjectionHandler* greift, um die Kamera zu kalibrieren, auf die *CharucoUtility* zu, dies sollte aber pro neuer Installation nur einmal geschehen müssen, weil die Kameraverzerrung als Konstante gehandelt wird und solange die Kalibration einen Fehler aufweist, der kleiner ist als 0.3, wird die Kalibration auch nicht neu ausgeführt, sofern schon eine *Kalibrationsdatei* vorhanden ist. Die Standardbezeichnung für die *Kalibrationsdatei* ist *«camera_config.json»* und ist auch die Datei, nach welcher gesucht wird, sofern nicht anders übergeben bei der Instanzierung des Objekts.

Wichtige Funktionen

Die wichtigste Funktion dieser Klasse ist die `undistortImage()` Funktion. Hier handelt es sich um die lineare Umformung der Pixelmatrize des Frames, bei welcher die radiale Verzerrung der Linse *rausgerechnet* wird, damit danach darauf aufbauende Mathematik möglich ist. Die Funktion `findScreen()` wird nur einmal aufgerufen und gibt die Screenkoordinaten zurück, sofern ein Screen gefunden wurde, sonst gibt es ein `np.Array` mit vier Nulltupeln zurück.

4.5.3 Tracker

Aufgabe

Die Tracker-Klasse ist das Herzstück der Applikation. Sie ermöglicht es, einen Stream zu starten und zu beenden. Sie dient als Schnittstelle zum Tracker-/CalibrationThread und damit zum *ControllerWidget* des Graphic User Interface (GUI). Diese Klasse ist nur sehr schwer oder gar nicht zu testen, da sie einen Lifestream mit realer Information benötigt.

Abhängigkeiten

Diese Klasse ist mit beinahe allen Klassen direkt oder indirekt verbunden. Sie besitzt als Memberklassen einen *ProjectionHandler* und eine *HandUtility.* Damit nur eine Klasse vom GUI angesteuert wird, besitzt sie Funktionen, in welchen sie Funktionsaufrufe an die anderen Klassen weiterleitet, wie zum Beispiel die `calibrateCamera()` Funktion.

Wichtige Funktionen

Die wichtigste Funktion dieser Klasse ist die `startTracking()` Funktion, womit ein Stream geöffnet, dann der Bildschirmerkennungsprozess gestartet wird und danach der Trackingloop läuft.

Execution Parameters

Um die Klasse initialisieren zu können braucht es einige Angaben. Beim Umfang dieser Angaben wurde darauf geachtet, möglichst wenige Informationen vom Benutzer zu verlangen, damit die Settings nicht endlos Zeit in Anspruch nehmen. Ein Beispiel eines *ExecutionParameters*-Dictionary-Objekts sieht wie folgt aus:

```
executionParameters = {
  "cameraResolution": (1280, 720),
  "screenResolution": (1920, 1080),
  "cameraID": 0,
  "cameraPropsFile": "default_camera_config.json",
  "extraGestures": ExtraGestures.MYPAINT
}
```

4.5.4 Handutility

Aufgabe

Die HandUtility-Klasse bekommt vom Tracker die Position der erkannten Hände und ermittelt daraus, welche Aktionen ausgeführt werden sollen.

Abhängigkeiten

Diese Klasse hat keine Abhängigkeiten.

Wichtige Funktionen

Für die HandUtility-Klasse gibt es zwei wichtige Funktionen. Einerseits `addPosition(hand)`, was vom Tracker aufgerufen wird und der HandUtility neue Koordinaten einer Hand übergibt, und andererseits `getStatus()`, was dem Tracker zurückgibt, welcher Button gedrückt werden soll, wie auch die Koordinaten, an welche die Maus bewegt werden muss.

Vorgehen

Wenn der HandUtility neue Koordinaten einer Hand übergeben werden, wird zuerst mit Hilfe des relativen Abstandes verschiedener Fingerspitzen die Geste bestimmt. Abhängig davon wird dann die Position des Klickens ausgerechnet (bei Links- und Rechtsklick). Bei X wird dann diese Geste herausgegeben, welche in den letzten acht Frames am häufigsten erkannt wurde, vorausgesetzt, dass diese ein Minimum von zweimal vorkommt. Die Position, welche zurückgegeben wird, ist die durchschnittliche Position der letzten acht Frames, bei denen die zurückgegebene Geste erkannt wurde.

4.5.5 TrackingThread

Aufgabe

Die *TrackingThread*-Klasse ermöglicht es dem GUI weiterhin bedienbar zu sein, währenddem das Tracking ausgeführt wird. Wäre der Loop, der die Bilddaten auswertet, direkt ins GUI eingebunden, dann würde das GUI stecken bleiben und man müsste per Keyboard-Interrupt den Prozess beenden.

Abhängigkeiten

Diese Klasse ist logischerweise abhängig von der *Tracker*-Klasse, denn sonst wäre ein Tracking in dem Sinne nicht möglich. Den Code für den Tracker in die *TrackingThread*-Klasse einzufügen, wäre vom Standpunkt «Seperation of Concerns» eine denkbar unglückliche Lösung. Die *TrackingThread*-Klasse beherbergt somit lediglich Funktionen, welche ihr von PyQt5 zur Verfügung gestellt wurden, damit ist ein Unit-Testing in dem Sinne nicht zwangsläufig notwendig.

Wichtige Funktionen

Die wichtigste Funktion der *TrackingThread*-Klasse ist, dass sie ein PyQt-Signal beinhaltet. Dieses Signal ermöglicht es dem Thread, sich beim *ControllerWidget* zu melden, sobald was schief geht.

4.5.6 Gitlab-Pipeline für Unittests

Gitlab Pipeline

Da Gitlab schon ein automatisiertes Testverfahren anbietet, welches in einem Container, online auf der Cloud ausgeführt wird, konnte ein Grossteil des automatisierten Testens mit einer Gitlab-Pipeline abgehandelt werden.

Umfang

Um möglichst nur Code zu testen, der von den Studierenden geschrieben wurde, hat das Team sich auf die wesentlichen Klassen beschränkt. Der Tracker bildet da eine besondere Ausnahme, weil die Funktionalitäten des Trackers hauptsächlich mit einem Life-Stream testbar sind. Für alles was vom Team per Stub-Stream getestet werden konnte, wurden entsprechende Unittests geschrieben.

Pipeline-Sequenz

Die Sequenz der Pipeline sieht wie folgt aus:

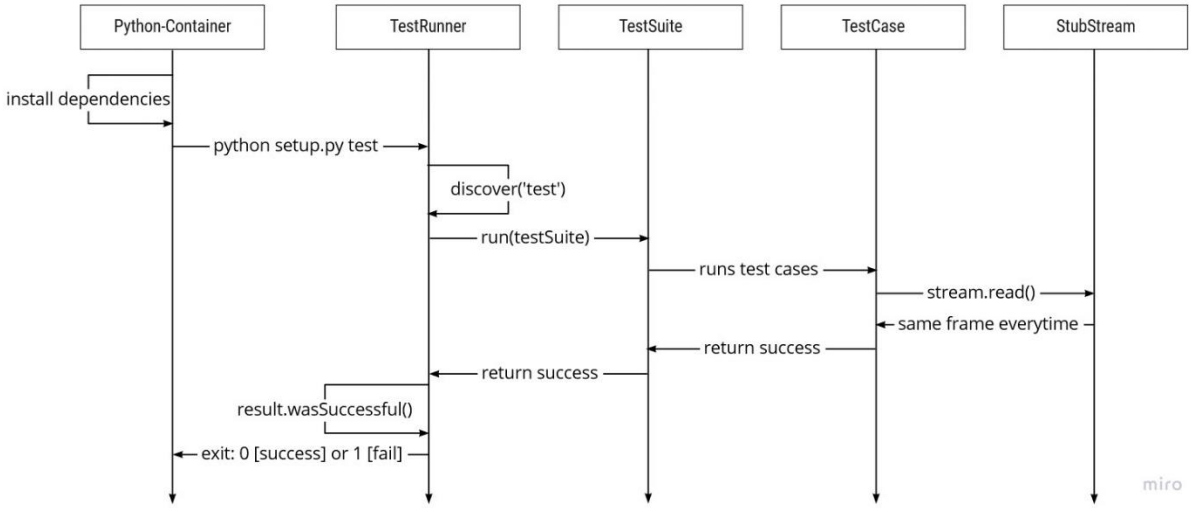


Abbildung 10: Sequenzdiagramm Pipeline

Da der *TestRunner* am Schluss tatsächlich eine 1 als Exit-Status zurückgibt, falls die *TestSuite* einen Test darunter hatte, der schief gelaufen ist, wird die Pipeline auch im Gitlab-Webinterface rot eingezeichnet. Dies wäre nicht der Fall, wenn in setup.py nicht nach dem Status des Resultats der Tests gefragt würde.

Pipeline-Code

Der tatsächliche Code in der Datei `gitlab-ci.yml` ist vergleichsweise winzig.

```
image: python:3.9

before_script:
  - apt-get update
  - apt-get install ffmpeg libsm6 libxext6 -y
  - python -V  # Print out python version for debugging
  - pip install -r testrequirements.txt

test:
  script:
    - python setup.py test
```

4.5.7 Manuelle- und Usability-Tests

Umfang

Die Tests, welche die MediaPipe-API betreffen, werden in diesem Kapitel nicht abgehandelt. Dieses Kapitel befasst sich mit den durchgeführten Usability-Tests und deren Resultaten.

Manuelle Tests

Während des Entwicklungsprozesses wurden mehrfach manuelle Tests durchgeführt. Folgende Tests waren dabei enthalten.

4.5.7.1 Kamera Kalibration

Beschreibung

Test zur Kalibration der Kamera mittels eines Life-Streams.

Voraussetzung

Geräte in Position und Lichtverhältnisse angepasst

Ablauf

Vorgehensweise Schritt für Schritt:

#	Aktion	Erwartung
1	Starten des Scripts `charucoutility.py`	Öffnen eines Fensters mit dem momentanen Input der Videokamera. Nach manueller Eingabe von «W» beziehungsweise «E» wird das Bild heller respektive dunkler. Nach Eingabe von «S» startet die Auswertung der entdeckten Charuco-Indize. Dann wird die errechnete Kamera-Matrix auf der Konsole ausgegeben.
2	Script endet automatisch	Ablesen der Werte. Ein `err`-Wert unter 0.3 und ein `avg_detection_ratio`-Wert über 0.5 hat sich in den folgenden Test-Cases als ausreichend erwiesen.
3	Falls nötig anpassen der Lichtverhältnisse	Bessere Resultate in den Werten.

Kriterien

Der Test ist erfolgreich, wenn die folgenden Test-Werte erreicht wurden: `err`-Wert unter 0.3 und `avg_detection_ratio`-Wert über 0.5

4.5.7.2 Bildschirm Erkennung

Beschreibung Test zur Erkennung des Bildschirms auf dem Life-Stream.

Voraussetzung Geräte in Position und Lichtverhältnisse angepasst.

Ablauf Vorgehensweise Schritt für Schritt:

#	Aktion	Erwartung
1	Starten des Scripts `projectionhandler.py`	Öffnen eines Fensters mit einer farbigen Kontur um dem Bildschirm, sofern der Bildschirm gefunden wurde, sonst eine Ausgabe auf der Konsole mit [ERROR] und der Angabe was schief gelaufen ist.
2	Drücken der Whitespace	Schliessen des Fensters

Kriterien Der Test ist erfolgreich, wenn auf dem Input-Bild der Bildschirm gefunden und korrekt eingezeichnet wurde.

4.5.7.3 Gestik Erkennung

Beschreibung Test zur Erkennung der Position einer Gestik auf dem Life-Stream.

Voraussetzung Geräte in Position und Lichtverhältnisse angepasst, Kamera kalibriert.

Ablauf Vorgehensweise Schritt für Schritt:

#	Aktion	Erwartung
1	Starten des Scripts `tracker.py`	Bildschirmerkennung wird ausgeführt. Life-Stream wird gestartet.
2	Mit der Hand so nah wie möglich auf den Bildschirm oder das projizierte Bild gehen und dann Zeigefinger und Daumen zusammendrücken.	An der Stelle zwischen Zeigefinger und Daumen wird ein Links-Klick ausgeführt.

Kriterien Der Test ist erfolgreich, wenn auf dem Gerät, wo der Tracker läuft, an der Stelle zwischen Zeigefinger und Daumen ein Links-Klick ausgeführt wird.

4.5.7.4 Usability-Tests

Durchführung

Die Usability-Tests wurden jeweils in einem Setting mit direktem Kontakt zum Studierenden ausgeführt. Da der Entwickler damit direkt ansprechbar war, entstand die grosse Herausforderung, dass die Testperson nicht zu viele Fragen gleich beantwortet erhielt, denn das Ziel der Tests war es herauszufinden, ob ein Anwender mit Hilfe der grafischen Oberfläche die grundlegendsten Features der Applikation nutzen konnte. Entsprechend der Idee von User-Centered-Design lernt das Entwicklungsteam am meisten, wenn möglichst viele Missverständnisse erkannt und dokumentiert werden.

Funktionsumfang

Entgegen der Praxis, wo zuerst mit einem Mockup getestet wird, hat sich das Team dazu entschieden, die Tests mit der laufenden Applikation zu machen, da der Code schon zum grossen Teil bestanden hat.

Ergebnisse

Das wichtigste Ergebnis war die Anpassung der Instruktionen, welche vom Team ausgeführt wurde. Hinzu kommt eine grundsätzlich positive Bilanz bezüglich einer möglichen Weiterentwicklung der Applikation und ein kollektiver Wunsch nach einer besseren Form der Gestik-Erkennung. Auch die Gesten waren offenbar nicht für alle Teilnehmer gleich von Anfang an nachvollziehbar. Dem wurde entgegengewirkt, indem auf den Instruktionen nun Echtbilder verwendet werden und nicht mehr wie gehabt Piktogramme.

Resultate

Die Resultate der Usability-Tests sind nachzulesen im Testkonzept mit den entsprechenden Protokollen im Anhang.

4.6 Stand der Arbeit und Zukunftsaussicht

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	06.01.2022

Kapitelinhalt

In diesem Kapitel wird kurz geschildert, was als Resultat der Arbeit herausgekommen ist. Dabei wird erst kurz auf die sichtbaren Ergebnisse, die Teile der Arbeit, die noch nicht umgesetzt werden konnten und die daraus gewonnenen Kenntnisse eingegangen, dann werden Möglichkeiten aufgezeigt, was als weiterführende Schritte unternommen werden könnte und wie ein Vorgehen aussehen könnte.

4.6.1 Stand der Arbeit

Erledigte Arbeit

In der Arbeit sind folgende Ergebnisse erarbeitet worden:

- Testergebnisse bezüglich der MediaPipe API
- Schwierigkeiten beim Einsatz der API namentlich:
  - Ungenügende Erkennung bei projizierten Farben
  - Ungenügende Erkennung bei projizierten Mustern
- Schon ausprobierte Lösungsansätze für diese Schwierigkeiten und die Ergebnisse
- Applikation mit folgenden Features:
  - Kalibration der integrierten Webkamera zur Korrektur der Bildverzerrung
  - Erkennen eines externen Bildschirms auf einem Life-Stream
  - Erkennen von 3 analytisch unterscheidbaren Gesten mittels Analyse eines Life-Streams
  - Auslösen von Klicks auf dem trackenden Gerät
  - Grafische Benutzeroberfläche zum Bedienen der Applikationsfunktionen
- Vorschau-Button für den Video-Stream

Offene Arbeiten

Die folgenden Arbeiten wurden vom Team zwar auf die Liste der notwendigen Arbeiten gestellt, allerdings konnten sie bis zum Ende der Arbeit nicht mehr ausgeführt werden.

- Verbesserung der Bildschirmerkennung durch Einsatz einer Radon Transformation um die Eckpunkte des Bildschirms zu berechnen
- Verbesserung des Moving-Average durch Einsatz von Exponential-Smoothing
- Verbesserung der Instruktionen, durch Einsatz von weiteren Hinweisfenstern insbesondere ein «detect screen» Fenster
- *ExraGesture* über eine Keyboardeingabe konfigurierbar machen

4.6.2 Mögliche weitere Features

Neuronales Netzwerk

Der wohl wichtigste Schritt, der die Applikation um einen grossen Teil verbessern würde, wäre das Erstellen eines neuronalen Netzwerks, welches bestenfalls mit Videos dazu trainiert wird, Handgesten zu erkennen. Dazu müssten Bilder oder Videos erstellt werden in einem entsprechenden Setting, das heisst vor einem Beamer, damit das Netzwerk schon beim Trainingsvorgang mit der Schwierigkeit konfrontiert ist, dass eine Hand auch von einer gesättigten Farbe belichtet sein kann.

Dynamische Gesten

Die Eingrenzung auf analytisch berechenbare Gesten kommt daher, dass dynamische Gesten nur erkannt werden können, wenn man sie entweder als fixe Folgen hart-codiert oder via Deep-Learning erkennt. Diese Erweiterung könnte genauso, wie das Problem mit den Farben und Mustern, durch ein neuronales Netzwerk angegangen werden.

Vollumfängliche Bedienung

Den Bedienungsprozess jeweils zu starten und zu beenden hat den Vorteil, dass ausserhalb der Präsentation keine unerwarteten Reaktionen zu erwarten sind. Allerdings hat es den Nachteil, dass der Trackingvorgang jeweils gestartet und gestoppt werden muss. Wenn allerdings dynamische Gesten denkbar wären, dann könnte die momentane Benutzeroberfläche ganz weggelassen werden und eine Geste für Start und Stopp gewählt werden.

Auswählbare Gesten

Die Gesten sollten vom Benutzer konfigurierbar sein. Bestenfalls könnte jede beliebige Folge von dynamischen Bewegungen als Bedienungseingabe ausgewertet werden und damit wären Gesten, die für den einen Benutzer nicht nachvollziehbar erscheinen, für den anderen aber natürlich sind, einfach konfigurierbar.

4.6.3 Die weiteren Schritte

Datensammlung	Erstellen einer Sammlung von Daten, mit klar erkennbaren, dynamischen Gesten. Dann die Klassifizierung dieser Daten durch menschliches Betrachten. Möglicherweise zusätzlich eine Interpolation dieser Daten, um noch mehr Trainingsdaten zu erlangen.
Deep-Learning	Erstellen eines Modells für ein Neuronales Netzwerk zur Erkennung von dynamischen Gesten. Umsetzung eines Prototypen für dieses Netzwerk. Training dieses Netzwerks mit dem erstellten Trainings-Set und anschliessend Überprüfung der Ergebnisse.
Field-Testing	Entsprechende Field-Tests mit dem neuen Netzwerk und zum Vergleich die bereits bestehende API. Sofern der Wert an erkannten Gesten beim neuen Netzwerk höher ist, als der Wert bei der bestehenden Software, ist das Ziel erreicht.

4.7 Projektaufbau und Planung

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Lukas Schiltknecht	02.12.2021

4.7.1 Einleitung zum Projektaufbau

Projekteinteilung

Das Projekt ist im Grossen und Ganzen in zwei Phasen aufgeteilt, die erste Phase hat zum Hauptziel, die Features und Möglichkeiten der MediaPipe API aufzuzeigen, womit auch beabsichtigt ist, die Grenzen der Softwarekomponente zu finden. Diese Phase ist gefolgt von einer Zwischenpräsentation und in der zweiten Phase wird ein entsprechend kleineres Software-Projekt umgesetzt, wo nach Möglichkeit so viel wie möglich von den beabsichtigten Funktionen umzusetzen ist.

Versuchsphase

Das Ziel der Versuchsphase ist es, die MediaPipe-API kennen zu lernen, mit der Absicht diese Softwarekomponente danach einsetzen zu können, um die in der Analyse beschriebenen Use Cases abzudecken.

Zwischen-präsentation

In der Zwischenpräsentation werden die Ergebnisse der bisherigen Arbeit vorgestellt und damit auch die Grenzen, sowie die bisher versuchten Lösungsansätze. Diese Präsentation ist gefolgt von einer Entscheidungsphase, wo beschlossen wird, welche Features endgültig in der Softwareentwicklungsphase umgesetzt werden sollen und welche Kompromisse dafür in Kauf genommen werden.

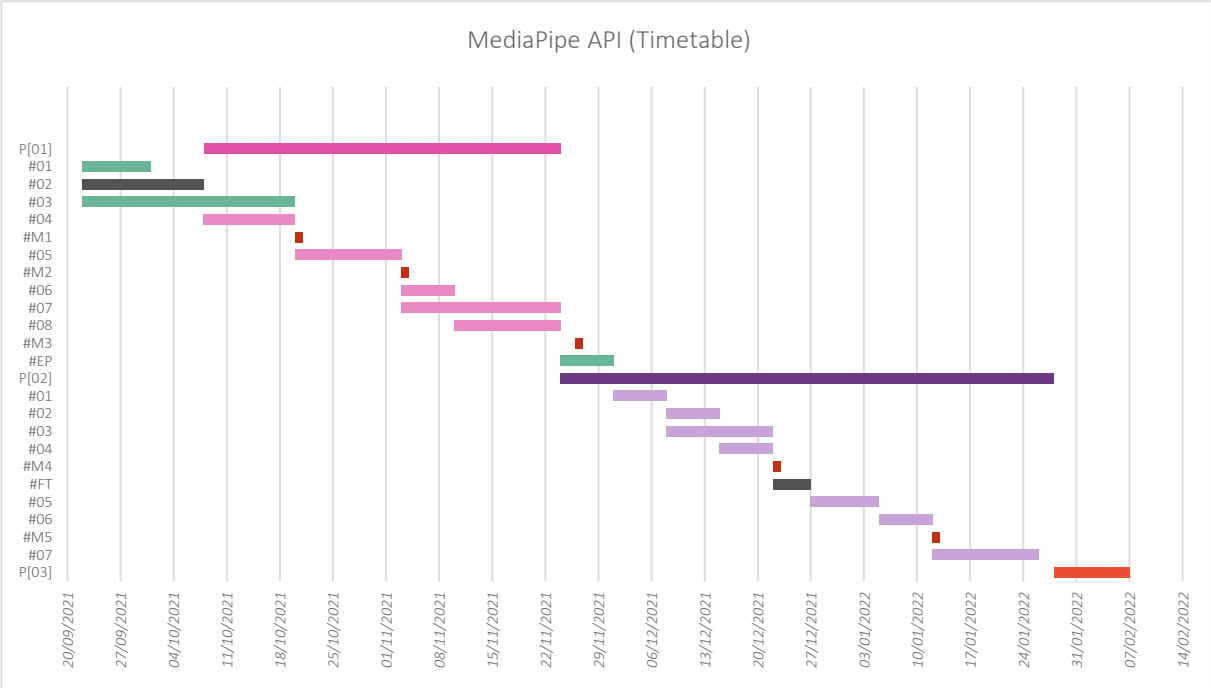
Software-entwicklung

Die Softwareentwicklungsphase entspricht am besten dem klassischen Projekt, wie es an der OST unterrichtet wird, mit einer Projektplanung, einem Architekturentwurf, einer Alphaversion direkt gefolgt von User-Tests, den daraus resultierenden Anpassungen und der Umsetzung der Betaversion.

4.7.2 Projektübersicht

Meilensteine

- *M1:* Prototyp
- *M2:* Testresultate
- *M3:* Zwischenpräsentation
- *M4:* Projektplan
- *M5:* Alphaversion
- *M6:* Betaversion
- *M7:* Abgabe



P[01]: Phase 1, Versuchsphase bezüglich der MediaPipe [Bestandaufnahme IST]
#01: Arbeitsbereiche einrichten
#02: Ausfall durch Operation
#03: Mediapipe einrichten und erste Versuche zur Handerkennung tätigen
#04: Bildschirm/Projektorerkennung entwickeln
#M1: Prototyp
#05: Anhand definierter Hintergrunddimensionen und Testszenarien Grenzen der MediaPipe austesten
M2: Testresultate
#06: Versuch mit Unschärfe das Muster-Problem zu lösen
#07: Versuch mit Farbkorrektur das Farb-Problem zu lösen
#08: Versuch mit «Selfie-Segmentation» beide Probleme zu lösen
#M3: Zwischenpräsentation
#09: Entscheidungsphase (Konsequenzen ziehen aus Phase 2)

P[02]: Phase 2, Softwareentwicklung [Planung IST]
#01: Planungsphase [Erstellen von Zeitplan, Use Cases, Softwareanforderungen]
#02: Architektur anpassen
#03: Implementation Features
#04: Testing
#M4: Alphaversion (feature-komplett)
#FT: Feiertage
#05: Usertestszenarien durchführen
#06: Testresultate einfliessen lassen
#M5: Betaversion
#07: Letzte Anpassungen

P[03]: Phase 3, Vorbereitung der Präsentation

4.7.3 Team und Arbeitsschwerpunkte

Lukas Schiltknecht

Schwerpunkte: Bildschirmerkennung, Kamerakalibration, Inputstabilisierung

Entwicklerkenntnisse: JS, Python, C++, C#, Java

Nathanael Gall

Schwerpunkte: Gestenerkennung, Testing

Entwicklerkenntnisse: Python, C#, Java

4.7.4 Aufwandschätzung, Zeitplan, Projektplan

Spital

Lukas Schiltknecht hatte am Anfang der Arbeit einen Unfall. Dies führte dazu, dass das Zeitbudget über insgesamt 18 Wochen verteilt wurde, weil er dadurch in den ersten Wochen nur eingeschränkt arbeiten konnte.

Übersicht

Die Bachelorarbeit gibt 12 Credits und hat entsprechend ein ungefähres Zeitbudget von 30 h/etc * 12 etcs * 2 Studenten = 720 Stunden. Aufgrund der experimentellen Art des Auftrags und der Abhängigkeit des Endziels von den Ergebnissen dieser Experimente wurde der Zeitplan nicht am Anfang gemacht. Ohne dies zeitlich einzuschränken, wurde zuerst eine ausführliche Systemanalyse gemacht. Als sich ein Bild herauszeichnete, was in etwa möglich ist, waren davon noch 268 Stunden übrig. Mit diesem Budget wurde in Woche 11 dann der Projektplan gemacht.

Aufwandschätzung

Der geschätzte Aufwand in Stunden umfasst sowohl Implementation als auch Dokumentation der Sprints und wird in geschätzten Aufwand und den Erwartungswert aufgeteilt, welcher aus der Risikoanalyse hervorgeht.

Sprint	Zeitaufwand	Erwartungswert Risiko
#01 Planungsphase	30 h	0 h
#02 Architektur anpassen	5 h	6 h
#03 Implementation Features	65 h	19 h
#04 Testing	10 h	6 h
#05 Userszenarien durchführen	10 h	7 h
#06 Testszenarien einfliessen lassen	30 h	11 h
#07 Letzte Anpassungen	45 h	6h
Total	**215 h**	**55 h**

Tabelle 2: Stundenbudget Entwicklungsphase

4.7.5 Risiken

Risikoanalyse

Bei diesem Projekt sind Risiken wie folgt behandelt worden: Zuerst wurde eine Liste mit vorhersehbaren Problemen erstellt. Danach wurden die Eintrittswahrscheinlichkeit und die Zeitkosten zur Behebung geschätzt. Schlussendlich wurden dann die erwarteten Zeitkosten im Zeitplan budgetiert.

	Beschreibung	h	Pr	Erwartungswert Risiko
R1	Probleme mit der Architektur und Mediapipe (#03, #06)	100	5 %	5 h
R2	Uneinigkeiten der Studenten, über Vorgehensweise (Alle Sprints)	36	100 %	36 h
R3	Implementation schwieriger oder aufwendiger als erwartet (#03, #06)	50	20 %	10 h
R4	Erschweren des Testens und Besprechens durch neue Coronamassnahmen (alle Sprints primär #05)	10	10 %	1 h
R5	Ein Student wird krank was zu einem Bottleneck führt (alle Sprints, Primär #03)	10	30 %	3 h
				55 h

Tabelle 3: Risikomatrix

Massnahmen

Für einige dieser Risiken wurden entweder Lösungsstrategien oder Präventivmassnahmen entwickelt.
R4: Die Meetings können einfach per Videokonferenz gehalten werden. Für das Testen kommt es auf die Härte der Massnahmen an.
R5: Die zeitlichen Kosten können eingeschränkt werden durch Modularisierung und sinnvolle Arbeitstrennung.

4.7.6 Scrum +

Hinweis zur Organisation

Wie schon im einleitenden Kapitel beschrieben ist in der Versuchsphase sehr flexibel und sehr offen in Bezug auf die Ergebnisse gearbeitet worden, um die Einstellung des Teams so kreativ und experimentfreudig wie möglich zu halten. Dies geschah immer in der Hinsicht, dass die Ergebnisse danach zu einer Entscheidung führen, was als Software-Ergebnis mit der verbleibenden Zeit umsetzbar ist und welche Themen für ein zukünftiges Team zur Lösung noch anstehen würden.

Scrum +

Die Software-Entwicklungsphase orientiert sich mehrheitlich an dem Projektmodell, welches an der OST im Modul SE2 unterrichtet wurde, dieses beinhaltet die von der agilen Entwicklung bekannten Sprints, in diesem Fall mit einer Dauer von 3 Wochen und die vom RUP-Modell bekannten Meilensteine.

Teamsitzungen

Während des gesamten Projekts wurden kontinuierlich Teamsitzungen abgehalten, in denen jeweils die drei Punkte *Stand der Arbeit, Besprochene Punkte* und *Folgende Schritte* besprochen wurden. Die stichwortartigen Sitzungsprotokolle befinden sich im Anhang.

4.8 Projektmonitoring – Ist

Änderungsnachweis

Version	Änderungsgrund	Autor	Datum
1.0	Erstellung	Nathanael Gall	18.01.2022

4.8.1 Zeitmanagement

Allgemein

Das Zeitmanagement lässt sich in zwei Teile aufsplitten. Einerseits geht es darum zu zeigen, wie das Projekt vorwärts gekommen ist, im Vergleich mit dem Zeitplan. Andererseits geht es darum, zu dokumentieren, wie gross der tatsächliche Arbeitsaufwand gewesen ist.

Zeitplan

Aufgrund der Aufgabenstellung wurde der Zeitplan erst am Ende der Analysephase erstellt. Dadurch kann das Einhalten dieses Zeitplans erst ab [P2] Phase 2: Softwareentwicklung eingeschätzt werden. Die geplanten Ziele wurden, abgesehen von [M4] Alpha Version, wie geplant erreicht. Für [M4] mussten über die Feiertage noch einige Bugfixes gemacht werden, die Arbeit war danach wieder genau im Zeitplan.

Aufwand

Der Zeitaufwand der Studenten ist hier noch in einem Graphen aufgelistet. Der erwartete Aufwand wären 360 Stunden pro Person. Der tatsächliche Aufwand lag bei 723 Stunden.

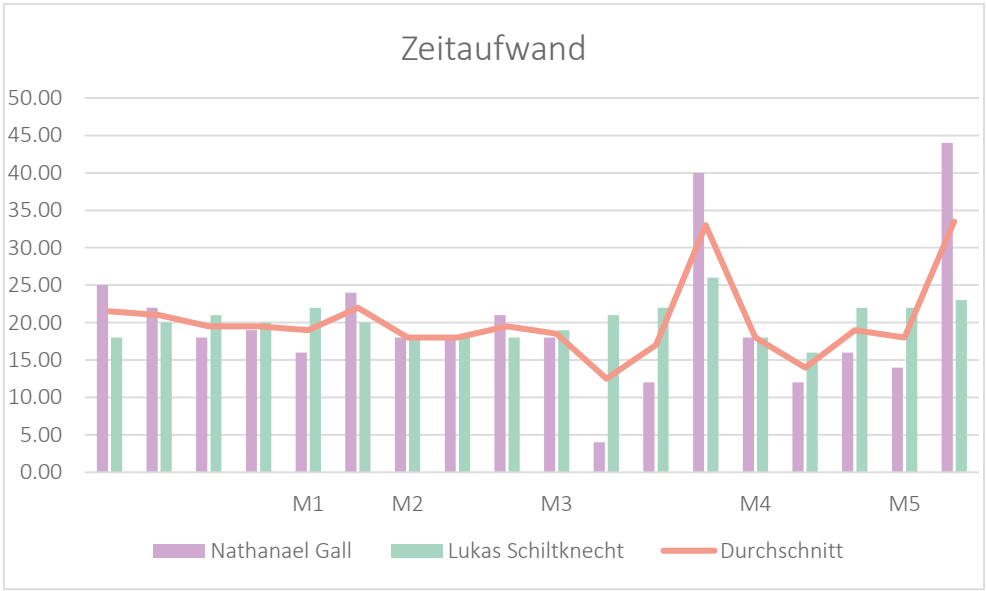


Abbildung 11: Zeitaufwand

4.8.2 Codestatistik

Code Statistik

Mit Cloc [7] wurden Folgende Statistiken über die fertige Anwendung erhoben:

language	files	blanc	comment	code
Python	24	365	344	1455

Tabelle 4: Statistiken über die Anwendung

Git Verzeichnis

Die Statistik über das Git Verzeichnis wurde aus dem Verzeichnis herausgelesen.

	Nathanael Gall	Lukas Schiltknecht
files changed	88	490
lines added	1945	8530
lines deleted	572	4749
delta	1373	3933
number of commits	18	115

Tabelle 5: Statistiken über das Git Verzeichnis

4.8.3 Beschlussprotokolle

Allgemein

Die wöchentlichen Meetings mit dem Dozenten wurden jeweils in einem Textfile protokolliert. Diese befinden sich im Abgabeordner unter Meetings. Sie wurden jeweils in drei Kapitel aufgeteilt.

Stand der Arbeit

Unter Stand der Arbeit wurde am Anfang der Meetings jeweils kurz zusammengefasst, welche Fortschritte in der letzten Woche gemacht wurden. Dort gab es jeweils auch eine Code Demonstration, welche dem Dozenten Aufschluss gab, darüber, wie gut die implementierten Änderungen funktionierten.

Besprochene Punkte

Unter besprochene Punkte stehen Ideen, Anstösse und Verbesserungsvorschläge aufgelistet, welche meistens vom Dozenten kamen und während des Meetings besprochen wurden.

Folgende Schritte

Am Schluss der Meetings wurde festgehalten, welche Arbeitsschritte in der jeweils folgenden Woche angegangen werden würden.

5. Softwaredokumentation

5.1 Installationsschritte

ZIP-Verzeichnis

Das mit der Arbeit mitgelieferte Verzeichnis enthält einen Ordner `code` für den Sourcecode der Software. Alle folgenden Schritte gehen von der Annahme aus, dass sich der Benutzer das ZIP-Verzeichnis entpackt hat und sich in dem Ordner für den Sourcecode befindet.

Python

Der Programmcode wurde in Python geschrieben, daher ist es notwendig, die entsprechende Version von Python zu installieren. Das Entwicklerteam empfiehlt die Version mit welcher die Software erstellt und getestet wurde also:
`Python 3.9.6`

Wie dies gemacht wird, ist von Betriebssystem zu Betriebssystem unterschiedlich. Hier ist ein entsprechender Link zur Homepage der Organisation.

Installation

Sobald die vorgeschlagene Version installiert ist, wird der Befehl `pip` auf der Kommandozeile verfügbar sein. Dies bedeutet folgende Schritte sind nun durchführbar:

1. Öffnen einer Kommandozeile. (Windows Explorer > Ordner-Suchbalken > «cmd» eingeben > «Enter» drücken)
2. Eingabe des Befehls: `pip install -f requirements.txt`
3. Eingabe zum Starten des Programms: `python freecursor/freecursor.py`

Wichtig: Hier gilt es zu beachten, dass die Software immer aus dem Grundverzeichnis gestartet werden sollte, denn alle relativen Pfade sind aus dem Verzeichnis `code` angegeben. Sonst erscheint eine Fehlermeldung, dass die QSS-Datei nicht gefunden werden konnte.

5.2 Benutzeranweisungen

Inhalt

Dieses Kapitel beschreibt die einzelnen Funktionen in Kurzform. Die ausgeführten manuellen Testverfahren sind im Kapitel über die Tests beschrieben.

Ausgangszustand

Wenn die Applikation gestartet wird, ist der Button für Tracking ausgegraut, weil keine entsprechende Datei bezüglich der Kalibration für die Kamera besteht. Nach dem Aufstarten sollte aber, sofern eine interne oder externe Kamera dem System zugänglich ist, mindestens eine Kamera unter «Device Camera» aufgeführt sein, denn beim Aufstarten werden iterativ Video-Streams geöffnet, auf welchen

versucht wird, ein Bild auszulesen. Wenn dies gelingt, wird der Stream und die entsprechende Auflösung automatisch gelistet.

Start Preview Um zu sehen, ob auf der ausgewählten Kamera auch tatsächlich ein Bild erscheint, kann eine Vorschau gestartet werden.

Calibrate Camera Um eine Kamera-Kalibration zu erstellen, drückt man auf den entsprechenden Knopf und befolgt die Anweisungen.

Tracking Um den Tracking-Vorgang zu starten, drückt man auf «Start-Tracking» und befolgt die Anweisungen. Sofern danach die Kamera läuft und kein Fehler angezeigt wird, ist davon auszugehen, dass die Software die Gesten verfolgt und in Aktionen umsetzt.

Extra-Gesture Die zwei verschiedenen Presets für die «Spiderman»-Geste ermöglichen es sowohl Microsoft-PowerPoint als auch die Software MyPaint etwas einfacher zu bedienen. Die vorbelegten Tastatureingaben werden beim Aufstarten des Tracking-Vorgangs angezeigt.

Save Settings Die Einstellungen der Bedienungsoberfläche können über Save-Settings gespeichert werden. Dies beinhaltet aber nicht die Kamera-Kalibration, denn diese wird automatisch abgespeichert, sofern sie erfolgreich war.

5.3 Code-Wartbarkeit

Kommentare Da eine Referenz in der Regel schon veraltet ist, wenn sie erstellt wird, hat sich das Entwicklerteam entschieden, die Wartbarkeit und Nachvollziehbarkeit des Codes über die entsprechenden Kommentarbereiche im Code zu lösen, welche auch in Visual-Studio-Code jeweils angezeigt werden, sobald man über eine Funktion oder Klasse fährt. Diese Vorgehensweise macht es einfacher Code anzupassen, da die Referenz nicht zusätzlich angepasst werden muss.

[NOTE] Sofern Kommentare mit `[NOTE]` gekennzeichnet sind, dienen sie dazu, einem Entwickler Informationen zur Weiterentwicklung zu geben. In den meisten Fällen handelt es sich um Bereiche, welche für Debugging besonders wichtig sind, damit man möglichst schnell an die relevanten Ausgaben kommt.

Lesbarkeit Entgegen der Aussage, dass *lesbarer Code keine Kommentare braucht,* wird diese Software bewusst mit Hilfe von automatischem Highlighting der IDE geschrieben und dies dient auch zu einem grossen Teil der Orientierung.

6. Literatur und Quellenverzeichnis

Herkunft der Vorlage

Das Dokument wurde auf der Basis einer Vorlage für technische Berichte erstellt. Die Vorlage ist ein Element des «Werkzeugkastens Technische Berichte» der Hochschule für Technik Rapperswil. Sie orientiert sich an Prinzipien des strukturierten Schreibens.

7. Bibliografie

[1] M. #914806, «learn.sparkfun.com/tutorials,» 6 February 2019. [Online]. Available: https://learn.sparkfun.com/tutorials/computer-vision-and-projection-mapping-in-python/all.

[2] Google, «MediaPipe Documentation,» 2020. [Online]. Available: https://google.github.io/mediapipe/.

[3] V. Bazarevsky, «arxiv.org,» Cornell University, 18 06 2020. [Online]. Available: https://arxiv.org/abs/2006.10214. [Zugriff am 10 11 2021].

[4] Google, «google.github.io/mediapipe/solutions/selfie_segmentation.html,» 2020. [Online]. Available: https://google.github.io/mediapipe/solutions/selfie_segmentation.html.

[5] eastWillow, «opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/,» 1 1 2016. [Online]. Available: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html.

[6] J. W., «calib.io,» 15 11 2018. [Online]. Available: https://calib.io/blogs/knowledge-base/calibration-patterns-explained.

[7] A. Danial, «github.com,» [Online]. Available: https://github.com/AlDanial/cloc. [Zugriff am 20 01 2022].

8. Verzeichnisse

8.1 Glossar und Abkürzungsverzeichnis

GUI	Graphic User Interface, eine visuelle Bedienoberfläche für die Software.
PyQt5	Python Bibliothek zur Erstellung eines GUI.
Moving-Average	Eine Durchschnittsberechnung über eine Menge von Werten, die sich ständig erneuert und mit jedem neuen Wert wird jeweils der älteste Wert gelöscht.
Repository	Die Code-Basis, für ein bestimmtes zusammenhängendes Projekt.
Gitlab	Versionsmanagementplattform, welche mehrere Zusatzfunktionen anbietet.
Gitlab-Pipeline	Ein automatisierter Workflow von Gitlab, um die auf einem Repository vorhandene Code-Basis bei jedem Commit auf den Main-Branch zu testen.
Life-Stream	Video-Stream in diesem Fall eingefangen über eine Webcam, welche im Gerät vorhanden ist.
Stub-Stream	Testobjekt, um den Bild-Input eines realen Life-Streams vorzutäuschen.
OpenCV	Code-Library, die unter anderem zum Öffnen und Auslesen von Kamera-Input verwendet wird.
Charuco-Indize	Bildzeichen, die von der OpenCV-Bibliothek erkannt werden und zur Kamera-Kalibration verwendet werden können.
MediaPipe API	Von Google zur Verfügung gestellte KI-Bibliothek, um diverses Bildmaterial auszulesen und Objekte wie Körperteile zu erkennen.
Object Tracing	Funktionalität, welche in einem Bild Objekte aus der realen Welt erkennt.
Motion Tracking	Funktionalität, welche in einem Film Bewegungen bekannter Objekte verfolgt.
Exponential-Smoothing	Ein statistisches Verfahren, um die Werte einer Resultatmenge so zu gewichten, dass die neuen Werte stärker gewichtet werden als die älteren Werte.
IDE	Kurzform für Intelligent Development Environment, in diesem Fall ist damit die Software Visual-Studio-Code von Microsoft gemeint.
Highlighting	Code Highlighting, ist eine Funktion, welche von einer IDE angeboten wird, welche den Code unterschiedlich einfärbt, damit man sich optisch orientieren kann.

8.2 Abbildungen

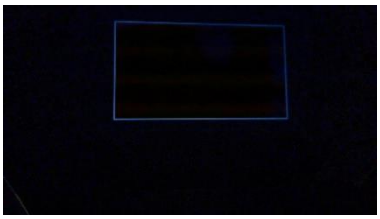
8.3 Tabellen

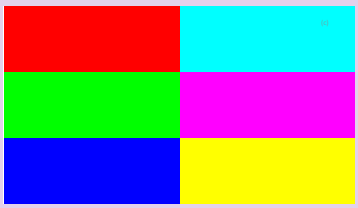
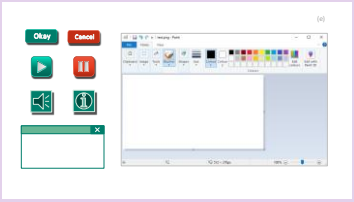
9. Anhang

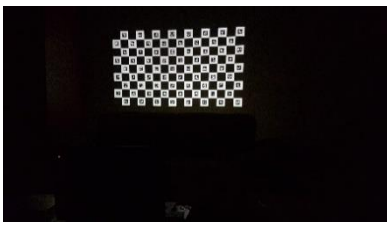
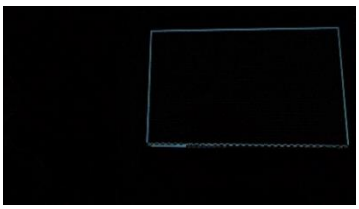
9.1 Testdokumentation

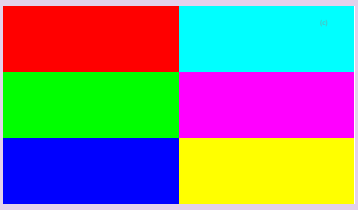
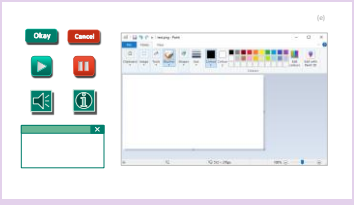
9.1.1 Ursprüngliche Tests

9.1.1.1 Wohnzimmer Nathanael

Situation		
Beamer:	Kleiner Portabler Beamer	
Kamera:	Laptopkamera Thinkpad T450	
Leinwand:	Wohnzimmerwand	
Lichtsituation 1:		Licht kommt von der Raumbeleuchtung. Kein Sonnenlicht.
Charuco:		
Error:	0.185043	
Gefundene Charucos:	41.06 %	
Bildschirm erkennen:		Nicht möglich (könnte auch Softwarefehler sein).
Mediapipe:		
Folie	Bild	Erfolg

	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.
	Nicht vorhanden, da Leinwand nicht gefunden.	Gelb funktioniert gut, beim Rest der Farben wird die Hand nicht erkannt.
	Nicht vorhanden, da Leinwand nicht gefunden.	Bei Gelb funktioniert es einwandfrei, vor Grün und Rot wird die offene Hand mit Rücken zur Wand teilweise erkannt.
	Nicht vorhanden, da Leinwand nicht gefunden.	Teilweise erkannt vor den grössten Buttons, sonst funktioniert es überall.
	Nicht vorhanden, da Leinwand nicht gefunden.	Text, Bild Oben, Bild Unten funktioniert einwandfrei. Bild Mitte: Nur offene Hand.

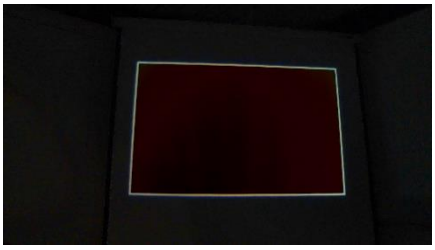
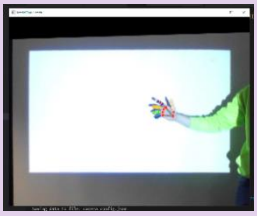
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert meistens.
	Nicht vorhanden, da Leinwand nicht gefunden.	Hände auf Folie erkannt. Funktioniert auf Weiss einwandfrei. Auf Schwarz funktioniert es nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert meistens.
Lichtsituation 0:		Kein Licht.
Charuco:		
Error:	0.185043	
Gefundene Charucos:	41.06 %	
Bildschirm erkennen:		Nicht möglich (könnte auch Softwarefehler sein).
Mediapipe:		

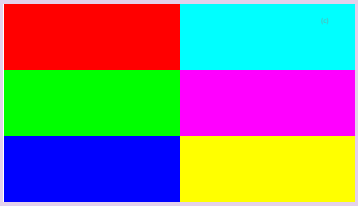
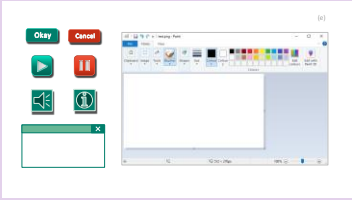
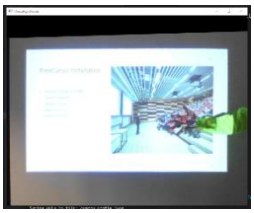
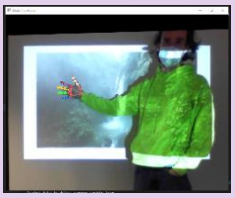
Folie	Bild	Erfolg
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Gelb funktioniert offene Hand teilweise, Rest funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Gelb funktioniert offene Hand teilweise, Rest funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.

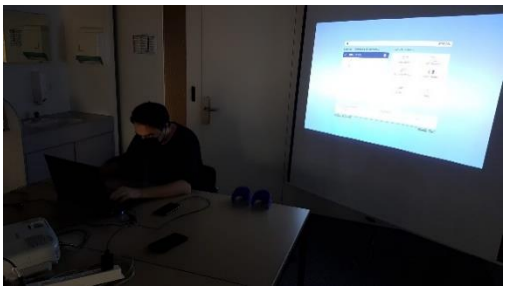
	Nicht vorhanden, da Leinwand nicht gefunden.	Text funktioniert einwandfrei. Bild Mitte, Bild oben, Bild unten: Offene Hand teilweise
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.
	Nicht vorhanden da Leinwand nicht gefunden.	Funktioniert selten.

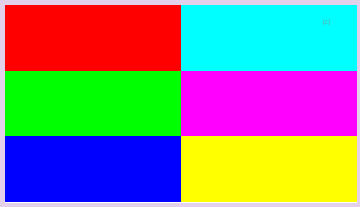
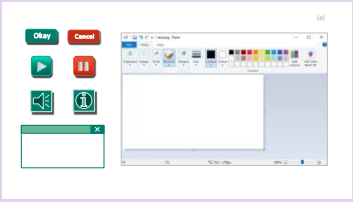
9.1.2 Sitzungszimmer 1224

Situation	
Beamer:	Epson Beamer
Kamera:	Laptopkamera Dell (Lukas)
Leinwand:	Wohnzimmerwand

Lichtsituation 3:		Tageslicht und Zimmerbeleuchtung
Charuco:		
Error:	0.3177	
Gefundene Charucos:	51.87%	
Bildschirm erkennen:		Einwandfrei
Mediapipe:		
Folie	**Bild**	**Erfolg**
		Funktioniert meistens.
		Funktioniert einwandfrei.

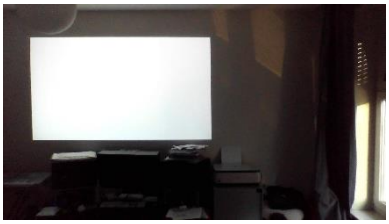
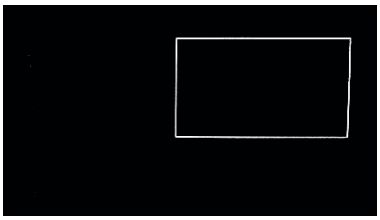
		Gelb teilweise, der Rest nicht.
		Bei Gelb funktioniert es meistens. Rot und Grün selten, der Rest nicht.
		Funktioniert auf der weissen Zeichenfläche teilweise, sonst nirgends.
		Funktioniert nirgends.
		Funktioniert meistens.
		Hände auf Folie nicht erkannt. Funktioniert auf Schwarz meistens. Auf Weiss funktioniert es nicht.

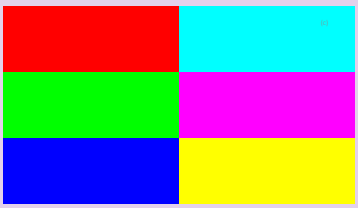
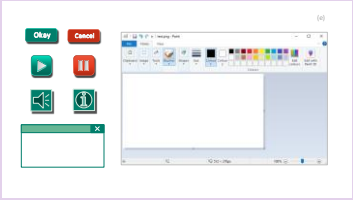
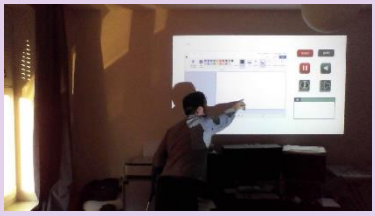
		Funktioniert teilweise.
Lichtsituation 1:		Nur natürliches Licht
Charuco:		
Error:	Fehlgeschlagen, Raum muss heller sein	
Gefundene Charucos:	-	
Bildschirm erkennen:		Nicht möglich (könnte auch Softwarefehler sein)
Mediapipe:		
Folie	**Bild**	**Erfolg**
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.

	Nicht vorhanden, da Leinwand nicht gefunden.	Gelb funktioniert offene Hand teilweise, Rest funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Gelb funktioniert offene Hand teilweise, Rest funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert einwandfrei.
	Nicht vorhanden, da Leinwand nicht gefunden.	Text funktioniert einwandfrei. Bild Mitte, Bild oben, Bild unten: Offene Hand teilweise
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.
	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert nicht.

	Nicht vorhanden, da Leinwand nicht gefunden.	Funktioniert selten.

9.1.3 Test der Selfie-Segmentation-Bild-Auslöschung

Situation		
Beamer:	Beamer für Präsentationen (ca. 3000 lm)	
Kamera:	Laptopkamera Dell (1280x1080)	
Leinwand:	Wohnzimmerwand	
Lichtsituation 1:		Lichteinstrahlung von Links ist Sonnenlicht
Charuco:		
Error:	0.2753744488275171	
Gefundene Charucos:	40.88 %	
Bildschirm erkennen:		Fehlerfrei
Mediapipe:		
Folie	Bild	Erfolg

		Erkannter Bildbereich ist zu gross und wächst dazu, oder beginnt zu flimmern.
		Erkannter Bildbereich ist etwa angemessen, aber flimmert.
		Erkannter Bildbereich ist zu gross und wächst dazu, oder beginnt zu flimmern.
		Erkannter Bildbereich ist zu gross und wächst dazu, oder beginnt zu flimmern.
		Erkannter Bildbereich ist zu klein und flimmert.

		Über Bild ist der erkannte Teil zu gross und schliesst Teile des Bildes mit ein. Bild wird auch erkannt, wenn niemand davor steht.
		Funktioniert erstaunlich viel besser als sonst, aber flimmert trotzdem.
		Vollkommen zufällig.
		Grösstenteils zufällig und eher selten hat man Glück, wenn der Bereich erkannt wird, wo eine Person über dem Bildschirm ist.

9.2 Usability-Tests Freecursor

9.2.1 Hypothese:

- «Es ist möglich ohne Zusatzhilfe eine Kamerakalibration erstellen.»
- «Das Gestentracking ist anhand der Anweisungen nachvollziehbar.»
- «Eine Powerpoint kann mittels der Software bedient werden.»
- «Mittels der Software kann ein Herz gezeichnet werden.»

9.2.2 Testgruppe:

- Primary Users (Studenten)
- Secondary Users (berufstätige Hochschulabgänger)

9.2.3 Testmethode:

- Direkte Usertests mit persönlichem Kontakt, *nach Möglichkeit* ohne Hilfe der Begleiter. (sonst vermerkt)

9.2.4 Durchführung [Testperson #01]

9.2.4.1 Vorinterview:

1. IT Kenntnisse: **++, +, -, - -**
2. Präsentationserfahrung: **++, +, -, - -**
3. Wann hast du das letzte Mal eine Präsentation gehalten?
4. Wie oft im Jahr hast du eine Besprechung, in der ein Whiteboard genutzt wird?
5. In welcher Branche arbeitest du?
6. Angaben über Alter, Geschlecht, Händigkeit und Hautfarbe [festhalten]

(Secondary Users)

1.	2.	3.	4.	5.	6.
-	+	Vor *3* Jahren	*4* mal pro Jahr	*Branche Gesundheit*	68/w/rh/w

9.2.4.2 Test-Cases

#	Test-Case	Fragestellung	Erwartung	Resultat
1	*Kamera Kalibrieren*	Du möchtest, dass die Kamera kalibriert ist, damit das Tracking der Gestik funktioniert. Was tust du als nächstes?	Auf den Kalibrieren Button drücken, die Anweisungen befolgen und eine Kalibration durchführen.	Versucht während Instructions die Helligkeit anzupassen. Sucht nach Feedback für Bildschirmposition. Macht Kalibration durch und speichert die «Settings» in der Annahme, dass die Kalibrationsdaten gemeint sind.
2	*Herz zeichnen*	Du möchtest die Software dazu benutzen ein Herz mit MyPaint zu zeichnen. Wie würdest du das anstellen?	Auf den Tracking Knopf drücken, MyPaint öffnen, ein Herz zeichnen.	Versteht nicht ganz, warum sie «Start Tracking drücken soll. Gestiken sind nicht ganz klar dargestellt.
3	*Leinwand verschieben (MyPaint)*	Du möchtest das Herz etwas verschieben. Was machst du nun?	Rechtsklick-Gestik formen, Hand bewegen.	Funktioniert meist, wie beabsichtigt. Nach einem Klick wird Rechtsklick auch immer noch beim Linksklick ausgelöst.
4	*PowerPoint*	Du möchtest eine Präsentation zeigen, ohne den Cursor zu benutzen. Was musst du dazu machen?	PowerPoint öffnen via Linksklick, mit Linksklick vorwärts mit Spiderman rückwärts.	Startet Tracking wie gewohnt, sah nicht, wo man die Extra-Gestik einstellt.
5	*Einstellungen speichern*	Du möchtest die Einstellungen abspeichern. Was wirst du dazu tun müssen?	Button, um die Einstellungen zu speichern, drücken.	Einwandfrei nachvollziehbar ausgeführt.

9.2.4.3 Post-Interview:

1. Konntest du nachvollziehen wie eine Kalibration gemacht werden kann?
   Nein, kein Feedback während den Instruktionen.
2. Wie fühlt sich die Bedienung der Software im Allgemeinen an?
   Unklar, ohne Beispiel wie Gesten funktionieren hätte man sie nicht hingekriegt.
3. Was würdest du dir wünschen, damit du die Software auch im Alltag einsetzen würdest?
   Viel Zeit, um auszuprobieren, was sie alles kann.
4. Sind die Einstellungen für dich nachvollziehbar?
   Nein, ich hätte gedacht, dass Save Settings die Kalibration speichert oder den Bildschirm oder so was.
5. Würdest du eine weiterentwickelte Version der Software zu Präsentationszwecken verwenden?
   Ja, unter Umständen schon, sofern ich die Gesten trainiert habe und nachvollziehen kann, wie sie funtionieren.

9.2.5 Durchführung [Testperson #02]

9.2.5.1 Vorinterview:

1. IT Kenntnisse: **++, +, -, - -**
2. Präsentationserfahrung: **++, +, -, - -**
3. Wann hast du das letzte Mal eine Präsentation gehalten?
4. Wie oft im Jahr hast du eine Besprechung, in der ein Whiteboard genutzt wird?
5. In welcher Branche arbeitest du?
6. Angaben über Alter, Geschlecht, Händigkeit und Hautfarbe [festhalten]

(Secondary User)

1.	2.	3.	4.	5.	6.
++	++	Vor *3* Monaten	*6* mal pro Jahr	*Branche Beratung*	31/w/lh/w

9.2.5.2 Test-Cases

#	Test-Case	Fragestellung	Erwartung	Resultat
1	*Kamera Kalibrieren*	Du möchtest, dass die Kamera kalibriert ist, damit das Tracking der Gestik funktioniert. Was tust du als nächstes?	Auf den Kalibrieren Button drücken, die Anweisungen befolgen und eine Kalibration durchführen.	Fragt muss ich nun 'S' und Pfeil drücken? (grinst, weil sie weiss, dass es bloss eine unklare Darstellung ist.) Weist darauf hin, dass Maximierung auch automatisch geht. Führt Kalibration aus.
2	*Herz zeichnen*	Du möchtest die Software dazu benutzen ein Herz mit MyPaint zu zeichnen. Wie würdest du das anstellen?	Auf den Tracking Knopf drücken, MyPaint öffnen, ein Herz zeichnen.	Positionierung der Person ist unklar. Zeichnet ein Herz. Erkennung funktioniert besser rechtshändig.
3	*Leinwand verschieben (MyPaint)*	Du möchtest das Herz etwas verschieben. Was machst du nun?	Rechtsklick-Gestik formen, Hand bewegen.	Rechtsklick funktioniert einwandfrei beim ersten Mal. Danach wird auch bei Linksklick ein Rechtsklick ausgeführt.
4	*PowerPoint*	Du möchtest eine Präsentation zeigen, ohne den Cursor zu benutzen. Was musst du dazu machen?	PowerPoint öffnen via Linksklick, mit Linksklick vorwärts mit Spiderman rückwärts.	Fragt, ob die Kamera nochmals kalibriert werden muss. Test Case funktioniert wie beabsichtigt, ausser, dass die Spiderman-Geste nicht das macht, was sie sollte.
5	*Einstellungen speichern*	Du möchtest die Einstellungen abspeichern. Was wirst du dazu tun müssen?	Button, um die Einstellungen zu speichern, drücken.	Test Case verläuft wie beabsichtigt.

9.2.5.3 Post-Interview:

1. Konntest du nachvollziehen wie eine Kalibration gemacht werden kann?
   Ja, hat funktioniert.
2. Wie fühlt sich die Bedienung der Software im Allgemeinen an?
   Fühlt sich sehr straight forward an. Etwas sehr simpel aber im Moment gibt es noch nicht so viele Optionen. Gestures in den Einstellungen könnten noch etwas nachvollziehbarer dargestellt werden.
3. Was würdest du dir wünschen, damit du die Software auch im Alltag einsetzen würdest?
   Es wäre praktisch, wenn man keinen Laptop benutzen müsste, beispielsweise eine Smartphone-Kamera oder so. Schnellere Funktion und Reaktion, genauere Reaktion. Klarere Erklärung welche Gestures wann enden und wann beginnen; Hilfescreen/Page.
4. Sind die Einstellungen für dich nachvollziehbar?
   MyPaint oder PowerPoint sind etwas unklar. Startbuttons sind klar. Vielleicht Hinweis, dass Kalibration nur einmal gemacht werden muss.
5. Würdest du eine weiterentwickelte Version der Software zu Präsentationszwecken verwenden?
   Generell ja, je nachdem welche Eigenschaften der Raum hat. Aber es muss noch etwas mehr bieten als ein Präsentationspointer.

9.2.6 Durchführung [Testperson #03]

9.2.6.1 Vorinterview:

7. IT Kenntnisse: **++, +, -, - -**
8. Präsentationserfahrung: **++, +, -, - -**
9. Wann hast du das letzte Mal eine Präsentation gehalten?
10. Wie oft im Jahr hast du eine Besprechung, in der ein Whiteboard genutzt wird?
11. In welcher Branche arbeitest du?
12. Angaben über Alter, Geschlecht, Händigkeit und Hautfarbe [festhalten]

(Primary User)

1.	2.	3.	4.	5.	6.
+	+	1 Monat	*Sehr selten*	*Ingenieur*	27/w/rh/w

9.2.6.2 Test-Cases

#	Test-Case	Fragestellung	Erwartung	Resultat
1	*Kamera Kalibrieren*	Du möchtest, dass die Kamera kalibriert ist, damit das Tracking der Gestik funktioniert. Was tust du als nächstes?	Auf den Kalibrieren Button drücken, die Anweisungen befolgen und eine Kalibration durchführen.	Versteht nicht, dass es noch nicht direkt losgeht.
2	*Herz zeichnen*	Du möchtest die Software dazu benutzen ein Herz mit MyPaint zu zeichnen. Wie würdest du das anstellen?	Auf den Tracking Knopf drücken, MyPaint öffnen, ein Herz zeichnen.	Unklar, wie lange die Person ausserhalb des Bilds bleiben muss. Unklar, welche Gesten was machen.
3	*Leinwand verschieben (MyPaint)*	Du möchtest das Herz etwas verschieben. Was machst du nun?	Rechtsklick-Gestik formen, Hand bewegen.	Funktioniert.
4	*PowerPoint*	Du möchtest eine Präsentation zeigen, ohne den Cursor zu benutzen. Was musst du dazu machen?	PowerPoint öffnen via Linksklick, mit Linksklick vorwärts mit Spiderman rückwärts.	Unklar, welche Gesten was machen.
5	*Einstellungen speichern*	Du möchtest die Einstellungen abspeichern. Was wirst du dazu tun müssen?	Button, um die Einstellungen zu speichern, drücken.	Test Case verläuft wie beabsichtigt.

9.2.6.3 Post-Interview:

6. Konntest du nachvollziehen, wie eine Kalibration gemacht werden kann?
   Ja, ausser dass es nicht direkt losgeht.
7. Wie fühlt sich die Bedienung der Software im Allgemeinen an?
   Finger verkrampfen. Linksklick unnatürlich.
8. Was würdest du dir wünschen, damit du die Software auch im Alltag einsetzen würdest?
   Schnellere Reaktion (bei Klick sofort Klick auslösen).
9. Sind die Einstellungen für dich nachvollziehbar?
   Ja.
10. Würdest du eine weiterentwickelte Version der Software zu Präsentationszwecken verwenden?
    Glaub schon, wenn es so weit ausgereift ist, dass es konsistent funktioniert.

9.2.7 Durchführung [Testperson #04]

9.2.7.1 Vorinterview:

13. IT Kenntnisse: **++, +, -, - -**
14. Präsentationserfahrung: **++, +, -, - -**
15. Wann hast du das letzte Mal eine Präsentation gehalten?
16. Wie oft im Jahr hast du eine Besprechung, in der ein Whiteboard genutzt wird?
17. In welcher Branche arbeitest du?
18. Angaben über Alter, Geschlecht, Händigkeit und Hautfarbe [festhalten]

(Primary User)

1.	2.	3.	4.	5.	6.
+	+	1 Jahr	*0*	*Gaming*	27/m/rh/w

9.2.7.2 Test-Cases

#	Test-Case	Fragestellung	Erwartung	Resultat
1	*Kamera Kalibrieren*	Du möchtest, dass die Kamera kalibriert ist, damit das Tracking der Gestik funktioniert. Was tust du als nächstes?	Auf den Kalibrieren Button drücken, die Anweisungen befolgen und eine Kalibration durchführen.	Weiss nicht, wann das das Befolgen der Instructions losgeht.
2	*Herz zeichnen*	Du möchtest die Software dazu benutzen ein Herz mit MyPaint zu zeichnen. Wie würdest du das anstellen?	Auf den Tracking Knopf drücken, MyPaint öffnen, ein Herz zeichnen.	Unklar, wo die Hand sein muss, damit sie erkannt wird. Unklar, was Gesten machen.
3	*Leinwand verschieben (MyPaint)*	Du möchtest das Herz etwas verschieben. Was machst du nun?	Rechtsklick-Gestik formen, Hand bewegen.	Zieht beim Rechtsklick die Finger an. Sonst genau nach Plan.
4	*PowerPoint*	Du möchtest eine Präsentation zeigen, ohne den Cursor zu benutzen. Was musst du dazu machen?	PowerPoint öffnen via Linksklick, mit Linksklick vorwärts mit Spiderman rückwärts.	Nach wie vor nicht ganz klar, welche Gesten was bedeuten.
5	*Einstellungen speichern*	Du möchtest die Einstellungen abspeichern. Was wirst du dazu tun müssen?	Button, um die Einstellungen zu speichern, drücken.	Test Case verläuft wie beabsichtigt.

9.2.7.3 Post-Interview:

11. Konntest du nachvollziehen, wie eine Kalibration gemacht werden kann?
    Ja.
12. Wie fühlt sich die Bedienung der Software im Allgemeinen an?
    Ungewohnt.
13. Was würdest du dir wünschen, damit du die Software auch im Alltag einsetzen würdest?
    Handgesten selber bestimmen.
14. Sind die Einstellungen für dich nachvollziehbar?
    Ja.
15. Würdest du eine weiterentwickelte Version der Software zu Präsentationszwecken verwenden?
    Eventuell, ja.